[Recurrent Neural Network (RNN) architecture explained in detail (towardsmachinelearning.org)]

RNN Architecture

What is RNN ?

Recurrent Neural Networks or RNNs, are very important variant of Neural Networks heavily used in NLP.

They're a class of neural networks that allow previous outputs to be used as inputs while having hidden states.

RNN has concept of Memory which Remembers all information about what has been calculated till time step t.

RNNs are called recurrent because they perform same task for every element of a sequence,

With the output depended on previous computations.

Recurrent Neural Networks (RNNs) are a type of neural network architecture designed for sequential data processing. Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a memory of previous inputs in their internal state. This makes them well-suited for tasks involving sequences, such as time series prediction, natural language processing, and speech recognition.

The intuition behind RNNs lies in their ability to capture dependencies and patterns in sequential data by considering the context of past inputs. This is achieved through the recurrent connections, which enable information to persist across different time steps.

The strength of RNNs lies in their ability to model dependencies over time. However, traditional RNNs have limitations, such as difficulty in learning long-term dependencies (vanishing or exploding gradient problems). More advanced variations, like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), have been developed to address these issues and improve the learning of long-range dependencies.

In summary, the intuition behind RNNs is to use recurrent connections to capture sequential dependencies and patterns, making them powerful for tasks involving time-series or sequential data.

Why RNNs?

In a general neural network, an input is fed to an input layer and is further processed through number of hidden layers and a final output is produced, with an assumption that two successive inputs are independent of each other or input at time step t has no relation with input at timestep t-1.

However this assumption is not true in a number of real-life scenarios. For instance, if one wants to predict the price of a stock at a given time or wants to predict the next word in a sequence then it is imperative that dependence on previous observations is considered.

To understand the need of RNNs or how RNNs can be helpful , let's understand it with one real time incident that happened recently.

You must have come across a recent incident where **Pakistan batsman Umar Akmal** has been trolled after he posted a photo on Twitter with an obvious error. And the caption of the post was 'M**other from another Brother** '.

And following this incident there has been many such sentences surfaced over internet like below-

- If being crime is arrest then sexy me
- You don't have to be well to travel rich
- If I'm bad , you're my dad
- Policy is the best honesty
- Health is injurious to smoking
- If opportunity doesn't door then build a knock
- Don't happy be worry
- Cure is best prevention
- Everything is war in fair and love
- A day a doctor, keeps an apple away
- I love to cry in walking so nobody see I'm raining
- Blood is in my cricket
- Consumption of health is injurious to liquor
- Always a ravian , once a ravian
- Work hard in success , let your silence make noise

So you see a little jumble in the words made the sentence incoherent . There are multiple such tasks in everyday life which get completely disrupted when their sequence is disturbed.

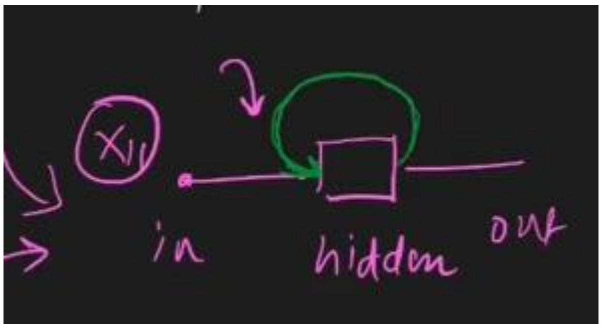
For instance, sentences that we just saw above- the sequence of words define their meaning, a time series data – where time defines the occurrence of events, the data of a genome sequence- where every sequence has a different meaning. There are multiple such cases wherein the sequence of information determines the event itself.

So if we’re trying to use such data to predict any reasonable output, we need a network ,which has access to some prior knowledge about the data to completely understand it. That’s where Recurrent neural networks come to rescue.

To understand what is memory in RNNs , what is recurrence unit in RNN, how do they store information of previous sequence , let’s first understand the architecture of RNNs.

How RNN Works?

RNN mai 1 State hota hai



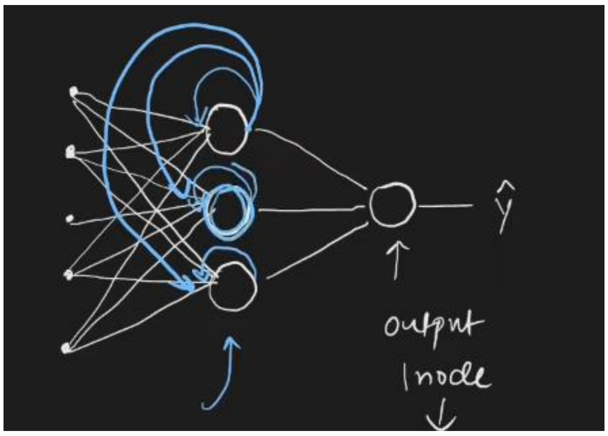
RNN mai concept hai State ka

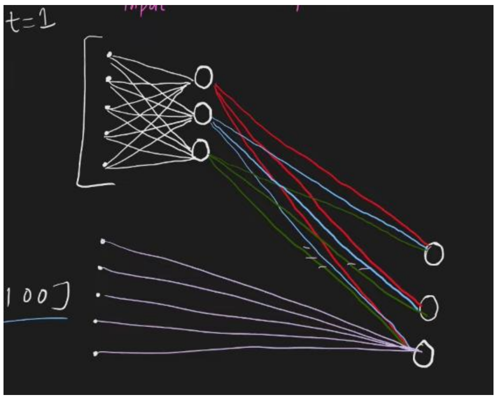
Jo Connection hai vo RNN ko RNN banata hai

RNN mai jo hidden layer hai vo vapas palat ke Feedback deta hai

Yeah connection hie RNN ka sbhse core funda hai

Har Node ka jo Output hoga vo Ushi Node keliye Input ka kaam karega


$\hat{y}$
output
1 node

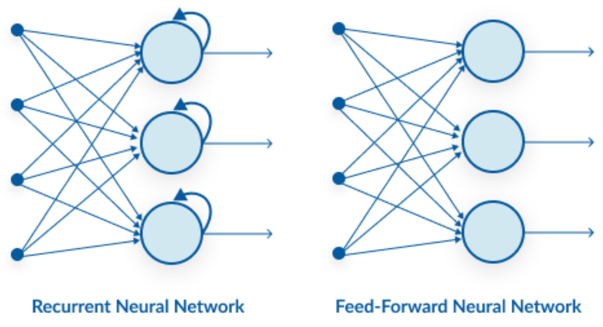

t=1
input
100]

RNN vs Feed Forward Neural Network

A feed-forward neural network has only 1 route of Information Flow:

From input layer to Output Layer, passing through all hidden layers.

The data flows across network in straight route, never going through same Neuron/Node twice



Feed-forward neural networks are poor predictions of what will happen next because they have no memory of the information they receive. Because it simply analyses the current input, a feed-forward network has no idea of temporal order. Apart from its training, it has no memory of what transpired in the past.

The information is in an RNN cycle via a loop. Before making a judgment, it evaluates the current input as well as what it has learned from past inputs. A recurrent neural network, on the other hand, may recall due to internal memory. It produces output, copies it, and then returns it to the network.

Architecture of RNN

Recurrent Neural Networks (RNNs) keep memory through their hidden states. The hidden state of an RNN is like the neural network's memory, capturing information from the current input and retaining knowledge from previous time steps. Let's break down how RNNs maintain this memory:

1. **Initialization:**
   - At the beginning (time step 0), the RNN is initialized with an initial hidden state (h_0). This initial hidden state can be set to zeros or learned during training.
2. **Updating Hidden State:**
   - At each time step (t), the RNN takes the current input (x_t) and the previous hidden state (h_{t-1}).
   - It combines these two pieces of information, often using a weighted sum, and applies an activation function to produce the new hidden state (h_t).
3. **Memory Effect:**
   - The key idea is that the new hidden state (h_t) now contains information not only from the current input (x_t) but also from the previous hidden state (h_{t-1}). This creates a form of memory, allowing the RNN to capture and remember relevant information over time.
4. **Sequential Accumulation:**
   - As the RNN processes input sequences, the hidden state at each time step accumulates information from all the preceding time steps. This sequential accumulation of information allows the RNN to maintain context and capture dependencies over the entire sequence.
5. **Long-Term Dependencies (Challenges):**
   - While RNNs are capable of maintaining short-term dependencies, they often face challenges in learning long-term dependencies due to issues like vanishing or exploding gradients. This has led to the development of more advanced architectures like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) that are designed to better capture long-term dependencies.

6.

Unfolding RNN

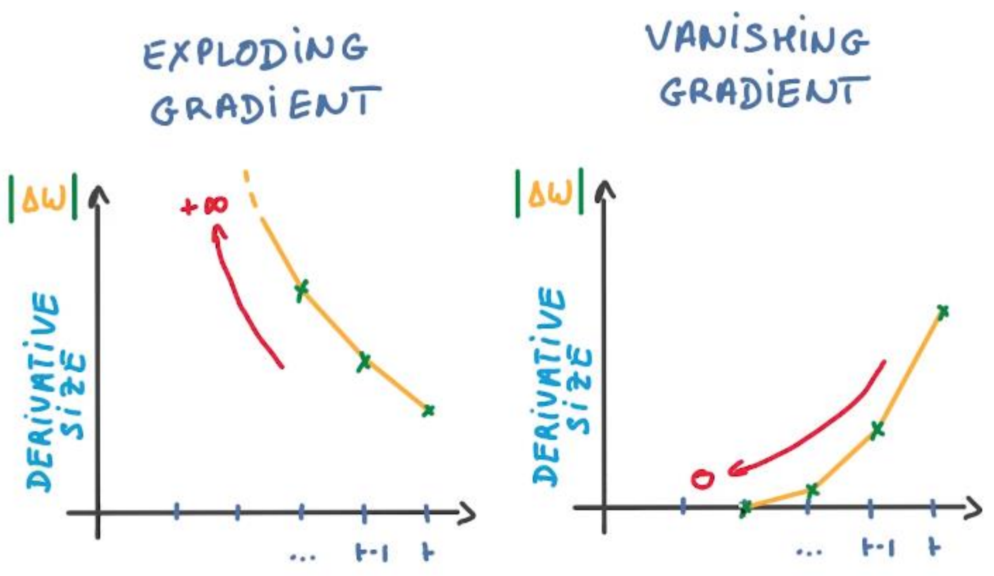
A RNN can be thought of as Multiple Copies of a feedforward network, each passing a message to a successor.

Why Recurrent Neural Network

Jo Hidden layer hai vo Reoccur horaha hai , that’s why we say it Recurrent Neural Network.

2 Issues of Standard RNN

There are two key challenges that RNNs have had to overcome, but in order to comprehend them, one must first grasp what a gradient is.



With regard to its inputs, a gradient is a partial derivative. If you're not sure what that implies, consider this: a gradient quantifies how much the output of a function varies when the inputs are changed slightly.

A function's slope is also known as its gradient. The steeper the slope, the faster a model can learn, the higher the gradient. The model, on the other hand, will stop learning if the slope is zero. A gradient is used to measure the change in all weights in relation to the change in error.

Exploding Gradients: Exploding gradients occur when the algorithm gives the weights an absurdly high priority for no apparent reason. Fortunately, truncating or squashing the gradients is a simple solution to this problem.

Vanishing Gradients: Vanishing gradients occur when the gradient values are too small, causing the model to stop learning or take far too long. This was a big issue in the 1990s, and it was far more difficult to address than the exploding gradients. Fortunately, Sepp Hochreiter and Juergen Schmidhuber’s LSTM concept solved the problem.