Homework 5: Neural Networks for Classifying Fashion MNIST

Yicheng Duan

AMATH 482

03/13/2020

Abstract

In this homework, we need to use Neural Networks to classify Fashion MNIST dataset. We developed two different types of Neural Networks, fully connected and convolutional, to train our data. We then compare the results of two methods.

Introduction and Overview

We acquire our whole fashion MNIST data set from TensorFlow's keras datasets. This fashion MNIST dataset contains 60,000 training images and 10,000 test images all stored in arrays once we pulled from keras.

Each of those images has their own label store in separate arrays. Labeled as:

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

Then we select first 5000 images from our 60,000 training images as our validation dataset. Then we reduce the images color information to number between 1 to 0, which suitable for our neural Networks build using TensorFlow.

TensorFlow is a famous and popular package that build Neural Networks by user demanded. We can use TensorFlow to build our fully connected and convolutional Neural Networks with different parameters.

We use TensorFlow to build our fully connected Neural Networks like:

```
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_regularizer=tf.keras.regularizers.L1L2(l1=0.0001,l2=0.
0001))


model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    my_dense_layer(320),
    my_dense_layer(130),
    my_dense_layer(10, activation="softmax")
])
```

Using dense layers to fully connect each layer.

We finally use confusion_matrix from sklearn.metrics to plot our confusion matrix to visualize our error and compare.

Theoretical Background

Neural Networks

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Fully connected Neural Networks

A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from $\mathbb{R}$ m to $\mathbb{R}$ n. Each output dimension depends on each input dimension. Pictorially, a fully connected layer is represented as follows in figure1-1.

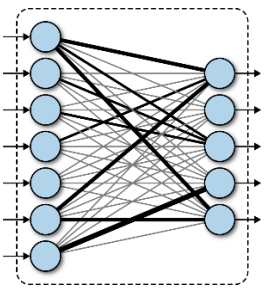
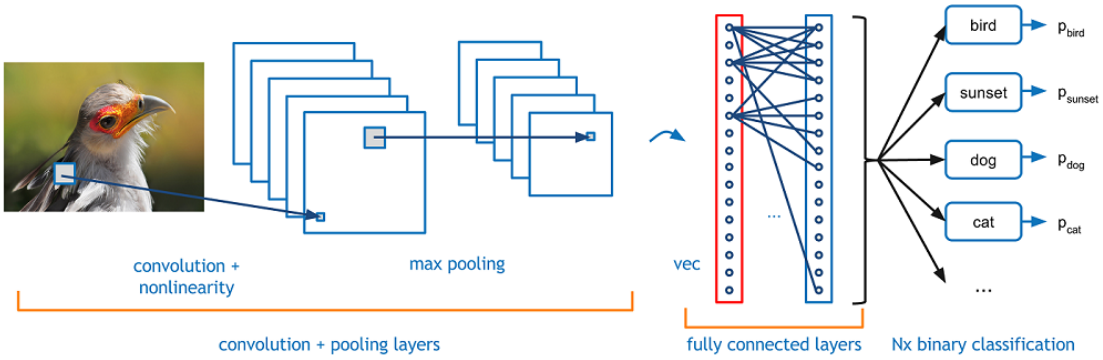


Figure 1-1

Convolutional Neural Networks



Convolutional Neural Networks are very similar to ordinary Full connected Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

Algorithm Development

We first import different packages that we need.

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

We use NumPy and Pandas to put our data into matrixes; TensorFlow to build our Neural Networks; Matplotlib to draw plots; sklearn.metrics to generate our confusion matrix, which a matrix that show the details of our test results, our error distribution.

Then, we import data into X_train_full, y_train_full, X_test and y_test arrays.

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

X_train_full is the array that contains our 60,000 images for training and y_train_full for their labels. X_test is the array that contains our 10,000 images for testing, which also have a y_test array for labeling.

Then, we select first 5000 images from X_train_full and y_train_full for validation. We section out these 5000 images from our training set to prevent cheating. By divide 255 to all our images, we have datasets contain only gray scale number from 0 to 1, ready feeding to Neural Networks:

```
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:]
```

For Convolutional Neural Networks, we build an extra axis on our image for training:

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

Now, we build our Neural Networks using TensorFlow.

For full connected Neural Networks, we first flatten the training dataset and have two hidden layer, one width 300 and another width 130. They both use an activation rule of Rectified Linear Unit with L1L2 regularizes. Finally, a 10 neural with activation "softmax" for classification. All fully connected.

```
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_regula
rizer=tf.keras.regularizers.L1L2(l1=0.0001,l2=0.0001))


model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    my_dense_layer(320),
    my_dense_layer(130),
    my_dense_layer(10, activation="softmax")
])
```

For Convolutional Neural Networks, we first have six filter 5x5 conv layers with zero padding. Then we put a MaxPooling layer after. Then, we put a 16 filter 5x5 conv layer, attached with a MaxPooling layer. Then, a 120 filters 5x5 conv layer. After that, we flatten the data and full connect an 84 neural layer. Finally, we use a 10 neural for classification.

```
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_regula
rizer=tf.keras.regularizers.l2(0.0001))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="tanh", padding="vali
d")

model = tf.keras.models.Sequential([
    my_conv_layer(6,5,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(16,5),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(120,5),
    tf.keras.layers.Flatten(),
    my_dense_layer(84),
    my_dense_layer(10, activation="softmax")
])
```

Then we compile the model and train it with our training dataset. Record the history into matrix and plot the history.

```
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=5, validation_data=(X_valid,y_vali
d))
```

Plot:

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

Then we evaluate our model using our test data set.

```
model.evaluate(X_test,y_test)
```

Generate our confusion matrix for our test data. Finally, plot a table for our confusion matrix.

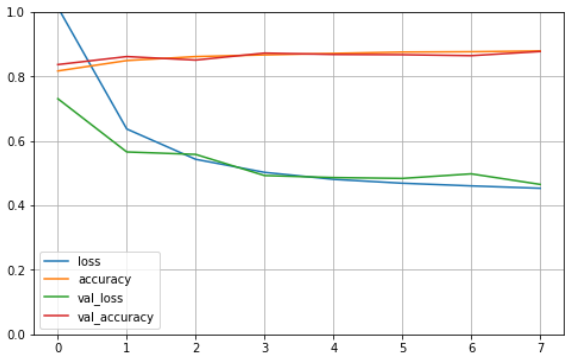
```
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
```

```
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), l
oc='center', cellLoc='center')
fig.tight_layout()
plt.savefig('conf_mat_2.png')
```

Computational Results

Fully connected Neural Network:

We got an 86.8% test evaluation accuracy on our model over 8 epochs. We can see our history of training:



History of training(full-connected)

We have little over fitting around 6th round of training. The performance of this model this average.

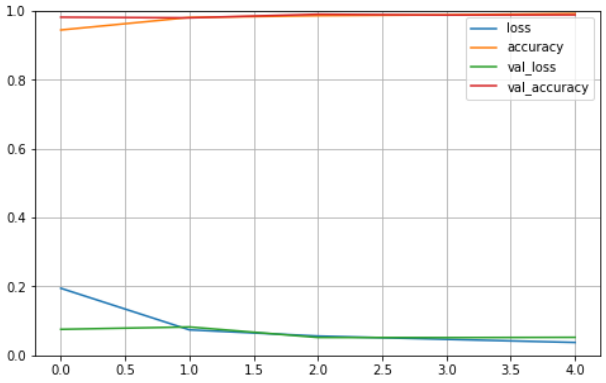
	0	1	2	3	4	5	6	7	8	9
0	836	1	15	17	6	1	112	1	10	1
1	3	970	1	18	3	0	4	0	1	0
2	14	1	767	6	151	0	59	0	2	0
3	37	12	15	849	51	0	33	0	3	0
4	0	0	86	22	860	0	31	0	1	0
5	0	0	0	1	0	928	0	37	1	33
6	132	1	89	21	147	1	594	0	15	0
7	0	0	0	0	0	10	0	950	0	40
8	3	1	4	4	13	2	4	4	965	0
9	0	0	0	0	0	5	0	33	1	961

Confusion matrix(fully-connected)

By seen our confusion matrix of test evaluation, the common error of this model this mix up between No.4 and No.2, No,4 and No.6, No.0 and No.6. which corresponding the mix up of Coat and Pullover, Coat and Shirt, T-shirt and Shirt.

Convolutional Neural Networks:

We got a 98.84% accuracy on our test evaluation over 5 epochs.



History of training (Convolutional)

From our history of training, we see almost non overfitting. Our accuracy of validation is at a high level.

	0	1	2	3	4	5	6	7	8	9
0	972	0	2	0	0	0	2	1	3	0
1	0	1121	1	0	0	0	2	7	4	0
2	1	0	1026	0	0	0	0	5	0	0
3	0	0	3	1000	0	3	0	2	2	0
4	1	0	0	0	974	0	4	2	0	1
5	2	0	0	10	0	877	2	1	0	0
6	3	1	1	0	1	2	948	0	2	0
7	1	0	3	0	0	0	0	1020	1	3
8	2	0	5	2	1	0	1	3	960	0
9	3	0	0	4	10	2	0	2	2	986

Confusion matrix (Convolutional)

Our Confusion matrix shows us almost perfect results. The error occurs mostly at mixing up with No.4 with No.9 which between Coat and Ankle boot.

Summary and Conclusions

We see a huge difference between different Neural Networks. The evolution of Neural Networks is tremendous in recent years. The correct parameter and arrangement of Neural layers are critical for the training efficiency.

Python Code: