Machine Learning Project Serie 1:

IMDB Movie Review Sentiment Classification

Sub-Episode 2.1: Expanding Neural Networks in Keras

This episode focuses on transfering the model to Keras and expanding the previously used techniques (fully connected neural networks) on much larger number of features on each datapoint.

I. Importing Libraries

In [1]:
```
import numpy as np
import os
import pathlib
import time
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
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(xs[: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(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)
```

```
Total words count: 88584
```

In [5]:
```
word_list = list(dict(sorted(word_dict.items(), key=lambda item: item[1])))
word_list.insert(0, str(0)) # To make the index of each word matches it position in original dictionary
```

In [6]:
```
print("10 most used words:", word_list[:10])
```

```
10 most used words: ['0', 'the', 'and', 'a', 'of', 'to', 'is', 'br', 'in', 'it']
```

III. Data Preprocessing

In [7]:
```
max_features = 20000  # Choose maximum n most used words
required_len = 1    # But exclude the words that does not match required length

def choose_word_index():
    chosen_indexes = []
    i = 0
    while len(chosen_indexes) < max_features:
        if len(word_list[i]) >= required_len:
            chosen_indexes.append(i)
        i += 1
    return chosen_indexes
```

In [8]:
```
accounted_word_indexes = choose_word_index()
nof_features = len(accounted_word_indexes)
print("Number of words picked: " + str(nof_features))
```

```
Number of words picked: 20000
```

In [9]:
```
def construct_input(initial_input):
    new_input = []
    for word_index in accounted_word_indexes:
        if word_index in initial_input:
            new_input.append(1)
        else:
            new_input.append(0)
    return new_input
```

In [10]:
```
# TRAINING SET:
start_time = time.time()

new_train_len = len(x_train)
new_x_train     = np.zeros((new_train_len, nof_features))
new_y_train     = np.zeros((new_train_len, 1))

for i in range(new_train_len):
    new_x_train[i] = construct_input(x_train[i])
    new_y_train[i] =                 y_train[i]

print("Processing training set took approximately", round((time.time() - start_time)/60), "minutes.\n")
```

```
Processing training set took approximately 25 minutes.
```

In [11]:
```
# TESTING SET:
start_time = time.time()

new_test_len = len(x_test)
new_x_test     = np.zeros((new_test_len, nof_features))
new_y_test     = np.zeros((new_test_len, 1))

for i in range(new_test_len):
    new_x_test[i] = construct_input(x_test[i])
    new_y_test[i] =                 y_test[i]

print("Processing testing set took approximately", round((time.time() - start_time)/60), "minutes.\n")
```

```
Processing testing set took approximately 24 minutes.
```

IV. Machine Learning Model:

The model is also expanded in the number of hidden layers and units.

In [133…
```
def model_NN():
    X_input = Input(shape=(nof_features))

    X = Dense(500, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-5)
                    )(X_input)

    X = Dense(250, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-5)
                    )(X)

    X = Dense(125, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-5)
                    )(X)

    X = 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)
                    )(X)

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

    return model
```

In [134…
```
model_NN = model_NN()
model_NN.summary()
```

```
Model: "model_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_25 (InputLayer)        [(None, 20000)]           0
_________________________________________________________________
dense_119 (Dense)            (None, 500)               10000500
_________________________________________________________________
dense_120 (Dense)            (None, 250)               125250
_________________________________________________________________
dense_121 (Dense)            (None, 125)               31375
_________________________________________________________________
dense_122 (Dense)            (None, 1)                 126
=================================================================
Total params: 10,157,251
Trainable params: 10,157,251
Non-trainable params: 0
_________________________________________________________________
```

V. Training model:

In [135…
```
# Optimizer for the model
opt_1 = Adam(lr=0.05, decay=1e-6)

model_NN.compile(optimizer=opt_1,
                 loss='binary_crossentropy',
                 metrics=[tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", threshold=0.5)])
```

In [136…
```
start_time = time.time()

model_NN.fit(new_x_train, np.array(y_train).reshape(25000, 1), epochs=20, batch_size=1000)

print("Training model took approximately", round((time.time() - start_time)/60), "minutes.\n")
```

```
Epoch 1/20
25/25 [==============================] - 12s 398ms/step - loss: 36.8182 - binary_accuracy: 0.5193
Epoch 2/20
25/25 [==============================] - 9s 349ms/step - loss: 10.2172 - binary_accuracy: 0.8067
Epoch 3/20
25/25 [==============================] - 9s 342ms/step - loss: 5.1671 - binary_accuracy: 0.9367
Epoch 4/20
25/25 [==============================] - 9s 344ms/step - loss: 3.3625 - binary_accuracy: 0.9478
Epoch 5/20
25/25 [==============================] - 9s 353ms/step - loss: 2.7739 - binary_accuracy: 0.9384
Epoch 6/20
25/25 [==============================] - 9s 371ms/step - loss: 2.3077 - binary_accuracy: 0.9511
Epoch 7/20
25/25 [==============================] - 9s 370ms/step - loss: 2.0912 - binary_accuracy: 0.9433
Epoch 8/20
25/25 [==============================] - 11s 434ms/step - loss: 1.8806 - binary_accuracy: 0.9523
Epoch 9/20
25/25 [==============================] - 9s 349ms/step - loss: 2.2256 - binary_accuracy: 0.8752
Epoch 10/20
25/25 [==============================] - 9s 376ms/step - loss: 1.9624 - binary_accuracy: 0.9350
Epoch 11/20
25/25 [==============================] - 10s 391ms/step - loss: 1.7057 - binary_accuracy: 0.9513
Epoch 12/20
25/25 [==============================] - 12s 479ms/step - loss: 1.5638 - binary_accuracy: 0.9546
Epoch 13/20
25/25 [==============================] - 11s 450ms/step - loss: 1.5529 - binary_accuracy: 0.9520
Epoch 14/20
25/25 [==============================] - 9s 369ms/step - loss: 1.6798 - binary_accuracy: 0.9218
Epoch 15/20
25/25 [==============================] - 9s 360ms/step - loss: 1.4655 - binary_accuracy: 0.9547
Epoch 16/20
25/25 [==============================] - 9s 352ms/step - loss: 1.3746 - binary_accuracy: 0.9502
Epoch 17/20
25/25 [==============================] - 10s 380ms/step - loss: 1.6426 - binary_accuracy: 0.9166
Epoch 18/20
25/25 [==============================] - 9s 372ms/step - loss: 1.5991 - binary_accuracy: 0.9510
Epoch 19/20
25/25 [==============================] - 9s 355ms/step - loss: 1.5518 - binary_accuracy: 0.9446
Epoch 20/20
25/25 [==============================] - 9s 355ms/step - loss: 1.4361 - binary_accuracy: 0.9545
Training model took approximately 3 minutes.
```

VI. Testing model:

The model was chosen as best performance after a few tries so there is no need to split-up training set and validate it.

In [137…
```
model_NN.evaluate(new_x_test, np.array(y_test).reshape(25000, 1))
```

```
782/782 [==============================] - 20s 25ms/step - loss: 1.5770 - binary_accuracy: 0.8591
```

Out[137…
```
[1.577048420906067, 0.8591200113296509]
```

VII. Summary:

-	Loss	Accuracy	Sample size
Training	1.43	95%	25,000
Testing	1.58	86%	25,000

VIII. Thank you:

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