**Introduction**

* What is RNN?
* Type of RNN?
* Architectures of RNN?
* How RNN Learning?
* Math And Algorithm in RNN
* Applications?
* Compare RNN vs CNN

1. What is RNN

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed for processing sequential data, where there are dependencies between elements over time. Unlike traditional neural networks (such as feedforward neural networks), which process input data independently, RNNs maintain a **hidden state** that retains information from previous steps in the sequence, allowing the network to "remember" past inputs.

**Key features of RNN:**

* **Hidden state:** RNNs can retain information from previous steps through a hidden state, which is updated at each time step.
* **Recurrent connections:** RNNs have connections that loop back (recurrent connections), allowing the hidden state at the current time step to depend on the state from the previous time step.
* **Sequential data processing:** RNNs are ideal for sequential data such as time series, text, audio, or video, where the relationship between previous and subsequent elements is crucial.

A diagram of a diagram of a diagram

Description automatically generated

1. Architectures of RNN

* **Standard RNN**: The simplest form of RNN, where the network has a single hidden state that loops back to itself over time. At each time step, the hidden state is updated based on the current input and the previous hidden state. Simple sequential tasks like basic time series prediction or language modeling (short sequences).
* **LSTM** : LSTM is an advanced RNN type designed to address the vanishing gradient problem in Vanilla RNNs. It uses special gating mechanisms (input gate, forget gate, and output gate) to control the flow of information and decide what to remember or forget over time. Language translation, speech recognition, long-term time series forecasting, and video analysis.
* **GRU** : GRU is a simplified version of LSTM that also addresses the vanishing gradient problem. It combines the input and forget gates into a single update gate, making it computationally faster than LSTM while still capable of learning long-term dependencies. Tasks similar to those suited for LSTM, such as language modeling, speech recognition, and time series prediction.
* **Bidiretional RNN** : In a bidirectional RNN, the model processes the input sequence in both forward and backward directions. This allows the network to capture information from both the past and the future contexts at each time step. Named entity recognition (NER), machine translation, speech recognition, and other natural language processing tasks.
* **Deep RNN** : A deep RNN has multiple hidden layers between the input and output, allowing the network to learn more abstract and complex representations of the data. This leads to better performance on more complex tasks. Advanced natural language processing, speech recognition, video analysis, and other tasks involving hierarchical structures of information.

1. Types of RNN

* **One To One:** The above diagram represents the structure of the Vanilla Neural Network.  It is used to solve general machine learning problems that have only one input and output.

A diagram of a single output

Description automatically generated

Example **Classification of Images, Simple Text, Simple Regression.**

* **One To Many:** A single input and several outputs describe a one-to-many  Recurrent Neural Network.

A diagram of multiples

Description automatically generatedExample **Text Generation, Text to Speech.**

* **Many To One:** This RNN creates a single output from the given series of inputs.

A diagram of a single or multiplying

Description automatically generated

Example **Speech Emotion Recognition, Text Summarization.**

* **Many To Many:** This RNN receives a set of inputs and produces a set of outputs. There are two main types of Many-to-Many models in RNN: Input and output have the **same lengths and different lenghts**
* Same lengths: Each element in the input sequence corresponds to an element in the output sequence. This is a common form of Many-to-Many model in tasks where each step of input has a corresponding output.
* Different lenghts: The input and output sequences can have different lengths. This is particularly important in tasks like machine translation, where the length of the source sentence and the translated sentence may differ.

1. How RNN learning and Algorithms

Recurrent Neural Networks (RNNs) learn by processing sequential data step-by-step and updating their internal memory (hidden state) to capture information from previous time steps. The learning process of RNNs involves three main stages: **forward propagation**, **backpropagation through time (BPTT)**, and **gradient descent**. Here’s how RNNs learn in detail:

* Forward propagation:
* Input processing: RNNs process sequences by taking one input at each time step (e.g., a word in a sentence, a data point in a time series). The network updates its hidden state at each step based on the current input and the previous hidden state.

**ht​=f(W[hh]​⋅h[t−1]​+W[xh]​⋅xt​)**

ht​ is the hidden state at time t,

ht−1 ​ is the previous hidden state,

xt​ is the input at time t,

W[hh]​ and W[xh]​ are weight matrices, and

f is the activation function (commonly tanh or ReLU).

* Output generation: The output at each time step is calculated using the hidden state:

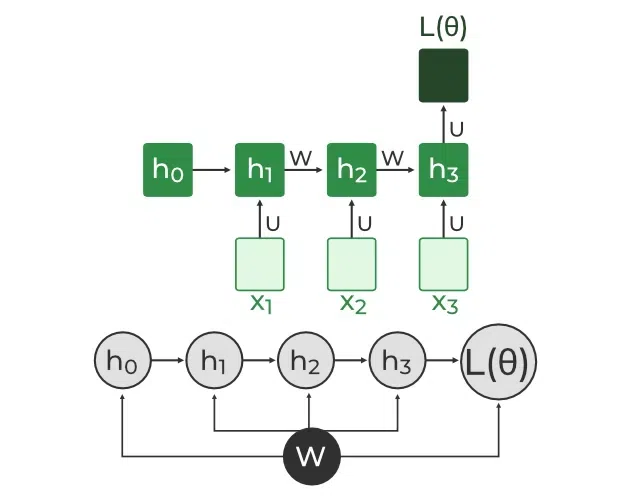
**yt​=W[hy]​⋅ht​**

yt​ is the output at time t,

W[hy]​ is the weight matrix for the output.

* Backpropagation
* **Objective**: The goal of learning in RNNs is to minimize the error between the predicted output and the actual target. This is done by calculating the **loss function** (e.g., cross-entropy for classification tasks or mean squared error for regression tasks) and updating the network’s weights using gradient descent.
* **Backpropagating the Error**: In RNNs, the error is propagated not only through the current time step but also **back in time** to previous time steps. This process is called **Backpropagation Through Time (BPTT)**.
* **Gradients with Respect to Weights**: In BPTT, the gradients are calculated for the weight matrices by computing how the loss at the current step depends on the weights and the hidden states from previous steps.

**∂L(θ)​/ ∂W = (T, t=1)∑​ ​(∂L(θ)​/ ∂W)**



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.

* Gradient descent
* **Weight Update**: Once the gradients are calculated through BPTT, the network updates its weights using gradient descent (or a variant like **stochastic gradient descent** or **Adam**). This is done to reduce the error or loss:

W=W−η∂L/∂W

W is the weight matrix,

η is the learning rate, and

∂L/∂W is the gradient of the loss with respect to the weight. ​

1. Math in RNN

* Activation Function

Tanh : f(x)=tanh(x)= (e^x – e^(-x) ) / (e^x + e^(-x) )

ReLU: f(x)= max(0, x)

SoftMax (output): g(xi) = (e^xi) / [(C, j=1) ∑e^xj]

* Lost Function : L(yt, yt’​)=−(C, c=1)∑( (yt)^c log(yt’^x​)

y: real value

y’: predicted value

1. Compare

|  |  |  |
| --- | --- | --- |
|  | RNN | CNN |
| **Data Handling** | designed to handle sequential data, such as time series, text, or audio, where the order of inputs matters. RNNs can maintain information from previous steps (via hidden states) and make predictions based on the sequence. | on the other hand, are primarily designed for spatial data like images or videos. CNNs are adept at recognizing patterns and structures (such as edges, shapes) in grid-like data. They process data in parallel across different regions of the input. |
| **Architecture** | RNNs have a recurrent architecture where the output of one step is fed back into the network as input for the next step. This makes RNNs suitable for tasks where data dependencies over time are important, but it can also lead to issues like vanishing gradients. | CNNs use convolutional layers to apply filters to input data, detecting spatial hierarchies in the input. They are structured with layers of convolutions, pooling, and fully connected layers. CNNs excel at capturing local spatial features, but they are not designed to handle sequential dependencies across time. |

1. Applications

* **Natural Language Processing (NLP)**
* **Language Modeling**: RNNs predict the next word in a sentence based on the previous words, used in systems like autocomplete or text generation.
* **Text Generation**: RNNs generate coherent text sequences, given an initial input, which is useful in creative writing and dialogue systems (chatbots).
* **Machine Translation**: RNNs, especially in Seq2Seq architectures, translate text from one language to another (e.g., English to French).
* **Speech Recognition and Synthesis**
* **Speech-to-Text Conversion**: RNNs are used to recognize speech and convert it into written text.
* **Text-to-Speech Convertion:** RNNs can generate human-like speech from text input, improving voice assistants and accessibility tools.
* **Time-Series Analysis and Forecasting**
* **Weather Forecasting**: RNNs can analyze past weather data to make predictions about future conditions.
* **Demand Forecasting**: Businesses use RNNs to predict product demand in different time periods, helping optimize inventory management and resource allocation.
* **Music Generation:** RNNs can generate music sequences, learning patterns from existing music compositions and producing new compositions.
* **Video Analysis and Action Recognition**
* **Video Captioning**: RNNs analyze video frames to generate descriptive text that explains the content of the video.
* **Action Recognition**: RNNs identify specific actions (e.g., running, jumping) in video sequences by learning the temporal patterns in the movement.
* **Video Classification**: RNNs classify entire video clips into categories, such as sports, music, or news, based on the content.
* **Robotics and Autonomous Systems**
* **Robot Navigation**: RNNs help robots learn paths or recognize environments based on past sensor inputs.
* **Autonomous Vehicles**: RNNs assist in processing sensor data to help autonomous vehicles make decisions based on the sequence of events during driving.
* **Language Generation and Dialogue Systems**
* **Chatbots:** RNNs power chatbots and virtual assistants by generating coherent responses in conversations.
* **Storytelling:**RNNs can create stories or narratives based on input prompts.