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
```