Assignment 3: Neural Networks and Deep Learning

UVA CS4774

March 30th, 2022

Owen Richards

Multilayer Perceptron (MLP):

Summary of my Network:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_19 (Dense)             (None, 512)               401920

 dense_20 (Dense)             (None, 256)               131328

 dense_21 (Dense)             (None, 128)               32896

 dense_22 (Dense)             (None, 64)                8256

 dense_23 (Dense)             (None, 32)                2080

 dense_24 (Dense)             (None, 10)                330

=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

Fig 1: Summary of Network

This is a multilayer perceptron (MLP) model. This model is an artificial neural network that generates a set of outputs from a set of inputs. There are several layers of input nodes connected as a directed graph between the input and output layer in an MLP. My model uses six layers of input and the MLP uses backpropagation for training the network. This model is a deep learning method. Additionally, in the model above I am using trainable parameters which value is adjusted during the training as per their gradient. Whereas non trainable parameters are values no optimized during the training as per their gradient. MLPs can approximate any continuous function by combining several neurons which is organized in multiple layers. The first layer is the input layer and last layer is the output layer. The middle layers are the hidden layers and in my case is use ‘sigmoid’ with the final layer having activation of ‘softmax’. In my MLP it takes in the input size of 28*28 and outputs the first layer of 512 using a sigmoid activation. This pattern is continued for a few more layers until the final layer where it input is 32 and the output is 10 with an activation of ‘softmax’. This finds the best approximation to the classifier that maps the input to the output by a mapping. Also, the units and the input to the next layer are the same. Finally, the loss function verifies that my MLP model is correct.

Plots:

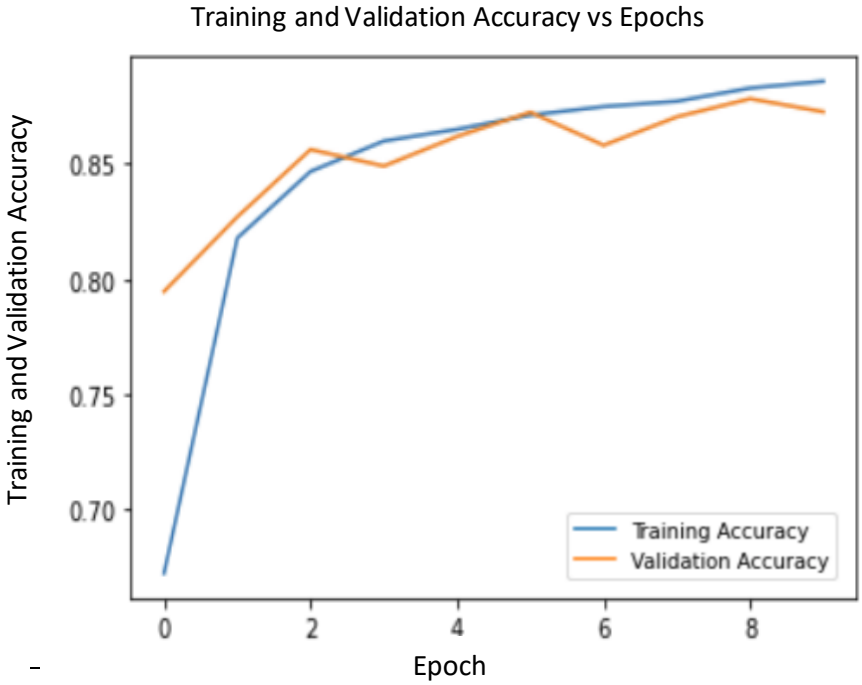

Fig 2: Training and Validation Accuracy vs Epoch

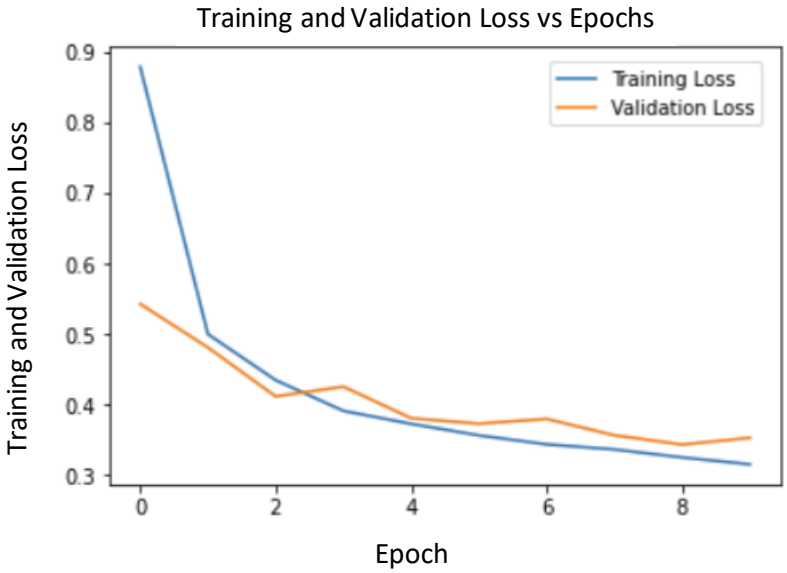

Fig 3: Training and Validation Loss vs Epoch

The plots in Figure 2 and Figure 3 show no problem with my model with a validation accuracy starting around 80% on the first epoch. I decided to use 10 epochs because more each epoch took time and I found that only 10 epochs were needed to sufficiently show that the model works. Any more epochs would just take a longer time and would not change anything about the trend I already saw. The plots verify that my MLP model works as expected.

Visualizations of Weights:

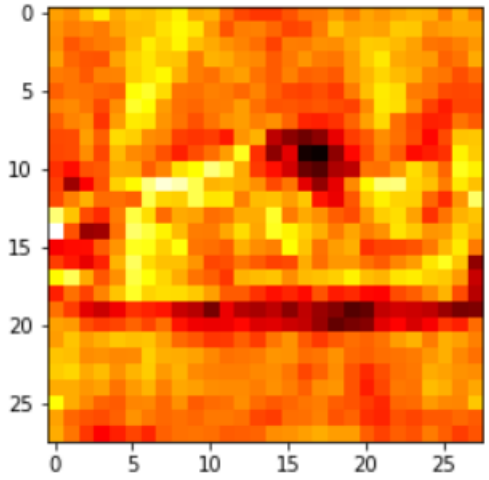

Fig 4: Visualization of Weights for a Shoe

This is a visualization of the weights feeding into the unit of my first hidden layer by displaying greater weight values as lighter pixels and lower weights as darker pixels. For the unit above is the input is a shoe and the visualization of weights is attempting to us the different weighs to create a visual image of a shoe. This visualization resembles a shoe which once again verifies my MLP model.

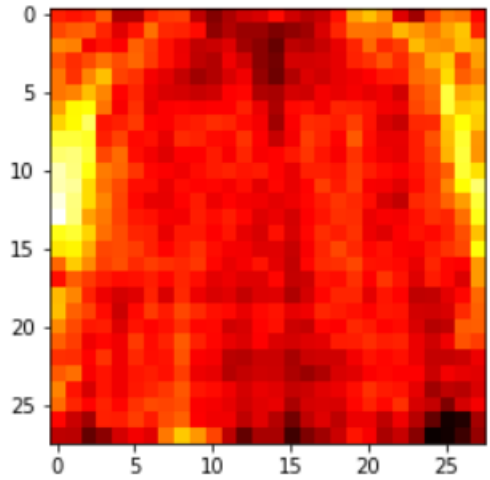

Fig 5: Visualization of Weights for a Button Up Shirt

In figure 5, the visualize weights function is causes this unit to resemble an image of a button up shirt. The unit is using the different weight values feeding into my hidden layer to create an image by attributing different weights to different pixel colors.

Mean Squared Error for a Loss Function:

Mean square error is a good choice for a loss function for regression problems. However, for classification problems it is better to use binary cross-entropy since MSE is a bad choice for binary classification. This comes down to two main reasons. The first reason is that with mean squared error, the underlying data has been generated from a normal distribution. This dataset can be classified into two categories which means it is not a normal distribution. This makes choosing MSE as a bad choice. The second reason is that the MSE is non-convex for binary classification. Often gradient-based techniques are used to find the optimal values for coefficient by minimizing the loss function. If the loss function is not convex, it is not guaranteed that we will read the global minima, rather we might get stuck at the local minima. If MSE was to be used for this problem, it is not guaranteed we minimize the cost function. MSE expects to have real-valued inputs in range $(-\infty, \infty)$. Conversely, binary classification models output probabilities in range (0,1) though the sigmoid function.

Convolutional Neural Network (CNN):

Summary of my CNN:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320

max_pooling2d_3 (MaxPooling  (None, 13, 13, 32)        0
2D)

conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496

max_pooling2d_4 (MaxPooling  (None, 5, 5, 64)          0
2D)

conv2d_5 (Conv2D)            (None, 3, 3, 128)         73856

max_pooling2d_5 (MaxPooling  (None, 1, 1, 128)         0
2D)

flatten_1 (Flatten)          (None, 128)               0

dropout_1 (Dropout)          (None, 128)               0

dense_25 (Dense)             (None, 10)                1290

=================================================================
Total params: 93,962
Trainable params: 93,962
Non-trainable params: 0
_________________________________________________________________
```

Fig 6: Summary of CNN

This is my convolutional neural network (CNN) summary. Prior to CNNs, manual time-consuming methods were used to identify objects in images. However, CNNs provide a scalable approach to image classification. A CNN is a deep learning algorithm which takes in an input image and the assigns importance to various aspects. The CNN assigns weights and biases to aspects in an image which can be able to differentiate one aspect to another. In a CNN it requires a pre-processing in a ConvNet. ConvNet can learn filters and reduces the image into a form which is easier to process without losing any key features. There are three main layers in CNNs which are the convolution layer, the pooling layer, and the fully connected layer. In the convolutional layer it coverts, all the pixels in its receptive field into a single value. In my model I use Conv2D with a stride of (1,1), kernel size of (3,3) activation of relu, and a filter of 32 for the first convolution layer. The pooling layer replaces the output of the network at certain

locations by deriving a summary statistic of the nearby output. In my model I use pool size of (2,2) and a stride of 2. The fully connected layer performs the task of classification based on the features extracted through the previous layers and their different filters. The convolutional and pooling layers tend to use relu functions, but the fully connected layers a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1. In my model I used the same idea as described about using a convolution layer, a pooling layer, and a fully connected layer. The convolution and pooling layer used relu and the fully connected used softmax. My model works by correctly implementing all three layers and the results of this can be seen in the two following plots.

Plots:

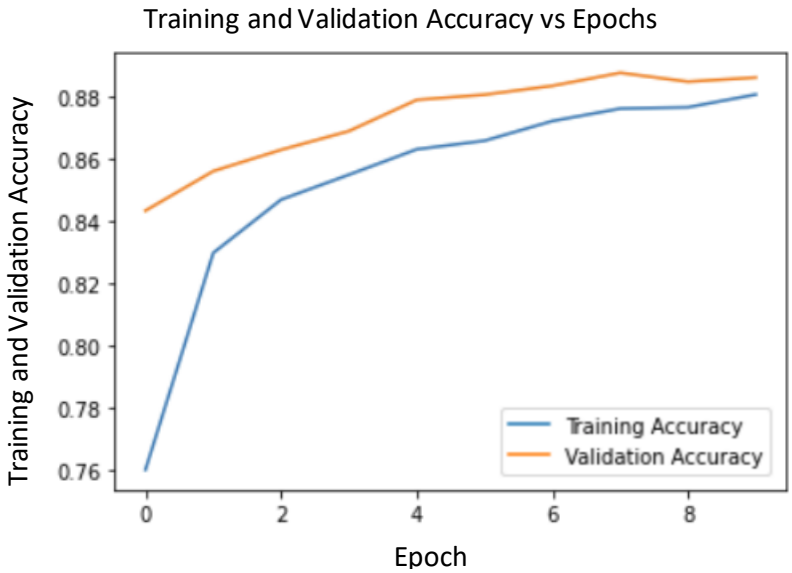

Fig 7: Training and Validation Accuracy vs Epoch

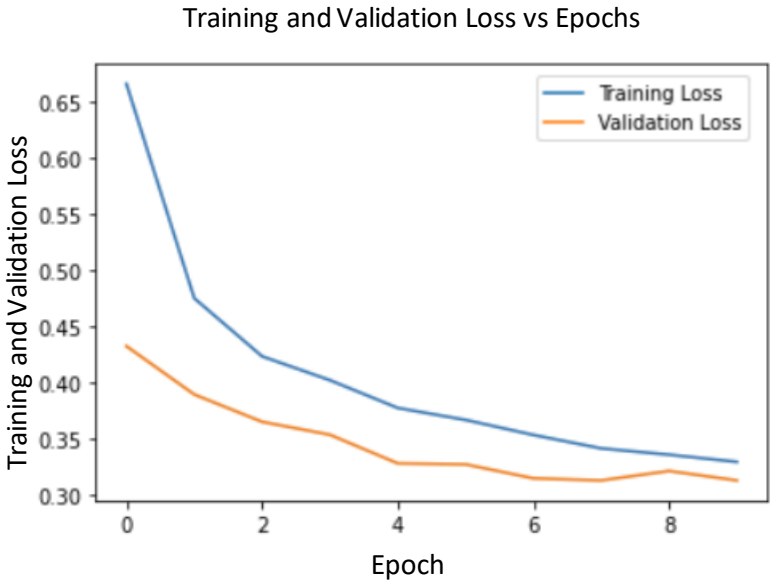

Fig 8: Training and Validation Loss vs Epoch

The plots in Figure 7 and Figure 8 show no problems with my model with a validation accuracy starting around 85% on the first epoch. The one aspect of the plots that I would want to change is the separation of the training and validation lines. Even though that both plots show an obvious trend, I would want to see the training and validation line to touch. However, this is not a problem with my model. Regarding the number of epochs I used, I used the same number as I did for MLP which is 10. I chose 10 because of consistency between MLP and CNN but also because I found that only 10 epochs were needed to sufficiently show that the model works. Any more epochs would just take a longer time and would not change anything about the trend I already saw. The plots verify that my CNN model works as expected.

Matrices Outputted by First Convolution Layer:

The number of matrices that are outputted by my first convolution layer when it receives a single testing image is 32 matrices.

Dimensions of These Matrices:

The dimensions of these matrices would be 26x26 after the first convolution layer.

Dimensions After First Maxpooling Layer:

The dimensions of one of these matrices after passing though my first maxpooling layer would be 13x13.

PCA and Logistic Regression:

Prediction Error Results:

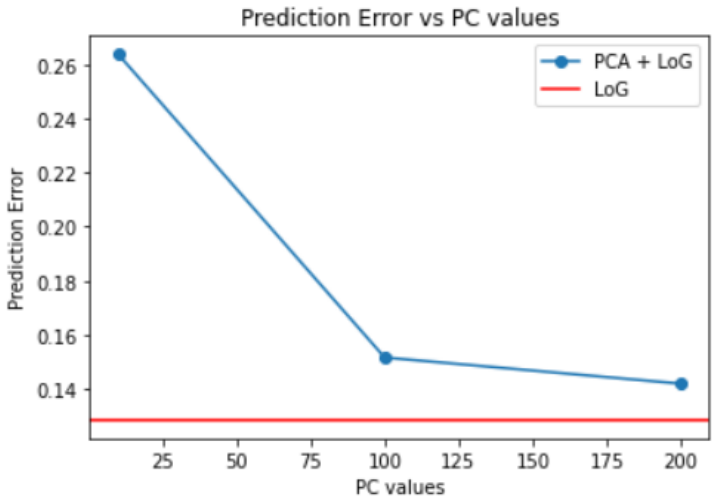

Fig 9: Prediction Error vs PC Values

The three different principal component values I chose for plot shown in Figure 9 was 10, 100, and 200. As shown in the figure above as the value of the principal component increases, the prediction error reduces. However, the figure also illustrates that running the LogRegression without PCA on the dataset will be the lower bound of the prediction error. Prediction error of LoG with PCA is always greater than LoG without PCA. The Principal Components Analysis approximates a high dimensional data set with a lower dimensional linear subspace. Finally, below is a figure which displays each PC value and its respected prediction error.

Type of Regression	PC Value	Prediction Error
PCA + LogRegression	10	0.26373333
PCA + LogRegression	100	0.15166666
PCA + LogRegression	200	0.14200000
LogRegression (No PCA)	Continuous	0.12856666

Compare Results with MLP and CNN:

When comparing the results with MLP and CNN it can be seen that the logistic regression produces less error on the training data than either the MLP or CNN. Additionally, the accuracy of the MLP and CNN can be roughly compared to 1 – LogRegression without PCA. When comparing MLP and CNN with the PCA + LoG regression, the errors are similar. Finally, both PCA + LoG, MLP and CNN share the trend that as the number of principle components or epochs increase, the better the loss and error gets. Below is a table summarizing all classifiers, this table shows the comparison between MLP, CNN and the LogRegression and PCA regressions.

Epoch	MLP Validation Loss Optimizer = Adam Learning Rate = 0.004 Hidden Dim = 128 Hidden Layer = relu Activation = activation	CNN Validation Loss Optimizer = Adam Learning Rate = 0.004 Hidden Dim = 128 Hidden Layer = relu Activation = activation	Logistic Regression Loss
1	0.457	0.385	PC = 10: 0.264
2	0.432	0.343	PC = 100: 0.151
3	0.431	0.374	PC = 200: 0.142
4	0.391	0.336	No PCA: 0.128
5	0.387	0.331	
6	0.379	0.333	
7	0.357	0.328	
8	0.335	0.319	
9	0.336	0.297	
10	0.332	0.292	

Discussion of Running the LogRegression and PCA+LoG on Datasets:

Running LogRegression on the dataset performs better than when running PCA+LoG on the dataset. The performance of first processing the data directly to the LoG model is better than using PCA. This can be seen by the logistic regression loss of both. The difference between the two approaches is that PCA will not consider the response variable but only the variance of the independent variable. Whereas the logistic regression will consider how each independent variable impact on response variable. In this case LogRegression performs better.