**Neural Networks validation with MNIST data set**

a. Implementation: First I have imported my data set and divided the data set into train, dev and test in the ratio of 70, 20 and 10. I did one-hot encoding on my categorical data. The number of records in train, dev and test after splitting happened to be 49000, 14000 and 7000. I have defined my activation functions sigmoid, ReLU and Tanh with respective calculations. I implemented a function to calculate my loss and then a function for forward propagation where we did product of weights with the outputs from neurons. I defined a function for back propagation where I am calculating gradients of function with respect to weights and bias. I have initialized other parameters like number of neurons required, learning rate and batch\_size which are adjustable. I have implemented only for single layer. And finally I calculated train and test loss.

Convergence: We find convergence when we run our model with different number of neurons, learning rate, batch size and epochs. In my case I found convergence at 19th epoch with learning rate 3, batch size 128 and number of neurons 56. I observed both my training and dev loss are comparatively low and from 20th epoch, the train and dev loss seemed to increase. This is because of change in weights for every iteration. Also if we change number of neurons, we may achieve better results but it also takes more computation power. I found this can be an ideal configuration for the given data set. Also I tested my model with less number of neurons (24) where I observed my train and dev loss dropped to 0.09 and 0.155 and later it started increasing (at epoch 18: train loss: 0.109 and dev loss: 0.209). So decreasing number of neurons didn’t give me best results.

b. In order to validate my implementation, I used XOR data. When I have run my model on XOR data, it gave me satisfactory results. My model is able to predict the correct output accurately. i.e., as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | Expected output | Actual output |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

c.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S.No. | Activation Function | Learning rate | Batch size | epochs | # of neurons | Train loss | dev loss | Accuracy |
| 1 | Sigmoid | 12 | 500 | 50 | 72 | 0.086 | 0.192 | 0.97 |
| 2 | Sigmoid | 0.3 | 200 | 19 | 56 | 0.118 | 0.111 | 0.98 |
| 3 | Sigmoid | 0.03 | 200 | 50 | 56 | 0.072 | 0.113 | 0.98 |
| 4 | ReLU | 0.3 | 200 | 25 | 50 | 0.096 | 0.103 | 0.98 |

Explanation: As shown above in the table, I tried my implementation with different configurations.

For 1st configuration, I observed gradual decrease in train and dev loss to 0.127 and 0.192 till 14th epoch and then dev loss started increasing and overall it didn’t give me best result as compared to others. This may be due to more learning rate and batch size.

For 2nd configuration, I observed my model did well with less learning rate and batch size and I have used same activation function. Interesting fact is that here I reduced number of neurons and yet my model did well.

For 3rd configuration, I decreased my learning rate further to 0.3 and I run the model for 50 epochs, I observed my loss was decreasing but at a slow rate. And even after 50 epochs, my dev loss is still more than the previous configuration where I found less dev loss at 19th epoch.

For 4th configuration, I changed my activation function to ReLU and observed it did little better than the sigmoid function. I noticed very slight change in behavior of the model compared to sigmoid. With almost similar configuration, I found ReLU did better than sigmoid.