Homework 4
600.482/682 Deep Learning
Fall 2019

Jason Zhang jzhan127

October 5, 2019

Due Fri. 10/04 11:59pm.
Please submit a zip containing a report (LaTeX generated PDF) and
1 python Jupyter Notebook (one for all problems)
to Gradescope with entry code MKDPGK

1. We have learned that a two-layer MLP can model polygonal decision boundaries. You are given data sampled from a "two-pentagons" decision boundary (lecture 5, slide 89). Data points within the pentagons are considered "true" (labeled as 1) while data points outside are considered "false" (labeled as 0). The data can be downloaded here: https://piazza.com/class_profile/get_resource/jxcftju833c25t/k1277l1l1v41yi. The decision boundary can be modeled with a total of $2 \times 5 + 2 + 1$ neurons with threshold activation function. There is no need to manually split the data, only consider results on this one single dataset for this toy example. Please load the data via *np.load()* function. You will obtain a numpy array with a shape of $(60000, 3)$. The first two colomns are x, y coordinates and the third one represent the labels. The vertices of the polygons are (500, 1000), (300, 800), (400, 600), (600, 600), (700, 800), (500, 600), (100, 400), (300, 200), (700, 200), (900, 400). **Please use the first** 50000 **points as your training set and the rest 10000 as your test set.** Only Numpy, Scipy, OpenCV, Matplotlib, and Pillow are allowed for this problem. Using advanced python packages may end up point loss.

   (a) **Manually** setup an MLP with threshold activation by selecting weights that can model the above decision boundary and verify that the all the 60000 samples in the data are classified correctly. Use

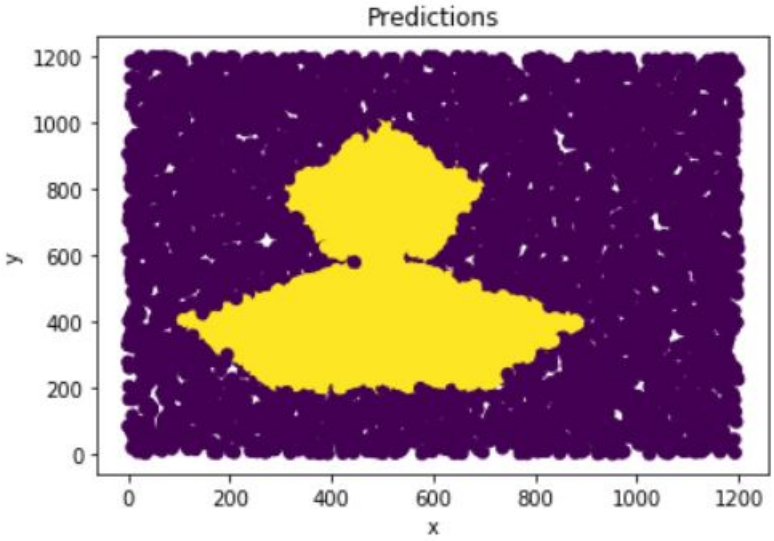
   $$y_i = \begin{cases} 1 & w_i^T x + b_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

   as the threshold activation function. You will also need to implement the and gate and or gate in your MLP to get the correct model. In your report, show how you set up the model; visualize the prediction result on the test set with your model. You may use different color points in your plot to indicate different classes.

   Answer:

   To set up this model I had to first construct the lines of the boundary that we want to separate our data by. Given the points of the vertices of the boundary shape, I found the slope, which will be the associated weight for our prediction. Once we have the slopes and intercepts for each of the boundary edges we use those to determine if a particular sample is within that boundary or not, which will effectively tell us its classification (1, 0). Now since we have those two different pentagons that a data point could potentially lie in, we must construct an AND gate for each of the separate pentagons(to see which one it lies in), if a data point lies in either one of the shapes, our neuron will fire(we will return 1). For that we will need to construct an OR gate. In the first layer we construct the two AND gates needed for each of the shapes. For each shape, we check if a point is inside by checking if a point is above or below the boundary lines, if all boundary lines are satisfied, then we know that the point is within the shape, else if is not. We do this effectively by multiplying the boolean values of each edge satisfaction, and if we get a 1

that means we satisfied all boundary conditions for the edges of that shape, otherwise it will return 0. Now we aggregate this data for both the shapes, and if either one returns that a point is in their shape, then we will return that this data point is within the boundary. To implement this, we simply sum the output of each of the AND layers, if we get a value greater than 0, then we know that it is in the boundary, otherwise it is not.



(b) Now, we have found one network model that can perfectly classify this problem. The above manual setup used threshold activations that are non-trainable, because it is not differentiable when $w_i^T x + b_i = 0$. Consequently, please replace all the 'threshold' and the and/or gate with the sigmoid activation,

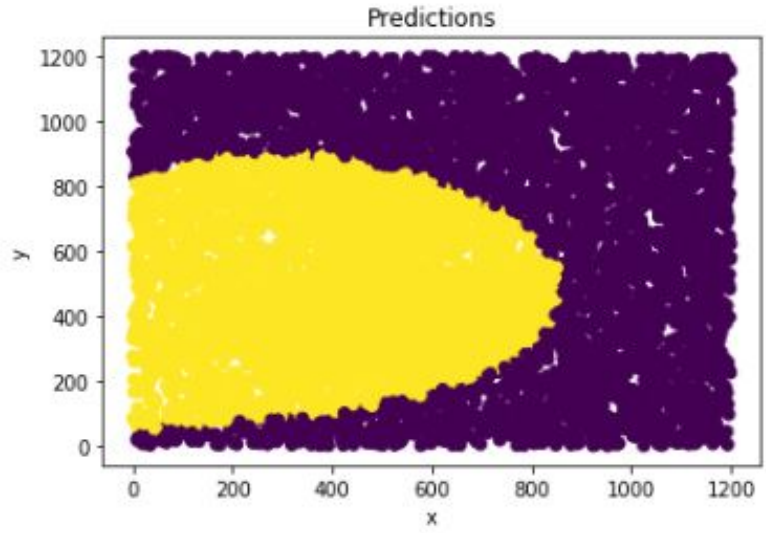
$$y_i = \frac{1}{1 + e^{-(w_i^T x + b_i)}}$$

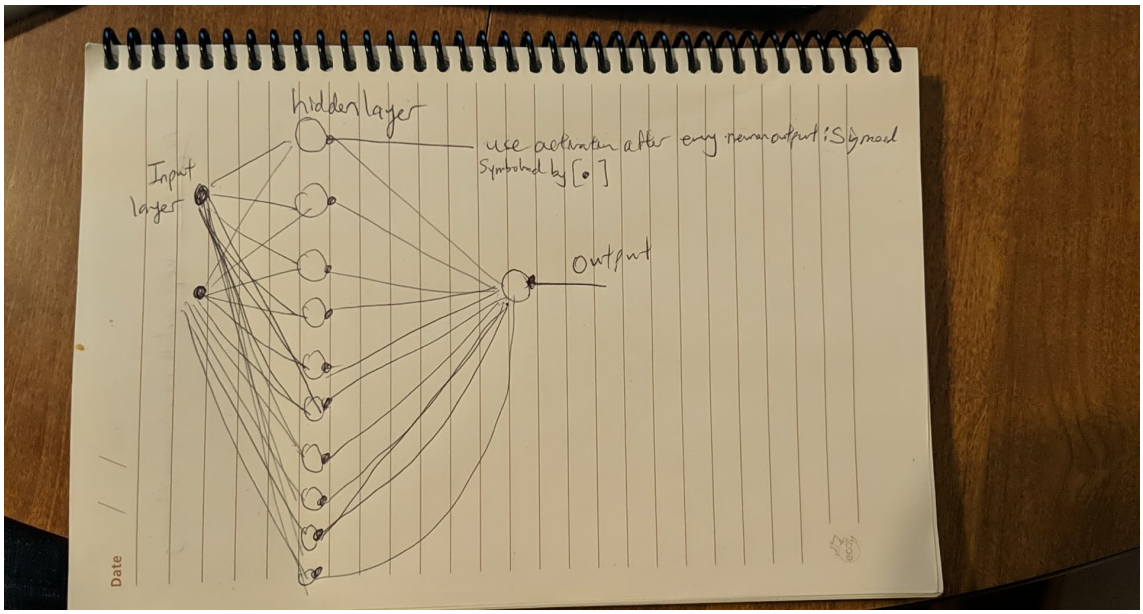
and change every node/perceptron such that its parameters can be trained. Normalize the data if you think it is necessary. Initialize the weight using the method we learnt in class and use gradient descent to optimize the parameters. Visualize your test output. In visualization, consider the last layer output > 0.5 as "true", i. e. output is 1. Please try training your model for multiple times (e.g. 5) and report your findings. Train for at least 300 epochs for each training. Are you able to find a solution that performs almost as good as the "manual" solution in (a)? In your report, please attach the visualization of the prediction for the test set; the training loss and testing accuracy change w.r.t. the epoch. Also, briefly sketch your MLP to show the neurons, the connections between the neurons, and where you introduce the activation function. You may draw it on a paper and attach the scan.

Answer:

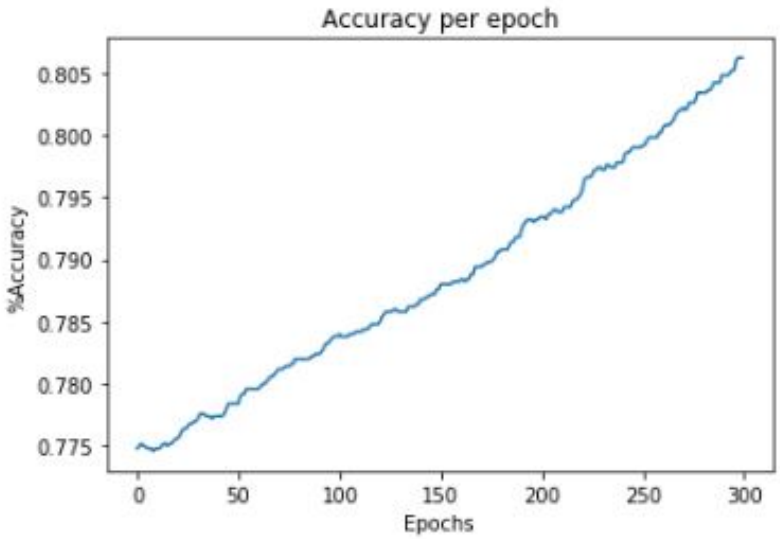
For my solution, I had to train with a batch of the given examples due to the fact that my computer does not have enough memory to store that many gradients (on the order of 50000). Thus I tested with a batch that is about 5000 and ran it for 300 iterations and 5 times. The accuracy was about 81% which is relatively good for a simple neural net. I used He initialization for the weights.

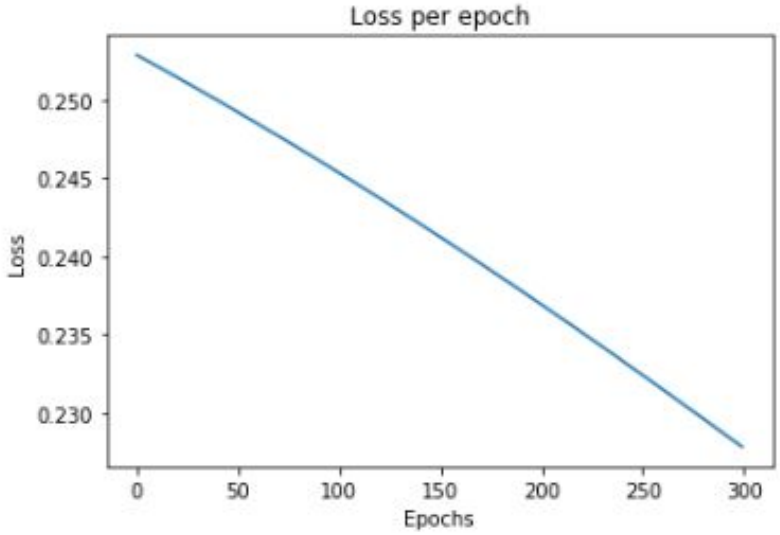
```
Accuracy: 81.94%
```





```
Accuracy: 77.48%
Epoch: 0, Training Accuracy: 77.48%
Epoch: 100, Training Accuracy: 78.40%
Epoch: 200, Training Accuracy: 79.34%
```





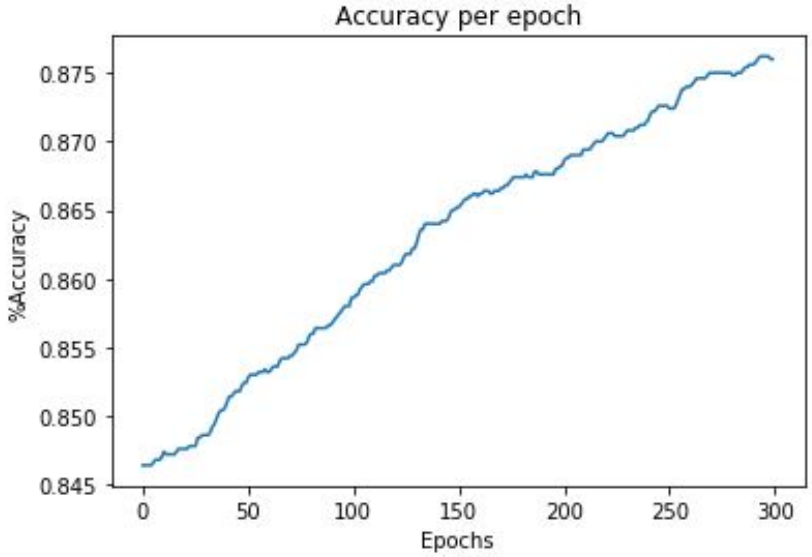

```
Accuracy: 80.62%
```

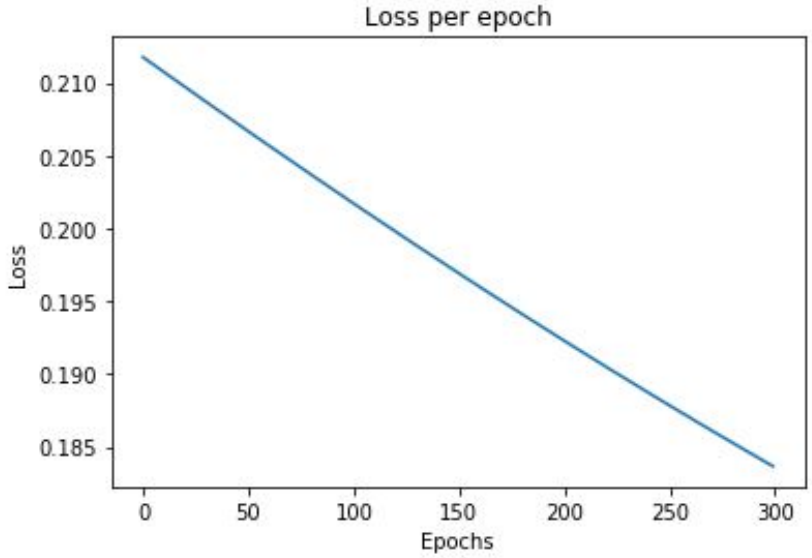
(c) Now, define an MLP with a larger capacity (increase the depth and width) and see whether you can achieve higher classification accuracy. Visualize your prediction on the test set and briefly sketch the MLP in your report.

Answer:

I increased the number of neurons in the hidden layer. I tested with more layers, but I ended up having more over fitting issues leading to worse accuracy. With increased neurons in the first layer, 10 to 30, I was able to achieve a better classification accuracy of about 90%. Looking at the prediction, it resembles the tight bounds of 1a more than 1b, thus it is a better model. Having more neurons in the hidden layer makes for better accuracy since more neurons are used for feature detection so we can have more specificity in our boundary.

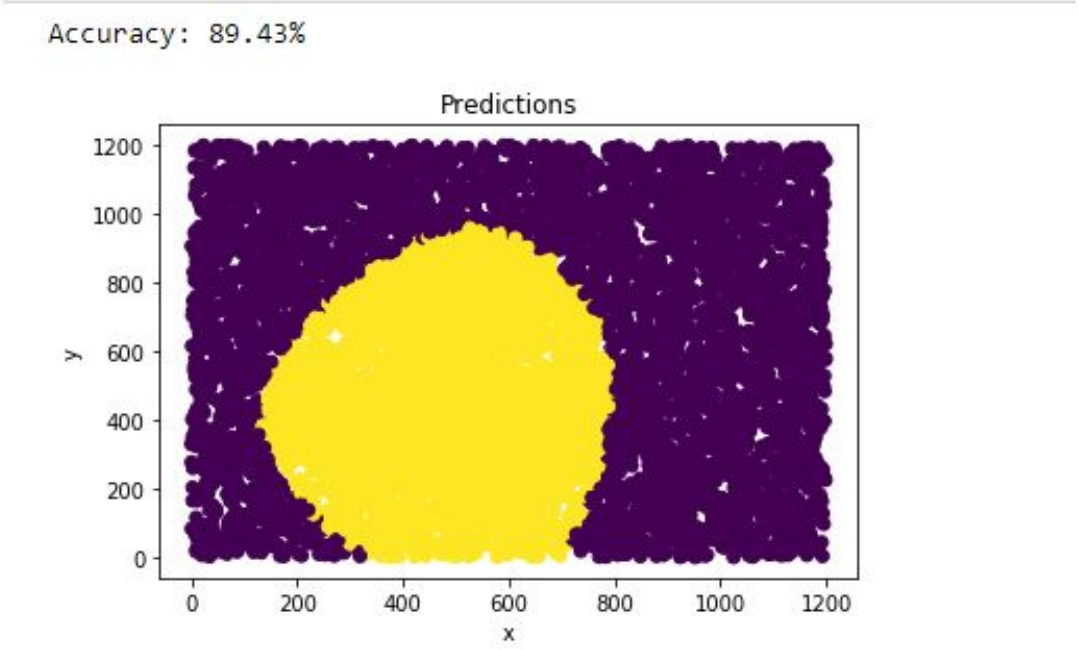
```
Epoch: 0, Training Accuracy: 84.64%
Epoch: 100, Training Accuracy: 85.86%
Epoch: 200, Training Accuracy: 86.86%
```

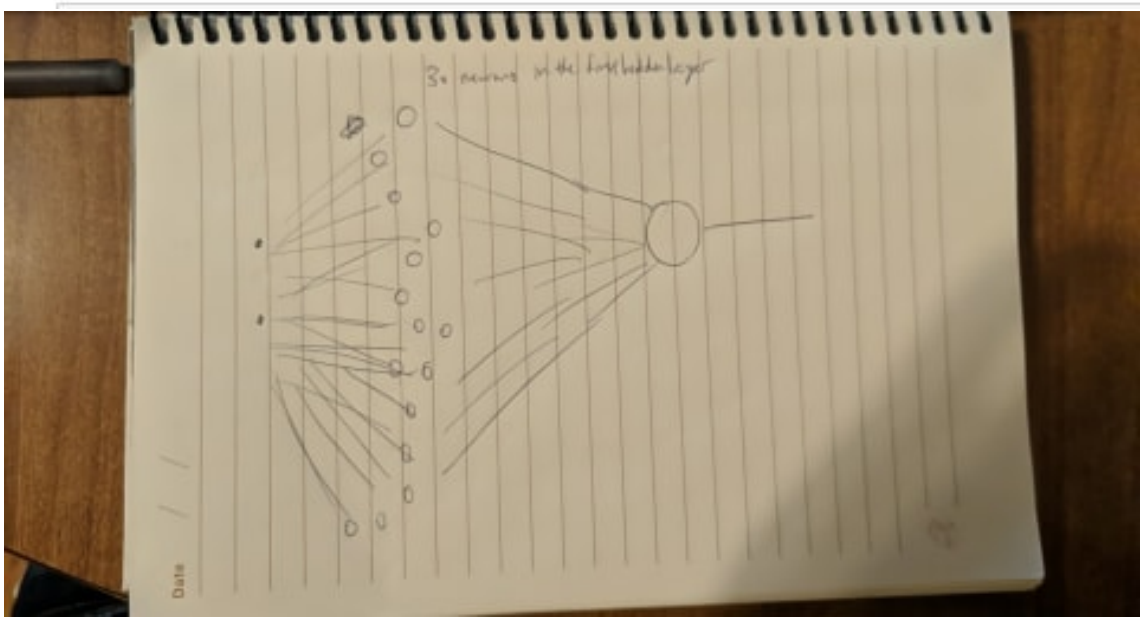





```
Accuracy: 87.6%
```

Classification Accuracy:



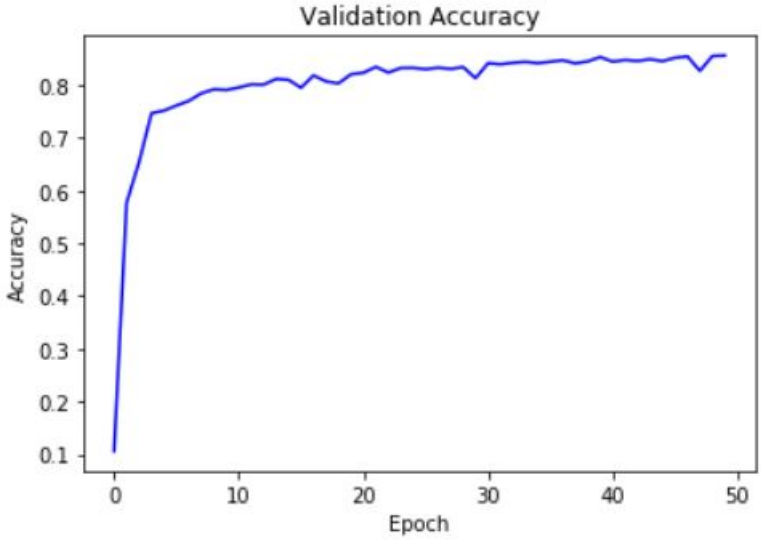


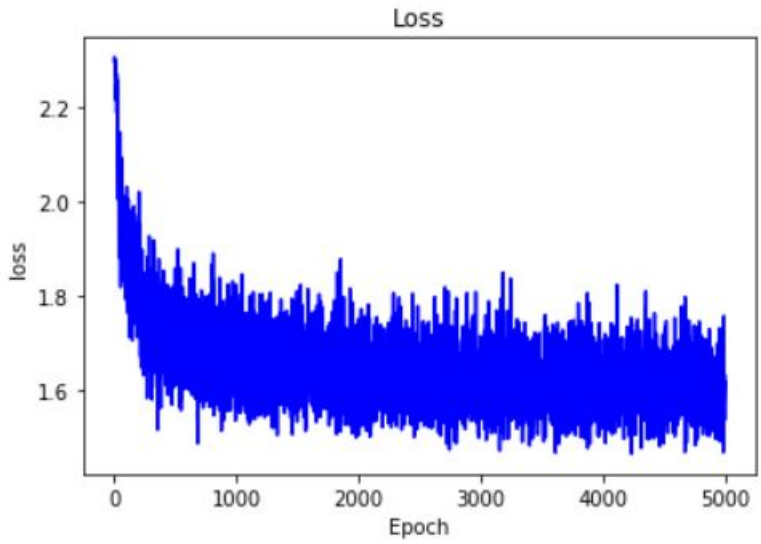
2. In this problem, we start to look into the convolutional neural networks (CNN). We will be working with the FashionMNIST dataset. Pytorch is allowed to use in this problem. You may look up here to install Pytorch on your computer: https://pytorch.org/get-started/locally/. The full FashionMNIST dataset can be downloaded from the official website here: https://github.com/zalandoresearch/fashion-mnist. Or you may use the Pytorch API for loading the FashionMNIST dataset. It goes like this: "from torchvision.datasets import FashionMNIST". For more details, look up here: https://pytorch.org/docs/stable/torchvision/datasets.html#fashion-mnist

   (a) Design a small CNN (e.g. 3-4 hidden layers) using the tools we learnt in class (e.g. convolution and pooling layers, activation functions, regularization). **Briefly** describe your architecture in your report and explain why you design it this way (e.g. I use ReLU activation here because...) **Please design the network architecture by yourself and do not discuss with your colleagues or use code available online for this subproblem. This is NOT about performance.** For reference, if you achieve more than 85% accuracy on the test set, move on to the next subproblem.

Answer:
My network is as follows:
Conv1: A convolutional layer that has 16 filters to learn, each with a 3x3 kernel and stride of 1.
Conv2: A convolutional layer that takes 16 filters and outputs 32 filters using a 3x3 kernel with stride 1.
FC1: A fully connected layer that takes a vector of the filters in Conv2 after flattening the output.
FC2: A fully connected layer that has 64 neurons and outputs 10 a vector of 10 representing the class scores.
I pass my input initially into Conv1 and put it through a ReLU activation function and max pool the output. I use ReLU here to introduce nonlinearity into my output. This makes for ease of differentiability and better generalization. ReLU helps with the vanishing gradient problem as well so I felt it was beneficial to use it as the activation function. I then max pool the output in order to reduce the feature map output from each filter, this reduces computational complexity(fewer stored parameters), and max pool will output the largest value in a window, which will output the strongest features in a feature map, which could be things like edges, etc. I then pass on this output to the next Conv layer Conv2. Conv2 will do the same thing as Conv1 and the reasoning is the same, the only difference is that Conv2 outputs more feature maps(more channels). The reasoning is because I want to be able to learn/discern more features in the base feature maps from Conv1. This will be quintessential for learning more complex shapes in images. I learn less filters in Conv1 due to the fact that I don't need much specificity in low level features. After Conv2, I flatten the feature maps from Conv2 and pass it to my fully connected layer FC1. This layer is to aggregate all the complex features and try to make linear combinations of those features. I then pass the output through another ReLU to help with gradient computations and nonlinearity before I pass the output to my final layer FC2. FC2 will make the final linear combinations of features before being passed through a softmax function, which will return probabilities associated with each class label. The FC and softmax layers are to make these combinations of the features we learned in the convolutional layers and output a probability denoting the probability that a particular image belongs to a particular class. I use softmax because it is a popular function that is well suited for multiclass predictions, since it outputs probabilities of belonging to each class. Finally I use cross entropy loss as my loss function which goes perfectly with the softmax activation since they both are probabilistic estimators of a distribution, thus cross entropy can give a good estimation of error for softmax outputs since it will determine how well the softmax outputs model the true distribution.

(b) Train your network and report the results on the test data of Fashion MNIST. Visualize your training loss and testing accuracy w.r.t the training epoch and attach this plot to your report.
Answer:
This model achieved 85% accuracy on the test data set.





(c) Compare your architecture and results to one or more colleagues in our class. **Briefly** state the differences between your and your colleagues' architectures and results, and reason about why one or the other may perform better. **It really doesn't matter if your model does not outperform your colleagues'. We care more about the reasoning but not the result. So spend more time in reasoning.**

Answer:

For one of my colleagues that I compared to, my network achieved better scores than theirs: 85% to 76%. The differences are mainly parameter differences. We both have a similar network in the sense that there are convolutional layers and fully connected layers before decision making, but we differed in number of channels per layer, padding usage, and activation function usage. The other network used sigmoid activation rather than softmax activation and had only about 6 channels outputting from the convolutional layer. Mine had more convolutional layers with more channels per layer. My network is deeper and has more features maps which in turn means it can discover more details in images. My network also properly uses softmax activation for my output layer instead of sigmoid. Softmax would be more useful in this problem due to the fact that softmax is suppresses outputs for other classes and outputs for the class that has the strongest probability. Sigmoid will just return probabilities of the image being of a particular class, but there could be strong probability for multiple classes, making it less informative.

(d) Now, design an improved architecture based on your discussions and try to improve on

your model's performance. Implement at least one of dropout and batch normalization in your model; use data augmentation; play with different optimizers. Try to beat the model you have in (a). Report your best results on the test set of Fashion MNIST. Visualize your training loss and testing accuracy w.r.t the training epoch and attach this plot to your report.

Answer:

After comparing with other models, I changed my implementation to use a larger kernel size in order to learn larger features per convolution. I then add the proper padding to not decrease the the size of the image too much to lose data. Then to decrease possible overfitting I added batch norm layers and dropout after the convolutional layers in order to decrease possibility of overtraining. Then I use only one fully convolutional layer with softmax in order to get the proper class output. I also tried using ADAM which increased performance due to the fact that ADAM lets us learn parameters for the learning rate IE by changing the momentum. Adding any data augmentation IE by image rotation decreased the accuracy overall, this may be due to the fact that my network is not deep enough. The best accuracy I was able to achieve was about 90% with the current network in the notebook.