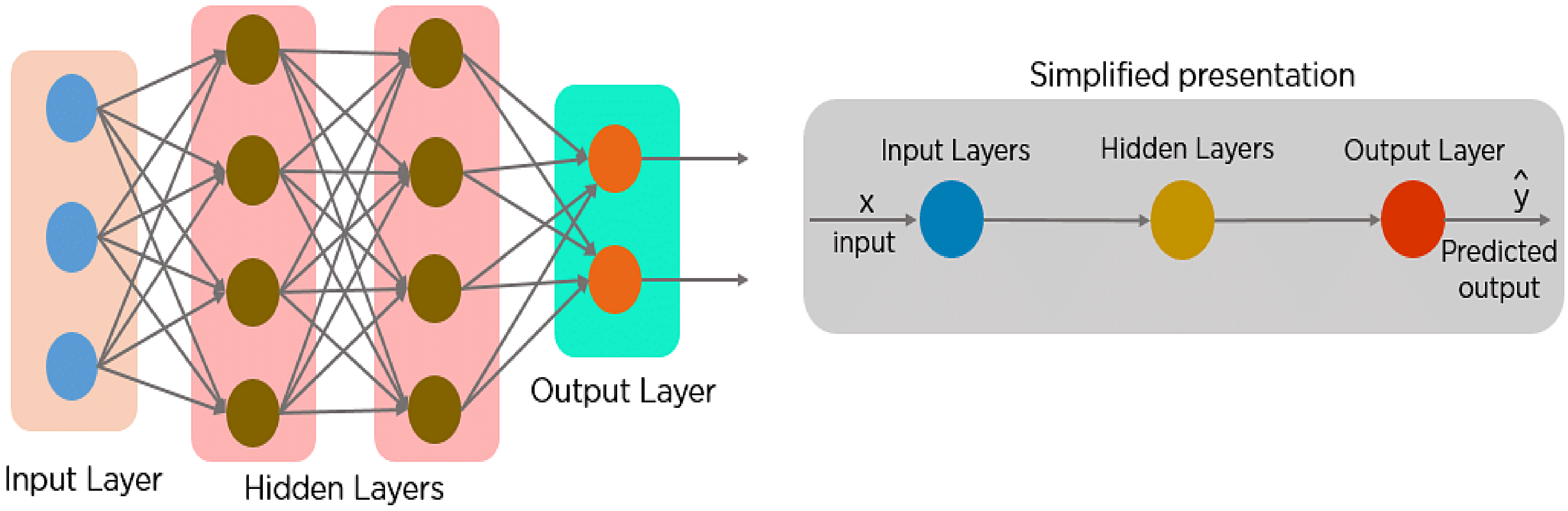
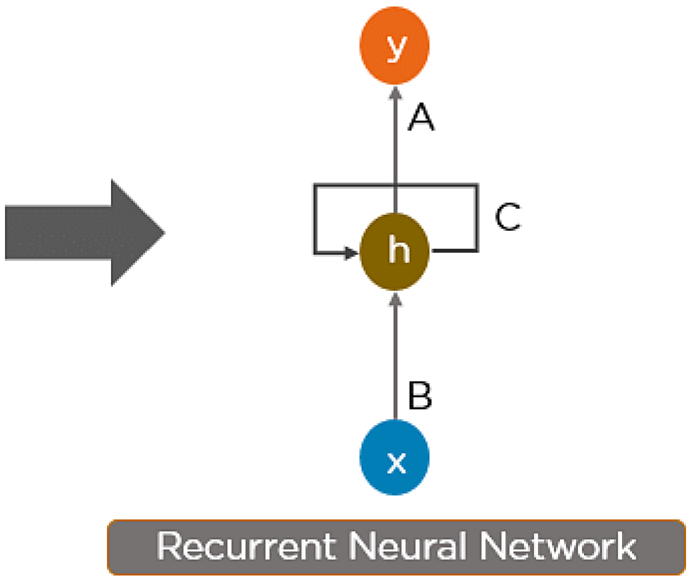
RNN (Recurrent Neural Network)

- RNN (Recurrent Neural Network) is a type of artificial neural network designed for sequential data processing. Unlike traditional feedforward networks, RNNs have loops that allow information to persist across time steps, making them well-suited for tasks like time-series forecasting, natural language processing (NLP), and speech recognition.





Architecture of RNN

- The core idea of RNN is to use the same **weight parameters** at each time step while processing sequential data. The hidden state is updated based on the current input and the previous hidden state.

Mathematical Formulation

- x_t: Input at time step t
- h_t: Hidden state at time step t
- y_t: Output at time step t
- W_{hh}, W_{xh}, W_{hy}: Weight matrices
- b_h, b_y: Bias terms
- f: Activation function (like tanh, ReLU)

The RNN can be expressed mathematically as:

Forward Pass:

$$h_t = f(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h)$$

$$y_t = g(W_{hy} \cdot h_t + b_y)$$

where:

- h_t: Hidden state (memory) at time t
- x_t: Input at time t
- y_t: Output at time t
- f: Non-linear activation function (commonly tanh or ReLU)
- g: Activation function for output (like softmax)

example

Task: Given the sentence **"I love deep learning"**, predict the next word.

1. **Input Sequence**:
    - Convert words into embeddings:

$$x_1 = \text{"I"}, x_2 = \text{"love"}, x_3 = \text{"deep"}, x_4 = \text{"learning"}$$

2. **Hidden State Updates**:
    - $h_1 = f(W_x x_1 + W_h h_0 + b)$
    - $h_2 = f(W_x x_2 + W_h h_1 + b)$
    - $h_3 = f(W_x x_3 + W_h h_2 + b)$
    - $h_4 = f(W_x x_4 + W_h h_3 + b)$

3. **Output Prediction**:
    - Compute probabilities for the next word:

$$y_5 = \text{softmax}(W_y h_4 + b_y)$$

    - If **"models"** has the highest probability, the model predicts: **"I love deep learning models"**.

Example of RNN Processing a Sentence

- **Sentence:** "The cat sat on the ___“

Time Step	Input xtx_txt	Hidden State hth_tht	Output yty_tyt (Prediction)
1	"The"	H1	"cat"
2	"cat"	h2	"sat"
3	"sat"	h3	"on"
4	"on"	h4	"the"
5	"the"	h5	**"mat" (Correct)**

Problem Statement:

- **Predict the future stock price** based on the past 5 days' stock prices using **Recurrent Neural Networks (RNN)**

Day	Stock Price ($)
Day 1	100
Day 2	102
Day 3	104
Day 4	106
Day 5	108
Day 6 (Predict)	?

Input size = 1 (single stock price per day)Hidden size = 2 (two hidden neurons)
Output size = 1 (predicted stock price)
Activation function = Tanh
W x =0.5Wh=0.2Wy=bh=0, by=0,h0=0 (initial hidden state is zero)

How BPTT Works?

- **Unrolling the RNN:**
  - Since an RNN processes sequences, we **unroll** the network across multiple time steps.
  - This allows us to apply standard backpropagation as if it were a deep feedforward network.
- **Forward Pass:**
  - Compute the hidden states ht for each time step t
  - Compute the output yt using the hidden states.
- **Calculate Loss:**
  - Compute the error L between the predicted output yt and the actual output.
- **Backward Pass (Backpropagation Through Time):**
  - Compute gradients of the loss with respect to weights by propagating errors **back in time**.
  - Adjust weights Wx,Wh,Wy using gradient descent.

Problem with RNN: Vanishing Gradient

Why RNN Fails for Long Sequences?

- When training an RNN, we use **Backpropagation Through Time (BPTT)** to update the weights.
- However, as we go back many steps, gradients **become too small (vanish)**.
- This means that **early information (like "The cat") is forgotten** when predicting later words.

Example:
Input: "I grew up in France. I speak fluent ___"
RNN Prediction: "English" (Forgets "France")
Correct Answer: "French“

Reason: "France" was too far back, and the RNN forgot it.

- **Solution:** Use **LSTM (Long Short-Term Memory)**, which **remembers important words** for a long time.

Exploding Gradients

What happens?

- If weights **W_x and W_h** are large (e.g., > 1), their repeated multiplication **amplifies gradients exponentially**.
- The gradient **becomes too large (→ infinity)** and causes **instability** in training.

Why does it happen?

- The **derivatives** of some activation functions (like **ReLU**) can be **large** in certain cases.
- Long sequences cause the multiplication of large values, leading to **overflow** in computations.

Effects:
Loss function fluctuates wildly
Weights update too aggressively, leading to poor convergence

- **Fixes:**
  ✓ Use **Gradient Clipping** (limit max gradient value)
  ✓ Use **Layer Normalization or Batch Normalization**
  ✓ Reduce **learning rate**