Neural Networks for Classifying Fashion MNIST

AMATH 482
Lahari Gorantla

Abstract

Neural networks are computational algorithms that provide the groundwork for machine learning. They are modeled after the processing network of neurons in the brain. After training the algorithm with a training and validating set of input and output data, neural networks can analyze a complete new set of test data into outputs. For this lab, we built a fully-connected neural network and a convolutional neural network with Tensorflow in Python. We explore the accuracies of both networks in analyzing an MNIST Fashion Data Set.

I. Introduction and Overview

Machine learning can utilize neural network algorithms, built from a training data set, to predict classifications of a new test set. These neural networks consist of multiple densely interconnected nodes organized into layers that "feed-forward" from input layer through the hidden layers to the output layer. Two of the neural networks that will be built in this lab report are the fully-connected neural network and the convolutional neural network. The dataset used to explore the built networks is the Fashion-MNIST dataset.

The Fashion-MNIST is a dataset from a fashion and style company called Zalando. This dataset is a part of the Tensorflow and Keras Python package. There are 60,000 images to train the neural networks with and 10,000 images to test the network with. There are 10 classes of clothing: T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. Each of the article of clothing is given a number label from 0 to 9, respectively. The goal of this lab report is to explore different hyper-parameters in the network architecture and to maximize the accuracy for each network while minimizing overfitting the data.

II. Theoretical Background

Fully connected neural network
A fully connected neural network has fully interconnected layers, each input node is connected to each hidden node and the trend continues until the output. Each connection has a certain learnable weight parameter, w_i, associated with it. The neural network could be described as such:

$$x_2 = \sigma(A_1 x_1 + b_1) \qquad (1)$$
$$y = \sigma(A_2 x_2 + b_2) \qquad (2)$$

Where x_1 denotes the input layer, x_2 denotes the hidden layer, and y denotes the output layer. The matrices A_1 and A_2 contain all the weight parameters, and are multiplied by x_1 via the dot product rule. The b_1 and b_2 vectors hold the bias values between the interconnected layers and can be used to shift around the threshold. The σ denotes step activation function that all the vectors are inputted into. If the weighted sum is greater than the threshold, then the nonlinearity σ function expresses the neuron as an active neuron. If the weighted sum is lesser than the threshold, the nonlinearity σ function expresses the neuron as an inactive neuron.

Convolutional Neural Network (CNN)
Similarly, to the fully-connected neural network, both networks utilize learnable weights and biases. After receiving an input, a dot product is taken, and a non-linearity function is applied to it. The difference is the input layer is taken as a 3-dimensional system where the third dimension is a color input. 1 represents a greyscale image while 3 represents an RBG image. Then a convolution layer (also called a filter) of smaller dimensions than the input image performs a convolution operation on the input image. The filter is composed of weights of its own. The sum of the products of this operation becomes the new output layer that is inputted into the consecutive hidden layer. The general layout of a CNN is input layer, convolution layer, activation layer, a pooling layer, etc. until the final output is a fully-connected layer. The pooling layer inserted between convolution layers is intended to reduce the spatial size of the representation to decrease the number of parameters and computation.

Activation Function
An activation function can determine whether a neuron is being fired or not. Activation functions can introduce non-linear complexities to the linear regression input. To increase the power of learning high-dimensional complex data and properly represent the non-linear mappings between inputs and outputs, an activation function is important. Several types of activation functions that exist include sigmoidal, tanh, and rectified linear units (ReLu). The activation function used to build the neural networks in this lab report is the ReLu. It is mathematically represented as R(x) = max(0, x). Basically, if x is less than 0 then R(x) = 0 and if x is equal to or greater than 0, then R(x) = 0. It is the most commonly used activation function in neural network building. A limitation is that it can only be used for hidden layers. The output layer requires a different activation such as softmax, as seen in equation 3. Softmax takes a vector of numbers and turns them into a probability distribution that sum up to 1 to represent a list of possible outcomes by taking exponents of each output and then normalizing each number by sum of the exponents.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}} \qquad (3)$$

Backpropagation
To find the optimal set of weights based on the error rate from the previous epoch (aka iteration), backpropagation is used. It trains the neural network by reducing error rates and calculates the gradient of the loss function with respect to the weights. Neural networks are trained using the stochastic gradient descent and through a loss function, such as cross-entropy or mean squared error. We want to find a partial derivative of the loss function with respect to each of the weights that is equal to 0. However, that is difficult so a stochastic gradient descent is utilized to find the most minimized error. The learning rate denotes the rate moving down the gradient to minimize the loss function. It is important to find a learning rate that is small enough to not overstep the global minimum and big enough to limit computational time. The goal of the learning curve is to decrease loss for every iteration. Cross entropy works well for a multinomial distribution given a logistic output that heavily penalizes were you are predicting the wrong output. It is used for classification problems while MSE is used for regression problems.

III. Algorithm Implementation and Development

Processing the Dataset
In order to build neural networks we imported numpy, tensorflow, matplotlib.pyplot, pandas, and the confusion_matrix from sklearn.metrics. Then the Fashion MNIST dataset from Tensorflow is imported. 60,000 images are used as training data while 10,000 images are used as testing data; each image has a dimension of 28 by 28 pixels. 5,000 images from the training data set are relabeled to be utilized as the validation testing data set. The images are divided by 255.0 to convert the images from uint8 to float values and to normalize it.

Fully Connected Neural Networks
To make a feed-forward neural network with organized layers, *Sequential* from tf.keras.models is used. The first input layer *Flatten*s the image from a 28 by 28 image into a 1 by 784 vector. Within the *Sequential* input, the *Dense* function is used to create 3 hidden layers with 300 nodes, 150 nodes, and 50 nodes. The hidden layers used the activation function "relu" while using a kernel regularizer of 0.00001. Regularizers decreases the weight of the loss function and is applied to the output of the activation function. The output layer has 10 nodes to output the 10 classes with an activation of "softmax". Then the model is compiled with the loss function "sparse categorical crossentropy" and an *Adam* optimizer with a learning rate of 0.0001 and metrics set to "accuracy". The *Adam* optimizer, which stands for Adaptive Moment Estimation, uses adaptive learning rates for each parameter. Once the model and the backpropagation specific are built, the training model can be fitted and validated with the 5,000 image validation data set. The epoch hyper-parameter was specified as 20 to consider the tradeoff between accuracy and computational time.

Convolutional Neural Networks
After processing the dataset, there is another step for the CNN. An extra dimension needs to be added to the 2D image. To make a feed-forward neural network with organized layers, *Sequential* from tf.keras.models is used. Then, there is an alteration between convolution and pooling layers. Unless specified both the convolution and dense layers have an activation of "relu" and a kernel initializer of "he_uniform". The dense layer (*Dense*) has a kernel regularizer of 0.0001 and the convolution layers have a padding set to "valid". The "valid" is intended to shape output to its true size without padding of 0's. The first convolution layer (*Conv2D*) is made with a filter set to 32, filter size set to 5 by 5, padding set to "same", and an input shape of 28 by 28 by 1. The 1 denotes the grey scale attribute of the images. The "same" is intended to keep the size of the feature maps the same as the size of the input; it works by filling it in with 0's. Then this layer is followed by a layer that makes an *AveragePooling2D* of pool size 2 by 2. It will downscale both the vertical and horizontal spatial dimensions by half. The stride is also 2 by 2. The second convolution layer has a filter set 64, and filter size set to 5 by 5, followed by an *AveragePooling2D* layer with a pool size of 2 by 2. Then the third convolution layer has a filter set to 128 with a filter size set to 5 by 5. Then this layer is *Flatten*ed before using a dense layer of 128 and then outputting to a dense layer with 10 outputs and a "softmax" activation function. Then, the model is compiled with the loss function "sparse categorical crossentropy" and an *Adam* optimizer with a learning rate of 0.001 and metrics set to "accuracy". Once the model and the backpropagation specific are built, the training model can be fitted and validated with the

5,000 image validation data set. The epoch hyper-parameter was specified as 10 to consider the tradeoff between accuracy and computational time.

Obtaining Results

The results considered for both models include the training accuracy, the validation accuracy, testing accuracy, the confusion matrix of the tested dataset, and the training history. The confusion matrix displays how each of the test images were classified by the algorithm versus their true classifications.

IV. Computational Results

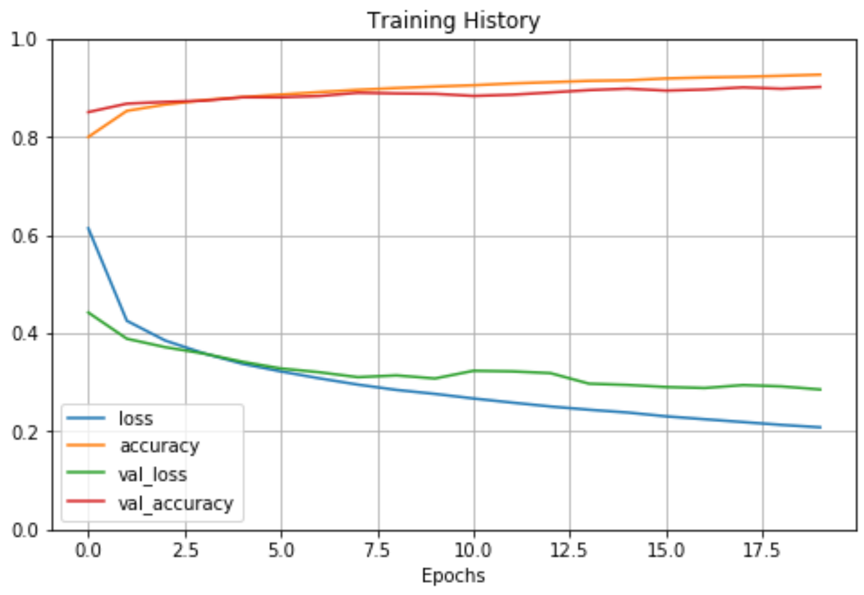


Figure 01. The learning curve plot for a fully-connected neural network. It displays the accuracies and losses of the dataset as well as the validation loss and accuracy.

```
[[865   0  22  21   6   0  78   2   6   0]
 [  4 972   1  17   2   0   3   0   1   0]
 [ 15   1 823  12  91   0  56   1   1   0]
 [ 22  10  19 879  40   0  26   0   4   0]
 [  1   1  77  20 850   0  50   0   1   0]
 [  0   0   0   1   0 962   0  27   0  10]
 [124   3  75  25  70   0 692   0  11   0]
 [  0   0   0   0   0  10   0 972   0  18]
 [  6   0   6   5   5   1   2   8 967   0]
 [  0   0   0   0   0   8   1  51   0 940]]
```

Figure 02. The confusion matrix for a fully-connected neural network. The row represents actual y output while the column represents the predicted output. Each are in the order of T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

For the fully-connected neural network, the training dataset accuracy was 92.74%, validation dataset accuracy was 90.22%, and the testing dataset accuracy was 89.22%. The algorithm overfit the data, which is why the validation accuracy is about 3% lower and the testing dataset accuracy was about 4% lower than the training accuracy. Even when the hyper-parameters were manipulated, it was difficult to decrease the overfitting while keeping the validation accuracy above a 90%. Increasing the validation of the accuracy felt very impossible to go above a 95% felt impossible to do.

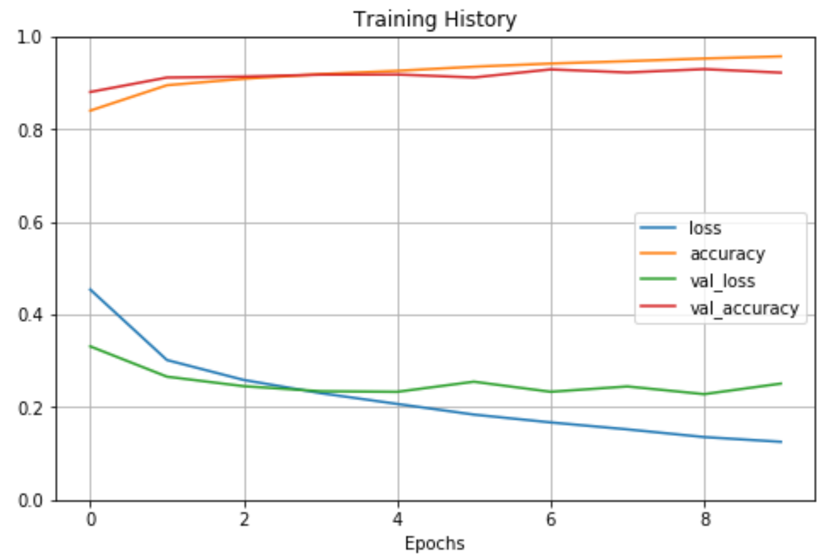
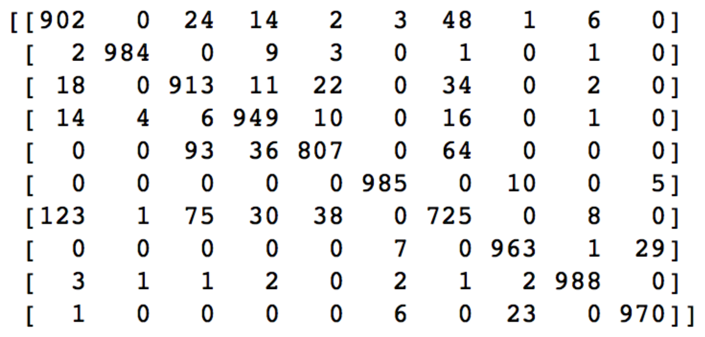


Figure 03. The learning curve plot for a convolutional neural network. It displays the accuracies and losses of the dataset as well as the validation loss and accuracy.



```
[[902   0  24  14   2   3  48   1   6   0]
 [  2 984   0   9   3   0   1   0   1   0]
 [ 18   0 913  11  22   0  34   0   2   0]
 [ 14   4   6 949  10   0  16   0   1   0]
 [  0   0  93  36 807   0  64   0   0   0]
 [  0   0   0   0   0 985   0  10   0   5]
 [123   1  75  30  38   0 725   0   8   0]
 [  0   0   0   0   0   7   0 963   1  29]
 [  3   1   1   2   0   2   1   2 988   0]
 [  1   0   0   0   0   6   0  23   0 970]]
```

Figure 04. The confusion matrix for a convolutional neural network. The row represents actual y output while the column represents the predicted output. Each are in the order of T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

For the convolutional neural network, the training dataset accuracy was 95.75%, validation dataset accuracy was 92.24%, and the testing dataset accuracy was 91.86%. Again, there was some overfit as seen by the 3% difference between training accuracy and validation accuracy. Although, it should be noted that for the convolutional neural network, the accuracy for all values is greater! It still felt difficult to get the validation accuracy above the 95% threshold.

When the confusion matrices in figures 03 and 04 are compared, it can be seen along the diagonal that more of the articles of clothing in the convolutional neural network are correctly classified compared to those in the fully-connected neural network. In addition, the validation loss and training loss function are smaller for the convolutional neural network.

V. Summary and Conclusion

As seen in the computational results, the convolutional neural network produced higher accuracy as it classified more articles of clothing correctly. It did it only in 10 epochs compared to the 20 epochs used for the fully-connected neural network. Ultimately, both neural networks did a good job classifying the Fashion MNIST dataset but they couldn't cross the 95% validation accuracy threshold. The convolutional neural network is better compared to the fully-connected neural network because of the more complex architecture set by convolving and pooling.

Appendix A. Python functions used and brief implementation explanation

Flatten – flattens the 2D input into a 1 by n vector
Dense – creates a densely connected neural network
Conv2D – creates a 2D spatial convolution; used in convolutional neural network
Fit – fits the x and y training data to build an algorithm that iterates over a given number of epochs and validates with another validation data set
Sequential – adds a list of layers to the model
Compile – configures the model for training with defined loss, optimizer, and metrics
predict_classes – takes in the x-testing dataset and outputs a predicted y
confusion_matrix – creates a matrix with the real y and predicted y
AveragePooling2D – average pooling operation for spatial data; reduces the size of input by specified amount

Appendix B. Python Code

Fully-Connected Neural Network

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid = X_train_full[:5000]/255.0
X_train = X_train_full[5000:]/255.0
X_test = X_test/255.0

y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]
from functools import partial
my_dense_layer = partial(tf.keras.layers.Dense, activation="relu",
kernel_regularizer=tf.keras.regularizers.l2(0.00001))

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=[28, 28]),
  my_dense_layer(300),
  my_dense_layer(150),
  my_dense_layer(50),
    my_dense_layer(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy", #what is the probability of our prediction
             optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
             metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20, validation_data=[X_valid,y_valid])
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show
plt.title('Training History')
plt.xlabel('Epochs')
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)
```

Convolutional Neural Network

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid = X_train_full[:5000] / 255.0
X_train = X_train_full[5000:] / 255.0
X_test = X_test / 255.0
y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]
X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]
from functools import partial
my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_initializer = 'he_uniform',
kernel_regularizer=tf.keras.regularizers.l2(0.0001))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="relu", kernel_initializer = 'he_uniform',
padding="valid")
model = tf.keras.models.Sequential([
    my_conv_layer(32,5,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.AveragePooling2D(2),
    my_conv_layer(64,5),
    tf.keras.layers.AveragePooling2D(2),
    my_conv_layer(128,5),
    tf.keras.layers.Flatten(),
    my_dense_layer(128),
    my_dense_layer(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
            optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
            metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid,y_valid))
```

```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.title('Training History')
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)
```