1.1

UNIVERSITY OF NAIROBI

TEXT CLASSIFICATION AND ANALYSIS USING MACHINE LEARNING

NAME: ODHIAMBO ROWLAND OTIENO
REG NO. F17/1601/2012

SUPERVISOR: MR. OSCAR ONDENG
EXAMINER: XXX

Project report submitted in partial fulfillment of the requirement for the award of the degree of Bachelor of Science in Electrical and Electronic Engineering at the University of Nairobi

Date of Submission: XX/XX/2017

Department of Electrical and Information Engineering

DECLARATION OF ORIGINALITY FORM

This form must be completed and signed for all works submitted to the University for Examination.

Name of Student: Odhiambo Roland Otieno
Registration No: F17/1601/2012
College: College Architecture and Engineering
Faculty/School: School of Engineering
Department: Electrical and Information Engineering
Course Name: Bachelor of Science in Electrical & Electronic Engineering
Title of the Work: Text Classification and Analysis Using Machine Learning

DECLARATION

1. I understand what Plagiarism is and I am aware of the University's policy in this regard.
2. I declare that this final year report is my original work and has not been submitted elsewhere for examination, award of a degree or publication. Where other people's work or my own work has been used, this has properly been acknowledged and referenced in accordance with the University of Nairobi's requirements.
3. I have not sought or used the services of any professional agencies to produce this work.
4. I have not allowed, and shall not allow anyone to copy my work with the intention of passing it off as his/her own work.
5. I understand that any false claim in respect of this work shall result in disciplinary action, in accordance with University Plagiarism Policy

Signature: ……………
Date: ………………….

DEDICATION

For Mum, in memoriam.

ACKNOWLEDGEMENT

I would like to express my gratitude and appreciation to my supervisor, Mr. Oscar Ondeng who encouraged me and dedicated his time in guiding me to achieve the project’s objectives.

Special thanks to Siraj Arawal, for his assistance in understanding the various technicalities of the underlying algorithms that I used. I would also like to extend my thanks to my friends and classmates for their opinions, useful critiques of this work and offering to proofread this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1.Background

Text classification is the task of finding a given class or category for a given set of texts. It falls under the category of Natural Language Processing (NLP) which is a vast area that is concerned with the interaction between computers and human language. Generally, in classification, we try to generate a classification function using a trainable set of coefficients W which can give the correlation between a certain 'feature' X and a class Y. This classifier model first, must be trained with a training dataset, and thereafter it can be used to classify new or unseen data. During the training, we must find the model coefficients which minimize the error the most. If the set of training examples is chosen correctly, the classifier model should predict the class probabilities of the actual documents with a similar accuracy (as the training examples).

After construction, such a model classifier could for example tell us that a document containing the words "Wayne Rooney scored the winning goal in the 88th minute". should be categorized as a sports article, while documents containing the words "inflation", "investment" or "funding" should be categorized as a Finance article. A second classifier could tell us that mails starting with "Dear Customer/Guest/Sir" (instead of your name) and containing words like "Great opportunity", "discount" or "one-time offer" can be classified as spam. A third usage of text classification is Sentiment Analysis. Here the purpose is to determine the subjective value of a text-document, i.e. how positive or negative is the content of a text document.

From the three examples, we can see three uses of text classification models: *topic classification*, *spam filtering* and *sentiment analysis* for which machine learning classifiers work quite well and perform better than most trained professionals.

1.2.Problem Description

Humans can clearly recognize the message/sentiment from a text with very little effort. However, bulk text classification task, in the dynamic world where the amount of textual information is enormous and the context of classification is always varying, is monotonous and tiresome even for a trained professional. How to manually sort or do analysis huge amounts of data is a fundamental difficulty in many companies, organizations and governments that deal with huge inflow of textual data daily. These come in the form of emails, SMSs, product reviews, applications forms (e.g. loans, university admissions forms) etc.

These challenges also make traditional text classification methodologies and algorithms very slow and unreliable. Traditionally you would sequentially program a text classification algorithm specific to a context. This involves writing explicit, fine-detailed specialized instructions on how to detect and recognize features intrinsic to the specific problem say in spam classification one would create a function to detect spam words. The same algorithm could not be used in sentiment analysis, since you had to create a separate function to detect words that express sentiment.

Machine learning on the other hand is able to solve these difficulties by finding effective algorithms on its own.

1.3.Project Justification

While a single text classification is not very difficult for human brains, bulk classification can be tedious and it can take a huge amount of time before we can gain insights from analyzing huge amounts of data. Machines by nature do not have the natural intuition to be able to distinguish, classify or recognize texts, but we can be able to create an artificial intuition into the machine by training it on how to dynamically learn salient features in text and then be able to recognize what kind of class to classify it. This project seeks to solve text classification using machine learning methodologies.

1.4.Project Objectives

To design, build and test a machine learning algorithm to classify texts.

- To implement the machine learning algorithm in a system that can be used to detect, analyze, recognize text features and classify the texts based on a predefined set of classes or categories.
- To implement a machine learning algorithm that will analyze text sentiment
- To implement a machine learning algorithm that will filter spam
- To deploy the machine learning system on a live environment and demonstrate its use.

2. LITERATURE

2.1. Machine Learning

The term machine learning refers to the automated detection of meaningful patterns and regularities in data. In the past couple of decades, it has become a common tool in almost any task that requires information extraction from large data sets. More formally, Machine learning is described by Tom Mitchell as:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

[11] Mitchell, T, "Machine Learning", ISBN 0-07-042807-7, p.2 ,McGraw Hill, 1997.

Example: playing sudoku.

- E = the experience of playing many games of sudoku
- T = the task of playing sudoku.
- P = the probability that the program will win the next sudoku game.

We are surrounded by a machine learning based technology: search engines and recommender systems learn how to bring us the best results (while placing relevant, targeted and profitable ads), anti-spam software learns to filter our email messages, and credit card transactions are secured by a software that learns how to detect frauds. Digital cameras learn to detect faces and intelligent personal assistance applications on smart-phones learn to recognize voice commands. Cars are equipped with accident prevention systems that are built using machine learning algorithms and at the same time being self-driven. Machine learning is also widely used in scientific applications such as bio-informatics, medicine, and astronomy.

One common feature of these applications is that, in contrast to more traditional uses of computers (see Fig 2.0), in these cases, due to the complexity of the patterns that need to be detected, a human programmer cannot provide an explicit, fine detailed specification (program) of how such tasks should be executed. Taking example from intelligent beings, many of our skills are acquired or refined through learning from our experience (rather than following explicit instructions given to us). Machine learning tools are concerned with endowing programs with the ability to "learn" and adapt.

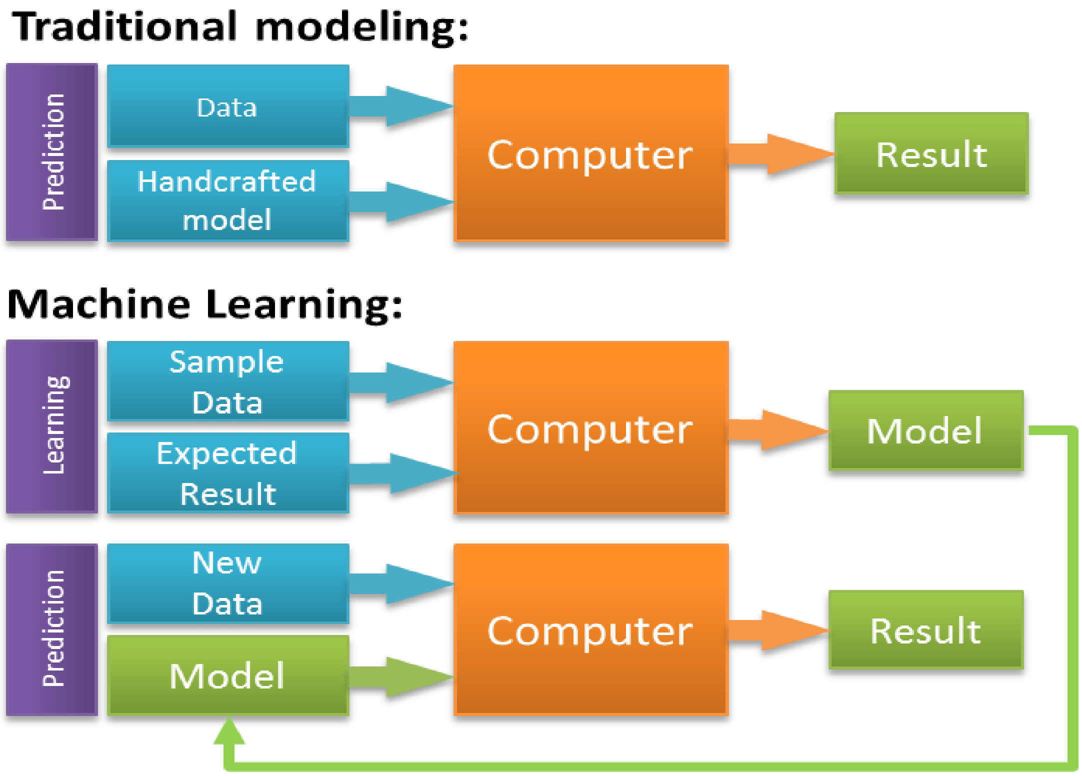


Fig 2.0: Traditional Modelling vs Machine Learning

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. See Fig 2.1 below.

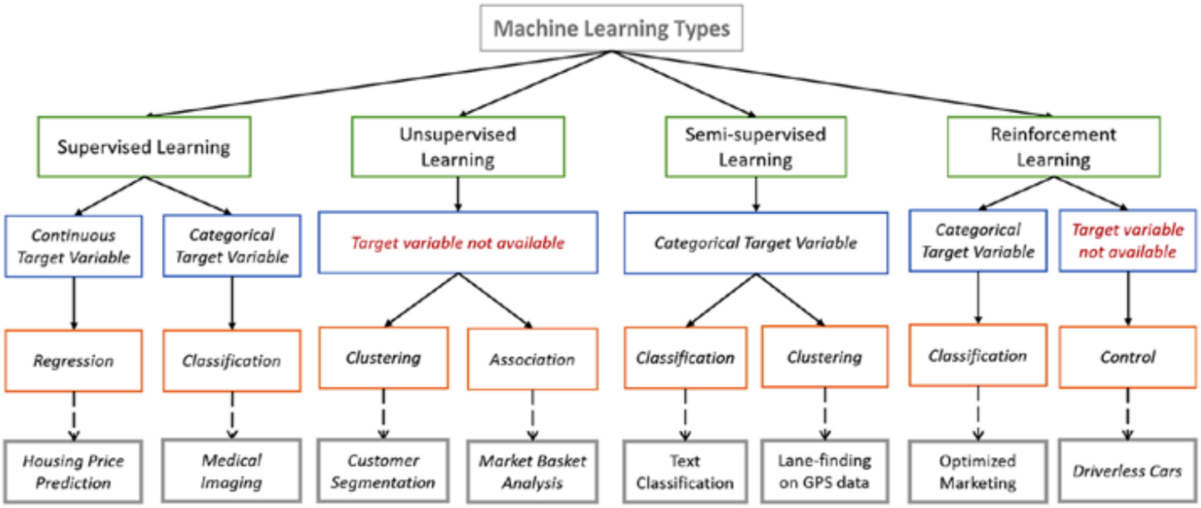


Fig 2.1: Machine Learning types

These are

- ***Supervised learning***: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.

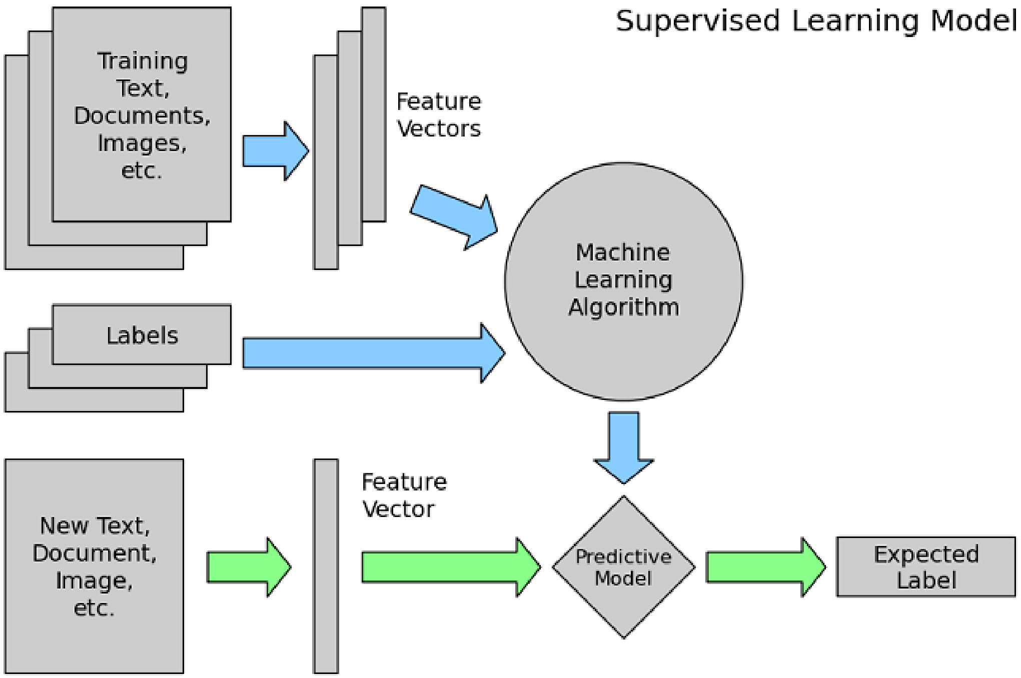

Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Fig 2.2: Supervised Machine Learning

In Fig 2.2 above, we can see that we feed both the input, and the labels to the machine learning model.

Through this technique we can be able to derive functions that map the inputs to the expected output, from which we can be able to predict a unique output from any unique input. Commonly used methods of supervised classification include: -

- ➔ Naive Bayes
- ➔ Decision Trees
- ➔ K-Nearest Neighbors
- ➔ Neural Networks
- ➔ Support Vector Machines

- ***Unsupervised learning***: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal (discovering hidden patterns in data) or a means towards an end (feature learning). It can be used to represent input patterns in a way that reflects the hidden statistical structure of the overall collection of input patterns. In this way it can be able to learn how to best cluster, rather

Than correctly labelling the unlabeled data. Commonly used methods of unsupervised classification include: -

- ➔ K-Means Clustering
- ➔ Method of Maximum Likelihood
- ➔ Anomaly Detection
- ➔ Neural Networks (Generative Adversarial Networks)
- ➔ Bayesian Networks

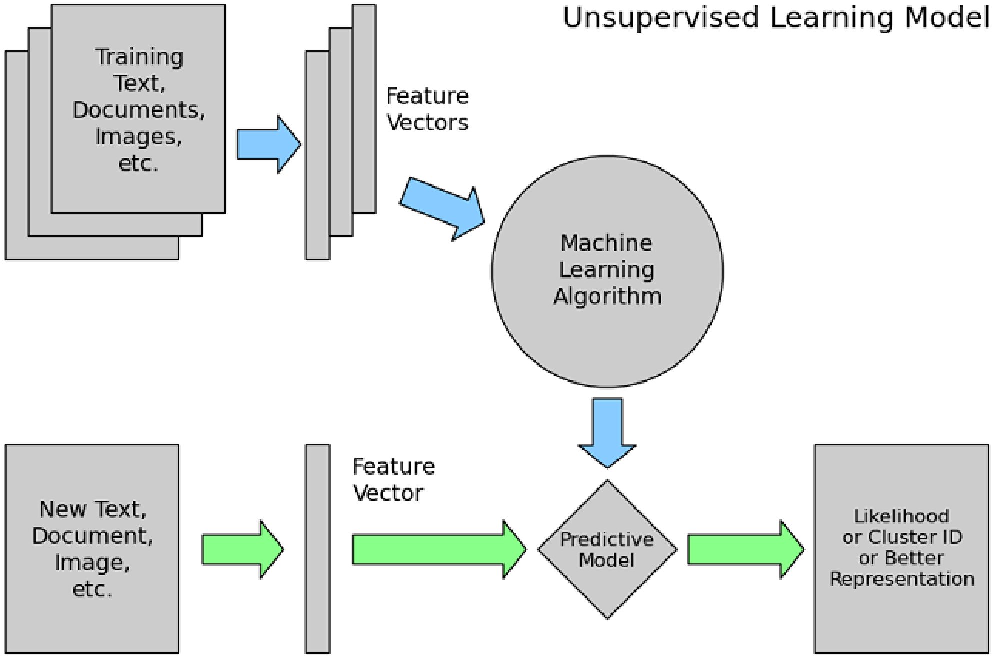

Unsupervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Likelihood or Cluster ID or Better Representation

Fig 2.3: Unsupervised Machine Learning

In Fig 2.3 above, we can see that we feed only the input, to the machine learning model. Unlike the supervised learning, the labels are not provided in this case.

In unsupervised learning, the goal is harder because there are no pre-determined categorizations or labels

- ***Reinforcement learning***: A computer program (agent) interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent. The program is provided feedback in terms of rewards and punishments as it navigates its problem space. It is often used to help computers learn how to play games, using a current state and *t*arget function. For example, if you want to teach a computer how to play sudoku, it will need to know the current state of the board, and identify the move with the greatest chance to win the game given the current state, called the target function. Since the computer doesn't know anything (since its knowledge table is empty) about playing chess, it will start by randomly selecting each move, and playing the chess game to completion. When the game ends and the computer wins or loses, that grade (positive for a win, negative for a loss) will be propagated back to each move in that game (it will have learnt the co-relation of state, action and value). When the computer plays another game, it will then have some information to feed to its target function that decides what move to choose. Over time, the computer will get better and better at playing chess (since it has filled up its knowledge table), all based on the rewards and punishments allocated to specific moves given a win or a loss.

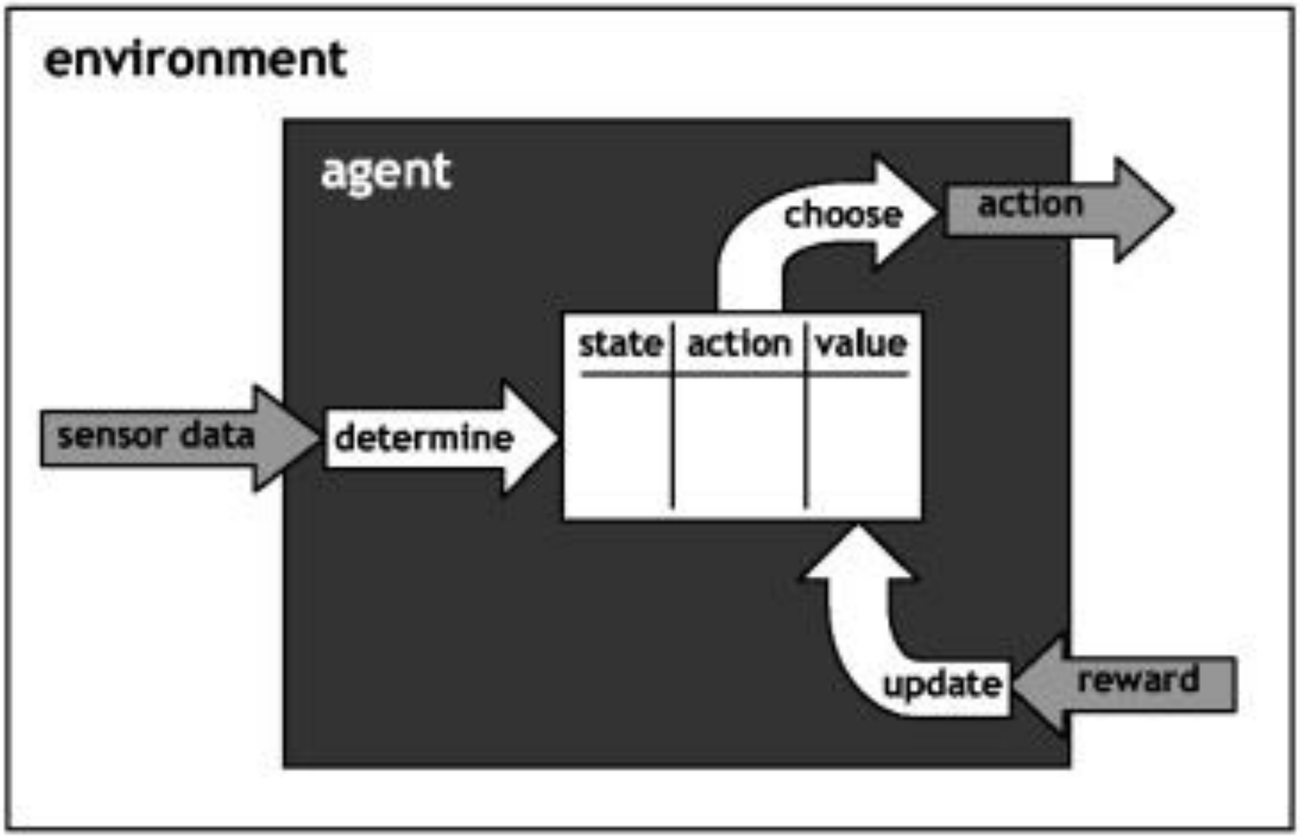


Fig 2.4: Reinforcement Learning Machine Learning

As you can see in Fig 2.4 above the knowledge table is empty at the start of the reinforcement training, but as the agent learns the correlation of state, action and value with experience, it will always know the next based action to take given some state.

This technique is also used in robotics for example teaching robots how to perform specific tasks such as to walking or traversing a particular terrain. It is also used in self-driving vehicles and autonomous drones.

2.2. Text Classification

Text Classification is the task of assigning one or more classes or categories to a piece of text or document according to their content. Classes are selected from a previously established taxonomy (a hierarchy of categories or classes). The piece of text could be a document, news article, search query, email, tweet, support tickets, customer feedback, user product review etc. Applications of text classification include categorizing newspaper articles and contents into categories, organizing web pages into hierarchical categories, **filtering spam email**, predicting user intent from search queries, routing support tickets, **sentiment analysis** and analyzing customer feedback.

It is a sensitive classification that may be affected by many factors during the data collection hence the process should take care of all preprocessing tasks required for automatic classification once the training data I collected. Designing a suitable text processing procedure is a prerequisite for a successful classification of the data. Non-parametric classifiers such as **neural network**, decision tree classifier, and knowledge-based classification have increasingly become important approaches for multisource text classification. Effective use of multiple features of randomly and independently sensed data and the selection of a suitable classification method are especially significant for improving classification accuracy. Therefore the:

- Collection of the data to be examined or used for training
- Selection of a classification system and training samples
- Data pre-processing tasks (extracting text, tokenization, stop word removal and lemmatization)
- Feature extraction and selection

Are very crucial steps. Beyond this point, it is important to examine the:

- Selection of a suitable classification algorithm e.g. maximum likelihood, Neural Networks, decision tree classifier, Naive-Bayes, minimum distance etc.
- Evaluation of classification performance using cross-validation and testing accuracy

2.2.1. Sentiment Analysis and Classification

Sentiment analysis refers to the use of natural language processing (NLP) techniques, text analysis and computational linguistics to systematically identify, extract, quantify, and study affective states such as user emotion and subjective information such as opinion. Sentiment analysis is widely applied to classify client materials such as product reviews, survey responses, online and social media posts. A basic task in sentiment analysis is classifying the polarity of a given text i.e. whether the expressed opinion in a text, is positive, negative, or neutral.

In advanced cases, beyond polarity, sentiment classification looks, for instance, at emotional states such as "angry", "sad", and "happy". Sentiment analysis finds great use various applications such as recommender systems where items can be recommended to other users based on the sentiment of others.

The accuracy of a sentiment analysis system is, in principle, how well it agrees with human judgments. This is usually measured by **Accuracy**, **F-Score, MCC-Score** which are discussed in section 2.5.

2.2.2. Spam Filtering

Spam filtering is the process of distinguishing spam from non-spam messages. A good percentage of the messages coming to my personal mailbox is spam. Automatic mail filtering seems to be the most effective method for countering spam now and a tight competition between spammers and spam-filtering methods is going on. The finer the anti-spam methods get, so do the tricks of the spammers. Only several years ago most of the spam could be reliably dealt with by blocking e-mails coming from certain addresses or filtering out messages with certain subject lines.

There are two general approaches to mail filtering: knowledge engineering (KE) and machine learning (ML). In the former case, a set of rules is created according to which messages are categorized as spam or legitimate mail. A typical rule of this kind could look like:

```
if the subject of a message has common spam words:
    then the message is spam
else:
    then the message is not spam
```

A set of such rules should be created either by the user of the filter, or by some other authority (e.g. the software company that provides a rule-based spam-filtering tool). The major drawback of this method is that the set of rules must be constantly updated, and maintaining it is not convenient for most users. Therefore, it is better when spam filtering can be able to learn and adjust to trends. Since machine learning does not require explicit specification of such rules in code, a machine learning model can learn the classification rules from a pre-classified class of messages and on its own develop an algorithm**.**

2.3. Classifiers

There are many machine learning classifiers one could use for text classification. In no particular order they include: Neural networks (NN), Support vector machines (SVM), Naive Bayes (NB), Decision trees (DT) and Random Forest (RF).

No one algorithm is always better than others. As stated by Wolpert and Macready, "Any two algorithms are equivalent when their performance is averaged across all possible problems… We show that all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A. Starting from this we analyze a number of the other a priori characteristics of the search problem, like its geometry and its information-theoretic aspects. This analysis allows us to derive mathematical benchmarks for assessing a particular search algorithm's performance…"

[11] Wolpert, D.H., Macready, W.G., "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67, 1997.

For a given application, the "best" one is generally one that is most closely aligned to the application in terms of the assumptions it makes, the kinds of data it can handle, the hypotheses it can represent, and so on.

It is a good idea to characterize the data according to criteria such as:

- Do I have a very large data set or a modest one?
- Is the feature dimensionality high?
- Are features numerical continuous, discrete or a mix, and/or can they be transformed if necessary?
- Is problem linearly separable?
- Are there likely to be redundant, noisy, or irrelevant features?
- Do I want to be able to inspect the model generated and try to make sense of it?

By answering these, you can eliminate some algorithms and identify others as potentially relevant, and then maybe end up with a small set of candidate methods that you have intelligently chosen as likely to be useful. In answering these five questions, I find that:

- Organizations will have large datasets due to the huge amount of information they collect. In this project, I’ll be using a large dataset of about 300,000 rows of texts for the text classification problem of sentiment analysis.
- The feature dimensionality is high since it is based on the total no. of unique words in dataset.
- Features are textual, but convertible to numerical form and from one form to the other.
- Text classification is largely a linearly separable problem as shown in Fig 2.5.

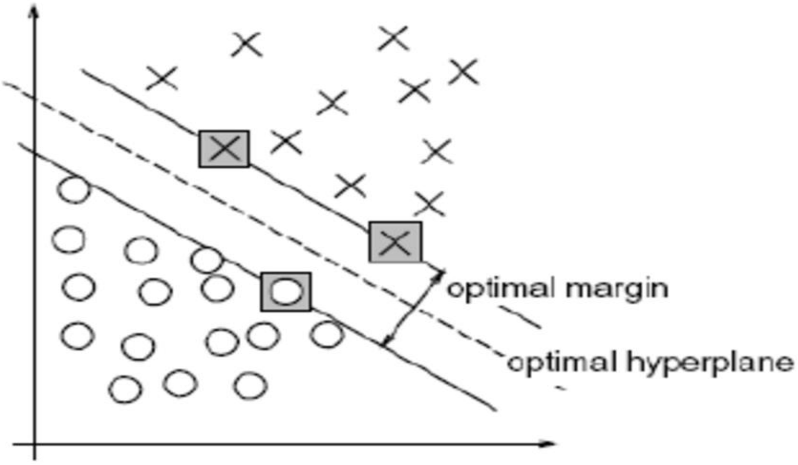


Fig 2.5: Linear separability of two classes by an optimum hyperplane

- Text, given that not all words have sentiment or connection to being spam, is definitely a noisy problem, especially given that a majority of words that make up a textual dataset are not nouns, verbs or adjectives but conjunctions and punctuation which occur in almost every unique sentence with high frequency.
- Yes, it would be important to be able to inspect the model, and check out how it generalizes both during and after training.

An artificial neural network (as discussed in section 2.3.1), answers all of the above questions and is a suitable consideration for text classification, especially when there are huge amounts of data available. Naive-Bayes, works a little differently from Neural Networks since it is probabilistic based and is a worthy control experiment. This project will only focus on the two algorithms, Naive-Bayes and Neural Networks, for both sentimental analysis and spam filtering, the results of which will be compared at the end.

2.3.1. Artificial Neural Networks (ANN)

Neural networks are a biologically-inspired programming paradigm which enables a computer to learn from observational data. The simplest definition of a neural network, is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.

[10] Maureen Caudill, “Neural Network Primer: Part I”, AI Expert, Part I, Feb. 1989.

They are processing elements (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the mammalian cerebral cortex but on much smaller scales. A large ANN might have hundreds or thousands of processor units, whereas a mammalian brain has billions of neurons (Fig 2.5) with a corresponding increase in magnitude of their overall interaction and emergent behavior. One of the promises of neural networks is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction. Research in this area attempts to make better representations and create models to learn these representations from large-scale unlabeled data. Some of the representations are inspired by advances in neuroscience and are loosely based on interpretation of information processing and communication patterns in a nervous system, such as neural coding which attempts to define a relationship between various stimuli and associated neuronal responses in the brain.

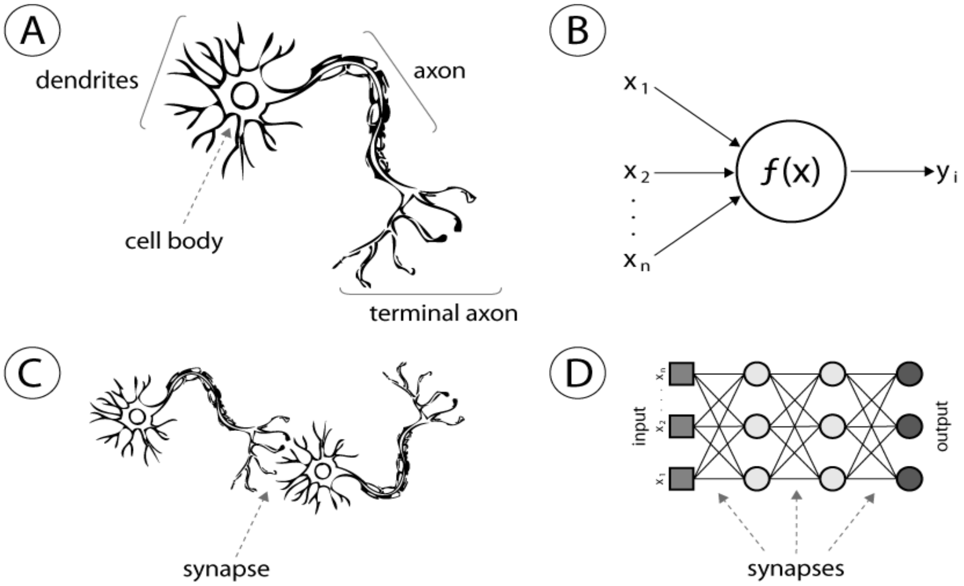


Fig 2.5: Cerebral Neural Nerves Unit vs Artificial Neural Network Unit

ANN's and deep learning currently provide the best solutions to many problems in pattern recognition. Its major advantage is its ability to identify patterns and regularities in given data and generalize well in predicting output when given new unlabeled input. With large sets of data available, neural networks can be able to solve many forms of pattern recognition tasks.

Text classification is one such task. Basically, a machine model is given a set of labelled input data i.e. text (e.g. a set of spam and non-spam emails or SMS), and by processing these texts, it can learn the salient features of either. The process is a supervised set of trainings where an estimated prediction of the text's class is repeatedly compared with the expected output to get a mean square error (MSE) margin. In each iteration of the training the machine model works towards reducing this margin of error until it gets to the lowest attainable error margin. At that point, we say the machine has been trained well, to some accuracy level of which we are confident that when given a test set it can be able to accurately recognize and classify the text correctly. The same can be applied to a input data such as speech (a set of female and male voices), images (e.g a set of dogs and another set of cats). In general, At a very high level, a neural network is a machine learning model that takes an input vector, applies a bunch of complicated functions to it ("neurons"), compares the output to a target result, and then tweaks the functions to get closer to this target ("back propagation").

a. Areas of Application

The neural network and its intrinsic pattern recognition behavior has various areas of application such:

- ➔ Handwritten digit classification
- ➔ Number plate recognition
- ➔ Recognition of fiscal coins of different dimension, shape or color.
- ➔ Fraud detection in financial transactions
- ➔ Signature verification using parallel neural networks
- ➔ Natural language processing
- ➔ Sign language recognition
- ➔ Face detection
- ➔ Traffic sign recognition
- ➔ Path planning for autonomous drones
- ➔ Computer vision for self-driving vehicles (Lane-finding)
- ➔ Prediction of trajectory re-entry of spacecraft

Depending on the nature of the application and the strength of the internal data patterns you can generally expect a ANN to train quite well. This applies to problems where the relationships may be quite dynamic or nonlinear. They provide an analytical alternative to conventional techniques which are often limited by strict assumptions of normality, linearity, variable independence etc. Because an ANN can capture many kinds of relationships it allows the user to quickly and relatively easily model phenomena which otherwise may have been very difficult or impossible to explain otherwise.

2.3.2. Naive-Bayes

Naive Bayes classifiers approach the classification task from a statistical point of view. The starting point is that the probability of a class vector **C** is given by the posterior probability **P(C|D)** given a feature vector **D**. Here **D** refers to a vector of all the words in the entire training set. It is given by **D** $(\mathbf{d_1\, d_2\, d_3\, d_4 \ldots d_n})$, where $\mathbf{d_n}$ is the $\mathbf{n_{th}}$ word of feature vector D. C refers to the output vector and is given by **C** $(\mathbf{c_1, c_2})$ where $\mathbf{c_1}$ and $\mathbf{c_2}$ are classes/categories.

P(D|C) denotes the probability of obtaining a text with feature vector D from class C**.** Usually we know something about these distributions (e.g. we would know that the probability of receiving a text containing the word **d_n =** ***discount*** from the category **1** is zero. For text classification, this probability can simply be calculated by calculating the frequency of word **d_n** in class C=c_i relative to the total number of words in class C=c_i.

What we want to know is, given a text D, what class C "produced" it? That is, we want to know the probability P(C|D). And this is exactly what we get if we use the Bayes' rule:

$$P(C|D) = \frac{P(D|C)*P(C))}{P(D))}$$

Eq 2.0: Bayes Rule

- *P*(*C*|*D*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*C*) is the prior probability of *class*.
- *P*(*D*|*C*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*D*) is the prior probability of *predictor*.

Since P(D) is the same for all classes, hence is a common denominator, we can simplify the Bayes equation to:

$$P(C|D) = P(D|C) * P(C))$$

Eq 2.1: Non-standardised Bayes Rule

We need to multiply the class probability with all the prior-probabilities of the individual words belonging to that class. The question then is, how do we know what the prior-probabilities of the words are? For a supervised machine learning algorithm, we can estimate the prior-probabilities with a training set with texts that are already labeled with their classes. With this training set we can train the model and obtain values for the prior probabilities. This trained model can then be used for classifying unlabeled texts.

Hence if **P(C=0|D) > P(C=1|D), t**hen classify the text as class **C=0,** otherwise classify it as class **C=1.**

2.4. Algorithms

2.4.1. Deep Learning Using Neural Networks

Deep learning (Fig 2.6) is part of a broader family of machine learning methods based on stacking several neural networks on top of each other. An observation (e.g., an image) can be represented in many ways such as a vector of intensity values per pixel, or in a more abstract way as a set of edges, regions of particular shape, etc. Some representations are better than others at simplifying the learning task (e.g. face recognition or facial expression recognition in Fig 2.2).

As another example, text, may be represented in terms of a vector of the word frequency count or the binary absence/absence (0/1) of a word in text.

Deep Learning is the algorithm of choice for most classification recognition problems, where large datasets exist or complex features form our data. It is a multilayer neural network learning algorithm which emerged in recent years. It has brought a new wave to machine learning, and making artificial intelligence and human-computer interaction advance with big strides. After training, when an arbitrary input pattern is present which contains noise or is incomplete, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training.

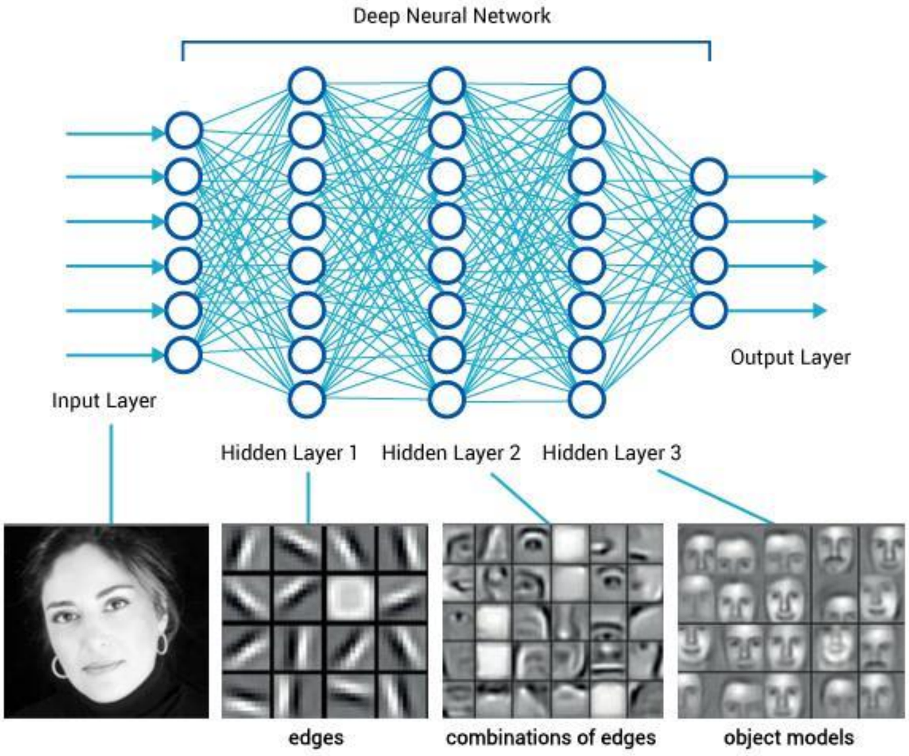


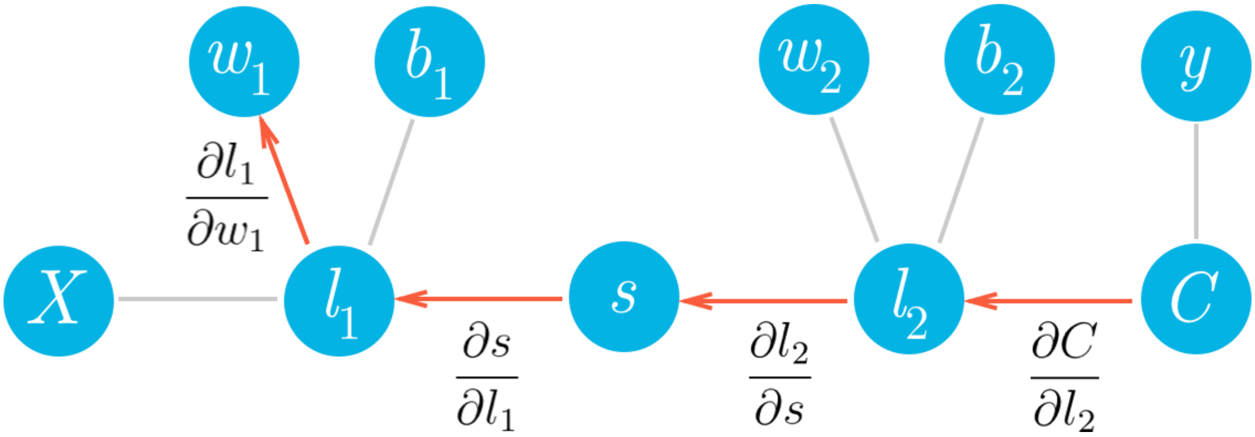
Fig 2.6: Image classification using deep neural networks

Deep Learning is applied to handwritten character recognition, spam classification, stock prediction, image recognition and many other areas.

2.4.2. Forward and Back Propagation

The backward propagation of errors or backpropagation, is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two-phase cycle, propagation and weight update. When an input vector is fed into the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The backpropagation algorithm looks for the minimum of the error function in weight space using the method of **gradient descent**. In gradient descent, the output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output. It is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer.

The combination of weights which minimizes the error function is considered to be a solution of the learning problem. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function. For this reason, popular activation functions that are in use include the tangent hyperbolic function, sigmoid function since they are continuous and differentiable at all points.



w_1
b_1
w_2
b_2
y
$\frac{\partial l_1}{\partial w_1}$
X
l_1
s
l_2
C
$\frac{\partial s}{\partial l_1}$
$\frac{\partial l_2}{\partial s}$
$\frac{\partial C}{\partial l_2}$

Fig 2.7: Back-propagation intuition, C-predicted output, y-expected output, w-weights, b-bias, l-layer outputs, s-sigmoid of layer outputs

Backpropagation (Fig 2.7), uses these error values to calculate the gradient of the loss function with respect to the weights in the network. In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, in an attempt to minimize the loss function. The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves (update weights ($\mathbf{w}_i$) and bias($\mathbf{b}_i$) values) in such a way that the different neurons learn to recognize different features of the input.

2.4.3. Gradient Descent

Gradient Descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient until we reach some global minima (As seen in Fig 2.8). In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to coefficients in linear regression such as weights and bias in neural networks.

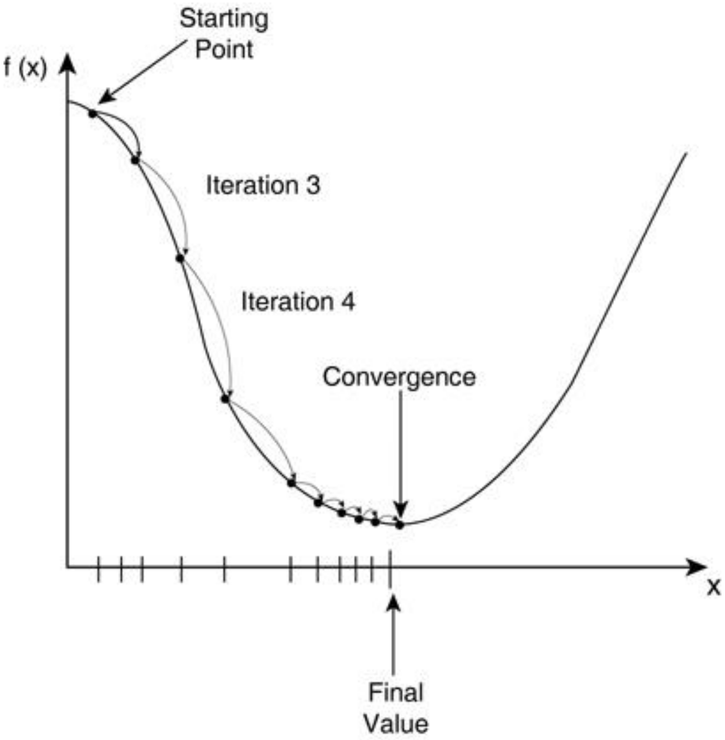


Fig 2.8: Gradient descent

The changing of the weights and biases in order to come up with values that minimize the error of the prediction occurs during the backpropagation with the help of the gradient descent algorithm. Essentially, this is what enables a network to learn.

2.4.4. Early Stopping

It is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Overfitting (shown in Fig 2.11) occurs when a model fits the data in the training set well, while incurring larger generalization error. Gradient descent updates the model so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on unseen (validation) data outside of the training set. Past that point, however, improving the model's fit to the training data comes at the expense of increased generalization error. Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit.

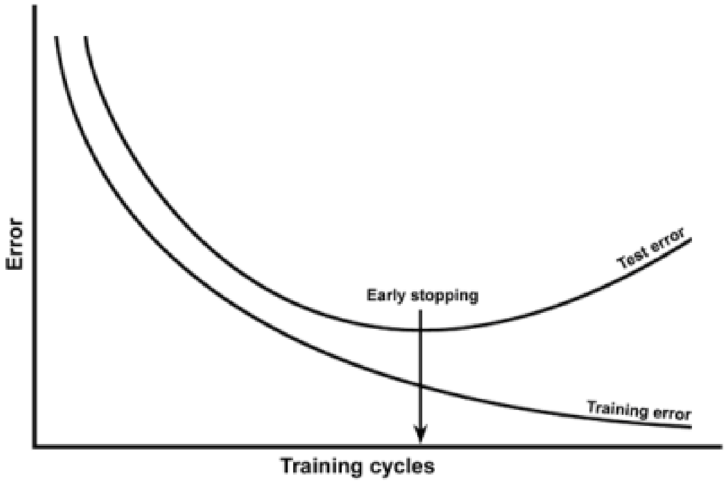


Fig 2.9: Early stopping technique

2.4.5. Hyper Parameter Selection and Optimization

The neural network hyperparameters can be broadly classified into two categories:

- ➔ Optimizer Parameters - these are parameters which can be used to tweak the training process in terms of speed of computation, amount of resources required for computation. It includes:
  - Learning rate
  - Number of iterations. /epochs
  - Batch size
- ➔ Model Parameters - these define the structure of the neural network and includes:
  - Network architecture e.g. number of layers, number of hidden layer units, number of output units, type of network i.e. perceptron, CNN, RNN etc.
- ➔ Trainable Parameters - these are the model specifications which the model attempts to train over time by updating their values in a way such that the loss function **Q(x_i,w_i,b_i)** is minimized. They are repeatedly updated during the training process in each iteration using back-propagation. They include the:
  - Weights and bias which are a measure of the sensitivity between the network's input and the prediction.

NB: **Q(x_i,w_i,b_i)** is the mean square error (MSE) of the predicted and expected output.

a. Optimizer parameters

i. Learning Rate

The learning rate is one of the most important parameters in the neural network and is often defined in the context of optimization and minimization of the loss function. Neural networks are often trained by gradient descent on the trainable parameters (weights and biases). This means that at each iteration we use gradient descent and backpropagation to calculate the derivative of the loss function **Q(x_i, wi,b_i)** with respect to each trainable parameters then subtract it from the respective trainable parameter. However, if you actually try that, the trainable parameter will change far too much in each iteration, which will make them "overcorrect" and the loss will actually increase/diverge. So, in practice, people usually multiply each derivative by a small value called the "l**earning rate**" before they subtract it from its corresponding weight.

Therefore, the learning rate determines how quickly or how slowly you want to update the parameters and nudge them towards the optimum values. Small learning rate takes a long time to converge at the global minima point since the training error reduces too slowly. Very large learning rate will prevent convergence. A cue of the effect of learning rate can be gotten from Table 2.0 The best learning rate depends on the problem at hand, as well as on the architecture of the model being optimized. Since small learning rates take a long time to converge, for any problem it is advisable to start from a large learning rate and gradually decrease the learning rate as the training progresses until an optimum value is figured out.

Validation Error	**Learning Rate Verdict**	**Action to Take on Learning Rate**
Decreasing during training	Good	-
Decreasing too slowly	Bad	Increase the learning rate
Increasing during training	Bad	Decrease the learning rate

Table 2.0: Effect of the Choice of the Learning Rate on Validation Error

ii. Number of Iterations/Training Cycles/ Epochs

It defines how many times we would like to go over the training samples. We can use many iterations (epochs) as long as the training error keeps reducing. Ideally we can use a fixed number of epochs but that doesn't often scale well for increasing amounts of training samples later on or when other sensitive hyperparameters are also changed. Luckily, we can use a technique called early stopping.

This works by monitoring our validation error (test error during training) and stopping the training when error starts to increase and has been almost constant as shown in Fig 2.9 and discussed in section 2.4.4.

iii. Mini-Batch Size

The mini-batch size defines the number of training samples we use in each iteration. For large datasets, it would save time when we train over only a randomly picked sample from our total training data population. A mini-batch size equal to the total number of samples implies that training is done over all the training data in each iteration. Computing the gradient of a batch generally involves computing the loss function **Q(x_i,w_i,b_i)** over each training example in the batch and summing over the functions. In particular, gradient computation is roughly linear in the batch size. Therefore, for a dataset of 100,000 samples, we would loop over 100,000-mini-batch samples in each iteration (epoch). This is computationally expensive since the amount of work done (in terms of number of gradient computations) per iteration is very high. If we are considering a mini-batch size of 100 and 10,000. So, it's going to take about 100x longer to compute the gradient of a 10,000-mini-batch than a 100-mini-batch.

Choice of the mini-batch size value is therefore very important and has an effect on the:

- ➔ Resources (memory) required for the training process
- ➔ Speed required for the training

Since this data has to be stored in memory for random access during computation, the mini-batch size is limited by the size and power of the available resources since exceeding the memory size will cause out of memory errors (overflow).

Recommended values for the mini-batch depend on the available depend on the availability of resources i.e. RAM memory, Graphics card and other computer resources. Typical values are [**1,2,4,8,16,32,64,128,256,1024**], all in powers of 2.

With a constant learning rate, a very large mini-batch size decreases the accuracy beyond a certain point. It has been observed in practice that when using a larger mini-batch size there is a significant degradation in the quality of the model, as measured by its ability to generalize. This is because we need to change the learning rate, if the batch size changes.

b. Model Parameters

By following a small set of clear rules, one can programmatically set a competent network architecture (i.e., the number and type of neuronal layers and the number of neurons comprising each layer).

Following this schema gives a competent architecture but probably not an optimal one hence once this network is initialized, you can iteratively tune the configuration during training using a number of ancillary algorithms; one family of these works by **pruning** nodes based on (small) values of the weight vector after a certain number of training epochs--in other words, eliminating unnecessary/redundant nodes.

Since every NN has three types of layers: input, hidden, and output. Creating the NN architecture therefore means coming up with values for the number of layers of each type and the number of nodes in each of these layers.

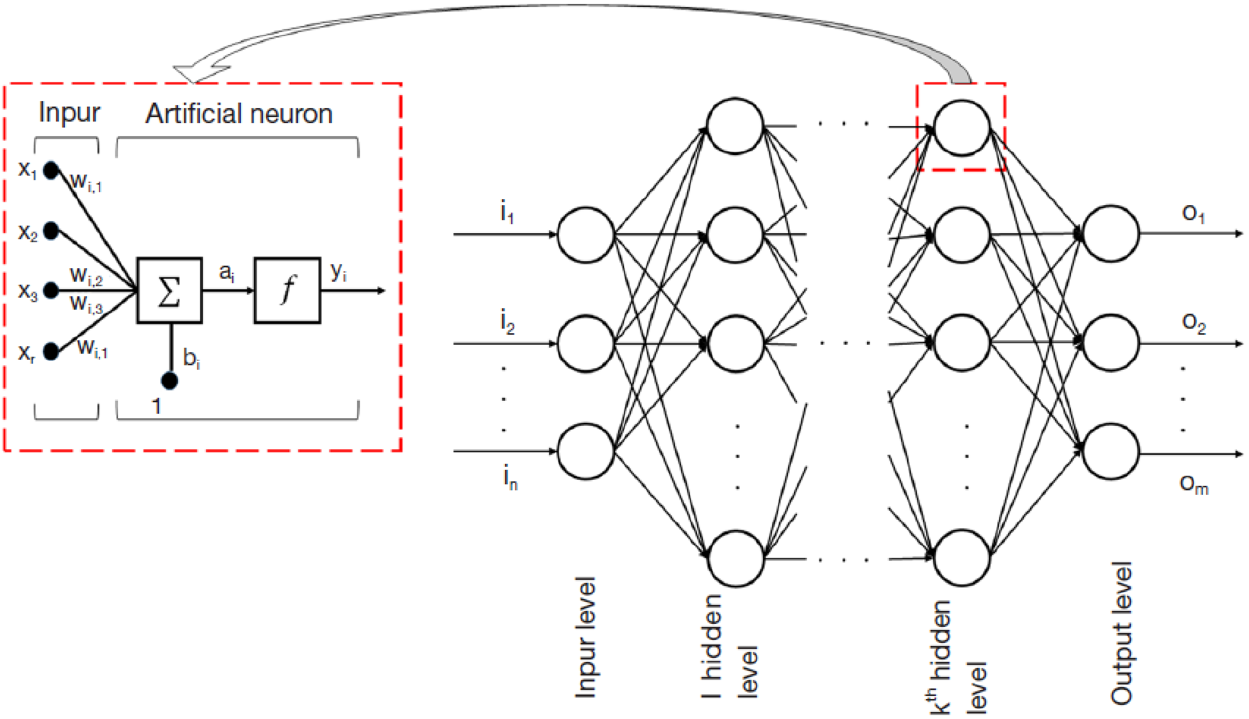


Fig 2.10: A basic neural network setup

i. <u>Input Layer</u>

Every Neural Network has exactly one input layer. With respect to the number of neurons comprising this layer, this parameter is completely and uniquely determined once you know the shape of your training data. Specifically, the number of neurons comprising that layer is equal to the number of features (columns) in your data. Some neural networks configurations add one additional node for a bias term.

ii. <u>Output Layer</u>

Like the Input layer, every neural network has exactly one output layer. Determining its size (number of neurons) is simple; it is completely determined by the chosen model configuration. Is your neural network going running in Machine Mode or Regression Mode (the ML convention of using a term that is also used in statistics but assigning a different meaning to it is very confusing). Machine mode: returns a class label (e.g., "Premium Account"/"Basic Account", “Positive”/”

Negative”). The Regression Mode returns a value (e.g., price).

If the neural network regresses, then the output layer has a single node. If the neural network is a classifier, then it also has a single node unless a softmax function is used in which case the output layer has one node per class label in your model.

iii. Hidden Layer

So those few rules set the number of layers and size (neurons/layer) for both the input and output layers. That leaves the hidden layers. How many hidden layers? Well if your data is linearly separable -no. 0 Table 2.1 (which you often know by the time you begin coding a neural network) then you don't need any hidden layers at all. Of course, you don't need an NN to resolve your data either, but it will still do the job. Beyond that, there's a mountain of commentary on the question of hidden layer configuration in neural networks. One issue within this subject on which there is a consensus is the performance difference from adding additional hidden layers: the situations in which performance improves with a second (or third, etc.) hidden layer are very small (I will verify this in results). One hidden layer is sufficient for the large majority of problems. Problems that require two hidden layers are rarely encountered. However, neural networks with two hidden layers can represent functions with any kind of shape. There is currently no theoretical reason to use neural networks with any more than two hidden layers. In fact, for many practical problems, there is no reason to use any more than one hidden layer.

No. of Hidden Layer Units	**Consequence**	**Example**
0	Only capable of representing linear separable functions or decisions.	y=x, PASSTHROUGH
1	Can approximate any function that contains a continuous mapping from one finite space to another.	y=wx, y=wx+b , AND, OR, NOT
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy	$y=x^2$, $y=wx^3+b$, XOR

Table 2.1: Effect of the Choice of the Number of Hidden Layer Units

So, what about size of the hidden layer(s) -- how many neurons? There are some empirically-derived rules-of-thumb, of these, the most commonly relied on is *'the optimal size of the hidden layer is usually between the size of the input and size of the output layers*'.

As we will see, in general neural networks with more hidden layers (going deeper) get increasingly harder to train and thus those problems that require a sizeable number of hidden layers are considered "non-solvable" by neural networks. The only exception to this are Convolutional Neural Networks (CNNs) which perform better the deeper they are.

In sum, for most problems, one could probably get decent performance (even without a second optimization step) by setting the hidden layer configuration using just two rules:

1) The number of hidden layers equals one
2) The number of neurons in that layer is the mean of the neurons in the input and output layers
3) The number of hidden neurons should be between the size of the input layer and the size of the output layer
4) The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer
5) The number of hidden neurons should be less than twice the size of the input layer

Deciding the number of hidden neuron layers is only a small part of the problem. You must also determine how many neurons will be in each of these hidden layers. Both the number of hidden layers and the number of neurons in each of these hidden layers must be carefully considered. The number of hidden layer units is a measure of the model’s learning capacity. If we give the network too few learning capacity (using too few neurons in the hidden layers), it will result in something called **underfitting**. Underfitting (shown in Fig 2.11) occurs when there are too few neurons in the hidden layers to adequately learn and detect the signals in a complicated data set.

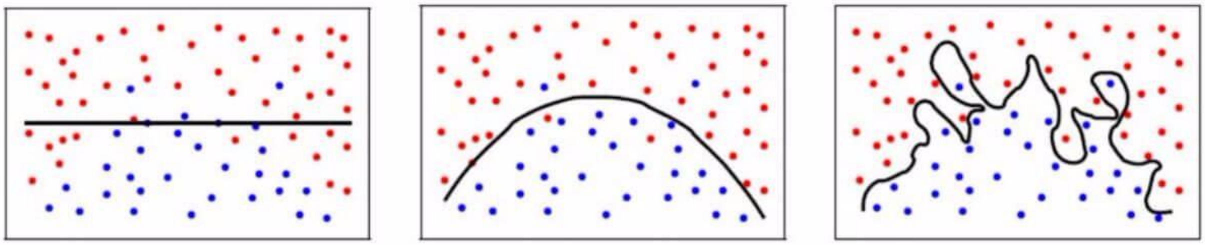


Fig 2.11 : Underfitting, Right fit and overfitting illustration

If we give the network too much learning capacity (using too many neurons in the hidden layers) it can result in several problems. First, too many neurons in the hidden layers may result in

overfitting (memorize the dataset). Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. The effect of this is that the training accuracy is so much better than the validation accuracy. Overfitting, right-fit and underfitting are illustrated in Fig 2.10.

In an online commentary with ***Steffen B Petersen · Aalborg University,*** he suggested that:

> “In order to secure the ability of the network to generalize, the number of nodes has to be kept as low as possible. If you have a large excess of nodes, you network becomes a memory bank that can recall the training set to perfection, but does not perform well on samples that was not part of the training set.”

A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers.

In general, keep adding more hidden layer units up to just before the validation accuracy starts getting worse. A universal approximation theorem of neural networks is that a neural network with a bounded, continuous activation functions and given enough hidden units in only one hidden layer can approximate any function.

2.4.6. Optimization of the Network Configuration (Pruning)

Pruning describes a set of techniques to trim network size (by nodes not layers) to improve computational performance and sometimes resolution performance. The gist of these techniques is removing nodes from the network during training by identifying those nodes which, if removed from the network, would not noticeably affect network performance (i.e., resolution of the data). (Even without using a formal pruning technique, you can get a rough idea of which nodes are not important by looking at your weight matrix after training; look weights very close to zero--it's the nodes on either end of those weights that are often removed during pruning.)
Put another way, by applying a pruning algorithm to your network during training, you can approach optimal network configuration by:

- ➔ Reducing any redundancies in the network
- ➔ Remove focus from the unimportant parts to the important aspects during training
- ➔ Improve the signal-to-noise ratio (SNR), which in turn increases the speed of the training, and this makes it possible to use more data for training.
- ➔ Eliminate outliers i.e. in sentiment analysis eliminate the words with highest frequency

and those that do not have a frequency (Frequent and infrequent).

2.4.7. Bag-of-Words Vectorization Model

The **bag-of-words model** is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

The bag-of-words model is mainly used as a tool of feature generation. After transforming the text into a "bag of words", we can calculate various measures to characterize the text. The most common type of characteristics, or features calculated from the Bag-of-words model is word frequency, namely, the number of times a term appears in the text. It is a common model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier.

A neural network can only understand numbers and therefore a question arises: How do we convert our text into meaningful data i.e. numerical representation for machine learning? One common approach is called Bag-of-Words model. The Bag-of-Words model learns a vocabulary from all of the texts, then models each text by counting the number of times each word appears.

For example, if we consider the following two sentences:

Sentence 1: "*The cat sat on the hat*"

Sentence 2: "*The dog ate the cat and the hat*"

From these two sentences, our vocabulary is as follows: {the*, cat, sat, on, hat, dog, ate, and*}

To get our bags-of-words, we count the number of times each word occurs in each sentence. In Sentence 1, "*the*" appears twice, and "*cat*", "*sat*", "*on*", and "*hat*" each appear once, so the feature vector for Sentence 1 is:

{*the, cat, sat, on, hat, dog, ate, and*}: ***Sentence 1***: {2, 1, 1, 1, 1, 0, 0, 0}

Similarly, the features for Sentence 2 are:

{*the, cat, sat, on, hat, dog, ate, and*}: ***Sentence 2***: {3, 1, 0, 0, 1, 1, 1, 1}

However, word frequencies are not necessarily the best representation for the text. Common words like "the", “him”, “on”, "a", "to", which are noise, are almost always the terms with the highest frequencies in the text. Thus, having a high raw count does not necessarily mean that the corresponding word is more important (and thus should be weighted more). If we therefore use the frequency of the words as the numerical representation of the words in text to feed into any machine learning algorithm, the accuracy may not be that satisfactory.

In order to address this problem, one of the most popular ways to "normalize" the term frequencies is to weight a term by the inverse of document frequency. Additionally, for the specific purpose of classification, supervised alternatives have been developed that take into account the class label of a document or text. Lastly, binary (presence/absence or 1/0) weighting is used in place of frequencies for some problems.

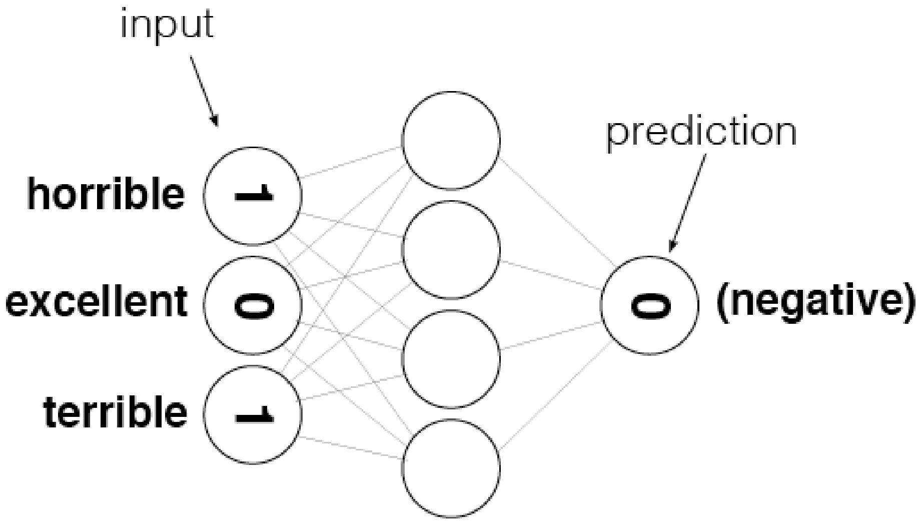


Fig 2.12: Binary (Absence or 0) Weighting

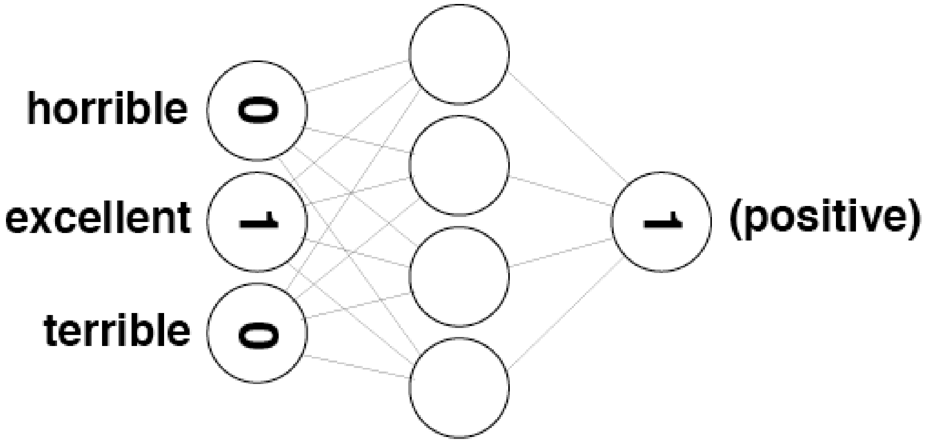


Fig 2.13: Binary (Presence or 1) Weighting

2.5. Model Evaluation

2.5.1. Cross-Validation

Cross validation is a model evaluation method to indicate how well the machine model will do when it is asked to make new predictions for data it has not already seen. The basic idea of cross-validation is that we do not use the entire data set when training a model. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on “new” data. The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the *validation dataset*), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent and unseen dataset.

Suppose we have a model with one or more unknown parameters, and a data set to which the model can be fit (the training data set). The fitting process optimizes the model parameters to make the model fit the training data as well as possible. If we then take an independent sample of validation data from the same population as the training data, it will generally turn out that the model does not fit the validation data as well as it fits the training data. This is called overfitting, since the model has crammed the training data. It is particularly likely to happen when the size of the training data set is small, or when the number of parameters in the model is large. Cross-validation is a way to predict the fit of a model to a hypothetical validation set.

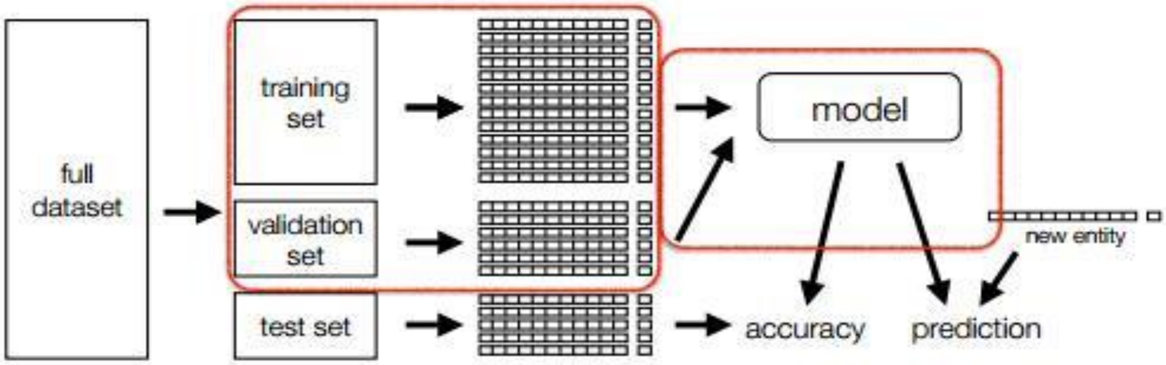
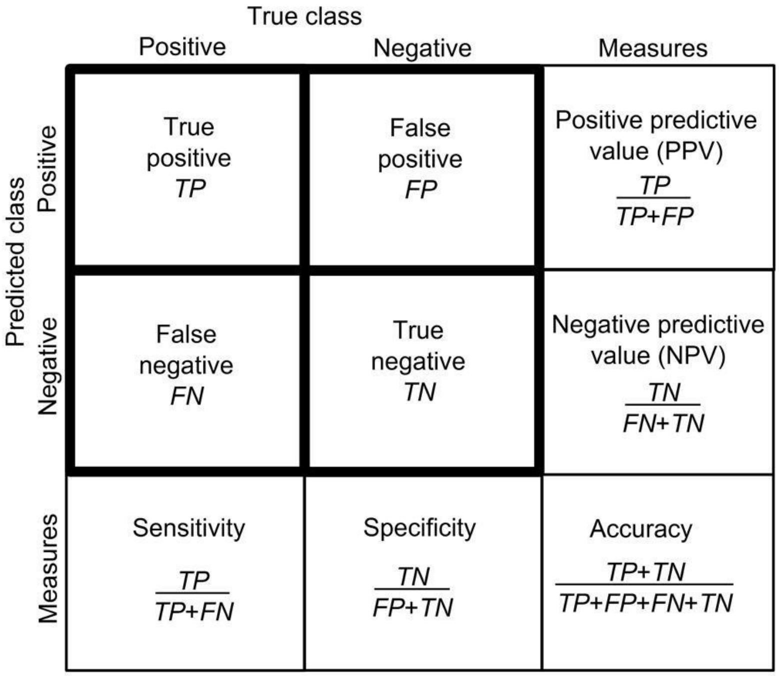


Fig 2.14: Cross Validation

In general, it is worthy to note that cross-validation only yields meaningful results if the validation set and training set are drawn from the same population and only if human biases are controlled. Since the parameters of the machine model are tuned using the training set, we generally expect the validation and testing accuracy to be lower than the training accuracy as shown in ***Fig 3.4***.

2.5.2. Confusion Matrix

A confusion matrix, as illustrated in Fig 2.15, is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It is relatively simple to understand, but the related terminology can be confusing.



		True class		
		Positive	Negative	Measures
Predicted class	Positive	True positive *TP*	False positive *FP*	Positive predictive value (PPV) $\frac{TP}{TP+FP}$
	Negative	False negative *FN*	True negative *TN*	Negative predictive value (NPV) $\frac{TN}{FN+TN}$
Measures		Sensitivity $\frac{TP}{TP+FN}$	Specificity $\frac{TN}{FP+TN}$	Accuracy $\frac{TP+TN}{TP+FP+FN+TN}$

Fig 2.15: Different metrics we can get from the confusion matrix

- Precision (PPV), p tells us what proportion of texts were classified as having positive sentiment were actually of positive sentiment.
- Recall (Sensitivity), r tells us what proportion of texts that actually had positive sentiment were classified as positive sentiment.

2.5.3. <u>F-Score</u>

In statistical analysis of binary classification, the **F-score** (also **F-measure**) is a measure of a test's accuracy. It considers both the *precision p* and the *recall(sensitivity) r* of the test to compute the F-score.

$$F - Score = \frac{2pr}{p + r}$$

***Eq 2.2:** F-Score formula*

The precision, p is the number of correctly classified examples divided by the total number of classified examples. The recall, r is the number of correctly classified examples divided by the actual number of examples in the training set. The F-score can be interpreted as a weighted average of the precision and recall, where an F-score reaches its best value at 1 and worst at 0. we instead measure 2 values, precision and recall. A more intuitive way of looking at it is that: precision, p is how accurate you are about saying that somebody will like your show. if I said that 8 people will like the show, and 6 of them do, then my accuracy is 75%. Recall, r is how good you are at identifying the ones that like it. There are 10 people who like the show, and I was able to identify 6 of them as people who would like it, I have a 60% recall.

The F-score is also used in machine learning. Note, however, that the F-measures do not take the true negatives (TN) into account, and that measures such as the Matthews correlation coefficient (MCC), may be preferable to assess the performance of a binary classifier, especially in cases where we have an unbalanced dataset.

2.5.4. MCC-Score

The *Matthews correlation coefficient (MCC)* is used in machine learning as a measure of the quality of binary (two-class) classifications, introduced by biochemist Brian W. Matthews in 1975. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between −1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and −1 indicates total disagreement between prediction and observation.

$$MCC - Score = \frac{TP * TN - FN * FP}{\sqrt{(TP+FN)(TP+FP)(TN+FN)(FN+FP)}}$$

Eq 2.3: MCC-Score formula

For determining the accuracy of a single classifier model, or comparing the results of different classifiers, the recall, precision, F-score, MCC-score, accuracy will be taken into consideration. F-Score is similar to MCC-Score for a balanced dataset. After training the machine learning model on the training dataset, the different validation techniques should yield as high as possible values which are near identical values if not similar. An intolerable difference in the values only implies overfitting or underfitting, in which case the model will not generalize well on unseen data.

3. METHODOLOGY AND DESIGN

3.1. Tools

3.1.1. Jupyter Notebook

This is a browser integrated python IDE. It allows one to writes python code and execute it stepwise. it is every good for functional programming with python.

3.1.2. Python

This is an interpreted programming language that has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library that can be able to support numeric and scientific computations, hence this makes it a perfect choice for this project.

3.1.3. Python Libraries

the standard python libraries that the project uses are:

1) Pandas, Numpy, for numerical computations and data structure.
2) Bokeh, matplotlib for graphing and visualization

3.1.4. Data Sources

DeepLearningNet - http://deeplearning.net/datasets/
This is an open source collection of different sets of data that can be used for benchmarking deep learning algorithms. The amazon reviews database for use in Sentiment analysis, and the spam database are all downloaded from there.

3.2. Network Architecture

The architecture used for the sentiment analysis neural network is a three layer neural network with only:

- ➔ 1 - input layer
- ➔ 1 - hidden layer
- ➔ 1 - output layer

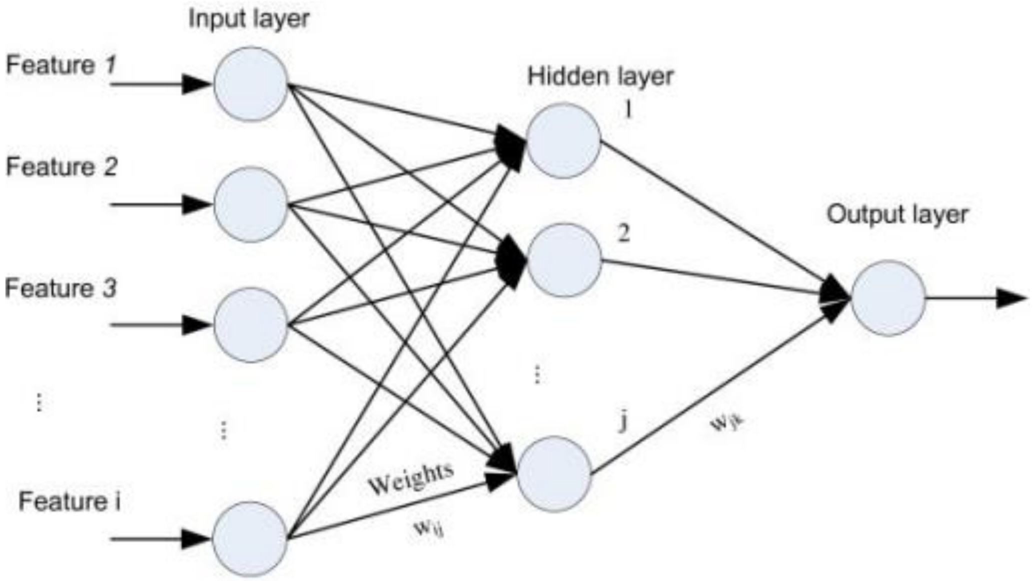


Fig 3.1: The 3-Layer Neural Network Architecture

The number of features is equal to the number of unique words in our training dataset. The bag-of-words model is used to convert the words into a numeric input, which can be used to weight the words.

3.3. Pseudocode

```
#Initialize the network parameters
Initialize all network data inputs [features, X], output [labels, Y] and hyperparameters :
Initialize all weights and bias with small random numbers, typically between -1 and 1 :

repeat:
        for every pattern in the training set:
                Present the pattern to the network:
                Randomly initialize the trainable parameters- weight and bias
               #Propagated the input feed forward through the network:
                for each layer in the network:
                        for every node in the layer
                                1. Calculate the weight sum of the inputs to the node
                                2. Add the bias threshold to the sum
                                3. Calculate the activation for the node
                        end
                end

               #Propagate the errors backward through the network
```

```
                for every node in the output layer:
                        calculate the error signal
                end

                for all hidden layers:
                        for every node in the layer
                                1. Calculate the node's signal error
                                2. Update each node's weight in the network
                        end
                end

               #Calculate Global Error
                Calculate the Error Function:
        end
while ((number of iterations < than specified) AND (Error Function is > than specified))
```

3.4. Mathematical Breakdown

The dataset has features $\mathbf{x_i}$ and expected outputs $\mathbf{y_i}$ as parameters to our network. Since the problem is linearly separable i.e. there is a single hyperplane which can separate the two classes, we assume a linear equation of the form $\mathbf{y=wx_i +b_i}$ **.** Then $\mathbf{b_i}$ and $\mathbf{w_i}$ are the trainable hyperparameters (learnables) which the network has to learn in order to find the values that minimize the error most, if not zero.

The neural network has the following parameters:

- ➔ Input matrix, X
- ➔ Expected output matrix, Y
- ➔ Predicted output, P
- ➔ Weight matrix, W
- ➔ Bias matrix, B
- ➔ Activation function, σ
- ➔ Loss function Q
- ➔ Mean squared error, MSE
- ➔ Learning rate, η

The neural network is supposed to minimize the loss in order to give us a line of best fit of the form **P = WX + B**. Since **X** is fixed, the gradient descent algorithm will repeatedly adjust the values of **W**, and **B** until it gives the least possible error.

A standard gradient descent algorithm involves the following steps:

1. For each record in the training data:
   a. Make a forward pass through the network, calculating the output.
   ➔ Output of the input layer
   $S(w,x,b) = \sum_{i=0}^{n} \quad (WX \ + \ B)$
   ➔ Output of the hidden layer
   $P(S(w,x,b)) = \sigma(s)$

   b. Calculate the loss function,
   ➔ Mean square error
   $Q(P(S(w,x))) \ = MSE \ (P(s)) \ = 0.5 \ (Y - \ P(s))^{2}$

   c. Calculate the gradient of the loss function in the output unit
   ➔ Gradient of the loss function
   $\frac{\partial(Q(P(s)))}{\partial W} = [(\frac{\partial Q}{\partial P} \times \frac{\partial P}{\partial S} \times \frac{\partial S}{\partial W})]$
   $= -(Y - P) \times \sigma \times X$
   $= -\delta \times X$

   d. Calculate the weight step for hidden-to-output layer weights
   ➔ $\Delta W \ = \ -\eta \times \delta \times X$

   e. Update the weight, vector hidden-to-output weights
   ➔ $W = \ W \ + \Delta W$

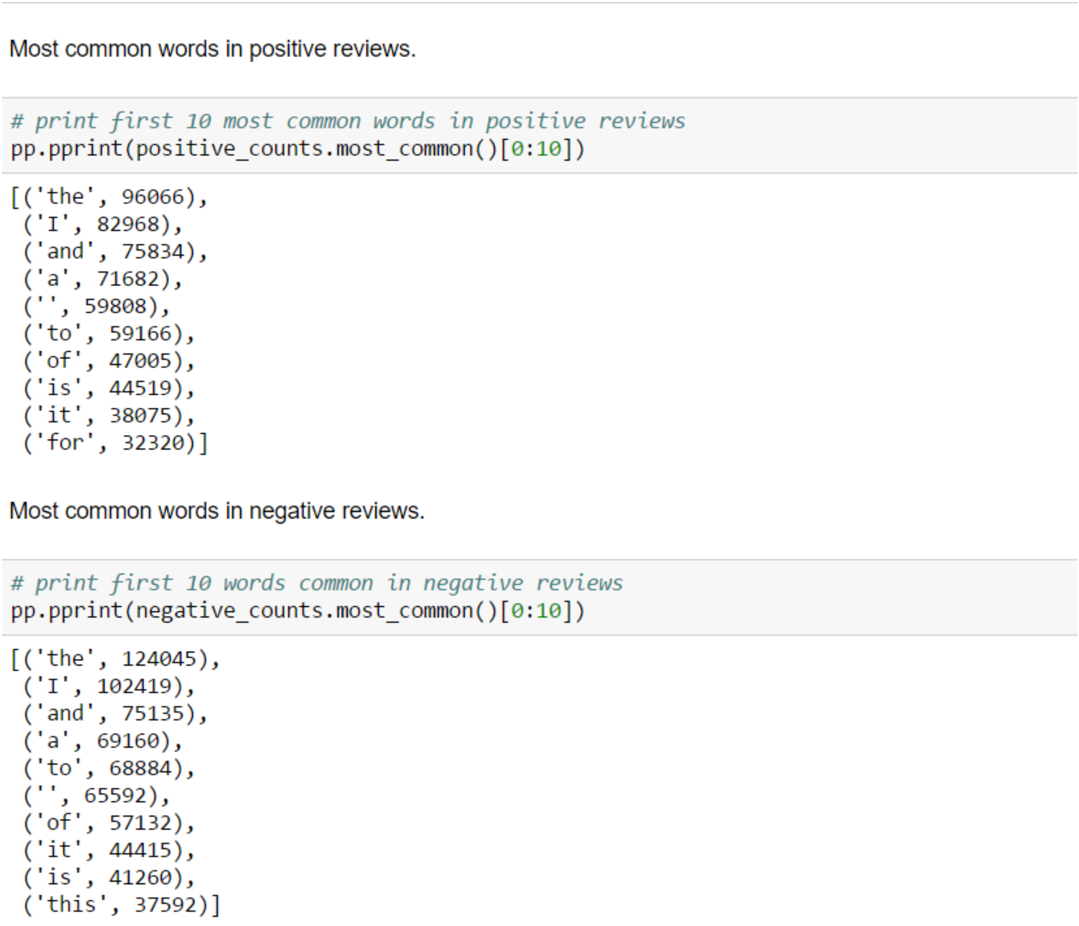
   f. Repeat steps e and d for the input-to-hidden layer weights
   ➔ $W = \ W \ + \Delta W$

2. Repeat step 2 for e number of iterations/epochs.

3.5. Predictive Theory

Right before starting the sentimental analysis and spam classification using neural networks, there is need to certify a theory that summarizes the structure of the data that will be fed into the network i.e. features. Since, textual information, like any other signal can contain a significant amount of noise. A general theory, will be able to help us distinguish between noise and spam or sentimental words. It is a question of GIGO (Garbage in Garbage Out) hence the Signal-to-Noise-Ratio (SNR) should be high as possible.

If we take the frequency of each word as the numerical representation of the words and feed into the neural network, then most of the words that will contribute to the weights of our neural network are really neutral words that have no meaning to the whether a text is spam or not. For example most common words like, ., !, !!, ?, the, and, to, this, that have no revelation.



Most common words in positive reviews.

```
# print first 10 most common words in positive reviews
pp.pprint(positive_counts.most_common()[0:10])
```

```
[('the', 96066),
 ('I', 82968),
 ('and', 75834),
 ('a', 71682),
 ('', 59808),
 ('to', 59166),
 ('of', 47005),
 ('is', 44519),
 ('it', 38075),
 ('for', 32320)]
```

Most common words in negative reviews.

```
# print first 10 words common in negative reviews
pp.pprint(negative_counts.most_common()[0:10])
```

```
[('the', 124045),
 ('I', 102419),
 ('and', 75135),
 ('a', 69160),
 ('to', 68884),
 ('', 65592),
 ('of', 57132),
 ('it', 44415),
 ('is', 41260),
 ('this', 37592)]
```

Fig 2.6:With noise, Similarity of most common words in both positive and negative reviews.

Because of that, it would really increase the efficiency and accuracy of our network by eliminating their contribution. We can do that by using the ratio of frequency of word in one class to its frequency in another class.

$$interclassratio = \frac{frequencyofwordinclassA}{frequencyofwordinclassB}$$

For example, for sentiment analysis with positive/negative classes, we get the ratio of the frequency of a word in negative reviews to that of the same word in positive reviews. This gives the affinity of certain words to appear in either a positive or negative review.

A graph of the interclass ratio for the sentiment analysis data used is as follows:

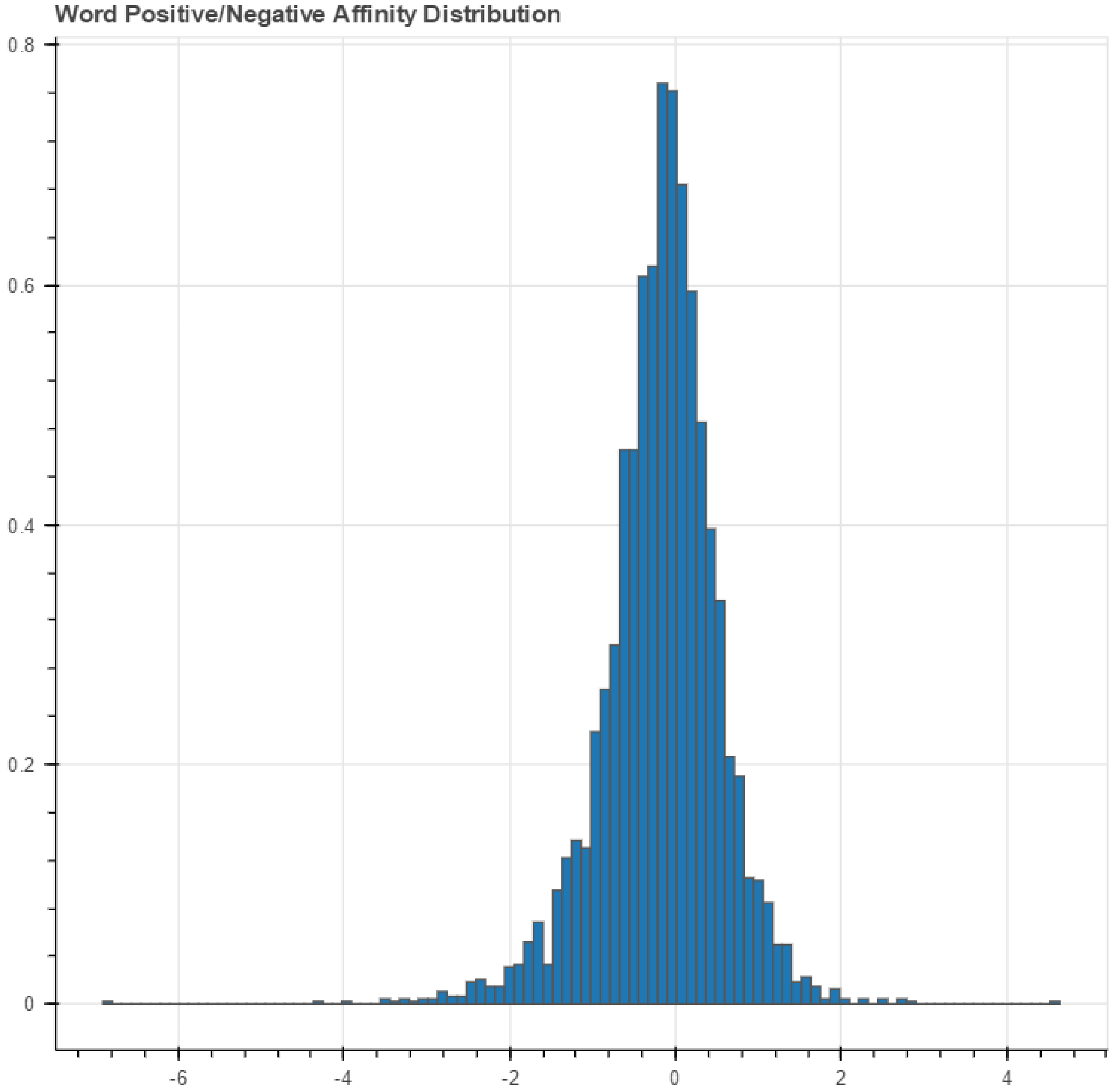
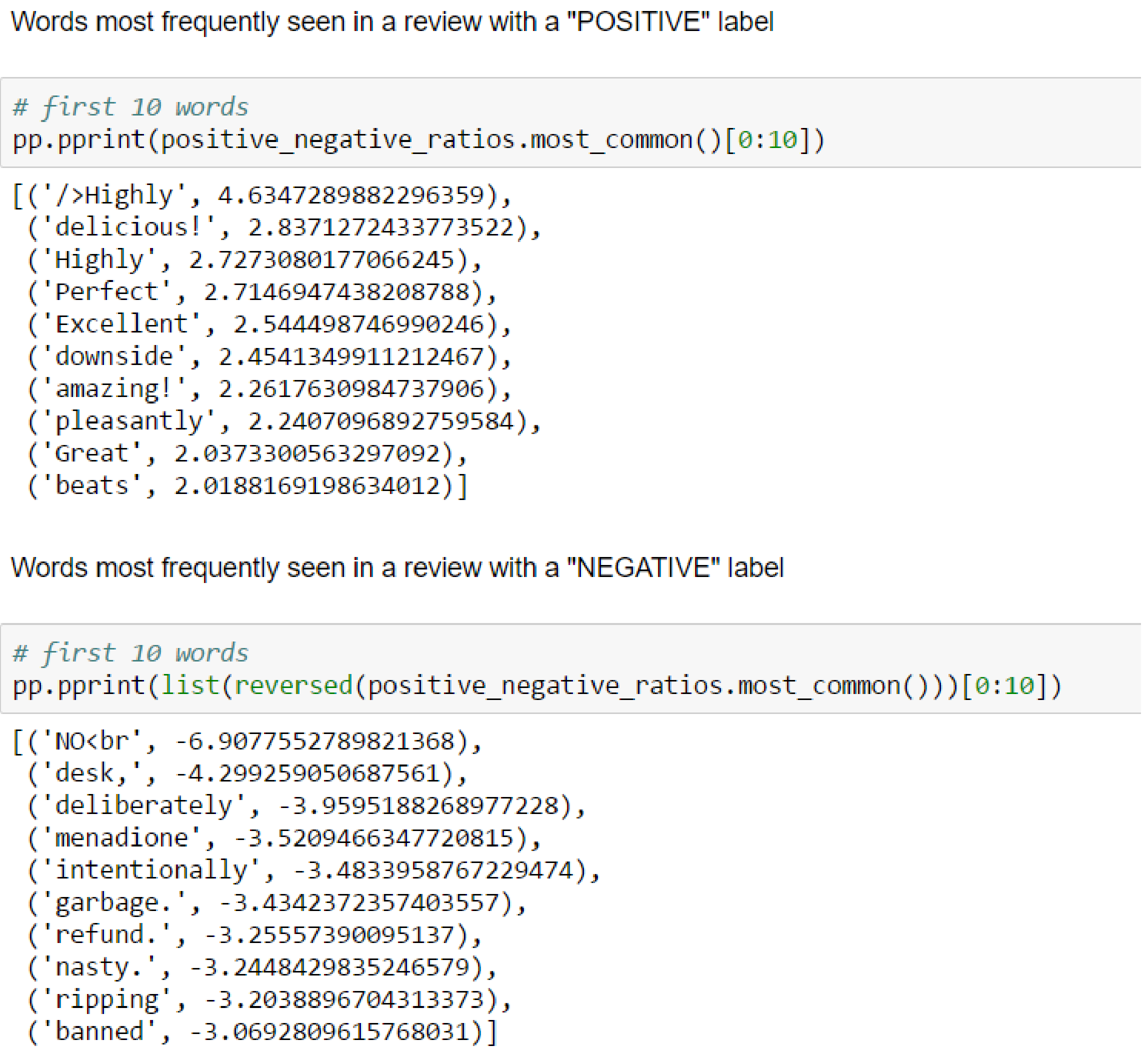


Fig 3.2 Interclass affinity distributions. Look at the distribution below, where these words (the middle of our distribution) are just noise being added to our network, and in the process reducing the efficiency and accuracy.

NB: The natural logarithm of the ratios was the used to make ratios greater than one positive and ratios less than one negative.



Words most frequently seen in a review with a "POSITIVE" label

```
# first 10 words
pp.pprint(positive_negative_ratios.most_common()[0:10])
```

```
[('/>Highly', 4.6347289882296359),
 ('delicious!', 2.8371272433773522),
 ('Highly', 2.7273080177066245),
 ('Perfect', 2.7146947438208788),
 ('Excellent', 2.544498746990246),
 ('downside', 2.4541349911212467),
 ('amazing!', 2.2617630984737906),
 ('pleasantly', 2.2407096892759584),
 ('Great', 2.0373300563297092),
 ('beats', 2.0188169198634012)]
```

Words most frequently seen in a review with a "NEGATIVE" label

```
# first 10 words
pp.pprint(list(reversed(positive_negative_ratios.most_common()))[0:10])
```

```
[('NO<br', -6.9077552789821368),
 ('desk,', -4.299259050687561),
 ('deliberately', -3.9595188268977228),
 ('menadione', -3.5209466347720815),
 ('intentionally', -3.4833958767229474),
 ('garbage.', -3.4342372357403557),
 ('refund.', -3.25557390095137),
 ('nasty.', -3.2448429835246579),
 ('ripping', -3.2038896704313373),
 ('banned', -3.0692809615768031)]
```

Fig 2.7: Denoised - Similarity of most common words in both positive and negative reviews.

In the neural network implementation, I used a variable called polarity to eliminate the words with affinity very close to zero, 0, on either side of the affinity distribution, since that is where the majority of the noise lies. I also, choose to eliminate words with very low frequency using a cut-off threshold, since a word that occurs only a few times is clearly not rich in information.

3.6. Procedure of Training

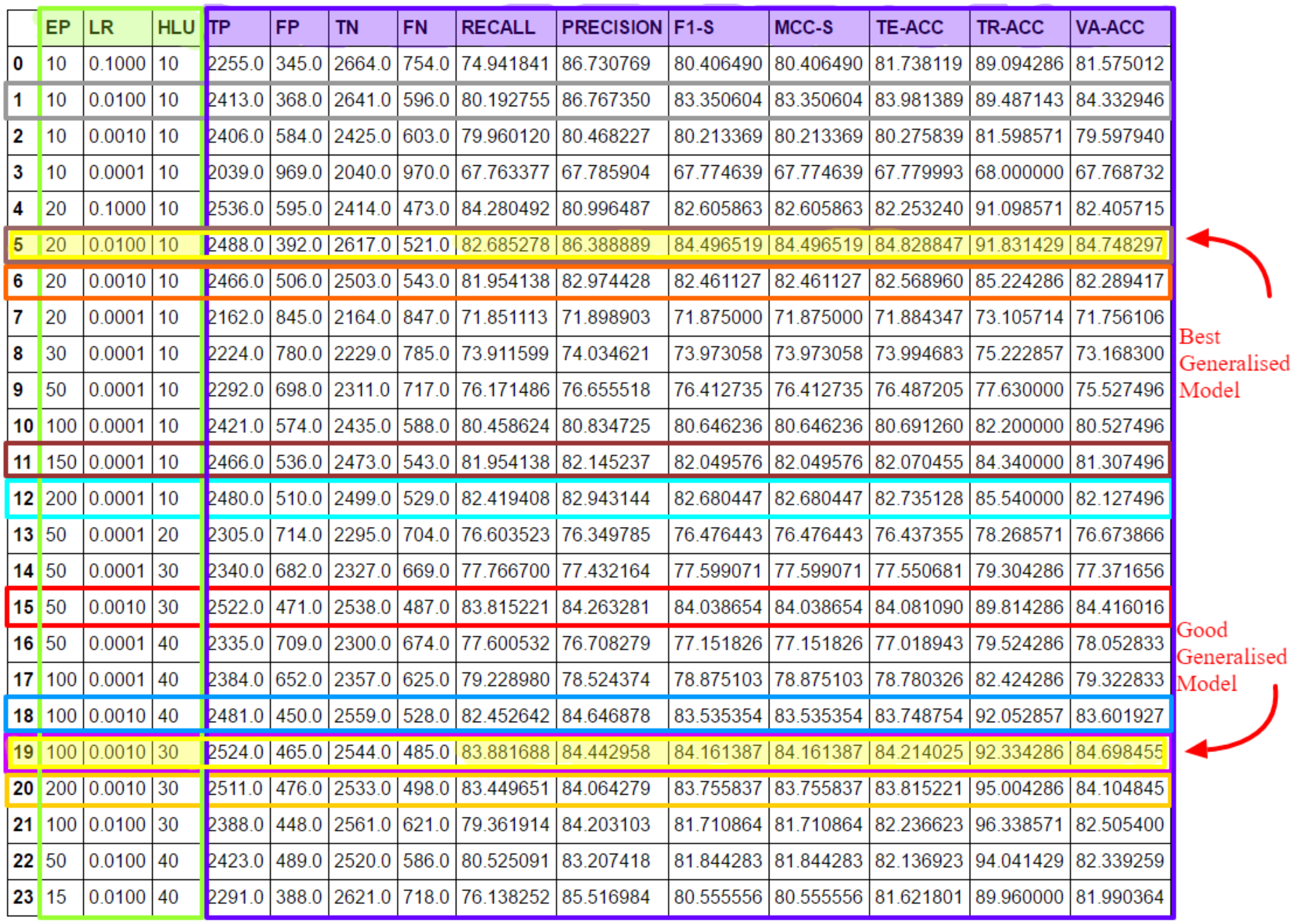
The hyper-parameters which are of interest in this particular problem are the learning rate, no. of hidden unit nodes and the no. of epochs. Each machine learning problem is normally unique and finding the empirical values is often as a result of experimentation, although there starting values are chosen as per the discussion in the literature review. After the neural network was setup as per the configuration steps above, it was trained on a set of different combination of parameters in order to come up with the best set that would not only give the best accuracy, but also avoid extreme overfitting or underfitting of the training data.

In addition, the validation methods i.e. cross validation accuracy, F1-Score and MCC-Score are used in combination at the end of the training to come up with the best model of the neural network that generalizes well with both seen and unseen data, the implication of which means that the validation methods should yield very near identical values that are as high as possible.

4. EXPERIMENTAL RESULTS AND ANALYSIS

4.1. Training Data

The following data was collected for different combinations of the hyperparameters and the accuracies obtained for each configuration tabulated as below:



	EP	LR	HLU	TP	FP	TN	FN	RECALL	PRECISION	F1-S	MCC-S	TE-ACC	TR-ACC	VA-ACC
0	10	0.1000	10	2255.0	345.0	2664.0	754.0	74.941841	86.730769	80.406490	80.406490	81.738119	89.094286	81.575012
1	10	0.0100	10	2413.0	368.0	2641.0	596.0	80.192755	86.767350	83.350604	83.350604	83.981389	89.487143	84.332946
2	10	0.0010	10	2406.0	584.0	2425.0	603.0	79.960120	80.468227	80.213369	80.213369	80.275839	81.598571	79.597940
3	10	0.0001	10	2039.0	969.0	2040.0	970.0	67.763377	67.785904	67.774639	67.774639	67.779993	68.000000	67.768732
4	20	0.1000	10	2536.0	595.0	2414.0	473.0	84.280492	80.996487	82.605863	82.605863	82.253240	91.098571	82.405715
5	20	0.0100	10	2488.0	392.0	2617.0	521.0	82.685278	86.388889	84.496519	84.496519	84.828847	91.831429	84.748297
6	20	0.0010	10	2466.0	506.0	2503.0	543.0	81.954138	82.974428	82.461127	82.461127	82.568960	85.224286	82.289417
7	20	0.0001	10	2162.0	845.0	2164.0	847.0	71.851113	71.898903	71.875000	71.875000	71.884347	73.105714	71.756106
8	30	0.0001	10	2224.0	780.0	2229.0	785.0	73.911599	74.034621	73.973058	73.973058	73.994683	75.222857	73.168300
9	50	0.0001	10	2292.0	698.0	2311.0	717.0	76.171486	76.655518	76.412735	76.412735	76.487205	77.630000	75.527496
10	100	0.0001	10	2421.0	574.0	2435.0	588.0	80.458624	80.834725	80.646236	80.646236	80.691260	82.200000	80.527496
11	150	0.0001	10	2466.0	536.0	2473.0	543.0	81.954138	82.145237	82.049576	82.049576	82.070455	84.340000	81.307496
12	200	0.0001	10	2480.0	510.0	2499.0	529.0	82.419408	82.943144	82.680447	82.680447	82.735128	85.540000	82.127496
13	50	0.0001	20	2305.0	714.0	2295.0	704.0	76.603523	76.349785	76.476443	76.476443	76.437355	78.268571	76.673866
14	50	0.0001	30	2340.0	682.0	2327.0	669.0	77.766700	77.432164	77.599071	77.599071	77.550681	79.304286	77.371656
15	50	0.0010	30	2522.0	471.0	2538.0	487.0	83.815221	84.263281	84.038654	84.038654	84.081090	89.814286	84.416016
16	50	0.0001	40	2335.0	709.0	2300.0	674.0	77.600532	76.708279	77.151826	77.151826	77.018943	79.524286	78.052833
17	100	0.0001	40	2384.0	652.0	2357.0	625.0	79.228980	78.524374	78.875103	78.875103	78.780326	82.424286	79.322833
18	100	0.0010	40	2481.0	450.0	2559.0	528.0	82.452642	84.646878	83.535354	83.535354	83.748754	92.052857	83.601927
19	100	0.0010	30	2524.0	465.0	2544.0	485.0	83.881688	84.442958	84.161387	84.161387	84.214025	92.334286	84.698455
20	200	0.0010	30	2511.0	476.0	2533.0	498.0	83.449651	84.064279	83.755837	83.755837	83.815221	95.004286	84.104845
21	100	0.0100	30	2388.0	448.0	2561.0	621.0	79.361914	84.203103	81.710864	81.710864	82.236623	96.338571	82.505400
22	50	0.0100	40	2423.0	489.0	2520.0	586.0	80.525091	83.207418	81.844283	81.844283	82.136923	94.041429	82.339259
23	15	0.0100	40	2291.0	388.0	2621.0	718.0	76.138252	85.516984	80.555556	80.555556	81.621801	89.960000	81.990364

Best Generalised Model

Good Generalised Model

Table 4.0: Table of Validation of Results of the Neural Network

NB:

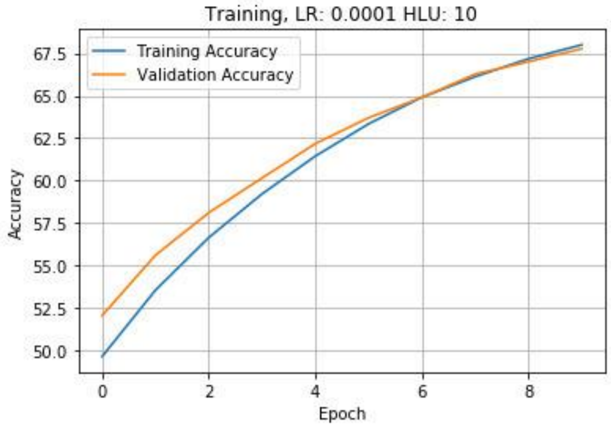
EP: No. of Epochs TP: True Positive MCC-S: MCC-Score

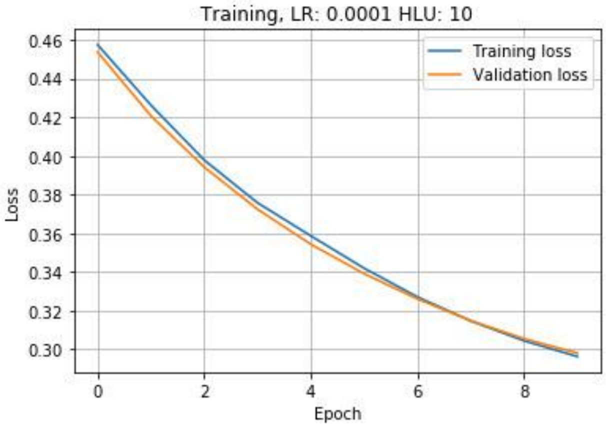
LR – Learning Rate FP: False Positive TE-ACC: Testing Accuracy

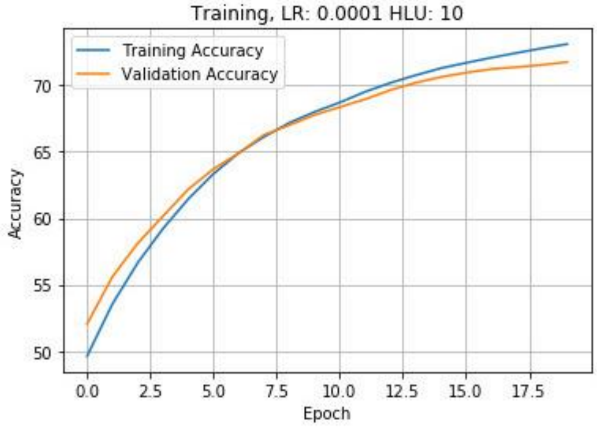
HLU – No. of hidden layer units TN: True Negative TR-ACC: Training Accuracy

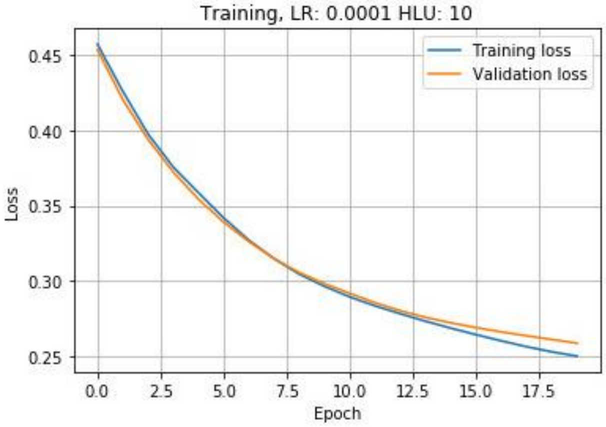
FN: False Negative F1-S: F1-Score VA-ACC: Validation Accuracy

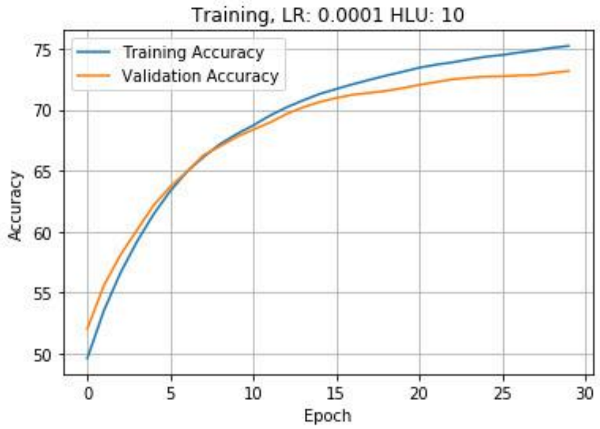
The corresponding validation\training loss\accuracy graphs for the same parameters are as below:

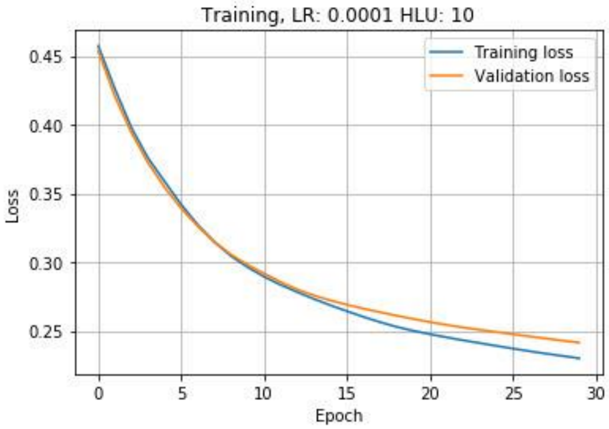


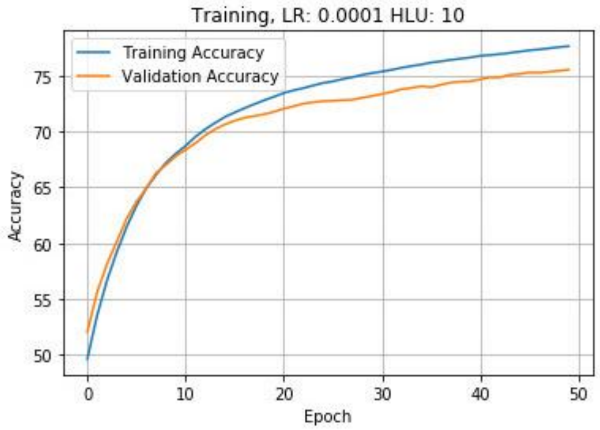


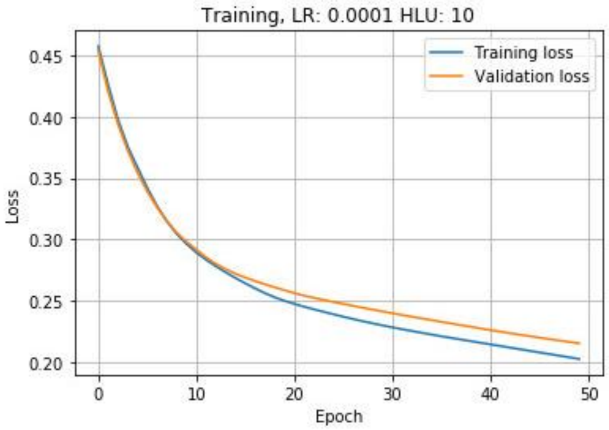


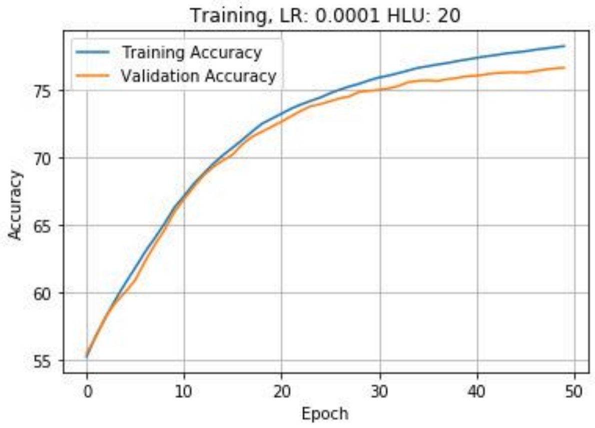


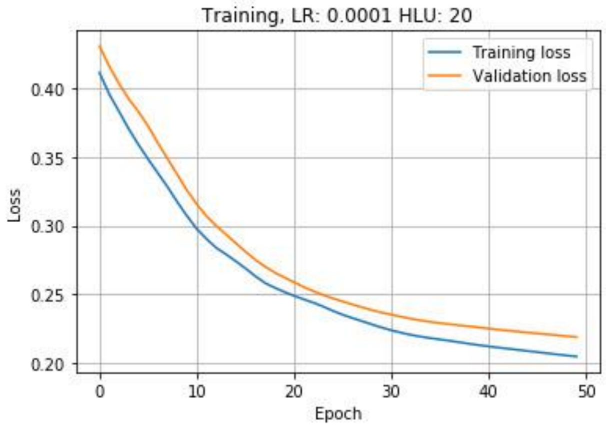

Training, LR: 0.0001 HLU: 10
Training Accuracy
Validation Accuracy
Accuracy
75
70
65
60
55
50
0
5
10
15
20
25
30
Epoch

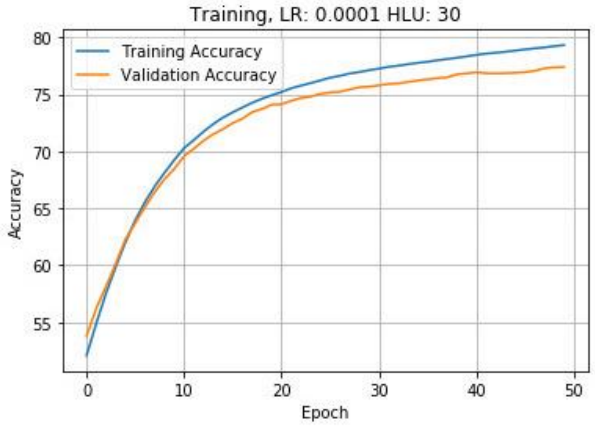

Training, LR: 0.0001 HLU: 10
Training loss
Validation loss
Loss
0.45
0.40
0.35
0.30
0.25
0
5
10
15
20
25
30
Epoch

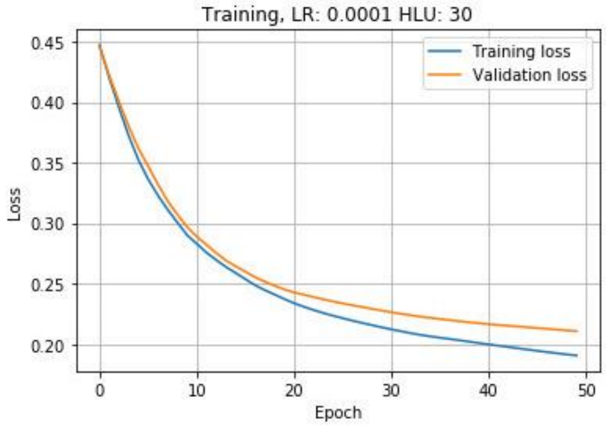

Training, LR: 0.0001 HLU: 10
Training Accuracy
Validation Accuracy
Accuracy
75
70
65
60
55
50
0
10
20
30
40
50
Epoch

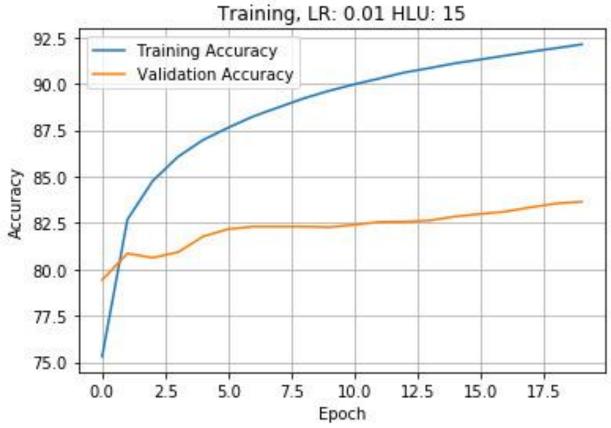

Training, LR: 0.0001 HLU: 10
Training loss
Validation loss
Loss
0.45
0.40
0.35
0.30
0.25
0.20
0
10
20
30
40
50
Epoch

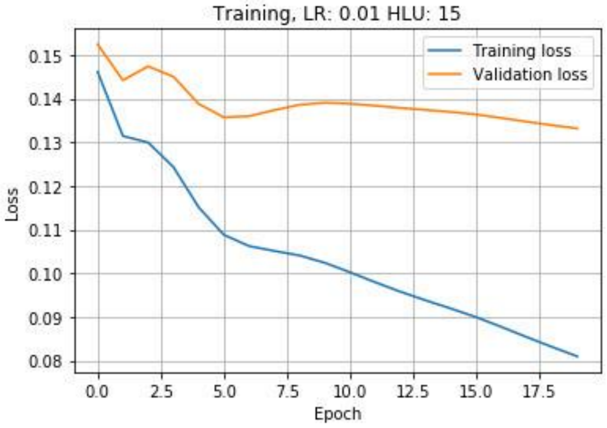

Training, LR: 0.0001 HLU: 20
Training Accuracy
Validation Accuracy
Accuracy
75
70
65
60
55
0
10
20
30
40
50
Epoch

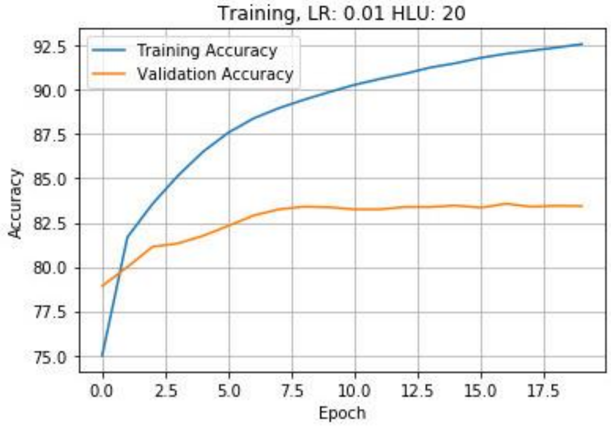

Training, LR: 0.0001 HLU: 20
Training loss
Validation loss
Loss
0.40
0.35
0.30
0.25
0.20
0
10
20
30
40
50
Epoch

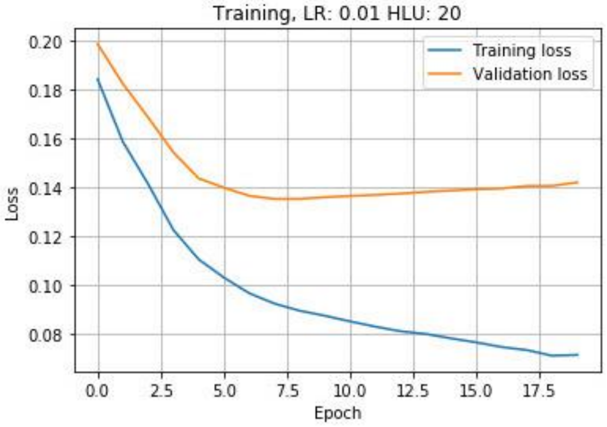

Training, LR: 0.0001 HLU: 30
Training Accuracy
Validation Accuracy
Accuracy
80
75
70
65
60
55
0
10
20
30
40
50
Epoch

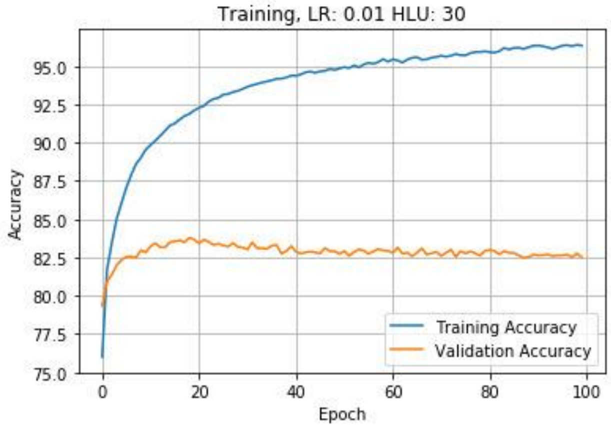

Training, LR: 0.0001 HLU: 30
Training loss
Validation loss
Loss
0.45
0.40
0.35
0.30
0.25
0.20
0
10
20
30
40
50
Epoch

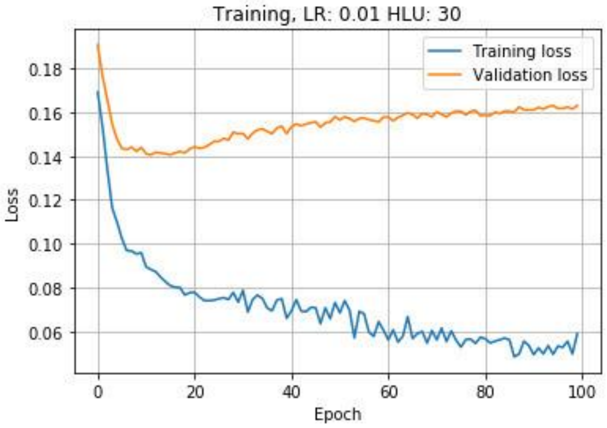

Training, LR: 0.01 HLU: 15
Training Accuracy
Validation Accuracy
Accuracy
92.5
90.0
87.5
85.0
82.5
80.0
77.5
75.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

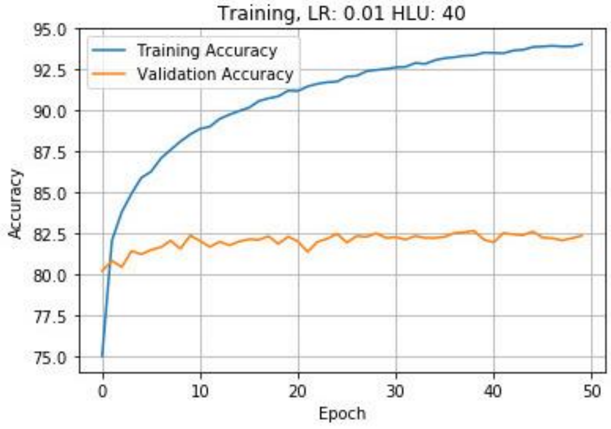

Training, LR: 0.01 HLU: 15
Training loss
Validation loss
Loss
0.15
0.14
0.13
0.12
0.11
0.10
0.09
0.08
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

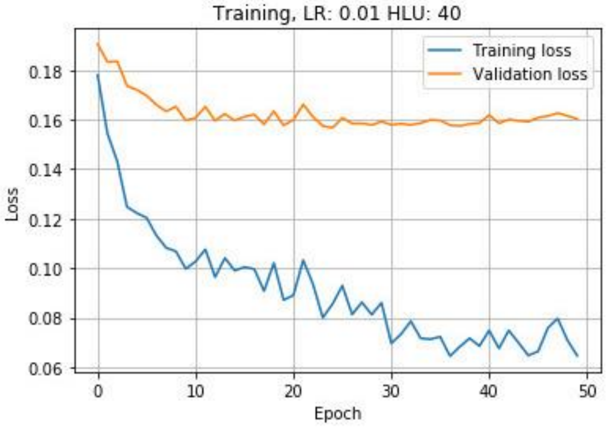

Training, LR: 0.01 HLU: 20
Training Accuracy
Validation Accuracy
Accuracy
92.5
90.0
87.5
85.0
82.5
80.0
77.5
75.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

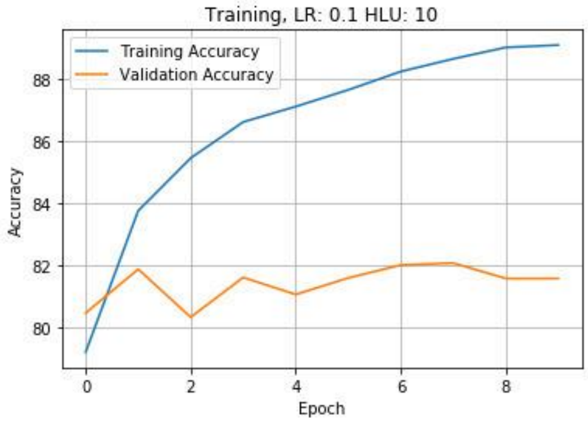

Training, LR: 0.01 HLU: 20
Training loss
Validation loss
Loss
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

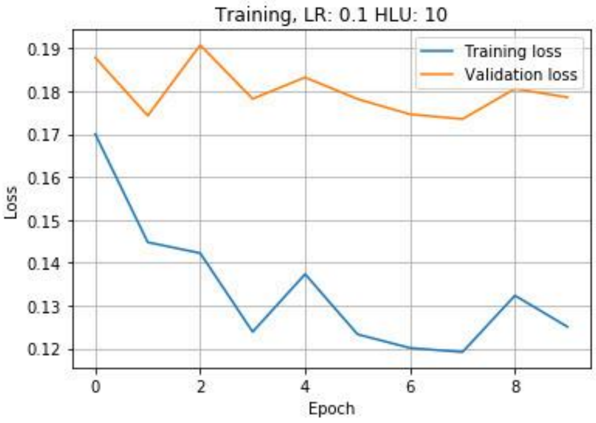

Training, LR: 0.01 HLU: 30
Training Accuracy
Validation Accuracy
Accuracy
95.0
92.5
90.0
87.5
85.0
82.5
80.0
77.5
75.0
0
20
40
60
80
100
Epoch

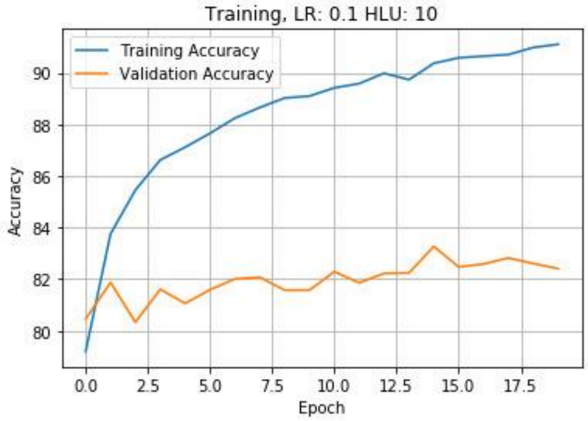

Training, LR: 0.01 HLU: 30
Training loss
Validation loss
Loss
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0
20
40
60
80
100
Epoch

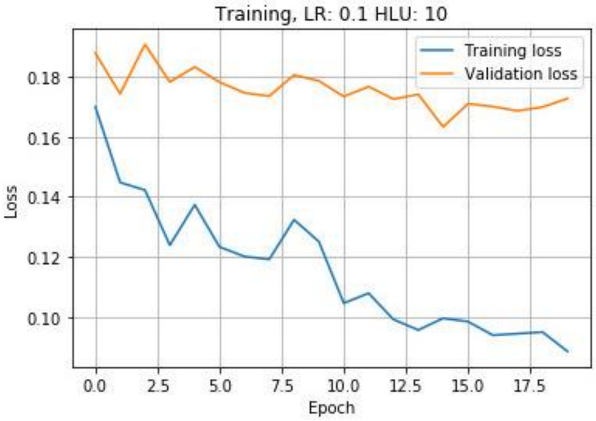

Training, LR: 0.01 HLU: 40
Training Accuracy
Validation Accuracy
Accuracy
95.0
92.5
90.0
87.5
85.0
82.5
80.0
77.5
75.0
0
10
20
30
40
50
Epoch

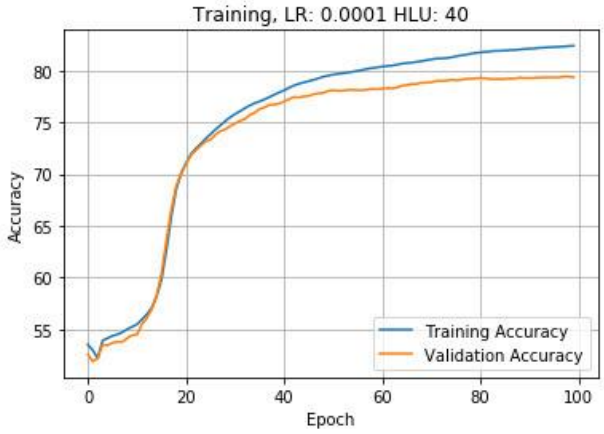

Training, LR: 0.01 HLU: 40
Training loss
Validation loss
Loss
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0
10
20
30
40
50
Epoch

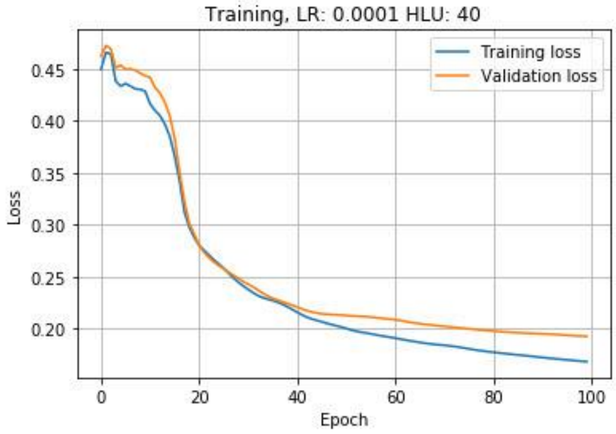

Training, LR: 0.1 HLU: 10
Training Accuracy
Validation Accuracy
Accuracy
Epoch
80
82
84
86
88
0
2
4
6
8

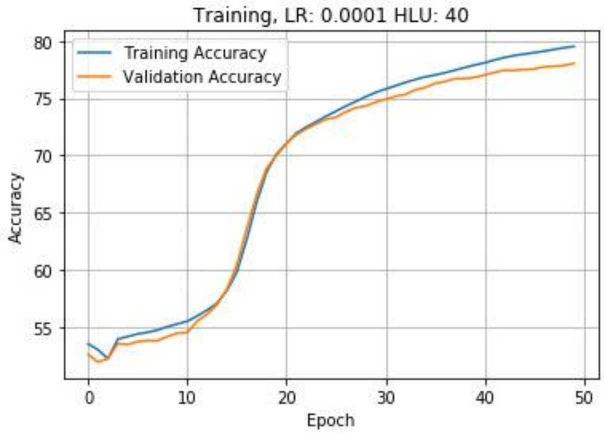

Training, LR: 0.1 HLU: 10
Training loss
Validation loss
Loss
Epoch
0.12
0.13
0.14
0.15
0.16
0.17
0.18
0.19
0
2
4
6
8

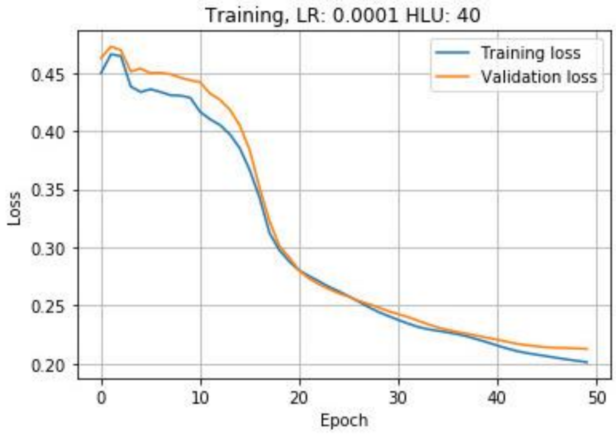

Training, LR: 0.1 HLU: 10
Training Accuracy
Validation Accuracy
Accuracy
Epoch
80
82
84
86
88
90
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

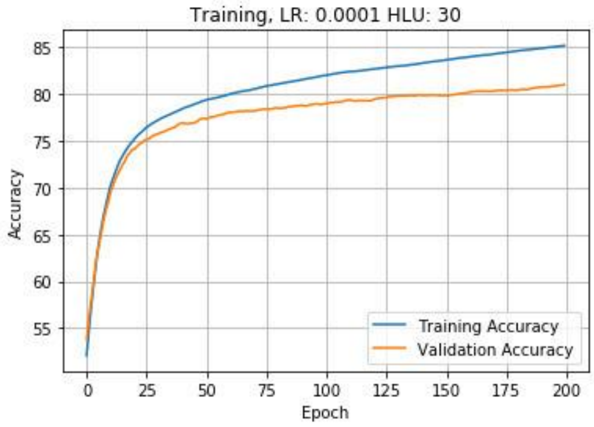

Training, LR: 0.1 HLU: 10
Training loss
Validation loss
Loss
Epoch
0.10
0.12
0.14
0.16
0.18
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

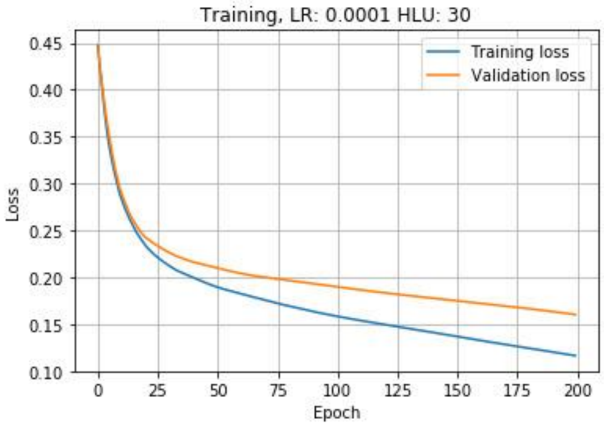

Training, LR: 0.0001 HLU: 40
Training Accuracy
Validation Accuracy
Accuracy
Epoch
55
60
65
70
75
80
0
20
40
60
80
100

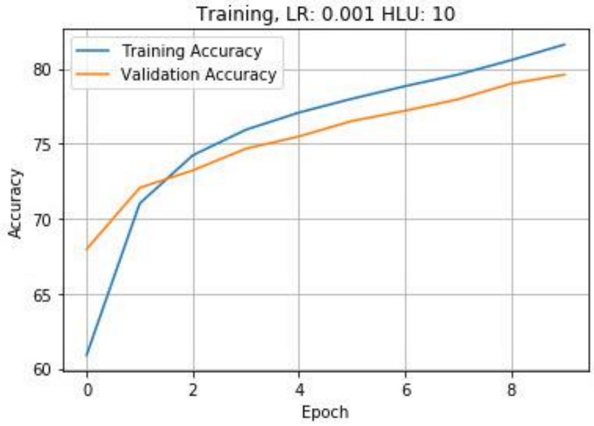

Training, LR: 0.0001 HLU: 40
Training loss
Validation loss
Loss
Epoch
0.20
0.25
0.30
0.35
0.40
0.45
0
20
40
60
80
100

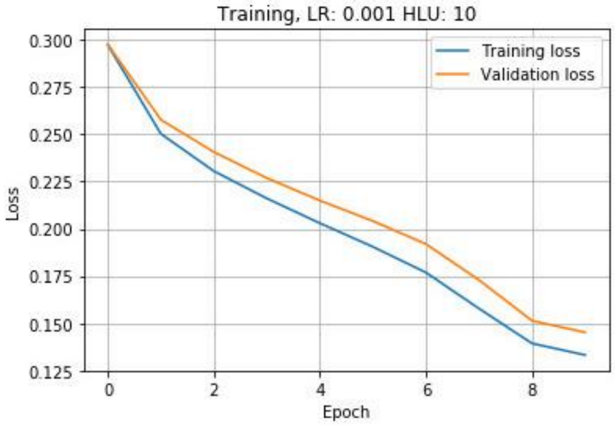

Training, LR: 0.0001 HLU: 40
Training Accuracy
Validation Accuracy
Accuracy
Epoch
55
60
65
70
75
80
0
10
20
30
40
50

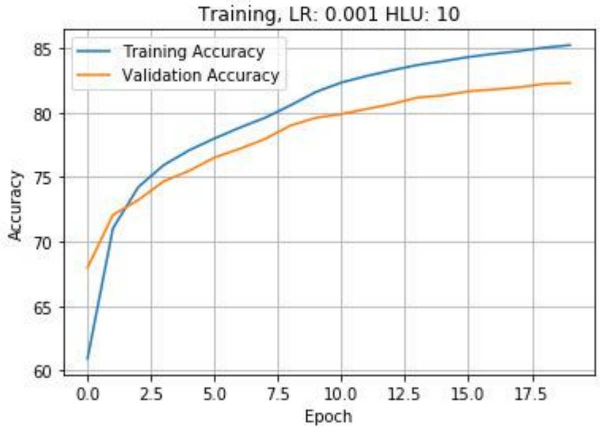

Training, LR: 0.0001 HLU: 40
Training loss
Validation loss
Loss
Epoch
0.20
0.25
0.30
0.35
0.40
0.45
0
10
20
30
40
50

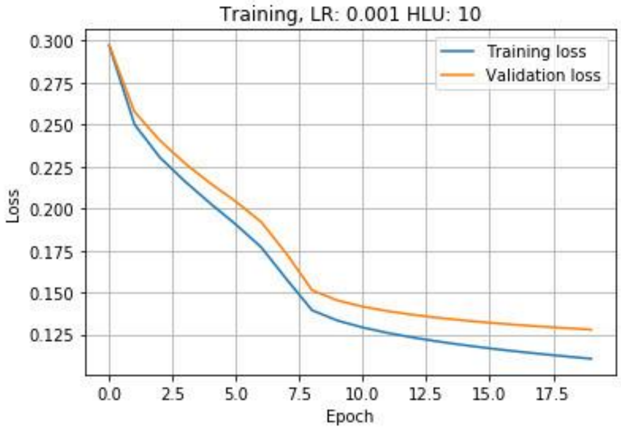

Training, LR: 0.0001 HLU: 30
Accuracy
85
80
75
70
65
60
55
0
25
50
75
100
125
150
175
200
Epoch
Training Accuracy
Validation Accuracy

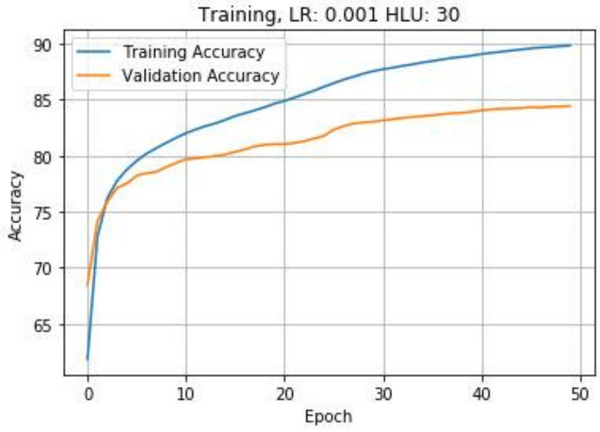

Training, LR: 0.0001 HLU: 30
Loss
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
25
50
75
100
125
150
175
200
Epoch
Training loss
Validation loss

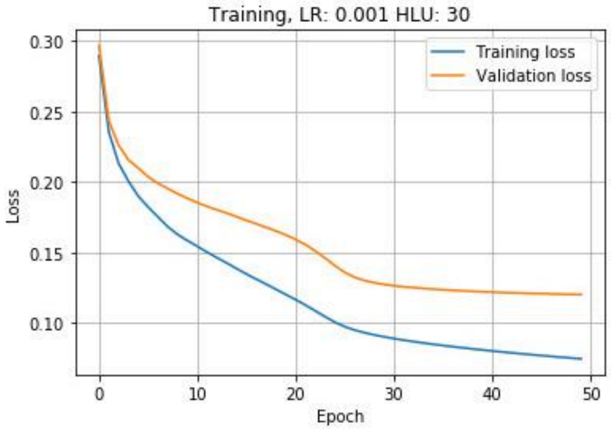

Training, LR: 0.001 HLU: 10
Accuracy
80
75
70
65
60
0
2
4
6
8
Epoch
Training Accuracy
Validation Accuracy

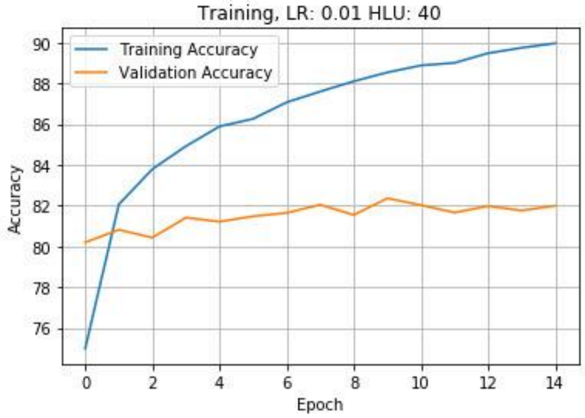

Training, LR: 0.001 HLU: 10
Loss
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0
2
4
6
8
Epoch
Training loss
Validation loss

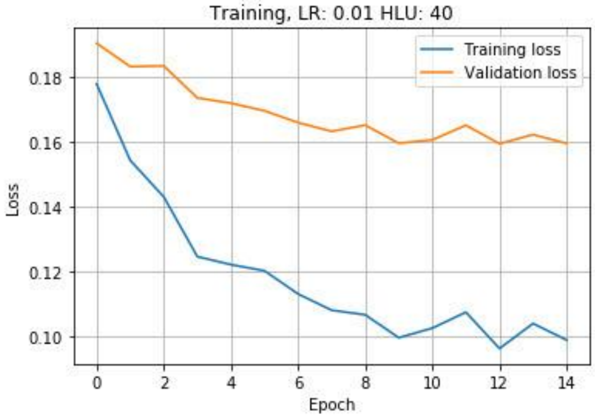

Training, LR: 0.001 HLU: 10
Accuracy
85
80
75
70
65
60
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
Training Accuracy
Validation Accuracy

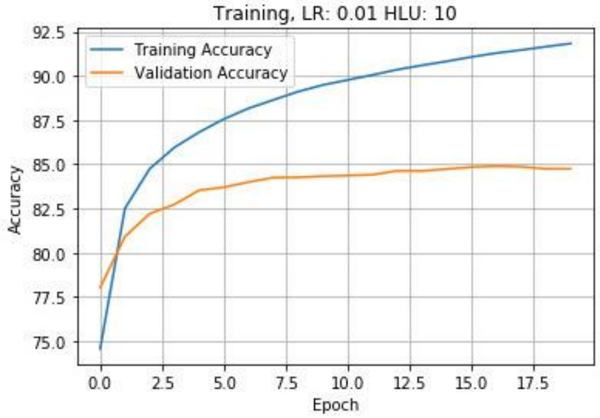

Training, LR: 0.001 HLU: 10
Loss
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
Training loss
Validation loss

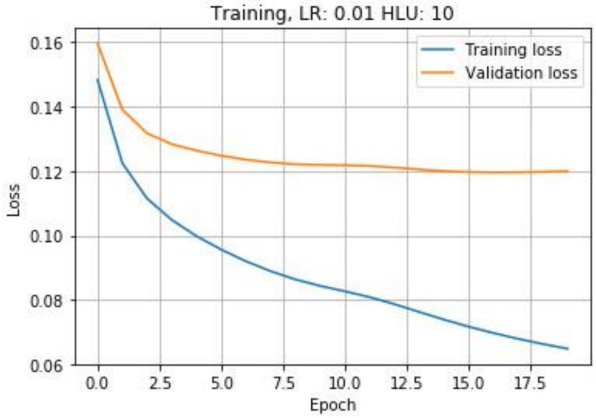

Training, LR: 0.001 HLU: 30
Accuracy
90
85
80
75
70
65
0
10
20
30
40
50
Epoch
Training Accuracy
Validation Accuracy

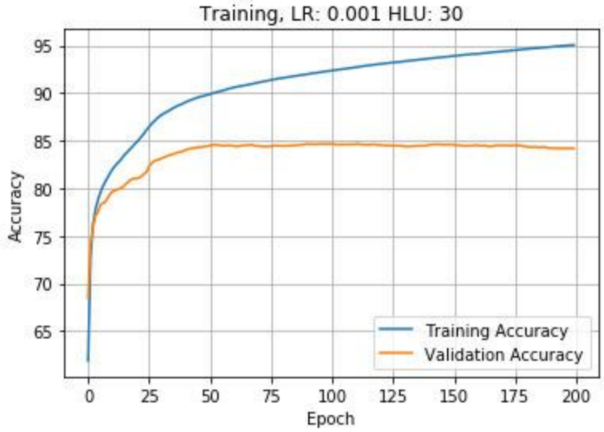

Training, LR: 0.001 HLU: 30
Loss
0.30
0.25
0.20
0.15
0.10
0
10
20
30
40
50
Epoch
Training loss
Validation loss

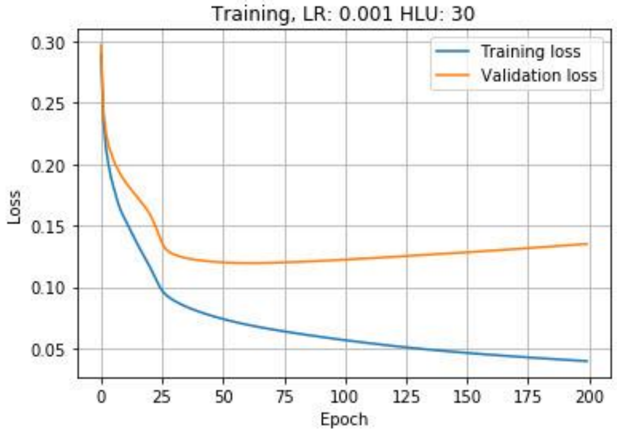

Training, LR: 0.01 HLU: 40
Training Accuracy
Validation Accuracy
Accuracy
Epoch
90
88
86
84
82
80
78
76
0
2
4
6
8
10
12
14

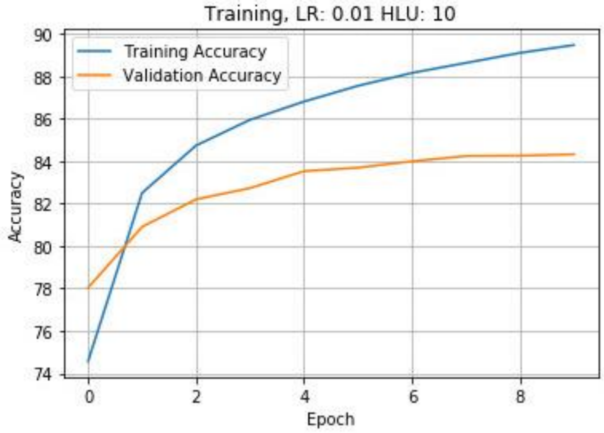

Training, LR: 0.01 HLU: 40
Training loss
Validation loss
Loss
Epoch
0.18
0.16
0.14
0.12
0.10
0
2
4
6
8
10
12
14

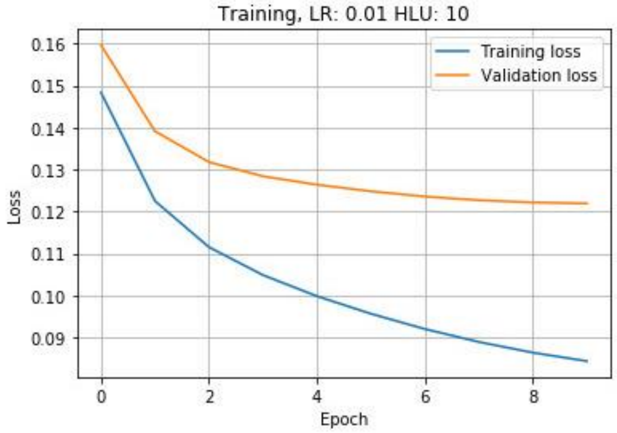

Training, LR: 0.01 HLU: 10
Training Accuracy
Validation Accuracy
Accuracy
Epoch
92.5
90.0
87.5
85.0
82.5
80.0
77.5
75.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

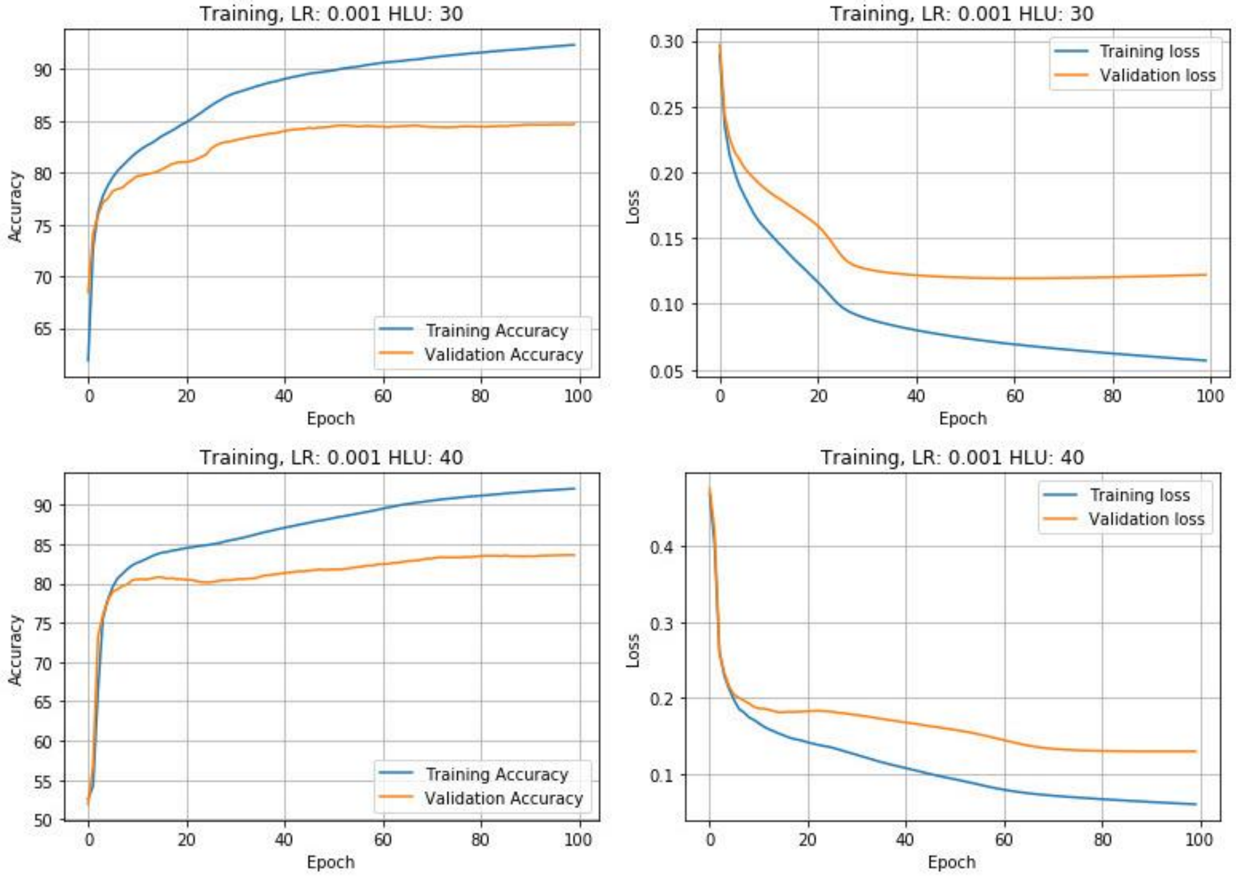

Training, LR: 0.01 HLU: 10
Training loss
Validation loss
Loss
Epoch
0.16
0.14
0.12
0.10
0.08
0.06
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training, LR: 0.001 HLU: 30
Training Accuracy
Validation Accuracy
Accuracy
Epoch
95
90
85
80
75
70
65
0
25
50
75
100
125
150
175
200


Training, LR: 0.001 HLU: 30
Training loss
Validation loss
Loss
Epoch
0.30
0.25
0.20
0.15
0.10
0.05
0
25
50
75
100
125
150
175
200


Training, LR: 0.01 HLU: 10
Training Accuracy
Validation Accuracy
Accuracy
Epoch
90
88
86
84
82
80
78
76
74
0
2
4
6
8


Training, LR: 0.01 HLU: 10
Training loss
Validation loss
Loss
Epoch
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
0
2
4
6
8



Precision can be seen as a measure of exactness or quality of these results, whereas recall is a measure of completeness or quantity. In simple terms, high precision means that this algorithm returned substantially more relevant results than irrelevant ones, while high recall means that it returned most of the relevant results. If a machine learning algorithm is good at recall, it doesn't mean that algorithm is good at precision. That's why we also need F1 score which is the (harmonic) mean of recall and precision to evaluate it. However, since F1-Score can be biased when our data is unbalanced, MCC-Score, will get us get the true accuracy for the machine learning model for when the dataset is unbalanced.

In the results tabulated, the MCC-Score is equal to the F1-Score since we had a balanced dataset for training, validation and testing of the sentiment analysis.

4.2. **Data Analysis**

A good model should have high but very close values for PRECISION, RECALL, F1-S, MCCS, TE-ACC, TR-ACC and VA-ACC. As per the results table, the one that fits the criteria most is the 5th and 19th model in the table. However, since the 19th model gives the highest of the model validation values, therefore, its hyperparameters are the most optimum values for the sentiment neural network that we have trained, since it has the most generality.

Those are: - **LR** = 0.01 - **EP** = 20 - **HLU** = 10

4.2.1. Load Trained Model

5. CONCLUSION AND RECOMMENDATION

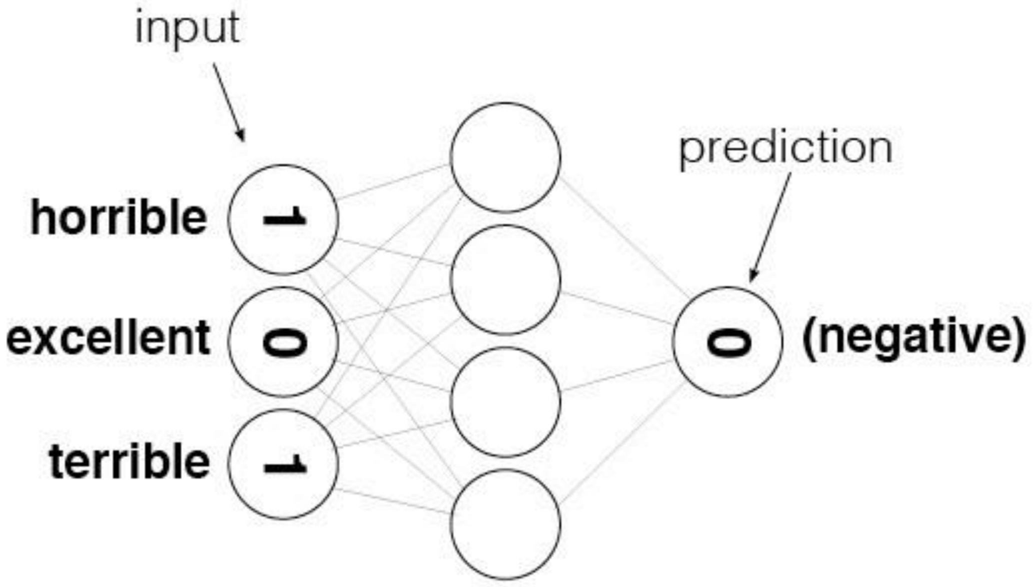
6. APPENDIX

6.1.1. Load Trained Model

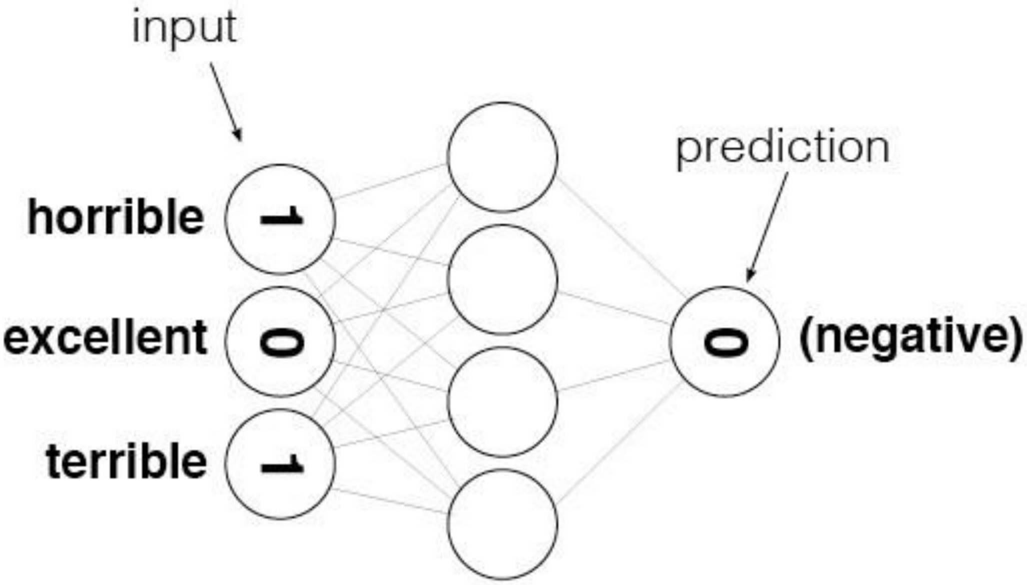
The following is a demonstration of how the trained model, with the above parameters performs on unseen text that has some sentiment.

```
In [1]:# Import needed libraries

 %matplotlib inline

%config InlineBackend.figure_format = 'retina'

import cPickle as cPickle


In [2]: %load toolkit/EfficientLowNoiseSentimentalNeuralNetwork.py

    Predictions Criteria

    if(prediction > 0.5):

        Sentiment = POSITIVE

    if(prediction < 0.5):

        Sentiment = NEGATIVE

In [3]: # Load the Trained Model from disk

model_trained_filepath = 'models/se_model_lr0.01epoch_20hlu_10.sav'

loaded_model = cPickle.load(open(model_trained_filepath, 'rb'))

# Test on some NEGATIVE statements

In [4]: loaded_model.predict("This product sucks!! I bought it and it proved disgusting. Do not buy.”)

Out[4]: array([ 0.00281025])

In [5]: loaded_model.predict("I hate how this product was badly packaged. It was delivered while
damaged”)

Out[5]: array([ 0.28054857])

In [6]: Image(filename='illustration/binary_negative.png')
```

```
Out[6]: Loading image…
```



```
# Test on some POSITIVE statements

In [8]: loaded_model.predict("I enjoyed it. It is a very awesome product")

Out[8]: array([ 0.95822964])

In [9]: loaded_model.predict("This product is excellent and I would definitely recommend it to others out there”)

Out[9]: array([ 0.65593228])

In [10]: Image(filename='illustration/binary_negative.png')

Out[10]: Loading image…
```


input
prediction
horrible
1
excellent
0
terrible
1
0
(negative)

7. REFERENCES

[1] Ian GoodFellow, Yoshua Bengio and Aaron Courville“Deep Learning” MIT Press, 2016

[2] Michael Nielsen, “Neural networks and Deep Learning” Determination Press, 2015.

[3] Andrew W. Task, “Grokking Deep Learning” Manning, 2016.

[4] Pang, Bo and Lee, Lilian, “A Sentimental Education: Sentiment Analysis using Subjectivity Summarization Based on Minimum Cuts”, In Proc. of ACL, 2004

[5] Pang, Bo and Lee, Lilian, “Opinion Mining and Sentiment Analysis”, Foundation and Trends, vol.2, no. 1-2, pp.1–135, Jul. 2008.

[6] Forman, George, “An Extensive Empirical Study of Feature Selection Metrics for Text Classification”, Journal of Machine Learning Research, cpt.3:pp.1289–1305, 2003

[7] Luciano Barbosa and Junlan Feng, “Robust Sentiment Detection on Twitter from Biased and Noisy Data”,Proceedings of the 23rd International Conference on Computational Linguistics: Posters, pp.36–44, 2010

[8] B. Klimt and Y. Yang, “The Enron corpus: A New Data Set for E-mail Classification Research”, In Proceedings of the European Conference on Machine Learning, pp.217–226, 2004.

[9] Shafi’i Muhammad Abdulhamid, Muhammad Shafie Abd Latiff, Haruna Chiroma, Oluwafemi Osho, Gaddafi Abdul-Salaam, Adamu I. Abubakar, and Tutut Herawan, “A Review on Mobile SMS Spam Filtering Techniques”, IEEE, May 2017

[10] Maureen Caudill, “Neural Network Primer: Part I”, AI Expert, Part I, Feb. 1989.

[11] Mitchell, T, “Machine Learning”, ISBN 0-07-042807-7, p.2 ,McGraw Hill, 1997.

[12] Wolpert, D.H., Macready, W.G., "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67, 1997.