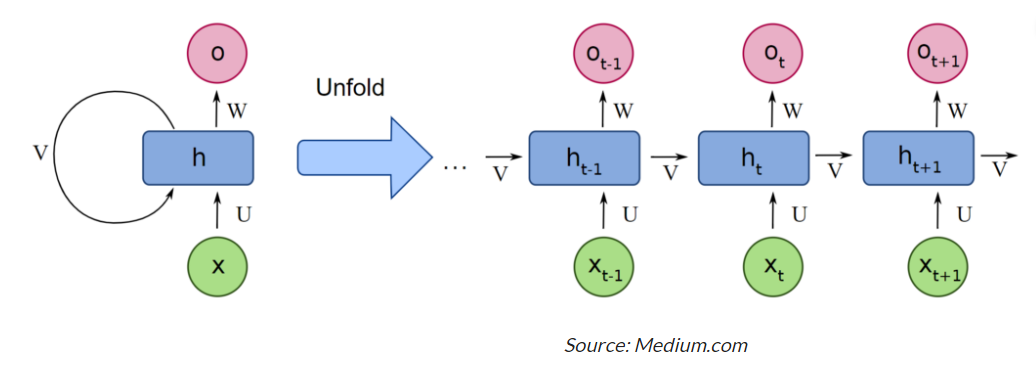
A Brief Overview of Recurrent Neural Networks (RNN)

Source: <https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/>

# Introduction on Recurrent Neural Networks

A Deep Learning approach for modelling sequential data is Recurrent Neural Networks (RNN). RNNs were the standard suggestion for working with sequential data before the advent of attention models. Specific parameters for each element of the sequence may be required by a deep feedforward model. It may also be unable to generalize to variable-length sequences.



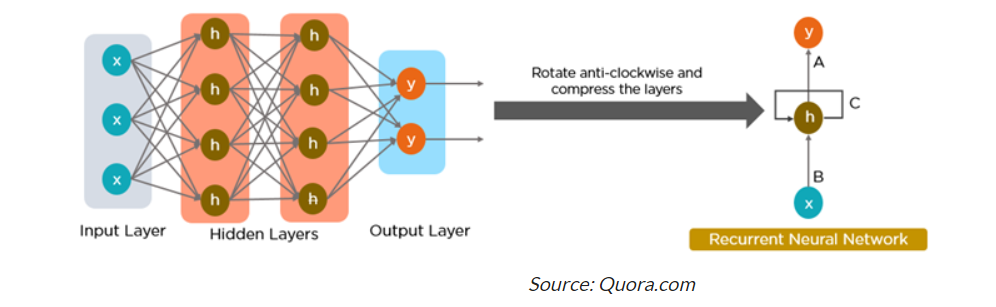
Recurrent Neural Networks use the same weights for each element of the sequence, decreasing the number of parameters and allowing the model to generalize to sequences of varying lengths. RNNs generalize to structured data other than sequential data, such as geographical or graphical data, because of its design.

Recurrent neural networks, like many other deep learning techniques, are relatively old. They were first developed in the 1980s, but we didn’t appreciate their full potential until lately. The advent of long short-term memory (LSTM) in the 1990s, combined with an increase in computational power and the vast amounts of data that we now have to deal with, has really pushed RNNs to the forefront.

# What is a Recurrent Neural Network (RNN)?

Neural networks imitate the function of the human brain in the fields of AI, machine learning, and deep learning, allowing computer programs to recognize patterns and solve common issues.

RNNs are a type of neural network that can be used to model sequence data. RNNs, which are formed from feedforward networks, are similar to human brains in their behaviour. Simply said, recurrent neural networks can anticipate sequential data in a way that other algorithms can’t.

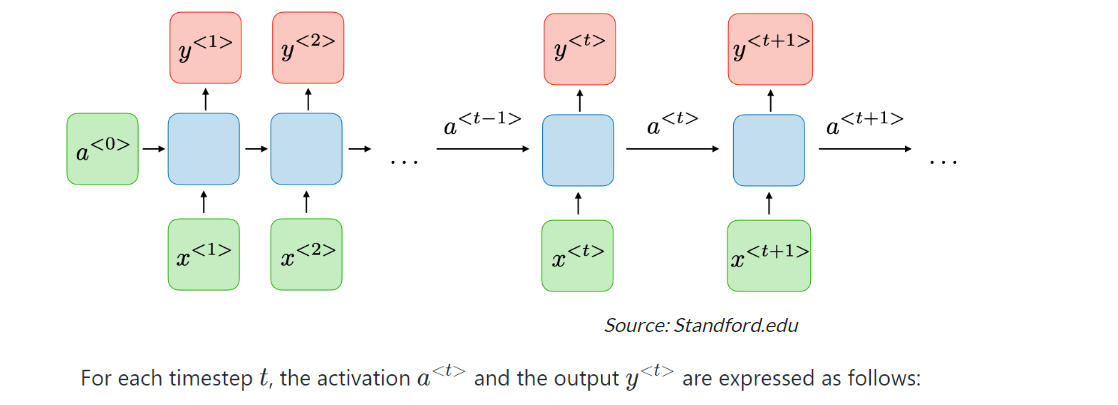


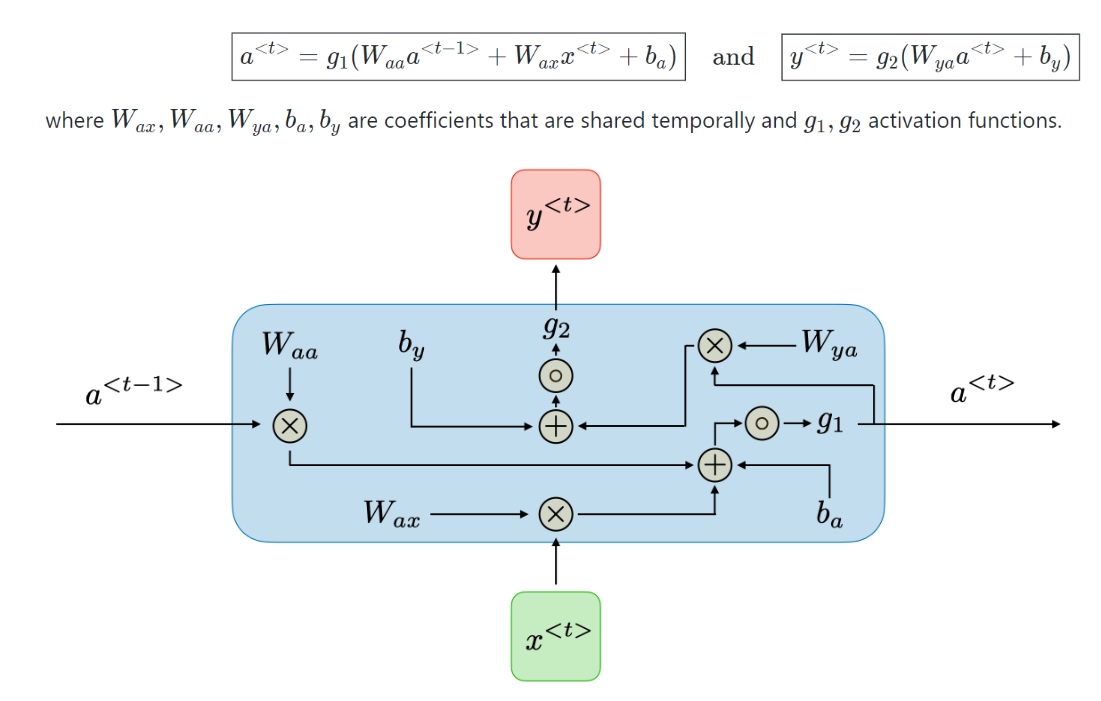
All of the inputs and outputs in standard neural networks are independent of one another, however in some circumstances, such as when predicting the next word of a phrase, the prior words are necessary, and so the previous words must be remembered. As a result, RNN was created, which used a Hidden Layer to overcome the problem. The most important component of RNN is the Hidden state, which remembers specific information about a sequence.

RNNs have a Memory that stores all information about the calculations. It employs the same settings for each input since it produces the same outcome by performing the same task on all inputs or hidden layers.

# The Architecture of a Traditional RNN

RNNs are a type of neural network that has hidden states and allows past outputs to be used as inputs. They usually go like this:





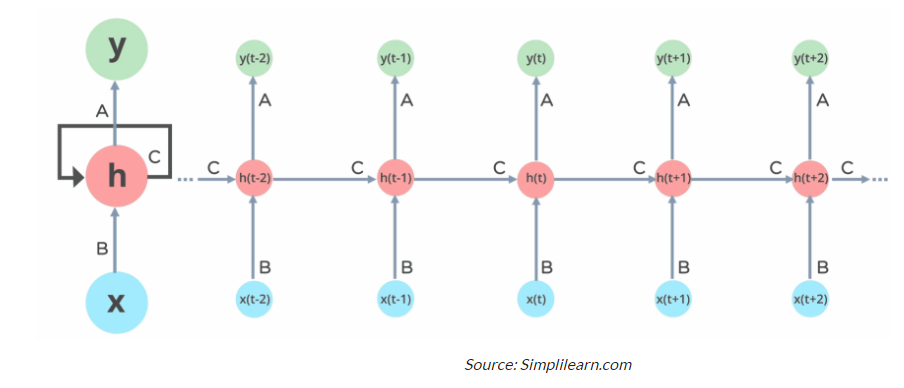
RNN architecture can vary depending on the problem you’re trying to solve. From those with a single input and output to those with many (with variations between).

Below are some examples of RNN architectures that can help you better understand this.

* **One To One:**There is only one pair here. A one-to-one architecture is used in traditional neural networks.
* **One To Many:**A single input in a one-to-many network might result in numerous outputs. One too many networks are used in the production of music, for example.
* **Many To One:**In this scenario, a single output is produced by combining many inputs from distinct time steps. Sentiment analysis and emotion identification use such networks, in which the class label is determined by a sequence of words.
* **Many To Many:**For many to many, there are numerous options. Two inputs yield three outputs. Machine translation systems, such as English to French or vice versa translation systems, use many to many networks.

# How does Recurrent Neural Networks work?

The information in recurrent neural networks cycles through a loop to the middle hidden layer.



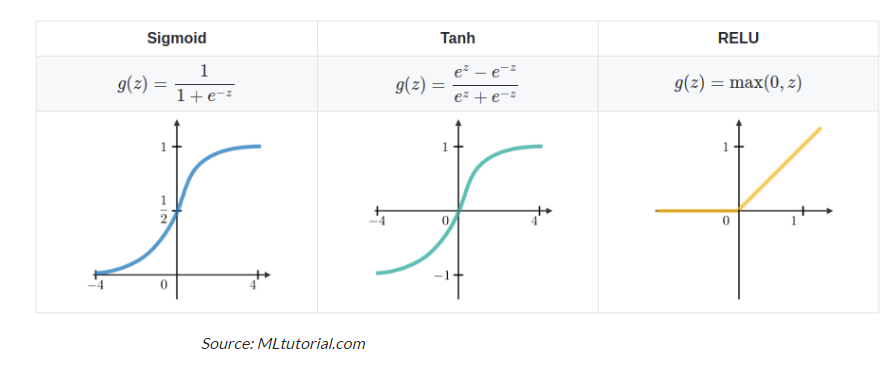
The input layer **x**receives and processes the neural network’s input before passing it on to the middle layer.

Multiple hidden layers can be found in the middle layer **h**, each with its own activation functions, weights, and biases. You can utilize a recurrent neural network if the various parameters of different hidden layers are not impacted by the preceding layer, i.e. There is no memory in the neural network.

The different activation functions, weights, and biases will be standardized by the Recurrent Neural Network, ensuring that each hidden layer has the same characteristics. Rather than constructing numerous hidden layers, it will create only one and loop over it as many times as necessary.

# Common Activation Functions

A neuron’s activation function dictates whether it should be turned on or off. Nonlinear functions usually transform a neuron’s output to a number between 0 and 1 or -1 and 1.



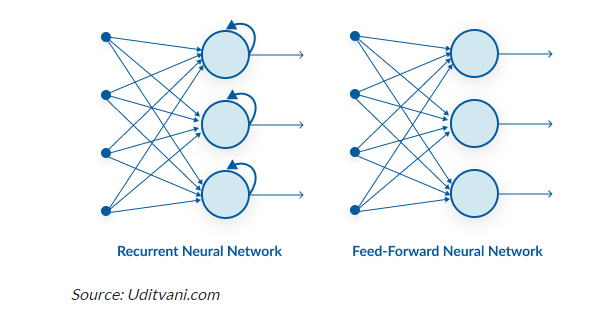
The following are some of the most commonly utilized functions:

* **Sigmoid:**The formula **g(z) = 1/(1 + e^-z)** is used to express this.
* **Tanh:**The formula **g(z) = (e^-z – e^-z)/(e^-z + e^-z)** is used to express this.
* **Relu:**The formula **g(z) = max(0 , z)** is used to express this.

# Recurrent Neural Network Vs Feedforward Neural Network

A feed-forward neural network has only one route of information flow: from the input layer to the output layer, passing through the hidden layers. The data flows across the network in a straight route, never going through the same node twice.

The information flow between an RNN and a feed-forward neural network is depicted in the two figures below.



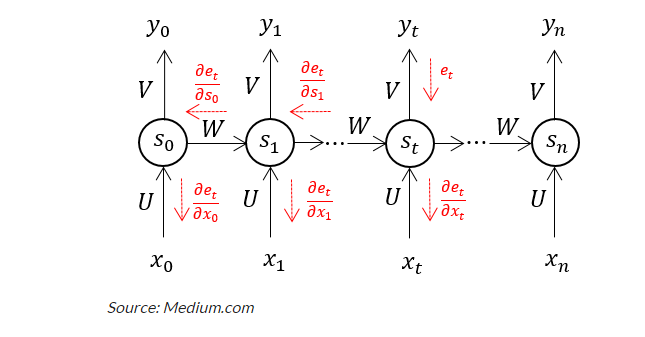
Feed-forward neural networks are poor predictions of what will happen next because they have no memory of the information they receive. Because it simply analyses the current input, a feed-forward network has no idea of temporal order. Apart from its training, it has no memory of what transpired in the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

# Backpropagation Through Time (BPTT)

When we apply a Backpropagation algorithm to a Recurrent Neural Network with time series data as its input, we call it backpropagation through time.

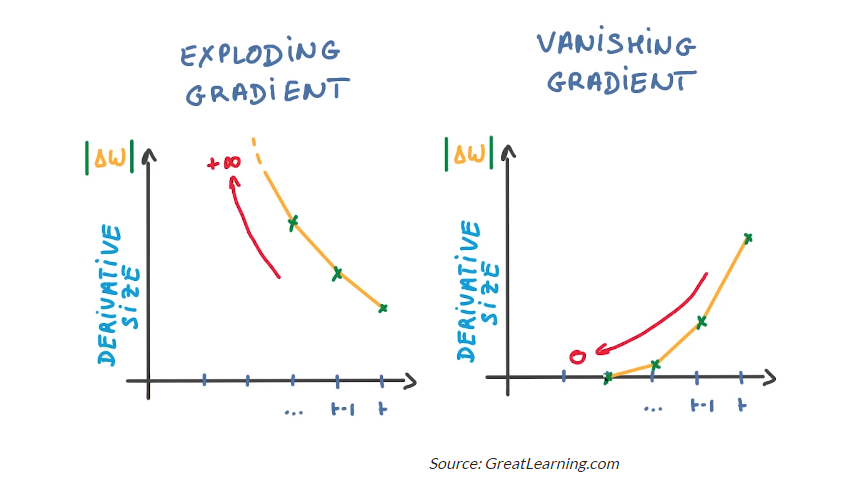
A single input is sent into the network at a time in a normal RNN, and a single output is obtained. Backpropagation, on the other hand, uses both the current and prior inputs as input. This is referred to as a timestep, and one timestep will consist of multiple time series data points entering the RNN at the same time.



The output of the neural network is used to calculate and collect the errors once it has trained on a time set and given you an output. The network is then rolled back up, and weights are recalculated and adjusted to account for the faults

# Two issues of Standard RNNs

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.

# RNN Applications

Recurrent Neural Networks are used to tackle a variety of problems involving sequence data. There are many different types of sequence data, but the following are the most common: Audio, Text, Video, Biological sequences.

Using RNN models and sequence datasets, you may tackle a variety of problems, including :

* Speech recognition
* Generation of music
* Automated Translations
* Analysis of video action
* Sequence study of the genome and DNA

# Conclusion

* Recurrent Neural Networks are a versatile tool that can be used in a variety of situations. They’re employed in a variety of methods for language modelling and text generators. They’re also employed in voice recognition.
* This type of neural network is used to create labels for images that aren’t tagged when paired with Convolutional Neural Networks. It’s incredible how well this combination works.
* However, there is one flaw with recurrent neural networks. They have trouble learning long-range dependencies, which means they don’t comprehend relationships between data that are separated by several steps.
* When anticipating words, for example, we may require more context than simply one prior word. This is known as the vanishing gradient problem, and it is solved using a special type of Recurrent Neural Network called Long-Short Term Memory Networks (LSTM), which is a larger topic that will be discussed in future articles.

Code: [assignment\_internship/Basic Python Implementation (RNN with Keras).ipynb at main · Hntam812/assignment\_internship (github.com)](https://github.com/Hntam812/assignment_internship/blob/main/Basic%20Python%20Implementation%20(RNN%20with%20Keras).ipynb)

LSTM Vs GRU in Recurrent Neural Network: A Comparative Study

Long Short Term Memory in short LSTM is a special kind of RNN capable of learning long term sequences. They were introduced by Schmidhuber and Hochreiter in 1997. It is explicitly designed to avoid long term dependency problems. Remembering the long sequences for a long period of time is its way of working.

A [recurrent neural network](https://analyticsindiamag.com/implementing-a-recurrent-neural-network-rnn-from-scratch/) is a type of [ANN](https://analyticsindiamag.com/ann-with-linear-regression/) that is used when users want to perform predictive operations on sequential or [time-series](https://analyticsindiamag.com/guide-to-implementing-time-series-analysis-predicting-bitcoin-price-with-rnn/) based data. These Deep learning layers are commonly used for ordinal or temporal problems such as [Natural Language Processing](https://analyticsindiamag.com/how-to-identify-entities-in-nlp/), Neural Machine Translation, [automated image captioning tasks](https://analyticsindiamag.com/hands-on-guide-to-effective-image-captioning-using-attention-mechanism/) and likewise. Today’s modern voice assistance devices such as Google Assistance, Alexa, Siri are incorporated with these layers to fulfil hassle-free experiences for users.

In RNN to train networks, we backpropagate through time and at each time step or loop operation gradient is being calculated and the gradient is used to update the weights in the networks. Now if the effect of the previous sequence on the layer is small then the relative gradient is calculated small. Then if the gradient of the previous layer is smaller then this makes weights to be assigned to the context smaller and this effect is observed when we deal with longer sequences. Due to this network does not learn the effect of earlier inputs and thus causing the short term memory problem.

To overcome this problem specialized versions of RNN are created like LSTM, GRU,…

# ****Working of LSTM****

**What is Long Short Term Memory or LSTM?**

Long Short Term Memory in short LSTM is a special kind of RNN capable of learning long term sequences. They were introduced by Schmidhuber and Hochreiter in 1997. It is explicitly designed to avoid long term dependency problems. Remembering the long sequences for a long period of time is its way of working.

The popularity of LSTM is due to the Getting mechanism involved with each LSTM cell. In a normal RNN cell, the input at the time stamp and hidden state from the previous time step is passed through the activation layer to obtain a new state. Whereas in LSTM the process is slightly complex, as you can see in the above architecture at each time it takes input from three different states like the current input state, the short term memory from the previous cell and lastly the long term memory.

These cells use the gates to regulate the information to be kept or discarded at loop operation before passing on the long term and short term information to the next cell. We can imagine these gates as Filters that remove unwanted selected and irrelevant information. There are a total of three gates that LSTM uses as Input Gate, Forget Gate, and Output Gate.

Input Gate

The input gate decides what information will be stored in long term memory. It only works with the information from the current input and short term memory from the previous step. At this gate, it filters out the information from variables that are not useful.

### **Forget Gate**

The forget decides which information from long term memory be kept or discarded and this is done by multiplying the incoming long term memory by a forget vector generated by the current input and incoming short memory.

### **Output Gate**

The output gate will take the current input, the previous short term memory and newly computed long term memory to produce new short term memory which will be passed on to the cell in the next time step. The output of the current time step can also be drawn from this hidden state.

So this is all about the mechanism of LSTM to realise this with practical implementation. [Here](https://colab.research.google.com/drive/1-c36k29ptuMUXUrQzH5SrQ-24t5pFZZL?usp=sharing) I have demonstrated the LSTM use case in which you can check input and output sequences with their shape.

# ****Working of GRU****

**What is Gated Recurrent Unit or GRU?**

The workflow of the Gated Recurrent Unit, in short GRU, is the same as the RNN but the difference is in the operation and gates associated with each GRU unit. To solve the problem faced by standard RNN, GRU incorporates the two gate operating mechanisms called Update gate and Reset gate.

### **Update gate**

The update gate is responsible for determining the amount of previous information that needs to pass along the next state. This is really powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient.

### **Reset gate**

The reset gate is used from the model to decide how much of the past information is needed to neglect; in short, it decides whether the previous cell state is important or not.

First, the reset gate comes into action it stores relevant information from the past time step into new memory content. Then it multiplies the input vector and hidden state with their weights. Next, it calculates element-wise multiplication between the reset gate and previously hidden state multiple. After summing up the above steps the non-linear activation function is applied and the next sequence is generated.

This is all about the operation of GRU, the practical examples are included in the notebooks.

# ****What is the difference between GRU & LSTM?****

The few differencing points are as follows:  
The GRU has two gates, LSTM has three gates  
GRU does not possess any internal memory, they don’t have an output gate that is present in LSTM  
In LSTM the input gate and target gate are coupled by an update gate and in GRU reset gate is applied directly to the previous hidden state. In LSTM the responsibility of reset gate is taken by the two gates i.e., input and target.

# Conclusion

Through this article, we have understood the basic difference between the RNN, LSTM and GRU units. From working of both layers i.e., LSTM and GRU, GRU uses less training parameter and therefore uses less memory and executes faster than LSTM whereas LSTM is more accurate on a larger dataset. One can choose LSTM if you are dealing with large sequences and accuracy is concerned, GRU is used when you have less memory consumption and want faster results.

## **10.1.2. Implementation from Scratch**

we first load The Time Machine dataset.

### **1. Initializing Model Parameters**

Next, we need to define and initialize the model parameters. As previously, the hyperparameter num\_hiddens dictates the number of hidden units. We initialize weights following a Gaussian distribution with 0.01 standard deviation, and we set the biases to 0

The actual model is defined as described above, consisting of three gates and an input node. Note that only the hidden state is passed to the output layer.