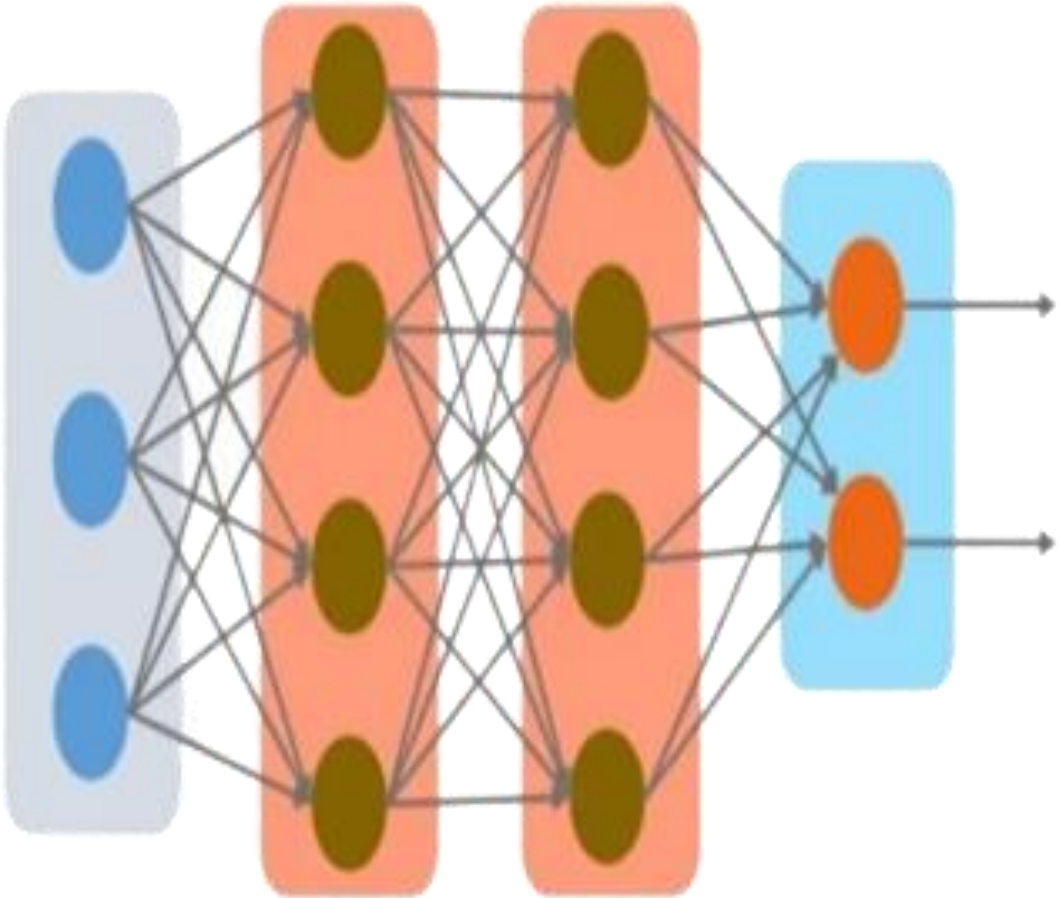
Unit 3
Recurrent Neural Networks

Recurrent Neural Networks

- Recurrent Neural Networks are the family of feed-forward neural networks.
- RNN is a type of Neural Network where the output from the previous step is fed as input to the current step.
- Main and most important feature of RNN is **Hidden state**, remembers some information about a sequence ,referred to as *Memory State.*
- RNN allow for both parallel and sequential computation.



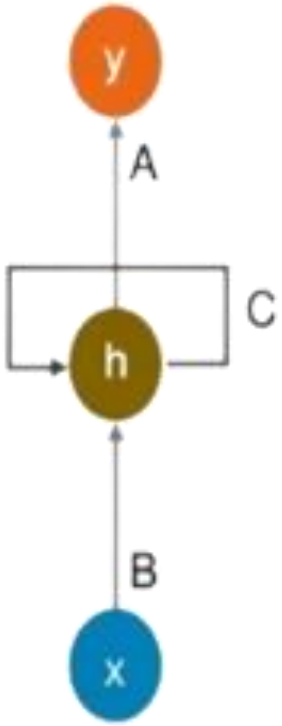
Feed Forward Neural Network

01 cannot handle sequential data

02 considers only the current input

03 cannot memorize previous inputs

Solution - **Recurrent Neural Networks**



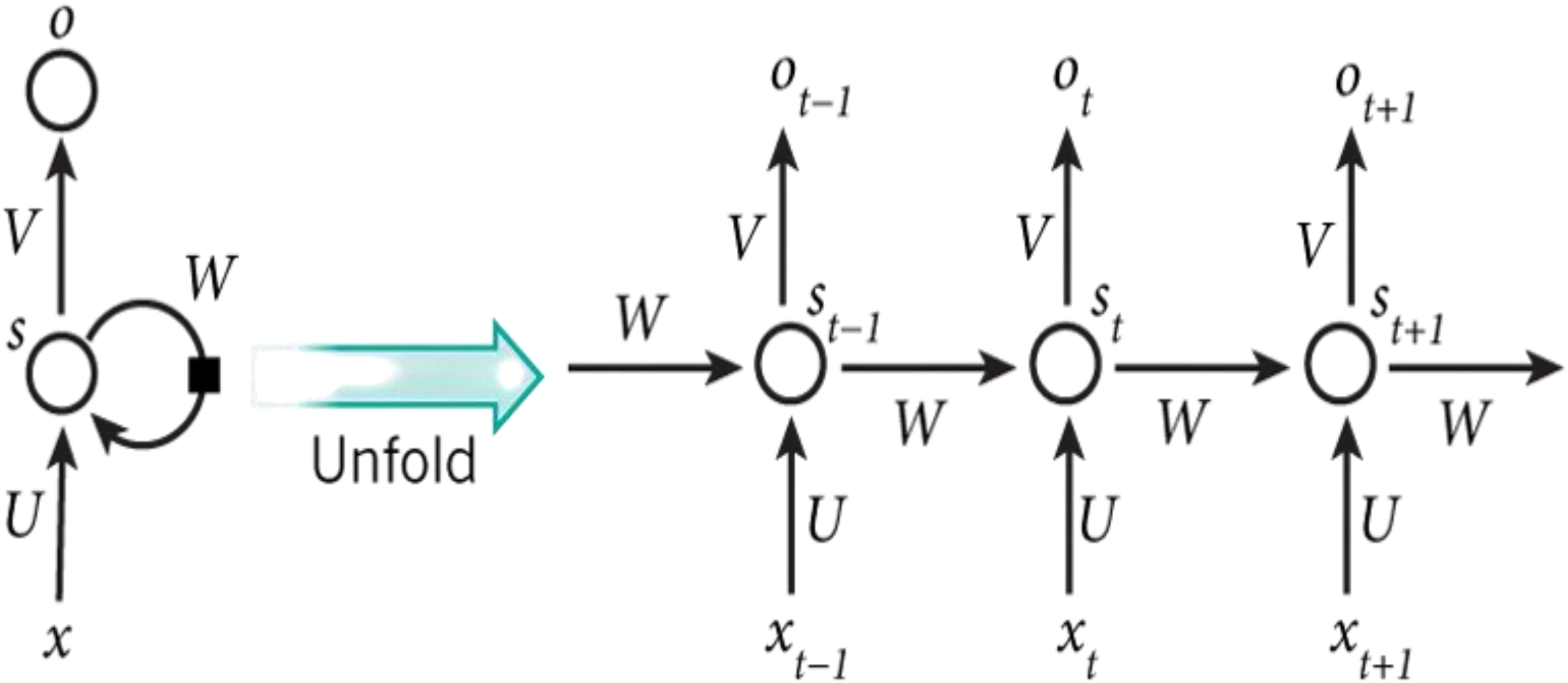
Recurrent Neural Network

Network

01 can handle sequential data

02 considers the current input and also the previously received inputs

03 can memorize previous inputs due to its internal memory



x_t: input at time t, s_t: hidden state at time t, and O_t: output at time t

- **Formula to calculate current state:**
- $$S_t = f(S_{t-1}, x_t)$$
- S_t is the current state, S_{t-1} is the previous state and x_t is the current input**.**

The equation applying after activation function (tanh) is:

$$S_t = \tanh(W_{ss} S_{t-1} + W_{xs} x_t)$$

W_{ss} : weight at recurrent neuron, W_{xs} : weight at input neuron

The output state can be calculated as:

$$O_t = W_{sy} S_t$$

O_t is the output state, W_{sy}: weight at output layer, S_t: current state

- **Advantages of Recurrent Neural Network**

1. Useful in time series prediction only, feature to remember previous inputs as well.

   Called as Long Short Term Memory.

1. Recurrent neural networks are even used with convolutional layers.

Disadvantages of Recurrent Neural Network

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. Cannot process very long sequences if using tanh or relu as an activation function.

Applications of Recurrent Neural Network

1.Language Modelling and Generating Text

2.Speech Recognition

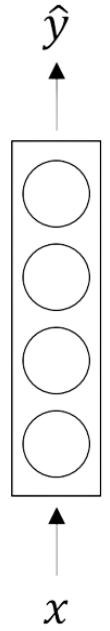
3.Machine Translation

4.Image Recognition, Face detection

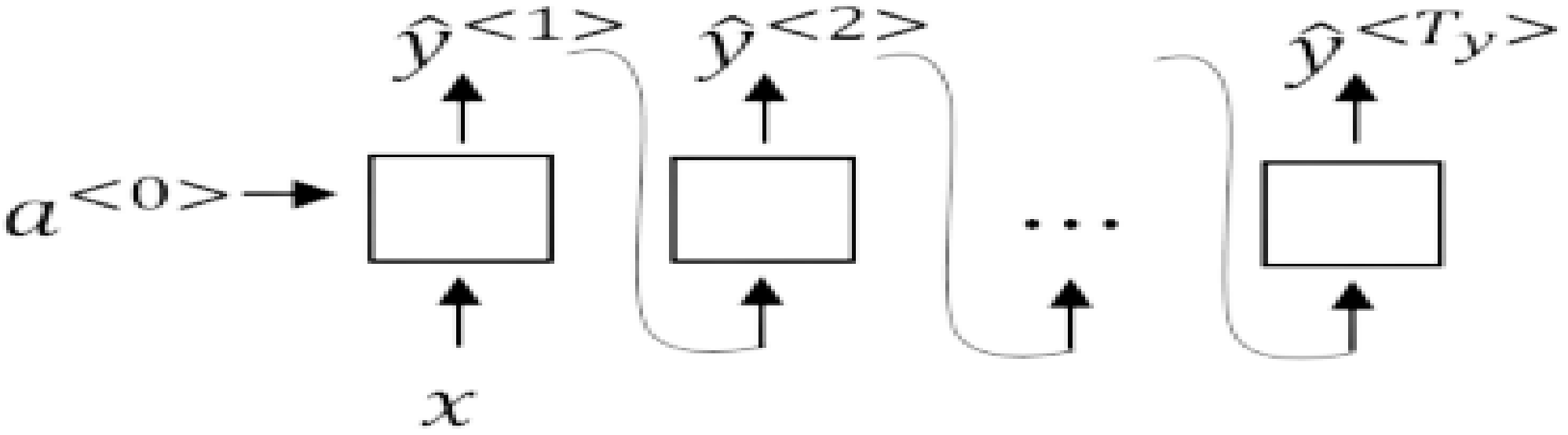
5.Time series Forecasting

Types of Recurrent Neural Networks

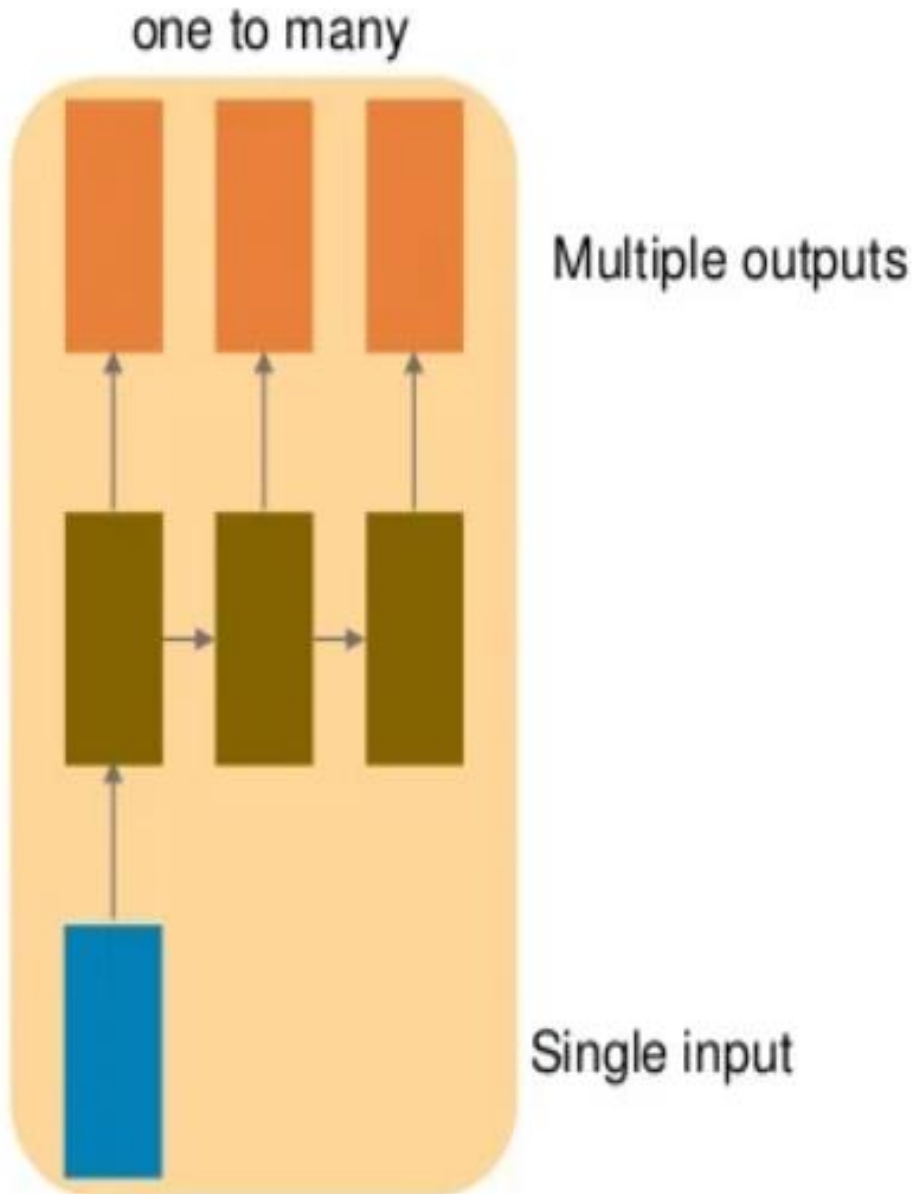
- **One to one**
- Model is similar to a single layer neural network, provides linear predictions
- Mostly used fixed-size input ‘x’ and fixed-size output ‘y’ (example: image classification).
- Also known as Vanilla Neural Network.



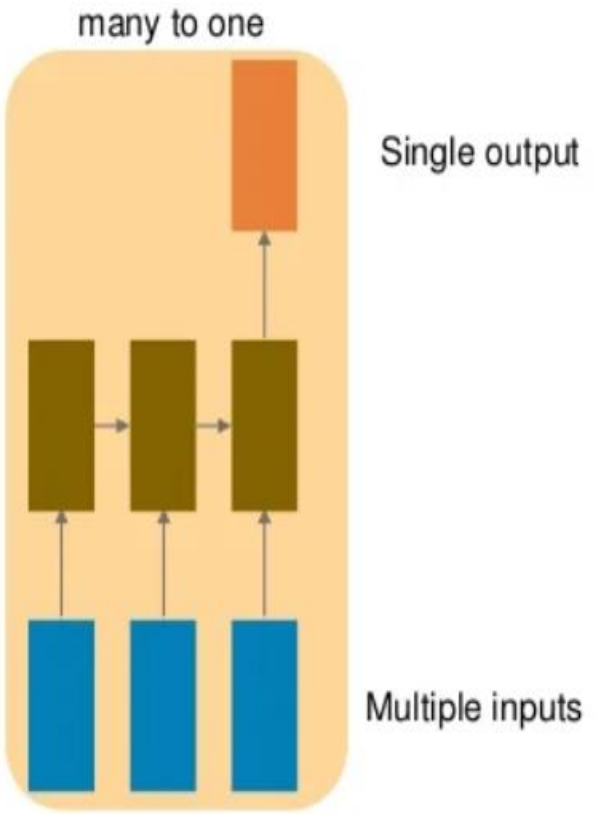
- **One to many:**
- Consist of a single input ‘x’, activation ‘a’, and multiple outputs ‘y’
- In some cases, it propagates the output ‘y’ to the next RNN units.



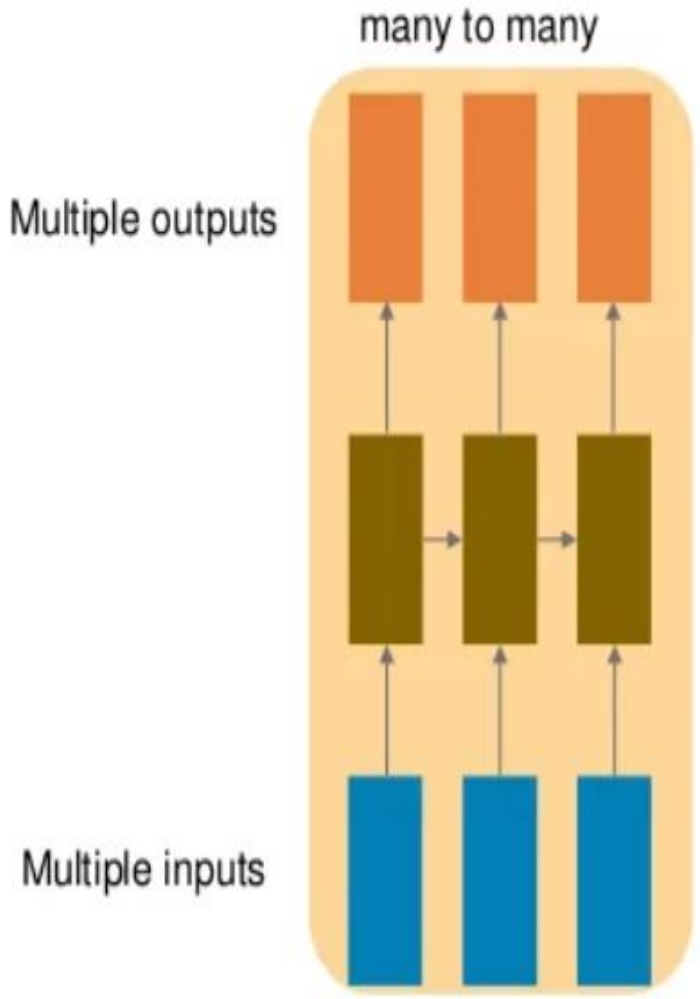
One to many


one to many
Multiple outputs
Single input

- **Many to one**
- Consist of multiple inputs ‘x’ (such as words or sentences), activation ‘a’ and produce a single output ‘y’ at the end.
- Type of architecture is mostly used to perform sentiment analysis.



- **Many to many**
- Multiple inputs ‘x’, activations ‘a’ which are propagated through the network to produce output ‘y’.

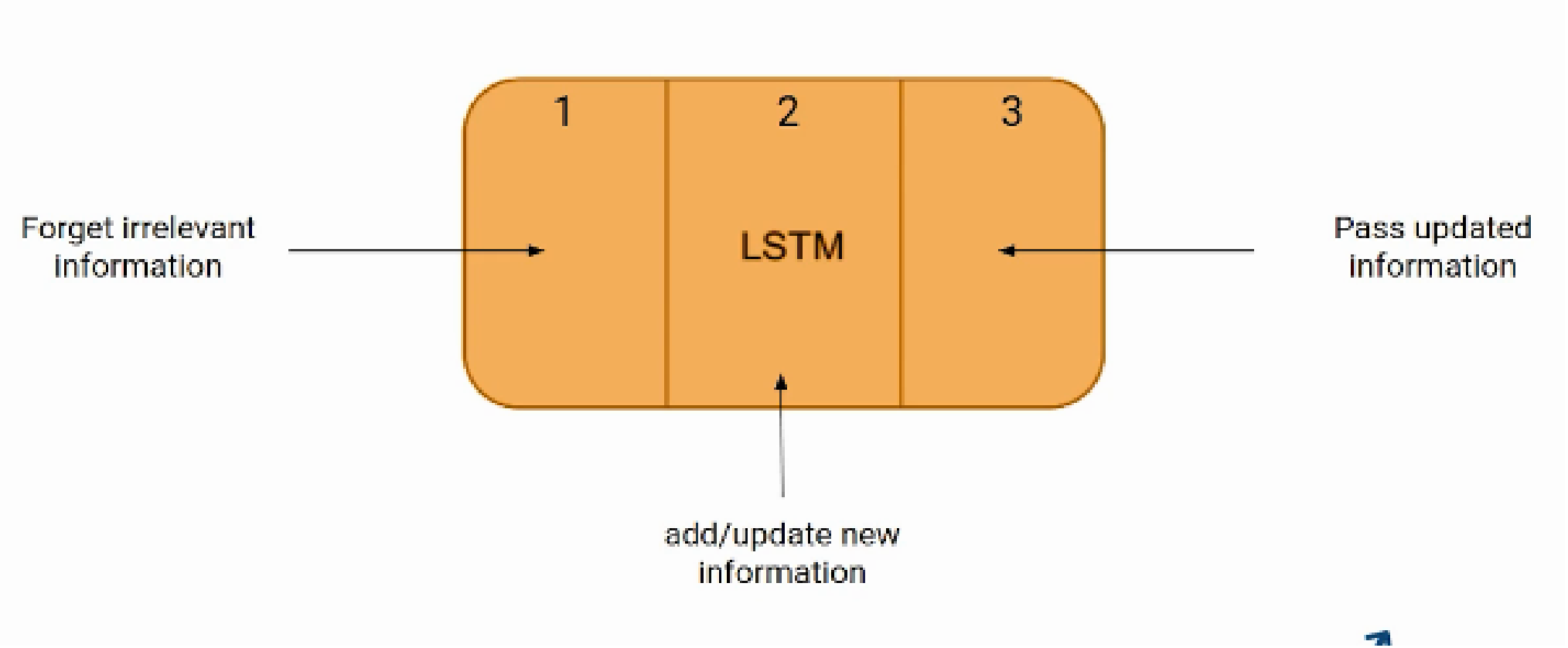


Long Short Term Memory (LSTM)

- LSTM is an artificial recurrent neural network (RNN) architecture used in the field of deep learning.
- It was proposed in 1997 by **Sepp Hochreiter** and **Jurgen schmidhuber**.
- Unlike standard feed-forward neural networks, LSTM has feedback connections.
- It can process single data points (such as images) and entire sequences of data (such as speech or video).
- Long short-term memory (LSTM) network is the most popular solution to the **vanishing gradient problem**.

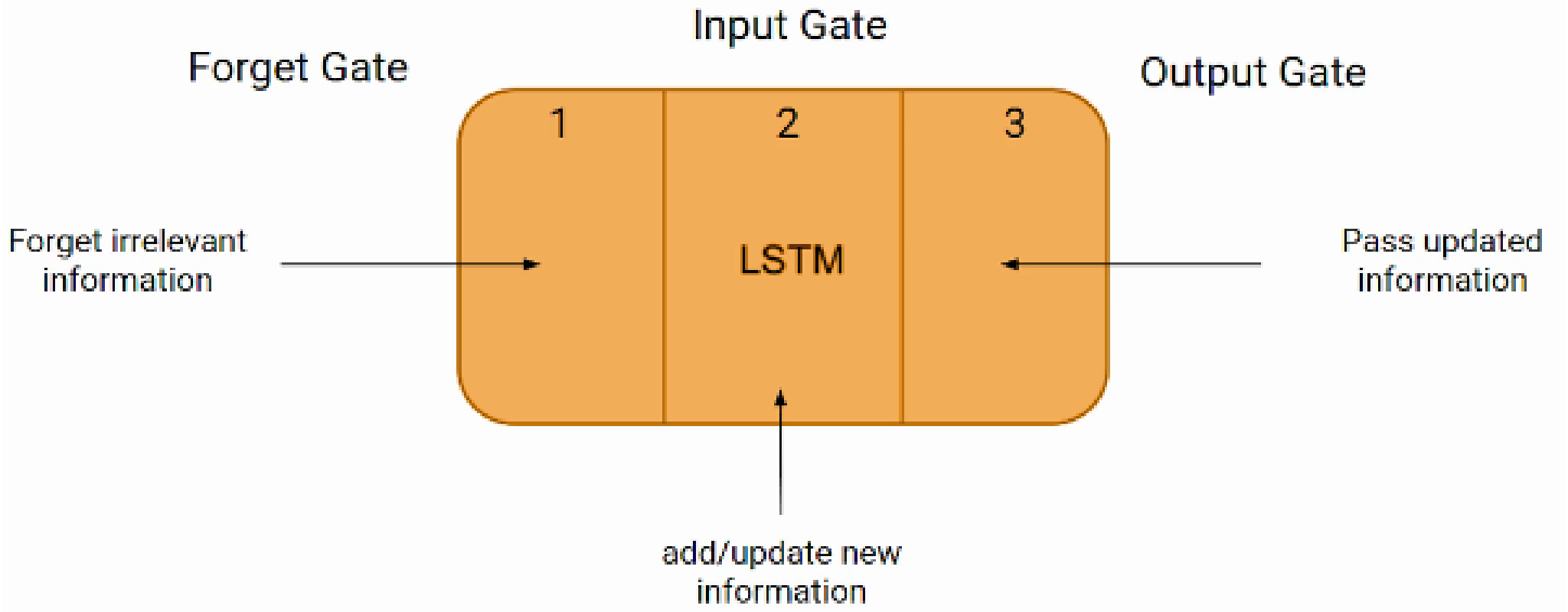
LSTM Architecture

- The LSTM network architecture consists of three parts.

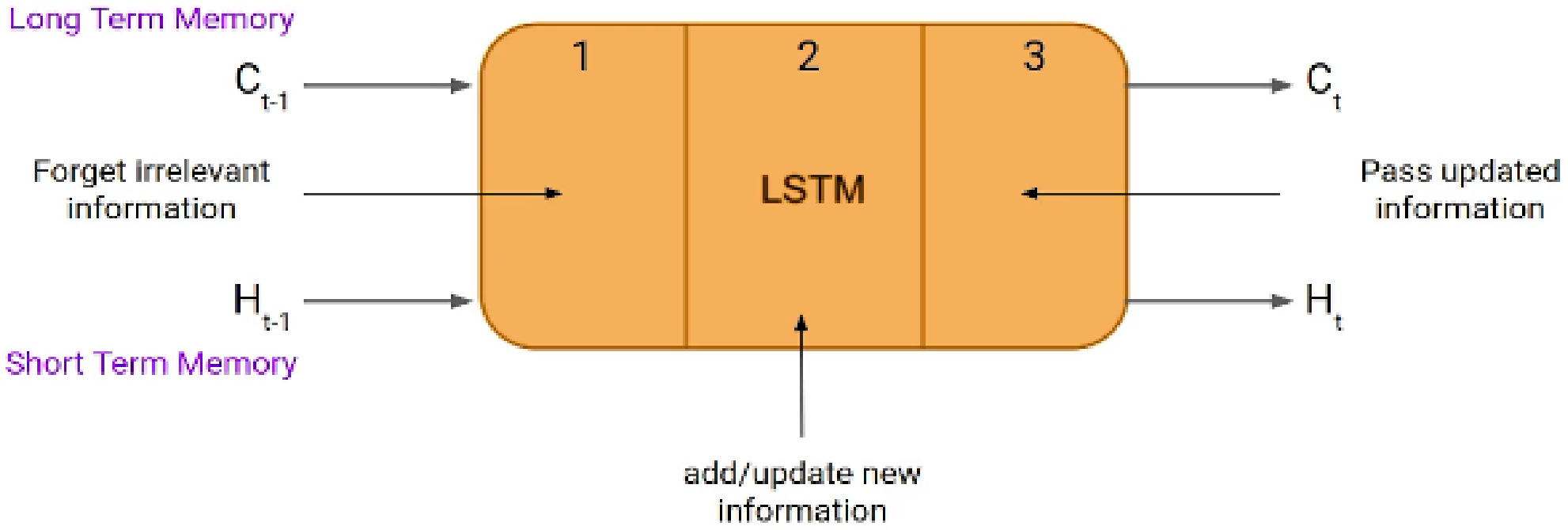


LSTM Architecture

- First part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten.
- Second part, the cell tries to learn new information from the input to this cell.
- At last, third part, the cell passes the updated information from the current timestamp to the next timestamp.
- One cycle of LSTM is considered a **single-time step**.
- Three parts of an LSTM unit are known as **gates**.
- The first gate is called **Forget gate, the** second gate is known as **the Input gate,** and the last one is **the Output gate**.
- Control the flow of information in and out of the memory cell or lstm cell.

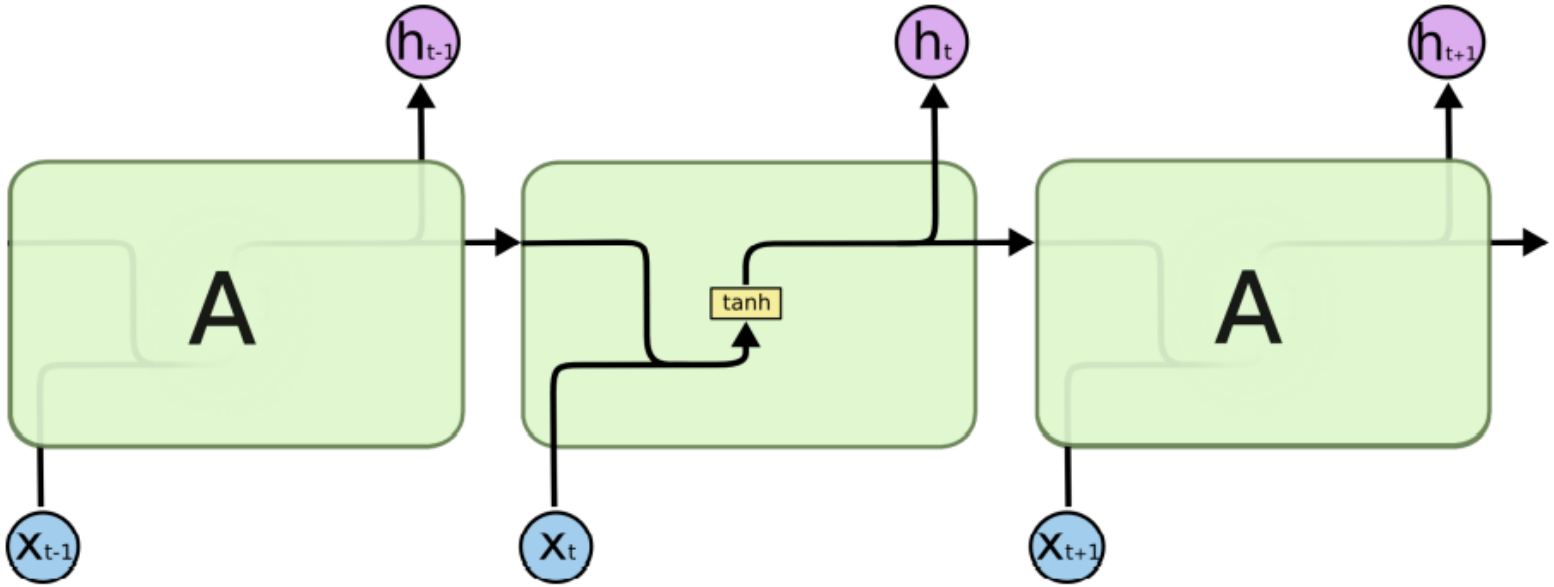

Input Gate
Forget Gate
Output Gate
1
2
3
Forget irrelevant information
LSTM
Pass updated information
add/update new information

- LSTM has a hidden state where H(t-1) represents the hidden state of the previous timestamp.
- Ht is the hidden state of the current timestamp.
- LSTM also has a cell state represented by C(t-1) and C(t) for the previous and current timestamps.
- **Hidden state is known as Short term memory, and the cell state is known as Long term memory**.

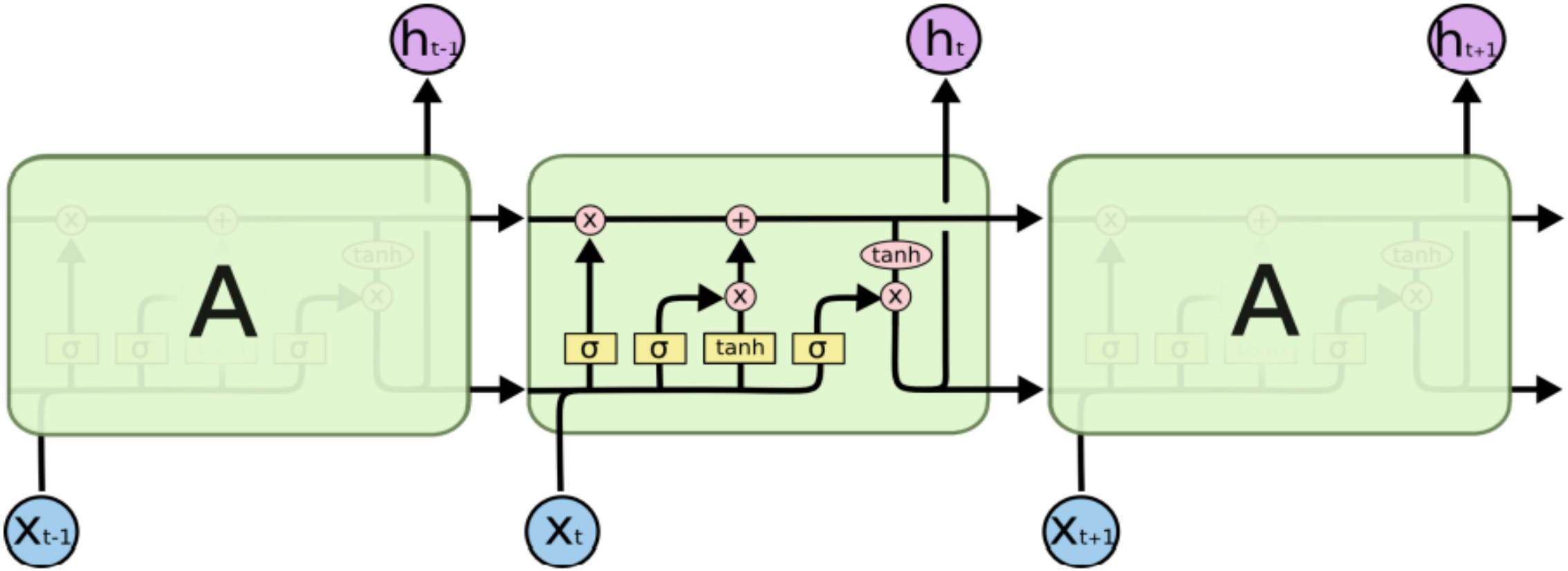


LSTM Networks

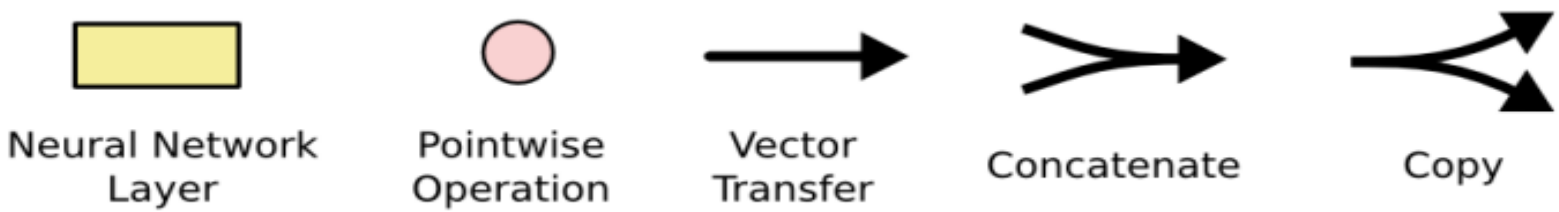
- LSTMs are explicitly designed to avoid the long-term dependency problem.
- All recurrent neural networks have the form of a chain of repeating modules of neural network.
- In standard RNNs, repeating module - a single tanh layer.



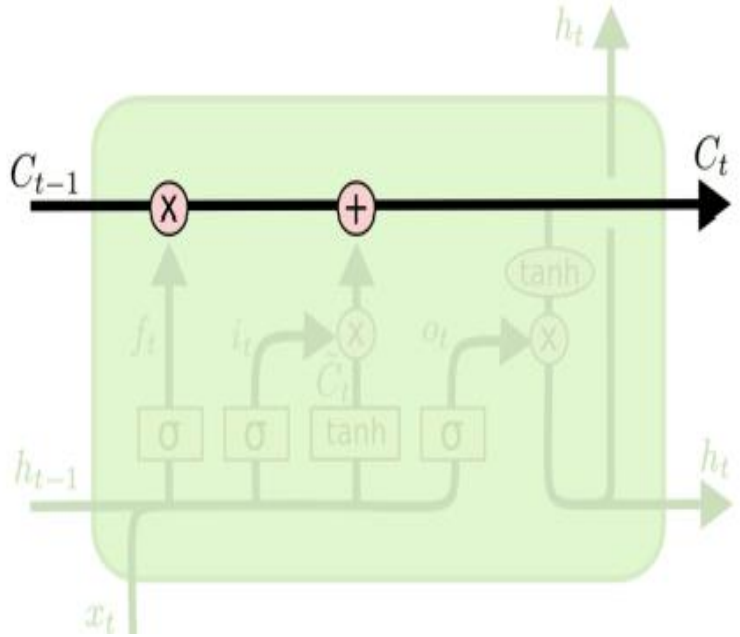
The repeating module in a standard RNN contains a single layer.



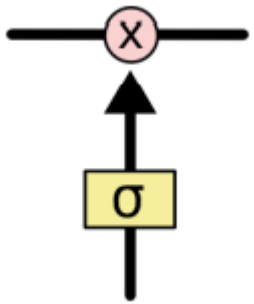
The repeating module in an LSTM contains four interacting layers.



LSTMs - Cell State & Gates

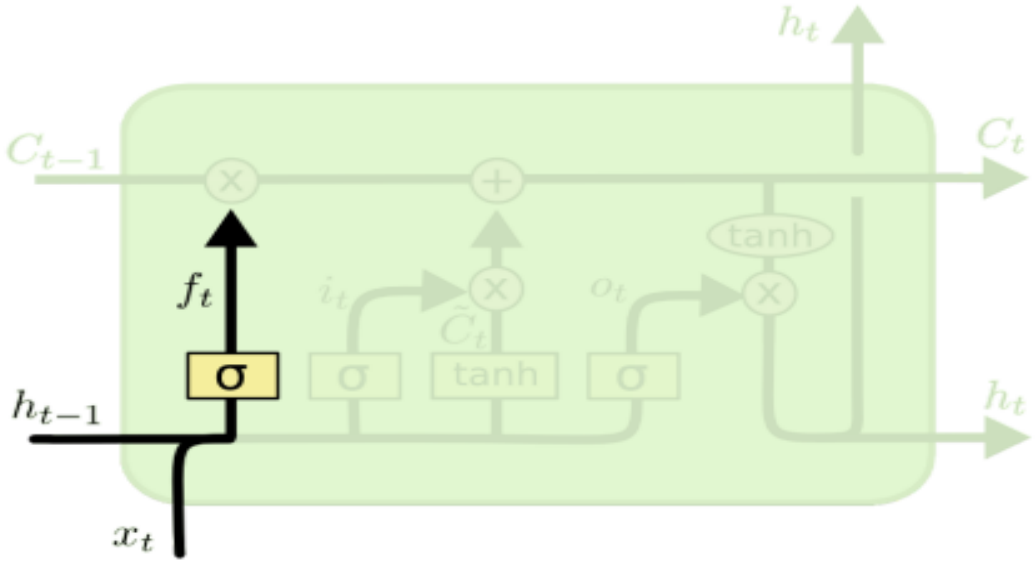


-Key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
-Cell state is kind of like a conveyor belt.
- It runs straight down the entire chain, with only some minor linear interactions.
- It's very easy for information to just flow along it unchanged.



- The LSTM does have the ability to remove or add information to the cell state.
  - Regulated by structures called <u>gates.</u>
  - Gates are a way to optionally let information through.
- They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

- Sigmoid layer outputs numbers between zero and one.
- Zero -“let nothing through,” & one - “let everything through!”



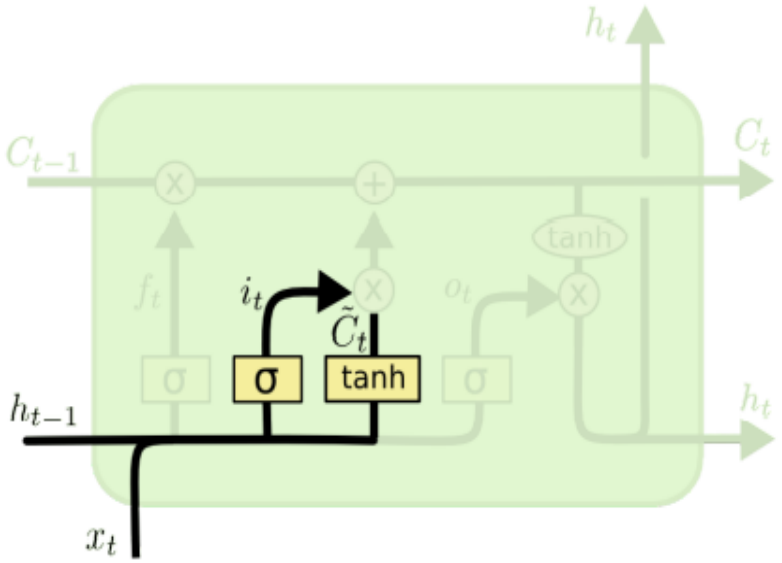
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

First step in LSTM is to decide what information we're going to throw away from the cell state.

Decision is made by a sigmoid layer called the "forget gate layer."

It looks at ht-1 and xt , and outputs a number between 0 and 1 for each number in the cell state Ct-1.

1 represents "completely keep this" while a 0 represents "completely get rid of this.

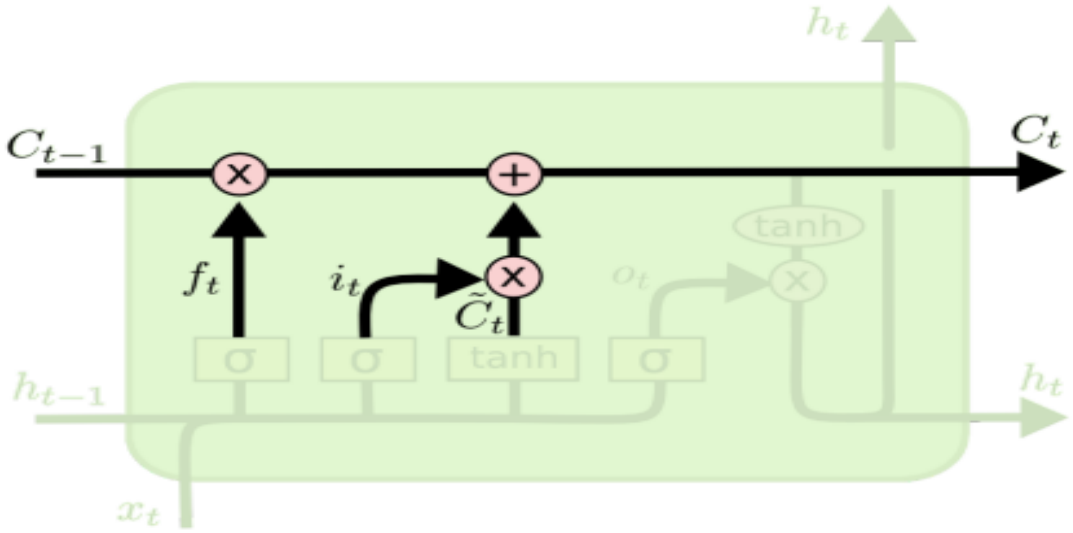


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

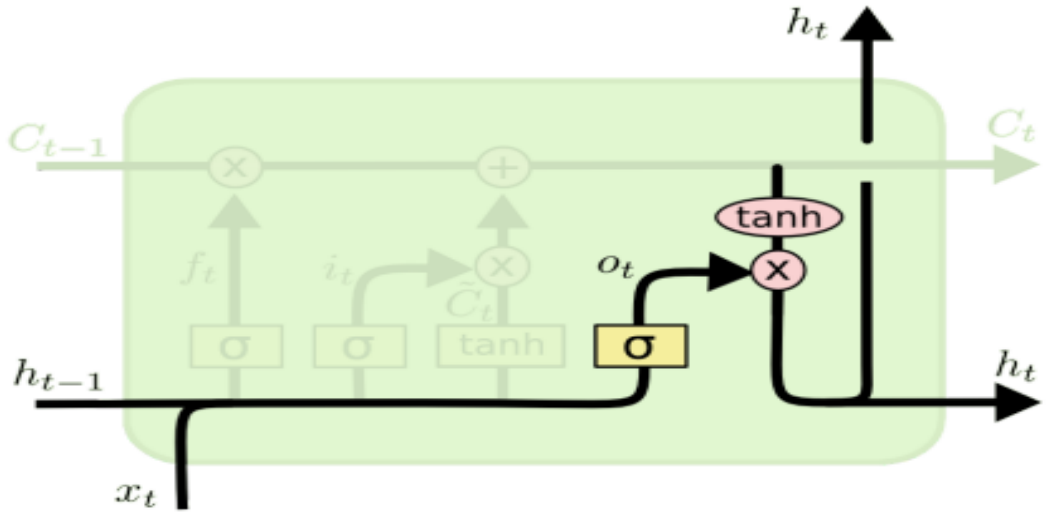
The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update.

A tanh layer creates a vector of new candidate values, Ct , that could be added to the state.



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

- It's now time to update the old cell state,*Ct-1* , into the new cell state *Ct* .
- We multiply the old state by *f*t, forgetting the things we decided to forget earlier.
- Then we add *i*t * *C*t. This is the new candidate values.



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

$$h_t = o_t * \tanh\left(C_t\right)$$

- Finally, we need to decide what we're going to output.
- First, run a sigmoid layer which decides what parts of the cell state going to output.
- Then, put the cell state through tanh and multiply it by the output of the sigmoid gate.
- Only output the parts we decided.

Encoder Decoder Architecture

Encoder Decoder Architecture

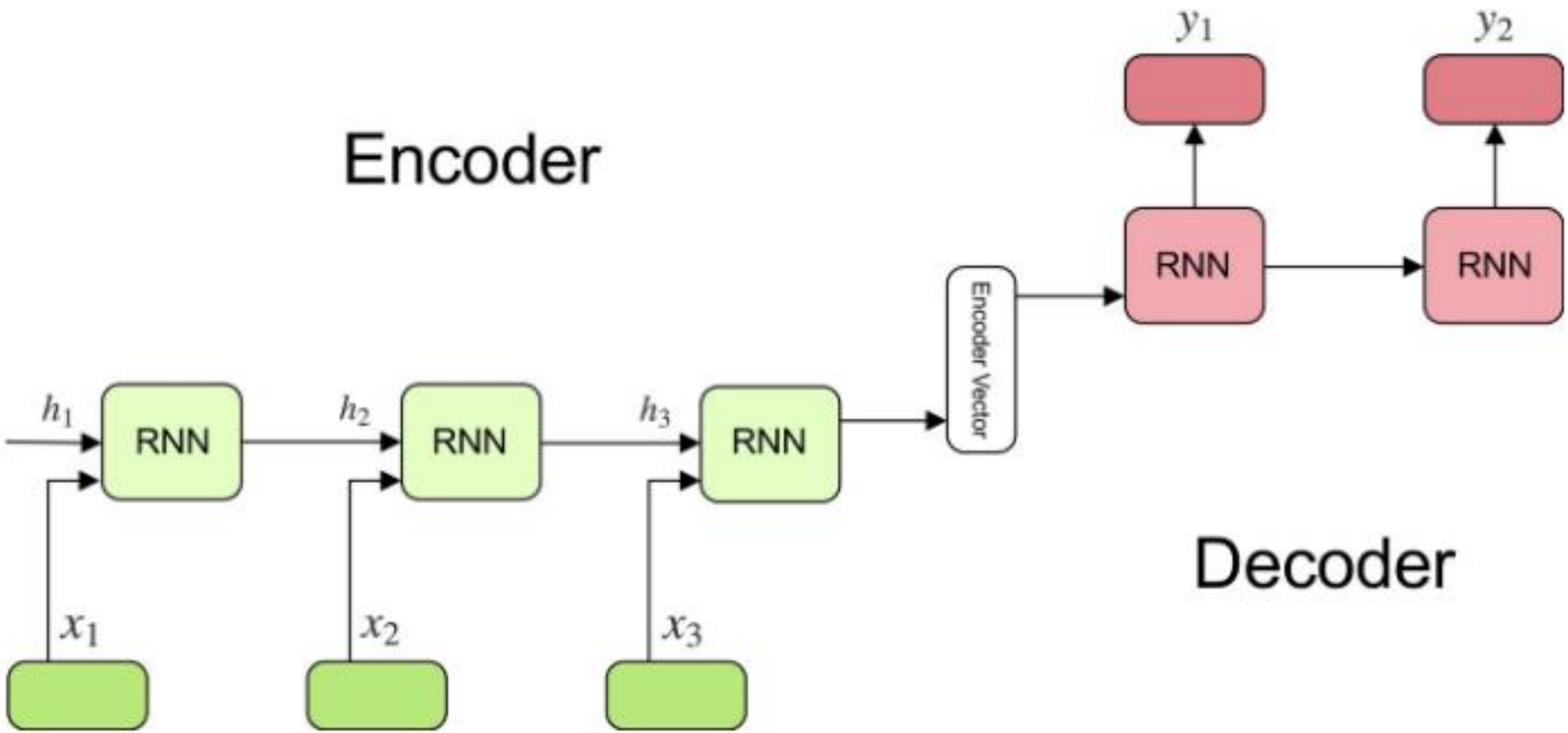
- There are three main blocks in the encoder-decoder model,

- Encoder
- Hidden Vector
- Decoder

- The Encoder will convert the input sequence into a single-dimensional vector (hidden vector).
- The decoder will convert the hidden vector into the output sequence.

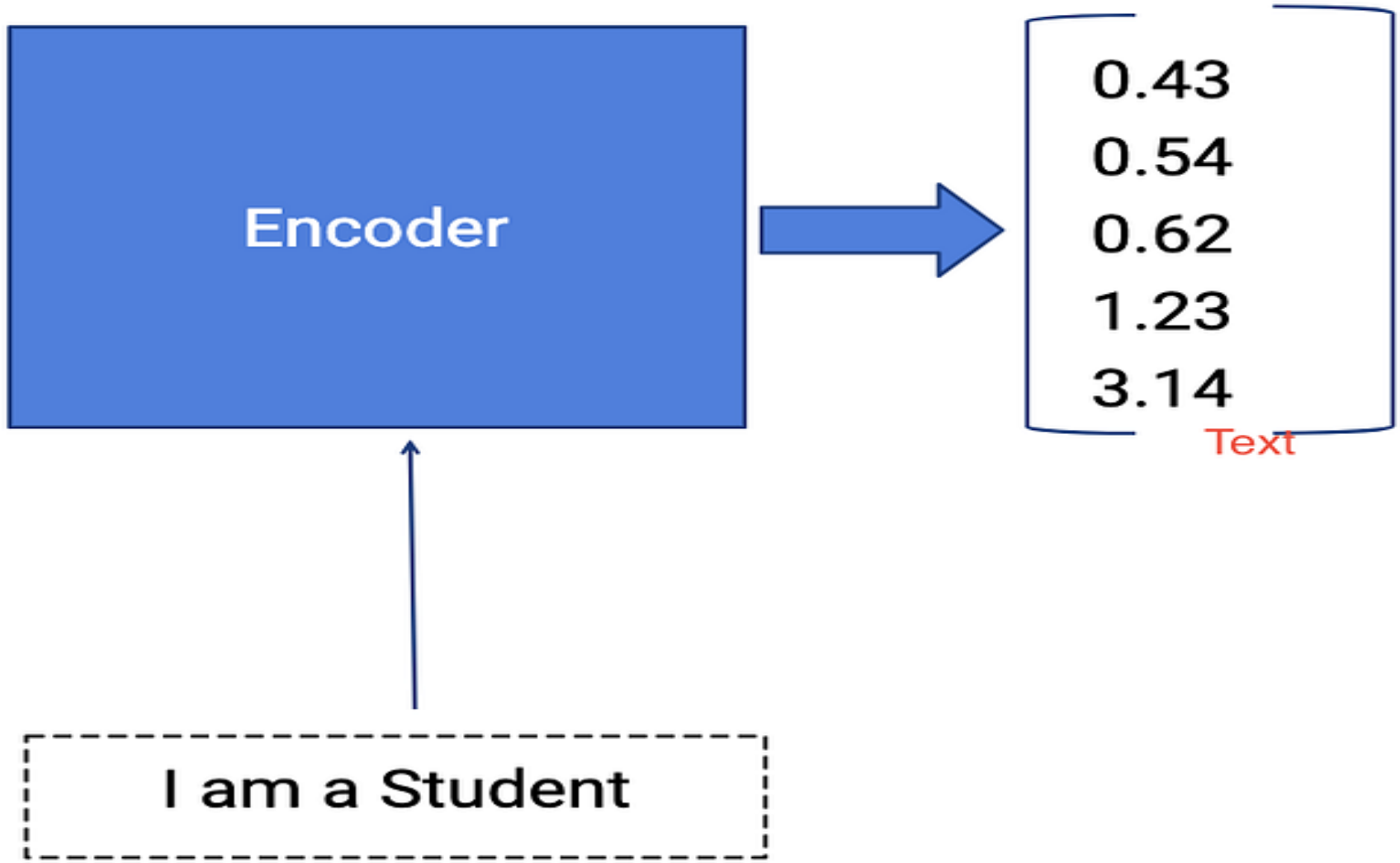
Encoder Decoder Architecture

- Designed to tackle a specific task known as sequence-to-sequence learning.
- RNNs are very good at working with models in which the dimensionality of both inputs and outputs is known and fixed.
- Examples :
- Machine Translation or Question Answering models translate an input sentence to an output of a different length.
- Sequence-to-sequence learning attempts to map the input to output sequence of various sizes.



Encoder Decoder architecture

- Multiple RNN cells can be stacked together to form the encoder. RNN reads each inputs sequentially.
- For every timestep (each input) t, the hidden state (hidden vector) h is updated according to the input at that timestep X[i].
- After all the inputs are read by encoder model, the final hidden state of the model represents the context/summary of the whole input sequence.
- Example: Consider the input sequence “I am a Student” to be encoded. There will be totally 4 timesteps (4 tokens) for the Encoder model.
- At each time step, the hidden state h will be updated using the previous hidden state and the current input.

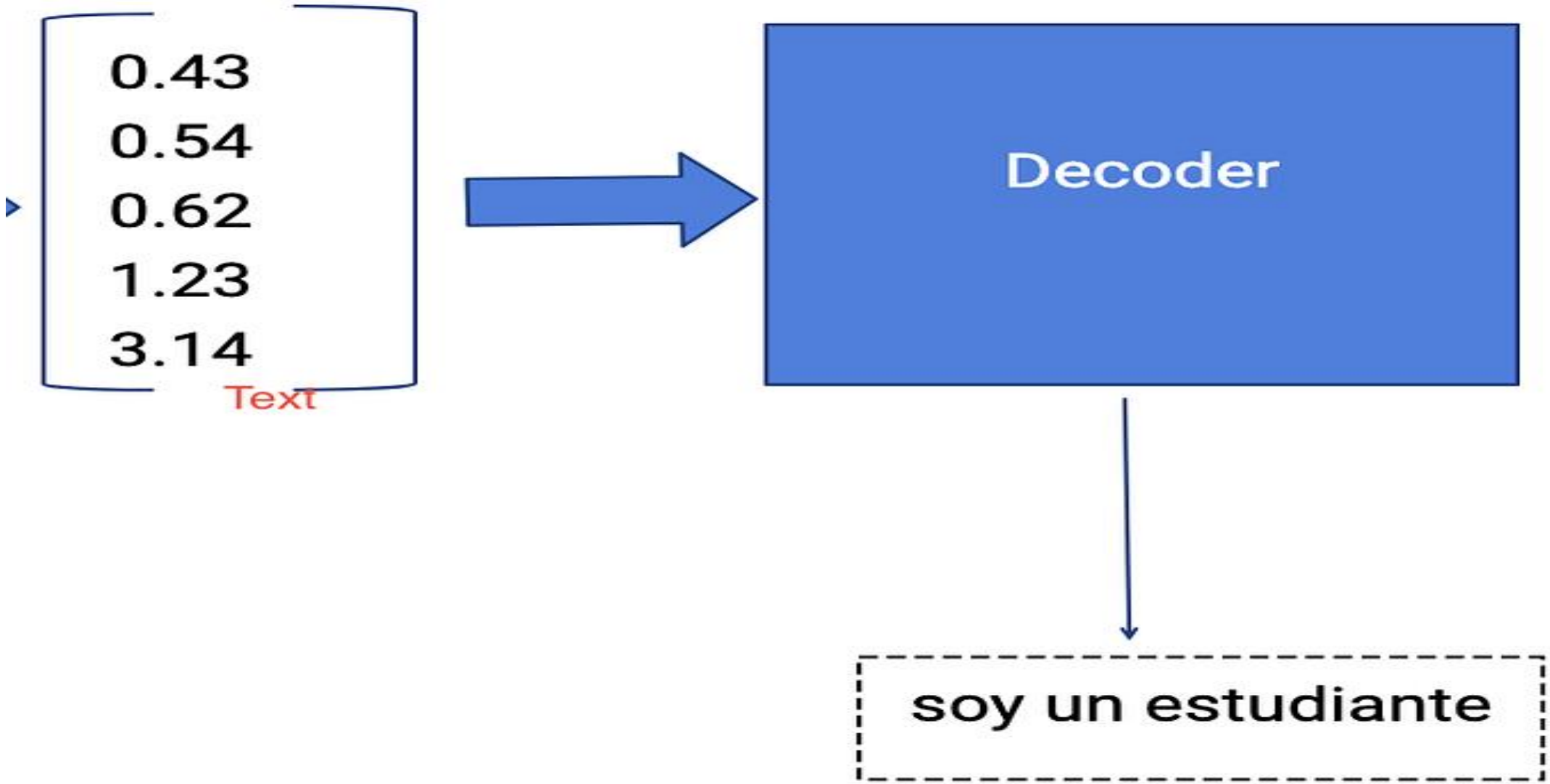

Encoder
0.43
0.54
0.62
1.23
3.14
Text
I am a Student

- At the first timestep t1, the previous hidden state h0 will be considered as zero or randomly chosen. So the first RNN cell will update the current hidden state with the first input and h0.
- Each layer outputs two things — updated hidden state and the output for each stage.
- The outputs at each stage are rejected and only the hidden states will be propagated to the next layer.
- The hidden states *h_i* are computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

- At second timestep t2, the hidden state h1 and the second input X[2] will be given as input , and the hidden state h2 will be updated according to both inputs.
- Then the hidden state h1 will be updated with the new input and will produce the hidden state h2.
- This happens for all the four stages.
- **Encoder Vector**
- This is the final hidden state produced from the encoder part of the model.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

- **Decoder**
- Decoder generates the output sequence by predicting the next output Yt given the hidden state ht.
- The input for the decoder is the final hidden vector obtained at the end of encoder model.
- Each layer will have three inputs, hidden vector from previous layer ht-1 and the previous layer output yt-1, original hidden vector h.
- At the first layer, the output vector of encoder and the random symbol START, empty hidden state ht-1 will be given as input, the outputs obtained will be y1 and updated hidden state h1
- The second layer will have the updated hidden state h1 and the previous output y1 and original hidden vector h as current inputs, produces the hidden vector h2 and output y2.

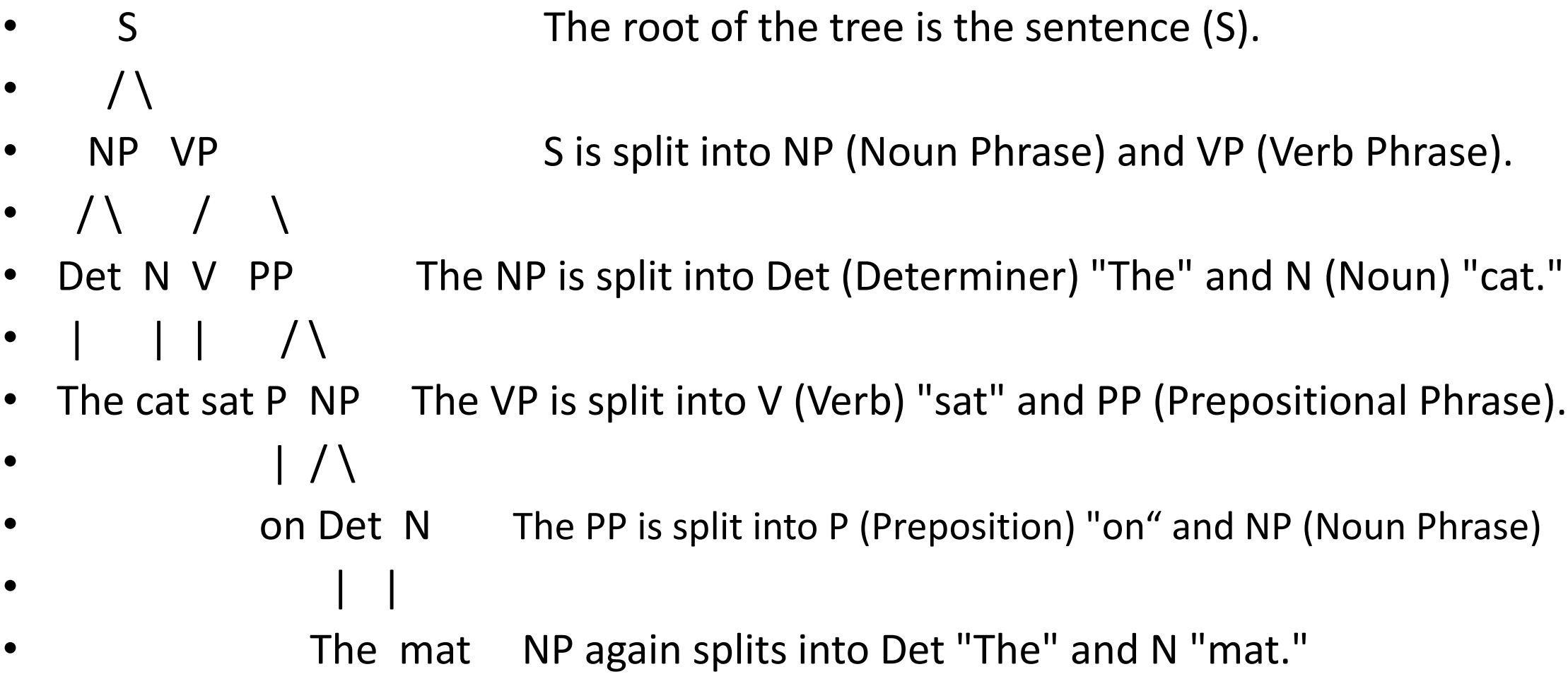

0.43
0.54
0.62
1.23
3.14
Text
Decoder
soy un estudiante

Recursive Neural Network (RvNNs)

Recursive Neural Network (RvNNs)

- A recursive neural network is a kind of deep neural network that can process structured data, such as trees or graphs, by applying the same set of weights recursively over the input.
- It can produce a structured output, such as a parse tree, or a scalar output, such as a sentiment score.
- Recursive neural networks are useful for natural language processing tasks that require understanding the hierarchical structure of sentences or document.
- The word recursive indicates that the neural network is applied to its output.

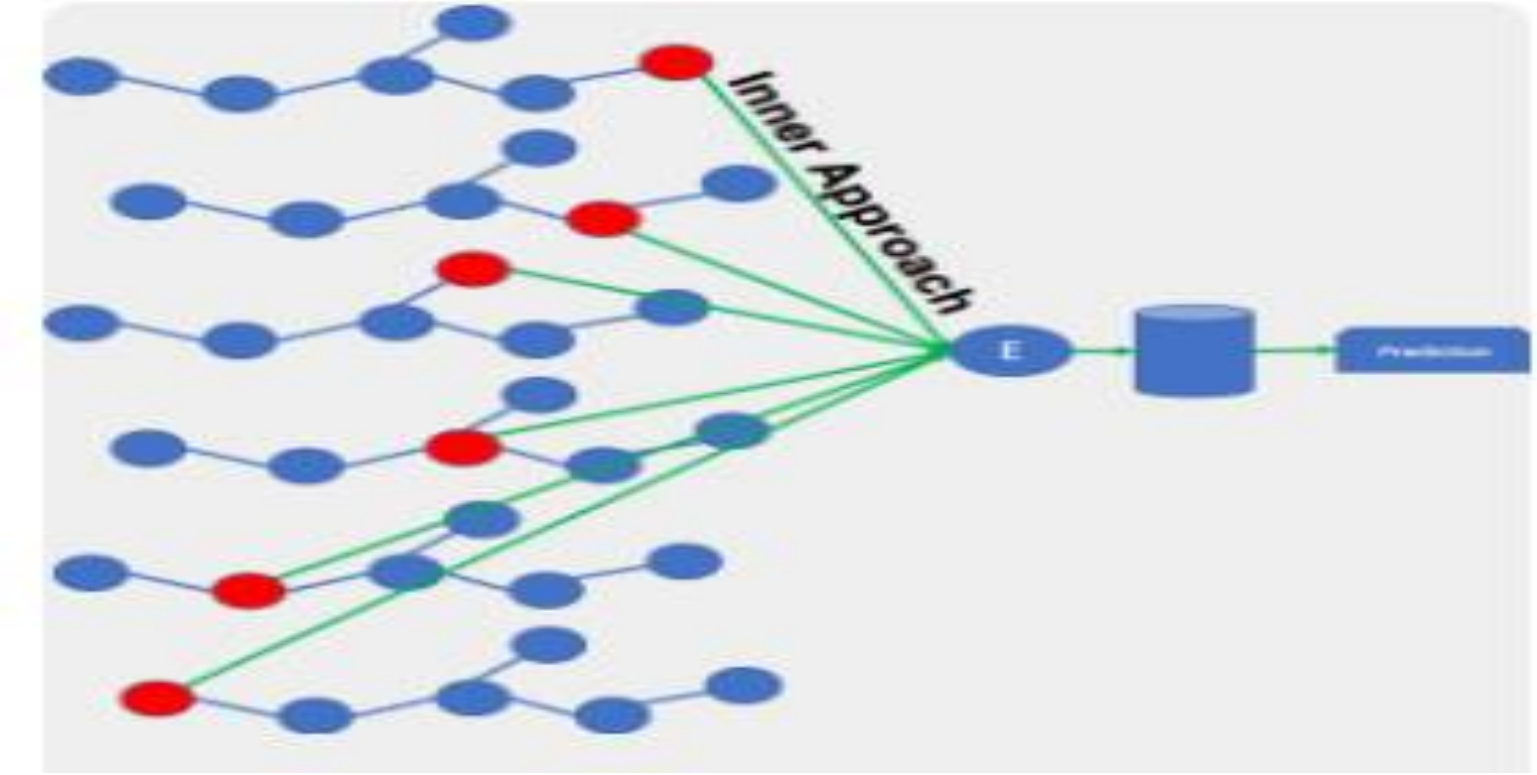
Parse tree for the sentence "The cat sat on the mat"



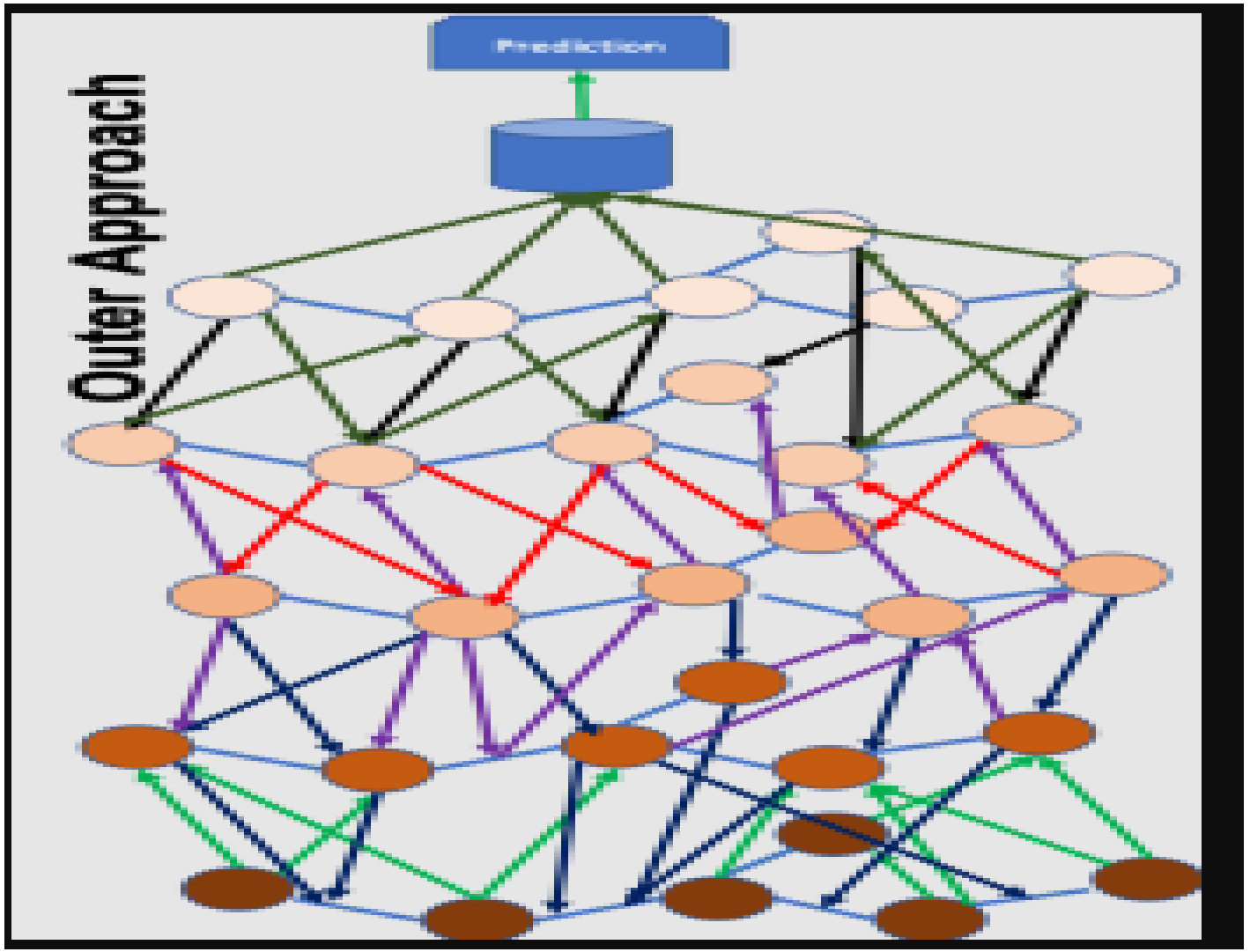
Recursive Neural Network (RvNNs)

- Tree-like topology allows branching connections and hierarchical structure.
- Recursive neural network approaches :
- ***Inner Approach & Outer Approach.***
- **Inner Approach:**
- The **Inner Approach** involves recursively applying the neural network starting from the innermost or deepest part of the structure and working outward.
- Essentially, the process begins with the most fundamental components (like the leaves of a tree) and combines them step by step to build higher-level representations.

- **Example in NLP:**
- For a sentence like "The cat sat on the mat,“.
- Inner Approach would start by combining the word representations of "The" and "cat" into a phrase representation (NP), then proceed to combine "sat" with "on the mat" (VP), and finally combine the NP and VP to represent the entire sentence.



- **Outer Approach**
- The **Outer Approach** starts from the root of the structure and recursively processes each part down to the leaves.
- The network first computes the overall structure's representation and then works its way down to refine and update the representations of the smaller components.
- **Example in NLP:**
- Outer Approach for the sentence "The cat sat on the mat,"
- The network would first form a general representation of the entire sentence (root node) and then recursively break it down into smaller components.
- Such as the noun phrase "The cat" and the verb phrase "sat on the mat," refining each part in context.


Prediction
Outer Approach

Inner Approach vs Outer Approach

Inner Approach	Outer Approach
This is about what's happening *inside* the network.	This is about how the network interacts with the *outside* world.
It focuses on how the network learns and processes information internally.	It focuses on how you prepare data for the network and how the network is used for specific tasks.
Think of it as understanding how the network combines smaller pieces of data (like words in a sentence) to create a bigger meaning.	Think of it as setting up the network for a particular job, like translating sentences or analyzing images.
It also involves how the network adjusts itself during training to improve performance.	It also involves checking how well the network is doing that job and making sure it's working correctly.

Applications of RvNN

- **1.Natural Language Processing (NLP)**
- **Sentiment Analysis**:
- RvNNs are used to analyze the sentiment of sentences or phrases by recursively combining word representations to form higher-level structures (like phrases and sentences) that capture the overall sentiment.
- **Syntax Parsing**:
- RvNNs can parse sentences into their syntactic components, generating parse trees that represent the grammatical structure of a sentence.

- **2. Machine Translation**
- RvNNs help in translating sentences from one language to another by understanding the hierarchical structure of sentences.
- **3. Image Processing and Scene Parsing**
- **Scene Understanding**:
- RvNNs are used to parse and understand scenes in images, breaking them down into hierarchies of objects and their relationships.
- This helps in tasks like object recognition and image segmentation.
- **3D Object Recognition**:
- Recognize and classify objects based on their hierarchical structure in three-dimensional space.

- **4. Visual Question Answering (VQA)**
- RvNNs can be used in tasks where visual and textual data are combined, such as answering questions about images.
- The recursive network processes the hierarchical structure of both the image scene and the question to generate an appropriate answer.
- **5. Speech Recognition and Processing**
- RvNNs can be employed to model the hierarchical structure of speech signals, capturing the relationships between phonemes, syllables, words, and sentences to improve speech recognition accuracy.
- **6. Document Summarization**
- RvNNs help by understanding the hierarchical structure of documents, identifying key sections, and generating coherent summaries that capture the essence of the content.

Assignment 3

Q1. What is RNN? Explain Types.

Q2. Compare Feedforward Neural Network and Recurrent Neural Network.

Q3. Interpret how LSTM proves efficient over RNN?

Q4. Explain Encoder Decoder Architecture.

Q5. Explain the Applications and drawbacks of RNN.

Q6. What is RvNN? Explain.

Q7. Compare Inner Approach vs Outer Approach RvNN.

Q8. What are the applications of RvNN.