Vanishing Gradient Problem

Module Name : Machine Learning And Neural Networks
Assignment: Machine Learning Tutorial

Prepared by:
Student Name : Masud Rana
Student ID : 23074889

What is Vanishing Gradient Problem?

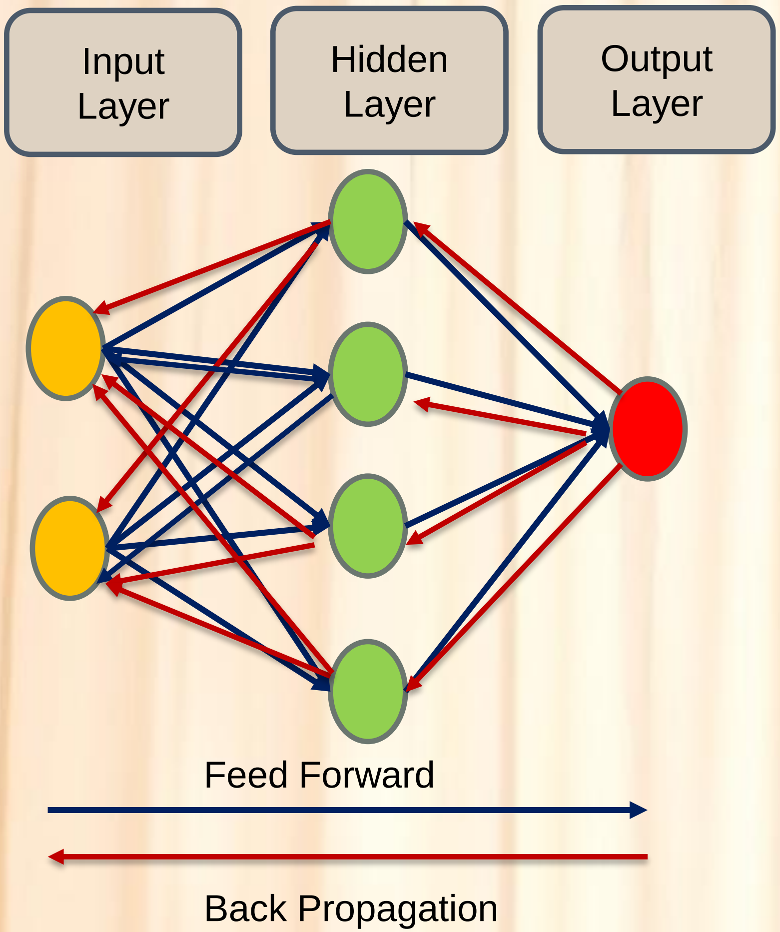
- Imagine I read a book
- Trying to remember first sentence after whole book
- Hardly remember as human memory fades over time
- RNNs also struggles to retain information from earlier time steps due to the **vanishing gradient problem**

What is Vanishing Gradient Problem?

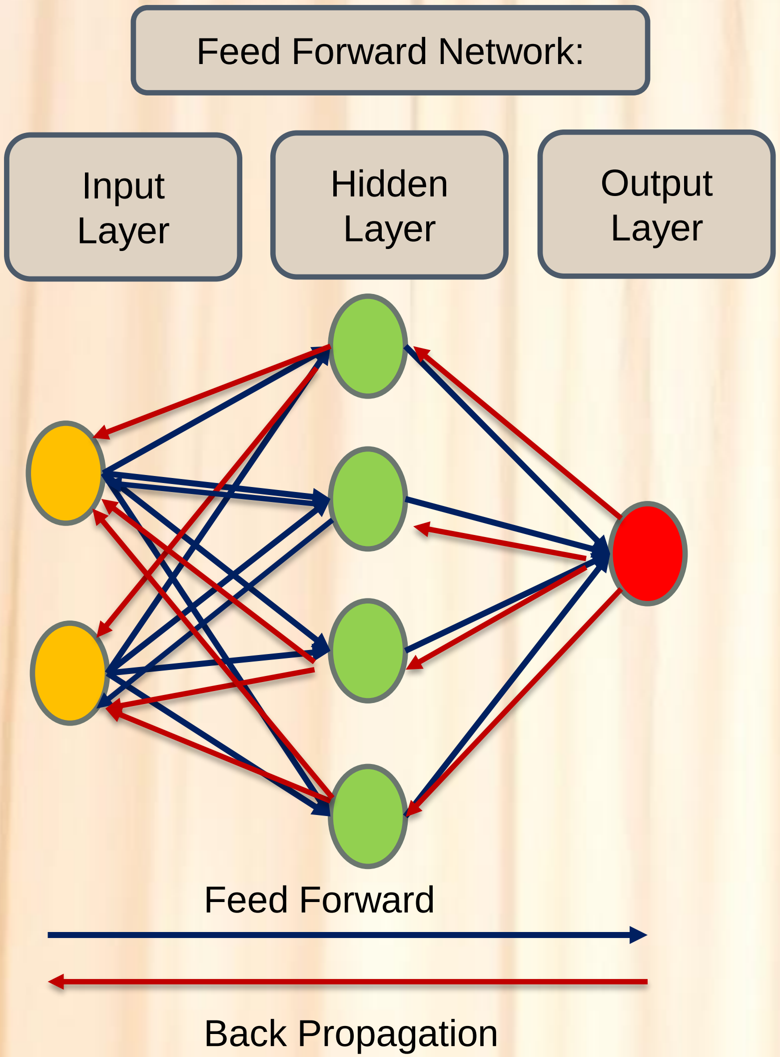
- RNNs are widely used for **sequence-based** tasks
- Vanishing gradient problem occurs during **training** period of RNNs
- Prevents RNNs from learning **long term dependencies**
- Vanishing gradient problem occurs when the **gradients** used to update **weights** during **back propagation** become exceedingly small
- Almost no updates from earlier layers & leads to slow learning
- Hence, RNNs struggle to capture long-term dependencies in sequential data

Training Neural Network

Feed Forward Network:



Training Neural Network



Forward Propagation:

- Compute the output of the neural network using the given inputs.

Assuming a simple neural network with:

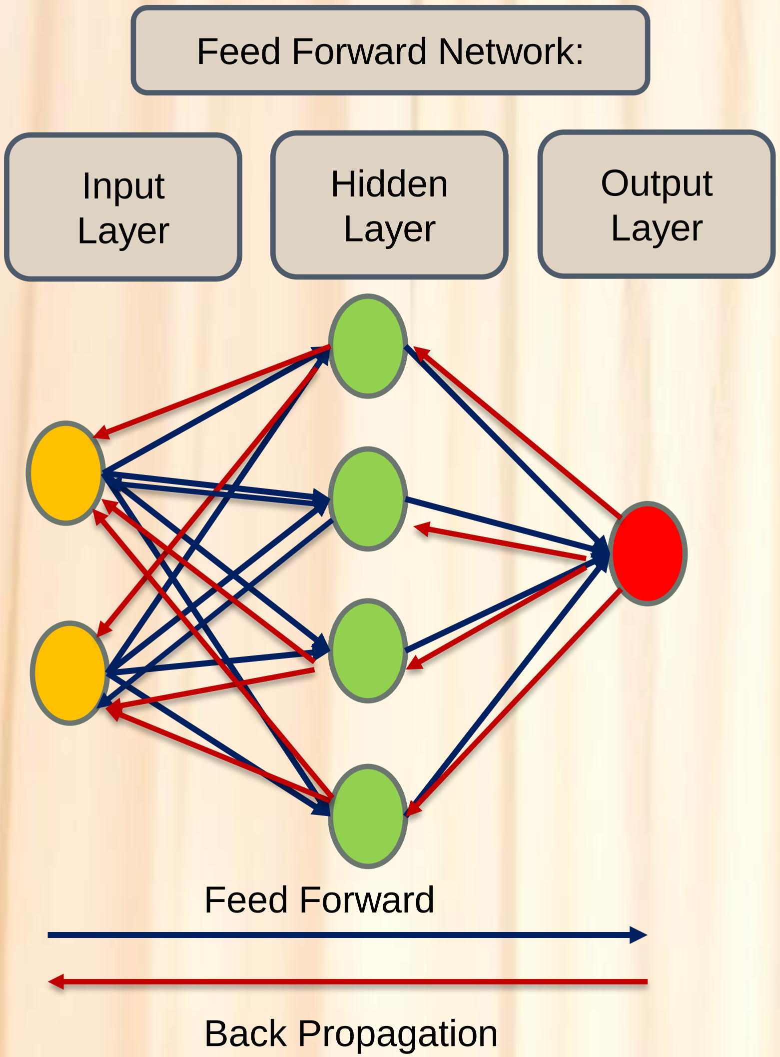
- Input $\boldsymbol{x}$, weights $\boldsymbol{w}$, bias $\boldsymbol{b}$, activation function $\boldsymbol{f}$, and output $\boldsymbol{y}$^ .
- Loss function $\boldsymbol{L}$.

Forward pass Equation:

$$z = w \cdot x + b$$

$$y\text{\^{}} = f(z)$$

Training Neural Network



Compute Loss:

- Compare the predicted output $\boldsymbol{y\hat{}}$ with the actual target $\boldsymbol{y}_{\textbf{true}}$ using a loss function.

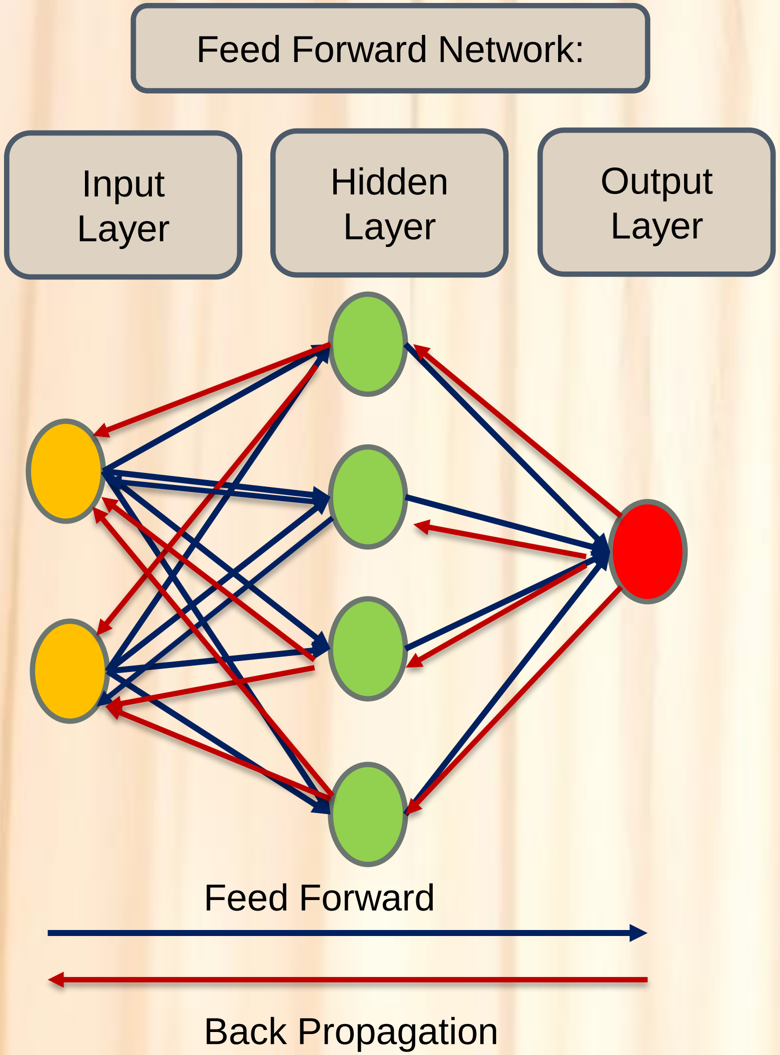
Assuming a simple neural network with:

- Loss function $\boldsymbol{L}$.
- **N** is the number of training samples.
- $\boldsymbol{y}_{\textbf{true}}$ is the actual output for the $\boldsymbol{i}_{\textbf{th}}$ sample.
- $\boldsymbol{y\hat{}i}$ is the predicted output for the $\boldsymbol{i}_{\textbf{th}}$ sample.

Equation for Loss Function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{true},i} - \hat{y}_i)^2$$

Training Neural Network



Backward Propagation (Compute Gradients):

- Compute the **gradient** of the **loss function** with respect to each **weight** using the chain rule.

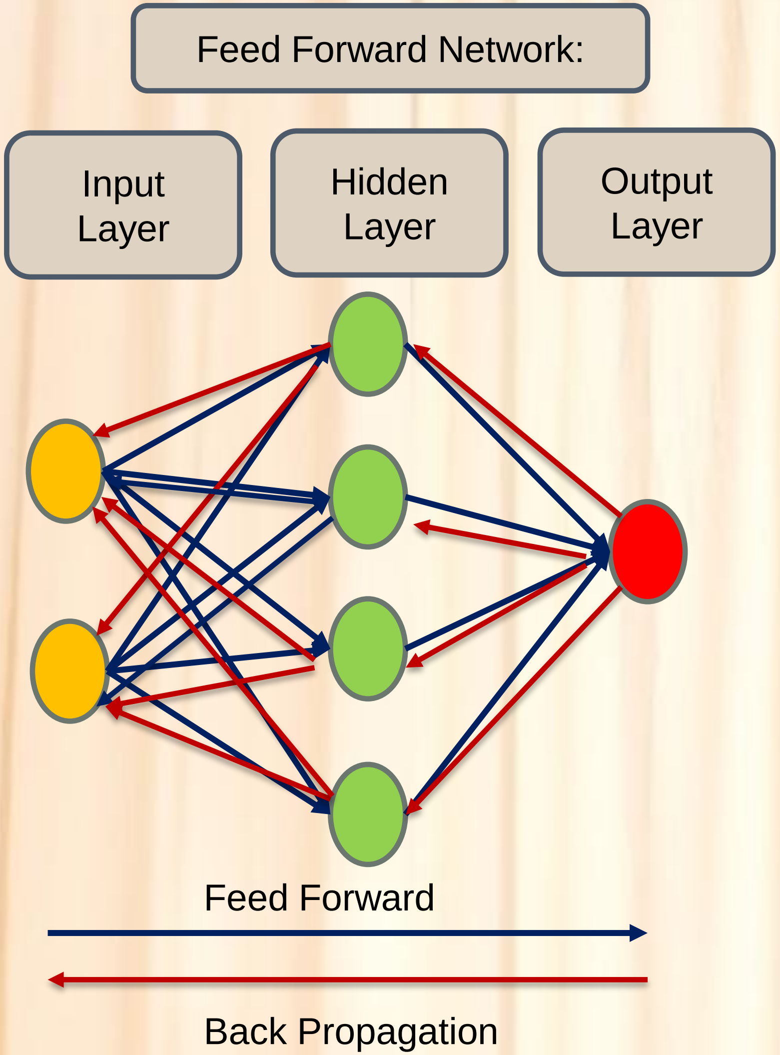
Gradient (Using Chain Rule):

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Example:

Gradient = 0.03 * 0.05
= 0.0015

Training Neural Network



Update Weights (Gradient Descent Step):

- Adjust the weights using **gradient descent** to minimize loss.

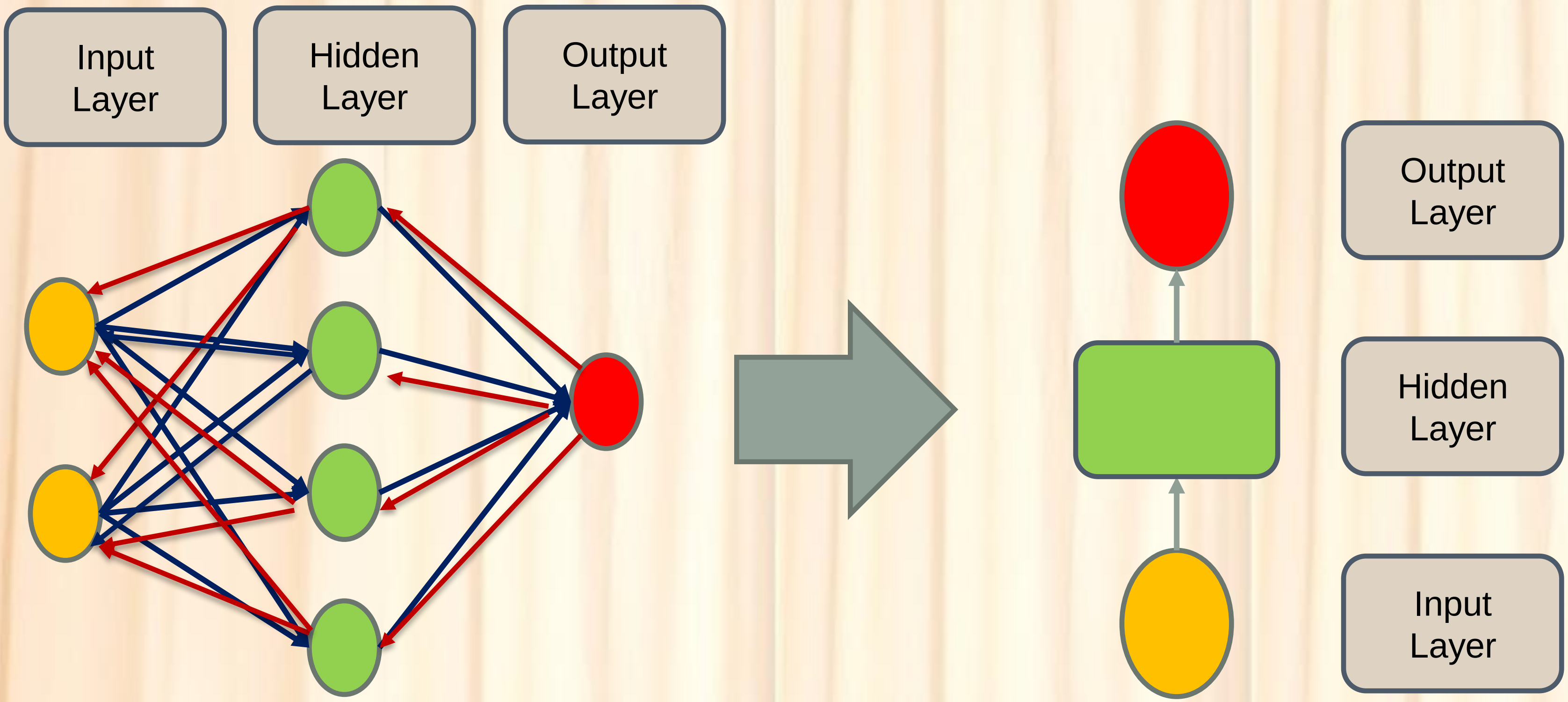
Parameter Adjustment Equation:

$$w = w - \eta \frac{\partial L}{\partial w}$$

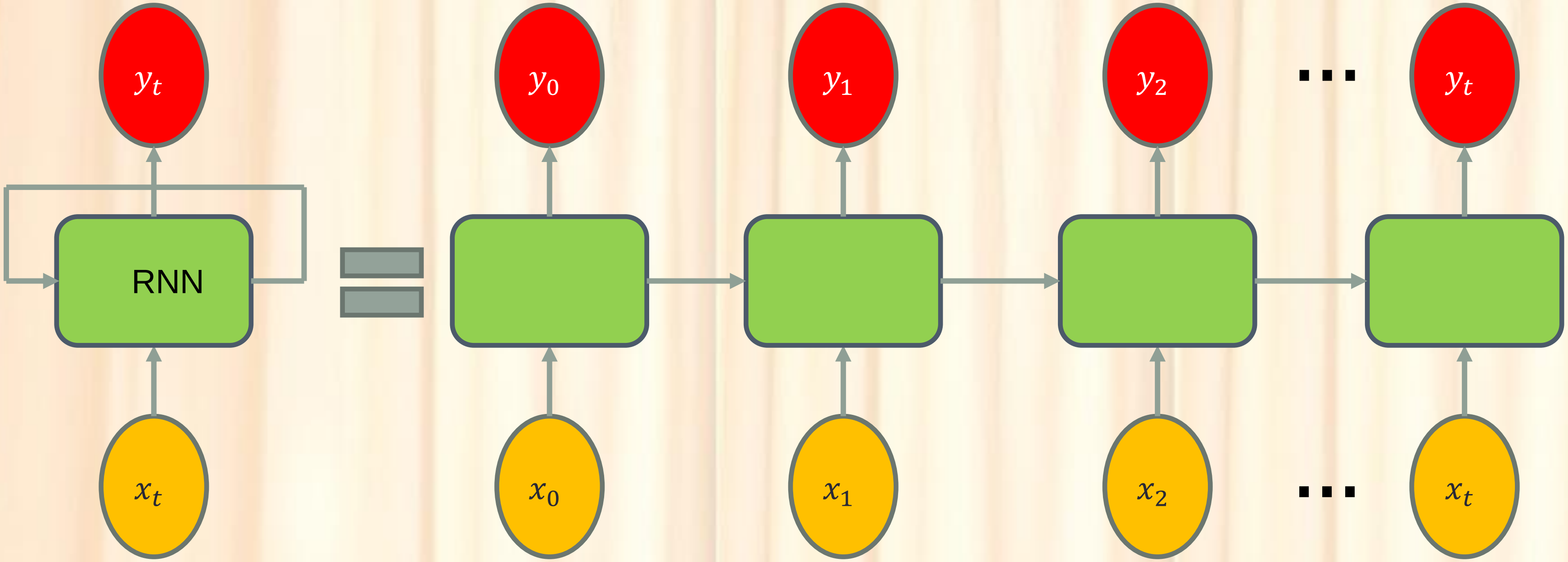
$$b = b - \eta \frac{\partial L}{\partial b}$$

where η is the **learning rate**.(Ex.: 0.0001)

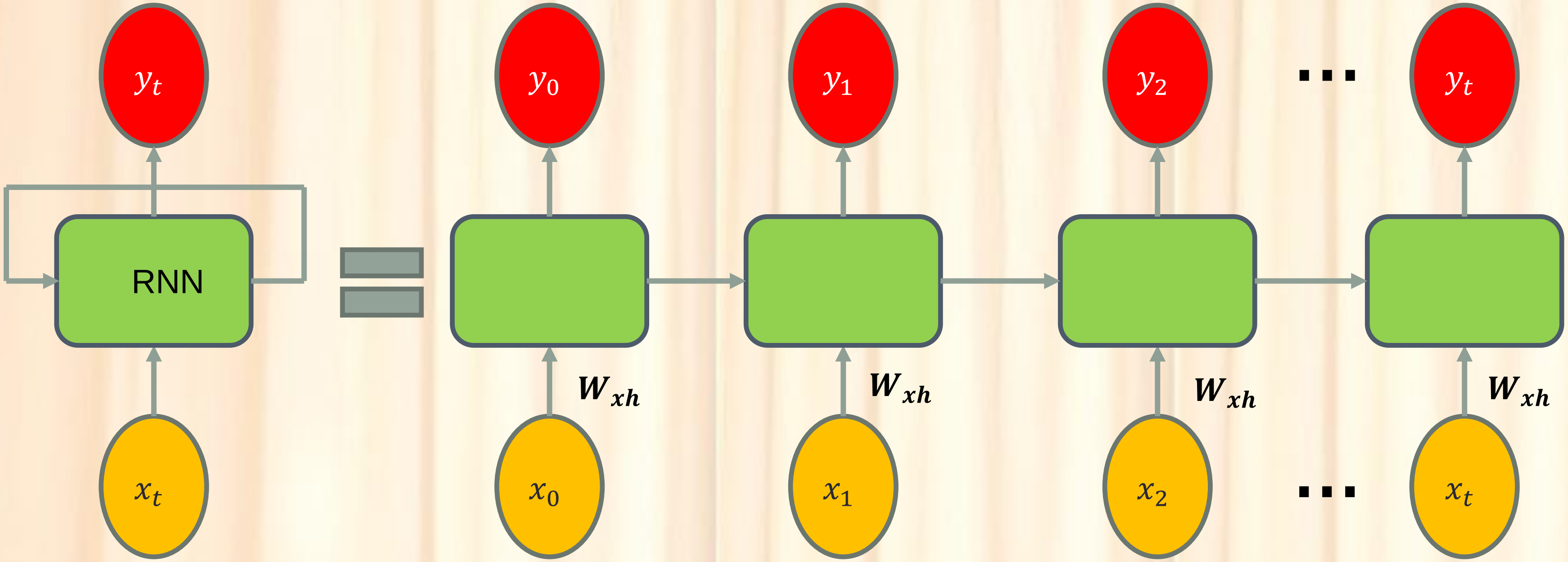
How RNN Works



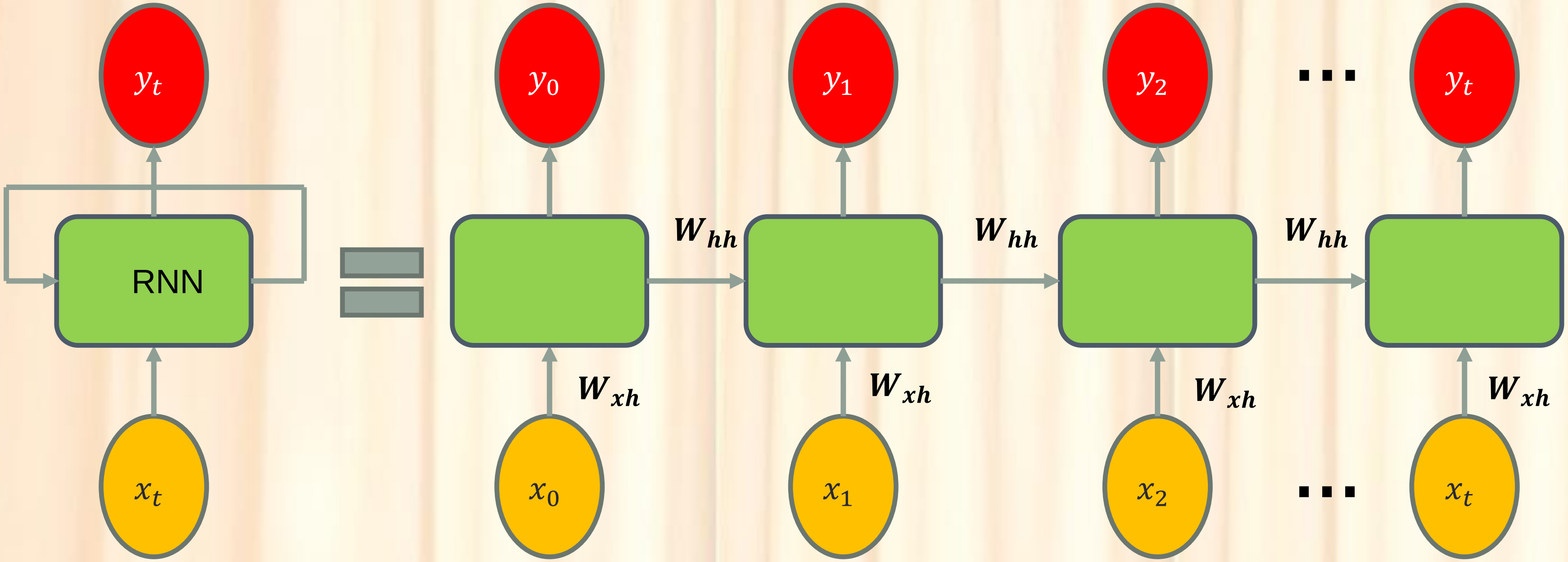
How RNN Works



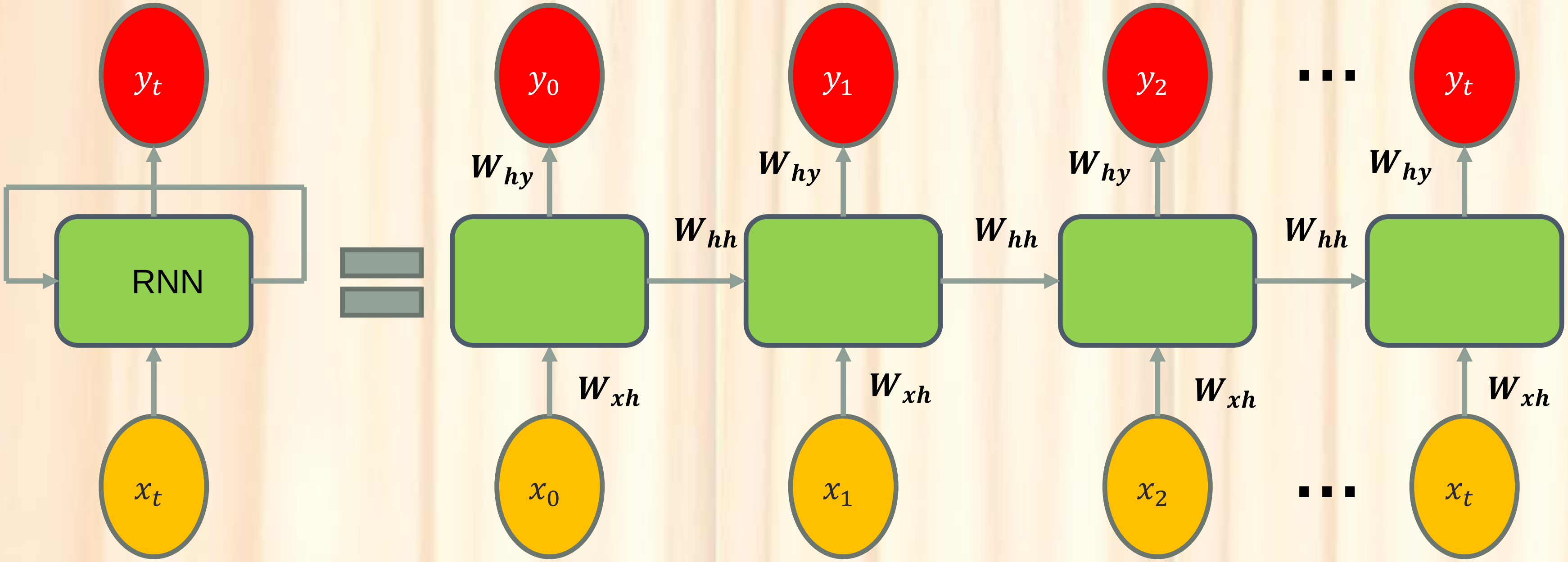
How RNN Works



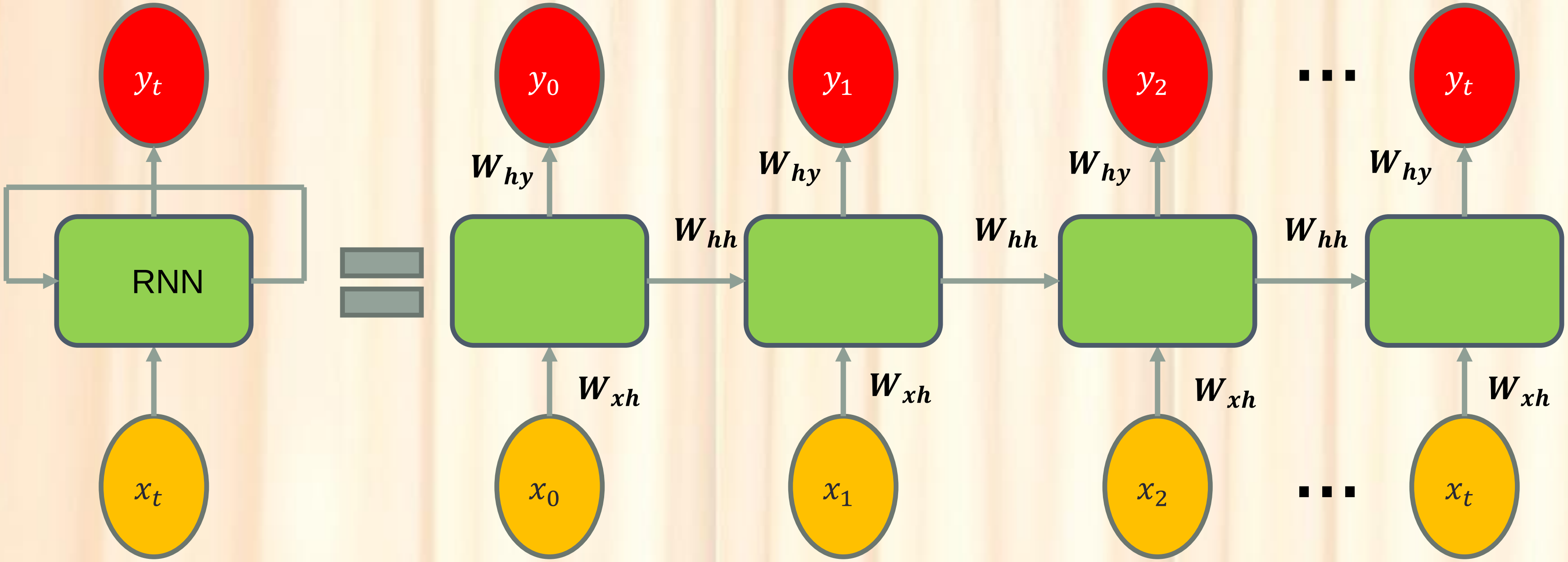
How RNN Works



How RNN Works

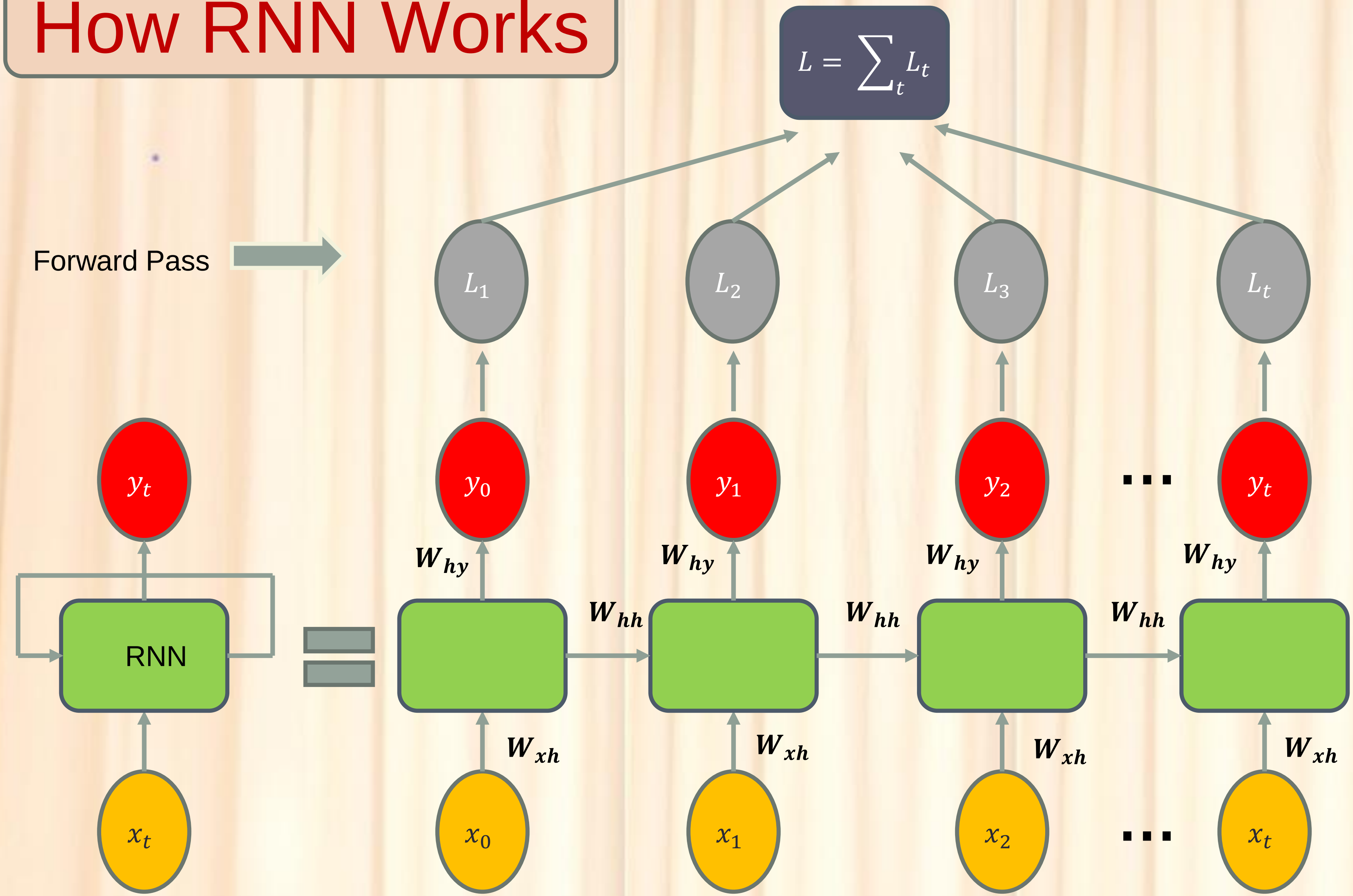


How RNN Works

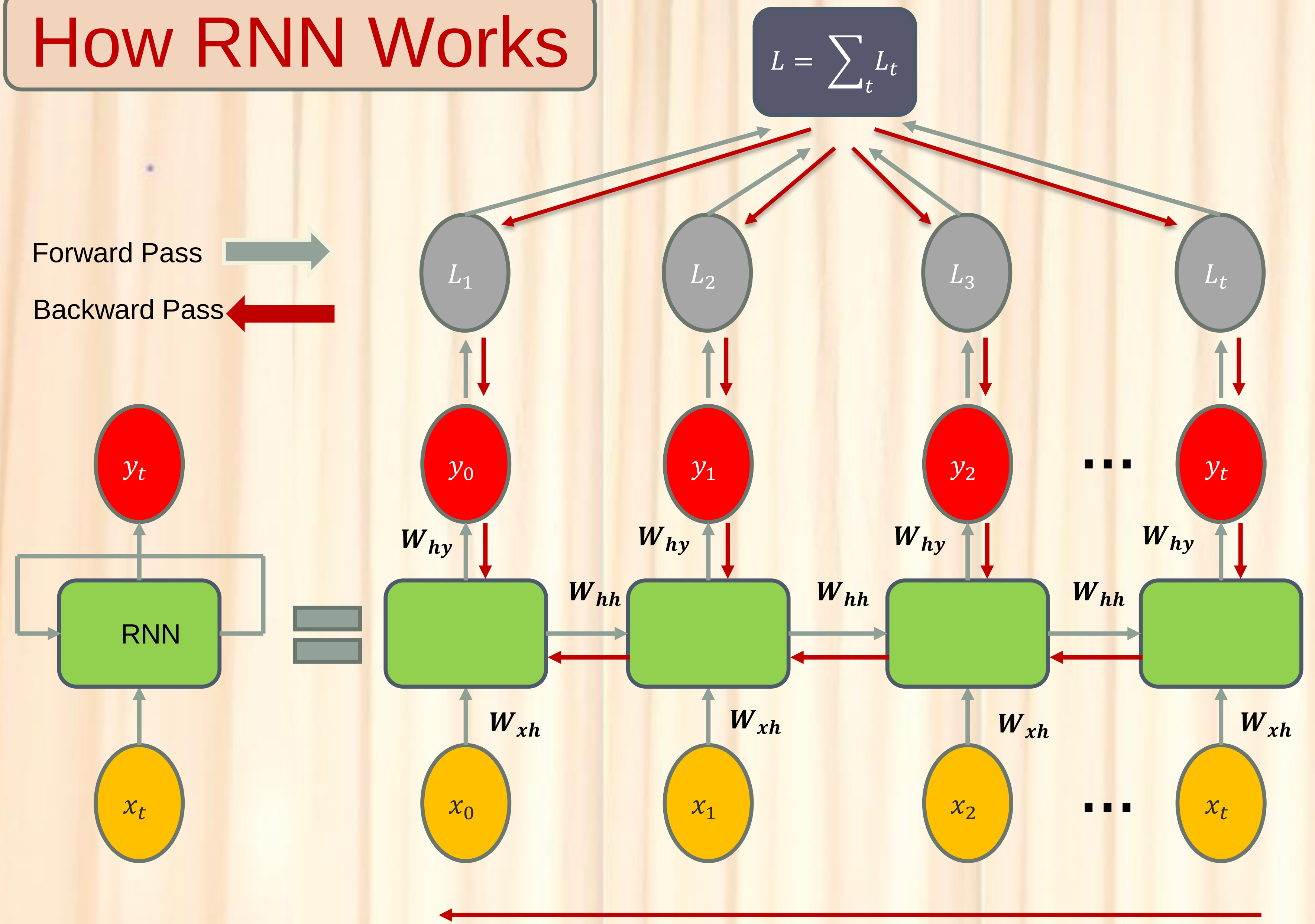


Reuse the same weight matrices at each time step

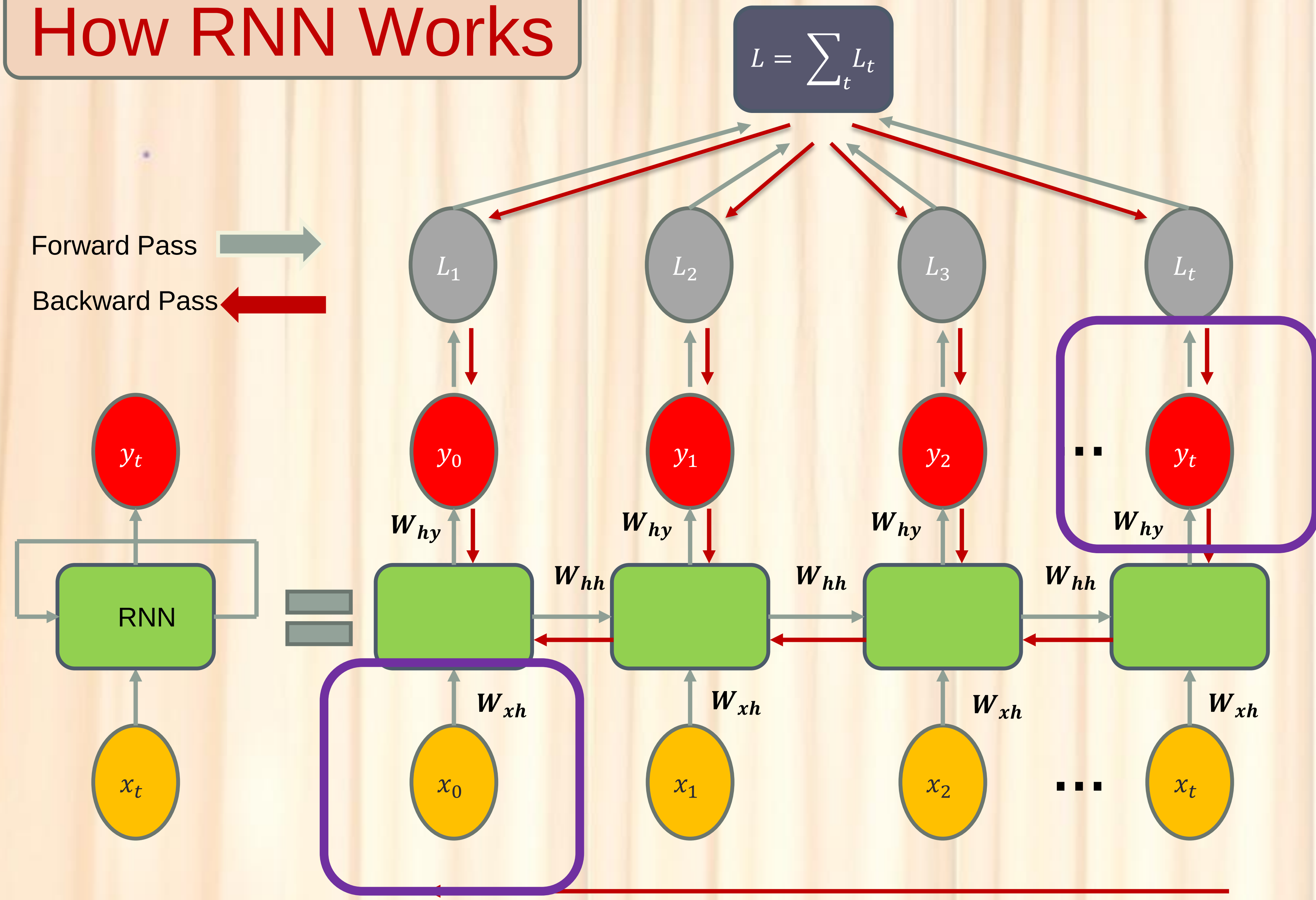
How RNN Works



How RNN Works



How RNN Works



Back propagation Through Time (BPTT)

Back propagation Through Time (BPTT) is the process of training an RNN by computing **gradients** over multiple time steps.

In BPTT, the same weight matrix is used repeatedly at each time step, leading to the accumulation of gradients across time.

Back propagation Through Time (BPTT)

Key Repeated Calculations in BPTT:

Forward Pass: Repeated Application of Weights

- At each time step $\boldsymbol{t}$, the hidden state $\boldsymbol{h_t}$ is updated using the same recurrent weight $\boldsymbol{W_h}$:

  $$\boldsymbol{h_t = f(W_h h_{t-1} + W_x x_t + b)}$$

- The output is computed using another weight matrix $\boldsymbol{Wy}$:

  $$\boldsymbol{y\text{^} = f(W_y h_t + c)}$$

Back propagation Through Time (BPTT)

Key Repeated Calculations in BPTT:

Backward Pass: Repeated Gradient Multiplication

- Gradients are propagated backward in time, computing derivatives at each step using the chain rule.
- The gradient of the loss function L with respect to the weight W_h is computed recursively:

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W_h}$$

Back propagation Through Time (BPTT)

Key Repeated Calculations in BPTT:

Backward Pass: Repeated Gradient Multiplication

- The gradient of hidden state $\boldsymbol{h_t}$ with respect to $\boldsymbol{h_{t-1}}$ involves multiplying by $\boldsymbol{W_h}$ repeatedly:

$$\frac{\partial h_t}{\partial h_{t-1}} = W_h f'(h_{t-1})$$

Back propagation Through Time (BPTT)

Key Repeated Calculations in BPTT:

Backward Pass: Repeated Gradient Multiplication

- Over many time steps, this becomes:

$$\frac{\partial h_T}{\partial h_0} = W_h^T \prod_{t=1}^{T} f'(h_t)$$

- If W_h is large >1 (e.g., 1.5), the gradient explodes.
- If W_h is small <1 (e.g., 0.5), repeated multiplication makes the gradient vanish.

Why are vanishing gradients a problem?

Multiply many small numbers together

↓

Error due to further back time steps have smaller and smaller gradients

↓

Bias parameter to capture short-term dependencies &
struggle to capture long-term dependencies

Difficulty in Learning Long-Term Dependencies Example

In language models, if an RNN tries to predict the last word in the sentence:

"The clouds are dark, it looks like it will _____." ***(rain)***

The model needs to retain the context from "**clouds are dark**" to predict "**rain**."

However, if the gradient vanishes, the model forgets earlier words and struggles to make an accurate prediction.

Solution to Vanishing Gradient Problem

1. Choose **Activation Functions** wisely which can mitigate this issue , for example we can choose ReLU , Tanh instead of sigmoid function

2. **Parameter Initialization** like initialize weights to identify matrix & biases to zero

3. Use **Gates** to selectively add or remove information within each recurrent unit with Gated cell such as LSTM , GRU etc

Long Short-Term Memory(LSTM) Networks

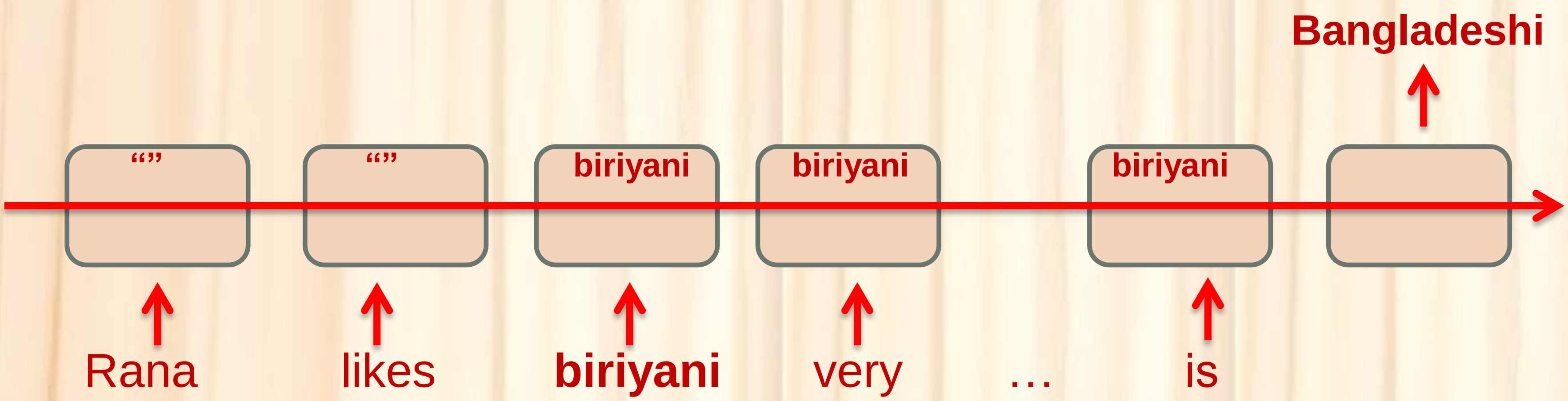
1. LSTM networks rely on a gated cell to track information through out many time steps

2. It was specially designed to address the **vanishing gradient problem** in traditional RNNs

3. They do this **Controlling the flow of information** across time steps, using a set of special mechanism called **gates** that regulate updates to the **hidden state** and **cell states**

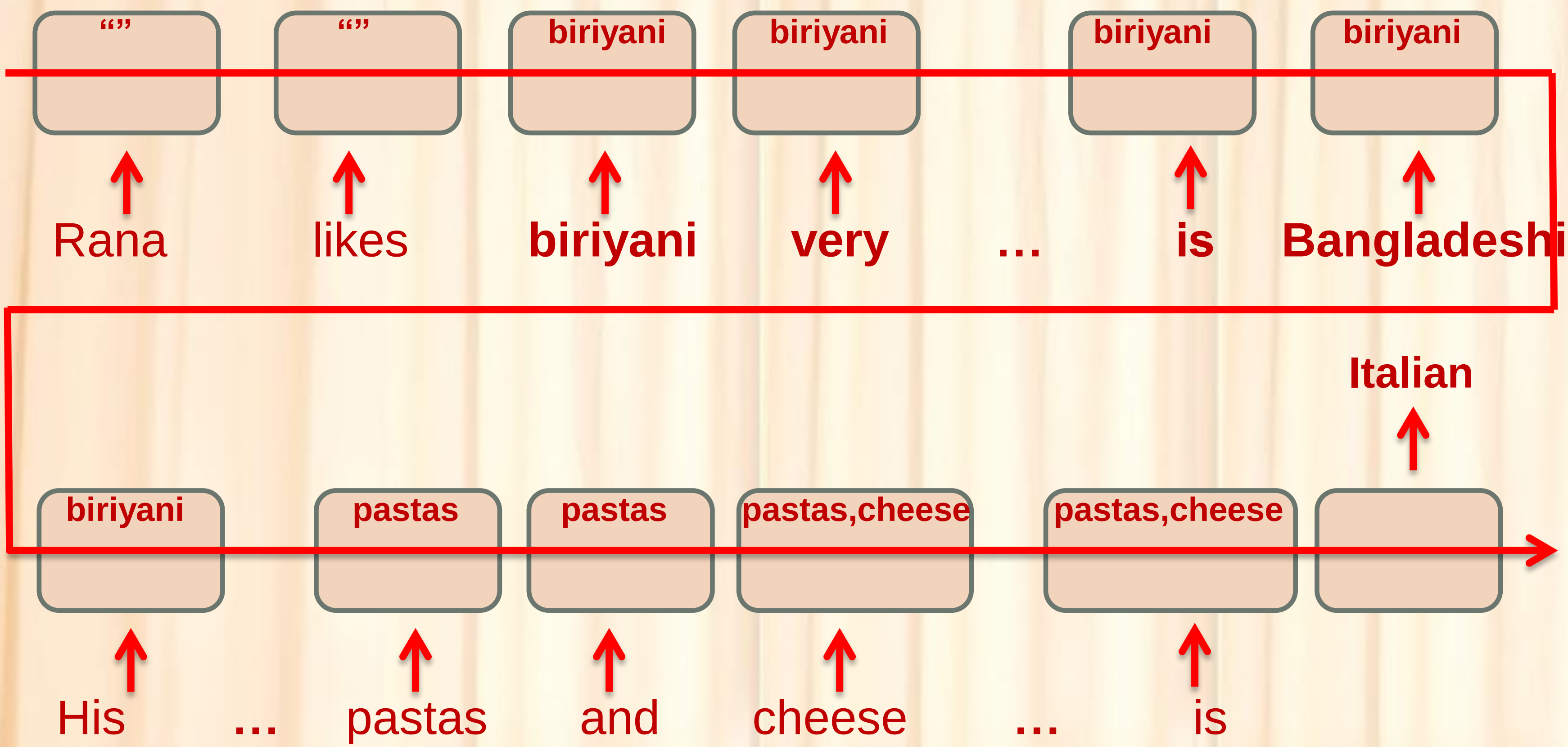
Example: Predict last word

Rana likes **biriyani** very much, so we can understand that his favourite cuisine is **Bangladeshi**.



Example: Predict last word

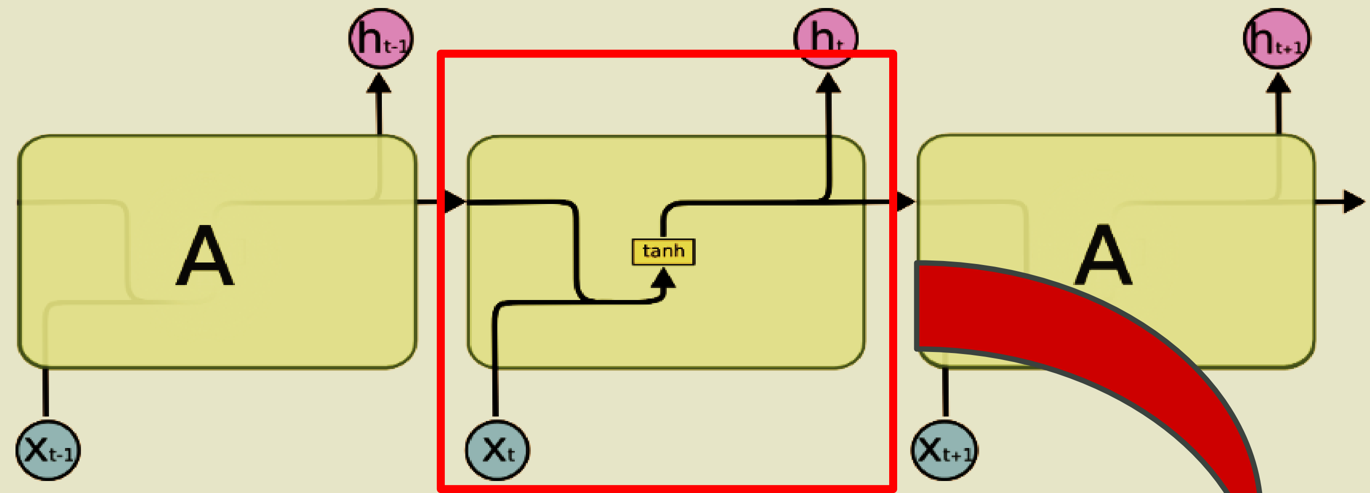
“Rana likes **biriyani** very much, so we can understand that his favourite cuisine is **Bangladeshi**.
His friend John however loves **pastas** and **cheese** that means John ’s favourite cuisine is **Italian**."



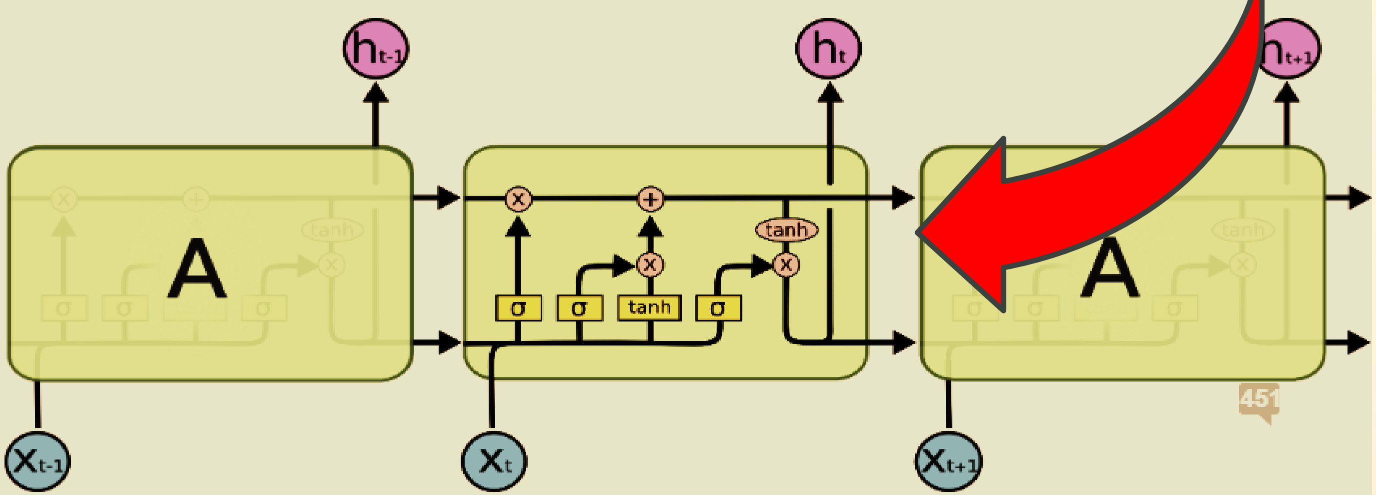
Key Components of LSTM

1. **Cell state (C_t):** Acts as a memory, carrying information across many time steps.
2. **Forget gate:** Decides what proportion of the previous cell state to "forget.“
3. **Input gate:** Determines which new information to store in the **cell state**.
4. **Output gate:** Decides what information from the **cell state** should be passed to the **hidden state** h_t and output.

Adding Extra Pathways Through Neuron



The repeating module in a standard RNN contains a single layer.



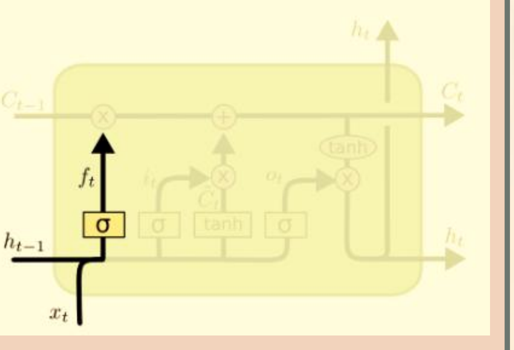
The repeating module in an LSTM contains four interacting layers.

How LSTMs Prevent Vanishing Gradients

1. Cell State and Forget Gate:

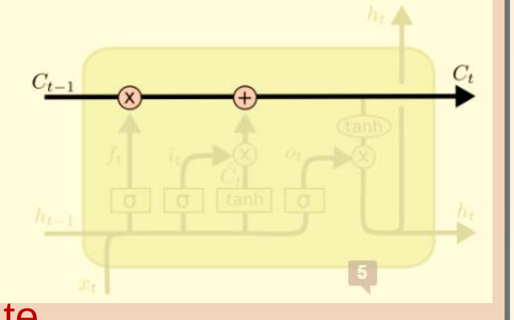
The **cell state** in an LSTM acts like a long-term memory. The **forget gate** controls what information from the previous cell state should be remembered and what should be forgotten.

- **Forget gate (f_t):** The forget gate looks at the previous hidden state h_{t-1} and the current input x_t , and produces a value between 0 and 1, which represents how much of the previous cell state C_{t-1} should be forgotten:



$$f_t = \sigma(Wf \cdot [h_{t-1}, x_t] + bf)$$

where σ is the **sigmoid activation function**.
If f_t=0, everything is forgotten.
If f_t=1, the entire previous cell state is retained.



The **cell state** is updated by combining the forgotten information and the new candidate information from the input gate: $C_t = f_t \cdot C_{t-1} + i_t \cdot C_{t}$ _hat

where i_t is the input gate and C_t _hat is the candidate cell state.

How LSTMs Prevent Vanishing Gradients

2. Input Gate:

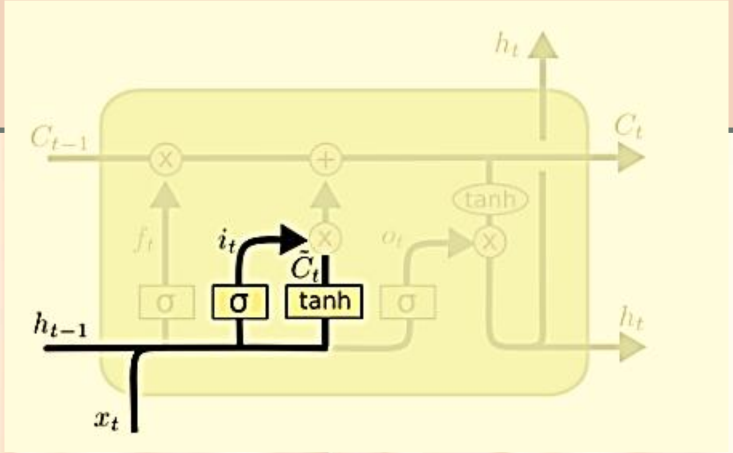
The input gate decides what new information to store in the cell state:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

The candidate cell state is computed as:

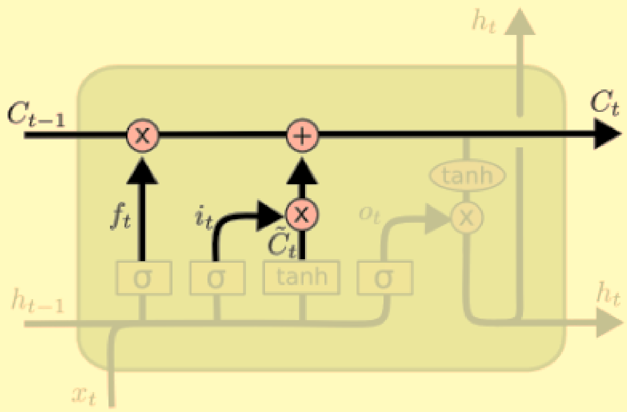
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The input gate allows LSTMs to selectively update the cell state, storing important information and blocking irrelevant information from being added.



How LSTMs Prevent Vanishing Gradients

Cell State, Forget and Input Gate get combined:



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

How LSTMs Prevent Vanishing Gradients

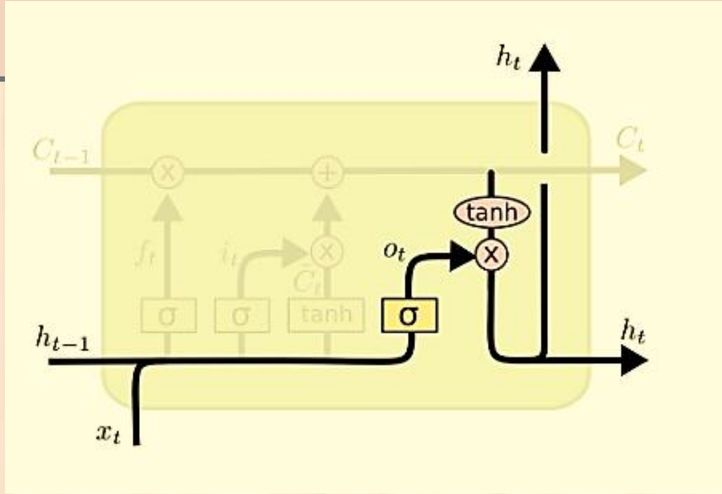
3. Output Gate:

The output gate determines what part of the cell state C_t should be passed as the hidden state h_t for the current time step:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

This ensures that only relevant information is passed to the next time step.



Python Implementation of Vanishing Gradient Problem

Github Link :

https://github.com/MasudRana2406/Python-Implementation-of-Vanishing-Gradient-Problem.git

Live Demonstrate in Youtube

YouTube Link : https://www.youtube.com/watch?v=TetKl1tGFBs

Reference

- **Website**

  colah's blog(*August 27, 2015*). 'Understanding LSTM Networks
' https://colah.github.io/posts/2015-08-Understanding-LSTMs/

  Medium. (2024). '*Forward and Backward Propagation in Multilayered Neural Networks: A Deep Dive'* *https://medium.com/@jainvidip/forward-and-backward-propagation-in-multilayered-neural-networks-a-deep-dive-d596e875dedf*

- **Research Paper (IEEE)**

  Yoshua Bengio, Patrice Simard and Paolo Frasconi, student member, "Learning Long-term Dependencies with Gradient Descent is Difficult'', vol. 5, no. 2, March 1994. https://www.comp.hkbu.edu.hk/~markus/teaching/comp7650/tnn-94-gradient.pdf

- **YouTube Video:**

  MIT 6.S191 (2024): Recurrent Neural Networks, Transformers, and Attention
  https://www.youtube.com/watch?v=dqoEU9Ac3ek&t=2210s