University of London International Programmes

Computing and Information Systems

Study Centre: SIM Global Education, Singapore

Topic:

CO3311 Neural Networks

Coursework 2

Prepared by:

Ong Kai Xuan

160268922

Name: Ong Kai Xuan SRN: 160268922
CO3311 cw2 Question 1: 1149 Words

Contents

Question 1) In around 1000 words describe the changes in neural network architectures and applications that have occurred during the second decade of this century, that is between 2010 and 2019. [1149 Words]

Introduction

This report will be based on the research on Artificial Neural Networks (ANNs). As compared to previous decade, today, ANNs architectures and applications are popular and are widely used by organisations and researchers to model real-world systems, as well as to understand behaviours, patterns and performances (Molaie et al, 2014). Hence, the objectives of this report are to detail understandings on ANNs around 2009 and around 2019, and also explaining the similarities, differences and the reasons for these similarities and differences.

Neural network types and sizes (architectures) and applications around 2009

Types

Firstly, there are many different types of ANNs around 2009. One of the more notable type of ANNs is ANN that uses mutation based Evolutionary Algorithms (EA) approach. According to Davoian & Lippe (2009), EA is faster in computation speed, simpler and reduces local minimum issues more effectively as compared to traditional gradient-descent learning approaches. Mutation based EA comprises of Evolutionary Programming and Evolutionary Strategies to provide ANNs with efficiencies by providing primary search operator to handle permutation problem and self-adaptive methods to change strategy parameters and distribution that are used in mutation (Davoian & Lippe, 2009).

Sizes/Architectures

Secondly, the architectures of ANNs around 2009 is commonly made up of several interconnected neurons that are arranged in layers which are comprised of input layer, hidden layers and output layer. According to Silva, Nedjah & Mourelle (2009), every layer are made up of a number of neurons which are connected by synaptic connections. The input layer consists of input neurons that are gathered by the network. The output layer are composed by the output neurons which are to be exported to the outside world of the network. Lastly, the hidden layers consists of hidden neurons to handle the training and the corresponding computation of the network (Silva, Nedjah & Mourelle, 2009).

Application

Lastly, one example of an application of ANN around 2009 is using Backpropagation Network for identification of various lung diseases. According to Gunasundari & Baskar (2009), the reason ANN is selected is because of its ability to easily identify and recognise diseases using CT scans, and by using ANN, efficiency can be achieved such that the amount of data that are required to be analysed are reduced. To begin, the ANN takes in 2D thoracic CT scan, in JPEG format, and a data set of thoraxes from trachea to below the diaphragm as input. Next, the role of the hidden layers are to calculate the covariance features for both left and right lungs that are obtained from each thoracic CT scan input. Once completed, the output layer will use the data from the hidden layers to calculate the error based on the desired classification of the lung diseases. Finally, the error will be back propagated, layer by layer in opposite direction, to recalculate the errors for all layers (Gunasundari & Baskar, 2009).

Neural network types and sizes (architectures) and applications around 2019

Types

Recently around 2019, there are two types of ANNs that are being used for analysis and training of data sets, and the two ANNs are Ensemble Neural Networks (ENN) and double convolutional Deep Neural Network (CDNN). Firstly, ENN is used in 2019 because it is ideal for solving real engineering problems, is robust, such that it is insensitive to the size of the training data sets, and also it is built based on the Bayesian framework to provide reasonable estimations, and answers to uncertainties (Chen, Chang, Meng & Zhang, 2019).

Secondly, CDNNs are ANNs that are used to classify huge number of images data and clustering them by performing calculations in its deep hidden layers (Albawi, Mohammed & Alzawi, 2017). According to Jakimovski & Davcev (2019), double CDNN are a combination of two CDNN and a max pooling layer. The role of the max pooling layer is to prevent over-fitting by reducing the size of the image data. Double CDNN is used over CDNN is because double CDNN provides more accuracy than traditional CDNN. Example to support this claim will be provided at the Application sub-section of this current section. Also, as compared to traditional CDNN, double CDNNs are more thorough and are able to easily use 3D objects as its input for training (Jakimovski & Davcev, 2019).

Sizes/Architectures

Secondly, the most common architectures of ANNs around 2019 is Deep Neural Networks (DNNs). DNNs also have an input layer and an output layer except that, as compared to ANNs, the hidden layers of DNNs are more complex, computational demanding and activation functions deeply affects how neurons influences the environment (Sarvepalli, 2015).

Application

Lastly, one example of an application of ANN around 2019 is a Double CDNN for detection of lung cancer stage. According to Jakimovski & Davcev (2019), training was done using both double CDNN and CDNN to compare the accuracy between both networks. A total of 6,080 cancerous images were used as inputs for both networks and after training both networks. After training, it is discovered that double CDNN has a 99.6% accuracy with 0.76 threshold value. On the other hand, CDNN only has an accuracy of 87% with a 0.70 threshold value. It is also discovered that double CDNN is capable of discovering cancer that are in stage 3 while CDNN cannot even detect cancer that are in stage 4 (Jakimovski & Davcev, 2019).

Comparison

The main similarities between 2009 and 2019

The main similarities between 2009 and 2019 are that ANNs are still used commonly for learning, classifying and prediction, and also ANNs are able to model complex real world problems with extreme behaviours.

The main differences between 2009 and 2019

The main differences between 2009 and 2019 in context of ANNs are limited computation power of hardware, and the lack of tools and libraries.

Reasons for these similarities and differences

The reasons to the similarities are because ANNs provides strong and reliable accuracy in solving real world problems and it is versatile such that it can takes in large data sets with many different attributes (Graupe, 2013). Another reason is that ANNs can always easily add new training examples to improve results' accuracy and usefulness (Shahin, Jaksa & Maier, 2009).

On the other hand, the reasons to the differences are because although CDNNs and RNNs are ANNs that were discovered before 2009, they were not widely used due to the limited computational power of hardware. This causes training of network to be inefficient such that it consume too much time. Another reason is that, with a lack of tools and libraries, users will tend to stick to traditional and easily found methods.

Conclusion

In conclusion, ANNs are a popular due to its success and its ability to provide robustness and accuracy since 2009. With the heightening of interest in the architecture and applications of ANNs these days, ANNs will definitely improve in the near future – being able to provide transparency and uncovering uncertainties for unsolved real world situations.

References

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. 2017 International Conference on Engineering and Technology (ICET). doi:10.1109/icengtechnol.2017.8308186

Chen, Y., Chang, H., Meng, J., & Zhang, D. (2019). Ensemble Neural Networks (ENN): A gradient-free stochastic method. Neural Networks, 110, 170-185. doi:10.1016/j.neunet.2018.11.009

Davoian, K., & Lippe, W. (2009). Mixing Different Search Biases in Evolutionary Learning Algorithms. Artificial Neural Networks – ICANN 2009 Lecture Notes in Computer Science, 111-120. doi:10.1007/978-3-642-04274-4_12

Graupe, D. (2013). Principles of Artificial Neural Networks. Advanced Series in Circuits and Systems. doi:10.1142/8868

Gunasundari, S., & Baskar, S. (2009). Application of Artificial Neural Network in identification of lung diseases. 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC). doi:10.1109/nabic.2009.5393702

Jakimovski, G., & Davcev, D. (2019). Using Double Convolution Neural Network for Lung Cancer Stage Detection. Applied Sciences, 9(3), 427. doi:10.3390/app9030427

Molaie, M., Falahian, R., Gharibzadeh, S., Jafari, S., & Sprott, J. C. (2014). Artificial neural networks: Powerful tools for modeling chaotic behavior in the nervous system. Frontiers in Computational Neuroscience, 8. doi:10.3389/fncom.2014.00040

Sarvepalli, Sarat Kumar. (2015). Deep Learning in Neural Networks: The science behind an Artificial Brain. 10.13140/RG.2.2.22512.71682.

Shahin, M. A., Jaksa, M. B., & Maier, H. R. (2009). Recent Advances and Future Challenges for Artificial Neural Systems in Geotechnical Engineering Applications. Advances in Artificial Neural Systems, 2009, 1-9. doi:10.1155/2009/308239

Silva, R. M., Nedjah, N., & Mourelle, L. D. (2009). Reconfigurable MAC-Based Architecture for Parallel Hardware Implementation on FPGAs of Artificial Neural Networks Using Fractional Fixed Point Representation. Artificial Neural Networks – ICANN 2009 Lecture Notes in Computer Science, 475-484. doi:10.1007/978-3-642-04274-4_50

Training and Developing a Backpropagation Network

1. Introduction

Backpropagation Network is a Multilayer Feedforward Neural Networks that utilises supervised learning technique to update and adjust weights based on the error between target and output (Primadusi, Cahyadi, Prasetyo & Wahyunggoro, 2016). A Backpropagation Network consists of consist of an input layer, a hidden layer and an output layer. For our training on this Backpropagation Network, there will be 24 units inside the input layer, 2 units inside the hidden layer, and 1 unit inside the output layer.

The architecture of the Backpropagation Network is as follows:

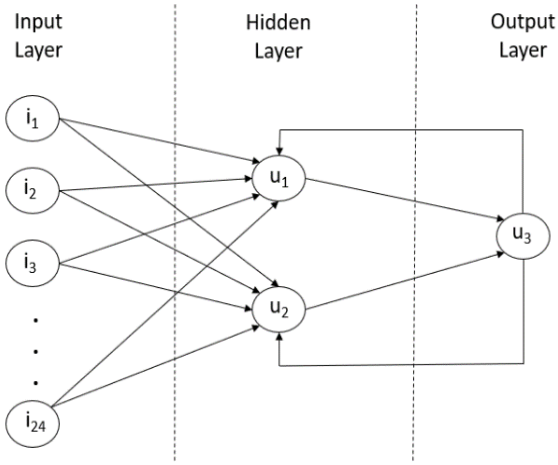

Figure 1: Backpropagation Network Architecture

Finally, the termination condition of the training will be when the network is trained to match the predicted cluster with the expected cluster. More will be explained in the following section.

2. Formulas and Measurement of Errors

Before we begin the development of a Backpropagation Network, we must first know the formulas and calculations behind it. The following formulas and measurements are based on materials from our CO3311 local lecturer, Koh (2018).

2.1. Defining Bias, Learning Rate and Calculation of Weights for All Attributes

Firstly, we need to initialise the bias, learning rate as well as the weights for both bias and the 24 inputs' 3 attributes.

This program requires user to choose their own learning rate. However, for this experiment, we will use 0.1.

Next, Both bias and its weights will be set to 1. Then, the weights representing the 3 attributes will be based on random – by selecting any value between each attribute's upper bound and lower bound.

2.2. Calculation of Target

Secondly, since Backpropagation utilises supervised learning, we will have an expected class for each set of input unit. We will then need to know the target so that we can compare the predicted class with the expected class.

To calculate the target, we will need to select a number to represent the number of expected classes. We will call this c. Next, each class belongs to a unique whole number from 1 to c inclusively. Hence, we will let i to represent the expected class.

The formula to calculate target is:

$t_i = \frac{\frac{i-1}{c} + \frac{i}{c}}{2}$, where i <= c

We will use 4 expected classes as an example. By using the formula above, we get:

$t_1 = \frac{\frac{1-1}{4} + \frac{1}{4}}{2} = 0.125$

$t_2 = \frac{\frac{2-1}{4} + \frac{2}{4}}{2} = 0.375$

$t_3 = \frac{\frac{3-1}{4} + \frac{3}{4}}{2} = 0.625$

$t_4 = \frac{\frac{4-1}{4} + \frac{4}{4}}{2} = 0.825$

2.3. Calculation of Net and Activation

Thirdly, we will need to know how to calculate the net as well as the activation function.

The formula to calculate the net is:
net = bias * W_{bias} + X * W_x + Y * W_y + Z * W_z

Sigmoidal activation function will be used to calculate the activation function:
a() = $\frac{1}{1+e^{-net}}$

2.4. Calculation of Predicted Outcome

Fourthly, once we have the target, we can calculate the predicted outcome by comparing the activation of the output layer against the target. We will again let c to represent the number of expected classes.

Formula to calculate the predicted outcome:

a() = i If a() = x where $\frac{i}{c} \geq x > \frac{i-1}{c}$

2.5. Measurement of Output Error

Fifthly, we will need to know how to calculate the output error so that we can update the weight of the output layer and the error of the hidden unit.

The formula to calculate the output error is:
output error = [target – a()] * a() * [1 – a()]

2.6. Measurement of Hidden Unit Error

Sixthly, we will need to know how to calculate the hidden unit error so that we can update the new weights of the hidden units

The formula to calculate the hidden unit error is:
hidden error = output error * a() * [1 – a()] * W

2.7. Calculation of Weight Change

Finally, we need to know how to calculate the new weights so that we can continue the next iteration. We will let variable ‘c’ represents an attribute value.

Formula to calculate the new weight:
W = W + (learning rate * hidden error * c)

Implementation of Network

To enable others to duplicate this development, this section will be described with enough details to do so.

This network will be implemented to have 2 units in the hidden layer and 1 unit in the output layer. Since this is a 3 input network, to begin, the network will require 5 parameters as inputs. The 5 parameters consist of the training dataset’s 3 attribute (X, Y and Z, each as an array), the expected classes for each set of input as an array, and a number representing the number of initial classes. The 2 units in the hidden layer will consist of a bias, bias weight, X, X weight, Y, Y weight, Z and Z weight. On the other hand, the unit in the output layer will have a bias, bias weight, and 2 inputs (activation1 and activation2) indicating the activation of hidden layer unit 1 and 2 respectively, and 2 weights representing the 2 inputs.

Next, the network requires user to declare the expected learning rate.

Then, user must define the weights to represent the X, Y and Z inputs. User can either select random weights (refer to Section 2.1.) or choose their own defined weights.

Finally, since we have already taken in the expected classes for each set of input as parameter, we will implement the network to define the target for each set of input (refer to Section 2.2.).

As a precaution, we will implement a check compare the length of the array between the first 4 parameter inputs defined earlier. If the length does not match, the network will send an error message to the user before terminating.

Once the above are done, we can begin implementing the Backpropagation Network. Firstly, we will begin by setting the bias and the bias weights for all 3 units to be 1.

Secondly, We will define 3 variable, 'numOfMatch', 'epoch' and 'iteration' and initialise them as 0, 1 and 1 respectively. The 'numOfMatch' variable will check if the predicted class matches the expected class, and if they do, the variable will increase by 1. Each epoch represents a complete training and each iteration represents a training with an input unit. Hence, 'iteration' will increase by 1 after a training with an input unit, whereas for 'epoch', it will increase by 1 only after a full training. A while loop will then be created to train the weights and will only terminate when the 'numOfMatch' matches the number of rows of the training dataset.

Thirdly, the network will start training by running through every row of the units in the training dataset. For each input unit, we will calculate the net and the activation of all hidden units, and then the output unit (Refer to section 2.3.). Once that is done, the output error will be calculated first (Refer to section 2.5.), and then the hidden units error will be calculated (Refer to section 2.6.). The next step of the implementation will be to calculate the new weights for the bias, X, Y and Z that belongs to the 2 hidden units, and bias, activation1 and activation2 that belongs to the output unit (Refer to section 2.7.). Once done, the network will calculate the predicted class and check if it matches the expected class (Refer to section 2.4.). If there is a match, 'numOfMatch' will increase by 1. The network will then continue to train with the next input unit, and will only complete once the whole training dataset has been iterated through.

Finally, once the training has been done with the training dataset, the network will check if the 'numOfMatch' matches the number of rows of the training dataset. If it does match, Training will be competed, and the new set of weights obtained can be used either for testing or for analysis. Else, 'epoch' will increase by 1, 'numOfMatch' will reset to 0 and the network will continue training using the new weights.

Table of Results

Training for 2, 3 and 4 Classes were done using the following training dataset and its classification for 2, 3 and 4 units as obtained from using the Kohonen network:

S/N	X	Y	Z	2 Unit	3 Unit	4 Unit
1	- 0.82	0.49	0.29	2	1	4
2	- 0.77	0.47	0.43	2	1	4
3	- 0.73	0.55	0.41	2	1	4
4	- 0.71	0.61	0.36	2	1	4
5	- 0.69	0.59	0.42	2	1	4
6	- 0.68	0.60	0.42	2	1	4
7	- 0.66	0.58	0.48	2	1	4
8	- 0.61	0.69	0.40	2	1	4
9	0.28	0.96	0.01	2	2	3
10	0.31	0.95	0.07	2	2	3
11	0.35	- 0.06	0.94	1	3	1
12	0.36	0.91	- 0.22	2	2	3
13	0.37	0.93	0.09	2	2	3
14	0.43	0.83	- 0.34	2	2	3
15	0.43	0.90	0.01	2	2	3
16	0.47	- 0.10	0.88	1	3	1
17	0.47	- 0.07	0.88	1	3	1
18	0.48	0.82	- 0.30	2	2	3
19	0.53	- 0.23	0.82	1	3	1
20	0.54	- 0.21	0.82	1	3	1
21	0.58	- 0.38	0.72	1	3	1
22	0.58	0.81	0.00	2	2	3
23	0.65	- 0.04	0.76	1	3	1
24	0.74	- 0.05	0.67	1	3	1

Figure 2: Classification of Training Dataset derived from Kohonen Network

The sub-section below contains details for training with 2, 3 and 4 Classes.

Training with 2 Classes

The initial training details are as follows:
Cluster size: 2
Learning Rate: 0.1
Random weight selected
Unit 1 Weight X: 0.64
Unit 1 Weight Y: 0.07
Unit 1 Weight Z: 0.64
Unit 2 Weight X: -0.56
Unit 2 Weight Y: 0.57
Unit 2 Weight Z: -0.02
Unit 3 Weight for Activation1: -0.04
Unit 3 Weight for Activation2: 0.07

After running the Backpropagation Network, the training is completed at epoch 87, iteration 2088. The new weights are as follows:

Unit 1 Weight[bias, x, y, z]:
0.649829499965222,
0.9649820510052031,
-0.5297217215060984,
0.8449200941287235

Unit 2 Weight[bias, x, y, z]:
0.873500914327623,
-0.7356730738122127,
0.7718075653407491,
-0.26756838755207585

Unit 3 Weight[bias, U1, U2]:
0.608472893762211,
-1.2794251019156175, 0.744630208681025

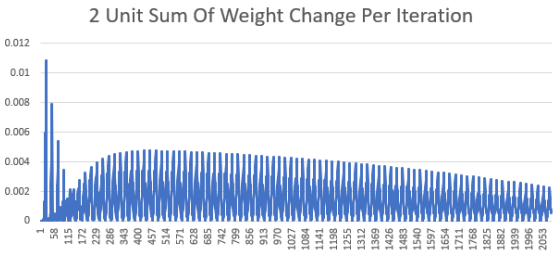

Figure 3: 2 Unit Sum of Weight Change (Iteration)

From figure 3, we can see that the Sum of Weight Change for each iteration is still going down at a consistent rate. This can suggest that there might be better sets of weight if we had continued the training, or changed the terminating condition.

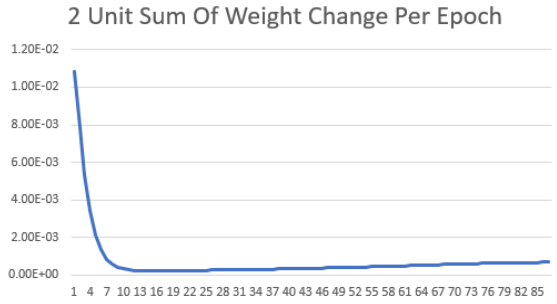

Figure 4:2 Unit Sum of Weight Change (Epoch)

From Figure 4, we can see that even though the Sum of Weight Change are gradual, it is still increasing even though the training has already been completed. We can also see that the graph in Figure 3 is going downwards but in Figure 4, the graph is going upwards. This can suggest that there is a high probability of getting better sets of weight if we had continued the training, or changed the terminating condition.

Training with 3 Classes

The initial training details are as follows:
Cluster size: 3
Learning Rate: 0.1
Random weight selected
Unit 1 Weight X: -0.4
Unit 1 Weight Y: -0.37
Unit 1 Weight Z: -0.01
Unit 2 Weight X: -0.45
Unit 2 Weight Y: -0.31
Unit 2 Weight Z: 0.75
Unit 3 Weight for Activation1: 0.04
Unit 3 Weight for Activation2: -0.12

After running the Backpropagation Network, the training is completed at epoch 135, iteration 3240. The new weights are as follows:

Unit 1 Weight[bias, x, y, z]:
0.6471552809084244,
-1.2362858023517496,
-0.7541773747527307,
0.19703675514370722

Unit 2 Weight[bias, x, y, z]:
0.31918645643274607,
-1.8795857889473735,
-1.0156628343826504,
1.077339929841308

Unit 3 Weight[bias, U1, U2]:
1.6228929692397789,
-0.9917354562424375,
-2.0499874325870175

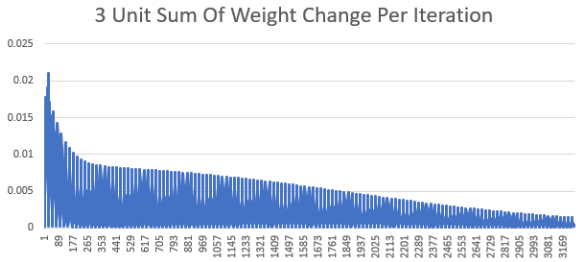

Figure 5: 3 Unit Sum of Weight Change (Iteration)

Similar to Figure 3, from Figure 5, we can see that the Sum of Weight Change for each iteration is still going down at a consistent rate, albeit almost reaching close to 0. This can suggest that there might be better sets of weight if we had continued the training, or changed the terminating condition.

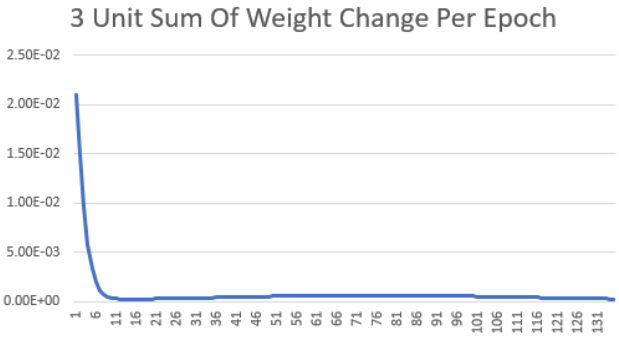

Figure 6: 3 Unit Sum of Weight Change (Epoch)

From figure 6, we can see that the Sum of Weight Change are gradual and are getting very close to 0. This can suggest that the training has already completed and the current weights that we have can be considered to be optimal.

Training with 4 Classes

The initial training details are as follows:
Cluster size: 4
Learning Rate: 0.1
Random weight selected
Unit 1 Weight X: -0.61
Unit 1 Weight Y: -0.17
Unit 1 Weight Z: 0.58
Unit 2 Weight X: 0.44
Unit 2 Weight Y: 0.56
Unit 2 Weight Z: 0.39
Unit 3 Weight for Activation1: 0.43
Unit 3 Weight for Activation2: -0.18

After running the Backpropagation Network, the training is completed at epoch 1797, iteration 43128. The new weights are as follows:

Unit 1 Weight[bias, x, y, z]:
0.1303222307503292,
-1.721301363835034,
0.2240348594904535,
-0.2872932809319455

Unit 2 Weight[bias, x, y, z]:
-0.3936336629957373,
1.8871889713550432,
-1.6186324013393536,
0.9487962076430033

Unit 3 Weight[bias, U1, U2]:
0.6900328385796128,
1.8356771228289455,
-3.678782447067824

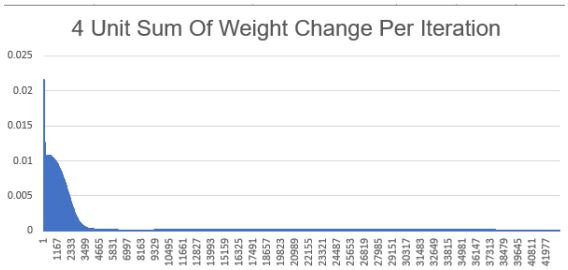

Figure 7: 4 Unit Sum of Weight Change (Iteration)

From Figure 7, unlike Figure 3 and Figure 5, we can see that there is only spiking of weights in the first 3500 iterations. After that, there is no more spikes and the weight change remains gradual. This shows that there is a high probability that we already had identified a suitable weight. But, to meet the terminating condition of 24 matches, the network is forced to go through overfitting.

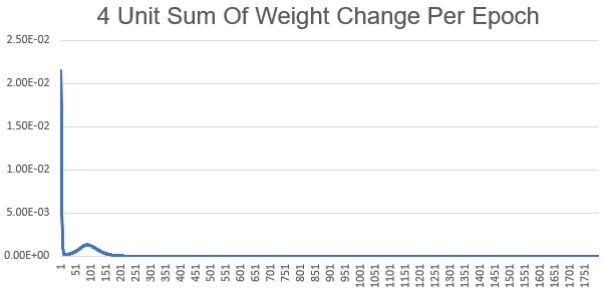

Figure 8: 4 Unit Sum of Weight Change (Epoch)

Similar to Figure 7, we can see the same thing for Figure 8. The training might have already been completed at around epoch 200. But, to meet the terminating condition of 24 matches, the network is forced to go through overfitting.

Analysis of Results

The next step would be to perform analysis using the new weights. New weights obtained from the training will be used against the testing set. The table below contains the testing set as well as the 'correct' classes for those points.

S/N	X	Y	Z	Classes out of the 2 Classes	Classes out of the 3 Classes	Classes out of the 4 Classes
1	-0.99	0.13	0.08	2	1	4
2	-0.98	0.20	0.01	2	1	4
3	-0.96	0.29	0.03	2	1	4
4	-0.94	0.29	0.18	2	1	4
5	-0.84	0.42	0.34	2	1	4
6	-0.77	0.41	0.48	2	1	4
7	-0.73	0.57	0.37	2	1	4
8	-0.70	0.47	0.53	2	1	4
9	0.31	-0.05	0.95	1	2	1
10	0.31	-0.15	0.94	1	2	1
11	0.39	-0.30	0.87	1	2	1
12	0.41	-0.14	0.90	1	2	1
13	0.46	0.89	0.04	2	3	3
14	0.47	0.87	-0.16	2	3	3
15	0.49	0.87	0.09	2	3	3
16	0.50	-0.21	0.84	1	2	1
17	0.52	-0.28	0.80	1	2	1
18	0.56	-0.35	0.75	1	2	1
19	0.57	-0.22	0.79	1	2	1
20	0.68	0.73	0.12	2	3	2
21	0.71	0.69	0.13	2	3	2
22	0.80	0.58	0.16	2	3	2
23	0.83	0.56	-0.02	2	3	2
24	0.84	0.54	0.10	2	3	2

Figure 9: Result for Classification of the Testing Set

From "Classes out of 2 classes", we can see that:
Class 1: S/N 9, 10, 11, 12, 16, 17, 18, 19
Class 2: S/N 1, 2, 3, 4, 5, 6, 7, 8, 13, 14, 15, 20, 21, 22, 23, 24

From "Class out of 3 classes", we can see that:
Class 1: S/N 1, 2, 3, 4, 5, 6, 7, 8
Class 2: S/N 9, 10, 11, 12, 16, 17, 18, 19
Class 3: S/N 13, 14, 15, 20, 21, 22, 23, 24

From "Class out of 4 classes", we can see that:
Class 1: S/N 9, 10, 11, 12, 16, 17, 18, 19
Class 2: S/N 20, 21, 22, 23, 24
Class 3: S/N 13, 14, 15
Class 4: S/N 1, 2, 3, 4, 5, 6, 7, 8

Conclusion

In conclusion, by comparing the results from "Classes out of the 2 classes" and "Classes out of the 3 classes", we can see that:

- Class 1 from Classes out of the 2 classes = Class 2 from Classes out of the 3 classes

- Class 2 from Classes out of the 2 classes = Class 1 + Class 3 from Classes out of the 3 classes

Hence, it is possible for us to say that both Class 1 and Class 3 of "Classes out of 3 classes" are closely related since both of them belongs to the same class, Class 2, from "Classes out of 2 classes".

Similarly, by comparing the results from "Classes out of the 3 classes" and "Classes out of the 4 classes", we can see that:

- Class 1 from Classes out of the 3 classes = Class 4 from Classes out of the 4 classes

- Class 2 from Classes out of the 3 classes = Class 1 from Classes out of the 4 classes

- Class 3 from Classes out of the 3 classes = Class 2 + Class 3 from Classes out of the 4 classes

Thus, it is possible for us to say that both Class 2 and Class 3 of "Classes out of 4 classes" are closely related since both of them belongs to the same class, Class 3, from "Classes out of 3 classes".

References

Koh, W. B. (2018). Chapter 6: Training Networks [PowerPoint slides]. Retrieved from Singapore Institute of Management CO3311

Primadusi, U., Cahyadi, A. I., Prasetyo, D., & Wahyunggoro, O. (2016). Backpropagation neural network models for LiFePO4 battery. doi:10.1063/1.4958527