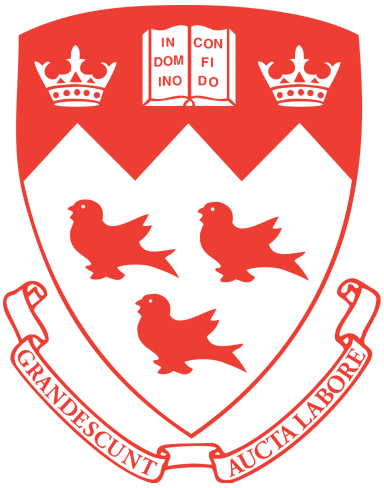
Project 2 Report
COMP 551 - Applied Machine Learning



By
Group 71
Alain Daccache - 260714615
Dylan Havelock - 260721075
Paul Attara - 260807280

November 22, 2020
McGill University

Abstract

In this project, we implement from scratch the Softmax Regression model fitted with a batch gradient descent optimizer. Our gradient descent implementation can accept a learning rate, momentum, and batch size as hyperparameters. We use a simple grid search to find a good combination of these hyperparameters. We then experiment using a termination condition to end the optimization early if the validation error stops decreasing before the gradient descent fully completes. In this case, we return the model with the best validation accuracy. Lastly, we compare the accuracy of our implementation with that of Scikit-Learn and Scikit-Learn's K-Nearest Neighbours using 5-fold cross validation.

Introduction

For this project, we implement a multi class logistic regression model. In our SoftmaxRegression class, the "fit" function takes as input an optimizer which in this case is our implementation of gradient descent. Our GradientDescent class accepts a learning rate, momentum, and batch size as hyper parameters. We use a grid search over these hyperparameters in order to analyze the effect of each hyperparameter on the performance and runtime of our implementation. For each combination of hyperparameters, we use 5-fold cross validation to get a best estimate of performance.

To analyze our implementation, we used the digits dataset provided directly by Scikit-Learn, and the letters dataset provided by OpenML, which we accessed using Scikit-Learn. Each of these datasets uses continuous features with either the digit or letter as the class label, which we one hot encoded. We also used the Iris dataset to run faster tests, given that it has only 4 input features and 150 instances.

Variants of logistic regression, such as Polyak's classical momentum [1] and Nesterov's accelerated gradient [2] attempt to update weights at each step of the gradient descent, concluding that those algorithms perform better (and converge faster) by using L2-regularization. We can therefore extend our implementation similarly. Their observations align with ours, specifically when noticing the 'saddle' effect that the change in the momentum parameter has on the number of iterations.

Datasets

We used datasets where the task is classification; specifically the digits and iris datasets from Scikit-Learn, as well as the letters dataset from OpenML, in order to ensure a variety between continuous and categorical features and targets. The datasets are summarized in Table 1.

	Digits Dataset	Iris Dataset	Letter Dataset
# of Classes	10 (digits from 0 to 9)	3 (Setosa, Versicolour, and Virginica)	26 (capital letters of the English alphabet)
# of Features	64 (each datapoint being an 8 x 8 image of integer pixels between 0 and 16)	4, petal and sepal lengths and widths	17 (i.e. statistical moments and edge counts)
# of Instances	1797 ~ 180 samples per class	150	20000

Preprocessing	Flattened the images from 8 x 8 to an array of 64	N/A	Sampled 2000 instances due to expensive training time

Table 1. Summary of datasets used.

To explore that data, we first visualized it by plotting the distribution of its instances amongst the classes, to prove that the datasets are balanced. For the digits and letters datasets, we also observed the distribution of the average pixel intensities across those classes, to give prior indications of the class characteristics (i.e. 1 should be less intense than an 8). Then, we analyzed the datasets by applying PCA, noticing the classes are separable, especially among the species of Iris, and the digits. This is visualized in Figure 1 in the appendix.

Results

Hyperparameter Optimization

We used a simple grid search to find the optimal set of hyperparameters with 5-fold cross validation for each combination of hyperparameters. We tested batch sizes of 16, 32, 48 and 54, learning rates of 0.010, 0.015, 0.020 and 0.025, and momentum parameters of 0.6, 0.7, 0.8 and 0.9. We can see how the hyperparameters of this grid search vary with respect to one another in Figures 2 and 3 of the appendix. We found the optimal parameters for the digits data set to be a batch size of 48, learning rate of 0.015 and momentum of 0.8. For the letters dataset, the optimal parameters were batch size of 32, learning rate of 0.02 and momentum of 0.9. Since the grid search takes a long time to run, we then used this combination of hyperparameters and varied one of them to observe the effect of each hyper parameter on run time and validation accuracy. The result can be seen in Figures 4, 5 and 6.

In Figure 4 we see the effect of batch size. We observe that the run time increases with the batch size, which makes sense given that it is more computationally intensive to compute the gradient on a larger set of data. We would also expect the validation accuracy to eventually decrease with a large enough batch size, since using larger batch sizes tends to lead to overfitting of the data, whereas using smaller batches can improve generalization. We do observe this as there is a drop off in validation accuracy with batch sizes greater than approximately 80. In Figure 5, we can see that as the learning rate increases, the run time decreases. We also see that the validation accuracy also decreases with learning rates greater than approximately 0.02. This is to be expected when increasing the learning rate, since at some point the learning rate will be too large and will begin to overshoot the target, leading to poorer performance. Interestingly, in Figure 6 we see that momentum has little effect on the validation accuracy. The run time, though, does trend downward as we increase the momentum rate. This is expected given that the purpose of momentum is to keep an exponential moving average of gradients, which helps to decrease oscillations and thus converge faster.

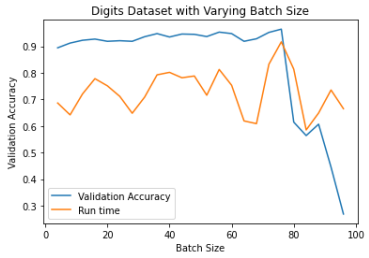


Figure 4. Varying Batch Size

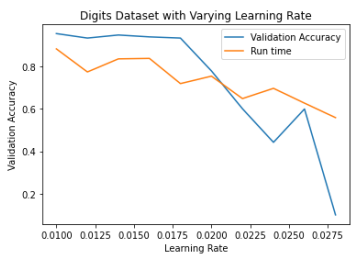


Figure 5. Varying Learning Rate

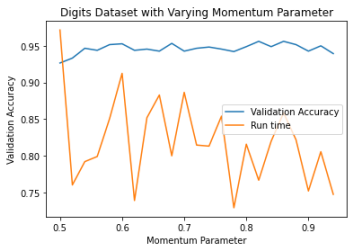


Figure 6. Varying Momentum Rate

Termination Condition

We implemented a termination condition to end training early if the validation error has not decreased for T iterations. We then return the model with the greatest validation accuracy. This both decreases the total number of iterations run thus improving the convergence time of the algorithm, and helps fight overfitting as we return the model with the best validation accuracy rather than the model at the end of the gradient descent. Figure 7 shows the optimization run with T = 10, and Figure 8 with T = 1000. We can see that in both cases they achieve roughly the same validation accuracy in the range of 0.97 to 0.98, but with T = 10 we achieve this with almost twice as few iterations.

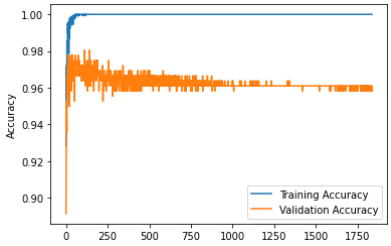


Figure 7. Digits dataset with termination condition T = 10

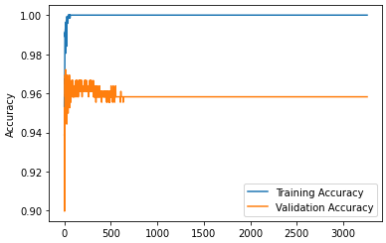


Figure 8. Digits dataset with termination condition T = 1000

Comparison with other classifier

We compared our implementation with the logistic regression and KNN classifiers (with K set to 10) from Scikit-Learn. All classifiers were run with 5-fold cross validation. As expected, the validation accuracy of our implementation and the off-the-shelf logistic regression implementation performed quite similarly with accuracies of 0.9665 and 0.9660 for the digits dataset, respectively. For the letters dataset, we also observed similar accuracies, our implementation achieving 0.7085 and Scikit-Learn achieving 0.7265. The KNN accuracy was also in the same approximate range as the logistic regression classifiers, achieving 0.9811 and 0.6905 for the digits and letters datasets, respectively.

Discussion & Conclusion

For hyper parameter optimization, a key takeaway is that we can use small batch sizes to both improve both run time and generalization. If we use less data for each iteration, as is done with batch gradient descent, our training avoids over fitting our model to the entire dataset. This has the added benefit of also converging more quickly than traditional gradient descent. In using batch gradient descent, it is important to also balance the learning rate and momentum rate used. As we saw, they both play a role in accuracy as well as run time. Additionally, we can be creative with the termination conditions to both improve run time and generalization. Given access to more compute power, we would have liked to experiment more with the letters dataset, but with 20,000 instances our models took extremely long to train, making it difficult to perform a comprehensive grid search.

Statement of Contributions

Paul Attara has worked on the termination condition, as well as on finding an effective formula for the cost during gradient descent. *Alain Daccache* has worked on data exploration, the initial versions of softmax regression and gradient descent, hyperparameters optimization, and part of the report. *Dylan Havelock* has worked on the majority of the report, improving on the softmax regression and gradient descent algorithms, and on comparing against other classifiers.

References

[1] B. Polyak, “Some methods of speeding up the convergence of iteration methods,” USSR Computational Mathematics and Mathematical Physics, vol. 4, no. 5, pp. 1–17, 1964.

[2] Y. Nesterov, “A method of solving a convex programming problem with convergence rate o (1/k2),” in Soviet Mathematics Doklady, vol. 27, no. 2, 1983, pp. 372–376

Appendix

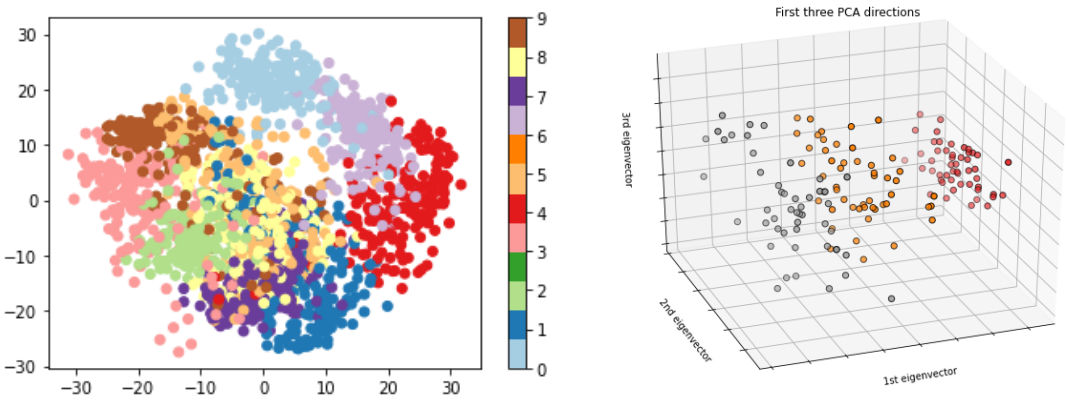


Figure 1. PCA applied to the digit classes (left) and the Iris classes, with only the Sepal width and length features (right)

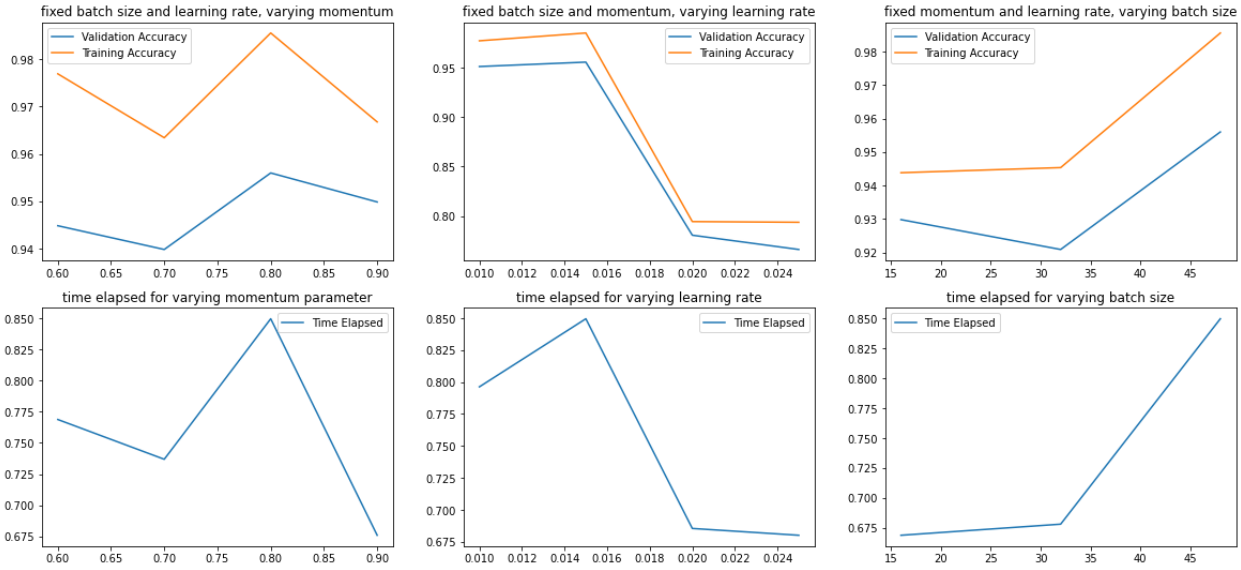


Figure 2. Top row: Digits dataset with varying hyperparameters. Bottom row: Elapsed while varying one hyper parameter

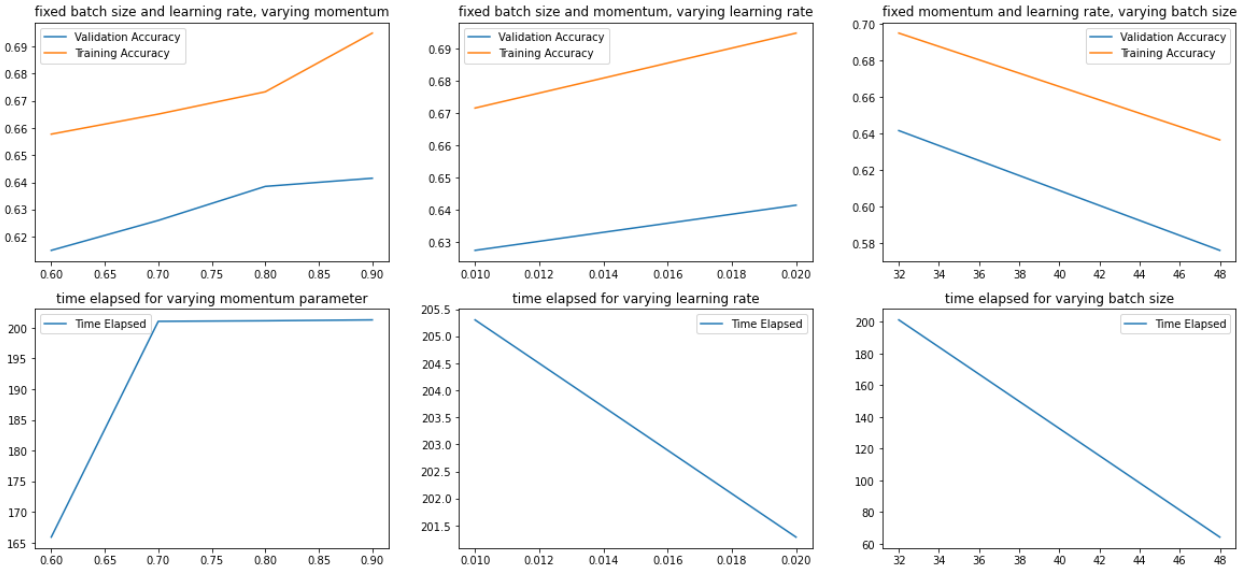


Figure 3. Top row: Letters dataset with varying hyperparameters. Bottom row: Elapsed while varying one hyper parameter