Homework 2
600.482/682 Deep Learning
Fall 2019

Jason Zhang

September 22, 2019

Due Fri 09/20 11:59pm.
Please submit a zip containing a report (LaTeX generated PDF) and
4 python Jupyter Notebooks (one for each problem but not subproblem)
to Gradescope with entry code MKDPGK

1. The goal of this problem is to minimize a function given a certain input using gradient descent by breaking down the overall function into smaller components via a computation graph. The function is defined as:

$$f(x_1, x_2, w_1, w_2) = \frac{1}{1 + e^{-(w_1x_1+w_2x_2)}} + 0.5(w_1^2 + w_2^2)\,.$$

   (a) Please calculate $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}$.
   Answer:

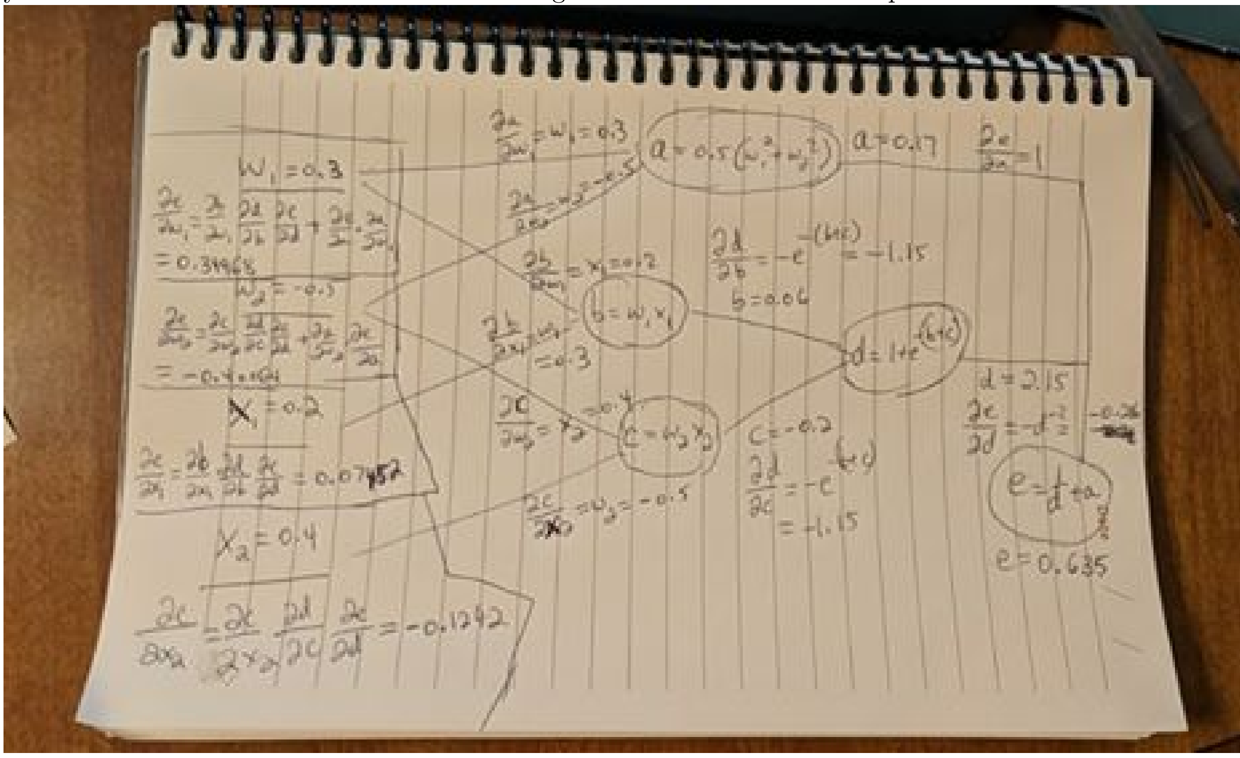
$$\begin{aligned}
\frac{\partial f}{\partial w_1} &= \frac{\partial f}{\partial w_1}(\frac{1}{1 + e^{-(w_1x_1+w_2x_2)}}) + \frac{\partial f}{\partial w_1}(0.5(w_1^2 + w_2^2)) \\
&= \frac{0 - e^{-(w_1x_1+w_2x_2)}x_1}{(1 + e^{-(w_1x_1+w_2x_2)})^2} + w_1 \\
&= \frac{-e^{-(w_1x_1+w_2x_2)}x_1}{(1 + e^{-(w_1x_1+w_2x_2)})^2} + w_1
\end{aligned}$$

$$\begin{aligned}
\frac{\partial f}{\partial w_2} &= \frac{\partial f}{\partial w_2}(\frac{1}{1 + e^{-(w_1x_1+w_2x_2)}}) + \frac{\partial f}{\partial w_2}(0.5(w_1^2 + w_2^2)) \\
&= \frac{0 - e^{-(w_1x_1+w_2x_2)}x_2}{(1 + e^{-(w_1x_1+w_2x_2)})^2} + w_2 \\
&= \frac{-e^{-(w_1x_1+w_2x_2)}x_2}{(1 + e^{-(w_1x_1+w_2x_2)})^2} + w_2
\end{aligned}$$

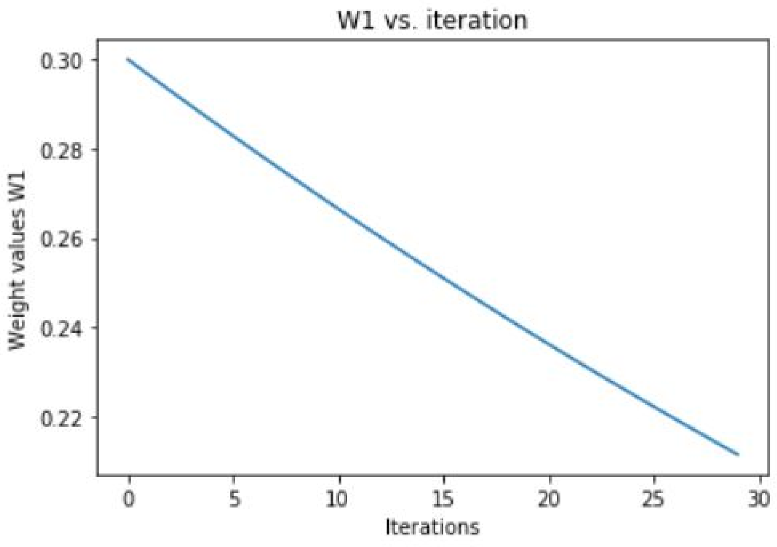
$$\begin{aligned}
\frac{\partial f}{\partial x_1} &= \frac{\partial f}{\partial x_1}(\frac{1}{1 + e^{-(w_1x_1+w_2x_2)}}) + \frac{\partial f}{\partial x_1}(0.5(w_1^2 + w_2^2)) \\
&= \frac{0 - e^{-(w_1x_1+w_2x_2)}w_1}{(1 + e^{-(w_1x_1+w_2x_2)})^2} + 0 \\
&= \frac{-e^{-(w_1x_1+w_2x_2)}w_1}{(1 + e^{-(w_1x_1+w_2x_2)})^2}
\end{aligned}$$

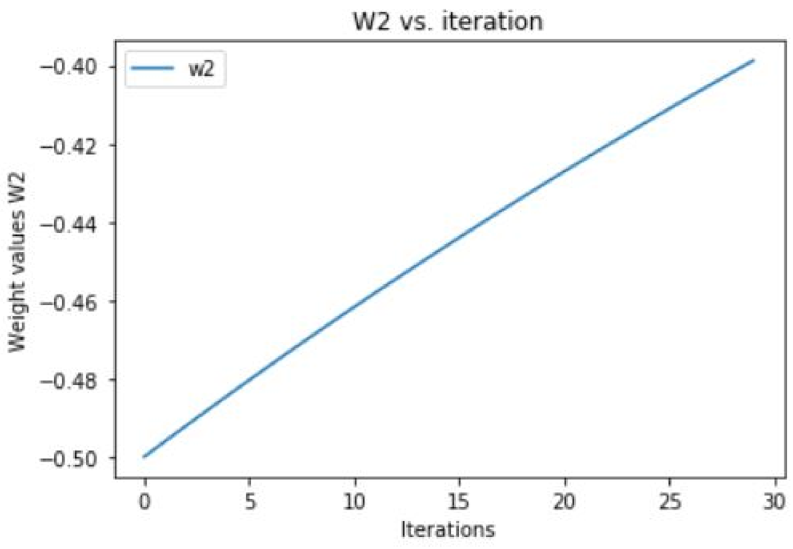
$$\begin{aligned}
\frac{\partial f}{\partial x_2} &= \frac{\partial f}{\partial w_2}(\frac{1}{1 + e^{-(w_1x_1+w_2x_2)}}) + \frac{\partial f}{\partial w_2}(0.5(w_1^2 + w_2^2)) \\
&= \frac{0 - e^{-(w_1x_1+w_2x_2)}w_2}{(1 + e^{-(w_1x_1+w_2x_2)})^2} + 0 \\
&= \frac{-e^{-(w_1x_1+w_2x_2)}w_2}{(1 + e^{-(w_1x_1+w_2x_2)})^2}
\end{aligned}$$

(b) Start with the following initialization: $w_1 = 0.3, w_2 = -0.5, x_1 = 0.2, x_2 = 0.4$, draw the computation graph. Please use backpropagation as we did in class.
You can draw the graph on paper and insert a photo into your report.
The goal is for you to practice working with computation graphs. As a consequence, you must include the intermediate values during the forward and backward pass.

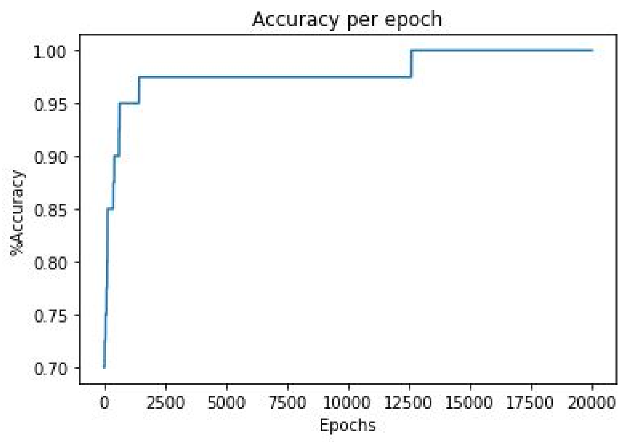


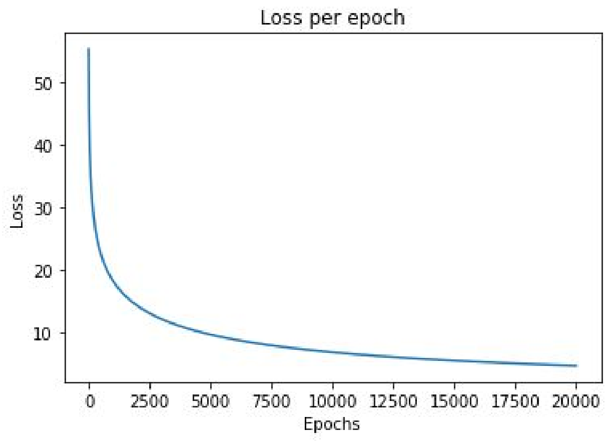
(c) Implement the above computation graph in a Jupyter Notebook using numpy. Use the values of (b) to initialize the weights and fix the input. Use a constant step size of 0.01. Plot the weight value w_1 and w_2 for 30 iterations in a single figure.





2. **Before you get to work, please carefully read the instructions for both Problem 2 and Problem 4. This will likely save you a lot of time, since if done properly, you will be able to re-use the code for Problem 2 in Problem 4.**
   Your goal in this exercise is to implement a linear classifier from scratch, train it on a toy dataset, and submit the results testing on the same dataset. Recall, that a linear classifier is given by $f(x, W, b) = Wx + b$. To quantify its performance, you will pass its output through a softmax function and then use Kullback-Leibler divergence to measure agreement with the target output.
   Write a python program that iteratively computes a small step in the direction of negative gradient. Apply your code to the data which can be downloaded here (`https://piazza.com/class_profile/get_resource/jxcftju833c25t/k0ib3w49ohs4il`). **Please go through the README.txt carefully before you start.** Initialize your optimization with all elements of W, b equal to zero. If your algorithm converges to a correct solution that can linearly separate the given data samples, submit that solution; if it does not, indicate which part of the dataset cannot be linearly separated.
   Plot the KL divergence loss and the classification accuracy w.r.t the iteration to see how they are changing as the iteration increases. Attach these two plots to your report. These visualizations would help you debug your code. So it is recommended to implement them early.

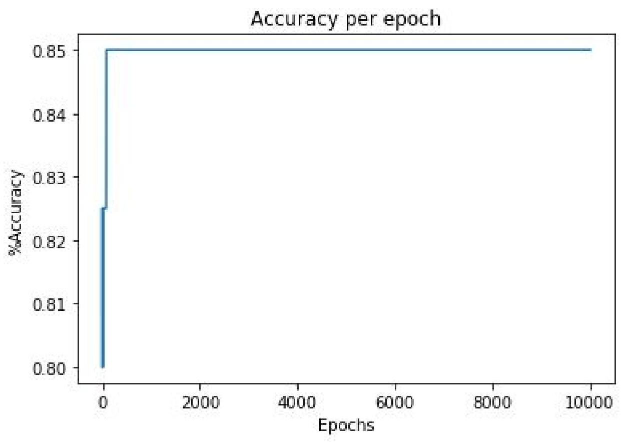


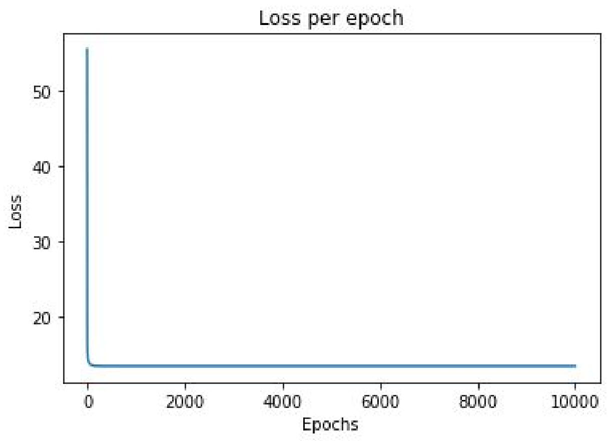


```
Accuracy: 100.0%
```

3. In this problem, we want you to compare the performance of single-layer and multi-layer classifiers.

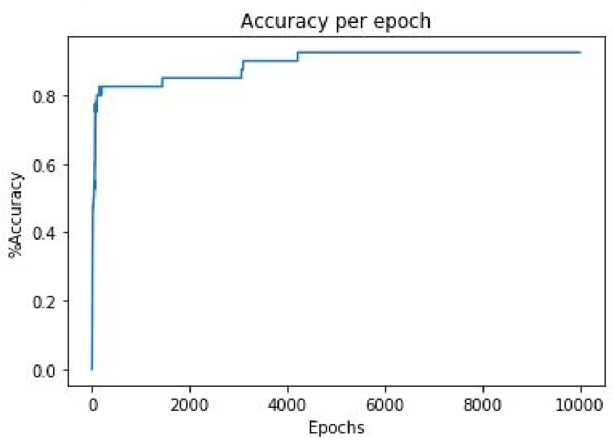
   (a) Please download the dataset here (`https://piazza.com/class_profile/get_resource/jxcftju833c25t/k0ib3w49ohs4il`). This data is also two dimensional. With this dataset, retrain the model developed in Problem 2 and report its classification accuracy on the same dataset.

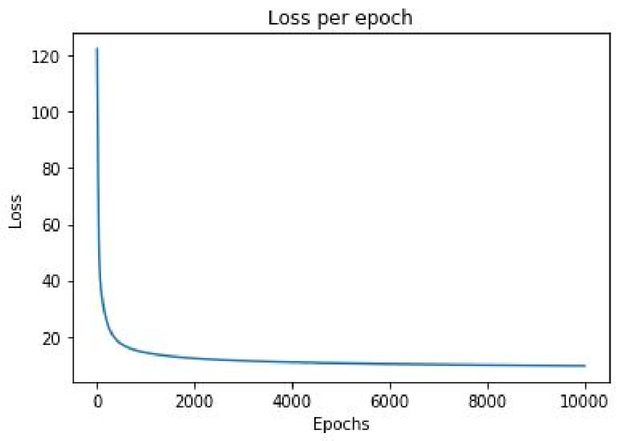




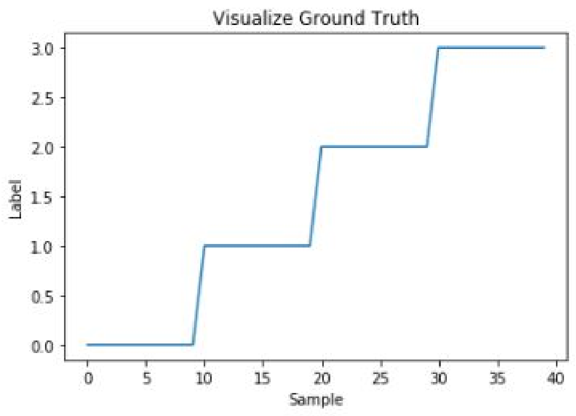
```
Accuracy: 85.0%
```

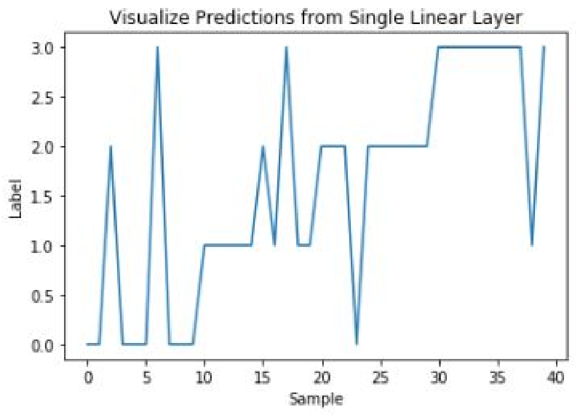
(b) Please add one more layer of perceptrons.* Use the ReLU activation functions for the first hidden layer.** For the output layer, use the softmax function. Initialize all weights randomly using the *numpy.random.randn()* function. Use the KL divergence loss. Train this model on the dataset and report its classification accuracy on the same dataset. Does it classify the data better than (a)?
Answer:
It classifies it better than (a), the deeper net allows for more feature selection.

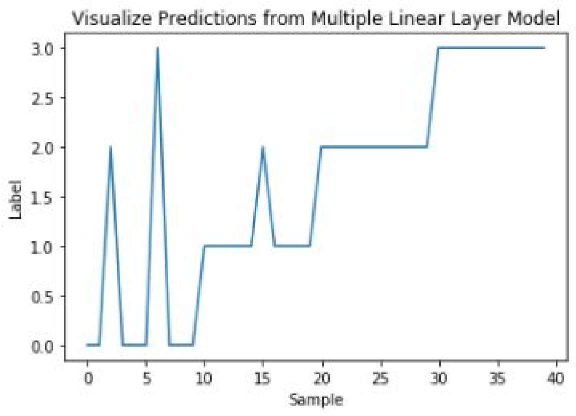




```
Accuracy: 92.5%
```







Please visualize the groundtruth labels and your model's predictions using in (a) and (b) as separate figures. Please also plot the loss vs the training iterations for both subproblems. Attach these figures to your report.

* If the output of a single layer is $f(x, W_1, b_1)$, then adding another layer basically means that the output is $g(f(x, W_1, b_1), W_2, b_2)$: the second layer uses the first layer's output as its input. Therefore, the output dimension of f should match the expected input dimension of g. In our case, the output dimension from the ReLU activation in the first layer should matach the input dimension of the second layer.

** The ReLU activation function is a nonlinear function defined as:

$$\mathrm{ReLU}(x) = \max(0, x)$$

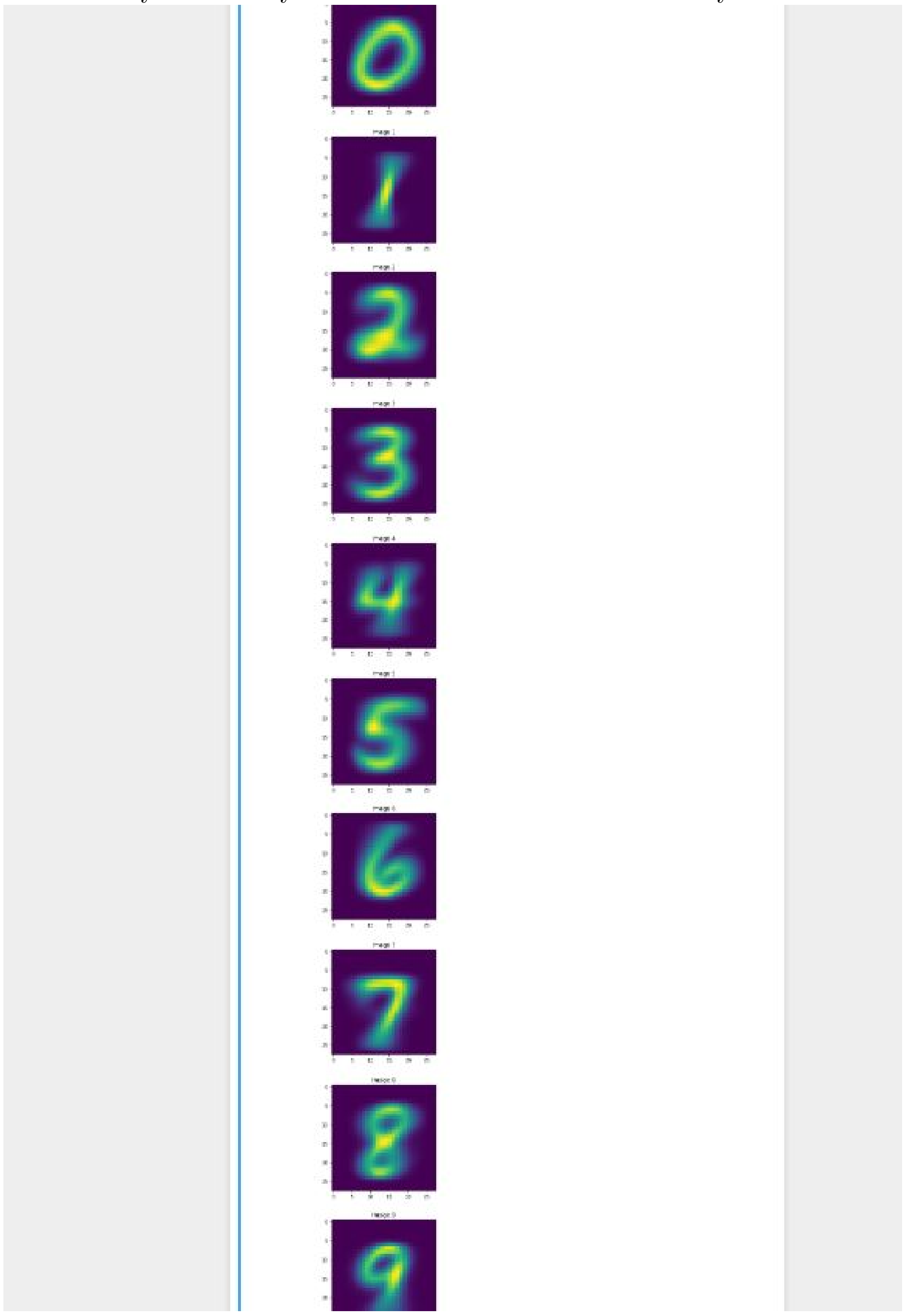
If we use ReLU as our activation function for the first layer, instead of $f(x, W_1, b_1) = W_1 x + b$, we have

$$f(x, W_1, b_1) = \mathrm{ReLU}(W_1 x + b) = \max(0, W_1 x + b)$$

The nonlinearity of the ReLU function adds nonlinearity to the model.

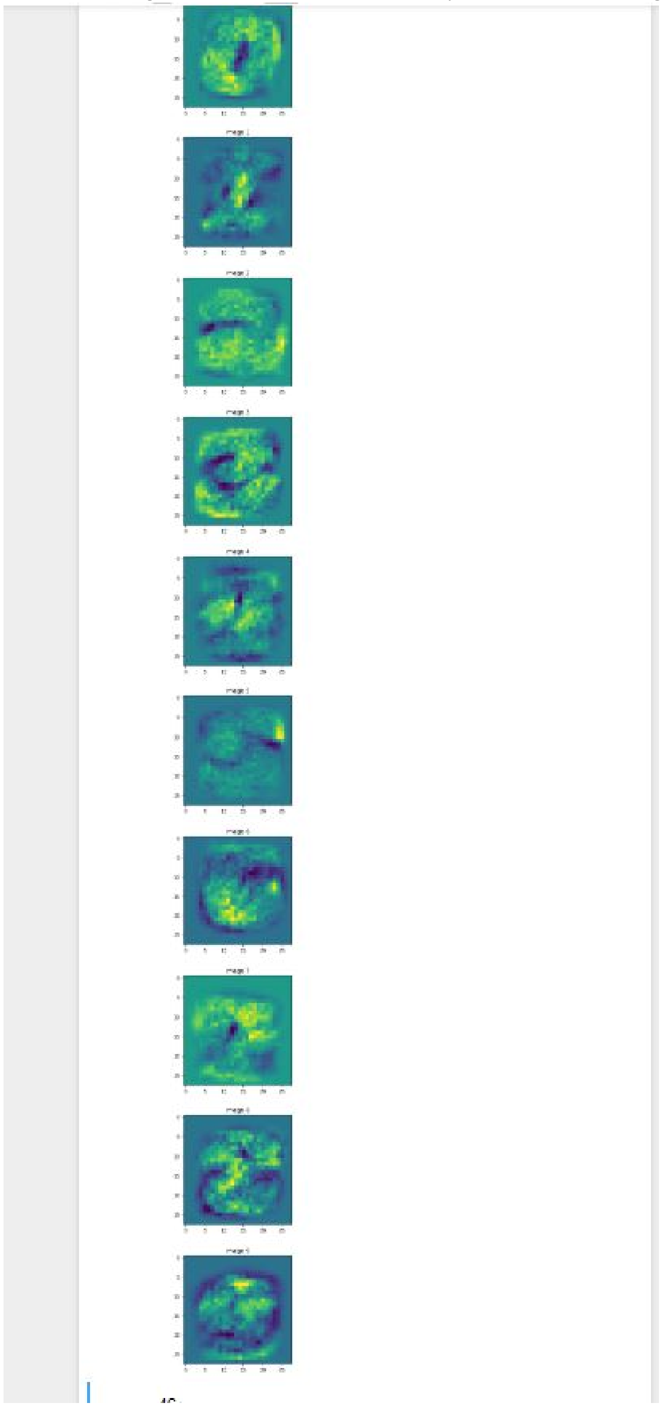
4. In this problem, we want to test what we learned on a more complex dataset. Please download the MNIST dataset (http://yann.lecun.com/exdb/mnist/), which has been divided into training set and test set.

   (a) Per digit, please compute an average "image" of that digit on the training set. Use the average of every digit as one row in a weight matrix to implement a linear classifier. Add softmax to the output. Visualize the average "image" ("templates") by reshaping each row to the original dimension of the MNIST dataset. Attach these "template" images to your report and report the accuracy of your linear classifier on the test set using the average "image" model.
   Answer:
   The accuracy achieved by this model was 63.1 Percent Accuracy



   (b) Now, let's try to improve on our naive "template" by training the weights. Similar to Problem 2, initialize the weights using the templates you have derived in (a), take small steps in the direction of the negative gradient and iteratively update the weights until the model converges. Visualize the weights as "template" images as we did in (a).

Attach these images in your report and report the accuracy of this model on the test set. How do these "template" images compare to those in (a)?
Answer:
These weights are more general, they look fuzzier which suggests higher variance in decision making. More clear means that the model overfits to specific handwritten number images. Our achieved accuracy for the test images is 86 Percent Accuracy.



(c) Generate your own handwritten images (at least 15 images), fix the image size the same as those in MNIST, and test your models in (a) and (b) with your image. Does it work? It is recommended to generate images which are similar to those in the MNIST dataset. Attach you generated images to your reports and report the test accuracy on them.
Answer:
Testing (a) and (b) on these generated images outputted an accuracy of 80 Percent for (a) and 87 Percent for(b). This suggests better generalization for the trained model. The images were purposefully generated through averaging random images through the

data set, this is to ensure properly machine readable data along with generalization testing since these images are still new data.