Assignment – 4

Task: Analyse text and Sequence data using Recurrent Neural networks.

As per the instructions provided, using the below conditions,

1. Cutoff reviews after 150 words
2. Restrict training samples to 100
3. Validate on 10,000 samples
4. Consider only the top 10,000 words
5. Consider both an embedding layer, and a pretrained word embedding.

This task is divided into three main categories:

1. Utilizing a custom-designed Recurrent Neural Network (RNN) and applying it to the IMDB dataset obtained from the web.

2. Employing a pre-trained embedding layer, such as Glove, which is a technique for embedding words. In this task, the Glove implementation is utilized.

3. Lastly, assessing performance by utilizing Long Short-Term Memory (LSTM) networks, considering their effectiveness across various scenarios.

Custom Designed RNN:

The implemented architecture is straightforward. We utilized an embedding layer followed by a flatten layer.

```
!pip install keras_preprocessing
from keras.datasets import imdb
from keras import preprocessing

from keras_preprocessing.sequence import pad_sequences

max_features = 10000 # nr of words to consider as features
maxlen = 150 # cuts off the text after this nr of words among the most common words, i.e. 'max_features'
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words =max_features) # loads the data as list of integers


x_train = x_train[:100]
y_train = y_train[:100]

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

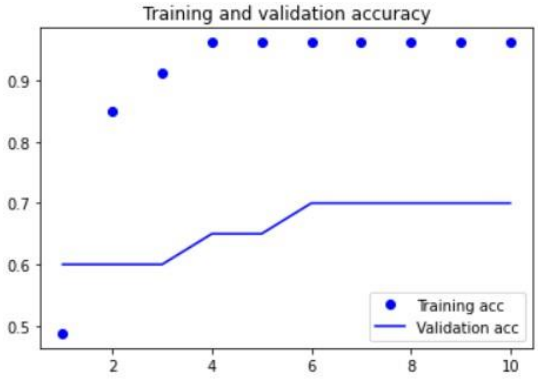
I used the Keras implementation to design the model with sequential layers including embedding, flattening, and dense layers.

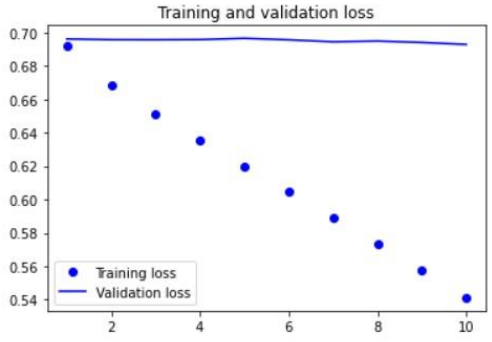
I employed the RMSprop optimizer, binary cross-entropy loss function, and accuracy as the metric.

After training for 10 epochs, the following results were obtained.

```
Epoch 10/10
3/3 [==============================] - 0s 12ms/step - loss: 0.5412 - acc: 0.9625 - val_loss: 0.6928 - val_acc: 0.7000
```

Below are the plots for the above task,





From the plots, it's evident that the model performs better on the training data. Notably, the validation accuracy remains consistent from epoch 6 to epoch 10, indicating a decent result.

Utilizing GloVe Embedding:

GloVe, which stands for Global Vectors for Word Representation, aims to capture semantic relationships between words in a text corpus by representing each word as a vector in a high-dimensional space. The GloVe algorithm utilizes co-occurrence statistics to learn these vector representations. It calculates co-occurrence probabilities between word pairs in a corpus and then applies matrix factorization to map these probabilities into a low-dimensional space. The resulting vectors encode the underlying relationships between words in the corpus.

I downloaded the GloVe file from the web and imported it into my code.

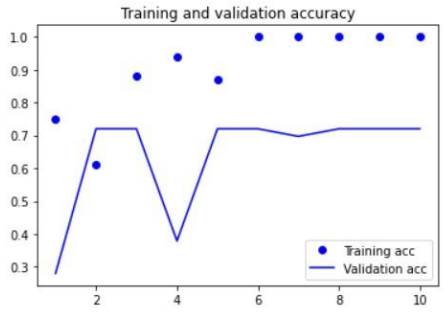
```
embedding_dim = 100 # GloVe contains 100-dimensional embedding vectors for 400.000 words

embedding_matrix = np.zeros((max_words, embedding_dim)) # embedding_matrix.shape (10000, 100)
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word) # embedding_vector.shape (100,)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector # Words not found in the mebedding index will all be
```

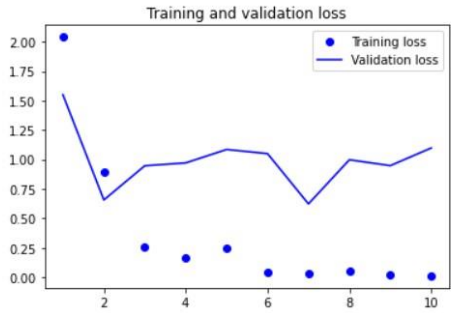
Model Definition

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length = maxlen))
model.add(Flatten())
model.add(Dense(32, activation = "relu"))
model.add(Dense(1, activation="sigmoid"))
model.summary()
```

And the results are below,

```
Epoch 10/10
4/4 [==============================] - 1s 188ms/step - loss: 0.0118 - acc: 1.0000 - val_loss: 1.0974 - val_acc: 0.7199
```





The results obtained were inferior compared to the custom-designed architecture. Perhaps by refining the structure or enhancing the design, we could achieve better outcomes.

Technique	**Training Accuracy**	**Validation Accuracy**	**Training Loss**	**Validation Loss**
Custom designed RNN	0.9625	0.7000	0.5412	0.6928
Using Glove Embedding	1.0000	0.7199	0.0118	1.0974

Based on the comparison above, it's evident that the GloVe embedding model exhibits slightly better validation accuracy than the custom-designed model.

Next, I increased the training sample size to 200 and applied the same process to both the embedded and pre-trained embedded models. The table below displays the results for both models:

Technique	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
Custom designed RNN	1.0000	0.6500	0.5422	0.6977
Using Glove Embedding	1.0000	0.7068	0.0173	0.6352

With the increase in training samples to 200, the validation accuracy of both the custom-designed model and the GloVe embedding model has significantly decreased compared to the models trained with 100 samples.

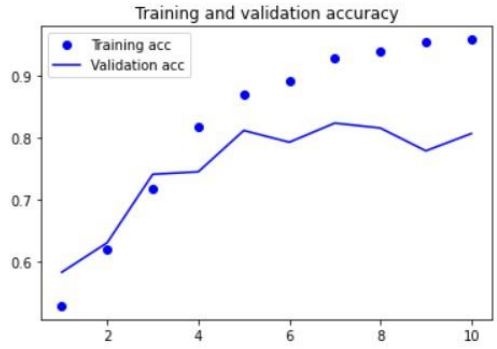
Utilizing LSTMs:

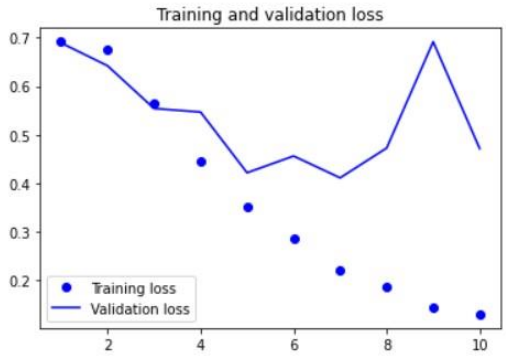
LSTMs, or Long Short-Term Memory networks, are designed to address the vanishing gradient problem encountered in traditional RNNs. This problem occurs when gradients become extremely small during backpropagation, making it challenging for the network to learn long-term dependencies.

```
from keras.models import Sequential
from keras.layers import Embedding
from keras.layers import LSTM
from keras.layers import Flatten, Dense

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation = "sigmoid"))

model.compile(optimizer = "rmsprop",
              loss = "binary_crossentropy",
              metrics = ["acc"])
history = model.fit(input_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
```

I incorporated LSTM into the design, and below are the accuracy results of the model:

Technique	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
Custom designed RNN	96.29	86.70	0.1090	0.345





Conclusion:

Training the models with a sample size of 100 resulted in better accuracy for both the custom-designed and GloVe embedding models compared to training with 200 samples. This suggests that increasing the training sample size leads to overfitting.

Comparatively, the GloVe embedding model outperformed the custom-designed model. This can be attributed to GloVe embedding being pre-trained on high-quality data, providing better performance.

Additionally, I implemented LSTM (Long Short-Term Memory) networks, which yielded the best results among all the models tested. Several factors may contribute to this, including differences in the training sample and the functioning of the model compared to others.