Natural Language Processing Using TensorFlow

Word Based Embedding using API's.

```
from tensorflow.keras.preprocessing.text import Tokenizer
sentences = [ 'I love my dog',
              'I love my cat',
              'You love my dog!' ]
tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

num_words → specifies max no. of words − 1 to keep when generating sequences.

Output: {'i': 3, 'my': 2, 'you': 6, 'love': 1, 'cat': 5, 'dog': 4}

Represent text in numerical representations which can be used to train neural network.

Take each word from list of sentences and assign integer using fit_on_texts(). Get the result by word_index property.

More frequent words → lower index.
By default → punctuation ignored & converted to lower. Changed through filters & lower arguments of Tokenizer.

Text To Sequence

out of vocab

```
sentences = ['I love my dog',
             'I love my cat', 'You love my dog!',
             'Do you think my dog is amazing?']
tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(sentences)
print(word_index)
print(sequences)
test_data = ['i really love my dog',
             'my dog loves my manatee']
test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

To Add Padding:

```
from tensorflow.keras.preprocessing.sequence import
                                        pad_sequences
padded = pad_sequences(sequences)
For padding after the sentence
padded = pad_sequences(sequences, padding='post')

padded = pad_sequences(sequences, padding = post,
truncating = 'post', maxlen = 5)
```

If losing info from pre, adding post makes it lose from end of the padding

have sentences of length not more than 5 to be padded.

NLP Using Tensorflow

using datasets.

Monday 28 OCT 2024

=> Sarcasm in news headlines.

is_sarcastic
headline
article_link

Loading the data:

```
import json
with open("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'])

Generate ward index
tokenizer = Tokenize.fit_on_texts(sentences)
word_index = tokenizer.ward_index
Create sequences & do padding.
```

NOTES

303-063 • WK 44 OCT 2024

Word Embeddings

29 tfds Tuesday

September 2024

M	T	W	T	F	S	S
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

October 2024

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

```
import tensorflow as tf
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews", with_info=
                True, as_supervised = True)
import numpy as np
train_data, test_data = imdb['train'], imdb['test']

training_sentences = []
training_labels = []
test_sentences = []
test_labels = []

for s,l in train_data:
    training_sentences.append(s.numpy().decode('utf-8'))
    training_labels.append(l.numpy())

for s,l in test_data:
    test_sentences.append(s.numpy().decode('utf8'))
    test_labels.append(l.numpy())

training_labels_final = np.array(training_labels)
test_labels_final = np.array(test_labels)

#hyperparameters
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type = 'post'
oov_tok = "<OOV>"
```

NOTES

Word Embedding. Vectors for each word with their associated sentiment.

Wednesday 30 OCT 2024

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

tokenizer = Tokenizer(num_words = vocab_size,
                      oov_tokens = oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences, maxlen =
                       max_length, truncating = trunc_type)
testing_sequences = tokenizer.texts_to_sequences
                    (testing_sentences)
testing_padded = pad_sequences(testing_sequences,
                               maxlen = max_length)

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size,
    embedding_dim, input_length = max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')])

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

num_epochs = 10
```

NOTES

```
model.fit(padded, training_labels_final,
          epochs = num_epochs,
          validation_data = (testing_padded,
          test_labels_final))

e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape)          # (10000, 16)

reverse_word_index = tokenizer.index_word

import io
out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
    word = reverse_word_index[word_num]
    embeddings = weights[word_num]
    out_m.write(word + "\n")
    out_v.write('\t'.join([str(x) for x in embeddings])
    + "\n")
out_v.close()
out_m.close()
```

```
#In colab
try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download('vecs.tsv')
  files.download('meta.tsv')
```

NOTES

306-060 • WK 44 NOV 2024

Building a classifier for sarcasm dataset.

01 Friday

October 2024

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November 2024

M	T	W	T	F	S	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

```
import json
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
                                                sequence  pad_sequences

# hyperparameters
vocab_size = 10000
embedding_dim = 16
max_length = 32
trunc_type = 'post'
padding_type = 'post'
oov_tok = '<OOV>'
training_size = 20000

# download the sarcasm dataset.
with open("/tmp/sarcasm.json", "r").as f:
    datastore = json.load(f)
sentences = []
labels = []

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

NOTES

```
training_sentences = sentences[0:training_size]
training_labels = labels[0:training_size]
testing_sentences = sentences[training_size:]
testing_labels = labels[training_size:]
```

```
#tokenize training_sentences
#sequence training_sentences & testing_sentences
#pad both.
#Convert the labels list to np.array.
#Create usual neural network - Sequential
#Use GlobalAveragePooling1D.
#Compile with binary_crossentropy

num_epochs = 30
model.fit(training_padded, training_labels,
epochs = num_epochs, validation_data =
(testing_padded, testing_labels), verbose = 2)

#Plotting the results.
import matplotlib.pyplot as plt
def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history['val_'+string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, 'val_'+string])
    plt.show()
```

Sunday 03

NOTES

```
plot_graphs(history, 'acc')
plot_graphs(history, 'loss')
```

Validation loss found to be increasing why?

DECEMBER

04 Monday

Solution: tweak the hyperparameters

vocab_size = 1000 embedding_dim = 32
max_length = 16

imdb subwords 8K using single layer LSTM

RNN - takes ordering of inputs into account.

LSTM - computes the state of current timestep & passes it on to the next timestep where the state is also updated. The process repeats until final timestep where the output computation is affected by all previous timestep states.

```
import tensorflow as tfds.
dataset, info = tfds.load('imdb_reviews/subwords 8K', with_info=True, as_supervised=True)
tokenizer = info.features['text'].encoder

BUFFER_SIZE = 10000
BATCH_SIZE = 256
train_data, test_data = dataset['train'],
                        dataset['test'],
train_dataset = train_data.shuffle(BUFFER_SIZE)

train_dataset = train_dataset.padded_batch(BATCH_SIZE)

test_dataset = test_data.padded_batch(BATCH_SIZE)
```

NOTES

Model building : Swap Flatten() & GlobalAveragePooling1D() with LstmLayer.

Tuesday 05 NOV 2024

rest it inside Bidirectional layer

So passing the sequence info goes both forward & backward.

```
embedding_dim = 64
lstm_dim = 64
dense_dim = 64
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size,
    embedding_dim),
    tf.keras.layers.Bidirectional(tf.keras.layers.
                                  LSTM(lstm_dim)),
    tf.keras.layers.Dense(dense_dim, activation='re
    tf.keras.layers.Dense(1, activation='sigmoid')])
```

To build Multilayer LSTM simply append another LSTM layer in your sequential model & enable return_sequences as True.

↙ Why?

Because LSTM layer expects a sequence input, so if previous input is also an LSTM, then it should output a sequence as well.

Multilayer LSTM model

```
embedding_dim = 64
lstm1_dim = 64
lstm2_dim = 32
dense_dim = 64
model = tf.keras.Sequential([
tf.keras.layers.Embedding([tokenizer.vocab_size,
                           embedding_dim),
```

```
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM
    (lstm1_dim, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.
    LSTM(lstm2_dim)),
    tf.keras.layers.Dense(dense_dim, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Generating text with Neural Networks

⇒ Building the vocab

```
data = "Song lyrics separated by \n"
corpus = data.split lower().split('\n')
toKenizer = Tokenizer()
toKenizer.fit_on_texts(corpus)
total_words = len(toKenizer.word_index) + 1
```

⇒ Preprocessing the data.

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.fit texts_to_sequences
    ([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequences = token_list[:i+1]
        input_sequences.append(n_gram_sequences)
max_seq_length = max([len(x) for x in input_
                      sequences])
```

NOTES

```
input_sequences = np.array(pad_sequences(input_
    sequences, maxlen = max_seq_length,
    padding='pre'))

xs, labels = input_sequences[:,:-1],
             input_sequences[:,-1]

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

sentence = corpus[0].split()
token_list = []
for word in sentence:
    token_list.append(tokenizer.word_index[word])

elem_number = 6
print(xs[elem_number])
print(tokenizer.sequences_to_texts([xs[elem_number]]))
print(ys[elem_number])
print(np.argmax(ys[elem_number]))
```

=> Build sequential model with embeddings (totalwords, 64, input length = max_seq_length - 1), LSTM (20) bidirectional and dense(total words, softmax)

=> compile with categorical crossentropy, adam & accuracy metrics.

NOTES

313-053 • WK 45
NOV 2024
08 Friday

October 2024

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November 2024

M	T	W	T	F	S	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Generating the text Algorithm

9
→ Feed the seed text to initiate the process
10 → Model predicts the index of the most probable next word.
11 → Look up the index in reverse word index dictionary
12 → Append the next word to the seed text.
→ Feed the result to the model again.

1
```
seed_text = "Pratik went to Dublin"
2 next_word = 100
for _ in range(next_words):
3
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
4   token_list = pad_sequences([token_list], maxlen=
      max_seq_length-1, ~~pad~~ padding='pre')
5   probabilities = model.predict(token_list)
    predicted = np.argmax(probabilities, axis=-1)[0]
6   #feed the model to get ~~proper~~ probabilities
    # for each index and get the index with
7   # highest probability
    if predicted != 0:
        output_word = tokenizer.index_word[predicted]
        seed_text += " " + output_word
```

NOTES

Text Generation with RNN

9

→ Import Tensorflow & other libraries
10 → Download Shakespeare dataset
→ Read the data
11 → Process the text
→ Vectorise the text
12 → Prediction task
→ Create training example & targets
1 → Create training batches
→ Build the model.
2 → Train the ~~model~~ model
→ Attach an optimizer, & loss func.
3 → Config checkpoints
→ Execute training
4 → Generate text
→ Export generator.
→ ~~Co Ce~~ Customize training.

Sunday 10

NOTES