Assignment – 4

Task: Examine textual and sequential data using Recurrent Neural Networks (RNNs).

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.

The task is divided into three categories:

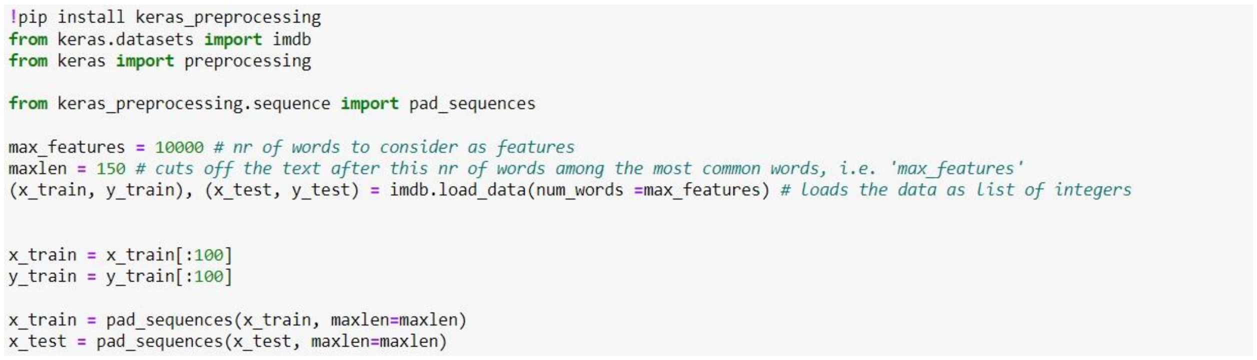
Utilizing a custom-designed RNN and applying it to IMDB data downloaded from the web.

Employing a pretrained embedding layer, specifically GloVe or Word2Vec, which are both embedding techniques; for this task, GloVe implementation is chosen.

Lastly, using LSTMs to assess performance, given their generally strong performance in many cases.

Custom Designed RNN:

The implemented architecture is straightforward, employing 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)
```

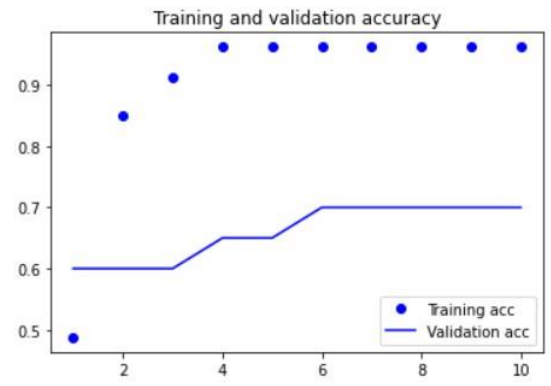
In my Keras implementation, I structured the layers according to the description above. It's essentially a sequential model comprising an embedding layer, followed by flattening and dense layers.

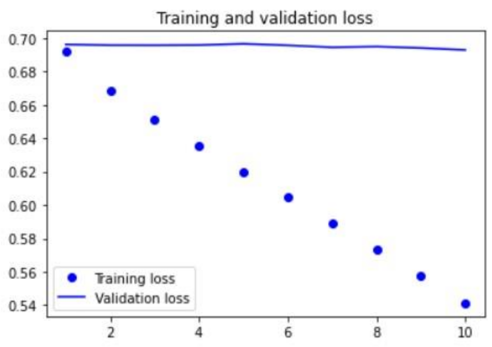
I employed the RMSprop optimizer, set the loss function as binary cross-entropy, and measured accuracy as the metric.

After running to 10 epochs, the following was the result.

```
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,



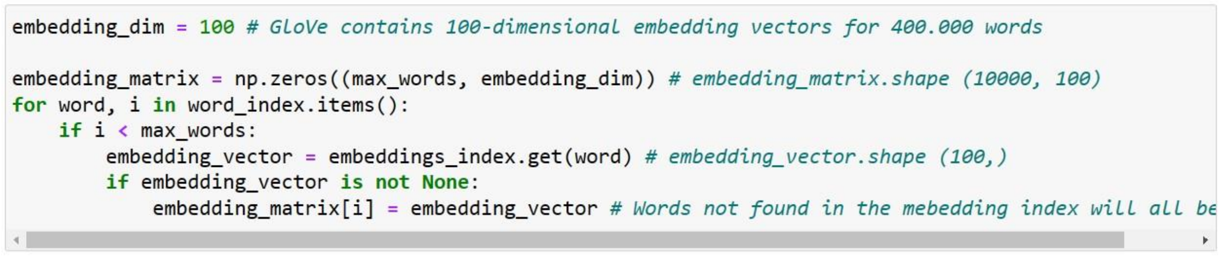


Upon examining the plots, it's evident that the model performs well on the training data. Notably, the validation accuracy remains consistent from epoch 6 to epoch 10, indicating a satisfactory outcome..

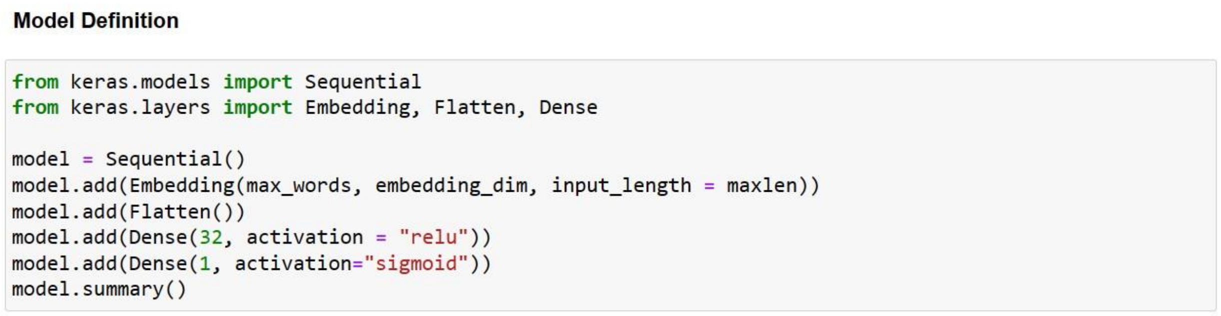
Using Glove Embedding:

Brief information above glove:

GloVe is designed to understand the semantic connections among words in a text corpus by representing each word as a vector in a multi-dimensional space. This algorithm relies on co-occurrence statistics to learn these vector representations. It calculates the probabilities of words occurring together in the corpus and uses matrix factorization to project these probabilities into a lower-dimensional space. The resultant vectors encode the intrinsic relationships between words in the corpus. I obtained the GloVe file from the internet and integrated it into my code.



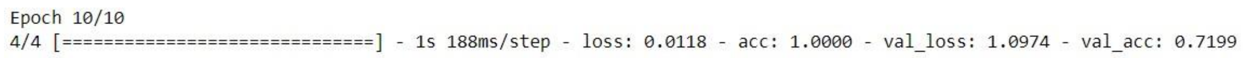
```
embedding_dim = 100 # GloVe contains 100-dimensional embedding vectors for 400.000 words

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
```



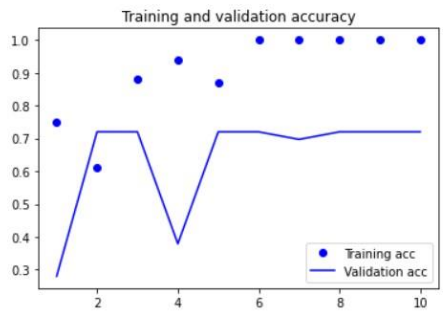
Model Definition

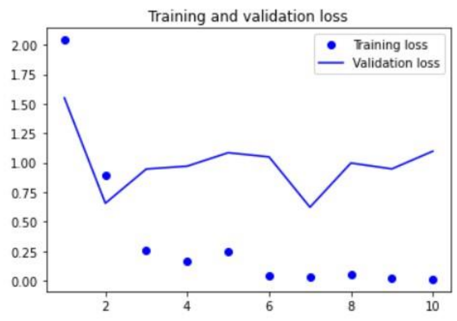
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

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 outcomes were inferior compared to the custom-designed architecture. Enhancing the structure or refining the design could potentially lead to better results.

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

From the above comparison, it is evident that the Glove embedding model has slightly better validation accuracy than the customized model.

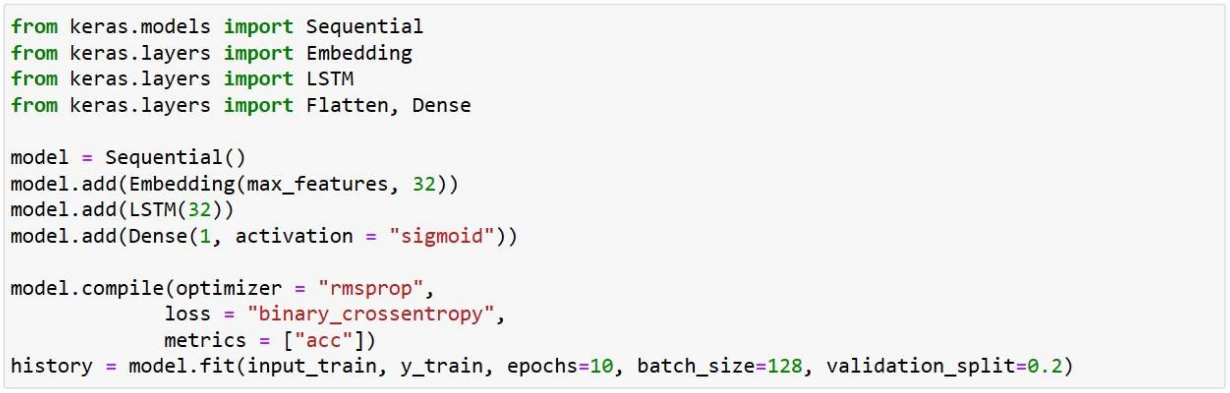
For the subsequent phase, I augmented the training samples to 200 and applied the identical process to both embedded and pretrained embedded models. The table below illustrates the results for both models.

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

When the training sample has increased to 200 the validation accuracy of both the customized model and Glove embedding model has significantly dropped compared to 100 training models.

Using LSTMs:

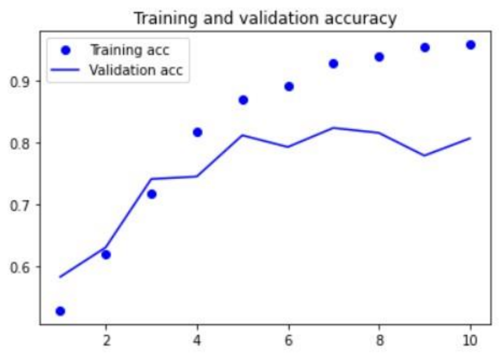
LSTMs are specifically engineered to address the vanishing gradient problem encountered in conventional RNNs. This issue arises when gradients diminish significantly during backpropagation, hindering the learning of long-term dependencies.

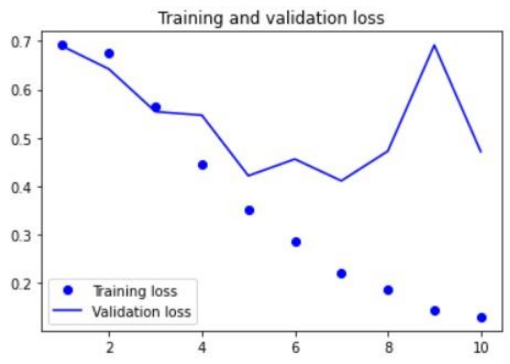


```
from keras.models import Sequential
from keras.layers import Embedding
from keras.layers import LSTM
from keras.layers import Flatten, Dense

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 added LSTM in the design, and we can observe the accuracy of the model below,

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





Conclusion:

When using 100 training samples, both the custom model and the GloVe embedding model exhibited superior accuracy compared to using 200 samples. This suggests that increasing the training sample size leads to overfitting. Notably, the GloVe embedding model outperformed the custom model, likely due to its pre-trained nature and high-quality training data.

Additionally, I explored LSTM (long short-term memory networks) towards the end, which yielded the best results among all models. Several factors could contribute to this, such as variations in the training sample and the unique functionality of the LSTM model compared to others.s