ENEL 525 F2020 – Final Project

Erik S

December 16th, 2020

Introduction

The COVID-19 pandemic has defined an unprecedented set of problems for researchers and scientists to solve. In this array of problems, the task of testing and correctly identifying positive and negative cases of COVID-19 has been of utmost importance.

This project aims to use the available technologies and techniques of machine learning to develop a network that can identify positive COVID-19 cases. This is done using a publicly-available dataset containing various measures of blood markers, ages, and SARS-CoV-2 test results of different patients. A summary of the information provided and the data given is has been presented in follows in Table 1.

Table 1: Blood Count Information Input Data

Column	Description	Data type
0	Row index (corresponding to the original file). The samples containing NaN/missing data are removed, so there are 598 individuals with full blood counts.	Integer
1	age (percentile group)	Continuous
2	rt-PCR SARS-CoV-2 test	Discrete Class [Positive, Negative]
3	hematocrit	Continuous
4	haemoglobin	Continuous
5	platelets	Continuous
6	mean platelet volume (MPV),	Continuous
7	red blood cells (RBC),	Continuous
8	lymphocytes	Continuous
9	mean corpuscular haemoglobin concentration (MCHC)	Continuous
10	leukocytes	Continuous
11	basophils	Continuous
12	mean corpuscular haemoglobin (MCH)	Continuous
13	eosinophils	Continuous
14	mean corpuscular volume (MCV)	Continuous
15	monocytes	Continuous
16	red blood cell distribution width (RBCDW).	Continuous

The task at hand is to design, configure, train, and test a machine learning architecture that can accurately and reliably predict positive COVID-19 cases using the blood count information as input data.

Methodology

The inputs of the problem include all columns in the given dataset. The inputs for the network are the same as above but will exclude column 0 (the row index of the original dataset) as this data is independent of the blood data or test results.

All the blood count data have been standardized to have a mean of 0 and a standard deviation of 1. Since this data is already appropriate to use to train a neural network, no cleaning of this data will be done. Since the ages are delivered as percentiles (ranging from 2-28), this input would proportionally have a much larger effect on the network and potentially skew results. Therefore, I chose to scale the age data between values of 0 and 1 to restrict the values into a range that would affect the weights less.

Input data and corresponding value ranges:

Input	Description	Data type	Input Values
1	age (percentile group)	Continuous	[0, 1]
2	hematocrit	Continuous	(-1, 1)
3	haemoglobin	Continuous	(-1, 1)
4	platelets	Continuous	(-1, 1)
5	mean platelet volume (MPV),	Continuous	(-1, 1)
6	red blood cells (RBC),	Continuous	(-1, 1)
7	lymphocytes	Continuous	(-1, 1)
8	mean corpuscular haemoglobin concentration (MCHC)	Continuous	(-1, 1)
9	leukocytes	Continuous	(-1, 1)
10	basophils	Continuous	(-1, 1)
11	mean corpuscular haemoglobin (MCH)	Continuous	(-1, 1)
12	eosinophils	Continuous	(-1, 1)
13	mean corpuscular volume (MCV)	Continuous	(-1, 1)
14	monocytes	Continuous	(-1, 1)
15	red blood cell distribution width (RBCDW).	Continuous	(-1, 1)

The target of the project is to correctly identify positive and negative cases, so the output of the network relies on a binary decision. As mentioned in chapter 22 of the course textbook[1], it can be more accurate to create two dedicated output neurons for each case (positive or negative) so two neurons in the output layer were used.

The output layer could then be interpreted as such:

	Output Layer Neuron	
	1	**2**
Positive	0	1
Negative	1	0

Network Design

To design the network I first had to decide on a learning algorithm. As the data provided was continuous, and therefore would cause varying levels of precision and error in the results, I decided that backpropagation would be the best algorithm to use.

Following was the network architecture, namely designing the hidden layers and the transfer functions. The network design was completed by using the 15 inputs on the input layer and 2 outputs on the output layer. I determined that the use of two hidden layers would be simple to implement, allow me to dynamically change the number of neurons in each layer, and have a level of complexity in the network that was fitting for the problem.

In the various configurations I used for the network, it appeared as though having 6 neurons provided a reasonable amount of complexity for the problem. It also helped avoid too many local minima in finding the minimum mean-squared-error when repeatedly iterating through the training data.

Finally, the transfer functions needed to be chosen. In backpropagation, the hidden layers function best if a non-linear function is used. The two options suggested in chapter 22 of the textbook [1] were the *logsig* and *tansig* functions. I decided to use tan-sigmoid function for the hidden and output layers as it allows a

Formula 1: Hyperbolic tangent sigmoid transfer function:

$$tansig(n) = \frac{2}{(1 + e^{-2n})} - 1$$

$$\frac{d}{dx} tansig(n) = f' = (1 - n^2)$$

Network diagrams:

Figure 1: Neural Network Diagram

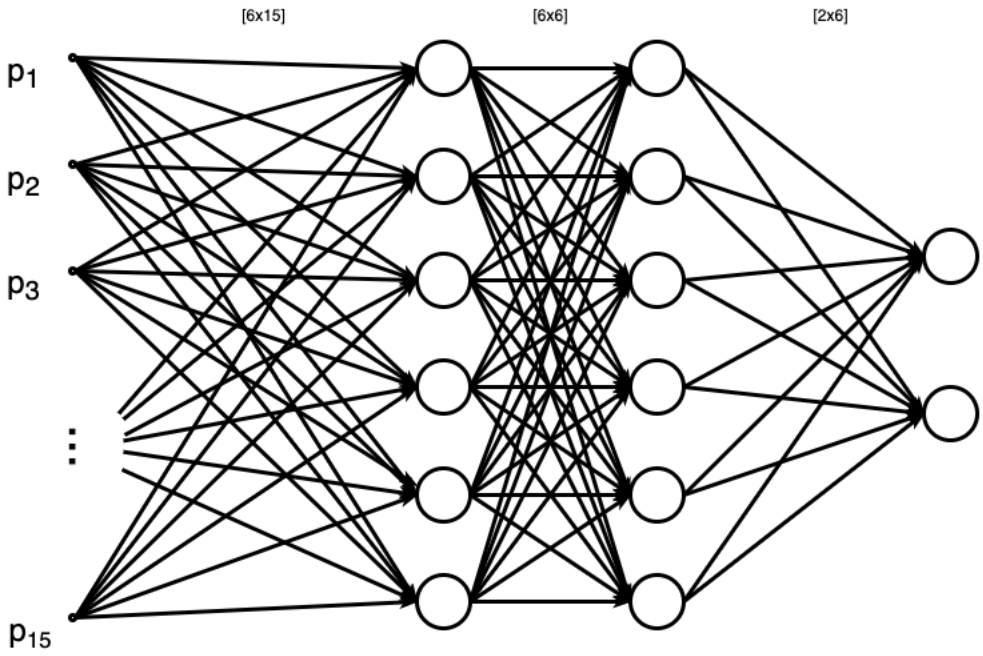


Figure 2: Hidden Layer Neuron

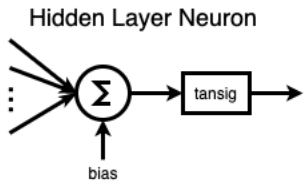
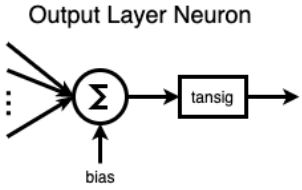


Figure 3: Output Layer Neuron



The training was done by using the training data to update the weights and biases.

Derivation of Backpropagation Learning Rule

Backpropagation derivation (inspired by Chapter 11 of the textbook [1]:

The backpropagation algorithm includes the three following steps:

Where $\vec{a} = output$, $\vec{e} = error$, $\vec{b} = bias$, and $\alpha = learning\ rate$

Step 1: Propagate inputs through the network

$$\vec{a}^{m+1} = tansig\left(\overrightarrow{W}^{m+1}p + \vec{b}^{m+1}\right) \quad \text{for } m = 0, 1, 2, \dots, L-1$$

$$\vec{a} = \vec{a}^{L}$$

$$\vec{e} = \ t - a$$

Step 2: Back propagate the sensitivities

$$\frac{\partial \hat{F}}{\partial \vec{n}^{L}} = -2F'^{L}(\vec{n}^{L})\vec{e}$$

$$\frac{\partial \hat{F}}{\partial \vec{n}^{m}} = F'^{m}(\vec{n}^{m})\left(\overrightarrow{W}^{m+1}\right)\frac{\partial \hat{F}}{\partial \vec{n}^{m+1}} \quad \text{for } m = L-1, L-2, \dots, 2, 1$$

Step 3: Update the weights and biases

$$\overrightarrow{W}^{m}(k+1) = \overrightarrow{W}^{m}(k) - \alpha\frac{\partial \hat{F}}{\partial \vec{n}^{m}}(\vec{a}^{m-1})$$

$$\vec{b}^{m}(k+1) = \vec{b}^{m}(k) - \alpha\frac{\partial \hat{F}}{\partial \vec{n}^{m}}$$

Using these equations we can implement the back-propagation algorithm used for this project.

Training and Testing Scheme

90% of the data was used for training and the remainder 10% of data was used for testing. To ensure good input data, I randomly distributed the data until the first 538 entries had roughly the same proportion of positive tests as the last 60 test points. In the dataset, 13.55% of the cases tested positive (81 positive / 598 total cases).

To minimize the mean squared error (MSE) of the neural network, I set the learning rate to 0.01 and the error threshold to 0.02.

Results and Discussion

The network required a considerable amount of altering to get the right balance of complexity and simplicity.

Final results:

Figure 4: Mean Squared Error at each Iteration

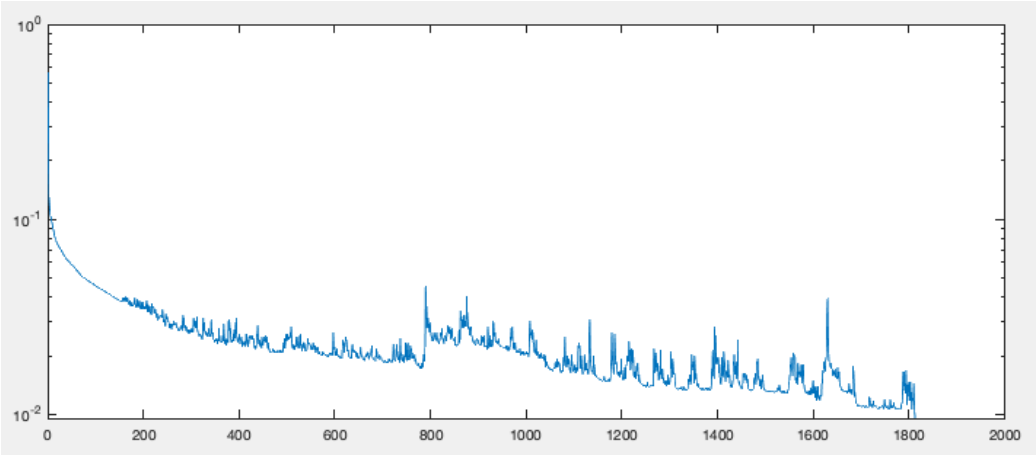
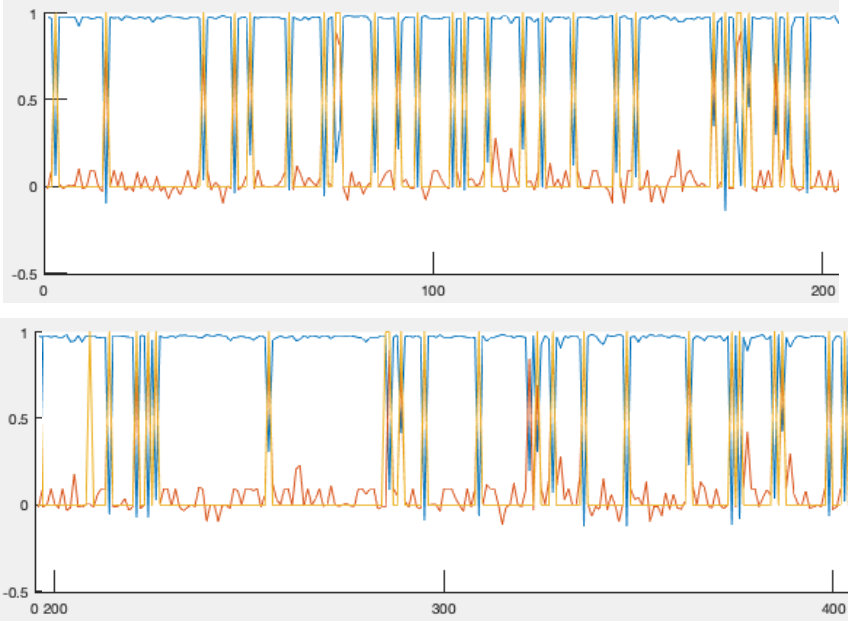


Figure 5: All Output vs Testing and Training Data



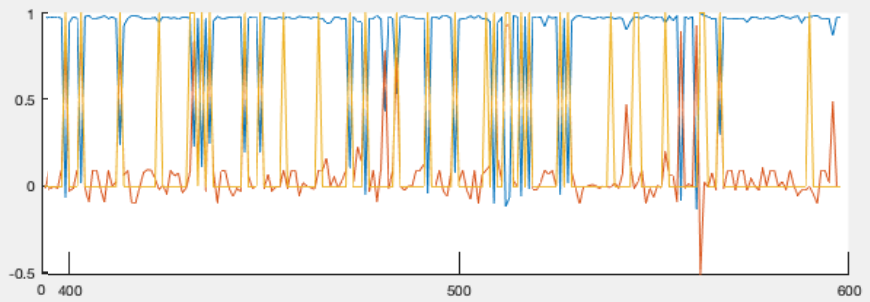
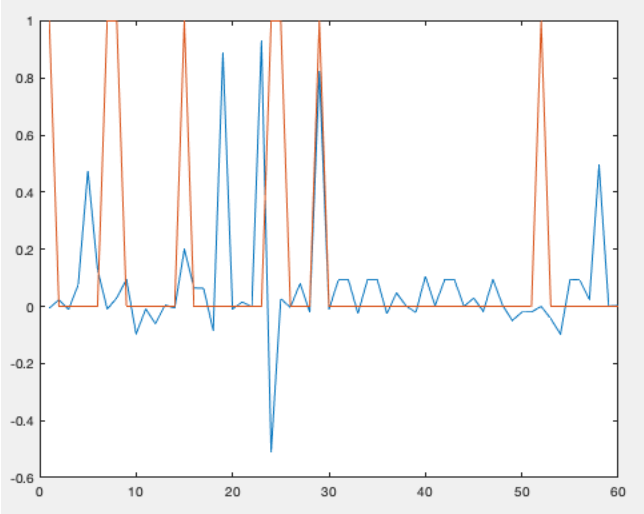


Figure 6: Output vs. Test Data



Note: Chart above only shows 2nd neuron in the output layer as output neurons are highly inversely-correlated

Conclusion

The mean-squared-error of the neural network with the training data was able to be reduced down to 0.02, however, the mean-squared-error of the test data was 0.0343. Based on the output data (figure 6), the network cannot correctly distinguish between positive and negative results.

This may be mainly due to the training data. As there is a different proportion of positive to negative cases in the network, the network is mainly trained to identify negative cases. If more data were available, or if potentially alternative training methods were explored, then potentially the error could be reduced.

As well, the use of backpropagation may have been inappropriate for this task. Other methods, such as Decision Tree or Random Forest modelling [2] may be more suitable for the problem. These methods, however, were beyond the scope of this course.

Regardless of the results of the network, valuable skills were learned about the real-world application of machine learning techniques. As well, I gained valuable knowledge about the importance of the design of neural network architecture for machine learning.

References

[1] Hagan, M. T., Demuth, H. B., Beale, M. H., & Jesús, O. (2014). Neural network design (2nd ed.). Stillwater, Oklahoma: Martin Hagan.

[2] Brinati, D., Campagner, A., Ferrari, D., Locatelli, M., Banfi, G., & Cabitza, F. (2020). Detection of COVID-19 Infection from Routine Blood Exams with Machine Learning: A Feasibility Study. Journal of medical systems, 44(8), 135. https://doi.org/10.1007/s10916-020-01597-4

Appendix

MATLAB code: (attached as .m file)