Logistic Regression

Outline

- Task Introduction
- Feature extraction
- Code
- Improvement Tips

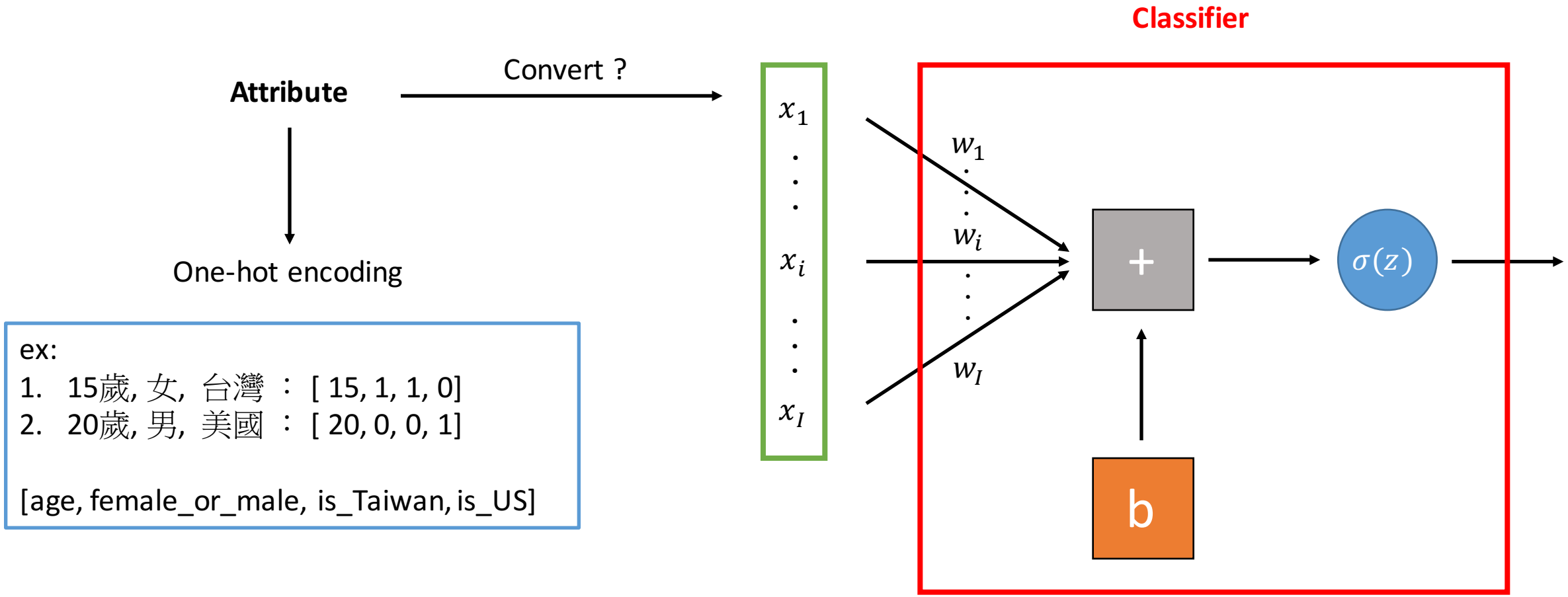
Task Introduction

- Binary Classification
- Adult Data Set

Attribute, Input → **Salary condition, Output**

age	workclass	fnlwgt	education	education_nur	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_we	native_country	
39	State-gov	77516	Bachelors	13	Never-marrie	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
50	Self-emp-not-	83311	Bachelors	13	Married-civ-s	Exec-manage	Husband	White	Male	0	0	13	United-States	<=50K
38	Private	215646	HS-grad	9	Divorced	Handlers-clea	Not-in-family	White	Male	0	0	40	United-States	<=50K
53	Private	234721	11th	7	Married-civ-s	Handlers-clea	Husband	Black	Male	0	0	40	United-States	<=50K
28	Private	338409	Bachelors	13	Married-civ-s	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
37	Private	284582	Masters	14	Married-civ-s	Exec-manage	Wife	White	Female	0	0	40	United-States	<=50K
49	Private	160187	9th	5	Married-spou	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica	<=50K
52	Self-emp-not-	209642	HS-grad	9	Married-civ-s	Exec-manage	Husband	White	Male	0	0	45	United-States	>50K
31	Private	45781	Masters	14	Never-marrie	Prof-specialty	Not-in-family	White	Female	14084	0	50	United-States	>50K
42	Private	159449	Bachelors	13	Married-civ-s	Exec-manage	Husband	White	Male	5178	0	40	United-States	>50K
37	Private	280464	Some-college	10	Married-civ-s	Exec-manage	Husband	Black	Male	0	0	80	United-States	>50K
30	State-gov	141297	Bachelors	13	Married-civ-s	Prof-specialty	Husband	Asian-Pac-Isl	Male	0	0	40	India	>50K
23	Private	122272	Bachelors	13	Never-marrie	Adm-clerical	Own-child	White	Female	0	0	30	United-States	<=50K
32	Private	205019	Assoc-acdm	12	Never-marrie	Sales	Not-in-family	Black	Male	0	0	50	United-States	<=50K

Feature extraction



Feature

Feature: One-hot encoding

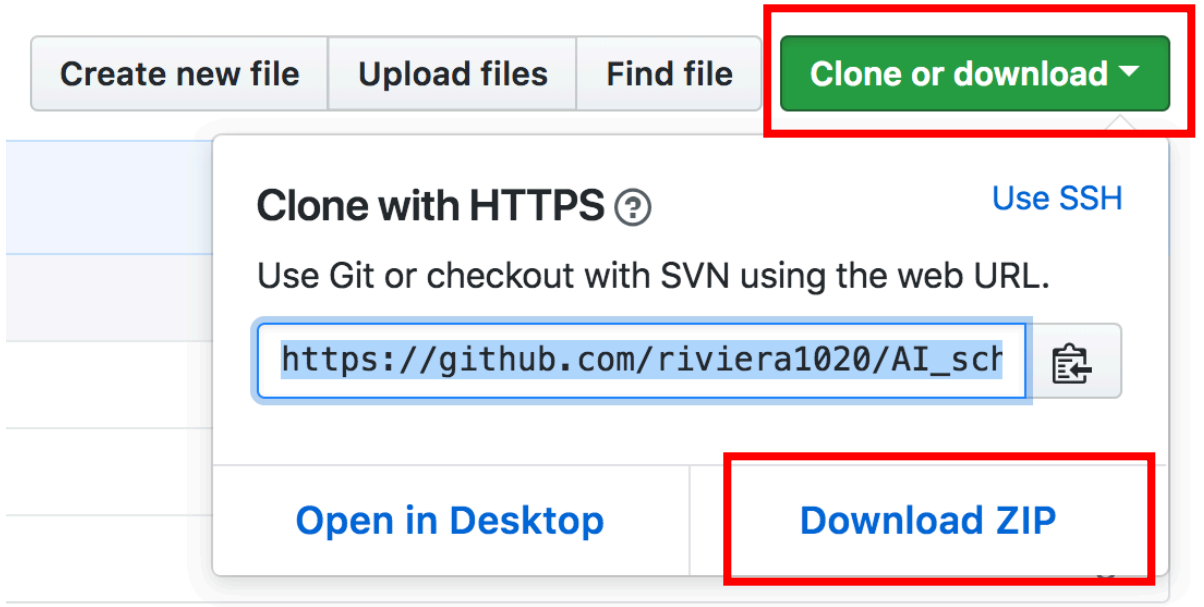
Label:
0 : <= 50k
1 : > 50K

```
age,fnlwgt,sex,capital_gain,capital_loss,hours_per_week, Federal-gov, Local-gov, Never-worked, Private, Self-
39,77516,1,2174,0,40,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,
50,83311,1,0,0,13,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
38,215646,1,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
53,234721,1,0,0,40,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
28,338409,0,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
37,284582,0,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
49,160187,0,0,0,16,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
52,209642,1,0,0,45,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
31,45781,0,14084,0,50,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0
42,159449,1,5178,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
37,280464,1,0,0,80,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
30,141297,1,0,0,40,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
23,122272,0,0,0,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
32,205019,1,0,0,50,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
40,121772,1,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
34,245487,1,0,0,45,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
25,176756,1,0,0,35,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
32,186824,1,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
38,28887,1,0,0,50,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
43,292175,0,0,0,45,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
40,193524,1,0,0,60,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
54,302146,0,0,0,20,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
35,76845,1,0,0,40,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
43,117037,1,0,2042,40,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
59,109015,0,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
56,216851,1,0,0,40,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
19,168294,1,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
54,180211,1,0,0,60,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
39,367260,1,0,0,80,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
49,193366,1,0,0,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
23,190709,1,0,0,52,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
20,266015,1,0,0,44,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
45,386940,1,0,1408,40,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
N  ± master  [1:X_train ]                                  unix | utf-8 | no ft   0%   1:1
[1:X_train ]
```

```
label
0
0
0
0
0
0
0
1
1
1
1
1
0
0
1
0
0
0
0
1
1
0
0
0
0
1
0
1
0
0
0
0
0
N  ± master  [1:Y_train ]
[1:Y_train ]
```

Download

- Link
  - Download zip
  - git clone https://github.com/riviera1020/AI_school_logistic_regression.git
- File
  - main.py : 主程式
  - answer.py : 正確答案
  - feature/ : 幫大家抽好啦
    - X_train, Y_train : training data
    - X_test, ans.csv : testing data
  - logitsic_params/ : 存model



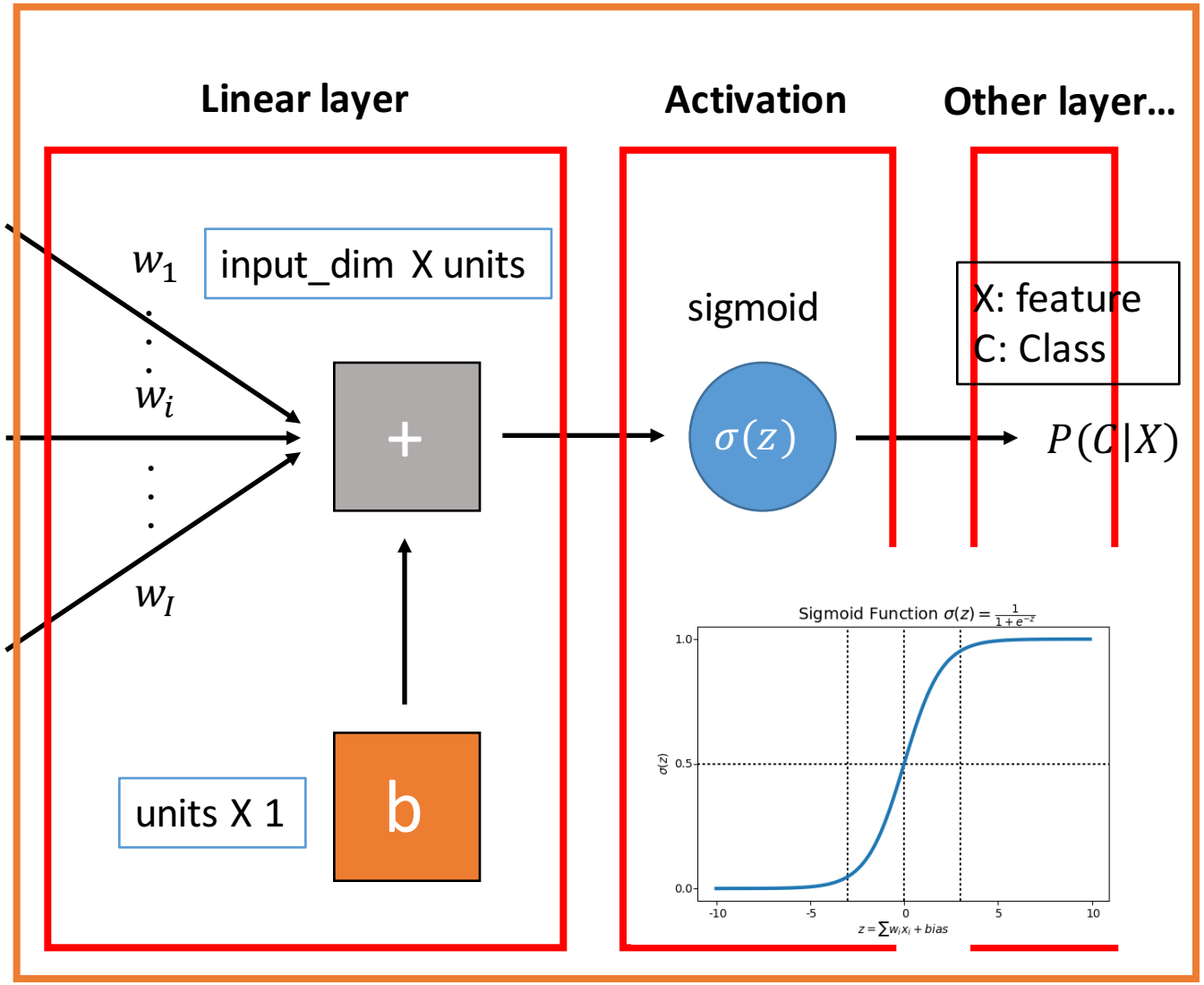
Code

- 用編輯器開main.py
  - Vim, Sublime text, NotePad
- 流程
  - Define model architecture
  - Define model optimizer
  - Compile model
  - Training
  - Inference
- Three step
  - Step 1: Function Set
  - Step 2: Goodness of a Function
  - Step 3: Find the best function

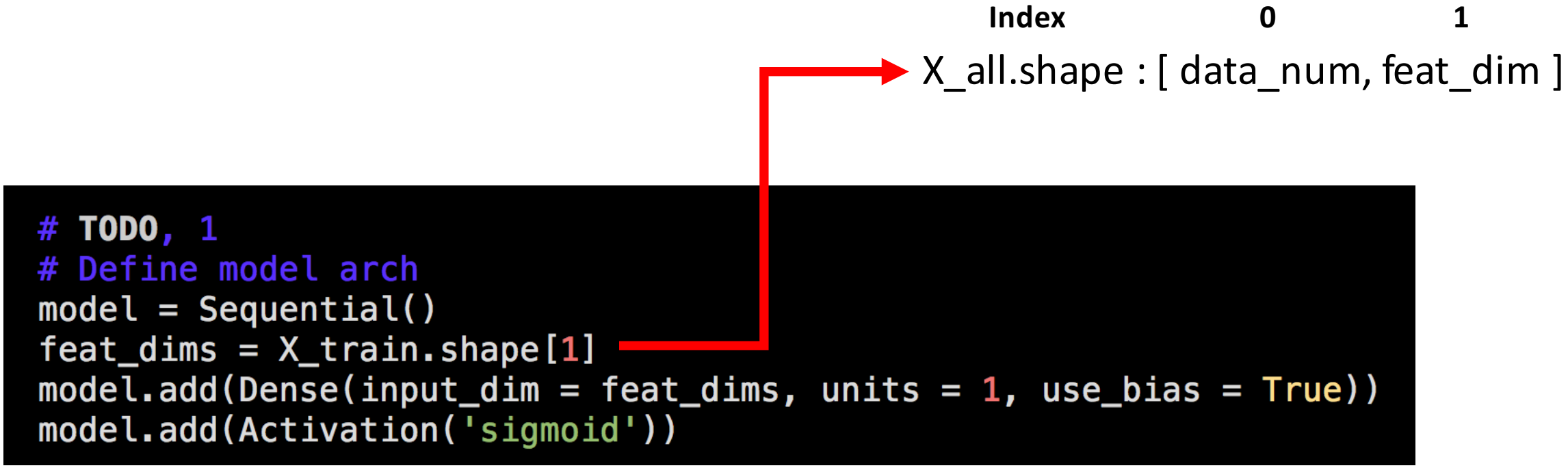
```
def train(X_train, Y_train):

    # TODO, 1
    # Define model arch

    # TODO, 2
    # Define optimizer

    # TODO, 3
    # Compile model

    # TODO, 4
    # Start training

def infer(X_test, Y_test):

    # TODO, 5
    # load and inference
```

Define model architecture

- 容器：Sequential()
  - model = Seuqential()
  - model.add(layer_object)
- Linear layer
  - Dense(input_dim,units,use_bias)
    - input_dim: feature的維度
    - units: 輸出的維度
    - use_bias: True
- Activation
  - Activation('sigmoid')



Define model architecture - Code



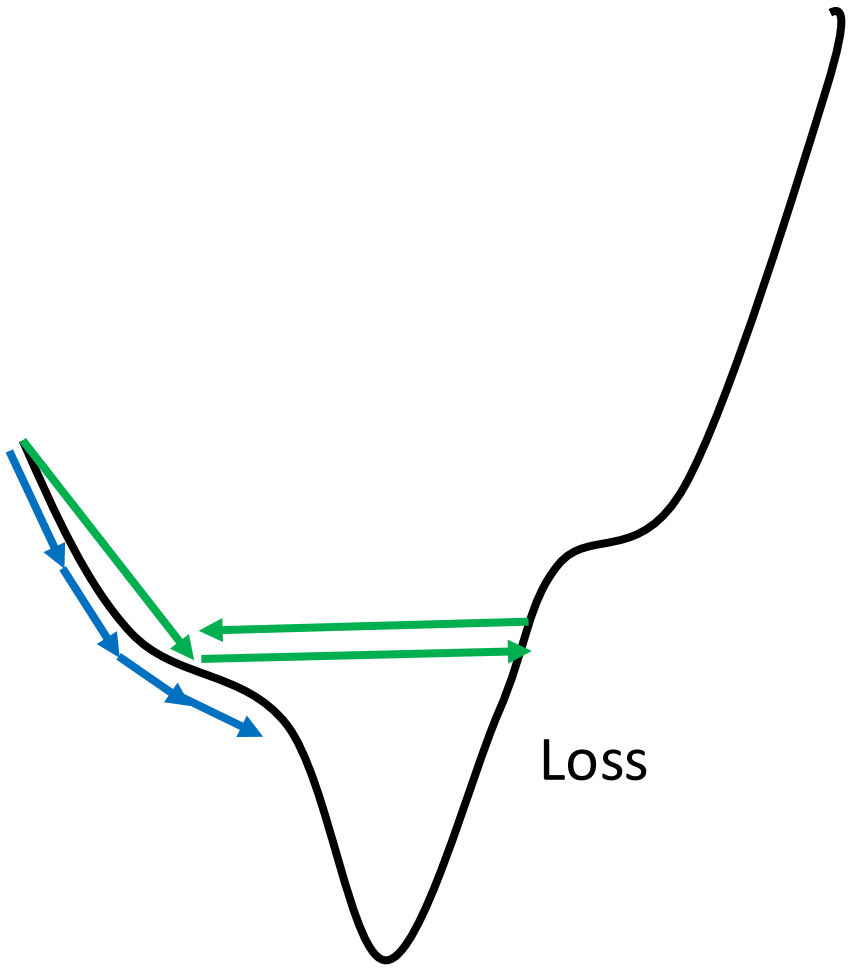
Step 1: Function Set

Define model optimizer

- Stochastic Gradient Descent

```
# TODO, 2
# Define optimizer
sgd = optimizers.SGD(lr=0.001)
```

- Arguments
  - lr : learning rate, 走多大步



Compile model

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

Batch size　　predict　　answer

```
# TODO, 3
# Compile model
model.compile(loss = 'binary_crossentropy',
              optimizer = sgd,
              metrics = ['accuracy'])
```

- Arguments
  - loss : cross entropy ⟶ Step 2: Goodness of a Function
  - optimizer : sgd ⟶ Step 3: Find the best function
  - metrics : 計算accuracy的方式

Start training

看data幾次

```
# TODO, 4
# Start training
model.fit(X_train, Y_train, epochs = 10)
model.save('./logistic_params/logistic.h5')
```

Training Code

```
def train(X_train, Y_train):

    # TODO, 1
    # Define model arch
    model = Sequential()
    feat_dims = X_train.shape[1]
    model.add(Dense(input_dim = feat_dims, units = 1, use_bias = True))
    model.add(Activation('sigmoid'))

    # TODO, 2
    # Define optimizer
    sgd = optimizers.SGD(lr=0.001)

    # TODO, 3
    # Compile model
    model.compile(loss = 'binary_crossentropy',
                  optimizer = sgd,
                  metrics = ['accuracy'])

    # TODO, 4
    # Start training
    model.fit(X_train, Y_train, epochs = 10)
    model.save('./logistic_params/logistic.h5')
```

Inference

```
def infer(X_test, Y_test):

    # TODO, 5
    # load and inference
    model = load_model('./logistic_params/logistic.h5')
    loss, acc = model.evaluate(X_test, Y_test)

    print('Accuracy : ' + str(acc))
```

Run Code

Type commands in command line

Train:

python main.py --train

Test:

python main.py --infer

Try to adjust the following parameters:

- epochs
- lr

```
:
name: GeForce GTX 960 major: 5 minor: 2 memoryClockRate(GHz): 1.291
pciBusID: 0000:01:00.0
totalMemory: 3.94GiB freeMemory: 3.25GiB
2018-08-19 17:57:00.091913: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1227] Device peer to peer matrix
2018-08-19 17:57:00.091925: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1233] DMA: 0 1
2018-08-19 17:57:00.091930: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1243] 0:   Y N
2018-08-19 17:57:00.091933: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1243] 1:   N Y
2018-08-19 17:57:00.091941: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1312] Adding visible gpu devices: 0,
 1
2018-08-19 17:57:00.417698: I tensorflow/core/common_runtime/gpu/gpu_device.cc:993] Creating TensorFlow device (/jo
b:localhost/replica:0/task:0/device:GPU:0 with 5641 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1060 6
GB, pci bus id: 0000:05:00.0, compute capability: 6.1)
2018-08-19 17:57:00.437818: I tensorflow/core/common_runtime/gpu/gpu_device.cc:993] Creating TensorFlow device (/jo
b:localhost/replica:0/task:0/device:GPU:1 with 2978 MB memory) -> physical GPU (device: 1, name: GeForce GTX 960, p
ci bus id: 0000:01:00.0, compute capability: 5.2)
32561/32561 [==============================] - 2s 62us/step - loss: 0.4722 - acc: 0.7804
Epoch 2/10
32561/32561 [==============================] - 1s 41us/step - loss: 0.3488 - acc: 0.8439
Epoch 3/10
32561/32561 [==============================] - 1s 40us/step - loss: 0.3345 - acc: 0.8465
Epoch 4/10
32561/32561 [==============================] - 1s 41us/step - loss: 0.3291 - acc: 0.8484
Epoch 5/10
32561/32561 [==============================] - 1s 40us/step - loss: 0.3261 - acc: 0.8492
Epoch 6/10
32561/32561 [==============================] - 1s 40us/step - loss: 0.3241 - acc: 0.8500
Epoch 7/10
32561/32561 [==============================] - 1s 40us/step - loss: 0.3228 - acc: 0.8503
Epoch 8/10
32561/32561 [==============================] - 1s 41us/step - loss: 0.3217 - acc: 0.8516
Epoch 9/10
32561/32561 [==============================] - 1s 40us/step - loss: 0.3211 - acc: 0.8511
Epoch 10/10
32561/32561 [==============================] - 1s 41us/step - loss: 0.3204 - acc: 0.8516
riviera1020@531:~/AI_School$
```

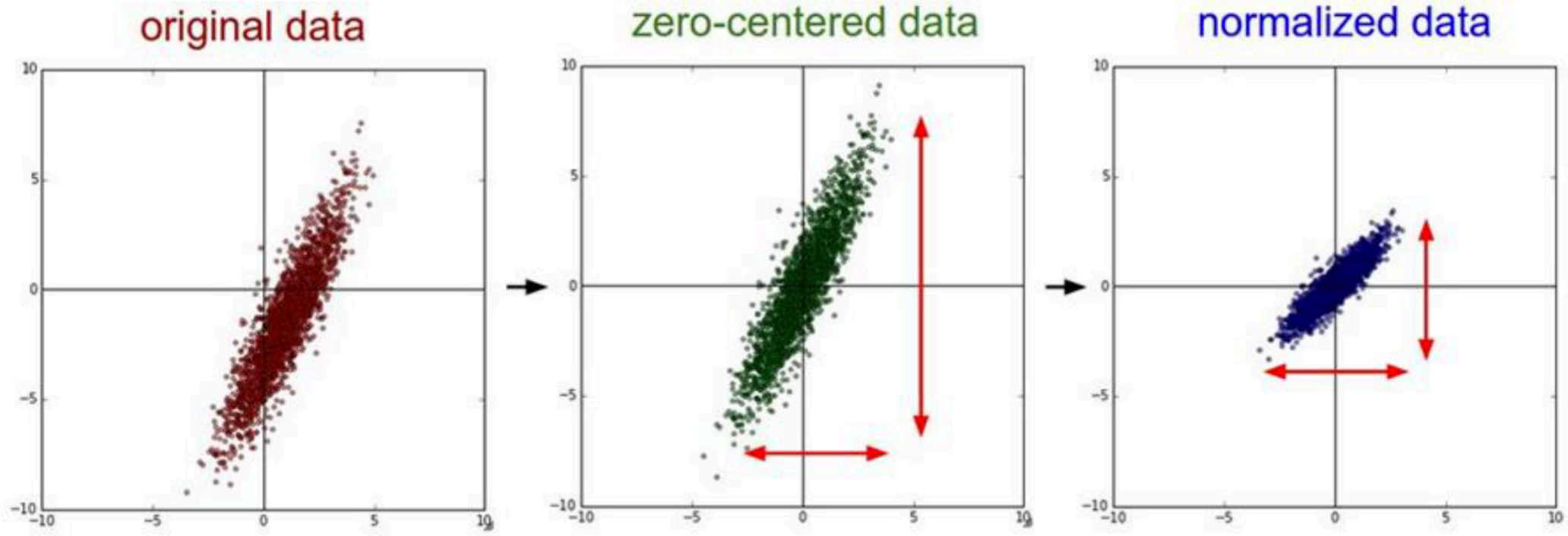
Improvement tips

- 你其實用過了
  - Normalize
- Overfitting
  - Early stop
- Gradient Descent
  - Momentum
  - Adaptive Learning Rate
- Deep Network
  - Other activation

Normalize

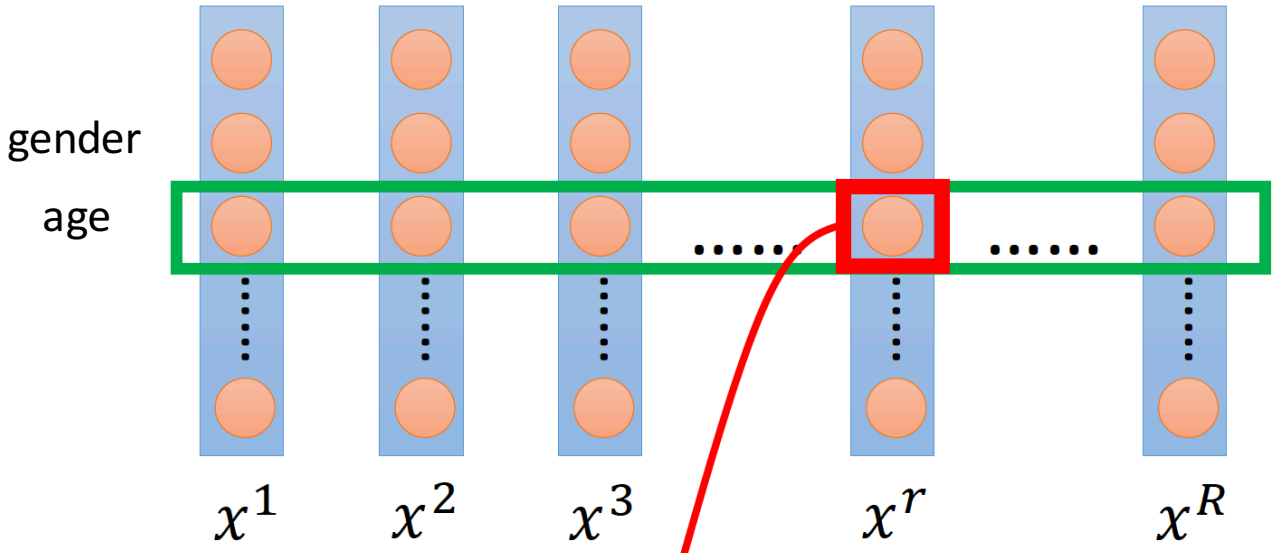
Attribute:
Age, captial_gain <-> one-hot

ex:
1. 15歲, 女, 台灣 ： [15, 1, 1, 0]
2. 20歲, 男, 美國 ： [20, 0, 0, 1]



Source of figure: http://cs231n.github.io/neural-networks-2/

Normalize - how



For each dimension i:

mean: m_i

standard deviation: σ_i

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

The means of all dimensions are 0, and the variances are all 1

Ref: 宏毅教授投影片

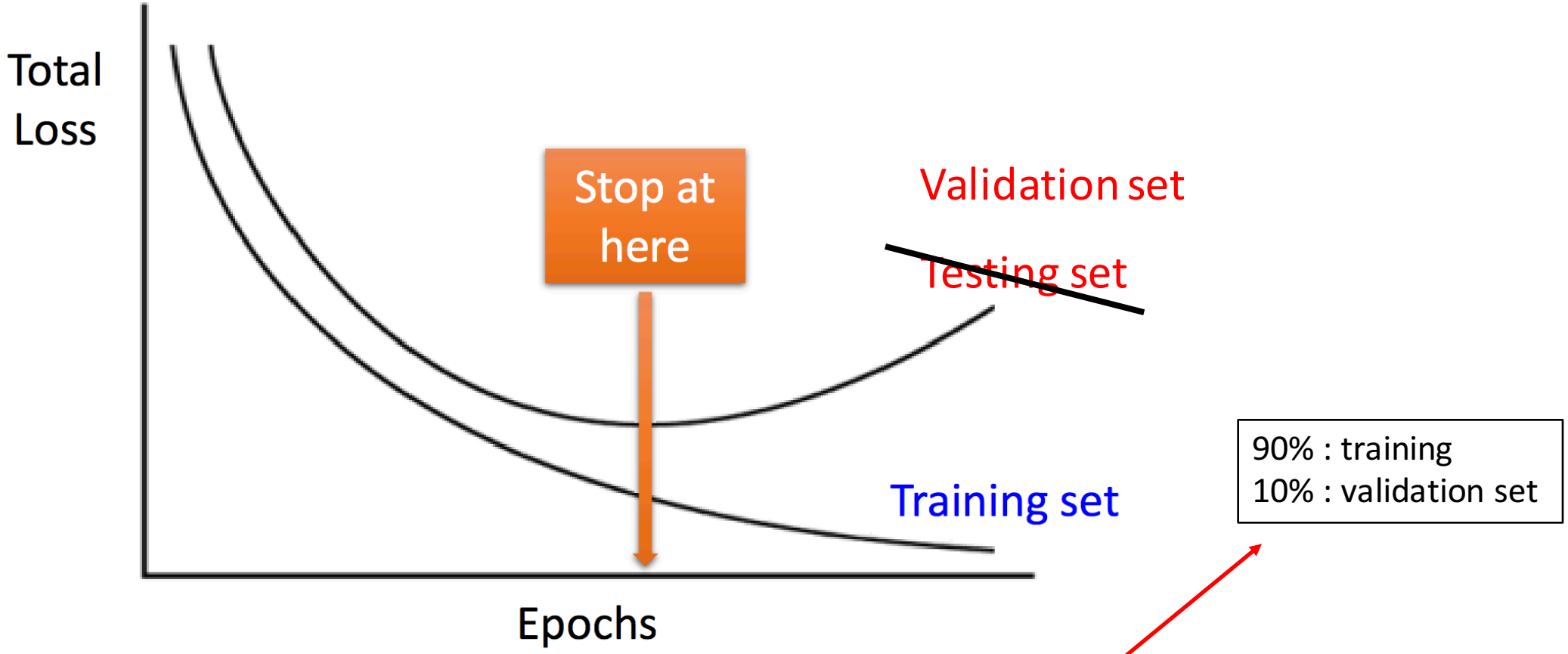
關掉Normalize

```
def main(opts):
    # Load feature and label
    X_train, Y_train, X_test, Y_test = load_data(opts.train_data_path,
                                                 opts.train_label_path,
                                                 opts.test_data_path,
                                                 opts.test_label_path)

    # Normalization
    #X_train, X_test = normalize(X_train, X_test)

    # To train or to infer
    if opts.train:
        train(X_train, Y_train)
    elif opts.infer:
        infer(X_test, Y_test)
    else:
        print("Error: Argument --train or --infer not found")
    return
```

Comment掉試試看

Overfitting – early stop

Implement Earlystop in keras

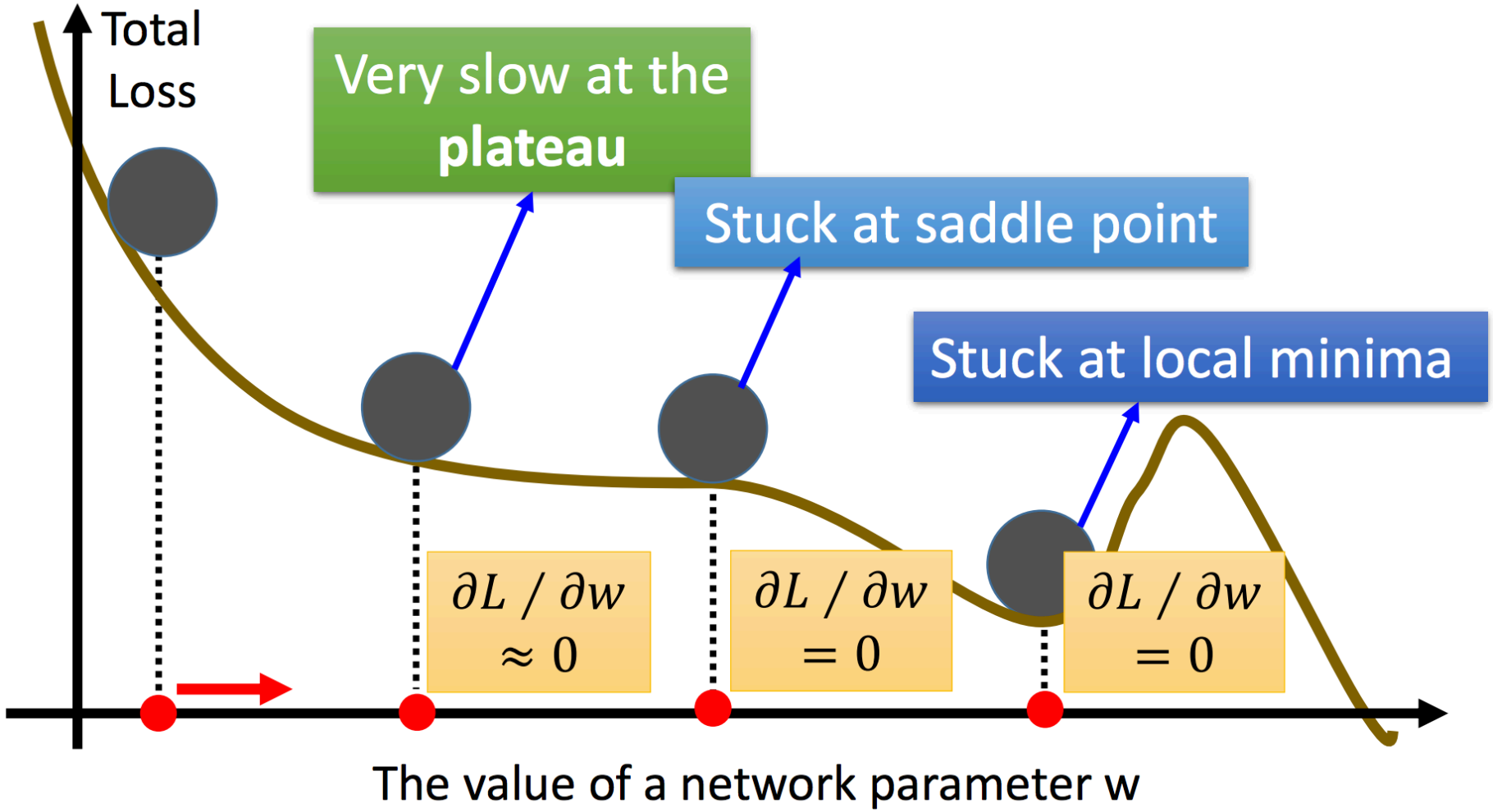


90% : training
10% : validation set

```
model.fit(X_all, Y_all, epochs = 10, validation_split = 0.1)
```

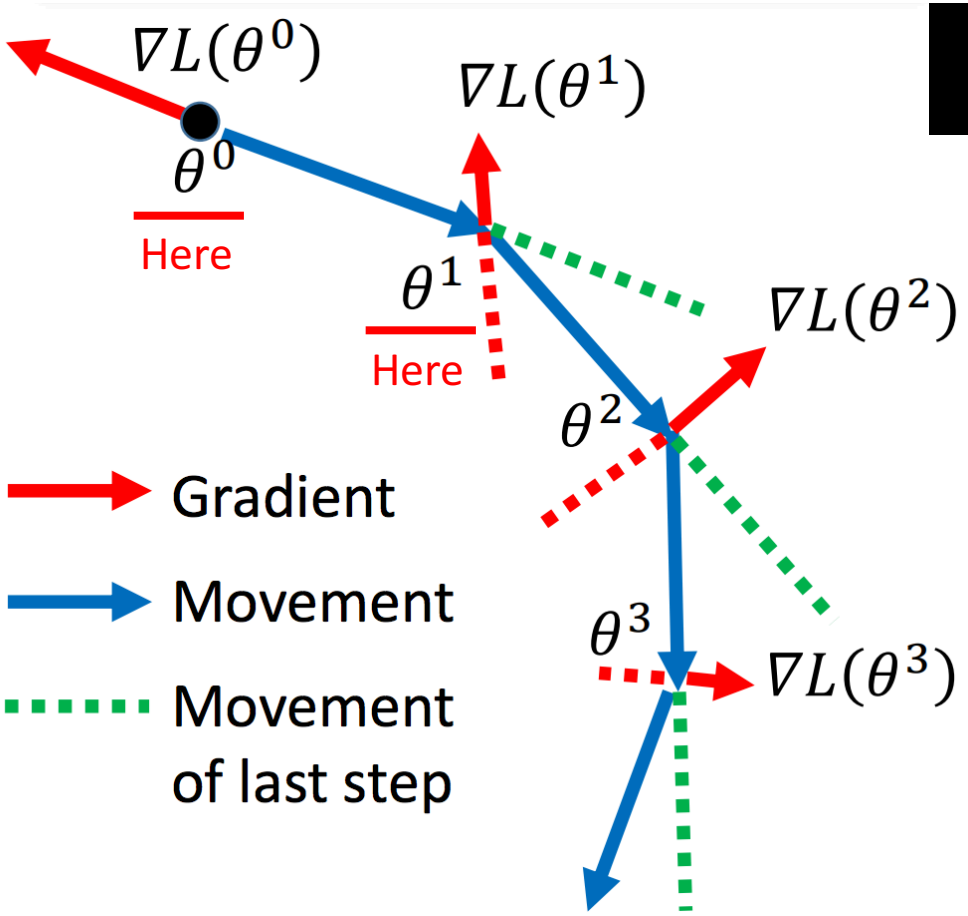
Ref: 宏毅教授投影片

Gradient Descent - momentum



Ref: 宏毅教授投影片

Gradient Descent - momentum




```
sgd = optimizers.SGD(lr=0.001, momentum=0.9)
```

λ

考慮前一次動量的程度

Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$

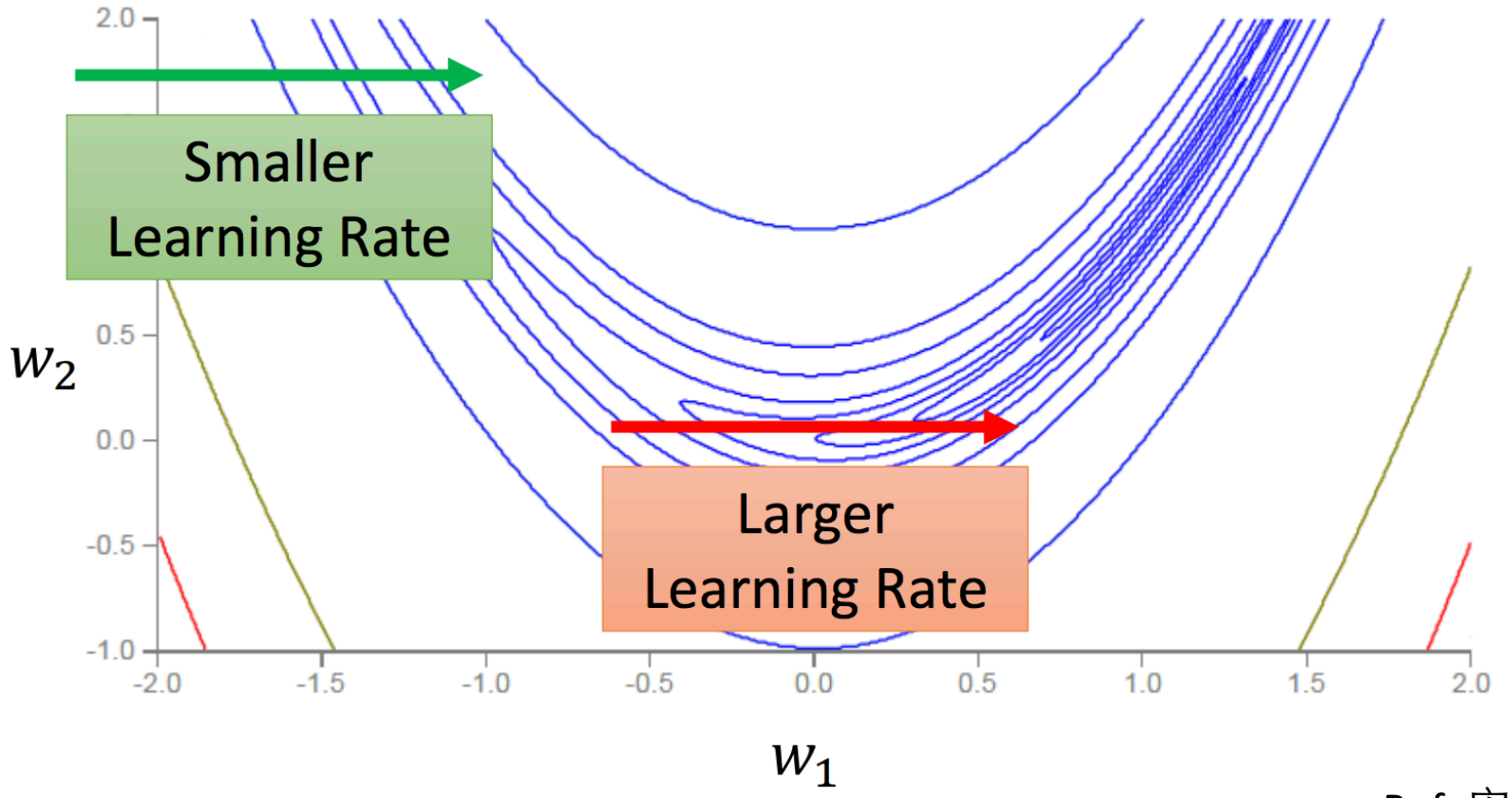
Move to $\theta^1 = \theta^0 + v^1$

Ref: 宏毅教授投影片

Gradient Descent – Adaptive learning rate

- RmsProp

Error Surface can be very complex when training NN.



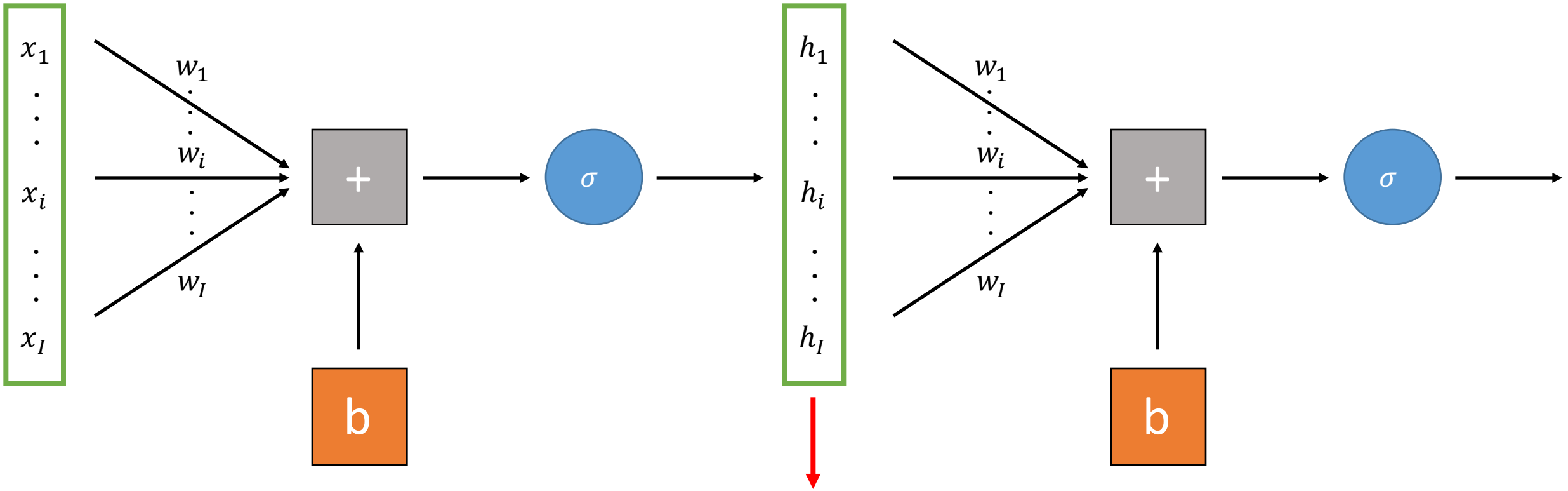
Ref: 宏毅教授投影片

Gradient Descent – Adaptive learning rate

- Adam: rmsprop + momentum

```
# TODO, 2
# Define optimizer
ada = optimizers.Adam(lr=0.001)

# TODO, 3
# Compile model
model.compile(loss = 'binary_crossentropy',
              optimizer = ada,
              metrics = ['accuracy'])
```

Deep Network

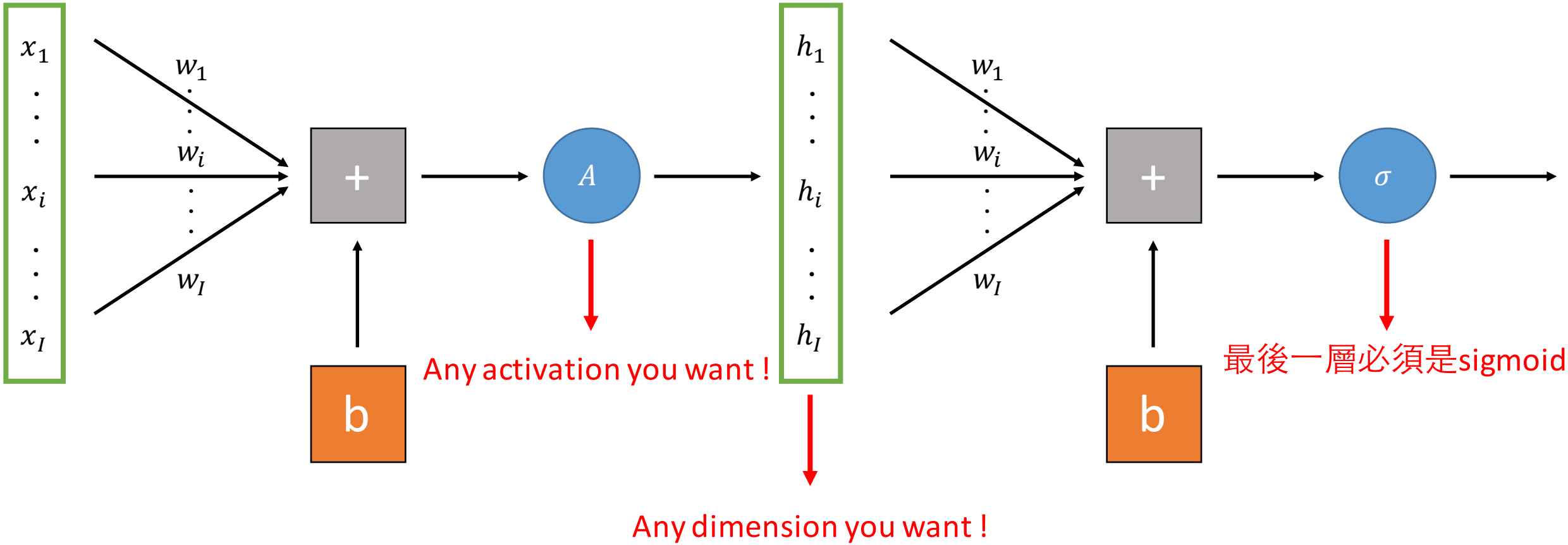


Deep Network

只需要在第一層宣告

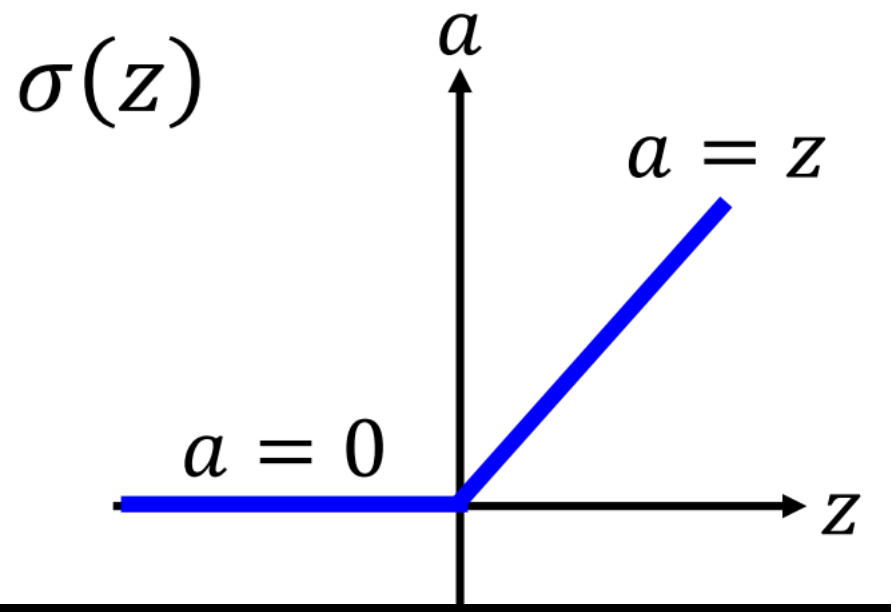

```
# TODO, 1
# Define model arch
model = Sequential()
feat_dims = X_train.shape[1]
model.add(Dense(input_dim = feat_dims, units = 1024, use_bias = True))
model.add(Activation('sigmoid'))
model.add(Dense(units = 512, use_bias = True))
model.add(Activation('sigmoid'))
model.add(Dense(units = 1, use_bias = True))
model.add(Activation('sigmoid'))
```

Deep Network



Other activation

- Rectified Linear Unit (ReLU)




```
# TODO, 1
# Define model arch
model = Sequential()
feat_dims = X_train.shape[1]
model.add(Dense(input_dim = feat_dims, units = 1024, use_bias = True))
model.add(Activation('relu'))
model.add(Dense(input_dim = feat_dims, units = 512, use_bias = True))
model.add(Activation('relu'))
model.add(Dense(input_dim = feat_dims, units = 1, use_bias = True))
model.add(Activation('sigmoid'))
```

Ref: 宏毅教授投影片

Implement Time

- Finish original logistic regression
- Adjust epochs, lr
- Try improvement tips

FAQ