In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

In [3]:
```
# Here in train_data the no.s represent the first movie review where each word corresponds to a word
train_data[0]
```

Out[3]:
```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
```

In [4]:
```
train_labels[0]
```

Out[4]:
```
1
```

In [5]:
```
test_data[0]
```

Out[5]:
```
[1,
 591,
 202,
 14,
 31,
 6,
 717,
 10,
 10,
 2,
 2,
 5,
 4,
 360,
 7,
 4,
 177,
 5760,
 394,
```

In [6]:
```
test_labels[0]
```

Out[6]:
```
0
```

WORD TO INDEX MAPPINGS

1. WORD_INDEX

In [7]:
```
word_index = imdb.get_word_index()
word_index
```

Out[7]:
```
{'fawn': 34701,
 'tsukino': 52006,
 'nunnery': 52007,
 'sonja': 16816,
 'vani': 63951,
 'woods': 1408,
 'spiders': 16115,
 'hanging': 2345,
 'woody': 2289,
 'trawling': 52008,
 "hold's": 52009,
 'comically': 11307,
 'localized': 40830,
 'disobeying': 30568,
 "'royale": 52010,
 "harpo's": 40831,
 'canet': 52011,
 'aileen': 19313,
 'acurately': 52012,
```

2. REVERSE_WORD_INDEX

In [8]:
```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
reverse_word_index
```

Out[8]:
```
{34701: 'fawn',
 52006: 'tsukino',
 52007: 'nunnery',
 16816: 'sonja',
 63951: 'vani',
 1408: 'woods',
 16115: 'spiders',
 2345: 'hanging',
 2289: 'woody',
 52008: 'trawling',
 52009: "hold's",
 11307: 'comically',
 40830: 'localized',
 30568: 'disobeying',
 52010: "'royale",
 40831: "harpo's",
 52011: 'canet',
 19313: 'aileen',
 52012: 'acurately',
```

3.DECODED_REVIEWS

In [9]:
```
decoded_reviews = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])
decoded_reviews
```

Out[9]:
```
"? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from the same 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 so 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 a t 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 think because the stars that play them all grown up are such a big profile for the whole film but these children are a mazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"
```

CREATING x_train, x_test, y_train, y_test

In [10]:
```
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
```

In [11]:
```
x_train = vectorize_sequences(train_data);
x_test = vectorize_sequences(test_data);
```

In [12]:
```
x_train
```

Out[12]:
```
array([[0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       ...,
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.]])
```

In [13]:
```
x_test
```

Out[13]:
```
array([[0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       ...,
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.]])
```

In [14]:
```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [15]:
```
y_train
```

Out[15]:
```
array([1., 0., 0., ..., 0., 1., 0.], dtype=float32)
```

In [16]:
```
y_test
```

Out[16]:
```
array([0., 1., 1., ..., 0., 0., 0.], dtype=float32)
```

Creating Model for NN architecture

In [17]:
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model
```

Out[17]:
```
<Sequential name=sequential, built=False>
```

Add layers to Model

In [18]:
```
model.add(Dense(16, input_shape=(10000, ), activation="relu", name="dense_1"))
model.add(Dense(16, activation="relu", name="dense_2"))
model.add(Dense(1, activation="sigmoid", name="dense_output"))
```

```
D:\py\Lib\site-packages\keras\src\layers\core\dense.py:88: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input
(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Compile to model(setting config for later training)

In [19]:
```
from keras import optimizers
from keras import losses
from keras import metrics

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

In [20]:
```
model.summary()
```

Model: "sequential"

Lay… (ty…	… …	… #
den… (De…	… …	…
den… (De…	… …	…
den… (De…	… …	…

Total params: 160,305 (626.19 KB)

Trainable params: 160,305 (626.19 KB)

Non-trainable params: 0 (0.00 B)

In [22]:
```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [24]:
```
history = model.fit(partial_x_train, partial_y_train, epochs = 20, validation_data=(x_val, y_val), batch_size=512, verbose = 1)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 609ms/step - binary_accuracy: 0.6914 - loss: 0.6196 - val_binary_accuracy: 0.8569 - val_loss: 0.4384
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - binary_accuracy: 0.8835 - loss: 0.3836 - val_binary_accuracy: 0.8810 - val_loss: 0.3356
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - binary_accuracy: 0.9132 - loss: 0.2753 - val_binary_accuracy: 0.8746 - val_loss: 0.3116
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - binary_accuracy: 0.9289 - loss: 0.2173 - val_binary_accuracy: 0.8828 - val_loss: 0.2902
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - binary_accuracy: 0.9412 - loss: 0.1833 - val_binary_accuracy: 0.8897 - val_loss: 0.2746
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - binary_accuracy: 0.9535 - loss: 0.1522 - val_binary_accuracy: 0.8869 - val_loss: 0.2810
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - binary_accuracy: 0.9583 - loss: 0.1364 - val_binary_accuracy: 0.8852 - val_loss: 0.2850
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - binary_accuracy: 0.9672 - loss: 0.1130 - val_binary_accuracy: 0.8826 - val_loss: 0.2968
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9743 - loss: 0.0953 - val_binary_accuracy: 0.8858 - val_loss: 0.3072
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - binary_accuracy: 0.9763 - loss: 0.0840 - val_binary_accuracy: 0.8819 - val_loss: 0.3418
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - binary_accuracy: 0.9823 - loss: 0.0692 - val_binary_accuracy: 0.8822 - val_loss: 0.3395
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9868 - loss: 0.0592 - val_binary_accuracy: 0.8762 - val_loss: 0.3641
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9891 - loss: 0.0519 - val_binary_accuracy: 0.8794 - val_loss: 0.3756
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - binary_accuracy: 0.9931 - loss: 0.0402 - val_binary_accuracy: 0.8758 - val_loss: 0.3997
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9944 - loss: 0.0344 - val_binary_accuracy: 0.8739 - val_loss: 0.4320
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9960 - loss: 0.0287 - val_binary_accuracy: 0.8751 - val_loss: 0.4400
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9974 - loss: 0.0224 - val_binary_accuracy: 0.8725 - val_loss: 0.4628
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9984 - loss: 0.0190 - val_binary_accuracy: 0.8730 - val_loss: 0.5040
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - binary_accuracy: 0.9992 - loss: 0.0152 - val_binary_accuracy: 0.8710 - val_loss: 0.5083
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - binary_accuracy: 0.9992 - loss: 0.0131 - val_binary_accuracy: 0.8717 - val_loss: 0.5296
```

In [26]:
```
mse_nn, mae_nn = model.evaluate(x_test, y_test)
print(mse_nn, " ", mae_nn)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - binary_accuracy: 0.8565 - loss: 0.5862
0.5738868713378906   0.8602799773216248
```

In [ ]: