Neural Networks as Classifiers for Fashion Items

Ofek Inbar

Abstract

We explored the Fashion-MNIST dataset and attempted to classify it using various Neural Network machine learning models.

1 Introduction and Overview

Part I of this study involved experimenting with different hyperparameters for a fully-connected Neural Network. After dividing the training data into a "training" set and a "validation" set, we experimented with different values for the parameters until we found a model we were happy with. Part II of this study was exactly the same, only the fully-connected Neural Network was swapped out for a Convolutional Neural Network.

2 Theoretical Background

Neural Networks are a particular type of data structure used in machine learning for a large number of applications. They have been frequently used for classifying inputs into one of a number of discrete categories, such as digit recognition in an image.

A Neural Network can be modeled as a directed graph that's divided into "layers". Edges in a neural network only connect nodes in one layer with nodes in the next. Each edge has a (real-number) weight and each node has a (real-number) "bias". The first layer of the network is known as the "input layer", because there is a node for each value in a single input (for a 28-by-28 pixel grayscale image, there are $28 \times 28 = 784$ inputs with values between 0 and 255). The value of each node in the next layer is computed as a linear combination of the edge weights from the first to the second layer and the input value for the corresponding node in the first layer, with the second layer's node's bias being added in at the end. Then, this value is passed through some non-linear function to map it to a value between 0 and 1. For example, to calculate the value of a single node p in the second layer was define $\dot{v}$ as a vector containing the input values to the neural network, $\dot{w_p}$ is a vector containing the weight of each edge between p and some node in the input layer, and b_p is the bias for node p. Then, the value of p is derived as $f(\dot{v} \cdot \dot{w} + b)$ where f is some non-linear function of the kind we mentioned before (like tanh). This value is computed for every node from the input layer all the way to the final layer, which is the output of the network (so there are as many nodes in this layer as there are categories for the classifier, for instance). The process of setting computing the optimal weights and biases to maximize the accuracy of the network in classifying the input data is called "training" the network, and there are several algorithms that accomplish this task.

Convolutional Neural Networks are simply Neural Networks that contain convolutional layers (often in addition to fully-connected layers). A convolutional layer attempts to learn the weights for k filters, each of which are $m \times m$ matrices. These filters are applied across each $m \times m$ "slice" of the previous layer.

3 Algorithm Implementation and Development

The first step was to divide the training data into a "training" set and a "validation" set. We selected the first 5000 points to include in the validation set and the rest became the training set. We then used `tensorflow` to build a fully-connected Neural Network and train it on the training set (with the validation set giving us a measure of whether we are "over-fitting" the model to our training data). Finally, we evaluate the model by running it on the test data and comparing it to the labels for that data.

In Part I we experimented with the number of layers of our network, the width/activation function/regularization factor of each of those layers, the optimizer (and its corresponding parameters), and the number of epochs. We tried networks with 6 hidden layers (the first two with 500 nodes, the next two with 300, and the last two with 100), 3 hidden layers (500, 300, 100 and 300, 300, 300), and 1 hidden layer (300 nodes). We experimented with Rectilinear Unit, Sigmoid, tanh, and Exponential activation functions. We experimented with regularization factors of 10^{-2} through 10^{-6}. We tried the Adam, Adadelta, Adagrad, Nadam, and Stochasting Gradient Descent optimizers, with learning rates of 10^{-2} through 10^{-4} for Adam and Nadam. Finally, we tried to vary the number of epochs between 5 and 20.

In Part II we experimented with the activation functions of the convolutional and full-connected layers, the number of epochs, the regularization factor for the fully-connected layers, and the number of convolutional layers.

4 Computational Results

The best we managed to do in part 1 was with a network two hidden layers (300 and 100 nodes respectively) with Rectilinear activations and a 10^{-6} regularization factor, the Nadam optimizer with learning rate 10^{-3}, and 10 epochs. The loss and accuracy of this model can be seen in Figure 1. The final values of accuracy, loss, validation accuracy, and validation loss were 0.9167, 0.2221, 0.8970, and 0.3152 respectively. We achieved an accuracy of 0.8867 and a loss

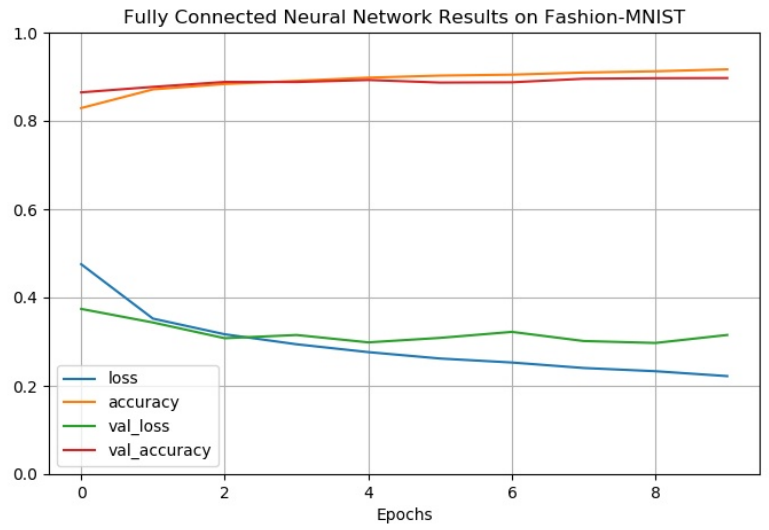

Figure 1: Loss and accuracy of fully-connected Neural Network

of 0.3485 on the test data. The confusion matrix for the training data is:

	Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot
Top	5006	0	93	97	10	0	330	0	7	0
Trouser	6	5354	3	71	8	0	2	0	0	0
Pullover	31	0	4663	55	461	0	280	0	6	0
Dress	59	1	14	5278	106	0	35	0	6	0
Coat	6	0	287	162	4835	0	220	0	2	0
Sandal	0	0	0	2	0	5401	0	85	1	18
Shirt	602	1	296	123	307	0	4161	0	17	0
Sneaker	0	0	0	0	0	6	0	5324	1	157
Bag	9	0	17	13	18	0	18	6	5427	2
Boot	0	1	0	0	0	18	0	84	0	5391

The confusion matrix for the test data is:

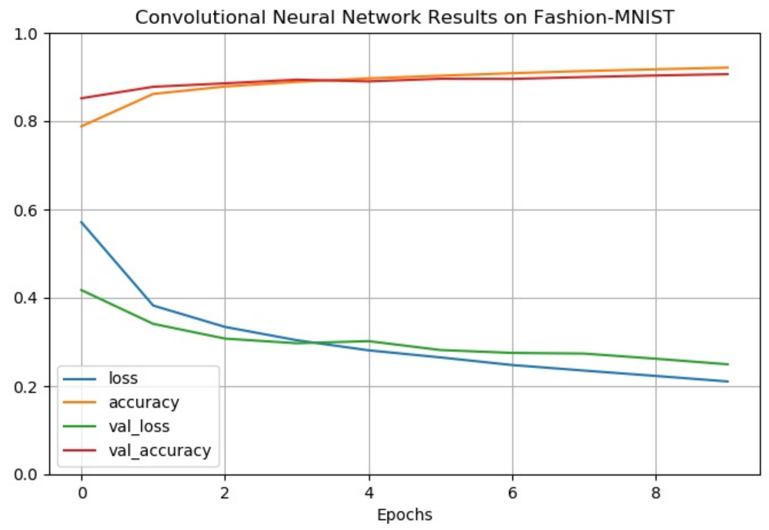

Figure 2: Loss and accuracy of Convolutional Neural Network

	Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot
Top	860	1	23	28	5	0	74	0	9	0
Trouser	6	962	1	24	5	0	1	0	1	0
Pullover	18	0	804	13	98	0	67	0	0	0
Dress	15	2	8	927	24	0	20	0	4	0
Coat	1	0	81	42	815	0	60	0	1	0
Sandal	0	0	0	2	0	952	0	31	2	13
Shirt	130	0	81	42	77	0	660	0	10	0
Sneaker	0	0	0	0	0	9	0	963	0	28
Bag	9	0	2	5	7	0	1	7	969	0
Boot	0	0	0	1	0	8	1	35	0	955

In part 2 we were very quickly able to get an accuracy above 90% for the validation data. The model was identical to LeNet-5, except every layer's activation function was set to Rectilinear Unit, the regularization factor for the two fully-connected layers was 10^{-6}, and the number of epochs was set to 10. The loss and accuracy of this model can be seen in Figure 2. The final values of accuracy, loss, validation accuracy, and validation loss were 0.9212, 0.2105, 0.9064, and 0.2494 respectively. We achieved an accuracy of 0.8958 and a loss of 0.2837 on the test data. The confusion matrix for the training data is:

	Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot
Top	4762	0	38	84	14	1	636	0	8	0
Trouser	4	5382	0	43	5	0	9	0	1	0
Pullover	44	1	4524	29	480	0	417	0	1	0
Dress	81	30	14	5135	135	0	103	0	1	0
Coat	5	3	135	114	4983	0	270	0	2	0
Sandal	0	0	0	1	0	5429	0	43	0	34
Shirt	358	3	163	73	292	0	4609	1	7	1
Sneaker	0	0	0	0	0	38	0	5182	0	268
Bag	4	0	2	7	12	4	25	4	5452	0
Boot	0	0	0	0	0	8	0	68	0	5418

The confusion matrix for the test data is:

	Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot
Top	813	1	15	18	4	0	139	0	10	0
Trouser	3	976	0	12	4	0	3	0	2	0
Pullover	12	0	793	7	95	0	90	0	3	0
Dress	15	8	9	894	31	0	40	0	3	0
Coat	0	0	38	28	874	0	57	0	3	0
Sandal	0	0	0	0	0	968	1	18	0	13
Shirt	90	0	44	25	84	0	751	0	6	0
Sneaker	0	0	0	0	0	13	0	942	0	45
Bag	4	0	3	4	2	2	8	3	974	0
Boot	0	0	0	0	0	5	0	21	1	973

5 Summary and Conclusions

Neural Networks are an incredibly versatile tool for solving a wide variety of optimization problems, and they are especially practical for image classification. Simple, fully-connected neural networks were able to get up to 89% accuracy in classifying fashion items, while convolutional networks were quickly able to exceed that value (getting 90% and up). The confusion matrices help understand how many misclassifications of each type occurred, and analyzing those it is easy to understand why the network had a difficult time differentiating between very similar items (such as Tops/Shirts and Pullovers/Coats). The Convolutional network was able to more accurately differentiate these items on the validation and test data sets.

6 Appendix B

The Python code for classifying the data and producing the figures is below:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
from functools import partial

mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()


X_valid = X_train_full[:5000] / 255
X_train = X_train_full[5000:] / 255
X_test = X_test / 255

y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]


def fully_connected():
    dense_layer = partial(tf.keras.layers.Dense, activation="relu",
                          kernel_regularizer=tf.keras.regularizers.l2(1e-6))

    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=[28, 28]),
        dense_layer(300),
        dense_layer(100),
        dense_layer(10, activation="softmax")
    ])

    model.compile(loss="sparse_categorical_crossentropy",
                  optimizer="nadam", metrics=["accuracy"])

    return (model.fit(X_train, y_train, epochs=10,
                      validation_data=(X_valid, y_valid)),
            model, 'Fully Connected Neural Network Results on Fashion-MNIST')


def convolutional():
    global X_valid, X_train, X_test

    X_train = X_train[..., np.newaxis]
    X_valid = X_valid[..., np.newaxis]
    X_test = X_test[..., np.newaxis]

    dense_layer = partial(tf.keras.layers.Dense, activation="relu",
                          kernel_regularizer=tf.keras.regularizers.l2(1e-6))
    conv_layer = partial(tf.keras.layers.Conv2D,
                         activation="relu", padding="valid")

    model = tf.keras.models.Sequential([
        conv_layer(6, 5, padding="same", input_shape=[28, 28, 1]),
        tf.keras.layers.AveragePooling2D(2),
        conv_layer(16, 5),
        tf.keras.layers.AveragePooling2D(2),
        conv_layer(120, 5),
        tf.keras.layers.Flatten(),
        dense_layer(84),
        dense_layer(10, activation="softmax")
    ])

    model.compile(loss="sparse_categorical_crossentropy",
                  optimizer="nadam",
                  metrics=["accuracy"])

    return (model.fit(X_train, y_train, epochs=10,
                      validation_data=(X_valid, y_valid)),
            model, 'Convolutional Neural Network Results on Fashion-MNIST')


get_model_history = fully_connected # can swap for convolutional
history, model, title = get_model_history()

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.title(title)
plt.xlabel('Epochs')
plt.show()

y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)

model.evaluate(X_test, y_test)
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
```