PSL-Week-RNN-text_classification

November 21, 2022

1 Text classification with an RNN

This text classification trains a recurrent neural network (RNN) on the IMDB large movie review dataset for sentiment analysis.

1.1 Setup

Before starting, it is interesting to check the capabilities we have in our server.

[1]:
```
!nvidia-smi #This is the command to know the GPU info
```

```
Sun Nov 20 23:09:02 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[ ]:
```
cat /proc/meminfo # This is the command to know the RAM memory we have
```

[3]:
```
!lscpu #We can also check the CPU processor we have in our machine
```

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  2
Core(s) per socket:  1
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) CPU @ 2.00GHz
Stepping:            3
CPU MHz:             2000.170
BogoMIPS:            4000.34
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            1024K
L3 cache:            39424K
NUMA node0 CPU(s):   0,1
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx
f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb
stibp fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx avx512f
avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt
xsavec xgetbv1 xsaves arat md_clear arch_capabilities
```

[4]:
```
import numpy as np

import tensorflow_datasets as tfds
import tensorflow as tf
```

Import `matplotlib` and create a helper function to plot graphs:

[5]:
```
import matplotlib.pyplot as plt


def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
```

1.2 Setup input pipeline

The IMDB large movie review dataset is a *binary classification* dataset—all the reviews have either a *positive* or *negative* sentiment.

[6]:
```
dataset, info = tfds.load('imdb_reviews', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

train_dataset.element_spec
```

[6]:
```
(TensorSpec(shape=(), dtype=tf.string, name=None),
 TensorSpec(shape=(), dtype=tf.int64, name=None))
```

[7]:
```
len(dataset['train'])
```

[7]:
```
25000
```

[8]:
```
len(dataset['test'])
```

[8]:
```
25000
```

Initially this returns a dataset of (text, label) pairs:

[9]:
```
for example, label in train_dataset.take(1):
  print('text: ', example.numpy())
  print('label: ', label.numpy())
```

```
text:  b"This was an absolutely terrible movie. Don't be lured in by Christopher
Walken or Michael Ironside. Both are great actors, but this must simply be their
worst role in history. Even their great acting could not redeem this movie's
ridiculous storyline. This movie is an early nineties US propaganda piece. The
most pathetic scenes were those when the Columbian rebels were making their
cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love
affair with Walken was nothing but a pathetic emotional plug in a movie that was
devoid of any real meaning. I am disappointed that there are movies like this,
ruining actor's like Christopher Walken's good name. I could barely sit through
it."
label:  0
```

Next shuffle the data for training and create batches of these `(text, label)` pairs:

[10]:
```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

[11]:
```
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).
↪prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

[12]:
```
for example, label in train_dataset.take(1):
  print('texts: ', example.numpy()[:3])
  print()
  print('labels: ', label.numpy()[:3])
```

```
texts:  [b"This was so lame that I turned the DVD off…maybe halfway through.
It was so weak, I couldn't even pay full enough attention to tell you how far in
I made it.Though I really wanted to believe that the depiction of the young
Carlito would be somewhat different, I just couldn't buy it. I don't really
blame the actors, because I think it was the script that may have fallen flat. I
did find myself laughing a few times, but I don't think those lines were
intended to be funny.<br /><br />It's only saving grace is that I bought it in a
2 DVD set and I would have paid the price I did for the original alone. This is
one of those cases when they should have let the classic stand alone."
 b'You remember the Spice Girls movie and how bad it was (besides the songs),
well their manager Simon Fuller (also this band\'s manager) makes the same error
putting S Club (another of my favourite bands) in their own film. S Club: Tina
Barrett, Jon Lee, Bradley Mcintosh, Jo O\'Meara, Hannah Spearritt and Rachel
Stevens (what happened to the seventh member, Paul Cattermole?) basically ask
their boss for a break, they go on, and while there they see themselves on TV!
Three of them swap, and vice versa, half discover they are clones made by a
greedy scientist, and the other half just get themselves in trouble. Also
starring Gareth Gates as a clone of himself. This film may have more of a plot
than Spice Girls\' film did, but besides the songs "Bring It All Back", "Don\'t
Stop Movin\'" and "Never Had A Dream Come True" this is no goo reason to see
this film. Not too long after the band split for good. Adequate!'
 b"First of all sorry for giving even a rating of 1 to this movie (nothing less
than this available). The film fails in every department be it screenplay,
direction, characterization or acting. <br /><br />1) To start with, the name of
the movie is really C class (though the movie itself match up to the name). 2)
Amitabh Bachchan tries his best to live up to the character but the weak script
coupled with pathetic direction ends up making him a humorous character. 3) In
Sholay Gabbar Singh has reward of 50,000 on him (which was convincing). Here in
Aag the figure was 100 crores for Babban (Amitabh Bachchan but poor man was
beaten by our so called hero's and had only few men bikes to commute (with all
automatic guns). Making a Sholay like movie in Mumbai type setup in modern time
doesn't look convincing. 4) As for Nisha Kothari, somebody needs to tell her
that she doesn't know acting. Why is Ram Gopal Verma casting her again and again
? 5) Mohanlal was good but there is hardly anything for him to do. 6) Sushant
Singh and Rajpal Yadav who are great actors are wasted in the movie. 7) Legendry
role played by Lila Misra (Mausi of Basanti) in Sholay is replaced cheaply in
this movie by some Gangu Mummy. Ramu please grow up and understand that there
needs to be some intellect in your movie. Enough of stupid characters in your
movie like Shiva and Aag. 8) Should not say anything about modern Jai and
Veeru..pathetic to the greatest extent.<br /><br />To summarize, I was shocked
to see this movie because it looks like a cheap and comic translation of
original classic. Please don't waste money and time on this movie. I think
```

```
watching Aap Ka Surror (which I thought was the worst movie possible) would be a
better idea than to see this horrible package of stupid characters, bad songs
and miserable direction.<br /><br />Thanks, Saurabh"]

labels:  [0 0 0]
```

1.3 Create the text encoder

The raw text needs to be processed before it can be used in a model. The simplest way to process text for training is using the `TextVectorization` function.

[13]:
```
VOCAB_SIZE = 1000
encoder = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label: text))
```

The `.adapt` method sets the layer's vocabulary. Here are the first 20 tokens.

[14]:
```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

[14]:
```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U14')
```

1.4 Create the model

1. This model is built with `tf.keras.Sequential`.
2. The first layer is the `encoder`, which converts the text to a sequence of token indices.
3. After the encoder is an embedding layer. An embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.
4. A recurrent neural network (RNN) processes sequence input by iterating through the elements. RNNs pass the outputs from one timestep to their input on the next timestep.

The code to implement this is below:

[15]:
```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
```

```
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Compile the Keras model to configure the training process:

[16]:
```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

1.5 Train the model

[17]:
```
import os
checkpoint_path = "training_rnn/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
```

[18]:
```
# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)
```

[19]:
```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30,
                    callbacks=[cp_callback])
```

```
Epoch 1/10
391/391 [==============================] - ETA: 0s - loss: 0.6471 - accuracy:
0.5680
Epoch 1: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 47s 91ms/step - loss: 0.6471 -
accuracy: 0.5680 - val_loss: 0.5087 - val_accuracy: 0.7797
Epoch 2/10
391/391 [==============================] - ETA: 0s - loss: 0.4151 - accuracy:
0.8038
Epoch 2: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 35s 89ms/step - loss: 0.4151 -
accuracy: 0.8038 - val_loss: 0.3689 - val_accuracy: 0.8411
Epoch 3/10
391/391 [==============================] - ETA: 0s - loss: 0.3418 - accuracy:
0.8480
Epoch 3: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 87ms/step - loss: 0.3418 -
accuracy: 0.8480 - val_loss: 0.3399 - val_accuracy: 0.8516
Epoch 4/10
```

```
391/391 [==============================] - ETA: 0s - loss: 0.3214 - accuracy:
0.8602
Epoch 4: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 87ms/step - loss: 0.3214 -
accuracy: 0.8602 - val_loss: 0.3442 - val_accuracy: 0.8552
Epoch 5/10
391/391 [==============================] - ETA: 0s - loss: 0.3123 - accuracy:
0.8641
Epoch 5: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 86ms/step - loss: 0.3123 -
accuracy: 0.8641 - val_loss: 0.3285 - val_accuracy: 0.8401
Epoch 6/10
391/391 [==============================] - ETA: 0s - loss: 0.3115 - accuracy:
0.8649
Epoch 6: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 86ms/step - loss: 0.3115 -
accuracy: 0.8649 - val_loss: 0.3277 - val_accuracy: 0.8594
Epoch 7/10
391/391 [==============================] - ETA: 0s - loss: 0.3032 - accuracy:
0.8688
Epoch 7: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 87ms/step - loss: 0.3032 -
accuracy: 0.8688 - val_loss: 0.3194 - val_accuracy: 0.8542
Epoch 8/10
391/391 [==============================] - ETA: 0s - loss: 0.3032 - accuracy:
0.8696
Epoch 8: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 87ms/step - loss: 0.3032 -
accuracy: 0.8696 - val_loss: 0.3211 - val_accuracy: 0.8469
Epoch 9/10
391/391 [==============================] - ETA: 0s - loss: 0.2989 - accuracy:
0.8715
Epoch 9: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 87ms/step - loss: 0.2989 -
accuracy: 0.8715 - val_loss: 0.3187 - val_accuracy: 0.8516
Epoch 10/10
391/391 [==============================] - ETA: 0s - loss: 0.2977 - accuracy:
0.8712
Epoch 10: saving model to training_rnn/cp.ckpt
391/391 [==============================] - 34s 86ms/step - loss: 0.2977 -
accuracy: 0.8712 - val_loss: 0.3180 - val_accuracy: 0.8589
```

[20]:
```
# This comand is to download the trained model so we don't need to retrain each␣
↪time we want to use it

#from google.colab import files
#os.system( "zip -r {} {}".format( 'training_rnn.zip' , 'training_rnn' ) )
```

```
#files.download('training_rnn.zip')
```

[21]:
```
# If we execute this evaluation withouth training the model, is going to␣
 ↪execute the default model
# The default model has no training so the accuracy is going to be low.

test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
391/391 [==============================] - 19s 49ms/step - loss: 0.3141 -
accuracy: 0.8618
Test Loss: 0.31409668922424316
Test Accuracy: 0.8617600202560425
```

[22]:
```
#We can use this command to upload our trained model, or just click on the␣
 ↪upload icon on the left upper corner.

#files.upload()
```

[23]:
```
#!unzip training_rnn.zip
```

[24]:
```
# Loads the weights
model.load_weights(checkpoint_path)
```

[24]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f3b3c529fd0>
```

[25]:
```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
391/391 [==============================] - 20s 50ms/step - loss: 0.3141 -
accuracy: 0.8618
Test Loss: 0.31409668922424316
Test Accuracy: 0.8617600202560425
```

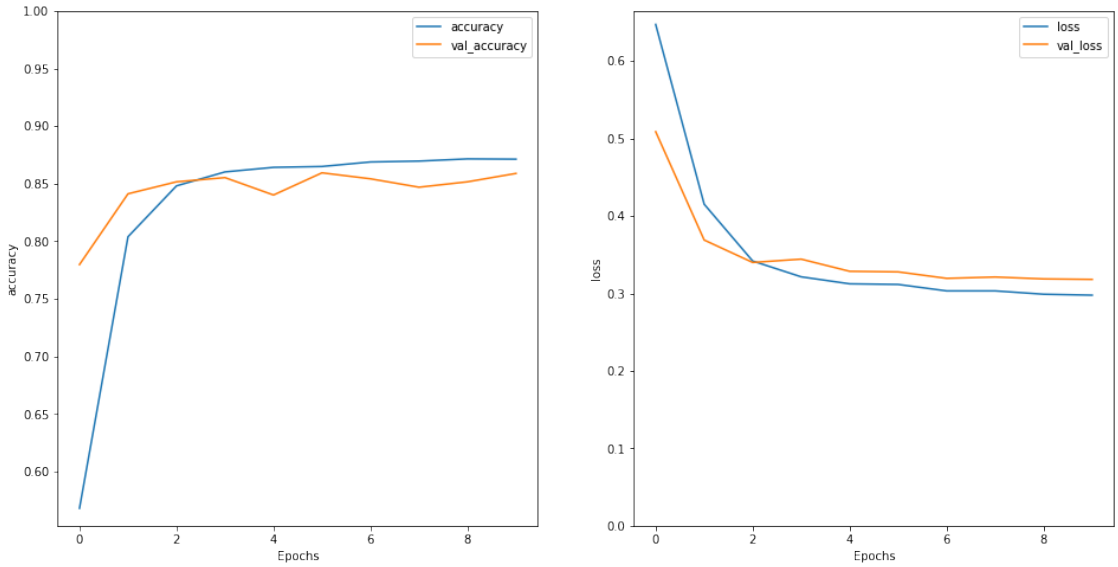
[25]:
```

```

[26]:
```
# These plots only works with the trained model during execution, as shows the␣
 ↪evolution of the accuracy and loss in the differents epochs during training
plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.ylim(None, 1)
```

```
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
plt.ylim(0, None)
```

[26]: (0.0, 0.6646199494600296)



Run a prediction on a new sentence:

If the prediction is >= 0.0, it is positive else it is negative.

[32]:
```
sample_text = ('The movie was the worst. The animation and the graphics '
               'were out of this world. I would not recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions)
```

```
1/1 [==============================] - 0s 19ms/step
[[-1.8472258]]
```

[33]:
```
sample_text = ('The movie was the best. The animation and the graphics '
               'were out of this world. I would highly recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions)
```

```
1/1 [==============================] - 0s 20ms/step
[[2.396447]]
```