Assignment 4: Text and Sequence

Atshaya Suresh, Anusha Banda

Questions:

1. Cutoff reviews after 150 words.
2. Restrict training samples to 100.
3. Validate on 10,000 samples.
4. Consider only the top 10,000 words.
5. Consider both an embedding layer, and a pretrained word embedding. Which approach
did better? Now try changing the number of training samples to determine at what point the embedding layer gives better performance.

Answer:

We started with downloading the file from Stanford page of Andrew Maas. The train/pos/ directory contains 12500 text files which are related to the positive IMDB reviews. The train/neg/ directory contains 12500 text files which are related to negative comments. As the first step since we have the standardized data, we did one-hot encoding of the text to create vectors. During the splitting of the data for training, validation, and testing- we took 100 samples for training, 10000 samples for validation and originally 25000 for testing.

First, we used the bag-of-words approach. In that, we cut off reviews after 150 words and considered only the top 10000 words. We trained and tested the binary unigram model. We call cache() on the datasets to cache them in memory: this way, we will only do the preprocessing once, during the first epoch, and we will reuse the preprocessed texts for the following epochs. This can only be done if the data is small enough to fit in memory. We ran 10 Epochs of the model. In the 10^{th} Epoch we got a training loss of 0.0790, accuracy of 0.975, validation loss of 0.5222 and accuracy of 0.8476. However, the model started overfitting in the 4^{th} Epoch itself where the validation loss is 0.4159 in the 3^{rd} Epoch and in the 4^{th} Epoch it is, 1.6233. So, in the 3^{rd} Epoch, the Validation Accuracy is 0.8206 which can be considered optimal. On the Test Accuracy we reached with this model, it is 0.7835 which is not so great. So, we are using an Embedding Layer that is trained from scratch using the data we have in

hand along with the actual task of classification of reviews. For this, we used 10000 samples for training, 12500 for Validation and 12500 for testing. The Embedding layer takes at least two arguments: the number of possible tokens and the dimensionality of the embeddings (here, 256). The Embedding layer is best understood as a dictionary that maps integer indices (which stand for specific words) to dense vectors. It takes integers as input, looks up these integers in an internal dictionary, and returns the associated vectors. It is effectively a dictionary lookup. It trains much faster than the one-hot model (since the LSTM only has to process 256-dimensional vectors instead of 20,000-dimensional), and its test accuracy is comparable (0.853).

Next, we built a model using a pretrained Word Embedding layer. First, we downloaded the GloVe word embeddings precomputed on the 2014 English Wikipedia dataset. This contains 400,000-word Vectors. Next, we prepared an embedding matrix that we can load into an Embedding layer. The advantage of a Pretrained Model like in Convolutional Networks is that we are leveraging the 100-dimensional pretrained GloVe embeddings instead of 128-dimensional learned embeddings we produced from our data. Using a Pretrained model improved our accuracy to 0.875. Since we have an exceptionally large dataset, leveraging a pretrained model did not create any massive impact though the accuracy improved a little. When we use a dataset that is not exhaustive and is small, leveraging a pretrained model is strongly recommended.

Assignment 4

May 5, 2024

0.0.1 Processing words as a sequence: The sequence model approach

Downloading the data

[1]:
```
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  16.4M       0  0:00:04  0:00:04 --:--:-- 19.7M
```

Preparing the data

[2]:
```
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"):
    val_category_dir = val_dir / category
    if not os.path.exists(val_category_dir):  # Check if directory exists
        os.makedirs(val_category_dir)  # Create directory if it doesn't exist
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples = 10000  # Use 10,000 validation samples
    val_files = files[-num_val_samples:]
    for fname in val_files:
        shutil.move(train_dir / category / fname,
                    val_category_dir / fname)

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)

from tensorflow.keras import layers

max_length = 150  # Cutoff reviews after 150 words
max_tokens = 10000  # Consider only the top 10,000 words
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)

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()

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}")
```

```
Found 5000 files belonging to 2 classes.
```

```
Found 25000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None)]            0
_________________________________________________________________
tf.one_hot (TFOpLambda)      (None, None, 10000)       0
_________________________________________________________________
bidirectional (Bidirectional (None, 64)                2568448
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 2,568,513
Trainable params: 2,568,513
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
157/157 [==============================] - 503s 3s/step - loss: 0.6655 -
accuracy: 0.5924 - val_loss: 0.7035 - val_accuracy: 0.6512
Epoch 2/10
157/157 [==============================] - 480s 3s/step - loss: 0.4772 -
accuracy: 0.8096 - val_loss: 0.6412 - val_accuracy: 0.6810
Epoch 3/10
157/157 [==============================] - 479s 3s/step - loss: 0.3592 -
accuracy: 0.8686 - val_loss: 0.4159 - val_accuracy: 0.8206
Epoch 4/10
157/157 [==============================] - 475s 3s/step - loss: 0.2830 -
accuracy: 0.9008 - val_loss: 1.6233 - val_accuracy: 0.6127
Epoch 5/10
157/157 [==============================] - 480s 3s/step - loss: 0.2306 -
accuracy: 0.9260 - val_loss: 0.4728 - val_accuracy: 0.8424
Epoch 6/10
157/157 [==============================] - 477s 3s/step - loss: 0.1876 -
accuracy: 0.9412 - val_loss: 0.6172 - val_accuracy: 0.8088
Epoch 7/10
157/157 [==============================] - 477s 3s/step - loss: 0.1402 -
accuracy: 0.9578 - val_loss: 0.4941 - val_accuracy: 0.8457
Epoch 8/10
157/157 [==============================] - 477s 3s/step - loss: 0.1263 -
accuracy: 0.9622 - val_loss: 0.4567 - val_accuracy: 0.8545
Epoch 9/10
157/157 [==============================] - 477s 3s/step - loss: 0.0884 -
accuracy: 0.9728 - val_loss: 0.5071 - val_accuracy: 0.8478
Epoch 10/10
```

```
157/157 [==============================] - 478s 3s/step - loss: 0.0790 -
accuracy: 0.9750 - val_loss: 0.5222 - val_accuracy: 0.8476
782/782 [==============================] - 287s 366ms/step - loss: 0.4696 -
accuracy: 0.7835
Test acc: 0.783
```

Understanding word embeddings

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

[ ]:
```
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}")
```

Understanding padding and masking Using an `Embedding` layer with masking enabled

[ ]:
```
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}")
```

Using pretrained word embeddings

[ ]:
```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

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.")
```

Preparing the GloVe word-embeddings matrix

[ ]:
```
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
```

[ ]:
```
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

[ ]:
```
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}")
```

Assignment 4(2)

May 5, 2024

This is a companion notebook for the book Deep Learning with Python, Second Edition. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

0.0.1 Processing words as a sequence: The sequence model approach

A first practical example Downloading the data

[1]:
```
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  14.8M      0  0:00:05  0:00:05 --:--:-- 16.9M
```

Preparing the data

[3]:
```
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, exist_ok=True)  # Add exist_ok=True argument
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples = int(0.2 * len(files))
    val_files = files[-num_val_samples:]

    for fname in val_files:
        shutil.move(train_dir / category / fname, val_dir / category / fname)
```

```
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 20000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

Preparing integer sequence datasets

[4]:
```
from tensorflow.keras import layers

max_length = 600
max_tokens = 20000
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)
```

A sequence model built on one-hot encoded vector sequences

[5]:
```
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()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None)]            0
_________________________________________________________________
tf.one_hot (TFOpLambda)      (None, None, 20000)       0
_________________________________________________________________
bidirectional (Bidirectional (None, 64)                5128448
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 5,128,513
Trainable params: 5,128,513
Non-trainable params: 0
_________________________________________________________________
```

Training a first basic sequence model

[ ]:
```
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}")
```

Understanding word embeddings

Learning word embeddings with the Embedding layer **Instantiating an `Embedding` layer**

[6]:
```
embedding_layer = layers.Embedding(input_dim=max_tokens, output_dim=256)
```

Model that uses an `Embedding` layer trained from scratch

[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_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_1 (Embedding)      (None, None, 256)         5120000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                73984
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 5,194,049
Trainable params: 5,194,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 215s 341ms/step - loss: 0.4718 -
accuracy: 0.7843 - val_loss: 0.2837 - val_accuracy: 0.8989
Epoch 2/10
625/625 [==============================] - 217s 347ms/step - loss: 0.3131 -
accuracy: 0.8851 - val_loss: 0.2141 - val_accuracy: 0.9244
Epoch 3/10
625/625 [==============================] - 212s 339ms/step - loss: 0.2399 -
accuracy: 0.9150 - val_loss: 0.2016 - val_accuracy: 0.9308
Epoch 4/10
625/625 [==============================] - 215s 343ms/step - loss: 0.2034 -
accuracy: 0.9310 - val_loss: 0.1754 - val_accuracy: 0.9405
Epoch 5/10
625/625 [==============================] - 213s 340ms/step - loss: 0.1812 -
```

```
accuracy: 0.9394 - val_loss: 0.3353 - val_accuracy: 0.8695
Epoch 6/10
625/625 [==============================] - 209s 335ms/step - loss: 0.1510 -
accuracy: 0.9513 - val_loss: 0.1416 - val_accuracy: 0.9537
Epoch 7/10
625/625 [==============================] - 209s 334ms/step - loss: 0.1253 -
accuracy: 0.9584 - val_loss: 0.2000 - val_accuracy: 0.9364
Epoch 8/10
625/625 [==============================] - 211s 337ms/step - loss: 0.1106 -
accuracy: 0.9640 - val_loss: 0.1326 - val_accuracy: 0.9593
Epoch 9/10
625/625 [==============================] - 212s 339ms/step - loss: 0.0889 -
accuracy: 0.9724 - val_loss: 0.1355 - val_accuracy: 0.9628
Epoch 10/10
625/625 [==============================] - 209s 335ms/step - loss: 0.0830 -
accuracy: 0.9738 - val_loss: 0.1447 - val_accuracy: 0.9632
782/782 [==============================] - 35s 44ms/step - loss: 0.5021 -
accuracy: 0.8530
Test acc: 0.853
```

Understanding padding and masking Using an Embedding layer with masking enabled

[ ]:
```
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}")
```

Using pretrained word embeddings

[8]:
```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2024-05-05 19:41:58--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)… 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80…
connected.
HTTP request sent, awaiting response… 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-05-05 19:41:58--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443…
connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-05-05 19:41:58--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)… 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

100%[=====================================>] 862,182,613 5.01MB/s   in 2m 39s

2024-05-05 19:44:38 (5.17 MB/s) - 'glove.6B.zip' saved [862182613/862182613]
```

Parsing the GloVe word-embeddings file

[9]:
```python
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.
```

Preparing the GloVe word-embeddings matrix

[10]:
```python
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

[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_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_2 (Embedding)      (None, None, 100)         2000000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 64)                34048
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
```

```
=================================================================
Total params: 2,034,113
Trainable params: 34,113
Non-trainable params: 2,000,000
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 222s 348ms/step - loss: 0.5775 -
accuracy: 0.6920 - val_loss: 0.4622 - val_accuracy: 0.7839
Epoch 2/10
625/625 [==============================] - 213s 341ms/step - loss: 0.4578 -
accuracy: 0.7893 - val_loss: 0.3916 - val_accuracy: 0.8280
Epoch 3/10
625/625 [==============================] - 213s 341ms/step - loss: 0.3995 -
accuracy: 0.8262 - val_loss: 0.3695 - val_accuracy: 0.8422
Epoch 4/10
625/625 [==============================] - 213s 341ms/step - loss: 0.3664 -
accuracy: 0.8429 - val_loss: 0.3241 - val_accuracy: 0.8616
Epoch 5/10
625/625 [==============================] - 212s 339ms/step - loss: 0.3407 -
accuracy: 0.8567 - val_loss: 0.3304 - val_accuracy: 0.8629
Epoch 6/10
625/625 [==============================] - 210s 335ms/step - loss: 0.3177 -
accuracy: 0.8701 - val_loss: 0.2820 - val_accuracy: 0.8834
Epoch 7/10
625/625 [==============================] - 209s 334ms/step - loss: 0.2975 -
accuracy: 0.8777 - val_loss: 0.2709 - val_accuracy: 0.8904
Epoch 8/10
625/625 [==============================] - 207s 331ms/step - loss: 0.2828 -
accuracy: 0.8824 - val_loss: 0.2657 - val_accuracy: 0.8925
Epoch 9/10
625/625 [==============================] - 210s 336ms/step - loss: 0.2674 -
accuracy: 0.8914 - val_loss: 0.2447 - val_accuracy: 0.9030
Epoch 10/10
625/625 [==============================] - 210s 336ms/step - loss: 0.2536 -
accuracy: 0.8961 - val_loss: 0.2388 - val_accuracy: 0.9064
782/782 [==============================] - 40s 49ms/step - loss: 0.2928 -
accuracy: 0.8750
Test acc: 0.875
```

[ ]:

[ ]: