1. **Explain the basic architecture of RNN cell.**

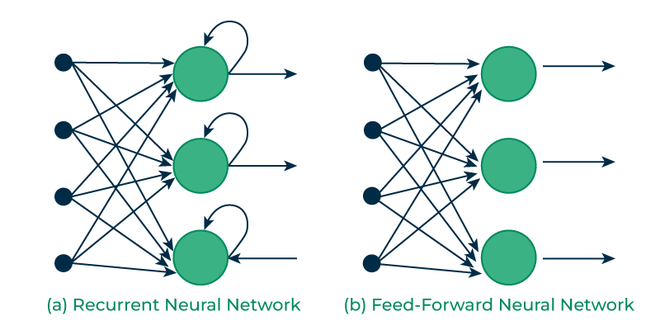
**What is Recurrent Neural Network (RNN)?**

Recurrent Neural Network(RNN) is a type of [Neural Network](https://www.geeksforgeeks.org/tag/neural-network/) where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as *Memory State*since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

**How RNN differs from Feedforward Neural Network?**

[Artificial neural networks](https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/) that do not have looping nodes are called feed forward neural networks. Because all information is only passed forward, this kind of neural network is also referred to as a [multi-layer neural network](https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/).

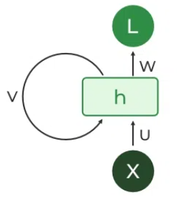
Information moves from the input layer to the output layer – if any hidden layers are present – unidirectionally in a feedforward neural network. These networks are appropriate for image classification tasks, for example, where input and output are independent. Nevertheless, their inability to retain previous inputs automatically renders them less useful for sequential data analysis.



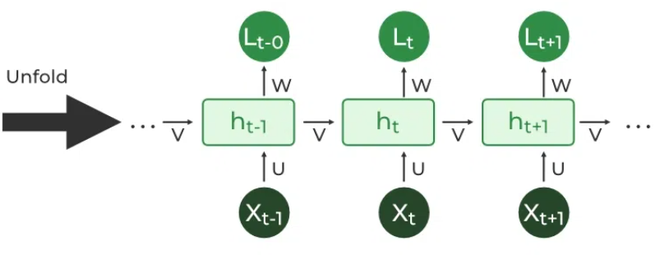
*Recurrent Vs Feedfoward networks*

**Recurrent Neuron and RNN Unfolding**

The fundamental processing unit in a Recurrent Neural Network (RNN) is a Recurrent Unit, which is not explicitly called a “Recurrent Neuron.” This unit has the unique ability to maintain a hidden state, allowing the network to capture sequential dependencies by remembering previous inputs while processing. [Long Short-Term Memory (LSTM)](https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/) and [Gated Recurrent Unit (GRU)](https://www.geeksforgeeks.org/gated-recurrent-unit-networks/) versions improve the RNN’s ability to handle long-term dependencies.



*Recurrent Neuron*



*RNN Unfolding*

**Types Of RNN**

There are four types of RNNs based on the number of inputs and outputs in the network.

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

**One to One**

This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.

*One to One RNN*

**One To Many**

In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.

*One to Many RNN*

**Many to One**

In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.

*Many to One RNN*

**Many to Many**

In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.

*Many to Many RNN*

**Recurrent Neural Network Architecture**

RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state  H**i**for every input **Xi . By using the following formulas:**

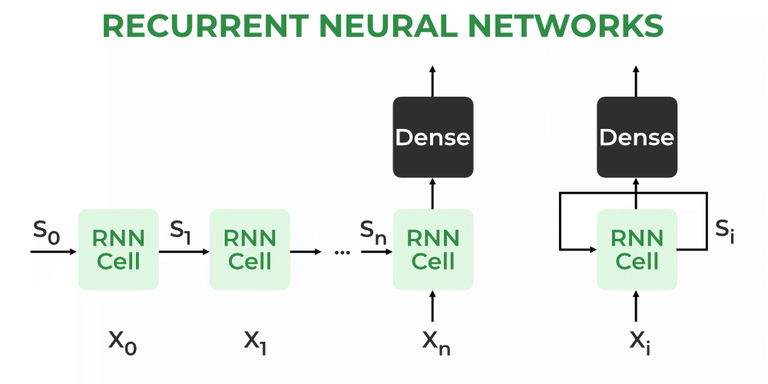
*h= σ(UX + Wh-1 + B)*

*Y = O(Vh + C)*

*Hence*

*Y = f (X, h , W, U, V, B, C)*

*Here S is the State matrix which has element si as the state of the network at timestep i  
The parameters in the network are W, U, V, c, b which are shared across timestep*



*Recurrent Neural Architecture*

**How does RNN work?**

The Recurrent Neural Network consists of multiple fixed [activation function](https://www.geeksforgeeks.org/activation-functions-neural-networks/) units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

**The formula for calculating the current state:**

where,

* ht -> current state
* ht-1 -> previous state
* xt -> input state

**Formula for applying Activation function(tanh)**

where,

* whh -> weight at recurrent neuron
* wxh -> weight at input neuron

**The formula for calculating output:**

* Yt -> output
* Why -> weight at output layer

These parameters are updated using [Backpropagation](https://www.geeksforgeeks.org/ml-back-propagation-through-time/). However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

**Backpropagation Through Time (BPTT)**

In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h1 then h2 then h3 so on. Hence we will apply backpropagation throughout all these hidden time states sequentially.

*Backpropagation Through Time (BPTT) In RNN*

* L(θ)(loss function) depends on h3
* h3 in turn depends on h2 and W
* h2 in turn depends on h1 and W
* h1 in turn depends on h0 and W
* where h0 is a constant starting state.

**For simplicity of this equation, we will apply backpropagation on only one row**

We already know how to compute this one as it is the same as any simple deep neural network backpropagation.

However, we will see how to apply backpropagation to this term

As we know h3 = σ(Wh2 + b)

And In such an ordered network, we can’t compute  by simply treating h3 as a constant because as it also depends on W. the total derivative  has two parts:

1. **Explicit:**treating all other inputs as constant
2. **Implicit:** Summing over all indirect paths from h3 to W

**Let us see how to do this**

**For simplicity, we will short-circuit some of the paths**

**Finally, we have**

**Where**

**Hence,**

This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

**Issues of Standard RNNs**

1. **Vanishing Gradient:**Text generation, machine translation, and stock market prediction are just a few examples of the time-dependent and sequential data problems that can be modelled with recurrent neural networks. You will discover, though, that the gradient problem makes training RNN difficult.
2. **Exploding Gradient:** An Exploding Gradient occurs when a neural network is being trained and the slope tends to grow exponentially rather than decay. Large error gradients that build up during training lead to very large updates to the neural network model weights, which is the source of this issue.

**Training through RNN**

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained using Backpropagation through time.

**Advantages and Disadvantages of Recurrent Neural Network**

**Advantages**

1. An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called [Long Short Term Memory](https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/).
2. Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

**Disadvantages**

1. [Gradient vanishing](https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/) and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

**Applications of Recurrent Neural Network**

1. Language Modelling and Generating Text
2. Speech Recognition
3. Machine Translation
4. Image Recognition, Face detection
5. Time series Forecasting

**Variation Of Recurrent Neural Network (RNN)**

To overcome the problems like vanishing gradient and exploding gradient descent several new advanced versions of RNNs are formed some of these are as;

1. Bidirectional Neural Network (BiNN)
2. Long Short-Term Memory (LSTM)

**Bidirectional Neural Network (BiNN)**

A BiNN is a variation of a Recurrent Neural Network in which the input information flows in both direction and then the output of both direction are combined to produce the input. BiNN is useful in situations when the context of the input is more important such as Nlp tasks and Time-series analysis problems.

**Long Short-Term Memory (LSTM)**

Long Short-Term Memory works on the read-write-and-forget principle where given the input information network reads and writes the most useful information from the data and it forgets about the information which is not important in predicting the output. For doing this three new gates are introduced in the RNN. In this way, only the selected information is passed through the network.

**Difference between RNN and Simple Neural Network**

RNN is considered to be the better version of deep neural when the data is sequential. There are significant differences between the RNN and deep neural networks  they are listed as:

| Recurrent Neural Network | Deep Neural Network |
| --- | --- |
| Weights are same across all the layers number of a Recurrent Neural Network | Weights are different for each layer of the network |
| Recurrent Neural Networks are used when the data is sequential and the number of inputs is not predefined. | A Simple Deep Neural network does not have any special method for sequential data also here the the number of inputs is fixed |
| The Numbers of parameter in the RNN are higher than in simple DNN | The Numbers of Parameter are lower than RNN |
| Exploding and vanishing gradients is the  the major drawback of RNN | These problems also occur in DNN but these are not the major problem with DNN |

1. **ExplainBackpropagation through time (BPTT)**

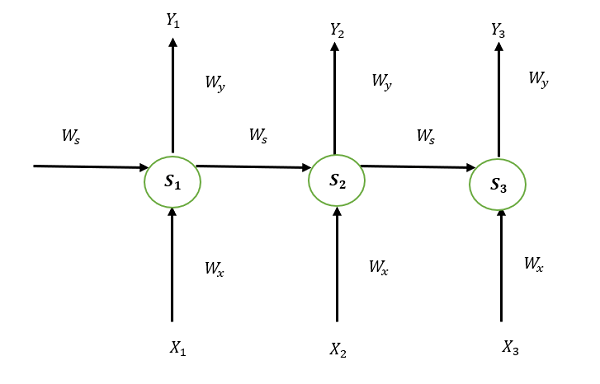
# Back Propagation through time - RNN

### Introduction:

Recurrent Neural Networks are those networks that deal with sequential data. They can predict outputs based on not only current inputs but also considering the inputs that were generated prior to it. The output of the present depends on the output of the present and the memory element (which includes the previous inputs).

To train these networks, we make use of traditional backpropagation with an added twist. We don't train the system on the exact time "t". We train it according to a particular time "t" as well as everything that has occurred prior to time "t" like the following: t-1, t-2, t-3.

Take a look at the following illustration of the RNN:



S1, S2, and S3 are the states that are hidden or memory units at the time of t1, t2, and t3, respectively, while Ws represents the matrix of weight that goes with it.

X1, X2, and X3 are the inputs for the time that is t1, t2, and t3, respectively, while Wx represents the weighted matrix that goes with it.

The numbers Y1, Y2, and Y3 are the outputs of t1, t2, and t3, respectively as well as Wy, the weighted matrix that goes with it.

For any time, t, we have the following two equations:

St = g1 (Wx xt + Ws St-1)  
Yt = g2 (WY St )

where g1 and g2 are activation functions.

We will now perform the back propagation at time t = 3.

Let the error function be:

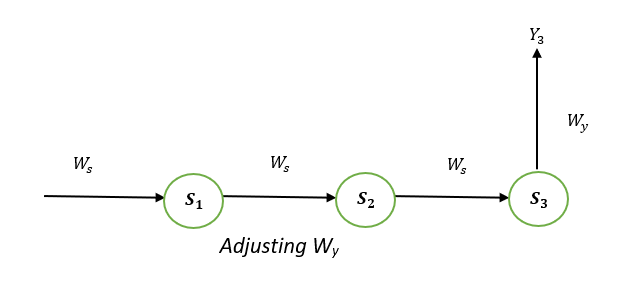
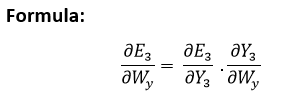
Et=(dt-Yt )2

Here, we employ the squared error, in which D3 is the desired output at a time t = 3.

In order to do backpropagation, it is necessary to change the weights that are associated with inputs, memory units, and outputs.

## Adjusting Wy

To better understand, we can look at the following image:

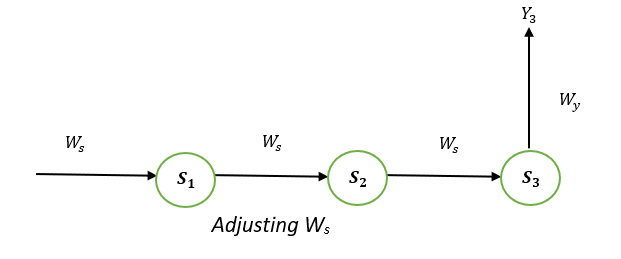
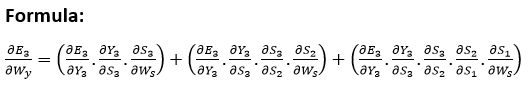
  
**Explanation:**

E3 is a function of Y3. Hence, we differentiate E3 with respect to Y3.

Y3 is a function of W3. Hence, we differentiate Y3 with respect to W3.

## Adjusting Ws

To better understand, we can look at the following image:

**Explanation:**

E3 is a function of the Y3. Therefore, we distinguish the E3 with respect to Y3. Y3 is a function of the S3. Therefore, we differentiate between Y3 with respect to S3.

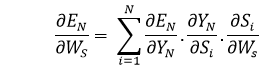
S3 is an element in the Ws. Therefore, we distinguish between S3 with respect to Ws.

But it's not enough to stop at this, therefore we have to think about the previous steps in time. We must also differentiate (partially) the error function in relation to the memory units S2 and S1, considering the weight matrix Ws.

It is essential to be aware that a memory unit, such as St, is the result of its predecessor memory unit, St-1.

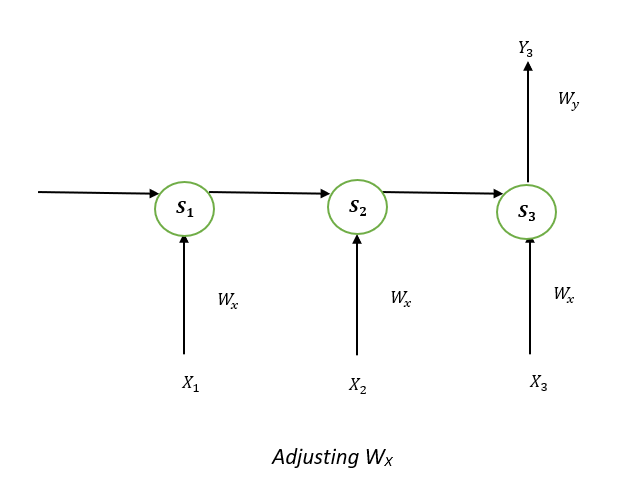
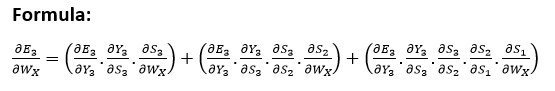
Therefore, we distinguish S3 from S2 and S2 from S1.

In general, we can describe this formula in terms of:



## Adjusting WX:

To better understand, we can look at the following image:

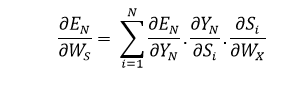
**Explanation:**

E3 is an effect in the Y3. Therefore, we distinguish the E3 with respect to Y3. Y3 is an outcome that is a function of the S3. Therefore, we distinguish the Y3 with respect to S3.

S3 is an element in the WX. Thus, we can distinguish the S3 with respect to WX.

We can't just stop at this, and therefore we also need to think about the preceding time steps. Therefore, we separate (partially) the error function in relation to the memory unit S2 and S1, considering the WX weighting matrix.

In general, we can define this formula in terms of:



**Limitations:**

This technique that uses the back Propagation over time (BPTT) is a method that can be employed for a limited amount of time intervals, like 8 or 10. If we continue to backpropagate and the gradient gets too small. This is known as the "Vanishing gradient" problem. This is because the value of information diminishes geometrically with time. Therefore, if the number of time steps is greater than 10 (Let's say), the data is effectively discarded.

## Going Beyond RNNs:

One of the most famous solutions to this issue is using what's known as Long-Short-Term Memory (LSTM for short) cells instead of conventional RNN cells. However, there could be another issue, referred to as the **explosion gradient** problem, in which the gradient becomes uncontrollably high.

**Solution:**

A well-known method is known as gradient clipping when for each time step, we will determine if the gradient **δ** is greater than the threshold. If it is, then we should normalize it.

1. **ExplainVanishing and exploding gradients**

# Vanishing and Exploding Gradients Problems in Deep Learning

In the realm of deep learning, the optimization process plays a crucial role in training neural networks. Gradient descent, a fundamental optimization algorithm, can sometimes encounter two common issues: vanishing gradients and exploding gradients. In this article, we will delve into these challenges, providing insights into what they are, why they occur, and how to mitigate them. We will build and train a model, and learn how to face vanishing and exploding problems.

## What is Vanishing Gradient?

The vanishing [gradient](https://www.geeksforgeeks.org/gradient-descent-in-linear-regression/) problem is a challenge that emerges during backpropagation when the derivatives or slopes of the activation functions become progressively smaller as we move backward through the layers of a neural network. This phenomenon is particularly prominent in deep networks with many layers, hindering the effective training of the model. The weight updates becomes extremely tiny, or even exponentially small, it can significantly prolong the training time, and in the worst-case scenario, it can halt the training process altogether.

### Why the Problem Occurs?

During backpropagation, the gradients propagate back through the layers of the network, they decrease significantly. This means that as they leave the output layer and return to the input layer, the gradients become progressively smaller. As a result, the weights associated with the initial levels, which accommodate these small gradients, are updated little or not at each iteration of the optimization process.

**The vanishing gradient problem** is particularly associated with the sigmoid and hyperbolic tangent (tanh) [activation functions](https://www.geeksforgeeks.org/activation-functions/) because their derivatives fall within the range of 0 to 0.25 and 0 to 1, respectively. Consequently, extreme weights becomes very small, causing the updated weights to closely resemble the original ones. This persistence of small updates contributes to the vanishing gradient issue.

The sigmoid and tanh functions limit the input values ​​to the ranges [0,1] and [-1,1], so that they saturate at 0 or 1 for sigmoid and -1 or 1 for Tanh. The derivatives at points becomes zero as they are moving. In these regions, especially when inputs are very small or large, the gradients are very close to zero. While this may not be a major concern in shallow networks with a few layers, it is a more pronounced issue in deep networks. When the inputs fall in saturated regions, the gradients approach zero, resulting in little update to the weights of the previous layer. In simple networks this does not pose much of a problem, but as more layers are added, these small gradients, which multiply between layers, decay significantly and consequently the first layer tears very slowly , and hinders overall model performance and can lead to convergence failure.

### How can we identify?

Identifying the vanishing gradient problem typically involves monitoring the training dynamics of a deep neural network.

* One key indicator is observing model weights**converging to 0** or stagnation in the improvement of the model’s performance metrics over training epochs.
* During training, if the **loss function fails to decrease**significantly, or if there is erratic behavior in the learning curves, it suggests that the gradients may be vanishing.
* Additionally, examining the gradients themselves during backpropagation can provide insights. **Visualization techniques**, such as gradient histograms or norms, can aid in assessing the distribution of gradients throughout the network.

### How can we solve the issue?

* **Batch Normalization**: Batch normalization normalizes the inputs of each layer, reducing internal covariate shift. This can help stabilize and accelerate the training process, allowing for more consistent gradient flow.
* **Activation function**: Activation function like **Rectified Linear Unit (ReLU)** can be used. With **ReLU,** the gradient is 0 for negative and zero input, and it is 1 for positive input, which helps alleviate the vanishing gradient issue. Therefore, ReLU operates by replacing poor enter values with 0, and 1 for fine enter values, it preserves the input unchanged.
* **Skip Connections and Residual Networks (ResNets)**: Skip connections, as seen in ResNets, allow the gradient to bypass certain layers during backpropagation. This facilitates the flow of information through the network, preventing gradients from vanishing.
* **Long Short-Term Memory Networks (LSTMs) and Gated Recurrent Units (GRUs)**: In the context of recurrent neural networks (RNNs), architectures like LSTMs and GRUs are designed to address the vanishing gradient problem in sequences by incorporating gating mechanisms .
* **Gradient Clipping**: Gradient clipping involves imposing a threshold on the gradients during backpropagation. Limit the magnitude of gradients during backpropagation, this can prevent them from becoming too small or exploding, which can also hinder learning.

1. **ExplainLong short-term memory (LSTM)**

# Deep Learning | Introduction to Long Short Term Memory

**Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber. LSTM is well-suited for sequence prediction tasks and excels in capturing long-term dependencies. Its applications extend to tasks involving time series and sequences. LSTM’s strength lies in its ability to grasp the order dependence crucial for solving intricate problems, such as machine translation and speech recognition. The article provides an in-depth introduction to LSTM, covering the LSTM model, architecture, working principles, and the critical role they play in various applications.**

## What is LSTM?

**A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as**[**language translation**](https://www.geeksforgeeks.org/language-translator-using-google-api-in-python/)**, speech recognition, and**[**time series forecasting**](https://www.geeksforgeeks.org/time-series-forecasting-using-recurrent-neural-networks-rnn-in-tensorflow/)**. LSTMs can also be used in combination with other neural network architectures, such as [Convolutional Neural Networks](https://www.geeksforgeeks.org/introduction-convolution-neural-network/) (CNNs) for image and video analysis.**

**The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell. The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.**

### Bidirectional LSTM

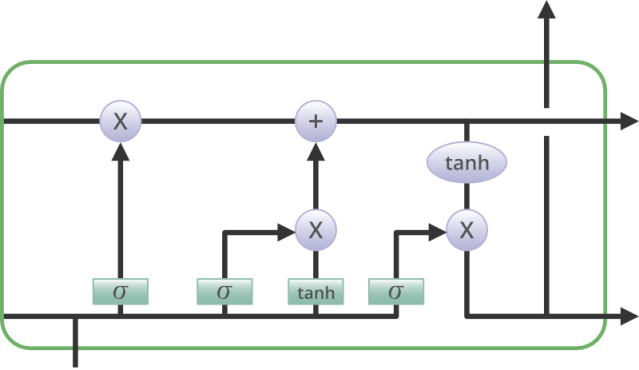
[**Bidirectional LSTM**](https://www.geeksforgeeks.org/bidirectional-lstm-in-nlp/)**(Bi LSTM/ BLSTM) is recurrent neural network (RNN) that is able to process sequential data in both forward and backward directions. This allows Bi LSTM to learn longer-range dependencies in sequential data than traditional LSTMs, which can only process sequential data in one direction.**

* **Bi LSTMs are made up of two LSTM networks, one that processes the input sequence in the forward direction and one that processes the input sequence in the backward direction. The outputs of the two LSTM networks are then combined to produce the final output.**
* **Bi LSTM have been shown to achieve state-of-the-art results on a wide variety of tasks, including machine translation, speech recognition, and text summarization.**

**LSTMs can be stacked to create deep LSTM networks, which can learn even more complex patterns in sequential data. Each LSTM layer captures different levels of abstraction and temporal dependencies in the input data.**

## Architecture and Working of LSTM

**LSTM architecture has a chain structure that contains four neural networks and different memory blocks called cells.**

****

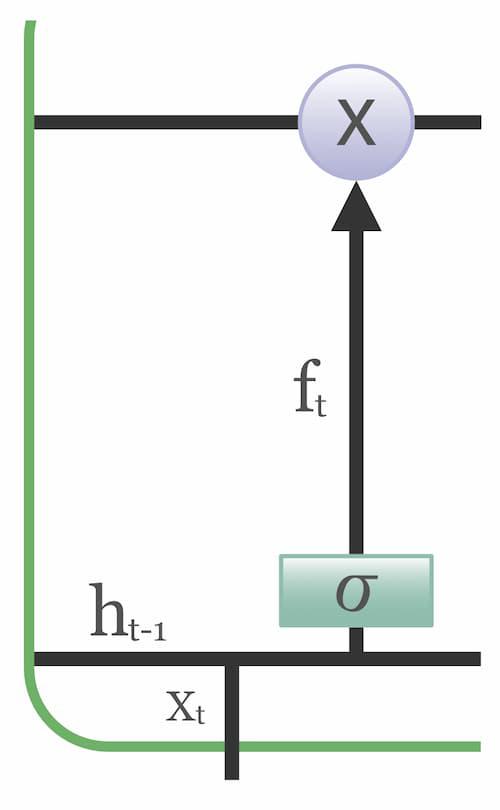
**Information is retained by the cells and the memory manipulations are done by the gates. There are three gates –**

### ****Forget Gate****

**The information that is no longer useful in the cell state is removed with the forget gate. Two inputs xt (input at the particular time) and ht-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use. The equation for the forget gate is:**

**where:**

* **W\_f represents the weight matrix associated with the forget gate.**
* **[h\_t-1, x\_t] denotes the concatenation of the current input and the previous hidden state.**
* **b\_f is the bias with the forget gate.**
* **σ is the sigmoid activation function.**

****

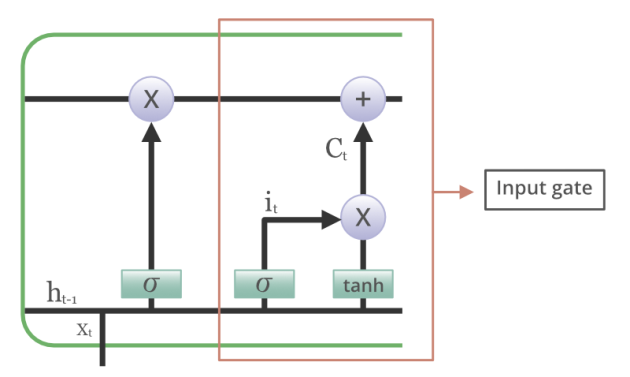
### ****Input gate****

**The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs ht-1 and xt. . Then, a vector is created using tanh function that gives an output from -1 to +1, which contains all the possible values from ht-1 and xt. At last, the values of the vector and the regulated values are multiplied to obtain the useful information. The equation for the input gate is:**

**We multiply the previous state by ft, disregarding the information we had previously chosen to ignore. Next, we include it∗Ct. This represents the updated candidate values, adjusted for the amount that we chose to update each state value.**

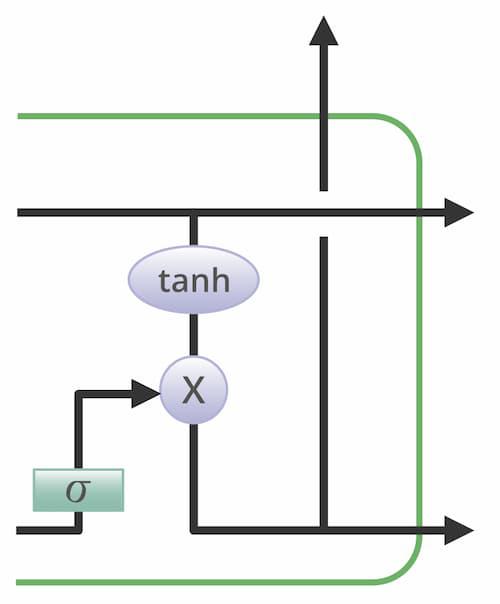
**where**

* **⊙ denotes element-wise multiplication**
* **tanh is tanh activation function**

****

### ****Output gate****

**The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs ht-1 and xt. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell. The equation for the output gate is:**

****

## LTSM vs RNN

| **Feature** | **LSTM (Long Short-term Memory)** | **RNN (Recurrent Neural Network)** |
| --- | --- | --- |
| **Memory** | **Has a special memory unit that allows it to learn long-term dependencies in sequential data** | **Does not have a memory unit** |
| **Directionality** | **Can be trained to process sequential data in both forward and backward directions** | **Can only be trained to process sequential data in one direction** |
| **Training** | **More difficult to train than RNN due to the complexity of the gates and memory unit** | **Easier to train than LSTM** |
| **Long-term dependency learning** | **Yes** | **Limited** |
| **Ability to learn sequential data** | **Yes** | **Yes** |
| **Applications** | **Machine translation, speech recognition, text summarization, natural language processing, time series forecasting** | **Natural language processing, machine translation, speech recognition, image processing, video processing** |

## ****Advantages and Disadvantages of LSTM****

**The advantages of LSTM (Long-Short Term Memory) are as follows:**

* **Long-term dependencies can be captured by LSTM networks. They have a memory cell that is capable of long-term information storage.**
* **In traditional RNNs, there is a problem of vanishing and exploding gradients when models are trained over long sequences. By using a gating mechanism that selectively recalls or forgets information, LSTM networks deal with this problem.**
* **LSTM enables the model to capture and remember the important context, even when there is a significant time gap between relevant events in the sequence. So where understanding context is important, LSTMS are used. eg. machine translation.**

**The disadvantages of LSTM (Long-Short Term Memory) are as follows:**

* **Compared to simpler architectures like feed-forward neural networks LSTM networks are computationally more expensive. This can limit their scalability for large-scale datasets or constrained environments.**
* **Training LSTM networks can be more time-consuming compared to simpler models due to their computational complexity. So training LSTMs often requires more data and longer training times to achieve high performance.**
* **Since it is processed word by word in a sequential manner, it is hard to parallelize the work of processing the sentences.**

## ****Applications of LSTM****

**Some of the famous applications of LSTM includes:**

* **Language Modeling: LSTMs have been used for natural language processing tasks such as language modeling, machine translation, and text summarization. They can be trained to generate coherent and grammatically correct sentences by learning the dependencies between words in a sentence.**
* **Speech Recognition: LSTMs have been used for speech recognition tasks such as transcribing speech to text and recognizing spoken commands. They can be trained to recognize patterns in speech and match them to the corresponding text.**
* **Time Series Forecasting: LSTMs have been used for time series forecasting tasks such as predicting stock prices, weather, and energy consumption. They can learn patterns in time series data and use them to make predictions about future events.**
* **Anomaly Detection: LSTMs have been used for anomaly detection tasks such as detecting fraud and network intrusion. They can be trained to identify patterns in data that deviate from the norm and flag them as potential anomalies.**
* **Recommender Systems: LSTMs have been used for recommendation tasks such as recommending movies, music, and books. They can learn patterns in user behavior and use them to make personalized recommendations.**
* **Video Analysis: LSTMs have been used for video analysis tasks such as object detection, activity recognition, and action classification. They can be used in combination with other neural network architectures, such as Convolutional Neural Networks (CNNs), to analyze video data and extract useful information.**

## Conclusion

**Long Short-Term Memory (LSTM) is a powerful type of recurrent neural network (RNN) that is well-suited for handling sequential data with long-term dependencies. It addresses the vanishing gradient problem, a common limitation of RNNs, by introducing a gating mechanism that controls the flow of information through the network. This allows LSTMs to learn and retain information from the past, making them effective for tasks like machine translation, speech recognition, and natural language processing.**

### ****1. What is LSTM?****

***LSTM is a type of recurrent neural network (RNN) that is designed to address the vanishing gradient problem, which is a common issue with RNNs. LSTMs have a special architecture that allows them to learn long-term dependencies in sequences of data, which makes them well-suited for tasks such as machine translation, speech recognition, and text generation.***

### ****2. How does LSTM work?****

***LSTMs use a cell state to store information about past inputs. This cell state is updated at each step of the network, and the network uses it to make predictions about the current input. The cell state is updated using a series of gates that control how much information is allowed to flow into and out of the cell.***

### ****3. What is the major difference between lstm and bidirectional lstm?****

***The vanishing gradient problem of the RNN is addressed by both LSTM and GRU, which differ in a few ways. These distinctions are as follows:***

* ***Bidirectional LSTM can utilize information from both past and future, whereas standard LSTM can only utilize past info.***
* ***Whereas GRU only employs two gates, LSTM uses three gates to compute the input of sequence data.***
* ***Compared to LSTM, GRUs are typically faster and simpler.***
* ***GRUs are favored for small datasets, while LSTMs are preferable for large datasets.***

### ****4. What is the difference between LSTM and Gated Recurrent Unit (GRU)?****

***LSTM has a cell state and gating mechanism which controls information flow, whereas GRU has a simpler single gate update mechanism. LSTM is more powerful but slower to train, while GRU is simpler and faster.***

### 5. What is difference between LSTM and RNN?

* ***RNNs have a simple recurrent structure with unidirectional information flow.***
* ***LSTMs have a gating mechanism that controls information flow and a cell state for long-term memory.***
* ***LSTMs generally outperform RNNs in tasks that require learning long-term dependencies.***

1. **ExplainGated recurrent unit (GRU)**

# Gated Recurrent Unit Networks

**Last Updated :**02 Mar, 2023

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks. Like LSTM, GRU can process sequential data such as text, speech, and time-series data.

The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step. The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate.

The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state. The output of the GRU is calculated based on the updated hidden state.

The equations used to calculate the reset gate, update gate, and hidden state of a GRU are as follows:

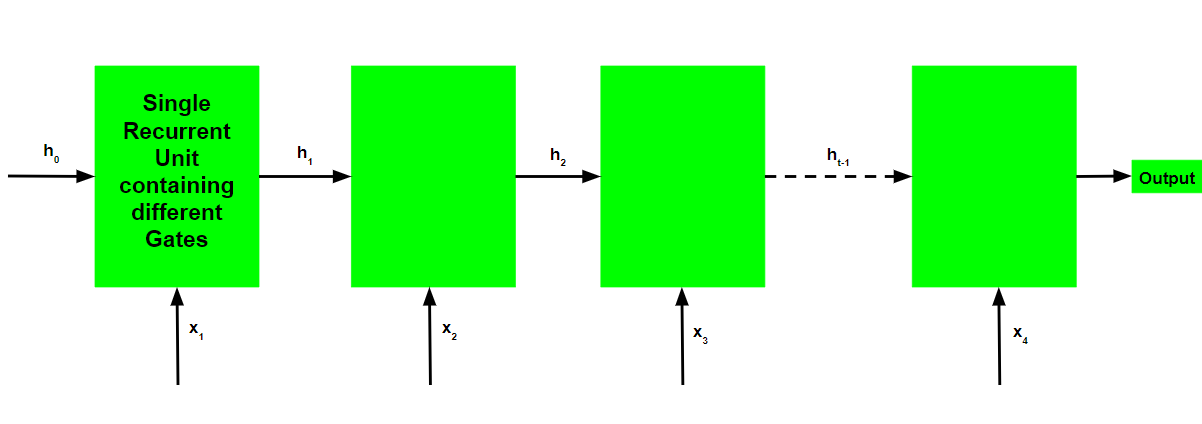
*Reset gate:****r\_t =******sigmoid(W\_r \* [h\_{t-1}, x\_t])*** *Update gate:****z\_t******=******sigmoid(W\_z \* [h\_{t-1}, x\_t])*** *Candidate hidden state:****h\_t’******= tanh(W\_h \* [r\_t \* h\_{t-1}, x\_t])*** *Hidden state:****h\_t = (1 – z\_t) \* h\_{t-1} + z\_t \* h\_t’*** *where W\_r, W\_z, and W\_h are learnable weight matrices, x\_t is the input at time step t, h\_{t-1} is the previous hidden state, and h\_t is the current hidden state.*

In summary, GRU networks are a type of RNN that use gating mechanisms to selectively update the hidden state at each time step, allowing them to effectively model sequential data. They have been shown to be effective in various natural language processing tasks, such as language modeling, machine translation, and speech recognition

**Prerequisites: Recurrent Neural Networks, Long Short Term Memory Networks**   
  
To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the **Long Short Term Memory Network(LSTM)**. One of the lesser-known but equally effective variations is the **Gated Recurrent Unit Network(GRU)**.   
  
Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit. The different gates of a GRU are as described below:-

1. **Update Gate(z):** It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
2. **Reset Gate(r):** It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.
3. **Current Memory Gate():** It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.



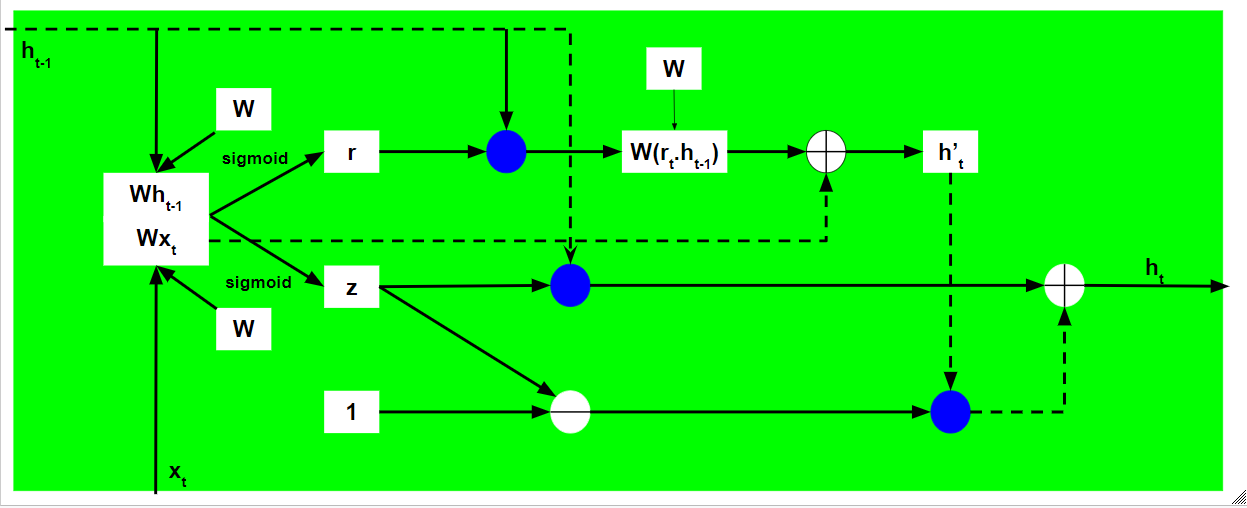
**Working of a Gated Recurrent Unit:**

* Take input the current input and the previous hidden state as vectors.
* Calculate the values of the three different gates by following the steps given below:-   
  1. For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
  2. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

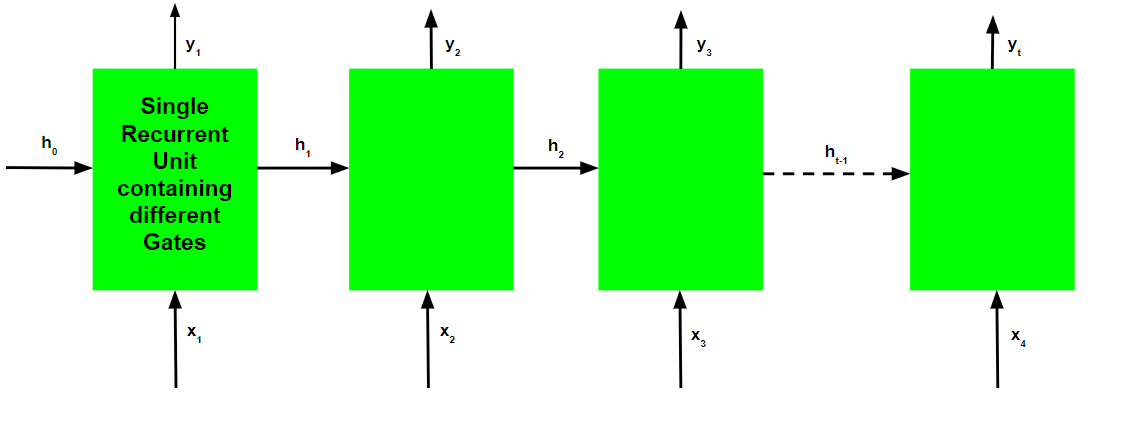
**Update Gate : Sigmoid Function**

**Reset Gate : Sigmoid Function**

* The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.
* To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.
* The above-stated working is stated as below:-



Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value). The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.   
  
Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.

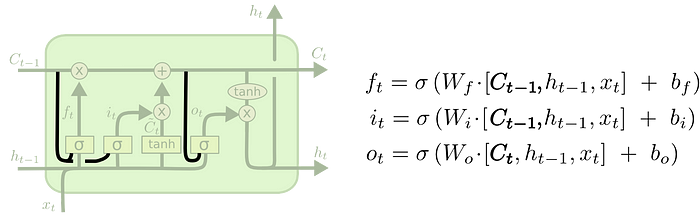


Note that just like the workflow, the training process for a GRU network is also diagrammatically similar to that of a basic Recurrent Neural Network and differs only in the internal working of each recurrent unit.   
  
The Back-Propagation Through Time Algorithm for a Gated Recurrent Unit Network is similar to that of a Long Short Term Memory Network and differs only in the differential chain formation.   
  
Let be the predicted output at each time step and be the actual output at each time step. Then the error at each time step is given by:-   
  
The total error is thus given by the summation of errors at all time steps.   
  
Similarly, the value can be calculated as the summation of the gradients at each time step.   
  
Using the chain rule and using the fact that is a function of and which indeed is a function of , the following expression arises:-   
  
Thus the total error gradient is given by the following:-   
  
Note that the gradient equation involves a chain of which looks similar to that of a basic Recurrent Neural Network but this equation works differently because of the internal workings of the derivatives of .   
  
**How do Gated Recurrent Units solve the problem of vanishing gradients?**   
  
The value of the gradients is controlled by the chain of derivatives starting from . Recall the expression for :-   
  
Using the above expression, the value for is:-   
Recall the expression for :-   
  
Using the above expression to calculate the value of :-   
  
Since both the update and reset gate use the sigmoid function as their activation function, both can take values either 0 or 1.   
  
**Case 1(z = 1):**   
In this case, irrespective of the value of , the term is equal to z which in turn is equal to 1.   
  
**Case 2A(z=0 and r=0):**   
  
In this case, the term is equal to 0.   
  
**Case 2B(z=0 and r=1):**   
  
In this case, the term is equal to . This value is controlled by the weight matrix which is trainable and thus the network learns to adjust the weights in such a way that the term comes closer to 1.   
  
Thus the Back-Propagation Through Time algorithm adjusts the respective weights in such a manner that the value of the chain of derivatives is as close to 1 as possible.

1. **Explain Peephole LSTM**

**Peephole Architecture**

Until now we have seen simple LSTM network but this architecture is modified along with time in each and every research paper. One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding “peephole connections.” This means that we let the gate layers look at the cell state.



In this peephole connection we can see that all the gates are having an input along with the cell state.

1. **Bidirectional RNNs**

# Bidirectional Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a particular class of neural networks that was created with the express purpose of processing sequential input, including speech, text, and time series data. RNNs process data as a sequence of vectors rather than feedforward neural networks, which process data as a fixed-length vector. Each vector is processed depending on the hidden state from the previous phase.

The network can store data from earlier steps in the sequence in a type of memory by computing the hidden state by taking into account both the current input and the hidden state from the previous phase. RNNs are thus well suited for jobs that call for knowledge of the context and connections among sequence elements.

Even though conventional RNNs can handle variable-length sequences, they sometimes have trouble with the vanishing gradient problem. Gradients during backpropagation become extremely small at this point, making it challenging for the network to learn from the data. Many [RNN](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/) versions, such [LSTMs](https://www.geeksforgeeks.org/understanding-of-lstm-networks/), and [GRUs](https://www.geeksforgeeks.org/gated-recurrent-unit-networks/), which use gating methods to regulate the flow of information and enhance learning, have been created to address this problem.

## ****Bi-directional Recurrent Neural Network****

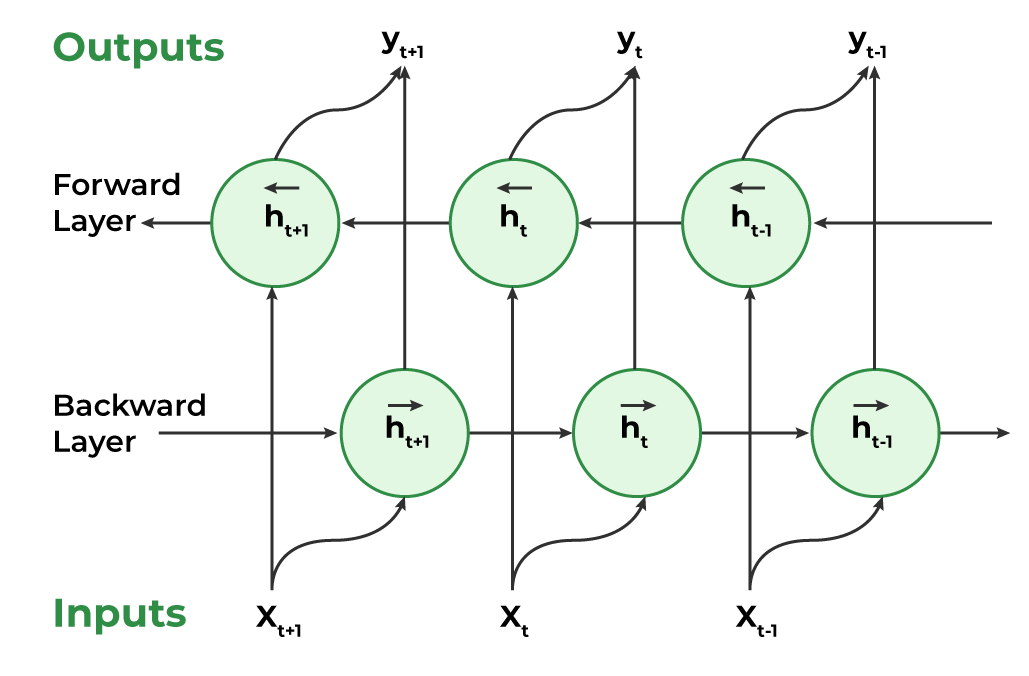
An architecture of a neural network called a bidirectional recurrent neural network (BRNN) is made to process sequential data. In order for the network to use information from both the past and future context in its predictions, BRNNs process input sequences in both the forward and backward directions. This is the main distinction between BRNNs and conventional recurrent neural networks.

A BRNN has two distinct recurrent hidden layers, one of which processes the input sequence forward and the other of which processes it backward. After that, the results from these hidden layers are collected and input into a prediction-making final layer. Any recurrent neural network cell, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit, can be used to create the recurrent hidden layers.

The BRNN functions similarly to conventional recurrent neural networks in the forward direction, updating the hidden state depending on the current input and the prior hidden state at each time step. The backward hidden layer, on the other hand, analyses the input sequence in the opposite manner, updating the hidden state based on the current input and the hidden state of the next time step.

Compared to conventional unidirectional recurrent neural networks, the accuracy of the BRNN is improved since it can process information in both directions and account for both past and future contexts. Because the two hidden layers can complement one another and give the final prediction layer more data, using two distinct hidden layers also offers a type of model regularisation.

In order to update the model parameters, the gradients are computed for both the forward and backward passes of the backpropagation through the time technique that is typically used to train BRNNs. The input sequence is processed by the BRNN in a single forward pass at inference time, and predictions are made based on the combined outputs of the two hidden layers. layers.

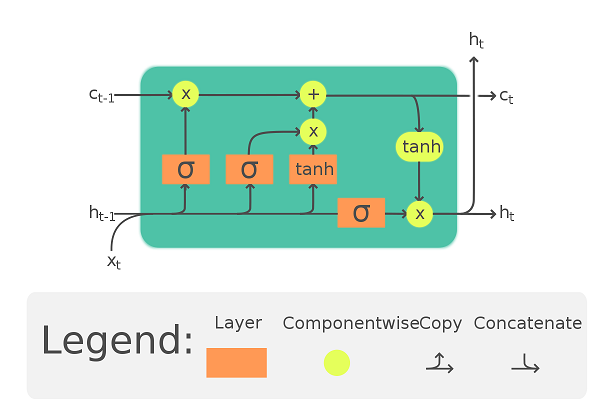


*Bi-directional Recurrent Neural Network*

### ****Working of Bidirectional Recurrent Neural Network****

1. **Inputting a sequence:** A sequence of data points, each represented as a vector with the same dimensionality, are fed into a BRNN. The sequence might have different lengths.
2. **Dual Processing:**  Both the forward and backward directions are used to process the data. On the basis of the input at that step and the hidden state at step t-1, the hidden state at time step t is determined in the forward direction. The input at step t and the hidden state at step t+1 are used to calculate the hidden state at step t in a reverse way.
3. **Computing the hidden state:** A non-linear activation function on the weighted sum of the input and previous hidden state is used to calculate the hidden state at each step. This creates a memory mechanism that enables the network to remember data from earlier steps in the process.
4. **Determining the output:** A non-linear activation function is used to determine the output at each step from the weighted sum of the hidden state and a number of output weights. This output has two options: it can be the final output or input for another layer in the network.
5. **Training:** The network is trained through a supervised learning approach where the goal is to minimize the discrepancy between the predicted output and the actual output. The network adjusts its weights in the input-to-hidden and hidden-to-output connections during training through backpropagation
6. **Explain the gates of LSTM with equations.**

It is special kind of **recurrent neural network(RNN)** that is capable of learning long term dependencies in data. This is achieved because the recurring module of the model has a combination of four layers interacting with each other.



There are three different gates in an LSTM cell: a forget gate, an input gate, and an output gate.

**Forget Gate :**

The forget gate decides which information needs attention and which can be ignored. The information from the current input X(t) and hidden state h(t-1) are passed through the sigmoid function. Sigmoid generates values between 0 and 1. It concludes whether the part of the old output is necessary (by giving the output closer to 1). This value of f(t) will later be used by the cell for point-by-point multiplication.

**Input Gate :**

The input gate performs the following operations to update the cell status. First, the current state X(t) and previously hidden state h(t-1) are passed into the second sigmoid function. The values are transformed between 0 (important) and 1 (not-important). Next, the same information of the hidden state and current state will be passed through the tanh function. To regulate the network, the tanh operator will create a vector (C~(t) ) with all the possible values between -1 and 1. The output values generated form the activation functions are ready for point-by-point multiplication.

**Output Gate :**

The output gate determines the value of the next hidden state. This state contains information on previous inputs. First, the values of the current state and previous hidden state are passed into the third sigmoid function. Then the new cell state generated from the cell state is passed through the tanh function. Both these outputs are multiplied point-by-point. Based upon the final value, the network decides which information the hidden state should carry. This hidden state is used for prediction.

1. **Explain BiLSTM**

# Bidirectional LSTM in NLP

In this article, we will first discuss bidirectional LSTMs and their architecture. We will then look into the implementation of a review system using Bidirectional LSTM. Finally, we will conclude this article while discussing the applications of bidirectional LSTM.

## Bidirectional LSTM (BiLSTM)

Bidirectional LSTM or BiLSTM is a term used for a sequence model which contains two [LSTM](https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/) layers, one for processing input in the forward direction and the other for processing in the backward direction. It is usually used in NLP-related tasks. The intuition behind this approach is that by processing data in both directions, the model is able to better understand the relationship between sequences (e.g. knowing the following and preceding words in a sentence).

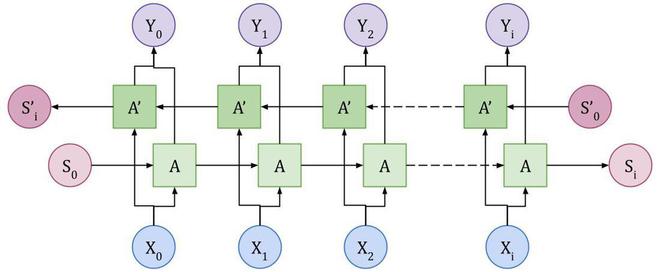
To better understand this let us see an example. The first statement is “Server can you bring me this dish” and the second statement is “He crashed the server”. In both these statements, the word server has different meanings and this relationship depends on the following and preceding words in the statement. The bidirectional LSTM helps the machine to understand this relationship better than compared with unidirectional LSTM. This ability of BiLSTM makes it a suitable architecture for tasks like [sentiment analysis](https://www.geeksforgeeks.org/what-is-sentiment-analysis/), [text classification](https://www.geeksforgeeks.org/text-mining-in-data-mining/), and [machine translation](https://www.geeksforgeeks.org/machine-translation-of-languages-in-artificial-intelligence/).

### Architecture

The architecture of bidirectional LSTM comprises of two unidirectional LSTMs which process the sequence in both forward and backward directions. This architecture can be interpreted as having two separate LSTM networks, one gets the sequence of tokens as it is while the other gets in the reverse order. Both of these LSTM network returns a probability vector as output and the final output is the combination of both of these probabilities. It can be represented as:

where,

* : Final probability vector of the network.
* : Probability vector from the forward LSTM network.
* : Probability vector from the backward LSTM network.



*Bidirectional LSTM layer Architecture*

Figure describes the architecture of the BiLSTM layer where is the input token, is the output token, and and are LSTM nodes. The final output of is the combination of and LSTM nodes.

Now, let us look into an implementation of a review system using BiLSTM layers in Python using the Tensorflow library. We would be performing sentiment analysis on the IMDB movie review dataset. We would implement the network from scratch and train it to identify if the review is positive or negative.

## Importing Libraries and Dataset

[**Python**](https://www.geeksforgeeks.org/python-programming-language/) libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

* [**Numpy**](https://www.geeksforgeeks.org/python-numpy/)– Numpy arrays are very fast and can perform large computations in a very short time.
* [**Matplotlib**](https://www.geeksforgeeks.org/matplotlib-tutorial/)– This library is used to draw visualizations.
* [**TensorFlow**](https://www.geeksforgeeks.org/introduction-to-tensorflow/) – This is an open-source library that is used for [Machine Learning](https://www.geeksforgeeks.org/machine-learning/) and Artificial intelligence and provides a range of functions to achieve complex functionalities with single lines of code.

1. **Explain BiGRU**

**What is a Bidirectional GRU?**

Before diving into the specifics of BiGRUs, let's take a closer look at GRUs. A GRU is a type of recurrent neural network, which means it can store information from previous inputs to make more informed predictions about future inputs. This is particularly useful for sequence processing tasks because the order of the inputs matters.

GRUs have a unique architecture that includes both reset and update gates. The reset gate determines which information from the previous input should be forgotten, and the update gate determines which parts of the current input should be stored in memory. By adjusting these gates during training, the GRU can learn to recognize patterns and make more accurate predictions.

A BiGRU, on the other hand, takes the input in both a forward and a backwards direction. This means that it actually contains two GRUs - one that processes the input in its original order, and another that processes it in reverse. By doing this, the BiGRU is able to efficiently capture contextual information from both preceding and succeeding inputs. This makes it a powerful tool for tasks such as named entity recognition, sentiment analysis, and machine translation.

**Applications of BiGRU**

BiGRUs have a wide range of applications due to their ability to process sequences efficiently and effectively. They are particularly useful for tasks that involve language or speech, such as:

* **Sentiment Analysis**: BiGRUs can be used to analyze text and determine the overall sentiment expressed.
* **Natural Language Understanding**: By analyzing text in context, BiGRUs can improve the accuracy of language understanding models.
* **Speech Recognition**: BiGRUs can help to improve speech recognition accuracy by modeling the relationship between phonemes and graphemes.
* **Machine Translation**: By processing text in both directions, BiGRUs can help to improve the quality of machine translated text.

**The Benefits of BiGRU**

So what makes BiGRUs so effective, and why are they so popular in the field of machine learning? Here are a few key benefits:

* **Efficiency**: BiGRUs are able to process inputs more efficiently than traditional recurrent neural networks, thanks to their ability to analyze the input from both directions simultaneously.
* **Contextual Awareness**: By analyzing the input in both directions, BiGRUs are able to better capture the context of each input and make more accurate predictions.
* **Accuracy**: BiGRUs are able to produce more accurate results than traditional recurrent neural networks, making them a popular choice for a wide range of sequence processing tasks.