Lab 3: Neural Networks

Daniel Sherman, *daniel.sherman@ryerson.ca, (501034155)*

I. Implementation Steps

The `tensorflow` (version 2.2.0) and `keras` libraries were implemented to design and implement neural networks. The Fashion MNIST dataset was used for classification. Some examples of images with labels in the dataset can be seen in Figure 1. The entire dataset was used, so the training set contained 60,000 samples and the testing set contained 10,000 samples in 10 different classes. Each network was trained and tested in separate datasets.

The 28 by 28 pixel images were simply normalized by the maximum grey level, and stores as 32-bit floating point values.

Fig. 1: Training images with labels from the Fashion MNIST dataset

Several different architectures of neural networks were investigated on the basis of accuracy. A 128-neuron network with 1 fully connected hidden layer was treated as the baseline case. A 100-neuron network with 5 fully connected hidden layers (20 neurons per layer) and 2 fully connected layers (50 neurons per layer) was investigated. A 1000-neuron network with 5 fully connected hidden layers (200 neurons per layer) and 10 fully connected hidden layers (100 neurons per layer) were investigated. A 5000-neuron network with 5 fully connected hidden layers (1000 per layer) and 10 fully connected layers (500 neurons per layer) were also investigated. Finally, a convolutional neural network architecture (from here) was also used. The specific architecture is seen in section A.

Each network was compiled using the '`adam`' and '`sgd`' compilers included in the `keras` library for comparison. Each network was trained for 10 epochs.

II. Discussion

The accuracy for specific compilers for each network used can be seen in Table I.

TABLE I: Accuracy scores for each network with different compilers

		Test Accuracy	
Total Neurons	**#Layers**	**adam**	**sgd**
128	1	0.8049	0.8138
100	5	0.6041	0.615
100	2	0.8068	0.8109
1000	5	0.3733	0.3314
1000	10	0.1	0.1
5000	5	0.1	0.1
5000	10	0.1	0.1

For the convolutional neural network, when using the '`adam`' compiler, the CNN had an accuracy of 0.9075, and 0.8233 with the '`sgd`' compiler.

For all of the networks, the '`sgd`' compiler performed better than the '`adam`' compiler, except for the case of the 1000-neuron network with 5 hidden layers.

It is more beneficial to utilize a network with less layers in every case (seen in Table I). With the larger networks, the test accuracy drops drastically while the training loss remains low, indicating overfitting to the training data. When using the really large networks (5000 neurons), the number of layers does not matter, as the degree of overfitting is so large.

The best model by far was the convolutional neural net with the '`adam`' compiler, for a number of reasons. After flattening, it is a relatively small network, only having 1 hidden layer of 256 neurons. Furthermore, the network performs 2 rounds of convolution and max pooling that helps the network determine what features are of significance for the images in the dataset.

Furthermore, utilizing a dropout layer also helps the network to prevent overfitting. Randomly dropping neurons and connections helps to add artificial noise to the network so that it learns the underlying pattern to the data and not the quirks of the dataset.

APPENDIX A
CNN ARCHITECTURE

TABLE II: CNN Architecture used

Model: "sequential"

Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 28, 28, 64)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	8224
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
dropout_1 (Dropout)	(None, 7, 7, 32)	0
flatten (Flatten)	(None, 1568)	0
dense (Dense)	(None, 256)	401664
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570