hw4_sequence

November 23, 2023

[5]:
```
# Preparing the data
                    # Please be advised, I downloaded the data and unzip it␣
 ↪from my computer then I put the data folder in the same path of my library␣
 ↪folder.
```

[ ]:
```
# !!!!!!!!!!!!the really tricky part for glove, is that even the data is␣
 ↪encoded in utf-8 but the model still report error.
# !!!!!it took me couple hours to solve this by changing the encoding to ANSI␣
 ↪which works better for english letters.
```

[27]:
```
import urllib.request
import zipfile
import os

url = 'http://nlp.stanford.edu/data/glove.6B.zip'
file_name = 'glove.6B.zip'

#
urllib.request.urlretrieve(url, file_name)

#
with zipfile.ZipFile(file_name, 'r') as zip_ref:
    zip_ref.extractall()

#
os.remove(file_name)
```

[ ]:
```
# set-up train valid test, 6.7s
```

[8]:
```
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(0.2 * 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 5000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

[7]:
```
# Preparing integer sequence datasets, 33.3s
```

[9]:
```
from tensorflow.keras import layers

max_length = 150
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)
```

[3]:
```
# vocabulary = text_vectorization.get_vocabulary()
```

[9]:
```
# A sequence model built on one-hot encoded vector sequences, 0.3s
```

[10]:
```
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 (TFOpLambda)     (None, None, 20000)       0

 bidirectional_1 (Bidirectio  (None, 64)               5128448
 nal)

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 5,128,513
Trainable params: 5,128,513
Non-trainable params: 0
_________________________________________________________________
```

[11]:
```
# Training a first basic sequence model  5m15s
                                                    # Test acc: 0.841 not that␣
 ↪good, or even you can say so bad. Becasue this is a naive model.  binary␣
 ↪unigram model 0.895(but with max word =600!!!!!!!!!), code is on another␣
 ↪file, because this notebook is for sequence model.
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Have to cut the max word␣
 ↪to 150, because, Initially, I use the default 600 and my GPU was OOM.!!!!!!!!
 ↪!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! LATER WILL TRY 300 TO␣
 ↪SEE IF THIS WILL PERFORM BETTER COMPARED TO 150!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 ↪!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

[11]:
```
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
625/625 [==============================] - 35s 53ms/step - loss: 0.5104 -
accuracy: 0.7545 - val_loss: 0.3515 - val_accuracy: 0.8446
Epoch 2/10
625/625 [==============================] - 28s 45ms/step - loss: 0.3236 -
accuracy: 0.8701 - val_loss: 0.3342 - val_accuracy: 0.8576
Epoch 3/10
625/625 [==============================] - 27s 43ms/step - loss: 0.2628 -
accuracy: 0.8968 - val_loss: 0.4067 - val_accuracy: 0.8546
Epoch 4/10
625/625 [==============================] - 27s 43ms/step - loss: 0.2233 -
accuracy: 0.9144 - val_loss: 0.3792 - val_accuracy: 0.8592
Epoch 5/10
625/625 [==============================] - 27s 43ms/step - loss: 0.1894 -
accuracy: 0.9295 - val_loss: 0.3924 - val_accuracy: 0.8532
Epoch 6/10
625/625 [==============================] - 27s 43ms/step - loss: 0.1625 -
accuracy: 0.9409 - val_loss: 0.3937 - val_accuracy: 0.8590
Epoch 7/10
625/625 [==============================] - 27s 43ms/step - loss: 0.1394 -
accuracy: 0.9500 - val_loss: 0.3934 - val_accuracy: 0.8528
Epoch 8/10
625/625 [==============================] - 27s 43ms/step - loss: 0.1053 -
accuracy: 0.9643 - val_loss: 0.4923 - val_accuracy: 0.8548
Epoch 9/10
625/625 [==============================] - 27s 43ms/step - loss: 0.0804 -
accuracy: 0.9719 - val_loss: 0.5940 - val_accuracy: 0.8450
Epoch 10/10
625/625 [==============================] - 27s 43ms/step - loss: 0.0608 -
accuracy: 0.9803 - val_loss: 0.6225 - val_accuracy: 0.8382
782/782 [==============================] - 31s 39ms/step - loss: 0.3611 -
accuracy: 0.8409
Test acc: 0.841
```

[ ]:
```
# Learning word embeddings with the Embedding layer
```

[12]:
```
# Instantiating an Embedding layer
embedding_layer = layers.Embedding(input_dim=max_tokens, output_dim=256)
```

[ ]:
```
# Model that uses an Embedding layer trained from scratch, 3m6s
                        # Test acc: 0.835, bad model even worse compared to␣
 ↪naive base model(0.846).
```

[13]:
```
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_2 (Embedding)     (None, None, 256)         5120000

 bidirectional_2 (Bidirectio  (None, 64)               73984
 nal)

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 5,194,049
Trainable params: 5,194,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 42s 63ms/step - loss: 0.4678 -
accuracy: 0.7829 - val_loss: 0.3604 - val_accuracy: 0.8434
Epoch 2/10
```

```
625/625 [==============================] - 11s 18ms/step - loss: 0.2969 -
accuracy: 0.8830 - val_loss: 0.4093 - val_accuracy: 0.8284
Epoch 3/10
625/625 [==============================] - 12s 19ms/step - loss: 0.2342 -
accuracy: 0.9113 - val_loss: 0.3898 - val_accuracy: 0.8438
Epoch 4/10
625/625 [==============================] - 11s 18ms/step - loss: 0.1871 -
accuracy: 0.9301 - val_loss: 0.4383 - val_accuracy: 0.8422
Epoch 5/10
625/625 [==============================] - 12s 19ms/step - loss: 0.1526 -
accuracy: 0.9440 - val_loss: 0.4210 - val_accuracy: 0.8470
Epoch 6/10
625/625 [==============================] - 12s 18ms/step - loss: 0.1170 -
accuracy: 0.9589 - val_loss: 0.4676 - val_accuracy: 0.8360
Epoch 7/10
625/625 [==============================] - 11s 18ms/step - loss: 0.0872 -
accuracy: 0.9701 - val_loss: 0.5490 - val_accuracy: 0.8394
Epoch 8/10
625/625 [==============================] - 11s 18ms/step - loss: 0.0653 -
accuracy: 0.9789 - val_loss: 0.5826 - val_accuracy: 0.8410
Epoch 9/10
625/625 [==============================] - 11s 18ms/step - loss: 0.0492 -
accuracy: 0.9835 - val_loss: 0.6555 - val_accuracy: 0.8396
Epoch 10/10
625/625 [==============================] - 11s 18ms/step - loss: 0.0326 -
accuracy: 0.9891 - val_loss: 0.8134 - val_accuracy: 0.8338
782/782 [==============================] - 41s 51ms/step - loss: 0.3746 -
accuracy: 0.8354
Test acc: 0.835
```

[ ]:
```
# Using an Embedding layer with masking enabled, 3min25s
                # Test acc: 0.838, a little better with the previous model (0.
 ↪835)
```

[14]:
```
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_3 (Embedding)     (None, None, 256)         5120000

 bidirectional_3 (Bidirectio  (None, 64)               73984
 nal)

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 5,194,049
Trainable params: 5,194,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 24s 32ms/step - loss: 0.4382 -
accuracy: 0.7994 - val_loss: 0.3346 - val_accuracy: 0.8534
Epoch 2/10
625/625 [==============================] - 19s 30ms/step - loss: 0.2756 -
accuracy: 0.8907 - val_loss: 0.3533 - val_accuracy: 0.8492
Epoch 3/10
625/625 [==============================] - 18s 29ms/step - loss: 0.2110 -
accuracy: 0.9197 - val_loss: 0.3806 - val_accuracy: 0.8552
Epoch 4/10
625/625 [==============================] - 18s 29ms/step - loss: 0.1572 -
accuracy: 0.9419 - val_loss: 0.4678 - val_accuracy: 0.8442
Epoch 5/10
625/625 [==============================] - 18s 29ms/step - loss: 0.1177 -
accuracy: 0.9582 - val_loss: 0.4705 - val_accuracy: 0.8516
Epoch 6/10
625/625 [==============================] - 18s 29ms/step - loss: 0.0850 -
accuracy: 0.9692 - val_loss: 0.4879 - val_accuracy: 0.8332
Epoch 7/10
625/625 [==============================] - 18s 29ms/step - loss: 0.0632 -
```

```
accuracy: 0.9773 - val_loss: 0.5848 - val_accuracy: 0.8358
Epoch 8/10
625/625 [==============================] - 19s 30ms/step - loss: 0.0448 -
accuracy: 0.9843 - val_loss: 0.6837 - val_accuracy: 0.8192
Epoch 9/10
625/625 [==============================] - 19s 30ms/step - loss: 0.0306 -
accuracy: 0.9899 - val_loss: 0.7022 - val_accuracy: 0.8188
Epoch 10/10
625/625 [==============================] - 19s 30ms/step - loss: 0.0219 -
accuracy: 0.9927 - val_loss: 0.7805 - val_accuracy: 0.8346
782/782 [==============================] - 12s 14ms/step - loss: 0.3662 -
accuracy: 0.8380
Test acc: 0.838
```

[ ]:
```python
# Parsing the GloVe word-embeddings file, 8.5 s
```

[15]:
```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.
```

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

[16]:
```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
```

[17]:
```python
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, 2m55
                        #                       !!!!!!!!!!!!!!Test acc: 0.824 ,␣
 ↪did not beat my nonpretrained model, ????!!!!!!!!! textbook gave one␣
 ↪explanatin for this, for this serise of trials , my sample size is large␣
 ↪enough to let even the most naive model to learn from embedding from scratch.
# !!!!!!!!!!!!I am expecting that train the same thing on a small sample will␣
 ↪show the previlage of pretrained model!!!!!!!!!!!!!!!!
```

[18]:
```
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_4 (Embedding)     (None, None, 100)         2000000

 bidirectional_4 (Bidirectio  (None, 64)               34048
 nal)

 dropout_4 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 1)                 65

=================================================================
```

```
Total params: 2,034,113
Trainable params: 34,113
Non-trainable params: 2,000,000
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 22s 28ms/step - loss: 0.5811 -
accuracy: 0.6873 - val_loss: 0.4729 - val_accuracy: 0.7728
Epoch 2/10
625/625 [==============================] - 15s 23ms/step - loss: 0.4799 -
accuracy: 0.7753 - val_loss: 0.5094 - val_accuracy: 0.7744
Epoch 3/10
625/625 [==============================] - 16s 25ms/step - loss: 0.4408 -
accuracy: 0.7994 - val_loss: 0.4728 - val_accuracy: 0.7840
Epoch 4/10
625/625 [==============================] - 16s 25ms/step - loss: 0.4105 -
accuracy: 0.8153 - val_loss: 0.4099 - val_accuracy: 0.8162
Epoch 5/10
625/625 [==============================] - 16s 26ms/step - loss: 0.3892 -
accuracy: 0.8253 - val_loss: 0.3988 - val_accuracy: 0.8242
Epoch 6/10
625/625 [==============================] - 15s 24ms/step - loss: 0.3680 -
accuracy: 0.8399 - val_loss: 0.4150 - val_accuracy: 0.8292
Epoch 7/10
625/625 [==============================] - 15s 23ms/step - loss: 0.3510 -
accuracy: 0.8454 - val_loss: 0.4728 - val_accuracy: 0.8136
Epoch 8/10
625/625 [==============================] - 15s 24ms/step - loss: 0.3329 -
accuracy: 0.8565 - val_loss: 0.4011 - val_accuracy: 0.8350
Epoch 9/10
625/625 [==============================] - 16s 25ms/step - loss: 0.3175 -
accuracy: 0.8658 - val_loss: 0.4038 - val_accuracy: 0.8358
Epoch 10/10
625/625 [==============================] - 16s 26ms/step - loss: 0.3057 -
accuracy: 0.8708 - val_loss: 0.4132 - val_accuracy: 0.8346
782/782 [==============================] - 11s 12ms/step - loss: 0.3933 -
accuracy: 0.8239
Test acc: 0.824
```

hw4_part2

November 23, 2023

[ ]:
```
# Recall from my part1 notebook, I did the following things
#                           1. I set up the max word cut off as 150
#                           2. I did not limit the train valid sample size␣
 ↪(using the whole data I have in hand)
                        # Because of this enriched data, pretrained model did␣
 ↪not perform super good.

# In this part2 notebook, I will shrink the training/ valid size.
```

[18]:
```
# set-up train valid test to 100(50/50) 10000(5000/5000) 25000(12500/12500)

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\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 = 50

#         # 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\aclImdb\train\pos'

#
all_files = os.listdir(folder_path)

#
num_files_to_keep = 50

#
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 100 files belonging to 2 classes.
Found 10000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

[21]:
```
# 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)
```

[23]:
```
# 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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 input_1 (InputLayer)        [(None, None)]            0

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

 bidirectional (Bidirectiona  (None, 64)               2568448
 l)

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 1)                 65

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

[ ]:
```
# Training a first basic sequence model  1m29s
# Test acc: 0.575 which is very bad, this is because we have limited training␣
 ↪size, so without embedding, model can not map the pattern                  ␣
 ↪
```

[24]:
```
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
4/4 [==============================] - 23s 4s/step - loss: 0.6911 - accuracy:
0.5500 - val_loss: 0.6928 - val_accuracy: 0.5068
Epoch 2/10
4/4 [==============================] - 4s 1s/step - loss: 0.6832 - accuracy:
0.6300 - val_loss: 0.6921 - val_accuracy: 0.5107
Epoch 3/10
4/4 [==============================] - 4s 1s/step - loss: 0.6710 - accuracy:
0.8500 - val_loss: 0.6911 - val_accuracy: 0.5291
Epoch 4/10
4/4 [==============================] - 4s 1s/step - loss: 0.6476 - accuracy:
0.8600 - val_loss: 0.9437 - val_accuracy: 0.5066
Epoch 5/10
4/4 [==============================] - 4s 1s/step - loss: 0.6571 - accuracy:
0.8000 - val_loss: 0.6893 - val_accuracy: 0.5327
Epoch 6/10
4/4 [==============================] - 4s 1s/step - loss: 0.4973 - accuracy:
```

```
0.8400 - val_loss: 0.6684 - val_accuracy: 0.5834
Epoch 7/10
4/4 [==============================] - 4s 1s/step - loss: 0.3790 - accuracy:
0.9500 - val_loss: 0.7758 - val_accuracy: 0.5583
Epoch 8/10
4/4 [==============================] - 4s 1s/step - loss: 0.3509 - accuracy:
0.9400 - val_loss: 0.7128 - val_accuracy: 0.6036
Epoch 9/10
4/4 [==============================] - 4s 1s/step - loss: 0.2008 - accuracy:
0.9600 - val_loss: 0.7507 - val_accuracy: 0.5983
Epoch 10/10
4/4 [==============================] - 4s 1s/step - loss: 0.1726 - accuracy:
0.9600 - val_loss: 0.9619 - val_accuracy: 0.5843
782/782 [==============================] - 31s 38ms/step - loss: 0.6699 -
accuracy: 0.5746
Test acc: 0.575
```

[25]:
```
# 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, 35.8s
# Test acc: 0.614,  improved.
# embedding is very useful with a limited training size.
```

[26]:
```
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_2 (Embedding)     (None, None, 256)         2560000

 bidirectional_1 (Bidirectio  (None, 64)               73984
 nal)

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,634,049
Trainable params: 2,634,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 5s 982ms/step - loss: 0.6927 - accuracy:
0.5000 - val_loss: 0.6926 - val_accuracy: 0.5221
Epoch 2/10
4/4 [==============================] - 3s 843ms/step - loss: 0.6682 - accuracy:
0.7700 - val_loss: 0.6957 - val_accuracy: 0.5372
Epoch 3/10
4/4 [==============================] - 3s 862ms/step - loss: 0.6342 - accuracy:
0.8000 - val_loss: 0.6995 - val_accuracy: 0.5451
Epoch 4/10
4/4 [==============================] - 3s 851ms/step - loss: 0.5752 - accuracy:
0.8300 - val_loss: 0.7030 - val_accuracy: 0.5499
Epoch 5/10
4/4 [==============================] - 3s 865ms/step - loss: 0.4567 - accuracy:
0.8600 - val_loss: 0.7497 - val_accuracy: 0.5459
Epoch 6/10
4/4 [==============================] - 3s 858ms/step - loss: 0.3207 - accuracy:
0.9100 - val_loss: 0.6955 - val_accuracy: 0.5895
Epoch 7/10
4/4 [==============================] - 3s 902ms/step - loss: 0.1974 - accuracy:
0.9400 - val_loss: 0.6625 - val_accuracy: 0.6187
Epoch 8/10
4/4 [==============================] - 3s 839ms/step - loss: 0.1219 - accuracy:
0.9900 - val_loss: 0.7034 - val_accuracy: 0.6130
Epoch 9/10
4/4 [==============================] - 3s 856ms/step - loss: 0.0816 - accuracy:
1.0000 - val_loss: 0.8337 - val_accuracy: 0.5967
Epoch 10/10
4/4 [==============================] - 3s 833ms/step - loss: 0.0520 - accuracy:
1.0000 - val_loss: 0.8965 - val_accuracy: 0.6043
```

```
782/782 [==============================] - 7s 8ms/step - loss: 0.6662 -
accuracy: 0.6142
Test acc: 0.614
```

[ ]:
```
# Using an Embedding layer with masking enabled, 47s
# Test acc: 0.633 improve again.
# mask is useful for this case.
```

[27]:
```
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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_3 (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
4/4 [==============================] - 9s 1s/step - loss: 0.6987 - accuracy:
0.4400 - val_loss: 0.6926 - val_accuracy: 0.5149
Epoch 2/10
4/4 [==============================] - 3s 987ms/step - loss: 0.6684 - accuracy:
0.8900 - val_loss: 0.6914 - val_accuracy: 0.5444
Epoch 3/10
4/4 [==============================] - 3s 975ms/step - loss: 0.6374 - accuracy:
0.9900 - val_loss: 0.6900 - val_accuracy: 0.5292
Epoch 4/10
4/4 [==============================] - 3s 1s/step - loss: 0.5807 - accuracy:
0.9800 - val_loss: 0.6861 - val_accuracy: 0.5351
Epoch 5/10
4/4 [==============================] - 3s 1s/step - loss: 0.4625 - accuracy:
0.9300 - val_loss: 0.6773 - val_accuracy: 0.5723
Epoch 6/10
4/4 [==============================] - 3s 1s/step - loss: 0.3313 - accuracy:
1.0000 - val_loss: 0.6471 - val_accuracy: 0.6290
Epoch 7/10
4/4 [==============================] - 3s 1s/step - loss: 0.1615 - accuracy:
1.0000 - val_loss: 0.6415 - val_accuracy: 0.6401
Epoch 8/10
4/4 [==============================] - 3s 992ms/step - loss: 0.1099 - accuracy:
1.0000 - val_loss: 0.6507 - val_accuracy: 0.6182
Epoch 9/10
4/4 [==============================] - 3s 1s/step - loss: 0.0708 - accuracy:
1.0000 - val_loss: 0.6579 - val_accuracy: 0.6358
Epoch 10/10
4/4 [==============================] - 3s 1s/step - loss: 0.0452 - accuracy:
1.0000 - val_loss: 0.7007 - val_accuracy: 0.6351
782/782 [==============================] - 9s 10ms/step - loss: 0.6482 -
accuracy: 0.6328
Test acc: 0.633
```

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

[29]:
```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.
```

[30]:
```
# Preparing the GloVe word-embeddings matrix, 0s

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
```

[31]:
```
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,  48.6s
# Test acc: 0.571, bad performance which beyond my expectation, becasue Ithink␣
 ↪with li
# limited data, pretrained model should really dominates among all trials.

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

[32]:
```
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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_4 (Embedding)     (None, None, 100)         1000000

 bidirectional_3 (Bidirectio  (None, 64)               34048
 nal)

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,034,113
Trainable params: 34,113
Non-trainable params: 1,000,000
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 9s 2s/step - loss: 0.7145 - accuracy:
0.4700 - val_loss: 0.6881 - val_accuracy: 0.5333
Epoch 2/10
4/4 [==============================] - 3s 1s/step - loss: 0.6578 - accuracy:
0.6100 - val_loss: 0.6822 - val_accuracy: 0.5624
Epoch 3/10
4/4 [==============================] - 3s 1s/step - loss: 0.7044 - accuracy:
0.4800 - val_loss: 0.6805 - val_accuracy: 0.5792
Epoch 4/10
4/4 [==============================] - 3s 964ms/step - loss: 0.6679 - accuracy:
0.6500 - val_loss: 0.6891 - val_accuracy: 0.5345
Epoch 5/10
4/4 [==============================] - 3s 1s/step - loss: 0.6756 - accuracy:
0.5700 - val_loss: 0.6816 - val_accuracy: 0.5520
Epoch 6/10
4/4 [==============================] - 3s 1s/step - loss: 0.6519 - accuracy:
0.6300 - val_loss: 0.7068 - val_accuracy: 0.5142
Epoch 7/10
4/4 [==============================] - 3s 1s/step - loss: 0.6377 - accuracy:
0.5900 - val_loss: 0.6753 - val_accuracy: 0.5744
```

```
Epoch 8/10
4/4 [==============================] - 3s 1s/step - loss: 0.6128 - accuracy:
0.7000 - val_loss: 0.6752 - val_accuracy: 0.5865
Epoch 9/10
4/4 [==============================] - 3s 1s/step - loss: 0.6158 - accuracy:
0.6600 - val_loss: 0.6739 - val_accuracy: 0.5858
Epoch 10/10
4/4 [==============================] - 3s 976ms/step - loss: 0.6084 - accuracy:
0.6900 - val_loss: 0.6976 - val_accuracy: 0.5453
782/782 [==============================] - 9s 9ms/step - loss: 0.6791 -
accuracy: 0.5706
Test acc: 0.571
```

hw4_part3

November 23, 2023

[ ]:
```
# my part 2 code, which is with 100 10000 25000 size shows
# even with embedding and pretrained model, the performance is not good.
# this could be due to limited traing size
# in this part 3 code, I will increase traing size to 10000(5000/5000)
# then ppl in my part 4 code, I will increase training size to 3000(1500/1500)
# please be noted, that max= 25000-10000(assign for valid)=15000
```

[8]:
```
# set-up train valid test to 10000(5000/5000) 10000(5000/5000) 25000(12500/
 ↪12500)

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_part3\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 = 5000

#         # 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_part3\aclImdb\train\pos'

#
all_files = os.listdir(folder_path)

#
num_files_to_keep = 5000

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

[9]:
```
# 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)
```

[10]:
```
# 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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0
```

```
 tf.one_hot (TFOpLambda)     (None, None, 10000)       0

 bidirectional (Bidirectiona  (None, 64)               2568448
 l)

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 1)                 65

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

[ ]:
```
# Training a first basic sequence model     2m33s
# Test acc: 0.832       , significantly improved compared to part 2 (100␣
 ↪training size)
```

[11]:
```
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
313/313 [==============================] - 25s 64ms/step - loss: 0.5903 -
accuracy: 0.6872 - val_loss: 0.4436 - val_accuracy: 0.8099
Epoch 2/10
313/313 [==============================] - 11s 34ms/step - loss: 0.4045 -
accuracy: 0.8451 - val_loss: 0.4656 - val_accuracy: 0.8029
Epoch 3/10
313/313 [==============================] - 11s 34ms/step - loss: 0.3239 -
accuracy: 0.8845 - val_loss: 0.3770 - val_accuracy: 0.8447
Epoch 4/10
313/313 [==============================] - 11s 34ms/step - loss: 0.2523 -
accuracy: 0.9098 - val_loss: 0.5399 - val_accuracy: 0.8284
Epoch 5/10
313/313 [==============================] - 11s 34ms/step - loss: 0.1993 -
accuracy: 0.9331 - val_loss: 0.3785 - val_accuracy: 0.8407
Epoch 6/10
313/313 [==============================] - 11s 34ms/step - loss: 0.1591 -
accuracy: 0.9460 - val_loss: 0.4934 - val_accuracy: 0.8183
Epoch 7/10
313/313 [==============================] - 11s 34ms/step - loss: 0.1219 -
```

```
accuracy: 0.9597 - val_loss: 0.4484 - val_accuracy: 0.8242
Epoch 8/10
313/313 [==============================] - 11s 34ms/step - loss: 0.1035 -
accuracy: 0.9682 - val_loss: 0.4944 - val_accuracy: 0.8067
Epoch 9/10
313/313 [==============================] - 11s 34ms/step - loss: 0.0786 -
accuracy: 0.9757 - val_loss: 1.4872 - val_accuracy: 0.7528
Epoch 10/10
313/313 [==============================] - 11s 34ms/step - loss: 0.0765 -
accuracy: 0.9771 - val_loss: 0.5549 - val_accuracy: 0.8114
782/782 [==============================] - 32s 40ms/step - loss: 0.4014 -
accuracy: 0.8324
Test acc: 0.832
```

[12]:
```
# 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, 1m25s
# Test acc: 0.826,  does not improved.
# embedding is not that useful when you have a large size of training
```

[13]:
```
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)         2560000

 bidirectional_1 (Bidirectio  (None, 64)               73984
 nal)

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,634,049
Trainable params: 2,634,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
313/313 [==============================] - 10s 25ms/step - loss: 0.5197 -
accuracy: 0.7458 - val_loss: 0.3838 - val_accuracy: 0.8402
Epoch 2/10
313/313 [==============================] - 7s 24ms/step - loss: 0.3290 -
accuracy: 0.8730 - val_loss: 0.4204 - val_accuracy: 0.8446
Epoch 3/10
313/313 [==============================] - 8s 24ms/step - loss: 0.2523 -
accuracy: 0.9076 - val_loss: 0.5638 - val_accuracy: 0.8094
Epoch 4/10
313/313 [==============================] - 7s 24ms/step - loss: 0.2027 -
accuracy: 0.9256 - val_loss: 0.4686 - val_accuracy: 0.8280
Epoch 5/10
313/313 [==============================] - 7s 24ms/step - loss: 0.1669 -
accuracy: 0.9429 - val_loss: 0.4620 - val_accuracy: 0.8343
Epoch 6/10
313/313 [==============================] - 7s 24ms/step - loss: 0.1315 -
accuracy: 0.9553 - val_loss: 0.4810 - val_accuracy: 0.8297
Epoch 7/10
313/313 [==============================] - 8s 24ms/step - loss: 0.1012 -
accuracy: 0.9660 - val_loss: 0.5283 - val_accuracy: 0.8315
Epoch 8/10
313/313 [==============================] - 8s 24ms/step - loss: 0.0770 -
accuracy: 0.9744 - val_loss: 0.9635 - val_accuracy: 0.7708
Epoch 9/10
313/313 [==============================] - 8s 24ms/step - loss: 0.0520 -
accuracy: 0.9838 - val_loss: 0.6769 - val_accuracy: 0.8233
Epoch 10/10
313/313 [==============================] - 8s 24ms/step - loss: 0.0408 -
accuracy: 0.9870 - val_loss: 1.0744 - val_accuracy: 0.7632
782/782 [==============================] - 7s 8ms/step - loss: 0.4010 -
accuracy: 0.8258
Test acc: 0.826
```

[ ]:
```
# Using an Embedding layer with masking enabled, 1m49s
# Test acc: 0.831 a little improve compared to the latest one.
# mask is useful for this case.
```

[14]:
```
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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_2 (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
313/313 [==============================] - 15s 33ms/step - loss: 0.4998 -
accuracy: 0.7578 - val_loss: 0.4665 - val_accuracy: 0.7741
```

```
Epoch 2/10
313/313 [==============================] - 9s 28ms/step - loss: 0.2972 -
accuracy: 0.8821 - val_loss: 0.3893 - val_accuracy: 0.8472
Epoch 3/10
313/313 [==============================] - 9s 28ms/step - loss: 0.2185 -
accuracy: 0.9168 - val_loss: 0.3992 - val_accuracy: 0.8480
Epoch 4/10
313/313 [==============================] - 9s 28ms/step - loss: 0.1562 -
accuracy: 0.9444 - val_loss: 1.0728 - val_accuracy: 0.7050
Epoch 5/10
313/313 [==============================] - 9s 28ms/step - loss: 0.1133 -
accuracy: 0.9599 - val_loss: 0.5367 - val_accuracy: 0.8292
Epoch 6/10
313/313 [==============================] - 9s 28ms/step - loss: 0.0791 -
accuracy: 0.9735 - val_loss: 0.5818 - val_accuracy: 0.8225
Epoch 7/10
313/313 [==============================] - 9s 27ms/step - loss: 0.0522 -
accuracy: 0.9821 - val_loss: 0.6493 - val_accuracy: 0.8176
Epoch 8/10
313/313 [==============================] - 9s 28ms/step - loss: 0.0346 -
accuracy: 0.9893 - val_loss: 0.6908 - val_accuracy: 0.8194
Epoch 9/10
313/313 [==============================] - 9s 28ms/step - loss: 0.0194 -
accuracy: 0.9949 - val_loss: 0.8471 - val_accuracy: 0.8105
Epoch 10/10
313/313 [==============================] - 9s 28ms/step - loss: 0.0151 -
accuracy: 0.9955 - val_loss: 0.9377 - val_accuracy: 0.8186
782/782 [==============================] - 9s 9ms/step - loss: 0.4203 -
accuracy: 0.8312
Test acc: 0.831
```

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

[15]:
```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.
```

[16]:
```
# Preparing the GloVe word-embeddings matrix, 0s

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
```

[17]:
```
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,  1m51s
# Test acc: 0.820, same-flat

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

[18]:
```
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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

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

 bidirectional_3 (Bidirectio  (None, 64)               34048
 nal)

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,034,113
Trainable params: 34,113
Non-trainable params: 1,000,000
_________________________________________________________________
Epoch 1/10
313/313 [==============================] - 15s 36ms/step - loss: 0.6426 -
accuracy: 0.6237 - val_loss: 0.5274 - val_accuracy: 0.7473
Epoch 2/10
313/313 [==============================] - 10s 31ms/step - loss: 0.5384 -
accuracy: 0.7345 - val_loss: 0.4948 - val_accuracy: 0.7660
Epoch 3/10
313/313 [==============================] - 10s 31ms/step - loss: 0.4843 -
accuracy: 0.7718 - val_loss: 0.4749 - val_accuracy: 0.7755
Epoch 4/10
313/313 [==============================] - 9s 29ms/step - loss: 0.4494 -
accuracy: 0.7918 - val_loss: 0.4933 - val_accuracy: 0.7521
Epoch 5/10
313/313 [==============================] - 10s 31ms/step - loss: 0.4311 -
accuracy: 0.8061 - val_loss: 0.4281 - val_accuracy: 0.7989
Epoch 6/10
313/313 [==============================] - 9s 29ms/step - loss: 0.4131 -
accuracy: 0.8139 - val_loss: 0.4413 - val_accuracy: 0.7854
Epoch 7/10
313/313 [==============================] - 9s 29ms/step - loss: 0.3906 -
accuracy: 0.8227 - val_loss: 0.4409 - val_accuracy: 0.7973
Epoch 8/10
313/313 [==============================] - 9s 29ms/step - loss: 0.3770 -
accuracy: 0.8331 - val_loss: 0.4318 - val_accuracy: 0.8135
Epoch 9/10
313/313 [==============================] - 10s 31ms/step - loss: 0.3570 -
accuracy: 0.8458 - val_loss: 0.3945 - val_accuracy: 0.8241
Epoch 10/10
```

```
313/313 [==============================] - 9s 29ms/step - loss: 0.3487 -
accuracy: 0.8499 - val_loss: 0.3958 - val_accuracy: 0.8197
782/782 [==============================] - 9s 10ms/step - loss: 0.3958 -
accuracy: 0.8196
Test acc: 0.820
```

hw4_part4

November 23, 2023

[ ]:
```
# then ppl in my part 4 code, I will increase training size to 3000(1500/1500)
# please be noted, that max= 25000-10000(assign for valid)=15000
```

[1]:
```
# set-up train valid test to 10000(5000/5000) 10000(5000/5000) 25000(12500/
 ↪12500)

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_part4\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 = 1500

#         # 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_part4\aclImdb\train\pos'

#
all_files = os.listdir(folder_path)

#
num_files_to_keep = 1500

#
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 3000 files belonging to 2 classes.
Found 10000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

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

[3]:
```
# 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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

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

 bidirectional (Bidirectiona  (None, 64)               2568448
 l)
```

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

 dense (Dense)               (None, 1)                 65

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

[4]:
```
# Training a first basic sequence model     1m47s
# Test acc: 0.787
```

[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
94/94 [==============================] - 20s 169ms/step - loss: 0.6911 -
accuracy: 0.5370 - val_loss: 0.6772 - val_accuracy: 0.6471
Epoch 2/10
94/94 [==============================] - 6s 63ms/step - loss: 0.6105 - accuracy:
0.6860 - val_loss: 0.5309 - val_accuracy: 0.7625
Epoch 3/10
94/94 [==============================] - 6s 63ms/step - loss: 0.4294 - accuracy:
0.8253 - val_loss: 0.5375 - val_accuracy: 0.7344
Epoch 4/10
94/94 [==============================] - 6s 63ms/step - loss: 0.3160 - accuracy:
0.8757 - val_loss: 0.5020 - val_accuracy: 0.7965
Epoch 5/10
94/94 [==============================] - 6s 62ms/step - loss: 0.2479 - accuracy:
0.9083 - val_loss: 0.6533 - val_accuracy: 0.7203
Epoch 6/10
94/94 [==============================] - 6s 62ms/step - loss: 0.2336 - accuracy:
0.9180 - val_loss: 0.5436 - val_accuracy: 0.7824
Epoch 7/10
94/94 [==============================] - 6s 62ms/step - loss: 0.1599 - accuracy:
0.9473 - val_loss: 0.6594 - val_accuracy: 0.7781
Epoch 8/10
94/94 [==============================] - 6s 62ms/step - loss: 0.1588 - accuracy:
0.9430 - val_loss: 0.5565 - val_accuracy: 0.7690
Epoch 9/10
```

```
94/94 [==============================] - 6s 62ms/step - loss: 0.1008 - accuracy:
0.9657 - val_loss: 0.6821 - val_accuracy: 0.7889
Epoch 10/10
94/94 [==============================] - 6s 62ms/step - loss: 0.0762 - accuracy:
0.9727 - val_loss: 0.6785 - val_accuracy: 0.7928
782/782 [==============================] - 32s 40ms/step - loss: 0.5234 -
accuracy: 0.7870
Test acc: 0.787
```

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

[7]:
```
# Model that uses an Embedding layer trained from scratch, 51s
# Test acc: 0.760,  does not improved.
# embedding is not that useful when you have a large size of training
```

[8]:
```
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)         2560000

 bidirectional_1 (Bidirectio  (None, 64)               73984
 nal)
```

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

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,634,049
Trainable params: 2,634,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
94/94 [==============================] - 6s 47ms/step - loss: 0.6544 - accuracy:
0.6060 - val_loss: 0.6115 - val_accuracy: 0.6587
Epoch 2/10
94/94 [==============================] - 4s 44ms/step - loss: 0.4171 - accuracy:
0.8377 - val_loss: 0.4763 - val_accuracy: 0.7736
Epoch 3/10
94/94 [==============================] - 4s 42ms/step - loss: 0.2372 - accuracy:
0.9170 - val_loss: 0.4873 - val_accuracy: 0.7749
Epoch 4/10
94/94 [==============================] - 4s 43ms/step - loss: 0.1386 - accuracy:
0.9510 - val_loss: 0.5740 - val_accuracy: 0.7849
Epoch 5/10
94/94 [==============================] - 4s 42ms/step - loss: 0.0635 - accuracy:
0.9827 - val_loss: 0.6868 - val_accuracy: 0.7910
Epoch 6/10
94/94 [==============================] - 4s 42ms/step - loss: 0.0566 - accuracy:
0.9827 - val_loss: 0.7031 - val_accuracy: 0.8034
Epoch 7/10
94/94 [==============================] - 4s 41ms/step - loss: 0.0300 - accuracy:
0.9917 - val_loss: 0.7309 - val_accuracy: 0.7972
Epoch 8/10
94/94 [==============================] - 4s 42ms/step - loss: 0.0187 - accuracy:
0.9943 - val_loss: 0.9624 - val_accuracy: 0.7948
Epoch 9/10
94/94 [==============================] - 4s 43ms/step - loss: 0.0271 - accuracy:
0.9927 - val_loss: 0.8571 - val_accuracy: 0.7741
Epoch 10/10
94/94 [==============================] - 4s 44ms/step - loss: 0.0157 - accuracy:
0.9960 - val_loss: 0.8434 - val_accuracy: 0.7871
782/782 [==============================] - 7s 9ms/step - loss: 0.4904 -
accuracy: 0.7605
Test acc: 0.760
```

[9]:
```
# Using an Embedding layer with masking enabled, 1m49s
# Test acc: 0.800 a little improve compared to the latest one.
# mask is useful for this case.
```

[10]:
```
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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_2 (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
94/94 [==============================] - 10s 65ms/step - loss: 0.6549 -
accuracy: 0.6037 - val_loss: 0.5286 - val_accuracy: 0.7475
Epoch 2/10
94/94 [==============================] - 5s 50ms/step - loss: 0.3947 - accuracy:
0.8327 - val_loss: 0.4258 - val_accuracy: 0.8047
Epoch 3/10
```

```
94/94 [==============================] - 5s 51ms/step - loss: 0.2287 - accuracy:
0.9117 - val_loss: 0.4841 - val_accuracy: 0.7983
Epoch 4/10
94/94 [==============================] - 5s 51ms/step - loss: 0.1224 - accuracy:
0.9587 - val_loss: 0.6003 - val_accuracy: 0.7968
Epoch 5/10
94/94 [==============================] - 5s 50ms/step - loss: 0.0596 - accuracy:
0.9823 - val_loss: 0.6838 - val_accuracy: 0.7925
Epoch 6/10
94/94 [==============================] - 5s 50ms/step - loss: 0.0365 - accuracy:
0.9877 - val_loss: 0.8371 - val_accuracy: 0.7939
Epoch 7/10
94/94 [==============================] - 5s 49ms/step - loss: 0.0169 - accuracy:
0.9947 - val_loss: 0.7747 - val_accuracy: 0.8006
Epoch 8/10
94/94 [==============================] - 5s 49ms/step - loss: 0.0129 - accuracy:
0.9970 - val_loss: 0.9324 - val_accuracy: 0.8028
Epoch 9/10
94/94 [==============================] - 5s 49ms/step - loss: 0.0035 - accuracy:
0.9990 - val_loss: 1.1798 - val_accuracy: 0.7969
Epoch 10/10
94/94 [==============================] - 5s 48ms/step - loss: 0.0131 - accuracy:
0.9963 - val_loss: 0.9934 - val_accuracy: 0.7939
782/782 [==============================] - 9s 9ms/step - loss: 0.4368 -
accuracy: 0.7999
Test acc: 0.800
```

[11]:
```python
# pretrained model
```

[17]:
```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.
```

[18]:
```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
```

[19]:
```
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,  1m9.9s
# Test acc: 0.779

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

[20]:
```
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_4 (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
94/94 [==============================] - 11s 75ms/step - loss: 0.6927 -
accuracy: 0.5433 - val_loss: 0.6708 - val_accuracy: 0.5738
Epoch 2/10
94/94 [==============================] - 5s 56ms/step - loss: 0.6566 - accuracy:
0.6067 - val_loss: 0.5988 - val_accuracy: 0.6891
Epoch 3/10
94/94 [==============================] - 5s 51ms/step - loss: 0.6063 - accuracy:
0.6707 - val_loss: 0.6339 - val_accuracy: 0.6473
Epoch 4/10
94/94 [==============================] - 5s 58ms/step - loss: 0.5773 - accuracy:
0.7007 - val_loss: 0.5753 - val_accuracy: 0.6982
Epoch 5/10
94/94 [==============================] - 5s 57ms/step - loss: 0.5535 - accuracy:
0.7227 - val_loss: 0.5182 - val_accuracy: 0.7556
Epoch 6/10
94/94 [==============================] - 5s 51ms/step - loss: 0.5217 - accuracy:
0.7440 - val_loss: 0.7002 - val_accuracy: 0.6547
Epoch 7/10
94/94 [==============================] - 6s 60ms/step - loss: 0.5053 - accuracy:
0.7563 - val_loss: 0.4837 - val_accuracy: 0.7727
Epoch 8/10
94/94 [==============================] - 5s 53ms/step - loss: 0.4846 - accuracy:
0.7690 - val_loss: 0.4892 - val_accuracy: 0.7675
Epoch 9/10
94/94 [==============================] - 5s 53ms/step - loss: 0.4658 - accuracy:
0.7880 - val_loss: 0.4848 - val_accuracy: 0.7710
Epoch 10/10
94/94 [==============================] - 6s 61ms/step - loss: 0.4539 - accuracy:
0.7977 - val_loss: 0.4621 - val_accuracy: 0.7825
782/782 [==============================] - 9s 10ms/step - loss: 0.4700 -
accuracy: 0.7795
```

```
Test acc: 0.779
```

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
```

hw4_part6

November 23, 2023

[ ]:
```
# final trial will be used high D glove moedel with all availiable data to
# build to expected best model
# release some of the word length to 400
```

[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)
    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 5000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

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

```
max_length = 400  # I think the max of length of my gpu that be handled is 400␣
 ↪not 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, 0.3s
```

[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()
# RNN-LSTM
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 tf.one_hot (TFOpLambda)     (None, None, 20000)       0

 bidirectional (Bidirectiona  (None, 64)               5128448
 l)

 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  7m56.8s
# Test acc: 0.878
```

[6]:
```
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
625/625 [==============================] - 51s 73ms/step - loss: 0.5210 -
accuracy: 0.7498 - val_loss: 0.3549 - val_accuracy: 0.8650
Epoch 2/10
625/625 [==============================] - 44s 70ms/step - loss: 0.3636 -
accuracy: 0.8664 - val_loss: 0.3023 - val_accuracy: 0.8748
Epoch 3/10
625/625 [==============================] - 43s 69ms/step - loss: 0.2700 -
accuracy: 0.9020 - val_loss: 0.3188 - val_accuracy: 0.8710
Epoch 4/10
625/625 [==============================] - 44s 70ms/step - loss: 0.2321 -
accuracy: 0.9191 - val_loss: 0.3021 - val_accuracy: 0.8878
Epoch 5/10
625/625 [==============================] - 44s 70ms/step - loss: 0.2048 -
accuracy: 0.9314 - val_loss: 0.5378 - val_accuracy: 0.8586
Epoch 6/10
625/625 [==============================] - 43s 70ms/step - loss: 0.1806 -
accuracy: 0.9398 - val_loss: 0.3078 - val_accuracy: 0.8832
Epoch 7/10
625/625 [==============================] - 44s 70ms/step - loss: 0.1505 -
accuracy: 0.9480 - val_loss: 0.3476 - val_accuracy: 0.8798
Epoch 8/10
625/625 [==============================] - 44s 70ms/step - loss: 0.1307 -
accuracy: 0.9559 - val_loss: 0.5844 - val_accuracy: 0.8706
Epoch 9/10
625/625 [==============================] - 44s 71ms/step - loss: 0.1137 -
accuracy: 0.9626 - val_loss: 0.3568 - val_accuracy: 0.8602
Epoch 10/10
```

```
625/625 [==============================] - 44s 70ms/step - loss: 0.0918 -
accuracy: 0.9713 - val_loss: 0.4962 - val_accuracy: 0.8804
782/782 [==============================] - 32s 40ms/step - loss: 0.3165 -
accuracy: 0.8779
Test acc: 0.878
```

[7]:
```
# 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, 4m33s
# Test acc: 0.849, embedding is not that useful when you have greate data␣
 ↪source for training.
```

[8]:
```
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 (Bidirectio  (None, 64)               73984
 nal)

 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 [==============================] - 39s 58ms/step - loss: 0.4861 -
accuracy: 0.7835 - val_loss: 0.3698 - val_accuracy: 0.8556
Epoch 2/10
625/625 [==============================] - 22s 35ms/step - loss: 0.3101 -
accuracy: 0.8846 - val_loss: 0.4703 - val_accuracy: 0.7758
Epoch 3/10
625/625 [==============================] - 22s 35ms/step - loss: 0.2491 -
accuracy: 0.9133 - val_loss: 0.3550 - val_accuracy: 0.8674
Epoch 4/10
625/625 [==============================] - 22s 35ms/step - loss: 0.2037 -
accuracy: 0.9309 - val_loss: 0.4144 - val_accuracy: 0.8630
Epoch 5/10
625/625 [==============================] - 22s 35ms/step - loss: 0.1711 -
accuracy: 0.9423 - val_loss: 0.4548 - val_accuracy: 0.8706
Epoch 6/10
625/625 [==============================] - 22s 35ms/step - loss: 0.1369 -
accuracy: 0.9526 - val_loss: 0.3856 - val_accuracy: 0.8706
Epoch 7/10
625/625 [==============================] - 22s 35ms/step - loss: 0.1082 -
accuracy: 0.9653 - val_loss: 0.4173 - val_accuracy: 0.8740
Epoch 8/10
625/625 [==============================] - 23s 37ms/step - loss: 0.0865 -
accuracy: 0.9722 - val_loss: 0.4445 - val_accuracy: 0.8708
Epoch 9/10
625/625 [==============================] - 23s 36ms/step - loss: 0.0756 -
accuracy: 0.9765 - val_loss: 0.4931 - val_accuracy: 0.8604
Epoch 10/10
625/625 [==============================] - 23s 37ms/step - loss: 0.0604 -
accuracy: 0.9815 - val_loss: 0.5164 - val_accuracy: 0.8708
782/782 [==============================] - 27s 33ms/step - loss: 0.3994 -
accuracy: 0.8490
Test acc: 0.849
```

[ ]:
```
# Using an Embedding layer with masking enabled, 4m29
                # Test acc: 0.874, a little better with the previous model, but␣
 ↪same as the naive model.
```

[9]:
```
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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_2 (Embedding)     (None, None, 256)         5120000

 bidirectional_2 (Bidirectio  (None, 64)               73984
 nal)

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 5,194,049
Trainable params: 5,194,049
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 31s 43ms/step - loss: 0.4061 -
accuracy: 0.8156 - val_loss: 0.2924 - val_accuracy: 0.8816
Epoch 2/10
625/625 [==============================] - 25s 39ms/step - loss: 0.2356 -
accuracy: 0.9109 - val_loss: 0.3206 - val_accuracy: 0.8812
Epoch 3/10
625/625 [==============================] - 25s 40ms/step - loss: 0.1690 -
accuracy: 0.9391 - val_loss: 0.5246 - val_accuracy: 0.8366
Epoch 4/10
```

```
625/625 [==============================] - 24s 39ms/step - loss: 0.1268 -
accuracy: 0.9556 - val_loss: 0.3862 - val_accuracy: 0.8518
Epoch 5/10
625/625 [==============================] - 24s 39ms/step - loss: 0.0945 -
accuracy: 0.9681 - val_loss: 0.4018 - val_accuracy: 0.8700
Epoch 6/10
625/625 [==============================] - 24s 39ms/step - loss: 0.0676 -
accuracy: 0.9764 - val_loss: 0.5169 - val_accuracy: 0.8550
Epoch 7/10
625/625 [==============================] - 24s 39ms/step - loss: 0.0499 -
accuracy: 0.9832 - val_loss: 0.5912 - val_accuracy: 0.8592
Epoch 8/10
625/625 [==============================] - 25s 40ms/step - loss: 0.0363 -
accuracy: 0.9878 - val_loss: 0.5559 - val_accuracy: 0.8650
Epoch 9/10
625/625 [==============================] - 25s 41ms/step - loss: 0.0233 -
accuracy: 0.9922 - val_loss: 0.6376 - val_accuracy: 0.8568
Epoch 10/10
625/625 [==============================] - 24s 39ms/step - loss: 0.0178 -
accuracy: 0.9947 - val_loss: 0.6269 - val_accuracy: 0.8658
782/782 [==============================] - 15s 17ms/step - loss: 0.3007 -
accuracy: 0.8740
Test acc: 0.874
```

[ ]:
```
# Parsing the GloVe word-embeddings file, 27s
```

[10]:
```
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.
```

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

[11]:
```
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
```

[12]:
```
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, 4m52s
# Test acc: 0.879 , did not beat my nonpretrained model, ????!!!!!!!!! textbook␣
 ↪gave one explanatin for this, for this serise of trials , my sample size is␣
 ↪large enough to let even the most naive model to learn from embedding from␣
 ↪scratch.
```

[13]:
```
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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_3 (Embedding)     (None, None, 300)         6000000
```

```
 bidirectional_3 (Bidirectio  (None, 64)              85248
 nal)

 dropout_3 (Dropout)         (None, 64)              0

 dense_3 (Dense)             (None, 1)               65

=================================================================
Total params: 6,085,313
Trainable params: 85,313
Non-trainable params: 6,000,000
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 36s 51ms/step - loss: 0.5093 -
accuracy: 0.7470 - val_loss: 0.3996 - val_accuracy: 0.8236
Epoch 2/10
625/625 [==============================] - 26s 41ms/step - loss: 0.3878 -
accuracy: 0.8298 - val_loss: 0.4476 - val_accuracy: 0.8210
Epoch 3/10
625/625 [==============================] - 29s 46ms/step - loss: 0.3469 -
accuracy: 0.8523 - val_loss: 0.3434 - val_accuracy: 0.8532
Epoch 4/10
625/625 [==============================] - 26s 41ms/step - loss: 0.3125 -
accuracy: 0.8706 - val_loss: 0.3523 - val_accuracy: 0.8498
Epoch 5/10
625/625 [==============================] - 25s 41ms/step - loss: 0.2869 -
accuracy: 0.8798 - val_loss: 0.3461 - val_accuracy: 0.8578
Epoch 6/10
625/625 [==============================] - 25s 40ms/step - loss: 0.2593 -
accuracy: 0.8960 - val_loss: 0.3612 - val_accuracy: 0.8566
Epoch 7/10
625/625 [==============================] - 25s 40ms/step - loss: 0.2405 -
accuracy: 0.9048 - val_loss: 0.3609 - val_accuracy: 0.8664
Epoch 8/10
625/625 [==============================] - 25s 40ms/step - loss: 0.2188 -
accuracy: 0.9145 - val_loss: 0.3478 - val_accuracy: 0.8668
Epoch 9/10
625/625 [==============================] - 25s 40ms/step - loss: 0.1940 -
accuracy: 0.9238 - val_loss: 0.3710 - val_accuracy: 0.8668
Epoch 10/10
625/625 [==============================] - 29s 47ms/step - loss: 0.1740 -
accuracy: 0.9337 - val_loss: 0.3317 - val_accuracy: 0.8742
782/782 [==============================] - 16s 18ms/step - loss: 0.3155 -
accuracy: 0.8794
Test acc: 0.879
```