Miniproject 2: Optimization and Text Classification

Caiwei Xiong, Xinyi Zhu, Zoe Ho Group 26

1 Abstract

In this project, we investigated how different variations of gradient descent can impact the convergence speed, with diabetes dataset. Ultimately, we want to predict if one has diabetes based on eight features, such as pregnancies, Glucose. We found that the quality of the final solution is the best when we use full batch size, and momentum does not have much impact since our learning rate is quite small. We also explored preprocessing techniques such as stop-word removal, lemmatization, and cross-validation with fake news dataset. The task is to predict if the articles are generated by a computer. We applied logistic regression model and achieved an accuracy of 0.682. With Megatron model, the accuracy was significantly increased to 0.9066.

2 Introduction

Our objective is to predict whether a patient has diabetes based on various features. We have implemented the Logistic Regression model [2], and further compared the results of different variations of gradient descent by implementing mini-batch stochastic gradient descent [4] as well as optimizing the algorithm with a momentum coefficient. [1]

News is how we learn about what is happening around the world, such as politics, finance, entertainment etc. Fake news spread the wrong information to the public and can potentially be a dangerous situation. The goal is to use Natural Language Processing techniques to preprocess and clean the data, so we can analyze and detect fake news based on their context. We have used the logistic regression model and the Megatron model as our base model.

3 Optimization (diabetes dataset)

The diabetes dataset consists of 600 training samples, 100 testing samples, and 68 validation samples. There are eight continuous features, such as 'Pregnancies', 'Glucose', and 'BloodPressure'. Two possible outcomes, having diabetes or not, were labeled as 1 or 0, respectively. No data cleaning or preprocessing was needed. We performed binary classification with logistic regression. The goal is to implement different variations of gradient descent and explore whether they have impacts on the convergence speed.

3.1 Convergence

We modified and ran the logistic regression code to serve as a baseline. During the experimental procedures, we started by setting the learning rate to 0.001 and max iterations to 1000. From figure 1, we can see that the training accuracy oscillates a lot as the number of epochs increases. This may result from a high learning rate, so we adjusted the learning rate to 0.0001 and trained the model using the maximum iteration of 1e4. According to figure 2, the training accuracy continuously goes up when the number of iterations is smaller than 4000, and it levels off afterward. The validation accuracy roughly follows the same pattern except that at around 8000th iterations, it goes further up. We wondered if adding more iterations would further improve the model's performance. We then performed another training with maximum iteration up to 1e5 and the learning rate remains unchanged (figure 3). It turns out that the accuracy does not increase much anymore, so we suppose that the model has converged to a solution. Note that the validation accuracy reached a maximum of 0.68 when the number of epochs is 12130.

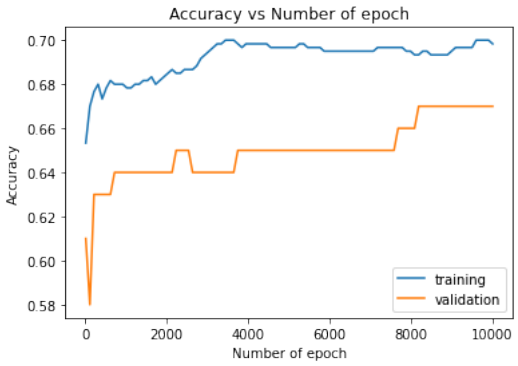

Figure 2. Accuracy vs number of epoch when learning rate is 1e-4, max iteration is 1e4

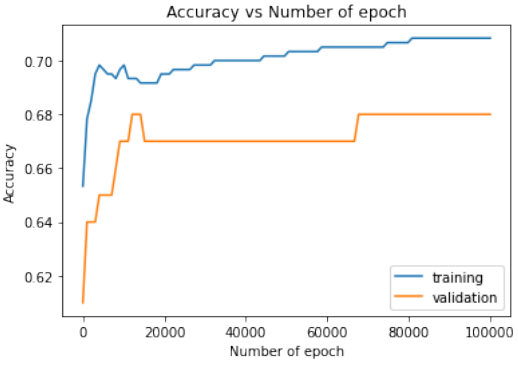

Figure 3. Accuracy vs number of epoch when learning rate is 1e-4, max iteration is 1e5

3.2 Mini-batch stochastic gradient descent

We implemented mini-batch stochastic gradient descent, and applied it to the shuffled training data, with different mini-batch sizes of 8, 16, 32, 40, 100, and 600, in which 600 is the full batch size.

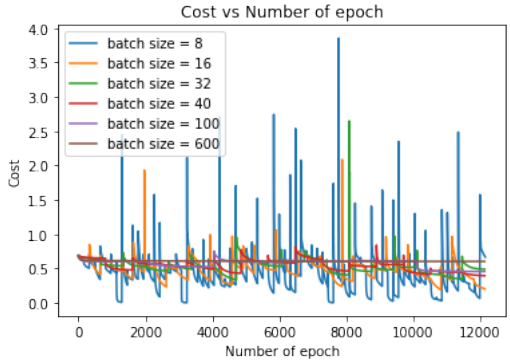

Figure 4. Cost functions for different mini-batch sizes

Under the condition that the model converges to a solution, if we look at the loss over time for the training data in figure 4, there are a lot of fluctuations when the batch size is small. Each step is very noisy since the model hasn't seen the "full picture", but on average, it is in the right direction. As the batch size increases, the graph becomes much smoother. We did not find significant difference on the convergence speed.

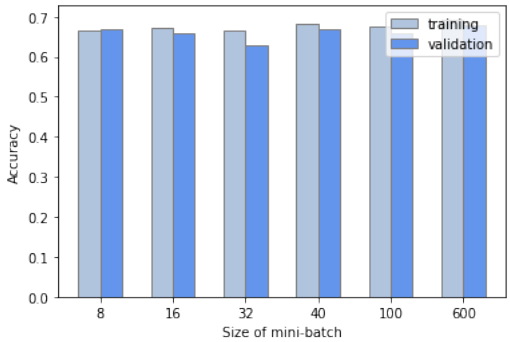

Figure 5. Training and validation accuracy for different mini-batch sizes

In this example, When batch size is equal to 32, the model has the lowest accuracy. This is not the general case, because if we shuffle the training data, the result is different, as each mini-batches would contain different samples. However, the full-batched version always has a slightly better performance than all of the mini-batched versions.

3.3 Momentum

Momentum is an optimizer to help with oscillations of stochastic gradient descent.(Formula will be in Appendix) We tried different values of the momentum coefficient, $\beta = [.2, .4, .6, .8, .9, .95, .99]$ and we found that with $\beta = .9$ and a learning rate larger than 0.0001 we are able to see how momentum reduces the oscillations when calculating the cost. We found that with a low learning rate, in our case where $\alpha = 0.0001$ the impact of applying momentum with a value less than 0.5 on the training time is statistically unnoticeable.

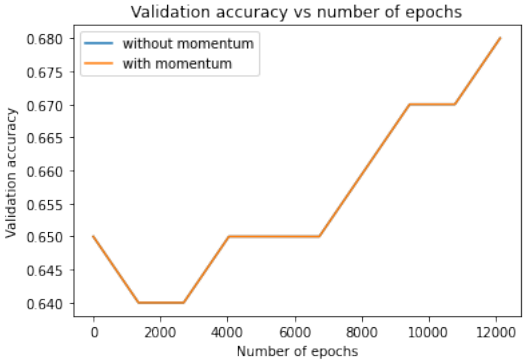

Figure 6. Training accuracy vs number of epochs

3.4 Mini-batch stochastic gradient descent with momentum

We will be using the smallest and largest batch sizes, 8 and 600 respectively. Further optimizing gradient descent with momentum, with a learning rate of 0.001, it is very clear that using momentum the oscillations are way less than only using mini batch gradient descent. As for a smaller $\alpha = 0.0001$, the difference is insignificant, it is hard to notice just by graph representation. Their costs differ after 5 decimal places. Using mini-batch stochastic gradient descent with momentum, we are able to achieve a training accuracy of 0.705 with $\beta = 0.6$ and a validation accuracy of 0.68 with $\beta = 0.9$.

4 Text classification (fake news dataset)

Text classification is the automatic process of predicting one or more categories given a piece of text. We choose to use the NLP (i.e., Natural Language Process) model Megatron to reach the goal of text classification.

4.1 No preprocessing with Logistic Regression

If we did not apply any data preprocessing procedure to the data set, according to the output from the Logistic regression. The test accuracy would be 0.71 and the validation accuracy would be 0.731. This result proofs

that logistic regression has pretty good ability to done the work for text classification, even without any data clean and regularization term.

4.2 Preprocess

The provided dataset is a collection of fake news containing 20000 training samples, 3000 testing samples, and 2000 validation samples, with two labels, 'text' and 'label.'

Remove noises: For the clean data process, we found that there are many useless characters such as punctuation, URLs, HTML's, mentions and numbers. In this case, we decided to remove all those noises to improve the accuracy of classification.

Remove Stopwords: Meanwhile, we remove the English stop words from `nltk` package, the highest frequency words in the text and the useless words such as year, month, th.

Lower case: Since we should expect "Hello" and "hello" as the exact words, the down-cases will be done to each word.

Tokenization: Tokenization is the process of turning a meaningful piece of data. There is no mathematical process, key or algorithm to transform the sensitive information into the token.

Lemmatization: Lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun.

4.3 Logistic Regression

Logistic regression is a classification algorithm used to solve a binary classification problem. The weighted combination of input features will be used to pass through the sigmoid function. The sigmoid function would be able to transform all real value input to the number between 0 and 1. [2]

Tf-Idf: Tf-Ids represents the term frequency-inverse document frequency. It calculates the normalized count where each word count is divided by the number of documents this word appears. (Formula see Appendix) [5]

The test accuracy would be 0.682, and the validation accuracy would be 0.7015. Without any regulation for the model, the dataset does not perform pretty good.

4.4 Megatron-Bert-345m-Uncased Model

Megatron is the model is a robust transformer that NVIDIA develops.[3] And the Megatron-Bert-345m-Uncased Model is pre-trained by 345 million datasets from Wikipedia.

The table output for the best epoch (i.e.,epoch 3):

label	precision	recall	$f1$
label_id:0	86.51	95.88	90.95
label_id:1	95.58	85.64	90.34
micro avg	90.66	90.66	90.66
macro avg	91.05	90.76	90.65
weighted avg	91.14	90.66	90.64

4.5 Future Improvement

There are several packages for the Part of speech tagging (i.e., POS) method to clarify the words. Generally, we only left the adjective and noun words in the text for text classification. In other words, we could try to use the Named Entity Recognition (i.e., NER) model to reach the goal of text classification.

And we could sent the script to the Compute Canada to reach the more times of epoches.

5 Discussion and conclusion

Since the diabetes dataset is small, applying momentum does not have a significant impact. With a higher learning rate like 0.001, it is easier to visualize how momentum optimizes the training time.

Even though the Logistic Regression performs well on the raw data, it did not reach the accuracy as the NLP method.

6 Contributions

Xinyi worked on finding the learning rate and the number of iterations and implemented mini-batch stochastic gradient descent.Zoe worked on optimizing gradient descent with momentum.Caiwei worked on the text classification part. All of us worked on the report together.

Appendix 1

The formula for the Tf-Idf:

$$\text{Tf-Idf}(t, d, D) = \text{Tf}(t, d).\text{Idf}(t, D)$$

where

$$\text{Tf}(t, d) = \log(1 + \text{freq}(t, d))$$

$$\text{Idf}(t, D) = \log\left(\frac{N}{\text{count}(d \in D, t \in d}\right)$$

The formula for Momentum:

$$\Delta w^t \leftarrow \beta \Delta w^{t-1} + (1 - \beta) \nabla J_{\mathbb{B}} \left(w^{t-1}\right) w^t \leftarrow w^{t-1} - \alpha \Delta w^t$$

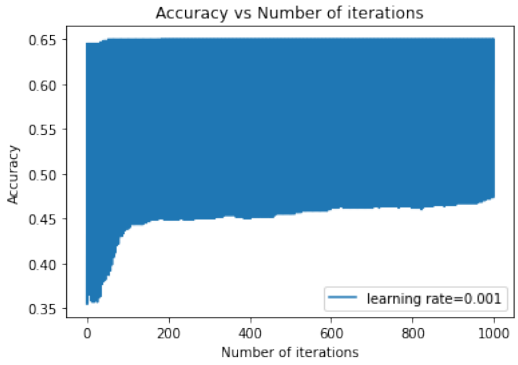

Figure 1. Training accuracy vs number of epochs when learning rate is 0.001

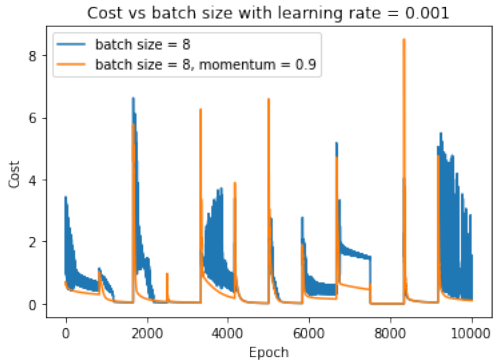

Figure 2. Cost vs batch size when learning rate is 0.001, batch size is 8, and momentum is 0.9

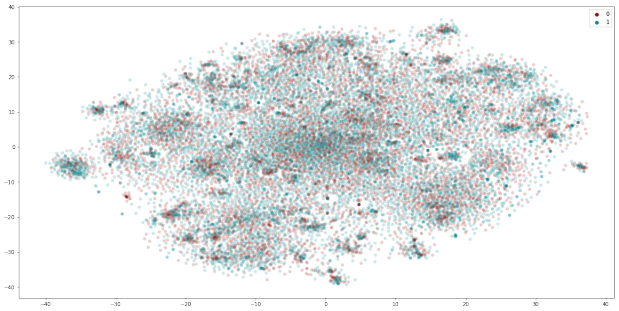

Figure 3. The scatter plot for the whole training data set for the fake_news

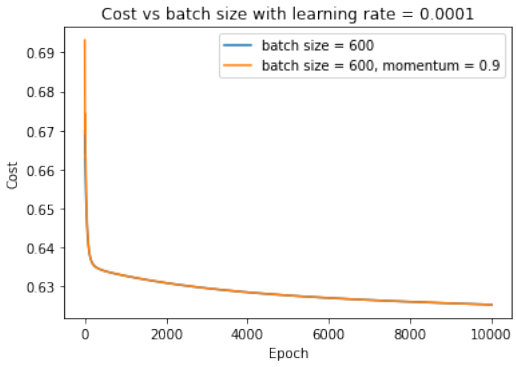

Figure 4. Cost vs number of epoch when learning rate is 1e-4, max iteration is 1e4, momentum (beta) is 0.9.

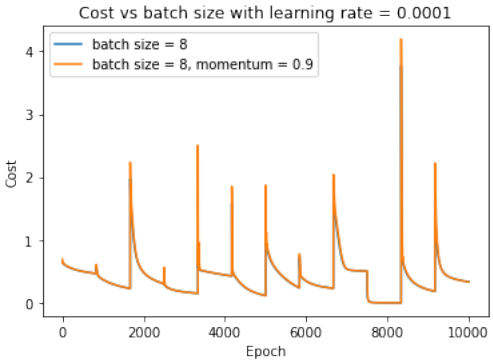

Figure 5. Cost vs number of epoch when learning rate is 1e-4, max iteration is 1e4, momentum (beta) is 0.9.

References

[1] Vitaly Bushaev. *Stochastic Gradient Descent with momentum*. URL: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d. (accessed: 23.10.2021).

[2] Kavita Ganesan. *Build Your First Text Classifier in Python with Logistic Regression*. URL: https://kavita-ganesan.com/news-classifier-with-logistic-regression-in-python/#.YXZLKi-96L0. (accessed: 23.10.2021).

[3] Github. *NVIDIA/Megatron-LM*. URL: https://github.com/NVIDIA/Megatron-LM. (accessed: 23.10.2021).

[4] Aishwarya V Srinivasan. *Stochastic Gradient Descent — Clearly Explained*. URL: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31. (accessed: 23.10.2021).

[5] Bruno Stecanella. *Understanding TF-ID: A Simple Introduction*. URL: https://monkeylearn.com/blog/what-is-tf-idf/. (accessed: 23.10.2021).