Deep_Learning_HW2

November 26, 2018

1 Anly 590 HW 2

1.1 Alex Archer

1.1.1 (1) Autoencoder

This first problem will deal with images in the mnist fashion data set.

```
In [98]: from keras.layers import Input, Dense, Embedding, LSTM, Bidirectional, SimpleRNN
         from keras.models import Model, Sequential
         from keras.datasets import mnist
         from keras import applications
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
         import pandas as pd
         from sklearn.model_selection import train_test_split
         from keras import optimizers
         from keras.layers.convolutional import Conv2D, MaxPooling2D,
         ZeroPadding2D, AveragePooling2D, Conv1D, MaxPooling1D
         from keras.callbacks import EarlyStopping
         from keras.preprocessing.image import ImageDataGenerator
         from keras.layers.normalization import BatchNormalization
         from keras.layers.core import Dense, Dropout, Activation, Flatten, Reshape
         from keras.optimizers import SGD, RMSprop
         from keras.utils import np_utils, to_categorical
         from keras.regularizers import l2
         from keras.preprocessing import sequence
         from sklearn.model_selection import train_test_split
         import string
         from sklearn.metrics import roc_auc_score
         import tensorflow as tf
         from keras.preprocessing.sequence import pad_sequences
```

```
In [99]: (x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

```
In [100]: # ==============================
          #  Reshaping data
```

```
# ============================
x_train = x_train.reshape(x_train.shape[0], 28, 28,1).astype('float32') / 255
x_test = x_test.reshape(x_test.shape[0], 28, 28,1).astype('float32') / 255
y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)
```

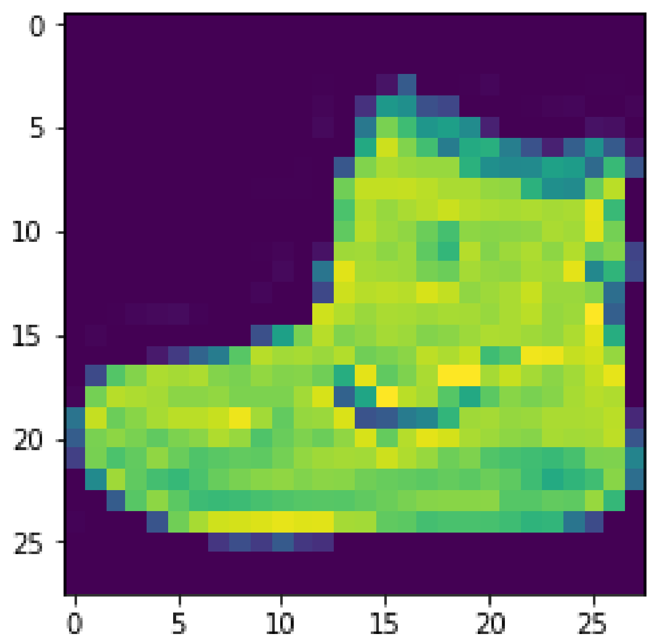
In [101]:
```
print(x_train.shape)
print(x_train[0].flatten().shape)
x_val = x_train.reshape(60000,28,28)
x_val = x_val.reshape(60000,784)
```

```
(60000, 28, 28, 1)
(784,)
```

Here is a sample image:

In [102]:
```
# example image
plt.imshow(x_train[0,:,:,0])
```

Out[102]: <matplotlib.image.AxesImage at 0x1e4fe37b70>



In [103]:
```
# auto encoder model
auto = Sequential()
auto.add(Conv2D(filters = 32,kernel_size=(3, 3),
                activation='relu', strides=(1, 1),
```

```
                        padding='valid',
                        input_shape=(28,28,1)))
    auto.add(Conv2D(filters = 32,kernel_size=(3, 3),
                        activation='relu', strides=(1, 1),
                        padding='valid'))
    auto.add(MaxPooling2D(pool_size=(2,2)))
    auto.add(Dropout(0.25))
    auto.add(Flatten())
    auto.add(Dense(64, activation='relu'))
    auto.add(Dropout(0.5))
    auto.add(Dense(784, activation='sigmoid'))
```

In [104]:
```
auto.compile(optimizer='adadelta', loss='mean_squared_error')
```

In [105]:
```
# fit model
auto.fit(x_train, x_val,
                epochs=5,
                batch_size=256,
                shuffle=True,
                verbose=1,
                validation_data=(x_train, x_val))
```

```
Train on 60000 samples, validate on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 129s 2ms/step - loss: 0.1303 - val_loss: 0.0856
Epoch 2/5
60000/60000 [==============================] - 131s 2ms/step - loss: 0.0894 - val_loss: 0.0709
Epoch 3/5
60000/60000 [==============================] - 131s 2ms/step - loss: 0.0703 - val_loss: 0.0507
Epoch 4/5
60000/60000 [==============================] - 131s 2ms/step - loss: 0.0583 - val_loss: 0.0442
Epoch 5/5
60000/60000 [==============================] - 130s 2ms/step - loss: 0.0532 - val_loss: 0.0410
```

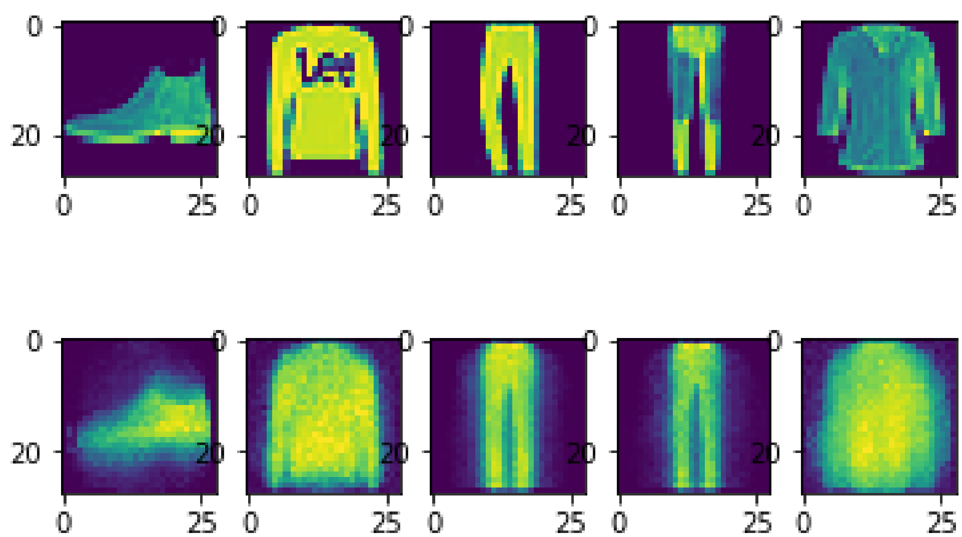
Out[105]:
```
<keras.callbacks.History at 0x1c3ae65898>
```

In [106]:
```
auto.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 12, 12, 32)        0         
```

```
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                294976
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 784)               50960
=================================================================
Total params: 355,504
Trainable params: 355,504
Non-trainable params: 0
_________________________________________________________________
```

In [107]:
```
# view some of the reconstructed images
n=5
for k in range(n):
    ax = plt.subplot(2, n, k+1)
    plt.imshow(x_test[k:k+1,:].reshape((28,28)))
    ax = plt.subplot(2, n, k+1 + n)
    reconstruction = auto.predict(x_test[k:k+1,:])
    #print(reconstruction.shape)
    reconstruction.resize((28,28))
    plt.imshow(reconstruction)
```



Above are some of the original images compared to their reconstructions via the auto encoder.

1.1.2 (2) Image Classification

1.1.3 (2.1) Deep CNN

In [108]:
```
# build model
dmodel = Sequential()

dmodel.add(Conv2D(filters = 32,kernel_size=(3, 3),
                  activation='relu', strides=(1, 1),
                  padding='valid', input_shape=(28,28,1)))
dmodel.add(Dropout(0.5))

dmodel.add(Conv2D(filters = 32,kernel_size=(3, 3),
                  activation='relu', strides=(1, 1),
                  padding='valid', input_shape=(28,28,1)))
dmodel.add(Dropout(0.5))

dmodel.add(Conv2D(filters = 32,kernel_size=(3, 3),
                  activation='relu', strides=(1, 1),
                  padding='valid', input_shape=(28,28,1)))
dmodel.add(MaxPooling2D(pool_size=(2, 2)))
dmodel.add(Dropout(0.5))

dmodel.add(Flatten())
dmodel.add(Dense(64))
dmodel.add(Dense(10))
dmodel.add(Activation('softmax'))
```

In [112]:
```
# train model
l_rate = 1
sgd = SGD(lr=l_rate, momentum=0.0, decay=0.0, nesterov=False)
dmodel.compile(loss='categorical_crossentropy', optimizer='adam',
               metrics=['accuracy'])
dmodel.fit(x_train, y_train, epochs=1,
           verbose=1, validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/1
60000/60000 [==============================] - 211s 4ms/step - loss: 0.3041 - acc: 0.8882 - val
```

Out[112]:
```
<keras.callbacks.History at 0xb20fdd278>
```

Training accuracy: 88.8%, test accuracy: 90.2%

In [123]:
```
# use predefined weights
conv_base = applications.VGG16(weights="imagenet", include_top=False,
                               input_shape=(32,32,3))
model2 = Sequential()
model2.add(conv_base)
```

```
model2.add(Flatten())
model2.add(Dense(128, activation="relu"))
model2.add(Dense(10, activation="sigmoid"))
model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 1, 1, 512)         14714688
_________________________________________________________________
flatten_6 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_20 (Dense)             (None, 10)                1290
=================================================================
Total params: 14,781,642
Trainable params: 14,781,642
Non-trainable params: 0
_________________________________________________________________
```

In [124]:
```
conv_base.trainable = False
model2.trainable_weights
```

Out[124]:
```
[<tf.Variable 'dense_19/kernel:0' shape=(512, 128) dtype=float32_ref>,
 <tf.Variable 'dense_19/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dense_20/kernel:0' shape=(128, 10) dtype=float32_ref>,
 <tf.Variable 'dense_20/bias:0' shape=(10,) dtype=float32_ref>]
```

In [125]:
```
model2.compile(loss="categorical_crossentropy", optimizer=RMSprop(1e-4),
               metrics=["acc"])
```

In [119]:
```
x_train2 = np.zeros(60000*32*32*3).reshape(60000,32,32,3)
x_test2 = np.zeros(10000*32*32*3).reshape(10000,32,32,3)

for i in range(0,x_train.shape[0]):
    x_train2[i,:,:,0] = np.pad(x_train[i,:,:,0],2,mode="constant")
    x_train2[i,:,:,1] = np.pad(x_train[i,:,:,0],2,mode="constant")
    x_train2[i,:,:,2] = np.pad(x_train[i,:,:,0],2,mode="constant")
```

In [126]:
```
x_train2.shape
```

Out[126]:
```
(60000, 32, 32, 3)
```

In [129]:
```
# fit the model
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
```

```
            rotation_range=20,
            width_shift_range=0.2,
            height_shift_range=0.2,
            horizontal_flip=True)

        datagen.fit(x_train2)

        model2.fit_generator(datagen.flow(x_train2, y_train, batch_size=32),
                             steps_per_epoch=len(x_train) / 32, epochs=1)
```

```
Epoch 1/1
1875/1875 [==============================] - 926s 494ms/step - loss: 0.5530 - acc: 0.7996
```

```
Out[129]: <keras.callbacks.History at 0x1e684c96d8>
```

This model achieved lower accuracy than the Deep CNN, though perhaps with more training the accuracy would increase.

1.1.4 (3) Text Classification

This problem will utilize different NNs to classify web urls as benign or malignant.

```
In [2]: # benign urls from text file
        benign = pd.read_csv("benign-urls.txt", header=None)
        benign.head()
```

```
Out[2]:                                0
        0                         # GOOGLE
        1        .0.blogger.gmodules.com
        2   .0.client-channel.google.com
        3              .0.docs.google.com
        4             .0.drive.google.com
```

```
In [3]: benign = benign.iloc[1:]
        benign.head()
```

```
Out[3]:                                0
        1        .0.blogger.gmodules.com
        2   .0.client-channel.google.com
        3              .0.docs.google.com
        4             .0.drive.google.com
        5                      .0.gvt0.cn
```

```
In [4]: # malignant urls from text file
        mal = pd.read_csv("malicious-urls.txt", header=None)
        mal.head()
```

Out[4]:
```
              0
0     .1337x.pl
1     .1link.io
2        .1n.pm
3  .22apple.com
4   .22find.com
```

In [5]:
```
benign["type"] = 0
mal["type"] = 1
```

In [6]:
```
# concat two dfs
text_class = pd.concat([benign, mal])
text_class.columns = ["url", "type"]
text_class.rename_axis(None)
text_class.head()
```

Out[6]:

	url	type
1	.0.blogger.gmodules.com	0
2	.0.client-channel.google.com	0
3	.0.docs.google.com	0
4	.0.drive.google.com	0
5	.0.gvt0.cn	0

In [7]:
```
text_class[::-1][:10]
```

Out[7]:

	url	type
1295	.zzbuckettownrock.com	1
1294	.zorpia.com	1
1293	.zophar.net	1
1292	.zoneiasiteoh.com	1
1291	.zkmobi.com	1
1290	.zippyshare.com	1
1289	.zipleisurefansstyles.com	1
1288	.zenmate.com	1
1287	.zend2.com	1
1286	.zebrametalmadnesscast.com	1

In [64]:
```
# train test split
x_train, x_test, y_train, y_test = train_test_split(text_class["url"],
                                                    text_class["type"], test_size=0.3
                                                    random_state=42)

x_train[0]
```

Out[64]: '.1337x.pl'

In [65]:
```
print(x_train.shape)
print(x_train[0])
```

```
(47146,)
.1337x.pl
```

```
In [66]: from keras.preprocessing.text import Tokenizer
         tokenizer = Tokenizer(char_level=True)
         tokenizer.fit_on_texts(x_train)
```

```
In [67]: # convert url strings to character strings
         x_train = tokenizer.texts_to_sequences(x_train)
         print(x_train[0])

         tokenizer = Tokenizer(char_level=True)
         tokenizer.fit_on_texts(x_test)

         x_test = tokenizer.texts_to_sequences(x_test)
         print(x_test[0])
```

```
[1, 19, 12, 6, 3, 1, 14, 2, 18, 1, 16, 28]
[1, 5, 10, 2, 6, 3, 1, 8, 30, 1, 28, 7, 5, 17, 3, 6, 5, 28, 30, 1, 4, 2, 8]
```

```
In [68]: print(type(x_train))
         len(x_train)
```

```
<class 'list'>
```

```
Out[68]: 47146
```

```
In [69]: # strings are of different lengths, so pad them
         x_train = pad_sequences(x_train, maxlen=86)
         x_test = pad_sequences(x_test, maxlen=86)
```

```
In [79]: x_train = np.array(x_train).reshape(47146,86)
         x_test = np.array(x_test).reshape(20206,86)
```

The first model will use a Simple RNN

```
In [80]: # build model
         model = Sequential()
         model.add(Embedding(input_length=86, input_dim=47146, output_dim=4))
         model.add(SimpleRNN(16))
         model.add(Dropout(0.1))
         model.add(Dense(1, activation='sigmoid'))
         model.compile('adam', 'binary_crossentropy', metrics=['accuracy'])
```

```
In [83]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_11 (Embedding)     (None, 86, 4)             188584
_________________________________________________________________
```

```
simple_rnn_8 (SimpleRNN)     (None, 16)                336
_________________________________________________________________
dropout_10 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 17
=================================================================
Total params: 188,937
Trainable params: 188,937
Non-trainable params: 0
_________________________________________________________________
```

```
In [86]: # fit model
         model.fit(x_train, y_train, epochs=2, verbose=1)
```

```
Epoch 1/2
47146/47146 [==============================] - 39s 829us/step - loss: 0.0594 - acc: 0.9810
Epoch 2/2
47146/47146 [==============================] - 38s 799us/step - loss: 0.0545 - acc: 0.9812
```

```
Out[86]: <keras.callbacks.History at 0x1a267781d0>
```

This model uses a 1D CNN

```
In [18]: model2 = Sequential()
         model2.add(Conv1D(filters = 32,kernel_size=(3),
                           activation='relu', strides=(1),
                           padding='valid',
                           input_shape=(86,1)))
         model2.add(Conv1D(filters = 32,kernel_size=(3),
                           activation='relu', strides=(1),
                           padding='valid'))
         model2.add(MaxPooling1D(pool_size=(1)))
         model2.add(Dropout(0.25))
         model2.add(Flatten())
         model2.add(Dense(64, activation='relu'))
         model2.add(Dropout(0.5))
         model2.add(Dense(1, activation='sigmoid'))
```

```
In [19]: model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 84, 32)            128
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 82, 32)            3104
_________________________________________________________________
```

```
max_pooling1d_1 (MaxPooling1 (None, 82, 32)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 82, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2624)             0
_________________________________________________________________
dense_2 (Dense)              (None, 64)               168000
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                65
=================================================================
Total params: 171,297
Trainable params: 171,297
Non-trainable params: 0
_________________________________________________________________
```

```
In [20]: model2.compile(optimizer="rmsprop", loss="binary_crossentropy",
                        metrics=["acc"])
```

```
In [25]: model2.fit(x_train, y_train,
                   epochs=2,
                   batch_size=32,
                  verbose = 1)
```

```
Epoch 1/2
47146/47146 [==============================] - 15s 309us/step - loss: 0.0788 - acc: 0.9800
Epoch 2/2
47146/47146 [==============================] - 15s 312us/step - loss: 0.0732 - acc: 0.9805
```

```
Out[25]: <keras.callbacks.History at 0x1a2294e2e8>
```

```
In [87]: y_pred = model.predict_proba(x_test)
         y_guess = (y_pred > .0025)*1
```

```
In [93]: # test accuracy
         round(sum((np.array(list(y_test)) == y_guess)[0]) / y_guess.shape[0],4)
```

```
Out[93]: 0.9813
```

```
In [94]: # AUC
         round(roc_auc_score(y_test, y_guess), 2)
```

```
Out[94]: 0.86
```

```
In [95]: # set a threshold
         x_test = x_test.reshape(20206,86,1)
         y_pred2 = model2.predict(x_test)
         y_guess2 = (y_pred2 > .0025)*1
```

```
In [96]: # test accuracy
         round(sum((np.array(list(y_test)) == y_guess2)[0]) / y_guess2.shape[0],4)

Out[96]: 0.9813

In [97]: # AUC
         round(roc_auc_score(y_test, y_guess2), 2)

Out[97]: 0.86
```

The two models have the same test accuracy and the same AUC. They likely get the same results for each url in the test set.