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
```