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 3000(1500/1500) 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
```