tensorflow-rnn

August 25, 2021

1 Setup

Import the libraries and methods required for the project.

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     import tensorflow as tf
     import tensorflow_datasets as tfds
     from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
     from tensorflow.keras import Sequential
     from tensorflow.keras.layers import (
         Embedding,
         Bidirectional,
         LSTM,
         Dense,
         Dropout
     )
     from tensorflow.keras.losses import BinaryCrossentropy
     from tensorflow.keras.optimizers import Adam
```

2 Helper function

Create a helper function to make metric plots.

```
[2]: 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])
```

3 Set up the input pipeline

Download the dataset.

```
[3]: dataset, info = tfds.load(name = "imdb_reviews",
                               as_supervised = True,
                                with_info = True)
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download:
80.23 MiB, generated: Unknown size, total: 80.23 MiB) to
/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0…
```

```
Dl Completed…: 0 url [00:00, ? url/s]
Dl Size…: 0 MiB [00:00, ? MiB/s]



0 examples [00:00, ? examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete3SQTLL/imdb_reviews-train.tfrecord
  0%|          | 0/25000 [00:00<?, ? examples/s]
0 examples [00:00, ? examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete3SQTLL/imdb_reviews-test.tfrecord
  0%|          | 0/25000 [00:00<?, ? examples/s]
0 examples [00:00, ? examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete3SQTLL/imdb_reviews-unsupervised.tfrecord
  0%|          | 0/50000 [00:00<?, ? examples/s]
```

```
Dataset imdb_reviews downloaded and prepared to
/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0. Subsequent calls will
reuse this data.
```

[4]:
```
train = dataset["train"]
test = dataset["test"]
```

[5]:
```
train.element_spec
```

[5]:
```
(TensorSpec(shape=(), dtype=tf.string, name=None),
 TensorSpec(shape=(), dtype=tf.int64, name=None))
```

Inspect the (text, label) pairs obtained.

[6]:
```
for example, label in train.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
```

Shuffle the data for training and create batches of these (text, label) pairs.

[7]:
```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

[8]:
```
train = train.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
test = test.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

[9]:
```
for example, label in train.take(1):
    print("Texts:", example.numpy()[:3])
    print()
    print("Labels:", label.numpy()[:3])
```

```
Texts: [b"I personally watched this to see the footage of the 60's and 70's. It
was fascinating to learn how the drug movement essentially started and became
pop culture and an eventual uncompromising force in life. The interviews of the
classic rock stars are titillating and humorous. You feel like you're in on a
secret and nodding your head at the same time…because it feels so good and
familiar. I loved it, all segments from 60's-present day. I highly recommend
this for all aspects, including rock music, the hipper movement, politics and
good 'ol history. I check marked the box saying this contains a spoiler, only
because I have no idea what some might consider a spoiler or not in this
regards, since I discussed what's in all 4 segments, so just wanted to be safe."
 b"Extremely poor action film starring the ever wooden Dolph Lundgren and
Brandon Lee trapped in a sidekick role that never seems to gel. The action is at
best average, a bit of nudity chucked in and yes Tia Carrera does use a body
double! <br /><br />The set-up is the usual renegade cop forced to break in a
new partner on a big case, the makers at least try to give the formula a twist
making Lundgren the cop with Oriental values and Lee the modern city slicker but
there is zero character development making it almost comical, Lundgrens oriental
warrior outfit for the big showdown has to be seen to be believed. The action
sequences are by the numbers and Lee(who would go on to make the excellent The
Crow) is never given the scope to show off any particular martial arts
brilliance. But given his illustrious parentage he must have been under a hell
of a lot of pressure and was far better served not having to live up to his
father by taking on a very different role in The Crow which showed what a unique
actor he may have become if not for his tragic and early death.<br /><br
/>Unless your a hardcore Lundgren fan or a fan of poor 80's style action movies
(think Cobra etc.) then avoid.<br /><br />Poor 3/10"
 b"This movie is so bad it hurts. The car doing 30 mph when it's supposed to go
100… the shift lever that's stuck (in Park!), the nurse that drives for almost
2 hours with the cell phone on the shoulder…can't any of the 2 morons take
this damn phone? There's nothing credible in this crap. I would be ashamed to be
seen in a movie like this!"]

Labels: [1 0 0]
```

4 Create the text encoder

Use the TextVectorization layer to encode each word in the text to an index. Create the layer and pass the dataset's text to the layer's .adapt() method.

[10]:
```
VOCAB_SIZE = 1000
encoder = TextVectorization(max_tokens = VOCAB_SIZE)
encoder.adapt(train.map(lambda text, label: text))
```

The vocabulary has been set. Print the first 20 tokens of the vocabulary. After the padding and the unknown tokens, the remaining tokens are in decreasing order of frequency.

[11]:
```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

[11]:
```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U14')
```

The tensors of indices are zero-padded to the longest sequence in the batch.

[12]:
```
encoded_example = encoder(example)[:3].numpy()

encoded_example
```

[12]:
```
array([[ 10,   1, 284, …,   0,   0,   0],
       [554, 330, 216, …,   0,   0,   0],
       [ 11,  18,   7, …,   0,   0,   0]])
```

With the default settings, this encoding process is not completely reversible. This is because firstly, the limit on the vocab size leads to the presence of unknown tokens in the vocab which can't be retraced back to their original words. Secondly, text is converted to lower case and stripped of punctuation during tokenization. Hence, upper case characters and punctuation marks cannot be regained.

[13]:
```
for n in range(3):
    print("Original:", example[n].numpy())
```

```python
    print("Round-trip:", " ".join(vocab[encoded_example[n]]))
    print()
```

```
Original: b"I personally watched this to see the footage of the 60's and 70's.
It was fascinating to learn how the drug movement essentially started and became
pop culture and an eventual uncompromising force in life. The interviews of the
classic rock stars are titillating and humorous. You feel like you're in on a
secret and nodding your head at the same time…because it feels so good and
familiar. I loved it, all segments from 60's-present day. I highly recommend
this for all aspects, including rock music, the hipper movement, politics and
good 'ol history. I check marked the box saying this contains a spoiler, only
because I have no idea what some might consider a spoiler or not in this
regards, since I discussed what's in all 4 segments, so just wanted to be safe."
Round-trip: i [UNK] watched this to see the footage of the [UNK] and 70s it was
[UNK] to learn how the [UNK] [UNK] [UNK] started and became [UNK] [UNK] and an
[UNK] [UNK] [UNK] in life the [UNK] of the classic rock stars are [UNK] and
[UNK] you feel like youre in on a secret and [UNK] your head at the same [UNK]
it feels so good and [UNK] i loved it all [UNK] from [UNK] day i highly
recommend this for all [UNK] including rock music the [UNK] [UNK] [UNK] and good
[UNK] history i check [UNK] the [UNK] saying this [UNK] a [UNK] only because i
have no idea what some might [UNK] a [UNK] or not in this [UNK] since i [UNK]
whats in all 4 [UNK] so just wanted to be [UNK]

Original: b"Extremely poor action film starring the ever wooden Dolph Lundgren
and Brandon Lee trapped in a sidekick role that never seems to gel. The action
is at best average, a bit of nudity chucked in and yes Tia Carrera does use a
body double! <br /><br />The set-up is the usual renegade cop forced to break in
a new partner on a big case, the makers at least try to give the formula a twist
making Lundgren the cop with Oriental values and Lee the modern city slicker but
there is zero character development making it almost comical, Lundgrens oriental
warrior outfit for the big showdown has to be seen to be believed. The action
sequences are by the numbers and Lee(who would go on to make the excellent The
Crow) is never given the scope to show off any particular martial arts
brilliance. But given his illustrious parentage he must have been under a hell
of a lot of pressure and was far better served not having to live up to his
father by taking on a very different role in The Crow which showed what a unique
actor he may have become if not for his tragic and early death.<br /><br
/>Unless your a hardcore Lundgren fan or a fan of poor 80's style action movies
(think Cobra etc.) then avoid.<br /><br />Poor 3/10"
Round-trip: extremely poor action film [UNK] the ever [UNK] [UNK] [UNK] and
[UNK] lee [UNK] in a [UNK] role that never seems to [UNK] the action is at best
average a bit of [UNK] [UNK] in and yes [UNK] [UNK] does use a body [UNK] br br
the [UNK] is the usual [UNK] [UNK] forced to [UNK] in a new [UNK] on a big case
the [UNK] at least try to give the [UNK] a twist making [UNK] the [UNK] with
[UNK] [UNK] and lee the modern city [UNK] but there is [UNK] character
development making it almost [UNK] [UNK] [UNK] [UNK] [UNK] for the big [UNK] has
to be seen to be [UNK] the action sequences are by the [UNK] and [UNK] would go
```

```
on to make the excellent the [UNK] is never given the [UNK] to show off any
particular [UNK] [UNK] [UNK] but given his [UNK] [UNK] he must have been under a
hell of a lot of [UNK] and was far better [UNK] not having to live up to his
father by taking on a very different role in the [UNK] which [UNK] what a unique
actor he may have become if not for his [UNK] and early [UNK] br unless your a
[UNK] [UNK] fan or a fan of poor 80s style action movies think [UNK] etc then
[UNK] br poor [UNK]

Original: b"This movie is so bad it hurts. The car doing 30 mph when it's
supposed to go 100… the shift lever that's stuck (in Park!), the nurse that
drives for almost 2 hours with the cell phone on the shoulder…can't any of the
2 morons take this damn phone? There's nothing credible in this crap. I would be
ashamed to be seen in a movie like this!"
Round-trip: this movie is so bad it [UNK] the car doing [UNK] [UNK] when its
supposed to go [UNK] the [UNK] [UNK] thats [UNK] in [UNK] the [UNK] that [UNK]
for almost 2 hours with the [UNK] [UNK] on the [UNK] any of the 2 [UNK] take
this [UNK] [UNK] theres nothing [UNK] in this crap i would be [UNK] to be seen
in a movie like this
```

5 Create the model

Define the model's architecture.

[14]:
```python
model = Sequential([
    encoder,
    Embedding(input_dim = len(encoder.get_vocabulary()),
              output_dim = 64,
              # use masking to handle the variable sequence lengths
              mask_zero = True),
    Bidirectional(layer = LSTM(units = 64,
                               return_sequences = True)),
    Bidirectional(layer = LSTM(units = 32)),
    Dense(units = 64, activation = "relu"),
    Dropout(rate = 0.5),
    Dense(units = 1)
])
```

Each layer after the Embedding layer supports masking.

[15]:
```python
[layer.supports_masking for layer in model.layers]
```

[15]:
```
[False, True, True, True, True, True, True]
```

To confirm the above, evaluate the same sentence twice. First, evaluate it without any padding.

[16]:
```python
# Predict on a sample text without padding
```

```
sample_text = "The movie was cool. The animation and the graphics were out of␣
↪this world. I would recommend this movie."
predictions = model.predict(np.array([sample_text]))

predictions[0]
```

[16]:
```
array([-0.00047284], dtype=float32)
```

Then, evaluate it in a batch with a longer sentence.

[17]:
```
# Predict on a sample text with padding

padding = "the " * 2000
predictions = model.predict(np.array([sample_text, padding]))

predictions[0]
```

[17]:
```
array([-0.00047284], dtype=float32)
```

Observe that the results with and without padding are identical.

Compile the model with a loss function, an optimizer and a metric.

[18]:
```
model.compile(loss = BinaryCrossentropy(from_logits = True),
              optimizer = Adam(learning_rate = 1e-4),
              metrics = ["accuracy"])
```

6 Train the model

Train the model for the desired number of epochs.

[19]:
```
history = model.fit(train,
                    epochs = 30,
                    validation_data = test)
```

```
Epoch 1/30
391/391 [==============================] - 99s 218ms/step - loss: 0.6825 -
accuracy: 0.5184 - val_loss: 0.4255 - val_accuracy: 0.8094
Epoch 2/30
391/391 [==============================] - 81s 207ms/step - loss: 0.4151 -
accuracy: 0.8090 - val_loss: 0.3441 - val_accuracy: 0.8494
Epoch 3/30
391/391 [==============================] - 82s 208ms/step - loss: 0.3476 -
accuracy: 0.8497 - val_loss: 0.3221 - val_accuracy: 0.8535
Epoch 4/30
391/391 [==============================] - 82s 208ms/step - loss: 0.3204 -
accuracy: 0.8639 - val_loss: 0.3176 - val_accuracy: 0.8602
Epoch 5/30
```

```
391/391 [==============================] - 81s 206ms/step - loss: 0.3202 -
accuracy: 0.8628 - val_loss: 0.3123 - val_accuracy: 0.8637
Epoch 6/30
391/391 [==============================] - 82s 208ms/step - loss: 0.3066 -
accuracy: 0.8679 - val_loss: 0.3149 - val_accuracy: 0.8558
Epoch 7/30
391/391 [==============================] - 82s 208ms/step - loss: 0.3014 -
accuracy: 0.8706 - val_loss: 0.3336 - val_accuracy: 0.8619
Epoch 8/30
391/391 [==============================] - 81s 207ms/step - loss: 0.2977 -
accuracy: 0.8734 - val_loss: 0.3193 - val_accuracy: 0.8492
Epoch 9/30
391/391 [==============================] - 81s 206ms/step - loss: 0.2982 -
accuracy: 0.8698 - val_loss: 0.3157 - val_accuracy: 0.8573
Epoch 10/30
391/391 [==============================] - 81s 206ms/step - loss: 0.2970 -
accuracy: 0.8706 - val_loss: 0.3163 - val_accuracy: 0.8596
Epoch 11/30
391/391 [==============================] - 82s 207ms/step - loss: 0.2924 -
accuracy: 0.8733 - val_loss: 0.3195 - val_accuracy: 0.8480
Epoch 12/30
391/391 [==============================] - 81s 207ms/step - loss: 0.2892 -
accuracy: 0.8731 - val_loss: 0.3181 - val_accuracy: 0.8622
Epoch 13/30
391/391 [==============================] - 81s 206ms/step - loss: 0.2859 -
accuracy: 0.8801 - val_loss: 0.3278 - val_accuracy: 0.8602
Epoch 14/30
391/391 [==============================] - 81s 205ms/step - loss: 0.2866 -
accuracy: 0.8754 - val_loss: 0.3249 - val_accuracy: 0.8608
Epoch 15/30
391/391 [==============================] - 81s 206ms/step - loss: 0.2815 -
accuracy: 0.8778 - val_loss: 0.3225 - val_accuracy: 0.8506
Epoch 16/30
391/391 [==============================] - 81s 206ms/step - loss: 0.2755 -
accuracy: 0.8820 - val_loss: 0.3251 - val_accuracy: 0.8462
Epoch 17/30
391/391 [==============================] - 79s 202ms/step - loss: 0.2816 -
accuracy: 0.8770 - val_loss: 0.3221 - val_accuracy: 0.8560
Epoch 18/30
391/391 [==============================] - 79s 201ms/step - loss: 0.2762 -
accuracy: 0.8809 - val_loss: 0.3392 - val_accuracy: 0.8551
Epoch 19/30
391/391 [==============================] - 79s 200ms/step - loss: 0.2744 -
accuracy: 0.8804 - val_loss: 0.3339 - val_accuracy: 0.8451
Epoch 20/30
391/391 [==============================] - 81s 207ms/step - loss: 0.2715 -
accuracy: 0.8823 - val_loss: 0.3378 - val_accuracy: 0.8593
Epoch 21/30
```

```
391/391 [==============================] - 81s 207ms/step - loss: 0.2670 -
accuracy: 0.8864 - val_loss: 0.3322 - val_accuracy: 0.8484
Epoch 22/30
391/391 [==============================] - 82s 208ms/step - loss: 0.2633 -
accuracy: 0.8846 - val_loss: 0.3334 - val_accuracy: 0.8544
Epoch 23/30
391/391 [==============================] - 81s 207ms/step - loss: 0.2633 -
accuracy: 0.8840 - val_loss: 0.3377 - val_accuracy: 0.8551
Epoch 24/30
391/391 [==============================] - 81s 206ms/step - loss: 0.2613 -
accuracy: 0.8876 - val_loss: 0.3411 - val_accuracy: 0.8556
Epoch 25/30
391/391 [==============================] - 81s 207ms/step - loss: 0.2554 -
accuracy: 0.8899 - val_loss: 0.3397 - val_accuracy: 0.8514
Epoch 26/30
391/391 [==============================] - 82s 208ms/step - loss: 0.2539 -
accuracy: 0.8890 - val_loss: 0.3495 - val_accuracy: 0.8538
Epoch 27/30
391/391 [==============================] - 82s 207ms/step - loss: 0.2519 -
accuracy: 0.8907 - val_loss: 0.3567 - val_accuracy: 0.8417
Epoch 28/30
391/391 [==============================] - 82s 207ms/step - loss: 0.2475 -
accuracy: 0.8910 - val_loss: 0.3549 - val_accuracy: 0.8482
Epoch 29/30
391/391 [==============================] - 82s 207ms/step - loss: 0.2436 -
accuracy: 0.8973 - val_loss: 0.3531 - val_accuracy: 0.8537
Epoch 30/30
391/391 [==============================] - 82s 208ms/step - loss: 0.2402 -
accuracy: 0.8965 - val_loss: 0.3540 - val_accuracy: 0.8516
```

Evaluate the model on the test set.

[20]:
```
test_loss, test_accuracy = model.evaluate(test)

print("Test loss:", test_loss)
print("Test accuracy:", test_accuracy)
```

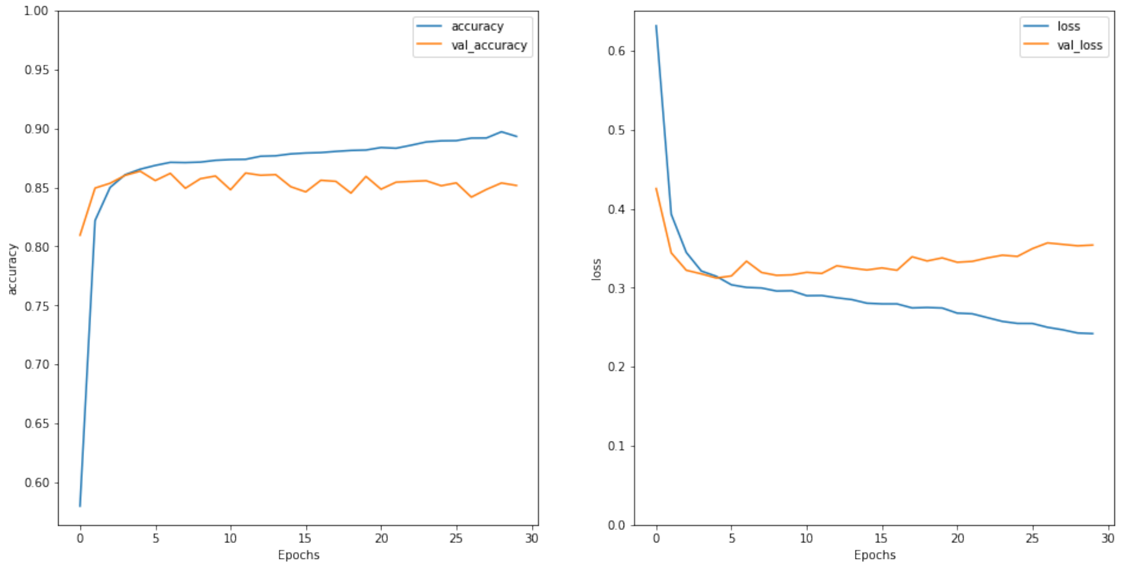
```
391/391 [==============================] - 26s 66ms/step - loss: 0.3540 -
accuracy: 0.8516
Test loss: 0.3539840579032898
Test accuracy: 0.8515999913215637
```

Plot the metrics for the training process.

[21]:
```
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)
```

```
[21]: (0.0, 0.6509137965738774)
```



7 Make predictions on new data

Come up with a new review of your own and let the model figure out whether it's positive or negative.

```
[22]: sample_text = "The movie was cool. The animation and the graphics were out of␣
       ↪this world. I would recommend this movie."
      predictions = model.predict(np.array([sample_text]))

      predictions[0]
```

```
[22]: array([-0.27691582], dtype=float32)
```

Save the model to disk.

```
[23]: model.save("tensorflow-model")
```