The University of Hong Kong

ELEC6604 Neural Networks, Fuzzy Systems and Genetic Algorithms

Assignment 2

Lecturer: Dr. G. Pang

Student Name: Zhao Yunqing

UID: 3035541156

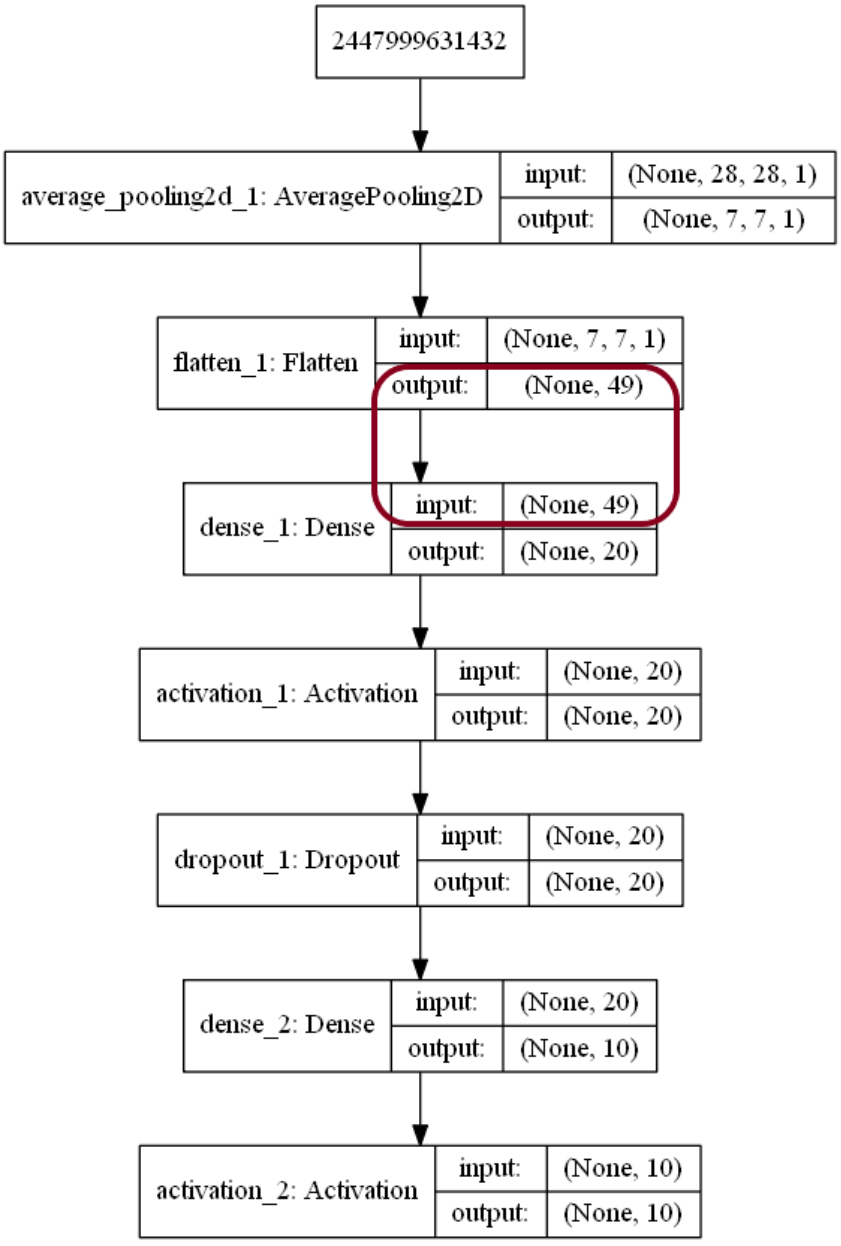
12/02/2019 Hong Kong

Contents

Part 1 Keras Framework

1.1 One Hidden Layer

In this part, I use the Keras framework to build my neural network model, firstly, the one hidden layer with 20 neurons. The network detail is as Graph 1. (Some parameters will be changed later, just see the structure.)



Graph 1
Visualize the Structure of Network

To meet the need of 7*7 input, I first use the average pooling to get the

mean value for every 4*4 pixels, using stride = (4, 4) and then flatten the input pixels to finally get the input shape = (49,).

1.1.1 Vary the hidden layer nodes

As we can see from the table 1, under the condition of only one hidden layer, the training result will be various by changing the neurons of the hidden layer, in this case, the learning rate is set to **0.01**.

Neurons	Train Loss	Train Acc	Val. Loss.	Val. Acc.	**Test Loss**	**Test Acc**
5	0.0707	0.4633	0.0685	0.5257	**0.0684**	**0.527**
10	0.0597	0.5842	0.0566	0.6448	**0.0565**	**0.6411**
20	0.0527	0.6591	0.0499	0.6996	**0.0497**	**0.7038**
45	0.0435	0.7430	0.0408	0.7762	**0.0410**	**0.7707**
200	0.0350	0.8116	0.0327	0.8324	**0.0329**	**0.8308**
2550	0.0364	0.8114	0.0335	0.8289	**0.0338**	**0.8290**

Table 1a
Performance of Different Hidden Layer Nodes (Training Epoch=**200**)

Hence, we can clearly see that to some extent, the more neurons in the hidden layer, the higher accuracy achieved by the neural network. When the number of neurons is low enough, the no-good result shown by the test set.

This is called "**Underfitting**" in Machine Learning field, which means the less experience learnt from the given dataset, or say, weakness learning ability of the network. In this case, we should add the number of neurons in the hidden layer to strength the learning ability of our artificial neural network.

However, if we set the number of neurons extremely high (e.g. 2550 as above), relatively bad results will be given, the loss value increases, and accuracy decreases. From the table 1 we know that the great results will be at the training set, and the decreasing trend of accuracy at test set. Given that the trend, we may conclude that the abnormal high number of neurons will get further poor result like this.

It is because the **"Overfitting"** of the training data set, which means the neural network "Remember" the regulations of the training set, but very slow convergence speed of loss or parameters of network, even the loss

will increase at the test set conversely. Thus, we can only set the parameters of the neural network in a proper degree.

1.1.2 Extra Discussion

In the above Training process, I introduced the layer of Dropout, which is a method of preventing overfitting, by decaying the weights of specified proportion (0.2 in this case) of the connections between the neurons. The test result without Dropout layers is as follows.

Neurons	Train Loss	Train Acc	Val. Loss.	Val. Acc.	**Test Loss**	**Test Acc**
5	0.0735	0.4358	0.0735	0.4392	**0.0733**	**0.4406**
10	0.0640	0.5599	0.0637	0.5644	**0.0633**	**0.5626**
20	0.0516	0.6485	0.0506	0.6545	**0.0505**	**0.6639**
45	0.0421	0.7654	0.0408	0.7786	**0.0411**	**0.7742**
200	0.0357	0.8129	0.0341	0.8261	**0.0323**	**0.8371**
2550	0.0323	0.8299	0.0305	0.8362	**0.0327**	**0.8338**

Table 2b
Performance of Different Hidden Layer Nodes (Training Epoch=**200**)

We can easily draw the conclusion from Table 1b that without Dropout, the performance decays of the whole neural network, when neuron is of a small number, while better when that is of a relatively big number.

I think the reason is because the network layer is naïve, and if we insist to use Dropout, it will decrease the complexity that was simple originally (less neurons), thus make the result even worse when compared to Table 1a .

But when there are too many neurons, the Dropout layer works, we can see from the last several rows of Table 1b.

However, if I set the parameters of the Dropout layer so high (normally $\geqslant$ 0.5), the total result will not be good, even worse.

All in all, for further convenience, I **will not** use the Dropout Layer based on the potential decay of the network performance as I mentioned above, just in this Assignment 2.

1.1.3 Vary the Learning Rate

Learning rate can be demonstrated to the speed of learning of our neural network, different learning rate can be expressed to different speed when training the parameters, that we want to achieve.

However, you cannot change the learning rate causally, because that will introduce some potential risks, constrained by many factors.

In my training record, firstly I simply use the Learning Rate = **0.005** (We recommend not high value as our learning rate primarily), and then **0.01**, **0.1, 0.5**, **1.0**, **5.0, 50, 100, 200** to see the difference among them.

In this case, every training process is under the condition of **Epoch = 200**, number of **Hidden Layer Neurons** equals to **45**.

L.R.	Train Loss	Train Acc	Val. Loss.	Val. Acc.	**Test Loss**	**Test Acc**
0.005	0.0623	0.6061	0.0616	0.6150	**0.0616**	**0.6064**
0.01	0.0421	0.7654	0.0408	0.7786	**0.0411**	**0.7742**
0.1	0.0176	0.8840	0.0165	0.8907	**0.0162**	**0.8966**
0.5	0.0104	0.9341	0.0101	0.9362	**0.0096**	**0.9388**
5	0.0046	0.9730	0.0065	0.9573	**0.0049**	**0.9522**
50	0.0194	0.8601	0.0192	0.8573	**0.0218**	**0.8542**
100	0.1802	0.0989	0.1805	0.0975	**0.1808**	**0.0958**
200	0.1793	0.1035	0.1784	0.1081	**0.1794**	**0.1028**

Table 2
Performance of Different Learning Rate (Training Epoch=**200**)

As above, different learning rates result in the respective answer, the result indicate that we should set appropriate learning rate.

Small learning rate express the meaning of slower learning speed of model, by the way, in this case, since our epoch number is only 200, so the result is no-good, and the model may has not reached to the global optimum point yet.

And higher learning rate may perform better in this situation, due to the limited iterations (e.g. compare learning rate equals 5 and 0.5).

Hence, if we set larger number of training epoch, the result will be better in the case of the small learning rate (Table 3 and Table 4).

Epoch	Train Loss	Train Acc	Val. Loss.	Val. Acc.	Test Loss	Test Acc
200	0.0421	0.7654	0.0408	0.7786	**0.0411**	**0.7742**
2000	0.0175	0.8856	0.0165	0.8928	**0.0161**	**0.8962**

Table 3

Performance of Different Learning Rate (Learning Rate = 0.01)

Epoch	Train Loss	Train Acc	Val. Loss.	Val. Acc.	Test Loss	Test Acc
200	0.0104	0.9341	0.0101	0.9362	**0.0096**	**0.9388**
2000	0.0044	0.9746	0.0064	0.9587	**0.0059**	**0.962**

Table 4

Performance of Different Learning Rate (Learning Rate = 0.5)

So, the epoch should be set suitable as well, as an important hyper parameter when training our network.

I also assume that the bigger learning rate may be a supplement of the weakness of shallow neural network by its simple structure, thus the bigger learning rates show the better result generally under the small iteration condition, just an individual perspective.

Finally, the bigger learning rate means the faster learning speed. However, infinite increasing the learning rate will not bring the perfect result. The loss value may explode by setting the big learning rate, or it will simply lead to the oscillation in the learning process, or our model may fall into the local optimum. (Table 5)

L.R.	Train Loss	Train Acc	Val. Loss.	Val. Acc.	Test Loss	Test Acc
0.5	0.0104	0.9341	0.0101	0.9362	**0.0096**	**0.9388**
50	0.0194	0.8601	0.0192	0.8573	**0.0218**	**0.8542**
100	0.1802	0.0989	0.1805	0.0975	**0.1808**	**0.0958**
200	0.1793	0.1035	0.1784	0.1081	**0.1794**	**0.1028**

Table 5

Performance of Different Learning Rate (Epoch = 200)

As a conclusion, it is recommended that the smaller learning step to be used primarily, which can prevent falling into traps and oscillation, then we can optimize it by gradually increasing the learning rate. Then we can gradually increase it to explore the best value.

1.2 Double Hidden Layers

Firstly, I visualize the structure of the neural network. The network uses the 2 hidden layers, which is deeper than before, it may be equipped higher learning ability. The detail of the double hidden layer network is as below, some parameters will be changed later.

Layer (type)	Output Shape	Param #
average_pooling2d_1 (Average	(None, 7, 7, 1)	0
flatten_1 (Flatten)	(None, 49)	0
dense_1 (Dense)	(None, 5)	250
activation_1 (Activation)	(None, 5)	0
dense_2 (Dense)	(None, 5)	30
activation_2 (Activation)	(None, 5)	0
dense_3 (Dense)	(None, 10)	60
activation_3 (Activation)	(None, 10)	0

Graph 2
Visualize the Structure of Network

1.2.1 Varying the Hidden Layer Nodes:

Similarly, I set the various hidden layer nodes, in different network layers, forming different combinations (Table 6). In the following tables, the pair (A, B) means the number of A neurons in hidden layer 1, and number of B neurons in hidden layer 2.

Same as before, the fewer hidden layer nodes, the weaker learning ability, however not that the more is better, which can lead to overfitting, being expressed with slowly accuracy up or no change or going down directly.

Neurons	Train Loss	Train Acc	Val. Loss.	Val. Acc.	**Test Loss**	**Test Acc**
(5,5)	0.0204	0.8667	0.0193	0.8726	**0.0192**	**0.8742**
(10, 15)	0.0116	0.9258	0.0114	0.9262	**0.0110**	**0.9281**
(25, 25)	0.0075	0.9520	0.0083	0.9464	**0.0077**	**0.9498**
(45, 30)	0.0069	0.9574	0.0079	0.9489	**0.0071**	**0.9544**
(200, 200)	0.0063	0.9616	0.0072	0.9530	**0.0075**	**0.9525**

Table 6
Performance of Different Hidden Layer Nodes Pair (Training Epoch=200, lr = 0.5)

1.2.2 Choice of Gradient Descent Process

Since in One Hidden Layer part, we get the relatively good result when learning rate equals to 0.5, and 0.5 is a reasonable value as well, so I choose to use 0.5 as learning rate in this part, and epoch is 200.

From Table 6 we find the neuron pair reflect the effect when training the network, the result shows that we should also choose appropriate neurons as well, from above, maybe the (45, 30) is the most suitable.

Besides, we may also choose a special kind of method for gradient decent "**Momentum**", which can speed up the convergence of the network.

Neurons	Train Loss	Train Acc	Val. Loss.	Val. Acc.	**Test Loss**	**Test Acc**
(5,5)	0.0187	0.8790	0.0186	0.8787	**0.0177**	**0.8849**
(10, 15)	0.0098	0.9366	0.0102	0.9343	**0.0099**	**0.9366**
(25, 25)	0.0046	0.9718	0.0080	0.9507	**0.0072**	**0.9546**
(45, 30)	0.0025	0.9862	0.0074	0.9543	**0.0068**	**0.9583**
(200, 200)	0.0005	0.9970	0.0062	0.9622	**0.0067**	**0.9585**

Table 7
Performance of Network with Momentum (Training Epoch=200, lr = 0.5)

From Table 7, we may find that in the Momentum mode, in limited number of iterations, we can get better result in the same epoch, since the speed of convergence of our ANN is faster, with the right training direction and restrain the oscillation of the training process.

Hence, in next training process, I will choose the learning rate equals to 0.5, and using Momentum gradient descent mode.

1.3 Three Hidden Layers

In this part, as I mentioned before, I will use primary learning rate 0.5 and using Momentum mode, the training process will last 200 iterations as well.

Similarly, I set the various hidden layer nodes, in different network layers, forming different combinations (Table 8).

In the following tables, the pair (A, B, C) means the number of A neurons in hidden layer 1, and number of B neurons in hidden layer 2 and C in hidden layer 3.

Neurons	Train Loss	Train Acc	Val. Loss.	Val. Acc.	**Test Loss**	**Test Acc**
(5, 5, 5)	0.0212	0.8613	0.0208	0.8642	**0.0203**	**0.8687**
(10,15,15)	0.0097	0.9378	0.0105	0.9345	**0.0101**	**0.9356**
(25,25,25)	0.0049	0.9694	0.0083	0.9498	**0.0083**	**0.9487**
(45, 30, 30)	0.0031	0.9814	0.0079	0.9528	**0.0079**	**0.9536**
(200, 200, 200)	0.0003	0.9980	0.0062	0.9625	**0.0077**	**0.9546**

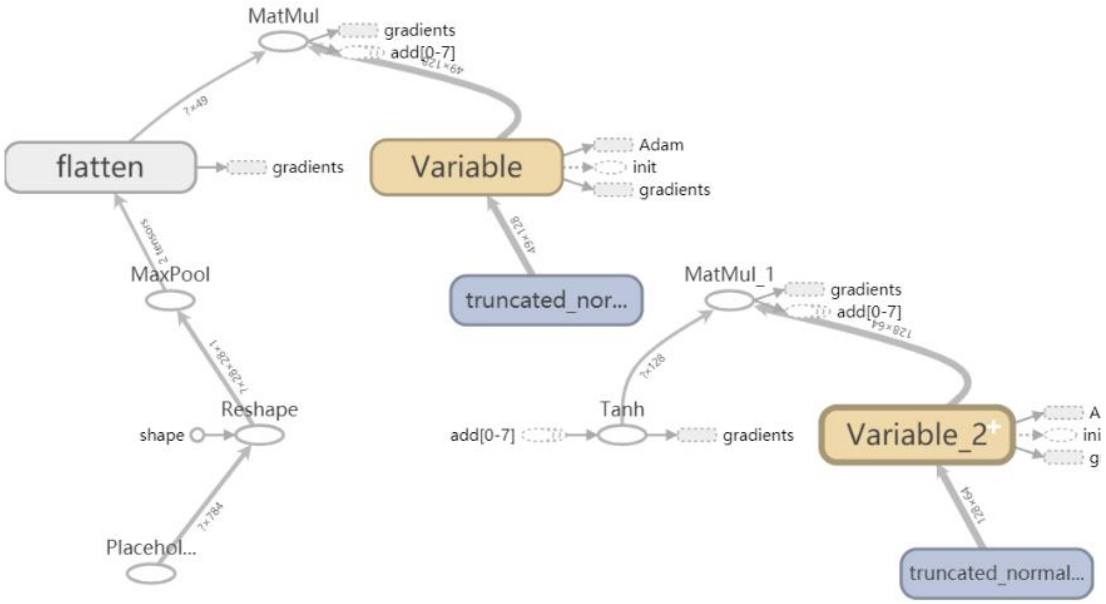
Table 8
Performance of Network with Momentum (Training Epoch=200, lr = 0.5)

From the Table 8 as above, we still get the good result, however a little weaker performance when compared to Table 7, with 2 hidden layers. I assume that it has the overfitting to some extent, or just normal fluctuation of the result, maybe next experiment result will be better.

Part 2 TensorFlow Framework

In this part, I used the 1, 2, 3 hidden layers in my network, and get the similar result of part 1, for detail code, please see the attachment file.

2.1 Visualize the Network Structure

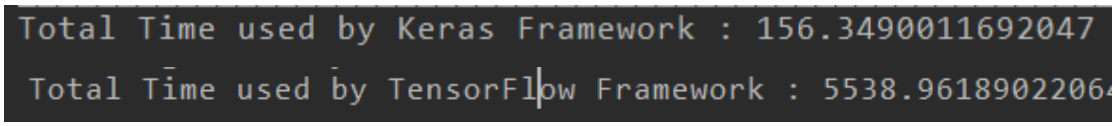


Graph 3 Part of the Network

I use the **TensorBoard** to visualize the whole network, with three hidden layers and a SoftMax activated output layer. Due to the Limited space, please refer to the attachment file for the whole Graph.

2.2 Time Used Comparison

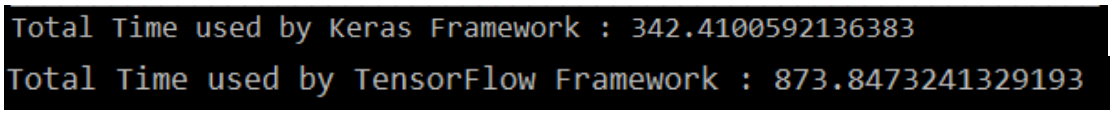
Firstly, I deploy the code on my Personal Computer, equipped with I7-7700HQ, and GTX1050ti, the consumption of time by Keras or TensorFlow is as follows: (In this case, I use CPU version Keras and TensorFlow running in my local computer)



Graph 4

The reason why time used by TensorFlow is far more than that of Keras may be the loop by loop, which means the complexity of $O(n^2)$.

Then, I upload my python code onto the remote GPU server, with 1080ti equipment. However, the result is not that good of Keras, but speed up deeply by TensorFlow.



Graph 5

Here is the problem. We can clearly see that apparent positive difference between TensorFlow case. The reason is still not clearly, but I find that the more complex the network, the more apparent difference, esp. Keras framework, maybe the inner mechanism caused the result.

The similar situation posted by someone on the Stack Overflow. (https://stackoverflow.com/questions/42097115/keras-tensorflow-backend-slower-on-gpu-than-on-cpu-when-training-certain-netwo)

Besides asking for answer from Internet and make self-thinking, **I will also push the question during the Assignment to Discussion Forum via HKU Moodle, inviting my fellow classmates to join the discussion.**

So, we may consider using the normal neural network in local PC, and more complex one on remote stronger GPU server.

Part 3 Summary and Conclusions

In this Assignment 2 , I set different number of hidden layers, different number of neurons(pair) in various layer, discussed some tricks (like Momentum and Dropout Layer) used which can promote the performance

of the whole network, varied different learning rate to discover the relatively best one.

I found that there is no solid definition of “Best Parameters”, the best parameters can only be found at the process of training neural network, making comparison among different settings, and make your unique decision. However, in this process, there are also some general rules we should obey to help us in our training and experiment or scientific research.

The first one is to set the appropriate learning rate, if you have no idea, just start from a smaller value, then gradually make it bigger to find the most suitable one. In this process, smaller learning rate means the weaker learning ability, under the condition of limited epoch, a relatively big learning rate may get closer to the optimum point. While under the enough big iteration number, if we hold on the big value of learning rate, it may fall into the local optimum value, and cause the oscillation and then never meet the best answer.

Some extra tools can be of help like Dropout Layer, but I also suggest smaller value, taking “robust” and “stable” of the structure into consideration.

When involved with depth of our network, we may find that when network become deeper, the time needed to finish training grow as well, which means the more complex and larger scale of parameters to be fit.

Theoretically speaking, deeper network or network with more parameters will strength up the learning performance, however we must notice that the more parameters or the deeper network structure, the higher risk of overfitting and unstable state. Hence, we should also deal with this and find the best parameters or use some other models like Convolutional Neural Network to pursuit higher accuracy.