Mini Project 2: Optimization and Text Classification

Anthony Porporino (260863300), Charles Bourbeau (260868653), Sym (260846061)

Abstract - This project explores different gradient descent algorithms and compares their convergence speed on a logistic regression model. The classification task is to determine whether or not a person has diabetes given 8 features of medical information. Mini batch stochastic gradient descent and momentum will be analyzed to see how these variations affect convergence speed and accuracy. The second part of this project analyzes different text pre-processing techniques and how they can improve the accuracy of a logistic regression model classifying sentences into 2 classes; human generated and computer generated.

I - Introduction

The main task of the project was to explore how variations of the gradient descent algorithm affect convergence speed and accuracy of the final model. The variations that will be looked at are mini batch stochastic gradient descent and momentum. We found that using regular gradient descent and a learning rate of 4 produces a model that converges after 400 epochs of training and produces an accuracy of 67% on the validation set and 70% on the test data set.

Mini batch stochastic gradient descent is a variation of regular gradient descent. The idea of this variation is to calculate the gradient based on a subset or 'batch' of the data instead of the full dataset with every iteration. Different batch sizes were experimented with and it was found that a batch size of 28 increased convergence speed from 400 to 37 epochs while keeping accuracy the same.

This project also explored the idea of momentum. Momentum applied to a gradient descent algorithm will update the weights based on a weighted exponential moving average. This means that more importance is given to gradients that are more recent. Three values of momentum were applied to the algorithm with learning rate 4, and it was found that it significantly increased the accuracy of the model by 10% but also decreased the convergence speed.

The second part of this project explored how preprocessing decisions affect the accuracy of a logistic regression model for a text classification task. For this project, bag of words vectorizing technique was used. This technique creates a corpus of words from the training data set, and will vectorize new sentences using this corpus. The context and position of words is lost, but the word choice and word count is encoded in each vector. The decisions explored included; whether to use unigram, bigrams, or both, removing stop words or not, cleaning the data using a regex expression or not, and finally using a variation of bag of word representation called TFIDF (term frequency, inverse document frequency) which gives more weight to rarer words. It was found that using both unigrams and bigrams and TFIDF produced the best results with an accuracy of 77% on the test set. We also observed that removing stop words did not affect the accuracy of the final model.

II - Datasets

The first dataset contains 770 entries of medical information from people as well as an indication of whether or not they have diabetes. There are 8 medical features given: number of pregnancies, glucose level, blood pressure, skin thickness, insulin level, body mass index and diabetes pedigree functions (likelihood of diabetes due to family history).

The second dataset contains 20,000 entries. It contained only one feature which was a paragraph of text that was either human generated or computer generated. A label (0 or 1) indicated which category the text belonged to.

III - Results

Figure 1 shows a graph of the training and validation accuracy vs the number of epochs used to train. The supplied code shows experiments showing why a learning rate of 4 was chosen.

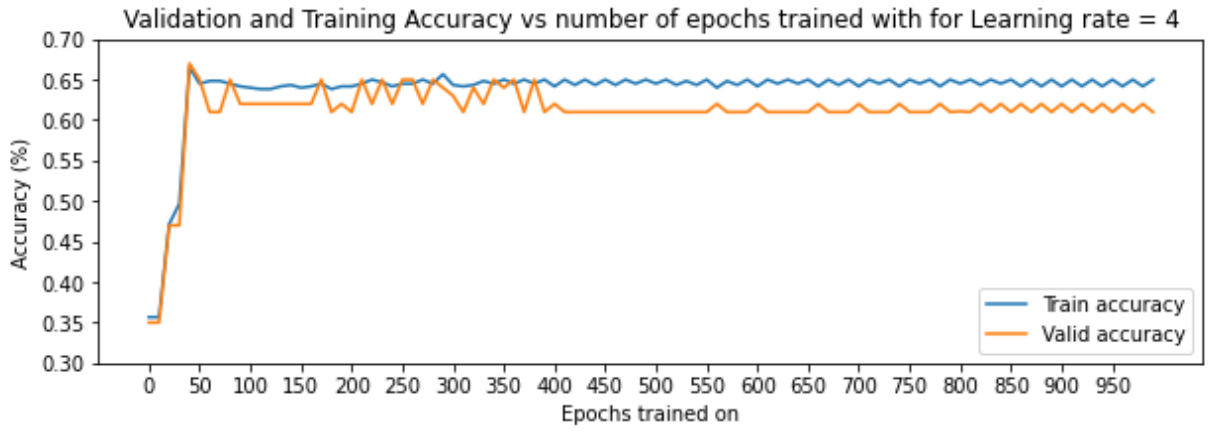

Figure 1: Graph of validation set accuracy of algorithm with learning rate: 4 vs max number epochs allowed

The norm of the gradient does not reach zero, however we can observe that from 400 epochs onwards the validation accuracy never increases. This graph only shows data for up to 1,000 epochs, but it was run for up to 100,000 epochs and the validation accuracy never increased. We can conclude that the model did converge to a solution at 400 epochs with a training accuracy of 67%.

Using mini batch stochastic gradient descent, we can speed up convergence. Table 2 shows the training, validation and test accuracies as well the number of epochs needed to converge for different mini batch sizes. The first row is for regular gradient descent (baseline with full batch).

Batch Size	Grandient Norm	Iters	Epochs	Train acc	Val acc	Test acc
NaN	40.5518	400	400	0.65	0.61	0.676471
5	0	35	0	0.653333	0.62	0.705882
8	0	22	0	0.605	0.51	0.573529
12	0	171	3	0.653333	0.65	0.705882
16	0	1803	47	0.658333	0.61	0.676471
20	0	11832	394	0.636667	0.56	0.617647
25	53.2743	9600	400	0.651667	0.61	0.676471
28	0	814	37	0.643333	0.62	0.691176
32	19.2816	7600	400	0.668333	0.64	0.661765

Figure 2: Table of accuracies and epochs for different batch sizes

For certain mini batch sizes such as 16 and 28, convergence speed drastically increased from 400 epochs to ~40 epochs and was caused by the gradient of the norm reaching zero. We can reason that the cause for the gradient reaching zero is that by reducing the amount of data points that are considered in the gradient computation, we are increasing the probability that all the prediction labels will match the true label, resulting in a gradient of 0. This phenomenon can be observed with a batch size of 5 and 8, where after 35 and 22 iterations, we have already encountered the case where all the predicted labels were correct, making the fit algorithm halt. However, in many cases where the gradient norm reached 0, the test

accuracy for those batch sizes remained similar to what was initially achieved with full batch. One could argue that when the algorithm halts from the gradient norm in gradient descent, the bigger the batch size, the better the solution that results from our fit, because a higher number of correct predictions is required for it to halt. However, this is not what was observed from our experiments.

Lastly, we experimented with momentum for full batch and mini batch gradient descent. Our results show that it takes much longer to converge using any of the 3 different values for momentum tested (.9, .95, .99). We also see greater oscillations between iterations when using momentum. Both of these observations do not match up with what is theoretically supposed to occur. These seemingly incorrect observations can be explained by the fact that our learning rate is too large. This could explain why each step has such large differences in accuracy. Another interesting observation is that adding momentum increased the accuracies for training and validation set to 81%.

Using momentum on small and large mini batch sizes produced different results. On the large sizes such as 32 and 64, the results are similar to regular gradient descent with momentum where convergence speed decreases but accuracy improves significantly. For small mini batch sizes, similar to without momentum, it converges before going through one epoch. The convergence speed is therefore much quicker, but again accuracy is far below standard results.

For part 2, a simple logistic regression classifier taking text as input (in vector format via bag of words) yielded a score of 73%. Introducing TFIDF helped in our final model, however, to a negligible degree. Cleaning the text by performing regex to remove non letter characters from our final test set led to a substantial increase in classification rate as seen from the graph in our notebook. This preprocessing step of standardizing our test data could be included as part of the model.

IV - Discussion and Conclusion

The main conclusion we can make from the experiments conducted on gradient descent variations is that mini batch stochastic gradient descent of an appropriate size can increase convergence speed and cause the gradient norm to reach zero. We should only be satisfied that the gradient norm reaches zero if the mini batch size is reasonably big (3-5% of total data) because this decreases the probability of it being 0 out of pure chance.

Another interesting conclusion is that momentum can decrease convergence speed while also increasing accuracy. One explanation for this is that with a learning rate of size 4 adding momentum causes the weights to change more dramatically with each update. This large learning rate can also explain the oscillations observed. Additionally, we could explain the decrease in convergence speed by noting that the gradients computed early in the computation will still influence later gradients, making the algorithm overshoot the minima it is directed towards. The tradeoff is then that we have to run more iterations to allow the most recent gradients to become influential and correct the course that the current weight vector is taking in the D-dimensional space towards the optimum solution. Hence we see how the convergence speed could slow down. We can observe this phenomenon in our experiments by examining Figure 3 in the appendix, where the accuracy oscillates. Interpreting Figure 3 with the above explanation, these oscillations could represent the model overshooting the optimal solution due to previous gradients. One interesting thing to point out is that when we say that momentum reduces oscillation, we actually mean that it reduces the oscillation of the values of the weight vector, and not the oscillation of accuracy. Although these two are correlated, if a small change in the values of the weight vector highly influences the accuracy score, one could argue that momentum still reduces the oscillation of the weight vector, even if the experiment shows oscillation in the accuracy scores.

It is also possible that the observed convergence in the first experiment did not occur at 400 but took much longer than that. We ran the algorithm for up to 100,000 epochs and did not see any increase in performance, however it is possible that it could start increasing if left for longer. This means that momentum could have actually sped up that convergence speed to a reasonable epoch size and produced the same optimum accuracy of 81%.

For text classification, the conclusions we came to was that the choice of hyperparameters matters to a very large degree. It is important to understand exactly what type of data is to be expected by your classifier as this will help you make educated guesses of what the best collection of hyperparameters to use is. For example, removing Stop Words such as “the”, “a” etc is a common paradigm for certain NLP tasks such as sentiment analysis where these words only add unnecessary data. In our case, however, removing these words has little to no effect on classifying if text is human or computer generated. A possible future experiment is to use lemmatization and stemming. These 2 techniques are similar and reduce words such as cars, car’s, cars’ all down to car.

V - Statement of Contributions

Charles worked on question 1 part 1 and 2. Anthony worked on question 1 part 3 and 4. Sym worked on question 2. All teammates however understood as well as played a role in all parts of the project including the report.

VI - Appendix

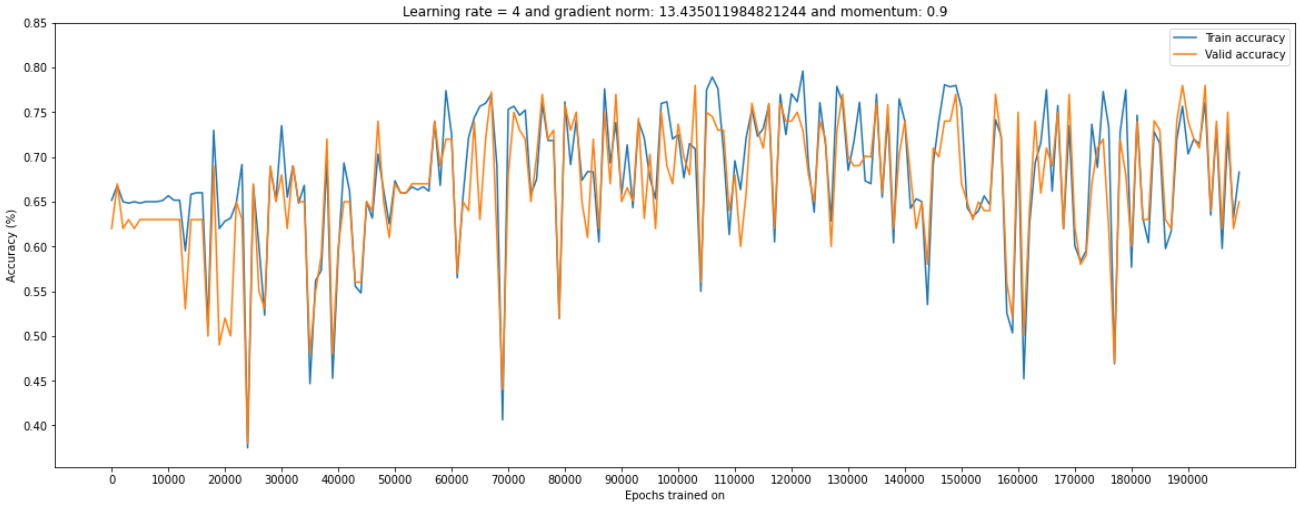

Figure 3: Accuracy vs epochs trained on using full batch and momentum of .9