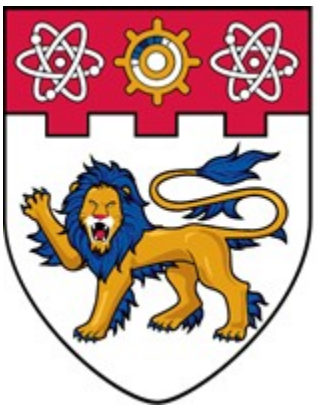

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

Artificial Intelligence

Autonomous Deep Quality Monitoring in Streaming Environments Reflection

18 September 2021

Done By:

Tang Kai Wen Alvin

Task 1

For this task, we are limited to the usage of a three-layer feed forward neural network as well as SGD optimizer for our neural network. That will mean that the number of hidden layers will be fixed at 2. The various things that we will be focusing on the ensure that we obtain the smallest testing errors will be adjusting the number of hidden nodes, the number of epochs, the learning rates as well as the mini-batch size.

To identify the best values for each, we will make observations with regards to each of the variables being change with respect to the classification error which will be discussed as the test error. It is important to note that it is very hard to classify the tables for each of the below observations as the training is constantly adapting to the various observations and will result in better or worst test error results due to having more than 1 variables being change in order to identify the various effects. The final observation is stated after running the training multiple times with the reason given for the chosen number.

For the number of hidden nodes, as it becomes too small, the test error becomes too big. However, if the number of hidden nodes exceed a certain amount, the test error plateaus at a high-test error. Hence, it will be important to balance that in order to minimize the testing errors. As such, the number of hidden nodes will be set to 8. The input units is set to the number of dimensions multiplied by the dimensions of the feature vector while the output units is the number of classes in classification problem.

For the number of epochs, there is an ideal number that is not too large or not too small to allow sufficient training before the test error remains constant. After which, the training will only eat into the time taken and become not efficient. For that the number of epochs will be set to 200 for it to settle on the lowest testing error and stabilize.

For the learning rate, we want it to be high enough such that it does not plateau at a high testing error but a lower testing error. Hence the starting learning rate cannot be too high as we divide it for every epoch. The one that I've settled for achieving the lowest learning rate is learning rate of 0.02 divided by 2 every 20 epochs.

Finally, for the batch size, 53 will be used for the training set in order to obtain the lowest learning rate. It has resulted in the lowest test error that we can obtain.

The results of these values will result in the test error being 2.15% as shown below:

```
epoch= 190  time= 5.020946264266968  loss= 0.26341018615624845  error= 12.288339932759602 percent lr= 3.90625e-05
test error  =  2.146892547607422 percent
```

It is important to note that before reading the report on the study of the various effects, the ideal number that are shown in each of the tasks 2, 3, and 4, are only specific to the training shown in the code for each of the tasks, transferring those ideal numbers will not work on task 1 as it uses different variables such as number of hidden nodes or the number of epochs. However, the understanding will still apply to task 1.

Task 2

For this task, we are to study the effect of network structure: hidden nodes and hidden layers, to the classification performance. For us to properly identify their significance, we need to properly apply what the different aspects of network configurations means. We can summarise the types of layers in an MLP as follows:

- **Input Layer**: Input variables, sometimes called visible layer
- **Hidden Layer**: Layer of nodes between the input and output layers. There may be one or more of these layers
- **Output Layer**: A layer of nodes that produce the output variables.

Finally, there are terms used to describe the shape of a neural network:

- **Size**: The number of nodes in the model.
- **Width**: The number of nodes in a specific layer.
- **Depth**: The number of layers in a neural network.

Thus, the different aspects of network configurations will be as follows:

- **Shallow Network**: Consists of only 1 to 2 hidden layers limited to few nodes in the hidden layers
- **Wide Network**: Consists of large number of nodes in the hidden layers limited to 1 to 2 hidden layers
- **Deep Network**: Consists of large number of layers in the network limited to few nodes in the hidden layers

	Number of Hidden Nodes	**Number of Hidden Layers**
Shallow Network	Few	Few
Wide Network	Large	Few
Deep Network	Few	Large

Table: Visualization of the various network configurations

As such, to see the effect of hidden nodes and hidden layers, a table will represent the relationship of hidden nodes and hidden layers with respect to the efficiency of the neural network base on the time taken and test error of the training method in the code. All other factors like learning rates and batch size will remain constant throughout the testing.

Note, the following time taken (2.d.p.) and test error (2.d.p.) will be when epoch is at 190.

Test case no.	# of Hidden Nodes	# of Hidden Layers	Network Configuration Aspect	Time Taken	Test Error
1.	1	1	Shallow	3.55	96.61%
2.	1	2	Shallow	4.37	75.59%
3.	2	1	Shallow	4.20	95.03%
4.	2	2	Shallow	3.98	92.54%
5.	20	2	Wide	4.25	100.00%
6.	48	2	Wide	5.05	100.00%
7.	72	2	Wide	5.69	75.71%
8.	96	2	Wide	5.08	100.00%
9.	2	3	Deep	4.14	77.85%
10.	2	4	Deep	4.32	97.85%
11.	2	5	Deep	4.57	75.71%
12.	2	6	Deep	5.43	75.59%
13.	2	7	Deep	5.68	75.59%
13.	2	10	Deep	6.27	75.59%
14.	2	20	Deep	9.12	75.59%
Ideal	72	6	Wide + Deep	10.37	25.08%

From observation, as the number of hidden layers increase, the time taken increases. Whereas as the number of hidden nodes approaches the ideal number of hidden nodes (i.e. around 1.5 times the input features), the test error drastically decreases.

For the first observation, this is because as the hidden layers increases, it becomes more computationally expensive in terms of the time needed to compute the result due to the large number of hidden layers to go through. As such, we can see the timing increases as the number of hidden layers increases.

As for the number of hidden nodes, though more hidden nodes is better, the test error increases after a certain point. The reason why it is capped at 72 is because of the input being 48. Different data will have their own ideal number of hidden nodes. Hence, through experimentation, 72 results in the most ideal test error as seen in the table. Lesser or larger than this will result in the increase in test error

The ideal hidden nodes and hidden layers is 72 and 6 respectively. More than that may result in a dip in test error or an increase in time taken to execute the test. This is due to the possibility that as we build a very wide network, we may run the chance of each layer just memorizing what the target output is and we end up with a neural network that fails to generalize to new data. Thus, a decently wide and sufficiently deep neural network would be ideal for the least number of test errors that is within reasonable time.

Task 3

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. It might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minima, it could also mean that we'll be taking a long time to converge — especially if we get stuck on a plateau region. The below diagram demonstrates various scenarios one can fall into when configuring the learning rate:

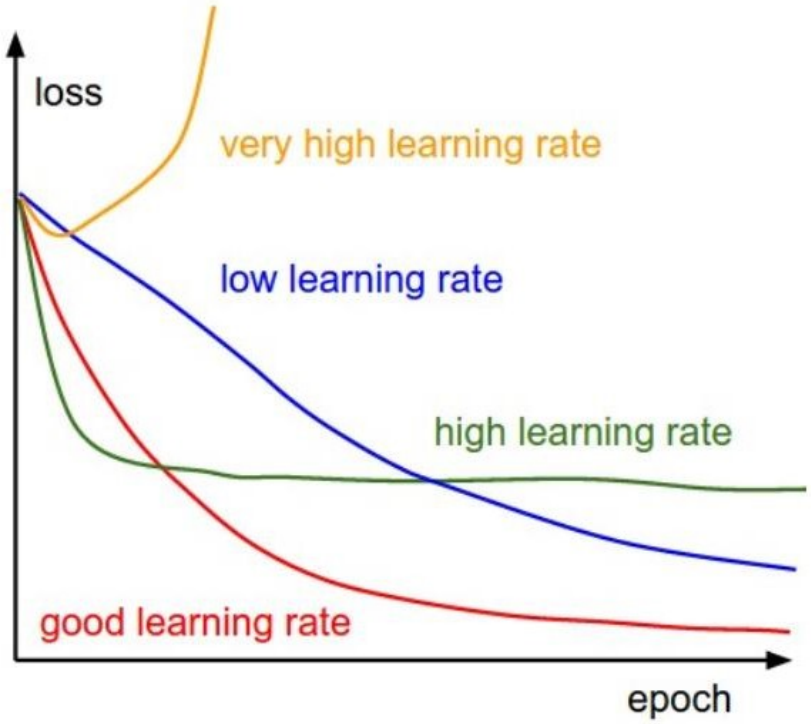

Diagram 1 – Effects of various learning rates on convergence

In this assignment, we will only be focusing on the effects of Stochastic Gradient Descent (SGD) and for us to see if the above diagram applies to our training method, all other variables will remain constant while we adjust the learning rate. The 3 various things we are recording is the initial test error at epoch = 0, the lowest point of the test error and the ending test error at epoch = 190 with the percentage to 2 decimal place. We will also analyze base of the diagram, what time of learning rate this is.

Test case no.	Learning rate	Initial test error	Lowest test error	Ending test error	Learning Rate Type
1.	0.01	100.00%	3.05%	10.85%	Default
2.	0.02	50.85%	1.69%	16.05%	High
3.	0.05	75.59%	14.01%	35.48%	Very High
4.	0.005	99.10%	1.81%	3.16%	Low
5.	0.001	85.65%	75.59%	75.59%	Low

From this, we can observe that if the learning rate remains constant for high learning rate like test case number 2 and 3, it will result in a drop from the initial high-test error and as the epoch increases, the ending test error will go back up after hitting its lowest test error. That's because high learning rate will decay the loss faster which results in the drop in test error. However, as the epoch increases, the learning rate becomes too high that causes the parameter update to become too chaotic and result in worse ending test error.

Whereas when we look at the low learning rate like test case number 4, it will result in a similar fashion that was shown in the diagram whereby there is a slow and steady decrease in the test error. However, once the learning rate goes too low, it will result in a plateau as we can see in test case number 5 whereby the test error stabilized.

Hence, from this experiment, it would be ideal if we start off from test case number 2 with learning rate of 0.02 to have a fast drop in initial test error but have it adapt so that the learning rate will decrease as the epoch increases to allow the error rate to stabilize at the final epoch = 190. In order to see if our hypothesis is true, the function monitoringTrainingWithAdaptiveLearningRate() is called that uses a weight decay of the learning rate from 0.02. We will fix the other variables and only changing the rate of which the learning rate is divided by (for every n epoch) as well as how much the learning rate is being divided/multiplied by

Test case no.	Divide learning rate by	For every n epoch where n is:	Initial test error	Lowest test error	Ending test error
1.	1.5	10	75.59%	5.54%	5.54%
2.	1.5	20	81.58%	2.60%	6.93%
3.	1.5	40	64.18%	5.42%	7.68%
4.	2	10	75.59%	24.63%	24.63%
5.	2	20	100.00%	2.60%	2.94%
6.	2	40	72.54%	3.39%	8.81%
7.	2.5	10	61.69%	9.60%	9.60%
8.	2.5	20	75.59%	4.86%	4.86%
9.	2.5	40	70.17%	6.33%	11.98%
10.	.5	10	73.56%	34.12%	75.59%
11.	.5	20	73.56%	3.95%	75.59%
12.	.5	40	73.56%	4.01%	77.91%

As we can see that if we keep on multiplying, the learning rate will be too high and thus the ending test error will become very high at the end of epoch = 190. From here, we can observe that diving learning rate through weight decay, if we do it for every 20 epoch, it will result in the lowest test error. This is probably case specific to this testing base on the batch size that we pre-defined. Thus, focusing on learning rates that are being divided for every 20 epochs, we can see that as the learning rate gets divided by a larger number, the lowest test error increases. This is probably because the learning rate that were effective before it became too small to change the training was larger for test case 5 as compared to test case 1 or 3. Hence, for the suggested adaptive learning rates to best fit our hypothesis before this testing, it would be recommended to have learning rate divided by 2 for every 20 epochs in order to result in the ideal ending test error for our testing data.

Task 4

Effect of mini-batch size

- Set it to 1(stochastic gradient descent)
- N (batch gradient descent)
- Be conclusive in the findings
- Mini-batch size depends on the problem size

What we can classify mini-batch size depends on the problem size. The mini-batch size must be a composite number that is derived from the prime factorization of the problem size. In this case, we have 2067 total for the training set which is about 70% of the total problem size. Hence we have mini-batch size of 1, 3, 13, 39, 53, 159, 689; where 1 is the stochastic gradient descent while the others are batch gradient descent. We will keep the rest of the variables constant while looking at various mini-batch size of the training set as previously discussed. To see the comparison, we will look at the various effects changing of batch size will have through the ending test error from its initial test error as well as the time taken needed to train. Note that the ending test error occurs when epoch is at 190 and the given test error and time taken will be to 2.d.p.

Test case no.	Batch Size of	Initial test error	Ending test error	Time taken to train (seconds)
1.	1	12.88%	15.59%	156.23
2.	3	66.44%	34.58%	54.88
3.	13	70.62%	17.51%	13.71
4.	39	75.59%	14.23%	5.50
5.	53	73.56%	3.84%	4.44
6.	159	76.05%	49.83%	3.16
7.	689	88.25%	50.85%	1.72

From the table, we can see that with a small batch size, the initial test error is the lowest. However, as the batch size increase, the initial test error increases as well. This is most likely due to the size of the batch being trained which will lead to more chances of having errors face initially.

However, when we look at the ending test error. We notice that it increases from the smallest batch size of 1 and drops down to 3.84% for test case number 5. But afterwards, the ending test error spike back up as the batch size increases. We can observe why this occur is because there is an ideal batch size for the specified training variables which would lead to the lowest ending test error at the end when epoch = 190. Having too large of a batch size will lead to less iteration which will lead to lower accuracy which explains why the ending test errors constantly increase as the size of the batches increases. Though having smaller size for batch size means there will be more iteration, it will lead to picking up unwanted “noise” that could lead the accuracy astray which could explain why test case number 2 and 3 had a higher test error than test case 4.

Thus, we can conclude that a small mini-batch size, but not too small, will result in a higher accuracy overall and require roughly the same amount of training time needed. Hence it will be ideal to select the training batch of 53 in order to balance out the time taken for the best ending test error it produces. So, it is most likely that for most problems, using batch gradient descent of a descent small mini-batch size will definitely be the best way to create the ideal neural network that performs the most efficient.

References

Diagram 1 - https://cs231n.github.io/neural-networks-3/