Practical No.: 02

Classification using Deep neural network (Any One from the following): Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition datasethttps://archive.ics.uci.edu/ml/datasets/letter+recognition

In [1]:
```
import numpy as np
from keras.datasets import imdb
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics


import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Load the data, keeping only 10,000 of the most frequently occuring words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 1
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
imdb.npz
17464789/17464789 [==============================] - 6s 0us/step
```

In [3]:
```
train_data[:2]
```

Out[3]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4,
173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 17
2, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 5
0, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 53
0, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12,
8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619,
5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407,
16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530,
476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071,
56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 553
5, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5,
16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4,
715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14,
69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9,
35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 9
52, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11,
3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123,
125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120,
5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5,
4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 138
2, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 7
8, 285, 16, 145, 95])],
      dtype=object)
```

In [4]:
```
train_labels
```

Out[4]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [5]:
```
# Check the first label
train_labels[0]
```

Out[5]:
```
1
```

In [6]:

```
# Here is a list of maximum indexes in every review --- we search the maximum index
print(type([max(sequence) for sequence in train_data]))

# Find the maximum of all max indexes
max([max(sequence) for sequence in train_data])
```

Out[6]:

```
<class 'list'>
9999
```

In [7]:

```
# step 1: load the dictionary mappings from word to integer index
word_index = imdb.get_word_index()

# step 2: reverse word index to map integer indexes to their respective words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

# Step 3: decode the review, mapping integer indices to words
#
# indices are off by 3 because 0, 1, and 2 are reserverd indices for "padding", "St
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]]

decoded_review
```

Out[7]:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
imdb_word_index.json
1641221/1641221 [==============================] - 1s 1us/step
"? this film was just brilliant casting location scenery story direction everyon
e's really suited the part they played and you could just imagine being there robe
rt ? is an amazing actor and now the same being director ? father came from the sa
me scottish island as myself so i loved the fact there was a real connection with
this film the witty remarks throughout the film were great it was just brilliant s
o much that i bought the film as soon as it was released for ? and would recommend
it to everyone to watch and the fly fishing was amazing really cried at the end it
was so sad and you know what they say if you cry at a film it must have been good
and this definitely was also ? to the two little boy's that played the ? of norman
and paul they were just brilliant children are often left out of the ? list i thin
k because the stars that play them all grown up are such a big profile for the who
le film but these children are amazing and should be praised for what they have do
ne don't you think the whole story was so lovely because it was true and was someo
ne's life after all that was shared with us all"
```

In [8]:

```
len(reverse_word_index)
```

Out[8]:

```
88584
```

In [9]:

```
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an all zero matrix
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                        # Sets specific indices of r
    return results

# Vectorize training Data
X_train = vectorize_sequences(train_data)

# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

In [10]:

```
X_train[0]
```

Out[10]:

```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [11]:

```
X_train.shape
```

Out[11]:

```
(25000, 10000)
```

In [12]:

```
y_train = np.asarray(train_labels).astype('float32')
y_test  = np.asarray(test_labels).astype('float32')
```

In [13]:

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [14]:

```
model.compile(
    optimizer=optimizers.RMSprop(learning_rate=0.001),
    loss = losses.binary_crossentropy,
    metrics = [metrics.binary_accuracy]
)
```

In [15]:

```
# Input for Validation
X_val = X_train[:10000]
partial_X_train = X_train[10000:]

# Labels for validation
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [16]:

```
history = model.fit(
    partial_X_train,
    partial_y_train,
    epochs=20,
    batch_size=512,
    validation_data=(X_val, y_val)
)
```

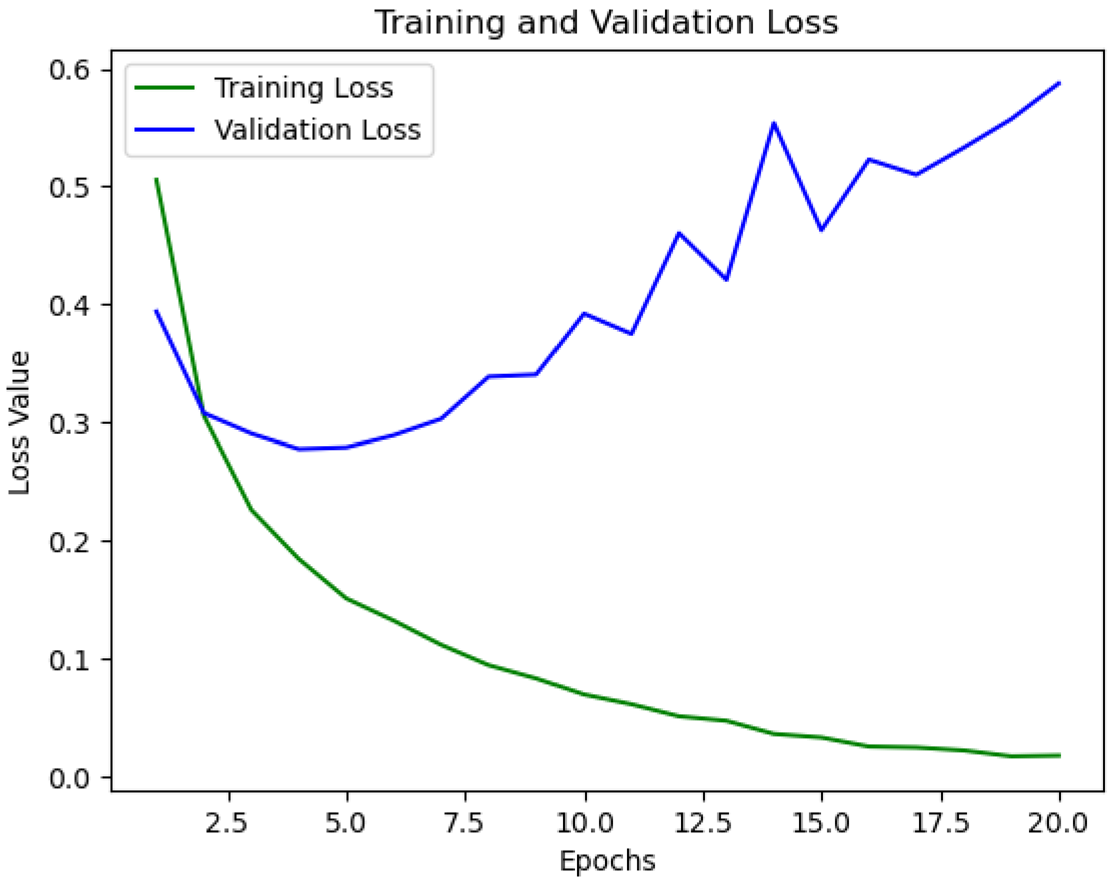
```
Epoch 1/20
30/30 [==============================] - 6s 170ms/step - loss: 0.5055 - binary_acc
uracy: 0.7803 - val_loss: 0.3939 - val_binary_accuracy: 0.8508
Epoch 2/20
30/30 [==============================] - 1s 26ms/step - loss: 0.3056 - binary_accu
racy: 0.8947 - val_loss: 0.3079 - val_binary_accuracy: 0.8823
Epoch 3/20
30/30 [==============================] - 1s 26ms/step - loss: 0.2260 - binary_accu
racy: 0.9250 - val_loss: 0.2905 - val_binary_accuracy: 0.8847
Epoch 4/20
30/30 [==============================] - 1s 25ms/step - loss: 0.1844 - binary_accu
racy: 0.9376 - val_loss: 0.2774 - val_binary_accuracy: 0.8893
Epoch 5/20
30/30 [==============================] - 1s 26ms/step - loss: 0.1509 - binary_accu
racy: 0.9517 - val_loss: 0.2788 - val_binary_accuracy: 0.8873
Epoch 6/20
30/30 [==============================] - 1s 26ms/step - loss: 0.1320 - binary_accu
racy: 0.9557 - val_loss: 0.2890 - val_binary_accuracy: 0.8873
Epoch 7/20
30/30 [==============================] - 1s 27ms/step - loss: 0.1115 - binary_accu
racy: 0.9639 - val_loss: 0.3029 - val_binary_accuracy: 0.8858
Epoch 8/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0942 - binary_accu
racy: 0.9715 - val_loss: 0.3387 - val_binary_accuracy: 0.8752
Epoch 9/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0830 - binary_accu
racy: 0.9745 - val_loss: 0.3404 - val_binary_accuracy: 0.8814
Epoch 10/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0694 - binary_accu
racy: 0.9814 - val_loss: 0.3920 - val_binary_accuracy: 0.8671
Epoch 11/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0611 - binary_accu
racy: 0.9842 - val_loss: 0.3748 - val_binary_accuracy: 0.8785
Epoch 12/20
30/30 [==============================] - 1s 28ms/step - loss: 0.0508 - binary_accu
racy: 0.9881 - val_loss: 0.4604 - val_binary_accuracy: 0.8665
Epoch 13/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0470 - binary_accu
racy: 0.9883 - val_loss: 0.4208 - val_binary_accuracy: 0.8748
Epoch 14/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0359 - binary_accu
racy: 0.9927 - val_loss: 0.5537 - val_binary_accuracy: 0.8562
Epoch 15/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0329 - binary_accu
racy: 0.9929 - val_loss: 0.4628 - val_binary_accuracy: 0.8736
Epoch 16/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0252 - binary_accu
racy: 0.9961 - val_loss: 0.5228 - val_binary_accuracy: 0.8688
Epoch 17/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0244 - binary_accu
racy: 0.9957 - val_loss: 0.5099 - val_binary_accuracy: 0.8711
Epoch 18/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0219 - binary_accu
racy: 0.9958 - val_loss: 0.5329 - val_binary_accuracy: 0.8704
Epoch 19/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0169 - binary_accu
racy: 0.9977 - val_loss: 0.5571 - val_binary_accuracy: 0.8692
Epoch 20/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0175 - binary_accu
racy: 0.9965 - val_loss: 0.5874 - val_binary_accuracy: 0.8645
```

In [17]:

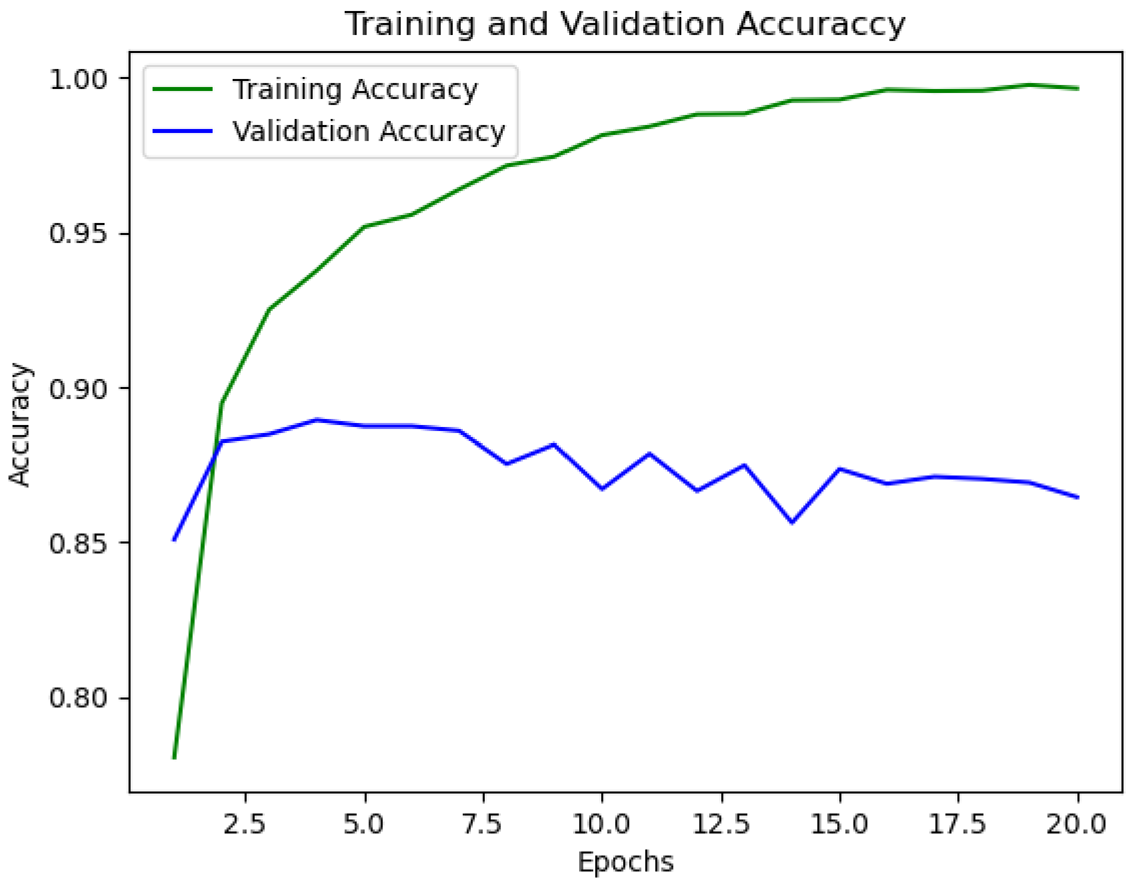
```
history_dict = history.history
history_dict.keys()
```

```
Out[17]: dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

```
In [18]: # Plotting losses
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'g', label="Training Loss")
plt.plot(epochs, val_loss_values, 'b', label="Validation Loss")

plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss Value')
plt.legend()

plt.show()
```



```
In [19]: # Training and Validation Accuracy

acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, acc_values, 'g', label="Training Accuracy")
plt.plot(epochs, val_acc_values, 'b', label="Validation Accuracy")

plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [20]:

```
model.fit(
    partial_X_train,
    partial_y_train,
    epochs=3,
    batch_size=512,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/3
30/30 [==============================] - 2s 79ms/step - loss: 0.0159 - binary_accu
racy: 0.9963 - val_loss: 0.5995 - val_binary_accuracy: 0.8699
Epoch 2/3
30/30 [==============================] - 1s 23ms/step - loss: 0.0076 - binary_accu
racy: 0.9998 - val_loss: 0.6424 - val_binary_accuracy: 0.8684
Epoch 3/3
30/30 [==============================] - 1s 21ms/step - loss: 0.0160 - binary_accu
racy: 0.9957 - val_loss: 0.6401 - val_binary_accuracy: 0.8683
```

Out[20]:

```
<keras.src.callbacks.History at 0x22730e844d0>
```

In [21]:

```
# Making Predictions for testing data
np.set_printoptions(suppress=True)
result = model.predict(X_test)
```

```
782/782 [==============================] - 3s 4ms/step
```

In [22]:

```
result
```

Out[22]:

```
array([[0.01144991],
       [1.        ],
       [0.8956549 ],
       ...,
       [0.00239313],
       [0.02422373],
       [0.9610478 ]], dtype=float32)
```

In [23]:

```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = np.round(score)
```

In [24]:

```
mae = metrics.mean_absolute_error(y_pred, y_test)
mae
```

Out[24]:

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.14284>
```