Deep Learning - Homework 2

Group #14: Ellie Li (yl3883), Qingyuan Dong (qd2145) -

March 12, 2019

1 Impact of different activation functions and optimization on learning

In the sample code *example_5_hidden_layer.jpynb*

(a) what is the exact number of parameters we are trying to learn?

Total number of parameters:

$$785 * 400 + 401 * 200 + 201 * 100 + 101 * 50 + 51 * 25 = 420,625$$

(b) use different activation functions in your architecture (e.g. sigmoid for 1st layer, tanh for 2nd layer, sigmoid for 3rd layer, reLU for 4th layer, and leaky reLu for 5th layer) and assess its impact on accuracy?

The different activation functions in our architecture do not work well. From the example code (see *Q1-architecture.ipynb* for detailed code), we use the loss_2 function, which is the cross-entropy.

The accuracy of original model was about 0.9824 and the execution time was 704.606 seconds:

```
Optimization Finished!
Test Accuracy: 0.9824
Execution time (seconds) was 704.606
```

After changing our architecture, we got a worse accuracy and a slower execution time, which was 0.8793 and 511.172 respectively:

```
Epoch: 097 cost function= 0.2327641  Validation Error: 0.11379998922348022
Epoch: 099 cost function= 0.2327764  Validation Error: 0.11320000886917114
Optimization Finished!
Test Accuracy: 0.8793
Execution time (seconds) was 511.172
```

This shows that the model architecture is an art. And it plays an important role in our model.

(c) for part (b) keep the same architecture, just use different optimization routine and assess its impact on accuracy?

In this case, we chose to use the Adam optimization, and it did not work well. The accuracy of original model was about 0.8646 and the execution time was 466.625 seconds. After changing our architecture, we got a worse accuracy and a slower execution time. Which was 0.8871 and 713.129 respectively:

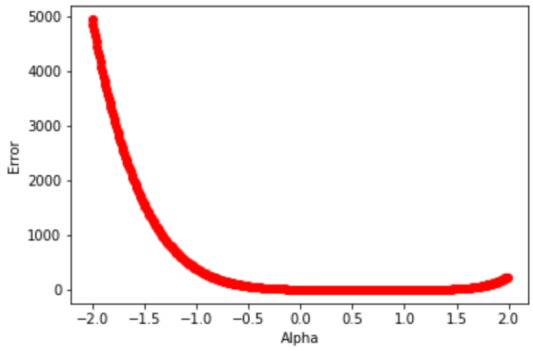
```
Epoch: 099 cost function= 2.3025854  Validation Error: 0.9042000025510788
Optimization Finished!
Test Accuracy: 0.098
Execution time (seconds) was 547.951
```

2 Visualization of the lost function

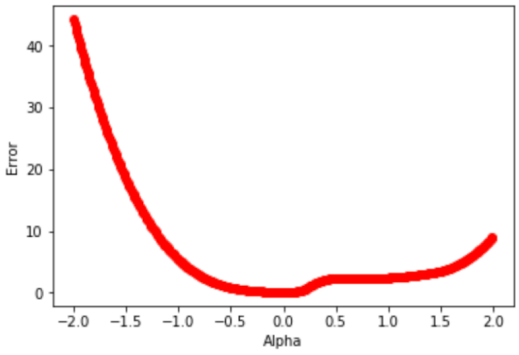
Use sample code example *5_layer_interpolation.jpynb* and architecture and optimization routine in parts (b) & (c) of **Problem 1** to assess the loss function by interpolation, namely

(a) impact of different architecture on the loss function surface

After implementing the new architecture, we find there is a turning point after $\alpha = 0.0$:



(a) Original Achitecture

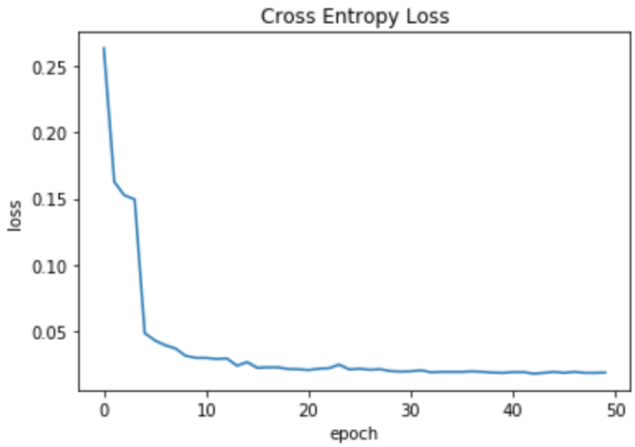


(b) New Achitecture

(b) assessing the path traveled through the loss function having same architecture but using different optimization routine

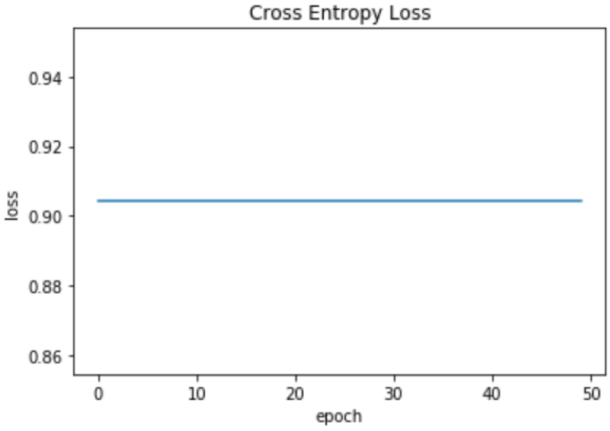
Using the Gradient Optimization:
From the graph below, we know that the Cross-Entropy Loss changed a lot from epoch 0 to 10. After 10, it became smoother.

```
Optimization Finished!
Test Accuracy: 0.9761
Execution time (seconds) was 481.540
```



Using the Adam Optimization:
From the graph below, we know that the Cross-Entropy Loss is stable all the time.

```
Test Accuracy: 0.098
Execution time (seconds) was 530.491
```



(Detailed code please see attached file *Q2-visualization.ipynb*)

3 Convolutional Neural Networks

In the sample code *example_CNN.jpynb*

(a) what is the number of parameters we are trying to learn?

The neural network contains 2 convolutional layers with max pooling and 1 fully-connected layer and 1 output layer. First consider each batch:

- *Convolution 1:* Input size is $28 * 28 * 1$, output size is $14 * 14 * 32$, and filter size is $5 * 5$, with stride $= 1$. Pooling process includes no new parameters. So the number of parameters needs to learn is:

$$5 \times 5 \times 1 \times 32 + 32 = 832$$

- *Convolution 2:* Input size is $14 * 14 * 32$, output size is $7 * 7 * 64$, and filter size is $5 * 5$, with stride $= 1$. So the number of parameters needs to learn is:

$$5 \times 5 \times 32 \times 64 + 64 = 51,264$$

- *Fully Connected:* Input size is $7 * 7 * 64$, output size is $1024 * 1$, and weight size is the same as input $7 * 7$, with stride $= 1$. So the number of parameters needs to learn is:

$$7 \times 7 \times 64 \times 1024 + 1024 = 3,212,288$$

- *Output:* Input size is $1024 * 1$, output size is $10 * 1$. So the number of parameters needs to learn is:

$$(1024 + 1) \times 10 = 10,250$$

Considering we have a size of batch $= 200$, the total number of parameters this neural network needs to learn is:

$$200 \times (832 + 51,264 + 3,212,288 + 10,250) = 654,926,800$$

(b) add one more convolutional layer with max pooling?

I added one more convolutional layer "convolutional_layer_3" between the "convolutional_layer_2" and "fully_connected". I used a $5 * 5 * 128$ size filter, and keep everything else the same as before. (Please see attached *Q3-cnn-addlayer.jpynb* for detailed code.)

(c) what was the impact of one extra convolutional layer on accuracy?

The extra convolutional layer didn't improve the accuracy of CNN. I got an accuracy of 99.24% after training for 50 epochs and the execution times is 13524.98s, while the original CNN has a 99.42% accuracy with only 6886.74s execution time.

4 Batch Normalization

In the sample code *example_CNN_Batch_Normalization.jpynb*, what was the impact of batch normalization of learning (speed & accuracy)?

I used *example_CNN.ipynb* and *example_CNN_Batch_Normalization.ipynb* to examine the impact of batch normalization of learning in CNN. I carefully checked each parameter of the two model and to keep them the same except for batch normalization. For time saving, I run 10 epochs for both CNN, and below attached is the training track, test accuracy, and the running time of both model:

```
Epoch: 0001 cost = 0.262095391
Validation Error: 0.014800012111663818
Epoch: 0002 cost = 0.054710089
Validation Error: 0.011399984359741211
Epoch: 0003 cost = 0.037006298
Validation Error: 0.010399997234344482
Epoch: 0004 cost = 0.026399111
Validation Error: 0.008800029754638672
Epoch: 0005 cost = 0.020849007
Validation Error: 0.008199989795684814
Epoch: 0006 cost = 0.016803488
Validation Error: 0.008400022983551025
Epoch: 0007 cost = 0.013074324
Validation Error: 0.009199976921081543
Epoch: 0008 cost = 0.012141597
Validation Error: 0.007799983024597168
Epoch: 0009 cost = 0.010560767
Validation Error: 0.007600009441375732
Epoch: 0010 cost = 0.008557542
Validation Error: 0.0073999762535095215
Optimization Done
Test Accuracy: 0.9926
Execution time was 1944.983
```

(a) CNN without Batch Normalization

```
Epoch: 0001 cost = 1.813049037
Validation Error: 0.4959999918937683
Epoch: 0002 cost = 0.303314686
Validation Error: 0.013599991798400879
Epoch: 0003 cost = 0.045839630
Validation Error: 0.009000003337860107
Epoch: 0004 cost = 0.027348253
Validation Error: 0.008599996566772461
Epoch: 0005 cost = 0.019954986
Validation Error: 0.00959998369216919
Epoch: 0006 cost = 0.014335834
Validation Error: 0.00959998369216919
Epoch: 0007 cost = 0.011934711
Validation Error: 0.009199976921081543
Epoch: 0008 cost = 0.011038586
Validation Error: 0.007200002670288086
Epoch: 0009 cost = 0.008451579
Validation Error: 0.008000016212463379
Epoch: 0010 cost = 0.008287311
Validation Error: 0.0067999958992004395
Optimization Done
Test Accuracy: 0.991
Execution time was 2732.430
```

(b) CNN with Batch Normalization

As shown in the figures above, using batch normalization in each layer has increased the execution time from 1945s to 2732s, around 35% of the original running time, while the final test accuracy after 10 epochs of learning fell slightly from 99.26% to 99.1%. (Detailed codes are attached as *Q4-cnn.ipynb* and *Q4-cnn-batch.ipynb*)