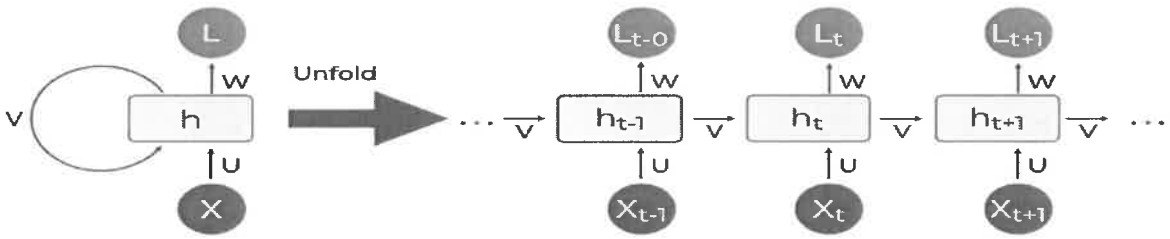
UNIT IV

SEQUENCE MODELLING – RECURRENT AND RECURSIVE NETS

Recurrent Neural Networks:

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

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.



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:

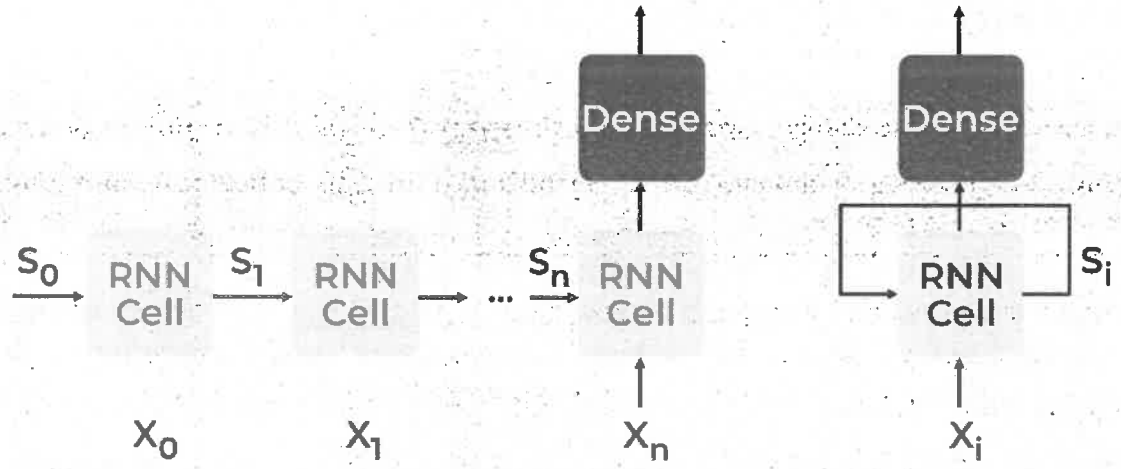
$$h = \sigma (UX + Wh_{-1} + B)$$

$$Y = O (Vh + C) \text{ Hence}$$

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

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 NETWORKS



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):

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

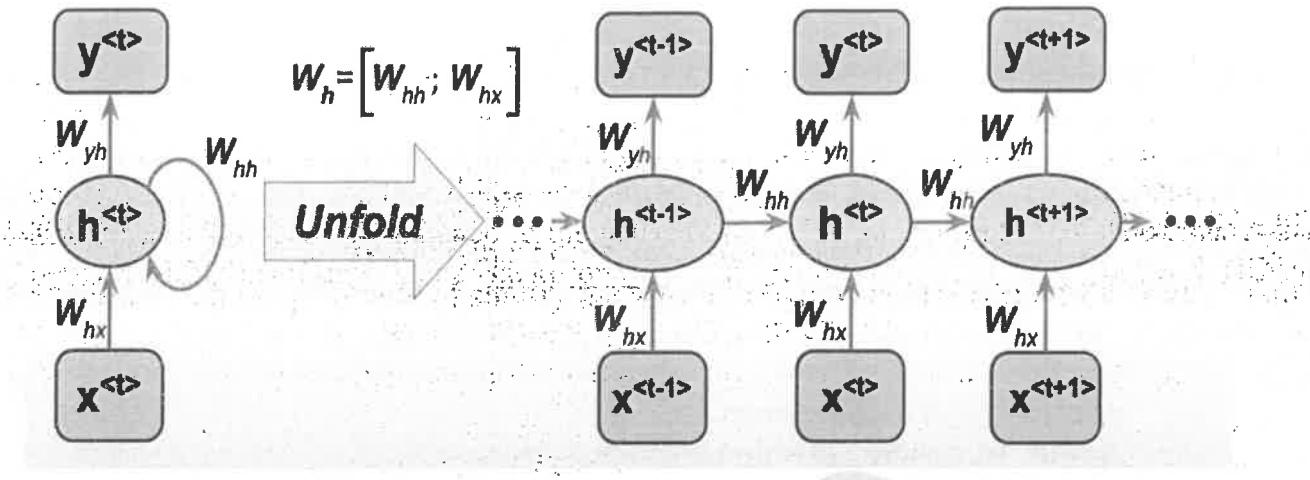
where:

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



- A basic Recurrent Neural Network (RNN) is a type of neural network designed for sequence data. Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a hidden state that captures information about previous inputs in the sequence. This makes RNNs well-suited for tasks involving sequential or time-series data.
- The architecture of a basic RNN consists of the following components:

1. **Input Layer:**
The input layer represents the input features for each time step in the sequence.

2. Recurrent Connection:
The key feature of an RNN is the recurrent connection, which allows information to persist across different time steps. At each time step, the hidden state from the previous time step is used in combination with the current input to produce the output and update the hidden state.

3. Hidden State:
The hidden state captures information about previous inputs in the sequence. It is updated at each time step based on the current input and the previous hidden state.

4. Output Layer:
The output layer produces the output for the current time step based on the current input and the hidden state.

The update equations for a basic RNN can be expressed as follows, where h_t is the hidden state at time step t, x_t is the input at time step t, and y_t is the output at time step t:

$$h_t = \text{activation}(W_{hh} \cdot h_{t-1} + W_{hx} \cdot x_t + b_h)$$

$$y_t = W_{yh} \cdot h_t + b_y$$

In these equations:

- W_{hh} and W_{hx} are weight matrices for the recurrent and input connections, respectively.
- W_{yh} is the weight matrix for the output layer.
- b_h and b_y are bias vectors for the hidden state and output layer, respectively.
- activation is an activation function applied element-wise.

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.

Advantages of Recurrent Neural Network

- 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.
- Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.
- Ability To Handle Variable-Length Sequences
- Memory Of Past Inputs
- Parameter Sharing
- Sequential Processing
- Flexibility
- Improved Accuracy

Disadvantages of Recurrent Neural Network

- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or Relu as an activation function.
- Vanishing And Exploding Gradients
- Computational Complexity
- Difficulty In Capturing Long-Term Dependencies
- Lack Of Parallelism
- Difficulty In Choosing the Right Architecture
- Difficulty In Interpreting the Output

Applications

RNN is good for sequences of data

- Language Modelling and Generating Text
- Speech Recognition
- Machine Translation
- Image Recognition, Face detection
- Time series Forecasting
- NLP… say a sentence is positive or negative
- Stock market prediction
- SPAM classifier.
- Machine Translation
- Video Tagging
- Text Summarization
- Call Centre Analysis
- Music composition…..

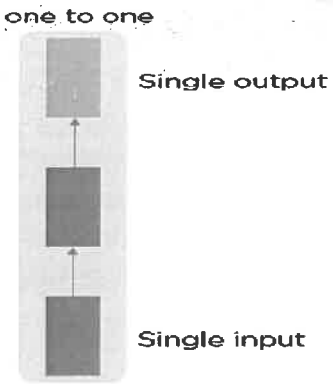
Types Of RNN

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

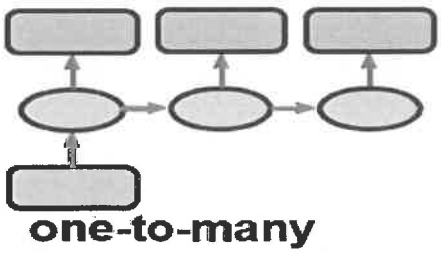
One to One :

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.



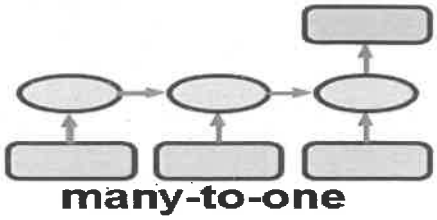
One To Many

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.



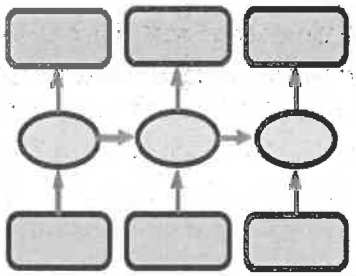
Many to One

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 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.



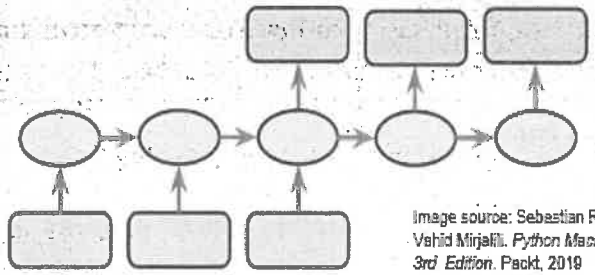
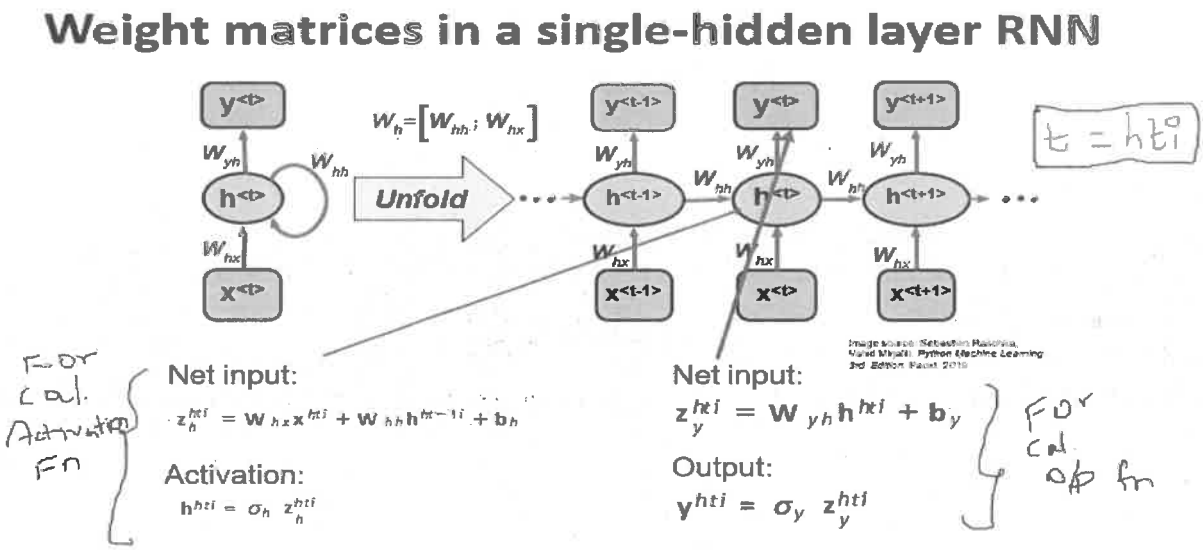


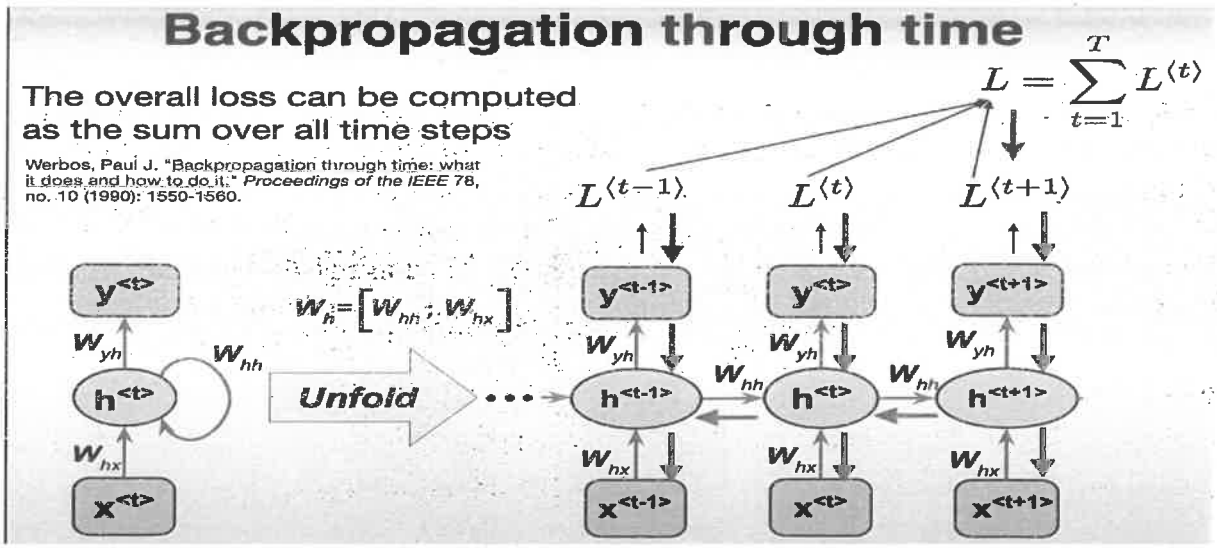
Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Packt, 2019

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.

- When we apply a Backpropagation algorithm to a Recurrent Neural Network with time series data as its input, we call it backpropagation through time.
- A single input is sent into the network at a time in a normal RNN, and a single output is obtained. Backpropagation, on the other hand, uses both the current and prior inputs as input. This is referred to as a timestep, and one timestep will consist of multiple time series data points entering the RNN at the same time.
- The output of the neural network is used to calculate and collect the errors once it has trained on a time set and given you an output. The network is then rolled back up, and weights are recalculated and adjusted to account for the faults.





$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

$$L = \sum_{t=1}^{T} L^{(t)} \qquad \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \boxed{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

This is very problematic:
Vanishing/Exploding gradient problem!

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

There are **some issues** with Back-propagation such as:

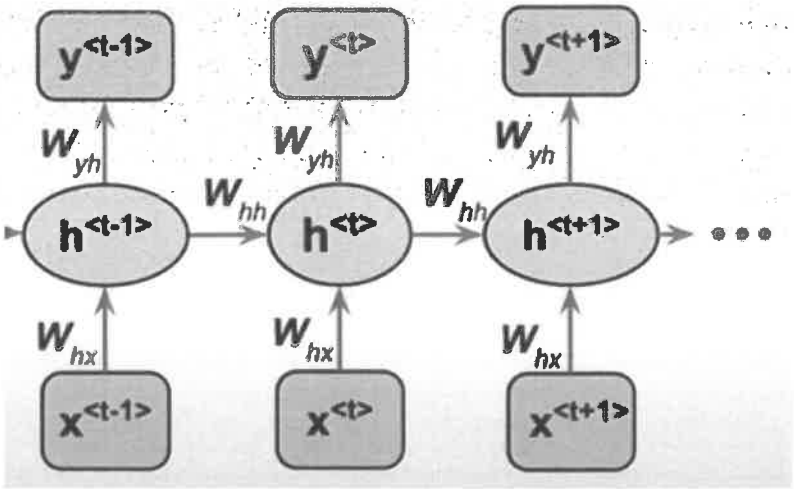
- **Vanishing gradients:** If weights are small, gradient shrinks exponentially. Network stops learning.
- ○ **Exploding gradients:** If weights are large, gradient grows exponentially. Weights fluctuate and become unstable.

Let us see how to do this
For simplicity, we will short-circuit some of the paths

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 h_t becomes h_{t-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.



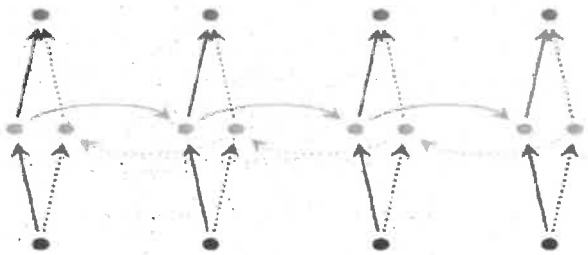
Variation Of Recurrent Neural Network (RNN)

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)

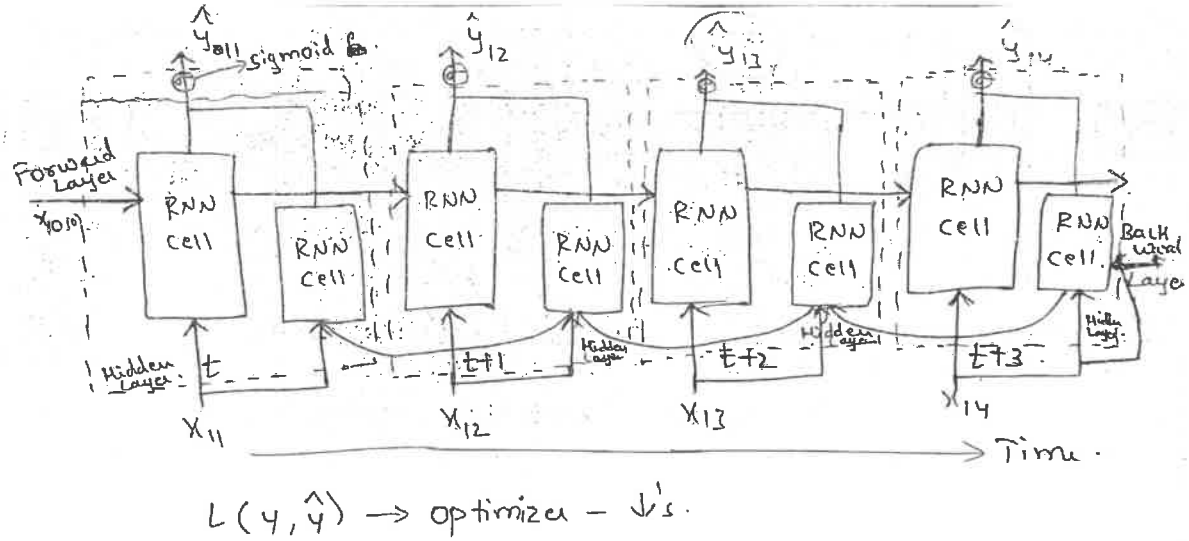
Bidirectional Neural Network (BiNN)

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 directions 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.



- 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.



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.
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. **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.
5. **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.

Advantages

- Context from both past and future
- Enhanced accuracy
- Efficient handling of variable-length sequences

Disadvantages

- Computational complexity

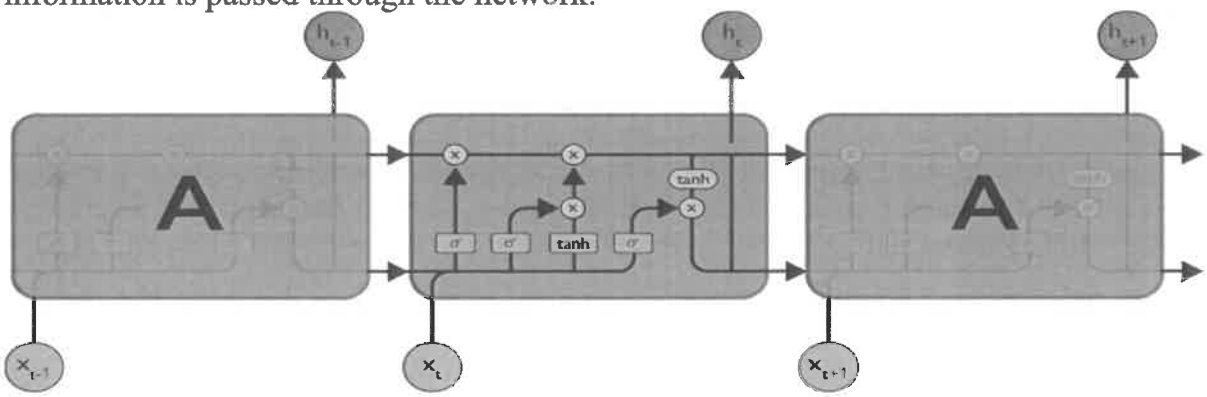
- Long training time
- Difficulty in parallelization
- Overfitting

Applications

- Handwriting Recognition
- Speech Recognition
- Dependency Parsing
- Natural Language Processing

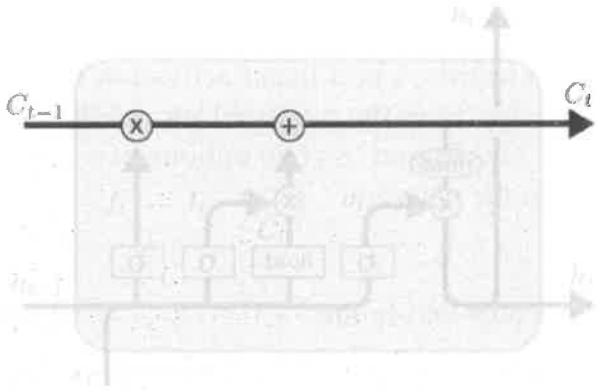
Long Short-Term Memory (LSTM)

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 these three new gates are introduced in the RNN. In this way, only the selected information is passed through the network.



The Core Idea Behind LSTMs

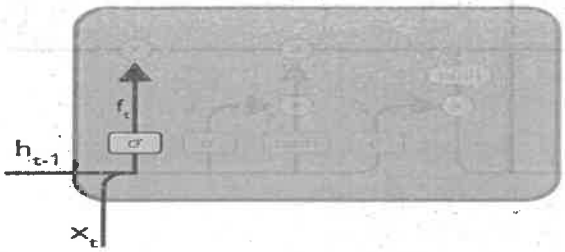
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. An LSTM has three of these gates, to protect and control the cell state.



Step-by-step approach to understand LSTM networks

Step 1: Decide how much past data it should remember:

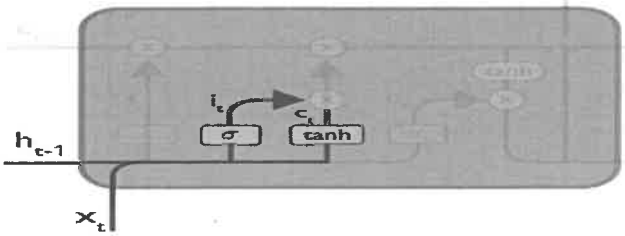
The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at the previous state (h_{t-1}) along with the current input x_t and computes the function. The calculation is done by considering the new input and the previous timestamp which eventually leads to the output of a number between 0 and 1 for each number in that cell state.



$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$w_f = Weight$
$h_{t-1} = Output\ from\ previous\ timestamp$
$x_t = New\ input$
$b_f = Bias$

Step 2: Decide how much this unit adds to the current state:
In the second layer, there are two parts. One is the sigmoid function, and the other is the tanh function.
In the sigmoid function, it decides which values to let through (0 or 1).
Tanh function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

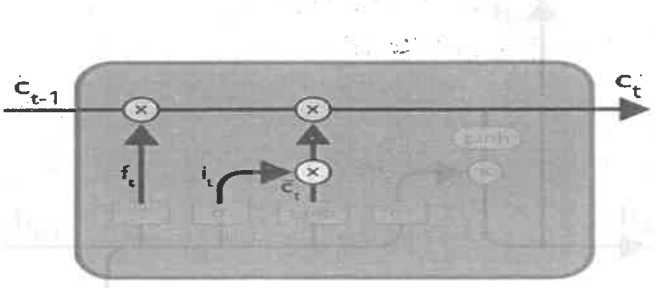


$$i_t = \sigma\,(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

i_t = input gate
Determines which information to let through based on its significance in the current time step

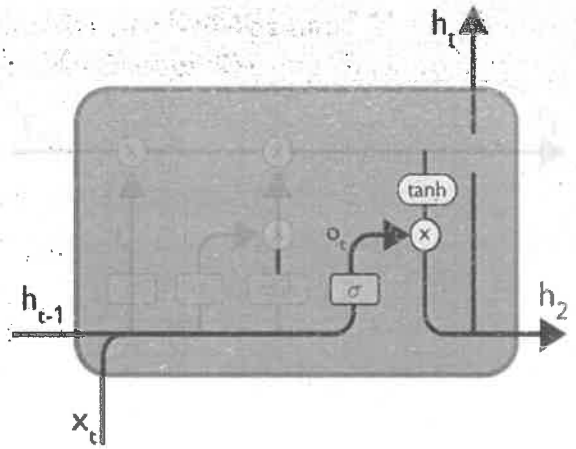
Step 3: Update old cell into new cell

- Now, we will update the old cell state C_{t-1}, into the new cell state C_t.
- First, we multiply the old state (C_{t-1}) by f(t), forgetting the things we decided to leave behind earlier.
- Then, we add $i_t * \tilde{c}_t$. This is the new candidate values, scaled by how much we decided to update each state value.
- In the second step, we decided to do make use of the data which is only required at that stage.
- In the third step, we actually implement it.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step 4: Decide What Part of the Current Cell State Makes It to the Output:
The fourth step is to decide what the output will be. First, we run a sigmoid layer, which decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.



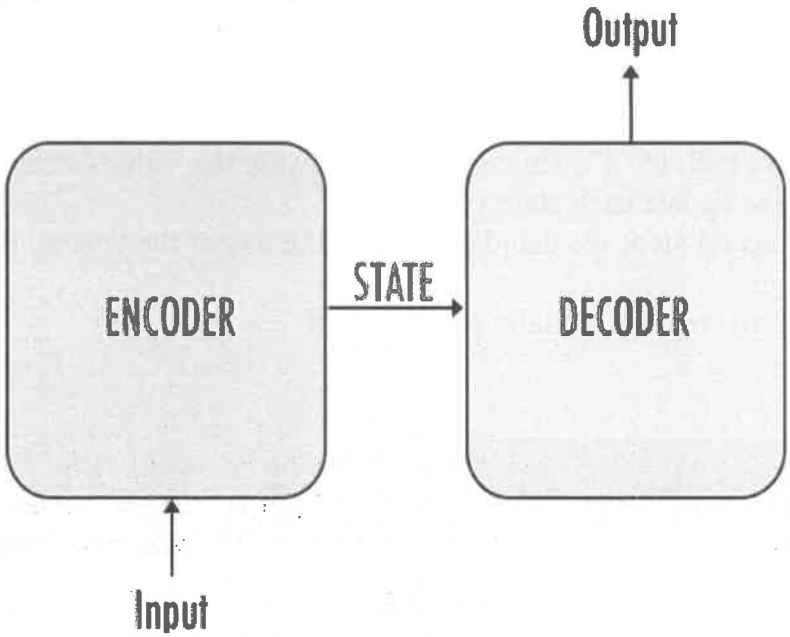
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

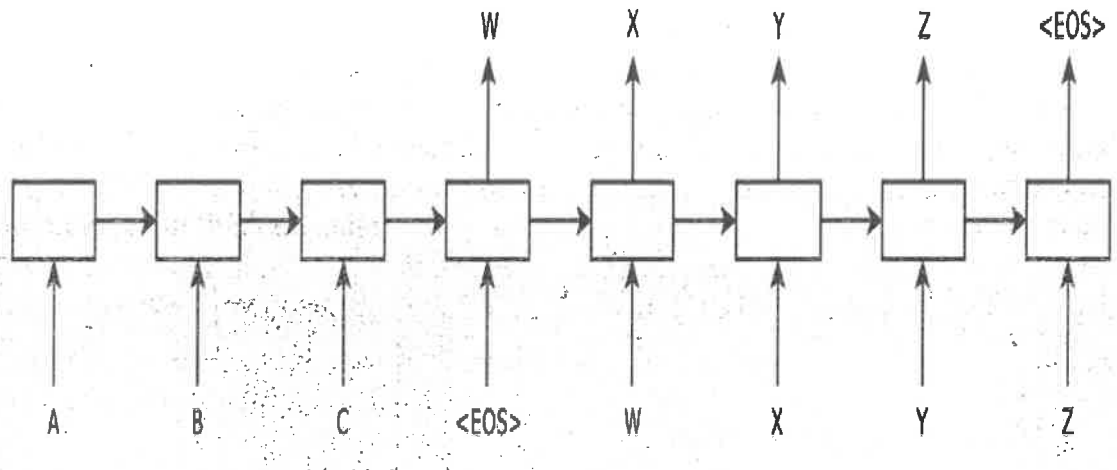
$$h_t = o_t * \tanh\left(C_t\right)$$

o_t = output gate
Allows the passed in information to impact the output in the current time step

Encoder-decoder sequence to sequence architectures

Sequence to Sequence (often abbreviated to seq2seq) models is a special class of Recurrent Neural Network architectures that we typically use (but not restricted) to solve complex Language problems. The RNN variant LSTM (Long Short-term Memory) is the most used cell in seq-seq learning tasks. The encoder-decoder architecture for recurrent neural networks is the standard neural machine translation method that rivals and, in some cases, outperforms classical statistical machine translation methods.





There are three main blocks in the encoder-decoder model,

- Encoder
- Context Vector
- Decoder

- The Encoder will convert the input sequence into a single-dimensional vector (hidden/context vector).
- The decoder will convert the hidden /context vector into the output sequence.
- Encoder-Decoder models are jointly trained to maximize the conditional probabilities of the target sequence given the input sequence.

Any hidden state h_t is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

The output y_t at time step t is computed using the formula:

$$y_t = softmax(W^S h_t)$$

Drawbacks of Encoder-Decoder Model :

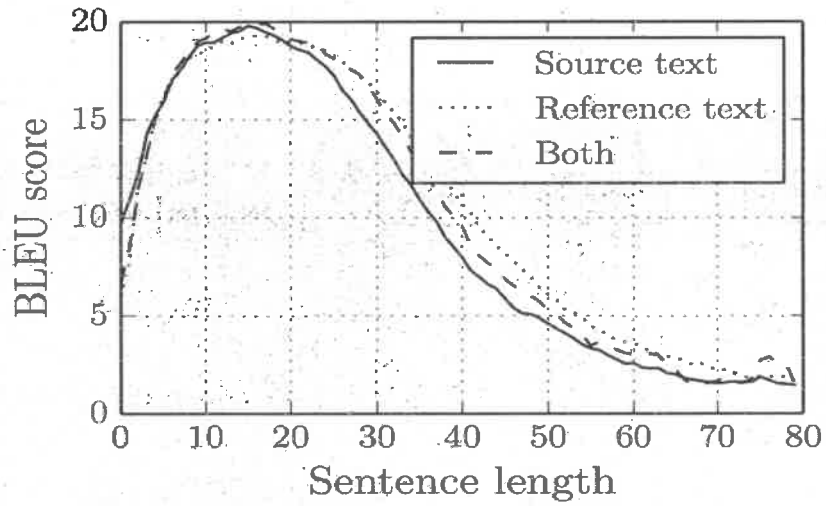
There are two primary drawbacks to this architecture, both related to length.

1. Firstly, as with humans, this architecture has very limited memory. That final hidden state of the LSTM, which we call S or W, is where you're trying to cram the entirety of the sentence you have to translate.
   - S or W is usually only a few hundred units (read: floating-point numbers) long — the more you try to force into this fixed dimensionality vector, the lossier the neural network is forced to be.
   - Thinking of neural networks in terms of the "lossy compression" they're required to perform is sometimes quite useful.

2) Second, as a general rule of thumb, the deeper a neural network is, the harder it is to train.

- For RNNs, the longer the sequence is, the deeper the neural network is along the time dimension.
- This results in vanishing gradients, where the gradient signal from the objective that the RNN learns from disappears as it travels backward.

- Even with RNNs specifically made to help prevent vanishing gradients, such as the LSTM, this is still a fundamental problem.



Furthermore, for more robust and lengthy sentences we have models like Attention Models and Transformers.

Applications

It possesses many applications such as

- Google's Machine Translation
- Question answering chatbots
- Speech recognition
- Time Series Application etc.,

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 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 major drawback of RNN	These problems also occur in DNN but these are not the major problem with DNN