DL EXPERIMENT NO 3

Getting started with neural networks: Classification and regression

Classifying movie reviews: A binary classification example

The IMDB dataset

This is a dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

Loading the IMDB dataset

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000) #nt
```

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

```
train_data[0]
```

```
[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,
      172,
      112,
      167,
      2,
      336,
      385,
      39,
      4,
      172,
      4536,
      1111,
      17,
      546,
      38,
      13,
      447,
      4,
      192,
      50,
      16,
      6,
      147,
      2025,
```

```
train_labels[0]
```

```
    1
```

```
max([max(sequence) for sequence in train_data])
```

```
    9999
```

Decoding reviews back to text

```
word_index = imdb.get_word_index()#i-3 because reserve
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
```

```
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/im
1641221/1641221 [==============================] - 0s 0us/step
```

```
decoded_review
```

```
'? this film was just brilliant casting location scenery story direction everyone's r
y suited the part they played and you could just imagine being there robert ? is an a
ng actor and now the same being director ? father came from the same scottish island
yself so i loved the fact there was a real connection with this film the witty remark
roughout the film were great it was just brilliant so much that i bought the film as
as it was released for ? and would recommend it to everyone to watch and the fly fish
was amazing really cried at the end it was so sad and you know what they say if you c
t a film it must have been good and this definitely was also ? to the two little boy
```

Preparing the data

Encoding the integer sequences via multi-hot encoding

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

```
x_train[0]
```

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

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

Building your model

Model definition

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
model = keras.Sequential([
    layers.Dense(10, activation="relu"),
    layers.Dense(5, activation="relu"),
   # layers.Dense(5, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

Compiling the model

```
model.compile(optimizer="SGD",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

Validating your approach

Setting aside a validation set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training your model

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=15,
                    batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/15
118/118 [==============================] - 3s 18ms/step - loss: 0.6885 - accuracy: 0
Epoch 2/15
118/118 [==============================] - 2s 13ms/step - loss: 0.6708 - accuracy: 0
Epoch 3/15
118/118 [==============================] - 2s 19ms/step - loss: 0.6411 - accuracy: 0
Epoch 4/15
118/118 [==============================] - 2s 15ms/step - loss: 0.6011 - accuracy: 0
Epoch 5/15
118/118 [==============================] - 1s 12ms/step - loss: 0.5548 - accuracy: 0
Epoch 6/15
118/118 [==============================] - 1s 11ms/step - loss: 0.5044 - accuracy: 0
Epoch 7/15
118/118 [==============================] - 2s 14ms/step - loss: 0.4558 - accuracy: 0
Epoch 8/15
118/118 [==============================] - 1s 12ms/step - loss: 0.4138 - accuracy: 0
Epoch 9/15
118/118 [==============================] - 1s 11ms/step - loss: 0.3800 - accuracy: 0
Epoch 10/15
118/118 [==============================] - 1s 12ms/step - loss: 0.3535 - accuracy: 0
```

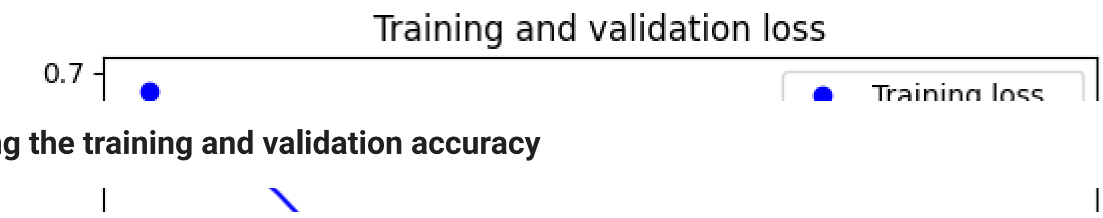
```
Epoch 11/15
118/118 [==============================] - 2s 18ms/step - loss: 0.3326 - accuracy: 0
Epoch 12/15
118/118 [==============================] - 2s 14ms/step - loss: 0.3153 - accuracy: 0
Epoch 13/15
118/118 [==============================] - 1s 12ms/step - loss: 0.2999 - accuracy: 0
Epoch 14/15
118/118 [==============================] - 2s 14ms/step - loss: 0.2873 - accuracy: 0
Epoch 15/15
118/118 [==============================] - 2s 14ms/step - loss: 0.2756 - accuracy: 0
```

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

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

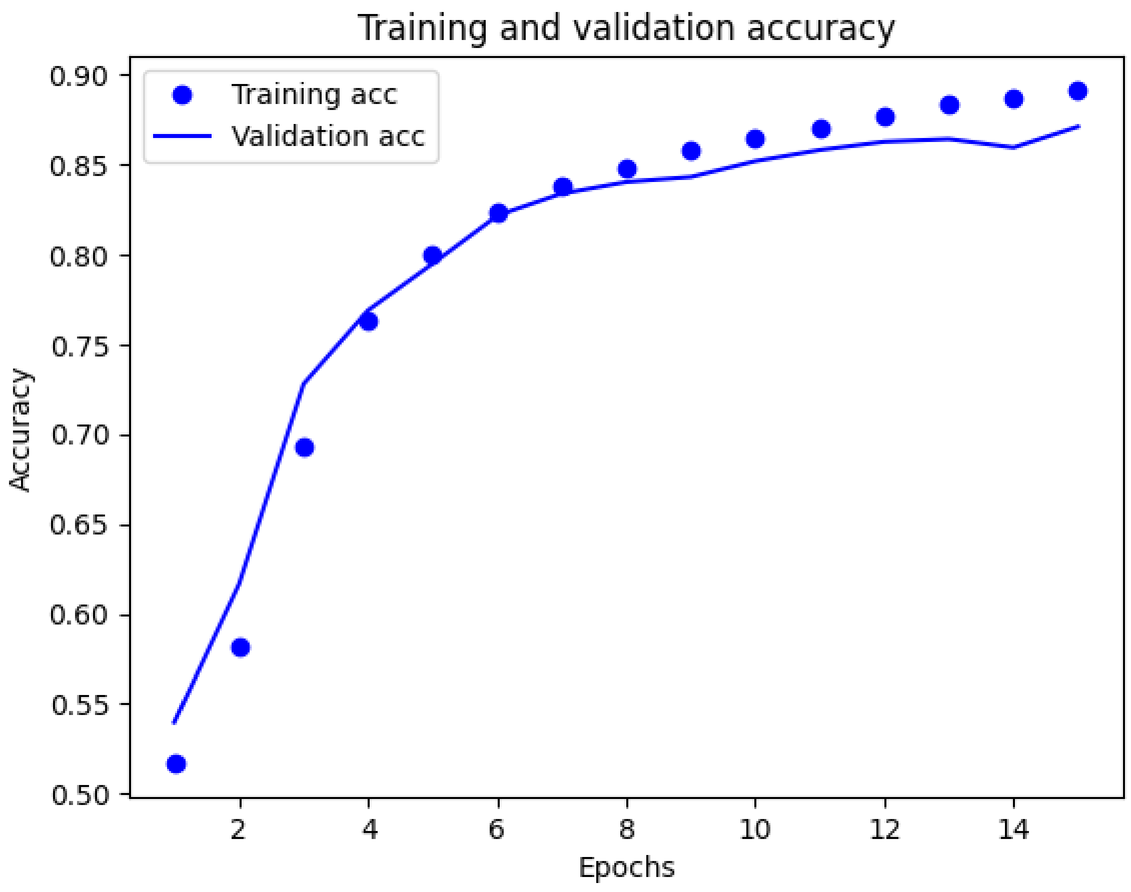
Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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")
plt.legend()
plt.show()
```



Plotting the training and validation accuracy

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



how is the model doing? Comment Improve the performance of the model by tuning the hyperparameters and report the set of best tuned hyperparameters.

Retraining a model from scratch

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 28ms/step - loss: 0.5539 - accuracy: 0.74
Epoch 2/4
49/49 [==============================] - 2s 37ms/step - loss: 0.3452 - accuracy: 0.89
Epoch 3/4
49/49 [==============================] - 2s 39ms/step - loss: 0.2500 - accuracy: 0.91
Epoch 4/4
49/49 [==============================] - 1s 28ms/step - loss: 0.2039 - accuracy: 0.92
782/782 [==============================] - 2s 2ms/step - loss: 0.2803 - accuracy: 0.8
```

```
results
```

```
[0.28030282258987427, 0.8880000114440918]
```

Using a trained model to generate predictions on new data

```
model.predict(x_test)
```

```
782/782 [==============================] - 2s 2ms/step
array([[0.23631732],
       [0.99252677],
       [0.90398574],
       ...,
       [0.14931314],
       [0.0904747 ],
       [0.5785891 ]], dtype=float32)
```

Further experiments

Classifying newswires: A multiclass classification example

The Reuters dataset

Loading the Reuters dataset

```
from tensorflow.keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reu
2110848/2110848 [==============================] - 0s 0us/step
```

```
#print the length of train data
print("train_data ", train_data.shape)
print("train_labels ", train_labels.shape)
```

```
train_data  (8982,)
train_labels  (8982,)
```

```
#print the length of test data
print("test_data ", test_data.shape)
print("test_labels ", test_labels.shape)
```

```
test_data  (2246,)
test_labels  (2246,)
```

```
#display any one training example
train_data[0]
```

```
[1,
 2,
 2,
 8,
 43,
 10,
 447,
 5,
 25,
 207,
 270,
 5,
 3095,
 111,
 16,
 369,
 186,
 90,
 67,
 7,
 89,
 5,
 19,
 102,
 6,
 19,
 124,
 15,
 90,
 67,
 84,
 22,
 482,
 26,
 7,
        48,
        4,
        49,
        8,
        864,
        39,
        209,
        154,
        6,
        151,
        6,
        83,
        11,
        15,
        22,
        155,
        11,
        15,
        7,
        48,
        9,
        4579,
        1005,
```

Decoding newswires back to text

```
#decode newswires back to test and display any one review
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in
train_data[0]])

print(decoded_review)
```

```
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reu
    550378/550378 [==============================] - 0s 0us/step
    ? ? ? said as a result of its december acquisition of space co it expects earnings pe
```

```
#display label of training example displayed above
print(train_labels[0])
```

```
    3
```

Preparing the data

Encoding the input data

```
#encode the input data by vectorization
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results


# Vectorize and Normalize train and test to tensors with 10k columns
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Encoding the labels

```
from keras.utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

Building your model

Model definition

```
#create your model from scratch
from keras import models
from keras import layers

model = models.Sequential([
    layers.Dense(16, activation="relu", input_shape=(10000,)),
    layers.Dense(16, activation="relu"),
    layers.Dense(46, activation="softmax") ##46 because the output has 46 mutually exclusi
])
```

Compiling the model

```
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

Validating your approach

Setting aside a validation set

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

```
print("x_val ", x_val.shape)
print("y_val ", y_val.shape)

print("partial_x_train ", partial_x_train.shape)
print("partial_y_train ", partial_y_train.shape)
```

```
x_val  (1000, 10000)
y_val  (1000, 46)
partial_x_train  (7982, 10000)
partial_y_train  (7982, 46)
```

Training the model

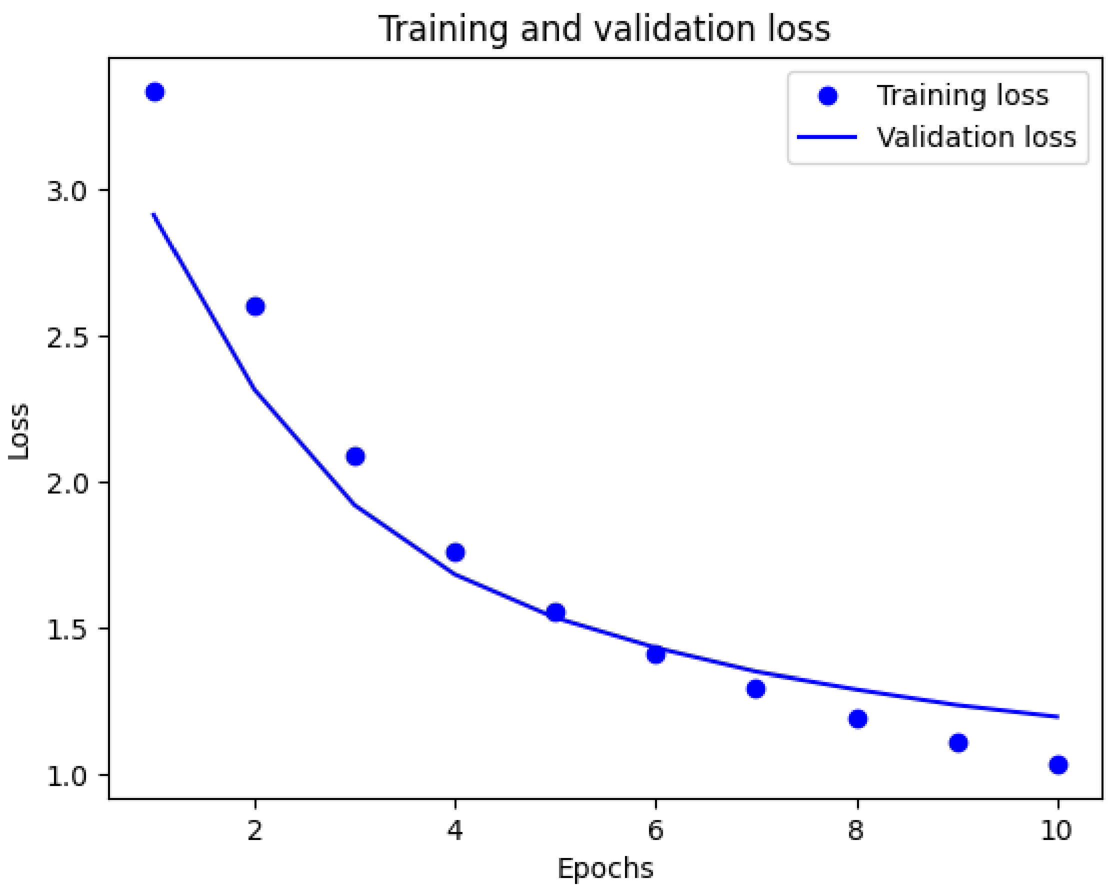
```
history = model.fit(partial_x_train, partial_y_train,
                    epochs=10, batch_size=512, validation_data=(x_val, y_val))

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

```
Epoch 1/10
16/16 [==============================] - 2s 62ms/step - loss: 3.3393 - accuracy: 0.22
Epoch 2/10
16/16 [==============================] - 1s 33ms/step - loss: 2.6034 - accuracy: 0.46
Epoch 3/10
16/16 [==============================] - 1s 34ms/step - loss: 2.0932 - accuracy: 0.56
Epoch 4/10
16/16 [==============================] - 1s 33ms/step - loss: 1.7657 - accuracy: 0.63
Epoch 5/10
16/16 [==============================] - 1s 35ms/step - loss: 1.5576 - accuracy: 0.64
Epoch 6/10
16/16 [==============================] - 1s 32ms/step - loss: 1.4110 - accuracy: 0.66
Epoch 7/10
16/16 [==============================] - 1s 33ms/step - loss: 1.2948 - accuracy: 0.69
Epoch 8/10
16/16 [==============================] - 1s 34ms/step - loss: 1.1957 - accuracy: 0.72
Epoch 9/10
16/16 [==============================] - 0s 31ms/step - loss: 1.1112 - accuracy: 0.74
Epoch 10/10
16/16 [==============================] - 1s 32ms/step - loss: 1.0367 - accuracy: 0.75
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

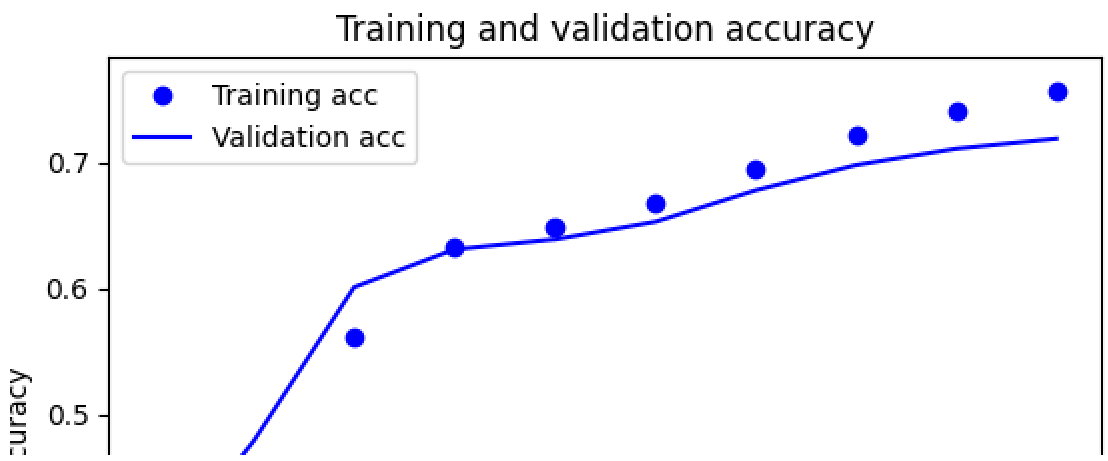
Plotting the training and validation loss

```
plt.clf()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Plotting the training and validation accuracy

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Here we can observe the validation and training accuracy is reaching only upto 75% which isnt a optimum number hence we have to tune the hyperparameters to get the optimum accuracy

Retraining a model from scratch by tuning the hyperparameters and report the results

```
model = keras.Sequential([
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(46, activation="sigmoid")
])

model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
model.fit(partial_x_train, partial_y_train, epochs=10, batch_size=128)

results = model.evaluate(x_test, one_hot_test_labels)
```

```
Epoch 1/10
63/63 [==============================] - 2s 27ms/step - loss: 1.6506 - accuracy: 0.6
Epoch 2/10
63/63 [==============================] - 2s 27ms/step - loss: 0.8440 - accuracy: 0.8
Epoch 3/10
63/63 [==============================] - 3s 44ms/step - loss: 0.5222 - accuracy: 0.8
Epoch 4/10
63/63 [==============================] - 2s 28ms/step - loss: 0.3426 - accuracy: 0.9
Epoch 5/10
63/63 [==============================] - 2s 27ms/step - loss: 0.2541 - accuracy: 0.9
Epoch 6/10
63/63 [==============================] - 2s 27ms/step - loss: 0.2128 - accuracy: 0.9
Epoch 7/10
63/63 [==============================] - 2s 27ms/step - loss: 0.1771 - accuracy: 0.9
Epoch 8/10
63/63 [==============================] - 2s 26ms/step - loss: 0.1679 - accuracy: 0.9
Epoch 9/10
63/63 [==============================] - 2s 27ms/step - loss: 0.1526 - accuracy: 0.9
Epoch 10/10
63/63 [==============================] - 3s 44ms/step - loss: 0.1461 - accuracy: 0.9
71/71 [==============================] - 0s 4ms/step - loss: 1.0814 - accuracy: 0.79
```

Generating predictions on new data

```
results
```

```
    [1.0813590288162231, 0.7934104800224304]
```

Summary

1] In this experiment, we performed the binary as well as multi-class classification on the imdb and reuters dataset.

2] To get to the optimum result, firstly we decoded the newswires, and also encoded the input data as well as output data (hot label encoding for multiclass classification). After splitting into respective validation and testing dataset, we defined the model and trained it, and plotted the graph to analyze the loss and accuracy.

3] By tuning the hyperparameters like batch_size, epochs, no of neurons in hidden layer, we got the best optimum results at the end,