



# Recurrent neural network

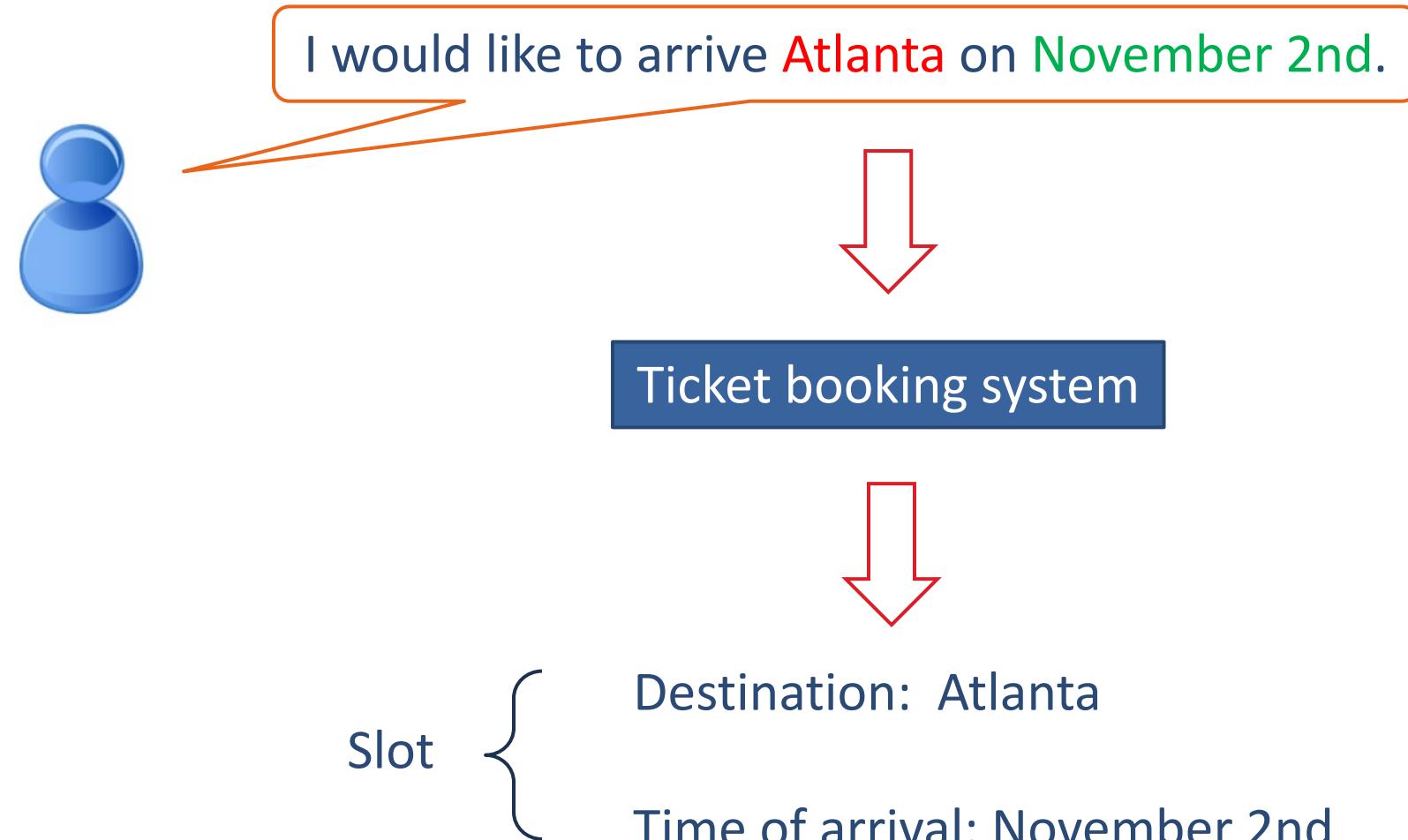
Kun Suo  
Computer Science, Kennesaw State University  
<https://kevinsuo.github.io/>



Start from examples

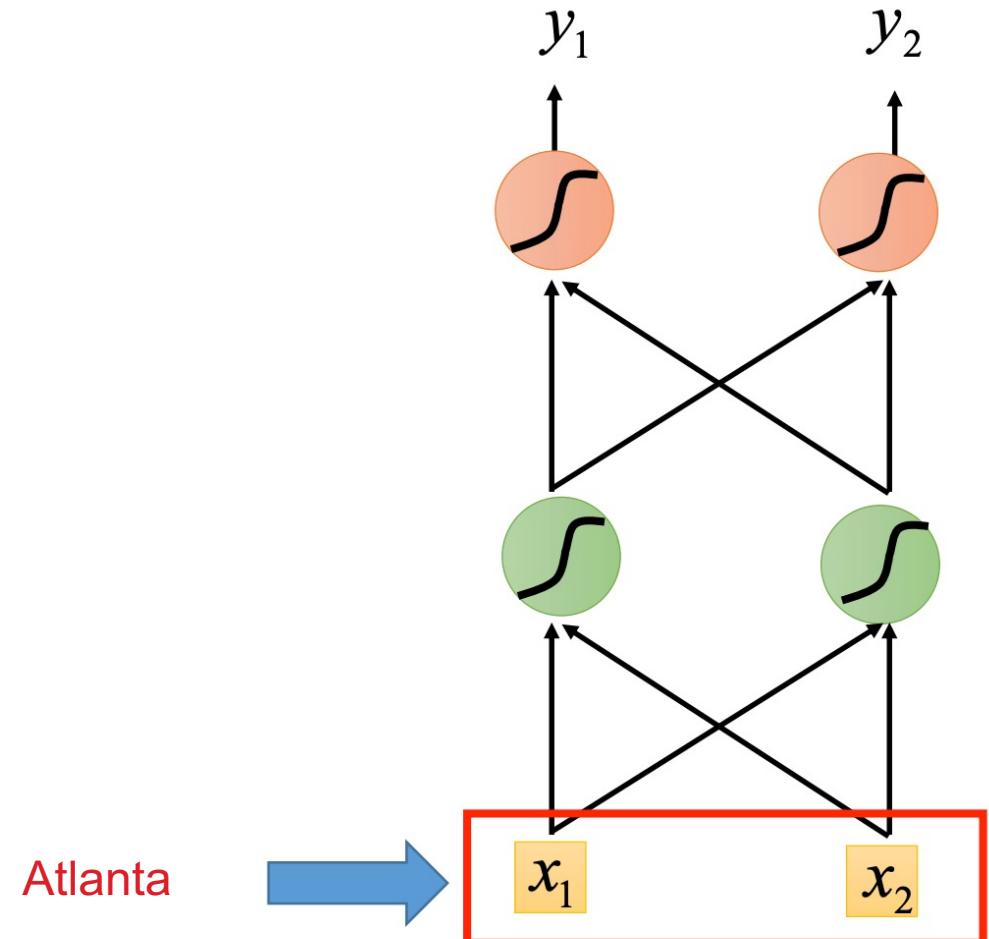
# Start from examples

## ► Slot filling



# Start from examples

- ▶ Solving slot filling by Feedforward network?
- ▶ Input: a word  
(Each word is represented as a vector)
- ▶ How to represent each word as a vector?



# How to represent each word as a vector?

## ► 1-of-N Encoding

► lexicon = {apple, bag, cat, dog, elephant}

► The vector is lexicon size.

► Each dimension corresponds to a word in the lexicon

► The dimension for the word is 1, and others are 0

$$\text{apple} = [1 \ 0 \ 0 \ 0 \ 0]$$

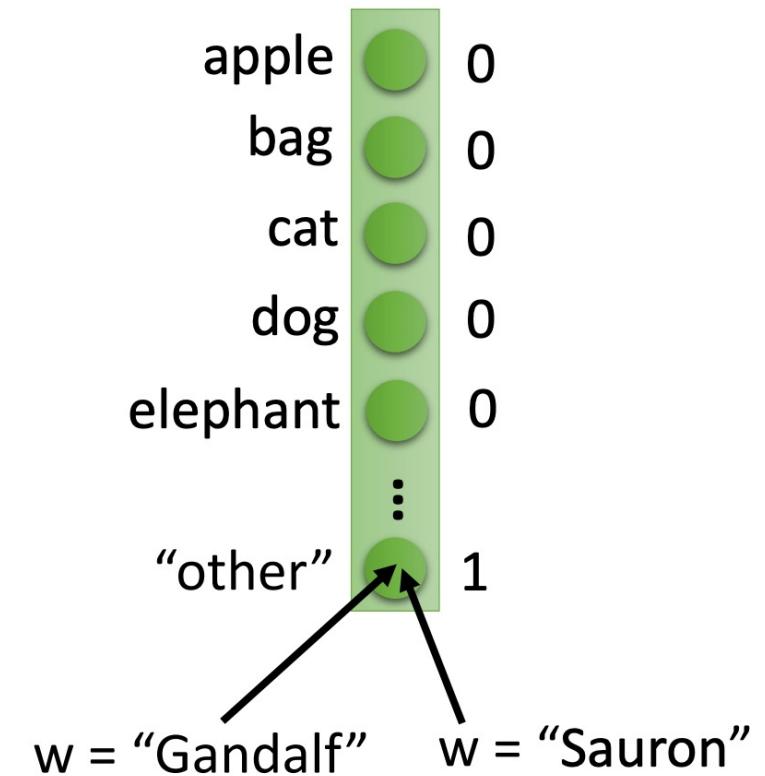
$$\text{bag} = [0 \ 1 \ 0 \ 0 \ 0]$$

$$\text{cat} = [0 \ 0 \ 1 \ 0 \ 0]$$

$$\text{dog} = [0 \ 0 \ 0 \ 1 \ 0]$$

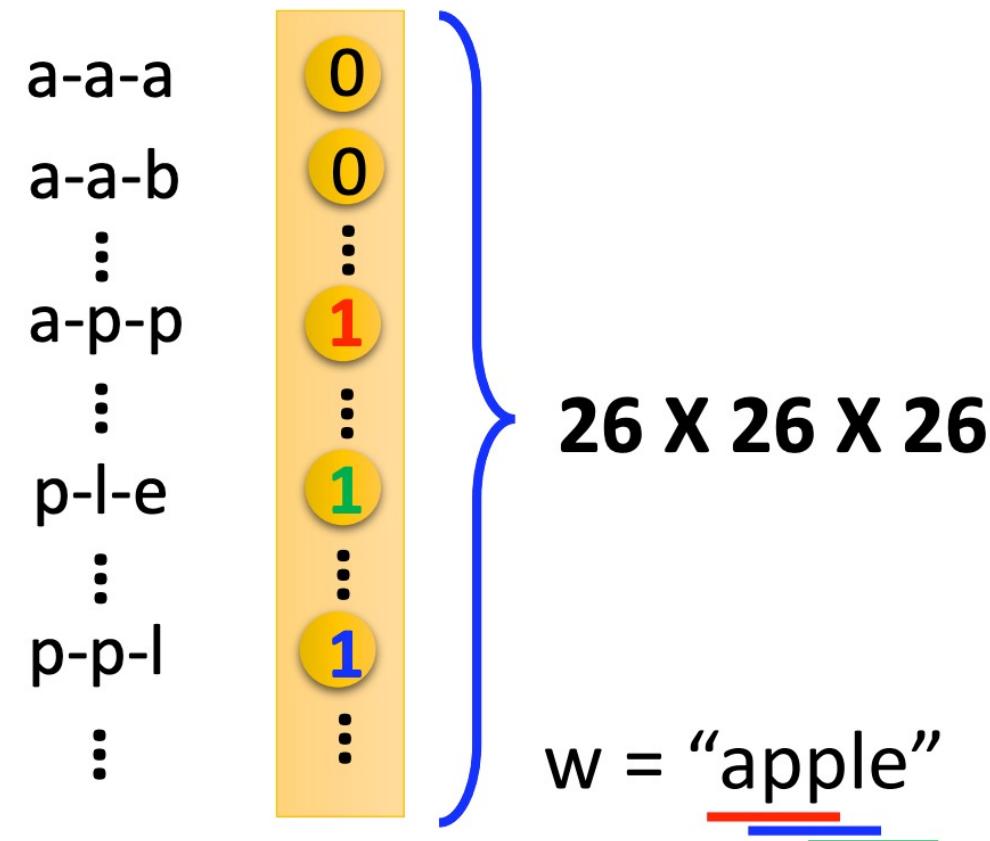
$$\text{elephant} = [0 \ 0 \ 0 \ 0 \ 1]$$

## Dimension for “Other”



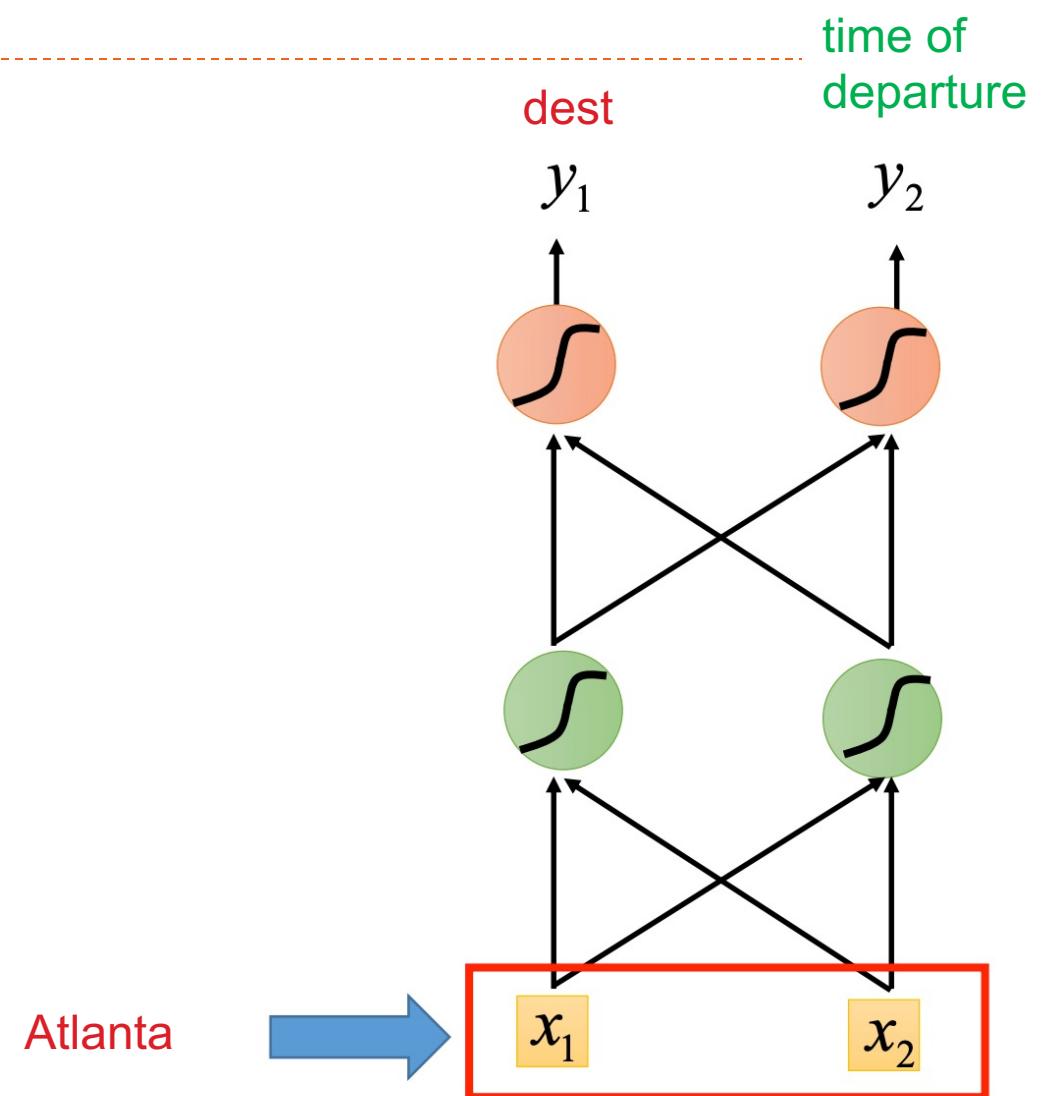
# How to represent each word as a vector?

## ► Word hashing



# Start from examples

- ▶ Solving slot filling by Feedforward network?
- ▶ Input: a word  
(Each word is represented as a vector)
- ▶ Output:  
Probability distribution that the input word belonging to the slots



# Start from examples



arrive Atlanta on November 2nd.

other dest other time time

Problem?



leave Atlanta on November 2nd.

place of departure

Neural network needs memory!

Atlanta



time of  
departure

dest

$y_1$

$y_2$

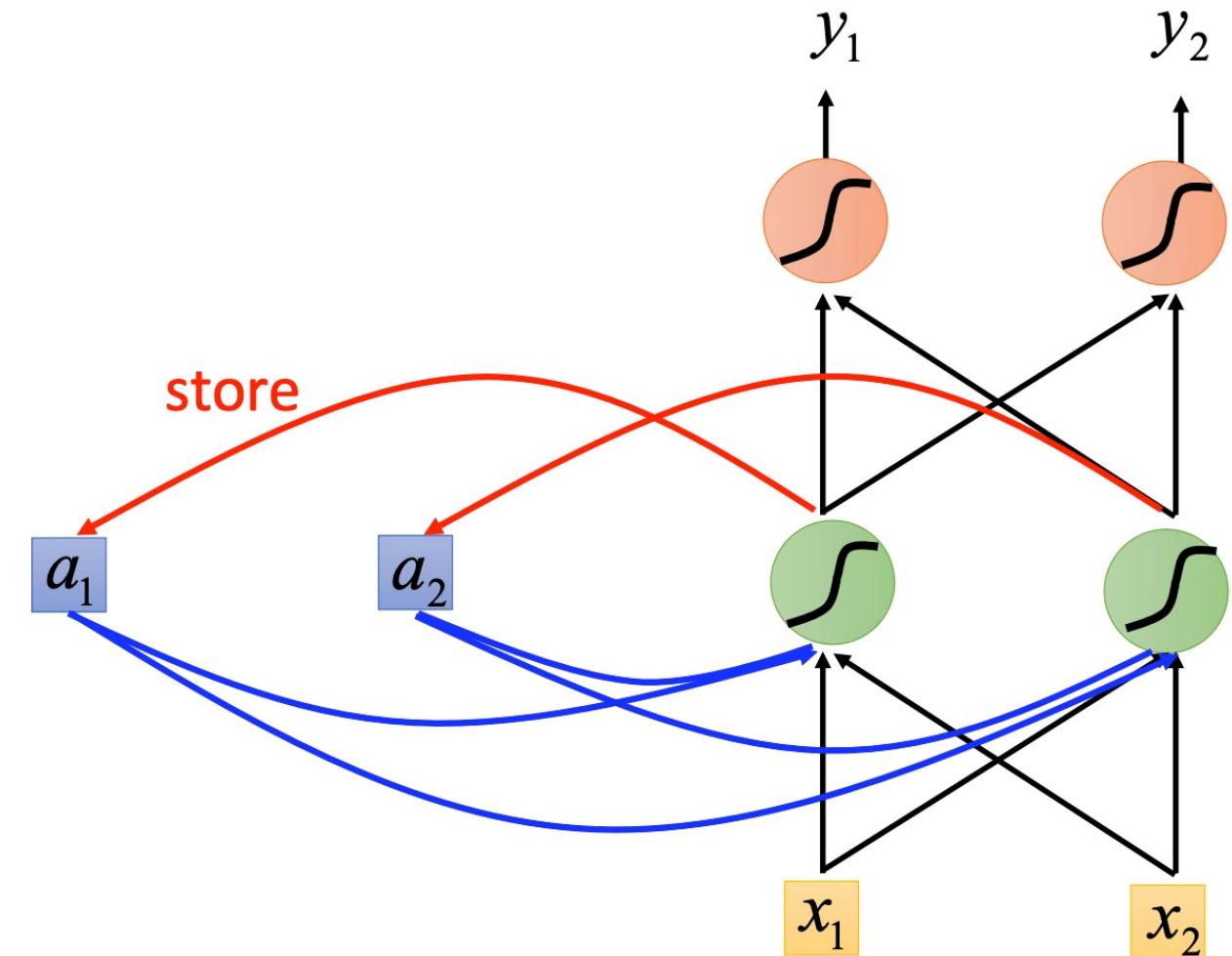


$x_1$

$x_2$

# Start from examples: Recurrent Neural Network (RNN)

- ▶ The output of hidden layer are stored in the memory.
- ▶ Memory can be considered as another input.



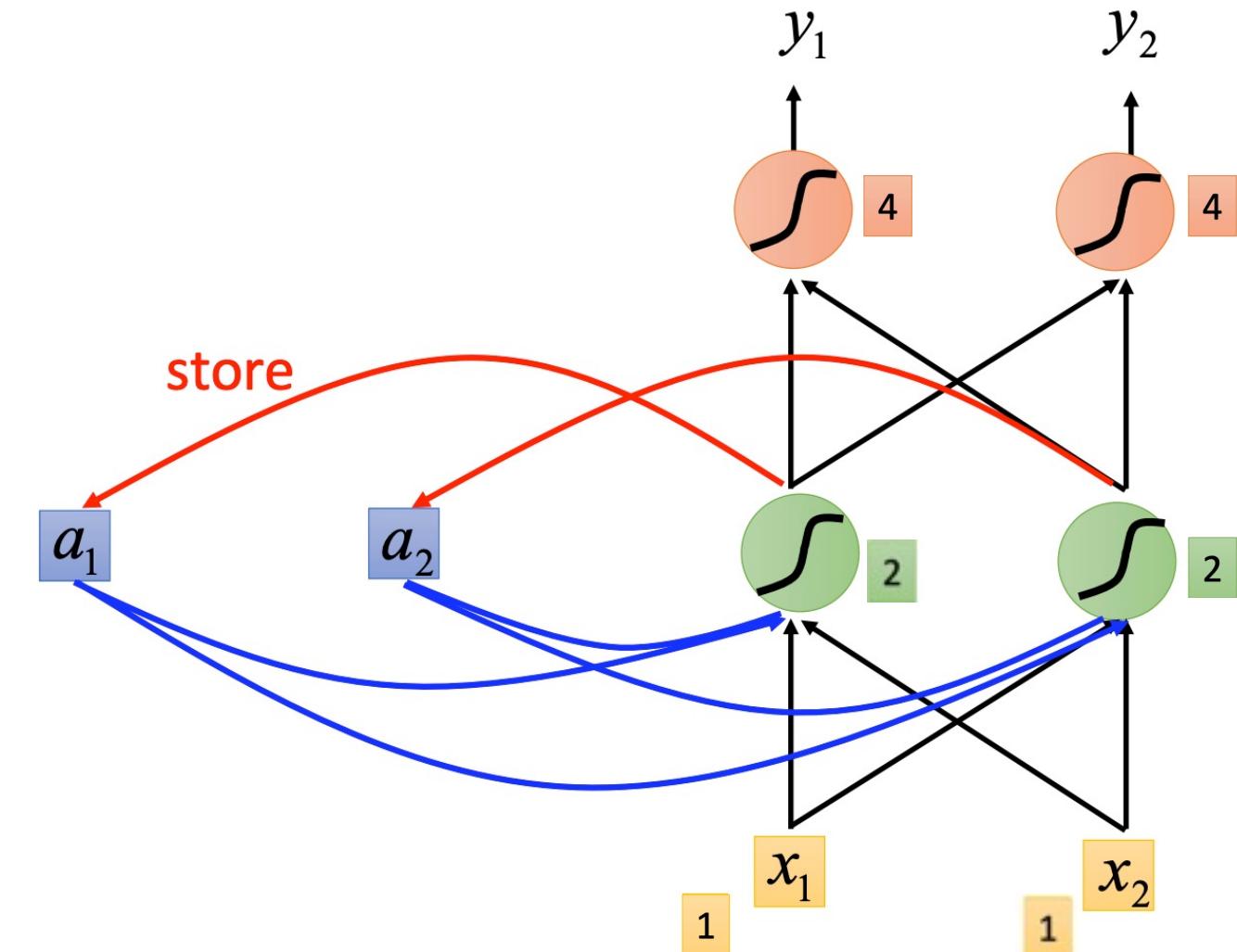
# Start from examples: Recurrent Neural Network (RNN)

- ▶ Suppose all the weights are “1”, no bias

- ▶ All activation functions are linear

- ▶ Input sequence:  $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

- ▶ The memory of  $a_1$  and  $a_2$  are 0 initially



# Start from examples: Recurrent Neural Network (RNN)

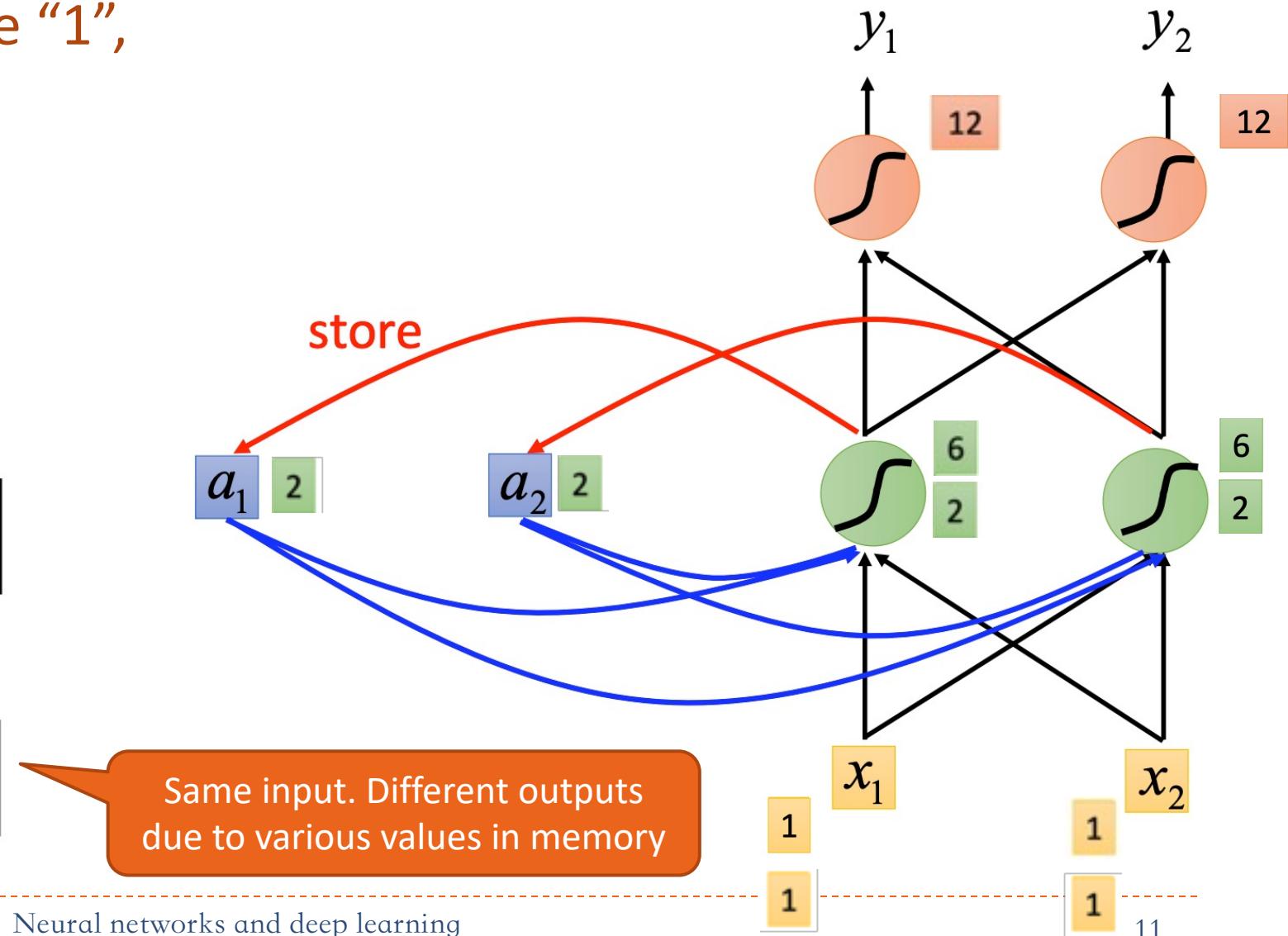
- ▶ Suppose all the weights are “1”, no bias

- ▶ All activation functions are linear

- ▶ Input sequence:  $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

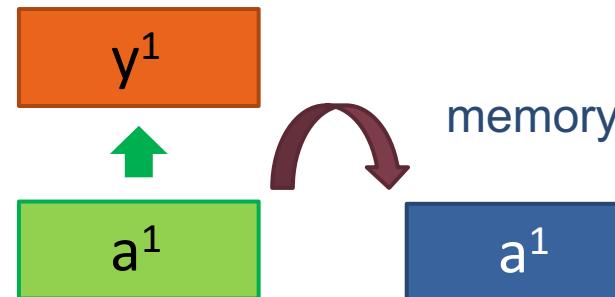
- ▶ Output sequence:

$$\begin{bmatrix} 4 \\ 4 \end{bmatrix} \begin{bmatrix} 12 \\ 12 \end{bmatrix}$$

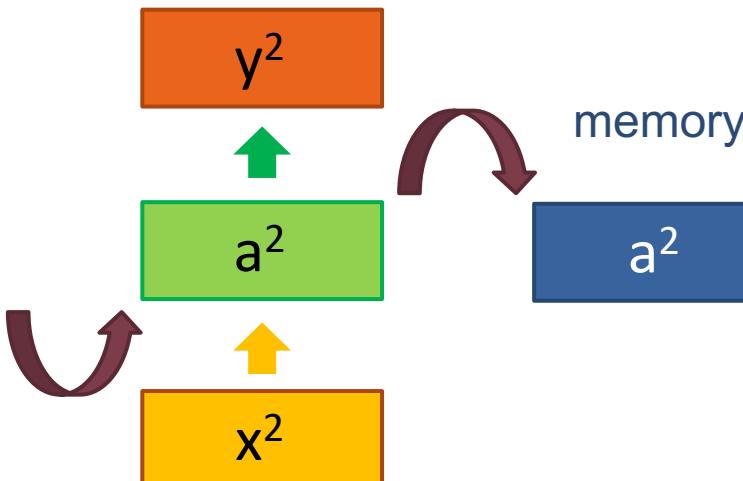


# Start from examples: Recurrent Neural Network (RNN)

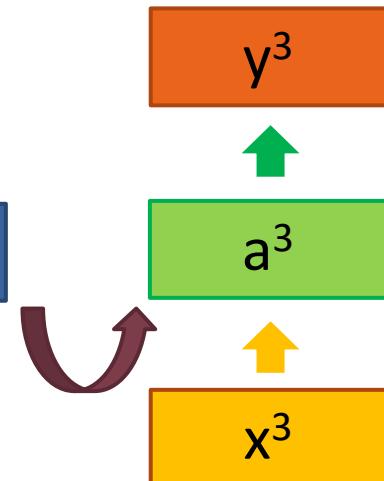
Probability of  
“arrive” in each slot



Probability of  
“Atlanta” in each slot

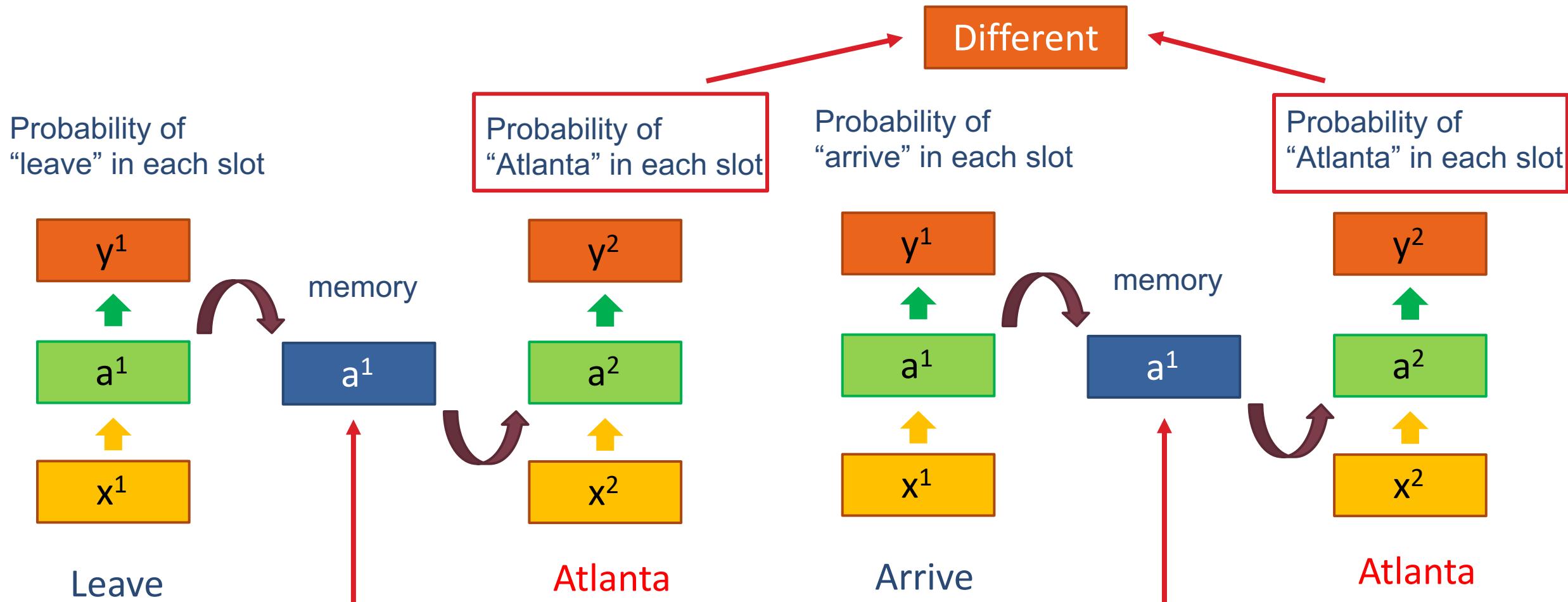


Probability of  
“on” in each slot



arrive Atlanta on November 2nd.

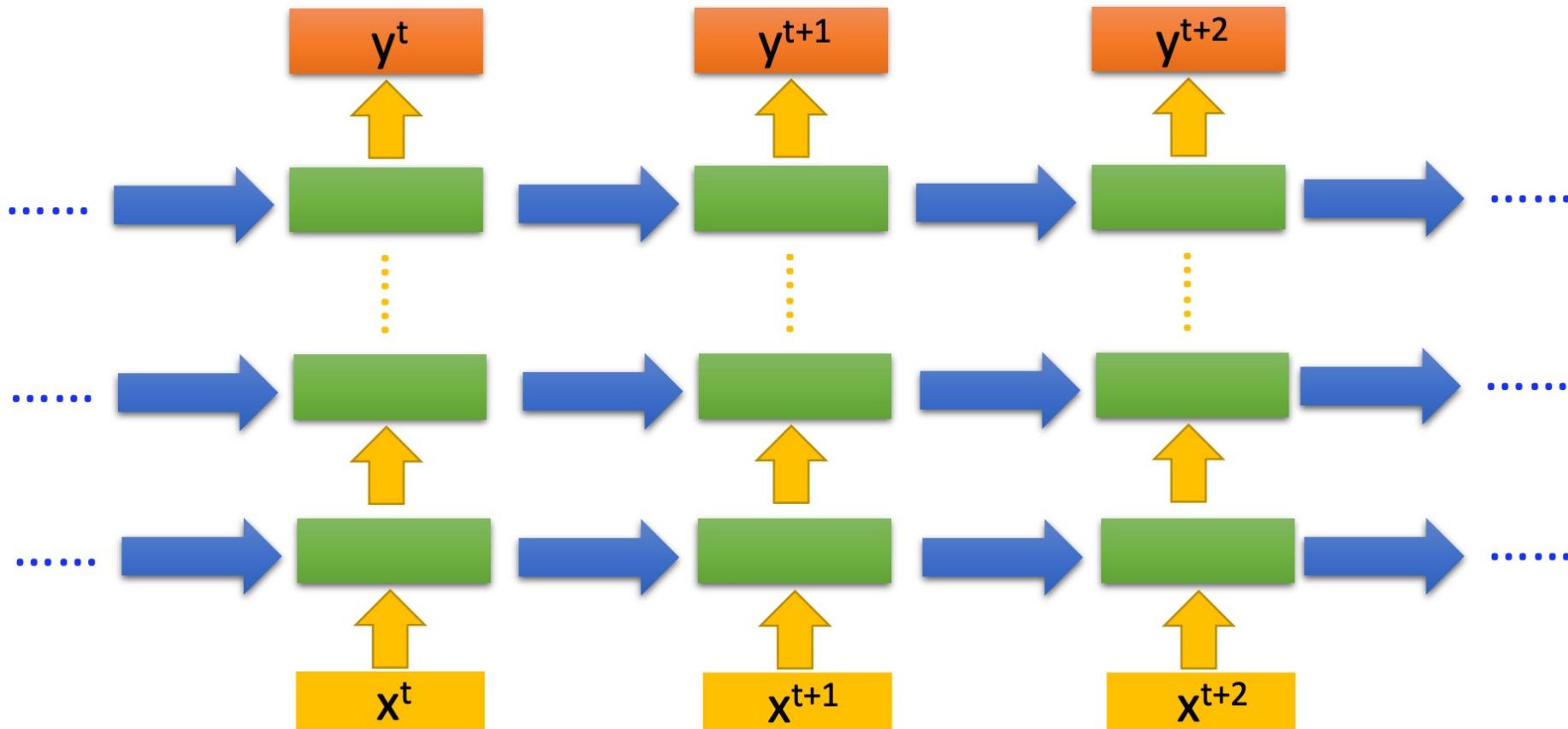
# Start from examples: Recurrent Neural Network (RNN)



The values stored in the memory is different.

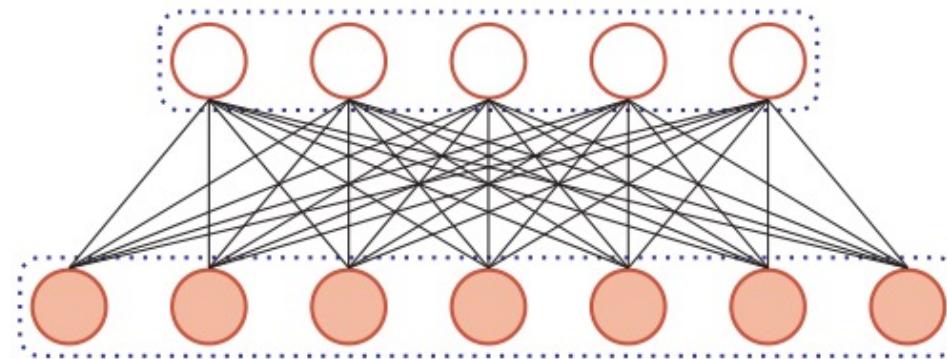
# Start from examples: Recurrent Neural Network (RNN)

► Of course, it can be deep ...

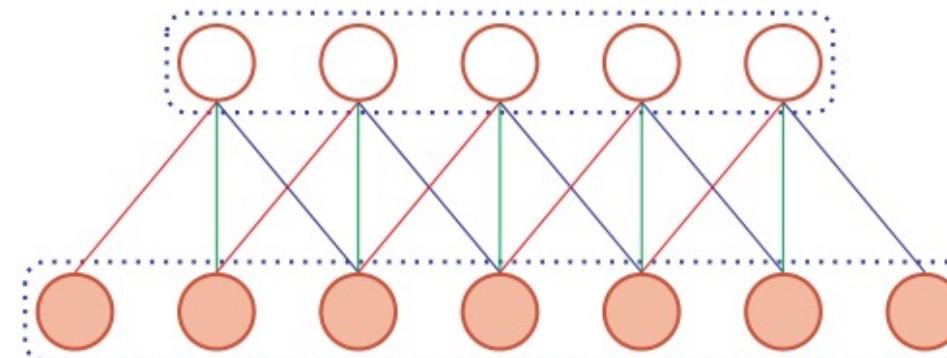


# Feedforward network

- ▶ Connections exist between layers, and there is **no connection** between nodes in each layer. (No loop)



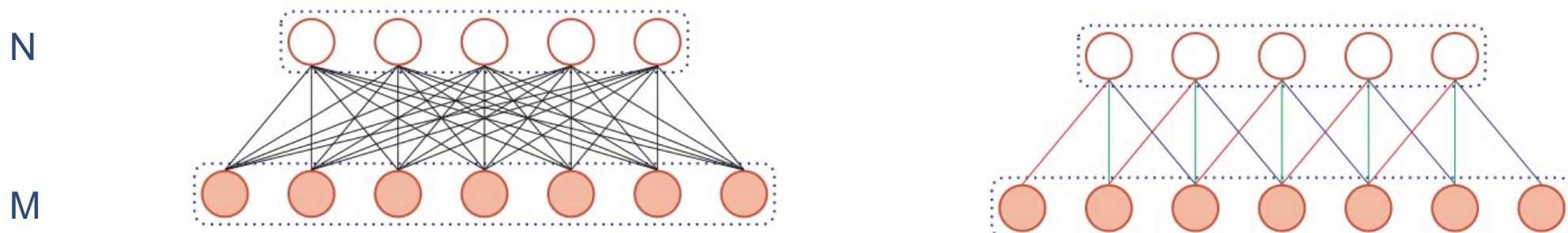
Feedforward neural network



Convolutional neural network

# Feedforward network

- ▶ The dimensions of input and output are fixed and cannot be changed arbitrarily.



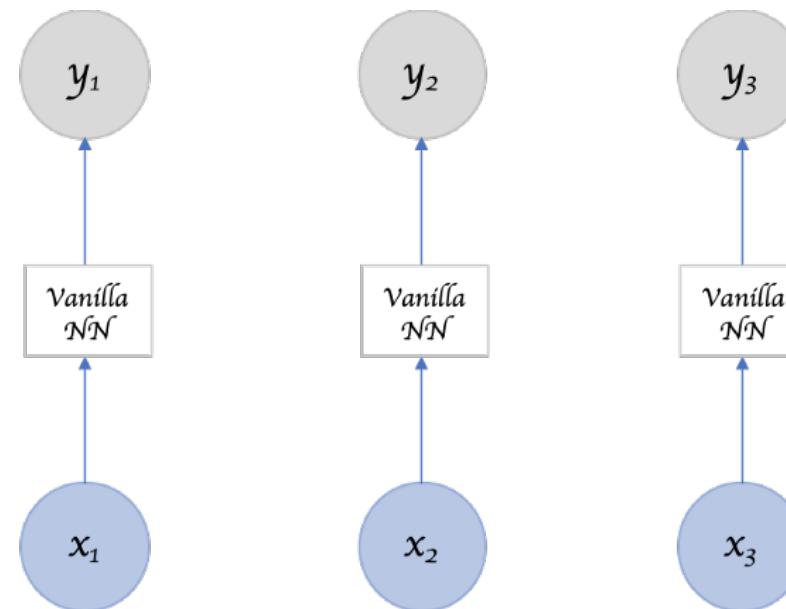
- ▶ Unable to process variable length sequence data (voice, video, text, etc.).



# Feedforward network

---

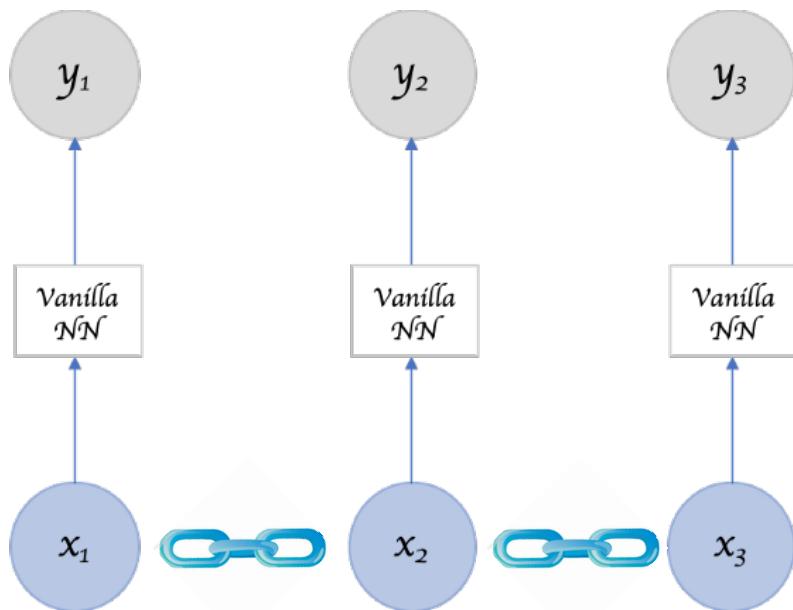
- ▶ Assumed that each input is independent, it means that the output of each network only depends on the current input.



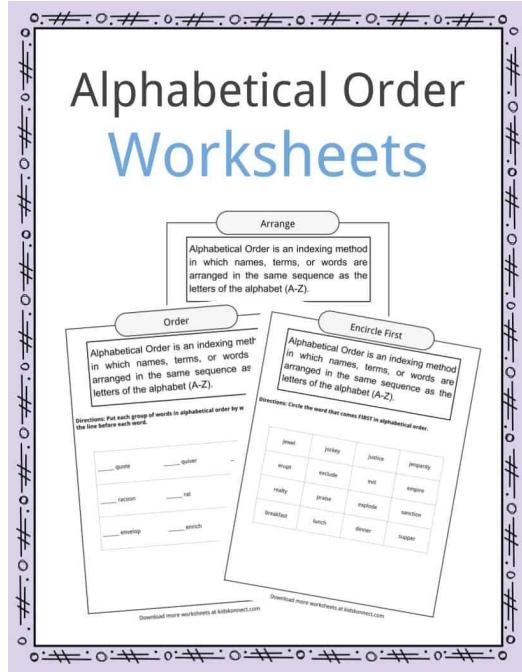
# Feedforward network

## ► Sequence data?

Example: cooking



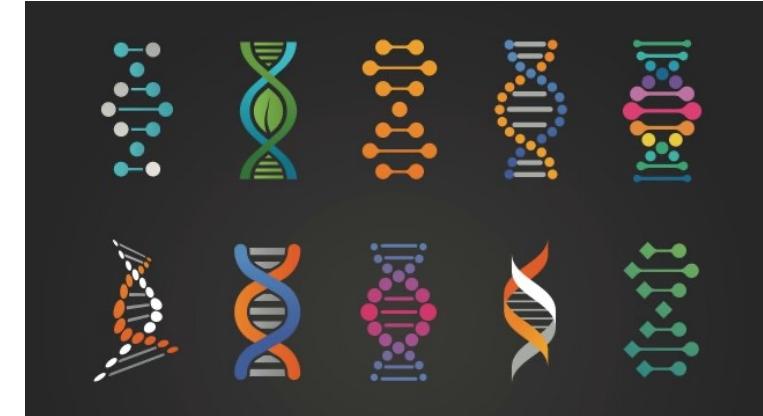
# Sequence data



Language: the order of words defines their meaning



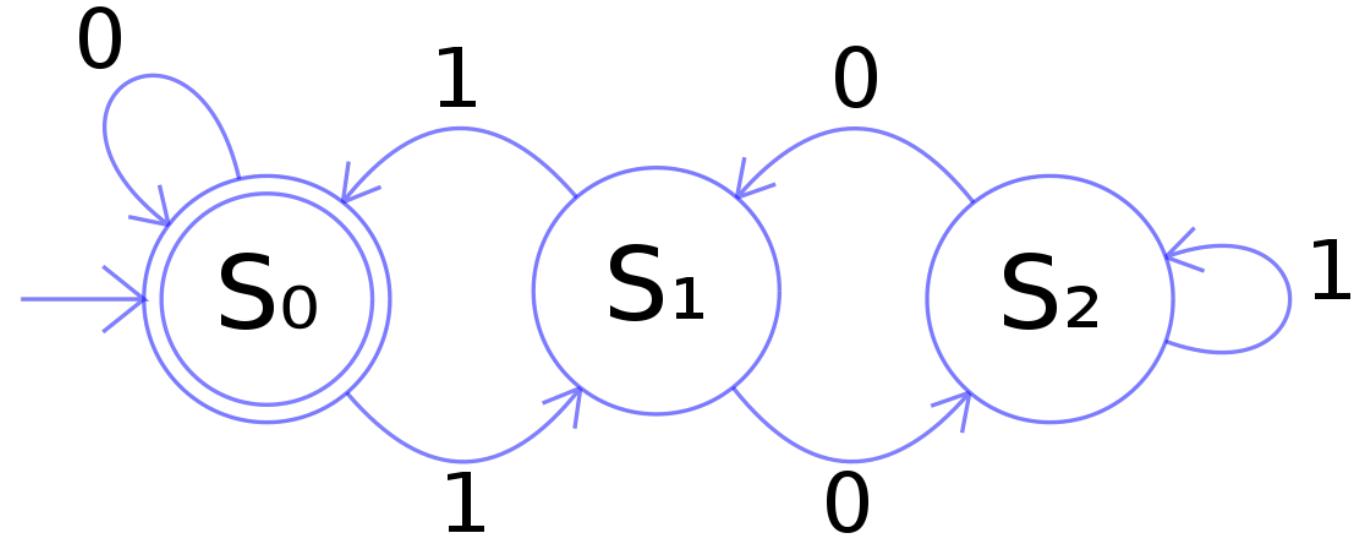
Time series data: time defines the occurrence of events



Genome sequence data: each sequence has a different meaning

# Finite Automata

---



How to use FNN to simulate a finite state automaton?

# Turing Machine

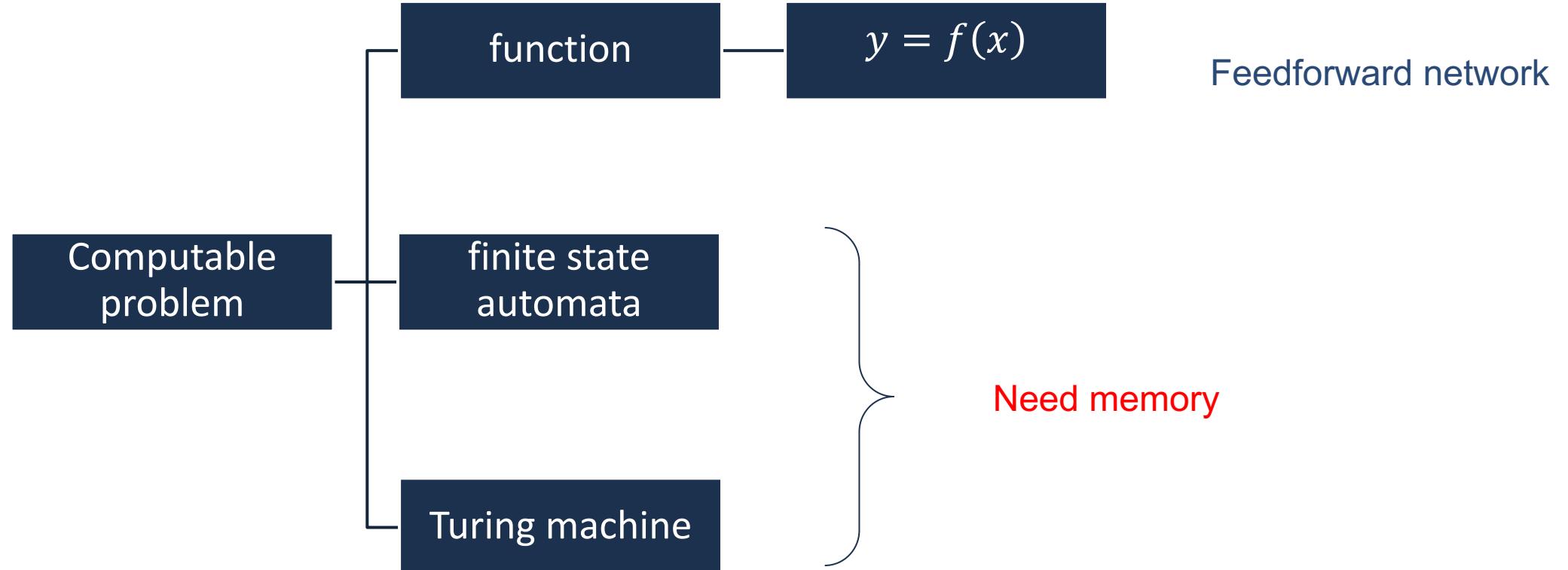
---

- ▶ An abstract mathematical model that can be used to simulate any calculable problem.



How to use FNN to simulate a Turing machine?

# Computable problem



How to add memory to the network?

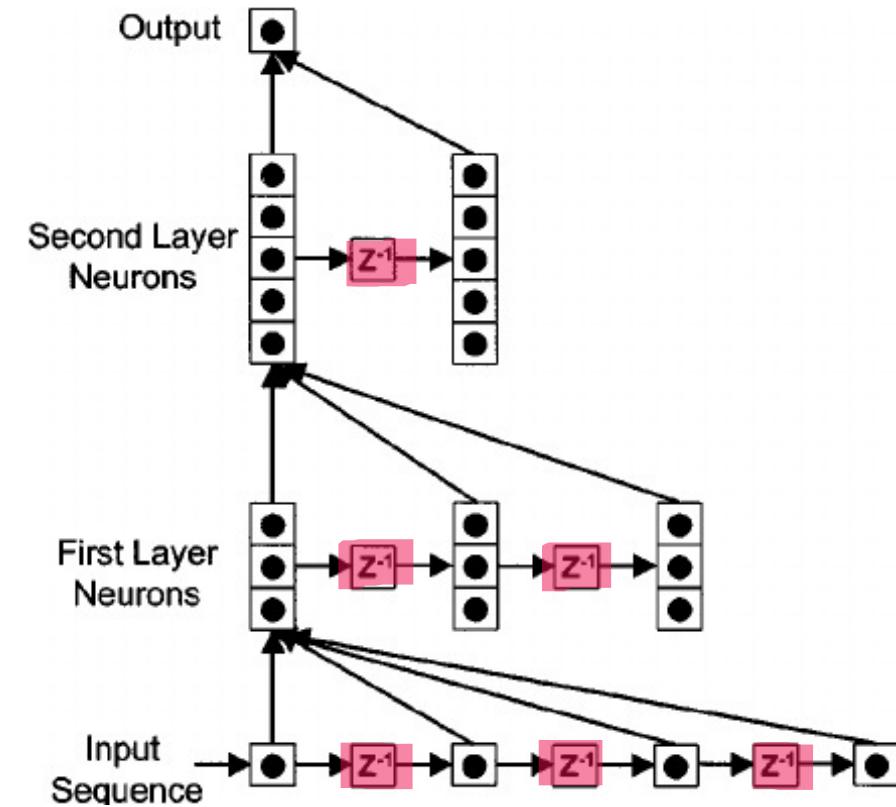
# How to add memory to the network?

## ► Time Delay Neural Network (TDNN)

- Establish an ***additional delay unit*** to store historical information of the network (can include input, output, hidden state, etc.)

$$\mathbf{h}_t^{(l)} = f(\mathbf{h}_t^{(l-1)}, \mathbf{h}_{t-1}^{(l-1)}, \dots, \mathbf{h}_{t-K}^{(l-1)})$$

- In this way, the feedforward network has the ability of short-term memory.



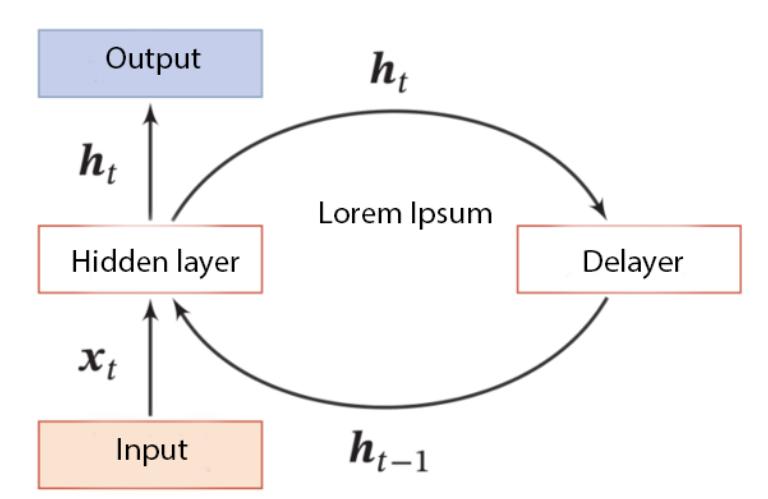
[https://www.researchgate.net/publication/12314435\\_Neural\\_system\\_identification\\_model\\_of\\_human\\_sound\\_localization](https://www.researchgate.net/publication/12314435_Neural_system_identification_model_of_human_sound_localization)

# Recurrent Neural Network

# Recurrent Neural Network (RNN)

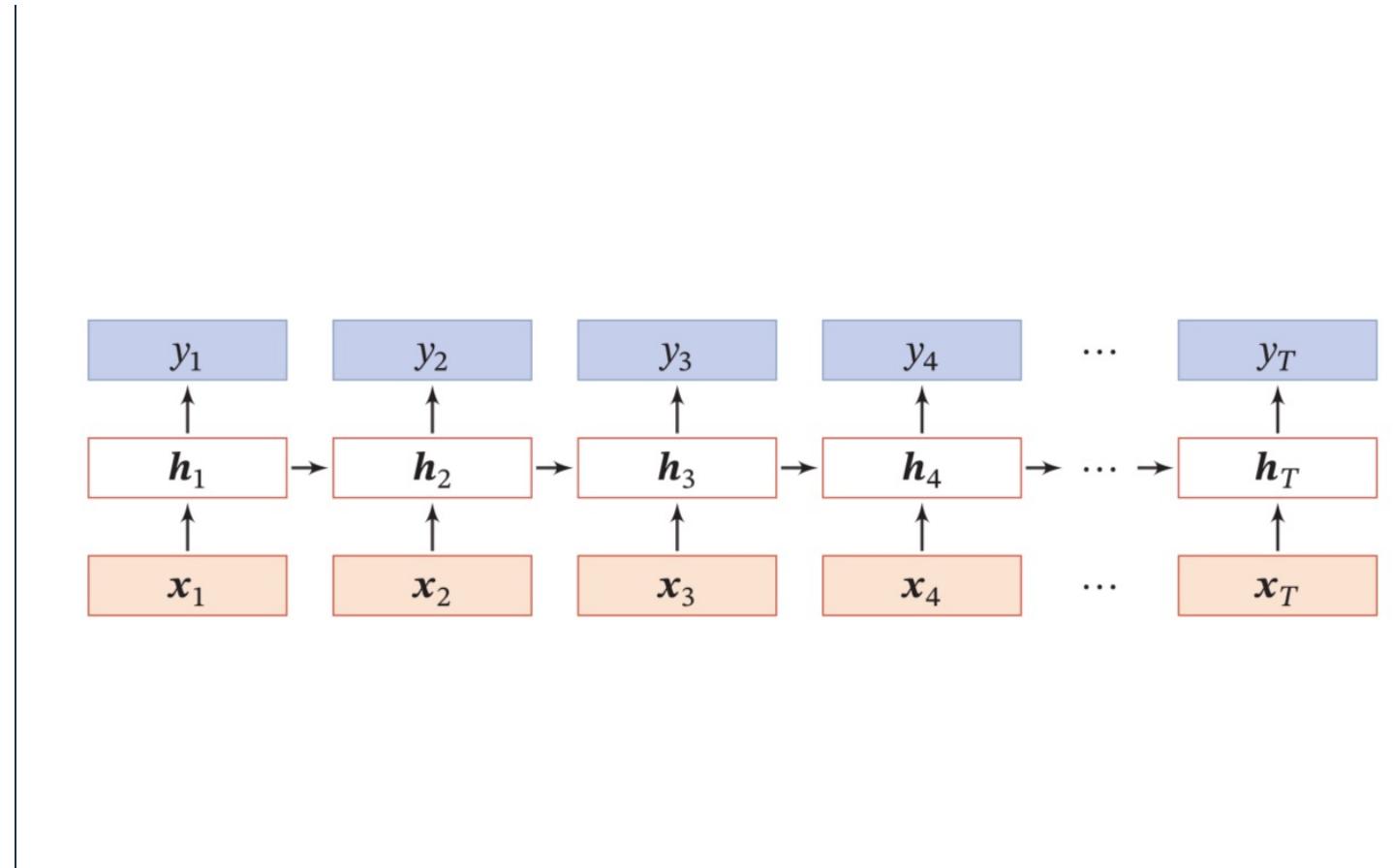
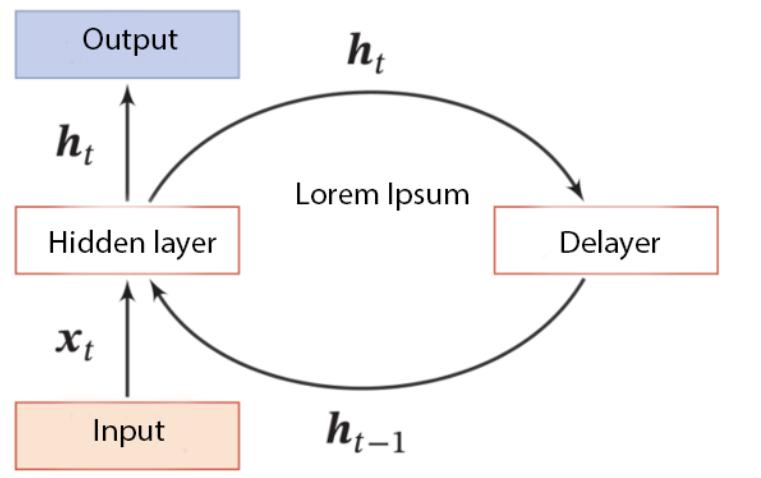
- ▶ RNN can process time series data of any length by using neurons with self-feedback.

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



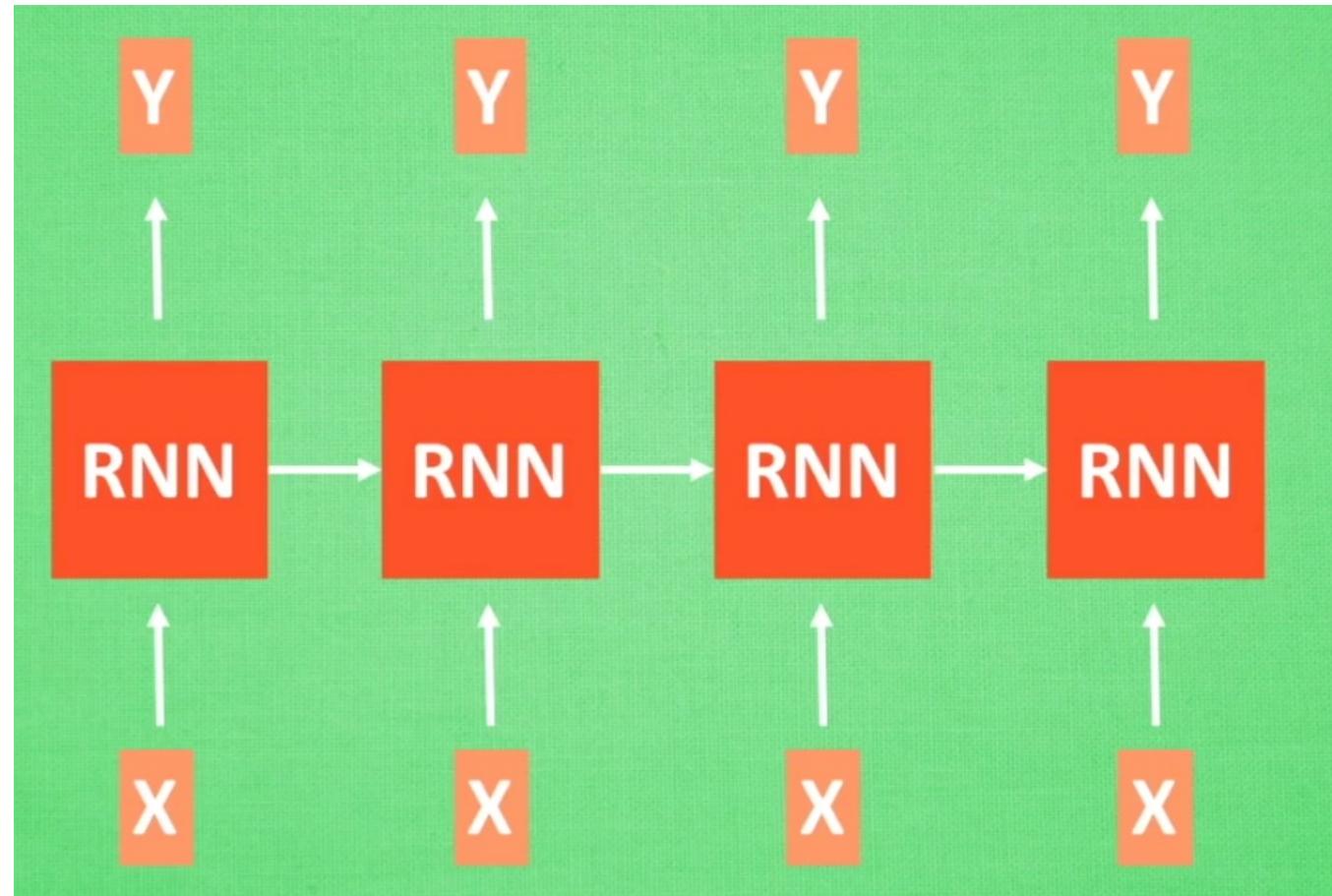
- ▶ RNNs are more in line with the structure of biological neural networks than feedforward neural networks .
- ▶ RNNs have been widely used in tasks such as speech recognition, language modeling, and natural language generation

# Expand by time



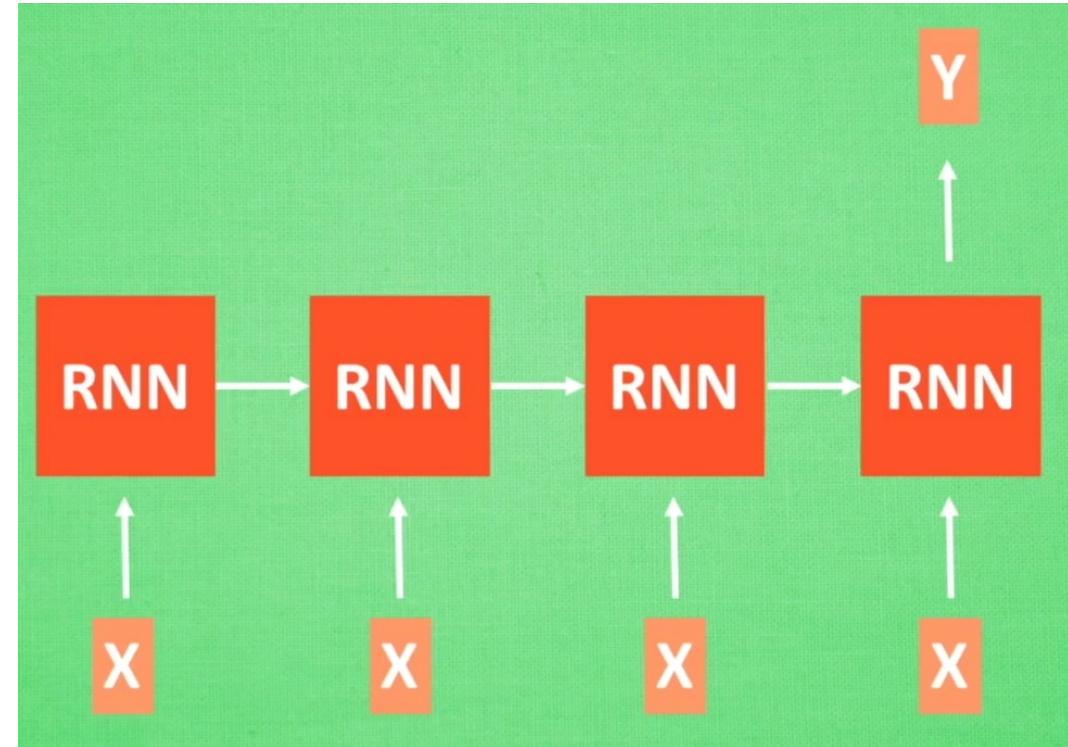
# Application of Recurrent Neural Network

- ▶ The form of RNN is not only this one, but its structure is very flexible.



# Application of Recurrent Neural Network

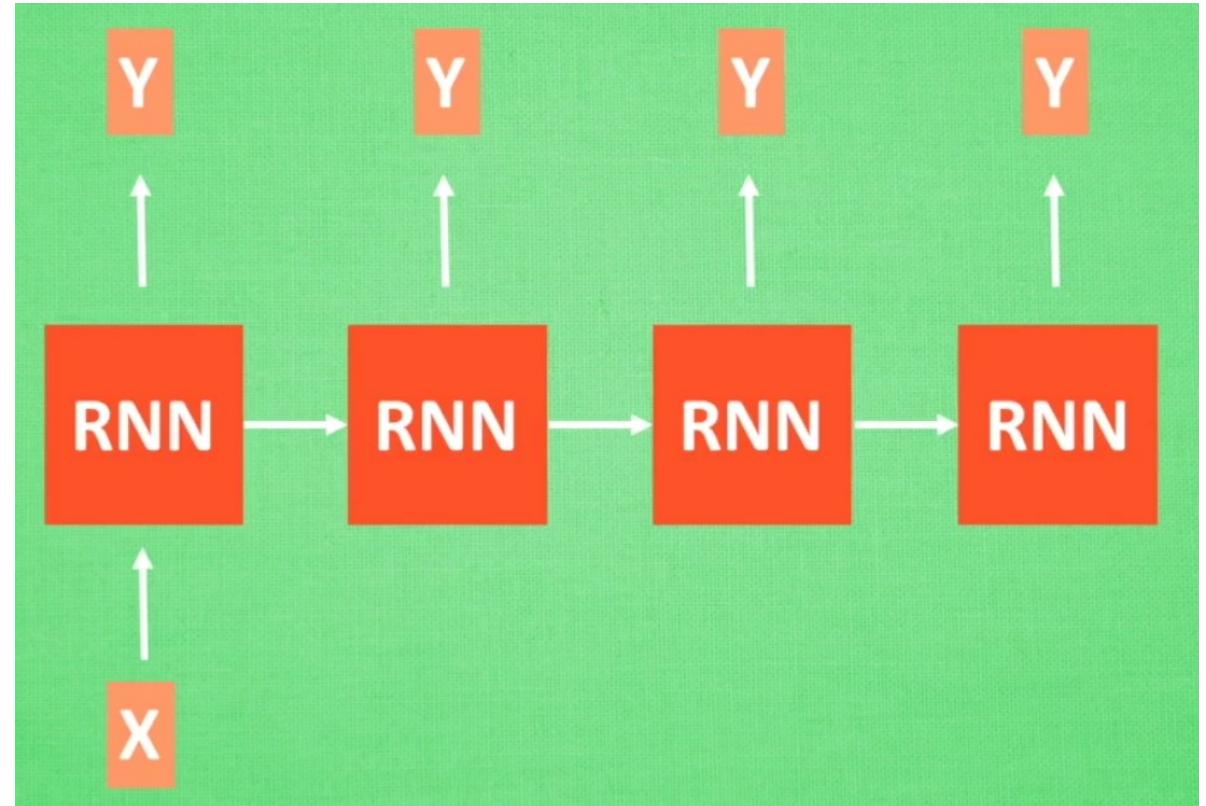
- ▶ To judge whether a person's words are positive or negative, we can use RNN that only outputs the judgment result at the last time point.



Many to one: Input is a vector sequence, but output is only one vector

# Application of Recurrent Neural Network

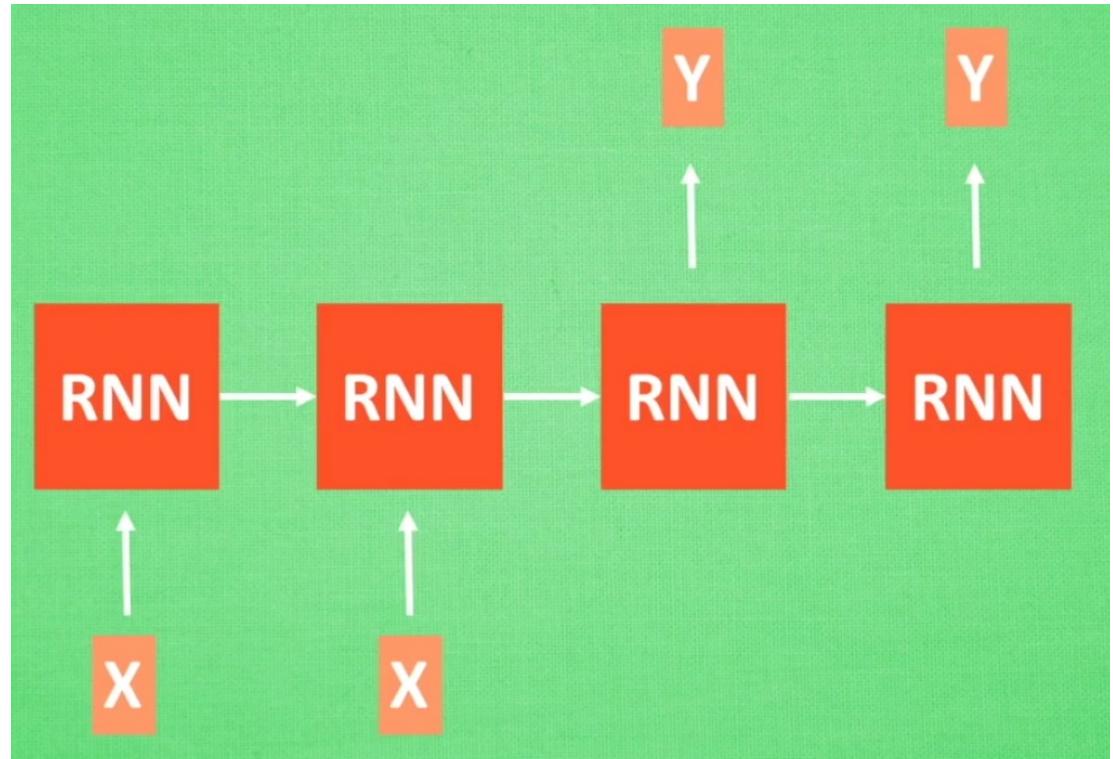
- ▶ Picture description RNN, we only need an X to replace the input picture, and then generate a paragraph describing the picture.



One to many: Input is only one vector, but output is a vector sequence

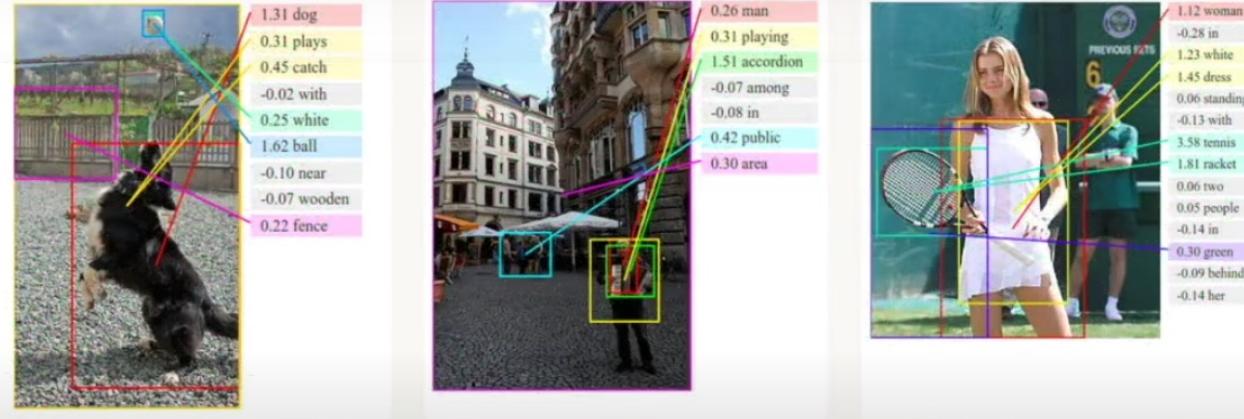
# Application of Recurrent Neural Network

- ▶ The RNN of language translation gives a paragraph of English, and then translates into Japanese.



Many to Many: Both input and output are both sequences

# More applications...



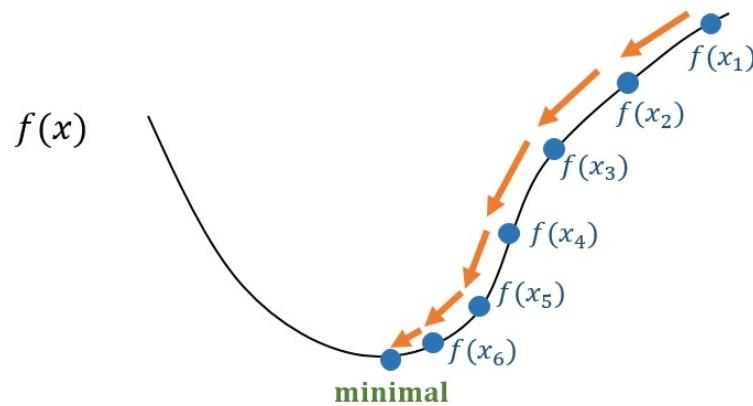
```
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
```

<https://openai.com/blog/musenet/>

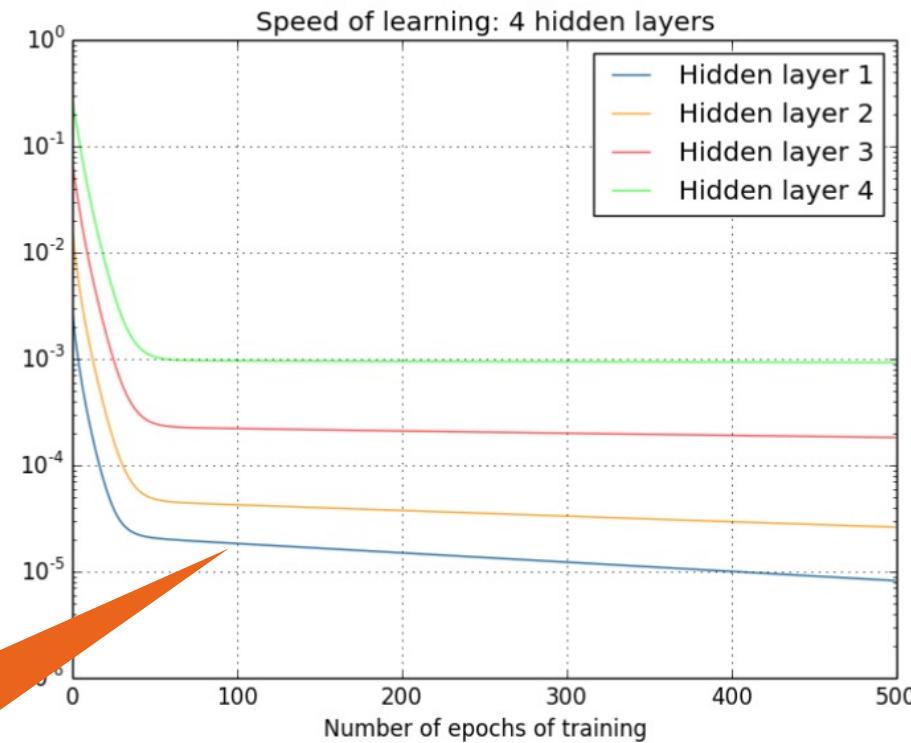
## GVP and GEP

# Gradient vanishing problem (GVP)

- As the number of network layers increases, the gradient update will decrease toward exponential decay
- Our weights cannot be updated, which eventually leads to training failure



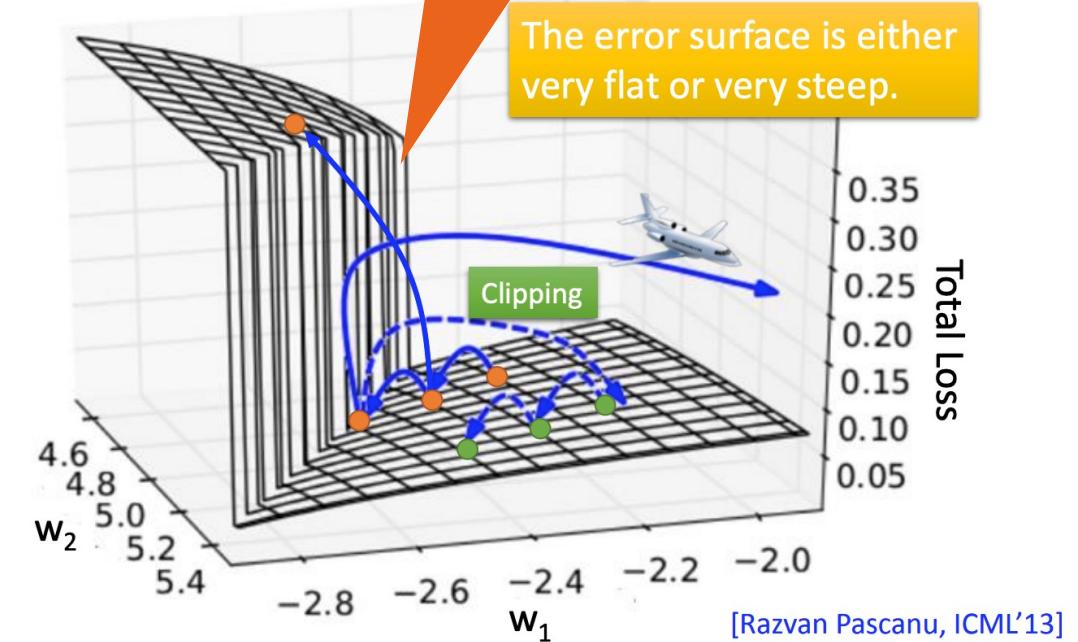
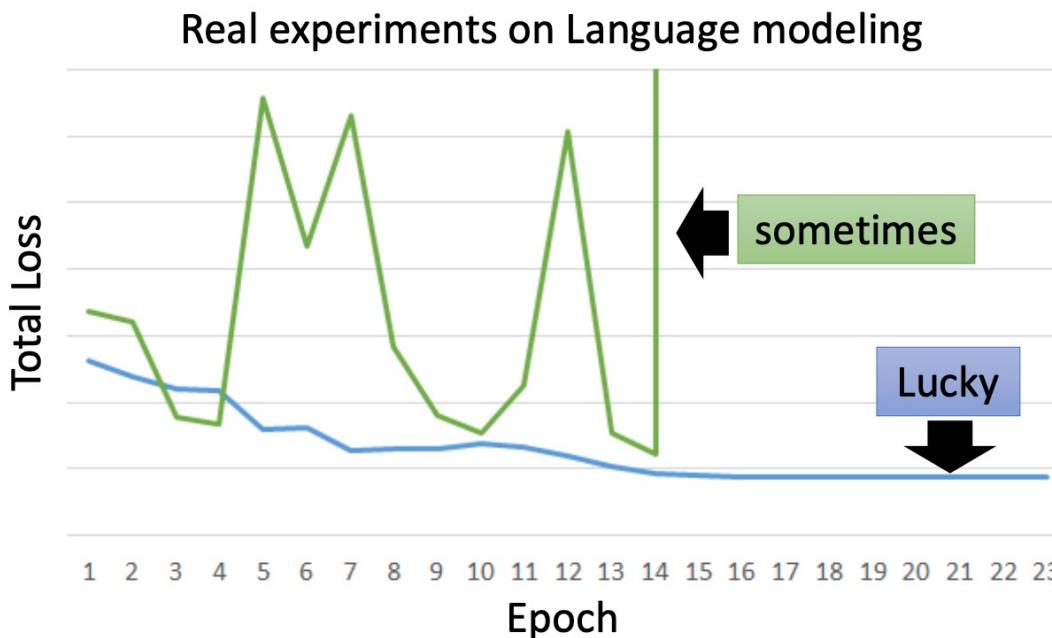
The update speed of the 4<sup>th</sup> hidden layer is two orders of magnitude slower than that of the 1<sup>st</sup> hidden layer



# Gradient exploding problem (GEP)

- As the number of network layers increases, gradient updates will increase exponentially
- The gradient is too large, thereby greatly updating network parameters, causing network instability

When encountering gradient wall, calculate the gradient at a certain point on the wall, the gradient will increase instantly and point to an undesirable location



# What is the consequence of GVP or GEP?

---

- ▶ The disappearance of the gradient will cause the network weights of the previous layer in our neural network to be unable to be updated, and thus stop learning
- ▶ Gradient explosion will make learning unstable, and parameter changes too large make it impossible to obtain optimal parameters
- ▶ Makes the network unable to learn well from the training data
  - ▶ E.g., recurrent neural networks(RNN) cannot learn on long input data sequences
  - ▶ E.g., the deep multi-layer perceptron network cannot update the weight because the weight value is NaN

# What is the reason behind GVP or GEP?

$$f_{i+1} = f(f_i * w_{i+1})$$

$$w \leftarrow w + \Delta w, \quad \Delta w = -\alpha \frac{\partial Loss}{w}$$

$$\begin{aligned}\Delta w_1 &= \frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial f_4} * \frac{\partial f_4}{\partial f_3} * \frac{\partial f_3}{\partial f_2} * \frac{\partial f_2}{\partial w_2} \\ &= \frac{\partial Loss}{\partial f_n} * \frac{\partial f_n}{\partial f_{n-1}} * \dots * \frac{\partial f_2}{\partial w_2}\end{aligned}$$

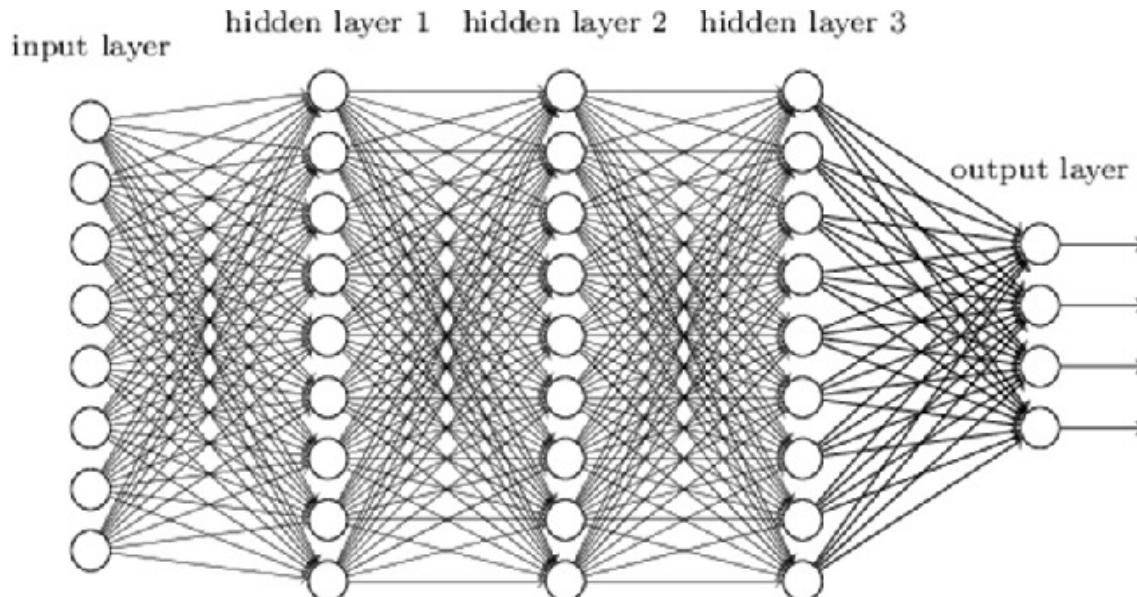


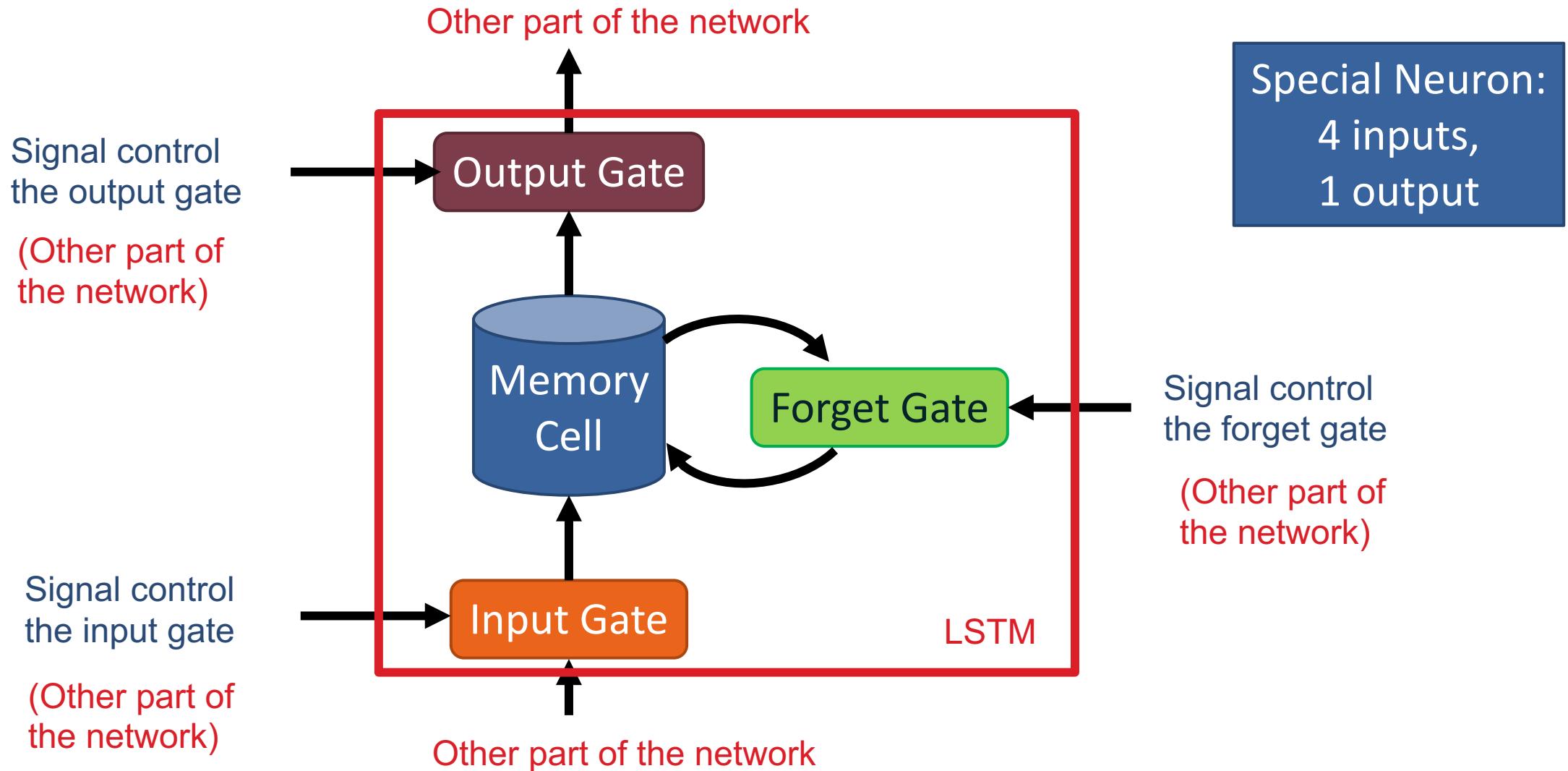
Fig. 2. An example of deep neural network with five layers

If this part is greater than 1, then when the number of layers increases, the final gradient update will increase exponentially, that is, gradient explosion occurs;

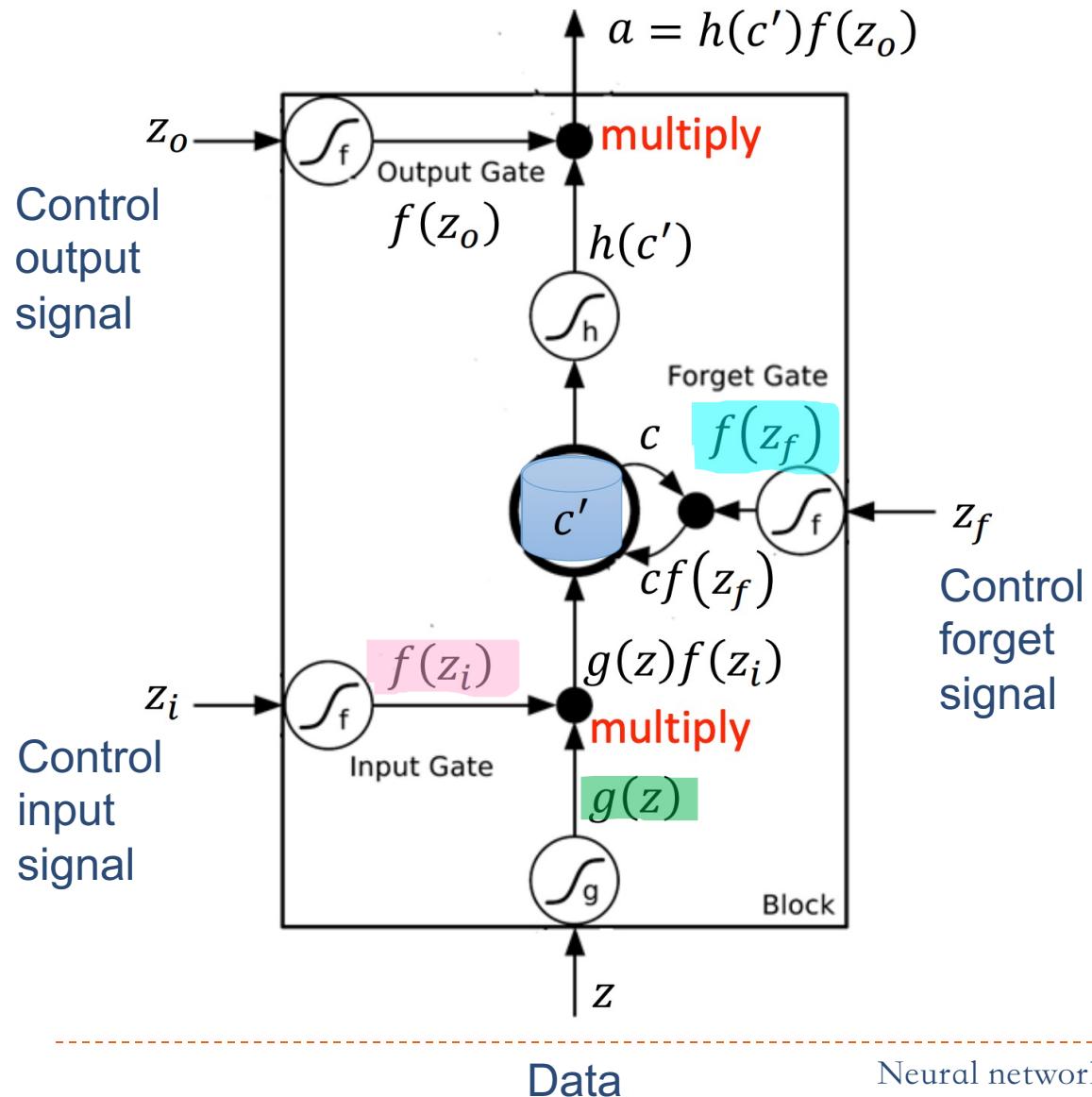
If this part is less than 1, then as the number of layers increases, the obtained gradient update information will decay exponentially, that is, the gradient disappears

# Long Short-term Memory

# Long Short-term Memory (LSTM)



# Long Short-term Memory (LSTM)



Activation function  $f$  is  
usually a sigmoid function

Between 0 and 1

Mimic open and close gate

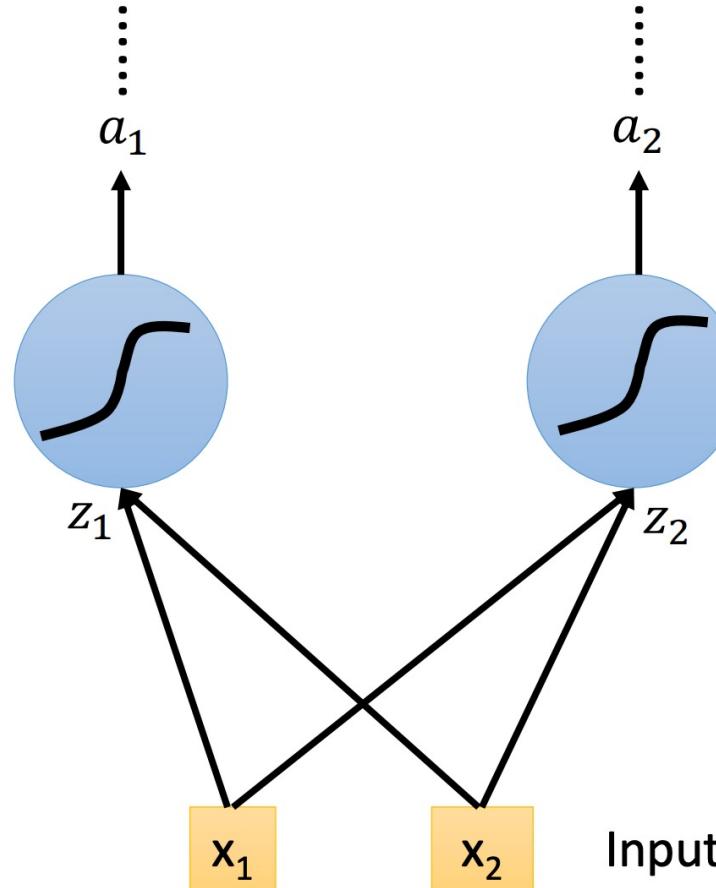
$$c' = g(z)f(z_i) + c f(z_f)$$

Current input

Previous input

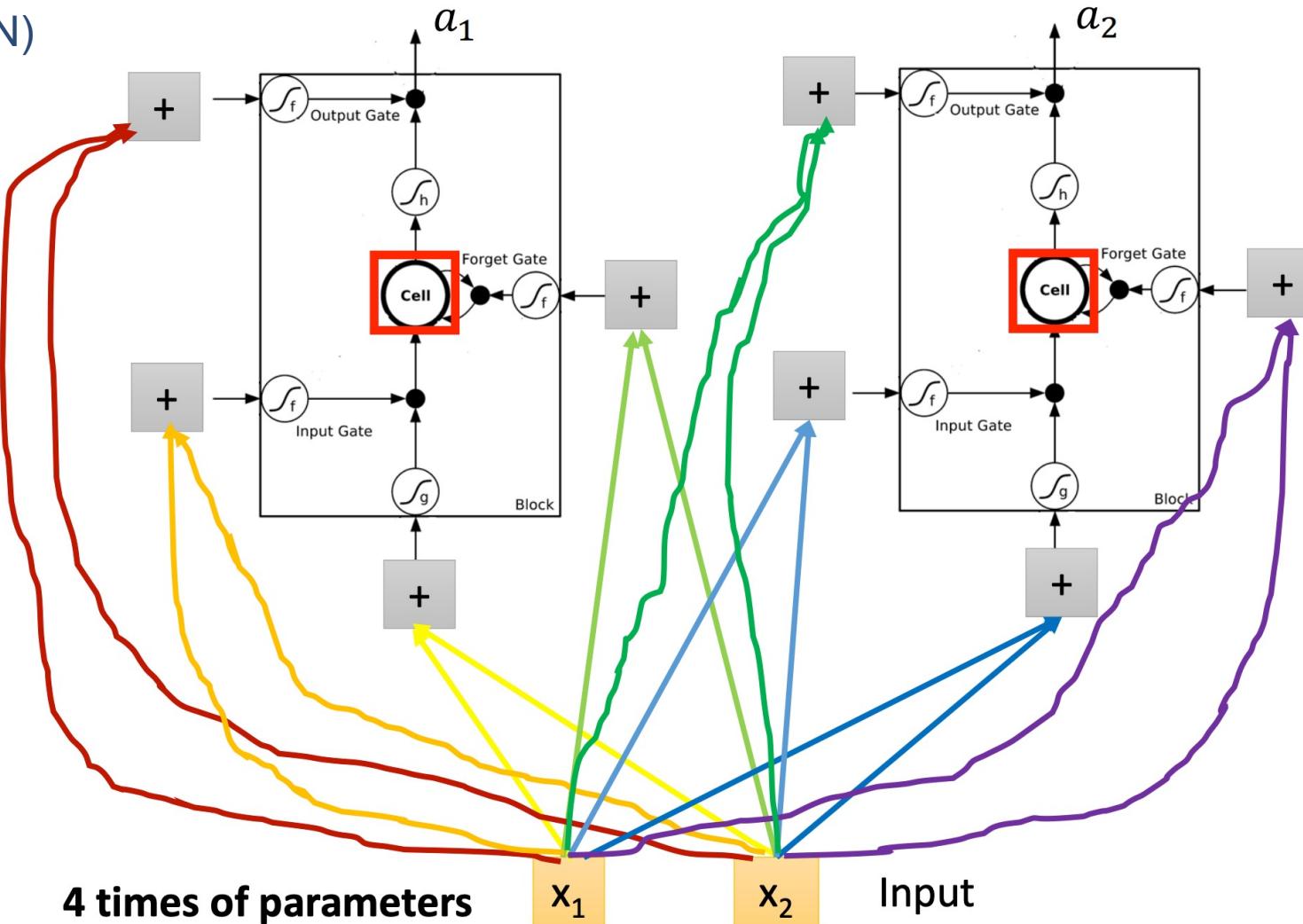
# Original Network (Linear format)

---



# Replace the neurons with LSTM

(LSTM format RNN)



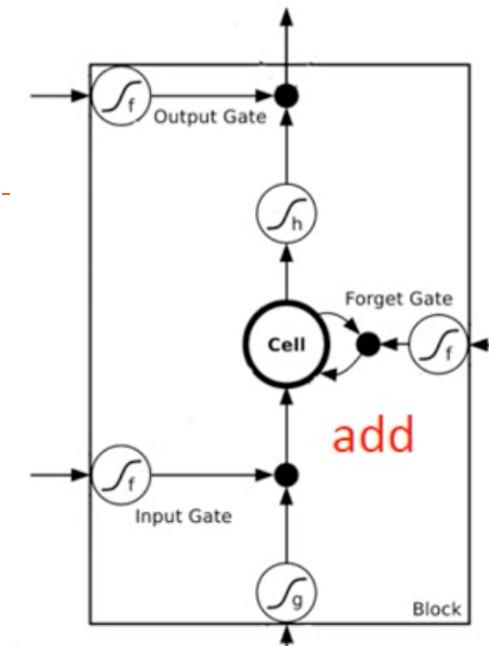
# LSTM to overcome GVP

## ► Long Short-term Memory

### ► Can deal with gradient vanishing

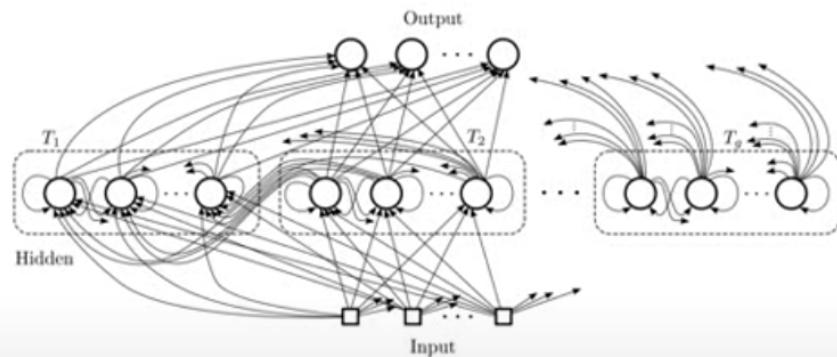
- Memory and input are added

- The influence never disappears unless gate is closed



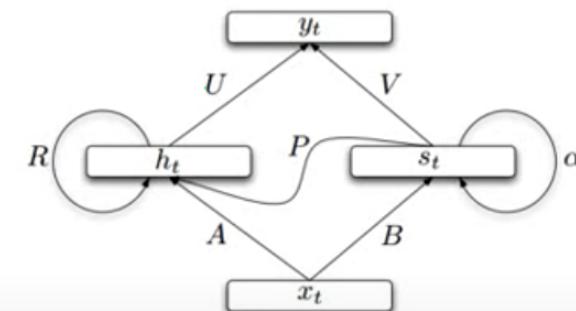
## ► Other techniques

### Clockwise RNN



[Jan Koutnik, JMLR'14]

### Structurally Constrained Recurrent Network (SCRN)

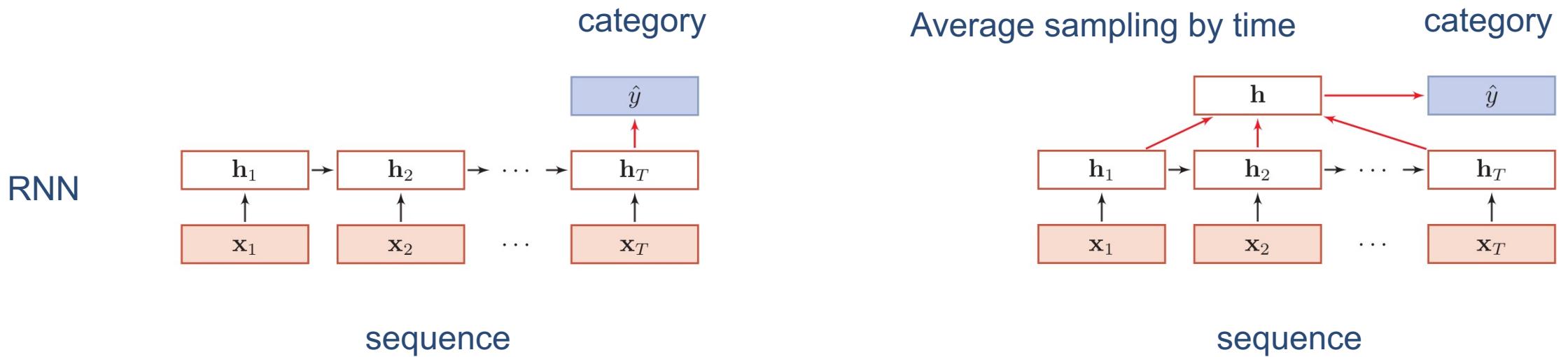


[Tomas Mikolov, ICLR'15]

# RNN applied to Machine Learning

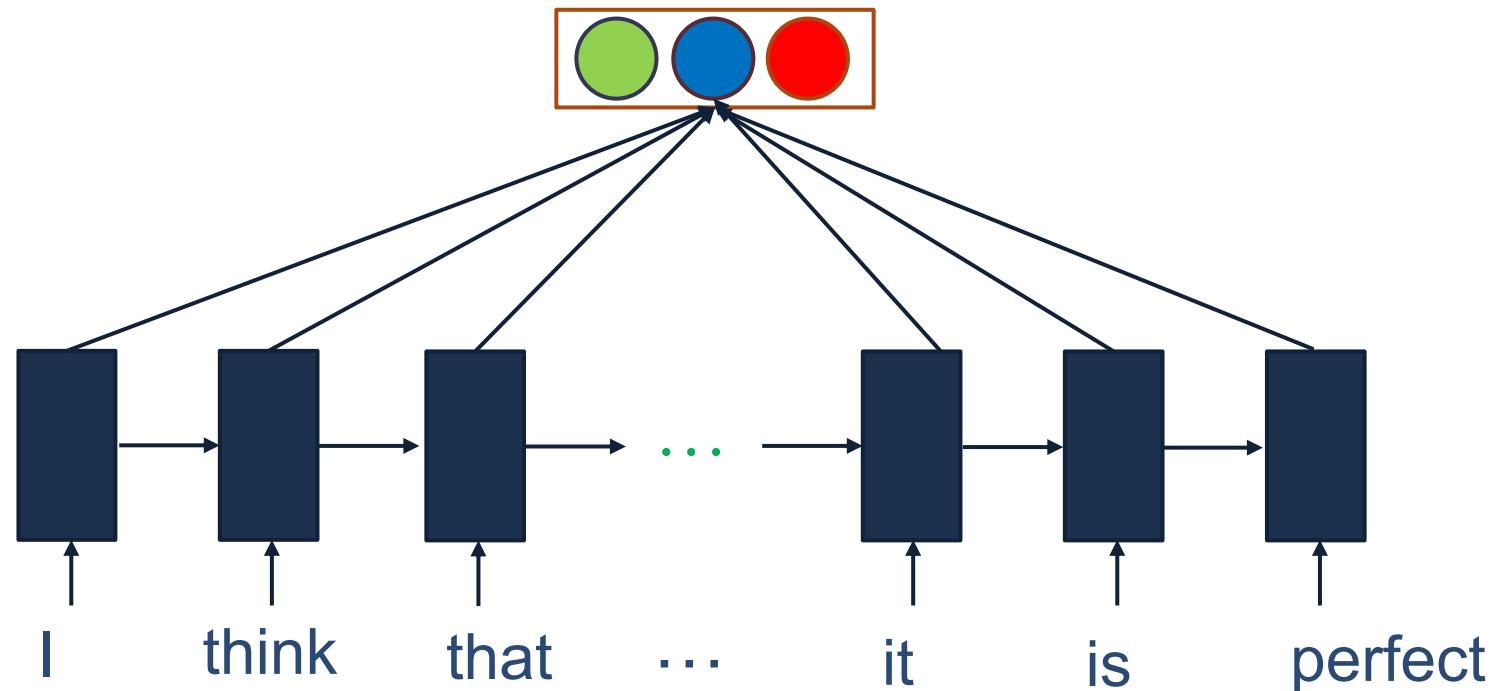
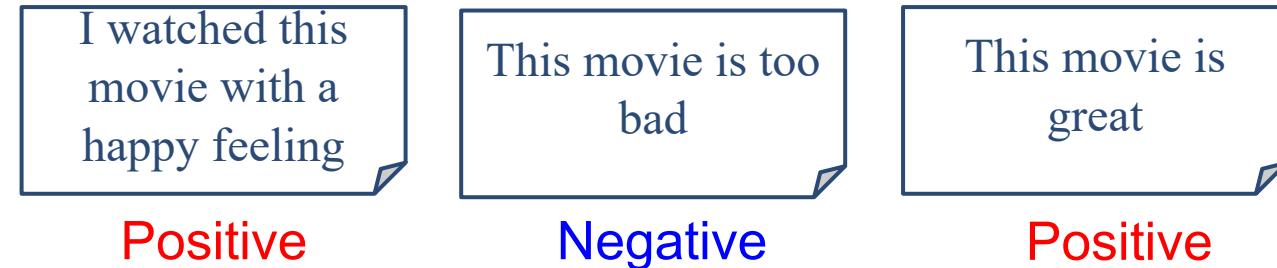
# Sequence

## ► From sequence to category



# From sequence to category

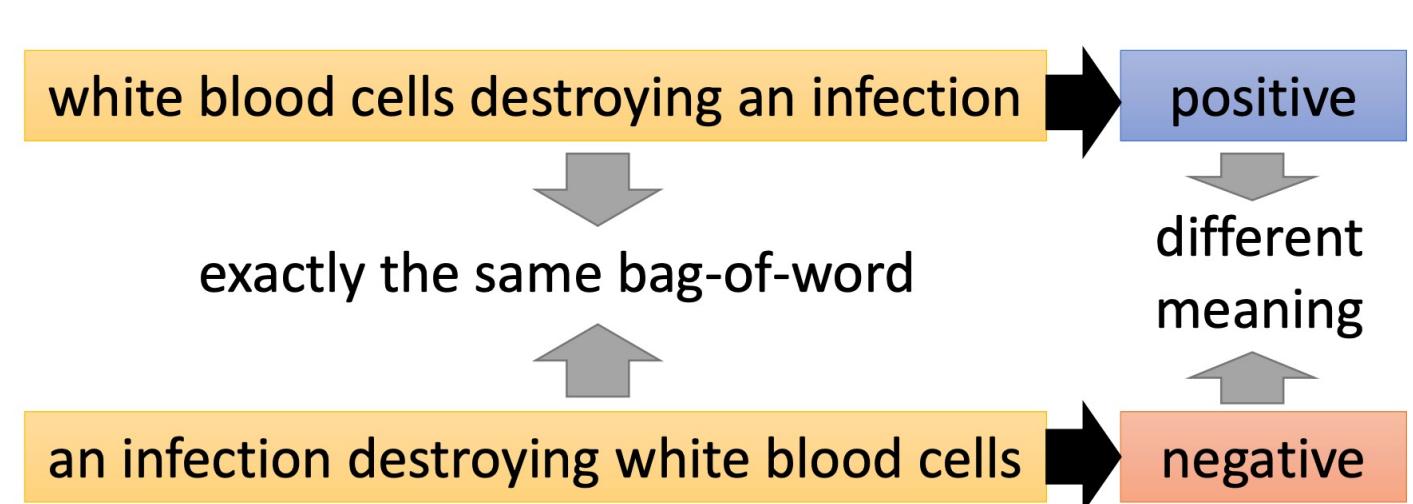
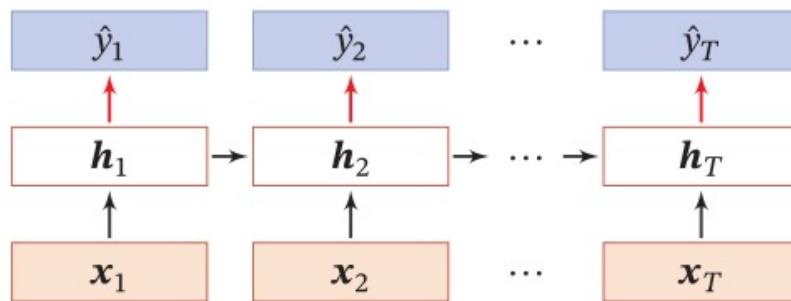
## ► Sentiment classification



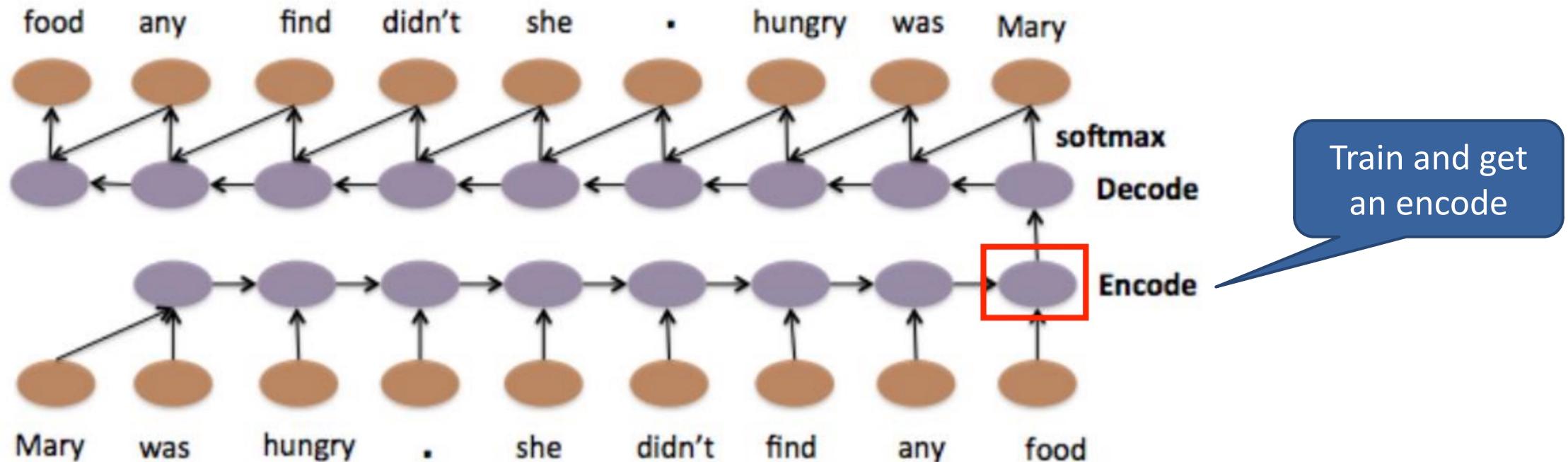
# From sequence to sequence

- Auto-encoder - Text

- To understand the meaning of a word sequence, the order of the words can not be ignored.



# From sequence to sequence



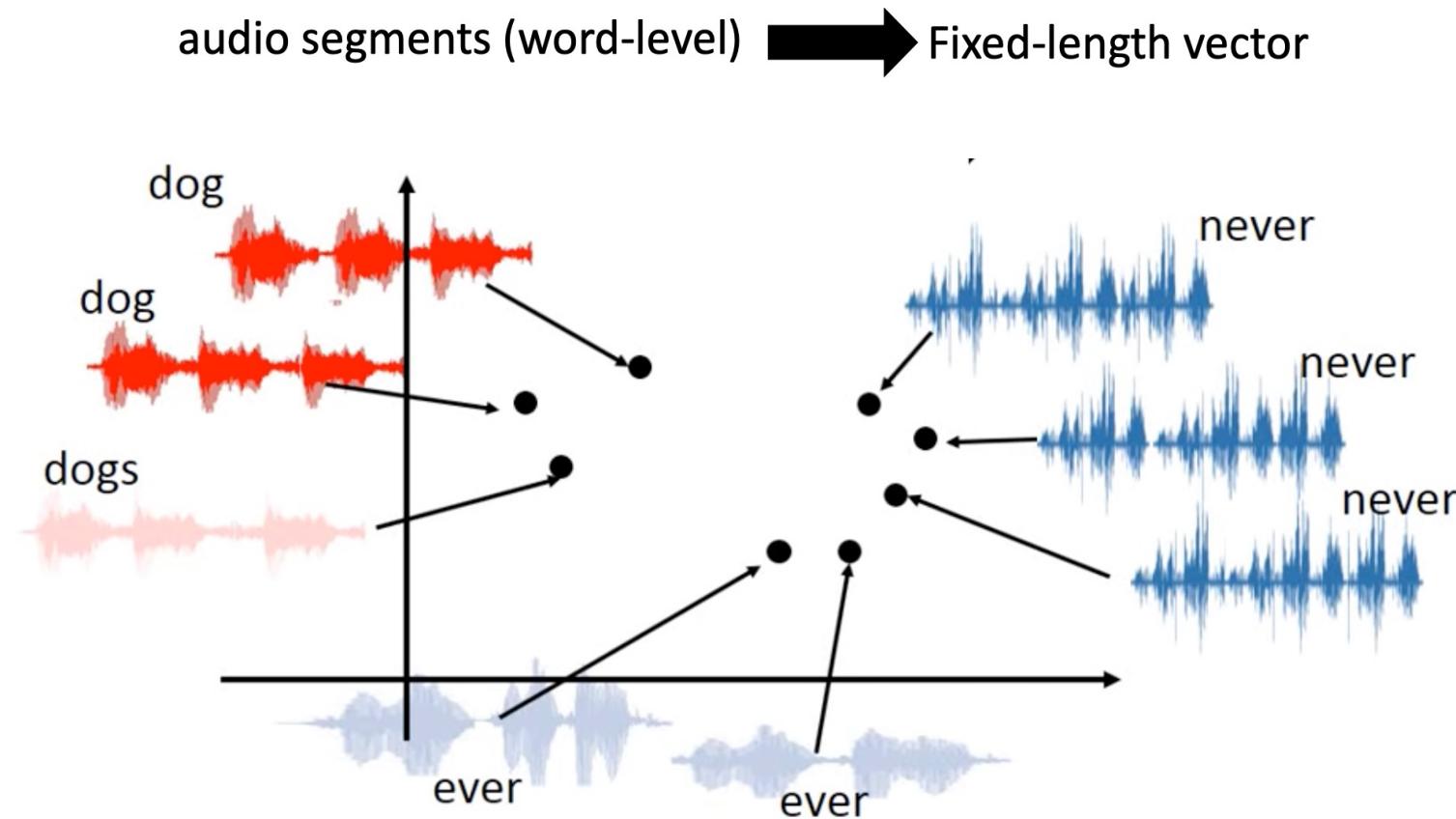
<https://arxiv.org/pdf/1506.01057.pdf>

# From sequence to sequence

## ► Auto-encoder – Speech

- Dimension reduction for a sequence with variable length

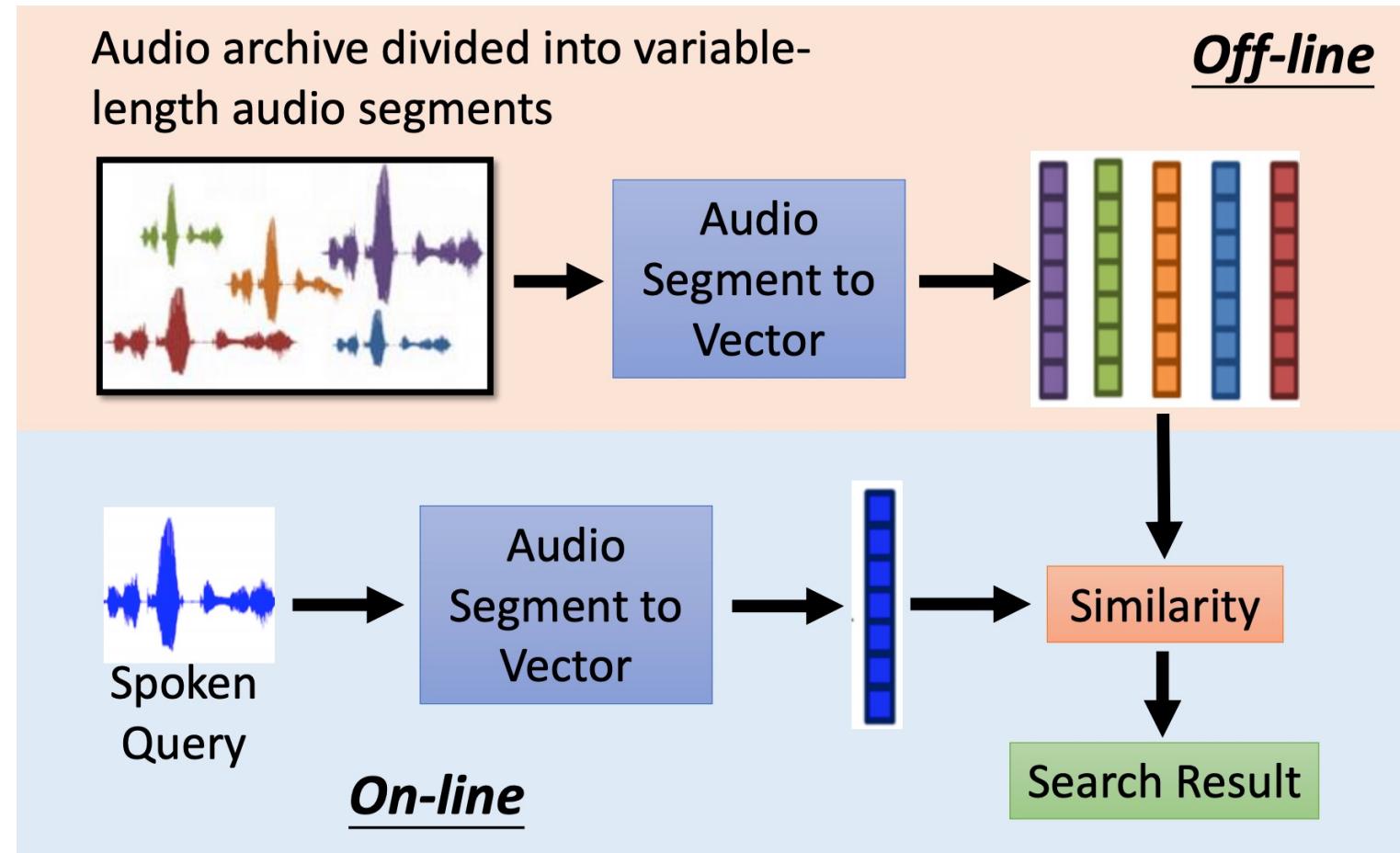
<https://arxiv.org/pdf/1603.00982.pdf>



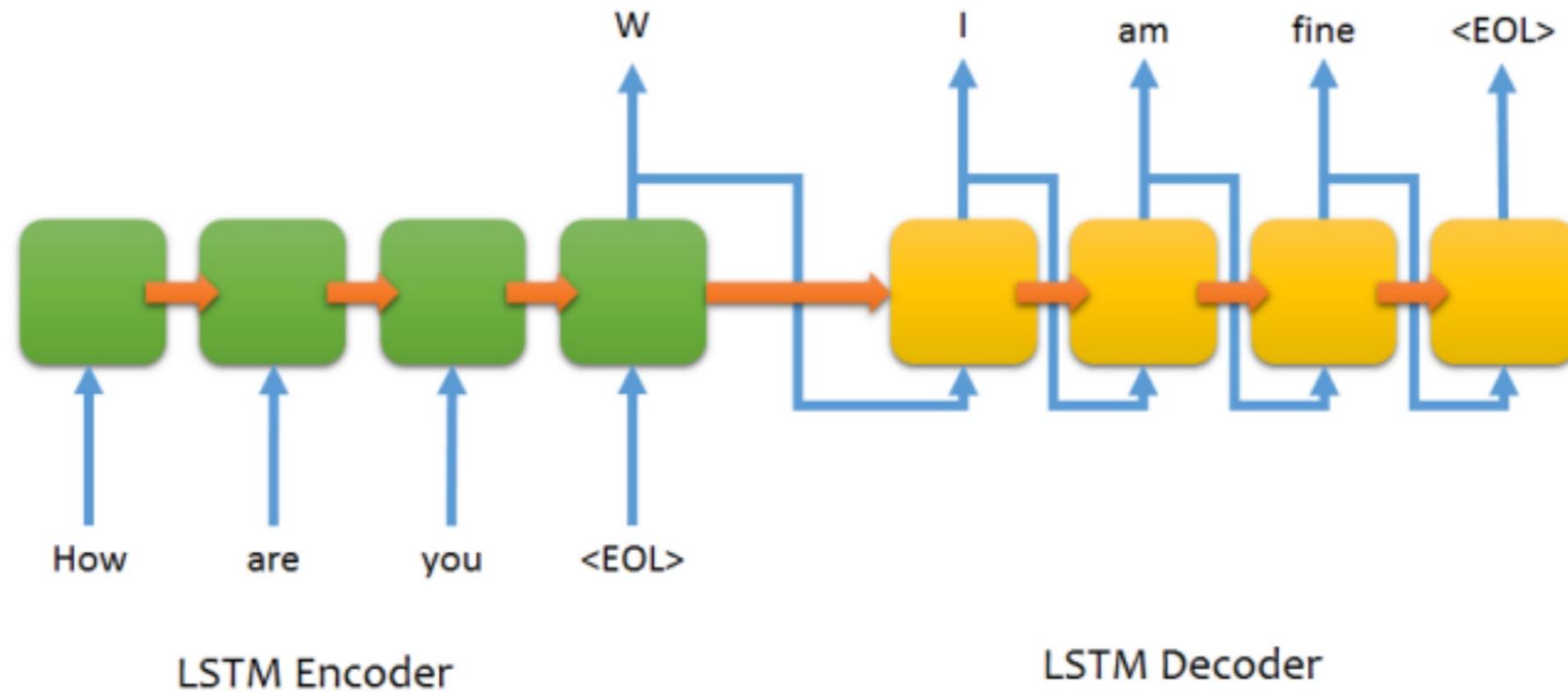
# From sequence to sequence

<https://arxiv.org/pdf/1603.00982.pdf>

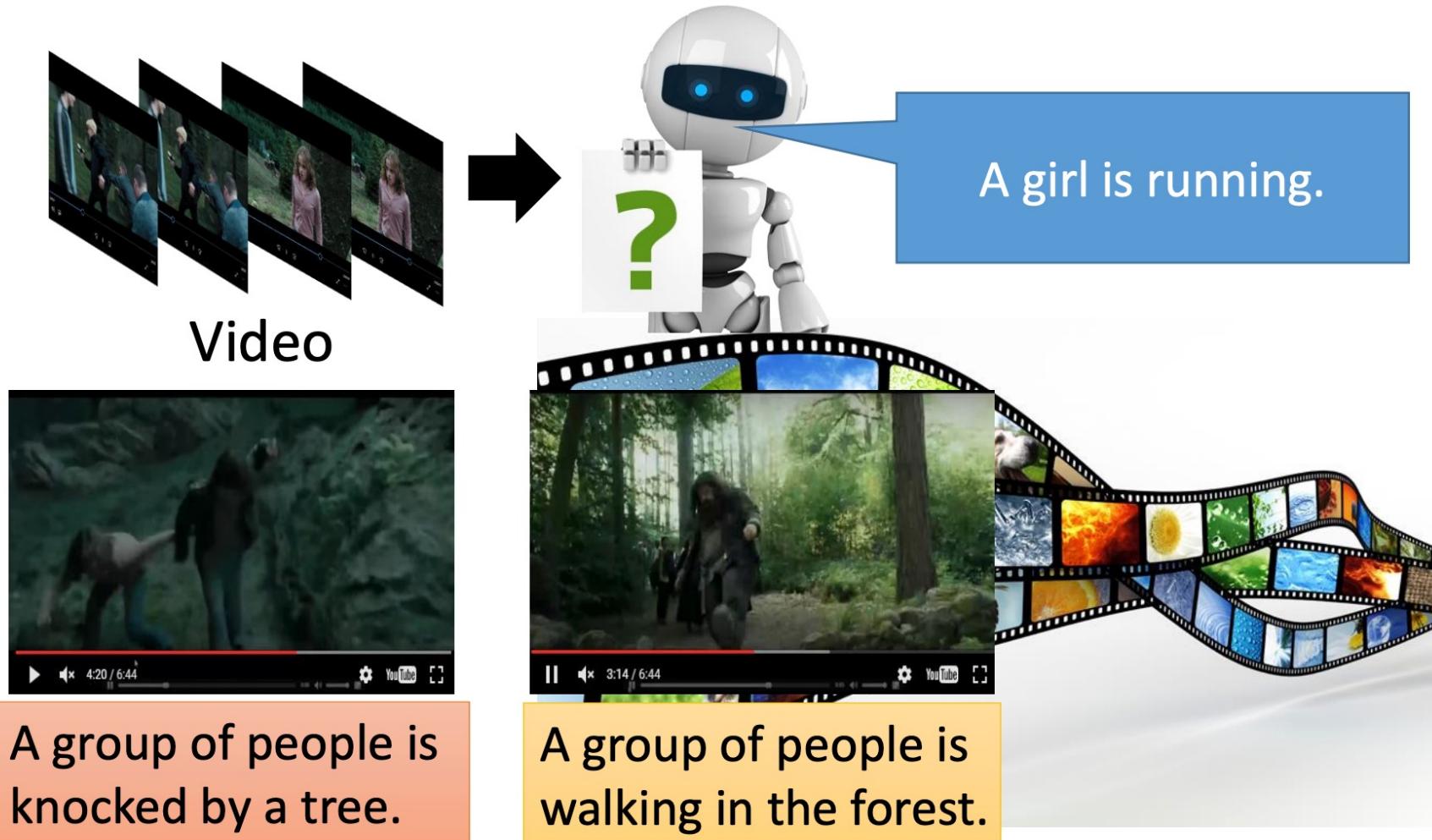
## ► Auto-encoder – Speech



# Application: Chat-bot

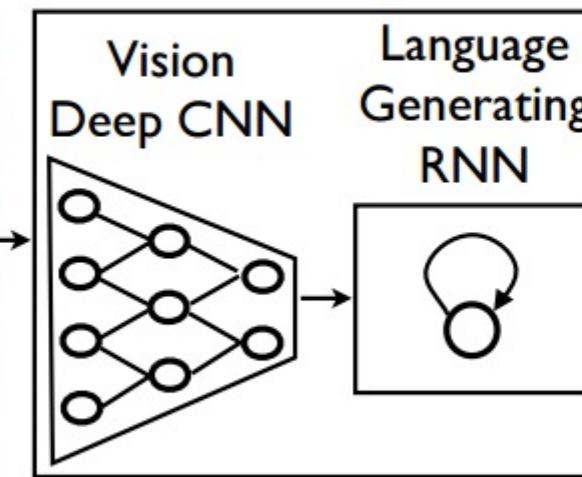


# Application: Video Caption Generation



# Add notes to pictures

---



**A group of people  
shopping at an  
outdoor market.**

**There are many  
vegetables at the  
fruit stand.**

# Add notes to pictures



Describes without errors

Describes with minor errors

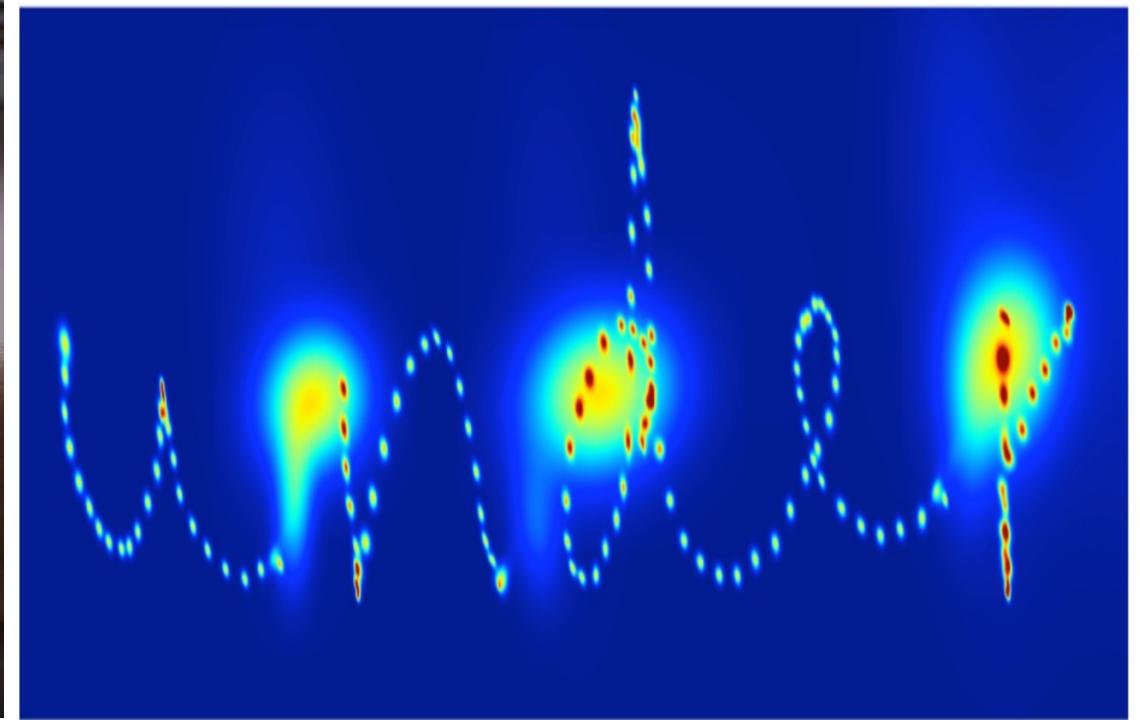
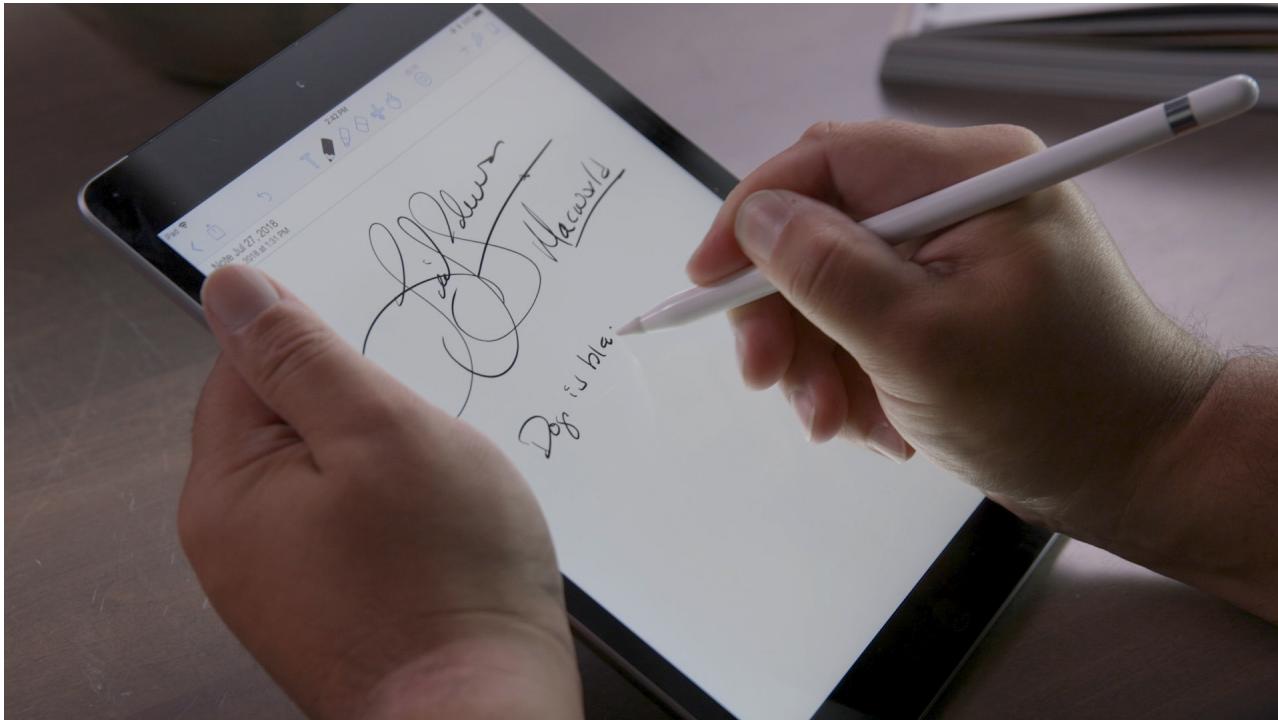
Somewhat related to the image

Unrelated to the image

Figure 5. A selection of evaluation results, grouped by human rating.

# Handwriting prediction

- The writing track is regarded as a series of points
- Calculate the trajectory for the next second based on the previous input and offset



# Recurrent neural network summary

---

- ▶ **Advantage:**

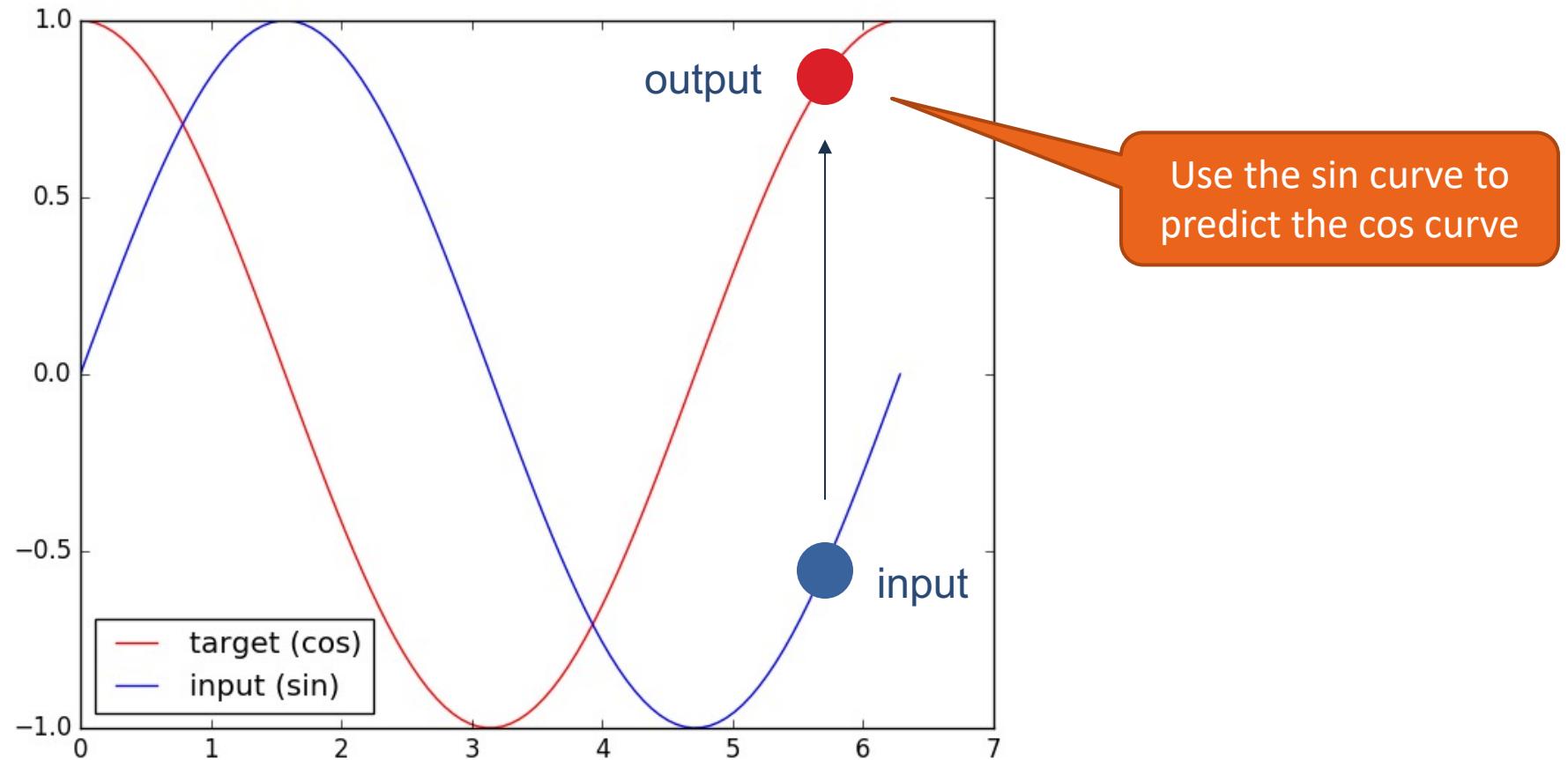
- ▶ Introduce memory

- ▶ **Disadvantage:**

- ▶ Memory capacity problem
  - ▶ Parallel ability

# Build an RNN

- ▶ Use RNN to predict time series



# Build an RNN

## ▶ Torch

▶ <http://torch.ch/>

▶ Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language.



## ▶ Hyper Parameters

```
1 import torch
2 from torch import nn
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # Hyper Parameters
7 TIME_STEP = 10      # rnn time step
8 INPUT_SIZE = 1       # rnn input size
9 LR = 0.02           # learning rate
10
```

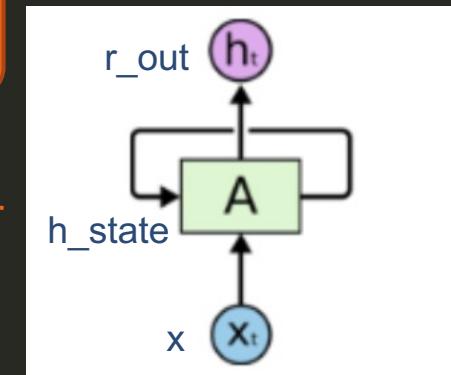
1 means every time step we give 1 point value of sin curve into RNN

# Build an RNN

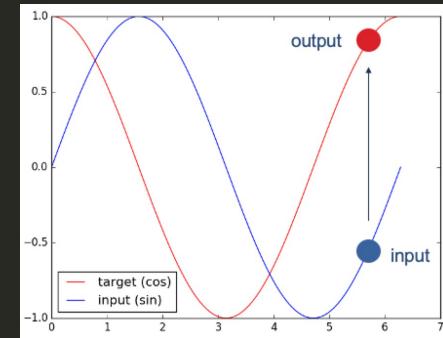
```
20 | class RNN(nn.Module):
21 |     def __init__(self):
22 |         super(RNN, self).__init__()
23 |
24 |         self.rnn = nn.RNN(
25 |             input_size=INPUT_SIZE,
26 |             hidden_size=32,      # rnn hidden unit
27 |             num_layers=1,        # number of rnn layer
28 |             batch_first=True,   # input & output will has batch size as 1s dimension. e.g. (batch, time_step, input_size)
29 |         )
30 |         self.out = nn.Linear(32, 1)
31 |
32 |
33 |     def forward(self, x, h_state):
34 |         # x (batch, time_step, input_size)
35 |         # h_state (n_layers, batch, hidden_size)
36 |         # r_out (batch, time_step, hidden_size)
37 |         r_out, h_state = self.rnn(x, h_state)
38 |
39 |         outs = []    # save all predictions
40 |         for time_step in range(r_out.size(1)):    # calculate output for each time step
41 |             outs.append(self.out(r_out[:, time_step, :]))
42 |         return torch.stack(outs, dim=1), h_state
43 |
44 |         # instead, for simplicity, you can replace above codes by follows
45 |         # r_out = r_out.view(-1, 32)
46 |         # outs = self.out(r_out)
47 |         # outs = outs.view(-1, TIME_STEP, 1)
48 |         # return outs, h_state
49 |
50 |         # or even simpler, since nn.Linear can accept inputs of any dimension
51 |         # and returns outputs with same dimension except for the last
52 |         # outs = self.out(r_out)
53 |         # return outs
54 |
55 |         rnn = RNN()
56 |         print(rnn)
```

Larger number of layer could increase the accuracy but sacrifice the computation time by involving huge amount of computation

Here we use Linear format RNN.  
You can also use LSTM format as:  
`self.out = nn.LSTM()`



Save all predictions



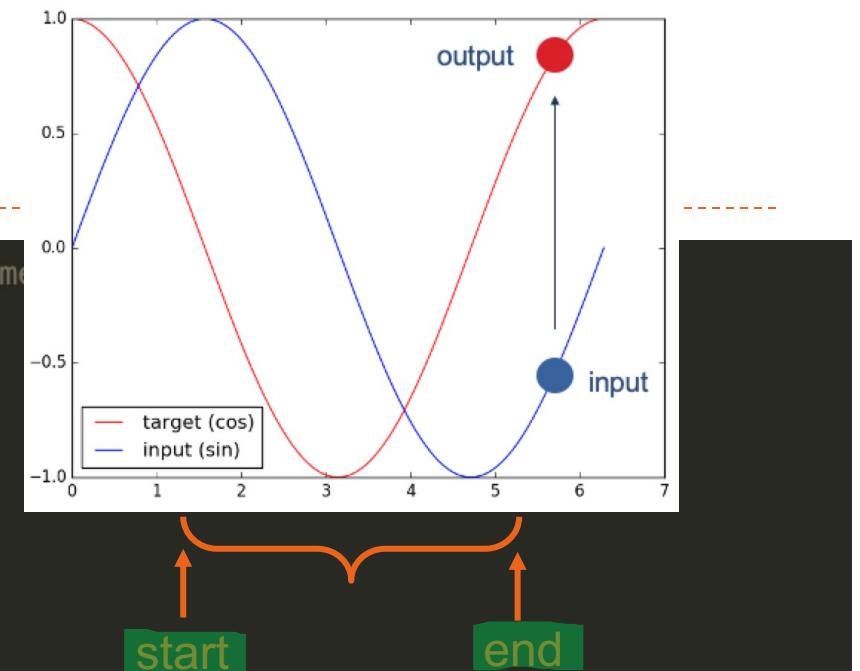
# Build an RNN

---

- ▶ Train idea:
  - ▶ Use  $x$  as the input sin value, and then  $y$  as the output cos value you want to fit.
  - ▶ Because the two curves (sin, cos) have a certain relationship, we can use sin to predict cos
  - ▶ RNN will understand their relationship and use the parameters inside to analyze how the points on the sin curve at this moment correspond to the points on the cos curve.

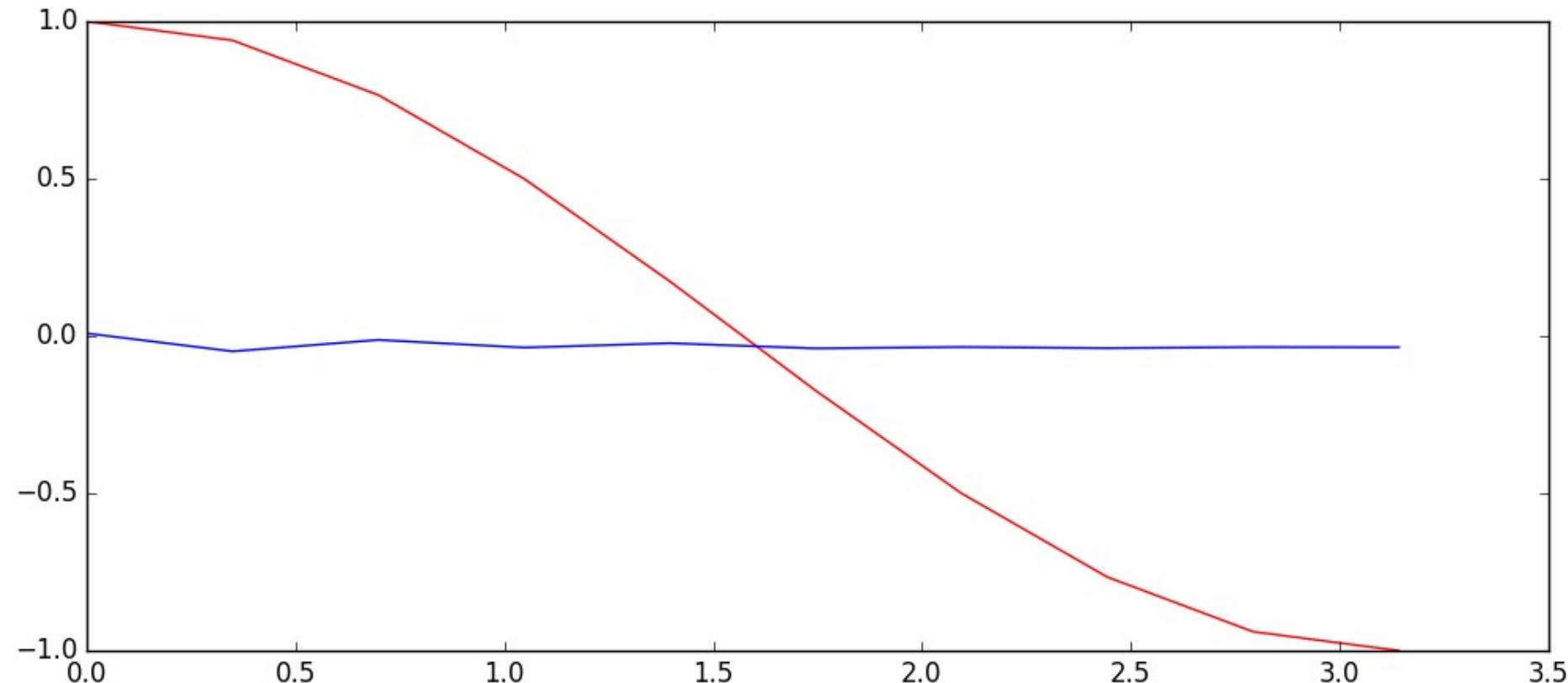
# Build an RNN

```
57
58     optimizer = torch.optim.Adam(rnn.parameters(), lr=LR)    # optimize all cnn parameters
59     loss_func = nn.MSELoss()
60
61     h_state = None      # for initial hidden state
62
63     plt.figure(1, figsize=(12, 5))
64     plt.ion()           # continuously plot
65
66     for step in range(100):
67         start, end = step * np.pi, (step+1)*np.pi    # time range
68         # use sin predicts cos
69         steps = np.linspace(start, end, TIME_STEP, dtype=np.float32, endpoint=False) # float32 for converting torch FloatTensor
70         x_np = np.sin(steps)
71         y_np = np.cos(steps)
72
73         x = torch.from_numpy(x_np[np.newaxis, :, np.newaxis])    # shape (batch, time_step, input_size)
74         y = torch.from_numpy(y_np[np.newaxis, :, np.newaxis])    # Encapsulate x_np into a three-dimensional vector
75
76         prediction, h_state = rnn(x, h_state)    # rnn output
77         # !! next step is important !!
78         h_state = h_state.data                  # repack the hidden state, break the connection from last iteration
79
80         loss = loss_func(prediction, y)          # calculate loss
81         optimizer.zero_grad()                  # clear gradients for this training step
82         loss.backward()                       # backpropagation, compute gradients
83         optimizer.step()                      # apply gradients
```



# Build an RNN

---



## Build an RNN

---

- ▶ Train a recurrent neural network on the IMDB large movie review dataset for sentiment analysis
- ▶ Large Movie Review Dataset
  - ▶ <http://ai.stanford.edu/~amaas/data/sentiment/>
  - ▶ This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. It provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing.



# Build an RNN

---

## ► Step 1: Setup

```
import tensorflow_datasets as tfds  
import tensorflow as tf
```

## ► Import matplotlib and create a helper function to plot graphs:

```
import matplotlib.pyplot as plt  
  
def plot_graphs(history, metric):  
    plt.plot(history.history[metric])  
    plt.plot(history.history['val_'+metric], '--')  
    plt.xlabel("Epochs")  
    plt.ylabel(metric)  
    plt.legend([metric, 'val_'+metric])
```

# Build an RNN

---

## ► Step 2: Setup input pipeline

- The IMDB large movie review dataset is a binary classification dataset—all the reviews have either a positive or negative sentiment.
- Download the dataset using TFDS. <https://www.tensorflow.org/datasets>

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

WARNING:absl:TFDS datasets with text encoding are deprecated and will be removed in a future version. Instead, you should use the plain text version and tokenize the text using `tensorflow\_text` (See: [https://www.tensorflow.org/tutorials/tensorflow\\_text/intro#tfdata\\_example](https://www.tensorflow.org/tutorials/tensorflow_text/intro#tfdata_example))

Downloading and preparing dataset imdb\_reviews/subwords8k/1.0.0 (download: 80.23 MiB, generated: Unknown size, total: 80.23 MiB) to /home/kbuilder/tensorflow\_datasets/imdb\_reviews/subwords8k/1.0.0... Shuffling and writing examples to /home/kbuilder/tensorflow\_datasets/imdb\_reviews/subwords8k/1.0.0.incomplete7GBYY4/imdb\_reviews-train.tfrecord Shuffling and writing examples to /home/kbuilder/tensorflow\_datasets/imdb\_reviews/subwords8k/1.0.0.incomplete7GBYY4/imdb\_reviews-test.tfrecord Shuffling and writing examples to /home/kbuilder/tensorflow\_datasets/imdb\_reviews/subwords8k/1.0.0.incomplete7GBYY4/imdb\_reviews-unsupervised.tfrecord Dataset imdb\_reviews downloaded and prepared to /home/kbuilder/tensorflow\_datasets/imdb\_reviews/subwords8k/1.0.0. Subsequent calls will reuse this data.

# Build an RNN

---

## ► Step 2: Setup input pipeline

- Data set info includes encoder (`tfds.features.text.SubwordTextEncoder`)

```
encoder = info.features['text'].encoder  
  
print('Vocabulary size: {}'.format(encoder.vocab_size))
```

Vocabulary size: 8185

# Build an RNN

---

## ► Step 2: Setup input pipeline

- This text encoder will encode any string in a reversible manner and fall back to byte encoding if necessary

```
sample_string = 'Hello TensorFlow.'  
  
encoded_string = encoder.encode(sample_string)  
print('Encoded string is {}'.format(encoded_string))  
  
original_string = encoder.decode(encoded_string)  
print('The original string: "{}"'.format(original_string))
```

Encoded string is [4025, 222, 6307, 2327, 4043, 2120, 7975] The original string: "Hello TensorFlow."

# Build an RNN

---

## ► Step 2: Setup input pipeline

- This text encoder will encode any string in a reversible manner and fall back to byte encoding if necessary

```
assert original_string == sample_string

for index in encoded_string:
    print('{} ----> {}'.format(index, encoder.decode([index])))
```

4025 ----> Hell  
222 ----> o  
6307 ----> Ten  
2327 ----> sor  
4043 ----> Fl  
2120 ----> ow  
7975 ----> .

# Build an RNN

---

## ► Step 3: Prepare data for training

- Create batches of these encoded strings. Use the padded\_batch method to pad sequence zeros to the length of the longest string in the batch:

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64

train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE)

test_dataset = test_dataset.padded_batch(BATCH_SIZE)
```

# Build an RNN

---

## ► Step 4: Create a model

- ▶ Build a `tf.keras.Sequential` model and start with the embedding vector layer. The embedding vector layer stores a vector for each word. When called, it converts a sequence of word indexes into a sequence of vectors. These vectors are trainable. (On enough data) After training, words with similar meanings usually have similar vectors.
- ▶ Compared with the equivalent operation of passing the one-hot encoding vector through the `tf.keras.layers.Dense` layer, this index search method is much more efficient.
- ▶ Recurrent Neural Networks (RNN) process sequence input by traversing elements. RNN passes the output from one time step to its input, and then to the next step.
- ▶ The `tf.keras.layers.Bidirectional` wrapper can also be used with RNN layers. This will propagate the input forward and backward through the RNN layer, and then connect the output. This helps RNN learn long-term dependencies.

# Build an RNN

---

## ► Step 4: Create a model

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

## ► Compile the Keras model to configure the training process:

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

# Build an RNN

---

## ► Step 5: Train a model

```
history = model.fit(train_dataset, epochs=10,  
                     validation_data=test_dataset,  
                     validation_steps=30)
```

Epoch 1/10 391/391 [=====] - 41s 105ms/step - loss: 0.6363 - accuracy: 0.5736 - val\_loss: 0.4592 - val\_accuracy: 0.8010  
Epoch 2/10 391/391 [=====] - 41s 105ms/step - loss: 0.3426 - accuracy: 0.8556 - val\_loss: 0.3710 - val\_accuracy: 0.8417  
Epoch 3/10 391/391 [=====] - 42s 107ms/step - loss: 0.2520 - accuracy: 0.9047 - val\_loss: 0.3444 - val\_accuracy: 0.8719  
Epoch 4/10 391/391 [=====] - 41s 105ms/step - loss: 0.2103 - accuracy: 0.9228 - val\_loss: 0.3348 - val\_accuracy: 0.8625  
Epoch 5/10 391/391 [=====] - 42s 106ms/step - loss: 0.1803 - accuracy: 0.9360 - val\_loss: 0.3591 - val\_accuracy: 0.8552

# Build an RNN

---

## ► Step 5: Train a model

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

391/391 [=====] - 17s 43ms/step - loss: 0.4305 - accuracy: 0.8477

Test Loss: 0.43051090836524963

Test Accuracy: 0.8476799726486206

# Build an RNN

---

## ► Step 5: Train a model

- If the forecast is  $\geq 0.5$ , it is positive, otherwise it is negative.

```
def pad_to_size(vec, size):
    zeros = [0] * (size - len(vec))
    vec.extend(zeros)
    return vec

def sample_predict(sample_pred_text, pad):
    encoded_sample_pred_text = encoder.encode(sample_pred_text)

    if pad:
        encoded_sample_pred_text = pad_to_size(encoded_sample_pred_text, 64)
    encoded_sample_pred_text = tf.cast(encoded_sample_pred_text, tf.float32)
    predictions = model.predict(tf.expand_dims(encoded_sample_pred_text, 0))

    return (predictions)
```

# Build an RNN

---

## ► Step 5: Train a model

- If the forecast is  $\geq 0.5$ , it is positive, otherwise it is negative.

```
# predict on a sample text without padding.

sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

[[ -0.11829309]]

```
# predict on a sample text with padding

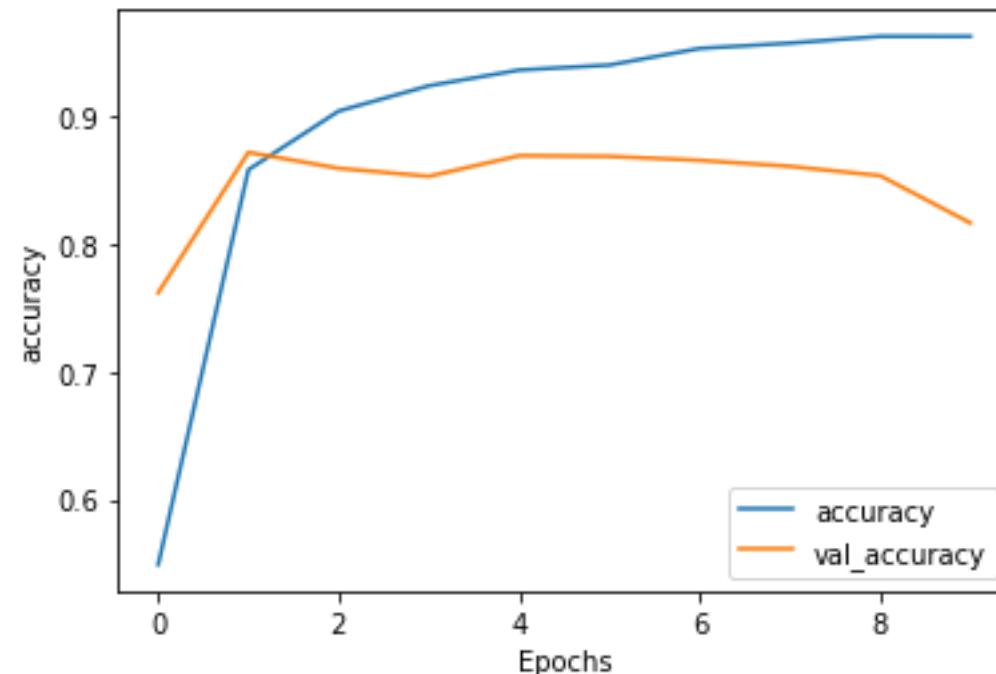
sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

[[ -1.162545]]

# Build an RNN

## ► Step 5: Train a model

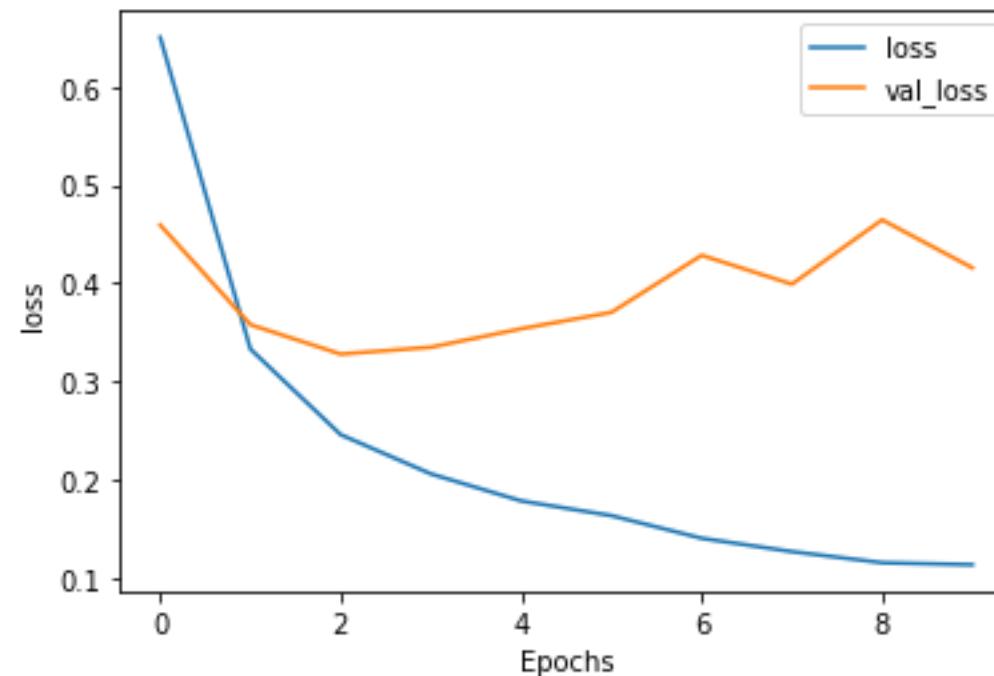
```
plot_graphs(history, 'accuracy')
```



# Build an RNN

## ► Step 5: Train a model

```
plot_graphs(history, 'loss')
```



# Build an RNN

## ► Step 6: add two or more LSTM layers

- The Keras loop layer has two available modes, which are controlled by the `return_sequences` constructor parameter

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])

history = model.fit(train_dataset, epochs=10,
                     validation_data=test_dataset,
                     validation_steps=30)
```

# Build an RNN

---

## ► Step 6: add two or more LSTM layers

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

391/391 [=====] - 30s 78ms/step - loss: 0.5205 - accuracy: 0.8572  
Test Loss: 0.5204932689666748  
Test Accuracy: 0.857200026512146

```
sample_pred_text = ('The movie was not good. The animation and the graphics
                   'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

[-2.6377363]]

# Build an RNN

---

## ► Step 6: add two or more LSTM layers

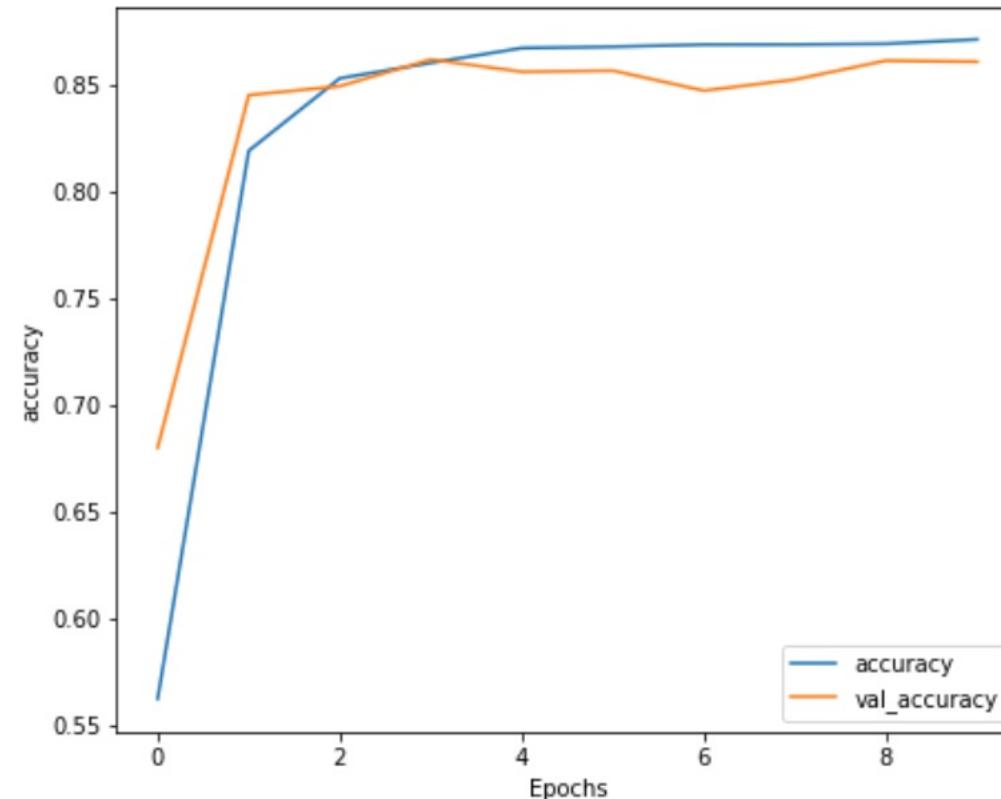
```
sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

[-3.0502243]]

# Build an RNN

## ► Step 6: add two or more LSTM layers

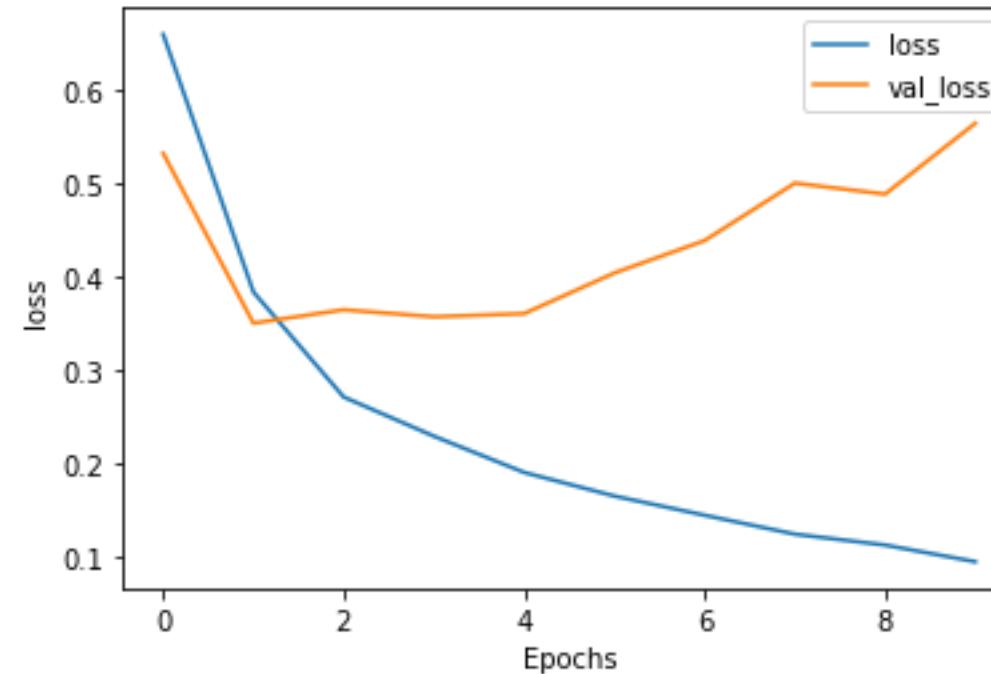
```
plot_graphs(history, 'accuracy')
```



# Build an RNN

## ► Step 6: add two or more LSTM layers

```
plot_graphs(history, 'loss')
```



# Build an RNN

```
python3 /Users/ksuo/Google Drive/github/CS7357/rnn  
the MLIR optimization passes are enabled (registered 2)  
391/391 [=====] - 309s 784ms/step - loss: 0.6841 - accuracy: 0.5147 - val_l  
oss: 0.4780 - val_accuracy: 0.7635  
Epoch 2/10  
391/391 [=====] - 306s 782ms/step - loss: 0.3999 - accuracy: 0.8221 - val_l  
oss: 0.3547 - val_accuracy: 0.8604  
Epoch 3/10  
391/391 [=====] - 308s 789ms/step - loss: 0.2645 - accuracy: 0.8972 - val_l  
oss: 0.3261 - val_accuracy: 0.8609  
Epoch 4/10  
391/391 [=====] - 312s 798ms/step - loss: 0.2129 - accuracy: 0.9218 - val_l  
oss: 0.3309 - val_accuracy: 0.8693  
Epoch 5/10  
391/391 [=====] - 310s 793ms/step - loss: 0.1780 - accuracy: 0.9380 - val_l  
oss: 0.3846 - val_accuracy: 0.8656  
Epoch 6/10  
391/391 [=====] - 310s 792ms/step - loss: 0.1672 - accuracy: 0.9402 - val_l  
oss: 0.3597 - val_accuracy: 0.8661  
Epoch 7/10  
391/391 [=====] - 312s 798ms/step - loss: 0.1431 - accuracy: 0.9530 - val_l  
oss: 0.3627 - val_accuracy: 0.8604  
Epoch 8/10  
391/391 [=====] - 317s 812ms/step - loss: 0.1363 - accuracy: 0.9532 - val_l  
oss: 0.4037 - val_accuracy: 0.8672  
Epoch 9/10  
391/391 [=====] - 321s 821ms/step - loss: 0.1204 - accuracy: 0.9604 - val_l  
oss: 0.4050 - val_accuracy: 0.8620  
Epoch 10/10  
391/391 [=====] - 327s 837ms/step - loss: 0.1124 - accuracy: 0.9640 - val_l  
oss: 0.4469 - val_accuracy: 0.8573  
391/391 [=====] - 83s 211ms/step - loss: 0.4457 - accuracy: 0.8553  
Test Loss: 0.4457261264324188  
Test Accuracy: 0.8553199768066406  
[[0.07003495]]  
[[-0.01062911]]
```

