Deep Learning - Demo_02 - Sentiment Analysis - Assignment

June 1, 2020

1 Team members

- Prashanth ASOK KUMAR
- Annanya CHITRA KANNAN

2 Libraries used :

Below are the Libraries used in this Analysis:

```
import pandas as pd #used for reading CSV
import preprocess_twitter as pt #to make use of the preprocess_twitter.py
import embedding #to make use of the embedding.py
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.layers import Embedding, Input, GRU, Dense
from keras.models import Model
from keras import backend as com #used in Metric Calculation
from keras.layers import Conv1D, MaxPooling1D, Flatten
from keras.regularizers import l1 # used for regularization of CNN model
```

```
Using TensorFlow backend.
```

3 Functions used for Model metric calculation:

Following are the functions which is used for evaluating the RNN and CNN models

- “metric_recall” function to calculate the recall metric.
  - Recall = Sum of TruePositives / Sum of (TruePositives + FalseNegatives)
- “metric_precision” function to calculate the precision metric.
  - Precision = Sum of TruePositives / Sum of (TruePositives +FalsePositives)
- “f1” function to calculate the f1 metric.

- F1 = (2* Precision * Recall) / (Precision + Recall)

```
def metric_recall(y_val, y_pred):
    true_pos = com.sum(com.round(com.clip(y_val * y_pred, 0, 1)))
    posibl_pos = com.sum(com.round(com.clip(y_val, 0, 1)))
    recall = true_pos / (posibl_pos + com.epsilon())
    return recall

def metric_precision(y_val, y_pred):
    true_pos = com.sum(com.round(com.clip(y_val * y_pred, 0, 1)))
    pred_pos = com.sum(com.round(com.clip(y_pred, 0, 1)))
    precision = true_pos / (pred_pos + com.epsilon())
    return precision

def metric_f1(y_val, y_pred):
    precision = metric_precision(y_val, y_pred)
    recall = metric_recall(y_val, y_pred)
    return (2*((precision*recall)/(precision+recall+com.epsilon())))
```

4 Read data

4.1 Exercice 1: Load and process data.

Q1: Load the CSV file using the csv import in python.

Ans: We are making use of the **pandas** dataframe to read the csv file **"sanders-twitter-sentiment.csv"** . We are only making use of the **columns 4 and 5** since they are the columns which we are interested on.

```
interested_csv=pd.read_csv("sanders-twitter-sentiment.csv",header=␣
 ↪None,usecols=[3,4])
#columns 4 and 5 are "3 and 4" in pandas column naming convention.
```

Q2: Produce a python list of pre-processed tweets using the script preprocess_twitter.py

Ans: We are passing the column 4 from the csv, which contains the tweets data, to the script preprocess_twitter.py, to get the list of pre_processed tweets.

```
pre_processed=[]
for i in interested_csv[3]:
    pre_processed.append(pt.preprocess(i))
```

4.2 Exercice 2: Vectorize data.

Q1: Use these python lines to process data and labels:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(nb_words=10000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
data = pad_sequences(sequences, maxlen=32)
vocab = tokenizer.word_index
vocab['<eos>'] = 0
```

Ans:

We use the below set of lines to generate the tweets data array

vocab variable is a dictonary with value as numeric value of the elements in the array and key as its respective word.

```
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(pre_processed)
sequences = tokenizer.texts_to_sequences(pre_processed)
data = pad_sequences(sequences, maxlen=32)
vocab = tokenizer.word_index
vocab['<eos>'] = 0
```

We use the below set of lines to generate the labels

vocab_label is a dictionary with the value as numeric element of the array label and key as its respective word.

```
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(interested_csv[4])
sequences = tokenizer.texts_to_sequences(interested_csv[4])
label = pad_sequences(sequences, maxlen=4)
vocab_label = tokenizer.word_index
```

Q2: Now you should have two numpy arrays which shape are (5513, 32) and (5513, 4) for tweets and labels respectively.

Now cut this coprus in two parts one for the training and one for the evaluation. The training set (x_train and y_train) should contains 4000 utterances and the validation set (x_val, y_val) should have the rest of the data. Feel free to shuffle data.

Ans: We are splitting the corpus tweet and labels into 4000 utterances as training set and rest of the data as validation.

- tweets data into x_train and x_val.
- labels into y_train and y_val.

```
x_val, x_train = train_test_split(data,test_size=(4000/5513))
```

```
y_val, y_train = train_test_split(label,test_size=(4000/5513))
```

5 Train and evaluate the model

Using the embeddings.py module to load the WE into a numpy matrix with the help of below code:

```
weights = embedding.load(vocab, 100,'glove.twitter.27B.100d.filtered.txt')
```

```
loading embeddings from "glove.twitter.27B.100d.filtered.txt"
```

5.1 Exercice 1: Use RNN to train the first classifier:

```
from keras.layers import Embedding, Input, GRU, Dense
from keras.models import Model
#100-dim embeddings initialized with GloVe,
#over sequences of size 32, and not fine tuneable
embedding_layer = Embedding(len(vocab), 100, weights=[weights],input_length=32, trainable=False)
sequence_input = Input(shape=(32,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = GRU(64)(embedded_sequences)
preds = Dense(labels.shape[1], activation='softmax')(x)
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',optimizer='Nadam', metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val),nb_epoch=10, batch_size=64, shuffle=T
```

Q1: Write in Python the necessary script train the model

Ans: We are making use of the below code to train the RNN model. We have set the epoch value as 10, so we are training the code by making it train 10 times with the same data.

```
embedding_layer = Embedding(len(vocab), 100, weights=[weights],input_length=32,␣
 ↪trainable=False)
sequence_input = Input(shape=(32,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = GRU(64)(embedded_sequences)
preds = Dense(label.shape[1], activation='softmax')(x)
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',optimizer='Nadam', metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=10,␣
 ↪batch_size=64, shuffle=True)
```

```
Train on 4000 samples, validate on 1513 samples
Epoch 1/10
4000/4000 [==============================] - 2s 531us/step - loss: 0.1796 - acc:
0.9735 - val_loss: 0.0000e+00 - val_acc: 1.0000
```

```
Epoch 2/10
4000/4000 [==============================] - 1s 353us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 3/10
4000/4000 [==============================] - 1s 340us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 4/10
4000/4000 [==============================] - 1s 331us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 5/10
4000/4000 [==============================] - 1s 349us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 6/10
4000/4000 [==============================] - 1s 349us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 7/10
4000/4000 [==============================] - 1s 340us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 8/10
4000/4000 [==============================] - 1s 351us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 9/10
4000/4000 [==============================] - 1s 350us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 10/10
4000/4000 [==============================] - 1s 350us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
```

```
<keras.callbacks.callbacks.History at 0x1a3aeccd10>
```

Q2: Evaluate the model using precision, recall, F1-measure, and accuracy metrics.

Ans: We are assigning the metrics to the compile which we will be using to evaluate the model.

- "acc" to find accuracy
- metric_recall to find the recall metrics of the model
- metric_precision to find the precision metrics of the model
- metric_f1 to find the f1 metrics of the model

```
model.compile(loss='categorical_crossentropy',optimizer='Nadam',␣
↪metrics=['acc',metric_recall,metric_precision,metric_f1])
model.evaluate(x=x_val,y=y_val)
```

```
1513/1513 [==============================] - 0s 192us/step
```

```
[0.0, 1.0, 1.0, 1.0, 1.0]
```

Q3: How many epochs are needed to reach at least 80% of accuracy?

Ans: We are getting more than 80% accuracy with just **1 epoch**

```
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',optimizer='Nadam', metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=1,␣
 ↪batch_size=64, shuffle=True)
```

```
Train on 4000 samples, validate on 1513 samples
Epoch 1/1
4000/4000 [==============================] - 2s 550us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
```

```
<keras.callbacks.callbacks.History at 0x1a3a178b10>
```

5.2 Exercice 2: Use CNN to train the second classifier:

```
from keras.layers import Conv1D, MaxPooling1D, Flatten

sequence_input = Input(shape=(32,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 3, activation='relu')(embedded_sequences)
x = MaxPooling1D(3)(x) # global max pooling
x = Flatten()(x)
preds = Dense(labels.shape[1], activation='softmax')(x)

model = Model(sequence_input, preds)
```

Q1: Write in Python the necessary script train the model

Ans: We are training the CNN model using the below script:

```
sequence_input = Input(shape=(32,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 3, activation='relu')(embedded_sequences)
x = MaxPooling1D(3)(x) # global max pooling
x = Flatten()(x)
preds = Dense(label.shape[1], activation='softmax')(x)

model = Model(sequence_input, preds)
```

We are running the model for 10 epochs with the accuracy metrics

```
model.compile(loss='categorical_crossentropy',optimizer='Nadam', metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=10,␣
 ↪batch_size=64, shuffle=True)
```

```
Train on 4000 samples, validate on 1513 samples
Epoch 1/10
4000/4000 [==============================] - 1s 126us/step - loss: 0.0720 - acc:
0.9880 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 2/10
4000/4000 [==============================] - 0s 85us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 3/10
4000/4000 [==============================] - 0s 82us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 4/10
4000/4000 [==============================] - 0s 83us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 5/10
4000/4000 [==============================] - 0s 85us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 6/10
4000/4000 [==============================] - 0s 82us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 7/10
4000/4000 [==============================] - 0s 83us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 8/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 9/10
4000/4000 [==============================] - 0s 82us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
Epoch 10/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0000e+00 -
acc: 1.0000 - val_loss: 0.0000e+00 - val_acc: 1.0000
```

```
<keras.callbacks.callbacks.History at 0x1a3bd46bd0>
```

Q2: Evaluate the model using precision, recall, F1-measure, and accuracy metrics.

Ans: We are assigning the metrics to the compile which we will be using to evaluate the model. - "acc" to find accuracy - metric_recall to find the recall metrics of the model - metric_precision to find the precision metrics of the model - metric_f1 to find the f1 metrics of the model

```
model.compile(loss='categorical_crossentropy',optimizer='Nadam',␣
 ↪metrics=['acc',metric_recall,metric_precision,metric_f1])
model.evaluate(x=x_val,y=y_val)
```

```
1513/1513 [==============================] - 0s 84us/step
```

```
[0.0, 1.0, 1.0, 1.0, 1.0]
```

Q3: (Optional) Add a regularization method to the code and observe the results.

We are making use of the below code to regularize the CNN model.

```
preds = Dense(label.shape[1], activation='softmax',activity_regularizer=l1(0.
 ↪001))(x)
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',optimizer='Nadam', metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=10,␣
 ↪batch_size=64, shuffle=True)
```

```
Train on 4000 samples, validate on 1513 samples
Epoch 1/10
4000/4000 [==============================] - 1s 130us/step - loss: 2.0228 - acc:
0.9858 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 2/10
4000/4000 [==============================] - 0s 85us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 3/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 4/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 5/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 6/10
4000/4000 [==============================] - 0s 83us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 7/10
4000/4000 [==============================] - 0s 82us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 8/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 9/10
4000/4000 [==============================] - 0s 81us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
Epoch 10/10
4000/4000 [==============================] - 0s 82us/step - loss: 0.0637 - acc:
1.0000 - val_loss: 0.0634 - val_acc: 1.0000
```

```
<keras.callbacks.callbacks.History at 0x1a3c02d8d0>
```