# RECURRENT NEURAL NETWORKS (RNNS)

**Dr P Bhagath M.Tech (IITG), Ph.D(IITG)**

Department of CSE,
LBRCE Mylavaram

April 1, 2024

# Content

# Introduction

- A Neural Network consists of different layers connected to each other, working on the structure and function of a human brain. It learns from huge volumes of data and uses complex algorithms to train a neural net
- **Example: Identify a dog's breed based on their features**
  - The image pixels of two different breeds of dogs are fed to the input layer of the neural network
  - The image pixels are then processed in the hidden layers for feature extraction
  - The output layer produces the result to identify if it's a German Shepherd or a Labrador
  - **Such networks do not require memorizing the past output**

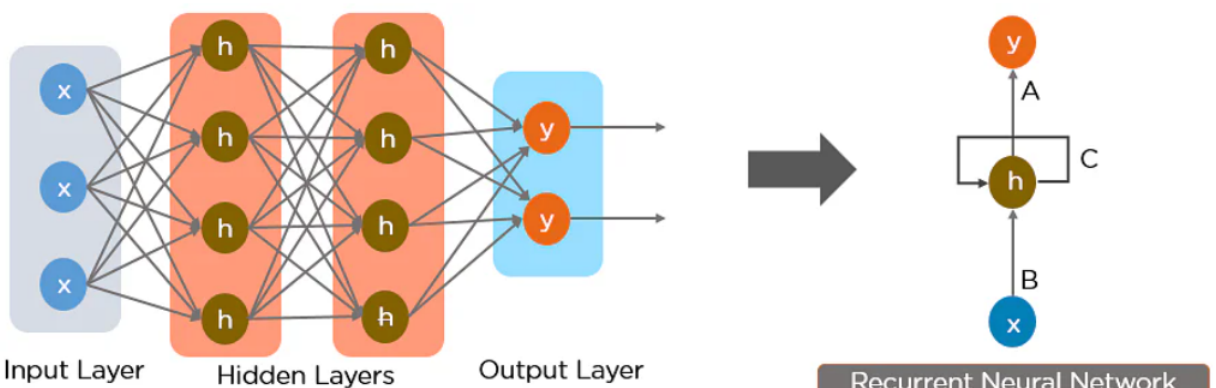
# Applications of Various Neural Networks

- **Feed-Forward Neural Network:** Used for general Regression and Classification problems
- **Convolutional Neural Network:** Used for object detection and image classification
- **Deep Belief Network:** Used in healthcare sectors for cancer detection
- **RNN:** Used for speech recognition, voice recognition, time series prediction, and natural language processing

# What is RNN?

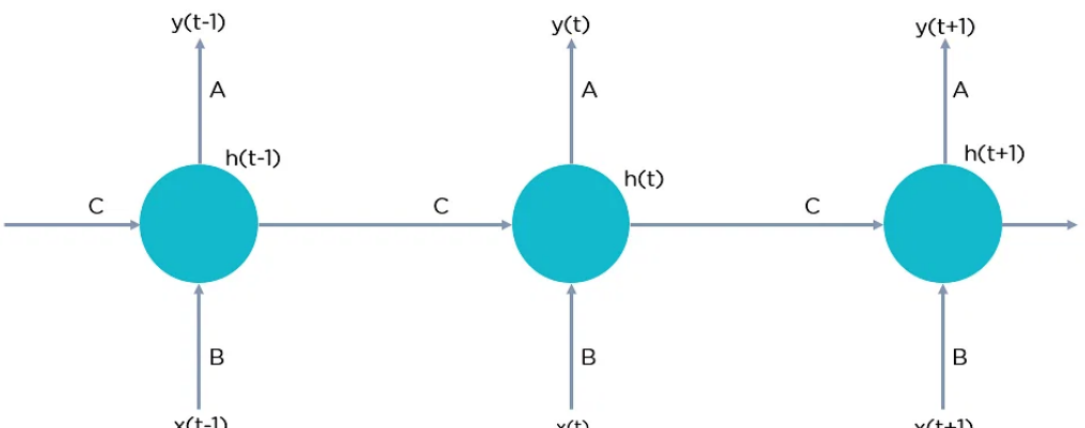
- RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer

- The nodes in different layers of the neural network are compressed to form a single layer of recurrent neural networks. A, B, and C are the parameters of the network.



# WHAT IS RNN?

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




$h(t) = f_c(h(t-1), x(t))$

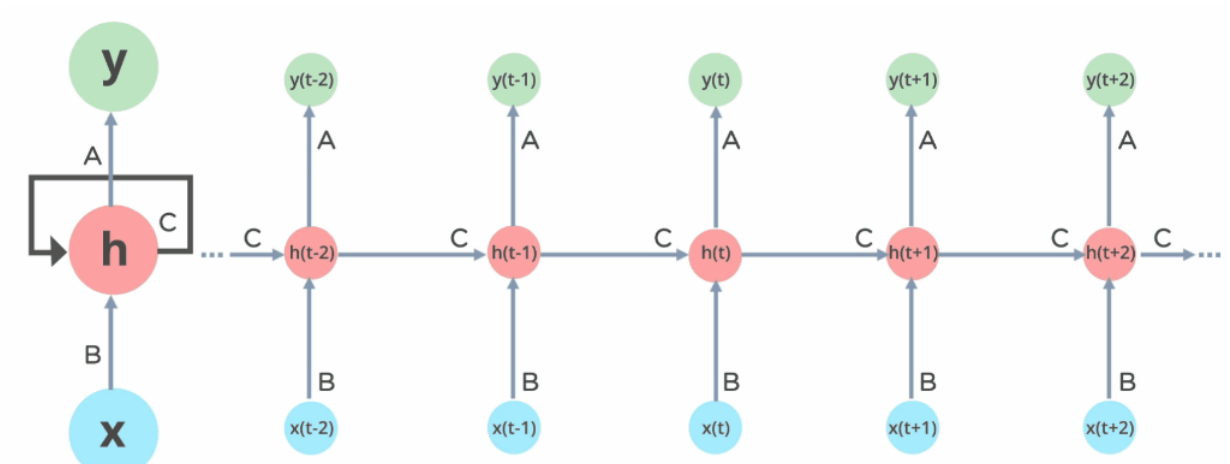
h(t) = new state
$f_c$ = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

# WHY RNN?

- RNN were created because there were a few issues in the feed-forward neural network:
  - **Cannot handle sequential data**
  - Considers only the current input
  - Cannot memorize previous inputs
  - The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

# How RNN works?

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

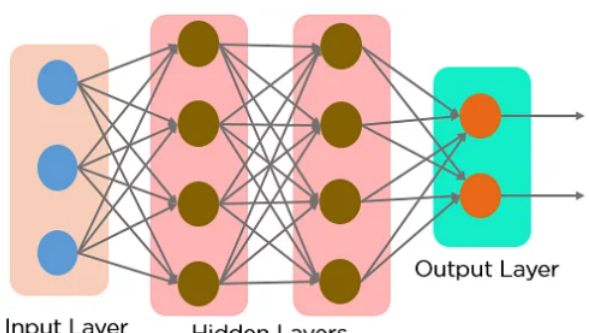


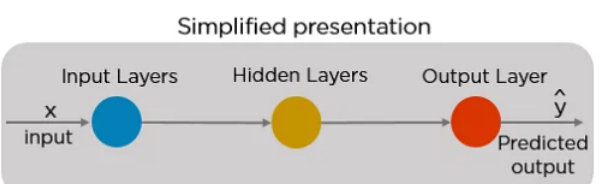
# How RNN works

- **The input layer 'x'** takes in the input to the neural network and processes it and passes it onto the middle layer.
- **The middle layer 'h'** can consist of multiple hidden layers, each with its own activation functions and weights and biases.
- If you have a neural network where the various parameters of different hidden layers are not affected by the previous layer, ie: the neural network does not have memory, then you can use a recurrent neural network.
- The Recurrent Neural Network will standardize the different activation functions and weights and biases so that each hidden layer has the same parameters. Then, instead of creating multiple hidden layers, it will create one and loop over it as many times as required.

# Feed Forward Networks vs RNN

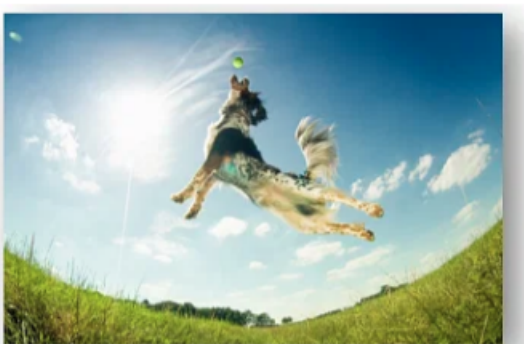
- A feed-forward neural network allows information to flow only in the forward direction, from the input nodes, through the hidden layers, and to the output nodes. **There are no cycles or loops in the network.**
- In a feed-forward neural network, the decisions are based on the current input. It doesn't memorize the past data, and there's no future scope. Feed-forward neural networks are used in general regression and classification problems.



# Applications of RNN-1 Image Captioning

RNNs are used to caption an image by analyzing the activities present.



"A Dog catching a ball in mid air"

# Applications of RNN-2 NLP


When it rains, look for rainbows.
When it's dark, look for stars.

Positive Sentiment

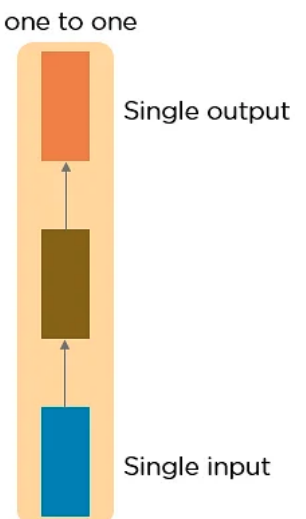
# Applications of RNN-3 Machine Translation

Here the person is speaking in English and it is getting translated into Chinese, Italian, French, German and Spanish languages

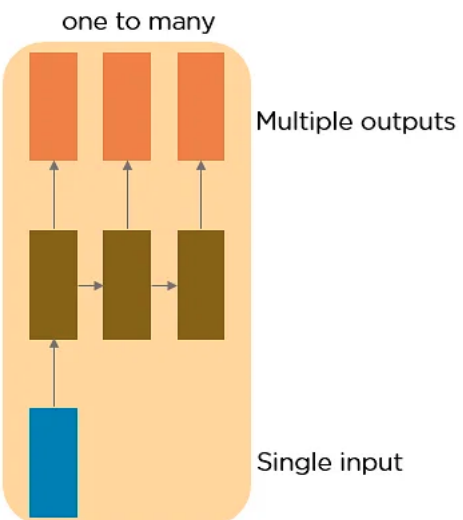
# Types of RNN

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

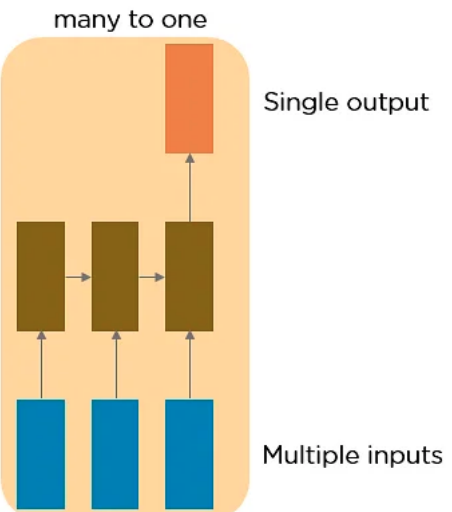
# One to One



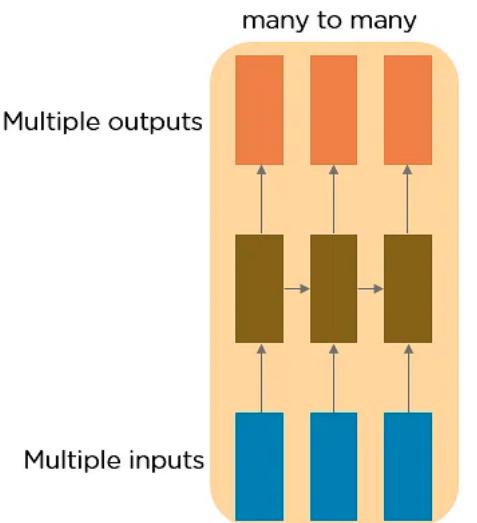
# One to Many



# Many to One



# Many to Many

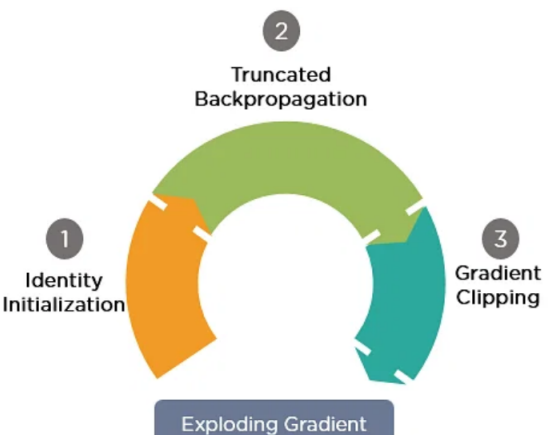


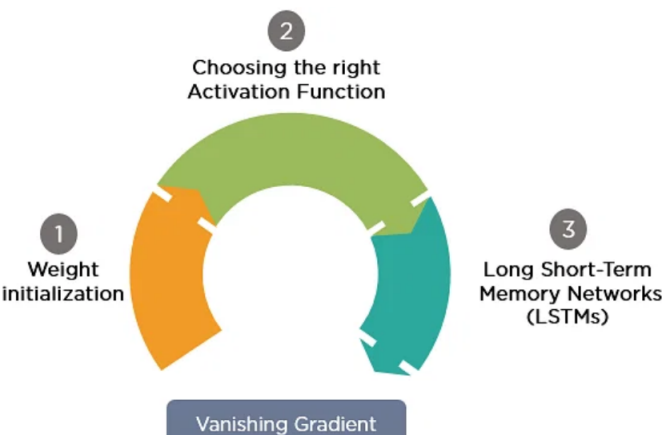
# Issues of Standard RNNs

1. **Vanishing Gradient Problem:**
   Recurrent Neural Networks enable you to model time-dependent and sequential data problems, such as stock market prediction, machine translation, and text generation. You will find, however, RNN is hard to train because of the gradient problem.
   RNNs suffer from the problem of vanishing gradients. The gradients carry information used in the RNN, and when the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.

2. **Exploding Gradient Problem:** While training a neural network, if the slope tends to grow exponentially instead of decaying, this is called an Exploding Gradient. This problem arises when large error gradients accumulate, resulting in very large updates to the neural network model weights during the training process.
   Long training time, poor performance, and bad accuracy are the major issues in gradient problems

# Solutions


2
Truncated
Backpropagation
1
Identity
Initialization
3
Gradient
Clipping
Exploding Gradient


2
Choosing the right
Activation Function
1
Weight
initialization
3
Long Short-Term
Memory Networks
(LSTMs)
Vanishing Gradient

# Identity Initialization

- The **identity RNN architecture** is a kind of RNN where the activation functions are all ReLU, and the recurrent weights are initialized to the identity matrix.
- The ReLU activation function clips the activations to be nonnegative, but for nonnegative activations, it's equivalent to the identity function.

# Truncated Backpropagation

- The input is considered as fixed-length subsequences in truncated backpropagation through time (TBPTT). The hidden state of the previous subsequence is passed on as input to the following subsequence in the forward pass. On the other hand, the computed gradient values are dropped at the end of each subsequence as we move back in gradient computation.
- The gradient values at time $t'$ are employed in every time step t if $t < t'$ in normal backpropagation. If $t' - t$ exceeds the subsequence length, the gradients do not flow from $t'$ to t in shortened backpropagation.

# Gradient Clipping

- The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.
- **Gradient Clipping** is a technique used to cope with the **exploding gradient problem** sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.

## Solutions to Vanishing Gradient

- **Weight Initialization**
- from keras.models import Sequential
  from keras.layers import Dense, Activation
  model = Sequential([
  Dense(16, input_shape=(1,5), activation='relu'),
  Dense(32, activation='relu',kernel_initializer='glorot_uniform'),
  Dense(2, activation='softmax') ])
- **Choosing the right activation function**
- **LSTMs**

# What is Backpropagation?

**Definition** **Backpropagation** is a training algorithm that is used for training neural networks. When training a neural network, we are actually tuning the weights of the network to minimize the error with respect to the already available true values(labels) by using the Backpropagation algorithm. It is a supervised learning algorithm as we find errors with respect to already given labels.

**Algorithm**

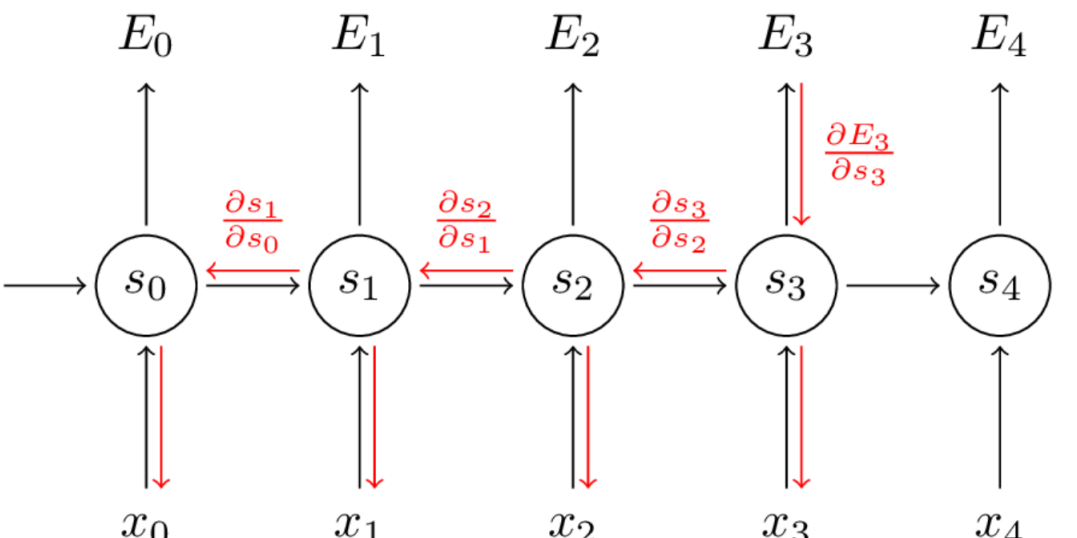
1. Present a training input pattern and propagate it through the network to get an output.
2. Compare the predicted outputs to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Use these calculated derivatives to adjust the weights to minimize the error.
5. Repeat.

# BACKPROPAGATION THROUGH TIME (BPTT)

1. RNN uses a technique called **Backpropagation through time to backpropagate through the network** to adjust their weights so that we can reduce the error in the network. It got its name "through time" as in RNN we deal with sequential data and every time we go back it's like going back in time towards the past

2. In the BPTT step, we calculate the partial derivative at each weight in the network. So if we are in time t = 3, then we consider the derivative of E3 with respect to that of S3. Now, x3 is also connected to s3. So, its derivative is also considered. Now if we see s3 is connected to s2 so s3 is depending on the value from s2 and here derivative of s3 with respect to s2 is also considered. This acts as a chain rule and we accumulate all the dependency with their derivatives and use it for error calculation.

# BPTT


$E_0$
$E_1$
$E_2$
$E_3$
$E_4$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
$\frac{\partial E_3}{\partial s_3}$
$s_0$
$s_1$
$s_2$
$s_3$
$s_4$
$x_0$
$x_1$
$x_2$
$x_3$
$x_4$

# Advantages and Disadvantages

| S No | Advantages | Disadvantages |
|---|---|---|
| 1. | Possibility of processing input of any length | Computation being slow |
| 1. | Possibility of processing input of any length | Computation being slow |
| 2. | Model size not increasing with size of input | Difficulty of accessing information from a long time ago |
| 3. | Computation takes into account historical information | Cannot consider any future input for the current state |
| 4. | Weights are shared across time | – |

# GATES

| Type of Gate | Role |
|---|---|
| Update gate | How much past should matter now? |
| Relevance gate | Drop previous information? |
| Forget gate | Erase a cell or not? |
| Output gate | How much to reveal of a cell? |

- Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

# LONG SHORT TERM MEMORY (LSTM)S

- Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

# Variants of RNN

- Bidirectional RNN
-

**Reach me: bhagath@lbrce.ac.in**