THE PREDICTION OF CERTAIN PROTEIN FUNCTION BY ANALOGOUS SEQUENCES USING DEEP NEURAL NETWORK APPROACH

A Project

presented to

the Faculty of the Department of Computer Science

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

PEI HU

Dr. Jianlin Cheng, Project Supervisor

Dec 2016

The undersigned, appointed by the dean of the Graduate School, have examined the project entitled

IMPROVED PREDICTION OF CERTAIN PROTEIN FUNCTION BY

ANALOGOUS SEQUENCES USING

DEEP NEURAL NETWORK APPROACH

presented by Pei Hu,

a candidate for the degree of master of science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Professor Dong Xu

Professor Yunxin Zhao

Professor Jianlin Cheng

ACKNOWLEDGEMENTS

To begin with, I would devote me deepest and most sincere gratitude to my advisor Prof. Jianlin Cheng without hesitation, not only for his attentive guidance but also for his kind patience throughout the time when I was doing the research. He also encouraged me a lot during when process when I was pursuing the degree of master of science, not just in academic aspect, but in life and future as well. His support is one of the very critical elements towards the accomplishment of the project.

Furthermore, I would like to thank my committee members, Prof. Dong Xu, Prof. Yunxin Zhao because of their patient reviewing towards my project, and suggestions for me to improve that. Their warm-hearted help is also very crucial to me.

In addition, I would like to express my very sincere thanks to Jie Hou, Rui Xie, Jilong Li, and Renzhi Cao for their selfless help and advice. They also enlightened me to form up the methods and models of this project under their guidance.

Last but not the least, I would thank my family and my friends for their supports and unrequited love all along from the deep of my heart.

TABLE OF CONTENTS

Appendix

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

Understanding functions of proteins is very crucial to understand the process of life in molecular level. Also, it can be very useful in benefiting human beings' life, by helping people understand many diseases where certain proteins are substituted falsely, for example. Tradition ways of determining the functions of proteins are fairly expensive by doing experiments because of the mass amount of labor and resource invested. Here, a classification model is proposed based on deep learning to determine whether proteins feature a certain function annotated by Gene Ontology (GO) terms, which also can be scaled to predict multiple functions. It is found that the results generated by this model based on deep learning are considerably improved comparing with results by Basic Local Alignment Search Tool (BLAST) developed by National Center for Biotechnology Information (NCBI), also based on analogous sequences, on dataset of SWISS-PROT, which is a non-redundant sequence database annotated and reviewed manually.

This project has made several contributions. On one hand, the model based on deep learning is brought up using multilayer perceptron and stacked denoising autoencoders, after utilizing BLAST, generating feature, and preprocessing. After that, the model is improved by using dropout technique to prevent the model from overfitting where it can be compared with the other approach, BLAST, under this circumstance. On the other hand, a software package is developed for users to utilize, get predictions and act as references.

CHAPTER 1

Introduction

In this chapter, we are going to introduce the very outline in protein function prediction which would be discussed in detail later on. Also, the background and progress of this researching area would also be elaborated. Furthermore, a rough sketch of each chapter would be given to help get the basic idea of this paper.

1.1 Introduction to Protein Function Prediction

Protein function prediction is one of the most concerned topics in bioinformatics area nowadays because of its importance to understanding life form at the genome level. Protein function is very critical for people to understand the biological processes, thus, be more familiar with many aspects of life. For instance, the disease may occur on people if some proteins in the body were substituted with the other proteins, especially those are malign[1].

Traditionally, protein function is determined by experiments manually to get the most accurate result by investing a considerable amount of human labor and resources. For example, a recent approach like a combination of fluorescent like microscopy with electron microscopy was taken for good results[2].

Therefore, computational methodologies are adapted to determine protein function by predictions using features which are already acquired, in order to reduce the cost of the procedure to determine the function. Many approaches have been made to settle the matter. In this way, develop a proper approach in a certain way to solve the problem is pretty helpful in this area.

In most circumstances, approaches tend to predict protein function is concerning the very procedure of machine learning used as a classification tool, based on features extracted. In this way, a robust and good-performing machine learning methods is important as well towards the problem. Deep learning came into researchers' sight in recent years for its outstanding performance in many fields. By extracting proper features concerning protein function prediction, this method is quite promising.

1.2 Existing Approaches and Challenges

A variety of methods have been proposed, which includes approaches making use of sequence similarity[3], approaches based on evolutionary analysis[4], approaches based on protein structure[5], approaches utilizing protein fusion and interaction[6], and approaches by classifying protein family[7].

Getting predictions by aligning sequences is one of the most intuitive and primitive approaches. Also, in most cases, analogous sequences approach is the very

premier step in many other approaches like using protein-protein interaction, protein domain co-occurrence networks[8]. In this way, improving final result by getting a good preliminary result by aligning target sequences with other sequences is very important.

Noises in sequence searching and aligning procedure may affect the prediction accuracy of protein function, even though many other factors, protein structure[9], for example, may also affect the accuracy of the prediction. Position-Specific Iterated BLAST (PSI-BLAST) is used widely in the direction of sequence-alignment-based approaches thanks to its speed, sensitivity and proper filtering of low complexity regions[10].

Basic Local Alignment Search Tool (BLAST) is a program used for sequence similarity searching[11]. By given a database of sequences, BLAST can be used to compare the target sequence with the sequences in the database. Heuristic strategies were used to align the target sequence with the sequences in the database. Also, more statistical information was provided during the process of aligning, like expect value. Once two sequences, one of which is target sequence, the other one of which is selected from the database, is given, BLAST would seek two equal-length segment sequence to get aggregate score by aligning them. In this procedure, a scored substitution matrix is used to determine a score when a pair of amino acid *i* and *j* is aligned. Mostly, BLOSUM-62 is used as substitutiona matrix**Error! Reference source not found.**. So, we would also take BLOSUM-62 as substitution matrix in this project. When the scored of pairs of segments cannot be improved by extensions or trimming, the pairs of alignment would be called high-scoring segment pairs (HSPs). The normalized score S' for HSPs can be formulized as:

$$S' = \frac{\lambda S - \ln K}{\ln 2}$$

Where S is nominal HSP score, λ and K are two calculable parameters.

PSI-BLAST is actually BLAST applied to position-specific score matrices. In this way, a position-specific matrix is applied during the aligning process instead of using the very sequence to get the position-specific substitution score. Thus, the more specific score can be generated from PSI-BLAST for a larger scale of scoring purpose.

By using BLAST, a most analogous protein sequence can be found, whose function can be used as a reference to predict the function of the original sequence. However, concerning of giving one certain protein function annotation, the accuracy of simply adapting the most analogous protein sequence would not show a good result because sometimes the function of the most analogous sequence might not necessarily be the same with the target protein sequence given. Also, this method cannot give a definite protein function annotation among analogous sequences' properties without using specific criteria to decide. A machine learning method can be used to determine which specific function annotation is most likely to be predicted as the function of the target protein which needs to be predicted.

1.3 Project Outline

The overview of the Project would be demonstrated as below.

In the following chapter, say, chapter 2, the prediction model based on deep learning would be elaborated in detail. This model is developed based on Pylearn2[13], which is a machine learning research library in Python developed by LISA lab at

Université de Montréal. This method mainly consists of three steps. The first step is feature extracting by getting proper features from the raw data gathered from open-source scientific and academic sources. The second step is developing the protein function prediction software based on deep learning, training multilayer perceptions and stacked autoencoders with features given through back propagation, also with overfitting-prevention techniques adapted. The last step is using cross-validation to get the more accurate result on the model, also comparing with the result of BLAST-based prediction methodology.

In Chapter 3, we will discuss preventing overfitting for MLP in deep learning by using dropout and early stopping techniques, which are adapted to the model to generalize the model better.

In Chapter 4, we will summarize the results given by the implemented model. Also, we will discuss the future work and limitations.

In Appendix A, we will have a brief introduction to pylearn2, which is mostly used in the implementation of the model discussed in this project. We will also provide the scripts used for pre-processing and YAML files used in the implementation, which will also be illustrated. In addition, a basic user manual would also be provided in how to use the software package, including the input and output requirements and guideline in how to run the programs and predictions.

Broadly speaking, this project has made two contributions. On one hand, it provided an approach to protein function prediction by utilizing deep learning, which can be a promising tool to get a fairly good preliminary result for more comprehensive

approaches. On the other hand, a software package was provided to get predictions, which can be used for academic area and also the community in order to promote the research in protein function prediction.

CHAPTER 2

Prediction Model based on Deep Learning

2.1 Abstract

In this chapter, we will introduce our methods, including datasets, and preprocessing procedures, prediction models based on deep learning.

We will describe what are protein functions and how to annotate these protein functions. After that, we are going to introduce the source of our dataset, and how do we extract features and targets for both training data, as well as testing data. Also, we will discuss how to preprocess the data so that we can use those data for the prediction model. The prediction model is based on deep learning, which consists of multilayer perceptron and stacked denoising autoencoders. Firstly, we need to train the autoencoders by using stochastic gradient descent with features which have already been preprocessed. Secondly, we need to train the multilayer perceptron with the stacked autoencoders which have already been trained. During the process, backpropagation is also used during the process of optimization.

After we get the result from the model, we will compare the result with the result using PSI-BLAST to see if we could get a better performance.

2.2 Background

In this section, we will introduce what protein functions are, and how we would annotate these functions. Also, we will introduce the source dataset we used in our model.

2.2.1 Introduction to Protein Function

Protein function is a pretty intuitive concept from these two words literally. These functions are very important considered as an aspect of protein characteristics. In this project, we will use a widely-used annotation method to annotate the protein functions referred, the Gene Ontology database. The Gene Ontology database provides structured, controlled vocabularies and classifications biologically[14]. Gene Ontology terms would be used to describe the properties and the functions of the proteins, which is generally used in the community of researching area related to protein function research. There's three major domains included in Gene Ontology vocabularies: Cellular Component, Molecular Function, and Biological Process. For Cellular Component, it means parts of a cell or extracellular environment. For Molecular Function, it stands for activities of gene products at molecular level. For Biological Process, it provides the meaning of operations, process or events with symbols of beginning and ending defined, say, the function of integrated living units, including cells, organs, etc.

In this project, we are going to use Gene Ontology terms, say, GO terms, to describe the functions we are going to predict. By using these terms, there are many evident benefit: they are widely-accepted in scientific communities; they are much easier and more efficient to utilize in most models, also ours; etc.

2.3 Introduction to Data Sets

In this section, we are going to describe the raw data set we used in our model. Also, we are going to illustrate the way in that we preprocess the data. By describing the data, and how it is preprocessed would help a lot in understanding the way the model described in this project works. In another word, it would be put even more obvious by telling the reader the more detailed information about the input data of the model.

2.3.1 SWISS-PROT

In order to get the most accurate and sophisticated situation in which this project could be landed, the very initiative step is to find out an appropriate data set. In this way, the data set should come from the real research results, with good accuracy, decent quality, low-redundancy and great annotation representation. SWISS-PROT seems to be the legitimate option for a decent data set[15]. SWISS-PROT is a protein knowledge database which contains current knowledge of amino acid sequences. The knowledge would include the overview of amino acid sequences, experimental results, characteristic features, and conclusions drawn in researches, etc. SWISS-PROT is manually annotated to ensure the high accuracy, high quality and low redundancy of the very knowledge base.

For each of the entry of SWISS-PROT, say, each piece of amino acid sequence, protein name, description, taxonomic data and citation information would be definitely included. Also, additional information would be provided, like protein function, enzyme-specific information, biologically related domains and sites, secondary structure, quaternary structure, and polymorphisms, and so on. In this way, we can get the information about each protein's function if it is contained in the database of SWISS-PORT. After that, we can extract the UniProtKB Accession number (AC number) as the identifier of each sequences of protein, for example, P00321. If more than one AC number is provided for each of the sequences, we would only use the first AC number listed, which is also called primary accession number, according to the convention in the area of biological research. At the same time, we can also extract the GO term as the identifier of the protein function of each sequence of protein, for example, GO:0046782. In most circumstances, there would be more than one GO terms for each of the sequences. Some of the sequences would probably be with no GO terms attached in the database of course. In this way, we will only adapt the sequences of protein with GO terms attached because the functions of those sequence are evident and meaningful. The SWISS-PROT database that we used was released in November 2015, which can be found in: ftp://ftp.uniprot.org/pub/databases/uniprot/previous_releases/release-2015_11/knowledgebase/knowledgebase2015_11.tar.gz. However, SWISS-PROT data set we used in this project can also be updated up to date till the version whichever user wanted. The only thing user need to do is to replace "uniprot_sprot.dat" and "uniprot_sprot.fasta" with the files with the same names that they desire. For example,

the latest version till this paper is written can be found in this directory:

ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/.

```
ID   001R_FRG3G              Reviewed;         256 AA.
AC   Q6GZX4;
DT   28-JUN-2011, integrated into UniProtKB/Swiss-Prot.
DT   19-JUL-2004, sequence version 1.
DT   01-APR-2015, entry version 30.
DE   RecName: Full=Putative transcription factor 001R;
GN   ORFNames=FV3-001R;
OS   Frog virus 3 (isolate Goorha) (FV-3).
OC   Viruses; dsDNA viruses, no RNA stage; Iridoviridae; Ranavirus.
OX   NCBI_TaxID=654924;
OH   NCBI_TaxID=8295; Ambystoma (mole salamanders).
OH   NCBI_TaxID=30343; Hyla versicolor (chameleon treefrog).
OH   NCBI_TaxID=8404; Lithobates pipiens (Northern leopard frog) (Rana pipiens).
OH   NCBI_TaxID=8316; Notophthalmus viridescens (Eastern newt) (Triturus viridescens).
OH   NCBI_TaxID=45438; Rana sylvatica (Wood frog).
RN   [1]
RP   NUCLEOTIDE SEQUENCE [LARGE SCALE GENOMIC DNA].
RX   PubMed=15165820; DOI=10.1016/j.virol.2004.02.019;
RA   Tan W.G., Barkman T.J., Gregory Chinchar V., Essani K.;
RT   "Comparative genomic analyses of frog virus 3, type species of the
RT   genus Ranavirus (family Iridoviridae).";
RL   Virology 323:70-84(2004).
CC   -!- FUNCTION: Transcription activation. {ECO:0000305}.
DR   EMBL; AY548484; AAT09660.1; -; Genomic_DNA.
DR   RefSeq; YP_031579.1; NC_005946.1.
DR   ProteinModelPortal; Q6GZX4; -.
DR   GeneID; 2947773; -.
DR   KEGG; vg:2947773; -.
DR   Proteomes; UP000008770; Genome.
DR   GO; GO:0006355; P:regulation of transcription, DNA-templated; IEA:UniProtKB-KW.
DR   GO; GO:0046782; P:regulation of viral transcription; IEA:InterPro.
DR   GO; GO:0006351; P:transcription, DNA-templated; IEA:UniProtKB-KW.
DR   InterPro; IPR007031; Poxvirus_VLTF3.
DR   Pfam; PF04947; Pox_VLTF3; 1.
PE   4: Predicted;
KW   Activator; Complete proteome; Reference proteome; Transcription;
KW   Transcription regulation.
FT   CHAIN         1    256       Putative transcription factor 001R.
FT                                /FTId=PRO_0000410512.
FT   COMPBIAS     14     17       Poly-Arg.
SQ   SEQUENCE   256 AA;  29735 MW;  B4840739BF7D4121 CRC64;
     MAFSAEDVLK EYDRRRRMEA LLLSLYYPND RKLLDYKEWS PPRVQVECPK APVEWNNPPS
     EKGLIVGHFS GIKYKGEKAQ ASEVDVNKMC CWVSKFKDAM RRYQGIQTCK IPGKVLSDLD
     AKIKAYNLTV EGVEGFVRYS RVTKQHVAAF LKELRHSKQY ENVNLIHYIL TDKRVDIQHL
     EKDLVKDFKA LVESAHRMRQ GHMINVKYIL YQLLKKHGHG PDGPDILTVK TGSKGVLYDD
     SFRKIYTDLG WKFTPL
//
ID   002L_FRG3G              Reviewed;         320 AA.
AC   Q6GZX3;
DT   28-JUN-2011, integrated into UniProtKB/Swiss-Prot.
DT   19-JUL-2004, sequence version 1.
DT   16-SEP-2015, entry version 31.
DE   RecName: Full=Uncharacterized protein 002L;
GN   ORFNames=FV3-002L;
OS   Frog virus 3 (isolate Goorha) (FV-3).
OC   Viruses; dsDNA viruses, no RNA stage; Iridoviridae; Ranavirus.
OX   NCBI_TaxID=654924;
```

Figure 1 Example Entries of SWISS-PROT Data File

Figure 1 is showing the content in “uniprot_sprot.dat”, which is the data file of SWISS-PROT, including almost all the information that we discussed above. Each entry is separated by a line consisted of two slashes, “//”. We can get the AC number from lines starting with “AC”, right after the tabulator sign. Additionally, we can fetch GO terms associated with that AC number in the same entry, in lines started with “DR”, tabulator sign and “GO;”. Also, the content in “uniprot_sprot.fasta”, which is the sequence file of SWISS-PROT, mainly containing AC numbers, name and sequence data of protein sequences. Each entry is started with “>sp”. The AC number is listed right after that. As soon as the first line of the entry is passed, sequence data is followed right after that line, which may be separated in many lines in order to make the format

neat and easy to read and process. The content of the sequence file is shown in Figure 2.

```
>sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate Goorha) GN=FV3-001R PE=4 SV=1
MAFSAEDVLKEYDRRRRMEALLLSLYYPNDRKLLDYKEWSPPRVQVECPKAPVEWNNPPS
EKGLIVGHFSGIKYKGEKAQASEVDVNKMCCWVSKFKDAMRRYQGIQTCKIPGKVLSDLD
AKIKAYNLTVEGVEGFVRYSRVTKQHVAAFLKELRHSKQYENVNLIHYILTDKRVDIQHL
EKDLVKDFKALVESAHRMRQGHMINVKYILYQLLKKHGHGPDGPDILTVKTGSKGVLYDD
SFRKIYTDLGWKFTPL
>sp|Q6GZX3|002L_FRG3G Uncharacterized protein 002L OS=Frog virus 3 (isolate Goorha) GN=FV3-002L PE=4 SV=1
MSIIGATRLQNDKSDTYSAGPCYAGGCSAFTPRGTCGKDWDLGEQTCASGFCTSQPLCAR
IKKTQVCGLRYSSKGKDPLVSAEWDSRGAPYVRCTYDADLIDTQAQVDQFVSMFGESPSL
AERYCMRGVKNTAGELVSRVSSDADPAGGWCRKWYSAHRGPDQDAALGSFCIKNPGAADC
KCINRASDPVYQKVKTLHAYPDQCWYVPCAADVGELKMGTQRDTPTNCPTQVCQIVFNML
DDGSVTMDDVKNTINCDFSKYVPPPPPPKPTPPTPPTPPTPPTPPTPPTPRPVHNRK
VMFFVAGAVLVAILISTVRW
>sp|Q197F8|002R_IIV3 Uncharacterized protein 002R OS=Invertebrate iridescent virus 3 GN=IIV3-002R PE=4 SV=1
MASNTVSAQGGSNRPVRDFSNIQDVAQFLLFDPIWNEQPGSIVPWKMNREQALAERYPEL
QTSEPSEDYSGPVESLELLPLEIKLDIMQYLSWEQISWCKHPWLWTRWYKDNVVRVSAIT
FEDFQREYAFPEKIQEIHFTDTRAEEIKAILETTPNVTRLVIRRIDDMNYNTHGDLGLDD
LEFLTHLMVEDACGFTDFWAPSLTHLTIKNLDMHPRWFGPVMDGIKSMQSTLKYLYIFET
YGVNKPFVQWCTDNIETFYCTNSYRYENVPRPIYVWVLFQEDEWHGYRVEDNKFHRRYMY
STILHKRDTDWVENNPLKTPAQVEMYKFLLRISQLNRDGTGYESDSDPENEHFDDESFSS
GEEDSSDEDDPTWAPDSDDSDWETETEEEPSVAARILEKGKLTITNLMKSLGFKPKPKKI
QSIDRYFCSLDSNYNSEDEDFEYDSDSEDDDSDSEDDC
>sp|Q197F7|003L_IIV3 Uncharacterized protein 003L OS=Invertebrate iridescent virus 3 GN=IIV3-003L PE=4 SV=1
MYQAINPCPQSWYGSPQLEREIVCKMSGAPHYPNYYPVHPNALGGAWFDTSLNARSLTTT
PSLTTCTPPSLAACTPPTSLGMVDSPPHINPPRRIGTLCFDFGSAKSPQRCECVASDRPS
TTSNTAPDTYRLLITNSKTRKNNYGTCRLEPLTYGI
>sp|Q6GZX2|003R_FRG3G Uncharacterized protein 3R OS=Frog virus 3 (isolate Goorha) GN=FV3-003R PE=3 SV=1
MARPLLGKTSSVRRRLESLSACSIFFFLRKFCQKMASLVFLNSPVYQMSNILLTERRQVD
RAMGGSDDDGVMVVALSPSDFKTVLGSALLAVERDMVHVVPKYLQTPGILHDMLVLLTPI
FGEALSVDMSGATDVMVQQIATAGFVDVDPLHSSVSWKDNVSCPVALLAVSNAVRTMMGQ
PCQVTLIIDVGTQNILRDLVNLPVEMSGDLQVMAYTKDPLGKVPAVGVSVFDSGSVQKGD
AHSVGAPDGLVSFHTHPVSSAVELNYHAGWPSNVDMSSLLTMKNLMHVVVAEEGLWTMAR
TLSMQRLTKVLTDAEKDVMRAAAFNLFLPLNELRVMGTKDSNNKSLKTYFEVFETFTIGA
LMKHSGVTPTAFVDRRWLDNTIYHMGFIPWGRDMRFVVEYDLDGTNPFLNTVPTLMSVKR
KAKIQEMFDNMVSRMVTS
>sp|Q6GZX1|004R_FRG3G Uncharacterized protein 004R OS=Frog virus 3 (isolate Goorha) GN=FV3-004R PE=4 SV=1
MNAKYDTDQGVGRMLFLGTIGLAVVVGGLMAYGYYYDGKTPSSGTSFHTASPSFSSRYRY
>sp|Q197F5|005L_IIV3 Uncharacterized protein 005L OS=Invertebrate iridescent virus 3 GN=IIV3-005L PE=3 SV=1
MRYTVLIALQGALLLLLLIDDGQGQSPYPYPGMPCNSSRQCGLGTCVHSRCAHCSSDGTL
CSPEDPTMVWPCCPESSCQLVVGLPSLVNHYNCLPNQCTDSSQCPGGFGCMTRRSKCELC
KADGEACNSPYLDWRKDKECCSGYCHTEARGLEGVCIDPKKIFCTPKNPWQLAPYPPSYH
QPTTLRPPTSLYDSWLMSGFLVKSTTAPSTQEEEDDY
>sp|Q6GZX0|005R_FRG3G Uncharacterized protein 005R OS=Frog virus 3 (isolate Goorha) GN=FV3-005R PE=4 SV=1
MQNPLPEVMSPEHDKRTTTPMSKEANKFIRELDKKPGDLAVVSDFVKRNTGKRLPIGKRS
NLYVRICDLSGTIYMGETFILESWEELYLPEPTKMEVLGTLESCCGIPPFPEWIVMVGED
QCVYAYGDEEILLFAYSVKQLVEEGIQETGISYKYPDDISDVDEEVLQQDEEIQKIRKKT
REFVDKDAQEFQDFLNSLDASLLS
```

Figure 2 Example Entries of SWISS-PROT Sequence File

2.4 Deep Learning Classification Model

It is evident to tell that the model is actually a classification model, since one sequence can only possess one certain protein function or not. In this way, what we are going to do is to build a classification model utilizing deep learning.

Here, we are going to exhibit the work flow of this specific model in Figure 3:

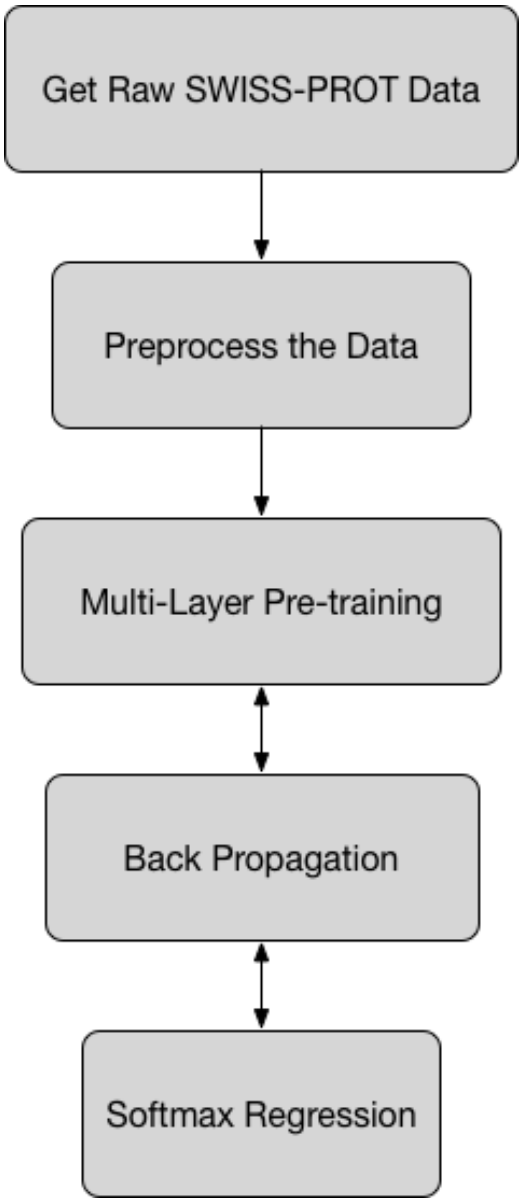

Figure 3 Deep Learning Process Workflow

As it is revealed in the work flow chart listed above, we can tell that, the first step of everything is getting the raw SWISS-PROT data. After that, we need to preprocess the raw data, in order to get the features and labels which are reasonable enough to train the model. In addition, the stacked auto-encoder would pre-train the model until it reached the top layer of softmax regression. The model would finetune with back propagation. Finally, this process would stop until the training algorithm has already converged.

2.4.1 Softmax Regression

Softmax regression is also called multinominal logistic regression, which is a more generalized form of logistic regression. We would assume the labels are all binary, for logistic regression, $y^{(i)} = \{0, 1\}$. However, softmax regression would allow us to handle more classes, thus, $y^{(i)} = \{1,2, \dots N\}$, where N stands for the number of classes. For training set $\{\left(x^{(1)}, y^{(1)}\right), \dots, (x^{(m)}, y^{(m)})\}$, with training index from 1 to m, where $y^{(i)} \in \{1,2, \dots, N\}$, in which N is the number of classes. Statistically, we would need the hypothesis to estimate the probability $P(y = n|x)$ for training entry x has the probability belongs to class n. The hypothesis:

$$h_\theta(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ P(y = 2|x; \theta) \\ \vdots \\ P(y = N|x; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{N} \exp\left(\theta^{(j)T} x\right)} \begin{bmatrix} \exp\left(\theta^{(1)T} x\right) \\ \exp\left(\theta^{(2)T} x\right) \\ \vdots \\ \exp\left(\theta^{(N)T} x\right) \end{bmatrix},$$

where $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(N)} \in \Re^n$ are the parameters of our model.

The softmax regression's cost function are usually defined in a form shown as follow:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{n=1}^{N} 1\{y^{(i)} = k\} \log \frac{\exp(\theta^{(n)T}x^{(i)})}{\sum_{j=1}^{N}\exp(\theta^{(j)T}x^{(i)})}\right].$$

In order to get the minimum value of $J(\theta)$, we would need to use an iterative optimization algorithm, say, in this case, stochastic gradient descent. We would take derivatives of the cost function, and use that derivative to minimize $J(\theta)$. The gradient would be described in the following formula:

$$\nabla_{\theta^{(n)}} J(\theta) = -\sum_{i=1}^{m}[x^{(i)}(1\{y^{(1)} = n\} - P(y^{(i)} = k|x^{(i)};\theta))],$$

where

$$P\left(y^{(i)} = k \middle| x^{(i)};\theta\right) = \frac{\exp(\theta^{(n)T}x^{(i)})}{\sum_{j=1}^{N}\exp(\theta^{(j)T}x^{(i)})}.$$

In the formula above, $\nabla_{\theta^{(n)}} J(\theta)$ stands for the partial derivative of $J(\theta)$ of class n's parameters, which can also be represented as $\frac{\partial J(\theta)}{\partial \theta_n}$.

2.4.2 Brief Introduction to Deep Neural Network

Deep neural network is a newly emerged machine learning technique, which is based on a set of algorithms getting the high level abstraction of data using multiple processing layers[16]. It was originally brought out by Dr. Geoffrey Hinton, to imitate the information abstraction happens in our brains[17]. Thanks to these multiple processing layers, deep neural network would be able to getting much more complex features of the original data.

2.5 Multilayer Perceptron

A multilayer perceptron (MLP) is a type of feedforward deep neural network, consisted with multiple layers of nodes in a directed graph, with each layer connected to the other next layer fully. Aside from the input nodes, all nodes are a neuron with nonlinear activation function. Also, the multiplayer perceptron uses backpropagation to train the neural network[18].

2.5.1 Activation Function

In order to make MLP difference normal two-layer models, some neurons' activation functions have to be nonlinear. Two most widely used activation functions are both sigmoid functions. They can be formulated as follow:

$$y(v_i) = \tanh(v_i)$$

andd

$$y(v_i) = (1 + e^{-v_i})^{-1}.$$

The range for both activation functions are -1 to 1, 0 to 1 respectively.

2.5.2 Layers

MLP is always considered as deep neural network because MLP consists of at least three layers of nonlinear activating nodes. In addition, those nodes are fully connected between adjacent layers.

2.5.3 Learning Utilizing Backpropagation

MLP changes nodes' weights gradually with data inputted. Those changes of weights are based on the error in the output, comparing with the expected result and utilizing backpropagation. Thus, MLP is evident to be categorized into supervised learning. The error between output node j in the nth input data point can be described as:

$$e_j(n) = d_j(n) - y_j(n),$$

where d is the target value and y is the output value generated by MLP. In order to minimize the error of the entire output, we would make correction towards the weight of each node. We would formulate the error correction as:

$$\varepsilon(n) = \frac{1}{2}\sum_j e_j^2(n).$$

We would use gradient descent to change the weight of each node, the weight change would be:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial v_j(n)} y_i(n),$$

where y_i is the output of the previous neuron, while η is the learning rate, and $v_j(n)$ is induced local field, which would let the weight meet its convergence reasonably.

It can be proved that the derivative in the previous formula can be simplified to:

$$-\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n)),$$

where ϕ' is actually the derivative of the activation function. It's hard to change the weights of a hidden node, but the relevant derivative is:

$$-\frac{\partial \varepsilon(n)}{\partial v_j(n)} = \phi'(v_j(n)) \sum_k -\frac{\partial \varepsilon(n)}{\partial v_k(n)} w_{kj}(n).$$

This change of weight always depends on the change in weights of the kth nodes, which represents the output layer. So whenever the changes of weight of nodes happen, the output layer weights would change first according to the derivative of the activation function, which is basically the concept of the back propagation.

2.6 Stacked Denoising Autoencoders

An autoencoders is an artificial neural network which is used for unsupervised learning of efficient codings[19]. The main goal of an auto-encoder is to learn a compressed, distributed representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. The autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

2.6.1 Autoencoders

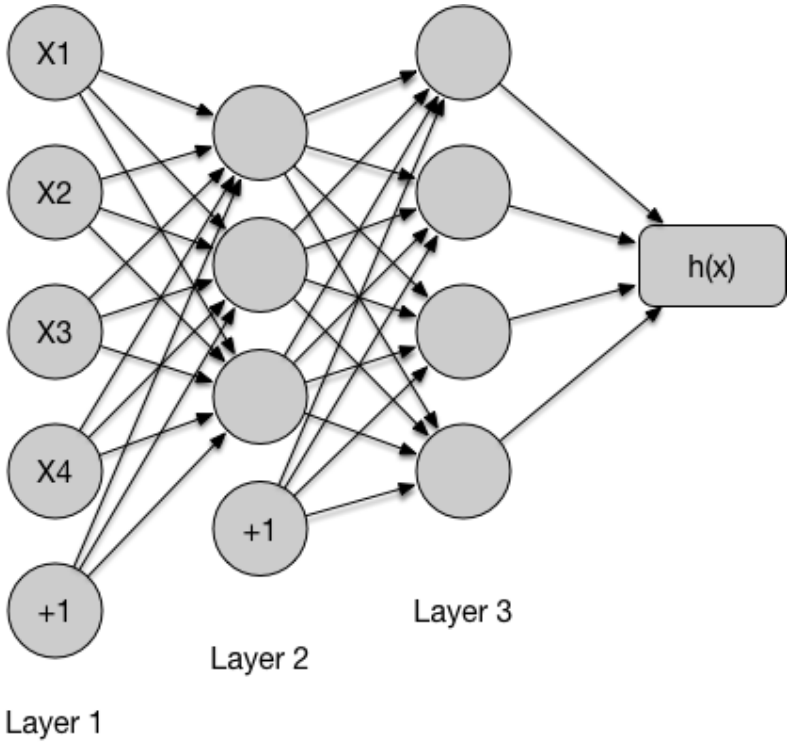

Figure 4 Autoencoder Example

The autoencoder tries to learn a function $h_{W,b}(x) \approx x$. In other words, it is trying to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x. A simple form of the autoencoder is a feed- forward, non-recurrent neural net, with an input layer, an output layer and one or more hidden layers connecting them. Although MLP is similar to a neural network, there is a difference between an MLP and an autoencoder. For an autoencoder, the output layer has equally many nodes as the input layer, and instead of training it to predict some target value y given inputs x, an autoencoder is trained to reconstruct its own inputs x and optimize through minimizing its objective function. In fact, a simple autoencoder often ends up learning a low-dimensional representation very similar to PCAs.

The training algorithm can be described like this. Firstly, for each input x, do a feed-forward pass to compute the value of all nodes in the hidden layers after activation, then at the output layer to obtain an output x. Then it measures the deviation of x from the input x (a common method is using mean squared error). Thirdly, back propagate the error through the net and perform weight updates (a common method is using the stochastic gradient descent algorithm). The activation functions that are used in autoencoders are ususally the same activation functions that are used in MLP, say, the sigmoid functions.

After these steps, the output of the autoencoder can be treated as the abstract representation of the input.

The average activation of hidden unit j, which is an average over the training data set would be formulated as follow, given $a_j^{(2)}$ represents the activation of hidden unit j:

$$\hat{\rho} = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}\left(x^{(i)}\right)\right],$$

Given ρ is a 'sparsity parameter', which is mostly a very small value, we would like ρ to be the same value as $\hat{\rho}$. In order to satisfy the constraint, the hidden unit's activation must mostly be near 0.

Penalty function is applied in order to optimize the objective preventing $\hat{\rho}$ deviating significantly from ρ. Kullback-Leibler divergence, which also called KL divergence, is often adapted to perform the role as the penalty function. The penalty

term that would give reasonable results is formulated as follow, given s_2 is the number of neurons in the hidden layer:

$$\sum_{j=1}^{s_2} KL(\rho||\widehat{\rho_j}),$$

where

$$KL(\rho\big|\big|\widehat{\rho_j}\big) = \rho \log\frac{\rho}{\widehat{\rho_j}} + (1-\rho)\log\frac{1-\rho}{1-\widehat{\rho_j}},$$

say, the Kullback-Leibler (KL) divergence. $KL(\rho\big|\big|\widehat{\rho_j}\big) = 0$ if $\widehat{\rho_j} = \rho$. In this case, the overall cost function would be:

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} KL(\rho||\widehat{\rho_j}),$$

where $J(W,b)$ is the cost function used in back propagation. The derivative of the cost function can be transformed into the formula listed follow:

$$\delta_i^{(2)} = ((\Big(\sum_{j=1}^{s_2} W_{ji}^{(2)}\delta_j^{(3)}\Big) + \beta(-\frac{\rho}{\widehat{\rho}} + \frac{1-\rho}{1-\widehat{\rho_i}}))f'(z_i^{(2)}).$$

2.6.2 Corrupted Level

In stacked denoising autoencoders, the partially corrupted output is denoised. The representation using stacked denoising autocoder can be still very robust even if the input is corrupted, thus it would be useful for recovering the corresponding clean input.

Recognizing the features within the noise that will allow it to classify the input is the main goal of the network. During the training of the network, a model is generated

and an objective function such as squared error measures the distance between that model and the benchmark through, it can be represented as the equation following:

$$L(xz) = ||x - z||^2,$$

or

$$L_H(xz) = -\sum_d^k [x_k \log z_k - (1 - x_k) \log(1 - \log z_k)],$$

where the former equation represents the square error for real value x, while the later equation is the cross-entropy objective for binary value x.

The procedure can be illustrated in the following figure:

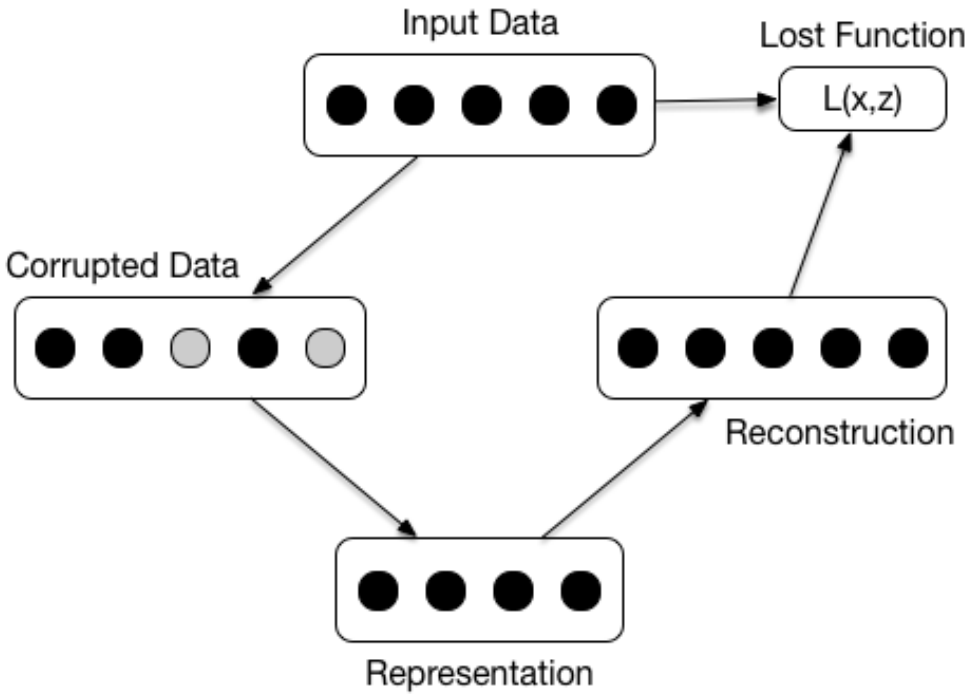

Figure 5 Example of an Autoencoder Corruption Model

The autoencoders firstly stochastically corrupts the input data. Then it maps to one representation. After that, it reconstructs the input via decoder producing the reconstructed result, where the reconstruction error is measured by loss $L_H(x, z)$.

2.7 Model for Protein Function Prediction Based on Deep Learning

By using deep learning architectures, we are able to predict if a sequence contains certain protein function in a machine learning methodology.

On one hand, we can produce gradually high-level representations of the input features through unsupervised learning approach layer by layer, by pre-training those layers. On the other hand, we would utilize supervised learning, say, the softmax regression to train the model getting the final result based on the high-level representation from the latest layer.

The model can be illustrated in a figure listed as the following:

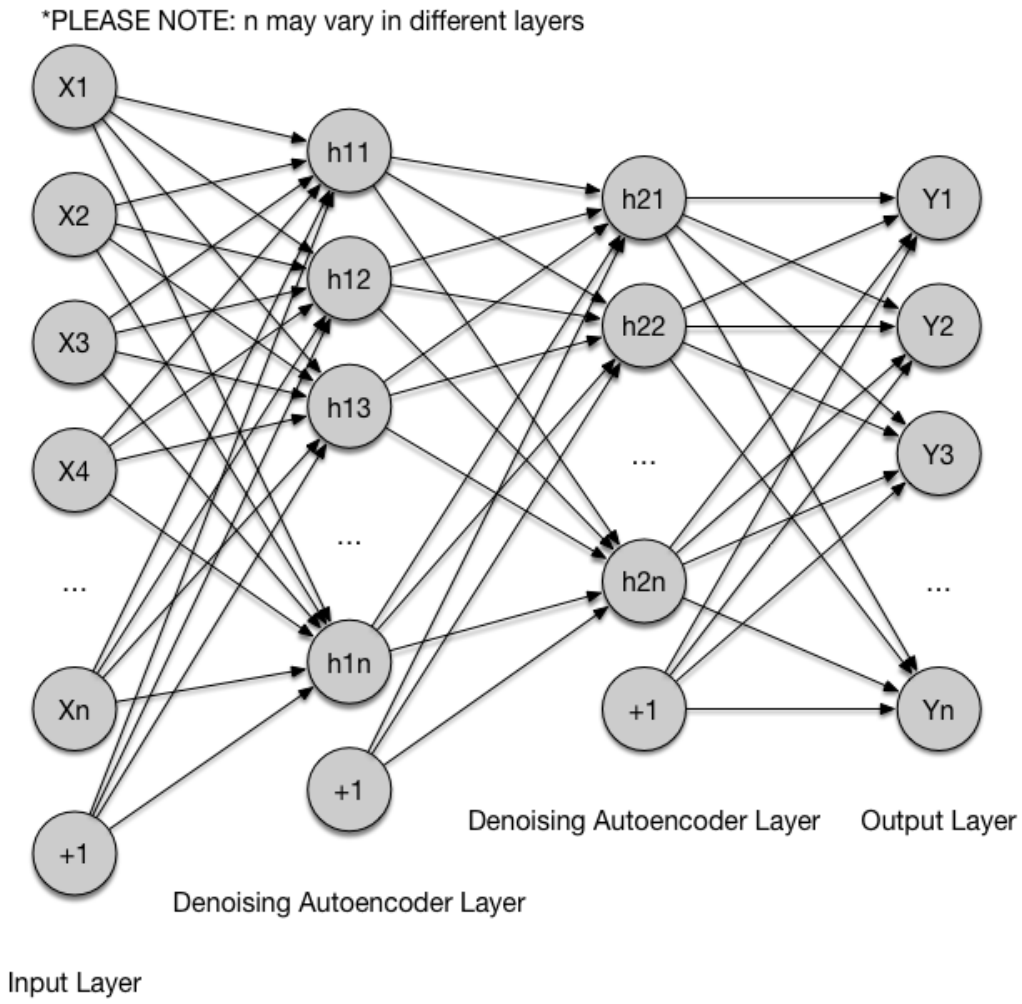

Figure 6 Model of the Deep Learning Architecture

Meanwhile, the training work flow of our model is represented as following:

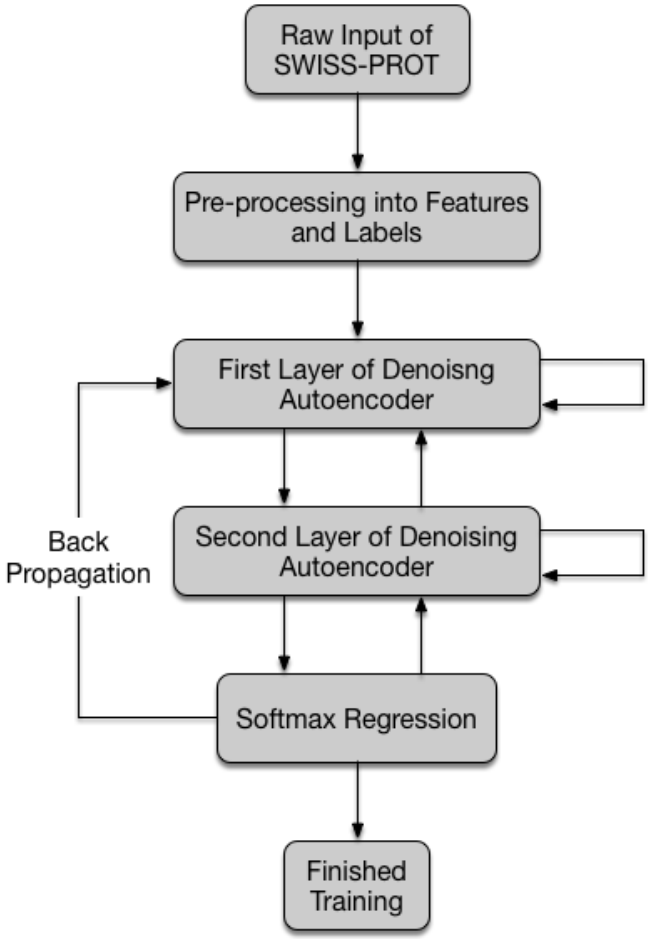

Figure 7 Training Work Flow of our Model

The first step is getting the raw data from SWISS-PROT database. Then, we pre-process the data and transform it into the training features and labels, which can be processed in this deep learning architecture. After that, the features would go through two layers of denoising autoencoders, which would generate the high-level abstracted features for the last layer of softmax regression to use, where the two denoising autoencoder layers would pre-train themselves in unsupervised learning criteria. Last but not the least, when the training reached the regression layer, the model would get trained as a whole, including two layers of denoising autoencoders, and one last regression layer, using back propagation.

2.8 Adapt Dropout to Prevent Model from Overfitting

In order to prevent the neural networks from overfitting, which is a serious problem in those neural networks, a technique called dropout is often adapted to address this problem[21]. The essence of this dropout technique is to randomly drop units from the neutral network along with their connections during the training period, which prevents units from co-adapting too much.

2.8.1 Introduction to Dropout Technique

Even though deep neural networks contain multiple non-linear hidden layers, which makes them capable of dealing complicated relationships between inputs and outputs, however, some of these relationships may be the result of sampling noise, which result in the overfitting of the model.

By dropping a unit out, it is commonly known as temporarily removing it from the neural network for every iteration, with all of its connections, where there exists a probability p deciding whether the specific unit needs to be dropped or not. This probability p can be determined by selection during the validation period in cases where cross validation is used. However, in most cases, $p = 0.5$ is picked because it seems to be the optimal probability in those circumstances.

The general idea can be illustrated in the following figure:

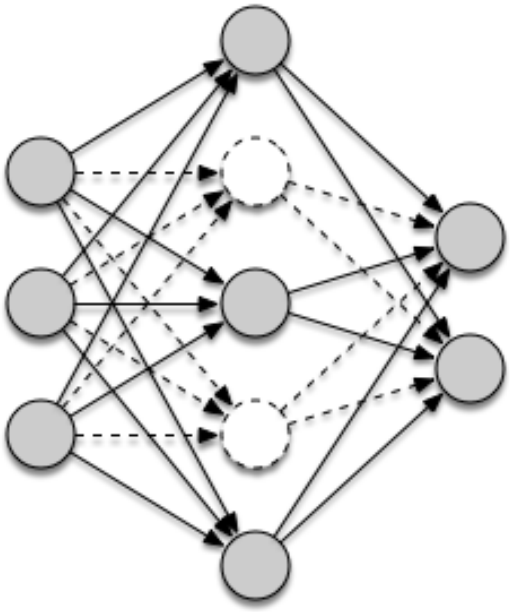

Figure 8 Illustration of Dropout

2.8.2 Early Stopping Technique

Early stopping technique is also a very important regularization form used in preventing models from overfitting when the model is getting trained using iteration-based machine learning techniques, such as gradient descent methodologies[22]. By adapting early stopping, the model can be improved on data outside of the training set, in a way of decreasing generalization error at the cost of decreasing learner's fit. In our methods, we would stop training after the model has ceased to improve after 80 epochs.

CHAPTER 3

Results and Summary

3.1 Results

Since we have already developed our model, we need to see how our model works. Since the training dataset and testing dataset is stochastically selected independently from the way larger dataset. Also, we do not need to modify too many parameters in our project. Thus, cross validation is not needed in this project. In this case, we would split the pre-processed data set into two independent part, training data set and test data set.

We would use the statistical measures of performance of a binary classification test, known as classification function to evaluate our model, such as, accuracy, precision and recall. Accuracy is actually the accurate prediction percentage among all the entries of the input; precision is the fraction of retrieved instances that are relevant; recall is the fraction of relevant instances that are retrieved.

Table 1 Condition Confusion Matrix

Total Population	Predicted Condition Positive	Predicted Condition Negative
Condition Positive	True Positive	False Negative
Condition Negative	False Positive	True Negative

Given the table listed above, precision and recall are defined as follow:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

and

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative},$$

while

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}.$$

Also,

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

We would consider F1 score as one crucial reference towards the performance of our model. It would consider both precision and recall of the test to compute the score, where it's range would be from 0 to 1, the higher the better. It is mathematically the harmonic mean of precision and recall.

3.1.1 Results with Different Learning Rate

In some cases, the learning rate of MLP can differ a lot on the result of stack autoencoders. In this section, we can tell if there is a great difference influencing the final result. Thus, selecting an appropriate learning rate is crucial to our model. The learning rate is selected from 0 to 1. Usually, when the learning rate is too large, the model might not converge. However, when the learning rate is too small, it would take a lot of time in the learning process, and also take the risk of getting caught into the local optima, which is definitely not we want when we are training our model. Our result can be listed in the following table:

Table 2 Table of Results with different Learning Rate

Leaning Rate (MLP)	Accuracy	Precision	Recall	F1
0.5	0.5325	0.237903226	0.324770642	0.274631497
0.1	0.789	0.825396825	0.286238532	0.42506812
0.05	0.76	0.849462366	0.144954128	0.247648903
0.01	0.7355	1	0.029357798	0.057040998
0.005	0.7525	0.946428571	0.097247706	0.176372712
0.001	0.7295	1	0.00733945	0.014571949

Also, it can be illustrated in the following figure:

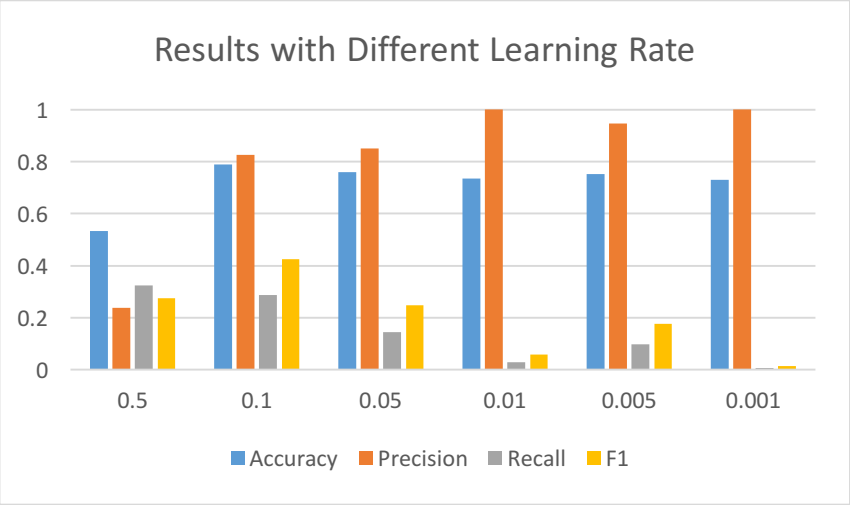

Figure 9 Results with Different Learning Rate

With the increment of the learning rate, the model is getting better in precision, but being not satisfying in recall. And the accuracy is decreasing at the same time, which means the model is sacrificing the generalization, to increase the precision with the increment of the learning rate, thus also decrease the overall accuracy. Note that when the learning rate is 0.5, the training process fails to converge. When we trying to seek for a grading criteria for the model, we would prefer the overall accuracy and the F1 score. In this case, the learning rate of 0.1 is proved to be the most generalized model, thus most useful.

3.1.2 Results with or without Dropout Technique

For the case where the learning rate is 0.1 or 0.01, we would see the results' difference with dropout adapted or not. We would list the result as follow, in tables and figures.

When the learning rate is 0.1, the results would be:

Table 3 Results of Adapting Dropout when the Learning Rate is 0.1

Dropout Technique Used	Accuracy	Precision	Recall	F1
Yes	0.789	0.825396825	0.286238532	0.42506812
No	0.758	0.568848758	0.462385321	0.510121457

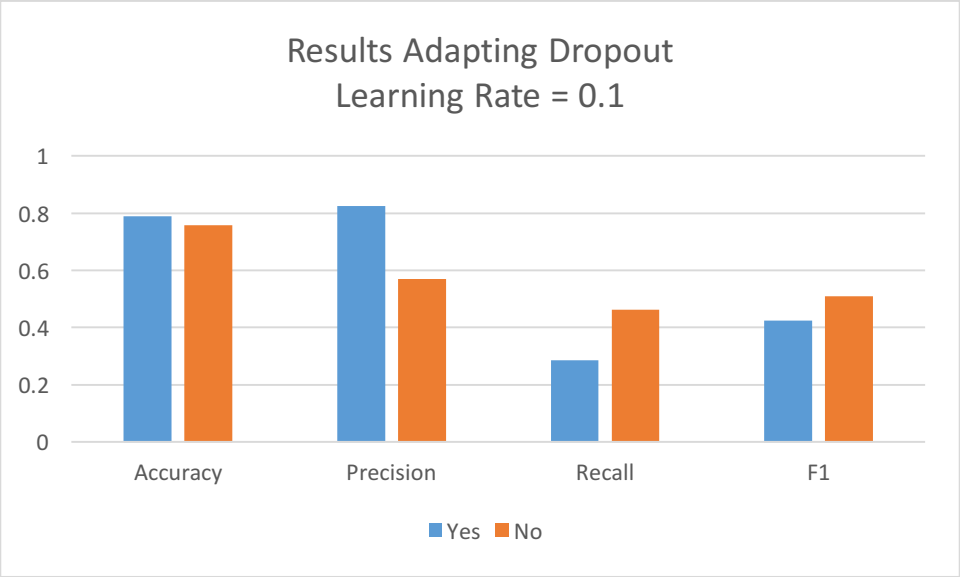

Figure 10 Figure of Results of Adapting Dropout when the Learning Rate is 0.1

When the learning rate is 0.01, the results would be:

Table 4 Results of Adapting Dropout when the Learning Rate is 0.01

Dropout Technique Used	Accuracy	Precision	Recall	F1
Yes	0.7355	1	0.029357798	0.057040998
No	0.739	0.529715762	0.376146789	0.439914163

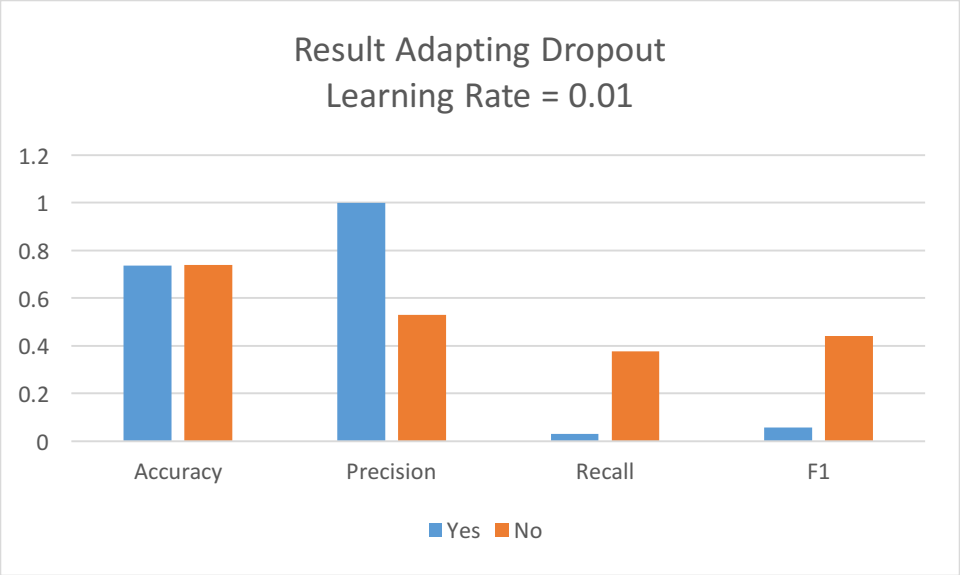

Figure 11 Figure of Results of Adapting Dropout when the Learning Rate is 0.1

It can be easily told that by adapting the dropout technique, the model is prevented from overfitting, thus increasing the accuracy on test data set.

3.1.3 Results only using PSI-BLAST

By selecting one of GO term list from most analogous sequence by PSI-BLAST, the accuracy of this methods is 0.270716793677.

3.2 Summary

Protein function prediction is a crucial part in Bioinformatics research. Many approaches have been developed to achieve the best result possible in order to reduce the cost and labor used in traditional experimental research.

Even though this project is a limited work towards the decent solution of the grand task, this project has pointed allusion to the use of broader machine learning approaches, say, deep learning, to solve the problem.

We used stacked denoising autoencoders to train our classification model in order to extract useful features. Then multilayer perceptron is adapted to get the final model using back propagation.

Since this project is a part of experimental work, instead of solving the problem thoroughly. The result of this project may not outperform many existing methods out there in research area. However, it can help give a reference, better than using pure PSI-BLAST. Also, it has revealed the great potential that deep neural networks can help solve the great task of protein function prediction, being a sophisticated machine learning approach.

In addition, the feature selection is critical to the solution of one task. By using analogous sequences as features would help determine the function of a protein. However, only using these features would probably not yield to a satisfying result, because the function of a protein would not fully be determined by the analogy of different sequences, except they are the same protein. Also, the definition of analogy is

vague, the tool used in our project is PSI-BLAST, which is certainly not the only option used in selecting analogous sequences.

The dropout technique can well control the problem of overfitting, which has been proved to be a practically technique used in deep neural networks in preventing overfitting.

3.3 Limitations and Future Work

Although the model we developed can predict a certain function of protein, it would probably not do a fascinating job based on the features we selected, which states the limitation of using only analogous features.

Firstly, our model can only determine whether a sequence owns a certain protein function due to the limitation of time, labor and calculation. However, if we run our model several time, with determining different protein function each time, we would be able to predict a very wide range of protein functions, which is also one step closer to the greater task of protein function prediction.

Secondly, we can build a web server later on, helping people get a referable result using our model with their own dataset.

Thirdly, we would try to use more features which seemingly closer to functions of proteins, like domain information, etc. As we can tell the importance of feature selection after this project, we are more willingly to discover more valuable features to help solve the problem with a much higher accuracy and better result.

Finally, we would try to use different machine learning approaches, comparing them, and getting their specialty and weakness on different features, including the analogous features we have selected in this project. Thus we could decide which machine learning to use with different features selected, which would give us better results.

Appendix

Supplementary Information

In this part we will introduce how our software package works, including some of the code and illustration. Also we will introduce the machine learning framework toolkit Pylearn2.

A.1 Introduction to Pylearn2

Pylearn2 is a machine learning library developed by LISA laboratory at Université de Montréal. This library is tend to facilitate the research using machine learning. Pylearn2 is built from re-usable parts, which can be combined and used independently. One of the greatest features of Pylearn2 is it can utilize both CPU and GPU functionality, thanks to the dependency of another lower level library, Theano.

There are three most important components which is used to implement most of the features, which are Dataset, Model and TrainingAlgorithm classes. A Dataset provides the data to be trained on. A Model is collect of parameters, which can specify the library what to do towards the dataset inputted. After above two elements are specified, a TrainingAlgorithm can train the model accordingly. The TrainingAlgorithm can minimize gradually the Cost which is specified in model until a TerminationCriterion is reached.

Even though the library is not complete with limited number of Models implemented because the library is bleed edging implementation, and only for research use. Pylearn2 can still make a great contribution to our project.

A.2 Preparation of the Raw Input Data

The environment required is a Linux system with latest versions (at least versions updated till the end of 2015) of Python2.7 with Biopython module, SciPy stack, NCBI BLAST with local Swiss-Prot database, and Pylearn2 installed and configured properly, especially when the machine is equipped with NVIDIA® CUDA graphic cards to accelerate the training of the model considerably. As for me, I have chosen the Ubuntu 15.10 with these software package installed and configured both on my local VMware® virtual machine as well as on an instance of Amazon Web Service™ (AWS) with AWS Command Line Interface installed.

In order to make this step easier to operate for users, we have already made a Linux shell file, generateDataset.sh to preprocessing and prepare the data automatically into the form which can be trained later.

```
#!/bin/bash
clear
if [ -d "sequences" ]
then
echo "Deleting and making directory \"sequences\""
rm -r sequences
mkdir sequences
sleep 3
clear
else
echo "Making directory \"sequences\""
mkdir sequences
sleep 3
clear
fi
echo "Running gen_AC_GO.py"
python gen_AC_GO.py
sleep 3
clear
echo "Running pickGO.py"
python pickGO.py
sleep 3
clear
echo "Running gen_AC_topGO.py"
python gen_AC_topGO.py
sleep 3
clear
echo "Running sel_AC_topGO.py"
python sel_AC_topGO.py
sleep 3
clear
echo "Running parse_sprot.py"
python parse_sprot.py
sleep 3
clear
echo "Running blast_sequences.py"
python blast_sequences.py
sleep 3
clear
echo "Running prepare_dataset.py"
python prepare_dataset.py
sleep 3
clear
#echo "Stopping the instance!"
#sleep 5
#ec2-stop-instances i-1a5755c1 --region us-west-2
```

Figure 12 Content of generateDataset.sh

Note that in the code block shown above, the last part has already been commented out which can be used in an AWS instance i-1a5755c1 located in Oregon specifically, in order to shutting down that instance automatically. Similarly, other AWS command line commands are also applicable depend on the customization towards this shell script file to automatize the process of the preparation of raw data.

Python script ‘gen_AC_GO.py’ is used to fetch information from ‘uniprot_sprot.dat’ described previously, like protein sequences’ AC numbers and their GO terms. As the output, ‘AC_GO.dat’ contains the result from the previous script in a file format acting like hash table, which can be loaded for later use, in other scripts, for

example. ‘AC_BIN.dat’ is also generated to store AC numbers of sequences and their binary unique ID number arranged automatically by the script, which can be used as features.

```
Q6GZX4  0006355,0046782,0006351,
Q6GZX3  0033644,0016021,
Q6GZX1  0033644,0016021,
Q6GZW6  0005524,0003677,0004386,
Q6GZW5  0033644,0016021,
Q6GZW4  0033644,0016021,
Q197E7  0033644,0016021,
Q6GZV6  0005524,0004674,
Q6GZV5  0033644,0016021,
Q197D8  0033644,0016021,
Q6GZU9  0016301,
Q6GZU8  0003677,0003899,0006351,
Q6GZU4  0006353,
Q6GZU2  0033644,0016021,
Q91G63  0016021,
Q197C3  0033644,0016021,
Q6GZT9  0016787,
Q6GZT3  0033644,0016021,
Q197B6  0005524,0004674,
Q91G50  0016021,0034257,
Q6GZS4  0003854,0006694,
Q6GZS3  0033644,0016021,
Q6GZS1  0005524,0003677,0004386,
O55703  0016021,
Q6GZR8  0016740,
O55704  0016021,
O55705  0016021,
Q6GZR1  0042981,
Q91G40  0016021,
```

Figure 13 Some of the entries in ‘AC_GO.dat’

```
Q6GZX4  00000000000000000000
Q6GZX3  00000000000000000001
Q197F8  00000000000000000010
Q197F7  00000000000000000011
Q6GZX2  00000000000000000100
Q6GZX1  00000000000000000101
Q197F5  00000000000000000110
Q6GZX0  00000000000000000111
Q91G88  00000000000000001000
Q6GZW9  00000000000000001001
Q6GZW8  00000000000000001010
Q197F3  00000000000000001011
Q197F2  00000000000000001100
Q6GZW6  00000000000000001101
Q91G85  00000000000000001110
Q6GZW5  00000000000000001111
Q197E9  00000000000000010000
Q6GZW4  00000000000000010001
Q6GZW3  00000000000000010010
Q197E7  00000000000000010011
Q6GZW2  00000000000000010100
Q6GZW1  00000000000000010101
Q6GZW0  00000000000000010110
Q6GZV8  00000000000000010111
Q6GZV7  00000000000000011000
Q6GZV6  00000000000000011001
Q6GZV5  00000000000000011010
Q6GZV4  00000000000000011011
Q197D8  00000000000000011100
Q6GZV2  00000000000000011101
Q197D7  00000000000000011110
Q6GZV1  00000000000000011111
Q197D5  00000000000000100000
Q91G70  00000000000000100001
```

Figure 14 Some of the entries in 'AC_BIN.dat'

'pickGO.py' is another python script tending to select a certain number of GO terms which own the property of most occurrence rate. That certain number can be adjusted accordingly in the script as a variable called 'num'. The default value is 1, in order to pick the most frequently appeared GO term to make the best example. The output is also written in a file called 'topGO.dat', with a key of the GO term(s), and a value of the number of its occurrence in the form of a hash table.

Python script 'gen_AC_topGO.py' is used to generate a hash table with keys of AC numbers, which is all the AC numbers acquired from 'AC_GO.dat', and values

indicating whether the specific GO term, which is selected in ‘topGO.dat’, is one of the feature to those AC numbers accordingly. Note that, there is a variable in the script named ‘num’, which should be the same value as the one in ‘pickGO.py’, indicating how many GO terms are identified as existing or not in the SWISS-PROT database for each specific AC number. As the output, ‘AC_topGO.dat’ is generated to store the result for later use.

Concerning the script ‘sel_AC_topGO.py’, we can select a certain number of entries from ‘AC_topGO.dat’ as the sample, in order to shorten the process of training and processing, giving the reasonable experimental result as the representative of the entire dataset. The sampling process is based on simple random sampling according to occurrence rate of the specific GO term. Additionally, user could change the rate accordingly to satisfy the very customized need. Also, the variable named ‘num’ should meet the requirement described previously to maintain the consistency, in case of the error or wrong results. In the case of this project, we would select 10000 entries of data to proceed to the preprocessing process later on. The output file is named ‘sel_AC_topGO.dat’, which is almost as the same format as ‘AC_topGO.dat’.

```
Q91X05  0
Q6LYX4  0
B1JBP2  0
Q5L575  1
A4YZQ6  0
B7N1T5  1
B5YZG7  1
A3CRK8  0
C1ETP1  0
Q6PBN5  0
Q86YW9  0
A8AEP3  0
P84613  0
Q5RAC9  0
Q70Y11  0
Q82JI1  0
Q3TX08  0
B2AGB7  0
P84771  0
C3LKW7  0
Q9RH13  0
Q9SG10  0
O04376  0
Q646E9  0
P05106  0
Q8VF65  0
Q8TCY9  1
Q9AMJ8  1
C4K2D9  1
```

Figure 15 Sample of Result of 'sel_AC_topGO.py'

When it comes to the python script called 'parse_sprot.py', the script is mainly designed to get the amino acid sequence from 'uniprot_sprot.fasta' based on AC numbers selected which are stored in 'sel_AC_topGO.dat', and store each AC number and its amino acid sequence into a file named after the AC number, ending with the bioinformatics sequence extension, in FASTA format. A directory is made at the present working directory, named 'sequence', with the purpose of storing those FASTA files at that directory. In order to simplify the process of sequence-extracting, SeqIO module of Biopython is utilized to parse 'uniprot_sprot.fasta' into a dictionary, which saved a lot of working in doing regular expression search like we did in previous scripts.

```
>sp|A0LLG2|ATPF_SYNFM ATP synthase subunit b OS=Syntrophobacter fumaroxidans (strain DSM 10017 / MPOB) GN=atpF
    PE=3 SV=1
MHAGVRGKKERAKFVWPLLGAGLLLAAEGVAWASGGGGEHGGGHLNWSDFLARTLVFVIT
FSILFKLLKKPIAGFFSSRKAEIQRLLSELELKQKEAEQNHAECKAKLAALEVETKKIVD
ELIAEGEVERQKIIEAAEKQADYLRQQADVAIQQEIKAAREKLKLEISELSVAAAEEILR
KNMKAKDQDRLVRDFMKRVVEAK
```

Figure 16 The Content in A0LLG2.fasta in 'sequence' Directory

In order to extract the features from analogous proteins, we would have to use alignment search tools to find those proteins, where we found PSI-BLAST decently useful on finishing this task. Thus, 'blast_sequences.py' is being implemented to do PSI-BLAST automatically on the sequences selected. It utilized the module called 'NcbipsiblastCommandline' from Biopython, in order to align the sequence, receiving multiple analogous sequences from the database selected, which can be used as the features for training. Please note that running this script may have to take a very long period of time, due to every time-consuming PSI-BLAST procedure for each sequence. The script would store every PSI-BLAST result into a log file in 'sequence' directory generated by shell script 'generateDataset.sh'. Thus, we would pick a moderately amount of dataset, to speed up the training and preprocessing process, and receiving a reasonable result to prove the effectiveness of the tool we developed. After the process of the script is accomplished, it would generate an output file called 'sel_AC_Blast.dat', which is made up by the AC numbers for the according sequences, as well as the 20 sets of binary AC number and according e-value as its features that is going to be used to train the deep learning network, where each set is consist of 20 binary digits to represent the AC number of the analogous sequence and 10 binary digits to represent

the e-value for that sequence, which would be 600 binary digits as the feature for one target sequence with an identical AC number. In addition, the script would record the elapsing time and the estimated time of finishing the task, which would help users identify how much it would take to finish this specific task. Here is the result comes out of the script.

Figure 17 The content in 'sel_AC_Blast.dat'

Finally, the script "prepare_dataset.py" is the last-step script which is needed to preprocess the raw dataset, getting the data ready as the input of the deep learning

framework in NumPy format, say, ‘train_features.npy’, ‘train_labels.npy’, ‘test_features.npy’, and ‘test_labels.npy’.

A.3 Wrapping up the Input Data for Pylearn2

Even though we have already transformed the input data into NumPy format, it is still not enough. We would need to store that data into a DenseDesignMatrix class format of Pylearn2, which is a type of format used to store simple dataset, where the data containing features would be represented as dense matrices.

```
import numpy as np
from pylearn2.datasets.dense_design_matrix import DenseDesignMatrix

def load_data(features, labels, start = 0, stop = None):
X = np.load(features)
y = np.load(labels)
X = X[start:stop, :]
y = y[start:stop, :]
return DenseDesignMatrix(X = X, y = y)
```

Figure 18 The Python Script of Data Wrapper

A.4 Training Process of the Deep Learning Framework

It is pretty simple to train the deep learning framework by simply using commands like:

$ python train_framework.py

After this step, four python pickle files would be generated to store models generated accordingly towards two autoencoder layers, and a multilayer perceptron layer. To be more specific, those models stored in pickle files are parameters trained after the process of training. For ‘framework_1.pkl’ and ‘framework_2.pkl’, they are models generated from those two autoencoder layers, while the following two models are generated from the multilayer perceptron layer, for ‘framework_top.pkl’ and

'framework_best.pkl'. Additionally, for the last two pickle files, the former one is the model generated after the final layer, while the later one is the final model with most optimized parameters.

For the script 'train_framework.py', it can be used to store training initiation parameters, including feature data path, label data path, numbers of visible and hidden nodes, etc., for each layer of the training framework. Also, it is also gifted the ability to call the training function to train the framework layer by layer.

```
from pylearn2.config import yaml_parse

layer1_yaml = open('Layer1.yaml', 'r').read()
hyper_params_l1 = { 'features' :'train_features.npy',
                    'labels' :'train_labels.npy',
                    'start' :0,
                    'stop' :8000,
                    'nvis' :600,
                    'nhid' :250,
                    'batch_size' :100,
                    'monitoring_batches' :5,
                    'max_epochs' :500,
                    'save_path' :'.'}
layer1_yaml = layer1_yaml % (hyper_params_l1)
print layer1_yaml
train = yaml_parse.load(layer1_yaml)
train.main_loop()
```

Figure 19 Part of the Script 'train_framework.py'

A.5 The YAML files

Pylearn2 uses YAML files, a type of format which can be easily read by human intuitively, to describe experiment configuration, including specifications of model, dataset and training algorithm. It is evident to tell many information about model, dataset and algorithms, even to write information towards YAML file. Pylearn2 would read in these files, and generate the model using the specified dataset and algorithm

accordingly, calling the corresponding configuration from python codes written in the library, and passing the parameters to those codes.

```
!obj:pylearn2.train.Train {
    dataset: &train !obj:dataset_wrapper.load_data {
        features: %(features)s,
        labels: %(labels)s,
        start: %(start)i,
        stop: %(stop)i
    },
    model: !obj:pylearn2.models.autoencoder.DenoisingAutoencoder {
        nvis : %(nvis)i,
        nhid : %(nhid)i,
        irange : 0.05,
        corruptor: !obj:pylearn2.corruption.BinomialCorruptor {
            corruption_level: .2,
        },
        act_enc: "tanh",
        act_dec: null,    # Linear activation on the decoder side.
    },
    algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
        learning_rate : 1e-3,
        batch_size : %(batch_size)i,
        monitoring_batches : %(monitoring_batches)i,
        monitoring_dataset : *train,
        cost : !obj:pylearn2.costs.autoencoder.MeanSquaredReconstructionError {},
        termination_criterion : !obj:pylearn2.termination_criteria.EpochCounter {
            max_epochs: %(max_epochs)i,
        },
    },
    save_path: "%(save_path)s/framework_1.pkl",
    save_freq: 1
}
```

Figure 20 Content in Layer1.yaml

A.6 Deep Learning Architecture

Developing deep learning architecture using those YAML files to define the deep learning architecture would be much more sufficient than hard coding, which could help experimenters focus more on the architecture itself including, but not limited to parameter setting, algorithm picking, and layer combination.

In our project, we would use multilayer perceptron, after two layers of stacked denoising autoencoders. By writing YAML files, the stacked denoising autoencoders can be built with two layers of denoising autoencoders, as well as the last layer of multilayer perceptron, also known as MLP. In this project, two denoising autoencoders

would be described in ‘Layer1.yaml’, and ‘Layer2.yaml’. On the other hand, MLP would be represented in ‘Layer3.yaml’. In the following figure, we would reveal part of the code in MLP layer instead of the other two denoising autoencoders, one of which has already been shown in the previous section.

```
!obj:pylearn2.train.Train {
    dataset: &train !obj:dataset_wrapper.load_data {
    features: %(features)s,
    labels: %(labels)s,
    start: %(start)i,
    stop: %(stop)i
},
model: !obj:pylearn2.models.mlp.MLP {
    batch_size: %(batch_size)i,
    layers: [
        !obj:pylearn2.models.mlp.PretrainedLayer {
            layer_name: 'h1',
            layer_content: !pkl: "%(save_path)s/framework_1.pkl"
            },
        !obj:pylearn2.models.mlp.PretrainedLayer {
            layer_name: 'h2',
            layer_content: !pkl: "%(save_path)s/framework_2.pkl"
        },
        !obj:pylearn2.models.mlp.Softmax {
            max_col_norm: 1.9365,
            layer_name: 'y',
            n_classes: %(n_class)i,
            irange: .005
        }
    ],
    nvis: %(nvis)i, # need fix here
},
algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
    learning_rate: .05,
    learning_rule: !obj:pylearn2.training_algorithms.learning_rule.Momentum {
    init_momentum: .5,
},
monitoring_dataset:
{
    'train' : *train,
    'test'  : !obj:dataset_wrapper.load_data {
        features: 'test_features.npy',
        labels: 'test_labels.npy',
        start: %(test_start)i,
        stop: %(test_stop)i
    },
},
```

Figure 21 Part of ‘Layer3.yaml’

Also, techniques like dropout, early stopping is also added into the YAML files in order to let Pylearn2 take account of them. In this case, in the YAML file of MLP, say ‘Layer3.yaml’.

```
cost: !obj:pylearn2.costs.mlp.dropout.Dropout {
input_include_probs: {'h1': .8, 'h2' : .8, 'y' : .8},
input_scales: {'h1' : 2.0, 'h2' : 2.0, 'y' : 2.0}
},
train_iteration_mode: even_shuffled_sequential,
monitor_iteration_mode: even_shuffled_sequential,
termination_criterion: !obj:pylearn2.termination_criteria.And {
criteria: [
            !obj:pylearn2.termination_criteria.MonitorBased {
            channel_name: "train_y_misclass",
            prop_decrease: 0.,
            N: 80
            },
            !obj:pylearn2.termination_criteria.EpochCounter {
            max_epochs: %(max_epochs)i
            }
            ]
},
update_callbacks: !obj:pylearn2.training_algorithms.sgd.ExponentialDecay {
decay_factor: 1.00004,
min_lr: .000001
}
},
extensions: [
             !obj:pylearn2.training_algorithms.learning_rule.MomentumAdjustor {
```

Figure 22 Techniques Applied in the YAML File

A.7 Making Predictions

In this part, we would introduce how we are going to deal with the prediction part. Actually, we are using ‘get_prediction.py’ to predict test features, getting the result that if one sequence has certain protein function. We would need three more arguments to run this script, such as model file, test feature file, and output file. Please note that the default parameters have already been set in the script. The default model file would be ‘framework_best.pkl’; the default test feature file would be ‘test_features.npy’; and the default output file would be ‘output.txt’. In order to run the script, it is recommended to use the default file names, thus you wouldn’t need to specify all the parameters, say, the file names. The default command and more specific command would be shown as follow:

$ python get_prediction.py

and

$ python get_prediction.py model.pkl features.npy output.txt

The output file would be one binary digit each line standing for the prediction if the it is in that specific class.

In addition, if you want to predict his own sequence file. Script file ‘make_features.py’ would take the input file ‘input.fasta’, say, the default input file name, and output the file ‘test_features.npy’. After this step, you need to run ‘get_prediction.py’ again to get the finally result in ‘output.txt’. The commands would be listed as follow:

$ python make_features.py

$ python get_predictions.py

A.8 Evaluation Scripts

We would use ‘calc_precision.py’ to evaluate the precision, accuracy and recall of the predicted model.

Also, we would use ‘pure_blast.py’ to get the accuracy using only PSI-BLAST to get the prediction, in order to compare with result that is generated by our model.

Bibliography

[1] Heizmann, C. W., Fritz, G., & Schafer, B. W. (2002). S100 proteins: structure, functions, and pathology. Front Biosci, 7(1), 1356-1368.

[2] Giepmans, B. N., Adams, S. R., Ellisman, M. H., & Tsien, R. Y. (2006). The fluorescent toolbox for assessing protein location and function. *Science*,*312*(5771), 217-224.

[3] Bork, P., & Koonin, E. V. (1998). Predicting functions from protein sequences—where are the bottlenecks?. Nature genetics, 18(4), 313-318.

[4] Benner, S. A., Chamberlin, S. G., Liberles, D. A., Govindarajan, S., & Knecht, L. (2000). Functional inferences from reconstructed evolutionary biology involving rectified databases–an evolutionarily grounded approach to functional genomics. Research in microbiology, 151(2), 97-106.

[5] Watson, J. D., Laskowski, R. A., & Thornton, J. M. (2005). Predicting protein function from sequence and structural data. Current opinion in structural biology, 15(3), 275-284.

[6] Marcotte, E. M., Pellegrini, M., Ng, H. L., Rice, D. W., Yeates, T. O., & Eisenberg, D. (1999). Detecting protein function and protein-protein interactions from genome sequences. Science, 285(5428), 751-753.

[7] Enright, A. J., Van Dongen, S., & Ouzounis, C. A. (2002). An efficient algorithm for large-scale detection of protein families. Nucleic acids research,30(7), 1575-1584.

[8] Wang, Z., Cao, R., & Cheng, J. (2013). Three-level prediction of protein function by combining profile-sequence search, profile-profile search, and domain co-

occurrence networks. BMC Bioinformatics, 14(3), S3-S3. doi:10.1186/1471-2105-14-S3-S3

[9] Thomas E. Creighton. (1993). Proteins: structures and molecular properties. Macmillan.

[10] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research,25(17), 3389-3402.

[11] Johnson, M., Zaretskaya, I., Raytselis, Y., Merezhuk, Y., McGinnis, S., & Madden, T. L. (2008). NCBI BLAST: a better web interface. Nucleic acids research, 36(suppl 2), W5-W9.

[12] Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice.Nucleic acids research, 22(22), 4673-4680.

[13] Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., ... & Bengio, Y. (2013). Pylearn2: a machine learning research library. arXiv preprint arXiv:1308.4214.

[14] Gene Ontology Consortium. (2004). The Gene Ontology (GO) database and informatics resource. Nucleic acids research, 32(suppl 1), D258-D261.

[15] Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M. C., Estreicher, A., Gasteiger, E., ... & Pilbout, S. (2003). The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. Nucleic acids research,31(1), 365-370.

[16] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[17] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

[18] Ruck, D. W., Rogers, S. K., Kabrisky, M., Oxley, M. E., & Suter, B. W. (1990). The multilayer perceptron as an approximation to a Bayes optimal discriminant function. IEEE Transactions on Neural Networks, 1(4), 296-298.

[19] Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1), 1-127.

[20] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec), 3371-3408.

[21] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.Journal of Machine Learning Research, 15(1), 1929-1958.

[22] Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. Neural computation, 7(2), 219-269.