RNN

RECURRENT NEURAL NETWORK

CONTENTS

INTRODUCTION TO RNN

A Recurrent Neural Network (RNN) is a type of artificial neural network designed for sequential data processing.

Unlike traditional neural networks, RNNs have connections that form a loop, allowing them to maintain a hidden state or memory.

This memory enables RNNs to capture dependencies between elements in a sequence, making them well-suited for tasks such as natural language processing, time series analysis, and speech recognition.

RNNs have widespread applications in various domains due to their capacity to handle sequential data.

SEQUENTIAL DATA PROCESSING

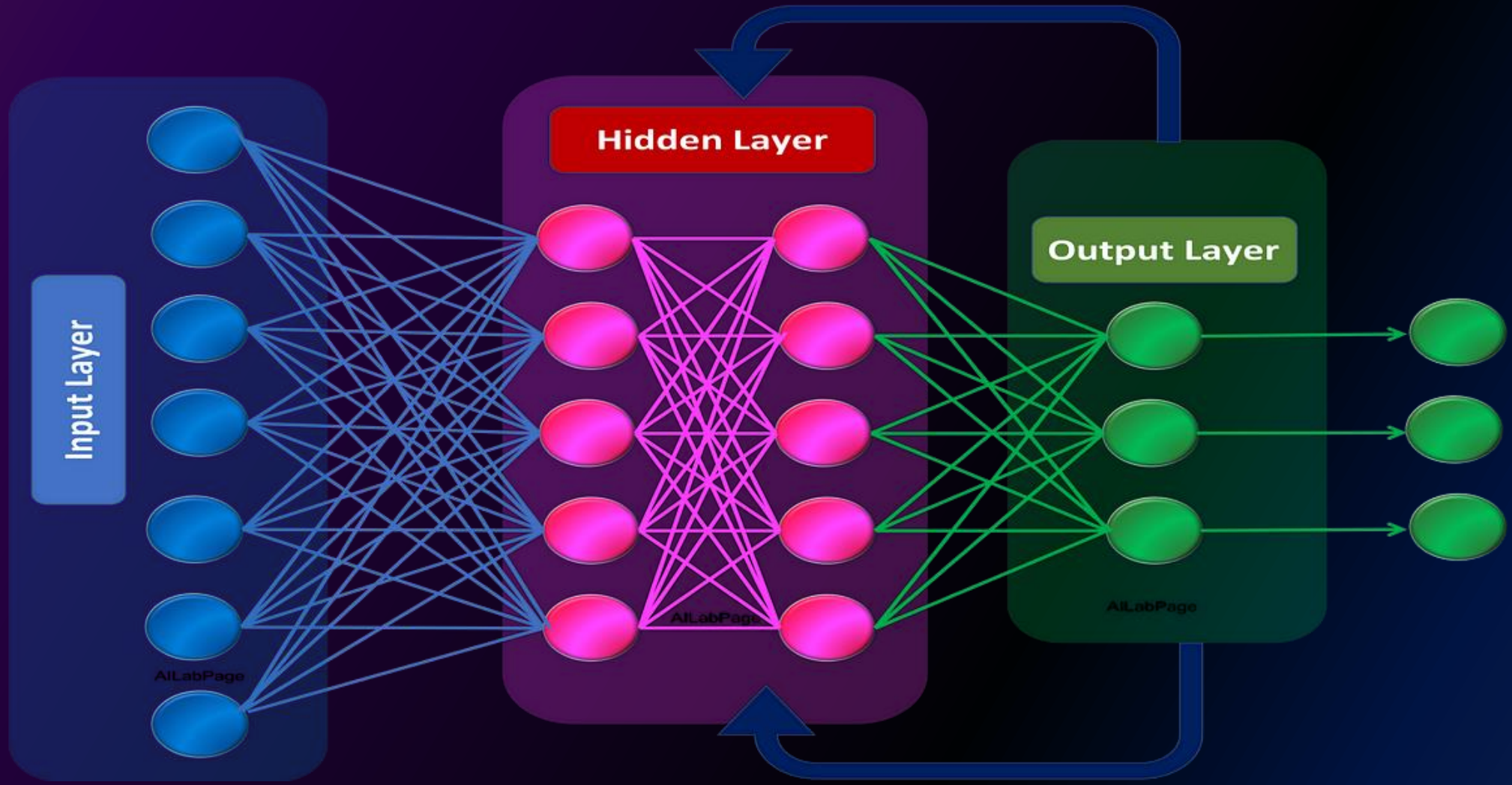
Sequential data processing refers to the handling and analysis of data that has an inherent order or sequence.

This type of data is often encountered in various fields, including natural language, time series analysis, speech recognition, and more.

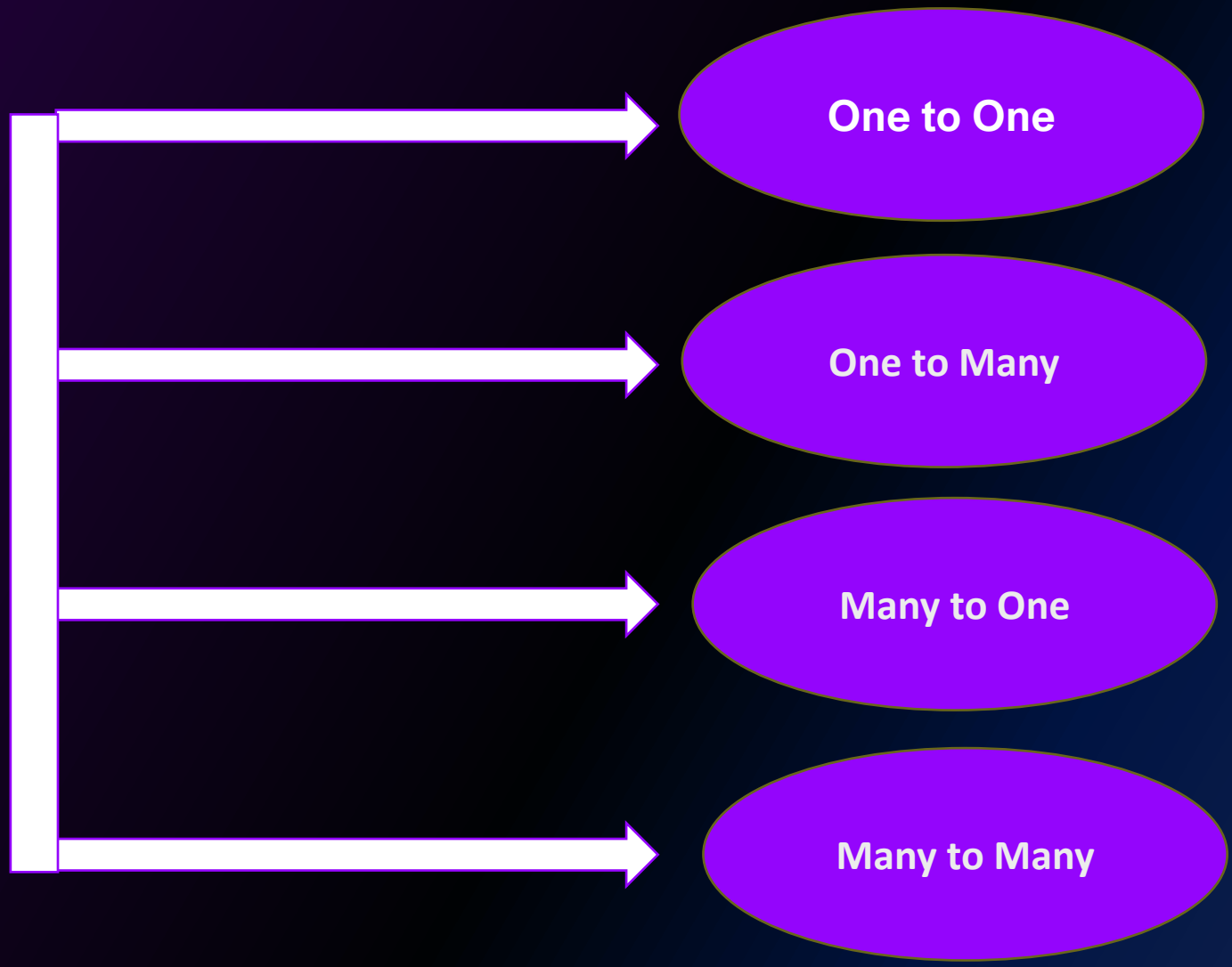
Understanding sequential data processing is crucial for tasks where the context or relationship between elements in the sequence matters.

Recurrent Neural Networks (RNNs) are particularly well-suited for such tasks.

RECURRENT NEURAL NETWORKS



TYPES OF RNN



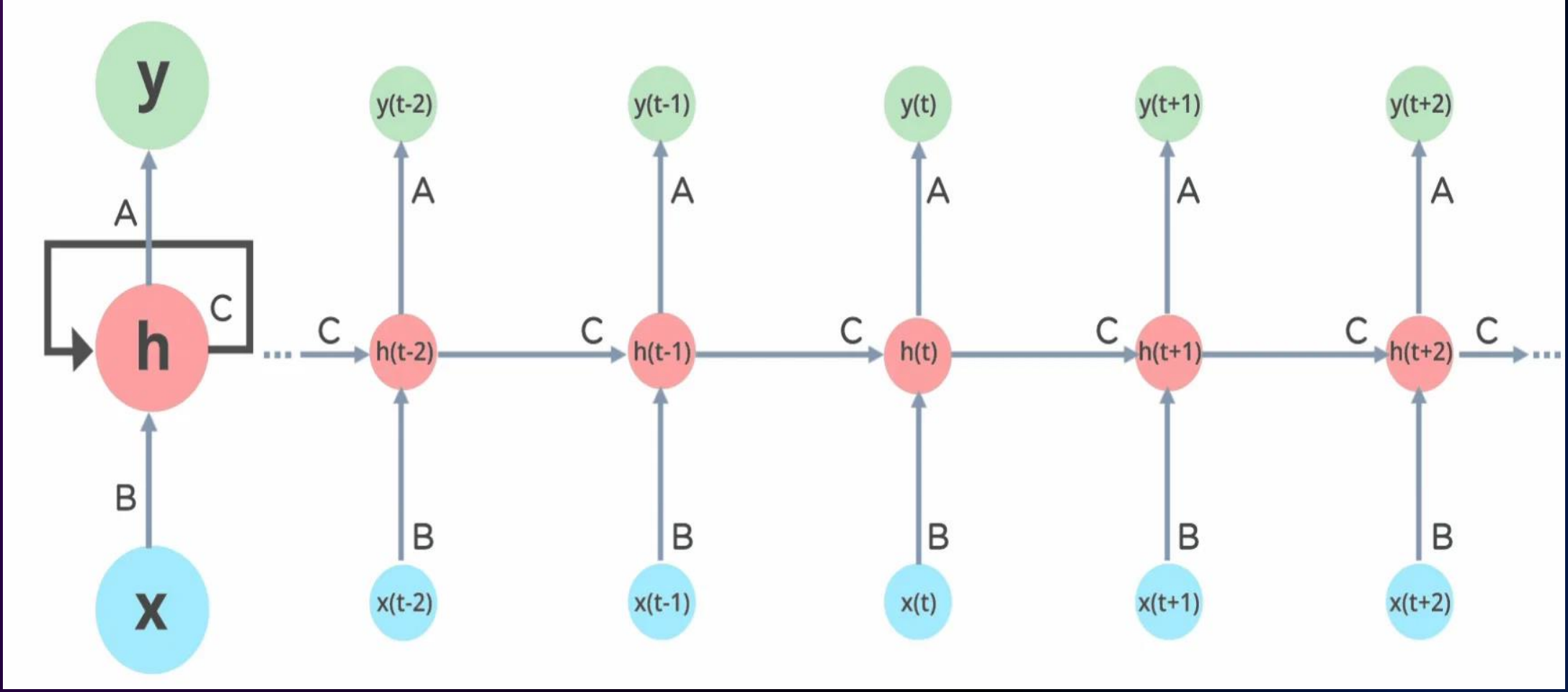
1.One to One: This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.

2.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.

3.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.

4.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.

WORKING OF RNN



1.Input Processing: At each time step x_t, the RNN receives an input x_t. This input can be a single data point or a sequence of data points, depending on the task. The input is typically represented as a vector or a sequence of vectors.

2.Hidden State Update: The input x_t is combined with the previous hidden state h_{t-1} to compute a new hidden state h_t using a set of weights W and biases b. This process can be mathematically represented as: $h_t = \sigma(Wx_t + Uh_{t-1} + b)$ where σ is an activation function, such as the hyperbolic tangent (tanh) or the rectified linear unit (ReLU).

3.Output Computation: The hidden state h_t is then used to compute an output y_t at the current time step. This output can represent predictions, classifications, or any other relevant information derived from the input sequence. The output is computed using another set of weights and biases: $y_t = \text{softmax}(Vh_t + c)$ where softmax is often used to convert the output into a probability distribution over different classes or categories.

4.Recurrent Connections: The key characteristic of an RNN is its recurrent connections, which allow information to flow from one time step to the next. The hidden state h_t serves as a form of memory that captures information from previous time steps, enabling the network to model temporal dependencies in sequential data.

5.Training: During training, the parameters (weights and biases) of the RNN are learned from labeled data using optimization algorithms such as gradient descent. The goal is to minimize a loss function that measures the difference between the predicted output y_t and the actual target for each time step.

6.Backpropagation Through Time (BPTT): To update the parameters of the RNN, gradients are computed using backpropagation through time (BPTT), which is an extension of the standard backpropagation algorithm. BPTT involves computing gradients at each time step and then aggregating them over the entire sequence to update the parameters.

7.Inference: Once the RNN is trained, it can be used for inference on new, unseen data. The input sequence is fed into the network, and the output sequence is computed iteratively using the learned parameters and the recurrent connections.

HARDWARE ARCHITECTURE OF RNN

1.Processing Units: Traditional central processing units (CPUs) are commonly used for training and inference of RNNs. However, due to the parallelizable nature of RNN computations, graphics processing units (GPUs) have become popular for accelerating training and inference tasks, offering significant speedups over CPUs.

2.Specialized Accelerators: To further improve the performance of RNNs, specialized hardware accelerators tailored for deep learning tasks have been developed. These accelerators, such as tensor processing units (TPUs) from Google or neural processing units (NPUs) from various manufacturers, are optimized for the high computational demands of neural network workloads, including RNNs.

3.Memory Hierarchy: RNNs often require large amounts of memory to store parameters, intermediate activations, and gradients during training. Efficient utilization of memory hierarchy, including fast on-chip caches, high-bandwidth memory (HBM), and system memory (RAM), is essential for minimizing data movement and maximizing throughput.

4.Parallelism and Pipelining: Hardware architectures for RNNs leverage parallelism and pipelining techniques to exploit concurrency and improve efficiency. Parallelism can be achieved at various levels, including data parallelism (processing multiple inputs simultaneously) and model parallelism (distributing different parts of the model across multiple processing units).

5.Optimized Libraries and Frameworks: Hardware vendors often provide optimized libraries and frameworks for deep learning, including RNNs, to take advantage of the underlying hardware architecture efficiently. These libraries provide optimized implementations of RNN operations, such as matrix multiplications and activation functions, tailored for specific hardware platforms.

6.FPGA and ASIC Implementations: Field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) provide customizable and highly efficient hardware platforms for implementing RNNs. These platforms offer flexibility in optimizing the hardware architecture for specific RNN models and applications, achieving high performance and energy efficiency.

ISSUES OF STANDARD RNN

1.Vanishing Gradient: As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

2.Exploding Gradient: The exploding gradient problem is a challenge encountered during training deep neural networks. It occurs when the gradients of the network's loss function with respect to the weights (parameters) become excessively large.

ADVANTAGES OF RNN

1.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.

2.Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

DISADVANTAGES OF RNN

1. Gradient vanishing and exploding problems.

2. Training an RNN is a very difficult task.

3. It cannot process very long sequences if using tanh or relu as an activation function.

THANK YOU

Priya Samanta

Sourik Saha