A0191508A Qiu Hai
CS5242 Neural Network and Deep Learning 2018/18 Sem 1
Assignment 1

Question 1:

For a two hidden layers fully connected neural network with identity function as activation function:

$$O_1 = \sigma(X * W_{0,1} + b_1) = X * W_{0,1} + b_1 \quad (1)$$

$$O_3 = ((X * W_{0,1} + b_1) * W_{1,2} + b_2) * W_{2,3} + b_3 \quad (2)$$

$$= X * W_{0,1} * W_{1,2} * W_{2,3} + b_1 * W_{1,2} * W_{2,3} + b_2 * W_{2,3} + b_3$$

For a fully connected NN without hidden layers

$$O = \sigma(X * W + b) = X * W + b \quad (3)$$

So in order to make the coefficients of the equations (2) and (3) regarding X the same

$$W = W_{0,1} * W_{1,2} * W_{2,3} \quad , \quad b = b_1 * W_{1,2} * W_{2,3} + b_2 * W_{2,3} + b_3$$

To calculate the bias for the second NN, we can set zeros as input for the first NN. In this case, the output will be the bias for the second NN:

$$O_3 = ((\,[0, 0, ...0] * W_{0,1} + b_1) * W_{1,2} + b_2) * W_{2,3} + b_3$$

$$= b_1 * W_{1,2} * W_{2,3} + b_2 * W_{2,3} + b_3$$

$$= b$$

Question 2:

My first observation was that in order to get the best training result, pick a proper learning rate is extremely important. If the learning rate is too high, you may see a steep drop of the cost at the beginning, then the learning curve becomes very choppy and fail to converge, no obvious improvement can be observed at the end. This is because a high learning rate will cause the gradient descent overshoot and wobble around the minimum cost. On the other hand if the learning rate is too low, the gradient descent will be slow and the network will take longer time to train.

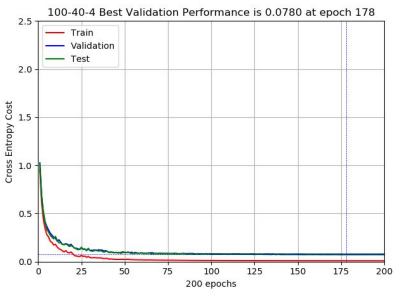
In order to balance the training result and training time, I adopted a step decay scheduled method to gradually reduce the learning rate along the training process, so the learning speed can be fast at the beginning and we can still get a result close to the minimum at the end. This method can be easily implemented by:

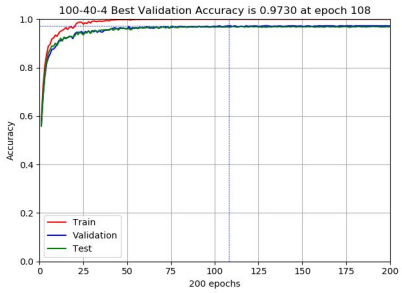
```
learning_rate = initail_lr* drop^floor(epoch / epochs_drop)
```

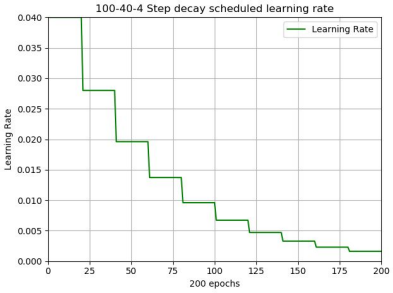
More advanced, an adaptive learning rate can be adopted to adjust the weights according to the weight changes in previous epochs.

Out of the original testing data set, I randomly choose 40% of them as validation data, and 60% as testing data. I choose a batch size of 32, which is also a trade off between speed and performance (too big, the change of gradient will be generalized; too small, take longer time)

1) NN 14-100-40-4:

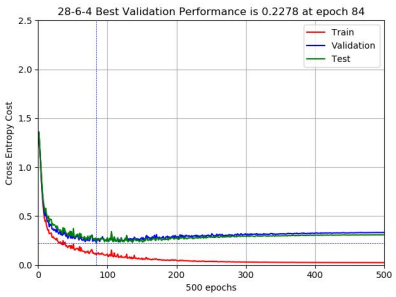


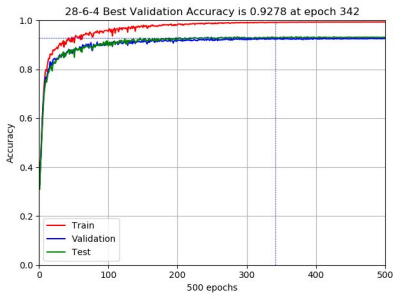


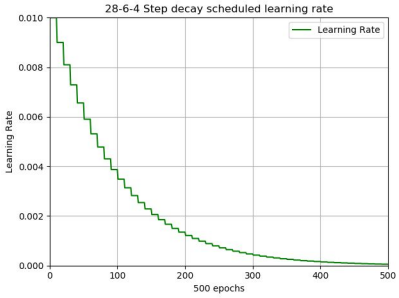


This network give us the best performance among the three networks, and also the shortest time to train. Thanks to the simple inner structure, the gradient changes won't vanish through back propagation, which give us a network easy to train.

2) NN 14-28*6-4:

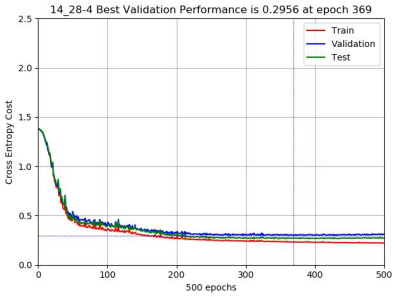


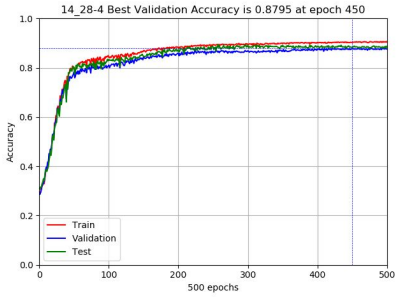


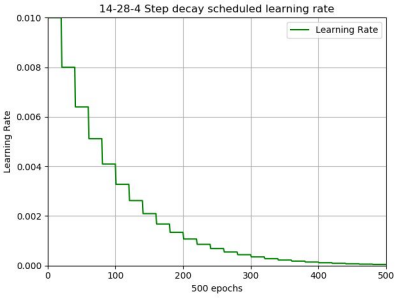


We can observe the training cost starts to overfit after 84 epochs. which is an indication to end training. On the other hand, we can also observe that the best validation accuracy performance occured 258 epochs after the min validation cost occurred, if we end training too early, we won't reach the best accuracy. Keeping in minds that the cost and accuracy is calculated differently, so sometimes it's better not to stop training when the training cost starts to overfit.

3) NN 14-14*28-4:







This network give us the worst performance and also the longest time to train (and really easy to stuck into a local minimum stopping any improvement), which is counter-intuitive at the first glance since people usually think a more complexed NN should be able to handle more challenging problems, not to mention the easy ones. But the fact is that more layers bring more problems in this case.

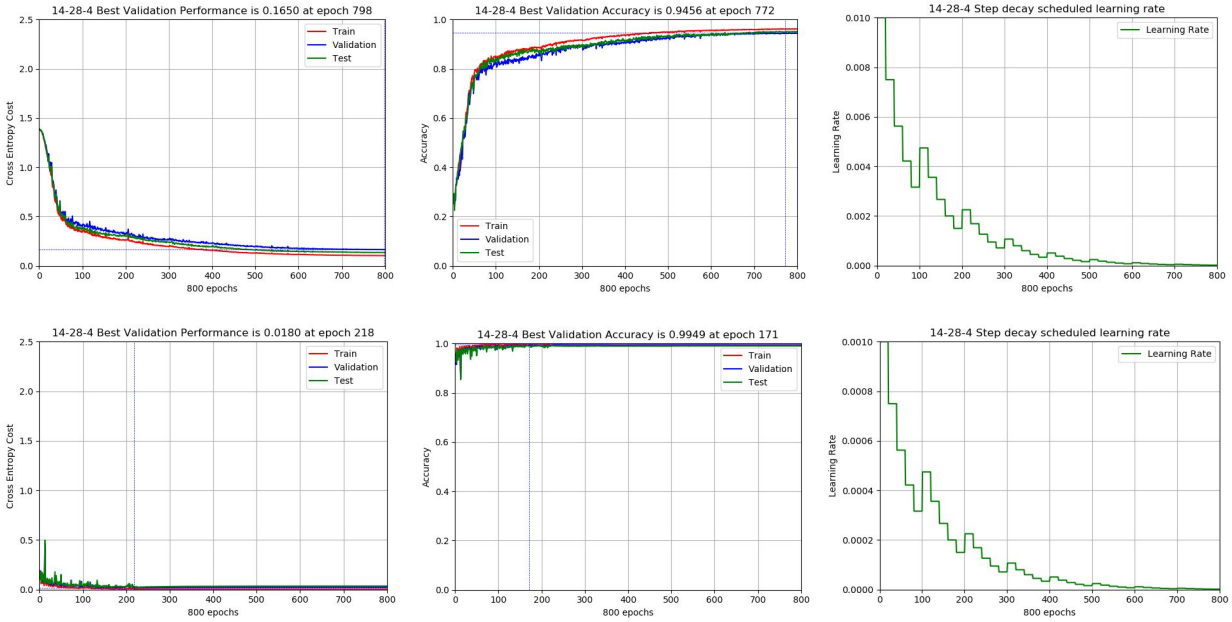
In order to get satisfied result, the initial weights and bias, as well as the learning rate should be chosen carefully, otherwise after input being propagated through this many layers, the output can be easily overflow or underflow. To avoid overflow, we need to preprocess the softmax

inputs by subtracting each input by the value of the maximum input, so all the inputs will be smaller than or equal zero, preventing the exponential function output an extremely large value cannot be represented by float number.

$$softmax(X) = softmax(X - max(X)), \quad \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{-m}}{e^{-m}} \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - m}}{\sum_j e^{x_j - m}}$$

Vanish gradient problem is also critical in this network, which means it really hard to learn and tune the parameters of the earlier layers in the network. A slight adjustment in the early layer will either vanish or explode in the output, make the gradient descent hard to find the minimum, thus causing the worst result.

3.1) NN 14-14*28-4 (with restart):



In this part, I tried an experimental method called “warm restart”. While the learning rate is decayed on schedule, we increase the base of learning rate cyclically, It helps because, if we get stuck on saddle points, increasing the learning rate allows more rapid traversal of saddle point plateaus[1]. The learning rate is calculated by:

```
l_rate = init_lr*2^floor(epoch/cyc)* drop^floor(epoch / epochs_drop)
```

I trained the network with 4 consecutive rounds of 800 epochs (total 3200 epochs, I showed the graph of the first and last round). The final result is surprisingly good, we got a validation accuracy of 0.9949 which is the best out of three, and no obvious overfitting is observed. So with proper training method, this network may able to discover some hidden logic between input and output which cannot be discovered by a NN with simple structure.

[1] Understanding Learning Rates and How It Improves Performance in Deep Learning https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10