**Assignment - 4**

1. Can you think of a few applications for a sequence-to-sequence RNN? What about a sequence-to-vector RNN? And a vector-to-sequence RNN?

Ans: Sequence-to-sequence RNNs: Machine translation, text summarization, speech recognition, conversational agents.

* + Sequence-to-vector RNNs: Sentiment analysis, document classification, image captioning.
  + Vector-to-sequence RNNs: Language generation, music generation, video description generation.

1. Why do people use encoder–decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?

Ans: Encoder-decoder RNNs are preferred over plain sequence-to-sequence RNNs for automatic translation because they allow for variable-length input and output sequences. The encoder processes the input sequence and produces a fixed-size context vector that captures the semantic meaning of the input. This context vector is then used by the decoder to generate the output sequence, allowing the model to handle translations of varying lengths.

1. How could you combine a convolutional neural network with an RNN to classify videos?

Ans: To classify videos using a combination of convolutional neural networks (CNNs) and recurrent neural networks (RNNs), you can use a CNN to extract features from individual frames of the video and then feed these features into an RNN to model temporal dependencies between frames. The CNN acts as a feature extractor, while the RNN processes the sequence of feature vectors to make predictions about the video's content.

1. What are the advantages of building an RNN using dynamic\_rnn() rather than static\_rnn()?

Ans: The advantages of building an RNN using dynamic\_rnn() rather than static\_rnn() include better memory management and computational efficiency. dynamic\_rnn() dynamically unrolls the RNN graph during execution, allowing it to handle variable-length sequences more efficiently. This can lead to faster training and inference times, especially when dealing with large datasets or sequences of varying lengths.

1. How can you deal with variable-length input sequences? What about variable-length output sequences?

Ans: Variable-length input sequences can be handled by padding shorter sequences to match the length of the longest sequence in the dataset. Variable-length output sequences can be handled similarly by using special tokens to indicate the end of a sequence or by masking irrelevant parts of the output during training. Additionally, techniques such as teacher forcing and beam search can be used during inference to handle variable-length output sequences more effectively.

1. What is a common way to distribute training and execution of a deep RNN across multiple GPUs?

Ans: A common way to distribute training and execution of a deep RNN across multiple GPUs is to use data parallelism. In this approach, the model and the training data are split across multiple GPUs, with each GPU processing a subset of the data and updating a shared set of model parameters. This allows for parallel computation of gradients and speeds up the training process. frameworks like TensorFlow and PyTorch provide built-in support for distributed training, making it easier to scale RNNs across multiple GPUs.