hw4_part5

November 23, 2023

[1]:
```
# set-up train valid test to

import os, pathlib, shutil, random
from tensorflow import keras
batch_size = 32
base_dir = pathlib.Path("aclImdb")
val_dir = base_dir / "val"
train_dir = base_dir / "train"
for category in ("neg", "pos"):
    os.makedirs(val_dir / category)
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples = int(0.4 * len(files))
    val_files = files[-num_val_samples:]
    for fname in val_files:
        shutil.move(train_dir / category / fname,
                    val_dir / category / fname)


# I have no idea about how to keep the training set to a specific number
# so I did it in a very straight non-fancy way
# previous block(code) has transfer 20% training data to valid data (25000*0.2)

# so here I only need to keep the neg and pos folder under train folder to 50␣
 ↪obs randomly.

#      # path of folder
folder_path = r'C:\Users\zhong\Desktop\HW4_part5\aclImdb\train\neg'

#        # go through all file in the folder
all_files = os.listdir(folder_path)

#         # set up the number of file that I want to keep
num_files_to_keep = 300

#         # randomly keep the file
files_to_keep = random.sample(all_files, num_files_to_keep)
```

```
#       # delete the file that I dont need.
for file_name in all_files:
    file_path = os.path.join(folder_path, file_name)
    if file_name not in files_to_keep:
        os.remove(file_path)

# Repeat the same thing for pos folder.

#
folder_path = r'C:\Users\zhong\Desktop\HW4_part5\aclImdb\train\pos'

#
all_files = os.listdir(folder_path)

#
num_files_to_keep = 300

#
files_to_keep = random.sample(all_files, num_files_to_keep)

#
for file_name in all_files:
    file_path = os.path.join(folder_path, file_name)
    if file_name not in files_to_keep:
        os.remove(file_path)


# show the number of train valid test
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=batch_size
)
val_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/val", batch_size=batch_size
)
test_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=batch_size
)
text_only_train_ds = train_ds.map(lambda x, y: x)
```

```
Found 600 files belonging to 2 classes.
Found 10000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

[3]:
```
# Preparing integer sequence datasets

from tensorflow.keras import layers
```

```
max_length = 150
max_tokens = 10000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

[4]:
```
# A sequence model built on one-hot encoded vector sequences

import tensorflow as tf
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
# RNN-LSTM
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None)]            0

 tf.one_hot_1 (TFOpLambda)   (None, None, 10000)       0

 bidirectional_1 (Bidirectio  (None, 64)               2568448
 nal)

 dropout_1 (Dropout)         (None, 64)                0
```

```
 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,568,513
Trainable params: 2,568,513
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
# Training a first basic sequence model     1m22s
# Test acc: 0.704
```

[5]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint("one_hot_bidir_lstm.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,
          callbacks=callbacks)
model = keras.models.load_model("one_hot_bidir_lstm.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Epoch 1/10
19/19 [==============================] - 10s 428ms/step - loss: 0.6928 -
accuracy: 0.5217 - val_loss: 0.6925 - val_accuracy: 0.5497
Epoch 2/10
19/19 [==============================] - 4s 239ms/step - loss: 0.6864 -
accuracy: 0.6117 - val_loss: 0.6903 - val_accuracy: 0.5142
Epoch 3/10
19/19 [==============================] - 4s 238ms/step - loss: 0.6611 -
accuracy: 0.7700 - val_loss: 0.6807 - val_accuracy: 0.5929
Epoch 4/10
19/19 [==============================] - 4s 236ms/step - loss: 0.5596 -
accuracy: 0.7700 - val_loss: 0.6168 - val_accuracy: 0.6567
Epoch 5/10
19/19 [==============================] - 4s 235ms/step - loss: 0.4405 -
accuracy: 0.8500 - val_loss: 0.7401 - val_accuracy: 0.6278
Epoch 6/10
19/19 [==============================] - 4s 235ms/step - loss: 0.3005 -
accuracy: 0.9133 - val_loss: 0.5876 - val_accuracy: 0.7016
Epoch 7/10
19/19 [==============================] - 4s 235ms/step - loss: 0.2216 -
accuracy: 0.9283 - val_loss: 1.3416 - val_accuracy: 0.5455
Epoch 8/10
19/19 [==============================] - 4s 234ms/step - loss: 0.2332 -
accuracy: 0.9250 - val_loss: 0.6312 - val_accuracy: 0.7165
Epoch 9/10
19/19 [==============================] - 4s 234ms/step - loss: 0.1103 -
accuracy: 0.9700 - val_loss: 0.7277 - val_accuracy: 0.7084
Epoch 10/10
```

```
19/19 [==============================] - 4s 235ms/step - loss: 0.1373 -
accuracy: 0.9867 - val_loss: 0.6871 - val_accuracy: 0.7206
782/782 [==============================] - 33s 41ms/step - loss: 0.5885 -
accuracy: 0.7036
Test acc: 0.704
```

[6]:
```
# Learning word embeddings with the Embedding layer
# Instantiating an Embedding layer
embedding_layer = layers.Embedding(input_dim=max_tokens, output_dim=256)
```

[ ]:
```
# Model that uses an Embedding layer trained from scratch, 37.5s
# Test acc:  0.718,  improved a little
# embedding is useful when you have a large size of training
```

[7]:
```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
model = keras.models.load_model("embeddings_bidir_gru.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_1 (Embedding)     (None, None, 256)         2560000

 bidirectional_2 (Bidirectio  (None, 64)               73984
 nal)

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65
```

```
=================================================================
Total params: 2,634,049
Trainable params: 2,634,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
19/19 [==============================] - 5s 174ms/step - loss: 0.6924 -
accuracy: 0.5417 - val_loss: 0.6912 - val_accuracy: 0.5391
Epoch 2/10
19/19 [==============================] - 3s 152ms/step - loss: 0.6586 -
accuracy: 0.7383 - val_loss: 0.6518 - val_accuracy: 0.6327
Epoch 3/10
19/19 [==============================] - 3s 151ms/step - loss: 0.4832 -
accuracy: 0.8333 - val_loss: 0.6203 - val_accuracy: 0.6678
Epoch 4/10
19/19 [==============================] - 3s 149ms/step - loss: 0.2613 -
accuracy: 0.9517 - val_loss: 0.5567 - val_accuracy: 0.7168
Epoch 5/10
19/19 [==============================] - 3s 147ms/step - loss: 0.1261 -
accuracy: 0.9850 - val_loss: 0.7610 - val_accuracy: 0.6816
Epoch 6/10
19/19 [==============================] - 3s 146ms/step - loss: 0.0493 -
accuracy: 0.9933 - val_loss: 0.9884 - val_accuracy: 0.6560
Epoch 7/10
19/19 [==============================] - 3s 149ms/step - loss: 0.0413 -
accuracy: 0.9967 - val_loss: 0.7608 - val_accuracy: 0.7176
Epoch 8/10
19/19 [==============================] - 3s 150ms/step - loss: 0.0289 -
accuracy: 0.9983 - val_loss: 0.7435 - val_accuracy: 0.6951
Epoch 9/10
19/19 [==============================] - 3s 147ms/step - loss: 0.0201 -
accuracy: 0.9983 - val_loss: 0.7072 - val_accuracy: 0.6920
Epoch 10/10
19/19 [==============================] - 3s 148ms/step - loss: 0.0095 -
accuracy: 1.0000 - val_loss: 0.8769 - val_accuracy: 0.7179
782/782 [==============================] - 7s 8ms/step - loss: 0.5627 -
accuracy: 0.7178
Test acc: 0.718
```

[ ]:
```
# Using an Embedding layer with masking enabled, 1m49s
# Test acc: 0.716 same-flat
# mask is not that useful for this case.
```

[8]:
```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(
    input_dim=max_tokens, output_dim=256, mask_zero=True)(inputs)
```

```
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru_with_masking.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
model = keras.models.load_model("embeddings_bidir_gru_with_masking.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_2 (Embedding)     (None, None, 256)         2560000

 bidirectional_3 (Bidirectio  (None, 64)               73984
 nal)

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,634,049
Trainable params: 2,634,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
19/19 [==============================] - 9s 259ms/step - loss: 0.6927 -
accuracy: 0.5300 - val_loss: 0.6886 - val_accuracy: 0.5595
Epoch 2/10
19/19 [==============================] - 3s 179ms/step - loss: 0.6157 -
accuracy: 0.7950 - val_loss: 0.6234 - val_accuracy: 0.6742
Epoch 3/10
19/19 [==============================] - 3s 172ms/step - loss: 0.4051 -
accuracy: 0.8683 - val_loss: 0.6520 - val_accuracy: 0.6890
Epoch 4/10
```

```
19/19 [==============================] - 3s 177ms/step - loss: 0.1139 -
accuracy: 0.9817 - val_loss: 0.6230 - val_accuracy: 0.7237
Epoch 5/10
19/19 [==============================] - 3s 179ms/step - loss: 0.0527 -
accuracy: 0.9917 - val_loss: 0.7655 - val_accuracy: 0.7199
Epoch 6/10
19/19 [==============================] - 3s 186ms/step - loss: 0.0208 -
accuracy: 0.9983 - val_loss: 0.9429 - val_accuracy: 0.7016
Epoch 7/10
19/19 [==============================] - 3s 186ms/step - loss: 0.0128 -
accuracy: 0.9983 - val_loss: 1.0571 - val_accuracy: 0.6894
Epoch 8/10
19/19 [==============================] - 3s 179ms/step - loss: 0.0426 -
accuracy: 0.9817 - val_loss: 0.9275 - val_accuracy: 0.7149
Epoch 9/10
19/19 [==============================] - 3s 179ms/step - loss: 0.0084 -
accuracy: 0.9983 - val_loss: 0.8543 - val_accuracy: 0.7378
Epoch 10/10
19/19 [==============================] - 3s 179ms/step - loss: 0.0041 -
accuracy: 1.0000 - val_loss: 1.0246 - val_accuracy: 0.7283
782/782 [==============================] - 9s 9ms/step - loss: 0.6434 -
accuracy: 0.7158
Test acc: 0.716
```

[ ]:
```python
# pretrained model
```

[9]:
```python
# Parsing the GloVe word-embeddings file
import numpy as np
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print(f"Found {len(embeddings_index)} word vectors.")
```

```
Found 400000 word vectors.
```

[10]:
```python
# Preparing the GloVe word-embeddings matrix, Os

embedding_dim = 100

vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))
```

```
embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

[11]:
```
embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
```

[ ]:
```
# Model that uses a pretrained Embedding layer,  55.6s
# Test acc: 0.645

###################need to increase the training size to find a "breakeven␣
 ↪point" for this
```

[12]:
```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
model = keras.models.load_model("glove_embeddings_sequence_model.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 100)         1000000
```

```
 bidirectional_4 (Bidirectio  (None, 64)              34048
 nal)

 dropout_4 (Dropout)         (None, 64)              0

 dense_4 (Dense)             (None, 1)               65

=================================================================
Total params: 1,034,113
Trainable params: 34,113
Non-trainable params: 1,000,000
_________________________________________________________________
Epoch 1/10
19/19 [==============================] - 10s 297ms/step - loss: 0.7138 -
accuracy: 0.4917 - val_loss: 0.6839 - val_accuracy: 0.5608
Epoch 2/10
19/19 [==============================] - 4s 217ms/step - loss: 0.6853 -
accuracy: 0.6017 - val_loss: 0.6772 - val_accuracy: 0.5869
Epoch 3/10
19/19 [==============================] - 3s 187ms/step - loss: 0.6785 -
accuracy: 0.5817 - val_loss: 0.7025 - val_accuracy: 0.5209
Epoch 4/10
19/19 [==============================] - 3s 182ms/step - loss: 0.6703 -
accuracy: 0.5883 - val_loss: 0.6925 - val_accuracy: 0.5426
Epoch 5/10
19/19 [==============================] - 4s 214ms/step - loss: 0.6453 -
accuracy: 0.6400 - val_loss: 0.6643 - val_accuracy: 0.6011
Epoch 6/10
19/19 [==============================] - 4s 215ms/step - loss: 0.6268 -
accuracy: 0.6833 - val_loss: 0.6436 - val_accuracy: 0.6321
Epoch 7/10
19/19 [==============================] - 3s 187ms/step - loss: 0.6040 -
accuracy: 0.6683 - val_loss: 0.6630 - val_accuracy: 0.6081
Epoch 8/10
19/19 [==============================] - 4s 234ms/step - loss: 0.5922 -
accuracy: 0.6717 - val_loss: 0.6219 - val_accuracy: 0.6604
Epoch 9/10
19/19 [==============================] - 3s 183ms/step - loss: 0.5917 -
accuracy: 0.6883 - val_loss: 0.6363 - val_accuracy: 0.6394
Epoch 10/10
19/19 [==============================] - 3s 185ms/step - loss: 0.5274 -
accuracy: 0.7500 - val_loss: 0.6720 - val_accuracy: 0.6312
782/782 [==============================] - 10s 10ms/step - loss: 0.6312 -
accuracy: 0.6454
Test acc: 0.645
```

[ ]:
```
# try glove6b300d
```

[13]:
```
# Parsing the GloVe word-embeddings file
import numpy as np
path_to_glove_file = "glove.6B.300d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print(f"Found {len(embeddings_index)} word vectors.")
```

```
Found 400000 word vectors.
```

[15]:
```
# Preparing the GloVe word-embeddings matrix, 0s

embedding_dim = 300

vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))

embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

[16]:
```
embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
```

[ ]:
```
# Model that uses a pretrained Embedding layer,  1m4.4s
# Test acc: 0.715

###################need to increase the training size to find a "breakeven␣
 ↪point" for this
```

[17]:
```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
```

```
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
model = keras.models.load_model("glove_embeddings_sequence_model.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, None)]            0

 embedding_4 (Embedding)     (None, None, 300)         3000000

 bidirectional_5 (Bidirectio  (None, 64)               85248
 nal)

 dropout_5 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 3,085,313
Trainable params: 85,313
Non-trainable params: 3,000,000
_________________________________________________________________
Epoch 1/10
19/19 [==============================] - 12s 394ms/step - loss: 0.7039 -
accuracy: 0.4867 - val_loss: 0.6856 - val_accuracy: 0.5371
Epoch 2/10
19/19 [==============================] - 5s 292ms/step - loss: 0.6624 -
accuracy: 0.6050 - val_loss: 0.6639 - val_accuracy: 0.6082
Epoch 3/10
19/19 [==============================] - 5s 288ms/step - loss: 0.6229 -
accuracy: 0.6600 - val_loss: 0.6427 - val_accuracy: 0.6327
Epoch 4/10
19/19 [==============================] - 5s 287ms/step - loss: 0.5748 -
```

```
accuracy: 0.6983 - val_loss: 0.6230 - val_accuracy: 0.6444
Epoch 5/10
19/19 [==============================] - 4s 193ms/step - loss: 0.5270 -
accuracy: 0.7267 - val_loss: 0.6454 - val_accuracy: 0.6390
Epoch 6/10
19/19 [==============================] - 5s 280ms/step - loss: 0.5067 -
accuracy: 0.7633 - val_loss: 0.5930 - val_accuracy: 0.6903
Epoch 7/10
19/19 [==============================] - 5s 284ms/step - loss: 0.4500 -
accuracy: 0.8033 - val_loss: 0.5605 - val_accuracy: 0.7131
Epoch 8/10
19/19 [==============================] - 3s 190ms/step - loss: 0.4372 -
accuracy: 0.8200 - val_loss: 0.5819 - val_accuracy: 0.7102
Epoch 9/10
19/19 [==============================] - 3s 184ms/step - loss: 0.3894 -
accuracy: 0.8367 - val_loss: 0.5980 - val_accuracy: 0.6909
Epoch 10/10
19/19 [==============================] - 3s 186ms/step - loss: 0.4062 -
accuracy: 0.8133 - val_loss: 0.5638 - val_accuracy: 0.7092
782/782 [==============================] - 9s 10ms/step - loss: 0.5601 -
accuracy: 0.7146
Test acc: 0.715
```

[18]:
```python
# try glove6b200d
# Parsing the GloVe word-embeddings file
import numpy as np
path_to_glove_file = "glove.6B.200d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print(f"Found {len(embeddings_index)} word vectors.")

# Preparing the GloVe word-embeddings matrix, 0s

embedding_dim = 200

vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))

embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
```

```
        embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
```

```
Found 400000 word vectors.
```

[ ]:
```
# Model that uses a pretrained Embedding layer,  57,7s
# Test acc: 0.683

###################need to increase the training size to find a "breakeven␣
 ↪point" for this
```

[19]:
```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
model = keras.models.load_model("glove_embeddings_sequence_model.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, None)]            0

 embedding_5 (Embedding)     (None, None, 200)         2000000
```

```
 bidirectional_6 (Bidirectio  (None, 64)              59648
 nal)

 dropout_6 (Dropout)         (None, 64)              0

 dense_6 (Dense)             (None, 1)               65

=================================================================
Total params: 2,059,713
Trainable params: 59,713
Non-trainable params: 2,000,000
_________________________________________________________________
Epoch 1/10
19/19 [==============================] - 11s 332ms/step - loss: 0.6963 -
accuracy: 0.5350 - val_loss: 0.6814 - val_accuracy: 0.5705
Epoch 2/10
19/19 [==============================] - 5s 249ms/step - loss: 0.6677 -
accuracy: 0.5900 - val_loss: 0.6712 - val_accuracy: 0.5966
Epoch 3/10
19/19 [==============================] - 5s 250ms/step - loss: 0.6560 -
accuracy: 0.6150 - val_loss: 0.6586 - val_accuracy: 0.6220
Epoch 4/10
19/19 [==============================] - 3s 187ms/step - loss: 0.6302 -
accuracy: 0.6700 - val_loss: 0.6737 - val_accuracy: 0.5696
Epoch 5/10
19/19 [==============================] - 3s 180ms/step - loss: 0.5961 -
accuracy: 0.6750 - val_loss: 0.7103 - val_accuracy: 0.5591
Epoch 6/10
19/19 [==============================] - 4s 244ms/step - loss: 0.5748 -
accuracy: 0.7100 - val_loss: 0.6275 - val_accuracy: 0.6518
Epoch 7/10
19/19 [==============================] - 4s 244ms/step - loss: 0.5224 -
accuracy: 0.7317 - val_loss: 0.5839 - val_accuracy: 0.6940
Epoch 8/10
19/19 [==============================] - 3s 184ms/step - loss: 0.5239 -
accuracy: 0.7533 - val_loss: 0.5865 - val_accuracy: 0.6865
Epoch 9/10
19/19 [==============================] - 3s 182ms/step - loss: 0.4789 -
accuracy: 0.7833 - val_loss: 0.6129 - val_accuracy: 0.6547
Epoch 10/10
19/19 [==============================] - 3s 186ms/step - loss: 0.4544 -
accuracy: 0.7917 - val_loss: 0.7027 - val_accuracy: 0.6352
782/782 [==============================] - 9s 10ms/step - loss: 0.5926 -
accuracy: 0.6834
Test acc: 0.683
```