Lab Session #12

Introduction

Welcome to our final Lab #12. This time, we'll experiment with *Deep Learning* techniques.

Follow-up Lab #12

Following are some sample scripts for last lab's questions:

Task #1: CNN Sentiment analysis

Since you are familiar with the linguistic feature extraction, let's try to use Convolution Neural Network(CNN) to perform a sentiment analysis task on IMDB movie review dataset. You can read more about applying CNN to language related task from here.

Download the dataset from here. IMDB

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Today we'll be using Keras library for its balance of friendly API and versatility. Keras by default uses TensorFlow as its backenend.

Sequential() is a class that is a neural net abstraction that gives you access to the basic API of Keras, specifically the methods compile and fit, which will do the heavy lifting of building the underlying weights and their interconnected relationships (compile), calculating the errors in training, and most importantly applying backpropagation (fit). Epochs, batch_size, and optimizer are all hyperparameters that will require tuning.

Import the following libraries, and set a seed value:

```
from keras.models import Sequential
from keras.layers import Conv1D, Dense, Dropout, Activation, MaxPooling1D, Flatten
import csv
from random import shuffle
import spacy
from sklearn.model_selection import train_test_split
import numpy as np

np.random.seed(1337)
```

Let's read the data file and create a datastructure in the form of a tuple, where the target is the first element and text as the second:

```
def pre_process_data(filepath):
    print('Preprocessing data...')
    dataset = []
    with open(filepath, 'r') as in_file:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        csv_reader = csv.reader(in_file, delimiter=',')
        next(csv_reader, None)
        for row in csv_reader:
            dataset.append((int(row[1]), row[0]))

    shuffle(dataset)
    return dataset
```

Once we have the data now we'll have to extract all word embeddings for each token in each data point we have in our dataset. We'll be using Spacy to extract the word embeddings. (Make sure you have the spacy library and the Large or medium language model already downloaded):

```
def tokenize_and_vectorize(dataset):
    print('Vectorizing data...')
    nlp = spacy.load('en_core_web_md')
    vectorized_data = []
    for sample in dataset:
        doc = nlp(sample[1])
        sample_vec = []
        for token in doc:
            try:
                sample_vec.append(token.vector)
            except KeyError:
                pass
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        vectorized_data.append(sample_vec)
    return vectorized_data
```

Next we extract all the target labels from the dataset:

```
def collect_expected(dataset):
    print('Target extraction...')
    # Extract target values from the dataset
    expected = []
    for sample in dataset:
        expected.append(sample[0])
    return expected
```

Even though reviews within the dataset vary is the size of length, CNN expects all the datapoints to have a fixed size of tokens within. Let's create a function to truncate or pad with 0s, all the datapoints within the dataset to have the same length:

```
def pad_trunc(data, maxlen):
    # For a given dataset pad with zero vectors or truncate to maxlen
    new_data = []

    # Create a vector of 0s the length of our word vectors
    zero_vector = [0] * len(data[0][0])
    for sample in data:
        if len(sample) > maxlen:
            temp = sample[:maxlen]
        elif len(sample) < maxlen:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
            temp = sample
            additional_elems = maxlen - len(sample)
            for _ in range(additional_elems):
                temp.append(zero_vector)
        else:
            temp = sample
        new_data.append(temp)
    return new_data
```

Since we are done with creating all necessary functions, let's start working on calling them:

```
dataset = pre_process_data('<path_to_data_file>')
vectorized_data = tokenize_and_vectorize(dataset)
expected = collect_expected(dataset)
```

Now let's split the data into train and test using sklearn's train_test_split function:

```
x_train, x_test, y_train, y_test = train_test_split(vectorized_data, expected, test_size=.20, random_state=40)
```

Let's initialize some parameters we'll be requiring to train our CNN model:

```
max_len = 400
batch_size = 32
embedding_dim = 300
filters = 250
kernel_size = 3
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
hidden_dim = 250
epochs = 2
num_classes = 1
```

Now let's try to normalize the sample datapoint to a fixed size and convert all datapoints to a numpy datastructure:

```
x_train = pad_trunc(x_train, max_len)
x_test = pad_trunc(x_test, max_len)

x_train = np.reshape(x_train, (len(x_train), max_len, embedding_dim))
y_train = np.array(y_train)
x_test = np.reshape(x_test, (len(x_test), max_len, embedding_dim))
y_test = np.array(y_test)
```

Finally we are done with the processing the input for our model. Let's now construct our CNN architecture:

```
model = Sequential()
model.add(Conv1D(filters=filters,
                 kernel_size=kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1,
                 input_shape=(max_len, embedding_dim)))

model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
model.add(Dense(hidden_dim))
model.add(Dropout(0.2))
model.add(Activation('relu'))

model.add(Dense(num_classes))
model.add(Activation('sigmoid'))
```

Once we've constructed the full architecture now let's compile it:

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

Now let's fine tune our model according to our training data:

```
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test))
```

Let's now try to print the system's performance with regard to our test data:

```
loss, acc = model.evaluate(x_test, y_test, verbose=0)

print ('Test Accuracy: %f' % (acc * 100))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

This is an additional step you can try on your own. Usually it is a better practice to save your model and it's weights, so that you don't have to train your model everytime you want to predict. Instead you can load the saved model and weights back again and use them for prediction. Refer to documentation as to how to save and load keras trained models.

Now's it's time to put our system to test. Let's predict. Given the following text, try to make a prediction:

```
sample_1 = """I always wrote this series off as being a complete stink-fest because Jim Belushi was involved in it,
```

```
vec_list = tokenize_and_vectorize([(0, sample_1)])
test_vec_list  = pad_trunc(vec_list, max_len)
test_vec = np.reshape(test_vec_list, (len(test_vec_list), max_len, embedding_dim))
print(model.predict(test_vec))
```

Task #2: Transformers

Let's see how we can work with state-of-the art transformer models to solve some NLP tasks. Instead of training the models ourselves, we will use the pre-trained models available from HuggingFace.

Before starting with the task check if you have tensorflow and trasformer library installed. If you do not have the libraries installed then use the following commands to install.

```
pip install tensorflow
pip install transformers
```

Task #2.1: Extractive Question Answering

Extractive Question Answering is the task of extracting an answer from a text given a question. We can either add the model to the pipeline or use a model and a tokenizer.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

To add the pretrained model in pipeline use the following command:

```
nlp = pipeline("question-answering")
```

Now lets try with question answering using a model and a tokenizer. A tokenizer is in charge of preparing the inputs for a model. The library contains tokenizers for all the models.

For this lab we will use TensorFlow. Now lets import the required libraries.

```
from transformers import AutoTokenizer, TFAutoModelForQuestionAnswering
import tensorflow as tf
```

Instantiate a tokenizer and a model from the checkpoint name. The model is identified as a BERT model and loads it with the weights stored in the checkpoint.

```
tokenizer = AutoTokenizer.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")
model = TFAutoModelForQuestionAnswering.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")
```

Now let us define passage and some questions based on the passage.

```
text = """In 1991, the remains of Russian Tsar Nicholas II and his family (except for Alexei and Maria) are discover
questions = ["Who became famous?","What was discovered in 1991?"]
```

For each question in the questions list we shall compute the confidence score. Here we will iterate over the questions and build a sequence from the text and the current question, with the correct model-specific separators token type ids and attention masks. Pass this sequence through the model. This outputs a range of scores across the entire sequence tokens (question and text), for both the start and

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

end positions. Compute the softmax of the result to get probabilities over the tokens. Convert the tokens from the identified start and stop values to string and print the result.

```
for question in questions:
    #tokenize the question
    inputs = tokenizer(question, text, add_special_tokens=True, return_tensors="tf")
    #Get the ids array details
    input_ids = inputs["input_ids"].numpy()[0]
    #Instantiates the model classes of the library (with a question answering head) from a configuration.
    outputs = model(inputs)
    #Get the start scores
    answer_start_scores = outputs.start_logits
    #Get the end scores
    answer_end_scores = outputs.end_logits
    # Get the most likely beginning of answer with the argmax of the score
    answer_start = tf.argmax(answer_start_scores, axis=1).numpy()[0]
    # Get the most likely end of answer with the argmax of the score
    answer_end = (tf.argmax(answer_end_scores, axis=1) + 1).numpy()[0]
    #Select the ids based on the scores and convert them into strings.
    answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(input_ids[answer_start:answer_end]))
    #Print the question and result
    print(f"Question: {question}")
    print(f"Answer: {answer}")
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Validate the generated answer. Try experimenting with different passage and different sets of questions.

For more information on Extractive Question Answering, please refer to this link.

Task #2.2: Text Generation

Text Generation is also known as open-ended text generation. Here we create a coherent portion of text that is a continuation from the given context. We can add the pretrained model to NLP pipeline or use model and tokenizer.

To add the pretrained transformer to NLP pipeline use the following command:

```
text_generator = pipeline("text-generation")
```

Now lets try generating the text using TensorFlow and use model and tokenizer. To begin, lets import the necessary libraries.

```
from transformers import TFAutoModelWithLMHead, AutoTokenizer
```

Instantiate a tokenizer and a model

```
model = TFAutoModelWithLMHead.from_pretrained("xlnet-base-cased")
tokenizer = AutoTokenizer.from_pretrained("xlnet-base-cased")
```

Define the passage and the text from where the transformer will start with text generation.

```
text = """In 1991, the remains of Russian Tsar Nicholas II and his family (except for Alexei and Maria) are discover
prompt = "Today the weather is really nice and I am planning on "
```

Converts a text to a sequence of ids (integer), using the tokenizer and vocabulary.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
inputs = tokenizer.encode(text + prompt, add_special_tokens=False, return_tensors="tf")
```

Length of the converted sequence of ids in a text, using the tokenizer and vocabulary with options to remove special tokens and clean up tokenization spaces.

```
prompt_length = len(tokenizer.decode(inputs[0], skip_special_tokens=True, clean_up_tokenization_spaces=True))
```

Using the generate() function to generate the text. We are also providing maximum length of the generated text.

```
outputs = model.generate(inputs, max_length=250, do_sample=True, top_p=0.95, top_k=60)
```

Contact the initial prompt text and generated text. Here we are using decoder to get the generated text.

```
generated = prompt + tokenizer.decode(outputs[0])[prompt_length:]
```

Print the generated text.

```
print(generated)
```

For more information on the Text Generation, have a look at this link.

That's all for this course!

Last modified: Tuesday, 13 April 2021, 7:00 PM

Jump to...

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD