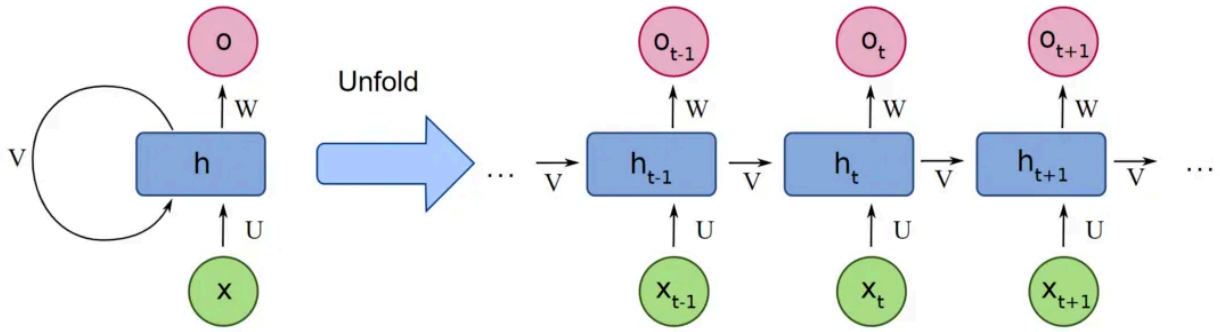
www.analyticsvidhya.com /blog/2022/03/a-brief-overview-of-recurrent-neural-networks-…

What is Recurrent Neural Networks (RNN)?

Debasish Kalita ⁝ 16-20 minutes ⁝ 3/11/2022

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).



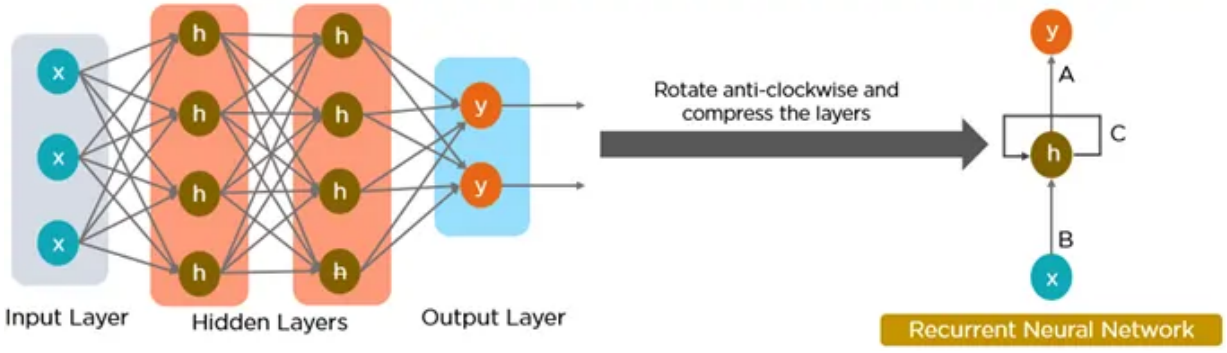
Unlike standard neural networks that excel at tasks like image recognition, RNNs boast a unique superpower – memory! This internal memory allows them to analyze sequential data, where the information order is crucial. Imagine having a conversation – you need to remember what was said earlier to understand the current flow. Similarly, RNNs can analyze sequences like speech or text, making them perfect for machine translation and voice recognition tasks. 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.

1. Understanding Recurrent Neural Network (RNN)
2. What Makes RNN Special?
3. Architecture of a Traditional RNN
4. How do Recurrent Neural Networks Work?

Understanding 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 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. They employ the same settings for each input since they produce the same outcome by performing the same task on all inputs or hidden layers.

Also Read: Introduction to Autoencoders

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 speech recognition, machine translation, natural language processing (NLP), and text generation.
- **Contextual Understanding:** RNNs can analyze the current input about 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, allowing them to adapt to changing patterns within a sequence.

Also Read: Introduction to Convolution Neural Networks

RNN Architecture

RNNs are a type of neural network that have hidden states and allow past outputs to be used as inputs. They usually follow a certain architecture. One example is the Deep RNN: by stacking multiple RNN layers on top of each other, deep RNNs create a more complex architecture. This allows them to

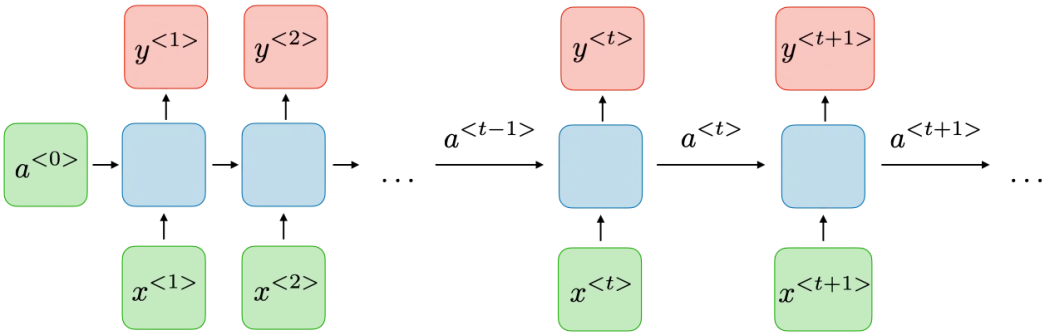
capture intricate relationships within very long sequences of data. They are particularly useful for tasks where the order of elements spans long stretches.

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

Architecture of a Traditional RNN

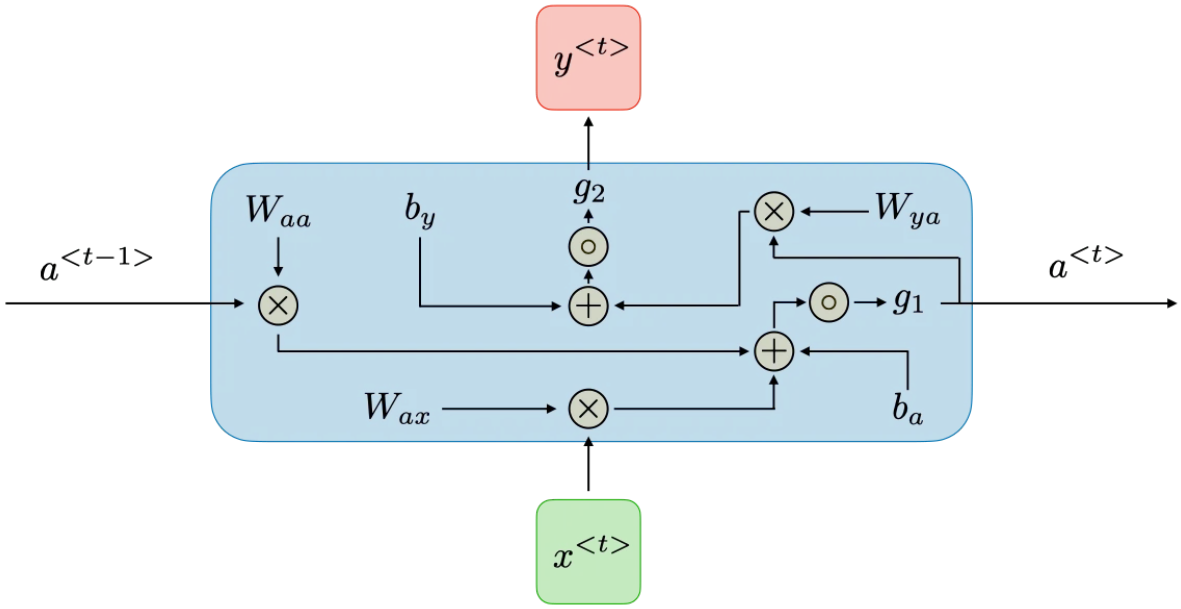
RNNs are a type of neural network with hidden states and allow past outputs to be used as inputs. They usually go like this:



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.



RNN architecture can vary depending on the problem you're trying to solve. It can range from those with a single input and output to those with many (with variations between).

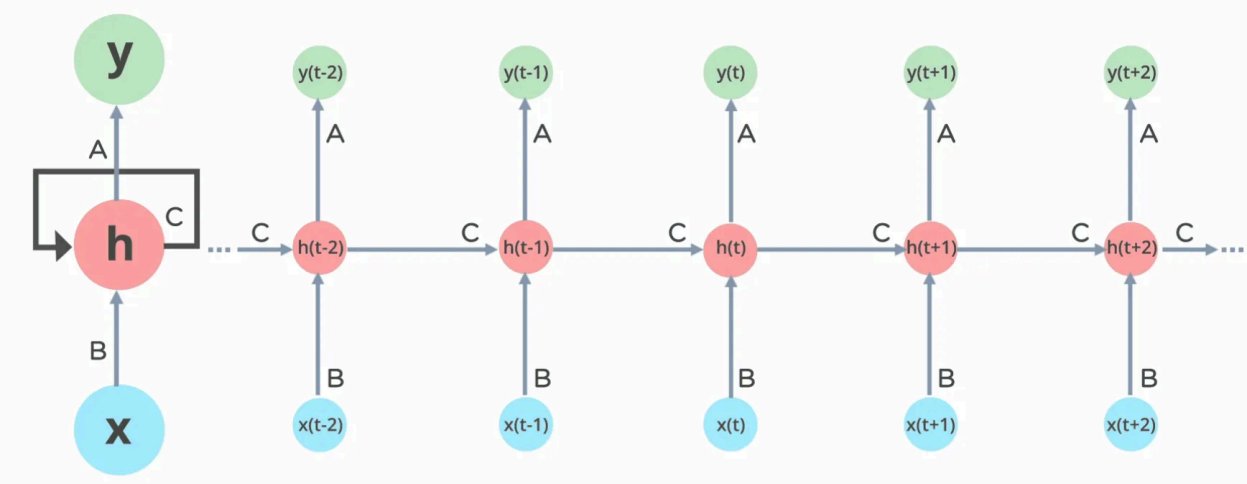
Below are some RNN architectures that can help you better understand this.

- **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 music production, for example.

- **Many-to-one:** A single output combines inputs from distinct time steps in this scenario. Sentiment analysis and emotion identification use such networks, in which a sequence of words determines the class label.
- **Many-to-Many:** For many-to-many, there are numerous options. Input and output are sequences of potentially different lengths. Machine translation systems, such as English to French or vice versa translation systems, use many-to-many networks.

How do Recurrent Neural Networks Work?

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



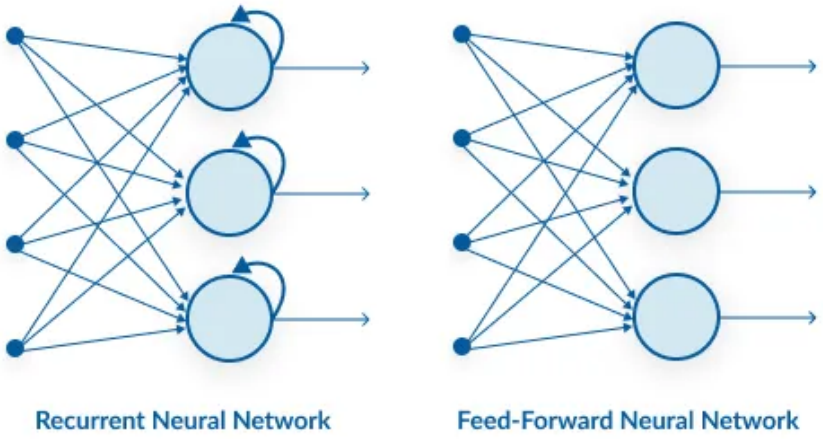
The input layer **x** receives and processes the neural network's input before passing it on to the middle layer.

In the middle layer h, multiple hidden layers can be found, each with its activation functions, weights, and biases. There's a single hidden layer that **repeats over time steps**. The same weights are used across all steps. The hidden state is updated recursively using the current input and the previous hidden state.

The recurrent neural network will standardize the different activation functions, weights, and biases, 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.

Recurrent Neural Network vs Feed-forward Neural Network



Feature	Feed-forward Neural Network (FNN)	Recurrent Neural Network (RNN)
Data Flow	Straight through (no loops)	Loops through time (with memory)
Memory	No memory of past inputs	Maintains memory via hidden states
Input Assumption	Inputs are independent	Inputs are sequentially dependent
Architecture	Static layers, no time dimension	Unfolds across time steps
Training	Standard backpropagation	Backpropagation Through Time (BPTT)

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.

In a normal RNN, a single input is sent into the network at a time, and a single output is obtained. On the other hand, backpropagation 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 simultaneously.

Once the neural network has trained on a set and given you an output, its output is used to calculate and collect the errors. 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.

About its inputs, a gradient is a partial derivative. If you're unsure 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, and 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 about 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 the Different Variations of RNN?

Researchers have introduced new, advanced RNN architectures to overcome issues like vanishing and exploding gradient descent that hinder learning in long sequences.

- **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:** 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:** RNNs translate languages more accurately, like Google Translate by analysing sentence structure and context.
- **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 generate 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:** RNNs can analyze sentiment in social media posts, reviews, or surveys by understanding the context and flow of text.
- **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.

Advantages and Disadvantages of RNNs

Advantages of RNNs	Disadvantages of RNNs
Handle sequential data effectively, including text, speech, and time series.	Prone to vanishing and exploding gradient problems, hindering learning.
Process inputs of any length, unlike feedforward neural networks.	Training can be challenging, especially for long sequences.
Share weights across time steps, enhancing training efficiency.	Computationally slower than other neural network architectures.

Basic Python Implementation (RNN with Keras)

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.

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


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 powerful and versatile tools with a wide range of applications. They are commonly used in language modeling, text

generation, and 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!

This article was published as part of the Data Science Blogathon.

A graduate in Computer Science and Engineering from Tezpur Central University. Currently, I am pursuing my M.Tech in Computer Science and Engineering in the Department of CSE at NIT Durgapur. I expect to Postgraduate in the spring, 2022. A Grounded and Solution-oriented Computer Engineer with a wide variety of experiences. Adept at motivating self and others. Passionate about programming and educating the next generation of technology users and innovators.