1. **Describe in your own words (one-two paragraphs, half a page max) what circumstances call for an RNN (what advantages does it offer or what problem does it address over a normal DNN?). [3 points]**

The simple ANN suffers from the following problems:

- Fixed length issue. This means that we need to predefine the input size. One workaround is to have a fixed input size of the max length of the sentences in the document while training. However, the problem with such an approach is inclusion of unnecessary zeros (in padding). Let's say the average length of the sentence in document is 30 but the max length is 100. This leads to increased computation costs as we are going to have to train more parameters. Also, what if the input size is 200 during testing?

- Secondly, the word embeddings are sent into an ANN at one go. This leads to loss of sequential information. In NLP tasks like sentiment classification, machine translation or voice-to-text, it is important to retain the sequential order of words for better accuracy.

To resolve the above issues, we use RNN architecture which, in addition to ANN capabilities, also consists of a memory state (hidden state). RNN is inherently designed to capture sequential nature. For each input, the neuron takes a weighted sum of the previous output, the current input word and the bias and then sends it to the activation function. This helps retain the sequential order of the sentences, as words are processed one by one (in timesteps). Important to note that 'recurrent' means that the same layer is used again and again, with the same shared weights (no back propagation) for a particular sentence.

2. **What is padding in the context of RNN and why is it needed? [2 points]**

In contrast to ANNs which handle well-defined tabular data with fixed feature vectors, the RNNs (especially in case of language processing) can have varying lengths of inputs. This means we need to fix the input feature length; so often we use padding. Padding adds zeros/padding tokens at the start or end of the sequences which are shorter than the desired length (maximum length of sentence in a document).

```
array([[ 6,  4,  0,  0,  0,  0,  0],
       [ 3,  7,  8,  9,  3, 10,  0],
       [11,  2, 12,  3,  4,  0,  0],
       [13, 14, 15, 16, 17,  0,  0],
       [18,  5, 19, 20, 21, 22, 23],
       [24,  2, 25,  0,  0,  0,  0],
       [26, 27,  2, 28,  0,  0,  0],
       [29, 30, 31,  2, 32, 33, 34],
       [35, 36, 37, 38,  0,  0,  0],
       [39, 40,  5, 41, 42,  0,  0]],
```

In this image, I have used zero padding to increase the size of the sentences which are shorter than the maximum length of the sentence in the entire document. So, we use padding to turn our varying input lengths of data points into a fixed input size. However, this approach often leads to sparse vector problem.

3. **Apply RNN on IMDB dataset as we did for last assignment w.r.t. autoencoder. Report your loss after 100 epochs. [5 points]**

I implemented a SimpleRNN-based binary classification model using Keras on the IMDB sentiment dataset. The model consists of a SimpleRNN layer with 32 units followed by a Dense output layer. After training for 100 epochs:

Final training loss was 0.6898

Final validation loss was 0.6936

The accuracy remained close to 50 percent throughout training and validation, indicating that the model was not able to learn effective sentiment representations from the data. This outcome suggests that a more complex RNN architecture like LSTM or GRU may be more suitable for this task.

Model: "sequential_3"

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, 32)	1,088
dense_1 (Dense)	(None, 1)	33

Total params: 1,121 (4.38 KB)
Trainable params: 1,121 (4.38 KB)
Non-trainable params: 0 (0.00 B)

Loss: Binarycrossentropy as it's a 1/0 sentiment classification task.
Optimizer: Adam

```
Epoch 100/100
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.5139 - loss: 0.6898 - val_accuracy: 0.5072 - val_loss: 0.6936
```

Accuracy hovers around 50 percent, which is equivalent to random guessing. This suggests the model did not learn meaningful patterns from the data. Possible reasons:

1. We chose a small sentence length of 20
2. We used integer encodings. Perhaps, word embeddings can yield a better result as those are dense, capturing semantic information which is crucial for sentiment classification tasks.