CPEN 400D Guest Lecture: Recurrent Neural Networks

Xiaoxiao Li, Ph.D.

xiaoxiao.li@ece.ubc.ca

Department of Electrical and Computer Engineering

University of British Columbia

Outline

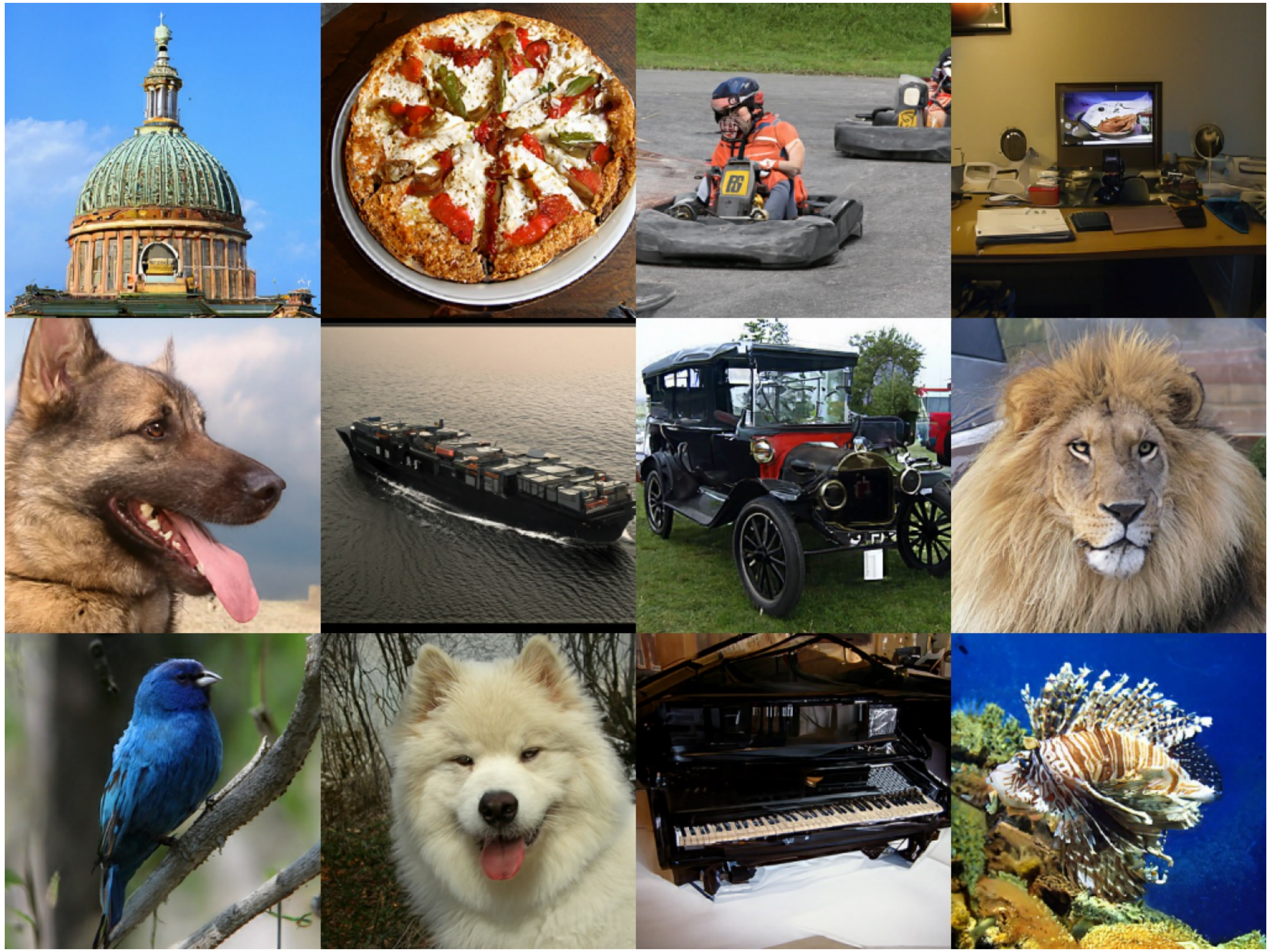
- Part 1. Background
- Part 2. RNN basis
- Part 3. Long Short-Term Memory Networks

Outline

- Part 1. Background
- Part 2. RNN basis
- Part 3. Long Short-Term Memory Networks

Why should you listen to this lecture?

Image Data



https://ai.googleblog.com/2021/07/high-fidelity-image-generation-using.html

Convolutional Neural Networks (CNN)

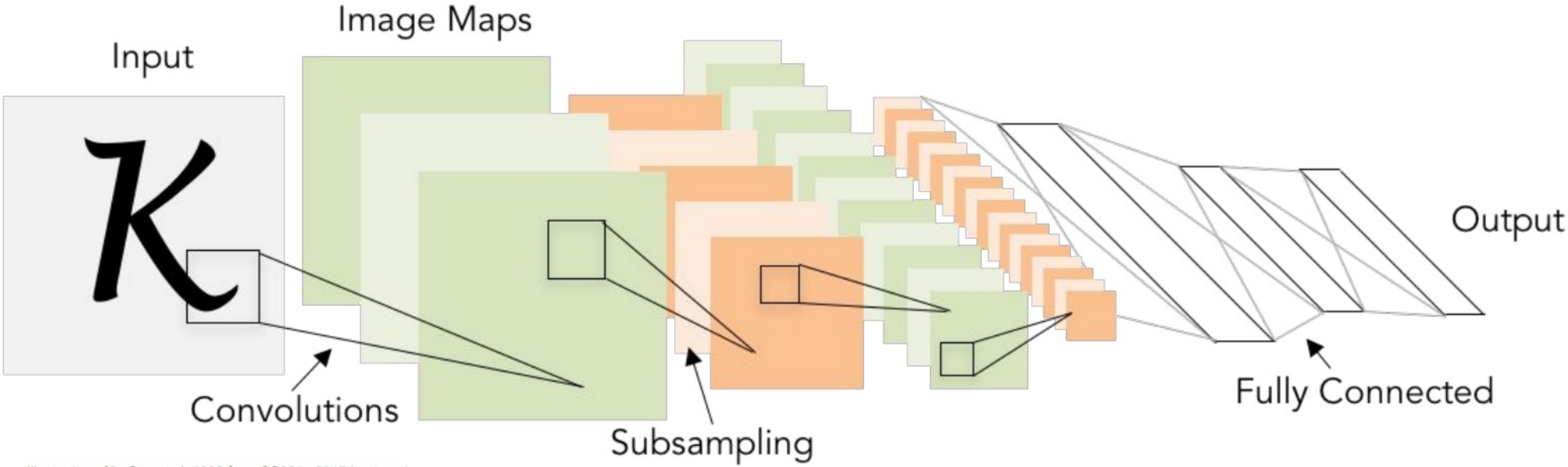
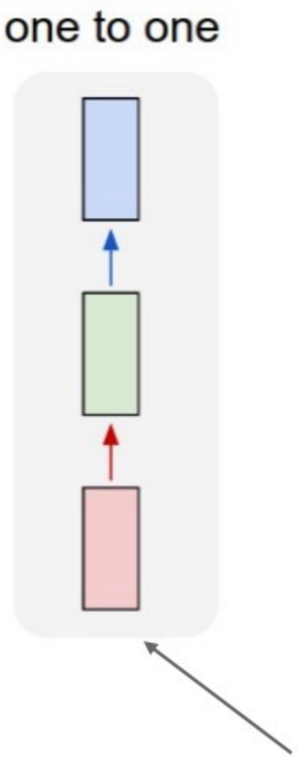


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

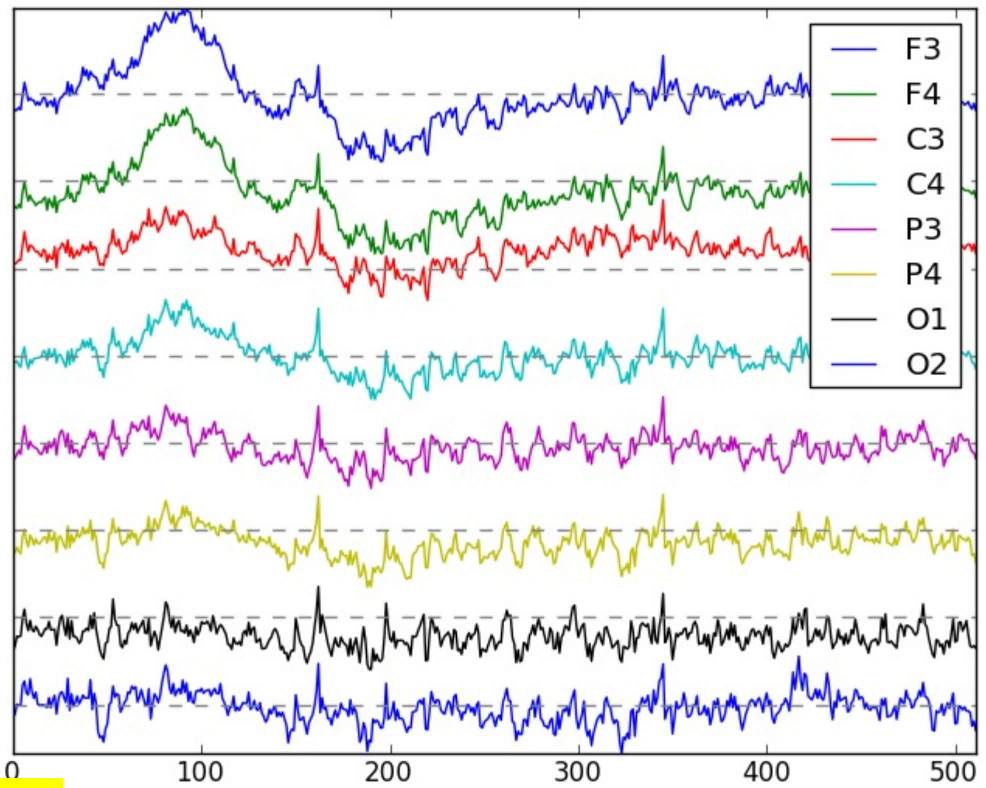
CNN inputs and outputs



e.g. **Image classification**
Image -> Label

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
  - Weather prediction
  - EEG analysis

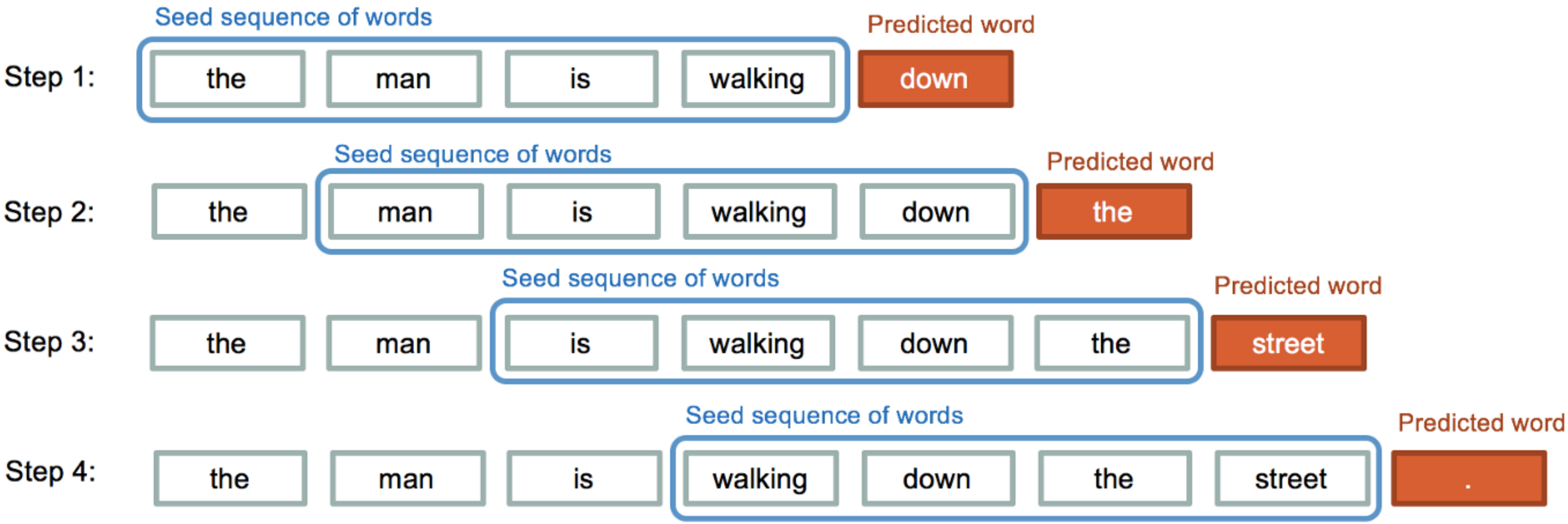


- Simple solution – FCN or CNN
  - Fixed input/output

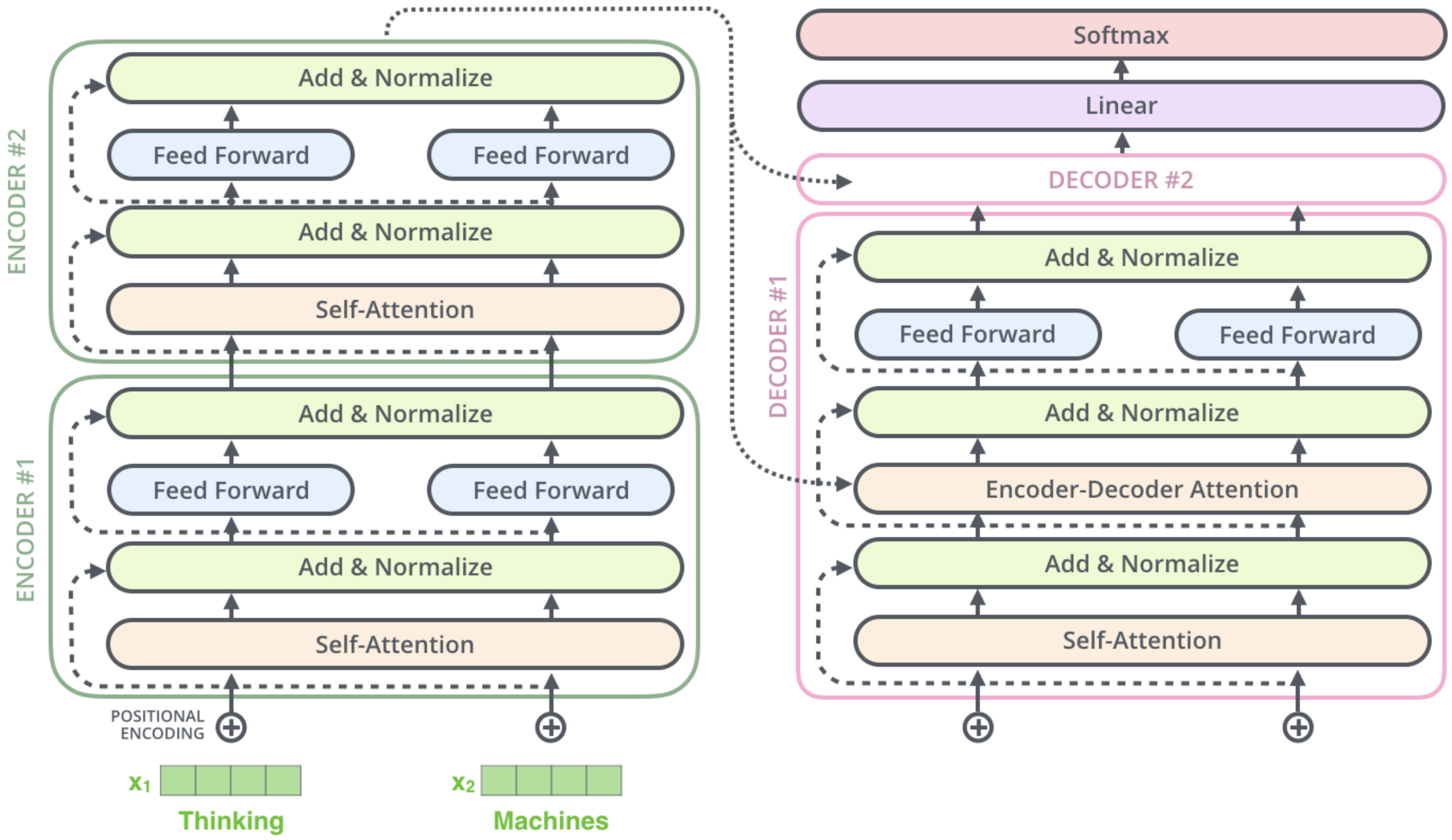
Sequential data (like sentence) has various length

Sequential data with various length

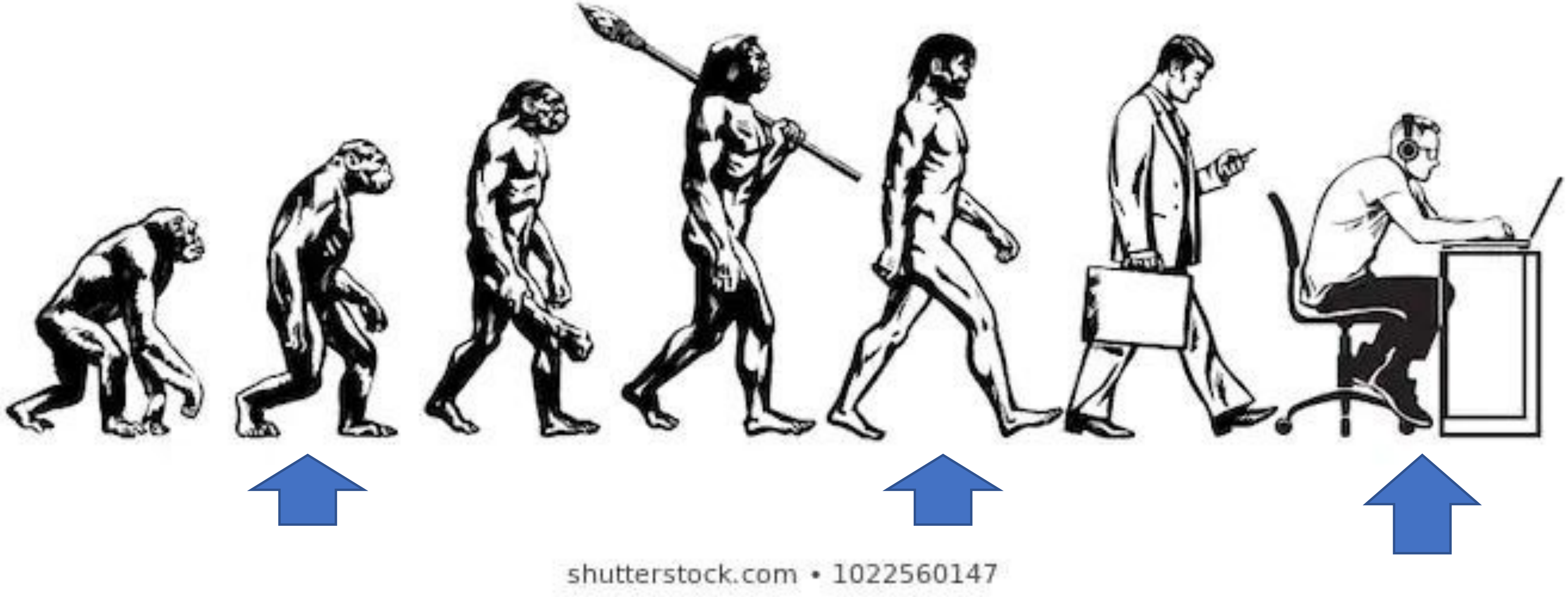
- How to take a variable length sequence as input?
- How to predict a variable length sequence as output?



Transformer



Evolution

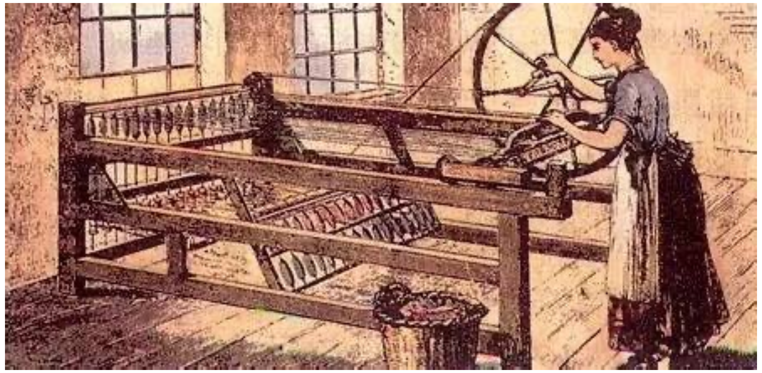


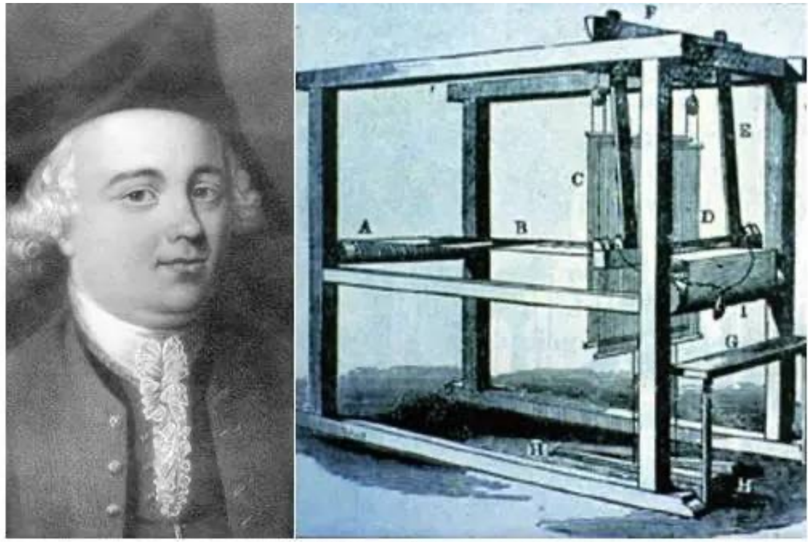
This lecture:
Recurrent Neural Network (RNN)

Your last lecture:
Transformer

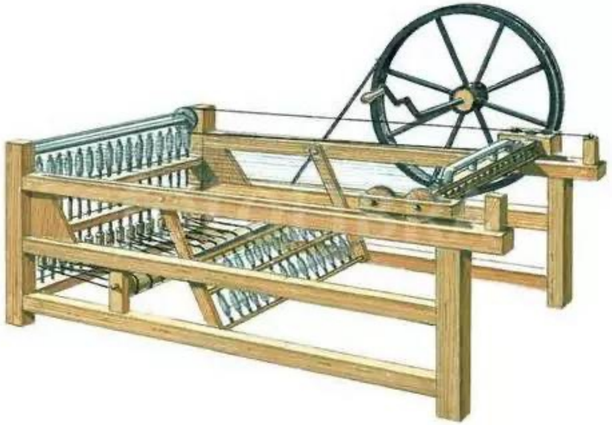
This week:
GTP 4

Industrial Revolution – production of fabrics





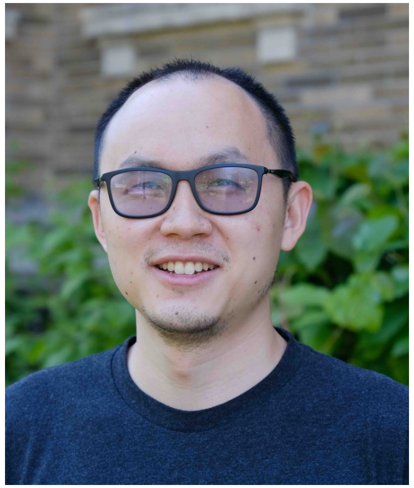
In 1773, John Kaye invented the flying shuttle for weaving cloth



In 1768, Hargreaves received a patent for the invention of the Jenny spinning machine.

Engels said: the first invention that turned the situation of British workers upside down.

Why do we still need to learn RNN?



It is on our syllabus.

- RNNs can still be useful in certain applications where the input data is inherently sequential or when computational resources are limited.
- The choice between RNNs and Transformers largely depends on the specific problem, dataset, and computational constraints.

Advantages of RNN and TF

Transformer Advantages:

- Long-range dependencies: self-attention captures relationships between distant tokens.
- Parallelization: processes input tokens simultaneously, faster training and inference.
- Scalability: state-of-the-art results in various NLP tasks, e.g., BERT, GPT, T5.

RNN Advantages:

- Sequential processing: natural fit for time series, speech recognition, language modeling.
- Parameter efficiency: shared weights across time steps.

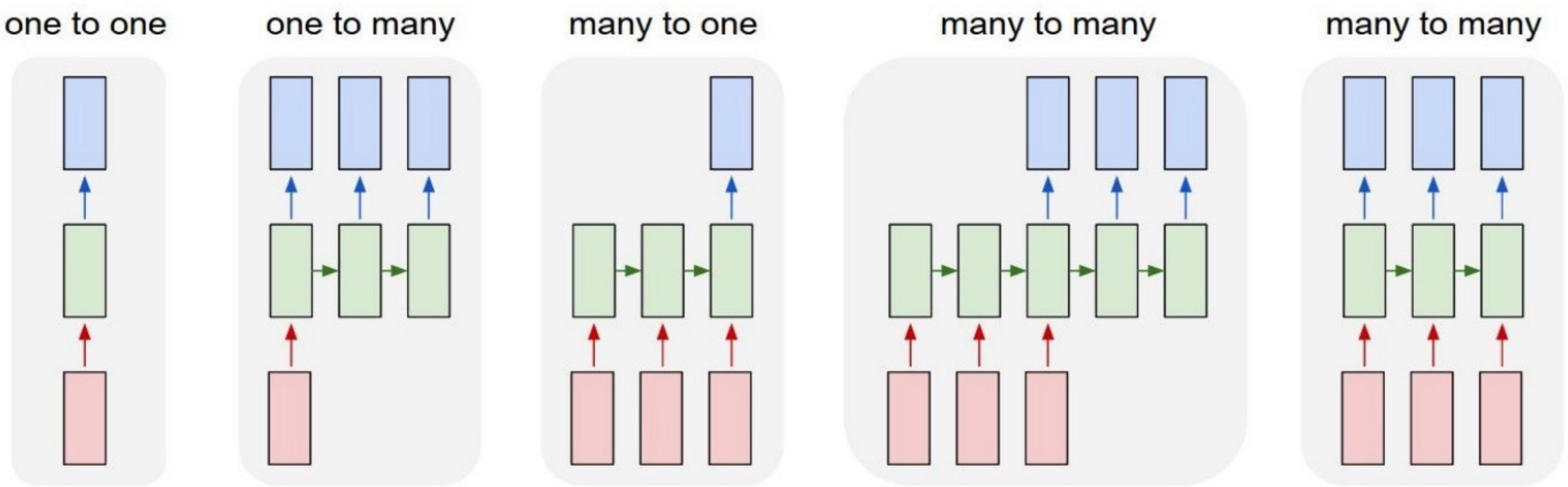
While Transformers have generally outperformed RNNs in many NLP tasks

When shall we use RNN?

- Handling short sequences: RNNs can be more suitable for short sequences where long-range dependencies are less critical.
- Limited resources: RNNs have fewer parameters, making them more computationally efficient and easier to train on limited hardware.
- Real-time processing: RNNs are better suited for real-time or online processing, where the input sequence is generated incrementally.
- Inherently sequential data: Some tasks, like speech recognition or time series prediction, naturally benefit from RNNs' sequential processing.

Example Inputs of Outputs on Sequential Data Analysis

Inputs & Outputs of RNN



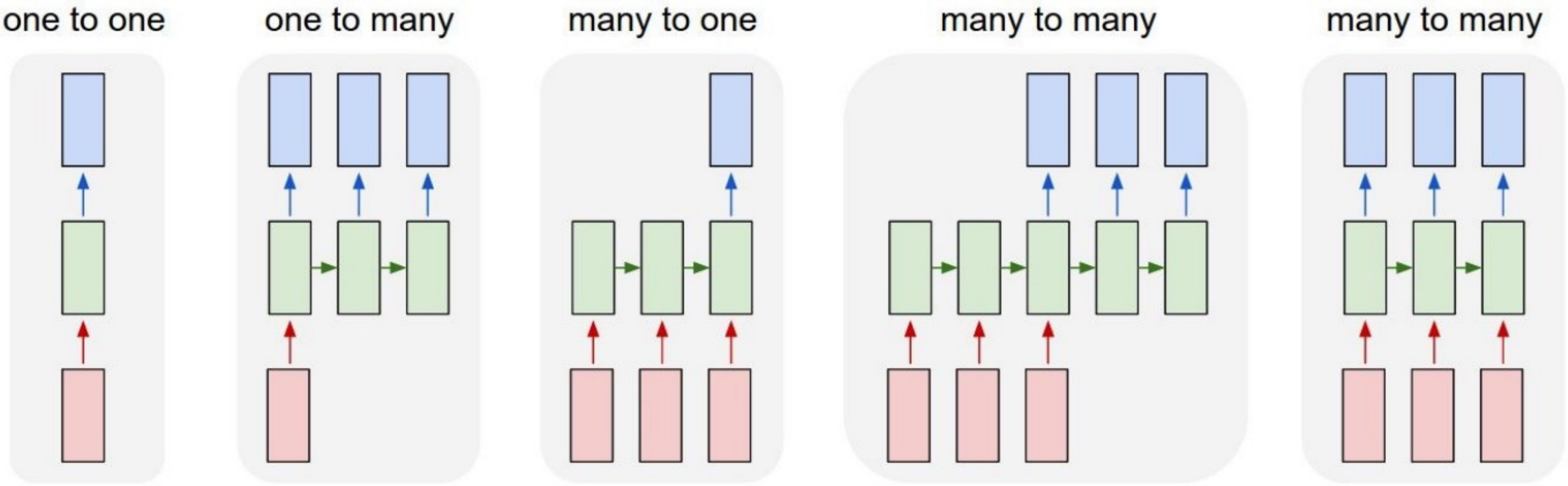
CNN

e.g. **Image Captioning**
image -> sequence of words

Source: Stanford CS231n


Captioning Model
A cat sitting on the road
ProjectPro

Inputs & Outputs of RNN



CNN

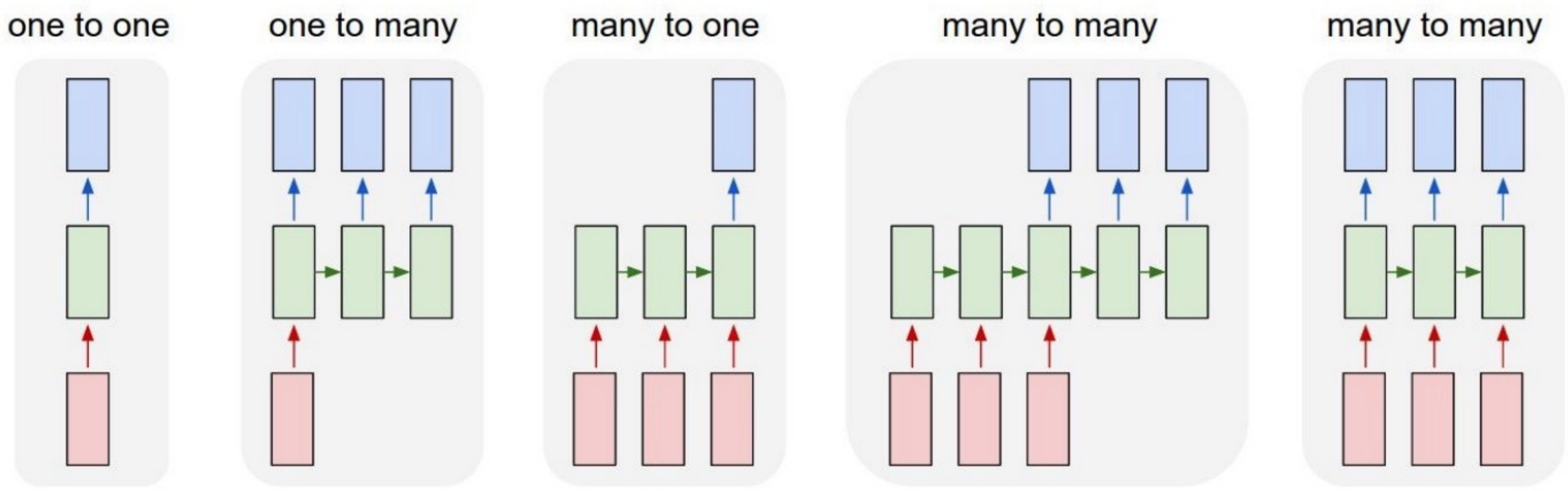
e.g. **Sentiment Classification**
sequence of words -> sentiment

Source: Stanford CS231n

"I love this movie.
I've seen it many times
and it's still awesome." → 

"This movie is bad.
I don't like it it all.
It's terrible." → 

Inputs & Outputs of RNN



CNN

e.g. **Machine Translation**
seq of words -> seq of words

Source: Stanford CS231n

Chinese - detected

English

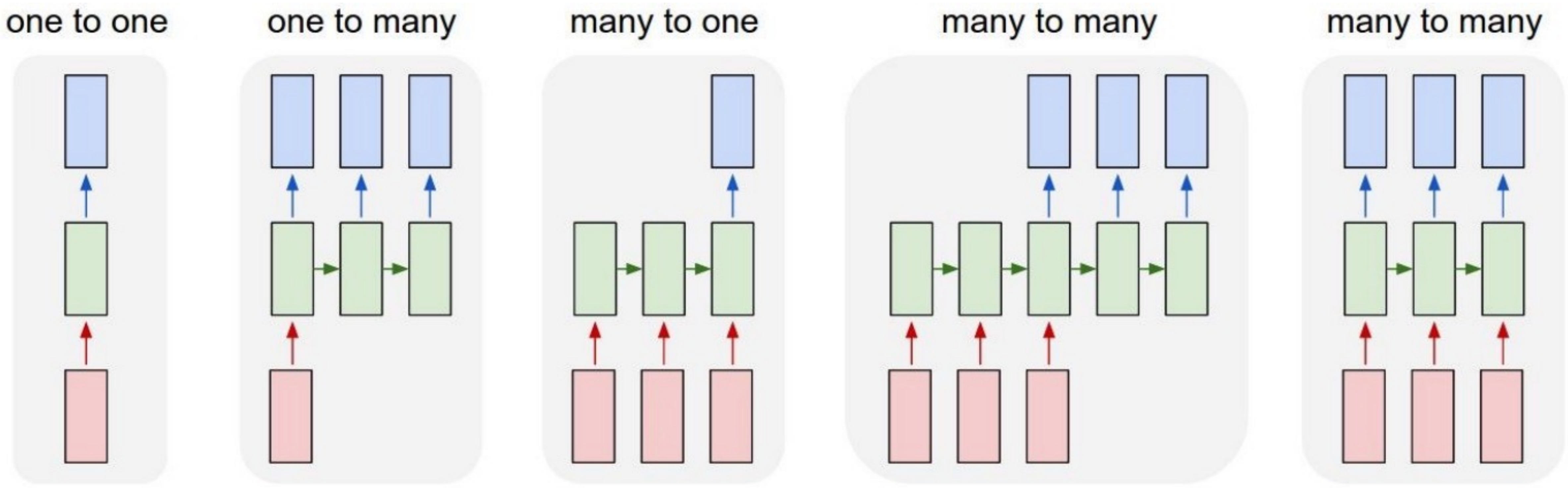
明月松间照，清泉石上流。

Míngyuè sōng jiān zhào, qīngquán shí shàngliú.

The bright moon shines among the pines, and the clear spring stones flow upwards.

Open in Google Translate • Feedback

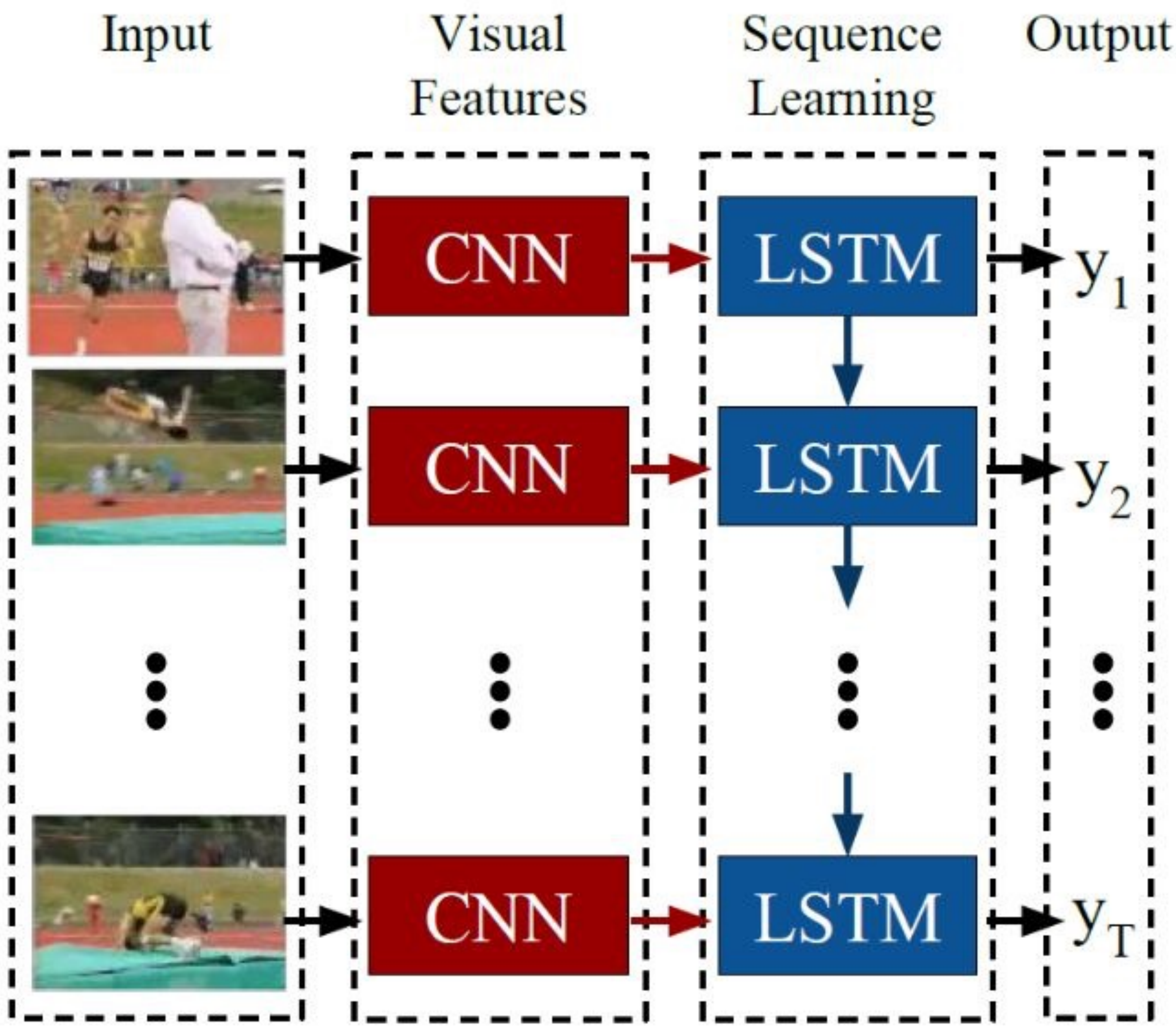
Inputs & Outputs of RNN



CNN

e.g. **Video classification on frame level**

Source: Stanford CS231n

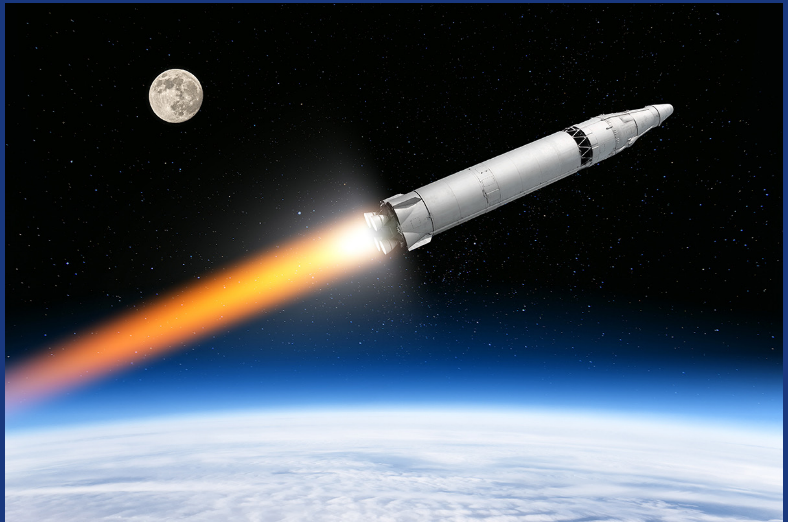


https://imerit.net/blog/using-neural-networks-for-video-classification-blog-all-pbm/

Outline

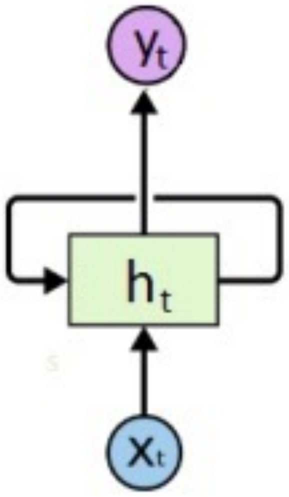
- Part 1. Background
- Part 2. RNN basis
- Part 3. Long Short-Term Memory Networks

RNN Formulation



RNN Cell Unit

- Feedforward network: a neural network with no loops
- RNNs store information about previous data in the “state”
- Recurrently feeds output of activation function to itself



Formula of RNN

Recurrent neural networks (RNNs) are networks with loops, allowing information to persist [Rumelhart et al., 1986].

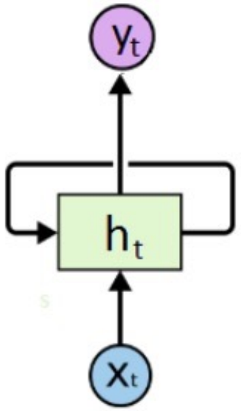
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W



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

RNN hidden state update

Recurrent neural networks (RNNs) are networks with loops, allowing information to persist [Rumelhart et al., 1986].

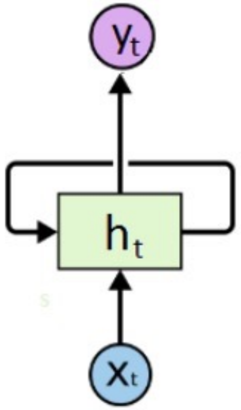
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step



State variable

- Have memory that keeps track of information observed so far
- Maps from the entire history of previous inputs to each output

RNN output generation

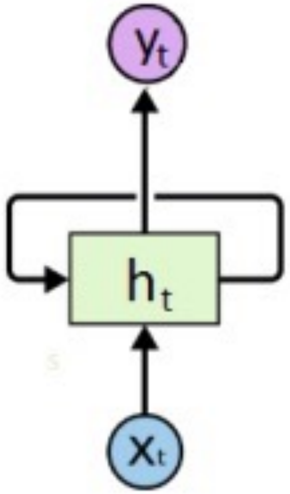
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

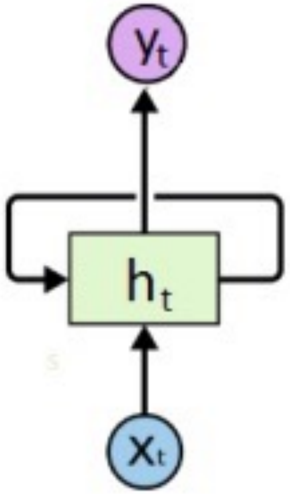
output

new state

another function with parameters W_o



Formula of RNN (Vanilla)



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

(also bias term)

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

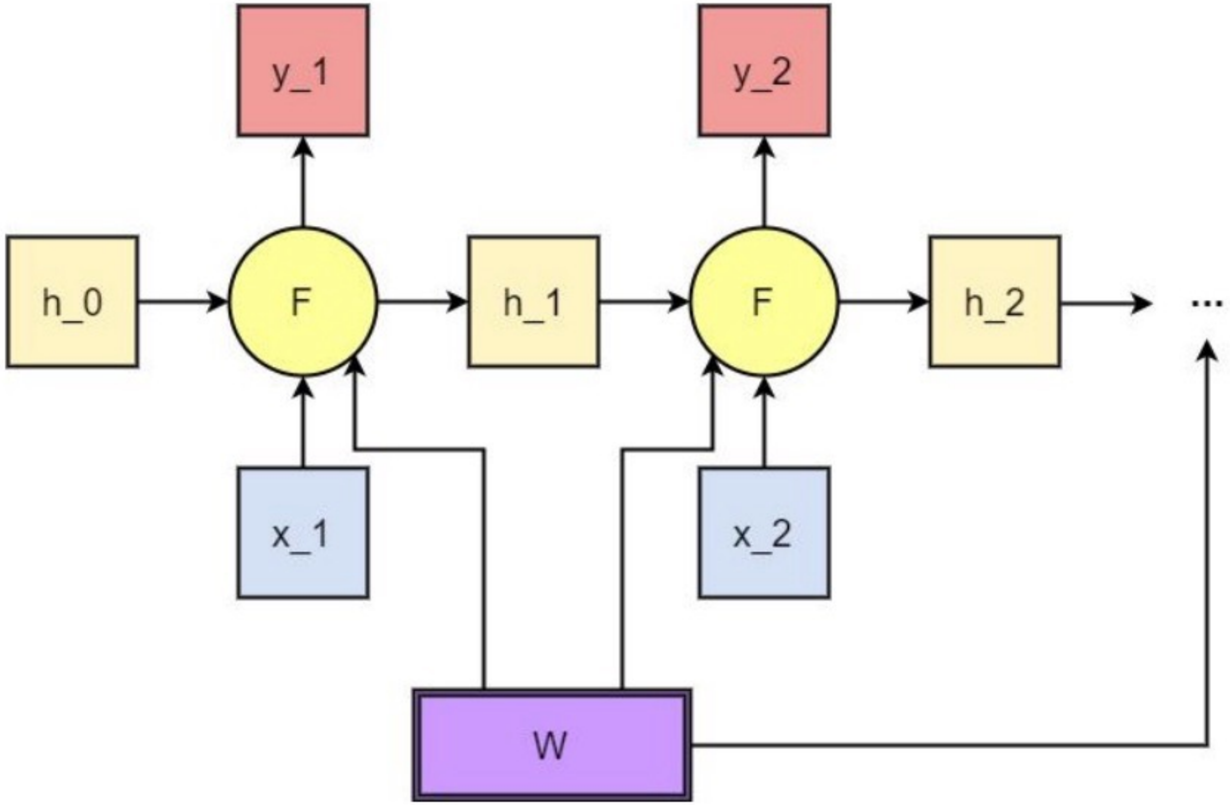
$$y_t = W_{hy}h_t$$

- x_t is the input at time t.
- h_t is the hidden state (memory) at time t.
- y_t is the output at time t.
- W_{hh}, W_{hx}, W_{hy} are distinct weights.
  - weights are the same at all time steps.

RNN Computational Graph

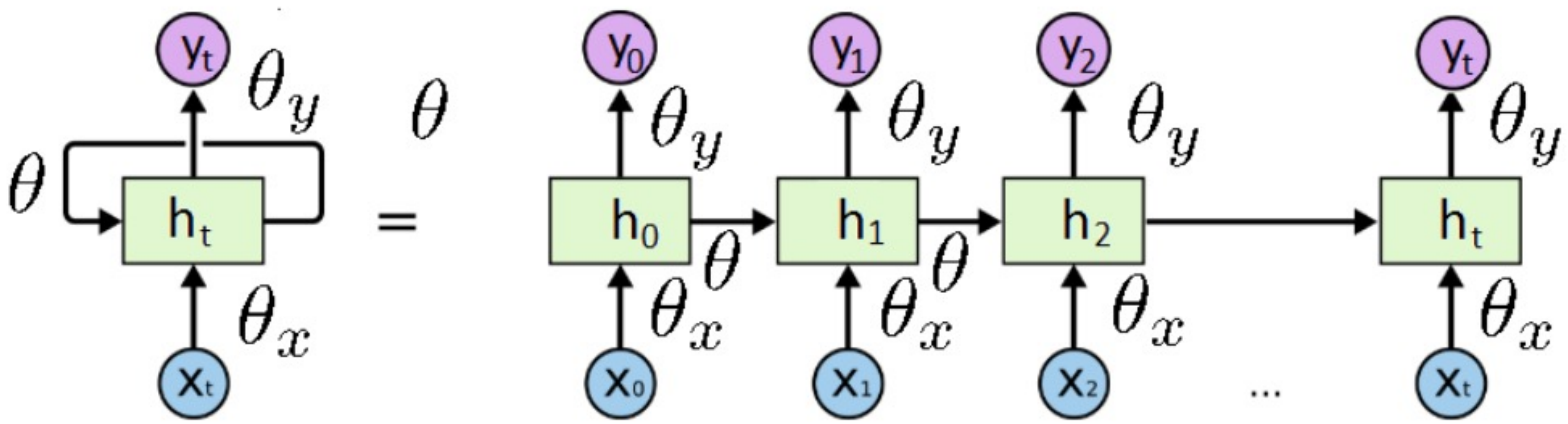
with shared (tied) weights

$$(h_1, y_1) = F(h_0, x_1, \boxed{W})$$
$$(h_2, y_2) = F(h_1, x_2, \boxed{W})$$



Parameter sharing

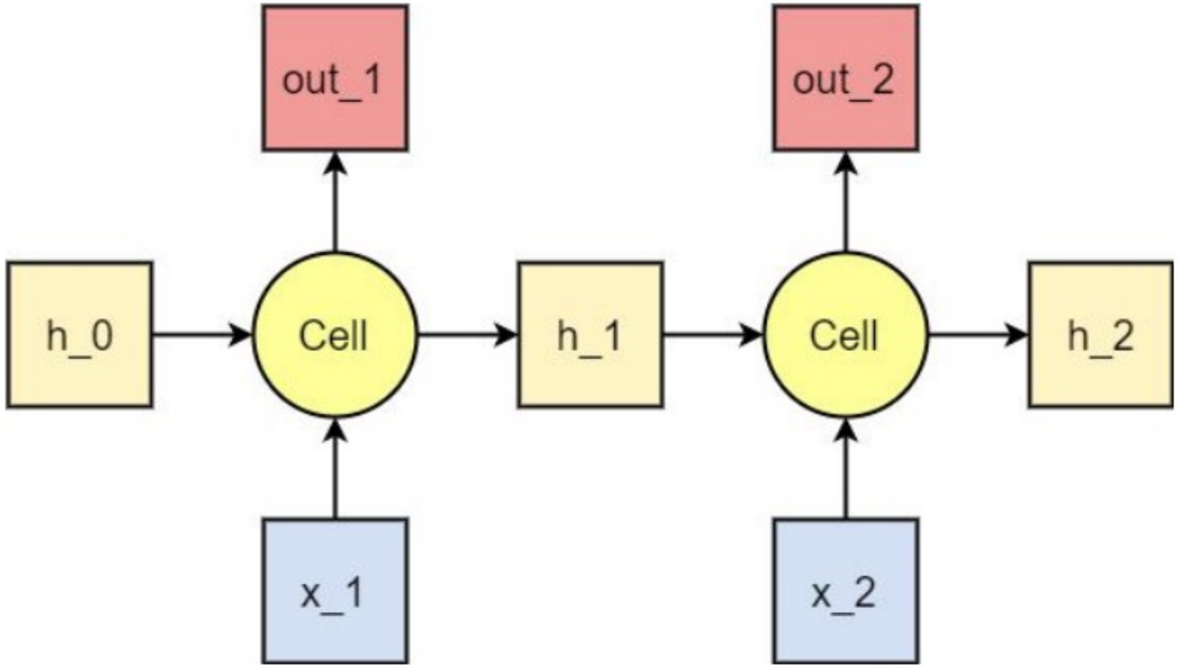
- RNNs can be thought of as multiple copies of the same network, each passing a message to a successor.



- The same function and the same set of parameters are used at every time step.

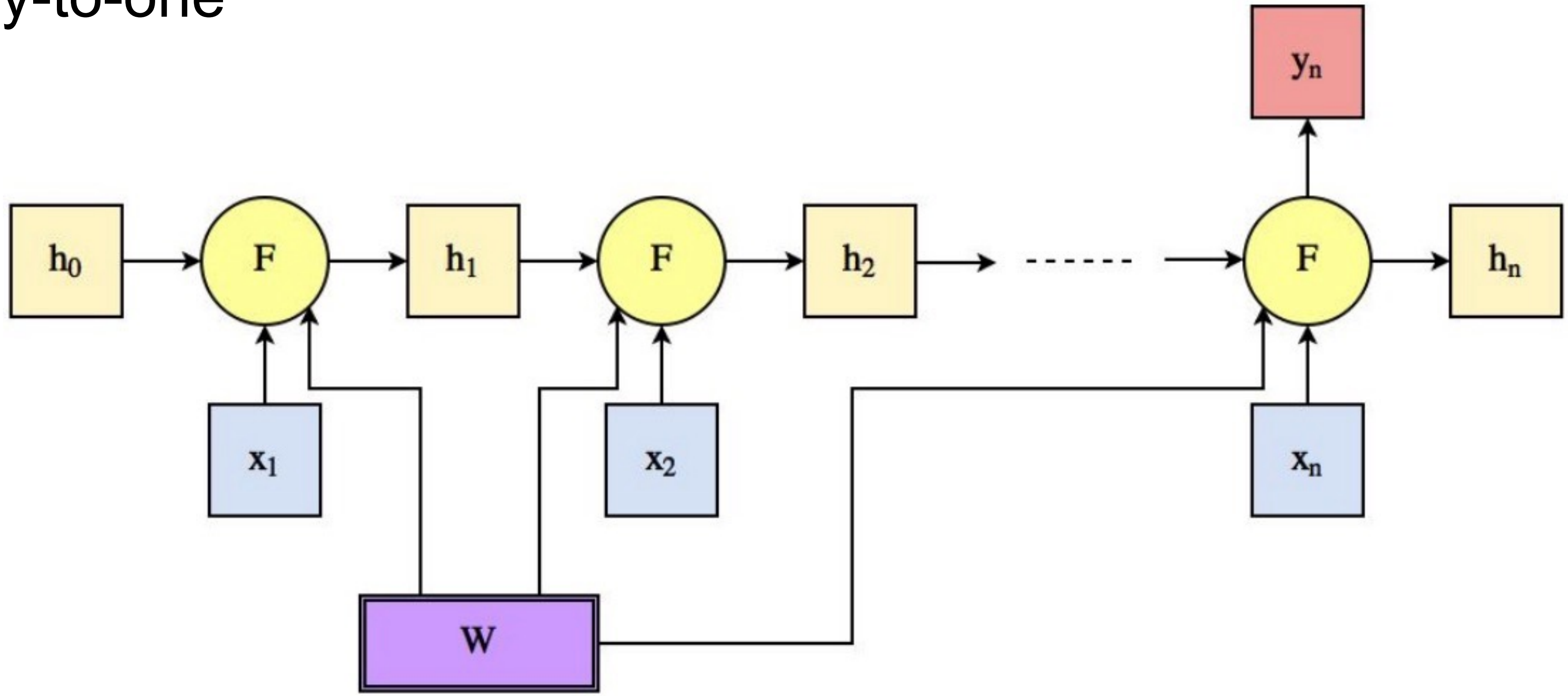
RNN Computational Graph

(x_1, x_2) comprises a length-2 sequence



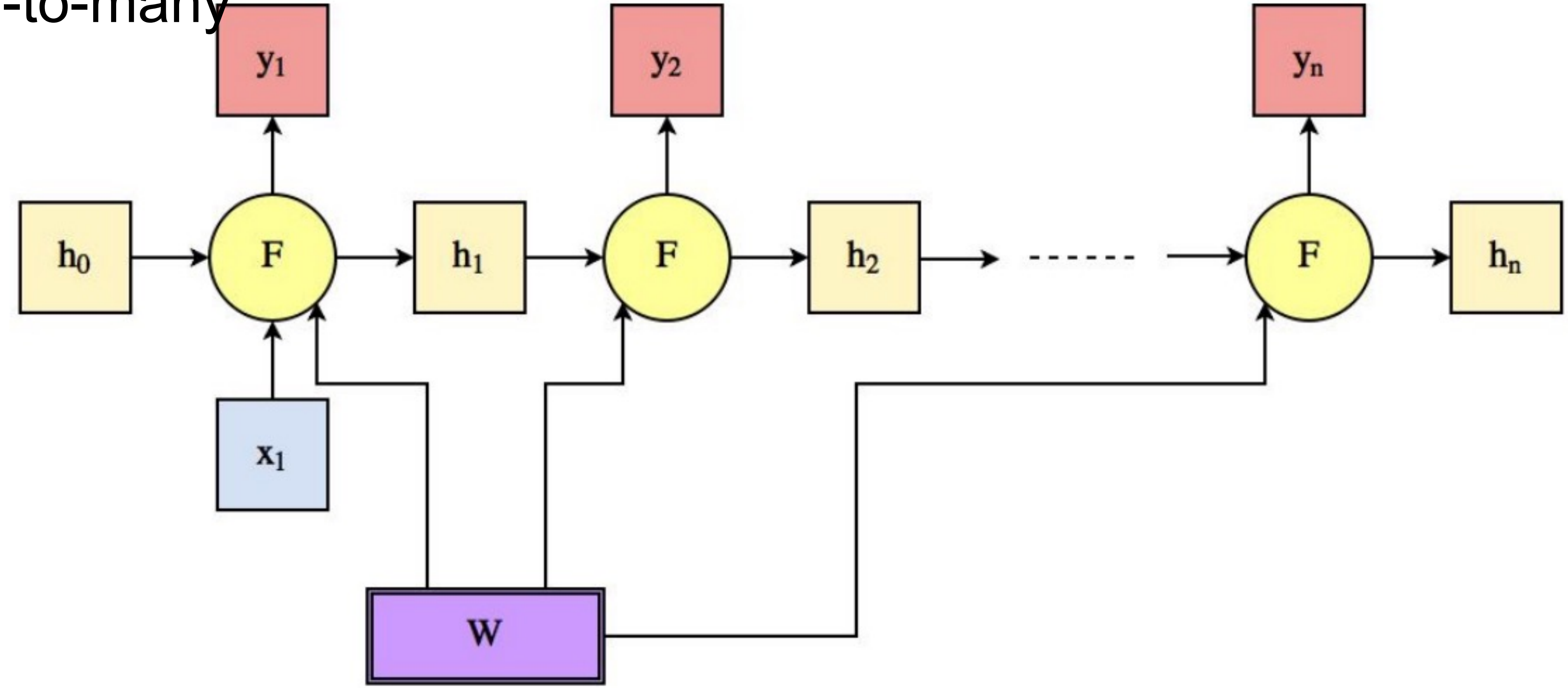
RNN Computational Graph

- Many-to-one



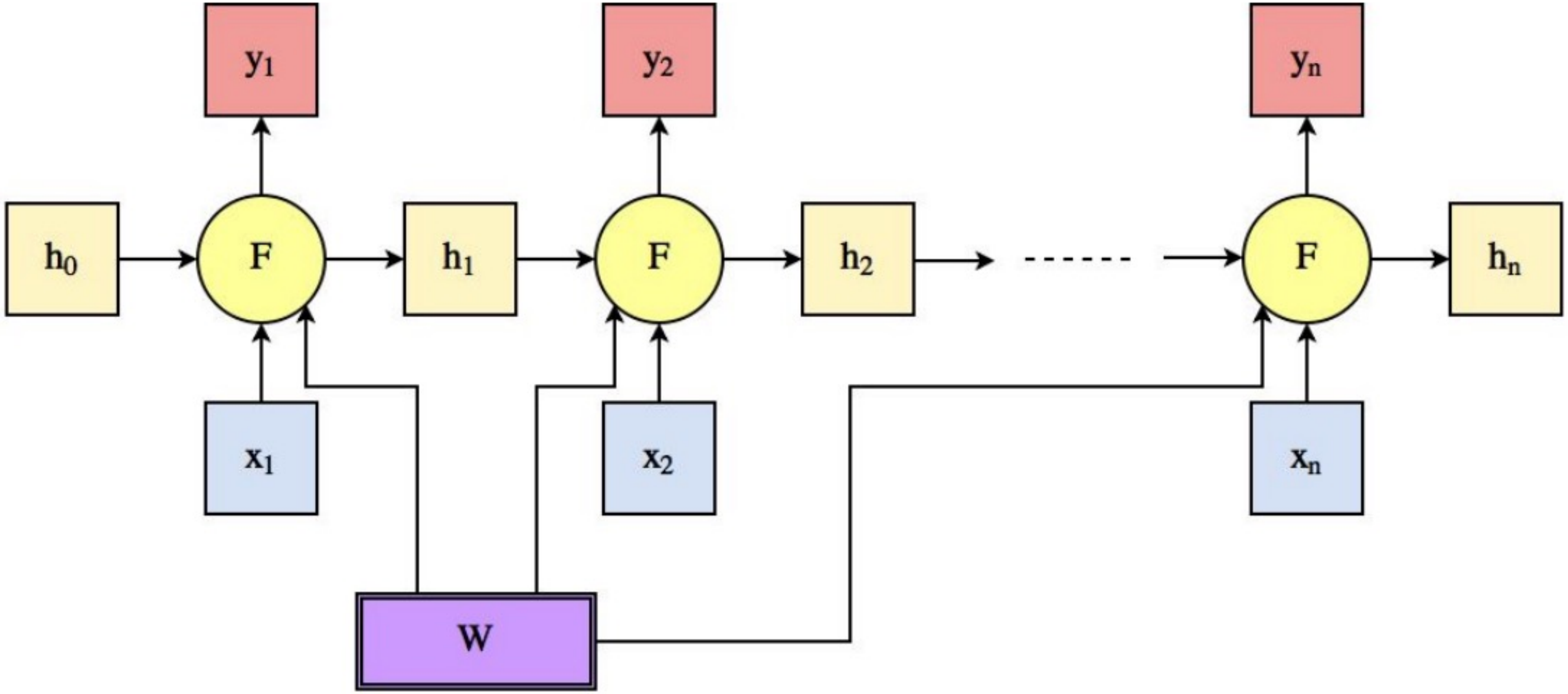
RNN Computational Graph

- One-to-many



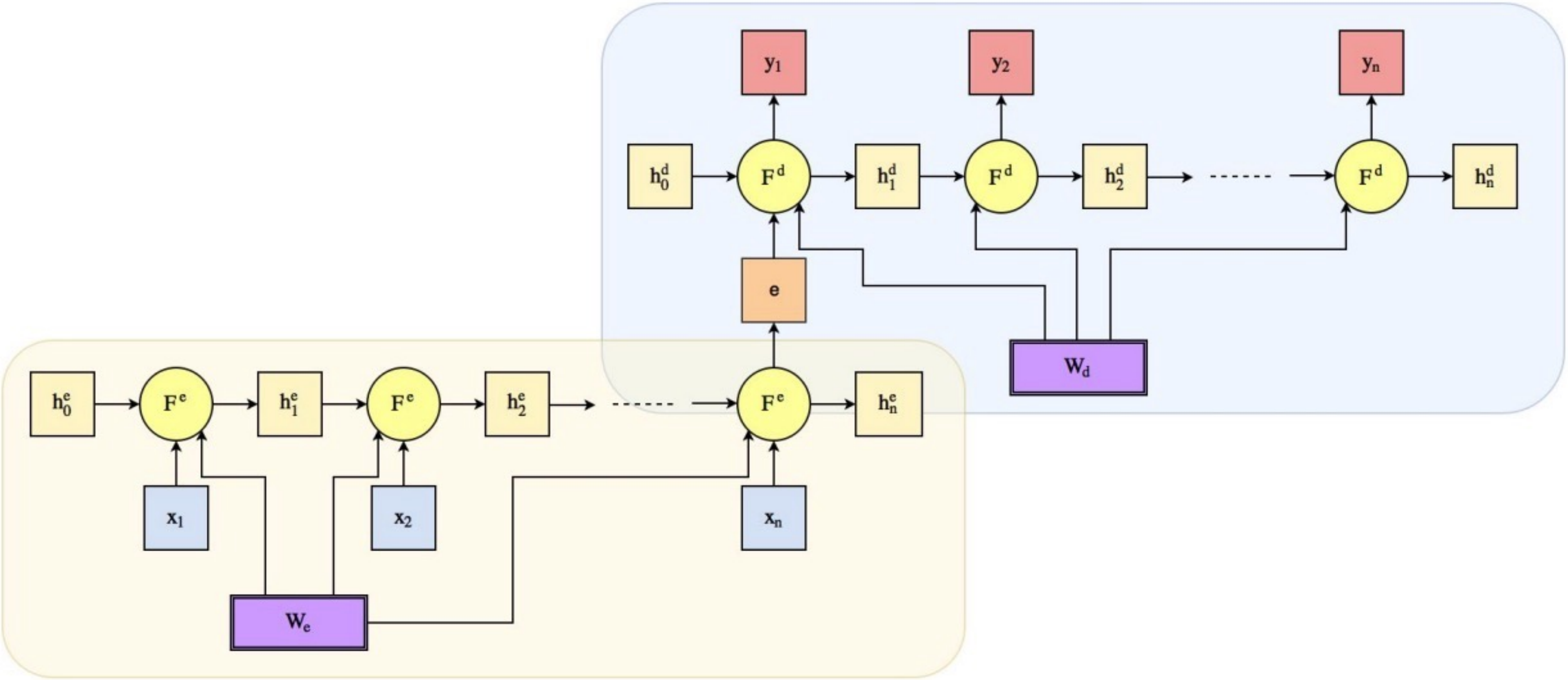
RNN Computational Graph

- Many-to-many



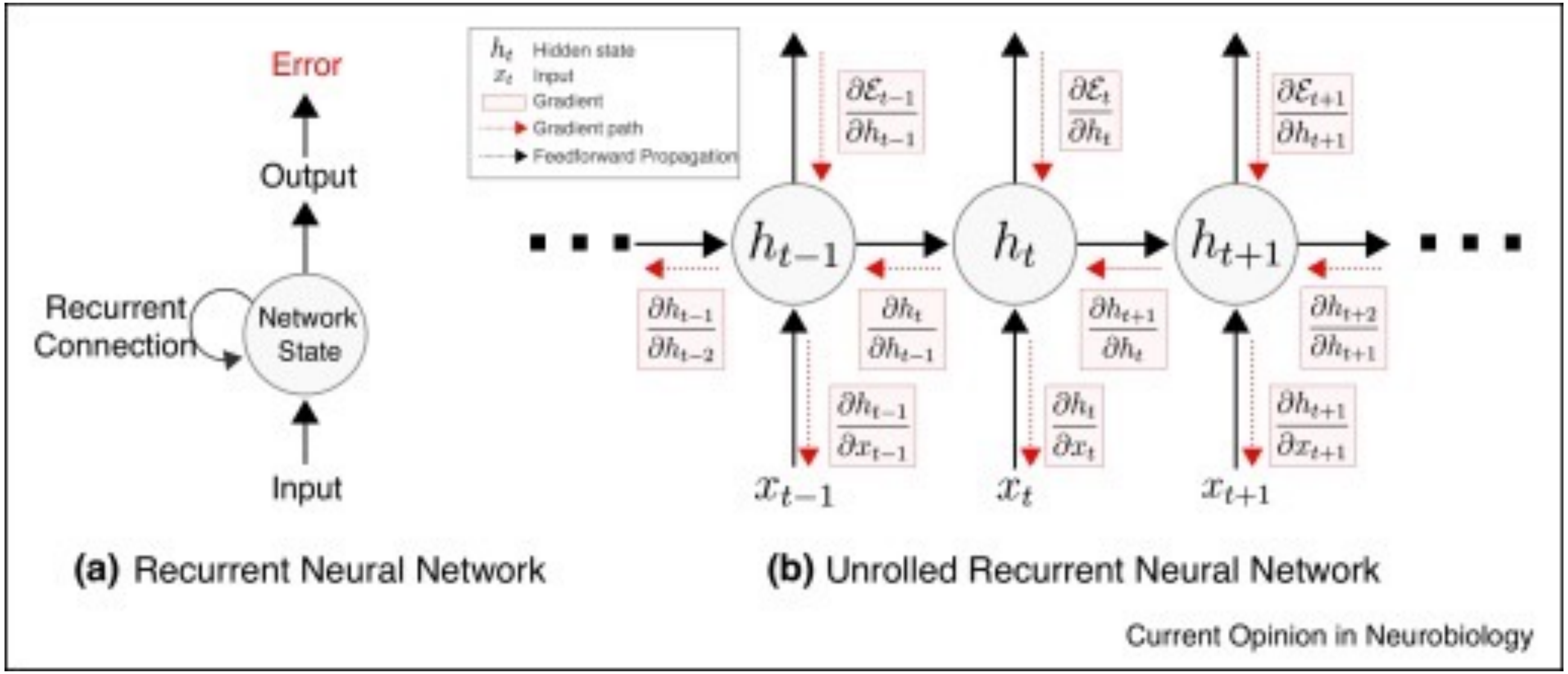
RNN Computational Graph

- Many-to-Many: Many-to-One + One-to-Many



Optimizing RNN

- Using the generalized back-propagation algorithm one can obtain the so-called **Back-Propagation Through Time** algorithm.

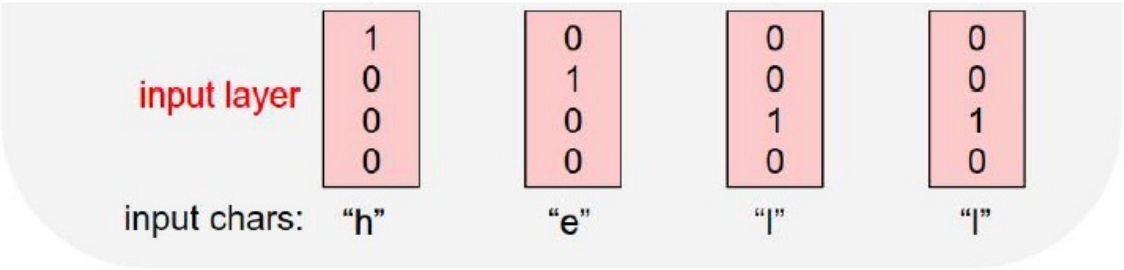


Examples

Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

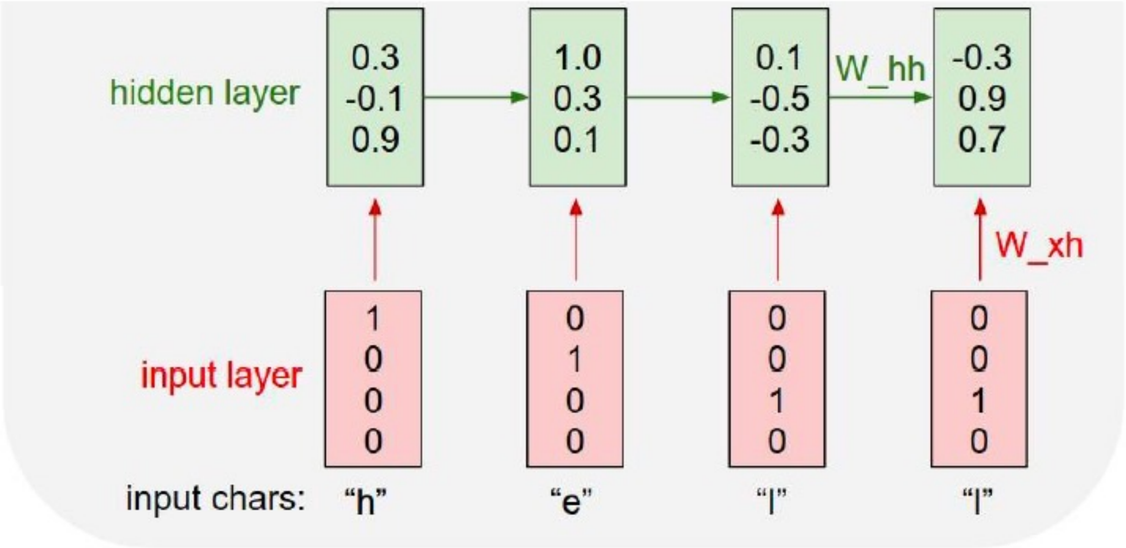


Character-level Language Model

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

Vocabulary:
[h,e,l,o]

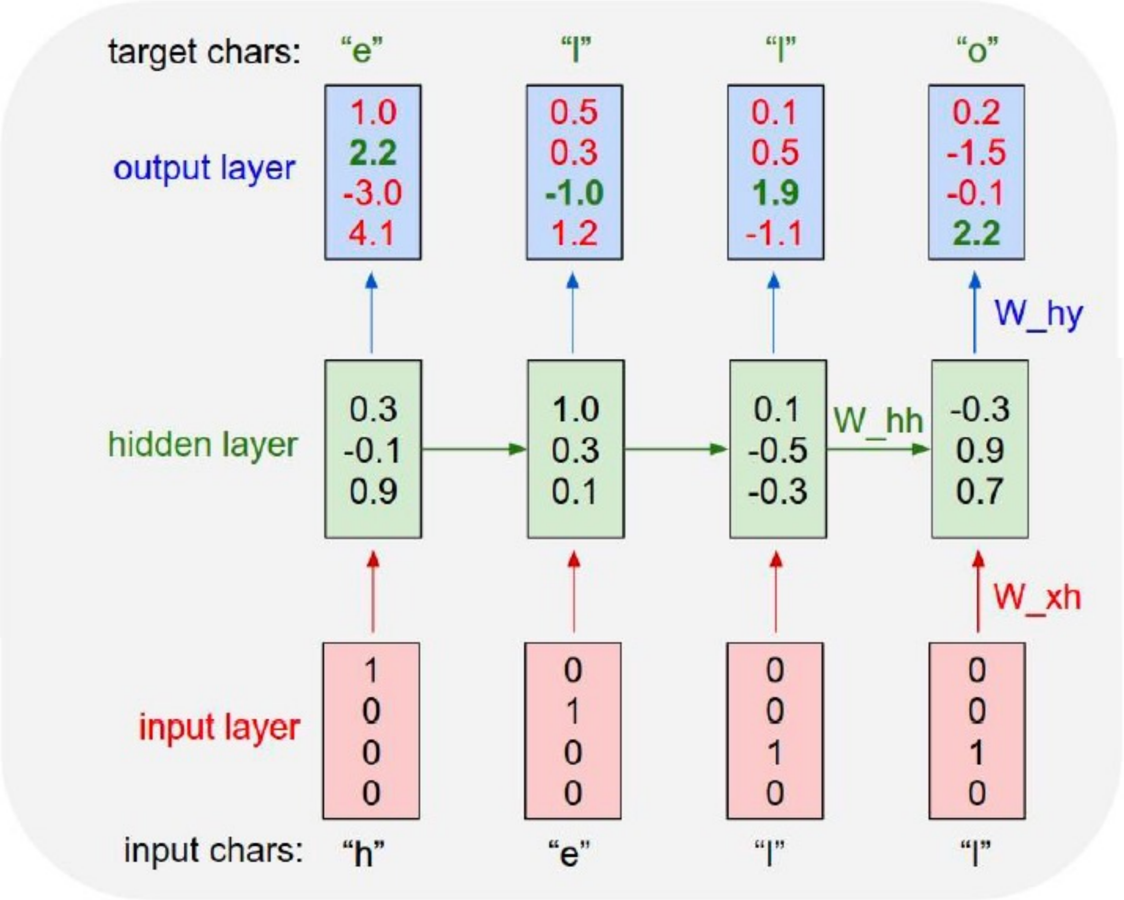
Example training sequence:
“hello”



Character-level Language Model

Vocabulary:
[h,e,l,o]

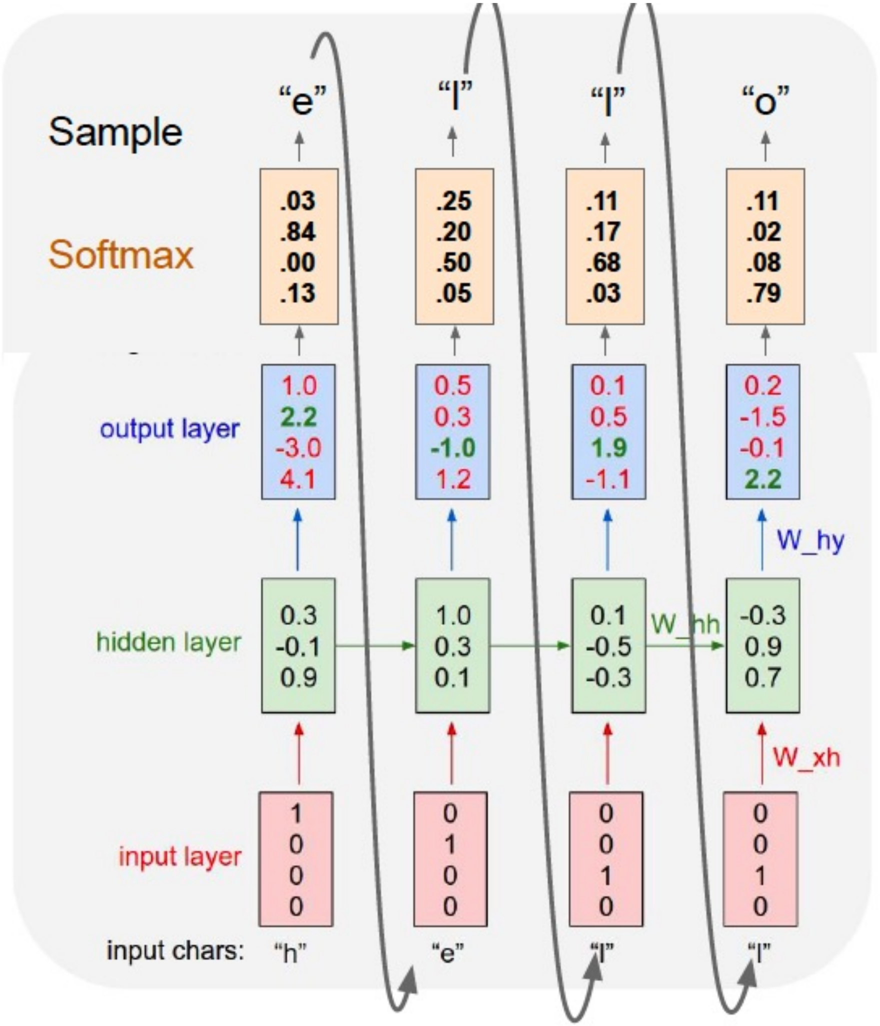
Example training sequence:
“hello”



Character-level Language Model

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



Outline

- Part 1. Background
- Part 2. RNN basis
- Part 3. Long Short-Term Memory Networks
- Part 4. Guest lecture – Self-supervised learning

The Problem of Long-term Dependencies

- In RNNs, during the gradient back propagation phase, the gradient signal can end up being multiplied many times.

$$\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \theta}$$

- If the gradients are large
  - Exploding gradients, learning diverges
  - Solution: clip the gradients to a certain max value.
- If the gradients are small
  - Vanishing gradients, learning very slow or stops
  - Solution: introducing memory via LSTM, GRU, etc.

The Problem of Long-term Dependencies

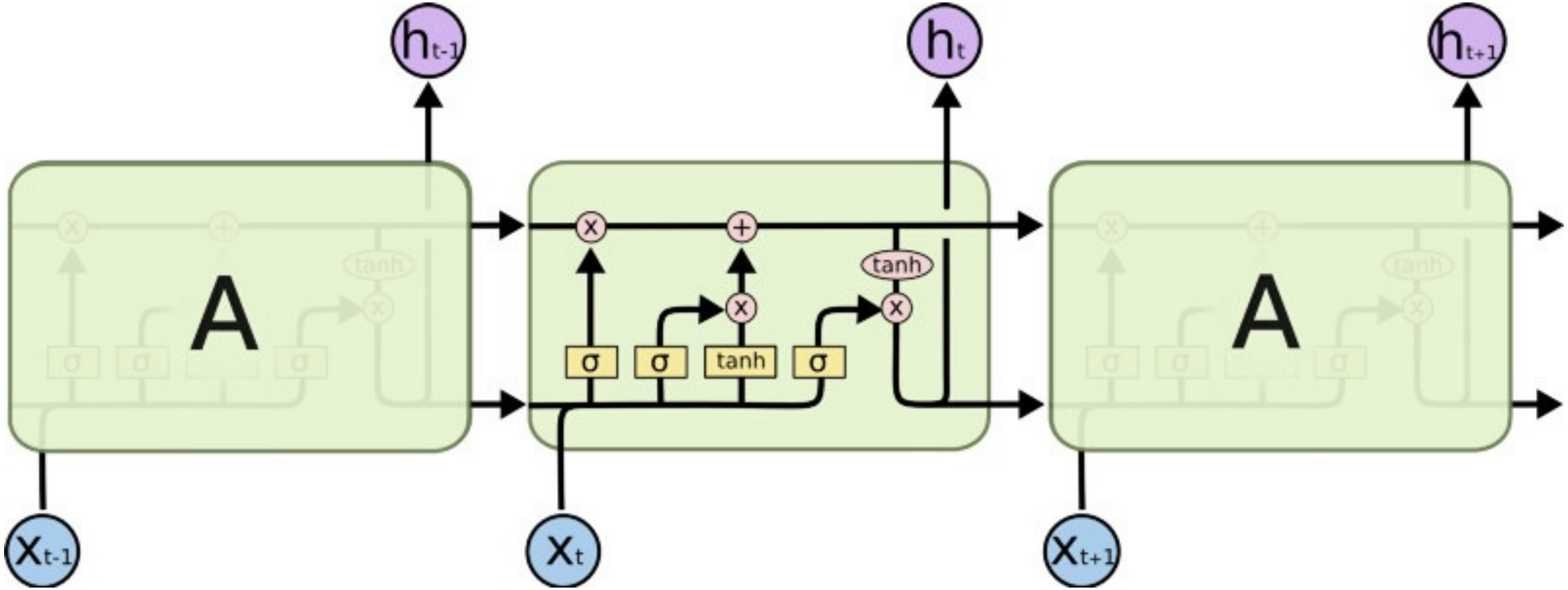
- In RNNs, during the gradient back propagation phase, the gradient signal can end up being multiplied many times.

$$\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \theta}$$

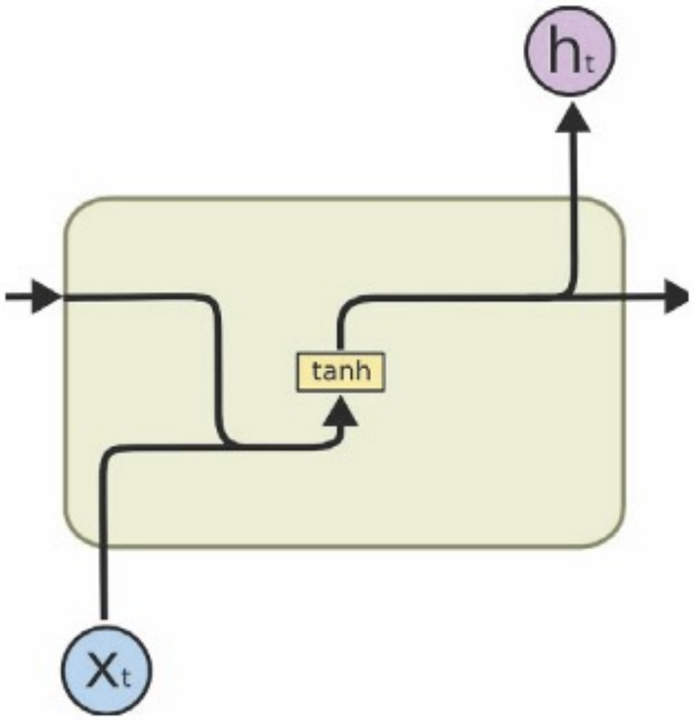
$$\mathbf{h}_t = \theta \phi(\mathbf{h}_{t-1}) + \theta_x \mathbf{x}_t$$

Long Short-Term Memory Networks

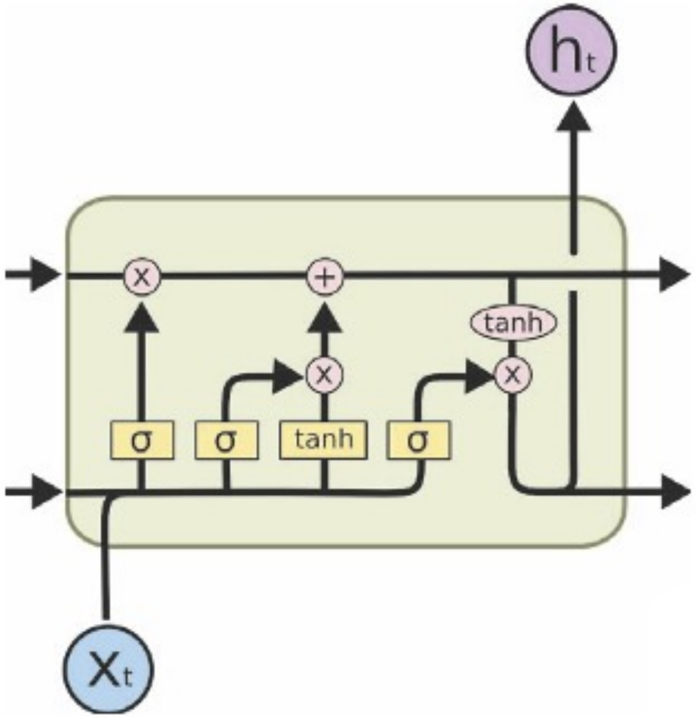
- Long Short-Term Memory (LSTM) networks are RNNs capable of learning long-term dependencies



Vanilla RNN vs LSTM



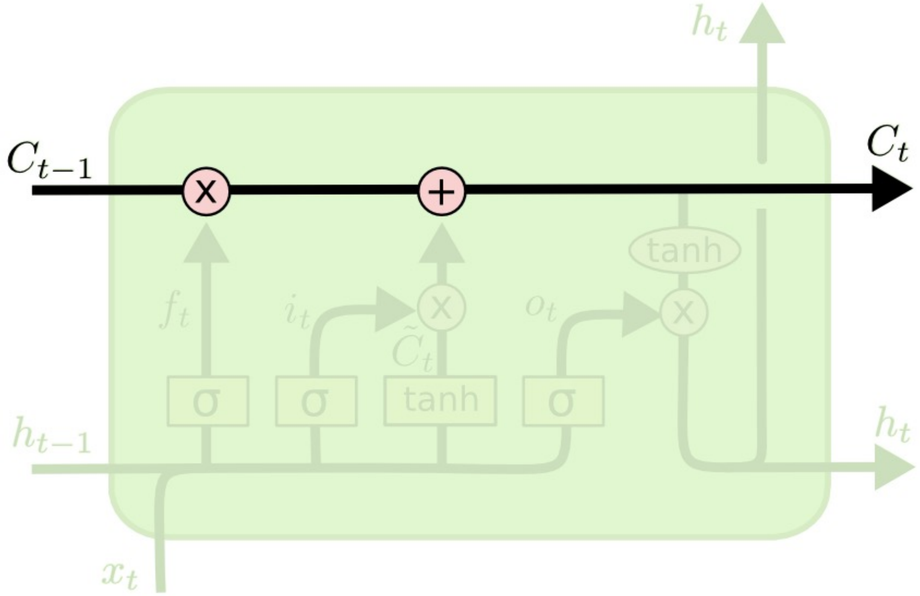
(a) RNN

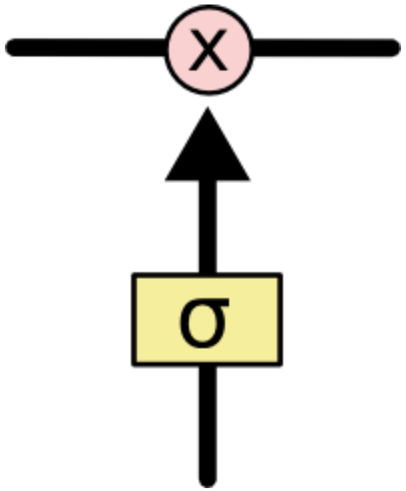


(b) LSTM

The Core Idea – Cell State

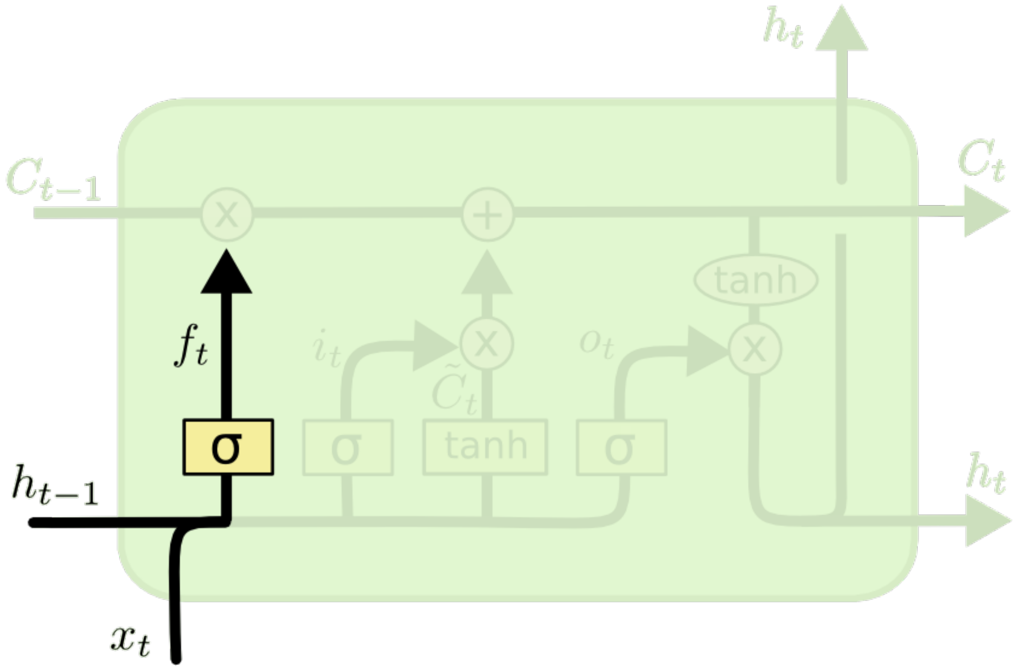
- The cell state is like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.





Step-by-Step LSTM Walk Through

- Forget gate layer:

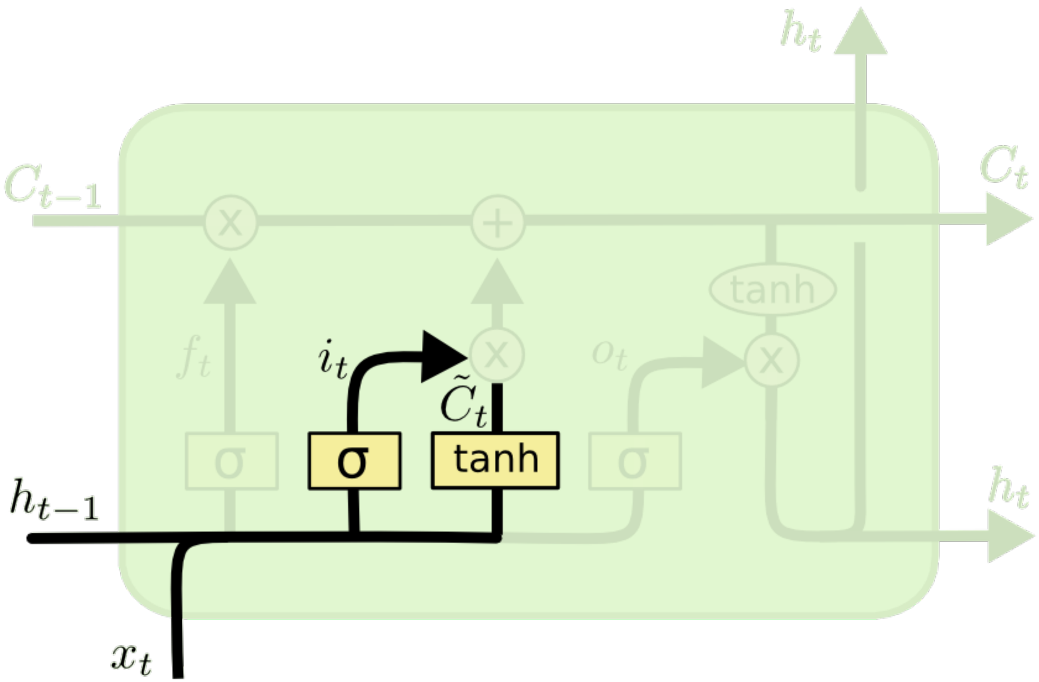


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Output: 0/1

Step-by-Step LSTM Walk Through

- Input gate layer + tanh layer:

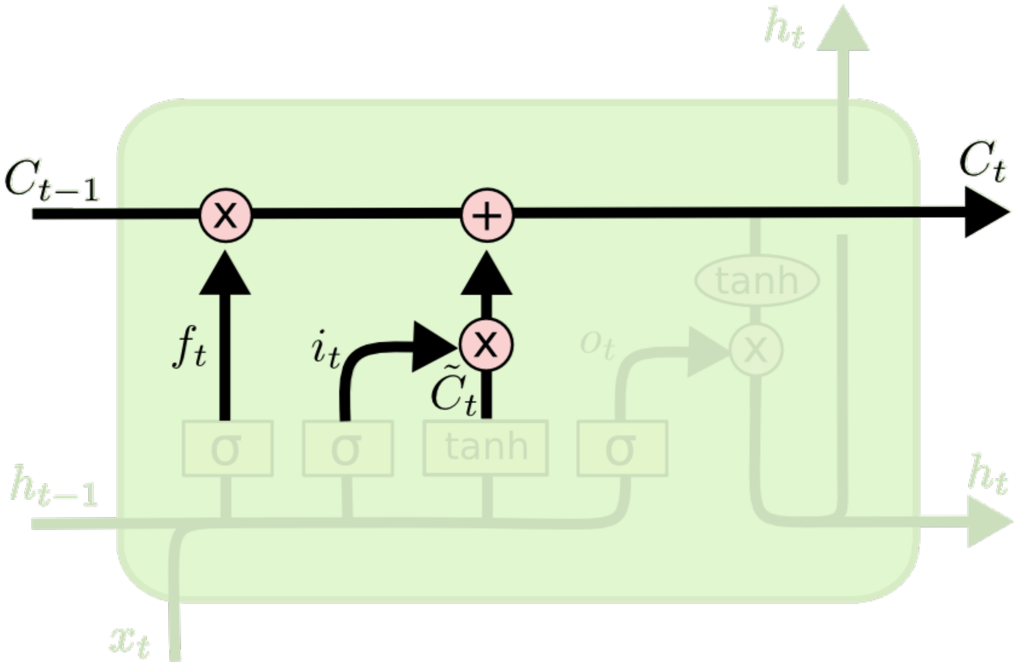


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Step-by-Step LSTM Walk Through

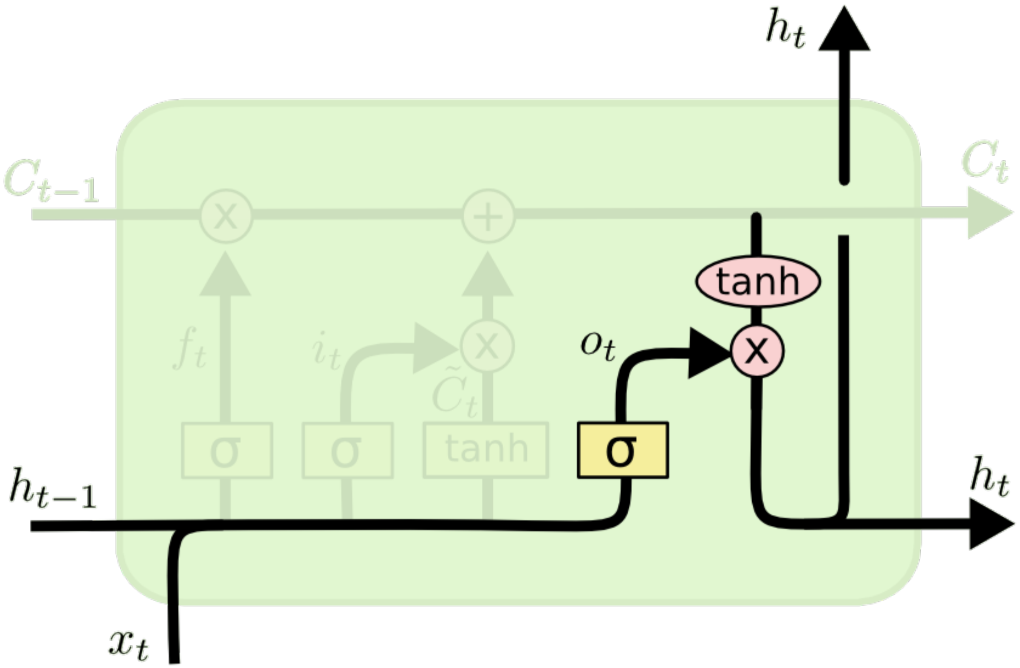
- Update Cell States:



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

Step-by-Step LSTM Walk Through

- Output:

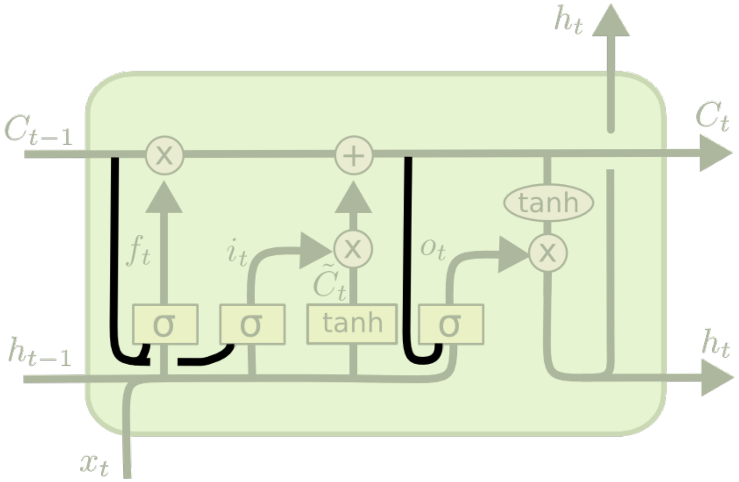


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

LSTM

Allows “peeping into the memory”



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

A memory cell using logistic and linear units with multiplicative interactions:

- Information gets into the cell whenever its input gate is on.
- Information is thrown away from the cell whenever its forget gate is off.
- Information can be read from the cell by turning on its output gate.