

MTK DLP

Lab1 - Backpropagation

TA 鍾嘉峻

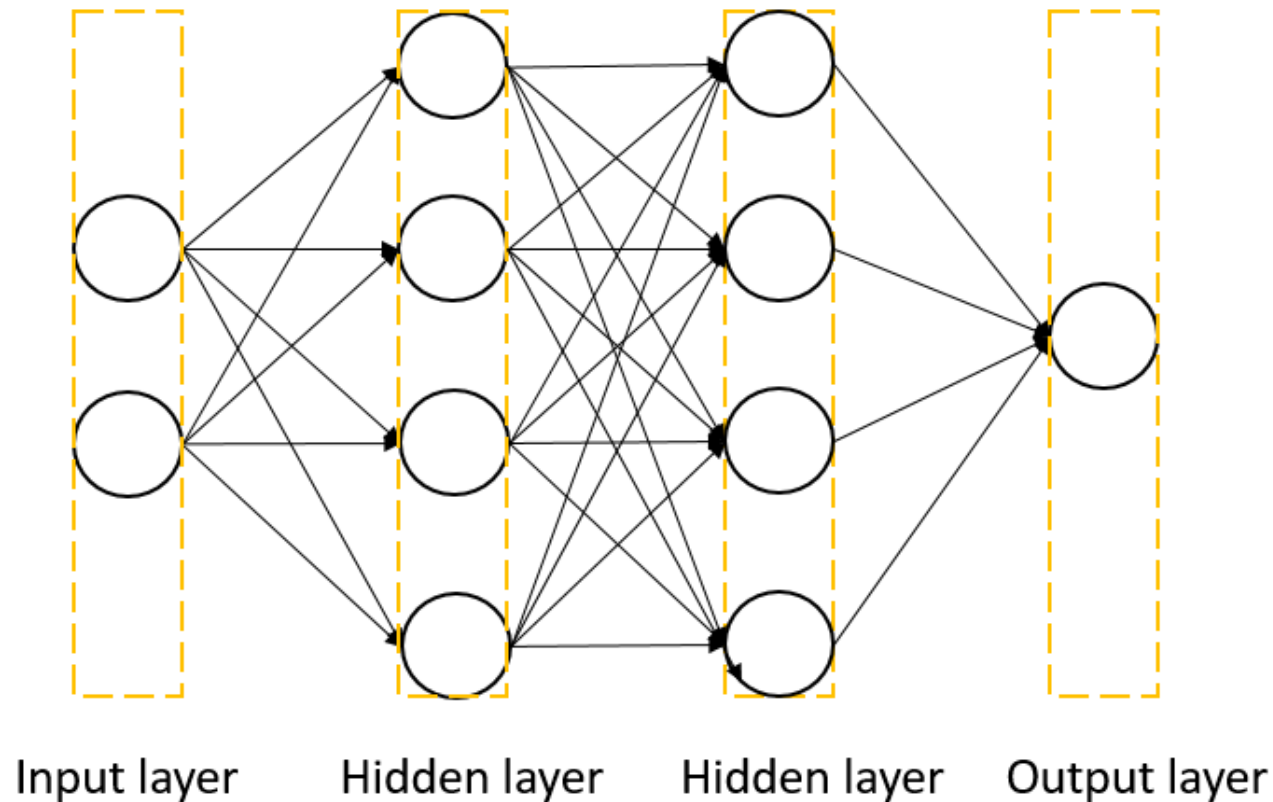
Aug 19, 2020

Outline

- Lab Objective
- Important Date
- Lab Description
- Scoring Criteria

Lab Objective

- In this lab, you will need to understand and implement a simple neural network with forward and backward pass using two hidden layers



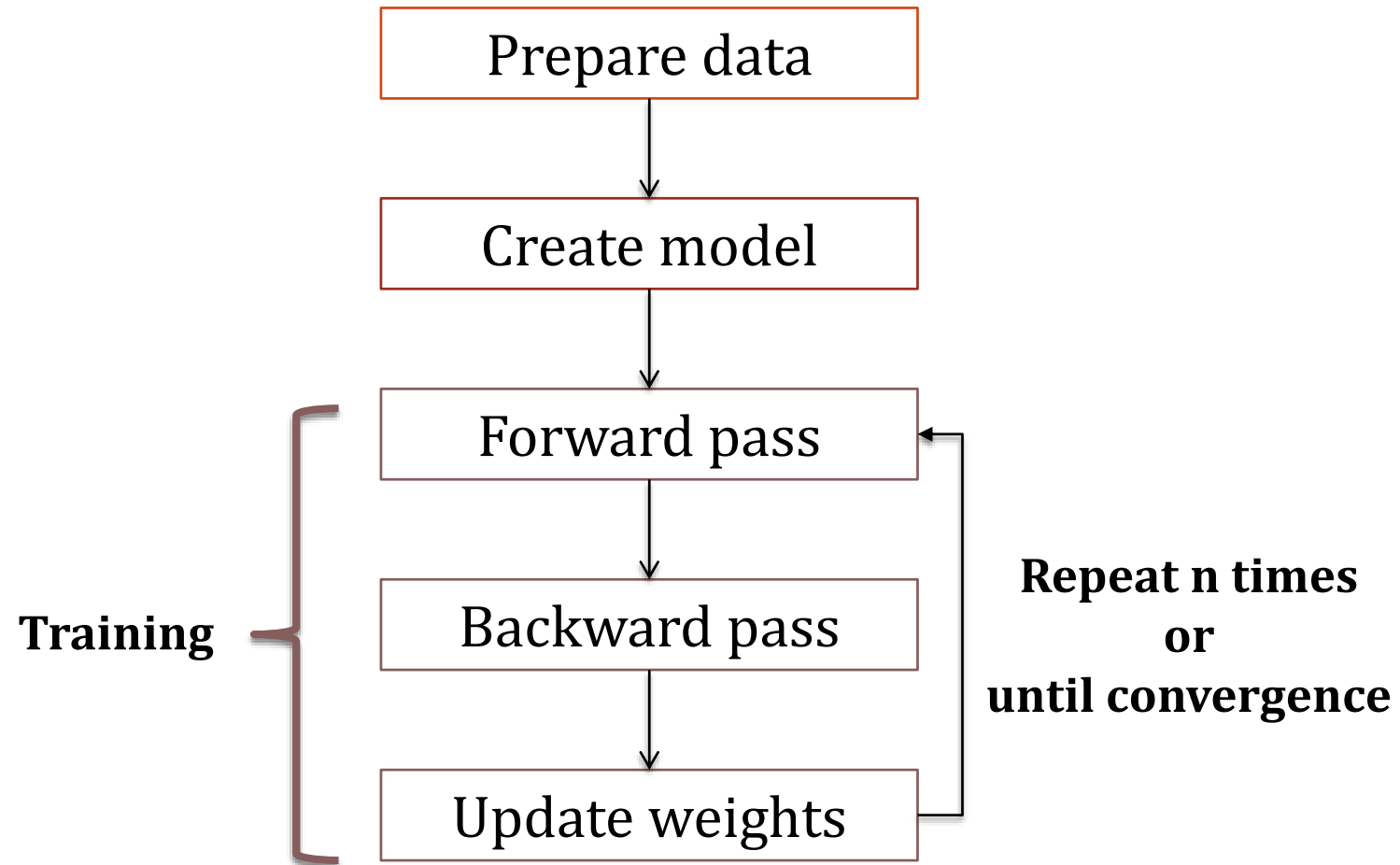
Important Date

- Report Submission Deadline: 9/2 (Wed) 11:59 a.m
- Demo date: 9/2 (Wed)
- Zip all files in one file
 - Report (.pdf)
 - Source code
- name it like 「DLP_LAB1_yourID_name.zip」
 - ex: 「DLP_LAB1_0756172_鍾嘉峻.zip」
- Email to
 - Zivzhong.cs07g@nctu.edu.tw

Lab Description

- Implement a simple neural network with two hidden layers
- You can only use **Numpy** and other **python standard libraries**.
- Plot your comparison figure showing the predictions and ground truth.
- Plot your learning curve (loss, epoch).
- Print the accuracy of your prediction.

Lab Description – Flowchart



Lab Implementation Steps

- Data prepare
- Create a model
- Train
- Test
- Plot result

Lab Implementation Steps

- Data prepare
- Create a model
- Train
- Test
 - We don't need to do this this time
- Plot result

Lab Implementation Steps

- Data prepare
- Create a model
- Train
- ~~Test~~
 - ~~We don't need to do this this time~~
- Plot result

Just show the loss of training set we create!!

Lab Implementation

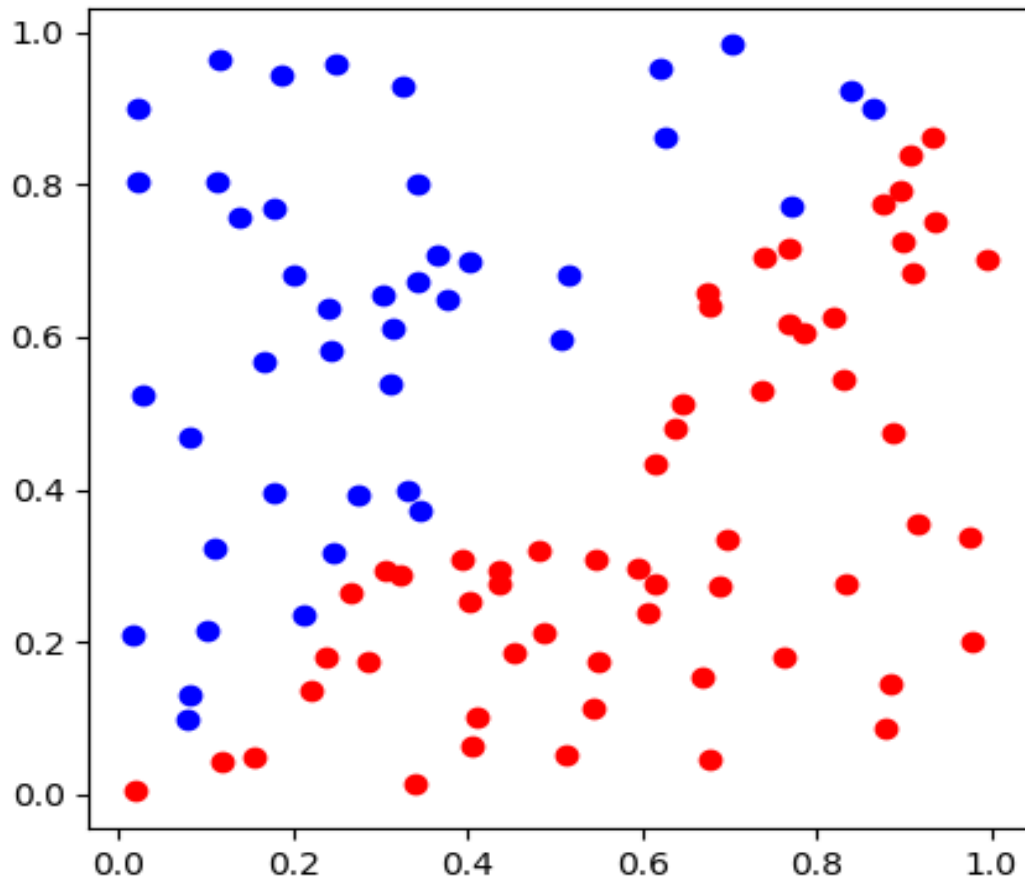
- Data prepare
- Create a model
- Train
- Plot result

```
1 import numpy as np
2
3
4 def generate_linear(n=100):
11
12 def generate_XOR_easy():
23
24 def show_result(x, y, y_pred):
40
41 def derivative(f, *args):
43
44 def sigmoid(x, derivative=False):
48
49 def mse(y_pred, y_data, derivative=False):
53
54 class NN:
98
99 nn = NN(dim=[2, 3, 3, 1])
100 lr, epoch, done = 0.8, 0, False
101
102 X, Y = generate_XOR_easy()
103 #X, Y = generate_linear()
104 while not done:
105     loss = []
106     for x, y in zip(X, Y):
107         x, y = x.reshape(1, -1), y.reshape(1, -1)
108         y_pred = nn.predict(x)
109         nn.backprop(y)
110         nn.update(lr)
111         loss += [mse(y_pred, y)]
112         epoch += 1
113         if epoch % 5000 == 0:
114             print('epoch', epoch, 'loss:', loss[-1])
115     done = all(np.array(loss) < 0.04)
116
117 Y_pred = [int(nn.predict(x.reshape(1, -1))) >= 0.5) for x in X]
118 show_result(X, Y, Y_pred)
```

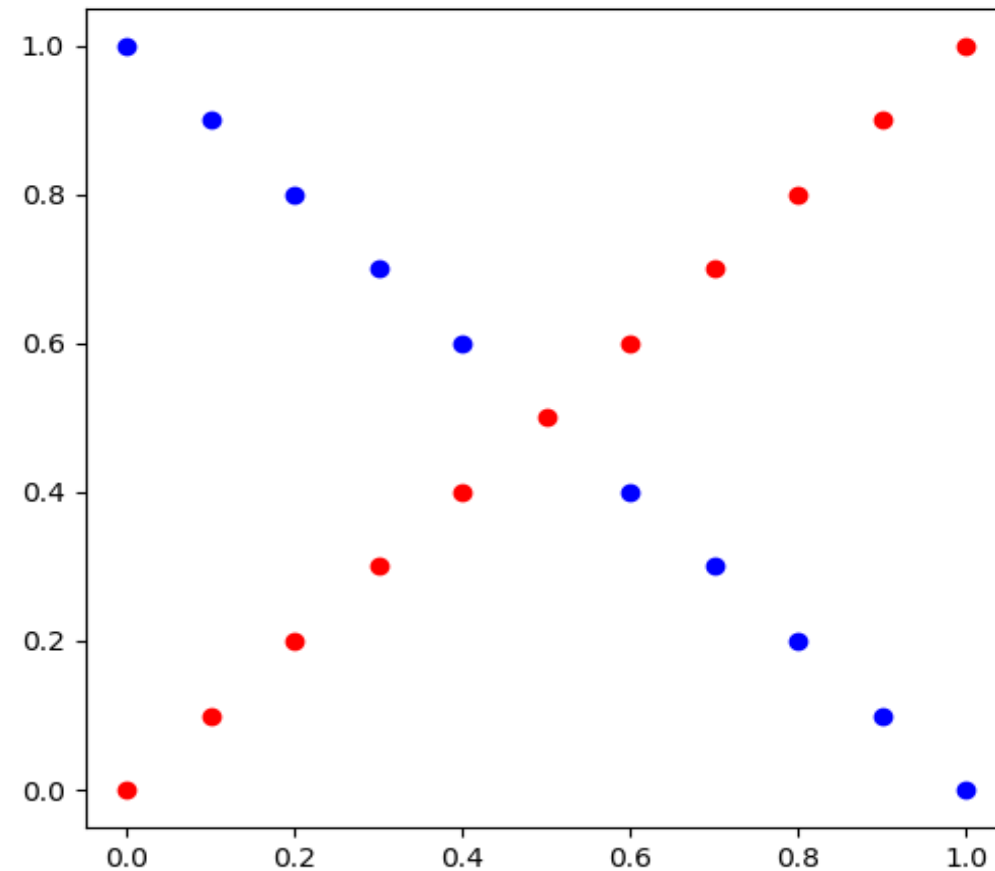
Lab Implementation Steps

- Data prepare
 - Use the functions TA prepare
- Create a model
- Train
- Plot result

Lab Description - Data



generate_linear()



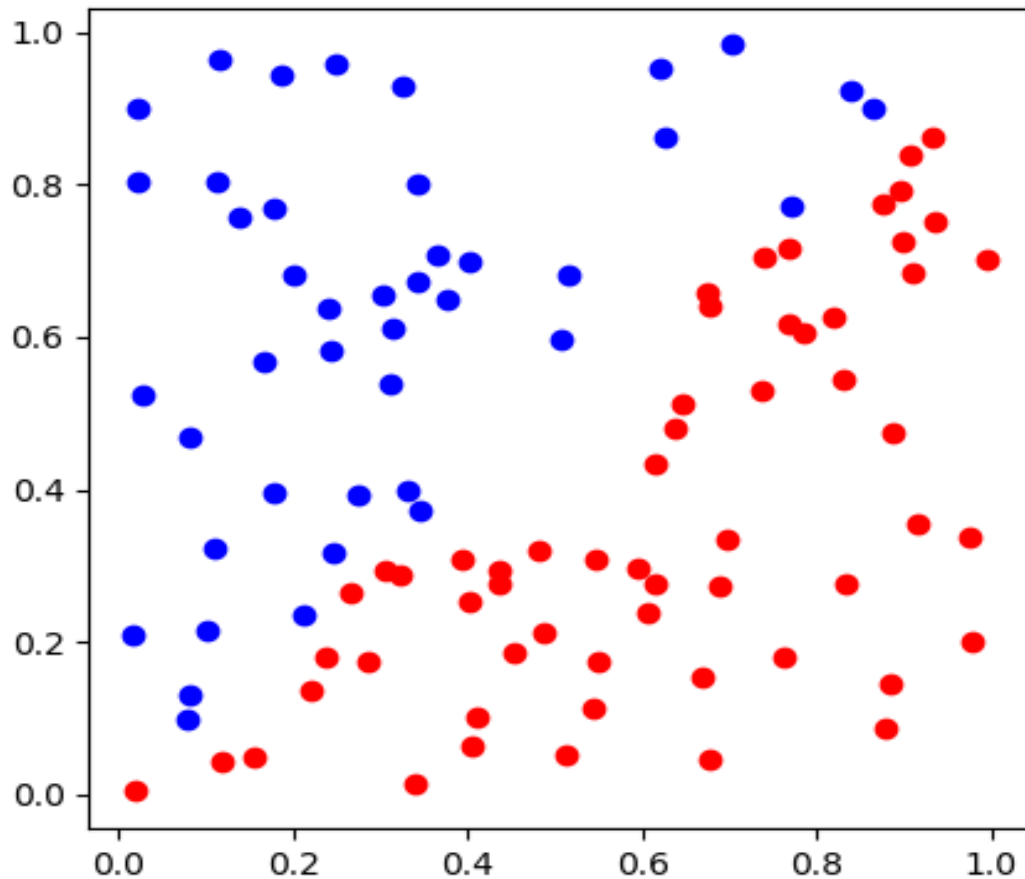
generate_XOR_easy()

Lab Description - Data

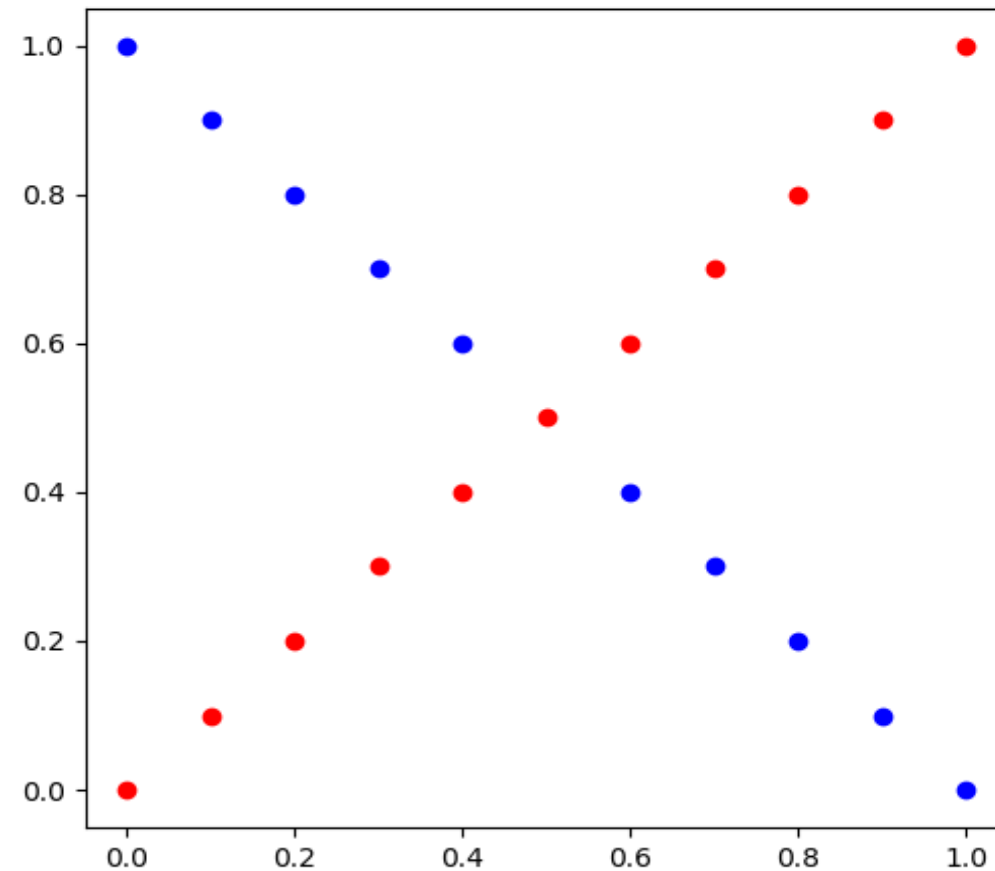
- generate_linear()

```
def generate_linear(n=100):  
    import numpy as np  
    pts = np.random.uniform(0, 1, (n, 2))  
    inputs = []  
    labels = []  
    for pt in pts:  
        inputs.append([pt[0], pt[1]])  
        distance = (pt[0]-pt[1])/1.414  
        if pt[0] > pt[1]:  
            labels.append(0)  
        else:  
            labels.append(1)  
    return np.array(inputs), np.array(labels).reshape(n, 1)
```

Lab Description - Data



generate_linear()



generate_XOR_easy()

Lab Description - Data

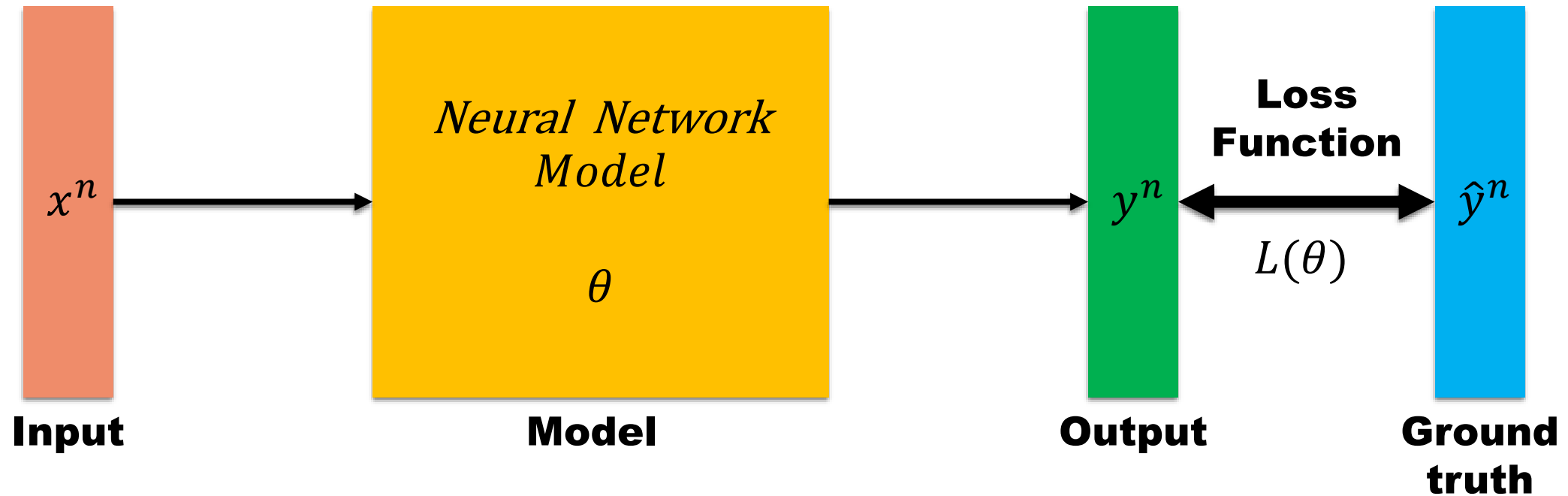
- generate_XOR_easy()

```
def generate_XOR_easy():  
    import numpy as np  
    inputs = []  
    labels = []  
  
    for i in range(11):  
        inputs.append([0.1*i, 0.1*i])  
        labels.append(0)  
  
        if 0.1*i == 0.5:  
            continue  
  
        inputs.append([0.1*i, 1-0.1*i])  
        labels.append(1)  
  
    return np.array(inputs), np.array(labels).reshape(21, 1)
```

Lab Implementation Steps

- Data prepare
- Create a model
 - You can only use numpy or other standard python library
- Train
- Plot result

Lab Description



$$\theta = \{w_1, w_2, w_3, w_4, \dots\}$$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta) / \partial w_1 \\ \partial L(\theta) / \partial w_2 \\ \partial L(\theta) / \partial w_3 \\ \vdots \end{bmatrix}$$

Compute $\nabla L(\theta^0)$

Compute $\nabla L(\theta^1)$

Compute $\nabla L(\theta^2)$

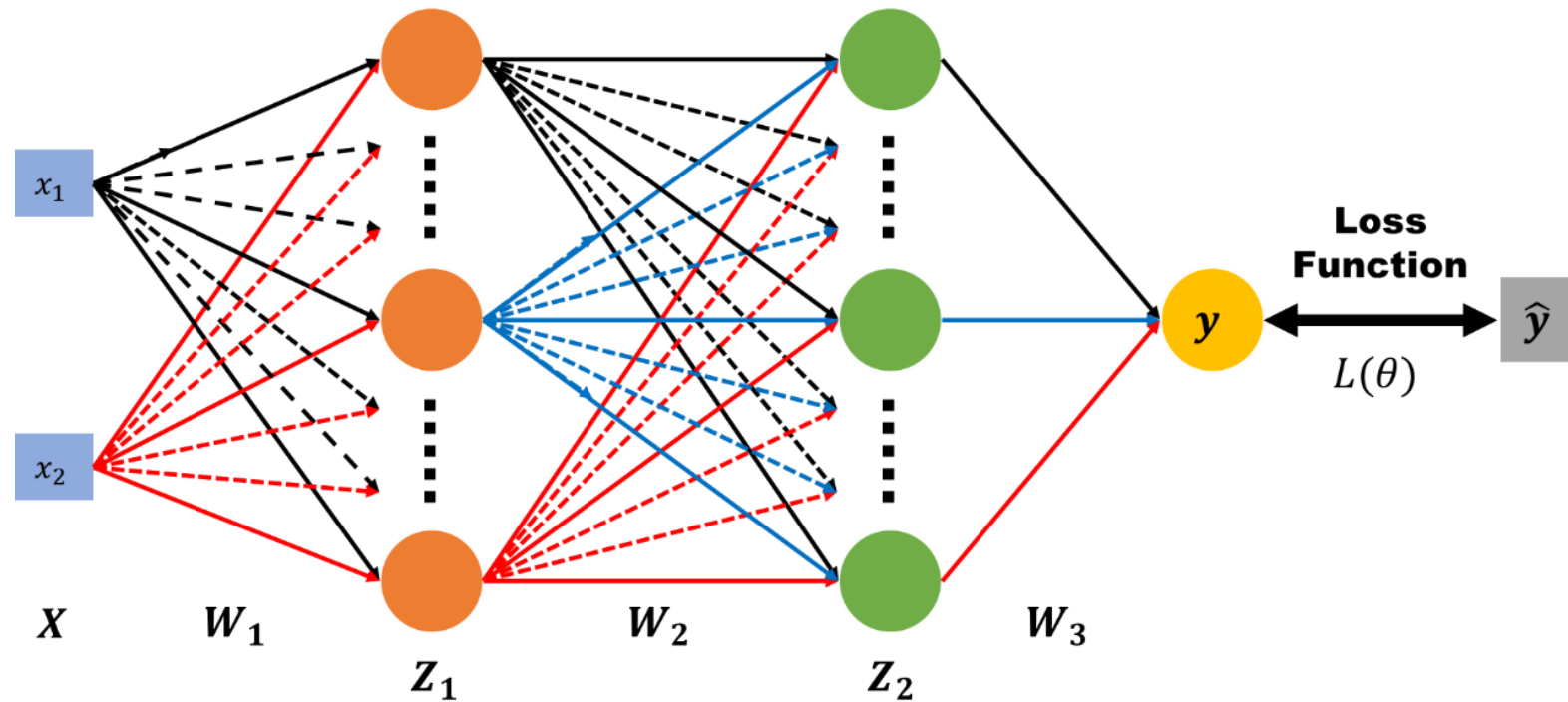
$$\theta^1 = \theta^0 - \rho \nabla L(\theta^0)$$

$$\theta^2 = \theta^1 - \rho \nabla L(\theta^1)$$

$$\theta^3 = \theta^2 - \rho \nabla L(\theta^2)$$

ρ : Learning rate

Lab Description – Architecture



$X : [x_1, x_2]$ $y : \text{outputs}$ $\hat{y} : \text{ground truth}$

$W_1, W_2, W_3 : \text{weight matrix of network layers}$

Lab Description – Architecture

- Define 4 layers
 - One input layer
 - Two hidden layers
 - One output layer
- Define 3 groups of weights
 - Between any two adjacent layers

Lab Description – Create a Model

```
54 class NN:
55     """Dense Layer"""
56
57     def init (self, dim, activation=sigmoid, loss=mse):
58
59
60
61
62
63
64
65
66
67
68
69     def predict(self, X):
70
71
72
73
74
75
76
77
78     def backprop(self, y data):
79
80
81
82
83
84
85
86
87
88
89
```

```
99 nn = NN(dim=[2, 3, 3, 1])
```

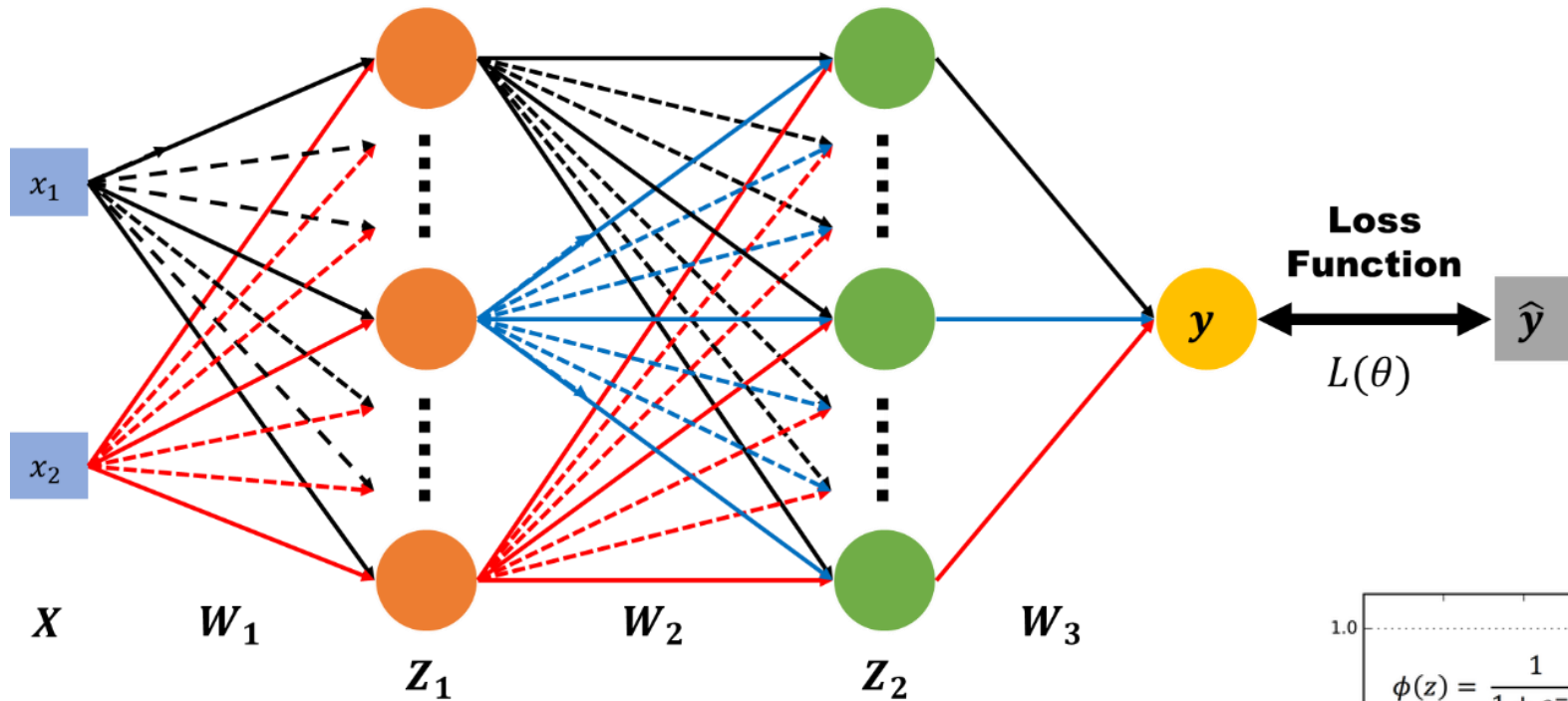
Lab Description – Create a Model

```
57 def __init__(self, dim, activation=sigmoid, loss=mse):
58     def init_weights(d):
63
64         self.layers = [None] * len(dim)
65         self.weights = [init_weights(d) for d in zip(dim[:-1], dim[1:])]
66         self.act = activation
67         self.loss = loss
68
99 nn = NN(dim=[2, 3, 3, 1])
```

Lab Implementation Steps

- Data prepare
- Create a model
- Train
 - Loss
 - Forward
 - Backward
- Plot result

Lab Description – Forward

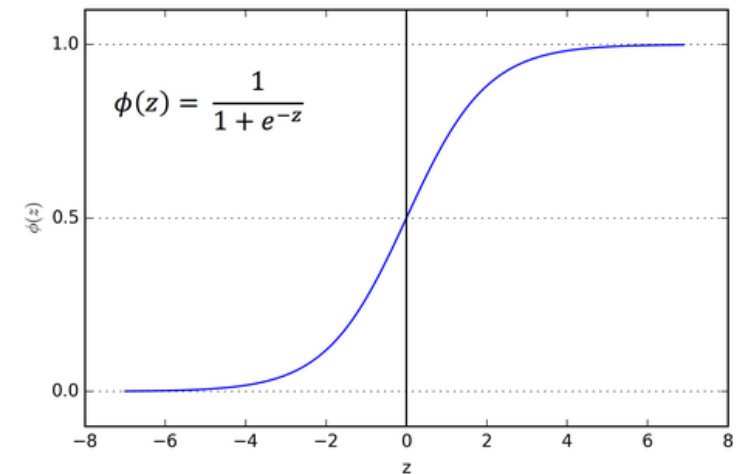


$$Z_1 = \sigma(XW_1)$$

$$Z_2 = \sigma(Z_1W_2)$$

$$y = \sigma(Z_2W_3)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Lab Description – Training Part

```
54 class NN:
55     """Dense Layer"""
56
57     def init (self, dim, activation=sigmoid, loss=mse):
58
59
60
61
62
63
64
65
66
67
68
69     def predict(self, X):
70
71
72
73
74
75
76
77
78     def backprop(self, y data):
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93     def update(self, lr):
94
95
96
97
98
99
```


Lab Description – Loss

- MSE Loss

```
48  
49 def mse(y_pred, y_data, derivative=False):  
53
```

Lab Description – Forward(Predict)

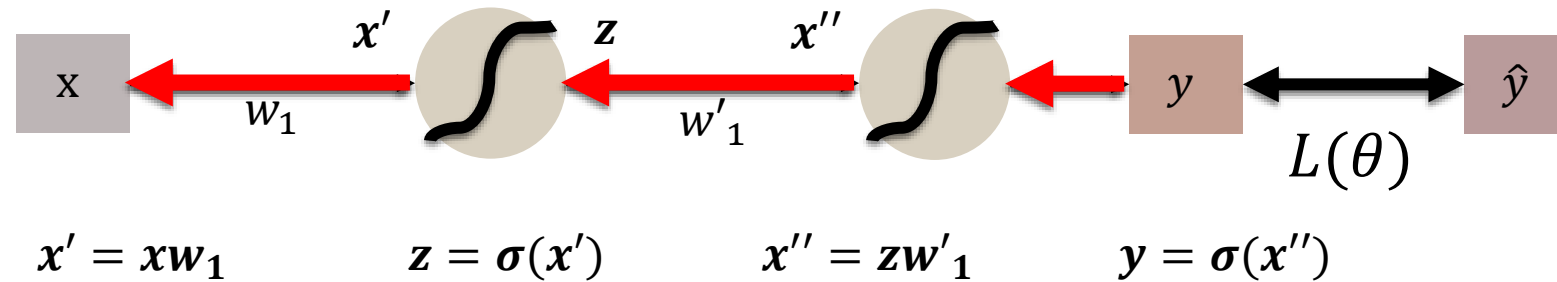
- Forward(Predict)
 - $Z = X$
 - for i, W in enumerate(self.weights):
 - $Z = \text{self.act}(Z @ W)$
 - Remember move to next layer
 - (e.g `self.layers[next_layer] = np.array(Z)`)

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

$$\sigma'(y) = \sigma(y)(1 - \sigma(y))$$

- Ref Link:
 - <https://medium.com/pyradise/%E4%BD%BF%E7%94%A8-python-%E4%BE%86%E8%AA%8D%E8%AD%98%E7%9F%A9%E9%99%A3-915376207187>

Lab Description – Backward



Chain rule

$$y = g(x) \quad z = h(y)$$

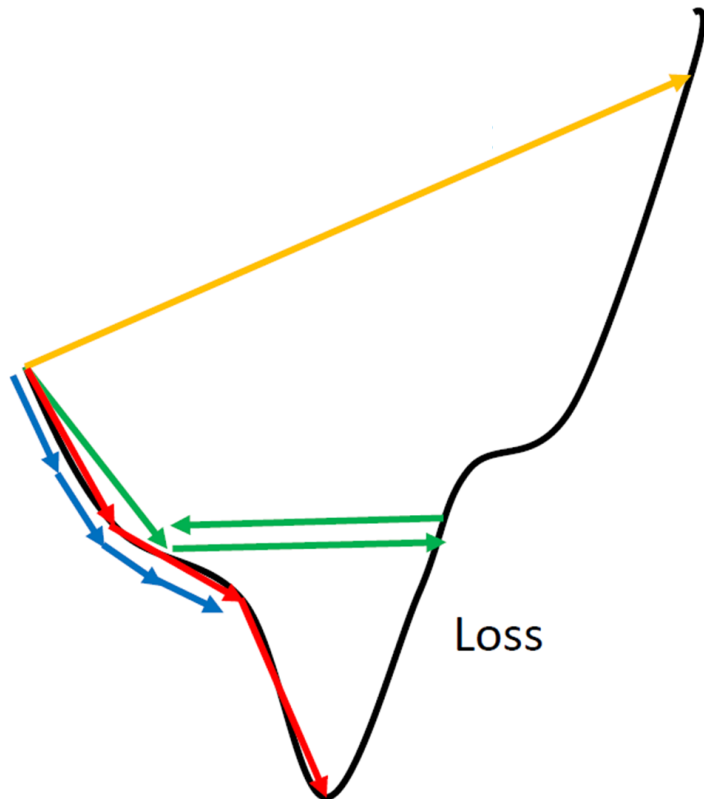
$$\mathbf{x} \xrightarrow{g()} \mathbf{y} \xrightarrow{h()} \mathbf{z}$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

$$\begin{aligned} \frac{\partial L(\theta)}{\partial w_1} &= \frac{\partial y}{\partial w_1} \frac{\partial L(\theta)}{\partial y} \\ &= \frac{\partial x''}{\partial w_1} \frac{\partial y}{\partial x''} \frac{\partial L(\theta)}{\partial y} \\ &= \frac{\partial z}{\partial w_1} \frac{\partial x''}{\partial z} \frac{\partial y}{\partial x''} \frac{\partial L(\theta)}{\partial y} \\ &= \frac{\partial x'}{\partial w_1} \frac{\partial z}{\partial x'} \frac{\partial z}{\partial x''} \frac{\partial y}{\partial x''} \frac{\partial L(\theta)}{\partial y} \end{aligned}$$

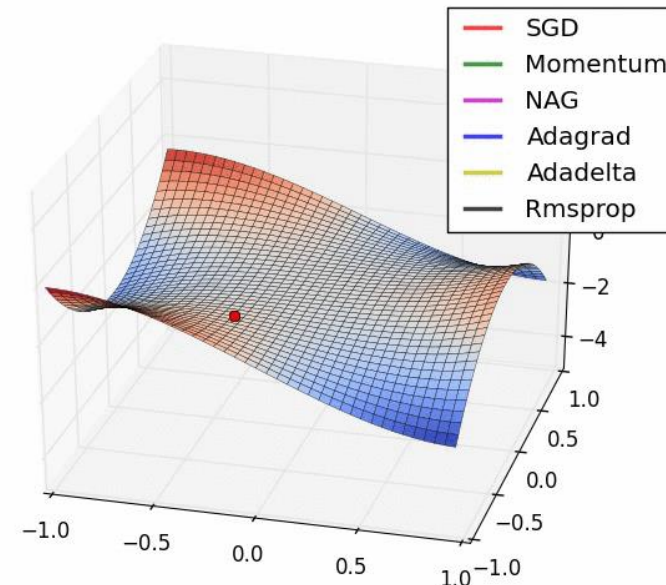
Lab Description – Gradient descent

Network Parameters $\theta = \{w_1, w_2, w_3, w_4, \dots\}$



$$\begin{aligned}\theta^1 &= \theta^0 - \rho \nabla L(\theta^0) \\ \theta^2 &= \theta^1 - \rho \nabla L(\theta^1) \\ \theta^3 &= \theta^2 - \rho \nabla L(\theta^2)\end{aligned}$$

ρ : Learning rate



Lab Description – Backward & Update

- Backward the gradient & update

```
54 class NN:
55     """Dense Layer"""
56
57     def init (self, dim, activation=sigmoid, loss=mse):
58
59
60
61
62
63
64
65
66
67
68
69     def predict(self, X):
70
71
72
73
74
75
76
77
78     def backprop(self, y data):
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93     def update(self, lr):
94
95
96
97
98
99
100
```

Lab Description – Backward & Update

- Backward the gradient & update

- Gradient

```
# dw = dL / dW
dw = np.kron(Z.T, dz)
dW.append(np.array(dw))
# dz = dL / dZ * s'(Z)
dz = np.multiply(dz @ W.T, derivative(self.act, Z))
dZ.append(np.array(dz))
```

- Update

```
self.weights[i] -= lr * dw
```

Lab Implementation Steps

- Data prepare
- Create a model
- Train
- Plot result
 - Screenshot of loss
 - Plot the prediction & ground truth

Lab Description - Prediction

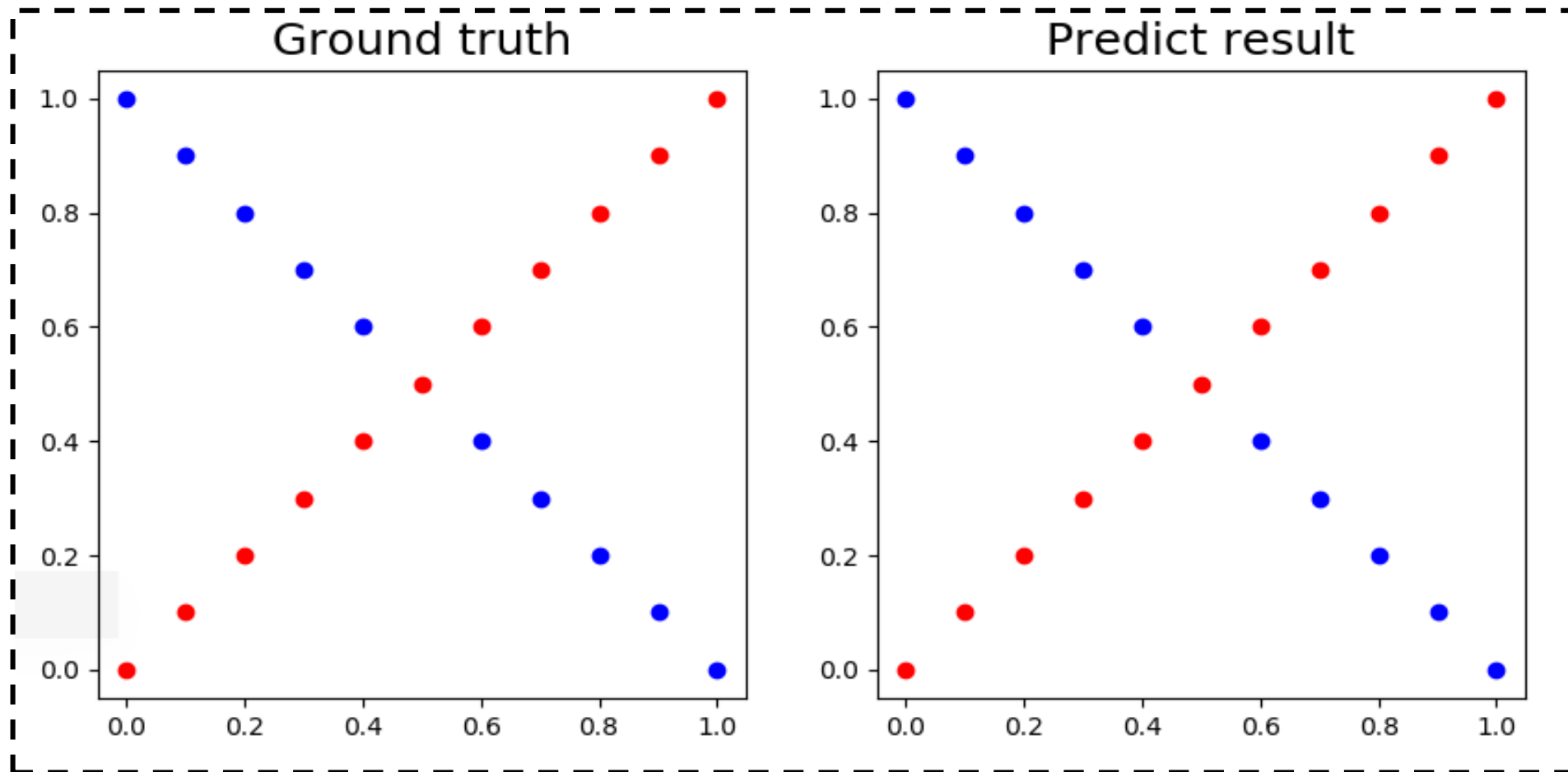
- In the training, you need to print loss
- In the testing, you need to show your predictions, also the accuracy

```
epoch 10000 loss : 0.16234523253277644
epoch 15000 loss : 0.2524336634177614
epoch 20000 loss : 0.1590783047540092
epoch 25000 loss : 0.22099447030234853
epoch 30000 loss : 0.3292173477217561
epoch 35000 loss : 0.40406233282426085
epoch 40000 loss : 0.43052897480298924
epoch 45000 loss : 0.4207525735586605
epoch 50000 loss : 0.3934759509342479
epoch 55000 loss : 0.3615008372106921
epoch 60000 loss : 0.33077879872648525
epoch 65000 loss : 0.30333537090819584
epoch 70000 loss : 0.2794858089741792
epoch 75000 loss : 0.25892812312991587
epoch 80000 loss : 0.24119780823897027
epoch 85000 loss : 0.22583656353511342
epoch 90000 loss : 0.21244497028971704
epoch 95000 loss : 0.2006912468389013
```

```
[[0.01025062]
 [0.99730607]
 [0.02141321]
 [0.99722154]
 [0.03578171]
 [0.99701922]
 [0.04397049]
 [0.99574117]
 [0.04162245]
 [0.92902792]
 [0.03348791]
 [0.02511045]
 [0.94093942]
 [0.01870069]
 [0.99622948]
 [0.01431959]
 [0.99434455]
 [0.01143039]
 [0.98992477]
 [0.00952752]
 [0.98385905]]
```


Lab Description - Prediction

- Visualize the predictions and ground truth at the end of the training process



Lab Description - Prediction

- Visualize the predictions and ground truth at the end of the training process

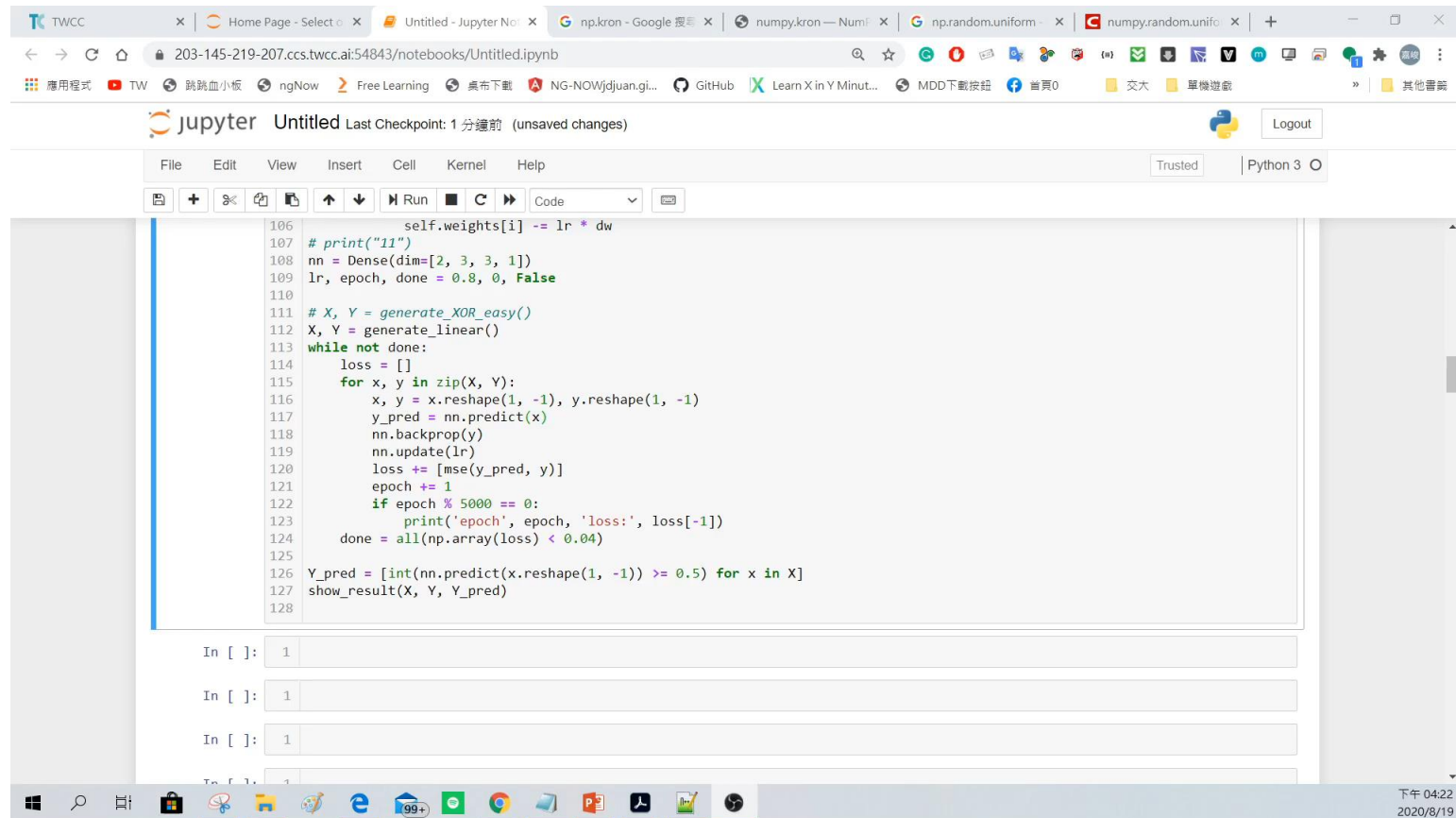
```
def show_result(x, y, pred_y):
    import matplotlib.pyplot as plt
    plt.subplot(1,2,1)
    plt.title('Ground truth', fontsize=18)
    for i in range(x.shape[0]):
        if y[i] == 0:
            plt.plot(x[i][0], x[i][1], 'ro')
        else:
            plt.plot(x[i][0], x[i][1], 'bo')

    plt.subplot(1,2,2)
    plt.title('Predict result', fontsize=18)
    for i in range(x.shape[0]):
        if pred_y[i] == 0:
            plt.plot(x[i][0], x[i][1], 'ro')
        else:
            plt.plot(x[i][0], x[i][1], 'bo')

    plt.show()
```

Demo

- Video Link: <https://youtu.be/OdwA9SaMh4E>



The screenshot displays a Jupyter Notebook environment with a browser window at the top showing the URL `203-145-219-207.ccs.twcc.ai:54843/notebooks/Untitled.ipynb`. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for file operations and execution. The code editor shows a Python script for training a neural network. The script defines a `Dense` layer, generates XOR and linear data, and implements a training loop with backpropagation and MSE loss. The output area at the bottom shows three input prompts, each followed by the number `1`.

```
106         self.weights[i] -= lr * dw
107     # print("11")
108     nn = Dense(dim=[2, 3, 3, 1])
109     lr, epoch, done = 0.8, 0, False
110
111     # X, Y = generate_XOR_easy()
112     X, Y = generate_linear()
113     while not done:
114         loss = []
115         for x, y in zip(X, Y):
116             x, y = x.reshape(1, -1), y.reshape(1, -1)
117             y_pred = nn.predict(x)
118             nn.backprop(y)
119             nn.update(lr)
120             loss += [mse(y_pred, y)]
121             epoch += 1
122             if epoch % 5000 == 0:
123                 print('epoch', epoch, 'loss:', loss[-1])
124             done = all(np.array(loss) < 0.04)
125
126     Y_pred = [int(nn.predict(x.reshape(1, -1)) >= 0.5) for x in X]
127     show_result(X, Y, Y_pred)
128
```

In []: 1

In []: 1

In []: 1

In []: 1

下午 04:22
2020/8/19

Browser tabs: TWCC, Home Page - Select, Untitled - Jupyter Notebook, np.kron - Google 搜尋, numpy.kron — NumPy, np.random.uniform, numpy.random.unif...

Address bar: 203-145-219-207.ccs.twcc.ai:54843/notebooks/Untitled.ipynb

Navigation bar: 應用程式, TW, 跳跳血小板, ngNow, Free Learning, 桌布下載, NG-NOWjdjuan.gi..., GitHub, Learn X in Y Minut..., MDD下載按鈕, 首頁0, 交大, 單機遊戲, 其他書籤

Jupyter interface: jupyter Untitled Last Checkpoint: 1 分鐘前 (unsaved changes) Python 3

Menu: File Edit View Insert Cell Kernel Help

Toolbar: Save, New, Copy, Paste, Undo, Redo, Run, Stop, Refresh, Code

```
106         self.weights[i] -= lr * dw
107     # print("11")
108     nn = Dense(dim=[2, 3, 3, 1])
109     lr, epoch, done = 0.8, 0, False
110
111     # X, Y = generate_XOR_easy()
112     X, Y = generate_linear()
113     while not done:
114         loss = []
115         for x, y in zip(X, Y):
116             x, y = x.reshape(1, -1), y.reshape(1, -1)
117             y_pred = nn.predict(x)
118             nn.backprop(y)
119             nn.update(lr)
120             loss += [mse(y_pred, y)]
121             epoch += 1
122             if epoch % 5000 == 0:
123                 print('epoch', epoch, 'loss:', loss[-1])
124         done = all(np.array(loss) < 0.04)
125
126     Y_pred = [int(nn.predict(x.reshape(1, -1)) >= 0.5) for x in X]
127     show_result(X, Y, Y_pred)
128
```

Input prompts: In []: 1

Scoring Criteria

- Report (40%)
- Demo(60%)
 - Experimental results (40%)
 - Questions (20%)
- Late report or demo
 - Score $*0.8$

Reference

1. <http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html>
2. http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML17_2.html

Backup Slide

- In python, you may observe some weird things
 - $0.1+0.2 \neq 0.3$ (because IEEE 754
 - $0.2+0.1 \neq 0.3$
 - $0.1+0.2 == 0.2+0.1$

Backup Slide

- In python, you may observe some weird things
 - $0.1+0.2 \neq 0.3$ (because IEEE 754
 - $0.2+0.1 \neq 0.3$
 - $0.1+0.2 == 0.2+0.1$
 - $0.1+0.2+0.1+0.2 \neq 0.6$
 - $0.2+0.1+0.2+0.1 == 0.6$
 - $0.1+0.2+0.1+0.2 \neq 0.2+0.1+0.2+0.1$


```
(base) pc3433@pc3433-B360-HD3:~$ python
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.2 + 0.1
0.30000000000000004
>>> 0.1 + 0.2 == 0.2 + 0.1
True
>>> 0.1 + 0.2 + 0.1 + 0.2
0.6000000000000001
>>> 0.2 + 0.1 + 0.2 + 0.1
0.6
>>> a = 0.1 + 0.2 + 0.1 + 0.2
>>> b = 0.2 + 0.1 + 0.2 + 0.1
>>> a == b
False
>>> █
```

Backup Slide

- In python, you may observe some weird things
 - $0.1+0.2 \neq 0.3$ (because IEEE 754
 - $0.2+0.1 \neq 0.3$
 - $0.1+0.2 == 0.2+0.1$
 - $0.1+0.2+0.1+0.2 \neq 0.6$
 - $0.2+0.1+0.2+0.1 == 0.6$
 - $0.1+0.2+0.1+0.2 \neq 0.2+0.1+0.2+0.1$
- Check WTFPython
 - <https://github.com/satwikkansal/wtfpython>