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
```