**RNN (Recurrent Neural Network)**

Recurrent Neural Networks (RNN) are a type of Neural Network where the output from the previous step is fed as input to the current step. RNN's are mainly used for, Sequence Classification — Sentiment Classification & Video Classification. Sequence Labelling — Part of speech tagging & Named entity recognition.

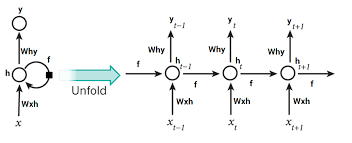


Fig: Fully connected Recurrent Neural Network

Here, “x” is the input layer, “h” is the hidden layer, and “y” is the output layer. At any given time, t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output

**Why Recurrent Neural Networks?**

RNN were created because there were a few issues in the feed-forward neural network:

* Cannot handle sequential data
* Considers only the current input
* Cannot memorize previous inputs

The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

**How Does Recurrent Neural Networks Work?**

In Recurrent Neural networks, the information cycles through a loop to the middle-hidden layer.

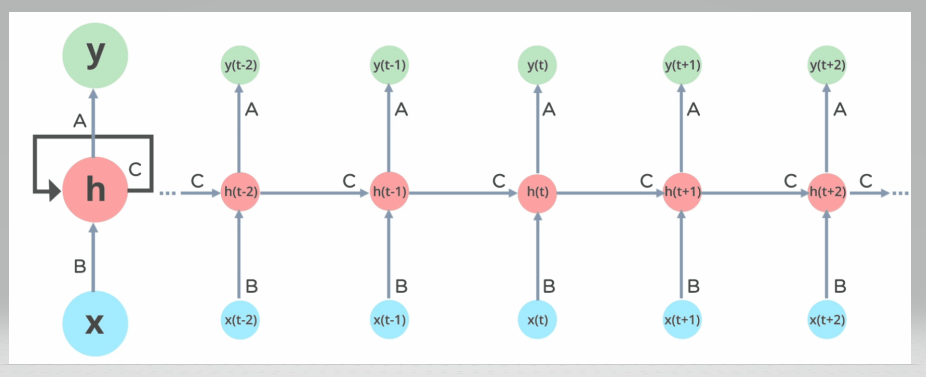


Fig: Working of Recurrent Neural Network

The input layer ‘x’ takes in the input to the neural network and processes it and passes it onto the middle layer.

The middle layer ‘h’ can consist of multiple hidden layers, each with its own activation functions and weights and biases. If you have a neural network where the various parameters of different hidden layers are not affected by the previous layer, i.e.: the neural network does not have memory, then you can use a recurrent neural network.

The Recurrent Neural Network will standardize the different activation functions and weights and biases so that each hidden layer has the same parameters. Then, instead of creating multiple hidden layers, it will create one and loop over it as many times as required.

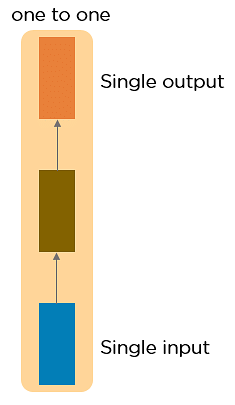
**Types of Recurrent Neural Networks**

There are four types of Recurrent Neural Networks:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

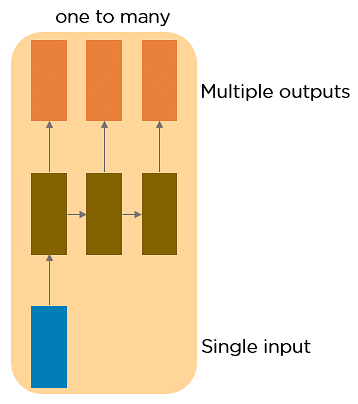
**One to One RNN**

This type of neural network is known as the Vanilla Neural Network. It's used for general machine learning problems, which has a single input and a single output.



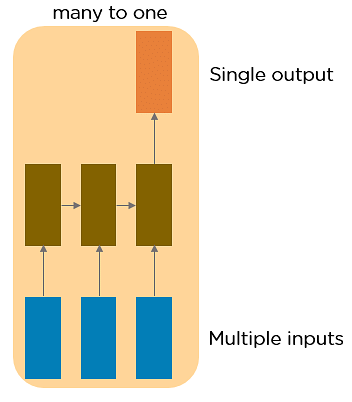
**One to Many RNN**

This type of neural network has a single input and multiple outputs. An example of this is the image caption.



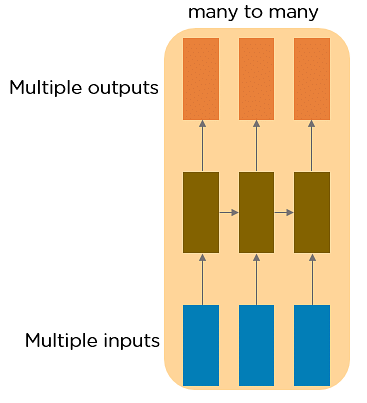
**Many to One RNN**

This RNN takes a sequence of inputs and generates a single output. Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.



**Many to Many RNN**

This RNN takes a sequence of inputs and generates a sequence of outputs. Machine translation is one of the examples.



**Two Issues of Standard RNNs**

**1. Vanishing Gradient Problem**

Recurrent Neural Networks enable you to model time-dependent and sequential data problems, such as stock market prediction, machine translation, and text generation. You will find, however, RNN is hard to train because of the gradient problem.

RNNs suffer from the problem of vanishing gradients. The gradients carry information used in the RNN, and when the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.

**2. Exploding Gradient Problem**

While training a neural network, if the slope tends to grow exponentially instead of decaying, this is called an Exploding Gradient. This problem arises when large error gradients accumulate, resulting in very large updates to the neural network model weights during the training process.

**Gradient Problem Solutions:**

1. **Vanishing Gradient Problem**

* Weight Initialization
* Echo State Network
* Long Short-Term Memory Network (LSTM)

1. **Exploding Gradient Problem**

* Truncated Backpropagation
* Penalties
* Gradient Clipping

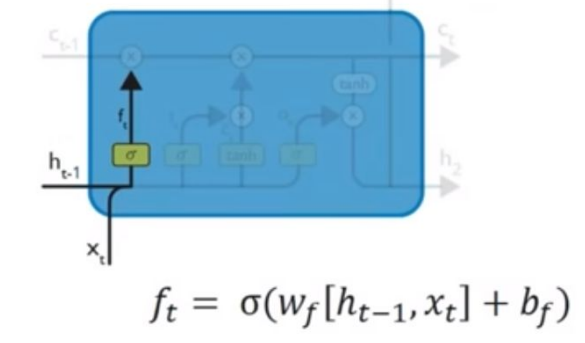
**Long Short-Term Memory Networks**

LSTMs are a special kind of RNN — capable of learning long-term dependencies by remembering information for long periods is the default behavior.

**Workings of LSTMs:**

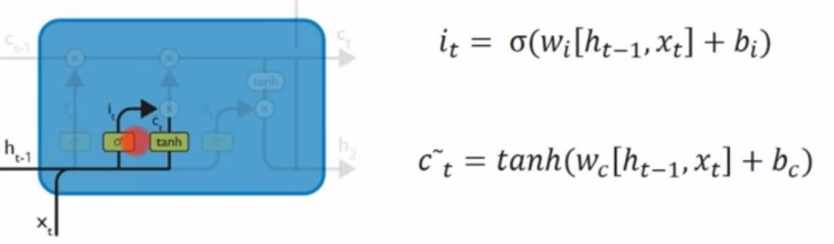
**Step 1: Decide How Much Past Data It Should Remember**

The first step in the LSTM is to identify those information that are not required and will be thrown away from the cell state. This decision is made by a sigmoid layer called as forget gate layer.

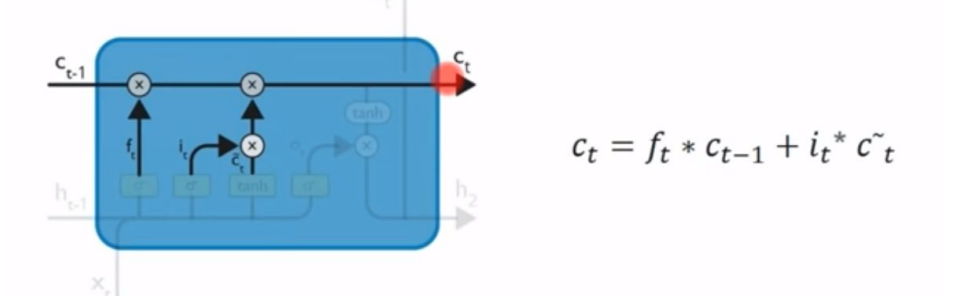


**Step 2: Decide How Much This Unit Adds to the Current Stat**

The next step is to decide, what new information we’re going to store in the cell state. This whole process comprises of the following steps. A sigmoid layer called the input gate layer decides which value will be updated. Next, a tanh layer creates a vector of new candidate values, that could be added to the state.



**Step 3:** In this step, we will update the old cell state, Ct-1, into the new cell state Ct. First, we multiply the old state (Ct-1) by Ft, forgetting the things we decided to forget earlier. Then, we add It \* Ct. This is the candidate values, scaled by how much we decided to update each state value.



**Step 4:** We will run a sigmoid layer which decide what parts of the cell state we’re going to output. Then, we put the cell state through tanh(push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

