Machine Learning Project Serie 1:

IMDB Movie Review Sentiment Classification

Episode 4: Attention Mechanism

This episode focuses on fitting and testing data with Recurrent Neural Network (LSTM variant) combined with the simplified version of Attention Mechanism. Word embedding matrix is still being used.

I. Importing Libraries

In [1]:
```
import numpy as np
import os
import pathlib
import tensorflow as tf
from tensorflow.keras import regularizers
from keras.layers import Bidirectional, Concatenate, Permute, Dot, Input, LSTM, Multiply, Embedding, Reshape, I
from keras.layers import RepeatVector, Dense, Activation, Lambda, Softmax, SimpleRNN
from keras.optimizers import Adam, SGD
from keras.utils import to_categorical
from keras.models import load_model, Model
import keras.backend as K
import keras
```

II. Extracting Data

In [2]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.imdb.load_data()
```

```
<__array_function__ internals>:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If yo
u meant to do this, you must specify 'dtype=object' when creating the ndarray
/Users/tieubinh03/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/datasets/imdb.py:159: Visib
leDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tu
ples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dt
ype=object' when creating the ndarray
  x_train, y_train = np.array(x_train[idx]), np.array(labels[idx])
/Users/tieubinh03/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/datasets/imdb.py:160: Visib
leDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tu
ples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dt
ype=object' when creating the ndarray
  x_test, y_test = np.array(x_test[idx]), np.array(labels[idx])
```

In [3]:
```
word_dict = tf.keras.datasets.imdb.get_word_index(path="imdb_word_index.json")
```

In [4]:
```
vocab_len = len(word_dict)
print("Total words count:", vocab_len)
```

```
Total words count: 88584
```

III. Data Preprocessing

In [124...
```
chosen_cmt_len = 200
max_index = 20000

def padding(initial_x):
    output = np.zeros((chosen_cmt_len))
    for i in range(chosen_cmt_len):
        if i < len(initial_x) and initial_x[i] < max_index:
            output[i] = initial_x[i]
        else:
            output[i] = 0
    return output
```

In [125...
```
x_train_padded = np.zeros((len(x_train), chosen_cmt_len))
for i in range(len(x_train)):
    x_train_padded[i] = padding(x_train[i])
```

In [126...
```
x_test_padded = np.zeros((len(x_test), chosen_cmt_len))
for i in range(len(x_test)):
    x_test_padded[i] = padding(x_test[i])
```

IV. Machine Learning Model:

In [127...
```
h_s = 20
e_s = 20
```

In [128...
```
# Creating model:
def model():

    X_input = Input(shape=(chosen_cmt_len,))

    embedding = Embedding(max_index, e_s)(X_input)

    drop = Dropout(0.95)(embedding)

    score = LSTM(chosen_cmt_len, activation="softmax")(drop)

    dot = Dot(axes=1)([embedding, score])

    drop = Dropout(0.95)(dot)

    output = Dense(1, activation='sigmoid',
                   kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
                   bias_regularizer=regularizers.l2(1e-4),
                   activity_regularizer=regularizers.l2(1e-5)
                  )(drop)

    model = Model(inputs = X_input, outputs = output)

    return model
```

In [129...
```
model = model()
model.summary()
```

```
Model: "model_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_21 (InputLayer)        [(None, 200)]             0         
_________________________________________________________________
embedding_20 (Embedding)     (None, 200, 20)           400000    
_________________________________________________________________
dropout_39 (Dropout)         (None, 200, 20)           0         
_________________________________________________________________
lstm_14 (LSTM)               (None, 200)               176800    
_________________________________________________________________
dot_20 (Dot)                 (None, 20)                0         
_________________________________________________________________
dropout_40 (Dropout)         (None, 20)                0         
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 21        
=================================================================
Total params: 576,821
Trainable params: 576,821
Non-trainable params: 0
_________________________________________________________________
```

In [130...
```
# Optimizer for the model
learning_rate = 3e-3
opt = Adam(lr=learning_rate, decay=1e-6)
model.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=[tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", threshold=0.5)])
```

In [131...
```
# Storing histories
histories = []
testings  = []

# Track testing accuracy
curr_acc = 0
curr_acc = 0.01

# Max testing accuracy
max_acc = 0

# Fitting and evaluating the model after epochs
epoch = 1

# Keep training as long as testing accuracy on testing set is still increasing
while epoch < 21 or prev_acc < curr_acc:
    # Fitting
    print("Epoch:", epoch)
    print("Fitting data:")
    history = model.fit(x = x_train_padded, y = np.array(y_train).reshape(25000, 1), epochs=1, batch_size=1000)

    # Evaluating
    print("Testing data:")
    testing = model.evaluate(x_test_padded, np.array(y_test).reshape(25000, 1))

    # Assigning max accuracy
    if testing[1] > max_acc:
        max_acc = testing[1]

    # Assigning test accuracy
    prev_acc = curr_acc
    curr_acc = testing[1]

    # Adjust learning rate
    if prev_acc > curr_acc:
        learning_rate /= 10
        opt = Adam(lr=learning_rate, decay=1e-5)
        model.compile(optimizer=opt,
                      loss='binary_crossentropy',
                      metrics=[tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", threshold=0.5)])

    # Storing
    histories.append(history)
    testings.append(testing)

    epoch += 1
    print("\n")

print("Optimal testing accuracy is: {:.2f}%".format(max_acc * 100))
```

```
Epoch: 1
Fitting data:
25/25 [==============================] - 74s 3s/step - loss: 0.6926 - binary_accuracy: 0.5111
Testing data:
782/782 [==============================] - 48s 62ms/step - loss: 0.6895 - binary_accuracy: 0.6783


Epoch: 2
Fitting data:
25/25 [==============================] - 72s 3s/step - loss: 0.6857 - binary_accuracy: 0.5996
Testing data:
782/782 [==============================] - 53s 68ms/step - loss: 0.6809 - binary_accuracy: 0.7223


Epoch: 3
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.6731 - binary_accuracy: 0.6260
Testing data:
782/782 [==============================] - 56s 72ms/step - loss: 0.6636 - binary_accuracy: 0.7371


Epoch: 4
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.6518 - binary_accuracy: 0.6509
Testing data:
782/782 [==============================] - 55s 71ms/step - loss: 0.6366 - binary_accuracy: 0.7529


Epoch: 5
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.6248 - binary_accuracy: 0.6672
Testing data:
782/782 [==============================] - 51s 65ms/step - loss: 0.6037 - binary_accuracy: 0.7698


Epoch: 6
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.5991 - binary_accuracy: 0.6850
Testing data:
782/782 [==============================] - 50s 64ms/step - loss: 0.5699 - binary_accuracy: 0.7968


Epoch: 7
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.5727 - binary_accuracy: 0.6979
Testing data:
782/782 [==============================] - 50s 64ms/step - loss: 0.5379 - binary_accuracy: 0.8086


Epoch: 8
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.5515 - binary_accuracy: 0.7103
Testing data:
782/782 [==============================] - 51s 65ms/step - loss: 0.5101 - binary_accuracy: 0.8244


Epoch: 9
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.5313 - binary_accuracy: 0.7175
Testing data:
782/782 [==============================] - 51s 65ms/step - loss: 0.4857 - binary_accuracy: 0.8335


Epoch: 10
Fitting data:
25/25 [==============================] - 74s 3s/step - loss: 0.5174 - binary_accuracy: 0.7240
Testing data:
782/782 [==============================] - 51s 65ms/step - loss: 0.4649 - binary_accuracy: 0.8394


Epoch: 11
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.5044 - binary_accuracy: 0.7284
Testing data:
782/782 [==============================] - 51s 65ms/step - loss: 0.4476 - binary_accuracy: 0.8435


Epoch: 12
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4932 - binary_accuracy: 0.7372
Testing data:
782/782 [==============================] - 51s 65ms/step - loss: 0.4337 - binary_accuracy: 0.8446


Epoch: 13
Fitting data:
25/25 [==============================] - 74s 3s/step - loss: 0.4856 - binary_accuracy: 0.7356
Testing data:
782/782 [==============================] - 51s 66ms/step - loss: 0.4213 - binary_accuracy: 0.8492


Epoch: 14
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4767 - binary_accuracy: 0.7402
Testing data:
782/782 [==============================] - 52s 66ms/step - loss: 0.4120 - binary_accuracy: 0.8523


Epoch: 15
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4688 - binary_accuracy: 0.7478
Testing data:
782/782 [==============================] - 52s 66ms/step - loss: 0.4028 - binary_accuracy: 0.8525


Epoch: 16
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4647 - binary_accuracy: 0.7471
Testing data:
782/782 [==============================] - 52s 66ms/step - loss: 0.3964 - binary_accuracy: 0.8556


Epoch: 17
Fitting data:
25/25 [==============================] - 74s 3s/step - loss: 0.4568 - binary_accuracy: 0.7519
Testing data:
782/782 [==============================] - 52s 66ms/step - loss: 0.3909 - binary_accuracy: 0.8550


Epoch: 18
Fitting data:
25/25 [==============================] - 74s 3s/step - loss: 0.4566 - binary_accuracy: 0.7547
Testing data:
782/782 [==============================] - 54s 69ms/step - loss: 0.3899 - binary_accuracy: 0.8555


Epoch: 19
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4522 - binary_accuracy: 0.7526
Testing data:
782/782 [==============================] - 54s 69ms/step - loss: 0.3891 - binary_accuracy: 0.8559


Epoch: 20
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4512 - binary_accuracy: 0.7507
Testing data:
782/782 [==============================] - 53s 68ms/step - loss: 0.3884 - binary_accuracy: 0.8564


Epoch: 21
Fitting data:
25/25 [==============================] - 73s 3s/step - loss: 0.4521 - binary_accuracy: 0.7518
Testing data:
782/782 [==============================] - 54s 69ms/step - loss: 0.3878 - binary_accuracy: 0.8560


Optimal testing accuracy is: 85.64%
```

V. Summary:

LSTM with simplified Attention mechanism took longer to train but has lower testing score. My prediction is that the data given was quite structually simple, hence, the structure of LSTM or other kinds of Recurrent Neural Network over-complicated the relationship between features and labels. The model was also tested with GRU, and SimpleRNN (not shown), but none of those achieved higher testing score.

	Loss	Accuracy	Sample size
Training (with Dropout)	0.45	75.2%	25,000
Testing	0.33	85.6%	25,000

Training score is lower than testing score as the Dropout was not turned off. Next project episode will turn off Dropout on final training evaluation.

VIII. Thank you:

Thank you for viewing my project. See you in the next episode.