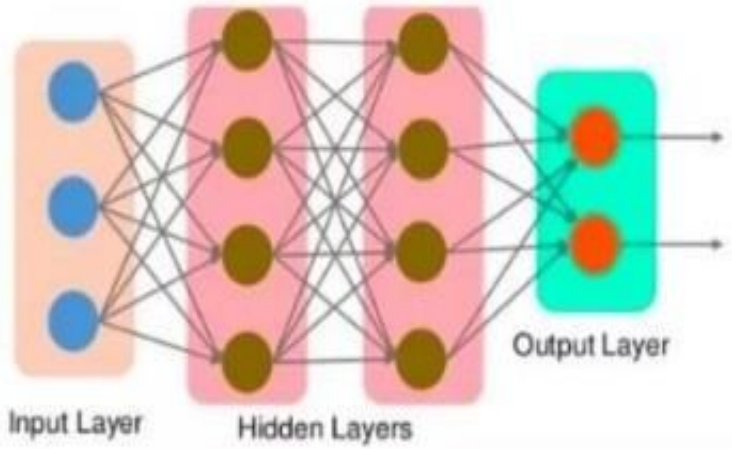
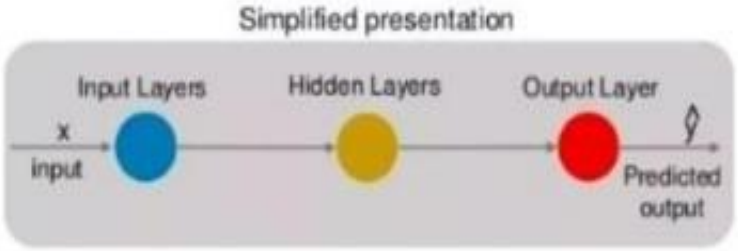
Recurrent Neural Networks

Feed Forward

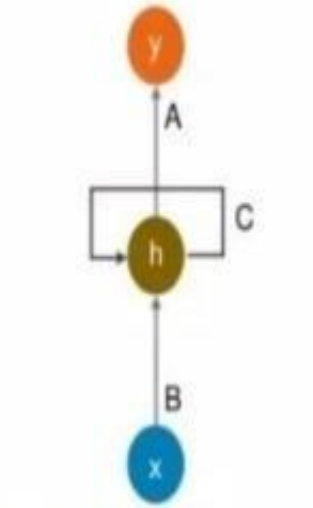
- In feed forward neural network, information always flows in one direction[forward], from the input nodes to the hidden layers[if any], and then to output nodes. There are no cycles or loops in the network.
- Decisions are based on current input
- No memory about the past
- No future scope

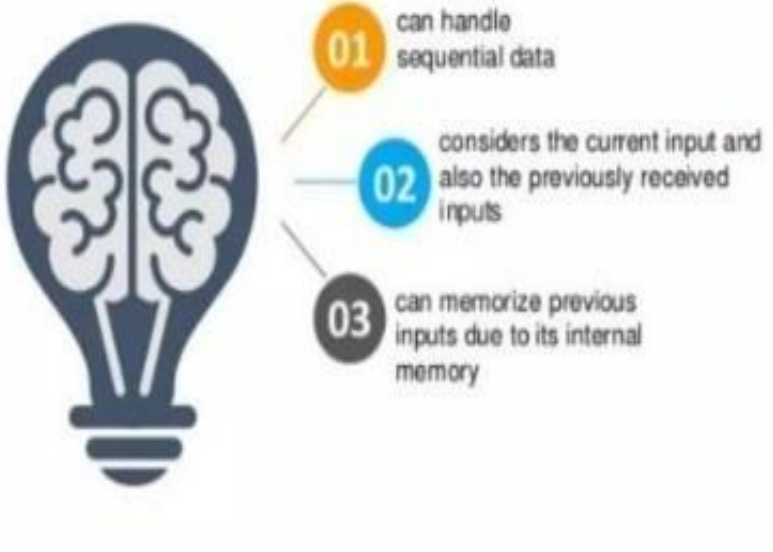




RNN

- Recurrent Neural Network works on the principle of saving the output layer and feeding this back to the input in order to predict the output of the layer.

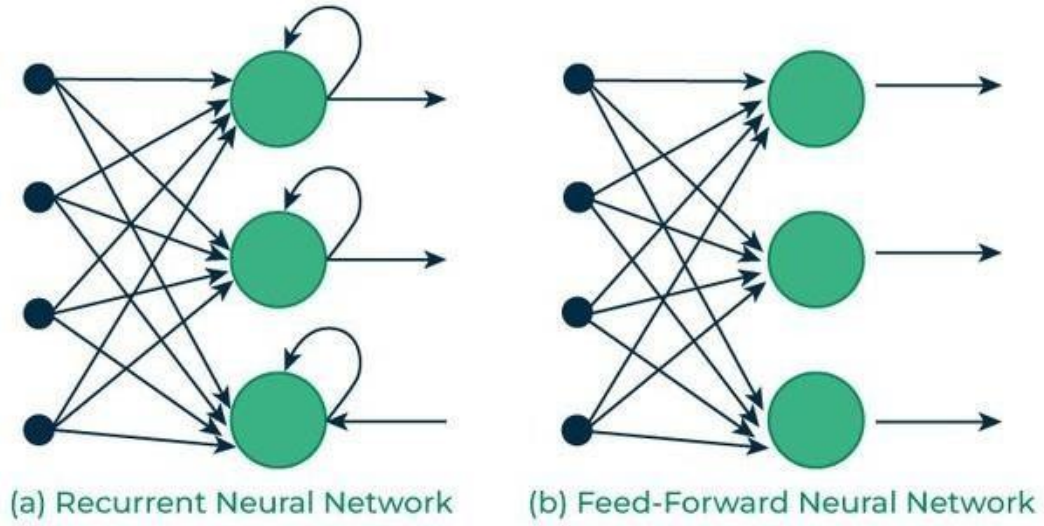




- Recurrent Neural Network is basically a generalization of feed-forward neural network that has an internal memory. RNNs are a special kind of neural networks that are designed to effectively deal with sequential data. This kind of data includes time series (a list of values of some parameters over a certain period of time) text documents, which can be seen as a sequence of words, or audio, which can be seen as a sequence of sound frequencies over time.
- RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. For making a decision, it considers the current input and the output that it has learned from the previous input.
- Cells that are a function of inputs from previous time steps are also known as memory cells.
- Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.

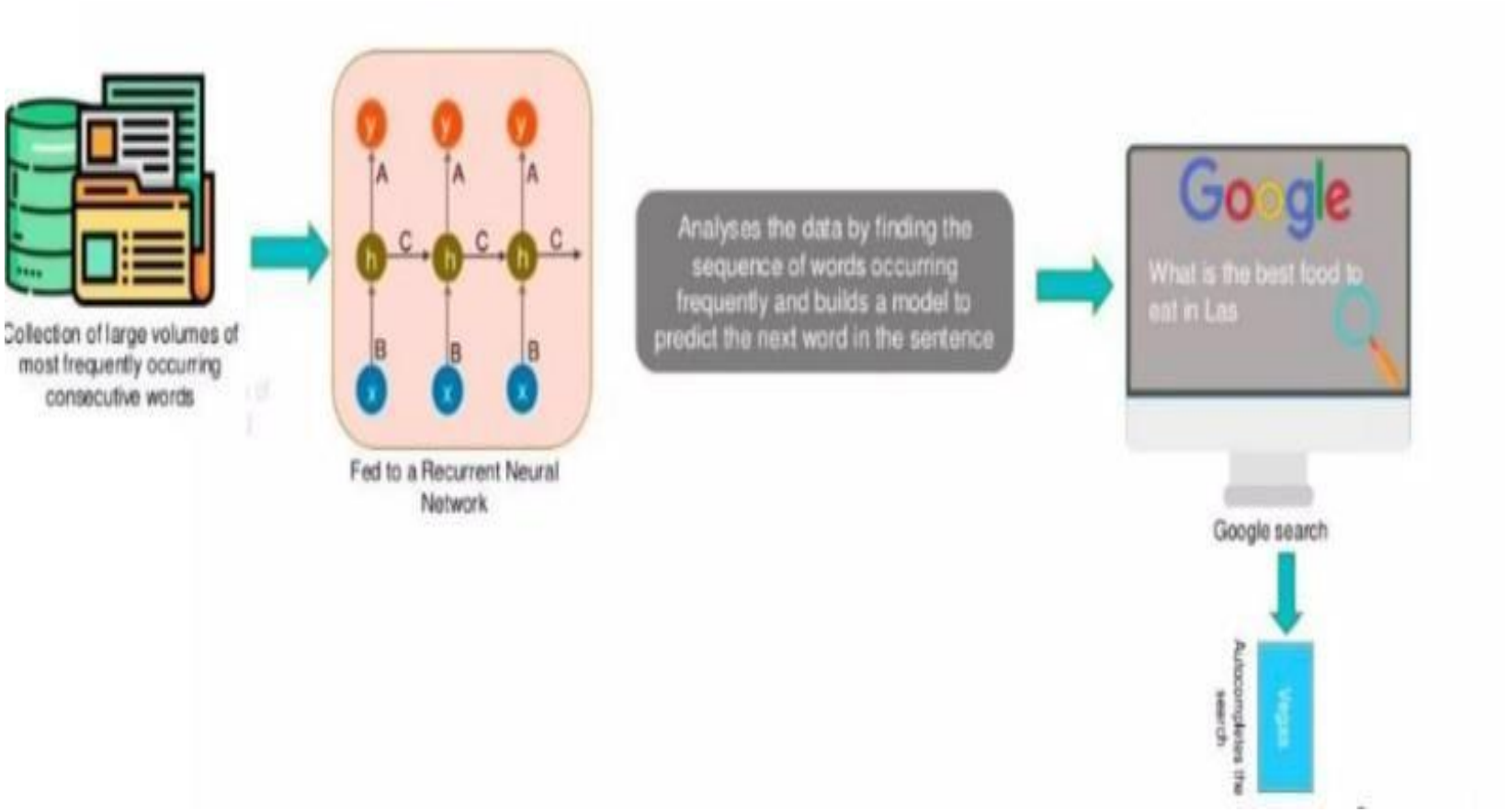
What is RNN?

Recurrent Neural Networks (RNN) is a neural network in which the output of the current state (or layer) is fed back to the previous or current state as inputs.



(a) Recurrent Neural Network (b) Feed-Forward Neural Network

Example


Collection of large volumes of most frequently occurring consecutive words
Fed to a Recurrent Neural Network
Analyses the data by finding the sequence of words occurring frequently and builds a model to predict the next word in the sentence
Google
What is the best food to eat in Las
Google search

Why RNN

- The basic challenge of classic feed-forward neural network is that it has no memory, that is, each training example given as input to the model is treated independent of each other. In order to work with sequential data with such models we need to show them the entire sequence in one go as one training example. This is problematic because number of words in a sentence could vary and more importantly this is not how we tend to process a sentence in our head.
- When we read a sentence, we read it word by word, keep the prior words / context in memory and then update our understanding based on the new words which we incrementally read to understand the whole sentence. This is the basic idea behind the RNNs — they iterate through the elements of input sequence while maintaining a internal "state", which encodes everything which it has seen so far. The "state" of the RNN is reset when processing two different and independent sequences.
- Recurrent neural networks are a special type of neural network where the outputs from previous time steps are fed as input to the current time step.
- Another distinguishing characteristic of recurrent networks is that they share parameters across each layer of the network. While feed forward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network. That said, these weights are still adjusted in the through the processes of backpropagation and gradient descent to facilitate reinforcement learning.

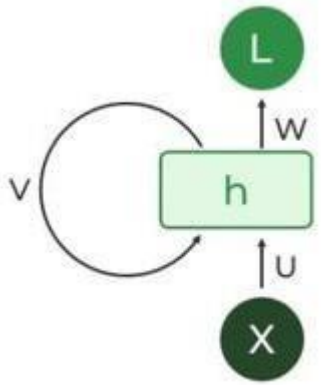
Components of RNN

There are two main components of RNN:

- Recurrent Neurons
- RNN Unfolding

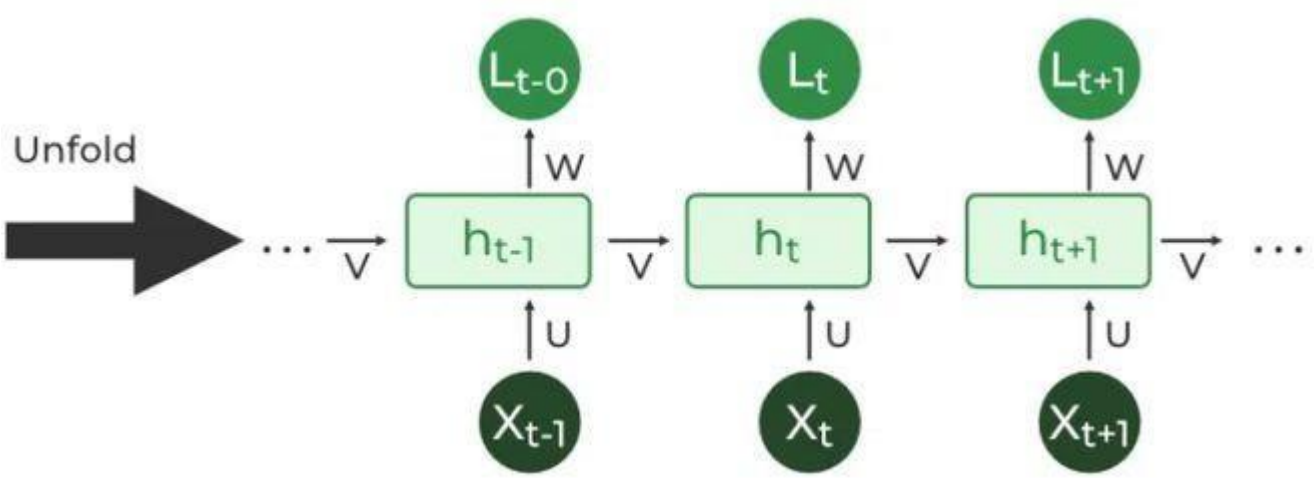
Recurrent Neurons

Recurrent Neurons are Neuron Structures which use their output of the current state as an input of the next state.



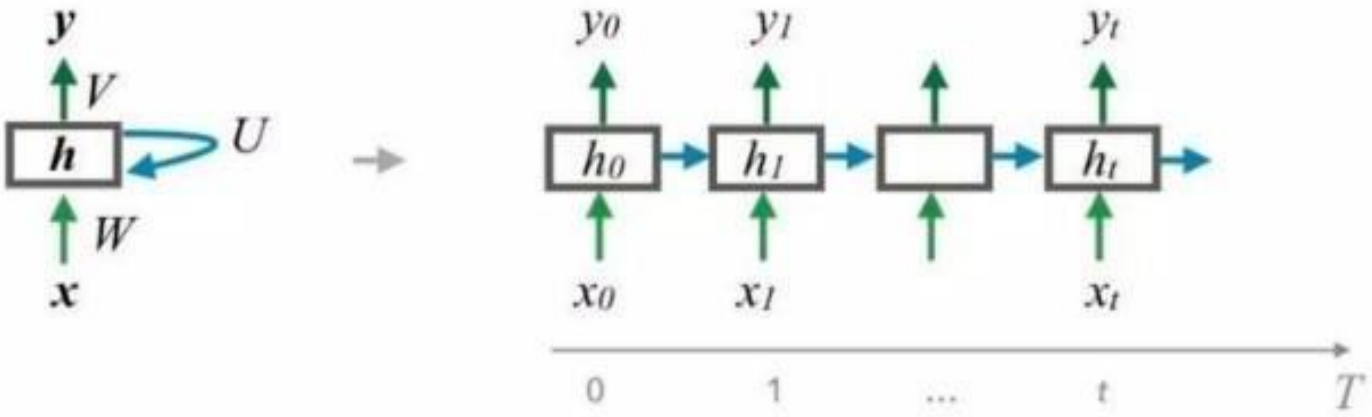
RNN Unfolding

Unfolding (or Unrolling) is a process in which a recurrent neural layer can be expanded into multiple neural layers each representing each state of time.



Intuition

- Recurrent Neural Network (RNN) is a neural network model proposed in the 80's for modelling time series.
- The structure of the network is similar to feedforward neural network, with the distinction that it allows a recurrent hidden state whose activation at each time is dependent on that of the previous time (cycle).

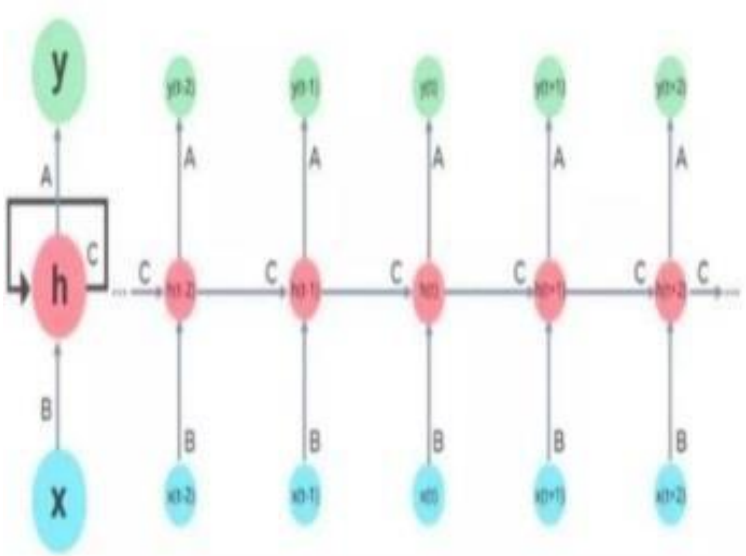


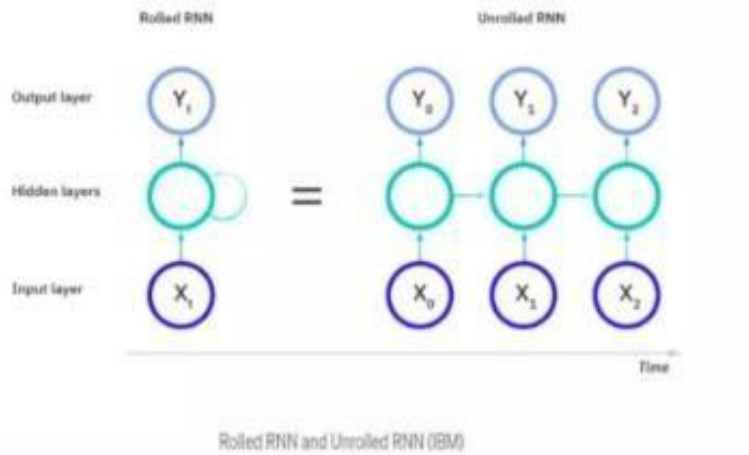
$$y_{(t)} = \phi(x_{(t)}^{T} \cdot w_x + y_{(t-1)}^{T} \cdot w_y + b)$$

ϕ() is the activation function like ReLU

bias

How does RNN Looks Like





Rolled RNN and Unrolled RNN (IBM)

Why Unfolding?

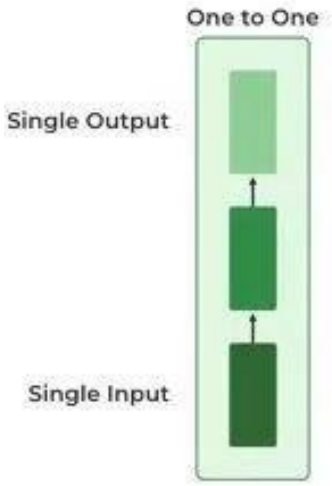
Unfolding enables **Backpropagation through Time (BPTT)** which enhances the neural layer to learn its dependencies with the previous state layers which affects learning sequential data positively.

Types of RNNs

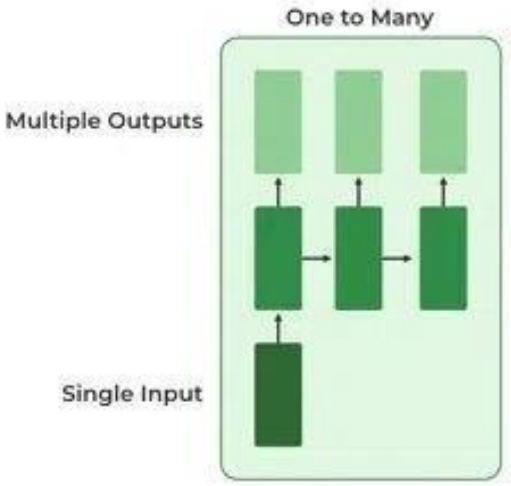
- **One to One model:** One Input given, one output got.
- **One to Many model:** One input given multiple output got.
- **Many to One model:** Multiple inputs given, one input got.
- **Many to Many model:** Multiple inputs given, multiple outputs got.

Types of Neurons

One to One:

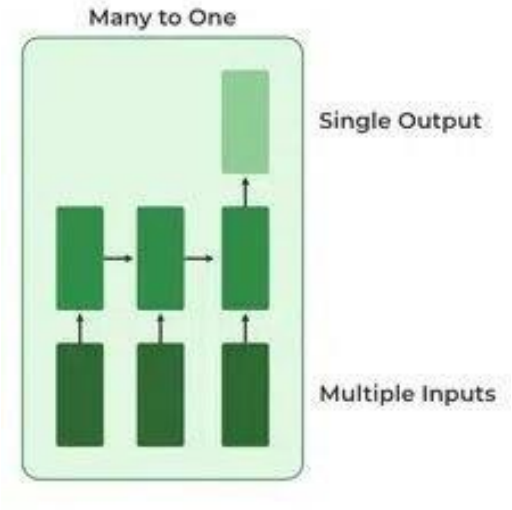


One to Many:

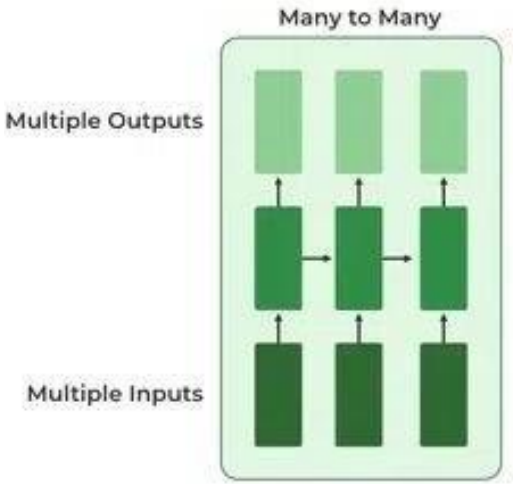


Types of Neurons

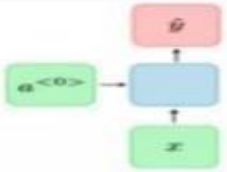
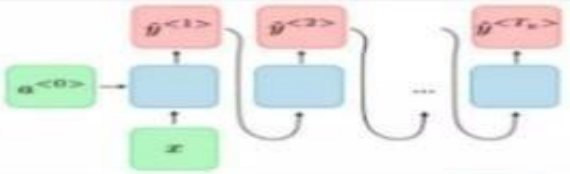
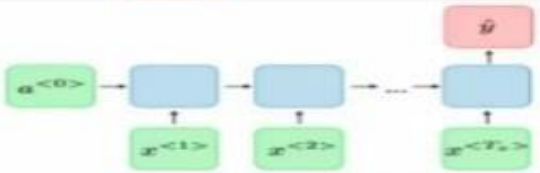
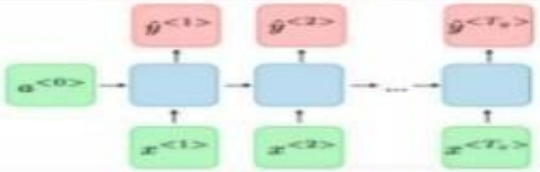
Many to One:



Many to Many:



Example

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition

Examples of RNN models

- **Vanilla RNN**
- **Bidirectional RNN**
- **Long Short-Term Memory (RNN)**
- **Gated Recurrent Units (GRU)**

Advantages of RNN

- **Sequential Memory:** RNNs retain information from previous inputs, making them ideal for time-series predictions where past data is crucial. This capability is often called **Long Short-Term Memory (LSTM)**.
- **Enhanced Pixel Neighborhoods:** RNNs can be combined with convolutional layers to capture extended pixel neighborhoods, improving performance in image and video data processing.

Limitations of RNN

The gradient of RNN often either **diminishes or increases uncontrollably**. These phenomenons are called **Vanishing Gradient or Exploding Gradient** respectively.

Limitations

- The problem with RNNs is that as time passes by and they get fed more and more new data, **they start to "*forget*"** about the previous data they have seen *(vanishing gradient problem)*, as it gets **diluted** between the new data, the transformation from activation function, and the weight multiplication. This means they have a **good *short term memory***, but a slight problem when trying to remember things that have happened a while ago (data they have seen many time steps in the past).
- The more time steps we have, the more chance we have of back-propagation gradients either accumulating and exploding or vanishing down to nothing.
- Consider the following representation of a recurrent neural network:

$$h_t = f(h_{t-1}, x_t)$$

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1})$$

- Here, **ht** is the new state *(current time stamp)*, **ht-1** is the previous state *(previous time stamp)* while **x□** is the current input.
- Where ***U*** and ***V*** are the weight matrices connecting the inputs and the recurrent outputs respectively. We then often will perform a softmax of all **h□** the outputs.

Vanishing Gradient

Notice, however, that if we go back three time steps in our recurrent neural network, we have the following:

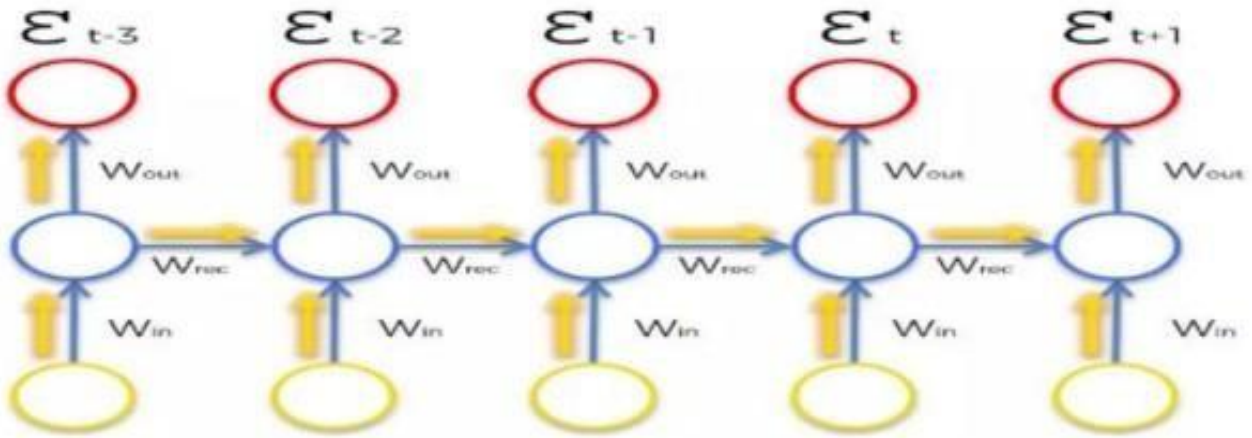
$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{V}(\sigma(\mathbf{U}\mathbf{x}_{t-1} + \mathbf{V}(\sigma(\mathbf{U}\mathbf{x}_{t-2}))))...$$

From the above you can see, as we work our way back in time, we are essentially adding deeper and deeper layers to our network. This causes a problem — consider the gradient of the error with respect to the weight matrix ***U*** during back-propagation through time, it looks something like this:

$$\frac{\partial E_3}{\partial U} = \frac{\partial E_3}{\partial out_3}\frac{\partial out_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial U}\cdots$$

- The equation above is only a rough approximation of what is going on during back-propagation through time. Each of these gradients will involve calculating the gradient of the sigmoid function. The problem with the sigmoid function occurs when the input values are such that the output is close to either 0 or 1 — at this point, the gradient is very small (saturating).
- For ex:- Lets say the value decreased like 0.863 →0.532 →0.356 →0.192 →0.117 →0.086 →0.023 →0.019..
- you can see that there is no much change in last 3 iterations.
- It means that when you multiply many sigmoid gradients together you are multiplying many values which are potentially much less than zero — this leads to a vanishing gradient problem.

Exploding Gradient



The mathematics that computes this change is multiplicative, which means that the gradient calculated in a step that is deep in the neural network will be multiplied back through the weights earlier in the network. Said differently, the gradient calculated deep in the network is "diluted" as it moves back through the net, which can cause the gradient to vanish - giving the name to the vanishing gradient problem!

The actual factor that is multiplied through a recurrent neural network in the backpropagation algorithm is referred to by the mathematical variable Wrec. It poses two problems:

- When Wrec is small, we experience a vanishing gradient problem
- When Wrec is large, we experience an exploding gradient problem

Note that both of these problems are generally referred to by the simpler name of the "vanishing gradient problem".

Solving Vanishing Gradient

- Weight initialization is one technique that can be used to solve the vanishing gradient problem. It involves artificially creating an initial value for weights in a neural network to prevent the backpropagation algorithm from assigning weights that are unrealistically small.
- You could also use echo state networks, which is a specific type of neural network designed to avoid the vanishing gradient problem
- The most important solution to the vanishing gradient problem is a specific type of neural network called Long Short-Term Memory Networks (LSTMs), which were pioneered by Sepp Hochreiter and Jürgen Schmidhuber. Recall that Mr. Hochreiter was the scientist who originally discovered the vanishing gradient problem.
- LSTMs are used in problems primarily related to speech recognition, with one of the most notable examples being Google using an LSTM for speech recognition in 2015 and experiencing a 49% decrease in transcription errors.
- LSTMs are considered to be the go-to neural net for scientists interested in implementing recurrent neural networks.

Solving the Exploding Gradient Problem

- For exploding gradients, it is possible to use a modified version of the backpropagation algorithm called truncated backpropagation. The truncated backpropagation algorithm limits that number of timesteps that the backproporation will be performed on, stopping the algorithm before the exploding gradient problem occurs.
- You can also introduce penalties, which are hard-coded techniques for reduces a backpropagation's impact as it moves through shallower layers in a neural network.
- Lastly, you could introduce gradient clipping, which introduces an artificial ceiling that limits how large the gradient can become in a backpropagation algorithm.

Uses of RNNs

The main use of RNNs is to perform training in Sequential data.

For Eg,

- **Time Series Prediction:** RNNs can be used to predict Time Series Predictions like Forecasting, Stock Market Analysis etc.
- **Natural Language Processing (NLP):** Sentiment Analysis, Language Modelling etc., are done using RNNs. Earlier Chatbots (LLMs) and Language Translators (LTMs) use RNN.
- **Speech Recognition:** RNNs capture temporal patterns in speech data, aiding in speech-to-text and other audio-related applications.