Machine Learning Project Serie 1:

IMDB Movie Review Sentiment Classification

Episode 5: Convolutional Neural Network

This episode focuses on fitting and testing data with the Convolutional Neural Network. Word embedding technique 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, Flatten
from keras.layers import RepeatVector, Dense, Activation, Lambda, Softmax, Conv1D
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()
```

Out[2]:
```
<__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 you meant to do this, you must specify 'dtype=object' when creating the ndarray
/Users/tieubinh33/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/datasets/imdb.py:159: 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 you meant to do this, you must specify 'dtype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/Users/tieubinh33/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/datasets/imdb.py:160: 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 you meant to do this, you must specify 'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[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)
```

Out[4]:
```
Total words count: 88584
```

III. Data Preprocessing

In [5]:
```
chosen_cmt_len = 2000
max_index = 25000

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 [6]:
```
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 [7]:
```
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 [8]:
```
e_s = 20
```

In [9]:
```
# Creating model:
def model():

    # Retrieving inputs
    X_input = Input(shape=(chosen_cmt_len,))

    # Embedding meanings
    embedding = Embedding(max_index, e_s)(X_input)

    drop = Dropout(0.9)(embedding)

    conv_1 = Conv1D(filters=1, kernel_size=5, strides=1, activation='tanh', padding='causal')(drop)

    drop = Dropout(0.9)(conv_1)

    flatten = Flatten()(drop)

    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)
                  )(flatten)

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

    return model
```

In [10]:
```
model = model()
model.summary()
```

Out[10]:
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 2000)]            0
_________________________________________________________________
embedding (Embedding)        (None, 2000, 20)          500000
_________________________________________________________________
dropout (Dropout)            (None, 2000, 20)          0
_________________________________________________________________
conv1d (Conv1D)              (None, 2000, 1)           101
_________________________________________________________________
dropout_1 (Dropout)          (None, 2000, 1)           0
_________________________________________________________________
flatten (Flatten)            (None, 2000)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 2001
=================================================================
Total params: 502,102
Trainable params: 502,102
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
# Optimizer for the model
learning_rate = 5e-3
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)])
```

In [15]:
```
# Storing histories
histories = []
testings  = []

# Track testing accuracy
prev_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:
    # 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))
```

Out[15]:
```
Epoch: 1
Fitting data:
25/25 [==============================] - 20s 758ms/step - loss: 0.7088 - binary_accuracy: 0.4986
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.6934 - binary_accuracy: 0.5176


Epoch: 2
Fitting data:
25/25 [==============================] - 20s 801ms/step - loss: 0.6968 - binary_accuracy: 0.5058
Testing data:
782/782 [==============================] - 6s 7ms/step - loss: 0.6927 - binary_accuracy: 0.5280


Epoch: 3
Fitting data:
25/25 [==============================] - 19s 754ms/step - loss: 0.6917 - binary_accuracy: 0.5269
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.6824 - binary_accuracy: 0.7065


Epoch: 4
Fitting data:
25/25 [==============================] - 19s 763ms/step - loss: 0.6509 - binary_accuracy: 0.6144
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.5462 - binary_accuracy: 0.7737


Epoch: 5
Fitting data:
25/25 [==============================] - 20s 778ms/step - loss: 0.5459 - binary_accuracy: 0.7244
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.4200 - binary_accuracy: 0.8412


Epoch: 6
Fitting data:
25/25 [==============================] - 20s 769ms/step - loss: 0.4937 - binary_accuracy: 0.7660
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.3943 - binary_accuracy: 0.8682


Epoch: 7
Fitting data:
25/25 [==============================] - 20s 777ms/step - loss: 0.4633 - binary_accuracy: 0.7852
Testing data:
782/782 [==============================] - 7s 8ms/step - loss: 0.3659 - binary_accuracy: 0.8757


Epoch: 8
Fitting data:
25/25 [==============================] - 19s 751ms/step - loss: 0.4385 - binary_accuracy: 0.7990
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.3564 - binary_accuracy: 0.8793


Epoch: 9
Fitting data:
25/25 [==============================] - 19s 756ms/step - loss: 0.4319 - binary_accuracy: 0.8054
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3515 - binary_accuracy: 0.8817


Epoch: 10
Fitting data:
25/25 [==============================] - 20s 790ms/step - loss: 0.4141 - binary_accuracy: 0.8121
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3388 - binary_accuracy: 0.8839


Epoch: 11
Fitting data:
25/25 [==============================] - 20s 776ms/step - loss: 0.4070 - binary_accuracy: 0.8187
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.3416 - binary_accuracy: 0.8765


Epoch: 12
Fitting data:
25/25 [==============================] - 20s 764ms/step - loss: 0.3945 - binary_accuracy: 0.8264
Testing data:
782/782 [==============================] - 6s 8ms/step - loss: 0.3354 - binary_accuracy: 0.8843


Epoch: 13
Fitting data:
25/25 [==============================] - 19s 776ms/step - loss: 0.3940 - binary_accuracy: 0.8236
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3347 - binary_accuracy: 0.8839


Epoch: 14
Fitting data:
25/25 [==============================] - 20s 764ms/step - loss: 0.3945 - binary_accuracy: 0.8243
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3335 - binary_accuracy: 0.8854


Epoch: 15
Fitting data:
25/25 [==============================] - 19s 777ms/step - loss: 0.3876 - binary_accuracy: 0.8291
Testing data:
782/782 [==============================] - 7s 8ms/step - loss: 0.3335 - binary_accuracy: 0.8856


Epoch: 16
Fitting data:
25/25 [==============================] - 20s 775ms/step - loss: 0.3956 - binary_accuracy: 0.8228
Testing data:
782/782 [==============================] - 7s 8ms/step - loss: 0.3333 - binary_accuracy: 0.8858


Epoch: 17
Fitting data:
25/25 [==============================] - 20s 778ms/step - loss: 0.3934 - binary_accuracy: 0.8272
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3332 - binary_accuracy: 0.8858


Epoch: 18
Fitting data:
25/25 [==============================] - 20s 785ms/step - loss: 0.3905 - binary_accuracy: 0.8281
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3332 - binary_accuracy: 0.8858


Epoch: 19
Fitting data:
25/25 [==============================] - 19s 778ms/step - loss: 0.3949 - binary_accuracy: 0.8246
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3332 - binary_accuracy: 0.8860


Epoch: 20
Fitting data:
25/25 [==============================] - 20s 790ms/step - loss: 0.3923 - binary_accuracy: 0.8251
Testing data:
782/782 [==============================] - 7s 9ms/step - loss: 0.3332 - binary_accuracy: 0.8859


Optimal testing accuracy is: 88.60%
```

In [16]:
```
model.evaluate(x_train_padded, np.array(y_train).reshape(25000, 1))
```

Out[16]:
```
782/782 [==============================] - 6s 7ms/step - loss: 0.2474 - binary_accuracy: 0.9342
[0.24780705571174622, 0.9339600205421448]
```

V. Summary:

Convolutional Network performed the best on a similar number of parameters to other models' (approx 500,000 parameters) and very little training time.

-	Loss	Accuracy	Sample size
Training	1.25	93.4%	25,000
Testing	0.33	88.6%	25,000

The data in the given sample seems to have simple general structures as shallower neural networks work much better than deep ones (this was a sub-test not being shown in this report).

VIII. Thank you:

Thank you for viewing my project. This is the final episode of this serie. One of the important things I learnt from this serie was that deeper neural networks do not always mean better performances. If the depth and complexity of the network is increased exceeding some specific threshold, it will hurt performance. The right model should be chosen based on analysis of the complexity and structure of the given data. At the beginning of the project, I was trying to create a model that can generalize well for data from various distributions, but that model-centric approach seems to be not as efficient as I thought, data-centric might still be a more useful approach in most cases, as stated by Professor Andrew Ng.

That's it for this serie. Hope to see you in the next serie.