ECE 521 Assignment 2

# Work Breakdown

|  |  |
| --- | --- |
| Group Member Name | Contribution Percentage |
| Jixong Deng | 33% |
| Jeffrey Kirman | 33% |
| Connor Smith | 33% |

# Part 1: Linear regression

Linear regression is performed using the following minibatch stochastic gradient descent (SGD) algorithm:

|  |
| --- |
| **Init:** **w** = **0** # Weights vector  **xj** = [1 x­j1 xj2 ... xjd]T # Single in feature  **X** = [**x1** ... **xn**] # Matrix of all in-features  **y** = [**y1** ... **yn**] # Vector of all targets  **Step:** for each epoch  Shuffle **X -> X’** # Randomly shuffle the vectors in **X** into **X’**  for each mini-batch: **X’** = [**x’(mini-batch\_size)\*i** ... **x’(mini-batch\_size)\*(i+1)**]  **gt** = ∇Ein(**w(t)**) # Calculate the gradient of the loss  **w(t+1)** = **w(t)** – η**gt** # Update the weights, where η is the learning factor  **Result:** return **w** |

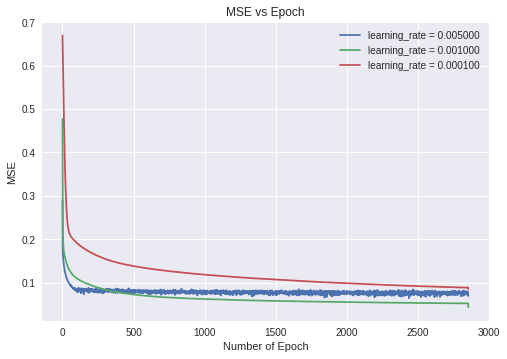
The **SGD** function (**linear\_regression2.py** in *Appendix A*) implements this. This function creates a graph in which with placeholders for input matrix **X** and target vector **y**. It then calculates the MSE loss according to the following equation:

Where λ is the decay constant. The gradients are then calculated using a function in the tensorflow API, tf.gradients. Finally, the weights are updated.

After the graph is created the function calculates the number of epochs based on the iterations and mini-batch size. It then shuffles an array of indices and then uses them to index the matrix of in-features into a mini-batch. feed\_dict is then used to feed the inputs into the placeholders and run the graph for one iteration.

## Part 1 – 1

The results of changing the learning rate (η) can be seen in the figure below. Here we can see the trade-off between higher and lower learning rates. For higher rates we see quicker convergence but at a higher MSE. For lower rates the opposite is true, where the MSE converges slower but at a lower value. Therefore, the best η should be chosen to maintain quick convergence as well as a low value of convergence. The best η in this case is 0.001.



## Part 1 – 2

The best mini-batch size in terms of training time is 500. This makes sense since each update will take less

# Appendix A – Python code

## Linear\_regression2.py

|  |
| --- |
|  |