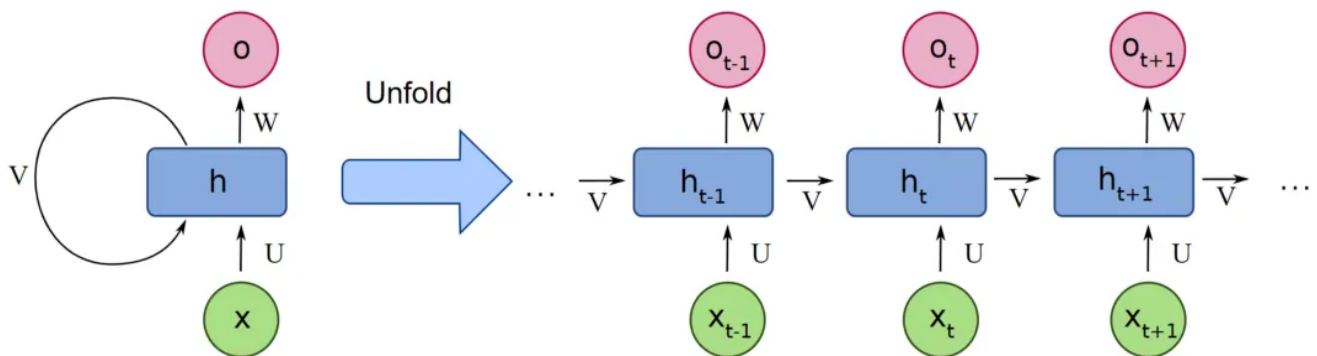


# What is Recurrent Neural Networks (RNN)?

## Introduction

Ever wonder how chatbots understand your questions or how apps like Siri and voice search can decipher your spoken requests? The secret weapon behind these impressive feats is a type of artificial intelligence called Recurrent Neural Networks (RNNs).



Unlike standard neural networks that excel at tasks like image recognition, RNNs boast a unique superpower – memory! This internal memory allows them to analyze sequential data, where the order of information is crucial. Imagine having a conversation – you need to remember what was said earlier to understand the current flow. Similarly, RNNs can analyze sequences like speech or text, making them perfect for tasks like machine translation and voice recognition. Although RNNs have been around since the 1980s, recent advancements like Long Short-Term Memory (LSTM) and the explosion of big data have unleashed their true potential.

In this article, you will explore the significance of RNN neural networks (RNN) in machine learning and deep learning. We will discuss the RNN model's capabilities and its applications in RNN in deep learning.

## Table of contents



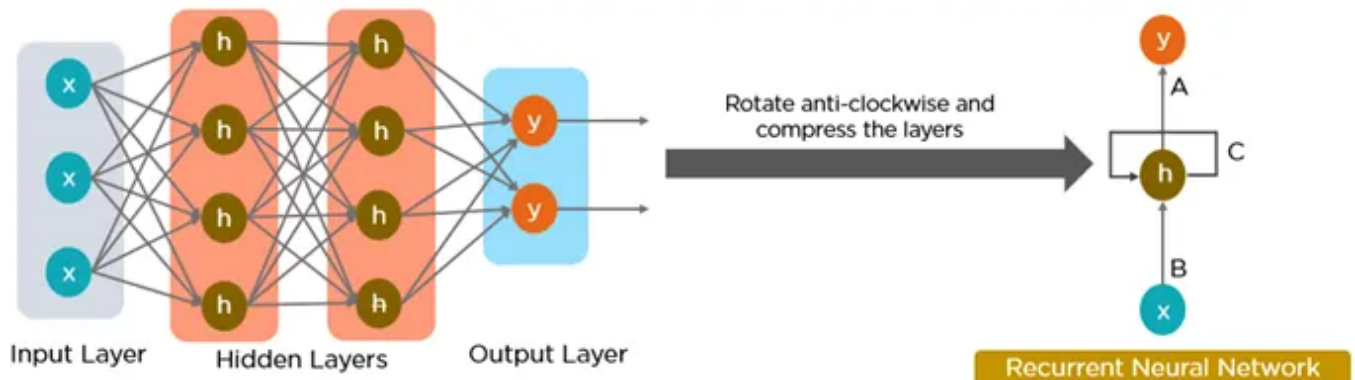
1. What is a Recurrent Neural Network (RNN)?
2. What Makes RNN Special?
3. The Architecture of a Traditional RNN
4. How does Recurrent Neural Networks work?
5. Common Activation Functions
6. Advantages and disadvantages of RNN
7. Recurrent Neural Network vs Feedforward Neural Network
8. Backpropagation Through Time (BPTT)
9. Two Issues of Standard RNNs

## What is a Recurrent Neural Network (RNN)?

Recurrent Neural networks imitate the function of the human brain in the fields of Data science, Artificial intelligence, machine learning, and deep learning, allowing computer programs to recognize patterns and solve

common issues.

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



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

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

## What Makes RNN Special?

Recurrent neural networks (RNNs) set themselves apart from other neural networks with their unique capabilities:

- **Internal Memory:** This is the key feature of RNNs. It allows them to remember past inputs and use that context when processing new information.
- **Sequential Data Processing:** Because of their memory, RNNs are exceptional at handling sequential data where the order of elements matters. This makes them ideal for tasks like speech recognition, machine translation, natural language processing(nlp) and text generation.
- **Contextual Understanding:** RNNs can analyze the current input in relation to what they've "seen" before. This contextual understanding is crucial for tasks where meaning depends on prior information.
- **Dynamic Processing:** RNNs can continuously update their internal memory as they process new data. This allows them to adapt to changing patterns within a sequence.

## RNN Architecture

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

Here's a breakdown of its key components:

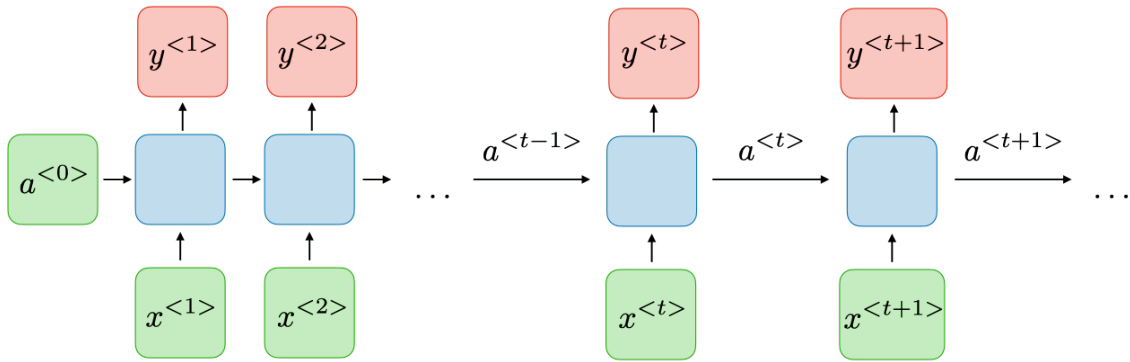
- **Input Layer:** This layer receives the initial element of the sequence

data. For example, in a sentence, it might receive the first word as a vector representation.

- **Hidden Layer:** The heart of the RNN, the hidden layer contains a set of interconnected neurons. Each neuron processes the current input along with the information from the previous hidden layer's state. This "state" captures the network's memory of past inputs, allowing it to understand the current element in context.
- **Activation Function:** This function introduces non-linearity into the network, enabling it to learn complex patterns. It transforms the combined input from the current input layer and the previous hidden layer state before passing it on.
- **Output Layer:** The output layer generates the network's prediction based on the processed information. In a language model, it might predict the next word in the sequence.
- **Recurrent Connection:** A key distinction of RNNs is the recurrent connection within the hidden layer. This connection allows the network to pass the hidden state information (the network's memory) to the next time step. It's like passing a baton in a relay race, carrying information about previous inputs forward

## The Architecture of a Traditional RNN

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

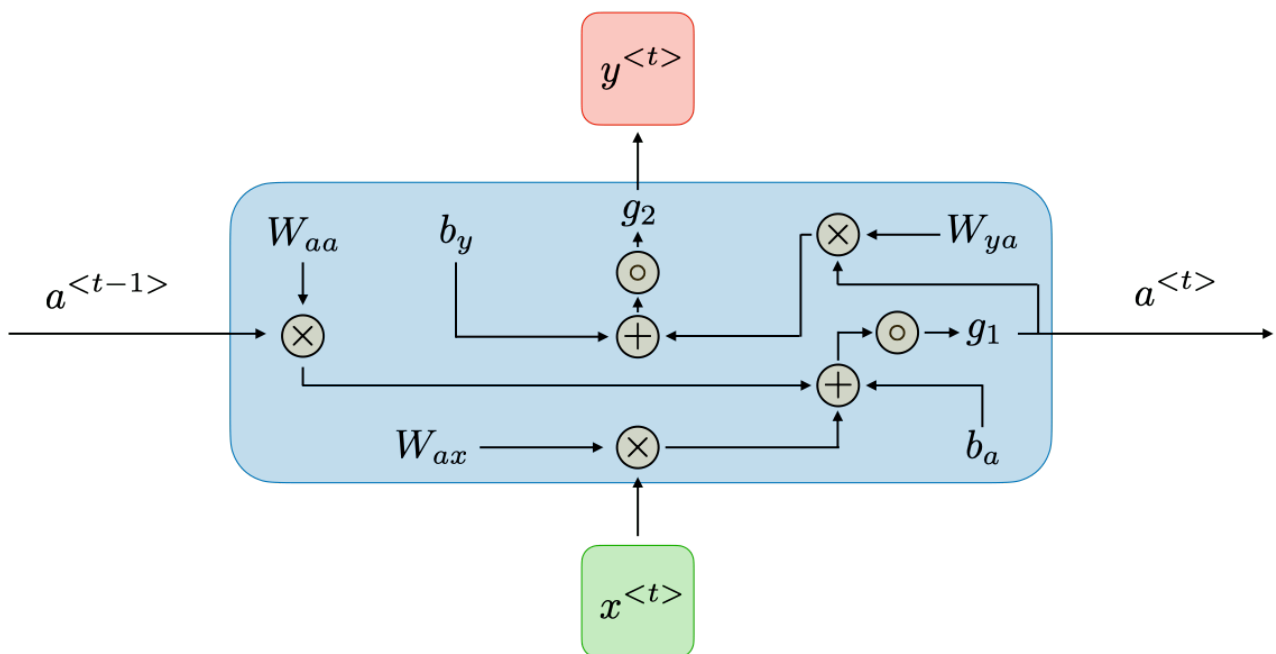


Source: Stanford.edu

For each timestep  $t$ , the activation  $a^{<t>}$  and the output  $y^{<t>}$  are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where  $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$  are coefficients that are shared temporally and  $g_1, g_2$  activation functions.



Source: Stanford.edu

RNN architecture can vary depending on the problem you're trying to solve.

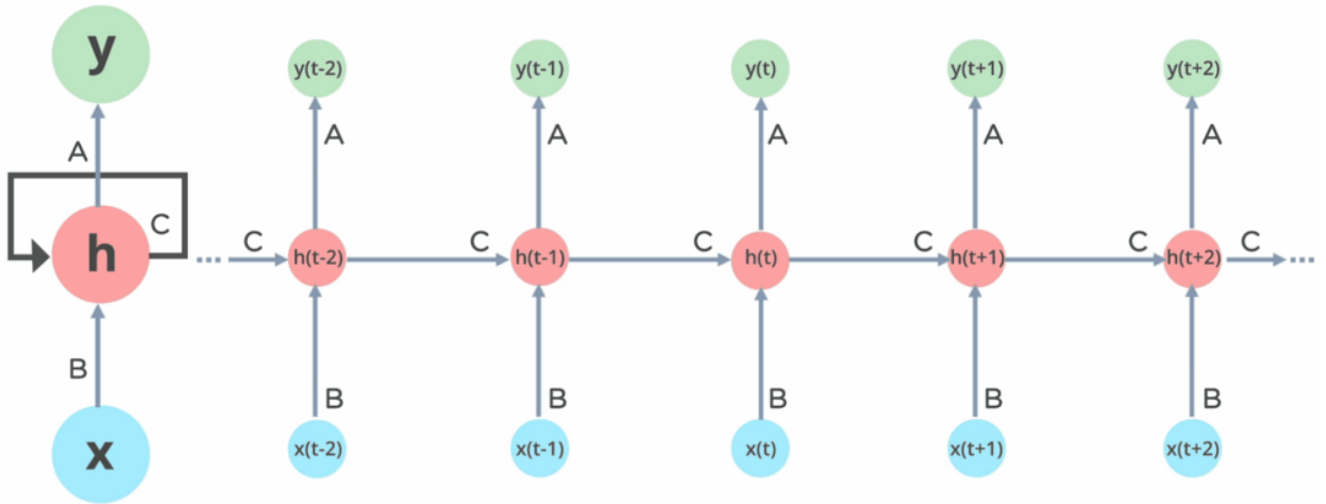
From those with a single input and output to those with many (with variations between).

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

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

## How does Recurrent Neural Networks work?

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



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

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

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

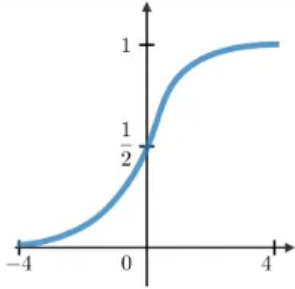
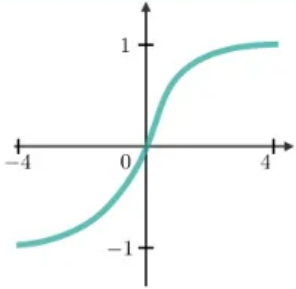
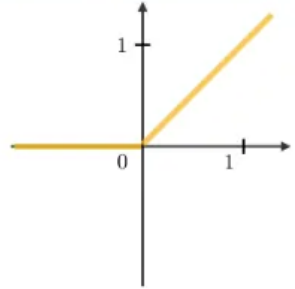
## Common Activation Functions

A neuron's activation function dictates whether it should be turned on or off. Nonlinear functions usually transform a neuron's output to a number



between 0 and 1 or -1 and 1.

The following are some of the most commonly utilized functions:

| Sigmoid   | Tanh  | RELU  |
|---|---|---|
| $g(z) = \frac{1}{1 + e^{-z}}$   | $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$  | $g(z) = \max(0, z)$   |
|  |  |  |

- **Sigmoid Function ( $\sigma(x)$ )**

- Formula:  $\sigma(x) = 1 / (1 + e^{(-x)})$
- Behavior: Squishes any real number between 0 and 1.

- **Hyperbolic Tangent ( $\tanh(x)$ )**

- Formula:  $\tanh(x) = (e^x - e^{(-x)}) / (e^x + e^{(-x)})$
- Behavior: Squeezes any real number between -1 and 1.

- **Rectified Linear Unit (ReLU)(x)**

- Formula:  $\text{ReLU}(x) = \max(0, x)$
- Behavior: Outputs the input value if positive, otherwise outputs 0.

- **Leaky ReLU (Leaky ReLU(x))**

- Formula:  $\text{Leaky ReLU}(x) = \max(\alpha * x, x)$  (where  $\alpha$  is a small positive value, typically 0.01)
- Behavior: Similar to ReLU, but for negative inputs, it outputs a small

fraction of the input instead of 0.

- **Softmax (softmax(x))**

- Formula:  $\text{softmax}(x)_i = \exp(x_i) / \sum(\exp(x_j))$  (where  $i$  represents an element in the vector  $x$  and  $\Sigma$  denotes the sum over all elements  $j$  in  $x$ )
- Behavior: Converts a vector of real numbers into a probability distribution where all elements sum to 1.

## **Advantages and disadvantages of RNN**

### **Advantages of RNNs:**

- Handle sequential data effectively, including text, speech, and time series.
- Process inputs of any length, unlike feedforward neural networks.
- Share weights across time steps, enhancing training efficiency.

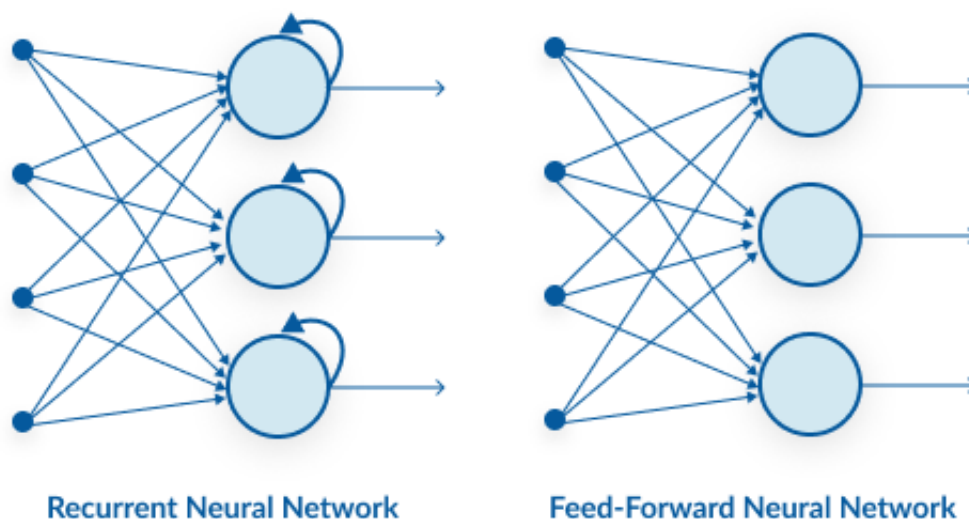
### **Disadvantages of RNNs:**

- Prone to vanishing and exploding gradient problems, hindering learning.
- Training can be challenging, especially for long sequences.
- Computationally slower than other neural network architectures.

## **Recurrent Neural Network vs Feedforward Neural Network**

## Information Flow

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



- **FNNs:** A feed-forward neural network has only one route of information flow: from the input layer to the output layer, passing through the hidden layers. The data flows across the network in a straight route, never going through the same node twice. A feed-forward neural network can perform simple classification, regression, or recognition tasks, but it can't remember the previous input that it has processed that's why FNNs are poor predictions of what will happen next because they have no memory of the information they receive. Because it simply analyses the current input, a feed-forward network has no idea of temporal order. Apart from its training, it has no memory of what transpired in the past.
- **RNNs:** The information is in an RNN cycle via a loop. Before making a judgment, it evaluates the current input as well as what it has learned

from past inputs. A [recurrent neural network](#), on the other hand, may recall due to internal memory. It produces output, copies it, and then returns it to the network.

## Data Type

- **FNNs:** Typically work best with **fixed-length inputs and outputs**. They excel at pattern recognition tasks where the data points are independent of each other. For instance, image recognition or spam email classification.
- **RNNs:** Shine in handling **sequential data**, where the order and relationships between elements matter. This makes them ideal for tasks like speech recognition, machine translation, and text generation where the meaning unfolds over time.

## Application

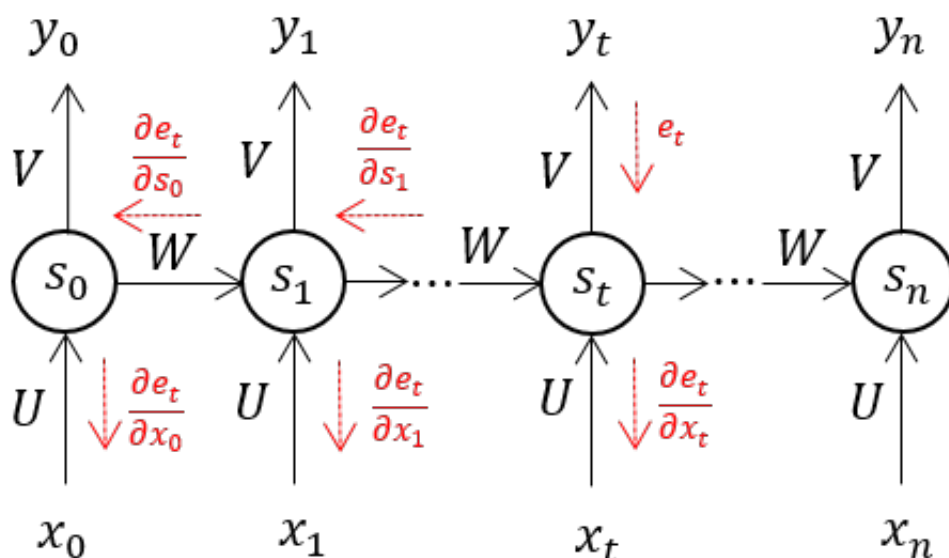
- **FNN:** Power applications like image recognition, medical diagnosis (analyzing X-rays to detect abnormalities), image classification and spam filtering (identifying unwanted emails).
- **RNNs:** Drive tasks like speech recognition (understanding spoken language), machine translation (converting text from one language to another), text generation (creating chatbots or writing different content formats), and time series forecasting (predicting stock prices or weather patterns).

# Backpropagation Through Time (BPTT)

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

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

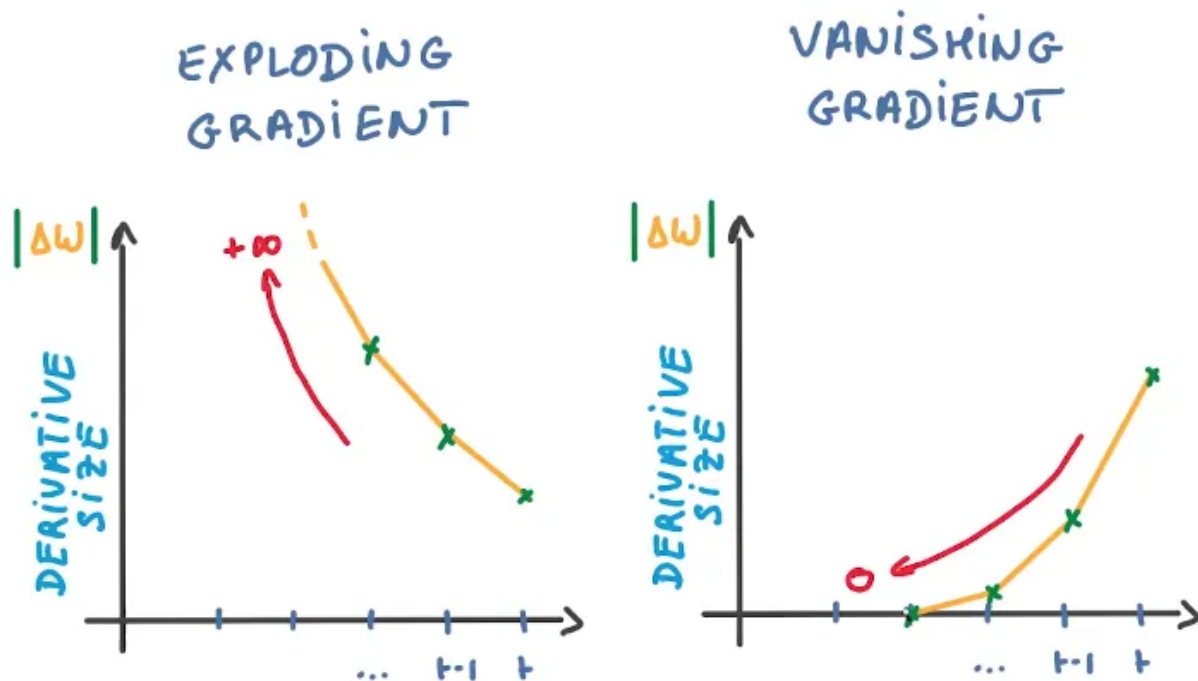
Also, you can checkout this [article](#)!



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

## Two Issues of Standard RNNs

RNNs have had to overcome two key challenges, but to comprehend them, one must first grasp what a gradient is.



With regard to its inputs, a gradient is a partial derivative. If you're not sure what that implies, consider this: a gradient quantifies how much the output of a function varies when the inputs are changed slightly.

A function's slope is also known as its gradient. The steeper the slope, the faster a model can learn, the higher the gradient. The model, on the other hand, will stop learning if the slope is zero. A gradient is used to measure the change in all weights in relation to the change in error.

- **Exploding Gradients:** Exploding gradients occur when the algorithm gives the weights an absurdly high priority for no apparent reason.

Fortunately, truncating or squashing the gradients is a simple solution to this problem.

- **Vanishing Gradients:** Vanishing gradients occur when the gradient values are too small, causing the model to stop learning or take far too long. This was a big issue in the 1990s, and it was far more difficult to address than the exploding gradients. Fortunately, Sepp Hochreiter and Juergen Schmidhuber's LSTM concept solved the problem.

## What Are Different Variations of RNN?

To overcome issues like vanishing and exploding gradient descents that hinder learning in long sequences, researchers have introduced new, advanced RNN architectures.

- **Long Short-Term Memory ([LSTM](#)):** A popular choice for complex tasks. LSTM networks introduce gates i.e., input gate, output gate and forget gate that control the flow of information within the network, allowing them to learn long-term dependencies more effectively than vanilla RNNs.
- **Gated Recurrent Unit ([GRU](#)):** Similar to LSTMs, GRUs use gates to manage information flow. However, they have a simpler architecture, making them faster to train while maintaining good performance. This makes them a good balance between complexity and efficiency.
- **[Bidirectional RNN:](#)** This variation processes data in both forward and backward directions. This allows it to capture context from both sides of

a sequence, which is useful for tasks like sentiment analysis where understanding the entire sentence is crucial.

- **Deep RNN:** By stacking multiple RNN layers on top of each other, deep RNNs create a more complex architecture. This allows them to capture intricate relationships within very long sequences of data. They are particularly useful for tasks where the order of elements spans long stretches.

## RNN Applications

Recurrent neural networks (RNNs) shine in tasks involving sequential data, where order and context are crucial. Let's explore some real-world use cases. Using RNN models and sequence datasets, you may tackle a variety of problems, including :

- **Speech Recognition:** RNNs power virtual assistants like Siri and Alexa, allowing them to understand spoken language and respond accordingly.
- **Machine Translation:** By analyzing sentence structure and context, RNNs translate languages more accurately, like Google Translate.
- **Text Generation:** RNNs are behind chatbots that can hold conversations and even creative writing tools that generate different text formats.
- **Time Series Forecasting:** RNNs analyze financial data to predict stock prices or weather patterns based on historical trends.



- **Music Generation:** RNNs can compose music by learning patterns from existing pieces and generating new melodies or accompaniments.
- **Video Captioning:** RNNs analyze video content and automatically generate captions, making video browsing more accessible.
- **Anomaly Detection:** RNNs can learn normal patterns in data streams (e.g., network traffic) and detect anomalies that might indicate fraud or system failures.
- **Sentiment Analysis:** By understanding the context and flow of text, RNNs can analyze sentiment in social media posts, reviews, or surveys.
- **Stock Market Recommendation:** RNNs can analyze market trends and news to suggest potential investment opportunities.
- **Sequence study of the genome and DNA:** RNNs can analyze sequential data in genomes and DNA to identify patterns and predict gene function or disease risk.

## Basic Python Implementation (RNN with Keras)

### Import the required libraries

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

[Copy Code](#)

Here's a simple Sequential model that processes integer sequences, embeds each integer into a 64-dimensional vector, and then uses an LSTM

layer to handle the sequence of vectors.

Copy Code

```
model = keras.Sequential()
model.add(layers.Embedding(input_dim=1000, output_dim=64))
model.add(layers.LSTM(128))
model.add(layers.Dense(10))
model.summary()
```

## Output:

Model: "sequential"

| Layer (type)              | Output Shape     | Param # |
|---------------------------|------------------|---------|
| =====                     |                  |         |
| embedding (Embedding)     | (None, None, 64) | 64000   |
| -----                     |                  |         |
| lstm (LSTM)               | (None, 128)      | 98816   |
| -----                     |                  |         |
| dense (Dense)             | (None, 10)       | 1290    |
| =====                     |                  |         |
| Total params: 164,106     |                  |         |
| Trainable params: 164,106 |                  |         |
| Non-trainable params: 0   |                  |         |

## Conclusion

Recurrent [Neural Networks](#) (RNNs) are a powerful and versatile tool with a wide range of applications. They are commonly used in language modeling and text generation, as well as voice recognition systems. One of the key advantages of RNNs is their ability to process sequential data and capture long-range dependencies. When paired with Convolutional Neural Networks (CNNs), they can effectively create labels for untagged images, demonstrating a powerful synergy between the two types of neural

networks.

However, one challenge with traditional RNNs is their struggle with learning long-range dependencies, which refers to the difficulty in understanding relationships between data points that are far apart in the sequence. This limitation is often referred to as the vanishing gradient problem. To address this issue, a specialized type of RNN called Long-Short Term Memory Networks (LSTM) has been developed, and this will be explored further in future articles. RNNs, with their ability to process sequential data, have revolutionized various fields, and their impact continues to grow with ongoing research and advancements.

Hope you find this information on RNN architecture and recurrent neural networks in deep learning helpful and insightful!