Predictions of Human Activities and Postural Transitions based on Accelerometer and Gyroscope of the Smartphone Devices

By

Rujia Yuan

A Senior Thesis Submitted to the Faculty of the

DEPARTMENT OF MATHEMATICAL STATISTICS

at WAKE FOREST UNIVERSITY

May, 2022

Winston-Salem, North Carolina

Supervised By: Dr. Sneha Jadhav

Acknowledgement

I would like to give my big thanks and to express my great appreciation to my senior thesis advisor, Dr. Jadhav. I feel so grateful and fortunate for having her as my supervisor, who is always there for help, support, and encouragement during my senior thesis research work. As a student knowing nothing for neural networks in the beginning, she is so considerate and sweet for guiding me into this new world.

Abstract

Human activity recognition (HAR) can offer accurate feedbacks about people's health status, and therefore it could help evaluate citizens' overall healthy conditions. In the previous studies, the researchers presented a new dataset that had been created using inertial data from smartphone accelerometers and gyroscopes to target the recognition of different human activities. The dataset was collected from a group of 30 volunteers with the ages between 19 and 48, who performed a protocol of activities dividing into 6 main activities and 6 postural transitions. The researchers utilized a multi-class Support Vector Machine (SVM) classifier on this dataset and got convincing results to show that the use of smartphones to collect data and to perform HAR is feasible and more unobtrusive and less invasive than other special purpose solutions like wearable sensors. In my project, I utilize a neural network model to analyze the dataset and do predictions for human activities and postural transitions based on 561 different features. The reasons for utilizing a neural network model are that it can learn and model the complicated nonlinear relationships between the inputs and outputs, and it can learn and improve continuously, which make this method superior. I also construct a multinomial logistic regression model and a multi-class SVM model to make comparisons. With approximately 70% of the data as training data and remaining 30% as testing data, I get the results as the neural network shows a prediction accuracy of 93.4%, while other twos get 88.1% and 84.3%.

1. Introduction

1.1 Basic Background

Human activity recognition (HAR) aims to provide information on human physical activity and to detect simple or complex actions in a real-world setting. In HAR, various human activities such as walking, running, playing soccer, and walking upstairs are recognized. As a topic with increasing heat, HAR has gradually become mature for use and has been adopted by more and more scholars, as it provides personalized support for many different fields of study such as medicine, human-computer interaction, sociology. Specifically, it can offer accurate feedbacks about people's health status, and therefore it could help evaluate citizens' overall healthy conditions. In the article *A Public Domain Dataset for Human Activity Recognition Using Smartphones*, Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz worked together to present a new dataset that has been created using inertial data from smartphone accelerometers and gyroscopes to target the recognition of different human activities. The dataset was collected from a group of 30 volunteers with the ages between 19 and 48, who performed a protocol of activities dividing into 6 main activities and 6 postural transitions: 1. Walking, 2. Walking upstairs, 3. Walking downstairs, 4. Sitting, 5. Standing, 6. Laying, 7. Stand to sit, 8. Sit to stand, 9. Sit to lie, 10. Lie to sit, 11. Stand to lie, 12. Lie to stand. They utilized a multi class Support Vector Machine (SVM) classifier on this dataset and got convincing results to show the use of

smartphones, in addition to be more unobtrusive and less invasive than other special purpose solutions like wearable sensors, is a feasible way to work for effectively performing HAR. Therefore, based on this dataset, which has been proved as reliable and trustable, my research topic is to make predictions of human activities and postural transitions based on accelerometer and gyroscope of the smartphone devices.

1.2 Dataset Background

The data were collected from a group of 30 volunteers within the ages from 19 to 30. There were 10929 instances, with 7767 (approximately 70%) as training data and 3162 (30%) as testing data. While all of the volunteers were moving and displaying the 6 activities and 6 postural transitions mentioned above, they were requested to wear smartphones on the waist to capture the 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz using the embedded accelerometer and gyroscope of the device. The features selected for this database come from the accelerometer and gyroscope 3-axial (x-z-y axials) raw signals tAcc-XYZ and tGyro-XYZ. Then they were filtered using a median filter and a 3rd order low pass Butterworth filter with a corner frequency of 20 Hz to remove noise. Similarly, the acceleration signal was then separated into body and gravity acceleration signals (tBodyAcc-XYZ and tGravityAcc-XYZ) using another low pass Butterworth filter with a corner frequency of 0.3 Hz. Subsequently, the body linear acceleration and angular velocity were derived in time to obtain Jerk signals (tBodyAccJerk-XYZ and tBodyGyroJerk-XYZ). Also,

the magnitude of these three-dimensional signals were calculated using the Euclidean norm (tBodyAccMag, tGravityAccMag, tBodyAccJerkMag, tBodyGyroMag, tBodyGyroJerkMag). Finally, a Fast Fourier Transform (FFT) was applied to some of these signals producing fBodyAcc-XYZ, fBodyAccJerk-XYZ, fBodyGyro-XYZ, fBodyAccJerkMag, fBodyGyroMag, fBodyGyroJerkMag. To better analyze the data, each variable mentioned above was estimated by its mean value, standard deviation, median absolute deviation, largest value in array, smallest value in array, signal magnitude area, energy measure, interquartile range, signal entropy, autoregression coefficients with Burg order equal to 4, correlation coefficient between two signals, index of the frequency component with largest magnitude, weighted average of the frequency components to obtain a mean frequency, skewness of the frequency domain signal, kurtosis of the frequency domain signal, energy of a frequency interval within the 64 bins of the FFT of each window, and angle between to vectors. Following above, the data set ended up having 561 features to make analysis.

2. Methods

2.1 Basic Background of Neural Networks

Deep learning is one subfield of machine learning, while neural networks are the central of its algorithms – Neural networks are a form of machine learning in which a computer learns to carry out a piece of work by being trained on examples. Neural network method is

effective and worthy utilizing, since it has a lot of advantages, which are superior compared to other methods. Some pros are that the neural networks can learn and model the complicated nonlinear relationships between the inputs and outputs, reveal the hidden relationships, patterns, and predictions behind the model, and they can continuously learn and improve. Despite neural networks do have some disadvantages – such as the needs of a huge amount of data and the large cost of computations – they are still preferred algorithms to use while facing complex problems.

2.2 Core Elements

When we go in the neural network more deeply, we begin to think about its core components – layers, models, loss functions, and optimizers. Layers are the fundamental data structure in neural networks, while models are networks of layers. These two things combine the neural network architecture. We also need the last two – loss functions and optimizers – to help us polish and perfect the neural networks. We have to carefully choose the objective functions (loss functions,) because the right loss functions minimize the loss in the trainings and measure success for our tasks. And optimizer, based on the loss function, helps to determine how our neural networks will be updated. All of the four elemental components constitute our final achievement – an appropriate neural network.

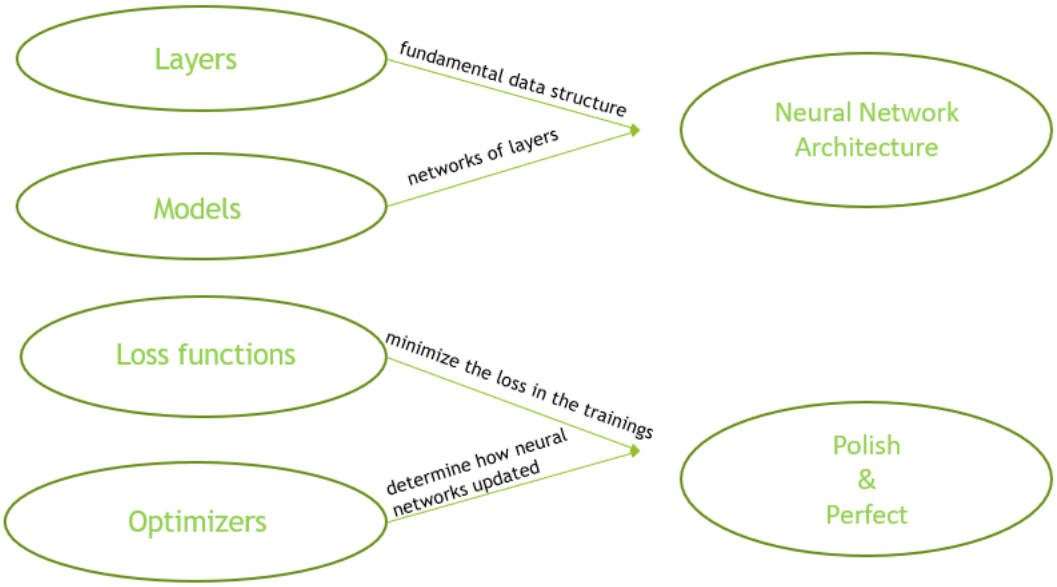


Figure1.1

2.3 Specific Details

Let us first take a look at a simple neural network with only one hidden layer. We can divide the layers into three parts –

1) an input layer, in which we get the input values ($a^{[0]}$), where a stands for activations, refer to the values that different layers of neural network are passing on to the subsequent layers.

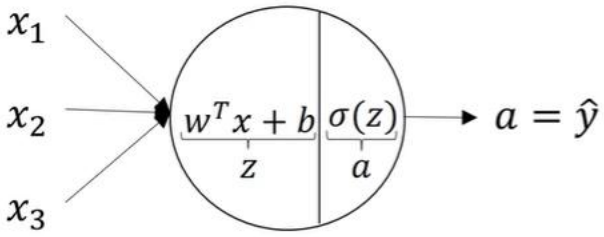
2) a hidden layer, where we generate some sets of activations, represented by two parts:

$$\boldsymbol{z^{[1]} = w^{[1]T} x + b^{[1]}}$$

$$\boldsymbol{a^{[1]} = (\sigma z^{[1]})}$$

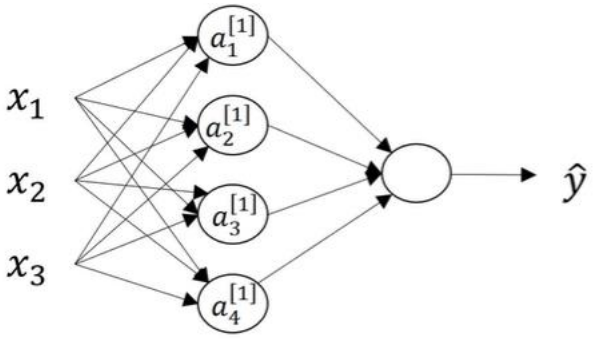
To be specific, “w” here refers to weight, which signifies the strength of the connection between the two nodes. It decides the impact that the particular input needs in the summation function. “b” refers to bias. Weights and bias work together to transform the input value x into a new value z. Then we use “σ” as the activation function to transform z into a, which is the value we want. (If in a neural network with only one hidden layer, then a is just the same as the output value).

3) an output layer, where we can generate the predicted values $\hat{y}$, which is always represented as $a^{[2]}$.

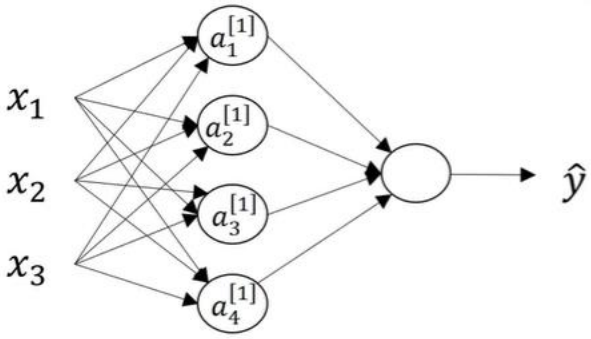


$$z = w^T x + b$$

$$a = \sigma(z)$$



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Figure1.2

In the training set, the true values for the nodes in the middle (hidden layer) are not observed, so we cannot see it in the training set. What happens in the hidden layer is computing a neural network's output and vectorizing those values, as shown above from the figures – it is a single training example.

What happened with multiple training examples is just similar to the process of a single training example. Let we denote x as $a^{[2]} = \hat{y}$, then we could get the general equation for $x^{(m)}$ as:

$$\boldsymbol{x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}}$$

Where [2] means layer 2 and (m) denotes the training example i. Similarly, with the $x^{(m)}$, for i = 1 to m, we could obtain the following equations:

$$\boldsymbol{z^{[1](i)} = w^{(i)} x^{(1)}}$$

$$\boldsymbol{a^{[1](i)} = \sigma(z^{[1](i)})}$$

Repeating the above process, we will end up getting three matrices X, Z, A, containing different training examples in different columns, as:

$$\boldsymbol{X = [x^{(1)}\ x^{(2)} \ldots x^{(m)}]}$$

$$\boldsymbol{Z = [z^{[1](1)}\ z^{[1](2)} \ldots z^{[1](m)}]}$$

$$\boldsymbol{A = [a^{[1](1)}\ a^{[1](2)} \ldots a^{[1](m)}]}$$

We now get the correct implementations in the network with vectorization, across multiple training examples.

Noticing that we need to use activation functions both in the hidden layer and the output layer, as the σs in both $a^{[1]} = \sigma(z^{[1]})$ and $a^{[2]} = \sigma(z^{[2]})$ all refer to activation function. The activation functions are directly related to how fast the neural network

learns, and therefore, how to choose a consistent and efficiency activation function is worthy discussing. We mainly have four types of activation functions –

1) sigmoid ($a = \frac{1}{1+e^{-z}}$)

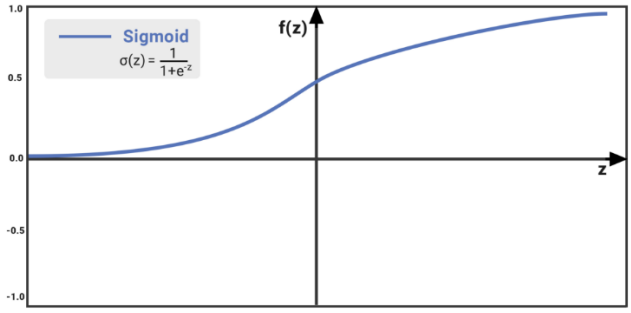


Figure 1.3

2) tanh ($a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$)

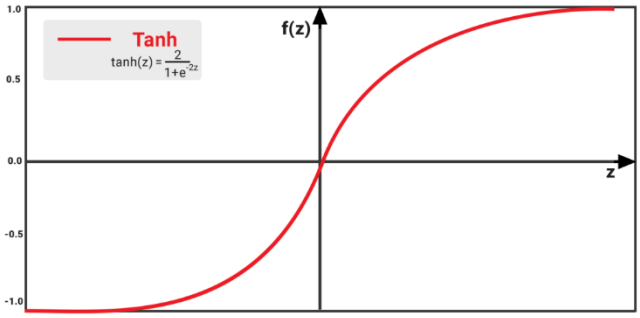


Figure 1.4

3) Rel U ($a = \max\{0, z\}$)

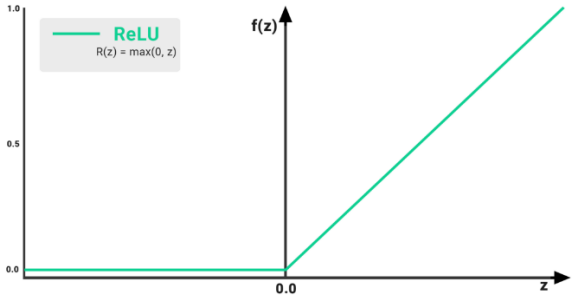


Figure 1.5

4) leaky Rel U ($a = \max\{0.01z, z\}$)

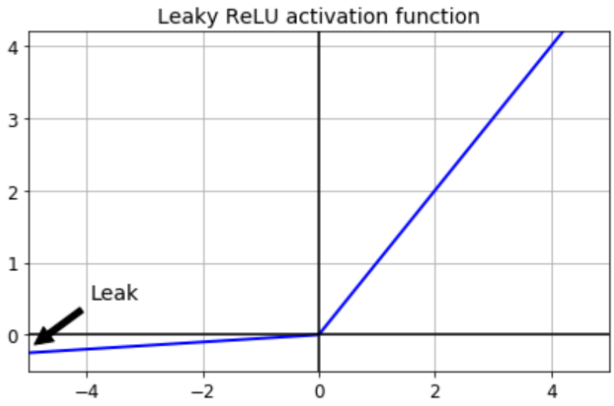


Figure 1.6

When we take a look into the derivatives of these four kinds of activation functions, we will get separately the slope of the first two functions as $a(1-a)$ and $1-(\tanh(z))^2$. The first slope tells us that when z is large or small enough, the slope of the sigmoid function is roughly zero; when z is equal to 0, however, the slope will be equal to one quarter. Similarly, the second slope tells that when z is large or small enough, the slope of the tanh function is roughly zero; whereas the slope would equal to 1 when z is equal to zero. The derivatives of the last two functions, RelU and leaky RelU, could be a little

different since we never need to find out the situation when z equals to 0 – it does not matter. We only need to figure out the cases where z is larger or smaller than zero. On the one hand, for the RelU function where g(z) = max (0, z), g'(z) would be 0 if z is smaller than 0, and it would be equal to 1 is z is larger than 0. On the other hand, for the leaky RelU function where g(z) = max (0.01z, z), g'(z) would be 0.01 if z is smaller than 0, and it would be 1 when z is larger than 0, which is just the same as the RelU's. We never use the first one for output layer, and the second one is much superior compared to the first one. The last two are most common used now, as using linear activation functions makes neural network learn faster – because there is less of effect of the slope function going to zero, which slows down learning. Even though the above sentence seems like convincing evidence that linear activation functions are better than non-linear ones, we still need non-linear functions, since if we use linear activation function, $a^{[1]}$ is just represented as a linear function of input, having no hidden layers and computing the linear function no matter how many layers we have. While we do these steps in R, we are supposed to do random initialization, as we cannot initialize with all zeros, which would cause computing the same functions. Also, while doing random initialization, we want the random values we choose are small, which can faster the learning.

Loss function, or cost function, is a method of evaluating how badly the algorithm models the dataset and of giving the difference between actual values and predicted values. In general, the better your prediction is, the lower number the loss function will output. There are four major categories of loss functions: regressive loss functions,

classification loss functions, embedding loss functions, and visualizing loss functions. The most common loss function that we always use for observing the accuracy is mean squared error (MSE). For a single training example, the loss function would be like:

$$J\left(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{n} L(\hat{y}, y)$$

To update our ways based on the loss we have collected, we use optimization to help us minimize the calculated error and modify the weights. To achieve this, we use optimization functions to calculate the gradient (the partial derivatives of loss functions), as shown below in figure 1.7. One of the most common used optimization methods is Gradient descent, which calculates gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minima.

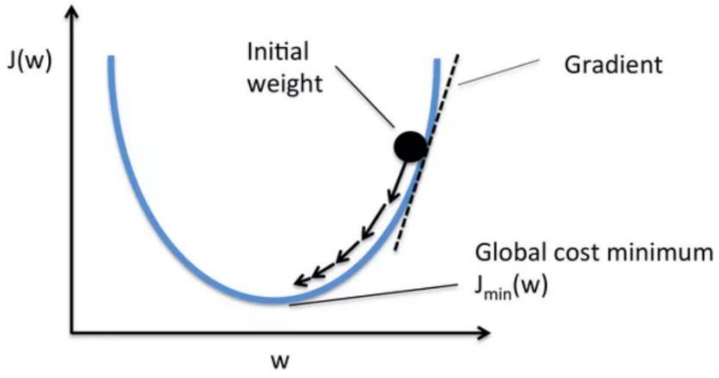


Figure 1.7

When we mention that weights are modified in the opposite direction of the calculated gradient, we are using back propagation, which is an algorithm used to train neural networks, with access to the loss function values and along with an optimization

routine. We use chain rules and doing backwards to the logistic regression. Instead of the process that we use the input values to get z, and then use z to get a, and then use a to get the predicted value, back propagation requires us to get a from the predicted value, and get z from a. By doing this, we need to use derivatives.

2.4 My Neural Network Method

I apply a three-hidden-layer neural network model to the dataset. Before constructing the model, I first do the data cleaning, making sure there is no missing or null data in x training dataset, y training dataset, x testing dataset, and y testing dataset. To make sure the x and y dataset are suitable for neural network working, I transform the y training data into matrix, then I do one hot encoding to the y training data matrix, transforming the numbers 1-12 to the matrix that only contains 0 and 1. One hot encoding could help neural networks to do a better job in prediction. I also transform x training data to the matrix as well. Then I construct a keras sequential model, which is a five-layer neural network model, with one input layer, three hidden layers, and one output layer.

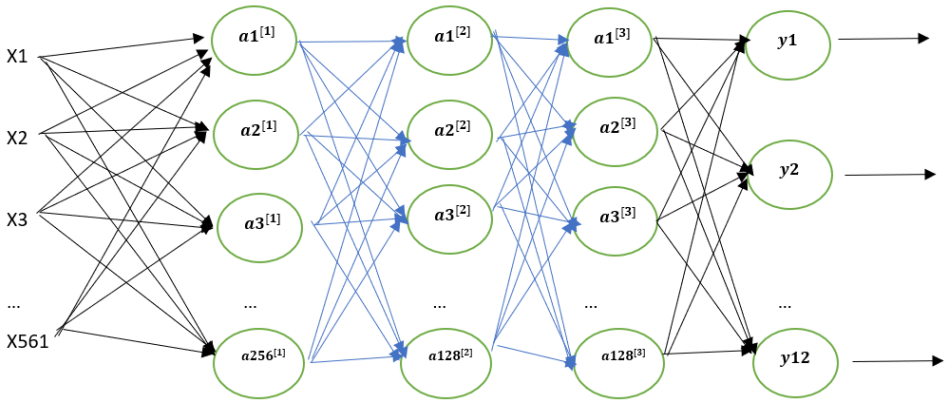


Figure 1.8

For this model, I construct 3 hidden layers, utilizing relu activation functions. As the figure I draw which was shown above, the input values are 561, as I have 561 different features in my dataset. For the first hidden layer, I give 256 units, and I give 128 units for the second and third hidden layers. I give a relatively large number of units as I have a big dataset, and I want the more units could help me gain better and more accurate prediction results. For the output layer, I give units 12 as the same number of columns of y and uses softmax function as activation function. After constructing the model, I compile the model using categorical_crossentrophy as loss function, which is fit to be used when we have two or more categorical variables. Categorical_crossentrophy is one kind of classification loss functions. It calculates the loss as

$$Loss = -\sum_{i=1}^{Output\ size} y_i \cdot log\hat{y}_i$$

I use Adam as optimizer function, and I choose accuracy be the metrics that we want to test on. I have tried both Adam and SGD as both of them are popular and advantageous optimization functions, and I find out Adam works better this time. Then I fit the training data to the model, using 100 epochs with 50 batch size and splitting 0.1 of the training data into validation set. I choose these numbers big as I want to get more accurate results while testing my model, but the same time I choose such numbers not so big since I do not want to face the issues of overfitting. I plot the fitting trend and do one hot encoding to the y testing data again, after transforming both x testing data and y testing data into matrices. Then I use the transformed testing data to evaluate the model and see the accuracy of it.

To make comparisons to see which model the best is to do predictions, I also construct a multinomial logistic regression model and a multi-class support vector machines model, and I will discuss my results in the following part.

3. Analysis

3.1 Results

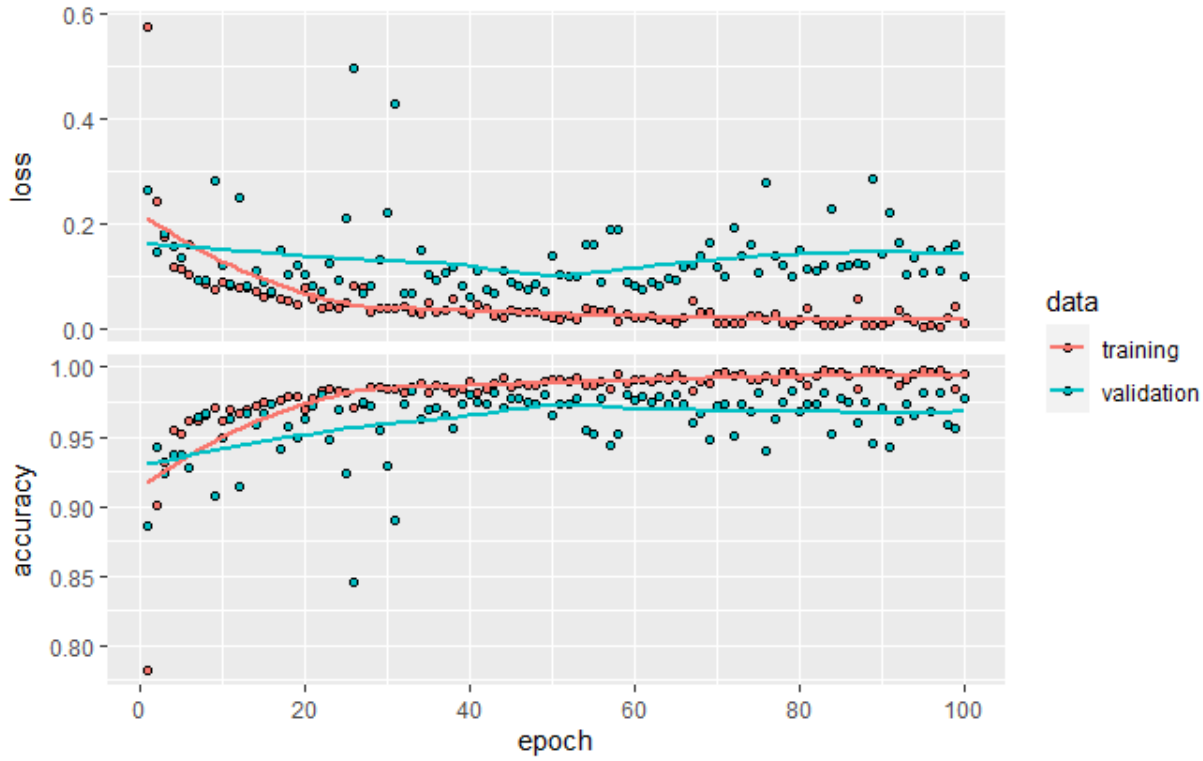


Figure 1.9

From the plot I get for my training data from the neural network model, I can see the overall trend for the accuracy of prediction is increasing, and the overall trend for the loss is decreasing, which shows that my neural network model does not overfit the data. Applying this neural network model to the testing data, I get the results of accuracy as approximately 93.39%. For the multinomial logistic regression model, I use it to do predictions for the testing data and get the accuracy of 88.14%. Moreover, for the multi-class support vector machines model I use to test predictions for the testing data, I get the resulting accuracy as 84.28%.

3.2 Conclusion

Generally, based on the results of my 3 methods, I could conclude that neural network model works the best. However, from the other researchers who previously used this dataset, they made the accuracy as 96% from a multi-class support vector machines model. Therefore, I believe that there is still a long way for me to improve and develop my model to make it gain more accurate and exact results. Something that could be developed for my model is that maybe I can increase the numbers of hidden layers, but this could lead to a higher cost of computations, which may let my computer run a much longer time to get the result. Another approach that could be applied is that I can decrease the numbers of features, remaining the important ones and removing the ones that are not closely connected to do the predictions. Overall, I believe 93.39% is not a bad guess, but there are still some more spaces for me to develop and perfect.

Reference

1. Figure 1.2 Neural Networks and Deep Learning, Andrew Ng
2. Figure1.3, 1.4, 1.5 Activation Functions in Neural Networks, Hamza Mahmood
3. Figure 1.6 Training Neural Networks for price prediction with TensorFlow, Jan Majewski
4. Figure 1.7 Loss Functions and Optimization Algorithms. Demystified, Apoorva Agrawal
5. Deep Learning with R, Francois Chollet
6. Jorge-L. Reyes-Ortiz, Luca Oneto, Albert SamÃ , Xavier Parra, Davide Anguita. Transition-Aware Human Activity Recognition Using Smartphones.
7. A Public Domain Dataset for Human Activity Recognition Using Smartphones, Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz
8. Golestani, N., Moghaddam, M. Human activity recognition using magnetic induction-based motion signals and deep recurrent neural networks
9. Jorge-L. Reyes-Ortiz, Luca Oneto, Albert SamÃ , Xavier Parra, Davide Anguita. Transition-Aware Human Activity Recognition Using Smartphones.