Natural Language Processing in TensorFlow

Week 1 - Sentiment in text

This week they talked about word representations and the transformation of sentences in vectors (tokenization).

Below there is a snippet of getting some texts from JSON files and printing the tokenization.

```
!wget                                    --no-check-certificate                                    \
https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sarcasm.json \
   -O /tmp/sarcasm.json

import json

with open("/tmp/sarcasm.json", 'r') as f:
   datastore = json.load(f)


sentences = []
labels = []
urls = []
for item in datastore:
   sentences.append(item['headline'])
   labels.append(item['is_sarcastic'])
   urls.append(item['article_link'])


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

tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)

word_index = tokenizer.word_index
print(len(word_index))
       29657

print(word_index)
{'<OOV>': 1, 'to': 2, 'of': 3, 'the': 4, 'in': 5, 'for': 6, 'a': 7, 'on':
8, 'and': 9, 'with': 10, 'is': 11, 'new': 12,
```

```
sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, padding='post')

print(padded[0])
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0   308 15115   679  3337  2298    48   382  2576
 15116     6  2577  8434]

print(padded.shape)
(26709, 40)
```

Week 2 - Word Embeddings

This week they talked about tokenizing sentences (and sentences to sequences) and also about word embedding, what it is and how they are created using neural networks.

Below there are some snippets on how to tokenize sentences.

```
vocab_size = 1000
oov_tok = '<OOV>'
max_length = 120
trunc_type = 'post'

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_sentences)

train_sequences = tokenizer.texts_to_sequences(train_sentences)
train_padded = pad_sequences(train_sequences, maxlen=max_length, truncating=trunc_type)
```

Below there are some snippets on how to train the neural networks to obtain the word embeddings.

```
embedding_dim = 16

model = tf.keras.Sequential([
 tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
 tf.keras.layers.GlobalAveragePooling1D(),
 tf.keras.layers.Dense(24, activation='relu'),
 tf.keras.layers.Dense(6, activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

num_epochs = 30
history = model.fit(
        Train_padded,
        training_label_seq,
        epochs=num_epochs,
        validation_data=(validation_padded, validation_label_seq),
        verbose=2
)
```

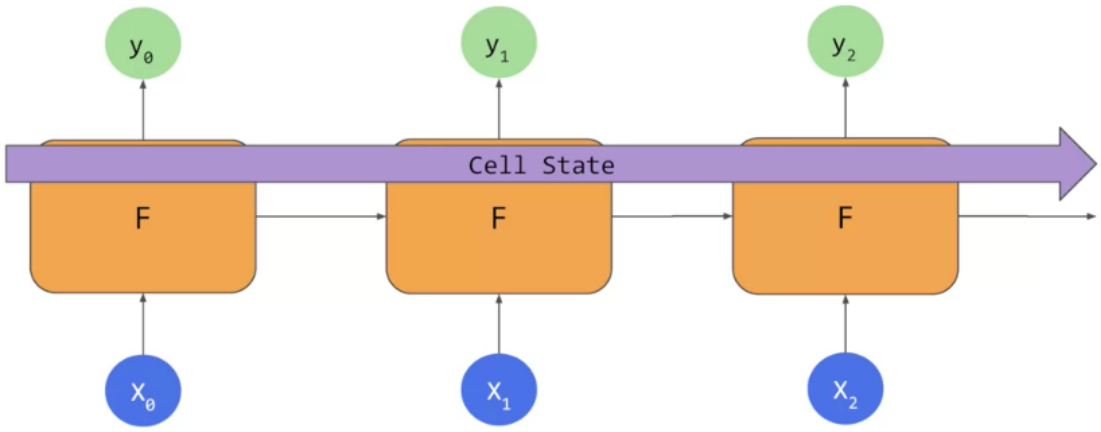
Week 3 - Sequence models

This week they talked about sequence models (LSTM, e.g.) and how those types of models can appropriately balance the importance of words in different parts of the sentence for it's meaning.

Below there is a snippet of a NN using dropout (to mitigate overfitting), convolution, pooling and LSTM.

```
model = tf.keras.Sequential([
   tf.keras.layers.Embedding(
       vocab_size+1,
       embedding_dim,
       input_length=max_length,
       weights=[embeddings_matrix],
       trainable=False),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.Conv1D(64, 5, activation='relu'),
   tf.keras.layers.MaxPooling1D(pool_size=4),
   tf.keras.layers.LSTM(64),
   tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

The logic of context passing over cells in an LSTM neural network.



Week 4 - Sequence models and literature

This week they talked about sequence models and predicting next words in a given sentence.

In the snippet below there is the preprocessing of the data that will be the input of the model (xs and ys).

```
input_sequences = []
for line in corpus:
 token_list = tokenizer.texts_to_sequences([line])[0]
 for i in range(1, len(token_list)):
   n_gram_sequence = token_list[:i+1]
   input_sequences.append(n_gram_sequence)

# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
xs, labels = input_sequences[:,:-1],input_sequences[:,-1]

ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

The xs are all the words in the line but the last one, and, the ys are all the last words of each line.

In the snippet below there is the model structure and training.

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(xs, ys, epochs=500, verbose=1)
```

Note here that max_sequence_len is defined as the maximum length taking into account all lines/sentences inputted.

Lastly, in the snippet below there is the concatenation of the prediction, word by word, in the initial sentence.

```
seed_text = "Laurence went to dublin"
next_words = 100
for _ in range(next_words):
 token_list = tokenizer.texts_to_sequences([seed_text])[0]
 token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
 predicted = model.predict_classes(token_list, verbose=0)
 output_word = ""
 for word, index in tokenizer.word_index.items():
   if index == predicted:
     output_word = word
     break
 seed_text += " " + output_word
print(seed_text)
```

```
Laurence went to dublin dancing round merry the plenty as water water red
rose together (AND SO ON)
```