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
```

[ ]:

```
# pretrained model
```

[15]:

```
# 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
```