PROJECT 2

KARISHMA JAIN

Aditya Kashyap

Problem 2: The PEGASOS algorithm receives as input two parameters:

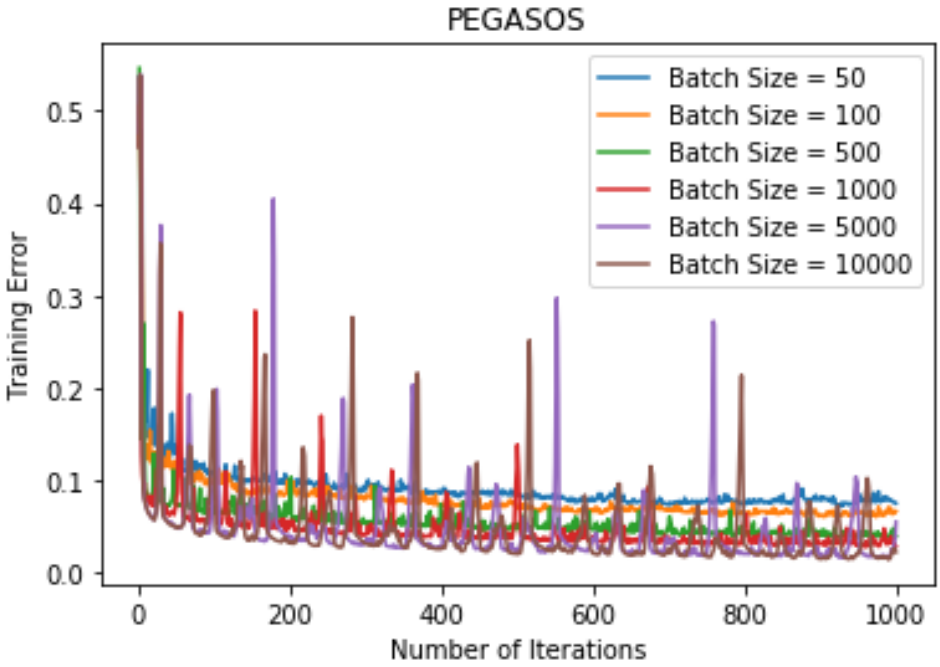
- The number of iterations to perform.
- The batch size that is the number of examples to use to calculate the sub-gradients.

The number of iterations that we have chosen is 0.001. Lambda is inversely proportional to the learning rate.

When lambda is high say in the range of 1, the learning rate is very low, so the update in w is low and the convergence towards minima is very slow and will take large number of iterations to converge. In such a case the update is so less such that there is not much of learning that is happening.

For very low values of lambda, the learning rate is high. Here the w update is high, which means it will converge faster but very high chances of crossing the minima and going on the other side and can keep oscillating. It might also get stuck in local minima. Hence, in this case, the weights fluctuate rapidly leading to overshoot which could prevent us from reaching the global minima.

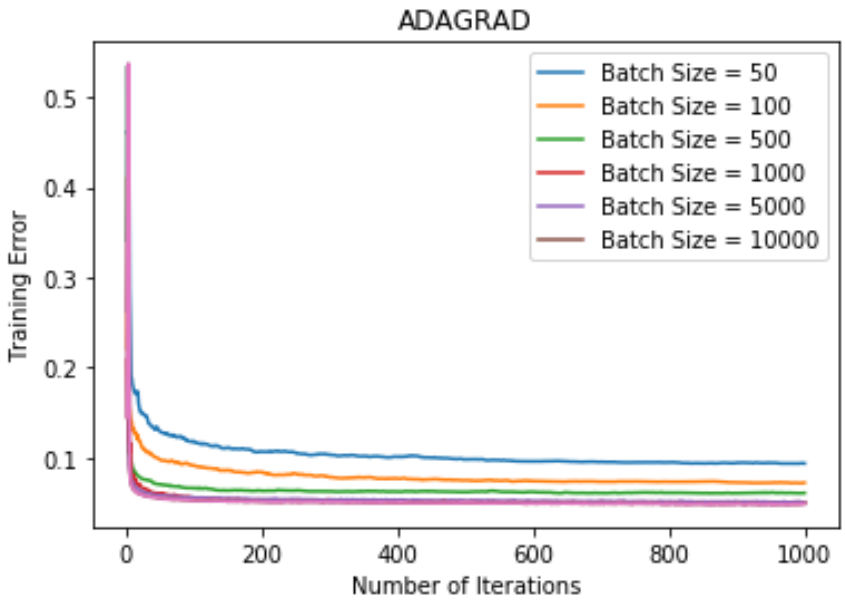
Thereby, we choose lambda which is neither too high or too low for the above situations to occur. We have chosen lambda in between which is 0.001.



As we can see, for the batch size of 5000 and 10000, the training error is the least. We select the batch size of 10000.

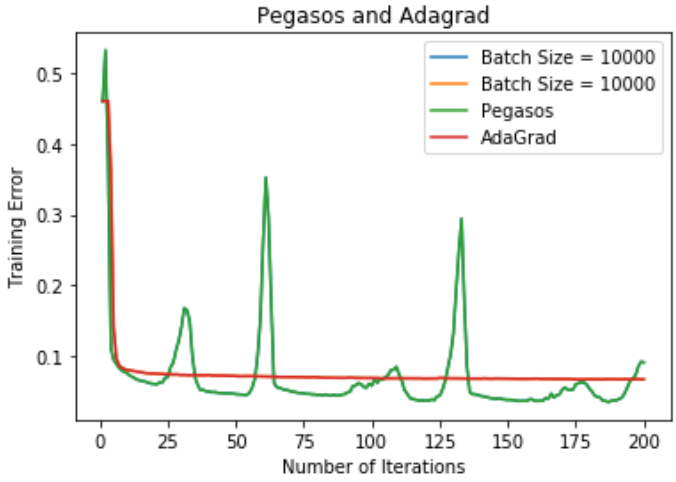
Optimal B = 10000.

Problem 3.



As we can observe from the above graph, if we take the size of the batch to be very small(B = 50,100, 500); the training error is high and it will take large number of iterations to converge. For the values of B above 500, the training error is almost the same and it is converging for almost the same number of iterations.

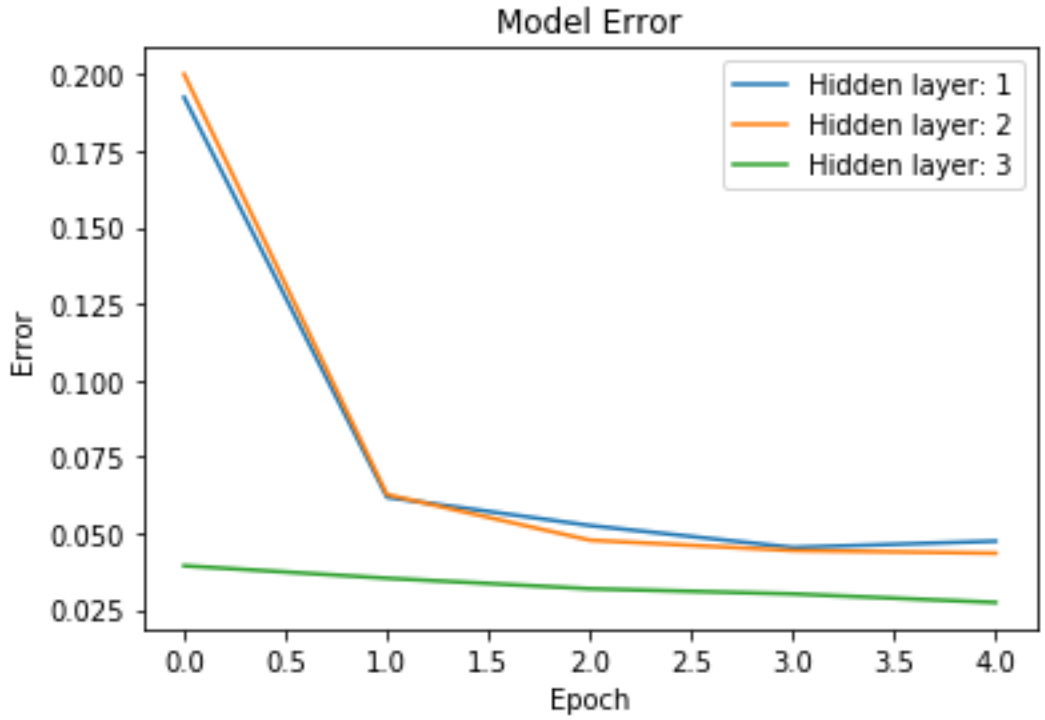
Hence, for Adagrad, we choose the batch size of 10,000 as the training error is the least.



As we can see from the above plot, the training error for Adagrad and Pegasos is similar, but Adagrad converges faster and smoothly than Pegasos as the value of the learning rate is different for every for feature. Adagrad is a boosting algorithm, which gives variable importance to different features. Adagrad boosts the features which are making the function to converge to the global minima. Thereby, Adagrad converges faster.

Problem 4.

(a)



(b)

Training Error for different hidden layers(HL) and hidden units(HU) in each layer.

	HU = 50	HU = 100	HU = 150	HU = 200	HU = 300
HL =1	0.051	0.050	0.052	0.047	0.047
HL = 2	0.048	0.051	0.048	0.047	0.050
HL = 3	0.418	0.048	0.050	0.461	0.461
HL = 4	0.051	0.053	0.056	0.053	0.448

As we can observe, the training error for Hidden Layer 1 for Hidden units 200 and Hidden units 300, and for Hidden Layer 2 for Hidden units 200 is the least which is 0.047. The training error for Hidden Layers 4 for 300 Hidden units is 0.448.

Hidden Layers capture the pattern in the input. At different stages of the hidden layers the level of abstraction of information increases. So increasing the number of hidden layers makes the network learn the intricate patterns in the data. But, increasing the number the number of layers will also increase memory consumption. Since, as we observe in the above table, the training error is least for three combinations. We choose 1 hidden layer as it will save memory out of the three possibilities.

For deciding the number of hidden nodes, we choose 200 from 200 and 300 as for less number of hidden nodes, the training time would be less.

Problem 5:

CLASSIFIER	Test Error	Time taken to run
PEGASOS	0.08	2 minutes 50 seconds
ADAGRAD	0.08	1 minute 39 seconds
Neural Networks	0.09	7 minutes 12 seconds

As we observe, Pegasos and Adagrad perform slightly better (can be said almost equally good) than Neural Networks.

AdaGrad penalizes the learning rate too harshly for parameters which are frequently updated and gives more learning rate to sparse parameters, parameters that are not updated as frequently. In the problem given to us, the data is very sparse. Hence, Adagrad performs slightly or equally good as neural networks.

Since neural network does non-linear mapping by including the ReLU function, in our dataset which is highly sparse, the data might not have much non-linear dependencies and hence the performance for adagrad and pegasos is slightly better than neural networks.