NUS-ISS
Pattern Recognition using Machine Learning System

When time is a factor

by Dr. Tan Jen Hong

The other application

with time

Pattern Recognition

Time Series Prediction

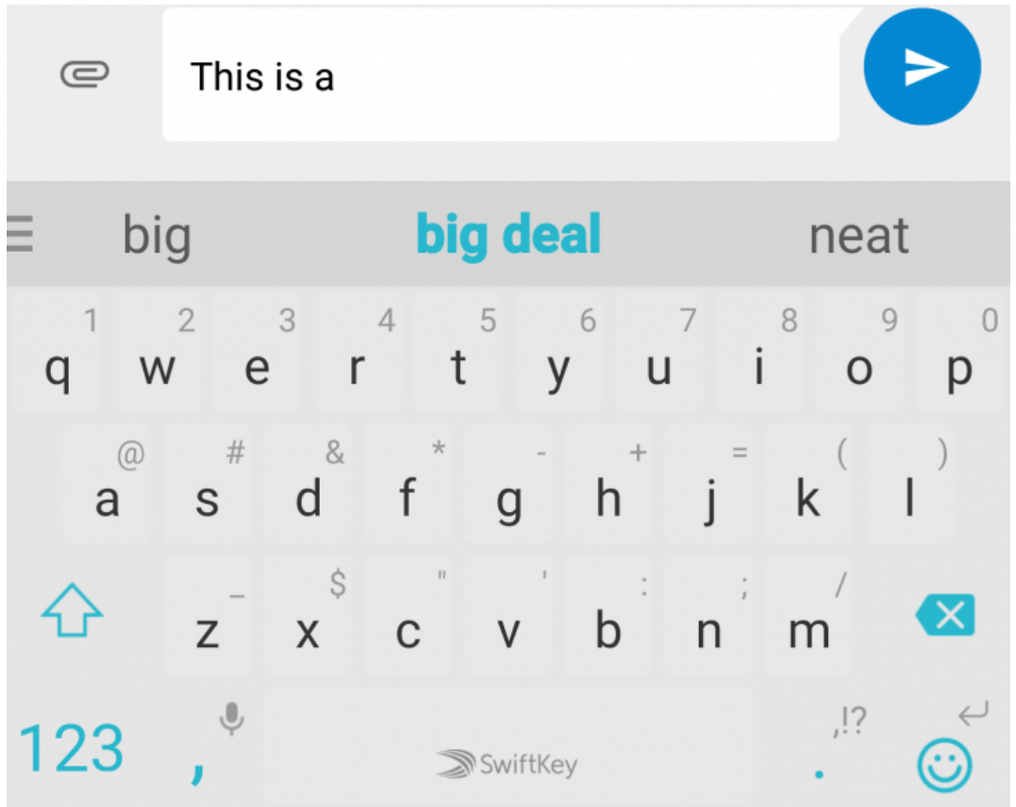
NN

Neural Networks can be applied to

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

The related inputs

Not independent



- 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

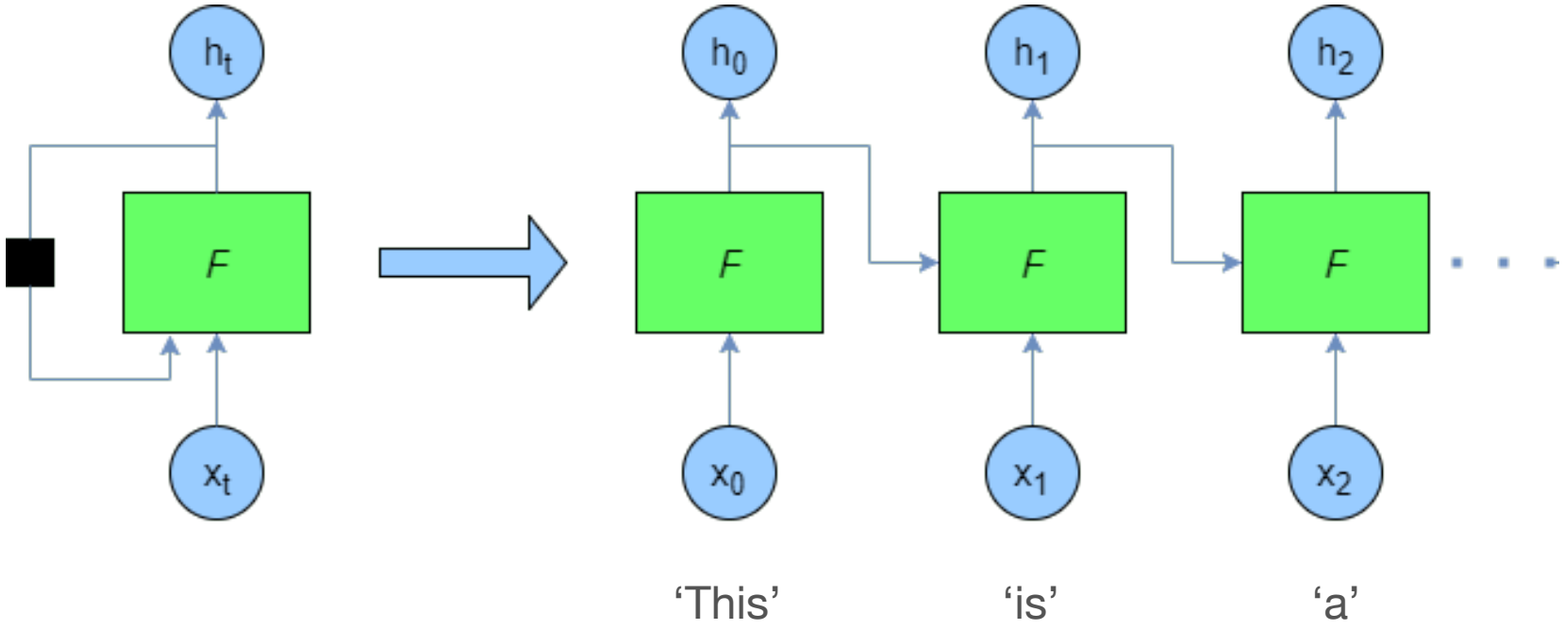
Source:https://www.androidpolice.com/2015/11/06/swiftkey-update-to-v6-0-with-redesigned-settings-menu-double-word-prediction-and-more/

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

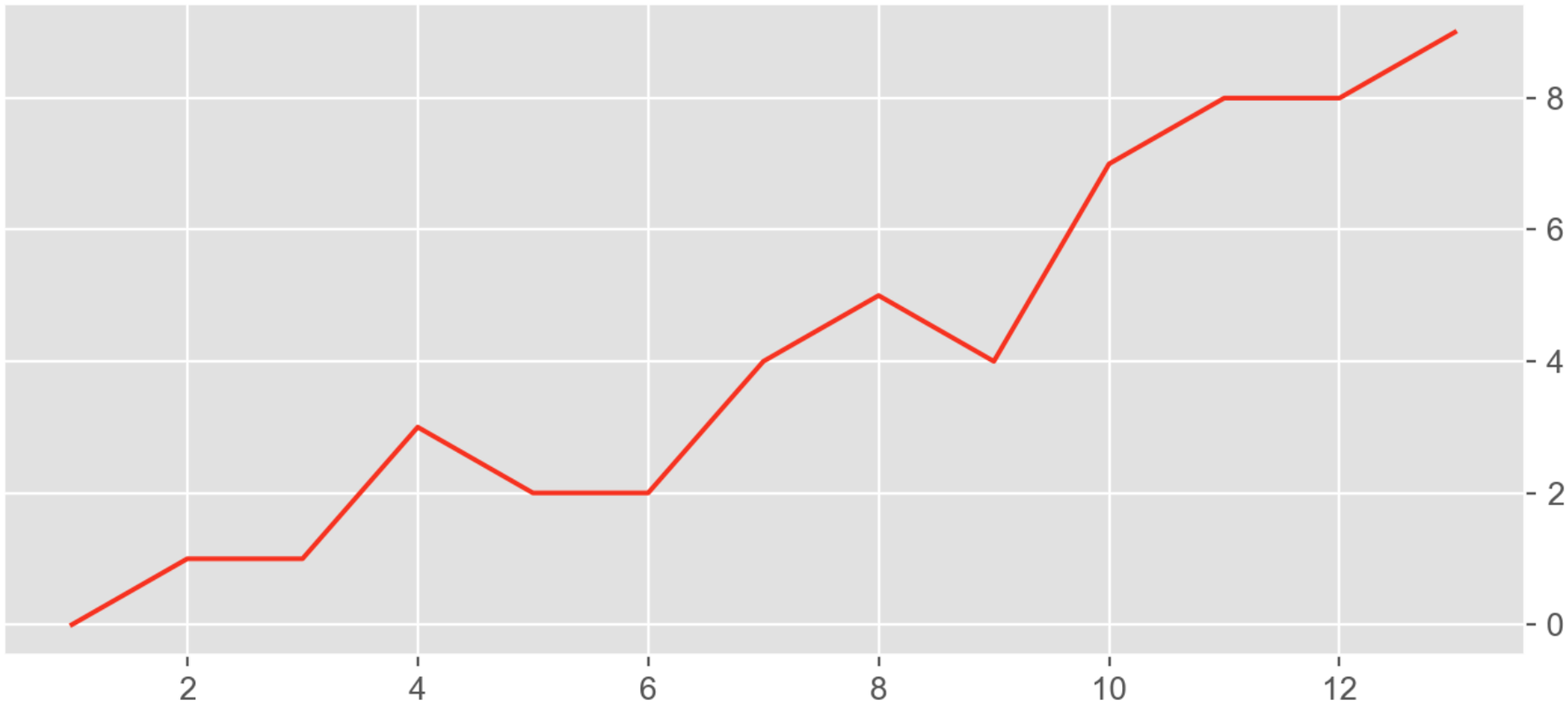
	CNN	RNN / LSTM
Usage	Suitable for spatial data, e.g. image, video	Suitable for temporal data (sequential data), e.g. text, speech
Capability	Considered more powerful than RNN	Less powerful and slower in calculation
Input	Take fixed size inputs and generate fixed size outputs	Can handle arbitrary input/output 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, ?
```

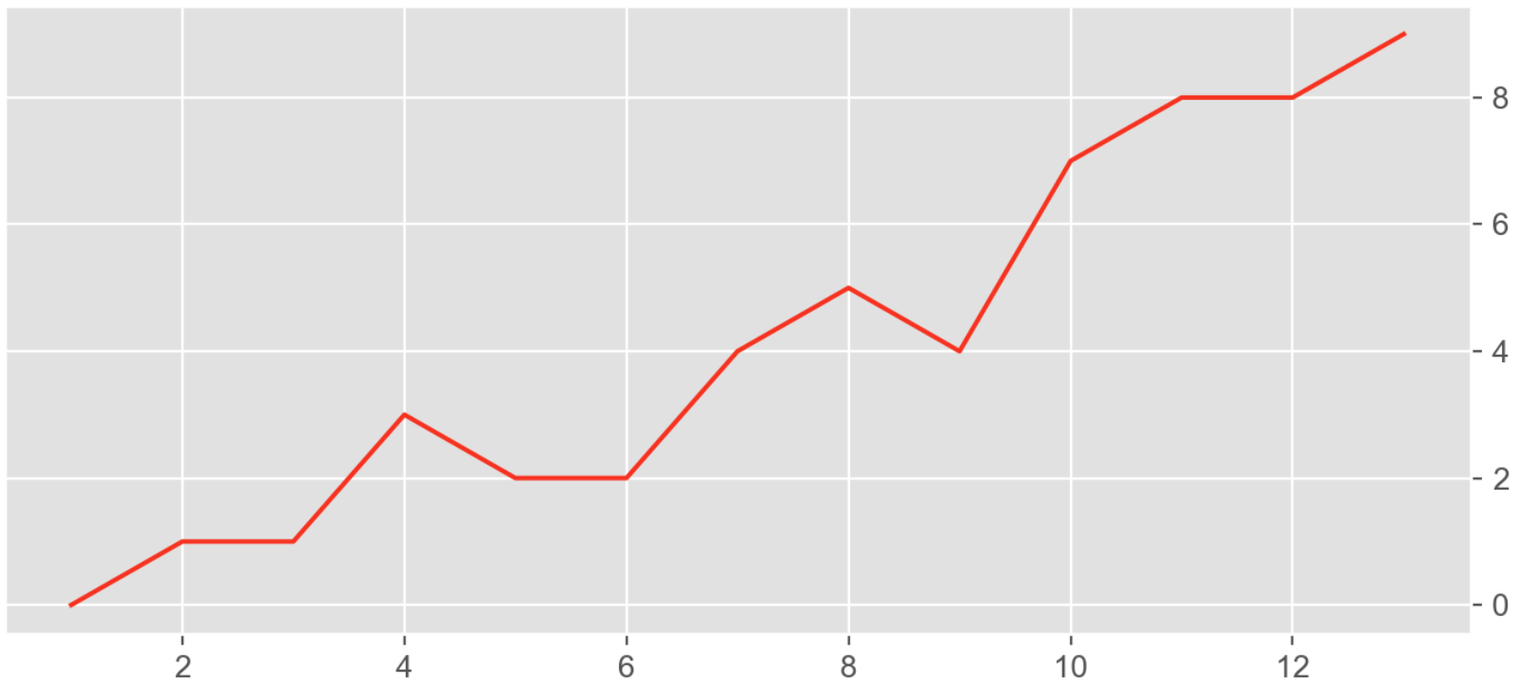


Time step

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

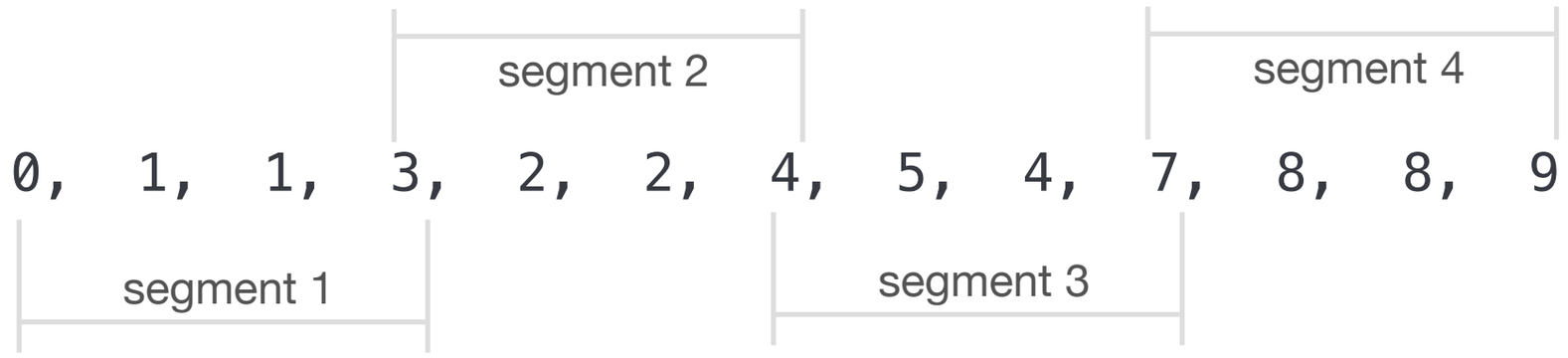
- Instead, we chop the series into segments, each segment with a fixed amount of time steps, called length



makeSteps

The procedure

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



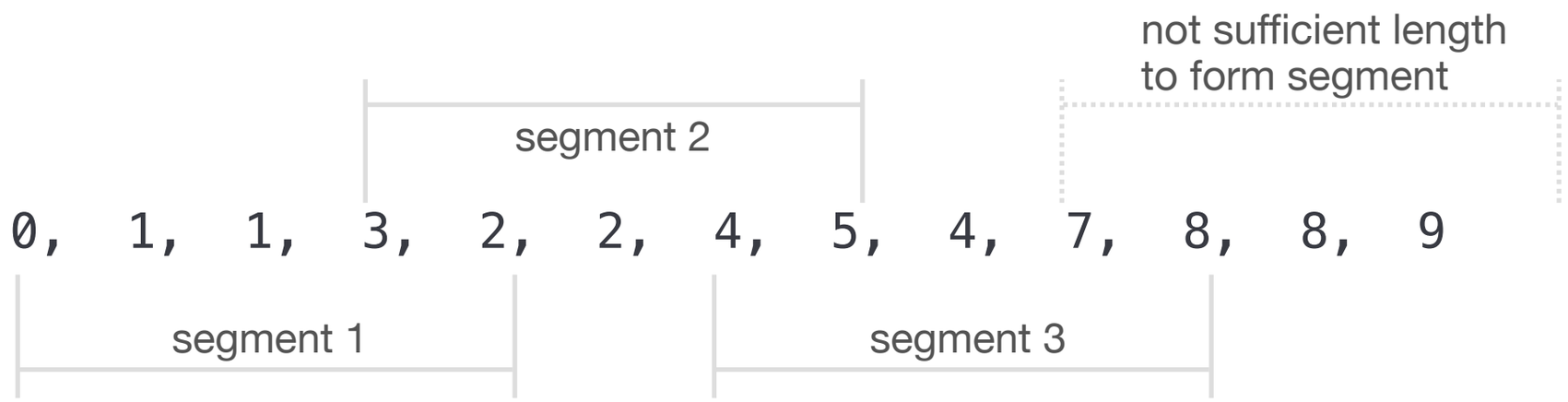
- The preprocessed input:

```
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

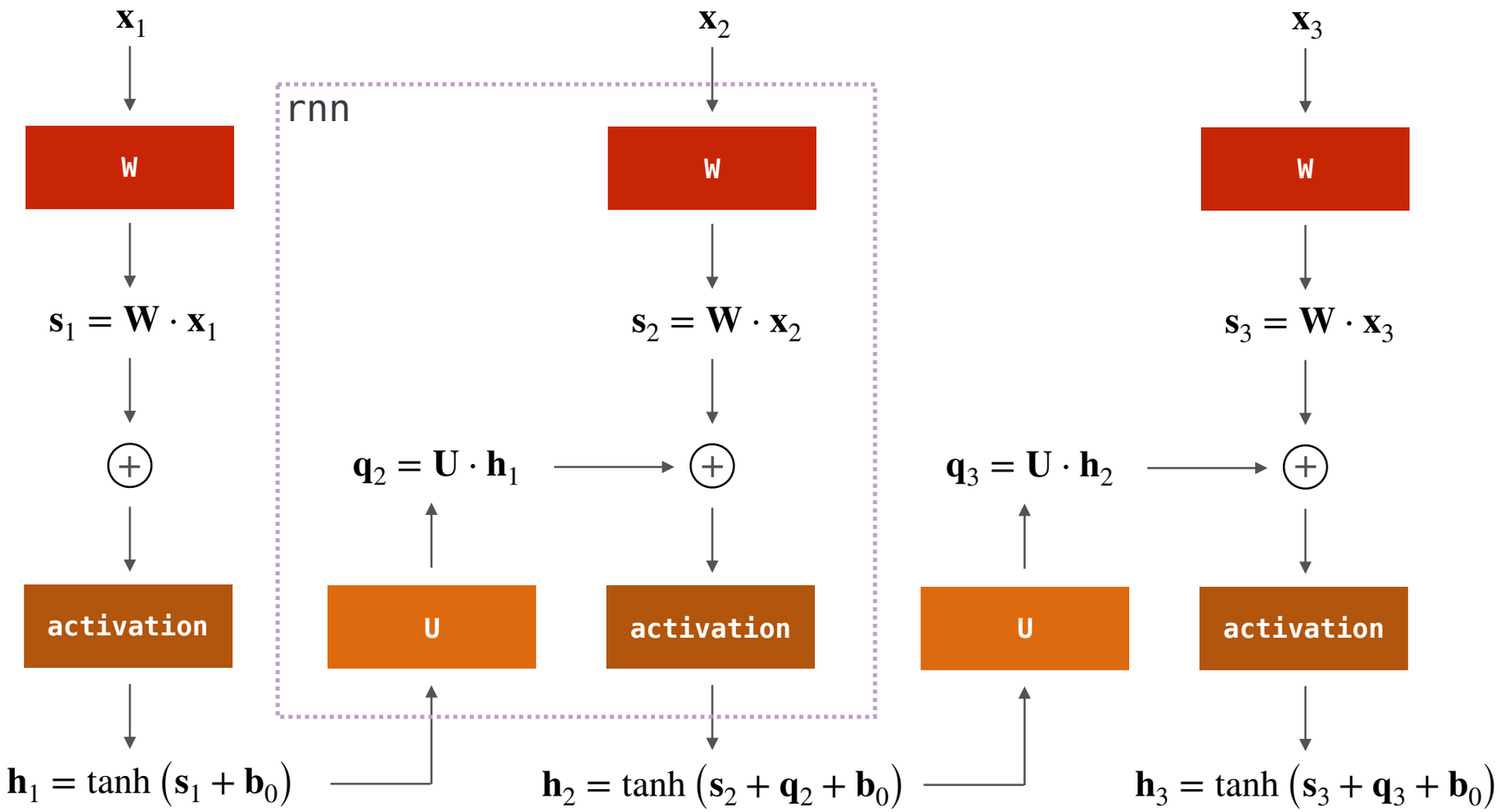
```
0,  1,  1,  3,  2
3,  2,  2,  4,  5
4,  5,  4,  7,  8
```

RNN

The working

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

```
0,  1,  1,  3,  2   ⟶  x1
3,  2,  2,  4,  5   ⟶  x2
4,  5,  4,  7,  8   ⟶  x3
```



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

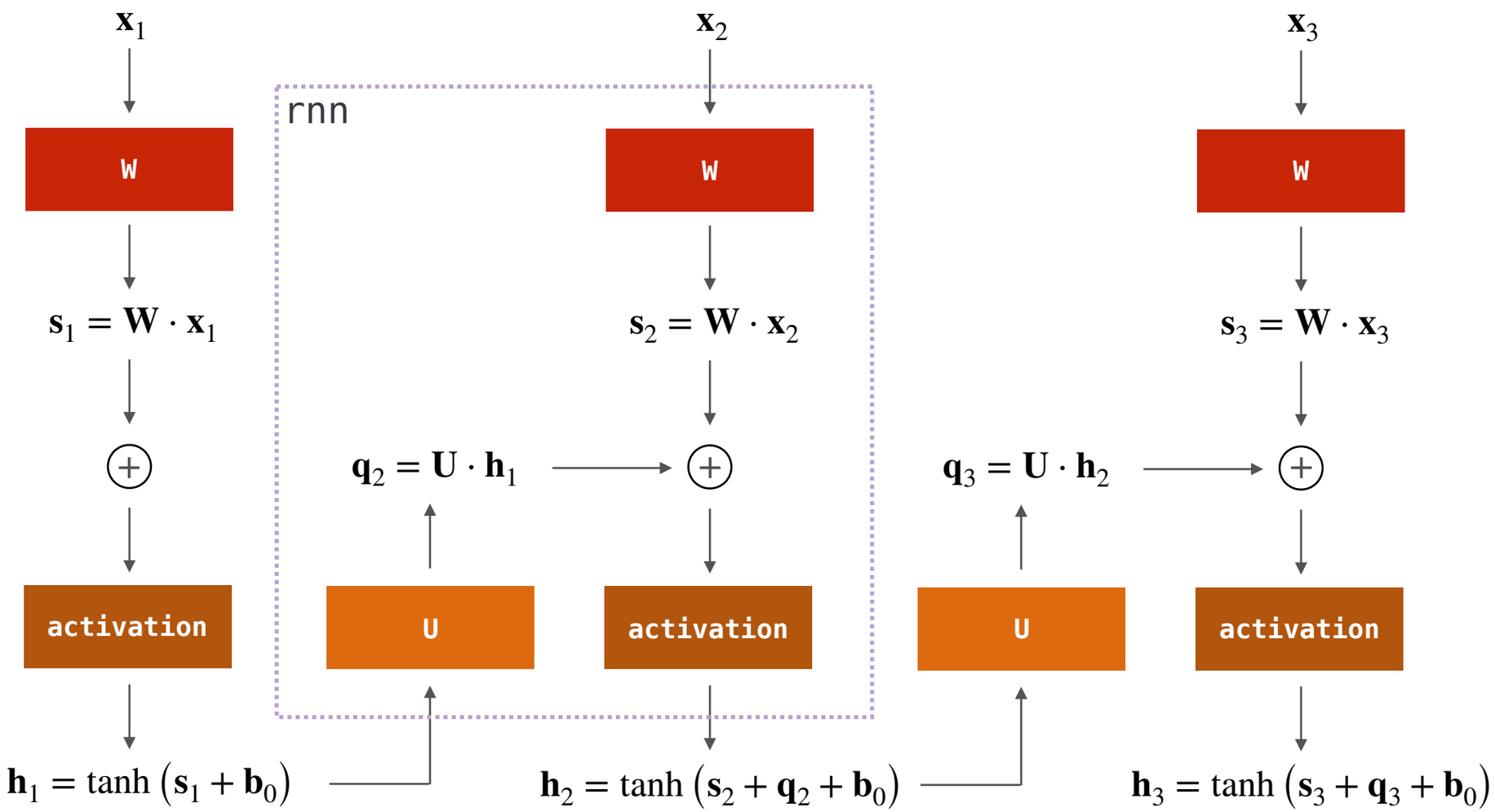
$$\mathbf{x}_1 = [0 \quad 1 \quad 1 \quad 3 \quad 2]$$

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

RNN

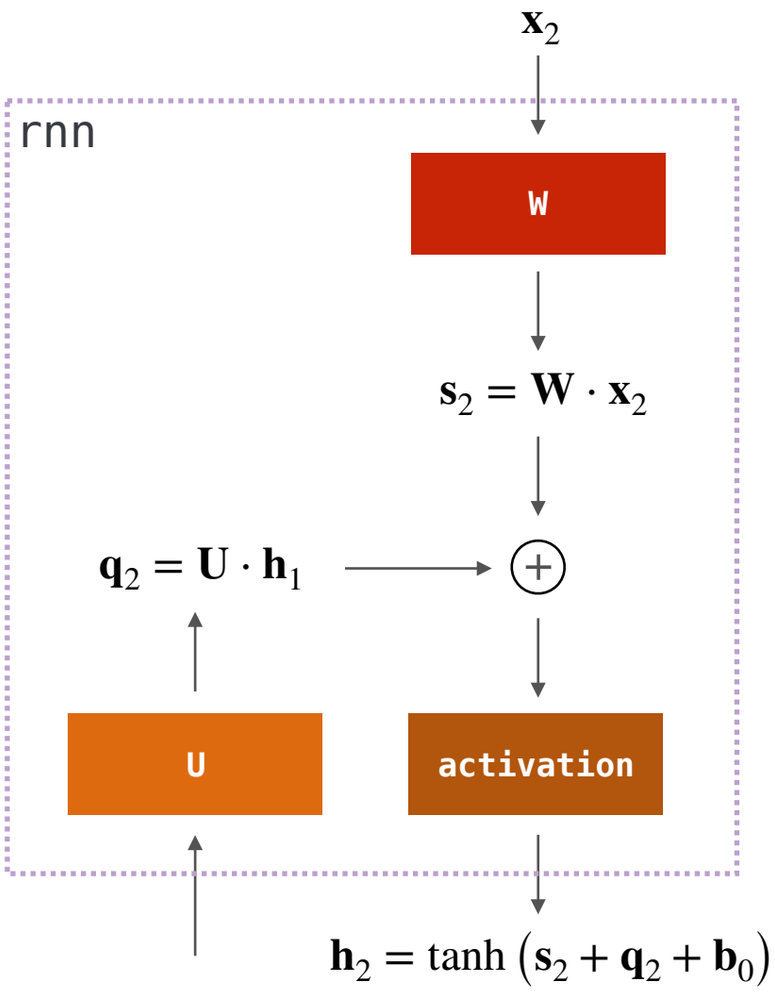
The working

- 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
=================================================================
Total params: 3,137
Trainable params: 3,137
Non-trainable params: 0
_________________________________________________________________
```

→ **?**

→ 32 x 1 + 1 = 33

↑ 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:

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

from $\mathbf{W}$ | from $\mathbf{U}$ | from $\mathbf{b}_0$

```
_________________________________________________________________
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
_________________________________________________________________
```

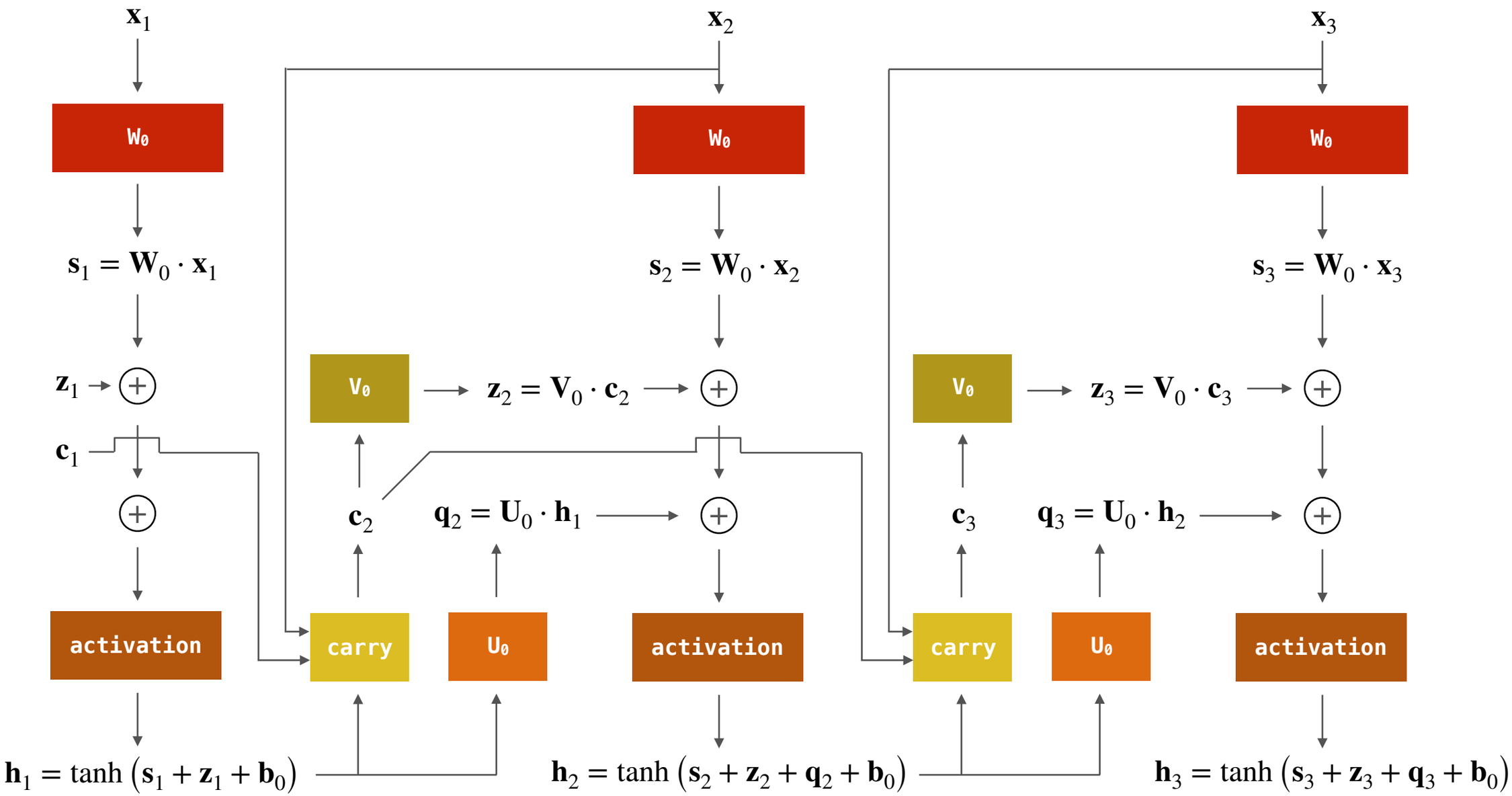
→ 32 x 64 + 32 x 32 + 32 = 2048 + 1024 + 32 = 3104

→ 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

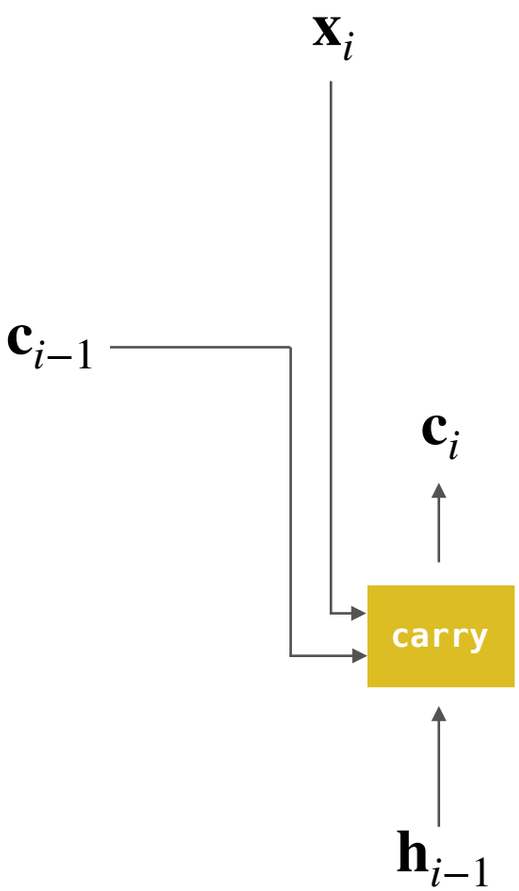
The internal working of 'carry'

- First we define

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

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

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



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

$$\mathbf{c}_i = \mathbf{i}_i \odot \mathbf{g}_i + \mathbf{c}_{i-1} \odot \mathbf{f}_i$$

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         → 32 x 1 + 1 = 33
=================================================================
Total params: 12,449
Trainable params: 12,449
Non-trainable params: 0
_________________________________________________________________
```

↑ 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:

$$p = \left(l_{\text{out}} \times l_{\text{in}}\right) + \left(l_{\text{out}} \times l_{\text{out}}\right) + \left(l_{\text{out}} \times l_{\text{out}} + l_{\text{out}} \times l_{\text{in}} + l_{\text{out}}\right) \times 3 + l_{\text{out}}$$

from	from	from	from
$\mathbf{W}_0$	$\mathbf{U}_0$	$\mathbf{U}_i$, $\mathbf{W}_i$, $\mathbf{b}_i$ $\mathbf{U}_f$, $\mathbf{W}_f$, $\mathbf{b}_f$ $\mathbf{U}_h$, $\mathbf{W}_h$, $\mathbf{b}_h$	$\mathbf{b}_0$

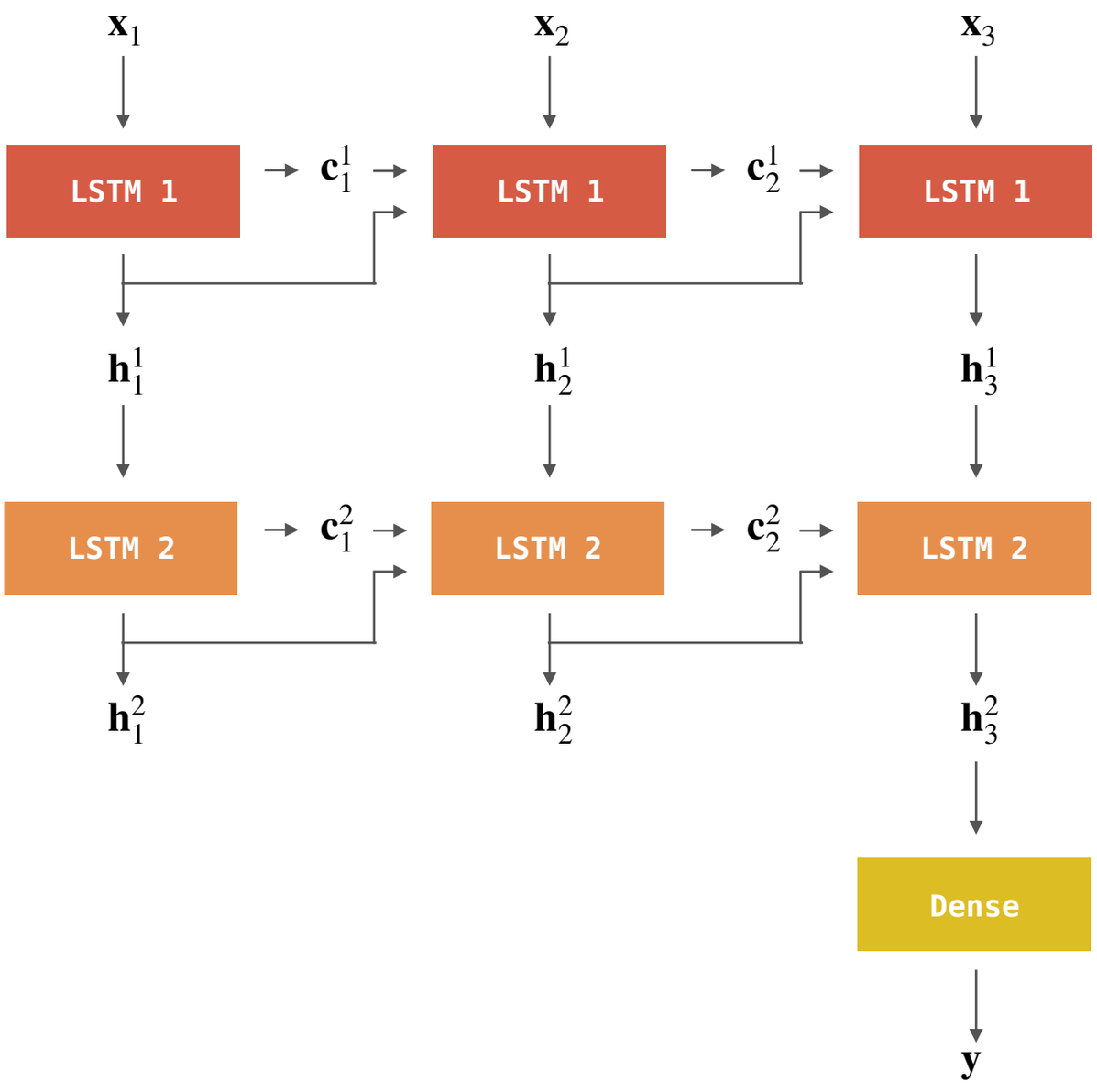
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 16, 64)            0
_________________________________________________________________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

```
  32x64 + 32x32 + (32x32 + 32x64 + 32)x3 + 32
= 2048 + 1024 + 3104x3 + 32
= 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

ECG Signals

Normal and Coronary Artery Disease

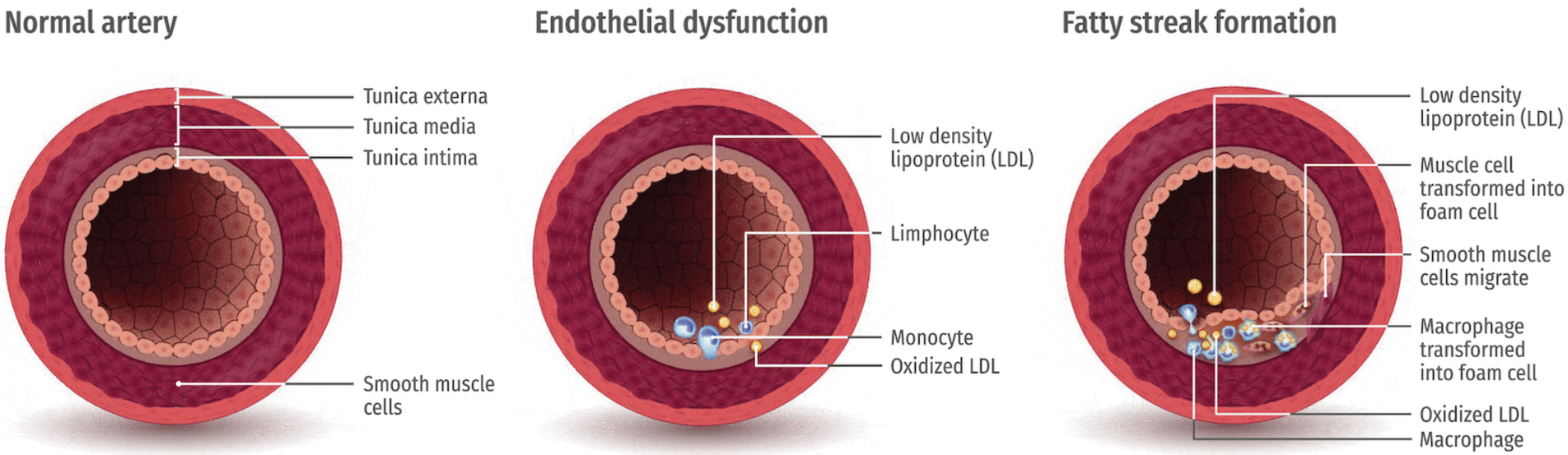
- Lead II ECG from open source PhysioNet database
- Sampling frequency: 257 Hz
- 5 seconds long, consisting of 1285 data points

Source: https://www.sciencedirect.com/science/article/pii/S0010482517304201

ECG Signals

Normal and Coronary Artery Disease

- The pathology of coronary artery disease

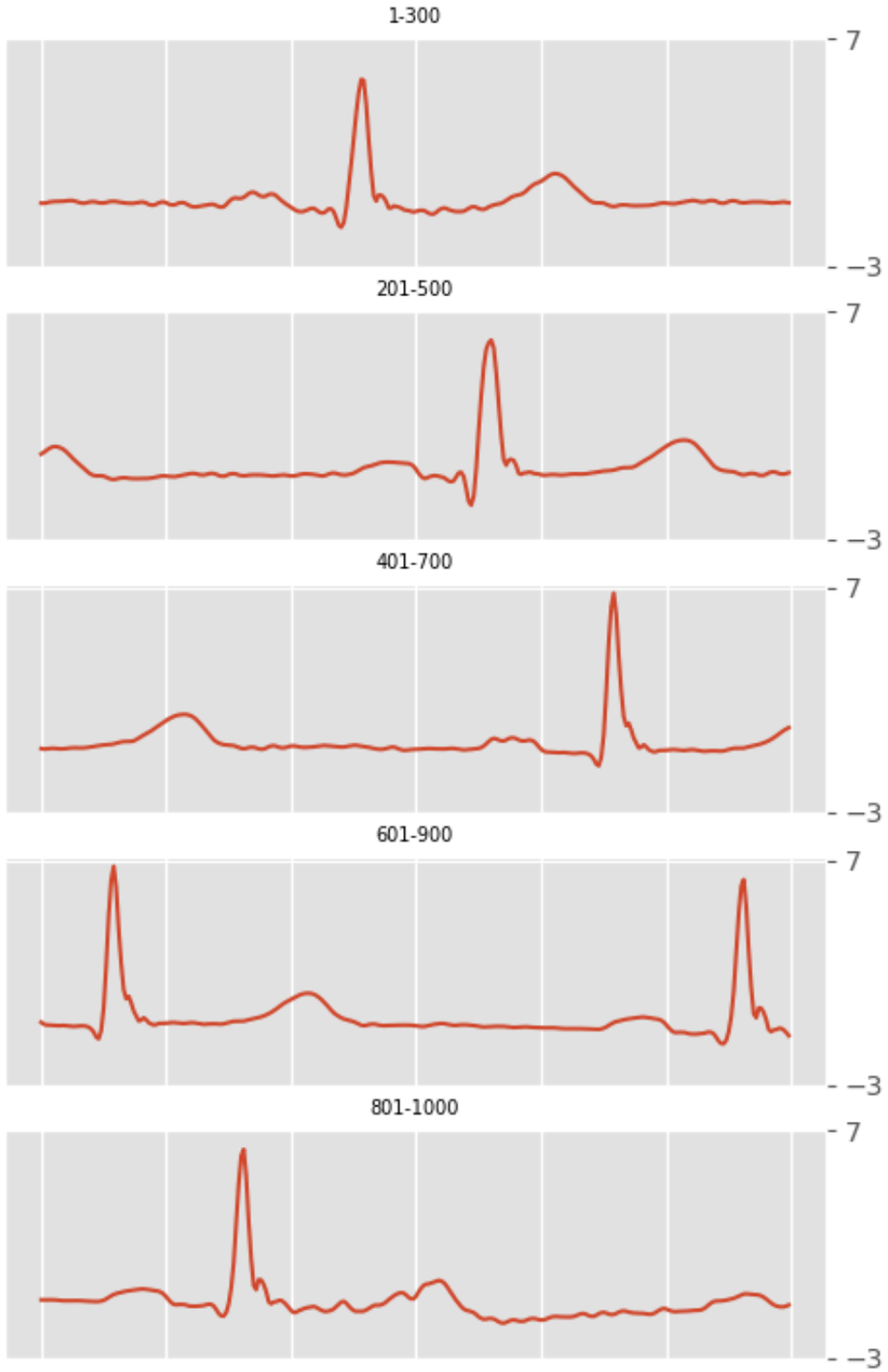


Source: https://www.sciencedirect.com/science/article/abs/pii/S0950705117302769

Normal ECG

in segments

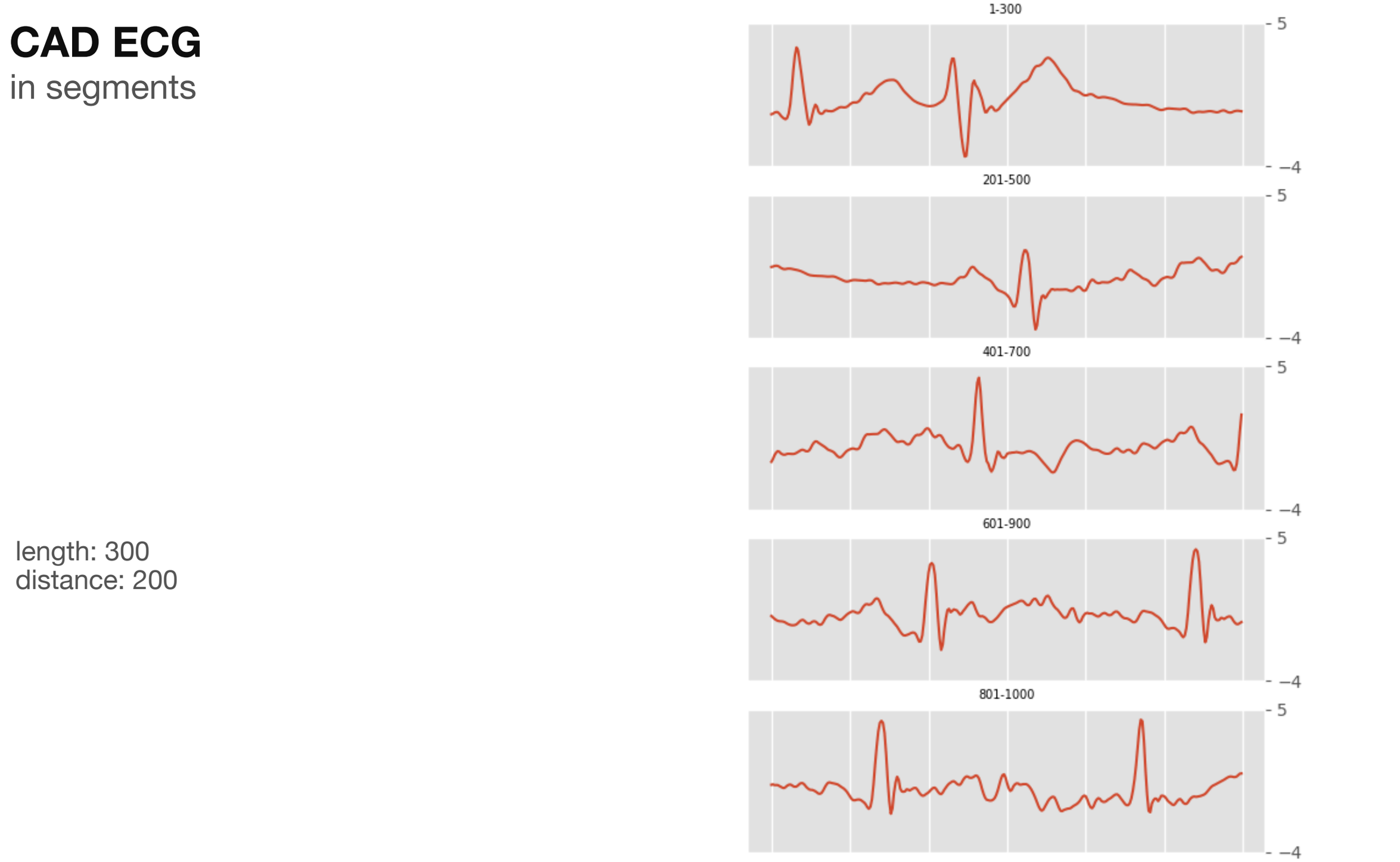
length: 300
distance: 200



CAD ECG

in segments

length: 300
distance: 200



ECG

1. Import libraries

- Import all the things that we are going to use in this problem

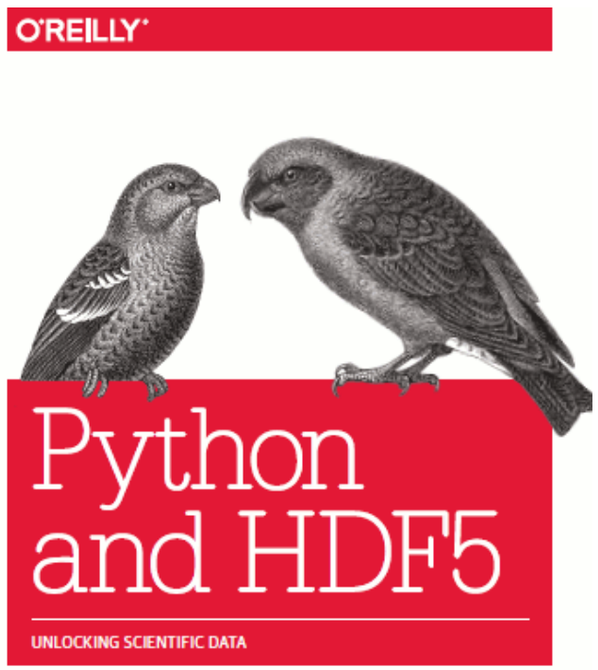
- h5py is required to load ECG data

```
> import h5py
> import numpy as np
> import sklearn.metrics as metrics
> import matplotlib.pyplot as plt

> from tensorflow.keras.callbacks import ModelCheckpoint,CSVLogger
> from tensorflow.keras.models import Model
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.layers import Input
> from tensorflow.keras.layers import LSTM
```

HDF5

Something about h5py ...



- h5py is the pythonic interface to the HDF5 binary data format
- HDF5: Hierarchical Data Format, designed to store and organize large amounts of data
- File extension: .h5, .hdf5
- Can think HDF as a file system within a file, let you organize data hierarchically
- Keras stores model in this format

ECG

2. Matplotlib setup

- Use 'ggplot' style to plot our training and testing result
- The setup uses 'ggplot' style for plot
- Also, for y axis, the labels and ticks put on right rather than left

```
> plt.style.use('ggplot')
> plt.rcParams['ytick.right']      = True
> plt.rcParams['ytick.labelright']= True
> plt.rcParams['ytick.left']       = False
> plt.rcParams['ytick.labelleft'] = False
> plt.rcParams['font.family']      = 'Arial'
```

ECG

3. Data preparation, part 1

- Load the hdf5 file and extract the data within

```
> f       = h5py.File('cad5sec.mat')
> X       = f["data"]
> Y       = f["classLabel"]
```

- if we type 'f' in ipython console, it will return

```
<HDF5 file "cad5sec.mat" (mode r+)>
```

- if we type 'X' and 'Y' respectively in ipython console, it will give

```
<HDF5 dataset "data": shape (1285, 38120), type "<f8">
<HDF5 dataset "classLabel": shape (1, 38120), type "<f8">
```

ECG

3. Data preparation, part 1

- Need to convert 'X' and 'Y' into numpy array and do a transpose on the data

```
> data     = np.array(X)
> data     = np.transpose(data)
> label    = np.array(Y)
> label    = np.transpose(label)
```

Name ▲	Type	Size
data	float64	(38120, 1285)
label	float64	(38120, 1)

- The first 32000 rows are 'normal' ECG, the rest are 'CAD' ECG

```
> nor      = data[0:32000]
> cad      = data[32000:38120]
```

Name ▲	Type	Size
cad	float64	(6120, 1285)
nor	float64	(32000, 1285)

ECG

3. Data preparation, part 2

- Create the function to turn each sample into fixed number of segments (controlled by length and distance

- Note: This function is slightly different with what was shown before. The previous function works on only one sample, this function works on multiple samples at one go

```
> def makeSteps(dat, length, dist):
    width           = dat.shape[1]
    numOfSteps      = int(np.floor((width-length)/dist)+1)

    segments        = np.zeros([dat.shape[0],numOfSteps,length],
                               dtype=dat.dtype)

    for l in range(numOfSteps):
        segments[:,l,:]     = dat[:,(l*dist):(l*dist+length)]

    return segments
```

pre-allocate the array

ECG

3. Data preparation, part 3

- Manually split the data into training and testing set

```
> print('Create dataset...')
> trNor     = nor[0:28800].copy()
> tsNor     = nor[28800:32000].copy()
> trCad     = cad[0:5000].copy()
> tsCad     = cad[5000:6120].copy()
```

```
(28800, 1285)
(3200, 1285)
(5000, 1285)
(1120, 1285)
```

- Set the length and the distance, and convert each sample into segments

```
> length    = 36
> dist      = 24

> print('Finalizing all the data ....')
> trNorS    = makeSteps(trNor, length, dist)
> tsNorS    = makeSteps(tsNor, length, dist)
> trCadS    = makeSteps(trCad, length, dist)
> tsCadS    = makeSteps(tsCad, length, dist)
```

```
(28800, 53, 36)
(3200, 53, 36)
(5000, 53, 36)
(1120, 53, 36)
```

ECG

3. Data preparation, part 3

- Combining all together

```
> trDat     = np.vstack([trNorS,trCadS])
> tsDat     = np.vstack([tsNorS,tsCadS])

> trLbl     = np.vstack([np.zeros([trNorS.shape[0],1]),
                         np.ones([trCadS.shape[0],1])])
> tsLbl     = np.vstack([np.zeros([tsNorS.shape[0],1]),
                         np.ones([tsCadS.shape[0],1])])
```

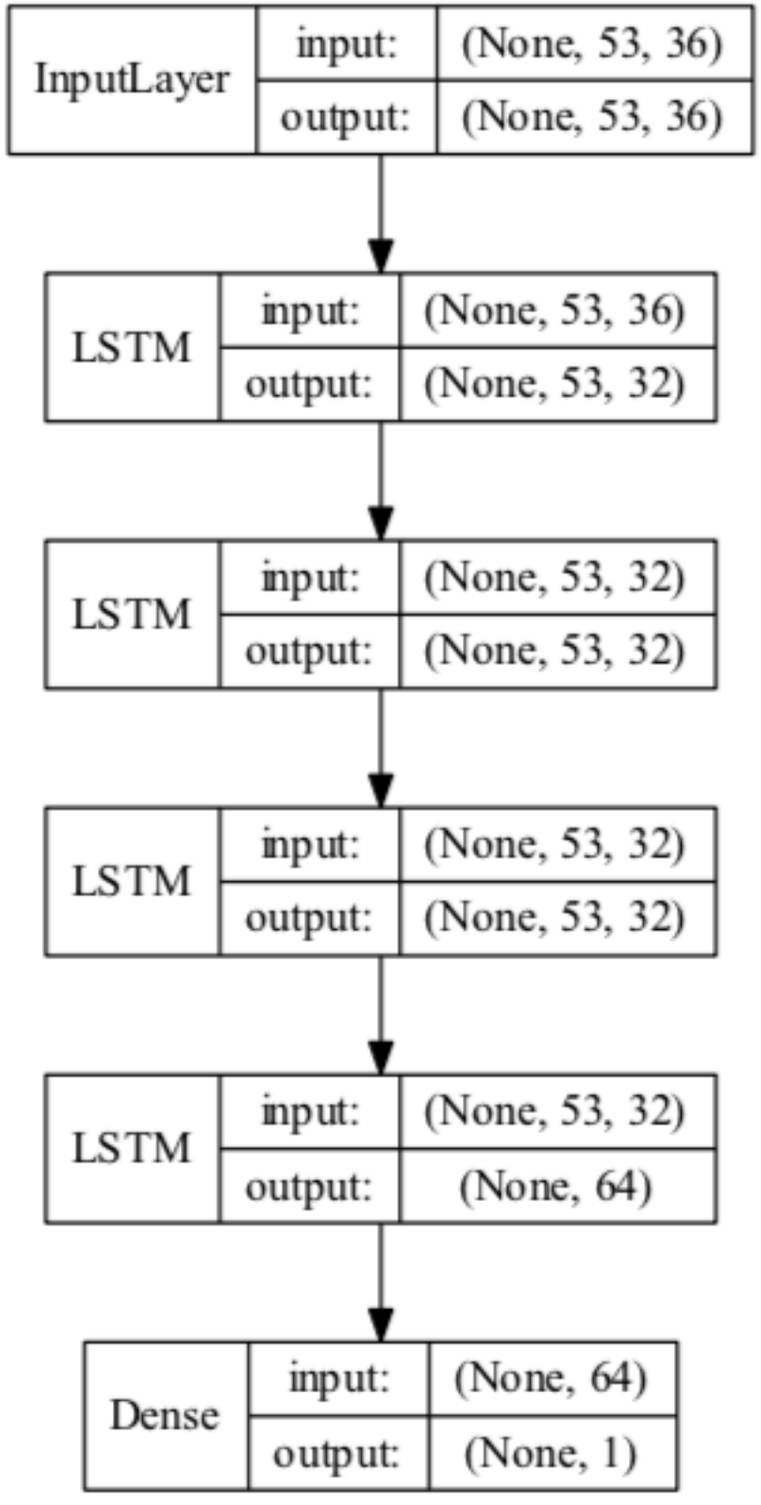
'Normal' set to 0
'Cad' set to 1

trDat	float64	(33800, 53, 36)
trLbl	float64	(33800, 1)

tsDat	float64	(4320, 53, 36)
tsLbl	float64	(4320, 1)

The net

4. Define model



The net

4. Define model

```
> seed = 7
> np.random.seed(seed)
> modelname   = 'wks3_2_1'
> def createModel():
    inputs= Input(shape=(trDat.shape[1],length))
    y     = LSTM(32,
                 return_sequences=True,
                 dropout=0.25,
                 recurrent_dropout=0.25)(inputs)
    y     = LSTM(32,
                 return_sequences=True,
                 dropout=0.5,
                 recurrent_dropout=0.25)(y)
y     = LSTM(32,
             return_sequences=True,
             dropout=0.25)(y)
y     = LSTM(64, dropout=0.25)(y)
y     = Dense(1, activation='sigmoid')(y)

model = Model(inputs=inputs,outputs=y)
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
return model
```

- Note 1: The dropout layer doesn't work for LSTM / RNN. If want to do dropout, set value to the 'dropout' argument

- Note 2: We can apply dropout on the recurrent part in LSTM (the U_0)

- Note 3: We use 'binary_crossentropy', not 'categorical_crossentropy', because we have only 1 output, and the value is either 1 or 0

The net

4. Define model

- 'model' for training; 'modelGo' for final evaluation

```
> model       = createModel()
> modelGo     = createModel()

> model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 53, 36)            0
_________________________________________________________________
lstm (LSTM)                  (None, 53, 32)            8832
_________________________________________________________________
lstm_1 (LSTM)                (None, 53, 32)            8320
_________________________________________________________________
lstm_2 (LSTM)                (None, 53, 32)            8320
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                24832
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 50,369
Trainable params: 50,369
Non-trainable params: 0
_________________________________________________________________
```

The net

4. Define model

- Create checkpoints to save model during training and save training data into csv

- ‘monitor’ can be ‘val_acc’ or ‘val_loss’

- When set to ‘val_acc’, ‘mode’ must be ‘max’; when set to ‘val_loss’, ‘mode’ must be ‘min’

```
> filepath        = modelname + ".hdf5"
> checkpoint      = ModelCheckpoint(filepath,
                                    monitor='val_acc',
                                    verbose=0,
                                    save_best_only=True,
                                    mode='max')


> csv_logger      = CSVLogger(modelname + '.csv')
> callbacks_list  = [checkpoint,csv_logger]
```

The net

5. Train model

- The line for training

```
> model.fit(trDat,
            trLbl,
            validation_data=(tsDat, tsLbl),
            epochs=40,
            batch_size=128,
            shuffle=True,
            callbacks=callbacks_list)
```

```
Train on 33800 samples, validate on 4320 samples
Epoch 1/40
33800/33800 [==============================] - 100s 3ms/sample - loss: 0.2614 - acc: 0.9047 - val_loss: 1.0719 - val_acc: 0.7426
Epoch 2/40
33800/33800 [==============================] - 96s 3ms/sample - loss: 0.1154 - acc: 0.9616 - val_loss: 0.7840 - val_acc: 0.7681
Epoch 3/40
33800/33800 [==============================] - 96s 3ms/sample - loss: 0.0912 - acc: 0.9705 - val_loss: 0.7976 - val_acc: 0.7898
Epoch 4/40
33800/33800 [==============================] - 97s 3ms/sample - loss: 0.0797 - acc: 0.9750 - val_loss: 0.6873 - val_acc: 0.8343
Epoch 5/40
33800/33800 [==============================] - 97s 3ms/sample - loss: 0.0716 - acc: 0.9772 - val_loss: 0.8630 - val_acc: 0.8188
......
```

The net

6. Result

- Classification result

```
Best accuracy (on testing dataset): 88.38%
             precision    recall  f1-score   support

     Normal     0.8766    0.9812    0.9260      3200
        CAD     0.9187    0.6054    0.7298      1120

avg / total     0.8875    0.8838    0.8751      4320
```

- Confusion matrix

```
[[3140   60]
 [ 442  678]]
```

Feature extraction

Not necessary

- In previous examples, we simply feed the original data (although with a bit of re-arrangement) into the net
- Feature extraction is missing!
- Without feature extraction, learning is slower and not optimal
- Should perform rounds of feature extraction, reduce the amount of recurrent calculation so that training is faster

Conv1D

size 3

	Input					Kernel					Intermediate Output		Output	
	1	0	1	0		0	1	0	1		7		7	7
	1	1	3	2		0	0	2	0		6		6	13
	1	1	0	1		0	1	0	0		8		8	10
	2	3	2	1										
	0	2	0	1										
	1	0	1	0		2	1	0	0		7			
	1	1	3	2		0	0	0	1		13			
	1	1	0	1		0	3	0	0		10			
	2	3	2	1										
	0	2	0	1										

Kernel size (in one direction): 3
Number of filters: 2

Conv 1D

Determine the output size

- M_r, M_c: Number of rows and columns in the output, respectively
- W_r: Number of rows in the input
- F_r: Filter size (along the rows)
- P_r: Amount of padding (along the rows)
- S_r,: Stride (along the rows)
- N_F,: The number of filters

$$M_r = \left[\frac{W_r - F_r + 2P_r}{S_r} \right] + 1$$

$$M_c = N_F$$

Conv 1D

Calculate the parameters

- For each filter, the number of parameters is

$$W_c \times F_r + 1$$

- Thus, for N_F amount of filters, the number of parameters is

$$N_F \times \left(W_c \times F_r + 1\right)$$

M_r, M_c: Number of rows and columns in the output, respectively

W_c: Number of columns in the input

F_r: Filter size (along the rows)

N_F,: The number of filters

MaxPooling1D

size 3

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2		Input							Output					
3														
4		1	0	1	0				1	1	3	2		
5		0	1	3	2				2	3	2	1		
6		1	1	2	1									
7		2	3	0	0									
8		0	2	0	1									
9														
10														

Kernel size (in one direction): 2
Stride (in one direction): 2

Stacked Conv1D LSTM

ECG classification in Keras

```
length    = 24
dist      = 6
```

```
> def createModel():
    inputs  = Input(shape=(trDat.shape[1],length))
    y       = Conv1D(32, 5, activation='relu')(inputs)
    y       = Dropout(0.25)(y)
    y       = Conv1D(32, 5, activation='relu')(y)
    y       = MaxPooling1D(2)(y)
    y       = Conv1D(48, 5, activation='relu')(y)
    y       = Dropout(0.5)(y)
    y       = Conv1D(48, 5, activation='relu')(y)
    y       = MaxPooling1D(2)(y)
    y       = Conv1D(64, 5, activation='relu')(y)
    y       = Dropout(0.5)(y)
    y       = Conv1D(64, 5, activation='relu')(y)
    y       = MaxPooling1D(2)(y)
    y       = LSTM(8,
                   return_sequences=True,
                   dropout=0.5,
                   recurrent_dropout=0.5)(y)
    y       = LSTM(4,
                   return_sequences=True,
                   dropout=0.5,
                   recurrent_dropout=0.5)(y)
    y       = LSTM(2)(y)
    y       = Dense(1, activation='sigmoid')(y)

    model = Model(inputs=inputs,outputs=y)
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

Stacked Conv1D LSTM

ECG classification in Keras

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 211, 24)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 207, 32)           3872
_________________________________________________________________
dropout (Dropout)            (None, 207, 32)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 203, 32)           5152
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 101, 32)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 97, 48)            7728
_________________________________________________________________
dropout_1 (Dropout)          (None, 97, 48)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 93, 48)            11568
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 46, 48)            0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 42, 64)            15424
_________________________________________________________________
dropout_2 (Dropout)          (None, 42, 64)            0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 38, 64)            20544
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 19, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 19, 8)             2336
_________________________________________________________________
lstm_1 (LSTM)                (None, 19, 4)             208
_________________________________________________________________
lstm_2 (LSTM)                (None, 2)                 56
_________________________________________________________________
dense (Dense)                (None, 1)                 3
=================================================================
Total params: 66,891
Trainable params: 66,891
```

Stacked Conv1D LSTM

The result

- Classification result

```
Best accuracy (on testing dataset): 96.44%
             precision    recall  f1-score   support

     Normal     0.9997    0.9522    0.9754      3200
        CAD     0.8797    0.9991    0.9356      1120

avg / total     0.9686    0.9644    0.9651      4320
```

- Confusion matrix

```
[[3047  153]
 [   1 1119]]
```