Saaif Ahmed

661925946

ahmeds7

2/17/2022

Introduction to Deep Learning: Programming Assignment #1 Report

1: Multi-Class Logistic Regression Learning

For a scenario where $y \in \{1, \dots, K\}$ we can use a multi-classifier logistic regression learning model to build a reasonably accurate system that can predict y by using a series of weights $\vec{w}_1, \dots, \vec{w}_k$ in a matrix Θ such that if the data set is $N \times M$ then $W^{M \times K}$ with each $\overrightarrow{w_i}^{M \times 1}$ and passing them into a soft max function for logistic regression.

In order to build towards an accurate model we must minimize the negative logarithmic likelihood of these weights against the training sample to make a prediction. In this case we have our data set $D^{M \times N}$ that has outputs $y \in \{1,2,3,4,5\}$ and input parameters $\vec{x}[m]$. The output y follows the 1 of K encoding and as such is an output vector $\vec{y}[m]$. In other words $D = \{\vec{x}[m], \vec{y}[m]\}, m = 1,2,\dots,M$. We have a matrix $\Theta = (\vec{\mathrm{w}}_{\mathrm{k}}, \mathrm{w}_{\mathrm{k}_0})$, where k=1,2,3, 4, 5, $\mathbf{W_k}$ is a vector for kth discriminant function, and $\mathrm{W}_{\mathrm{k},0}$ is its bias. We define the loss function as:

$$L(D:\Theta) = -\sum_{m=1}^{M}\sum_{(k=1)}^{K} \vec{y}[m][k] \log \sigma_M(\vec{x}^T[m]\Theta_k)$$

Such that $\sigma_M()$ is the soft-max function.

$$\sigma_M(\vec{x}^T[m]\Theta_k) = \frac{e^{\vec{x}^T[m]\Theta_k}}{\sum_{k=1}^{K} e^{\vec{x}^T[m]\Theta_k}}$$

To build the model over many iterations we build the gradient descent method for the logistic regression. We define the gradient of the loss function as:

$$\nabla_\theta L(D:\Theta) = [\nabla_{\Theta_1} L(D:\Theta_1), \dots, \nabla_{\Theta_k} L(D:\Theta_k)]$$

With the gradient of each Θ_k computed as:

$$\frac{\partial L(D:\Theta)}{\partial \Theta_k} = -\sum_{m=1}^{M} \{\vec{y}[m][k] - \sigma_M(\vec{x}^T[m]\Theta_k)\}\vec{x}[m]$$

With this gradient we can update the Θ matrix at iteration t as

$$\Theta^t = \Theta^{t-1} - \eta \nabla_\theta L(D:\Theta)$$

With η as a hyper parameter of the model known as the “learning rate”.

In the report below we analyze the performance of an implemented multi-classifier logistic regression model on the MNIST dataset. The objective is to build a model that accurately predicts the numbers on an image that is 28 by 28 pixels. The image is grayscale and is unrolled into a 784x1 vector. For the training set there are 25112 images. Thus our input data $X^{25112\times784}$ with output labels $\vec{y}^{5\times1}$. As such $\Theta^{785\times5}$ with each $\vec{w}_k^{785\times1}$ since each $\overrightarrow{w_k}$ has an additional w_0 bias.

2: Experiment and Build of the Model

Hyper Parameters: For our experiment we have a few hyper parameters that directly contribute to the learning of the parameters $\overrightarrow{w_k}$.

Learning rate η is a hyper parameter that multiplies the gradient for the theta updates. This model chose $\eta = 0.0001$. With too large of a learning rate the loss function will not converge. Therefore we choose small enough η such that the function begins to converge.

Iterations: another hyper parameter that contributes to how many times we update Θ. To generate the date, 100 iterations was chosen to be a suitable amount to achieve an accurate Θ.

Dimensions of dataset D: a hyper parameter that determined our gradient descent method. This model used full batch gradient descent, using all of the data in D. As $D = \{\vec{x}[m], \vec{y}[m]\}, m = 1,2,..,M$ and input data $X^{25112\times784}$ with output labels $\vec{y}^{5\times1}$ theta was $\Theta^{785\times5}$ with each $\vec{w}_k^{785\times1}$. This led to computationally expensive gradients, but also allowed conversion to happen earlier.

Other Model Parameters and Information: Θ was initialized to all zero values in the requisite dimension. The learning rate and number of iterations was determined via trial and error. Furthermore while the Θ matrix was updated at all iterations, the errors, accuracy, and overall loss was only taken at every 10^{th} interval to improve runtime execution.

3: Results and Performance

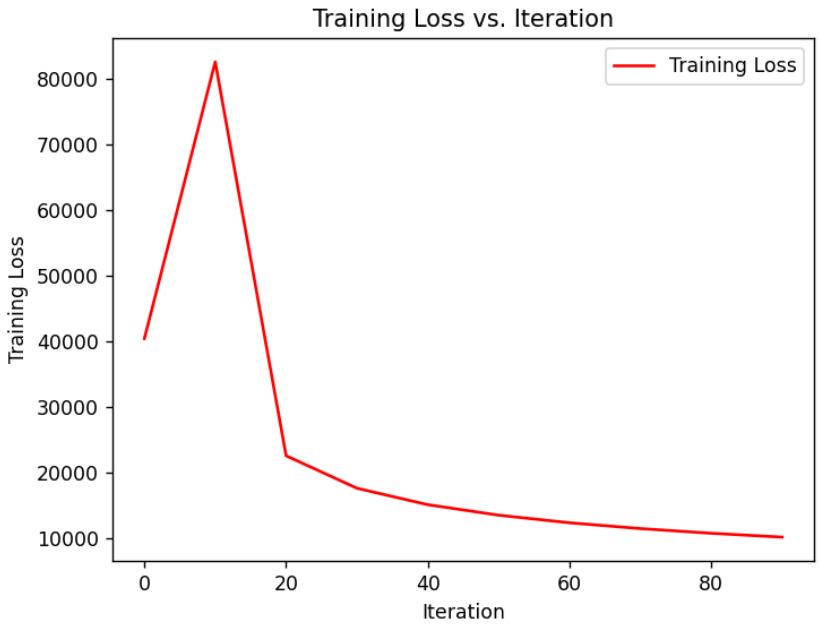

Figure 1: Training Loss vs. Iteration

Above we have the models training loss per iteration of the weight matrix update. While the trend is correct for batch gradient descent, the numbers are very large. This is due to a lack of normalization in the loss function. Even if the individual cost of each row of the dataset was small, due to the size of the dataset (>25000) the cost did add up over time. What could have been done to avoid is this is to add a $\frac{1}{M}$ term in front of the $L(D:\Theta)$. This option was not taken in order to preserve the integrity of the multi-class logistic regression theory in the code of the model since it would have to be added to the gradient function $\nabla_{\Theta} L(D:\Theta)$ as well.

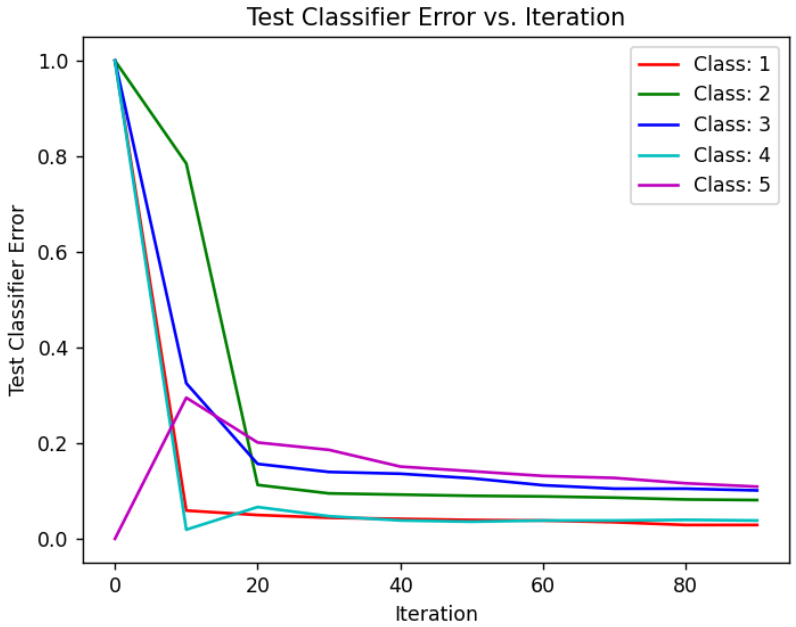

Figure 2: Test Classifier Error vs. Iteration

Plotted above is the error rate of the model against the testing set. The model used the $\arg\max(\sigma_m())$ in order to make a prediction of the classifier label. Initially the learned weight parameters were not able to accurately predict but over the course of the iterations the model was able to predict the input classes very well. Error rates were no higher than about 10% and were as low as 1%. The accuracy could be given by computing $1 - \overrightarrow{w_k}_{error\_rate}$.

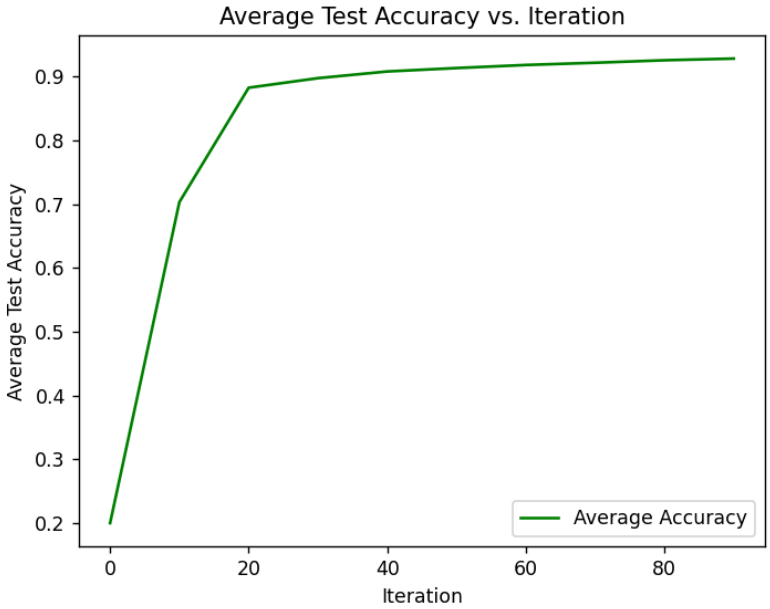

Figure 3: Average Test Accuracy vs. Iteration.

Plotted above is the average testing accuracy of the whole Θ matrix against the testing set. It took many iterations for the model to learn accurate enough parameters to achieve the desired accuracy. However by 60 iterations the model had achieve >90% accuracy overall and by the 100^{th} iteration the model had achieved >93% accuracy.

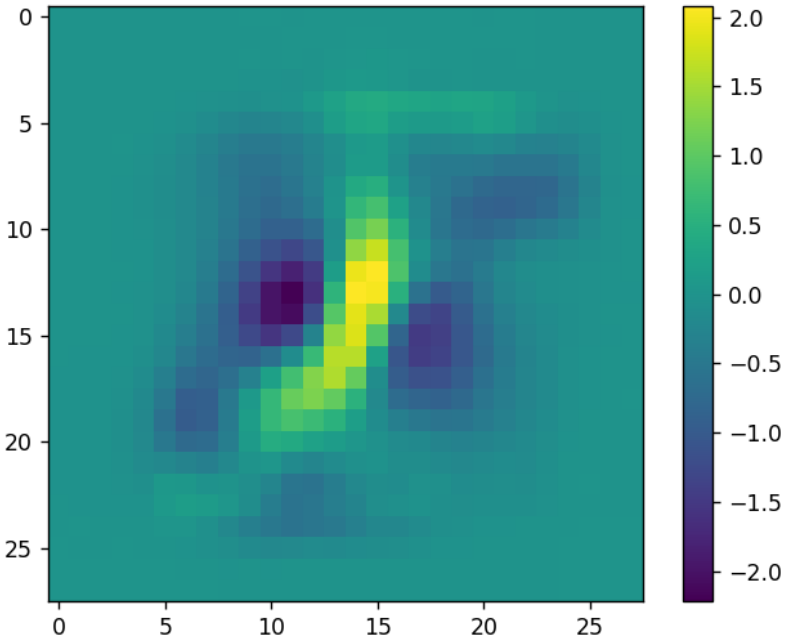

Figure 4: Parameter 1 Image

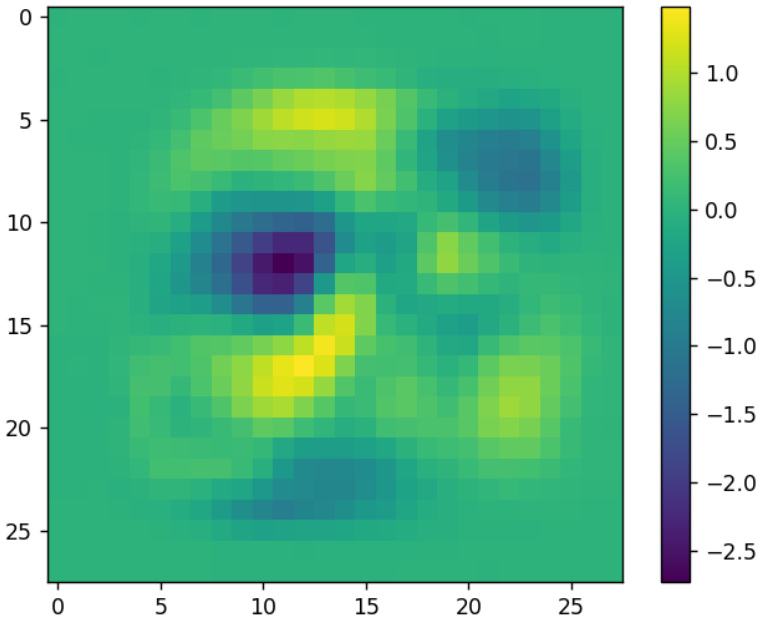

Figure 5: Parameter 2 Image

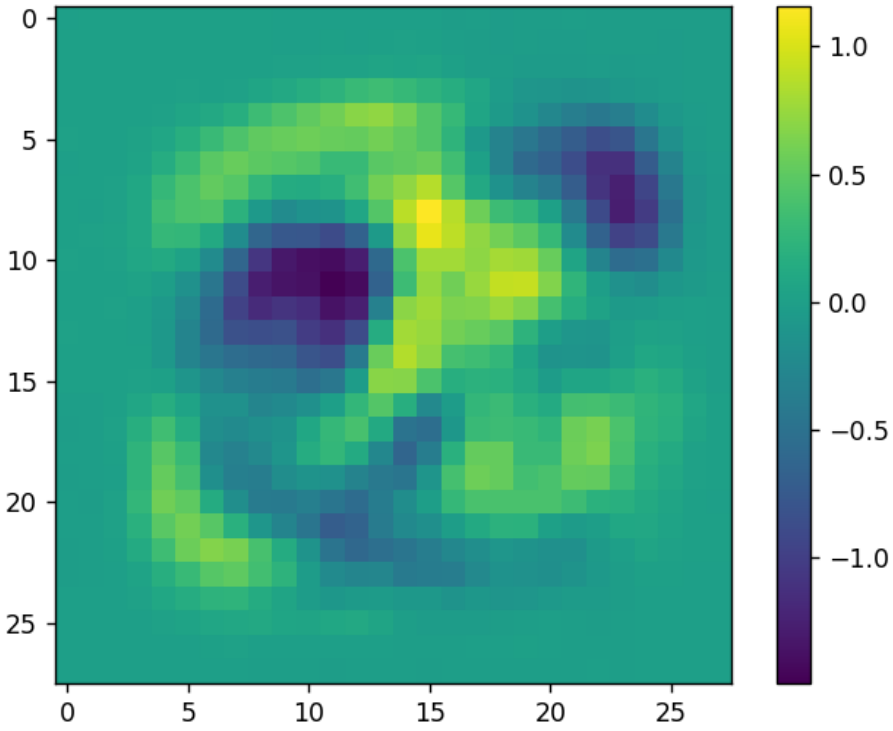

Figure 6: Parameter 3 Image

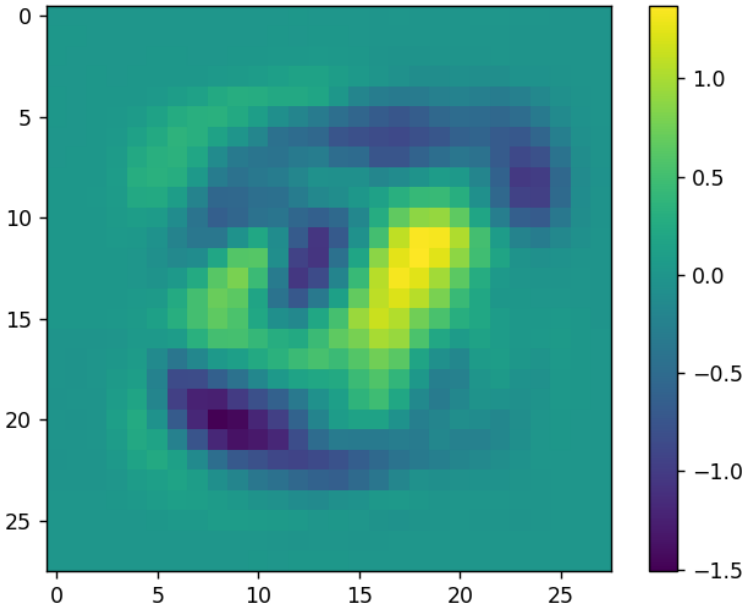

Figure 7: Parameter 4 Image

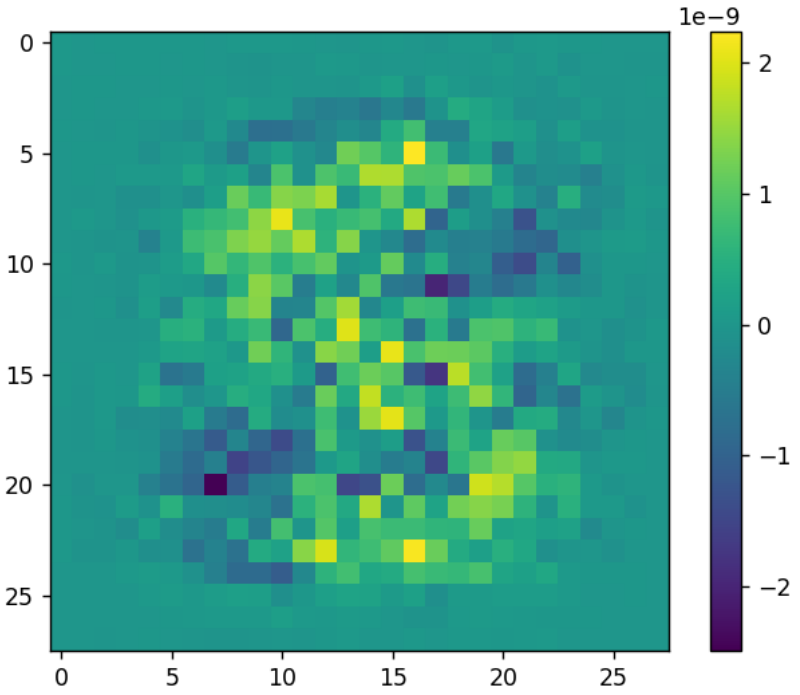

Figure 8: Parameter 5 Image

The figures 4-8 above depict the learned parameters $\Theta = [\Theta_1, \Theta_2, \Theta_3, \Theta_4, \Theta_5]$: $\Theta^{784\times5}$ and $\Theta_k^{784\times1}$. There are 784 elements because the images that the model learned from were 28x28 images. The color bar indicates the value of the element in each Θ_k. Yellow being a more positive value. What can be noted is that by observing the positive weights, or tracing the more yellow areas of the images, the number each parameter is meant to predict the output of can be drawn by that trace. This can indicate that the model has learned well that those pixels on an image are meant to line up with the number Θ_k is meant to predict.

Section 4: Conclusion and Model Usage Information

The model seems to be working well, and more iterations should allow it converge better and become more accurate. With a high prediction accuracy rate the multi-class logistic regression is proven to be a reasonable machine learning model. And with the gradient descending, minimizing the negative log likelihood is a good way of learning the parameter matrix Θ for each class. However, it is undetermined whether the model is over fit or not. Testing was conducted on a set of data not used for training, however the images used may not be different enough to prevent overfitting. Some points of improvement to the model and dataset could be: feeding in images of a different rotation, feeding in more images outside of grayscale, distinguishing between letters and numbers.

Notes for Running the Code:

The code is commented and blocked out with descriptions on what each block does. The imports listed at the top of the python file describe what packages must be installed to the environment for it to run. The code makes use of using python's pickle feature to save data in steps such that re-computation is not needed. Blocks are clearly labeled on what to comment and uncomment in order to run certain features of the code. There is a potential that lines 10 and 11 may need to be commented out if the model is run on a system other than Windows. It should not interfere with the computation of the model, or the generation of loss/accuracy graphs but it may prevent the Θ_k images from being created. Other assumptions include that files are accessible in the same named folders as listen in the code or in the same directory as the main.py file.