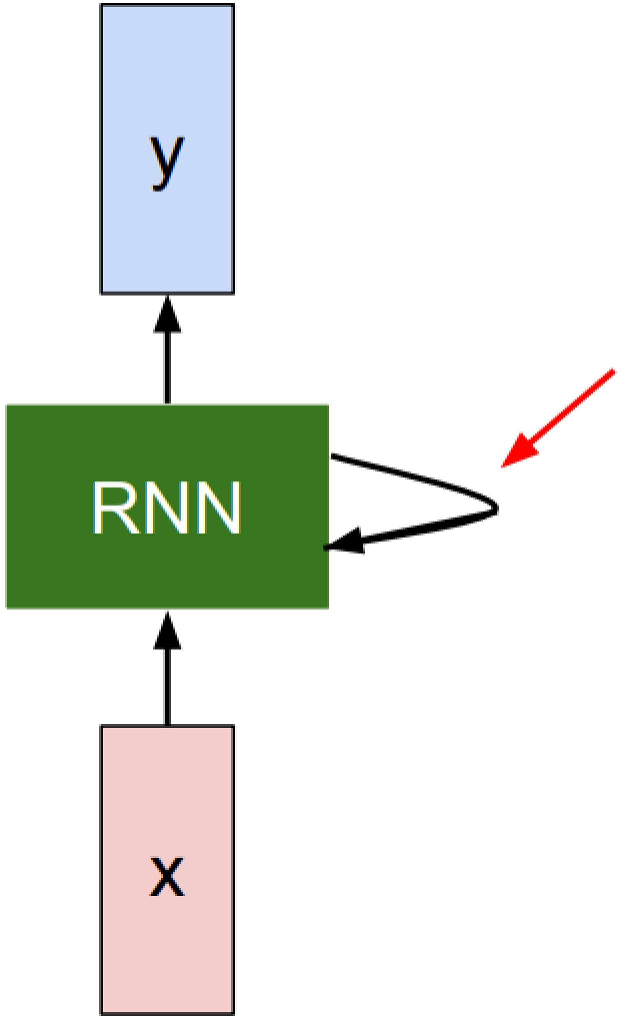
Sequence models

Use case:

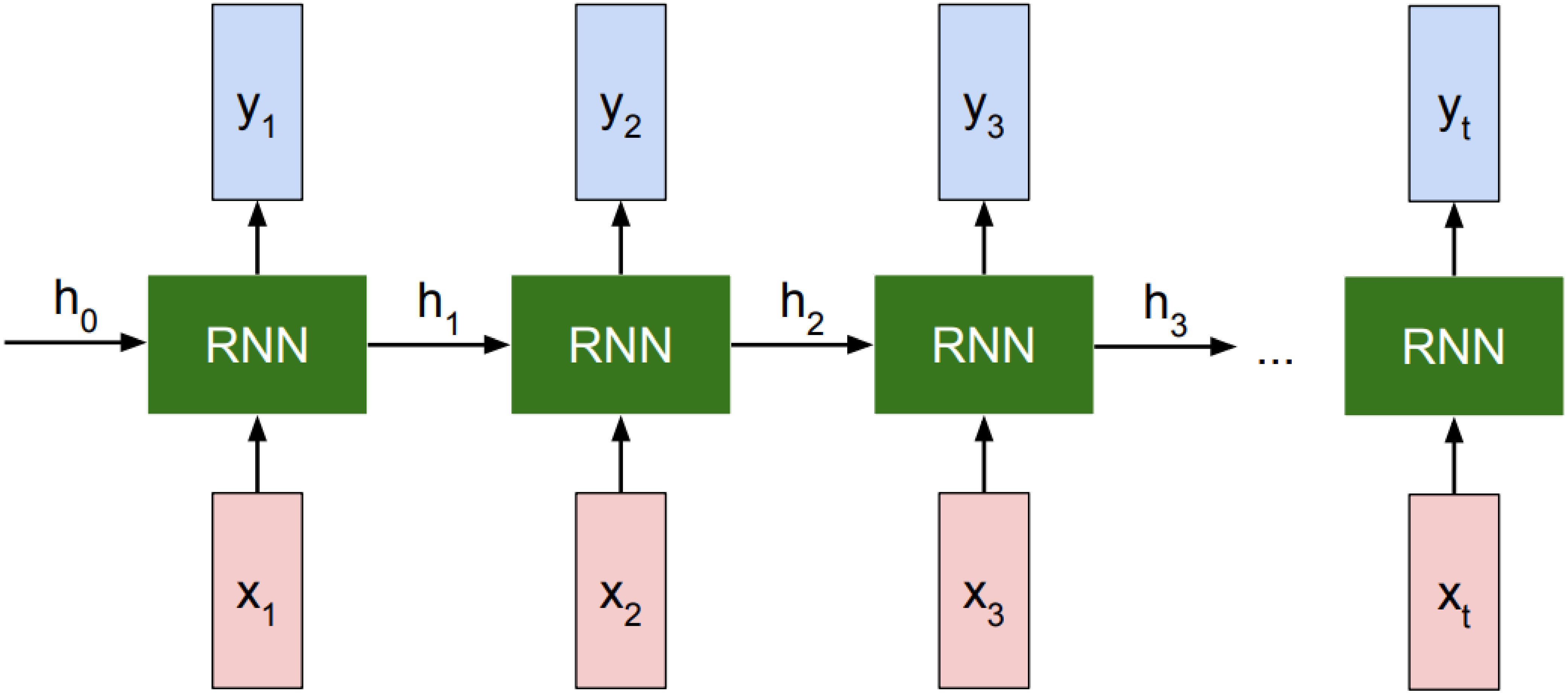
- Sentiment analysis
- Time-series forecasting
- Machine translation
- NLP tasks

Recurrent Neural Network



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Unrolled RNN



RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

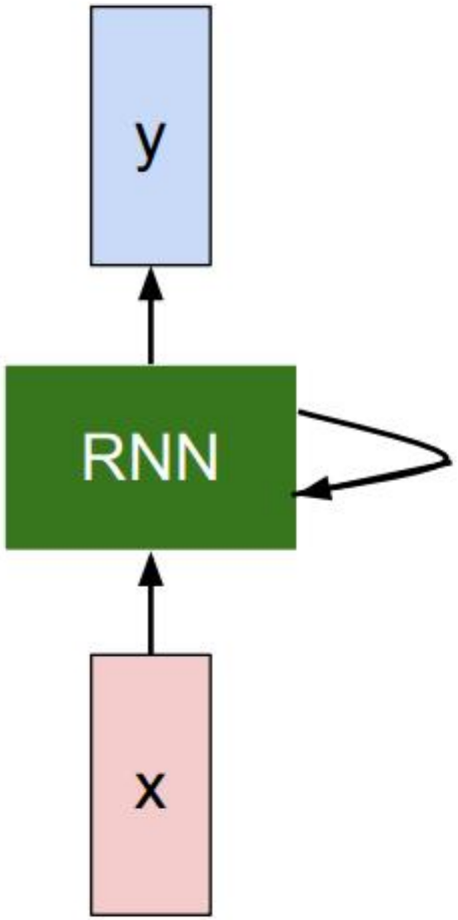
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W



RNN output generation

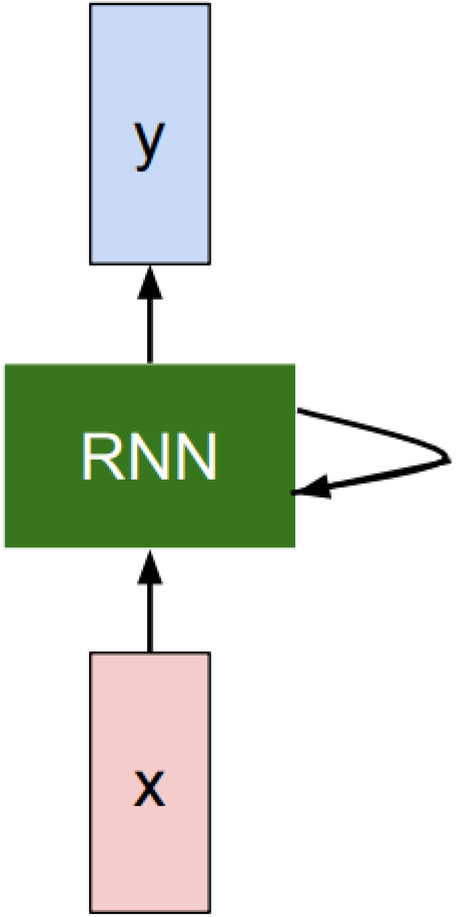
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$\boxed{y_t} = \boxed{f_{W_{hy}}}(\boxed{h_t})$$

output

new state

another function with parameters W_o

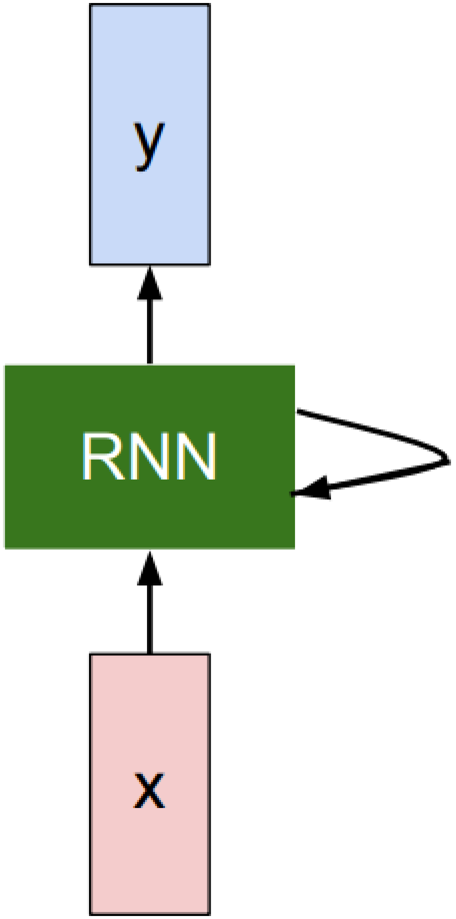


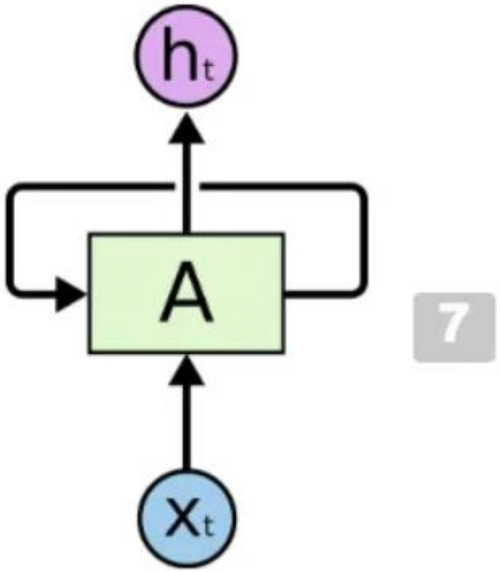
Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

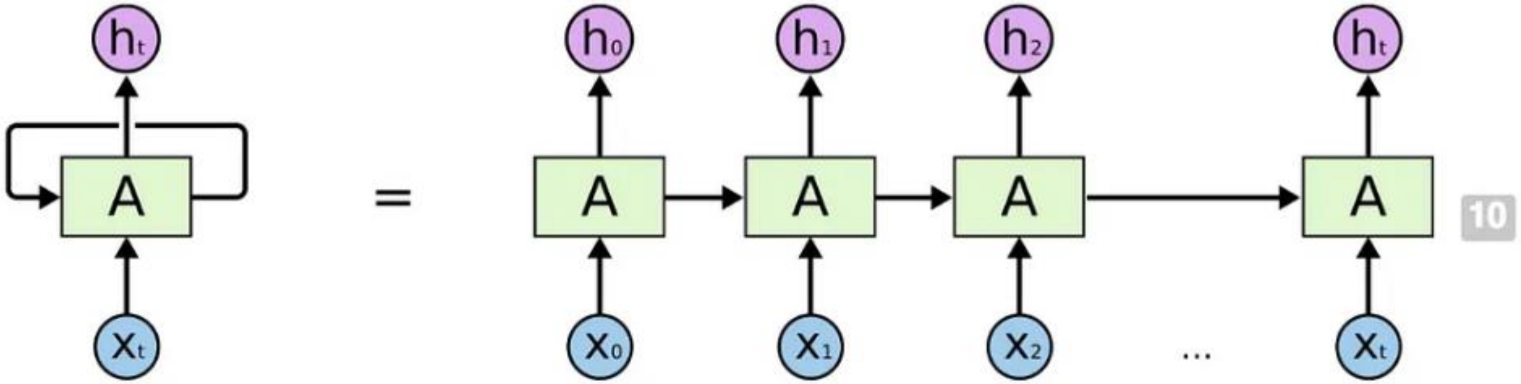
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.

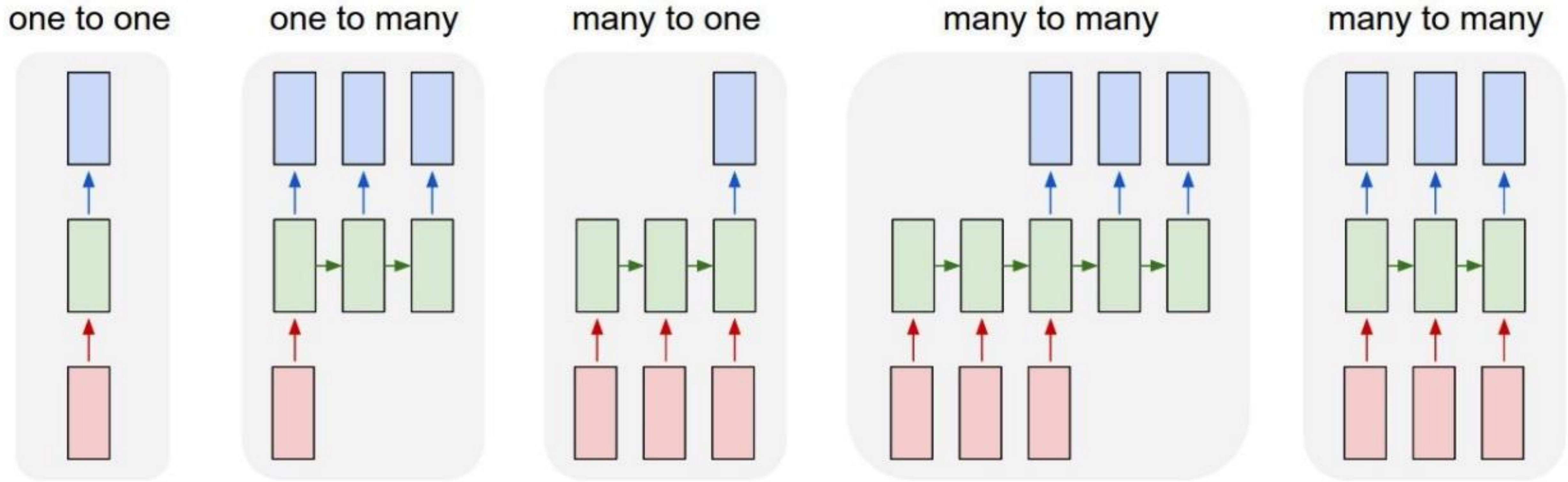




Recurrent Neural Networks have loops.



Recurrent Neural Networks: Process Sequences



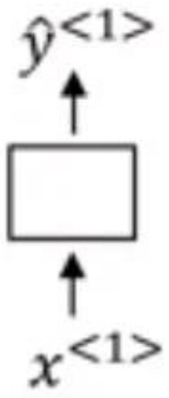
one to one

one to many

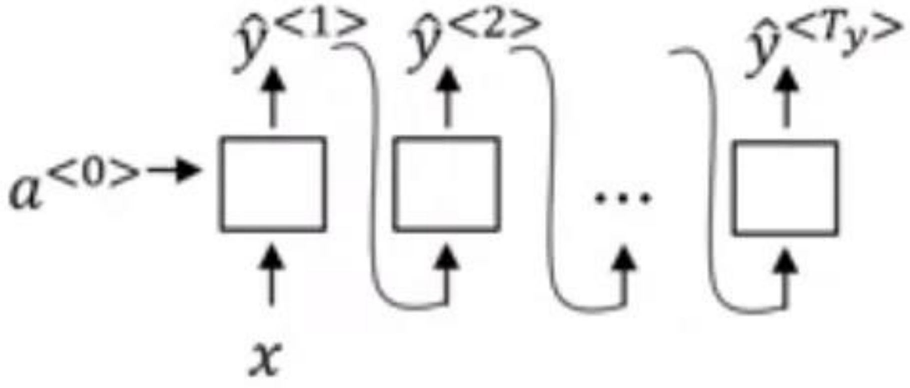
many to one

many to many

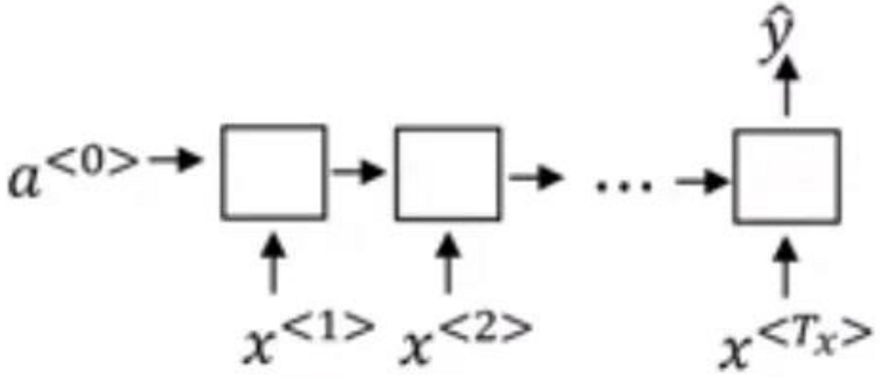
many to many



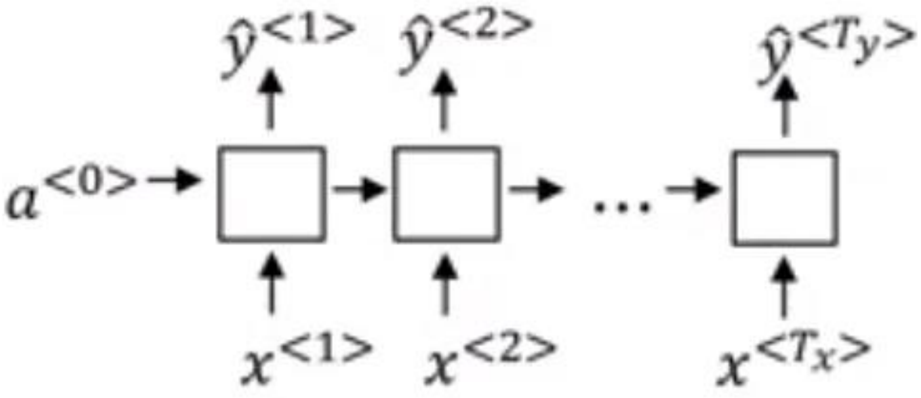
One to one



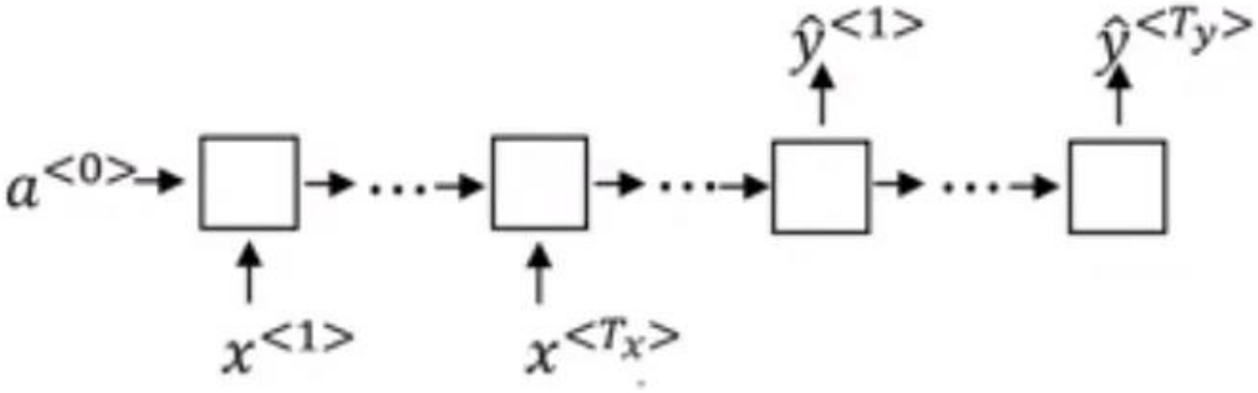
One to many



Many to one

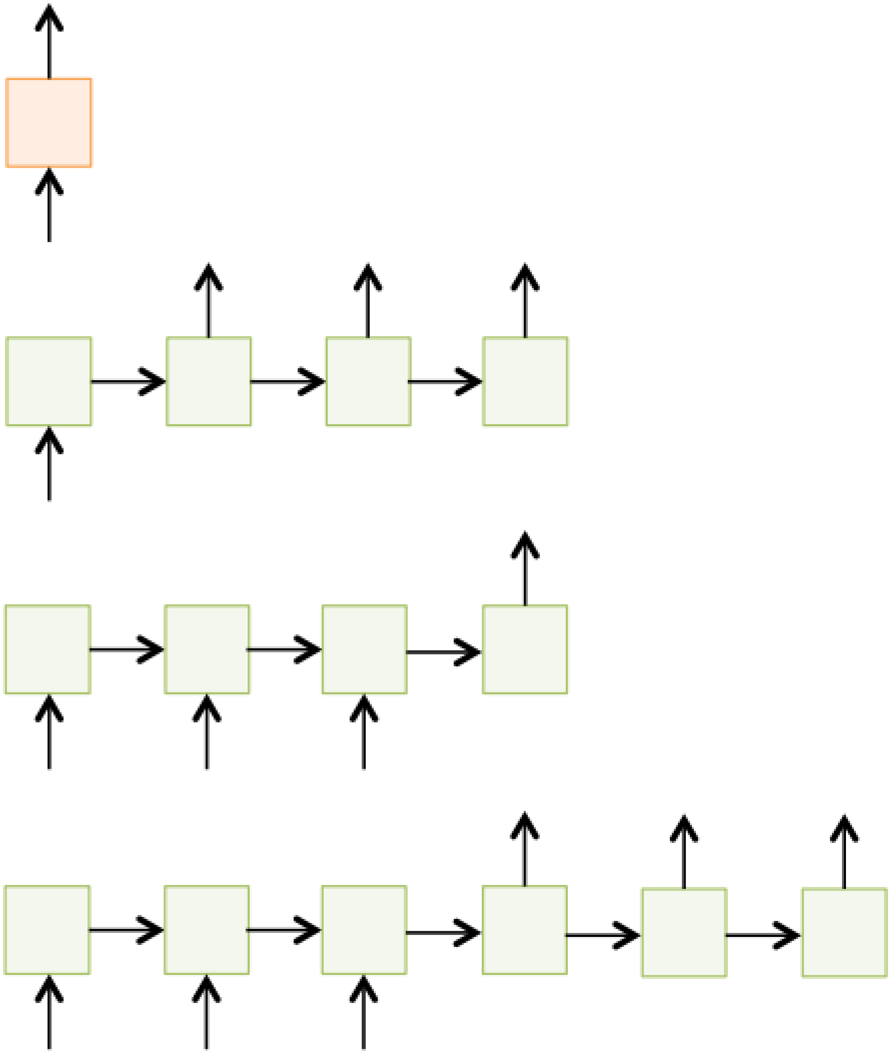


Many to many



Many to many

Input – Output Scenarios



Single - Single	Feed-forward Network
Single - Multiple	Image Captioning
Multiple - Single	Sentiment Classification
Multiple - Multiple	Translation

Sentiment Classification

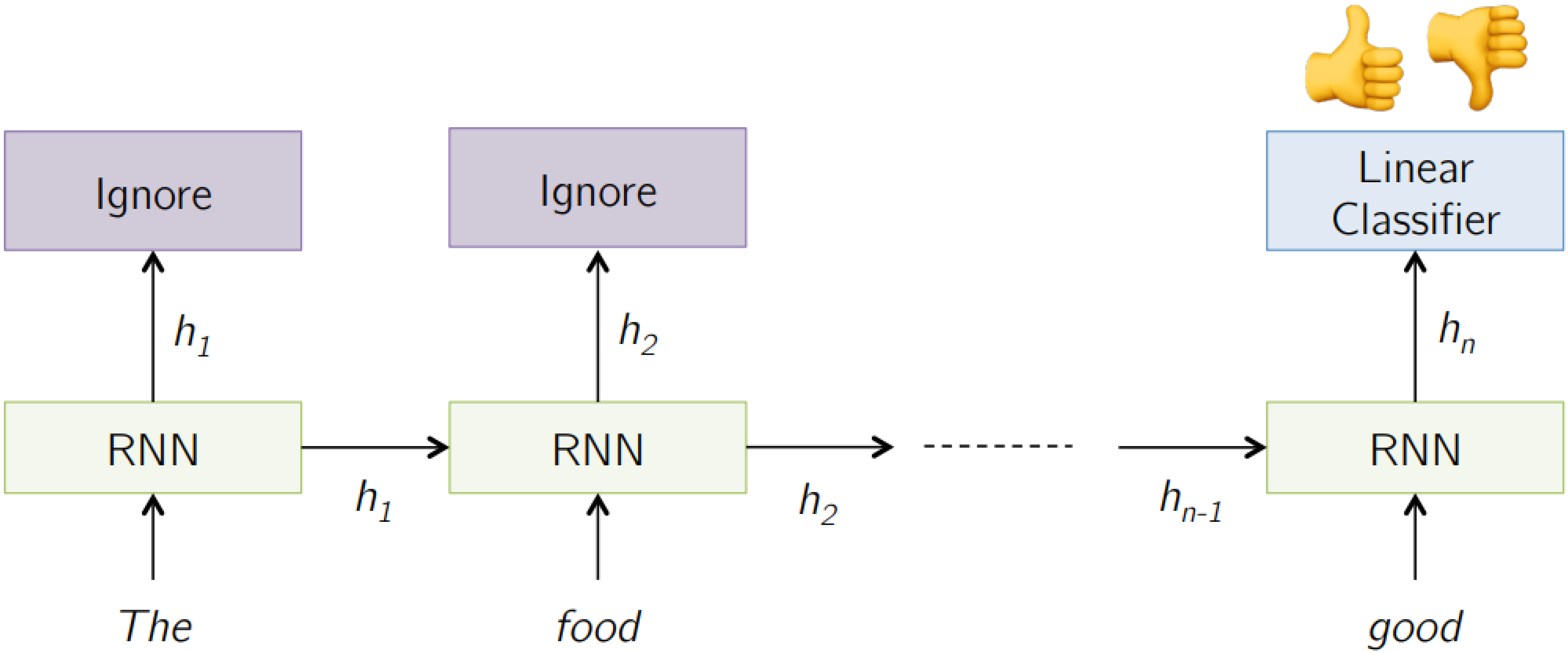
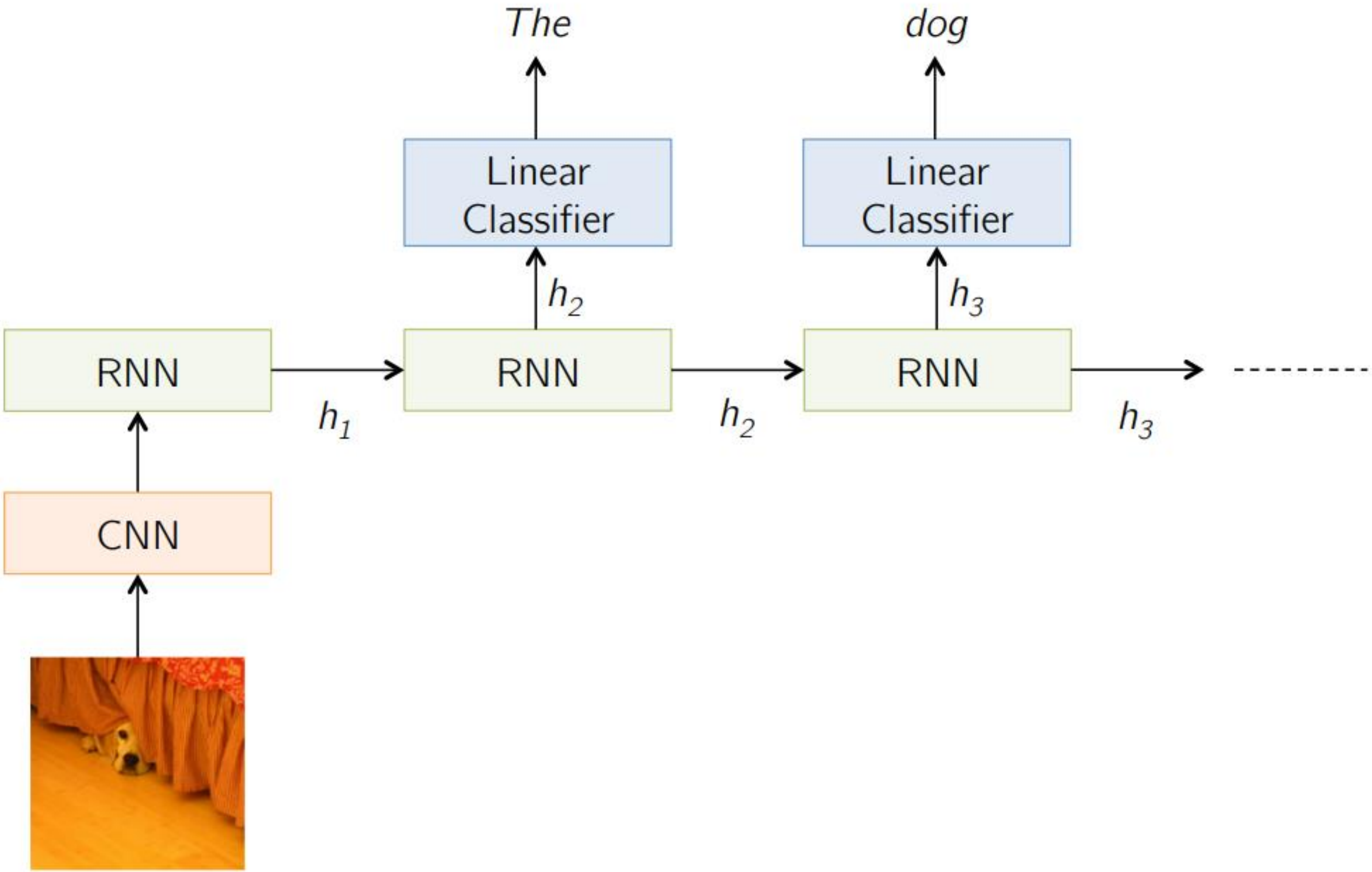


Image Captioning



RNN Outputs: Image Captions

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

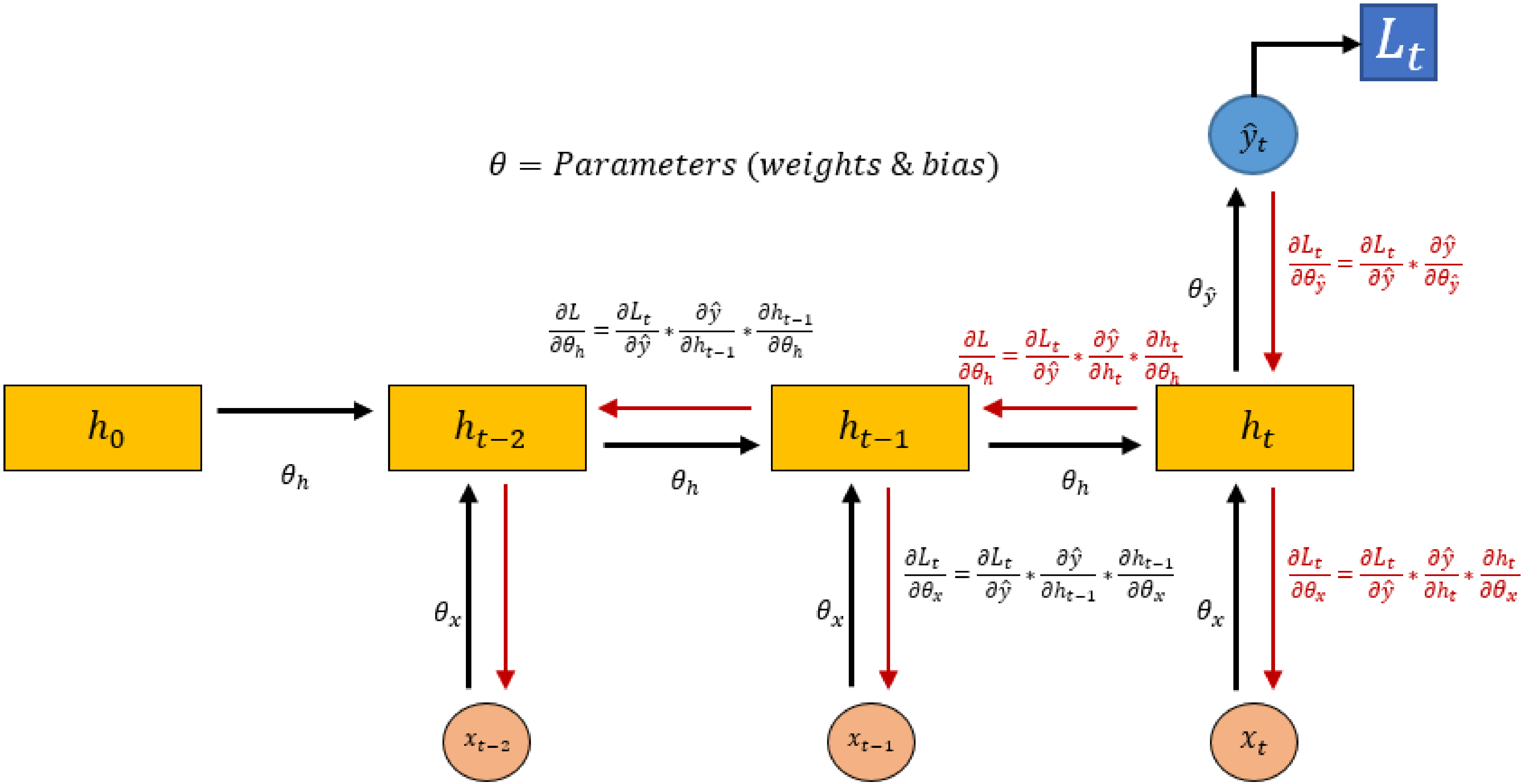
A close up of a cat laying on a couch.

Features of RNN

- The sequence nature of features is handled by computing a forward pass by propagating the state (hidden) across the time-dependent features.
- Forward pass → Unroll in time
- Backpropagation through time
- Different configurations depending upon the application
- Now, even though RNNs are quite powerful, they suffer from **Vanishing gradient problem** which hinders them from using long-term information, they are good for storing memory of 3-4 instances of past iterations but a larger number of instances don't provide good results so we don't just use regular RNNs. Instead, we use a better variation of RNNs:

  Long Short Term Networks(LSTM).

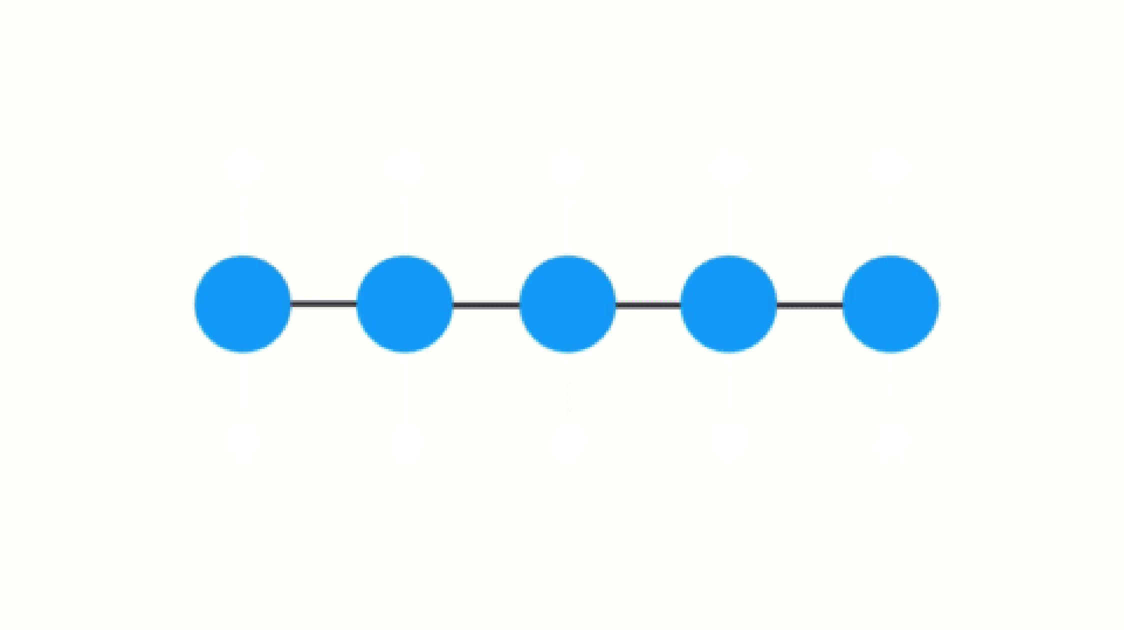
$\theta = Parameters\ (weights\ \&\ bias)$



L_t
$\hat{y}_t$
$\theta_{\hat{y}}$
$\frac{\partial L_t}{\partial \theta_{\hat{y}}} = \frac{\partial L_t}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \theta_{\hat{y}}}$
$\frac{\partial L}{\partial \theta_h} = \frac{\partial L_t}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_{t-1}} * \frac{\partial h_{t-1}}{\partial \theta_h}$
$\frac{\partial L}{\partial \theta_h} = \frac{\partial L_t}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_t} * \frac{\partial h_t}{\partial \theta_h}$
h_0
h_{t-2}
h_{t-1}
h_t
θ_h
θ_h
θ_h
θ_x
θ_x
θ_x
$\frac{\partial L_t}{\partial \theta_x} = \frac{\partial L_t}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_{t-1}} * \frac{\partial h_{t-1}}{\partial \theta_x}$
$\frac{\partial L_t}{\partial \theta_x} = \frac{\partial L_t}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_t} * \frac{\partial h_t}{\partial \theta_x}$
x_{t-2}
x_{t-1}
x_t

The main limitation of RNNs is that RNNs can’t remember very long sequences and get into the problem of vanishing gradient.

- What is the vanishing gradient problem?



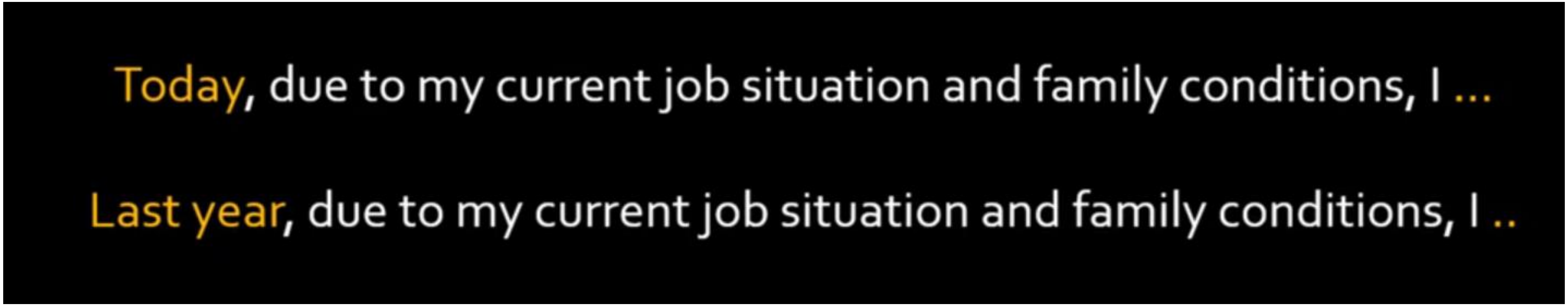
What is Vanishing Gradient problem?

- The vanishing gradient problem is a difficulty found in training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, each of the neural network's weights receives an update proportional to the partial derivative of the error function concerning the current weight in each training iteration.
- The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may stop the neural network from further training. As one example of the problem cause, traditional activation functions such as the hyperbolic tangent function have gradients in the range (0, 1), and backpropagation computes gradients by the chain rule. This has the effect of multiplying n of these small numbers to compute gradients of the "front" layers in an n-layer network, meaning that the gradient (error signal) decreases exponentially with n while the front layers train very slowly.

The Problem, Short-term Memory

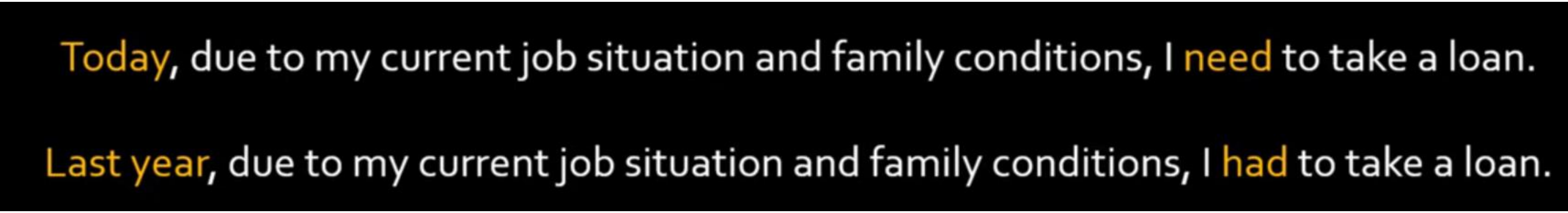
- Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

- During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem occurs when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.



Today, due to my current job situation and family conditions, I ...

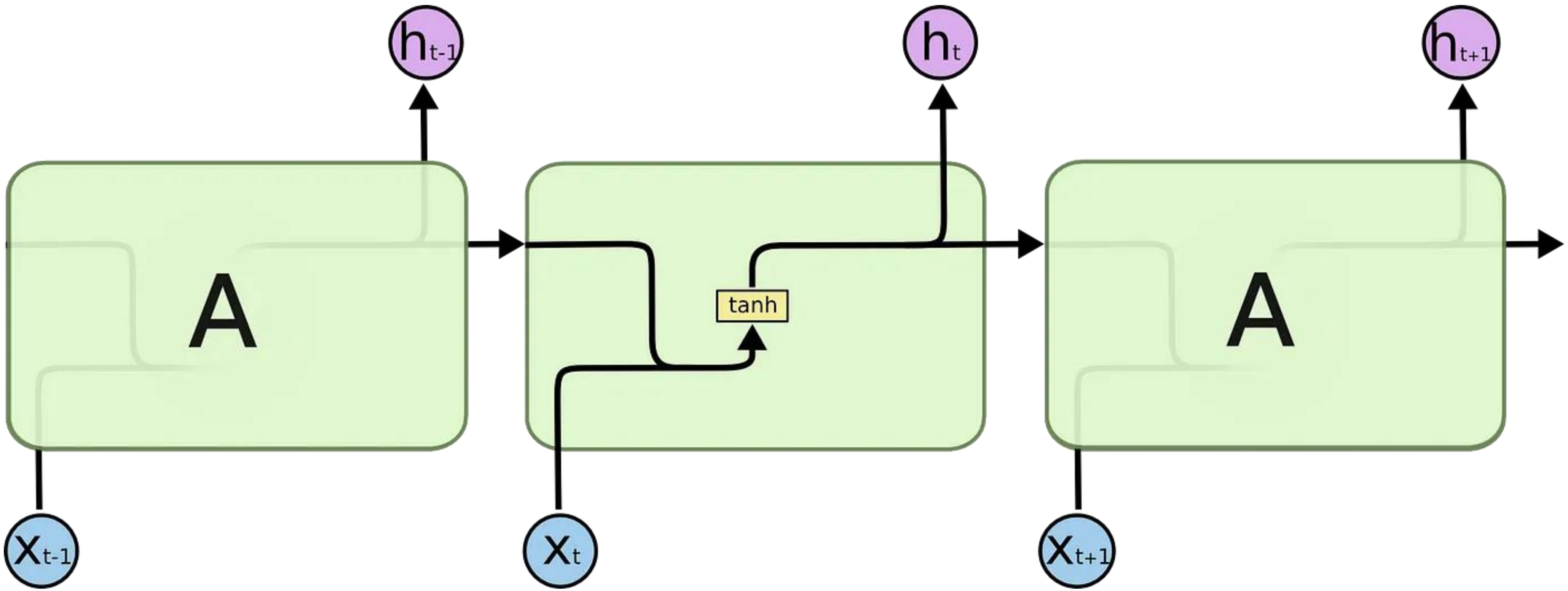
Last year, due to my current job situation and family conditions, I ..

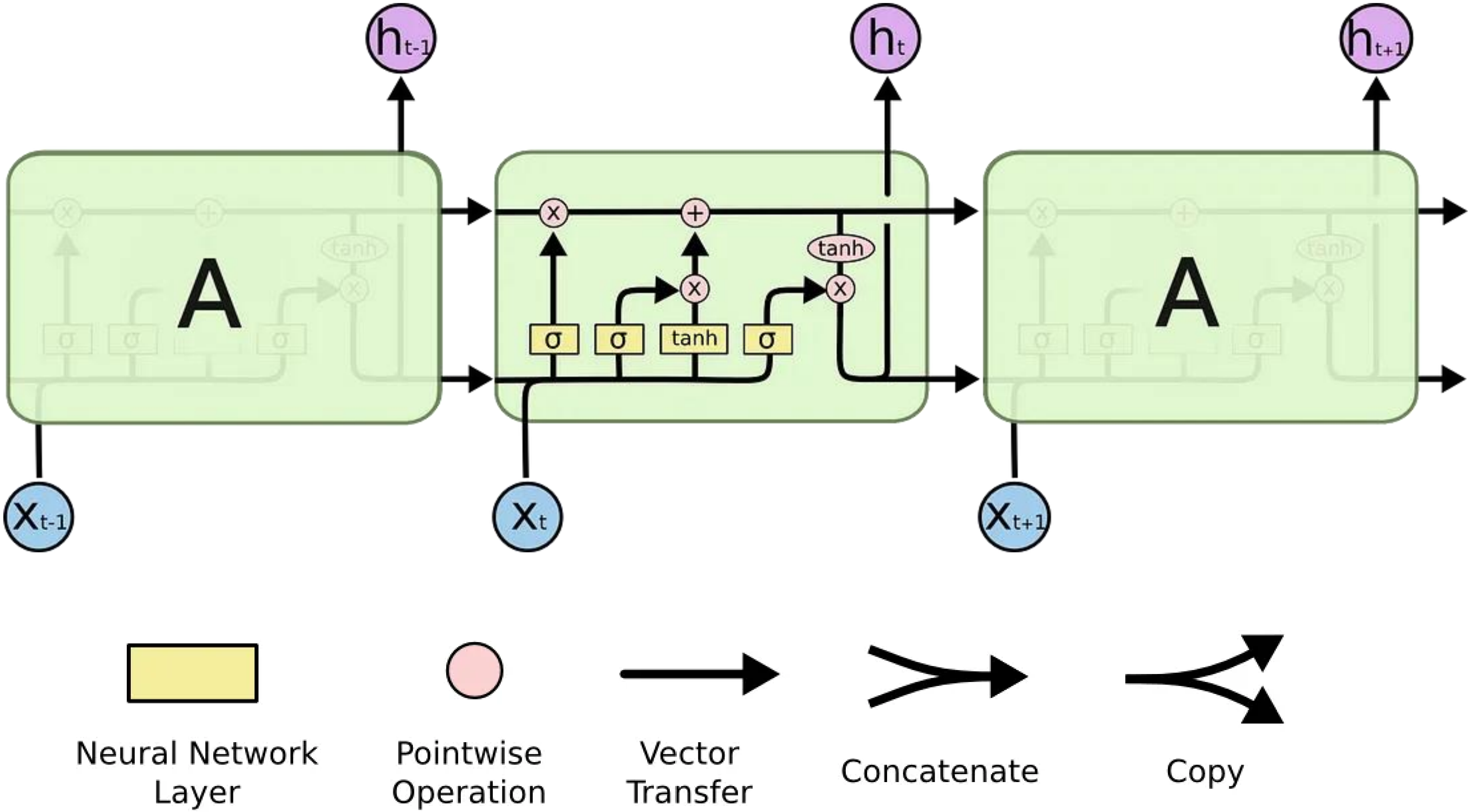


Today, due to my current job situation and family conditions, I need to take a loan.

Last year, due to my current job situation and family conditions, I had to take a loan.

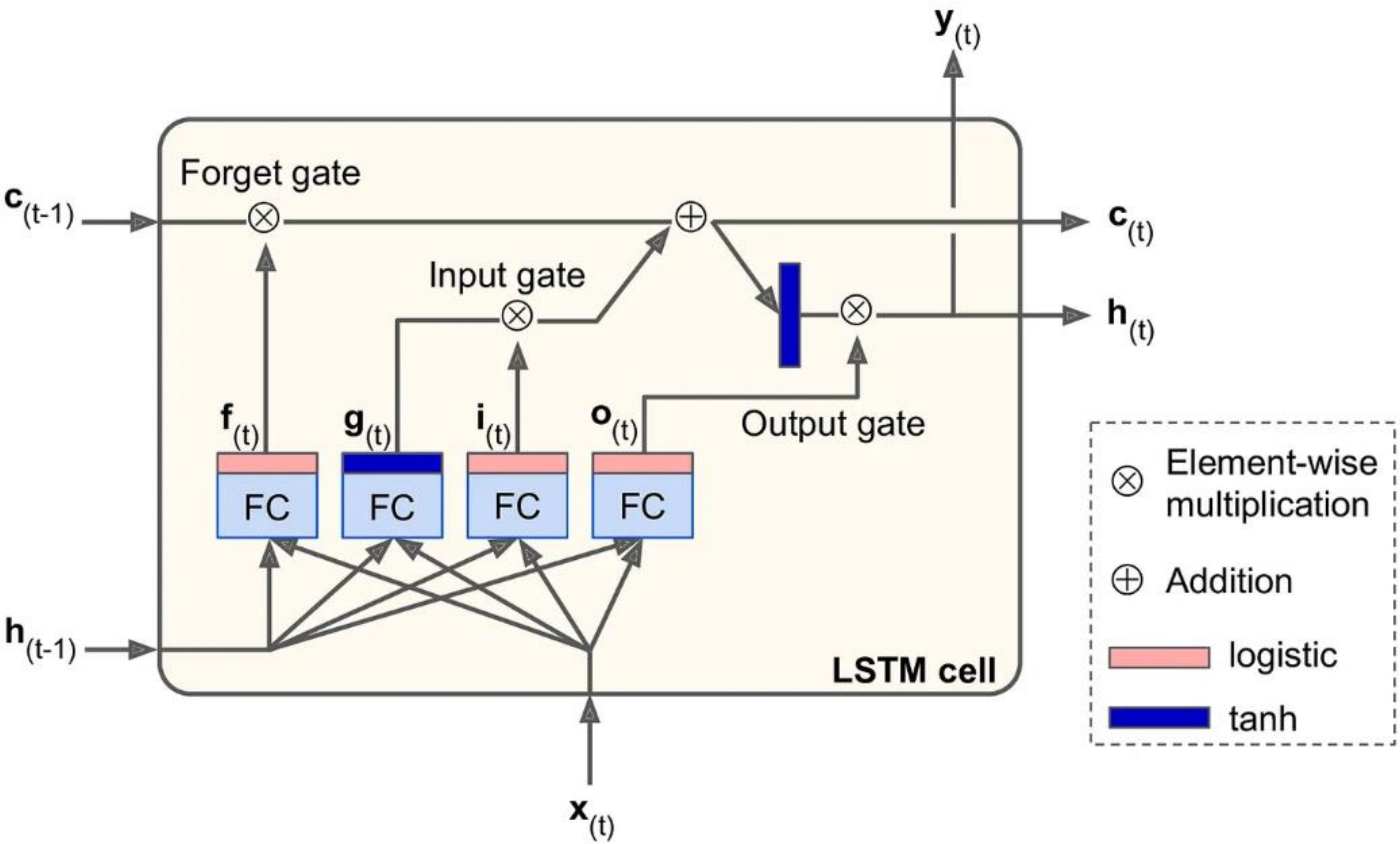
Simple RNN

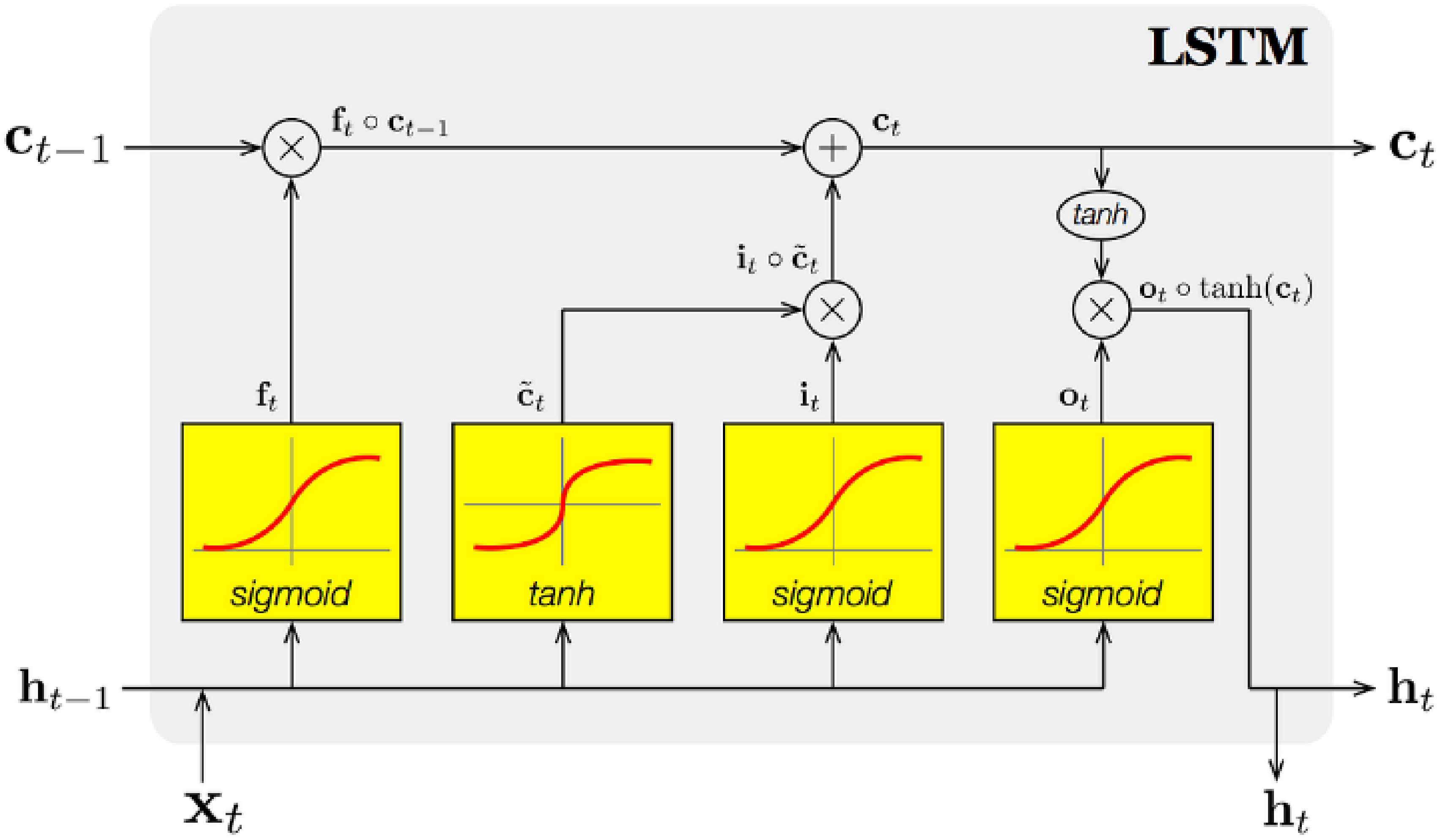




h_{t-1}
h_t
h_{t+1}
A
×
+
tanh
σ
σ
tanh
σ
×
×
A
x_{t-1}
x_t
x_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

LSTM Architecture





Gating variables

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

Candidate (memory) cell state

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$

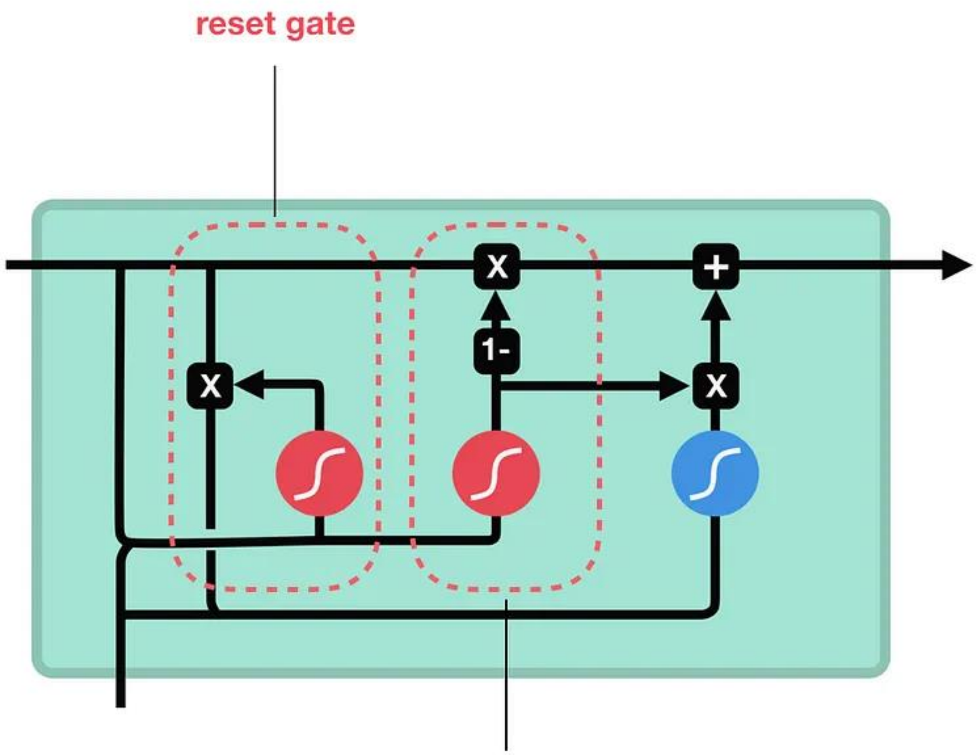
Cell & Hidden state

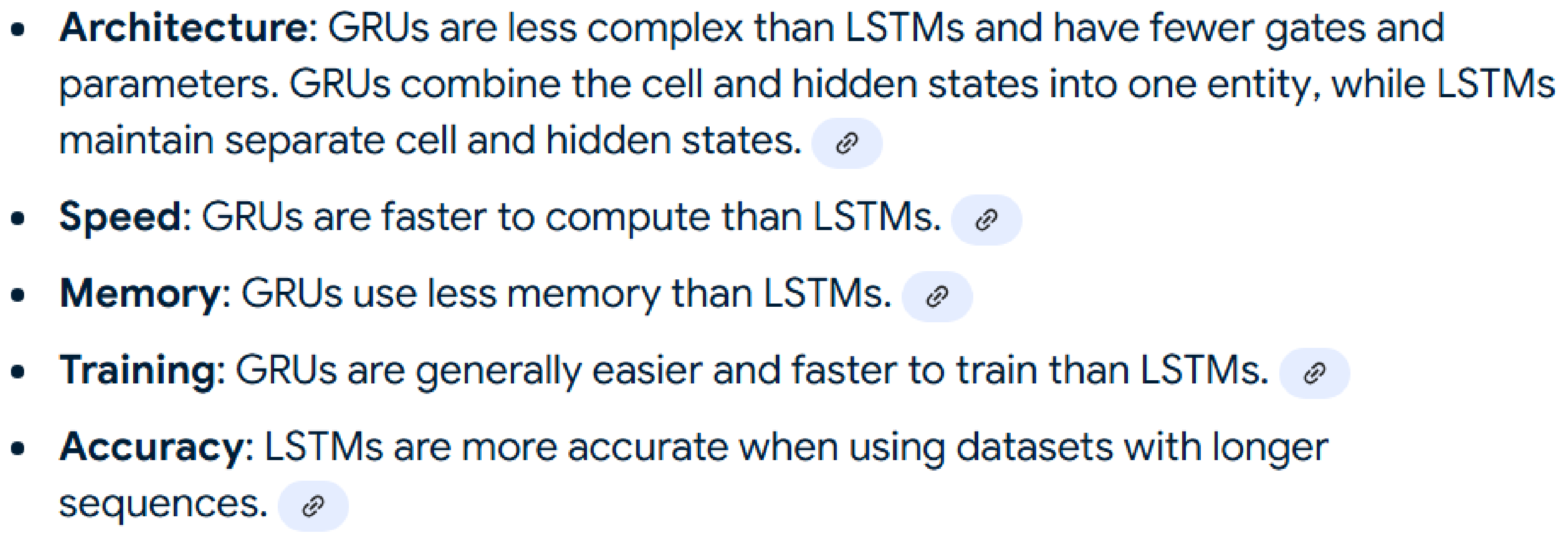
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

LSTM parameter number = 4 × ((x + h) × h +h)

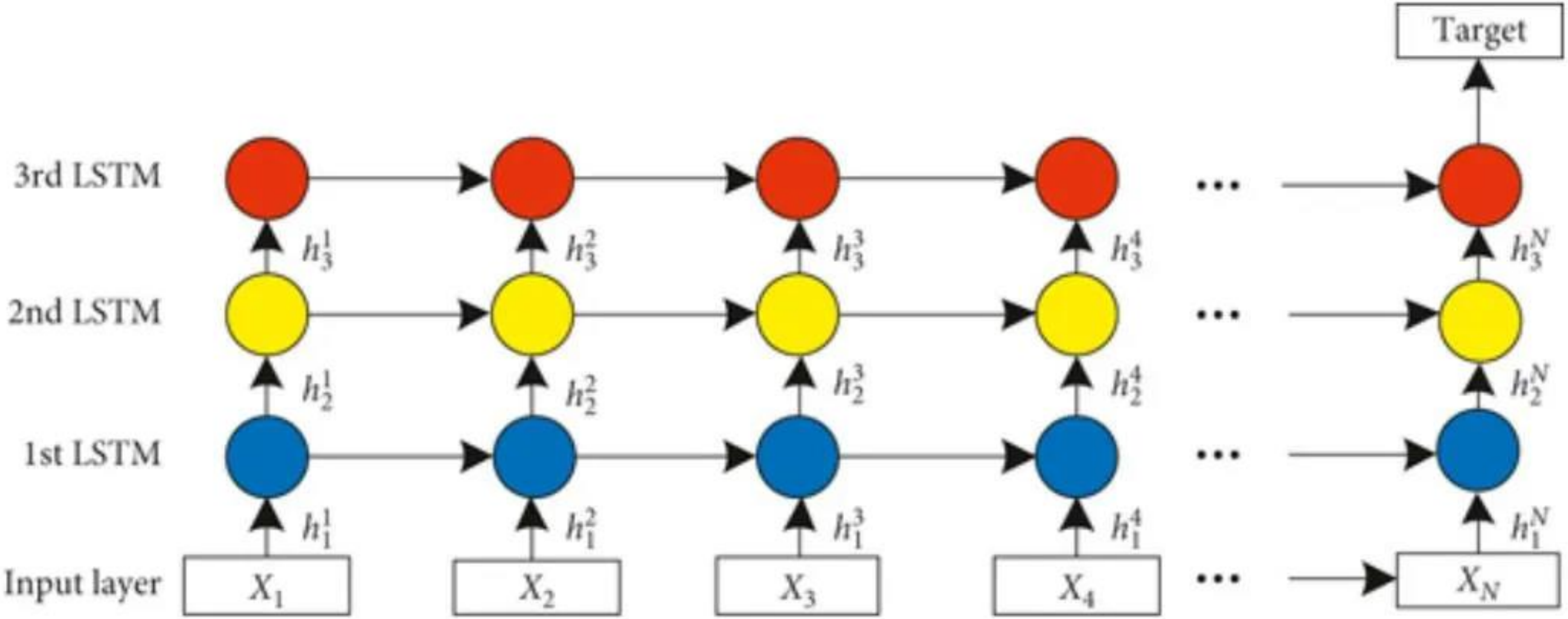
Gated Recurrent Units (GRU)

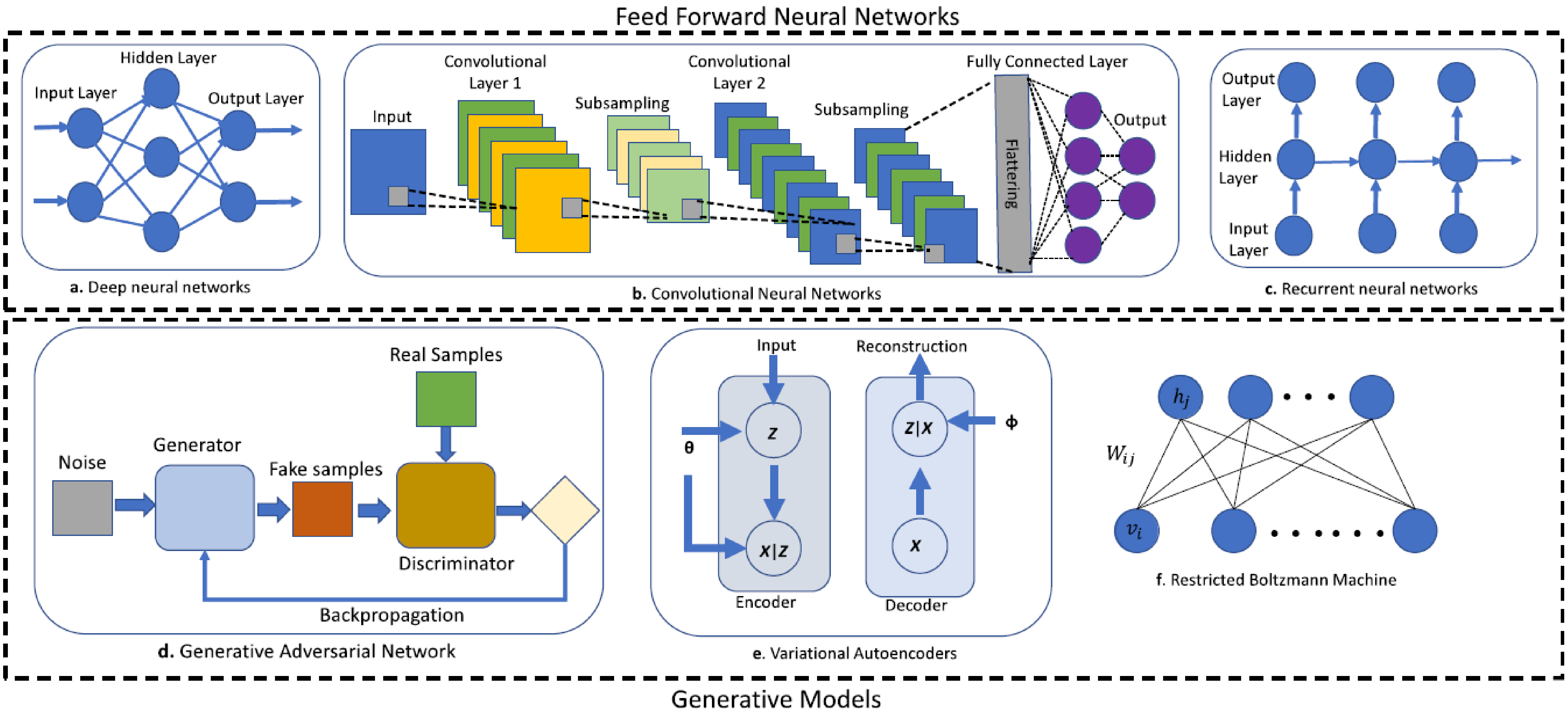




- **Architecture**: GRUs are less complex than LSTMs and have fewer gates and parameters. GRUs combine the cell and hidden states into one entity, while LSTMs maintain separate cell and hidden states.
- **Speed**: GRUs are faster to compute than LSTMs.
- **Memory**: GRUs use less memory than LSTMs.
- **Training**: GRUs are generally easier and faster to train than LSTMs.
- **Accuracy**: LSTMs are more accurate when using datasets with longer sequences.

Stacked RNNs





a. Deep neural networks

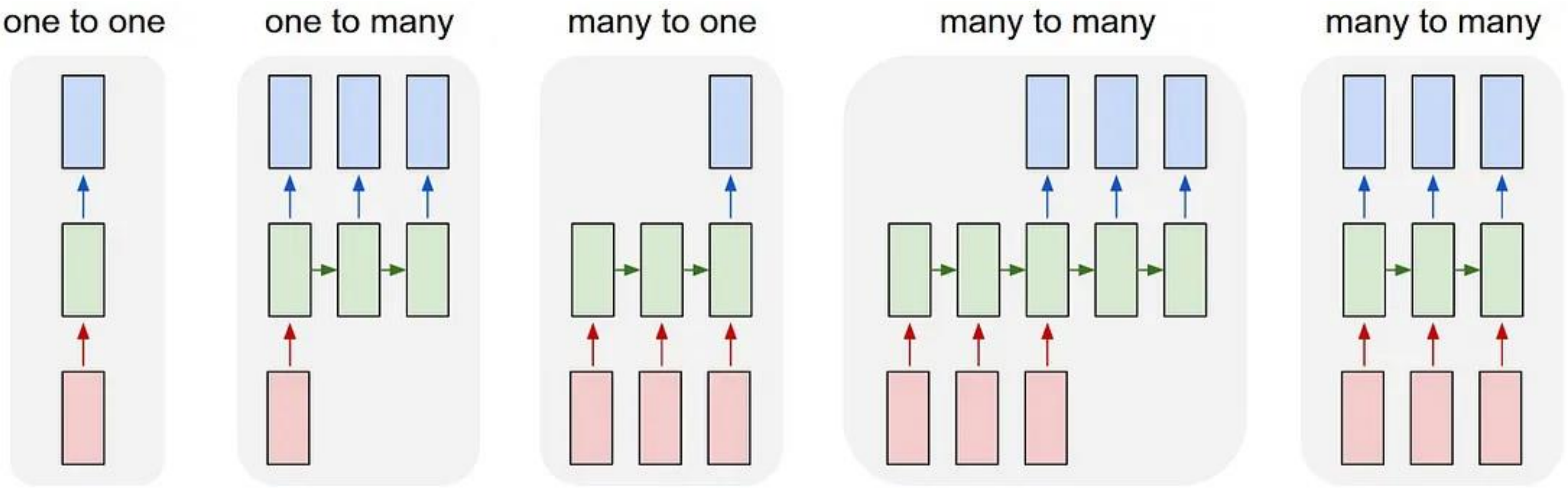
b. Convolutional Neural Networks

c. Recurrent neural networks

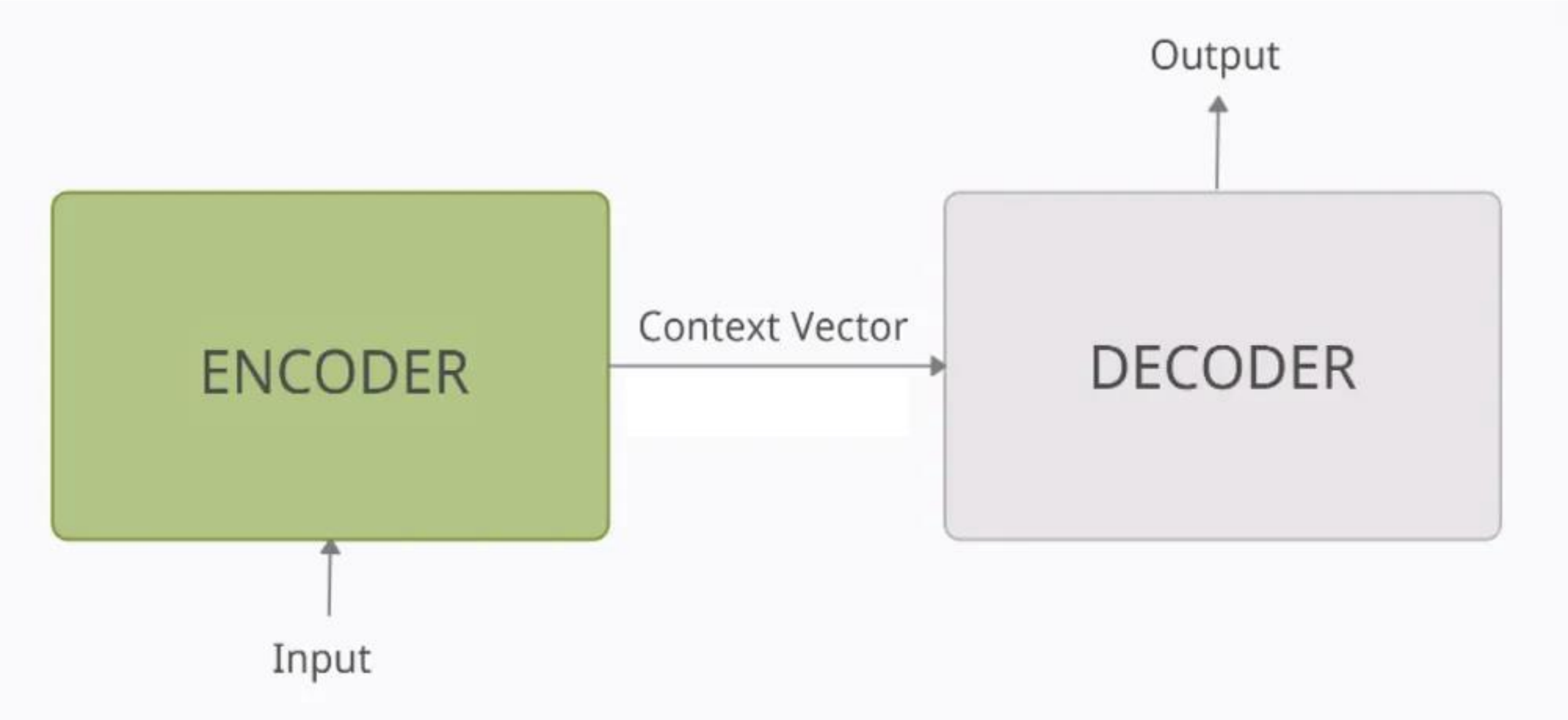
d. Generative Adversarial Network

e. Variational Autoencoders

f. Restricted Boltzmann Machine

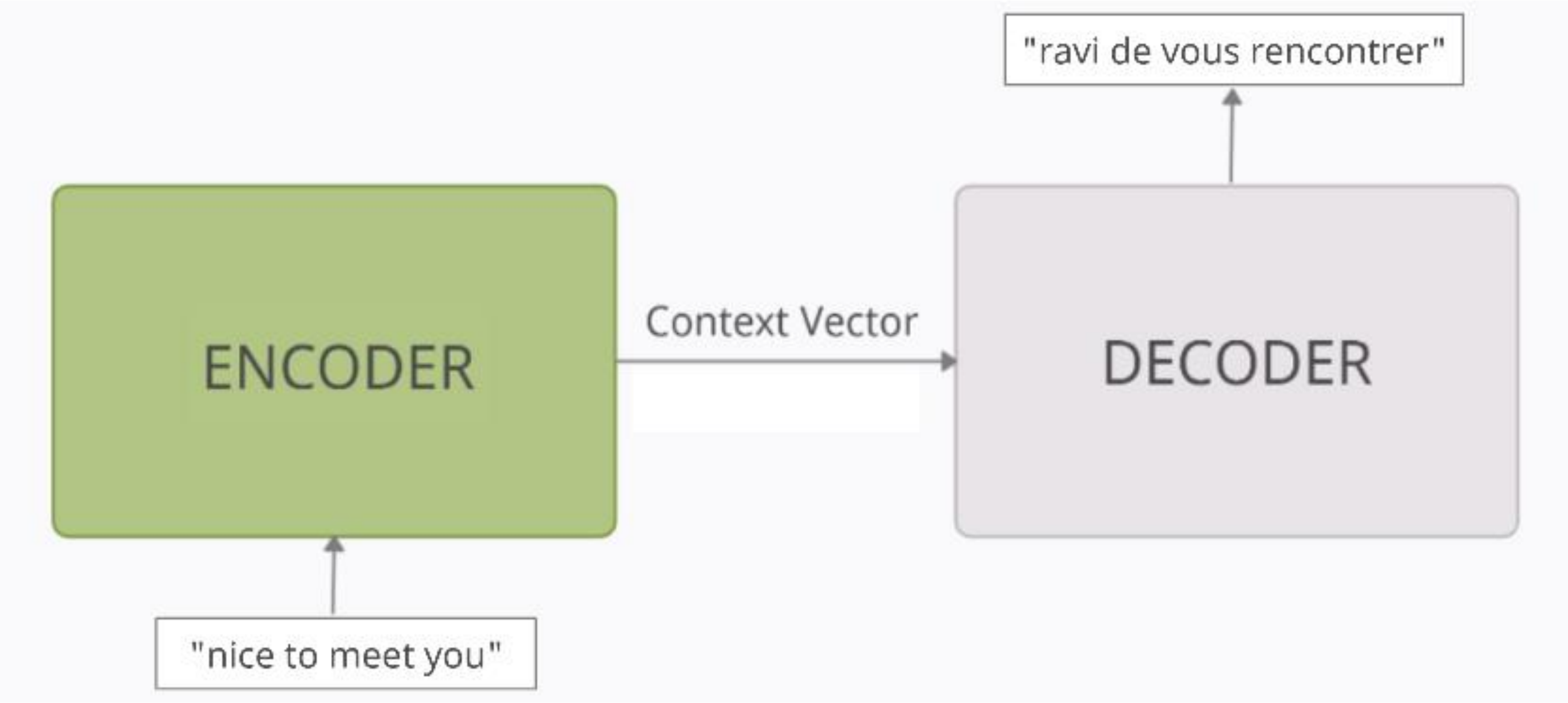

one to one
one to many
many to one
many to many
many to many

Sequence-to-Sequence Problems

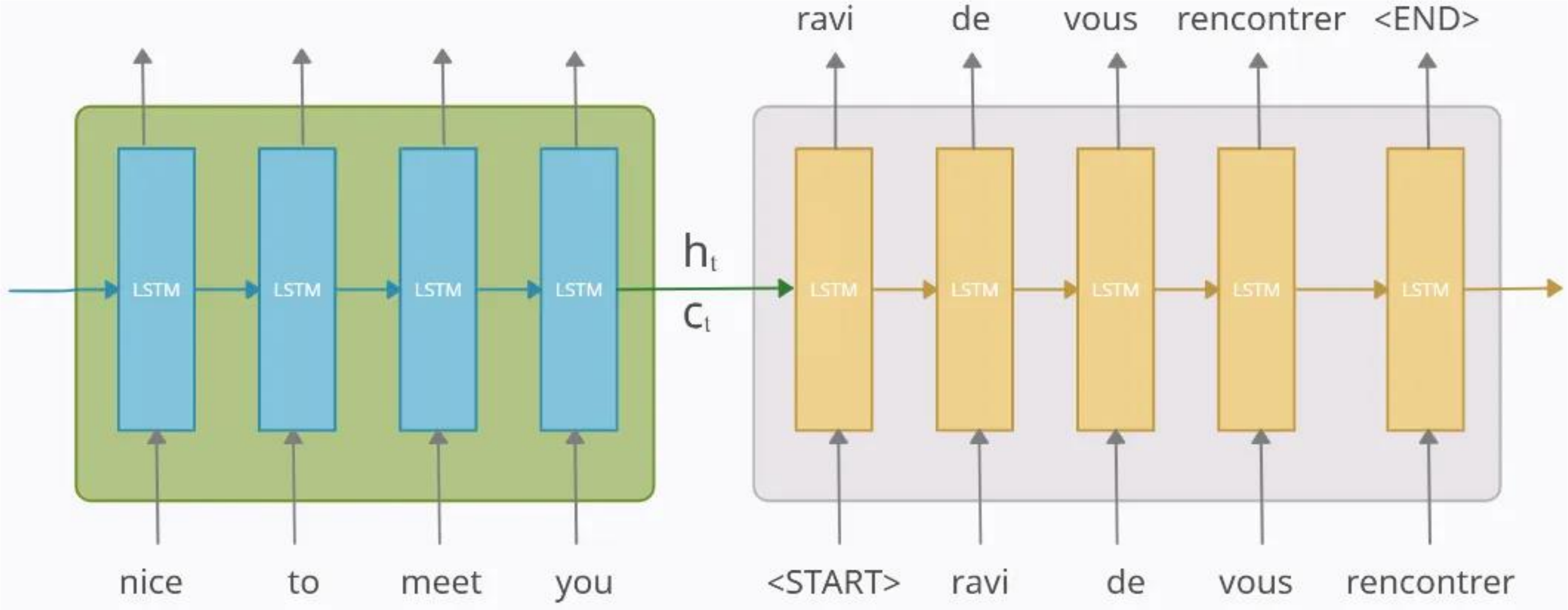


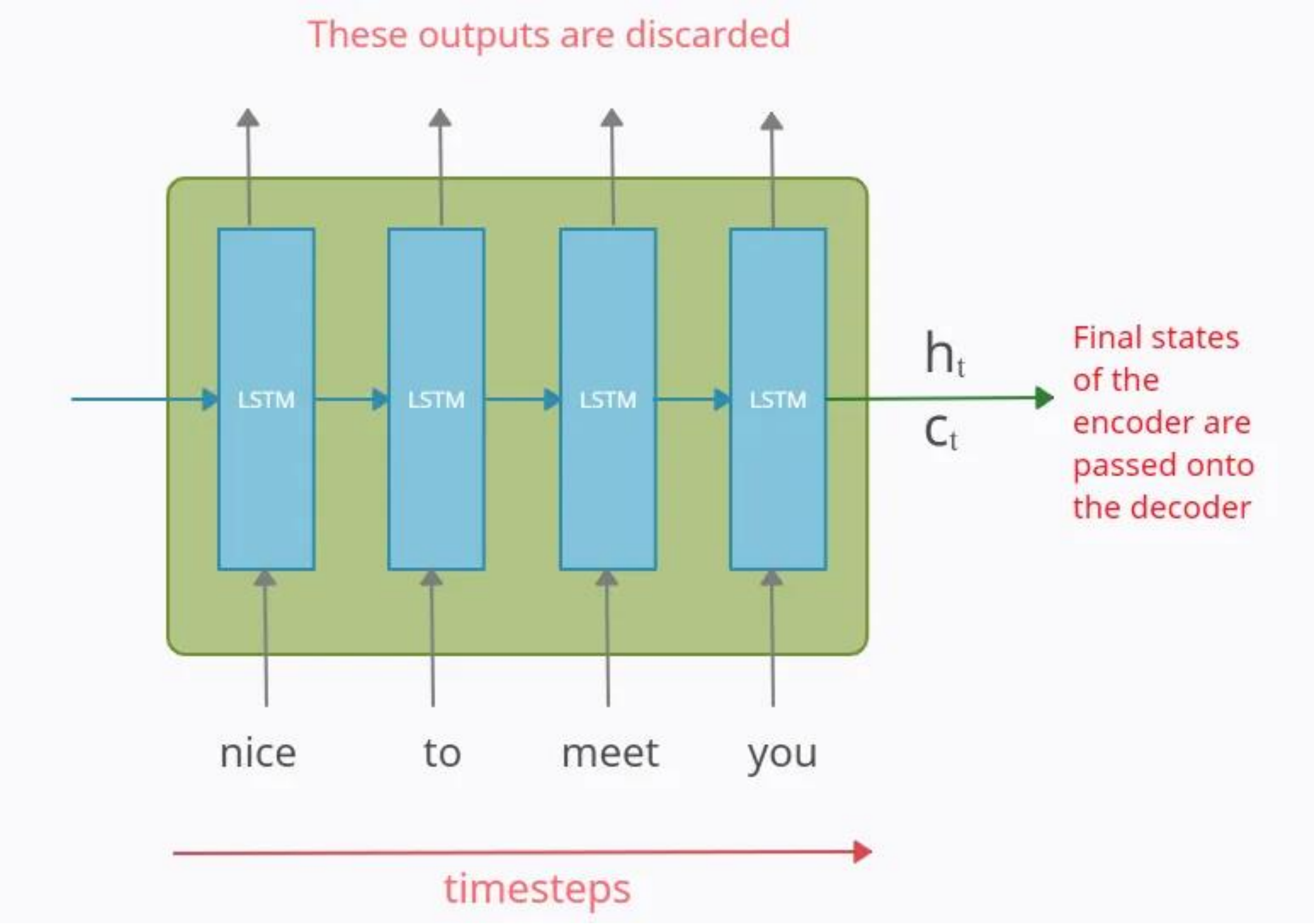
Sequence-to-Sequence Problems

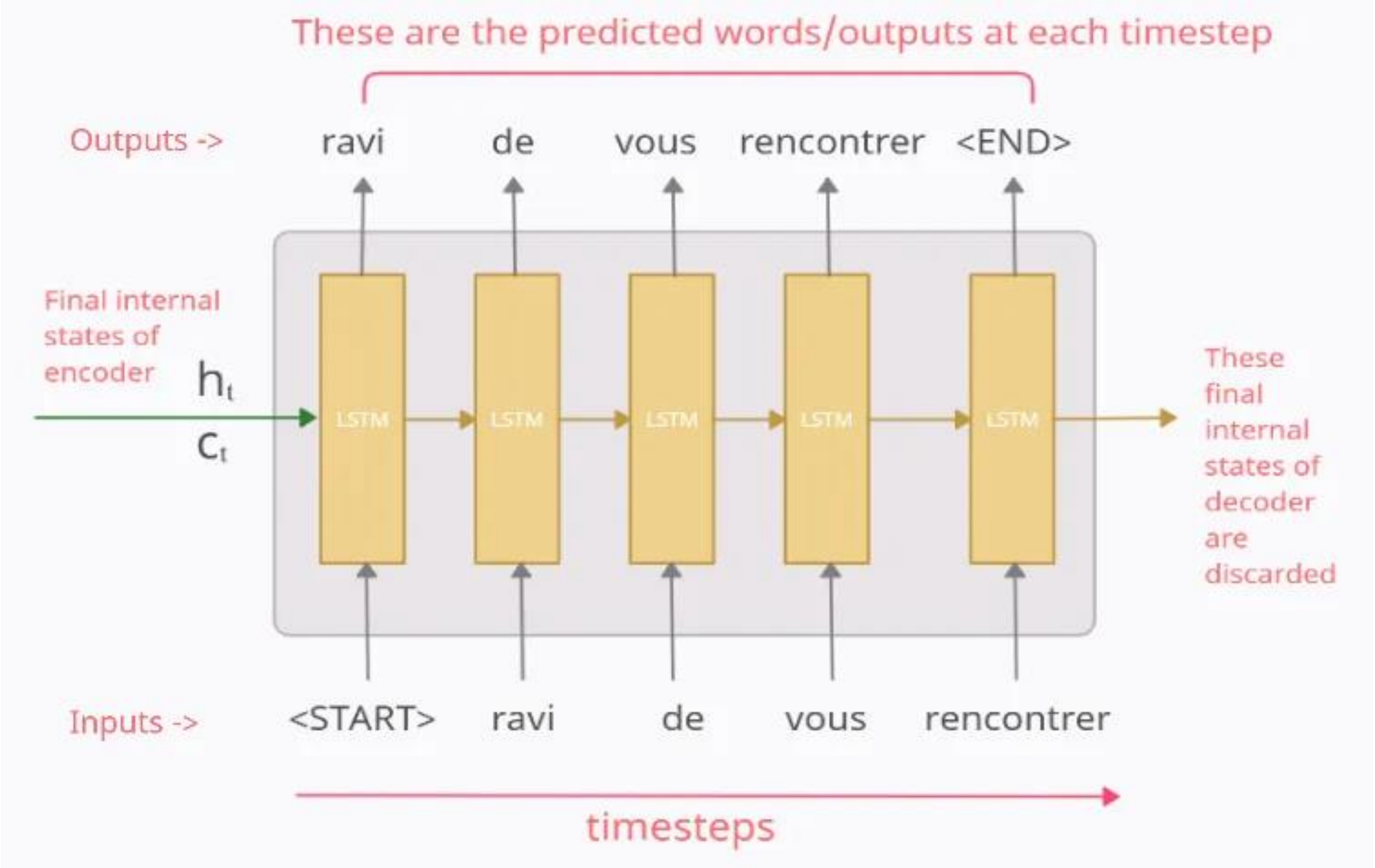
- Input: English sentence: “nice to meet you”
- Output: French translation: “ravi de vous rencontrer”



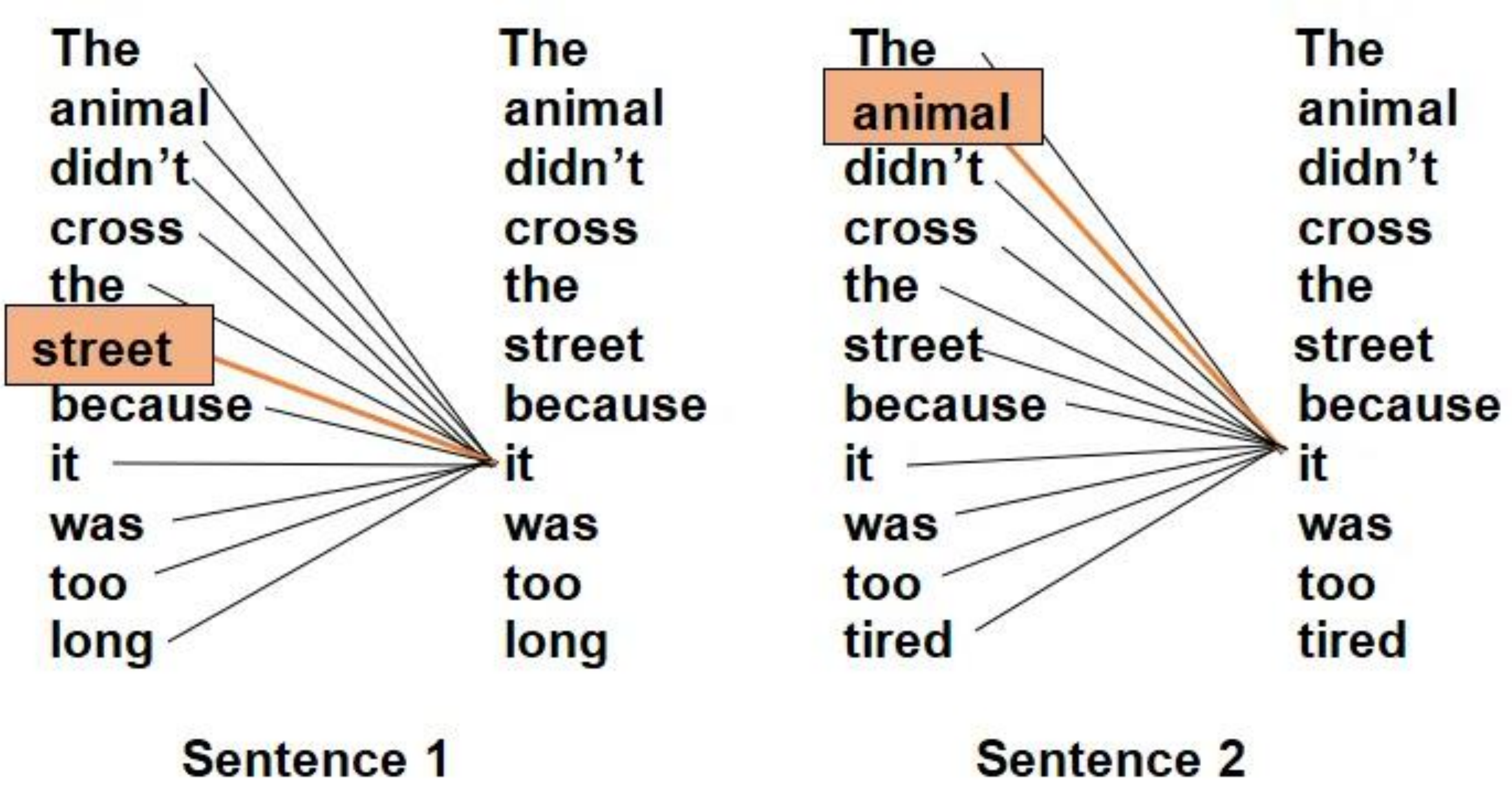
What’s under the hood?




These outputs are discarded
LSTM
LSTM
LSTM
LSTM
h_t
c_t
Final states of the encoder are passed onto the decoder
nice
to
meet
you
timesteps


These are the predicted words/outputs at each timestep
Outputs ->
ravi
de
vous
rencontrer
<END>
Final internal states of encoder
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
These final internal states of decoder are discarded
Inputs ->
<START>
ravi
de
vous
rencontrer
timesteps

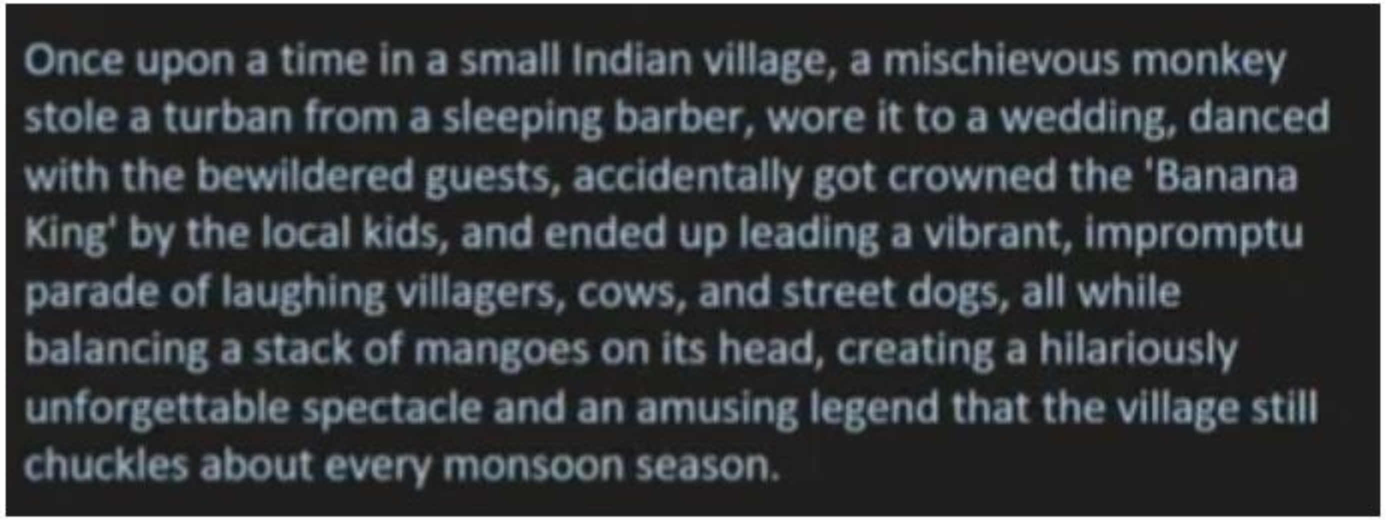
Attention Mechanism in Sequence to Sequence Model



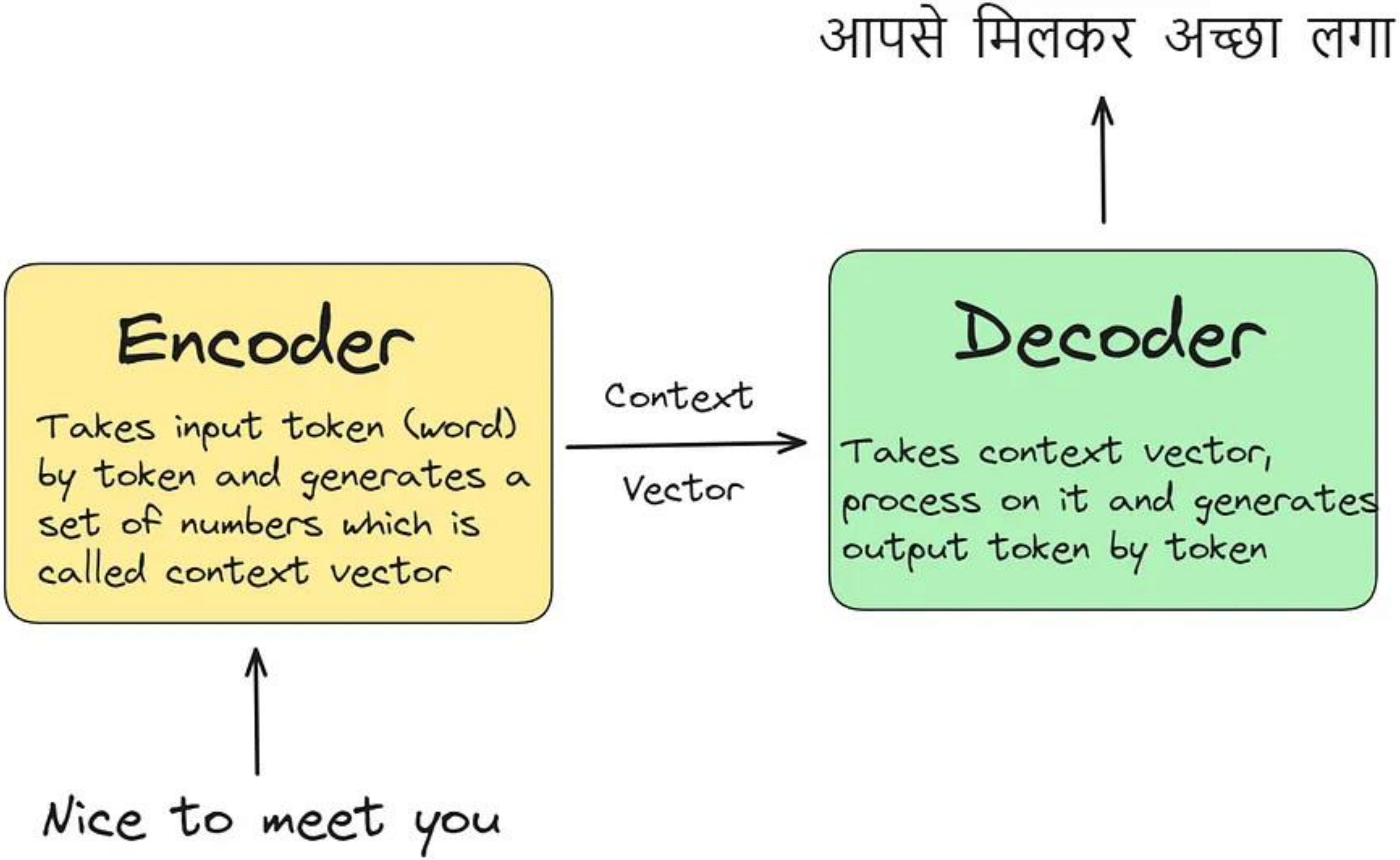
Inspired by the human cognitive process of selectively focusing on specific aspects of input data, attention mechanisms allow models to dynamically allocate computational resources to the most relevant components of a given input.

Problem in Encoder Decoder Architecture : Need of Attention

Problem at Encoder Side : In natural language processing, encoder-decoder architectures face a challenge similar to that encountered by humans when attempting to process lengthy sentences. Imagine trying to translate a long, complex sentence into another language after only glancing at it once. It’s nearly impossible to retain the entire context and accurately translate it, especially when dealing with sentences containing more than 25 words.



Once upon a time in a small Indian village, a mischievous monkey stole a turban from a sleeping barber, wore it to a wedding, danced with the bewildered guests, accidentally got crowned the 'Banana King' by the local kids, and ended up leading a vibrant, impromptu parade of laughing villagers, cows, and street dogs, all while balancing a stack of mangoes on its head, creating a hilariously unforgettable spectacle and an amusing legend that the village still chuckles about every monsoon season.



आपसे मिलकर अच्छा लगा

Encoder

Takes input token (word) by token and generates a set of numbers which is called context vector

Context Vector

Decoder

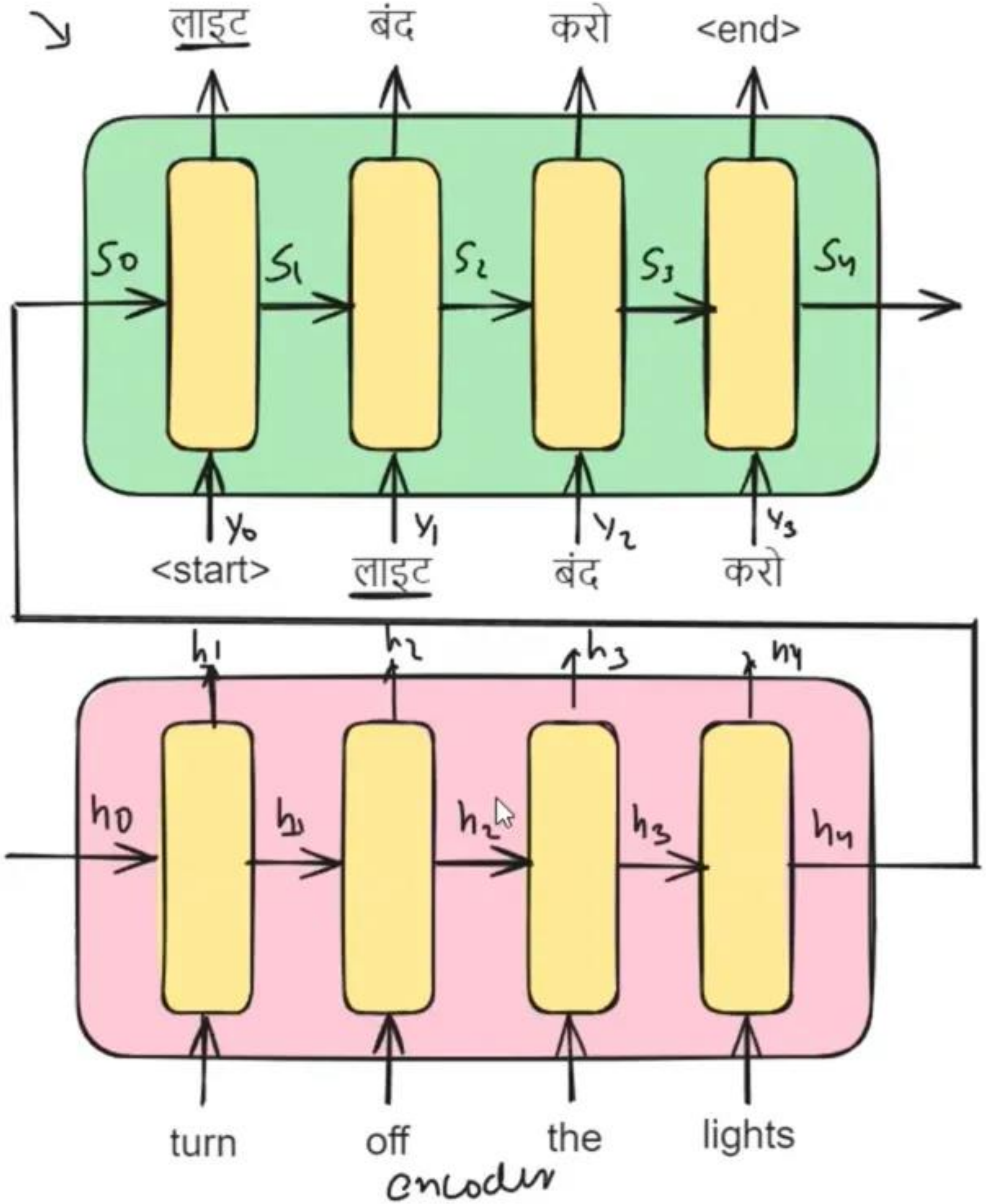
Takes context vector, process on it and generates output token by token

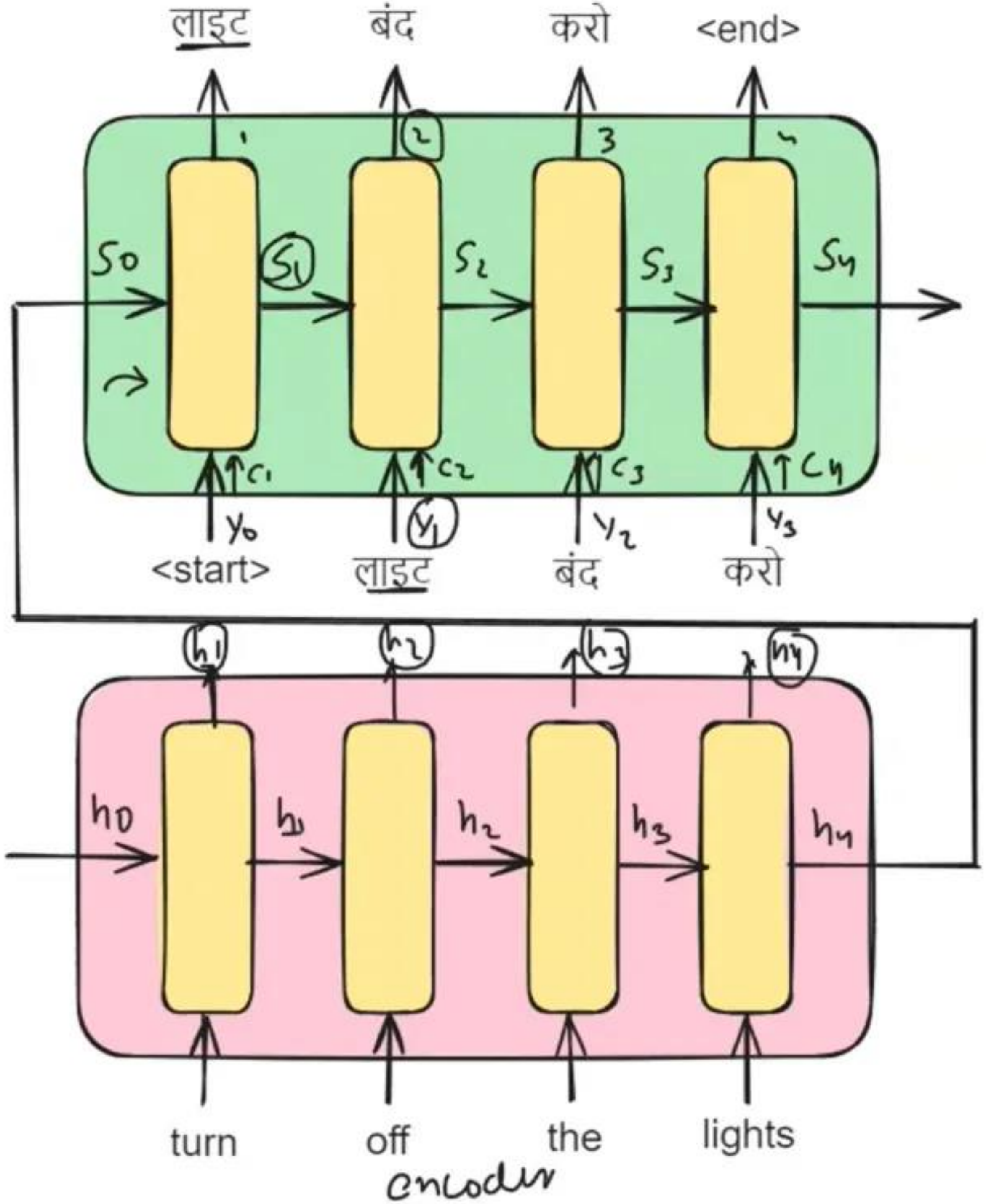
Nice to meet you

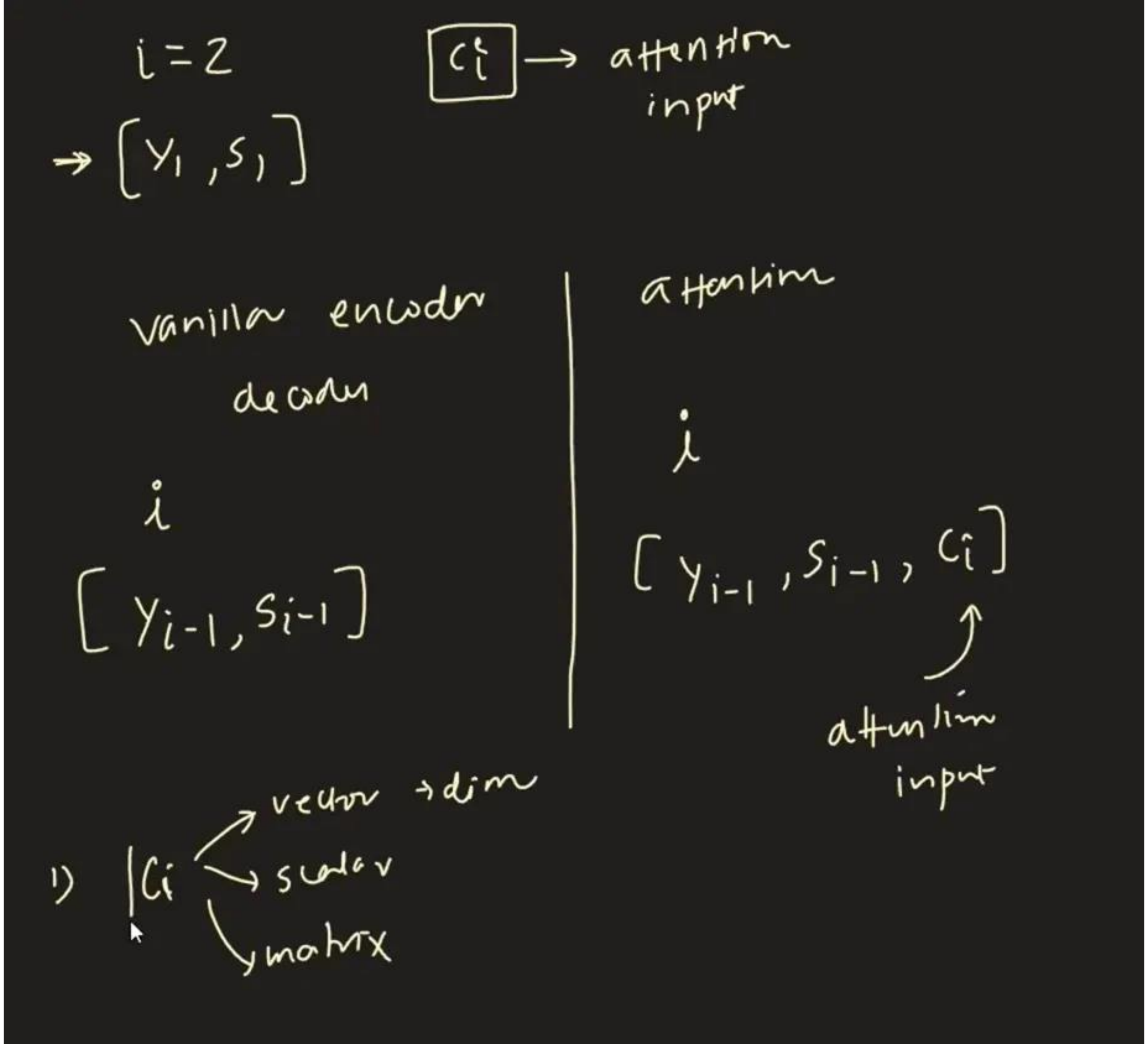
What is Attention Mechanism ?

The attention mechanism enables models to selectively focus on relevant parts of the input sequence while generating each element of the output sequence, improving accuracy and capturing long-range dependencies in tasks like machine translation. Let’s take an example to understand it :

Below is the encoder decoder diagram is shown where we translate English to Hindi. we pass the input ‘turn off the light.” and we get the output from decoder side is ‘लाइट बंद करो’.

लाइट
बंद
करो
<end>
s_0
s_1
s_2
s_3
s_4
y_0
y_1
y_2
y_3
<start>
लाइट
बंद
करो
h_1
h_2
h_3
h_4
h_0
h_1
h_2
h_3
h_4
turn
off
the
lights
encoder


लाइट
बंद
करो
<end>
1
2
3
4
S0
S1
S2
S3
S4
c1
c2
c3
c4
y0
y1
y2
y3
<start>
लाइट
बंद
करो
h1
h2
h3
h4
h0
h1
h2
h3
h4
turn
off
the
lights
encoder



$i=2$

$\boxed{c_i} \rightarrow$ attention input

$\rightarrow [y_1, s_1]$

vanilla encoder decoder

i

$[y_{i-1}, s_{i-1}]$

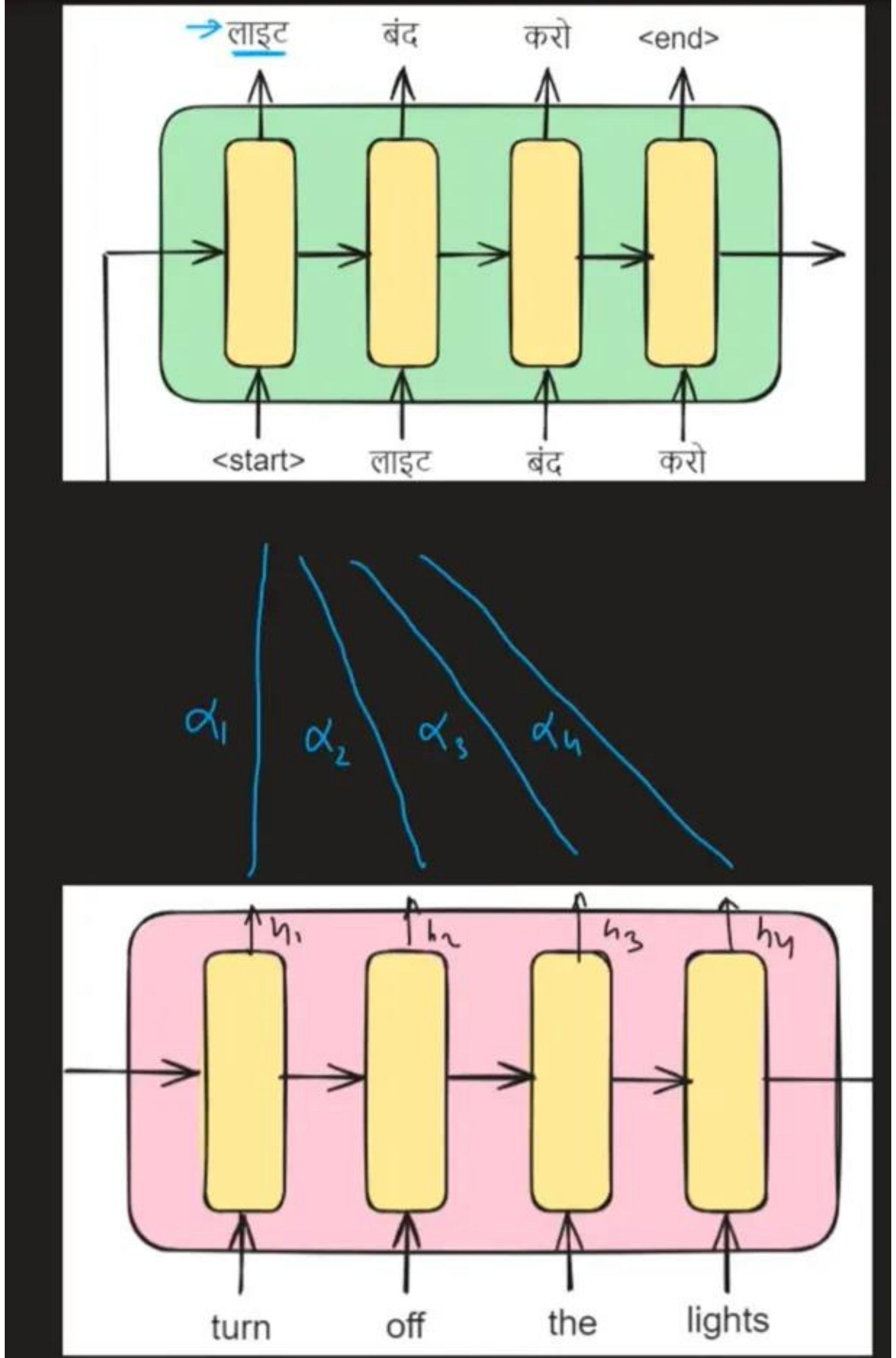
attention

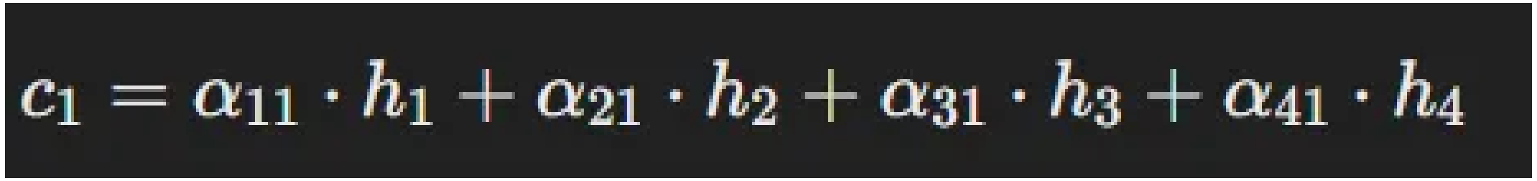
i

$[y_{i-1}, s_{i-1}, c_i]$

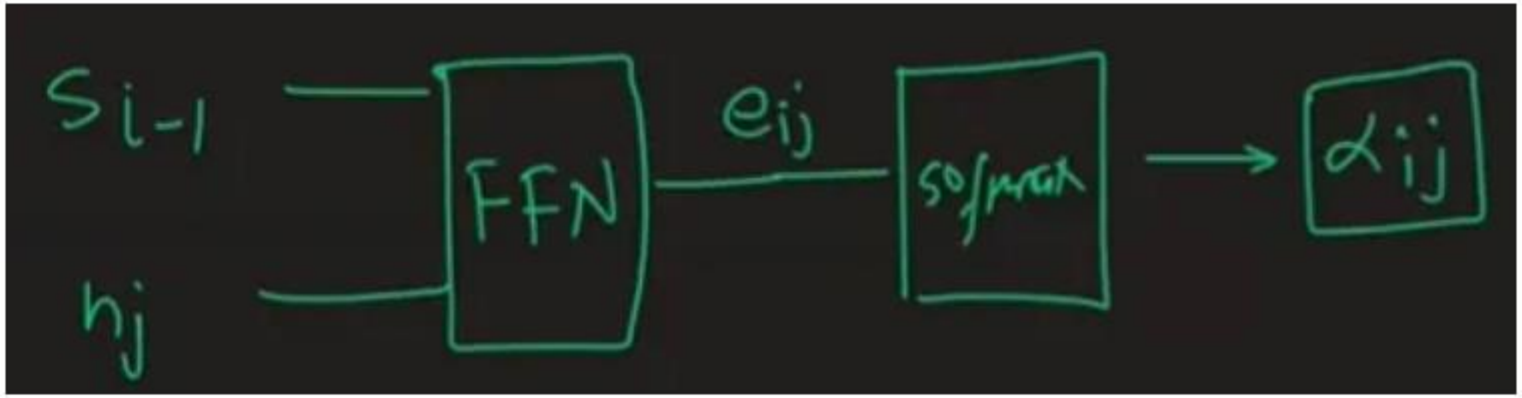
$\uparrow$ attention input

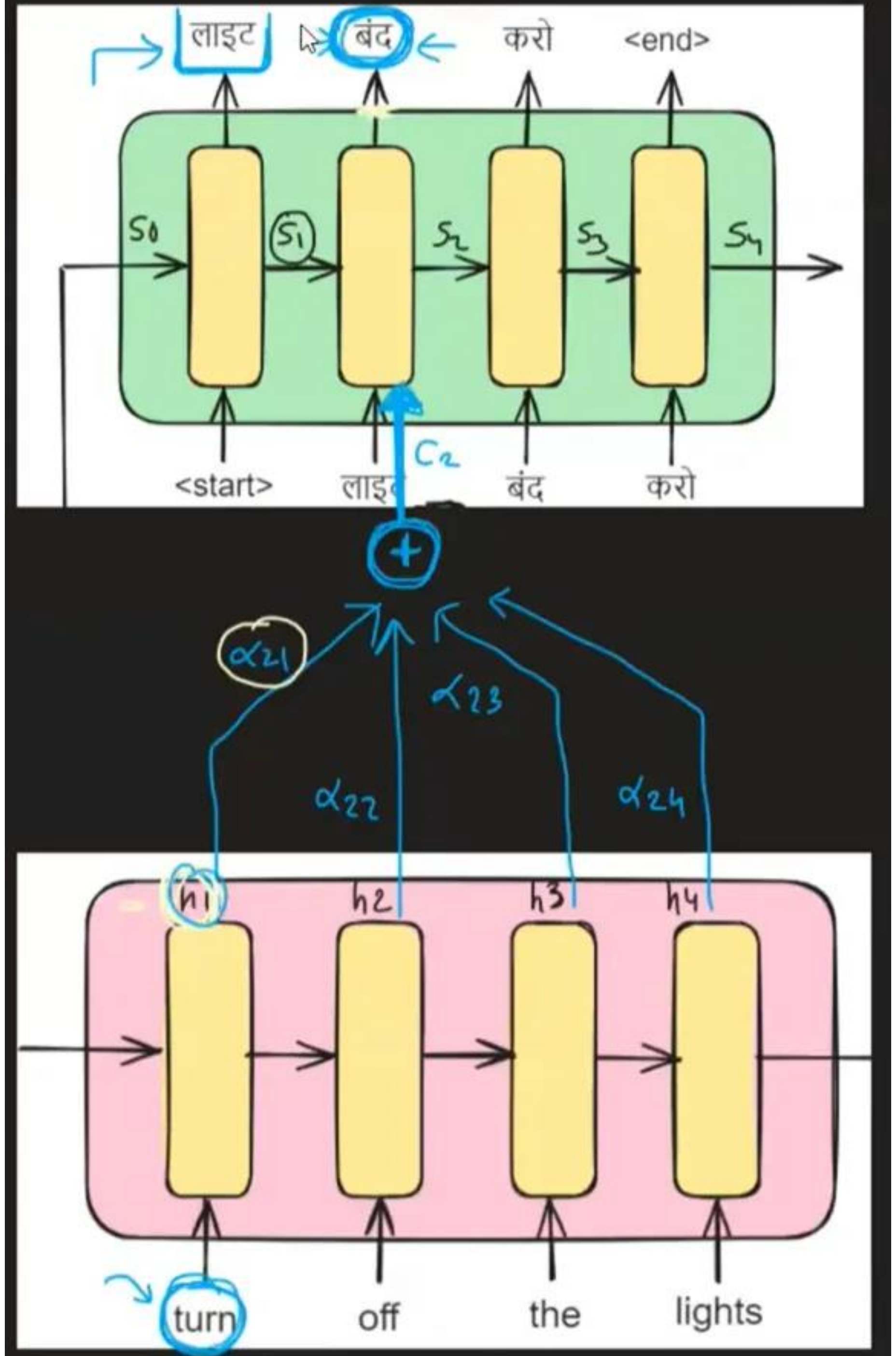
1) $|c_i$ → vector → dim
→ scalar
→ matrix

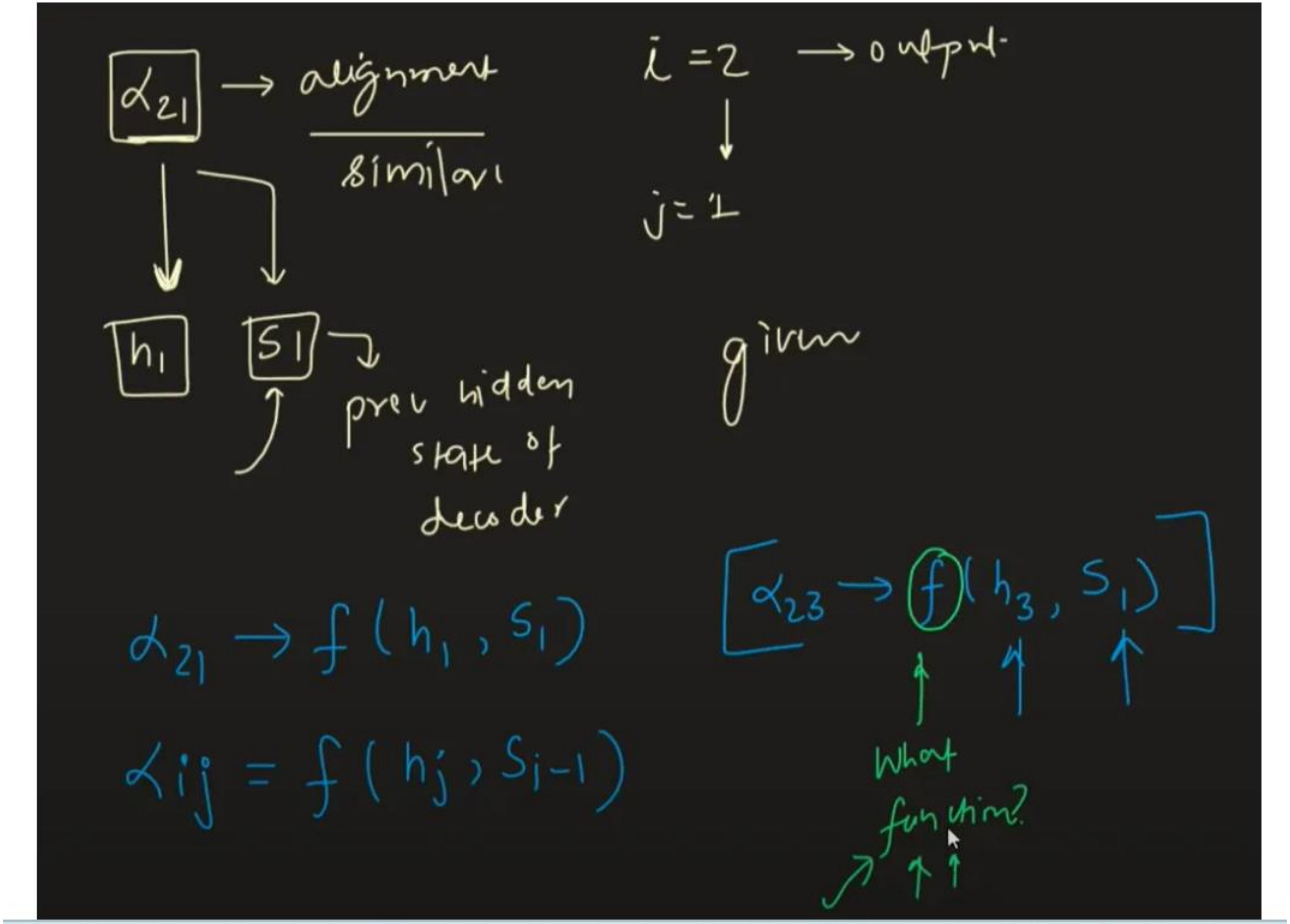

लाइट
बंद
करो
<end>
<start>
लाइट
बंद
करो
α_1
α_2
α_3
α_4
h_1
h_2
h_3
h_4
turn
off
the
lights



$$c_1 = \alpha_{11} \cdot h_1 + \alpha_{21} \cdot h_2 + \alpha_{31} \cdot h_3 + \alpha_{41} \cdot h_4$$




लाइट
बंद
करो
<end>
s_0
s_1
s_2
s_3
s_4
<start>
लाइट
C_2
बंद
करो
+
α_{21}
α_{23}
α_{22}
α_{24}
h1
h2
h3
h4
turn
off
the
lights



$\boxed{\alpha_{21}} \rightarrow \dfrac{\text{alignment}}{\text{similari}}$

$\boxed{h_1}$ $\boxed{S1}$ → prev hidden state of decoder

$i = 2 \rightarrow$ output

$\downarrow$

$j = 1$

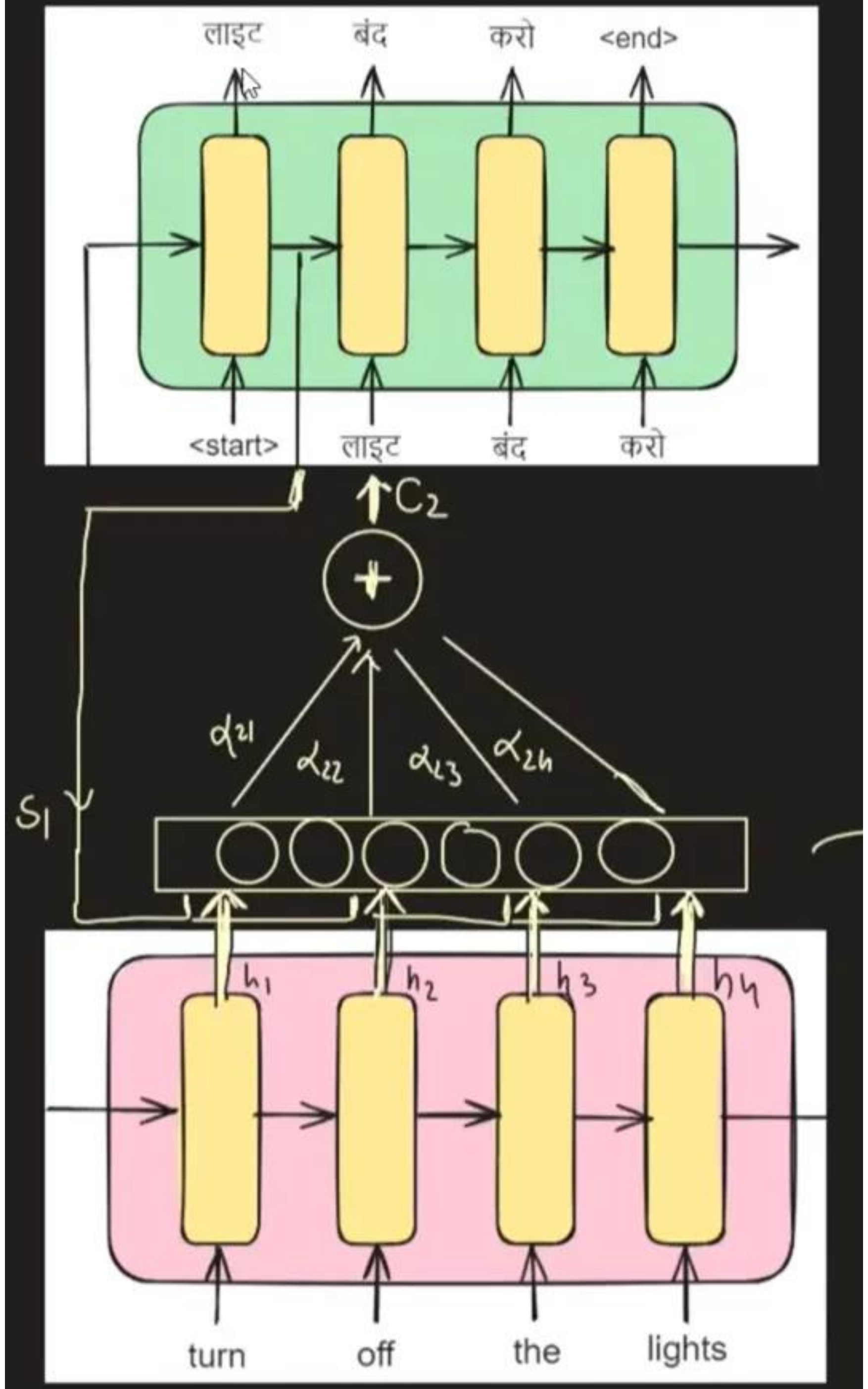
given

$\alpha_{21} \rightarrow f(h_1, s_1)$

$\alpha_{ij} = f(h_j, s_{i-1})$

$\left[\alpha_{23} \rightarrow \boxed{f}(h_3, s_1)\right]$

What function?


लाइट
बंद
करो
<end>
<start>
लाइट
बंद
करो
C2
+
α21
α22
α23
α24
S1
h1
h2
h3
h4
turn
off
the
lights