Artificial Neural Networks for Classifying Fashion MNIST

Arman Naderi

Abstract:

Artificial neural networks have wide utility in data analysis applications because of their ability to continually and flexibly optimize network parameters. In this paper, I examine the effectiveness of two types of neural network architectures for image classification. By training, validating, and then testing a fully-connected neural network and a convolutional neural network, I am able to determine which architecture has the most utility when classifying the images of the Fashion Modified National Institute of Standards and Technology (MNIST).

Introduction and Overview:

Artificial neural networks, abbreviated as neural networks, are a set of algorithms that allow for the recognition of patterns in a dataset through the optimization of guesses based off of task-specific rules. Neural networks have particular utility in image classification algorithms because of their ability to autonomously generate identifying features for a given image or set of images. For image processing applications, the neural network trains on a set of manually labelled images, classifies those images based on an initial guess, and then optimizes the network parameters for that guess in order to achieve a more accurate result. There are a variety of parameters that can change the way the network functions as a classifier: these parameters can range from optimization of loss functions to the method by which the network analyzes data. In this paper, I examine two fundamentally different neural network architectures: fully connected or dense neural networks, and convolutional neural networks.

The images that I classify in this paper is the Fashion dataset from the Modified National Institute of Standards and Technology (MNIST). This dataset contains 60,000, 28 pixel by 28 pixel images of 10 different classes of clothing. The clothing items are labelled as follows:

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

Theoretical Background:

Fully connected neural networks is a networks in which all neurons between layers are interconnected. Each connection has a weight parameter and each layer has bias associated with it. The network is summarized in equation 1:

$$x_2 = \sigma(A_1 x_1 + b_1) \quad \text{1.a}$$

$$y = \sigma(A_2 x_2 + b_2) \quad \text{1.b}$$

where A is the weight matrix, x_1 is the input layer, x_2 is the hidden layer, b is the bias, and y is the output. Sigma denotes the activation function for which if the evaluation of the inner function is above threshold, the neuron is active, and if not, the neuron is inactive.

Convolutional neural networks act similar to fully connected neural networks with the qualification that it incorporates convolution layers, or filters, that parses through the data being examined. There can be multiple filters per network layer, and these filters can be weighted separately as well. After these convolutional layers are put through an activation layer, they are pooled to reduce the number of parameters and duration of computation. This sequence of steps repeats until the final output layer is reached.

Algorithm Implementation and Development:

Fully connected (dense) neural network:

The activation function used for each of the fully connected neurons was "relu" or the rectified linear unit, which is a curve with a slope equal to 0 for x<0 and 1 for x>0. The lambda by which all weights were multiplied by was 0.00001. The network was 4 layers deep, where in the first layer was 500 neurons wide, the second layer was 50 neurons wide, the third layer was 250 neurons wide, and the final output layer was 10 neurons wide with the "softmax" activation function. The model was compiled with the "sparse categorical crossentropy" loss function, and optimized with "Adam" at a learning rate of 0.0001, with the accuracy parameter set.

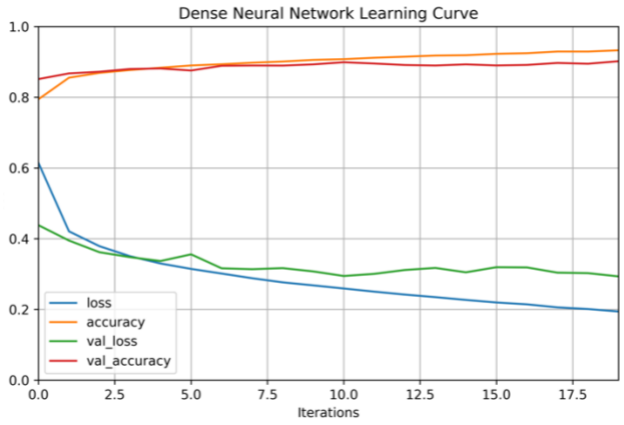
Convolutional neural network:

The activation function used for each of the fully connected neurons was "relu" or the rectified linear unit, which is a curve with a slope equal to 0 for x<0 and 1 for x>0. The lambda by which all weights were multiplied by was 0.0001. The convolution functions had the "same" padding and also used the "relu" activation function. The network was 5 layers deep, where in the first three layers were convolutional and 16 neurons wide, 32 neurons wide, and 64 neurons wide, respectively, each with a 3 x 3 filter size. Between these convolution layers was average pooling with a 2 x 2 pool size. The fourth layer was a dense layer 64 neurons layer. The final output layer was also dense and 10 neurons wide with the "softmax" activation function. The model was compiled with the "sparse categorical crossentropy" loss function, and optimized with "Adam" at a learning rate of 0.001, with the accuracy parameter set.

Computational Results:

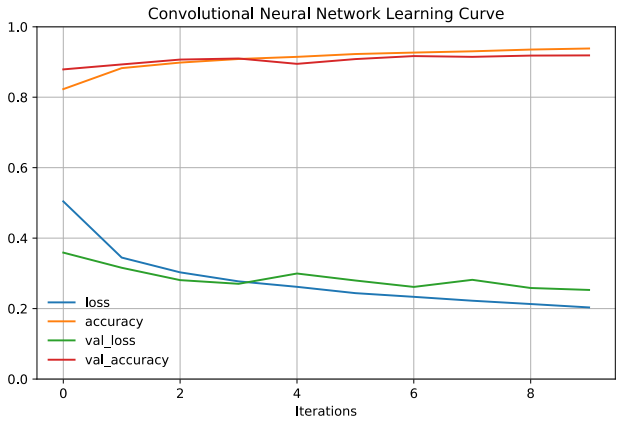
The dense neural network's accuracy was 0.8922 and had a loss of 0.3226. The learning curve and confusion matrix are shown in Figure 1. The convolutional neural network's accuracy was 0.9167 and had a loss of 0.2036. The learning curve and confusion matrix are shown in Figure 2. Both algorithms were overtrained as indicated by the fact that the training curve surpassed the validation curve relatively early in the number of iterations. It is important to note however that the convulational neural network

maintained a high accuracy while also having significantly less loss between training and validation when compared to the dense neural network. By comparing the confusion matrices between networks, both architectures struggled on the same classes of clothing items. However, as reflected in the accuracy, the convolutional neural network was slightly better when identifying differences between coats and pullovers as compared to the dense neural network. However, the dense neural network was better at identifying shirts against t-shirts and tops when compared to the convolutional neural network.



	0	1	2	3	4	5	6	7	8	9
0	850	2	7	19	5	2	111	0	4	0
1	3	961	4	21	6	0	5	0	0	0
2	11	0	787	17	100	1	84	0	0	0
3	16	3	7	915	25	0	31	0	3	0
4	1	0	68	31	838	0	60	0	2	0
5	0	0	0	1	0	972	0	19	0	8
6	108	0	62	26	72	0	727	0	5	0
7	0	0	0	0	0	15	0	970	1	14
8	6	0	3	6	4	2	12	6	961	0
9	0	0	0	0	0	8	1	50	0	941

Figure 1: Dense Neural Network Learning Curve (left) and the output confusion matrix (right) on test data after training and validation.



	0	1	2	3	4	5	6	7	8	9
0	827	0	26	17	7	1	119	0	3	0
1	3	976	1	11	1	0	8	0	0	0
2	10	1	870	3	64	0	51	0	1	0
3	6	2	19	919	34	0	19	0	1	0
4	0	1	34	17	903	0	45	0	0	0
5	0	0	0	0	0	987	0	8	0	5
6	76	1	58	20	71	0	770	0	4	0
7	0	0	0	0	0	8	0	982	0	10
8	2	1	4	1	3	3	1	1	984	0
9	0	0	0	0	0	5	0	46	0	949

Figure 2: Convolutional Neural Network Learning curve (left) and the output confusion matrix (right) on test data after training and validation.

Appendix B:
CNN:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
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] / 255.0
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_regularizer=tf.keras.regularizers.l2(0.0001))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="relu", padding="same")

model = tf.keras.models.Sequential([
   my_conv_layer(16,(3,3),padding="same",input_shape=[28,28,1]),
   tf.keras.layers.AveragePooling2D(2,2),
   my_conv_layer(32,(3,3)),
   tf.keras.layers.AveragePooling2D(2,2),
   my_conv_layer(64,(3,3)),
   tf.keras.layers.Flatten(),
   my_dense_layer(64),
   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=10, validation_data=(X_valid,y_valid))
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.gca().set_xlim(0,9)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Convolutional Neural Network Learning Curve')
plt.savefig('CNN_Learning_Curve.png')
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)
fig, ax = plt.subplots()
```

```
# hide axes
fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')

# create table and save to file
df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10), loc='center',
cellLoc='center')
fig.tight_layout()
plt.savefig('conf_mat_cnn.png', dpi=600)
```

Dense:

```
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(500),
   my_dense_layer(50),
   my_dense_layer(250),
   my_dense_layer(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
          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.gca().set_xlim(0,19)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Dense Neural Network Learning Curve')
plt.show()
plt.savefig('DNN_Learning_Curve.png')
model.evaluate(X_test, y_test)
fig, ax = plt.subplots()

# hide axes
fig.patch.set_visible(False)
```

```
ax.axis('off')
ax.axis('tight')

# create table and save to file
df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10), loc='center',
cellLoc='center')
fig.tight_layout()
plt.savefig('conf_mat_dense.png')
```