MẠNG NEURON VÀ ỨNG DỤNG TRONG XỬ LÝ TÍN HIỆU

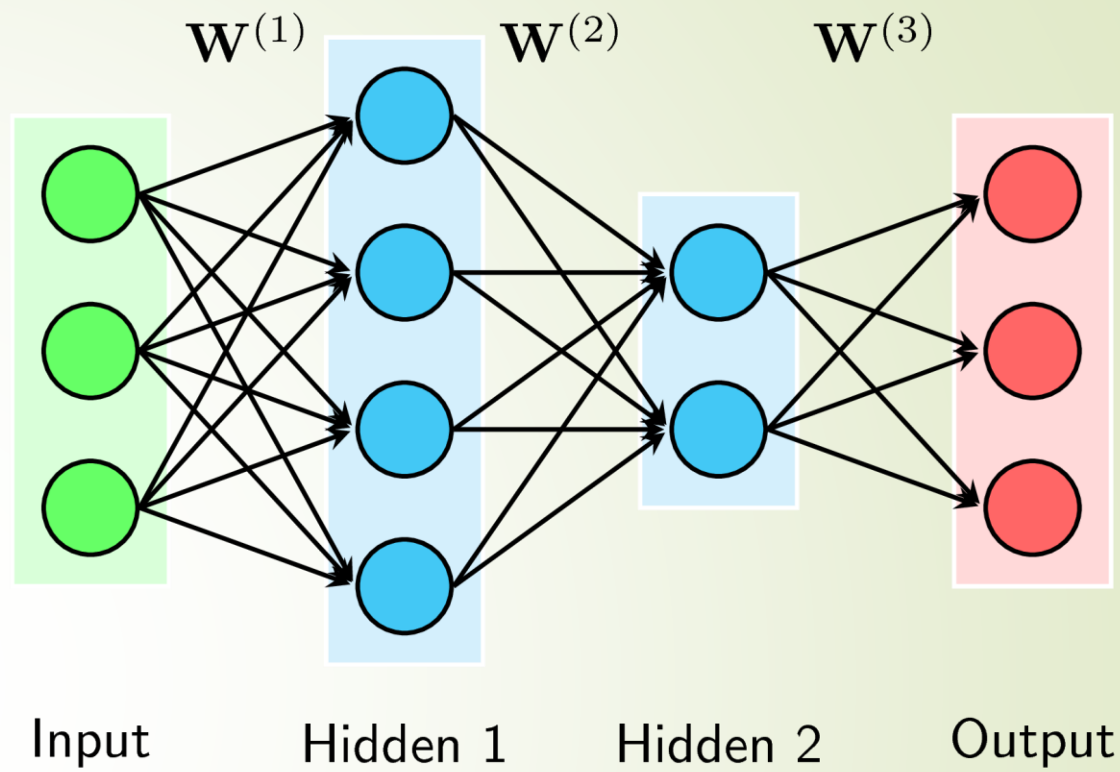
TS. TRẦN MẠNH CƯỜNG
TS. NGUYỄN THÚY BÌNH
BỘ MÔN KỸ THUẬT ĐIỆN TỬ
Email: thuybinh_ktdt@utc.edu.vn

Multi-layer Perceptron và Backpropagation

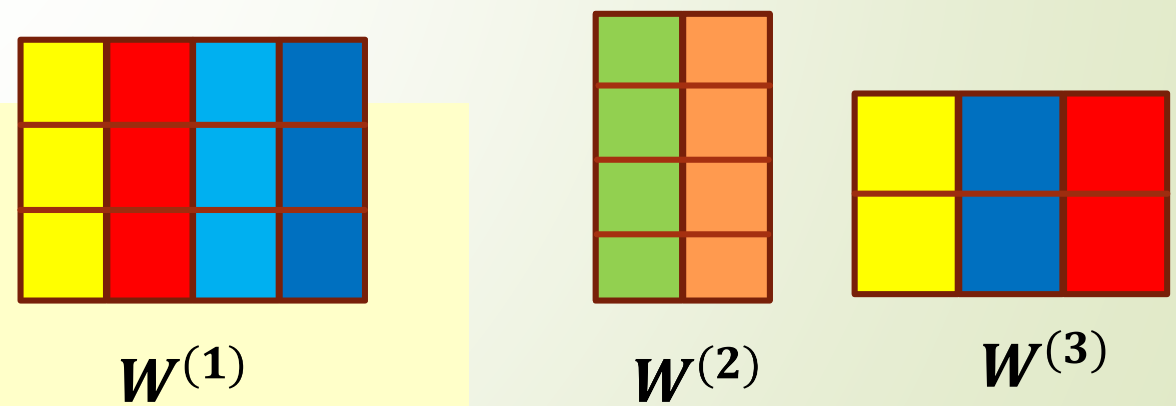
1. Giới thiệu
2. Hàm mất mát và phương pháp tối ưu
3. Ví dụ

1. Giới thiệu

- ❑ Đầu vào: Input layer
- ❑ Đầu ra: Output layer
- ❑ Lớp ẩn: Hidden layers
- ❑ Số lượng các lớp trong MLP = số lớp ẩn (hiddent layers) +1

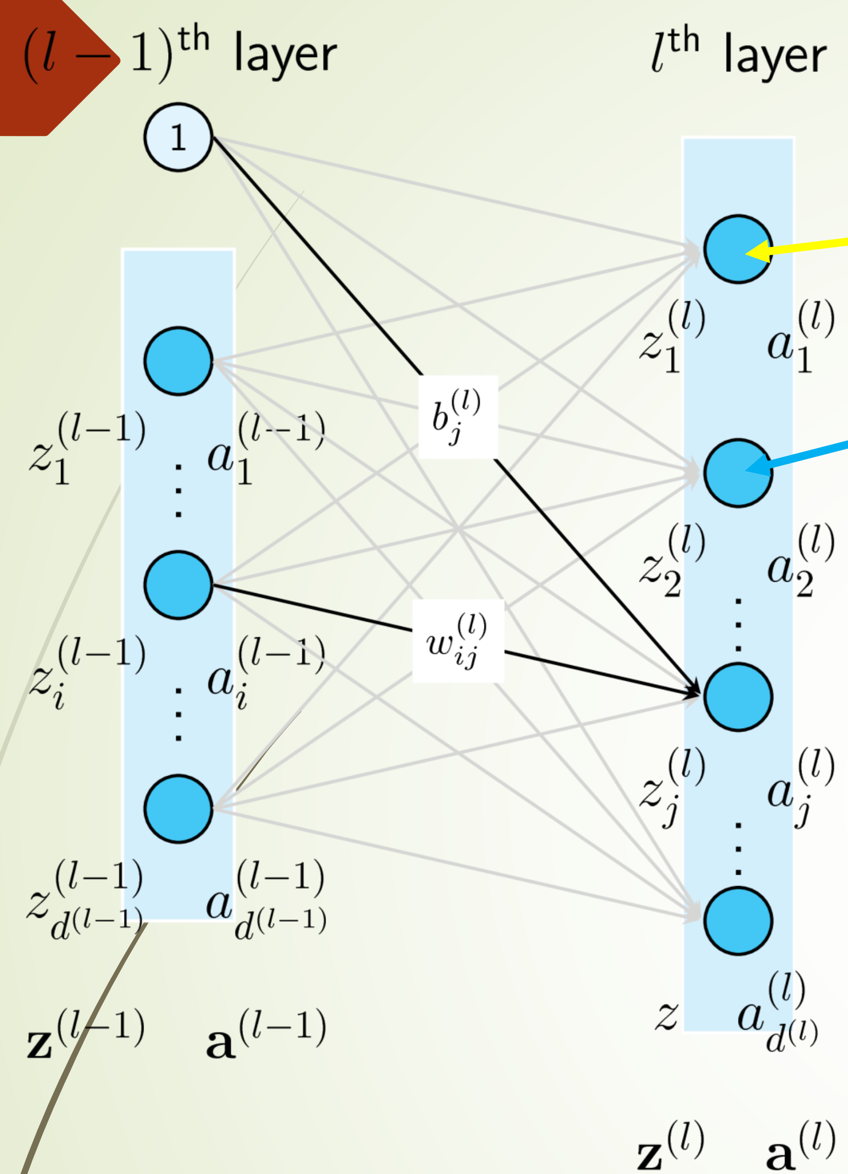


- ❑ Mỗi node trong một layer: unit
- ❑ Đầu vào của mỗi hidden layer: ***z***
- ❑ Đầu ra của mỗi hidden layer: **a**
- ❑ Đầu ra tại unit thứ *i*, lớp thứ *l*: $a_i^{(l)}$
- ❑ Số lượng unit của lớp thứ *l*: $d^{(l)} \rightarrow \mathbf{a}^{(l)} \in R^{d^{(l)}}$



$$\begin{bmatrix} 2 & 5 & 7 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 9 \end{bmatrix} = 65$$

1. Giới thiệu



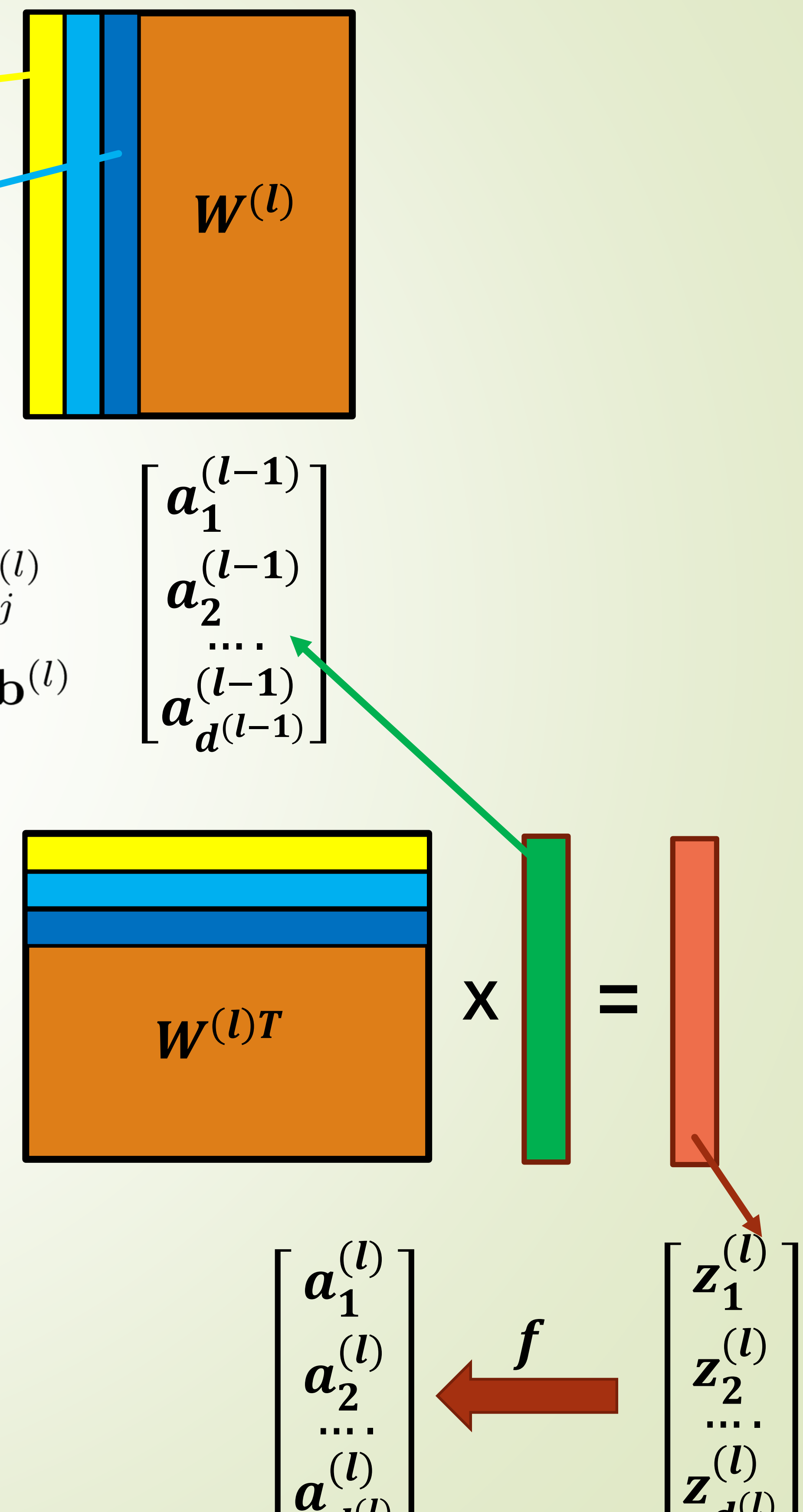
$$\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$$

$$\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}$$

$$z_j^{(l)} = \mathbf{w}_j^{(l)T}\mathbf{a}^{(l-1)} + b_j^{(l)}$$

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

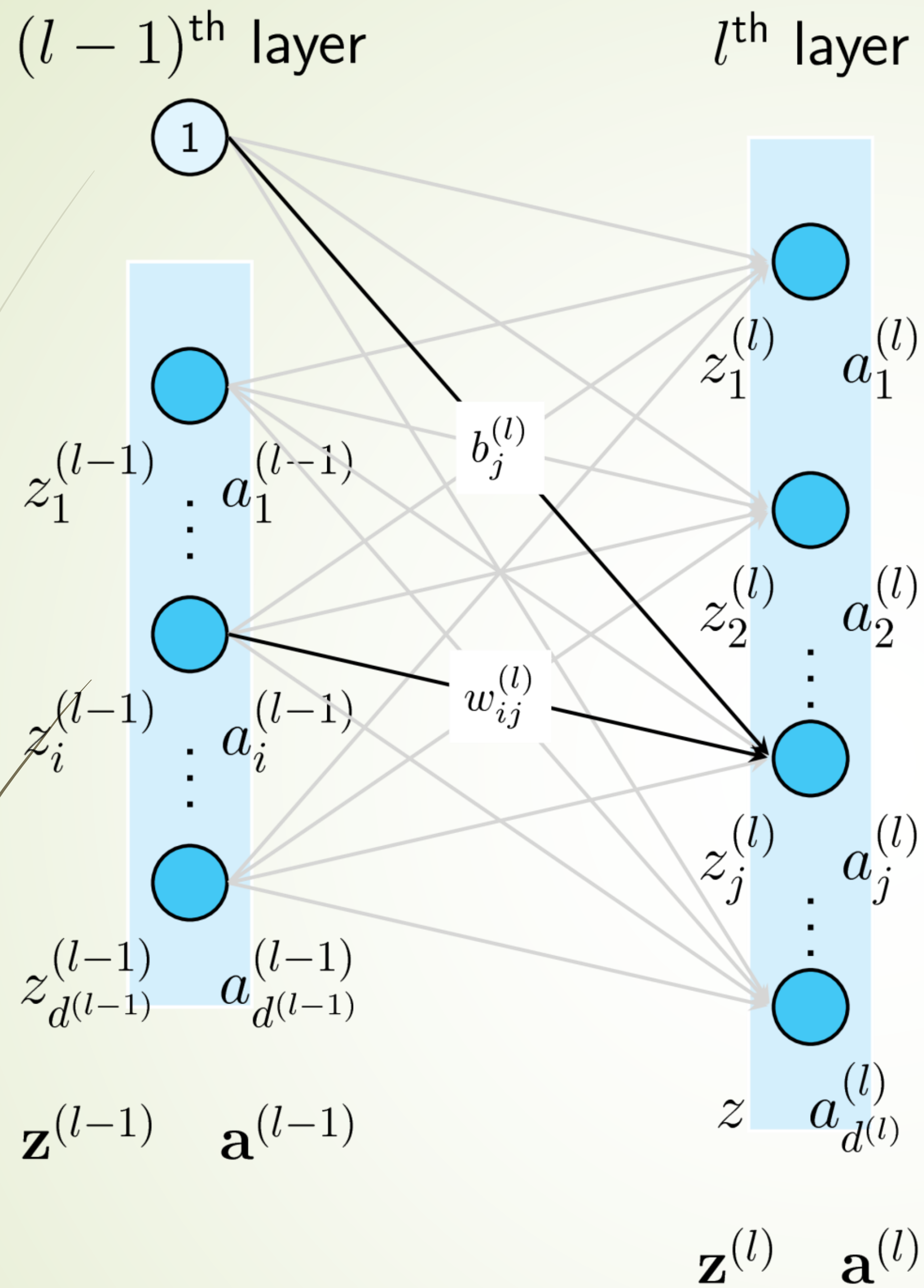


- ❑ MLP có L layers: → L ma trận trọng số

  $$\mathbf{W}^{(l)} \in R^{d^{(l-1)} \times d^{(l)}}; \quad l = 1, 2, 3, \dots, L$$

- ❑ $w_{ij}^{(l)}$: kết nối từ lớp thứ $(l-1)$ đến lớp thứ (l)
- ❑ Bias của layer thứ (l): $b^{(l)} \in R^{d^{(l)}}$

1. Giới thiệu



$$\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$$

$$\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}$$

$$z_j^{(l)} = \mathbf{w}_j^{(l)T} \mathbf{a}^{(l-1)} + b_j^{(l)}$$

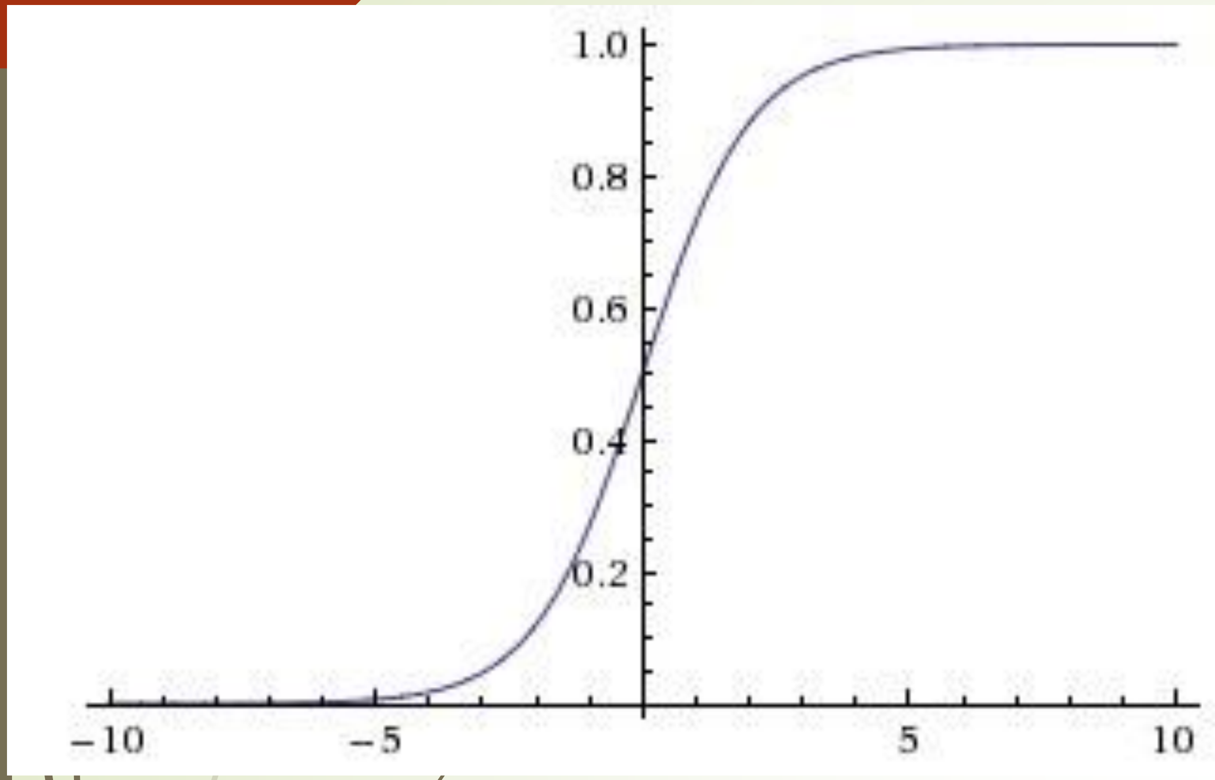
$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

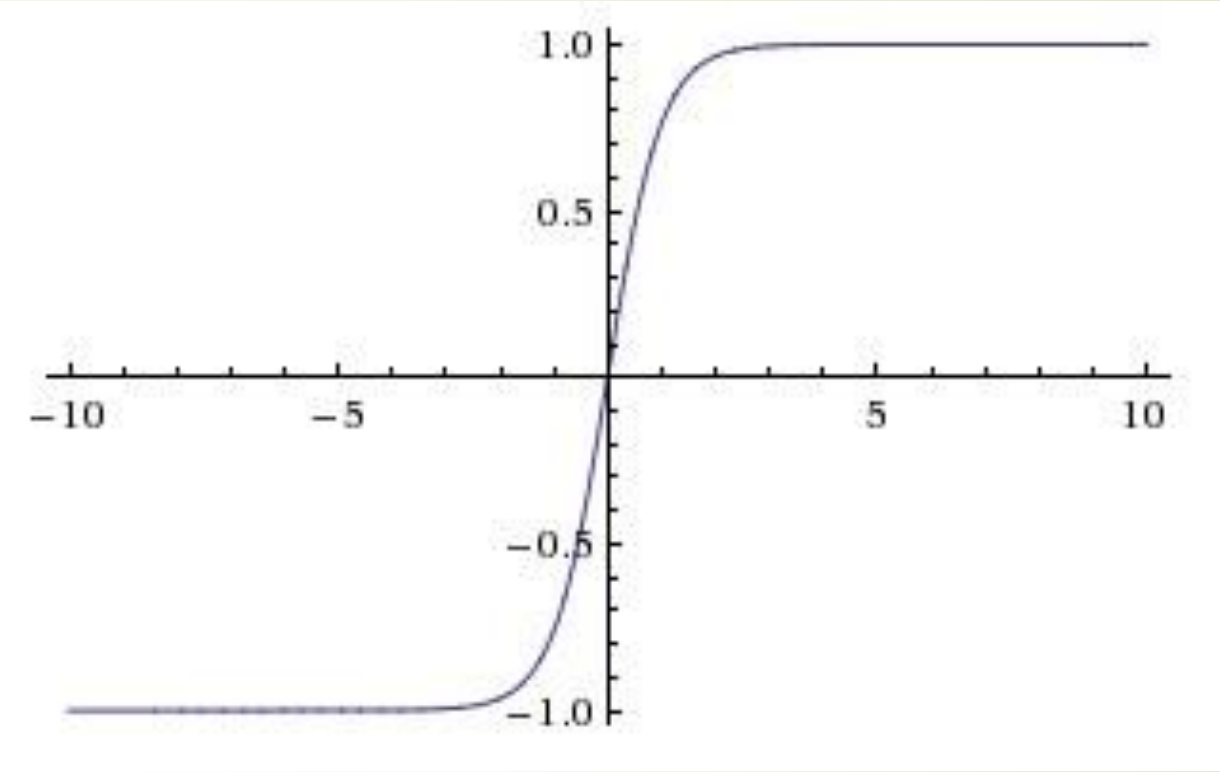
- Đầu ra của mỗi unit: $a_i^{(l)} = f(\mathbf{w}_i^{(l)T} \mathbf{a}^{(l-1)} + b_i^{(l)})$

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

1. Giới thiệu

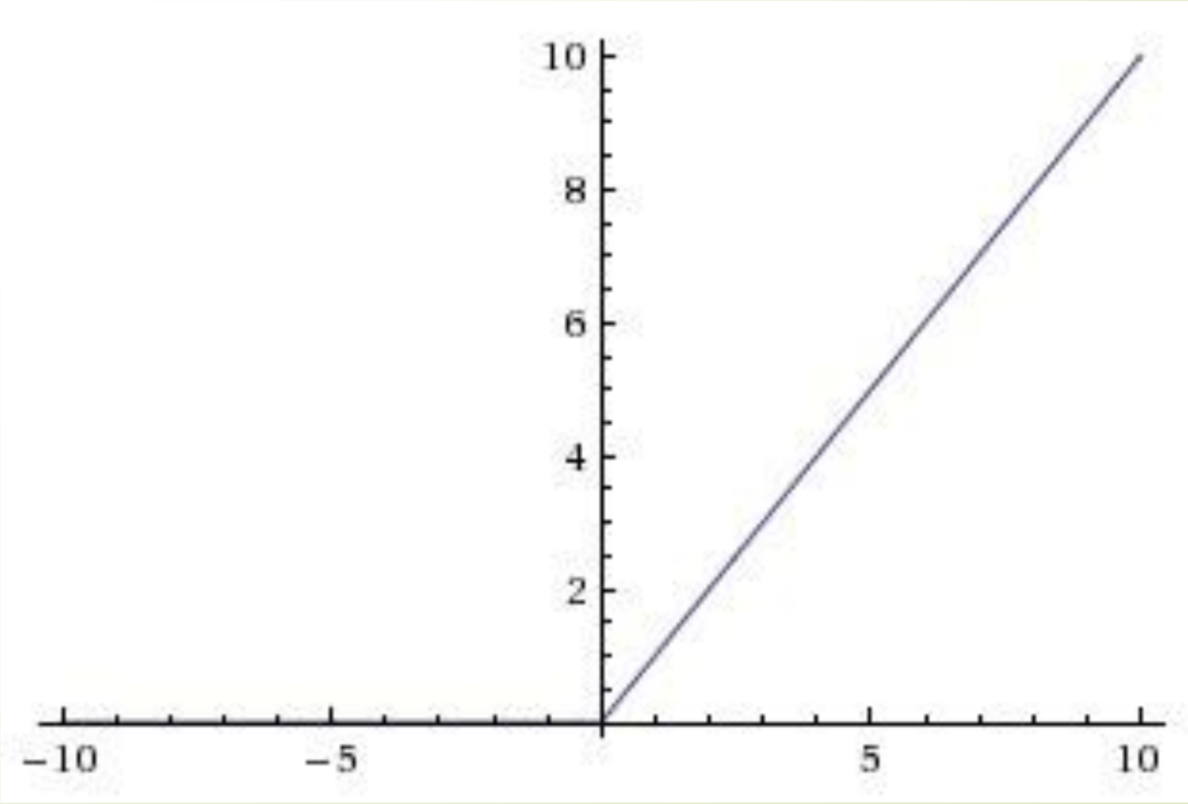


Hàm sigmoid



Hàm tanh

Hàm ReLU



2. Back propagation

❑ Feedforwad

- Đầu vào: $\mathbf{a}^{(0)} = \mathbf{x}$
- Đầu vào của mỗi lớp ẩn:

$$z_i^{(l)} = \mathbf{w}_i^{(l)T}\mathbf{a}^{(l-1)} + b_i^{(l)} \Rightarrow \mathbf{z}^{(l)} = \mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad l = 1, 2, \dots, L$$

- Đầu ra của mỗi lớp ẩn: $\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}), \quad l = 1, 2, \dots, L$
- Đầu ra dự đoán: $\hat{\mathbf{y}} = \mathbf{a}^{(L)}$

2. Back propagation

❑ Loss function $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$

$$J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{y}_n - \hat{\mathbf{y}}_n||_2^2 = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{y}_n - \mathbf{a}_n^{(L)}||_2^2$$

Ma trận trọng số **bias**

❑ Đạo hàm tại layer thứ (*l*)

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}}; \frac{\partial J}{\partial \mathbf{b}^{(l)}}, \quad l = 1, 2, \dots, L$$

Tính toán phức tạp

Back propagation: Tính đạo hàm ngược từ layer cuối cùng đến layer đầu tiên

2. Back propagation

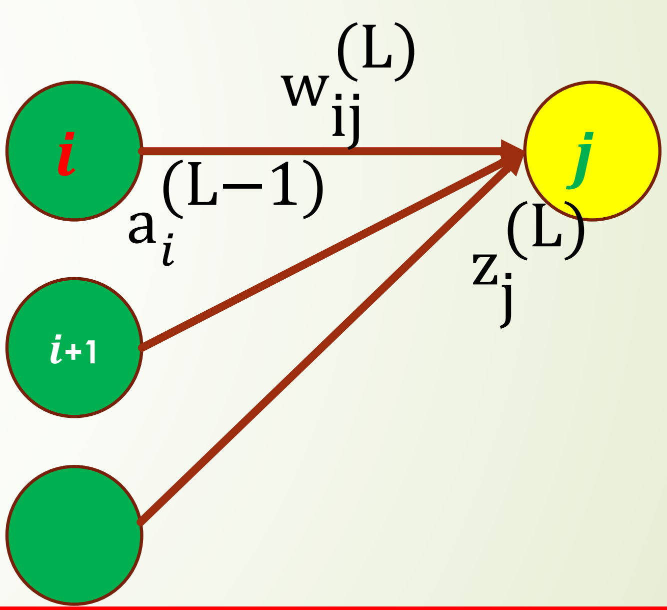
❖ Đạo hàm tại lớp thứ cuối cùng

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = e_j^{(L)} a_i^{(L-1)}$$

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}$$

❖ Đạo hàm tại lớp thứ (l)

- **Đạo hàm theo ma trận trọng số**



$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} . \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} a_i^{(l-1)}$$

$$e_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial a_j^{(l)}} . \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \left(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} . \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) f'(z_j^{(l)}) = \left(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)} \right) f'(z_j^{(l)})$$

$$\left(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)} \right) f'(z_j^{(l)}) = \left(\mathbf{w}_{j:}^{(l+1)} \mathbf{e}^{(l+1)} \right) f'(z_j^{(l)})$$

2. Back propagation

$$\left(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)}\right) f'(z_j^{(l)}) = \left(\mathbf{w}_{j:}^{(l+1)} \mathbf{e}^{(l+1)}\right) f'(z_j^{(l)})$$

Vector hàng

$$\mathbf{e}^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, \dots, e_{d^{(l+1)}}^{(l+1)}]^T \in \mathbb{R}^{d^{(l+1)} \times 1}$$

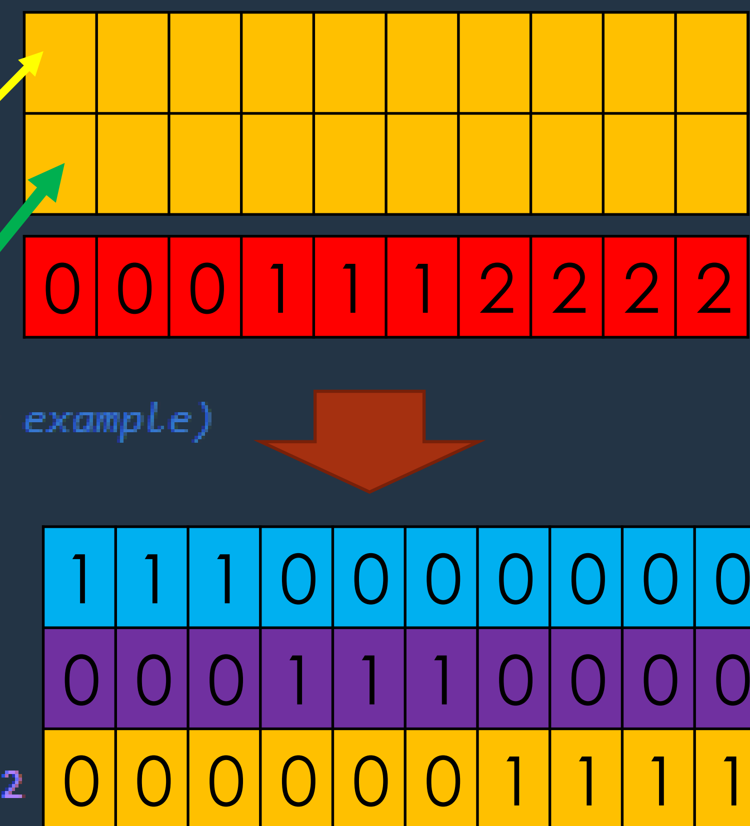
Vector cột

- **Đạo hàm theo bias**

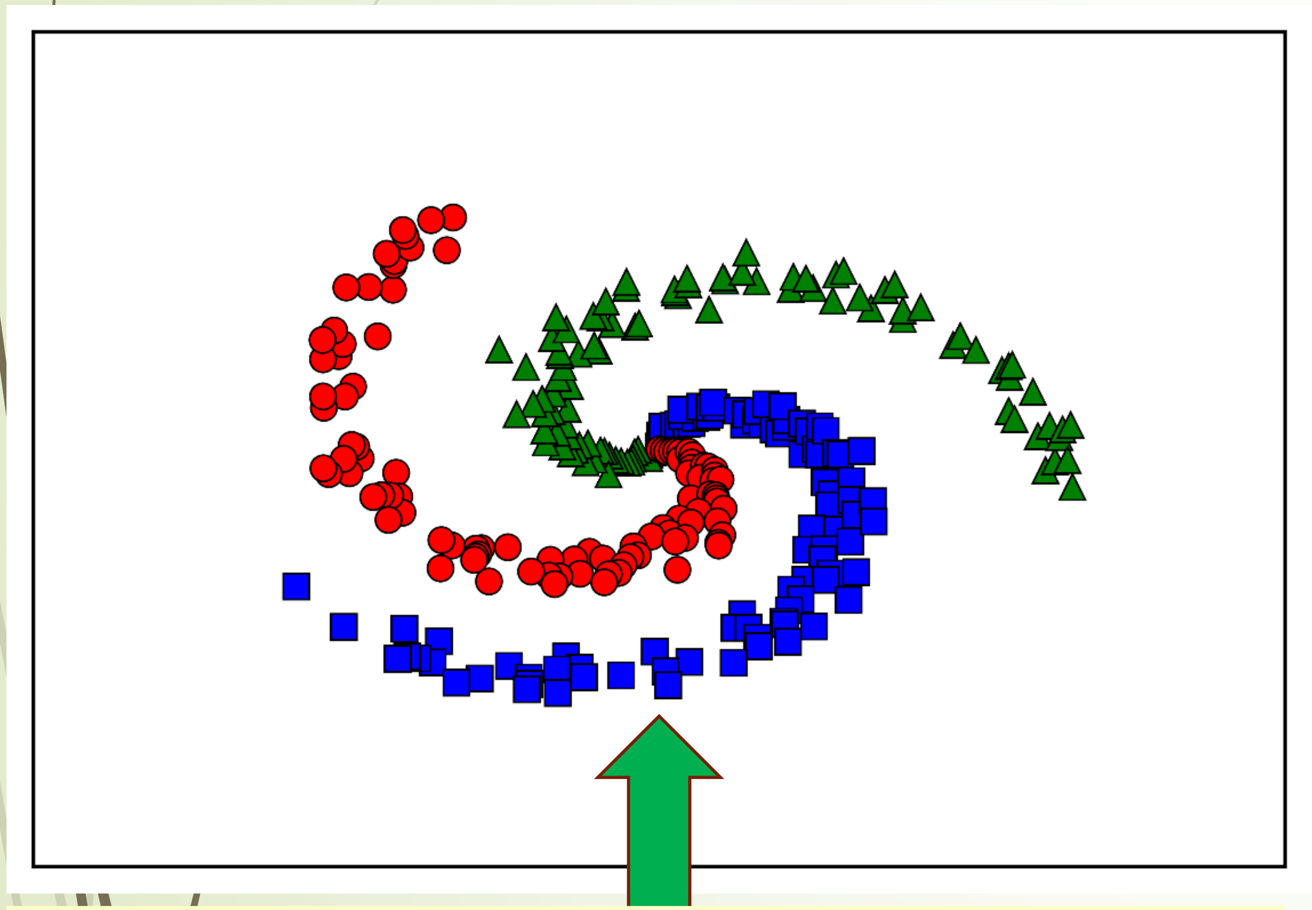
$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)}$$

Ví dụ

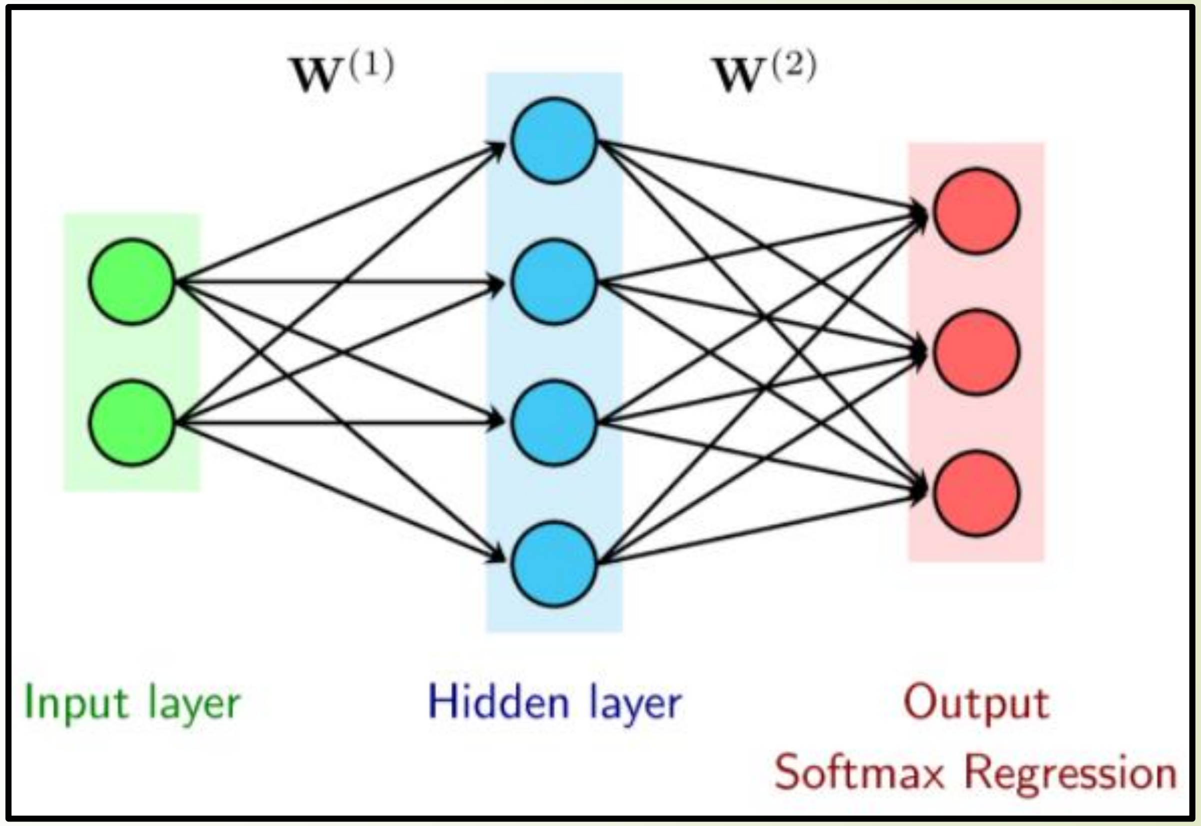
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt

N = 100 # number of points per class
d0 = 2 # dimensionality
C = 3 # number of classes
X = np.zeros((d0, N*C)) # data matrix (each row = single example)
y = np.zeros(N*C, dtype='uint8') # class labels

for j in xrange(C):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2
  X[:,ix] = np.c_[r*np.sin(t), r*np.cos(t)].T
  y[ix] = j
# lets visualize the data:
# plt.scatter(X[:N, 0], X[:N, 1], c=y[:N], s=40, cmap=plt.cm.Spectral)

plt.plot(X[0, :N], X[1, :N], 'bs', markersize = 7);
plt.plot(X[0, N:2*N], X[1, N:2*N], 'ro', markersize = 7);
plt.plot(X[0, 2*N:], X[1, 2*N:], 'g^', markersize = 7);
# plt.axis('off')
plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])
cur_axes = plt.gca()
cur_axes.axes.get_xaxis().set_ticks([])
```



Ví dụ

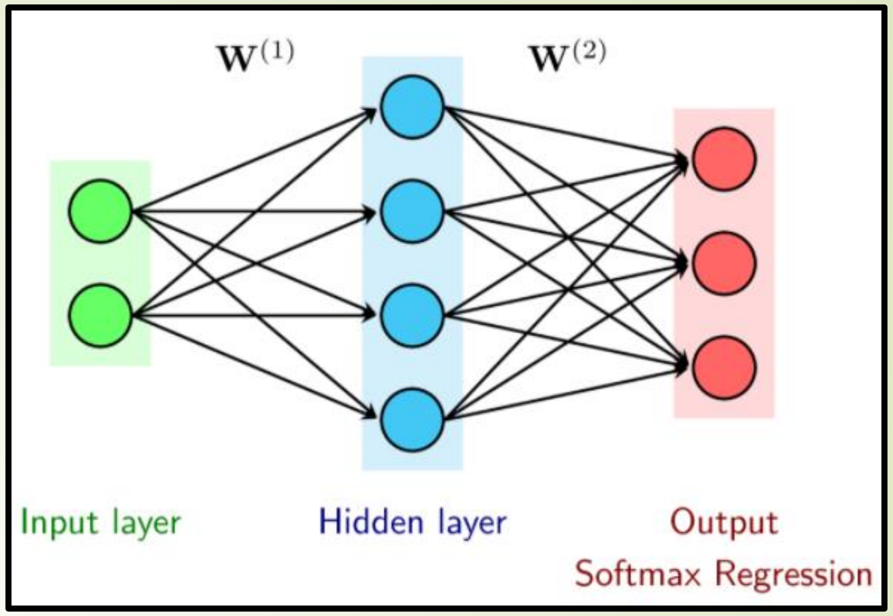


Softmax Regression không thực hiện được vì boundary giữa các class tạo bởi Softmax Regression phải có dạng tuyến tính (linear)



ReLU: $f(s) = \max(s, 0)$;
$f'(s) = 0$ *nếu* $s \leq 0$ *và* $f'(s) = 1$ *nếu* $s > 0$

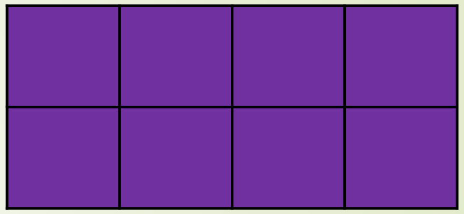
Ví dụ



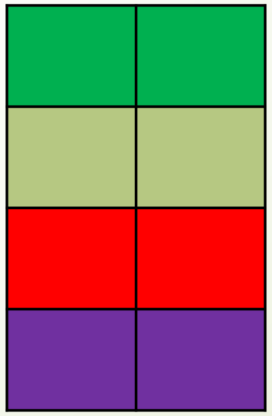
Feed forward

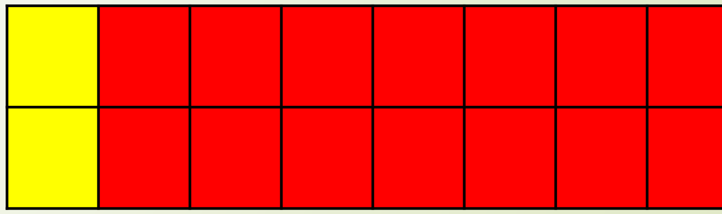
Input: $\boxed{A^{(0)} = X}$

Hidden layer: $\boxed{\mathbf{Z}^{(1)} = \mathbf{W}^{(1)T}\mathbf{X}}$ ⇒ $\boxed{\mathbf{A}^{(1)} = \max(\mathbf{Z}^{(1)}, \mathbf{0})}$



Output layer: $\boxed{\mathbf{Z}^{(2)} = \mathbf{W}^{(2)T}\mathbf{A}^{(1)}}$ ⇒ $\boxed{\hat{\mathbf{Y}} = \mathbf{A}^{(2)} = \text{softmax}(\mathbf{Z}^{(2)})}$





Loss function

$$J \triangleq J(\mathbf{W}, \mathbf{b}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji}\log(\hat{y}_{ji})$$

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \sum_{i=1}^{N} \mathbf{x}_i \mathbf{e}_i^T = \mathbf{X}\mathbf{E}^T$$

⇒

$$\frac{\partial J(W)}{\partial W} = X(\hat{Y} - Y)$$

Ví dụ

Back propagation

Output:

$$\mathbf{E}^{(2)} = \frac{\partial J}{\partial \mathbf{Z}^{(2)}} = \frac{1}{N}(\hat{\mathbf{Y}} - \mathbf{Y})$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \mathbf{A}^{(1)}\mathbf{E}^{(2)T}$$

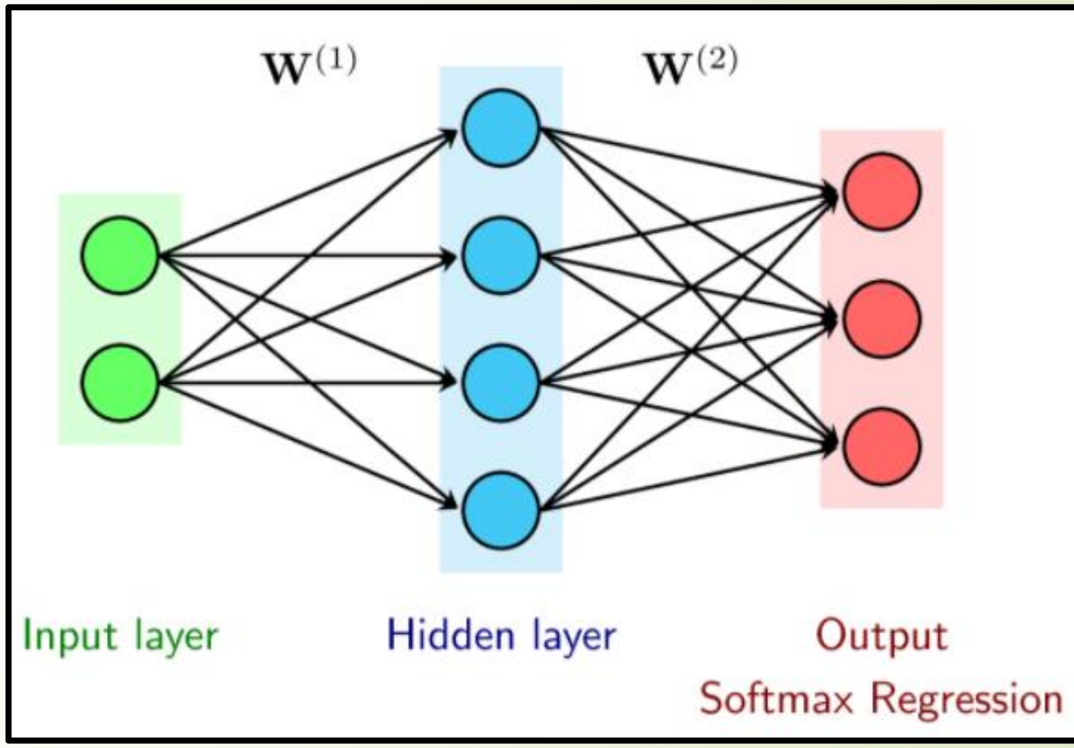
$$\frac{\partial J}{\partial \mathbf{b}^{(2)}} = \sum_{n=1}^{N} \mathbf{e}_n^{(2)}$$

Hidden layer:

$$\mathbf{E}^{(1)} = \left(\mathbf{W}^{(2)}\mathbf{E}^{(2)}\right) \odot f'(\mathbf{Z}^{(1)})$$

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \mathbf{A}^{(0)}\mathbf{E}^{(1)T} = \mathbf{X}\mathbf{E}^{(1)T}$$

$$\frac{\partial J}{\partial \mathbf{b}^{(1)}} = \sum_{n=1}^{N} \mathbf{e}_n^{(1)}$$

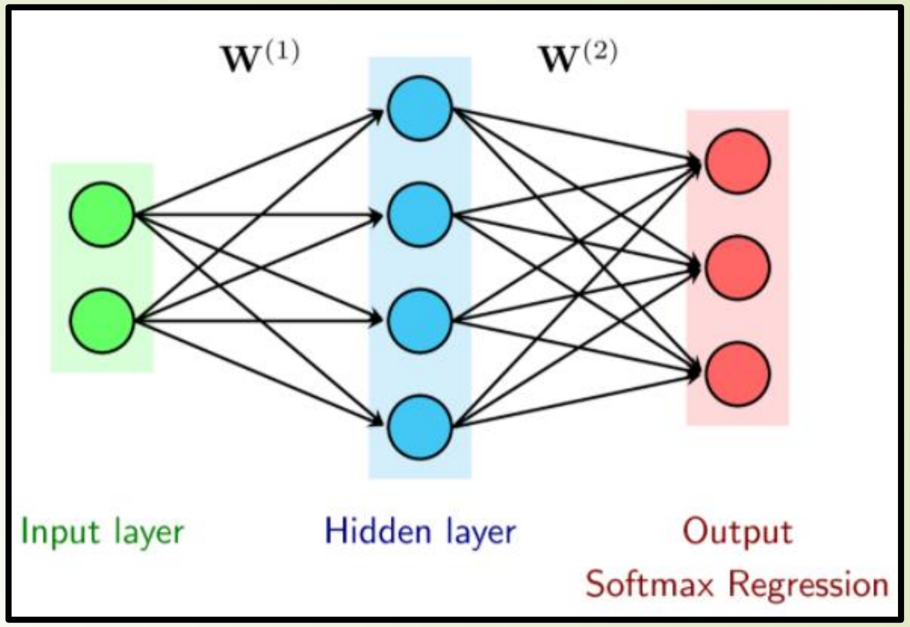


Ví dụ

```
def softmax(V):
    e_V = np.exp(V - np.max(V, axis = 0, keepdims = True))
    Z = e_V / e_V.sum(axis = 0)
    return Z

## One-hot coding
from scipy import sparse
def convert_labels(y, C = 3):
    Y = sparse.coo_matrix((np.ones_like(y),
        (y, np.arange(len(y)))), shape = (C, len(y))).toarray()
    return Y

# cost or loss function
def cost(Y, Yhat):
    return -np.sum(Y*np.log(Yhat))/Y.shape[1]
```

```python
d0 = 2
d1 = h = 100 # size of hidden layer
d2 = C = 3
# initialize parameters randomly
W1 = 0.01*np.random.randn(d0, d1)
b1 = np.zeros((d1, 1))
W2 = 0.01*np.random.randn(d1, d2)
b2 = np.zeros((d2, 1))

Y = convert_labels(y, C)
N = X.shape[1]
eta = 1 # learning rate
for i in xrange(10000):
    ## Feedforward
    Z1 = np.dot(W1.T, X) + b1
    A1 = np.maximum(Z1, 0)
    Z2 = np.dot(W2.T, A1) + b2
    Yhat = softmax(Z2)

    # print loss after each 1000 iterations
    if i %1000 == 0:
        # compute the loss: average cross-entropy loss
        loss = cost(Y, Yhat)
        print("iter %d, loss: %f" %(i, loss))
```

Ví dụ




```
# backpropagation
E2 = (Yhat - Y )/N
dW2 = np.dot(A1, E2.T)
db2 = np.sum(E2, axis = 1, keepdims = True)
E1 = np.dot(W2, E2)
E1[Z1 <= 0] = 0 # gradient of ReLU
dW1 = np.dot(X, E1.T)
db1 = np.sum(E1, axis = 1, keepdims = True)

# Gradient Descent update
W1 += -eta*dW1
b1 += -eta*db1
W2 += -eta*dW2
b2 += -eta*db2
```

Ví dụ

```
iter 0, loss: 1.098815
iter 1000, loss: 0.150974
iter 2000, loss: 0.057996
iter 3000, loss: 0.039621
iter 4000, loss: 0.032148
iter 5000, loss: 0.028054
iter 6000, loss: 0.025346
iter 7000, loss: 0.023311
iter 8000, loss: 0.021727
iter 9000, loss: 0.020585
```

```
Z1 = np.dot(W1.T, X) + b1
A1 = np.maximum(Z1, 0)
Z2 = np.dot(W2.T, A1) + b2
predicted_class = np.argmax(Z2, axis=0)
print('training accuracy: %.2f %%' % (100*np.mean(predicted_class == y)))
```

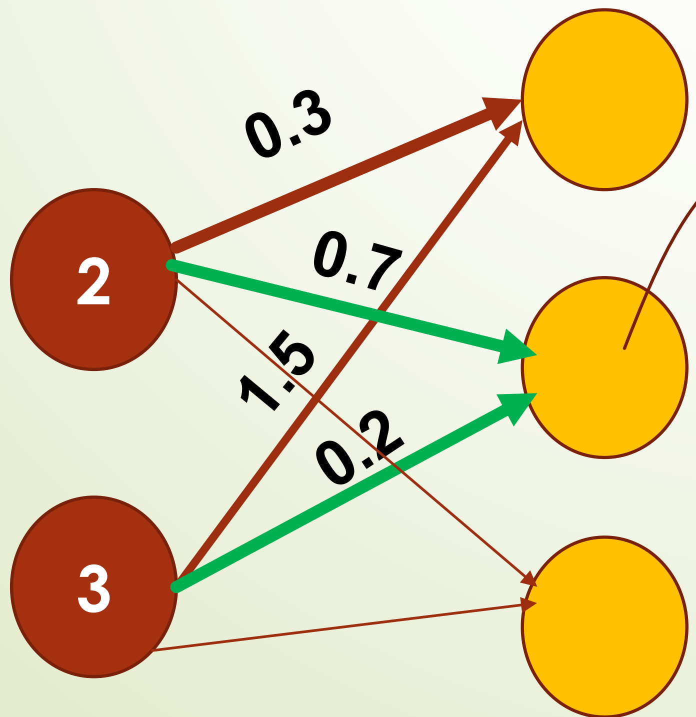
```
training accuracy: 99.33 %
```

0.6	0.5	0.4	0.2						
0.2	0.2	0.5	0.1						
0.2	0.3	0.1	0.7						

Predict	0	0	1	2						
label	0	0	2	1						
	1	1	0	0						

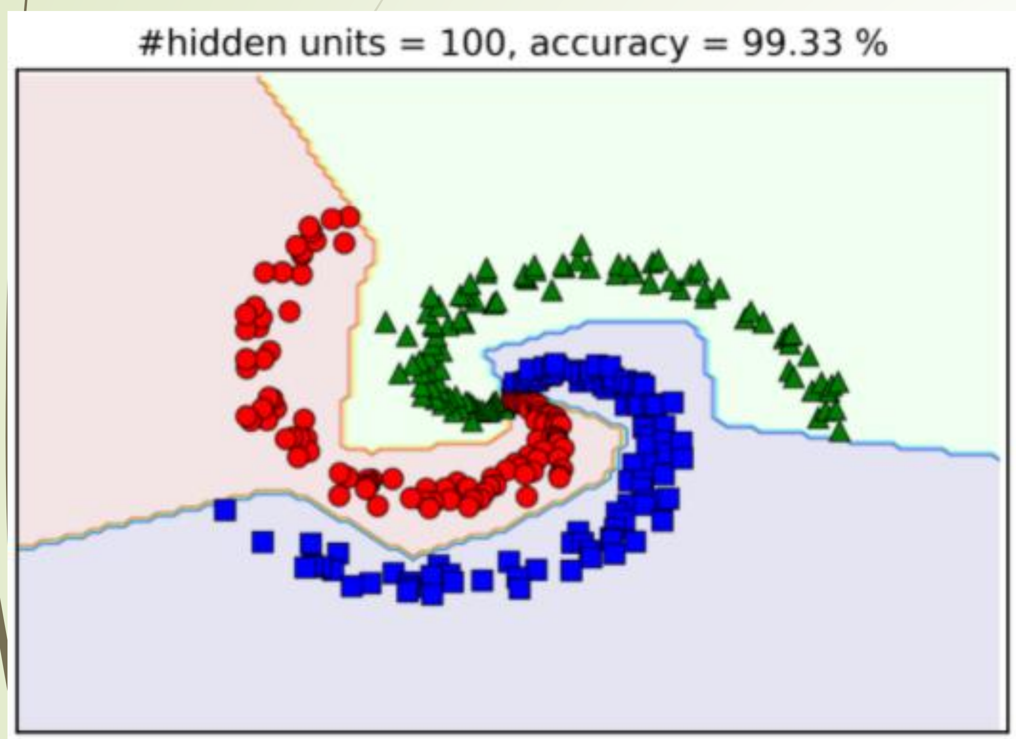
$$\begin{bmatrix}0.3 & 1.5\\ 0.7 & 0.2\\ 0.9 & 0.1\end{bmatrix} \times \begin{bmatrix}2\\ 3\end{bmatrix} = \begin{bmatrix}5.1\\ 2.0\\ 2.1\end{bmatrix}$$

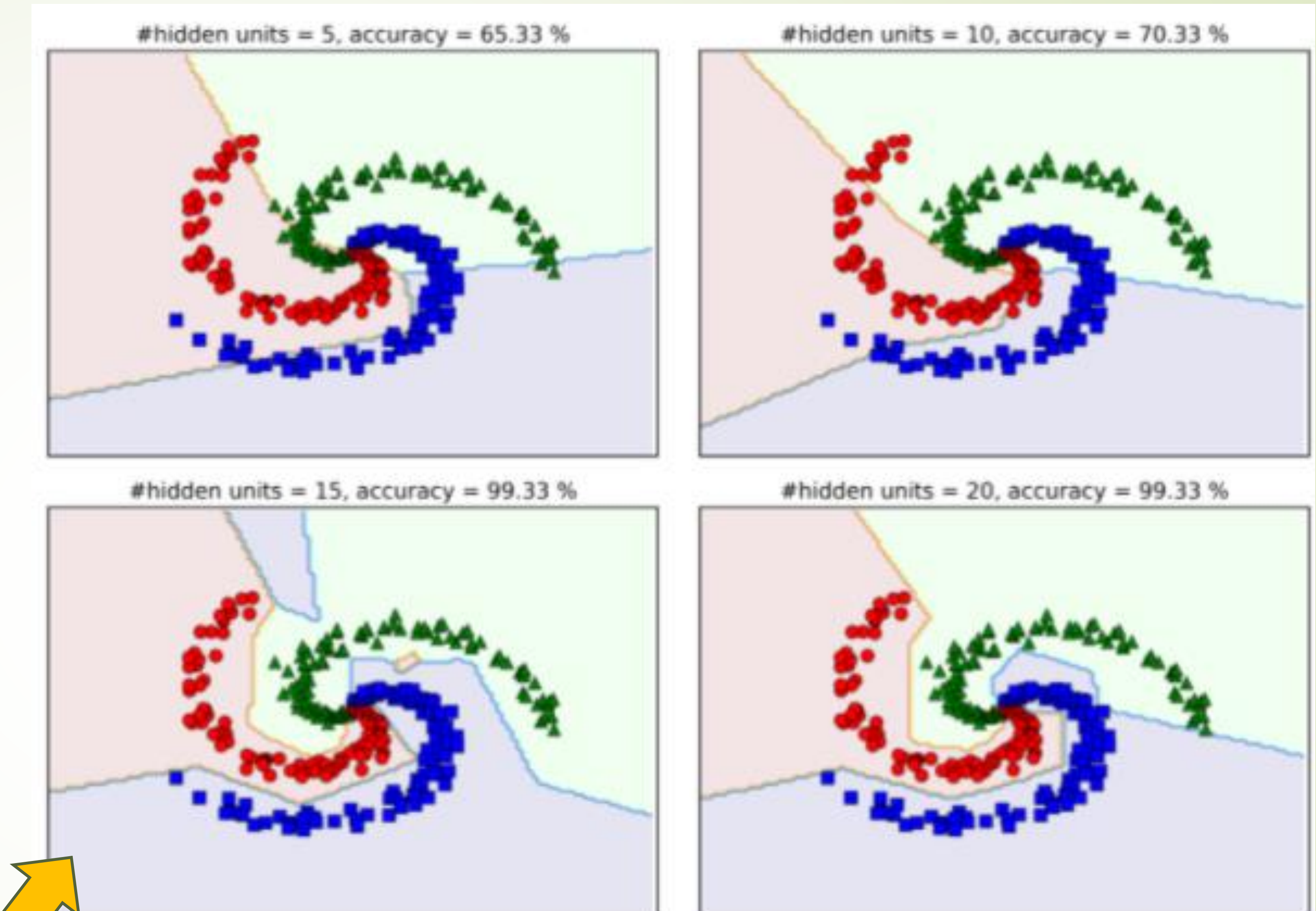
Z



0.3	0.7	0.9
1.5	0.2	0.1

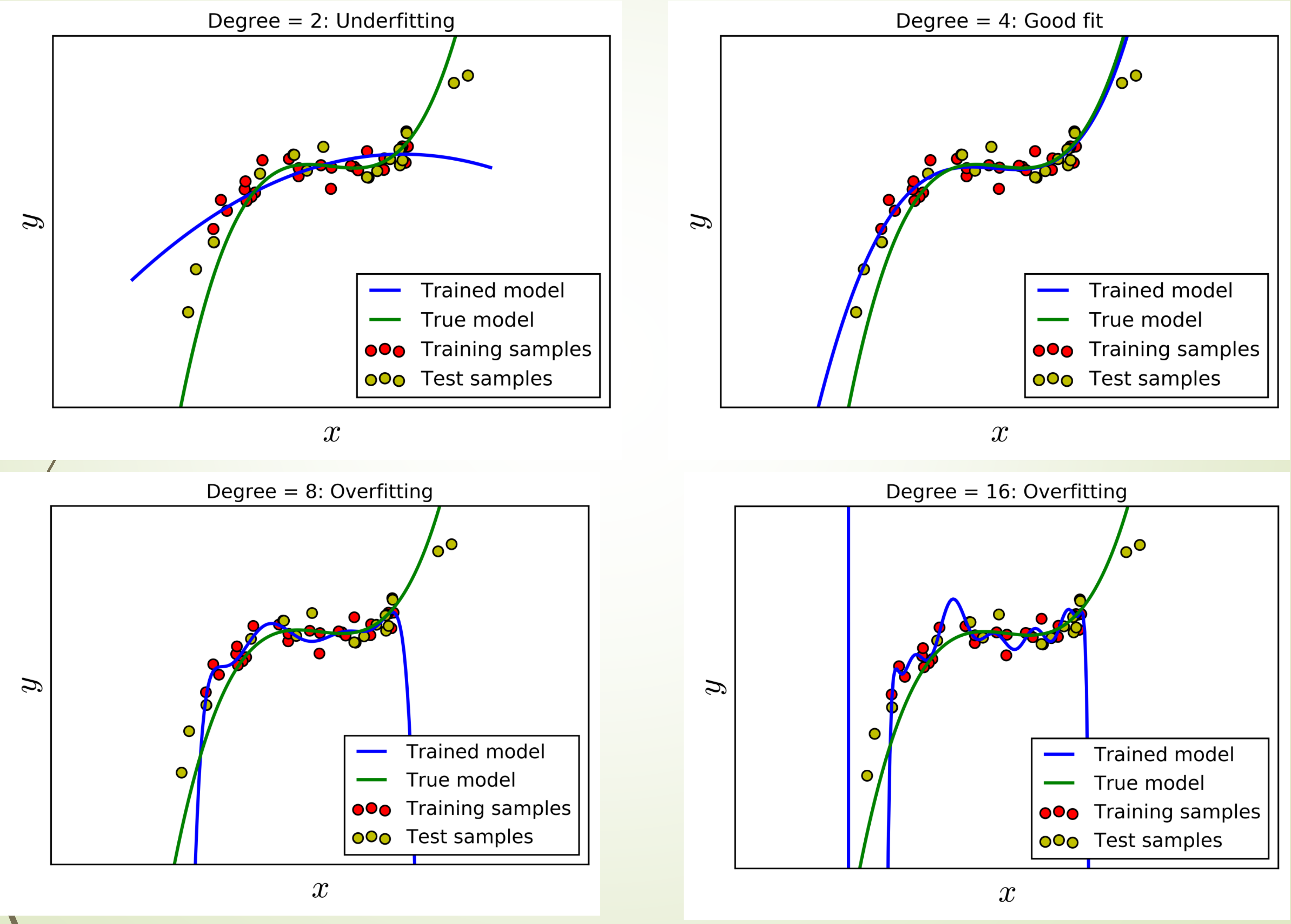
Ví dụ





Số lượng hiddent unit tăng, độ chính xác của mô hình tăng

Overfitting



Overfitting

- Overfitting: mô hình tìm được *quá khớp* với dữ liệu training
- Quá khớp có thể dẫn đến việc dự đoán nhầm nhiễu, và chất lượng mô hình không còn tốt trên dữ liệu test
- Overfitting xảy ra khi mô hình quá phức tạp để mô phỏng training data

Tránh hiện tượng Overfitting?

Regression:

$$\text{train error} = \frac{1}{N_{\text{train}}} \sum_{\text{training set}} \|\mathbf{y} - \hat{\mathbf{y}}\|_p^2$$

Classification:

$$J \triangleq J(\mathbf{W}, \mathbf{b}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ji} \log(\hat{y}_{ji})$$

Overfitting

Regression:

$$\text{train error} = \frac{1}{N_{\text{train}}} \sum_{\text{training set}} \|\mathbf{y} - \hat{\mathbf{y}}\|_p^2$$

$$\text{test error} = \frac{1}{N_{\text{test}}} \sum_{\text{test set}} \|\mathbf{y} - \hat{\mathbf{y}}\|_p^2$$

- ❑ **Overfitting:** train error nhỏ, test error lớn
- ❑ **Underfitting:** train error lớn, test error lớn
- ❑ **Fitting:** train error nhỏ, test error nhỏ

Validation

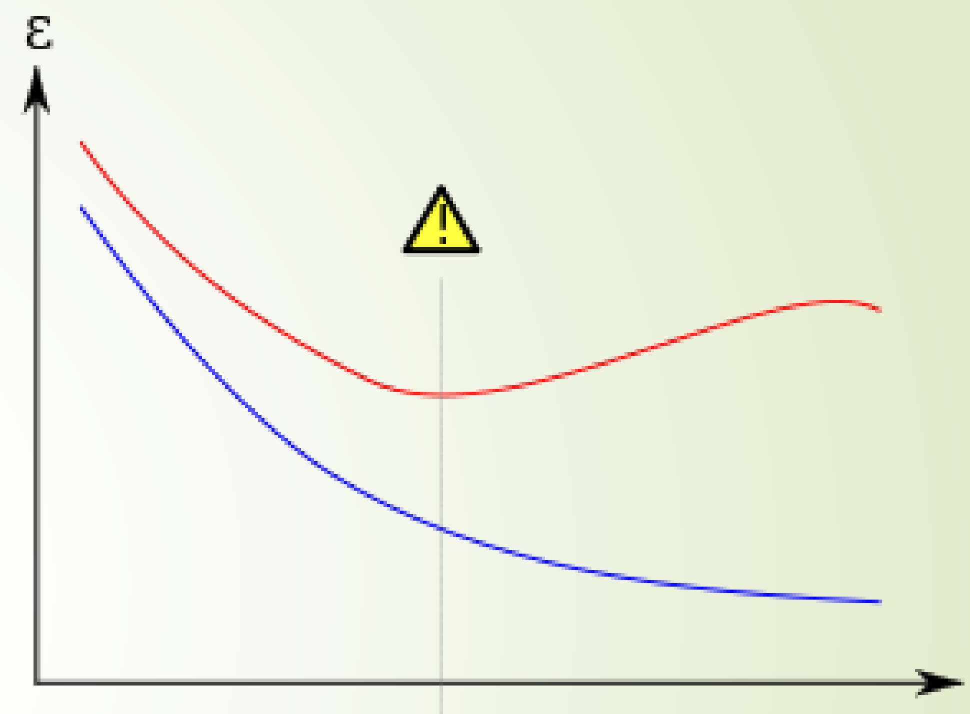
Regularization

Validation

- **Validation**
  - Chia tập huấn luyện (training set): training set và validation set
  - Xây dựng mô hình: training error và validation error đều nhỏ
- **Cross-validation**
  - Chia tập training thành *k* phần bằng nhau
  - Lấy 1 phần làm dữ liệu validation; *(k-1)* phần còn lại được dùng làm dữ liệu huấn luyện
  - Thực hiện *k* lần huấn luyện: → *k-fold cross validation*
  - *Mô hình cuối được xác định dựa trên trung bình của các train error và validation error*

Regularization

- **Early stopping:** dừng thuật toán trước khi hàm mất mát đạt giá trị quá nhỏ, giúp tránh overfitting



- **Regularized loss function**

$$J_{\text{reg}}(\theta) = J(\theta) + \lambda R(\theta)$$

  - *l2* Regularization

$$R(\mathbf{w}) = \|\mathbf{w}\|_2^2$$

$$\frac{\partial J_{\text{reg}}}{\partial \mathbf{w}} = \frac{\partial J}{\partial \mathbf{w}} + \lambda \mathbf{w}$$

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

weight decay

  - Tikhonov regularization

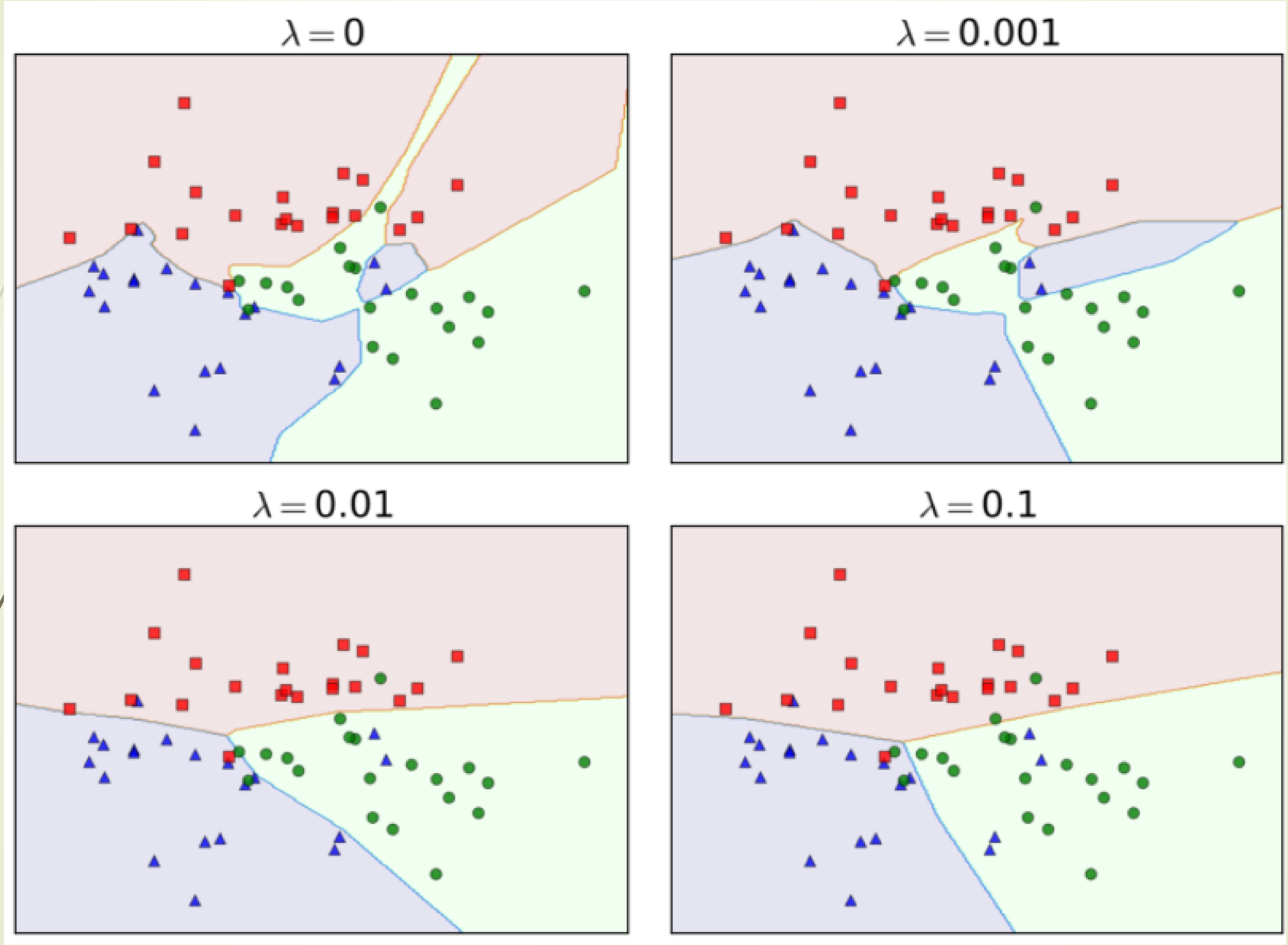
$$\lambda R(\mathbf{w}) = \|\Gamma \mathbf{w}\|_2^2$$

Ví dụ

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(4)

means = [[-1, -1], [1, -1], [0, 1]]
cov = [[1, 0], [0, 1]]
N = 20
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)
```

$$\boxed{\lambda R(\mathbf{W}) = \lambda \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_F^2}$$

Ví dụ



https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/15_overfitting/Weight%20Decay.ipynb

Bài tập

Bài 1

- Tạo 100 điểm ngẫu nhiên xung quanh đường thẳng 3x+5
- Viết chương trình Python sử dụng thuật toán gradient descent để xây dựng mô hình hồi quy tuyến tính (linear regression) với 100 điểm dữ liệu đầu vào trên

Bài 2

- Cho hàm: f(x) $= 3x^2 + 8\sin 2x$
- Sử dụng thuật toán gradient descent tìm giá trị nhỏ nhất của hàm $f(x)$