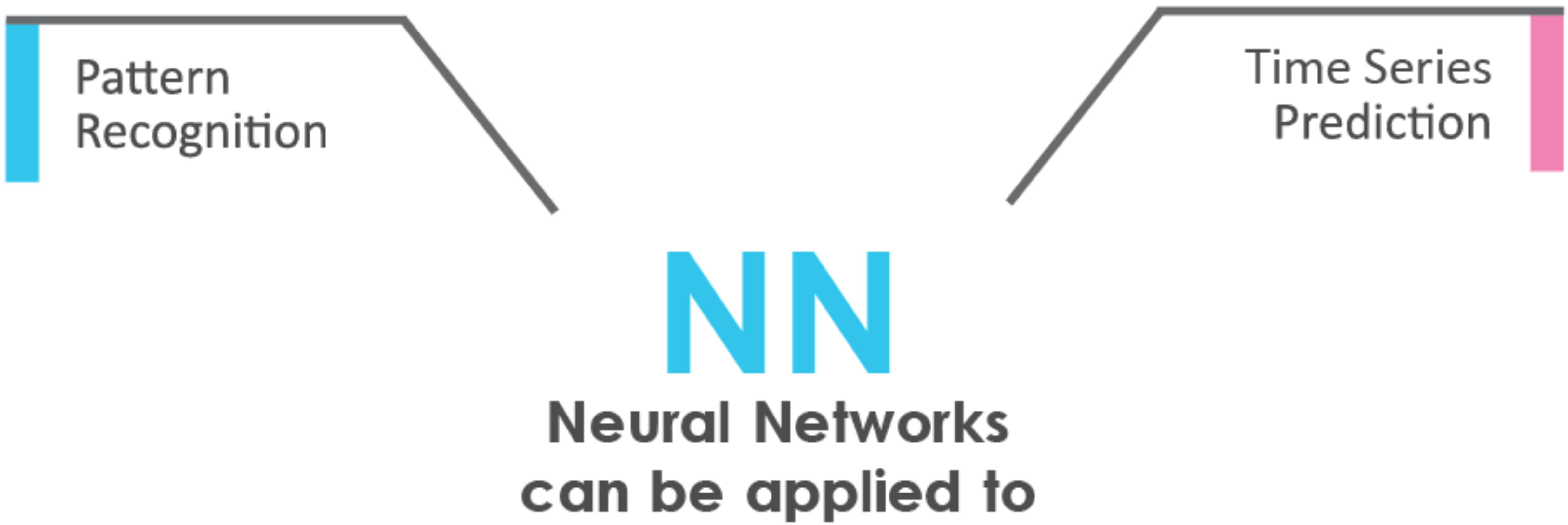
NUS-ISS
Pattern Recognition using Machine Learning System

Module 7 - Solving temporal sequential problems using recurrent neural networks

by Dr. Tan Jen Hong

When time is a factor

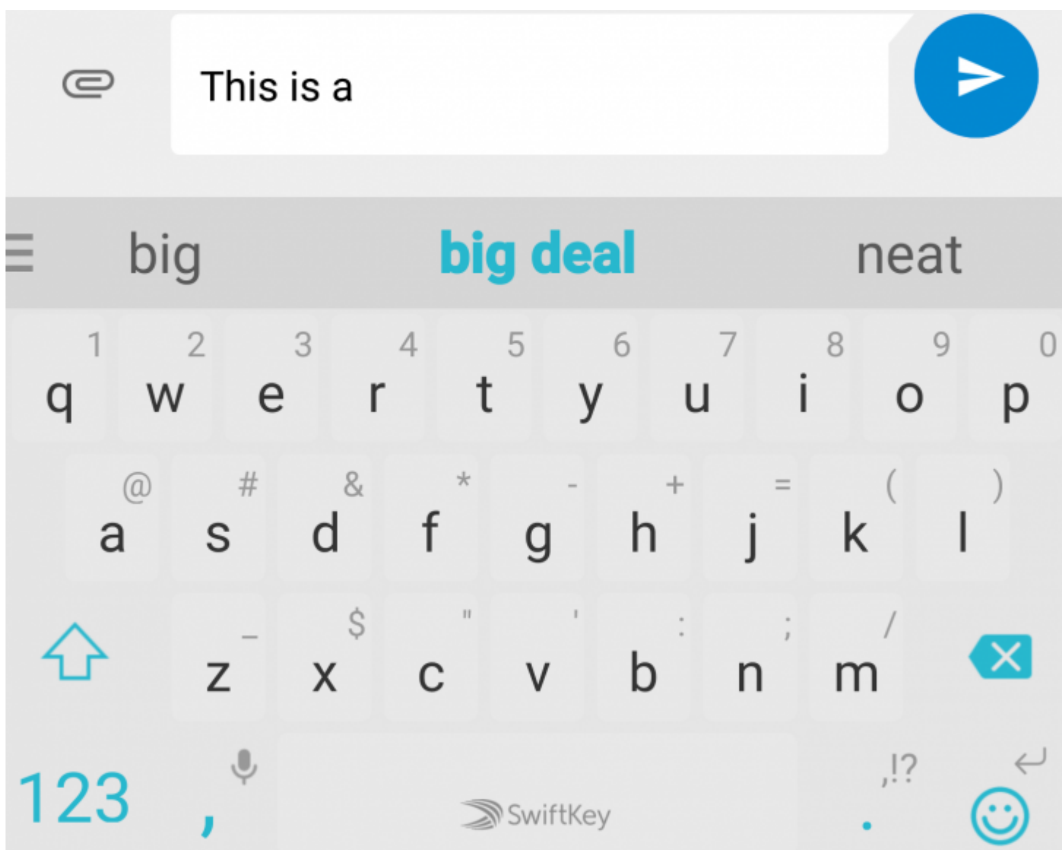
The other application
with time



Source: http://www.cvisiontech.com/resources/ocr-primer/ocr-neural-networks-and-other-machine-learning-techniques.html

The related inputs

Not independent



Source:https://www.androidpolice.com/2015/11/06/swiftkey-update-to-v6-0-with-redesigned-settings-menu-double-word-prediction-and-more/

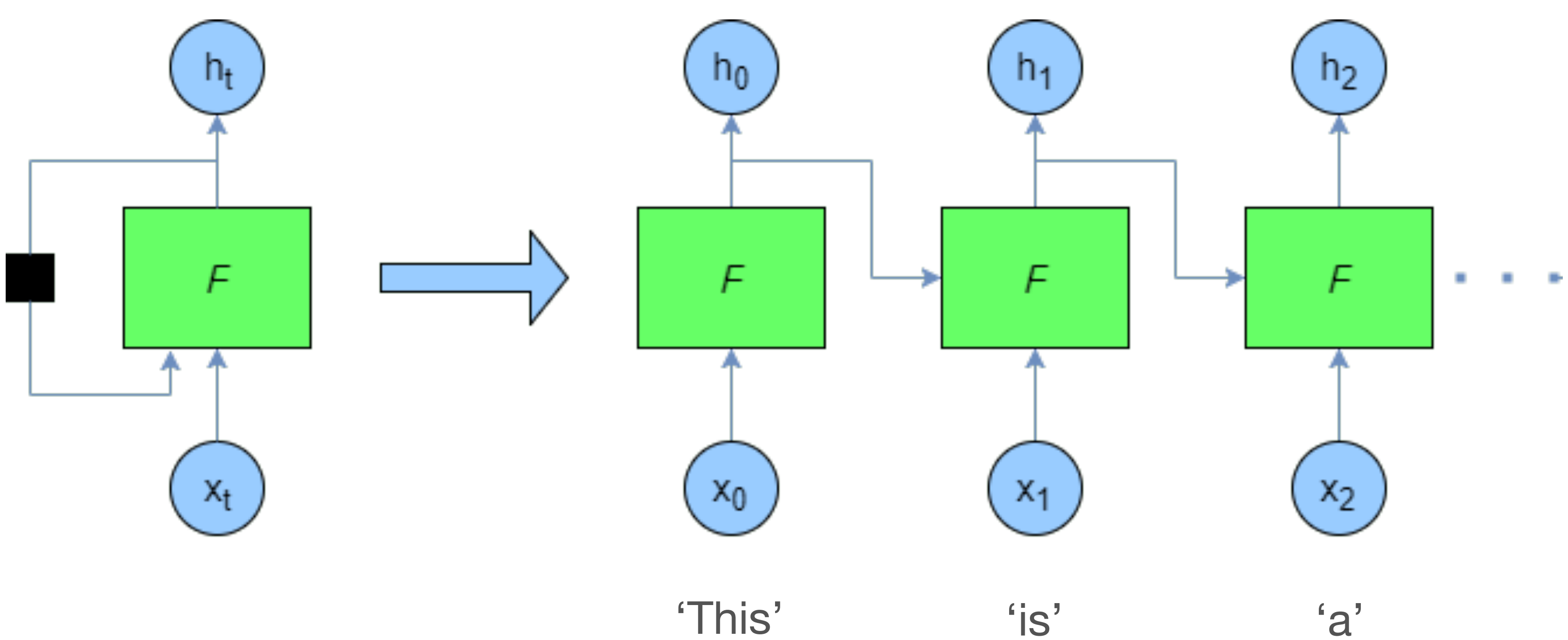
- So far the nets introduced assume inputs are independent from each other
- Implication: the inputs that came before and the inputs that will come after has no relationship
- But for some tasks/problems, this is not true
- E.g. if. you want to predict which word to come in a sentence, you better know what have been typed/said before

Recurrent neural network

In short, RNN

- Recurrent neural network: a net that perform the same calculation on elements/segments from a sequence

- The output of a current element/ segment depends on the outputs from the previous elements/ segments



Source: https://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow/

CNN vs RNN

Comparison

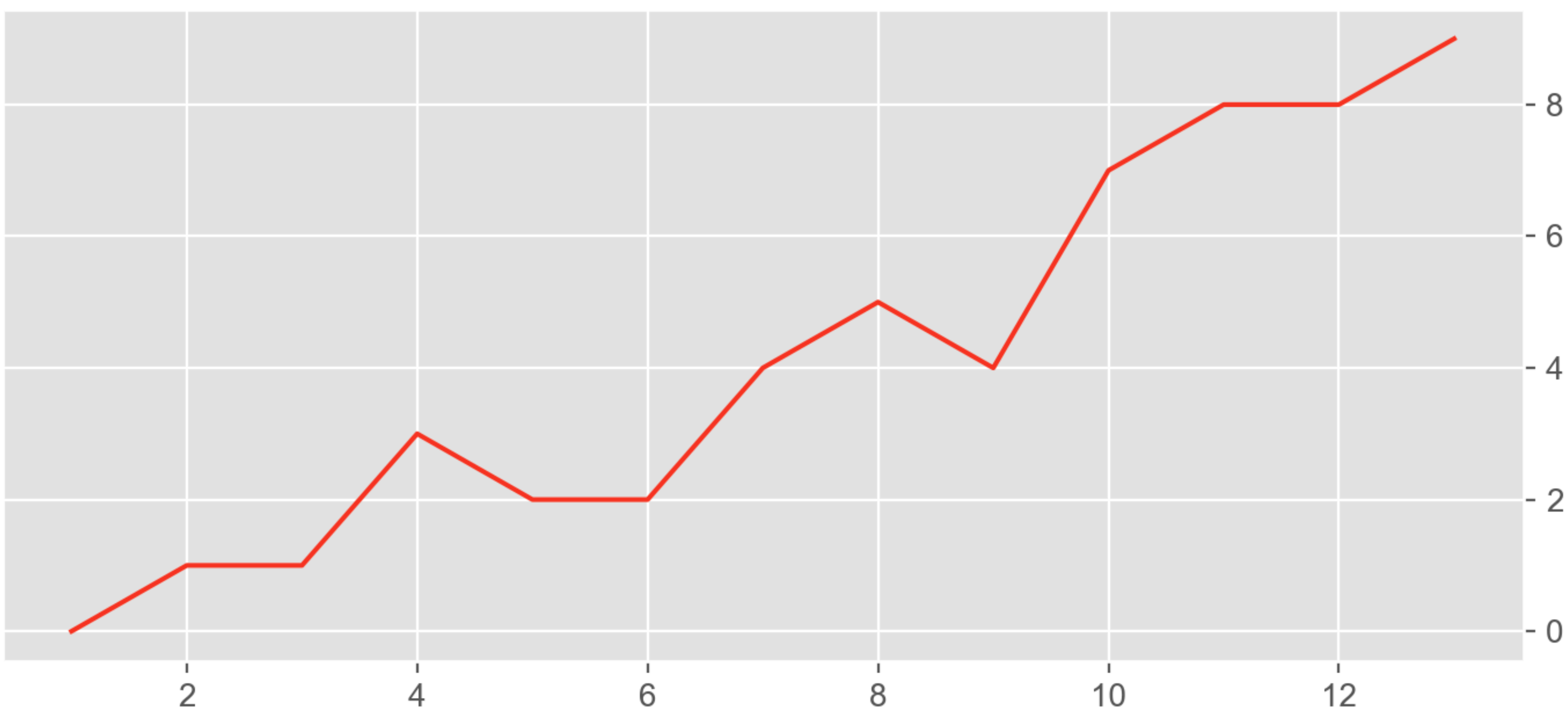
	CNN	RNN / LSTM
Usage	Suitable for spatial data, e.g. image, video	Suitable for temporal data (sequential data), e.g. text, speech
Capability	Considered more powerful than RNN	Less powerful and slower in calculation
Input	Take fixed size inputs and generate fixed size outputs	Can handle arbitrary input/output lengths
Nature	Use local connectivity pattern (through 2D convolution)	Use time series information

Time step

Prediction

- Assume we have a series of 13 values, and we would like to predict the next value

```
0, 1, 1, 3, 2, 2, 4, 5, 4, 7, 8, 8, 9, ?
```

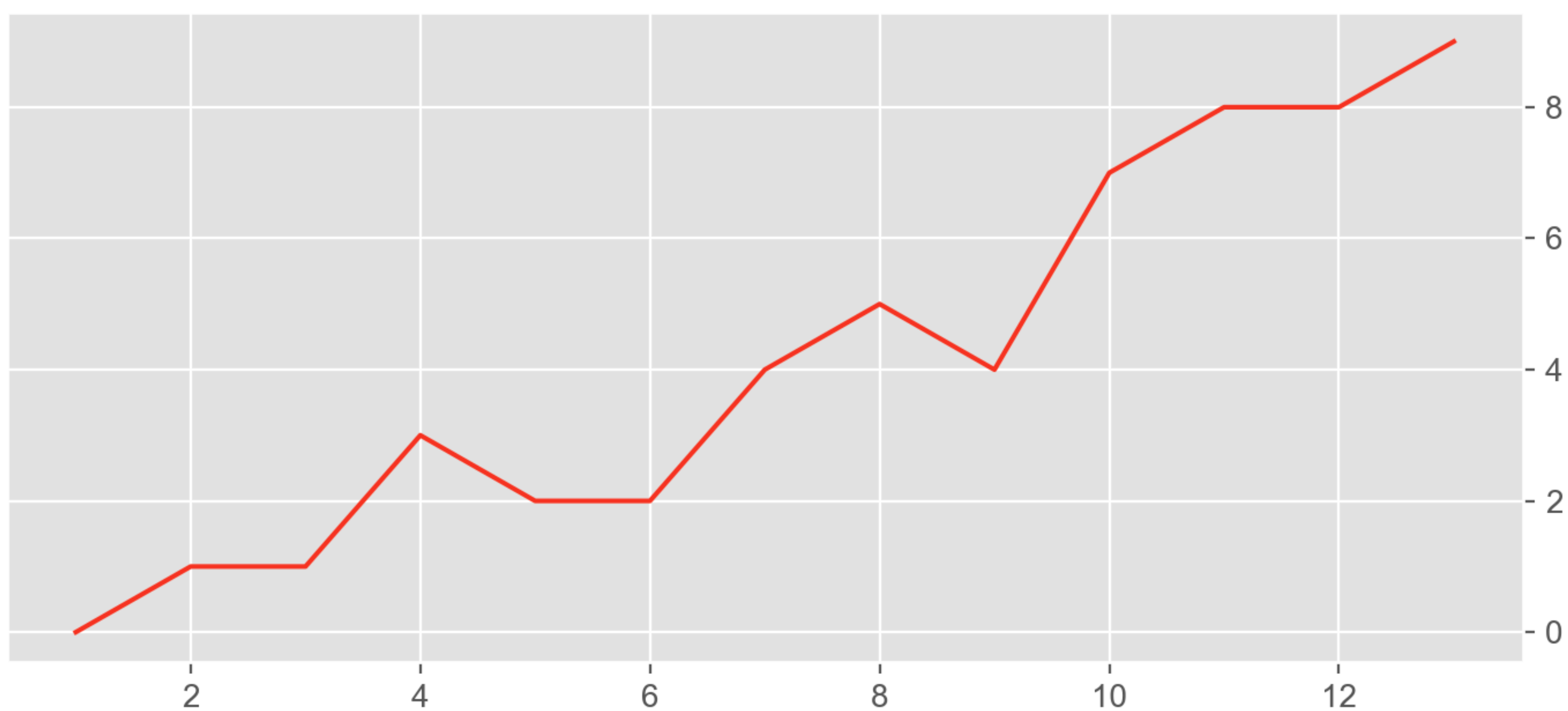


Time step

Make segments

- For time series problem, we don't feed in the entire series into the net, as this will not have the 'recurrent' learning involved

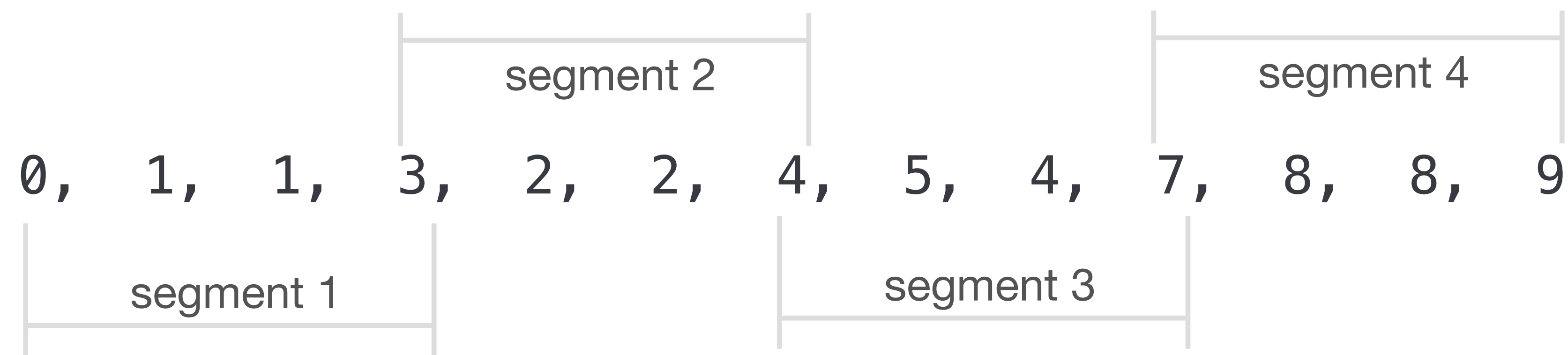
- Instead, we chop the series into segments, each segment with a fixed amount of time steps, called length



makeSteps

The procedure

- Assume we want to have a length of 4, and a distance of 3



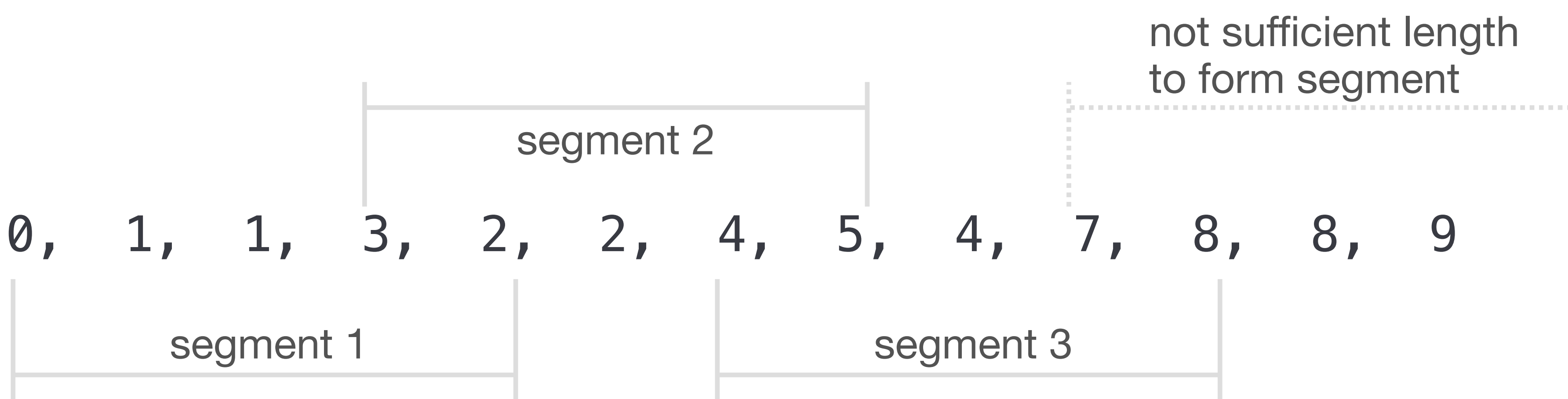
- The preprocessed input:

```
0,  1,  1,  3
3,  2,  2,  4
4,  5,  4,  7
7,  8,  8,  9
```

makeSteps

The procedure

- Assume we want to have a length of 5, and a distance of 3



- The preprocessed input:

```
0,  1,  1,  3,  2
3,  2,  2,  4,  5
4,  5,  4,  7,  8
```

Time for exercise

Exercise

The procedure

- Write a function that can generate the desired preprocessed input from a 1D series/signals with the below signature

```
> def makeSteps(dat, length, dist):
```

```
0,  1,  1,  3,  2,  2,  4,  5,  4,  7,  8,  8,  9
```

- length 4, distance 3

```
0,  1,  1,  3
3,  2,  2,  4
4,  5,  4,  7
7,  8,  8,  9
```

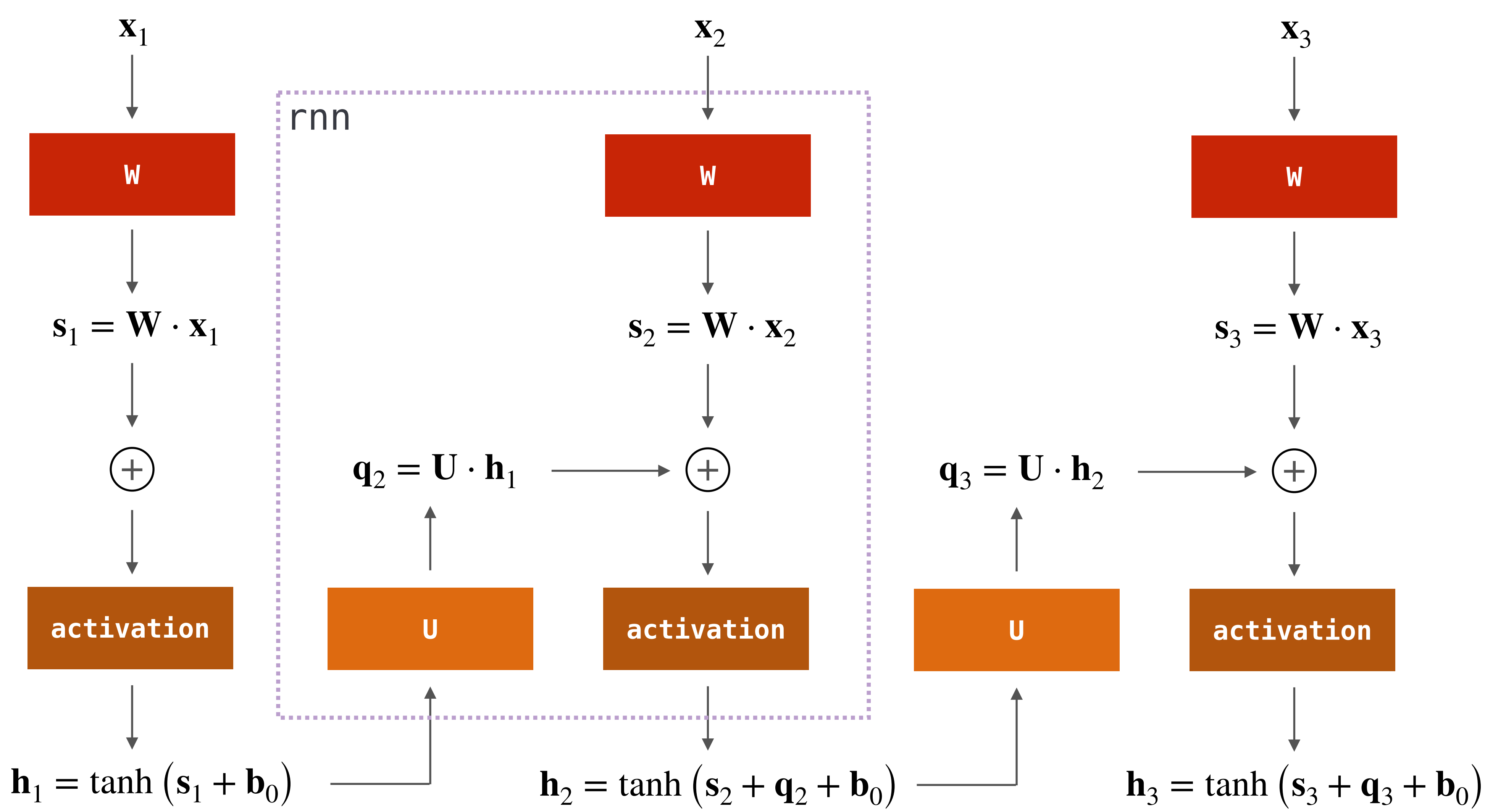
- length 5, distance 3

```
0,  1,  1,  3,  2
3,  2,  2,  4,  5
4,  5,  4,  7,  8
```

RNN
The working

- For RNN, the we feed the input segment by segment or row by row

0,	1,	1,	3,	2	$\longrightarrow$	$\mathbf{x}_1$
3,	2,	2,	4,	5	$\longrightarrow$	$\mathbf{x}_2$
4,	5,	4,	7,	8	$\longrightarrow$	$\mathbf{x}_3$



Dot product
The working

- What is the output of

$$\mathbf{W} \cdot \mathbf{x}$$

- Assume the $\mathbf{W}$ is a m x n matrix, $\mathbf{x}$ is n x 1 matrix, the matrix-vector dot product is

$$\mathbf{W} \cdot \mathbf{x} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + \cdots + w_{1n}x_n \\ w_{21}x_1 + w_{22}x_2 + \cdots + w_{2n}x_n \\ \vdots \\ w_{m1}x_1 + w_{m2}x_2 + \cdots + w_{mn}x_n \end{bmatrix}$$

- The m determines the size of the product output, i.e. the output feature size

Dot product
The working

- What is the output of

$$\mathbf{W} \cdot \mathbf{x}_1$$

- Assume

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 2 & 0 & 0 & -1 & 1 \end{bmatrix}$$

- and we know

$$\mathbf{x}_1 = [0 \quad 1 \quad 1 \quad 3 \quad 2]$$

Dot product

The working

- What is the output of

$$\mathbf{W} \cdot \mathbf{x}_1$$

- Assume

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 2 & 0 & 0 & -1 & 1 \end{bmatrix}$$

- and we know

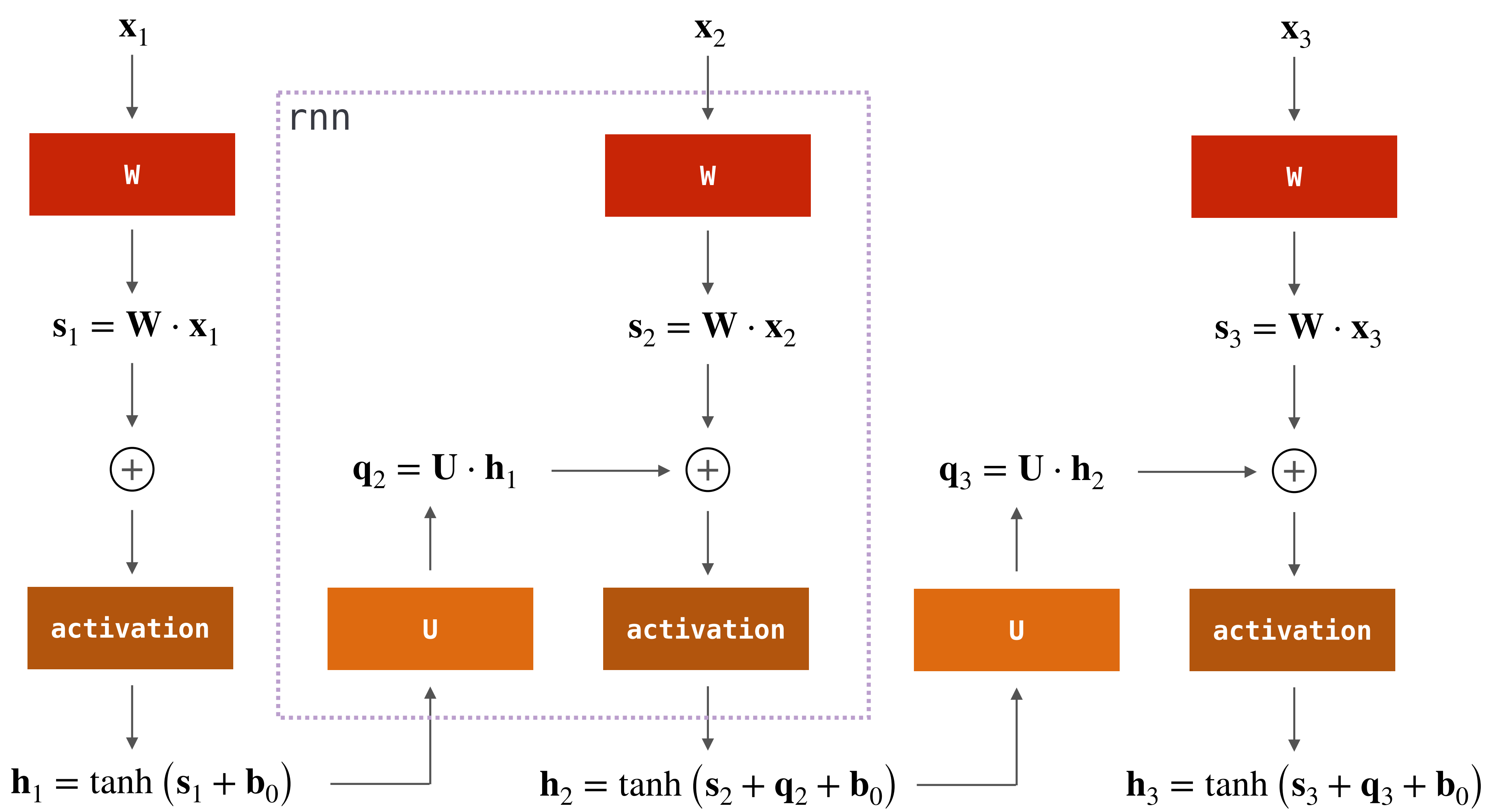
$$\mathbf{x}_1 = [0 \quad 1 \quad 1 \quad 3 \quad 2]$$

$$\mathbf{W} \cdot \mathbf{x}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 2 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \times 0 + 0 \times 1 + 0 \times 1 + 1 \times 3 + (-1) \times 2 \\ 2 \times 0 + 0 \times 1 + 0 \times 1 + (-1) \times 3 + 1 \times 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

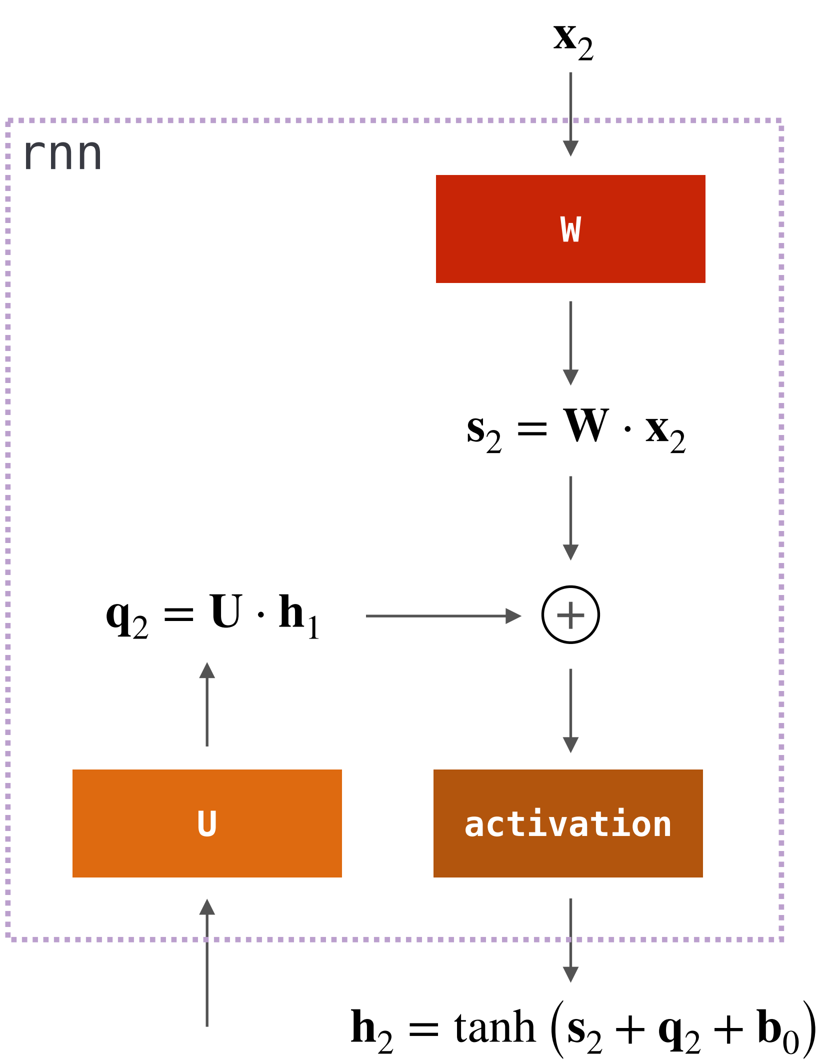
RNN

The working

- Note: The same U and W for $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$
- U and W are changed ***only*** during updating phase of training
- U is a square matrix



RNN
The problem



- Ideally, we want rnn to have long memories, so that it can connect relationships among data points far before and after a point of interest

- If this is possible, good for language understanding / translation, or how events in stock market correlate to each other

- But, in rnn, the more we perform recurrent operation, mathematically, that is equivalent to adding more layers to the net

- So vanishing gradient comes in ...

RNN

How to build rnn in Keras

- Use `SimpleRNN` for RNN layers

```
> from tensorflow.keras.layers import Input
> from tensorflow.keras.layers import SimpleRNN
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.models import Model

> inputs       = Input(shape=(16,64))
> y            = SimpleRNN(32)(inputs)
> y            = Dense(1, activation='sigmoid')(y)

> model        = Model(inputs=inputs,outputs=y)

> model.summary()
```

- How many segments in a single sample? what is the length of each segment? What is the size of the output vector from the RNN layer?

RNN

How to build rnn in Keras

```
> from tensorflow.keras.layers import Input
> from tensorflow.keras.layers import SimpleRNN
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.models import Model

> inputs      = Input(shape=(16,64))
> y           = SimpleRNN(32)(inputs)
> y           = Dense(1, activation='sigmoid')(y)

> model       = Model(inputs=inputs,outputs=y)

> model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 16, 64)            0

simple_rnn (SimpleRNN)       (None, 32)                3104      → ?

dense (Dense)                (None, 1)                 33        → 32 x 1 + 1 = 33
=================================================================
Total params: 3,137
Trainable params: 3,137
Non-trainable params: 0
```

the bias of the single output neuron

RNN

Parameter calculation

- Let l_{in} denote the number of input feature (the length of each segment)
- Let l_{out} denote the number of the output feature
- The number of parameters:

$$p = (l_{\text{out}} \times l_{\text{in}}) + (l_{\text{out}} \times l_{\text{out}}) + l_{\text{out}}$$

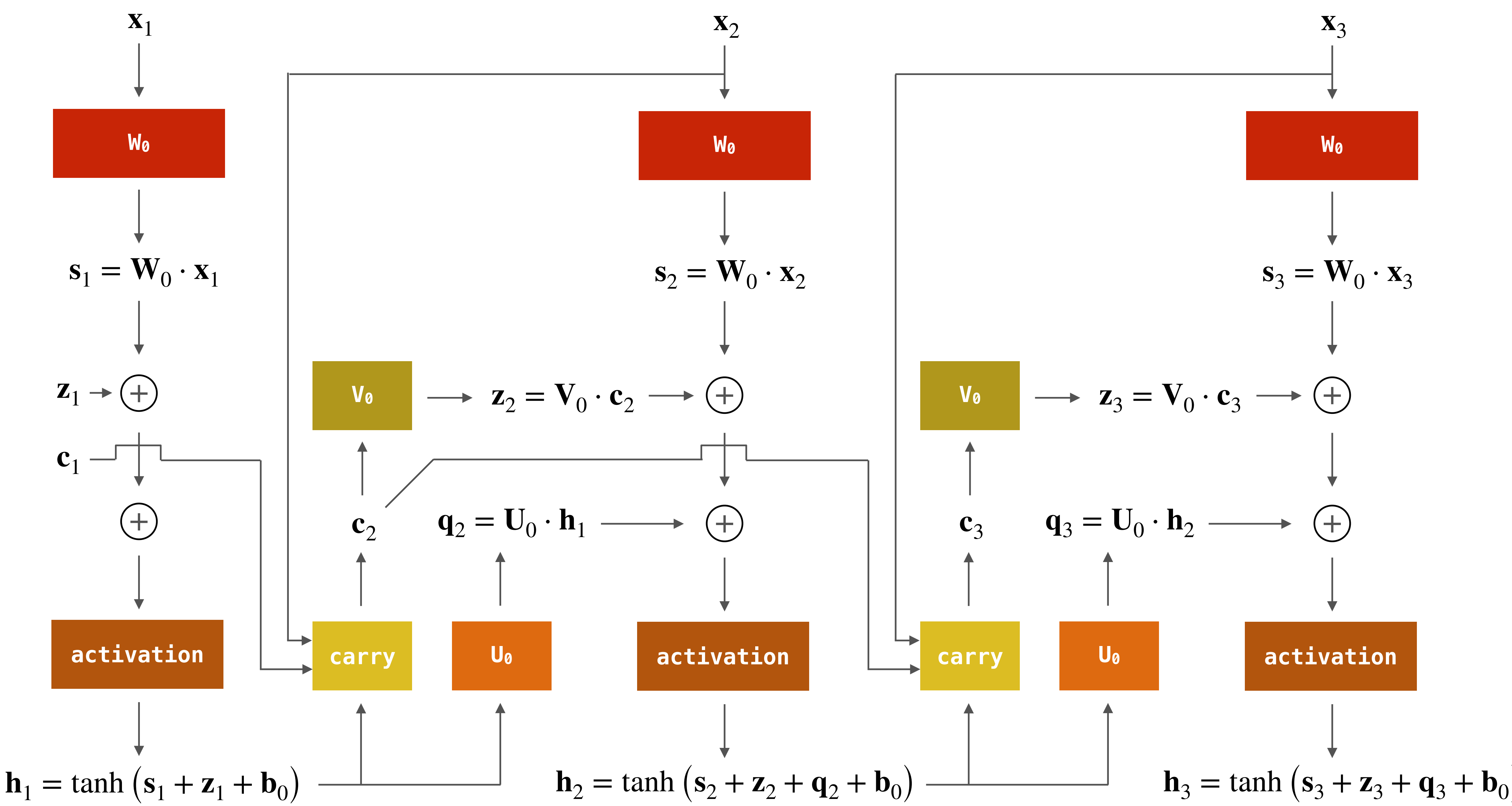
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 16, 64)            0
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                3104
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 3,137
Trainable params: 3,137
Non-trainable params: 0
_________________________________________________________________
```

→ 32 x 64 + 32 x 32 + 32 = 2048 + 1024 + 32 = 3104

→ 32 x 1 + 1 = 33

LSTM
The working

- The working of LSTM in Keras (according to Francois Chollet)



Multiplication

Element-wise

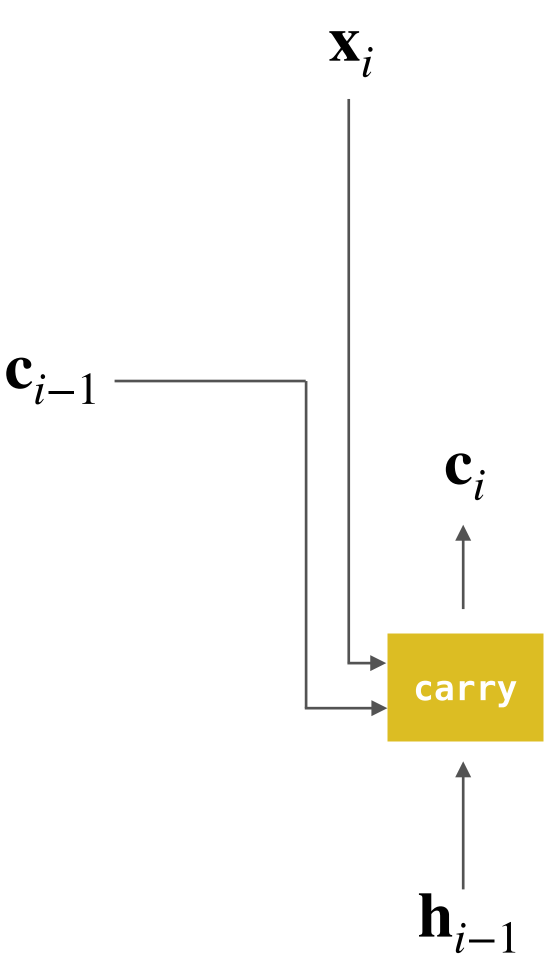
- The output of element-wise multiplication is

$$\mathbf{v} \odot \mathbf{x} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \odot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} v_1 x_1 \\ v_2 x_2 \\ \vdots \\ v_n x_n \end{bmatrix}$$

- The length of $\mathbf{v}$ and $\mathbf{x}$ must be equal

LSTM

The internal working of 'carry'



- First we define

$$\mathfrak{w}_i = \sigma\left(\mathbf{U}_{\mathfrak{w}} \cdot \mathbf{h}_{i-1} + \mathbf{W}_{\mathfrak{w}} \cdot \mathbf{x}_i + \mathbf{b}_{\mathfrak{w}}\right)$$

$$\mathfrak{f}_i = \sigma\left(\mathbf{U}_{\mathfrak{f}} \cdot \mathbf{h}_{i-1} + \mathbf{W}_{\mathfrak{f}} \cdot \mathbf{x}_i + \mathbf{b}_{\mathfrak{f}}\right)$$

$$\mathfrak{h}_i = \sigma\left(\mathbf{U}_{\mathfrak{h}} \cdot \mathbf{h}_{i-1} + \mathbf{W}_{\mathfrak{h}} \cdot \mathbf{x}_i + \mathbf{b}_{\mathfrak{h}}\right)$$

- σ is the sigmoid function, and let's denote ⊙ as element-wise multiplication, and we have

$$\mathbf{c}_i = \mathfrak{w}_i \odot \mathfrak{h}_{i-1} + \mathbf{c}_{i-1} \odot \mathfrak{f}_i$$

LSTM

How to build lstm in Keras

```
> from tensorflow.keras.layers import Input
> from tensorflow.keras.layers import LSTM
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.models import Model

> inputs      = Input(shape=(16,64))
> y           = LSTM(32)(inputs)
> y           = Dense(1, activation='sigmoid')(y)

> model       = Model(inputs=inputs,outputs=y)

> model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 16, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 32)                12416
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 12,449
Trainable params: 12,449
Non-trainable params: 0
_________________________________________________________________
```

?

32 x 1 + 1 = 33

the bias of the single output neuron

LSTM
Parameters calculation

- Let l_{in} denote the number of input feature (the length of each segment), l_{out} denote the number of the output feature

- The number of parameters:

$$p = (l_{\text{out}} \times l_{\text{out}} + l_{\text{out}} \times l_{\text{in}} + l_{\text{out}}) \times 4$$

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 16, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 32)                12416
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 12,449
Trainable params: 12,449
Non-trainable params: 0
_________________________________________________________________
```

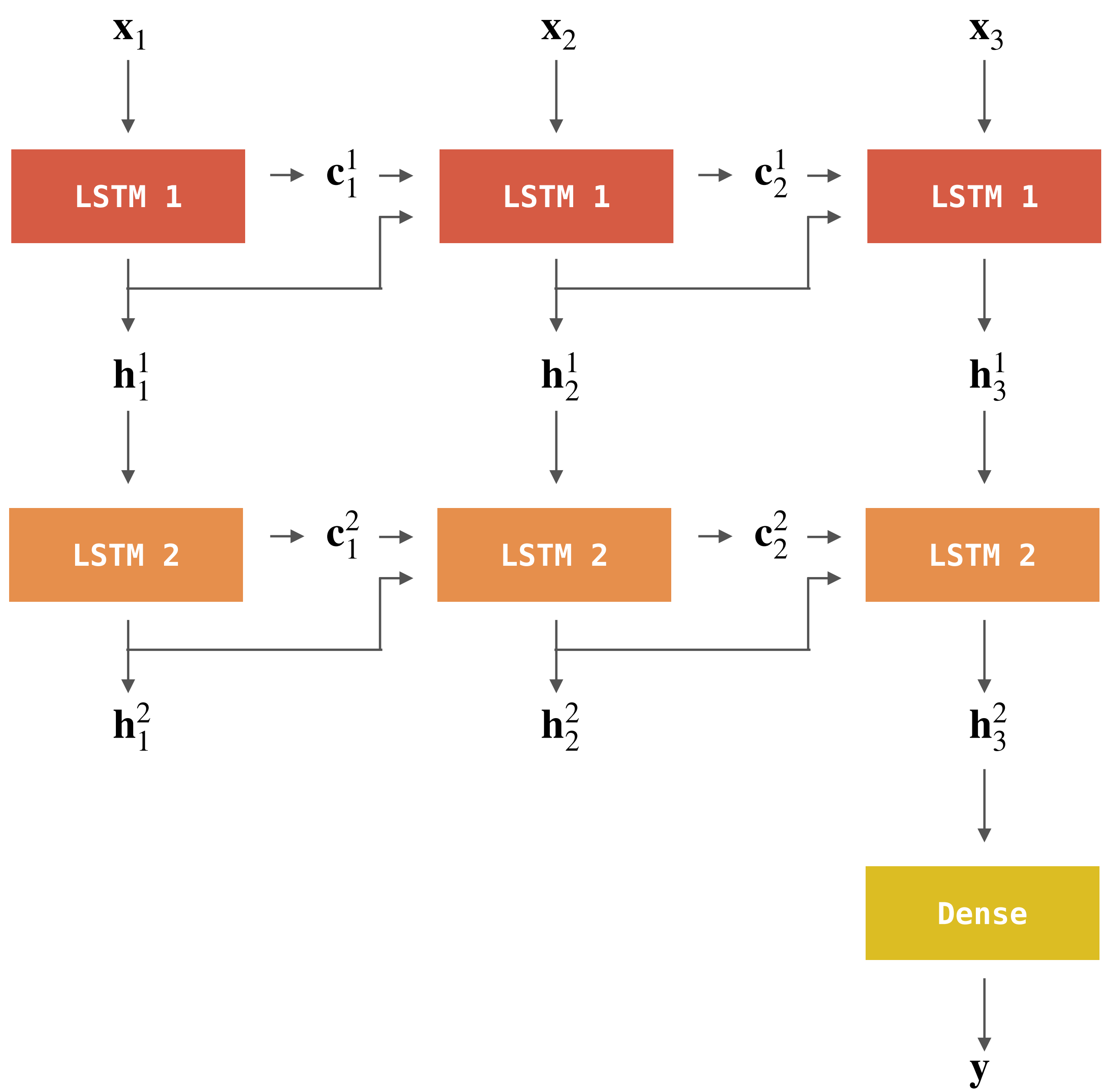
```
 (32x32 + 32x64 + 32) x 4
= 3104 x 4
= 12416
```

Stacking

Multiple recurrent layers

- It is possible to stack recurrent layers and make the net deeper

- Note: for LSTM 2 to Dense, only the final output sequence fed into Dense layer



LSTM

How to build stacked lstm in Keras

```
> inputs  = Input(shape=(3,5))
> y       = LSTM(7,return_sequences=True)(inputs)
> y       = LSTM(9)(y)
> y       = Dense(1, activation='sigmoid')(y)

> model   = Model(inputs=inputs,outputs=y)
> model.summary()
```

To stack lstm, return_sequences must be set to True

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 3, 5)              0

lstm_1 (LSTM)                (None, 3, 7)              364

lstm_2 (LSTM)                (None, 9)                 612

dense_2 (Dense)              (None, 1)                 10
=================================================================
Total params: 986
Trainable params: 986
Non-trainable params: 0
```

The output shape is `(None, 3, 7)` because of returned sequences, else it will simply be `(None, 7)`

The number of rows of the output of a lstm that returns sequences, is always equal to **the number of rows** of the input to the unit