Assignment 3: Gradient Descent Optimization

Zhengmin Yang
University of Toronto

March 26, 2020

1 Introduction

Code Structure The code is divided into two parts in two separate files, `a3_mod.py` and `a3_gradient.py`. The code is heavily modularized, with each calculation step of the respective algorithms performed by helper functions. `a3_gradient.py` deals with the binary classification aspect of the assignment. The public methods `run_GD` and `run_SGD` are designed to be customized based the user's preferences, and allows the user to modify hyperparamters such as the learning rate and the number of iterations.
In `a3_mod.py` the code design strategies are similar. It is also highly modularized, with a separate function for calling the log sigmoid as well as a function `sgd` for the user to call to train the neural network model and observe the results. It also allows the user to customize many options such as setting the learning rate, number of iterations, and whether to save the resulting images to disk for later use.

Objectives In this report we will explore the application of gradient descent to modelling classification problems using machine learning. Both binary classification and multiclass classification will be considered. For binary classification, the prediction function is a sigmoid function (1) and the negative log likelihood, which is also the loss function, is given (2).

$$sigmoid(x) = \frac{1}{1+e^x} \tag{1}$$

$$\log Pr(y|w, X) = \sum_{i=1}^{N} y^{(i)} \log(\hat{f}(x^{(i)}; w)) + (1 - y^{(i)}) \log(1 - \hat{f}(x^{(i)}; w)) \tag{2}$$

Note that the log likelihood will equal $-\infty$ if the predicted class probability is $\hat{f}(x^{(i)}; w) = 1$ but the actual label is 0. This means that the model is telling us that the class is not 0 with 100% confidence although it is 0. This makes sense from a mathematical perspective. We are trying to maximize the likelihood of making a correct prediction, i.e. the log likelihood. Since this prediction is completely incorrect, it is in our best interest not to choose it. With a value of $-\infty$ this ensures that when we apply the argmax function we will never choose the value of w that gave us this completely incorrect prediction.

For multiclass classification, the prediction function is the softmax function (3) and the negative log likelihood (loss function) is given by (4).

$$softmax(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}} \tag{3}$$

$$\log Pr(Y|X, w) = \sum_{i=1}^{N} \sum_{j=1}^{K} y_j^{(i)} \log(\hat{f}_j(x^{(i)}, w)) \tag{4}$$

Both full batch gradient descent and SGD utilize the fact that we are able to minimize the loss with respect to the weights w by moving the weights in a direction opposite the gradient of the loss (with respect to the weights). The amount that we move by can be calculated by using a line search algorithm (included in the file submission but not used) or set as a parameter η, also known as learning rate (used for this assignment). The weight update at the (k+1)th iteration is given by the following:

$$w^{(k+1)} = w^{(k)} - \eta \nabla \log Pr(y|w^{(k)}, X) \tag{5}$$

2 Binary Classification

2.1 Full Batch Gradient Descent for Binary Classification

Full batch gradient descent uses the entire set of training data as inputs to the loss function to calculate the direction to move the weight vector. The following figure shows losses with three different learning rates.

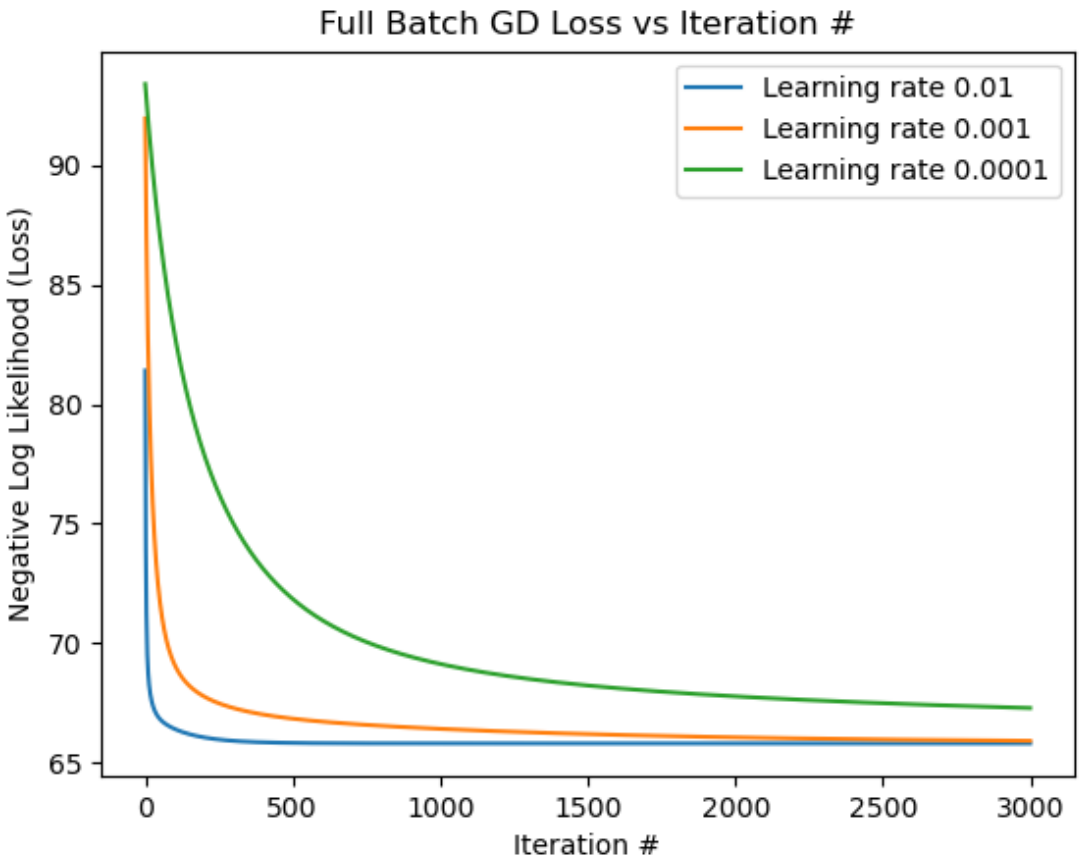

Figure 1: Loss for varying values of η.

As we can see, the learning rate of 0.01 corresponds to the smallest loss. The largest testing accuracies correspond to learning rates of 0.01 and 0.0001, and the smallest testing loss corresponds to the largest learning rate, 0.01. This correlates well with the trends set by the training loss, which indicates that a slower learning rate corresponds to a larger loss is therefore more inaccurate. This is because a smaller learning rate corresponds to a slower convergence, as seen in the graph. For example, a learning rate of 0.01 allows the loss to converge very quickly at roughly the 200th iteration. However the loss converges slower for a learning rate of 0.001, at around the 2500th iteration, while the loss has yet to converge at the 500th iteration for a learning rate of 0.0001. This makes sense because the learning rate corresponds to the step size that we take every time we update the weight vector; a larger step size results in a faster descent towards a minimum and hence a faster convergence rate.

Learning Rate	Testing Accuracy	Testing Loss	Training Loss
0.01	0.733	6.91	65.8
0.001	0.667	7.00	66.8
0.0001	0.733	7.06	77.9

Table 1: Testing Accuracy, Testing Loss, and Training Loss for Different Learning Rates

2.2 Stochastic Gradient Descent for Binary Classification

Stochastic Gradient Descent differs from Full Batch Gradient Descent in that it randomly chooses a training point as an input to the loss function in order to find the gradient. Because of this random process, the loss curves that it generates are jagged. It can be shown that SGD is an unbiased estimator of the full batch gradient descent despite being less expensive computationally. The results below agree heuristically with the results previously obtained by using full batch gradient descent.

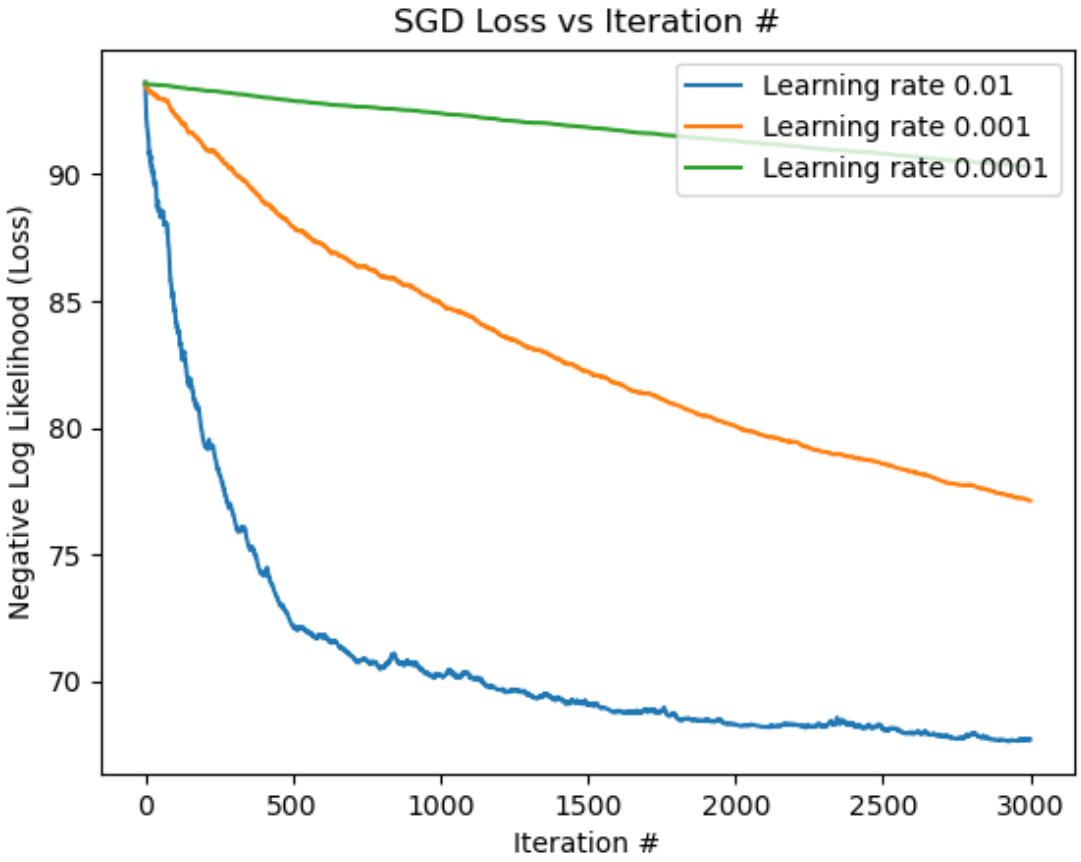

Figure 2: Loss for varying values of η.

Again, this confirms that the learning rate of 0.01 corresponds to the smallest training loss. The highest testing accuracies correspond to learning rates of 0.01 and 0.001 while the lowest testing loss corresponds to the highest learning rate like in full batch gradient descent. This also correlates directly with the convergence rates of the loss function. For example, we can see that the loss

corresponding to a learning rate of 0.01 shows signs of convergence after about the 2000th iteration, while the losses for the other two learning rates are still converging and do not show any sign of convergence even after the 300th iteration. Judging by the shapes of the curves it is apparent that the curve corresponding to a learning rate of 0.001 is moving towards convergence faster than the curve corresponding to a learning rate of 0.01.

Learning Rate	Testing Accuracy	Testing Loss	Training Loss
0.01	0.733	6.79	67.9
0.001	0.733	8.33	76.5
0.0001	0.667	10.0	90.4

Table 2: Testing Accuracy, Testing Loss, and Training Loss for Different Learning Rates

2.3 Comparison Between Full Batch GD and SGD

2.3.1 Convergence Rates

Full batch gradient descent (FBGD) converges a lot faster than SGD in terms of the number of iterations performed. However due to the fact that we use all of the training data to compute the gradient it is more computationally expensive than SGD. This tradeoff is not apparent in small datasets such as this, where the time to run the models is roughly the same (around 0.4 seconds per model) but will become apparent in larger datasets where FBGD's computation time will slow down dramatically due to the dependence of its computation on the size of the dataset. SGD on the other hand, while converging in a higher number of iterations will run faster due to its computational independence on the size of the dataset.

2.3.2 Losses

FGBD and SGD present similar losses for learning rates of 0.01 and 0.001, suggesting that SGD is a good estimator of FBGD while being less computationally expensive. In addition, as sjhown in class, the expected value of an SGD update is equivalent to an FBGD update, again confirming their similarities. They show different results for learning rate = 0.0001 because the SGD loss has not yet converged and did not show any sign of convergence. Thus SGD's training loss is much higher than FBGD's.

2.4 Test Log Likelihood as a Metric

The testing log likelihood could be a more suitable metric than test accuracy for measuring performance. This is because the signmoid gives a range of probabilities, which is then mapped to one class or another. The probabilities could be thought of as confidences in whether an input belongs to a certain class. However the testing accuracy loses this confidence information when it maps the outputs of the sigmoid onto a class. By using the log likelihood we are also able to determine whether the model its confidence in giving the correct output in addition to whether is really giving the correct output.

3 Stochastic Gradient Descent for Multiclass Classification

SGD can also be applied to train neural networks used in multiclass classification. Multiclass classification uses the softmax activation function (3) to predict the class of a input. The negative log likelihood in (4) is used to as a measure of loss of the model. In this case we are predicting the

actual value of handwritten single digit numbers which are organized into 10 classes, one class for each digit. We are using the MINST dataset and splitting it into training, validation, and testing sets.
During each training iteration we are using a batch of 250 randomly chosen sets of inputs and outputs to determine the weight update. We then calculate the subsequent loss on the validation dataset to give us a more accurate picture of the model's loss given unseen real-world data. The model is validated over the entire validation set, and both the average training and validation losses are plotted for two separate learning rates, as shown below in Figures 3 and 4.
Note that we choose the average loss as it is more indicative of model performance rather than the total loss, which is dependent on the size of the dataset. The total loss adds up the losses of each point, meaning that it monotonically increases as the number of points in a batch increases.
The weights of the inputs to the hidden layers, which use the ReLU activation, were initialized according to guidelines set by He et al., and the weights of the inputs to the final softmax activation layer uses Xavier Activation. All biases were initialized to 0.

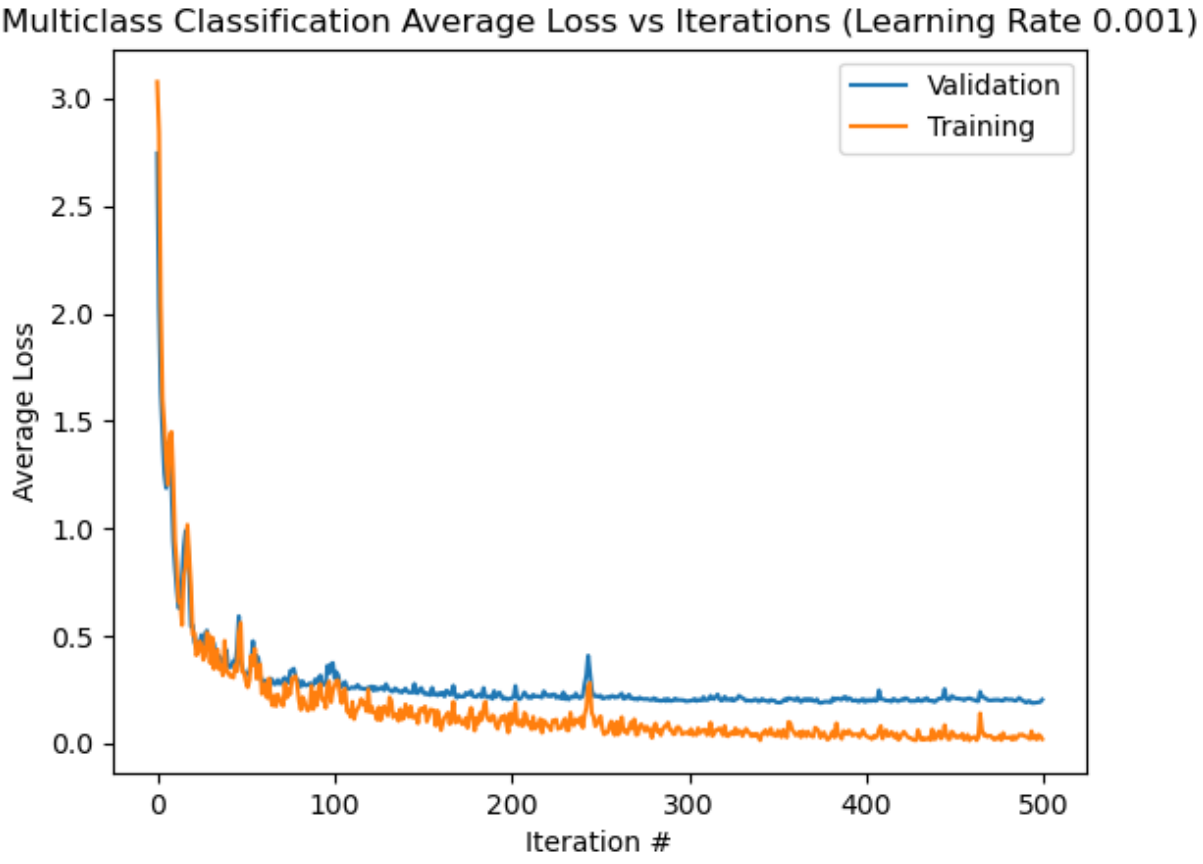

Figure 3: Loss for Multiclass SGD Learning Rate 0.001 of η.

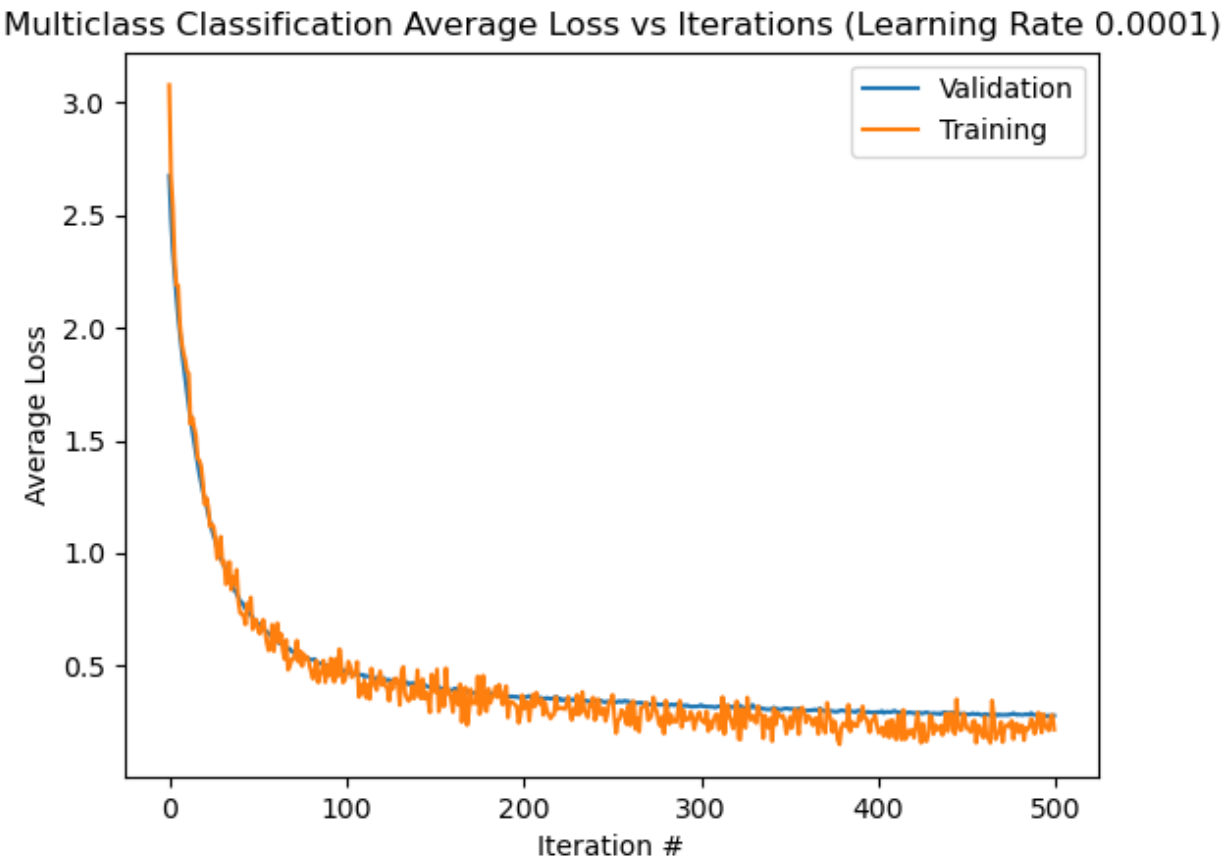

Figure 4: Loss for Multiclass SGD Learning Rate 0.0001 of η.

Quantitative end results are listed below, and also includes the average loss on the test dataset. The average validation loss and testing loss values were taken at the last iteration.

Learning Rate	Testing Loss	Validation Loss	Training Loss
0.001	0.176	0.203	0.020
0.0001	0.327	0.362	0.369

Table 3: Average Testing Loss, Validation Loss, and Training Loss for Different Learning Rates

As we can see, the model is not overfitting. This is observed by several factors. Firstly, the testing loss matches closely with the validation loss. In addition, the validation loss is converging at a minimal value and is not going up. This means that the model is able to generalize well to foreign datasets that it has not encountered before during training.

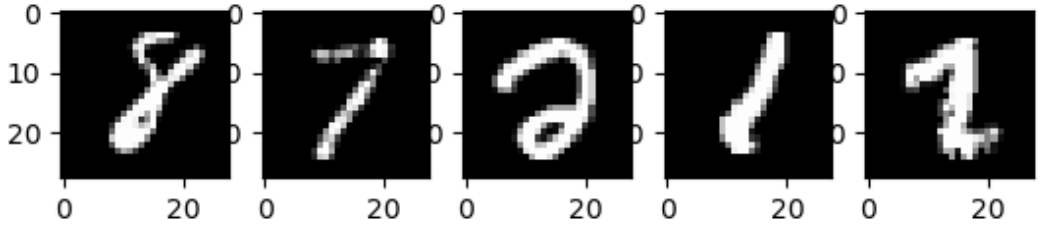

Figure 5: Images with low model class prediction confidence

3.1 Numerical Stability of Log Softmax

The softmax function is composed of various exponential functions, which can grow very large. While modern computers have the ability to represent very small or very large numbers, there is the off chance that the number is so large it can overflow, causing an incorrect arithmetic result. Therefore, in order to prevent such cases of overflow, we use the log softmax in our implementation where we subtract each exponent term of each exponential by the largest value in the input vector, as shown in the following equation (6).

$$\log softmax(x_j) = x_j - (a + \log \sum_{i=1}^{K} e^{x_i - a}) \tag{6}$$

Where a is the largest value in the input vector $\mathbf{x} \in \mathbb{R}^K$.
This ensures that our result will not overflow while also aiding us in our ultimate goal of finding the log likelihood (4), where we would need to take the log of the softmax anyway.

3.2 Effect of Inputs on SGD Prediction Confidence

The confidence of a class prediction of an input is also heavily affected by the types of inputs that it receives. Those that are more irregular than most other inputs are more difficult for the model

to classify. As an example, the following five images all resulted in a model confidence of less than 50 percent.
As we can see, the numbers are written irregularly and have more deviation from the "ideal" shape of the number that we would see on a digital screen, for example. Some are irregularly shaped while others are blurry and lack the distinguishing shape of a regular number. Without more complex features of a convolutional neural network that allow for location based feature extraction, it would be difficult for the regular neural network to classify these input images as strictly belonging to one class. The model is unsure of which class it belongs to which is why its prediction confidence is very low.