UIO

FYS-STK4155
Project 2

November 9th, 2020

Kjell R. Christensen

ABSTRACT

Today's data analysts are faced with a never-ending compromise between gathering data, to the extent that it "clearly" concludes on trends, but on the flip side, what sort of relevance do your conclusion have, if this always drags out in time an becomes yesterday's news.

As a compromise this project will look into the options of processing only part of the dataset by using Stochastic Gradient Descent (SGD).

The project will show that SGD will give a good indicator for a solution, that might be sufficient enough, for the stakeholder.

INTRODUCTION

In a developing society where humans leave an escalating amount of digital trace every single day, data analysts are faced with a huge challenge in the amount of data to process.

For a continuous motivation and search for improvement, large amount of data is analyzed every day, to spot correlations, patterns, but also outliers for “your” every day´s expected behavior.

Today’s data analysts are faced with a never-ending compromise between gathering data, to the extent that it “clearly” concludes on trends. The flip side, what sort of relevance do your conclusion have, if this always drags out in time an becomes yesterday’s news.

Even though computer power has shown to follow More’s law, the data for the last couple of decades has out pace this.

We will use the SGD on the logistic regression and also implement as solution for the FFNN, and process limited amount of a data set.

METHODS

The analyses are done in a msCode environment and is using standard libraries as specified in the *.py heading. All “imports” are based on standard libraries, commonly used for ML, and routines and code generated for this project, is located only in the python file.

We have used two datasets, one test-set, generated by extracting random numbers and a dataset from MNIST. Due to problems with tuning our first NN, we have loaded the breast cancer from sklearn.

The program is modularized, based on separate tasks, and we have modularized in such a way that the program itself, can run one, several or all modules, in one go for analyses. A separate test environment is, hence, not needed.

The reader can at any time reproduce any test-results or plots, by activating the selected part in the MainModule().

All plots are located in the ./Plots folder and they are named, based on function, parameters and methods. The naming convention is also indicating the size of the sample data, to ease the setup, and re-run the tests.

FORMALISM

For the linear regression we know that the Normal Equations might give an analytical solution for the design and coefficient matrix. Tempting as it might be, it should be used as a reference rather than a solution. It will give an “absolute” optimal solution for the equation, hence overfitting our model for the test data. Test point and regression line shown in plot (generated_data_plot)

For batch gradient descent, the routine is quite sensitive to regularization parameter (Lrate * Gradient), and will not always converge towards the optimal. The following plot will show different Lrates and some of them are lacking conversion (gradient_descent_plot).

When we look at the “full” or batch gradient descent, the cost is high due to the size of the design matrix, but det gradient move very direct towards to optimum. Depending on the processing power, the mine-batch sizes can be adjusted, in accordance with the hw-platform.

Looking at the mini-batch and stochastic gradient descent, the routine is computational more cost efficient, but they don’t converge that easily towards the optimum, as shown in the next plot (gradient_descent_path_plot)

Both min-batch and stochastic solutions can be easily tuned to fit your hw-platform for testing, and make the routines suitable for rough predictions.

As can been seen, the stochastic gradient will not reach the optimum, and will continue to “jump” around. But still, it’s a good and rough estimator and is computational sheep.

For the logistic regression, we find that the Sigmoid activation function for the output layer do a reasonable job and clearly converge. An even better solution is using the cross-entropy.

Cross-entropy, as a routine, has a gradient that faster tends towards infinity for yhat, hence penalize the cost function more, for x-values close to the boundaries of [-1,1].
The penalty for wrong predictions is larger and the training converge faster by using the cross-entropy compared to sigmoid.

For FFNN we don’t get convergence for 1-hidden layer. The routine gives prediction between 10-19% for most values, and predictions looks randomized. There is not enough non-linearity, using only one layer.

Adding 2-4 extra hidden layers gives good predictions. Correlating the changes done to decreasing the learning-rate with increased number of epochs moves prediction towards 90%. For the sample set of breast cancer from sklearn, the data set seams limited, to gain better predictions by increasing hidden layers beyond 4-layers. The model overfits, beyond 4-layers and prediction for test data lacks behind more and more (2-3%). See excel sheet in catalog, generated for the testcases for the trends.

RESULTS & DISCUSSIONS

The Stochastic Gradient Descent (SDG), was implemented using standard functionality and we would liked to have momentum as part of our solution, but will be implemented in the next phase of the project. We would also like to see a comparison between using the sigmoid and cross-entropy as classification function in the output layer.

For the FFNN, our first implementation was with just one hidden layer and did not manage to predict, so no conversion. Our test result for MNIST gave an average of 10% for the 10 classes and with the prediction levels set at 0,5 (50%), no predictions were found for the data set. We need more than 2-hidden layers to gain enough non-linearity to get a conversion. To test multilayer networks we implemented the breast cancer. A clear conclusion: We need a minimum of two hidden-layers for a multi classification problem.

The expansion to a flexible multi-layer solution got stuck based on the timeframe and we used a fallback option where we tested the breast cancer data set. The solution was limited in its implementation and did not handle high numbers of epochs, due to variables running towards infinity.

With a 3+ -hidden layer solution we managed a prediction of 86%. Se all test results in FFNNTests
For epochs larger that 300, the network converged quite nicely.

For the breast cancer data our implementation was good, but we will implement SGD for the FFNN, when we approach larger datasets.

CONCLUSIONS & EXPECTATIONS

Based on the algorithm and the theory we can easily see the convenient of using SGD. The methods avoid the high dimensionality and hence gives a good approximation. The solution included is basic and we will pursue adding the momentum to the functionality for the next phase. We would also like to compare the sigmoid and the cross-entropy routines for classification. We expect the cross-entropy to be slightly better, but need to be confirmed to optimize this model.

For the logistic regression classification, we found that the sigmoid activation function was surprisingly good, but also for this implement, we would like to test out the cross-entropy, for further improvement.

The FFNN was hard to implement with multiple hidden layers and we needed to use a fall back solution. We were not able to implement a stable solution for large number of epochs, that most likely would have given us even a better prediction, on the breast cancer data. We run int div/zero problem and we need to pursue this further in the next project. Also, for this implementation we used Sigmoid for the prediction in the out layer and our findings favor the cross-entropy also for the neural network.

However, in between instability we manage to test 5-hidden layers, with 64-neurons in the layers, and pushed the training predictions rate up towards 93%. At that stage we concluded that the network overfitted and test predictions started to lack behind by 2-3%.

REFERENCES

[1] Morten Hjorth-Jensen, Computational Physics, Lecture Notes and sample code, Fall 2020, https://github.com/CompPhysics/MachineLearning

[2] Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning, Second Edition.
DOI https://**doi**.org/**10.1007**/978-0-387-84858-7; Copyright Information Springer-Verlag New York 2009; Publisher Name Springer, New York, NY; eBook ...

[3Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, Aurelien Geron. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow [2nd ed.] 978-1-492-03264-9. 174 47 66MB. English Pages 1150 Year 2019.

[4]Data Science from Scratch-First Principles with Python, Grus.
Afficher les collections Cacher les collections. Identifiant. ISBN : **978-1-491-90142-7**. ISBN : **978-1-491-90142-7**. ISBN : 1-491-90142-X. ISBN : 1-491-90142-X.

[5] Practical Statistics for Data Scientists - Peter Bruce & Andrew Bruce
978-1-491-95296-2 [M] www.allitebooks.com Dedication We would like to dedicate this book to the memories of our parents Victor G. Bruce and Nancy C. Bruce, ...

[6]Python Machine Learning, Sebastian Raschka & Vahid Mirjalili
23. sep. 2017 — Python Machine Learning - von Sebastian Raschka, Vahid Mirjalili (ISBN **978-1-78712-593-3**) bestellen. Schnelle Lieferung, auch auf ...

[7]A Primer on Scientific Programming with Python, Hans Petter Langtangen
ISBN **978-3-662-49886-6**; Free shipping for individuals worldwide. Please be advised Covid-19 shipping restrictions apply. Please review prior to ordering.

[8]A Neural Networks And Deep Larning, Michael Nielsen / Dec 2019
http://neuralnetworksanddeeplearning.com/index.html