CS 515 – Deep Learning

Spring 2020

PS1 – 100pts

Goal: Hands-on experience with gradient descent and different activation and cost functions.

Software: Use Python (maybe Google Colab) and the Scikit-learn library. If you must use something else, please let me know.

Submit: Submit a single python notebook or its share link (if Colab notebook) and a 1-2 page report (filling this document).

Please submit at Sucourse.

Q1 – 50pts – Multiple linear regression

Given the data given in data1.csv, where `y = 5*x1+20*x2-10+np.random.uniform(-1, 1)`, train a **single layer NN** (no hidden node=single layer of weights) to learn the function from the given samples, using gradient descent. The number of output nodes and activation is determined by the problem. You are expected to i**mplement the batch gradient descent code** yourself. Add code to display the error as a function of time.

Try learning rates such as 0.01, 0.001, 0.0001 etc. as suitable; if the weights are diverging, stop. Once you got y_hat values that are close (close enough), answer the following:

1) 40pts - What are the learned weights, corresponding MSE and the learning rate you used to find them, at what epoch?

Learning rate λ	**MSE (root mean sq. error) on train data**	**w1,w2, b found**	**Number of epochs**
1	6209.92	28.49, 120.51, 213.95	10000
0.1	0.54	14.63, 61.64, 124.57	10000
0.01	0.54	14.63, 61.64, 124.57	10000
0.001	0.54	14.63, 61.64, 124.57	10000
0.0001	368.69	13.48, 53.59, 107.64	10000

2) 5pts- What happens when λ is too large?

Since w1, w2, and b diverge from their expected values, mse becomes relatively higher when learning rate is smaller. This can be seen from the figure above. In the first row, learning rate is 10 times bigger than second row's, but as it can be seen w values and b are away from the desired values.

3) 5pts - If you can't find the exact parameters (w1=5, w2=20 and b=-10), **what could be the reason(s)?**

Even though I applied L2 regularization, the result was not changed. I think my model suffers from high bias issue. It is not learning the data enough. Therefor its mean squared error is high. That's why I couldn't get the correct values for w and b. High bias issue can be solved with a couple of methods. The first thing to try should be trying a bigger network with more layers and hidden units. The second thing is to train the model longer by using more advanced algorithms like ADAM, RMSprop. The last but not least thing to try is to search better NN architecture.

Q2 – 50pts – Digit Classification

Using the **MNIST** data (28x28 greyscale scaled and centered digit images) available under Resources/HW-data, build a digit classifier. Here I am **not** expecting you to expand/use your own gradient descent code, but instead to use Scikit-learn library (where the MLP call will be one line).

This dataset is very famous and available online with the same, specified **test subset** (10,000 samples). You can read about it if you wish.

You should use:

- **a single hidden layer (with 50-100 hidden units) with Logistic or ReLu activations**
- **10 output nodes with softmax activation with categorical cross-entropy**
- **stochastic gradient descent with mini-batch size of 1.**

1) 25pts – Submit your code and fill the following table according to your experiment. Also specify any other meta-parameter (not learned ones, but the ones you set) also.

Learning rate λ	Number of hidden units	Mini-batch size	Any other parameters to mention (regularization etc.)	Number of iterations	BEST Classification accuracy on TEST data
0.1	10	1	-	10	94.15%
0.01	10	1	-	10	92.32%
0.001	10	1	-	10	88.10%
0.1	10	1	Nesterov	10	94.01%
0.1	100	1	-	10	98.00%

2) 10pts – Experiment (just play with) with using Logistic activation (as compared to ReLus) in hidden and output nodes & MSE as error function. What happens in terms of accuracy/speed?

Fitting the model described in the last row of above table just took 49.91 seconds with 98.00% accuracy on Test data. After I just change all activations to sigmoid and error function to MSE, it was 49.48 seconds with 88.6% accuracy on Test data.

As it is seen from my trials, it is not recommended to use sigmoid as hidden unit activations and not use mean squared error.

3) 5pts – Comment on your error cases: Give the outputs (P(C0=1|**x**),…,P(C9=1|**x**) for some of the error cases or your confusion matrix.

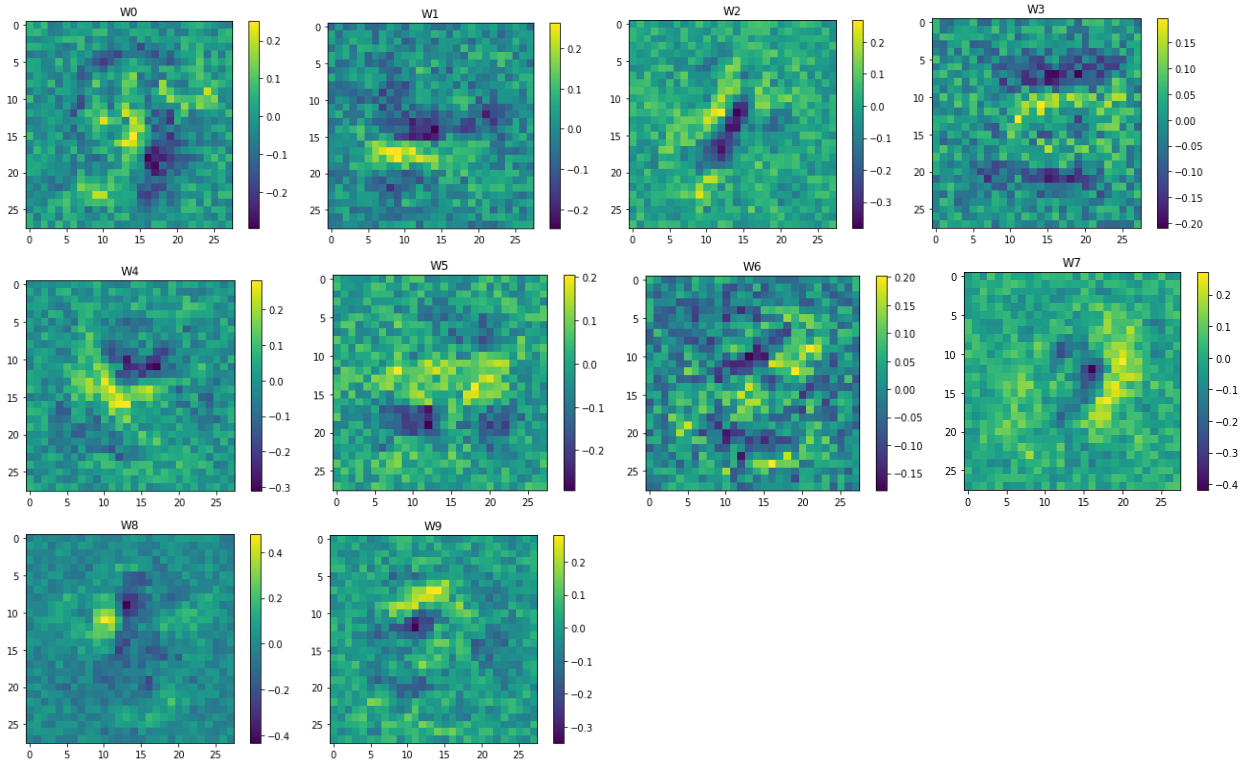
```
CM = confusion_matrix(test_labels_notCategorical, test_predictions)
CM
```

```
array([[ 970,    1,    0,    2,    2,    0,    1,    2,    1,    1],
       [   0, 1120,    3,    3,    0,    1,    3,    0,    5,    0],
       [   3,    1, 1000,   11,    4,    0,    0,    6,    7,    0],
       [   0,    0,    1, 1003,    0,    0,    0,    1,    4,    1],
       [   2,    0,    6,    1,  955,    0,    2,    3,    2,   11],
       [   3,    0,    0,   17,    2,  860,    5,    0,    5,    0],
       [   4,    3,    2,    1,    5,    5,  936,    0,    2,    0],
       [   1,    2,    7,    6,    0,    0,    0, 1001,    3,    8],
       [   3,    0,    4,    7,    2,    2,    1,    3,  950,    2],
       [   4,    4,    0,    9,    7,    3,    0,    5,    5,  972]],
      dtype=int64)
```

I looked closely to the confusion matrix of the model described in the last row of above table. It can be found in the jupyter notebook as CM label. My model mostly failed on predicting class 5 since smallest number in the diagonal of confusion matrix is in that class. The reason for that fact can be the test set is not distributed to each class equally. But I checked and it seems it is equally distributed to each class.17 error cases issue is the biggest one in confusion matrix. It stems from predicting class 5, but it gives class 3 prediction 17 times instead. So even this result is consistent with the previous one, which indicates that the current model mostly failed on predicting class 5.

4) 10pts - Display the hidden unit weights (as a 28x28 image) for some 10 nodes.

First 10 weights of hidden layer out of 100 units (the model represented in the last column of part 1's figure) can be seen below.



5) 0pts – Can you see if there are any "dead" RELU nodes (not active for any output) **or** how many ReLu nodes are active (firing) for one input sample on average, as most of them are not active.

As it can be seen the figures above, filtering of W8 and W9 is not that apparent like other Ws. That's why I can say even looking 10 weights out of total 100 weights(since I have 100 units in hidden layer in my model), just 8 is seem active.

Therefore, it can be said that smaller than 100 units would enough for hidden layer. When we look at the table in the first part of this question, keeping the same 0.1 learning rate, going from 10 hidden units to 100 only improves testing performance about 4%. Is 4% extra performance worth despite extra computation power and about 10 times higher parameters to train? It depends on your hardware and cost of time.