Text and Sequence

Phani Varshitha, Durga Chowdary

We will be using IMDB data for this text and sequence problem. Firstly, we need to create a validation set with 80% of training dataset and setting apart 20% for training.

Reading Data

```
import os, shutil , pathlib , random
base_dir =
pathlib.Path("C:/Users/varshitha/Downloads/aclImdb_v1/aclImdb")
val_dir= base_dir/"validation"
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 = 5000
    val_file = files[:num_val_samples]
    for fname in val_file:
        shutil.move(train_dir/category/fname,
                    val_dir/category,fname)
```

Making a small training sample as well :

```
train_dir_1 =base_dir/"train1"
for category in ("neg","pos"):
     os.makedirs(train_dir_1/category)
     files= os.listdir(train_dir/category)
     random.Random(1337).shuffle(files)
     num_train_samples = 50
     train_file = files[:num_train_samples]
     for fname in train_file:
        shutil.move(train_dir/category/fname,
                    train_dir_1/category,fname)
```

Reading our datasets :

```
from tensorflow import keras
batch_size = 32
train = keras.utils.text_dataset_from_directory
(train_dir_1,batch_size=batch_size)
validation=keras.utils.text_dataset_from_directory(val_dir,batch_size=
batch_size)
test=keras.utils.text_dataset_from_directory(base_dir/
  "test",batch_size=batch_size)
```

```
Found 100 files belonging to 2 classes.
Found 10000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

Trying sequencing model:

Preparing dataset for this model:

```
from tensorflow.keras import layers
max_length = 150 # Cutting off values after 150 words
max_tokens = 10000 # Considering only top 10,000 words
text_vectorization = layers.TextVectorization(
      max_tokens=max_tokens,
      output_mode="int",
      output_sequence_length=max_length,
)
text_only_train_ds = train.map(lambda x, y: x)
# Turning text to vectors
text_vectorization.adapt(text_only_train_ds)
int_train_ds = train.map(
lambda x, y: (text_vectorization(x), y), num_parallel_calls=4)
int_val_ds = validation.map(
lambda x, y: (text_vectorization(x), y), num_parallel_calls=4)
int_test_ds = test.map(
lambda x, y: (text_vectorization(x), y), num_parallel_calls=4)
```

Model Construction - Embedding Layer:

```
import tensorflow as tf
inputs=keras.Input(shape=(None,), dtype="int64")
embedded= layers.Embedding(input_dim=max_tokens, output_dim=256,
 mask_zero=True)(inputs)
# We have turned mask on because training bi-directional LSTM can take
longer time
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()
```

```
Model: "functional_3"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
```

```
Param # ┃ Connected to           ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━
━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━┫
│ input_layer_1 (InputLayer)      │ (None, None)               │
0 │ -                             │
├─────────────────────────────────┼────────────────────────────┼─────────
────────┼─────────────────────────┤
│ embedding_1 (Embedding)         │ (None, None, 256)          │
2,560,000 │ input_layer_1[0][0]      │
├─────────────────────────────────┼────────────────────────────┼─────────
────────┼─────────────────────────┤
│ not_equal_1 (NotEqual)          │ (None, None)               │
0 │ input_layer_1[0][0]           │
├─────────────────────────────────┼────────────────────────────┼─────────
────────┼─────────────────────────┤
│ bidirectional_1                 │ (None, 64)                 │
73,984 │ embedding_1[0][0],          │
│ (Bidirectional)                 │                            │
│ not_equal_1[0][0]             │
├─────────────────────────────────┼────────────────────────────┼─────────
────────┼─────────────────────────┤
│ dropout_1 (Dropout)             │ (None, 64)                 │
0 │ bidirectional_1[0][0]         │
├─────────────────────────────────┼────────────────────────────┼─────────
────────┼─────────────────────────┤
│ dense_1 (Dense)                 │ (None, 1)                  │
65 │ dropout_1[0][0]                │
└─────────────────────────────────┴────────────────────────────┴─────────
────────┴─────────────────────────┘

 Total params: 2,634,049 (10.05 MB)

 Trainable params: 2,634,049 (10.05 MB)

 Non-trainable params: 0 (0.00 B)
```

Fitting the model on our testing data:

```
model.fit(int_train_ds,
          validation_data=int_val_ds,
          epochs=10)

Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 23s 4s/step - accuracy: 0.5226 - loss: 0.6918
- val_accuracy: 0.4982 - val_loss: 0.6931
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 4s/step - accuracy: 0.6032 - loss: 0.6872
- val_accuracy: 0.5197 - val_loss: 0.6925
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.7837 - loss: 0.6763
```

```
- val_accuracy: 0.5188 - val_loss: 0.6921
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.8556 - loss: 0.6620
- val_accuracy: 0.5247 - val_loss: 0.6916
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.9340 - loss: 0.6511
- val_accuracy: 0.5335 - val_loss: 0.6910
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.9048 - loss: 0.6312
- val_accuracy: 0.5441 - val_loss: 0.6898
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 4s/step - accuracy: 0.9600 - loss: 0.6107
- val_accuracy: 0.5514 - val_loss: 0.6877
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.9847 - loss: 0.5755
- val_accuracy: 0.5562 - val_loss: 0.6855
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 13s 4s/step - accuracy: 0.9630 - loss: 0.5245
- val_accuracy: 0.5554 - val_loss: 0.6813
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.9507 - loss: 0.4501
- val_accuracy: 0.5017 - val_loss: 0.8331

<keras.src.callbacks.history.History at 0x2bebf1874d0>
```

Testing this model:

```
print("\n Model's accuracy:",round (model.evaluate(int_test_ds)
[1]*100,2),"%")

782/782 ━━━━━━━━━━━━━━━━━━━━ 30s 38ms/step - accuracy: 0.5012 - loss:
0.8357

 Model's accuracy: 50.19 %
```

Hence, our first model's accuracy with LSTM and embedding is just 55.44% which is quite low. we will now try a pre-trained word embedding.82

Model Construction - Pretrained word embedded

Parsing after downloading the glove pretrained work-embedding

```
import numpy as np path_to_glove_file =
"C:/Users/varshitha/Downloads/glove.6B/glove.6B.100d.txt"

embeddings_index={}
with open(path_to_glove_file, encoding = "utf-8") as f:
    for line in f:
        words,coefs=line.split(maxsplit=1)
```

```
        coefs = np.fromstring(coefs,"f", sep=" ")
        embeddings_index[words]=coefs
```

Preparing a matrix of GloVe :

```
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
```

Making an embedding layer with this embedded matrix :

```
embedding_layer= layers.Embedding(max_tokens,
                                  embedding_dim,

embeddings_initializer=keras.initializers.
 Constant(embedding_matrix),
                                  trainable=False,
                                  mask_zero=True)
```

Making a final model with pretrained work-embedding :

```
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
("C:/Users/varshitha/Downloads/aclImdb_v1/aclImdb/
glove_embeddings_sequence_model.keras",
                                                  save_best_only=True)
]
```

```
Model: "functional_5"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape                          ┃
```

```
Param # ┃ Connected to          ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━
━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer_2 (InputLayer)     │ (None, None)          │
0 │ -                         │
├────────────────────────────────┼───────────────────────┼─────────
────────┼──────────────────────────────┤
│ embedding_2 (Embedding)        │ (None, None, 100)     │
1,000,000 │ input_layer_2[0][0]       │
├────────────────────────────────┼───────────────────────┼─────────
────────┼──────────────────────────────┤
│ not_equal_2 (NotEqual)         │ (None, None)          │
0 │ input_layer_2[0][0]       │
├────────────────────────────────┼───────────────────────┼─────────
────────┼──────────────────────────────┤
│ bidirectional_2                │ (None, 64)            │
34,048 │ embedding_2[0][0],        │
│ (Bidirectional)                │                       │
│ not_equal_2[0][0]         │
├────────────────────────────────┼───────────────────────┼─────────
────────┼──────────────────────────────┤
│ dropout_2 (Dropout)            │ (None, 64)            │
0 │ bidirectional_2[0][0]     │
├────────────────────────────────┼───────────────────────┼─────────
────────┼──────────────────────────────┤
│ dense_2 (Dense)                │ (None, 1)             │
65 │ dropout_2[0][0]           │
└────────────────────────────────┴───────────────────────┴─────────
────────┴──────────────────────────────┘

 Total params: 1,034,113 (3.94 MB)

 Trainable params: 1,034,113 (3.94 MB)

 Non-trainable params: 0 (0.00 B)
```

Training this model on our dataset :

```
model.fit (int_train_ds, validation_data=int_val_ds , epochs= 10,
callbacks=callbacks)

Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 10s/step - accuracy: 0.4906 - loss:
0.7368 - val_accuracy: 0.4999 - val_loss: 0.7029
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 40s 9s/step - accuracy: 0.5486 - loss: 0.6926
- val_accuracy: 0.5317 - val_loss: 0.6893
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 37s 8s/step - accuracy: 0.6558 - loss: 0.6394
- val_accuracy: 0.5038 - val_loss: 0.7104
```

```
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 28s 9s/step - accuracy: 0.5860 - loss: 0.6914
- val_accuracy: 0.5431 - val_loss: 0.6874
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 27s 9s/step - accuracy: 0.6252 - loss: 0.6620
- val_accuracy: 0.5547 - val_loss: 0.6845
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 24s 8s/step - accuracy: 0.6975 - loss: 0.6273
- val_accuracy: 0.5353 - val_loss: 0.6896
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 27s 9s/step - accuracy: 0.6270 - loss: 0.6577
- val_accuracy: 0.5706 - val_loss: 0.6812
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 39s 8s/step - accuracy: 0.6948 - loss: 0.6232
- val_accuracy: 0.5380 - val_loss: 0.6892
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 54s 18s/step - accuracy: 0.7139 - loss:
0.5944 - val_accuracy: 0.5730 - val_loss: 0.6793
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 53s 17s/step - accuracy: 0.7682 - loss:
0.5537 - val_accuracy: 0.5334 - val_loss: 0.6954

<keras.src.callbacks.history.History at 0x2bed611db50>
```

Testing this Model on our dataset:

```
print("\n Model's Accuracy :", round (model.evaluate (int_test_ds)
[1]*100,2))

782/782 ━━━━━━━━━━━━━━━━━━━━ 199s 253ms/step - accuracy: 0.5301 -
loss: 0.6969

 Model's Accuracy : 53.04
```

Pre-trained embedding is not really helpful in this case. Hence,training from scratch worked better for this dataset. Now, we will try to increase training sample size and then train our model again:

Increase training size by 7000 samples

```
for category in ("neg","pos"):
    files= os.listdir(train_dir/category)
    random.Random(1337).shuffle(files)
    num_train_samples =3500
    train_file = files[:num_train_samples]
    for fname in train_file:
        shutil.move(train_dir/category/fname,
                    train_dir_1/category,fname)
```

Making a training dataset again

```
train =
keras.utils.text_dataset_from_directory(train_dir_1,batch_size=batch_s
ize)
int_train_ds = train.map(
lambda x, y : (text_vectorization(x) , y ), num_parallel_calls=4)

Found 7100 files belonging to 2 classes.
```

Training the last pretrained embedding model with new training dataset :

```
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,
callbacks=callbacks)

Epoch 1/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 139s 621ms/step - accuracy: 0.5844 -
loss: 0.6709 - val_accuracy: 0.6898 - val_loss: 0.5861
Epoch 2/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 86s 385ms/step - accuracy: 0.7198 - loss:
0.5662 - val_accuracy: 0.7718 - val_loss: 0.4806
Epoch 3/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 140s 373ms/step - accuracy: 0.7685 -
loss: 0.4908 - val_accuracy: 0.7652 - val_loss: 0.5023
Epoch 4/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 104s 470ms/step - accuracy: 0.8033 -
loss: 0.4349 - val_accuracy: 0.8094 - val_loss: 0.4181
Epoch 5/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 106s 478ms/step - accuracy: 0.8297 -
loss: 0.3894 - val_accuracy: 0.8149 - val_loss: 0.4129
Epoch 6/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 84s 378ms/step - accuracy: 0.8478 - loss:
0.3579 - val_accuracy: 0.8029 - val_loss: 0.4538
Epoch 7/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 144s 384ms/step - accuracy: 0.8670 -
loss: 0.3169 - val_accuracy: 0.8287 - val_loss: 0.3922
Epoch 8/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 84s 379ms/step - accuracy: 0.8820 - loss:
0.2971 - val_accuracy: 0.8142 - val_loss: 0.4180
Epoch 9/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 85s 381ms/step - accuracy: 0.8914 - loss:
0.2713 - val_accuracy: 0.8266 - val_loss: 0.3963
Epoch 10/10
222/222 ━━━━━━━━━━━━━━━━━━━━ 88s 396ms/step - accuracy: 0.8998 - loss:
0.2497 - val_accuracy: 0.8250 - val_loss: 0.4088

<keras.src.callbacks.history.History at 0x2bed0cd0d10>
```

Testing the model now :

```
print ("\n Model's Acurracy:" ,round(model.evaluate(int_test_ds)
[1]*100,2))
```

Increasing samples did not really increase any acuracy.

Increasing training sample again by 7000

```
for category in ("neg","pos"):
    files= os.listdir(train_dir/category)
    random.Random(1337).shuffle(files)
    num_train_samples = 3500
    train_file = files[:num_train_samples]
    for fname in train_file:
        shutil.move(train_dir/category/fname,
                    train_dir_1/category,fname)
```

Reading a new training set:

```
train =
keras.utils.text_dataset_from_directory(train_dir_1,batch_size=batch_s
ize)
int_train_ds = train.map(
lambda x, y: (text_vectorization(x), y), num_parallel_calls=4)

Found 14100 files belonging to 2 classes.
```

Training this model again

```
 model.fit(int_train_ds, validation_data=int_val_ds, epochs=10)

Epoch 1/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 197s 445ms/step - accuracy: 0.8671 -
loss: 0.3212 - val_accuracy: 0.8352 - val_loss: 0.3775
Epoch 2/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 162s 366ms/step - accuracy: 0.8837 -
loss: 0.2852 - val_accuracy: 0.8148 - val_loss: 0.4872
Epoch 3/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 151s 341ms/step - accuracy: 0.8965 -
loss: 0.2641 - val_accuracy: 0.8386 - val_loss: 0.3732
Epoch 4/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 145s 329ms/step - accuracy: 0.9093 -
loss: 0.2357 - val_accuracy: 0.8429 - val_loss: 0.4187
Epoch 5/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 147s 334ms/step - accuracy: 0.9153 -
loss: 0.2159 - val_accuracy: 0.8393 - val_loss: 0.4293
Epoch 6/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 142s 321ms/step - accuracy: 0.9274 -
loss: 0.1928 - val_accuracy: 0.8322 - val_loss: 0.4755
Epoch 7/10
```

```
441/441 ━━━━━━━━━━━━━━━━━━━━ 143s 322ms/step - accuracy: 0.9361 -
loss: 0.1704 - val_accuracy: 0.8375 - val_loss: 0.4606
Epoch 8/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 143s 324ms/step - accuracy: 0.9441 -
loss: 0.1482 - val_accuracy: 0.8333 - val_loss: 0.5175
Epoch 9/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 157s 356ms/step - accuracy: 0.9544 -
loss: 0.1259 - val_accuracy: 0.8307 - val_loss: 0.5537
Epoch 10/10
441/441 ━━━━━━━━━━━━━━━━━━━━ 142s 322ms/step - accuracy: 0.9641 -
loss: 0.1095 - val_accuracy: 0.8253 - val_loss: 0.6179

<keras.src.callbacks.history.History at 0x2beebdb2d10>
```

Testing this model:

```
print("\n Model's Accuracy:",round(model.evaluate(int_test_ds)
[1]*100,2))

514/782 ━━━━━━━━━━━━━━━━━━━━ 21s 79ms/step - accuracy: 0.8294 - loss:
0.6172
```