Unit 4. Recurrent Neural Networks (RNN)

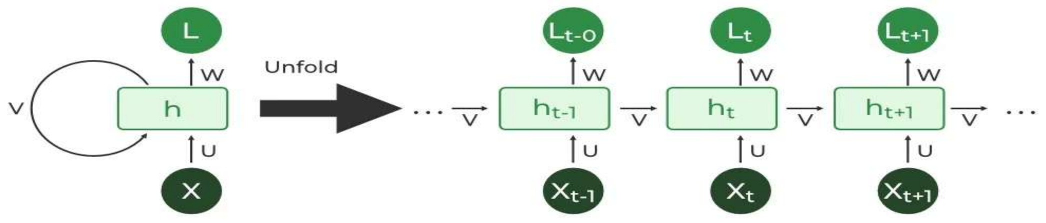
Recurrent Neural Network (RNN), Why recurrent networks? RNN explained, Deep RNNs, Recursive neural networks, Step function, Tanh function, RNN in memory, LSTMs and GRUs, Long Short Term Memory (LSTM), Working components of LSTMs, Core idea behind LSTMs, LSTM: A simple walk through, Gated Recurrent Unit (GRU), GRU design steps, Fully gated vs minimal gated architecture of GRU, Working of RNN's, Recurrent neural networks, Backpropagation through timeline in RNN, Backpropagation through Computational graphs, Problem Statement 1, Problem Statement 2, Complex recurrent neural networks, Over-fitting and under-fitting, Detect and avoid overfitting, Prevent of overfitting an approach on model and data, Multi-layered RNNs, Stacked LSTM, Stacked LSTM architecture, Multi-directional RNNs, Difference between LSTM and BI-LSTM, One-dimensional sequence processing, CNN and RNN.

4.1 Recurrent Neural Network (RNN)

What is Recurrent Neural Network (RNN)?

Recurrent Neural Network(RNN) is a type of 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, but 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.



REcurrent neural network

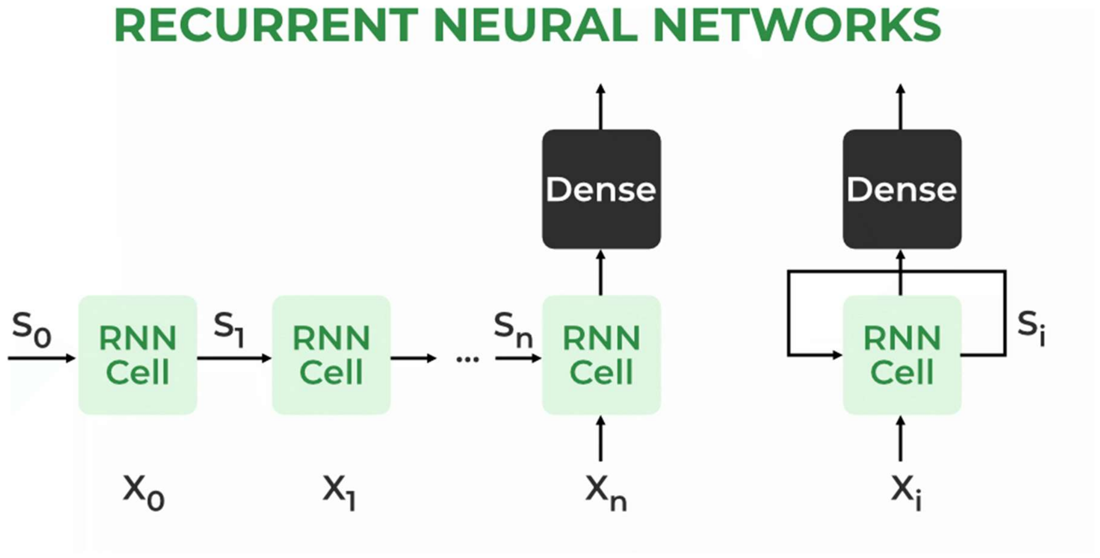
4.1.1 Architecture Of Recurrent Neural Network

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 X_i . **By using the following formulas:**

$h= \sigma(UX + Wh_{-1} + B)$

$Y = O(Vh + C)$ 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



What is Recurrent Neural Network

How RNN works

The Recurrent Neural Network consists of multiple fixed activation function 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:

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

where:
h_t -> current state
h_{t-1} -> previous state
x_t -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh (W_{hh}h_{t-1}+ W_{xh}x_t)$$

where:
w_{hh} -> weight at recurrent neuron
w_{xh} -> weight at input neuron

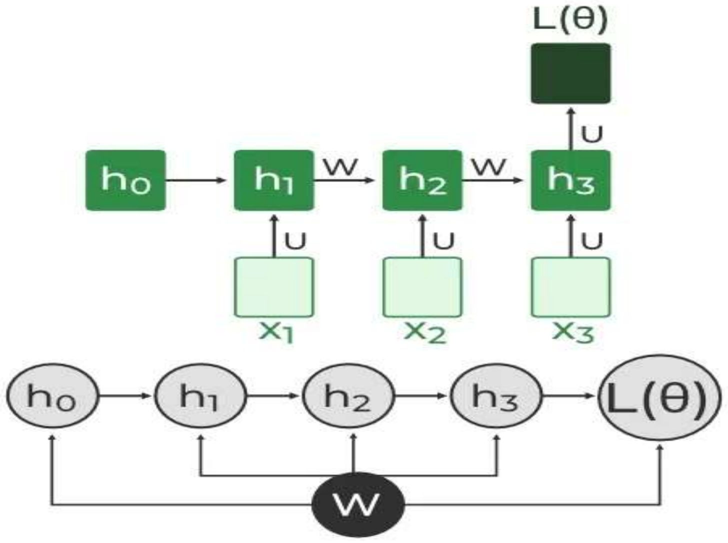
The formula for calculating output:

$$y_t = W_{hy}h_t$$

Y_t -> output
W_{hy} -> weight at output layer
These parameters are updated using Backpropagation. 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.

$$\frac{\partial \mathbf{L}(\theta)}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathbf{L}(\theta)}{\partial W}$$

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

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

We already know how to compute this one as it is the same as any simple deep neural network backpropagation. $\frac{\partial L(\theta)}{\partial h_3}$ .However, we will see how to apply backpropagation to this term $\frac{\partial h_3}{\partial W}$

```
                    As we know h3 = σ(Wh2 + b)
```

And In such an ordered network, we can't compute $\frac{\partial h_3}{\partial W}$ by simply treating h3 as a constant because as it also depends on W. the total derivative $\frac{\partial h_3}{\partial W}$ has two parts

1. **Explicit:** $\frac{\partial h_3 +}{\partial W}$ treating all other inputs as constant
2. **Implicit:** Summing over all indirect paths from h_3 to W

Let us see how to do this

$$\begin{aligned}\frac{\partial h_3}{\partial W} &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} \\ &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\left[\frac{\partial h_2^+}{\partial W} + \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W}\right] \\ &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\left[\frac{\partial h_1^+}{\partial W}\right]\end{aligned}$$

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

$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_1}\frac{\partial h_1^+}{\partial W}$$

Finally, we have

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \cdot \frac{\partial h_3}{\partial W}$$

Where

$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

Hence,

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

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

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.

The advantages of Recurrent Neural Networks (RNNs) are:

1. Ability to Process Sequential Data: RNNs can process sequential data of varying lengths, making them useful in applications such as natural language processing, speech recognition, and time-series analysis.
2. Memory: RNNs have the ability to retain information about the previous inputs in the sequence through the use of hidden states. This enables RNNs to perform tasks such as predicting the next word in a sentence or forecasting stock prices.
3. Versatility: RNNs can be used for a wide variety of tasks, including classification, regression, and sequence-to-sequence mapping.
4. Flexibility: RNNs can be combined with other neural network architectures, such as Convolutional Neural Networks (CNNs) or feedforward neural networks, to create hybrid models for specific tasks.

However, there are also some disadvantages of RNNs:

1. Vanishing Gradient Problem: The vanishing gradient problem can occur in RNNs, particularly in those with many layers or long sequences, making it difficult to learn long-term dependencies.
2. Computationally Expensive: RNNs can be computationally expensive, particularly when processing long sequences or using complex architectures.
3. Lack of Interpretability: RNNs can be difficult to interpret, particularly in terms of understanding how the network is making predictions or decisions.
4. Overall, while RNNs have some disadvantages, their ability to process sequential data and retain memory of previous inputs make them a powerful tool for many machine learning applications.

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

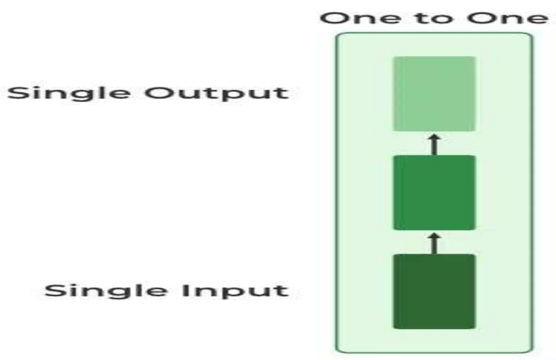
Types Of RNN

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

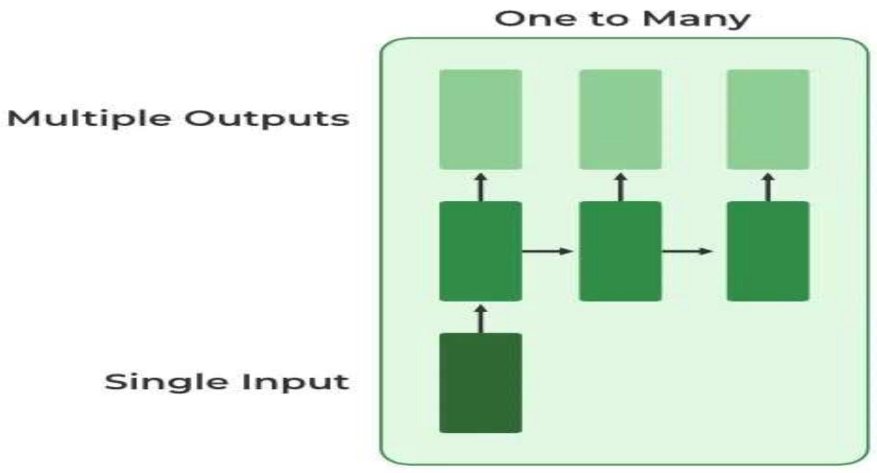
This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.



One to One RNN

One To Many

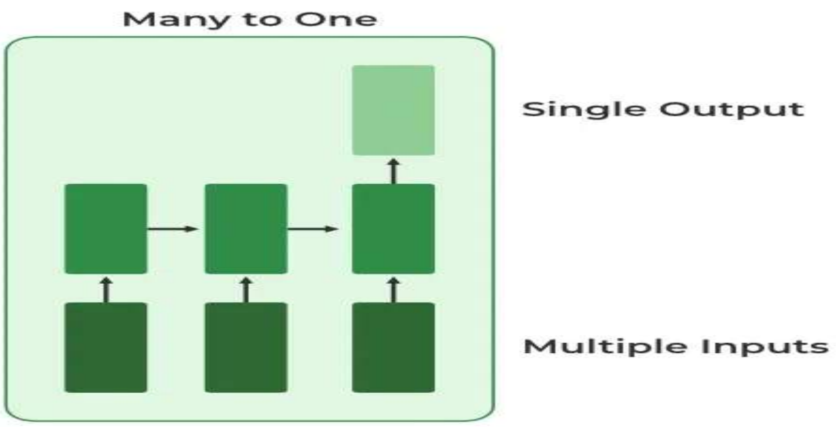
In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.



One To Many RNN

Many to One

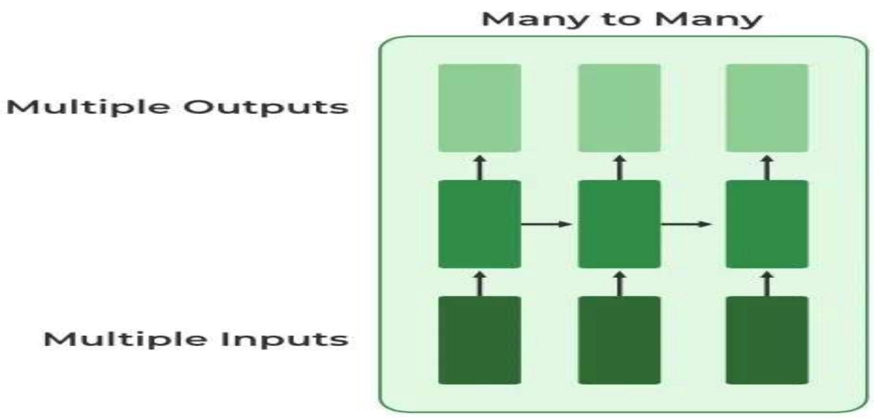
In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.



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

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

4.2 Long Short Term Memory Networks (LSTM)

Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for tasks such as language translation, speech recognition, and time series forecasting.

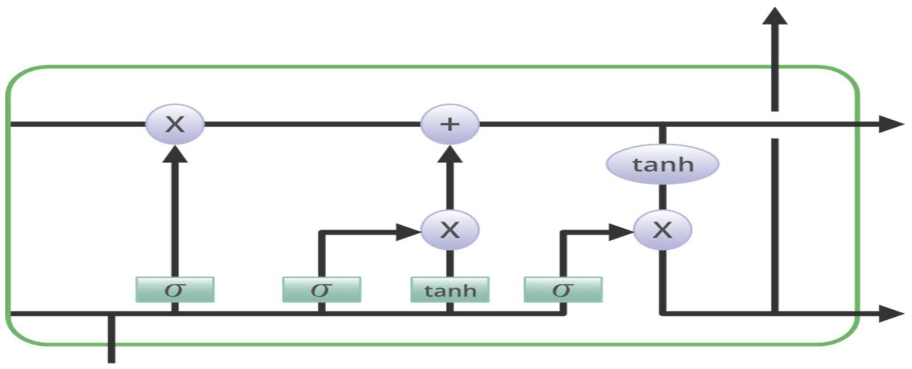
A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period of time. The memory cell 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.

LSTMs can be stacked to create deep LSTM networks, which can learn even more complex patterns in sequential data. LSTMs can also be used in combination with other neural network architectures, such as Convolutional Neural Networks (CNNs) for image and video analysis.

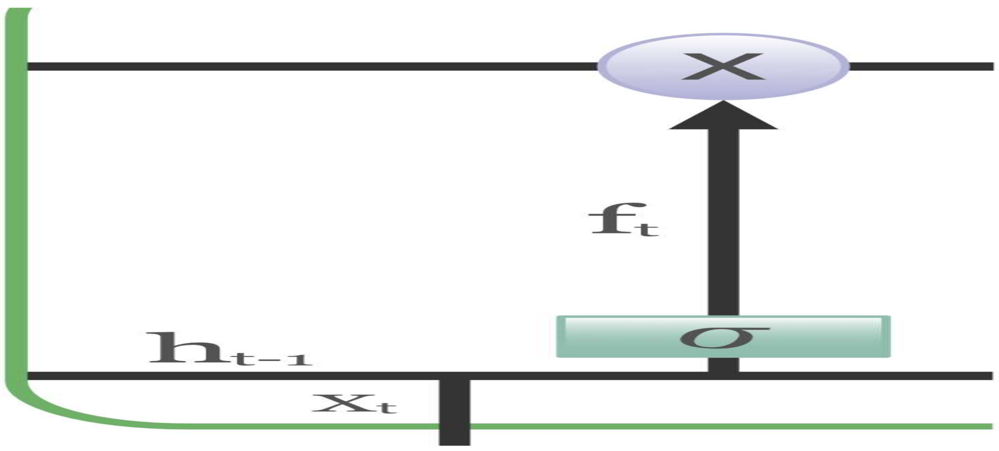
Structure of LSTM:

LSTM 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 –
1. Forget Gate: The information that is no longer useful in the cell state is removed with the forget gate. Two inputs *x_t* (input at the particular time) and *h_t-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:
f_t = σ(W_f · [h_t-1, x_t] + b_f)
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.



2. 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 *h_t-1* and *x_t*. Then, a vector is created using *tanh* function that gives an output from -1 to +1, which contains all the possible values from h_t-1 and *x_t*. 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:

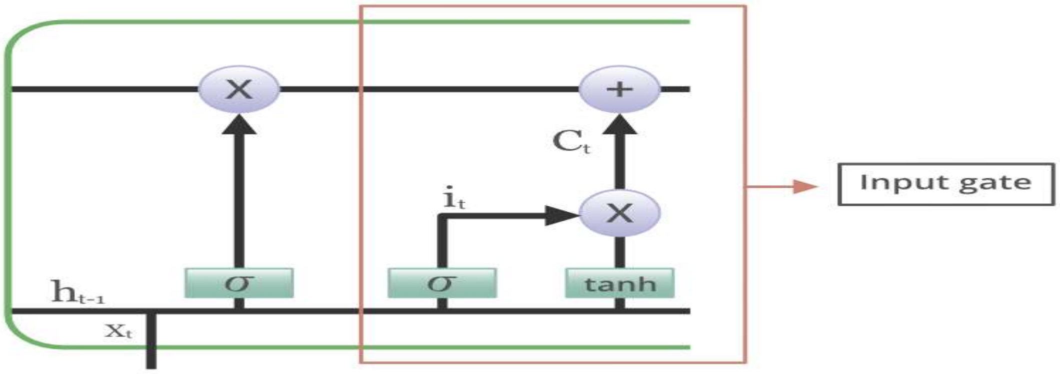
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$

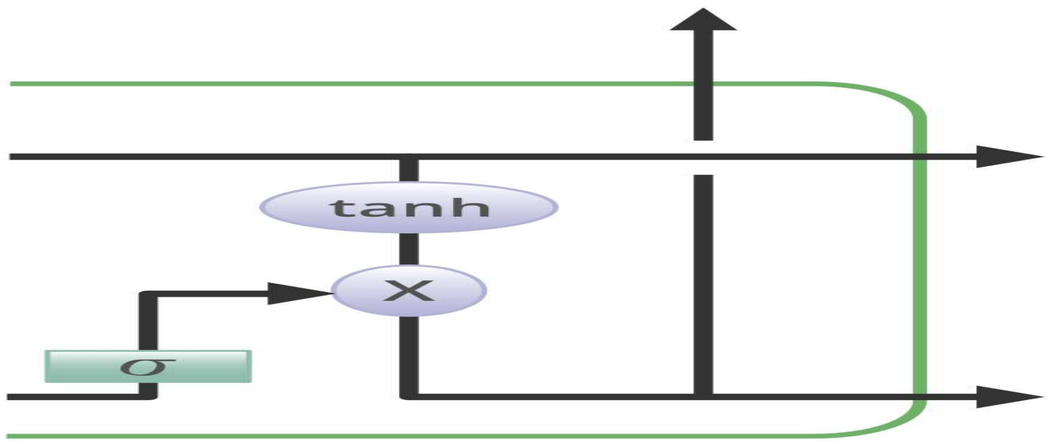
where

- $\odot$ denotes element-wise multiplication
- tanh is tanh activation function

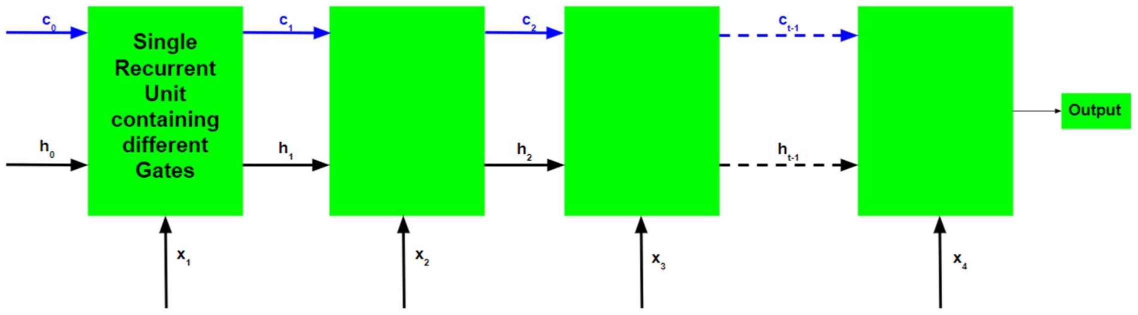


3. Output gate: The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs *h_t-1* and *x_t*. 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:

$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$



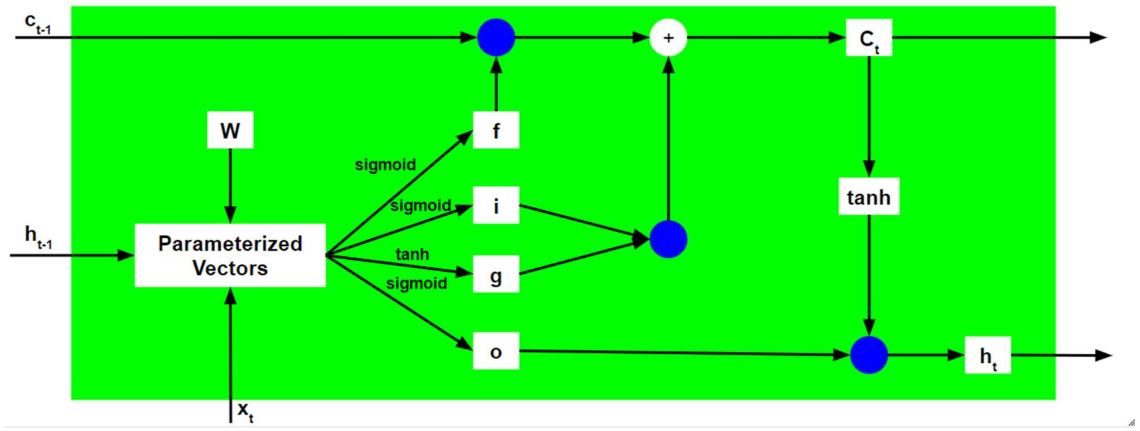
The basic workflow of a Long Short Term Memory Network is similar to the workflow of a Recurrent Neural Network with the only difference being that the Internal Cell State is also passed forward along with the Hidden State.



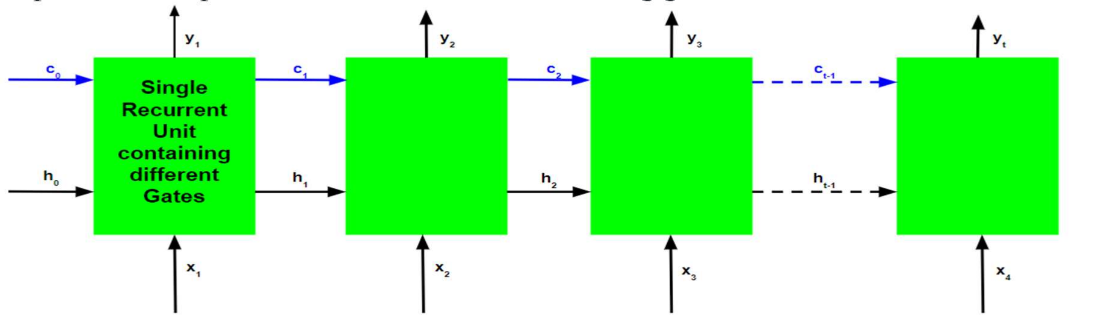
Working of an LSTM recurrent unit:

1. Take input the current input, the previous hidden state, and the previous internal cell state.
2. Calculate the values of the four different gates by following the below steps:-
   - For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.
   - 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.
3. Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then add the two vectors.
   $c_t = i \odot g + f \odot c_{t-1}$
4. Calculate the current hidden state by first taking the element-wise hyperbolic tangent of the current internal cell state vector and then performing element-wise multiplication with the output gate.

The above-stated working is illustrated as below:-



Note that the blue circles denote element-wise multiplication. 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, an LSTM network also generates an output at each time step and this output is used to train the network using gradient descent.



The only main difference between the Back-Propagation algorithms of Recurrent Neural Networks and Long Short Term Memory Networks is related to the mathematics of the algorithm.

Let $\overline{y}_t$be the predicted output at each time step and $y_t$be the actual output at each time step. Then the error at each time step is given by:-

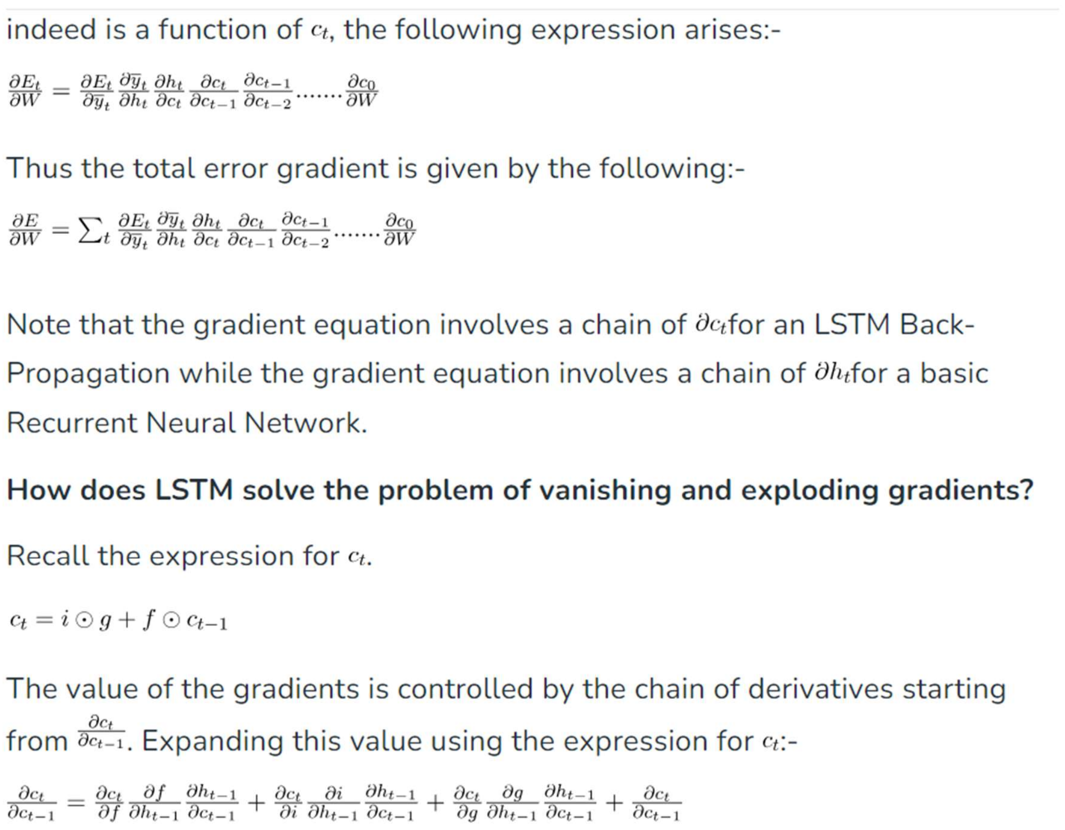
$$E_t = -y_t log(\overline{y}_t)$$

The total error is thus given by the summation of errors at all time steps.

$$E = \sum_t E_t$$
$$\Rightarrow E = \sum_t -y_t log(\overline{y}_t)$$

Similarly, the value $\frac{\partial E}{\partial W}$can be calculated as the summation of the gradients at each time step.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$



indeed is a function of c_t, the following expression arises:-

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \overline{y}_t}\frac{\partial \overline{y}_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}}\frac{\partial c_{t-1}}{\partial c_{t-2}}\cdots\cdots\frac{\partial c_0}{\partial W}$$

Thus the total error gradient is given by the following:-

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial \overline{y}_t}\frac{\partial \overline{y}_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}}\frac{\partial c_{t-1}}{\partial c_{t-2}}\cdots\cdots\frac{\partial c_0}{\partial W}$$

Note that the gradient equation involves a chain of $\partial c_t$for an LSTM Back-Propagation while the gradient equation involves a chain of $\partial h_t$for a basic Recurrent Neural Network.

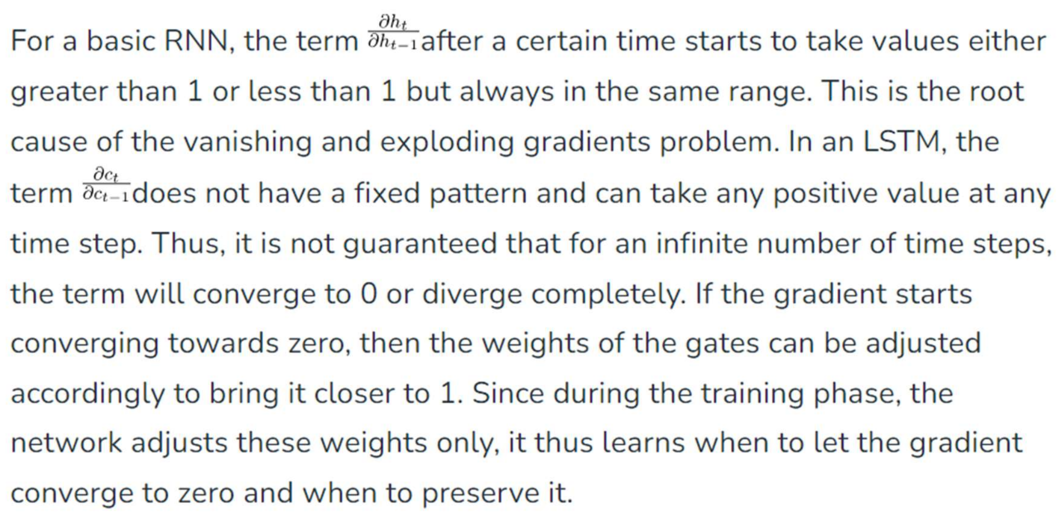
How does LSTM solve the problem of vanishing and exploding gradients?

Recall the expression for c_t.

$$c_t = i \odot g + f \odot c_{t-1}$$

The value of the gradients is controlled by the chain of derivatives starting from $\frac{\partial c_t}{\partial c_{t-1}}$. Expanding this value using the expression for c_t:-

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial c_t}{\partial f}\frac{\partial f}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial c_{t-1}} + \frac{\partial c_t}{\partial i}\frac{\partial i}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial c_{t-1}} + \frac{\partial c_t}{\partial g}\frac{\partial g}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial c_{t-1}} + \frac{\partial c_t}{\partial c_{t-1}}$$



For a basic RNN, the term $\frac{\partial h_t}{\partial h_{t-1}}$after a certain time starts to take values either greater than 1 or less than 1 but always in the same range. This is the root cause of the vanishing and exploding gradients problem. In an LSTM, the term $\frac{\partial c_t}{\partial c_{t-1}}$does not have a fixed pattern and can take any positive value at any time step. Thus, it is not guaranteed that for an infinite number of time steps, the term will converge to 0 or diverge completely. If the gradient starts converging towards zero, then the weights of the gates can be adjusted accordingly to bring it closer to 1. Since during the training phase, the network adjusts these weights only, it thus learns when to let the gradient converge to zero and when to preserve it.

Advantages of LSTM

1. Long-term dependencies can be captured by LSTM networks. They have a memory cell that is capable of long-term information storage.
2. 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.
3. 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.

Disadvantages of LSTM

1. 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.
2. 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.
3. Since it is processed word by word in a sequential manner, it is hard to parallelize the work of processing the sentences.

Some of the famous applications of LSTM includes:

1. Long Short-Term Memory (LSTM) is a powerful type of Recurrent Neural Network (RNN) that has been used in a wide range of applications. Here are a few famous applications of LSTM:
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.

4.3 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over long short term memory (LSTM). GRU uses less memory and is faster than LSTM, however, LSTM is more accurate when using datasets with longer sequences.

Also, GRUs address the vanishing gradient problem (values used to update network weights) from which vanilla recurrent neural networks suffer. If the grading shrinks over time as it back propagates, it may become too small to affect learning, thus making the neural net untrainable.

If layer in a neural net can't learn, RNN's can essentially "forget" longer sequences.

GRUs solve this problem through the use of two gates, the update gate and reset gate. These gates decide what information is allowed through to the output and can be trained to retain information from farther back. This allows it to pass relevant information down a chain of events to make better predictions.

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.

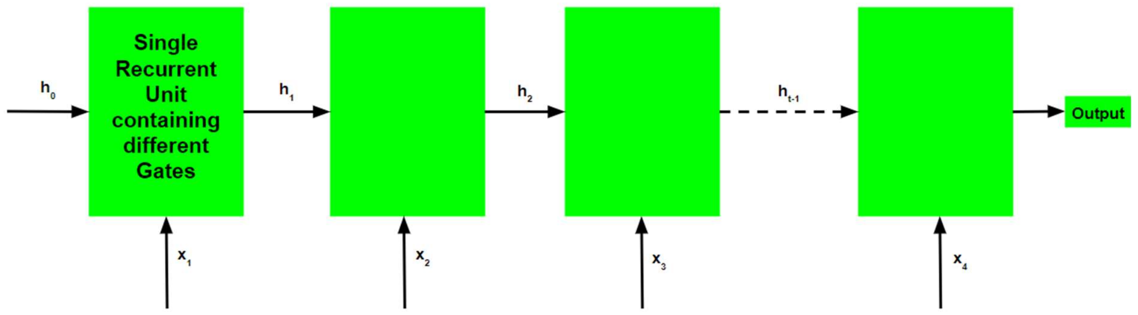
The equations used to calculate the reset gate, update gate, and hidden state of a GRU are as follows:

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
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($\bar{h}_t$):** 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.

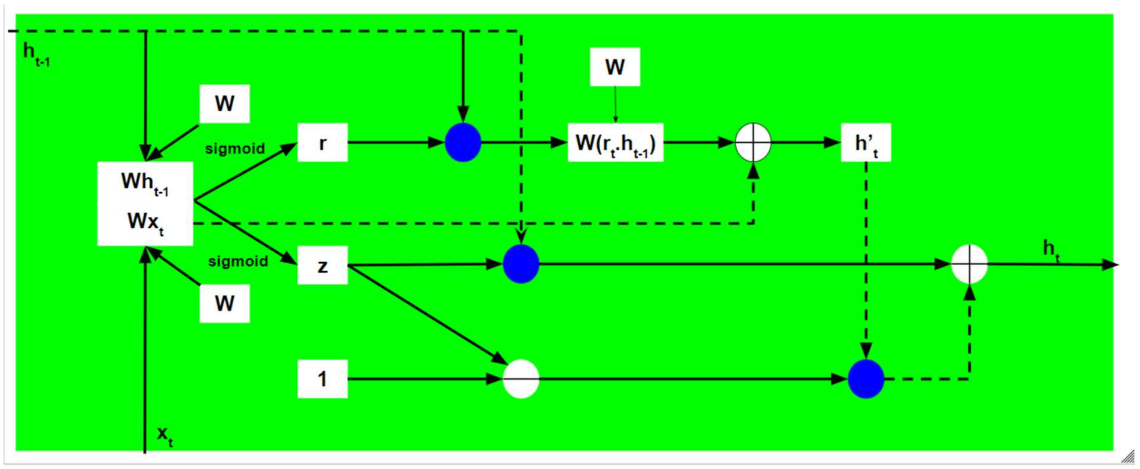
   The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and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



Working of a Gated Recurrent Unit:

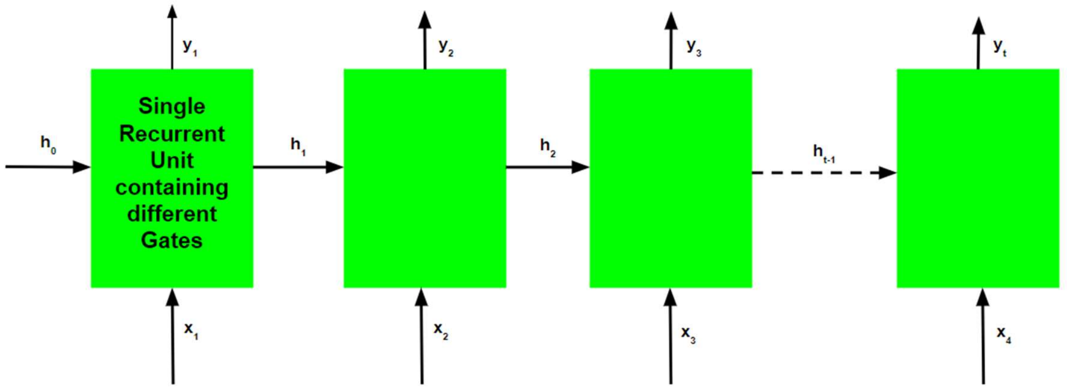
- Take input the current input and the previous hidden state as vectors.
- Calculate the values of the three different gates by following the steps given below:-
    1. For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (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.
  $$\overline{h}_t = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$$
- 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.
  $$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$$
  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:-

$$E_t = -y_t log(\overline{y}_t)$$

The total error is thus given by the summation of errors at all time steps.

$$E = \sum_t E_t$$

$$\Rightarrow E = \sum_t -y_t log(\overline{y}_t)$$

Similarly, the value $\frac{\partial E}{\partial W}$can be calculated as the summation of the gradients at each time step.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

Using the chain rule and using the fact that $\overline{y}_t$is a function of $h_t$and which indeed is a function of $\overline{h}_t$, the following expression arises:-

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \overline{y}_t}\frac{\partial \overline{y}_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}......\frac{\partial h_0}{\partial W}$$

Thus the total error gradient is given by the following:-

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial \overline{y}_t}\frac{\partial \overline{y}_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}......\frac{\partial h_0}{\partial W}$$

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 $\frac{\partial h_t}{\partial h_{t-1}}$. Recall the expression for h_t:-

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$$

Using the above expression, the value for $\frac{\partial h_t}{\partial h_{t-1}}$is:-

$$\frac{\partial h_t}{\partial h_{t-1}} = z + (1 - z)\frac{\partial \overline{h}_t}{\partial h_{t-1}}$$

Recall the expression for $\overline{h}_t$:-

$$\overline{h}_t = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$$

Using the above expression to calculate the value of $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$:-

$$\frac{\partial \overline{h_t}}{\partial h_{t-1}} = \frac{\partial (tanh(W \odot x_t + W \odot (r_t \odot h_{t-1})))}{\partial h_{t-1}} \Rightarrow \frac{\partial \overline{h_t}}{\partial h_{t-1}} = (1 - \overline{h}_t^2)(W \odot r)$$

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 r, the term $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$is equal to z which in turn is equal to 1.

Case 2A(z=0 and r=0):

In this case, the term $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$is equal to 0.

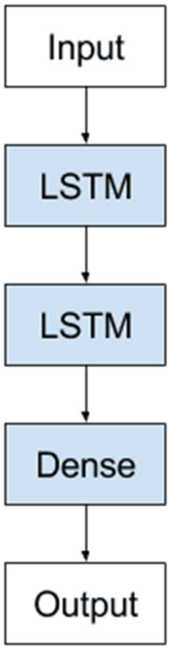
Case 2B(z=0 and r=1):

In this case, the term $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$is equal to $(1 - \overline{h}_t^2)(W)$. 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 $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$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.

4.4 Stacked LSTM :

A Stacked LSTM architecture can be defined as an **LSTM model comprised of multiple LSTM layers**. An LSTM layer above provides a sequence output rather than a single value output to the LSTM layer below. Specifically, one output per input time step, rather than one output time step for all input time steps.



4.5 Overfitting and Underfitting

Overfitting and Underfitting are the two main problems that occur in machine learning and degrade the performance of the machine learning models.The main goal of each machine learning model is **to generalize well**. Here **generalization** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input. It means after providing training on the dataset, it can produce reliable and accurate output. Hence, the underfitting and overfitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.
Before understanding the overfitting and underfitting, let's understand some basic term that will help to understand this topic well:

- **Signal:** It refers to the true underlying pattern of the data that helps the machine learning model to learn from the data.
- **Noise:** Noise is unnecessary and irrelevant data that reduces the performance of the model.
- **Bias:** Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.

- **Variance:** If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

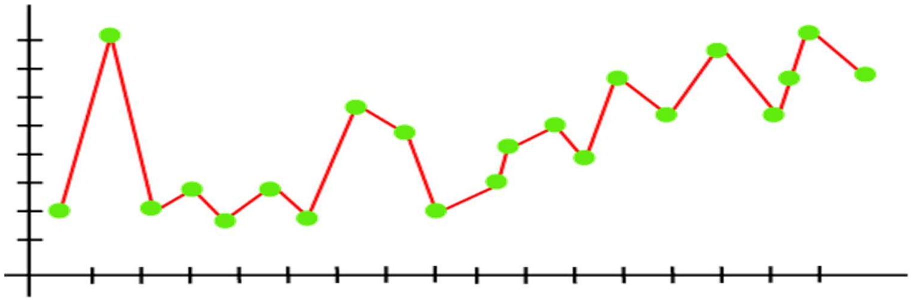
4.5.1 Overfitting

Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model. The overfitted model has **low bias** and **high variance.**

The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model.

Overfitting is the main problem that occurs in supervised learning.

Example: The concept of the overfitting can be understood by the below graph of the linear regression output:



As we can see from the above graph, the model tries to cover all the data points present in the scatter plot. It may look efficient, but in reality, it is not so. Because the goal of the regression model to find the best fit line, but here we have not got any best fit, so, it will generate the prediction errors.

4.5.2 How to detect overfit models

To understand the accuracy of machine learning models, it's important to test for model fitness. K-fold cross-validation is one of the most popular techniques to assess accuracy of the model.

In k-folds cross-validation, data is split into k equally sized subsets, which are also called "folds." One of the k-folds will act as the test set, also known as the holdout set or validation set, and the remaining folds will train the model. This process repeats until each of the fold has acted as a holdout fold. After each evaluation, a score is retained and when all iterations have completed, the scores are averaged to assess the performance of the overall model.

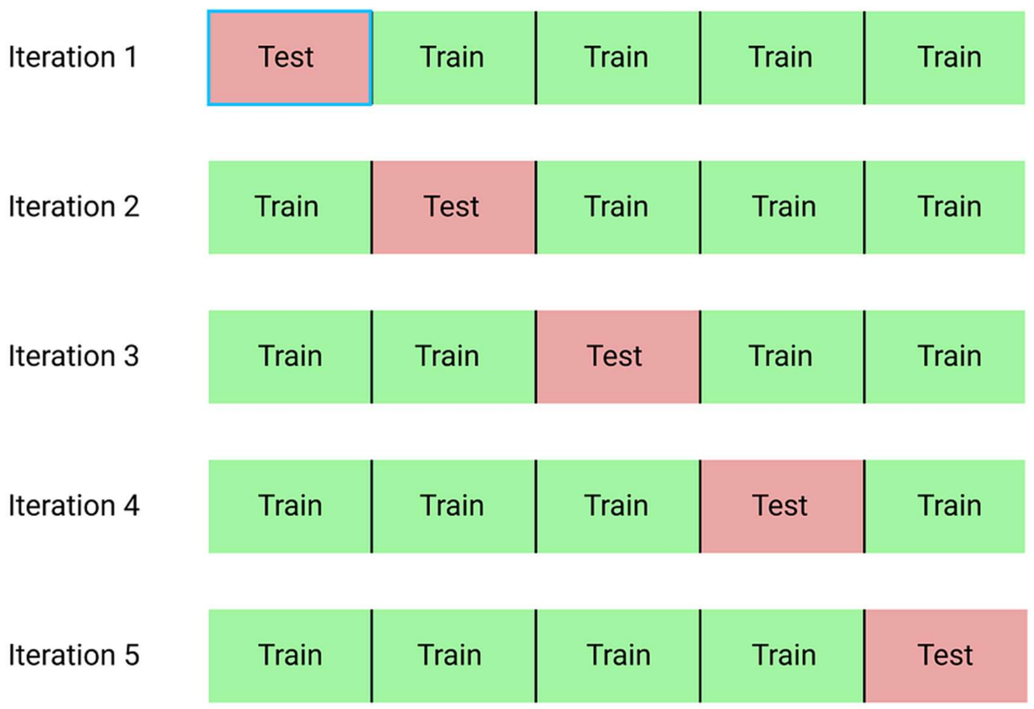
4.5.3 How to avoid the Overfitting in Model

Both overfitting and underfitting cause the degraded performance of the machine learning model. But the main cause is overfitting, so there are some ways by which we can reduce the occurrence of overfitting in our model.

- **Cross-Validation**
- **Training with more data**
- **Removing features**
- **Early stopping the training**
- **Regularization**
- **Ensembling**

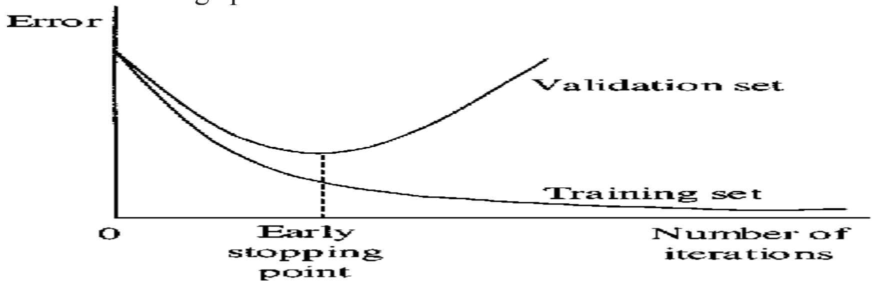
1. **Cross Validation**

   We can split our dataset into *k* groups (k-fold cross-validation). We let one of the groups to be the testing set (please see hold-out explanation) and the others as the training set, and repeat this process until each individual group has been used as the testing set (e.g., *k* repeats). Unlike hold-out, cross-validation allows all data to be eventually used for training but is also more computationally expensive than hold-out.



2. **Early stopping:**

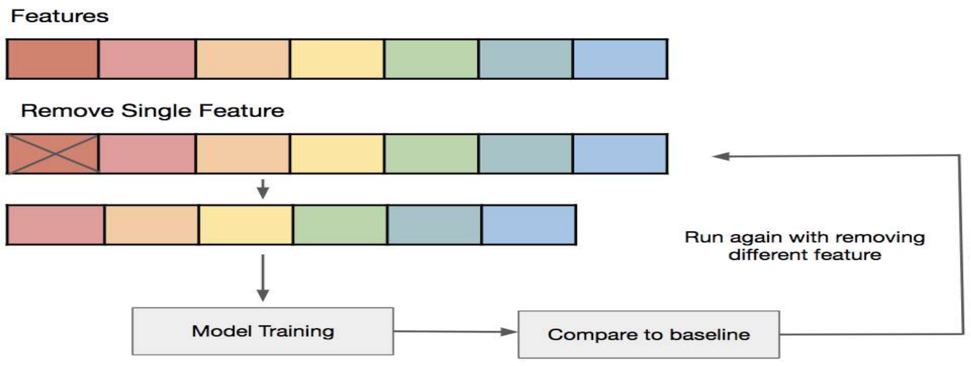
   We can first train our model for an arbitrarily large number of epochs and plot the validation loss graph (e.g., using hold-out). Once the validation loss begins to degrade (e.g., stops decreasing but rather begins increasing), we stop the training and save the current model. We can implement this either by monitoring the loss graph or set an early stopping trigger. The saved model would be the optimal model for generalization among different training epoch values.



3. **Train with more data:**

   Expanding the training set to include more data can increase the accuracy of the model by providing more opportunities to parse out the dominant relationship among the input and output variables. That said, this is a more effective method when clean, relevant data is injected into the model. Otherwise, you could just continue to add more complexity to the model, causing it to overfit.

4. **Data augmentation:**
   A larger dataset would reduce overfitting. If we cannot gather more data and are constrained to the data we have in our current dataset, we can apply data augmentation to artificially increase the size of our dataset. For example, if we are training for an image classification task, we can perform various image transformations to our image dataset (e.g., flipping, rotating, rescaling, shifting).
5. **Feature selection:**
   When you build a model, you'll have a number of parameters or features that are used to predict a given outcome, but many times, these features can be redundant to others. Feature selection is the process of identifying the most important ones within the training data and then eliminating the irrelevant or redundant ones. This is commonly mistaken for dimensionality reduction, but it is different. However, both methods help to simplify your model to establish the dominant trend in the data. We can simply test out different features, train individual models for these features, and evaluate generalization capabilities, or use one of the various widely used feature selection methods.



6. **Regularization:**
   If overfitting occurs when a model is too complex, it makes sense for us to reduce the number of features. But what if we don't know which inputs to eliminate during the feature selection process? If we don't know which features to remove from our model, regularization methods can be particularly helpful. Regularization applies a "penalty" to the input parameters with the larger coefficients, which subsequently limits the amount of variance in the model. While there are a number of regularization methods, such as L1 regularization, Lasso regularization, and dropout, they all seek to identify and reduce the noise within the data.

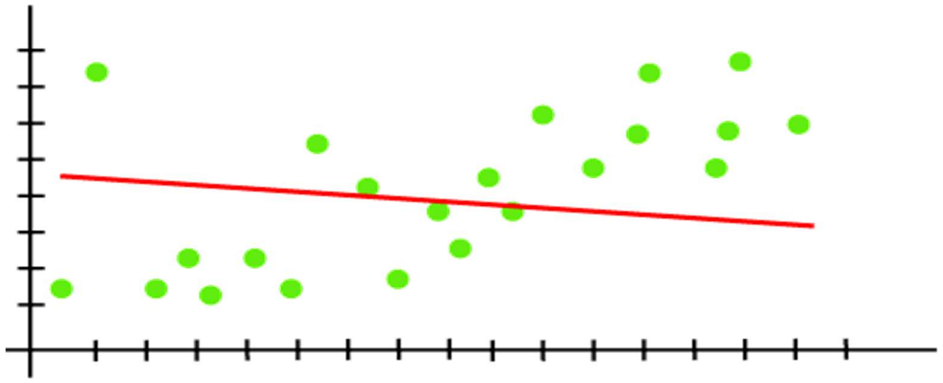
L1 Regularization	**L2 Regularization**
1. L1 penalizes sum of absolute values of weights.	1. L2 penalizes sum of square values of weights.
2. L1 generates model that is simple and interpretable.	2. L2 regularization is able to learn complex data patterns.
3. L1 is robust to outliers.	3. L2 is not robust to outliers.

7. **Ensemble methods:**

Ensemble learning methods are made up of a set of classifiers—e.g. decision trees—and their predictions are aggregated to identify the most popular result. The most well-known ensemble methods are bagging and boosting. In bagging, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these models are then trained independently, and depending on the type of task—i.e. regression or classification—the average or majority of those predictions yield a more accurate estimate. This is commonly used to reduce variance within a noisy dataset.

4.5.4 Underfitting

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data.
In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions.
An underfitted model has high bias and low variance.
Example: We can understand the underfitting using below output of the linear regression model:



As we can see from the above diagram, the model is unable to capture the data points present in the plot.

How to avoid underfitting:

- By increasing the training time of the model.
- By increasing the number of features.