Recurrent Neural Networks

-The most widely used sequential model

Hao Dong

2021, Peking University

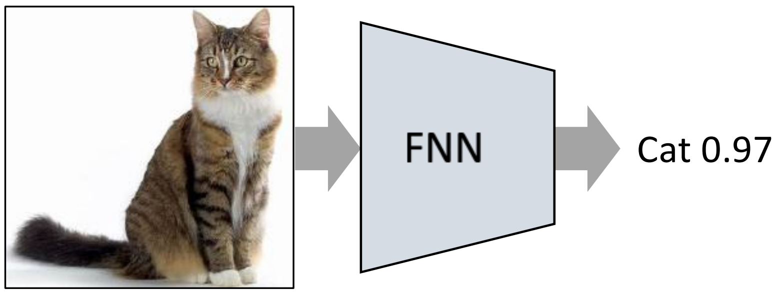
Recurrent Neural Networks

- Motivation
- Before We Begin: Word Representation
- Sequential Data
- Vanilla Recurrent Neural Network
- Long Short-Term Memory
- Time-series Applications

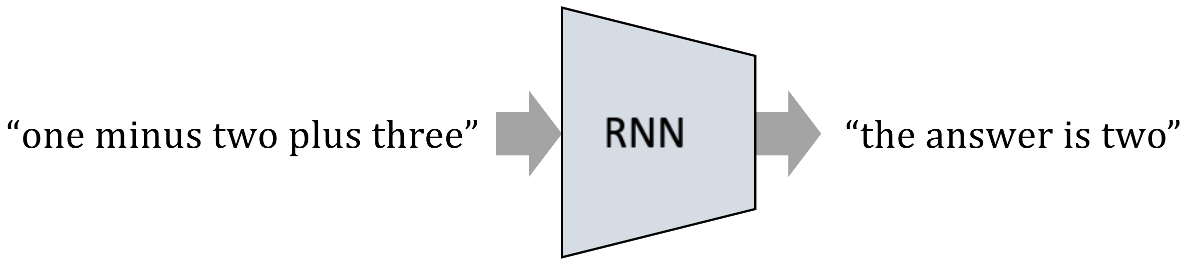
Motivation

Motivation

- Both Multi-layer Perceptron and Convolutional Neural Networks take one data sample as the input and output one result, are categorised as feed-forward neural networks (FNNs) as they only pass data layer-by-layer and obtain one output for one input (e.g., an image in input with an output of a class label)
- There are many time-series data sets, such as language, video, and bio-signals that could not fit into this framework
- The recurrent neural network (RNN) is a deep learning architecture designed for processing time-series data.



Feed-forward Neural Network



Motivation

Anti-spam

Signal Analysis

Language Translation

Image Captioning

Chatbot

Video Analysis

Sentence Generation

Before We Begin:
Word Representation

Before We Begin: Word Representation

- One-hot Vector
  - Recurrent Neural Network receives words one by one. but let see how a word is represented in computer.
  - Simple method 1: One-hot vector

"Deep Learning is very useful"

00001

00010

00100

01000

10000

Word representation

Disadvantages:

- Large vocabulary will leads to "Curse of Dimensionality".
- All word representations are independent! Too sparse!

Before We Begin: Word Representation

PEKING UNIVERSITY

- Bag of Words
  - Simple method 2: Bag of Words
    use the word frequencies to represent the sentence

“we like TensorLayer, do we?”

Word	Frequency
we	2
like	1
TensorLayer	1
do	1

→ [2, 1, 1, 1]

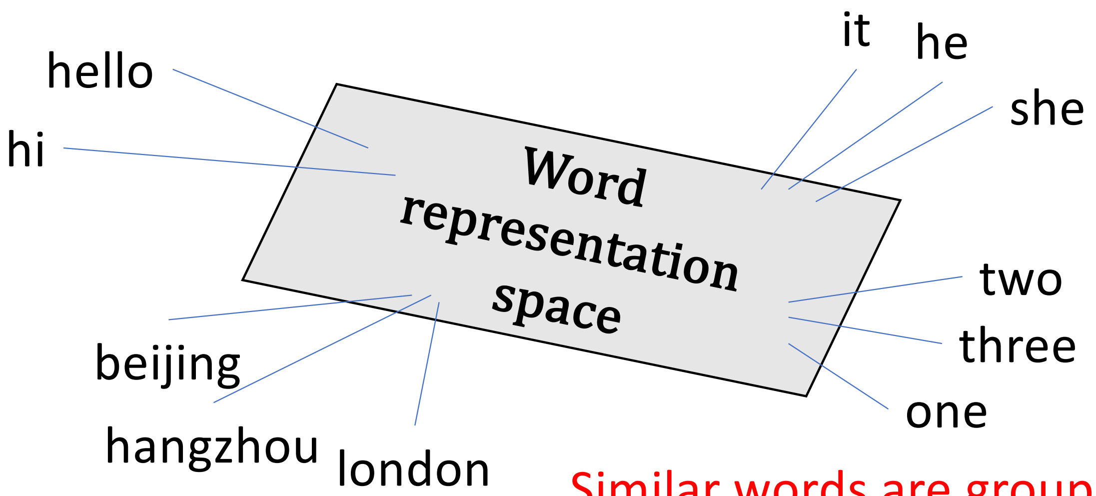
Sentence (text) representation

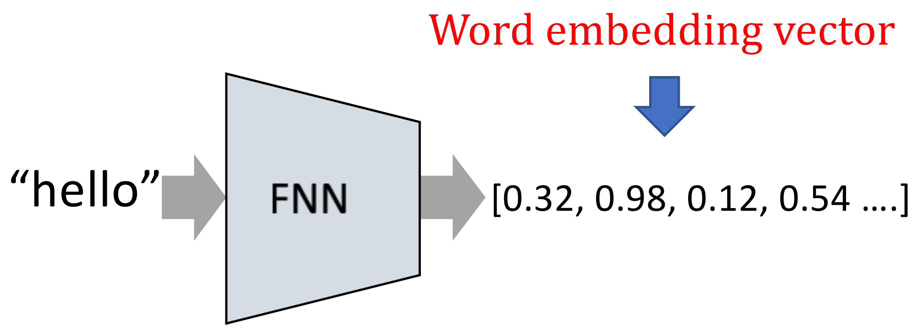
Disadvantages:

- Large vocabulary will lead to “Curse of Dimensionality”.
- Missing the information of the word locations.

Before We Begin: Word Representation

- Word Embedding
  - Represent a word using a vector of floating-point numbers.





Similar words are grouped together

Before We Begin: Word Representation

- Word Embedding
  - Given a vocabulary with 5 words, we can have a word embedding table that each word has 3 feature values.

Each word has a unique ID

Word	ID (Row Index)
---	0
deep	1
learning	2
is	3
popular	4

One-hot vector — Word embedding table — Word embedding vector

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} -0.916 & -0.837 & 0.184 \\ 2.372 & 0.706 & 1.124 \\ 1.464 & -0.688 & 1.304 \\ -0.466 & -1.457 & 0.249 \\ -0.506 & 0.539 & 0.088 \end{bmatrix} = \begin{bmatrix} -0.466 & -1.457 & 0.249 \end{bmatrix}$$

We need to find a "good" table!

In practice, we will not multiple the one-hot vector and the word embedding table, to save time, we directly use the word ID as the row index to find the embedding vector from the table

Before We Begin: Word Representation

- Ideal Word Embedding
  - Low dimension == High-level features to represent the words
  - Contains semantic information of the words

Similar words, such as "cat"-"tiger" and "water"-"liquid", should have similar word representation.

Semantic information allows semantic operations:

King − Man + Women = Queen
Paris − France + Italy = Rome

The features in the word embedding contain information, such as "gender" and "location".

Before We Begin: Word Representation

PEKING UNIVERSITY

- Learn Word Embedding
  - Existing algorithms learn the word embedding table by reading a large text document to find the patterns, which is one type of self-supervised learning.

- Use the text document as the training data without any labels.
- Find similar words by comparing the context.

- This is a blue bird
- This is a yellow bird
- This is a red bird
- This is a green bird

As the ”color” words located in similar locations of the sentences, we can group the “color” words together.

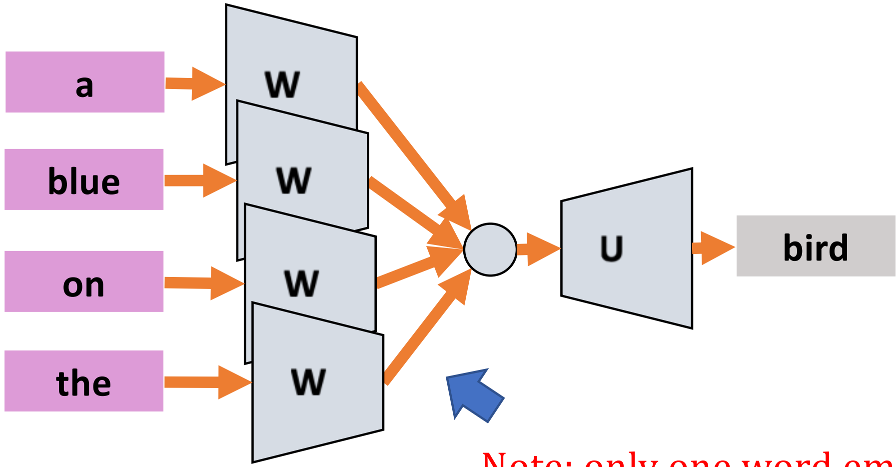
Before We Begin: Word Representation

- Word2Vec
  - Google 2013
  - Word2Vec = Skip-Gram/CBOW+ Negative Sampling

Before We Begin: Word Representation

- Word2Vec
  - **Continuous Bag-of-Words (CBOW): predicts the middle word using the context.**
    - In “a blue bird on the tree”, the context of “bird” is [“a” , “blue” , “on” , “the”, “ tree”]

this	is	a	blue	bird	on	the	tree



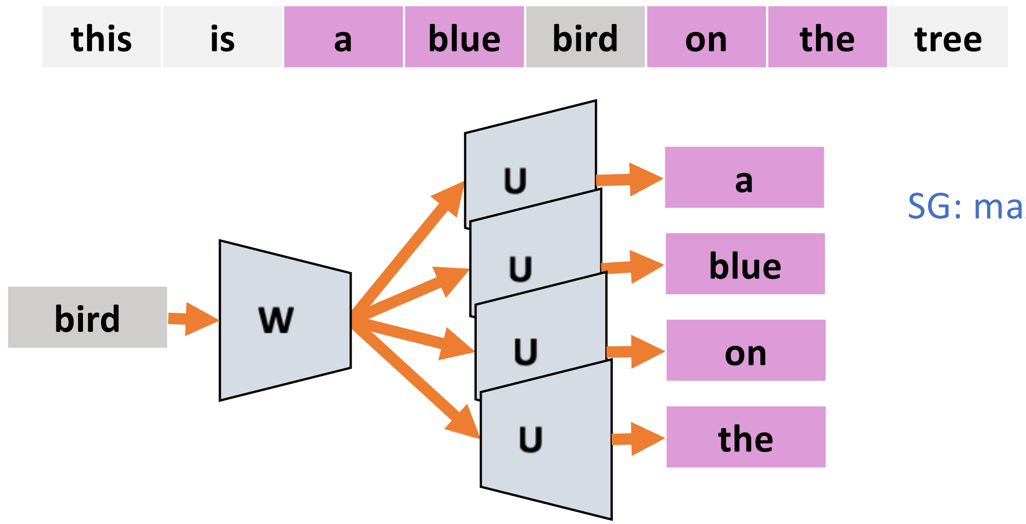
CBOW: maximise the probability of the middle word

In this example, we only consider 4 words on left and right, i.e., window size is 2.

Note: only one word embedding table here, different inputs reuse the same table.

Before We Begin: Word Representation

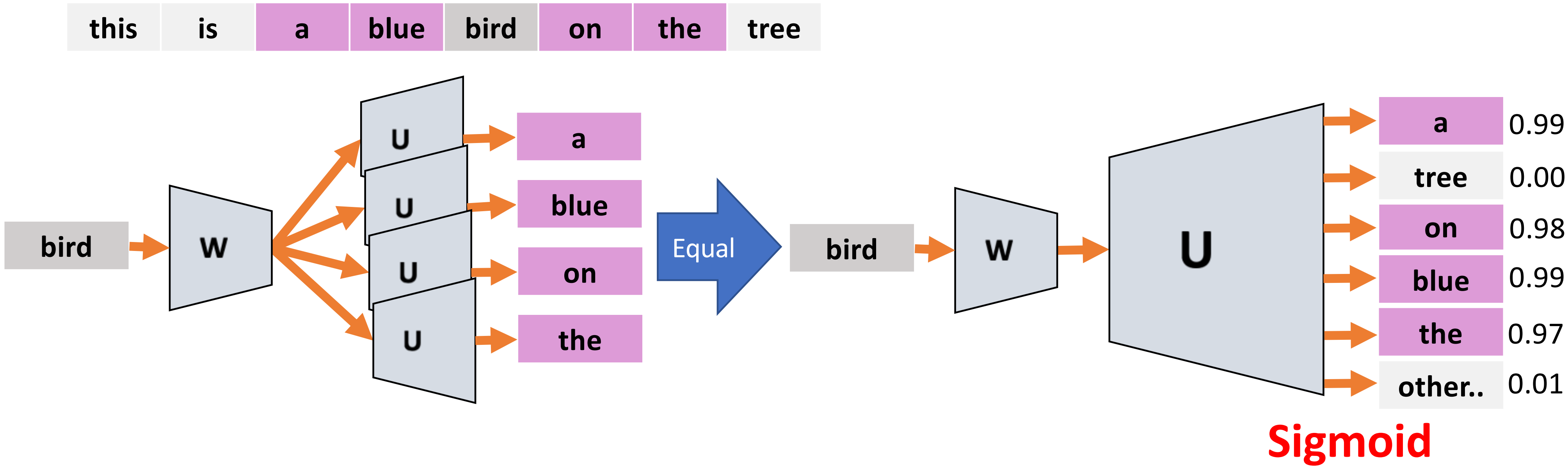
- Word2Vec
  - **Skip-Gram (SG) is opposite to CBOW, but for the same purpose**
    - CBOW predicts the middle word using the context, while SG predicts the context using the middle word. In “a blue bird on the tree”, the input of SG is “bird”, the outputs are [“a” , “blue” , “on” , “the” , “tree”]



SG: maximise the probability of the context

Before We Begin: Word Representation

- Word2Vec
  - **Skip-Gram (SG) is opposite to CBOW, but for the same propose**
    - CBOW predicts the middle word using the context, while SG predicts the context using the middle word. In “a blue bird on the tree”, the input of SG is “bird”, the outputs are [“a” , “blue” , “on” , “the” , “tree”]



Before We Begin: Word Representation

- Word2Vec
  - **Noise-Contrastive Estimation (NCE)**
    - Skip-Gram has multiple target outputs, so we use Sigmoid instead of Softmax.
      Each word in the vocabulary is separated into either positive and negative sample, and we classify each word independently.
    - A large vocabulary will lead to large computational cost.
      We use Negative Sampling to speed up the computation of the loss function， by randomly sample N negative samples from the vocabulary.
    - This method is called **Noise-Contrastive Estimation**:

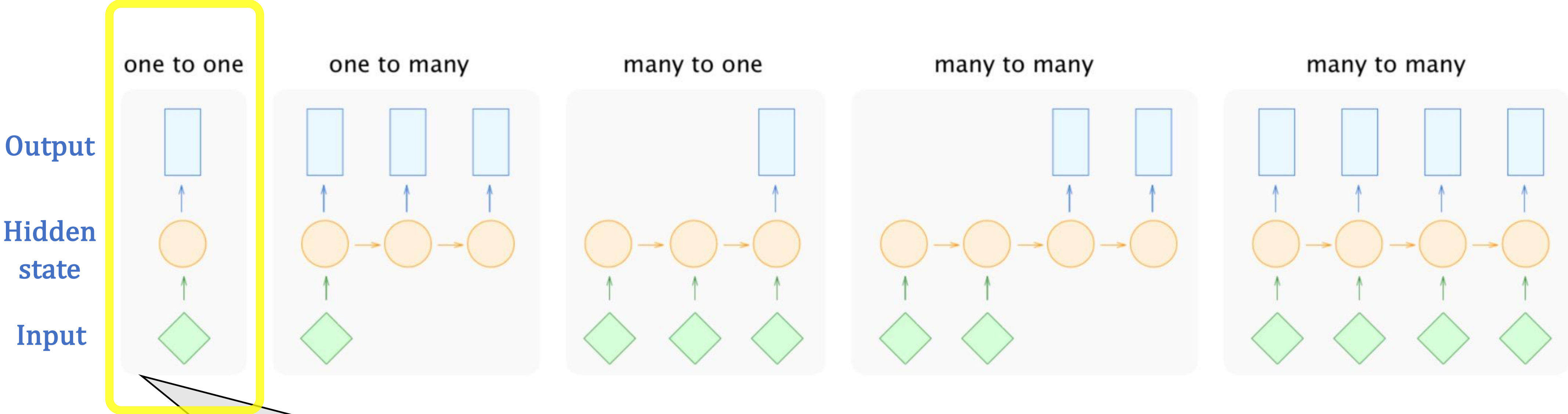
$$E = -\left(\sum_{i \in pos} \log(y_i) + \sum_{j \in neg} \log(-y_j)\right)$$

Positive samples → $i \in pos$

Randomly select N negative samples → $j \in neg$

Sequential Data

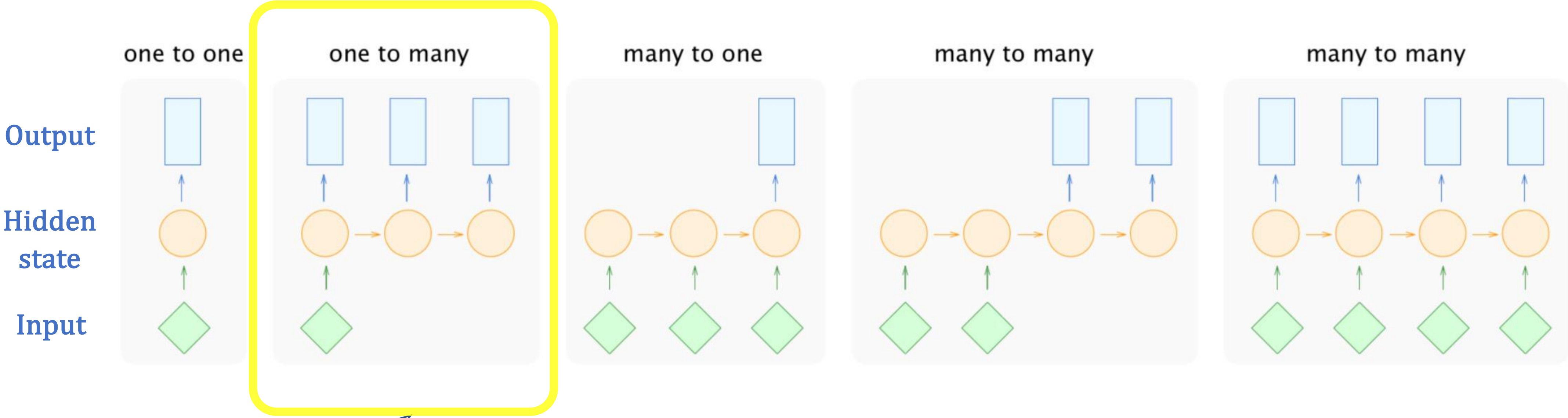
Sequential Data



Feedforward Neural Networks

Non time-series problems
such as image classification and object detection

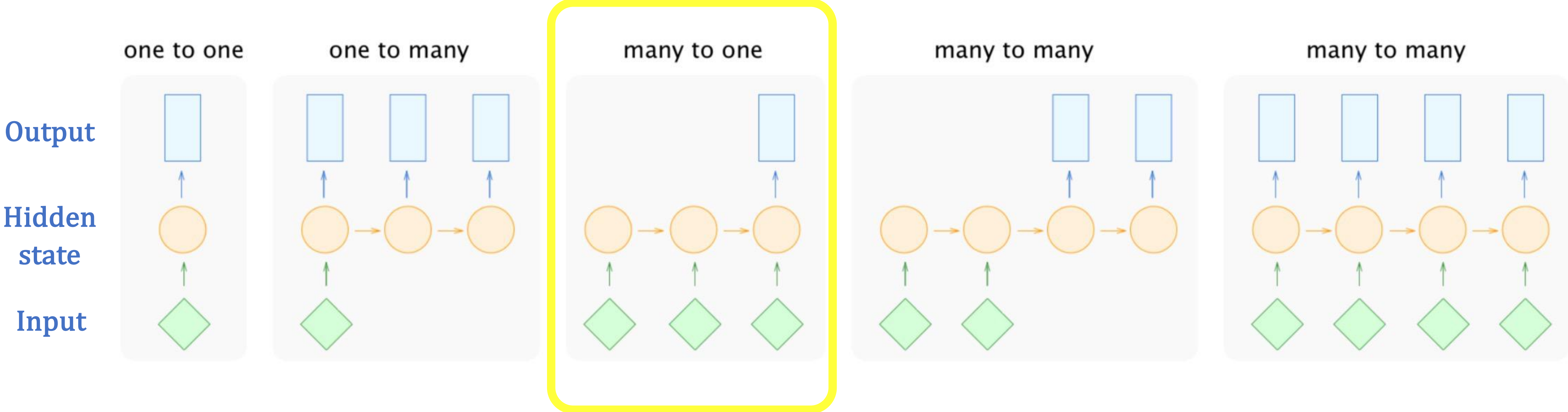
Sequential Data



One data input and many data outputs in order

Image captioning: input an image to generate a sentence

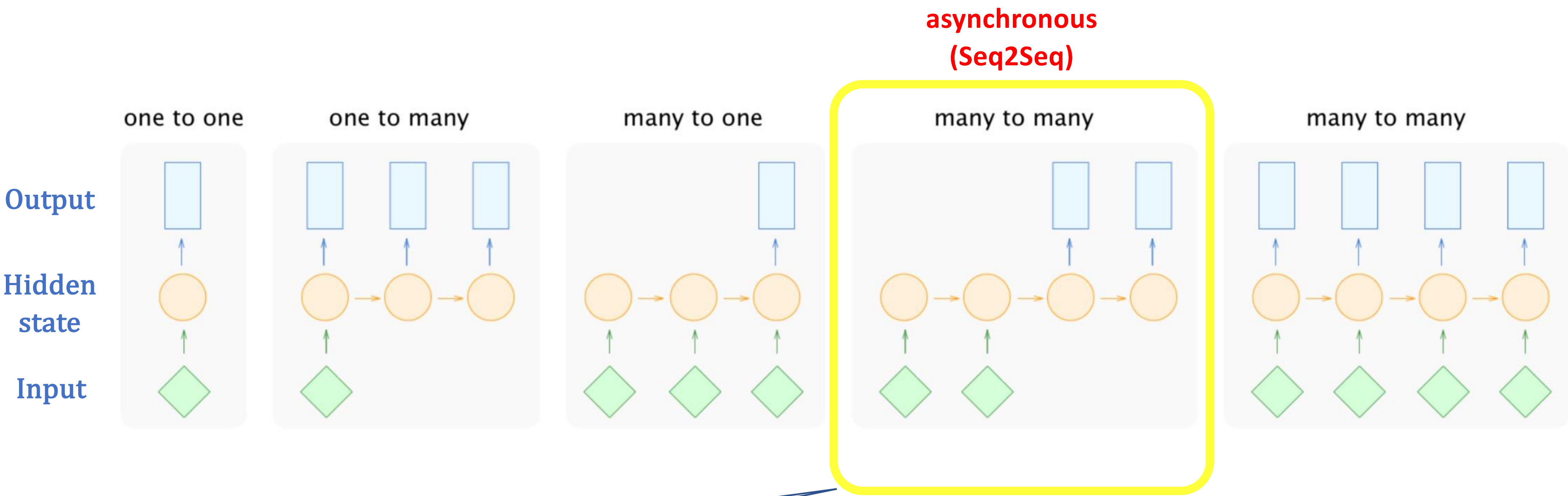
Sequential Data



Multiple data inputs and a single data output

Sentence sentiment classification: input a sentence in order and output the probability of happiness.

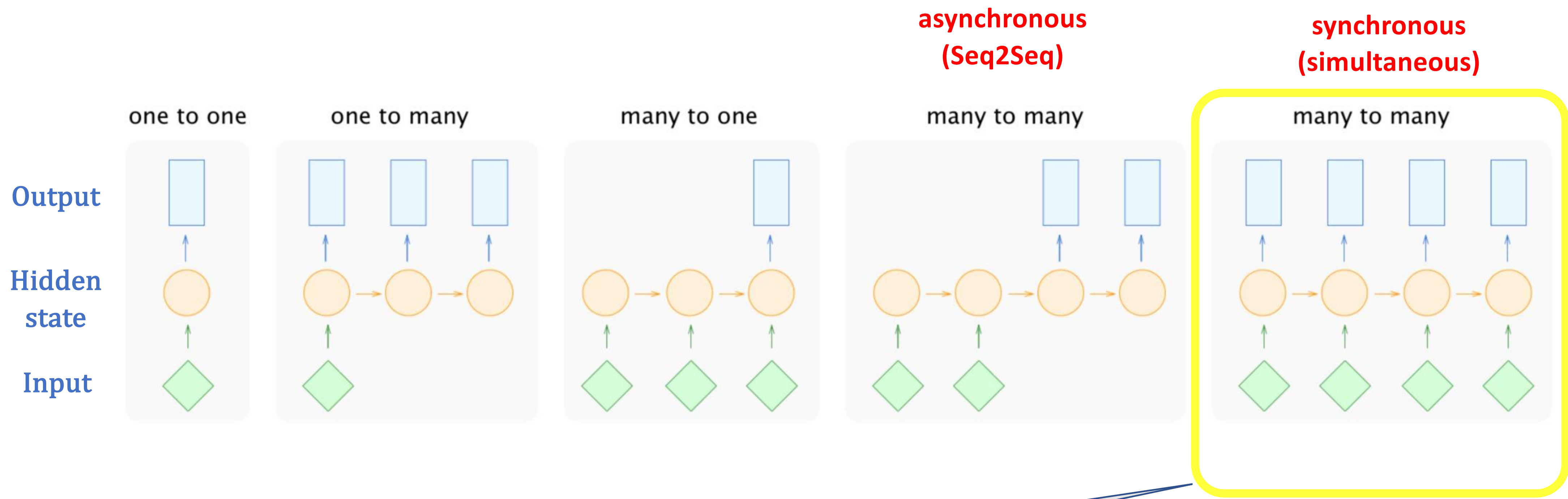
Sequential Data



Multiple data inputs and multiple data outputs

Language translation: input the entire sentence to the model before starting to generate the translated sentence

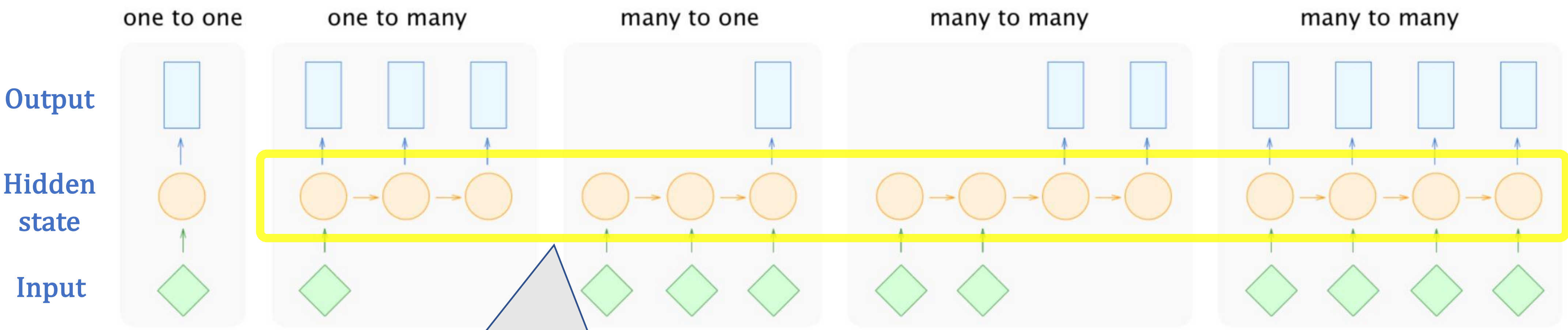
Sequential Data



Multiple data inputs and multiple data outputs

Weather prediction: input the information to the model in every time-step and output the predict weather condition.

Sequential Data



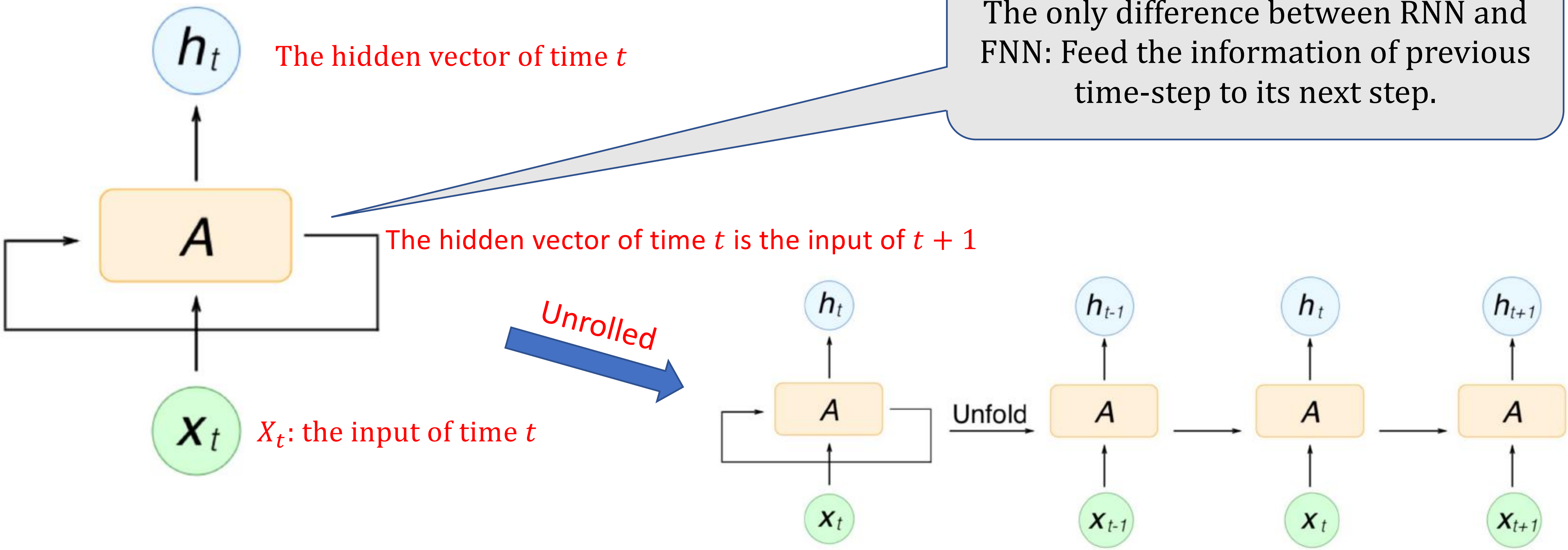
Recurrent Neural Nets
Store and process the temporal information

Vanilla Recurrent Neural Network

Vanilla Recurrent Neural Network

- Hidden Vector (State)

The only difference between RNN and FNN: Feed the information of previous time-step to its next step.



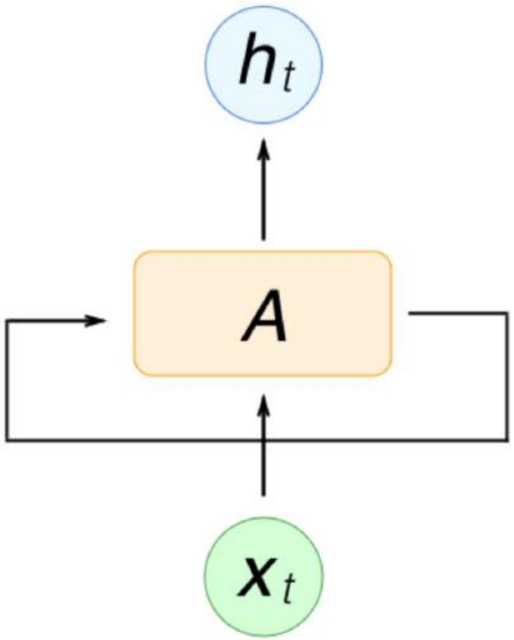
Vanilla Recurrent Neural Network

- Processing Temporal Information

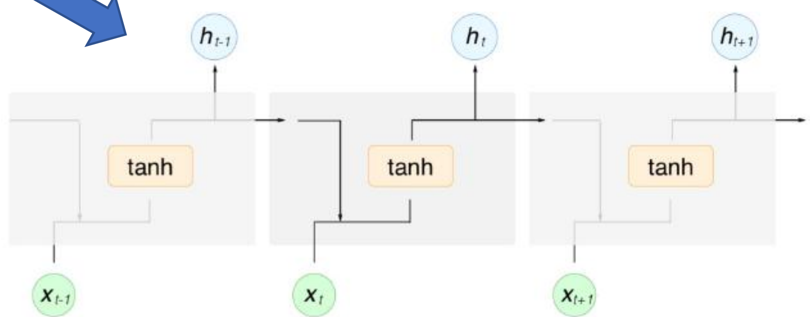
Pass information to the future

I am from China, so I speak **Chinese**

X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7



Unrolled



h_0 is a zero vector, h_{t-1} contains the information of the previous time-step.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

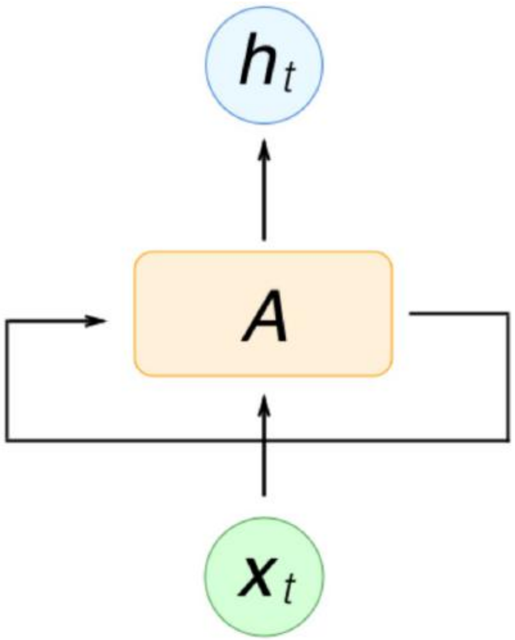
W、U 、 b : Matrix and bias

σ_h、 σ_y : activation, σ_h is *tanh*

y_t : output at time t

Vanilla Recurrent Neural Network

- Limitation: Long-Term Dependency Problem



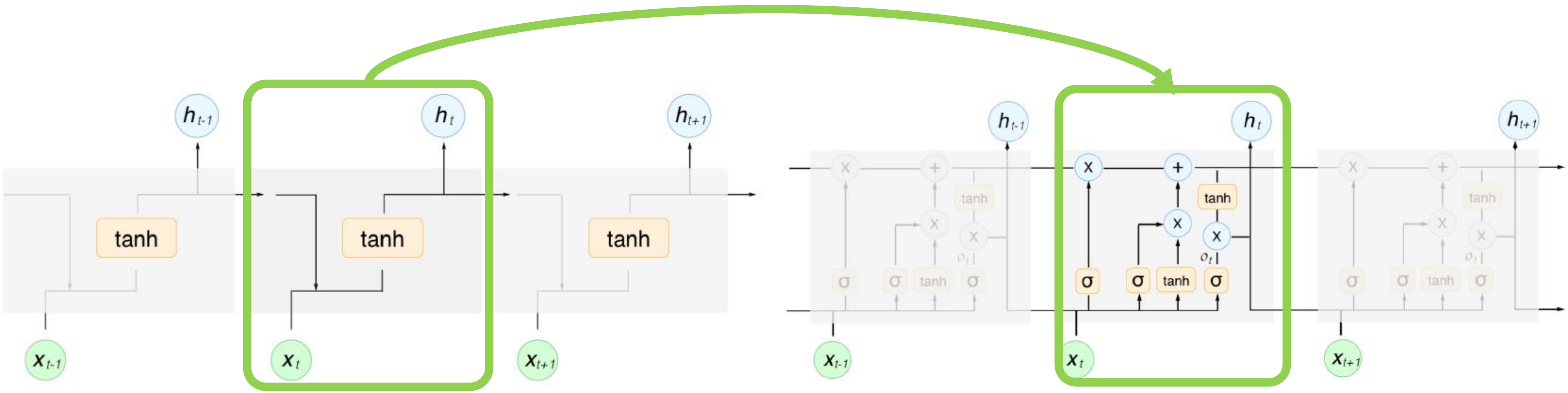
I am from China, and I live in the UK and US for 10 years, my mother language is ___?_____

Difficult to maintain information for a long term.

Long Short-Term Memory

Long Short-Term Memory

- For solving the Long-Term Dependency Problem



Vanilla RNN LSTM

Long Short-Term Memory

- Gate Function

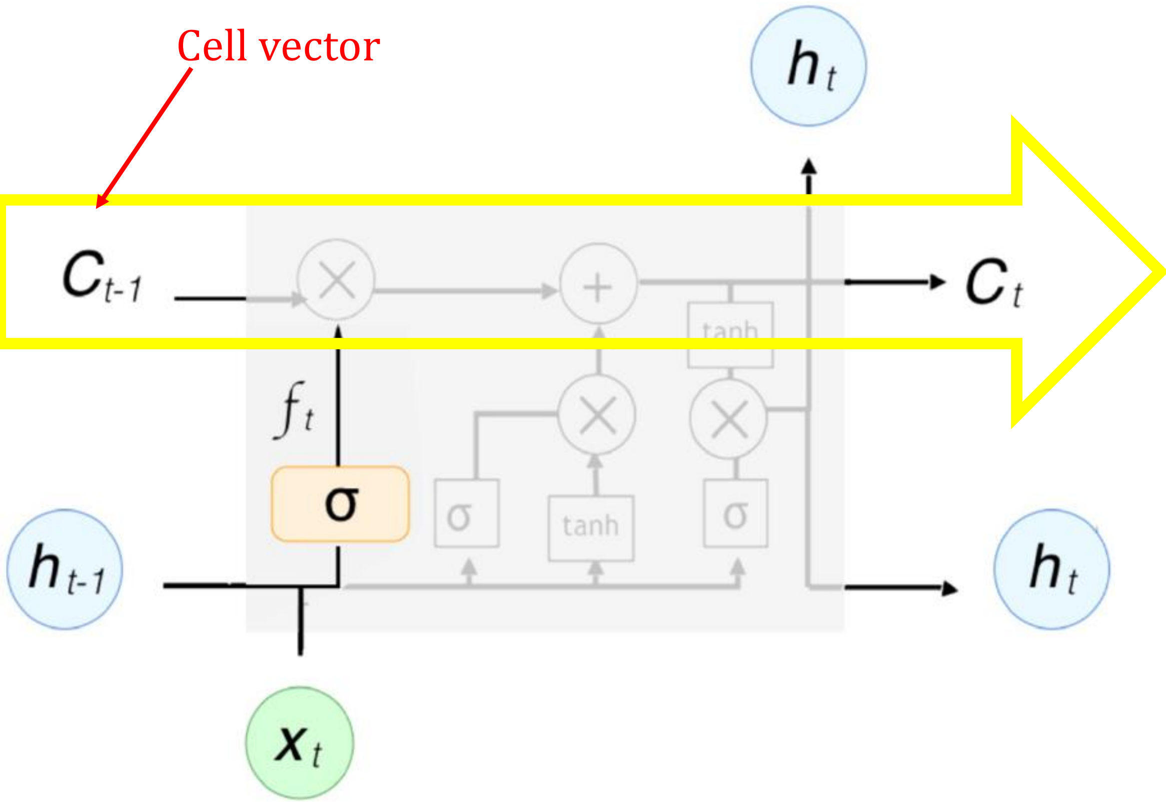
Input Vector		Gate Vector		Output Vector
0.53		0.01		0.0053
-0.32		0.99		-0.3168
0.72	⊙	0.98	=	0.7056
0.45		0.04		0.0018
1.23		0.98		1.2054
-0.24		0.02		-0.0048

0 ~ 1

- RNN has a hidden vector, LSTM has both hidden and cell vectors.
- The values in gate vector are varying from 0 to 1, 0 means "close|, 1 means "open."
- "Filter" the information by multiple the input vector and gate vector element-wisely.

Long Short-Term Memory

- Forget Gate

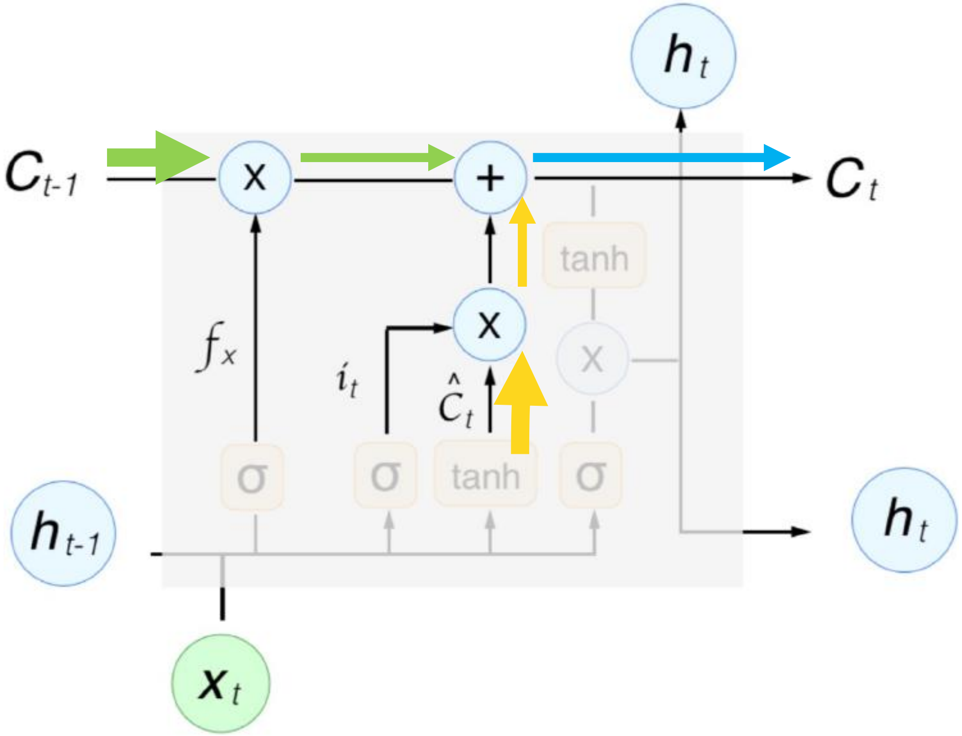


- Compute the forget gate vector:

$$f_t = sigmoid(\underbrace{[h_{t-1}, x_t]}_{\text{Concatenate two vectors}}\boldsymbol{W}_f + b_f)$$

Long Short-Term Memory

- Input Gate



- Compute input gate vector

$$i_t = sigmoid([h_{t-1}, x_t]\boldsymbol{W}_i + b_i)$$

- Compute information vector

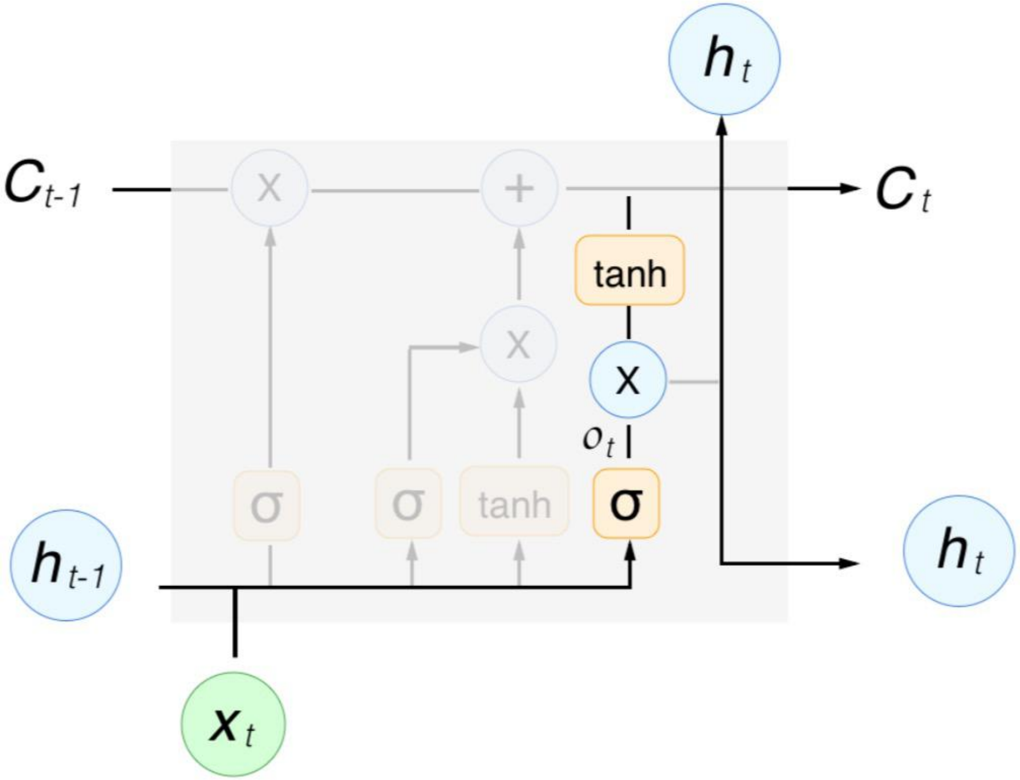
$$\hat{C}_t = \tanh([h_{t-1}, x_t]\boldsymbol{W}_C + b_C)$$

- Compute new cell vector

$$C_t = \underbrace{f_t \odot C_{t-1}}_{\text{Forget previous information}} + \underbrace{i_t \odot \hat{C}_t}_{\text{Input new information}}$$

Long Short-Term Memory

- Output Gate



- Compute output gate vector

$$o_t = sigmoid([h_{t-1}, x_t]\boldsymbol{W}_o + b_o)$$

- Compute new hidden vector

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

Long Short-Term Memory

- Questions

  - Could we use ReLU to replace Sigmoid in the Gate function?
  - Why we use tanh rather than Sigmoid when feeding information to a vector?

Long Short-Term Memory

- Variants of LSTM

  - LSTM was invented in 1997, several variants of LSTM exist, including the Gate Recurrent Unit (GRU). However, Greff et al. [1] analyzed eight LSTM variants on three representative tasks, including speech recognition, handwriting recognition, and polyphonic music modeling, and summarised the results of 5,400 experimental runs (representing 15 years of CPU time). This review suggests that none of the LSTM variants provides significant improvements to the standard LSTM.

  - Gated Recurrent Unit (GRU) does not have cell state and reduce computational cost and memory usage of LSTM.

  - ...

- Transformer, GPT ....

[1] K. Greff, R. K. Srivastava, J. Koutn ́ık, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," IEEE TNNLS, 2017.

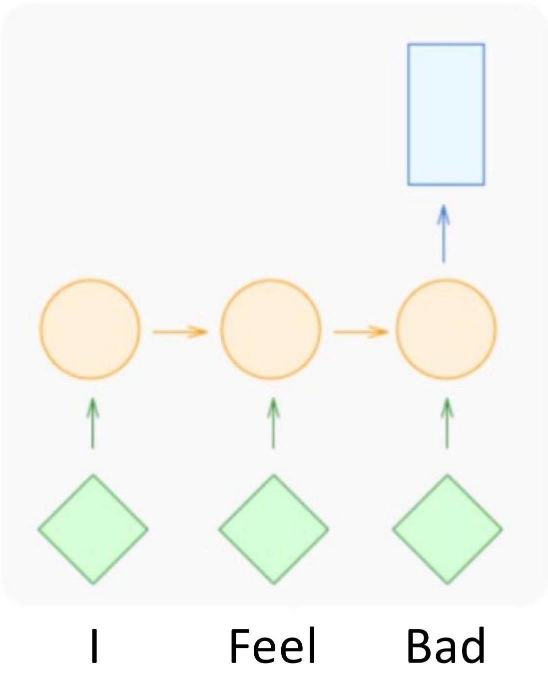
Time-series Applications

Time-series Applications

- Many-to-one: Sentence Sentiment Classification

Positive/Negative

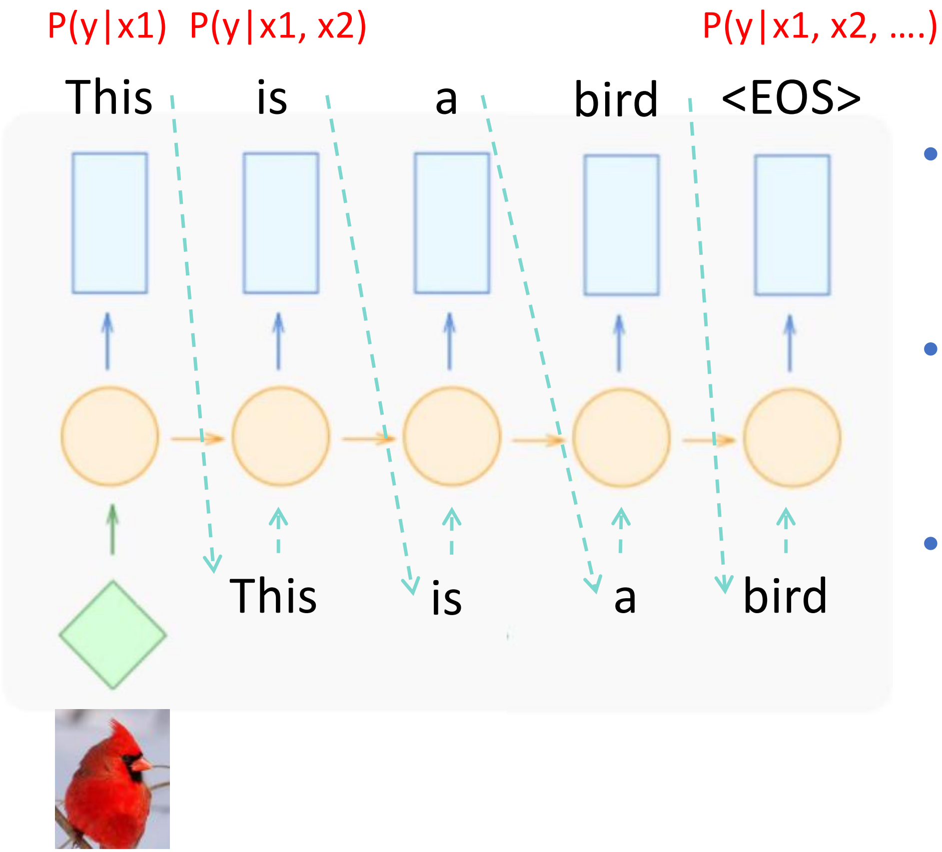
0.88 P(**Y**|x1, x2, x3 ...)



- Use the last output to compute the loss.
- Stack a fully connected layer with Softmax on the top of the hidden vector.

Time-series Applications

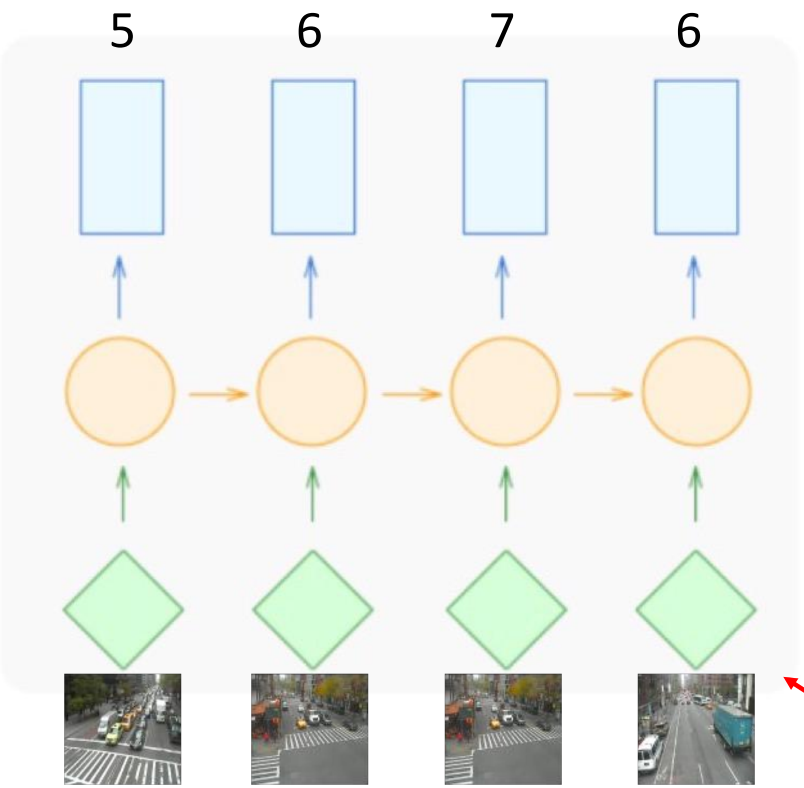
- One-to-many: Image Captioning



- Use the output of every time-step as the input of its next step.
- Terminate the process when the output is a special token for end of sentence (EOS)
- Use all outputs to compute the loss, e.g., averaged cross-entropy of all outputs.

Time-series Applications

- Synchronous Many-to-Many: Traffic Counting

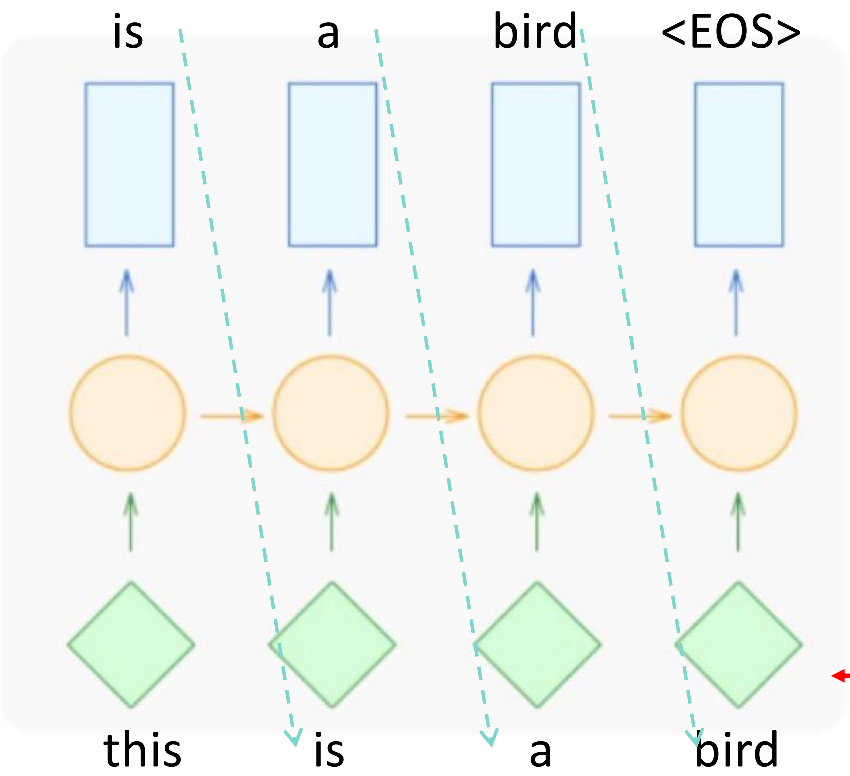


- During training, a pre-defined sequence length is required to compute the loss.
- During testing, input data samples one by one.

Sequence length = 4

Time-series Applications

- Synchronous Many-to-Many: Text Generation/Language Modelling

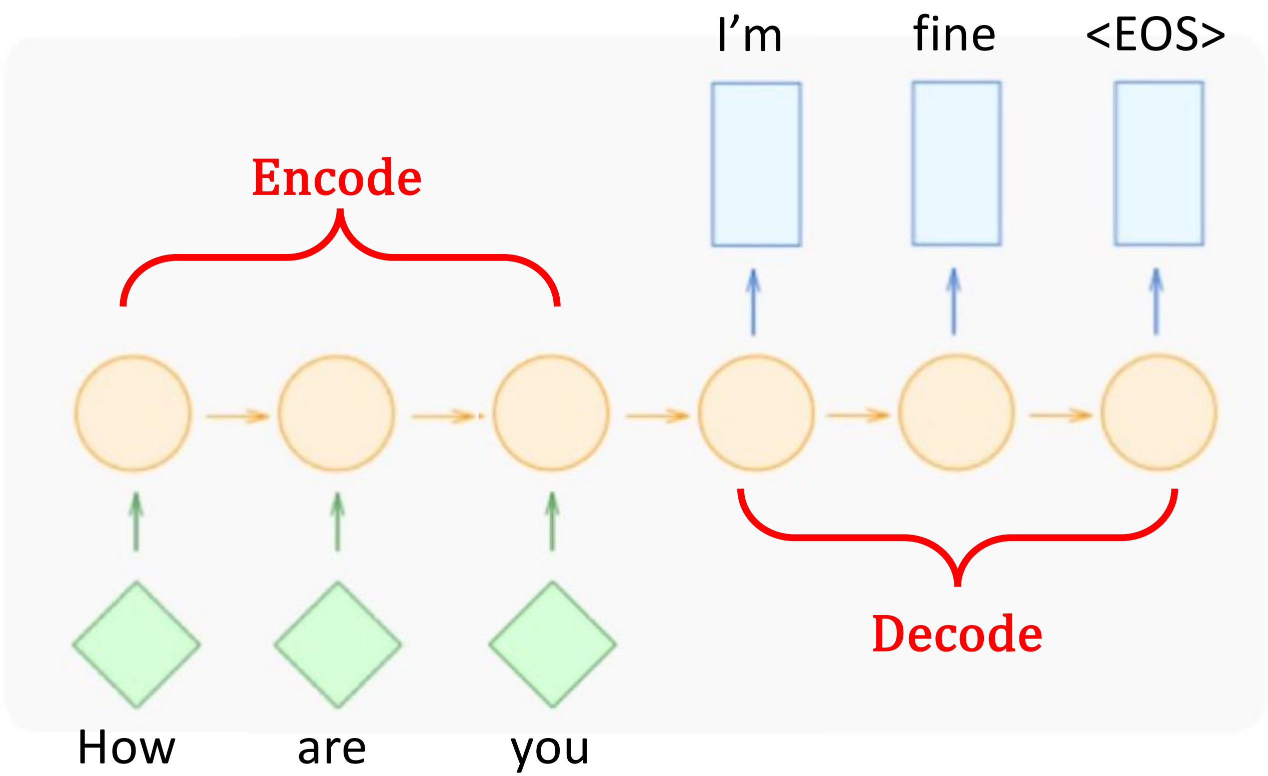


- The output of each step is equal to the input of its next step.

During testing, input "this," output the entire sentence

Time-series Applications

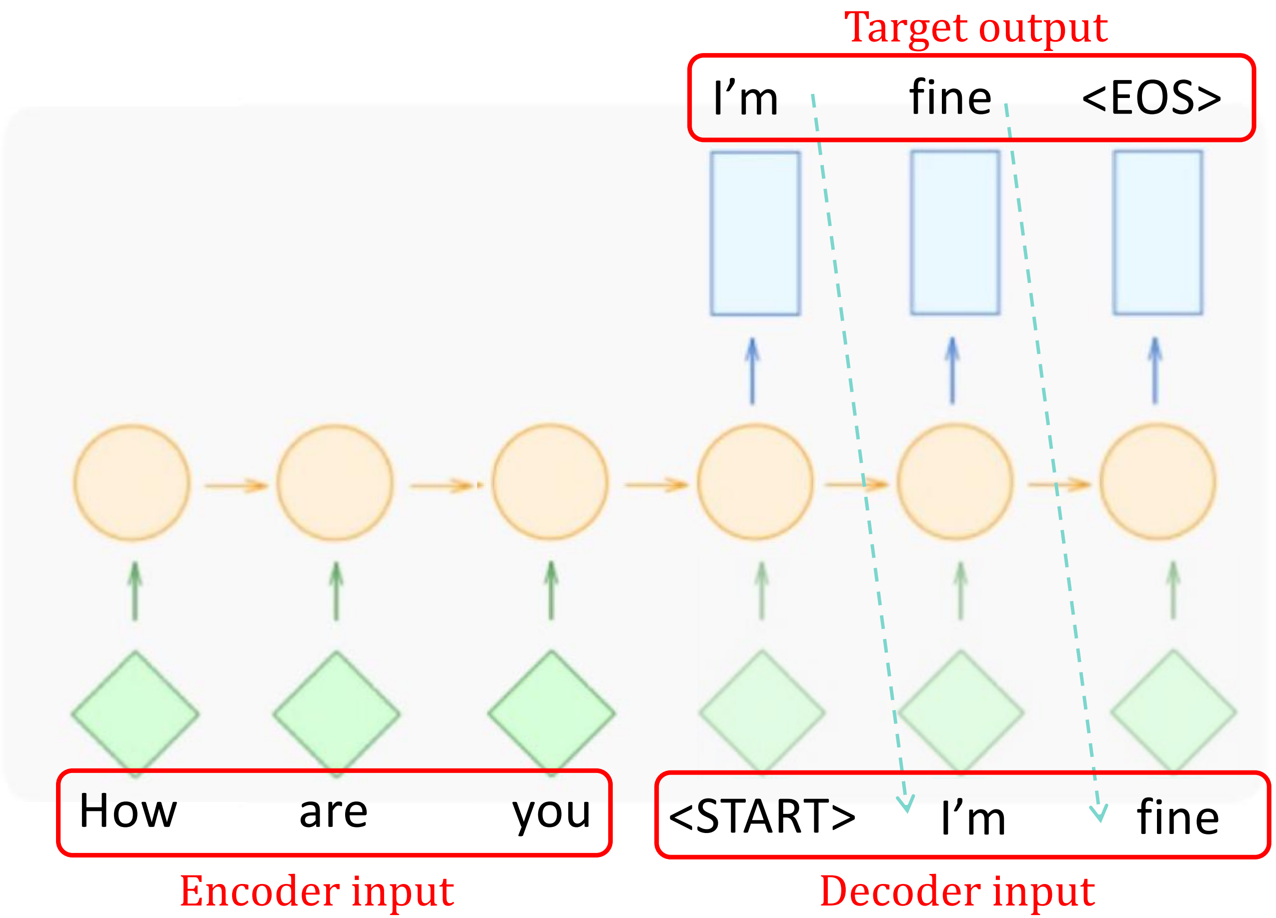
- Asynchronous Many-to-Many (Seq2Seq): Chatbot



- Encode the input sequential data before starting to output the sequential result.

Time-series Applications

- Asynchronous Many-to-Many: Chatbot



- Encode the sequential input data before starting to output the result.
- During training, add an EOS token on the target output and add a START token on the decoder input.

北京大學
PEKING UNIVERSITY

Summary

Recurrent Neural Networks

- Motivation
  - Time-series data
- Before We Begin: Word Representation
  - one-hot vector, BOW, word embedding, Word2Vec, CBOW, Skip-Gram, negative sampling, NCE
- Sequential Data
  - one-to-many, many-to-one, asynchronous many-to-many, synchronous many-to-many
- Vanilla Recurrent Neural Network
  - Hidden vector (state), long-term dependency problem
- Long Short-Term Memory
  - Cell vector (state), forget gate, input gate, output gate
- Time-series Applications
  - one-to-many, many-to-one, asynchronous many-to-many, synchronous many-to-many
  - Details of training and testing (inferencing)

Recurrent Neural Networks

- References

- Word2Vec Parameter Learning Explained. *Rong Xin. arXiv. 2016*
- Deep Learning, NLP, and Representation. *Colah Blog. 2014*
- Natural Language Processing with Deep Learning. *Christopher manning, Richard Socher. Stanford University. 2017*

Thanks