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

Logistic Regression

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

1. Giới thiệu

- Mô hình tuyến tính (linear models):

$$y = f(\mathrm{w}^T \mathrm{x})$$

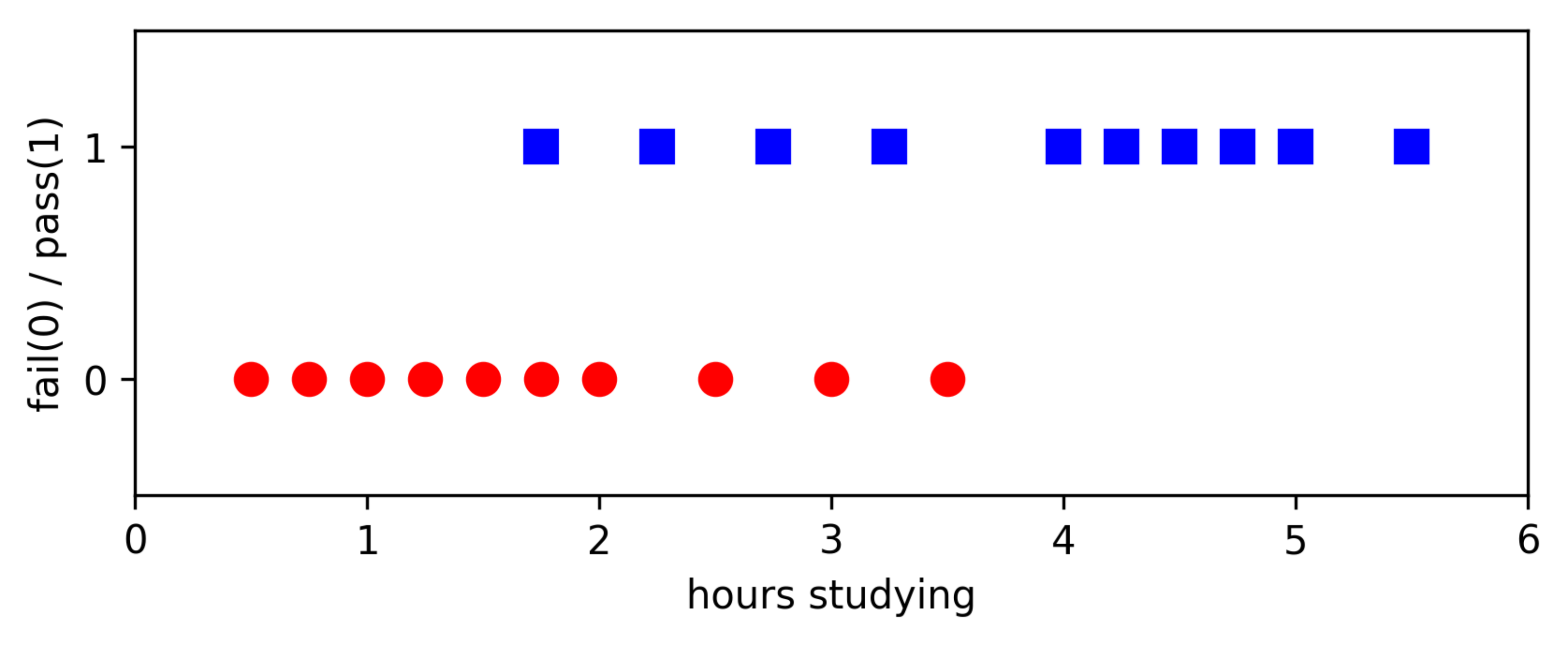
  - Linear Regression: $y = \mathrm{w}^T x$ → dự đoán đầu ra không bị giới hạn
  - Perceptron Learning Algorithm: hard threshold → binary classification

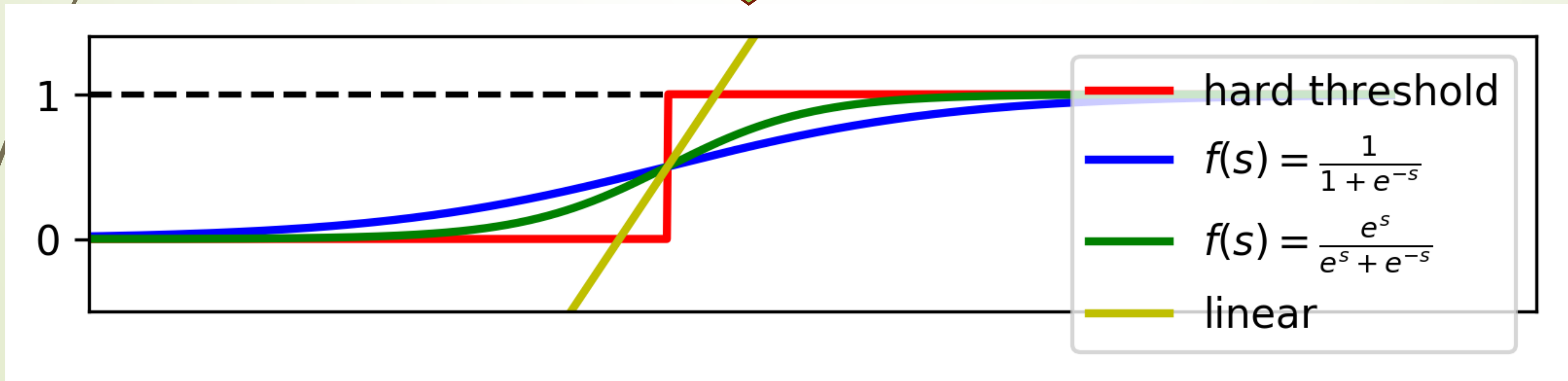
Một nhóm 20 sinh viên dành thời gian trong khoảng từ 0 đến 6 giờ cho việc ôn thi. Thời gian ôn thi này ảnh hưởng đến xác suất sinh viên vượt qua kỳ thi như thế nào?

Kết quả thu được như sau:

Hours	Pass	Hours	Pass
.5	0	2.75	1
.75	0	3	0
1	0	3.25	1
1.25	0	3.5	0
1.5	0	4	1
1.75	0	4.25	1
1.75	1	4.5	1
2	0	4.75	1
2.25	1	5	1
2.5	0	5.5	1

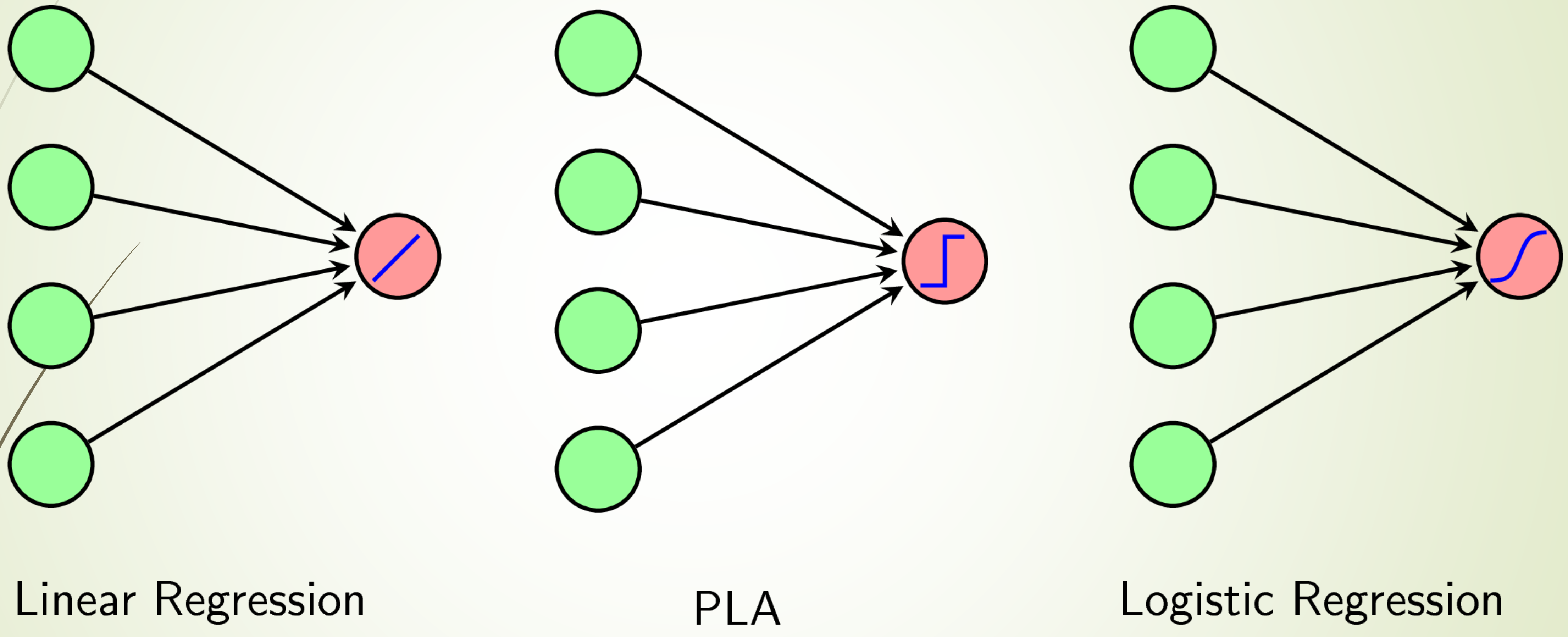
1. Giới thiệu





Cả hai mô hình linear regression và PLA (hard threshold)đều không phù hợp.

1. Giới thiệu



1. Giới thiệu

- **Hàm Sigmoid:**

$$f(s) = \frac{1}{1+e^{-s}} \triangleq \sigma(s)$$

$$\lim_{s \to -\infty} \sigma(s) = 0; \quad \lim_{s \to +\infty} \sigma(s) = 1$$

$$\sigma'(s) = \frac{e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} \frac{e^{-s}}{1+e^{-s}} = \sigma(s)(1-\sigma(s))$$

- **Hàm Tanh:**

$$\tanh(s) = \frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$$

$$\tanh(s) = 2\sigma(2s) - 1$$

2. Hàm mất mát

Xác suất để một điểm dữ liệu x rơi vào Class 1:

$$P(y_i = 1|\mathrm{x}_i;\mathrm{w}) = f(\mathrm{w}^T x_i) \qquad (1)$$

Xác suất để một điểm dữ liệu x rơi vào Class 0:

$$P(y_i = 0|\mathrm{x}_i;\mathrm{w}) = 1 - f(\mathrm{w}^T x_i) \qquad (2)$$

Đặt $z_i = f(\mathrm{w}^T \mathrm{x}_i)$, viết lại biểu thức (1) và (2)

$$P(y_i|\mathrm{x}_i;\mathrm{w}) = z_i^{y_i}(1 - z_i)^{1-y_i}$$

Xét tập huấn luyện:

$$\mathrm{X} = [\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \dots, \mathrm{x}_N]$$

với các đầu ra mong muốn (nhãn):

$$\mathrm{y} = [y_1, y_2, y_3, \dots, y_N]$$

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} \, \boldsymbol{P}(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})$$

maximum likelihood estimation với hàm số phía sau argmax được gọi là *likelihood function*

2. Hàm mất mát

Các điểm dữ liệu độc lập với nhau:

$$\mathrm{P}(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_{i=1}^{N} P(y_i|\mathbf{x}_i;\mathbf{w}) = \prod_{i=1}^{N} z_i^{y_i}(1-z_i)^{1-y_i}$$

Mục tiêu: Tối thiểu hóa hàm mất mát

Hàm mất mát:

$$\mathrm{J}(\mathbf{w}) = -log\mathrm{P}(\mathbf{y}|\mathbf{X};\mathbf{w}) = -\sum_{i=1}^{N}[y_i log z_i + (1-y_i)log(1-z_i)]$$

cross entropy ⇨ đo *khoảng cách* giữa hai phân phối

3. Tối ưu hàm mất mát

Hàm mất mát tại một điểm:

$$J(\mathrm{w};\, \mathrm{x}_i, y_i) = -\big(y_i log z_i + (1 - y_i) log(1 - z_i)\big)$$

Đạo hàm

$$\frac{\partial J(\mathbf{w}, \mathbf{x}_i, y_i)}{\partial \mathbf{w}} = -\left(\frac{y_i}{z_i} - \frac{1 - y_i}{1 - z_i}\right)\frac{\partial z_i}{\partial \mathbf{w}} = \frac{z_i - y_i}{z_i(1 - z_i)}\frac{\partial z_i}{\partial \mathbf{w}}$$

Tìm hàm $z =$ … $x)$ sao cho mẫu số … triệt tiêu

Đặt s= $w^T x$

$$\frac{\partial z_i}{\partial \mathbf{w}} = \frac{\partial z_i}{\partial \mathbf{s}} \cdot \frac{\partial s}{\partial \mathbf{w}} = \frac{\partial z_i}{\partial \mathbf{s}}\mathbf{x}$$

$$\frac{\partial z}{\partial \mathbf{s}} = z(1 - z)$$

3. Tối ưu hàm mất mát

$$\frac{\partial z}{z(1-z)} = \partial s \Leftrightarrow \left(\frac{1}{z} + \frac{1}{1-z}\right)\partial z = \partial s \Leftrightarrow \log z - \log(1-z) = s$$

$$\log \frac{z}{1-z} = s \Leftrightarrow \frac{z}{1-z} = e^s \Leftrightarrow z = e^s(1-z) \Leftrightarrow z = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}} = \sigma(s)$$

Thay vào phương trình Gradient

$$\frac{\partial J(\mathbf{w}; \mathbf{x}_i, y_i)}{\partial \mathbf{w}} = (z_i - y_i)\mathbf{x}_i$$

Thuật toán GD $\mathbf{w} = \mathbf{w} + \eta(y_i - z_i)\mathbf{x}_i$

4. Ví dụ

Hours	Pass	Hours	Pass
.5	0	2.75	1
.75	0	3	0
1	0	3.25	1
1.25	0	3.5	0
1.5	0	4	1
1.75	0	4.25	1
1.75	1	4.5	1
2	0	4.75	1
2.25	1	5	1
2.5	0	5.5	1

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

X = np.array([[0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50,
              2.75, 3.00, 3.25, 3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50]])
y = np.array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1])

# extended data
X = np.concatenate((np.ones((1, X.shape[1])), X), axis = 0)
```

4. Ví dụ

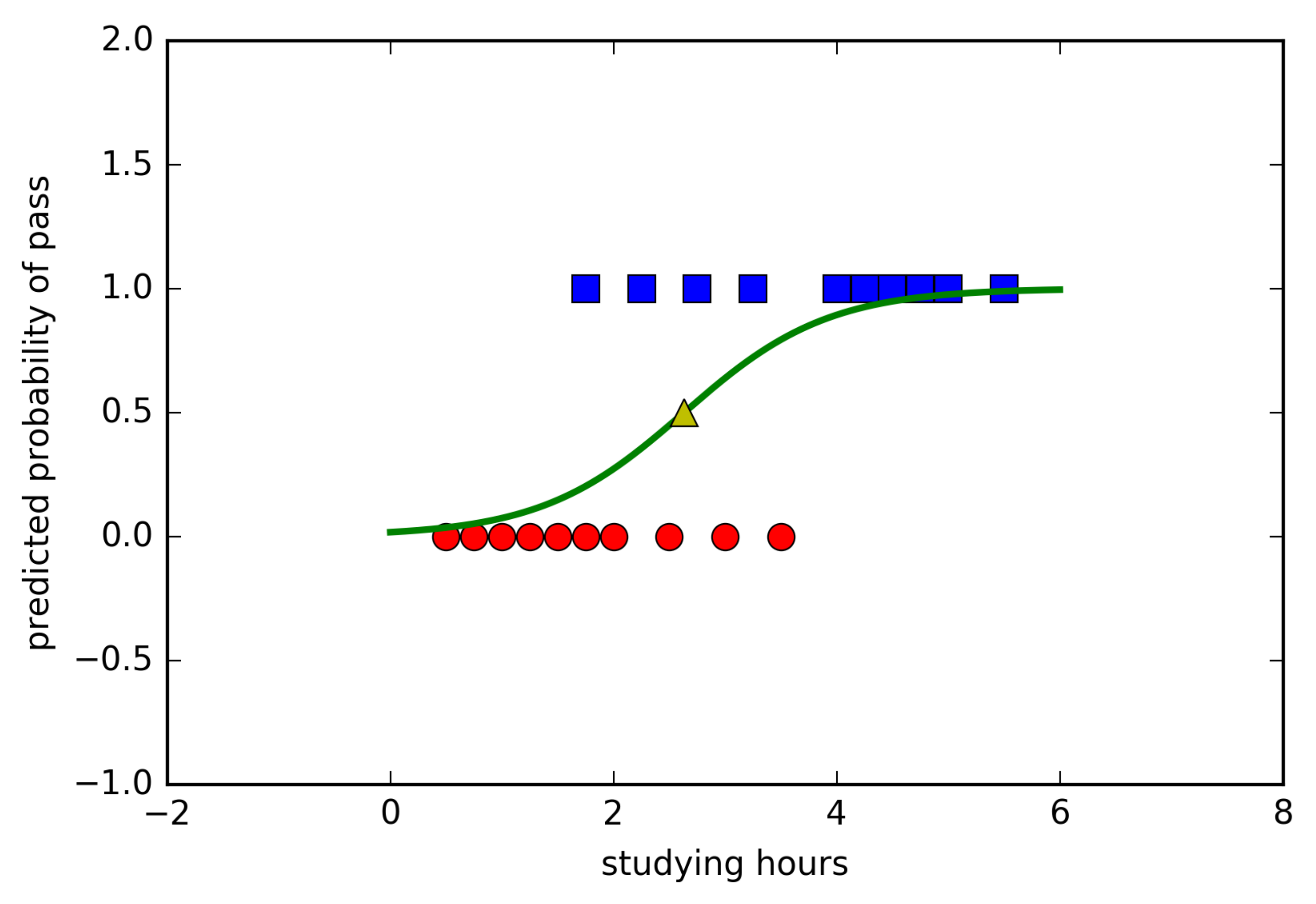
```
def sigmoid(s):
    return 1/(1 + np.exp(-s))

def logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count = 10000):
    w = [w_init]
    it = 0
    N = X.shape[1]
    d = X.shape[0]
    count = 0
    check_w_after = 20
    while count < max_count:
        # mix data
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[:, i].reshape(d, 1)
            yi = y[i]
            zi = sigmoid(np.dot(w[-1].T, xi))
            w_new = w[-1] + eta*(yi - zi)*xi
            count += 1
            # stopping criteria
            if count%check_w_after == 0:
                if np.linalg.norm(w_new - w[-check_w_after]) < tol:
                    return w
            w.append(w_new)
    return w
eta = .05
d = X.shape[0]
w_init = np.random.randn(d, 1)

w = logistic_sigmoid_regression(X, y, w_init, eta)
print(w[-1])
```

4. Ví dụ

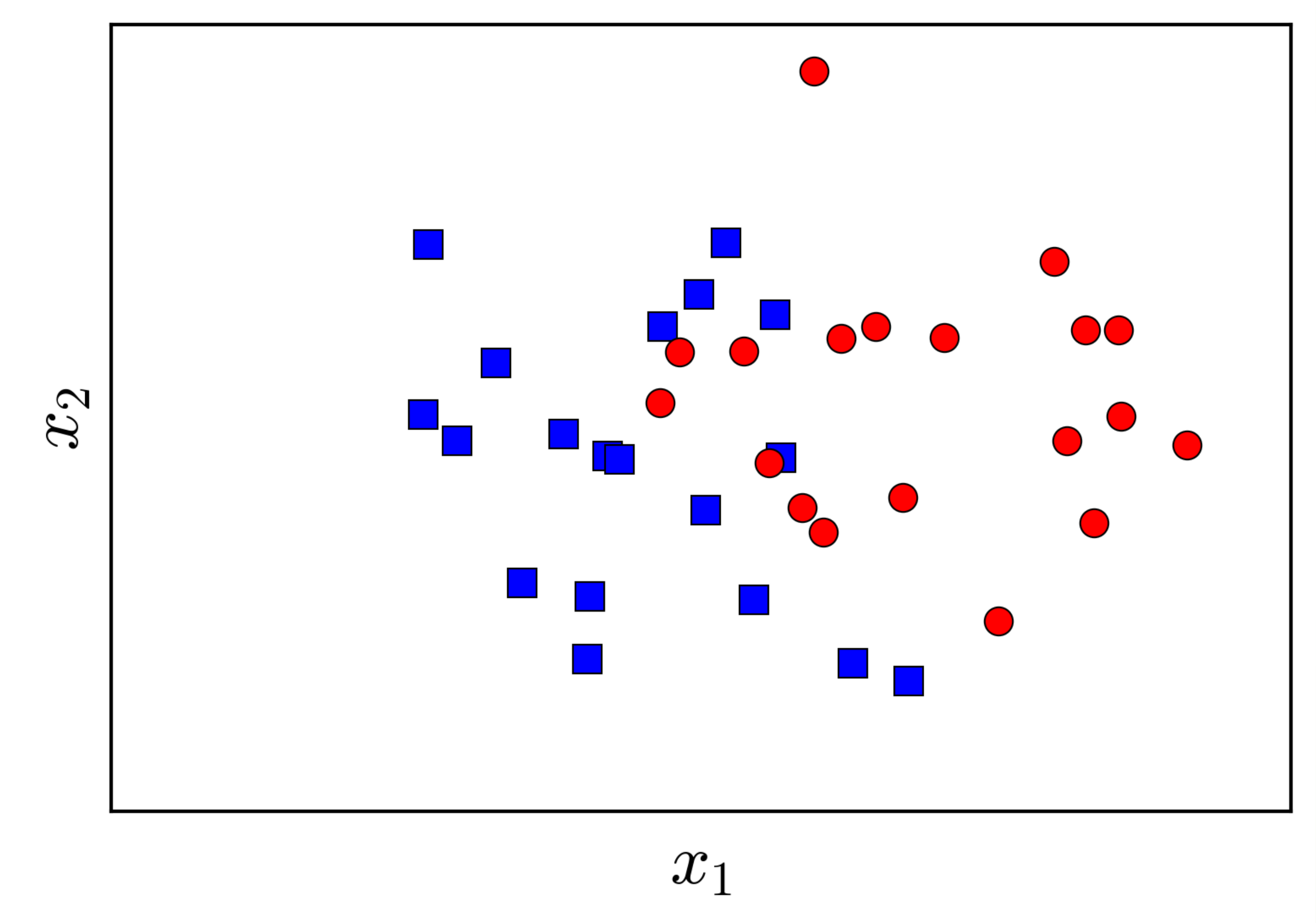
```
print(sigmoid(np.dot(w[-1].T, X)))
```

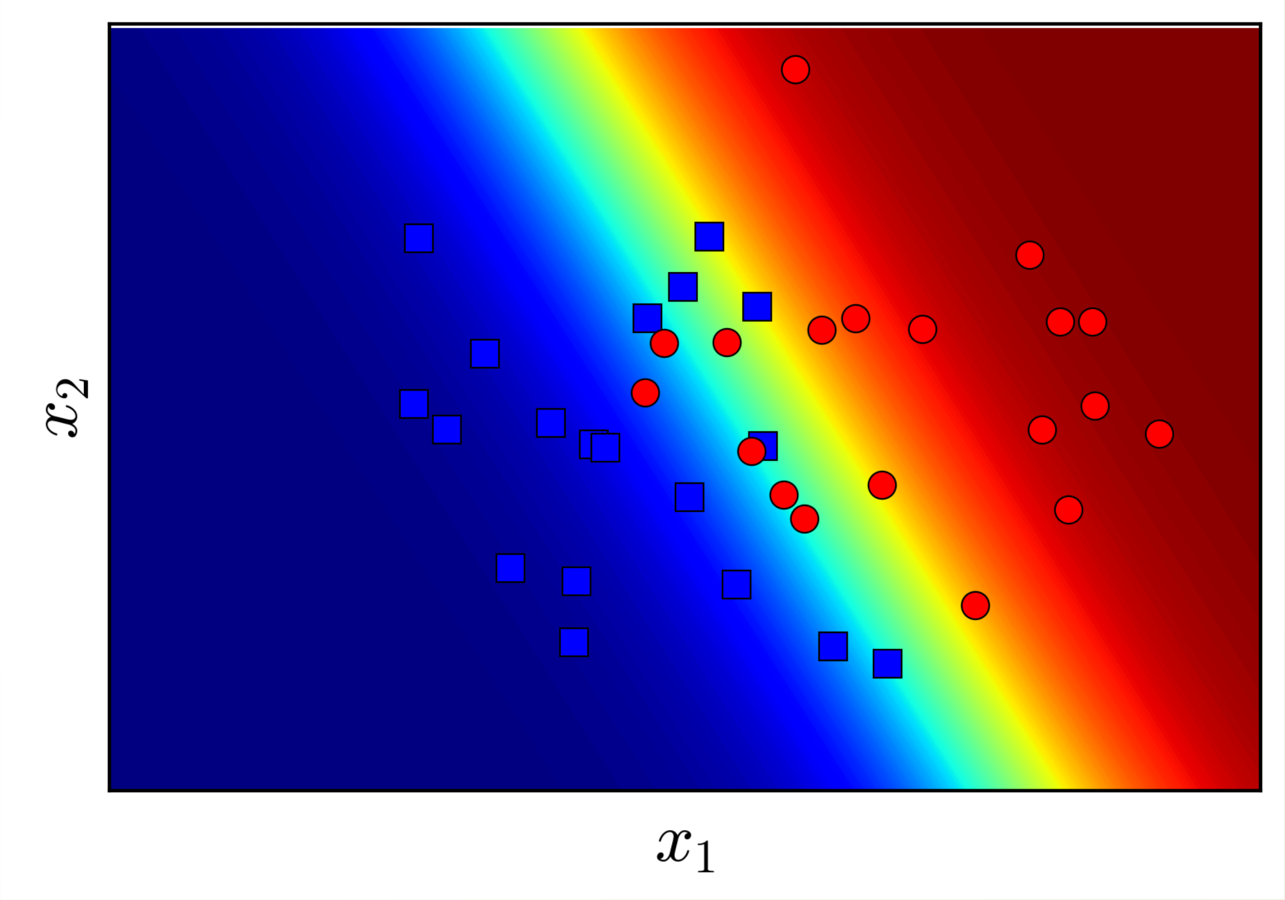
```
[[ 0.03281144  0.04694533  0.06674738  0.09407764  0.13102736  0.17961209
   0.17961209  0.24121129  0.31580406  0.40126557  0.49318368  0.58556493
   0.67229611  0.74866712  0.86263755  0.90117058  0.92977426  0.95055357
   0.96541314  0.98329067]]
```

```
X0 = X[1, np.where(y == 0)][0]
y0 = y[np.where(y == 0)]
X1 = X[1, np.where(y == 1)][0]
y1 = y[np.where(y == 1)]

plt.plot(X0, y0, 'ro', markersize = 8)
plt.plot(X1, y1, 'bs', markersize = 8)

xx = np.linspace(0, 6, 1000)
w0 = w[-1][0][0]
w1 = w[-1][1][0]
threshold = -w0/w1
yy = sigmoid(w0 + w1*xx)
plt.axis([-2, 8, -1, 2])
plt.plot(xx, yy, 'g-', linewidth = 2)
plt.plot(threshold, .5, 'y^', markersize = 8)
plt.xlabel('studying hours')
plt.ylabel('predicted probability of pass')
plt.show()
```



Logistic Regression với dữ liệu hai chiều





Một số tính chất Logistic Regression

- **Logistic regression được sử dụng nhiều cho bài toán classification**
  - Xác suất để điểm dữ liệu x thuộc lớp (class) y=1:
    $$P(y = 1|\mathbf{x}; \mathbf{w})$$
  - Xác suất để điểm dữ liệu x thuộc lớp (class) y=0:
    $$P(y = 0|\mathbf{x}; \mathbf{w})$$
  - Nếu $P(y = 1|\mathbf{x}; \mathbf{w}) > 0.5$: $\mathbf{x}$ thuộc lớp $y = 1$;
  - Nếu $P(y = 1|\mathbf{x}; \mathbf{w}) < 0.5$: $\mathbf{x}$ thuộc lớp y =0
- **Boundary tạo bởi logistic regression có dạng tuyến tính**

$$P(y = 1|\mathbf{x}; \mathbf{w}) > 0.5 \leftrightarrow \frac{1}{1 + e^{-\mathbf{w}^T\mathrm{x}}} > 0.5$$

$$e^{-\mathrm{w}^T\mathrm{x}} < 1 \leftrightarrow \mathrm{w}^T\mathrm{x} > 0$$

$$\Updownarrow$$

$$w_0x_0 + w_1x_1 + \cdots + w_mx_m > 0$$

Một số tính chất Logistic Regression

- Logistic regression không làm việc được với kiểu dữ liệu phi tuyến (non-linear data)
- Logistic regression yêu cầu các điểm dữ liệu được coi là độc lập với nhau.

Binary classification

- **Bài toán phân biệt giới tính (nam/nữ) dựa trên ảnh khuôn mặt**
- **Cơ sở dữ liệu: AR Face database**
  - 4000 ảnh RGB, 126 người (70 nam và 56 nữ)
  - Mỗi người có khoảng 26 ảnh
  - Điều kiện chiếu sáng, sắc thái biểu cảm khác nhau, che mắt hoặc miệng….
  - Ảnh được chụp tại 2 thời điểm cách nhau 2 tuần
- **Cơ sở dữ liệu: AR Face database thu gọn**
  - 2600 bức ảnh từ 50 nam và 50 nữ
  - *cropped* với kích thước 165 x 120 (pixel)

Men

Women

Binary classification

- **Tên ảnh: G-xxx-yy.bmp**
  - G: M(man)/W(woman)
  - xxx: ID (001-005)
  - yy: số thứ tự của ảnh cho mỗi ID (01-26)
- **Training: 25 nam và 25 nữ**
- **Test: 25 nam và 25 nữ**
- **Chỉ lấy các khuôn mặt không bị che bởi kính hoặc khăn**

Binary classification

Khai báo thư viện

```
import numpy as np
from sklearn import linear_model           # for logistic regression
from sklearn.metrics import accuracy_score # for evaluation
from scipy import misc                     # for loading image
np.random.seed(1)                          # for fixing random values
```

Phân chia tập train và test

```
path = '../data/AR/' # path to the database
train_ids = np.arange(1, 26)
test_ids = np.arange(26, 50)
view_ids = np.hstack((np.arange(1, 8), np.arange(14, 21)))
```

Tạo random projection matrix

```
D = 165*120 # original dimension
d = 500 # new dimension

# generate the projection matrix
ProjectionMatrix = np.random.randn(D, d)
```

Binary classification

Xây dựng danh sách các tên files

```
def build_list_fn(pre, img_ids, view_ids):
    """
    INPUT:
        pre = 'M-' or 'W-'
        img_ids: indexes of images
        view_ids: indexes of views
    OUTPUT:
        a list of filenames
    """
    list_fn = []
    for im_id in img_ids:
        for v_id in view_ids:
            fn = path + pre + str(im_id).zfill(3) + '-' + \
                str(v_id).zfill(2) + '.bmp'
            list_fn.append(fn)
    return list_fn
```

Binary classification

```
def rgb2gray(rgb):
#     Y' = 0.299 R + 0.587 G + 0.114 B
    return rgb[:,:,0]*.299 + rgb[:, :, 1]*.587 + rgb[:, :, 2]*.114

# feature extraction
def vectorize_img(filename):
    # load image
    rgb = misc.imread(filename)
    # convert to gray scale
    gray = rgb2gray(rgb)
    # vectorization each row is a data point
    im_vec = gray.reshape(1, D)
    return im_vec
```

```
def build_data_matrix(img_ids, view_ids):
    total_imgs = img_ids.shape[0]*view_ids.shape[0]*2

    X_full = np.zeros((total_imgs, D))
    y = np.hstack((np.zeros((total_imgs/2, )), np.ones((total_imgs/2, ))))

    list_fn_m = build_list_fn('M-', img_ids, view_ids)
    list_fn_w = build_list_fn('W-', img_ids, view_ids)
    list_fn = list_fn_m + list_fn_w

    for i in range(len(list_fn)):
        X_full[i, :] = vectorize_img(list_fn[i])

    X = np.dot(X_full, ProjectionMatrix)
    return (X, y)

(X_train_full, y_train) = build_data_matrix(train_ids, view_ids)
x_mean = X_train_full.mean(axis = 0)
x_var  = X_train_full.var(axis = 0)
```

Binary classification

```
(X_train_full, y_train) = build_data_matrix(train_ids, view_ids)
x_mean = X_train_full.mean(axis = 0)
x_var  = X_train_full.var(axis = 0)

def feature_extraction(X):
    return (X - x_mean)/x_var

X_train = feature_extraction(X_train_full)
X_train_full = None ## free this variable

(X_test_full, y_test) = build_data_matrix(test_ids, view_ids)
X_test = feature_extraction(X_test_full)
X_test_full = None
```

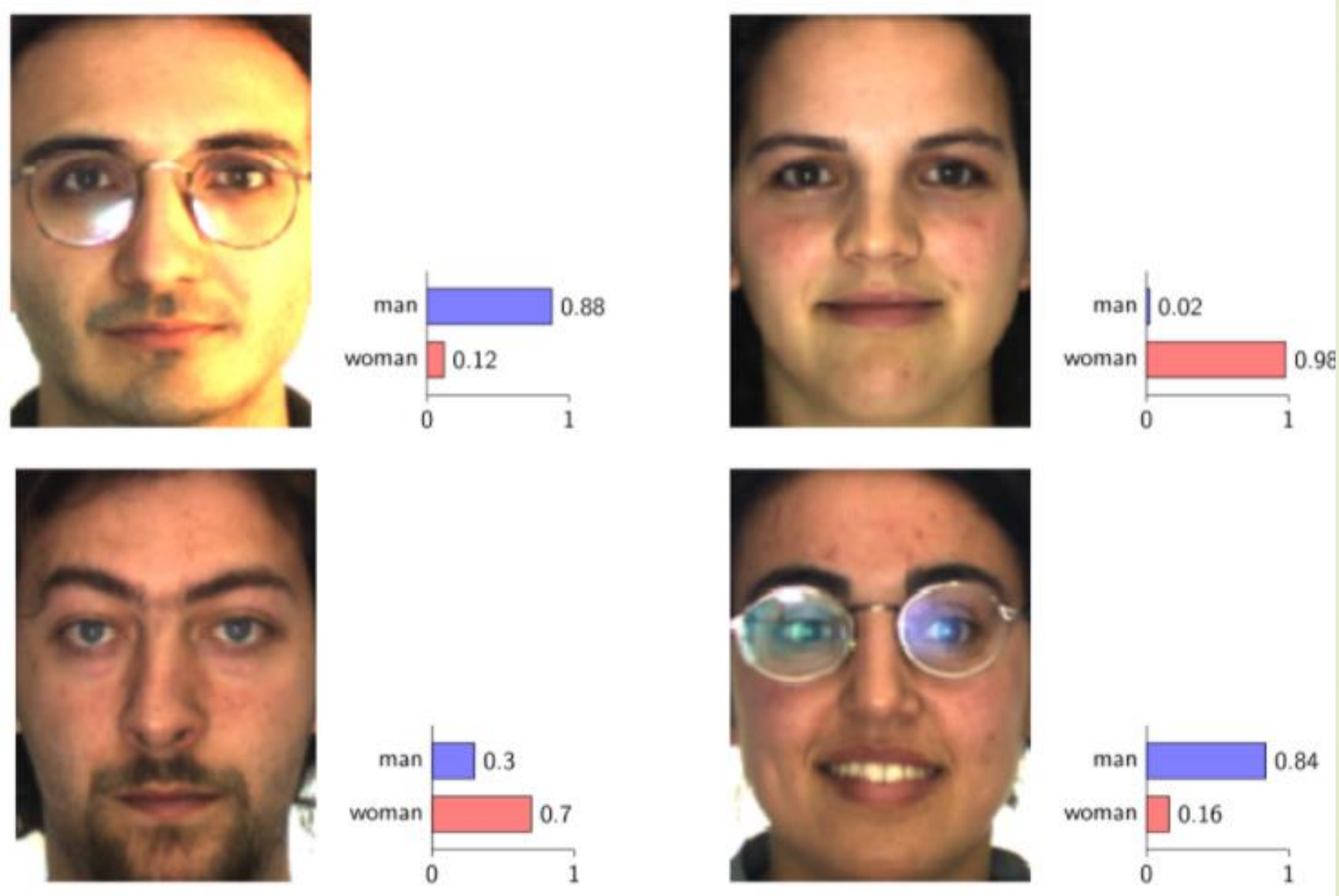
```
logreg = linear_model.LogisticRegression(C=1e5) # just a big number
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
print "Accuracy: %.2f %%" %(100*accuracy_score(y_test, y_pred))
```

```
Accuracy: 90.33 %
```

Binary classification

```
def feature_extraction_fn(fn):
    """
    extract feature from filename
    """
    # vectorize
    im = vectorize_img(fn)
    # project
    im1 = np.dot(im, ProjectionMatrix)
    # standardization
    return feature_extraction(im1)

fn1 = path + 'M-036-18.bmp'
fn2 = path + 'W-045-01.bmp'
fn3 = path + 'M-048-01.bmp'
fn4 = path + 'W-027-02.bmp'

x1 = feature_extraction_fn(fn1)
p1 = logreg.predict_proba(x1)
print(p1)

x2 = feature_extraction_fn(fn2)
p2 = logreg.predict_proba(x2)
print(p2)

x3 = feature_extraction_fn(fn3)
p3 = logreg.predict_proba(x3)
print(p3)

x4 = feature_extraction_fn(fn4)
p4 = logreg.predict_proba(x4)
print(p4)
```

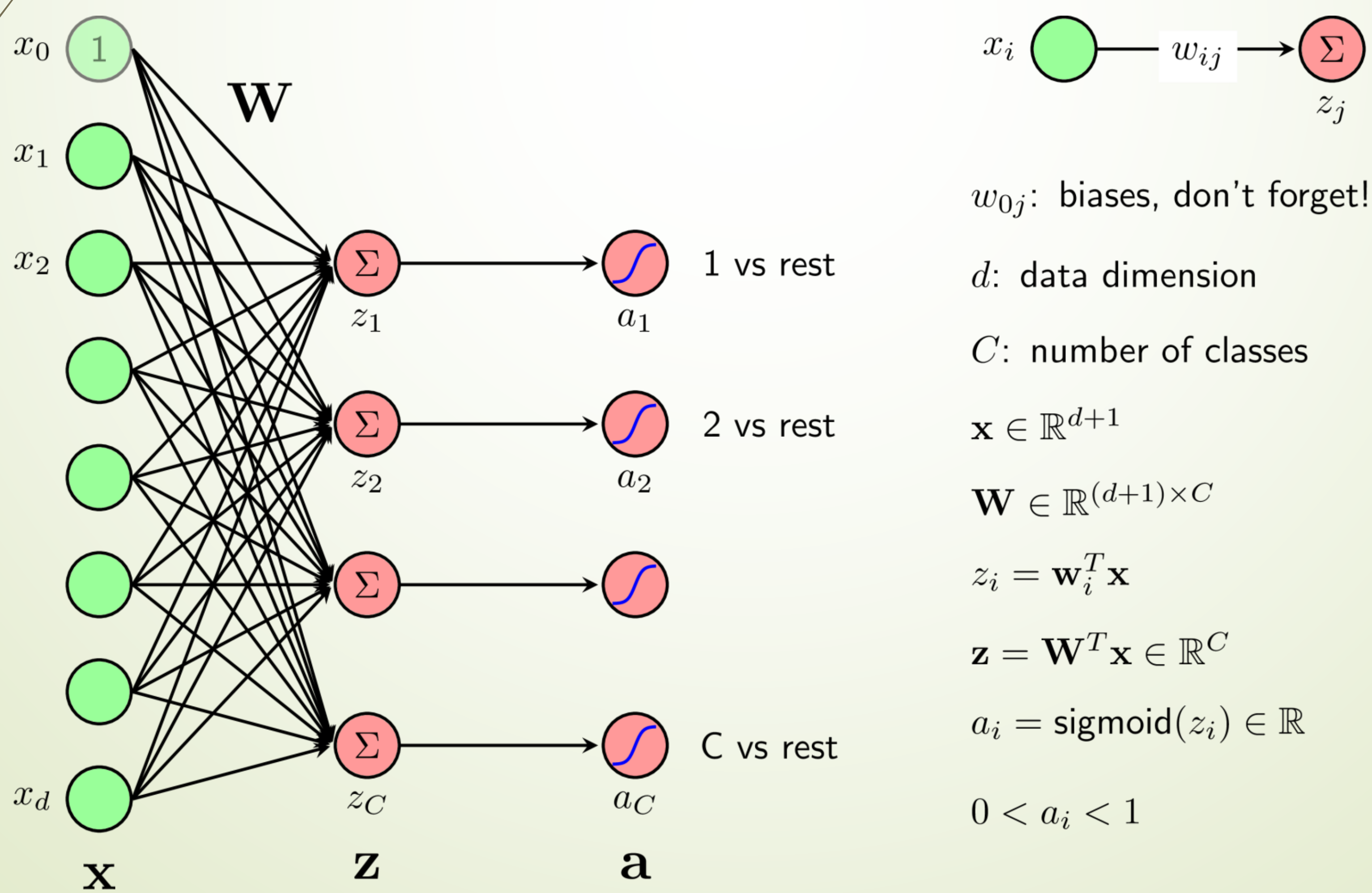
Binary classification



Softmax Regression

- Các bài toán phân lớp trong thực tế thường có nhiều lớp
- Softmax regression là một phương pháp mở rộng của Logistic regression
- Nếu số lớp là C→ cần xây dựng C logistic regression:

$$a_i = sigmoid(z_i) = sigmoid(\mathbf{w}_i^T \mathbf{x})$$



w_{0j}: biases, don't forget!

d: data dimension

C: number of classes

$\mathbf{x} \in \mathbb{R}^{d+1}$

$\mathbf{W} \in \mathbb{R}^{(d+1)\times C}$

$z_i = \mathbf{w}_i^T \mathbf{x}$

$\mathbf{z} = \mathbf{W}^T \mathbf{x} \in \mathbb{R}^C$

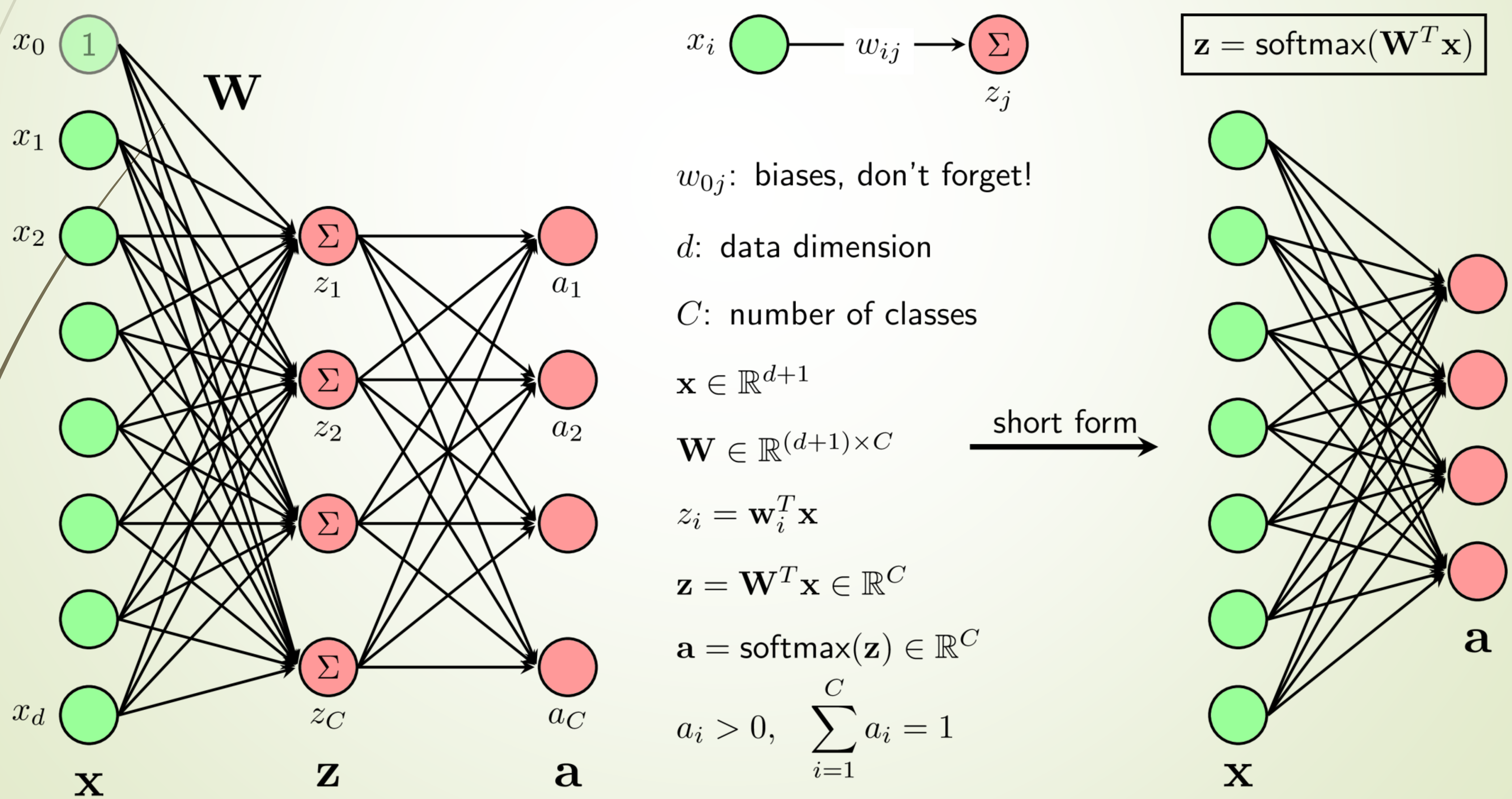
$a_i = \text{sigmoid}(z_i) \in \mathbb{R}$

$0 < a_i < 1$

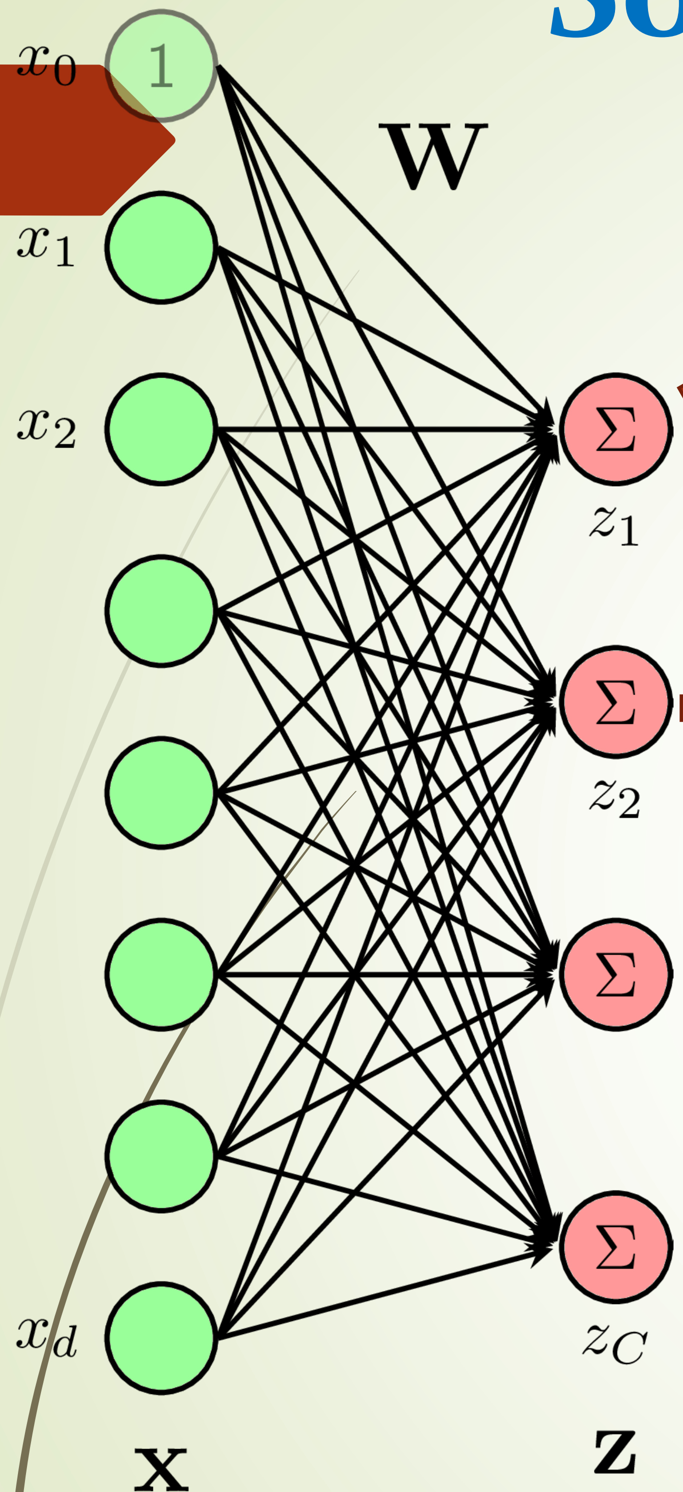
Softmax Regression

- Softmax Regression:

$$P(y_k = i|\mathrm{x}_k; \mathbf{W}) = \frac{\exp(\mathrm{w}_i^T \mathbf{x}_k)}{\sum_{j=1}^{C} \exp(\mathbf{w}_j^T \mathbf{x}_k)}, \qquad \forall i = 1, 2, \dots, C$$



Softmax Regression



$$\mathbf{w}_1 = \begin{bmatrix} w_{01} \\ w_{11} \\ w_{21} \\ \vdots \\ w_{d1} \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} w_{02} \\ w_{12} \\ w_{22} \\ \vdots \\ w_{d2} \end{bmatrix} \quad \mathbf{w}_C = \begin{bmatrix} w_{0C} \\ w_{1C} \\ w_{2C} \\ \vdots \\ w_{dC} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

$$z_1 = \mathbf{w}_1^T\mathbf{x}$$

$$z_C = \mathbf{w}_C^T\mathbf{x}$$

$$z_2 = \mathbf{w}_2^T\mathbf{x}$$

$$e^Z = \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ e^{z_3} \\ \vdots \\ e^{z_C} \end{bmatrix}$$

$$\mathrm{W} = \begin{bmatrix} \boxed{\begin{matrix} w_{01} \\ w_{11} \\ w_{21} \\ \vdots \\ w_{d1} \end{matrix}} & \begin{matrix} w_{02} \\ w_{12} \\ w_{22} \\ \vdots \\ w_{d2} \end{matrix} & \cdots & \begin{matrix} w_{0C} \\ w_{1C} \\ w_{2C} \\ \vdots \\ w_{dC} \end{matrix} \end{bmatrix} \Rightarrow \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_C \end{bmatrix}$$

$$P(y = 1|\mathrm{x};\mathrm{W}) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_C}}$$

$$P(y = 2|\mathrm{x};\mathrm{W}) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + \cdots + e^{z_C}}$$

$$P(y = C|\mathrm{x};\mathrm{W}) = \frac{e^{z_C}}{e^{z_1} + e^{z_2} + \cdots + e^{z_C}}$$

Softmax Regression

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \quad \forall i = 1, 2, \dots, C$$

```
import numpy as np

def softmax(Z):
    """
    Compute softmax values for each sets of scores in V.
    each column of V is a set of score.
    """
    e_Z = np.exp(Z)
    A = e_Z / e_Z.sum(axis = 0)
    return A
```

$$\frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} = \frac{\exp(-c)\exp(z_i)}{\exp(-c)\sum_{j=1}^{C} \exp(z_j)} = \frac{\exp(z_i - c)}{\sum_{j=1}^{C} \exp(z_j - c)}$$

Phiên bản ổn định hơn của Softmax Regression

```
def softmax_stable(Z):
    """
    Compute softmax values for each sets of scores in Z.
    each column of Z is a set of score.
    """
    e_Z = np.exp(Z - np.max(Z, axis = 0, keepdims = True))
    A = e_Z / e_Z.sum(axis = 0)
    return A
```

Softmax Regression

- Hàm mất mát (loss function)
  - Đầu ra mong muốn: one hot coding

$$y_i = [y_{1i}\ y_{2i}\ y_{3i} \dots y_{Ci}]$$

Trong đó, C: số lớp (classes);

$y_{ji} = 1$ nếu dữ liệu đầu vào thuộc lớp thứ j

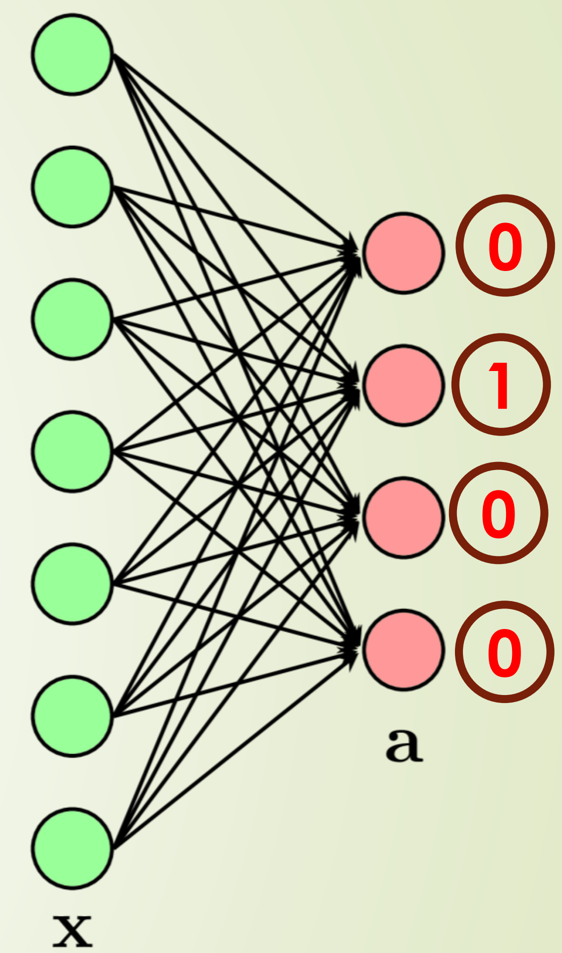
(các phần tử còn lại bằng 0)

- Cross entropy

$$J(\mathbf{W}; \mathbf{x}_i, \mathbf{y}_i) = -\sum_{j=1}^{C} y_{ji} \log(a_{ji})$$

$$J(\mathbf{W}; \mathbf{X}, \mathbf{Y}) = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} \log(a_{ji}) = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} \log\left(\frac{\exp(\mathbf{w}_j^T\mathbf{x}_i)}{\sum_{k=1}^{C}\exp(\mathbf{w}_k^T\mathbf{x}_i)}\right)$$

$$\mathrm{a} = softmax(\mathrm{W}^T\mathrm{x})$$



Softmax Regression

- Tối ưu hàm mất mát

$$J_i(\mathbf{W}) \triangleq J(\mathbf{W}; \mathbf{x}_i, \mathbf{y}_i) =$$

$$= -\sum_{j=1}^{C} y_{ji} \log\left(\frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^{C} \exp(\mathbf{w}_k^T \mathbf{x}_i)}\right) = -\sum_{j=1}^{C}\left(y_{ji}\mathbf{w}_j^T \mathbf{x}_i - y_{ji}\log\left(\sum_{k=1}^{C}\exp(\mathbf{w}_k^T \mathbf{x}_i)\right)\right)$$

$$J_i(\mathbf{W}) = -\sum_{j=1}^{C} y_{ji}\mathbf{w}_j^T \mathbf{x}_i + \log\left(\sum_{k=1}^{C}\exp(\mathbf{w}_k^T \mathbf{x}_i)\right)$$

$$\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}} = \left[\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{w}_1}, \frac{\partial J_i(\mathbf{W})}{\partial \mathbf{w}_2}, \dots, \frac{\partial J_i(\mathbf{W})}{\partial \mathbf{w}_C}\right]$$

Softmax Regression

- Tối ưu hàm mất mát

$$\boxed{\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{w}_j} = -y_{ji}\mathbf{x}_i + \frac{\exp(\mathbf{w}_j^T\mathbf{x}_i)}{\sum_{k=1}^C \exp(\mathbf{w}_k^T\mathbf{x}_i)}\mathbf{x}_i = -y_{ji}\mathbf{x}_i + a_{ji}\mathbf{x}_i = \mathbf{x}_i(a_{ji} - y_{ji}) = e_{ji}\mathbf{x}_i}$$

$$\boxed{\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}} = \mathbf{x}_i[e_{1i}, e_{2i}, \dots, e_{Ci}] = \mathbf{x}_i\mathbf{e}_i^T}$$

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

$$\boxed{\mathbf{E} = \mathbf{A} - \mathbf{Y}}$$

Softmax Regression

```python
import numpy as np

# randomly generate data
N = 2 # number of training sample
d = 2 # data dimension
C = 3 # number of classes

X = np.random.randn(d, N)
y = np.random.randint(0, 3, (N,))
```

```python
## One-hot coding
from scipy import sparse
def convert_labels(y, C = C):
    """
    convert 1d label to a matrix label: each column of this
    matrix coresponding to 1 element in y. In i-th column of Y,
    only one non-zeros element located in the y[i]-th position,
    and = 1 ex: y = [0, 2, 1, 0], and 3 classes then return

            [[1, 0, 0, 1],
             [0, 0, 1, 0],
             [0, 1, 0, 0]]
    """
    Y = sparse.coo_matrix((np.ones_like(y),
        (y, np.arange(len(y)))), shape = (C, len(y))).toarray()
    return Y

Y = convert_labels(y, C)
```

Softmax Regression

```
# cost or loss function
def cost(X, Y, W):
    A = softmax(W.T.dot(X))
    return -np.sum(Y*np.log(A))

W_init = np.random.randn(d, C)

def grad(X, Y, W):
    A = softmax((W.T.dot(X)))
    E = A - Y
    return X.dot(E.T)

def numerical_grad(X, Y, W, cost):
    eps = 1e-6
    g = np.zeros_like(W)
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            W_p = W.copy()
            W_n = W.copy()
            W_p[i, j] += eps
            W_n[i, j] -= eps
            g[i,j] = (cost(X, Y, W_p) - cost(X, Y, W_n))/(2*eps)
    return g

g1 = grad(X, Y, W_init)
g2 = numerical_grad(X, Y, W_init, cost)

print(np.linalg.norm(g1 - g2))
```

Softmax Regression

```
def softmax_regression(X, y, W_init, eta, tol = 1e-4, max_count = 10000):
    W = [W_init]
    C = W_init.shape[1]
    Y = convert_labels(y, C)
    it = 0
    N = X.shape[1]
    d = X.shape[0]

    count = 0
    check_w_after = 20
    while count < max_count:
        # mix data
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[:, i].reshape(d, 1)
            yi = Y[:, i].reshape(C, 1)
            ai = softmax(np.dot(W[-1].T, xi))
            W_new = W[-1] + eta*xi.dot((yi - ai).T)
            count += 1
            # stopping criteria
            if count%check_w_after == 0:
                if np.linalg.norm(W_new - W[-check_w_after]) < tol:
                    return W
            W.append(W_new)
    return W
eta = .05
d = X.shape[0]
W_init = np.random.randn(d, C)

W = softmax_regression(X, y, W_init, eta)
# W[-1] is the solution, W is all history of weights
```