Types of RNN
One to many ---- Image to Description of image

Many to one ---- Sentiment Analysis

→Sequence to Sequence models
Many to many (Synchronous) ------- Input length and output length must be same
→ POS tagging (part of speech) , NER (Name Entity Recognition)

Many to many (Asynchronous)------ Input length and output length are different
→ Machine Translation

Seq2Seq tasks
→text summarization
→Question answer
→Chatbot
→speech to text

Language modeling --→ train NLP/DL for predicting next word
Example---- I live in india and capital of india is ______ -→Delhi

Context vector →cv is set of numbers representation of English sentence
Encoder →context vector (Botleneck layer)→ Decoder
Encoder is model consisting of multiple LSTMs, it may be BiLSTM,Stacked LSTM .
Summarizes input into context vector

An attention mechanism is a key concept in the field of machine learning, particularly in the context of sequence-to-sequence (Seq2Seq) models with encoder-decoder architecture. Instead of processing an entire input sequence all at once, attention mechanisms allow the model to focus on specific parts of the input sequence while generating the output sequence. This mimics the human ability to selectively attend to different elements when processing information.

Bahdanau Attention and Luong Attention are two mechanisms used in the context of sequence-to-sequence models, especially in machine translation tasks. These attention mechanisms allow the model to focus on different parts of the input sequence when generating each element of the output sequence. While Bahdanau Attention introduces alignment scores, Luong Attention simplifies this process by using a global score. Understanding the differences between these attention mechanisms is crucial for optimizing performance in sequence-based tasks.

Scaling Self Attention in Scaled Dot Product Attention is crucial for stabilizing training, optimizing dataset utilization, and improving the model's ability to focus on relevant information within sequences by standardizing the variance of dot products

How Self-Attention Works
For a given input sequence (e.g., a sentence), self-attention follows these steps:
Convert Words into Embeddings
Each word in the input is converted into a vector representation (embedding).
Generate Query, Key, and Value Matrices
Each word embedding is transformed into three vectors:
Query (Q): Represents the current word (What am I looking for?)
Key (K): Represents other words (What do I have?)
Value (V): Contains the actual word information

How self attention works
Suppose word is “money bank”
Word converted into embedded word
3 metrics created Wv,Wk,Wq
6 vectors are generated by dot product of emoney and ebank
Qmoney and kmoney are multiplied to generate similarity scores S11,S12

Similarity scores are scaled by dividing it by sqrt of dimensions of k.

Softmax is applied to generate probabilities.

Word is predicted.

Multi-head Attention enhances the expressiveness and representational capacity of Transformers by allowing the model to attend to different parts of the input data simultaneously. By utilizing multiple attention heads, the model can capture diverse patterns and relationships in the data, enabling more effective information processing and feature extraction. This mechanism enhances the model's ability to handle complex sequences and tasks in natural language processing and other domains.

Positional Encoding in Transformers
Transformers, unlike RNNs, do not have a built-in notion of word order since they process all tokens simultaneously. To enable the model to understand the relative positions of words in a sequence, Positional Encoding is introduced.
Why is Positional Encoding Needed?
Self-attention in Transformers treats input tokens as a set rather than a sequence, meaning it lacks a sense of order. However, in language and sequential data, word order is crucial for meaning. Positional encoding helps inject order information into input embeddings so that the model can distinguish between different positions in a sentence.

Layer Normalization in Transformers

Layer Normalization (Layer Norm) is a crucial technique used in Transformers to stabilize training and improve convergence. Unlike Batch Normalization (Batch Norm), which normalizes across batches, Layer Norm operates independently on each input sample, making it better suited for NLP tasks and sequential data.

Why is Layer Normalization Needed?

Reduces Internal Covariate Shift: Helps keep activations in a stable range, improving training efficiency.

Ensures Stable Gradients: Prevents exploding or vanishing gradients, especially in deep models.

Independent of Batch Size: Unlike Batch Norm, it works well even for a single input sequence.

How Does Layer Normalization Work?

Given an input vector **X** of shape (d_{model}) (i.e., feature dimension), Layer Norm applies the following transformation:

$$\hat{x}_i = \frac{x_i - \mu}{\sigma}$$

$$y_i = \gamma \hat{x}_i + \beta$$

where:

- μ is the **mean** of all elements in the feature dimension.
- σ is the **standard deviation** of all elements in the feature dimension.
- γ and β are **learnable parameters** for scaling and shifting.
- $\hat{x}_i$ is the normalized value, and y_i is the final output.

Masked Multi-head Attention is used in transformer models to ensure that each token in a sequence only attends to previous tokens and itself, not future tokens. This masking is essential for autoregressive tasks like language generation, enabling the model to generate sequences one step at a time without peeking ahead. By employing multiple attention heads, the model captures diverse contextual information while maintaining the sequential integrity.

Cross Attention is a mechanism in transformer models where the attention is applied between different sequences, typically between the output of one layer and the input of another. It allows the model to focus on relevant parts of the input sequence while generating output, making it crucial in tasks like translation, where the decoder attends to the encoder's output to generate accurate translations.

The Decoder in a transformer architecture generates output sequences by attending to both the previous tokens (via masked self-attention) and the encoder's output (via cross-attention). Each decoder layer consists of multi-head self-attention, cross-attention, and feed-forward layers. This structure allows the model to generate coherent sequences by considering both past outputs and relevant input context, making it effective for tasks like text generation and translation.

Inference in transformers involves generating predictions from the trained model. During inference, the decoder predicts one token at a time, using previously generated tokens and attending to the encoder's output. The process continues iteratively until the end of the sequence is reached, making it suitable for tasks like language translation and text generation.