دانشگاه کردستان
University of Kurdistan
زانکۆی کوردستان

Department of Computer Engineering
University of Kurdistan

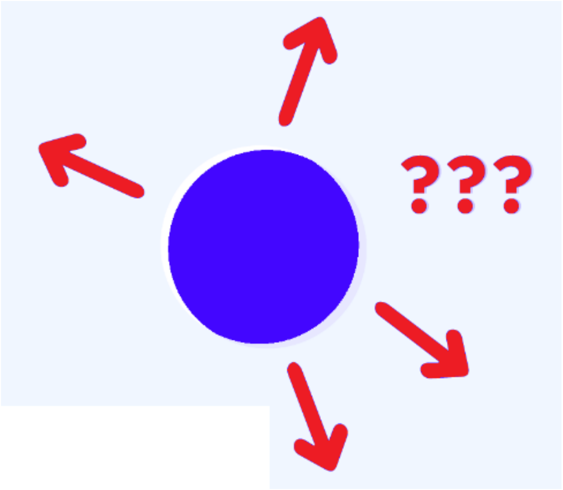
Deep Learning (Graduate level)

Recurrent Neural Networks (RNN)

By: Dr. Alireza Abdollahpouri

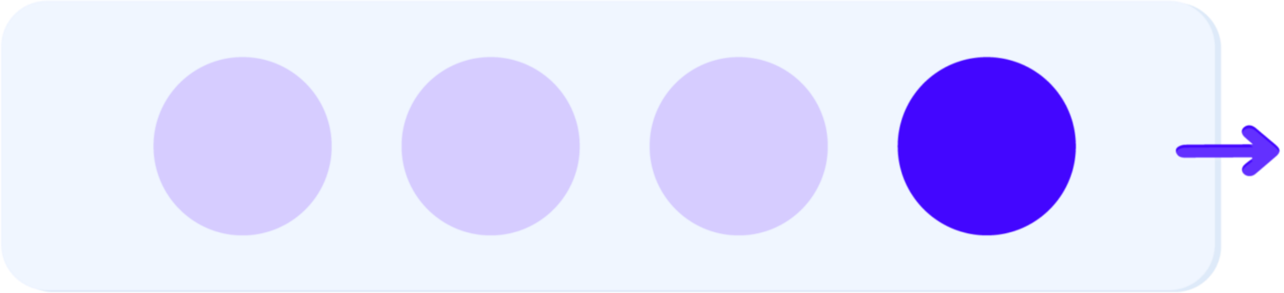
Recurrent Neural Networks (RNN)

Given an image of a ball, can you predict where it will go next?



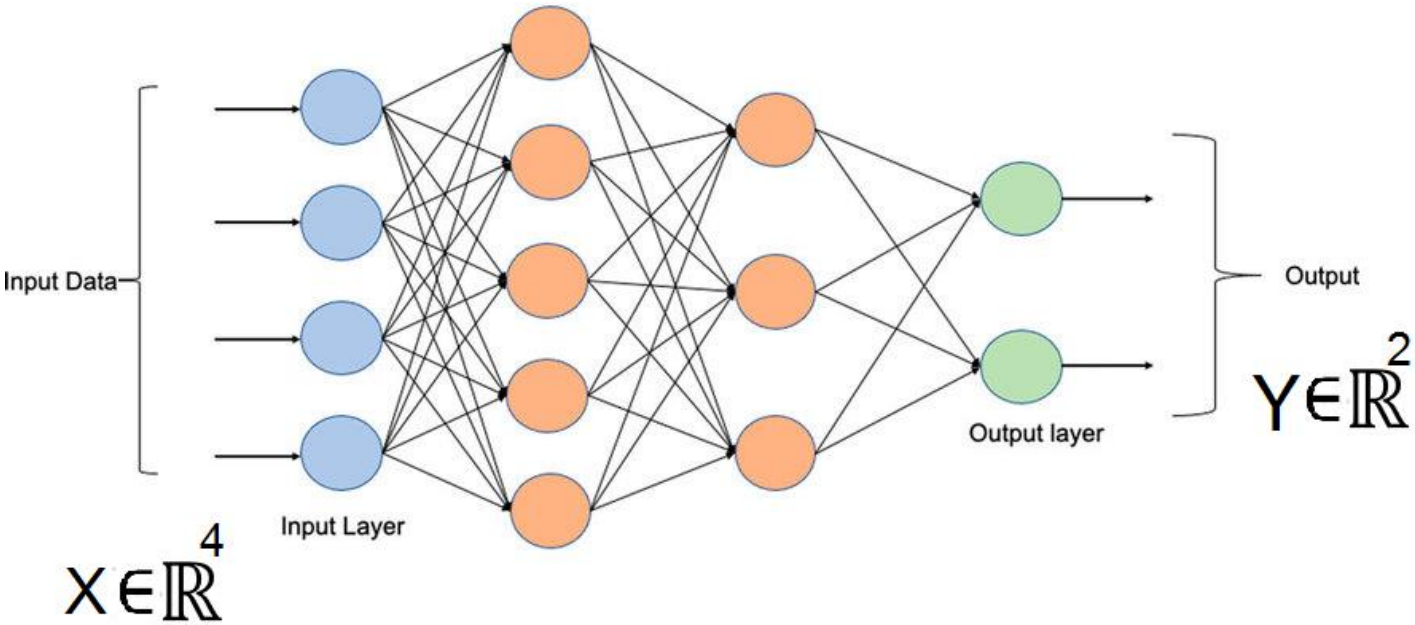
Recurrent Neural Networks (RNN)

Given an image of a ball, can you predict where it will go next?



Limitations of Feed forward Networks

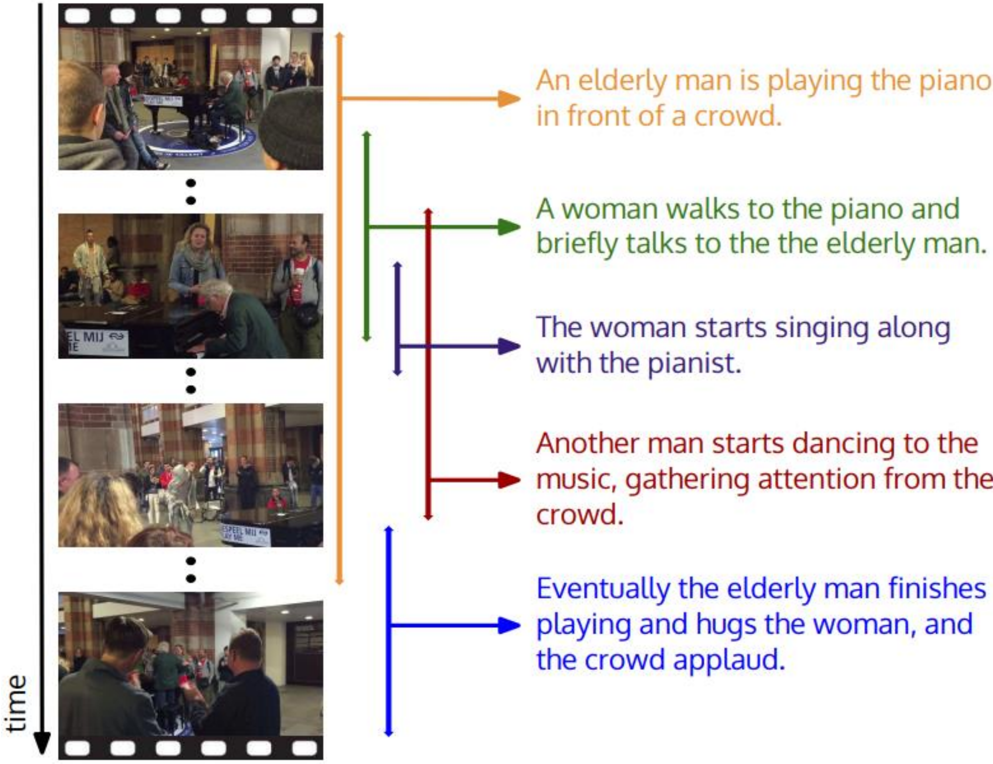
- Information flows only in the forward direction. **No cycles** or Loops
- Decisions are based on current input, **no memory** about the past
- Doesn’t know how to handle **sequential data**
- Inability to handle **variable-length** input



Why are existing convnets insufficient?

Variable sequence length inputs and outputs!

Example task: **video captioning**
Input video can have variable number of frames
Output captions can be variable length.



An elderly man is playing the piano in front of a crowd.

A woman walks to the piano and briefly talks to the the elderly man.

The woman starts singing along with the pianist.

Another man starts dancing to the music, gathering attention from the crowd.

Eventually the elderly man finishes playing and hugs the woman, and the crowd applaud.

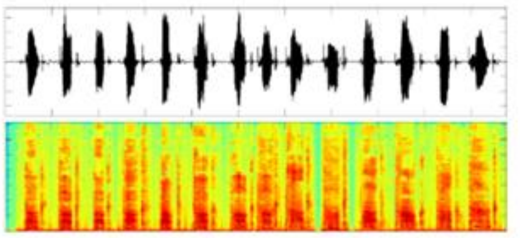
Recurrent Neural Networks (RNN)

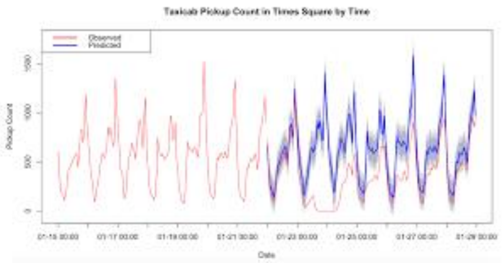
- Models **temporal** information
- Hidden states as a function of inputs and previous time step information

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta)$$

- Temporal information is important in many applications

1. Natural Language Processing (Machine Translation, Text Generation)
2. Time Series Forecasting
3. Speech Recognition
4. Video Analysis

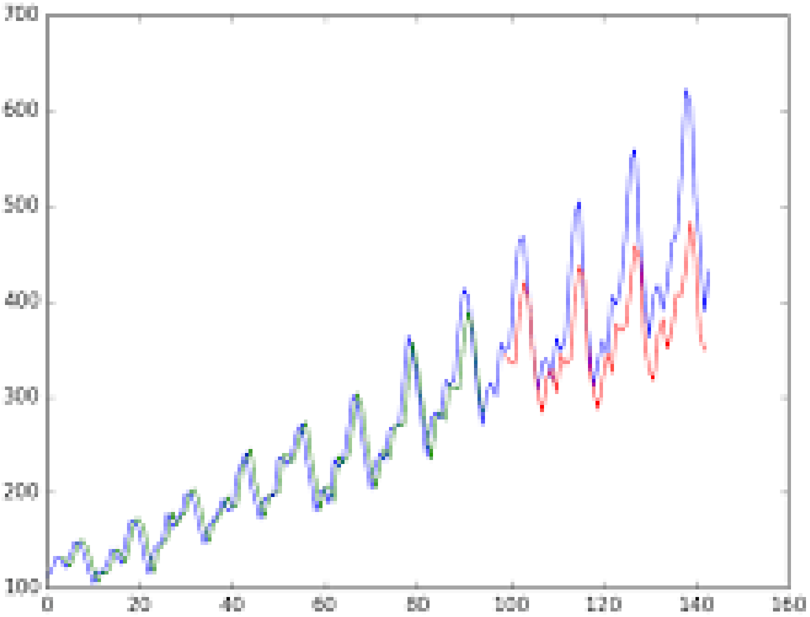




Sequential Data

Sometimes the sequence of data matters:

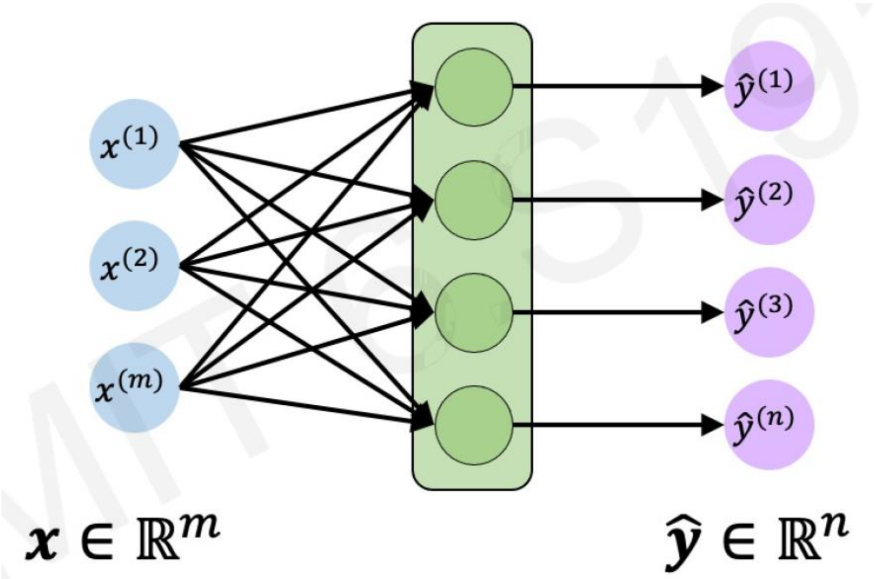
- **Sentences:** "The dog bit the cat" ≠ "The cat bit the dog“
- **DNA sequences:** ATG = start codon ≠ GTA (codes for different amino acid)

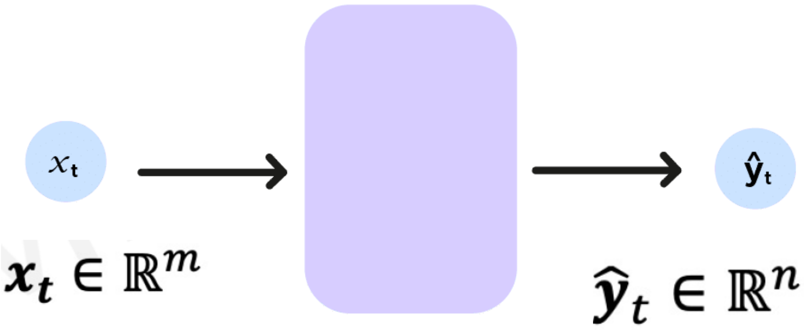


The clouds are in the .... (Answer: sky)

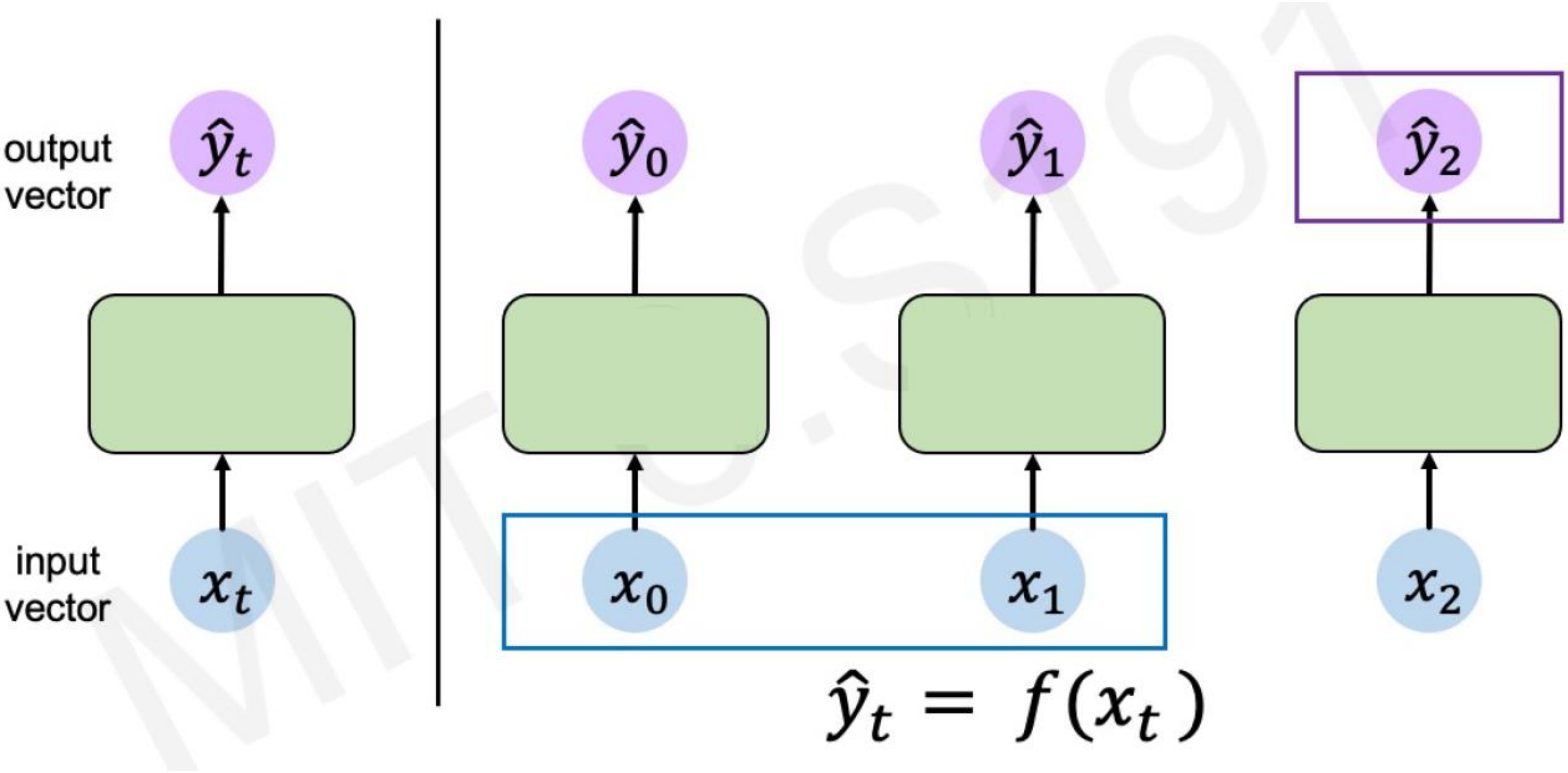
University of Kurdistan

Feed-Forward Network



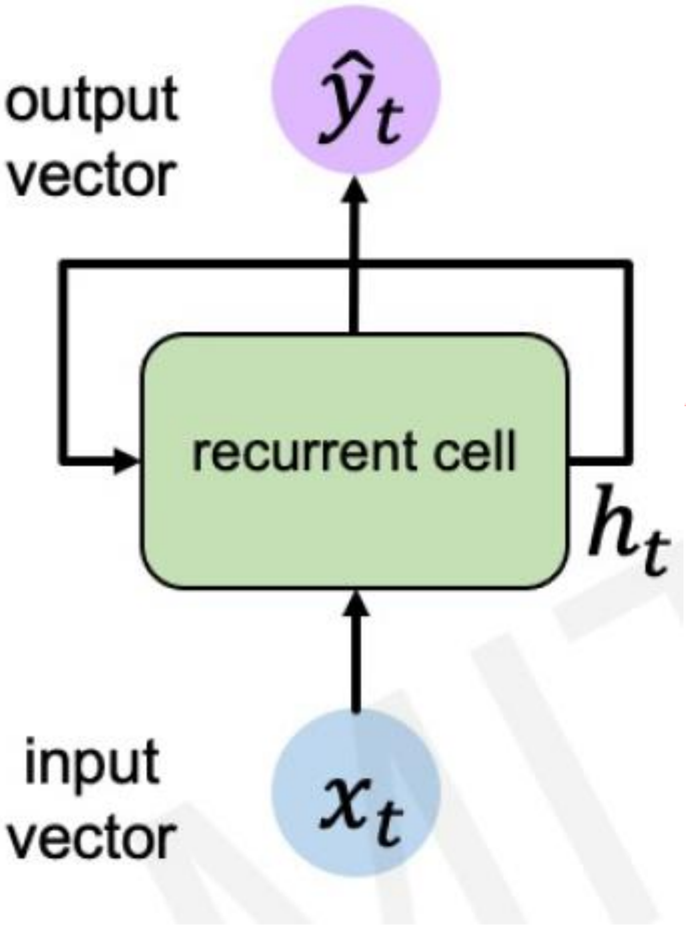

x_t
$\hat{y}_t$
$\boldsymbol{x}_t \in \mathbb{R}^m$
$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

Handling Individual Time steps



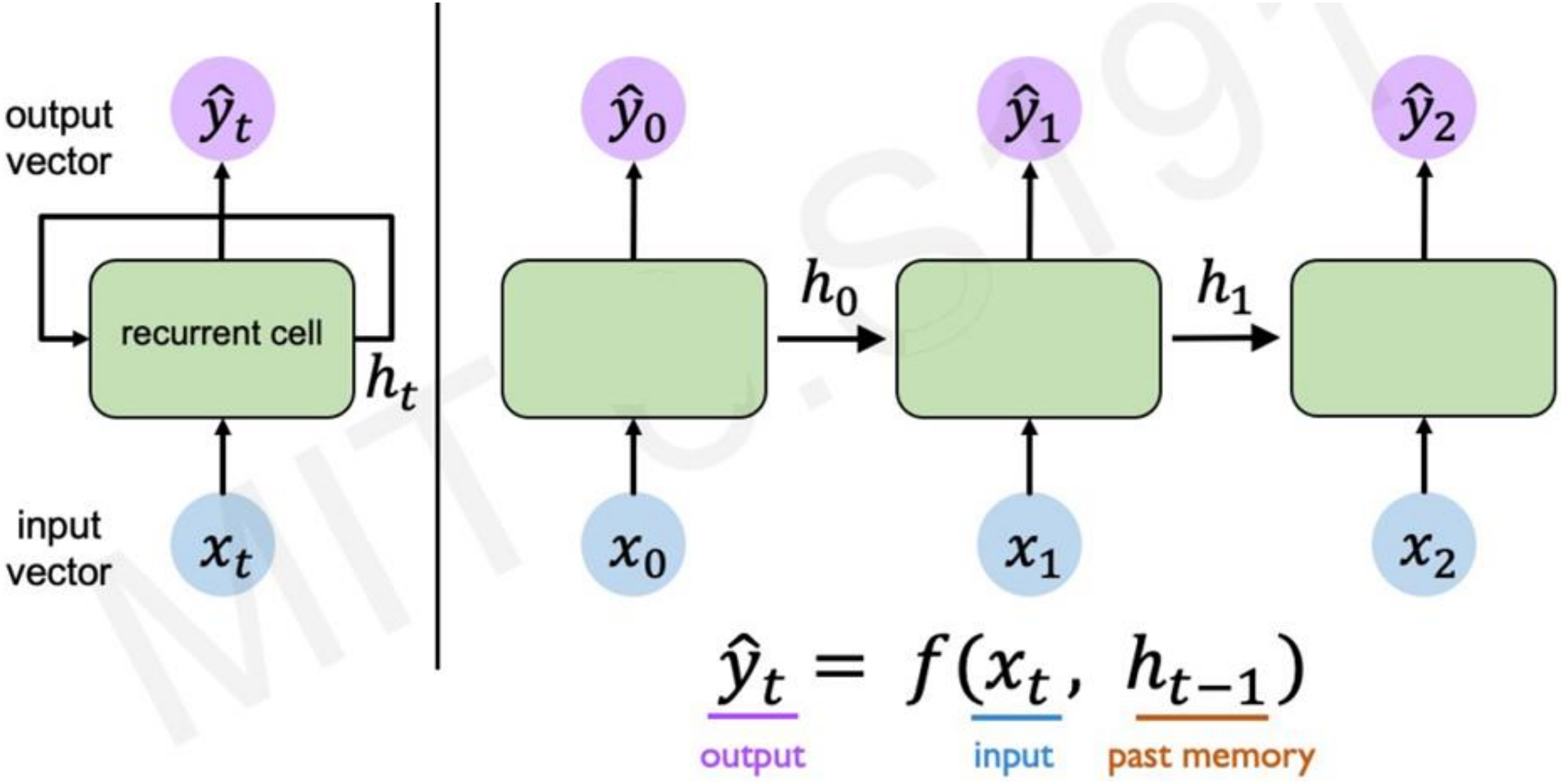
$$\hat{y}_t = f(x_t)$$

RNN: Internal State



Key idea: RNNs have an "internal state" that is updated as a sequence is processed. You can think of it as "memory".

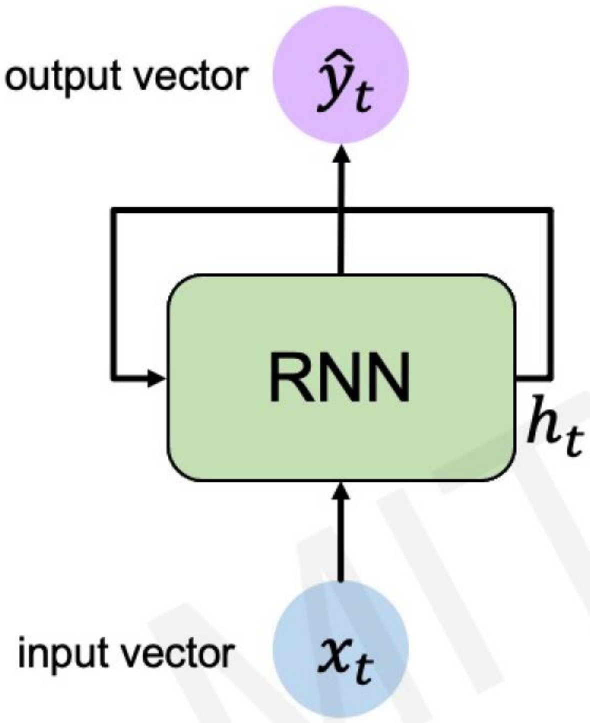
RNN: Unrolling through time



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}},\ \underbrace{h_{t-1}}_{\text{past memory}})$$

Unrolled RNN

Recurrent Neural Network



Apply a **recurrence relation** at every time step to process a sequence:

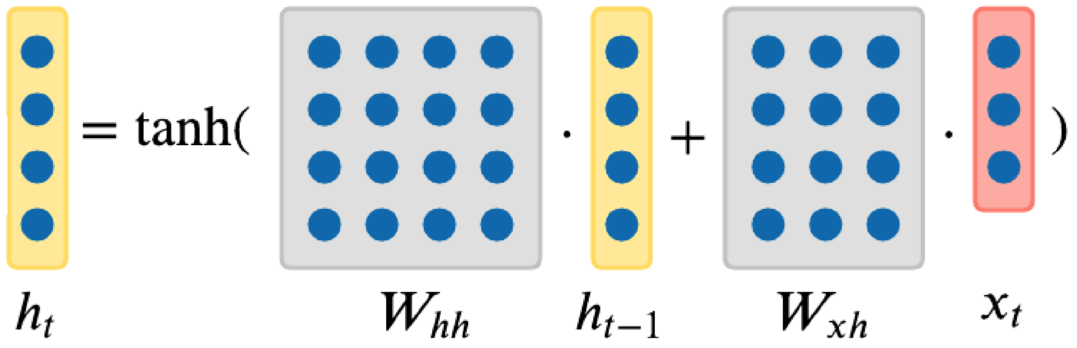
$$\boxed{h_t} = \boxed{f_W}(\boxed{x_t}, \boxed{h_{t-1}})$$

cell state — function with weights W — input — old state

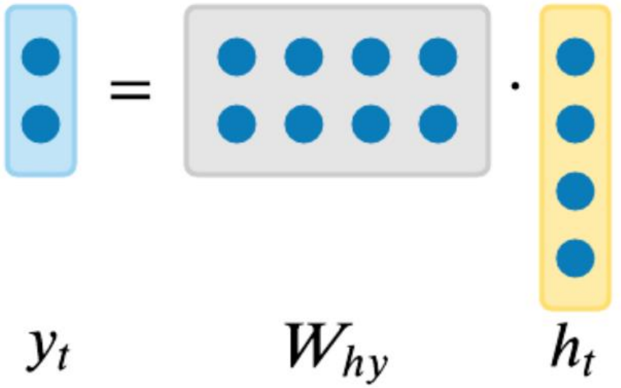
Note: the same function and set of parameters are used at every time step

RNN: State ad output computations

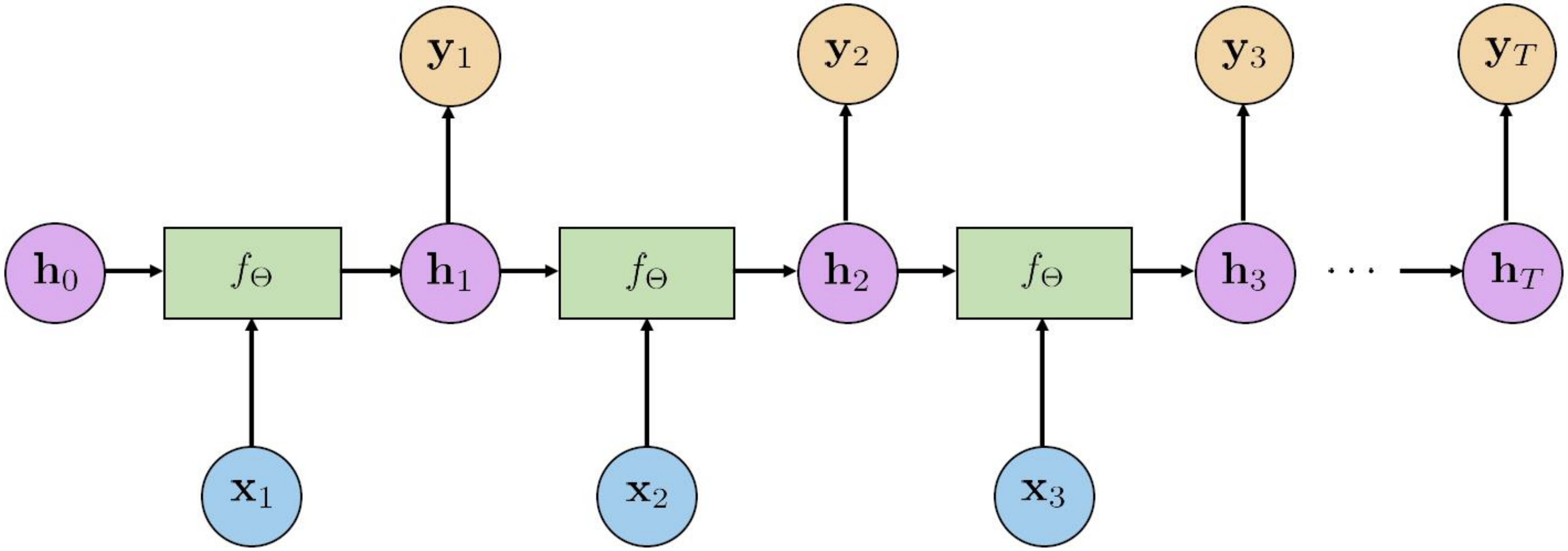
$$h_t = tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$



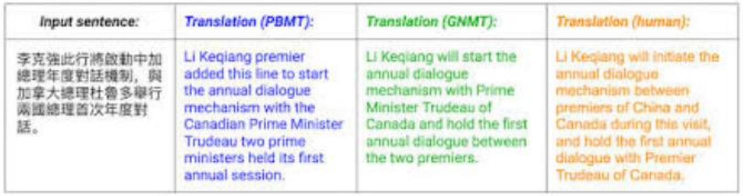
$$y_t = W_{hy} h_t$$



RNN: Computation Graph (Many to Many)

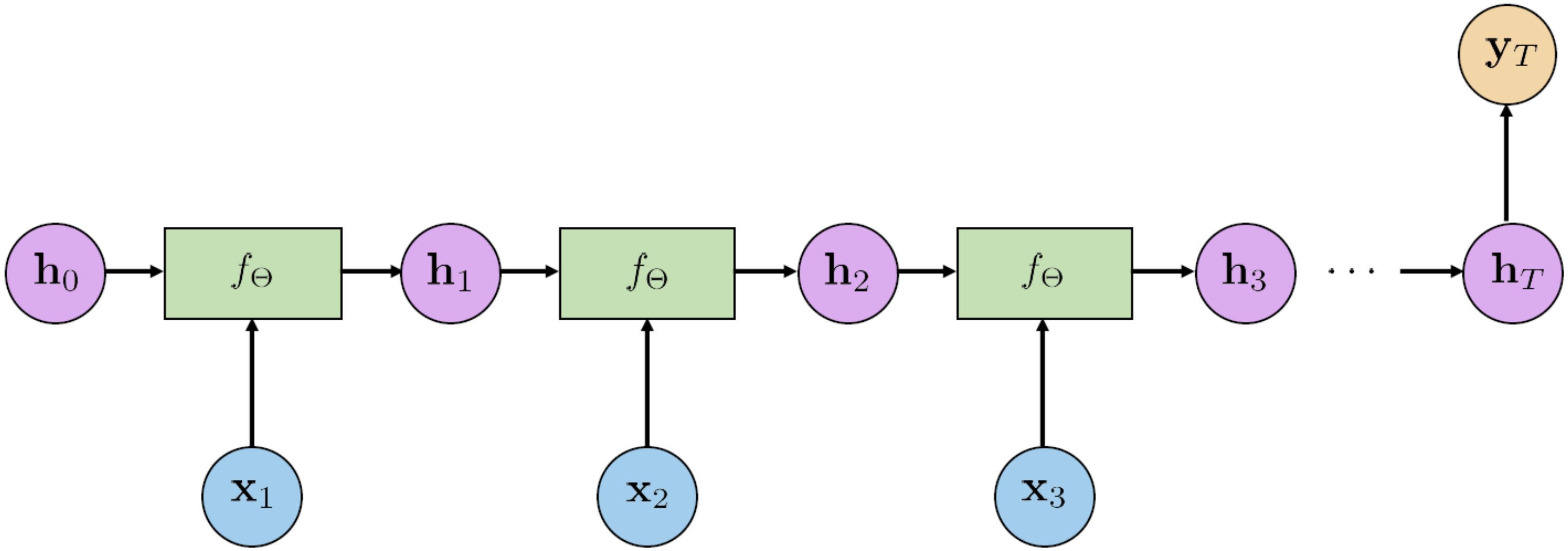


e.g., **Machine Translation**
(Sequence of words → Sequence of words)



Input sentence:	*Translation (PBMT):*	*Translation (GNMT):*	*Translation (human):*
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

RNN: Computation Graph (Many to one)

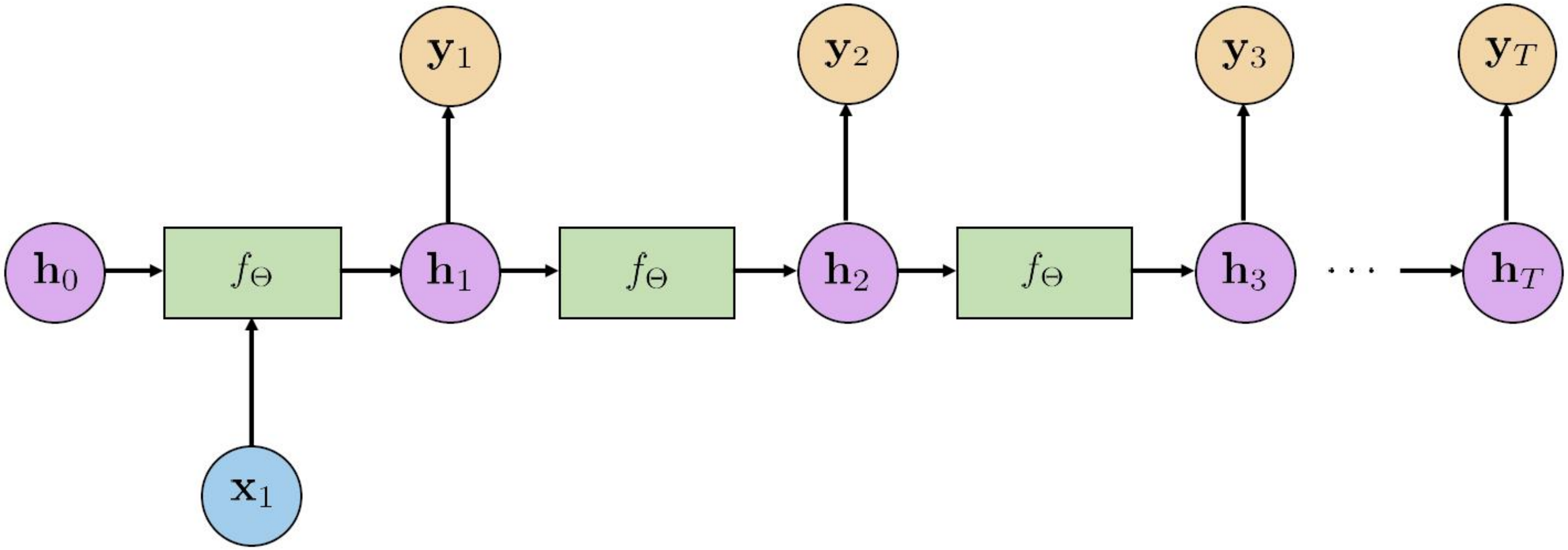


e.g., **Sentiment Classification**
(Sequence of words → sentiment)

This Class is very interesting!

😀

RNN: Computation Graph (One to Many)



e.g., **Image Captioning**
(Image → sequence of words)

No errors

A white teddy bear sitting in the grass

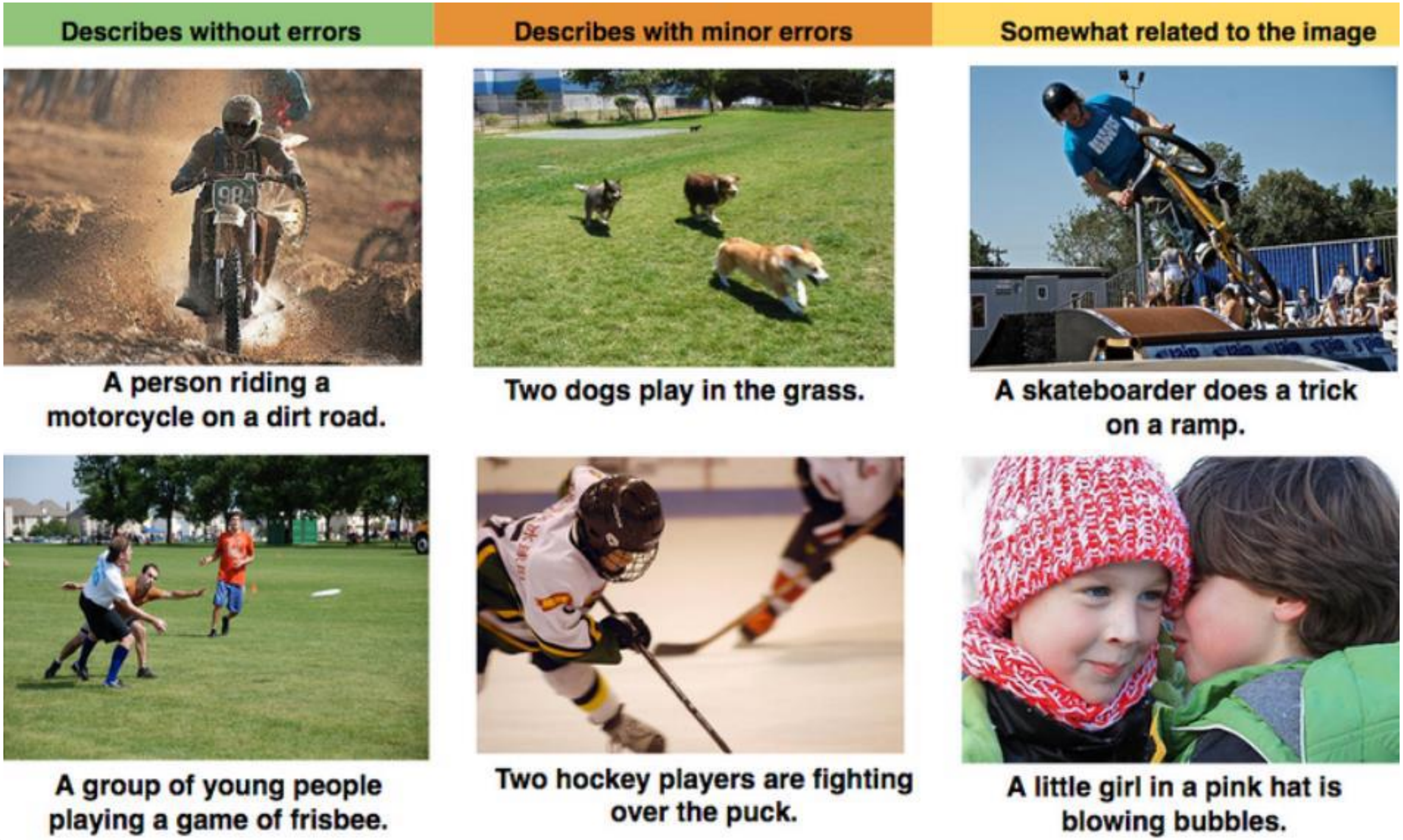
Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

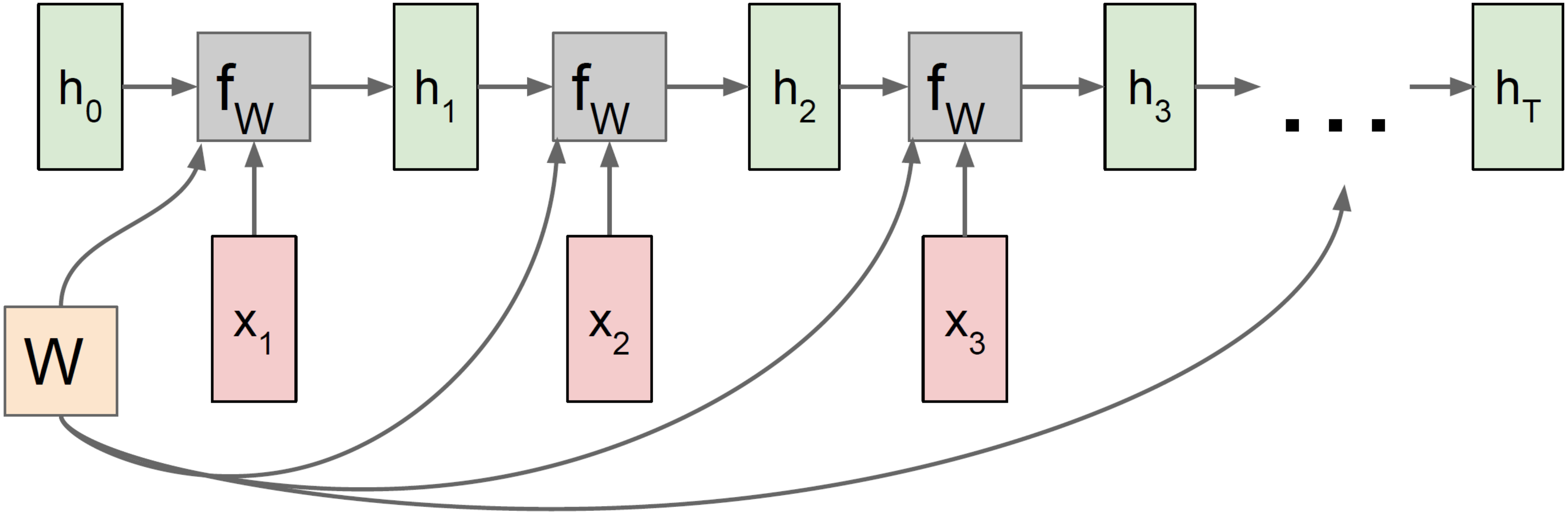
A woman is holding a cat in her hand

RNNs- Image Captioning Examples



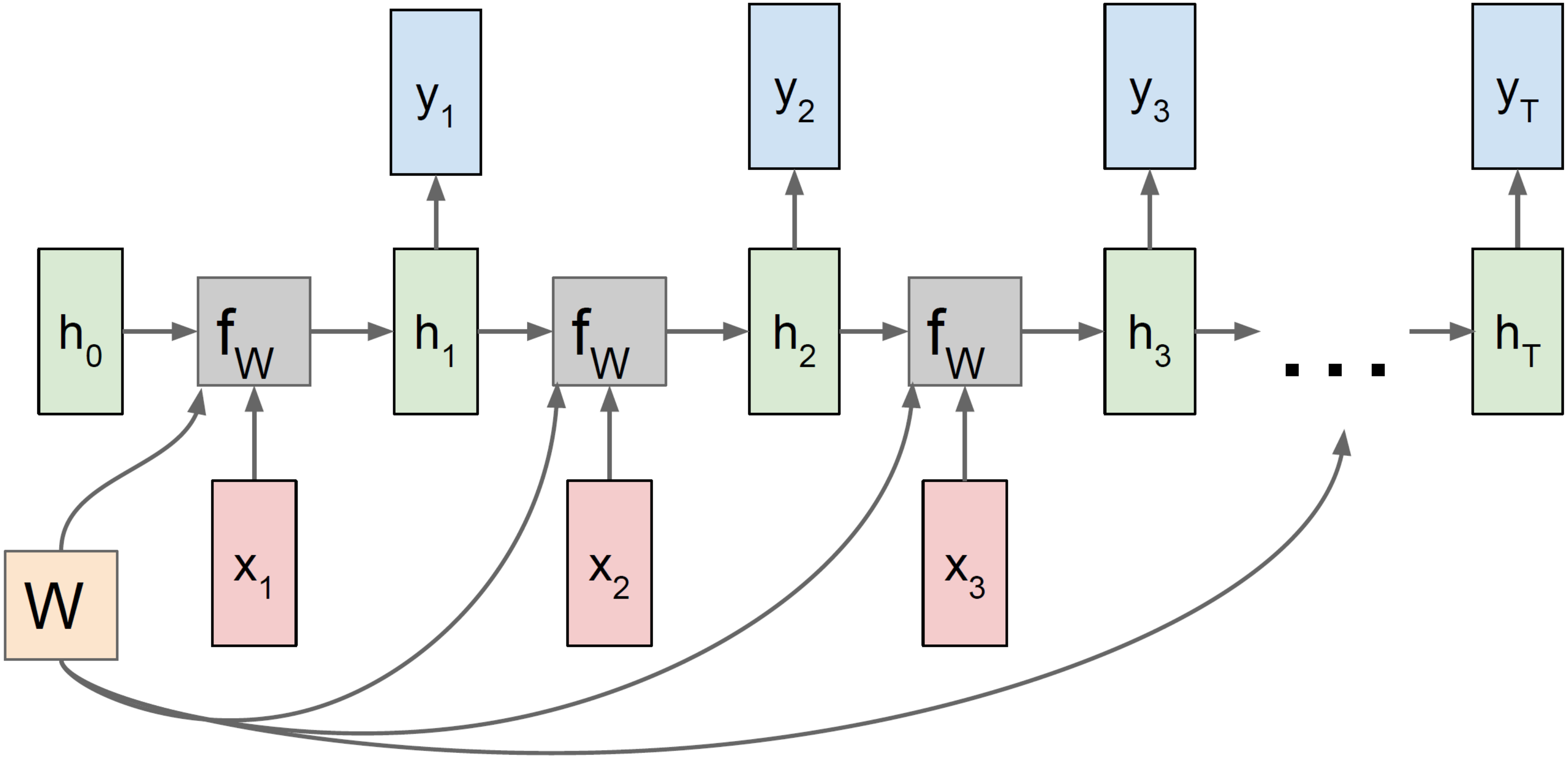
Describes without errors	Describes with minor errors	Somewhat related to the image
A person riding a motorcycle on a dirt road.	Two dogs play in the grass.	A skateboarder does a trick on a ramp.
A group of young people playing a game of frisbee.	Two hockey players are fighting over the puck.	A little girl in a pink hat is blowing bubbles.

RNN: Computational graph

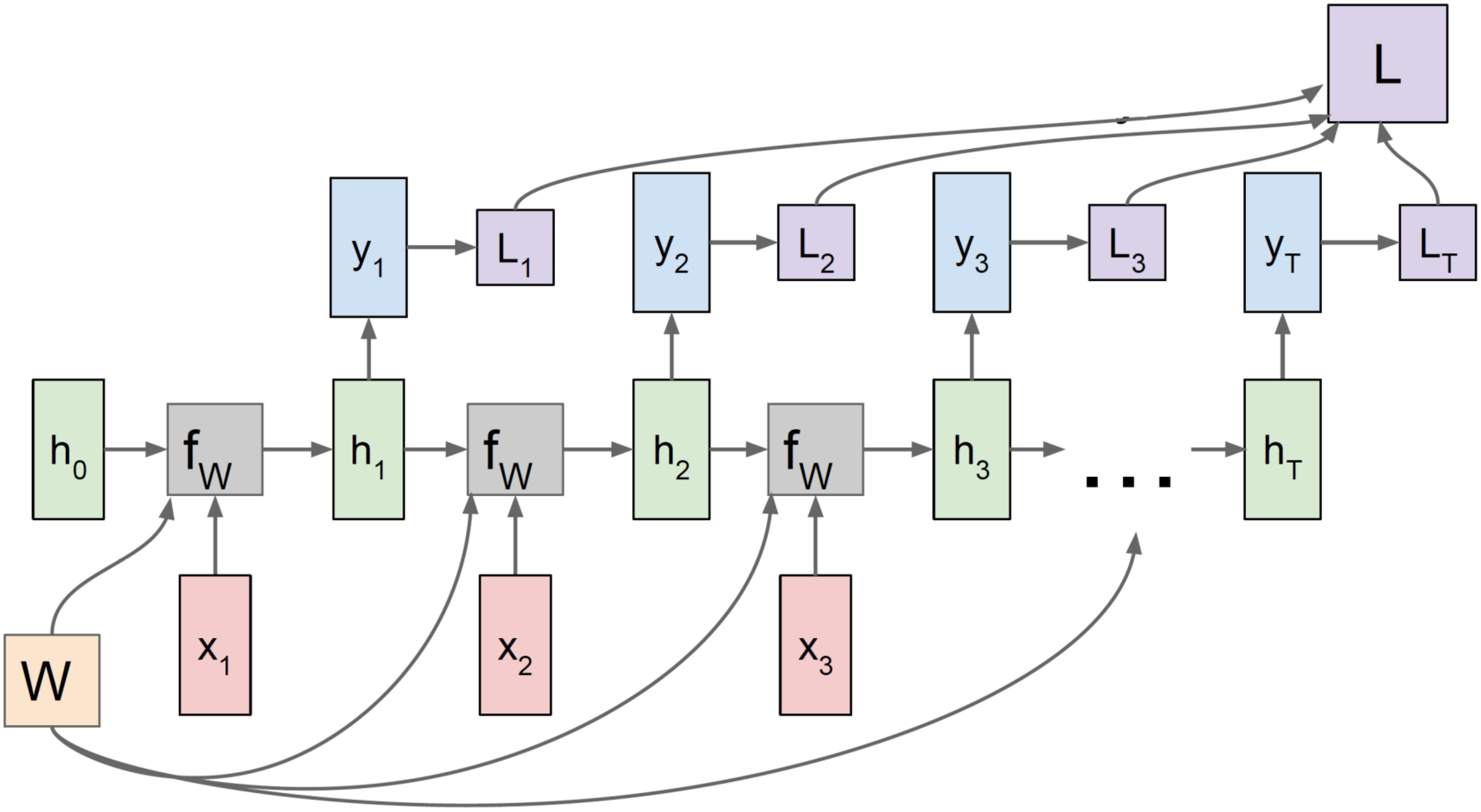


Re-use the same weight matrix at every time-step.

RNN: Computational graph: Many-to-many

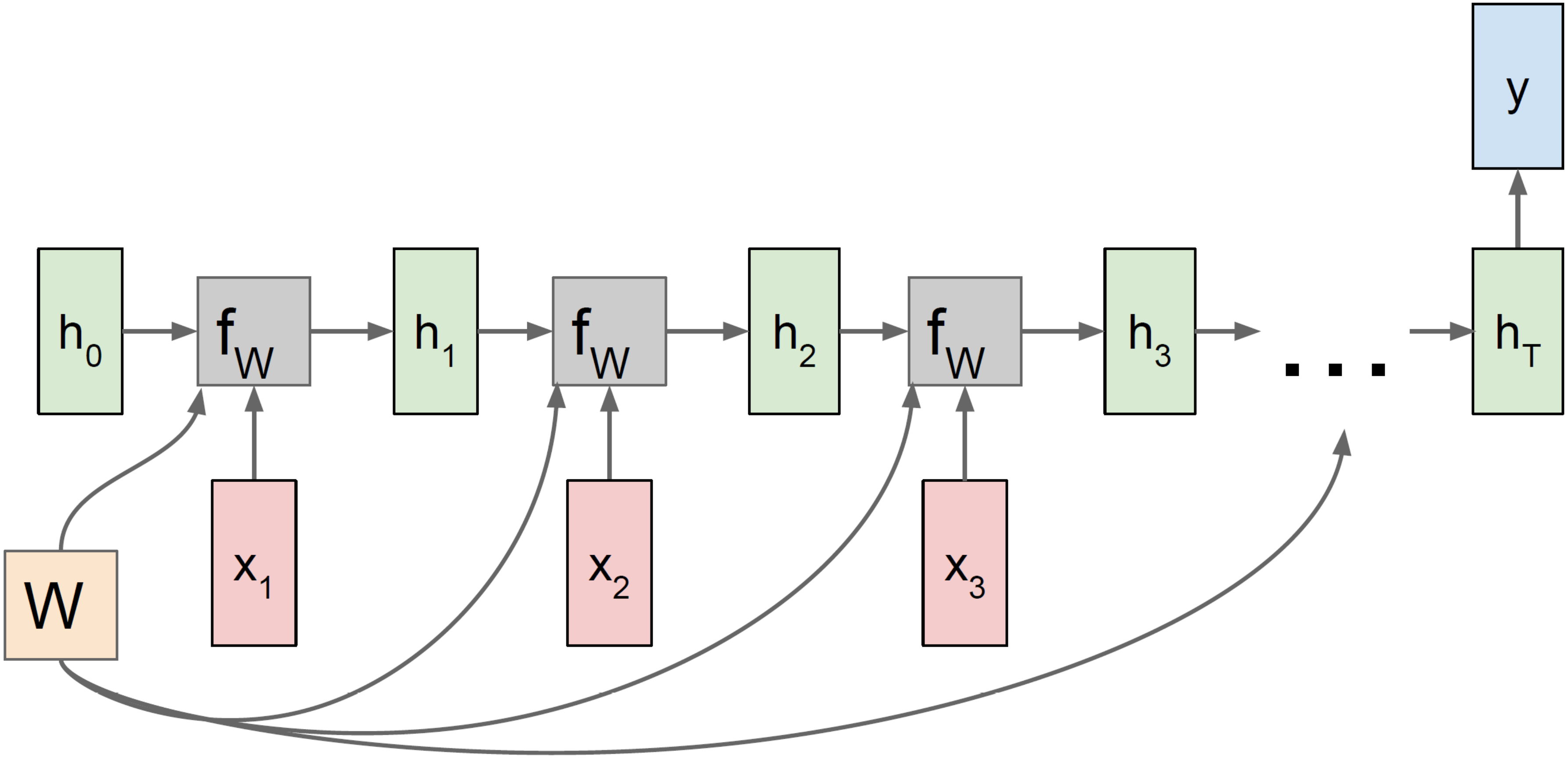


RNN: Computational graph: Many-to-many

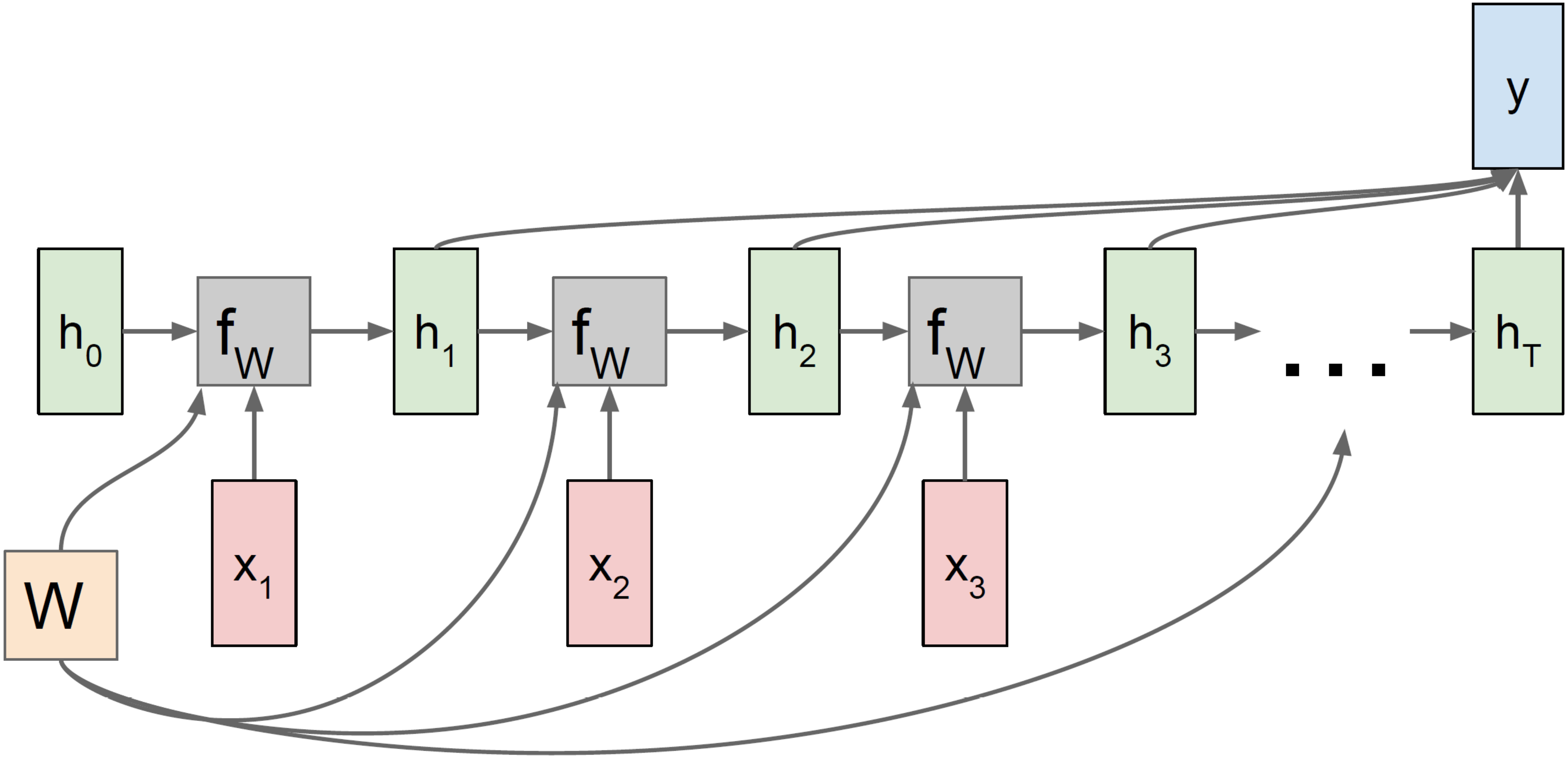


The loss is the sum of the loss at each slot.

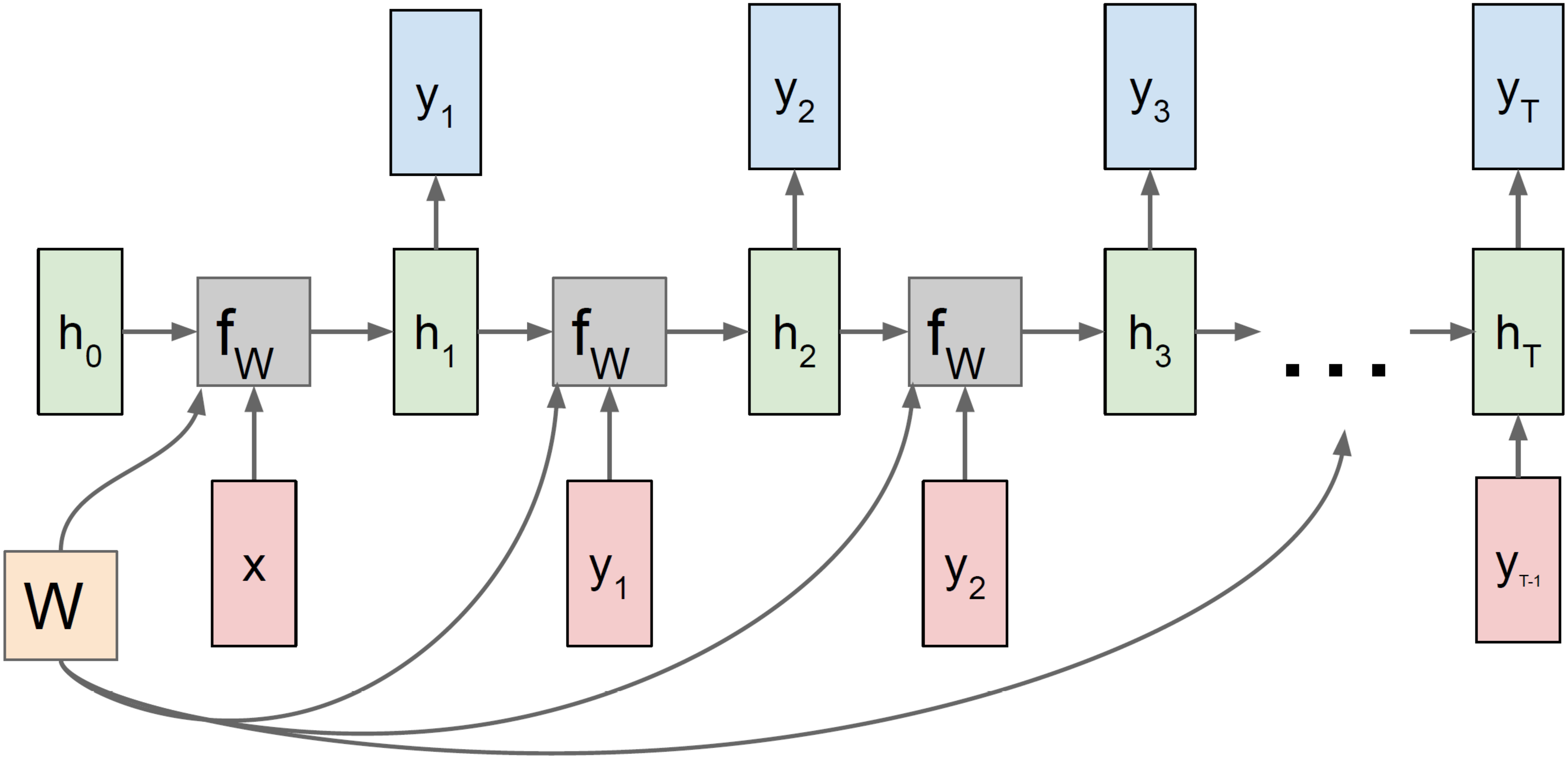
RNN: Computational graph: Many-to-one



RNN: Computational graph: Many-to-one



RNN: Computational graph: One-to-many

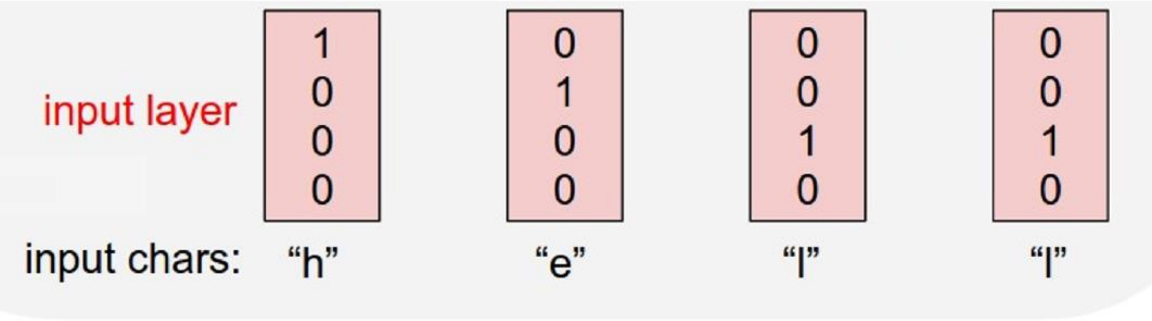


Language Modeling Example

Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



input layer	1 0 0 0	0 1 0 0	0 0 1 0	0 0 1 0
input chars:	“h”	“e”	“l”	“l”

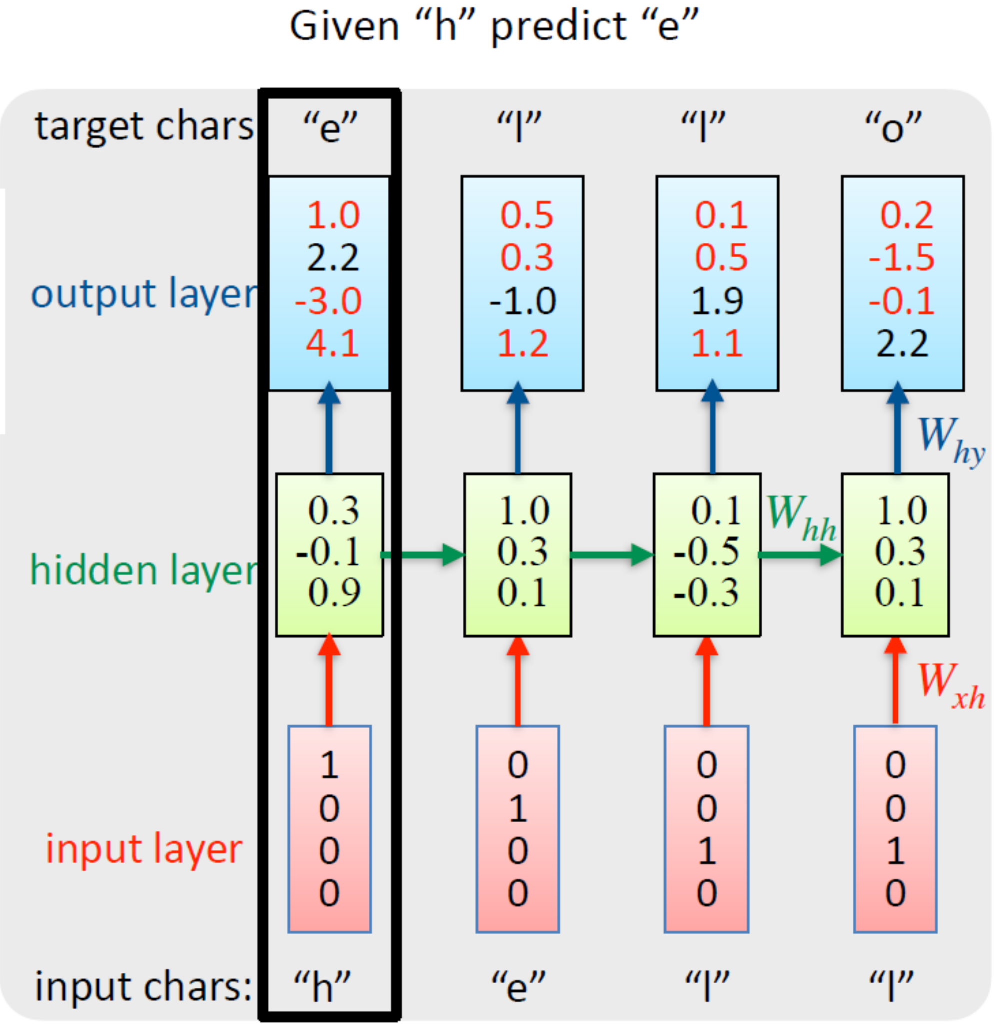
Language Modeling Example

$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

Training sequence: “hello”
Vocabulary: [h, e, l, o]

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \text{"h"} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \text{"e"}$$

$$\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \text{"l"} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \text{"o"}$$



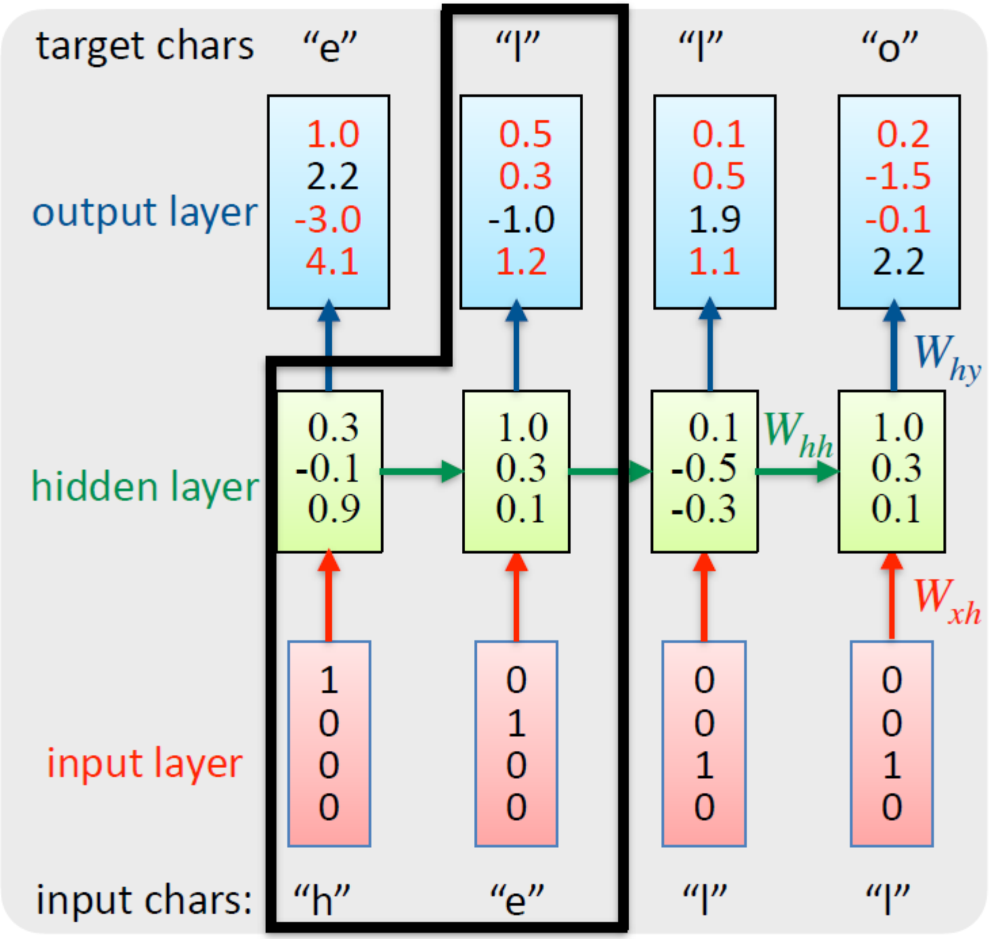
Language Modeling Example

Training sequence: “hello”
Vocabulary: [h, e, l, o]

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \text{"h"} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \text{"e"}$$

$$\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \text{"l"} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \text{"o"}$$

Given “he” predict “l”



Language Modeling Example

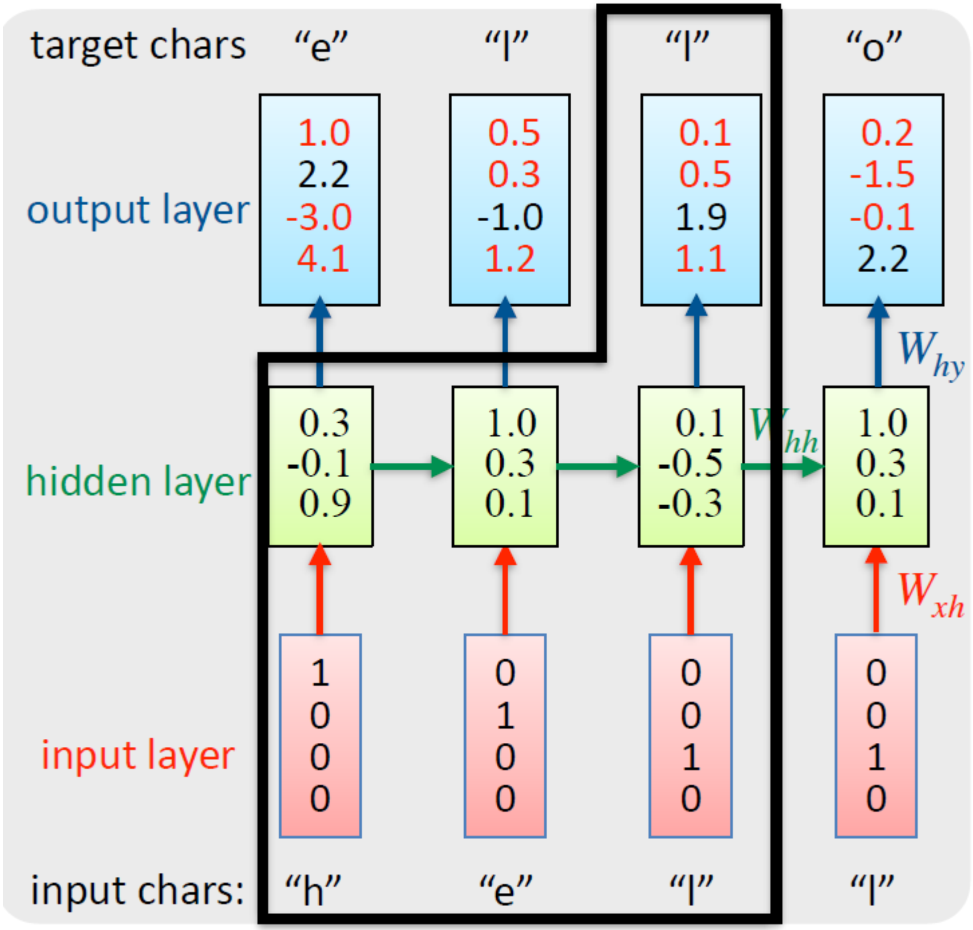
Training sequence: “hello”

Vocabulary: [h, e, l, o]

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \text{"h"} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \text{"e"}$$

$$\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \text{"l"} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \text{"o"}$$

Given “hel” predict “l”



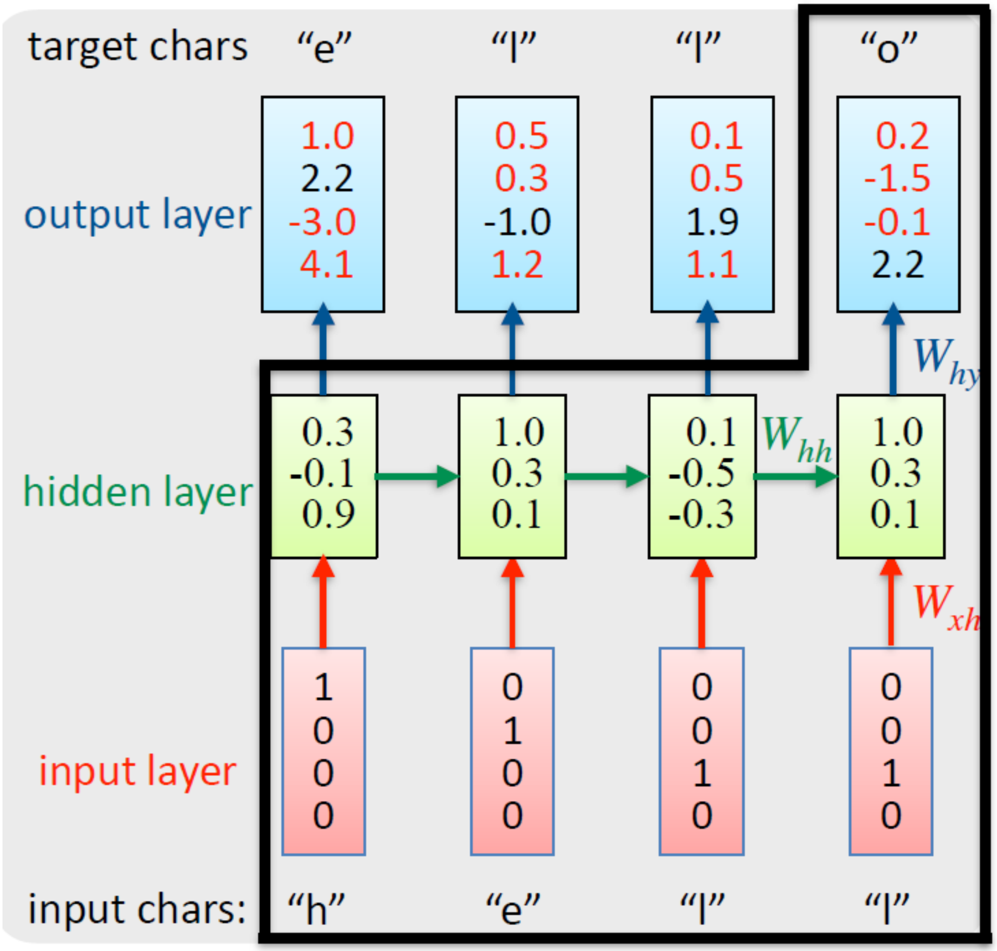
Language Modeling Example

Training sequence: “hello”
Vocabulary: [h, e, l, o]

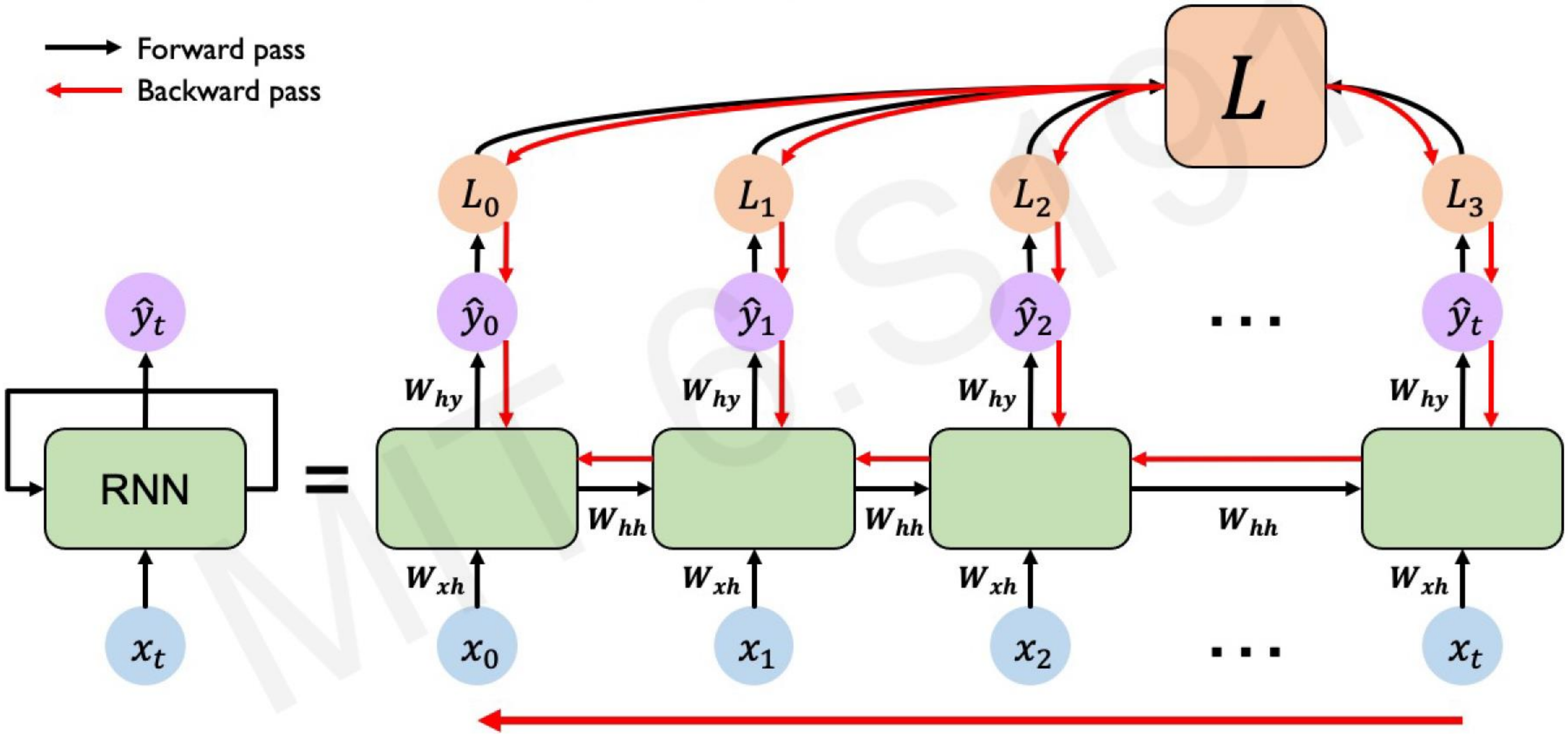
$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = "h" \qquad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = "e"$$

$$\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = "l" \qquad \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = "o"$$

Given “hell” predict “o”

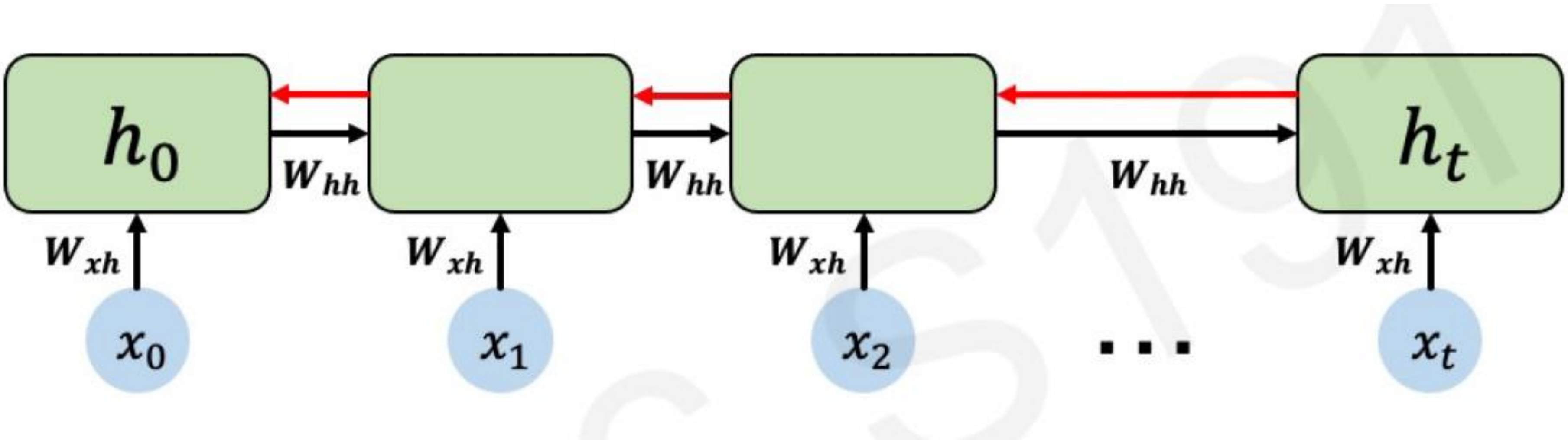


RNN Training -Backpropagation Through Time



Unfold the network across time steps and apply backpropagation to calculate gradients and update weights throughout the entire temporal sequence.

RNN Training -Backpropagation Through Time



RNN Gradient flow

Computing the gradient wrt h_0 involves **many factors of** W_{hh} + **repeated gradient computation!**

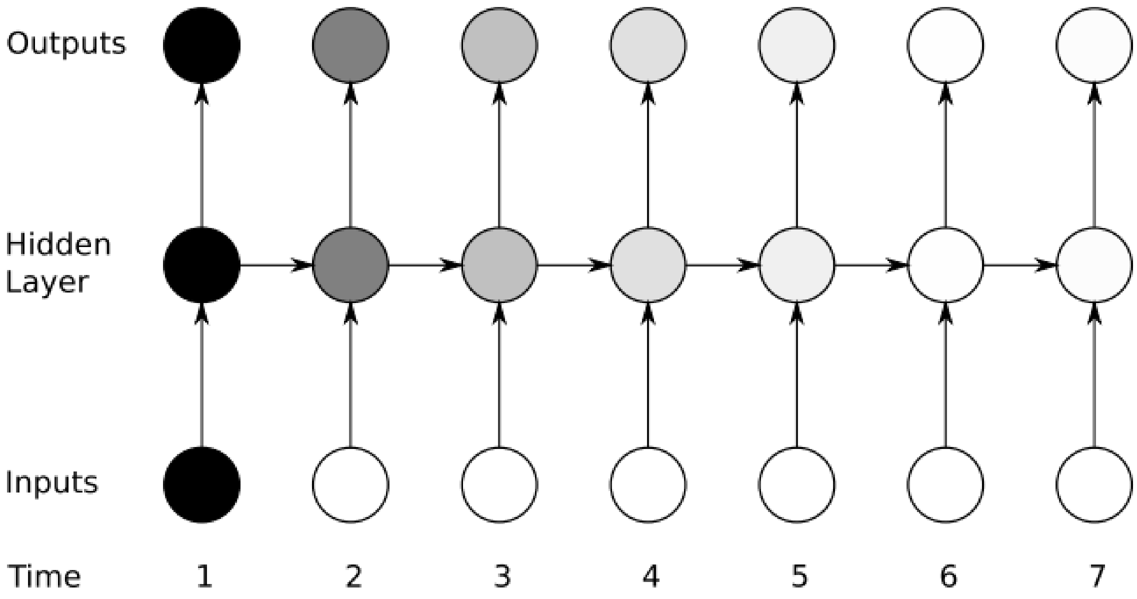
Many values >1 → **Exploding** Gradient

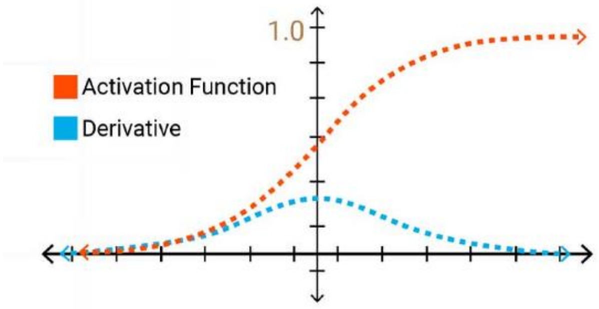
Many values <1 → **Vanishing** Gradient

Vanishing Gradient Over Time

This is more problematic in vanilla RNN (with tanh/sigmoid activation)

- When trying to handle long temporal dependency
- The gradient vanishes over time



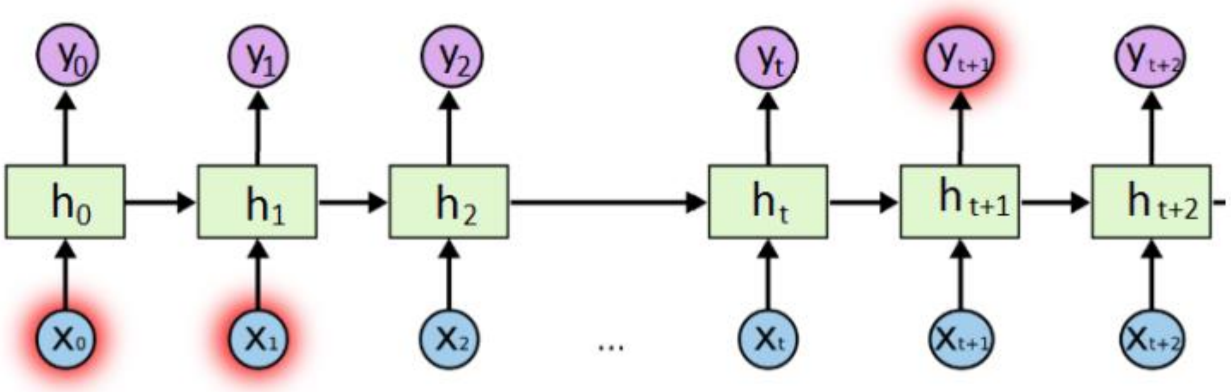


The Problem of Long-term Dependencies

RNNs are good with short sequences such as:
The sky is ….   (Answer: blue)

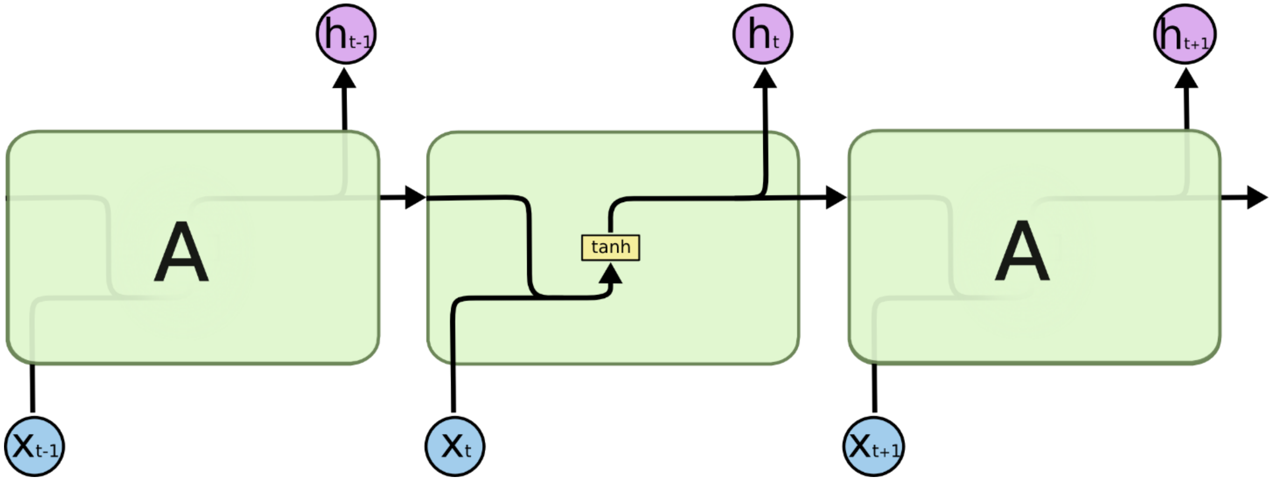
But..

This is the 10th day of wildfires in California area. There is smoke everywhere, and the sky is ….

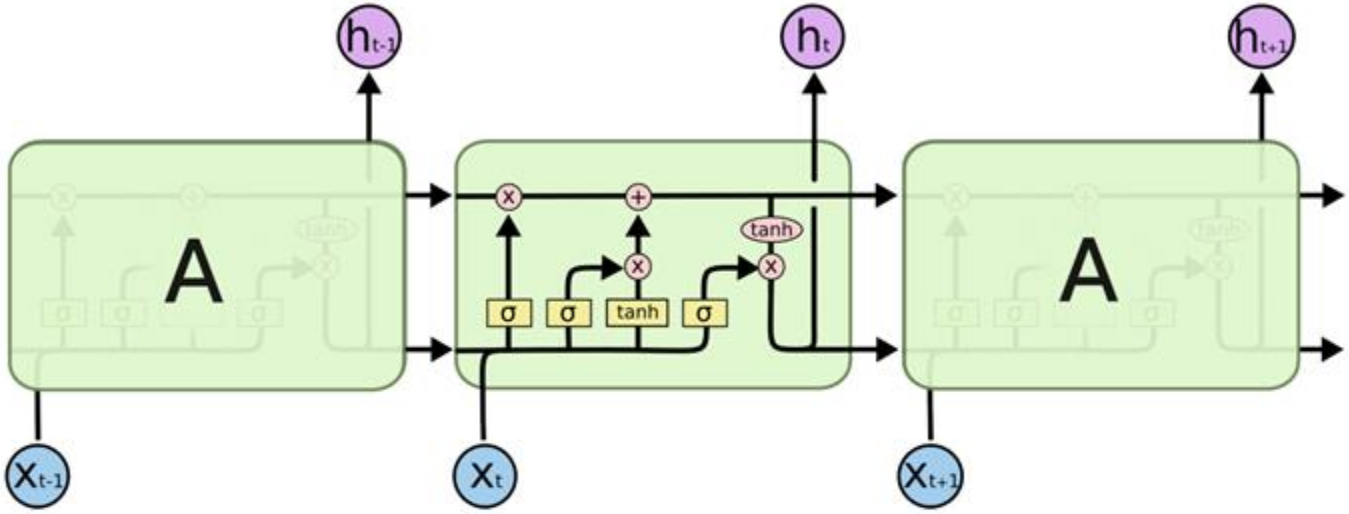


Long Short-Term Memory (LSTM)

RNN

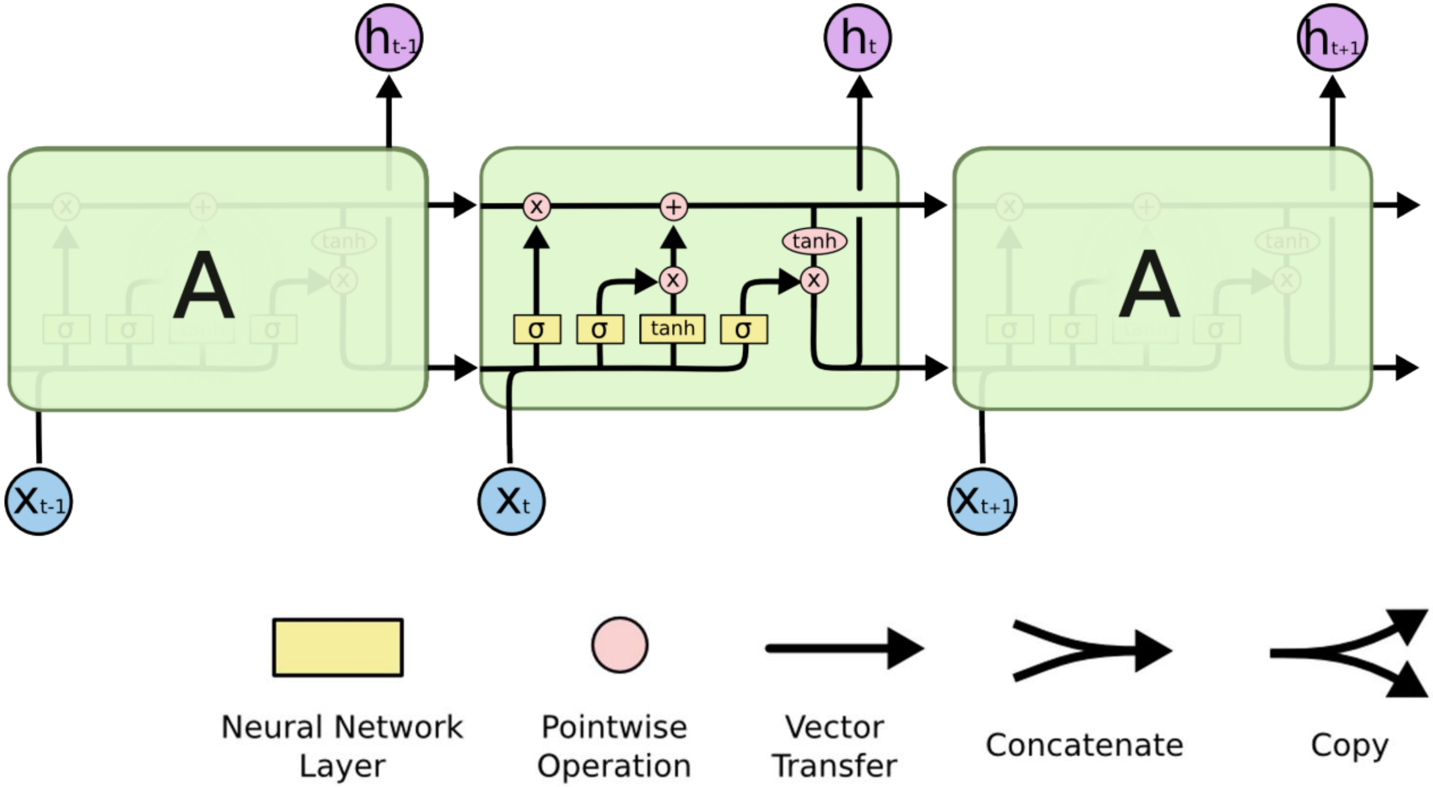


LSTM



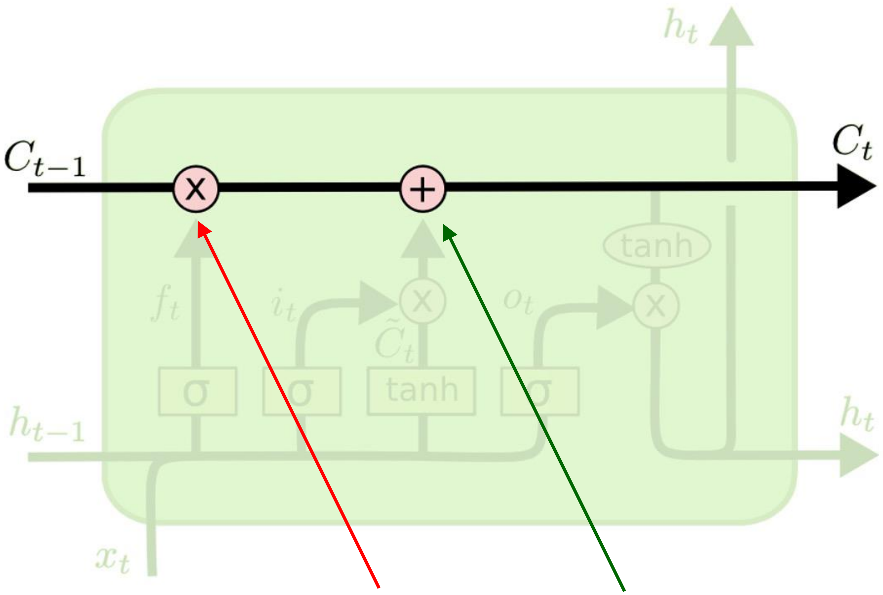
Long Short-Term Memory (LSTM)

LSTM networks are RNNs capable of learning long-term dependencies



LSTM: Cell State (long-term memory)

The core idea behind LSTMs is the **cell state**.



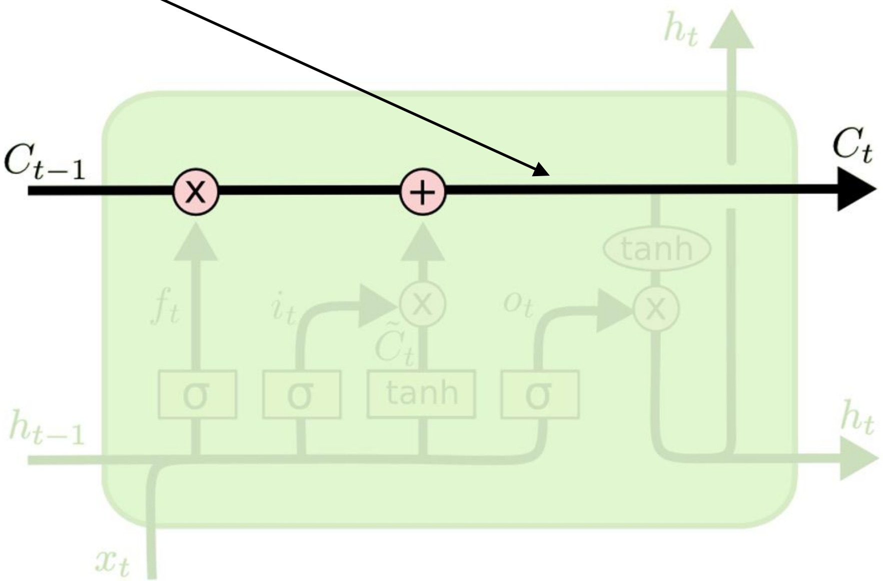
Information flows through this path

The LSTM has the ability to **remove** or **add** information to the cell state: thanks to **gates**

A stable pathway mitigates the vanishing gradient problem and is the core mechanism enabling LSTMs to capture long-range dependencies.

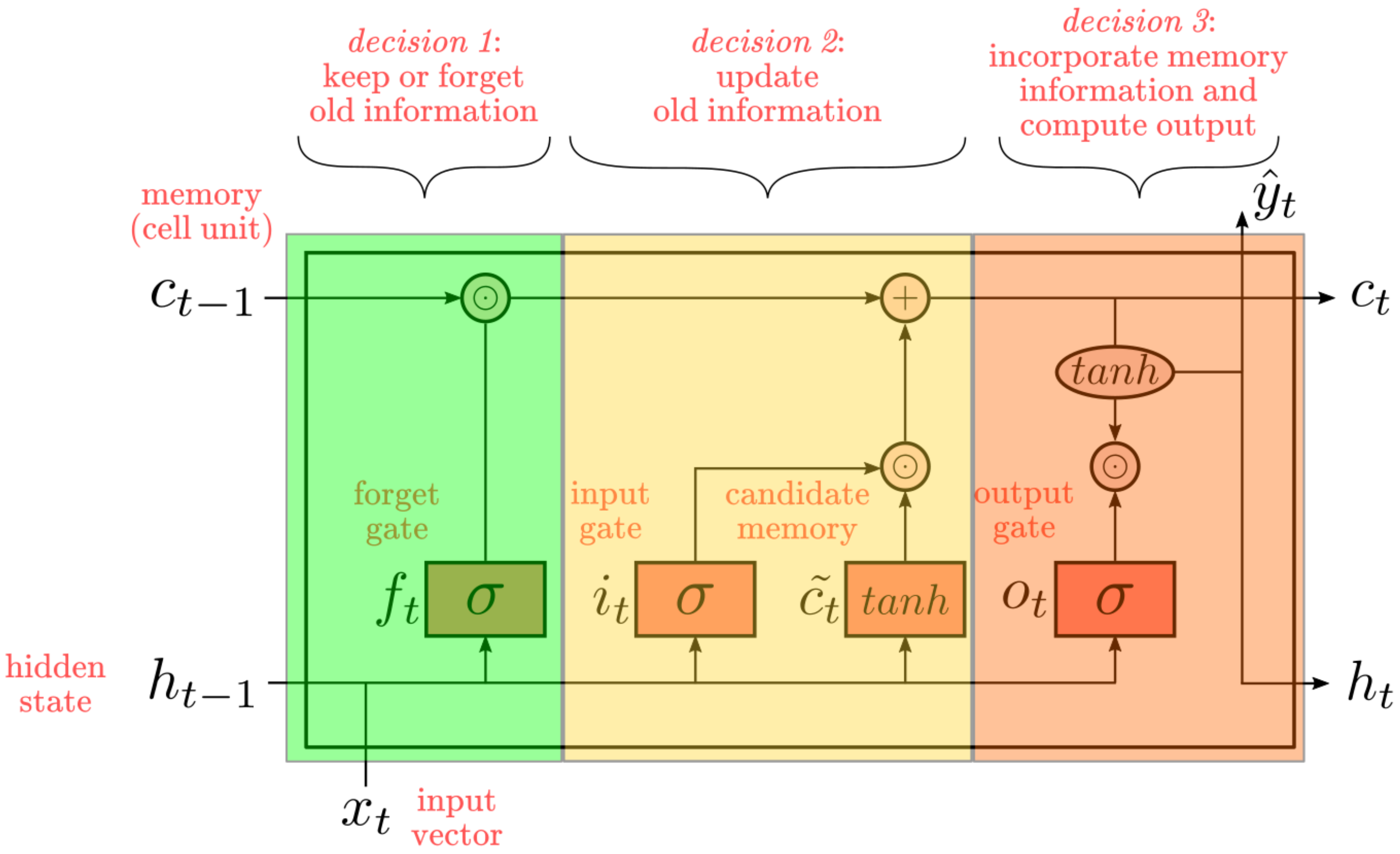
LSTM: Cell State (long-term memory)

No weights



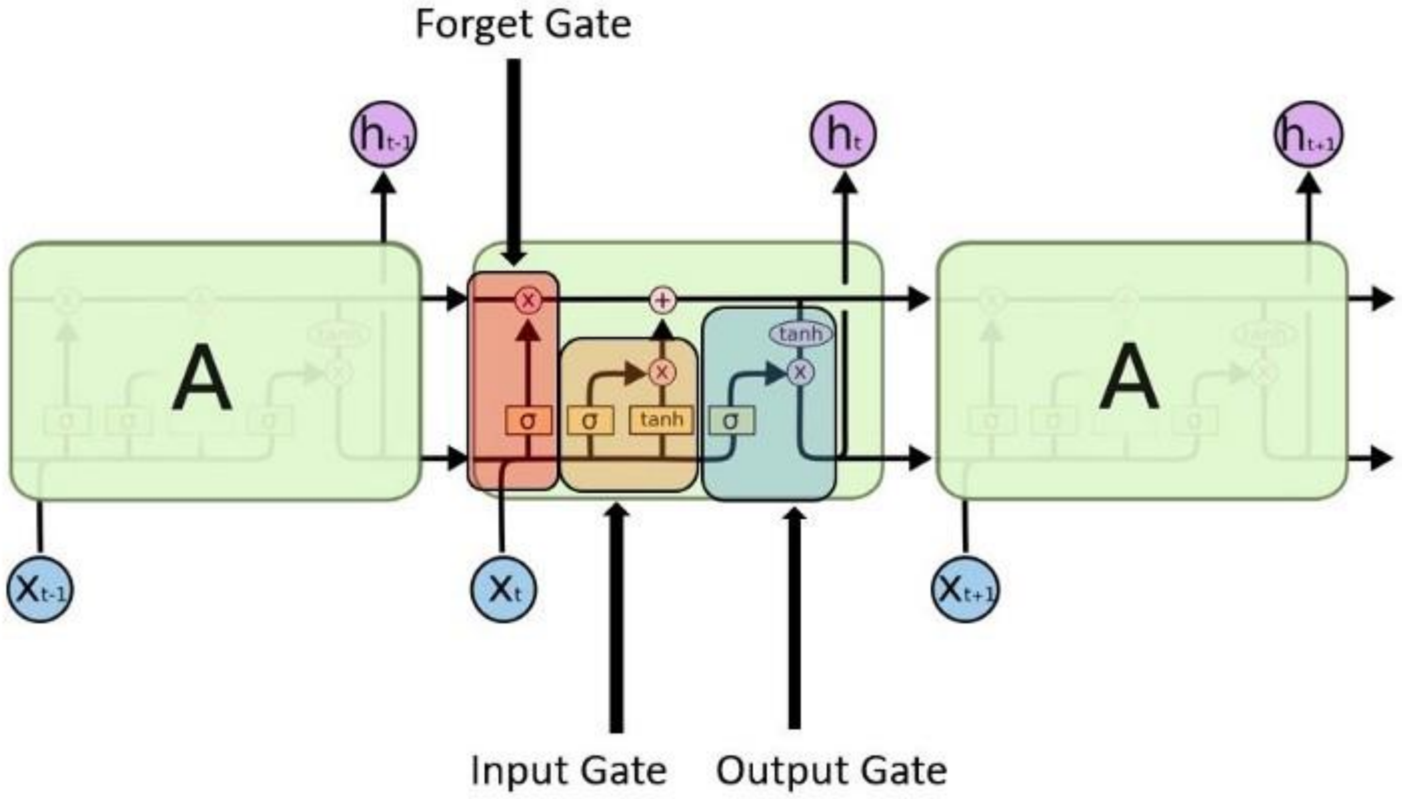
The lack of weights in this path allows long term memories to flow through a series of unrolled units without gradient vanishing/exploding

LSTM gates



The gates are the components that *regulate* what information flows onto, stays on, and gets output from this conveyor belt.

LSTM gates

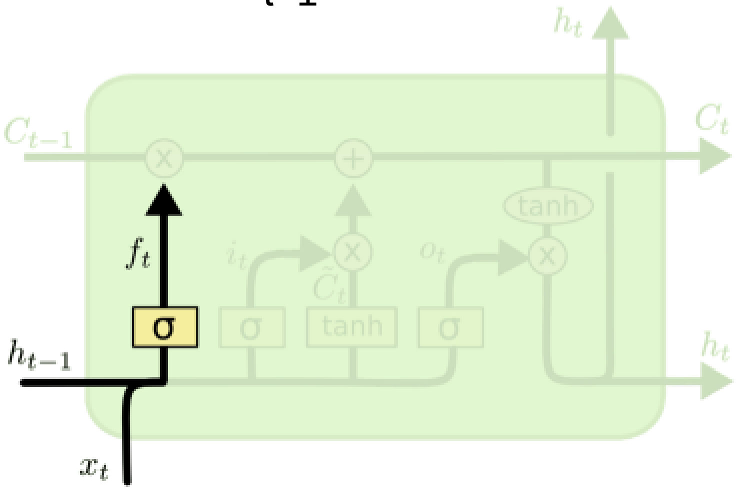


1. **Forget gate**: Controls what old information to discard
2. **Input gate**: Controls what new information to store
3. **Output gate**: Controls what to output

LSTM gates: Forget Gate

Step:1 Decides how much (what percentage) of the past you should remember.

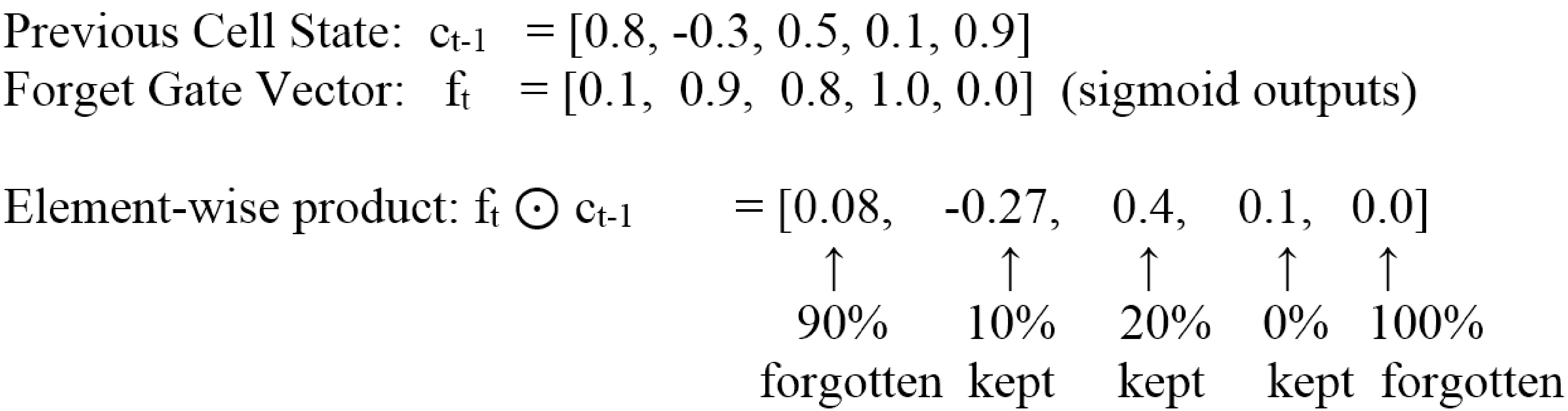
This gate decides which information to be omitted in from the cell in that particular time stamp. It is decided by the sigmoid function. it looks at the previous state(h_{t-1}) and the content input(X_t) and outputs a number between 0(omit this) and 1(keep this) for each number in the cell state C_{t-1}.



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

1 represents “completely keep this”
0 represents “completely forget this.”

LSTM gates: Forget Gate

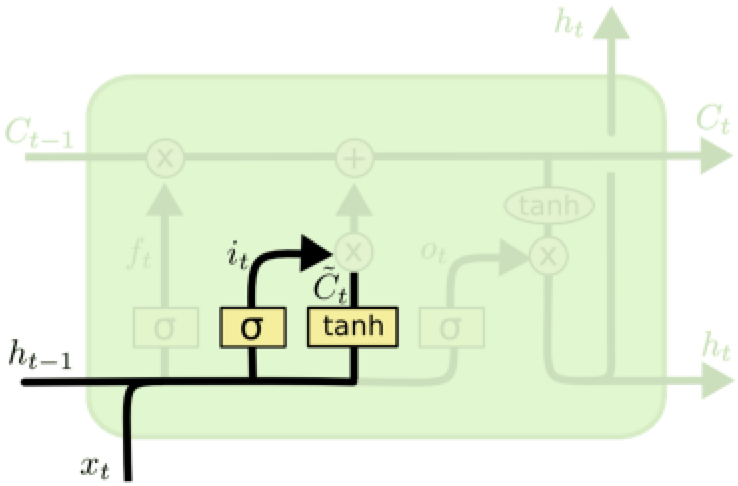


Previous Cell State: c_{t-1} = [0.8, -0.3, 0.5, 0.1, 0.9]
Forget Gate Vector: f_t = [0.1, 0.9, 0.8, 1.0, 0.0] (sigmoid outputs)

Element-wise product: $f_t \odot c_{t-1}$ = [0.08, -0.27, 0.4, 0.1, 0.0]
↑ ↑ ↑ ↑ ↑
90% 10% 20% 0% 100%
forgotten kept kept kept forgotten

LSTM gates: Input Gate

Step2: Decide what new information we're going to store in the cell state.



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

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

Input gate layer: decides which values we will update

Tanh layer: creates a vector of new candidate values

LSTM gates: Input Gate

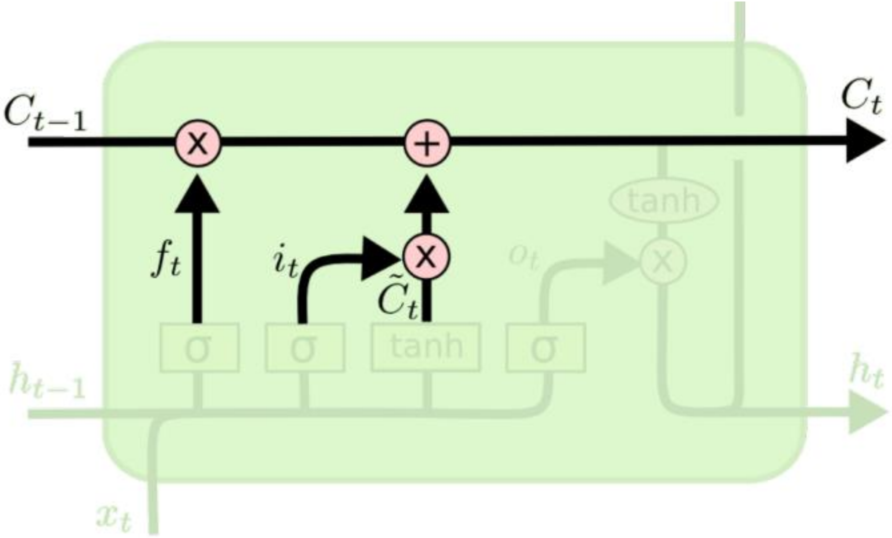
Sigmoid function decides which values to let through 0,1. tanh function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1.

EX1: **'The movie's special effects were simply amazing**,' where the sigmoid gate allowed the word 'amazing' to pass through with full strength (1) and the tanh function then weighted its emotional value as **+0.93** to reflect its strong positive importance."

EX2: **'The service was shockingly awful**,' where the sigmoid gate identified 'awful' as a key sentiment marker and allowed it through with full force (~1), and the tanh function then weighted its emotional impact as **-0.95** to capture its severe negative importance.“

LSTM gates: Input Gate

Step 3: Update the cell state



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

Combines:

- What to forget (forget gate × old cell state)
- What to add (input gate × candidate cell state)

Example with Numbers

Suppose for a single memory cell:

C_{t-1} = 0.8 (old memory: "it's sunny")

f_t = 0.9 (keep 90% of old memory)

i_t = 0.7 (add 70% of new candidate)

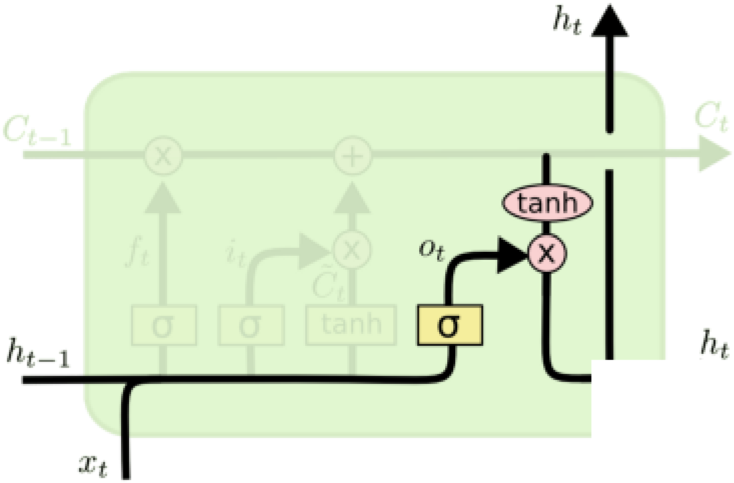
$\tilde{C}_t$ = 0.6 (candidate: "temperature is 25°

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$
$$= 0.9 \times 0.8 + 0.7 \times 0.6$$
$$= 0.72 + 0.42$$

= 1.14 (new cell state: "it's sunny and temperature is 25°)

LSTM gates: Output Gate

Step4: Decides which part of the current cell makes it to the output. (Decide what is the output)



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

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

Sigmoid layer decides which part of cell state is selected for output.

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

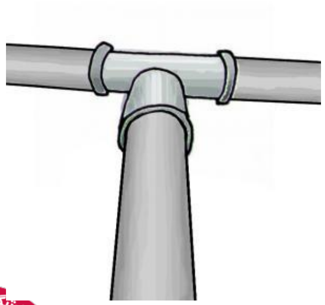
$$h_t = o_t * \tanh\left(C_t\right)$$

tanh layer gives weights to the values (-1 to 1).

LSTM: Analogy

- The **Sigmoid gates** are like the **faucet handles**. It decides how much of the liquid gets through (0% to 100%).

- The **Tanh function** is like the **water treatment plant**. It creates the clean, normalized water (candidate information) that is ready to flow into the pipe (the cell state). It determines the *property* of the liquid (e.g., its pH level, scored from -1 [acidic] to +1 [alkaline])



- The Cell State (C_t) is the main pipe itself, carrying water over long distances with little loss.

This elegant combination of a "decision-making" function (Sigmoid) and a "content-creating" function (Tanh), coupled with the additive cell state update, is what gives the LSTM its remarkable ability to capture long-term dependencies.

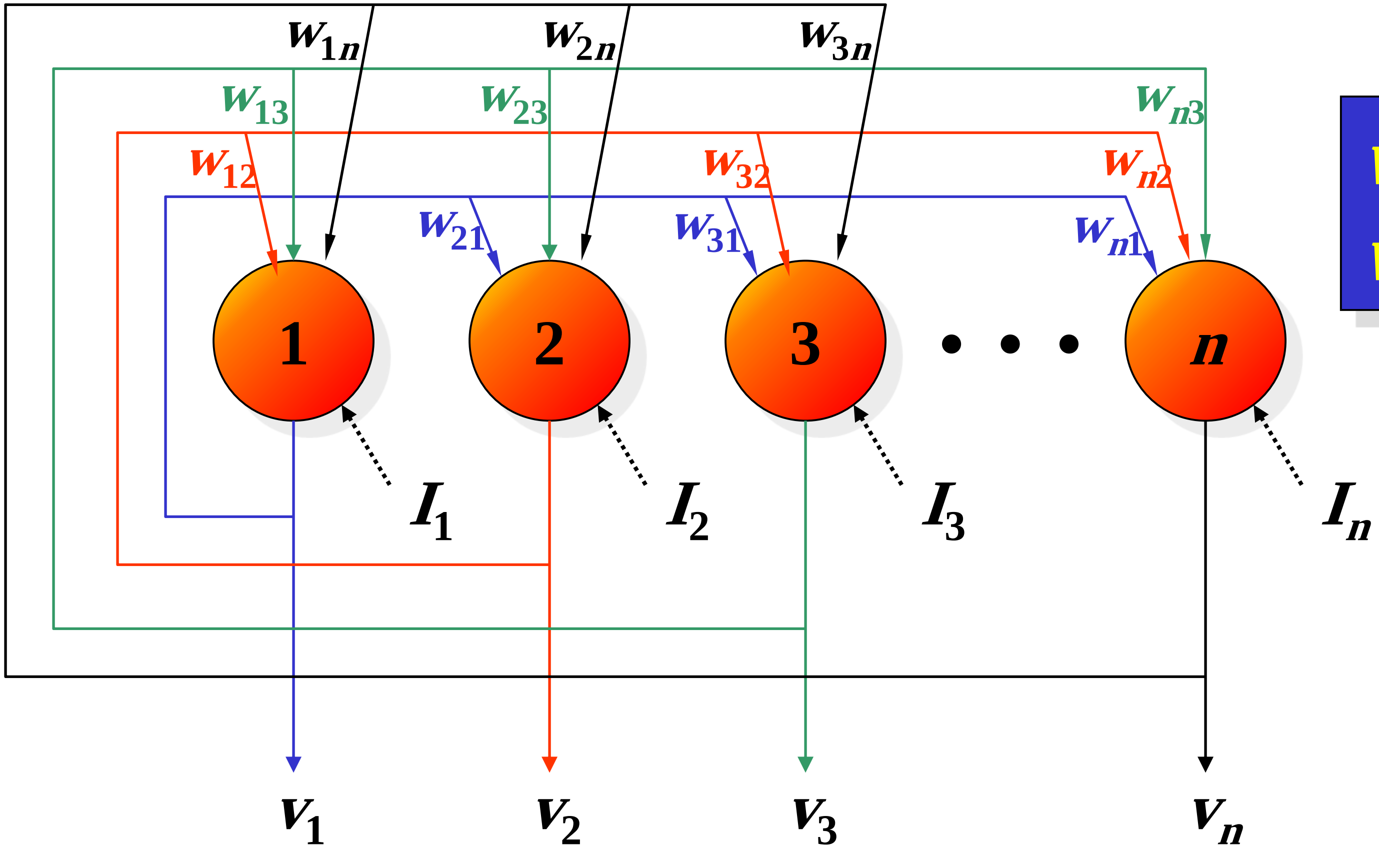
Intelligence emerges not from what you know, but from what you choose to forget. LSTMs encode this wisdom mathematically—they become intelligent by learning what's worth ignoring.

Hopfield Neural network

- Hopfield networks are energy-based **recurrent neural networks** that function as associative memories, converging to stable attractor states to store and retrieve patterns through symmetric connections and Hebbian learning.

- It is a single layer neural network with feedbacks.

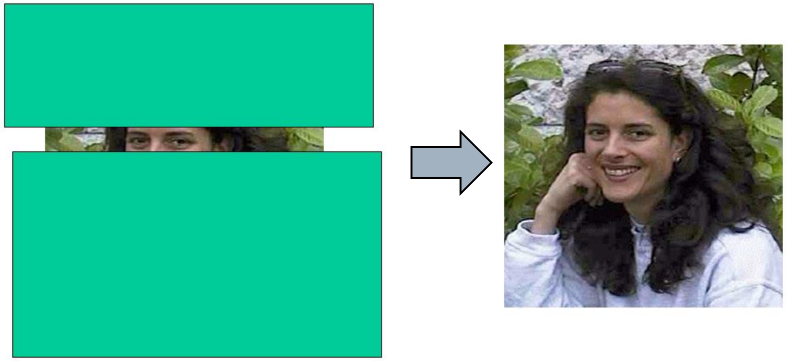
Hopfield Neural network

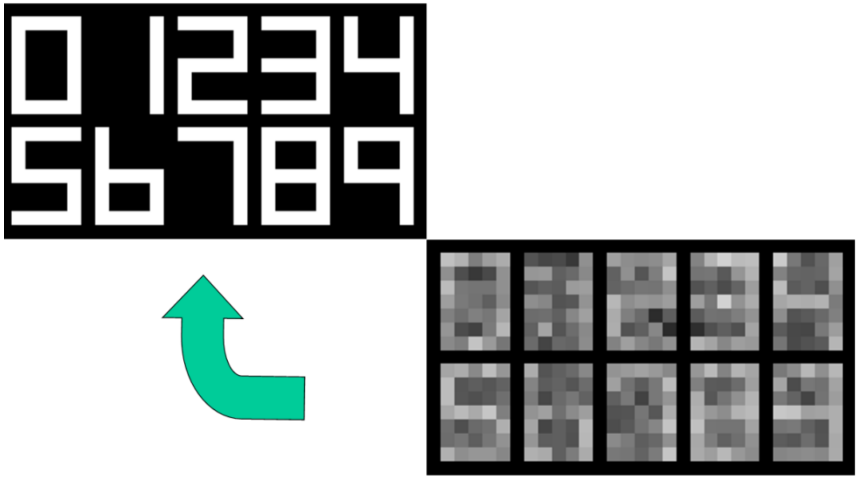


$$w_{ij} = w_{ji}$$

$$w_{ii} = 0$$

Hopfield Neural network





Hopfield Neural network

Hebbian learning in Hopfield networks follows the principle "neurons that fire together, wire together" to create associative memory

To store a set of binary patterns, the weight matrix W = is given by:

$$w_{ij} = \sum_{p} \left(2s_i(p) - 1\right)\left(2s_i(p) - 1\right), \quad i \neq j \quad ; \qquad w_{ii} = 0$$

To store a set of bipolar patterns, the weight matrix W = is given by:

$$w_{ij} = \sum_{p} s_i(p) s_j(p), \quad i \neq j \quad ; \quad w_{ii} = 0$$

Hopfield Neural network

Step 0. Initialize weights to store patterns (Hebbian rule).

While activations of the net are not converged, do Steps 1-7.

Step 1. For each input vector x, do Steps 2-6.

Step 2. Set initial activations of net equal to the external input vector x: $y_i = x_i \quad , \quad i = 1, \ldots, n$

Step 3. Do Steps 4-6 for each unit (Units should be updated in random order.)

Step 4. Compute net input: $y\_in_i = x_i + \sum_j y_j w_{ji}$

Step 5. Determine activation (output signal):

$$y_i = \begin{cases} 1 & if \quad y\_in_i > \theta_i \\ y_i & if \quad y\_in_i = \theta_i \\ 0 & if \quad y\_in_i < \theta_i . \end{cases}$$

Step 6. Broadcast the value of y_i to all other units. (This updates the activation vector.)

Step 7. Test for convergence.

Hopfield Neural network - example

The vector (1, 1, 1,0) (or its bipolar equivalent (1, 1, 1, - 1)) was stored in a net

The weight matrix is bipolar $$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

The input vector is x = (0, 0, 1, 0)

For this example the update order is $Y_1 \quad Y_4 \ Y_3 \quad Y_2$

Hopfield Neural network - example

$$y = (0, 0, 1, 0)$$

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Choose unit Y_1 to update its activation:

$$y\_in_1 = x_1 + \sum_j y_j w_{j1} = 0 + 1$$

$$y\_in_1 > 0 \rightarrow y_1 = 1 \quad \Rightarrow \quad y = (\bigcirc\!\!\!\!1, 0, 1, 0)$$

Choose unit Y_4 to update its activation:

$$y\_in_4 = x_4 + \sum_j y_j w_{j4} = 0 + (-2)$$

$$y\_in_4 < 0 \rightarrow y_4 = 0 \quad \Rightarrow \quad y = (1, 0, 1, 0)$$

Hopfield Neural network - example

Choose unit Y_3 to update its activation:

$$y = (1, 0, 1, 0)$$

$$y\_in_3 = x_3 + \sum_j y_j w_{j3} = 1 + 1$$

$$y\_in_3 > 0 \rightarrow \quad y_3 = 1 \quad \Rightarrow \quad y = (1, 0, 1, 0)$$

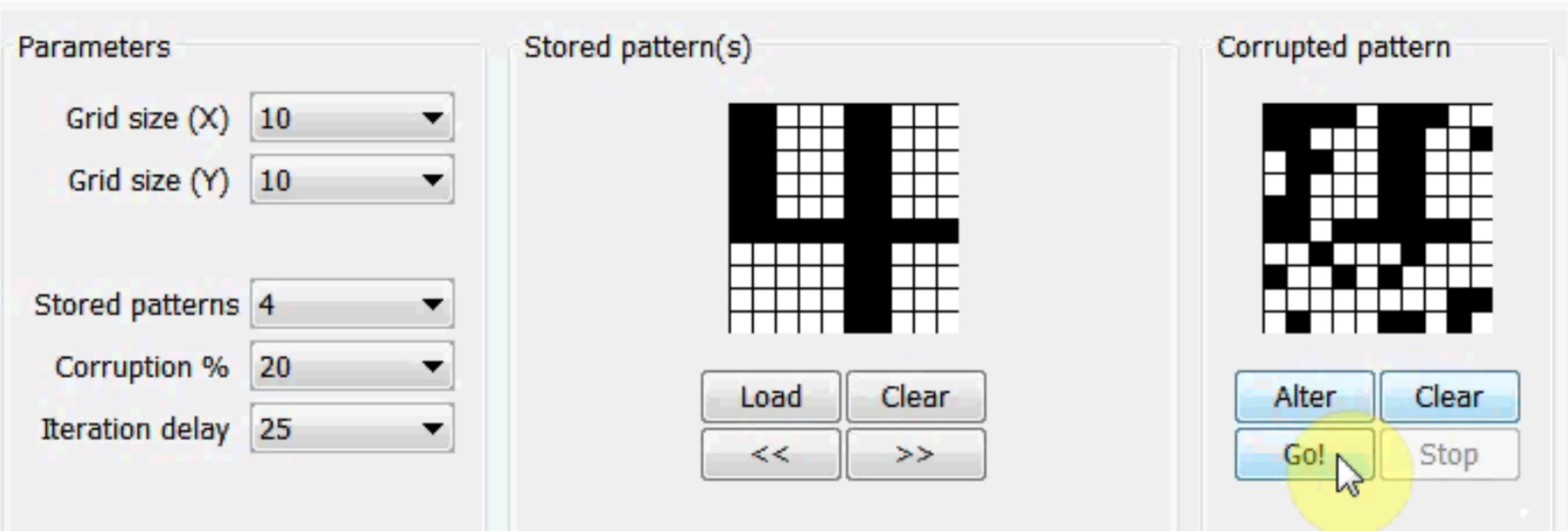
Choose unit Y_2 to update its activation:

$$y\_in_2 = x_2 + \sum_j y_j w_{j2} = 0 + 2$$

$$y\_in_2 > 0 \rightarrow \quad y_2 = 1 \quad \Rightarrow \quad y = (1, 1, 1, 0)$$

further iterations do not change the activation of any unit. The net has converged to the stored vector.

Hopfield Neural network - example



Parameters

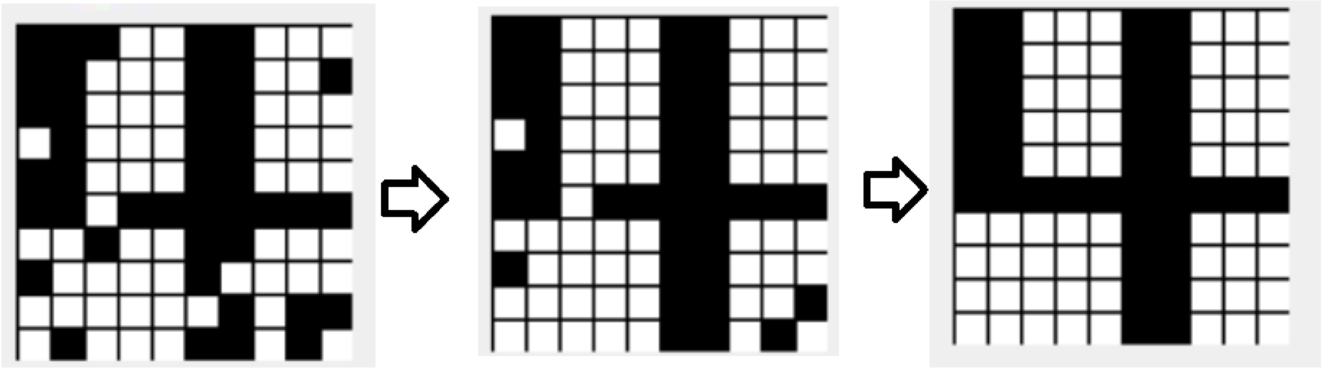
Grid size (X) 10

Grid size (Y) 10

Stored patterns 4

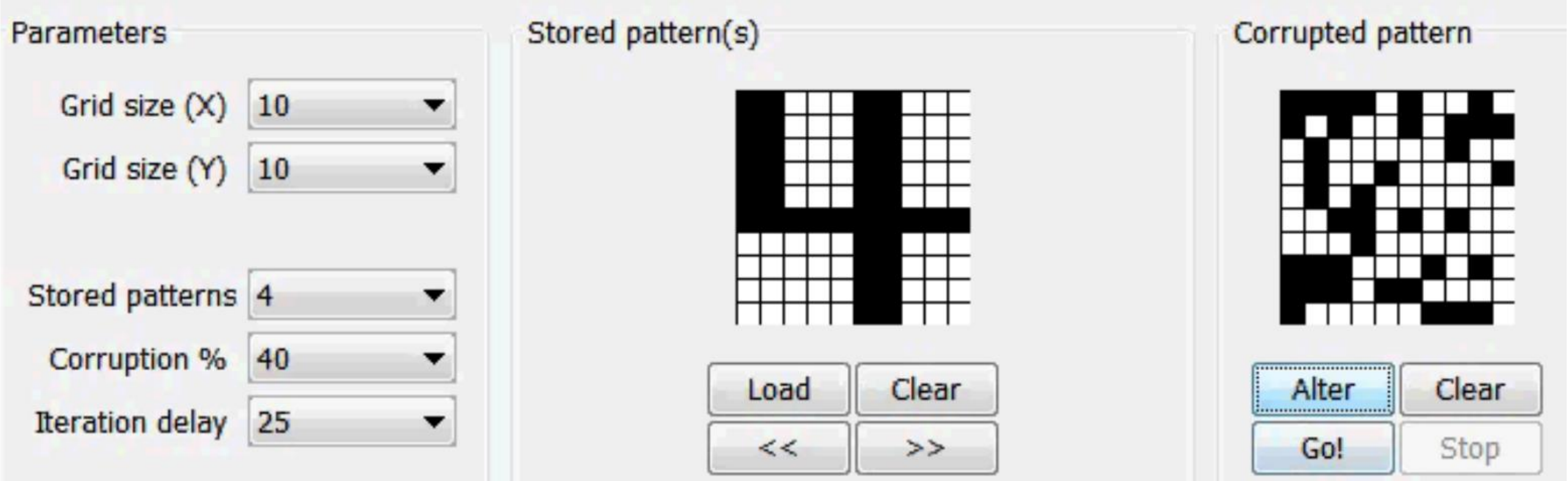
Corruption % 20

Iteration delay 25



Perfect Recall

Hopfield Neural network - example



Parameters

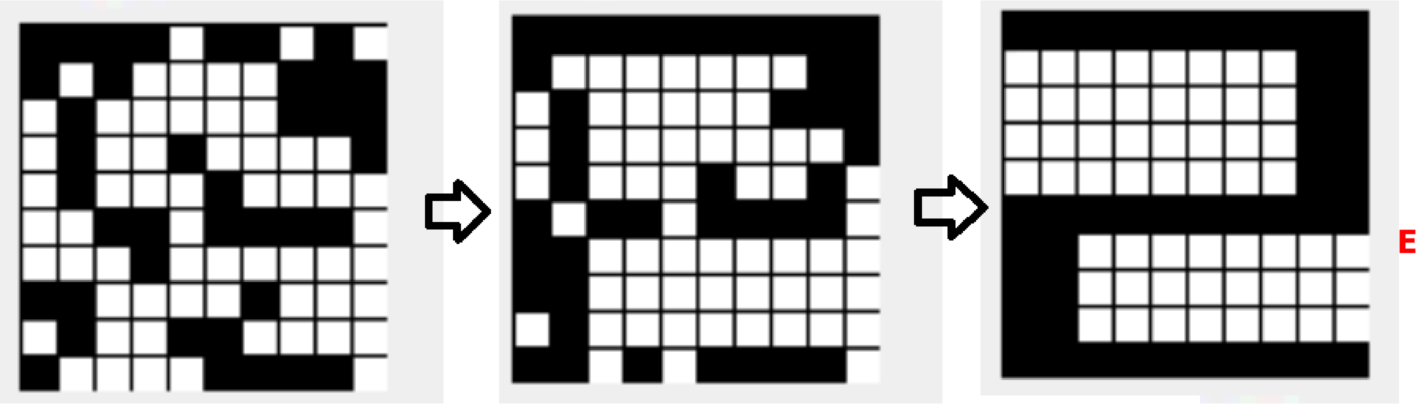
Grid size (X) 10

Grid size (Y) 10

Stored patterns 4

Corruption % 40

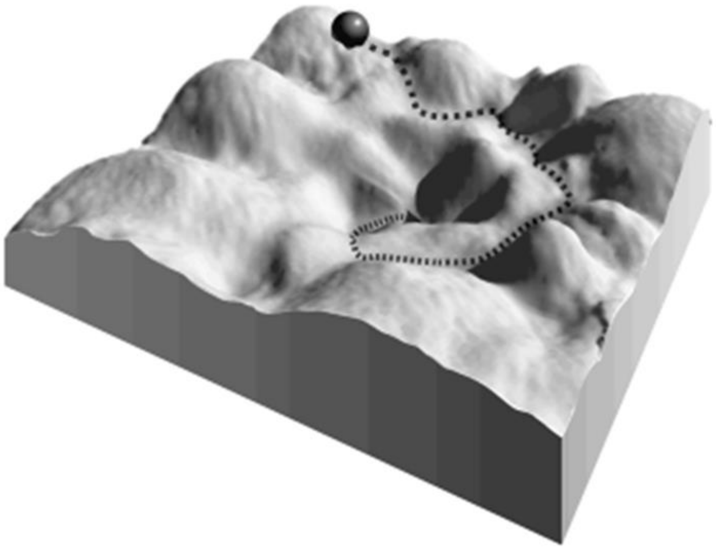
Iteration delay 25



Erroneous Recall

Hopfield network- Energy function

Hopfield nets have a scalar value associated with each state of the network, referred to as the "energy", E, of the network, where:



$$E = -0.5 \sum_{i \neq j} \sum_{j} y_i y_j w_{ij} + \sum_{i} \theta_i y_i$$

$$\Delta E = -\left[\sum_{j} y_j w_{ij} - \theta_i \right] \Delta y_i$$

$$\Delta E < 0$$

a Hopfield network constantly decreases its energy

Hopfield network- example

Problem Statement

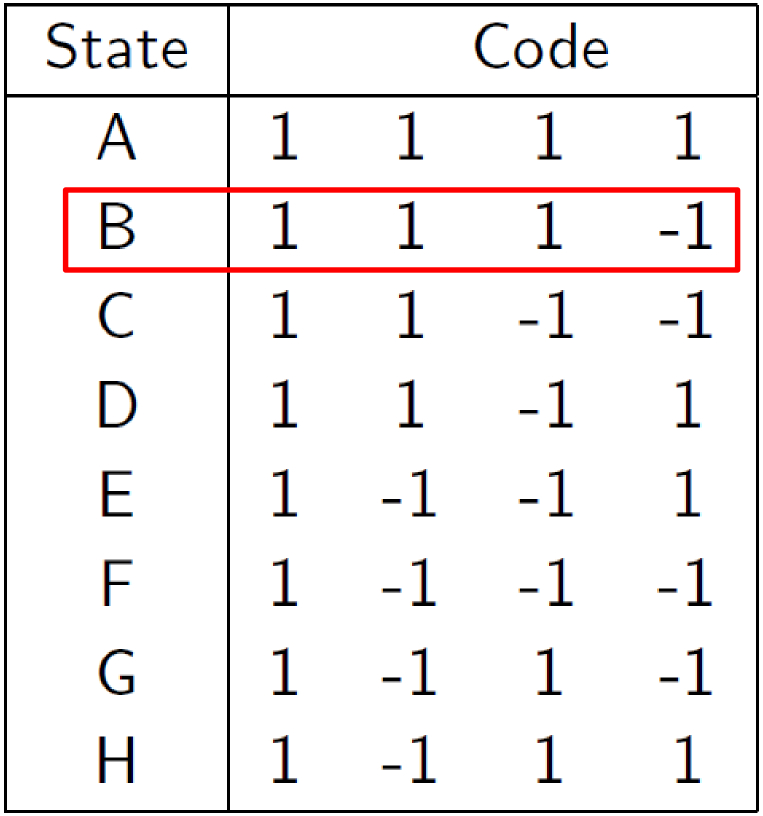
- We need to store a **fundamental pattern (memory)** given by the vector $O = [1, 1, 1, -1]^T$ in a four node binary Hopefield network.
- Presume that the threshold parameters are all equal to zero.

Establishing Connection Weights

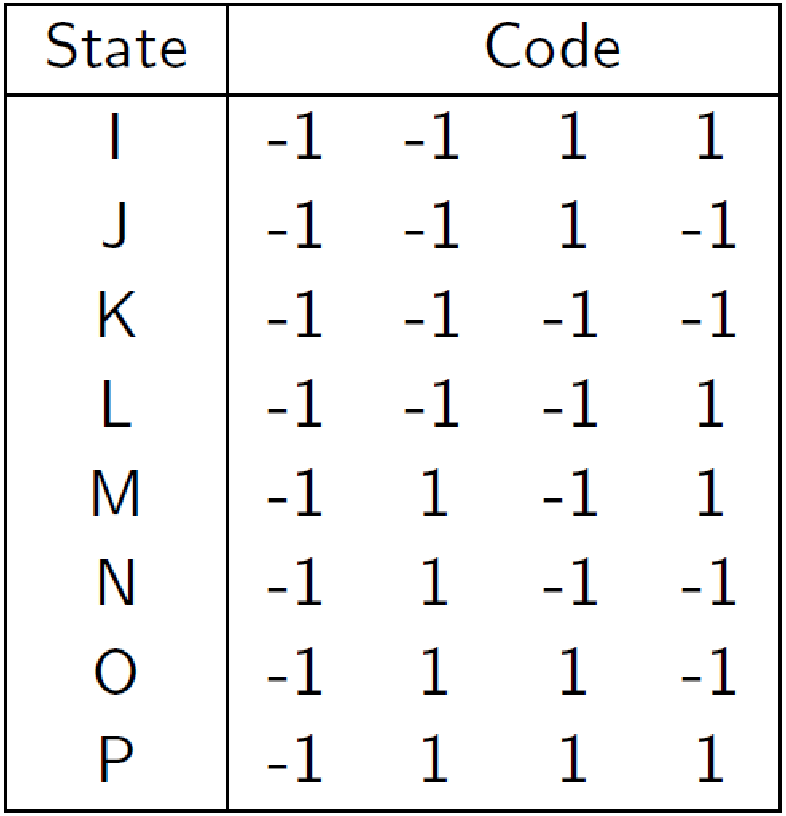
$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Hopfield network- example

Network' States and Their Code: Total number of states = 16



State	Code			
A	1	1	1	1
B	1	1	1	-1
C	1	1	-1	-1
D	1	1	-1	1
E	1	-1	-1	1
F	1	-1	-1	-1
G	1	-1	1	-1
H	1	-1	1	1



State	Code			
I	-1	-1	1	1
J	-1	-1	1	-1
K	-1	-1	-1	-1
L	-1	-1	-1	1
M	-1	1	-1	1
N	-1	1	-1	-1
O	-1	1	1	-1
P	-1	1	1	1

Hopfield network- example

Calculating energy function for all states: (θ=0)

$$E = -1/2 \sum_{i=1}^{4} \sum_{j=1}^{4} w_{ij} o_i o_j$$

$$\begin{aligned} E = -1/2(& w_{11} o_1 o_1 + w_{12} o_1 o_2 + w_{13} o_1 o_3 + w_{14} o_1 o_4 + \\ & w_{21} o_2 o_1 + w_{22} o_2 o_2 + w_{23} o_2 o_3 + w_{24} o_2 o_4 + \\ & w_{31} o_3 o_1 + w_{32} o_3 o_2 + w_{33} o_3 o_3 + w_{34} o_3 o_4 + \\ & w_{41} o_4 o_1 + w_{42} o_4 o_2 + w_{43} o_4 o_3 + w_{44} o_4 o_4) \end{aligned}$$

Hopfield network- example

For state A = [O_1, O_2, O_3, O_4] [1 , 1, 1, 1]

$$
\begin{aligned}
E = -1/2(0 &+ (1)(1)(1) + (1)(1)(1) + (-1)(1)(1)+ \\
&(1)(1)(1) + 0 + (1)(1)(1) + (-1)(1)(1)+ \\
&(1)(1)(1) + (1)(1)(1) + 0 + (-1)(1)(1)+ \\
&(-1)(1)(1) + (-1)(1)(1) + (-1)(1)(1) + 0)
\end{aligned}
$$

$$
\begin{aligned}
E = -1/2(0 &+ 1 + 1 - 1+ \\
&1 + 0 + 1 - 1+ \\
&1 + 1 + 0 - 1+ \\
&-1 - 1 - 1 + 0)
\end{aligned}
$$

$$E = -1/2(6 - 6) = 0$$

Hopfield network- example

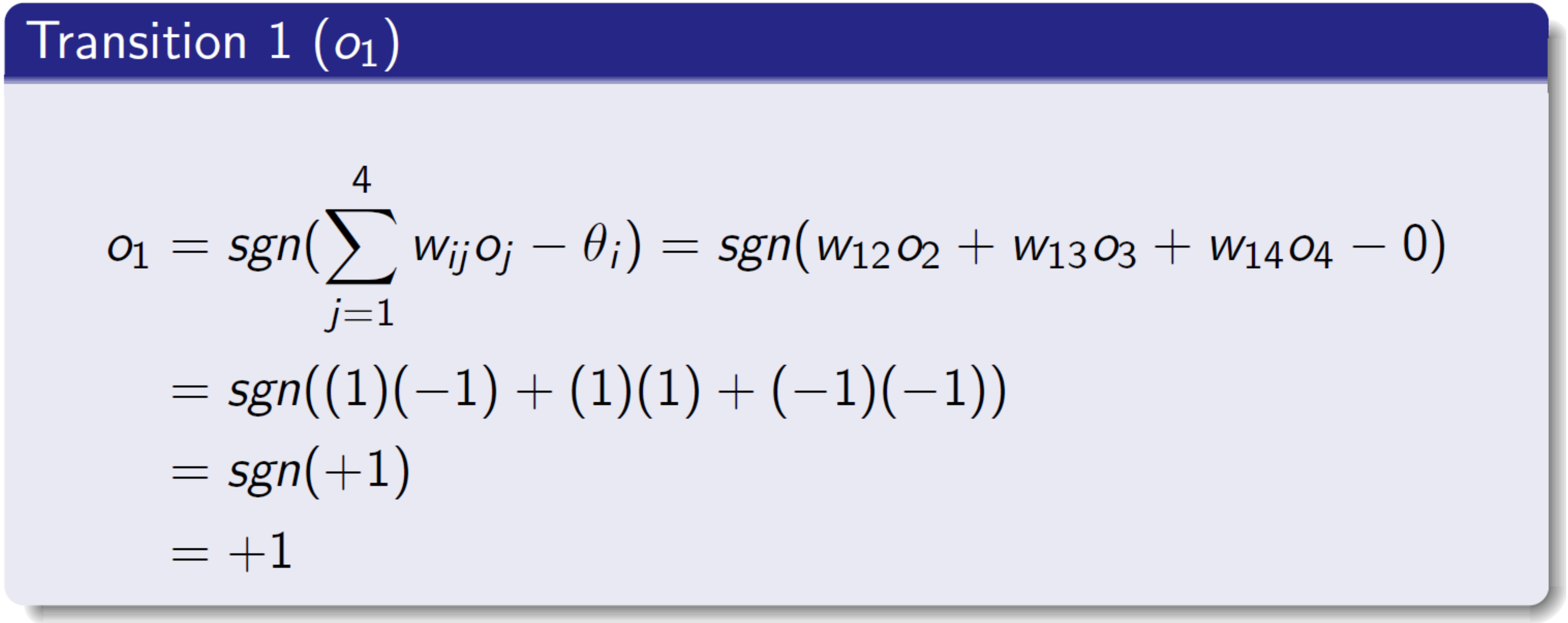
State	Code				Energy
A	1	1	1	1	0
B	1	1	1	-1	-6
C	1	1	-1	-1	0
D	1	1	-1	1	2
E	1	-1	-1	1	0
F	1	-1	-1	-1	2
G	1	-1	1	-1	0
H	1	-1	1	1	2
I	-1	-1	1	1	0
J	-1	-1	1	-1	2
K	-1	-1	-1	-1	0
L	-1	-1	-1	1	-6
M	-1	1	-1	1	0
N	-1	1	-1	-1	2
O	-1	1	1	-1	0
P	-1	1	1	-1	2

Similarly, we can compute the energy level of the other states.

Minimum energy (Stable states)

Hopfield network- example

State Transition for State J = [−1 , −1 , 1 , −1]



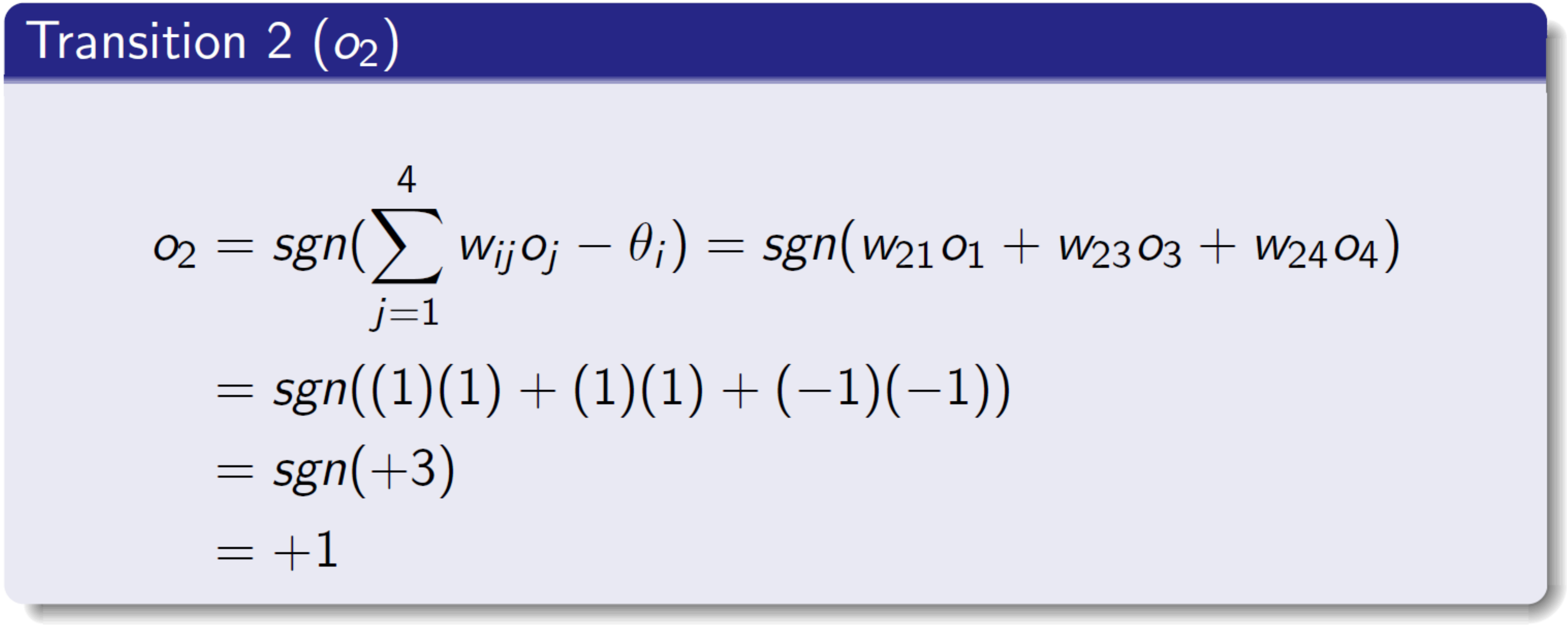
Transition 1 (o_1)

$$o_1 = sgn(\sum_{j=1}^{4} w_{ij}o_j - \theta_i) = sgn(w_{12}o_2 + w_{13}o_3 + w_{14}o_4 - 0)$$
$$= sgn((1)(-1) + (1)(1) + (-1)(-1))$$
$$= sgn(+1)$$
$$= +1$$

As a result, the first component of the state J changes from −1 to 1. In other words, the state J transits to the state G

$$J = [-1, -1, 1, -1]^T \ (2) \rightarrow G = [1, -1, 1, -1]^T \ (0)$$

Hopfield network- example



Transition 2 (o_2)

$$
\begin{aligned}
o_2 &= sgn(\sum_{j=1}^{4} w_{ij} o_j - \theta_i) = sgn(w_{21} o_1 + w_{23} o_3 + w_{24} o_4) \\
&= sgn((1)(1) + (1)(1) + (-1)(-1)) \\
&= sgn(+3) \\
&= +1
\end{aligned}
$$

As a result, the second component of the state G changes from -1 to 1. In other words, the state G transits to the state B

B= [1, 1, 1, -1]

Hopfield network- example

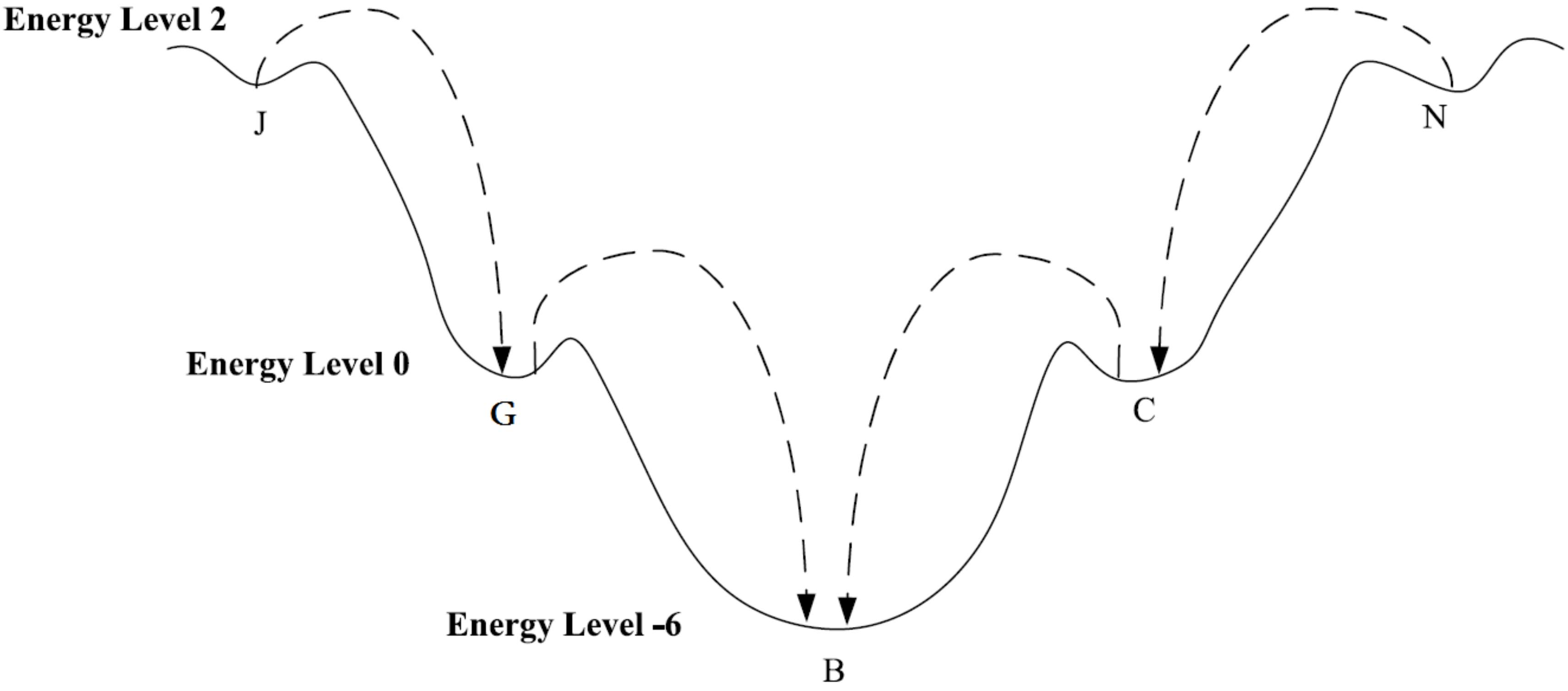
As state B is a fundamental pattern, no more transition will occur

$$
\begin{aligned}
o_3 &= sgn(\sum_{j=1}^{4} w_{ij} o_j - \theta_i) = sgn(w_{31} o_1 + w_{32} o_2 + w_{34} o_4) \\
&= sgn((1)(1) + (1)(1) + (-1)(-1)) \\
&= sgn(+3) \\
&= +1
\end{aligned}
$$

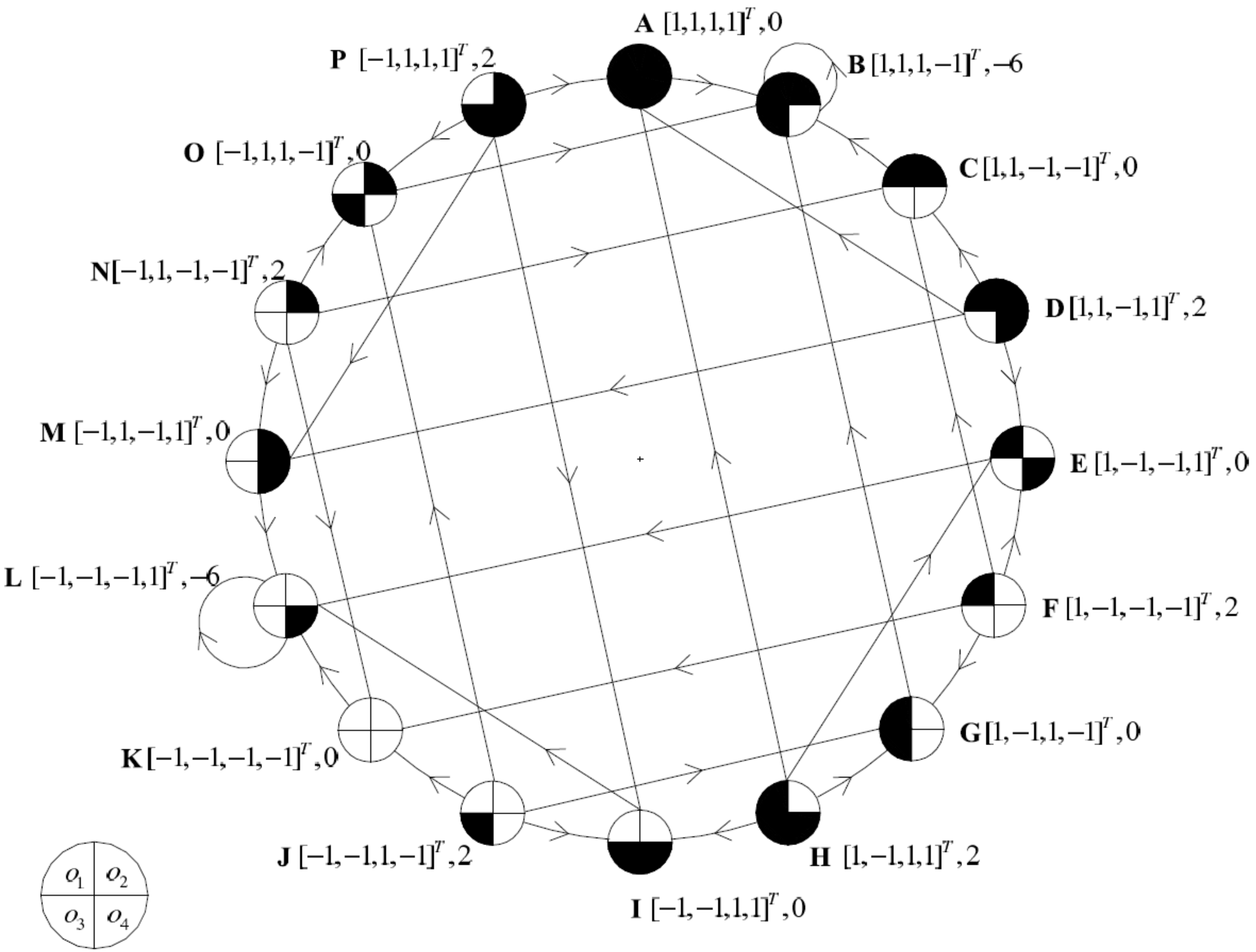
$$
\begin{aligned}
o_4 &= sgn(\sum_{j=1}^{4} w_{ij} o_j - \theta_i) = sgn(w_{41} o_1 + w_{42} o_2 + w_{43} o_3) \\
&= sgn((-1)(1) + (-1)(1) + (-1)(1)) \\
&= sgn(-3) \\
&= -1
\end{aligned}
$$

$$B = [1, 1, 1, -1]^T \ (-6) \rightarrow B = [1, 1, 1, -1]^T \ (-6)$$

Hopfield network- example



Hopfield network- example



Storage Capacity of Hopfield Net

- Binary

$$P \approx 0.15n$$

- Bipolar

$$P \approx \frac{n}{2\log_2 n}$$

P: # of patterns that can be stored an recalled in a net with reasonable accuracy

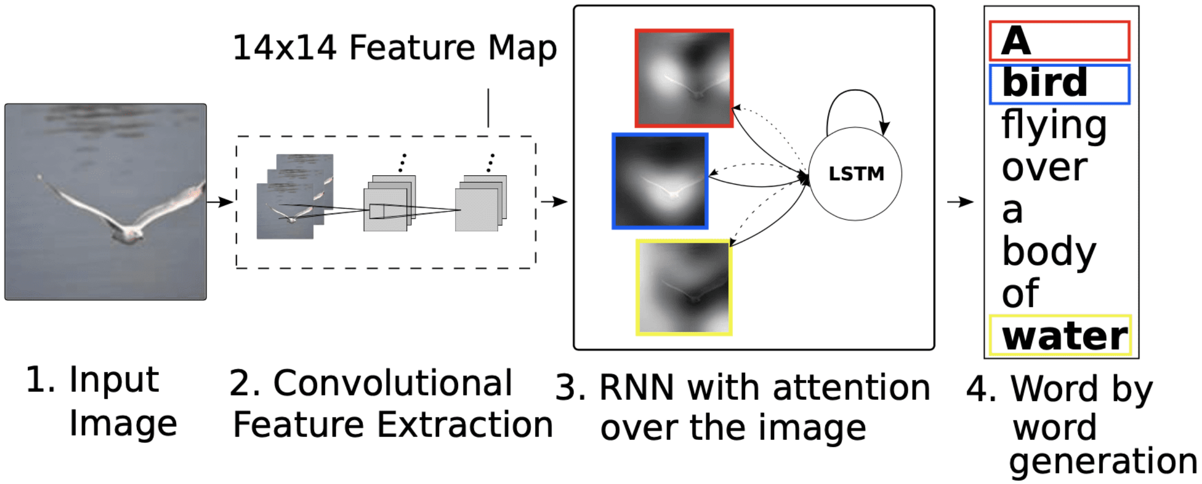
n: # of neurons in the net

LSTM vs Hopfield Networks: Key Differences

Aspect	LSTM	Hopfield Network
Primary Purpose	**Sequence modeling**, time series prediction (Temporal)	**Associative memory**, pattern completion (Spatial)
Memory Type	Gated memory cells	Energy-based attractor states
Training	Backpropagation through time	Hebbian learning (usually)
Time Dynamics	Sequential processing	Converges to equilibrium
Output	Next prediction in sequence	Retrieved memory pattern
Capacity	Virtually unlimited (state-based)	Limited (≈ 0.15N patterns for N neurons)

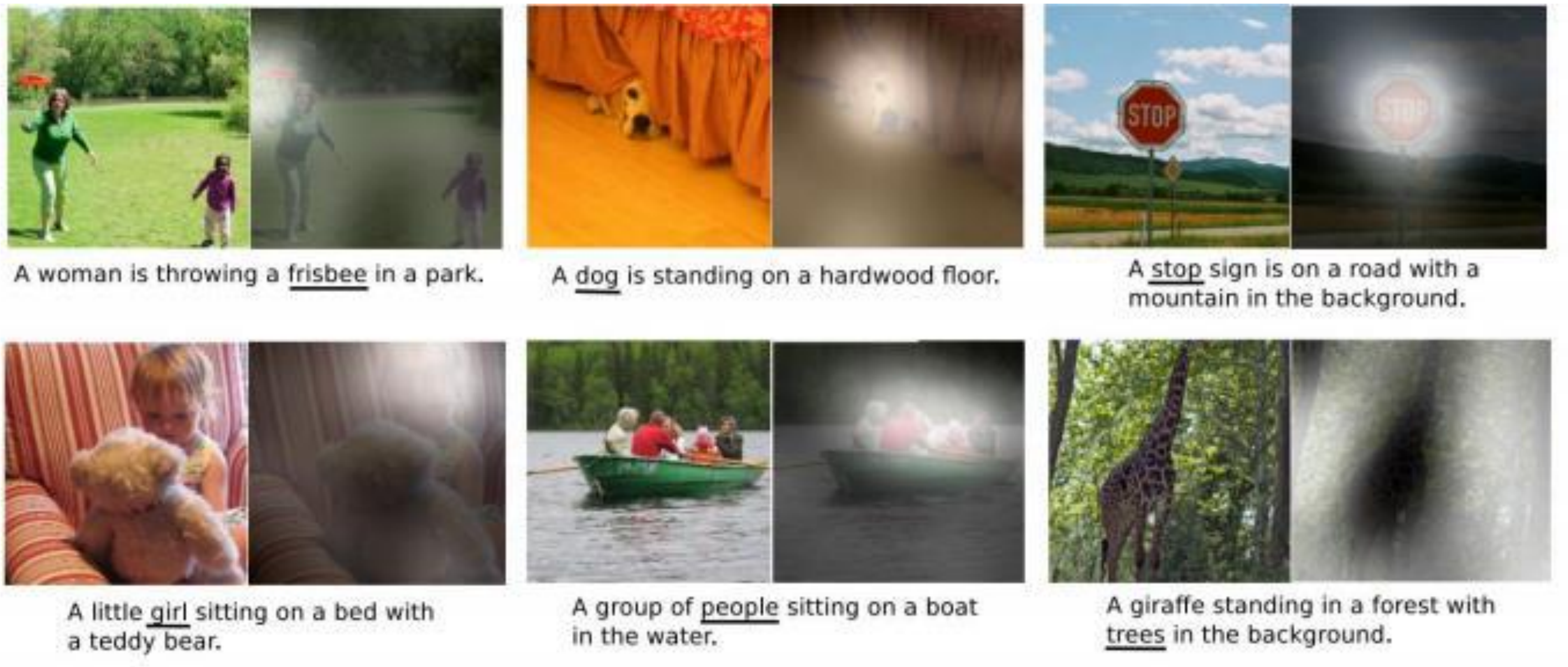
Attention Mechanism

In psychology, attention is the cognitive process of selectively concentrating on one or a few things while ignoring others.


14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

Attention mechanisms in Transformers are modern, differentiable implementations of continuous Hopfield networks that perform associative memory retrieval through softmax-weighted pattern recall.

Attention Mechanism Examples



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

The **Attention Mechanism** is a revolutionary concept that allows a neural network to **dynamically focus on the most relevant parts of its input** when producing an output, much like how human attention works.

Questions