COSE474 Deep Learning: Project #1: MLP Implementation

Project report REUTELSTERZ ELIAS

Description of the code:

In the first code fragment, I initialized and computed the different components of the 2-layer neural network (z1, p1, z2). I used the same notation as in our assignment and script. Because we use N different data points in each batch-iteration I decided to generate z1, p1, z2 as N rows with one row for every datapoint.

In the second section, the dataloss and the two L2 regularizations were calculated. Here one could discuss, that in p2 we only need the entries p2[I, y[i]] but later in the calculation of the gradient of p2 we also need the whole matrix. The total loss function contains the data loss and the regularization loss parts.

In the backpropagation section, I calculated the gradients for all data points and finally averaged them. To do this, I first had to calculate the four different gradients (b2 and W2 we already knew from the 2nd assignment). I calculated them for all data points in the order b2 -> W2 -> b1 -> W1, so that I can use the intermediate results for the next calculation. Finally, I had to add the derivative of the regularization terms to the gradients of W1 and W2 and store everything in the map grads["xxx"].

In the function train, first a randomized selection of the training data points with the size batch_size had to be determined. Then the parameters were updated.

Finally, in the predict function, we were able to call the loss function without argument y to calculate the most likely labels, since it uses our final parameters to determine the scores for the labels.

Results:

Because the validation accuracy was just as high as the training accuracy, I tried to test out how many hidden layers and iterations I can use without letting the gap be too big and still gaining test accuracy. I also increased the learning rate, so that we get away of the linear fitting and get an exponential fitting. I Additionally decreased the batch_size (but increased the iteration count in a higher relative relation) and it worked out for a better performance.

With trial and error in the fine-tuning I managed to get the rates:

- Training accuracy: 0.622
- Validation accuracy: 0.464
- Test accuracy: 0.476

while using the hyperparameters:

- hidden_size = 80
- num_iters = 5000
- batch_size = 50
- learning_rate = 1e-3
- learning_rate_decay = 0.95
- reg = 0.2

Discussion:

While working on the project, I encountered two problems whose solution strategies I had to weigh. On the one hand, I tried to solve as many calculations as possible with direct matrix calculation to get the best performance. This was difficult, because I first calculated the gradients with the individual functions and could not always insert the case distinctions into the Numpy formula. Also, in some cases I find it more descriptive to go through the rows and columns one at a time and calculate the functions one at a time. In the end I managed to get rid of all for-loops to get a good performance.

The second problem was tuning the model. Here it is difficult to develop a feeling for which change, which consequences. For example, how high the deviation between training accuracy and test accuracy should be optimally, without getting an over- or underfitted model.