p6

November 7, 2024

[1]:

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout

# Load dataset
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocabulary_size)

print('Loaded dataset with {} training samples, {} test samples'.
 ↪format(len(X_train), len(X_test)))

# Inspect a sample review and its label
print('---review---')
print(X_train[6])
print('---label---')
print(y_train[6])

# Map review back to original words
word2id = imdb.get_word_index()
id2word = {i: word for word, i in word2id.items()}

print('---review with words---')
print([id2word.get(i, ' ') for i in X_train[6]])
print('---label---')
print(y_train[6])

# Maximum and minimum review lengths
print('Maximum review length: {}'.format(len(max((X_train + X_test), key=len))))
print('Minimum review length: {}'.format(len(min((X_train + X_test), key=len))))

# Pad sequences
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)

# Design RNN model
```

```
model = Sequential()
embedding_size = 32
model.add(Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))

# Model summary
print(model.summary())

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

# Train model
batch_size = 64
num_epochs = 3
X_valid, y_valid = X_train[:batch_size], y_train[:batch_size]
X_train2, y_train2 = X_train[batch_size:], y_train[batch_size:]

model.fit(X_train2, y_train2, validation_data=(X_valid, y_valid),␣
 ↪batch_size=batch_size, epochs=num_epochs)

# Evaluate model
scores = model.evaluate(X_test, y_test, verbose=0)
print('Test accuracy:', scores[1])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17464789/17464789            0s
0us/step
Loaded dataset with 25000 training samples, 25000 test samples
---review---
[1, 2, 365, 1234, 5, 1156, 354, 11, 14, 2, 2, 7, 1016, 2, 2, 356, 44, 4, 1349,
500, 746, 5, 200, 4, 4132, 11, 2, 2, 1117, 1831, 2, 5, 4831, 26, 6, 2, 4183, 17,
369, 37, 215, 1345, 143, 2, 5, 1838, 8, 1974, 15, 36, 119, 257, 85, 52, 486, 9,
6, 2, 2, 63, 271, 6, 196, 96, 949, 4121, 4, 2, 7, 4, 2212, 2436, 819, 63, 47,
77, 2, 180, 6, 227, 11, 94, 2494, 2, 13, 423, 4, 168, 7, 4, 22, 5, 89, 665, 71,
270, 56, 5, 13, 197, 12, 161, 2, 99, 76, 23, 2, 7, 419, 665, 40, 91, 85, 108, 7,
4, 2084, 5, 4773, 81, 55, 52, 1901]
---label---
1
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb_word_index.json
1641221/1641221            0s
0us/step
---review with words---
['the', 'and', 'full', 'involving', 'to', 'impressive', 'boring', 'this', 'as',
'and', 'and', 'br', 'villain', 'and', 'and', 'need', 'has', 'of', 'costumes',
'b', 'message', 'to', 'may', 'of', 'props', 'this', 'and', 'and', 'concept',
'issue', 'and', 'to', "god's", 'he', 'is', 'and', 'unfolds', 'movie', 'women',
'like', "isn't", 'surely', "i'm", 'and', 'to', 'toward', 'in', "here's", 'for',
'from', 'did', 'having', 'because', 'very', 'quality', 'it', 'is', 'and', 'and',
'really', 'book', 'is', 'both', 'too', 'worked', 'carl', 'of', 'and', 'br',
'of', 'reviewer', 'closer', 'figure', 'really', 'there', 'will', 'and',
'things', 'is', 'far', 'this', 'make', 'mistakes', 'and', 'was', "couldn't",
'of', 'few', 'br', 'of', 'you', 'to', "don't", 'female', 'than', 'place', 'she',
'to', 'was', 'between', 'that', 'nothing', 'and', 'movies', 'get', 'are', 'and',
'br', 'yes', 'female', 'just', 'its', 'because', 'many', 'br', 'of', 'overly',
'to', 'descent', 'people', 'time', 'very', 'bland']
---label---
1
Maximum review length: 2697
Minimum review length: 70
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Model: "sequential"


 Layer (type)                          Output Shape                         ␣
↪Param #

 embedding (Embedding)                 ?                                  0␣
↪(unbuilt)

 lstm (LSTM)                           ?                                  0␣
↪(unbuilt)

 dense (Dense)                         ?                                  0␣
↪(unbuilt)


 Total params: 0 (0.00 B)


 Trainable params: 0 (0.00 B)


 Non-trainable params: 0 (0.00 B)


None
Epoch 1/3
 14/390             4:56 789ms/step -
```

```
accuracy: 0.5248 - loss: 0.6928
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-1-3dc1473a222c> in <cell line: 55>()
     53 X_train2, y_train2 = X_train[batch_size:], y_train[batch_size:]
     54 
---> 55 model.fit(X_train2, y_train2, validation_data=(X_valid, y_valid),␣
 ↪batch_size=batch_size, epochs=num_epochs)
     56 
     57 # Evaluate model

/usr/local/lib/python3.10/dist-packages/keras/src/utils/traceback_utils.py in␣
 ↪error_handler(*args, **kwargs)
    115         filtered_tb = None
    116         try:
--> 117             return fn(*args, **kwargs)
    118         except Exception as e:
    119             filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/trainer.py␣
 ↪in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split,␣
 ↪validation_data, shuffle, class_weight, sample_weight, initial_epoch,␣
 ↪steps_per_epoch, validation_steps, validation_batch_size, validation_freq)
    316                 for step, iterator in epoch_iterator.enumerate_epoch():
    317                     callbacks.on_train_batch_begin(step)
--> 318                     logs = self.train_function(iterator)
    319                     logs = self._pythonify_logs(logs)
    320                     callbacks.on_train_batch_end(step, logs)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/util/traceback_utils.
 ↪py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py in __call__(self, *args, **kwds)
    831 
    832       with OptionalXlaContext(self._jit_compile):
--> 833         result = self._call(*args, **kwds)
    834 
    835       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py in _call(self, *args, **kwds)
```

```
    876       # In this case we have not created variables on the first call. S␣
 ↪we can
    877       # run the first trace but we should fail if variables are created
--> 878       results = tracing_compilation.call_function(

    879           args, kwds, self._variable_creation_config
    880       )

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/tracing_compilation.py in call_function(args, kwargs,␣
 ↪tracing_options)
    137   bound_args = function.function_type.bind(*args, **kwargs)
    138   flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139   return function._call_flat(  # pylint: disable=protected-access

    140       flat_inputs, captured_inputs=function.captured_inputs
    141   )

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/concrete_function.py in _call_flat(self, tensor_inputs,␣
 ↪captured_inputs)
   1320         and executing_eagerly):
   1321       # No tape is watching; skip to running the function.
-> 1322       return self._inference_function.call_preflattened(args)
   1323     forward_backward = self._select_forward_and_backward_functions(
   1324         args,

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py in call_preflattened(self, args)
    214   def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215     """Calls with flattened tensor inputs and returns the structured␣
 ↪output."""
--> 216     flat_outputs = self.call_flat(*args)
    217     return self.function_type.pack_output(flat_outputs)
    218

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py in call_flat(self, *args)
    249         with record.stop_recording():
    250           if self._bound_context.executing_eagerly():
--> 251             outputs = self._bound_context.call_function(

    252                 self.name,
    253                 list(args),

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/context.py in␣
 ↪call_function(self, name, tensor_inputs, num_outputs)
   1550     cancellation_context = cancellation.context()
   1551     if cancellation_context is None:
```

```
-> 1552         outputs = execute.execute(

   1553             name.decode("utf-8"),
   1554             num_outputs=num_outputs,

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in␣
 ↪quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,␣
 ↪op_name,
     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt:
```