

Hugh Cartwright *Editor*

# Artificial Neural Networks

*Third Edition*



Humana Press

# METHODS IN MOLECULAR BIOLOGY

*Series Editor*

John M. Walker

School of Life and Medical Sciences

University of Hertfordshire

Hatfield, Hertfordshire, UK

For further volumes:  
<http://www.springer.com/series/7651>

For over 35 years, biological scientists have come to rely on the research protocols and methodologies in the critically acclaimed *Methods in Molecular Biology* series. The series was the first to introduce the step-by-step protocols approach that has become the standard in all biomedical protocol publishing. Each protocol is provided in readily-reproducible step-by-step fashion, opening with an introductory overview, a list of the materials and reagents needed to complete the experiment, and followed by a detailed procedure that is supported with a helpful notes section offering tips and tricks of the trade as well as troubleshooting advice. These hallmark features were introduced by series editor Dr. John Walker and constitute the key ingredient in each and every volume of the *Methods in Molecular Biology* series. Tested and trusted, comprehensive and reliable, all protocols from the series are indexed in PubMed.

# **Artificial Neural Networks**

**Third Edition**

Edited by

**Hugh Cartwright**

*Chemistry, Oxford University, Oxford, UK*



*Editor*

Hugh Cartwright  
Chemistry  
Oxford University  
Oxford, UK

ISSN 1064-3745

ISSN 1940-6029 (electronic)

Methods in Molecular Biology

ISBN 978-1-0716-0825-8

ISBN 978-1-0716-0826-5 (eBook)

<https://doi.org/10.1007/978-1-0716-0826-5>

© Springer Science+Business Media, LLC, part of Springer Nature 2021

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Humana imprint is published by the registered company Springer Science+Business Media, LLC, part of Springer Nature.

The registered company address is: 1 New York Plaza, New York, NY 10004, U.S.A.

---

## Preface

Two decades ago it would have been hard to foresee the remarkable growth in the use of artificial intelligence (AI) in the physical and life sciences. But there is a simple explanation for that rise: AI tools work.

Software is now readily available for Artificial Neural Networks, Genetic Algorithms, Deep Learning, Random Forests, Support Vector Machines, and other methods. While the software is not always trivial to use, it is becoming both more user-friendly and more powerful; this is encouraging scientists, whatever their specialization, to dive in.

This book showcases some of the studies now being pursued in the life sciences: topics range from the identification of genotype-phenotype correlations to the use of machine learning to evaluate biomedical time series; from *de novo* drug design to using recursive neural networks in the scoring of protein models; from studies of gene regulation in bacteria to the application of machine learning to the assessment of tumor tissue, and many more.

“Traditional” methods of analysis are in no imminent danger of being pushed out of the door by AI. On the contrary, these well-tested methods are increasingly being combined with the newer computational tools to enhance understanding of the large and complex datasets that the life sciences can generate.

As in earlier editions, readers who are intrigued by the applications discussed in these chapters will find practical details to help them apply the methods of AI in their own work.

*Oxford, UK*

*Hugh Cartwright*

---

# Contents

<i>Preface</i> .....	v
<i>Contributors</i> .....	ix
<b>1 Identifying Genotype–Phenotype Correlations via Integrative Mutation Analysis.....</b>	<b>1</b>
<i>Edward Airey, Stephanie Portelli, Joicymara S. Xavier, Yoo Chan Myung, Michael Silk, Malancha Karmakar, João P. L. Velloso, Carlos H. M. Rodrigues, Hardik H. Parate, Anjali Garg, Raghad Al-Jarf, Lucy Barr, Juliana A. Geraldo, Pâmela M. Rezende, Douglas E. V. Pires, and David B. Ascher</i>	
<b>2 Machine Learning for Biomedical Time Series Classification: From Shapelets to Deep Learning.....</b>	<b>33</b>
<i>Christian Bock, Michael Moor, Catherine R. Jutzeler, and Karsten Borgwardt</i>	
<b>3 Siamese Neural Networks: An Overview .....</b>	<b>73</b>
<i>Davide Chicco</i>	
<b>4 Computational Methods for Elucidating Gene Expression Regulation in Bacteria .....</b>	<b>95</b>
<i>Kratika Naskulwar, Ruben Chevez-Guardado, and Lourdes Peña-Castillo</i>	
<b>5 Neuroevolutionary Algorithms Applied for Modeling Some Biochemical Separation Processes .....</b>	<b>115</b>
<i>Silvia Curteanu, Elena-Niculina Dragoi, Alexandra Cristina Blaga, Anca Irina Galaction, and Dan Cascaval</i>	
<b>6 Computational Approaches for De Novo Drug Design: Past, Present, and Future .....</b>	<b>139</b>
<i>Xuhan Liu, Adriaan P. IJzerman, and Gerard J. P. van Westen</i>	
<b>7 Data Integration Using Advances in Machine Learning in Drug Discovery and Molecular Biology .....</b>	<b>167</b>
<i>Irene Lena Hudson</i>	
<b>8 Building and Interpreting Artificial Neural Network Models for Biological Systems .....</b>	<b>185</b>
<i>T. Murlidharan Nair</i>	
<b>9 A Novel Computational Approach for Biomarker Detection for Gene Expression-Based Computer-Aided Diagnostic Systems for Breast Cancer .....</b>	<b>195</b>
<i>Ali Al-Yousef and Sandhya Samarasinghe</i>	
<b>10 Applying Machine Learning for Integration of Multi-Modal Genomics Data and Imaging Data to Quantify Heterogeneity in Tumour Tissues .....</b>	<b>209</b>
<i>Xiao Tan, Andrew T. Su, Hamideh Hajiabadi, Minh Tran, and Quan Nguyen</i>	

11	Leverage Large-Scale Biological Networks to Decipher the Genetic Basis of Human Diseases Using Machine Learning ..... <i>Hao Wang, Jiaxin Yang, and Jianrong Wang</i>	229
12	Predicting Host Phenotype Based on Gut Microbiome Using a Convolutional Neural Network Approach ..... <i>Derek Reiman, Ali M. Farhat, and Yang Dai</i>	249
13	Predicting Hot Spots Using a Deep Neural Network Approach ..... <i>António J. Preto, Pedro Matos-Filipe, José G. de Almeida, Joana Mourão, and Irina S. Moreira</i>	267
14	Using Neural Networks for Relation Extraction from Biomedical Literature ..... <i>Diana Sousa, Andre Lamurias, and Francisco M. Couto</i>	289
15	A Hybrid Levenberg–Marquardt Algorithm on a Recursive Neural Network for Scoring Protein Models ..... <i>Eshel Faraggi, Robert L. Jernigan, and Andrzej Kloczkowski</i>	307
16	Secure and Scalable Collection of Biomedical Data for Machine Learning Applications ..... <i>Charles Fracchia</i>	317
17	AI-Based Methods and Technologies to Develop Wearable Devices for Prosthetics and Predictions of Degenerative Diseases ..... <i>Mario Malcangi</i>	337
	<i>Index</i> .....	355

---

## Contributors

- EDWARD AIREY • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*
- RAGHAD AL-JARF • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*
- ALI AL-YOUSEF • *Department of Computer Science, Faculty of Computer and Information Technology, Jerash University, Jerash, Jordan; Complex Systems, Big Data and Informatics Initiative (CSBII), Lincoln University, Christchurch, New Zealand*
- DAVID B. ASCHER • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia; Department of Biochemistry, Cambridge University, Cambridge, UK*
- LUCY BARR • *Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*
- ALEXANDRA CRISTINA BLAGA • *Faculty of Chemical Engineering and Environmental Protection “Cristofor Simionescu”, “Gheorghe Asachi” Technical University of Iasi, Iasi, Romania*
- CHRISTIAN BOCK • *Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland; SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland*
- KARSTEN BORGWARDT • *Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland; SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland*
- DAN CASCaval • *Faculty of Chemical Engineering and Environmental Protection “Cristofor Simionescu”, “Gheorghe Asachi” Technical University of Iasi, Iasi, Romania*
- RUBEN CHEVEZ-GUARDADO • *Department of Computer Science, Memorial University of Newfoundland, St. John’s, NL, Canada*
- DAVIDE CHICCO • *Krembil Research Institute, Toronto, ON, Canada*
- FRANCISCO M. COUTO • *LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisbon, Portugal*
- SILVIA CURTEANU • *Faculty of Chemical Engineering and Environmental Protection “Cristofor Simionescu”, “Gheorghe Asachi” Technical University of Iasi, Iasi, Romania*
- YANG DAI • *Department of Bioengineering, University of Illinois at Chicago, Chicago, IL, USA*
- JOSÉ G. DE ALMEIDA • *Center for Innovative Biomedicine and Biotechnology, University of Coimbra, Coimbra, Portugal; Center for Neuroscience and Cell Biology, University of Coimbra, Coimbra, Portugal*

- ELENA-NICULINA DRAGOI • *Faculty of Chemical Engineering and Environmental Protection “Cristofor Simionescu”, “Gheorghe Asachi” Technical University of Iasi, Iasi, Romania*
- ESHEL FARAGGI • *Research and Information Systems, LLC, Indianapolis, IN, USA; Department of Physics, Indiana University Purdue University Indianapolis, Indianapolis, IN, USA*
- ALI M. FARHAT • *College of Medicine, University of Illinois at Chicago, Chicago, IL, USA*
- CHARLES FRACCHIA • *BioBright, Boston, MA, USA*
- ANCA IRINA GALACTION • *Faculty of Medical Bioengineering, “Grigore T. Popa” University of Medicine and Pharmacy, Iasi, Romania*
- ANJALI GARG • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*
- JULIANA A. GERALDO • *Instituto René Rachou, Fundação Oswaldo Cruz, Belo Horizonte, Brazil*
- HAMIDEH HAJIABADI • *Division of Genetics and Genomics, Institute for Molecular Bioscience, The University of Queensland, Brisbane, QLD, Australia*
- IRENE LENA HUDSON • *Mathematical Sciences, School of Science, RMIT University, Melbourne, VIC, Australia*
- ADRIAAN P. IJZERMAN • *Drug Discovery and Safety, Leiden Academic Centre for Drug Research, Leiden, The Netherlands*
- ROBERT L. JERNIGAN • *Roy J. Carver Department of Biochemistry, Biophysics and Molecular Biology, Iowa State University, Ames, IA, USA*
- CATHERINE R. JUTZELER • *Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland; SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland*
- MALANCHA KARMAKAR • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*
- ANDRZEJ KLOCKOWSKI • *Battelle Center for Mathematical Medicine, The Research Institute at Nationwide Children’s Hospital, Columbus, OH, USA; Department of Pediatrics, The Ohio State University, Columbus, OH, USA*
- ANDRE LAMURIAS • *LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisbon, Portugal*
- XUHAN LIU • *Drug Discovery and Safety, Leiden Academic Centre for Drug Research, Leiden, The Netherlands*
- MARIO MALCANGI • *Computer Science Department, Università degli Studi di Milano, Milan, Italy*
- PEDRO MATOS-FILIPE • *Center for Innovative Biomedicine and Biotechnology, University of Coimbra, Coimbra, Portugal; Center for Neuroscience and Cell Biology, University of Coimbra, Coimbra, Portugal*
- MICHAEL MOOR • *Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland; SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland*
- IRINA S. MOREIRA • *Center for Innovative Biomedicine and Biotechnology, University of Coimbra, Coimbra, Portugal; Center for Neuroscience and Cell Biology, University of Coimbra, Coimbra, Portugal; Department of Life Sciences, University of Coimbra, Coimbra, Portugal*

JOANA MOURÃO • *Center for Innovative Biomedicine and Biotechnology, University of Coimbra, Coimbra, Portugal; Center for Neuroscience and Cell Biology, University of Coimbra, Coimbra, Portugal; Institute for Interdisciplinary Research, University of Coimbra, Coimbra, Portugal*

YOO CHAN MYUNG • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*

T. MURLIDHARAN NAIR • *Indiana University South Bend, South Bend, IN, USA*

KRATIKA NASKULWAR • *Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada*

QUAN NGUYEN • *Division of Genetics and Genomics, Institute for Molecular Bioscience, The University of Queensland, Brisbane, QLD, Australia*

HARDIK H. PARATE • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*

LOURDES PEÑA-CASTILLO • *Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada; Department of Biology, Memorial University of Newfoundland, St. John's, NL, Canada*

DOUGLAS E. V. PIRES • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia; School of Computing and Information Systems, University of Melbourne, Melbourne, VIC, Australia*

STEPHANIE PORTELLI • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*

ANTÓNIO J. PRETO • *Center for Innovative Biomedicine and Biotechnology, University of Coimbra, Coimbra, Portugal; Center for Neuroscience and Cell Biology, University of Coimbra, Coimbra, Portugal; Institute for Interdisciplinary Research, University of Coimbra, Coimbra, Portugal*

DEREK REIMAN • *Department of Bioengineering, University of Illinois at Chicago, Chicago, IL, USA*

PÂMELA M. REZENDE • *Instituto René Rachou, Fundação Oswaldo Cruz, Belo Horizonte, Brazil*

CARLOS H. M. RODRIGUES • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*

SANDHYA SAMARASINGHE • *Complex Systems, Big Data and Informatics Initiative (CSBII), Lincoln University, Christchurch, New Zealand*

- MICHAEL SILK • *Structural Biology and Bioinformatics, Department of Biochemistry and Molecular Biology, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; ACRF Facility for Innovative Cancer Drug Discovery, Bio21 Institute, University of Melbourne, Melbourne, VIC, Australia; Computational Biology and Clinical Informatics, Baker Heart and Diabetes Institute, Melbourne, VIC, Australia*
- DIANA SOUSA • *LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisbon, Portugal*
- ANDREW T. SU • *Division of Genetics and Genomics, Institute for Molecular Bioscience, The University of Queensland, Brisbane, QLD, Australia*
- XIAO TAN • *Division of Genetics and Genomics, Institute for Molecular Bioscience, The University of Queensland, Brisbane, QLD, Australia*
- MINH TRAN • *Division of Genetics and Genomics, Institute for Molecular Bioscience, The University of Queensland, Brisbane, QLD, Australia*
- GERARD J. P. VAN WESTEN • *Drug Discovery and Safety, Leiden Academic Centre for Drug Research, Leiden, The Netherlands*
- JOÃO P. L. VELLOSO • *Instituto René Rachou, Fundação Oswaldo Cruz, Belo Horizonte, Brazil*
- HAO WANG • *Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI, USA*
- JIANRONG WANG • *Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI, USA*
- JOICYMARA S. XAVIER • *Instituto René Rachou, Fundação Oswaldo Cruz, Belo Horizonte, Brazil*
- JIAXIN YANG • *Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI, USA*



# Chapter 1

## Identifying Genotype–Phenotype Correlations via Integrative Mutation Analysis

**Edward Airey, Stephanie Portelli, Joicymara S. Xavier, Yoo Chan Myung, Michael Silk, Malancha Karmakar, João P. L. Velloso, Carlos H. M. Rodrigues, Hardik H. Parate, Anjali Garg, Raghad Al-Jarf, Lucy Barr, Juliana A. Geraldo, Pâmela M. Rezende, Douglas E. V. Pires , and David B. Ascher**

### Abstract

Mutations in protein-coding regions can lead to large biological changes and are associated with genetic conditions, including cancers and Mendelian diseases, as well as drug resistance. Although whole genome and exome sequencing help to elucidate potential genotype–phenotype correlations, there is a large gap between the identification of new variants and deciphering their molecular consequences. A comprehensive understanding of these mechanistic consequences is crucial to better understand and treat diseases in a more personalized and effective way. This is particularly relevant considering estimates that over 80% of mutations associated with a disease are incorrectly assumed to be causative. A thorough analysis of potential effects of mutations is required to correctly identify the molecular mechanisms of disease and enable the distinction between disease-causing and non-disease-causing variation within a gene. Here we present an overview of our integrative mutation analysis platform, which focuses on refining the current genotype–phenotype correlation methods by using the wealth of protein structural information.

**Key words** Genotype–phenotype correlations, Graph-based signatures, mCSM, Mutation, Protein structure, Protein interactions

---

### 1 Introduction

Proteins are versatile molecules, responsible for orchestrating a wide range of biological processes. They comprise a single polypeptide chain of amino acids, which folds in 3D space into dynamic structures. How a protein folds is important for determining its functions, including activities and interactions with other molecules. These structures are highly coordinated and conserved across evolution, and small perturbations in the amino acid sequence can disrupt these shapes, functions, and interactions [1, 2]. While

missense mutations, causing a change to a single amino acid, are generally less structurally disruptive than nonsense mutations, their effects are highly variable and can be wide-ranging, making their molecular consequences harder to determine. Despite their subtle effects, missense substitutions are related with many different genetic conditions, including cancer, Mendelian diseases, and the emergence of drug resistance.

The introduction of a missense mutation can have many molecular effects, including altering how the protein folds, its dynamics, posttranslational modifications, half-life, localization, activity, and molecular interactions [3]. When analyzing a new mutation, an integrative approach is therefore important to consider the effects it might have on all of these aspects. This enables the identification of specific functional, and structural changes imparted by the mutations, which is essential for a molecular understanding. It can also explain why mutations in the same protein might lead to different diseases, why mutations might cluster in 3D space and how those genetic changes present phenotypically.

Although many assume that an unfavorable phenotype (e.g., pathogenic, drug-resistant) is the result of large, overall destabilizing mutations, mutations with milder effects are often more prevalent in a population, as they are generally under less selective pressure [4, 5]. For example, by assessing mutations in three different tuberculosis proteins that lead to resistance, we have shown that the most frequent resistant mutations were more likely to be associated with overall mild functional effects, and associated reduced fitness cost, allowing for increased prevalence within the bacterial population [4].

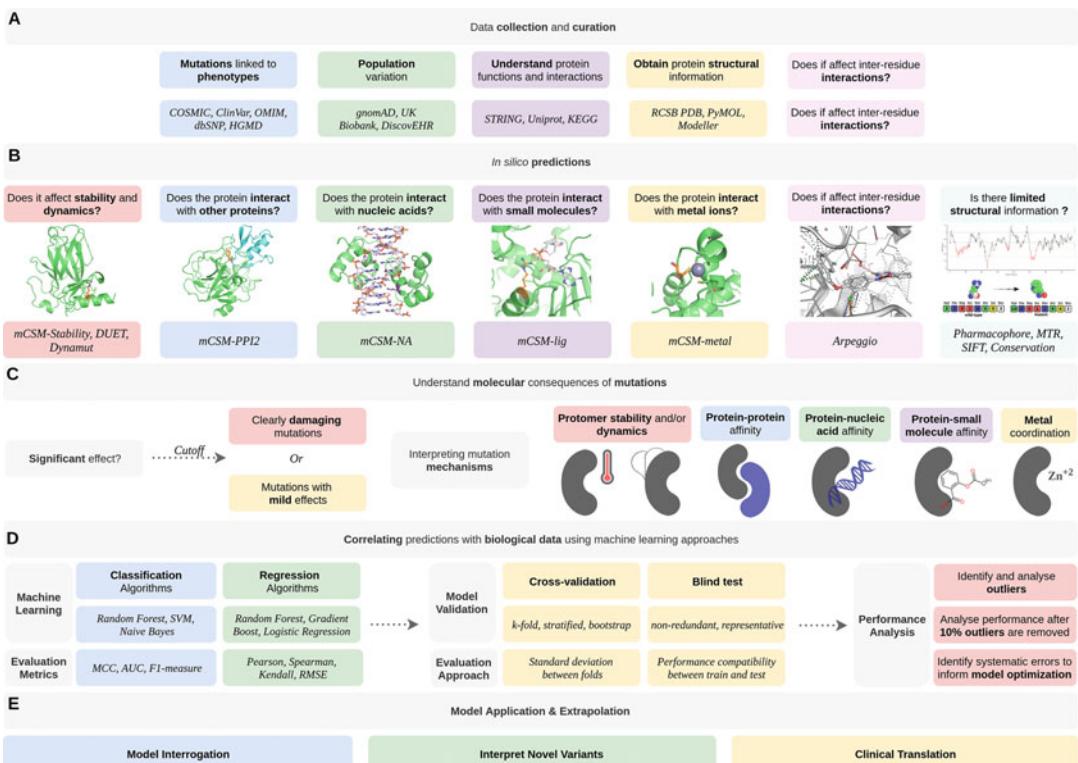
Experimentally elucidating the biophysical effects of mutations is an expensive and time-consuming task, usually limited to a few variants in proteins with amenable assays. Over the years, the accumulation of information of experimentally characterized mutations has enabled the development and improvement of computational mutational analysis tools [6]. These computational platforms have shown to be invaluable assets to decipher genotype–phenotype correlations in cancer [7–19], Mendelian diseases [20–26], and detection of antimicrobial resistance [4, 15, 27–35], guiding clinical decisions and driving further research. Here, we introduce a general computational pipeline that uses *in silico* biophysical predictions and machine learning approaches to harness the wealth of available biological and protein structural information and give insights into genotype–phenotype correlation for clinical use [10].

The mutation cutoff scanning matrix (mCSM) platform is the only comprehensive collection of *in silico* tools for quantitatively predicting the effects of missense mutations on protein folding, structure, dynamics, and interactions. It includes tools which calculate all possible molecular interactions (Arpeggio [36]), account for changes in protein stability (mCSM-Stability [37], SDM [38],

DUET [39], mCSM-membrane [40], dynamics (DynaMut [41]), protein interactions with other proteins (mCSM-PPI [37], mCSM-PPI2 [42], mCSM-AB [43], mCSM-AB2 [44], mmCSM-AB [45], nucleic acids (mCSM-DNA [37], mCSM-NA [46]), and small molecule ligands (mCSM-lig [47], CSM-lig [48]).

These tools were built using the concept of graph-based signatures [49, 50], which represent the geometry and physicochemical properties of the wild-type protein structure environment as a network or graph, composed of a series of nodes, describing the local mutation environment, and edges, describing the distances between interacting “layers” of surrounding residues. Information on the mutation is captured using the pharmacophore change between the wild-type and the mutant residue, including whether charges or hydrogen donors/acceptors have been gained or lost [37].

This platform allows for accurate biophysical predictions, which, when complemented with other protein analytical tools, can provide a detailed landscape on the specific mutational effects on a protein. We have implemented these within an analytical and supervised machine learning predictive pipeline (Fig. 1), to enable easy and fast characterization of novel mutations and their likely



**Fig. 1** An overview of the mechanistic characterization of mutations and their biological consequences, to guide the development of tools to predict phenotypic outcomes

clinical phenotypes. This approach has been shown to have big implications in diagnostic and personalized medicine in the post-genomic era.

---

## 2 Materials

### 2.1 Data Curation

#### 2.1.1 Mutation Curation

The foremost requirement for training a machine learning model is appropriate high-quality experimental/clinical data, with suitable representation of the classes under comparison. For human disease, a wealth of freely accessible collections of curated data exist. Previously reported mutations through publications and functional studies are available from dbSNP [51], the largest freely available repository of genetic variation. Variants with evidence of pathogenicity can be viewed from the Human Gene Mutation Database (HGMD) [52] and ClinVar [53], and from disease-specific datasets such as the Catalogue of Somatic Mutations in Cancer (COSMIC). Standing variation is available from genomic sequencing efforts of healthy populations, including over 140,000 healthy humans in gnomAD [54] and 50,000 whole exomes currently available in UK Biobank [55].

When combining data from multiple sources, it is important that all datapoints are comparable. If using genetic coordinates, they should be found on the same assembly of the genome (e.g., GRCh38 vs GRCh37). The mutations themselves (whether reported as genetic or amino acid changes) must be reported on the same transcript, as most genes have multiple reported coding sequences.

#### 2.1.2 Protein Structure Curation

The sequence and functional information for a specific protein can be obtained from Uniprot (<https://www.uniprot.org/>) [56]. To run the mCSM tools we need crystallographic structures, which can be downloaded from the Protein Data Bank (PDB;<http://www.rcsb.org/>) [57] or generated via homology modeling or molecular docking (to run mCSM-PPI, mCSM-Lig, or mCSM-NA). Once we have the variant information collected from the resources in Subheading 2.1.1, we map these variants on to the identified protein structures to help visualize the spread and identify potential hotspots, which is easily done using visualization software such as PyMol, as it enables selection of residues being mutated in a 3D manner.

### 2.2 An Overview of Computational Tools to Analyze Missense Mutations

Over the past two decades there has been an unprecedented growth in both computational power and the amount of biological data available. This has facilitated the development of numerous sequence (Table 1) and structural (Table 2) based computational tools to guide mutation characterization.

**Table 1**  
Available sequence-based predictive tools for mutation analysis

<b>Protein stability and dynamics</b>		<b>Corr.<sup>a</sup></b>
<b>Method</b>		
I-Mutant 2.0		0.62
Auto-Mute		0.64 <sup>a</sup>
MUpro		0.75
DynaMine		0.63
DDGun		0.49
INPS-MD/3D		0.58
iStable		0.56 <sup>b</sup>
iPTREEE - STAB		0.70
ProMaya		0.79

<sup>a</sup>Pearson's correlation

<sup>b</sup>MCC

**Table 2**  
Available structure-based predictive tools for mutation analysis

<b>Protein stability and dynamics</b>		<b>Protein–protein affinity</b>		<b>Protein–nucleic acid affinity</b>		<b>Protein–small molecule affinity</b>	
<b>Method</b>	<b>Corr.<sup>a</sup></b>	<b>Method</b>	<b>Corr.<sup>b</sup></b>	<b>Method</b>	<b>Corr.<sup>c</sup></b>	<b>Method</b>	<b>Corr.<sup>d</sup></b>
mCSM-Stability	0.69	mCSM-PPI	0.16	mCSM-NA	0.70	mCSM-lig	0.63
DUET	0.68	mCSM-PPI2	0.42				
DynaMut	0.70	BeAtMuSiC	0.28				
SDM2	0.61	MutaBind	0.41				
STRUM	0.79	FoldX	0.12				
PopMuSiC 2.1	0.63	MMPBSA	0.19				
CUPSAT	0.78						
Eris	0.75						
INPS-MD/3D	0.72						

<sup>a</sup>Pearson's correlation when evaluated on blind-test sets derived from the ProTherm database

<sup>b</sup>Kendall rank correlation coefficient on 1007 single-point mutations from CAPRI (T55)

<sup>c</sup>Pearson's correlation on 331 single-point mutations from 38 protein–nucleic acid complexes

<sup>d</sup>Pearson's correlation on 763 single-point mutations from 200 protein–ligand complexes

The mCSM platform is the only available approach to consider all possible molecular effects and has therefore formed the central component of our mutational analysis pipeline. All mCSM

Platform tools are available freely as websites compatible with most web-browsers, but Google Chrome is recommended. A summary of these methods and links to access them is described in Table 3.

**Table 3**  
**Computational tools available in the mCSM platform**

mCSM tool	Type	Function
Arpeggio <sup>a</sup>	Protein interaction	Calculates 13 different types of interactions between atoms including hydrogen bonds, halogen bonds, carbonyl interactions, and others.
MTR-Viewer <sup>b</sup>	Missense tolerance	A measure of a gene's regional tolerance to missense variation.
mCSM-Stability <sup>c</sup>	Stability	Predict the effects of a mutation on the overall protein stability
SDM2 <sup>d</sup>	Stability	Predicts the change in protein stability due to a single mutation using conformationally constrained environment-dependent amino acid substitution tables.
DUET <sup>e</sup>	Stability	Uses mCSM-Stability and SDM2 in order to create a consensus prediction the effects of a mutation on protein stability
DynaMut <sup>f</sup>	Flexibility	Looks to predict the effects of a mutation on protein stability, flexibility, and dynamics
mCSM-PPI <sup>g</sup>	Protein interaction	Predicts the effects of a mutation within a specified protein on its impact with overall protein-protein interactions.
mCSM-PPI2 <sup>h</sup>	Protein interaction	Creates a similar prediction to PPI but incorporates the effects of mutations on interresidue noncovalent interaction network using graph kernels, evolutionary information, complex network metrics, and energetic terms.
mCSM-DNA <sup>i</sup>	Protein interaction	Predicts the impact of mutations on the protein interaction with DNA.
mCSM-NA <sup>j</sup>	Protein interaction	Predicts the impact of mutations on the protein interaction with nucleic acids, and uses pharmacophore and information about nucleic acid properties.
mCSM-Lig <sup>k</sup>	Protein interaction	Predicts the effects of single-point mutations on the stability of a protein-ligand complex.

<sup>a</sup><http://biosig.unimelb.edu.au/arpeggioweb/>

<sup>b</sup><http://biosig.unimelb.edu.au/mtr-viewer/>

<sup>c</sup><http://biosig.unimelb.edu.au/mcsm/stability>

<sup>d</sup><http://marid.bioc.cam.ac.uk/sdm2>

<sup>e</sup><http://biosig.unimelb.edu.au/duet/>

<sup>f</sup><http://biosig.unimelb.edu.au/dynamut/>

<sup>g</sup>[http://biosig.unimelb.edu.au/mcsm/protein\\_protein](http://biosig.unimelb.edu.au/mcsm/protein_protein)

<sup>h</sup>[http://biosig.unimelb.edu.au/mcsm\\_ppi2/](http://biosig.unimelb.edu.au/mcsm_ppi2/)

<sup>i</sup>[http://biosig.unimelb.edu.au/mcsm/protein\\_dna](http://biosig.unimelb.edu.au/mcsm/protein_dna)

<sup>j</sup>[http://biosig.unimelb.edu.au/mcsm\\_na/](http://biosig.unimelb.edu.au/mcsm_na/)

<sup>k</sup>[http://biosig.unimelb.edu.au/mcsm\\_lig/](http://biosig.unimelb.edu.au/mcsm_lig/)

---

### 3 Methods

#### 3.1 Predicting and Analyzing Structural and Biophysical Effects of Mutations Using the mCSM Platform

The mCSM methods can be categorized by purpose. As shown in Fig. 1, methods are chosen depending on interactions made, and what structural information is available. Below we discuss how each type of predictor can be used and interpreted.

- The user should choose the appropriate tools based on what information is available on their protein of interest (Fig. 1).
- In general, each mCSM tool requires a wild-type protein file, in the PDB format, and the single-point mutation or a list of mutations. Some tools may require additional specific information; Table 4 shows the inputs required for each tool. Notes 1 and 2 highlight some common issues with the submission inputs.

#### 3.2 mCSM Platform Output

##### 3.2.1 Arpeggio

The results of Arpeggio are shown in Fig. 2.

- After submitting a job, an overview of the type and number of atomic interactions within the protein is shown (Fig. 2a). Arpeggio calculates all types of molecular interactions (Table 5), which are displayed and downloadable along with a visual representation of the atomic contacts overlaid on the protein structure (Fig. 2b).
- The number of each interaction/contact and PyMOL session files can be downloaded for a more detailed analysis.

##### 3.2.2 MTR-Viewer

##### Gene Viewer

- The MTR gene viewer [5] results page (Fig. 3) shows predicted MTR scores in an interactive line graph with a control panel which allows users to adjust the window size and the ethnicity for MTR estimates. A line graph (Fig. 3a) displays regions that have high variation, low-MTR scored; those in red are most likely to be pathogenic. Any ethnicity-specific MTR scores are shown in blue on the line graph.

- The first lollipop plot (Fig. 3b) shows observed missense (yellow) and synonymous (green) variations based on gnomeAD.
- If the gene of interest is a ClinVar pathogenic gene, their pathogenic (red) and benign (blue) missense variants are displayed under the gnomeAD lollipop plot (Fig. 3c).
- Users can browse results of alternative-transcript (Fig. 3d) of the given query if available.

##### Variant Query

- The variant query result page (Fig. 4) shows MTR scores for each user-supplied missense variant, providing the estimated regional intolerance. Low MTR scores indicate stronger purifying selection within the population. Users can also press “view” next to a variant to show its position within its gene transcript.

**Table 4**  
**Information required to run each mCSM program**

		<b>Inputs</b>	
<b>mCSM tool</b>	<b>Task</b>	<b>Step 1</b>	<b>Step 2</b>
Arpeggio	Calculate	Molecule in PDB format or PDB accession code.	Select desired interaction calculation. You can select any (including multiple) part of the PDB file using the syntax: /1/2/3 Where: 1. Chain ID. 2. Residue number. 3. Atom name.
MTR-Viewer	Gene Variant Viewer Queries	Gene, ensembl ID, or Refseq ID Variants as GrCh37 genomic coordinates.	Select window size and overlay sub-population
mCSM-Stability, mCSM-PPI, mCSM-DNA	Prediction	Wild-type protein file in PDB format. For mCSM-PPI and mCSM-DNA, the structure of the complex in PDB format is required.	Single mutation (code and mutation chain), file with a list of mutations and its respective chains or code of residue and the mutation chain.
SDM2	Prediction	Wild-type protein structure in a PDB format or PDB accession code.	Single mutation (code and mutation chain) or residue/position code and the mutation chain.
DUET	Prediction	Wild-type protein structure in a PDB format or PDB accession code.	Single mutation (code and mutation chain)
DynaMut	Analysis	Wild-type protein structure in a PDB format or PDB accession code.	The selection of a Force Field and email (optional field).
	Prediction	Wild-type protein structure in a PDB format or PDB accession code.	Single mutation (code and mutation chain) or file with a list of mutations and its respective chains, and email (optional field).
mCSM-PPI2	Prediction	The structure of the complex in PDB format or corresponding PDB accession code.	Single mutation (code and mutation chain) or file with a list of mutations and its respective chains, and email (optional field).
	Analysis	The structure of the complex in PDB format or corresponding PDB accession code.	Mutation details (alanine scanning or saturation mutagenesis) and email (optional field).
mCSM-NA	Prediction	The structure of the complex in PDB format or corresponding PDB accession code.	Single mutation (code and mutation chain) or file with a list of mutations and its respective chains, and the selection of the Nucleic Acid Type.
mCSM-Lig	Prediction	The structure of the complex in PDB format or corresponding PDB accession code.	Single mutation (code and mutation chain) and ligand information (three-letter ligand ID and estimated wild-type affinity).

**A**

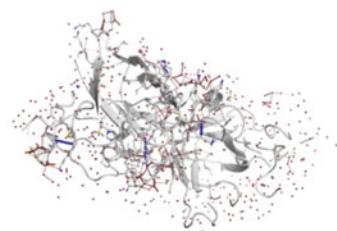
### Overview [2C9V.pdb]

Mutually Exclusive Interactions	
Total number of contacts	1853
Of which VdW interactions	27
Of which VdW clash interactions	188
Of which covalent interactions	8
Of which covalent clash interactions	0
Of which proximal	1626

Feature Contacts	
Hydrogen bonds	18
Water mediated hydrogen bonds	181
Weak hydrogen bonds	15
Water mediated weak hydrogen bonds	17
Halogen bonds	0
Ionic interactions	9
Metal complex interactions	0
Aromatic contacts	0
Hydrophobic contacts	32
Carbonyl interactions	7

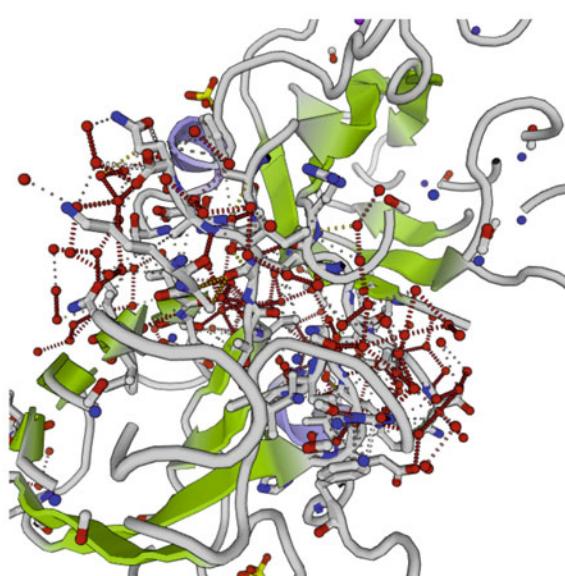
**Download All Results**

**B**

### Visualisation

Info WebGL must be available in your browser and enabled. [Check compatibility](#). This WebGL visualisation provides an overview of van der Waals distance interactions; for detailed analysis please download the [PyMOL session file](#).



**Fig. 2** Output of the Arpeggio tool. **(a)** Overview of the output for the inputted protein including the different types of interactions. **(b)** Visualization of the interactions shown on a protein structure

**Table 5**  
**Atomic interactions calculated by Arpeggio**

Atomic interaction	Description	Arpeggio class	Bond energy (kJ/mol)
Van der Waals (dipole)	Permanent, induced and instantaneous dipoles	VWD	1–9
Hydrophobic	Between aliphatic and aromatic atoms	Hydrophobic	4–12
Hydrogen bond	Between carboxyl, amide, imidazole, guanidine, amino, hydroxyl and phenolic groups	Hydrogen bonds, weak hydrogen bond, polar contacts, halogen bonds, carbonyl interactions	8–40
Pi interactions	From/to rings	Aromatic contacts	6–70
Electrostatic	Between carboxyl and amino groups	Ionic interactions, metal complex	42–84

### 3.2.3 mCSM-Stability/ PPI/DNA

The impact of mutations on protein stability, protein–protein binding affinity, and protein–DNA affinity can be predicted by mCSM-Stability, mCSM-PPI, mCSM-DNA with three types of prediction; single, multiple and systematic mutation.

#### Single Mutation

- If the single mutation option is selected in one of the tools within the mCSM platform, it will be shown on a results page after processing. This information includes the predicted value changes (protein stability, protein–protein interaction, protein–DNA interaction) as measured by the change in Gibbs Free Energy  $\Delta\Delta G$  kcal/mol (Fig. 5), which is classified as highly destabilizing ( $\Delta\Delta G \leq -2$  kcal/mol), destabilizing ( $-2 \text{ kcal/mol} < \Delta\Delta G < 0$  kcal/mol), stabilizing ( $0 \text{ kcal/mol} \leq \Delta\Delta G < 2$  kcal/mol), or highly stabilizing ( $\Delta\Delta G \geq 2$  kcal/mol).
- If the structure of a complex is submitted to mCSM-Stability, it will calculate the predicted change in stability of the entire complex. It is therefore often advisable to also run predictions on a PDB file containing the protomer chain alone.
- For mCSM-PPI and mCSM-DNA, for mutations further than 12 Å from the interaction, the mCSM predictions are not considered, and are set to 0, as the graph-based signatures capture a smaller radius of environmental data, and there are fewer mutations located further away than 12 Å in the datasets used to train the methods.
- Also shown is an interactive 3D visual representation of the uploaded PDB file (Fig. 5a, right).

**A** Gene Viewer

Gene, Ensembl ID or Refseq ID

Select window size (codons)

- 21
  - 31 (default)
  - 41
- Submit

Overlay sub-population

- All populations (default)
  - Latino
  - Non-Finnish European
  - South Asian
- Submit

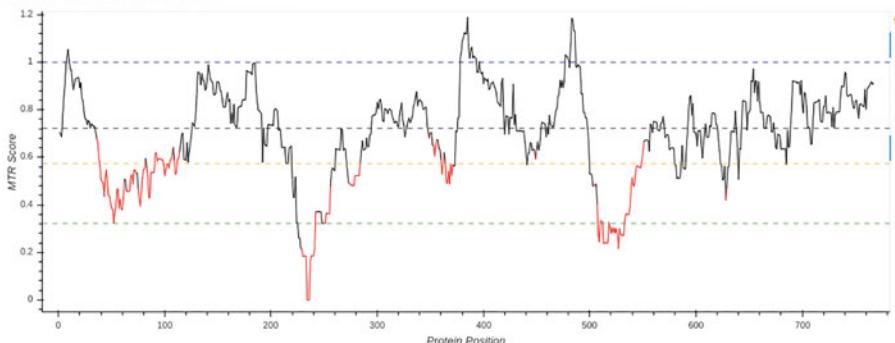
Download

Download MTR table

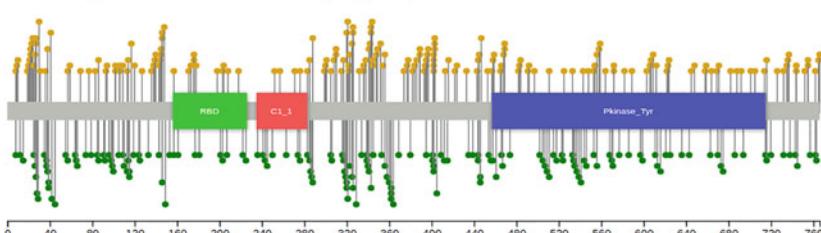
Download

Download MTR flat file

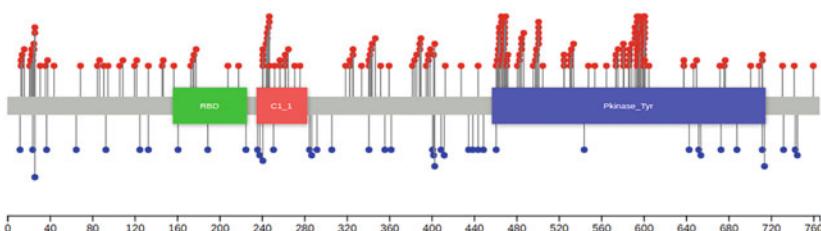
BRAF // ENST00000288602

**B**

gnomAD Variation (Yellow = Missense, Green = Synonymous)

**C**

ClinVar Variation (Red = Pathogenic missense, Blue = Benign missense)



Lollipops shown for canonical-matching UniProt accession where a valid Pfam domain can be retrieved.

**D**

Alternate matches (Currently selected in bold)

Feature	HGNC Symbol	CCDS	RefSeq	Canonical
ENST00000288602	BRAF	CCDS5863	NM_004333	Yes
ENST00000479537	BRAF	None	No match	-
ENST00000497784	BRAF	None	No match	-

**Fig. 3** The MTR Gene Viewer result page. **(a)** The line graph shows MTR scores in red for variations distant from neutrality across the transcript according to selected window size (codons) and subpopulation option. **(b)** The lollipop plot shows observed gnomAD variation in yellow and green for missense and synonymous variation. **(c)** The second lollipop plot displays pathogenic (red) and benign (blue) missense variants based on ClinVar annotation. **(d)** The alternate transcripts can be shown in a table with RefSeq ID

Chrom	Genomic Pos	Ref	Alt	Feature	Protein Pos	Consequence	Mis + Syn tally	Observed ratio	Expected ratio	MTR	FDR	View
19	58048839	G	A	ENST00000240719	143	missense_variant	10	1	0.806	1.241	None	<button>View</button>
19	58048839	G	A	ENST00000376233	156	missense_variant	10	1	0.806	1.241	0.574	<button>View</button>
19	58048839	G	C	ENST00000240719	143	missense_variant	10	1	0.806	1.241	None	<button>View</button>
19	58048839	G	C	ENST00000376233	156	missense_variant	10	1	0.806	1.241	0.574	<button>View</button>
19	58048839	G	T	ENST00000240719	143	missense_variant	10	1	0.806	1.241	None	<button>View</button>
19	58048839	G	T	ENST00000376233	156	missense_variant	10	1	0.806	1.241	0.574	<button>View</button>

**Fig. 4** MTR Variant Queries result page. Calculated results and information for the given input variants (or a CSV). User can check the details through MTR Gene Viewer by clicking on the view button

#### Multiple or Systematic

- If the option for inputting a list of mutations or systematic was used to analyze the PDB file, then after processing, results will be shown in tabulated form (Fig. 5b), including mutation specific information such as the residue solvent accessibility (RSA), as well as the predicted  $\Delta\Delta G$ .
- Each result is also classified, using the predicted  $\Delta\Delta G$  value, as highly destabilizing, destabilizing, stabilizing, or highly stabilizing.
- Users can search the result table or download results into a tab-separated text file.

#### 3.2.4 SDM

SDM uses environment-specific amino acid substitution tables [38] and structural features including residue depth [15] and packing density to predict the impact of mutations on protein stability. The result page of single and list mutation is as follows.

#### Single Mutation

- The single mutation result page (Fig. 6a) provides predicted protein stability changes ( $\Delta\Delta G$ ), in addition to structural information implemented in SDM including secondary structure, RSA, residue depth and residue occluded packing density (OSP), sidechain–sidechain hydrogen bond (HBOND\_SS), sidechain–main chain amide hydrogen bond (HBOND\_SN), and sidechain–main chain carbonyl hydrogen bond (HBOND\_SO). The integrated 3D viewer also shows the

mCSM

Protein Stability

Protein-Protein

Protein-DNA

Data sets

Contact

Acknowledgments

About

**A****Protein Stability Change Upon Mutation****Predicted Stability Change ( $\Delta\Delta G$ ):****-1.219 Kcal/mol (Destabilizing)****Mutation:**

Wild-type: R  
 Position: 282  
 Mutant-type: W  
 Chain: A



mCSM

Protein Stability

Protein-Protein

Protein-DNA

Data sets

Contact

Acknowledgments

About

**B****Protein-Protein Affinity Change Upon Mutation****Predicted Protein-Protein Affinity Change ( $\Delta\Delta G$ ):**

Index	PDB File	Chain	Wild Residue	Residue Position	Mutant Residue	RSA (%)	Predicted $\Delta\Delta G$	Outcome
1	1cse.pdb	I	L	37	A	21.5	0.043	Stabilizing
2	1cse.pdb	I	L	37	V	21.5	-0.119	Destabilizing
3	1cse.pdb	I	L	37	G	21.5	0.109	Stabilizing
4	1cse.pdb	I	L	37	S	21.5	0.177	Stabilizing
5	1cse.pdb	I	L	37	W	21.5	-0.599	Destabilizing
6	1cse.pdb	I	L	37	T	21.5	0.063	Stabilizing
7	1cse.pdb	I	L	37	Q	21.5	-0.21	Destabilizing
8	1cse.pdb	I	L	37	E	21.5	-0.618	Destabilizing
9	1cse.pdb	I	L	37	C	21.5	-0.39	Destabilizing
10	1cse.pdb	I	L	37	R	21.5	-0.177	Destabilizing

Showing 1 to 10 of 19 entries

← Previous 1 2 Next →

**Fig. 5** Result pages for mCSM-Stability, mCSM-PPI and mCSM-DNA. **(a)** mCSM-Stability (single mutation) and **(b)** mCSM-PPI (multiple/systematic mutation). **(a)** The single prediction for example mCSM-Stability page supports 3D interactive viewer for structural analysis. **(b)** The results and information from multiple/systematic prediction for example mCSM-PPI are shown in a table

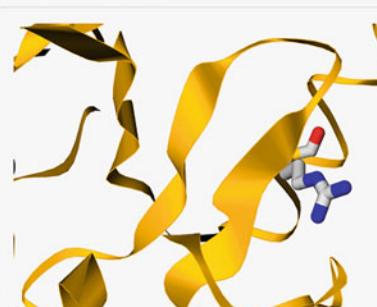
Site Directed Mutator  Predict Help Data sets Contact Acknowledgements Related Resources

### A Change In Protein Stability Upon Mutation

**Predicted pseudo  $\Delta\Delta G$ :**  
**-0.06 (Reduced stability)**

**Mutation:**  
PDB name: 2OCJ.pdb  
Chain: A  
Wild-type: ARG  
Position: 282  
Mutant-type: TRP

Environment	Wild type Mutant
Secondary structure	H H
Solvent accessibility (%)	16.1 19.9
DEPTH (Å)	5.1 4.8
OSP	0.55 0.51
HBOND_SS	True True
HBOND_SN	False False
HBOND_SO	True False



View Rotate Translate Zoom Slab Reset view

**C Run another prediction** **M Molecule Visualization**

**B**

**Predicted Stability Change ( $\Delta\Delta G$ ):**

Index	PDB File	Chain ID	Mutation	WT_SSE	WT_RSA (%)	WT_DEPTH (Å)	WT_OSP	WT_SS	WT_SN	WT_SO	MT_SSE	MT_RSA (%)	MT_DEPTH (Å)	MT_OSP	MT_SS	MT_SN	MT_SO	Predicted	
																		$\Delta\Delta G$	Outcome
1	2ocj.pdb	A	R282W	H	16.1	5.1	0.55	True	False	True	H	19.9	4.8	0.51	True	False	False	-0.06	Reduced stability
2	2ocj.pdb	A	Y236F	E	0.0	11.6	0.49	True	False	False	E	1.0	11.5	0.48	False	False	False	-0.49	Reduced stability
3	2ocj.pdb	A	N239Y	p	39.0	3.8	0.46	False	False	False	p	39.1	3.8	0.41	False	False	False	-0.93	Reduced stability
4	2ocj.pdb	A	C242S	p	16.7	4.1	0.43	True	True	True	p	22.6	3.8	0.34	True	False	False	0.48	Increased stability

Showing 1 to 4 of 4 entries

**C Run another prediction** **D Download results**

**Fig. 6** SDM prediction results for single and list prediction. **(a)** The single prediction displays the predicted  $\Delta\Delta G$  with information used on the left panel and 3D structure in a ribbon (protein) and a stick (wild-type amino acid) representation. **(b)** The list prediction gives detailed structural information and predicted  $\Delta\Delta G$  in a tabulated form highlighted according to stabilizing (blue) and destabilizing (red) mutation

structure and its wild-type amino acids in ribbon and stick representation.

- Stability changes ( $\Delta\Delta G$ ) are shown in red with a negative sign if the mutation is predicted to be destabilizing, and in blue with a positive sign if the mutation is predicted to be stabilizing.

**Multiple Mutations**

- The predicted SDM  $\Delta\Delta G$  for a given mutation list is displayed in a tabulated format (Fig. 6b) with their structural features. Users can download all mutant PDB structures and their predicted values in individual files.

**3.2.5 DUET****Single Mutation**

- The DUET result page (Fig. 7a) provides the predicted stability changes ( $\Delta\Delta G$ ) with integrated features such as secondary structure and stability changes from mCSM and SDM. While DUET refers to both mCSM and SDM scores, the prediction result can vary between the two methods.
- In the structure viewer (Fig. 7a right), the wild-type amino acid is shown in stick form and users can download the corresponding mutant structure file in PDB format.

**Systematic Mutations**

- With the systematic prediction (Fig. 7b), users can examine the predicted changes in protein stability using DUET, mCSM, and SDM for all nineteen possible mutations at a given residue position.
- The predictions and the structural information used to calculate the DUET scores are displayed in a downloadable table.

**3.2.6 DynaMut**

Users can use DynaMut to assess the impact of mutations on protein dynamics and stability with single and list mutation prediction.

**Single Mutation**

- The results of mutational effects on protein dynamics and stability are shown in Fig. 8a:  $\Delta\Delta G$  predictions, interatomic interactions, deformation and fluctuation analysis.
- The  $\Delta\Delta G$  prediction page provides predicted values from normal mode analysis (NMA)-based prediction ( $\Delta\Delta G$  ENCoM), vibrational entropy energy changes ( $\Delta\Delta S_{\text{Vib}}$  ENCoM), and other structure-based stability predictions ( $\Delta\Delta G$  mCSM,  $\Delta\Delta G$  SDM,  $\Delta\Delta G$  DUET). Users can visually assess mutational effects on protein flexibility which is colored on the protein structure by vibrational entropy (Fig. 8b) for the region gaining (red) or losing (blue) flexibility. This 3D representation can be downloaded into a Pymol session, high resolution image and CSV file.
- Through the interatomic interactions tab, users can compare molecular interactions between wild-type and mutant structures. The PDB structure with interatomic interactions can be retrieved as a Pymol session file.
- The mutational effects on protein dynamics are shown in the deformation and fluctuation tab. Users can evaluate changes in the amount of local flexibility and atomic fluctuation upon mutation in 3D visual representation; results are downloadable as a CSV file and a Pymol session file.

**A** DUET - Protein Stability Change Upon Mutation

mCSM Predicted Stability Change ( $\Delta\Delta G$ ):  
-2.365 Kcal/mol (Destabilizing)

SDM Predicted Stability Change ( $\Delta\Delta G$ ):  
-3.36 Kcal/mol (Destabilizing)

DUET Predicted Stability Change ( $\Delta\Delta G$ ):  
-2.664 Kcal/mol (Destabilizing)

**Mutation:**  
Wild-type: ILE  
Position: 232  
Mutant-type: THR  
Chain: A  
Secondary structure: Loop or irregular

Run another prediction
Download mutant PDB file
Molecule Visualization

Rotate  
 Translate  
 Zoom  
 Slab

DUET
Protein Stability
Help
Contact
Acknowledgments
Related Resources

**B** Protein Stability Change Upon Mutation

Predicted Stability Change ( $\Delta\Delta G$ ):
Search:

Index	Chain	Wild Residue	Residue Position	Mutant Residue	RSA (%)	mCSM predicted $\Delta\Delta G$	SDM predicted $\Delta\Delta G$	DUET predicted $\Delta\Delta G$
1	A	I	232	A	9.2	-2.372	-4.27	-3.071
2	A	I	232	V	9.2	-1.408	-1.91	-1.588
3	A	I	232	L	9.2	-0.959	-0.58	-0.737
4	A	I	232	G	9.2	-2.871	-2.05	-3.22
5	A	I	232	S	9.2	-2.694	-2.55	-2.879
6	A	I	232	W	9.2	-1.759	-1.16	-1.696
7	A	I	232	T	9.2	-2.365	-1.53	-2.343
8	A	I	232	Q	9.2	-1.943	-1.25	-1.832
9	A	I	232	E	9.2	-2.167	-0.84	-1.994
10	A	I	232	C	9.2	-1.509	-1.31	-1.559

Showing 1 to 10 of 19 entries
← Previous
1
2
Next →

Run another prediction

**Fig. 7** DUET result pages for single and systematic prediction. **(a)** The single prediction result of DUET shows predicted  $\Delta\Delta G$  across SDM and mCSM-Stability with mutation details. **(b)** Systematic prediction results including  $\Delta\Delta G$  from DUET, SDM and mCSM-Stability and relative solvent accessible area of wild-type structure

## DynaMut - Prediction Outcomes

[Run another prediction](#)

## Submission details

Wild-type: ILE

Position: 232

Mutant: THR

Chain: A

**A**[ΔΔG Predictions](#)[Interatomic Interactions](#)[Deformation and Fluctuation Analysis](#)

## Prediction Outcome

ΔΔG: **-1.942 kcal/mol (Destabilizing)**

## NMA Based Predictions

ΔΔG ENCoM: **-0.331 kcal/mol (Destabilizing)**

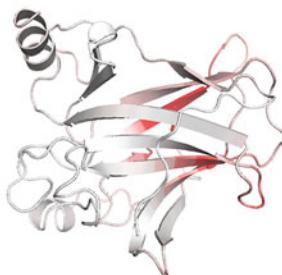
## Other Structure-Based Predictions

ΔΔG mCSM: **-2.365 kcal/mol (Destabilizing)**ΔΔG SDM: **-3.360 kcal/mol (Destabilizing)**ΔΔG DUET: **-2.664 kcal/mol (Destabilizing)****B**

## Δ Vibrational Entropy Energy Between Wild-Type and Mutant

ΔΔS<sub>Vib</sub> ENCoM: **0.414 kcal.mol<sup>-1</sup>.K<sup>-1</sup>** (Increase of molecule flexibility)

Δ Vibrational Entropy Energy | Visual representation

Amino acids colored according to the vibrational entropy change upon mutation. **BLUE** represents a rigidification of the structure and **RED** a gain in flexibility

## Download Resources

[Pymol Sessions](#)[High Resolution Images](#)[Additional data](#)  [Δ Flexibility Analysis](#) [Δ Flexibility Analysis](#) [ΔΔS per Residue](#) [Eigenvectors Wild-type](#) [Eigenvectors Mutant](#)

## DynaMut - Predictions Outcomes

**C**

#	AA from	AA to	Position	Prediction ΔΔG ENCoM	ΔΔS ENCoM	ΔΔG DynaMut	Action
1	C	A	109	<b>-0.218 kcal/mol</b>	0.272 kcal.mol <sup>-1</sup> .K <sup>-1</sup>	<b>-1.314 kcal/mol</b>	<a href="#"></a>
2	H	A	126	<b>-1.024 kcal/mol</b>	1.28 kcal.mol <sup>-1</sup> .K <sup>-1</sup>	<b>-1.105 kcal/mol</b>	<a href="#"></a>
3	C	A	121	<b>-0.306 kcal/mol</b>	0.382 kcal.mol <sup>-1</sup> .K <sup>-1</sup>	<b>0.646 kcal/mol</b>	<a href="#"></a>
4	C	A	64	<b>-0.593 kcal/mol</b>	0.742 kcal.mol <sup>-1</sup> .K <sup>-1</sup>	<b>-1.96 kcal/mol</b>	<a href="#"></a>

[Run another prediction](#) [Download results](#) [Download resources](#)

**Fig. 8** DynaMut result pages. The single prediction shows predicted DynaMut ΔΔG (a, left) and predicted protein stability (ΔΔG) from mCSM-Stability, SDM and DUET and flexibility changes (ΔΔG ENCoM). Users can check vibrational energy changes upon mutation in the panel B. For a multiple mutation, (b) list prediction result page shows predicted DynaMut ΔΔG and links to access the corresponding single prediction in table

- Multiple Mutations**
- For a given mutation list, DynaMut gives all predicted values, including  $\Delta\Delta G_{\text{Stability}}$  ENCoM,  $\Delta\Delta S_{\text{Vib}}$  ENCoM, and  $\Delta\Delta G_{\text{Stability}}$  DynaMut, in table format (Fig. 8c). A more detailed analysis is available through the single prediction page of each mutation by clicking on the “Detail” button.
- 3.2.7 mCSM-PPI2**
- mCSM-PPI2 supports two types of protein–protein affinity prediction: mutation prediction and binding analysis. Mutation prediction gives predicted protein–protein affinity changes based on a given protein–protein complex and the mutation information. Binding analysis considers interface residues within 5 Å from different chains in the complex structure for alanine scanning and saturation mutagenesis.
- Single Mutation**
- mCSM-PPI2 displays predicted binding affinity changes ( $\Delta\Delta G$ ) upon mutation in two classes, destabilizing and stabilizing. Mutation details such as the distance to the interface from the given mutation position are also shown (Fig. 9).
  - For mutations further than 12 Å from the interaction, the mCSM predictions are not considered, and are set to 0, as the graph-based signatures capture a smaller radius of environmental data, and there were fewer mutations located further away than 12 Å in the datasets used to train the methods.
  - Users can assess the mutational impact in atomic/residue level through a 3D interactive viewer and a 2D graph. The molecular viewer provides Arpeggio inter/intra interactions for wild-type and mutant structures and the interaction changes between wild-type and mutant allows for investigation of the relationship between nonbonded interaction and protein–protein affinity. For residue-level analysis, the 2D graph can be used to study interresidue interactions of wild-type and mutant in a simple and user-friendly representation.
- List Mutation**
- For multiple mutation analysis, the result page tabulates predicted  $\Delta\Delta G$  with mutation details. Users can access detailed results of each mutation through the single mutation result page and download all entries as a CSV file.
- Alanine Scanning**
- To identify residues with a greater contribution to the energy of binding (hot-spot) at the interface of interaction, alanine scanning can be used by predicting protein–protein binding affinity changes upon mutations to alanine across all identified interface residues. The predicted  $\Delta\Delta G$  values are displayed in table, bar chart, and 3D viewer (Fig. 10a).
  - Users can assess the effects of alanine mutation on the interface residues through a bar graph and 3D viewer colored in red and blue for destabilizing and stabilizing mutations, respectively.

**mCSM-PPI2**

Run Data Help Contact Acknowledgements Related Resources

### Single-Point Mutation

**Results**

**Predicted Affinity Change ( $\Delta\Delta G^{\text{Affinity}}$ )**  
**-2.019 kcal/mol**  
(Decreasing affinity)

**Mutation Details**

Chain: I  
Position: 45  
Wild-type: LEU  
Mutant: GLY  
Distance from closest partner: 2.72 Å

**Interactive Viewer**

Structure: Wild-type | Background: White | Representation: Cartoon | Color Scheme: by Chain

Interactions: Clash, Aromatic, VDW, Hydrophobic, Hydrogen Bond, Carbonyl, Ionic, Polar

RESET SPIN SCREENSHOT DOWNLOAD FULLSCREEN HELP

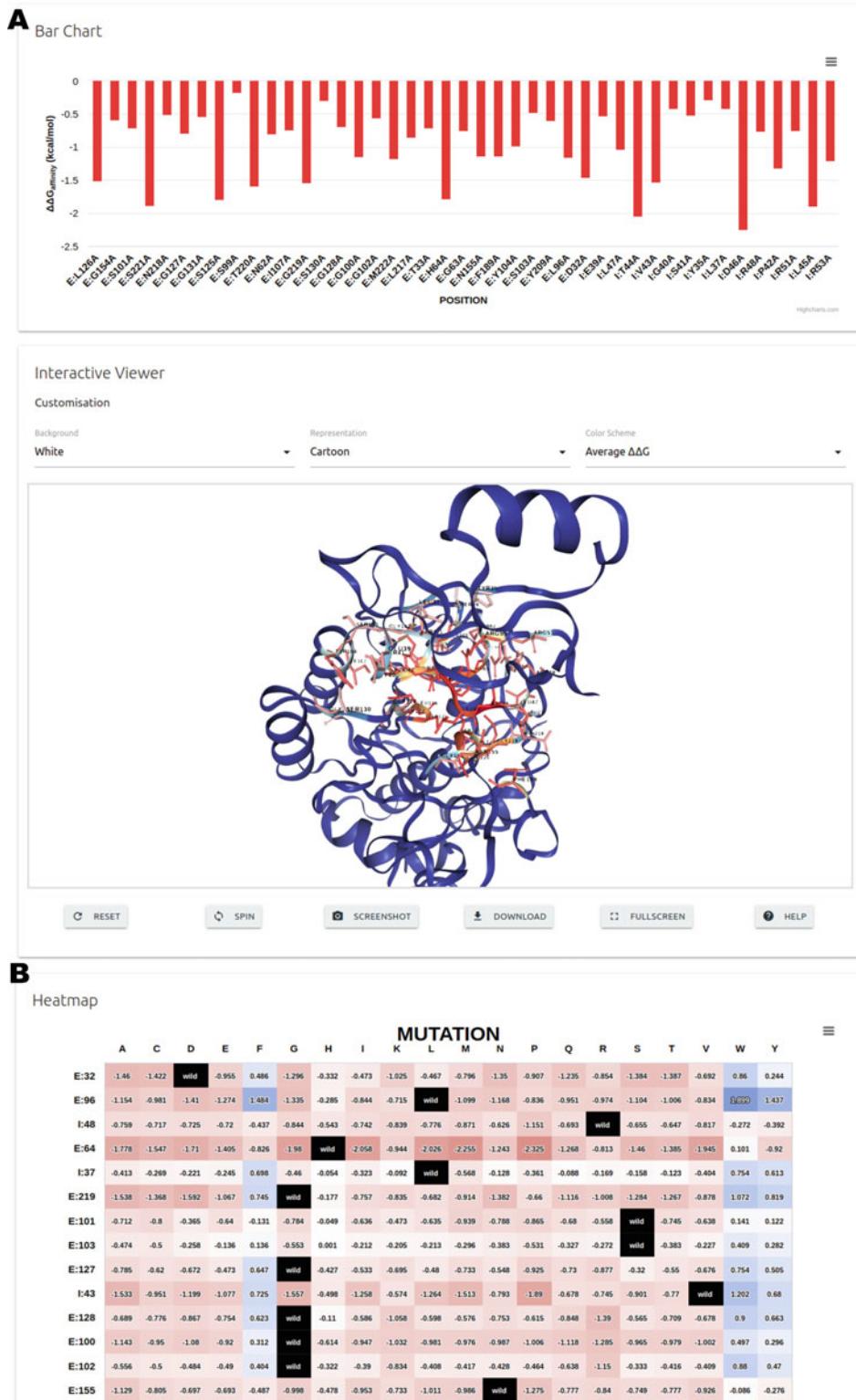
**Interactions: Wild-Type x Mutant**

Controls

Wild-Type (dashed lines) | Mutant (straight lines)

Carbonyl, Ionic, Hydrogen bonds, Polar Hbond, Ring-Ring | Aromatic, VDW, Hydrophobic, Clash

**Fig. 9** mCSM-PPI2 single prediction result page. The predicted  $\Delta\Delta G$  is shown along with two interaction viewers: 3D interactive molecule viewer for atomic interaction analysis and 2D diagram for residue-level interaction analysis



**Fig. 10** mCSM-PPI2 interface scanning result pages. The result pages of (a) alanine scanning and (b) saturation mutagenesis provide a bar chart and a heatmap colored by predicted  $\Delta\Delta G$  and average predicted  $\Delta\Delta G$  from the nineteen possible mutations, respectively

**Saturation Mutagenesis**

- The saturation mutagenesis provides the most exhaustive prediction, showing predicted  $\Delta\Delta G$  for all identified interface residues when they are changed into nineteen different amino acids. The results are shown in table, heatmap, and 3D molecule viewer, and the interface residues of the 3D viewer are colored by the average  $\Delta\Delta G$  of all mutations for each residue.

**3.2.8 mCSM-NA****Single Prediction**

- The predicted protein–nucleic acid affinity changes on a given structure are shown (Fig. 11a) with other properties such as the type of nucleic acid, solvent accessibility of wild-type protein, and predicted mutational effects from mCSM-Stability.
- For mutations further than 12 Å from the interaction, the mCSM predictions are not considered, and are set to 0, as the graph-based signatures capture a smaller radius of environmental data, and there were fewer mutations located further away than 12 Å in the datasets used to train the methods.
- The molecule visualization panel shows the protein–nucleic acid complex with the wild-type amino acid, and the mutation as a stick representation. mCSM-NA allows users to further investigate inter/intraresidue interactions by downloading Pymol session file.

**List Mutation**

- mCSN-NA provides predicted protein–nucleic acid affinity changes, wild-type RSA, and mutation information for a given list of mutations in a table which is also downloadable in TSV format.

**3.2.9 mCSM-lig**

- mCSM-lig predicts affinity changes (log affinity fold) between a protein and its ligand upon mutation (Fig. 12a) using additional information such as the closest distance between wild-type residue and ligand and the protein stability change (Kcal/mol) from DUET. The stabilizing and destabilizing mutations are shown in positive and negative values respectively.
- For mutations further than 12 Å from the interaction, the mCSM predictions are not considered, and are set to 0, as the graph-based signatures capture a smaller radius of environmental data, and there were fewer mutations located further away than 12 Å in the datasets used to train the methods.
- The wild-type amino acid and ligand are shown in stick and sphere representations in 3D molecule viewer, respectively.

**3.3 Identification of Driving Molecular Consequences**

The outputs of the predictive tools described above provide the basis for an initial heuristic examination. When trying to interpret the molecular consequences of a specific variant, it is important to remember that phenotypic outcomes are often the result of the

mCSM-NA Q Predict Data Help Contact Acknowledgements Related Resources

## *mCSM-NA: Prediction Results*

**A**

Prediction details

**Predicted Affinity Change ( $\Delta\Delta G$ ):**  
-4.516 Kcal/mol (Reduced affinity)

**Mutation:**  
Wild-type: ARG  
Position: 118  
Mutant-type: ALA  
Chain: A

**Other Properties:**  
Nucleic Acid Type: dsDNA  
Solvent accessibility: 26.9 %  
Stability effect: -1.412 Kcal/mol (Destabilising)

[Run another prediction](#)

**B**

Molecule visualization

Viewing options

Color by Chain

Main chain as Thick ribbon

Background color: White

[Apply](#) [Take screenshot](#)

Mouse options

Rotate  
 Translate  
 Zoom  
 Slab

[Reset view](#)

[Download Pymol interactions](#)

mCSM-NA Q Predict Data Help Contact Acknowledgements Related Resources

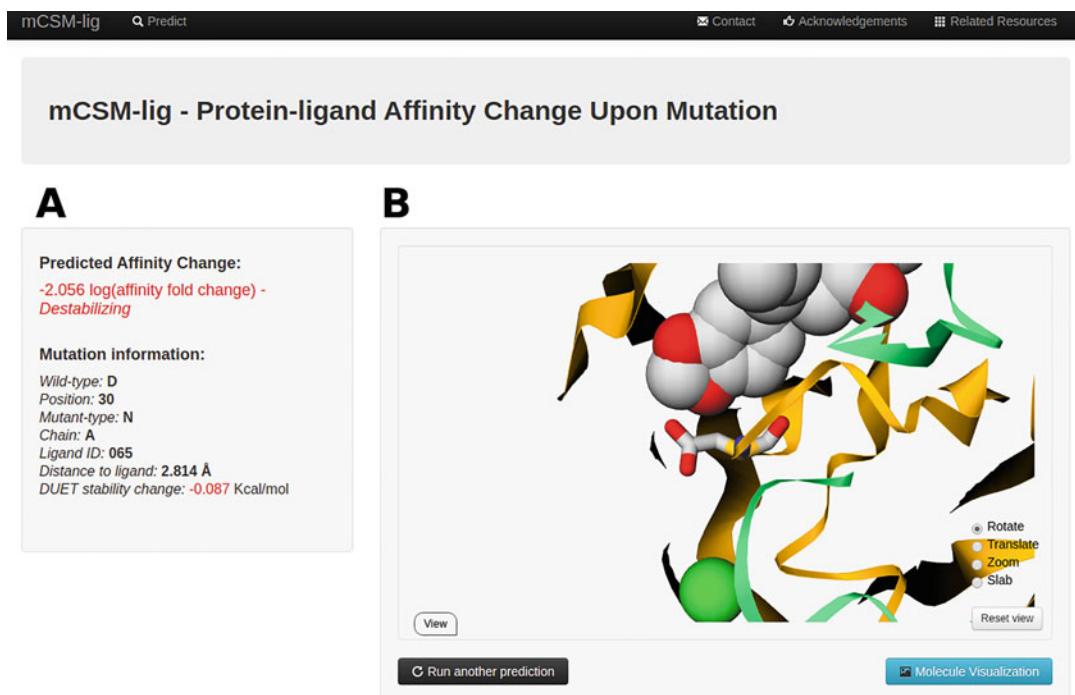
## *mCSM-NA: Prediction Results*

**C**

Index	PDB File	Chain	Wild-type Residue	Residue Position	Mutant Residue	Wild-type RSA(%)	Predicted $\Delta\Delta G$	Outcome
1	1AAV.pdb	A	H	153	K	6.3	0.924	Increased affinity
2	1AAV.pdb	A	E	177	Y	6.6	1.742	Increased affinity
3	1AAV.pdb	A	D	120	H	5.9	2.606	Increased affinity

[C Run another prediction](#) [Download results](#)

**Fig. 11** mCSM-NA result pages for single and list mutation prediction. In the single prediction result page, predicted protein–DNA affinity changes and mutation information are displayed in the prediction details (a) and the 3D viewer shows protein–DNA complex and wild-type amino acid in a ribbon and stick representation (b). The results of list prediction are shown in a tabulated form (c) and users can save the results in a TSV format



**Fig. 12** mCSM-lig result page. **(a)** The predicted affinity change between protein and ligand upon mutation is shown in logarithm scale. **(b)** The protein and ligand are displayed in 3D viewer with a ribbon (for protein), a stick (for wild-type amino acid), and a sphere (for ligand) representation

combination of multiple molecular changes. For coding mutations, we initially ask ourselves three questions:

1. Is the mutation within 5 Å of an interface? If so, is the mutation more likely to disrupt the interaction ( $\Delta\Delta G < \pm 0.5$  kcal/mol) based on the corresponding mCSM output (e.g., mCSM-PPI, mCSM-DNA, mCSM-NA, mCSM-Lig)? If the mutation is further than 12 Å away, it is less likely to disrupt the interaction directly, so the mCSM predictions are less reliable.
2. Is the mutation likely to disrupt protein folding and stability? mCSM-Stability, SDM, DUET, and DynaMut provide insight into this, with mutations leading to  $\Delta\Delta G < \pm 0.5$  kcal/mol more likely to have a significant biological effect. Mutations at buried residues are more likely to have a larger effect on protein stability.
3. Is the mutation a special case that is more likely to lead to disruption of the protein due to unique geometry restraints of the residues (*see Notes 3 and 4*)?

To more exhaustively explore how mutations in a protein lead to a phenotype, and to identify those molecular features that best

capture the driving of the molecular mechanisms, an investigation into the performance of each inputted feature should be conducted in order to construct the highest performing predictive model.

A more robust method for selecting which features are most informative can be performed using feature selection in R, a statistical programming language. While R is powerful enough itself to create classification models, we can also use it to measure which features from our predictive tools' output are most effective in stratifying mutations. Two effective approaches are:

1. A random forest classification algorithm to measure feature importance using a set of mutations with known class labels (e.g., pathogenic/nonpathogenic, deleterious/nondeleterious).
2. The Boruta Algorithm performs permutations of the data to statistically compare each feature's importance with that attainable at random, and uses this to eliminate uninformative features. The package in R provides a graphical output using boxplots.

Features that score highly provide evidence that the molecular consequence that they measure is relevant to how mutations lead to the phenotype of interest. The algorithm can also highlight correlation between features. When two or more features are highly correlated and are likely measuring the same information, only one should be used in subsequent predictive model development to remove redundancy, minimize noise and avoid bias from weighting a model in favor of a particular attribute. The model should also have the fewest possible features that perform best. Using too many features may generate a model that performs accurately on training data but cannot be generalized to real-world data.

### **3.4 Machine Learning Phenotypes: Building a Predictive Classifier**

An initial understanding of molecular mechanisms imparted by disease-causing mutations is a crucial step toward establishing a genotype–phenotype correlation. However, manual analysis of different results can often miss underlying, statistically significant relationships among different mutational measurements, which can help relate them to the phenotype. Machine learning, and in particular supervised learning, addresses this issue by providing a set of tools for the efficient analysis of labeled data (e.g., experimentally characterized mutations) in order to derive a model that describes a phenomenon, aiming for generalization (applying it to unseen data). The identification of patterns and associations within the data will further help the predictive model establish a distinction between mutations within the same gene leading to different phenotypes, and hence the development of an effective predictive tool that can be used to interpret novel clinical variants.

Here, our goal is to build a machine learning classifier to distinguish between pathogenic vs. nonpathogenic mutations in a

given gene. Multiple steps are required to obtain a nonbiased, accurate predictor:

1. Dataset curation: Machine learning algorithms require a well-curated dataset. In a supervised machine learning approach, all data labels (here, pathogenic or nonpathogenic for each mutation) must be known in order to enable correlations to be assessed between labels (e.g., phenotypes) and features/properties used as evidence to represent each data point (e.g., mutations). The quality of a classifier directly depends on the quality of the data used to build it, so accurate clinical sources are required to justify labeling mutations as pathogenic or nonpathogenic. In this case, generally, nonpathogenic variants can be curated from population variant databases such as GnomAD, usually taking into account frequent mutations. Even common variants, however, may still be linked to a disease, especially if it is a weakly penetrative mutation or recessive condition, which would add noise to the data set and thus complicate the task of building a general predictive model. In situations where other biologically relevant information is present, such as cellular fitness cost, it is essential that this type of information is present for every mutation in a dataset, as a supervised algorithm cannot handle missing data labels. The initial dataset should contain a representative set of mutations within all phenotype classes (pathogenic and nonpathogenic), and ideally, present a balanced number of instances between classes, to minimize biases toward overrepresented classes in the resultant model. More details on metrics used to evaluate the performance of predictive models on an imbalanced dataset are discussed below.
2. Feature generation: The feature generation stage is crucial as it provides descriptive information about each mutation, to be used by the learning algorithm to finally classify the phenotype of a mutation. As described above, features can encompass a diverse range of mutational information:
  - (a) Protein stability and dynamics (mCSM-Stability, DUET, SDM, Dynamut).
  - (b) Protein functional changes such as changes in affinity for other proteins (mCSM-PPI2), nucleic acids (mCSM-NA), and ligands (mCSM-lig).
  - (c) At the residue level, changes in protein pharmacophore and local residue environment such as changes in interatomic interactions (Arpeggio) are also important, as some mutations at the same locus can have different phenotypes.
  - (d) Sequence-level predictors (SIFT, Polyphen, SNAP2).

- (e) Evolutionary-based predictors (ConSurf), population based mutational tolerance (MTR-Viewer), as well as amino acid substitution matrices (e.g., PAM30, BLOSUM62, PSSM) offer added information on the likelihood of one mutation to change into another.

Feature generation is directly dependent on the wild-type biological functions of the protein, which is why an understanding of the biological relevance is important at the very beginning of this process.

- 3. Training and Testing sets: The data collected must be divided into training and testing sets to assess the generalization power of a classifier, that is, its ability to correctly predict on new data, and to ensure that it has not been over- or undertrained. Data used to train the model should be different, nonredundant, from the data used to test the model. It is common practice to divide the original dataset into Training and Test sets at the start of learning. For small datasets, a large proportion of the data may need to be segregated into the Test set to provide sufficient data to accurately measure performance of the trained model. This can be done in a bootstrapping procedure or through cross-validation, when the original data set is divided into  $k$ -folds and each is taken iteratively as the test set while remaining data are used in training ( $k$ -fold cross-validation).
- 4. Feature selection: The features selected for training can strongly influence accuracy, so it is important to select only informative features, and eliminate irrelevant or nondiscriminative ones, which are a common source of noise. Feature selection can also help reduce overfitting and reduce training time, as it aims to generate simpler, more concise models. Feature selection methods provided in the Python machine learning library, Scikit-Learn [58], include univariate selection, feature importance, correlation matrix, and recursive feature elimination or addition. Alternatively, forward stepwise selection can be performed as a greedy heuristic in which features are included iteratively, one at a time, based on their individual performance contributions.
- 5. Machine learning platforms: Different tools have been developed for implementing machine learning. Some offer a graphical user interface (GUI), such as Weka [59], while some run as python packages through the command line, such as Scikit-Learn. Different packages for different programming languages offer similar algorithms and options to adjust the algorithm parameters according to specific tasks. The major classification algorithms we test are Naive Bayes, Decision Trees, K-Nearest Neighbor, Support Vector Machines, and Ensemble Classifiers. It is good practice to compare

representative algorithms of each class, provided that the algorithm is compatible with the dataset type. Within weka, this can be done automatically using the auto-weka function. In cases where the training set is unbalanced, oversampling or undersampling of the training data can be used to achieve a better representation of classes within the classification model-building stage, preventing model bias in always detecting the predominant class and achieving a false high performance.

6. Model validation: The primary tool in the validation of a model is the use of a nonredundant independent test set, also called blind test.

Validation can be furthered using internal data testing such as  $k$ -fold cross validation, in which the dataset is divided into  $k$  subsets. One subset is used as a test set, while the remaining  $(k - 1)$  subsets are used to train a model. The process is repeated  $k$  times, until all the data have been used in both training and test sets. The final model performance is calculated as the average of the performances of all  $k$  iterations. We will often vary  $k$  based on the size of the dataset. When the training set is small (e.g., ~200 data points), we may use leave-one-out validation, where  $k$  is equal to the size of the dataset. An important aspect when selecting predictive models is consistency in performance between the training and test sets. This usually indicates a robust model, within which discrepancies (e.g., a much higher performance on training than with the test set) might indicate overfitting.

7. Model evaluation: Several different evaluation metrics may be used for classification tasks. These are generally calculated on values obtained from a confusion matrix, which is a summary of the data points, and their actual and predicted phenotypes (Table 6).
8. From the distributions of data points within the matrix, descriptive metrics can be calculated:
  - (a) accuracy (number of correct predictions:  $[(TP + TN)/TOTAL]$ ),
  - (b) precision (rate of correctly predicted positive instances from all assigned as positives:  $[TP/(TP + FP)]$ ),

**Table 6**  
**Description of a confusion matrix**

Predicted value	Actual value	
	Positive	Negative
Positive	True positive	False positive
Negative	False negative	True negative

- (c) recall (rate of correctly predicted positive instances from all real positive instances:  $[TP/(TP + FN)]$ ),
- (d) *f*-score (a weighted average of recall and precision), and,
- (e) Matthews correlation coefficient (MCC) a balanced measure between true positives and true negatives

$$[(TP \times TN) - (FP \times FN)] / \sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}$$

where TP = True positive; TN = True negative; FP = False positive; and FN = False negative.

Classifier performance can also be described graphically using a Receiver Operating Characteristic curve, which compares the TP Rate and TN Rate. The closer the area under the curve is to 1, the better the classifier performance.

These metrics should be used in a combinatorial fashion across all elements of training, test, and cross-validation stages to compare model performance during different stages of classifier optimization. When the dataset is imbalanced, balanced measures such as MCC should be prioritized, as other measures might bias for an overtrained model on the dominant dataset.

## 4 Notes

1. Often following curation, the distribution of number of pathogenic and benign mutations is unbalanced, which can affect efforts to build predictive tools using machine learning. Two approaches that can help include oversampling of the underrepresented class, or undersampling of the overrepresented class. Evaluation metrics that are less biased toward unbalanced classes, such as the Matthew's correlation coefficient, precision-recall curves, and Kendall correlations, should also be preferentially used.
2. The chain ID for the provided PDB file is a mandatory field for all the structure-based methods; blank characters are not allowed. It is possible that homology modeling tools might not automatically add a chain ID. If this is the case, the user will need to modify the PDB file prior to submission to the servers. Several tools exist to perform this task (e.g., <http://www.canoz.com/sdh/renam pdbchain.pl>).
3. Special cases: Mutations to and from prolines. Prolines are the only amino acid whose amino group is connected to the side-chain, which in the context of the peptide bond greatly limits torsional angles. The nature of this residue therefore needs to be taken into account while analyzing mutation effects. For instance, (1) mutations to prolines in the middle of alpha-

helices can introduce kinks, affecting local structure and (2) since prolines are commonly found in turns and loops, their substitution might interfere with the formation of secondary structures such as hairpins.

4. Special cases: mutations of positive-phi glycines. Similarly to prolines, positive phi glycines, while rare in experimental structures, deserve special consideration due to their torsional angles. Glycines are the only residues capable of adopting positive-phi angles. These glycines are usually conserved across evolution, meaning that mutations on positive-phi glycines, especially on loops and hairpins, tend to be destabilizing.

## Acknowledgments

This work was supported by Australian Government Research Training Program Scholarships [to S.P., M.K., Y.M., C.H.M.R.]; the Jack Brockhoff Foundation [JBF 4186, 2016 to D.B.A.]; a Newton Fund RCUK-CONFAP Grant awarded by The Medical Research Council (MRC) and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) [MR/M026302/1 to D.B.A. and D.E.V.P.]; and the National Health and Medical Research Council of Australia [APP1072476 to D.B.A.].

## References

1. Jatana N, Ascher DB, Pires DEV et al (2019) Human LC3 and GABARAP subfamily members achieve functional specificity via specific structural modulations. *Autophagy*:1–17. <https://doi.org/10.1080/15548627.2019.1606636>
2. Abayakoon P, Jin Y, Lingford JP et al (2018) Structural and biochemical insights into the function and evolution of sulfoquinovosidases. *ACS Cent Sci* 4(9):1266–1273. <https://doi.org/10.1021/acscentsci.8b00453>
3. Ascher DB, Cromer BA, Morton CJ et al (2011) Regulation of insulin-regulated membrane aminopeptidase activity by its C-terminal domain. *Biochemistry* 50(13):2611–2622. <https://doi.org/10.1021/bi101893w>
4. Portelli S, Phelan JE, Ascher DB et al (2018) Understanding molecular consequences of putative drug resistant mutations in *Mycobacterium tuberculosis*. *Sci Rep* 8(1):15356. <https://doi.org/10.1038/s41598-018-33370-6>
5. Silk M, Petrovski S, Ascher DB (2019) MTR-Viewer: identifying regions within genes under purifying selection. *Nucleic Acids Res* 47(W1):W121–W126. <https://doi.org/10.1093/nar/gkz457>
6. Pires DE, Blundell TL, Ascher DB (2015) Platinum: a database of experimentally measured effects of mutations on structurally defined protein-ligand complexes. *Nucleic Acids Res* 43(Database issue):D387–D391. <https://doi.org/10.1093/nar/gku966>
7. Lucy G, Douglas EVP, Álvaro O-N et al (2014) An integrated computational approach can classify VHL missense mutations according to risk of clear cell renal carcinoma. *Human Molecular Genetics*, 23(22):5976–5988. <https://doi.org/10.1093/hmg/ddu321>
8. Blaszczyk M, Harmer NJ, Chirgadze DY et al (2015) Achieving high signal-to-noise in cell regulatory systems: spatial organization of multiprotein transmembrane assemblies of FGFR and MET receptors. *Prog Biophys Mol Biol* 118(3):103–111. <https://doi.org/10.1016/j.pbiomolbio.2015.04.007>
9. Jafri M, Wake NC, Ascher DB et al (2015) Germline mutations in the CDKN2B tumor suppressor gene predispose to renal cell carcinoma. *Cancer Discov* 5(7):723–729. <https://doi.org/10.1158/2159-8290.CD-14-1096>

10. Pacitto A, Ascher DB, Wong LH et al (2015) Lst4, the yeast Fnip1/2 orthologue, is a DENN-family protein. *Open Biol* 5(12):150174. <https://doi.org/10.1098/rsob.150174>
11. Pires DE, Chen J, Blundell TL et al (2016) In silico functional dissection of saturation mutagenesis: interpreting the relationship between phenotypes and changes in protein stability, interactions and activity. *Sci Rep* 6:19848. <https://doi.org/10.1038/srep19848>
12. Albanaz ATS, Rodrigues CHM, Pires DEV et al (2017) Combating mutations in genetic disease and drug resistance: understanding molecular mechanisms to guide drug design. *Expert Opin Drug Discov* 12(6):553–563. <https://doi.org/10.1080/17460441.2017.1322579>
13. Casey RT, Ascher DB, Rattenberry E et al (2017) SDHA related tumorigenesis: a new case series and literature review for variant interpretation and pathogenicity. *Mol Genet Genomic Med* 5(3):237–250. <https://doi.org/10.1002/mgg3.279>
14. Jubb HC, Pandurangan AP, Turner MA et al (2017) Mutations at protein-protein interfaces: small changes over big surfaces have large impacts on human health. *Prog Biophys Mol Biol* 128:3–13. <https://doi.org/10.1016/j.pbiomolbio.2016.10.002>
15. Pandurangan AP, Ascher DB, Thomas SE et al (2017) Genomes, structural biology and drug discovery: combating the impacts of mutations in genetic disease and antibiotic resistance. *Biochem Soc Trans* 45(2):303–311. <https://doi.org/10.1042/BST20160422>
16. Sibanda BL, Chirgadze DY, Ascher DB et al (2017) DNA-PKcs structure suggests an allosteric mechanism modulating DNA double-strand break repair. *Science* 355(6324):520–524. <https://doi.org/10.1126/science.aak9654>
17. Rodrigues CH, Ascher DB, Pires DE (2018) Kinact: a computational approach for predicting activating missense mutations in protein kinases. *Nucleic Acids Res* 46(W1):W127–W132. <https://doi.org/10.1093/nar/gky375>
18. Hnizda A, Fabry M, Moriyama T et al (2018) Relapsed acute lymphoblastic leukemia-specific mutations in NT5C2 cluster into hotspots driving intersubunit stimulation. *Leukemia* 32(6):1393–1403. <https://doi.org/10.1038/s41375-018-0073-5>
19. Andrews KA, Ascher DB, Pires DEV et al (2018) Tumour risks and genotype-phenotype correlations associated with germline variants in succinate dehydrogenase subunit genes SDHB, SDHC and SDHD. *J Med Genet* 55(6):384–394. <https://doi.org/10.1136/jmedgenet-2017-105127>
20. Usher JL, Ascher DB, Pires DE et al (2015) Analysis of HGD gene mutations in patients with alkaptonuria from the United Kingdom: identification of novel mutations. *JIMD Rep* 24:3–11. [https://doi.org/10.1007/8904\\_2014\\_380](https://doi.org/10.1007/8904_2014_380)
21. Nemethova M, Radvanszky J, Kadasi L et al (2016) Twelve novel HGD gene variants identified in 99 alkaptonuria patients: focus on ‘black bone disease’ in Italy. *Eur J Hum Genet* 24(1):66–72. <https://doi.org/10.1038/ejhg.2015.60>
22. Ramdzan YM, Trubetskoy MM, Ormsby AR et al (2017) Huntingtin inclusions trigger cellular quiescence, deactivate apoptosis, and lead to delayed necrosis. *Cell Rep* 19(5):919–927. <https://doi.org/10.1016/j.celrep.2017.04.029>
23. Traynelis J, Silk M, Wang Q et al (2017) Optimizing genomic medicine in epilepsy through a gene-customized approach to missense variant interpretation. *Genome Res* 27(10):1715–1729. <https://doi.org/10.1101/gr.226589.117>
24. Trezza A, Bernini A, Langella A et al (2017) A computational approach from gene to structure analysis of the human ABCA4 transporter involved in genetic retinal diseases. *Invest Ophthalmol Vis Sci* 58(12):5320–5328. <https://doi.org/10.1167/iovs.17-22158>
25. Ascher DB, Spiga O, Sekelska M et al (2019) Homogentisate 1,2-dioxygenase (HGD) gene variants, their analysis and genotype-phenotype correlations in the largest cohort of patients with AKU. *Eur J Hum Genet* 27(6):888–902. <https://doi.org/10.1038/s41431-019-0354-0>
26. Soardi FC, Machado-Silva A, Linhares ND et al (2017) Familial STAG2 germline mutation defines a new human cohesinopathy. *NPJ Genom Med* 2:7. <https://doi.org/10.1038/s41525-017-0009-4>
27. Phelan J, Coll F, McNerney R et al (2016) Mycobacterium tuberculosis whole genome sequencing and protein structure modelling provides insights into anti-tuberculosis drug resistance. *BMC Med* 14:31. <https://doi.org/10.1186/s12916-016-0575-9>
28. Silvino AC, Costa GL, Araujo FC et al (2016) Variation in human cytochrome P-450 drug-metabolism genes: a gateway to the understanding of Plasmodium vivax relapses. *PLoS One* 11(7):e0160172. <https://doi.org/10.1371/journal.pone.0160172>

29. White RR, Ponsford AH, Weekes MP et al (2016) Ubiquitin-dependent modification of skeletal muscle by the parasitic nematode, *Trichinella spiralis*. *PLoS Pathog* 12(11):e1005977. <https://doi.org/10.1371/journal.ppat.1005977>
30. Hawkey J, Ascher DB, Judd LM et al (2018) Evolution of carbapenem resistance in *Acinetobacter baumannii* during a prolonged infection. *Microb Genom* 4(3). <https://doi.org/10.1099/mgen.0.000165>
31. Holt KE, McAdam P, Thai PVK et al (2018) Frequent transmission of the *Mycobacterium tuberculosis* Beijing lineage and positive selection for the EsxW Beijing variant in Vietnam. *Nat Genet* 50(6):849–856. <https://doi.org/10.1038/s41588-018-0117-9>
32. Karmakar M, Globan M, Fyfe JAM et al (2018) Analysis of a Novel pncA mutation for susceptibility to pyrazinamide therapy. *Am J Respir Crit Care Med* 198(4):541–544. <https://doi.org/10.1164/rccm.201712-2572LE>
33. Vediithi SC, Malhotra S, Das M et al (2018) Structural implications of mutations conferring rifampin resistance in *Mycobacterium leprae*. *Sci Rep* 8(1):5016. <https://doi.org/10.1038/s41598-018-23423-1>
34. Karmakar M, Rodrigues CHM, Holt KE et al (2019) Empirical ways to identify novel Bedaquiline resistance mutations in AtpE. *PLoS One* 14(5):e0217169. <https://doi.org/10.1371/journal.pone.0217169>
35. Ascher DB, Wielens J, Nero TL et al (2014) Potent hepatitis C inhibitors bind directly to NS5A and reduce its affinity for RNA. *Sci Rep* 4:4765. <https://doi.org/10.1038/srep04765>
36. Jubb HC, Higueruelo AP, Ochoa-Montano B et al (2017) Arpeggio: a web server for calculating and visualising interatomic interactions in protein structures. *J Mol Biol* 429 (3):365–371. <https://doi.org/10.1016/j.jmb.2016.12.004>
37. Pires DE, Ascher DB, Blundell TL (2014) mCSM: predicting the effects of mutations in proteins using graph-based signatures. *Bioinformatics* 30(3):335–342. <https://doi.org/10.1093/bioinformatics/btt691>
38. Pandurangan AP, Ochoa-Montano B, Ascher DB et al (2017) SDM: a server for predicting effects of mutations on protein stability. *Nucleic Acids Res* 45(W1):W229–W235. <https://doi.org/10.1093/nar/gkx439>
39. Pires DE, Ascher DB, Blundell TL (2014) DUET: a server for predicting effects of mutations on protein stability using an integrated computational approach. *Nucleic Acids Res* 42(Web Server issue):W314–W319. <https://doi.org/10.1093/nar/gku411>
40. Douglas EVP, Carlos HMR, David BA et al (2020) mCSM-membrane: predicting the effects of mutations on transmembrane proteins. *Nucleic Acids Research*, gkaa416. <https://doi.org/10.1093/nar/gkaa416>
41. Rodrigues CH, Pires DE, Ascher DB (2018) DynaMut: predicting the impact of mutations on protein conformation, flexibility and stability. *Nucleic Acids Res* 46(W1):W350–W355. <https://doi.org/10.1093/nar/gky300>
42. Rodrigues CHM, Myung Y, Pires DEV et al (2019) mCSM-PPI2: predicting the effects of mutations on protein-protein interactions. *Nucleic Acids Res* 47(W1):W338–W344. <https://doi.org/10.1093/nar/gkz383>
43. Pires DE, Ascher DB (2016) mCSM-AB: a web server for predicting antibody-antigen affinity changes upon mutation with graph-based signatures. *Nucleic Acids Res* 44(W1):W469–W473. <https://doi.org/10.1093/nar/gkw458>
44. Yoochan M, Carlos HMR, David BA, Douglas EVP et al (2020) mCSM-AB2: guiding rational antibody design using graphbased signatures, *Bioinformatics*. 36(5):1453–1459. <https://doi.org/10.1093/bioinformatics/btz779>
45. Yoochan M, Douglas EVP, David BA et al. mmCSM-AB: guiding rational antibody engineering through multiple point mutations, *Nucleic Acids Research*, gkaa389. <https://doi.org/10.1093/nar/gkaa389>
46. Pires DEV, Ascher DB (2017) mCSM-NA: predicting the effects of mutations on protein-nucleic acids interactions. *Nucleic Acids Res* 45 (W1):W241–W246. <https://doi.org/10.1093/nar/gkx236>
47. Pires DE, Blundell TL, Ascher DB (2016) mCSM-lig: quantifying the effects of mutations on protein-small molecule affinity in genetic disease and emergence of drug resistance. *Sci Rep* 6:29575. <https://doi.org/10.1038/srep29575>
48. Pires DE, Ascher DB (2016) CSM-lig: a web server for assessing and comparing protein-small molecule affinities. *Nucleic Acids Res* 44 (W1):W557–W561. <https://doi.org/10.1093/nar/gkw390>
49. Douglas EVP et al (2011) Cutoff Scanning Matrix (CSM): structural classification and function prediction by protein inter-residue distance patterns. *BMC genomics* (12) No. S4. BioMed Central
50. Douglas EVP, Raquel CM-M, Carlos HS, Frederico FC, Wagner M Jr (2013) aCSM: noise-free graphbased signatures to large-scale receptor-based ligand prediction, *Bioinformatics* 29 (7):855–861. <https://doi.org/10.1093/bioinformatics/btt058>

51. Sherry ST, Ward MH, Kholodov M et al (2001) dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res* 29(1):308–311. <https://doi.org/10.1093/nar/29.1.308>
52. Stenson PD, Mort M, Ball EV et al (2017) The Human Gene Mutation Database: towards a comprehensive repository of inherited mutation data for medical research, genetic diagnosis and next-generation sequencing studies. *Hum Genet* 136(6):665–677. <https://doi.org/10.1007/s00439-017-1779-6>
53. Landrum MJ, Lee JM, Benson M et al (2018) ClinVar: improving access to variant interpretations and supporting evidence. *Nucleic Acids Res* 46(D1):D1062–D1067. <https://doi.org/10.1093/nar/gkx1153>
54. Karczewski KJ, Francioli LC, Tiao G et al (2019) Variation across 141,456 human exomes and genomes reveals the spectrum of loss-of-function intolerance across human protein-coding genes. *bioRxiv*:531210. <https://doi.org/10.1101/531210>
55. Sudlow C, Gallacher J, Allen N et al (2015) UK biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS Med* 12(3): e1001779. <https://doi.org/10.1371/journal.pmed.1001779>
56. UniProt Consortium T (2018) UniProt: the universal protein knowledgebase. *Nucleic Acids Res* 46(5):2699. <https://doi.org/10.1093/nar/gky092>
57. Rose PW, Prlic A, Altunkaya A et al (2017) The RCSB protein data bank: integrative view of protein, gene and 3D structural information. *Nucleic Acids Res* 45(D1):D271–D281. <https://doi.org/10.1093/nar/gkw1000>
58. Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in python. *J Mach Learn Res* 12:2825–2830
59. Witten IH, Frank E, Hall MA et al (2016) Data mining, fourth edition: practical machine learning tools and techniques. Morgan Kaufmann, Burlington



# Chapter 2

## Machine Learning for Biomedical Time Series Classification: From Shapelets to Deep Learning

Christian Bock, Michael Moor, Catherine R. Jutzeler,  
and Karsten Borgwardt

### Abstract

With the biomedical field generating large quantities of time series data, there has been a growing interest in developing and refining machine learning methods that allow its mining and exploitation. Classification is one of the most important and challenging machine learning tasks related to time series. Many biomedical phenomena, such as the brain's activity or blood pressure, change over time. The objective of this chapter is to provide a gentle introduction to time series classification. In the first part we describe the characteristics of time series data and challenges in its analysis. The second part provides an overview of common machine learning methods used for time series classification. A real-world use case, the early recognition of sepsis, demonstrates the applicability of the methods discussed.

**Key words** Time series, Classification, Onset detection, Subsequence mining, Deep learning

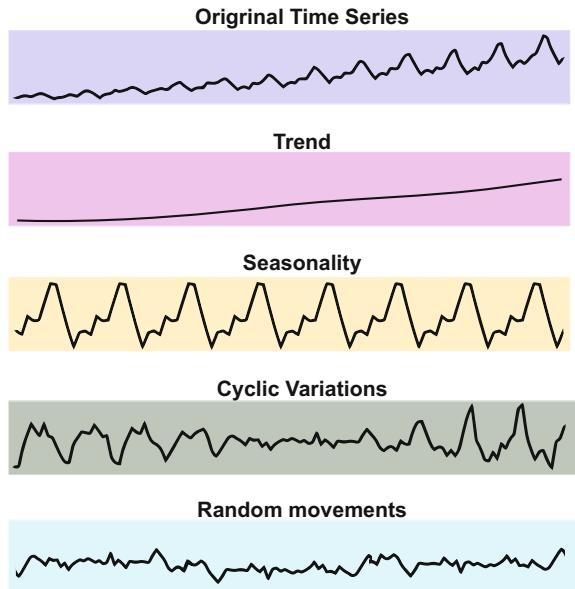
---

### 1 Introduction

Over the last decade, there has been an unparalleled growth of time series data accessible through biomedical databases (e.g., MIMIC, UKBiobank), smartphone apps (e.g., step counter, exercise tracker, health information), and biosensor technology (e.g., heart rate dynamics monitor, sleep monitor) [1–3]. This availability of massive volumes and different sources of time series data presents unprecedented opportunities for time series mining. As the name indicates, time series are defined as sequences of observations on a variable of interest indexed in time. Biomedical time series are often large in size, high-dimensional, and gathered sequentially and iteratively. As a result of the repeated data collection, time series data typically exhibit different kinds of systematic patterns. The most

---

Christian Bock and Michael Moor contributed equally to this work.



**Fig. 1** Time series and its components. Four major patterns can characterize a time series: trend, seasonality, and random or irregular movements due to short-term, unsystematic noise or error

common are trends, seasonalities, cyclic variations, as well as randomness and noise (as shown in Fig. 1):

- *Secular or general trends* are the smooth, general, long-term, mean tendency of time series, unrelated to calendar, cyclic, or irregular effects. A trend is considered positive, negative, or stationary, depending on whether the time series is characterized by an increasing, decreasing, or stable long-term pattern, respectively. An example of a trend would be a long-term decrease in children's mortality as a consequence of influenza vaccinations [4].
- *Seasonal fluctuations or seasonality* are systematic and calendar-related fluctuations that occur during the same month(s) or quarter of every year. For instance, seasonal fluctuations can be observed in longitudinal data on body mass and composition (e.g., increased amount of brown adipose fat in the winter compared to the summer [5]).
- Cyclic variations describe any repeating patterns that are not related to the calendar. The duration of a cycle depends on the type of biomedical data source being analyzed. Examples of cyclic fluctuations are the female hormone levels within the menstrual cycle or the regulation of the daily body temperature [6, 7].

- *Random or irregular movements* are a hallmark of every time series. They occur unsystematically and are purely random and irregular. These fluctuations are unforeseen, uncontrollable, unpredictable, and erratic. Sampling and measurement errors are typical examples of irregular movements that cause variation in the data (e.g., patient movement in ECG recordings [8]).

In comparison to other (cross-sectional) data types, time series data is characterized by a temporal component. This component adds important information to the input variables and their changes over time. Time series have a natural temporal ordering and each time unit can only capture at most one data measurement. Modifying the order of a time series will likely change the meaning of the data. The temporal ordering makes time series data unique in the data space, as the data points within a time series typically display serial dependence: a data point at some time in a time series is statistically dependent on preceding data points of the same time series. Closely linked to the notion of serial dependence is autocorrelation, which refers to the degree of similarity between the values of the same variable over successive time intervals (i.e., internal correlation within a time series). For instance, one would expect the blood sugar concentration at 8 am to be more similar to the concentration at 9 am than to the concentration at 12 pm. If blood sugar concentration values that are temporally closer are indeed more similar than those farther apart, then the data is deemed autocorrelated. There are numerous methods to determine the presence and degree of autocorrelation, including the Durbin–Watson test [9], but these are beyond the scope of this chapter. For further details, the interested reader is referred to [9–11]. Both serial dependence and autocorrelation greatly limit the applicability of many conventional statistical methods, as the assumptions of statistical independence and identical distribution of the data are violated. Thus, tailored statistical and machine learning (ML) methods have been developed to extract meaningful statistics and other characteristics of time series while taking account of the serial dependence and autocorrelation.

Popular data mining tasks related to temporal data [12] include (1) forecasting time series based on historical data, (2) recognizing time series patterns [13], (3) detecting anomalies, transients, or onsets of biomedical signals (e.g., onset of myocardial infarction) [14], and (4) classifying time series or subsequences based on their behavior over time [15–17]. The focus of this chapter is on the classification of time series. Before delving into methodology, we will first discuss some major challenges of biomedical time series data, define the notations that are used throughout the chapter, and introduce our biomedical running example. The goal of the latter is to strengthen understanding of the methods discussed and their applicability to real-world biomedical data.

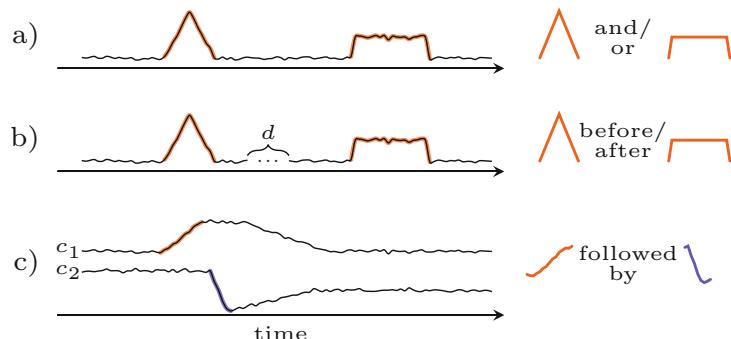
## 1.1 Challenges of Biomedical Time Series Data Analysis

### 1.1.1 Time Series Signals

Compared to data sets in which each sample can be described by an  $n$ -dimensional set of uncorrelated static features (e.g., cell size, body mass index, ethnicity), only two features are offered by a univariate time series: The point in time of the signal and the observed signal itself. This exacerbates the analysis of time series in multiple ways: Firstly, each observation must always be considered as tuple of observation time and observation value; secondly, a single isolated observation  $x$  at time point  $i$  is meaningless without an understanding of the underlying dynamical process and the relation to preceding and subsequent observations (serial dependence). In contrast, observing the mass of a sample is self-explanatory and—ideally—*independent* from other features of the data set.

An intuitive way of deriving interpretable and actionable features from univariate and multivariate time series is to think of them in terms of sets of subsequences (*see* Definition 3 in Subheading 1.2). Subsequences and their usage in a variety of time series analysis tasks have been studied intensively for the last decade [17–19] and—due to their interpretability—proved to be an invaluable tool. Figure 2 illustrates some common subsequence schemes that might explain class membership.

Each scheme might be interesting in several time series tasks. The logical concatenation of subsequences (a), the temporal ordering of subsequences (b), as well as the interaction of different channels (c) are often characteristic for a certain subpopulation of



**Fig. 2** Schematic illustration of different types of subsequences. On the left side, noisy time series are shown. On the right side, the possible underlying patterns that are indicative for class membership are plotted. (a) shows the case in which a logical combination of specific subsequences is important without taking their temporal order into account. (b) illustrates subsequences with temporal dependence over a certain time horizon  $d$ . (c) exemplifies how subsequences can interact between different time series channels (here  $c_1$  and  $c_2$ ). In this case, an increase in channel 1 followed by a decrease in channel 2 is indicative for class membership

interest (e.g., classification or onset detection). The challenge is to find respective patterns with no or limited a priori knowledge about their shape, position, or class affiliation.

### 1.1.2 Time Series Labels

Supervised ML in various domains remains limited by one key factor: the lack of *annotated* data. One major cause for this issue is that in many applications, annotating data instances requires the costly efforts of domain experts. For example, if we were to build a classifier to detect lung cancer nodules in radiographic images, to enable a supervised learning setting we first need a large image dataset for which cancerous lesions are manually annotated by domain experts such as radiologists. This bottleneck has spawned alternative routes to perform ML *without* labels, which is known as unsupervised learning (for an overview, refer to ref. 20).

How does this issue translate to the classification of time series? In recent years, big databases of labeled time series have been made accessible, such as the UCR time series archive (currently listing 128 datasets) [21] or the MIMIC database [2], providing electronic health records including time series of clinical and laboratory measurements and diagnosis codes. However, time series datasets in which labels are equipped with *time-stamps* (referring to particular events that should, for instance, be predicted with the preceding time series) are still rare. Especially in the biomedical domain, time-annotated labels are often necessary for rendering the classification task meaningful and unspoiled. If we were to predict whether an in-hospital patient will develop a stroke during her stay, it would be trivial to predict this endpoint using interventional data *after* the stroke diagnosis was made (e.g., whether the patient was transferred to a stroke unit). To make this classification task more meaningful, we need to be able to pinpoint the stroke event to a time-stamp, such that input data *preceding* this event can be used for prediction. Time-stamped labels can be challenging to derive, as for many studied phenotypes, no explicit rules for defining exact event time windows are available, rendering manual annotation by domain experts indispensable.

### 1.1.3 Irregular Sampling

Most algorithms analyzing time series require them to be sampled at evenly spaced time intervals. However, real-world time series—especially in the biomedical domain—are often sampled at *irregular* time intervals. This mismatch is a crucial challenge of many time series tasks which frequently has to be addressed before further analysis is possible. There are several types of irregularities at hand: A time series could be sampled evenly spaced, but there are *missing* values. Alternatively, a time series is fully observed (no missing values), but at *irregular* time intervals. Furthermore, a time series could be both irregularly spaced and show missing values. Besides addressing irregular and incomplete (with missing values) time series with imputation techniques, methods have been

proposed that can process irregular time series directly [22]. In this chapter, we will briefly touch on irregular time series and imputation strategies when discussing real-world applications.

## 1.2 Notation

We now introduce some mathematical definitions that help us formalize different time series tasks and the methods to be presented.<sup>1</sup>

**Definition 1 (Univariate Time Series):** A univariate time series  $T_U$  of length  $m > 0$  can be denoted as a sequence of tuples consisting of observation time point  $t_i$  and respective observation value  $v_i$ :

$$T_U = \{(t_0, v_0), \dots, (t_{m-1}, v_{m-1})\}$$

**Definition 2 (Multivariate Time Series):** Given a set  $C = \{c_0, \dots, c_{p-1}\}$  containing  $p$  channels of lengths  $m_0, \dots, m_{p-1}$ , we write as follows:

$$T_M = \{(t_0, v_0, c_0), \dots, (t_{m_0-1}, v_{m_0-1}, c_0), \dots, (t_0, v_0, c_{p-1}), \dots, (t_{m_{p-1}-1}, v_{m_{p-1}-1}, c_{p-1})\}$$

Each observation of a channel is defined through a triplet of time point  $t$ , observation value  $v$ , and channel index  $c$ .

**Definition 3 (Subsequence):** A subsequence  $S$  of length  $w$  at time point  $l$  from a univariate time series  $T$  is the sequence of its  $w$  consecutive observation values:

$$S = (s_l, \dots, s_{l+w-1})$$

**Definition 4 (Time Series Data Set):** A data set  $D$  of  $n - 1$  univariate variable-length time series is defined as follows:

$$\begin{aligned} D &= \{\{(t_0, v_0), \dots, (t_{m_0-1}, v_{m_0-1})\}^{[0]}, \dots, \{(t_0, v_0), \dots, (t_{m_n-1}, v_{m_n-1})\}^{[n]}\} \\ &= \{T_U^{[0]}, \dots, T_U^{[n]}\}, \end{aligned}$$

where  $m_i$  is the time series length of the  $i$ th sample.

**Definition 5 (Labeled Time Series Data Set):** A labeled time series data set is a time series data set with the extension that the  $i$ th sample  $T^{[i]}$  is associated with a class label  $y_i \in \mathcal{Y}$ :

$$D_L = \{(T^{[0]}, y_0), \dots, (T^{[n]}, y_n)\}$$

We differentiate between binary class labels, where  $\mathcal{Y} = \{0, 1\}$ , and multiclass labels, where  $\mathcal{Y} = \{0, \dots, k\}$  with  $k \in \mathbb{N}$ .

---

<sup>1</sup> While we try to be consistent in our usage of notation, it might be necessary to slightly diverge from initially introduced notation to keep equations more readable. If this is the case, we will clarify this in the text.

**Definition 6 (Metric):** A distance function  $d(\cdot, \cdot)$  between two objects  $a \in X$ , and  $b \in X$  is a map  $d: X \times X \rightarrow [0, \infty)$  that measures their dissimilarity. If the distance is close to 0 the objects are considered similar. Large values indicate dissimilar objects. A distance is called metric if and only if all of the following conditions are met:

1.  $d(a, b) \geq 0$  (nonnegativity)
2.  $d(a, b) = 0$  if and only if  $a = b$  (identity of indiscernibles)
3.  $d(a, b) = d(b, a)$  (symmetry)
4. With  $c \in X$ :  $d(a, c) \leq d(a, b) + d(b, c)$  (triangle inequality)

### 1.3 Our Running Example: Sepsis Prediction

Our chapter covers recent time series classification methods from a rather technical perspective. In order to make the presented concepts intuitively more accessible, we spark them into life by applying them to a real-world running example at every section ending. To conclude the introductory section and prepare for the remainder of this chapter, we briefly introduce our exemplary application task, sepsis prediction.

Leveraging data derived from electronic health records of intensive care patients, our aim is to predict whether a patient will develop sepsis, which is defined as a life-threatening and dysregulated host response to infection [23]. For each patient, we are provided with a multivariate time series (comprising monitoring and laboratory variables) combined with a binary class label (sepsis case or control). For certain applications, we will also be interested in the time-stamp associated with the class label, such that we can predict an *onset* of sepsis. More details about the specific applications are provided in the application figures.

## 2 Time Series Classification Tasks

In time series classification (TSC), we aim to predict a time series' associated class label by leveraging global or local time series features. Due to the ordered nature of time series data, classification of time series differs from a regular classification problem and has to be treated accordingly. There are essentially two main branches of time series classification, namely, *feature-based* and *distance-based*. The former includes a feature extraction step prior to classification. Commonly extracted time series features comprise statistical moments, maximum and minimum, features extracted from the discrete Fourier or Wavelet transform (DFT and DWT, respectively), or path signatures [24]. The goal of time series classification is then to learn the importance of features (or attributes) with respect to class membership. As with other supervised ML methods, the class labels are known in advance and the algorithm is trained on an example dataset. A biomedical example is the

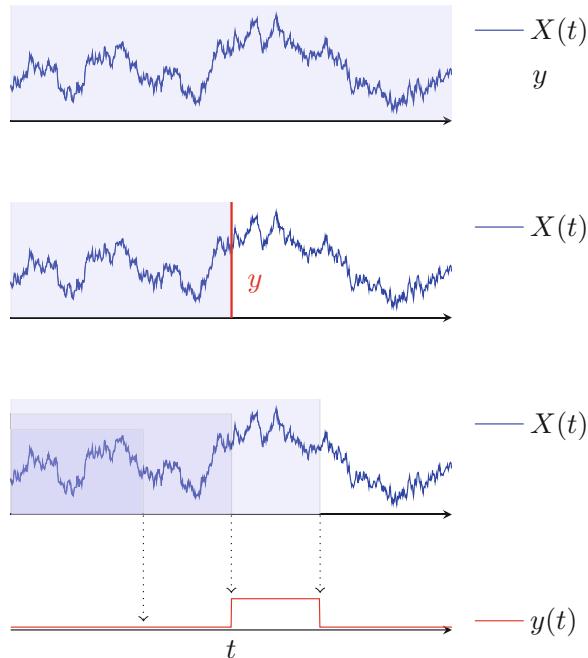
feature-based classification of electrocardiograms (ECG) of healthy individuals and those of patients with myocardial infarction. The length, width, maximum, and shape of the so-called QRS complex of the ECG signals are unambiguous features that will enable the classification algorithm to distinguish these two groups. Subsequently, when the classifier is presented with an unlabeled data point (from a patient or a healthy individual), it can automatically assign the time series to the correct class. The feature-based classification methods suffer, however, from two major limitations: (1) feature extraction is time-consuming and often depends on the knowledge of a domain expert; and (2) this can lead to a significant loss of information regarding the original signal. Distance-based classification methods circumvent the feature extraction phase and instead implement a definition of suitable distances, among which the most common one is dynamic time warping (DTW). Briefly, DTW aligns two time series in an optimal manner, thereby providing a measure of alignment and distances between the two time series. A notable advantage of DTW over other distance-based methods (e.g., Euclidean distance) is that DTW can be used on time series of variable length. The classification can then be performed based on the distances by means of a nearest neighbour classifier. Detailed description of how to implement the  $k$  nearest neighbor ( $k$ -NN) algorithm combined with DTW is given in Subheading 3.1.1. Before immersing further into TSC methods in Subheading 3, this section introduces frequently encountered *types* of TSC problems. This classification is based on two fundamental questions:

1. Which part of a time series is used as input data for TSC?
2. When do we make a prediction in TSC?

## 2.1 Whole Time Series Classification

In whole time series classification (WSC), for a dataset of  $n$  samples, we are provided a set of tuples  $\{\{T^{[0]}, y_0\}, \dots, \{T^{[n]}, y_n\}\}$  which for the sample  $i$  pairs its *entire* time series  $T^{[i]} \in \mathcal{T}$  with *one* associated class label  $y_i \in \mathcal{Y}$ . The goal is then to learn a mapping  $f : \mathcal{T} \rightarrow \mathcal{Y}$ . Using  $f$ , we predict the class label  $y_*$  of an unseen test time series  $T^{[*]}$  with  $y_* = f(T^{[*]})$ . Here, the class label  $y_i$  does *not* necessarily correspond to a specific event taking place during the time window to be studied. Rather,  $y_i$  represents the class membership of the time series  $T^{[i]}$  with respect to a *phenotype* or *outcome* that can lack a timestamp or even dependence on time in general.

To illustrate this, consider the task of analyzing audio streams of recorded speech. Predicting whether such a time series corresponds to a male or female speaker refers to a label independent of time, whereas the recognition of malicious content, such as hate speech, would refer to a specific *event* (statement in the speech). Nevertheless, both types of labels are used for WSC. For many applications, WSC tasks are plagued with considerable noise, such



**Fig. 3** Illustration of different types of time series classification. We highlight time dependence with  $(t)$ . First row: Whole series classification: the time series  $X(t)$  is provided with a label  $y$ ; second row: window-based classification: the time series  $X(t)$  is provided with a time-stamped label  $y$  (red); third row: timepointwise classification: Both input time series  $X(t)$  and label  $y(t)$  are sequences. We highlight three prediction steps leveraging all input time series up until the given point in time. In all panels, the filled boxes (blue) indicate which time series window is used as input for prediction

that the identification of the task-relevant *scale(s)* and *subsequence(s)* of an input time series remain critical challenges. To reveal task-relevant *scales*, multiscale representations, such as wavelet or Fourier transforms, offer straightforward solutions [25]. The second challenge, how to identify relevant time series *subsequences*, will be explored in Subheading 3.1.2.

In the next section, we discuss further types of time series classification (see Fig. 3 for an illustration), focusing on time-stamped labels. In contrast to those approaches, methodologically, WSC shares a considerable overlap with the broader domain of sequence classification, where attributes are still *ordered*; however, their order does not necessarily reflect a time axis. A time-agnostic example would be the classification of genetic sequences [26].

## 2.2 Onset Detection

Onset detection is a subtype of time series classification in which—as opposed to whole series classification—class labels are provided with a *time-stamp* [25]. In the biomedical domain, this might refer to any time-dependent outcome or event of interest: the *start* of a

phenotype (e.g., a patient develops a stroke) or a *change* of a phenotype (e.g., a heart failure patient is hemodynamically stable and suddenly decompensates).

In the literature, there are two main approaches for treating time-stamped labels in time series classification:

### 2.2.1 TimePointWise Classification

The class label of each point in time is predicted. This concept has previously been introduced as “strong temporal classification” [27]. For a given prediction step, the time series up until this point is included for prediction. As a major benefit of this approach over standard time series classification, a classifier’s performance (and therefore relevant points in time) can be examined over time. For instance, if a certain pattern in the time series is predictive of a subsequent class label, the performance of the classifier would be close to such a pattern, whereas windows missing any informative features achieve lower performance. In contrast, a whole series classifier is oblivious with respect to which feature at which point in time was responsible for a given prediction unless specific feature selection or attention mechanisms [28] are included. Furthermore, timepointwise classification naturally allows for *causal* predictions, in the sense that no future data is used for predicting a class label in the present, which can be a limitation for whole series classification. For example, Harutyunyan et al. [29] proposed a phenotyping task for intensive care data, where time series of health records are used to predict a patient’s phenotype in the form of diagnostic codes. Crucially, these codes lack a time-stamp. In cases like this, standard time series classification is limited to a retrospective setup, such that predictive features potentially stem from measurements and interventions that *follow* the diagnosis, instead of preceding it.

At first, timepointwise classification seems straightforward; however, this approach necessitates multiple experimental design choices: For instance, how should unlabeled points in time be treated? Does the task at hand allow for discarding those points in time from classification, or would this bias the data set with respect to the label availability?

Furthermore, when classifying on the time point level, the next challenge is the *evaluation* of a timepointwise classifier. In many biomedical applications, for correct predictions to be useful they must occur in a certain window before an event or outcome of interest. For example, if an ICU alarm system for patient deterioration sounds an alarm at the moment (or shortly before) a patient destabilizes, it might be too late and not help in *preventing* the outcome and could therefore be treated as a false-negative prediction. Likewise, if the alarm went off several days too early, the alarm should be treated as a false positive, even if the patient ultimately deteriorates.

For these challenges there is no one-fits-all solution; rather, they need to be reviewed diligently with domain experts when setting up a timepointwise onset detection task.

While recent surveys on time series classification focused foremost on subtyping the input time series as well as the involved algorithms [30, 31], there remains a lack in distinguishing classification tasks with respect to the time series labels, which we deem particularly relevant for biomedical applications. Thus, next we describe an approach to onset detection which, to the best of our knowledge, has not been explicitly specified, yet has been widely used in applications [32–34].

### 2.2.2 Time Series Classification on Time Series Windows

As an alternative to timepointwise classification, time-stamped labels have been leveraged for classifying time series windows that *precede* the class label's time-stamp [35]. This can be understood as a compromise between time pointwise classification and whole time series classification. Here, the prediction windows are also *causal* (in the sense that there is no leakage from future input data) with the advantage that certain problems associated with timepointwise classification can be circumvented. For instance, there is no need for handcrafted time thresholds for specifying the confusion matrix as in timepointwise classification. As an obvious drawback of this approach, the window selection can be rather ad hoc if the task at hand does not predetermine which time windows are meaningful to use for prediction. For instance, if, on the one hand, a general practitioner uses a classifier to predict an adverse outcome of her patients, it would be a nontrivial choice of experimental design, how far the classifier should be able to look back in medical patient history. On the other hand, if an intensive care clinician wants a classifier to predict adverse outcomes based on his patient's current stay, the time series window would be clearly predefined (starting at admission until adverse outcome).

In Fig. 5, a medical application of an onset detection task is exemplified [34]. For this, we focus on the second strategy, that is, to predict time-stamped class labels based on a preceding time series window.

#### Onset Detection for Sepsis Prediction

We highlight onset detection paradigmatically by means of reviewing a medical application of time series classification: to detect the onset of sepsis in the intensive care unit [34]. While sepsis will serve as a use case, the discussed concepts, key steps, and challenges are applicable to a broad range of biomedical time series applications.

### Task: Detecting Sepsis Onset

Our phenotype, sepsis, is defined as a life-threatening, dysregulated host response to infection [23] that necessitates antibiotic treatment within a critical time window after its onset. Each hour delaying the initiation of antibiotic intervention dramatically increases mortality [36]. Therefore, early identification of sepsis by leveraging the abundance of health record data to create in-hospital alert systems may facilitate a timelier response, thereby reducing in-hospital mortality [37].

### Label Extraction

For onset detection, a class label requires a time-stamp. This additional information can enforce that solely information from the past and present is used to predict a future target. However, in biomedical applications, it can be tricky to associate a precise time-stamp with a class label. In our example task aiming at detecting sepsis, we first need to *annotate* patient time series with a sepsis onset time. Even though electronic health record datasets track diagnosis codes, these are not generally provided with a time-stamp. In our application, this hindrance thus has to be addressed by implementing a recent sepsis definition [23] that allows for temporal resolution to derive a sepsis onset time. Defining and extracting labels should be considered as the most crucial step for setting up an onset detection task. Bias arising in this step can drastically affect the difficulty of downstream tasks, as we will see in the next paragraph.

### Classification Setup

Recent literature on ML for sepsis detection focused on the classification of time series subsequences [32, 34, 38], by using a time series window preceding the sepsis onset to predict the class label. This refers to the second strategy for implementing onset detection as introduced in Subheading 2.2. A nontrivial challenge of this strategy (irrespective of application domain) is the window selection of controls, that is, defining a time-stamp for the negative class. For instance, Futoma et al. [32] aligned the case's sepsis onset time with the discharge time of controls, such that their model is provided with the following input for prediction:

- The patient's time series from intensive care admission up until sepsis onset for cases.
- The time series of the patient's entire intensive care stay for controls.

This approach is problematic for two reasons:

1. Cropping only the case time series can bias the classification task, such that trivial features (e.g., length of the time series) might already be sufficient to solve it.
2. Comparing the last  $n$  hours preceding a sepsis onset with the  $n$  hours preceding a control's discharge is simplifying the investigated task as controls shortly before discharge are by definition stable enough to shortly be discharged.

### Case-Control Matching

In their follow-up paper, Futoma et al. [35] corrected for this by introducing a more elaborate case-control alignment scheme: to align sepsis onset time with a virtual control onset time, which for controls occurs after the same percentage of hospital stay length as sepsis onset of their matched case, respectively. To address the possibility that case and controls have different distributions of time series length, a matching in absolute hours has been proposed [34]. Futoma et al. observed a *drastic* drop in performance of their initially presented sepsis prediction model: 2 h before sepsis onset, the area under the precision-recall curve dropped from above 0.9 to below 0.5 [35].

In contrast to time series window classification, only recently, the 2019 PhysioNet Challenge has introduced a *timepointwise* onset detection task to the sepsis application [39]. As previously indicated in Subheading 1.2, for this type of analysis, the organizers of the challenge chose thresholds and utility functions that would map positive and negative predictions to utility values depending on the nearby positive and negative class labels.

---

## 3 Methods for Time Series Classification

This section aims to guide the reader through the methodological aspects of common distance-based approaches and deep learning approaches to time series classification. To foster the conceptual understanding of the methods discussed, a real-world running example and illustrative examples complement this section. Finally, the reader is provided with suggestions for freely available tools for the implementation of the methods discussed.

### 3.1 Distance-Based Time Series Classification

The first class of classification methods rely on some notion of distance between two time series or time series subsequences. Intuitively, a distance can be seen as a proxy for dissimilarity. Two time series that are in close proximity (i.e., they have a small distance) under some distance measure, are likely to come from

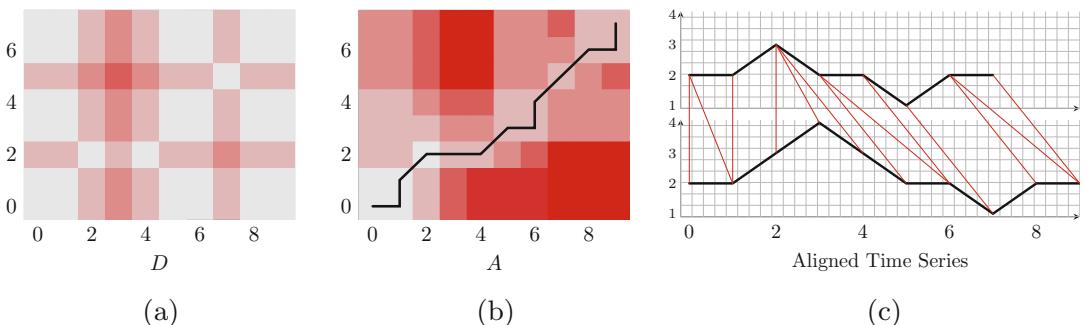
the same class. In the following paragraphs we will first introduce a distance measure for time series that allows alignment of time series of unequal length: dynamic time warping. We will then show how DTW can be used to derive a strong baseline for whole time series classification using a  $k$ -nearest neighbors classifier. Subsequently, we introduce the concept of time series shapelets, that is, subsequences that maximize classification performance. We will then briefly introduce a method that takes a statistical view on shapelets and is rooted in the field of significant pattern mining.

### 3.1.1 Dynamic Time Warping and Nearest Neighbors

A simple, yet effective way to classify time series utilizes a  $k$ -NN approach combined with a time series specific distance measure, namely, DTW distance [40]. DTW is an alignment strategy for time series of possibly different lengths from which a distance measure (not a metric) can be derived. The DTW algorithm first aligns two time  $T^{[a]}$  and  $T^{[b]}$  series by finding an optimal match of all measurements. To find a matching, an accumulated cost matrix  $A \in \mathbb{R}^{m^{[a]}+1 \times m^{[b]}+1}$  is constructed. Each entry of  $A$  can be recursively computed in the following way:

$$A_{i,j} = D_{i,j} + \min(A_{i,j-1}, A_{i-1,j}, A_{i-1,j-1}) \quad (1)$$

where  $D$  is an  $m^{[a]} \times m^{[b]}$  distance matrix with  $D_{i,j} = d(v^{[a]}_i, v^{[b]}_j)$ . We initialize  $A_{0,0} = 0$ ,  $A_{i,0} = \infty$ , and  $A_{0,j} = \infty$  for all  $1 \leq i \leq m^{[a]}$ , and  $1 \leq j \leq m^{[b]}$ . Once  $A$  is obtained, the distance between both time series is defined as  $\text{DTW}_{\text{dist}}(T^{[a]}, T^{[b]}) = A_{m,m'}$ . Figure 4 illustrates how such an alignment is constructed. To obtain the alignment of measurements, a so-called warping path is constructed. The warping path starts at  $(0, 0)$  and ends at  $(m, m')$ . At each step, one out of three options is chosen: (1) step right, (2) step up, or (3) step up and right. To construct the warping path, at each step,



**Fig. 4** Visualization of the DTW alignment of two time series of unequal lengths. **(a)** The distance matrix of individual observations. The indices on the axes refer to the individual measurements in the time series. The chosen distance metric is the absolute difference  $d(v^{[a]}_i, v^{[b]}_j) = |v^{[a]}_i - v^{[b]}_j|$ . **(b)** shows the accumulated cost matrix as constructed by Eq. 1 with the warping path. For illustrative purposes, the first row and column (infinity initializations) of  $A$  are not shown. **(c)** Visualization of both time series and the alignment of individual measurements (red lines) according to the warping path in  $A$

the cell with the lowest accumulative cost is chosen. Each cell represents the cheapest alignment costs up to this point.  $\text{DTW}_{\text{dist}}$  is the accumulated distance at the end of the alignment ( $A_{m,m'}$ ).

Using this distance measure in combination with a simple  $k$ -NN classifier, as explained in the next section, has been proven to work well for a large variety of data sets [21, 41].

Extensions of DTW exist in which the warping path is constrained in how far it can deviate from the diagonal of  $A$ . Imagine a band along the diagonal of a certain width in which the warping path can lie. The width of this diagonal is called warping window width and it limits the number of points a measurement can be mapped to. Optimizing the warping window width on a given data set is an active field of research; [42] provides a good overview.

### $k$ -nearest Neighbor Algorithm

Given a labeled time series data set  $D_L$  and a query time series  $T_q$  for which we would like to predict its class label  $y_q$ , we proceed in the following way: We first compute  $\text{DTW}_{\text{dist}}$  between  $T_q$  and all time series from the data set. Then, the  $k$  time series closest to  $T_q$ , as measured by the DTW distance, are selected to determine the predicted label. When  $k = 1$ ,  $T_q$  will receive the label of its closest neighbour. When  $k > 1$ , a majority vote will determine the predicted label and—if necessary—ties are broken randomly [43]. A DTW 1-NN classifier can be formulated as the following minimization problem:

$$T^{[i]} = \underset{T' \in D}{\operatorname{argmin}} (\text{DTW}_{\text{dist}}(T_q, T')) \Rightarrow y_q = y_i,$$

where  $T^{[i]}$  is the closest neighbour to  $T_q$  in DTW-space and  $y_i$  is its label.

$k$ -NN is considered a *lazy learner* as no learning phase is needed; in order to classify a query sample, the distances to all data points needs to be computed. To reduce the number of distance computations, one can make use of the triangle inequality, which is by definition fulfilled when we choose our distance measure  $d(\cdot, \cdot)$  to be a metric. Note however, that  $\text{DTW}_{\text{dist}}$  does not fulfill the triangle inequality, and using this speed up with DTW can lead to incorrect conclusions. The hyperparameter  $k$  is typically chosen via nested cross-validation (see ref. 44 for an introduction to model selection).

### Dynamic Time Warping for Sepsis Prediction

In the previous section, we introduced DTW and its application to nearest neighbors (NN) classifiers. Here, we illustrate DTW-based NN classification on our example application, sepsis prediction. To this end, we refer to recent work proposing (among other approaches) DTW- $k$ NN for sepsis prediction [34].

## Dataset and Filtering

The MIMIC-III critical care database was used [2]. Patients were included as a *case* if a sepsis onset occurs during their stay in ICU (according to Sepsis-3, [23]). Controls were defined as those patients who do *not* develop sepsis during their stay, and additionally receive no sepsis-related ICD-9 billing code (to ensure no sepsis patient who had their onset *before* admission would end up in the controls). Following these criteria, the authors retrieve 1797 sepsis cases and 17,276 controls. With the aim of predicting sepsis *early*, they followed [38] in removing instances where sepsis onset occurs earlier than 7 h after admission to enable a 7-h prediction window. However, to preserve a realistic class balance of around 10%, this criterion is applied only after the case-control matching (more details below). Furthermore, the authors excluded the following:

- Pediatric patients under 15 years old.
- ICU stays with relevant data missing (e.g., no admission/discharge time, or no chart data available).
- Patients not logged via Metavision.

Following the class balance, the authors matched each case to ten controls for assigning a matched control onset time equal to the sepsis onset time (in hours after admission) of the case, respectively. After those filtering steps, for 570 cases and 5618 controls 44 irregularly measured laboratory and vital parameters as extracted up to 48 h before onset (and after admission) were used for binary classification.

## Experimental Setup

To challenge the previous state-of-the-art sepsis prediction method, MGP-RNN [32], the authors propose a DTW- $k$ NN classifier as well as a deep learning method MGP-TCN which combines the multitask gaussian process (MGP) adapter framework [32] with a temporal convolutional network (TCN) (for more details, refer to Subheading 3.2). As an additional baseline, a standard TCN is used. For the standard TCN and the DTW- $k$ NN, the time series were binned in hourly averages while carry-forward imputation was applied (i.e., for empty bins the last previously observed value is used), whereas for the two MGP-based methods, the MGP performs the imputation and the subsequent neural network performs the classification.

For training, three iterations of random splitting (80% training, 10% each for validation and testing) were applied. Each time series channel (corresponding to a vital or laboratory variable) was  $Z$ -scored using the respective train split statistics.

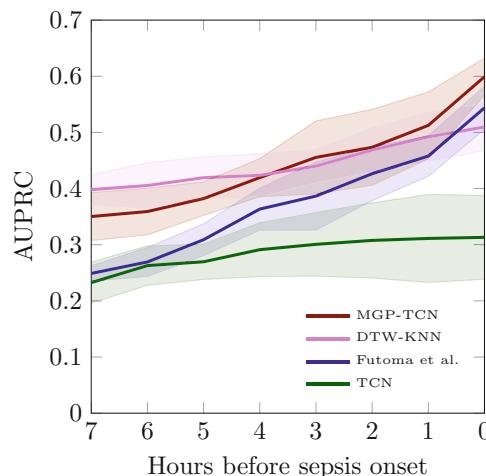
Due to substantial class imbalance (9.2% case prevalence), all models were evaluated in terms of area under the precision–recall curve (AUPRC) on the test split. To evaluate *timeliness* of predictions, for a  $n$ -hour prediction horizon, the time series input data after  $n$  hours before onset are discarded, resulting in a performance plot over varying horizon up to 7 h (see figure on the right, as adapted from [34]).

## Results

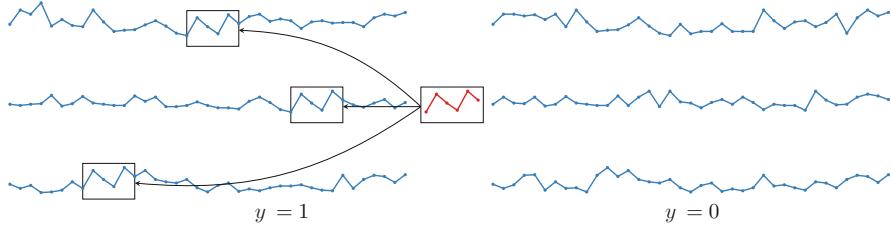
Figure 5 shows the prediction performance in terms of AUPRC. Both the DTW- $k$ NN as well as MGP-TCN outperform the previous state-of-the-art MGP-RNN [32]. In this figure, we observe that both DTW- $k$ NN and TCN show a more moderate gain in performance when approaching sepsis onset as opposed to the MGP-imputed models, which could be explained by the underlying carry-forward imputation scheme.

### 3.1.2 Shapelets

In 2009, Ye and Keogh first introduced a novel data mining approach to classify time series based on short time series subsequences—the so-called *time series shapelets* [17]. Time series shapelets, illustrated in Fig. 6, are defined as subsequences that maximize predictive power. Their usefulness has been demonstrated in a variety of biomedical applications, including genomics [45], sepsis diagnosis [46, 47], mortality prediction [48], and gait recognition [49]. Since shapelets are subsequences from the data set we wish to analyze, they are by nature interpretable. Interpretable subsequences that distinguish cases (e.g., patients with sepsis) and controls (e.g., patients without sepsis) can be of fundamental



**Fig. 5** Predictive performance (AUPRC) at different prediction horizons



**Fig. 6** Illustration of a shapelet (red) that is enriched in all samples with a positive class label  $y = 1$ . We consider a subsequence to be present in a time series if its minimal distance along all points in the time series is less than or equal to a given threshold  $\theta$ . Reproduced from [47] with permission from Oxford University Press

importance to gain new biomedical knowledge. In the following paragraphs we lay out the necessary definitions and steps to find shapelets.

First, to measure the predictive power of a classifier, we introduce the concept of entropy of a labelled data set  $D_L$  with binary class labels  $\mathcal{Y} = \{0, 1\}$ :

**Definition 7 (Entropy):** Let  $|\mathcal{Y}_0|$  be the number of samples that belong to class 0 and  $|\mathcal{Y}_1|$  the number of time series belonging to class 1. Furthermore, let  $p(\mathcal{Y}_0) = \frac{|\mathcal{Y}_0|}{|D_L|}$  and  $p(\mathcal{Y}_1) = \frac{|\mathcal{Y}_1|}{|D_L|}$ . The entropy of the data set is then defined as follows:

$$I(D_L) = -p(\mathcal{Y}_0) \log_2 p(\mathcal{Y}_0) - p(\mathcal{Y}_1) \log_2 p(\mathcal{Y}_1)$$

Informally, entropy [50] is a measure of disorder which is intuitively understandable if we consider the case of a “pure” or “ordered” data set that only contains samples from one class (e.g.,  $|\mathcal{Y}_1|=|D_L|$ ). In this situation, the entropy of the data set is 0, as in “zero disorder.” In the other extreme, two classes are equally distributed among the samples of the data set (e.g.,  $|\mathcal{Y}_0|=|\mathcal{Y}_1|$ ). The entropy of such a data set takes its maximal value of 1. Let us now assume we want to evaluate a binary classification algorithm that splits a data set into two sets:  $D_0$  which contains all samples that the classifier predicts to belong to class 0, and  $D_1$ , composed of all samples which are classified to belong to class 1. The entropy of this split can be defined by summing the weighted entropy of both sets:

$$I_{\text{split}}(D_L, D_0, D_1) = \frac{|D_0|}{|D_L|} I(D_0) + \frac{|D_1|}{|D_L|} I(D_1).$$

To judge the performance of this classifier, we compute the difference of entropy before and after the split. This measure is called information gain:

$$I_G(D_L, D_0, D_1) = I(D_L) - I_{\text{split}}(D_L, D_0, D_1)$$

We will need this measure later to find the time series subsequence that splits the data set best. But before continuing this thread, we now briefly address the problem of finding a set of suitable subsequences and how we can use them to split a data set in the abovementioned way.

### Collect Shapelet Candidates

The first question to ask when we want to find one or more subsequences for whole time series classification is: Which subsequences are most discriminative? Since the set of all real-valued sequences of any length is infinitely large, the search space must be restricted. In a first step, we need to define how long our motif of interest (subsequence) could be. This value or range is typically domain-specific. For the sake of clarity, we define a value  $w$  to be the exact length of the time series subsequence we are interested in. Sticking to all real-valued sequences of length  $w$ , we still obtain infinitely many possible subsequences. To further reduce the search space, we can restrict ourselves to subsequences that originate from the data set by using the notion of a sliding window of width  $w$  and stride  $s$  where the latter defines the distance or shift of two neighboring windows.

We begin subsequence extraction by considering only the first  $w$  measurements of a time series and add its sequence  $\{v_0, \dots, v_{w-1}\}$  to our list of shapelet candidates  $\mathcal{S}$ . Next, we shift the window by  $s$  and add the sequence that is now covered by the window ( $\{v_{0+s}, \dots, v_{s+w-1}\}$ ). We move this window until no subsequence of length  $w$  can be extracted anymore. Generally speaking, this leads to  $\lceil(m-w+1)/s\rceil$  subsequences for each time series in our data set. To provide an exact solution, we would like to consider all possible, even overlapping, subsequences and hence set  $s=1$ .

### Compute Distances Between Candidates and Time Series

To use a time series subsequence for classification, we need to know whether it appears in a time series or not. Naïvely, we could take a shapelet candidate and determine whether we find an exact match in a time series. More formally, we could count the appearance of subsequence  $S$  if and only if we find a  $p$  such that  $\{s_0, \dots, s_{w-1}\} = \{v_p, \dots, v_{p+w-1}\}$  for  $0 \leq p \leq m-w$ . This, however, is an extremely strict rule, as it does not allow for even a single deviation. In real-world data sets, not only are *exact* matches are rare, but we are also not interested in them. Instead, we are interested in sequences that express a specific *overall shape* as earlier illustrated in Fig. 6. To allow for a certain flexibility in our match finding (e.g., counting a match if  $\{s_0, \dots, s_{w-1}\} = \{v_p \pm \epsilon, \dots, v_{p+w-1} \pm \epsilon\}$  for small  $\epsilon$ ), we need a measure that describes the dissimilarity of a subsequence and a time series. If the dissimilarity is small enough or lies under a specific threshold  $\theta \in R$ , we claim that  $S$  occurred in  $T$ . Let us first define a dissimilarity, or distance

measure for two  $w$ -length univariate time series  $X$  and  $\Upsilon$  as their Euclidean distance:

$$\text{dist}(X, \Upsilon) = \sqrt{\sum_{i=0}^{w-1} (x_i - y_i)^2}, \quad (2)$$

where  $x_i, y_i$  is the observation value at time point  $i$  from  $X$  and  $\Upsilon$ , respectively. Given Eq. 2, we now define the distance of a short subsequence  $S$  and a longer time series  $T$  as follows:

$$\text{dist}(S, T) = \min_j (\text{dist}(S, \{v_j, \dots, v_{j+w}\})) \quad (3)$$

We calculate the distance according to Eq. 2 at all possible time points of  $T$  and choose  $\text{dist}(S, T)$  to be the smallest one.

#### Find the Best Candidate

Equation 3 allows us to construct a set  $\Theta$  that contains the distances between all time series from the data set and all shapelet candidates from  $\mathcal{S}$ . These distances can now be used as a threshold  $\theta$  to derive a simple classification rule: If  $\text{dist}(S, T) < \theta$ , we say  $T$  belongs to class 0, otherwise it is classified as belonging to class 1. Coming back to the notion of entropy and information gain, for each subsequence–threshold pair, we can now split the data set into two sets:  $D_{\text{dist}(S, T) < \theta}$  with time series whose distance to the subsequence is less than  $\theta$  and conversely,  $D_{\text{dist}(S, T) \geq \theta}$  and compute their information gain:

$$I_G(D_L, D_{\text{dist}(S, T) < \theta}, D_{\text{dist}(S, T) \geq \theta}) = I(D_L) - I_{\text{split}}(D_L, D_{\text{dist}(S, T) < \theta}, D_{\text{dist}(S, T) \geq \theta})$$

We can now obtain the shapelet–threshold pair with the highest information gain by solving the following optimization problem:

$$\underset{\theta \in \Theta, S \in \mathcal{S}}{\text{argmax}} (I_G(D_L, D_{\text{dist}(S, T) < \theta}, D_{\text{dist}(S, T) \geq \theta}))$$

However, often it is much more useful to employ multiple shapelets to solve a classification task. Decision trees with multiple levels (*see* ref. 17) or a random forest classifier are two suitable and simple approaches that can be utilized to do so. In fact, shapelet-based classifiers embody a whole category of methods which extend the brute force approach of shapelet mining as presented here. Three prominent extensions are fast shapelets [51] which are based on a discretized representation of time series, learned shapelets [52] which uses the gradient descent algorithm to find shapelets, and the shapelet transform [53] where distances to shapelets are used as features.

#### Significant Subsequence Mining

One drawback of evaluating subsequences solely on their classification performance is the lack of a notion of their statistical significance. If the ultimate goal is to identify a short time series that is enriched in a class, classification accuracy is not necessarily the best

fitting measure. Especially in the life sciences, it is important to understand to what extent a subsequence is associated with a given phenotype in terms of a  $p$ -value. Assessing the statistical significance of subsequences comes with a variety of statistical challenges which are addressed in the following paragraphs. The first method to cover such aspects was introduced by Bock et al. [47], and can be summarized by the following steps:

1. Collect all *shapelet candidates*<sup>2</sup> and compute the distances to all time series as described above.
2. Assess the statistical significance of shapelet candidates.
3. Discard shapelet candidates that are not statistically significant after multiple hypothesis correction.

#### Collect Shapelet Candidates

While the restrictions we impose on the shapelet candidate generation process (i.e., a sliding window on the data set) reduces the search space to a certain degree, the number of most discriminative shapelets can still be extremely large and the problem of multiple hypothesis testing [54] becomes an issue. What this means and how we can cope with it will be described in the following paragraphs.

#### Assess the Statistical Significance of Shapelet Candidates

Measuring the statistical association of patterns with a binary class variable (e.g., phenotype) is the core problem addressed by the field of significant pattern mining (SPM) (see ref. 55 for an introduction to the field). The premise of SPM is to measure the statistical association between a *pattern indicator variable* and a binary class label (e.g., case or control) while efficiently solving the multiple hypotheses testing problem. The binary *pattern indicator variable* denotes whether a given pattern is present in a sample or not. In combination with the interpretability of time series subsequences, SPM serves as a powerful tool to find understandable patterns within a sound statistical framework.

In order to perform a statistical test, we need to define a null hypothesis ( $H_0$ ). In the realm of subsequence mining, we define the null hypothesis in the following way: There is no nonrandom association between the frequency of a given subsequence and the binary class label. A measure for this nonrandom association is the  $p$ -value. It is the probability of obtaining a test statistic value that is at least as extreme as the observed one, assuming statistical independence between subsequence and class label. We deem a pattern significant if its  $p$ -value falls below a certain threshold which is typically chosen beforehand and set to  $\alpha$ . To compute a  $p$ -value, we define a contingency table and perform a statistical test appropriate for discrete data (e.g., Fisher's exact test [56]) in the following way:

---

<sup>2</sup> We will refer to subsequences that are statistically significantly associated with a class label as *shapelets*. Note that this term is typically used for subsequences that are maximizing information gain.

**Table 1**

**Contingency table used to compute the  $p$ -value of a subsequence  $S$  at a distance threshold  $\theta$ .**  $a_S$  equals the number of time series that belong to class  $y = 1$  and whose distance to subsequence  $S$  is less than or equal to the current threshold  $\theta$ .  $c_S$  counts the time series that belong to class  $y = 0$  and whose distance to subsequence  $S$  is greater than the current threshold  $\theta$ .  $r_S$  is the number of time series in which  $S$  occurs and vice versa for  $q_S$ .  $n_1$  is the number of time series belonging to class 1

Class label	$\text{dist}(S, T) \leq \theta$	$\text{dist}(S, T) > \theta$	
$y = 1$	$a_S$	$b_S$	$n_1$
$y = 0$	$d_S$	$c_S$	$n_0$
	$r_S$	$q_S$	$n$

Let  $\mathcal{S}$  be the set of all  $\lceil(m - w + 1)/s\rceil$  subsequences and the set of all distances between each time series in  $D_L$  and all elements in  $\mathcal{S}$ , following the distance calculation in Eq. 3. This leads to  $|\Theta| = n(\lceil(m - w + 1)/s\rceil)$  distance values, each of which we will use as a distance threshold  $\theta$  to define the cells of a contingency table as shown in Table 1.

The number of individual hypothesis tests to be performed ( $|\Theta|$ ) depends on the data set size, the length of the time series, the selected subsequence size  $w$ , and stride  $s$ . As this number can be large even for smaller data sets, it becomes more and more likely to falsely deem a subsequence significant (multiple testing problem). Falsely rejecting a true null hypothesis is also called Type 1 Error. For example, a set of 100 time series of length 100 and subsequence size  $w = 10$  leads to  $100 \cdot 91 = 9,100$  subsequences. For each subsequence, a statistical test will be performed at 100 thresholds which leaves us with  $100 \cdot 9,100 \approx 90,000$  hypotheses to be tested. The number of false positives can be expected to be  $\alpha|\Theta|$  and since  $|\Theta| >> n$ , this will typically lead to an extremely high number of Type I Errors (e.g.,  $0.05 \cdot 90,000 = 4,500$  for the example above and with significance threshold  $\alpha = 0.05$ ). A popular way of counteracting this problem is to reduce the significance threshold  $\alpha$  to be  $\delta_{\text{bon}} = \frac{\alpha}{|\Theta|}$  [57]. However, as the number of performed tests is extremely high, this method turns out to be extremely conservative, resulting in an extreme loss of statistical power (i.e., the probability of rejecting a false null hypothesis). A less conservative way of correcting for multiple tests was proposed by Tarone [58] and will be discussed next.

#### Multiple Hypothesis Correction

When we deal with discrete test statistics based on contingency tables (e.g., Fisher's exact test or Pearson's  $\chi^2$  test), only a finite number of possible  $p$ -values exists. This is intuitively understandable, as for a fixed data set, there is only a finite number of possible table configurations and hence a *minimum attainable p-value*  $p_{\min}$  among these. Terada et al. showed that for Fisher's exact test,  $p_{\min}$  is

nonzero and only depends on the size of the data set  $n$ , the number of cases  $n_1$ , and the number of occurrences of the pattern  $r_s$  [59]. To understand why this is the case, we have to consider the most extreme table configurations (the ones which lead to the smallest possible  $p$ -values) while keeping the aforementioned variables fixed. In one configuration, the pattern *only* occurs in the samples with label  $y = 1$  ( $a_s = r_s$ ). The second extreme configuration arises when the pattern *only* occurs in the controls ( $d_s = r_s$ ). If  $p_{\min}$  of a pattern is larger than the significance threshold, it will never (even in the most extreme configuration) become significant; such patterns are called *untestable patterns* or *untestable hypotheses*. Let  $n_t$  be the number of testable patterns. Tarone's adjusted significance threshold is then defined as  $\delta_{\text{tar}} = \max \{\delta | \delta \cdot n_t \leq \alpha\}$ , a value typically much higher than  $\delta_{\text{bon}}$  which leads to an increase in statistical power. Computing  $\delta_{\text{tar}}$  is an iterative process in which the testability is determined for each pattern and—if testable—added to the list of testable patterns. This iteratively increases  $n_t$  and has the consequence that we have to adjust  $\delta$  to fulfil the condition  $\delta \cdot n_t \leq \alpha$ . Taking care that this condition is fulfilled is equivalent to controlling the familywise error rate (FWER). FWER is the probability of drawing at least one false positive conclusion at significance threshold  $\delta$ :  $\text{FWER}(\delta) = \text{Prob}(\text{FP}(\delta) \geq 1)$ , where  $\text{FP}(\delta)$  is the number of false positives at  $\delta$ . In order to do so, we reduce  $\delta$  as long as the condition is not fulfilled. In practice, we chose the next relevant  $p$ -value from a nonincreasingly sorted set of precalculated values. All relevant  $p$ -values can be precomputed using the closed form of the minimum attainable  $p$ -value from Eq. 4 (following the notation introduced in [55]) for  $r_s \in \{0, \dots, n\}$ .

$$p_{\min}(r_s) := \begin{cases} 1 - F_{\chi^2} \left( (n-1) \frac{n_b}{n_a} \frac{r_s}{n - r_s} \right) & \text{if } 0 \leq r_s < n_a \\ 1 - F_{\chi^2} \left( (n-1) \frac{n_a}{n_b} \frac{n - r_s}{r_s} \right) & \text{if } n_a \leq r_s < \frac{n}{2}, \\ 1 - F_{\chi^2} \left( (n-1) \frac{n_a}{n_b} \frac{r_s}{n - r_s} \right) & \text{if } \frac{n}{2} \leq r_s < n_b, \\ 1 - F_{\chi^2} \left( (n-1) \frac{n_b}{n_a} \frac{n - r_s}{r_s} \right) & \text{otherwise.} \end{cases} \quad (4)$$

where  $F_{\chi^2}(\cdot)$  denotes the cumulative density function of a  $\chi^2$ -distribution with one degree of freedom,  $n_a = \min(n_1, n - n_1)$ , and  $n_b = \max(n_1, n - n_1)$ .

Note that by iteratively adjusting the current significance threshold, the number of testable hypotheses changes and what was testable at the threshold at iteration  $j$  might not be testable at iteration  $j + 1$  and we would remove it from the list of testable patterns. Once we iterated over all subsequences, the final adjusted significant threshold  $\delta_{\text{tar}}$  is reached and all patterns which remain in the list of testable patterns are significant under this threshold.

In this chapter, we have introduced the fundamentals of significant shapelet mining. There are still several open challenges to be solved. The first one pertains to the reduction of shapelet candidates to a set of subsequences that is sufficiently large, and reasonably small. The first factor is necessary to retain any expressive subsequences while the second is important to reduce the burden of computation. To further decrease computational costs, Bock et al. introduced a contingency table based pruning criterion by bounding  $p$ -values of partially filled contingency tables (*see* ref. 47). One could also think of less conservative correction approaches that control for the false discovery rate (FDR). Another relevant issue is the choice of distance measure. The distance we introduced in Eq. 3 does not fulfil the condition of identity of indiscernible and hence does not qualify as metric. It is an open question to what extent a suitable metric could increase the quality of discovered shapelets both in classification algorithms and in statistical shapelet mining.

### Significant Shapelet Mining for Sepsis

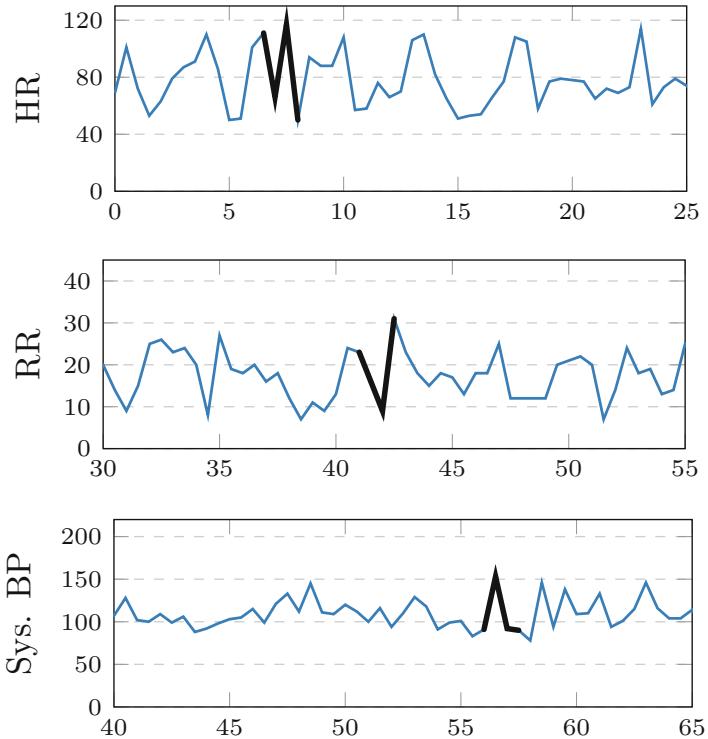
Bock et al. applied significant shapelet mining to identify shapelets that are statistically associated with sepsis from the MIMIC-III data set. Three vital parameters that are routinely measured during a patient's ICU stay and play an important role in the assessment of sepsis-related organ failure were analyzed: heart rate (HR), respiratory rate (RR), and systolic blood pressure (Sys. BP).

Figure 7 shows the three most significant (i.e., lowest  $p$ -value) subsequences that are associated with sepsis in black and the time series they originate from in blue. It was observed that a low frequency heart rate variability could be related to sepsis status indicating sudden hemodynamic instability. For respiratory rate, a shapelet was identified that shows a sudden drop followed by a sharp increase. As pointed out by the authors, this observation is independent of the established link between high RR and septic patients and might require further investigation. Similar observations were made for the characteristic spike that was found in the measurements of systolic blood pressure.

## 3.2 Deep Learning for Time Series Classification

Over the last few years, deep learning (DL), a form of nonlinear hierarchical representation learning based on neural networks, has been extremely successful in a myriad of tasks including language translation [60], object recognition [61], or the prediction of acute kidney injury [62], just to name a few.

The aim of this section is to point the reader to important literature and the most relevant concepts for TSC with DL methods. A thorough assessment of DL for TSC has been recently



**Fig. 7** Subsequences that are statistically significantly associated with sepsis.  
Reproduced from [47] with permission from Oxford University Press

published and provides an overview of relevant methods with their advantages and drawbacks [31]. Successful DL approaches include convolutional neural networks (CNN) [63], residual networks [64], and long short-term memory (LSTM) [65] recurrent neural networks (RNN) [66]. CNNs are particularly well-suited for time series as they are able to learn local features. This is conceptually similar to shapelet extraction but allows for learning of higher order interactions by stacking multiple convolutional filters. Furthermore, class activation maps (CAM) [67], initially introduced for CNNs and to localize class-specific image regions, allow us to “open the black box” of deep neural networks and gain an understanding of the regions of a time series that contribute the most to a network’s classification decision. Furthermore, the LSTM based RNN was the first approach that takes excessive long-term dependencies into account while exhibiting much more stable training behavior. Overall, on benchmark data sets like the UCR time series archive [21] the performance gain over methods that do not utilize NNs is reported to be nonsignificant. That being said, an increasing number of real-world problems are being tackled by harnessing the strengths of DL, ranging from ECG risk stratification [68] to audio signal classification [69] and speech recognition [70].

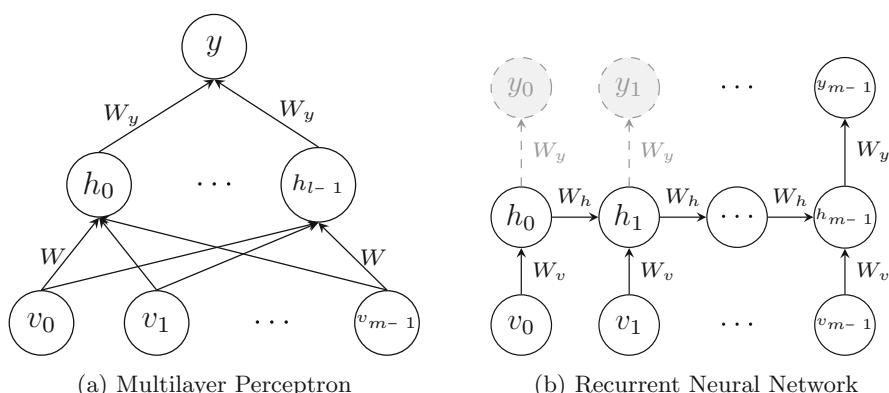
In the following we will briefly introduce relevant neural network approaches suitable for classifying time series data. We will also give a short introduction to the multilayer perceptron as it serves as a basic concept for understanding the more advanced sequencing modeling approaches.

### 3.2.1 Multilayer Perceptron (MLP)

The MLP is the simplest version of a feedforward neural network and has close ties to logistic regression. In fact, an NN without any hidden layers whose output is computed by a logistic function is equivalent to logistic regression. A defining characteristic of DNNs however, are the nonlinearities in their hidden layers. Popular nonlinear functions—also called activation functions—include the sigmoid/logistic function, a rectifier function (used in rectified linear units (ReLU)), the hyperbolic tangent (tanh), or the Swish function [71]. The learning process of an NN is split into two steps, a forward pass and a backward pass. In the forward pass, we apply “learnt” knowledge and predict for example the class membership of a time series. Typically, our NN does not make a *binary* classification decision. Instead, we use a *softmax* function that returns two values which sum up to one and where the second value represents the predicted probability that the input time series belongs to class one. For a multiclass classification task, the forward pass of an NN with one hidden layer, as illustrated in Fig. 8a, can be expressed in the following two equations:

$$b = \sigma(Wv)$$

$$y = \text{softmax}(W_y b),$$



**Fig. 8** (a) Schematic illustration of a simple MLP with one hidden layer and  $l$  hidden neurons. (b) An RNN with self-connected hidden representations for each input value. The architecture of RNNs allows not only the classification of a simple input stream, but also assignment of labels to each individual time step ( $y_0, y_1, \dots$ ). Such configurations can be referred to as “many-to-one” (many inputs with one output) and “many-to-many” (many inputs with many outputs), respectively

with  $v$  being the vector that contains all measurements of a time series,  $\sigma(\cdot)$  being the rectifier function  $\sigma(x) = \max(0, x)$ , and  $W \in \mathbb{R}^{l' \times m}$  and  $W_y \in \mathbb{R}^{2 \times l}$  are weight matrices whose entries will be learnt. The  $m$ -th entry of  $W$  is called bias, which allows learning of a decision hyperplane that does not pass the origin. In order to match  $W$ 's dimensions, we set  $v_m = 1$ . The first dimension of  $W_y$  is due to the fact that we are dealing with a binary classification task and we want  $y$  to be a two-dimensional vector. The “learning” takes place in the backward pass, where we evaluate how far the prediction was from the actual class label, and update the values in  $W$  and  $W_y$  accordingly. This is called *error back propagation* [66] or *reverse mode automatic differentiation* and is the algorithm that made learning in large artificial neural networks possible. However, for brevity, we will not further discuss it here and refer the reader to [72] for an accessible introduction. In brief, the magnitude of the prediction error is measured through a pre-defined loss function (e.g., categorical cross-entropy) which is propagated through the network via the gradient descent algorithm, or extensions thereof. This allows us to update the weight matrices' values according to their contribution to the prediction result.

### 3.2.2 Recurrent Neural Networks (RNNs)

RNNs are an extension of feedforward neural networks which allows the aggregation and storage of information from previous inputs. The core mechanism is a self-connected hidden layer that can make use of the past context by weighing its importance with respect to the learning task (e.g., classification) accordingly [73]. Figure 8b illustrates an “unrolled” RNN which contains a hidden representation  $h_t$  for each input value  $v_t$ . A forward pass to update the different entities can be expressed by the following equations:

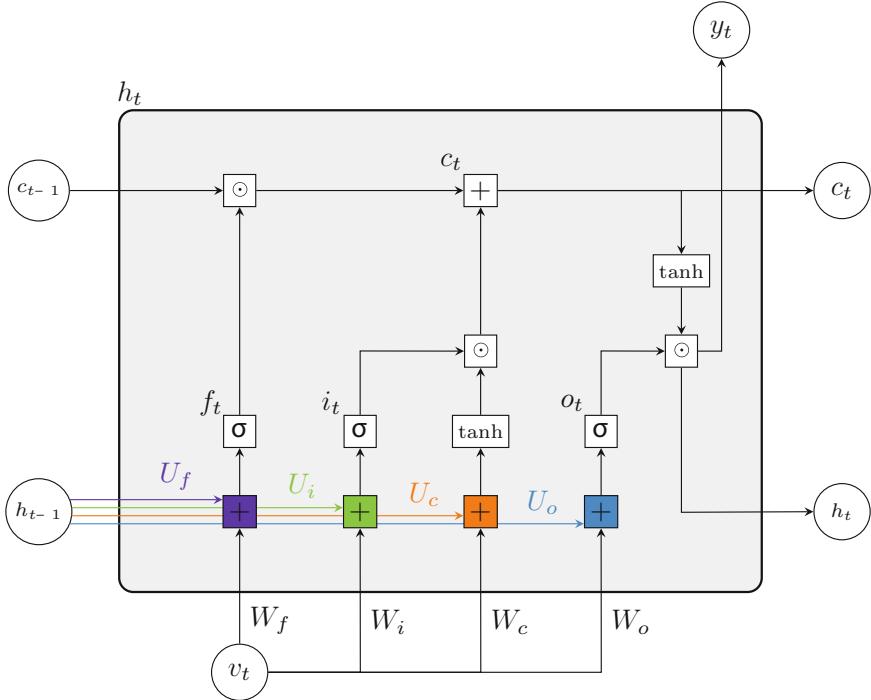
$$\begin{aligned} h_t &= \sigma(W_h h_{t-1} + W_v v_t) \\ y_t &= \text{softmax}(W_y h_t) \end{aligned}$$

The learnable parameters of an RNN are updated with an extension of the standard back-propagation algorithm called back-propagation through time (BPTT) [74].

#### Long Short-Term Memory

One fundamental shortcoming that limits the practicability of RNNs as described above, is the problem of vanishing and exploding gradients that inhibit learning long-term dependencies. We refer the interested reader to [75] and [76] for a thorough analysis of the problem. To circumvent this problem, Hochreiter and Schmidhuber introduced the LSTM unit for RNNs as depicted in Fig. 9.

In brief, the LSTM unit allows for learning of the importance and irrelevance of past and current values through gating mechanisms imposed on linear transformations of the current input  $v_t$  and

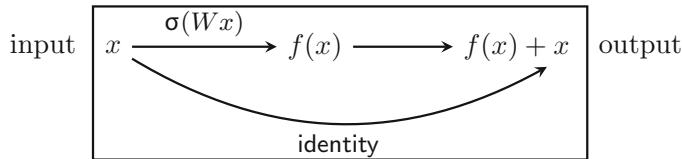


**Fig. 9** An LSTM unit and its mathematical operations to generate a label for time step  $t$ ,  $y_t$  and the inputs for the next LSTM unit ( $c_t$  and  $h_t$ ). The current input value  $v_t$  is combined in multiple ways (indicated by different colors) with the output of the preceding unit.  $f_t$  and  $i_t$  are often referred to as *forget gate* and *input gate*, respectively, as their sigmoid functions “gate” how much the past state and current input can influence the current state of the cell  $c_t$ . The *output gate*  $o_t$  controls to which extend the current cell state influences the next unit.  $\odot$  is the Hadamard or elementwise product

the output of the previous LSTM unit. The following update rules summarize the inner workings of an LSTM unit with  $p$  memory cells in the forward pass:

$$\begin{aligned}
 f_t &= \sigma(W_f v_t + U_f h_{t-1} + b_f) \\
 i_t &= \sigma(W_i v_t + U_i h_{t-1} + b_i) \\
 c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c v_t + U_c h_{t-1} + b_c) \\
 o_t &= \sigma(W_o v_t + U_o h_{t-1} + b_o) \\
 h_t &= o_t \odot \sigma(c_t)
 \end{aligned}$$

Note that a single LSTM unit can contain multiple memory cells by choosing weight matrices and bias vectors with the appropriate dimensions:  $W \in \mathbb{R}^{p \times m}$ ,  $U \in \mathbb{R}^{p \times p}$ , and  $b \in \mathbb{R}^p$ . LSTM based RNNs are omnipresent and have been used to algorithmically generate image captions [77], for machine translation [78], and to detect mitosis in breast cancer histology [79], just to name a few fields. Since their introduction in 1997, LSTMs have been further



**Fig. 10** Illustration of a residual block. The input  $x$  is mapped to  $f(x)$ . The output of the residual block is then the sum  $f(x) + x$  which is similar to an additional connection of  $x$  that “skips”  $f(x)$ . We represent  $f(x)$  as a single layer; however, in practice  $f(x)$  could be replaced with a stack of  $k$  layers  $\sigma(W_1(\dots\sigma(W_kx)))$

developed and a well-established successor is the gated recurrent unit (GRU) [80]. GRUs are a simplification of the LSTM where input, forget, and output gates are dismissed and replaced by a reset and an update gate. A thorough assessment of the advantages and disadvantages of both approaches in sequence modeling is given in [81].

### 3.2.3 Residual Networks (ResNets)

For decades, skip connections between nonadjacent neurons have been discussed in the literature. Reference [64] showed that certain skip connections, *residual* connections, facilitate the training of deep neural networks involving up to 1000 layers, which before that was hard to achieve. Residual connections only apply the identity mapping as opposed to further parametrization.

Residual connections are typically embedded into the so-called residual blocks, which map an input  $x$  to  $f(x) + x$  whereas  $f(x) = \sigma(Wx)$  represents a conventional nonlinear transformation including the weight matrix  $W$  and the activation  $\sigma$ . In principle,  $f(x)$  can also represent a stack of layers instead of just one, indicating that several neurons are skipped (Fig. 10).

Neural networks that include residual blocks are referred to as residual networks, or ResNets [64]. Even though residual networks were originally popularized for challenging computer vision tasks such as Imagenet [82], ResNets have also shown promising results for time series classification [31]. They have also been employed in more involved architectures, such as temporal convolutional networks, which combine residual blocks with causally, dilated convolutions (for more details, refer to subsequent paragraphs).

### 3.2.4 Convolutional Neural Networks (CNNs)

Early forms of convolutional neural networks (CNNs) have been around since the 1980s [83]. Even though CNNs are biologically motivated by local receptive fields of the retina, they are not restricted to computer vision tasks, but were used for time series tasks at an early stage [84]. CNNs achieved a breakthrough by winning the 2012 ImageNet challenge [85]. Their distinguishing feature, *convolutions*, can be seen as a filter sliding over the input data to extract local features (typically 2D filters for images and 1D filters for time series). Furthermore, the hierarchical stacking of convolutional layers together with pooling layers allow for the

extraction of global features. For more details regarding pooling layers, refer to Chapter 9 of [72]. Formally, the convolution operation is defined for two real-valued functions. However, in the machine learning context, typically *discrete* convolutions are used which we are going to introduce. We define the *discrete* convolution of the time series values  $\mathbf{v}$  with the filter  $\mathbf{f}$  of length  $l$  as

$$(\mathbf{v} * \mathbf{f})(k) = \sum_{m=0}^{l-1} v_{k+m} \cdot f_m.$$

Instead of filter, many libraries use the term kernel and refer to its length as “kernel size.” Here, we follow [72] in highlighting an intuitive formalism for convolutions which is frequently used in machine learning libraries. Technically, this refers to cross-correlation, which is the same operation with the filter being flipped. Alternatively, [31] proposed the following *centred* notation for formalizing convolutions:

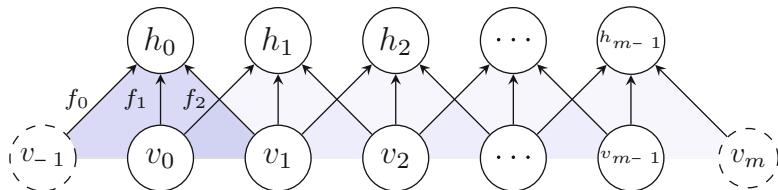
$$(\mathbf{v} * \mathbf{f})(k) = \langle \mathbf{v}_{k-\lfloor l/2 \rfloor : k+\lfloor l/2 \rfloor}, \mathbf{f} \rangle,$$

where  $\langle \cdot, \cdot \rangle$  denotes the dot product, and  $\lfloor \cdot \rfloor$  the floor operation (which we added to ensure whole numbered fractions).

For time series, they symmetrically process values from the past and the future. A disadvantage of the *centred* notation is that it is not clearly defined for filters of even length.

In Fig. 11, a convolutional layer using a 1D filter is displayed. The dashed neurons at both ends of the time series represent padded zeros (here, the padding equals 1) for controlling the convolution’s output shape. After computing the hidden unit  $h_0$  the filter strides one neuron to the right. In general, the *stride* of a convolution is a hyperparameter and for values greater than one implies that the filter is not *sliding* but *jumping* the neurons. For an input time series of length  $m$ , a convolution with a filter of length  $l$ , padding  $p$ , and stride  $s$  results in an output of length  $\lfloor ((m + 2p - l)/s) + 1 \rfloor$ .

In practice, convolutional layers are stacked and in each convolutional layer multiple filters are used, which for each filter results



**Fig. 11** Visualization of a 1D convolutional layer with a filter of length 3 and stride 1. The arrows and the blue triangles indicate how the filter  $\mathbf{f}$  convolves over the time series values  $v_t$ . At both margins, dashed nodes represent “padded zeros” which are dummy neurons used to control the shape of the convolution output. The blue triangles indicate that the same filter  $\mathbf{f}$  slides over the time series, which for a small enough number of filters drastically reduces the number of parameters as compared to an MLP

in a feature map. For classification, CNNs conventionally involve a stack of convolutional (and pooling) layers as followed by few fully connected layers for mapping feature maps to an output layer containing one neuron per class [85]. The final output can be normalized, such that the output scores represent probabilities of class membership. However, replacing the last fully connected (or dense) layers with convolutional ones in the so-called fully convolutional networks has been shown to speed up learning while allowing for interpretability (e.g., in the form of image segmentations) [86].

### 3.2.5 Temporal Convolutional Networks (TCNs)

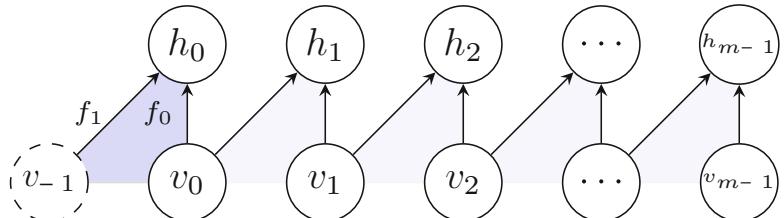
Applying convolutions to time series has been a popular extension to CNNs, temporal convolutional networks (TCNs) [87]. The authors of [88] observed three aspects which distinguish TCNs from generic CNNs:

1. *Sequence to sequence*: The output of a TCN has the same length as its input.
2. *Causal convolutions*: The convolution operation only uses data from the present and the past.
3. *Long effective memory*: TCNs employed dilated convolutions which, with increasing network depth, allows for an exponentially growing receptive field (and thus effective memory).

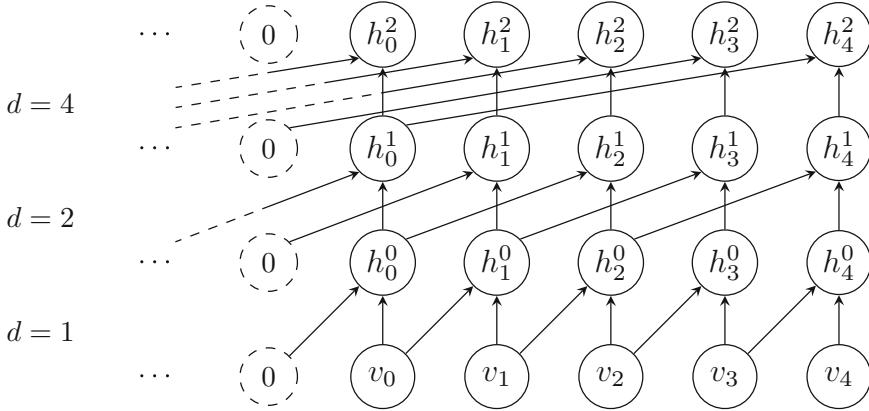
In an empirical study benchmarking TCNs against recurrent architectures in various sequence modeling tasks, they yielded compelling results, leaving the authors to conclude that “the pre-eminence enjoyed by recurrent networks in sequence modeling may be largely a vestige of history” [88].

Interestingly, dilated convolutions for time series have been previously used, most prominently in [89] and [90], while the earliest predecessor, time-delay neural networks, was proposed in the late 1980s [91].

Next, we introduce *causal* (Fig. 12) and *dilated* convolutions (Fig. 13). For this, we follow the notation as described in [89]. A



**Fig. 12** Visualization of a 1D *causal* convolutional layer with filter length 2. We define a causal, discrete convolution to be  $d$ -dilated with  $(\mathbf{v} *_d \mathbf{f})(k) = \sum_{j=i+d_j}^k v_i \cdot f_j$  for  $d \in \mathbb{N}_{>0}$ . For  $d = 1$ , this coincides with the standard causal convolution



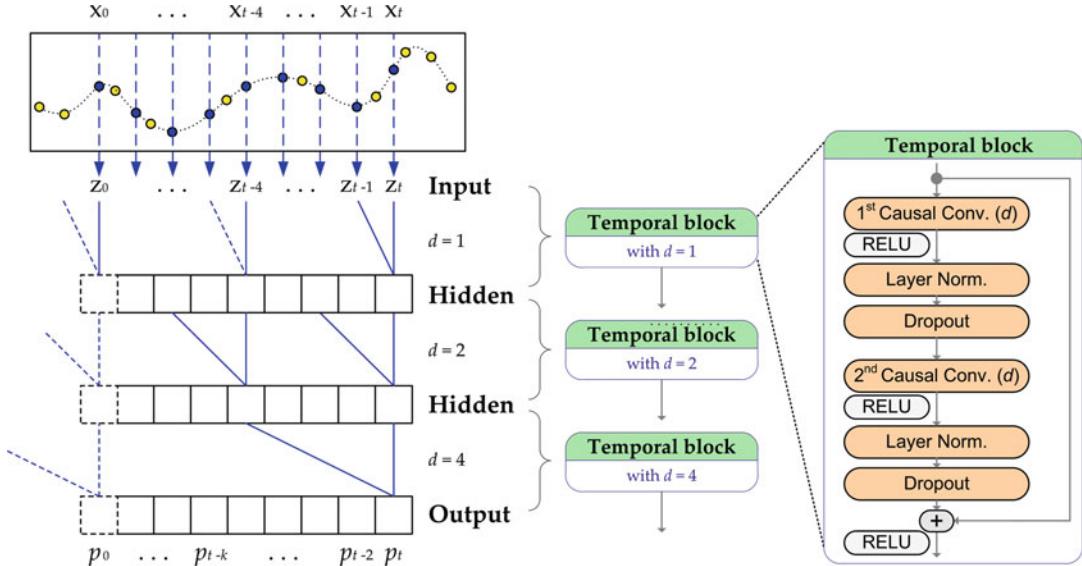
**Fig. 13** Visualization of dilated convolutions. For each hidden unit, we indicate the layer in the exponent. The dilations are stacked in an exponential manner, such that the dilation of layer  $k$  is  $2^k$ , allowing for large receptive fields. For illustrative purposes, we highlight one padded zero per layer and refrain from displaying more as this number grows exponentially with network depth

discrete convolution is *causal* if the convolution was computed solely with input data from the present and past—but not from the future. We define a causal convolution of the time series values  $\mathbf{v}$  with the filter  $\mathbf{f}$  as

$$(\mathbf{v} * \mathbf{f})(k) = \sum_{j=i+j} v_i \cdot f_j$$

Note that this notation [89] uses a formal convolution as the filter  $\mathbf{f}$  is flipped along the time axis. The  $k$ -th entry of the convolution refers to the dot product of the filter  $\mathbf{f}$  with the subsequence of  $\mathbf{v}$  defined by the indices that, together with the indices of  $\mathbf{f}$ , pairwise sum up to  $k$ .

In Fig. 13, we schematically display dilated convolutions. This is a mere visualization of stacked convolutional layers. In practice, TCNs are equipped with additional operations, such as nonlinearities in the form of ReLU activations. To speed up training, TCNs employ residual connections and normalization layers (weight normalization [92] or layer normalization [93]). Additionally, to avoid overfitting *dropout* is used: during training a randomly chosen fraction of neurons are “switched off.” This forces the network not to rely too much on single neurons but to extract relevant features via multiple paths along the neurons. TCNs are structured in stacked temporal blocks which gather the above operations, for more details, refer to Fig. 14.



**Fig. 14** TCN Illustration reused from [34] under retained copyright of the author

### Temporal Convolutional Networks for Sepsis Prediction

In this application box, we highlight a recent application of TCNs for predicting sepsis [34]. To this end, we highlight a TCN architecture in more detail and illustrate how they can be applied to medical time series data. In the figure on the right, we display the TCN architecture described in [34]. After preprocessing, the network is fed a regularly spaced time series, that is, a time series sampled at equal time intervals. Here, the authors notate the time series values with  $z_t$ . The TCN consists of a stack of temporal blocks. Each block combines a residual connection (indicated with the detouring arrow on the right) with two dilated convolutional layers. The authors additionally employ layer normalization which helps speed up the training.

Notably, the exact architecture specifications have varied in the literature. Not only have authors switched between different normalization approaches (weight normalization versus layer normalization), they also modified the dilation of convolutions. [87] propose to dilate the convolutions exponentially *within* one temporal block, whereas [88] and [34] only apply the dilation scheme over a stack of temporal blocks, such that (as in the figure indicated) each temporal block employs one dilation value  $d$ .

Our discussed TCN architecture has been set up in two ways [34]. First, the authors applied carry-forward imputation to fill missing values and to arrive at a regularly spaced time series to feed this data to a standard TCN. Second, they employed a more sophisticated method, a multitask gaussian process (MGP) [94], to impute the missing values to then feed the retrieved time series to the TCN. With regard to predicting sepsis, the performance of both approaches is displayed in the Fig. 5 under the names TCN and MGP-TCN. This indicates that more elaborate imputation schemes can have a significant impact on the performance of subsequent neural network classifiers. As an addendum for the technically adept reader, the MGP-TCN method has been trained end-to-end by making use of the gaussian process adapter framework [95], allowing for the parameters of the MGP and the TCN to be trained jointly with backpropagation.

### 3.3 Tools

We end by providing a list (*see* Table 2) of software packages and libraries that might be of interest to the reader. This list is not exhaustive but proved useful for a wide range of applications in the realm of time series analysis.

---

## 4 Summary

Over recent decades, the biomedical field has generated unparalleled amounts of temporal data that offer unique possibilities for time series mining. Exploitation of these data remains a challenge, particularly because time series data require specialized analysis techniques and tools. One major goal of time series analysis is the classification of time series or subsequences based on their behavior over time. In this chapter, we first introduced the reader to common distance-based time series classification methods, namely dynamic time warping in combination with nearest neighbors as well as time series subsequence mining (i.e., shapelets). In addition to the methodological details, we highlighted the strengths and weaknesses of these methods by applying them to a real-world biomedical problem—detection of sepsis onset. Some steps of the conventional time series classification methods can be quite labor-intensive (e.g., manual feature engineering), particularly with large datasets. With the advent of deep learning models for time series classification, many steps can be automated. In addition, we discussed the methodological concepts of three deep learning models: recurrent neural networks, convolutional neural networks, and temporal convolutional networks. In summary, the chapter provides a comprehensive introduction to common distance-based and

**Table 2**  
**Useful software packages and libraries for time series analysis**

Name	Link	Primary tasks	Implementation
sktime	<a href="https://github.com/alan-turing-institute/sktime">https://github.com/alan-turing-institute/sktime</a>	Currently forecasting and classification	Python
tslearn	<a href="https://github.com/rtavenar/tslearn">https://github.com/rtavenar/tslearn</a>	Clustering, DTW calculation, classification	Python
dtaidistance	<a href="https://github.com/wannesm/dtaidistance">https://github.com/wannesm/dtaidistance</a>	Fast DTW computation	Python
matrixprofile-ts	<a href="https://github.com/target/matrixprofile-ts">https://github.com/target/matrixprofile-ts</a>	Computation of the matrix profile. Used for a wide range of time series tasks	Python
statsmodels	<a href="https://www.statsmodels.org">https://www.statsmodels.org</a>	Forecasting	Python
forecast	<a href="https://cran.r-project.org/web/packages/forecast/index.html">https://cran.r-project.org/web/packages/forecast/index.html</a>	Forecasting	R
S3M [47]	<a href="https://github.com/BorgwardtLab/S3M">https://github.com/BorgwardtLab/S3M</a>	Significant Subsequence Mining	Command line tool
Supplementary material of [30]	<a href="http://www.timeseriesclassification.com/code.php">http://www.timeseriesclassification.com/code.php</a>	Classification	Java
dl-t-tsc	<a href="https://github.com/hfawaz/dl-4-tsc">https://github.com/hfawaz/dl-4-tsc</a>	Classification	Python
tensorflow	<a href="https://www.tensorflow.org">https://www.tensorflow.org</a>	Deep Learning Framework	Python, C++, JavaScript, Java, Go, Swift
pytorch	<a href="https://pytorch.org/">https://pytorch.org/</a>	Deep Learning Framework	Python, C++
keras	<a href="https://keras.io/">https://keras.io/</a>	Deep Learning Framework	Python
TCN [88]	<a href="https://github.com/locuslab/TCN">https://github.com/locuslab/TCN</a>	Classification	Python, pytorch
MGP-TCN [34]	<a href="https://github.com/BorgwardtLab/mgp-tcn">https://github.com/BorgwardtLab/mgp-tcn</a>	Classification	Python, tensorflow

deep learning models for classification of time series and provides the reader with useful sources to successfully apply these models to real world biomedical data sets.

---

## Acknowledgments

The authors would like to thank Dr. Bastian Rieck, Dr. Damian Roqueiro, and Max Horn for their valuable input and discussion.

## References

1. Sudlow C, Gallacher J, Allen N et al (2015) UK biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS Med* 12: e1001779
2. Johnson AEW, Pollard TJ, Shen L et al (2016) MIMIC-III, a freely accessible critical care database. *Sci Data* 3:160035
3. Miller G (2012) The smartphone psychology manifesto. *Perspect Psychol Sci* 7:221–237
4. Ent MMVX van den, Brown DW, Hoekstra EJ et al (2011) Measles mortality reduction contributes substantially to reduction of all cause mortality among children less than five years of age, 1990–2008. <https://doi.org/10.1093/infdis/jir081>
5. Au-Yong ITH, Thorn N, Ganatra R et al (2009) Brown adipose tissue and seasonal variation in humans. *Diabetes* 58:2583–2587
6. Refinetti R, Menaker M (1992) The circadian rhythm of body temperature. *Physiol Behav* 51:613–637
7. Reed BG, Carr BR (2018) The normal menstrual cycle and the control of ovulation. In: Feingold KR, Anawalt B, Boyce A et al (eds) Endotext. MDText.com, South Dartmouth, MA
8. Nagai S, Anzai D, Wang J (2017) Motion artefact removals for wearable ECG using stationary wavelet transform. *Healthc Technol Lett* 4:138–141
9. Durbin J, Watson GS (1950) Testing for serial correlation in least squares regression. I. *Biometrika* 37:409–428
10. Bence JR (1995) Analysis of short time series: correcting for autocorrelation. *Ecology* 76:628–639
11. Peña D, Tiao GC, Tsay RS (2011) A course in time series analysis. Wiley, New York
12. Kurbalija V, Radovanović M, Geler Z et al (2010) A framework for time-series analysis. In: Artificial intelligence: methodology, systems, and applications. Springer, Berlin, pp 42–51
13. Warren Liao T (2005) Clustering of time series data—a survey. *Pattern Recognit* 38:1857–1874
14. Malhotra P, Vig L, Shroff G et al (2015) Long short term memory networks for anomaly detection in time series. In: Proceedings. Presses universitaires de Louvain, p 89
15. De Gooijer JG (2017) Elements of nonlinear time series analysis and forecasting. Springer, Cham
16. Kirchgässner G, Wolters J (2008) Introduction to modern time series analysis. Springer Science & Business Media, Berlin
17. Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, NY, pp 947–956
18. Zhu Y, Imamura M, Nikovski D et al (2018) Time series chains: a novel tool for time series data mining. <https://doi.org/10.24963/ijcai.2018/764>
19. Yeh CM, Zhu Y, Ulanova L et al (2016) Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: 2016 IEEE 16th international conference on data mining (ICDM). pp 1317–1322
20. Celebi ME, Aydin K (eds) (2016) Unsupervised learning algorithms. Springer, Cham
21. Dau HA, Bagnall A, Kamgar K et al (2018) The UCR time series archive. <http://arxiv.org/abs/1810.07758>
22. Che Z, Purushotham S, Cho K et al (2018) Recurrent neural networks for multivariate time series with missing values. *Sci Rep* 8:6085
23. Singer M, Deutschman CS, Seymour CW et al (2016) The third international consensus definitions for sepsis and septic shock (Sepsis-3). *JAMA* 315:801–810
24. Chevyrev I and Kormilitzin A (2016). A primer on the signature method in machine learning. <http://arxiv.org/abs/1603.03788>
25. Aggarwal CC (2015) Data mining: the textbook. Springer, New York
26. Rizzo R, Fiannaca A, La Rosa M et al (2016) A deep learning approach to DNA sequence classification. In: Computational intelligence methods for bioinformatics and biostatistics. Springer, New York
27. Kadous MW, Sammut C (2005) Classification of multivariate time series and structured data using constructive induction. *Mach Learn* 58:179–216
28. Vaswani A, Shazeer N, Parmar N et al (2017) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S et al (eds) Advances in neural information processing systems 30. Curran Associates, Red Hook, pp 5998–6008
29. Harutyunyan H, Khachatrian H, Kale DC et al (2019) Multitask learning and benchmarking with clinical time series data. <https://doi.org/10.1038/s41597-019-0103-9>

30. Bagnall A, Lines J, Bostrom A et al (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. <https://doi.org/10.1007/s10618-016-0483-9>
31. Ismail Fawaz H, Forestier G, Weber J et al (2019) Deep learning for time series classification: a review. *Data Min Knowl Discov* 33:917–963
32. Futoma J, Hariharan S, Heller K (2017) Learning to detect sepsis with a multitask Gaussian process RNN classifier, In: Proceedings of the 34th international conference on machine learning—volume 70. JMLR.org, Sydney, NSW, pp 1174–1182
33. Calvert JS, Price DA, Chettipally UK et al (2016) A computational approach to early sepsis detection. *Comput Biol Med* 74:69–73
34. Moor M, Horn M, Rieck B et al (2019) Early recognition of sepsis with Gaussian process temporal convolutional networks and dynamic time warping.
35. Futoma J, Hariharan S, Sendak M et al (2017) An improved multi-output Gaussian process RNN with real-time validation for early sepsis detection. <http://arxiv.org/abs/1708.05894>
36. Ferrer R, Martin-Lloeches I, Phillips G et al (2014) Empiric antibiotic treatment reduces mortality in severe sepsis and septic shock from the first hour: results from a guideline-based performance improvement program. *Crit Care Med* 42:1749–1755
37. Shimabukuro DW, Barton CW, Feldman MD et al (2017) Effect of a machine learning-based severe sepsis prediction algorithm on patient survival and hospital length of stay: a randomised clinical trial. *BMJ Open Respir Res* 4: e000234
38. Desautels T, Calvert J, Hoffman J et al (2016) Prediction of sepsis in the intensive care unit with minimal electronic health record data: a machine learning approach. *JMIR Med Inform* 4:e28
39. Reyna M, Josef C, Jeter R et al (2019) Early prediction of sepsis from clinical data: the PhysioNet/computing in cardiology challenge 2019. *Crit Care Med*
40. Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans Acoust* 26:43–49
41. Xi X, Keogh E, Shelton C et al (2006) Fast time series classification using numerosity reduction. In: Proceedings of the 23rd international conference on machine learning. ACM, New York, NY, pp 1033–1040
42. Dau HA, Silva DF, Petitjean F et al (2018) Optimizing dynamic time warping's window width for time series data mining applications. <https://doi.org/10.1007/s10618-018-0565-y>
43. Hastie T, Tibshirani R, Friedman J et al (2005) The elements of statistical learning: data mining, inference and prediction. *Math Intelligencer* 27:83–85
44. Bishop CM (2006) Pattern recognition and machine learning. Springer, New York
45. Ghalwash MF, Obradovic Z (2012) Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC Bioinformatics* 13:195
46. Ghalwash M, Radosavljevic V, Obradovic Z (2013) Early diagnosis and its benefits in sepsis blood purification treatment. In: 2013 IEEE international conference on healthcare informatics. pp 523–528
47. Bock C, Gumbisch T, Moor M et al (2018) Association mapping in biomedical time series via statistically significant shapelet mining. *Bioinformatics* 34:i438–i446
48. Xu J, Zhang Y, Zhang P et al (2017) Data mining on icu mortality prediction using early temporal data: a survey. *Int J Inf Technol Decis Mak* 16:117–159
49. Shanjina T, Sivakumar PB (2012) Human gait recognition and classification using time series shapelets. In: 2012 international conference on advances in computing and communications. pp 31–34
50. Shannon CE, Weaver W (1949) The mathematical theory of communication. University of Illinois Press, Champaign
51. Rakthanmanon T, Keogh E (2011) Fast-shapelets: a fast algorithm for discovering robust time series shapelets. In: Proceedings of 11th SIAM international conference on data mining,
52. Grabocka J, Schilling N, Wistuba M et al (2014) Learning time-series shapelets. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, NY, pp 392–401
53. Hills J, Lines J, Baranauskas E et al (2014) Classification of time series by shapelet transformation. *Data Min Knowl Discov* 28:851–881
54. Dudoit S, van der Laan MJ (2007) Multiple testing procedures with applications to genomics. Springer Science & Business Media, Berlin
55. Llinares-Lopez F, Borgwardt K (2019) Machine learning for biomarker discovery: significant pattern mining. In: Pržulj N (ed) Analyzing network data in biology and

- medicine: an interdisciplinary textbook for biological, medical and computational scientists. Cambridge University Press, Cambridge, pp 313–368
56. Fisher RA (1922) On the interpretation of  $\chi^2$  from contingency tables, and the calculation of P. *J R Stat Soc* 85:87–94
  57. Bonferroni CE (1936) Teoria statistica delle classi e calcolo delle probabilità. Libreria internazionale Seeber, Firenze
  58. Tarone RE (1990) A modified Bonferroni method for discrete data. *Biometrics* 46:515–522
  59. Terada A, Okada-Hatakeyama M, Tsuda K et al (2013) Statistical significance of combinatorial regulations. *Proc Natl Acad Sci U S A* 110:12996–13001
  60. Devlin J, Chang M-W, Lee K et al (2018) BERT: pre-training of deep bidirectional transformers for language understanding. <http://arxiv.org/abs/1810.04805>
  61. Girshick R, Donahue J, Darrell T et al (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 580–587
  62. Tomašev N, Glorot X, Rae JW et al (2019) A clinically applicable approach to continuous prediction of future acute kidney injury. *Nature* 572:116–119
  63. LeCun Y, Bottou L, Bengio Y et al (1998) Gradient-based learning applied to document recognition. <https://doi.org/10.1109/5.726791>
  64. He K, Zhang X, Ren S et al (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 770–778
  65. Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Comput* 9:1735–1780
  66. Rumelhart DE, Hinton GE, Williams RJ et al (1988) Learning representations by back-propagating errors. *Cogn Model* 5:1
  67. Zhou B, Khosla A, Lapedriza A et al (2016) Learning deep features for discriminative localization. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 2921–2929
  68. Shanmugam D, Blalock D, Guttag J (2018) Multiple instance learning for ECG risk stratification. <http://arxiv.org/abs/1812.00475>
  69. Brueckner R, Schulter B (2014) Social signal classification using deep blstm recurrent neural networks. In 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp 4823–4827
  70. Xiong W, Wu L, Alleva F et al (2018) The Microsoft 2017 conversational speech recognition system. In 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp 5934–5938
  71. Ramachandran P, Zoph B, Le QV (2017) Searching for activation functions. <http://arxiv.org/abs/1710.05941>
  72. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
  73. Graves A, Liwicki M, Fernández S et al (2009) A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans Pattern Anal Mach Intell* 31:855–868
  74. <https://doi.org/10.21236/ada164453>
  75. Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient descent is difficult. *IEEE Trans Neural Netw* 5:157–166
  76. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: International conference on machine learning. pp 1310–1318
  77. Vinyals O, Toshev A, Bengio S et al (2015) Show and tell: a neural image caption generator, In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 3156–3164
  78. Wu Y, Schuster M, Chen Z et al (2016) Google’s neural machine translation system: bridging the gap between human and machine translation. <http://arxiv.org/abs/1609.08144>
  79. Cireşan DC, Giusti A, Gambardella LM et al (2013) Mitosis detection in breast cancer histology images with deep neural networks. *Med Image Comput Comput Assist Interv* 16:411–418
  80. Cho K, Merrienboer B van, Gulcehre C et al (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. <https://doi.org/10.3115/v1/d14-1179>
  81. Chung J, Gulcehre C, Cho K et al (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. <http://arxiv.org/abs/1412.3555>
  82. Russakovsky O, Deng J, Su H et al (2015) ImageNet large scale visual recognition challenge. *Int J Comput Vis* 115:211–252
  83. Fukushima K (1980) Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol Cybern* 36:193–202

84. LeCun Y, Bengio Y et al (1995) Convolutional networks for images, speech, and time series. 3361:1995
85. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L et al (eds) Advances in neural information processing systems 25. Curran Associates, Red Hook, pp 1097–1105
86. Long J, Shelhamer E, Darrell T (2015) Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440
87. Lea C, Flynn MD, Vidal R et al (2017) Temporal convolutional networks for action segmentation and detection. In: proceedings of the IEEE conference on computer vision and pattern recognition. pp 156–165
88. Bai S, Zico Kolter J, Koltun V (2018) An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. <http://arxiv.org/abs/1803.01271>
89. Yu F, Koltun V (2015) Multi-scale context aggregation by dilated convolutions. <http://arxiv.org/abs/1511.07122>
90. Oord A van den, Dieleman S, Zen H et al (2016) WaveNet: a generative model for raw audio. <http://arxiv.org/abs/1609.03499>
91. Waibel A, Hanazawa T, Hinton G et al (1989) Phoneme recognition using time-delay neural networks. IEEE Trans Acoust 37:328–339
92. Salimans T, Kingma DP (2016) Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In: Advances in neural information processing systems. pp 901–909
93. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. <http://arxiv.org/abs/1607.06450>
94. Bonilla EV, Chai KM, Williams C (2008) Multi-task Gaussian process prediction. In: Platt JC, Koller D, Singer Y et al (eds) Advances in neural information processing systems 20. Curran Associates, Red Hook, pp 153–160
95. Li SC-X, Marlin BM (2016) A scalable end-to-end Gaussian process adapter for irregularly sampled time series classification. In: Lee DD, Sugiyama M, Luxburg UV et al (eds) Advances in neural information processing systems 29. Curran Associates, Red Hook, pp 1804–1812



# Chapter 3

## Siamese Neural Networks: An Overview

Davide Chicco

### Abstract

Similarity has always been a key aspect in computer science and statistics. Any time two element vectors are compared, many different similarity approaches can be used, depending on the final goal of the comparison (Euclidean distance, Pearson correlation coefficient, Spearman's rank correlation coefficient, and others). But if the comparison has to be applied to more complex data samples, with features having different dimensionality and types which might need compression before processing, these measures would be unsuitable. In these cases, a *siamese neural network* may be the best choice: it consists of two identical artificial neural networks each capable of learning the hidden representation of an input vector. The two neural networks are both feedforward perceptrons, and employ error back-propagation during training; they work parallelly in tandem and compare their outputs at the end, usually through a cosine distance. The output generated by a siamese neural network execution can be considered the semantic similarity between the projected representation of the two input vectors. In this overview we first describe the siamese neural network architecture, and then we outline its main applications in a number of computational fields since its appearance in 1994. Additionally, we list the programming languages, software packages, tutorials, and guides that can be practically used by readers to implement this powerful machine learning model.

**Key words** Siamese neural networks, Artificial neural networks, Semantic similarity, Neural networks, Deep learning, Siamese networks, Overview, Review, Survey

---

### 1 Introduction

Since the dawn of computer science, researchers have looked for statistical tools to compare two lists of elements, in a purely mathematical or semantic way. For this purpose, scientists designed and developed several similarity rates and coefficients in the past, making them suited for different meanings in different contexts.

For a purely statistical comparison of two elements, metrics such as Euclidean distance, cosine distance, or Manhattan distance [1] could be the best choice, because they highlight the geometric differences between two elements. If the comparison is focused on the correlation, Pearson correlation coefficient [2], Spearman's  $\rho$  rank coefficient [3, 4], Kendall  $\tau$  distance [4, 5], and Goodman–Kruskal's  $\gamma$  rank correlation [6] may be appropriate tools to show

any rank correspondence between two vectors of elements. If the comparison is between two elements structured as an ontology tree, Resnik [7], Jian [8, 9], and Lin [10] measures are particularly useful, because they value the two structure sub-tree of the two elements compared and compute scores based on them. These semantic similarity measures have been employed for several tasks in the past, for example, the comparison between lists of Gene Ontology (GO) annotations in bioinformatics [11, 12]. For information retrieval semantic similarity, scientists have also proposed more complex unsupervised methods, such as latent semantic indexing (LSI) [13, 14].

All these rates and measures work well when used to compare two lists of elements having the same meanings and data types. However, they cannot work in cases where two lists contain elements having different data types and/or different meanings, which relate to different features. For example, if we needed to compare the data of handwritten signatures collected through a pad device, to check if the two signatures come from the same person, the aforementioned measures would be unsuitable. The pad records data coming from different features: speed of the hand signing, position of the characters, dimension of the characters, total space occupied by the signature, distance between the characters, and others. In this case, a structured method able to process and project all the different features into a unique hidden representation would be necessary for the desired scope. In addition, a learning phase where the computational system would learn how to distinguish correct signatures from forgeries would be necessary. Both these tasks can be achieved by machine learning [15]. Precisely to solve this just-described forgery recognition problem, Bromley et al. [16] introduced the *siamese neural network* in 1994.

Since its appearance, this model has been employed in several different fields, and has become widespread in computer science and information technology applications. This overview summarizes the studies which have employed the siamese neural network so far [17], and describes the available resources about it. To the best of our knowledge, no overview or survey about siamese neural networks exists in the scientific literature to date.

We organize the rest of the chapter as follows. After this Introduction, we first describe the model training algorithm and architecture (Subheading 2), and then report some theoretical studies and the main applications in several fields (Subheading 3). We then report software packages, some tutorials, and guides to help readers program their own siamese neural network application script (Subheading 4), and finally we outline some conclusions and future directions (Subheading 5).

---

## 2 Siamese Neural Network: The Architecture

The siamese neural network algorithm was first introduced by Bromley et al. [16] to detect forged signatures in 1994. Before that, Baldi and Chauvin [18] introduced a similar artificial neural network able to recognize fingerprints, though by a different name. In the study by Bromley et al. [16], by comparing two handwritten signatures, this siamese neural network was able to state if the two signatures were both original or if one was a forgery.

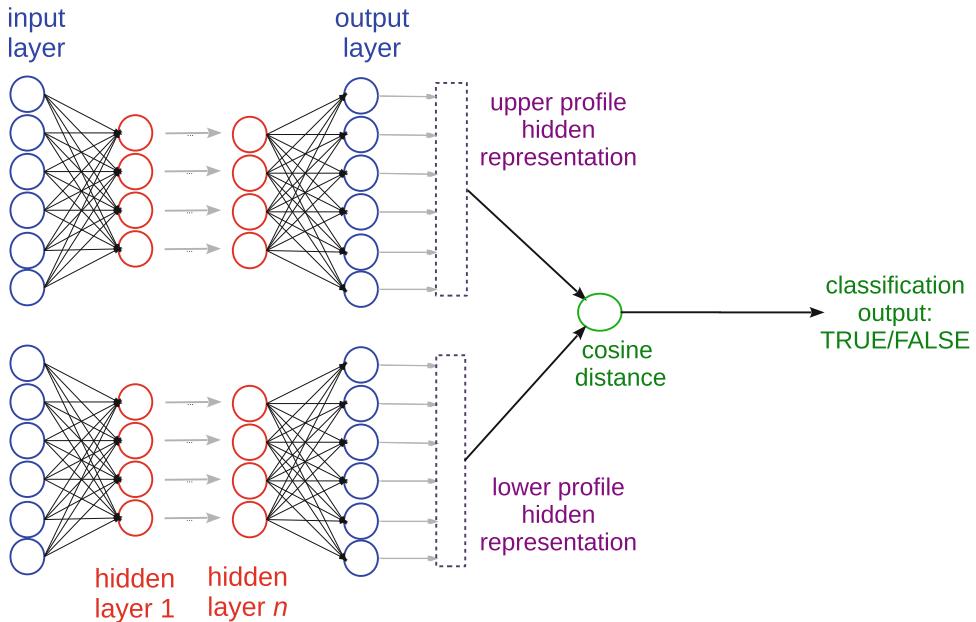
An artificial neural network is a machine learning model made of several layers of neurons, which are processing units inspired by biological neurons [19–21]. Typical supervised neural networks are called *feedforward*, and are based on the perceptron model [22, 23]. In a feedforward neural network, each neuron of the first layer reads in an input real value, multiplies it by a weight, and sends the result to all the neurons in the following layer. In common neural network representations, scientists consider the left-most layer as the first layer and the right-most layer as the output layer (Fig. 1). The neurons of each layer beyond the input layer do the same job and send the results to the neurons of the next layer, until the final layer (the process goes from left to right). In a supervised neural network, the final layer then sends its results to a one-neuron output layer that produces the neural network real-valued result (Fig. 1).

During training, the neural network compares the values produced by the neural network with its corresponding ground truth, and computes the statistical error (usually the mean square error or the cross-entropy error). Afterwards, the neural network sends the error back to the previous layers and updates its neuron weights accordingly, through a technique called *error back-propagation* [24]. In representation terms, back-propagation happens from right to left (Fig. 1).

The training stops when the neural network reaches the maximum number of iterations initially set. Once the model is trained, it can then be applied to the test set: the neural network will process each test data instance (only once, from the first layer on the left to the last layer on the right) and will generate a predicted value. Once the neural network has generated a predicted value for each test set instance, the programmer can use it to compute a confusion matrix.

To evaluate a confusion matrix, we suggest using the Matthews correlation coefficient (MCC) or the precision–recall (PR) curve rather than potentially misleading rates such as accuracy,  $F_1$  score, or receiver operating characteristic (ROC) curve [15, 25, 26].

Computer scientists have used the just-described feedforward neural network architecture with back-propagation in thousands of studies in the past, especially for binary classification and prediction



**Fig. 1** Representation of the structure of the siamese neural network model. The data are processed from left to right. The value of the cosine distance is a measure of the similarity between the input pair of data instances, as final output

(for example, to predict patient outcome and prognosis in biomedical settings [27–29]).

Feedforward neural networks with error back-propagation are employed in siamese neural networks, as well. The siamese neural network architecture, in fact, contains two identical feedforward neural networks joined at their output (Fig. 1), which work parallelly in tandem. Each neural network contains a traditional perceptron model [22]. During training, each neural network reads a profile made of real values, and processes its values at each layer. The neural network activates some of the neurons based upon these values, updates its weights through error back-propagation [24], and at the end it generates an output profile that is compared with the output of the other neural network.

The algorithm compares the output of the upper neural network and the output of the lower neural network through a distance metric (Fig. 1): a cosine distance in the original model [16].

Through this similarity measure, the neural network states that the two profiles are different (cosine similarity value in the  $[-1, 0]$  interval) or similar (cosine similarity value in the  $[0, +1]$  range). The algorithm then labels the data instance as *positive* if in the former case, or as *negative* if in the latter case. The final output value can finally be compared with its corresponding ground truth value; all the classified outputs can be employed to generate a confusion matrix.

---

### 3 Applications and Theoretical Studies

In this section, we list and describe a variety of applications and theoretical studies of the siamese neural network in different computational fields. We looked for publications containing the keywords “siamese neural network(s)” and “siamese network(s)” in the literature search engines Google Scholar [30], Scopus [31], Digital Bibliography & Library Project (DBLP) [32], and PubMed [33].

In our literature search, we gave priority to scientific papers published in peer-reviewed journals over conference proceedings, book chapters, and preprints. Following the advice of Ernst [34], we believe that journal publications allow authors to describe their methods and applications in greater details than conference proceedings, where papers usually need to be concise instead. While we prioritized journal articles, we included around forty references to conference proceedings’ papers in this overview as well.

We report applications from the following fields:

- audio and speech signal processing (Subheading 3.1);
- biology (Subheading 3.2);
- chemistry and pharmacology (Subheading 3.3);
- geometry and graphics (Subheading 3.4);
- image analysis (Subheading 3.5);
- medicine and health (Subheading 3.6);
- optics and physics (Subheading 3.7);
- robotics (Subheading 3.8);
- sensor-based activity recognition (Subheading 3.9);
- software development (Subheading 3.10);
- text mining (Subheading 3.11);
- video analysis (Subheading 3.12).

We conclude this section by describing computer science theory studies (Subheading 3.13).

#### 3.1 Audio and Speech Signal Processing

Siamese neural networks have been employed in several applications in the audio and speech signal processing field.

Tholliere et al. [35], for example, merged a dynamic-time warping based spoken term discovery (STD) system with a siamese deep neural network for automatic discovery of linguistic units from raw speech. The authors used their system to recognize phonetic terms and tested it on two datasets: Buckeye Corpus of conversational American English [36] and the NCHLT Speech Corpus of South African Languages [37], both containing several hours of speech from different speakers.

In another article, Chen et al. [38] used a regularized deep siamese neural network to extract speaker-specific information from a spectral representation called Mel Frequency Cepstral Coefficients (MFCCs). Their article reports several mathematical details about the neural network employed, and describes the application of their model to two linguistic datasets (one of the English conversations and one of the Chinese conversations).

Manocha et al. [39] took advantage of a siamese model to detect all the semantically similar audio clips in an input audio recording. The authors encoded the audio signal into a vector representation using a siamese neural network, which encodes files belonging to the same audio class in a similar way.

Zhang et al. [40] used a siamese convolutional neural network to assess vocal imitations. Their model extracts features from vocal imitations and from original sounds, and computes the similarity between them. The authors applied their model to the VocalSketch Data Set [41].

Švec et al. [42] employed a siamese recurrent neural network for spoken term detection (STD). The authors utilized two jointly-trained siamese neural networks to project the representation of a searched term into a pronunciation embedding space. They then computed the relative distance between these embeddings as a similarity score. The authors tested their method on a spoken dataset of English words and on another of Czech words.

Shon et al. [43] employed the model to recognize Arabic dialects from Arabic speech content found in media broadcasts. They created an Arabic dialect identification system for the 2017 Multi-Genre Broadcast challenge (MGB-3) and showed its effectiveness at this task.

Gündoğdu et al. [44] used a siamese neural network for keyword search within spoken content retrieval. Their model tried to minimize the problem of the out of vocabulary (OOV) terms. The authors applied their method to several datasets: IARPA Babel Program's Turkish, Pashto, and Zulu.

Siddhant et al. [45] employed a deep siamese neural network for accent identification from speech signals. They tested their model on the CSLU Foreign Accented English (FAE) corpus.

Zeghidour et al. [46] developed a “triamese” neural network, a three-piece variant of the traditional siamese neural network, to detect phonetic similarities between speech signals. They showed the effectiveness of their model by applying it to the LibriSpeech corpus.

### 3.2 Biology

Siamese neural networks have been employed in several biological applications too.

Zheng et al. [47] used this model to develop an approach called *SiamEse Neural network for Sequence Embedding (SENSE)* for the comparison of alignment-free DNA sequences. The authors

implemented the  $k$ -mer method [48] through a siamese convolutional neural network, and applied it to the Qiita and RT988 datasets [49].

Jindal et al. [50] proposed the usage of siamese neural networks to classify chromosomes and for karyotyping. They took advantage of this model to predict whether a pair of input chromosome images are similar or dissimilar, and applied their approach to an unspecified dataset of chromosome images from hospital patients.

Recently, Szubert et al. [51] presented *iris*, a siamese neural network-based technique for visualization and interpretation of single-cell datasets. The authors proposed their approach as an alternative to  $t$ -distributed Stochastic Neighbor Embedding ( $t$ -SNE) algorithm [52], a popular algorithm for large dataset dimensionality reduction visualization [53]. It is worth mentioning that the code repository of the *iris* algorithm is the most accessed and exploited free siamese neural network code repository available on GitHub to date (Subheading 4.2).

### 3.3 Chemistry and Pharmacology

Siamese neural networks have been used by Jeon et al. [54] for drug discovery purposes. The authors introduced a new application, called *ReSimNet*, which uses a siamese neural network to compute the transcriptional response similarity between two chemical compounds. The authors state that *ReSimNet* can be employed to discover drugs that are similar to already-existing drugs which demonstrated effectiveness on specific diseases.

### 3.4 Geometry and Graphics

In this field, we report a preprint by Sun et al. [55], where the authors used a siamese neural network for the analysis of spectral shape descriptors, which are key components of computational geometry. To calculate the Euclidean distance between the elements of two spectral shape descriptors, the authors employed a siamese neural network to project the spectral shape descriptors on to a suitable metric space.

### 3.5 Image Analysis

Image analysis is the field with the highest number of applications for the siamese neural network.

As mentioned earlier, Baldi and Chauvin [18] used a model similar to siamese neural networks to recognize fingerprints from images.

Chopra et al. [56] employed a siamese neural network for face verification. The authors used this model to assess semantic similarity between processed images.

Paisios et al. [57] presented a unique application of siamese neural networks for fashion: their tool processes images and can recognize similar clothes.

Yi et al. [58] employed a siamese neural network for deep metric learning, while Lefebvre et al. [59] used it for visual similarity between images.

In their article, Berlemont et al. [60] took advantage of the siamese model for symbolic gesture recognition. They tested their application on inertial Micro Electro Mechanical Systems (MEMS) data collected through smartphones.

Kassis et al. [61] presented a unique application of the siamese neural network in the field of history. They developed an application able to assist historians in historical manuscript alignment, by processing images of handwritten pages of ancient historical texts, and determining their similarity level.

Liu et al. [62] and He et al. [63] showed an interesting application of siamese neural networks for another topic: recognition and classification of images from remote sensing. Remote sensing is the acquisition of images of faraway objects without physical contact, and often refers to Earth images collected by space satellites. Liu et al. [62] presented an application of machine learning for classification of remote sensing images, and He et al. [63] for similarity matching of remote sensing images.

Taigman et al. [64] from Facebook Inc. developed *DeepFace*, an application of the siamese neural network for face verification from photos. Recognizing faces and facial expression intensity is the goal of an article by Sabri and Kurita [65].

Hanif [66] developed an enhanced two-channel and siamese neural network for image patch matching, and tested their method on the UBC image patch benchmark dataset [67].

### 3.5.1 Handwritten Forgery Detection

As we mentioned earlier, Bromley et al. [16] introduced the siamese neural network model in 1994 originally to detect forgeries among handwritten signatures. Since this application was the original and first usage of the model, other authors have taken advantage of it to solve the same problem. Therefore, we decided this application deserved a subsection on its own within the image analysis section.

Guyon et al. [68] released an advanced version of Bromley's application [16], able to recognize forgeries from pen-writing features.

Grafilon et al. [69] employed a siamese neural network to develop a mobile application able to detect forgeries. After the camera of the smart phone takes a photo of a signature, their Android app analyzes the signature image, and compares it to other signatures for verification. The authors claim their application "might help the authorities in the never ending battle against crime, especially against forgers and thieves" [69], and we agree with their view that this application might useful to law enforcement.

In their paper, Du et al. [70] presented a convolutional siamese neural network to detect forgeries from images of handwritten signatures. The authors implemented several neural network architectures (VGG13 [71], GoogLeNet [72], and Residual

Networks—ResNet [73]) and tested them on the IAM Database, which contains images of approximately 1.5 thousand pages of scanned handwritten English text written by around 700 different writers [74].

The convolutional siamese neural network is also the model employed by Dey et al. [75] in their application, *SigNet*, developed to identify forgeries. The authors tested their model on the CEDAR signature database [76]. It is interesting to note that, in their paper, Dey et al. [75] used the Euclidean distance for the comparison between the outputs of the two parallel neural networks, rather than the cosine distance employed by Bromley and colleagues in their original siamese neural network design [16].

Finally, Ahrabian and Babaali [77] presented a use of the siamese neural network to detect false signatures tested on a Japanese dataset: the SigWiComp2013 Japanese dataset [78, 79]. Their paper boasts a detailed description of the neural network model employed.

### 3.5.2 One-Shot Image Recognition

The paper by Koch et al. [80] about one-shot image recognition deserves a special mention in our overview, given the number of citations (1,247 on Google Scholar to date 15th June 2020), which is the highest among the application papers collected in this survey.

A *one-shot learning* setting is a framework where a computational system must correctly make predictions given only a single training data instance (an image, in this case [80]).

These authors from the University of Toronto developed a convolutional neural network containing a siamese neural network, and tested it to recognize images on the Omniglot dataset [81]. This paper had a large impact on one-shot image recognition, and has become a pillar in the field, even inspiring other practitioners to write tutorials about how to implement it [82, 83].

## 3.6 Medicine and Health

Several scientists used siamese neural networks for applications in medicine and health as well.

Wang et al. [84] employed this model to detect spinal metastasis in magnetic resonance imaging (MRI) images. They applied this method to a dataset of magnetic resonance images collected at the Peking University Third Hospital, and evaluated its performance through the free-response receiver operating characteristic (FROC) curve [85].

Parajuli et al. [86] took advantage of a siamese model to track cardiac motion. The authors developed Flow Network Based Tracking (FNT) enhanced with siamese neural networks, and tested it on the KU Leuven synthetic data of cardiomyopathy echo sequences.

Chung and Weng [87] developed a deep siamese convolutional neural network for content-based medical image retrieval (CBMIR), and they tested it on a diabetic retinopathy fundus image dataset.

Zeng et al. [88] faced the same scientific problem: they proposed a binocular siamese-like convolutional neural network to automatically diagnose diabetic retinopathy from retinal fundus photographs. They applied their method to the dataset of the Kaggle diabetic retinopathy competition [89].

Another application of siamese neural networks to medical image was proposed by Wang et al. [90] to detect microcalcification in mammograms. The authors developed a context-sensitive deep neural network (DNN) especially capable at reducing the number of false positives (FP), which are local image patterns that resemble microcalcifications without actually being them.

Patil et al. [91] proposed a siamese long short-term memory (LSTM) neural network to classify brain fiber tracts in tractography segmentation images.

Liu et al. [92] employed a siamese neural network to identify brain asymmetries related to Alzheimer's disease in T1 scans. They tested their approach on ADNI 1, ADNIGO, ADNI 2, and BIOCARD databases [93].

### **3.7 Optics and Physics**

Siamese neural networks have applications in optics and physics, too. Zou et al. [94] employed this model for laser vision seam tracking. Their system can process weld images and can resist the interference of spatter and arc in the welding process.

De Baets et al. [95] used a siamese neural network for non-intrusive load monitoring. Their application can detect unidentified appliances in the total power consumption of a household, by using a siamese neural network that compares the known energy instances and the unknown energy instances.

### **3.8 Robotics**

A few authors have taken advantage of siamese neural networks in robotics.

Utkin et al. [96, 97] employed this model to detect anomalous behaviors of robots in multi-robot systems (MRS).

In their preprint, Zeng et al. [98] applied the siamese neural network (among other methods) to make robots capable of grasping and identifying both known and new objects in obstructed environments.

### **3.9 Sensor-Based Activity Recognition**

In activity recognition, we found a siamese neural network application in an article of Berlemont et al. [99], where the authors employed this method for human action recognition (HAR). The authors applied their method to the ChAirGest dataset [100], containing data collected through sensors on humans in motion.

### **3.10 Software Development**

Zhao et al. [101, 102] have published two articles describing the application of siamese neural networks for software defect prediction (SDP). They tested their applications on the U.S. National Aeronautics and Space Administration (NASA) metrics data program (MDP) repository [103].

### **3.11 Text Mining and Natural Language Processing (NLP)**

In the text mining community, several scientists have taken advantage of siamese neural networks.

Yih et al. [104] proposed Similarity Learning via Siamese Neural Network (S2Net), a technique able to discriminatively learn concept vector representations of text words. They showed the results of their approach applied to a Wikipedia dataset.

Long and Wei [105] employed several neural network models for semantic parsing in natural language programming. Among the models tested, the authors include the siamese neural network, although it was outperformed by a recurrent neural network (RNN).

In their article, Kumar et al. [106] employed the siamese neural network to recognize clickbaits in online media outlets. They proposed a sophisticated neural network architecture that incorporates a recurrent neural network, a siamese neural network, and a neural attention mechanism. The authors tested their approach on approximately 20 thousand social media posts from Twitter [107].

Zhu et al. [108] proposed an application of siamese neural networks to learn sentence representations. The authors implemented a Dependency-based Long Short-Term Memory (D-LSTM) neural network to first get the primary sentence information and then to produce supporting components, and tested their approach on the SICK dataset [109].

Sandouk and Chen [110, 111] presented two applications of siamese neural networks employed to identify music tags, and applied their approach to several musical tags datasets (CAL500, MagTag5K, and Million Song Dataset).

González et al. [112] proposed a natural language processing application of the siamese neural network for extractive summarization, meaning that their technique can extrapolate most relevant sentences in a document.

Das et al. [113] used the model to detect similar questions on question and answer websites such as Yahoo Answers, Baidu, Zhi-dao, Quora, and Stack Overflow.

### **3.12 Video Analysis**

Since the siamese neural network showed its effectiveness in image processing (Subheading 3.5), it has been employed in several video analysis applications too. In fact, videos are a series of images, and can provide information about the movement and the presence of objects and people in any context.

Ryoo et al. [114] used a multi-siamese convolutional neural network to recognize activity in videos having extremely low resolution.

Liu et al. [115] developed PROVID, a framework that processes videos for vehicle re-identification. PROVID takes advantage of a siamese neural network to verify license plates.

Kovač et al. [116] analyzed videos to recognize gait, meaning speed changes and other features related to how humans walk. They tested their approach on the OU-ISIR gait dataset [117].

Gait feature recognition is also the goal of an article by Liu et al. [118], who employed a siamese neural network to process images and identify humans based on their gait.

Zhang et al. [119] worked on the same scientific goal. The authors used a siamese neural network to recognize humans from their gait, based on surveillance videos.

In their article, Zhang et al. [120] employed a siamese convolutional neural network for arbitrary object tracking: given a video, their application is able to match the patch of the target in the first video frame with candidate matches in the following frames. The authors tested their approach on the OTB-100 dataset [121].

Object tracking is also the topic of the article published by Lee and Kim [122]. In their study, the authors employed a variant of the siamese neural network, called *feature pyramid siamese network (FPSN)* as a similarity metric for their architecture. The goal of their study is multiple object tracking from videos.

Also regarding object tracking, Yang et al. [123] proposed a model called *a Region-based Multi-Scale Fully Convolutional Siamese Network (R-MSFCN)* for real-time visual object tracking from videoclips. They reported the results of their tests on the OTB dataset [121] and VOT2015 dataset [124].

Wu et al. [125] used a siamese neural network to develop an application for person re-identification in video-surveillance. In this article, the authors released a *siamese attention* architecture that simultaneously learns spatio-temporal video representations and their similarity metrics.

Kuai et al. [126] employed a siamese neural network model, called *SiamFC*, to produce a distractor-aware tracking system which incorporates a correlation filter tracker. Similar to what Yang et al. [123], they tested their method on the OTB and VOT2015 datasets.

Wang et al. [127] arranged a siamesed fully convolutional neural network (FCN) and employed for road detection from the perspective of moving vehicles. Their application can detect road boundaries and contours from input videoclips recorded from the drivers' perspective, and can have a huge impact on autonomous driving.

Mobahi et al. [128] used the model for a temporal coherence learning task: they employed a deep siamese neural network to recognize the same objects from sequences of video frames.

### 3.13 Theory Studies

Since the appearance of siamese neural networks in 1994, the computer science community has also released several theoretical studies related to the siamese model and its possible improvements.

Masci et al. [129], for example, developed a similarity-preserving hashing model based on the siamese neural network, and showed some applications on multimedia information retrieval.

In his doctoral thesis at University of Toronto, Liu [130] proposed a *probabilistic siamese neural network*, a variant of the original siamese neural network that incorporates a Gaussian probability similarity from the *probabilistic neural network* [27, 131].

Shaham and Lederman [132], instead, investigated the effectiveness of siamese neural networks in common variable learning, that is the ability of a computational system to discriminate between different sources of variability present in data.

Ibraheem [133] published a study on the objective functions for siamese neural networks, while Hadsell et al. [134] developed a siamese neural network-based model for dimensionality reduction mapping.

Zheng et al. [135] presented two metric learning methods for pairwise verification, and showed some results both on face verification and on speaker verification.

## 4 Software Packages, Tutorials, and Guides

In this section, we list the main software packages and libraries (Subheading 4.1), some tutorials (Subheading 4.2), and guides (Subheading 4.3) for the readers who would like to implement a siamese neural network in their own software scripts.

### 4.1 Software Packages

Several software packages and libraries can be used to implement siamese neural networks. The most popular open source ones released with permissive free software licenses are the following:

- TensorFlow [136, 137]
- PyTorch [138, 139]
- Torch [140, 141]
- Keras [142, 143]
- MXNet [144, 145]

Among all these programming libraries, Torch is the only platform built on Lua [146]. All the other listed packages can be used with several programming language front-ends, with Python being the most popular one. Python 3, in fact, can be employed to run TensorFlow, PyTorch, Keras, and MXNet, and is one of the most popular programming languages in the machine learning community (for example, among Kaggle users [147]).

The Keras library can also be run in the R software environment, even with TensorFlow back-end [148]. Some data science practitioners claimed TensorFlow to be the fastest programming package for deep learning in 2018 [149].

We take this opportunity to invite the readers to use only free and open-source software and programming languages with permissive free licenses, and to avoid proprietary software. Free and open-source software usage, in fact, fosters collaboration between researchers and institutes [15], provides more freedom to make unrestricted use of, and to study, copy, modify, and redistribute the developed scripts [150], and usually has no cost.

## 4.2 Tutorials

Several tutorials on how to implement siamese neural networks are available online. Holländer on his personal blog called *Becoming Human* [151], for example, proposed an implementation of this method in PyTorch to classify MNIST images [152].

Arpan, from the Kaggle community [153], released a tutorial on how to use siamese neural networks to recognize images of icebergs from images of ships, in Keras.

Keras is also the programming library used by Thoma on the tutorial he posted on his personal website [154]. He provides a code script to detect coordinates of landmarks for autonomous cars.

Reni published a tutorial on the Innovation Incubator website [155], where he explains and provides PyTorch code on how to use a siamese neural network for offline signature classification. He applied his approach to the SigComp2011 Database [156].

Séguin [157], instead, provided a tutorial on siamese neural networks by using Mariana, a deep learning framework developed by Tencent Inc. Mariana is based on Theano [158], a formerly popular Python 2.7 deep learning platform whose development was abandoned in October 2017 [159]. This tutorial by Séguin shows how to create a siamese neural network on Mariana, but avoids reporting any specific example application [157].

Among the tutorials available online, we also would like to mention the free software code repositories available on GitHub [160], that are accessible by searching for the *siamese neural network* tag. Among them, a special mention is deserved by Bering Research for their implementation of the *ivis* algorithm [51] for dimensionality reduction of large datasets [161], and by Latkowski for his implementation of a siamese neural network built on a multihead attention mechanism [162].

## 4.3 Guides

In addition to code repositories and tutorials, some detailed guides about siamese neural networks are freely available in the internet.

On the Towards Data Science website, Doukkali published an accurate guide on how to use the model for face recognition from one-shot images [82], a topic that we already mentioned earlier in this survey (Subheading 3.5.2). Also on Towards Data Science, Sugimura published a detailed guide on this artificial neural network by providing an example on similarity between images of shoes [163].

One-shot learning is also the subject described by Bouma on a guide he published on his website [83], where he carefully describes the theoretical details of siamese neural networks, explains the goals of one-shot learning, and even provides the Keras software code of its implementation.

Among the guides available online to date, it is worth mentioning a free lecture on siamese neural networks available as a videoclip on Coursera [164]. Delivered by deeplearning.ai, this lecture is part of the Convolutional Neural Networks course within the Coursera Deep Learning specialization.

---

## 5 Discussion and Conclusions

Computing the semantic similarity between two lists or two objects is an important task in computer science. When the two objects can be represented in the same feature space, standard geometrical and statistical approaches can be used, such as Euclidean distance or others. But when two objects have data from different feature spaces, they need to be projected or represented in another space. And when there is the need to predict similar objects which might arrive in a future step, siamese neural networks can be the best choice.

In this manuscript, we explained how a siamese neural network works, described more than one hundred applications, reported software packages, tutorials, and guides which can be used for its programming implementation.

Since their appearance in 1994, computer scientists have applied siamese neural networks to a variety of different fields. Most of the applications happened to be in image analysis, which is the field where this model originally emerged. Image analysis applications include unique studies in fashion [57] and historical manuscript alignment [61], which stand out in this overview for their originality.

The capability of siamese neural networks to analyze images found extensive use in video analysis, as expected. The model also has several applications in traditional computer science fields such as audio and speech processing, text mining, and natural language processing, but was employed just few times in biology and robotics.

In the future, we expect to see further uses and more widespread application of siamese neural networks in multiple computer sciences studies and information technology tools.

## Acknowledgements

The author thanks Richard Zemel (University of Toronto) for his suggestion on siamese neural networks, Hugh M. Cartwright (Oxford University & University of Victoria) for his invitation to contribute to this book, and Trent Faultless (Toronto General Hospital) for the English proof-reading of the manuscript.

## References

1. Homayouni R, Heinrich K, Wei L, et al (2004) Gene clustering by latent semantic indexing of MEDLINE abstracts. *Bioinformatics* 21(1):104–115
2. Benesty J, Chen J, Huang Y, et al (2009) Pearson correlation coefficient. In: Noise reduction in speech processing. Springer, Berlin, pp 1–4
3. Binet A (1904) The proof and measurement of association between two things; general intelligence objectively determined and measured. *L'année psychologique* 11 (1):623–624
4. Chicco D, Ciceri E, Masseroli M (2014) Extended Spearman and Kendall coefficients for gene annotation list correlation. In: Proceedings of CIBB 2014 – the 11th international meeting on computational intelligence methods for bioinformatics and biostatistics, vol 8623. Springer, Berlin, pp 19–32
5. Kendall MG (1938) A new measure of rank correlation. *Biometrika* 30(1/2):81–93
6. Goodman LA, Kruskal WH (1963) Measures of association for cross classifications III: approximate sampling theory. *J Am Stat Assoc* 58(302):310–364
7. Resnik P (1999) Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language. *J Artif Intell Res* 11:95–130
8. Jiang X, Nariai N, Steffen M, et al (2008) Combining hierarchical inference in ontologies with heterogeneous data sources improves gene function prediction. In: Proceedings of IEEE BIBM 2008 – international conference on bioinformatics and biomedicine, pp 411–416
9. Jiang JJ, Conrath DW (1997) Semantic similarity based on corpus statistics and lexical taxonomy. arXiv preprint cmp-lg/9709008
10. Lin D (1998) An information-theoretic definition of similarity. In: Proceedings of ICML 1998 – the 15th international conference on machine learning, vol 98. Citeseer, pp 296–304
11. Chicco D, Palluzzi F, Masseroli M (2017) Novelty indicator for enhanced prioritization of predicted Gene Ontology annotations. *IEEE/ACM Trans Comput Biol Bioinf* 15 (3):954–965
12. Chicco D, Masseroli M (2015) Ontology-based prediction and prioritization of gene functional annotations. *IEEE/ACM Trans Comput Biol Bioinf* 13(2):248–260
13. Landauer TK, Dumais S (2008) Latent semantic analysis. *Scholarpedia* 3(11):4356
14. Chicco D, Masseroli M (2015) Software suite for gene and protein annotation prediction and similarity search. *IEEE/ACM Trans Comput Biol Bioinf* 12(4):837–843
15. Chicco D (2017) Ten quick tips for machine learning in computational biology. *BioData Min* 10(1):35
16. Bromley J, Guyon I, LeCun Y, et al (1994) Signature verification using a “siamese” time delay neural network. *Adv Neural Inf Process Syst* 6:737–744
17. Pautasso M (2013) Ten simple rules for writing a literature review. *PLoS Comput Biol* 9(7):1–4
18. Baldi P, Chauvin Y (1993) Neural networks for fingerprint recognition. *Neural Comput* 5 (3):402–418
19. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436
20. Cartwright HM (2008) Artificial neural networks in biology and chemistry – the evolution of a new analytical tool. *Methods Mol Biol* 458:1–13
21. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
22. Freund Y, Schapire RE (1999) Large margin classification using the perceptron algorithm. *Mach Learn* 37(3):277–296
23. Cartwright HM (2015) Artificial neural networks, 2nd edn., vol 1260. Methods in molecular biology. Springer, New York City

24. Rumelhart DE, Hinton GE, Williams RJ (1988) Learning representations by back-propagating errors. *Cogn Model* 5(3):1
25. Jurman G, Riccadonna S, Furlanello C (2012) A comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE* 7(8):e41882
26. Halligan S, Altman DG, Mallett S (2015) Disadvantages of using the area under the receiver operating characteristic curve to assess imaging tests: a discussion and proposal for an alternative approach. *Eur Radiol* 25(4):932–939
27. Chicco D, Rovelli C (2019) Computational prediction of diagnosis and feature selection on mesothelioma patient health records. *PLoS ONE* 14(1):e0208737
28. Cangelosi D, Pelassa S, Morini M, et al (2016) Artificial neural network classifier predicts neuroblastoma patients' outcome. *BMC Bioinf* 17(12):347
29. Maggio V, Chierici M, Jurman G, et al (2018) Distillation of the clinical algorithm improves prognosis by multi-task deep learning in high-risk neuroblastoma. *PLoS ONE* 13(12):e0208924
30. Google (2019) Google Scholar. <https://scholar.google.com>. Accessed 1 Aug 2019
31. Elsevier (2019) Scopus. <https://www.scopus.com>. Accessed 1 Aug 2019
32. Dagstuhl S (2019) Digital Bibliography & Library Project (DBLP) Computer Science Bibliography. <https://dblp.uni-trier.de>. Accessed 1 Aug 2019
33. National Center for Biotechnology Information (NCBI), U.S. National Library of Medicine (NLM) (2019) PubMed. <https://www.ncbi.nlm.nih.gov/pubmed/>. Accessed 1 Aug 2019
34. Ernst M (2019) Washington.edu – choosing a venue: conference or journal? <https://homes.cs.washington.edu/~mernst/advice/conferences-vs-journals.html>. Accessed 1 Aug 2019
35. Thiolliere R, Dunbar E, Synnaeve G, et al (2015) A hybrid dynamic time warping-deep neural network architecture for unsupervised acoustic modeling. In: Proceedings of INTERSPEECH 2015 – the 16th annual conference of the international Speech Communication Association
36. Pitt MA, Johnson K, Hume E, et al (2005) The Buckeye corpus of conversational speech: labeling conventions and a test of transcriber reliability. *Speech Commun* 45(1):89–95
37. Barnard E, Davel MH, Heerden Cv, et al (2014) The NCHLT speech corpus of the South African languages. In: Spoken language technologies for under-resourced languages
38. Chen K, Salman A (2011) Extracting speaker-specific information with a regularized siamese deep network. In: Advances in neural information processing systems, pp 298–306
39. Manocha P, Badlani R, Kumar A, et al (2018) Content-based representations of audio using siamese neural networks. In: Proceedings of ICASSP 2018 – the 2018 IEEE international conference on acoustics, speech and signal processing. IEEE, Piscataway, pp 3136–3140
40. Zhang Y, Pardo B, Duan Z (2018) Siamese style convolutional neural networks for sound search by vocal imitation. *IEEE/ACM Trans Audio Speech Lang Process* 27(2):429–441
41. Cartwright M, Pardo B (2015) Vocalsketch: vocally imitating audio concepts. In: Proceedings of CHI 2015 – the 33rd annual ACM conference on human factors in computing systems. ACM, New York, pp 43–46
42. Švec J, Šmídl L, Psutka JV (2017) An analysis of the RNN-based spoken term detection training. In: Proceedings of SPECOM 2017 – the 19th international conference on speech and computer Specom. Springer, Berlin, pp 119–129
43. Shon S, Ali A, Glass J (2017) MIT-QCRI Arabic dialect identification system for the 2017 multi-genre broadcast challenge. In: Proceedings of IEEE ASRU 2017 – the 2017 IEEE workshop on automatic speech recognition and understanding. IEEE, Piscataway, pp 374–380
44. Gündoğdu B, Yusuf B, Saraclar M (2017) Joint learning of distance metric and query model for posteriogram-based keyword search. *IEEE J Sel Top Sign Process* 11(8):1318–1328
45. Siddhant A, Jyothi P, Ganapathy S (2017) Leveraging native language speech for accent identification using deep siamese networks. In: Proceedings of ASRU 2017 – the 2017 IEEE workshop on automatic speech recognition and understanding. IEEE, Piscataway, pp 621–628
46. Zeghidour N, Synnaeve G, Usunier N, et al (2016) Joint learning of speaker and phonetic similarities with siamese networks. In: Proceedings of INTERSPEECH 2016 – the 17th annual conference of the international Speech Communication Association, pp 1295–1299
47. Zheng W, Yang L, Genco RJ, et al (2018) SENSE: siamese neural network for sequence embedding and alignment-free comparison. *Bioinformatics* 35(11):1820–1828

48. Kariin S, Burge C (1995) Dinucleotide relative abundance extremes: a genomic signature. *Trends Genetics* 11(7):283–290
49. Clemente JC, Pehrsson EC, Blaser MJ, et al (2015) The microbiome of uncontacted Amerindians. *Sci Adv* 1(3):e1500183
50. Jindal S, Gupta G, Yadav M, et al (2017) Siamese networks for chromosome classification. In: Proceedings of ICCV 2017 – the IEEE international conference on computer vision, pp 72–81
51. Szubert B, Cole JE, Monaco C, et al (2019) Structure-preserving visualisation of high dimensional single-cell datasets. *Sci Rep* 9 (1):8914
52. van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. *J Mach Learn Res*:2579–2605
53. Fernandes K, Chicco D, Cardoso JS, et al (2018) Supervised deep learning embeddings for the prediction of cervical cancer diagnosis. *Peer J Comput Sci* 4:e154
54. Jeon M, Park D, Lee J, et al (2019) ReSimNet: drug response similarity prediction using siamese neural networks. *Bioinformatics* 35:5249–5256
55. Sun Z, He Y, Gritsenko A, et al (2017) Deep spectral descriptors: learning the point-wise correspondence metric via siamese deep neural networks. arXiv preprint arXiv:1710.06368
56. Chopra S, Hadsell R, LeCun Y (2005) Learning a similarity metric discriminatively, with application to face verification. In: Proceedings of CVPR 2005 – the 2005 IEEE Computer Society conference on computer vision and pattern recognition, pp 539–546
57. Paisios N, Subramanian L, Rubinsteyn A (2012) Choosing which clothes to wear confidently: a tool for pattern matching. In: Proceedings of pervasive 2012 – the 10th conference on pervasive computing, workshop on frontiers in accessibility for pervasive computing
58. Yi D, Lei Z, Liao S, et al (2014) Deep metric learning for person re-identification. In: Proceedings of ICPR 2014 – the 22nd international conference on pattern recognition. IEEE, Piscataway, pp 34–39
59. Lefebvre G, Garcia C (2013) Learning a bag of features based nonlinear metric for facial similarity. In: Proceedings of IEEE AVSS 2013 – the 10th international conference on advanced video and signal based surveillance, pp 238–243
60. Berlement S, Lefebvre G, Duffner S, et al (2015) Siamese neural network based similarity metric for inertial gesture classification and rejection. In: Proceedings of IEEE FG 2015 – the 11th international conference and workshops on automatic face and gesture recognition, vol 1, pp 1–6
61. Kassis M, Nassour J, El-Sana J (2017) Alignment of historical hand-written manuscripts using siamese neural network. In: Proceedings of ICDAR 2017 – the 14th IAPR international conference on document analysis and recognition, vol 1. IEEE, Piscataway, pp 293–298
62. Liu X, Zhou Y, Zhao J, et al (2019) Siamese convolutional neural networks for remote sensing scene classification. *IEEE Geosci Remote Sens Lett* 16:1200–1204
63. He H, Chen M, Chen T, et al (2018) Matching of remote sensing images with complex background variations via Siamese convolutional neural network. *Remote Sens* 10 (2):355
64. Taigman Y, Yang M, Ranzato M, et al (2014) DeepFace: closing the gap to human-level performance in face verification. In: Proceedings of CVPR 2014 – the IEEE conference on computer vision and pattern recognition, pp 1701–1708
65. Sabri M, Kurita T (2018) Facial expression intensity estimation using Siamese and triplet networks. *Neurocomputing* 313:143–154
66. Hanif SM (2019) Patch match networks: improved two-channel and Siamese networks for image patch matching. *Pattern Recognit Lett* 120:54–61
67. Brown M, Hua G, Winder S (2010) Discriminative learning of local image descriptors. *IEEE Trans Pattern Anal Mach Intell* 33 (1):43–57
68. Guyon I, Bromley J, Matić N, et al (1996) Penacée: a neural net system for recognizing on-line handwriting. In: Models of neural networks III. Springer, Berlin, pp 255–279
69. Grafilon P, Aguilar IB, Lavarias ED, et al (2017) A signature comparing android mobile application utilizing feature extracting algorithms. *Int J Sci Technol Res* 6(8):45–50
70. Du W, Fang M, Shen M (2017) Siamese convolutional neural networks for authorship verification. Tech. rep. Stanford University
71. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
72. Szegedy C, Liu W, Jia Y, et al (2015) Going deeper with convolutions. In: Proceedings of CVPR 2015 – the 2015 IEEE conference on computer vision and pattern recognition, pp 1–9

73. He K, Zhang X, Ren S, et al (2016) Deep residual learning for image recognition. In: Proceedings of CVPR 2016 – the 2015 IEEE conference on computer vision and pattern recognition, pp 770–778
74. Marti UV, Bunke H (2002) The IAM-database: an English sentence database for offline handwriting recognition. *Int J Doc Anal Recognit* 5(1):39–46
75. Dey S, Dutta A, Toledo JI, et al (2017) SigNet: convolutional siamese network for writer independent offline signature verification. arXiv preprint arXiv:1707.02131
76. State University of New York at Buffalo, Center of Excellence for Document Analysis and Recognition (2019) Cedar.buffalo.edu/databases – CEDAR Resources. <https://cedar.buffalo.edu/Databases/>. Accessed 9 Aug 2019
77. Ahraryan K, Babaali B (2018) Usage of auto-encoders and Siamese networks for online handwritten signature verification. *Neural Comput Appl* 31:1–14
78. Malik MI, Liwicki M, Alewijnse L, et al (2013) ICDAR 2013 competitions on signature verification and writer identification for on-and offline skilled forgeries (SigWiComp 2013). In: Proceedings of ICDAR 2013 – the 12th international conference on document analysis and recognition. IEEE, Piscataway, pp 1477–1483
79. Malik MI (2019) Uab.es – signature verification and writer identification competitions for on- and offline skilled forgeries. [http://tc11.cvc.uab.es/datasets/SigWiComp2013\\_1](http://tc11.cvc.uab.es/datasets/SigWiComp2013_1). Accessed 9 Aug 2019
80. Koch G, Zemel R, Salakhutdinov R (2015) Siamese neural networks for one-shot image recognition. In: Proceedings of ICML 2015 – the 32nd international conference on machine learning, deep learning workshop, vol 2
81. Lake BM, Salakhutdinov R, Tenenbaum JB (2015) Human-level concept learning through probabilistic program induction. *Science* 350(6266):1332–1338
82. Doukkali F (2019) TowardsDataScience.com – One-shot learning: face recognition using siamese neural network. <https://towardsdatascience.com/one-shot-learning-face-recognition-using-siamese-neural-network-a13dcf739e>. Accessed 31 July 2019
83. Bouma S (2019) SorenBouma.github.io – One shot learning and siamese networks in Keras. <https://sorenbouma.github.io/blog/oneshot/>. Accessed 8 Aug 2019
84. Wang J, Fang Z, Lang N, et al (2017) A multi-resolution approach for spinal metastasis detection using deep Siamese neural networks. *Comput Biol Med* 84:137–146
85. Wang J, Nishikawa RM, Yang Y (2016) Improving the accuracy in detection of clustered microcalcifications with a context-sensitive classification model. *Med Phys* 43(1):159–170
86. Parajuli N, Lu A, Stendahl JC, et al (2017) Flow network based cardiac motion tracking leveraging learned feature matching. In: Proceedings of MIC-CAI 2017 – the 20th international conference on medical image computing and computer-assisted intervention. Springer, Berlin, pp 279–286
87. Chung YA, Weng WH (2017) Learning deep representations of medical images using siamese CNNs with application to content-based image retrieval. arXiv preprint arXiv:1711.08490
88. Zeng X, Chen H, Luo Y, et al (2019) Automated diabetic retinopathy detection based on binocular siamese-like convolutional neural network. *IEEE Access* 7:30744–30753
89. Team K (2019) Kaggle.com – Diabetic retinopathy detection. <https://www.kaggle.com/c/diabetic-retinopathy-detection/>. Accessed 9 Aug 2019
90. Wang J, Yang Y (2018) A context-sensitive deep learning approach for microcalcification detection in mammograms. *Pattern Recognit* 78:12–22
91. Patil SM, Nigam A, Bhavsar A, et al (2017) Siamese LSTM based fiber structural similarity network (FS2Net) for rotation invariant brain tractography segmentation. arXiv preprint arXiv:1712.09792
92. Liu CF, Padhy S, Ramachandran S, et al (2019) Using deep siamese neural networks for detection of brain asymmetries associated with Alzheimer's disease and mild cognitive impairment. *Magn Reson Imaging* 64:190–199
93. Soldan A, Pettigrew C, Lu Y, et al (2015) Relationship of medial temporal lobe atrophy, APOE genotype, and cognitive reserve in pre-clinical Alzheimer's disease. *Human Brain Mapp* 36(7):2826–2841
94. Zou Y, Li J, Chen X, et al (2018) Learning Siamese networks for laser vision seam tracking. *J Opt Soc Am A* 35(11):1805–1813
95. De Baets L, Develder C, Dhaene T, et al (2019) Detection of unidentified appliances in non-intrusive load monitoring using siamese neural networks. *Int J Electr Power Energy Syst* 104:645–653
96. Utkin LV, Zhuk YA, Zaborovsky VS (2017) An anomalous behavior detection of a robot

- system by using a hierarchical Siamese neural network. In: Proceedings of IEEE SCM 2017 – the XX IEEE international conference on soft computing and measurements, pp 630–634
97. Utkin LV, Zaborovsky VS, Popov SG (2017) Siamese neural network for intelligent information security control in multi-robot systems. *Autom Control Comput Sci* 51(8):881–887
  98. Zeng A, Song S, Yu KT, et al (2018) Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In: Proceedings of ICRA 2018 – the 2018 IEEE international conference on robotics and automation, pp 1–8
  99. Berlemont S, Lefebvre G, Duffner S, et al (2018) Class-balanced siamese neural networks. *Neurocomputing* 273:47–56
  100. Ruffieux S, Lalanne D, Mugellini E (2013) ChAirGest: a challenge for multimodal mid-air gesture recognition for close HCI. In: Proceedings of ICMI 2014 – the 15th ACM on international conference on multimodal interaction. ACM, New York, pp 483–488
  101. Zhao L, Shang Z, Zhao L, et al (2018) Siamese dense neural network for software defect prediction with small data. *IEEE Access* 7:7663–7677
  102. Zhao L, Shang Z, Zhao L, et al (2019) Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. *Neurocomputing* 352:64–74
  103. Gray D, Bowes D, Davey N, et al (2012) Reflections on the NASA MDP data sets. *IET Softw* 6(6):549–558
  104. Yih Wt, Toutanova K, Platt JC, et al (2011) Learning discriminative projections for text similarity measures. In: Proceedings of CoNLL 2011 – the 15th conference on computational natural language learning. Association for Computational Linguistics, Stroudsburg, pp 247–256
  105. Long R, Wei C (2015) Application of neural networks in the semantic parsing re-ranking problem. Tech. rep. Stanford University
  106. Kumar V, Khattar D, Gairola S, et al (2018) Identifying clickbait: a multi-strategy approach using neural networks. In: Proceedings of SIGIR 2018 – the 41st international ACM SIGIR conference on research & development in information retrieval, pp 1225–1228
  107. Pothast M, Gollub T, Komlossy K, et al (2018) Crowdsourcing a large corpus of clickbait on Twitter. In: Proceedings of COLING 2018 – the 27th international conference on computational linguistics, pp 1498–1507
  108. Zhu W, Yao T, Ni J, et al (2018) Dependency-based Siamese long short-term memory network for learning sentence representations. *PLoS ONE* 13(3):e0193919
  109. Marelli M, Menini S, Baroni M, et al (2014) A SICK cure for the evaluation of compositional distributional semantic models. In: Proceedings of LREC 2014 – the 9th international conference on language resources and evaluation. European Languages Resources Association, Paris, pp 216–223
  110. Sandouk U, Chen K (2016) Learning contextualized semantics from co-occurring terms via a Siamese architecture. *Neural Netw* 76:65–96
  111. Sandouk U, Chen K (2017) Learning contextualized music semantics from tags via a siamese neural network. *ACM Trans Intell Syst Technol* 8(2):24
  112. González J, Encarna S, García-Granada F, et al (2019) Siamese hierarchical attention networks for extractive summarization. *J Intell Fuzzy Syst* 36(5):4599–4607
  113. Das A, Yenala H, Chinnakotla M, et al (2016) Together we stand: siamese networks for similar question retrieval. In: Proceedings of ACL 2016 – the 54th annual meeting of the Association for Computational Linguistics (volume 1: long papers), pp 378–387
  114. Ryoo MS, Kim K, Yang HJ (2018) Extreme low resolution activity recognition with multi-siamese embedding learning. In: Proceedings of AAAI 2018 – the 32nd AAAI conference on artificial intelligence
  115. Liu X, Liu W, Mei T, et al (2017) PROVID: progressive and multimodal vehicle reidentification for large-scale urban surveillance. *IEEE Trans Multimedia* 20(3):645–658
  116. Kovač J, Štruc V, Peer P (2019) Frame-based classification for cross-speed gait recognition. *Multimedia Tools Appl* 78(5):5621–5643
  117. Makihara Y, Mannami H, Tsuji A, et al (2012) The OU-ISIR gait database comprising the treadmill dataset. *Inf Process Soc Jpn Trans Comput Vis Appl* 4:53–62
  118. Liu W, Zhang C, Ma H, et al (2018) Learning efficient spatial-temporal gait features with deep learning for human identification. *Neuroinformatics* 16(3–4):457–471
  119. Zhang C, Liu W, Ma H, et al (2016) Siamese neural network based gait recognition for human identification. In: Proceedings of ICASSP 2016 – the 41st IEEE international

- conference on acoustics, speech and signal processing, pp 2832–2836
120. Zhang H, Ni W, Yan W, et al (2018) Visual tracking using Siamese convolutional neural network with region proposal and domain specific updating. *Neurocomputing* 275:2645–2655
  121. Wu Y, Lim J, Yang MH (2015) Object tracking benchmark. *IEEE Trans Pattern Anal Mach Intell* 37(9):1834–1848
  122. Lee S, Kim E (2018) Multiple object tracking via feature pyramid Siamese networks. *IEEE Access* 7:8181–8194
  123. Yang L, Jiang P, Wang F, et al (2018) Robust real-time visual object tracking via multi-scale fully convolutional Siamese networks. *Multimedia Tools Appl* 77(17):22131–22143
  124. Kristan M, Matas J, Leonardis A, et al (2015) The visual object tracking vot2015 challenge results. In: Proceedings of ICCV 2015 – the IEEE international conference on computer vision workshops, pp 1–23
  125. Wu L, Wang Y, Gao J, et al (2019) Where-and-when to look: deep siamese attention networks for video-based person re-identification. *IEEE Trans Multimedia* 21(6):1412–1424
  126. Kuai Y, Wen G, Li D (2018) When correlation filters meet fully-convolutional Siamese networks for distractor-aware tracking. *Signal Process Image Commun* 64:107–117
  127. Wang Q, Gao J, Yuan Y (2018) Embedding structured contour and location prior in siamesed fully convolutional networks for road detection. *IEEE Trans Intell Transp Syst* 19(1):230–241
  128. Mobahi H, Collobert R, Weston J (2009) Deep learning from temporal coherence in video. In: Proceedings of ICML 2009 – the 26th annual international conference on machine learning, pp 737–744
  129. Masci J, Bronstein MM, Bronstein AM, et al (2013) Multimodal similarity-preserving hashing. *IEEE Trans Pattern Anal Mach Intell* 36(4):824–830
  130. Liu C (2013) Probabilistic Siamese Networks for Learning Representations. PhD thesis, University of Toronto
  131. Specht DF (1990) Probabilistic neural networks. *Neural Netw* 3(1):109–118
  132. Shaham U, Lederman RR (2018) Learning by coincidence: Siamese networks and common variable learning. *Pattern Recognit* 74:52–63
  133. Ibraheem AO (2019) On the choice of inter-class distance maximization term in siamese neural networks. *Neural Process Lett* 49(3):1527–1541
  134. Hadsell R, Chopra S, LeCun Y (2006) Dimensionality reduction by learning an invariant mapping. In: Proceedings of CVPR 2006 – the 2006 IEEE Computer Society conference on computer vision and pattern recognition, vol 2. IEEE, Piscataway, pp 1735–1742
  135. Zheng L, Duffner S, Idrissi K, et al (2016) Pairwise identity verification via linear concentric metric learning. *IEEE Trans Cybern* 48(1):324–335
  136. Abadi M, Barham P, Chen J, et al (2016) TensorFlow: a system for large-scale machine learning. In: Proceedings of OSDI 2016 – the 12th USENIX symposium on operating systems design and implementation, pp 265–283
  137. TensorFlow (2019) An end-to-end open source machine learning platform. <https://www.tensorflow.org>. Accessed 1 Aug 2019
  138. Paszke A, Gross S, Chintala S, et al (2017) Automatic differentiation in PyTorch. In: Proceedings of NIPS 2017 – the 31st conference on neural information processing systems, autodiff workshop
  139. PyTorch (2019) From research to production. <https://www.pytorch.org>. Accessed 1 Aug 2019
  140. Collobert R, Bengio S, Mariéthoz J (2002) Torch: a modular machine learning software library. Tech. rep. Idiap Research Institute
  141. Torch (2019) A scientific computing framework for LuaJIT. <https://www.torch.ch>. Accessed 1 Aug 2019
  142. Gulli A, Pal S (2017) Deep learning with Keras. Packt Publishing Limited, Birmingham.
  143. Keras (2019) The Python deep learning library. <https://www.keras.io>. Accessed 1 Aug 2019
  144. Chen T, Li M, Li Y, et al (2015) MXnet: a flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274
  145. MXNet (2019) Apache MXNet (Incubating), a flexible and efficient library for deep learning. <https://mxnet.apache.org>. Accessed 1 Aug 2019
  146. Ierusalimschy R (2006) Programming in Lua. Feisty Duck Digital Book Distribution, London
  147. Kaggle (2019) The state of data science & machine learning 2017. <https://www.kaggle.com/surveys/2017>. Accessed 6 Aug 2019

148. Falbel DD, Allaire JJ, Chollet F, et al (2019) Keras – R interface to Keras. <https://keras.rstudio.com/>. Accessed 22 Aug 2019
149. Hale J (2019) TowardsDataScience.com – Deep learning framework power scores 2018. <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>. Accessed 31 July 2019
150. Amant KS, Still B (2007) Handbook of research on open source software: technological, economic, and social perspectives. Information Science Reference, London
151. Holländer B (2019) BecomingHuman.ai – Siamese networks: algorithm, applications and PyTorch implementation. <https://becominghuman.ai/siamese-networks-algorithm-applications-and-pytorch-implementation-4ffa3304c18>. Accessed 31 July 2019
152. Deng L (2012) The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process Mag 29(6):141–142
153. Dhatt A (2019) Kaggle.com – Siamese neural networks, Python notebook using data from Package Data. <https://www.kaggle.com/arpandhatt/siamese-neural-networks>. Accessed 31 July 2019
154. Thoma M (2019) Martin-Thoma.com – Siamese Networks. <https://martin-thoma.com/siamese-networks/>. Accessed 31 July 2019
155. Reni RD (2019) InnovationIncubator.com – Siamese neural network (with PyTorch code example). <https://innovationincubator.com/siamese-neural-network-with-pytorch-code-example/>. Accessed 31 July 2019
156. Liwicki M, Malik MI, Van Den Heuvel CE, et al (2011) Signature verification competition for online and offline skilled forgeries (SigComp2011). In: Proceedings of ICDAR 2011 – the 11th international conference on document analysis and recognition. IEEE, Piscataway, pp 1480–1484
157. Séguin J (2019) Bioinfo.iric.ca – implementing a “siamese” neural network with Mariana 1.0. <https://bioinfo.iric.ca/implementing-a-siamese-neural-network-with-mariana-1-0/>. Accessed 31 July 2019
158. Al-Rfou R, Alain G, Almahairi Aea (2016) Theano: a Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688
159. Gee S (2019) I-Programmer.info – Theano to cease development after version 1.0. <https://www.i-programmer.info/news/105-artificial-intelligence/11183-theano-to-step-down-after-version-10.html>. Accessed 8 Aug 2019
160. Various Authors (2019) GitHub.com – topic: siamese-neural-network, 27 repositories. <https://github.com/topics/siamese-neural-network>. Accessed 31 July 2019
161. Research B (2019) GitHub.com/beringresearch – ivis. <https://github.com/beringresearch/ivis>. Accessed 8 Aug 2019
162. Latkowski T (2019) GitHub.com/tlatkowski – Siamese deep neural networks for semantic similarity. <https://github.com/tlatkowski/multihead-siamese-nets>. Accessed 8 Aug 2019
163. Sugimura M (2019) TowardsDataScience.com – Siamese networks and Stuart Weitzman boots. <https://towardsdatascience.com/siamese-networks-and-stuart-weitzman-boots-c414be7eff78>. Accessed 31 July 2019
164. deeplearning.ai (2019) Coursera.org – Convolutional neural networks: siamese network. <https://www.coursera.org/lecture/convolutional-neural-networks/siamese-network-bjhmj>. Accessed 31 July 2019



# Chapter 4

## Computational Methods for Elucidating Gene Expression Regulation in Bacteria

Kratika Naskulwar, Ruben Chevez-Guardado,  
and Lourdes Peña-Castillo 

### Abstract

Research over the past two decades has uncovered an unexpected complexity and intricacy of gene expression regulation in bacteria. Bacteria have (1) numerous small noncoding RNAs (sRNAs) which are ubiquitous regulators of gene expression, (2) a flexible and diverse promoter structure, and (3) transcription termination as another means of gene expression regulation.

To understand bacteria gene expression regulation, one needs to identify promoters, terminators, and sRNAs together with their targets. Here we describe the state of the art in computational methods to perform promoter recognition, sRNA identification, and sRNA target prediction. Additionally, we provide step-by-step instructions to use current approaches to perform these tasks.

**Key words** Bacteria gene expression regulation, Promoter recognition, sRNA identification, sRNA target prediction, Bioinformatics

---

### 1 Introduction

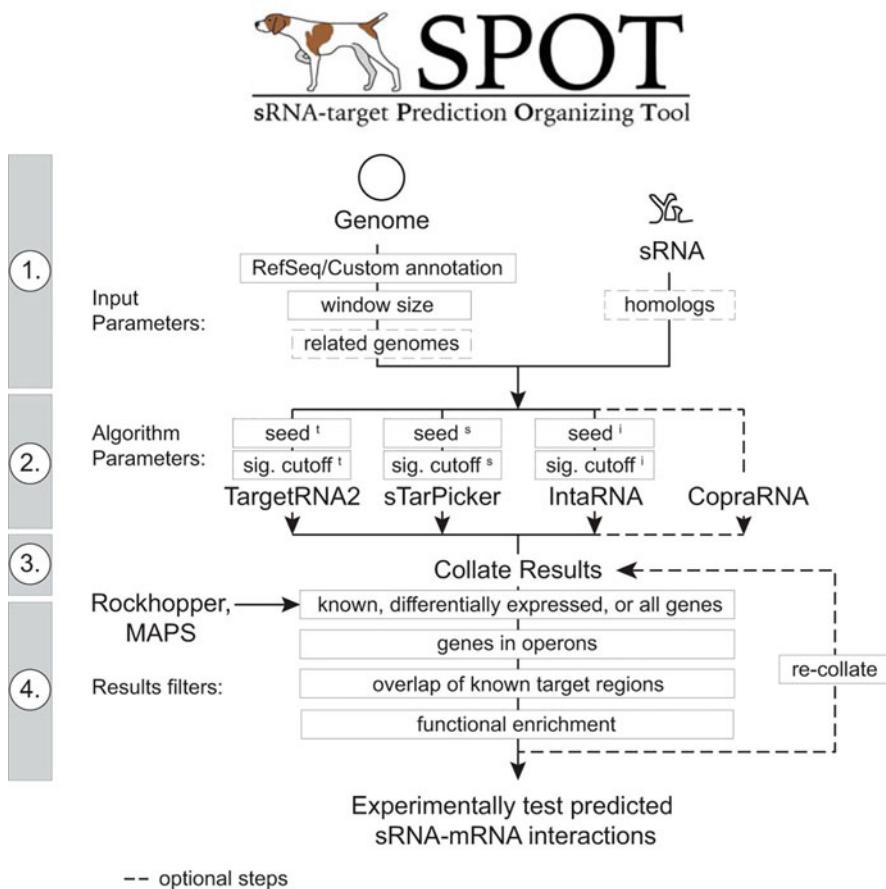
Studies in bacteria using RNA sequencing (RNA-seq) technologies have found a higher level of complexity and intricacy in bacterial transcriptomes than previously thought [1]. This complexity and intricacy in bacterial transcriptomes include numerous (hundreds) regulatory noncoding RNAs (sRNAs) [2, 3], flexible promoter sites [4], and gene expression regulation by transcription termination and/or attenuation [5].

Bacterial sRNAs are ubiquitous and plentiful regulators of gene expression involved in the control of many processes such as adaptive responses, stress responses, virulence, and pathogenicity. sRNAs have diverse mechanisms to control their target gene expression; but many of them act by antisense mechanisms on multiple target mRNAs. Most putative sRNAs have been discovered by analyzing RNA-seq data. There are several computational approaches for sRNA detection from RNA-seq data such as

APERO [6], ANNOgesic [7], and sRNA-Detect [8]. Among the existing tools, sRNA-Detect has the highest sRNA detection ability in terms of recall [6, 8]; however, the high number of small transcripts detected by sRNA-Detect suggests that several of them could be false positives. In this chapter, we describe how to use sRNA-Detect in combination with sRNACharP, a pipeline to characterize sRNAs, and sRNA-Ranking, a random-forest based classifier trained to rank sequences based on the likelihood of being bona fide sRNAs [9]. By ranking putative sRNAs detected by sRNA-Detect, the number of false positives can be reduced while maintaining a high recall rate.

To understand the functional role of an sRNA and the organization of its regulatory network, one needs to identify its mRNA targets. However, the rate of sRNA discovery has largely surpassed the rate to which sRNAs are been functionally characterized. There are several bioinformatic approaches for sRNA target prediction [10]. The most accurate approach, CopraRNA [11], depends on sequence conservation between bacterial species. However, as there are many species-specific sRNAs, and functionally equivalent sRNAs show very low sequence conservation [12], comparative genomics-based sRNA target prediction (as used by CopraRNA) is not suitable for the majority of sRNAs. SPOT [13] is a tool that combines up to four existing methods for sRNA target prediction (Fig. 1). SPOT's sensitivity (recall) is reported to be comparable to that of any individual method, while reducing the number of false positives [13]. Nevertheless, the number of false positives is still a limitation of *in silico* sRNA target prediction. In this chapter, we describe how to use two methods: CopraRNA webserver, and SPOT using Amazon Web Service (AWS) (<https://aws.amazon.com/>).

Accurate prediction of promoter regions is key to understanding gene expression regulation. The textbook description of bacterial genes is that they are assembled together into operons with a single promoter driving their expression. However, studies using dRNA-seq [14] and Cappable-seq [15] have detected roughly three times more transcription start sites (TSSs) than genes in several bacteria [15–17]. Furthermore, it has been shown that bacteria have diverse promoter features; for instance, in some bacteria and for some sigma factors, the –35 promoter site is degraded or completely absent [18, 19], and many promoter sites have been found within open reading frames (ORFs). Computational prediction of bacterial promoters from a genomic sequence remains a challenging problem, even though the first tools to tackle this problem came out decades ago. Most of the available tools are machine learning-based, as most of them use neural networks (NN) or support vector machines (SVMs). Many of the existing



**Fig. 1** Schematic representation of SPOT pipeline. Algorithms are run in parallel in step 2. Step 4 is optional. Figure is reproduced from [13] under the Creative Commons Attribution 4.0 International (CC BY 4.0) license (<https://creativecommons.org/licenses/by/4.0/>)

tools were trained and tested in one or two bacterial species (e.g., BProm [20], ZMethod [21], bacPP [22], CNNProm [23]), and their applicability to other bacterial species is arguable (although they are commonly used for other bacterial species). Shahmuradov et al. [24] showed that it is feasible to recognize promoters of different sigma classes and evaluated the performance of their tool (bTSSfinder) and other three tools on ten bacteria belonging to five different phyla. The best average sensitivity (recall) values obtained were 59% and 49% by bTSSfinder and BPROM, respectively. G4PromFinder [25] has been shown to outperform other tools for predicting promoters in GC-rich bacteria. In this chapter, we describe how to use bTSSfinder and G4PromFinder.

---

## 2 Materials

### 2.1 Hardware

A POSIX system (UNIX-like system such as Linux, OS X).

### 2.2 sRNA Detection and Prioritization

#### 2.2.1 Data

1. At least one SAM file (see <https://samtools.github.io/hts-specs/SAMv1.pdf> for a description of the format) with reads aligned to a reference genome. Read alignments in SAM format can be obtained using mapping programs such as BowTie2 [26] or BWA-MEM [27].

2. The reference genome in FASTA format (henceforth this file is referred to as genome.fasta).
3. The location of annotated transcripts (including rRNAs and tRNAs) in BED format (below this file is referred to as annotatedTranscripts.bed).
4. The location of protein-coding genes in the reference genome in BED format (henceforth this file is referred to as ORFs.bed) (*see Note 1*).

#### 2.2.2 Software

1. Python 2.7.
2. The HTSEQ python library [28] 0.5.4p5 ([https://htseq.readthedocs.io/en/release\\_0.9.1/](https://htseq.readthedocs.io/en/release_0.9.1/)).
3. Java ( $\geq 7$ ).
4. BEDtools [29] (<https://bedtools.readthedocs.io/en/latest/index.html>).
5. Nextflow [30] 0.27 (or higher) (<https://www.nextflow.io/>).
6. Docker 18.03 (or higher) (<https://www.docker.com/>).
7. R ( $\geq 3.0$ ) (<https://www.r-project.org/>).
8. The randomForest [31] (<https://CRAN.R-project.org/package=randomForest>) and optparse (<https://CRAN.R-project.org/package=optparse>) R libraries.
9. sRNA-Detect (<https://github.com/BioinformaticsLabAtMUN/sRNA-Detect>). This will be a single Python script.
10. sRNACharP (<https://github.com/BioinformaticsLabAtMUN/sRNACharP>). One need to download two files: sRNACharP.nf (a Nextflow file) and nextflow.config (a Nextflow configuration file).
11. sRNARanking (<https://github.com/BioinformaticsLabAtMUN/sRNARanking>). This will be an R script and an RDS file.

### **2.3 sRNA Target Prediction**

#### **2.3.1 Data**

1. Sequence of sRNA of interest in FASTA format using RefSeq identifier.
2. For CopraRNA, in addition to the sRNA of interest, the sequences of at least two homologous sRNA sequences to the sRNA of interest in FASTA format must be included in the same FASTA file. Sequences have to be from different organisms. The sequence names must be the individual genome RefSeq identifiers (in NZ\_\* or NC\_\* format).
3. For SPOT, the RefSeq identifier of the reference genome.

#### **2.3.2 Software**

1. For CopraRNA, a web browser.
2. For SPOT, an AWS account and an SFTP client such as Cyberduck.

### **2.4 Promoter Recognition**

#### **2.4.1 Data**

1. Genomic sequences in FASTA format. This file is henceforth referred to as genomeInput.fasta.

#### **2.4.2 Software**

1. A web browser.
2. For bTSSfinder, gfortran and libgfortran3.
3. For G4PromFinder, Python ( $\geq 3.7$ ) and the Python libraries biopython and openpyxl.

## **3 Methods**

### **3.1 sRNA Detection and Prioritization**

1. Upon download and/or installation of all required software as listed in Materials, detect putative sRNAs from RNA-seq data by running sRNA-Detect. In the terminal type the following in a single line:

```
python2.7 sRNADetect.py -samFile "file1.sam" -samFile  
"file2.sam" -out "output_sRNAs.gtf"
```

“file1.sam” and “file2.sam” are the names of the SAM files to use for sRNA detection. Multiple SAM files can be specified. “output\_sRNAs.gtf” is the name of the file sRNA-Detect will create. The output file will have the following columns: sequence name, program name (i.e., sRNADetect), transcript type (i.e., sRNA), start of the small transcript, end position of the small transcript, average read depth coverage of the small transcript, strand, and identifier generated by sRNA-Detect (*see Note 2*).

2. Remove transcripts overlapping annotated transcripts using BedTools' intersect tool by typing on the terminal in a single line the following:

```
bedtools intersect -wo -a output_sRNAs.gtf -b annotated-
Transcripts.bed -sorted -v -s -f 0.8 > sRNAs.bed
```

where output\_sRNAs.gtf is the file generated by sRNA-Detect in **step 1** and sRNAs.bed is a new file been created.

3. Pull the docker image for sRNACharP by typing on the terminal in a single line the following:

```
docker pull penacastillolab/srnacharp@sha256:c2a07f176d7c-
fe8cea3530bd76da05b30b182cdfe4d4b878f7d90e81f2d6a5f3
```

4. Create a folder called data and place in this folder the genome.fasta file, the sRNAs.bed file and the ORFs.bed.
5. Generate a feature table characterizing each putative sRNA in the file sRNAs.bed by executing sRNACharP in the terminal as follows:

```
nextflow run sRNACharP.nf --org="organism" --dir="/
ADD_PATH/data" --genome="genome.fasta" --sRNAs="sRNAs.
bed" --genomeAnnotation="ORFs.bed"
```

where “organism” should be replaced by the organism name, and ADD\_PATH should be replaced by the absolute path to the folder data (created in **step 4**) (*see Note 3*).

Figure 2 illustrates sample messages displayed on the terminal during a successful run of sRNACharP.

sRNACharP will create five text files into the working directory (i.e., the directory where sRNACharP was executed). One of these files is called *organism\_FeatureTable.tsv* (*organism* will be the name given with the parameter --org in **step 5**) which contains eight columns: sRNA identifier, free energy of predicted secondary structure, distance to the -10 position of the closest predicted promoter, distance to the closest predicted terminator, distance to the closest upstream ORF, a flag indicating whether the sRNA is on the same strand as the closest upstream ORF, distance to the closest downstream ORF, and a flag indicating whether the sRNA is on the same strand as the closest downstream ORF.

```

N E X T F L O W ~ version 0.27.0
Launching `sRNACharP.nf` [crazy_heYROVSKY] - revision: a560734176
[warm up] executor > local
[36/f67e03] Submitted process > createCRDFile
[80/28804e] Submitted process > getFASTAsRNAs
[72/6591a1] Submitted process > getUpstreamSequences (1)
[84/f7b852] Submitted process > getFreeEnergySS
[cb/bd7b45] Submitted process > runTranterm
[5d/6d813e] Submitted process > getPromoterSites (1)
[0b/eafc90] Submitted process > getPromoterSites (2)
[be/757ee4] Submitted process > getPromoterSites (3)
[b9/db95c2] Submitted process > getPromoterSites (4)
[b8/ad8cab] Submitted process > getPromoterSites (5)
[e6/5f8f87] Submitted process > parseTrantermResults (1)
[b5/53ee8a] Submitted process > getDistances (1)
[bb/b6a9ab] Submitted process > createAttributeTable (1)

Pipeline execution summary
-----
Completed at: Fri May 04 11:29:32 CEST 2018
Duration      : 17.3s
Success       : true
workDir       : /home/lpena/sRNACharP/work
exit status   : 0

```

**Fig. 2** Sample sRNACharP output messages. The number of submitted processes vary depending on the number of sRNA sequences included in the input. A successful run must show exit status 0 in the execution summary

6. Calculate the probability of each putative sRNA being an actual sRNA by running sRNARanking on the feature table generated by sRNACharP in **step 5**. To run sRNARanking type the following command as a single line in the terminal:

```
Rscript --vanilla RF_classifier4sRNA.R -i organism_FeatureTable.tsv -o outFile.txt
```

where organism\_FeatureTable.tsv is one of the files generated by sRNACharP and outFile.txt is the file generated by sRNARanking (*see Note 4*).

sRNARanking will generate a table with three columns: sRNA identifier, probability of not being an sRNA and probability of being an actual sRNA.

### 3.2 sRNA Target Prediction

#### 3.2.1 CopraRNA

1. On a web browser, navigate to the CopraRNA web server available at <http://rna.informatik.uni-freiburg.de/CopraRNA/Input.jsp>.
2. On CopraRNA web server, upload the FASTA file containing the sRNA sequences by clicking on the Choose File button, or type the sequences in the text box provided; and select the organism of interest (Fig. 3). *See Subheading 2 for a description of CopraRNA sequence requirements (see Note 5)*.

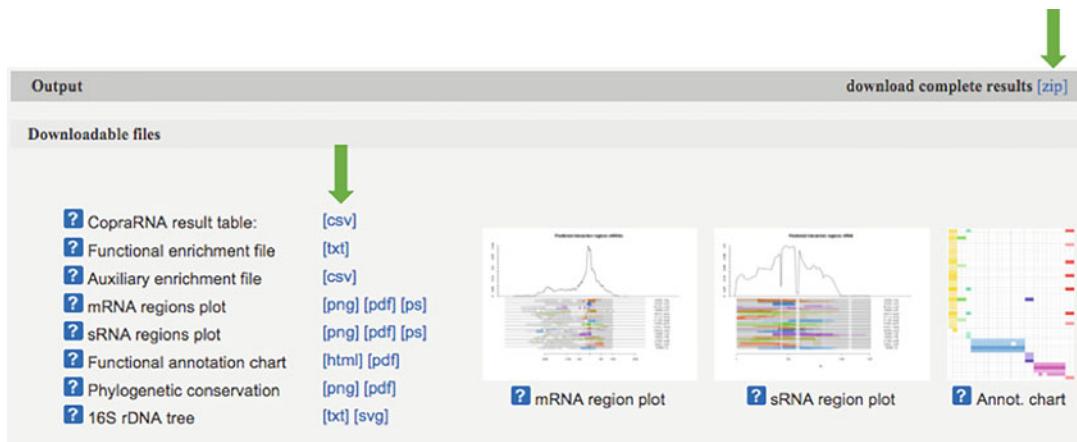
The screenshot shows the 'Sequence input' section of the CopraRNA web interface. It includes a text area for 'sRNA sequences' containing three entries: >NC\_000913, >NC\_009792, >NC\_013716, and >NC\_013716. A 'Choose File' button and a 'clear' button are available. Below this is a large blue 'FASTA' watermark. The 'Organism of interest' section lists three options: NC\_000913 (selected), NC\_009792, and NC\_013716. A red warning message at the bottom states: 'Warning: You entered less than 4 organisms. Prediction quality is much better with 4 or more input sequences if available.' A green arrow points to the right next to the message.

**Fig. 3** Screenshot of CopraRNA web server interface. Unless four sequences are given as input in FASTA format, a warning will be displayed (see Note 5). The organism of interest indicates the sRNA for which targets must be predicted

The screenshot shows the 'CopraRNA parameters' section. A red bracket on the left groups several parameters: 'Extract sequences around:', 'nt up (1-300)', 'nt down (1-300)', 'Consensus prediction:', 'p-value combination:', and 'p-value filtering (0=off)'. To the right of these are their respective input fields. Below this is the 'IntaRNA parameters' section, which includes 'Target folding window size:' and 'Target max. basepair distance:'. Default values are shown in the input fields.

**Fig. 4** Screenshot of CopraRNA available parameters. Default values are shown

3. [Optional] Modify the parameter values as seen fit. Information on each parameter can be found by clicking on the question mark symbol (?) on the left side of each parameter (Fig. 4).
4. [Optional] Enter a description of the job and an e-mail address. If an e-mail address is provided, user will get an e-mail notification upon CopraRNA job completion.
5. Click START to run CopraRNA. After clicking the START button, the user is redirected to a Result Retrieval page showing CopraRNA job details (see Note 6).

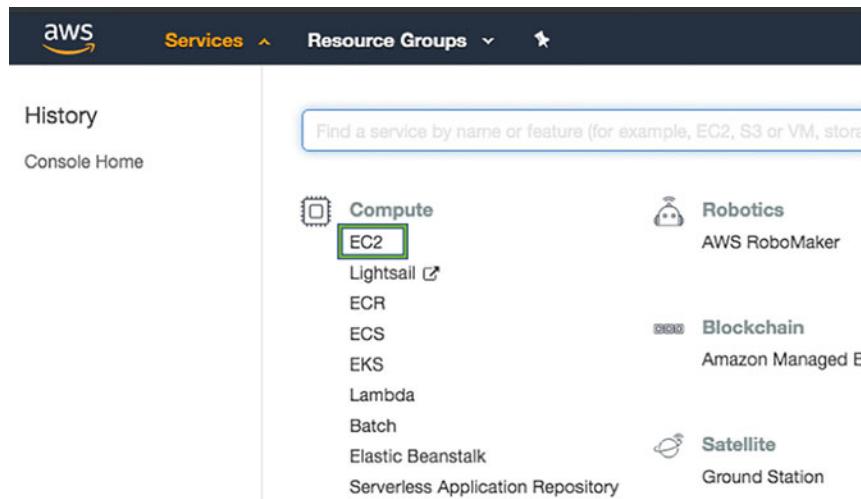


**Fig. 5** Screenshot of CopraRNA results. Files can be downloaded as a zip file by clicking on the link indicated by the top right green arrow or individually by clicking on the corresponding link. Links for individual files are pointed by the middle left green arrow. The main results are in the CopraRNA result table

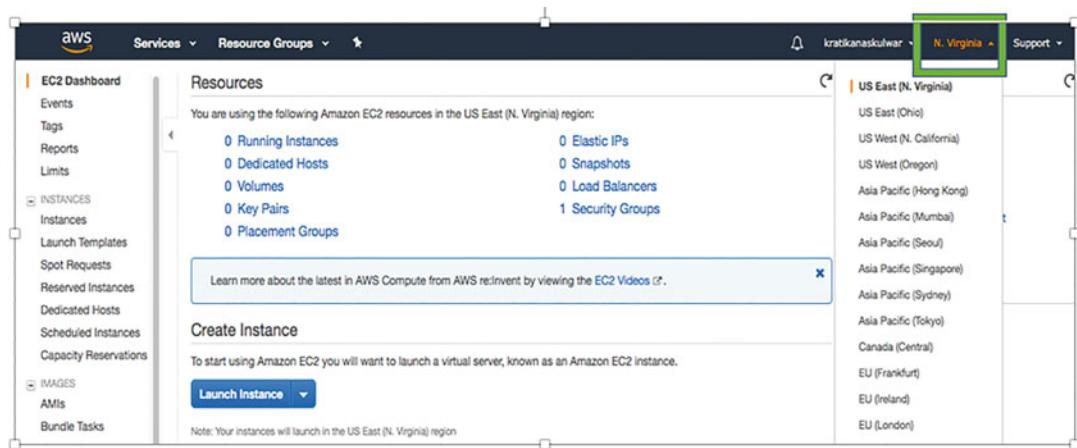
6. Once the job is completed, results can be retrieved from the link in the notification e-mail or the link shown in the Result Retrieval page.
7. CopraRNA generates several output files (Fig. 5). The main result table “CopraRNA result table” is provided in a CSV file containing a list of putative targets sorted by *p*-values. It is recommended to filter CopraRNA results using a *p*-value threshold of 0.01.
8. To interpret results, we recommend looking at experimentally validated targets. For example, CopraRNA predicted 200 putative targets of fnrS sRNA of *E. coli* with a *p*-value less than 0.01. In [10], 13 experimentally validated targets of fnrS are listed. Nine out of these 13 experimentally validated targets were predicted by CopraRNA. The remaining predicted targets could be novel targets or false positives.

### 3.2.2 SPOT (See Note 7)

1. If one does not have an AWS account, an AWS account needs to be created in the AWS portal available at <https://portal.aws.amazon.com/billing/signup#/start> by filling in the required information. Credit card details have to be provided during registration (see Note 8).
2. Log in to your AWS account. Upon successful login, one is redirected to the AWS management console.
3. On the AWS management console do the following:  
Click on the Services tab.  
Select EC2 (Elastic Computing Cloud) under Compute (Fig. 6).



**Fig. 6** Screenshot of the AWS management console. The option to be selected is indicated with a green square



**Fig. 7** Screenshot of the AWS management console. The green square indicates how to change the location of the Amazon EC2 resources

Change home region to “US East (N. Virginia)” (Fig. 7).

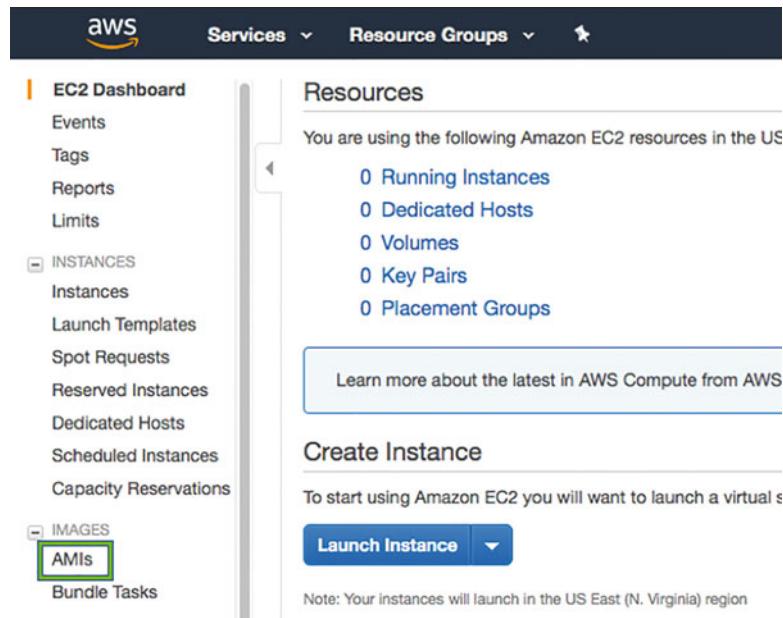
Choose AMIs under IMAGES from the right-hand side navigator (Fig. 8).

Switch to Public Images in the selection menu left to the search bar (Fig. 9).

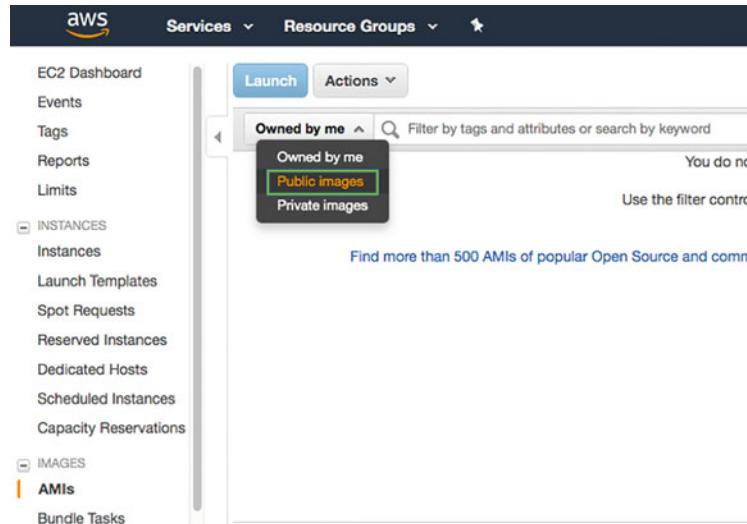
Enter “SPOTv1” on the search bar and search for it (Fig. 10).

Click the launch button and choose the instance type (Fig. 11) (*see Note 9*).

Click on “Next: Configure Instance Details” button on bottom-right and leave value as it is.



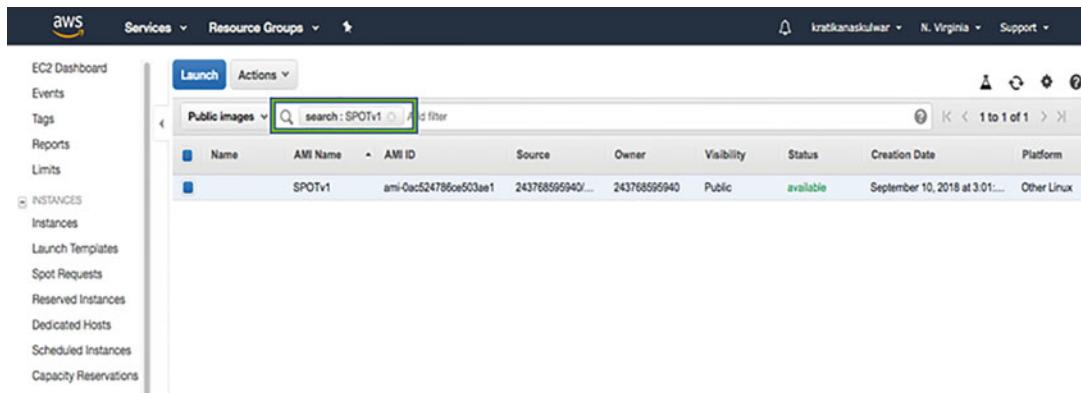
**Fig. 8** Screenshot of the AWS management console. The option to be selected is indicated with a green square



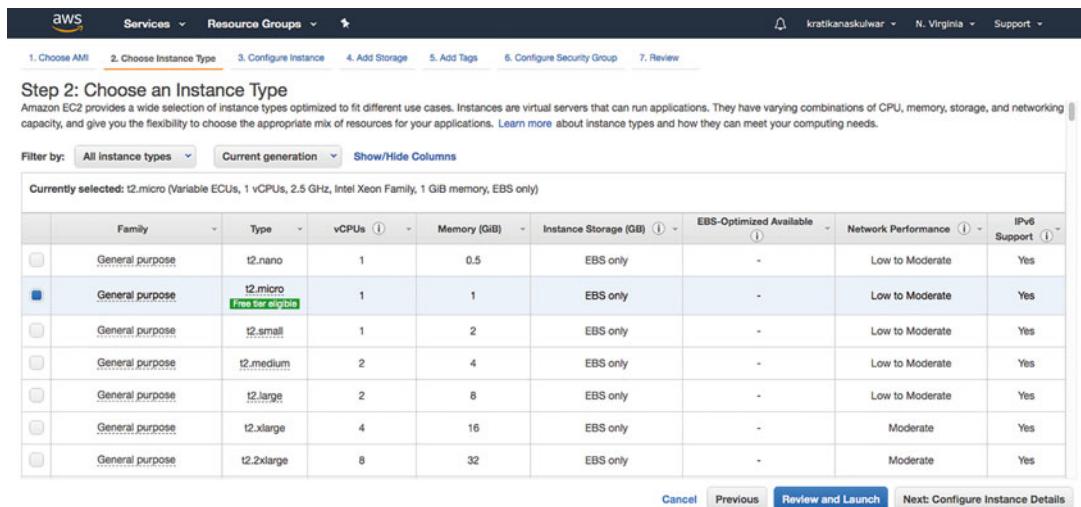
**Fig. 9** Screenshot of the AWS management console. The green square indicates how to select Public images

Click on “Next: Add Storage” button on the bottom right and adjust size to 30 GB.

Click on “Next: Add Tags” button on the bottom right and add a tag if desired.



**Fig. 10** Screenshot of the AWS management console. The green square indicates where to write SPOTv1 for searching on Public images



**Fig. 11** Screenshot of the AWS management console. Interface to choose an instance type. The only free instance is selected on the screen

Click on “Next: Configure Security Group” button on the bottom-right and leave the value as it is (a warning will be displayed).

Click on Review and Launch button on the bottom-right corner of the page, check the settings and warning and click on the Launch button on the bottom-right corner.

Select an existing key pair or create a new one. To create a new key pair, enter the desired key pair name and download the key pair by clicking on the “Download Key Pair” button. Download and save this key in the .ssh directory on your system (*see Note 10*).

## Launch Status



## How to connect to your instances

Your instances are launching, and it may take a few minutes until they are in the running state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to accrue, you stop or terminate your instances.

Click [View Instances](#) to monitor your instances' status. Once your instances are in the running state, you can connect to them from the Instances screen. [Find out](#) how to connect to your instances.

## ▼ Here are some helpful resources to get you started

- [How to connect to your Linux instance](#) • [Amazon EC2: User Guide](#)
- [Learn about AWS Free Usage Tier](#) • [Amazon EC2: Discussion Forum](#)

While your instances are launching you can also

**Fig. 12** Screenshot of the AWS management console. Window view after an instance is launched

```
(base) apples-MacBook-Pro:~ kratikanaskulwar$ ssh -i ~/.ssh/myawskey.pem ubuntu@54.205.242.77
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-1066-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 Get cloud support with Ubuntu Advantage Cloud Guest:
 http://www.ubuntu.com/business/services/cloud

104 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Mon Sep 10 16:56:47 2018 from 138.23.161.215
ubuntu@ip-172-31-36-72:~$ █
```

**Fig. 13** Sample message displayed on the terminal upon connection to an AWS EC2 instance. Once the prompt appears one can run programs on the AWS instance

Click the Launch Instances button in the bottom-right corner of the dialogue. A new window will appear (Fig. 12). From this window, one can view instances and see the details of the running instance. From the instance details record “Public DNS (IPv4)” and “IPv4 Public IP” from the description tab for future use.

4. Open a terminal window, and on the terminal log in to the AWS instance by typing in a single line:

```
ssh -I ~/.ssh/keyPairFileName username@IPv4 PublicIP
```

where keyPairFileName is the name of the file with the key pair, username (*see Note 11*), IPv4 is the “Public DNS (IPv4)” (recorded in **step 3**) and PublicIP is the “IPv4 Public IP” (recorded in **step 3**). After login into the AWS instance, one will see a new prompt (Fig. 13). One is now connected to an AWS instance of SPOTv1 AMI and can run SPOT.

```
[ubuntu@ip-172-31-86-195:~$ spot.pl

Usage /scripts/spot.pl
Input parameters:
  -r      Fasta file of sRNA query
  -a      RefSeq Accession number (assumes any local files have RefSeq number as their prefix)
  -o      output file prefix (default = TEST)
  -g      Use local GenBank or PTT&FNA files for all Programs? (default = N use latest from GenBan
k,
  CopraRNA cannot use local files)
  -n      Other genome RefSeq ids for CopraRNA listed in quotes "", current max is 5 genomes (def
ault ='')
  -m      Multisequence sRNA file for each genome in CopraRNA list (default ='')
  -x      Email address for job completion notification (default ='')

Algorithm parameters:
  -u      Number of nt upstream of start site to search (default = 60)
  -d      Number of nt downstream of start site to search (default = 60)
  -s      seed sizes for I, T, S e.g., '6 7 6' (defaults TargetRNA = 7, IntaRNA & Starpicker = 6)
  -c      P/Threshold value Cutoff for T, S, I e.g., '0.5 .001 un'
  (defaults Target = 0.05, Starpicker = 0.5, IntaRNA = top)

Results Filters:
  -b      Number of nt upstream of start site to filter results (default = -20)
  -e      Number of nt downstream of start site to filter results (default = 20)
  Note: -b and -e ignored if using a list (-l) or Rockhopper results (-t)
  Note: Set -b and -e to -u and -d to get all possible matches in results
  -l      List of up and/or down regulated genes, include binding coord if known e.g.,
  b1181\tdown\n
  b3826\tup\tsRNA_start\tsRNA_stop\tmrNA_start\tmrNA_stop\n
  OR
  -t      transcriptome expression file from Rockhopper *_transcripts.txt
  -f      Rockhopper fold change cutoff (default = 1.5)
  -q      Rockhopper q value cutoff (default = 0.01)
  -k      Rockhopper RPKM cutoff value (default = 100)
  -p      Operon file from DOOR-2 (http://csbl.bmb.uga.edu/DOOR/index.php) (optional)
  -w      Report all genes even if List or Rockhopper provided ? (default = No)
  -y      Exclude target predictions by only 1 method ? (default = Yes)
  Note: Does not apply to genes on List or significantly expressed from Rockhopper
  -z      Skip sRNA-mRNA detection steps, and just re-analyze data [Y]es (default = No)
  (Run in the same directory & requires original results files from each program)

Example run to examine entire genome target mRNAs:
/scripts/spot.pl -r sgrS.fasta -o stringent -a NC_000913

Example of running for select target mRNAs:
/scripts/spot.pl -r sgrS.fasta -l sgrS_diff.txt -u 150 -d 100 -o relaxed -a NC_000913 -c '0.5 0.001 un'
-g Y

Example of running for reanalysis of data with different filters:
/scripts/spot.pl -r sgrS.fasta -u 150 -d 100 -o changed_50_30 -a NC_000913 -c '0.5 0.001 un' -g Y -b -50
-e 30 -z Y

Example run using Rockhopper results and CopraRNA :
/scripts/spot.pl -r sgrS.fasta -t NC_000913_SgrS_transcripts.txt -o express -a NC_000913 -m sgrS_homologs.fasta -n 'NC_002695 NC_008563' -u 150 -d 100

ubuntu@ip-172-31-86-195:~$ ]]
```

**Fig. 14** SPOT's usage message displayed on the terminal after executing the Perl script spot.pl without arguments

5. Transfer input files to AWS instance using an SFTP client. The address of the server to connect is the “Public DNS (IPv4)” (recorded in step 3). Username, password and key pair file are the ones used to connect to the AWS instance.
6. On the terminal, type spot.pl to see SPOT's help (Fig. 14) (*see Note 12*).

7. SPOT predicts target for a single sRNA at a time. For instance, to run a SPOT job searching for targets of fnrS sRNA (sequence provided on the FASTA file fnrS.fasta) on *E. coli* genome (RefSeq ID NC\_000913) saving the results as fnrs\_results\_\* files in the working directory, and getting upon completion a notification e-mail to abcd@email.com, one needs to type on the terminal in a single line (*see Note 13*): spot.pl-r fnrS.fasta -o fnrs\_results -a NC\_000913 -x abcd@email.com
8. SPOT generates several output files. The main results are in Sheet 2 of the \*.xlsx file and the file \*\_summary.pdf contains an R generated plot of these results (Fig. 15).
9. We ran SPOT to search for fnrS sRNA targets on *E. coli* str. K12 genome using the instance type m5.2xlarge with 8 processors and 32 GB RAM. As with CopraRNA, we used default settings. There is a low agreement between CopraRNA and SPOT, for example, only two predicted targets were predicted by both tools.

### 3.3 Promoter Recognition

#### 3.3.1 bTSSfinder

1. On a web browser navigate to bTSSfinder web page at <https://www.cbrc.kaust.edu.sa/btssfinder>.
2. Select “bTSSfinder for Linux” under the “Download” tab to download the bTSSfinder program.
3. Unzip the file BTSSFINDER.zip.
4. On the terminal, go to the BTSSFINDER directory, add the BTSSFINDER directory to the path and set the variable bTSSfinder\_Data as follows (*see Note 14*):

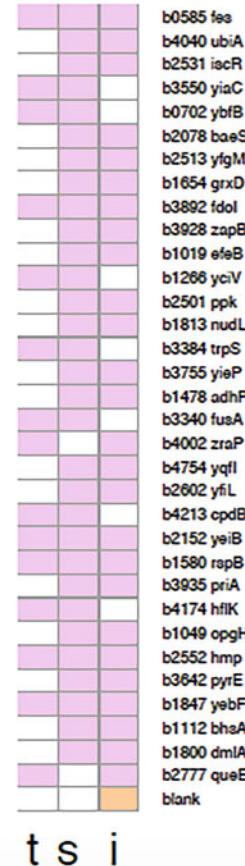
```
export bTSSfinder_Data=$(pwd)/Data
export PATH=$PWD:$PATH
```

5. To run bTSSfinder with default parameters type the following command in a single line on the terminal:

```
bTSSfinder -i genomeInput.fasta
```

bTSSfinder’s command options are shown in Fig. 16 (*see Note 15*).

6. bTSSfinder creates a BED file and a GFF file with the location of the predicted transcription start sites (TSSs), and the corresponding promoter sites.



**Fig. 15** Sample SPOT summary plot. The heatmap indicates which target prediction programs predicted each target: TargetRN2 (t), sTarpicker (s) or IntaRNA (i)

### 3.3.2 G4PromFinder

1. On the terminal clone G4PromFinder git repository with the following command: `git clone https://github.com/MarcoDiSalvo90/G4PromFinder.git`
2. Change to the G4PromFinder directory which should have the G4PromFinder.py script, a license file and a README file.

```
bTSSfinder -i - Input fasta file * REQUIRED
[-p - print (y or Y) or not print (n or N) the query sequence(s); Default: n]
[-o - Prefix for output file; Default: bTSSfinder]
[-x - a non-overlapping interval to select the highest-scoring TSS (post-processing); >=50,<=300; Default: 300]
[-n - Comma-separated A,C,G,T frequencies in the reference genome; Default: 0.25,0.25,0.25,0.25]
[-a - Neural Network Threshold for sigma70 or sigmaA promoters; -2<=parA<=2, Default: preset in training ]
[-b - Neural Network Threshold for sigma38 or sigmaC promoters; -2<=parB<=2, Default: preset in training ]
[-d - Neural Network Threshold for sigma32 or sigmaH promoters; -2<=parD<=2, Default: preset in training ]
[-e - Neural Network Threshold for sigma28 or sigmaF promoters; -2<=parE<=2, Default: preset in training ]
[-f - Neural Network Threshold for sigma24 or sigmaG promoters; -2<=parD<=2, Default: preset in training ]
[-h - 1 to search on the sense strand OR 2 for both strands, Default: 1 ]
[-t - Taxon (e or E - for E. coli, and c or C - for Cyanobacteria, Default: E ]
[-c - Search Criterion; if parC =
    70 ... search for sigma70 promoters (E. coli)
    38 ... search for sigma38 promoters (E. coli)
    32 ... search for sigma32 promoters (E. coli)
    28 ... search for sigma28 promoters (E. coli)
    24 ... search for sigma24 promoters (E. coli)
    A ... search for sigmaA promoters (cyanobacteria)
    C ... search for sigmaC promoters (cyanobacteria)
    F ... search for sigmaF promoters (cyanobacteria)
    G ... search for sigmaG promoters (cyanobacteria)
    H ... search for sigmaH promoters (cyanobacteria)
    1 ... search for the highest ranking promoter regardless of class in the chosen Taxon (option -t)
    5 ... [Default], report all promoter classes in the chosen Taxon (-t)
```

**Fig. 16** bTSSFinder command options

G4PromFinder is a genome-wide predictor of transcription promoters in bacterial genomes. It analyzes both the strands. It is recommended for GC-rich genomes

Input: Genome file  
 Output: a text file containing promoter coordinates and a file containing informations about them  
 Genome file must be in fasta format  
 Enter the input genome file name: █

**Fig. 17** G4PromFinder prompt to enter input genome file

- To run G4PromFinder, type in the terminal the following command:

```
python3 G4PromFinder.py
```

G4PromFinder will display a message on the terminal and then prompt the user to enter the filename of the FASTA file containing the query genome (Fig. 17) (*see Note 15*).

- G4PromFinder creates two files in the working directory: “ABOUT PROMOTERS.txt” and “Promoter coordinates.txt”. The “Promoter coordinates.txt” file contains the location of the predicted promoters.

## 4 Notes

- The sequence identifier in the BED files and the FASTA file must be exactly the same; otherwise, there will be run-time errors. The sequence identifier should not contain any whitespace.

2. Make sure this command is executed in the directory (folder) containing the sRNA-Detect script file; otherwise a “can’t open file” error message will be displayed.
3. Make sure nextflow is executed on the directory where the sRNACharP.nf file and the Nextflow config file are located. Additionally, note that the parameters to sRNACharP are given using a double dash (--) instead of a single dash. The complete command should be given in a single line.
4. Make sure the command is executed in a directory containing sRNARanking and the feature table generated by sRNACharP. Note that the first parameter is preceded by a double dash and the other parameters are preceded by a single dash.
5. CopraRNA requires a minimum of three sRNA sequences as input; however, four or more input sequences are recommended for better prediction quality.
6. In the CopraRNA webserver page is written that running CopraRNA can take around 24 h or more depending on the number of organisms, their genome sizes, and sequence lengths. Getting results from a job with eight organisms to predict the target of fnrS sRNA of *Escherichia coli* str. K-12 substr. MG1655 (NC\_000913) using default parameter settings took close to 7 h for us.
7. According to SPOT’s documentation (provided in [13]), SPOT can be run locally. However, as we were unable to obtain the installation files for all the sRNA target prediction methods used by SPOT, here we are only providing instructions to run SPOT using the Amazon Web Service (AWS) cloud Amazon Machine Image (AMI) setup provided by SPOT’s authors.
8. Depending on the application it may be possible to apply for AWS educate (<https://aws.amazon.com/education/awseducate/>) which provides AWS credits.
9. The only free instance available is “t2.micro” (1 Gb of RAM, 1 processor, and 30 Gb of storage); however, we were unable to run SPOT on this instance (after running for 8 h it stopped with a “Broken pipe” error message). We were able to run SPOT without CopraRNA (the default setting) on an instance with 30 GB RAM and 8 processors. Running on an instance with these specifications generates charges on the credit card provided. For information about AWS pricing see <https://aws.amazon.com/emr/pricing/> and <https://calculator.s3.amazonaws.com/index.html?s=EMR>. Make sure to terminate the instance by setting the instance state to Terminate. Be aware that closing the terminal after running SPOT will not terminate the instance. If the AWS instance is left running, charges continue to be applied according to the chosen instance.

10. One might need to change the file permission for the keypair file. To do this, on the .ssh directory type on the terminal chmod 400 keypairFile.pem.
11. The username is not the AWS username. There is a default username for each AMI type (<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-get-info-about-instance>). In our case, the corresponding username was ubuntu.
12. SPOT is distributed with test datasets available in the AWS instance under the directory example\_files.
13. It is suggested to use the screen -L command on the AWS server to ensure that jobs are not early aborted if the user account is logged out of the server. This command will precede the spot instruction.
14. These are the instruction for the bash shell.
15. Make sure the command is run on the same directory where the promoter finder program and the input genome file are located.

## Acknowledgments

Funding provided by a Discovery Grant to L.P.C. from the Natural Sciences and Engineering Research Council (NSERC).

## References

1. Hor J, Gorski SA, Vogel J (2018) Bacterial RNA biology on a genome scale. *Mol Cell* 70 (5):785–799
2. Dutta T, Srivastava S (2018) Small RNA-mediated regulation in bacteria: a growing palette of diverse mechanisms. *Gene* 656:60–72
3. Barquist L, Vogel J (2015) Accelerating discovery and functional analysis of small RNAs with new technologies. *Annu Rev Genet* 49:367–394. <https://doi.org/10.1146/annurev-genet-112414-054804>
4. Hook-Barnard IG, Hinton DM (2007) Transcription initiation by mix and match elements: flexibility for polymerase binding to bacterial promoters. *Gene Regul Syst Bio* 1:275–293
5. Turnbough CL Jr (2019) Regulation of bacterial gene expression by transcription attenuation. *Microbiol Mol Biol Rev* 83(3). <https://doi.org/10.1128/MMBR.00019-19>
6. Leonard S, Meyer S, Lacour S et al (2019) APERO: a genome-wide approach for identifying bacterial small RNAs from RNA-Seq data. *Nucleic Acids Res* 47(15):e88. <https://doi.org/10.1093/nar/gkz485>
7. Yu SH, Vogel J, Forstner KU (2018) ANNOgesic: a Swiss army knife for the RNA-seq based annotation of bacterial/archaeal genomes. *Gigascience* 7(9):giy096. <https://doi.org/10.1093/gigascience/giy096>
8. Pena-Castillo L, Gruell M, Mulligan ME et al (2016) Detection of bacterial small transcripts from Rna-Seq data: a comparative assessment. *Pac Symp Biocomput* 21:456–467
9. Eppenhof EJJ, Pena-Castillo L (2019) Prioritizing bona fide bacterial small RNAs with machine learning classifiers. *PeerJ* 7:e6304. <https://doi.org/10.7717/peerj.6304>
10. Pain A, Ott A, Amine H et al (2015) An assessment of bacterial small RNA target prediction programs. *RNA Biol* 12(5):509–513. <https://doi.org/10.1080/15476286.2015.1020269>
11. Wright PR, Richter AS, Papenfort K et al (2013) Comparative genomics boosts target prediction for bacterial small RNAs. *Proc Natl*

- Acad Sci U S A 110(37):E3487–E3496. <https://doi.org/10.1073/pnas.1303248110>
12. Wagner EG, Romby P (2015) Small RNAs in bacteria and archaea: who they are, what they do, and how they do it. *Adv Genet* 90:133–208. <https://doi.org/10.1016/bs.adgen.2015.05.001>
  13. King AM, Vanderpool CK, Degnan PH (2019) sRNA target prediction organizing tool (SPOT) integrates computational and experimental data to facilitate functional characterization of bacterial small RNAs. *mSphere* 4(1):e00561–e00518. <https://doi.org/10.1128/mSphere.00561-18>
  14. Sharma CM, Vogel J (2014) Differential RNA-seq: the approach behind and the biological insight gained. *Curr Opin Microbiol* 19:97–105. <https://doi.org/10.1016/j.mib.2014.06.010>
  15. Ettwiller L, Buswell J, Yigit E et al (2016) A novel enrichment strategy reveals unprecedented number of novel transcription start sites at single base resolution in a model prokaryote and the gut microbiome. *BMC Genomics* 17:199. <https://doi.org/10.1186/s12864-016-2539-z>
  16. Conway T, Creecy JP, Maddox SM et al (2014) Unprecedented high-resolution view of bacterial operon architecture revealed by RNA sequencing. *MBio* 5(4):e01442–e01414. <https://doi.org/10.1128/mBio.01442-14>
  17. Thomason MK, Bischler T, Eisenbart SK et al (2015) Global transcriptional start site mapping using differential RNA sequencing reveals novel antisense RNAs in *Escherichia coli*. *J Bacteriol* 197(1):18–28. <https://doi.org/10.1128/JB.02096-14>
  18. Guzina J, Djordjevic M (2016) Promoter recognition by extracytoplasmic function sigma factors: analyzing DNA and protein interaction motifs. *J Bacteriol* 198(14):1927–1938. <https://doi.org/10.1128/JB.00244-16>
  19. Zhu Y, Mao C, Ge X et al (2017) Characterization of a minimal type of promoter containing the –10 element and a guanine at the –14 or –13 position in mycobacteria. *J Bacteriol* 199(21):e00385–e00317. <https://doi.org/10.1128/JB.00385-17>
  20. Solovjev V, Salamov A (2011) Automatic annotation of microbial genomes and metagenomic sequences. In: Li RW (ed) Metagenomics and its applications in agriculture, biomedicine and environmental studies. Nova Science, Hauppauge, p 61
  21. Song K (2012) Recognition of prokaryotic promoters based on a novel variable-window Z-curve method. *Nucleic Acids Res* 40(3):963–971. <https://doi.org/10.1093/nar/gkr795>
  22. de Avila E Silva S, Echeverrigaray S, Gerhardt GJ (2011) BacPP: bacterial promoter prediction—a tool for accurate sigma-factor specific assignment in enterobacteria. *J Theor Biol* 287:92–99. <https://doi.org/10.1016/j.jtbi.2011.07.017>
  23. Umarov RK, Solovyev VV (2017) Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *PLoS One* 12(2):e0171410. <https://doi.org/10.1371/journal.pone.0171410>
  24. Shahmuradov IA, Mohamad Razali R, Bougouffa S et al (2017) bTSSfinder: a novel tool for the prediction of promoters in cyanobacteria and *Escherichia coli*. *Bioinformatics* 33(3):334–340. <https://doi.org/10.1093/bioinformatics/btw629>
  25. Di Salvo M, Pinatel E, Tala A et al (2018) G4PromFinder: an algorithm for predicting transcription promoters in GC-rich bacterial genomes based on AT-rich elements and G-quadruplex motifs. *BMC Bioinformatics* 19(1):36. <https://doi.org/10.1186/s12859-018-2049-x>
  26. Langmead B, Salzberg SL (2012) Fast gapped-read alignment with Bowtie 2. *Nat Methods* 9(4):357–359. <https://doi.org/10.1038/nmeth.1923>
  27. Li H, Durbin R (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics* 26(5):589–595. <https://doi.org/10.1093/bioinformatics/btp698>
  28. Anders S, Pyl PT, Huber W (2015) HTSeq—a Python framework to work with high-throughput sequencing data. *Bioinformatics* 31(2):166–169. <https://doi.org/10.1093/bioinformatics/btu638>
  29. Quinlan AR, Hall IM (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* 26(6):841–842. <https://doi.org/10.1093/bioinformatics/btq033>
  30. Di Tommaso P, Chatzou M, Floden EW et al (2017) Nextflow enables reproducible computational workflows. *Nat Biotechnol* 35(4):316–319. <https://doi.org/10.1038/nbt.3820>
  31. Liaw A, Wiener M (2002) Classification and regression by randomForest. *R News* 2(3):18–22



# Chapter 5

## Neuroevolutive Algorithms Applied for Modeling Some Biochemical Separation Processes

**Silvia Curteanu, Elena-Niculina Dragoi, Alexandra Cristina Blaga, Anca Irina Galaction, and Dan Cascaval**

### Abstract

Combining artificial neural networks with evolutive/bioinspired approaches is a technique that can solve a variety of issues regarding the topology determination and training for neural networks or for process optimization. In this chapter, the main mechanisms used in neuroevolution are discussed and some biochemical separation examples are given to underline the efficiency of these tools. For the current case studies (reactive extraction of folic acid and pertraction of vitamin C), the bioinspired metaheuristic included in the neuroevolutive procedures is represented by Differential Evolution, an algorithm that has shown a great potential to solve a variety of problems from multiple domains.

**Key words** Artificial neural networks, Differential evolution algorithm, Reactive extraction, Pertraction

---

## 1 Modeling Methodology Based on Neural Networks and Differential Evolution Algorithm

### 1.1 Artificial Neural Networks

In recent years, artificial neural networks (ANNs) have become preferred modeling tools in chemical engineering due to their ease in development and handling and the good results they frequently provide. Compared to classical modeling approaches (phenomenological modeling), ANNs do not require inner knowledge of the considered process, which is a real advantage, especially for complex processes whose mechanisms are not fully known. However, a basic condition in the development of good, credible neural models is related to the number and quality of available data. To evenly cover the search space and to provide the model with the necessary training exemplars, a sufficient quantity of data is required, depending on the complexity of the process and the objective of the study. Also, the experimental points must be evenly distributed in the problem space to cover the different characteristics of the process.

The efficiency of the neural model in solving a given problem also depends on other factors such as: type of neural network, method of determining its architecture and specific parameters values. From the multitude of existing types, multilayer neural networks with feed-forward propagation are the most used, proving in many situations (for various processes) the most suitable models.

The first step in using neural networks is to determine a suitable topology, optimal if possible (number of inputs, outputs; number of hidden layers; and neurons in each layer) and optimal internal parameters (weights, biases, and activation functions). This is an important stage that determines the result quality because it influences both the functionality and the performance of the models.

Structural and parametric optimization of neural networks is an essential step in model design. The system has mixed variables (the numbers of inputs, outputs, hidden layers; neurons in each intermediate layer; and type of transfer functions) which can all take a finite number of values, while weights and biases can take values in an unlimited range. For any given process, the neural network has specific inputs and outputs, determined by the data characteristics and modeling objective. The number of hidden layers and hidden neurons in them is determined partly heuristically and partly from repeated observations. Too large a neural network (too many layers and/or intermediate neurons) means that a large number of weights must be evaluated. For a standard ANN structure, it has been shown that neural networks with one or two hidden layers can, in most cases, provide accurate results. However, with the development of deep neural networks, higher performance models are being developed. In this chapter, the focus is on the standard ANN structure, a discussion regarding the specific aspects of deep ANNs deserving a separate section.

Various methods can be used to design a neural model [1], among them are: (1) trial and error, based on the creation of different topologies, with error evaluation at each step, choosing the topology to which the smallest error corresponds; (2) empirical or statistical methods, which study the influence of different internal parameters and choose the values that lead to the best performance; (3) hybrid methods, such as fuzzy inference; (4) constructive/destructive methods that develop the topology of a neural network by adding/removing neurons; and (5) evolutionary methods/bioinspired approaches, based on population optimization techniques and using principles found in nature. Each of these methods has advantages and disadvantages [1], so choice of the one to be used depends on the specific process, as well as prior user experience.

The use of *evolutionary algorithms* (EAs) to develop/test ANNs is known as neuroevolution. The area of EAs is in continuous development, nowadays EAs being included in a larger group

of algorithms that share the same principles but have various inspiration sources—nature-inspired metaheuristics (NIMs). Due to their similarities and general concepts, all NIMs can be combined with ANNs. However, the terminology indicating this combination has remained unchanged: *neuroevolution*. The evolutionary strategies applied to determine the topology of neural networks have the advantage of including elements such as: a balance between exploitation and exploration (at each step a stochastic selection is used for parents of the following generation), evaluation (via user definition of control parameters), parallelization (each individual can be trained independently), scalability (of calculation time and performance), and the possibility of bypassing local minima by specific methods [2].

The genetic algorithm (GA), one of the best known EAs, has been used to train and determine the topology of neural networks [3]. However, with the rising popularity of NIMs, other optimizers have started to be applied to ANN training/topology determination. Among these methods, Differential Evolution (DE) distinguished itself as a powerful and robust algorithm, with a small number of control parameters that can easily overcome some of the drawbacks of the GA. Although the number of DE + ANN applications in chemistry and biochemistry is not large, the results of existing applications suggest that this methodology has potential, being robust, flexible, and general, and easily adapted to different types of problems.

ANNs have proven useful in solving different types of problems and applications belonging to a range of different domains. Some examples that link neural networks to biochemical processes are dynamic modeling and temperature control of a yeast fermentation bioreactor [4, 5]; estimation of Lipozyme-catalyzed synthesis of wax ester from palm oil and oleyl alcohol [6]; approximation behavior of a fuel-ethanol fermentation process [7]; optimization of iturin A production [8]; modeling and optimization of ethanol production from glucose using *Saccharomyces cerevisiae* batch fermentation [9]; and temperature, pH, and stirring rate optimization of human soluble catechol O-methyl transferase in batch *Escherichia coli* culture [10].

## **1.2 Differential Evolution Algorithm**

The DE algorithm is a global optimization technique developed by Storn and Price in 1995 [11]. The basic principle is that of Darwinian evolution, according to which new solutions are introduced into the group of potential solutions by mathematically manipulating the existing ones. The algorithm has a simple and compact structure that uses ECT (evolutionary computation techniques) concepts in a stochastic direct search approach [12].

Like any EA, DE starts with a group of potential solutions randomly initialized. This initialization, is important because it has a significant influence on the final results. The next step,

mutation, involves the creation of new individuals and, in the classical version, is achieved by adding to the base vector (individual) a scaled differential term:

$$\omega_i = \alpha + F \cdot \beta \quad (1)$$

where  $\alpha$  is the base vector,  $\beta$  is the differential term, and  $F$  is the scaling factor. The scaling factor is an important parameter of the algorithm because it controls the speed at which the population evolves. The base vector can be selected at random (Rand) from the current population (target population) or as the best solution within it (Best). The mutation can be achieved by adding one, two or more terms to the base vector. Depending on these elements and how they are combined, different versions of DE can be identified, for example DE/Rand/2, DE/Best/2 etc.

Crossover is the stage in which new individuals are created based on the current population and the mutation vectors obtained in the previous step. The result of this process is a new population called the trial population. Both stages (mutation and crossover) contribute to the creation of the new population, mutation realizing diversity through its random nature and crossover defining the construction of the function. In general, two crossover variants, binomial or exponential, can be applied, differentiated by the position of characteristics from distinct individuals. In binomial crossover, components inherited from the mutant vector are arbitrarily selected and in exponential crossover they are grouped into one or two compact sequences [13].

At this stage, another important control parameter of the algorithm intervenes, the crossover rate ( $C_r$ ), which provides the means to exploit decomposability and to provide extra diversity [14]; its optimal value depends on the process being used and the crossover variant. The optimal values of  $F$  and  $C_r$  are correlated through an inter-dependency that is not yet fully understood, a fact that makes the determination of these parameters a difficult task.

In the selection step, individuals from the current and trial populations compete with each other to survive to the new generation. This type of selection is called one-to-one survivor selection.

The mutation, crossover and selection steps are repeated until a stop condition is fulfilled. Such a criterion can be represented by a predefined number of generations (iterations) or by a value assigned to the error. The purpose of the evolution is to explore the search space and to locate the minimum/maximum of the objective function. Finally, the solution is chosen as the individual with the best fitness function [15].

By combining different types of mutation, crossover, recombination, and stopping criteria, or introducing new methods into the algorithm itself, new DE variants (strategies) are created, which gives it flexibility and specificity.

Belonging to the class of evolutionary algorithms, DE has many elements similar to these; however, there are also significant differences—the roles of mutation and recombination are different. For an EA working with real values, mutation is a change with a random element and its effects are simulated with randomly generated increments through a predefined probability distribution function (PDF). In its classical version, DE uses a uniform PDF to randomly sample vector difference [14]. In an EA, crossover is used to combine the characteristics of different parents, whereas in DE, because the mutation is based on the recombination of individuals, the role of crossover consists in the construction of offspring [13].

Over time, in order to obtain a more robust and higher performance, many improvements have been proposed to DE. These follow several directions: (1) replacing the manual determination of the control parameters by adaptive or self-adaptive mechanisms; (2) introduction of new mutation strategies; and (3) hybrid DE combinations with other optimization techniques, especially local ones, for search refinement.

Regarding the first direction, the control parameters play an important role in DE, maintaining a balance between exploration and exploitation and, thus, in determining the effectiveness of the search. Various adaptive or self-adaptive methods have been proposed. For example, Thangaraj et al. [16] developed a DE algorithm called ACDE in which the selection of control parameters is automatic. Lu et al. [17] combined the adaptive dynamic mechanism of control parameters with a local random search operator to prevent premature convergence. Pat et al. [18] designed a DE algorithm with a self-adaptive trial vector generation strategy and control parameters. Each individual in the population was associated with a strategy list, a mutation-scaling list and a crossover rate list that were modified during the evolution process.

In biochemical engineering, DE, in simple variants or hybridized with other approaches, has been used for dynamic optimization of simultaneous saccharification and fermentation of starch in lactic acid [19], determination of temperature control leading to maximization of productivity and final bacteriocin titer in a system developed for bacteriocin production in *Lactococcus lactis* C7 [20], and online optimization of feed batch recombinant *Escherichia coli* fermentation, and hybridoma reactor and feed batch bioreactor for ethanol production using *Saccharomyces cerevisiae* [21].

### **1.3 Development of an Optimal Neural Network with DE**

Being a global search algorithm, suitable for large continuous spaces, DE can be efficiently and easily applied to determine the topology and parameters of a neural network. In addition, it is a powerful algorithm, with a fast convergence rate.

The evolution of ANNs (the determination of ANNs with evolutionary methods) can be achieved at three levels [22]:

(a) determination of connection weights, in other words, network training which is done by minimizing an error function, such as mean square error (MSE); (b) automatic determination of the network architecture in optimal or near optimal form; (c) adaptation of learning rules during evolution.

Evolution of the ANN may involve any one of a variety of combinations; the most common are (a) weights evolution (when a fixed topology is considered), (b) architecture evolution (where the training is performed by another non-EA algorithm such as Backpropagation); and (c) weight and architecture evolution.

In order for DE to work in the search space described by a neural network, it is necessary to apply an encoding that transforms the phenotype (the ANN characteristics considered for evolution) into genotype (an orderly structure formed from numbers, binary or otherwise). Literature presents various types of encoding that can be classified into direct, developmental, implicit, and indirect encoding methods [23]. In a direct coding scheme, each neuron and connection pair are explicitly specified (one-to-one mapping between the phenotype and genotype). The developmental representation is based on the genome used to direct the developmental process [24]. The third option, indirect coding, attempts to apply the principle of gene reuse in biological development [25].

When the neuroevolutive process is applied (the NIM is used to evolve the ANN), the information between DE and ANN flows in two ways: from DE to network and vice versa. At each generation, new individuals are created representing the encoded neural networks and each network (determined after decoding individuals) calculates the fitness function for DE. Then the algorithm uses this fitness function to determine the best individuals for the next generation.

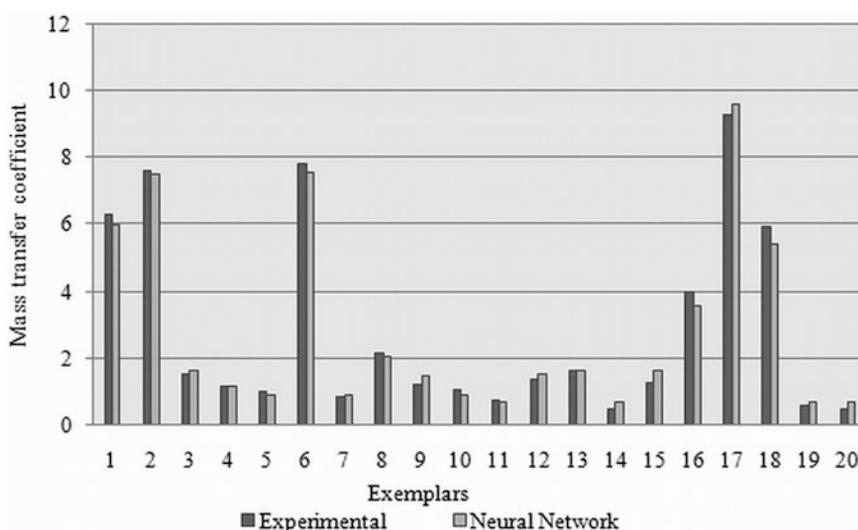
One of the first uses of the DE algorithm for neural network optimization was described by Fisher et al. [26] who intended to find a new training method. Plagianakos et al. [27] apply DE to train neural networks with discrete activation functions. Subudhi and Jena [12, 15] combined DE and Levenberg Marquardt algorithms to obtain a hybrid method of training a neural network for identifying nonlinear systems. An attempt to determine a partial ANN architecture using DE was made by Lahiri and Khalfé [28], the network being built by determining the best values for the number of neurons in the hidden layers, weights and activation functions. Bhuiyan [29] proposed a DE-based algorithm and Back-propagation algorithm to obtain the topology of a neural network and to perform partial training during evolution.

## 2 Differential Evolution-Based Neuroevolution Applications

In [30], DE is applied to determine the architecture and internal parameters of a neural network that has the role of modeling the mass transfer coefficient of oxygen in mixing bioreactors, based on the fact that oxygen is a decisive factor in the aerobic biosynthesis. The mass transfer coefficient of oxygen is correlated with viscosity, superficial air flow, specific power, and volumetric fraction of oxygen. Two DE variants, the classic variant with a self-adaptive mechanism and three initialization strategies (Random, Gauss, and Halton) were applied to individual or stack neural networks. The good results obtained proved the efficiency and flexibility of the method. Figure 1 shows an example of results, being a comparison between experimental data and those predicted by DE in the testing phase.

A variant of DE called by the authors SADE-NN was applied to determine the optimal architecture and parameters of neural networks for the classification of organic compounds according to their liquid crystal property [31]. SADE-NN has the following characteristics: the base vector is selected as the best individual in the population, the mutation includes two differential terms, the crossover is binomial, and a self-adaptation mechanism has been applied to determine the algorithm parameters. The integration of the neural network with DE was achieved through a direct coding scheme. Prediction of the liquid crystal property was correct with a probability of 90%.

Another variant called SADE-NN-1 was applied to predict the mass transfer coefficient of oxygen in mixing bioreactors,



**Fig. 1** Experimental data and predictions of a stack neural network for the mass transfer coefficient of oxygen

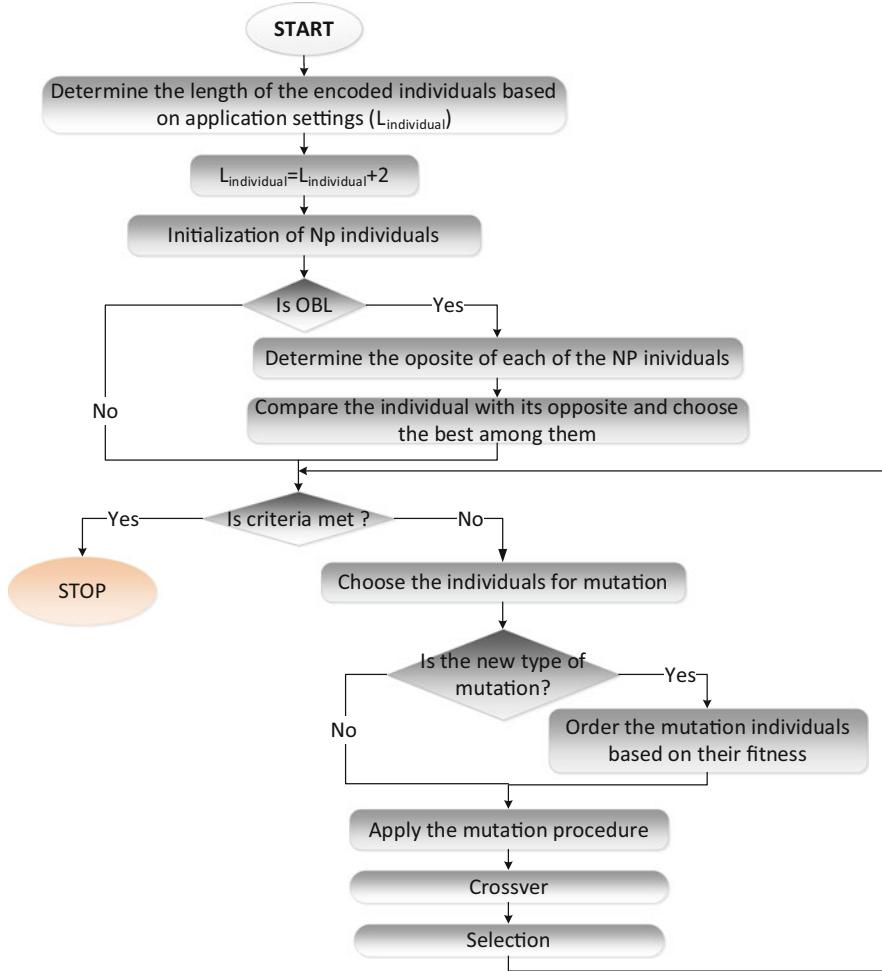
depending on the conditions under which the biochemical process takes place: biomass concentration, superficial air flow, volume fraction of oxygen and specific power. The same approach was used to optimize both the topology of a feed-forward neural network and the process itself [32] in two cases: bacteria fermentation and yeast fermentation. SADE-NN-1 is a combination of DE with ANN in which DE has the following characteristics: (a) it includes a self-adaptive mechanism in which the control parameters  $F$  and  $C_r$  are evolved with the population itself, the new values being determined by the same equations as the ones used to create new individuals; (b) eight types of activation functions were used: Linear, Hard Limit, Bipolar Sigmoid, Logarithmic Sigmoid, Tangent Sigmoid, Sinus, Radial Basis with a fixed center point at 0, and Triangular Basis; (c) two types of initialization have been applied: normal distribution and normal distribution combined with OBL (Opposition Based Learning, using the opposite number theory [33]). Initially, the population was generated as usual, with normal distribution. Then, the opposite individuals were determined using Eq. 1, the initial population being formed by the individuals with the best fitness function from the two groups. As the initial population quality is improved, the probability of obtaining a better performance at the end of the evolution process is raised; and (d) an altered mutation strategy has been used that consists in ordering the individuals according to their fitness value, given that better individuals will create better children.

$$\tilde{x}_i = i_{\min} + i_{\max} - x_i \quad (2)$$

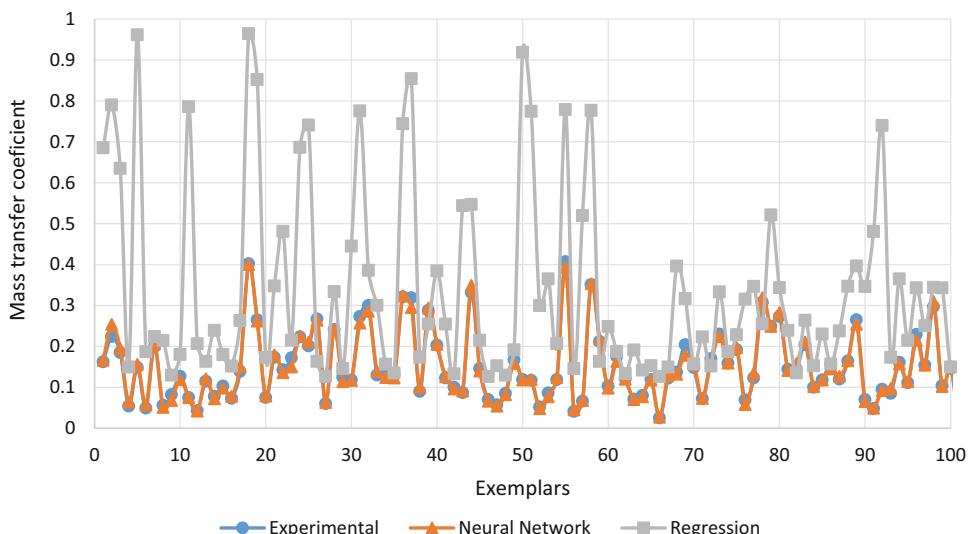
where  $i$  represents the  $i$ th parameter of the individual  $x$ ,  $\tilde{x}$  is the opposite of  $x$ , and  $i_{\min}$  and  $i_{\max}$  are the minimum and the maximum values allowed for the parameter  $i$ .

Figure 2 represents a simplified scheme of the SADE-NN-1 methodology, applied to determine the optimal topology of an MLP-type neural network.

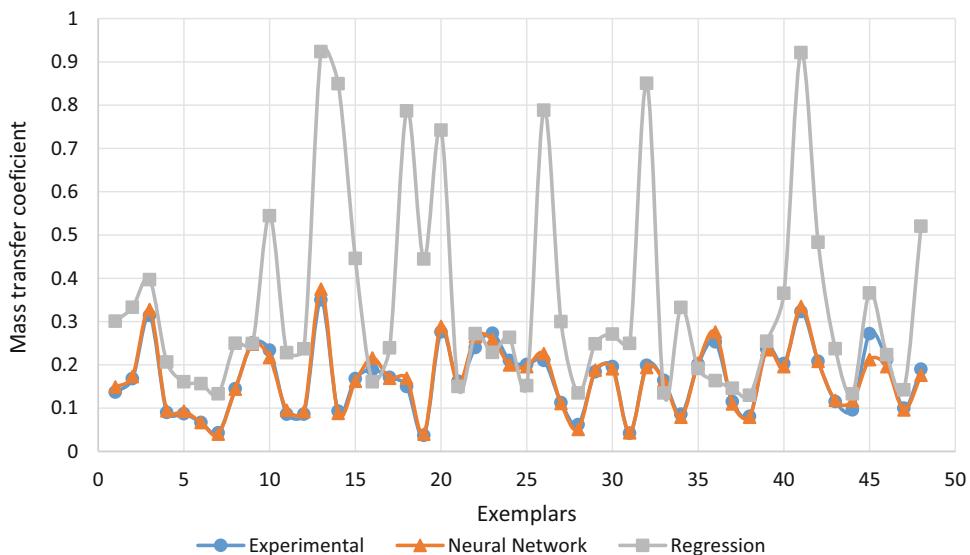
The result of running this software is an MLP network (4:5:1) that recorded the following performances:  $r$  (correlation between network predictions and experimental data) at training = 0.9878 and RMSE (root mean squared error) at training = 0.0219 (Fig. 3),  $r$  for testing = 0.9563 and RMSE for testing = 0.0353 (Fig. 4). Applying the multiple regression in Matlab, another type of model has been determined with the following performances during training and testing:  $r = 0.2329$  and RMSE = 0.2172;  $r = 0.2214$  and RMSE = 0.2125, respectively. Figures 3 and 4, in which the comparison is made point by point (experimental, ANN determined with DE and regression model in case of yeast fermentation), as well as the performances reflected by the correlation and error values, show that the neural model gives better results than the regression model.



**Fig. 2** Simplified scheme of the SADE-NN-1 methodology



**Fig. 3** Comparison between experimental data, MLP network predictions, and regression model predictions in the training phase (representative set of examples) for the mass transfer coefficient of oxygen in mixing bioreactors



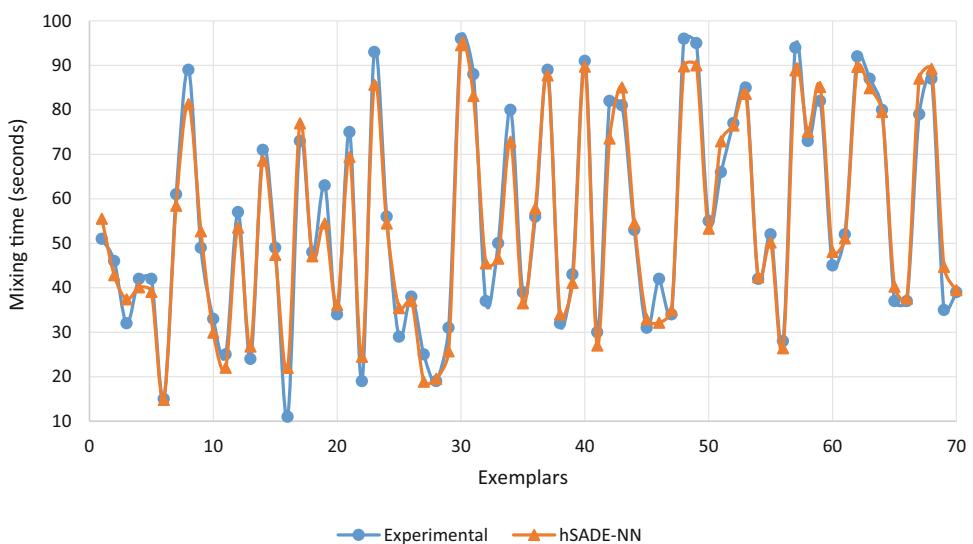
**Fig. 4** Comparisons between experimental data, MLP network predictions, and regression model predictions in the testing phase for the mass transfer coefficient of oxygen in mixing bioreactors

Another variant of DE called SADE-NN-2 was applied to the freeze-drying process of some pharmaceutical products [34]. The methodology is based on a simple DE algorithm, with self-adaptation combined with the Backpropagation algorithm. Therefore, two training procedures are applied to the neural network: first DE, which acts as a global search algorithm, and then Backpropagation which performs a local search, improving only the best solution obtained at the end of each generation. The characteristics of the DE optimizer in this version are the same as the ones from SADE-NN-1.

Regarding the freeze-drying process [33], ANN provides information on the state of the product depending on the operating conditions: heating fluid temperature, time and pressure in the drying chamber. The state of the product is identified by the temperature at the sublimation interface and by the thickness of the drying layer (the two outputs of the model). In fact, the effect of operating conditions on the state of the product depends on the state of the product itself, more exactly on what happened with the product in the past. Therefore, for rigorous modeling, a recurrent neural network was considered. In this example, SADE-NN-2 was used for modeling and monitoring the freeze-drying process. The utility of the application derives from the possibility of using the obtained model for the off-line determination of the “best” recipe (the shortest drying time) and for the inline optimization of the process, for example, in the Model Predictive Control algorithm. Within the monitoring, the neural model is a soft-sensor designed to estimate variables that cannot be measured in line (dry layer

thickness) based on the available measurements (product temperature).

hSADE-NN is another DE based variant that represents an improvement of SADE-NN-2. It includes a local optimization composed of Backpropagation and Random Search, with a role in refining the search and avoiding the local minima, thus improving, for each generation, the best solution found by DE. The motivation for using two local algorithms instead of one is related to the fact that sometimes the local optimization procedure becomes stuck and is not able to find the best solution. Using two approaches applied randomly increases the chance of eliminating such a problem. hSADE-NN was applied to a biochemical process—fermentation with *Yarrowia lipolytica* cells, with lipids as substrate, and taking place in a split-cylinder gas-lift bioreactor [35]. The main factors controlling the performance of gas-lift pneumatic bioreactors are nutrient and oxygen distribution inside the broths, temperature gradient, and extent of the shear stress. The uniform distribution of these parameters in the gas-lift bioreactor depends directly on the homogeneity of the mixing intensity. Thus, the neural model designed with hSADE-NN predicts mixing time (as a measure of mixing efficiency) based on yeast concentration, aeration rate and position on the riser or downcomer regions heights. An average absolute relative error less than 8.5% is obtained in modeling, which proves the efficiency of the applied procedure. Figure 5 reflects the very good match between the predictions of the optimal neural model, MLP (4:18:1), (a simple feed-forward neural network, with a single intermediate layer), and experimental data.



**Fig. 5** Comparison between experimental data and predictions obtained by MLP (4:18:1) for testing data, in modeling the mixing efficiency of a gas-lift bioreactor

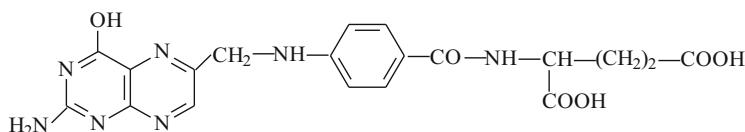
### 3 Practical Examples

Two examples from the biochemical field will be considered—reactive extraction of folic acid and vitamin C pertraction, for which experimental data were obtained in our laboratory. For modeling purposes, neural networks were developed using the variants of DE discussed above; comparison between the results obtained allows the selection of the best variant of DE methodology (consequently, the best neural network) for the available data.

#### **3.1 Reactive Extraction of Folic Acid (Vitamin B9)**

Separation and efficient concentration of a biosynthesis product, primary or secondary metabolite, is a complex problem, requiring a detailed analysis of the structural particularities of the compound considered in order to choose an appropriate method. Due to the multitude, diversity, and importance in human nutrition and human health, this category of compounds has always attracted the attention of scientists. Natural products for use in the pharmaceutical and food industry can be isolated directly from plants and animal organisms or may be the result of a biosynthesis process inside a bioreactor. The use of microorganisms for biochemical transformations in a biotechnological process offers the opportunity to produce a wide range of essential products, biologically active compounds from accessible and renewable sources. Separation of such a product is difficult because they typically have a complex structure, and may be thermally and chemically labile. To optimize the separation process, a new approach, namely reactive extraction, has been proposed as a selective route for the recovery of bioproducts [36–45].

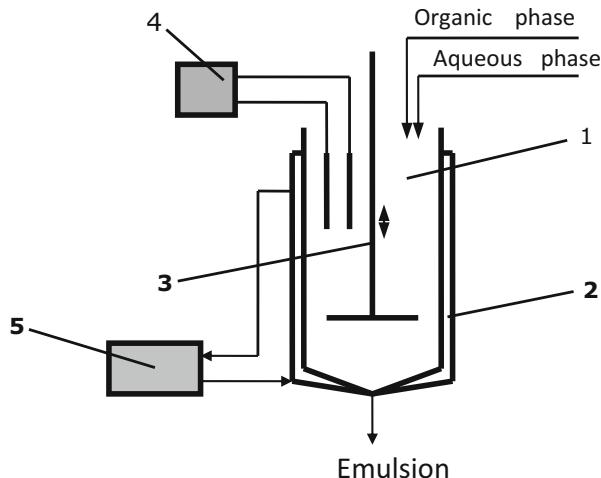
Folic acid, also called pteroylglutamic acid (Fig. 6), is part of the vitamin B group (vitamin B9), which plays an essential role in the synthesis of nucleic acids and some amino acids. It is widespread in plants, being isolated for the first time from spinach leaves in 1941. It has also been identified in bacteria extracts, proving its need for growth of these microorganisms, as well as in various organs of mammals. Folic acid can be obtained by a biosynthesis process, the product obtained having a bioavailability superior to that obtained by synthesis. Thus, exploiting the high metabolic capacity of *B. subtilis* mutant strains to biosynthesize folic acid on the glucose substrate, conversion of 0.16 moles of folic acid/100



**Fig. 6** Folic acid chemical structure

**Table 1**  
Dielectric constant values for solvents used at 20 °C

Solvent	n-Heptane	Chloroform	Dichloromethane
Dielectric constant	1.90	4.806	9.08



**Fig. 7** Extraction column (1—glass column; 2—thermostat jacket; 3—vibratory stirrer; 4—digital pH meter; 5—thermostat)

moles of glucose was achieved, about tenfold greater than that of wild strains.

As folic acid is insoluble in organic solvents, its separation by physical extraction is impossible. Accordingly, a series of experiments on the separation of folic acid by reactive extraction with Amberlite LA-2 was developed. Since the polarity of the solvent is a determining factor for extraction efficiency, a study of the mechanism and factors controlling the process was carried out in correlation with the polarity of three different solvents (Table 1).

This experimental study was carried out using an extraction column (Fig. 7) with vibratory mixing that offers the advantage of a very large interfacial contact area between phases. The resulting emulsion was removed at the bottom of the extraction column, and then separated into a centrifugal separator at a rate of 4000 rpm. Folic acid was extracted from aqueous solutions with an initial concentration of  $5.3 \times 10^{-2}$  g/l. The extraction was performed with solvents having a significantly different dielectric constant, in which Amberlite LA-2 was dissolved. The concentration of the extractant in organic solvents varied between 10 and 160 g/l. The volumetric ratio of the aqueous phase and the organic phase was 1 (20 ml of each phase). The pH of the aqueous phase was 5.2.

The extraction process was analyzed by means of folic acid concentrations, distribution coefficients, and extraction degree. For these parameters, the folic acid concentrations in the initial solution and raffinate were measured by high performance liquid chromatography (HPLC) with a Shim-pack CLC-ODS column (6 mm diameter and 150 mm length) and UV detector at 210 nm. The mobile phase consisted of a solution of 100 mM phosphate buffer (pH = 2.1) and 0.8 mM sodium octane sulfonate/acetonitrile in a volume ratio of 9/1. Determinations were made at 40 °C, the flow rate of the mobile phase being 1.5 ml/min.

The reactive extraction of folic acid,  $R(COOH)_2$ , with Amberlite LA-2, Q, occurs by means of an interfacial reaction with the formation of a strong hydrophobic compound. The interaction between the acid and the extractant could be of hydrogen bonding type, with undissociated carboxylic groups, or by ion-pair formation, if the acid dissociates in the aqueous solution. Furthermore, the compounds of acidic ( $[R(COOH)_2]_nQ$ ) or aminic ( $R(COOH)_2Q_n$ ) adducts type could be formed at the interface, depending on the chemical characteristics of the extraction system components and solvent polarity [46].

Due to the voluminous chemical structure of folic acid, which induces steric hindrances, as well as the much lower initial concentration of the acid compared to the extractant, no acid adducts are formed at the interface, with the possibility of formation of either compounds of type  $R(COOH)_2Q$  or  $R(COOH)_2Q_2$ , by neutralizing one or both of the -COOH groups of the folic acid structure, or of the type of amino adducts,  $R(COOH)_2Q_n$ , where  $n > 2$ . Formation of these molecular associations, in solvents with low dielectric constant, increases the hydrophobicity of the compound formed by the interfacial reaction.

An important influence on the reactive extraction is the pH of the aqueous phase. A study of the separation of folic acid by Amberlite LA-2 reactive extraction in three solvents with different polarities (n-heptane, chloroform, and dichloromethane) at different concentrations of the extractant and different pH values of the aqueous phase indicated that folic acid reactive extraction with Amberlite LA-2 is achieved by means of hydrogen bonds between the nondissociated carboxyl groups of the solute and the extractant established at the phase interface. The aqueous phase pH has a decisive role in the reactive extraction mechanism. By controlling the dissociation of the carboxyl groups, it can determine the molar ratio in which the folic acid reacts with the extractant (1:2 in the pH range  $< pK_1$ ,  $k_1 = 2 \times 10^{-5}$ , 1:1 for  $pK_1 < pH < pK_2$ ,  $k_2 = 1.58 \times 10^{-7}$ ). In both situations, for the interfacial compound formed, solubilization in the organic phase is accomplished by solvation by the extractant, a process favored by the increase of the dielectric constant of the solvent.

The solvent polarity exhibits a significant influence on the extent of reactive extraction of folic acid, its increase inducing the enhancement of separation process efficiency: for Amberlite LA-2 concentration of 80 g/l, the extraction degree in dichloromethane was about 2–2.7 times greater than that for extraction in n-heptane, this difference being increased by increasing the pH of the aqueous solution.

In this case, the efficiency of the process is modeled as a function of solvent type, pH and concentration of extractant. The solvent type is encoded using the dielectric constant (Table 1). After that, the available data (75 exemplars) was randomized, split into training (75%) and testing (25%) sets, and normalized. Randomization changes the position of each exemplar by randomly assigning them to one of the sets used for training or testing. The normalization has the role of translating all the inputs and outputs in the [0,1] interval. For all the three variants (SADE-NN-1, SADE-NN-2, and hSADE-NN), a set of 50 simulations was performed, the main statistical indicators being presented in Table 2. The settings used to generate the results from Table 2 are the following: number of iterations 100, number of individuals in the population (population size) 40, and for the topology, maximum 2 hidden layers with 20 and 10 neurons in the first and second hidden layers, respectively.

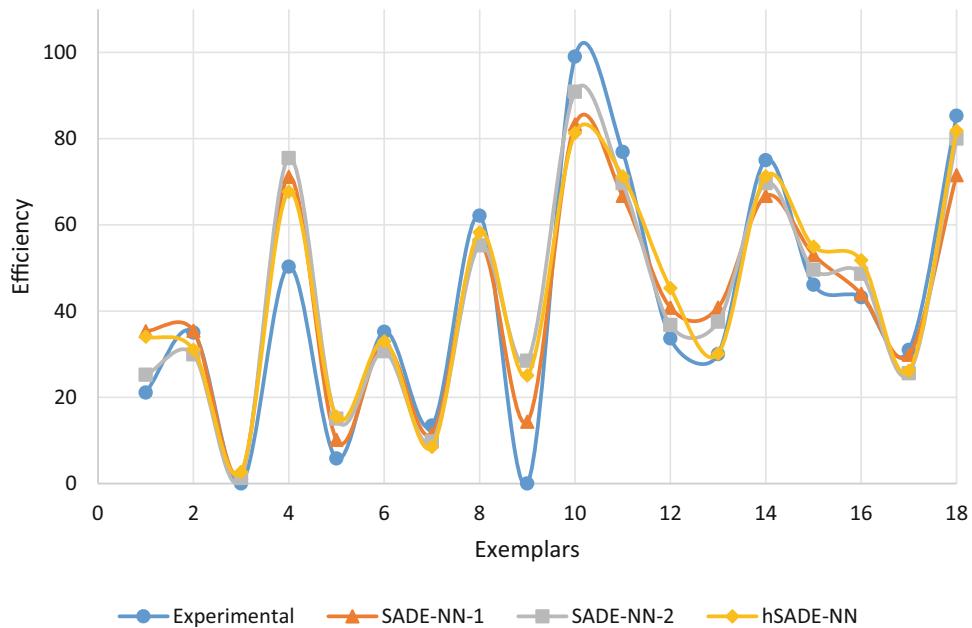
Table 2 shows that the gradual improvements to the base DE version are translated into improvements of performance of the models. This is also evident in the point by point comparison between the predictions generated with the three DE methods and experimental data (Fig. 8).

### **3.2 Facilitated Pertraction of Vitamin C**

Vitamin C, also called ascorbic acid, is one of the most well-known and studied vitamins, being biosynthesized by microbial, plant and animal organisms, except primates and guinea pigs. It was

**Table 2**  
**Statistical indicators for the determined models**

		<b>Fitness</b>	<b>MSE train</b>	<b>MSE test</b>	<b>Correl train</b>	<b>Correl test</b>	<b>Topology</b>
SADE-NN-1	Best	1681.112	0.000595	0.002207	0.967769	0.951965	3:04:01
	Worst	281.7915	0.003549	0.005189	0.788949	0.881509	3:07:01
	Average	736.6789	0.0017	0.006373	0.914036	0.839741	
SADE-NN-2	Best	2498.409	0.0004	0.002479	0.978296	0.937262	3:08:01
	Worst	495.8834	0.002017	0.009901	0.917013	0.762926	3:08:01
	Average	1357.497	0.00083	0.003909	0.956501	0.902348	
hSADE-NN	Best	3052.895	0.000328	0.002451	0.982909	0.942142	3:10:01
	Worst	371.8121	0.00269	0.003507	0.863568	0.906697	3:14:01
	Average	1736.495	0.000664	0.003045	0.965687	0.923807	



**Fig. 8** Comparison between experimental data and predictions generated with the three DE-based approaches for testing data in reactive extraction of vitamin B9

discovered by Albert Szent-Gyorgyi, a discovery for which he received the Nobel Prize in 1937. It is one of the most important vitamins, essential in human nutrition, with use in medicine and nutrition. It is a water-soluble vitamin involved in the metabolism of glucose, collagen, folic acid and certain aminated acids in the neutralization of free radicals and nitrosamines in immunological reactions that facilitate the absorption of iron by the digestive tract, constituting a cofactor for a series of enzymes. In the human body, vitamin C plays a complex and important role due to its antioxidant character. In this regard, it protects numerous biologically active compounds from oxidative degradation and strengthens the immune system. The lack of vitamin C in human food produces a disease called scurvy. In cosmetics, vitamin C is a valuable ingredient of many lotions and creams, due to its antioxidant and protective effect against UV rays. The main sources of vitamin C are vegetables and fruits, in which cells play a vital role in metabolic and growth processes.

Vitamin C is obtained by extraction from various plant materials, by chemical synthesis, by biosynthesis or by combined chemical synthesis and biosynthesis. At the industrial level, two processes are applied, namely biosynthesis coupled with chemical transformation (Reichstein process) and two-stage fermentation, with a worldwide production of over 70,000 tons/year, the largest producer being China (43,000 tons/year). Regardless of the method used for

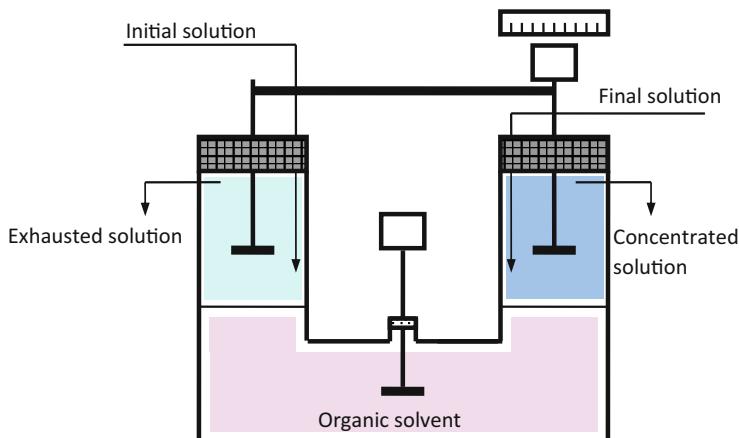
obtaining vitamin C, its recovery from the fermentation liquid is quite difficult.

Pertraction, that is extraction and transport through liquid membranes, is a relatively new technique for the separation of natural products. Pertraction can overcome the limitations imposed by direct contact between the organic and aqueous phase specific to extraction, namely, formation of stable emulsions and the need to regenerate the organic solvent, thus maintaining the advantages of continuous operation.

Pertraction is a combination in time and space of two well-known separation operations: liquid-liquid extraction and reextraction, offering significant advantages over the conventional extraction process. Liquid membranes are homogeneous membranes that function on the principle of dissolving a solute at the interface between the initial aqueous phase and the liquid membrane and releasing it at the interface between the membrane and the final aqueous phase, based on the concentration gradient between the interfaces. While solvent extraction is an equilibrium process, pertraction is governed by membrane transport kinetics. The amount of solute carried is not proportional to the membrane phase, as in the case of extraction, the liquid membrane being just a short-term mediator. However, in order to select the membrane solvent, several properties should be considered: hydrophobicity—to ensure immiscibility with aqueous phases, viscosity—to ensure high speeds of mass transfer (diffusion of solute), density, vapor pressure, but especially the dielectric constant, because one of the most important parameters that control the separation performance is the polarity of the solvent.

Compared with classical separation techniques, the use of liquid membranes presents a number of advantages: (1) use of small quantities of solvents, that are continuously regenerated; (2) reduction in the total time required to separate a product by combining extraction with reextraction; (3) the possibility of transporting a solute against its concentration gradient, if the gradient difference of the property controlling the process is maintained; (4) obtaining higher mass flows compared to polymeric or inorganic membranes due to the higher diffusion coefficients in liquids (by several order of magnitude); (5) high selectivity due to the use of a wide range of specific interactions in the membrane by the use of selective extracts; (6) low energy consumption; (7) compact installations; (8) low investment costs.

The experiments have been carried out using pertraction equipment that allows one to obtain and easily maintain the solvent layer between the two aqueous phases (free liquid membrane) [47]. The pertraction cell (Fig. 9) consists of a U-shaped glass pipe having an inner diameter of 45 mm and a total volume of 450 ml, the volume of each compartment being 150 ml.



**Fig. 9** Pertraction cell

The aqueous solutions are independently mixed by means of double blade stirrers of 6 mm diameter and 3 mm height, having a rotation speed of 500 rpm. In order to reach high diffusional rates through the solvent layer, the organic phase is mixed with a similar stirrer, at the same rotation speed. The area of mass transfer surface, both for extraction and for reextraction, was  $1.59 \times 10^{-3} \text{ m}^2$ . The interfaces between the phases remained flat and, hence, the interfacial area constant for the rotation speed used.

The experiments have been carried out in a pseudo steady-state regime, at the steady-state conditions related to the aqueous phases and unsteady-state mode related to the membrane phase. The aqueous solutions have been separately fed with a volumetric flow of 2.0 l/h, at 25 °C.

The liquid membrane phase consisted of dichloromethane in which was dissolved the carrier Amberlite LA-2, with concentrations between 0 and 100 g/l. The feed phase was an aqueous solution of 7.06 g/l vitamin C, and its pH-value varied between 2 and 6, adjusted with a solution of 4% sulfuric acid or 4% sodium hydroxide. The stripping phase consisted of a solution of sodium hydroxide with pH = 8–12.

The pertraction has been analyzed by means of initial and final mass flows and permeability factor. The initial mass flow represents the solute mass flow from the feed phase to the liquid membrane, while the final (overall) mass flow is the mass flow from the liquid membrane to the stripping phase. The permeability factor has been defined as the ratio between the final mass flow and the initial mass flow of solute. These parameters have been calculated by determining the compound's concentrations in the feed and stripping phases and by using the mass balance for the pertraction system. Concentrations in the aqueous phases have been measured by HPLC (Hamilton PRP-X300 column (150 mm × 4.1 mm, 5 µm), 4 mM sulfuric acid solution as mobile phase, detection being performed by UV absorbance at a wavelength of 210 nm, and flow rate of 0.5 ml/min).

Generally, the facilitated pertraction is strongly influenced by the pH-gradient between the feed and stripping phases (which controls the rates of extraction and reextraction processes), as well as by the carrier concentration inside the liquid membrane.

The general mechanism of facilitated pertraction of vitamin C with Amberlite LA-2 implies three successive steps: (1) the reactive extraction of vitamin C at the interface between the feed phase and solvent phase (liquid membrane) by interfacial formation of a solvated ammonium salt; (2) the diffusion of the interfacial product through the liquid membrane; and (3) the reextraction of vitamin C at the interface between the organic phase and stripping phase by interfacial formation of sodium ascorbate or dehydroascorbic acid.

In order to obtain the maximum values of the initial and final solute mass flows, a pH-value of feed phase below 2, pH-value of the stripping phase above 11 and Amberlite LA-2 concentration over 80 g/l are required. These conditions can be also applied for reaching the maximum value of the permeability factor, except that the value of the pH for the feed phase must be higher than 5.

In this case, the initial mass flow and the final mass flow are determined as a function of the pH of initial aqueous phase, pH of the final aqueous phase and of the amberlite LA2 concentration. The same approaches (SADE-NN-1, SADE-NN-2, and hSADE-NN) and settings as in the previous case study were used. The main objective was to show that gradual improvements to the DE base algorithm lead to gradual changes in the performance and capacity of the overall methodology to find good solutions.

The main statistical indicators for the models determined are presented in Table 3, where MSE indicates the mean squared error and Correl is the correlation.

**Table 3**  
**Statistical indicators for the determined models**

	<b>Fitness</b>	<b>MSE</b>		<b>Correl</b>		<b>Correl</b>		<b>Topology</b>
		<b>train</b>	<b>test</b>	<b>train</b> <b>output 1</b>	<b>train</b> <b>output 2</b>	<b>test</b> <b>output 1</b>	<b>test</b> <b>output 2</b>	
SADE- NN-1	Best	79.655	0.013	0.039	0.174	0.823	0.106	0.586 4:20:02
	Worst	37.657	0.027	0.040	-0.260	-0.173	0.632	-0.055 4:04:02
	Average	64.630	0.016	0.034	0.322	0.277	0.580	0.332
SADE- NN-2	Best	282.805	0.004	0.011	0.927	0.896	0.955	0.926 4:07:02
	Worst	56.260	0.018	0.045	-0.079	0.144	0.367	-0.595 4:18:02
	Average	106.759	0.010	0.029	0.605	0.560	0.607	0.609
hSADE- NN	Best	488.661	0.002	0.012	0.947	0.933	0.917	0.941 4:07:02
	Worst	45.783	0.022	0.036	0.312	0.435	0.876	-0.553 4:18:02
	Average	136.799	0.009	0.023	0.672	0.693	0.712	0.677

**Table 4**  
**Statistical indicators of the determined models for the initial mass flow**

		<b>Fitness</b>	<b>MSE train</b>	<b>MSE test</b>	<b>Correl train</b>	<b>Correl test</b>	<b>Topology</b>
SADE-NN-1	Best	298.5722	0.003349	0.016277	0.904115	0.82619	4:09:01
	Worst	57.75723	0.017314	0.033852	0.236269	0.547947	4:05:01
	Average	84.32629	0.013313	0.028141	0.538885	0.640994	
SADE-NN-2	Best	335.7415	0.002978	0.017336	0.934008	0.855111	4:04:01
	Worst	65.51115	0.015265	0.02644	0.409363	0.698283	4:14:01
	Average	99.63227	0.011597	0.029623	0.615431	0.659654	
hSADE-NN	Best	498.3178	0.002007	0.004506	0.946347	0.959604	4:14:01
	Worst	82.70078	0.012092	0.028037	0.603355	0.646549	4:17:01
	Average	274.7083	0.004665	0.011944	0.878663	0.89021	

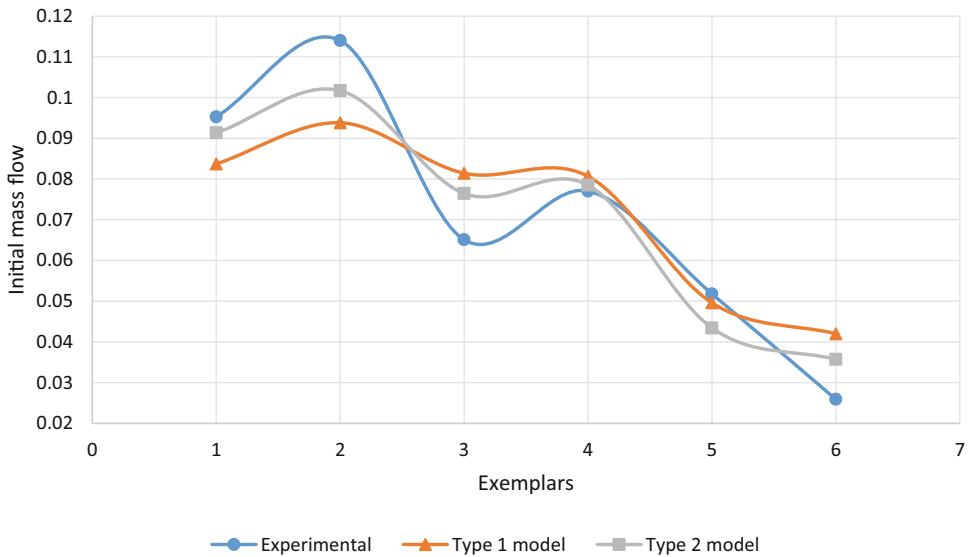
**Table 5**  
**Statistical indicators of the determined models for the final mass flow**

		<b>Fitness</b>	<b>MSE train</b>	<b>MSE test</b>	<b>Correl train</b>	<b>Correl test</b>	<b>Topology</b>
SADE-NN-1	Best	603.848	0.001656	0.009222	0.948388	0.912206	4:12:01
	Worst	64.46818	0.015512	0.008296	0.364157	0.611891	4:07:01
	Average	146.0592	0.010029	0.025241	0.624525	0.700837	
SADE-NN-2	Best	910.1513	0.001099	0.014107	0.96608	0.842717	4:17:01
	Worst	61.82792	0.016174	0.041729	0.432883	0.609068	4:11:01
	Average	178.4392	0.009583	0.026241	0.63365	0.688526	
hSADE-NN	Best	1140.362	0.000877	0.005153	0.973135	0.94476	4:04:01
	Worst	64.02392	0.015619	0.058979	0.826542	0.630103	4:15:01
	Average	340.266	0.00452	0.012509	0.871616	0.886054	

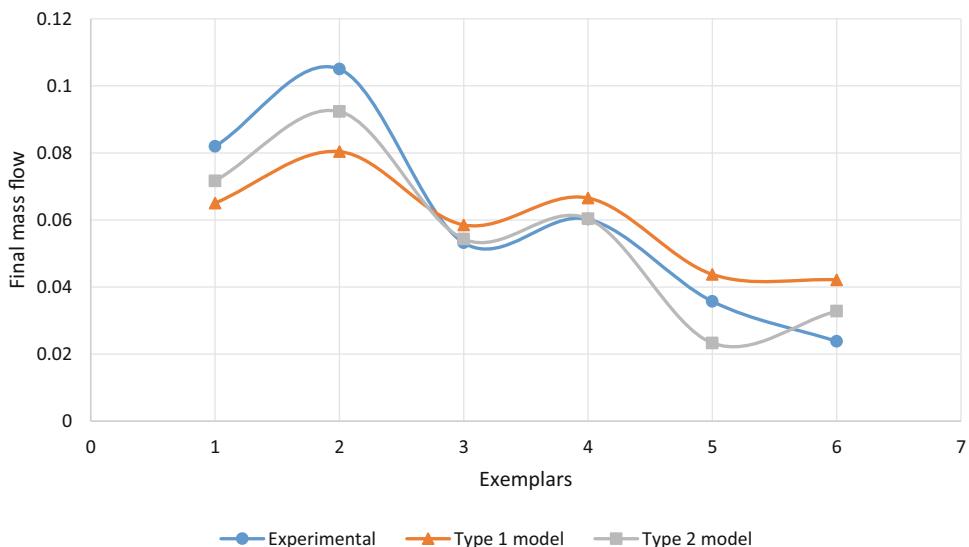
In Table 3, the determined ANNs cannot simultaneously model the two outputs with acceptable accuracy. This indicates that, due to the complexity of the relations between the inputs and outputs, each output must be modeled separately.

Therefore, using the same settings, a series of simulations for the initial mass flow (Table 4) and final mass flow (Table 5) were performed.

By comparing the statistic indexes in both training and testing phases, it can be seen that when separate models are generated, the performances for the initial mass flow and final mass flow of the vitamin C pertraction improve. This can also be seen in a point by point comparison between the experimental data and the predictions generated by hSADE-NN when the two outputs are considered simultaneously (Type 1 model) and when the outputs are modeled separately (Type 2 model) for the initial mass flow (Fig. 10) and final mass flow (Fig. 11).



**Fig. 10** Comparison between experimental data and predictions generated by the hSADE-NN approach for the initial mass flow



**Fig. 11** Comparison between experimental data and predictions generated by the hSADE-NN approach for the final mass flow

#### 4 Notes

Although the neuroevolutionary approach solves some specific issues related to the optimal topology and ANN parameter determination, its application must not be considered a panacea. The evolutionary/natural processes are relatively simple to imitate, a fact that is

reflected in the multitude of optimization mechanisms inspired from nature that have been proposed in recent years. However, they suffer from some drawbacks that also copy across into the neuroevolutionary mechanisms. Therefore, attention must be paid to the way in which the problem is formulated, how the neuroevolution is performed (at what level and what type) and how the mechanisms applied to reach the exploration-exploitation balances in the optimizer are controlled/used. Once these aspects are clarified, the application of the neuroevolutionary approach is simple and straightforward, the dataset describing the problem being solved representing the only aspect that needs changing when going from one process to another. In this work, it was shown how different bioprocess mechanisms can be efficiently modeled from the points of view of method accessibility, generality, and accuracy of the results. In terms of performance, the best solutions for both processes considered as case studies (reactive extraction of folic acid and vitamin C pertraction) were obtained with the neuroevolutionary variant (ANN + DE) that contains the highest number of strategies for diversity control (initialization improvement, new mutation variant, parameter self-adaptation, and hybrid local search) in the form of hSADE-NN.

## References

- Curteanu S, Cartwright HM (2011) Neural networks applied in chemistry. I. Determination of the optimal topology of multilayer perceptron neural networks. *J Chemometrics* 25(10):527–549
- Ragg T, Gutjahr S (1997) Automatic determination of optimal network topologies based on information theory and evolution. In: EURO-MICRO 97 proceedings of the 23rd EURO-MICRO conference: new frontiers of information technology (cat. no. 97TB100167)
- Cartwright HM, Curteanu S (2013) Neural networks applied in chemistry. II. Neuro-evolutionary techniques in process modeling and optimization. *Ind Eng Chem Res* 52 (36):12673–12688
- Ławryńczuk M (2008) Modelling and nonlinear predictive control of a yeast fermentation biochemical reactor using neural networks. *Chem Eng J* 145(2):290–307
- Nagy ZK (2007) Model based control of a yeast fermentation bioreactor using optimally designed artificial neural networks. *Chem Eng J* 127(1):95–109
- Basri M, Rahman RN, Ebrahimpour A, Salleh AB et al (2007) Comparison of estimation capabilities of response surface methodology (RSM) with artificial neural network (ANN) in lipase-catalyzed synthesis of palm-based wax ester. *BMC Biotechnol* 7:53
- da Cruz Meleiro LA, Von Zuben FJ, Maciel Filho R (2009) Constructive learning neural network applied to identification and control of a fuel-ethanol fermentation process. *Eng Apps Artific Intellig* 22(2):201–215
- Chen F, Li H, Xu Z, Hou S et al (2015) User-friendly optimization approach of fed-batch fermentation conditions for the production of iturin A using artificial neural networks and support vector machine. *Electron J Biotechnol* 18(4):273–280
- Esfahanian M, Nikzad M, Najafpour G, Ghoreyshi AA (2013) Modeling and optimization of ethanol fermentation using *Saccharomyces cerevisiae*: response surface methodology and artificial neural network. *Chem Ind Chem Eng Quart* 19(2):241–252
- Silva R, Ferreira S, Bonifacio MJ, Dias JM et al (2012) Optimization of fermentation conditions for the production of human soluble catechol-O-methyltransferase by *Escherichia coli* using artificial neural network. *J Biotechnol* 160(3–4):161–168
- Storn R, Price KV (1995) Differential evolution—a simple and efficient adaptive scheme for

- global optimization over continuous spaces. Tech. Report TR-95-012. International Computer Sciences Institute, Berkeley
12. Subudhi B, Jena D (2008) Differential evolution and levenberg marquardt trained neural network scheme for nonlinear system identification. *Neural Proc Lett* 27(3):285–296
  13. Zaharie D (2009) Influence of crossover on the behavior of differential evolution algorithms. *Appl Soft Comput* 9(3):1126–1138
  14. Price K, Storn RM, Lampinen JA (2006) Differential evolution: a practical approach to global optimization. Springer Science & Business Media, Berlin
  15. Subudhi B, Jena D (2009) An improved differential evolution trained neural network scheme for nonlinear system identification. *Int J Automat Comput* 6(2):137–144
  16. Thangaraj R, Pant M, Abraham A (2009) A simple adaptive differential evolution algorithm. In: 2009 world congress on nature and biologically inspired computing (NaBIC), IEEE
  17. Lu Y, Zhou J, Qin H, Li Y et al (2010) An adaptive hybrid differential evolution algorithm for dynamic economic dispatch with valve-point effects. *Expert Syst Appl* 37 (7):4842–4849
  18. Pan Q-K, Suganthan PN, Wang L, Gao L et al (2011) A differential evolution algorithm with self-adapting strategy and control parameters. *Compt Operat Res* 38(1):394–408
  19. Kapadi MD, Gudi RD (2004) Optimal control of fed-batch fermentation involving multiple feeds using differential evolution. *Process Biochem* 39(11):1709–1721
  20. Moonchai S, Madlho W, Jariyachavalit K, Shimizu H et al (2005) Application of a mathematical model and Differential Evolution algorithm approach to optimization of bacteriocin production by *Lactococcus lactis* C7. *Bio-process Biosyst Eng* 28(1):15–26
  21. Rocha M, Pinto JP, Rocha I, Ferreira EC (2007) Evaluating evolutionary algorithms and differential evolution for the online optimization of fermentation processes. In: European conference on evolutionary computation, machine learning and data mining in bioinformatics. Springer
  22. Yao X (1999) Evolving artificial neural networks. *Proc IEEE* 87(9):1423–1447
  23. Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. *Evol Intell* 1(1):47–62
  24. Dürr P, Mattiussi C, Floreano D (2006) Neuroevolution with analog genetic encoding. In: Runarsson T et al (eds) Parallel problem solving from nature—PPSN IX. Springer, Berlin, pp 671–680
  25. Mouret J-B, Doncieux S (2008) MENNAG: a modular, regular and hierarchical encoding for neural-networks based on attribute grammars. *Evolut Intell* 1(3):187–207
  26. Fischer MM, Reismann M, Hlaváčková-Schindler K (1999) Parameter estimation in neural spatial interaction modelling by a derivative free global optimization method. In: International conference on GeoComputation, 4, Fredericksburg, Virginia, USA
  27. Plagianakos V, Magoulas G, Nousis N, Vrahatis M (2001) Training multilayer networks with discrete activation functions. In: IJCNN'01. International joint conference on neural networks. Proceedings (cat. no. 01CH37222). IEEE
  28. Lahiri SK, Khalfé N (2010) Modeling of commercial ethylene oxide reactor: a hybrid approach by artificial neural network and differential evolution. *Int J Chem Reactor Eng* 8 (1). <https://doi.org/10.2202/1542-6580.2019>
  29. Bhuiyan MZA (2009) An algorithm for determining neural network architecture using differential evolution. In 2009 international conference on business intelligence and financial engineering. IEEE
  30. Dragoi E-N, Curteanu S, Leon F, Galaction A-I et al (2011) Modeling of oxygen mass transfer in the presence of oxygen-vectors using neural networks developed by differential evolution algorithm. *Eng Apps Artific Intellig* 24(7):1214–1226
  31. Drăgoi E-N, Curteanu S, Lisa C (2012) A neuro-evolutive technique applied for predicting the liquid crystalline property of some organic compounds. *Eng Optimiz* 44 (10):1261–1277
  32. Dragoi E-N, Curteanu S, Galaction A-I, Cascaval D (2013) Optimization methodology based on neural networks and self-adaptive differential evolution algorithm applied to an aerobic fermentation process. *App Soft Comp* 13 (1):222–238
  33. Tizhoosh HR (2005) Opposition-based learning: a new scheme for machine intelligence. In: International conference on computational intelligence for modeling, control and international conference on intelligent agents, web technologies and internet commerce, Vienna
  34. Dragoi E-N, Curteanu S, Fissore D (2012) Freeze-drying modeling and monitoring using a new neuro-evolutative technique. *Chem Eng Sci* 72:195–204

35. Dragoi E-N, Curteanu S, Cascaval D, Galaction A-I (2016) Artificial neural network modeling of mixing efficiency in a split-cylinder gas-lift bioreactor for *Yarrowia Lipolytica* suspensions. *Chem Eng Comms* 203(12):1600–1608
36. Mizzi B, Meyer M, Prat L, Augier F et al (2017) General design methodology for reactive liquid-liquid extraction: application to dicarboxylic acid recovery in fermentation broth. *Chem Eng Process* 113:20–34
37. Jessop PG (2011) Searching for green solvents. *Green Chem* 13(6):1391–1398
38. Sprakel L, Schuur B (2018) Solvent developments for liquid-liquid extraction of carboxylic acids in perspective. *Sep Purif Technol* 211:935–957
39. Demesa AG, Laari A, Tirronen E, Turunen I (2015) Comparison of solvents for the recovery of low-molecular carboxylic acids and furfural from aqueous solutions. *Chem Eng Res Design* 93:531–540
40. Fan Y, Cai D, Yang L, Chen X et al (2019) Extraction behavior of nicotinic acid and nicotinamide in ionic liquids. *Chem Eng Res Design* 146:336–343
41. Chemarin F, Moussa M, Allais F, Trelea I et al (2019) Recovery of 3-hydroxypropionic acid from organic phases after reactive extraction with amines in an alcohol-type solvent. *Sep Purif Technol* 219:260–267
42. Eda S, Borra A, Parthasarathy R, Bankupalli S et al (2018) Recovery of levulinic acid by reactive extraction using tri-n-octylamine in methyl isobutyl ketone: equilibrium and thermodynamic studies and optimization using Taguchi multivariate approach. *Sep Purif Technol* 197:314–324
43. Gorden J, Zeiner T, Sadowski G, Brandenburgsch C (2016) Recovery of cis, cis-muconic acid from organic phase after reactive extraction. *Sep Purif Technol* 169:1–8
44. Brouwer T, Blahusiak M, Babic K, Schuur B (2017) Reactive extraction and recovery of levulinic acid, formic acid and furfural from aqueous solutions containing sulphuric acid. *Sep Purif Technol* 185:186–195
45. Djas M, Henczka M (2018) Reactive extraction of carboxylic acids using organic solvents and supercritical fluids: a review. *Sep Purif Technol* 201:106–119
46. Galaction AI, Blaga AC, Cașcaval D, Folescu E (2005) Separation of vitamins by non-conventional techniques. Facilitated pertraction of vitamin C. *Rev Med Chir Soc Med Nat Iasi* 109(4):895–898
47. Galaction A-I, Blaga A-C, Cascaval D (2005) The influence of pH and solvent polarity on the mechanism and efficiency of folic acid extraction with Amberlite LA-2. *Chem Ind Chem Eng Quart* 11(2):63–68



# Chapter 6

## Computational Approaches for De Novo Drug Design: Past, Present, and Future

Xuhan Liu , Adriaan P. IJzerman , and Gerard J. P. van Westen

### Abstract

Drug discovery is time- and resource-consuming. To this end, computational approaches that are applied in de novo drug design play an important role to improve the efficiency and decrease costs to develop novel drugs. Over several decades, a variety of methods have been proposed and applied in practice. Traditionally, drug design problems are always taken as combinational optimization in discrete chemical space. Hence optimization methods were exploited to search for new drug molecules to meet multiple objectives. With the accumulation of data and the development of machine learning methods, computational drug design methods have gradually shifted to a new paradigm. There has been particular interest in the potential application of deep learning methods to drug design. In this chapter, we will give a brief description of these two different de novo methods, compare their application scopes and discuss their possible development in the future.

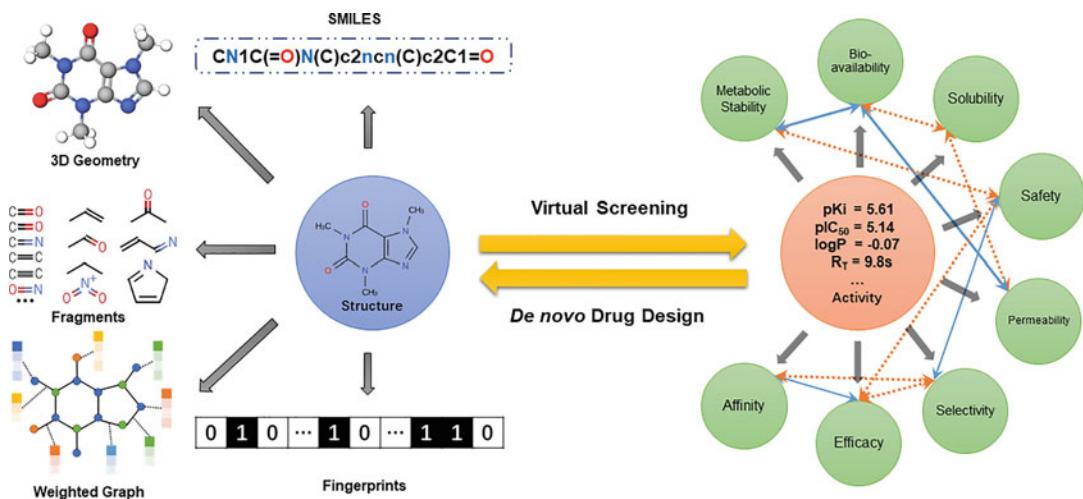
**Key words** Machine learning, Cheminformatics, Deep learning, Drug discovery

---

### 1 Introduction

Drug discovery is always considered to have a significant “serendipity” component—researchers need to identify a small fraction of feasible molecules with desired physicochemical and biological properties from the vast chemical space, which has been estimated to be comprised of  $10^{23}$  to  $10^{60}$  feasible drug-like molecules [1]. This number of potential candidate molecules is too large to screen experimentally [2]. Moreover, drug molecules have a high promiscuity [3], that is, each drug-like molecule has six protein targets on average, leading to unexpected toxicity and withdrawal of some FDA-approved drugs from the market [4]. These problems have contributed to an increase in the average cost to over one billion USD for the development of a new drug in a process that takes about 13 years to reach the market [5].

To this end, computer-aided drug discovery (CADD) aims to speed up the drug discovery process by integrating chemical and



**Fig. 1** Schematic overview of the interplay of two methods in computational drug discovery: virtual screening and de novo design. The left of the figure shows ways in which a molecule can be described for computational methods (see Subheading 2.1). On the right the multiobjective nature of the problem is shown. Properties are often contrary (orange arrows) and sometimes cooperative (blue arrows), but must be optimized simultaneously (see Subheading 2.2)

biological information about ligands and/or targets [6]. CADD is a broad field of research that includes de novo drug design and virtual screening methods (Fig. 1, center). De novo drug design suggests new molecules as starting points for chemical modifications that result in novel leads. By contrast, virtual screening methods try to uncover the hidden relationships between chemical structure and pharmacological activity. CADD has always been a combinatorial optimization problem with multiobjective optimization. Virtual screening methods provide a scoring function that mimics bioassays in order to guide the drug design algorithm to converge on the optimal molecule. Because it is impossible to enumerate every chemical entity in the chemical universe, CADD in practice does not lead to a globally optimal solution, but it narrows down the searching scope of chemical space and converges on a local or practical optimum [7].

In the past, machine learning methods, such as random forests, were mainly constructed for virtual screening, that is, given the structure of a chemical compound predict its biological activity. With the increased availability of (public) data and development of computer sciences (e.g., the introduction of GPU computation), machine learning methods have also found their way to the field of de novo drug design. Deep learning (DL) methods in particular have attracted increasing attention as a promising approach for drug discovery [8]. DL methods are an extension of artificial neural networks that add a variety of multiple hidden layers, thus making the network significantly deeper [9]. In 2012, deep convolutional

neural networks (CNNs) were proposed and became a breakthrough in image classification [10]. Subsequently, generative adversarial networks (GANs) were developed for image generation and, by 2014, these had significantly improved the quality of generated images [11]. Based on these achievements, the DL methods could also provide a series of solutions for prediction, generation, and decision-making in other data rich fields beyond image recognition and natural language processing [8, 12]. In drug discovery, DL has catalyzed an explosion of applications for de novo drug design since Gómez-Bombarelli et al. applied variational autoencoders (VAE) to generate SMILES-based chemical compounds in 2016 [13].

As traditional optimization algorithms and recent DL methods are quite distinct, it is necessary to make a clear comparison between both methods. In the following paragraphs, we will give more theoretical details of these two different methods and their application in the field of drug design. We will also discuss the advantages and disadvantages of both of them and possible directions of their combination in the future.

---

## 2 De Novo Drug Design

Due to the discreteness of chemical space, drug design is intuitively rendered into a combinatorial optimization problem. The solution of this drug design problem is searching for an optimal combination of building blocks to find the best solution according to the required conditions. Based on the difference of the building blocks, drug design algorithms can be classified into atom-based and fragment-based methods. The atom-based methods are the more intuitive approaches and easily construct a variety of novel structures, but are more time-consuming and less able to converge to the best solutions. In contrast, fragment-based methods reduce the chemical space dramatically by predefining the fragment library and are consequently faster in searching for optimal molecules than atom-based methods, although the diversity is lower compared to atom-based methods. However, the drug design problem cannot be solved completely because an increase in fragments leads to a combinatorial explosion of chemical space, making an exhaustive search impossible. Therefore, more efficient molecular representations need to be developed to suggest novel potential drug-like molecules efficiently in addition to, or as an alternative for, the known atomistic and fragment-based representations.

Usually, drug molecules are organic compounds with physicochemical properties optimal for drug-like molecules, such as Lipinski's rule of five. Moreover, sufficient on-target affinity and avoiding off-target affinity are additional objectives that need to be met.

Drug de novo design can be further classified into structure-based and ligand-based methods based on whether 3D structure information is available and included [7, 14]. In structure-based drug design, the 3D structure of a protein target is required for guiding ligand design but prior knowledge of other ligands is unnecessary. The optimal ligands are commonly obtained by calculating the binding energy when combining at the protein active site to interact with the protein. On the contrary, ligand-based methods do not exploit protein target structure information but require the prior knowledge comprised of known ligands of given structures which are used to measure their similarity with generated molecules.

## 2.1 Molecular Representations

Chemical compounds are not a random cluster of atoms and functional groups, but rather have a definite structure represented by the arrangement of chemical bonds between atoms and information on the geometric 3D shape. This information needs to be represented computationally for algorithms to be able to predict properties of these molecules (Fig. 1). Ideally, the full 3D shape geometry is used for construction of a fitness function in structure-based optimization methods, such as docking or molecular dynamics [15]. However, these 3D approaches always consume more computational resources and time; they also require the computational generation of conformers, a process which can be prone to error.

To circumvent this requirement 2D approaches are used. As the key to properties of the molecules lies in fragments with a specific connection pattern of the atoms, molecules can be represented as a bag of fragments which can be perturbated easily for generating new molecules (in the form of a binary bit string). This molecular fingerprint can also be used as input for virtual screening [16]. A downside to fingerprints is that the connectivity information linking the individual fragments is not available. Hence various different molecules can be generated with the same combination of fragments. Moreover, while each fragment of the molecule can be mapped to one bit in a fingerprint by a hash function, such as ECFP [17], the fingerprint is always irreversible. A fingerprint cannot be reconstructed into a molecule, so it is impossible to use the molecular fingerprint directly for drug design. All in all, there is no single 2D or 3D representation that seems to meet all criteria as all can be considered to have their advantages and disadvantages [18].

To circumvent the loss of connectivity information, graph based methods are used. The most natural molecular representation is an undirected graph where the atoms and bonds are nodes and edges, respectively [19]. These graphs can be reversibly converted into a text format using a preset grammar such as simplified molecular-input line-entry specification (SMILES). Analogous to natural language processing, SMILES is regarded as a chemical

language and directly used in deep learning models for molecular generation. However, as SMILES follows a fixed grammar, generated texts can easily lead to invalid molecules. To solve this problem, some groups attempted to decompose SMILES into a sequence of rules from a context free grammar and improved linear molecular representation, such as DeepSMILES [20], Randomized SMILES [21], and SELFIES [22]. An advanced representation is directly storing the graph into multidimensional tensors, including type of atoms and edges, and connectivity information. This representation can make sure the molecular graph can be generated immediately without considering grammar; however, it is still computationally expensive.

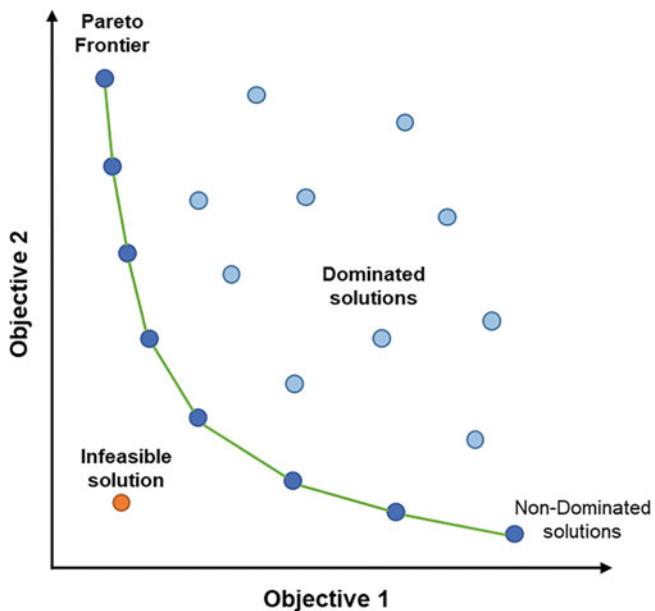
## 2.2 Multiple Objectives

As specified above, drug design is always a multiobjective problem (MOP) and designed compounds need to meet many criteria as drug candidates (efficacy, selectivity, safety, permeability, solubility, metabolic stability, synthesizability, etc.). (Fig. 1). Some of these objectives are not independent but contradictory, meaning that if an optimum is achieved on one objective it has been at the expense of making a compromise on other objectives. Unlike single-objective problems (SOP), where the best solution is on the top of ranking sorted by the scalar score of each candidate solution, the ranking of candidates in a MOP is more complicated because of conflicting objectives [14]. A straightforward method of dealing with this complication is to convert the multiple objectives into a single objective by weighted summing of scores for each objective [23].

$$f(n) = \sum_{i=1}^N w_i p_i$$

where  $f(n)$  is the fitness function and  $w_i$  is predefined by users as the weight of  $i$ th objective  $p_i$ . However, it is challenging to determine these weights because they specify a single pattern of compromise for these objectives, which can trap an optimization algorithm and lead to unreasonable solutions (Fig. 2).

In order to strike a better balance between each objective, MOP algorithms produce a set of solutions representing various compromises among the objectives. The solutions are mapped out on a hypersurface in the search space, termed Pareto Front [24]. A solution dominates another one if it is equivalent or better in all objectives and better in at least one objective compared with all other solutions. Solutions with the most appropriate compromise among the individual objectives can be identified through Pareto ranking. Several Pareto ranking algorithms have been developed (e.g., SPEA [25], NSGA [26], SMS-EMOA [27]). However, all of them are computationally expensive for large numbers of objectives and data points and lead to nonconvergence of the solutions in contradiction of the SOP [23].



**Fig. 2** Pareto frontier in multiobjective optimization. Take two objectives as an example, nondominated solutions form a boundary called Pareto frontier which separates the infeasible solutions in the lower left region from dominated solutions in the upper right region

### 3 Optimization Methods

In applications of drug design, the most popular searching algorithms are evolutionary algorithms (EAs), particle swarm optimization (PSO), and simulated annealing (SA). In the following paragraphs, we will briefly introduce their mathematical theories and their application in drug discovery (Table 1).

#### 3.1 Evolutionary Algorithms

EAs are population-based metaheuristic optimization algorithms inspired by biological evolution to mimic the genetic operators, such as “reproduction,” “mutation,” and “crossover” [44]. In the population, a pair of individuals is randomly selected for each time and play the role of parents to “reproduce” the offspring through “mutation” and “crossover” for population expansion. The scoring function, also called a fitness function in EAs, determines which individual can survive and replace the least-fit individual in the population. The surviving individuals in the updated population are selected as the new parents for next generation. For each iteration of the evolutionary cycle, the average fitness score of individuals in the population is considered. The average fitness will only increase every single generation if there is no mutation, since in this case one is always throwing away the weakest members of the population.

**Table 1**  
**Current optimization methods for de novo drug design**

Methods	Method	Molecule representation	Objectives	References
LigBuilder	GA	3D geometry	Affinity (thrombin and dihydrofolate reductase) and bioavailability score	Wang et al. [28]
LEA	GA	SMILES	Analogs fitness (retinoid and salicylic acid) and physicochemical properties	Douguet et al. [29]
ADAPT	GA	Fragment	Docking score (cathepsin D, dihydrofolate reductase, and HIV-1 reverse transcriptase), RO5	Pegg et al. [30]
PEP	GA	Fragment	Force field-based binding energy (Caspase 1, 3, and 8)	Budin et al. [31]
SYNOPSIS	GA, SA	Reactivity	Electric dipole moment, affinity to binding site (HIV-1 reverse transcriptase)	Vinkers et al. [32]
LEA3D	GA	Fragment	Molecular properties, affinity to binding site (thymidine monophosphate kinase)	Douguet et al. [33]
GANDI	GA	Fragment	2D/3D similarities and force field-based binding energy (cyclin-dependent kinase 2)	Dey et al. [34]
Molecule Commander	GA	Fragment	Adenosine receptor A1 pharmacophore, off-target selectivity (Adenosine receptor A <sub>2A</sub> , A <sub>2B</sub> and A <sub>3</sub> ), and ADMET properties	van der Horst et al. [35]
Molecule evaluator	GP	Tree SMILES	QSAR functions, docking, experiments, similarity to template molecules (neuraminidase inhibitor)	Lameijer et al. [36]
MEGA	GP	Graph	Binding affinity score (Estrogen receptor), similarity score and RO5	Nicolaou et al. [37]
FLUX	ES	Fragment	Similarity to template molecules (tyrosine kinase inhibitor, Factor Xa inhibitor)	Fechner et al. [38]
TOPAS	ES	Fragment	2D structural/topological pharmacophore similarity to template (thrombin inhibitor)	Schneider et al. [39]
MOLig	SA	Fragment	Force field-based binding energy (RecA), similarity to template molecules, oral bioavailability	Sengupta et al. [40]
CONCERTS	SA	Fragment	Force field-based binding energy (FK506 binding protein, HIV-1 aspartyl protease)	Pearlman et al. [41]
SkelGen	SA	Fragment	Binding affinity prediction score (DNA gyrase and estrogen receptor)	Dean et al. [42]
COLIBREE	PSO	Fragment	Similarity to template molecules (PPAR ligands)	Hartenfeller et al. [43]

However, if “mutation” is part of the algorithm, as it is in Genetic algorithms (*see* below), the average fitness may decrease from one generation to the next and will almost inevitably do so quite frequently as the process homes in on an optimal solution. The process continues until a termination criterion is reached, that is, the objective function score of optimal solution is no longer being improved or number of feasible solutions is sufficient. Currently, EAs are the most sophisticated algorithm used for drug de novo design in practice.

There are several major algorithmic techniques in use in EAs, examples include genetic algorithms, genetic programming, and evolutionary strategies [45]. Genetic algorithms (GAs) are one of the most fundamental and widely used EAs. GAs need to encode the phenotype (molecular structure) by means of a “chromosome” as the simulation of natural selection [46]. For example, Wang et al. developed a software named *LigBuilder*, in which each molecule was decomposed into a series of fragments from the building-block library to be used as chromosome [28]. The mutation operator was defined to allow only carbon, nitrogen, and oxygen atoms of the molecules with the same hybridization state to mutate to each other. During the process, fragments were combined to generate a new population through randomly selecting a growing site on the seed structure and addition of a fragment from the building-block library. Each molecule was represented with its SMILES sequence as the “chromosome.” Similarly, Douguet et al. defined allowable crossover points and mutation rules were generated for breeding valid SMILES as the next generation in their method deemed *LEA* [29].

In GAs, there are fixed data structures (despite the linearity of the chromosome) to organize the variables which need to be optimized. But if these variables are interdependent through an explicit relationship, such as procedural or functional representation, genetic programming (GP) is a more suitable method to realize the EA principles [47]. In GP, the chromosomes are always represented as trees rather than the fixed-length strings of GAs. Crossover is implemented as a recombination of subtrees between two parents, while mutation selects and alters a random node or edge of the tree depending on its type. Usage of a SMILES representation as a “chromosome” is troublesome for genetic operators, because SMILES per se is a grammatically constrained linear string and the random mutation and crossover will produce a large number of invalid SMILES. Lameijer et al. solved this problem in their software, named *Molecule Evaluator* based on a SMILES representation employing a GP [36]. In Molecule Evaluator, TreeSMILES are defined as the tree structure being transformed from the SMILES according to its grammar, in which each node and edge denoted the atom and bond, respectively. Every node or edge has an operator function, making mathematical expressions easy to evolve and evaluate.

Evolutionary strategies (ES) are a third EA technique using the concepts of adaptation and evolution. In contrast to GAs, selection in ES is based on a fitness ranking rather than fitness values, although mutation and selection also play an important role for breeding [48]. ES operates on the parent and the result of its mutants. In ES, a number of mutants are generated which compete with the parent, wherein the best mutant becomes the parent of the next generation. For example, Flux implemented a simplistic  $(1, \lambda)$ -ES without adaptive step-size control and defined the crossover and mutation generators on the fragment-based “reaction tree” of each pair of parents [38]. Selection was performed only among the offspring and the parent died out, which could facilitate escaping local optima in the fitness landscape. Another method, TOPAS, used a simple  $(1, \lambda)$ -ES with adaptive parameters [39]. During the stochastic search process, there were  $\lambda = 100$  variants generated through virtual synthesis for each iteration. The distribution of Tanimoto similarity with their parents was controlled by a step-size parameter, which guaranteed that the chemical space of the population adapts to the local shape of the fitness landscape. Similarly, only one variant with the best fitness score became the parent of the next generation while the current parent was discarded.

### **3.2 Particle Swarm Optimization (PSO)**

PSO solves the optimization problem based on the observation of collective intelligence in many natural systems that individuals cooperate with each other to improve not only their collective performance but also each individual’s performance on a given task [49]. Similar to EAs, PSO also is a population-based method. In PSO a population, known as a swarm, contains a series of candidate solutions (called particles). The swarm needs to be initialized to represent the position in the search space, and the individuals should have initial velocities. In addition, each particle has its own memory to record the best fitness of its past for communication with others. In each iteration, the fitness score of each individual’s position is calculated to register the position with highest fitness. Subsequently the velocity of each particle is randomly influenced by three factors: one is the position of the particle with the highest fitness (i.e., best match with the fitness function) in its neighborhood. Secondly, there is the position of highest fitness ever encountered previously. Finally, there is a random component in the updated velocities, which is an adjustment to the velocity in a direction entirely uncorrelated with all other particle properties. If there were no such random component, and the optimal solution lay entirely outside the initial bounds of the swarm it is unlikely the swarm would be able to find it. The new position of each particle is calculated based on its updated velocity. One of the key points is how to define the topology of the swarm to determine its neighbors to avoid being trapped in a local minimum.

The PSO algorithm was frequently used in continuous search spaces. In order to be applied in the discrete search space of drug-like molecules, Hartenfeller et al. replaced the concept of velocity of each particle with the quality vector and developed *COLIBREE* for drug design [43]. In *COLIBREE*, each molecule is represented as a combination of building blocks and linkers. The fitness function is defined as the similarity between reference ligands and generated molecules under chemically advanced template search (CATS) descriptors. Each particle stores the current search point (a molecule) and a quality vector which represents a relative probability for every fragment in the library to be chosen in the next search step for constructing the molecule. During the optimization cycle, each particle created a new molecule and updated its memory after the fitness was evaluated. The quality vector was incremented if the fragment had been part of the molecule stored in the memory of the current particle. In the end, good solutions have a higher probability to be chosen for molecule construction in subsequent search steps.

### 3.3 Simulated Annealing

For the purpose of estimating a global optimum of an objective function, Simulated Annealing (SA) is based on the cooling and crystallizing behavior of chemical substances. This behavior is affected by both the temperature and the thermodynamic free energy. In general, SA sets the initial temperature and chooses a random point as the initial solution. It then works iteratively in steps during which the temperature is progressively decreased from an initial value to zero. For each iteration, a new point is randomly selected from the points close to the current one as the solution. Subsequently, a probability score is calculated based on whether the quality of the new solution is better than the current solution or not, and the algorithm decides which solution will be adopted to replace the current solution. This probability is affected by the temperature, that is, the temperature controls the balance of exploration/exploitation strategies. If the initial temperature is too low or cooling is too fast, the algorithm will not effectively explore the search space. Conversely, when the temperature is set too high, the algorithm will take too long to converge. The key point of SA is the strategy about how to choose a new solution, which has a significant impact on its performance.

Sengupta et al. developed *MOLig* with the SA algorithm in 2012 [40]. This method encoded each molecule into a tree-like representation which was stored as an array of positive integers. In this array numbers symbolized a molecular fragment and specified the connectivity pattern. For each iteration, there were several perturbation operators being defined for generating molecules as a new solution and it would be determined by temperature related probability whether this new solution would replace the current one. The iteration would terminate once the temperature was

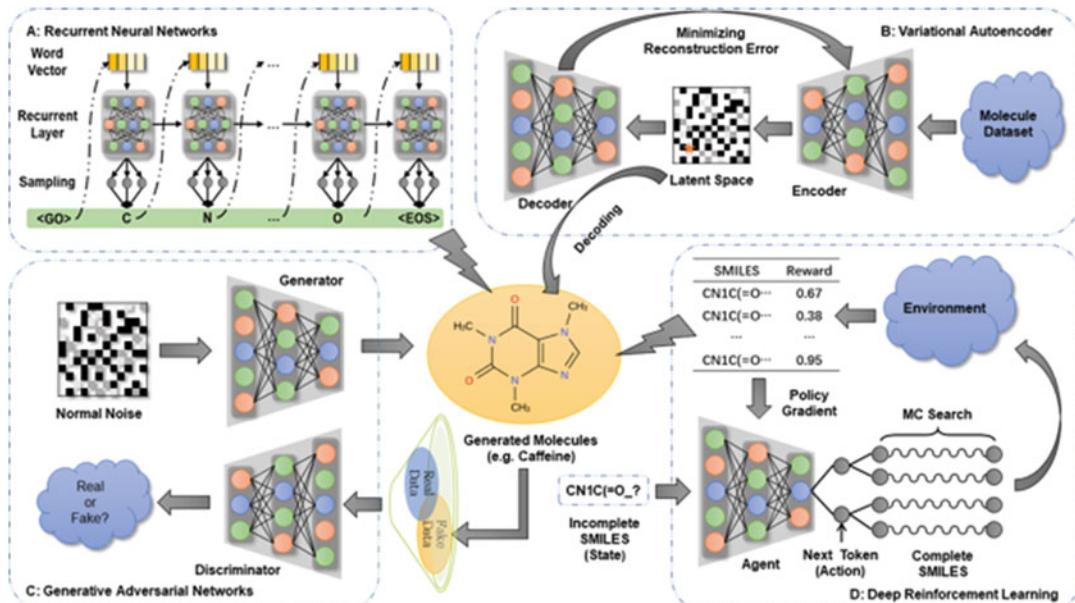
reduced to zero. In addition, *CONCERTS* [41] and *SkelGen* [42] are other structure-based de novo design methods based on the SA algorithm.

## 4 Deep Learning Algorithms

The common basic DL architectures used in de novo drug design are recurrent neural networks (RNNs), variational autoencoder (VAE), deep reinforcement learning (RL), and generative adversarial networks (GANs) (Fig. 3). Most studies of DL applications combine two or more models to address specific issues. In the following paragraphs, we give the details about these architectures and how these models can be applied in drug design. We also list and categorize these methods based on these DL architectures in Table 2.

### 4.1 Recurrent Neural Networks (RNNs)

RNNs are formed as directed graph that includes a temporal component [81]. Contrary to feedforward networks, the network can contain cyclic connections that allow them to remember things from prior inputs when generating output. RNNs can process sequential data effectively and have shown excellent performance in the field of natural language processing (NLP) such as handwriting [82] or speech recognition [83]. RNNs deal with words in text



**Fig. 3** Four basic deep learning architectures commonly used in de novo drug design, including recurrent neural networks (a), variational autoencoder (b), generative adversarial networks (c) and deep reinforcement learning (d)

**Table 2**  
**The current DL-based de novo drug design methods**

Methods	Molecular representations	Architectures	Database	Objectives	References
LatentGAN	SMILES	VAE, GAN	ChEMBL, ExCAPE-DB	Affinity to EGFR, HTR1A and S1P1R	Oleksi, et al. [50]
ANTC	SMILES	DNC, GAN, RL	ChemDiv	Similarity, diversity, QED, and presence of sp3-rich fragments	Putin et al. [51]
SMILES	AAE	ChEMBL, ExCAPE-DB	DRD2	Affinity to DRD2	Blaschke et al. [52]
ChemVAE	SMILES	VAE	QM9, ZINC	SAS and QED	Gómez-Bombarelli et al. [13]
ChemTS	SMILES	RNN, MCTS	ZINC	$\log P$ SAS and ring penalty	Yang et al. [53]
SSVAE	SMILES	VAE	ZINC	Drug-likeness	Kang et al. [54]
		VAE, BO	ZINC	$\log P$ , SAS, QED, and ring penalty	Griffiths et al. [55]
SMILES		RNN, TL	ChEMBL	Affinity to PPAR and RXR	Merk et al. [56]
SMILES		VAE, GTM	ChEMBL	Affinity to A <sub>2a</sub> R	Sattarov et al. [57]
ReLEASE	SMILES	RL	ZINC, ChEMBL	Affinity to JAK2	Popova et al. [58]
SMILES	AAE	ZINC		Affinity to JAK2 and JAK3	Polykovskiy et al. [59]
SMILES	RNN, TL	ChEMBL		Targeting the 5-HT <sub>2A</sub> receptor; malaria and Golden Staph	Segler et al. [60]
SMILES	RNN	ChEMBL		Affinity to PPAR $\gamma$ , TRPM8, and trypsin	Gupta et al. [61]
SMILES, Inchi	RNN, PSO	ChEMBL, SureChEMBL		$\log P$ , SAS, QED, and affinity to EGFR and BACE1	Winter et al. [62]
SMILES	RNN	ChEMBL, GDB-8	Diversity		Bjerrum et al. [63]
SMILES	VAE	ZINC		Drug-likeness	Lim et al. [64]

DrugEx	SMILES	RL, RNN	ZINC, ChEMBL	Diversity and affinity to A <sub>2A</sub> AR	Liu et al. [65]
REINVENT	SMILES	RL, RNN	ChEMBL	Affinity to DRD2	Olivecrona et al. [66]
MoldQN	Atoms/Bonds	RL	ChEMBL, ZINC	logP, SAS, and QED	Zhou et al. [67]
ORGAN	SMILES	RNN, RL, GAN	GDB-17, ChEMBL	logP, SAS, and QED	Guimaraes et al. [68]
RANC	SMILES	DNC, RL, GAN	ZINC, ChemDiv	Drug-likeness	Putin et al. [69]
SD-VAE	SMILES	VAE	ZINC	Validation of molecule	Dai et al. [70]
GrammarVAE	SMILES	VAE, BO	ZINC	Validation of molecule	Kusner et al. [71]
LigDream	SMILES, 3D Geometry	VAE, CNN, RNN	ZINC, DUDE	Affinity to A <sub>2A</sub> AR, THRΒ, and KIT	Skalic et al. [72]
	3D geometry	GCN	scPDB, BMOAD	Affinity to given protein	Aumentado-Armstrong [73]
GraphVAE	Graph	VAE	QM9	Validation of molecule	Simonovsky et al. [74]
CGVAE	Graph	VAE	QM9, ZINC, CEPDB	QED	Liu et al. [75]
GCPN	Graph	GCN, RL	ZINC	logP, SAS, and QED	Yu et al. [76]
JT-VAE	Graph	VAE	ZINC	logP, SAS, and Ring Penalty	Jin et al. [77]
MolecularRNN	Graph	RNN, RL	ZINC	logP, SAS, and QED	Popova et al. [78]
MolGAN	Graph	GAN, RL, GCN	QM9, GDB-17		De Cao et al.
MOLECULE CHEF	Graph	VAE, GGN, USPTO RNN		Synthesizability	Bradshaw et al. [79]
DeepEMPO	Fragment	RL	ChEMBL	Affinity to DRD2 and DRD4	Ståhl et al. [80]

5-HT2A, 5-hydroxytryptamine receptor 2A; A2AR, Adenosine receptor A2A; BACE, Beta-secretase 1; DRD2 and DRD4, Dopamine receptor D2 and D4; EGFR, Receptor protein-tyrosine kinase; HTR1A, 5-hydroxytryptamine receptor 1A; JAK2 and JAK3, Tyrosine-protein kinase 2 and 3; PPAR, Peroxisome proliferator-activated receptor; QED, quantitative estimation of drug-likeness; RXR, Retinoic acid receptor; SAS, synthesizability score; SIPRL, Sphingosine-1-phosphate receptor 1; TRPM8, Transient receptor potential cation channel subfamily M member 8

step by step and deliver the current hidden information to the next step in the network with the same structure simultaneously. By analogy, the direct application of RNNs in drug design takes the linear molecular representations as input [53, 60, 61]. For example, SMILES are always preprocessed by being split into a sequence of tokens  $\mathbf{x}_{1:n} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ . The SMILES string is then prefixed with a start token  $\mathbf{x}_0$  as input feature and suffixed with the end token  $\mathbf{x}_{n+1}$  as the output labels. The RNN model  $\pi_\theta$  parametrized by  $\theta$  determines the probability distribution  $y_i$  of tokens based on  $\mathbf{x}_0: i-1$ :

$$\mathbf{h}_i = f_r(\mathbf{h}_{i-1}, \mathbf{x}_{i-1})$$

$$y_i = f_o(\mathbf{h}_i)$$

here,  $f_r$  denotes recurrent layers and receives the last hidden states  $\mathbf{h}_{i-1}$  and input features  $\mathbf{x}_{i-1}$  to calculate the current hidden states  $\mathbf{h}_i$ . In order to avert the problem of long-distance dependencies caused by gradients vanishing or exploding, many variational versions have been proposed, including two common implementations: long short-term memory (LSTM) [84] and gated recurrent unit (GRU) [85], which contain a memory cell and some different gates to determine forgotten and reserved information. In the end,  $\mathbf{h}_i$  are delivered to output layers  $f_o$  for calculation of output values  $y_i$  and commonly, the probability of each word in the vocabulary is computed by the SoftMax function. For the model training, the maximum likelihood estimation (MLE) is always chosen to calculate the loss function:

$$\mathcal{L}_{\text{MLE}} = \sum_{j=1}^m \sum_{i=1}^{n+1} \log \pi_\theta(\mathbf{x}_i | \mathbf{x}_{0:i-1})$$

here,  $m$  is the total number of samples with sequence length  $n$  in the training set. The MLE loss function can be optimized with the backpropagation algorithm commonly used for DL model training.

The RNN model always serves as one of the basic components in the more complicated DL architectures, which will be introduced in the following paragraphs. If used independently, RNN models are often beneficial for molecular library generation. For example, Segler et al. pretrained an RNN model on the ChEMBL database containing 1.4 million molecules and employed “transfer learning,” also called “fine-tuning” methods to make molecules focused on the chemical space for the 5-HT<sub>2A</sub> receptor [60]. To improve the efficiency of desired molecular generation, Yang et al. proposed a method they termed *ChemTS* by combining an RNN model with Monte Carlo tree search [53]. Subsequently this method was successfully applied and several molecules were synthesized and confirmed to be desirable chemical compounds [86]. To balance validity and diversity of molecular generation, Gupta et al. modified the SoftMax function as follows:

$$P_k = \frac{\exp\left(\frac{y}{T}\right)}{\sum_k \exp\left(\frac{y}{T}\right)}$$

by adding a temperature factor  $T$  to rescale the probability of each token  $k$  in the vocabulary [61]. If temperature is increased, the diversity of molecular generation will improve, but the validation rate will decrease. Arús-Pous et al. studied the performance of an RNN model for molecular generation on the GPB-13 dataset and found that it always fails to generate complex molecules with many rings and heteroatoms due to the syntax of SMILES [87].

## 4.2 Variational Autoencoders

Variational autoencoders (VAEs) are a frequently used DL method aiming to learn representations for dimensionality reduction in an unsupervised manner [88]. The architecture of autoencoders consists of an DL-based encoder and decoder. The encoder maps the high-dimensional input data into a latent space with lower dimensional representation, whereas the decoder reconstructs these representations in the latent space into the original inputs. VAEs are a probabilistic generative model based on a directed graph with an autoencoder-like structure, while its mathematical basis, which is derived from the theory of variational inference, has little to do with traditional autoencoders [89].

The datapoint  $\mathbf{z}$  in the latent space can be transformed into input data  $\mathbf{x}$  by the decoder which estimates the likelihood  $p_{\theta}(\mathbf{x}|\mathbf{z})$  with parameters  $\theta$ . In order to train the model, a straightforward approach is maximizing the distribution of input data  $p(\mathbf{x})$  which is approximated by  $p(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$  [90]. Due to the intractability of this integral, the encoder is introduced to learn a posterior  $q_{\varphi}(\mathbf{z}|\mathbf{x})$  parameterized by  $\varphi$ ; the formula for computing  $p(\mathbf{x})$  can be rewritten as follows:

$$\begin{aligned} \mathbb{E}_{q_{\varphi}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})] &= D_{\text{KL}}\left(q_{\varphi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}|\mathbf{x})\right) \\ &\quad + \mathbb{E}_{q_{\varphi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta}(\mathbf{x}, \mathbf{z}) - \log q_{\varphi}(\mathbf{z}|\mathbf{x})\right] \end{aligned}$$

The first term in the right hand side is Kullback–Leibler (KL) divergence and the second term is called the evidence lower bound (ELBO). Because of the nonnegativity of the KL divergence, the ELBO is a lower bound of the  $\log p(\mathbf{x})$  and is also rewritten as:

$$\mathcal{L}(\varphi, \theta) = \mathbb{E}_{q_{\varphi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta}(\mathbf{x}|\mathbf{z})\right] - D_{\text{KL}}\left(q_{\varphi}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right)$$

In order to obtain maximization of  $p(\mathbf{x})$ , ELBO can be regarded as an objective function and maximized for training both the encoder and decoder simultaneously. Commonly in VAEs,  $p(\mathbf{z})$  is assumed as a unit normal Gaussian distribution and  $q_{\varphi}(\mathbf{z}|\mathbf{x})$  is chosen as a factorized Gaussian distribution:

$$p(\mathbf{z}) \tilde{\mathcal{N}}(0, \mathbf{I})$$

$$q_{\phi}(\mathbf{z}|\mathbf{x}) \tilde{\mathcal{N}}(\mu, \text{diag}(\sigma^2))$$

and the output of the encoder is shifted to output the value of the mean and the variance for the Gaussian distribution. During the training process through backpropagation, the reconstruction error of the decoder is reduced by maximizing the first term of ELBO and the encoder estimates a more accurate posterior by minimizing the KL divergence with the true priori of latent variables.

In 2016, Gómez-Bombarelli et al. proposed ChemVAE which made the molecules and its descriptors reversible, that is, descriptors can not only be extracted in the continuous latent space by the encoder for prediction, but also be restored to the molecules by decoder for generation [13]. In addition, VAEs can also be extended for conditional generation to design molecules with desired properties [54, 64]. However, with a CNN encoder and an RNN decoder, the validation rate of SMILES generated by *ChemVAE* oscillated around 75%, which was far below the performance of pure RNN models (94–98%). To address this issue, Kusner et al. represented the grammar-based SMILES in a parsing tree form with context-free grammar. They introduced the grammar VAE (GVAE) model which directly encodes to and from the parsing tree to ensure the validation of generated SMILES [71]. Similarly, Dai et al. also proposed a syntax-directed variational autoencoder (SD-VAE) inspired by syntax-directed translation for syntax and semantics check [70]. In addition, Bjerrum et al. combined multiple different encoders to improve the diversity of generated molecules [63].

### 4.3 Deep Reinforcement Learning

Reinforcement learning (RL) is modeled as a Markov decision process for the interplay between an agent and an environment [91]. The goal of RL is optimizing the agent to maximize the accumulated rewards obtained from the environment by choosing effective actions. After the agent takes an action at the current step, the environment will adapt to this step by forming a new state. For the agent, a DL model can be employed to mimic the value, which predicts expected rewards of each action or each state/action pair, or policy function, which directly provides the probability of each action. For the SMILES-based drug design problem, an RNN is commonly used to model the policy function after being pretrained with an MLE loss function. At each step  $i$ , the action  $a_i$  is introduction of a token from the vocabulary chosen by the policy function based on the current state  $s_i$ , which contains all the tokens generated so far  $s_i = [a_1, \dots, a_{i-1}]$ . The accumulated rewards  $G_T$  are the simple sum of rewards over the total steps  $T$ . The aim of RL is to maximize the expected accumulated rewards:

$$J(\theta) = \mathbb{E}[G_T | s_0, \theta] = \sum_{i=1}^T \pi_\theta(a_t | s_i) \cdot R_i$$

Usually, the end reward  $R_T$  can be obtained immediately by the environment after the generation of SMILES has completed, the intermediate reward for the action at each step is estimated by Monte Carlo (MC) search with roll-out policy:

$$R_i = R(s_i) = \begin{cases} \frac{1}{N} \sum_{n=1}^N R(\tilde{s}_T^n), & \tilde{s}_T^n \in MC(s_i), \text{ for } t < T \\ R(s_T), & \text{for } t = T \end{cases}$$

Because of the certainty of states after the action taken by the agent, the MC search is always removed and  $R_i$  is simplified as the end reward  $R_T$ . The expected accumulated rewards have a simple form:

$$J(\theta) = \mathbb{E} [G_T | s_0, \theta] = R_T \sum_{i=1}^T \pi_\theta(a_t | s_i)$$

With the REINFORCE algorithm [92], parameters  $\theta$  in the RNN policy function can be derived as follows:

$$\nabla_\theta J(\theta) = \sum_{i=1}^T \mathbb{E}_{a_t \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a_t | s_i) \cdot R_i]$$

Popova et al. developed a method *ReLeaSE* in which a stack-augmented RNN model was used as the policy function trained with the REINFORCE algorithm. It was shown to work effectively for the generation of inhibitors toward Janus protein kinase 2 (JAK2) [58].

In addition to the policy gradient to train the policy function, Zhou et al. proposed another method *MolDQN* based on deep Q-learning to fit the Q-value function rather than the policy function [67]. Mathematically, for a policy  $\pi$ , the value of an action  $a$  on a state  $s$  can be defined as follows:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{i=t}^T R_i \right]$$

This action-value function calculates the future rewards of taking action  $a$  on state  $s$ , and subsequent actions decided by policy  $\pi$ . The optimal policy is defined as follows:

$$\pi^* = \arg \max_a Q_{\pi^*}(s, a)$$

and an RNN model parameterized by  $\theta$  is introduced to approximate the value function

$$V(s; \theta) = \max_a Q(s, a; \theta)$$

This approximator can be trained by minimizing the loss function of the following:

$$\mathcal{L}(\theta) = [R(s_i) + \gamma V(s_{i+1}, \theta) - Q(s_i, a_i; \theta)]^2$$

where  $\gamma$  is the discount factor. By comparing with other policy-based RL methods, Zhou et al. argued that deep Q-learning did not need any pretrained model and performed better than the policy gradient methods.

In order to improve the stability of RL training, Olivecrona et al. proposed a method named *REINVENT*[66], in which a new loss function was introduced based on the Bayesian formula for RL:

$$\mathcal{L}(\theta) = [\log P_{\text{Prior}}(s_T) + \sigma R(s_T) - \log P_{\text{Agent}}(s_T)]^2$$

The authors used all molecules in the ChEMBL database to pretrain an RNN model as the *Priori*. With the parameter  $\sigma$ , they integrated the reward  $R$  of each SMILES into the loss function. The final *Agent* model was regarded as the *Posteriori* and trained with the policy gradient. Finally, they successfully identified a plethora of active ligands against the dopamine D2 receptor (DRD2).

Subsequently, in order to improve the diversity of generated molecules, Liu et al. proposed a method *DrugEx* in which the action was not only determined by the agent policy  $G_\theta$ , but also by a fixed exploration policy  $G_\varphi$  which had an identical network architecture. During the training process an “exploring rate” ( $\epsilon$ , from 0.0 to 1.0) was defined to control which policy would take actions. At each step a random number in [0.0, 1.0] was generated. If the value was smaller than  $\epsilon$ , the  $G_\varphi$  would determine which token would be chosen, and vice versa. This method was successfully applied to the design of ligands toward the adenosine A<sub>2A</sub> receptor [65]. DrugEx was shown to better explore the chemical space for the A<sub>2A</sub> receptors and produce ligands with similar physicochemical properties to known ligands which included complex ring systems that the other methods it was compared to could not produce.

#### 4.4 Generative Adversarial Networks

GAN models were proposed as a great breakthrough method and have been extensively applied in image generation. A GAN contains two neural networks: the generator ( $G$ ) and the discriminator ( $D$ ), which contest with each other under game theory [11].  $G$  commits to generating fake data to the point of confusing  $D$  to mistake them for real samples in the training set. The discriminator on the other hand is responsible for distinguishing between the generated fake data and the real samples. During the training loop, a batch of fake data is generated by  $G$ , which is used subsequently for training both  $G$  and  $D$  accompanied with real data. The objective functions were originally defined as two parts for  $G$  and  $D$ , respectively:

$$\begin{aligned} \min_G V(G) &= \mathbb{E}_{\mathbf{x} \sim p_d(\mathbf{x})} [\log (1 - D(G(\mathbf{z})))] \\ \max_D V(D) &= \mathbb{E}_{\mathbf{x} \sim p_d(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log (1 - D(G(\mathbf{z})))] \end{aligned}$$

here,  $p_{\mathcal{Z}}(\mathbf{z})$  is the noise distribution,  $p_{\mathcal{D}}(\mathbf{x})$  is the data distribution in the training set and  $p_{\mathcal{G}}(\mathbf{x})$  is the data distribution in the generated set. These two objective functions can be joined together as a minmax game in which  $G$  wants to minimize  $V$  while  $D$  wants to maximize it. In order to provide a strong gradient signal to obtain the global optimality, the objective function for  $D$  is rewritten as follows:

$$\max_D V(D, G) = -\log(4) + 2 \cdot D_{\text{JS}}(p_{\mathcal{D}} \| p_{\mathcal{G}})$$

where  $D_{\text{JS}}(p_{\mathcal{D}} \| p_{\mathcal{G}})$  is the Jensen–Shannon divergence defined as follows:

$$D_{\text{JS}}(p_{\mathcal{D}} \| p_{\mathcal{G}}) = \frac{1}{2} D_{\text{KL}}\left(p_{\mathcal{D}} \parallel \frac{p_{\mathcal{D}} + p_{\mathcal{G}}}{2}\right) + \frac{1}{2} D_{\text{KL}}\left(p_{\mathcal{G}} \parallel \frac{p_{\mathcal{D}} + p_{\mathcal{G}}}{2}\right)$$

here, the  $D_{\text{KL}}$  is the KL divergence.

To overcome several difficulties of GANs, such as mode collapse or lack of informative convergence metrics, the Wasserstein GAN (WGAN) was proposed to ensure faster and more stable training [93]. This model replaces the Jenson–Shannon divergence with the Earth-Mover distance:

$$W(p, q) = \inf_{\gamma \in \Pi(p, q)} \mathbb{E}_{(x, y) \sim \gamma} \|x - y\|$$

where  $\Pi(p, q)$  denotes the set of all joint distributions  $\gamma(x, y)$  whose marginals are  $p$  and  $q$ , respectively. This distance results in a more reliable gradient signal which does not vanish during the training process. Besides the abovementioned GAN models, there are varying forms being proposed which have been collected in the GAN ZOO [94].

For drug design, a GAN model is commonly used. To ensure that the generated molecules have similar physiochemical properties to molecules in the training set, the GAN is combined with other neural networks to construct a hybrid DL model, such as the RL model and the VAE model. The first application of GANs for drug design was proposed in 2017, named *ORGAN*, in which a GAN model was trained under the RL framework for multiobjective optimization [68]. *ORGAN* contained one RNN generator for SMILES generation and a CNN discriminator to optimize the chemical space of generated molecules. They used linear combination methods to integrate the reward function given by discriminator ( $R_d$ ) and objective function ( $R_c$ ) into the final rewards ( $R$ ):

$$R(\mathbf{x}) = \lambda R_d(\mathbf{x}) + (1 - \lambda) R_c(\mathbf{x})$$

Here  $\lambda \in [0, 1]$  is a weight hyperparameter for balancing these two rewards. *ORGAN* has been demonstrated to dramatically improve the percentage of generated desired drug-like molecules compared to molecules in the training set based on properties,

including solubility and synthesizability. In addition, there are some other groups that also exploit the GAN model to develop their methods for molecular design, such as *MolGAN*[95], *RANC*[51], and *ATNC*[69].

Another GAN-based hybrid model is a combination with an adversarial autoencoder (AAE) by combining multiple VAEs [96]. Instead of minimizing KL divergence to decrease the gap between the latent distribution of output by the generator and the prespecified priori (e.g., a normal distribution), AAE uses adversarial training by introducing a DL-based model as discriminator  $D$  to tell the difference between the descriptors mapped by generated molecules and molecules in the training set, respectively. The objective function of the discriminator is written as follows:

$$\max_D V(D) = \mathbb{E}_{\mathbf{x} \tilde{p}_d(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \tilde{p}_g(\mathbf{x})} [\log (1 - D(\mathbf{x}))]$$

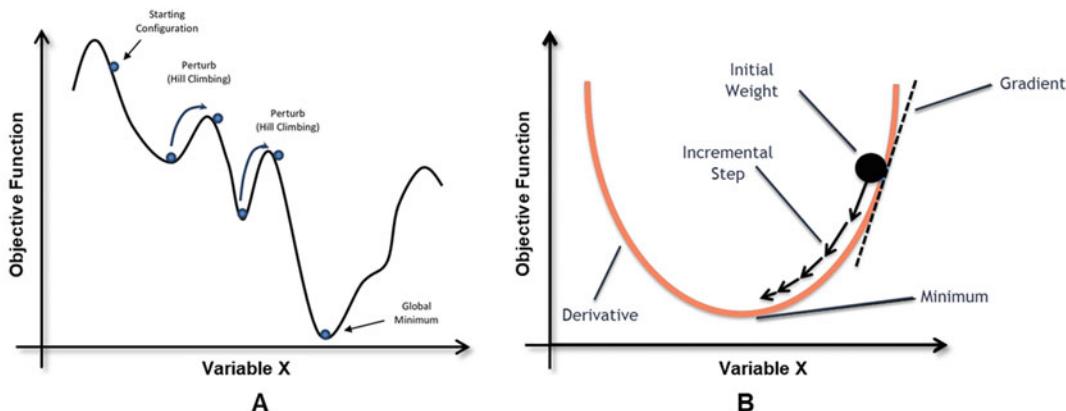
and the loss function for the VAE based generator is revised as follows:

$$\mathcal{L}(\varphi, \theta) = V(D) - \mathbb{E}_{q_\varphi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]$$

Blaschke et al. applied the AAE model for designing active ligands toward the dopamine D2 receptor [52]. In addition, Polykovskiy et al. also successfully applied this model for generating several novel inhibitors of Janus kinase 3 (JAK3) [59].

## 5 Competition or Cooperation?

Optimization methods and DL methods are different categories for drug design. Optimization methods search for the global minimum (or maximum) of the objective functions, which are always a non-convex function and have many local optima (Fig. 4a). In contrast, DL models obtain the optimal parameters with a backpropagation algorithm by minimizing the loss function; these are usually constructed as convex functions to ensure a unique minimum to be sought by gradient descent algorithms (Fig. 4b). Traditionally, there have been many successful cases in which drug candidates were found through optimization methods. But these methods do not share a unified framework and users need to define some procedures manually case by case based on their experience. In recent years deep learning methods have come to the attention of researchers who have shown interest in applying them in drug design. Based on similar basic DL architectures, more and more promising methods have been proposed to learn knowledge from the training set efficiently and generate novel molecules automatically. By comparison, optimization methods are usually population-based, meaning each individual can be manipulated directly and conveniently to construct a Pareto frontier for multiple objectives.



**Fig. 4** Objective functions for optimization methods (A) and deep learning methods (B). Usually, objective functions in optimization methods contain many local minima/maxima, while nonconvex objective functions (also called loss functions) are deliberately constructed in deep learning methods to ensure that a local minimum, if present, can be found by gradient descent algorithms

Deep learning methods, however, are typically model-based, which can be used anywhere and the learned information can be passed on to other models through transfer learning. However, current DL methods are still comparatively poor at handling the multiple objectives relevant for drug discovery; weighted summation is a common approach to tackle competitive objectives.

The paradigm shift from the optimization methods to machine learning methods is mainly caused by the availability of large public databases and breakthroughs made in the field of deep learning in image and text generation. When optimization methods dominated the field of de novo drug design, there was little public data available as prior knowledge. Optimization methods focused on the objective functions, which were summarized based on a limited number of ligands, and the data was only used to provide the initial states or form the rules as constraints for molecule generation. In the age of big data public online databases (Table 3) such as ChEMBL [97, 98], PubChem [99], ZINC [100], and DrugBank [101, 102], provide massive amounts of training data. Machine learning methods are now commonly used to extract useful information from this “big data” of drugs. Despite the current popularity of DL methods, it is worth noting that some researchers have questioned the performance of DL and benchmarked the performance different between DL and other optimization methods. For example, Yoshikawa et al. employed a grammatical evolution to develop a SMILES-based drug design algorithm, called ChemGE, which generated molecules with high binding affinity. They compared their method with three other DL methods, including CVAE, GVAE, and ChemTS. They found that with eight hours compute time, their method performed better than, or was comparable to DL methods. Similarly, Jensen proposed a graph-based

**Table 3**  
**Publicly and freely available data sources related to drug molecules**

Name	Descriptions	URL
ChEMBL	Curated database of bioactive molecules with drug-like properties.	<a href="https://www.ebi.ac.uk/chembl/">https://www.ebi.ac.uk/chembl/</a>
PubChem	Collection of freely accessible chemical information, including chemical and physical properties, biological activities, safety and toxicity information, patents, etc.	<a href="https://pubchem.ncbi.nlm.nih.gov/">https://pubchem.ncbi.nlm.nih.gov/</a>
DrugBank	Bioinformatics and cheminformatics resource that combines detailed drug data with comprehensive drug target information	<a href="https://www.drugbank.ca/">https://www.drugbank.ca/</a>
SureChEMBL	Database for chemical compounds in patents	<a href="https://www.surechembl.org">https://www.surechembl.org</a>
GDB	Combinatorically generated drug-like small molecule library	<a href="http://gdb.unibe.ch/">http://gdb.unibe.ch/</a>
PDB	3D structure of Macromolecular Structures (including ligands binding to active site of targets)	<a href="https://www.rcsb.org/">https://www.rcsb.org/</a>
QM9	Small organic molecules subset out of the GDB-17 with quantum chemical properties	<a href="http://www.quantum-machine.org/datasets/">http://www.quantum-machine.org/datasets/</a>
ExCAPE-DB	An integrated chemogenomic dataset collected from publicly available databases including structure, target information and activity annotations	<a href="https://solr.ideaconsult.net/search/excape/">https://solr.ideaconsult.net/search/excape/</a>
ZINC	Curated collection of commercially available chemical compounds	<a href="https://zinc15.docking.org/">https://zinc15.docking.org/</a>

GA approach for drug design which was shown to perform better than a SMILES-based RNN, the ChemTS, CVAE, and GVAE with much lower computational cost.

Despite the differences in their mode of operation, some groups have tried to combine these two classes of methods for drug design. For example, an end-to-end model can map each molecule from discrete chemical space into a continuous latent space, that is, the chemical structure can be converted into a numerical vector by the encoder. Such continuous representations are convenient for use in optimization and the resulting optima are subsequently reconstructed into the expected molecules by the decoder. For example, Sattarov et al. applied a generative topographic mapping (GTM) technique, the probabilistic counterpart of self-organizing maps based on Bayesian learning, in the continuous space constructed by a VAE model [57]. GTM was convenient for visualization of the latent space in which target zones can be used for generating novel molecular structures by sampling. They succeeded in generating focused libraries of potential ligands toward the adenosine A<sub>2A</sub> receptor. In addition, Winter et al.

constructed another end-to-end deep learning framework to construct a continuous space and exploited a PSO algorithm on this latent space. They were able to successfully generate ligands with a predicted high affinity to both EGFR and BACE1 [62].

## 6 Conclusion and Perspective

In this review, we give a brief description of algorithms used in drug de novo design, divided in optimization methods on the one hand and DL methods on the other hand. Traditionally, the drug design problem was always addressed as a combinatorial optimization problem. Hence optimization methods were dominant in drug design. With the rise of DL, more and more researchers shifted their interests from optimization algorithms to DL-based methods. The application of deep learning in drug de novo design caused a revolutionary pattern shift in drug discovery. However, DL methods have still a long way to go and traditional optimization algorithms still provide inspiration to improve the capability of drug de novo design. Currently, it is hard to say which kind of methods are dominant for all cases of drug design. Users should select methods based on their own conditions in practice. We also expect more sophisticated AI algorithms being proposed in the future to accelerate drug discovery.

## Acknowledgments

X.L. thanks the Chinese Scholarship Council (CSC) for funding.

*Competing interests:* The authors declare that they have no competing interests.

## References

- Polishchuk PG, Madzhidov TI, Varnek A (2013) Estimation of the size of drug-like chemical space based on GDB-17 data. *J Comput Aided Mol Des* 27(8):675–679. <https://doi.org/10.1007/s10822-013-9672-4>
- Macarron R, Banks MN, Bojanic D et al (2011) Impact of high-throughput screening in biomedical research. *Nat Rev Drug Discov* 10(3):188–195. <https://doi.org/10.1038/nrd3368>
- Giacomini KM, Krauss RM, Roden DM et al (2007) When good drugs go bad. *Nature* 446(7139):975–977. <https://doi.org/10.1038/446975a>
- Lounkine E, Keiser MJ, Whitebread S et al (2012) Large-scale prediction and testing of drug activity on side-effect targets. *Nature* 486(7403):361–367. <https://doi.org/10.1038/nature11159>
- Paul SM, Mytelka DS, Dunwiddie CT et al (2010) How to improve R&D productivity: the pharmaceutical industry’s grand challenge. *Nat Rev Drug Discov* 9(3):203–214. <https://doi.org/10.1038/nrd3078>
- Kapetanovic IM (2008) Computer-aided drug discovery and development (CADD): in silico-chemico-biological approach. *Chem Biol Interact* 171(2):165–176. <https://doi.org/10.1016/j.cbi.2006.12.006>

7. Schneider G, Fechner U (2005) Computer-based de novo design of drug-like molecules. *Nat Rev Drug Discov* 4(8):649–663. <https://doi.org/10.1038/nrd1799>
8. Chen H, Engkvist O, Wang Y et al (2018) The rise of deep learning in drug discovery. *Drug Discov Today*. <https://doi.org/10.1016/j.drudis.2018.01.039>
9. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444. <https://doi.org/10.1038/nature14539>
10. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Paper presented at the proceedings of the 25th international conference on neural information processing systems—volume 1, Lake Tahoe, Nevada.
11. Goodfellow IJ, Pouget-Abadie J, Mirza M et al (2014) Generative adversarial networks. *ArXiv:1406.2661*
12. Silver D, Huang A, Maddison CJ et al (2016) Mastering the game of go with deep neural networks and tree search. *Nature* 529 (7587):484–489. <https://doi.org/10.1038/nature16961>
13. Gomez-Bombarelli R, Wei JN, Duvenaud D et al (2018) Automatic chemical design using a data-driven continuous representation of molecules. *ACS Cent Sci* 4(2):268–276. <https://doi.org/10.1021/acscentsci.7b00572>
14. Nicolaou CA, Brown N (2013) Multi-objective optimization methods in drug design. *Drug Discov Today Technol* 10(3): e427–e435. <https://doi.org/10.1016/j.ddtec.2013.02.001>
15. Sanchez-Lengeling B, Aspuru-Guzik A (2018) Inverse molecular design using machine learning: generative models for matter engineering. *Science* 361 (6400):360–365. <https://doi.org/10.1126/science.aat2663>
16. van Westen GJP, Wegner JK, IJzerman AP et al (2011) Proteochemometric modeling as a tool to design selective compounds and for extrapolating to novel targets. *Med Chem Commun* 2(1):16–30. <https://doi.org/10.1039/C0MD00165A>
17. Rogers D, Hahn M (2010) Extended-connectivity fingerprints. *J Chem Inf Model* 50(5):742–754. <https://doi.org/10.1021/ci100050t>
18. von Lilienfeld OA (2013) First principles view on chemical compound space: gaining rigorous atomistic control of molecular properties. *Int J Quantum Chem* 113(12):1676–1689. <https://doi.org/10.1002/qua.24375>
19. Elton DC, Boukouvalas Z, Fuge MD et al (2019) Deep learning for molecular design—a review of the state of the art. *Mol Syst Design Eng* 4(4):828–849. <https://doi.org/10.1039/C9ME00039A>
20. Noel OB, Andrew D (2018) DeepSMILES: an adaptation of SMILES for use in machine-learning of chemical structures. doi:<https://doi.org/10.26434/chemrxiv.7097960.v1>
21. Josep A-P, Simon Viet J, Oleksii P et al (2019) Randomized SMILES strings improve the quality of molecular generative models. <https://doi.org/10.26434/chemrxiv.8639942.v2>
22. Krenn M, Häse F, Nigam A et al (2019) SELFIES: a robust representation of semantically constrained graphs with an example application in chemistry. *arXiv*. e-prints: arXiv:1905.13741
23. Emmerich MTM, Deutz AH (2018) A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Nat Comput* 17(3):585–609. <https://doi.org/10.1007/s11047-018-9685-y>
24. Mock WBT (2011) Pareto Optimality. In: Chatterjee DK (ed) Encyclopedia of global justice. Springer, Dordrecht, pp 808–809. [https://doi.org/10.1007/978-1-4419-9160-5\\_341](https://doi.org/10.1007/978-1-4419-9160-5_341)
25. Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: empirical results. *Evol Comput* 8 (2):173–195. <https://doi.org/10.1162/106365600568202>
26. Deb K, Pratap A, Agarwal S et al (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *Trans Evol Comp* 6(2):182–197. <https://doi.org/10.1109/4235.996017>
27. Emmerich M, Beume N, Naujoks B. (2005) An EMO Algorithm using the hypervolume measure as selection criterion. In: 2005 evolutionary multi-criterion optimization. Springer Berlin, pp 62–76
28. Wang R, Gao Y, Lai L (2000) LigBuilder: a multi-purpose program for structure-based drug design. *Mol Model Ann* 6(7):498–516. <https://doi.org/10.1007/s0089400060498>
29. Douguet D, Thoreau E, Grassy G (2000) A genetic algorithm for the automated generation of small organic molecules: drug design using an evolutionary algorithm. *J Comput Aided Mol Des* 14(5):449–466. <https://doi.org/10.1023/A:1008108423895>
30. Pegg SC, Haresco JJ, Kuntz ID (2001) A genetic algorithm for structure-based de novo design. *J Comput Aided Mol Des* 15

- (10):911–933. <https://doi.org/10.1023/a:1014389729000>
31. Budin N, Majeux N, Tenette-Souaille C et al (2001) Structure-based ligand design by a build-up approach and genetic algorithm search in conformational space. *J Comput Chem* 22(16):1956–1970. <https://doi.org/10.1002/jcc.1145>
  32. Vinkers HM, de Jonge MR, Daeyaert FF et al (2003) SYNOPSIS: SYNthesize and OPTimize System in Silico. *J Med Chem* 46 (13):2765–2773. <https://doi.org/10.1021/jm030809x>
  33. Douguet D, Munier-Lehmann H, Labesse G et al (2005) LEA3D: a computer-aided ligand design for structure-based drug design. *J Med Chem* 48(7):2457–2468. <https://doi.org/10.1021/jm0492296>
  34. Dey F, Caflisch A (2008) Fragment-based de novo ligand design by multiobjective evolutionary optimization. *J Chem Inf Model* 48 (3):679–690. <https://doi.org/10.1021/ci700424b>
  35. van der Horst E, Marques-Gallego P, Mulder-Krieger T et al (2012) Multi-objective evolutionary design of adenosine receptor ligands. *J Chem Inf Model* 52(7):1713–1721. <https://doi.org/10.1021/ci2005115>
  36. Lameijer EW, Kok JN, Back T et al (2006) The molecule evaluator. An interactive evolutionary algorithm for the design of drug-like molecules. *J Chem Inf Model* 46 (2):545–552. <https://doi.org/10.1021/ci050369d>
  37. Nicolaou CA, Apostolakis J, Pattichis CS (2009) De novo drug design using multiobjective evolutionary graphs. *J Chem Inf Model* 49(2):295–307. <https://doi.org/10.1021/ci800308h>
  38. Fechner U, Schneider G (2006) Flux (1): a virtual synthesis scheme for fragment-based de novo design. *J Chem Inf Model* 46 (2):699–707. <https://doi.org/10.1021/ci0503560>
  39. Schneider G, Lee ML, Stahl M et al (2000) De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks. *J Comput Aided Mol Des* 14 (5):487–494. <https://doi.org/10.1023/a:1008184403558>
  40. Sengupta S, Bandyopadhyay S (2012) De novo design of potential RecA inhibitors using multi objective optimization. *IEEE/ACM Trans Comput Biol Bioinform* 9 (4):1139–1154. <https://doi.org/10.1109/TCBB.2012.35>
  41. Pearlman DA, Murcko MA (1996) CONCERTS: dynamic connection of fragments as an approach to de novo ligand design. *J Med Chem* 39(8):1651–1663. <https://doi.org/10.1021/jm9507921>
  42. Dean PM, Firth-Clark S, Harris W et al (2006) SkelGen: a general tool for structure-based de novo ligand design. *Expert Opin Drug Discov* 1(2):179–189. <https://doi.org/10.1517/17460441.1.2.179>
  43. Hartenfeller M, Proschak E, Schuller A et al (2008) Concept of combinatorial de novo design of drug-like molecules by particle swarm optimization. *Chem Biol Drug Des* 72(1):16–26. <https://doi.org/10.1111/j.1747-0285.2008.00672.x>
  44. Vikhar PA (2016) Evolutionary algorithms: a critical review and its future prospects. In: 2016 international conference on global trends in signal processing, information computing and communication (ICGTSPICC), 22–24 Dec. 2016. pp 261–265. <https://doi.org/10.1109/ICGTSPICC.2016.7955308>
  45. Bäck T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, Oxford
  46. Mitchell M (1998) An introduction to genetic algorithms. MIT Press, Cambridge
  47. Neill MO, Ryan C (2001) Grammatical evolution. *IEEE Trans Evol Comput* 5 (4):349–358. <https://doi.org/10.1109/4235.942529>
  48. Hansen N, Kern S (2004) Evaluating the CMA evolution strategy on multimodal test functions. In: Yao X, Burke EK, Lozano JA et al (eds) Parallel problem solving from nature—PPSN VIII. Springer, Berlin, pp 282–291
  49. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95—international conference on neural networks, 27 Nov.–1 Dec. 1995, vol 1944. pp 1942–1948. <https://doi.org/10.1109/ICNN.1995.488968>
  50. Oleksii P, Simon J, Panagiotis-Christos K et al (2019) A de novo molecular generation method using latent vector based generative adversarial network. <https://doi.org/10.26434/chemrxiv.8299544.v2>
  51. Puttin E, Asadulaev A, Vanhaelen Q et al (2018) Adversarial threshold neural computer for molecular de novo design. *Mol Pharm* 15 (10):4386–4397. <https://doi.org/10.1021/acs.molpharmaceut.7b01137>
  52. Blaschke T, Olivecrona M, Engkvist O et al (2018) Application of generative autoencoder

- in de novo molecular design. Mol Informatics 37(1–2). <https://doi.org/10.1002/minf.201700123>
53. Yang X, Zhang J, Yoshizoe K et al (2017) ChemTS: an efficient python library for de novo molecular generation. Sci Technol Adv Mater 18(1):972–976. <https://doi.org/10.1080/14686996.2017.1401424>
  54. Kang S, Cho K (2019) Conditional molecular design with deep generative models. J Chem Inf Model 59(1):43–52. <https://doi.org/10.1021/acs.jcim.8b00263>
  55. Griffiths R-R, Hernández-Lobato JM (2017) Constrained Bayesian optimization for automatic chemical design. eprint arXiv:170905501:arXiv:1709.05501
  56. Merk D, Friedrich L, Grisoni F et al (2018) De novo design of bioactive small molecules by artificial intelligence. Mol Informatics 37(1–2). <https://doi.org/10.1002/minf.201700153>
  57. Sattarov B, Baskin II, Horvath D et al (2019) De novo molecular design by combining deep autoencoder recurrent neural networks with generative topographic mapping. J Chem Inf Model 59(3):1182–1196. <https://doi.org/10.1021/acs.jcim.8b00751>
  58. Popova M, Isayev O, Tropsha A (2018) Deep reinforcement learning for de novo drug design. Sci Adv 4(7):eaap7885. <https://doi.org/10.1126/sciadv.aap7885>
  59. Polykovskiy D, Zhebrak A, Vetrov D et al (2018) Entangled conditional adversarial autoencoder for de novo drug discovery. Mol Pharm 15(10):4398–4405. <https://doi.org/10.1021/acs.molpharmaceut.8b00839>
  60. Segler MHS, Kogej T, Tyrchan C et al (2018) Generating focused molecule libraries for drug discovery with recurrent neural networks. ACS Cent Sci 4(1):120–131. <https://doi.org/10.1021/acscentsci.7b00512>
  61. Gupta A, Muller AT, Huisman BJH et al (2018) Generative recurrent networks for de novo drug design. Mol Informatics 37(1–2). <https://doi.org/10.1002/minf.201700111>
  62. Winter R, Montanari F, Steffen A et al (2019) Efficient multi-objective molecular optimization in a continuous latent space. Chem Sci. <https://doi.org/10.1039/C9SC01928F>
  63. Bjerrum EJ, Sattarov B (2018) Improving chemical autoencoder latent space and molecular de novo generation diversity with hetero-encoders. Biomol Ther 8(4). <https://doi.org/10.3390/biom8040131>
  64. Lim J, Ryu S, Kim JW et al (2018) Molecular generative model based on conditional variational autoencoder for de novo molecular design. J Chem 10(1):31. <https://doi.org/10.1186/s13321-018-0286-7>
  65. Liu X, Ye K, van Vlijmen HWT et al (2019) An exploration strategy improves the diversity of de novo ligands using deep reinforcement learning: a case for the adenosine A2A receptor. J Chem 11(1):35. <https://doi.org/10.1186/s13321-019-0355-6>
  66. Olivecrona M, Blaschke T, Engkvist O et al (2017) Molecular de-novo design through deep reinforcement learning. J Chem 9(1):48. <https://doi.org/10.1186/s13321-017-0235-x>
  67. Zhou Z, Kearnes S, Li L et al (2018) Optimization of molecules via deep reinforcement learning. eprint arXiv:1810.08678:arXiv:1810.08678
  68. Lima Guimaraes G, Sanchez-Lengeling B, Outeiral C et al (2017) Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. arXiv e-prints: arXiv:1705.10843
  69. Putin E, Asadulaev A, Ivanenkov Y et al (2018) Reinforced adversarial neural computer for de novo molecular design. J Chem Inf Model 58(6):1194–1204. <https://doi.org/10.1021/acs.jcim.7b00690>
  70. Dai H, Tian Y, Dai B et al (2018) Syntax-directed variational autoencoder for structured data. arXiv e-prints
  71. Kusner MJ, Paige B, Hernández-Lobato JM (2017) Grammar variational autoencoder. eprint arXiv:170301925:arXiv:1703.01925
  72. Skalic M, Jimenez J, Sabbadin D et al (2019) Shape-based generative modeling for de novo drug design. J Chem Inf Model 59(3):1205–1214. <https://doi.org/10.1021/acs.jcim.8b00706>
  73. Aumentado-Armstrong T (2018) Latent molecular optimization for targeted therapeutic design. eprint arXiv:180902032:arXiv:1809.02032
  74. Simonovsky M, Komodakis N (2018) Graph-VAE: towards generation of small graphs using variational autoencoders. eprint arXiv:180203480:arXiv:1802.03480
  75. Liu Q, Allamanis M, Brockschmidt M et al (2018) Constrained graph variational autoencoders for molecule design. eprint arXiv:180509076:arXiv:1805.09076
  76. You J, Liu B, Ying R et al (2018) Graph convolutional policy network for goal-directed molecular graph generation. eprint arXiv:180602473:arXiv:1806.02473
  77. Jin W, Barzilay R, Jaakkola T (2018) Junction tree variational autoencoder for molecular

- graph generation. eprint arXiv:180204364: arXiv:1802.04364
78. Popova M, Shvets M, Oliva J et al (2019) MolecularRNN: generating realistic molecular graphs with optimized properties. eprint arXiv:190513372:arXiv:1905.13372
79. Bradshaw J, Paige B, Kusner MJ et al (2019) A model to search for synthesizable molecules. eprint arXiv:190605221: arXiv:1906.05221
80. Stahl N, Falkman G, Karlsson A et al (2019) Deep reinforcement learning for multiparameter optimization in de novo drug design. *J Chem Inf Model.* <https://doi.org/10.1021/acs.jcim.9b00325>
81. Miljanovic M (2012) Comparative analysis of recurrent and finite impulse response neural networks in time series prediction. *Ind J Comp Sci Eng* 3
82. Graves A, Liwicki M, Fernández S et al (2009) A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans Pattern Anal Mach Intell* 31 (5):855–868. <https://doi.org/10.1109/TPAMI.2008.137>
83. Sak H, Senior A, Beaufays F (2014) Long short-term memory recurrent neural network architectures for large scale acoustic modeling. Proceedings of the annual conference of the international speech communication association, INTERSPEECH:338–342
84. Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Comput* 9 (8):1735–1780
85. Chung J, Gulcehre C, Cho K et al (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. ArXiv:1412.3555
86. Sumita M, Yang X, Ishihara S et al (2018) Hunting for organic molecules with artificial intelligence: molecules optimized for desired excitation energies. *ACS Cent Sci* 4 (9):1126–1133. <https://doi.org/10.1021/acscentsci.8b00213>
87. Arus-Pous J, Blaschke T, Ulander S et al (2019) Exploring the GDB-13 chemical space using deep generative models. *J Chem* 11(1):20. <https://doi.org/10.1186/s13321-019-0341-z>
88. Kramer MA (1991) Nonlinear principal component analysis using autoassociative neural networks. *AICHE J* 37(2):233–243. <https://doi.org/10.1002/aic.690370209>
89. Kingma D, Welling M (2014) Auto-encoding variational bayes. arXiv e-prints: arXiv:1312.6114
90. Doersch C (2016) Tutorial on variational autoencoders. arXiv e-prints: arXiv:1606.05908
91. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. *J Artif Int Res* 4(1):237–285
92. Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn* 8 (3):229–256. <https://doi.org/10.1007/BF00992696>
93. Arjovsky M, Chintala S, Bottou L (2017) Wasserstein GAN. arXiv e-prints: arXiv:1701.07875
94. The GAN Zoo. <https://github.com/hindupuravinash/the-gan-zoo>
95. De Cao N, Kipf T (2018) MolGAN: an implicit generative model for small molecular graphs. eprint arXiv:180511973: arXiv:1805.11973
96. Makhzani A, Shlens J, Jaitly N et al (2015) Adversarial autoencoders. arXiv e-prints: arXiv:1511.05644
97. Gaulton A, Bellis LJ, Bento AP et al (2012) ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic Acids Res* 40 (Database issue):D1100–D1107. <https://doi.org/10.1093/nar/gkr777>
98. Mendez D, Gaulton A, Bento AP et al (2019) ChEMBL: towards direct deposition of bioassay data. *Nucleic Acids Res* 47(D1): D930–D940. <https://doi.org/10.1093/nar/gky1075>
99. Wang Y, Xiao J, Suzek TO et al (2009) PubChem: a public information system for analyzing bioactivities of small molecules. *Nucleic Acid Res* 37(Web Server issue): W623–W633. <https://doi.org/10.1093/nar/gkp456>
100. Sterling T, Irwin JJ (2015) ZINC 15—ligand discovery for everyone. *J Chem Inf Model* 55 (11):2324–2337. <https://doi.org/10.1021/acs.jcim.5b00559>
101. Wishart DS, Feunang YD, Guo AC et al (2018) DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Res* 46(D1):D1074–D1082. <https://doi.org/10.1093/nar/gkx1037>
102. Wishart DS, Knox C, Guo AC et al (2008) DrugBank: a knowledgebase for drugs, drug actions and drug targets. *Nucleic Acids Res* 36(Database issue):D901–D906. <https://doi.org/10.1093/nar/gkm958>



# Chapter 7

## Data Integration Using Advances in Machine Learning in Drug Discovery and Molecular Biology

Irene Lena Hudson

### Abstract

While the term artificial intelligence and the concept of deep learning are not new, recent advances in high-performance computing, the availability of large annotated data sets required for training, and novel frameworks for implementing deep neural networks have led to an unprecedented acceleration of the field of molecular (network) biology and pharmacogenomics. The need to align biological data to innovative machine learning has stimulated developments in both data integration (fusion) and knowledge representation, in the form of heterogeneous, multiplex, and biological networks or graphs. In this chapter we briefly introduce several popular neural network architectures used in deep learning, namely, the fully connected deep neural network, recurrent neural network, convolutional neural network, and the auto-encoder. Deep learning predictors, classifiers, and generators utilized in modern feature extraction may well assist interpretability and thus imbue AI tools with increased explication, potentially adding insights and advancements in novel chemistry and biology discovery.

The capability of learning representations from structures directly without using any predefined structure descriptor is an important feature distinguishing deep learning from other machine learning methods and makes the traditional feature selection and reduction procedures unnecessary. In this chapter we briefly show how these technologies are applied for data integration (fusion) and analysis in drug discovery research covering these areas: (1) application of convolutional neural networks to predict ligand–protein interactions; (2) application of deep learning in compound property and activity prediction; (3) de novo design through deep learning. We also: (1) discuss some aspects of future development of deep learning in drug discovery/chemistry; (2) provide references to published information; (3) provide recently advocated recommendations on using artificial intelligence and deep learning in -omics research and drug discovery.

**Key words** Data integration, Deep learning, Drug discovery, Chemistry, Network biology, Embeddings, Deep neural network, Recurrent neural network, Autoencoder, Convolution graph network

---

### 1 Introduction

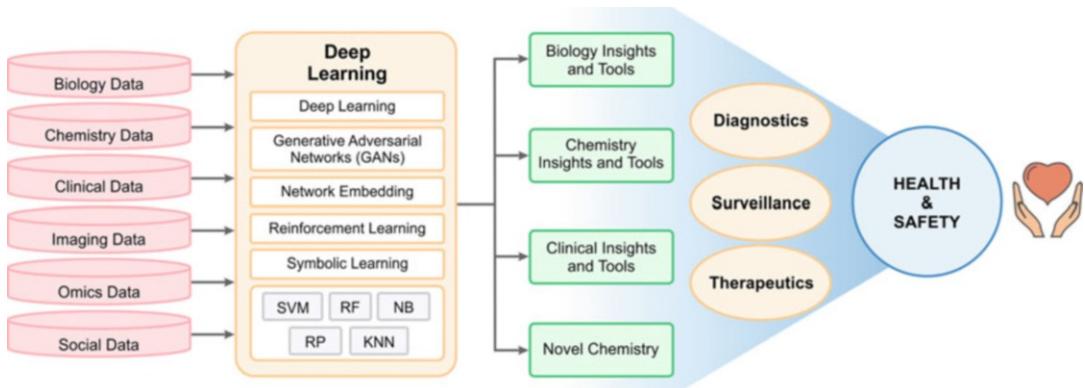
There are potentially great opportunities for improved biomedical data integration (fusion) given recent advances in next-generation machine learning such as deep learning (DL) [1]. Recent advances in high-performance computing, the availability of large annotated data sets required for training, and novel frameworks for

implementing deep neural networks (DNNs) have led to an unprecedented acceleration of the field of DL and its applications. This is particularly so at the intersection of molecular (network) biology, computational chemistry, and drug discovery, given the vast amounts of data being generated, at different scales and multiple dimensions [2].

Because DL methods attempt to construct hidden layers that learn features that best predict successful outcomes for a given task, they can identify novel patterns in complex datasets that would have been overlooked by other techniques [3, 4]. This makes DL an especially powerful tool for biological applications and enables extraction of the most predictive features from complex datasets.

The use of multiple omics techniques (i.e., genomics, transcriptomics, proteomics, and metabolomics) is becoming increasingly popular. However, extracting intrinsic valuable knowledge from omics big data remains a significant problem in bioinformatics, computational and molecular biology, and drug discovery. This is due to inherent data differences making integrating multiple omics platforms an ongoing challenge [5, 6]. The intention of data fusion is for new integrative methods to better collectively mine multiple different types of biological data and produce enriched holistic, systems-level biological understandings. As discussed in an excellent review by Zitnik et al. [7], methods for data integration (fusion) should have the following features: (1) they generate diverse types of prediction outputs, for example in the prediction of quantitative or categorical properties (labels, e.g., gene functions) for biomedical entities (e.g., genes); (2) they can be used in many studies to integrate a large number of networks, whether protein–protein and/or genetic interaction networks. Moreover, apart from predicting labels of individual entities, many integrative studies aim to predict relationships, that is, molecular interactions, functional associations, or causal relationships between biomedical entities (e.g., a drug’s structural similarity, a drug’s phenotypic similarity, and target similarity), to predict new relationships between a drug and proteins that the drug might target, the so-called drug–target interactions [8]. Data integration methods also aim to identify complex structures, such as gene modules or clusters detected in a combined gene interaction network [9], and to generate structured outputs, such as gene regulatory networks [10, 11].

DL is an emerging branch of machine learning, which has shown significant utility in machine vision, voice and signal processing, sequence and text prediction, and likewise in emergent computational biology [1, 11, 12]. Overall DL has several implementation modalities—these are the artificial neural networks (ANNs) [13], and deep structured learning, hierarchical learning, all of which traditionally apply a class of structured networks to infer quantitative properties between responses and causes within a



**Fig. 1** A schema of knowledge and applications of DL and other ML techniques. Adapted from Zhavoronkov [2]

group of data. The capability of learning representations from structures directly without using any predefined structure descriptor is an important feature distinguishing DL from other machine learning methods—it basically makes the traditional feature selection and reduction procedures unnecessary. It is worth noting that DL methods are also highly context specific, as is molecular biology per se. Accordingly, any newly developed or advocated approaches will need to be carefully selected, based on the nature of the different assortments of domain-specific models, specific types of data, and the varying types of biomedical outcomes [7]. One problem area of AI is its lack of interpretability, particularly in the area of drug discovery [2, 10]. However, DL predictors, classifiers, and generators utilized in modern feature extraction and feature selection techniques may well assist interpretability and thus imbue DL tools with increased explication, potentially adding insights in both chemistry and biology discovery [2]. A schema of the body of knowledge and the range of applications of DL and other machine learning techniques is given in Fig. 1.

In this chapter we briefly introduce several popular NN architectures used in DL, namely the fully connected deep neural network (DNN), recurrent neural network (RNN), convolutional neural network (CNN), and the autoencoder (AE). We briefly show how these technologies are applied for data integration and analysis in drug discovery research in (1) application of convolutional neural networks (CNNs) to predict ligand–protein interactions; (2) application of DL in compound property and activity prediction; (3) de novo design through DL. Furthermore, we: (1) discuss some aspects of future development of DL in drug discovery and chemistry; (2) provide references to published information and up to date software available on various platforms; (3) provide recently advocated recommendations on using DL in -omics research and drug discovery. An overview of principal terms used in DL is given in Table 1 of Ching et al. [14]. We also discuss

future developments and challenges of DL at the intersection of DL and network biology. Network biology is an area of research that deals with biological networks and large multidimensional datasets and which potentially impacts drug discovery, microbiome research, disease biology, and synthetic biology. Network biology has also been suggested as a powerful paradigm for representing, interpreting, and visualizing biological data [10, 11]. We overview network embedding, which is a standard approach to computing networks to transform biological data into the so-called vectorial data, so as to enable similarity search, clustering, and visualization [10, 11, 15].

---

## 2 Deep Learning: Next-Generation Machine Learning

DL is viewed as a modern reincarnation of ANNs which uses sophisticated, multilevel DNNs to create systems that perform feature detection from massive amounts of unlabeled or labeled training data [4]. The major difference between DL and traditional ANNs is the scale and complexity of the networks used. In neural networks, input features are fed to an input layer, and after a number of nonlinear transformations using hidden layers, the predictions are generated by an output layer. The learning process in ANNs refers to the process of obtaining the optimal set of model parameters that translate the features in the input data into accurate predictions of the labels. In biological applications, for example, features may contain one or more types of data, such as gene expression profiles, a genomic sequence, [protein–protein interactions](#), metabolite concentrations, or copy number alterations. In contrast, DL can have a large number of hidden layers because it uses more powerful CPU and GPU hardware, whereas traditional neural networks normally use one or two hidden layers because of hardware limitations. There remain ongoing and numerous algorithmic improvements in DL [1, 13, 16].

### 2.1 Deep Neural Networks

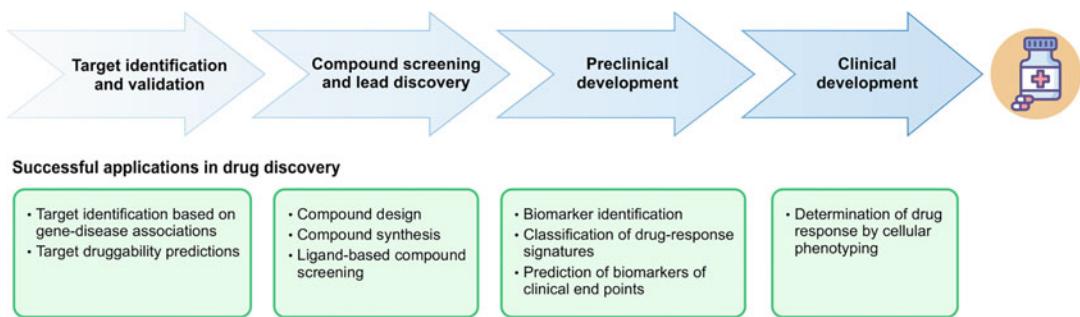
DNNs are efficient algorithms based on the use of compositional layers of neurons, with clear advantages to the challenges presented by -omics data. The DNN contains multiple hidden layers with each layer comprising hundreds of nonlinear process units. DNNs can accommodate a large number of input features and the neurons in different layers are able to automatically extract features at different hierarchical levels. Essentially a DNN is a neural network that includes multiple hidden layers; the greater the number of hidden layers, the deeper the neural network, allowing researchers to define the number and size of the hidden layers depending on the purpose of the learning model. The major advantage of DNNs is that they have several different flexible architectures. In the first architecture, namely, the deep CNNs, some of the hidden layers are only locally

(rather than globally) connected to the next hidden layer. CNNs have been shown to achieve best predictive performance in areas such as speech and image recognition by hierarchically composing simple local features into complex models [4, 14]. Graph convolution networks (GCNs) are a special type of CNN that can be applied to structured data in the form of graphs or networks. The second architecture is the RNN, which takes the form of a chain of repeating modules of neural networks in which connections between nodes form a directed graph along a sequence, allowing analysis of dynamic changes over time. The third network architecture is the deep autoencoder neural network, which is an unsupervised learning algorithm that applies backpropagation to project its input to its output for the purpose of dimension reduction [17]. This preserves the important random variables of the data, while removing the nonessential parts. As such, the deep autoencoder neural network is designed to precisely extract coding or representation features using data-driven learning [12, 18, 19]. For high-dimensional data, dimension reduction or compression is a necessity in preprocessing of raw data. Instead of predicting labels of input instances, the purpose of the decoder NN is to reconstruct its own inputs from fewer numbers of hidden units. Usually, the AE is used for nonlinear dimensionality reduction and of late AE concepts have been more widely used for learning generative models from data [17]. The fourth network architecture, generative adversarial networks (GANs), consist of any two networks (often a combination of feedforward neural networks and CNNs), where one is tasked to generate content and the other to classify that content [20, 21].

## 2.2 Drug Discovery

While the term AI and the concept of DL are not new, developments in high-performance computing, along with availability of large annotated data sets vital for training, and novel frameworks for executing DNNs have led to an unparalleled acceleration in drug discovery, and biomarker development, with the potential to advance novel chemistry [2, 22]. We refer the reader to Chen et al. [16] which describes recent work on applications of DL in drug discovery research and introduces many popular DL architectures. It is now appreciated that DL architectures potentially offer unlimited opportunities in computational pharmacology to predict and understand how drugs affect the human body, support decision making in the drug discovery pipeline process, improve clinical practice and circumvent unwanted side effects. For an excellent review, see Zitnik et al. [7].

The applications of DNNs in drug discovery have been numerous and include bioactivity prediction [23]. Recent applications of DL and machine intelligence also include in silico drug discovery and the development of methods, tools, and databases [16]. Also many pharmaceutical companies have performed the so-called



**Fig. 2** Successful ML applications in drug discovery: from target identification to clinical development. Adapted from Vamathevan et al. [21]

rational drug discovery through various -omics and structure-based drug development. Since 2017, for example, drug companies have been making much progress through the fusion of AI and more recently DL, with previously developed technologies for drug discovery [24] (see Fig. 1). We also refer the reader to the article by Goh et al. [25], regarding applications of DL in computational chemistry and relevant questions as to how accurate the predictions by DNNs in the area of chemistry are. In Fig. 2 we illustrate how these DL technologies are applied in drug discovery research.

### 2.3 Application of Convolutional Neural Networks to Predict Ligand–Protein Interactions

Assessing the interaction between a protein and a ligand is the crucial part of molecular docking programs and many scoring functions have been developed, either based on force fields or knowledge from existing protein–ligand complex structures [26]. Now CCNs and DNNs have recently been used to score protein–ligand interaction. A typical example is the investigation done by Ragoza et al. [27]; see also Pereira et al. [28]. As yet it has not yet been established whether CNNs consistently improve results compared to currently used scoring functions [29].

### 2.4 Application of Deep Learning in Compound Property and Activity Prediction

Fully connected DNNs have been adopted for activity prediction, when compounds are presented by the same number of molecular descriptors [30]. DNN models have achieved high accuracy in predicting drug indications and have proved valuable for drug repurposing (see Table 1). The DNN method can identify drugs with shared therapeutic and biological targets even when the compounds are structurally dissimilar, thereby uncovering previously unreported functional relationships between compounds [2]. We refer the reader also to a recent article by Rodrigues et al. [31]. These authors show how learning algorithms now provide an attractive means to deconvolute drug–target networks and generate statistically based research hypotheses to facilitate prioritization of macromolecular targets for bioactive agents—this all aided by the wealth of publicly available bioactivity data and advances in learning algorithms.

**Table 1**  
**Applications and development of deep learning in drug discovery and chemistry**

DL method	Aim, method, purpose, and outcome(s)	References
Training DNNs for predicting pharmacological properties of small molecules and drug repurposing	Training DNNs on transcriptional response data for prediction of pharmacological properties of small molecules and using transcriptomic data for drug repurposing.	[57]
Training DNNs for deep biomarker development	DNNs to develop biomarkers of human aging.	[58]
Training DNNs for novel chemistry	DNNs used for development of novel chemistry and biomedicine.	[54]
Drug repurposing via deep embeddings to identify new functional relationships between compounds	Development of a new method to assess the compound functional similarity based on gene expression data for drug repurposing. The method identified drugs with common therapeutic and biological targets for compounds that were also structurally dissimilar—establishing previously unreported functional relationships between compounds	[59]
Connectivity map for drug repurposing	Drug repurposing using gene expression profiles (transcriptional response profiles) using the Connectivity Map, to connect small molecules, genes, and diseases.	[59–61]
Information extraction from images: computed tomography	A patch-based CNN model combined with support vector machines to predict multidrug resistant patients with tuberculosis—was shown to achieve relatively high classification rates.	[62, 63]
A multitask deep AE to minimize drug side effects	A multitask deep AE was utilized for the prediction of the human cytochrome 450 inhibition, as a roadmap for reducing side effects.	[64]
Random Forest for predicting estrogen receptor binding	Machine learning techniques such as the AdaBoost, Bernoulli naïve Bayes, random forest, support vector classification, and DNNs were compared for predicting the estrogen receptor binding. The random forest outperformed the other algorithms.	[65]
CNN molecular graph in medicinal chemistry—selection of the representation of molecular structure	A CNN molecular graph, based on a geometric DL representation, outperformed methods trained on expert engineered features, noting that there are numerous representations of molecular structures, such as molecular fingerprints, string-based representations, and molecular graphs.	[66]

(continued)

**Table 1**  
(continued)

DL method	Aim, method, purpose, and outcome(s)	References
CNN trained on 2D and 3D images—in chemistry	The wave transform-based representation of the 3D molecular structure was presented. Such CNN-based AEs were shown to have superior than traditional voxel-based representation or the Gaussian blur of atoms for classification.	[67]
Generative chemistry (deep generative models to molecules)—AI imagination	Generative chemistry for drug discovery, biomarker development, and the design of novel materials: <ul style="list-style-type: none"><li>• An adversarial AE was shown generate new promising anticancer compounds.</li><li>• An application of the variational AE to generative chemistry with a Bayesian optimization of chemical properties was recently described.</li><li>• The reinforced adversarial neural computer for de novo molecular design was recently shown to facilitate machine learning in drug discovery.</li><li>• A proposed prototype-driven diversity network, based on variational AE enabled chemists to generate diverse molecules with the desired properties from a molecular prototype.</li><li>• For the first comprehensive review of the generative chemistry..</li></ul>	[22, 34, 68–71]
Molecular de novo design via the <ul style="list-style-type: none"><li>• Entangled conditional adversarial AE.</li><li>• Adversarial threshold neural computer.</li></ul>	Two novel generative models for molecular de novo design were proposed and obtained experimentally validated results: <ul style="list-style-type: none"><li>• The entangled conditional adversarial AE generates molecular structures on the basis of various properties, such as activity against a specific protein, solubility, or ease of synthesis.</li><li>• The adversarial threshold neural computer for molecular de novo design..</li></ul>	[35, 36]

*DL* deep learning, *DNNs* deep neural networks, *CNN* convolutional neural network, *AE* autoencoder, *AI* artificial intelligence

## 2.5 De Novo Design Through Deep Learning

Enabling NNs to learn directly from the molecular structure instead of using predefined molecular descriptors has been achieved via a variant of RNN, called UGRNN, which initially transforms molecular structures into vectors of the same length as the molecular representation and subsequently passes them to a fully connected NN layer to build models [32]. The UGRNN method [32] has demonstrated good capabilities in building predictive

solubility models, with comparable accuracy to models built using molecular descriptors. Another DL application is that of graph convolution models which employs NNs to automatically generate a molecular description vector and vector values learnt by training NNs [33]. We refer the reader to Table 1 for extensions of this approach, and Table 2 for embedded network frameworks advancing GCNs. Another interesting application of DL in chemoinformatics is the generation of new chemical structures through NNs—*see* Table 1. We refer the reader also to Putin et al. and Polykovskiy et al. [34–36]. RNNs have also been used as the basis of generating novel chemical structures [37].

## 2.6 Network Biology

Network biology involves representing, interpreting, and visualizing biological data with the aim of investigating and understanding complex interactions of biomolecules that contribute to the structures and functions of living cells. Network biology has benefited from ML largely via the use of network architectures [38, 39]. The diversity of approaches for network inference is extensive [40, 41]. These reverse-engineering approaches have shown impressive capabilities of learning patterns from input data to in turn generate biologically relevant networks. It has been reported that there are many future prospects for progressing network biology through the integration of multiomics datasets and phenotypic measurements with novel machine learning methods [39, 42] (Fig. 3).

## 2.7 Network Embedding

The need to align biological data to innovative machine learning methods has stimulated developments in both data integration and knowledge representation, in the form of heterogeneous, multiplex, and biological networks or graphs. Given the growth in the so-called knowledge-embedding techniques, large and complex biological networks can now be converted to a vector format that can be utilized by the majority of machine and DL implementations. Methods for network embedding aim to optimize the difference between the node similarities/distances in the original network space and their similarities/distances under the embedding, typically constrained to have a low dimension. Graph drawing algorithms are perhaps the best-known embedding techniques, commonly used to visualize a graph in 2D space. One frequent application of neural networks is to create vector embeddings of entities or assertions by training AE networks with inputs constructed from the graph [10]. The notion is to use such embeddings as inputs to an ML or DL analysis. Embedding approaches have the added advantage that algorithms making use of embeddings are often faster than their counterparts which operate on the original networks. These embeddings can then be the basis of computed similarities, for example, between drugs, proteins, and diseases [15, 41]. Biological applications of embeddings have

**Table 2**  
**Embedding approaches in pharmacogenomics [10]**

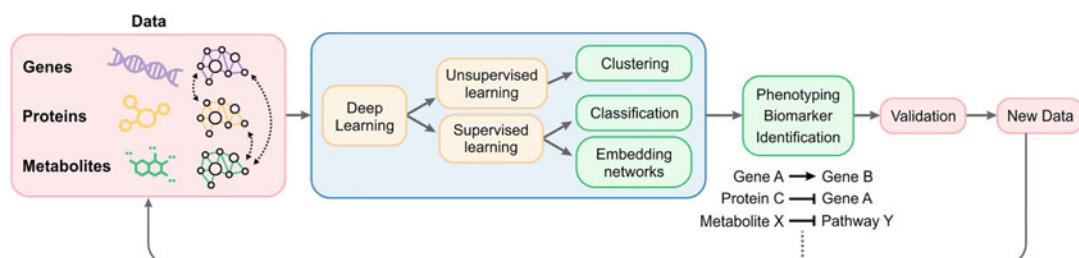
Network embedding via deep neural method	Approach, aim, purpose, and outcome(s)	References
Tools	Summary of network embedding tools that are used in the biomedical field is given in Table 2 of Nelson et al. [10].	[10]
Drug repurposing via deep embeddings	A new method to assess compound functional similarity based on gene expression data for drug repurposing was applied.  This method identifies drugs with common therapeutic/biological targets for compounds that are also structurally dissimilar, thus advancing the identification of previously unknown functional relationships between compounds.	[59]
Drug–target interaction prediction as a link prediction task on a graph, followed by embedding	The following node embedding methods can be used—node2vec, DeepWalk, and LINE to embed nodes into a compact vector space, preserving local network structure, as based on a graph of drugs/chemicals and the proteins with which they interact.  A neural network that learns node embeddings and predicts interactions directly from the graph, is an alternative so-called end-to-end approach.	[8, 44, 72–80].
Detecting drug–drug interactions via GCNs and learnt similarity scores	A method using GCNs to embed a multiview graph to fuse information from multiple views and which enhances learning interpretability was developed by Ma et al. [81].  Model features: <ul style="list-style-type: none"> <li>• Each drug is modeled as a node in a multiview drug association graph, where edges between drugs in different views encode diverse types of similarity between drugs.</li> <li>• The embedding paradigm learns a similarity score between any two drugs, using the scores to predict drug–drug interactions..</li> </ul>	[81–83]
Identification of side effects of drug combinations, via embedding, using graph neural networks based on a multimodal graph	An embedding approach that creates a multimodal graph of protein–protein interactions, drug–protein interactions, and drug–drug interactions, where each drug–drug interaction is labeled by a different edge type representing the type	[84]

(continued)

**Table 2**  
(continued)

Network embedding via deep neural method	Approach, aim, purpose, and outcome(s)	References
	<p>of the side effect was developed by Zitnik et al. [84].</p> <ul style="list-style-type: none"> <li>The approach is based on the multimodal graph and uses graph neural networks as an embedding methodology.</li> <li>The approach considers all types of side effects at once, and learns embeddings of side effects to indicate polypharmacy in patients..</li> </ul>	
Chemical prediction (drug efficacy or solubility) problems via molecular graph convolution models	<p>Chemical prediction problems are graph-level classification problems where individual data examples are graphs (rather than nodes) representing small molecules. Prediction tasks aim to predict molecular properties such as drug efficacy or solubility, to predict which drugs bind to which target proteins, and identify sites at which a particular candidate drug binds to a target protein.</p> <p>The input to a predictor is a small molecule, represented as a graph in which nodes and edges represent atoms and bonds between atoms, respectively.</p>	[69, 85–88]

GCNs graph convolution networks



**Fig. 3** Machine learning applications to build models from -omics data: a workflow. Adapted from Camacho et al. [39]

utilized vector embeddings: (1) for prediction or visualization in low dimensional spaces [43]; (2) to link prediction methods to generate hypotheses of prior unknown relationships [44]; and (3) to generate complex mechanistic accounts of experimental data.

Node and edge embeddings provide a powerful technique by which to propose relationships among entities via similarity functions and have been used to improve link prediction [44]. Similarity based hypotheses from networks and embedding include drug–drug and drug–target interactions, many of which were later applied to drug repurposing. Another method is the graph convolution network (GCN) [45], a semisupervised graph which defines a convolutional operator on the network, and iteratively aggregates embeddings of neighbors of a node, using the aggregated embedding as well as its own embedding at previous iterations to generate the node’s new representation [10]. Network approaches have also emerged as a promising way to address the well documented reducing efficiency of drug discovery pipelines and processes, which has led to a deficit of available treatments required for growing therapeutic needs [46, 47]. This highlights our need to improve our understanding of the therapeutic and side effects of drugs [48, 49].

While traditional chemoinformatic methods have employed vector descriptors of compound structures, as the standard input for their prediction tasks, having a common vector format via embedding may potentially transform drug discovery, by improving the accuracy of predictions and revealing connections between small molecules and other biological entities, such as targets or diseases [43]. Table 2 summarizes embedding methods in biology and drug discovery, the latter pertaining to the areas of drug–target prediction, drug–drug interaction prediction and prediction problems involving small molecules. We refer the reader to the review by Nelson et al., and also to a summary of network embedding tools used in the biomedical field given in Table 2 of Nelson et al. [10].

---

### 3 Challenges for the Future in Network Biology

In this section, we summarize areas of challenge and promising future directions in network biology.

#### 3.1 Dynamic Networks

Methods that have been proposed to tackle dynamic networks to date often assume that the node set is fixed and deal only with dynamics caused by edge deletion and addition.

#### 3.2 Hierarchical Network Structure

Most existing methods focus primarily on designing advanced encoding/decoding functions to capture node pairwise relationships, which only provide insights about local neighborhoods, not global hierarchical network structures characteristic of complex networks [50]. How to design effective network embedding methods that are capable of preserving hierarchical structures of networks remains a direction for further work.

### 3.3 Heterogeneous Networks

Existing network embedding methods largely deal with homogeneous networks, not able to be abstracted as heterogeneous networks with multiple types of nodes or edges. Learning embeddings for heterogeneous networks has been investigated using meta-paths [51].

### 3.4 Scalability

Although DL-based network embedding methods have achieved success, there remains a problem of efficiency, which will be exacerbated with the advent of real-life massive datasets with billions of nodes and edges. Developing computational paradigms for large-scale network processing could be an alternative way to improve efficiency [45], similar to using GPUs for traditional deep models built on grid structured data.

### 3.5 Interpretability

Despite superior performances of DL models, one limitation is their lack of interpretability. Different dimensions in the embedding space usually have no specific meaning, thus it is difficult to comprehend the underlying factors that have been preserved in the latent space. Interpretability aspects of ML models are currently receiving attention [52]. It has been suggested that complex networks can be better comprehended by (1) focusing on some portion of the full network, (2) computing summary statistics about the network, or (3) comparing with other networks, including cross-disciplinary comparisons [11].

---

## 4 Conclusion

The DNN is considered to be one of the most significant computational breakthroughs in the era of big data. The success of DL models has been enabled by the explosive growth in the volume of raw data in addition to significant progress in computing and use of powerful graphical processing units. By way of summary, recent applications of DL in biomedicine have demonstrated DNN's superior performance compared with other ML approaches in a number of biomedical problems [14], including those in image analysis [53], genomics [54], as well as drug discovery and repurposing [25, 55]. However, while studies have demonstrated DL and DNN's ability to solve complex bioinformatic tasks, the methods have several fundamental limitations. For instance, due to high representational power and model complexity, they suffer from overfitting and need large quantities of training data. Some noteworthy suggestions to overcome these difficulties follow. It was recently suggested that one possibility to help overcome this data hungry issue is to generate *in silico* data with properties of real data via GANs [39], which learn to create datasets that are similar to the training data, advocating that this ML approach could be readily extended to the multiomics datasets integral to network biology

[5, 6]. A simple approach would be to use GANs to generate larger expression datasets that can be used in the context of network inference to generate predictive models, for example, of transcriptional regulation. Another novel method has also recently been proposed that involves extracting features learned by neural networks, such as network-centric approaches and decomposition of predicted outputs by backpropagation onto specific input features [56]. Further studies have suggested that multitask deep networks are superior on a broad range of drug discovery datasets [55].

Finally, it has been suggested that software developers create more user-friendly omics integration of web tools and software [5], and advocated that DL advancements in ML are potentially highly beneficial for the integration and interpretation of multiomics data. Fundamental requirements to progress truly integrated, systems biology research in multiomics have been identified by Pinu et al. [5]. Noteworthy also is that McGillivray et al. [11] have supported the notion that network algorithms that include ML and network propagation methods will potentially provide improvements to network predictions but that it is vital to design efficient, scalable algorithms for large search spaces that provide accurate approximations of actual network properties; and that these predictions based on network analyses will in time require validation on a genomic scale.

---

## Acknowledgments

The author is most grateful for editorial and graphic design assistance from Dr. Sean A. Hudson (Hudson Software Development, Australia, [www.hudson-software.droppages.com](http://www.hudson-software.droppages.com)).

## References

1. Tang B, Pan Z, Yin K et al (2019) Recent advances of deep learning in bioinformatics and computational biology. *Front Genet* 10:214. <https://doi.org/10.3389/fgene.2019.00214>
2. Zhavoronkov A (2018) Artificial intelligence for drug discovery, biomarker development, and generation of novel chemistry. *Mol Pharm* 15(10):4311–4313. <https://doi.org/10.1021/acs.molpharmaceut.8b00930>
3. Angermueller C, Parnamaa T, Parts L et al (2016) Deep learning for computational biology. *Mol Syst Biol* 12(7):878. <https://doi.org/10.15252/msb.20156651>
4. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444. <https://doi.org/10.1038/nature14539>
5. Pinu FR, Goldansaz SA, Jaine J (2019) Translational metabolomics: current challenges and future opportunities. *Meta* 9(6). <https://doi.org/10.3390/metabo9060108>
6. Pinu FR, Beale DJ, Paten AM et al (2019) Systems biology and multi-omics integration: viewpoints from the metabolomics research community. *Meta* 9(4). <https://doi.org/10.3390/metabo9040076>
7. Zitnik M, Nguyen F, Wang B et al (2019) Machine learning for integrating data in biology and medicine: principles, practice, and opportunities. *Inf Fusion* 50:71–91. <https://doi.org/10.1016/j.inffus.2018.09.012>
8. Gonen M (2012) Predicting drug-target interactions from chemical and genomic kernels using Bayesian matrix factorization.

- Bioinformatics 28(18):2304–2310. <https://doi.org/10.1093/bioinformatics/bts360>
9. Zitnik M, Zupan B (2015) Gene network inference by fusing data from diverse distributions. Bioinformatics 31(12):i230–i239. <https://doi.org/10.1093/bioinformatics/btv258>
  10. Nelson W, Zitnik M, Wang B et al (2019) To embed or not: network embedding as a paradigm in computational biology. Front Genet 10(381):381. <https://doi.org/10.3389/fgene.2019.00381>
  11. McGillivray P, Clarke D, Meyerson W et al (2018) Network analysis as a grand unifier in biomedical data science. Ann Rev Biomed Data Sci 1(1):153–180. <https://doi.org/10.1146/annurev-biodatasci-080917-013444>
  12. Min S, Lee B, Yoon S (2017) Deep learning in bioinformatics. Brief Bioinform 18(5):851–869. <https://doi.org/10.1093/bib/bbw068>
  13. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117. <https://doi.org/10.1016/j.neunet.2014.09.003>
  14. Ching T, Himmelstein DS, Beaulieu-Jones BK et al (2018) Opportunities and obstacles for deep learning in biology and medicine. J R Soc Interface 15(141). <https://doi.org/10.1098/rsif.2017.0387>
  15. Cai H, Zheng VW, Chang KC-C (2018) A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans Knowl Data Eng 30(9):1616–1637. <https://doi.org/10.1109/tkde.2018.2807452>
  16. Chen H, Engkvist O, Wang Y et al (2018) The rise of deep learning in drug discovery. Drug Discov Today 23(6):1241–1250. <https://doi.org/10.1016/j.drudis.2018.01.039>
  17. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507. <https://doi.org/10.1126/science.1127647>
  18. Zeng K, Yu J, Wang R et al (2017) Coupled deep autoencoder for single image super-resolution. IEEE Trans Cybern 47(1):27–37. <https://doi.org/10.1109/TCYB.2015.2501373>
  19. Yang W, Liu Q, Wang S et al (2018) Down image recognition based on deep convolutional neural network. Informat Process Agricul 5(2):246–252. <https://doi.org/10.1016/j.inpa.2018.01.004>
  20. Smith JS, Roitberg AE, Isayev O (2018) Transforming computational drug discovery with machine learning and AI. ACS Med Chem Lett 9(11):1065–1069. <https://doi.org/10.1021/acsmmedchemlett.8b00437>
  21. Vamathevan J, Clark D, Czodrowski P et al (2019) Applications of machine learning in drug discovery and development. Nat Rev Drug Discov 18(6):463–477. <https://doi.org/10.1038/s41573-019-0024-5>
  22. Hochreiter S, Klambauer G, Rarey M (2018) Machine learning in drug discovery. J Chem Inf Model 58(9):1723–1724. <https://doi.org/10.1021/acs.jcim.8b00478>
  23. Rifaiglu AS, Atas H, Martin MJ et al (2018) Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. Brief Bioinform. <https://doi.org/10.1093/bib/bby061>
  24. Mak KK, Pichika MR (2019) Artificial intelligence in drug development: present status and future prospects. Drug Discov Today 24(3):773–780. <https://doi.org/10.1016/j.drudis.2018.11.014>
  25. Goh GB, Hodas NO, Vishnu A (2017) Deep learning for computational chemistry. J Comput Chem 38(16):1291–1307. <https://doi.org/10.1002/jcc.24764>
  26. Pagadala NS, Syed K, Tuszyński J (2017) Software for molecular docking: a review. Biophys Rev 9(2):91–102. <https://doi.org/10.1007/s12551-016-0247-1>
  27. Ragoza M, Hochuli J, Idrobo E et al (2017) Protein-ligand scoring with convolutional neural networks. J Chem Inf Model 57(4):942–957. <https://doi.org/10.1021/acs.jcim.6b00740>
  28. Pereira JC, Caffarena ER, Dos Santos CN (2016) Boosting Docking-Based Virtual Screening with Deep Learning. J Chem Inf Model 56(12):2495–2506. <https://doi.org/10.1021/acs.jcim.6b00355>
  29. Paladino A, Marchetti F, Rinaldi S et al (2017) Protein design: from computer models to artificial intelligence. Wiley Inter Rev Comput Mol Sci 7(5):e1318. <https://doi.org/10.1002/wcms.1318>
  30. Yoshiki K, Shinji H, Hitoshi G (2016) Molecular activity prediction using deep learning software library. 2016 international conference on advanced informatics: concepts, theory and application (ICAICTA):1–6
  31. Rodrigues T, Bernardes GJL (2019) Machine learning for target discovery in drug development. Curr Opin Chem Biol 56:16–22. <https://doi.org/10.1016/j.cbpa.2019.10.003>
  32. Lusci A, Pollastri G, Baldi P (2013) Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. J Chem Inf Model 53(7):1563–1575. <https://doi.org/10.1021/ci400187y>

33. Duvenaud D, Maclaurin D, Aguilera-Iparraguirre J et al (2015) Convolutional networks on graphs for learning molecular fingerprints. In: Paper presented at the 29th annual conference on neural information processing systems, NIPS 2015, Montreal, Canada, 7 through 12 December 2015
34. Putin E, Asadulaev A, Ivanenkov Y et al (2018) Reinforced adversarial neural computer for de novo molecular design. *J Chem Inf Model* 58(6):1194–1204. <https://doi.org/10.1021/acs.jcim.7b00690>
35. Putin E, Asadulaev A, Vanhaelen Q et al (2018) Adversarial threshold neural computer for molecular de novo design. *Mol Pharm* 15(10):4386–4397. <https://doi.org/10.1021/acs.molpharmaceut.7b01137>
36. Polykovskiy D, Zhebrak A, Vetrov D et al (2018) Entangled conditional adversarial auto-encoder for de novo drug discovery. *Mol Pharm* 15(10):4398–4405. <https://doi.org/10.1021/acs.molpharmaceut.8b00839>
37. Segler MHS, Kogej T, Tyrchan C et al (2018) Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Cent Sci* 4(1):120–131. <https://doi.org/10.1021/acscentsci.7b00512>
38. Margolin AA, Nemenman I, Basso K et al (2006) ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics* 7(1):S7. <https://doi.org/10.1186/1471-2105-7-S1-S7>
39. Camacho DM, Collins KM, Powers RK et al (2018) Next-generation machine learning for biological networks. *Cell* 173(7):1581–1592. <https://doi.org/10.1016/j.cell.2018.05.015>
40. De Smet R, Marchal K (2010) Advantages and limitations of current network inference methods. *Nat Rev Microbiol* 8(10):717–729. <https://doi.org/10.1038/nrmicro2419>
41. Lecca P, Priami C (2013) Biological network inference for drug discovery. *Drug Discov Today* 18(5-6):256–264. <https://doi.org/10.1016/j.drudis.2012.11.001>
42. Walsh LA, Alvarez MJ, Sabio EY et al (2017) An integrated systems biology approach identifies TRIM25 as a key determinant of breast cancer metastasis. *Cell Rep* 20(7):1623–1640. <https://doi.org/10.1016/j.celrep.2017.07.052>
43. Duran-Frigola M, Fernández-Torras A, Berthoni M et al (2019) Formatting biological big data for modern machine learning in drug discovery. *Wiley Inter Rev Comput Mol Sci* 9(6):e1408. <https://doi.org/10.1002/wcms.1408>
44. Crichton G, Guo Y, Pyysalo S et al (2018) Neural networks for link prediction in realistic biomedical graphs: a multi-dimensional evaluation of graph embedding-based approaches. *BMC Bioinformatics* 19(1):176. <https://doi.org/10.1186/s12859-018-2163-9>
45. Bronstein MM, Bruna J, LeCun Y et al (2017) Geometric deep learning: going beyond Euclidean data. *IEEE Signal Process Mag* 34(4):18–42. <https://doi.org/10.1109/MSP.2017.2693418>
46. Hodos RA, Kidd BA, Shameer K et al (2016) In silico methods for drug repurposing and pharmacology. *Wiley Interdiscip Rev Syst Biol Med* 8(3):186–210. <https://doi.org/10.1002/wsbm.1337>
47. Moffat JG, Vincent F, Lee JA et al (2017) Opportunities and challenges in phenotypic drug discovery: an industry perspective. *Nat Rev Drug Discov* 16(8):531–543. <https://doi.org/10.1038/nrd.2017.111>
48. Berger SI, Iyengar R (2009) Network analyses in systems pharmacology. *Bioinformatics* 25(19):2466–2472. <https://doi.org/10.1093/bioinformatics/btp465>
49. Hopkins AL (2008) Network pharmacology: the next paradigm in drug discovery. *Nat Chem Biol* 4(11):682–690. <https://doi.org/10.1038/nchembio.118>
50. Benson AR, Gleich DF, Leskovec J (2016) Higher-order organization of complex networks. *Science* 353(6295):163–166. <https://doi.org/10.1126/science.aad9029>
51. Yoon S, Lee D (2019) Meta-path based prioritization of functional drug actions with multi-level biological networks. *Sci Rep* 9(1):5469. <https://doi.org/10.1038/s41598-019-41814-w>
52. Montavon G, Samek W, Müller K-R (2018) Methods for interpreting and understanding deep neural networks. *Digital Signal Processing* 73:1–15. <https://doi.org/10.1016/j.dsp.2017.10.011>
53. Litjens G, Kooi T, Bejnordi BE et al (2017) A survey on deep learning in medical image analysis. *Med Image Anal* 42:60–88. <https://doi.org/10.1016/j.media.2017.07.005>
54. Mamoshina P, Vieira A, Putin E et al (2016) Applications of deep learning in biomedicine. *Mol Pharm* 13(5):1445–1454. <https://doi.org/10.1021/acs.molpharmaceut.5b00982>
55. Ramsundar B, Liu B, Wu Z et al (2017) Is multitask deep learning practical for pharma? *J Chem Inf Model* 57(8):2068–2076. <https://doi.org/10.1021/acs.jcim.7b00146>
56. Kalinin AA, Higgins GA, Reamaroon N et al (2018) Deep learning in pharmacogenomics: from gene regulation to patient stratification. *Pharmacogenomics* 19(7):629–650. <https://doi.org/10.2217/pgs-2018-0008>

57. Aliper A, Plis S, Artemov A et al (2016) Deep learning applications for predicting pharmacological properties of drugs and drug repurposing using transcriptomic data. *Mol Pharm* 13(7):2524–2530. <https://doi.org/10.1021/acs.molpharmaceut.6b00248>
58. Putin E, Mamoshina P, Aliper A et al (2016) Deep biomarkers of human aging: application of deep neural networks to biomarker development. *Aging (Albany NY)* 8(5):1021–1033. <https://doi.org/10.18632/aging.100968>
59. Donner Y, Kazmierczak S, Fortney K (2018) Drug repurposing using deep embeddings of gene expression profiles. *Mol Pharm* 15(10):4314–4325. <https://doi.org/10.1021/acs.molpharmaceut.8b00284>
60. Connectivity Map (CMap) (2019) Broad Institute. <http://www.broadinstitute.org/connectivity-map-cmap>. Accessed 1 Nov 2019
61. NIH LINCS Program (2019) National Institutes of Health (NIH). <http://www.lincsproject.org>. Accessed 1 Nov 2019
62. Hudson IL, Leemaqz SY, Shafi D et al (2017) Score function of violations and best cutpoint to identify druggable molecules and associated disease targets. In: Syme G, Hatton MacDonald D, Fulton B et al (eds) MODSIM2017, 22nd international congress on modelling and simulation. Modelling and Simulation Society of Australia and New Zealand, 2017. pp 487–393
63. Gao XW, Qian Y (2018) Prediction of multidrug-resistant TB from CT pulmonary images based on deep learning techniques. *Mol Pharm* 15(10):4326–4335. <https://doi.org/10.1021/acs.molpharmaceut.7b00875>
64. Li X, Xu Y, Lai L et al (2018) Prediction of human Cytochrome P450 inhibition using a multitask deep autoencoder neural network. *Mol Pharm* 15(10):4336–4345. <https://doi.org/10.1021/acs.molpharmaceut.8b00110>
65. Russo DP, Zorn KM, Clark AM et al (2018) Comparing multiple machine learning algorithms and metrics for estrogen receptor binding prediction. *Mol Pharm* 15(10):4361–4370. <https://doi.org/10.1021/acs.molpharmaceut.8b00546>
66. Hop P, Allgood B, Yu J (2018) Geometric deep learning autonomously learns chemical features that outperform those engineered by domain experts. *Mol Pharm* 15(10):4371–4377. <https://doi.org/10.1021/acs.molpharmaceut.7b01144>
67. Kuzminykh D, Polykovskiy D, Kadurin A et al (2018) 3D molecular representations based on the wave transform for convolutional neural networks. *Mol Pharm* 15(10):4378–4385. <https://doi.org/10.1021/acs.molpharmaceut.7b01134>
68. Kadurin A, Aliper A, Kazennov A et al (2017) The cornucopia of meaningful leads: applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget* 8(7):10883–10890. <https://doi.org/10.18632/oncotarget.14073>
69. Gomez-Bombarelli R, Wei JN, Duvenaud D et al (2018) Automatic chemical design using a data-driven continuous representation of molecules. *ACS Cent Sci* 4(2):268–276. <https://doi.org/10.1021/acscentsci.7b00572>
70. Harel S, Radinsky K (2018) Prototype-based compound discovery using deep generative models. *Mol Pharm* 15(10):4406–4416. <https://doi.org/10.1021/acs.molpharmaceut.8b00474>
71. Sanchez-Lengeling B, Aspuru-Guzik A (2018) Inverse molecular design using machine learning: generative models for matter engineering. *Science* 361(6400):360–365. <https://doi.org/10.1126/science.aat2663>
72. Grover A, Leskovec J (2016) node2vec. Paper presented at the proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining—KDD ’16
73. Perozzi B, Al-Rfou R, Skiena S (2014) DeepWalk. Paper presented at the proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining—KDD ’14
74. Tang J, Qu M, Wang M et al (2015) Line. Paper presented at the proceedings of the 24th international conference on World Wide Web—WWW ’15
75. Zitnik M, Zupan B (2016) Collective pairwise classification for multi-way analysis of disease and drug data. Paper presented at the biocomputing 2016. 18 Nov 2015
76. Luo Y, Zhao X, Zhou J et al (2017) A network integration approach for drug-target interaction prediction and computational drug repositioning from heterogeneous information. *Nat Commun* 8(1):573. <https://doi.org/10.1038/s41467-017-00680-8>
77. Wen M, Zhang Z, Niu S et al (2017) Deep-learning-based drug-target interaction prediction. *J Proteome Res* 16(4):1401–1409. <https://doi.org/10.1021/acs.jproteome.6b00618>
78. Lee I, Nam H (2018) Identification of drug-target interaction by a random walk with restart method on an interactome network. *BMC Bioinformatics* 19(Suppl 8):208. <https://doi.org/10.1186/s12859-018-2199-x>

79. Gao KY, Fokoue A, Luo H et al (2018) Interpretable drug target prediction using deep neural representation. Paper presented at the proceedings of the twenty-seventh international joint conference on artificial intelligence, 2018/07
80. Wan F, Hong L, Xiao A et al (2018) NeoDTI: neural integration of neighbor information from a heterogeneous network for discovering new drug–target interactions. *Bioinformatics* 35(1):104–111. <https://doi.org/10.1093/bioinformatics/bty543>
81. Ma T, Xiao C, Zhou J et al (2018) Drug similarity integration through attentive multi-view graph auto-encoders. Paper presented at the proceedings of the twenty-seventh international joint conference on artificial intelligence, July 2018.
82. Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks. arXiv. 1609.02907
83. Veličković P, Cucurull G, Casanova A et al (2018) Graph attention networks. arXiv 1710.10903v3
84. Zitnik M, Agrawal M, Leskovec J (2018) Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34 (13):i457–i466. <https://doi.org/10.1093/bioinformatics/bty294>
85. Coley CW, Barzilay R, Green WH et al (2017) Convolutional embedding of attributed molecular graphs for physical property prediction. *J Chem Inf Model* 57(8):1757–1772. <https://doi.org/10.1021/acs.jcim.6b00601>
86. Jin W, Coley C, Barzilay R et al (2017) Predicting organic reaction outcomes with weisfeiler-lehman network. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S (eds) Advances in neural information processing systems 30. Curran Associates, New York, NY, pp 2607–2616
87. Morris P, DaSilva Y, Clark E et al (2018) Convolutional neural networks for predicting molecular binding affinity to HIV-1 proteins. Paper presented at the proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics—BCB ’18
88. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: methods and applications. arXiv 1709.05584



# Chapter 8

## Building and Interpreting Artificial Neural Network Models for Biological Systems

T. Murlidharan Nair

### Abstract

Biology has become a data driven science largely due to the technological advances that have generated large volumes of data. To extract meaningful information from these data sets requires the use of sophisticated modeling approaches. Toward that, artificial neural network (ANN) based modeling is increasingly playing a very important role. The “black box” nature of ANNs acts as a barrier in providing biological interpretation of the model. Here, the basic steps toward building models for biological systems and interpreting them using calliper randomization approach to capture complex information are described.

**Key words** Artificial neural network, Calliper randomization, Interpreting black-box models

---

### 1 Introduction

Biological systems are complex entities involving a myriad of interactions that regulate biological processes in ways that we are only beginning to understand. There has been an acceleration in our understanding of biological systems, and this has largely been due to technological advancements and large-scale initiatives that have generated a plethora of valuable biological data.

Biological data may be broadly categorized as genomic, transcriptomic, and proteomic data (omic data). Genomic data refers to genomic DNA sequences obtained using next-generation sequencing methods. Transcriptomic data refers to data that quantifies transcripts obtained either by RNA sequencing or from microarrays. Proteomic data refers to the distribution of chemical moieties contained in proteins obtained by mass spectrometry. In addition, biological data may also be structural. This includes structures of biological macromolecules (DNA, RNA and proteins) obtained by X-ray crystallography, NMR, or cryo-electron microscopy. All omic data may be obtained in a high-throughput manner, while the throughput of structural data is comparatively lower, several

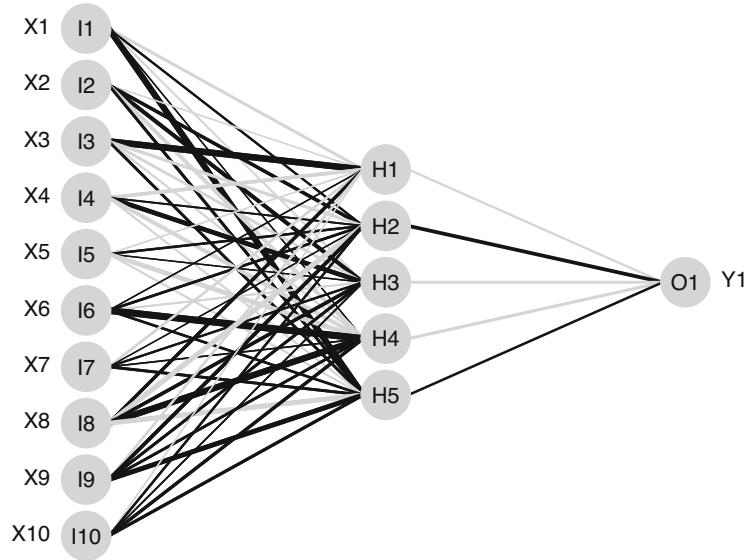
structural genomics projects have contributed to a significant increase in biological structural data.

When studying a particular biological system, the conventional approach has been to propose a hypothesis [1], and then verify the hypothesis using supportive data—often termed a hypothesis-first approach. In the era of high-throughput data, approaching problems by harvesting large-scale data has significant advantages over a conventional hypothesis-driven approach. This is because use of a data-driven approach makes it possible to detect things one was not expecting to *see* [2]. An outstanding example of this is the identification of the fusion of the BCR gene located on chromosome 22 with the ABL gene on chromosome 9, from high-throughput genomic data, even though it was known for decades that chronic myeloid leukemia patients had abnormalities associated with chromosomes 9 and 22 [3]. The precise identification of BCR-ABL gene fusion led to the understanding of the cause for the deregulated expression of the tyrosine kinase enzyme and to the development of imatinib that inhibits kinase enzyme [4–6]. This example clearly illustrates that, when studying biological systems, the quest has always been to identify the biological entity first and then understand what it does. Biological systems being complex, the task of identifying the entity responsible is never straightforward or simple. This is further complicated by the fact that biological molecules can act pleiotropically, and may simultaneously orchestrate changes in the behavior of a large cohort of responder molecules to create an altered phenotype. Theoretically, this behavior can be approximated as molecules exhibiting higher order correlations. Thus, analyzing high-throughput data to extract such information would help in generating computationally derived hypotheses. Artificial neural networks (ANNs) are a class of machine learning methods capable of deriving such information. This chapter describes the steps involved in building ANN models for biological systems and interpreting them using a technique called calliper randomization—a technique that helps identify features that are important in imparting knowledge to the ANN during training.

---

## 2 Methods

ANNs were initially modeled as mathematical approximations of the biological synapse and were meant to model the human brain [7]. ANNs, however, turned out to be very efficient in pattern recognition and have found more application as a pattern recognition machine than as a means of explaining how the brain functions. Building an ANN model for a system involves collecting relevant information about the system that can be used to generate a function that approximates its behavior. ANN models are built by



**Fig. 1** A three-layered neural network

presenting data associated with the system to be modeled to a network of computational units called artificial neural networks (Fig. 1).

The network learns the pattern associated with the system being modeled by iteratively updating the weight connections between computational units called neurons. ANNs have been exploited extensively in analyzing biological data. One of the main drawbacks of ANN models is that they are black-box models and do not reveal in a readily interpretable form any information about the system being modeled. However, it is possible to delineate the relative importance of features in imparting knowledge to the network using the calliper randomization approach that we will describe later in this chapter.

## 2.1 Modeling a Biological System Using a Neural Network

Modeling a biological system using an ANN begins by defining the system to be modeled and enumerating the variables that could explain the behavior of the system. Biological systems being complex entities; it is unlikely that we will be able to account for all the variables that describe such an entity, and hence only partial information is available to model it. For example, imagine the system being modeled involves building a model capable of distinguishing lung cancer types—adenocarcinoma (AC) and squamous cell carcinoma (SCC). While this problem can be approached from different angles, let us consider classifying them using their transcriptomes. This reduces the problem as the variables are thereby limited to the transcriptomes of the two types of cancers. Their transcriptomes now serve as input features that are experimentally derived. Although not stated explicitly, we are hypothesizing that there

exist features within the transcriptome that are capable of distinguishing AC from SCC. The transcriptomes are mapped to classes using an ANN that is indicative of the aforementioned types of cancers. The function/model that approximates this classification is the ANN model. An important first step to building an ANN model is to select a subset of features that is most relevant to be used as input to the neural net. Since transcriptomic data normally contains a very large number of transcripts ( $>50,000$ ), the process can be very computationally intensive.

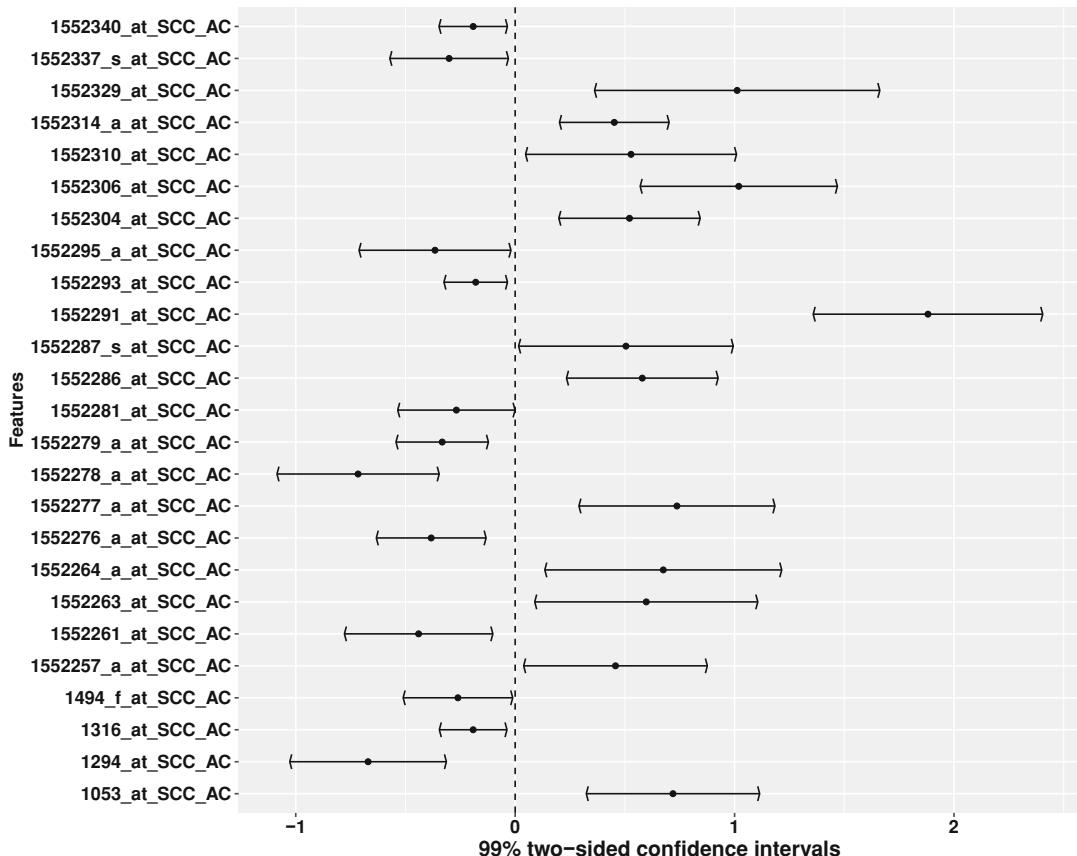
## 2.2 Feature Selection

The main goal of feature selection is to distinguish relevant features from irrelevant ones. If the value of a feature in a tumor sample is significantly different from the value of the same feature in a normal sample, then that feature is likely to be relevant. Selecting features from transcriptomes involves testing the differences in expression between many means. This can be conveniently achieved by using multiple comparisons [8]. Comparison of two means may be achieved using routine statistical testing methods like interval estimation or hypothesis testing. Comparisons of several means can be done using ANOVA based methods. A key drawback of this approach is that it does not identify which means were different; for this, we could use multiple comparison-based methods. The multcomp package provides methods to conveniently analyze data using multiple comparison-based methods. Multiple comparison of a subset of features that show significant difference between AC and SCC is shown in Fig. 2.

When several different treatments are involved, it is possible to score each significant comparison and rank features based on their scores [9]. The feature selection approach that we have mentioned here is only one of several such methods [10, 11]. It is noteworthy that the methods used in feature selection need not be too stringent as neural networks are capable of handling noisy data. If the feature selection process is too stringent there exists a possibility of filtering away features that may be associated with the system through second and higher order correlations. The choice of the feature selection method used depends on the system being modeled and should be carefully chosen [12].

## 2.3 Model Building

ANN models can be conveniently built in R using the Stuttgart Neural Network Simulator (RSNNS) [13]. Building a neural network model involves presenting the selected features to an artificial neural network for training. The number of neurons in the input and the output layer is determined by the system being modeled; however, both the number of hidden layers, and the number of neurons in each hidden layer that captures the nonlinearity of the system, are rather arbitrary and need to be optimized for the system under study. Training involves iteratively changing the weight connections between neurons in a manner that will optimally map the



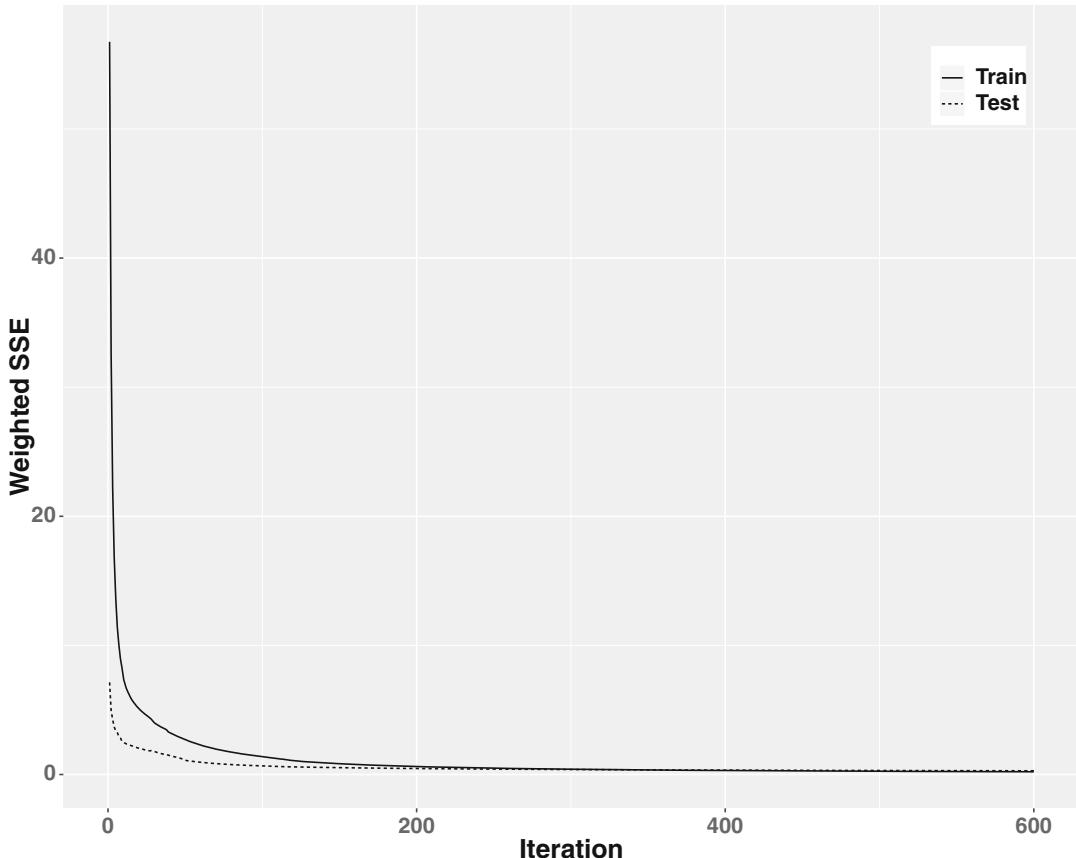
**Fig. 2** Multiple comparison of expression values for a subset of probes that show significant difference between SCC and AC. The ordinate represents probe IDs for SCC and AC that were compared

input onto the output. In the example being discussed, the input would be expression values of a subset of features that were deemed to be significant by the feature selection process, and the output would be 1 for AC and 0 for SCC.

It is important to mention that the data used for developing a neural network model need to be divided into training and test data sets. The weight connections are adjusted based on the training data only. The test data set is used to determine the performance of the model and is never used to adjust the weight connections. A well-trained generalized model should perform optimally not only on the training data set but also on the test data set (Fig. 3). If the network is overparametrized by using too many hidden layers and neurons, the resulting model will only memorize the training examples and perform poorly on the test data sets.

## 2.4 Evaluation of Neural Network Models

The evaluation of neural network models is done using the de facto performance measures of sensitivity and specificity. In the example under discussion, sensitivity measures the correctly classified



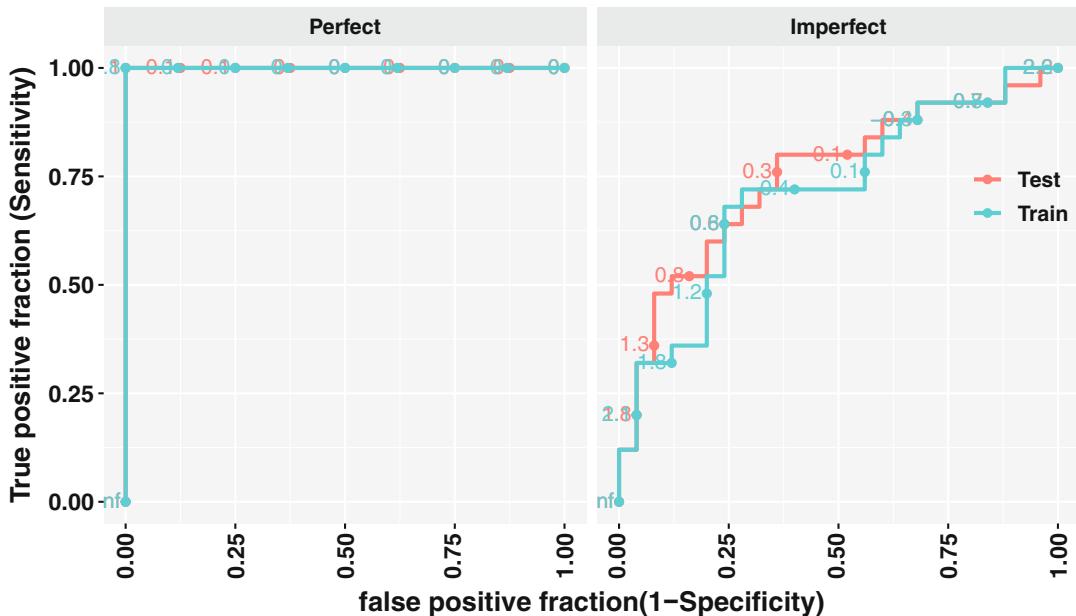
**Fig. 3** Error profile for the training and test data sets

samples, while specificity measures the proportion of AC classified as SCC and vice versa. It is now routine to depict this information graphically using a receiver operating characteristics (ROC) graph [14–16]. Briefly, ROCs are two-dimensional graphs in which the true positive (TP) rate (sensitivity or recall) is plotted on the ordinate or  $Y$  axis, and the false positive (FP) rate is plotted on the abscissa or  $X$  axis. These are defined as follows:

$$\text{TP rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (1)$$

$$\text{FP rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (2)$$

The ROC curve helps determine the performance of the neural network model and reveals the percent of samples within a particular class that are correctly classified (true positives or “hits”) as well as those that are incorrectly classified (false positives). In addition, the classification also generates true negatives (TN) and false negatives (FN) or “misses.” Since these are complements of the others, they can be ignored when constructing ROC curves. These curves



**Fig. 4** ROC curves depicting the performance of the ANN model for a perfect (left) and imperfect (right) classifier

are bow shaped. They rise from the lower left corner, where both percentages are zero, to the upper right corner, where both are one hundred, with a sharp bend for a perfect classifier (Fig. 4).

## 2.5 Calliper Randomization for Interpreting Neural Network Models

Neural network models are a “black box.” This is a potential serious limitation when applied to systems where it is necessary to interpret the model. Since neural networks are a parallel and distributed system, it is not possible to interpret the weights of the optimized model conveniently. Further, it is also possible to obtain two different models that have similar performance but whose weights may be completely different. An alternative to this is to perturb the input and evaluate the performance of the model. This approach was inspired by early experiments that were done to determine the principal component involved in carrying genetic information from the complex mixture of cell components [17, 18]. The approach helps provide insight into the system being modeled, assists in evaluating the relative importance of the features that are part of the input space, and derives the principal features among them. This approach, called calliper randomization, was first proposed by the author and applied to the analysis of biological sequences [19]. The approach has been extended to the analysis of microarrays [20], and can be conveniently applied to any learning model. The algorithm is briefly described below.

```

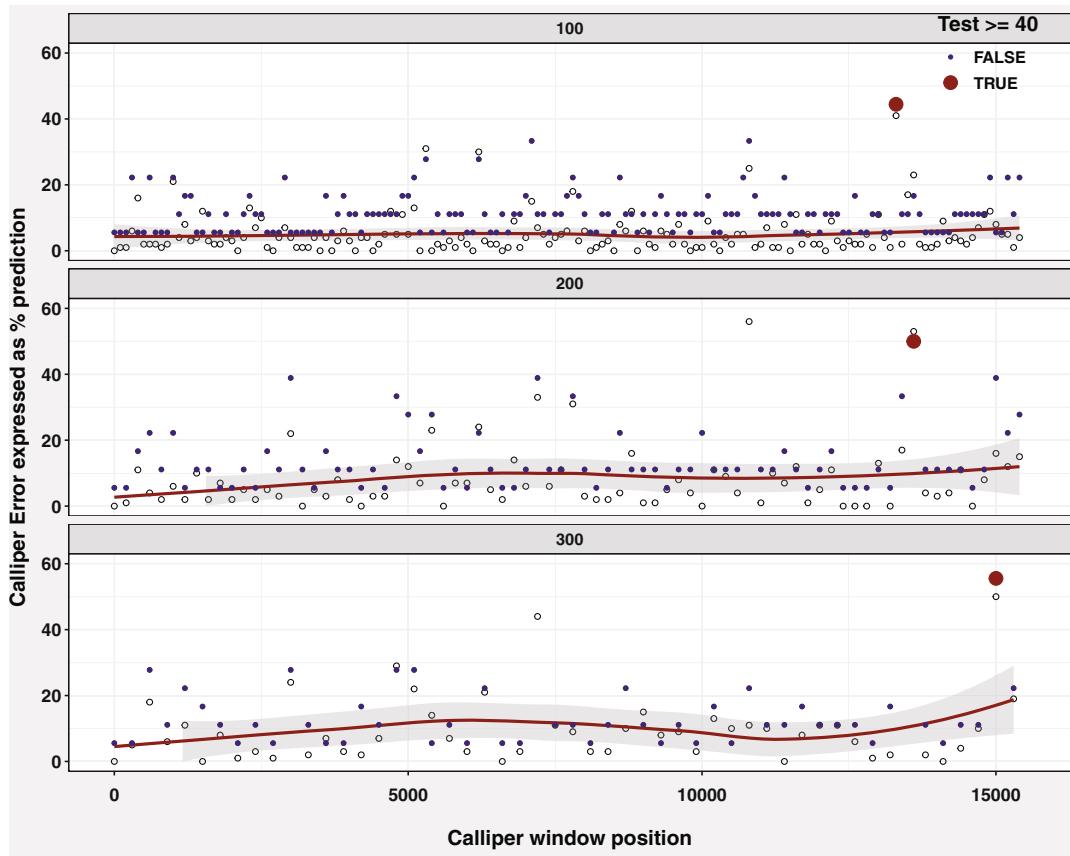
Algorithm 1 Calliper Randomization Algorithm
procedure CALLIPER RANDOMIZATION
    model  $\leftarrow$  ANNmodel
    calliperWindow  $\leftarrow$  required windowSize
    TestDataTmp  $\leftarrow$  TestData
    numFeatures  $\leftarrow$  number of features in one TestDataTmp
    For i  $\leftarrow$  1 to (numFeatures - calliperWindow) step 1
        predictTest  $\leftarrow$  predict(model, TestDataTmp)
        calliperError  $\leftarrow$  determineClassificationError(predictTest)
        TestDataTmp  $\leftarrow$  TestData
        For j  $\leftarrow$  i to (i + calliperWindow) step 1
            TestDataTmp[j]  $\leftarrow$  perturbed data
        end For
    end For i

```

The calliper plot of perturbed positions versus the calliperError provides a view of the features that are deemed important in imparting knowledge to the neural network. Those features, which, when perturbed, hamper the performance of the model, in this case by misclassifying the samples above a particular threshold (percentage or area under the curve (AUC) of the ROC curve), can be extracted for further functional analyses. The calliper plot for the example of AC and SCC is shown in Fig. 5.

Features which, when perturbed, contribute to missclassification the most are considered important in imparting knowledge to the network. In Fig. 5, the features that affect the prediction ability of the classifier by greater than 40% are depicted in larger dark circles. Since in this case a window of features is perturbed, all the features contained in the window are considered important and may be subject to further downstream analysis. Because the expression of genes is affected by other genes, it is possible to identify these by perturbing different subsets of features and evaluating the performance of the model.

Another important notion in using the calliper randomization is that of *directed callipers*—these are features that are known a priori to be important for proper functioning of the system under study. In such cases callipers may be placed at specific positions to include known or hypothesized features and those may be selectively perturbed. An example is the Shine–Dalgarno sequences and initiation codons which are known to be important for translation initiation, when building models to identify ribosome binding sites [21]. Selectively perturbing these would hamper the prediction capability of the model, indicating that these are key features of the ribosome binding site. This notion of directed callipers is not limited to sequence data and can be extended any type of data for which prior knowledge about features may be available.



**Fig. 5** Calliper error obtained using the neural network model for classifying AC and SCC

### 3 Summary

Understanding biological systems in terms of the individual components and their function is a complex endeavor, even when a plethora of data is available about the system under study. Using an ANN based modeling approach, it is possible to capture the behavior of the system. However, despite ANN models being able to capture the behavior of the system, they are “black-box” models and seem removed from being interpretable. This can be circumvented by using a biologically inspired perturbation method called calliper randomization. This method helps delineate the principal features from the complex data sets and may be key functional players biologically. These may be considered computationally derived hypotheses that can then be validated experimentally.

## Acknowledgments

I would like to thank IUSB for funding this work. This work is also supported partly by NSF award 1726218.

## References

1. Weinberg R (2010) Point: hypotheses first. *Nature* 464:678
2. Golub T (2010) Counterpoint: data first. *Nature* 464:679
3. Groffen J, Stephenson JR, Heisterkamp N et al (1984) Philadelphia chromosomal breakpoints are clustered within a limited region, bcr, on chromosome 22. *Cell* 36:93–99
4. Nowell PC (1962) The minute chromosome (Phl) in chronic granulocytic leukemia. *Blut* 8:65–66
5. Nowell PC (2007) Discovery of the Philadelphia chromosome: a personal perspective. *J Clin Invest* 117:2033–2035
6. Salesse S, Verfaillie CM (2002) BCR/ABL: from molecular mechanisms of leukemia induction to treatment of chronic myelogenous leukemia. *Oncogene* 21:8547–8559
7. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. *Nature* 323:533–536
8. Westfall PH (1997) Multiple testing of general contrasts using logical constraints and correlations. *J Am Stat Assoc* 92:299–306
9. Nair TM (2012) Analysis of isoform expression from splicing array using multiple comparisons. *Methods Mol Biol* 802:113–121
10. Urbanowicz RJ, Meeker M, La Cava W et al (2018) Relief-based feature selection: introduction and review. *J Biomed Inform* 85:189–203
11. Liang S, Ma A, Yang S et al (2018) A review of matched-pairs feature selection methods for gene expression data analysis. *Comput Struct Biotechnol J* 16:88–97
12. Liu H, Wong L (2003) Data mining tools for biological sequences. *J Bioinform Comput Biol* 1:139–167
13. Bergmeir C, Benítez JM (2012) Neural networks in R using the stuttgart neural network simulator: RSNNS. *J Stat Softw* 1(7)
14. Swets JA, Dawes RM, Monahan J (2000) Better decisions through science. *Sci Am* 283:82–87
15. Fawcett T (2006) An introduction to ROC analysis. *Pattern Recogn Lett* 27:861–874
16. Bewick V, Cheek L, Ball J (2004) Statistics review 13: receiver operating characteristic curves. *Crit Care* 8:508–512
17. Griffith F (1928) The significance of pneumococcal types. *J Hyg (Lond)* 27:113–159
18. Avery OT, Macleod CM, McCarty M (1944) Studies on the chemical nature of the substance inducing transformation of pneumococcal types : induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type iii. *J Exp Med* 79:137–158
19. Nair TM, Tambe SS, Kulkarni BD (1994) Application of artificial neural networks for prokaryotic transcription terminator prediction. *FEBS Lett* 346:273–277
20. Nair TM (2018) Statistical and artificial neural network-based analysis to understand complexity and heterogeneity in preeclampsia. *Comput Biol Chem* 75:222–230
21. Nair TM (1997) Calliper randomization: an artificial neural network based analysis of *E. coli* ribosome binding sites. *J Biomol Struct Dyn* 15:611–617



# Chapter 9

## A Novel Computational Approach for Biomarker Detection for Gene Expression-Based Computer-Aided Diagnostic Systems for Breast Cancer

Ali Al-Yousef and Sandhya Samarasinghe

### Abstract

Cancer produces complex cellular changes. Microarrays have become crucial to identifying genes involved in causing these changes; however, microarray data analysis is challenged by the high-dimensionality of data compared to the number of samples. This has contributed to inconsistent cancer biomarkers from various gene expression studies. Also, identification of crucial genes in cancer can be expedited through expression profiling of peripheral blood cells. We introduce a novel feature selection method for microarrays involving a two-step filtering process to select a minimum set of genes with greater consistency and relevance, and demonstrate that the selected gene set considerably enhances the diagnostic accuracy of cancer. The preliminary filtering (Bi-biological filter) involves building gene coexpression networks for cancer and healthy conditions using a topological overlap matrix (TOM) and finding cancer specific gene clusters using Spectral Clustering (SC). This is followed by a filtering step to extract a much-reduced set of crucial genes using best first search with support vector machine (BFS-SVM). Finally, artificial neural networks, SVM, and K-nearest neighbor classifiers are used to assess the predictive power of the selected genes as well as to select the most effective diagnostic system. The approach was applied to peripheral blood profiling for breast cancer where Bi-biological filter selected 415 biologically consistent genes, from which BFS-SVM extracted 13 highly cancer specific genes for breast cancer identification. ANN was the superior classifier with 93.2% classification accuracy, a 14% improvement over the study from which data were obtained for this study (Aaroe et al., *Breast Cancer Res* 12:R7, 2010).

**Key words** Breast cancer, Computer-aided diagnosis, Gene expression, Blood mRNA, Feature selection, Topological overlap, Spectral clustering, Data mining, Bioinformatics

---

### 1 Introduction

Cancer is a complex disease because it makes complex cellular changes. Microarrays have become a powerful approach to study cancer and identify changes produced within a cell. DNA microarrays observe the expression of thousands of genes in one sample

---

*Data:* Available from NCBI Gene Expression Omnibus: accession number GEO:GSE16443.

through gene expression profiling, which is important to capture a set of expressed genes that determines a cell phenotype.

Cell samples are required to design microarrays. There are two main sources of cell samples: tissues or blood cells. However, collecting tissue samples is not safe and such samples are not readily available, especially from healthy people [2]. By contrast, peripheral blood is readily available, easy to access and is a rich source of genetic information for studying human diseases. These properties make blood an attractive source for measuring gene expression [3]. Therefore, in this study, we used gene expression profiling of peripheral blood cells.

Recently, gene expression profiling of peripheral blood cells has been used for early detection of breast cancer [8]. However, microarray data analysis is challenged by the high-dimensionality of the data compared to the number of samples; therefore, selecting a small and consistent set of influential genes is crucial to effective classification. Most previous studies in this field have used either filters or wrappers to select a subset of genes that differentiate cancer from control cases [1, 4–7]. However, these methods have disadvantages; for example, wrapper methods take into account relations between features (genes) but are challenged by the dimensionality of the data, whereas filters ignore relations between features. For example, a Norwegian group analyzed the expression of 1368 genes extracted from peripheral blood cells of 56 women: 24 with breast cancer and 32 healthy, for early detection of breast cancer. They used a wrapper method (Nearest Shrinking Centroid method) for feature selection and correctly predicted 82% of the samples using 37 probes (29 genes) [8].

To confirm results of the above study, a larger sample of 11,217 genes extracted from 130 women has been analyzed [1]. The study used partial least square regression (PLSR) and jackknife testing [9] with dual leave-one-out cross-validation (a filter method) to reduce dimensionality of the data, by selecting the optimum number of latent variables for the classification. This is a set of variables selected by analyzing the covariance between the gene expression vectors and the class label. The regression also returns the regression coefficient for each gene; jackknife testing is used to select the variables that have a regression coefficient different from 0, with  $p$ -value  $<0.05$ . The study obtained a set of 738 probes that differentiated healthy from cancer samples with 79.5% prediction accuracy, 80.6% sensitivity, and 78.3% specificity. The study compared the 738 probes with the 29 genes obtained from the previous study [8] and found that 20 out of 29 genes were not significant in relation to the disease status in their study [1]. The expectation that a larger sample could increase prediction accuracy was not realized in this case, as it used filtering methods for feature selection that ignored biological relations between genes and selected genes based on the ability of individual genes to differentiate cancer from

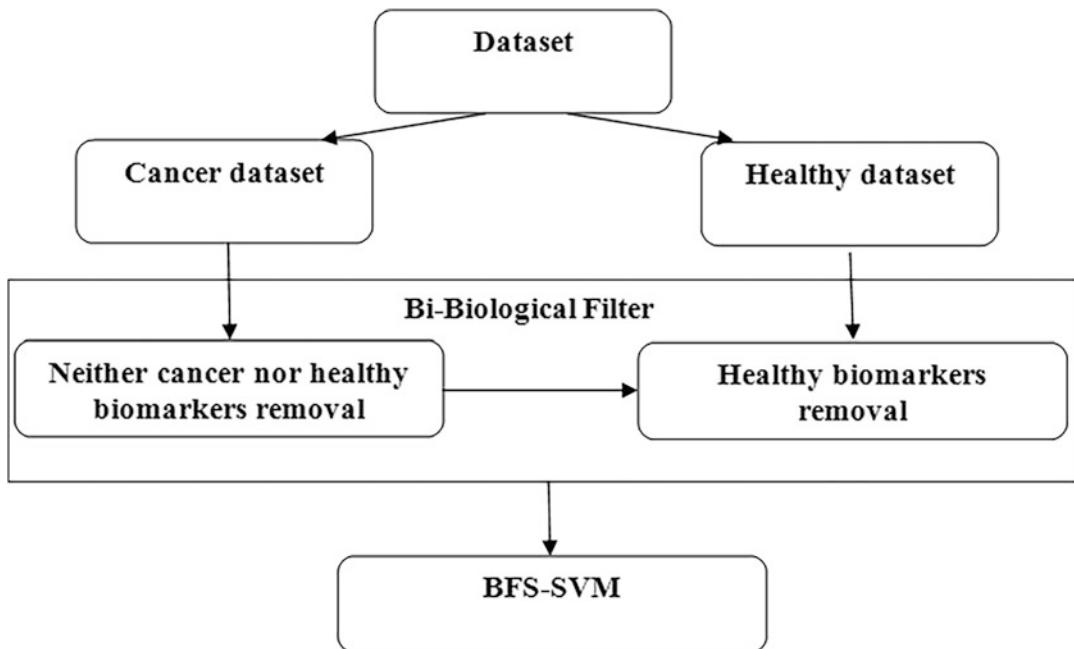
control cases. In addition, the method selected a large number of genes compared with the number of samples as inputs to the classifier, which can negatively affect the outcomes. Currently, accuracy of mRNA based CAD systems is about 79% [1]; this needs further enhancement to save lives of those with undetected cancer.

Gene coexpression provides key information to understand the functioning of living systems, in which the coexpressed genes are often involved in the same biological pathway [10]. Therefore, similarity between genes may reflect biological relations among them, where genes with high similarity may have similar biological function or be part of the same function [11, 12]. There are several ways to compute similarity between genes. The most common for gene expression data is Pearson correlation coefficient (PCC), which quantifies the linear relationship between a pair of genes (+1 indicating a perfect positive linear relationship and -1 a perfect negative linear relationship). However, PCC considers each pair of variables in isolation to other variables. From a biological perspective, relationships with other variables, genes or proteins, should be taken into account for consistent assessment of relevant genes implicated in cancer. A meaningful approach to increase diagnostic accuracy of mRNA-based CAD systems is to explore novel gene selection methods that consider biologically significant relations between genes to extract a minimal set of the most crucial genes and incorporate these to build a powerful CAD system. In this chapter, we introduce such a relations-based novel feature selection method to build a CAD system with much enhanced performance.

---

## 2 Materials and Methods

In this study we used 121 samples (67 malignant and 54 benign) collected by Ullevål University Hospital and Haukeland University Hospital in Norway from 2002–2004 (the second study mentioned in the Introduction). The malignant group contains 10 samples with ductal carcinoma *in situ* and 57 invasive carcinoma samples spread across a number of severity Grades from I to III. The 57 invasive carcinoma samples include 49 ductal, 4 lobular, and 4 other invasive types. The 54 benign cases include 12 benign cases and 42 normal cases with neither benign nor malignant findings. Each sample contained 11,217 genes (7351 known and 3866 unknown genes). Known genes are those that have gene symbols and IDs, whereas unknown genes are those currently lacking such identifiers. In this study, we only used the 7351 known genes for extracting cancer genes and building the CAD system. The data are already preprocessed, where the effect of the background has been removed, normalized, and summarized.



**Fig. 1** Flowchart for feature selection with Bi-Biological filter and best first search with SVM wrapper

In addition to high dimensionality, gene expression data can also contain noisy features which make the learning task very challenging. The process of removing noisy data (points that are irrelevant or redundant) or choosing a sub-set of relevant features from a given set of features is called feature selection [13, 14]. In this research, we propose a new two-step approach (BiBio-BFSS) to feature selection containing a preliminary Bi-biological filter (Bi-Bio) followed by a refined best first search (BFS) with SVM wrapper (BFSS): Bio-Bio filter obtains a large set of relevant genes and the BFSS wrapper selects a smaller set of the most influential genes (Fig. 1). The Bi-Bio filter itself contains two steps: neither cancer nor healthy biomarker elimination (genes expressed due to conditions other than cancer or health) followed by healthy biomarker removal. The output genes from Bi-Bio filter are used as inputs to BFSS wrapper to select a smaller set of genes for classification.

## 2.1 Bi-biological Filter: Preliminary Filtering

A biomarker of breast cancer should satisfy two conditions: firstly, it should be shared between any cancer sets. Secondly, it should be absent or insignificant in any healthy set. All previous studies have performed a direct comparison between a cancer set and a healthy set and selected biomarkers that are absent in the healthy set; this satisfies the second condition. But still there is a need to remove biomarkers that are not shared between cancer sets. These biomarkers may be related to another health issue—for example

“influenza”—or to noise in the gene expression data—for example noise that results from stress, anxiety, temperature, and so on.

The proposed Bi-biological filter is designed to satisfy both the above conditions by selecting a group of genes that are strongly related to breast cancer using gene coexpression networks [12]. The first step of the filter is designed to remove the biomarkers or groups of genes that are not shared between cancer cases (neither cancer nor healthy groups) (Subheading 2.1.1). The second step is to filter out healthy biomarkers from the selected biomarkers found in the first step (Subheading 2.1.2) using a similar approach.

### 2.1.1 Neither cancer nor healthy biomarker filtering

This is to remove the genes that are not shared between all cancer cases. For this, we randomly divided the malignant dataset into two sub-datasets M1 and M2; 33 samples assigned to M1 and 34 samples to M2. Then, the genes of each subset were divided into functional groups as follows:

1. Build coexpression network [12] as follows

- (a) Find the correlation matrix S (similarity matrix) using Pearson Correlation Coefficient (PCC) (Eq. 1). The output is an adjacency matrix of  $7351 \times 7351$ .

$$\text{PCC}_{ij} = \frac{n \sum_{u=1}^n x_{ju} x_{iu} - \sum_{u=1}^n x_{iu} \sum_{u=1}^n x_{ju}}{\sqrt{n \sum_{u=1}^n x_{iu}^2 - (\sum_{u=1}^n x_{iu})^2} \sqrt{n \sum_{u=1}^n x_{ju}^2 - (\sum_{u=1}^n x_{ju})^2}} \quad (1)$$

where  $n$  is the number of samples and  $x_{ju}$  is the expression value of gene  $j$  in sample  $u$ .

- (b) To highlight strong relations, dampen weak correlations and to convert the network to scale-free topology we power the absolute value of PCC by  $\beta$  (Eq. 2).

$$W_{ij} = |\text{PCC}_{ij}|^\beta \quad (2)$$

- 2. The above output matrix (W) represents the similarity between one pair of genes only. To involve other genes in the similarity consideration, we recompute similarity in terms of how many other genes are shared (topological overlap) between each gene pair as follows:

- (a) For each gene in the dataset, find the degree or the connectivity value  $c_i$  of the gene (Eq. 3).

$$c_i = \sum_{\substack{i=1 \\ i \neq j}}^N w_{ij} \quad (3)$$

where  $w_{ij}$  is the similarity value between genes  $i$  and  $j$  from Eq. 2 and  $N$  is the total number of genes (7351).

- (b) Now, for each pair of genes we find the number of shared genes. Assume the correlation between any pair of genes  $i$  and  $j$  is represented by  $w_{ij} \in \{0,1\}$ , where 0 indicates no correlation and 1 perfect correlation. To find the number of shared genes between a pair of genes, we compute the inner product of the two genes' correlation (similarity) vectors (Eq. 4) where  $I_{ij}$  is a number between 0 and  $N$  and  $w_{ki}$  is the similarity value of any gene  $k$  to gene  $i$ ; a high value of  $I_{ij}$  means that there is a high number of shared genes between genes  $i$  and  $j$  and a low value indicates no or few shared genes between them

$$I_{ij} = \sum_{\substack{k=1 \\ k \neq j, i}}^N w_{kj} w_{ki} \quad (4)$$

- (c) Now we use the topological overlap similarity between each gene pair (Eq. 5). The matrix containing all  $t_{ij}$  is called the topological overlap matrix (TOM) and is plotted in a weighted undirected graph where each gene is represented as a vertex and the edge between a pair of genes is weighted by the topological overlap similarity  $t_{ij}$  value.

$$t_{ij} = \frac{I_{ij} + w_{ij}}{\min(c_i, c_j) + 1 - w_{ij}} \quad (5)$$

#### Module Extraction

Clustering plays an important role in data analysis. In biology, especially with high dimensional data, clustering has been used to reduce the dimensionality of data by grouping the similar dimensions. The module is a group of genes working together to do a specific biological function. From the above definition, the genes that are strongly correlated to each other may belong to one pathway or do similar functions. To find the clusters of genes, this work uses Spectral Clustering. To apply spectral clustering on a TOM graph we execute the following steps as described by Ng, Jordan, and Weiss [15]:

1. Find the degree of each vertex on the graph (Eq. 6):

$$d_i = \sum_{j=1}^n t_{ij} \quad (6)$$

2. Next find the graph's normalized Laplacian matrix (l) (Eq. 7):

$$l_{ij} = \begin{cases} 1 & i = j \\ \frac{t_{ij}}{\sqrt{d_i d_j}} & i \neq j \end{cases} \quad (7)$$

#### Selection of Shared Clusters

The normalized spectral clustering is known to outperform the nonnormalized version in high dimensional data [15]. Therefore, we use normalized spectral clustering.

Now we perform spectral clustering as follows:

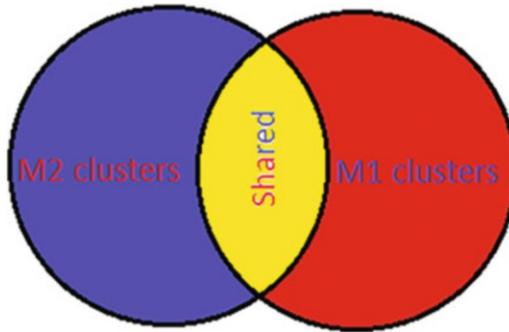
1. Find all eigenvalues and eigenvectors of the normalized Laplacian matrix.
2. The next step is to determine the number of sub-graphs or clusters in the gene co-expression graph. This is done by selecting K representative eigenvectors from the n eigenvectors of the graph, where K is much smaller than n, in order to reduce the dimensionality of the data and to select the number of clusters.
3. Let  $V \in R^{n \times k}$  be the matrix containing the selected eigenvectors  $v_1, v_2, v_3, \dots, v_k$  as columns. Form the matrix U by normalizing the row sums of V to 1 (Eq. 8). Now, genes of the original adjacency matrix are represented by a lower dimensional matrix  $U$ , where row  $i$  in  $U$  represents gene i in the original matrix:

$$u_{ij} = \frac{v_{ij}}{\left(\sum_{m=1}^k v_{im}^2\right)^{\frac{1}{2}}} \quad (8)$$

4. Gene clustering: for  $i = 1, \dots, n$ , let  $y_i \in R^k$  be the vector corresponding to  $i$ th row of U. Cluster the points  $y_i$  with k-means algorithm into clusters  $C_1, \dots, C_k$ .

By the end of the above step, the genes in M1 and M2 datasets have been divided into functional groups. In the current step, we select the shared groups of genes between M1 and M2 (Fig. 2).

A direct comparison between all genes in the groups is difficult; therefore, we select the hub genes for each cluster and the comparison takes place on hub gene level instead of all genes. To select the hub genes of a cluster, we find the within-cluster connectivity for each gene (Eq. 3) and select the m genes with the highest connectivity, a relatively small number, as hub genes. Selecting more than one hub gene allows partial similarity between a pair of clusters where clusters are considered shared, if one or more hub genes are shared between two clusters.



**Fig. 2** The shared clusters between M1 and M2 cancer subsets. Red color represents the clusters that were in M1 only, blue represents the clusters that were in M2 only, and yellow represents the shared clusters between M1 and M2

By the end of this step, neither cancer nor healthy clusters are removed and we keep only the shared groups that represent “cancer and healthy biomarkers.”

#### 2.1.2 Removal of Healthy Biomarkers

The next step is filtering out the healthy biomarkers from the selected biomarkers in the shared gene set and to keep only the probable cancer biomarkers by applying the following steps

1. Build the coexpression gene network for the healthy dataset and apply the spectral clustering to extract the healthy biomarkers as described above.
2. Extract the hub genes for the “healthy” clusters.
3. Find the shared clusters between the ‘healthy’ clusters found here and the clusters containing “cancer and healthy biomarkers” found previously.
4. Delete the shared clusters found in **step 3** from the “cancer and healthy biomarkers” set. This provides a set of clusters that are potential cancer biomarkers.

#### 2.2 Best First Search and SVM with Fivefold Cross-Validation Wrapper

In this step we use the forward BFS and the accuracy of SVM with K-fold cross-validation to find the subset of genes that is strongly related to breast cancer from the output genes of the bi-biological filter.

#### 2.3 Classification and Evaluation

We apply three supervised classifiers: Multilayer Feed Forward Neural Network (MFFNN) optimised/pruned by clustering correlated weighted hidden neuron outputs developed by [16, 17], Support Vector Machine (SVM) and Linear Discriminant Analysis (LDA). The results of each classifier are evaluated using False Positive rate (FP), False Negative rate (FN), Sensitivity, Specificity and Accuracy measures.

**Table 1**  
**The shared clusters between cancer1 and cancer2 datasets**

Shared clusters	Cancer 1 cluster ID	Cancer 2 cluster ID	Shared hub gene ID	Gene symbol
1	11	1	22,827	SIAHBP1
2	13	2	6139	RPL17
3	6	4	8786	RGS11
4	8	6	150,372	NFAM1
5	15	15	79,143	LENG4

For simplicity, each shared pair was given a new ID

### 3 Results and Discussion

#### 3.1 Feature Selection

Feature selection is divided into two main steps; Bi-biological filter and best first search supported SVM with fivefold cross-validation wrapper method.

##### 3.1.1 Neither Cancer nor Healthy Biomarker Filter

The cancer dataset is divided randomly into two subsets; cancer1 (M1) and cancer2 (M2). The cancer1 subset contains 33 samples and cancer2 contains 34 samples; both subsets contain 7351 mRNA genes. By applying spectral clustering on the TOM of cancer1 and TOM of cancer2 we found that the cancer1 dataset was divided into 17 clusters and cancer2 into 16 clusters. Then for each cluster we selected two hub genes and found five clusters shared between cancer1 and cancer2 datasets (Table 1). This removes all biomarkers which were not shared between cancer1 and cancer2 and only the shared ones are carried to the next step for filtering out healthy ones.

##### 3.1.2 Removal of Healthy Biomarkers

In this step, we analyzed the healthy dataset to find all healthy biomarkers. The analysis revealed that the genes in the healthy dataset were divided over 15 different clusters. Then, for each healthy cluster we extracted two representative hub genes. By comparing the healthy hub genes and the hub genes that were shared between cancer1 and cancer2 datasets (Table 1), we found that 2 out of 5 clusters (2 and 5 in Table 1) were found in the healthy dataset and considered to be healthy biomarkers. Thus, the remaining three clusters (1, 3, and 4 in Table 1) are considered to be the breast cancer biomarkers. Because we allowed partial similarity, the shared genes between pairs of clusters (cancer1-cancer2) were selected and considered to be breast cancer biomarkers. The total number of selected genes was 415.

Now, for genes in each cluster, we investigated the biological processes related to the genes of that cluster using The Database for Annotation, Visualization and Integrated Discovery (DAVID) [18] and selected the processes that had a False Discovery Rate (FDR) less than 20% and ( $p$  value  $< 0.05$ ).

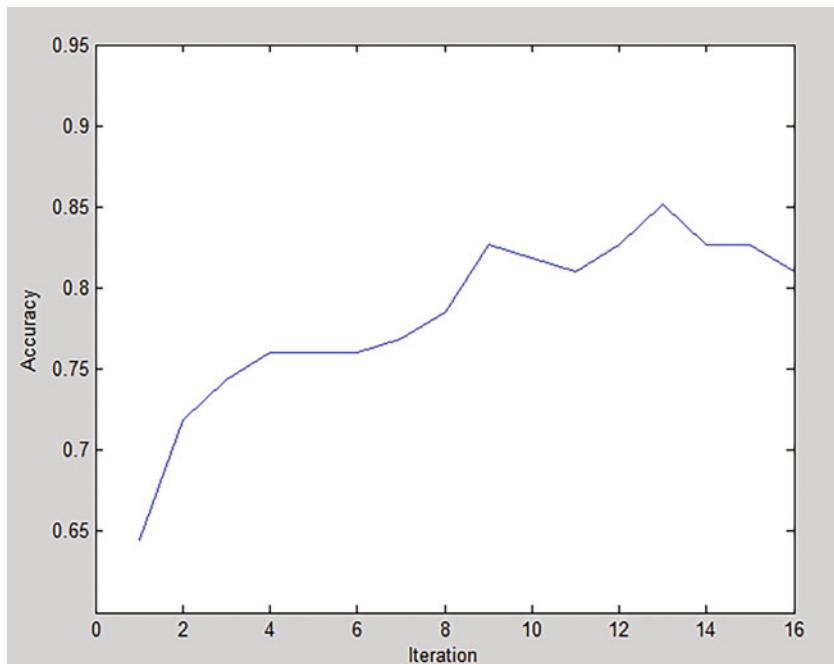
For the first cluster (1) we found apoptosis (GO:0006915) and regulation of apoptosis (GO:0042981) were the most important processes that differentiated normal and cancer cells. Several previous studies found apoptosis in relation to breast cancer where the apoptosis process in cancer patients was decreased compared to healthy ones [19–22]. For the second cluster (3) we found that, cellular respiration process (GO:0045333), which is the enzymatic release of energy from organic compounds that either requires oxygen (aerobic respiration) or does not (anaerobic respiration) (European Molecular Biology Laboratory, 2011) [23], and this was found in several studies in relation to breast cancer [24, 25]. Another process related to the genes of cluster (3), epithelial cell differentiation process (GO:0030855), the process in which a relatively unspecialized cell acquires specialized features of an epithelial cell [23], was found in relation to breast cancer in a study by Beitsch and Clifford [26]. In the last cluster (4), we found the processes in relation to producing energy necessary for cellular processes. Glucose is an important source of energy in the body and it is considered as fuel for a cell. Without energy, cells cannot perform their natural processes. Cancer cells are characterized by uncontrolled and rapid division so there is a need to provide cancer cells with a larger amount of glucose, thus speeding up the process of producing energy [27].

Genes in the same cluster are found to work together to carry out the same biological processes, which provides support for the biological relation between the genes. Furthermore, we find that the relationship between the biological processes of the groups and breast cancer are strong, which also provides support for the selected group of biomarkers.

### *3.1.3 Best First Search and SVM with Fivefold Cross-Validation Wrapper*

The Bi-biological filter selected 415 genes that were divided over three clusters; these clusters are potential breast cancer biomarkers. But we still needed to reduce the dimensionality of the data and select a subset of genes from the 415 genes for classification. To do this, we applied the BFS and SVM with five-fold cross-validation wrapper for gene selection as described previously. After 16 iterations, the algorithm was terminated because there were no improvements in iterations 14, 15, and 16 (cut off for number of failed iterations,  $m = 3$ ) (Fig. 3).

The highest accuracy obtained by the wrapper was at iteration number 13 with 85.1% classification accuracy; 88.05% sensitivity and 81.4% specificity using 13 genes (DAPK3, CTDSP1, CXX1,



**Fig. 3** The output accuracy from the best first search and SVM with fivefold cross-validation wrapper for 16 iterations. The X-axis represents the iteration number and the Y-axis represents the corresponding accuracy

RCOR3, MYL4, YWH, ABGMEB2, GPR78, ILK, HSPC171, ACAT2, PRKRIP1, and PP3856). Also, we found DAPK3 (Death-Associated Protein Kinase 3) obtained the highest individual classification accuracy (iteration 1).

### 3.2 Classifications

The 13 selected genes were used as input for the three classifiers, multilayer feed forward neural network (MFFNN) optimized with hidden network pruning by a novel method with fivefold cross-validation, SVM with leave-one-out cross-validation (LOOCV) and linear discriminant analysis (LDA) with LOOCV. Then the output results of each classifier were evaluated using the sensitivity, specificity, FP, FN, and accuracy measures (Table 2). From the outputs of different classifiers we found the MFFNN was the superior classifier with 94.02% sensitivity, 92.6% specificity and 93.4% accuracy.

Now, reviewing the FP and FN cases of the best classifier, MFFNN, we found 2 out of 4 FP cases had a cyst or benign tumour which means that the presence of benign findings in the breast may reduce the specificity of our system. The grades of FN cases were: one case of Grade I, two cases of Grade II and one case not classified. From these FN cases we found that our system correctly classified 92.8% of Grade I cases and 90.9% of Grade II cases and

**Table 2**  
**The performance of different classifiers using the 13 selected genes**

Classifier	SN	SP	FN rate	FP rate	Accuracy
SVM	86.6%	81.5%	16.98%	14.7%	84.3%
LDA	82.1%	79.6%	21.8%	16.7%	80.9%
MFFNN	94.02%	92.6%	7.4%	5.97%	93.4%

SN sensitivity, SP specificity, Ac accuracy

the local sensitivity values of both grades are close to the overall sensitivity of the system. This means that our system successfully predicts breast cancer in early stages.

Literature reviews suggest that there is only one blood-based mRNA CAD for early detection of breast cancer (BC-CAD), developed by Aaroe and colleagues [1]. This CAD system extracted 738 mRNA probes as breast cancer biomarkers using a filtering method. These biomarkers were used to classify the samples into healthy and cancer cases. The accuracy, sensitivity and specificity value for that CAD system [1] were 79.5%, 80.6%, 78.3%, respectively. By comparing our CAD system results with their system, we found that the accuracy of all classifiers in our CAD systems is better. Specifically, the diagnostic accuracy of LDA classifier was enhanced by 2% and the SVM improved the diagnostic accuracy by 5.5%. The significant improvement of our BC-CAD over Aaroe et al. [1] system was in the accuracy, sensitivity and specificity of the MFFNN with 93.4%, 94.02% and 92.6%, respectively, which means that about 14% of cancer cases misdiagnosed by Aaroe and co-workers are diagnosed correctly by our system and hence more lives could be saved. Furthermore, using 738 probes for classification reduces the performance of classifiers due to high dimensionality of data compared with 13 genes in our study, 56 times less than in their study.

#### 4 Conclusions

We introduced here a new method for breast cancer biomarker selection: Bi-Biological filter and Best first Search (BFS) supported SVM with K-fold cross-validation; we successfully identified 13 genes as breast cancer biomarkers in the blood. Also, we evaluated the diagnostic accuracy of three classifiers MLFFNN, LDA, and SVM. The best results were obtained using MFFNN with 93.4% classification. This is a significant improvement over the previous CAD system indicating that about 14% of cancer cases misdiagnosed in the previous CAD system [1] were diagnosed

correctly in our system and hence more lives could be saved. In future, a larger dataset will be used for validation of the performance of our method.

## Acknowledgements

This work was supported by a Scholarship from Jerash University in Jordan and support from Lincoln University, New Zealand.

## References

1. Aaroe J, Lindahl T, Dumeaux V et al (2010) Gene expression profiling of peripheral blood cells for early detection of breast cancer. *Breast Cancer Res* 12(1):R7
2. Marteau J-B, Mohr S, Pfister M et al (2005) Collection and storage of human blood cells for mRNA expression profiling: a 15-month stability study. *Clin Chem* 51(7):1250–1252
3. Fang X, Evans K, Willis RC et al (2006) High-throughput sample preparation from whole blood for gene expression analysis. *J Assoc Lab Automat* 11(6):381–386. <https://doi.org/10.1016/j.jala.2006.10.001>
4. Fan X, Shi L, Fang H et al (2010) DNA microarrays are predictive of cancer prognosis: a re-evaluation. *Clin Cancer Res* 16(2):629–636
5. Kretschmer C, Sterner-Kock A, Siedentopf F et al (2011) Identification of early molecular markers for breast cancer. *Mol Cancer* 10(1):15
6. Ma S, Kosorok MR, Huang J et al (2011) Incorporating higher-order representative features improves prediction in network-based cancer prognosis analysis. *BMC Med Genet* 4:5
7. Schrauder MG, Strick R, Schulz-Wendtland R et al (2012) Circulating micro-RNAs as potential blood-based markers for early stage breast cancer detection. *PLoS One* 7(1):E29770
8. Sharma P, Sahni NS, Tibshirani R et al (2005) Early detection of breast cancer based on gene-expression patterns in peripheral blood cells. *Breast Cancer Res* 7(5):R634–R644
9. Wu CFJ (1986) Jackknife, bootstrap and other resampling methods in regression analysis. *Ann Stat* 14(4):1261–1295
10. Obayashi T, Hayashi S, Shibaoka M et al (2008) coxpresdb: a database of coexpressed gene networks in mammals. *Nucleic Acids Res* 36(suppl 1):D77–D82
11. Yip A, Horvath S (2007) Gene network interconnectedness and the generalized topological overlap measure. *BMC Bioinformatics* 8(1):22
12. Zhang B, Horvath S (2005) A general framework for weighted gene co-expression network analysis. *Stat Appl Genet Mol Biol*
13. Blum AL, Langley P (1997) Selection of relevant features and examples in machine learning. *Artif Intell* 97(1–2):245–271
14. Gilad-Bachrach R, Navot A, Tishby N (2004) Margin based feature selection—theory and algorithms. In Proceedings of the twenty-first international conference on machine learning. ACM, Banff
15. Ng AY, Jordan MI, Weiss Y (2001) On spectral clustering: analysis and an algorithm. *Adv Neural Inf Proces Syst* 14
16. Samarasinghe S (2010) Neural networks for water system analysis: from fundamentals to complex pattern recognition. In Hydrocomplexity: new tools for solving wicked water problems, international association of hydrological science, Paris. pp 209–213
17. Al-Yousef A, Samarasinghe S (2011) Ultrasound based computer aided diagnosis of breast cancer: evaluation of a new feature of mass central regularity degree
18. Dennis G, Sherman BT, Hosack DA et al (2003) DAVID: database for annotation, visualization, and integrated discovery. *Genome Biol* 4(5):P3
19. Haldar S, Negrini M, Monne M et al (1994) Down-regulation of bcl-2 by p53 in breast cancer cells. *Cancer Res* 54(8):2095–2097
20. Parton M, Dowsett M, Smith I (2001) Studies of apoptosis in breast cancer. *BMJ* 322(7301):1528–1532
21. Feng Z, Marti A, Jahn B et al (1995) Glucocorticoid and progesterone inhibit involution and programmed cell death in the mouse mammary gland. *J Cell Biol* 131(4):1095–1103
22. Graham JD, Clarke CL (1997) Physiological action of progesterone in target tissues. *Endocr Rev* 18(4):502–519

23. European Molecular Biology Laboratory, EMBL-EBI (2011) European Bioinformatics Institute
24. Simonnet H, Alazard N, Pfeiffer K et al (2002) Low mitochondrial respiratory chain content correlates with tumor aggressiveness in renal cell carcinoma. *Carcinogenesis* 23(5):759–768
25. Warburg O (1956) On the origin of cancer cells. *Science* 123(3191):309–314
26. Beitsch PD, Clifford E (2000) Detection of carcinoma cells in the blood of breast cancer patients. *Am J Surg* 180(6):446–449
27. Annibaldi A, Widmann C (2010) Glucose metabolism in cancer cells. *Curr Opin Clin Nutr Metab Care* 13(4):466–470. <https://doi.org/10.1097/MCO.0b013e32833a5577>



# Chapter 10

## Applying Machine Learning for Integration of Multi-Modal Genomics Data and Imaging Data to Quantify Heterogeneity in Tumour Tissues

Xiao Tan, Andrew T. Su, Hamideh Hajiabadi, Minh Tran,  
and Quan Nguyen

### Abstract

With rapid advances in experimental instruments and protocols, imaging and sequencing data are being generated at an unprecedented rate contributing significantly to the current and coming big biomedical data. Meanwhile, unprecedented advances in computational infrastructure and analysis algorithms are realizing image-based digital diagnosis not only in radiology and cardiology but also oncology and other diseases. Machine learning methods, especially deep learning techniques, are already and broadly implemented in diverse technological and industrial sectors, but their applications in healthcare are just starting. Uniquely in biomedical research, a vast potential exists to integrate genomics data with histopathological imaging data. The integration has the potential to extend the pathologist's limits and boundaries, which may create breakthroughs in diagnosis, treatment, and monitoring at molecular and tissue levels. Moreover, the applications of genomics data are realizing the potential for personalized medicine, making diagnosis, treatment, monitoring, and prognosis more accurate. In this chapter, we discuss machine learning methods readily available for digital pathology applications, new prospects of integrating spatial genomics data on tissues with tissue morphology, and frontier approaches to combining genomics data with pathological imaging data. We present perspectives on how artificial intelligence can be synergized with molecular genomics and imaging to make breakthroughs in biomedical and translational research for computer-aided applications.

**Key words** Machine learning, Cancer, Spatial transcriptomics, Genomics, Gene expression, Histopathological images, Data integration

---

### 1 Introduction

Advances in sequencing and molecular imaging in biomedical research have generated big data for tissues and cells from patients' biopsies. Traditionally, imaging for cancer diagnosis is done by H&E staining, followed by pathologist's assessment and possible recommendations for follow-up immunostaining of tissue biopsies with specific cancer markers. However, the handful number of

markers that can be stained at one time limits the ability to quantify cancer heterogeneity. Of note, tumour heterogeneity is a significant cause of low treatment efficacy, as heterogeneity in drug susceptibility contributes to high recurrence rates, despite the apparently successful initial treatment. Current clinical practice to assess cancer heterogeneity is mainly based on visual examination of tissue morphology from H&E staining images and binary classification of presence or absence of cancer markers from immunostaining tissue images. Although tissue heterogeneity can be assessed at the macroscopic level by utilizing histopathological images of cancer tissue resections, which are collected for virtually all cancer patients, much more genetics–genomics molecular and tissue morphological information can be generated and utilized from these samples to better characterize cancer tissues. To integrate genomics (microscopic) and imaging (macroscopic) data, innovative approaches using deep neural network framework and fast parallel computation to train multiple layers of features in image and genomics data to better understand host-tumour interactions at the tissue level.

Current molecular assays to assess tumour heterogeneity are limited by the need to dissociate cells from native tissues, thus discarding the spatial and physiological context, and the lack of resolution to measure at the single-cell level. These two limitations are critical challenges to understanding immune cell and cancer cell subpopulations and their molecular interactions in tumours. Tumour heterogeneity derives from diverse cell type composition, different spatial arrangement of microenvironment, and the dynamic changes in cell-cell communications. In tumour ecosystems, multiple cell types communicate by ligand–receptor interactions [1, 2]. The successful application to treatment of immune checkpoint inhibitors proves that targeting ligand–receptor interactions can provide significant benefits for patients [3]. In colorectal cancer, immune cell types, density and location were shown to be more predictive of clinical outcomes than classical TNM (Tumour, lymph Nodes, and Metastasis) staging [4]. In squamous cell carcinoma (SCC), the high level of tumour heterogeneity calls for the identification of a panel of biomarkers to address individual treatment needs, and predictors of individual tumour capacity for invasive spread, rather than generic biomarkers [5]. However, our knowledge of cell–cell interactions, cell types, microenvironment, biomarkers and how they affect outcome is still limited.

New technologies such as spatial transcriptomics sequencing and FISH single-molecule imaging generate multi-modal data-types, which are major challenges for traditional machine learning approaches. Therefore, potential new class of spatial cancer biomarkers that are specific to cell types and regulators of tumour progression through targeting tumour–immune cell interactions can be revealed. Spatial transcriptomics and proteomics technologies add a spatial physiological context to expression data. Transcriptomics

and proteomics have been proven to be valuable for cancer precision medicine. For example, a recent atlas-level study shows shorter patient survival associated with up-regulation of genes involved in cell growth and with down-regulation of genes involved in cellular differentiation [6]. However, typical sequencing methods require cell dissociation, thus losing spatial information, which is the positional context of gene/protein expression of cells in a tissue. To bridge the gap between cellular and spatial information in studies of organs, spatial technologies have recently been developed, including spatial transcriptomics. A cutting-edge spatial gene expression profiling technology, Slide-seq, involves cryo-sectioning, H&E staining, light microscopy, tissue permeabilization, cDNA synthesis, tissue digestion, probe cleavage, Illumina RNA-Seq, and deconvolution of sequence reads based on spatial barcodes [7]. Slide-seq measures cellular gene products in intact tissues without a tissue dissociation step, thereby capturing native, *in vivo* physiological conditions. Slide-seq uses sequencing with spatial barcodes allowing to measure over 16,000 genes.

Together with the development of new experimental platforms, machine learning and deep learning methods as well as computational hardware development have been blooming recently, which create a solid ground for implementing advanced analysis strategies for big biomedical data. For example, infrastructure and solutions for data storage, transfer, and access have been rapidly improved, especially through cloud-based services. Meanwhile, deep learning methods for computer vision and multi-modal data integration are increasingly implemented both at research and production levels.

These developments are main factors that for utilizing artificial intelligence in cancer tissue diagnosis.

---

## 2 Background

To extract valuable information from both omics data and image data, deep learning as an advanced machine learning technique has heuristic performance and prospects due to its capacity of feature extraction [8]. The concept of deep learning was first to come up around the 2000s, which is artificial intelligence simulating the human brain and neural cell activity [9]. Although deep learning is an emerging subfield of machine learning, it has successful applications in voice and signal processing, sequence and text prediction, and in computational biology [10]. Several developing models in deep learning are artificial neural network (NN), convolutional neural network (CNN), transfer learning, and hierarchical learning. Below we discuss the overview of these methods and their applications in genomics.

## **2.1 Neural Network for Genomics Data Analysis**

The NN is artificial modules which can process information. Neurons in each adjacent layer can connect to each other, while in the same layer, there is no connection between neurons. The connection between input data and hidden neurons in sequential layers is based on weight and bias combined with an activation function. Deep neural network (DNN) trains data through multilayers, where outputs from one layer are inputs (features—neurons) in the next layer as high-level representation features [11]. These multilayer features are optimized with thousands of parameters for reducing error in classification, segmentation, and recognition tasks [12, 13].

For a gene expression matrix as an input, genes can be fed into sequential hidden layers step by step, and the informative latent features can be extracted and refined. Activation is a function to quantify the value passed to neuron and change the state of a neuron to “activated” or “inactivated”. Common activation functions include rectified linear unit (ReLU), tanh, and logistic functions like Sigmoid and Softmax. In the training step, loss Cross-Entropy is to measure the optimization performance by comparing estimated probability distribution and observed distribution. The neurons in output layers act as predictors where each response variable with a value from 0 to 1 passed from the previous layer indicates the probability for a specific type of cancer. The final output of cancer type is from the predictor which has the highest value. In addition, Dropout can be introduced as a regularization strategy to reduce overfitting which means model contains more parameters than can be justified by the data. In this case, some neurons will be isolated in training epoch.

Recently, more researchers focused on applying the NN model to genomics research. A DNN framework for the prognosis prediction of breast cancer by integrating Multi-dimensional Data including gene expression, copy number, and clinical data was recently reported [14]. The authors developed three different DNN for each type of data and integrated the results by score fusing to achieve an accuracy of 82.6% [14]. Another NN application called Cox-nnet was developed for prognosis prediction of survival based on high throughput transcriptomics data from TCGA [15]. Cox-nnet used a two-layer NN which contains one hidden layer and one Cox regression based output layer [15]. The authors demonstrated that Cox-nnet had the same or better predictive performance than traditional methods like linear regression and RF based on 10 TCGA RNA-Seq data [15].

## **2.2 Convolutional Neural Network (CNN) for Pathological Image Analysis**

CNN can extract feature from multiple arrays format dataset such as image [8]. The principle of CNN algorithm is to reduce the parameters and keep learning capacity [16]. In CNN, convolution kernel’s parameters are optimised by a backpropagation algorithm. CNN consists of linear convolution operation, followed by non-linear activators, pooling layers, and deep neural network

(DNN) classifier (Fig. 1). The convolutional layer is the core of CNN. The neuron in the convolutional layer can learn a certain feature at a specific spatial position from the input. Pooling layer or downsampling layer is a clustering method that reduce the dimensions of data by combining neuron clusters to one neuron. In CNN, patterns are learned hierarchically. Smaller patterns will be initially learned, and intricate patterns will be sequentially assembled by smaller patterns. In cancer imaging data, a single cancer cell can be identified as smaller patterns and pathological tissue can be identified as intricate patterns. Therefore, the inherent characteristic leads to CNN becomes a suitable model for cancer image analysis [17].

Recently, applications of CNN were reported in many biological and clinical fields, such as classification of medical images [18], nuclei detection [17], glomeruli localization [19], breast cancer diagnosis [20, 21], colon tumour [22], and glioma grading in brain tumours [23]. A research group developed a CNN framework for classification of lung patterns on computerized tomography (CT) scans and achieved an F-score of 85.5% [24]. Another application of CNN (inception v3) was used to classify cancer types LUAD (Lung adenocarcinoma), LUSC (Lung squamous cell carcinoma), and healthy lung tissue based on H&E image and achieved 0.97 AUC, Area Under the Curve [25]. In Esteva et al. [26] study of skin cancer diagnosis, they utilized pre-trained inception v3 CNN model to train their dermatologist-labelled dataset of 129,450 clinical images for classification and achieved performance on par with 21 board-certified dermatologists [26]. The development of automated cancer classification systems has been dependent on pathologist annotated tissue images; however, these qualitative annotations are variable between pathologists, time-consuming and often not at the pixel-level. Whereas these annotations rely on prior knowledge to link diseases with tissue morphological features, the combination of molecular information with tissue images is currently a major focus, as exemplified by efforts from TCGA and TCIA.

### **2.3 Deep Learning Methods for Combining Genomics and Imaging Data**

#### **2.3.1 Transfer Learning**

Transfer learning is a strategy frequently utilized in the analysis that two tasks are very similar [27–29]. Transfer learning enables a previously trained model to transfer its optimized parameters to a new model; therefore, it can reduce the training time and improve the learning efficiency in the initial phase. In particular, transfer learning can overcome the problem of imbalanced data by using the optimized parameters transferred from the balanced data trained model [28]. In addition, using transfer learning to extract features from both genomics data and imaging data for subsequent training in a new DNN model provides a new direction of integration of multi-modal datatypes.

One transfer learning approach is to use a network architecture achieving good results in natural images. Because of the absence of large annotated histopathological images, some researches have employed extensively annotated, publicly available images from computer vision tasks such as ImageNet to pre-train their network. This approach has been used by many pieces of research [31, 32]. For example, Nagpal et al. [31] use an architecture achieving significant results on ImageNet as pre-training. A U-net structure proposed by the study conducted by Isaksson et al. [32] uses semantic segmentation for prostate tissue. A recent study aiming to detect prostate cancer uses a publicly available breast histopathology dataset as pre-training. This study experimentally proves that this outperforms transfer learning from ImageNet dataset [33].

The other possible transfer learning method is to use a pre-trained model to extract features and then the extracted features will be sent to a classifier. These features are generated by extracting middle representations from the intermediate layers of a pre-trained neural network architecture. These techniques have been used by some researchers [34, 35]. The most popular way to apply transfer learning methods is to use a pre-trained model and then make them fine-tuned using the given data. There are various fine-tuning approaches that have been used such as fine-tuning all the layers, freezing the convolutional layers, fine-tuning that last few layers [36–38].

### 2.3.2 Autoencoder

As both genomics data and imaging are high dimensional, when integrating multi-modal datatypes for a DNN model, high dimensions of the dataset will be problematic in many aspects like efficiency on feature learning and consumption of learning time [39]. AE as an unsupervised neural model has the ability to extract or represent features from a high-dimensional dataset and performs as a dimension reduction tool [40]. There are two opposite steps in an AE framework. Step one is compressing and encoding a high-dimensional input into a low dimensional code. The next step is decoding a low dimensional code to reconstruct a high-dimensional output. Often, the output cannot fit the input precisely, and the variation can be measured with a loss function, such as mean squared error (MSE) of input and output. Compared to Principal Component Analysis (PCA), AE is more robust and effective in extracting data features for its non-linear transformation in hidden layers [28]. To avoid identical learning and improve the ability to capture important feature, two variations, denoising auto-encoder (DAE) and variational autoencoder (VAE), are widely used in image analysis. DAE takes input which is partially corrupted for training to get the original undistorted input [41]. VAE takes into account the distribution of latent variables and calculates an additional lost component in the training step [42]. AE model,

therefore, captures most important features and remove noise from a dataset.

AE is found in many deep learning applications to improve model performance. For imbalanced classification, synthetic minority over-sampling technique (SMOTE) was used to overcome the problem of imbalanced datasets and Stacked Denoising Autoencoder (SDAE) was then used to reduce data redundancy and noise generated by SMOTE [41]. AE can also play a role as a clustering method to reduce the dimensions of single-cell RNA-Seq data [43]. Comparing feature extraction performance, SDAE showed higher accuracy (98.26%) than other methods like PCA (96.52%) and KPCA (97.39%) when trained on a gene expression dataset of breast cancer [44]. To solve the high dimension problem of integration of multi-omics data, AE was also used to extract features from mRNA, miRNA, and methylation dataset [40]. After data integration, extracted features were used as the input in a Gaussian mixture clustering method to predict survival subtypes of bladder cancer [40].

### **3 Applying Deep Learning for Mining Biomedical Imaging and Molecular Genomics Data**

#### ***3.1 Interpretable Deep Learning in Medical Research***

Since the introduction of Convolutional Neural Networks (CNNs) and other complex deep architectures, many practical tasks including image classification, object detection, and image segmentation have shown unprecedented performance in term of accuracy. Unfortunately, the more complex the model is the harder to fully understand it is. Subsequently, deep learning (DL) is often considered as the black-box model and is usually questioned about its reliability and applicability for clinical trial [45]. The use of DL models in real medical practise is indeed too risky and the need for interpretation techniques and explainable DL architecture are becoming a more popular.

Interpretability is especially important in medicines application as it covers not only *scientific understanding, safety* but also *mismatched objectives* [46]. Despite the potential of current applications of AI in many different fields of medicine, there are still controversies around the performance and interpretable guarantee of DL methods. On the one hand, when dealing with clinical decision-making, it is always very important for clinicians to provide the comprehensive explanation of the current patient's situation. DL applications for genomics data are, no exception, also needed to be understandable to clinicians and patients for safety and potential mismatch validation. On the other hand, the advantage of DL over vanilla ML is that it can automatically fine-tune its features extractor in larger searching space. Therefore, it can

potentially provide new perspectives on disease or treatments with the novel features space. For scientific understanding purpose, capability to derive knowledge from a trained DL model for explanations is an emerging topic of research and indispensable, especially in cancer research using the DL approaches.

In fact, the definition for interpretable DNNs is a broad concept. However, it can be generalized into two definition [47]: (1) the interpretation is the mapping process from abstract concept (i.e. predicted class) into the domain that makes sense to human; (2) the explanation of features in interpretable domain, that contributed to a given example to produce a decision. The simple approach is based on the nature of raw data is often high dimensional, some dimension reduction algorithms, such as PCA, t-SNE or UMAP can be useful to transform the input data into a 2D or 3D space, which can be visualized easily. However, dimension reduction is not a good choice when it does not provide information about why one feature different from other feature of the same class. Feature visualization is a more universal approach by exploring in a way which users can look at the result and derive information by themselves.

An earliest idea to explore the way of visualizing what a unit computes in an arbitrary layer of a DNNs was presented in [48]. The goal is to look for input patterns which maximize the activation of a given unit, also known as *activation maximization* (AM). By keeping the output and weights of the model constant, AM looks for an image  $\mathbf{x}^*$  that maximize the activation  $h_{ij}(\theta, \mathbf{x})$  of given unit  $i$  from layer  $j$ . The idea can be formulate into Eq. 2

$$\mathbf{x}^* = \text{argmax} h_{ij}(\theta, \mathbf{x}) \quad (1)$$

A variant of AM also takes account of “expert” opinion by incorporating natural image priors into the objective function [49]. The modified activation maximization includes two regularizers

$$\mathbf{x}^* = \text{argmax}(h_{ij}(\theta, \mathbf{x}) - R_\theta(\mathbf{x})) \quad (2)$$

Here,  $R_\theta(\mathbf{x})$  refers to set of two regularizers: total variation and jitter, each of which penalizes the search in a different way to collective. There was several improved variant of AM but most had a shortcoming in that it does not explain how the entire network operates and each neural unit is visualized separately.

Another branch of research is to map regions within the input image that contribute the most to the output. Many researches have been done to further improve interpretability of more complex using saliency mapping, worthwhile mentioning are [50, 51] and [52]. As reported in [50], the class saliency map  $M \in \mathbb{R}^{m \times n}$  by calculating the gradient of target class  $c$  with respect to an input image  $I_0$  (with also  $m$  rows and  $n$  columns) follows Eq. 3

$$M_{ij} = \begin{cases} |w_{b(i,j)}| & \text{for grey-scale image} \\ \max_c |w_{b(i,j,color\_channel)}| & \text{for multi-channel image} \end{cases} \quad (3)$$

where  $b_{i,j,c}$  is the index of the element of  $w$  corresponding to the image pixel in the  $i$ -th row,  $j$ -th column and colour channel if the input is multiple-channel. Additionally,  $w$  is the derivative of class score  $S_c$  with respect to the image  $I$  at the point  $I_0$ :

$$w = \frac{\partial S_c}{\partial I} \quad (4)$$

Note that the saliency maps are extracted using a classification convolutional layers trained on the image labels and no additional information is required. Similar to saliency mapping approach, [53] proposed the use of a multi-layered deconvolutional network to project the extracted features activation back to the input pixel space. In the CNN, a deconvolutional layer is first attached to each of the CNN layers, iterate through each units of a layer then unpool the output, rectify and filter to reconstruct the changes made by previous layer. The computation approach of a devolution network with ReLU activation function is equivalent to the computation of the derivative of  $\partial S_c / \partial I$  in the gradient based visualization using saliency mapping [50].

A disadvantage of measuring the accuracy quantitatively of interpretable DL is no such label for each features or each layer output. Additionally, it is quite hard to validate a model for the interpretability task since each model start with random initial weights and learn the feature extraction differently. However, the advances in transfer learning, there are more and more DL models are built on top of other reliable pre-trained model, the problem with uncertain initial weights can be avoided as well as less information desirable layerwise visualization.

### **3.2 Suitable Loss Functions for Biomedical Data**

Biomedical data is usually unbalanced, unstructured, and multi-modal in distribution, which causes new challenges in training a neural network. For example, unbalanced data may result in a trained network with a high precision but a low recall rate, which means that the trained network is biased to the class having a higher number of samples. It is undesirable in medical applications as usually FNs are as much importance compared to the FPs. Researchers have proposed different ways to address this challenge. Some of them have focused on a loss of function addressing imbalanced data-related problems. In the following, we review some widely used and suitable loss functions in medical-related applications.

### 3.2.1 Cross-Entropy

Cross-Entropy is the most commonly used loss function for neural network. The predictions are always between 0 and 1 and are usually given by the Logistic/Sigmoid activation function defined as  $P(\hat{Y} = 0) = \frac{1}{1+e^{-x}} = \hat{p}$  and  $P(\hat{Y} = 1) = 1 - \frac{1}{1+e^{-x}} = 1 - \hat{p}$ . The predictions are interpreted as a probability. The training phase which is the minimization of the loss function corresponds to the maximization of the conditional log-likelihood of the data. That is the reason of good performance of this loss function. In binary classification, let  $P(Y=0)=p$  and  $P(Y=1)=1-p$ . Then cross entropy (CE) can be defined as Eq. 5. The value of the Cross-Entropy loss increases as the predicted probability deviates from the true label. A perfect model would have a log loss of 0. Cross-Entropy is defined as follows

$$l_{\text{CE}}(p, \hat{p}) = -(p \times \log(\hat{p}) + (1 - p) \log(1 - \hat{p})) \quad (5)$$

### 3.2.2 Weighted Cross Entropy

Weighted Cross Entropy is a variant of Cross-Entropy loss where all positive samples get weighted by a coefficient. It is inherited all the advantages of the regular Cross-Entropy and is usually used in terms of imbalanced data. For example, when there is an image with 10% black pixels and 90% white pixels, regular Cross-Entropy does not work very well. Weighted

$$l_{\text{WCE}} = -(\beta p \log(\hat{p}) + (1 - p) \log(1 - \hat{p}))$$

The parameter  $\beta$  is used to decrease the number of false negatives ( $\beta > 1$ ) or to decrease the number of false positives ( $\beta < 1$ ).

### 3.2.3 Focal Loss

Focal loss aims to down-weight the contribution of easy samples so that the CNN focuses more on challenging samples. In the following the mathematical representation of Focal loss is provided

$$l_{\text{FL}}(p, \hat{p}) = -(\alpha(1 - \hat{p})^\gamma p \log(\hat{p}) + (1 - \alpha)\hat{p}^\gamma(1 - p) \log(1 - \hat{p}))$$

### 3.2.4 Dice Loss

The Dice loss function achieved good results in medical image analysis [54] and is defined as follow:

$$\text{DL}(p, \hat{p}) = 1 - \frac{2p\hat{p} + 1}{p + \hat{p} + 1}$$

where  $p \in \{0, 1\}$  and  $0 \leq \hat{p} \leq 1$ .

The Dice loss function works better on image compare to work on single pixels. It returns a scalar value for the whole image as follows:

$$\text{DL}(p, \hat{p}) = 1 - \frac{2\sum_{w,b} p_{wh}\hat{p}_{wh} + 1}{\sum_{w,b} p_{wh} + \sum_{w,b} \hat{p}_{wh} + 1}$$

### 3.2.5 Combinations

Using multiple loss function is also quite popular in medical image analysis. The following function achieved good results in medical data competitions:

$$\text{CE}(p, \hat{p}) + \text{DL}(p, \hat{p})$$

where CE denotes Cross-Entropy and returns a tensor, while DL (Dice Loss), as explained in the previous paragraph, returns a scalar for each image in the batch.

Overall, the existing diverse collection of loss functions developed and optimized for training NN in non-biomedical fields offers a valuable resource for the selection and implementation of suitable functions while training biomedical datasets.

## 4 Improving Deep NN Implementation for Biomedical Genomics and Imaging Data

NN models have recently been applied for automated analysis of pathological images. These methods are dependent on a large sample size of training images, and these images require diagnostic labels manually annotated by trained pathologists [55]. These studies show that informative features from pathological images can be used for diagnosing diseases such as for cancer staging [25]. With suitable NN architectures and datasets, these models have proven high performance that is, in some cases, outperforming accuracy level achieved by pathologists, for example, in melanoma diagnosis [56]. Although using tissue morphology as the mere input for NN proves to be useful, recent advances in imaging and molecular profiling technologies open up a new resource for improving automated diagnosis.

As one of the pioneering groups in integrating molecular labels with pathological annotation, we develop two analytical software programs, named as HEMnet and SpaCell [57], which show the unexplored potential of NN based on pixel-level labelling (*see* Fig. 1). While HEMnet is designed for integrating H&E images with cancer-marker staining images, SpaCell utilizes a much richer data generated from the recent genomics technology known as spatial transcriptomics (ST). We describe below the NN implemented in SpaCell and HEMnet, focusing on aspects we introduced in previous sections such as, transfer learning, loss functions, and interpretability.

In both cases, data preprocessing for image data implements tiling strategy to solve two inherent challenges, the small sample size and the large high resolution whole slide images (WSI). Since the number of histological sample is often small and WSI contains gigapixels, tiling strategy converts WSI into thousands of small tiles. We also developed a normalization strategy to allow for

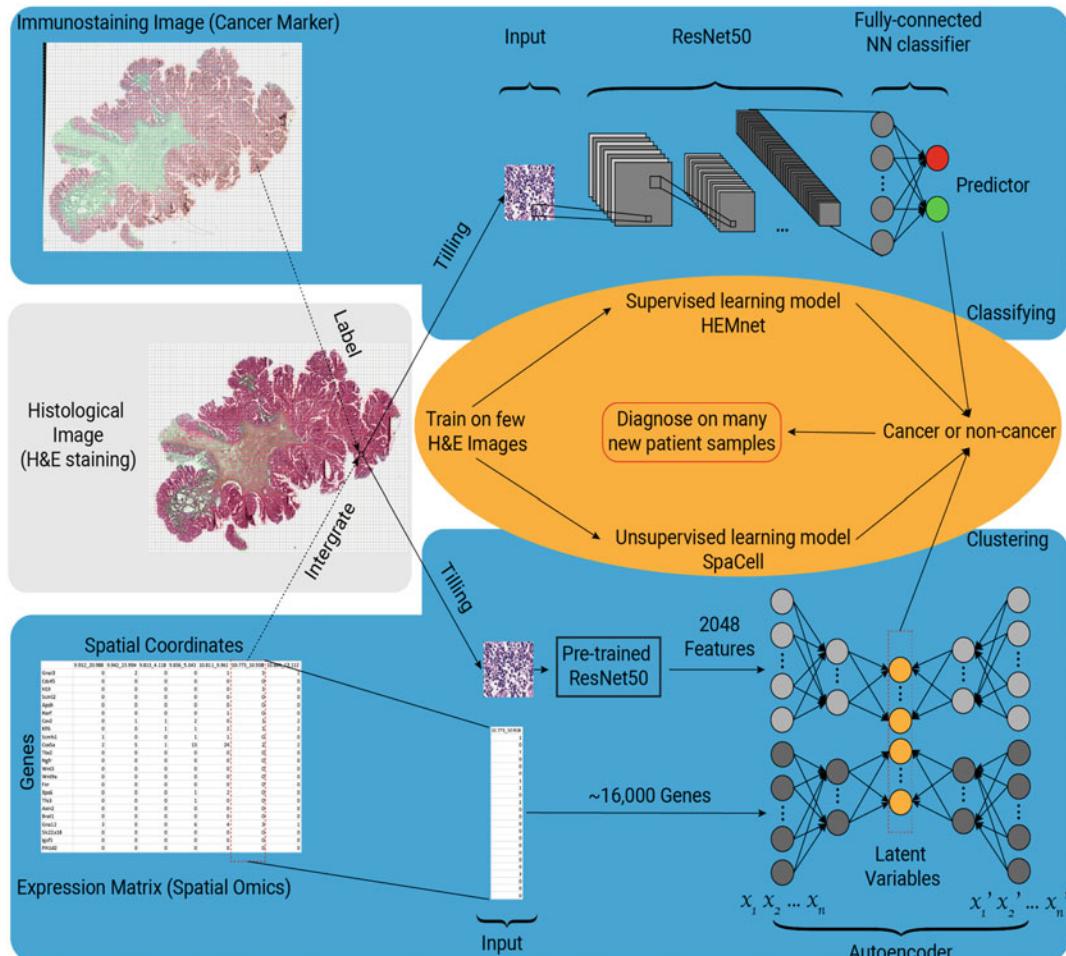
including a large number of images that contain technical variation. We applied colour cast removal and stain normalization. To increase model performance and generalizability, we perform augmentation steps such as random rotation and standardization of the tiles for each training step.

#### **4.1 Integrating Genomics Data and Tissue Imaging Data for Classifying Cancer Cell Types from H&E Images**

H&E images are abundant in cancer clinics but are under-utilized. Tissue diagnostic studies are carried out and interpreted by pathologists for virtually all cancer patients. This process is variable, qualitative, and is subject to availability of the pathologists. The overwhelming majority of these are stained with haematoxylin and eosin (H&E), and have a large impact on decision-making of physicians [13]. Deep neural network methodology is especially suitable for analysing multi-modal data and image data, where no assumption on distribution of features is required [12, 58, 59]. A number of proprietary algorithms for image interpretation have been approved by the Food and Drug Administration (FDA), for example, DNN application in diagnosing diabetes-related vision loss using eye images, and the list is expanding rapidly [59]. However, an important requirement is to increase the accuracy and interpretability of the diagnostic model [11]. In addition to traditional methods for increasing interpretability in deep learning models, biomedical data offer an unique opportunity to incorporate molecular signatures with imaging data to find underlying causes of morphological differences displayed in those images. However, current machine learning methods for analysing H&E imaging data do not utilize molecular data. We aim to combine molecular measurements with image pixel data to characterize tissue morphological images beyond pathological annotation with higher accuracy and interpretability. Some initial attempts have been successful with predicting genome instability [60].

We propose a novel analytical framework to transfer labels from cancer-marker stains to H&E images, followed by training DNN using pixel-level annotation (*see* Fig. 1). In this model, we developed a robust image registration pipeline.

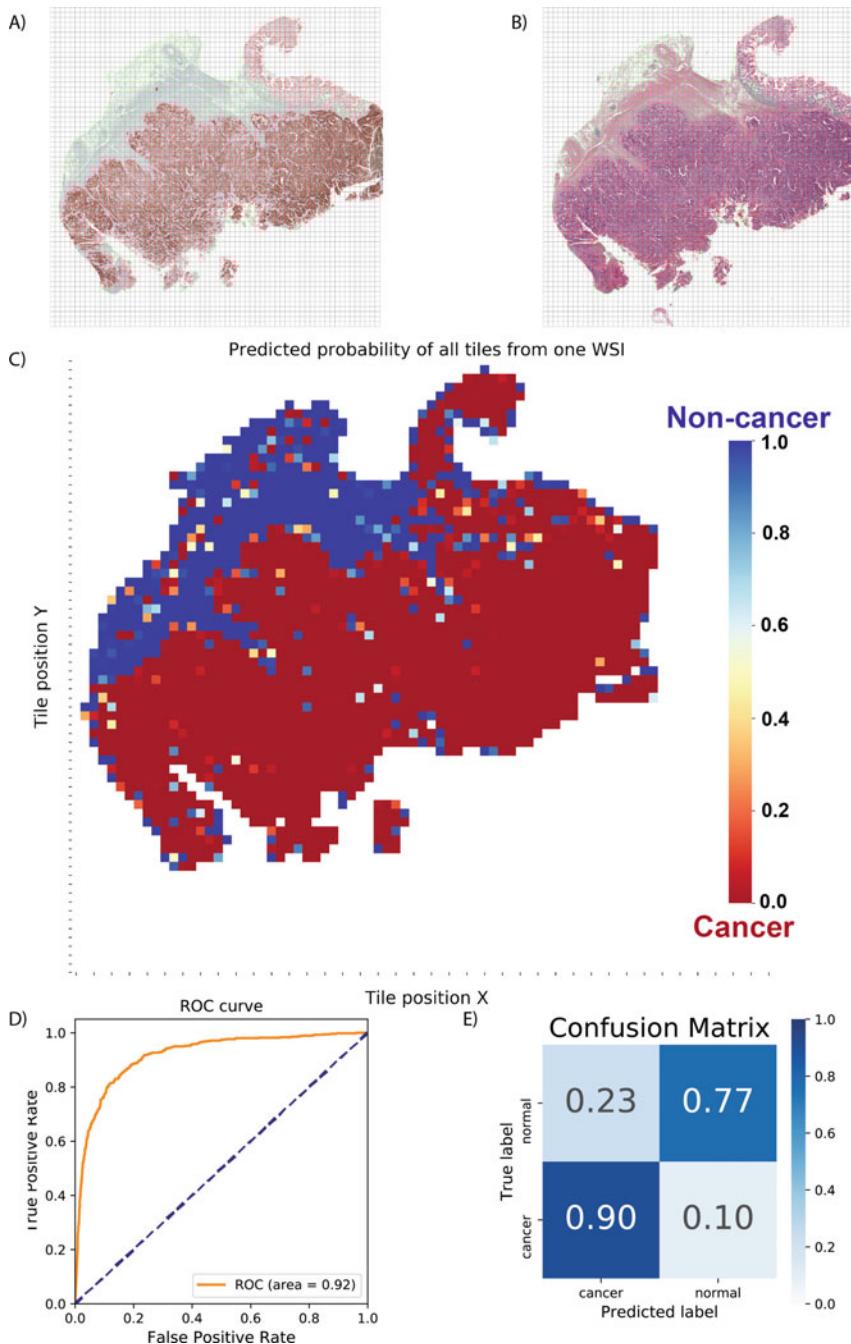
We show that, by label transferring and tile-level training, we could achieve a high performance model for predicting gastric cancer cells at pixel-level resolution (*see* Fig. 2). To establish a ground-truth classification, we applied an automated thresholding approach to identify TP53 positive pixel (suggesting cancer cells) compared to TP53 negative pixel (non-cancer cells). The ground-truth data allow for assessing model performance, which suggests high accuracy (*see* Fig. 2).



**Fig. 1** Two approaches to utilizing image pixel annotation and molecular profiling for cancer tissue diagnosis. H&E images are the main model inputs for classifying cancer and non-cancer cells in a tissue. For model training, HEMnet uses cancer-marker staining, while SpaCell uses transcriptome profiles of thousands of genes. Fully connected NNs, autoencoder, and transfer learning are used. Models are trained at tile level, allowing for accurate prediction is achieved at higher resolution compared to traditional pathological annotation

#### 4.2 Applications of Spatial Omics and Deep Learning in Characterizing Cancer Heterogeneity

Most of the current knowledge on cancer heterogeneity derives from studies where spatial information is technically discarded due to the need to dissociate cells from the native tissues. Cancers that arise from the same organ and histologically resemble one another can have quite different prognoses and responses to treatment. It is possible that deep analysis of the spatial transcriptomics will allow definition of clinically useful subsets of cell types and their molecular interactions in the cancer tissues that might determine targeted and personalized treatment. The use of spatial transcriptomics and machine learning can potentially better define the nature of the heterogeneities within apparently identical cancers, using two



**Fig. 2** HEMnet classifies cancer vs non-cancer regions in a H&E stained section of tumour tissue. Registration from a TP53 label (**a**) to a H&E image (**b**) was achieved. (**c**) Classification on H&E image is shown as a probability range for binary classes, where 0 (red) suggests cancer cells and 1 (blue) suggest normal cells. (**d**) and (**e**) are matrices for HEMnet model performance

common cancers where tumours that are similar by appearance are recognized to have a wide range of possible outcomes and responses to treatment. Understanding cell type composition, evolution, and interaction within a tumour tissue is required for accurate prognosis and monitoring of cancer and for developing immunotherapies. To date, many immunotherapies fail or encounter resistance and tumour recurrence due to the outgrowth of individual cell clones. Resistant cell clones and metastasis-initiating cells may be missed from the bulk tumour analysis.

Spatial transcriptomics data consist of both H&E image data and RNA sequencing data for the same tissue. This creates an ideal dataset for training a classification model that uses H&E image pixels as a standard input, but with much improved molecular labelling by RNA expression data. Spatial transcriptomics sequencing generates labels for 16,000 genes, not just a few known markers. Using spatial transcriptomics sequencing data, microenvironments and cell types are labelled on H&E images in an automated and unsupervised manner. With this unique dataset, it is possible to use the contexture information in histology images to characterize cancer. Notably, it is possible to apply transfer learning to utilize information from pre-trained models in non-medical image databases such as ImageNet database [58], allowing us to use high-level representation of images in other unrelated fields to mitigate the limitation in the small sample size in the training data. Furthermore, H&E image data for a large range of cancer types have paired genomics data from the TCGA (The Cancer Genome Atlas) database, creating an important resource for integrating multi-modal data to better characterize cancer.

However, existing methods for spatial data analysis use gene expression and the tissue images are mainly used for visualization purposes [61, 62]. These methods do not utilize image pixel information, which contain data about tissue morphology, and thus missing the quantitative links between tissue morphology with gene expression measured on the same tissue. We developed software called SpaCell, implemented innovative NN architectures that can utilize information from both H&E stained tissue images and gene expression matrices for either identification of cell types within one tissue slide or classification of disease progression within a whole dataset.

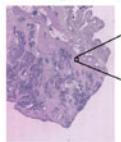
In the SpaCell model (*see* Fig. 3), for each spot image, a pre-trained ResNet50 convolutional neural network (CNN), which makes use of network weights trained from the ImageNet database, is used to extract a latent tile feature vector representing the high-level contexture of the spot image. Two autoencoder models are separately executed for each of the tile feature vector and its corresponding spot gene counts. Two latent spaces, with 20 neurons each, are obtained and combined into one space that is

A)

**Gene Model**

	3.MRN021.B03	3.977021.B08	1.399421.B04	4.875420.B05	3.947420.B01	2.387420.B02	-
Fam236a	0	1.977021.B08	1.399421.B04	4.875420.B05	3.947420.B01	2.387420.B02	-
Neft	0	5	0	2	2	10	-
Semaphorin	0	0	0	2	1	0	-
Tum122	0	0	0	2	0	0	-
Nissl	0	0	0	8	0	9	-
Utr	0	0	0	0	2	1	-
Prok3n	2	0	0	0	1	3	-
WIF10212204	0	0	0	0	0	0	-
Tube	0	0	0	0	0	0	-
Zfp106	0	2	0	2	1	0	-
Vpx	0	0	0	1	0	1	-
Arx12	0	0	0	0	0	1	-
Tc14	0	0	0	0	0	0	-
Filip1	0	2	0	0	0	0	-
Mab21	0	1	0	4	2	4	-
Abi1	0	0	0	0	0	0	-
Rpl10a-pc1	0	1	0	2	3	0	-
Apcd	0	2	0	1	1	2	-
Cdh2	0	0	2	1	2	11	-
Gm2237	0	0	0	0	0	0	-
Asptf1	1	1	0	9	1	4	-

B)

**Image Model**

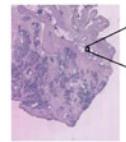
Pre-trained ResNet50

Autoencoder

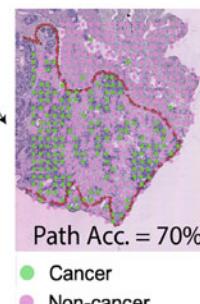
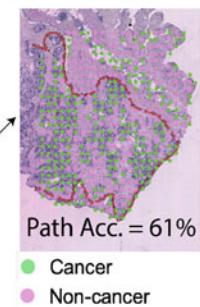
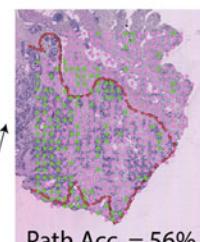
K-Means clustering

**Combined Model**

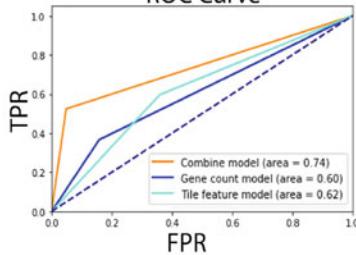
	3.MRN021.B03	2.877021.B08	1.399421.B04	4.875420.B05	3.947420.B01	2.387420.B02	-
Fam236a	0	1	0	2	2	10	-
Neft	0	0	2	0	1	0	-
Semaphorin	0	0	0	0	0	0	-
Tum122	0	0	0	0	0	0	-
Nissl	0	0	1	1	0	0	-
Utr	0	0	1	1	0	0	-
Prok3n	2	0	0	0	1	3	-
WIF10212204	0	0	0	0	0	0	-
Tube	0	0	1	0	0	0	-
Zfp106	0	0	0	1	0	1	-
Vpx	0	0	0	1	0	1	-
Arx12	0	0	0	0	1	1	-
Tc14	0	0	0	0	0	0	-
Filip1	0	2	0	0	0	0	-
Mab21	0	1	0	4	2	4	-
Abi1	0	0	0	0	0	0	-
Rpl10a-pc1	0	1	0	2	3	0	-
Apcd	0	2	9	1	0	2	-
Cdh2	0	0	2	1	2	11	-
Gm2237	0	1	0	9	1	4	-
Asptf1	1	1	0	9	1	4	-



Pre-trained ResNet50



C)

**ROC Curve**

P3.3	Combined model	Gene model	Image model
Accuracy	0.70	0.56	0.61
Precision	0.94	0.76	0.69
F-score	0.67	0.50	0.64

**Fig. 3** SpaCell for quantifying cancer heterogeneity. SpaCell can predict cancer cell and stromal cell in prostate cancer tissue at a high resolution and the prediction is consistent to pathological annotation. Each dot represents one tile with gene expression data and is classified by a combined model of both imaging and expression data, or one of the two data alone

representative for both image and count data. The latent variables are then used to perform clustering (default as K-means clustering) to identify cell types. We show that the combined model of both imaging and count data outperforms the models with just imaging or count data alone.

## 5 Conclusions

Building on the unique combination of genomics and spatial information generated by spatial technologies, there is a vast opportunity to implement an innovative analytical approach to produce software tools that can assist clinicians in automated and quantitative assessment of tumour heterogeneity. Example of clinical application is software for automated diagnosis of histopathological images, which are collected for virtually all cancer patients. Deep learning applications on biomedical data will be beneficial in multiple aspects: for clinicians, via rapid, accurate image interpretation; for health systems, by improving workflow; and for patients, by enabling them to process their own data to promote health [58, 59]. In a clinical setting, rapid and automated identification of the degree and nature of cancer-immune cell interactions can be instrumental in determining whether options for immunotherapy should be explored or whether more detailed and costly immune diagnostics are required. Digital-pathology-based diagnostic and prognostic biomarkers identified in this project will likely provide decision support for traditional pathologic interpretation in the clinical setting. The development of research in this area will contribute to the “omics” revolution in health care, promoting applications of advanced digital health intelligence, and contributing to a society with better access to disease prevention.

## Acknowledgements

We thank members in Nguyen’s Biomedical Machine Learning Lab for helpful discussion. This work has been supported by the Australian Research Council (ARC DECRA DE190100116), the University of Queensland, and the Genome Innovation Hub.

## References

- Ramilowski JA, Goldberg T, Harshbarger J, Kloppman E, Lizio M, Satagopam VP, Itoh M, Kawaji H, Carninci P, Rost B, Forrest ARR (2015) A draft network of ligand-receptor-mediated multicellular signalling in human. *Nat Commun* 6(1):7866
- Puram SV, Tiros I, Parikh AS, Patel AP, Yizhak K, Gillespie S, Rodman C, Luo CL, Mroz EA, Emerick KS, Deschler DG, Varvares MA, Mylvaganam R, Rozenblatt-Rosen O, Rocco JW, Faquin WC, Lin DT, Regev A, Bernstein BE (2017) Single-cell transcriptomic analysis of primary and metastatic tumor

- ecosystems in head and neck cancer. *Cell* 171(7):1611–1624.e24
3. Boutros C, Tarhini A, Routier E, Lambotte O, Ladurie FL, Carbonnel F, Izzeddine H, Marabelle A, Champiat S, Berdelou A, Lanoy E, Texier M, Libenciu C, Eggermont AMM, Soria JC, Mateus C, Robert C (2016) Safety profiles of anti-CTLA-4 and anti-PD-1 antibodies alone and in combination. *Nat Rev Clin Oncol* 13(8):473–486. <http://search.proquest.com/docview/1806076231/>. Accessed 7 Dec 2019
  4. Galon J, Costes A, Sanchez-Cabo F, Kirilovsky A, Mlecnik B, Lagorce-Pagès C, Tosolini M, Camus M, Berger A, Wind P, Zinzindohoué F, Bruneval P, Cugnenc PH, Trajanoski Z, Fridman WH, Pagès F (2006) Type, density, and location of immune cells within human colorectal tumors predict clinical outcome. *Science* 313(5795):1960
  5. Kivisaari A, Kähäri VM (2013) Squamous cell carcinoma of the skin: emerging need for novel biomarkers. *World J Clin Oncol* 4(4):85
  6. Uhlen M, Zhang C, Lee S, Sjöstedt E, Fagerberg L, Bidkhor G, Benfeitas R, Arif M, Liu Z, Edfors F, Sanli K, Von Feilitzen K, Oksvold P, Lundberg E, Hofer S, Nilsson P, Mattsson J, Schwenk JM, Brunnström H, Glimelius B, Sjöblom T, Edqvist PH, Djureinovic D, Micke P, Lindskog C, Mardinoglu A, Ponten F (2017) A pathology atlas of the human cancer transcriptome. *Science* 357(6352):pii: eaan2507
  7. Ståhl PL, Salmén F, Vickovic S, Lundmark A, Navarro JF, Magnusson J, Giacomello S, Asp M, Westholm JO, Huss M, Mollbrink A, Linnarsson S, Codeluppi S, Borg k, Pontén F, Costea PI, Sahlén P, Mulder J, Bergmann O, Lundeberg J, Frisén J (2016) Visualization and analysis of gene expression in tissue sections by spatial transcriptomics. *Science* 353(6294):78
  8. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444. <http://search.proquest.com/docview/1684430894/>. Accessed 7 Dec 2019
  9. Schmidhuber J (2015) Deep learning in neural networks: an overview. *Neural Netw* 61:85–117
  10. Libbrecht MW, Noble WS (2015) Machine learning applications in genetics and genomics. *Nat Rev Gen* 16(6):321–332
  11. Hu Z, Tang J, Wang Z, Zhang K, Zhang L, Sun Q (2018) Deep learning for image-based cancer detection and diagnosis A survey. *Pattern Recognition* 83:134–149
  12. Eraslan G, Avsec Ž, Gagneur J, Theis FJ (2019) Deep learning: new computational modelling techniques for genomics. *Nat Rev Genet* 20(7):389–403
  13. Wei JW, Tafe LJ, Linnik YA, Vaickus LJ, Tomita N, Hassanpour S (2019) Pathologist-level classification of histologic patterns on resected lung adenocarcinoma slides with deep neural networks. *Scientific Reports* 9(1):3358
  14. Sun D, Wang M, Li A (2019) A multimodal deep neural network for human breast cancer prognosis prediction by integrating multi-dimensional data. *IEEE/ACM Trans Comput Biol Bioinform* 16(3):841–850
  15. Ching T, Zhu X, Garmire L (2018) Cox-nnet: an artificial neural network method for prognosis prediction of high-throughput omics data. *Plos Comput Biol* 14(4):e1006076
  16. Krizhevsky A, Sutskever I, Hinton G (2017) ImageNet classification with deep convolutional neural networks. *Commun ACM* 60(6):84–90
  17. Xing F, Xie Y, Yang L (2016) An automatic learning-based framework for robust nucleus segmentation. *IEEE Trans Med Imag* 35(2):550–566
  18. Shen D, Wu G, Suk HI (2017) Deep learning in medical image analysis. *Annu Rev Biomed Eng* 19:221–248
  19. Simon O, Yacoub R, Jain S, Tomaszewski J, Sarder P (2018) Multi-radial LBP features as a tool for rapid glomerular detection and assessment in whole slide histopathology images. *Sci Rep* 8(1):2032–2032
  20. Cheng JZ, Ni D, Chou YH, Qin J, Tiu CM, Chang YC, Huang CS, Shen D, Chen CM (2016) Computer-aided diagnosis with deep learning architecture: applications to breast lesions in US images and pulmonary nodules in CT scans. *Sci Rep* 6(1):24454
  21. Cruz-Roa A, Gilmore H, Basavanhally A, Feldman M, Ganeshan S, Shih N, Tomaszewski J, González F, Madabhushi A (2017) Accurate and reproducible invasive breast cancer detection in whole-slide images: a deep learning approach for quantifying tumor extent. *Sci Rep* 7(1):46450. <http://search.proquest.com/docview/1903454183/>. Accessed 7 Dec 2019
  22. Sirinukunwattana K, Ahmed Raza SE, Tsang YW, Snead DRJ, Cree IA, Rajpoot NM (2016) Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. *IEEE Trans Med Imag* 35(5):1196–1206
  23. Ertosun M, Rubin D (2015) Automated grading of gliomas using deep learning in digital pathology images: a modular approach with

- ensemble of convolutional neural networks. *AMIA Annu Symp Proc* 2015:1899–1908
24. Anthimopoulos M, Christodoulidis S, Ebner L, Christe A, Mougiakakou S (2016) Lung pattern classification for interstitial lung diseases using a deep convolutional neural network. *IEEE Trans Med Imag* 35 (5):1207–1216
  25. Coudray N, Ocampo PS, Sakellaropoulos T, Narula N, Snuderl M, Fenyo D, Moreira AL (2018) Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning (report). *Nat Med* 24(10):1559
  26. Esteve A, Kuprel B, Novoa RA, Ko J, Swetter SM, Blau HM, Thrun S (2017) Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542(7639):115
  27. Pan SJ, Yang Q (2010) A survey on transfer learning. *IEEE Trans Knowl Data Eng* 22 (10):1345–1359
  28. Tang B, Pan Z, Yin K, Khateeb A (2019) Recent advances of deep learning in bioinformatics and computational biology. *Frontiers in Genetics* 10(214)
  29. Zeng K, Yu J, Wang R, Li C, Tao D (2017) Coupled deep autoencoder for single image super-resolution. *IEEE Trans Cybern* 47 (1):27–37
  30. Al-Stouhi S, Reddy CK (2016) Transfer Learning for Class Imbalance Problems with Inadequate Data. *Knowl Inf Syst* 48 (1):201–228
  31. Nagpal K, Foote D, Liu Y, Chen PHC, Wulczyn E, Tan F, Olson N, Smith JL, Mohtashamian A, Wren JH, et al (2019) Development and validation of a deep learning algorithm for improving Gleason scoring of prostate cancer. *NPJ Digit Med* 2(1):48
  32. Isaksson J, Arvidsson I, Åström K, Heyden A (2017) Semantic segmentation of microscopic images of h&e stained prostatic tissue using CNN. In: 2017 international joint conference on neural networks (IJCNN). IEEE, Piscataway, pp 1252–1256
  33. Khan UAH, Stürenberg C, Gencoglu O, Sandeman K, Heikkinen T, Rannikko A, Mirtti T (2019) Improving prostate cancer detection with breast histopathology images. *arXiv:190305769*
  34. Källén H, Molin J, Heyden A, Lundström C, Åström K (2016) Towards grading Gleason score using generically trained deep convolutional neural networks. In: 2016 IEEE 13th international symposium on biomedical imaging (ISBI). IEEE, Piscataway, pp 1163–1167
  35. Sermanet P, Eigen D, Zhang X, Mathieu M, Fergus R, LeCun Y (2013) Overfeat: integrated recognition, localization and detection using convolutional networks. *arXiv:13126229*
  36. Arvaniti E, Claassen M (2018) Coupling weak and strong supervision for classification of prostate cancer histopathology images. *arXiv:181107013*
  37. Arvaniti E, Fricker KS, Moret M, Rupp N, Hermanns T, Fankhauser C, Wey N, Wild PJ, Rueschoff JH, Claassen M (2018) Automated Gleason grading of prostate cancer tissue microarrays via deep learning. *Sci Rep* 8 (1):12054
  38. Campanella G, Silva VWK, Fuchs TJ (2018) Terabyte-scale deep multiple instance learning for classification and localization in pathology. *arXiv:180506983*
  39. Way GP, Greene CS (2018) Bayesian deep learning for single-cell analysis. *Nature Methods* 15(12):1009–1010
  40. Chaudhary K, Poirion O, Lu L, Huang S, Travers C, Garmire L (2018) Multi-modal meta-analysis of 1494 hepatocellular carcinoma samples reveals vast impacts of consensus driver genes on phenotypes. *BioRxiv*. <http://search.proquest.com/docview/2071227297/>. Accessed 7 Dec 2019
  41. Zhang C, Song J, Pei Z, Jiang J (2016) An imbalanced data classification algorithm of de-noising auto-encoder neural network based on smote. EDP Sciences, Les Ulis, vol 56. <http://search.proquest.com/docview/1786240651/>. Accessed 7 Dec 2019
  42. Way G, Greene C (2017) Extracting a biologically relevant latent space from cancer transcriptomes with variational autoencoders. *BioRxiv*. <http://search.proquest.com/docview/2071245134/>. Accessed 7 Dec 2019
  43. Lin C, Jain S, Kim H, Bar-Joseph Z (2017) Using neural networks for reducing the dimensions of single-cell RNA-seq data. *Nucleic Acids Res* 45(17):e156–e156. <http://search.proquest.com/docview/1947096259/>. Accessed 7 Dec 2019
  44. Danaee P, Ghaeini R, Hendrix DA (2017) A deep learning approach for cancer detection and relevant gene identification. In: Pacific symposium on biocomputing 2017. World Scientific, Singapore, pp 219–229
  45. Caruana R, Lou Y, Gehrke J, Koch P, Sturm M, Elhadad N (2015) Intelligible models for healthcare: predicting pneumonia risk and hospital 30-day readmission. In: Proceedings of the 21th ACM SIGKDD international

- conference on knowledge discovery and data mining. ACM, New York, pp 1721–1730
46. Doshi-Velez F, Kim B (2017) Towards a rigorous science of interpretable machine learning. arXiv:1702.08608
  47. Montavon G, Samek W, Müller KR (2018) Methods for interpreting and understanding deep neural networks. *Digit Signal Process* 73:1–15
  48. Erhan D, Bengio Y, Courville A, Vincent P (2009) Visualizing higher-layer features of a deep network. University of Montreal 1341 (3):1
  49. Nguyen AM, Yosinski J, Clune J (2016) Multi-faceted feature visualization: uncovering the different types of features learned by each neuron in deep neural networks. ArXiv abs/1602.03616
  50. Simonyan K, Vedaldi A, Zisserman A (2014) Deep inside convolutional networks: visualising image classification models and saliency maps. In: Workshop at international conference on learning representations
  51. Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A (2016) Learning deep features for discriminative localization. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2921–2929
  52. Dabkowski P, Gal Y (2017) Real time image saliency for black box classifiers. In: Proceedings of the 31st international conference on neural information processing systems, Curran Associates, NIPS'17, pp 6970–6979. <http://dl.acm.org/citation.cfm?id=3295222.3295440>. Accessed 7 Dec 2019
  53. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: European conference on computer vision. Springer, Berlin, pp 818–833
  54. Milletari F, Navab N, Ahmadi SA (2016) V-net: fully convolutional neural networks for volumetric medical image segmentation. In: 2016 fourth international conference on 3D vision (3DV). IEEE, Piscataway, pp 565–571
  55. Komura D, Ishikawa S (2018) Machine learning methods for histopathological image analysis. *Comput Struct Biotechnol J* 16:34–42. <https://doi.org/10.1016/j.csbj.2018.01.001>. <http://www.sciencedirect.com/science/article/pii/S2001037017300867>. Accessed 7 Dec 2019
  56. Esteva A, Kuprel B, Novoa RA, Ko J, Swetter SM, Blau HM, Thrun S (2017) Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542(7639):115–118
  57. Tan X, Su A, Tran M, Nguyen Q (2019) Spacell: integrating tissue morphology and spatial gene expression to predict disease cells. bioRxiv (Accepted Bioinformatics) <https://doi.org/10.1101/837211>. 837211
  58. Janda M, Soyer HP (2019) Can clinical decision making be enhanced by artificial intelligence? *British Journal of Dermatology* 180 (2):247–248
  59. Topol EJ (2019) High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine* 25(1):44–56
  60. Hekler A, Utikal JS, Enk AH, Solass W, Schmitt M, Klode J, Schadendorf D, Sondermann W, Franklin C, Bestvater F, Flraig MJ, Krahl D, von Kalle C, Fröhling S, Brinker TJ (2019) Deep learning outperformed 11 pathologists in the classification of histopathological melanoma images. *Eur J Cancer* 118:91–96. <https://doi.org/10.1016/j.ejca.2019.06.012>. <http://www.sciencedirect.com/science/article/pii/S0959804919303806>. Accessed 7 Dec 2019
  61. Navarro JF, Sjostrand J, Salmen F, Lundeberg J, Stahl PL (2017) ST Pipeline: an automated pipeline for spatial mapping of unique transcripts. *Bioinformatics* 33(16):2591–2593
  62. Dries R, Zhu Q, Dong R, Eng C-HL, Li H, Liu K, Fu Y, Zhao T, Sarkar A, Bao F et al (2020) Giotto, a toolbox for integrative analysis and visualization of spatial expression data. bioRxiv:701680



# Chapter 11

## Leverage Large-Scale Biological Networks to Decipher the Genetic Basis of Human Diseases Using Machine Learning

Hao Wang, Jiaxin Yang, and Jianrong Wang

### Abstract

A fundamental question in precision medicine is to quantitatively decode the genetic basis of complex human diseases, which will enable the development of predictive models of disease risks based on personal genome sequences. To account for the complex systems within different cellular contexts, large-scale regulatory networks are critical components to be integrated into the analysis. Based on the fast accumulation of multiomics and disease genetics data, advanced machine learning algorithms and efficient computational tools are becoming the driving force in predicting phenotypes from genotypes, identifying potential causal genetic variants, and revealing disease mechanisms. Here, we review the state-of-the-art methods for this topic and describe a computational pipeline that assembles a series of algorithms together to achieve improved disease genetics prediction through the delineation of regulatory circuitry step by step.

**Key words** Regulatory networks, Multiomics, Genetic variants, Disease mechanisms, Fine-mapping

---

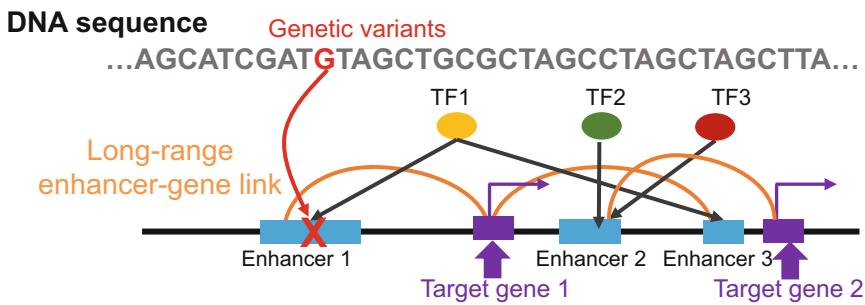
### 1 Introduction

Being able to precisely predict disease risks based on personal genomic sequences is one of the central questions in biomedical and bioinformatics research. To address this problem, we need to estimate mapping functions ( $f: \text{genetic variant} \rightarrow \text{disease status}$ ) that can link individual genetic variants to final disease outputs, and assign a probabilistic score to quantify the risk of observing a disease-related phenotype when specific genetic variants occur (Fig. 1), that is,  $P(\text{phenotype} | \text{genetic variants})$ .

Due to the complex nature of the human genome, the mapping functions for different genetic variants may be substantially different, depending on the biological roles of the genomic locations around specific genetic variants and how the genomic location

---

Hao Wang and Jiaxin Yang contributed equally to this work.



### Complex diseases:

- Breast cancer
- Alzheimer Disease
- Autoimmune Diseases

**Fig. 1** Large-scale regulatory networks associated with disease genetics. The disruptive effects of non-coding genetic variants propagate through complex regulatory networks (TF-enhancer-gene links) and finally induce complex disease phenotypes

interacts with other genomic loci in diverse biological processes. Therefore, accurate reconstruction of large-scale regulatory networks and network-based genetics modeling are two major tasks for advanced prediction of disease risks and characterization of disease mechanisms.

### **1.1 Traditional Genetics Approaches to Identify Disease-Associated Genetic Variants**

Genome-wide association studies (GWAS) have been the standard approach to look for genetic variants that are statistically correlated with specific complex diseases or traits [1, 2]. In a typical GWAS analysis for a specific disease (e.g., Alzheimer's disease), genome-wide genetic variants are profiled for a group of cases (i.e., patients with Alzheimer's disease phenotypes) and a group of controls (people without Alzheimer's disease phenotypes). In this case-control setting, each genetic variant (mainly refers to single nucleotide polymorphisms: SNP) is evaluated in turn. The frequency of occurrence of each genetic variant in cases is compared to its frequency of occurrence in controls. If the frequency in cases is significantly higher than the frequency in controls, based on the  $p$ -value calculated by Chi-squared test, then the specific genetic variant is predicted to be correlated with the disease. After testing all the genetic variants profiled in the genome, a set of significantly disease-associated SNPs will finally be identified.

One of the goals of GWAS analysis is to lay a quantitative foundation to predict disease risks so that people can build models (e.g., logistic regression or random forest models) where disease-associated SNPs are used as features and disease phenotypes are treated as response variables. Given the genomic sequence of a new patient, the model is expected to predict the risk of observing

disease phenotypes. But there are several fundamental limitations of the GWAS based approach, which have been widely recognized in the field [1–3]. The three major limitations are: (1) Low statistical power. Due to the huge number of genetic variants across the human genome (i.e. the number of hypotheses to test), many real disease-associated SNPs are evaluated to be not significant enough and are therefore missed in traditional GWAS (false negatives). (2) Unable to identify causal genetic variants. Due to the correlative nature of GWAS, the disease-associated SNPs may not be the actual causal variants that lead to disease phenotypes. This problem is further complicated by the fact that neighboring SNPs are usually correlated with each other across human populations (i.e. linkage disequilibrium). Among hundreds of SNPs located close to each other, it is computationally challenging to pinpoint which SNP is the causal one. (3) Lack of mechanistic understanding of disease. The GWAS approaches basically treat the underlying disease mechanisms as black box and are not able to determine why certain SNPs are correlated with disease phenotypes. But it is highly useful for new therapeutic development to know how the disruptive effects of specific SNPs at DNA sequence propagate to the organism level phenotypes. For example, since the majority (>97%) of the human genome are not protein-coding gene exons, i.e. are non-coding regions, significant SNPs from GWAS are usually not located in exons. Knowing the molecular mechanisms of these non-coding SNPs, such as which transcription factors bind there and which genes are regulated by the SNPs, can substantially improve our capacity to understand and prioritize disease-associated non-coding genetic variants. In the meantime, knowledge of disease mechanisms can also help to address the first two problems (low statistical power and identification of causal variants) by enabling sophisticated generative probabilistic models with prior knowledge flexibly incorporated, instead of black-box discriminative models.

## **1.2 Whole-Genome Annotation of Regulatory Elements**

In order to overcome these limitations, the genome sequence itself, along with the basic gene annotations, does not provide enough information. Additional functional genomics data are needed to provide a better annotation of the human genome, such as genomic locations of specific transcription factor binding sites, enhancers, insulators, chromatin accessible sites, and DNA methylated sites [4–9]. These diverse functional sites are collectively defined as regulatory elements, which can regulate tissue-specific gene expression at transcriptional level. Although not directly producing proteins, they are critical genomic units that can control the levels of specific proteins in different cellular contexts (e.g. tissues or cell-types). Therefore, the functional annotation of regulatory elements is highly useful for people to understand the non-coding portion of the human genome (>97%). Among different families of regulatory

elements, enhancers are the largest family and are characterized with complex features [10–13], such as combinatorial transcription factor (TF) binding within enhancers, multiple associated epigenetic signatures (e.g. histone modifications) and high levels of chromatin accessibility (Fig. 1).

Intuitively, assuming the genomic locations of tissue-specific enhancers are known along the whole human genome and across different tissues, a non-coding genetic variant located within a brain-specific enhancer may disrupt a specific gene's expression in the brain and may have higher prior probability to be a causal variant in Alzheimer's disease, compared to genetic variants located in genomic regions that are neither genes nor enhancers. Similarly, a genetic variant located within a T cell specific enhancer may be more likely to be a causal variant for autoimmune diseases.

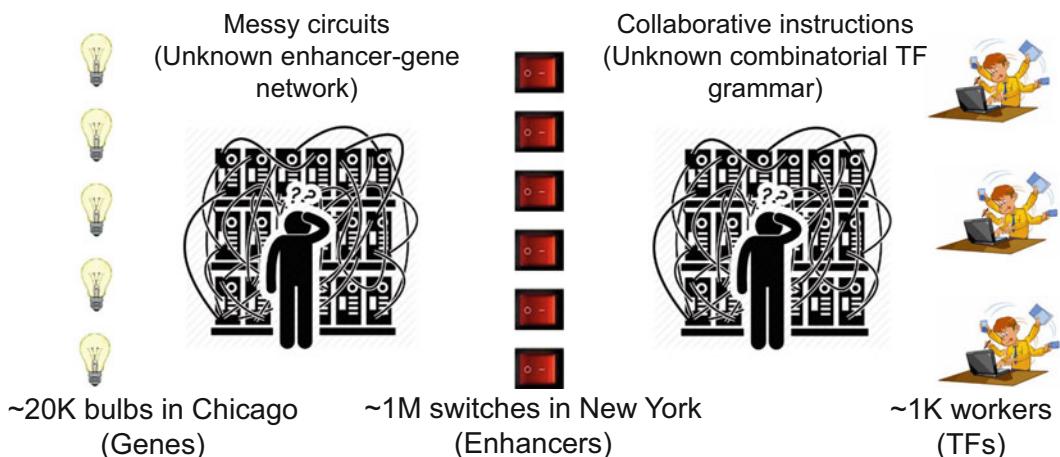
Therefore, as the first step in going beyond traditional GWAS analysis, algorithms to predict the genomic locations of regulatory elements (especially enhancers) in non-coding regions are urgently needed. Fortunately, next-generation sequencing-based techniques have produced large amounts of genome-wide omics data, such as ChIP-seq data of TF bindings and histone modifications, DNase-seq or ATAC-seq for chromatin accessibility, and bisulfite sequencing for DNA methylation, that can be used as features to predict enhancers in different tissues or cell-types [11, 12, 14].

### **1.3 Regulatory Networks Are Needed to Reveal the Mechanisms**

Enhancers are functionally important, because they are genomic hubs of combinatorial TF bindings and are responsible for precise regulation of specific target gene expression. Therefore, as the major components of tissue-specific transcriptional regulation, the complex interactions among upstream TFs, enhancers, and downstream target genes can be viewed as regulatory networks (Fig. 1). In general, multiple TFs bind to a specific enhancer, which is also tightly related with the epigenetic signal landscape around the enhancer, and turn on the enhancer's regulatory activity. Multiple enhancers may collaborate and regulate the transcription level of a specific target gene, and one enhancer may participate in the regulation of multiple genes. And the regulatory relationships are often tissue-specific. Another layer of complexity of tissue-specific regulatory networks comes from the long-range regulation of enhancers. For a specific enhancer element, the nearest gene in the genome may not be the target gene (Fig. 1). Considering the human genome as an article and a specific enhancer as a word, the precise interpretation of the word depends not only on the texts of local neighborhood but also distal contexts. Biologically, the long-range regulation of target genes by enhancers is mediated by 3D chromatin interactions or loops [15, 16]. Although the target gene's promoter may be located far away from the specific enhancer in the 1D genome, they may be located in proximity with each other in 3D space, based on chromatin interactions.

Knowing the structure of tissue-specific regulatory networks is apparently useful for genotype-based phenotype prediction. Based on the regulatory network, for a specific non-coding SNP in an enhancer, it is feasible to predict which TF's binding is disrupted, whether the enhancer's regulatory activity is turned off, and which specific target gene's expression may be affected. For a group of potential disease-associated SNPs, it is also useful to identify the group of target genes whose expression regulation may be disrupted and whether they are enriched in specific biological pathways. In addition, the regulatory networks also provide a foundation to predict the common TF binding sites that are disrupted by a group of disease-associated SNPs and further infer the master TF regulators underlying specific diseases. All these regulatory network-based predictions will lead to more in-depth understandings of disease mechanisms and how the disruptive effects of genetic variants propagate through regulatory networks.

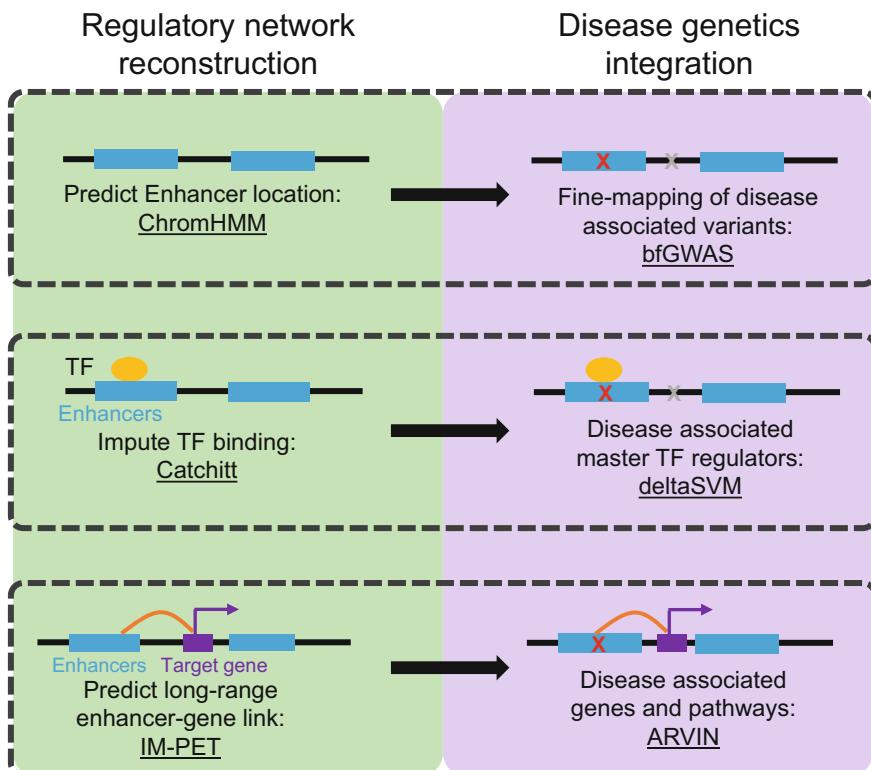
To further illustrate the complexity of tissue-specific regulatory networks, we can consider genes, the most important genomic units that may be directly related to diseases, as bulbs. And we can consider enhancers as switches that can control the bulbs (turn the bulbs on or off), and TFs as workers that send instructions to control the switches (Fig. 2). Multiple switches can control one bulb, and in the meantime, one switch may also control multiple bulbs. Similarly, there is also a multiple-to-multiple mapping from workers to switches. In addition, due to long-range control, the switches may be located in New York while the bulbs are in Chicago (Fig. 2). Furthermore, considering the scale of this system (i.e. >20k genes, ~1M enhancers, ~1k TFs), it is a significant challenging problem in machine learning to delineate the specific links in the network.



**Fig. 2** Illustration of the regulatory network complexity

### 1.4 Machine Learning Developments to Integrate Regulatory Networks for Advanced GWAS

Given the multiple benefits of network-based approaches to improve disease genetic analysis, a variety of machine learning algorithms are needed for each step, from identifying the specific genomic locations of regulatory elements to discovering the downstream dysregulated genes or upstream master TFs. Different modeling frameworks are needed, depending on the goal of each specific step. In this chapter, we present an algorithmic pipeline that assembles a series of machine learning algorithms to achieve an improved prediction of disease associated genetic variants and their related mechanisms. Along with the introduction of this pipeline, we provide systematic reviews of the machine learning algorithms included in the pipeline. Overall, there are three major steps (Fig. 3): (1) Predict enhancer locations in the human genome (i.e. identify ‘nodes’ for the networks), (2) Identify upstream TFs disrupted by SNPs (i.e. identify TF-enhancer edges in the networks), and (3) Identify downstream target genes dysregulated by SNPs (i.e. identify enhancer-gene edges in the networks). For each step, there are 2 major tasks: (1) Network re-construction and (2) Integration of networks with genetic variants. These algorithms not only help to discover SNPs underlying diseases more accurately, but also help to prioritize potential causal genetic variants and the perturbed regulatory pathways.



**Fig. 3** Flowchart of the suggested pipeline. The pipeline includes three steps and each step involves two algorithms

#### 1.4.1 Predict Regulatory Elements and Prioritize Disease-Associated SNPs

A tissue-specific regulatory network is a graph consisting of nodes and edges, where nodes are mainly non-coding regulatory elements or genes and edges represent regulatory links. Therefore, the first step of regulatory network re-construction is to identify the nodes, i.e. we need to know where they are in the human genome (Fig. 3). Since genes are already largely well annotated for the human genome, the major computational task is to identify the genomic locations of non-coding regulatory elements, e.g. enhancers. Traditionally, computational algorithms mainly rely on DNA sequence features to predict regulatory elements, such as sequence conservation across different species or TF motif scanning. Since the DNA sequence features are not tissue-specific, the prediction accuracy for functional regulatory elements, which are highly tissue-specific, is low. Based on recent development of next-generation sequencing based techniques, large amounts of ‘omics’ datasets have been profiled by several large consortia, such as ENCODE and Roadmap Epigenomics projects [17, 18] (Table 1).

These ‘omics’ datasets provide tissue-specific information of the functional states of potential regulatory elements along the

**Table 1**  
Large-scale data resources of multi-omics and genetic variants

Data resource names	Data types	Scalability	Website
ENCODE	DNase data	~200 DNase-Seq datasets in ~140 cell-types	<a href="https://www.encodeproject.org/">https://www.encodeproject.org/</a>
	TF binding data (ChIP-Seq)	~2000 ChIP-Seq datasets in ~100 cell-types	
	Gene expression data	100 total RNA-seq in 58 cell-types	
	Chromatin interaction	37 Hi-C datasets in 24 cell-types 52 ChIA-PET datasets in 27 cell types	
Roadmap Epigenomics	Epigenomic data	~100 H3K27ac/H3K9me3/ H3M4me1 Datasets in ~25 functional groups of cell-types	<a href="http://www.roadmapepigonomics.org/">http://www.roadmapepigonomics.org/</a>
	DNase data	~70 DNasel datasets in 44 functional groups of cell-types	
GTEX	SNPs	54 tissues	<a href="https://gtexportal.org/home/">https://gtexportal.org/home/</a>
Motif browser	Motif instances	Genome-wide binding of ~500 TFs	<a href="http://compbio.mit.edu/encode-motifs/">http://compbio.mit.edu/encode-motifs/</a>
3D genome browser	Hi-C ChIA-PET Capture Hi-C	>50 cell-types 6 cell-types >45 cell-types	<a href="http://promoter.bx.psu.edu/hi-c/">http://promoter.bx.psu.edu/hi-c/</a>

human genome, which are not directly provided by the DNA sequences alone. The major goal for this step is to explore and identify the combinatorial patterns of multiple ‘omics’ signal tracks that are predictive of regulatory elements, as for each specific genomic location it is no longer a single-valued signal (i.e. DNA sequence) but a high-dimensional array of ‘omics’ signals from different histone modifications, DNA methylation, chromatin accessibility and TF binding.

Generating an accurate genome-wide prediction of non-coding regulatory elements in this step is already a substantial achievement, since the predicted regulatory elements can be integrated with genetic variants (Table 1) and help to prioritize the real disease-associated SNPs (Fig. 3), compared to traditional GWAS analysis. From the probabilistic modeling perspective, predicted regulatory elements can help to set more biologically meaningful prior probabilities for genetic variants to be associated with diseases. Intuitively, SNPs located in regulatory elements are expected to have higher prior probabilities to induce diseases because they may cause dysregulation of gene expression, compared to other SNPs located in the genomic background. Instead of assuming uninformative priors across all SNPs in the genome (i.e. equal priors), the regulatory element annotation-based priors can help address the challenges of low statistical powers and fine-mapping of causal variants as we discussed above.

#### *1.4.2 Predict TF-Enhancer Links and Identify Master Regulators Disrupted by SNPs*

The second step in network re-construction is to link upstream TFs to specific enhancers, and the TF-enhancer links are a major type of regulatory links in the network (i.e. network edges). As we mentioned before, enhancers are functional in gene expression regulation because they are bound by specific combinations of different TFs. The binding of specific TFs can activate the enhancers to further regulate genes. Different enhancers are usually bound by different TFs, and there are groups of enhancers, i.e. co-regulated enhancer modules, which are usually bound by the same combinations of TFs. If a specific TF can bind to a specific enhancer, then there is a regulatory link between the node of the TF and the node of the enhancer, in the re-constructed networks. The basic approach to link TFs to specific enhancers is by scanning TF binding motifs in predicted enhancer regions. For example, if there is a strong sequence hit of GATA1 motif in a specific enhancer, it is assumed that GATA1 may have the potential to bind to the enhancer element. Due to the large false positives of sequence motif-based methods, genome-wide ChIP-seq data of specific TF binding is important for predicting TF-enhancer links, as TF ChIP-seq data represent real biological events of TF binding in-vivo. But due to the complexity of antibody preparation and cost, ChIP-seq data of TF binding is largely missing for different cell-types or

tissues. Therefore, an accurate computational algorithm to impute ChIP-seq signals of specific TF binding in specific cell-types along the human genome, based on features of DNA sequences and tissue-specific chromatin accessibility, is the major focus in this step of network re-construction.

The importance of imputed tissue-specific TF binding in enhancer regions, when combined with genetic variants, is to facilitate the identification of master TF regulators associated with diseases and reveal mechanistic insights. The imputed tissue-specific TF binding profiles will enable researchers to investigate whether disease-associated SNPs are enriched in certain TF's binding sites, and the enriched TFs are probably the master regulators to maintain normal cellular states. If a specific TF's binding sites are significantly disrupted by disease-associated SNPs, it indicates that the TF and its regulatory circuitry may be candidate targets for new therapeutic techniques. In the meantime, the imputed TF binding signals within enhancers can further help to prioritize the causal SNPs as SNPs located within TF binding sites are more likely to induce dysregulation of gene expression.

#### *1.4.3 Predict Enhancer-Gene Links and Discover Dysregulated Genes by SNPs*

The third step in regulatory network re-construction is to link specific enhancers to their downstream target genes, which is another major type of regulatory links in the network (Fig. 3). Enhancers are key non-coding genomic units because they play critical roles in precise tissue-specific regulation of gene expression. Knowing the downstream target genes that are regulated by specific enhancers provides a bridge to map non-coding SNPs to their effects in coding regions (i.e. gene expression changes). Although some enhancers regulate the nearest gene located in the genome, many of them participate in long-range regulation and their specific target genes may be located far away from them in the 1D genome. The long-range regulation of enhancers is biologically mediated by 3D chromatin interactions, where the distal target genes are placed in proximity to the corresponding enhancers by chromatin loops in 3D space. It is therefore one of the most challenging computational problems in bioinformatics. In order to predict the long-range target genes for specific enhancers, multiple 'omics' datasets across diverse panels of cell-types or tissues are needed.

The benefit of integrating the predicted enhancer-genes links with disease-associated non-coding SNPs is apparent: it enables the mechanistic interpretation of why certain non-coding genetic variants are associated with diseases. Considering that the majority of disease-associated SNPs are in non-coding regions and are enriched in enhancers, the predicted enhancer-gene links will expand our capability to understand how the disruptive effects of SNPs propagate through target gene expression perturbations and can also help to identify genes that are significantly associated with diseases.

Furthermore, based on the set of genes dysregulated by non-coding SNPs in enhancers, pathway enrichment analysis can be carried out to look for the affected biological processes underlying human diseases.

In order to address the machine learning problems listed above, the fast accumulation of multi-omics data and large cohorts of human disease genetics studies (Table 1) provides unique opportunities nowadays to develop novel computational algorithms and facilitate their wide implementation. This is a highly active research area with many new machine learning algorithms and types of computational software being developed [19, 20]. Based on our own testing and practical applications, we have selected a series of software and algorithms to use as a pipeline (Fig. 3) which can re-construct the regulatory networks step-by-step and leverage the network predictions to improve disease genetics studies. In the rest of this chapter, we will review the selected software through this pipeline, introduce their computational model design, and demonstrate their usage.

---

## 2 Software and Data Resources

The pipeline to be introduced mainly includes three steps for regulatory network re-construction. For each step, the predicted network results are integrated with genetic variants or DNA sequence features for improved disease analysis (Fig. 3). The six algorithms that we review and recommend here are: ChromHMM [21] (<http://compbio.mit.edu/ChromHMM/>), bfGWAS [22] (<https://github.com/yjingj/bfGWAS>), Catchitt [23] (<http://jstacs.de/index.php/Catchitt>), deltaSVM [24] (<http://www.beerlab.org/deltasvm/>), IM-PET [25] (<http://tanlab4generegulation.org/IM-PET.html>) and ARVIN [26] (<https://github.com/gaolong/arvin>). For software applications, there are large amounts of multi-omics datasets, SNP dataset and TF motif datasets, which were generated by several consortia [17, 18, 27–29] and are summarized in Table 1.

---

## 3 Methods

We describe the machine learning algorithms and software by following the three major steps in the pipeline. In each step, two algorithms will be introduced: one on regulatory network re-construction and one on integration with disease genetics.

### 3.1 Regulatory Element Prediction and Improved Fine-Mapping of Disease SNPs

#### 3.1.1 ChromHMM Algorithm

ChromHMM [21] is one of the most widely used algorithms to predict regulatory elements based on multiple epigenetic features. As we discussed in the introduction, traditional approaches based on DNA sequence features alone are not accurate enough, with many false positives, and are not sufficient to predict tissue-specific functional enhancers. ChromHMM was designed to leverage the tissue-specific epigenetic features, such as ChIP-seq signals of different histone modifications, because prior biological studies have shown that specific histone modification combinations are associated with different functional genomic units (e.g. enhancers, promoters, transcription regions and repressive domains). One of the major advantages of ChromHMM compared to other algorithms is the robustness to signal noise and a lower requirement of user-defined hyper-parameters.

*Model formulation of ChromHMM.* ChromHMM considers the regulatory element problem as a segmentation task: the human genome is composed with consecutive latent segments that are not directly observed, different segments have different generation probabilities of specific histone modifications, and there are spatial dependencies between segments. Thus, ChromHMM employs hidden Markov model (HMM) as the modeling framework to segment the human genome into different chromatin states, a subset of which correspond to regulatory elements. In the hidden Markov model, the hidden states are different chromatin states and each chromatin state has different emission probabilities to generate observable signals of different histone modifications. There is also a transition probability matrix to characterize the spatial relationship among different chromatin states. The observed data along the human genome are multiple signal tracks for different histone modifications, which are assumed to be generated by the underlying hidden chromatin state path in the model. The goal is to infer the optimal hidden chromatin state path which can maximize the probability of observing the given multi-tracks of histone modifications along the genome in the specific cell-type or tissue. The well-known Baum-Welch algorithm (a special case of the EM algorithm) is used to infer the final segmentation of the genome into chromatin states, along with the emission and transition probabilities. To assign the biological meanings to the learned hidden states, the segments of each state are compared with other genomic annotations (e.g. TSS, gene bodies, p300 binding sites) and biological labels are assigned based on the enriched annotations. For example, chromatin states that are enriched with H3K4me1 and H3K27ac histone modification peaks or p300 binding peaks are labeled as regulatory enhancer elements.

*Software usage and key parameters of ChromHMM.* Based on the instructions of ChromHMM [21], a typical command to run is: `java -jar ChromHMM.jar` plus settings of several parameters, including memory, number of processors, the directory of

binarized input data (i.e. histone modification ChIP-seq signal files), the directory for output and also the specific human genome assembly. The only major parameter that the user needs to decide is the number of hidden states for the hidden Markov model. In practice, it's suggested to try several different numbers (common choices include 5, 10, 15 and 18 states) and check the final results. The key outcomes from ChromHMM will be the genomic locations of different chromatin states. For our purpose, the locations of regulatory elements (especially enhancers) will be used for other steps (*see Note 1*).

### 3.1.2 *bfGWAS Algorithm*

*bfGWAS* [22] can be used to incorporate the predicted regulatory element locations into GWAS analysis to fine-map the disease-associated SNPs and, in the meantime, increase the statistical power (i.e. reduce the number of false negatives). Since the regulatory element information is integrated for improved estimation of prior probabilities of SNPs to be associated with diseases, *bfGWAS* outperforms traditional GWAS studies that rely on genotype and phenotype data alone.

*Model formulation of bfGWAS.* *bfGWAS* considers the identification of disease-associated SNPs as a variable selection problem and employs a Bayesian variable selection regression model as the inference approach. In the regression model, the disease phenotype of different individuals (including patients and controls) is treated as a responsible variable, which is a function of genotypes (i.e. the SNP profiles across different individuals). Each SNP is associated with an effect size, which follows a spike-and-slab prior. For the majority of SNPs, the effect size is near zero, while the disease-associated SNPs are associated with effect sizes that are significantly non-zero. Because of the huge number of SNPs in the human genome, the statistical power to identify SNPs with non-zero effect sizes is extremely low. *bfGWAS* borrows information from regulatory element annotations and incorporates the biological observation that potentially causal SNPs are enriched in regulatory element regions (e.g. enhancers, promoters, or general TF binding sites) compared to other genomic regions. Basically, for SNPs located in regulatory regions, the prior that has a non-zero effect size is expected to be higher, which is the key novelty of *bfGWAS* compared to basic GWAS. *bfGWAS* employs a Markov chain Monte Carlo (MCMC) strategy to simultaneously infer the priors for different types of genomic locations and the effect sizes of SNPs. And a parallel inference algorithm is used to scale up the computation.

*Software usage and key parameters of bfGWAS.* *bfGWAS* can be run using the generated analysis Makefile [22]. The key inputs include the genotype file in VCF, PLINK or Tab-Delimited formats, the phenotype file in Tab-Delimited format, and the genomic annotation file (e.g. annotation locations of genes and regulatory

elements) in Tab-Delimited format. The memory usage can also be specified by the user. The major outcomes will include the prior probabilities of being disease-associated for different types of genomic elements, along with the prioritized SNPs associated with the diseases.

### **3.2 TF Binding Imputation and Incorporation with Genetic Variants**

#### **3.2.1 Catchitt Algorithm**

Given the predicted regulatory elements along the human genome, specific TF binding sites within regulatory elements are needed to link TFs to regulatory elements in the regulatory networks. Since ChIP-seq data for most TFs in different cell-types are missing, imputed signals of TF binding are important to further help identify disease-associated non-coding SNPs. Catchitt is one of the top-scoring imputation algorithm for TF binding [23], which mainly relies on features of TF motif hits in DNA sequences and chromatin accessibility signals in the specific cell-type (assayed by DNA-seq or ATAC-seq).

*Model formulation of Catchitt.* To impute TF binding signals for specific cell types, Catchitt takes the combined advantages of DNA sequence motifs and chromatin accessibility. DNA sequence motifs for specific TFs are easy to obtain and can be calculated based on the genomic sequence alone. Chromatin accessibilities are highly cell-type specific and can indicate cell-type specific in-vivo TF binding sites and reduce the number of false positives. To do this, Catchitt first uses the available ChIP-seq data for a specific TF in a certain cell-type, and then trains a logistic regression model of the TF binding ChIP-seq signal, based on features of sequence motif hits, chromatin accessibility, and distances to nearby genes. Based on the trained logistic regression for the specific TF, Catchitt is able to impute the binding signals of the same TF in other cell-types, as long as chromatin accessibility data is available in other cell-types. To further improve the accuracy, Catchitt built an iterative training procedure to fine-tune the coefficients in the logistic regression model along with modifying the imputed signals of TF binding.

*Software usage and key parameters of Catchitt.* To run Catchitt, a series of commands need to be run using `java -jar Catchitt.jar <tool name>`, where the commands identify TF binding peaks of the available ChIP-seq for training, calculate chromatin accessibility signals, score motif matches along the human genome, and iteratively train the logistic regression model. The ChIP-seq file of TF binding can be either narrowPeak or broadPeak formats. The chromatin accessibility file (e.g. DNase-seq) need to be in bigwig format. The TF motifs PWM are from the HOCOMOCO database [30] by default. The major output is the probabilistic scores of TF binding along the human genome in a given cell-type, and therefore we can predict the links between specific TFs and specific enhancers in the re-constructed networks.

### 3.2.2 deltaSVM

#### Algorithm

The benefits of incorporating imputed TF binding signals into disease analysis come in two folds. First, it helps to further prioritize the potentially causal non-coding SNPs that are not only located within regulatory elements but also may disrupt TF binding. Second, based on the aggregated analysis of TF binding for a group of SNPs, it can indicate the potential master TF regulators that are enriched in disease-associated SNPs, providing new insights on disease mechanisms. We recommend to use deltaSVM [24] in the pipeline for this step, an algorithm which was originally developed to discover SNPs that may alter chromatin accessibility levels. In our practice, we found it is also a good tool to identify SNPs that may disrupt TF binding.

*Model formulation of deltaSVM.* deltaSVM aims to identify the discriminative sequence features in regulatory elements that are informative to classify regions with high chromatin accessibility levels (or TF binding) versus regions with low levels, by training a support vector machine (SVM). Given a training ChIP-seq dataset of a TF's genome-wide binding in a specific tissue, deltaSVM first generates genomic regions with high signal levels as positive training samples and regions with low signal levels as negative training samples. Each sample region is then represented as a high-dimensional vector, where each element is a gapped k-mer (i.e. DNA sequence features). Based on the implementation of SVM, deltaSVM is able to find the important k-mers that are predictive for high-levels of TF binding, and furthermore, to predict the level of TF binding for an arbitrary DNA sequence. In this way, deltaSVM predicts the level of TF binding for a specific genomic location based on the reference genomic sequence first, and then predicts the level based on the altered sequence with a specific SNP. The changed predicted TF binding level ('delta') induced by the SNP is then used as a metric to prioritize SNPs that may disrupt specific TF binding in regulatory elements, which have higher probability to be real disease-associated genetic variants. Furthermore, for a group of SNPs from GWAS studies, enrichment analysis can be used to discover specific TFs that are recurrently disrupted by GWAS SNPs, suggesting potential master TF regulators whose regulatory circuitry is substantially affected in diseases.

*Software usage and key parameters of deltaSVM.* deltaSVM can be run using the R package of gkmSVM with commands specifying gkmsvm\_delta. The major inputs include the reference genomic sequences and the altered sequences with genetic variants, both in FASTA formats.

### 3.3 Enhancer-Gene Link Prediction and Identify Disease-Associated Genes

#### 3.3.1 IM-PET Algorithm

Based on the predicted enhancers along the human genome and the TF-enhancer links, the last step in regulatory network re-construction is to identify the downstream target genes regulated by enhancers. As we discussed, enhancer regulation is highly cell-type specific and involves long-range enhancer-promoter interactions mediated by 3D chromatin loops. Therefore, nearest genes in the 1D genome may not be good estimates as target genes for specific enhancers. We recommend using the algorithm IM-PET [25] to link specific enhancers to their distal target genes, based on combined features of enhancer activity, gene expression, TF activity and genomic distances.

*Model formulation of IM-PET.* IM-PET is a supervised algorithm using experimental 3D chromatin map data for training. For a specific cell type, IM-PET considers enhancer-gene pairs that have 3D chromatin interactions as positive training samples and other random enhancer-gene pairs as negative samples. The four features for the prediction are: activity correlations between the enhancer and the gene across multiple cell-types, the correlation of expression between the TF (which has binding sites within the enhancer) and the gene, the co-evolution signal between the sequences of enhancer and gene, and the separation distance between enhancer and gene along the genome. The activity correlation between the enhancer and gene is the most important feature, since the target gene regulated by an enhancer is expected to show coordinated expression dynamics. The feature of distance is also critical, as experimental 3D chromatin interactions show a highly skewed distribution with respect to the distance. IM-PET employs the random forest method to make the predictions, considering the potential non-linear relations among the four features.

*Software usage and key parameters of IM-PET.* IM-PET was written in Perl and the basic command to run it is: perl IM-PET.pl, with several settings of the input files. The input files mainly include the genomic locations of enhancers, enhancer activity values and gene expression levels across different cell-types. The outcome of IM-PET is the re-constructed regulatory network including the long-range links between specific enhancers and genes (*see Note 2*).

#### 3.3.2 ARVIN Algorithm

Based on the re-constructed regulatory networks linking non-coding enhancers to genes, we are finally able to assign the functional interpretations of disease-associated SNPs with respect to their disruptive effects on gene expression and downstream biological pathways. By analyzing the biological functions of the target genes and their protein products (which have largely been annotated for the human genome), information can be obtained about the mechanisms of the non-coding SNPs associated with disease phenotypes. Moreover, aggregating non-coding SNPs located in enhancers can help to provide additional genes that

may be associated with diseases, which are usually missed by traditional protocols that focus only on coding region SNPs. We recommend using a recent algorithm ARVIN [26] to integrate enhancer-gene links with disease genetics.

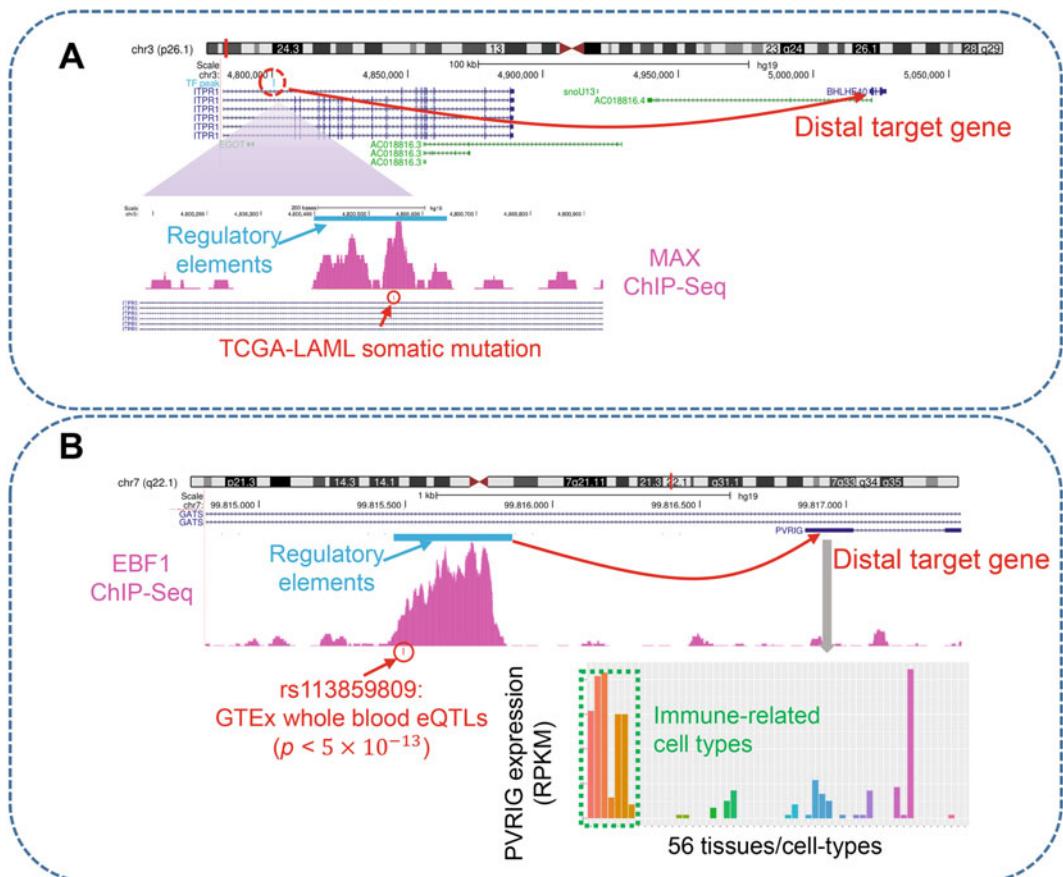
*Model formulation of ARVIN.* ARVIN is an integrative model which incorporates a series of regulatory or functional features of non-coding SNPs and topological signatures of the regulatory networks and gene-gene networks. The key feature in the model is the predicted enhancer-gene links which facilitate the information flow between the non-coding SNPs and the coding genes. One of the basic model assumptions is that if a gene is frequently regulated by enhancers which contain disease-associated SNPs, the gene is more likely to be disease associated also, even if there is no strong evidence from coding SNPs. For features from the local genomic contexts of non-coding SNPs, ARVIN leverages TF-motif disrupting scores induced by the SNPs, the epigenetic and chromatin accessibility signals and other basic genomic sequence information. For the features on the gene side, ARVIN innovatively leverages the gene-gene networks which characterize the functional similarity or association among genes. The assumption is that if a gene is located close to a known disease-gene in the graph neighborhood of the gene-gene network, it is also possibly associated with the disease. ARVIN employs random forest as the modeling framework to combine all these features together to predict novel disease-associated genes and non-coding regulatory SNPs.

*Software usage and key parameters of ARVIN.* To run ARVIN, based on its instructions, the major preparation is to collect all the necessary features for the integrative predictions. The input files include genotype file with SNP information, the predicted enhancer-gene links from IM-PET, the TF-motif disrupting scores, epigenetic signals, chromatin accessibility signals and other genomic sequence features, along with the differential gene expression dataset. The outcomes include the prioritized disease-associated genes and the corresponding non-coding SNPs (*see Note 3*).

### 3.4 Examples Illustrating Results from the Pipeline

Based on the three steps described above, the pipeline will be able to re-construct large-scale networks of cell-type specific regulatory elements, upstream TF to enhancer links and enhancer to downstream target gene links. In the meantime, the pipeline will also effectively integrate different components of the predicted network with genotype/phenotype datasets to reveal in-depth understandings of disease mechanisms. We illustrate what can be learned by running the software introduced earlier with two interesting examples based on our analysis (Fig. 4).

In the first example (Fig. 4a), we prioritized a non-coding somatic mutation as a potentially functional genetic variant, because this SNP is located in a predicted non-coding enhancer element and, more specifically, within the binding peaks of MAX



**Fig. 4** Examples of regulatory network-based analysis of disease genetics data. **(a)** A prioritized non-coding somatic mutation located in an enhancer and MAX binding site, whose candidate target gene is located  $>100$  kb away in the 1D genome. **(b)** A prioritized common genetic variant located in an enhancer and EBF1 binding site, whose target genes shows immune-cell specific expression patterns

within the enhancer, which indicates that the SNP may disrupt MAX binding and inactivate the enhancer element. MAX is a well-known TF involved in blood cancer. Furthermore, the predicted long-range ( $>100$  kb) target gene of the enhancer element is BHLHE40, which has also been shown to be related with cancer. In support of this prediction, the prioritized SNP is indeed a significant genetic variant associated with leukemia from TCGA studies. Similarly, in the second example (Fig. 4b), we prioritized a SNP (rs113859809) which is also located in a predicted enhancer element within an intron of the gene GATS and the predicted target gene is PVRIG, whose expression across diverse cell-types shows highly specific high expression in immune-related cells (T cell and B cells). More specifically, this SNP is located with the peak of EBF1, suggesting that this variant may disrupt EBF1 binding. EBF1 (Early B cell Factor 1) is an important transcription factor

in B cell development. As support of this prediction, this SNP is indeed a significant eQTL in blood tissues based on GTEx project.

These two examples demonstrate that we can substantially expand our view of the underlying biological processes associated with genetic variants in different diseases, including not only the prioritized non-coding variants but also the regulatory mechanisms. These predicted mechanisms will also provide valuable information to guide new experimental designs to further interrogate the genetic basis of complex human diseases.

---

## 4 Notes

### 1. Epigenetic features to predict enhancers

As suggested by the Roadmap Epigenomics project and the ENCODE project, the major histone modifications that can be helpful to predict enhancers and other regulatory elements are: H3K4me1, H3K27me3, H3K4me3, H3K9me3 and H3K36me3. These five histone marks have differential enrichments in enhancers, promoters, actively transcribed gene bodies and repressive domains. And these five marks have the largest coverage across many cell-types or tissues and are widely available to use. Additional useful histone modifications include H3K27ac, which is very informative in predicting cell-type specific active enhancers. Other epigenetic features that are suggested to use are DNA methylations and chromatin accessibility, if their data are available for the specific cell-type or tissue under study.

### 2. Training dataset to predict enhancer-gene links

IM-PET is a type of supervised machine learning software and relies on training data of 3D chromatin interactions. The preferred experimental 3D interaction data to train the model is ChIA-PET datasets, which have better resolution and lower noise levels than other experimental data of 3D chromatin. Considering that ChIA-PET data is not always available for different cell-types or tissues, an optional solution is to leverage Hi-C chromatin interactions data and combine the Hi-C interactions with histone modifications. Hi-C chromatin interactions are more stable across cell-types but they have limitations such as low resolution, high noise levels, and less cell-type specificity. These limitations can be mitigated by overlapping with cell-type specific histone modifications to improve the resolution of interacting fragments (e.g. enhancers), reduce noise, and substantially increase cell-type specificity.

3. Selection of the appropriate cell-type or tissue for the study of a specific disease

As the final goal is to delineate the genetic basis of specific diseases, which are usually associated with multiple cell-types, the training data and machine learning predictions need to be obtained or generated in the matching cell-types. But it is sometimes not clear based on prior biological knowledge which cell-types are associated with the specific disease under study. One strategy of cell-type selection, as suggested by the Roadmap Epigenomics project, is to use the basic GWAS significant SNPs of the specific disease and check the enrichment of overlaps of those SNPs with active enhancers or promoters. If the GWAS SNPs of the disease are globally enriched in enhancers of a subset of cell-types, those cell-types are more likely to be the associated cell-types and can be used in the described pipeline.

## Acknowledgements

Hao Wang, Jiaxin Yang and Jianrong Wang were supported by NIH R01GM131398. The authors would like to thank iCER at MSU for providing the high-performance computing facilities.

## References

1. Lander ES (2011) Initial impact of the sequencing of the human genome. *Nature* 470(7333):187–197. <https://doi.org/10.1038/nature09792>
2. Visscher PM, Wray NR, Zhang Q et al (2017) 10 years of GWAS discovery: biology, function, and translation. *Am J Hum Genet* 101(1):5–22. <https://doi.org/10.1016/j.ajhg.2017.06.005>
3. Tam V, Patel N, Turcotte M et al (2019) Benefits and limitations of genome-wide association studies. *Nat Rev Genet* 20(8):467–484. <https://doi.org/10.1038/s41576-019-0127-1>
4. Do C, Shearer A, Suzuki M et al (2017) Genetic-epigenetic interactions in cis: a major focus in the post-GWAS era. *Genome Biol* 18:120. <https://doi.org/10.1186/s13059-017-1250-y>
5. Gallagher MD, Chen-Plotkin AS (2018) The post-GWAS era: from association to function. *Am J Hum Genet* 102(5):717–730. <https://doi.org/10.1016/j.ajhg.2018.04.002>
6. Hawkins RD, Hon GC, Ren B et al (2010) Next-generation genomics: an integrative approach. *Nat Rev Genet* 11(7):476–486. <https://doi.org/10.1038/nrg2795>
7. Deplancke B, Alpern D, Gardeux V et al (2016) The genetics of transcription factor DNA binding variation. *Cell* 166(3):538–554. <https://doi.org/10.1016/j.cell.2016.07.012>
8. Watanabe K, Taskesen E, Bochoven A et al (2017) Functional mapping and annotation of genetic associations with FUMA. *Nat Commun* 8:1826. <https://doi.org/10.1038/s41467-017-01261-5>
9. Schaub MA, Boyle AP, Kundaje A et al (2012) Linking disease associations with regulatory information in the human genome. *Genome Res* 22(9):1748–1759. <https://doi.org/10.1101/gr.136127.111>
10. Shlyueva D, Stampfel G, Stark A et al (2014) Transcriptional enhancers: from properties to genome-wide predictions. *Nat Rev Genet* 15(4):272–286. <https://doi.org/10.1038/nrg3682>
11. Creyghton MP, Cheng AW, Wehsted GG et al (2010) Histone H3K27ac separates active from poised enhancers and predicts developmental state. *Proc Natl Acad Sci U S A* 107

- (50):21931–21936. <https://doi.org/10.1073/pnas.1016071107>
12. Kimura H (2013) Histone modifications for human epigenome analysis. *J Hum Genet* 58(7):439–445. <https://doi.org/10.1038/jhg.2013.66>
  13. Lister R, Pelizzola M, Dowen RH et al (2009) Human DNA methylomes at base resolution show widespread epigenomic differences. *Nature* 462(7271):315–322. <https://doi.org/10.1038/nature08514>
  14. Zhou VW, Goren A, Bernstein BE (2011) Charting histone modifications and the functional organization of mammalian genomes. *Nat Rev Genet* 12(1):7–18. <https://doi.org/10.1038/nrg2905>
  15. Schoenfelder S, Fraser P (2019) Long-range enhancer-promoter contacts in gene expression control. *Nat Rev Genet* 20(8):437–455. <https://doi.org/10.1038/s41576-019-0128-0>
  16. Heintzman ND, Hon GC, Hawkins RD et al (2009) Histone modifications at human enhancers reflect global cell-type-specific gene expression. *Nature* 459(7243):108–112. <https://doi.org/10.1038/nature07829>
  17. ENCODE Project Consortium (2012) An integrated encyclopedia of DNA elements in the human genome. *Nature* 489(7414):57–74. <https://doi.org/10.1038/nature11247>
  18. Roadmap Epigenomics Consortium (2015) Integrative analysis of 111 reference human epigenomes. *Nature* 518(7539):317–330. <https://doi.org/10.1038/nature14248>
  19. Valencia AM, Kadoch C (2019) Chromatin regulatory mechanisms and therapeutic opportunities in cancer. *Nat Cell Biol* 21(2):152–161. <https://doi.org/10.1038/s41556-018-0258-1>
  20. Kim K, Jang K, Yang W et al (2016) Chromatin structure-based prediction of recurrent non-coding mutations in cancer. *Nat Genet* 48(11):1321–1326. <https://doi.org/10.1038/ng.3682>
  21. Ernst J, Kellis M (2012) ChromHMM: automating chromatin-state discovery and characterization. *Nat Methods* 9(3):215–216. <https://doi.org/10.1038/nmeth.1906>
  22. Yang JJ, Fritzsche LG, Zhou X et al (2017) A scalable Bayesian method for integrating functional information in genome-wide association studies. *Am J Hum Genet* 101(3):404–416. <https://doi.org/10.1016/j.ajhg.2017.08.002>
  23. Keilwagen J, POSCH S, Grau J (2019) Accurate prediction of cell type-specific transcription factor binding. *Genome Biol* 20:9. <https://doi.org/10.1186/s13059-018-1614-y>
  24. Lee D, Gorkin DU, Baker M et al (2015) A method to predict the impact of regulatory variants from DNA sequence. *Nat Genet* 47(8):955. <https://doi.org/10.1038/ng.3331>
  25. He B, Chen C, Teng L et al (2014) Global view of enhancer-promoter interactome in human cells. *Proc Natl Acad Sci U S A* 111(21):E2191–E2199. <https://doi.org/10.1073/pnas.1320308111>
  26. Gao L, Uzun Y, Gao P et al (2018) Identifying noncoding risk variants using disease-relevant gene regulatory networks. *Nat Commun* 9:702. <https://doi.org/10.1038/s41467-018-03133-y>
  27. Lonsdale J, Thomas J, Salvatore M et al (2013) The genotype-tissue expression (GTEx) project. *Nat Genet* 45(6):580–585. <https://doi.org/10.1038/ng.2653>
  28. Kheradpour P, Kellis M (2014) Systematic discovery and characterization of regulatory motifs in ENCODE TF binding experiments. *Nucleic Acids Res* 42(5):2976–2987. <https://doi.org/10.1093/nar/gkt1249>
  29. Wang YL, Song F, Zhang B et al (2018) The 3D genome browser: a web-based browser for visualizing 3D genome organization and long-range chromatin interactions. *Genome Biol* 19:151. <https://doi.org/10.1186/s13059-018-1519-9>
  30. Kulakovskiy IV, Medvedeva YA, Schaefer U et al (2013) HOCOMOCO: a comprehensive collection of human transcription factor binding sites models. *Nucleic Acids Res* 41(D1):D195–D202. <https://doi.org/10.1093/nar/gks1089>



# Chapter 12

## Predicting Host Phenotype Based on Gut Microbiome Using a Convolutional Neural Network Approach

Derek Reiman, Ali M. Farhat, and Yang Dai

### Abstract

Accurate prediction of the host phenotypes from a microbial sample and identification of the associated microbial markers are important in understanding the impact of the microbiome on the pathogenesis and progression of various diseases within the host. A deep learning tool, PopPhy-CNN, has been developed for the task of predicting host phenotypes using a convolutional neural network (CNN). By representing samples as annotated taxonomic trees and further representing these trees as matrices, PopPhy-CNN utilizes the CNN's innate ability to explore locally similar microbes on the taxonomic tree. Furthermore, PopPhy-CNN can be used to evaluate the importance of each taxon in the prediction of host status. Here, we describe the underlying methodology, architecture, and core utility of PopPhy-CNN. We also demonstrate the use of PopPhy-CNN on a microbial dataset.

**Key words** Convolutional neural network, Deep learning, Microbial taxonomic abundance, Predict host phenotype

---

## 1 Introduction

### 1.1 PopPhy CNN Network

Metagenomic studies of the gut microbiome have linked dysbiosis of microbiome to many host diseases [1, 2]. A microbiome sample is usually described by a relative abundance profile of microbial taxa at one of the taxonomic levels. Gut microbiome data from a case-control study allows for the identification of association between bacterial taxa and the host phenotype by performing statistical tests [3]. The alternative approach to exploring gut microbiome data is the establishment of a predictive model for host phenotype, which in turn can facilitate the investigation of microbiome–host interactions. There have been a variety of studies evaluating the performance of predictive models using standard machine learning (ML) models such as random forests (RFs), least absolute shrinkage and selection operator (Lasso), and support vector machines (SVMs), among others [4, 5]. More recently, deep neural network (DNN) learning models (e.g., multilayer neural networks

[MLNNs] and convolutional neural networks [CNNs]) were investigated for their ability to exploit complex interactions between bacteria and host phenotypes [6–10]. The performance evaluation demonstrates the competitiveness of the DNN models. However, their potentials need to be comprehensively evaluated using large-scale datasets as they become available in the future. Therefore, making user-friendly DNN-based tools available to laboratory investigators is important.

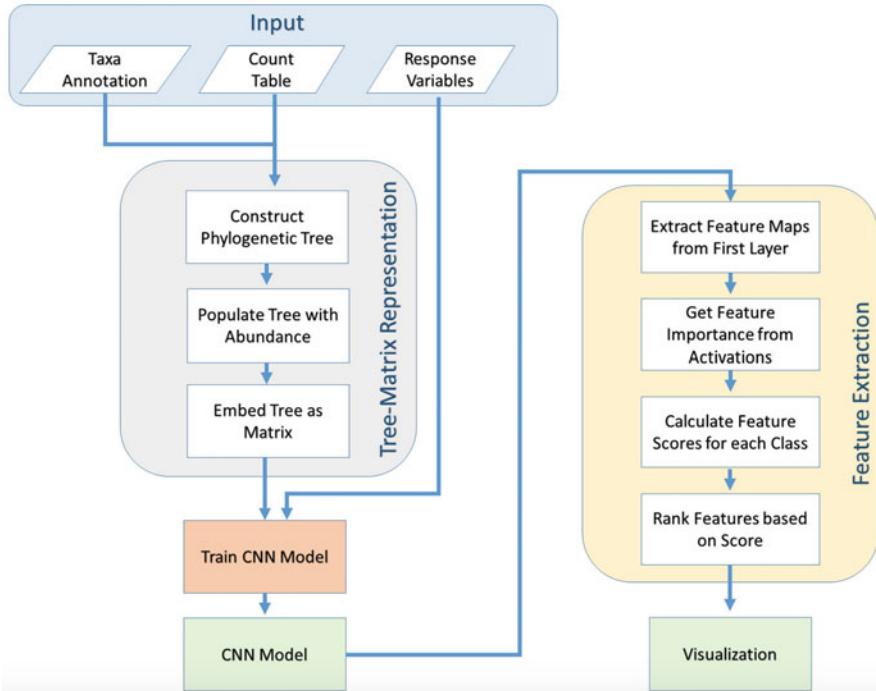
Training and testing using the standard ML models have become relatively straightforward given the availability of open source libraries such as *PyTorch* (<https://pytorch.org/>) and *scikit-learning* (<https://scikit-learn.org/stable/>), to name a couple. However, the development of predictive models from microbiome data using DNNs may present challenges to inexperienced users because the setup for model training requires the specification for many parameters related to the architecture of a neural network. In addition, in order to empower deep learning models, the relative abundance profiles of microbial taxa in a microbiome dataset are often required to be transformed into profiles using phylogenetic or taxonomic trees to better represent the relationship among the taxa originally observed as features [7–9, 11]. To facilitate the use of DNN models to explore complex structures in microbiome data, we will use our developed tool PopPhy-CNN [7, 8] to demonstrate the steps of obtaining a predictive model. Since omics studies using microbiome, metabolome, transcriptome, and other genetic and epigenetic profiles of a subject will become indispensable to human disease study [12], and given that DNNs are ideal modeling tools to integrate these omics data [13, 14], we believe that the protocol described in this chapter will be useful to users who begin to use DNNs to explore their omics data.

## **1.2 PopPhy-CNN Framework**

PopPhy-CNN uses convolutional neural networks to generate robust prediction models by utilizing the taxonomic relationship among taxa in a microbiome dataset [7, 8]. In addition to establishing a prediction model, PopPhy-CNN also facilitates the identification of microbial taxa that may be associated with the phenotype from the established prediction model. PopPhy-CNN takes the relative abundance profile of microbial taxa for each subject as input; thus a minimum level of effort is required to build models using DNNs. In the remainder of this section, we will briefly describe the framework of PopPhy-CNN as shown in Fig. 1.

## **1.3 Learning Model in PopPhy-CNN**

PopPhy-CNN takes advantage of the superior ability of CNN in generating convolution layers with multiple feature maps that capture the spatial information in training data. CNNs are a class of deep learning models that consider groups of local features in ML tasks. They have been effective in image processing as well as natural language processing. Since a microbiome profile is usually

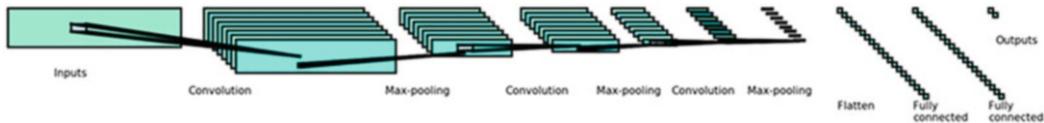


**Fig. 1** PopPhy-CNN flowchart from Reiman et al. [8]

represented by a vector of relative microbial taxa abundances in arbitrary orders, it is necessary to convert this information into a biological structure. To this end, we construct a taxonomic tree to preserve the relationship among the microbial taxa in the profiles. For a microbiome sample, the tree is then populated with the observed relative abundance of microbial taxa, and the populated tree is represented in a 2D matrix. This constructed matrix provides better spatial and quantitative information in the metagenomic data, which are more suitable to CNNs to learn in comparison to the vector of relative microbial taxa abundances in an arbitrary order. Once the relative abundance profiles of all samples in a dataset have been converted to the matrix format, they are ready to be used as input to a standard CNN architecture to train the predictive model.

A CNN architecture usually contains one or more convolutional layers, in which a set of kernels is used to capture various signals and generate a set of feature maps. This is usually followed with at least one fully connected hidden layer, followed lastly by the output layer. A generic structure of a CNN is shown in Fig. 2. We will describe in detail how to set up the CNN architecture in Subheading 3.

Given an input matrix  $M$ , a kernel  $k$  with a set of weights  $W^{(k)} \in R^{r \times s}$ , the velocity at position  $(i,j)$  is defined as:



**Fig. 2** A generic CNN architecture

$$vel^{(k)}(i, j) = \sum_{r=0}^m \sum_{s=0}^n M(i+r, j+s) \times W^{(k)}(m-r, n-s) \quad (1)$$

This velocity value is then passed to a nonlinear activation function and pooling is performed over the entire set. In our method, we use max pooling and the ReLU activation function for convolutional and fully connected layers. The ReLU activation function,  $\text{ReLU}(x) = \max\{0, x\}$  is chosen since it has been shown to reduce training time while still maintaining the nonlinearity provided by other activation functions [15]. In the output layer, we use a softmax activation in order to predict the class probabilities, assigning the predicted class to the highest probability.

Cross-entropy, defined as  $-\sum_c p_c \ln(\alpha_c)$ , is used in our cost function to ensure that the distribution of the output class resembles that of the input class. Here  $p_c$  is the true probability that the sample belongs to class  $c$  and  $\alpha_c$  is the predicted probability that the sample belongs to class  $c$  using the softmax activation, that is,  $\text{softmax}_c(\vec{x}) = e^{x_c} / \sum_{j \in C} e^{x_j}$ , where  $C$  represents the set of classes. When the output is categorical, the cross-entropy becomes identical to the negative log loss since all but one  $p_c$  will be 0, with  $p_c$ , where  $c$  is the true class, being equal to 1. This makes the loss of a single input as follows:

$$C' = -\ln(\alpha_c). \quad (2)$$

The model will be trained using the stochastic gradient descent (SGD) algorithm with the cost of a sample with class  $c$  defined as follows:

$$C = -\frac{n_{\text{total}}}{2 \times n_c} \ln(\alpha_c) + \lambda \sum_l \|W_l\|_2 \quad (3)$$

where  $n_{\text{total}}$  is the total number of samples;  $n_c$  is the number of samples with class  $c$ ,  $W_l$  is the weight matrix connecting layers  $l$  and  $l-1$ , and  $\lambda$  is a regularization penalty parameter for an L2-normalization of the network weights.

#### 1.4 Feature Ranking in PopPhy-CNN

Evaluating features from a trained CNN model is another utility of PopPhy-CNN. PopPhy-CNN will provide a list of important microbial taxa that are predictive to the host phenotype using a novel procedure to score and rank the features [8]. Our approach is different from the statistical tests and other regression-based

methods for microbiome-wide association studies (MWASs) [3, 16–19]. The design of PopPhy-CNN is motivated by the findings that a microbial signature for the host phenotype may be complex, involving simultaneous over- and underrepresentations of multiple microbial taxa from different taxonomic levels and potentially interacting with each other [5, 20]. Since CNNs are excellent in identifying local features that are useful for prediction, we expect that PopPhy-CNN not only can extract taxa at the leaf level but also taxa at the internal node of a branch of the taxonomic tree. The former by themselves alone may have weak association, and thus may be missed by other MWAS methods.

Our scoring scheme extracts learned features from the map activations in the first convolutional layer prior to subsampling. This allows us to evaluate which positions in the input contribute most to the highest activations in the learned CNNs. Briefly, we extract the feature maps generated by each training sample for a given class. Then, for each kernel, we select a subset of taxa consisting of the highest signals in each sample’s generated feature map, giving a set of features per sample. We then count the number of times a taxon was seen across all samples, keeping only the set of taxa that have been observed above a given frequency. Note that both the threshold for obtaining the subset of highest signals and the threshold for keeping taxa observed above a given frequency are customizable.

Once the set of taxa to be evaluated has been obtained, we calculate each taxon’s score by using the kernel to calculate how much the taxon contributed to the magnitude of the signal’s value, resulting in a value between  $-1$  and  $1$ . This is repeated for each kernel, and a taxon’s final score with respect to a given class is taken as the maximum observed score. The process is then repeated for each class value observed in the dataset. More details about the scoring procedure can be found in [8].

## 2 Materials

In this section, we describe the general requirement on input and computational environment of PopPhy-CNN, which can be downloaded from <https://github.com/YDaiLab/PopPhy-CNN>.

### 2.1 Collected Microbiome Data

PopPhy-CNN requires two kinds of data from individuals: (1) microbial population abundance within specific taxonomic groups identified from the metagenomic study and (2) their respective phenotypic descriptions. Experimentally, several methods are available to obtain metagenomic sequence data, such as 16S rRNA sequencing and whole metagenome shotgun sequencing as reviewed by Thomas et al. [21]. Furthermore, these measurements require additional analysis using computational tools to quantify

the abundance within each taxonomic group. There are several available tools for this analysis, but freely available software such as “QIIME” for 16S rRNA sequencing data [22] and “MG-RAST” for shotgun sequence data [23] have proven convenient and efficient.

## **2.2 Required Computational Tools for PopPhy-CNN**

A working python environment is required to properly train a PopPhy-CNN model using a microbiome dataset. There are several methods to run the tool, and it is up to the user’s preferred choice. The first method is by running *train.py* from the command line. This script will use the configuration file and perform model training with the tenfold cross-validation, feature extraction, and benchmarking against other machine learning models. Model evaluations, taxonomic feature scores, and network files are saved in the results directory. The second method is by using individual components of PopPhy-CNN in the python environment. For this we recommend the use of the Conda python environment manager, such as Anaconda (<https://www.anaconda.com/>), and Jupyter Notebooks to better visualize and experiment with the training and feature extraction procedures. Python dependencies can be imported into a Conda environment using the *PopPhy.yml* file. In addition, the user needs to install TensorFlow version 1.14 ([www.tensorflow.org](http://www.tensorflow.org)). To speed the training process on GPU equipped machines, GPU toolkits such as NVIDIA’s CUDA and cuDNN should also be installed. To visualize the output JSON files upon successfully running the model, the software Cytoscape [24] can be used to draw the phylogenetic tree with nodes, edges, and colors based on the calculated importance scores.

---

## **3 Methods**

There are three main steps in running PopPhy-CNN as shown in Fig. 1: (1) preparing the matrix input format from a phylogenetic tree structure to a two-dimensional matrix arrangement, (2) training the CNN architecture, and (3) evaluating and visualizing taxonomic features.

### **3.1 Preparing the Matrix Input**

#### **3.1.1 Input Format**

The user needs to ensure the data files are set up in a proper format. PopPhy-CNN requires two files. The first is a tab separated file where each row represents a taxonomic feature and each column represents a sample. There should be no header included in this file, and the first column of each row should contain the taxonomic label for that feature. The taxonomic label should be structured in a specific way. An example for *Actinomyces graevenitzii* is shown below:

k\_\_Bacteria|p\_\_Actinobacteria|c\_\_Actinobacteria|o\_\_Actinomycetales|f\_\_Actinomycetaceae|g\_\_Actinomyces|s\_\_Actinomyces \_ graevenitziihere “k” represents kingdom, “p” represents phylum, “c” represents class, “o” represents order, “f” represents family, “g” represents genus, and “s” represents genus and species pair. This file should be named *abundance.tsv*.

The second file should contain class values relevant to the prediction task. Again, there should be no header and the file should be row-matched to the abundance file. This file should be named *labels.txt*.

Both files need to be placed in a subdirectory within the *data/* directory. PopPhy-CNN will then run on this dataset when the name of the subdirectory is assigned to the *DataSet* flag in *config.py*. Sample data files are provided in the repository’s data folder.

### 3.1.2 Preprocessing Data

Using the Jupyter Notebook feature from JupyterLab, the first step is to read in the configuration file. This file contains all the customizable parameters as well as the directory name for the dataset to be analyzed and most of the functions in PopPhy-CNN use this configuration object as an input.

```
config = get_config()
```

The next step is to call the function *prepare\_data()* from the *utils.prepare\_data* module. The inputs to this function are the path to the dataset directory and the configuration object. In this function, the data is converted to relative abundance if it is not already in that format. Then lowly abundant microbes are filtered out based on a user specified threshold and the number of samples each microbe is present in.

```
my_maps, raw_x, tree_x, raw_features, tree_features, labels, label_set, g, feature_df = prepare_data(path, config)
```

### 3.1.3 Generating Matrix Representations of Populated Taxonomic Tree

Next in the *prepare\_data()* function, the taxonomic tree is built using the node and edge connections previously known from the biological tree of life (*see Note 1*). Then, for each sample, we assign the abundance values of the observed taxonomic features to their respective nodes and populate the entire tree by augmenting any node with children by the sum of its children’s abundance values. The sample’s populated tree is then represented as a matrix through a simple embedding process [8]. These matrices are returned by *prepare\_data()*, as well as the set of original taxonomic vectors and the set of vectorized trees, which may be used for benchmarking. The samples are in the same order across these three sets.

### 3.2 Training the CNN Model

#### 3.2.1 Constructing the CNN Architecture

```
popphy_model = PopPhyCNN((input_height, input_width), num_class, config)
```

In order to run PopPhy, the user must first construct a *PopPhyCNN()* model. The model takes the input matrix size, the number of classes, and the configuration object as inputs.

#### 3.2.2 Normalizing the Data

Before training a model, the data is log-transformed and then normalized using min–max normalization for each feature across the samples:

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This maps each of the features to be between 0 and 1, enforcing that each feature is positive. This assumption is important for PopPhy-CNN’s feature evaluation protocol.

#### 3.2.3 Training a PopPhy-CNN Model

Calling the *train()* function on the PopPhy-CNN model will perform the training of the CNN model. This function takes two inputs. The first input, which is required, is the set of data consisting of matrices and true classes. The labels first need to be converted to one-hot encoding (*see Note 2*). The second input, which is optional, is a vector of sample weights.

```
popphy_model.train(train, train_weights)
```

During training, PopPhy-CNN learns the weights across the neural network’s nodes and makes prediction based on those weights. The model is trained using early stopping using 10% of the training set as a held-out validation set. Once the performance of the validation set has stopped improving, the training ends and the weights of the model that performed best on the validation set is used. The model can then be evaluated using the *test()* function. This function takes as an input a paired dataset, similar to the *train()* function, where the set contains matrices and true classes.

```
preds, stats = popphy_model.test(test)
```

The *test()* function returns the predicted values of the test set, as well as a dictionary of evaluation metrics that include the area under the receiver operating curve (AUC) for binary predictions, Matthew’s correlation coefficient (MCC), precision, recall, and F1 score (*see Note 3*).

### 3.3 Evaluating and Visualizing Taxonomic Features

#### 3.3.1 Scoring Taxonomic Features

Once a PopPhy-CNN model has been trained, the scores for each feature can be obtained using the model's `get_feature_scores()` function. This function takes in the dataset to be used (here shown to be the training set), the graph object, the set of possible classes, the set of node labels in the tree, and the configuration file.

```
scores = popphy_model.get_feature_scores(train, g, label_set, tree_features, config)
```

The function returns a matrix of scores where the rows follow the same order as the feature list provided and the columns follow the same order as the set of labels provided. These scores should then be stored in a dictionary of DataFrames. In this dictionary, there should be a key for each possible class. The value for each key should be the DataFrame of scores for that class where each row is a taxonomic feature and each column is a separate run of evaluations. This allows PopPhy-CNN to evaluate scores over multiple iterations such as in cross-validation evaluation.

#### 3.3.2 Generating the Network File

The `generate_network()` function is used to obtain both a network dictionary as well as a final set of taxonomic scores. This function requires the graph object and set of class labels as input, as well as the dictionary of scores as described in Subheading 3.3.1.

```
network, tree_scores = generate_network(g, feature_scores, label_set)
```

In order to obtain a file that is usable for visualization in Cytoscape, the `network` variable needs to be converted to JSON and saved. The JSON file can then be imported into Cytoscape for viewing and colored using the style XML provided in the `cytoscape_style/` directory of the GitHub repository. A more explicit example of this is provided in the case study below.

## 4 A Demonstration Using a Cirrhosis Dataset

The cirrhosis dataset was taken from a study of 114 cirrhosis patients and 118 healthy subjects. After the preprocessing 269 taxa were retained at the species level [8]. The dataset can be obtained from our GitHub repository.

#### 4.1 Using PopPhy-CNN

The configuration file is read in using `get_config()` and the evaluation parameters are stored as their own variables for later use. A directory is then set up to store results and evaluations.

```

config = get_config()
filt_thresh = config.get('Evaluation', 'FilterThresh')
dataset = config.get('Evaluation', 'DataSet')
num_runs = int(config.get('Evaluation', 'NumberRuns'))
num_test = int(config.get('Evaluation', 'NumberTestSplits'))
path = "../data/" + dataset
results_dir = "../results/notebook_results/" + dataset

try:
    os.makedirs(results_dir)
except OSError:
    print ("Creation of the directory %s failed" % results_dir)
else:
    print ("Successfully created the directory %s" % results_dir)

Successfully created the directory ../results/notebook_results/Cirrhosis

```

Next, *prepare\_data()* is called in order to generate the tree structure as well as obtain the matrices representing the populated trees. The number of total class values from the dataset is calculated to decide AUC or MCC to be used for evaluation. Next, the samples and observed classes are shuffled. The detected classes are then converted to a one-hot encoding format. Lastly, a list of random seeds is generated based on the number of times to perform cross-validation.

```

my_maps, raw_x, tree_x, raw_features, tree_features, labels, label_set, g, feature_df = prepare_data(path, config)

num_class = len(np.unique(labels))
if num_class == 2:
    metric = "AUC"
else:
    metric = "MCC"

seed = np.random.randint(100)
np.random.seed(seed)
np.random.shuffle(my_maps)
np.random.seed(seed)
np.random.shuffle(raw_x)
np.random.seed(seed)
np.random.shuffle(tree_x)
np.random.seed(seed)
np.random.shuffle(labels)

n_values = np.max(labels) + 1
labels_oh = np.eye(n_values)[labels]

tree_row = my_maps.shape[1]
tree_col = my_maps.shape[2]

print("There are %d classes...%s" % (num_class, ", ".join(label_set)))
cv_list = ["Run_" + str(x) + "_CV_" + str(y) for x in range(num_runs) for y in range(num_test)]
seeds = np.random.randint(1000, size=num_runs)

```

```

There are 269 raw features...
Building tree structure...
Found tree file...
Populating trees...
There are 479 tree features...
There are 2 classes...n, cirrhosis

```

Now that the *prepare\_data()* has finished, a DataFrame to hold evaluation metrics is constructed. In addition, the dictionary for the taxonomic feature scores is constructed within each class value as a key, a DataFrame is created using the taxonomic feature labels for the row indices.

```

popphy_stat_df = pd.DataFrame(index=["AUC", "MCC", "Precision", "Recall", "F1"], columns=cv_list)

feature_scores = {}

for l in label_set:
    feature_scores[l] = pd.DataFrame(index=tree_features)

```

Next we will run a cross-validation procedure. For each random seed previously generated, we partition the dataset into a set of partitions stratified by the observed class response (here we perform tenfold cross-validation). Training sets and test sets are constructed, and data are log transformed. We then use min–max normalization on each feature across the samples so that each feature is in the range from 0 and 1. Lastly, we group the training abundance values with their observed class values and the testing abundance values with their observed class values.

```
run = 0
for seed in seeds:
    skf = StratifiedKFold(n_splits=num_test, shuffle=True, random_state=seed)
    fold = 0
    for train_index, test_index in skf.split(my_maps, labels):
        train_x, test_x = my_maps[train_index,:,:], my_maps[test_index,:,:]
        train_y, test_y = labels_oh[train_index,:], labels_oh[test_index,:]

        train_x = np.log(train_x + 1)
        test_x = np.log(test_x + 1)

        scaler = MinMaxScaler().fit(train_x.reshape(-1, tree_row * tree_col))
        train_x = np.clip(scaler.transform(train_x.reshape(-1, tree_row * tree_col)),
                          0, 1).reshape(-1, tree_row, tree_col)
        test_x = np.clip(scaler.transform(test_x.reshape(-1, tree_row * tree_col)),
                         0, 1).reshape(-1, tree_row, tree_col)

    train = [train_x, train_y]
    test = [test_x, test_y]
```

In order to handle any class imbalance, a set of sample weights is generated based on the number of times it is observed in the training set. These are calculated by dividing the number of observed samples with a specific class by twice the total number of samples.

```
c_prob = [0] * len(np.unique(labels))
train_weights = []

for l in np.unique(labels):
    a = float(len(labels))
    b = 2.0 * float((np.sum(labels==l)))
    c_prob[int(l)] = a/b

c_prob = np.array(c_prob).reshape(-1)

for l in np.argmax(train_y, 1):
    train_weights.append(c_prob[int(l)])
train_weights = np.array(train_weights)
```

With the normalized data and a set of sample weights, a PopPhy-CNN model is ready to be trained. A model object is constructed using the input shape, number of classes, and configuration parameters. The model is then trained using the training set and the sample weights, and then it is evaluated using the testing set. The evaluation metrics returned by *test()* are then stored in the evaluation DataFrame, where here the column is labeled based on the iteration number and cross-validated fold number. The feature scores are then calculated using *get\_feature\_scores()* and stored in the feature evaluation dictionary under their respective class label. The model is then destroyed, and the next cross-validated

partitioning is run, where the same process of training, evaluating, and scoring features is performed.

```
popphy_model = PopPhyCNN((tree_row, tree_col), num_class, config)
popphy_model.train(train, train_weights)
preds, stats = popphy_model.test(test)
if num_class == 2:
    popphy_stat_df.loc["AUC"]["Run_" + str(run) + "_CV_" + str(fold)] = stats["AUC"]
    popphy_stat_df.loc["MCC"]["Run_" + str(run) + "_CV_" + str(fold)] = stats["MCC"]
    popphy_stat_df.loc["Precision"]["Run_" + str(run) + "_CV_" + str(fold)] = stats["Precision"]
    popphy_stat_df.loc["Recall"]["Run_" + str(run) + "_CV_" + str(fold)] = stats["Recall"]
    popphy_stat_df.loc["F1"]["Run_" + str(run) + "_CV_" + str(fold)] = stats["F1"]

scores = popphy_model.get_feature_scores(train, g, label_set, tree_features, config)
for l in range(len(label_set)):
    score_list = scores[:, l]
    lab = label_set[l]
    feature_scores[lab]["Run_" + str(run) + "_CV_" + str(fold)] = score_list

popphy_model.destroy()
fold += 1
run += 1
```

After cross-validation, we can observe and save the set of evaluations. Below we show and save both the set of evaluations per each iteration as well as the set of mean evaluation metrics across all the training iterations.

```
popphy_stat_df.to_csv(results_dir + "/popphy_evaluation.csv")
popphy_stat_df
```

	Run_0_CV_0	Run_0_CV_1	Run_0_CV_2	Run_0_CV_3	Run_0_CV_4	Run_0_CV_5	Run_0_CV_6	Run_0_CV_7	Run_0_CV_8	Run_0_CV_9
AUC	0.972222	0.777778	0.979167	0.909722	0.833333	0.984848	0.871212	1	0.92562	0.85124
MCC	0.707107	0.338062	0.774597	0.53033	0.516459	0.825758	0.568182	1	0.730297	0.547723
Precision	0.875	0.671429	0.9	0.78125	0.775362	0.913043	0.785573	1	0.866667	0.775
Recall	0.833333	0.666667	0.875	0.75	0.73913	0.913043	0.782609	1	0.863636	0.772727
F1	0.828571	0.664336	0.873016	0.742857	0.73311	0.913043	0.782609	1	0.863354	0.772257

```
popphy_stat_df.mean(1).to_csv(results_dir + "/popphy_mean_evaluation.csv")
popphy_stat_df.mean(1)
```

```
AUC      0.910514
MCC      0.653851
Precision 0.834332
Recall   0.819615
F1       0.817315
dtype: float64
```

In addition, a final predictive model is generated using the entire dataset. Again, the data is log transformed and min–max normalized. A set of training sample weights is generated in a similar fashion as previously shown. A model is constructed and trained using the entire dataset and the trained model is saved for later use.

```
network, tree_scores = generate_network(g, feature_scores, label_set)

with open(results_dir + '/network.json', 'w') as json_file:
    json.dump(network, json_file, sort_keys=True, indent=4, separators=(',', ': '))
tree_scores.to_csv(results_dir + '/feature_scores.csv')
```

Lastly, we generate the network dictionary and tree scores. The tree scores are saved as a table in our results directory. The network dictionary is converted to JSON format and saved in order to be imported into Cytoscape.

```

log_x = np.log(my_maps + 1)
norm_x = MinMaxScaler().fit_transform(log_x.reshape(-1, tree_row * tree_col)).reshape(-1, tree_row, tree_col)
train = [norm_x, labels_oh]

train_weights = []

for l in np.unique(labels):
    a = float(len(labels))
    b = 2.0 * float((np.sum(labels==l)))
    c_prob[int(l)] = a/b

c_prob = np.array(c_prob).reshape(-1)

for l in np.argmax(labels_oh, 1):
    train_weights.append(c_prob[int(l)])
train_weights = np.array(train_weights)
popphy_model = PopPhyCNN((tree_row, tree_col), num_class, config)
popphy_model.train(train, train_weights)
popphy_model.model.save(results_dir + '/PopPhy-CNN.h5')

```

## 4.2 The Ranked List of Microbial Taxa

Now that the features have been evaluated, we can rank them to see which features the model found to be important. Here, we rank the features by their median score within each DataFrame separately. We then display the top 20 features found important for determining both the healthy state and disease state shown in Fig. 3.

```

from IPython.display import display, HTML
CSS = """
.output{
    flex-direction: row;
}
"""

df1 = feature_scores['n'].median(1).sort_values(ascending=False).to_frame().rename(\n    columns={0:"Score for Healthy State"}).head(20)
df2 = feature_scores['cirrhosis'].median(1).sort_values(ascending=False).to_frame().rename(\n    columns={0:"Score for Disease State"}).head(20)
display(df1)
display(df2)
HTML('<style>{}</style>'.format(CSS))

```

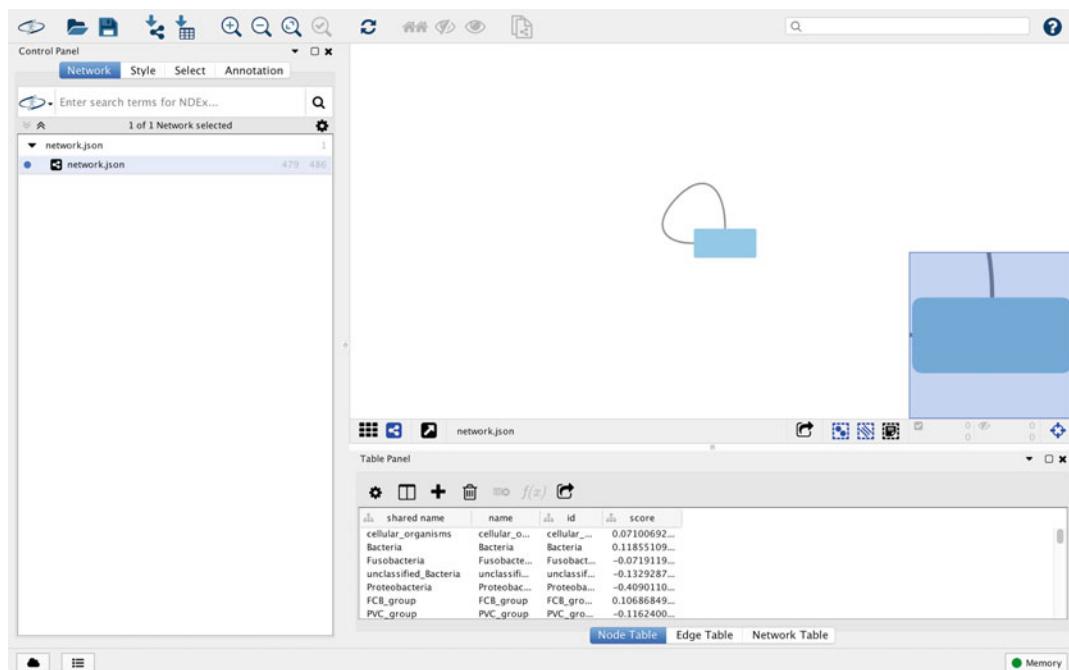
## 4.3 Visualizing Feature Scores

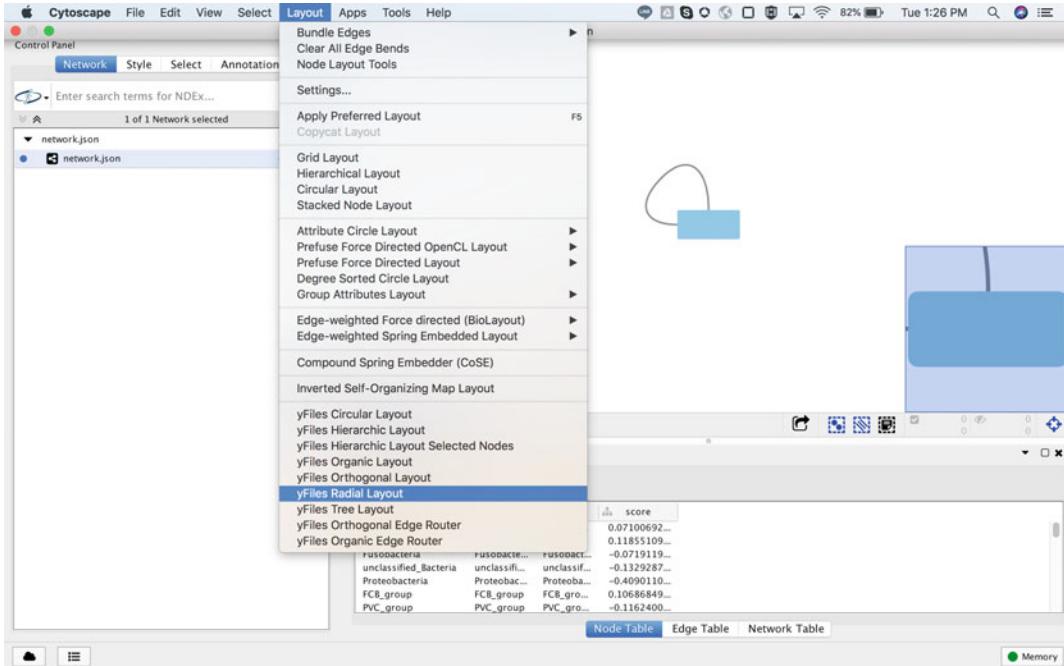
In order to visualize how each feature impacts health status, we recommend using the Cytoscape desktop application. Using Cytoscape, you can quickly import the network file, which we have saved as *network.json*. This will import the nodes and edges, as well as their respective score values, into Cytoscape and visualize it as shown in Figs. 4 and 5.

The default view has all the nodes and edges stacked on top of each other. Therefore, you must apply a layout style in order to be able to view the tree structure. In our experiences, we found that the radial layout from the *yFiles* layouts provided the best visualization for the tree structures. Since *yFiles* is not a default part of the Cytoscape installation, you may need to download the extension if you wish to use their layouts.

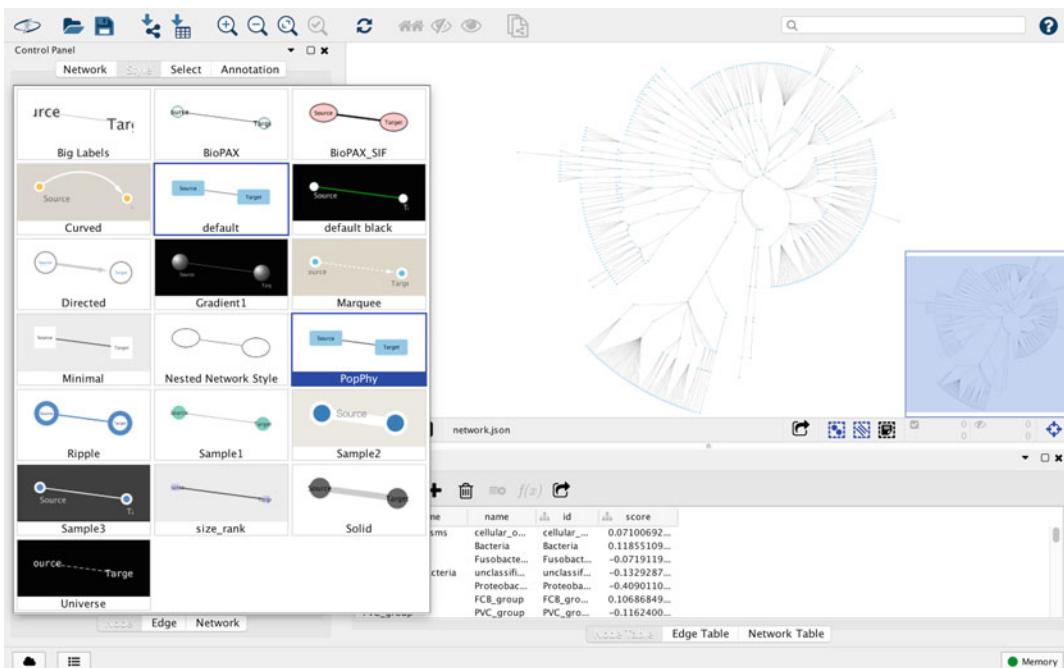
After the tree has a desirable layout, it is time to visualize the scores across the tree. We have provided a style sheet in the *cytoscape\_style* folder of our repository. For the desktop application of Cytoscape, you need to import the XML style sheet. Once it has been imported, you can select the PopPhy style from the style tab as shown in Fig. 6.

	Score for Healthy State		Score for Disease State
unclassified_Oscillibacter_species	0.466153	Bacilli	0.475100
Oscillibacter	0.424628	Streptococcus_anginosus_group	0.434224
Oscillospiraceae	0.398589	Actinomycetales	0.414761
Eubacterium_hallii	0.368270	Veillonellales	0.414649
Adlercreutzia_equalifaciens	0.341142	Lactobacillales	0.413934
unclassified_Megamonas_species	0.334842	Actinomyces	0.412720
Adlercreutzia	0.324558	Actinomycetaceae	0.412720
Collinsella	0.311867	Streptococcus_gordonii	0.401754
Coriobacteriaceae	0.310055	Megasphaera_micronutriformis	0.396978
Eubacterium	0.308148	Rothia	0.374707
Eggerthellaceae	0.292280	Rothia_mucilaginosa	0.367539
Selenomonadaceae	0.291965	Micrococcaceae	0.365119
Holdemania	0.288940	Micrococcales	0.352015
Eggerthellales	0.288286	Streptococcus_parasanguinis	0.330725
unclassified_Bilophila_species	0.279226	Bifidobacterium_dentium	0.323362
Tannerellaceae	0.275423	Veillonella_dispar	0.310273
Megamonas	0.265463	Streptococcus_mitis_oralis_pneumoniae_species	0.303952
Eubacteriaceae	0.259741	Selenomonas	0.298344
Eubacterium_ramulus	0.248477	Streptococcus_vestibularis	0.291222
Gordonibacter_pamelaeae	0.243363	Negativicutes	0.283533

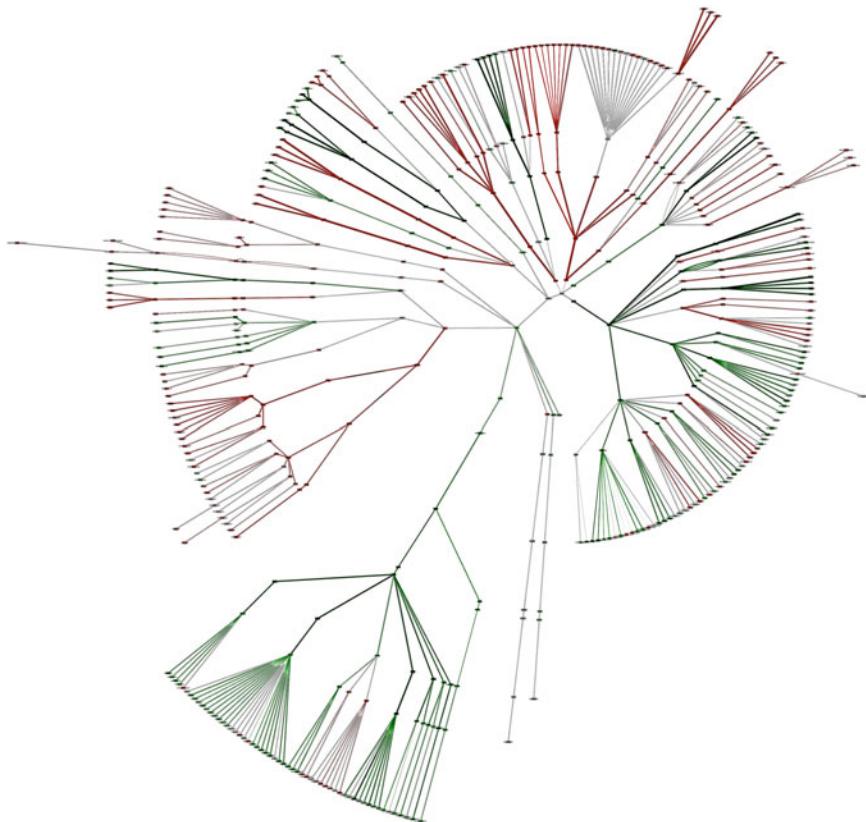
**Fig. 3** Top 20 features extracted from the Cirrhosis dataset using PopPhy-CNN**Fig. 4** Network visualization after being loaded into Cytoscape



**Fig. 5** Applying layout style to network



**Fig. 6** Applying PopPhy style to network for color



**Fig. 7** Final visualized tree after using PopPhy-CNN on Cirrhosis dataset

Once you have applied the PopPhy style, the tree should be colored green and red. For the Cirrhosis dataset, green edges and nodes mean that the taxa are important for a healthy status, while red edges and nodes are indicative of the Cirrhosis disease state. The boldness and width of the edges as well as the boldness of the color indicates the strength of the score, where the bolder the edge width and color are, the stronger the importance of the feature. The final visualization of the Cirrhosis tree after running PopPhy-CNN is shown in Fig. 7.

## 5 Conclusions

We have presented the protocol to train convolutional neural networks using our tool, PopPhy-CNN, from a microbiome dataset. PopPhy-CNN generates the performance evaluation of the predictive model and a ranked list of microbial taxa that are important for host phenotype prediction.

## 6 Notes

1. A tree file is provided in the *tree/* directory of the repository. Other trees can be used; however, it is important that they must be in Newick format in order to be parsed correctly.
2. One-hot encoding is a common format for multiclass prediction models. It converts each response value to a vector, whose length is the number of unique response values. Each vector will have a 1 at a single location (indicating the response) and will have 0's in every other position.  
 Healthy → [0, 1]  
 Dysbiosis → [1, 0]
3. MCC, precision, recall, and F1 score are defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}},$$

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}},$$

where TP, FP, TN, and FN are true positives, false positives, true negatives, and false negatives, respectively.

## References

1. Marchesi JR, Adams DH, Fava F et al (2016) The gut microbiota and host health: a new clinical frontier. *Gut* 65(2):330–339
2. Pascale A, Marchesi N, Marelli C et al (2018) Microbiota and metabolic diseases. *Endocrine* 61(3):357–371. <https://doi.org/10.1007/s12020-018-1605-5>
3. Hu J, Koh H, He L et al (2018) A two-stage microbial association mapping framework with advanced FDR control. *Microbiome* 6(1):131
4. Vangay P, Hillmann BM, Knights D (2019) Microbiome Learning Repo (ML Repo): A public repository of microbiome regression and classification tasks. *GigaScience* 8(5):giz042
5. Pasolli E, Truong DT, Malik F et al (2016) Machine learning meta-analysis of large metagenomic datasets: tools and biological insights. *PLoS Comput Biol* 12(7):e1004977
6. Ditzler G, Polikar R, Rosen G (2015) Multi-layer and recursive neural networks for metagenomic classification. *IEEE Trans Nanobioscience* 14(6):608–616
7. Reiman D, Metwally A, Dai Y (2017) Using convolutional neural networks to explore the microbiome. *Proc. 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* 4269–4272
8. Reiman D, Metwally AA, and Dai Y (2018) PopPhy-CNN: A Phylogenetic Tree Embedded Architecture for Convolution Neural Networks for Metagenomic Data. *bioRxiv*
9. Fioravanti D, Giarratano Y, Maggio V et al (2018) Phylogenetic convolutional neural networks in metagenomics. *BMC Bioinformatics* 19(2):49
10. Thanh Hai Nguyen, Yann Chevaleyre, Edi Prifti et al (2017) Deep Learning for Metagenomic Data: using 2D Embeddings and Convolutional Neural Networks. *arXiv:1712.00244*

11. Oudah M, Henschel A (2018) Taxonomy-aware feature engineering for microbiome classification. *BMC Bioinformatics* 19(1):227
12. Lloyd-Price J, Arze C, Ananthakrishnan AN et al (2019) Multi-omics of the gut microbial ecosystem in inflammatory bowel diseases. *Nature* 569(7758):655–662
13. Esteve A, Robicquet A, Ramsundar B et al (2019) A guide to deep learning in healthcare. *Nat Med* 25(1):24–29
14. Eraslan G, Avsec Ž, Gagneur J et al (2019) Deep learning: new computational modelling techniques for genomics. *Nat Rev Genet* 20(7):389–403
15. Lecun Y, Bottou L, Bengio Y et al (1998) Gradient-based learning applied to document recognition. *Proc IEEE* 86(11):2278–2324
16. Zhao N, Chen J, Carroll Ian M et al (2015) Testing in microbiome-profiling studies with MiRKAT, the microbiome regression-based kernel association test. *Am J Hum Genet* 96(5):797–807
17. Gilbert JA, Quinn RA, Debelius J et al (2016) Microbiome-wide association studies link dynamic microbial consortia to disease. *Nature* 535:94–103
18. Xia Y, Sun J (2017) Hypothesis testing and statistical analysis of microbiome. *Genes Dis* 4(3):138–148
19. Collins C, Didelot X (2018) A phylogenetic method to perform genome-wide association studies in microbes that accounts for population structure and recombination. *PLoS Comput Biol* 14(2):e1005958
20. Knights D, Parfrey LW, Zaneveld J et al (2011) Human-associated microbial signatures: examining their predictive value. *Cell Host Microbe* 10(4):292–296. <https://doi.org/10.1016/j.chom.2011.1009.1003>
21. Thomas T, Gilbert J, Meyer F (2012) Metagenomics—a guide from sampling to data analysis. *Microb Inform Exp* 2(1):3
22. Caporaso JG, Kuczynski J, Stombaugh J et al (2010) QIIME allows analysis of high-throughput community sequencing data. *Nat Methods* 7:335
23. Meyer F, Paarmann D, D’Souza M et al (2008) The metagenomics RAST server—a public resource for the automatic phylogenetic and functional analysis of metagenomes. *BMC Bioinformatics* 9:386
24. Shannon P, Markiel A, Ozier O et al (2003) Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res* 13(11):2498–2504



# Chapter 13

## Predicting Hot Spots Using a Deep Neural Network Approach

António J. Preto , Pedro Matos-Filipe , José G. de Almeida , Joana Mourão , and Irina S. Moreira

### Abstract

Targeting protein–protein interactions is a challenge and crucial task of the drug discovery process. A good starting point for rational drug design is the identification of hot spots (HS) at protein–protein interfaces, typically conserved residues that contribute most significantly to the binding. In this chapter, we depict point-by-point an in-house pipeline used for HS prediction using only sequence-based features from the well-known SpotOn dataset of soluble proteins (Moreira et al., Sci Rep 7:8007, 2017), through the implementation of a deep neural network. The presented pipeline is divided into three steps: (1) feature extraction, (2) deep learning classification, and (3) model evaluation. We present all the available resources, including code snippets, the main dataset, and the free and open-source modules/packages necessary for full replication of the protocol. The users should be able to develop an HS prediction model with accuracy, precision, recall, and AUROC of 0.96, 0.93, 0.91, and 0.86, respectively.

**Key words** Protein–protein interactions, Hot spots, Machine learning, Neural networks, Python, TensorFlow

### Abbreviations

FN	False negatives
FP	False positives
TN	True negatives
TP	True positives

---

### 1 Introduction

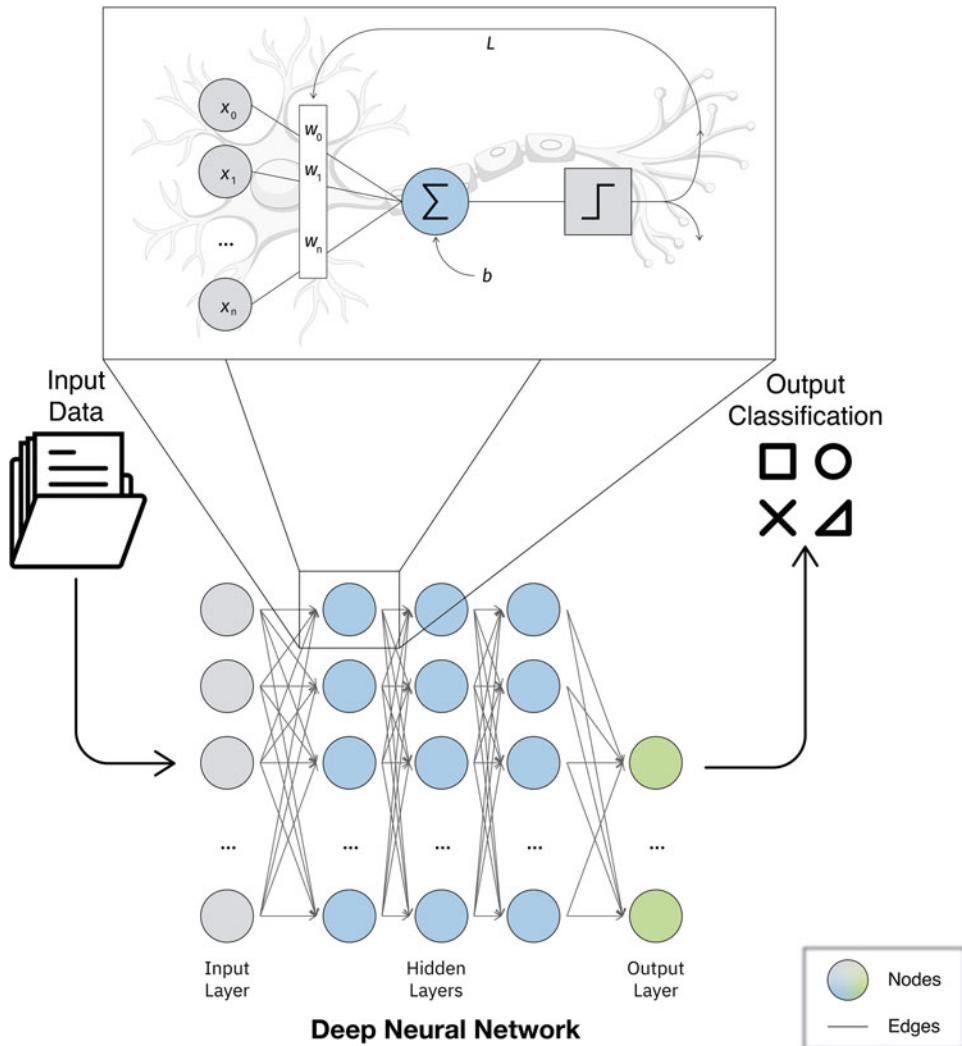
The human interactome is composed of approximately 650,000 protein–protein interactions (PPIs), which dynamically contribute to the understanding of cellular function and organization [1]. Detailed characterization of PPIs is key, as their dysregulation

is often involved in several diseases such as cancer, neurological disorders, metabolic diseases, and others [2]. As such, PPIs involved in disease pathways have become popular targets for the development of new diagnostic and therapeutic strategies [3, 4].

In PPIs, not all the residues contribute equally to the binding free energy and Hot-Spots (HS) are one of these cases. HS were defined as those residues that, upon alanine mutation, generate a variation of the free binding energy ( $\Delta\Delta G_{\text{binding}}$ ) of at least 2.0 kcal/mol [5, 6]. These are typically conserved residues and have been identified as crucial for the tight binding and stability of proteins to their partners [6]. Computational methods, in particular machine learning (ML), have been used in recent years as a viable option to overcome the technical issues (e.g., cost, time, accuracy) concerning experimental techniques, providing thorough insights and a high-throughput HS identification [7]. The key principle of this approach is that it can provide answers based on a mathematical representation by recognizing patterns within data [8, 9], avoiding the need of being explicitly programmed to achieve its goal. In HS prediction, a panoply of features (e.g., physicochemical properties, evolutionary scores, solvent-accessible area, binding energy scores) were extracted from protein interactions, and ML algorithms such as support vector machines—SVM [10], neural networks [11], and extreme gradient boosting [12] were applied to develop prediction models. Moreira et al. [5] proposed SpotOn, a model that uses sequence—and structure-based features in an ML ensemble method to predict Hot-Spots and non-Hot-Spots.

In this chapter, we depict point-by-point, an in-house pipeline used for HS prediction and based on a deep neural network (DNN) in the same dataset published in SpotOn [5]. This dataset continues to be the most relevant collection of important biological HS. Furthermore, its size is adequate to highlight the importance of being able to handle small datasets, a recurrent problem in the overlap between the biological sciences and data analysis. Tensor Flow in a familiar python-based fashion was the chosen platform for this analysis.

DNNs are a complex type of artificial neural network (ANN). Overall, DNNs assume a graph-based architecture [13], where mathematical operations, updated according to a loss function ( $L$ ), are performed in nodes connected by directed edges that carry weights ( $w_i$ ) conditioning those mathematical operations [14]. The nodes can be organized in layers, where the first layer accepts the inputs, and the last layer returns the outputs. In between, secondary operations are executed in hidden layers. In each layer, extra information can be added in the form of biases ( $b$ ) to enhance the final output of the DNN (these concepts are shown schematically in Fig. 1). While simpler ANNs comprise only one hidden layer, DNNs typically include more [15]. DNNs, like other



**Fig. 1** Representation of a DNN. Weights are represented by  $w_i$ , the input of the loss function is represented by  $L$  and the bias to the node is represented by  $b$

deep learning (DL) algorithms, allow for a greater understanding of abstract patterns within input data in comparison to simpler ML models [16].

## 2 Materials

All the materials used in this chapter are freely accessible through the Web. The provided code was tested in a 64-bit version of Linux Ubuntu 18.04 (Intel Xeon 40 Core 2.2 GHz, 126 GB RAM) and uses python version 3.7 and the associated free and open-source packages (*see* Subheading 2.2).

**Table 1**  
**Location of the SpotOn dataset and the one-hot encoded amino acid table**

Description	Location	Reference
SpotOn (spoton.Csv) <i>The SpotOn dataset table has suffered minimal transformation.</i>	<a href="https://github.com/MoreiraLAB/Deep-Neural-Networks-for-Hot-Spots-prediction">https://github.com/MoreiraLAB/Deep-Neural-Networks-for-Hot-Spots-prediction</a>	[5]
Amino acid identification (one-hot encoded)		—

**Table 2**  
**Information regarding Python and the associated free and open-source packages as well as TensorFlow, the deep-learning library for designing, building, and training ML models**

Name	Version	References
Biopython	1.74	[17]
NumPy	1.17	[18]
Pandas	0.25.1	[19]
Python	3.7.4	[20]
Scikit-learn	0.21.0	[21]
TensorFlow	1.14	[22]

## 2.1 Data

The SpotOn dataset (Table 1) used herein is constituted by 482 amino acids from 47 complexes. It was curated to ensure that every user can fully replicate the pipeline, solely from the code presented here and the tools and data available online.

## 2.2 Tools

The basic ML and python tools necessary to perform this tutorial are listed in Table 2.

---

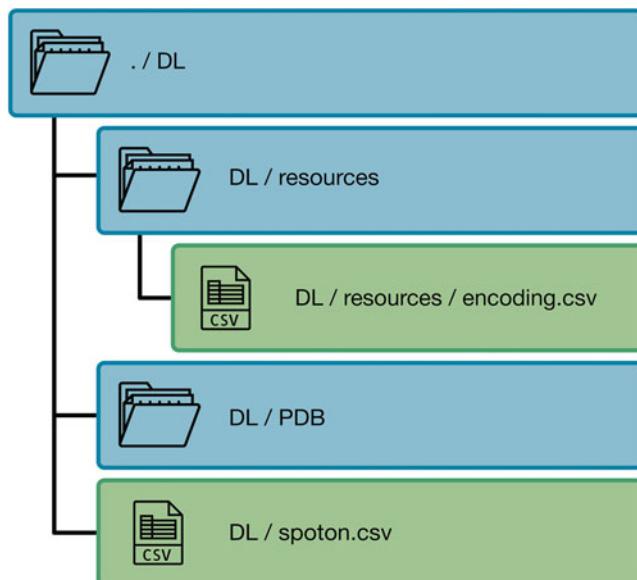
## 3 Methods

This section will cover our approach to predict HS in step-by-step fashion, with easily recognizable sequence-based features, through the implementation of a simple neural network. This tutorial makes use of the SpotOn dataset of soluble proteins, in which several amino acids are classified as Hot-spots or non-Hot-spots (Table 3). The information for the features will be acquired through the corresponding files attained from Protein Data Bank (PDB). These files have tridimensional information of the proteins shown by indicating the space coordinates of each atom. To show how it is possible to collect the data, we will also be displaying the

**Table 3**  
First four rows of the “spoton.csv” file

CPX	PDBChain	PDBResNo	PDBResName	Class
1A4Y	A	261	TRP	NS
1A4Y	A	263	TRP	NS
1A4Y	A	289	SER	NS
1A4Y	A	318	TRP	NS

The columns represent the PDBid (**CPX**), protein chain (**PDBChain**), residue number (**PDBResNo**), residue name (**PDBResName**), and hot-spot (**HS**) or non-hot-spots/null-spot (**NS**) labels (**Class**)



**Fig. 2** Folder structure to deploy the protocol. The python scripts should be added inside the “DL” folder. Blue boxes represent folders and green boxes represent “.csv” files

steps necessary to download the “.pdb” files. This chapter assumes some familiarity with Python.

The process will be split into three main steps with Code Snippets (C.S.) provided for full replication of the protocol: (Subheading 3.1) feature extraction, (Subheading 3.2) deep learning classification, and (Subheading 3.3) model evaluation. The folder structure (Fig. 2), depicts the organization required to run the code smoothly, as well as where the dataset should be located. All the forthcoming code should be integrated into scripts and run from the terminal/command line with `>>python script.py`. To run the code as it was originally run, you should have two scripts, the first containing the code from Subheading 3.1 and the second containing the code from Subheadings 3.2 and 3.3.

### 3.1 Feature Extraction

The first step of feature extraction is the acquisition of protein data from the “.pdb” files. In particular, for this protocol, we only used the amino acid number and the full sequence, which can be fetched from the column with the residue names of the “.pdb” files (Fig. 3).

In order to open and process the “.pdb” files we need several python packages (**C.S.1**): (1) to interact with the computer’s folder structure in order to access the files with the operative system package (**os**), (2) to use *biopython* for easily download and manipulation of “.pdb” files (**Bio**), and (3) to effortlessly manipulate tables (**pandas**).

```

1 import os
2 import Bio
3 from Bio.PDB import *
4 import pandas as pd

```

**C.S.1:** Importation of feature extraction packages.

MTRIX3	1	0.491450	0.812420	-0.313750		11.34353		1			
ATOM	1	N	SER	A	1	12.880	5.246	-4.370	1.00	44.14	N
ATOM	2	CA	SER	A	1	13.781	5.819	-3.336	1.00	45.16	C
ATOM	3	C	SER	A	1	13.894	4.890	-2.125	1.00	48.42	C
ATOM	4	O	SER	A	1	14.895	4.186	-1.978	1.00	53.15	O
ATOM	5	CB	SER	A	1	13.275	7.200	-2.886	1.00	46.62	C
ATOM	6	OG	SER	A	1	14.332	8.106	-2.580	1.00	42.51	O
ATOM	7	N	LEU	A	2	12.871	4.869	-1.268	1.00	46.55	N
ATOM	8	CA	LEU	A	2	12.913	4.037	-0.052	1.00	46.73	C
ATOM	9	C	LEU	A	2	11.851	2.953	0.145	1.00	42.13	C
ATOM	10	O	LEU	A	2	10.714	3.068	-0.302	1.00	40.38	O
ATOM	11	CB	LEU	A	2	12.910	4.931	1.195	1.00	47.50	C
ATOM	12	CG	LEU	A	2	14.131	5.819	1.429	1.00	48.89	C
ATOM	13	CD1	LEU	A	2	14.167	6.186	2.891	1.00	45.96	C
ATOM	14	CD2	LEU	A	2	15.422	5.103	1.000	1.00	49.50	C
ATOM	15	N	ASP	A	3	12.253	1.903	0.846	1.00	40.68	N
ATOM	16	CA	ASP	A	3	11.388	0.777	1.156	1.00	41.06	C
ATOM	17	C	ASP	A	3	11.970	0.123	2.411	1.00	39.23	C
ATOM	18	O	ASP	A	3	12.746	-0.824	2.334	1.00	40.54	O
ATOM	19	CB	ASP	A	3	11.375	-0.209	-0.011	1.00	41.53	C
ATOM	20	CG	ASP	A	3	10.345	-1.301	0.159	1.00	43.03	C
ATOM	21	OD1	ASP	A	3	10.022	-1.645	1.311	1.00	43.43	O

**Fig. 3** Representation of a “.pdb” file attained by opening it with a text editor. This file lists the residue name (green), the chain name (red), the residue number (blue) as well as the atom coordinates (yellow)

Having imported the necessary tools, we need to write a function to automatically download the “.pdb” files (**C.S.2**).

```
1 def get_unique(input_df):
2
3     from Bio.PDB import PDBList
4
5     unique_pdbs = input_df.CPX.unique()
6
7     pdbl = PDBList()
8
9     for single_pdb in unique_pdbs:
10         pdbl.retrieve_pdb_file(single_pdb, pdir='PDB', file_format = "pdb")
```

**C.S.2:** Use of *biopython* to download the “.pdb” files by iterating over a column (“CPX” corresponding to the complexes’ PDBid) with the “.pdb” file code identifiers.

To prepare in advance the extraction of protein sequences from the “.pdb” files and the opening of tables in comma separated values (.csv) format, we developed a “utilities” class, which comprises a set of helper functions, that will be useful throughout the remainder of the section (**C.S.3**). This step requires the user to have a “PDB” folder inside the same folder where the code is run (Fig. 2).

```
1 class utilities:
2
3     def __init__(self):
4
5         self.amino_acids = ['CYS', 'ASP', 'SER', 'GLN', 'LYS',
6                             'ILE', 'PRO', 'THR', 'PHE', 'ASN',
7                             'GLY', 'HIS', 'LEU', 'ARG', 'TRP',
8                             'ALA', 'VAL', 'GLU', 'TYR', 'MET']
9
10        self.converter = {'CYS': 'C', 'ASP': 'D', 'SER': 'S', 'GLN': 'Q',
11                         'LYS': 'K', 'ILE': 'I', 'PRO': 'P', 'THR': 'T',
12                         'PHE': 'F', 'ASN': 'N', 'GLY': 'G', 'HIS': 'H',
13                         'LEU': 'L', 'ARG': 'R', 'TRP': 'W', 'ALA': 'A',
```

```

13             'VAL': 'V', 'GLU': 'E', 'TYR': 'Y', 'MET': 'M'}
14
15     def table_opener(self, file_path, sep = ","):
16
17         opened_decoder = pd.read_csv(file_path, sep = sep, header = 0)
18
19     return opened_decoder

```

**C.S.3:** The “utilities” class with its “amino\_acids” and “converter” functions allow us to treat protein sequences and easily convert between the single letter and the three-letter amino acid codes, necessary to process the full sequence. The “table\_opener” function allows us to open a simple table easily and automatically generate a *pandas* data frame.

To systematically manipulate proteins, we need to store their sequence in a dictionary, that holds the proteins’ information, particularly, residue number and name (**C.S.4**).

```

1 def retrieve_sequence_raw(input_folder = "PDB", system_sep = "/"):
2
3     target_folder = os.getcwd() + system_sep + input_folder
4
5     output_dict = {}
6
7     for files in os.listdir(target_folder):
8
9         parser = PDBParser()
10
11        target_file = os.getcwd() + system_sep + input_folder +
12
13            system_sep + files
14
15        structure = parser.get_structure(files[0:-4], target_file)
16
17        pdb_id, pdb_dict = structure.id[3:], {}
18
19        for model in structure:
20
21            for chain in model:
22
23                chain_dict, chain_name, sequence = {}, chain.id, ""
24
25                for residue in chain:
26
27                    res_number, res_name = residue.get_full_id()[-1][1],
28
29                        residue.resname

```

```

15         if res_name in utilities().amino_acids:
16             single_letter = utilities().converter[res_name]
17             sequence += single_letter
18             chain_dict[res_number] = res_name
19             pdb_dict[chain_name] = chain_dict
20     output_dict[pdb_id] = pdb_dict
21
22     return output_dict

```

**C.S.4:** The “retrieve\_sequence\_raw” function mines the “PDB” folder in order to construct a dictionary that holds all the proteins’ numbered sequences.

To develop the proposed method, we also perform feature extraction. This is based upon straightforward features that are obtained from sequence alone. We built twenty features for each amino acid residue (**C.S.5**). These twenty features are a one-hot encoded version [23] of the target amino acid residue. So, they represent the twenty non-exotic amino acids and in only one of the columns a positive value could be found, while the remaining columns are filled with zero. For this, we built our one-hot amino acid encoding table (Fig. 4) in the form of a “.csv” file, stored in the “resources” folder (Fig. 2), which the user should add on the same folder of the script (*see Note 1* for additional remarks on this topic).

```

1 def generate_encoded(input_sequence):
2
3     output_table = []
4
5     encoded_table = utilities().table_opener(encoder_path)
6
7     for residue_number in input_sequence.keys():
8         residue_letter = utilities().converter[input_sequence[residue_number]]
9
10        encoded_residue = encoded_table.loc[encoded_table[class_id_name] ==
11                                              residue_letter].iloc[:,1:]
12
13        proper_row = [residue_number] + list(encoded_residue.values[0])
14
15        output_table.append(proper_row)
16
17    header = [class_id_output] + list(encoded_residue)
18
19    return pd.DataFrame(output_table, columns = header)

```

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
C	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
D	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
E	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
F	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
G	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
H	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
I	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
K	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
L	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
M	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
N	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
P	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

**Fig. 4** One-hot encoded amino acid table, considering the single-letter amino acid codes (*A* alanine, *C* cysteine, *D* aspartic acid, *E* glutamic acid, *F* phenylalanine, *G* glycine, *H* histidine, *I* isoleucine, *K* lysine, *L* leucine, *M* methionine, *N* asparagine, *P* proline, *Q* glutamine, *R* arginine, *S* serine, *T* threonine, *V* valine, *W* tryptophan, *Y* tyrosine)

**C.S.5:** The “generate\_encoded” scans a residue numbered dictionary from a protein chain and yields a table with their one-hot encoded version.

Iterating over the SpotOn table, upon use of the “converter” attribute from the “utilities” class, we can extract the one-hot encoded version of the amino acid, which is now able to be easily used as a feature.

Three more sequence-based features are also crucial for the fulfillment of this protocol. These features are simply the relative distance of every amino acid in the sequence to both the N- and C-termini of the protein and the relative distance to the target residue. Since, in this case, we only use the target residue, the last feature value is always zero. However, as such this does not affect the upcoming steps, and could be useful to the reader for other purposes, we kept this chunk of code. Thus, the “features” class (**C. S.6**), when iterating over the table from SpotOn, retrieves the one-hot encoded version of the sequence. This is performed by the “retrieve\_sequence” function which needs to be adapted depending on the dataset table, namely, by changing the names of the column tables. Finally, this Python class also has the “location\_features” function that takes the full sequence associated with the row in the target table and calculates the aforementioned relative positions for all the amino acids belonging to the sequence of the protein.

```
1 class features:
2
3     def __init__(self, row):
4         self.row = row
5
6     def retrieve_sequence(self, input_sequences):
7
8         self.complex_name = self.row["CPX"].lower()
9         self.chain = self.row["PDBChain"]
10        self.res_number = self.row["PDBResNo"]
11
12        chain_sequence = input_sequences[self.complex_name][self.chain]
13
14        encoded_sequence = generate_encoded(chain_sequence)
15
16        return encoded_sequence
17
18    def location_features(self, input_sequences, target_residue):
19
20        self.sequence_table = self.retrieve_sequence(input_sequences)
21
22        order_list = list(range(0, self.sequence_table.shape[0]))
23
24        ordering = pd.DataFrame(order_list, columns = ["order"])
25
26        inverse_ordering = pd.DataFrame(order_list[::-1], columns =
27
28            ["reverse_order"])
29
30        pseudo_distance = pd.DataFrame(list(range(0, target_residue -
31
32            self.sequence_table[class_id_output].iloc[0] + 1))[:-1]
33
34            + list(range(1, self.sequence_table[class_id_output].iloc[-1] -
35            target_residue + 1)), columns = ["pseudo_distance"])
36
37        return pd.concat([self.sequence_table, ordering / ordering.max(),
38
39            inverse_ordering / inverse_ordering.max(), pseudo_distance /
40
41            pseudo_distance.max()], axis = 1)
```

**C.S.6:** The “features” class merges the previous functions into a tool that takes as input a row from the target table and extracts twenty-three sequence-based features from the protein sequence to which the target residue belongs.

To iterate over the “spoton.csv” file, we run the “generate\_file” (**C.S.7**) function that takes as input the location of the referred table and the dictionary with the features. When iterating over the table, this function matches the target residues with the corresponding features and transforms the target label into a binary format. This function outputs two *pandas* data frames, one containing the original row identifiers from the SpotON table and the other containing the processed labels.

```

1 def generate_file(input_file, residues_features):
2
3     prepared_table, classes = [], []
4
5     for row in range(input_file.shape[0]):
6
7         current_row = input_file.iloc[row]
8
9         current_properties = pd.DataFrame(features(current_row)
10
11             .location_features(residues_features,
12
13             current_row[class_id_original]))
14
15         writeable_row = list(current_row.values) + \
16
17             current_properties.loc[current_properties[class_id_output]
18
19             == current_row[class_id_original]].values.tolist()[0]
20
21         if current_properties.isnull().any().any() == True: continue
22
23         prepared_table.append(writeable_row)
24
25         if current_row[class_name] == NS: classe = 0
26
27         elif current_row[class_name] == HS: classe = 1
28
29         classes.append(classe)
30
31
32     return pd.DataFrame(prepared_table),
33
34     pd.DataFrame(classes, columns = [class_name])

```

**C.S.7:** The “generate\_file” function iterates over the target table, matches the amino acid residues with their corresponding features, transforms the label into a binary form and outputs two

tables with the identifiers and features as well as the processed labels.

Before deploying the code, we defined a set of variables (**C.S.8**) containing most of the static information to be used. This set includes some of the file paths, usable variable strings as well as the names for the output files.

```

1 encoder_path = os.getcwd() + "/resources/encoding.csv"
2 target_table = "spoton.csv"
3 output_features_name = "spoton_clean.csv"
4 output_class_name = "class_clean.csv"
5 class_id_original = "PDBResNo"
6 class_id_output = "res_number"
7 class_id_name = "res_letter"
8 class_name = "Classe"
9 NS, HS = "NS", "HS"
```

#### **C.S.8:** Static variables to be used throughout the script.

Finally, we deploy the previous code to open the target file as a *pandas* data frame and download the “.pdb” files for all the proteins present in the SpotOn Table (**C.S.9**). Furthermore, we retrieve the dictionary containing the numbered sequences from the proteins present in SpotOn and use the target file and the processed sequences to extract the output data frames with the features and the label. Finally, we write these data frames into .csv files. These files will be used to perform deep learning classification in the following section.

```

1 opened_file = utilities().table_opener(target_table)
2 get_unique(opened_file)
3 residues_dict = retrieve_sequence_raw()
4 novel_features, classes = generate_file(opened_file, residues_dict)
5 novel_features.to_csv(output_features_name, sep = ",", index = False)
6 classes.to_csv(output_class_name, sep = ",", index = False)
```

#### **C.S.9:** Call the previous functions to use the whole pipeline and attain the files ready for Deep Learning deployment.

### 3.2 Deep Learning Classification

In the previous section, we finished our feature extraction deployment with two files: “spoton\_clean.csv”, containing the original identifiers and the extracted features, and “class\_clean.csv”, with the corresponding label for each residue in a processed format. This section makes use of these files as well as TensorFlow to construct a DNN that can classify the residues as hot spots or null spots. The code in this subsection has purposely been left unprocessed and more drawn out state, to allow the reader a clear understanding of the steps required to follow this part of the protocol.

Firstly, we import the needed packages (**C.S.10**). We use scikit-learn (*sklearn*) to perform random train-test split (*see Note 2* for details) and TensorFlow to perform the learning associated tasks. Finally, we also import numeric python (*numpy*) to handle different types of variables.

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 import tensorflow as tf
4 import numpy as np

```

**C.S.10:** The packages imported for the Deep Learning classification script.

We then write the function “encode\_binary” (**C.S.11**) to split our single label column from the previous steps into two columns, hence constructing a one-hot encoded version of the data that will make the upcoming steps simpler.

```

1 def encode_binary(input_col):
2
3     HS_col, NS_col = [], []
4
5     for class_value in input_col.values:
6
7         if class_value == 1: HS_col.append(1), NS_col.append(0)
8
9         elif class_value == 0: NS_col.append(1), HS_col.append(0)
10
11    output_df = pd.DataFrame()
12
13    output_df[NS], output_df[HS] = HS_col, NS_col
14
15    return output_df

```

**C.S.11:** The “encode\_binary” function takes a single column data frame and yields a two column one-hot encoded data frame version of the label.

Next, we construct the “neural\_network” function (**C.S.12**) that sets up a five-layered neural network (*see Note 3* for further details) in which each layer is built by multiplying the weights and adding the bias factors to the result of the activation of the previous layer. The activation was either performed with ReLU or the sigmoid function.

```

1 def neural_network(features):
2
3     layer_1 = tf.add(tf.matmul(features,
4
5         weights['hidden_1']), biases['bias_1'])
6
7     layer_2 = tf.add(tf.matmul(tf.nn.relu(layer_1),
8
9         weights['hidden_2']), biases['bias_2'])
10
11    layer_3 = tf.add(tf.matmul(tf.nn.relu(layer_2),
12
13        weights['hidden_3']), biases['bias_3'])
14
15    layer_4 = tf.add(tf.matmul(tf.nn.relu(layer_3),
16
17        weights['hidden_4']), biases['bias_4'])
18
19    layer_5 = tf.add(tf.matmul(tf.nn.relu(layer_4),
20
21        weights['hidden_5']), biases['bias_5'])
22
23    out_layer = tf.matmul(tf.nn.relu(layer_5),
24
25        weights['output'])
26
27    return out_layer

```

**C.S.12:** The “neural\_network” function sets up the input, the hidden and the output layers of the neural network to be later used.

To use the “neural\_network” function, we need to set up the starting weights and biases (**C.S.13**).

```

1 weights = {
2     'hidden_1': tf.Variable(tf.random_normal([num_input, num_hidden_1])),
3     'hidden_2': tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2])),
4     'hidden_3': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_3])),
5     'hidden_4': tf.Variable(tf.random_normal([num_hidden_3, num_hidden_4])),
6     'hidden_5': tf.Variable(tf.random_normal([num_hidden_4, num_hidden_5])),
7     'output': tf.Variable(tf.random_normal([num_hidden_5, num_classes])),
8 }
9
10 biases = {
11     'bias_1': tf.Variable(tf.random_normal([num_hidden_1])),
12     'bias_2': tf.Variable(tf.random_normal([num_hidden_2])),
13     'bias_3': tf.Variable(tf.random_normal([num_hidden_3])),
14     'bias_4': tf.Variable(tf.random_normal([num_hidden_4])),
15     'bias_5': tf.Variable(tf.random_normal([num_hidden_5])),
16 }
```

**C.S.13:** Set up the weights and biases of the neural network by yielding random values in a normal distribution.

We also need to set up some static neural network parameters for the neural network (**C.S.14**).

```

1 num_hidden_1 = 100
2 num_hidden_2 = 100
3 num_hidden_3 = 100
4 num_hidden_4 = 100
5 num_hidden_5 = 100
6 num_input = 23
7 num_classes = 2
8 display_step = 1
9 num_steps = 100000
10 learning_rate = 0.001
```

**C.S.14:** Set up the layer sizes, the number of steps, the learning rate and the batch size.

We set up TensorFlow placeholders to which the training data is fed ( $X$ ,  $Y$  variables). Furthermore, we also need to set up the “logits” variable, that can transform the output layer information into label prediction values. Next, we can calculate the loss with “softmax\_cross\_entropy\_with\_logits” and minimize this loss using an optimizer, that generally leads to good results (AdamOptimizer) [24], taking into consideration the learning rate (**C.S.15**). These are crucial steps to ensure the neural network can minimize the loss along the epochs (*see Note 4* for further details).

```

1 X = tf.placeholder("float", [None, num_input])
2 Y = tf.placeholder("int32", [None, num_classes])
3 logits = neural_network(X)
4 loss_calc = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
5 optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
6 train_op = optimizer.minimize(loss_calc)
```

**C.S.15:** Prepare the neural network output processing and loss minimization.

Similarly, to the training data, we need to prepare variables that can compare the predictions with the labeled values and evaluate the model (accuracy, AUROC, precision, and recall) (**C.S.16**).

```

1 output_pred = tf.argmax(logits, 1)
2 true_values = tf.argmax(Y, 1)
3 correct_pred = tf.equal(output_pred, true_values)
4 accuracy = tf.metrics.accuracy(labels = true_values, predictions =
    output_pred)
5 auroc = tf.metrics.auc(labels = true_values, predictions = output_pred),
6 precision = tf.metrics.precision(labels = true_values, predictions =
    output_pred),
7 recall = tf.metrics.recall(labels = true_values, predictions =
    output_pred)
```

**C.S.16:** Prepare the comparison of the predictions and labels and the output evaluation.

Having set up everything, we need to load our data and randomly split into train and test set, in this case, using a 70:30 ratio (**C.S.17**) (see Note 2 for details).

```

1 identifiers = pd.read_csv("spoton_clean.csv", sep = ",").iloc[:,0:4]
2 features = pd.read_csv("spoton_clean.csv", sep = ",").iloc[:,6:]
3 classes = pd.read_csv("class_clean.csv", sep = ",")
4 encoded_classes = encode_binary(classes)
5 X_train, X_test, y_train, y_test = train_test_split(features, encoded_classes,
test_size=0.3, random_state = 42)
```

**C.S.17:** Load the data and split it into a training and a test set.

Finally, we can initiate a TensorFlow session and deploy our neural network on the training set, printing out the loss and accuracy values of each epoch, which allows for the monitorization of the process. In the end, the model prints out the evaluation results for both the training and the test set (**C.S.18**).

```

1 init = tf.global_variables_initializer()
2 with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
3     sess.run(init)
4     sess.run(tf.initialize_local_variables())
5     for step in range(1, num_steps + 1):
6         sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
7         if step % display_step == 0 or step == 1:
8             loss, acc = sess.run([loss_calc, accuracy],
9                     feed_dict={X:batch_x, Y: batch_y})
10            print("Row:", row , "Step " + str(step) + ", Loss= " + \
11                  str(float(loss)) + ", Training Accuracy= " + \
12                  str(float(acc)))
13    print("Optimization Finished!")
14    print("Training Accuracy:", \
15        sess.run(accuracy, feed_dict={X: X_train, Y: y_train}))
```

```

12     print("Training AUROC:", \
13         sess.run(auroc, feed_dict={X: X_train, Y: y_train}))
14     print("Training Precision:", \
15         sess.run(precision, feed_dict={X: X_train, Y: y_train}))
16     print("Training Recall:", \
17         sess.run(recall, feed_dict={X: X_train, Y: y_train}))
18
19     print("Testing Accuracy:", \
20         sess.run(accuracy, feed_dict={X: X_test, Y: y_test}))
21     print("Testing AUROC:", \
22         sess.run(auroc, feed_dict={X: X_test, Y: y_test}))
23     print("Testing Precision:", \
24         sess.run(precision, feed_dict={X: X_test, Y: y_test}))
25     print("Testing Recall:", \
26         sess.run(recall, feed_dict={X: X_test, Y: y_test}))

```

**C.S.18:** Deploy the TensorFlow model and evaluate the results.

### 3.3 Metrics Used for Evaluating Model Performance

After deploying the pipeline indicated in the METHODS section, for the SpotOn dataset, we achieved the results presented in Table 4. The different metrics shown are (1) accuracy: represents the fraction of correct predictions by our model (Eq. 1); (2) precision: attempts to answer what fraction of positive identifications are actually correct (Eq. 2); (3) recall: represents the fraction of actual positives that were identified correctly by our algorithm (Eq. 3); and (4) AUROC: measures the two-dimensional area underneath the entire receiver operating characteristic (ROC) curve (Eq. 4). On a ROC curve, recall (*rec*) (Eq. 3) is plotted on the *y* axis and selectivity (*sel*) (Eq. 5) is plotted on the *x* axis.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (1)$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

**Table 4**  
**Results of the deployment of the METHODS section in the SpotOn dataset**

	<b>Training-set</b>	<b>Test-set</b>
Accuracy	0.95	0.96
Precision	0.99	0.93
Recall	0.96	0.91
AUROC	0.97	0.86

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

$$\text{AUROC} = \int_0^1 \text{rec}(\text{sel}(x)) \, dx \quad (4)$$

$$\text{selectivity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad (5)$$

---

## 4 Notes

1. Although we built our own table, it is also possible to generate an internal one-hot encoded version of the amino acids. A simple approach to do this would be to use the function LabelEncoder, from the *sklearn* package.
2. In ML, datasets are usually split into training (for adjusting the model's weights and biases) and test (for evaluation) sets. The fraction of the whole dataset reserved for training and testing stages is usually stapled at 70% and 30% respectively; however, this ratio may vary depending on the characteristics of the original dataset and the model itself [25].
3. The number of hidden layers and nodes included within them is often obtained via a grid-search procedure where various architectures are tested. The user then selects the best hyperparameter set to attain a high-performance algorithm.
4. The number of epochs corresponds to a hyperparameter that defines the number of times an entire dataset passed through the learning algorithm. An epoch that is usually too big to feed the network is divided into smaller batches, containing a lesser number of samples.

## Acknowledgments

This work was supported by the European Regional Development Fund (ERDF), through the Centro 2020 Regional Operational Programme under project CENTRO-01-0145-FEDER-000008: BrainHealth 2020 and through the COMPETE 2020—Operational Programme for Competitiveness and Internationalisation and Portuguese national funds via FCT—Fundação para a Ciência e a Tecnologia, under project[s] POCI-01-0145-FEDER-031356, PTDC/QUI-OUT/32243/2017, and UIDB/04539/2020. A. J. Preto was also supported by FCT through PhD scholarship SFRH/BD/144966/2019. I. S. Moreira was funded by the FCT Investigator Programme—IF/00578/2014 (co-financed by European Social Fund and Programa Operacional Potencial Humano). The authors would like also to acknowledge ERNEST—European Research Network on Signal Transduction, CA18133, and STRATAGEM—New diagnostic and therapeutic tools against multidrug-resistant tumors, CA17104.

## References

- Kotlyar M, Pastrello C, Malik Z et al (2019) IID 2018 update: context-specific physical protein–protein interactions in human, model organisms and domesticated species. *Nucleic Acids Res* 47:D581–D589
- Lage K (2014) Protein–protein interactions and genetic diseases: the interactome. *Biochim Biophys Acta Mol basis Dis* 1842:1971–1980
- Ran X, Gestwicki JE (2018) Inhibitors of protein–protein interactions (PPIs): an analysis of scaffold choices and buried surface area. *Curr Opin Chem Biol* 44:75–86
- Fry DC (2015) Targeting protein–protein interactions for drug discovery. *Protein–protein interactions. Methods Mol Biol* 1278:93–106
- Moreira IS, Koukos PI, Melo R et al (2017) SpotOn: high accuracy identification of protein–protein Interface hot-spots. *Sci Rep* 7:8007
- Moreira IS, Fernandes PA, Ramos MJ (2007) Hot spots—a review of the protein–protein interface determinant amino-acid residues. *Proteins* 68:803–812
- Melo R, Fieldhouse R, Melo A et al (2016) A machine learning approach for hot-spot detection at protein–protein interfaces. *Int J Mol Sci* 17:1215
- Sommer C, Gerlich DW (2013) Machine learning in cell biology—teaching computers to recognize phenotypes. *J Cell Sci* 126:5529–5539
- Libbrecht MW, Noble WS (2015) Machine learning applications in genetics and genomics. *Nat Rev Genet* 16:321–332
- Lise S, Buchan D, Pontil M et al (2011) Predictions of hot spot residues at protein–protein interfaces using support vector machines. *PLoS One* 6:e16774
- Ofran Y, Rost B (2007) ISIS: interaction sites identified from sequence. *Bioinformatics* 23: e13–e16
- Wang H, Liu C, Deng L (2018) Enhanced prediction of hot spots at protein–protein interfaces using extreme gradient boosting. *Sci Rep* 8:14285
- Jain AK, Jianchang M, Mohiuddin KM (1996) Artificial neural networks: a tutorial. *Computer (Long Beach Calif)* 29:31–44
- Gonzalez RC (2018) Deep convolutional neural networks [lecture notes]. *IEEE Signal Process Mag* 35:79–87
- Bengio Y (2009) Learning deep architectures for AI. *Found trends®. Mach Learn* 2:1–127
- LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521:436–444
- Cock PJA, Antao T, Chang JT et al (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 25:1422–1423
- van der Walt S, Colbert SC, Varoquaux G (2011) The NumPy Array: a structure for

- efficient numerical computation. *Comput Sci Eng* 13:22–30
- 19. McKinney W (2010) Data structures for statistical computing in python, in: proceeding of the 9th python in science Conf (SciPy 2010), Austin, Texas
  - 20. Rossum G van, Boer J de (1991) Linking a stub generator (AIL) to a prototyping language (python), In: EurOpen Conference Proceedings, Tromso, Norway
  - 21. Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in python. *J Mach Learn Res* 12:2825–2830
  - 22. Abadi M, Agarwal A, Barham P et al (2015) TensorFlow: large-scale machine learning on heterogeneous distributed systems, *preprint* available at arXiv:1603.04467
  - 23. Buckman J, Roy A, Raffel C et al (2018), Thermometer encoding: one hot way to resist adversarial examples. In: 6th international conference on learning representations (ICLR 2018), Vancouver, Canada
  - 24. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. *Preprint* available at arXiv:1412.6980
  - 25. Crowther PS, Cox RJ (2005) A method for optimal division of data sets for use in neural networks, presented at the knowledge-based intelligent information and engineering systems. KES 2005. In: Lecture notes in computer science, vol 3684. Springer, Berlin, Heidelberg



# Chapter 14

## Using Neural Networks for Relation Extraction from Biomedical Literature

Diana Sousa , Andre Lamurias , and Francisco M. Couto

### Abstract

Using different sources of information to support automated extracting of relations between biomedical concepts contributes to the development of our understanding of biological systems. The primary comprehensive source of these relations is biomedical literature. Several relation extraction approaches have been proposed to identify relations between concepts in biomedical literature, namely, using neural networks algorithms. The use of multichannel architectures composed of multiple data representations, as in deep neural networks, is leading to state-of-the-art results. The right combination of data representations can eventually lead us to even higher evaluation scores in relation extraction tasks. Thus, biomedical ontologies play a fundamental role by providing semantic and ancestry information about an entity. The incorporation of biomedical ontologies has already been proved to enhance previous state-of-the-art results.

**Key words** Relation extraction, Biomedical literature, Neural networks, Deep learning, Ontologies, External sources of knowledge

---

### 1 Introduction

Biomedical literature is the main medium that researchers use to share their findings, mainly in the form of articles, patents, and other types of written reports [1]. A researcher working on a specific topic needs to be up-to-date with all developments regarding the work done on the same topic. However, the volume of textual information available widely surpasses the ability of analysis by a researcher, even if restricting it to a domain-specific topic. Not only that, but the textual information available is usually in an unstructured or highly heterogeneous format. Thus, retrieving relevant information requires not only a considerable amount of manual effort but is also a time-consuming task.

Scientific articles are the primary source of knowledge for biomedical entities and their relations. These entities include human phenotypes, genes, proteins, chemicals, diseases, and other biomedical entities inserted in specific domains. A

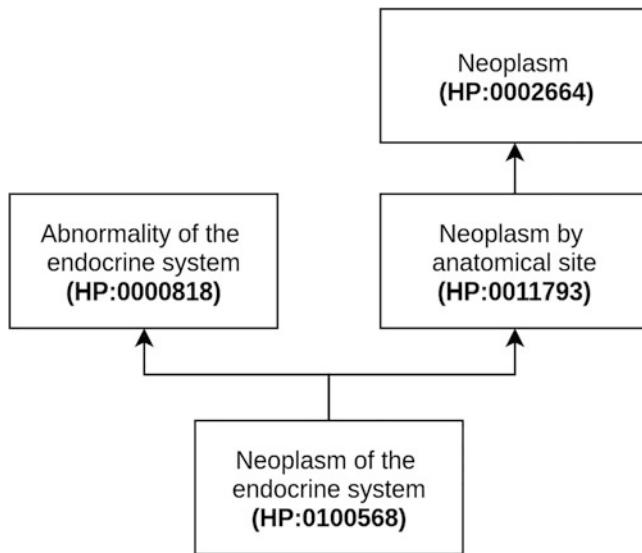
comprehensive source for articles on this topic is the PubMed platform [2], combining over 29 million citations while providing access to their metadata. Processing this volume of information is only feasible by using text mining solutions.

Automatic methods for information extraction (IE) aim at obtaining useful information from large data-sets [3]. Text mining uses IE methods to process text documents. Text mining systems usually include named-entity recognition (NER), named-entity linking (NEL), and relation extraction (RE) tasks. NER consists of recognizing entities mentioned in the text by identifying the offset of its first and last character. NEL consists of mapping the recognized entities to entries in a given knowledge base. RE consists of identifying relations between the entities mentioned in a given document.

RE can be performed by different methods, namely, by order of complexity, co-occurrence, pattern-based (manually and automatically created), rule-based (manually and automatically created), and machine learning (feature-based, kernel-based, and recurrent neural networks (RNN)).

In recent years, deep learning techniques, such as RNN, have proved to achieve outstanding results at various natural language processing (NLP) tasks, among them RE. The success of deep learning for biomedical NLP is in part due to the development of word vector models like Word2Vec [4], and, more recently, ELMo [5], BERT [6], GPT [7], Transformer-XL [8], and GPT-2 [9]. These models learn word vector representations that capture the syntactic and semantic word relationships and are known as word embeddings. Long Short-Term Memory (LSTM) RNN constitute a variant of artificial neural networks presented as an alternative to regular RNN [10]. LSTM networks deal with more complex sentences, making them more fitting for biomedical literature. To improve their results in a given domain, it is possible to integrate external sources of knowledge such as domain-specific ontologies.

The knowledge encoded in the various domain-specific ontologies, such as the Gene Ontology (GO) [11], the Chemical Entities of Biological Interest (ChEBI) ontology [12], and the Human Phenotype Ontology (HPO) [13] is deeply valuable for detection and classification of relations between different biomedical entities. Besides that, these ontologies make available important characteristics about each entity; they also provide us with the underlying semantics of the relations between those entities, such as *is-a* relations. For example, *neoplasm of the endocrine system* (HP:0100568), a phenotypic abnormality that describes a tumor (abnormal growth of tissue) of the endocrine system *is-a abnormality of the endocrine system* (HP:0000818), and *is-a neoplasm by anatomical site* (HP:0011793), which in turn *is-a neoplasm* (HP:0002664) (Fig. 1). The information provided by the ancestors is not expressed directly in the text and can support or disprove an



**Fig. 1** An excerpt of the HPO ontology showing the first ancestors of *neoplasm of the endocrine system* (HP:0100568), using **is-a** relationships

identified relation. Ontologies are formally organized in machine-readable formats, facilitating their integration in relation extraction models.

Using different sources of information, as additional data, to support automating searching for relations between biomedical concepts contributes to the development of pharmacogenomics, clinical trial screening, and adverse drug reaction identification [14]. Identifying new relations can help validate the results of recent research, and even propose new experimental hypotheses.

## 2 Related Work

This section presents the basic concepts and resources that support Relation Extraction (RE) deep learning techniques, namely, Natural Language Processing (NLP), Text Mining Primary Tasks, and initial approaches for Relation Extraction.

### 2.1 Natural Language Processing

Natural language processing (NLP) is an area in computer science that aims to derive meaning from unstructured or highly heterogeneous text written by humans. NLP covers several techniques that constitute preprocessing steps for the tasks described in Subheading 2.2. These NLP techniques have different goals and are often combined to obtain higher performance.

- **Tokenization:** has the purpose of breaking the text into tokens to be processed individually or as a sequence. These tokens are usually words but can also be phrases, numbers and other types

of elements. The most straightforward form of tokenization is breaking the input text by the use of whitespace or punctuation. However, with scientific biomedical literature, that is usually descriptive and formal, we have to account for complex entities like human phenotype terms (composed of multiple words), genes (represented by symbols), and other types of structured entities. These entities tend to be morphologically complex and need specialized tokenization pipelines. Some researchers use a compression algorithm [15], byte pair encoding (BPE), to account for biomedical vocabulary variability. BPE represents open vocabularies through a fixed-size vocabulary of variable-length character sequences, making it suitable for neural networks models.

- **Stemming and Lemmatization:** aims at reducing the variability of natural language by normalizing a token to its base form (stem) [16]. It can also take into account the context of the token, along with vocabulary and morphological analysis to determine the canonical form of the word (lemma). The stem can correspond only to a fragment of a word, but the lemma is always a real word. For instance, the stem of the word *having* is *hav* and the lemma is *have*.
- **Part-of-Speech (POS) Tagging:** consists of assigning each word of a sentence to the category where it belongs taking into account their context (e.g., verb or preposition). Each word can belong to more than one category. This feature is useful to gain information on the role of a word in a given sentence.
- **Parse Tree:** represents the syntactic structure of a sentence. There are two different types of parse trees: constituency-based parse trees and dependency-based parse trees. The main difference between the two is that the first distinguishes between the terminal and non-terminal nodes and the second does not (all nodes are terminal). In constituency-based parse trees, each node of the tree is either a *root* node, a *branch* node, or a *leaf* node. For each given sentence there is only one *root* node. The *branch* node connects to two or more *child* nodes, and the *leaf* node is terminal. These leaves correspond to the lexical tokens [17]. Dependency-based parse trees are usually simpler because they only identify the primary syntactic structure, leading to fewer nodes. Parse trees generate structures that are used as inputs for other algorithms and can be constructed based on supervised learning techniques.

## 2.2 Text Mining Primary Tasks

Text mining has become a widespread approach to identify and extract information from unstructured or highly heterogeneous text [18]. Text mining is used to extract facts and relationships in

a structured form that can be used to annotate specialized databases and to transfer knowledge between domains [19]. We may consider text mining as a sub-field of data mining. Thus, data mining algorithms can be applied if we transform text to a proper data representation, namely, numeric vectors. Even if in recent years text mining tools have evolved considerably in number and quality, there are still many challenges in applying text mining to scientific biomedical literature. The main challenges are the complexity and heterogeneity of the written resources, which make the retrieval of relevant information, that is, relations between entities, a nontrivial task.

Text Mining tools can target different tasks together or separately. Some of the primary tasks are named entity recognition (NER), named-entity linking (NEL), and relation extraction (RE).

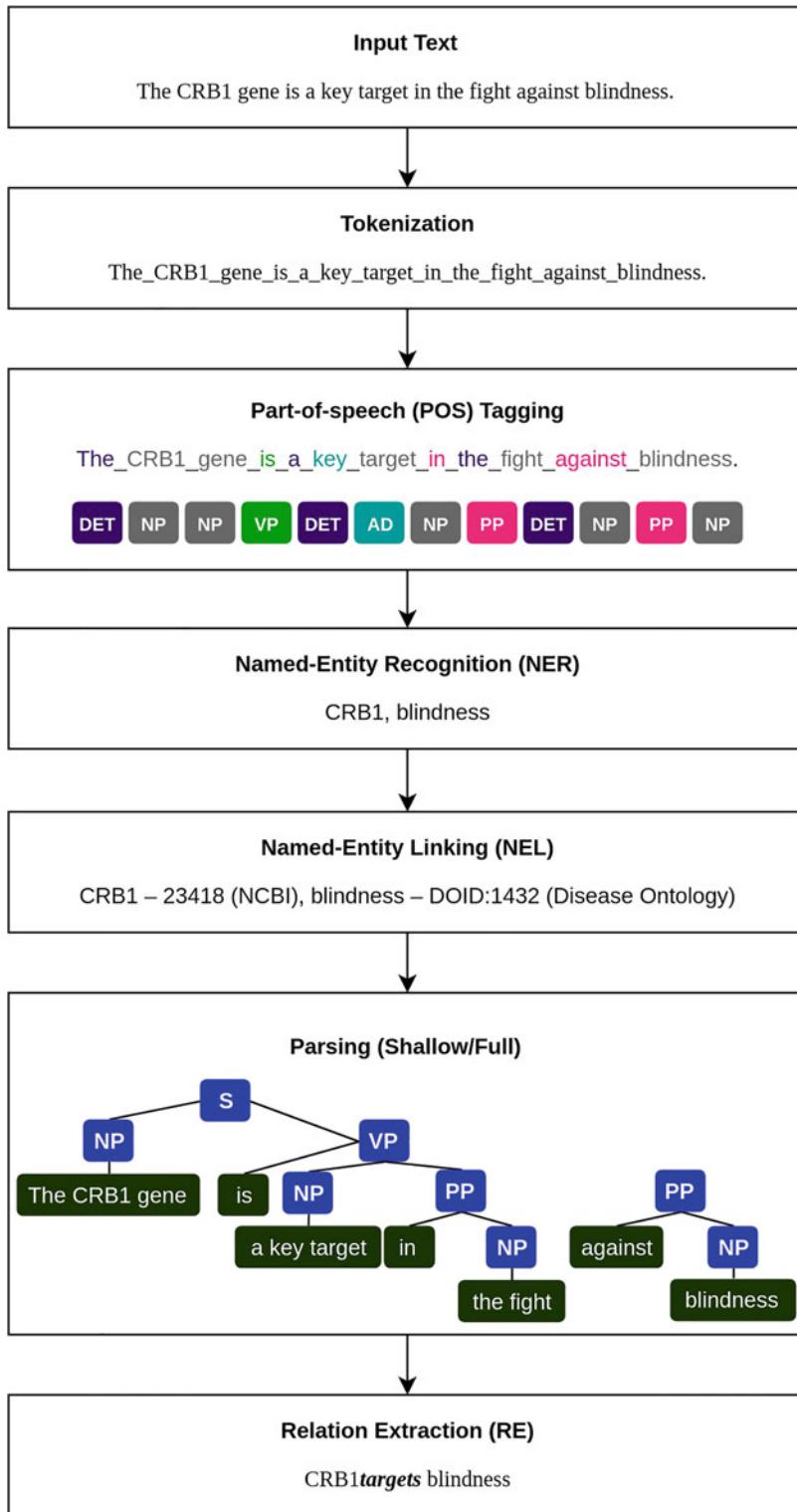
- **Named entity recognition (NER):** seeks to recognize and classify entities mentioned in the text by identifying the offset of its first and last character. The workflow of this task starts by splitting the text in tokens and then labeling them into categories (part-of-speech (POS) tagging).
- **Named-entity linking (NEL):** maps the recognized entities to entries in a given knowledge base. For instance, a gene can be written in multiple ways and mentioned by different names or acronyms in a text. NEL links all these different nomenclatures to one unique identifier. There are several organizations dedicated to providing identifiers, among them the National Center for Biotechnology Information (NCBI) for genes, and the Human Phenotype Ontology (HPO) for phenotypic abnormalities encountered in human diseases.
- **Relation extraction (RE):** identifies relations between entities (recognized manually or by NER) in a text. Tools mainly consider relations by the co-occurrence of the entities in the same sentence, but some progress is being made to extend this task to the full document (taking into account a global context) [20].

The workflow of a typical RE system is presented in Fig. 2.

### **2.3 Initial Approaches for Relation Extraction**

Through the years, several approaches have been proposed to extract relations from biomedical literature [22]. Most of these approaches work on a sentence level to perform RE, due to the inherent complexity of biomedical literature.

- **Co-occurrence:** assumes that if two entities are mentioned in the same sentence (co-occur), it is likely that they are related. Usually, the application of this approach results in a higher recall (most of the entities co-occurring in a sentence participate in a relation), and lower precision. Some methods use frequency-based scoring schemes to eliminate relations identified by chance



**Fig. 2** Workflow of a simplified RE system. (Text obtained from [21])

[23]. Nowadays, most applications use co-occurrence as a baseline against more complex approaches [24].

- **Pattern-based:** uses manually defined patterns and automatically generated patterns to extract relations. **Manually defined patterns** require domain expertise knowledge about the type of biomedical entities, their interactions, and the text subject at hand. Initial systems made use of regular expressions to match word patterns that reflected a relation between two entities [25], making use of a dictionary of words that express relation, such as *trigger* and *stimulate*. Later systems introduce part-of-speech (POS) tagging, but this has proven to be too naive, especially when applied to complex sentences, such as the ones that we typically find in biomedical literature [26]. Opposite to the co-occurrence approaches, manually defined patterns frequently achieve high precision but tend to have poor recall. This approach does not generalize well, and therefore is difficult to apply to new unseen data. **Automatically generated patterns** encompass two main approaches, bootstrapping with *seeds* [27] and leveraging of the corpora [28]. The bootstrapping method uses a small set of relations known as *seeds* (e.g., gene–disease pairs). The first step is to identify the *seeds* in the data-set and map the relation pattern they describe. The second step is to try to apply the mapped patterns to the data-set to identify new pairs of relations that follow the same construction. Finally, the original set of relations is expanded by adding these new pairs. When after repeating all previous steps no more pairs are found, the process ends. Some systems apply distant supervision techniques to keep track of the validity of the added patterns. Distant supervision uses existing knowledge base entries as gold standards to confirm or discard a relation. This method is susceptible to noisy patterns, as the original set of relations grows. On the other hand, the leveraging of the corpora method makes immediately use of the entire data-set to generate the patterns. This method requires a higher number of annotated relations and produces highly specific patterns that are unable to match new unseen data. Automatically generated patterns can achieve a higher recall than manually defined patterns, but overall the noisy patterns continue to damage the precision. Nevertheless, there are a few efforts to reduce the number of noisy patterns [29].
- **Rule-based:** also uses manually defined and automatically generated rules from the training data to extract relations. Depending on the systems, the differences between pattern-based and rule-based approaches can be minor. Ruled-based approaches not only use patterns but also additional restraints to cover issues that are difficult to express by patterns, such as checking for the negation of the relations [30]. Some ruled-based systems

distance themselves from pattern-based approaches by replacing regular expressions with heuristic algorithms and sets of procedures [31]. Like pattern-based approaches, ruled-based approaches tend to have poor recall, even though rules tend to be more flexible. The trade-off recall/precision can be improved using automatic methods for rule creation [32].

- **Machine learning (ML)-based:** usually makes use of large annotated biomedical corpora (supervised learning) to perform RE. These corpora are preprocessed using NLP tools and then used to train classification models. Beyond Neural Networks, described in detail in Subheading 3, it is possible to categorize ML methods into two main approaches, Feature-based and Kernel-based. **Feature-based approaches** represent each instance (e.g., a sentence) as a vector in an  $n$ -dimensional space. Support Vector Machines (SVM) classifiers tend to be used to solve problems of binary classification, and are considered *black-boxes* because there is no interference of the user in the classification process. These classifiers can use different features that are meant to represent the data characteristics (e.g., shortest path, bag-of-words (BOW), and POS tagging) [33]. **Kernel-based approaches** main idea is to quantify the similarity between the different instances in a data-set by computing the similarities of their representations [34]. Kernel-based approaches add the structural representation of instances (e.g., by using parse trees). These methods can use one kernel or a combination of kernels (e.g., graph, sub-tree (ST), and shallow linguistic (SL)).

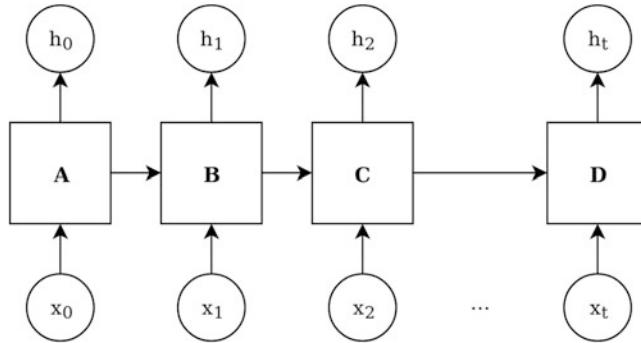
### 3 Neural Networks for Relation Extraction

Artificial neural networks have multiple different architectures implementations and variants. Often data representations are used as added sources of information to perform text mining tasks, and ontologies may even be used as external sources of information to enrich the model.

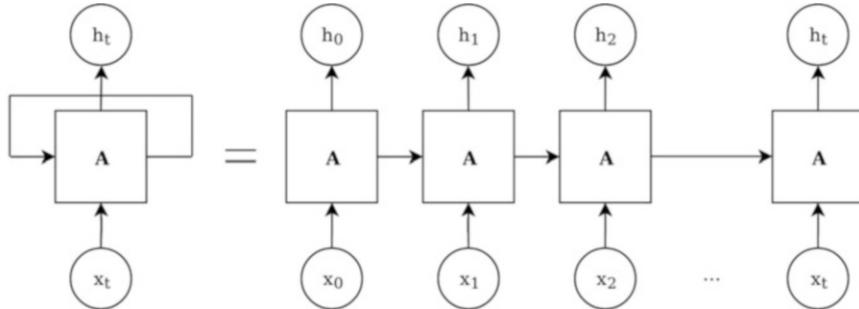
#### 3.1 Architectures

**Artificial neural networks** are a parallel combination of small processing units (nodes) which can acquire knowledge from the environment through a learning process and store the knowledge in the connections between the nodes [35] (represented by direct graphs [36]) (Fig. 3). The process is inspired by the biological brain function, in which each node corresponds to a *neuron* and the connections between the nodes represent the *synapses*.

**Recurrent neural networks** (RNN) is a type of artificial neural network where the connections between the nodes are able to follow a temporal sequence. This means that RNN can use their



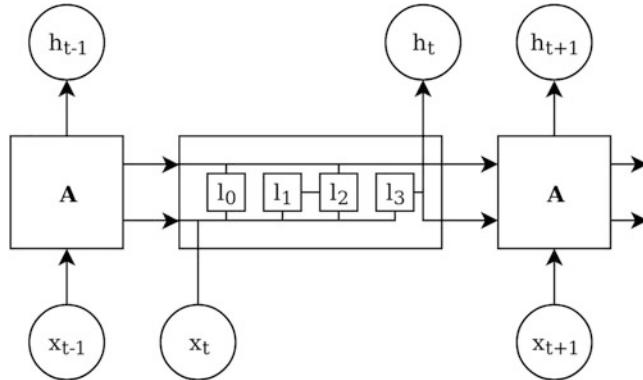
**Fig. 3** Architecture representation of an artificial neural networks model, where  $x_{0-t}$  represents the inputs and  $h_{0-t}$  the respective outputs, for each module from A to D



**Fig. 4** Architecture representation of a recurrent neural networks model, where  $x_{0-t}$  represents the inputs and  $h_{0-t}$  the respective outputs, for the repeating module A

internal state, or *memory*, to process each input sequence (Fig. 4). Deep learning techniques, such as RNN, aim to train classification models based on word embeddings, part-of-speech (POS) tagging, and other features. RNN classifiers have multilayer architectures, where each layer learns a different representation of the input data. This characteristic makes RNN classifiers flexible for application to multiple text mining tasks, without requiring task-specific feature engineering.

**Long short-term memory** (LSTM) networks are an alternative to regular RNN [10]. LSTMs are a type of RNN that handles long dependencies (e.g., sentences), making this classifier more suitable for the biomedical domain, where sentences are usually long and descriptive (Fig. 5). In recent years, the use of LSTMs to perform Relation Extraction (RE) tasks has become widespread in various domains, such as semantic relations between nominals [37]. **Bidirectional LSTMs** use two LSTM layers, at each step, one that reads the sentence from right to left, and other that reads from left to right. The combined output of both layers produces a final score for each step. Bidirectional LSTMs yield better results than traditional LSTMs when applied to the same data-sets [38].



**Fig. 5** Architecture representation of a long short-term memory networks model, where  $x_{0-t}$  represents the inputs and  $h_{0-t}$  the respective outputs, for the repeating module A, where each repeating module has four interacting layers ( $l_{0-3}$ )

### 3.2 Data Representations

The combination of multiple and different language and entity-related data representations is vital for the success of neural network models dedicated to RE tasks. Some of these features were already described in Subheading 2.1, such as POS tagging and parse trees.

**Shortest Dependency Path (SDP)** is a feature that identifies the words between two entities mentioned in the text, concentrating the most relevant information while decreasing noise [39].

**Word Embeddings** are fixed-sized numerical vectors that aim to capture the syntactic and semantic word relationships. These word vectors models use multiple different pretraining sources, for instance, Word2Vec [4] uses English Wikipedia, and BERT [6] uses both English Wikipedia and BooksCorpus. Early models, such as Word2Vec, learned one representation per word, but this proved to be problematic due to polysemous and homonymous words. Recently, most systems started to apply one embedding per word sense. One of the reasons why BERT outperforms previous methods is because it uses contextual models, meaning that it generates a unique representation for each word in a sentence. For instance, in the sentences fragments, *they got engaged*, and *students were very engaged in*, the word *engaged* for non-contextual models would have the same meaning. BERT also outperforms other word vector models that take into account the sentence context, such as ELMo [5] and ULMFit [40], due to being an unsupervised and deeply bidirectional pretrained language representation.

**WordNet Hypernyms** are a feature that helps to hierarchize entities, structuring words similar to direct acyclic graphs [41]. For example, *vegetable* is a hypernym of *tubers*, which in turn constitutes a hyponym of *vegetable*. This feature is comparable to an ontology

in the sense that a hierarchy relation is identified, but is missing the identification of the relations between the different terms.

Using different features as information sources feeding individual channels leads to multichannel architecture models. Multichannel approaches were already proven to be effective in RE tasks [39].

Regarding biomedical RE, LSTMs were successful in identifying drug–drug interactions [42], gene–mutation relations [43], and drug–mutation relations [44], among others. Some methods use domain-specific biomedical resources to train features for biomedical tasks. BioBERT [45] is a domain specific language representation model pretrained on large-scale biomedical corpora, based on BERT architecture. BioBERT, using minimal task-specific architecture modifications, significantly outperforms previous biomedical state-of-the-art models in the text mining primary tasks of named-entity recognition, named-entity linking, and RE. The BR-LSTM [46] model uses a multichannel approach with pretrained medical concept embeddings. Using the Unified Medical Language System (UMLS) concepts, BR-LSTM applies a medical concept embedding method developed by Vine et al. [47]. BO-LSTM [48] uses the relations provided by domain-specific ontologies to aid the identification and classification of relations between biomedical entities in biomedical literature.

### **3.3 Ontologies**

An ontology is a structured way of providing a common vocabulary in which shared knowledge is represented [49]. Word embeddings can learn how to detect relations between entities but manifest difficulties in grasping the semantics of each entity and their specific domain. Domain-specific ontologies provide and formalize this knowledge. Biomedical ontologies are usually structured as a directed acyclic graph, where each node corresponds to an entity and the edges correspond to known relations between those entities. Thus, a structured representation of the semantics between entities and their relations, an ontology, allows us to use it as an added feature to a machine learning classifier. Some of the biomedical entities structured in publicly available ontologies are genes properties/attributes (gene ontology (GO)), phenotypes (human phenotype ontology (HPO)), diseases (disease ontology (DO)), and chemicals (chemical entities of biological interest (ChEBI)). All of these entities participate in relations with different and same domain type entities. Hence, the information about each entity on a semantic level adds a new layer of knowledge to increase the performance of RE classifiers.

Non-biomedical models using ontologies as an added source of information to neural networks is becoming widespread for several tasks. Li et al. [50] propose using word sense definitions, provided by the WordNet ontology, to learn one embedding per word sense for word sense disambiguation tasks. Ma et al. [51] focus their work on semantic relations between ontologies and documents,

using the DBpedia ontology. Some researchers explored graph embedding techniques [52] that convert relations to a low dimensional space which represents the structure and properties of the graph. Other researchers have combined different sources of information, including ontological information, to do multi-label classification [53] and used ontology concepts to represent word tokens [54].

However, few authors have used biomedical ontologies to perform RE. Textpresso [55] is a text-mining system that works as a search engine of individual sentences, acquired from the full text of scientific articles, and articles themselves. It integrates biomedical ontological information (e.g., of genes, phenotypes, and proteins) allowing for article and sentence search query by term. The integration of the ontological information allows for semantic queries. This system helps database curation by automatically extracting biomedical relations. The IICE [56] system uses kernel-based support vector machines along with an ensemble classifier to identify and classify drug–drug interactions, linking each chemical compound to the ChEBI ontology. Tripodi et al. [57] system focus on drug–gene/protein interaction discovery to aid database curation, making use of ChEBI and GO ontologies. BO-LSTM [48] is the only model until now that incorporates ancestry information from biomedical ontologies with deep learning to extract relations from the text, specifically drug–drug interactions and gene–phenotype relations.

## 4 Evaluation Measures

The evaluation of machine learning systems is done by applying the trained models to a gold standard test-set, manually curated or annotated by domain experts and unseen by the system. For a Relation Extraction (RE) task, the gold standard test-set should correspond to the list of pairs of entities (e.g., phenotype–gene or gene–disease pairs) that co-occur in the same sentences and their relation (*Known* or *Unknown*).

To any given information extraction system it is necessary to define what constitutes a positive and negative result. In RE tasks the types of results possible are shown in Table 1.

The primary goal of a given information retrieval system is to maximize the number of TP (True Positives) and TN (True Negatives). To compare results obtained with different data-sets or different tools we have three distinct evaluation metrics: recall, precision and F-measure. Precision represents how often the results are correct, recall is the number of correct results identified and F-measure is a combination of both metrics that expresses overall performance, being the harmonic mean of precision and recall:

**Table 1**  
**Types of results obtained with an information extraction system for a RE task**

Annotator (Gold Standard)	System	Classification
Relation	Relation	True positive (TP)
	No relation	False negative (FN)
No relation	Relation	False positive (FP)
	No relation	True negative (TN)

**Table 2**  
**Biomedical RE systems current performance**

System	Precision	Recall	F-Measure
DLSTM [42]	0.7253	0.7149	0.7200
Graph LSTM (GOLD) [43]	0.4330	0.3050	0.3580
BioBERT [45]	0.8582	0.8640	0.8604
BR-LSTM [46]	0.7152	0.7079	0.7115
BO-LSTM [48]	0.6572	0.8184	0.7290

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad F\text{-measure} \\ = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (1)$$

The performance of the most recent systems dedicated to biomedical RE, described in Subheading 3.2, is shown in Table 2. These systems are not comparable, since each system is focused on the relations between different biomedical entities, and even addresses more than binary relations, such as the Graph LSTM (GOLD) system.

For RE tasks a human acceptable performance is usually around 85/90% in F-measure [58]. To facilitate the creation of gold standards we should strive for semi-automation, that is, employ automatic methods for corpora annotation (creating silver standard corpora), and then correct those annotations using domain-specific curators.

---

## 5 Conclusions

The way we communicate scientific knowledge is through scientific literature. At the current rate of document growth, the only way to process this amount of information is by using computational methods. The information obtained through these methods can lead to a better understanding of biological systems. When creating biomedical text mining systems, it is essential to take into account the specific characteristics of biomedical literature. Biological information follows different nomenclatures and levels of sentence complexity.

Automatic biomedical Relation Extraction (RE) still has a long way to go to achieve human-level performance scores. Over recent years, some innovative systems have successively achieved better results by making use of multiple knowledge sources and data representations. These systems not only rely on the training data but make use of different language and entity-related features, to create models that identify relations in highly heterogeneous text.

This chapter described the evolution of dedicated RE systems, up until neural networks. Neural networks, especially deep neural networks, which make use of multichannel architectures are composed of multiple features. Each system integrates different features distinctively. An optimal combination of features and the ideal features to perform biomedical RE tasks are still far from human level performance.

The knowledge encoded in biomedical ontologies plays a vital part in the development of deep neural networks systems, providing semantic and ancestry information for entities, such as genes, proteins, phenotypes, and diseases. One could also add semantic similarity measures as an additional information layer. Also, to accompany the growth of systems targeting different biomedical relations, there is a growing need for more domain-specific corpora, that can only be accomplished by automated corpus creation (silver standard corpora) [59].

Integrating different knowledge sources instead of relying solely on the training data for creating classification models will allow us not only to find relevant information for a particular problem quicker, but also to validate the results of recent research, and propose new experimental hypotheses.

---

## Acknowledgements

This work was supported by FCT through funding of DeST: Deep Semantic Tagger project, ref. PTDC/CCI-BIO/28685/2017 (<http://dest.rd.ciencias.ulisboa.pt/>), LASIGE Research Unit, ref.

UIDB/00408/2020, and PhD Scholarship, ref. SFRH/BD/145221/2019.

## References

1. Hearst MA (1999) Untangling text data mining. Paper presented at the 37th Annual Meeting of the Association for Computational Linguistics, College Park, Maryland, 20–26 June 1999. <https://doi.org/10.3115/1034678.1034679>
2. PubMed (1996) United States National Library of Medicine. <https://www.ncbi.nlm.nih.gov/pubmed/>. Accessed 05 Apr 2019
3. Lamurias A, Couto F (2019) Text mining for bioinformatics using biomedical literature. In: Ranganathan S, Nakai K, Schönbach C et al (eds) Encyclopedia of bioinformatics and computational biology, vol 1. Elsevier, Oxford, pp 602–611. <https://doi.org/10.1016/B978-0-12-809633-8.20409-3>
4. Mikolov T, Sutskever I, Chen K et al (2013) Distributed Representations of Words and Phrases and their Compositionality. Paper presented at the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, Nevada, 05–10 December 2013
5. Peters M, Neumann M, Iyyer M et al (2018) Deep Contextualized Word Representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, New Orleans, Louisiana, 01–06 June 2018
6. Devlin J, Chang M, Lee K et al (2018) BERT: pre-training of deep bidirectional transformers for language understanding. CoRR, arXiv:abs/1810.04805
7. Radford A, Narasimhan K, Salimans T et al (2018) Improving language understanding by generative pre-training. OpenAI. <https://openai.com/blog/language-unsupervised>. Accessed 02 May 2019
8. Dai Z, Yang Z, Yang Y et al (2019) Transformer-XL: attentive language models beyond a fixed-length context. CoRR, arXiv:abs/1901.02860
9. Radford A, Jeffrey W, Child R et al (2019) Language models are unsupervised multitask learners. OpenAI. <https://openai.com/blog/better-language-models/>. Accessed 02 May 2019
10. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9:1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
11. Ashburner M, Ball CA, Blake JA et al (2000) Gene ontology: tool for the unification of biology. Nat Genet 25:25–29. <https://doi.org/10.1038/75556>
12. Hastings J, Owen G, Dekker A et al (2016) ChEBI in 2016: improved services and an expanding collection of metabolites. Nucleic Acids Res D1(44):D1214–D1219. <https://doi.org/10.1093/nar/gkv1031>
13. Robinson PN, Mundlos S (2010) The human phenotype ontology. Clin Genet 77 (6):525–534. <https://doi.org/10.1111/j.1399-0004.2010.01436.x>
14. Luo Y, Uzuner Ö, Szolovits P (2017) Bridging semantics and syntax with graph algorithms—state-of-the-art of extracting biomedical relations. Brief Bioinform 18(1):160–178. <https://doi.org/10.1093/bib/bbw001>
15. Sennrich R, Haddow B, Birch A (2016) Neural machine translation of rare words with subword units. CoRR, arXiv:abs/1508.07909
16. Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, New York
17. Aho AV, Sethi R, Ullman JD (1986) Compilers: principles, techniques, and tools. Addison Wesley, Boston
18. Westergaard D, Stærfeldt HH, Tønsberg C et al (2018) A comprehensive and quantitative comparison of text-mining in 15 million full-text articles versus their corresponding abstracts. PLoS Comput Biol 14:1–16. <https://doi.org/10.1371/journal.pcbi.1005962>
19. Fleuren WWM, Alkema W (2015) Application of text mining in the biomedical domain. Methods 74:97–106. <https://doi.org/10.1016/j.ymeth.2015.01.015>
20. Singhal A, Simmons M, Lu Z (2016) Text mining genotype-phenotype relationships from biomedical literature for database curation and precision medicine. PLoS Comput Biol 12(11):e1005017. <https://doi.org/10.1371/journal.pcbi.1005017>
21. Alves CH, Wijnholds J (2018) AAV gene augmentation therapy for CRB1-associated retinitis Pigmentosa. In: Boon C, Wijnholds J (eds) Retinal gene therapy. Methods in molecular biology, vol 1715. Humana Press, New York, NY

22. Lamurias A, Clarke LA, Couto FM (2017) Extracting microRNA-gene relations from biomedical literature using distant supervision. *PLoS One* 12(3):e0171929. <https://doi.org/10.1371/journal.pone.0171929>
23. Zweigenbaum P, Demner-Fushman D, Yu H et al (2007) Frontiers of biomedical text mining: current progress. *Brief Bioinform* 8 (5):358–375. <https://doi.org/10.1093/bib/bbm045>
24. Bunescu R, Mooney R, Ramani A et al (2006) Integrating co-occurrence statistics with information extraction for robust retrieval of protein interactions from MEDLINE. In: proceedings of the HLT-NAACL BioNLP workshop on linking natural language and biology, New York, NY, 8 June 2006
25. Zhou D, He Y, Kwoh CK (2008) From biomedical literature to knowledge: mining protein-protein interactions. In: Smolinski TG, Milanova MG, Hassanien AE (eds) Computational intelligence in biomedicine and bioinformatics. Studies in computational intelligence, vol 151. Springer, Berlin, Heidelberg
26. Hao Y, Zhu X, Huang M et al (2005) Discovering patterns to extract protein–protein interactions from the literature: part II. *Bioinformatics* 21(15):3294–3300. <https://doi.org/10.1093/bioinformatics/bti493>
27. Wang HC, Chen YH, Kao HY et al (2011) Inference of transcriptional regulatory network by bootstrapping patterns. *Bioinformatics* 27:1422–1428. <https://doi.org/10.1093/bioinformatics/btr155>
28. Liu H, Komandur R, Verspoor K (2011) From graphs to events : a subgraph matching approach for information extraction from biomedical text. In: Proceedings of BioNLP Shared Task 2011 Workshop, Portland, Oregon, 24 June 2011
29. Nguyen QL, Tikk D, Leser U (2010) Simple tricks for improving pattern-based information extraction from the biomedical literature. *J Biomed Semantics* 1(1):9. <https://doi.org/10.1186/2041-1480-1-9>
30. Koike A, Niwa Y, Takagi T (2005) Automatic extraction of gene/protein biological functions from biomedical text. *Bioinformatics* 21:1227–1236. <https://doi.org/10.1093/bioinformatics/bti084>
31. Rinaldi F, Schneider G, Kaljurand K et al (2007) Mining of relations between proteins over biomedical scientific literature using a deep-linguistic approach. *Artif Intell Med* 39:127–136. <https://doi.org/10.1016/j.artmed.2006.08.005>
32. Xu Y, Hong K, Tsujii J et al (2014) Feature engineering combined with machine learning and rule-based methods for structured information extraction from narrative clinical discharge summaries. *J Am Med Inform Assoc* 19(5):824–832. <https://doi.org/10.1136/amiajnl-2011-000776>
33. Kim MY (2008) Detection of gene interactions based on syntactic relations. *J Biomed Biotechnol* 2008:371710. <https://doi.org/10.1155/2008/371710>
34. Giuliano C, Lavelli A, Romano L (2006) Exploiting shallow linguistic information for relation extraction from biomedical literature. In: proceedings of the 11th conference of the European chapter of the Association for Computational Linguistics, Trento, Italy, 03–07 April 2006
35. Haykin S (1998) Neural networks: a comprehensive foundation. Prentice Hall PTR, New Jersey
36. Guresen E, Kayakutlu G (2011) Definition of artificial neural networks with comparison to other networks. *Procedia Comput Sci* 3:426–433. <https://doi.org/10.1016/j.procs.2010.12.071>
37. Miwa M, Bansal M (2016) End-to-end relation extraction using LSTMs on sequences and tree structures. In: proceedings of the 54th annual meeting of the Association for Computational Linguistics, Berlin, Germany, 07–12 August 2016
38. Zhang S, Zheng D, Hu X et al (2015) Bidirectional long short-term memory networks for relation classification. In: proceedings of the 29th Pacific Asia conference on language, information and computation, 30 Oct–01 Nov 2015
39. Xu Y, Mou L, Li G et al (2015) Classifying relations via long short term memory networks along shortest dependency paths. In: proceedings of conference on empirical methods in natural language processing, Lisbon, Portugal, 17–21 Sept 2015
40. Howard J, Ruder S (2018) Universal language model fine-tuning for text classification. In: proceedings of the 56th annual meeting of the Association for Computational Linguistics, Melbourne, Australia, 15–20 July 2018
41. Fellbaum C (ed) (1998) WordNet: an electronic lexical database. The MIT Press, Cambridge
42. Wang W, Yang X, Yang C et al (2017) Dependency-based long short term memory network for drug-drug interaction extraction. *BMC Bioinformatics* 18(16):578. <https://doi.org/10.1186/s12859-017-1962-8>

43. Song L, Zhang Y, Wang Z et al (2018) N-ary relation extraction using graph-state LSTM. In: proceedings of the 2018 conference on empirical methods in natural language processing, Brussels, Belgium, 31 Oct–04 Nov 2018
44. Peng N, Poon H, Quirk C et al (2017) Cross-sentence N-ary relation extraction with graph LSTMs. *Trans Assoc Comput Linguistics* 5:101–115. [https://doi.org/10.1162/tacl\\_a\\_00049](https://doi.org/10.1162/tacl_a_00049)
45. Lee J, Yoon W, Kim S et al (2019) BioBERT: a pre-trained biomedical language representation model for biomedical text mining. CoRR, arXiv:abs/1901.08746
46. Xu B, Shi X, Zhao Z et al (2018) Leveraging biomedical resources in bi-LSTM for drug-drug interaction extraction. *IEEE Access* 6:33432–33439. <https://doi.org/10.1109/ACCESS.2018.2845840>
47. Vine LD, Zuccon G, Koopman B et al (2014) Medical semantic similarity with a neural language model. In: proceedings of the 23rd ACM international conference on conference on information and knowledge management CIKM, Shanghai, China, 03–07 Nov 2014
48. Lamurias A, Sousa D, Clarke LA et al (2018) BO-LSTM: classifying relations via long short-term memory networks along biomedical ontologies. *BMC Bioinformatics* 20:10. <https://doi.org/10.1186/s12859-018-2584-5>
49. Gruber TR (1993) A translation approach to portable ontology specifications. *Knowl Acquis* 5(2):199–220. <https://doi.org/10.1006/knac.1993.1008>
50. Li Q, Li T, Chang B (2016) Learning word sense embeddings from word sense definitions. In: Lin C-Y, Xue N, Zhao D et al (eds) *Natural language understanding and intelligent applications*, vol 10102. Springer, Cham, pp 224–235
51. Ma N, Zheng H-T, Xiao X (2017) An ontology-based latent semantic indexing approach using long short-term memory networks. *Web Big Data* 10366(2):185–199.
- <https://doi.org/10.1007/978-3-319-63579-8>
52. Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: a survey. *Knowl-Based Syst* 151:78–94. <https://doi.org/10.1016/j.knosys.2018.03.022>
53. Kong X, Cao B, Yu PS (2013) Multi-label classification by mining label and instance correlations from heterogeneous information networks. In: proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, Chicago, Illinois, 11–14 August 2013
54. Dasigi P, Ammar W, Dyer C et al (2017) Ontology-aware token embeddings for prepositional phrase attachment. In: proceedings of the 55th annual meeting of the Association for Computational Linguistics, Vancouver, Canada, 30 July–04 August 2017
55. Müller HM, Kenny EE, Sternberg PW (2004) Textpresso: an ontology-based information retrieval and extraction system for biological literature. *PLoS Biol* 2(11):e309. <https://doi.org/10.1371/journal.pbio.0020309>
56. Lamurias A, Ferreira JD, Couto FM (2014) Identifying interactions between chemical entities in biomedical text. *J Integr Bioinform* 11 (3):1–16. <https://doi.org/10.1515/jib-2014-247>
57. Tripodi I, Boguslav M, Haylu N et al (2017) Knowledge-base-enriched relation extraction. In: proceedings of the sixth BioCreative challenge evaluation workshop, Bethesda, Maryland, 18–20 October 2017
58. Aroyo L, Welty CA (2015) Truth is a lie: crowd truth and the seven myths of human annotation. *AI Mag* 36:15–24. <https://doi.org/10.1609/aimag.v36i1.2564>
59. Sousa D, Lamurias A, Couto FM (2019) A silver standard corpus of human phenotype-gene relations. In: proceedings of the 2019 conference of the north American chapter of the Association for Computational Linguistics: human language technologies, Minneapolis, Minnesota, 02–07 June 2018



# Chapter 15

## A Hybrid Levenberg–Marquardt Algorithm on a Recursive Neural Network for Scoring Protein Models

Eshel Faraggi, Robert L. Jernigan, and Andrzej Kloczkowski

### Abstract

We have studied the ability of three types of neural networks to predict the closeness of a given protein model to the native structure associated with its sequence. We show that a partial combination of the Levenberg–Marquardt algorithm and the back-propagation algorithm produced the best results, giving the lowest error and largest Pearson correlation coefficient. We also find, as previous studies, that adding associative memory to a neural network improves its performance. Additionally, we find that the hybrid method we propose was the most robust in the sense that other configurations of it experienced less decline in comparison to the other methods. We find that the hybrid networks also undergo more fluctuations on the path to convergence. We propose that these fluctuations allow for better sampling. Overall we find it may be beneficial to treat different parts of a neural network with varied computational approaches during optimization.

**Key words** Neural network, Levenberg–Marquardt algorithm, Associative memory, Protein structure, Model quality assessment

---

### 1 Introduction

The promise, deployment, and fear of machine learning applications has grown significantly in recent years. Improvements in hardware, software, and our evermore digital activities combine to create the environment where machine learning applications are finding their way into everyday items. Machine learning also holds a promise to help tackle question about complex systems such as living beings. On the fear side, applications using machine learning can be abused by oppressive regimes to control and restrict their population. Additionally, there is some fear of machine learning algorithms run amok, gaining self-evolving properties and getting loose. Here, we will focus on using machine learning to illuminate on parts of the protein structure problem.

The complexity of characterizing protein models, coupled with larger and larger datasets, represent a challenge to machine learners

[1–5]. On the one hand attempting to better characterize proteins leads to more input features describing the protein. This leads to a high dimension input feature space, and with it an increase in the abundance of local minima for the calculated error functions, and a reduction in the ability to find low error solutions using the standard back-propagation (BP) algorithm. On the other hand, the abundance of data means that a faster converging artificial learner will be able to cover a wider swath of the space of examples and hence obtain better learning. This tends to limit the usefulness of more elaborate algorithms that find multi-dimensional directions of change, rather than making changes along single dimensions as in the BP algorithm.

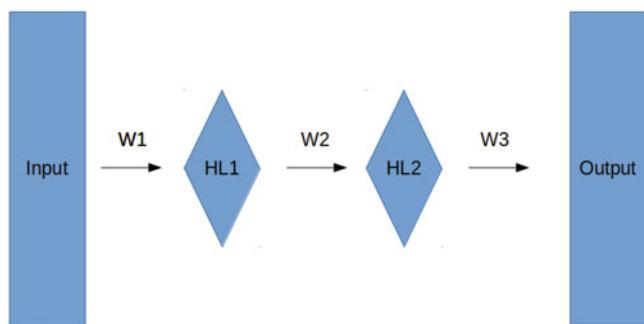
Here we present preliminary results for a hybrid of the Levenberg–Marquardt (LM) and the BP algorithm. In this approach a subset of the weights undergoes a Levenberg–Marquardt step at each epoch. This approach is useful in cases where a crucial part of the network is of a limited extent, while other parts may be very large. For scoring the closeness of a protein model to its native structure as would be displayed for example in an X-ray diffraction, a single real-valued number is enough. For example, one can use the RMSD (root mean square distance) between atoms/residues, or the Template Modeling score (TM-score) [6–8]. On the other hand, as far as we know, many numbers are needed to try and characterize a protein’s state. We split the network weights into two parts: a large part associated with protein characterization and a smaller part associated with coalescing this information into an estimate of the TM-score. We apply a BP epoch to all weights and then apply a second LM epoch on the small set of weights connecting to the single node output layer. One should note that for a lack of computer power we are unable to apply the LM algorithm to the entire neural network.

The problem of estimating how close a protein decoy model is to its native structure, if the native structure is unknown, is of significant importance in biology. An ability to do this would aid in estimating the effects of genetic variation and designing new therapies. As we have presented in previous work [3], we use accessible surface area [9], residue depth [10, 11], and residue configuration [3], to create partial sums, delineated by residue types, that characterize the protein. Using a training set of Protein Databank (PDB) [12, 13] structures and computational models we train our networks to use this information

---

## 2 Material and Methods

Except where otherwise noted, we use the same methodology as in the Seder1 scoring function [3]. Protein residue sequences are scored based on their similarity to the native protein structure of



**Fig. 1** General architecture of the neural network

the sequence as in the PDB. We used a two-layer feed-forward neural network with momentum, recently described in detail [14]. A diagram of the neural network architecture is given in Fig. 1.

HL1 and HL2 refer to the first and second hidden layers, respectively, and W1, W2, and W3 refer to the weights connecting the different layers. We used an all-connected network in which weights connect all the nodes of one layer to all the nodes of the next layer. We use a bias node for the input and hidden layers and a bipolar hyperbolic tangent activation function. For the activation parameter we use the symbol  $\alpha$ . Momentum refers to the contribution of the gradient calculated at the previous time step to the correction of the weights. We use the symbol  $p$  for the momentum parameter. To make this network a recurrent neural network the values of the nodes of both hidden layers are used as inputs for the following epoch. In this respect the network is said to have associative memory [15–18].

The LM algorithm locates a local minimum of the sum of squares of error. It alternates between the steepest descent and the Gauss–Newton method. When the error is large, the algorithm behaves more like a steepest descent method. When the error is lower, it becomes more like the Gauss–Newton method. Roughly it does this by calculating the local Jacobian and solving an error associated matrix equation. The implementation of the LM algorithm here follows that of Hagan and Menhaj [19], and Wilamowski et al. [20]. The reader is encouraged to consult the literature [19–24] for a review of the LM method.

The score we apply in this case is the closeness of a given protein structure to the native structure as given by the TM-score. Other scores can be used, for example, the probability a given mutation would cause harm. The TM-score is calculated from the distances between the residues of two aligned protein sequences:

$$\text{TM-score} = \max \left[ \frac{1}{L_t} \sum_{i=1}^{L_a} \frac{1}{1 + \left( \frac{d_i}{d_0} \right)^2} \right], \quad (1)$$

where the maximum is taken over all possible alignments.  $L_t$  is the length of the target protein,  $L_a$  is the length of the protein that is aligned to it,  $d_i$  is the distance between the residues at alignment location  $i$ , and  $d_0$  is a scaling distance, optimized to  $d_0 = 1.24\sqrt[3]{L_t - 15} - 1.8$ . With such calibration, the TM-score does not depend on the protein length, and varies between 0 and 1, with 1 indicating a perfect match. A TM-score below 0.2 indicates structurally unrelated proteins, while a TM-score greater than 0.5 indicates the two proteins belong to the same fold. We estimate the efficiency of the new neural network on the problem of estimating the “correctness” of protein models.

The input features we use are described in Table 1. A fuller description of the input features is provided with Seder1 [3]. Shortly, **res2res** refers to the partial energy sums between pairs of residues. In this case we binned distances according to the following bins (in Å): [0–3], [3,6], [6,9], and [9,∞]. **typ233w** contains information about the distance of individual atoms to the solvent and is a critical piece of information. Atoms associated with different residue types are distinguished from each other. Partial sums with respect to the inverse distance and its cube are calculated. A total of 233 heavy atom-residue types are used and the 234th type is used for unknown assignments. A stand-alone version of the DEPTH program [10, 11] is used to calculate the distance to the solvent. Distances are defined by the shortest paths to a water molecule in a solvated protein. **cmap** refers to a residue-residue contact map with a cutoff distance of 5 and 7 Å. In this case we used

**Table 1**  
**Neural network inputs**

Input feature type	Number of features
typ233w <sup>a</sup>	936
res2res <sup>b</sup>	6615
cmap <sup>c</sup>	882
Length and mass <sup>d</sup>	2
Seder1 <sup>e</sup>	1

<sup>a</sup>Residue delineated atom depth as described in Seder1 [3]

<sup>b</sup>Partial sums of residue-residue distances as described in Seder1

<sup>c</sup>Residue contact maps for cutoff distance of 5 and 7 Å

<sup>d</sup>Number of residues and atoms in protein chain

<sup>e</sup>Seder1 prediction

**Heuristic Code**

1. Randomize initial weights
2. While error for over-fit-protection set is decreasing
  - a. Calculate network node values
  - b. Record hidden layers node values as input (associative memory)
  - c. Apply one epoch of steepest descent back-propagation to all weights
  - d. Recalculate network node values
  - e. Record hidden layers node values as input (associative memory)
  - f. Apply one epoch of the LM algorithm to the output layer weights (W3)
  - g. Calculate new error over test set
3. Save weights if error for over-fit-protection set is reduced

**Fig. 2** Heuristic code for the algorithms studied. The full code represents the hybrid approach we propose here (HLM). Removing steps 2.d–f. we obtain a back-propagating algorithm with associative memory (MEM). Removing steps 2.d–f. and step 2.b. we obtain a standard back-propagating neural network (GNN)

the standard 20 residue types, and a 21st type reserved for unknown cases. A final  $z$ -score normalization was applied to all features. In total we have 8436 input features. The input and two hidden layers each have a bias neuron. A single output is produced, i.e., the output layer consists of a single neuron trained to predict the TM-score to native.

In Fig. 2 we give the heuristic code for the algorithms studied. The full algorithm represents the hybrid approach we propose here where we apply alternating BP and LM epochs. We shall refer to it as HLM. If steps 2.d–f. are removed, we obtain a back-propagating algorithm with associative memory. We refer to it as MEM. Finally, if in addition to steps 2.d–f. we remove also step 2.b., we obtain a back-propagating neural network. We refer to it as GNN. In what follows we compare the ability of these three approaches to predict the TM-score.

To train these three approaches we used a random set of 30,000 protein models. These models are for sequences for which native coordinates exist in the PDB [12, 13]. They are randomly selected from a list of protein models; this list is composed of native models from the PDB, server models from the Critical Assessment of protein Structure Prediction (CASP) [25], and decoy models from Decoys ‘R’Us [26]. We randomly select 5% of the training set as an over-fit-protection set, and do not use them for training. The remaining protein models are used to optimize the weights of the neural network so as to reduce the error between predicted and observed TM-score to native of a given model. To test these models we used another random set of 30,000 protein models that are distinct from the 30,000 models used for training and over-fit-protection.

### 3 Results

We wish to compare the effectiveness of the optimization of weights using the three methods (GNN, MEM, HLM). We do this by comparing which method obtained the best optimized weights and how robust was this optimization. For identical inputs, a neural network has various parameters that control its ability to learn. The networks presented here share the following parameters that determine their architecture and dynamics: the size of the hidden layers, the learning rate, the momentum, and the activation parameter. We studied the optimization of the network for variations of these parameters. The networks configurations with the lowest error and the variations of these parameters are given in Table 2. For the rest of the discussion we shall refer to these configurations as “best networks.” We also sampled variations in the learning rate and the momentum. For the learning rate we found all best networks had a value of  $\mu = 0.00178$ . For the momentum we found that GNN optimized best as in previous studies [3, 27–29] with  $p = 0.4$ . However, we found  $p = 0$  was the optimal value for MEM and HLM.

We start with an example of the convergence. For the three cases of the best networks, we give the training convergence curves in Fig. 3. In Fig. 3a we present the error for the over-fit-protection set as function of epoch. The upper curve in green is for the MEM network, the intermediate curve in blue is for the GNN network, and the bottom curve in red is for the HLM network. In Fig. 3b we give the error for the over-fit-protection set versus the error for the training set. The color assignments in this case are the same as in part a.

**Table 2**  
**Neural network inputs**

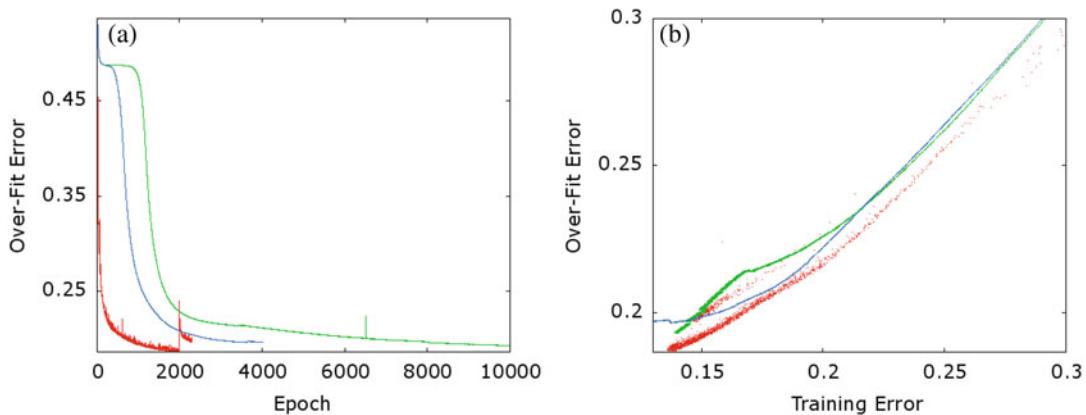
Item	Parameter		
	HL1 <sup>a</sup>	HL2 <sup>b</sup>	<i>a</i> <sup>c</sup>
Variation	3–21	3–21	0.01–0.17
<i>Optimal</i> <sup>d</sup>			
GNN	11	15	0.03
MEM	11	3	0.03
HLM	11	3	0.03

<sup>a</sup>Number of nodes in first hidden layer

<sup>b</sup>Number of nodes in second hidden layer

<sup>c</sup>Activation parameter for a hyperbolic tangent activation function

<sup>d</sup>Values for the best networks



**Fig. 3** Convergence plots for the error of the neural network. (a) Over-fit-protection set error versus epoch number. The outermost curve in green is for the MEM network, the intermediate curve in blue is for the GNN network, and the bottom curve in red is for the HLM network. (b) Over-fit-protection set error versus training error. The color assignments in this case are the same as in part (a)

**Table 3**  
Accuracy for best and top5 networks

Method	Error <sup>a</sup>		PCC <sup>b</sup>	
	Best <sup>c</sup>	Top5 <sup>d</sup>	Best	Top5
GNN	0.100	0.103	0.854	0.845
MEM	0.098	0.101	0.849	0.845
HLM	0.096	0.097	0.854	0.848

<sup>a</sup>Absolute-value error

<sup>b</sup>Pearson correlation coefficient

<sup>c</sup>Best network

<sup>d</sup>Average over five top performing architectures

The lowest absolute-value error for the best networks is given in Table 3. Additionally, we calculated the average absolute-value error for the top five architectures to study the robustness of the networks. These results are given in the table as *Top5*. The Pearson correlation coefficient (PCC) was also used to estimate the accuracy of the networks and is also given in Table 3.

## 4 Discussion

Optimization of the three types of networks we studied yielded identical values for the MEM and HLM versions with an 11-3 configuration for the hidden layers. The optimized GNN version showed an increase in the second hidden layer, possibly due to its

lack of memory. No variations in the activation parameter or the learning rate was found between the three optimized versions. However, while the top five networks for the HLM approach all had  $\mu = 0.00178$ , the MEM and GNN networks each had one case in the top five where  $\mu = 0.000178$ . For the momentum we found that GNN optimized best as in previous studies [3, 27–29] with  $p=0.4$ , while  $p=0$  was the optimal value for both MEM and HLM.

The convergence plot for the three cases shows that the HLM converges the fastest, followed by the MEM network and then the GNN network. Inspection of the curves in Fig. 3 also shows that the HLM suffered a significant reduction in accuracy around epoch 2000. This may be due to numerical problems when the parameter controlling the behavior of the LM algorithm becomes too small or too large. Since we are only interested in the best weights, this fluctuation does not affect the final results. Future studies may choose to investigate the effect of allowing more sampling time to the HLM algorithm. In general we see that the HLM convergence curve contains significantly more fluctuations than both MEM and GNN. The MEM network seems more fluctuating than the GNN network. While not shown, these trends are similar for the other networks attempted with differing number of hidden nodes and activation parameters. We propose that these fluctuations allow the HLM network to sample better the weight space.

In terms of the final optimized states, as shown in Table 3, the HLM network had the lowest absolute-value error and largest PCC, both in terms of top one and for top five. This was followed by the MEM and then the GNN network. We found that adding associative memory reduces the error by approximately 2% relative to the accuracy of GNN. Adding the hybrid LM steps reduces the error of the HLM algorithm by an additional 2% relative to the MEM algorithm. In total there is approximately a 4% reduction in error from GNN to HLM.

## 5 Conclusions

We have studied the ability of three types of neural networks to predict the closeness of a given protein model to the native structure associated with its sequence. Our comparison between the three methods showed that HLM is the best optimized, producing the lowest error and largest PCC, and that it was the most robust in the sense that other configurations of it experienced less decline than the other methods. We find that the HLM and then the MEM types of neural networks both undergo more fluctuations on the path to convergence. We propose that these fluctuations allow for better sampling. Overall we find it may be beneficial to treat different parts of a neural network with varied computational approaches during optimization.

## Acknowledgements

This work was funded in part by a National Science Foundation grant DBI 1661391, and National Institutes of Health grants R01 GM127701 and R01 GM127701-01S1. Additional support provided by the Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute, and through the National Science Foundation under Grant No. CNS-0521433.

## References

- Cheng J, Tegge AN, Baldi P (2008) Machine learning methods for protein structure prediction. *IEEE Rev Biomed Eng* 1:41–49
- Jain P, Garibaldi JM, Hirst JD (2009) Supervised machine learning algorithms for protein structure classification. *Comput Biol Chem* 33 (3):216–223
- Faraggi E, Kloczkowski, A (2014) A global machine learning based scoring function for protein structure prediction. *Proteins Struct Funct Bioinf* 82(5):752–759
- Lima AN, Philot EA, Trossini GHG, Scott LPB, Maltarollo VG, Honorio KM (2016) Use of machine learning approaches for novel drug discovery. *Expert Opin Drug Discov* 11 (3):225–239
- Baldi P (2018). Deep learning in biomedical data science. *Annu Rev Biomed Data Sci* 1:181–205
- Zhang Y, Skolnick J (2004) Scoring function for automated assessment of protein structure template quality. *Proteins Struct Funct Bioinf* 57(4):702–710
- Zhang Y, Skolnick J (2005) Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic Acids Res* 33 (7):2302–2309
- Xu J, Zhang Y (2010) How significant is a protein structure similarity with tm-score= 0.5? *Bioinformatics* 26(7):889–895
- Faraggi E, Zhou Y, Kloczkowski A (2014) Accurate single-sequence prediction of solvent accessible surface area using local and global features. *Proteins Struct Funct Bioinf* 82 (11):3170–3176
- Chakravarty S, Varadarajan R (1999) Residue depth: a novel parameter for the analysis of protein structure and stability. *Structure* 7 (7):723–732
- Tan KP, Varadarajan R, Madhusudhan MS (2011) Depth: a web server to compute depth and predict small-molecule binding cavities in proteins. *Nucleic Acids Res* 39(suppl\_2): W242–W248
- Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE (2000) The protein data bank. *Nucleic Acids Res* 28(1):235–242
- Berman H, Henrick K, Nakamura H (2003) Announcing the worldwide protein data bank. *Nat Struct Mol Biol* 10(12):980
- Faraggi E, Kloczkowski, A (2015) Genn: a general neural network for learning tabulated data with examples from protein structure prediction. In: *Artificial neural networks*. Springer, Berlin, pp 165–178
- Elman JL (1990) Finding structure in time. *Cogn Sci* 14(2):179–211
- Baldi P, Pollastri G, Andersen CAF, Brunak S (2000) Matching protein beta-sheet partners by feedforward and recurrent neural networks. In: *Proceedings of the 2000 conference on intelligent systems for molecular biology (ISMB00)*, La Jolla. AAAI Press, Palo Alto, pp 25–36
- Pollastri G, Baldi P, Fariselli P, Casadio R (2001) Improved prediction of the number of residue contacts in proteins by recurrent neural networks. *Bioinformatics* 17(suppl\_1): S234–S242
- Guler NF, Ubeyli ED, Guler I (2005) Recurrent neural networks employing Lyapunov exponents for EEG signals classification. *Expert Syst Appl* 29(3):506–514
- Hagan MT, Menhaj MB (1994) Training feed-forward networks with the Marquardt algorithm. *IEEE Trans Neural Netw* 5(6):989–993
- Wilamowski BM, Iplikci S, Kaynak O, Efe MO (2001). An algorithm for fast convergence in training neural networks. In: *IJCNN'01. International joint conference on neural networks, proceedings* (Cat. No. 01CH37222), vol 3. IEEE, Piscataway, pp 1778–1782

21. Moré JJ (1978) The Levenberg-Marquardt algorithm: implementation and theory. In: Numerical analysis. Springer, Berlin, pp 105–116
22. Battiti R (1992) First-and second-order methods for learning: between steepest descent and newton's method. *Neural Comput* 4(2):141–166
23. Lourakis MLA, Argyros AA (2005) Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment? In: Tenth IEEE international conference on computer vision (ICCV'05) volume 1, vol 2. IEEE, Piscataway, pp 1526–1531
24. Suratgar AA, Tavakoli MB, Hoseinabadi A (2007) Modified Levenberg-Marquardt method for neural networks training. *Int J Comput Inf Eng* 1(6):1745–1747
25. Moult J, Fidelis K, Kryshtafovych A, Schwede T, Tramontano A (2016) Critical assessment of methods of protein structure prediction: progress and new directions in round xi. *Proteins Struct Funct Bioinf* 84:4–14
26. Samudrala R, Levitt M (2000) Decoys 2018r2019us: a database of incorrect conformations to improve protein structure prediction. *Protein Sci* 9(7):1399–1401
27. Faraggi E, Xue B, Zhou Y (2009) Improving the prediction accuracy of residue solvent accessibility and real-value backbone torsion angles of proteins by guided-learning through a two-layer neural network. *Proteins Struct Funct Bioinf* 74(4):847–856
28. Faraggi E, Yang Y, Zhang S, Zhou Y (2009) Predicting continuous local structure and the effect of its substitution for secondary structure in fragment-free protein structure prediction. *Structure* 17(11):1515–1527
29. Faraggi E, Zhang T, Yang Y, Kurgan L, Zhou Y (2012) Spine x: improving protein secondary structure prediction by multistep learning coupled with prediction of solvent accessible surface area and backbone torsion angles. *J Comput Chem* 33(3):259–267



# Chapter 16

## Secure and Scalable Collection of Biomedical Data for Machine Learning Applications

Charles Fracchia

### Abstract

Recently, digitization of biomedical processes has accelerated, in no small part due to the use of machine learning techniques which require large amounts of labeled data. This chapter focuses on the prerequisite steps to the training of any algorithm: data collection and labeling. In particular, we tackle how data collection can be set up with scalability and security to avoid costly and delaying bottlenecks. Unprecedented amounts of data are now available to companies and academics, but digital tools in the biomedical field encounter a problem of scale, since high-throughput workflows such as high content imaging and sequencing can create several terabytes per day. Consequently data transport, aggregation, and processing is challenging.

A second challenge is maintenance of data security. Biomedical data can be personally identifiable, may constitute important trade-secrets, and be expensive to produce. Furthermore, human biomedical data is often immutable, as is the case with genetic information. These factors make securing this type of data imperative and urgent. Here we address best practices to achieve security, with a focus on practicality and scalability. We also address the challenge of obtaining usable, rich metadata from the collected data, which is a major challenge in the biomedical field because of the use of fragmented and proprietary formats. We detail tools and strategies for extracting metadata from biomedical scientific file formats and how this underutilized metadata plays a key role in creating labeled data for use in the training of neural networks.

**Key words** Data collection, Data encryption, Biomedical data, Data formats

---

### 1 Scalable Data Collection

Data collection scalability has become a significant and recent challenge for organizations that use modern instruments such as high throughput sequencers or high content imaging devices. These may output up to 10 TB/day, a volume that often exceeds the bandwidth constraints of cutting-edge biotech and healthcare companies. Furthermore, when operating at full throughput, an instrument may be limited to storing just one or two experiments worth of data. This adds pressure to a scientist's workflow and requires increased analysis speed. In many cases, scientists must either delete data or spend time and resources ensuring that the

data is transferred before the next experiment is queued up. Therefore, scalable data collection is often a rate-limiting step for companies seeking to operate at maximum throughput.

### ***1.1 On-Premise Versus Cloud Deployments***

In the last few years, industry has seen a dramatic shift away from local on-premises data collection solutions toward a cloud-first approach. On one hand, the total cost of ownership for on-premises solutions must include not only storage hardware, but facilities space, IT staff overhead, and networking hardware and maintenance. Without regular maintenance, networks and storage solutions can lead to degraded performance. The author has witnessed subpar performance at a number of top-ten pharmaceutical companies that were using locally administered data storage infrastructure.

When switching some of these large organizations to a cloud-centric model, we have experienced a  $10\times$ —or greater—data synchronization speedup compared to a local storage solution. On the financial side of the equation, cloud providers such as Amazon Web Services and Google cloud operate a pay-as-you-go model and are generous with free credits for on-boarding customers. What is more, the on-demand and scalable availability of computing resources in the cloud make today’s cloud environments more affordable and elastic to spikes in data load. From a flexibility perspective, infrastructure orchestration tools greatly help in managing the various deployed services using a software layer that is inherently more portable—no need to be onsite—and resilient to attacks.

The combination of these factors leads to a significant price differential in favor of cloud deployments, particularly for data-intensive workflows and when taking advantage of infrequent access data storage solutions. Combining the financial argument with the increased flexibility leads us—at the time of writing—to strongly recommend a cloud-first approach for data collection infrastructure.

### ***1.2 Operating System Constraints and Considerations***

While the popularity of Linux distributions has been increasing, the dominant operating system for biomedical laboratory instrumentation remains Windows. A widespread protocol for network data sharing created by Microsoft, and popularized through the Windows operating system, is the Server Message Block (SMB). SMB is popular in biomedical instrumentation as it is the default protocol for Windows network shares. However, SMB has a number of drawbacks when it comes to scalability and security of data collection [1].

The major drawback is that it has poor cross-operating system compatibility [2] and lacks robust filesystem notification hooks [3] that are necessary to collect hundreds of thousands of files created by an instrument in a single run. Practically speaking, these

drawbacks manifest themselves in incomplete data transfers, low data transfer rates, and even in some cases corrupt files. While other network file systems such as the Network File System [4] or the Andrew File System [5] have better reliability, they suffer from uneven operating system support and still lack robust file system hooks to make them practical at scale.

As a result of these incompatibilities, we strongly recommend that readers purchase adequate data storage drives and store data produced by instrumentation on the local machine directly instead of relying on network file systems. The precise definition of adequacy in this context is determined by three major factors: data volume per experimental run, total storage and read speed.

An adequate storage drive should allow saving of multiple experimental runs on a single drive. The ubiquity of platter-based hard drives with a capacity of 10 TB or more make this a viable solution even for gene sequencing workflows where individual runs can generate 1 TB of data. However, solid state drives (SSDs) have superior read and write speeds to platter-based hard drives [6] which may justify their price differential for specific experimental workflows. In particular, if the ratio of data access to data creation in a particular workflow is high, then SSDs can dramatically improve performance.

### **1.3 Network Topology Considerations**

When local drives are used to store experimental data, data synchronization must originate from the instrument computer. This has network topology and security implications that the reader needs to be aware of.

The most notable consequence is that the instrument computers must be connected to allow data centralization. Depending on the IT procedures and infrastructure of the environment, this may constitute a significant challenge. If such a challenge proves insurmountable, there exist offline solutions geared for large data volumes that can be employed. Though discouraged by the author due to complexity and induced lag time, you can learn more about them in the section titled “Substitute for Direct Instrument Connectivity.”

If laboratory instruments were not connected previously, this approach will lead to the creation of a new network segment which must be treated appropriately. As these instruments can generate vast amounts of data, and need substantial network bandwidth, it is important that IT administrators resource network capacity carefully. For high or very high throughput instruments, using state-of-the-art wiring such as CAT-6 or even fiber is necessary. When setting up such networks, IT administrators and scientists must select the network interface card carefully. We recommend researching the compatibility of drivers with the target operating system beforehand as this can severely limit the throughput or stability of the data collection solution. This is particularly

important since many instrument computer operating systems cannot be upgraded to the latest operating system version due to vendor software incompatibilities.

### **1.4 Virtualized Systems**

Fragile instrument software and operating system compatibility matrices are common in the biomedical field and upgrades often lead to incompatibilities and broken data acquisition pipelines. To combat this, organizations have tried to virtualize instrumentation computers to improve resilience. Virtual machine environments allow IT managers to rapidly spin-up new copies of known configurations to combat inadvertent breaking upgrades.

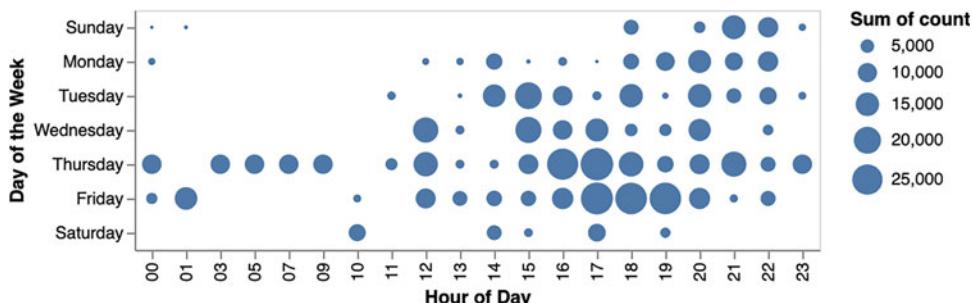
While this approach can work for some instrumentation, driver incompatibility and lower-level communication connectivity can become major roadblocks to implementing a virtualization strategy across every instrument. In many cases, communication with the instrument occurs over proprietary channels, often using USB or RS-232 as the carrier. We have witnessed a few occasions when the virtualization environment low-level drivers interfere with the instrument manufacturer's software in unpredicted ways that jeopardize the support contract with the instrumentation vendor.

As a result, we recommend that virtualized environments only be used for the most modern instrumentation and after obtaining assurance from the instrument vendor's technical team that this approach is viable.

### **1.5 Instrument-Specific Data-Writing Behaviors**

Different instruments can have very different data acquisition patterns and behaviors. Understanding those behaviors is key to effectively collecting data without corruption or unnecessary duplication.

As Fig. 1 illustrates, merely knowing when an instrument generates its data can reveal some interesting operational insights. In this case, it reveals that the instrument is used primarily during the afternoons and heavily used on Thursday and Friday, information that may be helpful when optimizing instrument utilization.



**Fig. 1** Instrument data creation activity by hour of the day and by day of the week. The counts are file creation events, which are a proxy for instrument utilization in this case, acquiring multiple files per well in a well plate

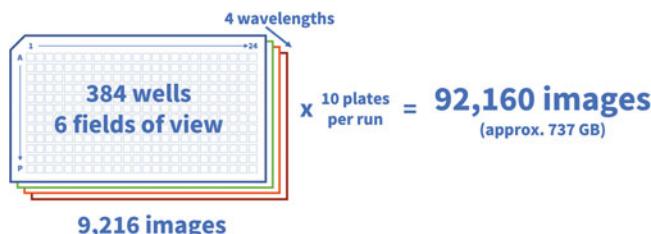
Beyond looking at when files are created, it is key to consider how the instrument is writing these files: is it writing them in a single block, appending to them; how big are the files; how long are the files kept in write-lock mode; and so on. Ignoring these aspects may lead to duplication, incomplete files or even file corruption.

This variation in file-writing behavior can be addressed by adding a small delay before uploading a file and checking that the file contents have not changed using a suitable cryptographic hash, such as SHA-512. If data is to be collected from machines operating Windows XP preceding Service Pack 3, specific hotfixes must be used to obtain the SHA-512 hash functionality [7]. Finally, in order to avoid issues with concurrent writing from multiple processes to the same file, we must ensure that the file is not being actively written to, by checking for the *EBUSY* status from the filesystem [8]; it is essential to ensure that the software data collection layer does not interfere with the data acquisition software.

### **1.6 Impact of Size and Number of Files on Data Collection Strategy**

One of the core challenges when synchronizing large amounts of data from laboratory workflows lies in how the data is stored: will it be in one large file or in multiple smaller files? Workflows such as DNA sequencing or confocal microscopy generate few but large files, sometimes exceeding a terabyte. By contrast, instruments such as high-content imagers will generate very many files, each of relatively modest size. For example, a high-content imager using 6 fields of view, 4 wavelengths, and a 384-well plate will generate 9,216 individual TIFF images. We show an example calculation of the necessary storage for a typical run on such an instrument is detailed in Fig. 2. As many of these systems are used in an automated fashion, it is not unusual for a single run to generate in excess of 500 GB of raw image data.

However, due to how the internet protocols operate at lower levels of the network stack, the precise way in which data is stored can have a dramatic effect on overall reliability, bandwidth usage and speed of the data synchronization operation. Different



**Fig. 2** Example calculation of a run on a high-content imager. Assuming that each image is typically around 8 MB, the total run described in the figure would total approximately 737 GB

approaches must be employed for each case to ensure maximum performance.

Large files should be chunked into smaller pieces, transferred, and then reassembled on the other end. The optimal chunk size depends on several factors, including the memory capacity of both the sending and receiving computers, CPU characteristics, and TCP window size; and is likely to be different for each environment. In the case of smaller files, and especially if they are very small (e.g., <1 MB), efficiency may suffer if files are sent one by one. When sending a large number of small files, we recommend batching as this will reduce the overhead per file.

When choosing a protocol to send the data, we recommend leveraging a well-supported, thoroughly assessed protocol that supports link encryption such as HTTPS. While a wide variety of file transfer protocols exist [9], using a widely deployed and well-studied protocol reduces the cost of adoption and maintenance for the technology. Furthermore, the HTTPS protocol is used by users to surf the internet, so rarely requires special configuration or specific opening of ports by IT staff. This is an important consideration for large corporate environments where IT staff are very resistant to opening any specific ports for an application as it is more complex to administer and secure.

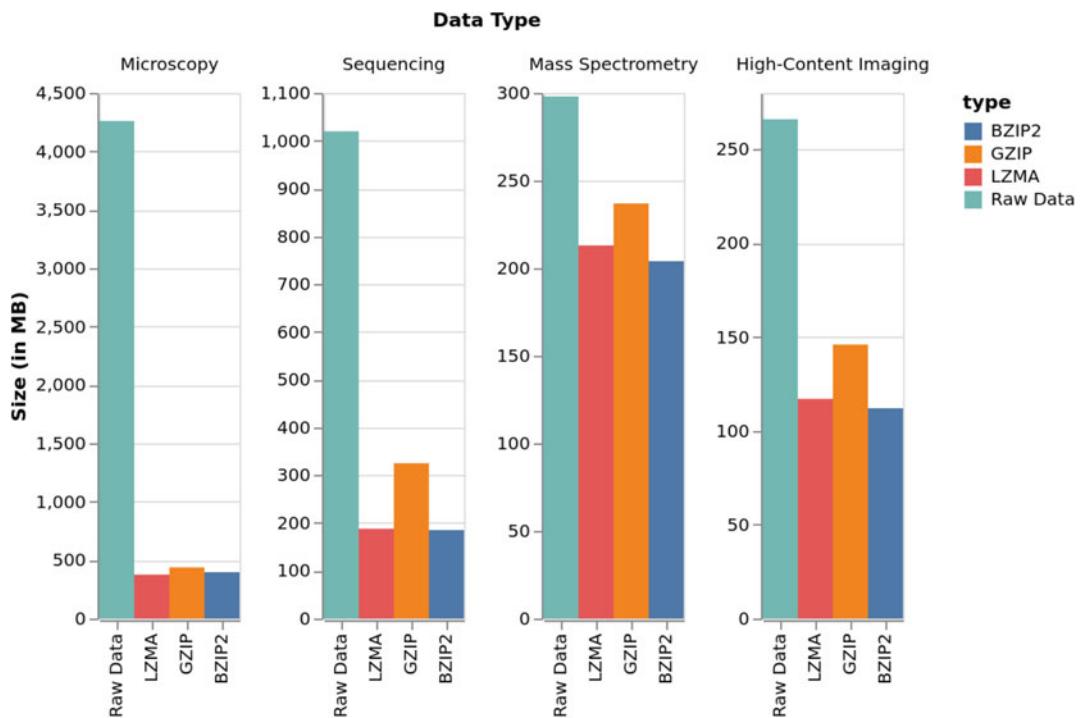
## 1.7 *Compression*

When synchronizing large amounts of data across the network, data compression can offer significant bandwidth savings. This is especially true for scientific data formats where the goal is to store raw data from an instrument in a lossless way. However, as file formats evolve and data sizes skyrocket, an increasing number of file formats are incorporating raw data compression capabilities. One example is the Zeiss CZI format that supports Lemple–Ziv–Welch (LZW) and JPEG compression for its raw pixel data [10].

However, beyond the compression that is built into the data standard directly by the instrument manufacturer, readers can—and should—take advantage of readily available compression utilities such as bzip2 [11], gzip [12] or the multitude of zip format utilities. When using such compression on scientific file formats common to the biomedical field, significant size reductions can be achieved, as shown in Fig. 3. We recommend that readers implement compression of instrumentation data before synchronization to the cloud-based environment. We further discuss compression and its impact on the security of the data synchronization system as a whole in the Secure Data Collection section below.

## 1.8 *Bandwidth Considerations*

An organization's ability to synchronize vast amounts of data to a cloud provider is highly dependent on the bandwidth of the internet connection. Readers should estimate bandwidth requirements carefully when selecting a broadband provider, even to the point of dictating the physical site of work.



**Fig. 3** Gains obtained by compressing different biological data types. All compression was performed at highest level the algorithm permits. Gains of  $10.73\times$  and  $5.51\times$  were observed on microscopy and sequencing representative datasets respectively

At present, optic fiber, often abbreviated to “fiber,” offers the largest bandwidth and scalability for businesses and residential consumers in urban environments. Crucially, fiber connectivity is most often provided in a symmetric mode, meaning that download and upload speeds are equal. Cable, satellite, or DSL connectivity are most often sold in asymmetric packages, with the download speed typically the advertised speed. Readers of this chapter may want to avoid asymmetric connectivity solutions as they typically do not provide large enough upload bandwidth and can suffer greatly from other consumers sharing the back-end infrastructure provided by the internet service provider (ISP) [13].

An example scenario is outlined in Table 1.

If large-scale data synchronization and analysis will play an important part in a project, and the place of work is not determined yet (e.g., early days of a new company), connectivity should be a key factor in location selection. It is all too easy to underestimate data requirements and their growth as an organization scale. In our experience, this can become a significant bottleneck to full throughput experimentation and lead to weeks to months of delay. Contacting local fiber carriers (not ISPs) is an effective way to identify locations that are close to existing or planned fiber

**Table 1****Data synchronization speed for various scenarios using data-intensive biomedical instrumentation**

<b>Instrument</b>	<b>Data volume per run</b>	<b>Synchronization time at 100 Mbps</b>	<b>Synchronization time at 1 Gbps</b>	<b>Synchronization time at 10 Gbps</b>
Mass Spectrometry	25 GB	33 min 20 s	3 min 20 s	20 s
Time-of-Flight Mass Spectrometry	500 GB	11 h 6 min 40 s	1 h 6 min 40 s	6 min 40 s
High-Content Imaging	1 TB	22 h 13 min 20 s	11 h 6 min 40 s	13 min 20 s
High-Throughput Sequencing	10 TB	9 days 6 h 13 min 20 s	22 h 13 min 20 s	2 h 13 min 20 s

Note that the data synchronization speeds used (100 Mbps, 1 Gbps, 10 Gbps) are round values of bandwidth capacity and real data synchronization speed is going to be much lower

conduits and minimize the infrastructure cost of installing such link. This strategy was highly effective for a private company the authors are aware of, providing a 50 Gbps uplink connectivity to a major cloud provider for a total infrastructure cost of under \$10,000. This was possible because they chose their operating location based on proximity to an existing fiber track. This allowed the company to solve what was otherwise an existential question for the core goal of the company and eliminated the data synchronization risk.

Finally, it is important to note that available bandwidth rarely operates at 100% capacity and will fluctuate independently of laboratory instrumentation. For example, in a typical organization where bandwidth is shared across the office, bandwidth will be reduced when workers are at their computer, sharing the same connection for work and laboratory data uploads. This can be mitigated by establishing Quality of Service (QoS) settings on a network or adding a second connection to increase available bandwidth and dedicate it to data collection activities. Furthermore, most ISPs only guarantee a fraction of the advertised bandwidth, usually in the range of 50–75%.

For all these reasons, combined with network topology considerations, we urge readers to procure bandwidth with a safety factor of 2× or 3× for typical laboratories. For laboratories with more than 1 instrument with large data needs, a safety factor of 5× is recommended. For example, if raw data synchronization needs to be around 30 Mbps, a 100 Mbps connection should suffice, but if instrumentation will generate 1 TB or more per week, we recommend a 1 Gbps connection at minimum.

### 1.9 Substitute for Direct Instrument Connectivity

When direct connectivity is either inadequate or not possible for infrastructural reasons, a significantly less attractive but useful solution that involves mailing physical drives can be used. Cloud service providers such as Amazon Web Services [14], Microsoft Azure [15], and Google Cloud [16] offer such solutions. These operate in a similar fashion: once ordered, the company delivers a ruggedized appliance filled with hard drives, the customer connects to the appliance using ethernet or fiber and copies data to the drives which are returned to the cloud provider to transfer the data to the cloud environment. With capacities from 40 terabytes to 1 petabyte, these solutions are often necessary to upload large volumes of historical data, though data may take days-to-weeks to become available in the cloud environment. This is not be an issue for historical data, but is in most cases inappropriate for continuous experimental workflows.

---

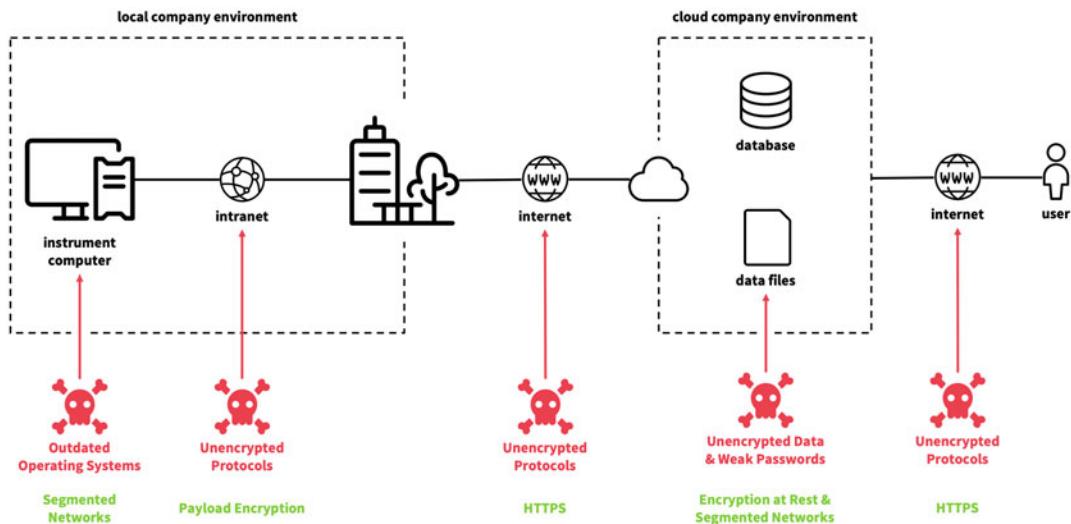
## 2 Secure Data Collection

Modern biomedical workflows involve digital data every step of the process: from electronic lab notebooks to digitally traceable manufacturing logs, and even to the electronic medical record entries completed by a nurse or physician. As this digitalization becomes more widespread, so does the need to secure the resulting data. Without strong data security, an entire organization's intellectual property and confidential research can be exposed, at a potential loss of billions of dollars. In this section we discuss the threat landscape that exists for organizations in the biomedical field, the factors contributing to potential vulnerabilities, and ways to minimize risk. To achieve good security, it is essential that secure practices be brought into an organization as early as possible. Attempting to build security into a software architecture after the fact is costly and difficult to achieve.

### 2.1 Threat Model and Landscape

In a world where the vast majority of data is digitized, protecting this information and the analysis results that derive from it is essential. It is therefore necessary to map out where data is coming from, where it is headed for collection or analysis and which channels are being used to transmit it. Figure 4 outlines a suggested configuration, communication channels, and potential threat sources.

It is vital that instruments and their operating environment be kept in a distinct network segment from the rest of the organization, where often-outdated operating systems make them easy targets for attackers. Once inside instrument software, the potential for lateral movement onto more business-critical segments, or even more simply for denial-of-service attacks on the instruments directly is very high. The most likely threat vectors for this segment



**Fig. 4** An overview of the data path from instrument back to the user, with vulnerability points. Securing these weak points in the infrastructure will force attackers to expend more energy and resources to obtain data

of the organization are unpatched or outdated operating systems on instrument machines; even now it is common to find Windows XP, 5 years after its support and security updates period ended [17]. When these machines require internet connectivity, firewall rules should be configured to allow connectivity only to and from the necessary domains, on a whitelist basis. These domains should not be overbroad, for example by avoiding Amazon Web Services wide domains, as attackers could easily deploy their attacks through such domains. Instead, create a subdomain that you or your data centralization vendor controls and whitelist this subdomain for the instrument operating system network segment. To prevent lateral movement from the instruments network, we also recommend that firewall rules block any traffic originating from the instrument networks across the local area network.

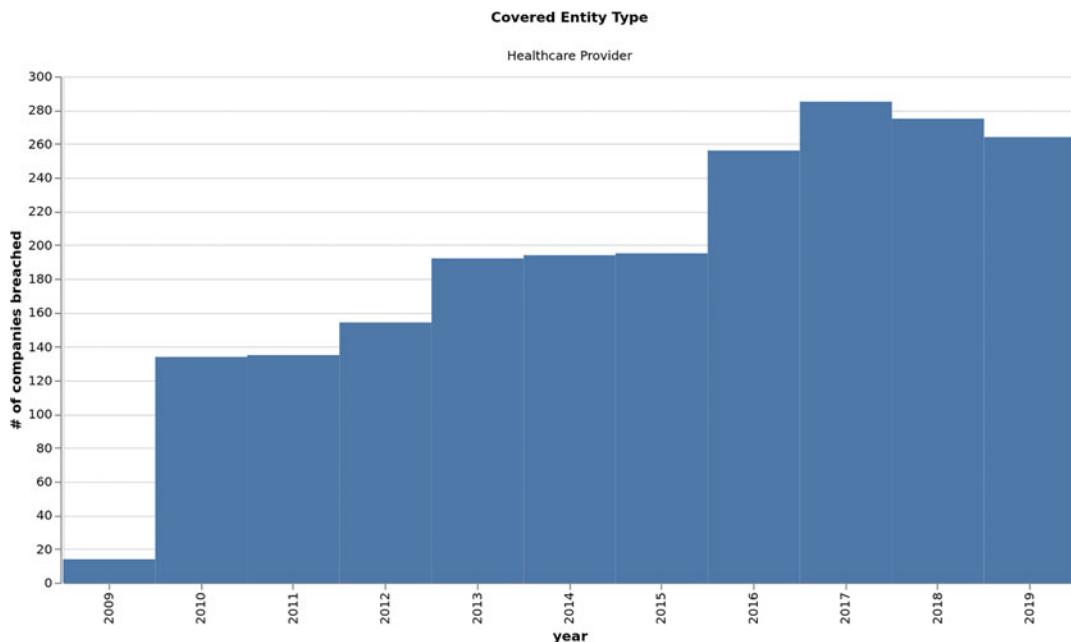
Once data collection pipelines are up and running, it is easy to realize the importance of securing the resources at the centralized point. Attackers will prioritize targets that represent high potential value: the highest value point in the process is where data gets centralized. If using a cloud environment such as Amazon Web Services, Google Cloud or Microsoft Azure, it is imperative that this environment be secured and properly segmented. While this is a wide topic that exceeds the scope of this chapter, we recommend that the following practices be followed: principle of least privilege, traffic origin whitelisting (not blacklisting), virtual private network access restrictions, segmentation using virtual private cloud (VPCs), granular user access and credentials management, periodic object store permissions audits, encryption at rest by default, and ephemeral activation of access keys during use.

## 2.2 Single-Tenant Versus Multitenant Considerations

When procuring solutions to set up some of the functions covered in this chapter, it is important to evaluate the architecture design provided by prospective vendors. The design of this infrastructure will either be single tenant or multitenant. In a multitenant mode, the provider shares network, storage, or other computational resources across multiple customers. This can be acceptable depending on the regulatory, security and accidental release constraints placed on data, but they must be carefully evaluated. Due to these constraints, and particularly if dealing with regulatory or anticipated-to-be regulatory data, it is advisable to focus on single-tenant solutions. If the solution can be deployed directly inside the organization's environment—in a dedicated VPC for example—this can dramatically simplify data ownership and custody questions. Using a single tenant model also provides a greater level of assurance with regard to access control lists (ACLs) and accidental release of proprietary data to other organizations.

## 2.3 End-to-End Encryption

Healthcare data breaches in the USA increased by 16% in the last year [18] and protecting against such incidents has become paramount (*see Fig. 5*). Encryption of data at every step of the life cycle increases the effort required by attackers to obtain data in a usable format. End-to-end encryption refers to the practice of encrypting



**Fig. 5** Growth in the number of healthcare provider companies breached from 2009 until 2019. The small number of breaches reported in 2009 is likely due to changes in reporting requirements and general lack of public information disclosure [18]

data from where it is generated, all the way to where it is consumed or processed. To achieve this, we will discuss how to implement encryption in transit and encryption at rest.

## ***2.4 Achieving Encryption in Transit***

As data flows from origin to destination, it is at risk of being intercepted by malicious actors. This is often referred to as a man-in-the-middle attack, because the attacker is listening to or intercepting the channel carrying the data. When sending data to a centralized cloud environment, data travels along public portions of the internet and therefore is open to eavesdropping and interception by third parties. One of the most ubiquitous and accessible ways to prevent such eavesdropping is to use transport layer security (TLS), which is already used when navigating to secure sites marked with HTTPS in the address [19]. Instructions on how to set up TLS for a specific environment are beyond the scope of this chapter, but we provide here context for its major components.

After creating a data collection infrastructure on a dedicated subdomain, one must create a certificate to (1) verify that data is sent to the intended place, and (2) encrypt the data being sent. Certificates can be self-signed or signed by a certificate authority. If you are deploying your data collection infrastructure in the cloud environment as suggested, you can obtain a certificate for the subdomain from the cloud provider directly at minimal cost. Implementing TLS by default for each connection makes it significantly more difficult for attackers to intercept your data. Note however that man-in-the-middle attacks can still occur by the creation of a second certificate signed by a compromised certificate authority [20–22]. It is therefore good practice to check the individual signature of the certificate you have created and control; this practice is called certificate pinning.

Finally, when setting up TLS for data collection infrastructure, you will be able to set the preferred key exchange algorithm and cipher used. Giving preference in your installation to an algorithm that allows perfect forward secrecy [23] such as the elliptic-curve Diffie-Hellman (ECDH) will thwart after-the-fact data decryption. Using ciphers that are generally accepted as safe—such as AES-GCM—further increases the security posture of the resulting solution.

## ***2.5 Additional Data Encryption***

When using transport layer security, the data will be encrypted in transit using keys negotiated between the computer sending the data and the server receiving it. While this is likely adequate for many situations, where a higher level of assurance and confidentiality is required another layer of encryption can be added by encrypting data on the instrument computer before it is sent. An attacker then needs multiple keys, stored in separate environments, to decrypt data being sent.

To achieve this second layer of encryption, we recommend readers use well-known encryption ciphers and their vetted implementation from supported software libraries. The current advanced encryption standard (AES) with key lengths of at least 256 bits should provide adequate security and speed. To further limit the potential impact of an attacker, a separate random key for each file that is transmitted will protect against replay attacks [24] and make any known-plaintext attacks [25] significantly harder for an attacker to carry out.

For this second layer of security to be adequate, the encryption key must be distributed to the server in a way that forces the attacker to break into a second system or method. By encrypting the per-file AES key using the server's public RSA key, we can now send the encrypted file and the encrypted AES key safely over the internet, knowing that only the server possessing the corresponding RSA private key can reverse the process and read the original file unencrypted.

## ***2.6 Achieving Encryption At Rest***

Once data is secured in transit, one must consider how to secure data when it is at rest, in other words when it is stored. The most important location for encryption at rest is the point of data centralization. If you are centralizing this data in a cloud environment, you will most likely store the data in an object store such as Amazon S3, Google Cloud Storage or Microsoft Azure Storage, which support at rest encryption using modern ciphers such as AES [26–28]. Some platforms turn this setting on by default, and we strongly encourage readers to ensure it is set. For additional security, certain cloud providers enable users to encrypt the data at rest using user-supplied keys, further segmenting the encrypted data in case of a breach at the cloud provider. This is a setting we encourage more security-minded or sensitive organizations to consider, though it is not likely to be needed for most users.

## ***2.7 Encryption and Compression***

When implementing compression for the synchronized data files, one must consider potential security implications. To obtain significant gains from compression, one must first compress the data and then encrypt it, since any sufficiently good encryption mechanism would yield a pseudorandom output, which in turn reduces the effectiveness of compression. On the other hand, any form of compression will reduce the entropy of the data and thus open the door to chosen plaintext and compression oracle attacks [29, 30]. However, for these attacks to be viable, the attacker must be able to input data into the data synchronization infrastructure and observe the corresponding changes in ciphertext. In our scenario, this would mean that the attackers have already gained access to the instruments or laboratory environment. However, not to be discounted, this potential vector of attack is one of the motivations behind our recommendation to use separate

pseudorandom keys, generated for each file. This strongly limits the gains an attacker may obtain for these attacks on the broader infrastructure. If this recommendation is followed, an attacker that successfully obtains the AES encryption key only has the key to a single file, not the whole platform. This mitigation therefore shifts the weakest point of security away from file encryption.

---

### 3 Metadata Extraction

When collecting large amounts of data from laboratory instruments, one naturally wonders how this data is stored, and whether auxiliary information describing experimental conditions and instrument context is accessible. With the increased digitization of the biomedical field, many instrument manufacturers have embedded significant amounts of contextual data in their formats. However, identifying and extracting this information can be challenging as manufacturers often consider their file formats proprietary and obscure their contents. In this section, we will define what we mean by metadata, give an overview of the diversity of file formats in the life sciences, outline techniques for identifying and extracting metadata from proprietary formats and finally provide guidance for storing and searching the extracted metadata.

#### **3.1 What Is “Metadata”?**

We define metadata as any information contained in an experiment’s data package that is not the raw numerical data obtained from the instrument. For example, in a Mass Spectrometry data package, only the trace data is considered raw data; everything else from method files to instrument settings is metadata, such as instrument settings, experimental acquisition status and time stamps, consumables history, instrument calibration results, user-selected analysis settings and much more.

#### **3.2 File Format Diversity in the Life Sciences**

The combination of distributed instrumentation development and strong academic contributions to the bioinformatics field over the years has led to a range of file formats (Table 2, [31]). In other fields, the creation of common standards and inexpensive licensing fees led to the wide adoption of a standard. An example is the GPIB standard initially developed by Hewlett-Packard [32].

In recent years, efforts have been made to create a unified file format for data interoperability and interchange in the biomedical field [33–35]. These efforts have so far failed to gather sufficient traction and have not seen widespread adoption beyond proof-of-concept deployments. At the time of writing, instrument vendors do not see data interchange as either a competitive advantage or a necessary feature and regard their data formats as important intellectual property to protect.

**Table 2**  
**Sample of biomedical file formats by field of study**

Experimental field	File type
Microscopy	.1sc, .2fl, .acff, .afi, .afm, .aim, .al3d, .ali, .am, .amiramesh, .apl, .arf, .avi, .bif, .bin, .bip, .bmp, .btf, .c01, .cfg, .ch5, .cif, .cr2, .crw, .csv, .cxz, .dat, .dcm, .dib, .dicom, .dm2, .dm3, .dm4, .dti, .dv, .eps, .epsi, .exp, .fdf, .fff, .ffr, .fits, .flex, .fli, .frm, .gel, .gif, .grey, .gz, .h5, .hdf, .hdr, .hed, .his, .htd, .html, .hx, .i2l, .ics, .ids, .im3, .img, .ims, .inr, .ipl, .ipm, .ipw, .j2k, .jp2, .jpf, .jpg, .jpk, .jpx, .klb, .l2d, .labels, .lei, .lif, .liff, .lim, .lms, .lsm, .map, .mea, .mnc, .mng, .mod, .mov, .mrc, .mrcs, .mrw, .msr, .mtb, .mv2, .naf, .nd, .nd2, .ndpi, .ndpis, .nef, .nhdr, .nii, .nrrd, .obf, .obsep, .oib, .oif, .oir, .ome, .par, .pbm, .pcoraw, .pcx, .pds, .pgm, .pic, .pict, .png, .pnl, .ppm, .pr3, .ps, .psd, .qptiff, .r3d, .raw, .rcpnl, .rec, .res, .scn, .sdt, .seq, .sif, .sld, .sm2, .sm3, .spc, .spe, .spi, .st, .stk, .stp, .svs, .sxm, .tf2, .tf8, .tfr, .tga, .tif, .tiff, .tnb, .top, .txt, .v, .vff, .vms, .vsi, .vws, .wat, .wlz, .wpi, .xdce, .xml, .xqd, .xqf, .xv, .xys, .zfp, .zfr, .zvi
Mass Spectrometry	.BAF, .D (folder), .DAT, .FID, .ita, .itm, .lcd, .MS, .PKL, .QGD, .RAW* (folder), .SMS, .spc, .t2d, .tdc, .TDF, .WIFF, .XMS, .YEP, ANDI-MS, JCAMP-DX, mzData, mzML, mzXML, netCDF
Genomics	SRA XML, MAGE-TAB (SDRF, IDF), BCR XML, SRA XML, SVS, TIFF, FASTQ, BAM, MAF, TSV, VCF, IDAT, ClustalW, CAF, BCL, BNF, VCF, GFF, GTF, ABI

The variety and a plethora of formats have made interoperability a difficult task

In a few cases, widely adopted data standards have been established in specific techniques or fields of study. An example is the field of flow cytometry and the FCS data file standard [36]. However, while the format is widely used, a plethora of programmatic parsers and software utility packages exist lacking a reference and fully verified implementation [37–40]. In fact, a review of the major Python libraries for reading the FCS format by the author found that only one library properly implemented delimiter characters; the other packages were unable to parse FCS data from an instrument vendor despite the standardized data format.

The experience we describe with the FCS data format is one of the better outcomes as a widely used standard exists. However, the ultimate failure to easily and reliably parse data from a variety of instruments is unfortunately representative of the current state of biomedical data formats and tooling.

Consequently, we suggest that users research which data formats are supported by an instrument during the procurement phase, requesting that vendors provide sample data files and verify that common output file formats such as CSV, XML, JSON are provided by the instrument. If the raw data is stored in binary format, we strongly recommend customers obtain the data format specification and necessary software tooling to extract this information directly from the vendor at the time of procurement or evaluation.

### 3.3 Reverse Engineering File Formats Manually to Identify Metadata of Interest

Most data files created in a biomedical environment are instrument or manufacturer specific. Many of these contain important contextual information on the experiment, its conditions and instrument settings, which can be used to automatically label datasets and provide labeled training data to machine learning algorithms. In this section we introduce the basics of reverse engineering biomedical file formats to extract this contextual metadata.

Data files can be encoded in a variety of ways and fall in two broad categories: text and binary. Text files are typically comprehensible by a human, even if in a specific format such as CSV, XML or JSON, and opening them with a text editor will reveal the information. Binary files on the other hand often use a proprietary format and will not be readable using a text editor. Instead, these are typically readable using a hexadecimal editor [41, 42]. An example of each file type is shown in Fig. 6.

The two major obstacles in reverse engineering text-encoded files are (1) detecting the encoding used, and (2) mapping the variables to their real-world meaning (e.g., “ex\_wv” indicating “excitation wavelength”). Detecting the text encoding is often a matter of guessing correctly and testing common types such as ascii,

20	21	22	23		20	21	22	23									
70	3A	2F	2F	<DataFileMethodReport xmlns="http://domain.tld/DataFileReport.xsd">	00	00	00	00	S.mD..6?..3 F..@...B...@.0l ...@.								
0A	20	20	3C	<MethodReport> <Version>5.0</Version>	CB	A1	FC	40	.V*L..[?..cQ*..v. W ..*u=..sy8.V.								
2F	56	65	72	<MethodName>redacted.m</MethodName>	00	00	00	00	s. ...6?..@.rk...@...B								
6D	3C	2F	4D	<MethodPath>D:\Redacted\redacted.m</MethodPath>	86	A6	CE	42	..@.0l ...@.V*L..[?..cQ*..v. W ..*u=								
65	64	61	63	<RCDevicesXml>&lt;?xml version="1.0" standalone="yes"?>&lt;RCDsRdlReport	D3	2A	75	3D	..sy8.V. 1 m&..6?								
20	3C	52	43	xmlns="http://domain.tld/DSRdlReport.xsd">&lt;ReportInfo&gt;	DC	C8	36	3F	92	63	51	2A	00	00	00	00	..v. W ..*u=..sy8.V.
22	20	73	74	&lt;FooterText&gt;D:\Redacted\redacted.m</FooterText&gt; &lt;Version&gt;5.0</Version> &lt;ReportInfo&gt;	CB	A1	FC	40	./P...6?x ..g..@...B...@.0l ...@.V*L..[?..cQ*..v. W ..*u=..sy8.V.								
70	6F	72	74	&lt;MethodInfo&gt; &lt;MethodInfo&gt; &lt;Version&gt;5.0</Version> &lt;MethodPath&gt;D:\Redacted\redacted.m</MethodPath>	86	A6	CE	42	W-M..6?< h...@...B								
65	70	6F	72	&lt;MethodDesc&gt;Redacted description</MethodDescription> &lt;Device&gt; &lt;DeviceName&gt;HTC/HTS&lt;/DeviceName&gt; &lt;DeviceType&gt;Iso. Pump&lt;/DeviceType&gt; &lt;Quat&gt; Quat. Pump&lt;/Quat&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	D3	2A	75	3D	..@.0l ...@.V*L..[?..cQ*..v. W ..*u=								
0D	0A	20	20	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	DC	C8	36	3F	..sy8.V. . .6?G.}??1..@...B								
00	00	00	00	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	92	63	51	2A	00	00	00	00	..v. W ..*u=..sy8.V.				
72	65	64	61	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	CB	A1	FC	40	x ..x..6?yo..<..@...B...@.0l ...@.V*L..[?..cQ*..v. W ..*u=..sy8.V.								
6C	74	3B	56	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	86	A6	CE	42	.. .6?G.}??1..@...B								
20	26	6C	74	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	D3	2A	75	3D	..@.0l ...@.V*L..[?..cQ*..v. W ..*u=								
66	6F	26	67	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	DB	C8	36	3F	..sy8.V. e...6?								
65	72	73	69	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	92	63	51	2A	5..3J..@...B...@.0l ...@.V*L..[?..cQ*								
64	61	63	74	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	00	00	00	00	..v. W ..*u=..sy8.V.								
3B	4D	65	74	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	CB	A1	FC	40	♦r<..6? n..9..@...B...@.0l ...@.V*L..[?..cQ*..v. W ..*u=..sy8.V.								
6D	26	6C	74	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	86	A6	CE	42	.. r...6?...;..@...B								
44	65	73	63	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	D3	2A	75	3D	..@.0l ...@.V*L..[?..cQ*..v. W ..*u=								
6C	74	3B	2F	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	DC	C8	36	3F	..sy8.V. . -Ci..6?								
74	68	6F	64	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	92	63	51	2A	.. .@...B...@.0l ...@.V*L..[?..cQ*								
26	6C	74	3B	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	00	00	00	00	..v. W ..*u=..sy8.V.								
26	6C	74	3B	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	CB	A1	FC	40	.....6?..U..@...B...@.0l ...@.V*L..[?..cQ*..v. W ..*u=								
20	20	20	26	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	86	A6	CE	42	.. .&6?..0 ...@...B								
3B	0D	0A	20	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	D3	2A	75	3D	..@.0l ...@.V*L..[?..cQ*..v. W ..*u=								
74	3B	0D	0A	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;	00	00	00	00									
61	6D	65	26	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;													
76	69	63	65	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;													
20	4E	61	6D	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;													
3B	0D	0A	20	&lt;Device&gt; &lt;DeviceName&gt; &lt;DeviceType&gt; &lt;InstrumentName&gt; &lt;InstrumentType&gt; &lt;InstrumentName&gt;Instrument Name&lt;/InstrumentName&gt;													

**Fig. 6** Views of a text file (left) and a binary file (right). The last four columns of the hexadecimal view of each file can be seen on the left of each image. This hexadecimal data encodes text (ASCII) and binary data respectively

utf-8, or unicode [43]. Examples of text-encoded biomedical file formats include FASTQ [44], mzML [45], and VCF [46].

Binary file formats, on the other hand, can be very difficult to reverse engineer. The encoding used by the instrument vendor's engineers can be totally arbitrary, and even designed to make reverse engineering more difficult [47]. The first step in attempting to reverse engineer a binary file is to visualize it in a hexadecimal editor [41, 42]. In some cases, sections of the binary file will be as readable as they were when encoded as text. In this case, it is imperative to obtain multiple sample files to confirm whether these sections always appear at the same offset in the file, or whether the location of this text metadata is variable. If the offset is fixed, you can statically extract the data from the binary files by copying the relevant section. It is however more common that the offset is dynamic and dependent on preceding information. In this case, it is necessary to derive the correct offset locations dynamically, so the definition of a file format becomes crucial to successful reverse engineering.

Rather than creating new formats from scratch, instrument vendors often extend or use format-approved extensions to common file formats (e.g., TIFF image format) to store instrument-specific metadata. We have found it beneficial to look at these file formats as a source of experimental context. In one particular case, we were able to find all settings of an imaging device hidden in a deliberately obfuscated extension of the format. Extracting that information enabled us to create a real time dashboard of the instrument's settings and work around firmware and instrument software bugs that were causing frequent errors for the customer.

While reverse engineering can be a daunting task requiring both creativity and perseverance, the benefits of accessing an experiment's full context without relying on human annotation is a game changer for the use of machine learning approaches in the biomedical sciences. Automatic metadata extraction allows organizations to create orders of magnitude more labeled data than before. Using this approach, the author's organization has been able to—among other successes—generate models capable of predicting catastrophic biomanufacturing errors hours in advance and double the precision of widely deployed instruments [48].

### 3.4 Storing Metadata

After extracting metadata, it is important to consider how to store this information in a way that will enable scalable search. While the subject of full text search is a highly researched [49–51], and commercially developed topic [52, 53], we focus here on practical ways to implement a simple and versatile system. Versatility is particularly important as the types of insights scientific teams will want to extract from the centralized data will change over time and can be difficult to predict in advance. Therefore, we strongly recommend focusing on a storage strategy that prioritizes flexibility

and uses a common data format that is well supported by publicly available tools. One such format is JSON. Storing the extracted information in a JSON object will enable you to take advantage of rapidly evolving functionalities in Postgres [54] or use technologies such as Lucene [55] for scalable—albeit expensive in a commercial environment—full-text search. If you plan to use full text searches on metadata records in a cloud environment, we strongly recommend you review the security settings of systems such as Elastic-Search as they can be difficult to successfully secure [56].

### ***3.5 Designing for Interoperability Using Application Programming Interfaces (APIs)***

A key design paradigm that should be built in from the beginning is the creation of documented application programming interfaces between the different services discussed in this chapter. Without such design, organizations are vulnerable to undocumented and rapidly deteriorating software infrastructure. This is particularly true for environments that see programming staff turn-over.

By building in APIs for uploading, extracting metadata from a file and finally searching the extracted records, the resulting infrastructure can evolve in more predictable steps toward a robust environment. In order to achieve this goal, we recommend readers implement the OpenAPI Specification [57], a widely used tool with numerous programming language capabilities.

## **References**

1. Microsoft (2017) Microsoft Security Bulletin MS17-010—Critical. <https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010>. Accessed 3 Sept 2019
2. Microsoft (2018) Microsoft SMB Protocol and CIFS Protocol Overview. <https://docs.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-and-cifs-protocol-overview>. Accessed 29 Aug 2019
3. Rob Sobers (2019) CIFS vs SMB: What's the Difference? <https://www.varonis.com/blog/cifs-vs-smb/>. Accessed 29 Aug 2019
4. Wikipedia (2019) Network File System. [https://en.wikipedia.org/wiki/Network\\_File\\_System](https://en.wikipedia.org/wiki/Network_File_System). Accessed 3 Sept 2019
5. Wikipedia (2019) Andrew File System. [https://en.wikipedia.org/wiki/Andrew\\_File\\_System](https://en.wikipedia.org/wiki/Andrew_File_System). Accessed 3 Sept 2019
6. Paul Rubens (2019) SSD versus HDD Speed. <https://www.enterprisestorageforum.com/storage-hardware/ssd-vs-hdd-speed.html>. Accessed 29 Aug 2019
7. Microsoft Technet Blog (2010) SHA2 and Windows. <https://blogs.technet.microsoft.com/pki/2010/09/30/sha2-and-windows/>. Accessed 3 Sept 2019
8. Python Documentation (2019) Standard errno system symbols. <https://docs.python.org/2/library/errno.html>. Accessed 3 Sept 2019
9. Wikipedia (2019) Comparison of file transfer protocols. [https://en.wikipedia.org/wiki/Comparison\\_of\\_file\\_transfer\\_protocols](https://en.wikipedia.org/wiki/Comparison_of_file_transfer_protocols). Accessed 5 Sept 2019
10. Zeiss (2019) CZI Format License Request. <https://www.zeiss.com/microscopy/us/products/microscope-software/czi/czi-download.html>. Accessed 5 Sept 2019
11. Wikipedia (2019) bzip2 file compression. <https://en.wikipedia.org/wiki/Bzip2>. Accessed 5 Sept 2019
12. Wikipedia (2019) gzip file format and software application. <https://en.wikipedia.org/wiki/Gzip>. Accessed 5 Sept 2019
13. HighSpeedInternet (2018) Why does my internet slow down at night? <https://www.highspeedinternet.com/resources/why-does-my-internet-slow-down-at-night>. Accessed 5 Sept 2019

14. Amazon Web Services (2019) AWS Snowball: Physically migrate petabyte-scale data sets into and out of AWS. <https://aws.amazon.com/snowball/>. Accessed 5 Sept 2019
15. Microsoft Azure (2019) Azure Data Box <https://azure.microsoft.com/en-us/services/databox/>. Accessed 5 Sept 2019
16. Google Cloud (2019) Introducing the Transfer Appliance: Sneakernet for the cloud era. <https://cloud.google.com/blog/products/gcp/introducing-transfer-appliance-sneakernet-for-the-cloud-era> Accessed 5 Sept 2019
17. Microsoft (2014) Support for Windows XP ended. <https://www.microsoft.com/en-us/microsoft-365/windows/end-of-windows-xp-support>. Accessed 10 Sept 2019
18. U.S. Department of Health and Human Services Office for Civil Rights (2019) Breach Portal: Notice to the Secretary of HHS Break of Unsecured Protected Health Information. [https://ocrportal.hhs.gov/ocr/breach/breach\\_report.jsf](https://ocrportal.hhs.gov/ocr/breach/breach_report.jsf). Accessed 7 Sept 2019
19. Wikipedia (2019) Transport Layer Security. [https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security](https://en.wikipedia.org/wiki/Transport_Layer_Security). Accessed 10 Sept 2019
20. Google Security (2015) Maintaining digital certificate security. <https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html>. Accessed 12 Sept 2019
21. Zetter, K. (2013) Google Discovers Fraudulent Digital Certificate Issued for Its Domain. In: Wired Magazine. <https://www.wired.com/2013/01/google-fraudulent-certificate/>. Accessed 12 Sept 2019
22. Wikipedia (2019) Kazakhstan man-in-the-middle attack. [https://en.wikipedia.org/wiki/Kazakhstan\\_man-in-the-middle\\_attack](https://en.wikipedia.org/wiki/Kazakhstan_man-in-the-middle_attack). Accessed 12 Sept 2019
23. Wikipedia (2019) Forward secrecy. [https://en.wikipedia.org/wiki/Forward\\_secrecy](https://en.wikipedia.org/wiki/Forward_secrecy). Accessed 12 Sept 2019
24. Wikipedia (2019) Replay attack: network attack type. [https://en.wikipedia.org/wiki/Replay\\_attack](https://en.wikipedia.org/wiki/Replay_attack). Accessed 14 Sept 2019
25. Wikipedia (2019) Known-plaintext attacks. [https://en.wikipedia.org/wiki/Known-plain\\_text\\_attack](https://en.wikipedia.org/wiki/Known-plain_text_attack) Accessed 14 Sept 2019
26. Amazon Web Services (2006) Protecting Data Using Server-Side Encryption. <https://docs.aws.amazon.com/AmazonS3/latest/dev/serv-side-encryption.html>. Accessed 7 Sept 2019
27. Google Cloud (2019) Encryption at Rest. <https://cloud.google.com/security/encryption-at-rest/>. Accessed 7 Sept 2019
28. Microsoft Azure (2019) Azure Storage encryption for data at rest. <https://docs.microsoft.com/en-us/azure/storage/common/storage-service-encryption> Accessed 7 September 7th, 2019
29. Wikipedia (2019) BREACH: Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext. <https://en.wikipedia.org/wiki/BREACH>. Accessed 7 Sept 2019
30. Wikipedia (2019) CRIME: Compression Ratio Info-leak Made Easy. <https://en.wikipedia.org/wiki/CRIME>. Accessed 7 Sept 2019
31. Wikipedia (2019) IEEE-488: short-range digital communications bus specification. <https://en.wikipedia.org/wiki/IEEE-488>. Accessed 7 Sept 2019
32. Toga AW, Foster I, Kesselman C et al (2015) Big biomedical data as the key resource for discovery science. J Am Med Information Assoc 22(6):1126–1131. <https://doi.org/10.1093/jamia/ocv077>
33. Allotrope Foundation (2019) The Allotrope Framework and Data Format. <https://www.allotrope.org/allotrope-framework> Accessed 8 September 2019
34. Pistoia Alliance (2019) Unified Data Model. <https://www.pistoiaalliance.org/projects/udm/>. Accessed 8 Sept 2019
35. National Center for Advancing Translational Sciences (2013) Biomedical Data Translator. <https://ncats.nih.gov/translator/about>. Accessed 8 Sept 2019
36. Spidlen J, Moore W, Parks D et al (2010) Data file standard for flow cytometry, Version FCS 3.1. J Cytometry A 77(1):97–100. <https://doi.org/10.1002/cyto.a.20825>
37. Python Package Index (2018) FlowCytometryTools python package. <https://pypi.org/project/FlowCytometryTools/>. Accessed 8 Sept 2019
38. Teague B (2019) Cytoflow GitHub code repository. <https://github.com/bpteague/cytoflow>. Accessed 8 Sept 2019
39. FlowPy (2016) FlowPy Code Repository and Documentation. <http://flowpy.wikidot.com/>. Accessed 8 Sept 2019
40. Scott White (2019) FlowIO python library for flow cytometry. <https://github.com/whitews/flowio>. Accessed 8 Sept 2019
41. Ridiculous Fish (2019) Hex Fiend hexadecimal editor for Mac OS X. <https://ridiculousfish.com/hexfiend/>. Accessed 8 Sept 2019
42. MH-Nexus (2019) HxD Hexadecimal editor for Windows. <https://mh-nexus.de/en/hxd/>. Accessed 8 Sept 2019

43. Zentgraf D (2015) What Every Programmer Absolutely, Positively Needs To Know About Encodings And Character Sets To Work With Text. <http://kunststube.net/encoding/>. Accessed 8 Sept 2019
44. Wikipedia (2019) FASTQ text based format for storing biological sequences and quality scores. [https://en.wikipedia.org/wiki/FASTQ\\_format](https://en.wikipedia.org/wiki/FASTQ_format). Accessed 8 Sept 2019
45. HUPO Proteomics Standards Initiative (2017) mzML file format specification for raw spectrometer data. <http://www.psidev.info/mzML>. Accessed 8 Sept 2019
46. Samtools (2019) Variant Call Format specification. <https://samtools.github.io/hts-specs/VCFv4.2.pdf>. Accessed 8 Sept 2019
47. Fracchia C, Dapello J (2016) Reverse engineering biomedical equipment for fun and open science. Presented at DEFCON24—6 August 2016
48. BioBright (2017) Tools Bring Superpowers to the Biology Lab. <https://www.businesswire.com/news/home/20170314005466/en/BioBright-Tools-Bring-%E2%80%98Super%EF%BF%BDowers%E2%80%99-Biology-Lab>. Accessed 8 Sept 2019
49. Hearst MA et al. (2007) BioText Search Engine: beyond abstract search. *Bioinformatics* 23 (17):2348–2351. <https://doi.org/10.1093/bioinformatics/btm301>
50. Amer-Yahia S, Shanmugasundaram J (2005) XML full-text search: challenges and opportunities. *Proceedings Hearst MA, Divoli A, Guturu H et al. (2007) BioText Search Engine: beyond abstract search. *Bioinformatics* 23 (17):2348–2351 of 31st international conference on Very large databases 1368–1368*
51. Xu S, McCusker J, Krauthammer M (2008) Yale Image Finder (YIF): a new search engine for retrieving biomedical images. *Bioinformatics* 24(17):1968–1970
52. Amazon Web Services (2019) Elasticsearch Service: Fully managed, scalable, and secure Elasticsearch service. <https://aws.amazon.com/elasticsearch-service/>. Accessed 8 Sept 2019
53. Microsoft Azure (2019) How full text search works in Azure Cognitive Search. <https://docs.microsoft.com/en-us/azure/search/search-lucene-query-architecture>. Accessed 8 Sept 2019
54. Postgresql (2019) JSON Functions and Operators. <https://www.postgresql.org/docs/current/functions-json.html>. Accessed 8 Sept 2019
55. Apache (2019) Lucene text search engine project. <https://lucene.apache.org/core/index.html> Accessed 8 Sept 2019
56. Elasticsearch (2019) Configuring security in Elasticsearch. <https://www.elastic.co/guide/en/elasticsearch/reference/current/configuring-security.html>. Accessed 8 Sept 2019
57. Swagger (2019) OpenAPI specification documentation. <https://swagger.io/docs/specification/about/>. Accessed 8 Sept 2019



# Chapter 17

## AI-Based Methods and Technologies to Develop Wearable Devices for Prosthetics and Predictions of Degenerative Diseases

Mario Malcangi

### Abstract

Neurodegenerative diseases, mainly amyotrophic lateral sclerosis, Parkinson, Alzheimer, and rarer diseases, have gained the attention of healthcare service providers due to their impact on the economy of countries where healthcare is a public service. These diseases increase with aging and affect the neuromotor cells and cognitive areas in the brain, causing serious disabilities in people affected by them.

Early prediction of these syndromes is the first strategy to be implemented, then the developing of prostheses that rehabilitate motion and the primary cognitive functions. Prostheses could recover some important disabilities such as motion and aphasia, reduce the cost of assistance and increase the life quality of people affected by neurodegenerative diseases.

Due to recent advances in the field of artificial intelligence (AI) (deep learning, brain-inspired computational paradigms, nonlinear predictions, neuro-fuzzy modeling), the early prediction of neurodegenerative diseases is possible using state-of-the-art computational technologies. The latest generation of artificial neural networks (ANNs) exploits capabilities such as online learning, fast training, high level knowledge representation, online evolution, learning by data and inferring rules.

Wearable electronics is also developing rapidly and represents an important enabling technology to deploy physical and practical (noninvasive) devices using AI-based models for early prediction of neurodegenerative diseases and of intelligent prostheses.

Here we describe how to apply advanced brain-inspired methods for inference and prediction, the evolving fuzzy neural network (EFuNN) paradigm and the spiking neural network (SNN) paradigm, and the system requirements to develop a wearable electronic prosthesis for functional rehabilitation.

**Key words** Prosthesis, Prediction, Artificial neural network, Brain-inspired ANNs, EFuNN, SNN, Wearable electronics, Vital signs

---

### 1 Introduction

Population aging, leading to neurodegenerative disorders such as dementia (Alzheimer's disease, Parkinson's disease, cerebrovascular diseases, and cognitive impairments) which can progress to dementia, is becoming a serious public health issue. Studies have been conducted and drugs developed to reverse or to stop the

degeneration but such diseases are age-related and multifactorial (genetic, epigenetic, environmental, and lifestyle dependent). Thus, there are only two effective solutions: prevention and rehabilitation, that means: early predictions and prosthetics.

In the earliest stages of investigation on prediction methods for degenerative diseases, linear prediction, based on hardcomputing (Digital Signal Processing), was the best practice but the nonlinear nature of disease prediction made such methods challenging.

Linear versus nonlinear methods applied to prediction are discussed emblematically in [1], in which linear and nonlinear methods, respectively, linear regression and artificial neural networks, adaptive neural network inference systems, were compared and it lead to a good coincidence between the predicted results for all developed models. The accuracy was found to be better for nonlinear models, mainly for adaptive neural network inference systems.

In [2] there is an exhaustive example of an application of linear methods for prediction of Parkinson disease by means of utterance. The main issue concerns optimal extraction of speech features by linear predictive coding (LPC) and some variants of this feature extracting method (PLP, MFCC, RASTA).

Features extraction and matching applying linear methods (hardcomputing) is challenging because such information is embedded in continuous signal streams which are not synchronous with the events that we need to match. Linear methods are based on algorithms such as FFT, LPC, and other windowing-based algorithms, so the primary issue consists in end-pointing the signal embedding the features.

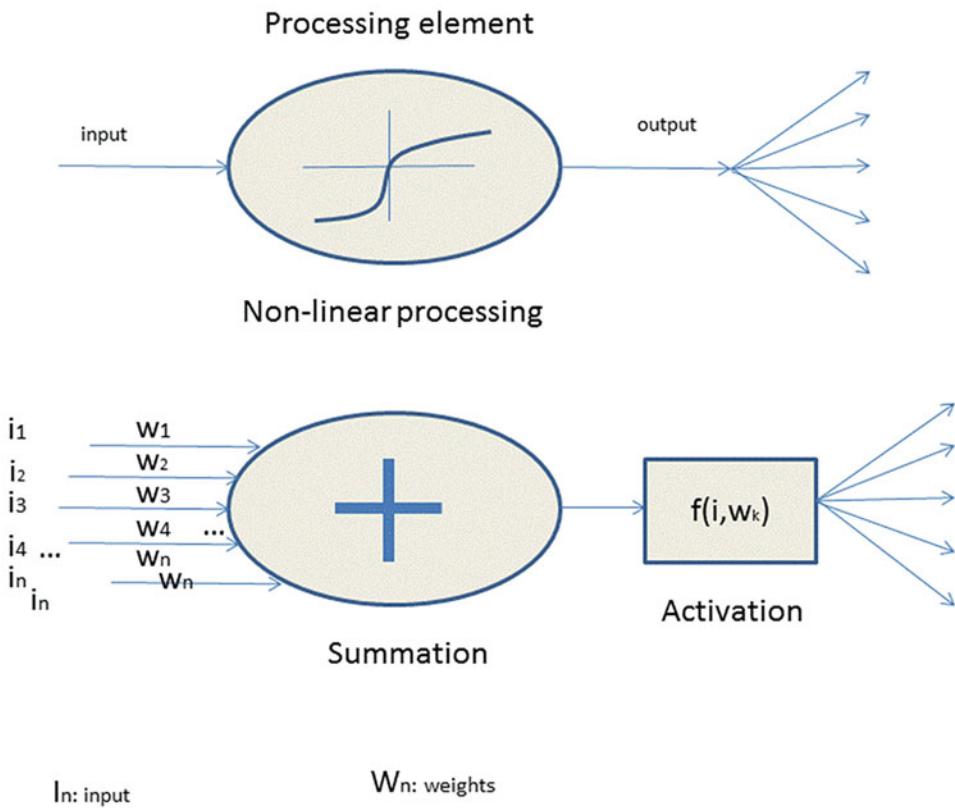
By applying nonlinear methods such as those of softcomputing, mostly the brain-inspired paradigms, this issue can be overcome so that features can be automatically extracted from continuous signals and online learning and matching can be applied.

## 2 Bioinspired Softcomputing Methods

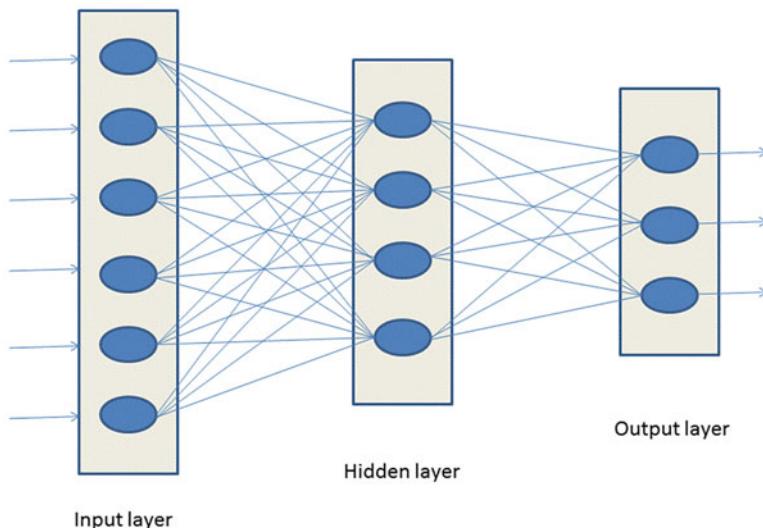
Artificial neural networks (ANNs) are universal parallel computing methods based on processor elements (PEs) Fig. 1 that can be infinitely networked by weighted connections (the synapses), Fig. 2. The PEs model, developed by Mc Culloch and Pitts in 1943, is similar to thresholded logic gates that led to the Multilayer Perceptron (the most popular ANN).

ANNs, were developed to learn by inspection of data, but the early paradigms (SOM, MLP, CNN, RNN) lacked the ability to handle the evolving nature of processes that involve spatiotemporal information.

Another issue concerning early ANN paradigms is the learning method: supervised or unsupervised. Supervised learning is learning by inspection of examples. Data objects (patterns) are



**Fig. 1** The processing element (PE) is a nonlinear processor that combines the linear operation (summation) with a nonlinear operation (activation)



**Fig. 2** The artificial neural network (ANN) is a highly parallel computing architecture in which PEs are grouped in fully or partially interconnected layers. The figure illustrates a feed-forward network

classified (labeled) in advance by an expert and fed to the ANN to set up its knowledge. By contrast, unsupervised learning is learning from data (not previously classified), applying data mining methods (clustering, vector quantization) that explores the data to find hidden features or grouping. Self-organizing maps (SOMs) could be considered one of the first successful approaches to unsupervised learning issues, since this paradigm is able to extract the bidimensional distribution of features from data. Kohonen [3] successfully applied SOMs to the “The Neural Phonetic Typewriter” phonetic speech-to-text automatic speech recognizer.

Evolving connectionist systems (ECOSs) [4] were developed further to enable ANNs to learn in adaptive and incremental mode from data that embeds the evolving featuring of the process. ECOSs paradigms were capable of extracting linguistic rules from data but not knowledge.

ANNs and ECOSs have been merged in new highly performing and bioinspired paradigms: the evolving fuzzy neural networks (EFuNNs) [5] and the spiking neural networks (SNNs) [6].

## 2.1 Evolving Fuzzy Neural Networks

The evolving fuzzy neural network (EFuNN) [5, 7] is a particular implementation of the ECOS [4] (evolving connectionist system) a biologically inspired framework [7]. This softcomputing paradigm evolves through incremental, online learning, both supervised and unsupervised.

EFuNNs are FuNNs that evolve online. FuNNs combine the ANNs paradigm with the fuzzy logic (FL) paradigm so that connections and nodes can change during the operation. The ANN embedded in the FuNN has a five layer feed-forward architecture (Fig. 3).

The first layer is the input layer to which the data are fed. The second is the membership layer, which implements the membership functions that enable the fuzzification of the crisp data at the first layer. The third layer implements the fuzzy logic rules, creating clusters of input–output association. Rules and connections evolve through learning, aggregating and pruning the nodes. The fourth layer calculates the match of the input data to the output through the output membership functions. The fifth deploys defuzzing to calculate the output crisp values.

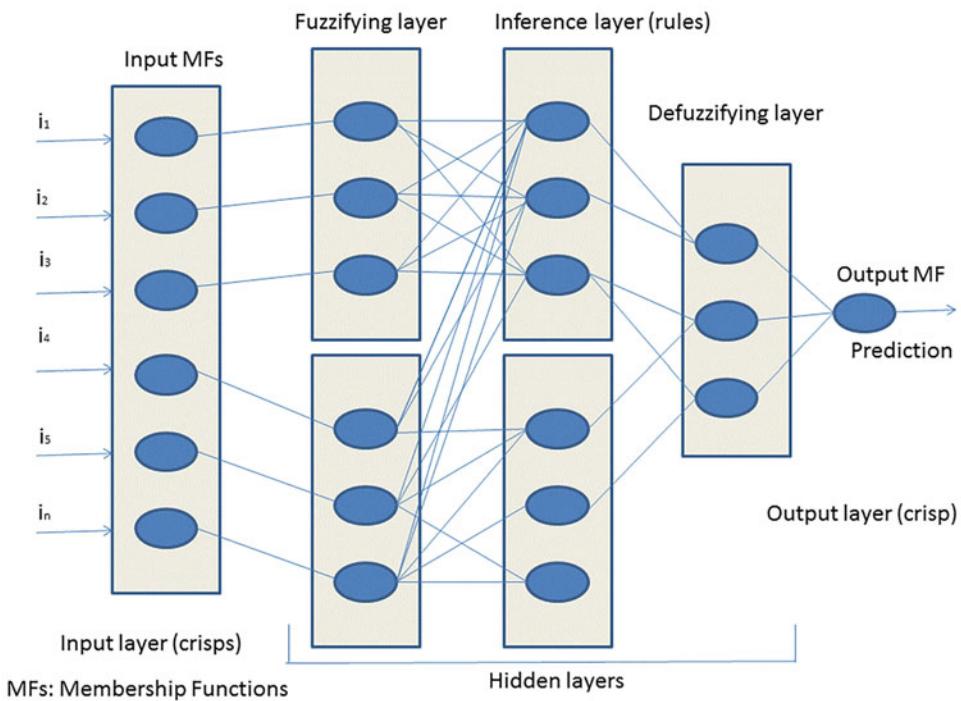
The EFuNN is able to evolve (the number of nodes and connections can change during operation) so that the rules and the memberships can adapt to new data at training time in incremental mode. The EFuNN learns from its data like ANNs and infers by rules like a fuzzy logic engine.

The EFuNN learning process (training) consists in preparing a labeled dataset such as (Fig. 4) the following:

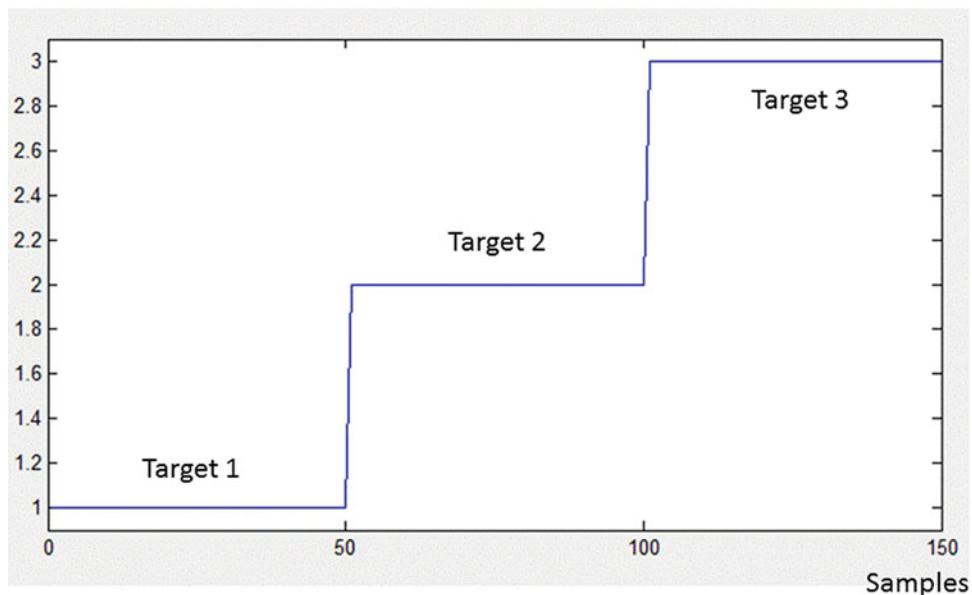
$V_1 \ V_2 \ V_3 \ V_4 \ V_5 \ V_6 \ V_7 \ V_8 \ V_9 \ V_{10} \ V_{11} \ V \dots \ V_j \dots \ V_N \dots L_n$

$V_j$ :  $j$ -th amplitude of the  $j$ -th sample of the  $n$ -th pattern

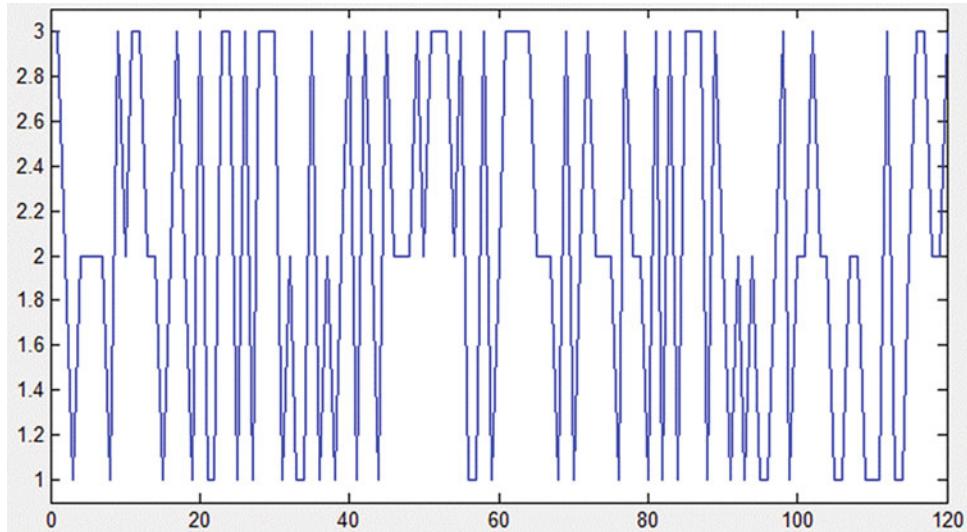
$L_n$ :  $n$ -th label associated to the  $n$ -th sequence.



**Fig. 3** EfUNN is a five layers Artificial Neural Network where each layer corresponds to a layer of a fuzzy logic engine



**Fig. 4** A dataset consisting of several labeled data sequences (samples) [8]



**Fig. 5** The dataset to train the ANN is split (80/20%) and randomized [8]

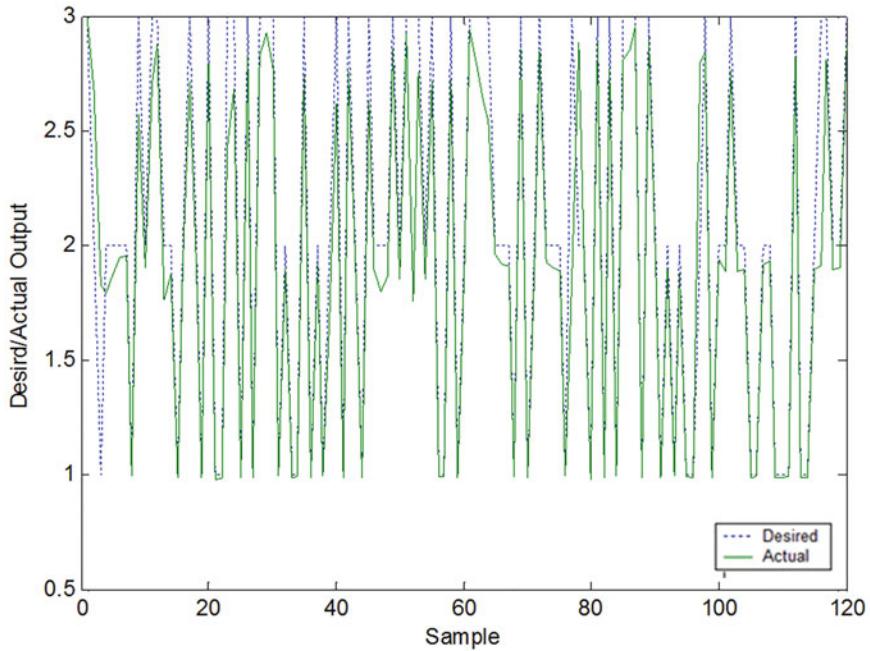
The following training and test examples have been simulated in the NEUCOM environment developed at the Knowledge Engineering & Discovery Research Institute [8] Auckland University of Technology. Feeding the data vectors to the EFuNN inputs executes one-step training (using a sample of 80% randomly extracted from the full dataset) (Figs. 5 and 6).

Then, the test (Fig. 7) is executed to ensure that the EFuNN has learned effectively and that it is able to infer. This uses a subset of the training dataset (20% randomly extracted from full dataset) (Figs. 7 and 8). If the test is successful then the EFuNN is ready to run. The execution code and the post training and test settings can be deployed on a processor to tackle run-time and real-time incoming data.

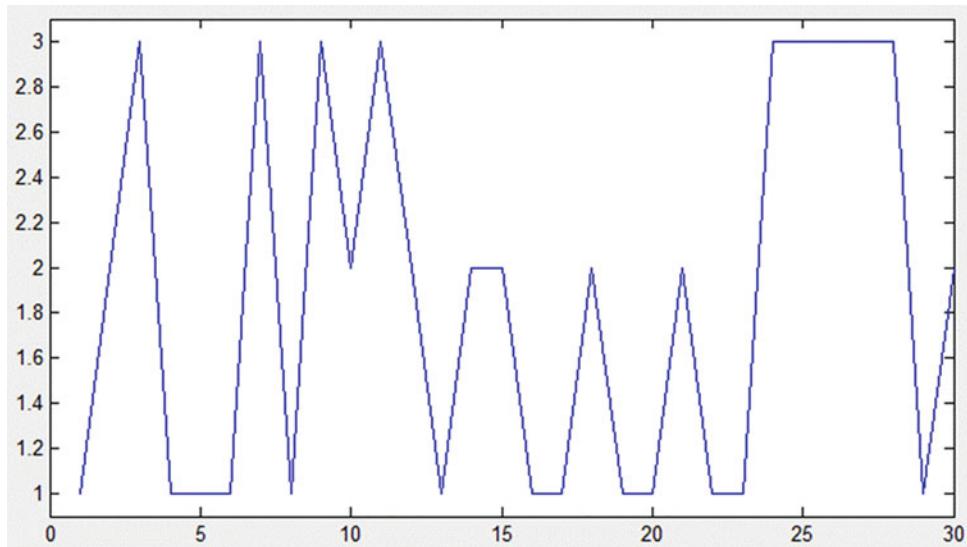
## 2.2 Spiking Neural Networks

Spiking neural networks (SNNs) [6] are bioinspired ANNs that encode and process the information as binary events (spikes) (Fig. 9). A biological example is the auditory system that encodes a sound wave as spikes in the cochlea by the hair cells connected to the auditory nerves that feed the auditory area of the cerebral cortex, where the audio signal is processed as spikes to match audio information (sounds, speech, acoustical vibrations, etc.).

The advantages of SNNs over other ANN paradigms are in its capability to encode and process the information as spikes. This implies a more compact representation of the information and asynchronous data processing, efficient real-time prediction, massive parallel processing, and energy efficiency.

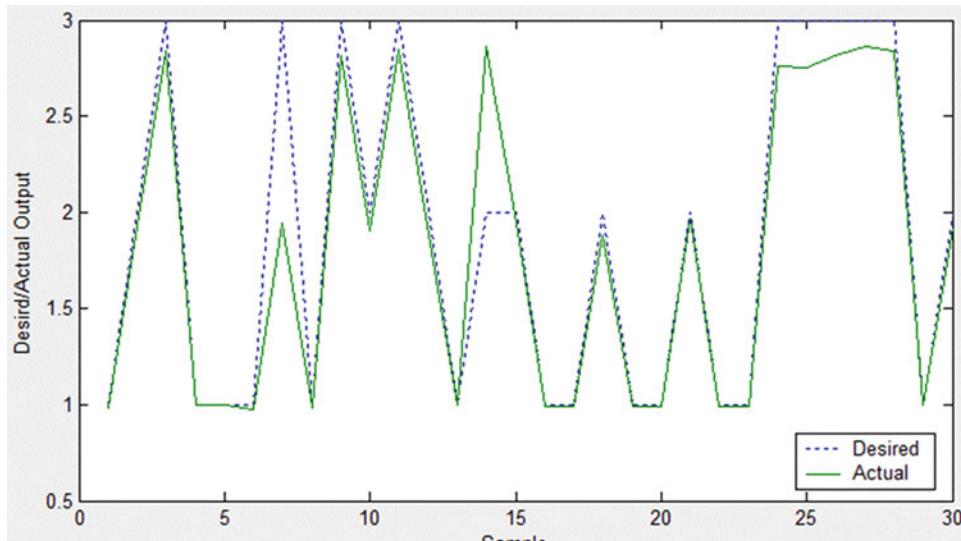


**Fig. 6** The split and randomized dataset (80%) used to train the EFuNN [8]

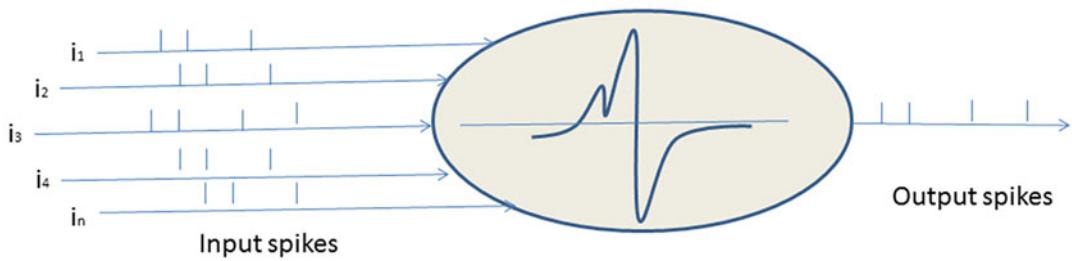


**Fig. 7** Dataset: split part (20% of the full dataset) used to test the EFuNN

The main issue of the SNN's implementation is the encoding of the real physical data into spikes in real-time according to the bio-inspired requirements of the prediction task to be accomplished.



**Fig. 8** The split and randomized (20%) dataset after the testing of the trained EFuNN [8]



**Fig. 9** Spiking Neural Network (SNN)

### 3 Technologies

Three main technologies have been developed to meet the requirements of the wearable designs for prosthetics and prediction of degenerative diseases: analog front-ends (AFEs), sensors, and application specific processors (ASPs). The primary requirements are ultrasmall size and ultralow power.

#### 3.1 Analog Front-End

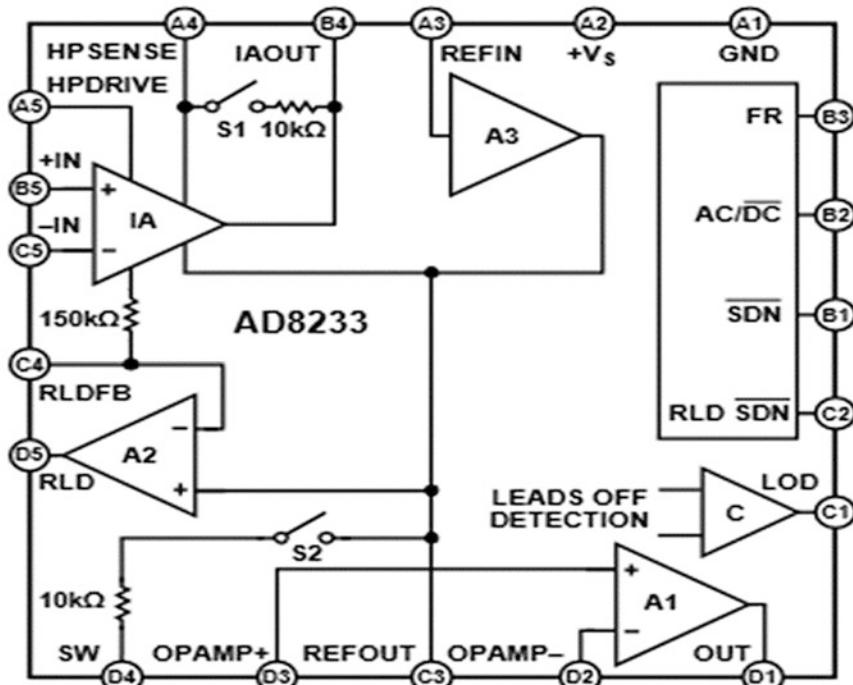
The bioelectric (biopotential) signals (electroencephalographic, electrocardiographic (ECG), electromyographic) are the primary sources of information to be processed by an ANN paradigm trained to predict. Such signals are analog, very low in amplitude (a few microVolts), and very noise sensitive. Consequently, they require special purpose analog electronics, namely Instrumentation

Amplifiers (IA), to recover the effective signal from the captured bioelectric signal, using a special amplification technique, Common Mode Rejection [9, 10].

Artifacts, power line and Radio Frequency interferences require the application of analog signal processing methods such as special purpose analog filters made by linear (analog) operational amplifiers and passive electronic components (resistor and capacitors).

All these requirements for capturing bioelectric signals define the analog front-end (AFE) subsystem, a complex analog electronic system. To meet the wearable requirements (ultrasmall size and ultralow power) the AFEs must be developed applying the ultimate microelectronic technologies (nanometric electronic integration processes).

Analog Devices Inc. (ADI) has a single ECG amplifier called AD 8233 (Fig. 10) which is an example of an industry standard AFE [11] targeted to wearable applications due to its ultrasmall size ( $2 \text{ mm} \times 1.7 \text{ mm}$ ) packaged in a Chip-Scale package (WLCSP), and its ultralow power consumption ( $50 \mu\text{W}$ ).



**Fig. 10** The analog front-end (AFE) AD 8233 from Analog Devices Inc. (ADI) integrates in a single Chip Scale Package with all the analog electronics (operational amplifiers) required to transform the bioelectric signal to the electric bipolar signal applicable to the analog-to-digital converter (ADC) (Courtesy of Analog Devices Inc. [www.analog.com](http://www.analog.com))

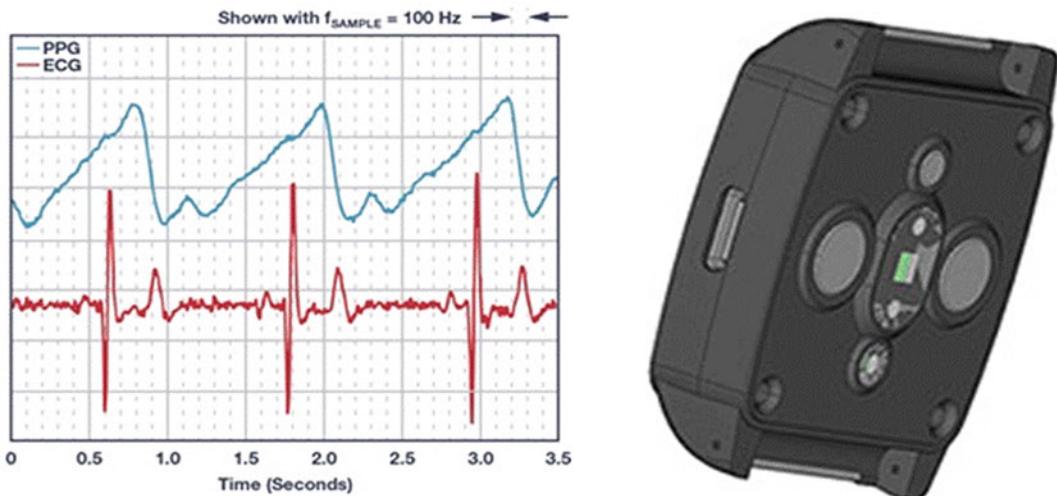
### 3.2 Sensors

#### 3.2.1 Photometric Sensors

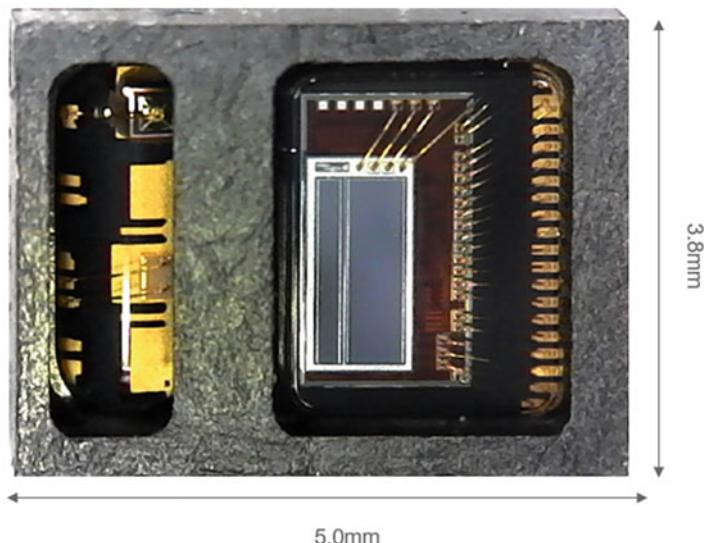
To capture the bioelectric signal in a clinical application (rest conditions) contact electrodes are the common practice. Contact electrodes are invasive and not suitable for wearable applications. Due to their weak contact with the skin, they require special conductive paste or gel to be applied on the skin for optimal conductivity.

A new class of sensors and methods for vital sign measurement has been developed so that electrodes are no longer required for ECG, heart rate measurements and other bioelectric measurements. The most popular is “photometric sensing” that uses light intensity for the measurements. A light emitting diode (LED) generates light, which is sent into the body, and a photodiode receives the reflected or transmitted light whose intensity is modulated by the transparency or opacity of the material being measured. Multiple LEDs of selected wavelengths pulse the light through the body tissue while the photodetector measures the resultant signal, which depends on the absorption of the light as a result of blood flow. The current is extremely small (a few nanoAmperes), so a transimpedance amplifier (TIA) is required to transform the very small photocurrent to a voltage, minimizing noise.

Analog devices developed the photometric front-end (PFE) ADPD107 [11] (Fig. 11), a fully integrated AFE including TIA, Ambient light rejection stage, ADC and LED drivers. It also has an on-board digital state machine, taking care of the timing, all in a tiny WLCSP package ( $2.46\text{ mm} \times 1.4\text{ mm}$ ) (Fig. 12). The ADPD



**Fig. 11** The Photometric Front-End (PFE) is the optical photoplethysmogram (PPG) noninvasive alternative to electric measurement of heart activity (ECG). The ADPD107 by Analog Devices Inc.(ADI) was integrated in a system-on-module (SoM) assembled in a watch (ADI GEN II) including two MCUs, the wireless communication and the AFE for vital sign measurements (Courtesy of Analog Devices Inc. [www.analog.com](http://www.analog.com))



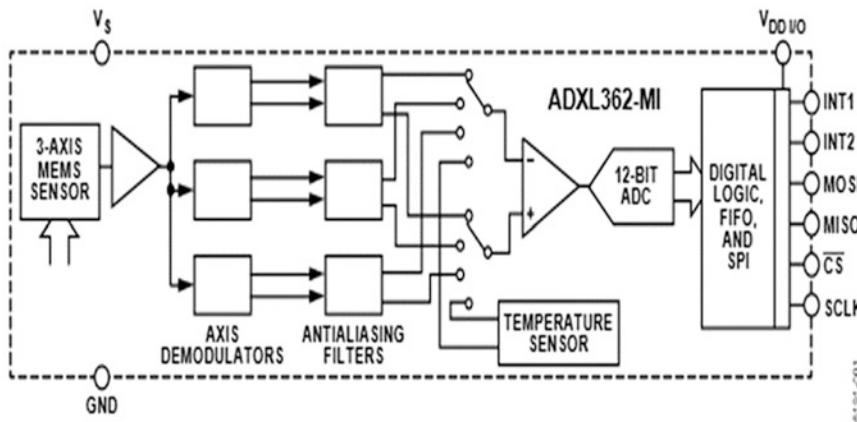
**Fig. 12** The highly integrated ADPD188GGZ module. This ultrasmall module includes the photometric front-end, photodiodes, and LEDs for PPG measurements. It is targeted for heart rate monitoring and heart rate variability (HRM/HRV). When other wavelengths are required, for example, for SpO<sub>2</sub> measurement, the module can drive external LEDs as well (Courtesy of Analog Devices Inc. [www.analog.com](http://www.analog.com))

107 integrates three 8 mA to 250 mA LED drivers. Heart rate and other vital signs can be measured with optical sensing rather than using contact electrodes, with very low invasiveness.

### 3.3 Inertial Sensors

Inertial sensors (accelerometers) are sensors that assist in data gathering for prediction activities. Inertial sensors (vibrational sensors) enable actigraphy, a noninvasive method for the measurement of human body rest/activity. Inertial sensors are mainly built on a Micro Electro Mechanical Systems (MEMS) technology that deploys all the required mechanical and electronic functions in very small ultralow power packages (a few millimeters square with under 1 µW consumption).

Analog Devices Inc. developed a very high performing accelerometer targeted to wearables, the ADXL362 (Fig. 13), a digital output MEMS [12], ultralow power (less than 2 µA at a 100 Hz output data rate on all 3 axis, 270 nA, in motion activation wake-up mode and 10 nA standby). Measurement ranges of ±2 g, ±4 g, and ± 8 g are available, with a resolution of 1 mg/LSB on the ±2 g range. The ADXL362-MI is a medical grade variant, targeted for implantable devices and available in a 3 mm × 3.25 mm × 1.06 mm LGA package.



**Fig. 13** The MEMS accelerometer ADXL362 from Analog Devices Inc. is an high-performing motion sensor of ultralow power in a 3 mm × 3.25 mm × 1.06 mm LGA package, suitable for wearable and motion measurements (actigraphy) (Courtesy of Analog Devices Inc. [www.analog.com](http://www.analog.com))

### 3.4 Application Specific Processors

ASPs are a class of computing architectures designed to perform efficiently for specific applications, in contrast to general-purpose processors.

Digital Signal Processors (DSPs) were the first processors with a computational architecture specifically designed to perform efficiently the mathematics of signal processing. DSPs are Reduced Instruction Set Computers based on Harvard Architecture and system-on-chip microelectronic process.

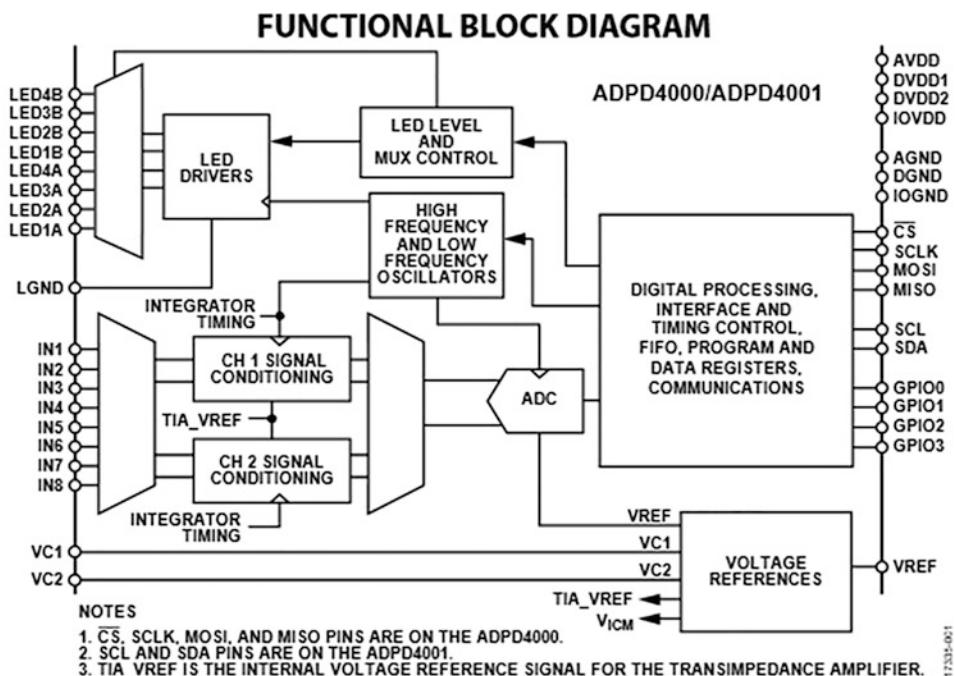
Softcomputing paradigms (ANNs and Fuzzy Logic) require specific computing operations that cannot be performed efficiently by a general purpose computer, so DSPs were first applied to deploy ANNs [13] and Fuzzy Logic to perform in real-time but DSPs are poorly suited to softcomputing so new application specific computing architectures were developed, such as application specific integrated circuits (ASICs). DSPs and ASICs meet the computing requirements of ANNs and fuzzy logic, but have high power demands, are expensive and large, and are consequently not suitable for wearable systems. ARM based MCUs, specifically the ARM Cortex-M core architecture, has been optimized for compute intensive applications. The architecture is a tradeoff between DSPs and general purpose MCUs. Most commercial MCUs embed a Cortex M core, a rich set of peripheral and a large memory footprint in very small packages. These MCUs are very low power and can execute an ANN-based application in real time.

An ultrasmall, ultralow power MCU ARM cortex M0+ core-based is the KL03 developed and produced by Freescale/NXP specifically for wearable applications. This 32-bit MCU is packaged in a Chip Scale Package (CSP) format (2 × 1.61 × 0.56 Pitch 0.4 mm) with an ultralow power consumption (77 nA in deep sleep, 50 µA/MHz).

### 3.5 Biomedical Sensor Front-End

The development of miniaturized, intelligent, and wireless sensors is a primary requirement for wearable system AI-based e-health applications. Multiple data gathering is needed in machine learning [14] e-health applications, because of the multifactorial dependency of the prediction frameworks. Multimodality in sensor data acquisition enables the application of new methods in vital sign measurements and processing; for example, ECG combined with PPG enables continuous and noninvasive blood pressure measurement [15].

An important innovation comes from Analog Devices Inc. who developed a biomedical sensor front-end (the ADPD4000/ADPD4001) on a single chip ( $3.11\text{ mm} \times 2.14\text{ mm}$ ,  $0.4\text{ mm}$  pitch, 33-ball WLCSP and 35-ball WLCSP) (Fig. 14). The ADPD4000-family operates as a complete multi parameter sensor front-end that enables different sensing modes (optical and bio-electric) and different measurements (ECG, PPG, EDA, impedance, and temperature). The ADPD4000/ADPD4001 is more than a multiple sensor system, as it also integrates all the functions required for interfacing with a digital system (AFE, ADC, serial ports, digital state machine). Additional processing functions such as digital filtering and ambient light rejection (60 dB up to 1 kHz) are available on chip.



**Fig. 14** Multimodal sensor front-end ADPD4000/ADPD4001 (Courtesy of Analog Devices Inc. [www.analog.com](http://www.analog.com))

The ADPD4000/ADPD4001 are effectively multimode measuring devices that can interface with many different sensors enabling synchronous measurements of PPG, electrocardiography (ECG), electrodermal activity, impedance, capacitance, and temperature measurements. A selection of operating modes has been built in to optimize each of the different sensor measurements supported.

A key feature of ADPD4000 and ADPD4001 is that they put on a single chip (few mm square) all the electronics currently available only on a system module (few cm square). This is a big step toward an integrated wearable system.

## 4 Application Frameworks

AI-based methods have been applied to successfully deploy prosthesis applications and diseases or physiologic status prediction applying the embedded electronics and the new emerging brain-inspired computing paradigms. We now briefly describe two case studies, the first concerning otorhinolaryngology (voice-impaired people) [16], the second concerning neurology [17, 18] (sleep onset in drivers).

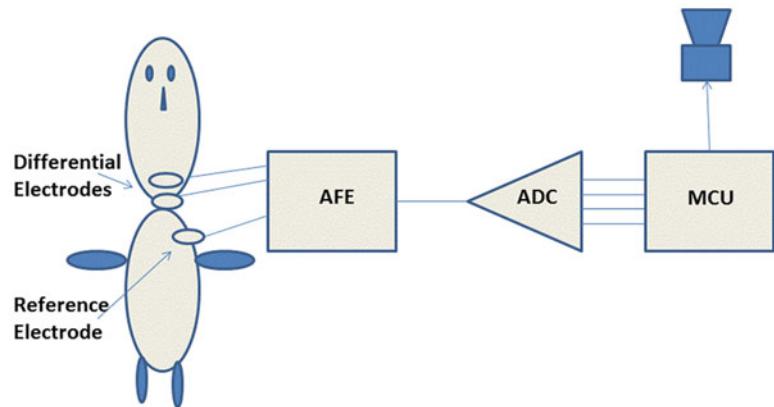
### 4.1 Prosthesis

For several clinical reasons, mainly larynx cancer, people lose vocal capabilities. Some (effective and less effective) prostheses have been proposed, involving the brain speech production area of the cortex (Broca), which can control the residual parts of the phonation organ (tongue, mouth cavities, nose, lung). Phonation activity in humans is a complex neural process that links language (utterance articulation) and neuromotor control. To each vocal sound there corresponds a sequence of myoelectric signals that drive the muscles to articulate the desired sound. From the myoelectric signal of the facial muscles we can predict the intended vocal sound. The predicted sound can be then artificially synthesized acoustically and transduced by a loudspeaker.

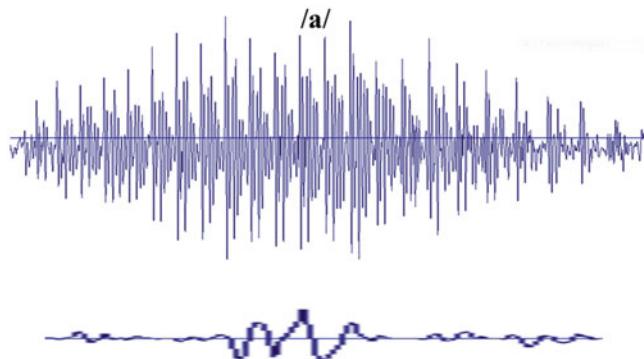
The framework to deploy a prosthesis for voice impaired people consists of the following steps:

- Build a labeled dataset to train a predicting ANN paradigm;
- Train & test the paradigm to predict the intended vocalization;
- Drive a speech synthesizer to generate the sound;
- Apply the sound to an amplified loudspeaker.

To build the labeled dataset the myoelectric signal must be captured by the muscle involved in the utterance, so a myoelectric data acquisition (DAQ) system must be developed as shown in Fig. 15:



**Fig. 15** Myoelectric data acquisition (DAQ) framework



**Fig. 16** Utterance of the voiced sound (vocal) /a/ and the corresponding myoelectric pattern

There are three surface electrodes: two for differential signal detection and one for reference:

- One analog front-end (AFE).
- One analog-to-digital (ADC) subsystem microcontrolled by an MCU (microcontrolled unit).

The surface electrodes are applied to the skin near the muscle involved in the utterance; two 1–2 cm apart, close to the muscle, and a third, reference electrode, far from them (on the shoulder—clavicle). The electrodes are wired to the AFE inputs, so a bipolar myoelectric signal is available at AFE outputs ready to be sampled by the ADC.

To window and label each myoelectric pattern with its corresponding voice sound code (e.g., the phoneme code) it is necessary to synchronously capture the myoelectric pattern corresponding to the audio (Fig. 16).

The dataset consists of a large number of patterns such as the following:

$$V_1 \ V_2 \ V_3 \ V_4 \ V_5 \ V_6 \ V_7 \ V_8 \ V_9 \ V_{10} \ V_{11} \ V \dots \ V_j \dots \ V_N \dots L_n$$

$V_j$ :  $j$ -th amplitude of the  $j$ -th sample of the  $n$ -th pattern

$L_n$ :  $n$ -th label associated to the  $n$ -th sequence.

Then it is possible to proceed to train the ANN (in this case the ANN is the EFuNN predicting paradigm).

After training we test the ability of the EFuNN to predict, then deploy the trained EFuNN and its setup on a MCU able to run it in real-time fed by the incoming myoelectric signal when a voice impaired subject tries to utter.

If wireless electrodes are available and a wrist worn MCU is wirelessly connected to them, a noninvasive voice prosthesis is then ready to run.

## 4.2 Prediction

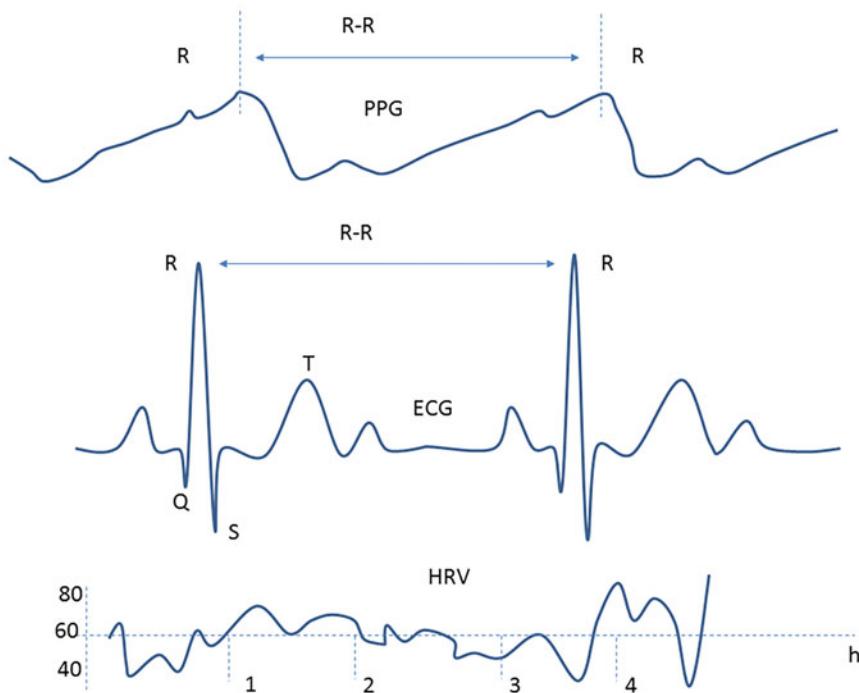
Prediction could be applied to the physiologic status and/or diseases of a subject. An interesting case of prediction related to an incoming physiologic status is the prediction of sleep onset of car drivers. This could be accomplished by a system that measures vital signs such as heart rate and its variability, applying a predictive ANN paradigm.

Drowsiness is under the control of the Autonomous Nervous System (ANS) that also controls the cardiac rhythm, and other physiologic activities such as respiration and galvanic skin response (GSR). A practical approach to deploy a system for prediction of the sleep onset could be based on measurement of heart rate variability because ANS activity is directly reflected in the variability of the HR. Sleep and wakefulness are directly related to the ANS activity balanced between the sympathetic and the parasympathetic modulation activities of the ANS. The awake state is controlled directly by the sympathetic modulation activity, and the sleep state by the parasympathetic modulation activity. So the transition from awake to sleep (sleep onset) corresponds to inversion of the balancing of this modulation activity.

Because the ANS modulation activity (sympathetic and parasympathetic) acts on HR, measuring the HRV it is possible to detect this inversion and to predict the sleep onset status.

To deploy a system for the early prediction of sleep onset of car drivers, the following framework is required:

- Build a labeled dataset to train an ANN paradigm for prediction;
- Train & test the paradigm that will predict sleep onset;
- Run the ANN on a MCU for real-time prediction and alerting;
- Deploy the system by wearable electronics.



**Fig. 17** R-R intervals measured from PPG and ECG; HRV is the sequence of R-R intervals

To build the labeled dataset the HR need to be measured. A set of surface contact metal electrodes for ECG measurements applied to the chest, or less invasively the PFE applied to the wrist or to the finger enables the measurement of R-R (ECG) (Fig. 17) intervals by the photoplethysmogram (PPG) to sample the HRV. Each time sleep onset occurs the HRV has been sampled and labeled.

The labeled dataset consists of a large set of labeled measurements (wake/sleep onset) such as:

$$\begin{aligned} V_1 & \ V_2 \ V_3 \ V_4 \ V_5 \ V_6 \ V_7 \ V_8 \ V_9 \ V_{10} \ V_{11} \ V_j \dots \ V_N \dots L_n \\ V_j: & j\text{-th amplitude of the } j\text{-th sample of the } n\text{-th pattern} \\ L_n: & n\text{-th label associated to the } n\text{-th sequence (awake, sleep onset).} \end{aligned}$$

When the dataset is ready, the ANN can be trained and tested. If the test is satisfactory, the ANN could be tested on real data and executed in real-time by a wearable MCU connected to the PFE.

---

## Acknowledgments

A special acknowledgment is due to Prof. Nikola Kasabov, Auckland University of Technology, Director KEDRI—Knowledge Engineering and Discovery Research Institute, for his invaluable suggestions on how to get the most from the EFuNN's evolving capabilities.

Acknowledgment is also due to Jan Hein Broeders (Analog Devices's healthcare business-development manager for EMEA) for his support and expertise in hardware prototyping, especially on the analog front-end subsystem.

## References

1. GómezSarduy JR, Gregio Di Santo K, Saidel MA (2016) Linear and non-linear methods for prediction of peak load at University of São Paulo. *Measurement* 78:187–201
2. Orozco-Arroyave JR, Arias-Londoño JD, Vargas-Bonilla JF et al (2013) Perceptual analysis of speech signals from people with Parkinson's disease. In: Ferrández Vicente JM, Álvarez Sánchez JR, de la Paz López F et al (eds) Natural and artificial models in computation and biology. IWINAC 2013. Lecture notes in computer science, vol 7930. Springer, Berlin, Heidelberg
3. Kohonen T (1988) The “neural” phonetic typewriter. *Computer* 21:1122
4. Kasabov N (2007) Evolving connectionist systems: the knowledge engineering approach. Springer, Heidelberg
5. Kasabov N, EFuNN, IEEE Tr SMC, 2001
6. Kasabov N (2019) Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence. Springer Series on Bio- and Neurosystems. Vol. 7, Springer-Verlag Berlin Heidelberg
7. Kasabov N (1998) Evolving fuzzy neural networks—algorithms, applications and biological motivation, In: Yamakawa and Matsumoto (eds.), Methodologies for the conception, design and application of the soft computing, World Computing 271-274
8. <http://www.kedri.aut.ac.nz/areas-of-expertise/data-mining-and-decision-support-systems/neucom>
9. Fogelman E, Galton I (2001) A digital common mode rejection technique for differential Analog-to-digital conversion. *IEEE Trans Circuits and Systems II* 48(3):255–271
10. Fortunado K (2018) Programmable gain instrumentation amplifiers: finding one that works for you. *Analog Dialogue* 52:4
11. Broeders J (2014) Wearable electronic devices monitor vital signs, activity level, and more. In *Analog Dialogue* 48(12). <https://www.analog.com/media/en/analog-dialogue/volume-48/number-4/articles/wearable-electronic-devices.pdf>
12. Murphy C (2017) Choosing the Most suitable accelerometer for your application. In: *Analog Dialogue* 51(4). <https://www.analog.com/media/en/analog-dialogue/volume-51/number-4/articles/choosing-the-most-suitable-mems-accelerometer-for-your-application-part-2.pdf>
13. Hämäläinen T, Klapuri H, Saarinen J et al (1997) Mapping of SOM and LVQ algorithms on a tree shape parallel computer system. *Parallel Comput* 23(3):271–289
14. Mincholé A, Camps J, Lyon A et al (2019) Machine learning in the electrocardiogram. *J Electrocardiol* doi. <https://doi.org/10.1016/j.jelectrocard.2019.08.008>. [Epub ahead of print]
15. Liang Y, Chen Z, Ward R et al (2018) Hypertension Assessment via ECG and PPG Signals: An Evaluation Using MIMIC Database. *Diagnostics* (Basel). MDPI;8(3):65
16. Malcangi M, Felisati G, Saibene A et al (2018) Myo-to-speech – evolving fuzzy-neural network prediction of speech utterances from myoelectric signals (Communications in Computer and Information Science). In: engineering applications of neural networks E. Pimenidis, C. Jayne (eds.) Springer nature, 158–168
17. Malcangi M (2016) Applying evolutionary methods for early prediction of sleep onset. *Neural Computing and Application* 27 (5):1165–1173
18. Malcangi M, Smirne S (2012) Heart rate variability analysis for prediction of sleep onset in car drivers. *J. sleep res.* 21(Supp. 1):307–308

# INDEX

## A

- Activation function ..... 58, 116, 120, 122, 212, 217, 218, 252, 309, 312  
Algorithm  
    evolutionary ..... 115–120, 144–147  
    genetic ..... v, 117, 146  
    Levenberg–Marquardt ..... 120, 307–314  
    neuro-evolutive ..... 115–136  
Alzheimer ..... 82, 230, 232, 337  
Amino acid ..... 1, 2, 4, 6, 12, 14, 15, 21–23, 26, 28, 126, 270, 272, 274–276, 278, 286  
Arpeggio ..... 2, 6–10, 18, 25  
Artificial intelligence (AI) ..... v, 161, 169, 171–174, 211, 215, 337–353  
Assay ..... 2, 210  
Atom-based method ..... 141  
Autocorrelation ..... 35  
Autoencoder ..... 149, 153, 154, 158, 169, 171, 174, 214–215, 221, 223

## B

- Backpropagation ..... 65, 124, 125, 152, 154, 158, 171, 180, 212  
Bacteria ..... v, 95–113, 122, 126, 250  
Bacterial promoter ..... 96  
Bandwidth ..... 317, 319, 321–324  
Binding ..... 18, 142, 145, 159, 160, 173, 192, 231–233, 235–237, 239–245, 268  
Binding affinity ..... 10, 18, 145, 159  
Bioinformatics ..... 74, 96, 160, 168, 179, 229, 237, 330  
Biomarker ..... 171, 173, 174, 195–205, 210, 225  
Biomedical  
    literature ..... 289–302  
    ontologies ..... 290, 291, 293, 296, 298–300, 302  
Biomedicine ..... 173, 179  
Bioreactor ..... 117, 119, 121, 123–126  
Biosensor ..... 33  
Black box ..... 57, 187, 191, 193, 215, 231, 296  
Bootstrapping ..... 26, 295  
Boruta ..... 24  
Building block ..... 141, 146, 148

## C

- Calliper randomization ..... 186, 187, 191–193  
Cancer ..... 2, 4, 37, 60, 174, 187, 188, 195–205, 209–216, 219–225, 245, 268, 350  
Causal relationship ..... 168  
Cellular differentiation ..... 211  
Chemical space ..... 139–141, 147, 152, 156, 157, 160  
Chromatin ..... 231, 232, 235–237, 239–244, 246  
Chromosome ..... 79, 146, 186  
Class  
    activation map ..... 57  
    membership ..... 36, 39, 40, 58, 63  
Classifiers ..... 24–28, 37, 40–43, 46–52, 66, 169, 191, 192, 197, 202, 205, 206, 213, 214, 296, 297, 299, 300  
ClinVar ..... 4, 7, 11  
Cloud ..... 103, 112, 211, 318, 322, 324–326, 328, 329, 334  
Combinatorial optimization ..... 140, 141, 161  
Complex ..... v, 5, 6, 8, 10, 18, 21, 22, 40, 74, 115, 126, 130, 153, 156, 168, 171, 172, 175, 177–179, 185, 186, 191, 193, 195, 215, 216, 229, 230, 232, 246, 250, 253, 268, 270, 273, 277, 290, 292, 295, 307, 322, 345, 350  
Compression ..... 292, 322, 323, 329, 330  
Confusion matrix ..... 27, 43, 75, 76  
Correlation ..... v, 1–29, 35, 62, 73, 75, 84, 122, 133, 188, 197, 199, 200, 243, 257, 313  
Cross-entropy ..... 59, 75, 212, 218, 219, 252, 283  
Cross-validation ..... 26–28, 47, 196, 202–206, 254, 257–260  
Curation ..... 4, 25, 28, 300

## D

- Data  
    biomedical ..... 33–35, 67, 211, 217, 219, 220, 225, 317–334  
    collection ..... 4, 33, 317–334  
    encryption ..... 328, 329

- Data (*cont.*)
- format ..... 322, 330, 331, 334
  - integration ..... 167–180, 209–225
  - mining ..... 35, 49, 215–219, 293, 340
  - representation ..... 296, 298
  - security ..... 318, 322, 325, 328
  - dbSNP ..... 4
  - Decision trees ..... 26, 52
  - Deep learning ..... v, 33–67, 85–87, 140, 143, 149–159, 161, 167, 170–178, 211, 213–217, 220–225, 250, 269–271, 279–285
  - Degenerative disease ..... 337–353
  - De novo ..... v, 139–161, 169, 174
  - Diagnostic system ..... 195–205
  - Differential evolution ..... 115–125
  - Dimensionality ..... 79, 85, 86, 153, 171, 196, 198, 200, 201, 204
  - Dimension reduction ..... 171, 214, 216
  - Disease ..... 2, 4, 24, 25, 79, 82, 130, 170, 173, 175, 178, 195, 196, 213, 216, 219, 223, 225, 229–247, 249, 250, 261, 264, 268, 289, 293, 299, 302, 337–353
  - Drug
    - design ..... v, 139–161
    - discovery ..... 139–141, 144, 159, 161, 167–180  - DUET ..... 3, 5, 6, 8, 15–17, 21, 23, 25
  - Durbin-Watson test ..... 35
  - Dynamic time warping (DTW) ..... 40, 46–49, 67, 77
- E**
- Electrocardiographic (ECG) ..... 35, 40, 57, 344–346, 349, 350, 353
  - Embedding ..... 78, 79, 170, 173, 175–179, 255, 290, 297–300, 338
  - Encryption ..... 322, 326–330
  - Engineering ..... 65, 115, 119, 175, 297, 332, 333
  - Enhancer ..... 231–241, 243–247
  - Enhancer-gene link ..... 230, 234, 237, 238, 243, 244, 246
  - Evolutionary strategies (ES) ..... 145–147
  - Exome ..... 4
- F**
- False
    - discovery rate ..... 56, 204
    - negative ..... 27, 28, 42, 202, 218, 231, 240, 265, 301
    - positive ..... 27, 42, 54, 55, 82, 96, 103, 190, 191, 218, 236, 239, 241, 301  - Feature extraction ..... 39, 40, 169, 211, 215, 217, 254, 271–280, 338
- Features ..... 12, 36, 74, 96, 152, 168, 187, 196, 210, 230, 250, 268, 290, 308, 326
- Feedforward perceptron ..... 57, 75
- Fermentation ..... 117, 119, 122, 130, 131
- Fine-mapping ..... 234, 236, 239, 240
- Fingerprints ..... 75, 79, 142, 173
- Fitness ..... 2, 25, 118, 120, 122, 123, 129, 133, 134, 142–148
- Fourier transform ..... 41
- G**
- Gene ..... v, 4, 7, 8, 11, 12, 24, 25, 74, 95–113, 120, 168, 170, 173, 176, 186, 195–205, 211, 212, 215, 223, 224, 230–232, 234–237, 239, 243–246, 293, 299, 319
  - Gene expression ..... 95–113, 173, 176, 195–205, 211, 212, 215, 223, 224, 232, 235–237, 243, 244
  - Genetic variants ..... 229–238, 241, 242, 244–246
  - Genome-wide association studies (GWAS) ..... 230–232, 234, 236, 240, 242, 247
  - Genomics ..... 4, 8, 49, 96, 99, 168, 170, 179, 180, 185, 186, 209–225, 229–237, 239–244, 331
  - Genotype ..... v, 1–29, 120, 233, 240, 244
  - Genotype-phenotype correlation ..... v, 1–29
  - Gibbs Free Energy ..... 10
  - Graph-based signature ..... 3, 10, 18, 21
  - Ground truth ..... 75, 76, 220
- H**
- Heterogeneity ..... 209–225, 293
  - Hidden
    - layer ..... 58–59, 116, 120, 129, 140, 168, 170, 171, 188, 189, 212, 214, 251, 268, 286, 309, 311–313
    - Markov model ..... 239, 240  - High throughput ..... 185, 186, 212, 268, 317, 319, 324
  - Hot-spots ..... 4, 18, 267–286
  - Human gene mutation database (HGMD) ..... 4
  - Hybrid methods ..... 116, 120
- I**
- Image analysis ..... 77, 79–81, 87, 212–214
  - Imaging ..... 81, 209–225, 317, 324, 333
  - Interactome ..... 267
  - Irregular time series ..... 37
- K**
- Kernels ..... 6, 62, 212, 251, 253, 256, 296
  - k-fold ..... 26, 27, 202, 206

**L**

- Ligand-receptor interaction ..... 210  
 Linear discriminant analysis (LDA) ..... 202, 205, 206  
 Lipinski's rule of ..... 5, 141  
 Logistic regression ..... 58, 230, 241  
 Long short term memory (LSTM) ..... 57, 59,  
     60, 82, 83, 152, 290, 297–299, 301

**Loss**

- dice ..... 218, 219  
 focal ..... 218  
 function ..... 59, 152, 154, 156,  
     158, 159, 214, 217–219, 268, 269

**M**

- Machine learning (ML) ..... v, 2–4, 24–28,  
     33–67, 74, 75, 80, 85, 96, 140, 167–180, 186,  
     209–225, 233–238, 246, 247, 249, 250, 254,  
     268–270, 286, 290, 296, 299, 300, 307,  
     317–334, 349  
 Magnetic resonance imaging (MRI) ..... 81  
 Manhattan distance ..... 73  
 Matthews correlation coefficient (MCC) ..... 5, 28,  
     75, 257, 258, 265  
 Mean square error (MSE) ..... 75, 120, 129,  
     133, 134, 214  
 Metadata ..... 290, 330, 332–334  
 Metaheuristics ..... 117, 144  
 Microarrays ..... 195, 196  
 Microbiome ..... 170, 249–265  
 Mining ..... 33, 35, 46, 49, 52, 53,  
     56, 66, 67, 77, 83, 87, 215–219, 290–293,  
     296, 297, 299, 302, 340  
 Molecular biology ..... 167–180, 204  
 Molecular representation ..... 142, 143, 152  
 mRNAs ..... 95, 96, 197, 203, 206, 215  
 Multilayer perceptron ..... 58, 338  
 Multiobjective optimisation ..... 140, 144  
 Mutation  
     analysis ..... 1–29  
     missense ..... 2, 4–6  
     non-pathogenic ..... 24, 25  
     nonsense ..... 2  
     pathogenic ..... 24, 25  
 Mutation cutoff scanning matrix (mCSM) ..... 2–8, 10,  
     13, 15–23, 25

**N**

- Naïve Bayes ..... 26, 173  
 Natural language processing (NLP) ..... 83, 87,  
     141, 142, 149, 250, 290–292, 296

**Network**

- biological ..... 170, 175, 229–247  
 convolutional graph ..... 47, 61  
 dynamic ..... 178  
 generative adversarial ..... 141, 149, 156, 171  
 neural  
     artificial ..... v, 59, 75, 86, 115–117,  
     140, 168, 185–193, 211, 268, 290, 296, 297,  
     338, 339, 341  
     autoencoder ..... 171  
     convolutional ..... 57, 61–63, 66,  
     78, 81–84, 87, 169, 172, 174, 211–213, 215,  
     223, 249–265  
     feedforward ..... 58, 59, 75, 76, 171  
     fuzzy ..... 340–342  
     recurrent ..... 57–59, 66, 78, 83,  
     124, 149–153, 169, 290, 296, 297, 309  
     recursive ..... v, 307–314  
     siamese ..... 73–87  
     spiking ..... 340, 342–344  
 regulatory ..... 96, 168, 230,  
     232–238, 241, 243–245  
 residual ..... 57, 61, 80  
 topology ..... 319–320, 324  
 Nucleic acid ..... 3, 6, 8, 21, 25, 126  
 Null hypothesis ..... 53, 54

**O**

- One-shot learning ..... 81, 87  
 Ontology ..... 74, 290, 291, 293,  
     296, 298–300, 302

**P**

- Parametric optimization ..... 116  
 Pareto Front ..... 143  
 Parse tree ..... 292, 296, 298  
 Partial least square regression (PLSR) ..... 196  
 Particle swarm optimisation (PSO) ..... 144, 145,  
     147–148, 150, 151  
 Pertraction ..... 126, 129–136  
 Phenomenological modelling ..... 115  
 Phenotype ..... 2, 4, 23–28,  
     37, 40–44, 53, 120, 146, 186, 196, 229–231,  
     233, 240, 243, 244, 249–265, 289, 292, 299,  
     300, 302  
 POS tagging ..... 292, 293, 295–298  
 Precision-recall (PR) curve ..... 28, 45, 75  
 Predictive classifier ..... 24–28  
 Predictor ..... 7, 25, 26, 169, 177, 210, 212  
 Principal Component Analysis (PCA) ..... 214–216

- Priors ..... 28, 39, 116, 142, 149, 159, 177, 192, 213, 216, 231, 232, 236, 239–241, 247, 253
- Promoter ..... 95–97, 99, 100, 109–111, 113, 232, 240, 243, 247
- Prosthetics ..... 337–352
- Protein
- coding ..... 98, 231
  - decoy model ..... 308
  - dynamics ..... 5, 15, 25
  - folding ..... 2, 23
  - interaction ..... 3, 6, 268, 300
  - structure ..... 3, 4, 7–9, 15, 307, 308
  - wild-type ..... 3, 7, 8, 21
- Protein Data Bank (PDB) ..... 4, 7, 8, 10, 12, 15, 28, 160, 270, 273, 275, 308, 309, 310
- PubChem ..... 159, 160
- PubMed ..... 77, 290
- Python ..... 26, 67, 85, 86, 98, 99, 254, 268–272, 276, 280, 331
- ## R
- Random forest ..... v, 24, 52, 96, 140, 173, 230, 243, 244
- Reactive extraction ..... 126–130, 133, 136
- Receiver operating characteristic (ROC)
- curve ..... 28, 75, 190–192, 285
- Rectified linear units (ReLU) ..... 57, 64, 212, 217, 252, 281
- Regularizer ..... 216
- Regulatory element ..... 231–232, 235–236, 239–242, 246
- Reinforcement learning (RL) ..... 149–151, 154–157
- Relation extraction ..... 289–302
- Residue ..... 3, 4, 8, 12, 15, 18, 21, 23, 25, 28, 29, 268, 271, 272, 274–276, 278, 280, 308–311
- ## S
- Seasonality ..... 34
- Secular trend ..... 34
- Self-organizing maps (SOMs) ..... 160, 338, 340
- Semantic similarity ..... 74, 79, 87, 302
- Separation ..... 115–136, 243
- Sepsis ..... 39, 43–45, 47–49, 56, 65–66
- Sepsis onset ..... 34, 39, 43–45, 47, 49, 65
- Sequencing ..... 4, 58, 209–211, 223, 232, 235, 253, 254, 319, 321, 323, 324
- Serial dependence ..... 36
- Shapelet ..... 33–66
- Sigmoid ..... 58, 60, 122, 212, 218
- Simplified molecular-input line-entry specification (SMILES) ..... 141–143, 145, 146, 150–157, 159, 160
- Simulated annealing (SA) ..... 144, 145, 148–149
- Single nucleotide polymorphisms (SNPs) ..... 230, 231, 233–247
- Small noncoding RNAs (sRNAs) ..... 95–97, 99–109, 112
- Softmax ..... 58–59, 152, 212, 252
- Software ..... v, 4, 66, 74, 77, 82, 85–87, 98, 99, 122, 146, 169, 180, 219, 225, 238–244, 246, 254, 307, 318, 320, 321, 325, 329, 331, 333, 334
- Spatial
- omics ..... 221–225
  - transcriptomics ..... 210, 211, 219, 223
- Spearman’s rank correlation coefficient ..... 73
- Spectral clustering (SC) ..... 200–203
- Stability ..... 2, 5, 6, 10–12, 14, 15, 17, 21, 23, 25, 143, 156, 268, 319
- Stemming ..... 292
- Subsequence mining ..... 53, 65, 66
- Support vector machine (SVMs) ..... v, 26, 96, 173, 198, 202–206, 242, 249, 268, 296, 300
- ## T
- Tanh ..... 58, 212
- Taxonomic
- abundance ..... 249
  - features ..... 254, 255, 257, 258
- TensorFlow ..... 66, 85, 254, 270, 280, 283–285
- Testing set ..... 26, 129, 259
- Text mining ..... 77, 83, 87, 290–293, 296, 297, 299, 300, 302
- Time series ..... v, 33–67
- Time-stamp ..... 37, 39–44
- Tissue morphology ..... 219, 223
- Tokenization ..... 291–292
- Topological overlap ..... 199, 200
- Topology ..... 116, 117, 119, 120, 122, 129, 133–135, 147, 199, 319–320, 324
- Training set ..... 27, 152, 156–158, 256, 257, 259, 284, 286, 308, 312
- Transcript ..... 4, 7, 11, 98–100, 185, 188
- Transcriptome ..... 95, 187, 188, 221, 250
- Transfer learning ..... 159, 211, 213, 214, 217, 219–221
- Tuberculosis ..... 2, 173
- Tumor ..... v, 188, 290

**U**

UK Biobank ..... 4

**V**

Validation ..... 26–28, 47, 48, 151,  
153, 154, 180, 196, 207, 215, 256

Virtual screening ..... 140, 142

**W**

Wavelet transform ..... 39, 41

Wearable electronics ..... 352  
Weka ..... 26, 27  
White box ..... 218  
Wrapper ..... 196, 198, 202–205

**Z**

ZINC ..... 150, 151, 159, 160