MATH 498: Foundations of Machine Learning 2022
Project 2 Multiclass Classification

junhuuu@umich.edu
xiaziy@umich.edu

April 22, 2022

Experimental Protocol

Suppose that someone wants to reproduce your results. Briefly describe the steps used to obtain the predictions starting from the raw data set downloaded from the project website. Use the following sections to explain your methodology. Feel free to add graphs or screenshots if you think it's necessary. The report should contain a maximum of 4 pages.

1 Tools

Fashion-MNIST is a dataset of Zalando's fashion article images and it's a slightly more challenging problem than regular MNIST. Here, 60,000 images are used to train the network and 10,000 images to evaluate how accurately the network learned to classify images. The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The labels are an array of integers, ranging from 0 to 9 corresponding to the class of clothing.
We need to use neuron networking to predict the label of the pictures. In order to do deep learning, we used Keras via tensorflow package. Keras is popular and well-regarded high-level deep learning API. It's built right into to TensorFlow — in addition to being an independent open source project. The Keras API supports this by specifying the "validation_data" argument to the model.fit() function when training the model, that will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.

2 Algorithm

We used CNN model as our algorithm to predict this muti-class classification. At the begining, we did the normalization for the whole data set. After carefully test different types and different number of layers of this deep learning. We finally chose the one with three major layers. The last two major layers are both combined by three small layers which are a convolution layer, a pooling layer, and a drop out layer. The first layer only consist the convolution layer. When choosing the loss function for out algorithm, we chose "parse categorical crossentropy", since this is what people used when classes of data set are mutually exclusive.

3 Features

In order to estimate the performance of a model for a given training run, we can further split the training set into a train and validation dataset. Performance on the train and validation dataset over each run can then be plotted to provide learning curves and insight into how well a model is learning the problem. Each training and test example is assigned to one of the following labels:0 T-shirt/top; 1 Trouser; 2 Pullover; 3 Dress; 4 Coat; 5 Sandal; 6 Shirt7 Sneaker; 8 Bag; 9 Ankle boot. Each row is a separate image. Column 1 is the class label.Remaining columns are pixel numbers (784 total).Each value is the darkness of the pixel (1 to 255). Since the image data in x_train and x_test is from 0 to 255 , we need to rescale this from 0 to 1 to fit with tensorflow.To do this we need to divide the x_train and x_test by 255 to normalize data.

4 Parameters

There are several parameters that we choose to modify to see the prediction results, which are number of epochs, number of neurons, number of hidden layers, drop out rate, activation functions, validation split rate and batch size.
Firstly, we need to find out the suitable number of layers that we need to have. After several trial, we choose three layers as our final choice. For number of epochs, we tried for 20, 30, 40 and 50, and finally find out that 30 gives out the best accuracy among these three. Thus, we choose to stick to 30 as out number of epochs. Besides, choosing appropriate number of neurons in each layer also plays an important role in getting high accuracy score. We tried several different numbers including 32, 64, 128, 256, and 512 and chose different combinations of them. Finally, we chose 32 for the first layer, 64 for the second layer, and 128 for the last layer.
Then, we adjust the drop out rate to see the accuracy. We tried several drop out rate, ranging from 0.2 to 0.5, and then we found that 0.5 gave the highest accuracy. The activation function part, we chose to use the most commonly used combination, which is "softmax" in the last layer, and "reLu" for the rest of the layers. Furthermore, for validation split rate, we tried 0.1 and 0.2, and we found out 0.1 gave us bette result. Lastly, we also tried 128, 256, 512, 640 and 1024 for the batch size. It turned out that 512 is the most appropriate one that gave the highest accuracy score, and lowest loss.
We can explore about various optimization algorithms which compute weights gradients and update them in order to minimize loss. Few of them are Gradient Descent, Adam, Adagrad, Adadelta, Nestrov Adam, RMSProp etc. In our model, we choose adam as optimizer.

5 Lessons Learned

When choosing the parameters, we should be very careful of being overfitting. For example, when we increase the number of epochs from 30 to 40 or even to 50. We can see slight increase on training accuracy, but also slight decrease in testing accuracy. And according to the result on Kaggle, we can see that when there are too much epochs, it is possible that overfitting happens.

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.python import keras
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, Conv2D, Dropout, M
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.models import load_model
import os
%matplotlib inline
```

In [ ]:
```
X_train = np.load('training_images.npy')
y_train = np.load('training_labels.npy')
X_test = np.load('test_images.npy')
```

In [ ]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
```

In [ ]:
```
X_train = X_train.astype('float32')   # change integers to 32-bit floating po
X_test = X_test.astype('float32')

X_train /= 255                        # normalize each value for each pixel f
X_test /= 255

X_train = X_train.reshape((60000, 28, 28, 1))
X_test = X_test.reshape((10000, 28, 28, 1))
```

In [ ]:
```
conv1 = layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1) )
```

In [ ]:
```
conv2 = layers.Conv2D(64, (3,3), activation='relu')
```

In [ ]:
```
conv3 = layers.Conv2D(128, (3,3), activation='relu')
```

In [ ]:
```
max_pool_1 = layers.MaxPooling2D((2,2))
max_pool_2 = layers.MaxPooling2D((2,2))
max_pool_3 = layers.MaxPooling2D((2,2))
```

In [ ]:
```
drop_1 = keras.layers.Dropout(0.5)
drop_2 = keras.layers.Dropout(0.5)
drop_3 = keras.layers.Dropout(0.5)
```

In [ ]:
```
flat_layer = layers.Flatten()
fc = layers.Dense(128, activation='relu')
output = layers.Dense(10, 'softmax')
```

In [ ]:
```
model = models.Sequential()

model.add(conv1)
# No Pooling Layer and Dropout layer for first Convolutional layer 'conv1'
model.add(conv2)
model.add(max_pool_2)
model.add(drop_2)
model.add(conv3)
model.add(max_pool_3)
model.add(drop_3)
model.add(flat_layer)
model.add(fc)
model.add(output)
```

In [ ]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [ ]:
```
model.fit(X_train, y_train, epochs=30, batch_size=512, shuffle=True, validati
```

```
Epoch 1/30
106/106 [==============================] - 57s 539ms/step - loss: 1.1641 - acc
uracy: 0.5706 - val_loss: 0.4784 - val_accuracy: 0.8380
Epoch 2/30
106/106 [==============================] - 53s 504ms/step - loss: 0.4933 - acc
uracy: 0.8188 - val_loss: 0.3751 - val_accuracy: 0.8753
Epoch 3/30
106/106 [==============================] - 56s 526ms/step - loss: 0.4272 - acc
uracy: 0.8445 - val_loss: 0.3416 - val_accuracy: 0.8818
```

```
Epoch 4/30
106/106 [==============================] - 56s 532ms/step - loss: 0.3794 - acc
uracy: 0.8606 - val_loss: 0.3182 - val_accuracy: 0.8948
Epoch 5/30
106/106 [==============================] - 56s 528ms/step - loss: 0.3364 - acc
uracy: 0.8784 - val_loss: 0.2779 - val_accuracy: 0.9035
Epoch 6/30
106/106 [==============================] - 58s 551ms/step - loss: 0.3114 - acc
uracy: 0.8875 - val_loss: 0.2665 - val_accuracy: 0.9085
Epoch 7/30
106/106 [==============================] - 73s 694ms/step - loss: 0.2988 - acc
uracy: 0.8905 - val_loss: 0.2466 - val_accuracy: 0.9163
Epoch 8/30
106/106 [==============================] - 53s 496ms/step - loss: 0.2827 - acc
uracy: 0.8944 - val_loss: 0.2470 - val_accuracy: 0.9137
Epoch 9/30
106/106 [==============================] - 55s 516ms/step - loss: 0.2672 - acc
uracy: 0.9013 - val_loss: 0.2275 - val_accuracy: 0.9193
Epoch 10/30
106/106 [==============================] - 56s 527ms/step - loss: 0.2510 - acc
uracy: 0.9073 - val_loss: 0.2271 - val_accuracy: 0.9208
Epoch 11/30
106/106 [==============================] - 54s 509ms/step - loss: 0.2446 - acc
uracy: 0.9113 - val_loss: 0.2269 - val_accuracy: 0.9185
Epoch 12/30
106/106 [==============================] - 68s 648ms/step - loss: 0.2421 - acc
uracy: 0.9108 - val_loss: 0.2100 - val_accuracy: 0.9253
Epoch 13/30
106/106 [==============================] - 102s 960ms/step - loss: 0.2341 - ac
curacy: 0.9151 - val_loss: 0.2042 - val_accuracy: 0.9285
Epoch 14/30
106/106 [==============================] - 100s 939ms/step - loss: 0.2244 - ac
curacy: 0.9182 - val_loss: 0.2051 - val_accuracy: 0.9293
Epoch 15/30
106/106 [==============================] - 98s 926ms/step - loss: 0.2161 - acc
uracy: 0.9198 - val_loss: 0.2015 - val_accuracy: 0.9282
Epoch 16/30
106/106 [==============================] - 99s 932ms/step - loss: 0.2130 - acc
uracy: 0.9204 - val_loss: 0.2002 - val_accuracy: 0.9305
Epoch 17/30
106/106 [==============================] - 101s 954ms/step - loss: 0.2089 - ac
curacy: 0.9234 - val_loss: 0.1902 - val_accuracy: 0.9342
Epoch 18/30
106/106 [==============================] - 100s 948ms/step - loss: 0.2037 - ac
curacy: 0.9238 - val_loss: 0.1954 - val_accuracy: 0.9317
Epoch 19/30
106/106 [==============================] - 99s 932ms/step - loss: 0.1987 - acc
uracy: 0.9266 - val_loss: 0.1850 - val_accuracy: 0.9358
Epoch 20/30
106/106 [==============================] - 98s 923ms/step - loss: 0.1915 - acc
uracy: 0.9308 - val_loss: 0.1871 - val_accuracy: 0.9323
Epoch 21/30
106/106 [==============================] - 99s 931ms/step - loss: 0.1878 - acc
```

```
uracy: 0.9297 - val_loss: 0.1815 - val_accuracy: 0.9357
Epoch 22/30
106/106 [==============================] - 99s 930ms/step - loss: 0.1829 - acc
uracy: 0.9325 - val_loss: 0.1787 - val_accuracy: 0.9370
Epoch 23/30
106/106 [==============================] - 98s 927ms/step - loss: 0.1767 - acc
uracy: 0.9332 - val_loss: 0.1821 - val_accuracy: 0.9360
Epoch 24/30
106/106 [==============================] - 96s 902ms/step - loss: 0.1754 - acc
uracy: 0.9353 - val_loss: 0.1761 - val_accuracy: 0.9380
Epoch 25/30
106/106 [==============================] - 97s 920ms/step - loss: 0.1750 - acc
uracy: 0.9345 - val_loss: 0.1774 - val_accuracy: 0.9355
Epoch 26/30
106/106 [==============================] - 98s 924ms/step - loss: 0.1679 - acc
uracy: 0.9355 - val_loss: 0.1734 - val_accuracy: 0.9380
Epoch 27/30
106/106 [==============================] - 96s 907ms/step - loss: 0.1645 - acc
uracy: 0.9384 - val_loss: 0.1735 - val_accuracy: 0.9373
Epoch 28/30
106/106 [==============================] - 98s 920ms/step - loss: 0.1619 - acc
uracy: 0.9389 - val_loss: 0.1764 - val_accuracy: 0.9358
Epoch 29/30
106/106 [==============================] - 97s 912ms/step - loss: 0.1580 - acc
uracy: 0.9408 - val_loss: 0.1741 - val_accuracy: 0.9360
Epoch 30/30
106/106 [==============================] - 97s 917ms/step - loss: 0.1559 - acc
uracy: 0.9409 - val_loss: 0.1756 - val_accuracy: 0.9372
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f89621894c0>
```

In [ ]:

```
model.predict(X_test)
```

Out[ ]:

```
array([[8.37147906e-02, 3.84798877e-05, 9.71522531e-04, ...,
        1.71396422e-07, 3.67848843e-05, 3.11473286e-06],
       [8.36527098e-11, 2.78041733e-14, 9.61827284e-11, ...,
        2.11799275e-07, 1.28576360e-11, 9.99996543e-01],
       [1.05089775e-05, 3.64192149e-14, 4.61700758e-08, ...,
        2.15900364e-09, 1.25583369e-07, 1.13255192e-04],
       ...,
       [1.43292732e-06, 2.43102125e-08, 2.82539804e-05, ...,
        7.50819140e-09, 4.52166489e-07, 4.69638735e-07],
       [2.97577557e-04, 1.56008859e-08, 2.08313704e-01, ...,
        1.04591260e-08, 1.91139407e-05, 6.64540494e-07],
       [1.64137043e-06, 1.24163515e-08, 6.88587605e-08, ...,
        9.99725997e-01, 7.14284806e-06, 2.17379711e-04]], dtype=float32)
```

In [ ]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 10, 10, 128)       73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
dropout_8 (Dropout)          (None, 5, 5, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               409728
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 503,690
Trainable params: 503,690
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
score = model.evaluate(X_train, y_train)
```

```
1875/1875 [==============================] - 48s 25ms/step - loss: 0.1124 - ac
curacy: 0.9622
```

In [ ]:

```
predicted_classes_probabilities = model.predict(X_test)
predicted_classes = np.argmax(predicted_classes_probabilities,axis=1)
```

In [ ]:

```
df={'LABEL':predicted_classes}
df=pd.DataFrame(data=df)
df['LABEL'].to_csv('output.csv',index_label='ID')
```

In [ ]:

```
predicted_classes.shape
```

Out[ ]:

```
(10000,)
```

In [ ]: