CSE 474: Project 2

Peter M. VanNostrand
Department of Computer Science
University at Buffalo
Buffalo, NY 14260
pmvannos@buffalo.edu

Abstract

In this project we developed three machine learning models. The first is a simple three-layer neural network written from scratch in Python. The second is a deeper neural network written in Keras, and the last is a convolutional neural network written also written using Keras. All three networks perform 10 class classification and were trained and tested on the MNIST Fashion dataset.

1 Introduction

Neural networks are a common form of machine learning algorithm often used for solving classification problems. In recent years neural networks, particularly convolutional networks, have become the state of the art for image recognition and classification. In this project we develop three neural network classifiers for the popular MNIST Fashion dataset. This dataset was created to replace the basic MNIST handwritten digit dataset as it became increasingly easy to achieve high performance with modern techniques. The MNIST Fashion dataset is detailed in Section 2, following by a description of the preprocessing performed on this data in Section 3. Next Section 4 details the architecture of the three models developed and Section 5 presents the performance of these models for different parameters.

2 Dataset

For this exercise the MNIST Fashion dataset was used for all training and validation. The dataset is comprised of 70,000 28x28 pixel grayscale images of 10 different types of clothing. Each picture has a corresponding integer label from 0-9 which correspond to image descriptions as listed in Table 1.***Error! Reference source not found.***

Table 1: MNIST Fashion Labels

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

As the images are 28x28 greyscale pixels there is only one channel in the image with pixel intensities ranging from 0-255, Therefore there are $28 \times 28 \times 1 = 784$ features per image.

2 Preprocessing

2.1 Part 1

In part on the MNIST dataset is loaded using the "util_mnist_reader.load_mnist()" function provided by the instructor. This function returns a pair of arrays, the first being an array of images represented as $n_{samples} \times 784$ matrix and the second being an array of $n_{samples}$ integers which represent the correct labels for the corresponding images. Calling this function twice with different specifiers allows us to obtain 60,000 training samples with labels as well as 10,000 testing samples with labels.

As all image pixel values range from [0,255] we divide both the training samples and testing samples by 255 to normalize these values to [0,1]. Then an extra feature of constant 1 is appended to the end of each sample to represent the bias term

Next the labels are converted into one hot encoding. In this encoding each label is represented by an array of size 10 holding all zeroes except for at the location of the correct label where a one is placed. This is done by creating a zeros array of the appropriate size and then iterating over the labels array and setting the corresponding locations to one

2.2 Part 2

In this section data preprocessing is done essentially the same way as in part 1. The training and testing datasets are loaded, normalized to one. In this case the one-hot matrices are created using the Keras utility function to_categorical which produces the same labels matrix as described above.

2.3 Part 3

In this section data is preprocessed differently than in parts one and two. In this case we would like to use 2D convolutional layers in our CNN, these require two-dimensional layers to work properly. To achieve this the function "k.datasets.fashion_mnist.load_data()" is called. This returns a pair of pairs. The first pair is the training images and labels and the second is the testing images and labels.

In this case the images are stored as two-dimensional arrays of 28x28 values, one per sample, and the labels are again stored as a one-dimensional array of integers. The load data function returns 60,000 training samples and 10,000 testing samples.

As in parts 1 and 2 the values range from [0,255] and so are normalized to [0,1] and the labels are converted to one-hot encoding as in part 2.

3 Architecture

3.1 Part 1

For part one a simple three-layer neural network is implemented. The layers of this network are as described

- Input layer – takes the 1x784 input features from an image as its activations
- Hidden layer – full connected layer with n_1 nodes. This takes the 784 activations from the previous layer and computes its activations as $a_1 = sigmoid(w_1 a_0)$
- Output layer – fully connected SoftMax layer with 10 nodes. This takes the previous activations a_1 and generates 10 outputs which correspond to the 10 possible classes using the equation $a_2 = softmax(w_2 a_1)$

This architecture was chosen as it is the smallest possible network with one hidden layer. This makes it simpler to create by hand. This network is shown in Figure 1. The SoftMax function is used in the output layer to ensure that the sum of all output activations is one, and therefore can be treated as a probability distribution.

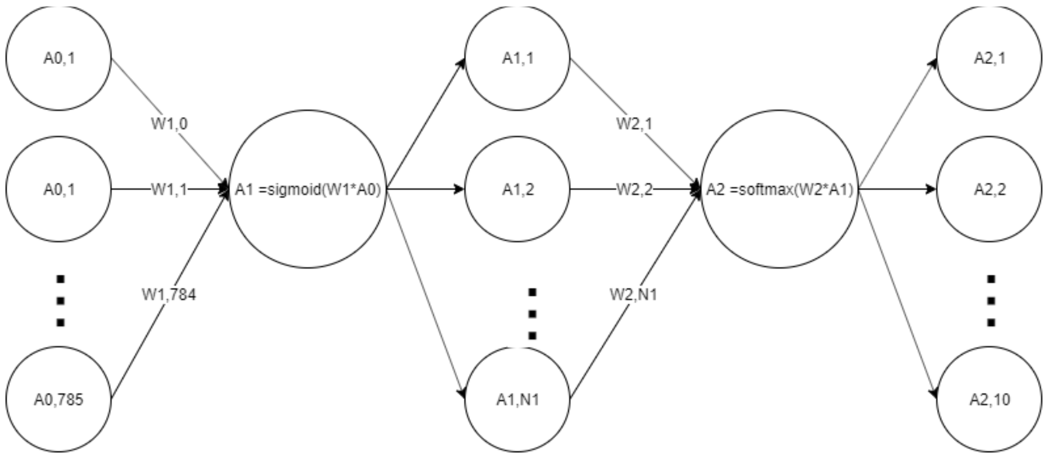

Figure 1: Part 1 Network Diagram

3.2 Part 2

For part 2 a deeper neural network is implemented consisting of the following layers

- Input layer – takes the 1x784 input pixels as activations
- Hidden layer – fully connected, computes activations using the ReLU function
- Hidden layer – fully connected, computes activations using the ReLU function
- Hidden layer – fully connected, computes activations using the ReLU function
- Output layer – fully connected, computes activations using the SoftMax function to produce 10 outputs corresponding to the 10 possible classes

This architecture was selected as it provides the network a great deal of flexibility while maintaining a relatively simple structure. The ReLU activation function is used in hidden layers to avoid gradient vanishing. SoftMax was used for the output layer so that the sum of all output activations is one, and therefore can be treated as a probability distribution.

3.3 Part 3

In part 3 a simple convolutional neural network is implemented with the following layers

- Input layer – takes 28x28x1 input pixels as activations
- Convolutional layer – 32 3x3 filters with ReLU activations
- Convolutional layer – 32 3x3 filters with ReLU activations
- Maxpool layer – 2x2 filter
- Dropout layer – rate = 0.25
- Fully connected – ReLU activation with 392 nodes
- Fully connected – SoftMax activation with 392 nodes

This network architecture was chosen as it is a simple CNN structure with robust performance. The two convolutional layers on the input reduce the dimensionality of the input data. Their filter sizes were selected to be 3x3 as large filter sizes would likely not work well on our small width images. The maxpool and dropout layers are added to further reduce the dimensionality and prevent the network from over-fitting the training data. Lastly the fully connected ReLU and SoftMax layers compute additional features and a final output which can be interpreted as a probability distribution.

4 Results

4.1 Part 1

Running our simple neural network, we find that following relationship between training accuracy and number of epochs

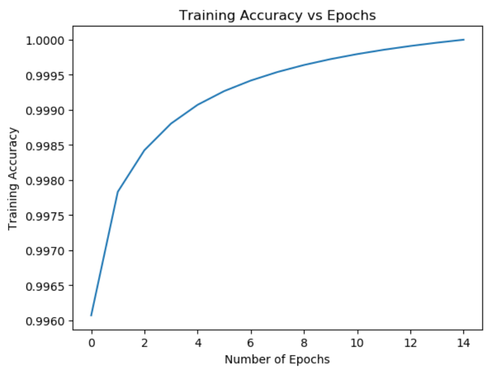

Figure 2: Part 1 Results

This training was performed with a learning rate of 0.03 and produced an accuracy of 82%

4.2 Part 2

Below we can see the results of running our CNN with a variety of different learning rates ranging from 0.001 to 0.1

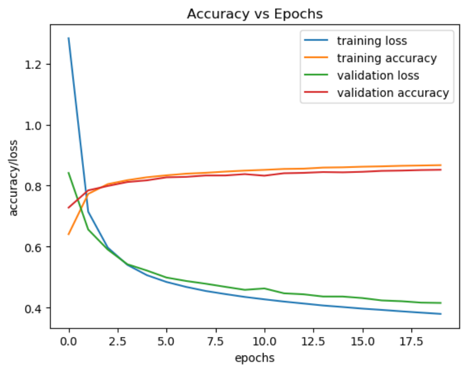

Figure 3: LR=0.001

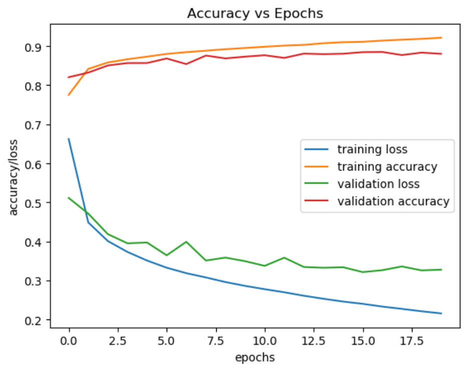

Figure 5: LR=0.01

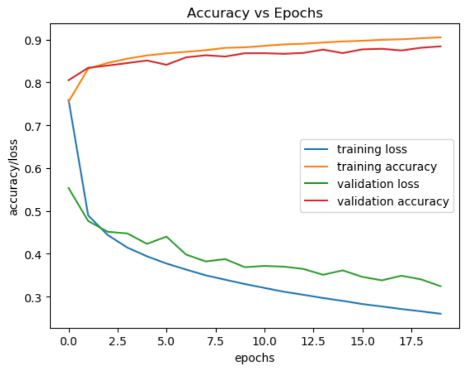

Figure 4: LR=0.005

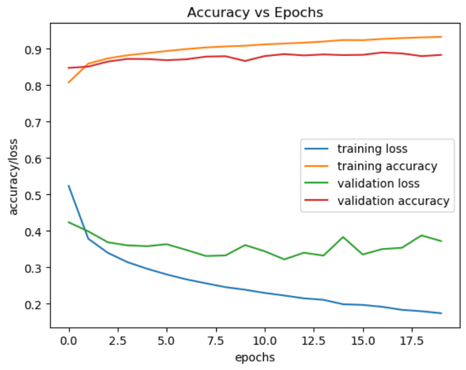

Figure 6: LR=0.1

From these plots we can see that for large learning rates the model over-fits the training data. This is evident because of the growing difference between the training loss and the validation loss. When these are large the model has overfit. Judging by these plots the learning rate that best balances accuracy with overfitting is LR=0.005 which has the following results

	precision	recall
top	0.84	0.82
trouser	0.99	0.97
pullover	0.78	0.81
dress	0.88	0.9
coat	0.8	0.83
sandal	0.96	0.96
shirt	0.72	0.68
sneaker	0.93	0.95
bag	0.97	0.97
ankle boot	0.96	0.95

The model had an overall validation accuracy of 88%

4.3 Part 3

Below we can see the results of running our CNN with a variety of different learning rates ranging from 0.001 to 0.5

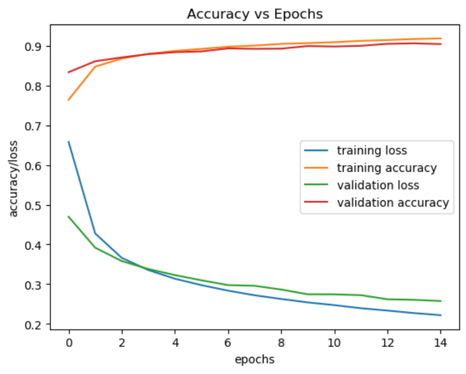

Figure 7: LR=0.001

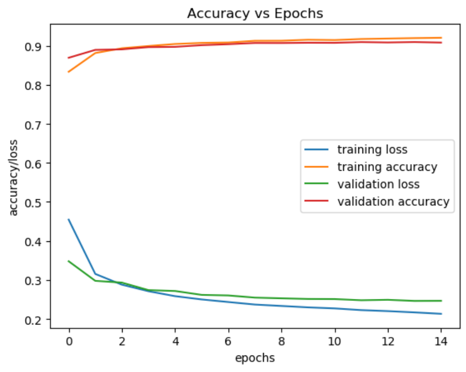

Figure 10: LR=0.1

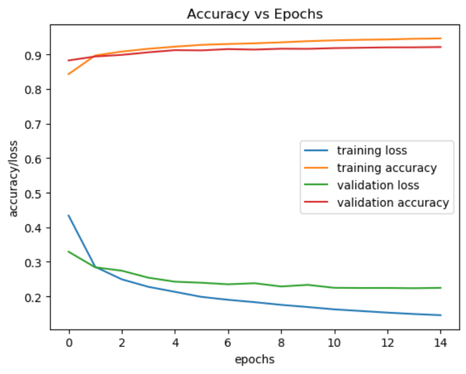

Figure 8: LR=0.01

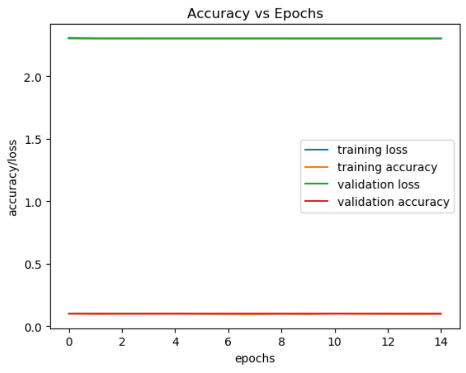

Figure 11: LR=0.5

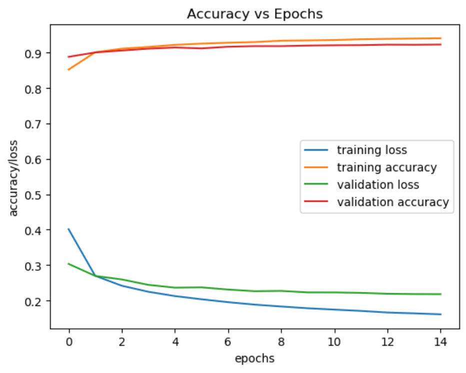

Figure 9: LR=0.05

In these images we can see that for most learning rates the model converges well within 15 epochs, but we can determine which is optimal. In the case of LR=0.5 we can see that the model fails to converge entirely. Outside of this we can see that for the remaining test cases there is a positive correlation between the learning rate and the accuracy of the model. From these plots we observe that the case of LR=0.1 has the best performance as it converged quickly and has a minimal difference between training and testing loss indicating that the model has not significantly overfit. For this case we achieved the following results

	Precision	Recall
top	0.87	0.87
trouser	0.99	0.98

pullover	0.9	0.87
dress	0.93	0.93
coat	0.86	0.9
sandal	0.9	0.98
shirt	0.78	0.77
sneaker	0.99	0.98
bag	0.99	0.98
ankle boot	0.97	0.97

The overall accuracy was 92%

In the following figures we change the number of nodes in the fully connected layer of this model. Comparing these to the earlier figure we see that it has no substantial effect and both configurations also result in a final validation accuracy of 92%

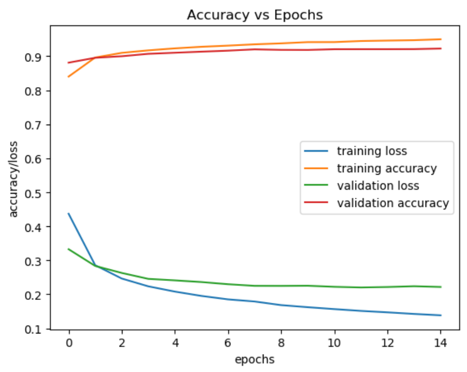

Figure 12: LR=0.01, N=600

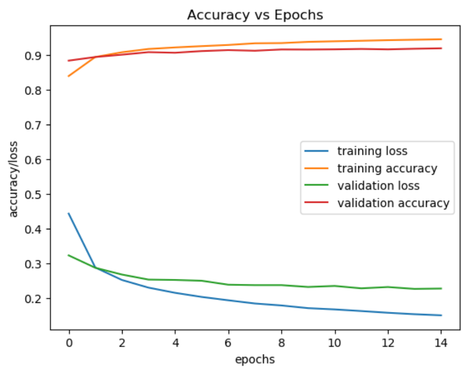

Figure 13: Part 3 LR-0.01, N=200

5 Conclusion

In this project we explored the popular MNIST Fashion dataset comprised of images of common pieces of clothing. Then we explored the necessary preprocessing that must be performed on this data to prepare it for use in neural network training. With our preprocessed data we presented three models for 10 class neural network classifiers compatible with out dataset. From test we saw that the deeper neural network of part 2 has improved performance than the minimal three-layer network of part 1. We then developed a convolutional neural network to perform the same classification task and found that in this case the CNN has improved performance over its simple neural network counterparts.