Feedforward Neural Network and Its Application to Data Classification

CSCI-200 JUNIOR INDEPENDENT STUDY FINAL PAPER

Minhwa Lee

April 28, 2020

Abstract

Feedforward Neural Network (FNN) is one of the Artificial Neural Networks (ANNs) that possesses the simplest architecture in its algorithm. This paper aims to provide a reader general explanation of the FNN. In addition, the paper presents an essential algorithm for improving the performance of the FNN - backpropagation. Through reading a high level of explanation of backpropgation, the reader understands how the Multilayer Perceptron (MLP) - one type of the FNN - corrects the error generated by the execution of its algorithm. Lastly, the paper discusses the applications of the MLP with backpropagation algorithm to solving real-world problems, such as pattern classification tasks. Examining two case studies of the applications of the MLP, the paper therefore provides an insight into how the FNN as well as the ANNs have been utilized as a machine learning method in the real world.

Contents

Abstract **ii**

1 Introduction **1**

1.1 Outline of The Thesis 1

1.2 General Background of Feedforward Neural Network (FNN) 2

1.2.1 Definition of Artificial Neural Network (ANN) 2

1.2.2 Properties of Artificial Neural Networks (ANNs) 2

1.2.3 Benefits and Applications of Artificial Neural Networks 2

1.3 Supervised Learning 3

1.3.1 Definition 3

2 Feedforward Neural Network (FNN) **4**

2.1 History of Feedforward Neural Network 4

2.2 Understanding The Algorithm 5

2.2.1 Artificial Neuron 5

2.2.2 Perceptron 6

2.2.3 The Activation Function 6

2.3 Comprehensive Explanation of Feedforward Neural Network 8

2.3.1 Network Architectures of the Feedforward Neural Network 8

2.3.2 Multi-layer Perceptron (MLP) with Backpropagation 9

2.4 Benefits and Weakness 11

3 Application of Feedforward Neural Network (FNN) to Data classification **12**

3.1 Case Study I: Iris Classification 12

3.1.1 Training and Testing the Data set 13

3.1.2 Analysis of The Result 13

3.2 Case Study II: Prediction of Adolescents' High Risk of Suicide in South Korea 14

3.2.1 Training the Dataset 15

3.2.2 Analysis of The Results 15

4 Conclusion and Future Research 16

5 Appendix 17

5.1 Python Code for Iris Classification 17

5.2 R code for Data cleaning of the High-Risk of Suicide Data set in South Korea 21

5.3 Python Code for High Risk of Suicide among Adolescents in South Korea . . 22

Bibliography 26

Chapter 1

Introduction

1.1 Outline of The Thesis

Artificial Neural Network (ANN) is a research area in the field of computer science that has continuously shown exceptional results in data analysis and impacted the world. Therefore, the interest in researching neural networks has been widely increasing among the technology industries and academia. An ANN that shapes the basis of neural networks and other artificial intelligence models is called Feedforward Neural Network (FNN). As an essential algorithm in ANNs, the FNN has influenced the development of various kinds of ANNs, in terms of doing several machine learning tasks. Therefore, this paper aims to provide an introduction to the FNN, a simple type of ANNs. Note that the paper assumes no prior knowledge of deep learning or any machine learning techniques associated with ANN or even FNN. Only calculus concepts are needed to read the paper.

First, Chapter 1 explores the general background literature of the FNN as an overview of the main thesis. This chapter provides general definitions of the ANNs and supervised learning that the FNN is based on. Chapter 2 investigates the FNN in a detailed manner. We explore the background of the FNN by examining several key concepts. Along with the general explanation of the algorithm, we also examine a technique of the backpropagation with a detailed example of showing the process of this method.

In Chapter 3, we then apply the concepts of the FNN to real-world problems, especially data classification tasks. The first case study of the application of the FNN is the Iris classification provided from UCI Machine-learning, which is widely known for classifying iris species using the neural network. The second case study of the application focuses on the prediction of the high risk of suicide using a FNN. Therefore, we gain our insights into how the FNN works for classification tasks.

1.2 General Background of Feedforward Neural Network (FNN)

1.2.1 Definition of Artificial Neural Network (ANN)

The development of artificial neural networks starts from our recognition of the human brain's working system. The human brain is able to set up a rule of doing tasks through a number of experiences. So, the ANN is defined as an algorithm that mimics the way that the human brain solves a given problem through a repetition of doing certain tasks [3] - which we call the "learning process." In addition, artificial neurons in the ANNs are also interconnected with certain strengths - also called synaptic weights [3]. This weight is used to store knowledge acquired from the learning process [3].

1.2.2 Properties of Artificial Neural Networks (ANNs)

First, ANNs can generalize the results of data analysis through training process, even from a large size of data sets [3]. Second, the ANNs work on variables that have nonlinear relationships [3]. Third, the ANNs also update the synaptic weights by applying several training examples [3]. Through the random process of training with many repetitions, the ANNs minimizes the errors on the outputs. The benefit of employing this principle is that the network requires no prior assumption for the data. Specifically, in terms of pattern classification problems, ANNs provide useful background about the particular pattern that is to be selected as well as the accuracy of that decision. This can therefore advance the performance of the network in distinguishing the pattern of certain data provided. Lastly, since the neurons in the ANNs are interconnected, they are also affected by other neurons in the network, if changes are made in the input signals [3].

1.2.3 Benefits and Applications of Artificial Neural Networks

Due to the capability to model complex and nonlinear tasks, ANNs have been widely used in many fields. The main tasks that ANNs commonly perform are classification, prediction, clustering, and associating [2]. ANNs can classify and even remember certain features from either the predefined classes or no prior assumption. They can also predict the output from the given inputs in the dataset.

In addition, many industries have used this method in their business. For example, business analytics companies use it for fraud detection, credit card attrition, or stock market prediction [2, 6]. Medical corporations can analyze cancer using ANNs, or even transplant

process optimization [6]. Since the ANNs work based on the data or any given knowledge for the inputs, they are continuously extending the domains of their application to many disciplines.

1.3 Supervised Learning

The ANNs involve the learning process which enables the networks to adapt to changing environments through training the given inputs. Since the Feedforward Neural Network (FNN) utilizes a supervised learning algorithm for its learning process, the understanding of supervised learning is essential before moving forward to the FNN.

1.3.1 Definition

Supervised learning first requires some inputs for the construction of the network [1,7]. Then, the result of the learning process is used for computing the error from the actual solution which is already known. The synaptic weights are continuously corrected according to the magnitude of the deviation. One important property of supervised learning is that it already has a known answer for the inputs [1,7].

Each training example consists of a pair of inputs and the expected output value. During training, supervised learning receives an input and searches for the input's pattern that matches what is expected to be its outputs [1,7]. Then, the algorithm takes new input, and it classifies the class of the new input, based on their priorly trained results.

Chapter 2

Feedforward Neural Network (FNN)

In this chapter, we focus on the Feedforward neural network (FNN) in terms of its constituents, algorithm, benefits, and weaknesses. We refer to a simple FNN in the supervised learning to the ANN that does not have any hidden units in the network [1].

2.1 History of Feedforward Neural Network

A psychologist Frank Rosenblatt proposed an idea of perceptron in 1958 [8]. This improves the study of neurons, by modeling a simple input-output relationship [3, 9]. In short, Rosenblatt claimed that this neuron takes in inputs, applies a weighted sum and returns 0 if the result is below the threshold and 1 otherwise [3, 7–9]. His research is a remarkable feat in history in that the weights that the neuron system devised by Rosenblatt are learnt through successively passed inputs. It also minimizes the error of the accuracy of the model.

Despite the rapid development of neural networks in the 1950s and 1960s, the research around neural networks stagnated due to low functionality of computers in that time period. In 1969, Marvin Minsky and Seymour Papert, respectively a founder and the director of the MIT AI Lab, stated in their book *Perceptrons* that Rosenblatt's single perceptron-based algorithm would not be extended to multilayered neural networks, the improved version of the single perceptron [6].

After the success of extending to multi-layered neural networks in 1986, groups of researchers re-discovered an idea of the networks, called backpropagation, which was already invented in the 1970s. This algorithm had the most popularity among researchers at that time period, and it is even currently being used as a key algorithm for minimizing the error [3, 6, 7, 9].

2.2 Understanding The Algorithm

This section thoroughly discusses general concepts related to FNNs. Based on the fact that ANNs mimic the human brain, most of the background of the FNN attributes to understanding the general concepts of the ANN.

2.2.1 Artificial Neuron

An artificial neuron denotes a basic unit of a neural network that is fundamental to the operation of the network [3]. Figure 2.1 describes the model of a neuron, each of which finally shapes the entire structure of the ANN.

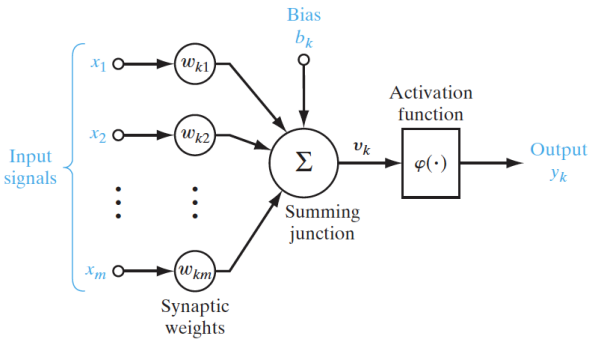


Figure 2.1: A model of neuron [3]

Figure 2.1 shows that the artificial neuron consists of three basic elements - synapses, adder, and an activation function. First, each synapses has a distinct weight w_{km} to be multiplied with the input signals x_m. An adder sums the input signals multiplied by the corresponding synaptic weight, as shown in Equation 2.1 below:

$$u_k = \sum_{j=1}^{m} (w_{kj})(x_j) \tag{2.1}$$

, where x_j is the input signals and w_{kj} is the respective synaptic weights of the neuron k.

Lastly, an activation function places a limit to the range of the output, thus having the output to be some finite value [3]. The bias b_k controls the net input of the activation function. Then we define the output of a neuron k as described in Equation 2.2 below:

$$y_k = \phi(u_k + b_k) = \phi(\sum_{j=1}^{m} (w_{kj})(x_j) + b_k) \tag{2.2}$$

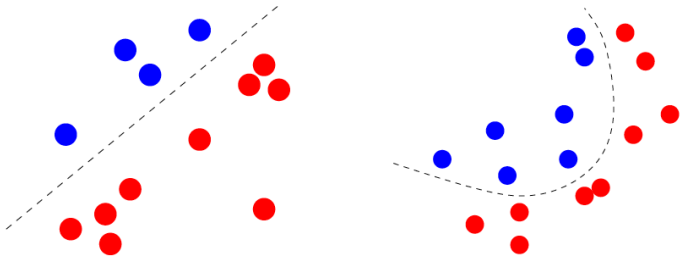
, where b_k is the bias, ϕ is the activation function, and y_k is the output of the neuron k [3].

2.2.2 Perceptron

Rosenblatt proposed the concept of a perceptron as the first model for supervised learning in 1958. This concept is almost the same as artificial neurons. A perceptron is, however, generally used for pattern classification tasks that are linearly separable [3]. Recall that the model of a neuron includes an activation function for prediction process. For a perceptron applied to binary pattern classification, the activation function ϕ is defined in Equation 2.3:

$$\phi(x) = \begin{cases} -1 & \text{if } x \in C_1 \\ 1 & \text{if } x \in C_2 \end{cases} \tag{2.3}$$

, where C_1 and C_2 represents the binary classes [3]. Next, as shown in Equation 2.2, the output $y_k = \phi(u_k + b_k)$ is used for classification as a result. For example, if the perceptron output y_k is -1, then the model assigns the input x_k to the class C_1 or to the class C_2 if y_k is 1. This also indicates that training examples are "linearly separable" [3]. In other words, there are two clearly-separated decision regions representing each class, as Figure 2.2 shows below.



(a) Linearly separable (b) Nonlinearly separable

Figure 2.2: An example of "Linearly Separable" Classification [1]

On the left plot in Figure 2.2, points with either red or blue points are separated by the line, sufficiently far from each other. However, on the right plot we observe that blue and red points on the different decision regions are close to each other, and we call this case "nonlinearly separable" [1].

2.2.3 The Activation Function

As stated earlier, the activation function determines the output of the neural network. Among various types of the activation function used, we introduce two popular activation functions that are commonly used in the training process.

The first type of the activation function is a threshold function. This function has binary

values - (0 and 1) or (−1 and 1). Equation 2.3 is an example of the threshold function, which is also shown in Figure 2.3.

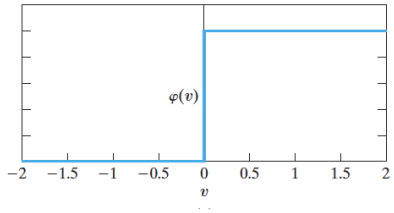


Figure 2.3: A threshold function [3]

Another popular activation function is a sigmoid function. It is an increasing function and has both linear and non-linear shapes. The first example of a sigmoid function is the logistic function. It is defined as shown in Equation 2.4 below:

$$\phi(x) = \frac{1}{1 + e^{(-x)}} \tag{2.4}$$

Another example of the sigmoid function is the hyperbolic tangent function $\phi(x) = \tanh x$, which is the shifted and scaled version of the logistic function ranging from -1 to 1 [3].

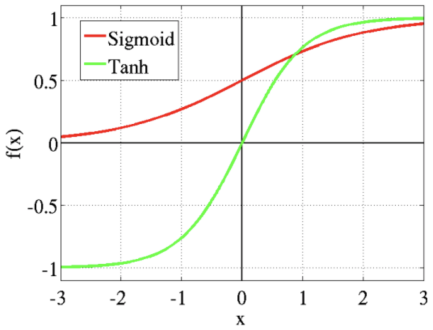


Figure 2.4: A Graph of Sigmoid Function

Unlike the threshold function, a sigmoid function has a range of continuous values from 0 to 1. This continuous property allows the differentiation of the sigmoid function.

2.3 Comprehensive Explanation of Feedforward Neural Network

2.3.1 Network Architectures of the Feedforward Neural Network

A feedforward neural network (FNN) is composed of neuron units that are linked to each other with directed edges, where the activation of a previous unit stimulates the activation of the forward unit but not vice versa [3]. This property of FNN enables the network to perform its algorithm in just one direction, from the input layer to the output layer.

The simplest type of FNNs is a single-layer FNN. This is composed of an input layer of "source units" and an output layer of neurons that are directly linked from the input layer [3]. However, a single-layer network only performs linear functions. This algorithm can therefore show poor performance on data sets that are not linearly separable [3].

The advanced type of a single-layer FNN is a multi-layer FNN. The distinct property of a multi-layer FNN is that it has one or more hidden layers which intervene between the input and output layer of the network [7]. The benefit of inserting hidden layers in the network is, the multi-layer FNN's capability of higher-order computations such as including nonlinear functions, so this can perform even under linearly inseparable settings, contrary to the single-layer FNN [3].

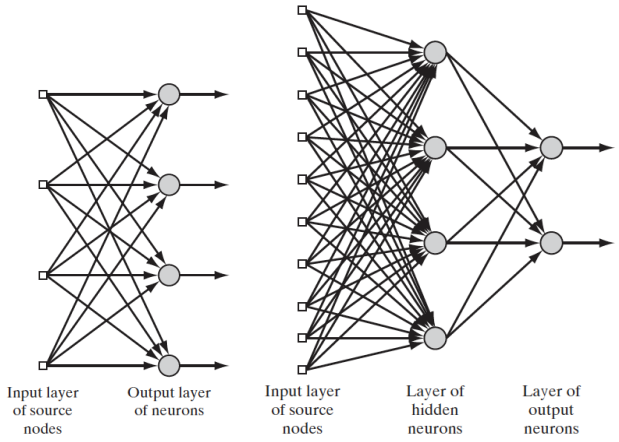


Figure 2.5: Single-layer FNN on the left, Multi-layer FNN on the right [3]

First, a source node in the input layer is linked to the neurons in the hidden layer, providing input vectors as a form of signals. Second, the output signal from the hidden layer is then connected to the layer of output neurons. Finally, the output signals of the output layer are the results of the entire network for the respective source nodes [3]. Therefore, it

is confirmed that a signal flow through the network moves in a forward direction, from left to right and layer by layer [3].

2.3.2 Multi-layer Perceptron (MLP) with Backpropagation

A multi-layer FNN is also known as a multi-layer Perceptron (MLP) in terms of the same constituents of its algorithm. In other words, the MLP also contains one or more hidden layers in its algorithm. However, this feature of a multi-layer perceptron has a deficiency in the algorithm. Due to the existence of hidden layers in the network, the MLP has to decide which activation pattern from the input layer should be delivered to the hidden layers [3]. In other words, the network requires more computational complexity in finding the optimal combination of the weights and the input neurons, as described in Equation 2.1.

A remedy for this weakness of the MLP is backpropagation. Assume there are a set of m training examples. Each time we input x_i into the network, the algorithm produces the corresponding computed output c_i from the training examples [1]. The goal of the training process here is to minimize the error between the target and the computed output, defined in Equation 2.5 below [1]:

$$E = \frac{1}{2}\sum_{i=1}^{m}(c_i - t_i)^2 \tag{2.5}$$

After training, we then test the minimized error function by using a new input example to identify whether this new one produces the same result as what the output through the network showed. Here, we use backpropagation to minimize the error E.

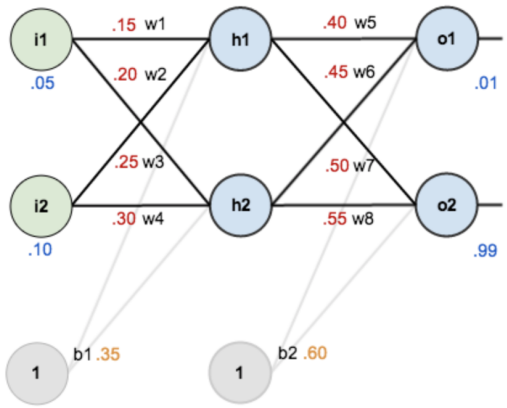


Figure 2.6: An example of the MLP with initialized weights [5]

1. Initialize weights Randomly

First, we randomly initialize weights of the network. The goal of the backpropagation is to update the weights in order to bring the error function E to a minimum. Figure 2.6 presents an example of the MLP with initialized weights. $i1$ and $i2$ are input nodes, $h1$ and $h2$ hidden units, $o1$ and $o2$ output units, and two 1 nodes are biases [5].

2. Forward propagation to obtain output values

We perform forward propagation through the network to get the output values. Based on Equation 2.1, 2.2, and 2.4, we compute the total net input for $h1$ and $h2$, under the assumption that the activation function is a logistic function.

3. Calculate the Total Error in Forward Propagation

Next, we calculate the total error generated in the forward propagation process, by using Equation 2.5 for the error function.

4. Backpropagation

Now we compute how much a change in weights affects the total error. This is calculated by the partial derivative (or gradient descent) of the Error E with respect to the weight w, expressed as $\frac{\partial E}{\partial w}$ [5]. This is same as $\frac{dE}{dw}$, so we use this expression in the whole section instead of $\frac{\partial E}{\partial w}$. Suppose that we determine to calculate how the total error is affected by a change in the weight w_1, we can describe the whole process in Equation 2.6:

$$\frac{dE_{total}}{dw_1} = \frac{dE_{total}}{do_{h1}} \times \frac{do_{h1}}{dnet_{h1}} \times \frac{dnet_{h1}}{dw_1} \tag{2.6}$$

, where o_{h1} represents the output from the hidden unit $h1$, net_{h1} the total net input for $h1$, and $w1$ is the weight connected from input $i1$ to $h1$. The interpretation of Equation 2.6 is that, we consider all the units in the forward layers that are affected by $w1$ [7]. In Figure 2.6, we observe that in the forward propagation procedure, $w1$ affects $h1$ whose output also affects $o1$ and $o2$ in the output layer through $w5$ and $w7$. Therefore, in terms of performing backpropagation, we move backward from the output layer to the input layer, considering which units in each layer are closely related to $w1$. Through this process, we calculate how much a change in $w1$ affects the total error defined by Equation 2.5.

5. Update the randomly initialized weight

Finally, we update the weight $w1$ by using Equation 2.7 described below [3, 5, 7]:

$$w_1 = w_1 - (\eta \times \frac{dE_{total}}{dw_1}) \tag{2.7}$$

, where η is a constant called learning rate. With the newly updated $w1$, we run through the entire process again to re-update the weight $w1$, thus minimizing the total error.

2.4 Benefits and Weakness

Since the FNN is the first and simplest form of ANNs, it is relatively easy to understand, especially for a single-layer FNN. However, a single-layer network only performs under linearly-separable data, which provoked the invention of multi-layer FNN (also called MLP).

We have also examined the MLP with the backpropagation method. This algorithm is capable of producing a satisfactory output under a non-linear setting [3, 7]. However, the backpropagation has some following deficiencies in its algorithm. First, a gradient descent model may not find the global minimum of the error function, rather seeking a local minimum. Also, as observed in the derivation of the backpropagation process, we need to know the activation functions in advance [3, 7].

Chapter 3

Application of Feedforward Neural Network (FNN) to Data classification

In Section 3, we discuss the application of the FNN to real-world problems, especially for data classification tasks. Data classification generally analyzes the pattern of a certain class that can be clearly observed from a combination of specific features that the class clearly shows among other classes. We now examine how accurately the model constructed with the FNN – to be specific, the multilayer perceptron with backpropagation – performs in classifying the patterns.

The main programming language that has been used for the software is Python with its machine-learning library called Scikit-learn in Tensorflow backend. Scikit-learn is an open-source machine learning library that provides efficient tools for the neural network. Also, a statistical software R has been used for reformatting the raw data and exporting it to a csv-format file. The library named 'dplyr' in R is used for data manipulation.

3.1 Case Study I: Iris Classification

The iris plant data set is highly known for the database that is mainly used for performing data classification tasks. This data set is obtained from UCI Machine Learning Repository. Consisting of 150 cases in total, the Iris data set contains three different classes of iris flower – each called 'Iris-Setosa,' 'Iris-Versicolor,' and 'Iris-Virginica.' Those three classes are represented in the column of a categorical variable named 'Species' in the dataset, as a dependent variable. The remaining four continuous attributes describes the sepal or petal length of the flower in centimeters. The predictive model of classification of the three species of iris is expected to discover patterns of each species from analyzing the sepal/petal length of the species, thus making a prediction of the class of each case in the data set. Note

that there are no missing values in the data set, so an accuracy of the prediction would be guaranteed to be higher and more trusted.

3.1.1 Training and Testing the Data set

First, we reformat the raw data set optimized to the MLP model. For the independent variables representing the sepal/petal length of a flower, we perform normalization in those columns. In other words, we normalize those independent variables so that all the data points in those columns are converted to be in the appropriate range [10]. For the dependent variable 'Species,' we convert each of three classes into three different numeric values, since the neural network requires numeric values.

Next, we separate the modified data set into either a training set or testing set, each consisting of 70% and 30% respectively. Then, we import a built-in function named MLP-Classififer from Scikit-learn library to build a model with the MLP with backpropagation. For the setting of the model, we have selected the stochastic gradient descent ('sgd') as the activation function, 10^{-3} for tolerance, 3000 for maximum number of iterations. For a feature of hidden layers, we have chosen one hidden layer with 10 hidden units. We finally predict the class of each instance in the testing set, by using this training process with the MLP model.

3.1.2 Analysis of The Result

Table 3.1 shows the result that the MLP model has performed for classifying the iris species. All 14 instances of Setosa and all 13 instances of Virginica are correctly classified. However, 17 instances of Versicolor are correctly classified out of 18 cases. Therefore, the percentage of accuracy of the classification by the MLP model is $\frac{41}{45} = 91.11\%$.

Table 3.1: Result of Classification of Iris Plant with the MLP with backpropagation

Iris Class	Total	Correctly Classified	Incorrectly Classified
Setosa	14	14	0
Versicolor	18	14	4
Virginica	13	13	0

Figure 3.1 describes the accuracy scores of the MLP model differed by the number of hidden units in one hidden layer. As observed, accuracy scores tend to become higher as the number of hidden units is greater or equal to 7. Accuracy scores range from 40% to 97.77%.

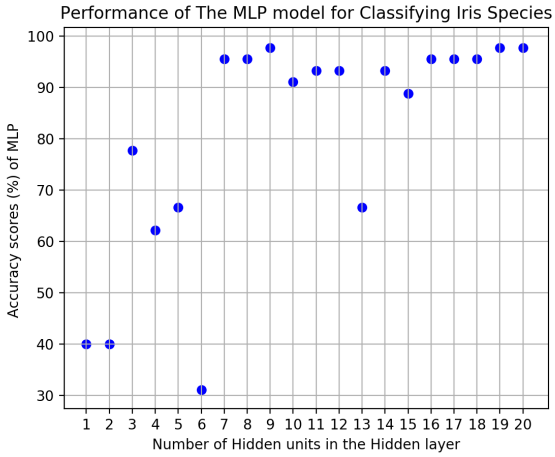


Figure 3.1: Performance of MLP Testing on Iris Dataset

3.2 Case Study II: Prediction of Adolescents' High Risk of Suicide in South Korea

This case study has a purpose of predicting and classifying the high risk of suicide in Korean Adolescents, using the MLP with backpropagation. The data set used in this study is extracted from the Korean Young Risk Behavior Web-based Survey (KYRBWS) in 2019 [4]. The data set is then reformatted to having a total of 35,536 responses. The dependent variable is a response to the following question: Did you ever consider suicide in the past 12 months? Also, independent variables used for the MLP model are presented in the Table 3.2.

Table 3.2: A list of Independent Variables

Category	Name
Socio-demographic factor	sex, academic achievement, family socioecnomic status
Health-related lifestyle	smoking, alcohol, substance use, sexual experience, physical activity frequency
Mental health & Violence	sadness, stress, weight, self-rated physical appearance, sleep satisfaction, school violence experience, school injury experience

3.2.1 Training the Dataset

First, we extract a sample from the data set to have only 1,777 cases. Using the same process as Section 3.1.1, we perform normalization in the columns of independent variables. For the dependent variable 'M_SUI_CON,' we also convert two responses - No and Yes - into two different numeric values - 1 and 2, respectively.

Likewise, we partition the sample into training set and testing set, each consisting of 70% and 30% respectively. The training set has a total of 1244 cases and the testing set consists of 533 cases. Next, using MLPClassififer in Scikit-learn library, we build a prediction model for the sample. Under the same setting that we have already used in section 3.1.1, we finally predict the class of each instance in the testing set.

3.2.2 Analysis of The Results

Table 3.3 shows the result that the MLP model has performed on the testing set for predicting high risk of suicide. The MLP model correctly classified a response 'yes' as 444 out of 448 instances. However, only 10 cases of the response 'no' are correctly classified out of 86 cases. Therefore, the percentage of accuracy of the classification by the MLP model is $\frac{454}{534} = 85.02\%$.

Table 3.3: Result of Prediction of Risk of Suicide with the MLP with backpropagation

Response	Total	Correctly Classified	Incorrectly Classified
Yes	448	444	4
No	86	10	76

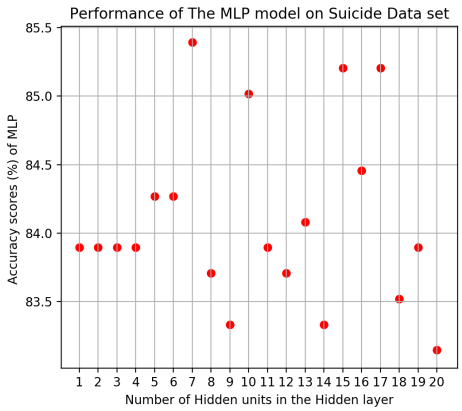


Figure 3.2: Performance of MLP Testing on the Suicide Dataset

Figure 3.2 describes how the MLP model accurately classified the responses using a different number of hidden units in one hidden layer. As observed, accuracy scores are similar to each other, which ranges from 83.15% to 85.39%.

Chapter 4

Conclusion and Future Research

As we have observed in Section 3, Feedforward Neural Network (FNN) has shown good performance in pattern classifications with a relatively easy algorithm, which confirms the great usability of the FNN to solving real-world problems. Also, the advancement of the FNN enables the researchers to actively participate in the development of more complex and efficient algorithms of the ANNs in the past decades, even furthering today's deep learning as well as artificial intelligence. Future research would be investigating other categories of the ANNs, such as Recurrent Neural Network (RNN), Convolutional Nerual Network (CNN), or Support Vector Machine (SVM). This research then discusses how those algorithms have been utilized in the field of data mining. Another possible area for future work is comparing the performance of the ANNs with the existing machine learning methods. Those further work, therefore, would provide us an insight into the power of the ANNs.

Chapter 5

Appendix

5.1 Python Code for Iris Classification

```
'''
// Name: Minhwa Lee
// Assignment: CS200 Junior IS Software
// Title: Software for Iris classification
// Course: CS 200
// Semester: Spring 2020
// Instructor: D. Byrnes
// Date: 04/25/2020
// Sources consulted: 'https://github.com/ashkanmradi/MLP-classifier/blob/master/main.py', 'pandas' and 'scikit-learn' library documentation website
https://pandas.pydata.org/docs/reference/index.html,https://scikit-learn.org/stable/modules/classes.html
// Program description: This program is to execute data classification of iris plants using multilayer perceptron with propagation.
It is written on Python Scikit-learn library.
// Known bugs: I've used 'warnings' to ignore all the unimportant warnings during running the program.
// Creativity: Except for for-loop codes that make a plot, all remaining codes are written by myself.


// Instructions:
1. Before running the program, you should install 'pandas' 'sklearn' and 'matplotlib' libraries first.
2. Then run the program then you will see the accuracy information as well as the plot.
```

```
'''

# Modules used for building the MLP model
import warnings
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt


if __name__ == "__main__":

    warnings.filterwarnings(action='ignore') # Ignore warnings

    # Import iris data set directly from the UCI Machine learning website
    iris_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

    # Assign column names to the data set
    names = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species']

    # Read dataset to pandas dataframe
    data = pd.read_csv(iris_url, names=names)

    # Place all independent variables in X and a dependent variable in Y
    X = data[
        ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
    Y = data[['Species']]

    # Normalize the data set
    norm_data = X.apply(lambda x: (x - x.min()) / (x.max() - x.min()))

    # Convert text to numeric values for NN performance
    target_col = Y.replace(
        ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], [0, 1, 2])

    df = pd.concat([norm_data, target_col], axis=1)
    # df is the modified dataset that we are going to apply MLP to.

    ## Data Separation  (Trainset vs Testset)
```

```
60
61      X_df = df[
62          ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'
    ]]
63      Y_df = df.Species
64
65      # Separate the entire data set into train set and test set, each
    taking up 70% and 30%.
66
67      train, test = train_test_split(df, test_size=0.3, random_state=1)
68
69      trainX = train[
70          ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'
    ]]
71      trainY = train.Species
72
73      testX = test[
74          ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'
    ]]
75      testY = test.Species
76
77      # Training and Testing
78
79      clf = MLPClassifier(solver='sgd', alpha=1e-3, max_iter=3000,
80                          hidden_layer_sizes=(10,), random_state=1)
81
82      clf.fit(trainX, trainY)
83
84      # Make a prediction and Check the accuracy of the MLP model
85
86      prediction = clf.predict(testX)
87
88      correct_setosa = 0
89      correct_versicolor = 0
90      correct_virginica = 0
91
92      for i in range(len(prediction)):
93          if prediction[i] == 0 and testY.values[i] == 0:
94              correct_setosa += 1 # If correctly classify iris-setosa, then
    increment by 1
95          elif prediction[i] == 1 and testY.values[i] == 1:
96              correct_versicolor += 1 # If correctly classify iris-
    versicolor, then increment by 1
```

```
97          elif prediction[i] == 2 and testY.values[i] == 2:
98              correct_virginica += 1 # If correctly classify iris-virginica, then increment by 1
99
100     prediction = list(prediction)
101     list_test_y = list(testY.values)
102
103     print("Correctly classified Setosa : ", correct_setosa)
104     print("Incorrectly classified Setosa: ", list_test_y.count(0) - correct_setosa)
105     print("Correctly classified Versicolor: ", correct_versicolor)
106     print("Incorrectly classified Veriscolor: ", list_test_y.count(1) - correct_versicolor)
107     print("Correctly classified Virginica: ", correct_virginica)
108     print("Incorrectly classified Virginica: ", list_test_y.count(2) - correct_virginica)
109
110     print('The accuracy of the MLP is:',
111           accuracy_score(testY, prediction) * 100)
112
113     # Make a plot for mean accuracy score vs number of hidden units in the hidden layer
114
115     lst_accuracy_plt = [] # A list representing accuracy of the MLP with
116     # each different number of hidden units
117
118     plt.figure(figsize=(5,5))
119     plt.grid(True)
120     axe = plt.axes()
121
122     for num_hidden_units in range(1, 21): # Set a maximum hidden units to 20
123         clf_hidden = MLPClassifier(solver='sgd', alpha=1e-3, max_iter=3000, hidden_layer_sizes=(num_hidden_units,), random_state=1)
124         clf_hidden.fit(trainX, trainY) # training process
125         clf_prediction = clf_hidden.predict(testX) # Testing process
126         clf_hidden_result = accuracy_score(testY, clf_prediction) # Compute Accuracy rate
127         lst_accuracy_plt.append(clf_hidden_result*100)
128         plt.scatter(num_hidden_units, clf_hidden_result * 100, c='blue')
129         axe.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
130         plt.xlabel('Number of Hidden units in the Hidden layer')
```

```
        plt.ylabel('Accuracy scores (%) of MLP ')
        plt.title('Performance of The MLP model for Classifying Iris Species')

    plt.show()
```

Listing 5.1: Python Code for Iris Classification

5.2 R code for Data cleaning of the High-Risk of Suicide Data set in South Korea

```

# Name: Minhwa Lee
# Assignment: CS200 Junior IS Software
# Title: Software for Data Cleaning for Suicide Data set
# Course: CS 200
# Semester: Spring 2020
# Instructor: D. Byrnes
# Date: 04/25/2020
# Sources consulted: 'https://github.com/ashkanmradi/MLP-classifier/blob/master/main.py', 'pandas' and 'scikit-learn' library documentation website
# https://pandas.pydata.org/docs/reference/index.html,https://scikit-learn.org/stable/modules/classes.html
# Program description: This program is to execute data classification of iris plants using multilayer perceptron with propagation.
#                      It is written on Python Scikit-learn library.
# Known bugs: I've used 'warnings' to ignore all the unimportant warnings during running the program.
# Creativity: Except for for-loop codes that make a plot, all remaining codes are written by myself.
# Instructions: Just run the program then you will see the accuracy information as well as the plot.


# 1. Import dplyr library and Open raw data

library(dplyr)

kyrbs2019_edit <- read.csv("~/Desktop/JuniorIS Software/kyrbs2019_edit.csv")
dat <- kyrbs2019_edit # dat is a copy of the original data set.
```

```
24
25 # 2. Data Cleaning and Renaming
26
27 # Parents educational attainment
28 dat <- dat[dat$E_EDU_F != 8888, ] # Remove un-related response for father
       column (non-response)
29 dat <- dat[dat$E_EDU_F != 9999, ] # (Single-mother family)
30
31 dat <- dat[dat$E_EDU_M != 8888, ] # Remove un-related response for mother
       column (non-response)
32 dat <- dat[dat$E_EDU_M != 9999, ] # (single-father family)
33
34 # Sexual Experience
35
36 dat <- dat[dat$S_SI != 9999, ] # Remove unrelated response ('n/a')
37
38 # School Violence Experience
39
40 dat$V_TRT[dat$V_TRT %in% c(2,3,4,5,6,7)] <- 2 # For making a binary
       response, we convert all 2- 7 values to 2 for saying 'yes'
41                                                 # For a response 'no' we
       keep 1.
42
43 # Physical Activity Frequence
44
45 dat$PA_VIG[dat$PA_VIG %in% c(1,2,3)] <- 1 # For those who are not
       frequently working out
46 dat$PA_VIG[dat$PA_VIG %in% c(4,5,6)] <- 2 # For those who work out
       frequently
47
48 # 4. Data Republishing with the newly updated dataframe
49 write.csv(dat, 'kyrbs2019_edit_1.csv')
```

Listing 5.2: R code for Data Cleaning of Suicide Data set

5.3 Python Code for High Risk of Suicide among Adolescents in South Korea

```
1 '''
2 // Name: Minhwa Lee
3 // Assignment: CS200 Junior IS Software
4 // Title: Software for Predicting High Risk of Suicide among Adolescents
       in South Korea
```

```
// Course: CS 200
// Semester: Spring 2020
// Instructor: D. Byrnes
// Date: 04/25/2020
// Sources consulted: 'https://github.com/ashkanmradi/MLP-classifier/blob/master/main.py', 'pandas' and 'scikit-learn' library documentation website
https://pandas.pydata.org/docs/reference/index.html,https://scikit-learn.org/stable/modules/classes.html
// Program description: This program is to execute data classification/prediction of high risk of suicide using multilayer perceptron with propagation.
It is written on Python Scikit-learn library.
// Known bugs: I've used 'warnings' to ignore all the unimportant warnings during running the program.
// Creativity: Except for for-loop codes that make a plot, all remaining codes are written by myself.


// Instructions:
1. Before running the program, you should install 'pandas' 'sklearn' and 'matplotlib' libraries first.
2. Then run the program then you will see the accuracy information as well as the plot.
'''


import warnings
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import os


if __name__ == "__main__":

    warnings.filterwarnings(action='ignore') # Ignore warnings

    # Import csv suicide data set from local directory
    os.listdir(os.getcwd())
    data = pd.read_csv("kyrbs2019_edit.csv", decimal=',')
```

```
39
40    data = pd.DataFrame(data) # Make an imported csv file a Dataframe that
      are usable in pandas library
41
42    # Generate a sample with only 5% of its original data size.
43    data = data.sample(frac=0.05, random_state=1)
44
45    ## Data Cleaning
46
47    # Reconstruct a data set with only selected features (independent
      variables vs dependent variable)
48    dat = data [['SEX', 'E_EDU_F', 'E_EDU_F', 'E_S_RCRD', 'E_SES', 'TC_LT'
      , 'AC_LT', 'DR_EXP', 'PA_VIG', 'S_SI', 'M_SAD', 'M_STR',
49    'PR_BI', 'PR_HT', 'M_SLP_EN', 'I_SCH_TRT', 'V_TRT', 'M_SUI_CON']].
      astype('int64')
50
51    # Place all independent variables in X and a dependent variable in Y
52    X = dat [['SEX', 'E_EDU_F', 'E_EDU_F', 'E_S_RCRD', 'E_SES', 'TC_LT',
53              'AC_LT', 'DR_EXP', 'PA_VIG', 'S_SI', 'M_SAD', 'M_STR',
54              'PR_BI', 'PR_HT', 'M_SLP_EN', 'I_SCH_TRT', 'V_TRT']]
55
56    # A Dependent variable
57    Y = dat[['M_SUI_CON']] # Dependent Variable - Suicide Attempt
58
59    # Data Normalization for independent variables set X
60    norm_data = X.apply(lambda x: (x-x.min()) / (x.max() - x.min()))
61
62    # df is the modified dataset that we are going to apply MLP to.
63    df = pd.concat([norm_data, Y], axis=1)
64
65    # Data Separation
66
67    # Separate the entire data set into train set and test set, each
      taking up 70% and 30%.
68    train, test = train_test_split(df, test_size=0.3, random_state=1)
69
70    trainX = train[['SEX', 'E_EDU_F', 'E_EDU_F', 'E_S_RCRD', 'E_SES', '
      TC_LT', 'AC_LT', 'DR_EXP', 'PA_VIG', 'S_SI', 'M_SAD', 'M_STR',
71    'PR_BI', 'PR_HT', 'M_SLP_EN', 'I_SCH_TRT', 'V_TRT']]
72
73    trainY = train[['M_SUI_CON']]
74
75    testX = test[['SEX', 'E_EDU_F', 'E_EDU_F', 'E_S_RCRD', 'E_SES', 'TC_LT
```

```
    ', 'AC_LT', 'DR_EXP', 'PA_VIG', 'S_SI', 'M_SAD', 'M_STR',
    'PR_BI', 'PR_HT', 'M_SLP_EN', 'I_SCH_TRT', 'V_TRT']]

    testY = test[['M_SUI_CON']]

    # Training and Testing

    clf = MLPClassifier(solver='sgd', alpha=1e-3, max_iter=3000,
                        hidden_layer_sizes=(10,), random_state=1)

    clf.fit(trainX, trainY)

    # Make a prediction and Check the accuracy of the MLP model
    prediction = clf.predict(testX)

    correct_1 = 0
    correct_2 = 0
    for i in range(len(prediction)):
        if prediction[i] == 1 and testY.values[i] == 1:
            correct_1 += 1 # If correctly classify 'no' then increment by
    1
        elif prediction[i] == 2 and testY.values[i] == 2:
            correct_2 += 1 # if correctly classify 'yes' then increment by
     1

    prediction = list(prediction)
    total_test_y = list(testY.values)

    print("Correctly classified No: ", correct_1)
    print("Incorrectly classified No: ", total_test_y.count(1) - correct_1
    )
    print("Correctly classified Yes: ", correct_2)
    print("Incorrectly classified Yes: ", total_test_y.count(2) -
    correct_2)

    print('The accuracy of the MLP is:', accuracy_score(testY, prediction)
     * 100)

    # Make a plot for mean accuracy score vs number of hidden units in the
     hidden layer

    lst_accuracy = []

```

```
112     plt.figure(figsize=(5,5))
113     plt.grid(True)
114 
115     axe = plt.axes()
116 
117     for num_hidden_units in range(1, 21): # Set a maximum hidden units to
    20
118         clf_hidden = MLPClassifier(solver='sgd', alpha=1e-3, max_iter
    =3000,
119                                    hidden_layer_sizes=(num_hidden_units,),
120                                    random_state=1)
121         clf_hidden.fit(trainX, trainY) # training process
122         clf_prediction = clf_hidden.predict(testX) # Testing process
123         clf_hidden_result = accuracy_score(testY, clf_prediction) #
    Compute Accuracy rate
124         lst_accuracy.append(clf_hidden_result*100)
125         plt.scatter(num_hidden_units, clf_hidden_result * 100, c='red')
126         axe.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
     16, 17, 18, 19, 20])
127         plt.xlabel('Number of Hidden units in the Hidden layer')
128         plt.ylabel('Accuracy scores (%) of MLP ')
129         plt.title('Performance of The MLP model on Suicide Data set')
130 
131     print(lst_accuracy)
132     plt.show()
```

Listing 5.3: Python Code for Suicide Prediction

Bibliography

[1] Kyunghyun Cho. 2014. *Foundations and Advances in Deep Learning*. Ph.D. Dissertation. Aalto University; Aalto-yliopisto, Espoo, Finland.

[2] Navdeep Singh Gill. 2019. Artificial Neural Network Applications and Algorithms. Retrieved April 23, 2020 from `https://www.xenonstack.com/blog/artificial-neural-network-applications/`

[3] Simon Haykin. 2004. *Neural Networks and Learning Machines* (3rd.ed). Pearson Prentice Hall.

[4] Jung Jun Su, Park Sung Jin, Kim Eun Young, Na Kyoung-Sae, Kim Young Jae, and Kim Kwanggi. 2019. Prediction models for high risk of suicide in Korean adolescents using machine learning techniques. *PloS One* 14, 6 (Jun. 2019), 1-12. DOI: https://doi.org/10.1371/journal.pone.0217639

[5] Matt Mazur. 2015. A Step by Step Backpropagation Example. Retrieved April 24, 2020 from `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`

[6] Eric Robert. 2000. Neural Network. Retrieved April 25, 2020 from `https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/index.html`

[7] Raul Rojas. 1996. *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, New York. DOI: https://doi.org/10.1007/978-3-642-61068-4

[8] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65 (1958), 386-408. DOI: https://doi.org/10.1037/h0042519.

[9] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85-117. DOI: https://doi.org/10.1016/j.neunet.2014.09.003

[10] Madhusmita Swain, Sanjit Dash, Sweta Dash, and Ayeskanta Mohapatra. 2012. An Approach for IRIS Plant Classification Using Neural Network. *International Journal on Soft Computing* 3 (Mar. 2012), 79-89.