FACULTY OF CHEMICAL AND METALLURGICAL ENGINEERING

DEPARTMENT OF BIOENGINEERING

CLASSIFICATION OF TUMOR GENE EXPRESSION USING ARTIFICAL NEURAL NETWORKS

ISTANBUL, 2018

T.C.

YILDIZ TECHNICAL UNIVERSITY

FACULTY OF CHEMICAL AND METALLURGICAL ENGINEERING

CLASSIFICATION OF TUMOR GENE EXPRESSION USING ARTIFICAL NEURAL NETWORKS

Taylan KABBANI

1405A902

GRADUATION THESIS
DEPARTMENT OF BIOENGINEERING

ADVISER
Assist. Prof. Dr. Alper YILMAZ

İSTANBUL, 2018

T.C.

YILDIZ TECHNICAL UNIVERSITY

FACULTY OF CHEMICAL AND METALLURGICAL ENGINEERING

CLASSIFICATION OF TUMOR GENE EXPRESSION USING ARTIFICAL NEURAL NETWORKS

This **GRADUATION THESIS** submitted by **Taylan KABBANI** is approved by the committee on 05/06/2018 in Yildiz Technical University, Faculty of Chemical and Metallurgical Engineering, Department of Bioengineering.

Thesis Adviser

Assist. Prof. Dr. Alper YILMAZ

Yıldız Technical University

Approved By the Examining Committee

Yıldız Technical University _____

Prof. Dr. Dilek BALIK

Yıldız Technical University _____

Doç. Dr. M. Murat ÖZMEN

Yıldız Technical University _____

Dr. Öğr. Üyesi Alper YILMAZ

ACKNOWLEDGEMENT

I take this opportunity to express gratitude to all the Department faculty members for their help and support. I place on record, my sincere thanks to Assist. Prof. Dr. Alper YILMAZ the supervisor of my thesis, without his assistance and dedicated involvement in every step throughout the process, this paper would have never been accomplished.

Getting through my dissertation required more than academic support, and I have many, many people to thank over the past four years.

Most importantly, none of this could have happened without my family, this dissertation stands as a testament to your unconditional love and encouragement

May,2018

Taylan KABBANI

Contents

LIST OF SYMBOLS

λ_0 Eigenvalue criterion

λ Eigenvalue

K Variable

n Observations

p Component

LIST OF ABBREVIATIONS

AE Auto Encoder

ANN Artificial Neural Network

API Application Programming Interface

BRCA BReast CAncer (susceptibility gene)

CNN Convolutional Neural Network

CPU Central Processing Unite

DBN Deep Belief Network

EEG Electroencephalography

GPU Graphics Processing Unite

MLP Multilayer Perceptron

MNIST Mixed National Institute of Standards and Technology

NLP Natural Language Processing

PCA Principle Component Analysis

RBM Restricted Boltzmann Machine

ReLU Rectified Linear Unit

RGB Red-Green-Blue

RNN Recurrent Neural Network

SDA Stacked Denoising Autoencoder

SVD Singular Value Decomposition

TCGA The Cancer Genomic Atlas

TPU Tensor Processing Unit

LIST OF FIGURES

page

ÖZET

YAPAY SİNİR AĞLARI KULLANARAK TÜMÖR GENİ EKSPRESYONUNUN SINIFLANDIRILMASI

Taylan KABBANI

Biyomühendislik Bölümü

Lisans Tezi

Assist. Prof. Dr. Alper YILMAZ

Tümör insan sağlığına ciddi zararlar verdiği için, tümör tedavisinde etkili tanı yöntemlerine acil ihtiyaç duyulmaktadır. Tümörün erken teşhisi, hastaların daha iyi tedavisi için oldukça önemlidir. Gen ifadesi verisi kullanılarak yapılan kanser saptaması, bu verilerin yüksek boyutsallığı ve karmaşıklığı nedeniyle saptamayı zorlaştırmaktadır. Yıllardır süren araştırmalardan sonra, kanserin klinik tanısında ve tümöre özgü belirteçlerin tanımlanmasında hala belirsizlikler vardır. Burada, kanser tespitine ve meme kanseri tanısı için kritik genlerin tanımlanmasına yönelik derin bir öğrenme yaklaşımı sunuyoruz. İlk olarak, BRCA'daki kritik genleri seçmek için PCA'yı bir boyut azaltma yöntemi olarak kullandık. Daha sonra, çıkarılan genler üzerinde derin öğrenme nöral ağı (MLP) geliştirdik. Son olarak, modelimizi, MLP'nin% 98'lik bir doğrulukla gen ekspresyonu verilerinden kanser numunelerini sınıflandırabildiğini göstermek için eğitim verilerimize dahil edilmeyen diğer örnekler üzerinde test ediyoruz.

YILDIZ TECHNICAL UNIVERSITY

FACULTY OF CHEMICAL AND METALLURGICAL ENGINEERING

ABSTRACT

CLASSIFICATION OF TUMOR GENE EXPRESSION USING ARTIFICAL NEURAL NETWORKS

Taylan KABBANI

Department of Bioengineering

B.Eng. Graduation Thesis

Assist. Prof. Dr. Alper YILMAZ

Since tumor is seriously harmful to human health, effective diagnosis measures are in urgent need for tumor therapy. Early detection of tumor is particularly important for better treatment of patients. Cancer detection from gene expression data continues to pose a challenge due to the high dimensionality and complexity of these data. After decades of research there is still uncertainty in the clinical diagnosis of cancer and the identification of tumor-specific markers. Here we present a deep learning approach to cancer detection, and to the identification of genes critical for the diagnosis of breast cancer. First, we used PCA as a dimensionality reduction method to choose the critical genes in BRCA. Next, we trained a deep learning neural network (MLP) on the extracted genes. Lastly, we test our model on other samples that were not included in our training data to illustrating that MLP are capable of classifying cancer samples from gene expression data with an accuracy of %98.

YILDIZ TECHNICAL UNIVERSITY

FACULTY OF CHEMICAL AND METALLURGICAL ENGINEERING

CHAPTER 1

INTRODUCTION

1.1 Literature Review

Tumors, which endanger human health, are part of the major malignant diseases in the world. Early detection of the tumor is under a very important meaning for the better treatment of patients [25]. Since 2005 after the development of high-throughput sequencing the amount of publicly available genomic data has exponentially increased over the years due to the science genomic projects like The Cancer Genomic Atlas (TCGA) and other projects that each of which deposit massive genomic datasets.

This exponential growth in the amount of biological data available, made machine learning algorithms applicable in the field of biology, allowing us to go beyond our mere description of the data and provide knowledge in the form of testable models.

MLP has a similar structure to the usual neural networks but includes more stacked layers. It is trained in a purely supervised manner that uses only labeled data. Since the training method is a process of optimization in high-dimensional parameter space, MLP is typically used when a large number of labeled data are available.

1.2 Objective of the Thesis

Artificial intelligence technologies are the first choice in large data and data mining today and the fact that the cancer genomic data cannot be interpreted via human power made artificial intelligence integration a necessity in the field of biology. In this study, we aimed to present the use of artificial intelligence technology as an aid to the clinician diagnosis stage for cancer, taking into consideration the effect of early diagnosis which is vital in cancer treatment.

1.3 Hypothesis

Developing a method of classifying cancers to specific diagnostic categories based on their gene expression using artificial neural networks can save a lot of time and effort which is crucial for the cancer patient and play an important role for better treatment. Furthermore, the pain of doing biopsy can be avoided, where just gene expressions are needed in this method of diagnosing.

CHAPTER 2

MACHINE LEARNING

Machine learning arise from the questions: Could a computer go beyond "whatever we know how to order it to perform" and learn how to perform a specified task? Could a computer suprise us? Rather than programmers crafting data processing rules by hand, could a computer automatically learn these rules by looking at data?

These questions open the door to a new programming paradigm. In classical programming humans have to input rules (a program) and data to enable the machine to run according to the preset rules (see Figure.1). With machine learning, humans input data as well as the answers expected from the data, and out come the rules. These rules can be applied to new data to produce original answers.

So machine learning in its simplest definition: is giving the machine system the ability to "learn", where learning in this context of machine learning describes an automatic search process for better presentation of the input data without being explicity programmed.

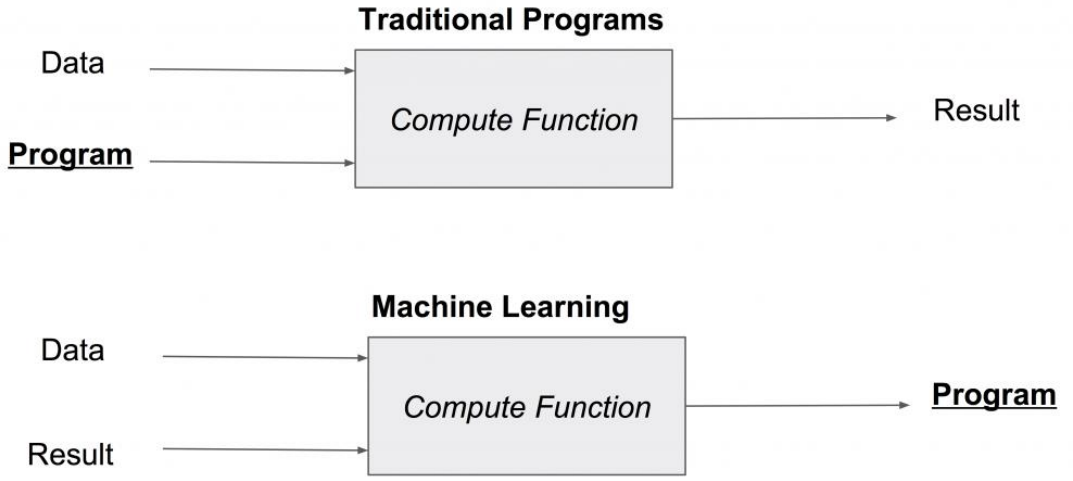


Figure.1 Machine learning: a new programming paradigm

2.1 MACHINE LEARNING ALGORITHMS

The performance and computational analysis of machine learning algorithms is a branch of statistics known as computational learning theory.

2.1.1 SUPERVISED LEARNING ALGORITHMS

In supervised learning the goal is often to get the computer to learn a classification system that we have created (E.g.Digit recognition). More generally, classification learning is appropriate for any problem where deducing a classification is useful and the classification is easy to determine (see Figure.2). In some cases, it might not even be necessary to give pre-determined classifications to every instance of a problem if the agent can work out the classifications for itself. This would be an example of unsupervised learning in a classification context [1].

In the area of supervised learning which deals much with classification. These are the algorithms types:

- Linear Classifiers
    1) Logistic Regression: is a classification algorithm rather than a regression algorithm, which is sometimes considered to be the "Hello world" of modern machine learning.
    2) Support vector machine (SVM): aim to solve classification problems by finding good decision boundaries between two sets of points belonging to two different categories. Applying an SVM to perceptual problems requires first extracting useful representation manually (feature engineering), which is difficult and brittle.
    3) Other algorithms such as: Naive Bayes Classifier and Perceptron.
- Quadratic Classifiers
- K-Means Clustering
- Boosting
- Decision Tree
- Random Forest
- Neural networks
- Bayesian Networks

OVER FITTING: inputs (often called the "training set"), are the examples from which the agent tries to learn but sometimes the model memorizing the training set rather than learning a more general classification technique negatively impacts the performance of the model on new data, which is a common problem in machine learning called over fitting.

2.1.2 UNSUPERVISED LEARNING ALGORITHMS

In unsupervised learning, all the observations are assumed to be caused by latent variables, that is, the observations are assumed to be at the end of the causal chain (see Figure.2). In practice, models for supervised learning often leave the probability for inputs undefined, enabling the computer to learn how to do something that we don't tell it how to do!

There are actually two approaches to unsupervised learning. The first approach is to teach the agent not by giving explicit categorizations but by using some sort of reward system to indicate success this type of training fits into the decision problem framework *this kind of learning can be powerful because it assumes no pre-discovered classification*.

A second type of unsupervised learning is called clustering. In this type of learning, the goal is not to maximize a utility function but simply to find similarities in the training data. *This is a data-driven approach that can work well when there is sufficient data* [1]

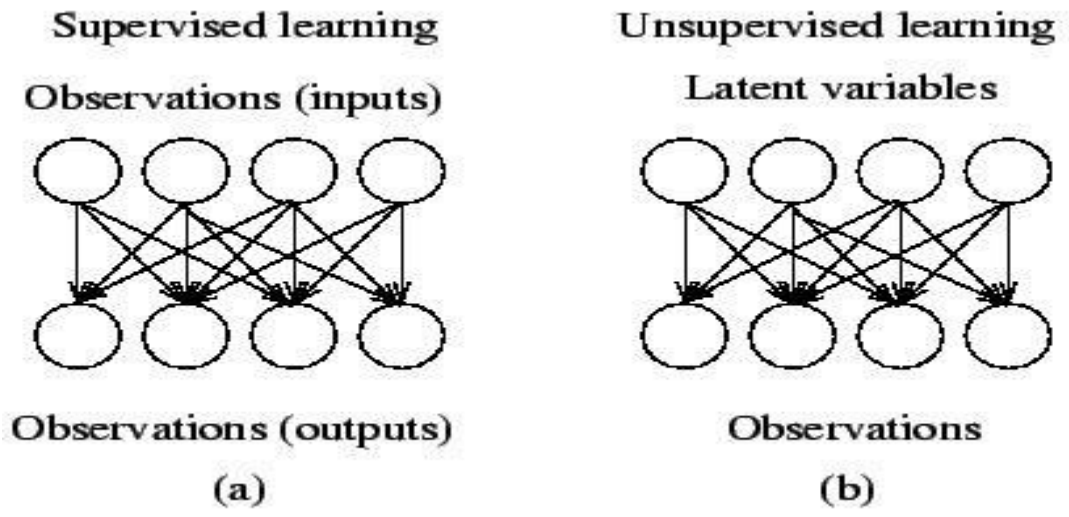


Figure.2 The causal structure of (a) supervised and (b) unsupervised learning [2]

- Unsupervised learning algorithm types:
    1) K-means
    2) Principal Component Analysis (PCA)
    3) Singular Value Decomposition (SVD)
    4) Hidden Makrov Model
    5) FP-Growth

Unfortunately, even unsupervised learning suffers from the problem of overfitting the training data. There's no silver bullet to avoid this problem.

2.1.3 OTHER MACHINE LEARNING ALGORITHMS

- SEMI-SUPERVISED LEARNING: which combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- REINFORCEMENT LEARNING: where the algorithm learns a policy of how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback that guides the learning algorithm.
- TRANSDUCTION: like supervised learning but does not explicitly construct a function instead tries to predict new outputs based on training inputs, training outputs and new inputs.
- LEARNING TO LEARN: where the algorithm learns its own inductive bias based on previous experience [3]

2.2 MACHINE LEARNING IN MOLECULAR BIOLOGY

The exponential growth in the amount of biological data available made machine learning algorithms applicable in the field of biology allowing us to go beyond our mere description of the data and provide knowledge in the form of testable models.

Life science applications of unsupervised and/or supervised machine learning techniques abound in the literature. For instance, gene expression data was successfully

used to classify patients in different clinical groups and to identify new disease group [4] while genetic code allowed prediction of the protein secondary structure [5]

Another interesting application of computational methods in biology is the management of complex experimental data. Microarray essays are the best known (but not the only) domain where this kind of data is collected. Complex experimental data raise two different problems. First, data need to be pre-processed (i.e. modified to be suitably used by machine learning algorithms.). Second the analysis of the data which depends on what we are looking for.

In the case of microarray data; the most typical applications are expression pattern identification, classification and genetic network induction [6]

Other biological domains where machine learning techniques are applied: systems biology, evolution and text mining in addition to all these applications; computational techniques are used to solve other problems, such as efficient primer design for PCR, biological image analysis and backtranslation of proteins (which is, given the degeneration of the genetic code, a complex combinatorial problem).

CHAPTER 3

DEEP LEARNING

Deep learning is a specific subfield of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations (see Figure.3). Modern deep learning often involves tens or even hundreds of successive layers of representations and they are all learned automatically from exposure to training data [7].

Note that the *deep* in *deep learning* isn't a reference to any kind of deeper understanding achieved by the approach rather it stands for this idea of successive layers of representations.

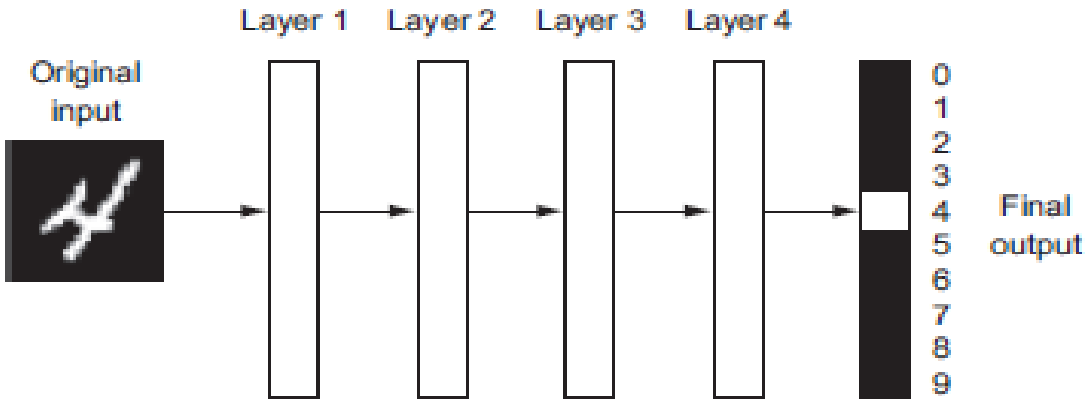


Figure.3 A deep neural network for digit classification [7]

In deep learning, these layered representations are (almost always) learned via models called neural network, structured in literal layers stacked on top of each other.

Although deep learning is fairly old subfield of machine learning, it only rose to prominence in the early 2010s, mainly driven by three forces :

- Hardware: throughout the 2000s, companies like NVIDIA and AMD have been investing billions of dollars in developing fast, massively parallel chips (graphical processing units [GPUs]). GPUs have been around in gaming applications since the early 1970's. The late 80's saw GPUs being added into consumer computers, and by 2018 they're absolutely standard. What makes a GPU unique is how it handles commands , it's the exact opposite of a CPU.
- Massive labeled datasets: the raw material that powers our intelligent machines, where over the past 20 years, following Moore's law (prediction made by American engineer Gordon Moore in 1965 that the number of transistors per silicon chip doubles every year) the game changer has been the rise of the internet, making it feasible to collect and distribute very large datasets for machine learning.
- Algorithms: The key issue was that of gradient propagation through deep stacks of layers,where the feed back signal used to train neural networks would fade away as the number of layers increased.In 2014, 2015 and 2016 advanced ways were discovered to help gradient propagation, such as batch normalization, residual connections,and depth wise separable convolutions. Today we can train from the scratch models that are thousands of layers deep [7].

3.1 WHAT MAKES DEEP LEARNING DIFFERENT

At the highest conceptual level, deep learning is similar to supervised machine learning, but deep learning offers better performance on many problems because of :

- *Simplicity:* One of the biggest challenges with traditional machine learning techniques(shallow learning) is *feature engineering*, in which the data scientist hypothesizes what data the machine learning algorithm will find useful by manually engineer good layers of representations for their data. Deep learning on the other hand, completely automates this step, so you can learn all features in one pass rather than having to engineer them yourself, this makes deep learning an extremely powerful tool for modern machine learning.
- *Scalability:* Deep learning is highly capable to parallelization on GPUs or TPUs, so it can take full advantage of moore's law. In addition, deep learning models are trained by iterating over small batches of data, allowing them to be trained on

datasets of arbitrary size. To meet the need of large amount of parallel computational power required, deep-learning researchers adopted graphics processing units (GPUs) because they have thousands of cores and can perform the operations necessary to train deep-learning networks.

- *Versatility and reusability:* Deep learning models can be trained on additional data without restarting from scratch,furthermore, trained deep-learning models are repurposable and thus reusable: for instance, it's possible to take a deep-learning model trained for images classification and drop it into a video-processing pipeline [7].
- *Purpose-built open source libraries are available:* Deep learning has its own set of libraries that are evolving quickly. Many types of deep-learning algorithms have been developed. The most popular are multilayered convolutional networks with back propagation, ideal for image and voice recognition, and recurrent neural networks, ideal for natural language processing (NLP). Popular open source deep-learning libraries include Caffe, Deeplearning4j, MXNet, TensorFlow, and Theano [8].

Although deep learning is a fairly old subfield of machine learning, it only rose to prominence in the early 2010s, in the few years since, it has achieved the following breakthroughs, all in historical difficult areas of machine learning:

- Near-human-level image classification.
- Near-human-level speech recognition.
- Near-human-level handwriting transcription.
- Improve text-to-speech conversion.
- Digital assistants such as Google now and Amazon Alexa.
- Near-human-level autonomous driving.
- Improve ad targeting, as used in Google, Baidu, and Bing.
- Improved search results on the web.
- Ability to answer natural-language questions.
- Superhuman Go playing.

3.2 DEEP LEARNING IN BIOINFORMATICS

In the era of 'big data' transformation of large quantities of data into valuable knowledge has become increasingly important in various domains [9], and bioinformatics is no exception. Significant amounts of biomedical data, including omics, image and signal data, have been accumulated, and the resulting potential for applications in biological and health-care research has caught the attention of both industry and academia [10].

The proper performance of conventional machine learning algorithms relies heavily on feature engineering which considered the bottleneck in applying machine learning algorithms in bioinformatics, however, deep learning has overcome this limitation facilitating the application of machine learning in bioinformatics studies (see Figure.4) splice junctions can be discovered from DNA sequences, finger joints can be recognized from X-ray images, lapses can be detected from electroencephalography (EEG) signals, and so on. Currently, CNNs are one of the most successful deep learning architectures owing to their outstanding capacity to analyze spatial information. Thanks to their developments in the field of object recognition, expectations of primary research achievements in bioinformatics will come from the biomedical imaging domain.

Despite the different data characteristics between normal and biomedical imaging, CNN will nonetheless offer straightforward applications compared to other domains [11]. relatively few studies have used CNNs to solve problems involving biological sequences, specifically gene expression regulation problems. for example , as an early approach, Denas et al [12] preprocessed ChIP-seq data into a two-dimensional matrix with the rows as transcription factor activity profiles for each gene and exploited a two-dimensional CNN similar to its use in image processing. Recently, more studies focused on directly

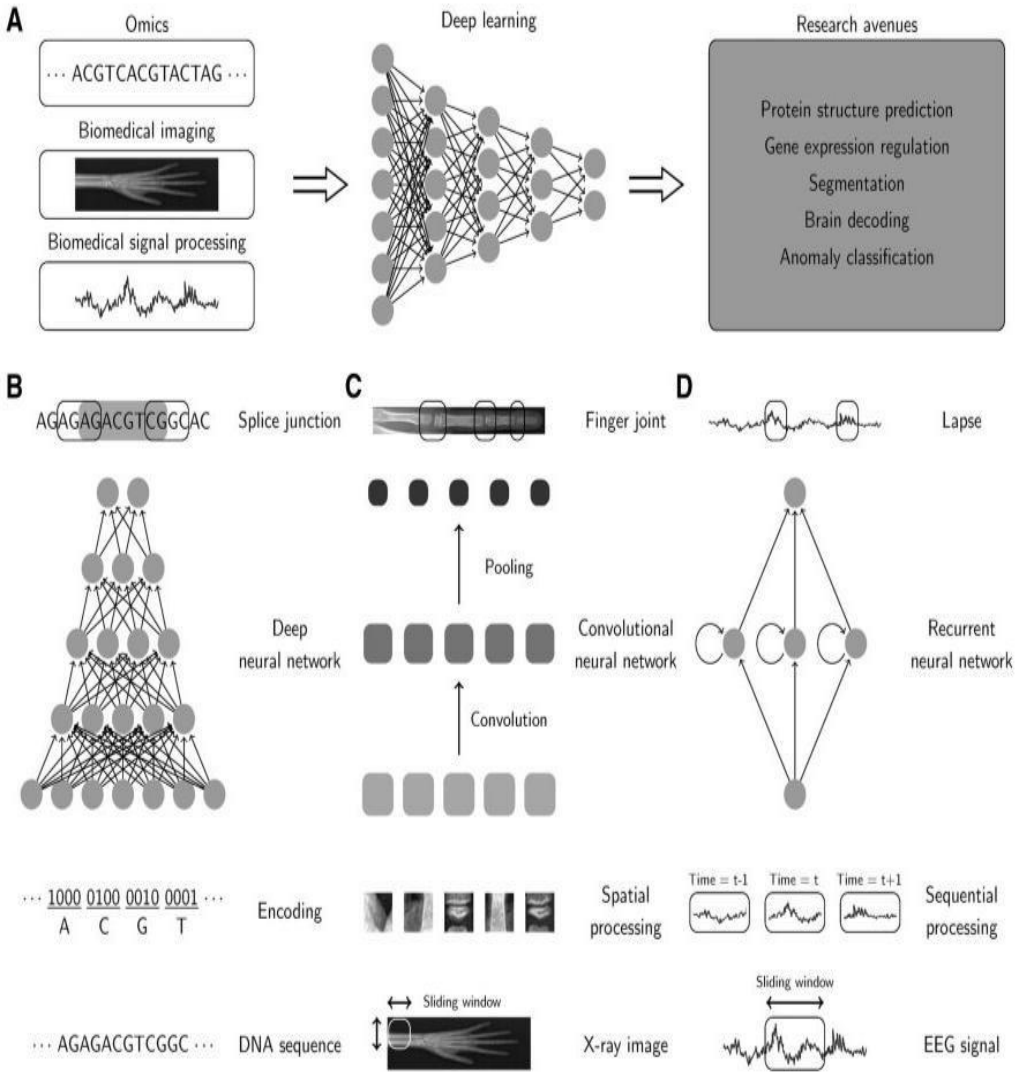


Figure.4 Application of deep learning in bioinformatics research. {A} Overview diagram with input data and research objectives. {B} A research example in the omics domain prediction of splice junctions in DNA sequence data with a deep neural network. {C} A research example in biomedical imaging. Finger joint detection from X-ray images with a convolutional neural network. {D} A research example in biomedical signal processing. Lapse detection from EEG signal with a recurrent neural network.

3.3 ARTIFICIAL NEURAL NETWORKS (ANN)

The successes of deep learning are built on a foundation of significant algorithmic details and generally can be understood in two parts: construction and training of deep learning architectures.

Deep learning architectures are basically “stacked neural networks”; that is, networks composed of several layers. The commonest type of artificial neural network consists of three layers of units:

- a layer of "input" units: the activity of the input units represents the raw information that is fed into the network.
- a layer of "hidden" units: the activity of each hidden units determined by the activities of the input units and the weights on the connections between the input and the hidden layers.
- a layer of "output" units: the behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units [13].

The layers are made of node (see Figure.5). A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input,these input-weight products are summed and the sum is passed through a node’s so-called activation function, to determine whether and to what extent that signal progresses further through the network to affect the ultimate outcome [14]

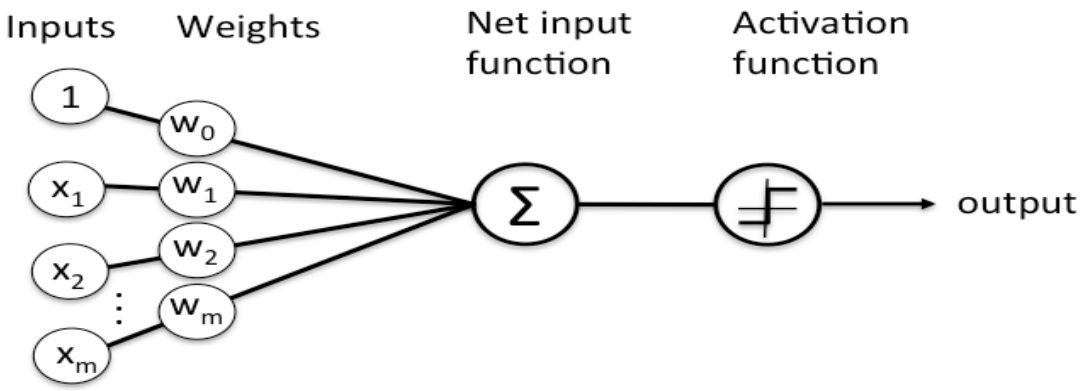


Figure.5 A Diagram of what one node might look like

The goal of *training* deep learning neural network is optimization of the weight parameters in each layer,which gradually combines simpler features into complex features so that the most suitable hierarchical representations can be learned from the data (is basically the learning that deep learning is all about). A single cycle of the optimization process is organized as follows:

all weights of the network in a way that slightly reduces the loss on this batch [7].

1. Draw a batch of training samples x and corresponding targets y.
2. Run the network on x (a step called the forward pass) to obtain predictions y_pred.
3. Compute the loss of the network on the batch, a measure of the mismatch between y_pred and y.
4. Update all weights of the network in a way that slightly reduces the loss on this batch [7].

3.3.1 KEY CONCEPTS OF ANN

- *Feed-forward network*

  Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of architecture is also referred to as bottom-up or top-down [13].

- *Feedback networks*

  Feedback networks can have signals travel in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent [13].

- *Activation Function*

  The activation function determines if the input value sufficiently large to trigger the firing of the neuron,where the input into the activation function is the sum of the weighted inputs plus biases from the preceding layer.

  The capability of ANNs to learn approximately any function, (given sufficient training data examples) are dependent on the appropriate selection of the Activation Function(s) present in the network. Activation functions enable the ANN to learn non-linear properties present in the data[15].

  Sigmoid activation functions had been very popular, ReLU is currently very popular (see Figure.6)

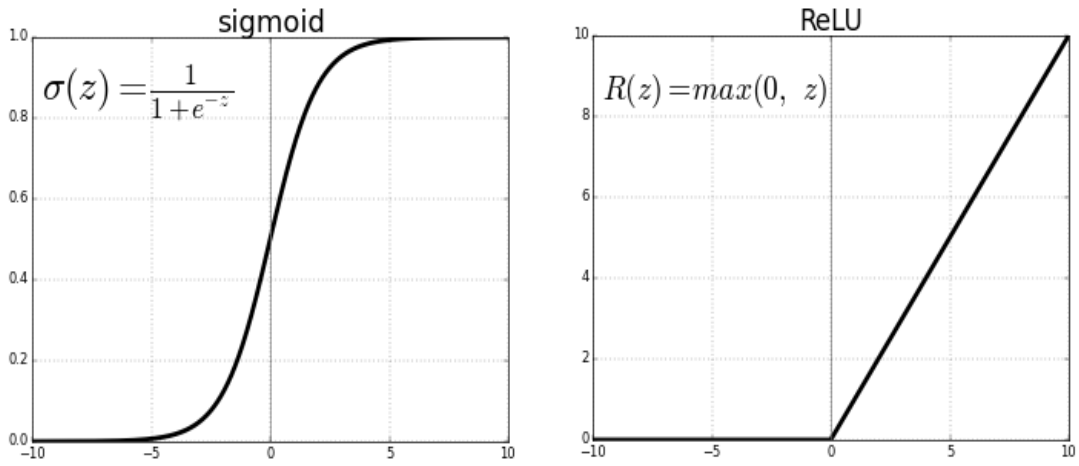


Figure.6 ReLU v/s Sigmoid

- *Loss Function* [16]

  In most learning networks, error is calculated as the difference between the actual output and the predicted output.The function that is used to compute this error is known as Loss Function. Different loss functions will give different errors for the

same prediction, and thus have a considerable effect on the performance of the model.

Different loss functions are used to deal with different type of tasks, for instance, cross-entropy for a two-class classification problem, categorical cross-entropy for a many-class classification problem, mean-squared error for a regression problem, connectionist temporal classification (CTC) for a sequence-learning problem, and so on.

- *Optimization Function* [16]

  The calculated error from the loss function is minimized using back **back propagation**, where the current error is typically propagated backwards to a previous layer, where it is used to modify the weights and biases in a way to minimize the error, the weights are modified using the *Optimization Function*. *Optimization functions* usually calculate the gradient i.e. the partial derivative of loss function with respect to weights, and the weights are modified in the opposite direction of the calculated gradient. This cycle is repeated until we reach the minimal of loss function. there are several optimization algorithms that can be used, below are the most used algorithms:

  1. ***Stochastic Gradient Decent:*** Gradient Descent calculates gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minimal. Stochastic Gradient Descent performs a parameter update for each training example unlike normal Gradient Descent which performs only one update. Thus it is much faster. Gradient Decent algorithms can further be improved by tuning important parameters like momentum, learning rate etc.

  2. ***Adagrad:*** is more preferable for a sparse data set as it makes big updates for infrequent parameters and small updates for frequent parameters. It uses a different learning rate for every parameter at a time step based on the past gradients which were computed for that parameter. Thus we do not need to manually tune the learning rate.

  3. ***Adam:*** stands for Adaptive Moment Estimation. It also calculates different learning rate. Adam works well in practice, is faster, and outperforms other techniques.

The activation function, loss function and optimization algorithm play a very important role in efficiently and effectively training a Model and produce accurate results. Different tasks require a different set of such functions to give the most optimum results.

3.3.2 ANN CATEGORIZATIONS

With new neural network architectures popping up every now and then, it's hard to keep track of them all. There are many types of artificial neural networks (ANN) each of which designed to satisfy the demand for specific data type and achieve a particular task. In this review we describe the most well studied and applied ANN.

3.3.2.1 MULTILAYER PERCEPTRON (MLP)

A multilayer perceptron (MLP) is a deep, artificial neural network, they are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer are capable of approximating any continuous function [17].

A perceptron: is an algorithm that classifies input by separating two categories with a straight line. Input is typically a feature vector x multiplied by weights w and added to a bias b: $y = w * x + b$.

MLP is trained in a purely supervised manner that uses only labeled data. Since the training method is a process of optimization in high-dimensional parameter space, MLP is typically used when a large number of labeled data are available.

MLPs are widely used in the field of biology , Chen et al. [18] applied MLP to both microarray and RNA-seq expression data to infer expression of up to 21000 target genes from only 1000 landmark genes. In terms of protein classification, Asgari et al. [20] adopted the skip-gram model, a widely known method in natural language processing, that can be considered a variant of MLP and showed that it could effectively learn a

distributed representation of biological sequences with general use for many omics applications, including protein family classification.

3.3.2.2 DEEP BELIEF NETWORK (DBN)

The building blocks of DBN *are restricted Boltzmann machines (RBMs)* (see Figure.7) in which each RBM layer communicates with both the previous and subsequent layers. The nodes of any single layer don't communicate with each other laterally.

This stack of RBMs might end with a a Softmax layer to create a classifier, or it may simply help cluster unlabeled data in an unsupervised learning scenario.

With the exception of the first and final layers, each layer in a deep-belief network has a double role: it serves as the hidden layer to the nodes that come before it, and as the input (or "visible") layer to the nodes that come after. It is a network built of single-layer networks.

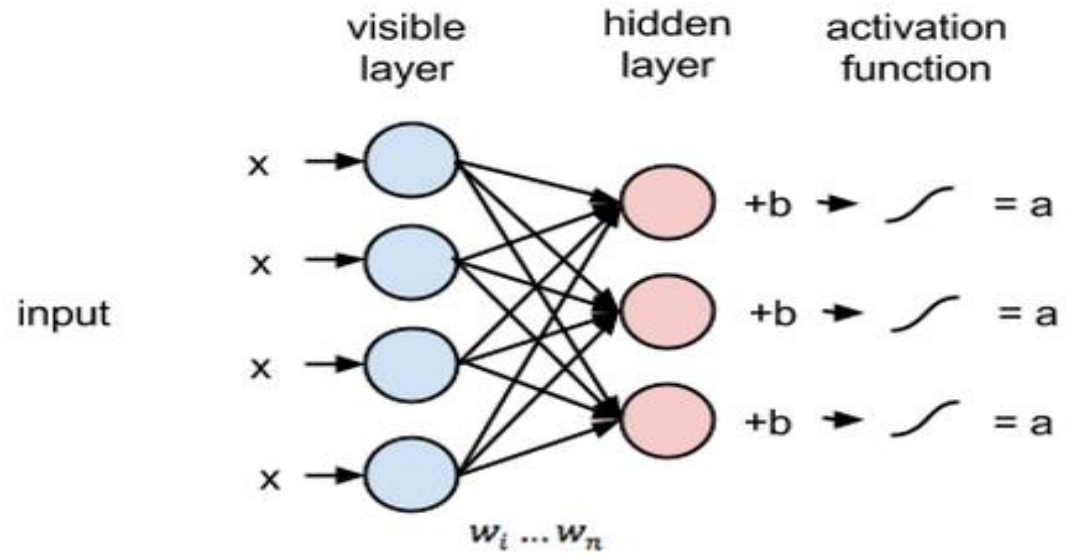


Figure.7 Restricted Boltzmann Machine

DBN have shown great capabilities in the area of gene expression regulation, for example, Lee et al. [21] utilized DBN in splice junction prediction, a major research avenue in understanding gene expression.

3.3.2.3 STACKED DENOISING AUTOENCODER (SDA)

There are times when it is extremely useful to figure out the underlying structure of a dataset. Having access to the most important data features gives you a lot of flexibility when you start applying labels. Autoencoders(AE) are an important family of neural networks that are well-suited for this task, RBM is an example of autoencoder with two layers.

Denoising autoencoders are an extension of the basic autoencoder and represent a stochastic version of it. Denoising autoencoders attempt to address identity-function risk by randomly corrupting input (i.e. introducing noise) that the autoencoder must then reconstruct, or denoise. Stacked Denoising Autoencoder is simply many denoising autoencoders strung together.

SDAs shuffle data around and learn about that data by attempting to reconstruct it. The act of shuffling is the noise, and the job of the network is to recognize the features within the noise that will allow it to classify the input. When a network is being trained, it generates a model, and measures the distance between that model and the benchmark through a loss function. Its attempts to minimize the loss function involve resembling the shuffled inputs and re-reconstructing the data, until it finds those inputs which bring its model closest to what it has been told is true [22].

SDA and DBN use AEs and RBMs as building blocks of the architectures, respectively. The main difference between these and MLP is that training is executed in two phases: unsupervised pre-training and supervised fine-tuning. First, in unsupervised pretraining the layers are stacked sequentially and trained in a layer-wise manner as an AE or RBM using unlabeled data. Afterwards, in supervised fine-tuning, an output classifier layer is stacked, and the whole neural network is optimized by retraining with labeled data. Since both SDA and DBN exploit unlabeled data and can help avoid overfitting, researchers are able to obtain fairly regularized results, even when labeled data are insufficient as is common in the real world [19].

3.3.2.4 RECURRENT NEURAL NETWORKS (RNNs)

RNN is an algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for Machine Learning problems that involve sequential data, such as sound, time series (sensor) data or written natural language, so If you want to predict the

next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations.

In RNN the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously (see Figure.8) this differs from a feedforward network which accept one input at a time and produce one output. The two images below illustrate the difference in the information flow between RNN and a Feed-Forward Neural Network.

Since omics data and biomedical signals are typically sequential and often considered languages of nature, the capabilities of RNNs for mapping a variable-length input sequence to another sequence or fixed-size prediction are promising for bioinformatics research [11].

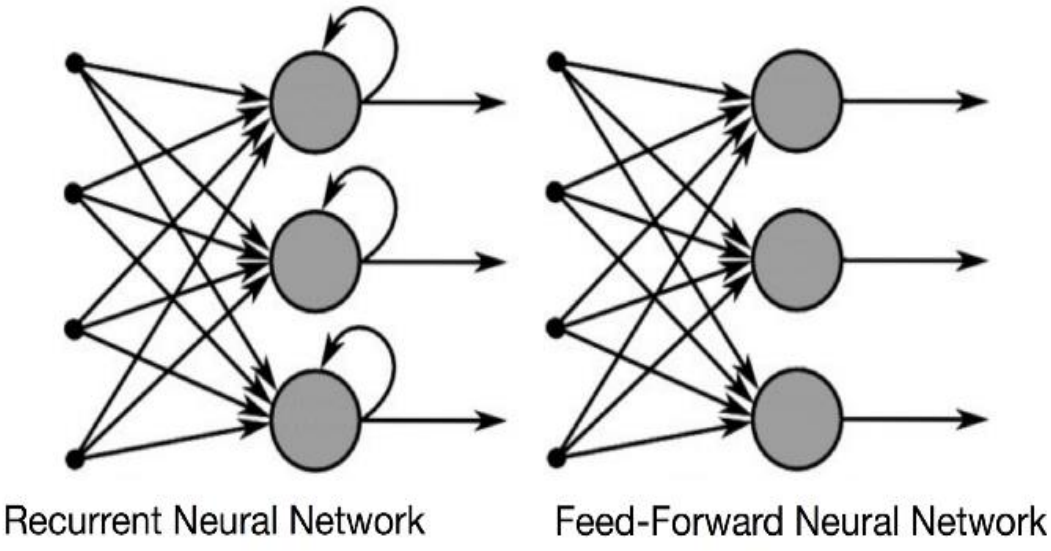


Figure.8 RNN

3.3.2.5 CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images, cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

Convolutional networks perceive images as volumes; i.e. three-dimensional objects, rather than flat canvases to be measured only by width and height. That's because digital color images have a red-blue-green (RGB) encoding, mixing those three colors to produce the color spectrum humans perceive. A convolutional network ingests such images as three separate strata of color stacked one on top of the other, so a convolutional network receives a normal color image as a rectangular box whose width and height are measured by the number of pixels along those dimensions, and whose depth is three layers deep, one for each letter in RGB. Those depth layers are referred to as channels [23].

The basic structure of CNNs consists of:

1) ***Convolution Layers:*** the core building block of a CNN, passing many filters over a single image, each one picking up a different signal, then take those filters (slices of the image's feature space) and map them one by one; creating a map of each place that feature occurs. By learning different portions of a feature space, convolutional layers allow for easily scalable and robust feature engineering. Furthermore, since identical patterns can appear regardless of the location in the data, filters are applied repeatedly across the entire dataset, which also improves training efficiency by reducing the number of parameters to learn.

2) ***Non-linear Layers:*** after a convolutional layer, input is passed through a nonlinear transform such as tanh or rectified linear unit, which will squash input values into a range between -1 and 1.

3) ***Pooling Layer:*** It is common to periodically insert a Pooling layer in-between successive convolution layers, Its function is to progressively reduce the spatial size of the representation by simply taking the largest value from one patch of an image, places

it in a new matrix next to the max values from other patches, and discards the rest of the information contained in the activation maps, that enables CNNs to handle somewhat different but semantically similar features and thus aggregate local features to identify more complex features.

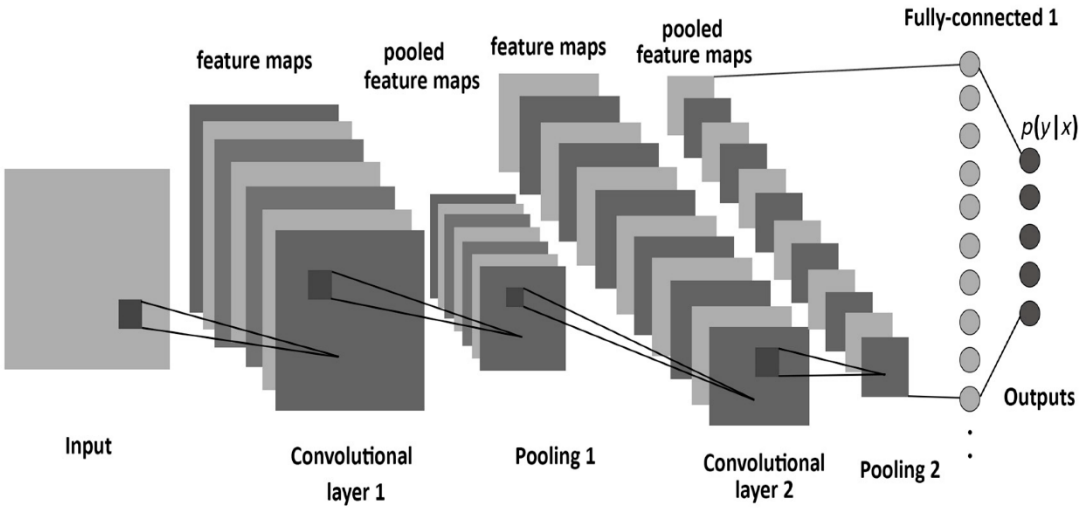


Figure.9 CNN

3.4 DEEP LEARNING WITH R (Keras)

Keras is a deep-learning framework that provides a convenient way to define and train almost any kind of deep-learning model. Keras was initially developed for researchers, with the aim of enabling fast experimentation.

Keras has over 150,000 users, ranging from academic researches and engineers at both start ups and large companies to graduate students and hobbyists. Keras is used by Google, Netflix, Uber, CETN, Yelp, Square, and hundreds of startups working on a wide range of problems.

several different backend engines can be plugged seamlessly into Keras. Currently, the three existing backend implementations are the TensorFlow backend, the Theano backend, and the Microsoft Cognitive Toolkit (CNTK) backend. Any piece of code that you write with Keras can be run with any of these backends without having to change anything in the code, where you can seamlessly switch between the two during development, which often proves useful-for instance, if one of these backends proves to be faster for a specific task.

Keras has the following key features:

- It allows the same code to run seamlessly on CPU or GPU.
- It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
- It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.

The Keras R package is compatible with R versions 3.2 and higher. The documentation for the R interface is available at https://keras.rstudio.com.

The main Keras project website can be found at https://keras.io.

3.5 MNIST DATASET

The MNIST dataset is a classic dataset in machine-learning community, where MNIST considered as “hello word” of deep learning, which has been around almost as long as the field itself and has been intensively studied.

It’s a set of 60,000 training images, plus 10,000 test images of handwritten digits, each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The MNIST dataset assembled by the National Institute of Standards and Technology (the NIST) in the 1980s.

Keras provides a multilayer perceptron (MLP) example that is trained on MNIST dataset to give a 98.40% test accuracy after 20 epochs.

(https://keras.rstudio.com/articles/examples/mnist_mlp.html)

CHAPTER 4

PRINCIPLE COMPONENT ANALYSIS

Principal component analysis (PCA) is a dimensionality reduction algorithm that can be used to significantly speed up unsupervised feature learning algorithms and can improve the neural network's convergence speed and the overall quality of results.

Principal Component Analysis seeks to transform the original variable to a new set of variables that are linear combinations of the variables in the data set, uncorrelated with each other, and ordered according to the amount of variation of the original variables that they explain.

4.1 THE MATHEMATICAL INTERPRETATION OF PCA

PCA relies on three statistical ideas:

1.Mean: simply the average value of all x in the set of **X**, which is found by dividing the sum of all data points by the number of data points n.

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

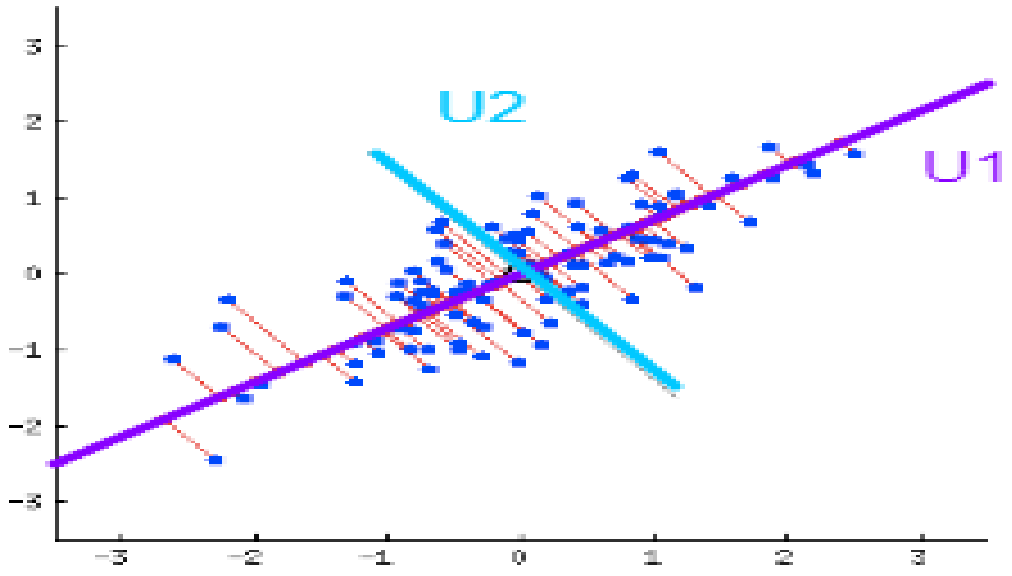
2.Standard Deviation: the square root of the average square distance of the data points to the mean.

$$s = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \bar{X})^2}{(n-1)}}$$

3.Variance: the measure of the data's spread (the square of the standard deviation).

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

- To understand how PCA works let us assume that we have a plot of age (x axis) and height (y axis) of individuals as below:



PCA will find a lower dimensional subspace to project the data on, from the plot we can see that U1 is the principle component that shows the highest direction of variation of our data, U2 is the secondary direction of variation, if it was three-dimensional data, a third component would fit (the number of principle component will increase by the increasing of data's dimensions).

- Covariance Matrix:

  Mathematically the first thing PCA does is finding the covariance matrix which is a matrix that describe the variance of the data, and the covariance among variables (an empirical description of data we observed), next, computing the eigenvectors of the covariance matrix and stack them in columns to form the matrix U:

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & & | \end{bmatrix}$$

- Eigenvectors: when a matrix performs a linear transformation, eigenvectors trace the lines of force it applies to the input.
- Eigenvalues: they are the measure of data's covariance, by ranking them from the highest to the lowest to get the principle component in order of significance.

➢ Rotating and reducing the data dimensions:

Because the eigenvectors of the covariance matrix are orthogonal to each other, they can be used to reorient the data from the x and y axes to the axes represented by the principle components (i.e. re-base the coordinate system of the dataset in a new space defined by its lines of greatest variance)

$$x_{\mathbf{rot}}^{(i)} = U^T x^{(i)} \quad (1)$$

where X_{rot} is an n dimension vector, and the few components are likely to be large, and the later components are likely to be small, so what simply PCA does is dropping the later (smaller) components of X_{rot} and just keep the first K components to define our data as a k-dimensional vector instead of n-dimensional vector (n < k).

from the equation (1) we can explain our data by the first principle component only, to reduce the data dimensions to one dimension, by setting:

$$\tilde{x}^{(i)} = x_{\mathbf{rot},1}^{(i)} = u_1^T x^{(i)}$$

➢ Number of the component to retain (explained further in the next section):

To decide how to set k, we will usually look at the percentage of variance retained for different values of k. Concretely, if k = n, then we have an exact approximation to the data, and we say that 100% of the variance is retained. I.e. all the variation of the original data is retained. Conversely, if k = 0, then we are approximating all the data with the zero vector, and thus 0% of the variance is retained.

More generally, let λ_j is the eigenvalue corresponding to the eigenvector U_j, then if we retain k principal components, the percentage of variance retained is given by:

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j}.$$

4.2 THE METHODS OF VARIABLE SELECTION

Four methods of variable selection based on PCA are explained in Jolliffe's book [27] B1Backward, B1Forward, B2 and B4.

4.2.1 B1Backward Method

In this method, a PCA is performed on the original matrix of K variables and n observations. The eigenvalues (λ) are used to select the number of component axes to evaluate based on some criterion λ_0. If p1 components have eigenvalues less than λ_0, then the eigenvector coefficients (i.e. loadings) on the remaining K- p1 components are evaluated starting with the last component (i.e. the component with the lowest l value). The variable associated with the highest eigenvector coefficient (i.e. the highest loading) is then discarded from each of the K - p1 components. Another PCA is performed on the remaining K - p1 variables and the selection process is repeated with the same criteria, such that K- p1 - p2 variables remain. Principal component analyses are repeated until all components have eigenvalues higher than λ_0.

Jolliffe (1972) did not evaluate method B1Backward, because he considered it too time consuming, but did suggest $\lambda_0 = 0.70$ as a suitable criterion and argued that a cut-off of $\lambda_0 = 1.0$ retains too few variables.

4.2.2 B1Forward Method

Like the B1Backword method but instead of discarding variables starting with the final component, variables are retained starting with the first.

4.2.3 Method B2

Method B2 is the same as B1Backward but requires only one PCA to be performed on the original K by n matrix. Based on some cut-off criteria, if p variables are to be retained then K – p variables are rejected backwards from the last component.

Krzanowski (1987) has suggested that it is acceptable to arbitrarily choose p variables from K variables, provided that the amount of variance retained by the resulting p components is acceptable. A good rule of thumb for the ratio of observations to variable is 3 :1.

4.2.4 Method B4

Method B4 retains variables by starting with the first component and keeping the variable with the highest loading. All K - p remaining variables are rejected. The criteria levels used to select p in methods B2.

PCA can be used to identify those variables that contain the most information. If resources become limited, the selected variables may provide a suggestion for future data collection in ecological studies. We suggest that ecologists use the B4 selection method and Broken-stick cut-o€ criteria to select subsets of variables from large environmental datasets to use in subsequent statistical analyses [28].

CHAPTER 5

MATERIALS AND METHODS

5.1 GENE EXPRESSION DATA

For our analysis we analyzed RNA-seq expression data from The Cancer Genome Atlas (TCGA) database for both tumor and healthy breast samples. These data consist of 1102 breast cancer samples and 113 healthy samples. By using Tidyverse and broom Packages in R, we download the data and tidied it.

```
```{r}
library(SummarizedExperiment)
library(tidyverse)
library(biobroom)
library(stringr)


if (!file.exists("tcga_brca_rnaseq_data.rds")) {
  download.file("https://s3-us-west-2.amazonaws.com/veri
analizi/tcga_brca_rnaseq_data.rds","tcga_brca_rnaseq_data.rds",
mode="wb")}


data <- readRDS("tcga_brca_rnaseq_data.rds")
exp_metadata <- as.tibble(colData(data))


names(rowRanges(data)) <- paste0(rowRanges(data)$ensembl_gene_id,"-
",rowRanges(data)$entrezgene)


tidy(data) %>% head()
```
```

```
exp_metadata %>%
  select(1:3,5,10,11,22,23,28:29,31:33,41:44,46,55) %>%
  mutate(barcode=str_replace_all(barcode,"-","."),

age_at_diagnosis=as.integer(ceiling(age_at_diagnosis/365.2424)))%>%
  inner_join(tidy(data), by=c("barcode"="sample")) %>%
  select(gene,name,value,everything()) %>%
  select(gene,name, value,sample) %>%
  spread(sample,value) # -> tcga_brca_rnaseq_data_wide
```

The data is in wide format, 21022 rows (gene) and 1216 columns (patient/sample). The column names contain patient ID, normal or tumor, cancer stage. This 3 information are united with "_" character.

gene	TCGA-3C-AAAU_Primary solid Tumor_stage x	TCGA-3C-AALI_Primary solid Tumor_stage iib	TCGA-3C-AALJ_Primary solid Tumor_stage iib	TCGA-3C-AALK_Primary solid Tumor_stage ia	TCGA-4H-AAAK_Primary solid Tumor_stage
ENSG00000000003-7105	188.1837	207.7216	1005.4397	1104.6752	942.5532
ENSG00000000005-64102	0.3447	1.0875	39.8912	1.2412	4.6809
ENSG00000000419-8813	524.2261	809.1354	967.3617	463.3844	486.3830
ENSG00000000457-57147	325.1408	1558.9396	335.8296	549.2553	416.2851

We divided our dataset to a train data with 972 samples (90 healthy and 882 cancer) and test data with 243 samples (23 healthy and 220 cancer)

5.2 DIMENSIONALITY REDUCTION USING PCA

PCA has been widely used in multivariate data analysis to reduce the dimensionality of the data. It has become a useful tool in microarray data analysis, where for a typical microarray data set, it is often difficult to compare the overall genes expression difference between samples or conduct the classification based on a very large number of genes [26]

For running PCA on our data, we had to change the structure of the data. PCA expects a matrix where samples (patients) are rows and measurements (genes) are columns.

Note that if we did not transpose the data we would ultimately get how are genes related of each other instead of extracting the genes that are most differently expressed in samples.

The gene expression values where normalized by applying the log () function. To avoid infinite values, genes with an expression values less than 1 are excluded.

```r
#Converting data, where genes become columns and samples are rows+
log function:
brca_data <- sample %>%
  gather("sample", "value", -gene)%>%
  group_by(gene)%>%
  filter(sum(value) > 0)%>%
  ungroup() %>%
  mutate(value = ifelse(value < 1, 1, value)) %>%
  mutate(value = log(value)) %>%
  spread(gene, value)%>%
  separate(sample, c("patient", "case", "stage"), sep = "_")
```

	patient	case	stage	ENSG00000000003-7105	ENSG00000000005-64102	ENSG00000000419-8813	ENSG00000000457-57147	ENSG00000000460-55732
1	TCGA-3C-AAAU	Primary solid Tumor	stage x	5.237419	0.00000000	6.261923	5.784258	4.822652
2	TCGA-3C-AALI	Primary solid Tumor	stage iib	5.336199	0.08388148	6.695966	7.351761	5.615554
3	TCGA-3C-AALJ	Primary solid Tumor	stage iib	6.913180	3.68615575	6.874572	5.816604	5.487641
4	TCGA-3C-AALK	Primary solid Tumor	stage ia	7.007307	0.21607865	6.138557	6.308563	5.385527

```r
#converting the data into a matrix:
```

```r
brca_matrix <- as.matrix(brca_data[,4:ncol(brca_data)])
row.names(brca_matrix) <- brca_data$patient

#converting cases to a numeric vector:
cases <- as.numeric(brca_data$case == "Primary solid
Tumor","Metastatic","Solid Tissue Normal")
```

Several functions from different packages are available in the *R software* for computing PCA:

- *prcomp*() and *princomp*() [built-in R *stats* package],
- *PCA*() [*FactoMineR* package]

- *dudi.pca*() [*ade4* package]
- *epPCA*() [*ExPosition* package]

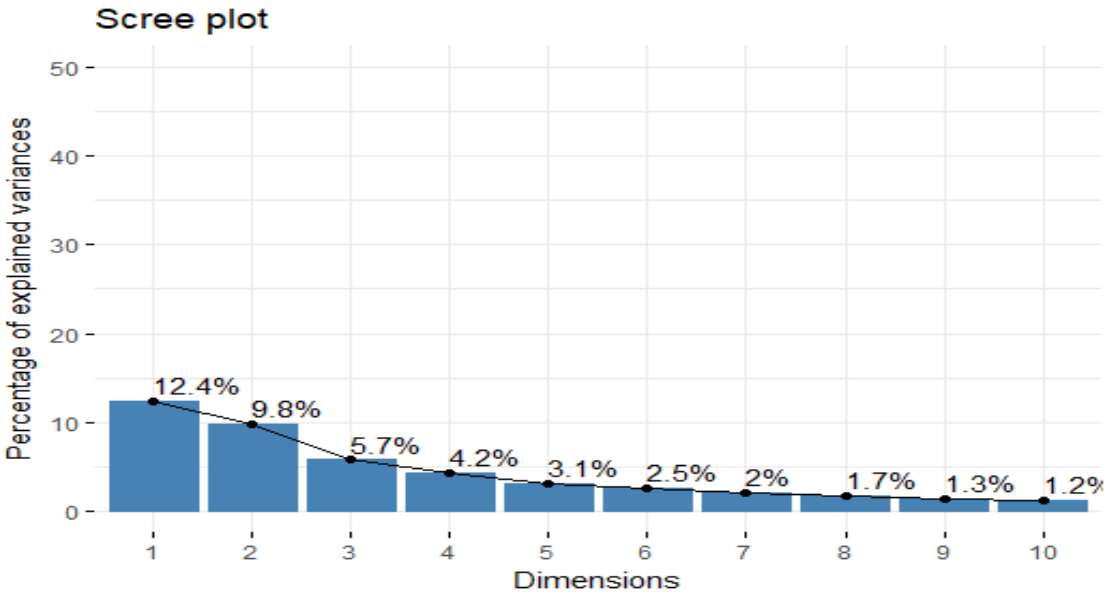
Here, we used the two packages FactoMineR (for the analysis) and factoextra (for ggplot2-based visualization)

```
#PCA:
library(FactoMineR)
library(factoextra)
library(ggplot2)
```{r}
res.pca<- PCA(brca_matrix, scale.unit = F,ncp =233,  graph = F)
```
```

The eigenvalues tell us how much of the original variance is captured in the direction of the corresponding eigenvector, which indicates how important each eigenvector is (large eigenvector means more important)

```
#Eigenvalue:
```{r}
eig.val <- get_eigenvalue(res.pca)

#plot of eigenvalues ordered from largest to the smallest
fviz_eig(res.pca, addlabels = TRUE, ylim = c(0, 50))
```
```



Unfortunately, there is no well- accepted objective way to decide how many principal components are enough to retain. This will depend on the specific field of application and the specific data set. One effective method is to determine the number of PCs that explain at least 80% of the variance in the data [27]. In our data 233 PCs explained 80.01262% of the variance in our data, therefore 233 PCs are retained.

Next, genes are selected according to their contributions to the 233 chosen PCs. Jolliffe [27] has proposed a procedure to choose variables with the largest absolute coefficients from each of the leading principle components. This procedure has been found to be quite effective in selecting informative variables.

The idea of this strategy can be extended to several genes (instead of just one gene) with the largest absolute coefficients from each leading principle component can be retained in the subset of the elected gene.

In order to get over the problem of bigger loadings in the last PCs which will lead to choose genes only from the last PC when using abs (), we used instead the percentage of contribution of variables in accounting for the variability in a given principle component.

The most differentially expressed 784 gene were selected and saved as RDS files

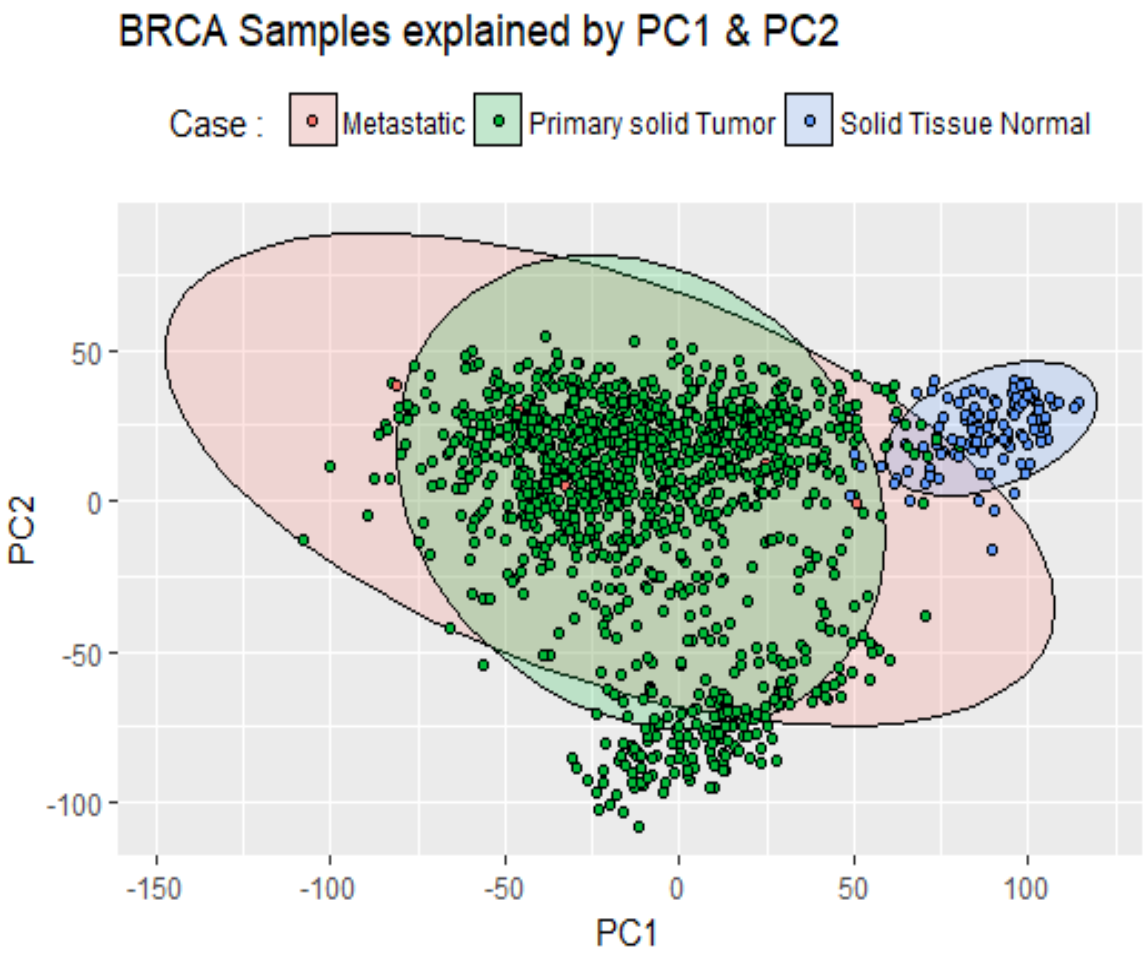
```{r}
var <- get_pca_var(res.pca)
top_genes <- apply(var$contrib, 1, max) %>% sort(decreasing = T) %>%
  head(784)
selected_genes <- as.data.frame(names(top_genes))
names(selected_genes)[1]<-paste("gene")
saveRDS(selected_genes, "selected_genes_pca.rds")

```

Some genes from the 784 selected genes:

1. ZIM2 gene (ENSG00000269699-5178)
2. MGAM gene (ENSG00000257335-8972)
3. TSIX gene (ENSG00000270641-9383)
4. CXADRP3 gene (ENSG00000265766-440224)
5. POLR2J2 gene (ENSG00000267645-246721)

```{r}
ind <- get_pca_ind(res.pca)
ind_pc1 <- as.data.frame(ind$coord[,1:233])
ggplot(ind_pc1, aes(ind$coord[,1], ind$coord[,2], col =
brca_data$case, fill = brca_data$case))+
  stat_ellipse(geom = "polygon", col ="black", alpha = 0.2)+
  geom_point(shape =21 , col ="black")+
  labs(title ="BRCA Samples explained by PC1 & PC2", x = "PC1" , y =
"PC2")+
  theme(legend.position = "top")+
  scale_fill_discrete(name = "Case : ")
```



5.3 PREPARING THE DATA FOR MLP

The MLP used in our study is designed to work on MNIST dataset, thus it expects a 4-dimensions tensor of shape (samples, height, width, channels).

By using array_reshape () function, we reshaped our data where every sample is a picture of 28 x 28 pixel and the are gray scale picture, so the train data has the dimension of (972, 28, 28, 1) and test data (243, 28, 28, 1)

Label arrays are also prepared for both train and test data. MLP will use the labels as targets where we set Normal = 0, Tumor or Metastatic = 1.

```{r}
library(tidyverse)
brca <- readRDS("brca_tidy_wide.rds")
selected_genes <- readRDS("selected_genes_pca.rds")
test_samples <- readRDS("test.rds")
train_samples <- readRDS("train.rds")

no_of_samples <- nrow(train_samples)

d_train <- brca %>%
  semi_join(selected_genes) %>%
  gather(sample, value, -gene) %>%
  semi_join(train_samples) %>%
  mutate(value =ifelse(value < 1, 1, value)) %>%
  #separate(sample, c("patient", "case", "stage"), sep="_") %>%
  mutate(log_value = log(value)) %>%
  select(-value) %>%
  spread(sample, log_value) %>%
  select(-gene) %>%
  as.matrix(byrow = TRUE) %>%
  { colnames(.) ->> train_sample_names
    dim(.) <- c(28,28,no_of_samples)
    .
  } %>%
  aperm(c(3,1,2))

dim(d_train)  # should be no_of_samples, 28 , 28
# d[1,1:28,1:28] should print data for first patient
 input <- array_reshape(d_train, c(no_of_samples, 28, 28, 1))

labels <- train_sample_names %>%
  as_data_frame() %>%
  filter(value !="gene") %>%
  # Normal=0, Tumor or Metastatic=1
  mutate(label= ifelse(grepl("Normal",value), 0, 1))  %T>%
  { count(., label) %>% print() }  %>%
  pull(label) %>%
  array()
'''
```

```{r}
no_of_test_samples <- nrow(test_samples)

d_test <- brca %>%
  semi_join(selected_genes) %>%
  gather(sample, value, -gene) %>%
  semi_join(test_samples) %>%
  mutate(value =ifelse(value < 1, 1, value)) %>%
  #separate(sample, c("patient", "case", "stage"), sep="_") %>%
  mutate(log_value = log(value)) %>%
  select(-value) %>%
  spread(sample, log_value) %>%
  select(-gene) %>%
  as.matrix(byrow = TRUE) %>%
  { colnames(.) ->> test_sample_names
    dim(.) <- c(28,28,no_of_test_samples)
    .
  } %>%
  aperm(c(3,1,2))

input_test <- array_reshape(d_test, c(no_of_test_samples, 28, 28,
1))

input_test <- array_reshape(input_test, c(nrow(d_test), 784))

labels_test <- test_sample_names %>%
  as_data_frame() %>%
  filter(value !="gene") %>%
  # Normal=0, Tumor or Metastatic=1
  mutate(label= ifelse(grepl("Normal",value), 0, 1))  %T>%
  { count(., label) %>% print() }  %>%
  pull(label) %>%
  array()
labels_test <- to_categorical(labels_test, num_classes)

```

5.4 TRAINING MLP

The MLP used in Keras on the MNIST data is used after changing layer units and add an additional layer_dense and layer_dropout to train on our samples.

```
```{r}
library(keras)
batch_size <- 128
num_classes <- 2
epochs <- 30

x_train <- input
y_train <- labels



x_train <- array_reshape(x_train, c(nrow(x_train), 784))

# Transform RGB values into [0,1] range
x_train <- x_train / 255

cat(nrow(x_train), 'train samples\n')

# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, num_classes)

# Define Model -----------------------------------------------------------
---------


model <- keras_model_sequential()
model %>%
  layer_dense(units = 1024, activation = 'relu', input_shape =
c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 512, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = 'softmax')
```
```

```r

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy'))

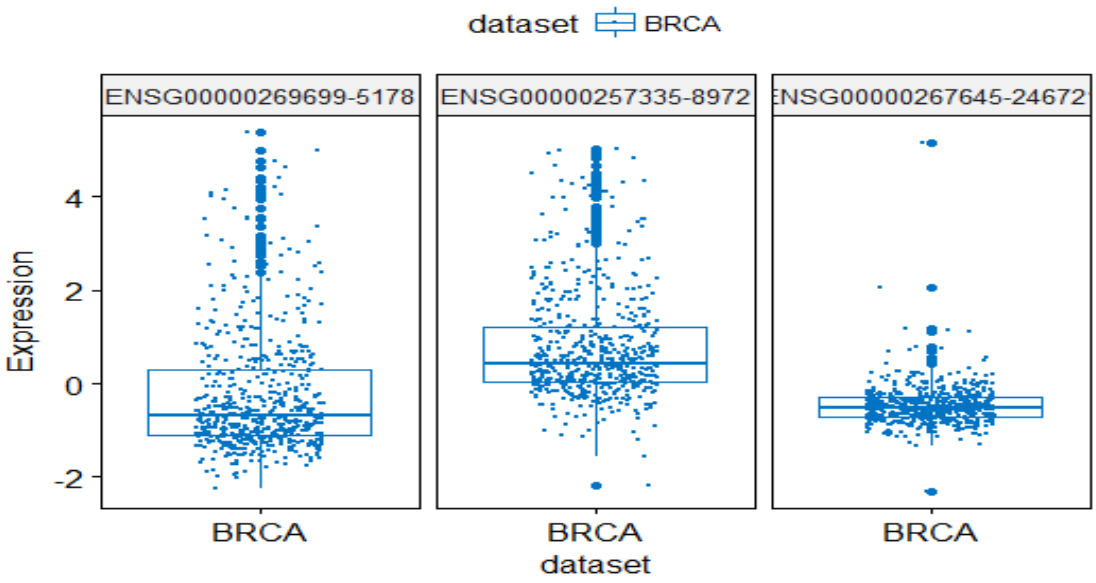
# Training & Evaluation
# Fit model to data
history <- model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  verbose = 1,
  validation_split = 0.2)
```

CHAPTER 6

RESULT AND DISCUSSION

We plot couple of our 784 selected genes to show the variance in samples, using ggpubr package:

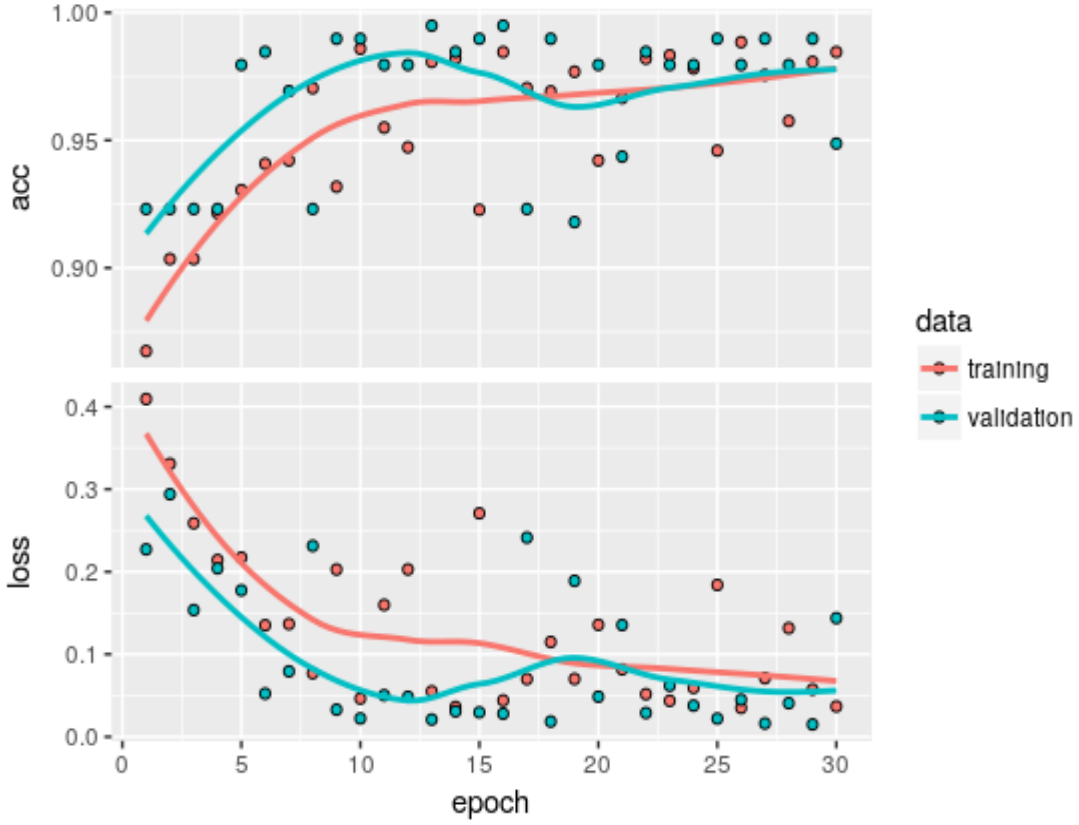
```
```{r}
library(RTCGA.mRNA)
expr <- expressionsTCGA(BRCA.mRNA, extract.cols = c("ZIM2", "MGAM","POLR2J2"))
expr$dataset <- gsub(pattern = ".mRNA", replacement = "", expr$dataset)
expr <- expr %>% rename("ENSG00000269699-5178" ="ZIM2", "ENSG00000257335-8972" = "MGAM", "ENSG00000267645-246721" = "POLR2J2")
```
ggboxplot(expr, x = "dataset",
          y = c("ENSG00000269699-5178", "ENSG00000257335-8972","ENSG00000267645-246721"),
          combine = TRUE,
          color = "dataset", palette = "jco",
          ylab = "Expression",
          add = "jitter",
          add.params = list(size = 0.1, jitter = 0.2))
```



The MLP where tested using the pre-prepared test data, and the test accuracy was %98.7.

```r
scores <- model %>% evaluate(
  input_test, labels_test, verbose = 1)

# Output metrics
cat('Test loss:', scores[[1]], '\n')
cat('Test accuracy:', scores[[2]], '\n')
plot(history)
```


1.00
0.95
0.90
acc
0.4
0.3
0.2
0.1
0.0
loss
0
5
10
15
20
25
30
epoch
data
training
validation

REFERENCES

[1]www.aihorizon.com/essays/generalai/supervised_unsupervised_machine_learning.htm

[2] http://www.cis.hut.fi/harri/thesis/valpola_thesis/node34.html

[3] Taiwo Oladipupo Ayodele (2010). Types of Machine Learning Algorithms, New Advances in Machine Learning,Yagang Zhang (Ed.), ISBN: 978-953-307-034-6,

[4] Alizadeh AA, Eisen MB, Davis RE, Ma C, Lossos IS, et al. (2000) Distinct type of diffuse large B-cell lymphoma identified by gene expression profiling.Nature 403: 503–510.

[5] Rost B, Sander C (1994) Combining evolutionary information and neural networks to predict protein secondary structure. Proteins 19: 55–72.

[6] Pedro Larran‹aga, Borja Calvo, Roberto Santana, Concha Bielza, Josu Galdiano, et al. (2005). Machine learning in bioinformatics.

[7] FRANCOIS CHOLLET, J.J.ALLAIRE.*_Deep Learning with R_*. New York: Manning Publications Co, 2018.

[8] Forrester. 2017. Five Factors That Make Deep Learning Different – Go Deep Baby!. [ONLINE] Available at: https://go.forrester.com/blogs/17-05-16-five_factors_that_make_deep_learning_different_go_deep_baby/.

[9] Manyika J, Chui M, Brown B, et al. Big data: the next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, 2011.

[10] Ferrucci D, Brown E, Chu-Carroll J, et al. Building Watson: an overview of the DeepQA project. AI Magazine 2010;31(3):59–79.

[11] Seonwoo Min, Byunghan Lee and Sungroh Yoon. Deep learning in bioinformatics.doi: 10.1093/bib/bbw068, 2016.

[12] Denas O, Taylor J. Deep modeling of gene expression regulation in an Erythropoiesis model. In: International Conference on Machine Learning workshop on Representation Learning.

[13] Neural Networks, https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introduction%20to%20neural%20networks .

[14] Nicholson, Chris V., and Adam Gibson. “Introduction to Deep Neural Networks(Deep Learning). Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, https://deeplearning4j.org/neuralnet-overview#intro

[15] “Artificial Neural Network Fundamentals.” Artificial Neural Network Fundamentals · UC Business Analytics R Programming Guide, http://uc-r.github.io/ann_fundamentals .

[16] Agrawal, Apoorva. “Loss Functions and Optimization Algorithms. Demystified.” Medium, Augmenting Humanity, 29 Sept. 2017, https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c

[17] Nicholson, Chris V., and Adam Gibson. “A Beginner's Guide to Multilayer Perceptrons. Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, https://deeplearning4j.org/multilayerperceptron.

[18] Chen Y, Li Y, Narayan R, et al. Gene expression inference with deep learning. Bioinformatics 2016;btw074.

[19] Erhan D, Bengio Y, Courville A, et al. Why does unsupervised pre-training help deep learning? J Mach Learn Res 2010;11:625–60.

[20] Asgari E, Mofrad MR. Continuous distributed representation of biological sequences for deep proteomics and genomics. PloS One 2015;10(11):e0141287.

[21] Lee T, Yoon S. Boosted categorical restricted boltzmann machine for computational prediction of splice junctions. In: International Conference on Machine Learning, Lille, France, 2015. p. 2483–92.

[22] Nicholson, Chris V., and Adam Gibson. “Stacked Denoising Autoencoders.”. Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, https://deeplearning4j.org/stackeddenoisingautoencoder.html.

[23] Nicholson, Chris V., and Adam Gibson. “A Beginner's Guide to Deep Convolutional Neural Networks (CNNs).” Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, https://deeplearning4j.org/convolutionalnetwork.

[24] Ufldl.stanford.edu.(2018). PCA - Ufldl. [online] Available at: http://ufldl.stanford.edu/wiki/index.php/PCA .

[25] Liu, Jian, et al. “Tumor Gene Expression Data Classification via Sample Expansion-Based Deep Learning.” Oncotarget, vol. 8, no. 65, 2017, doi:10.18632/oncotarget.22762.

[26] Wang, Antai, and Edmund A. Gehan. “Gene Selection for Microarray Data Analysis Using Principal Component Analysis.” Statistics in Medicine, vol. 24, no. 13, 2005, pp. 2069–2087., doi:10.1002/sim.2082.

[27] Jolliffe, I. T. “Discarding Variables in a Principal Component Analysis. II: Real Data.” Applied Statistics, vol. 22, no. 1, 1973, p. 21., doi:10.2307/2346300.

[28] King, Jacquelynne R., and Donald A. Jackson. “Variable Selection in Large Environmental Data Sets Using Principal Components Analysis.” Environmetrics, vol. 10, no. 1, 1999, pp. 67–77., doi:10.1002.