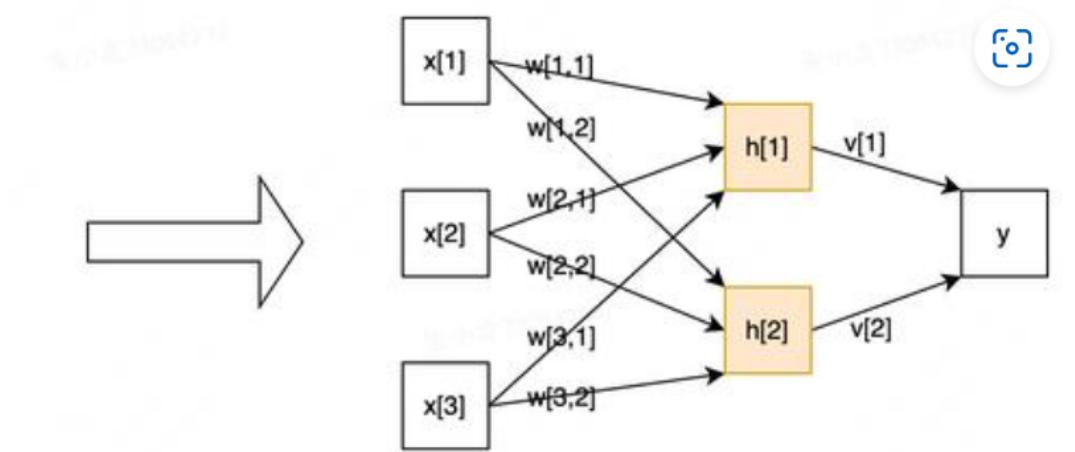
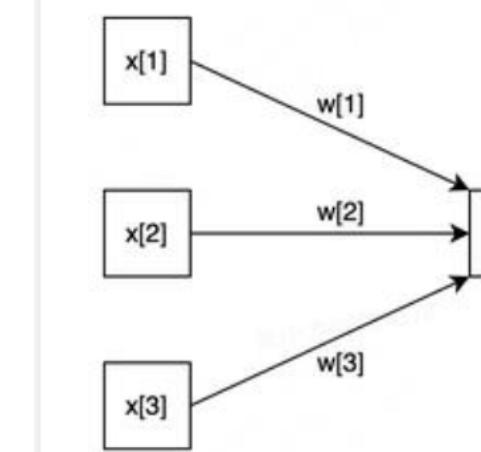
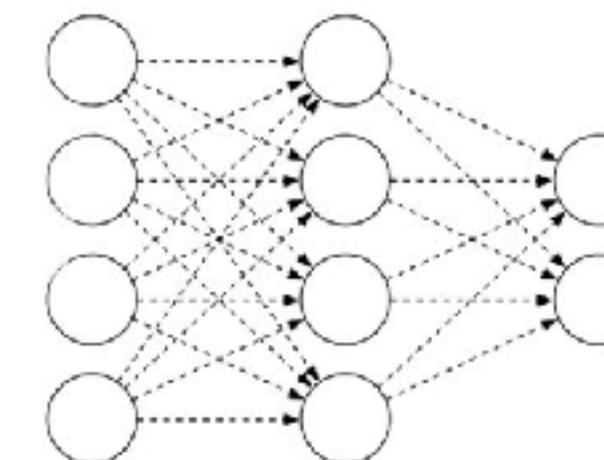


COMP0169: Machine Learning for Visual Computing

Linear Classifier and MultiLayer Perceptron



Lectures will be Recorded

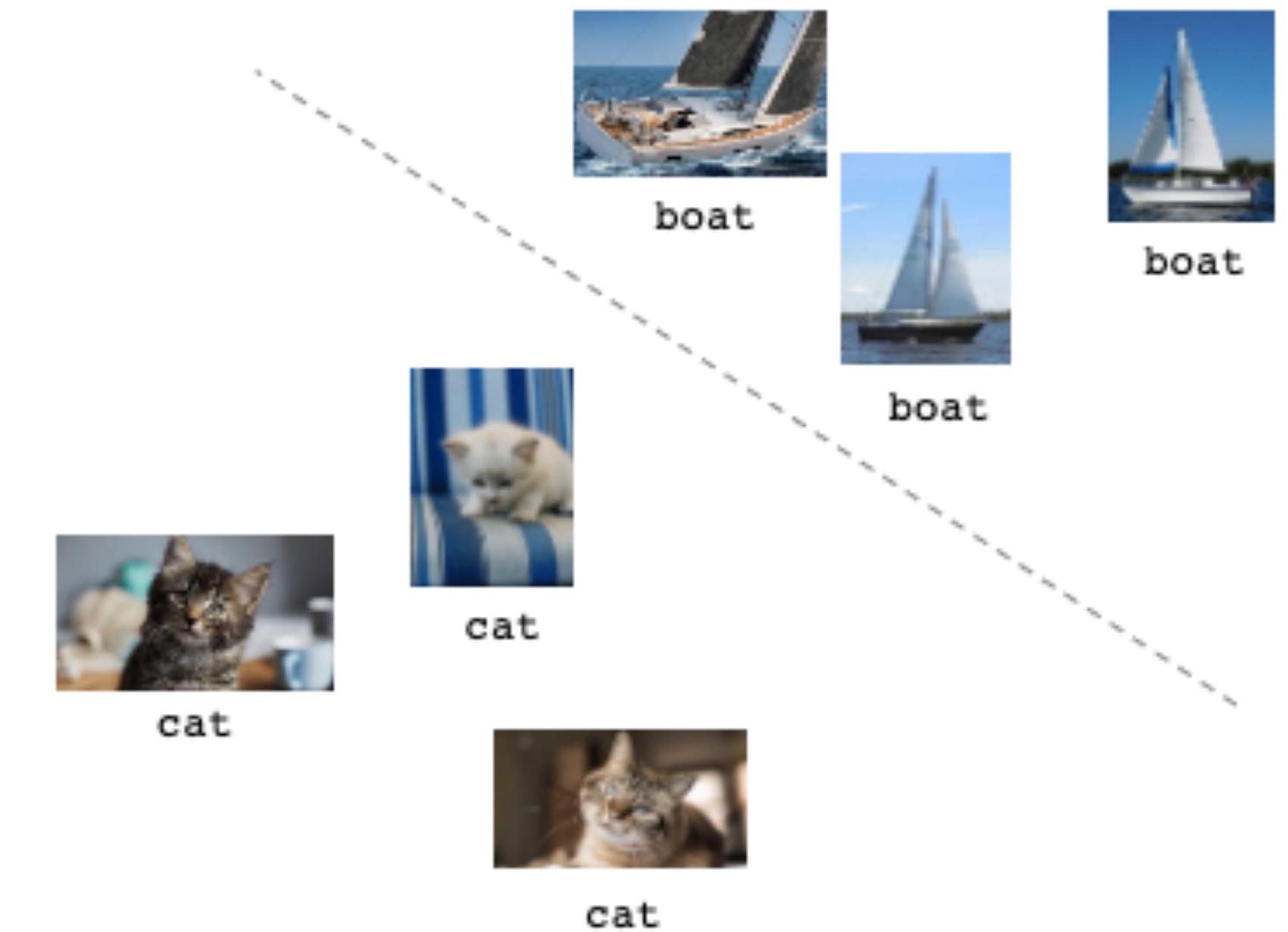
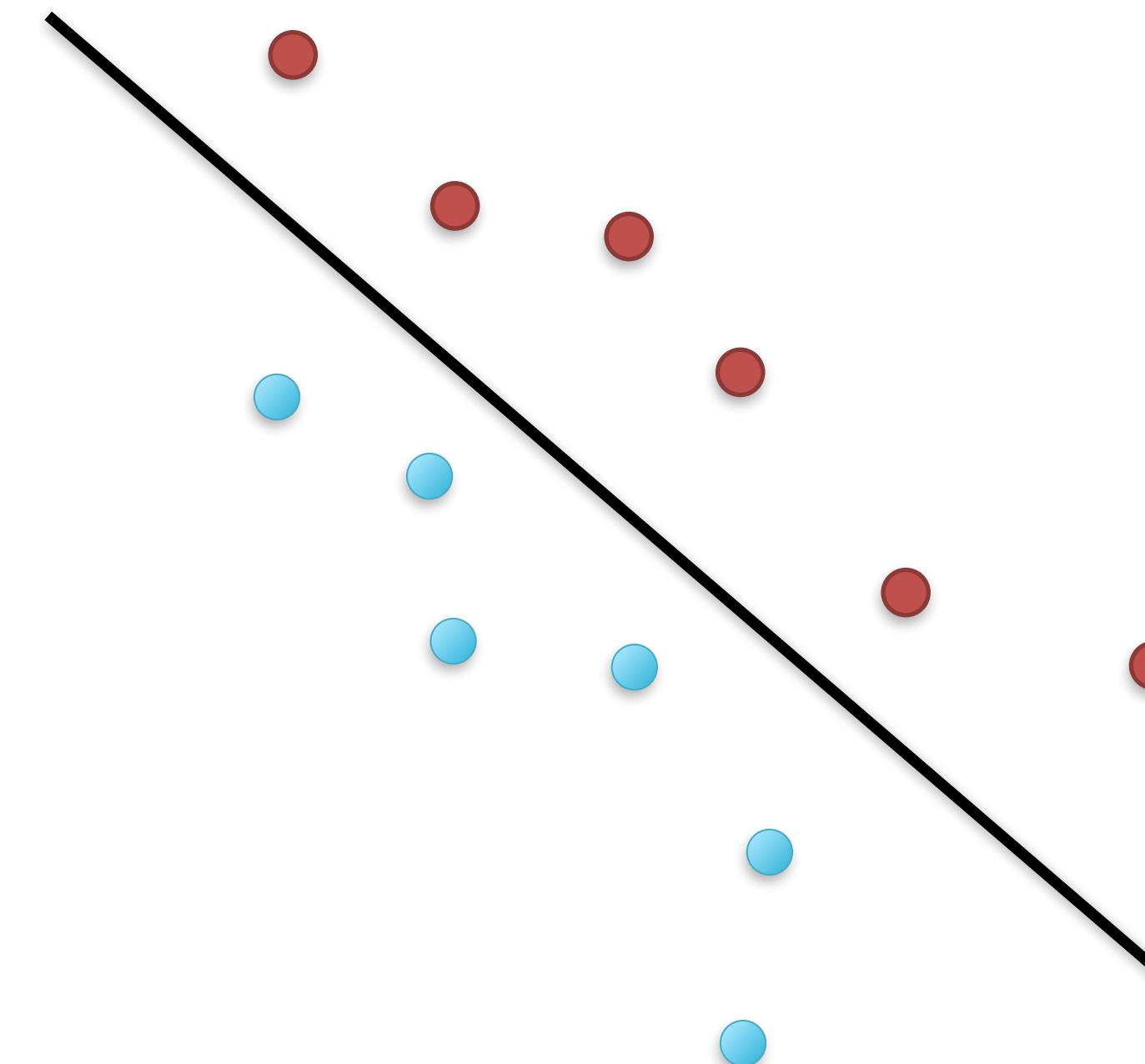
CW1 submission date

Recap

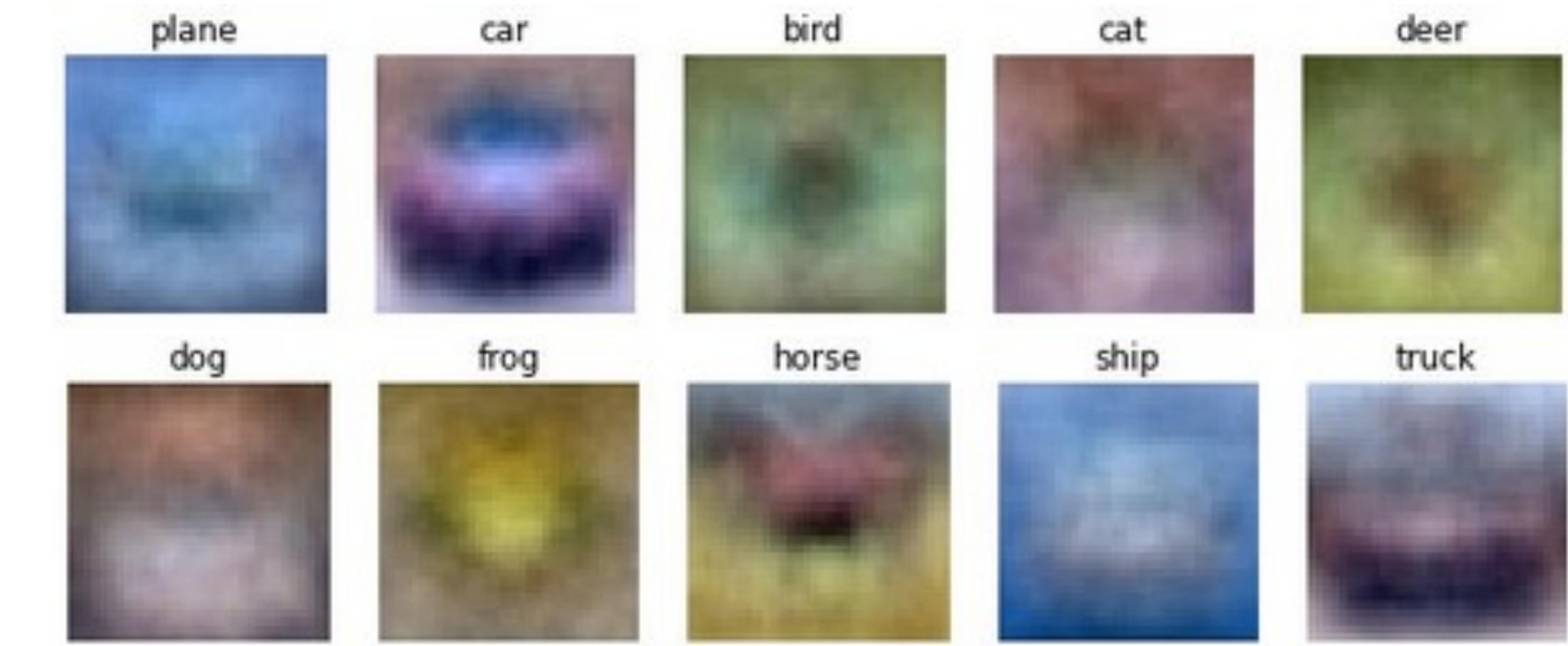
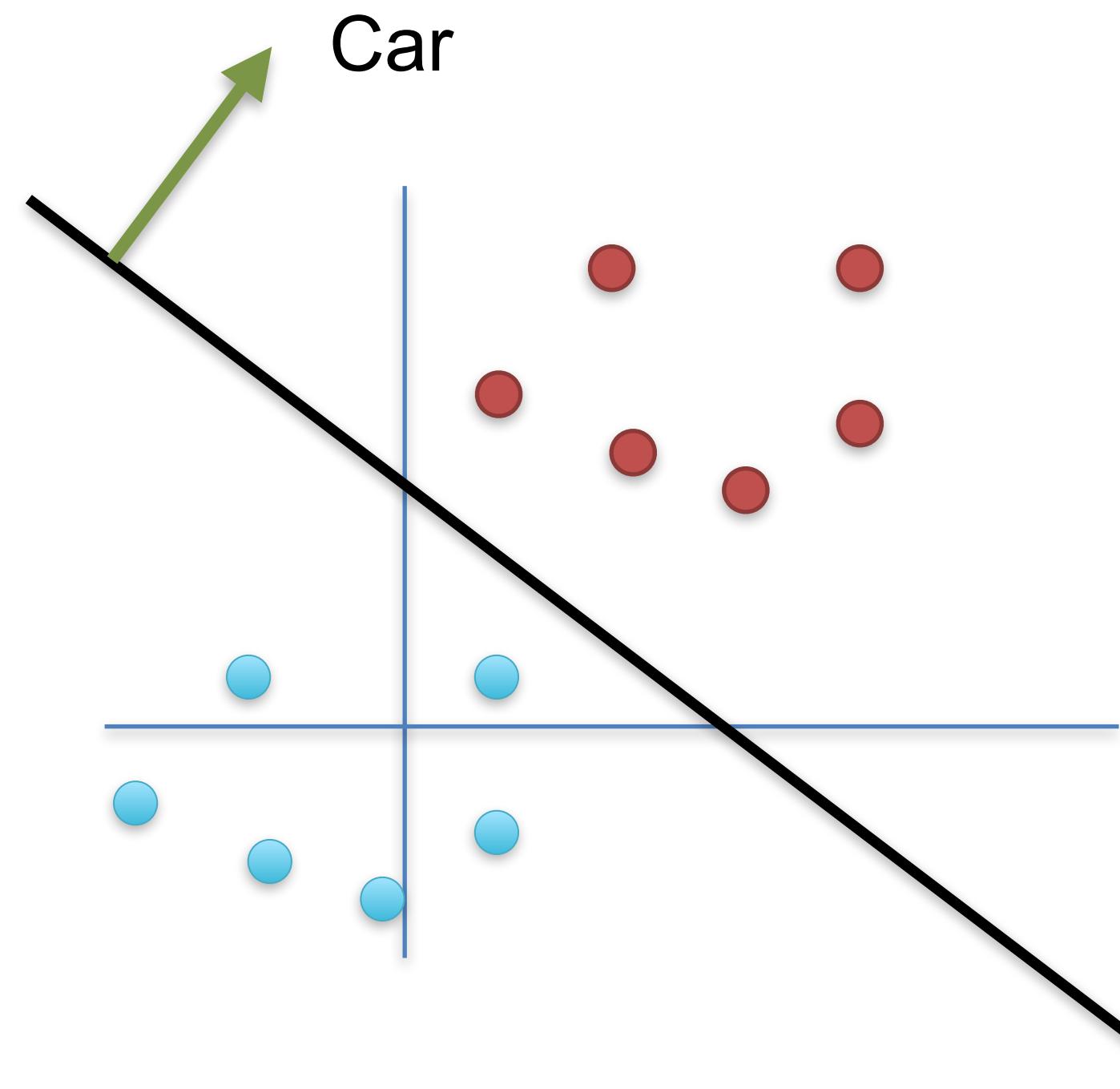
- PCA
- Bias Trick
- Normal Equation
- Extension to polynomial fit

$$\begin{aligned}y &= w_0 + w_1 x_1 + w_2 x_2 \dots \\&= \langle w_0, w_1, \dots \rangle \cdot \langle 1, x_1, x_2, \dots \rangle \\&= w^\top x\end{aligned}$$

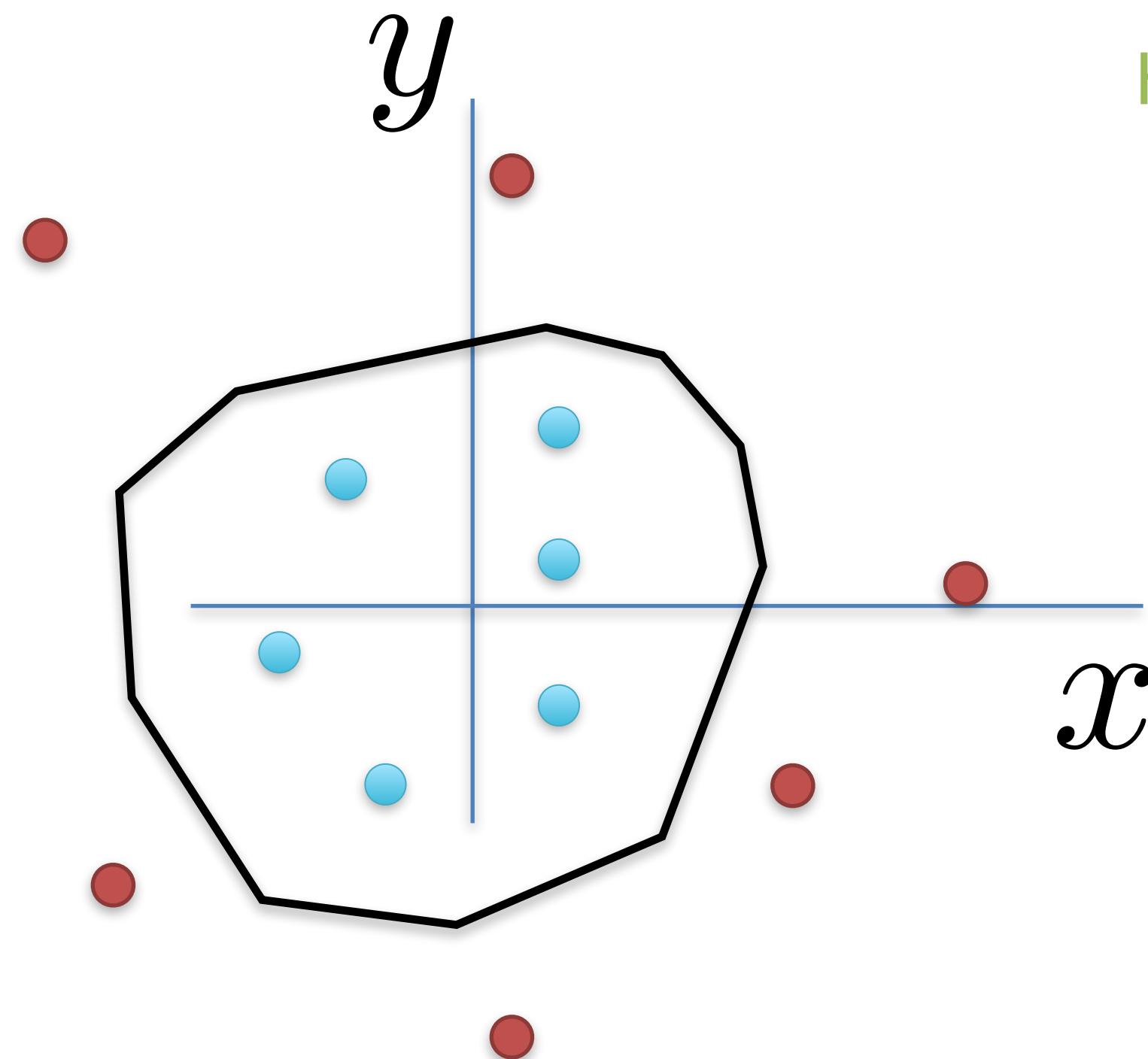
Simple Classifier



Classifier Geometric Viewpoint



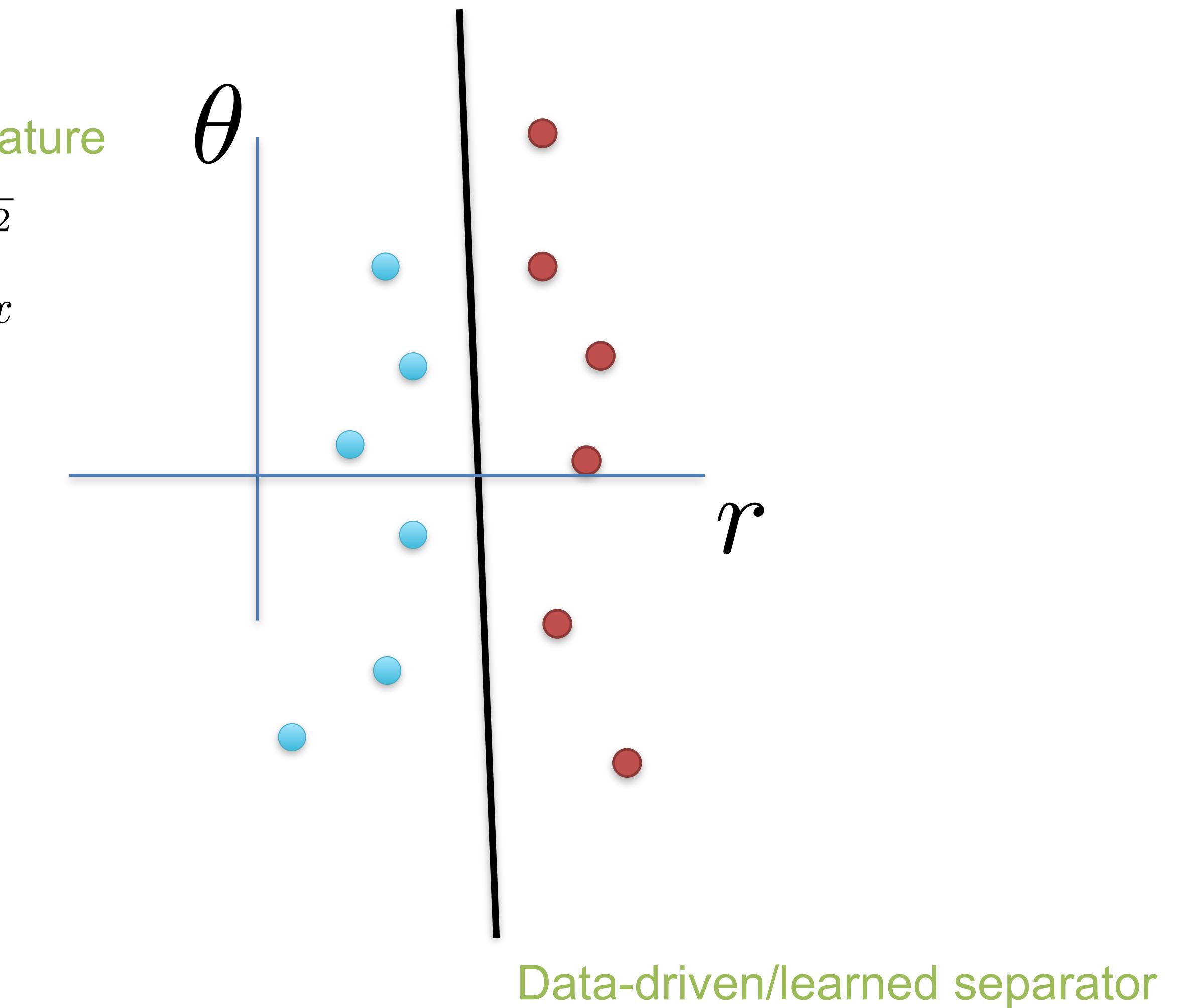
Classifier Geometric Viewpoint



Hand-coded feature

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1} y/x$$

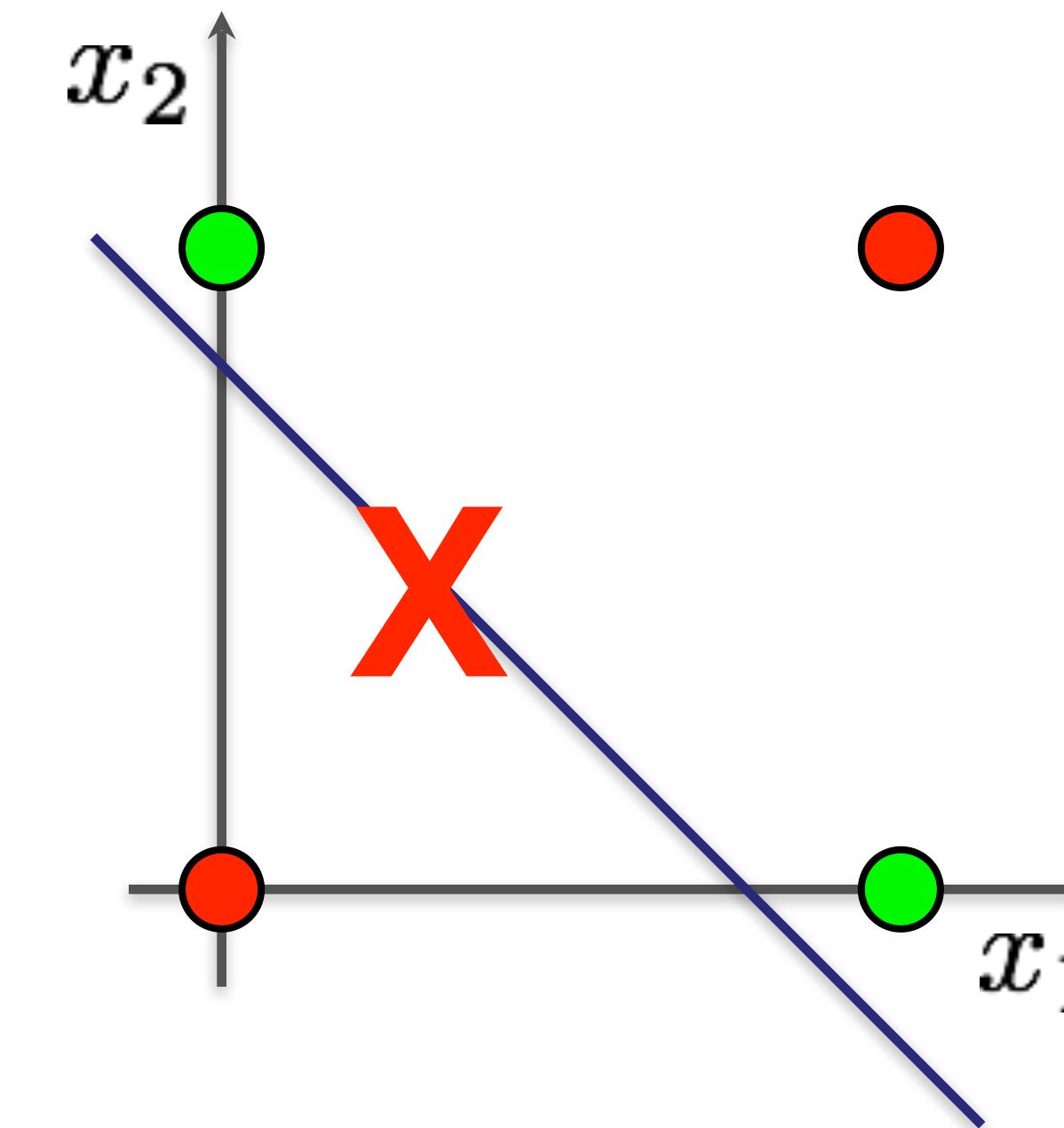


Data-driven/learned separator

XOR Problem

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = f(x_1, x_2)$$



← no line
can do it.

$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2) = \mathcal{H}(\mathbf{w}^T \mathbf{x})$$

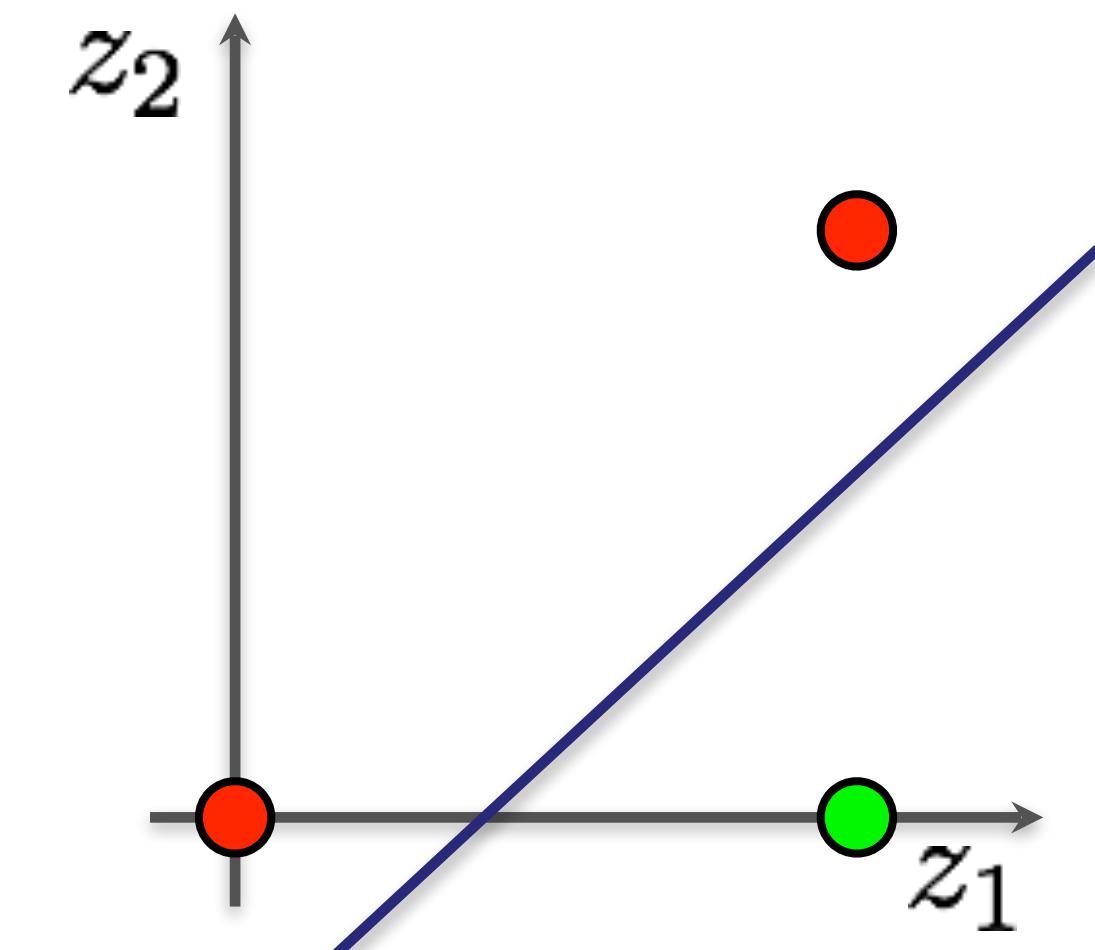
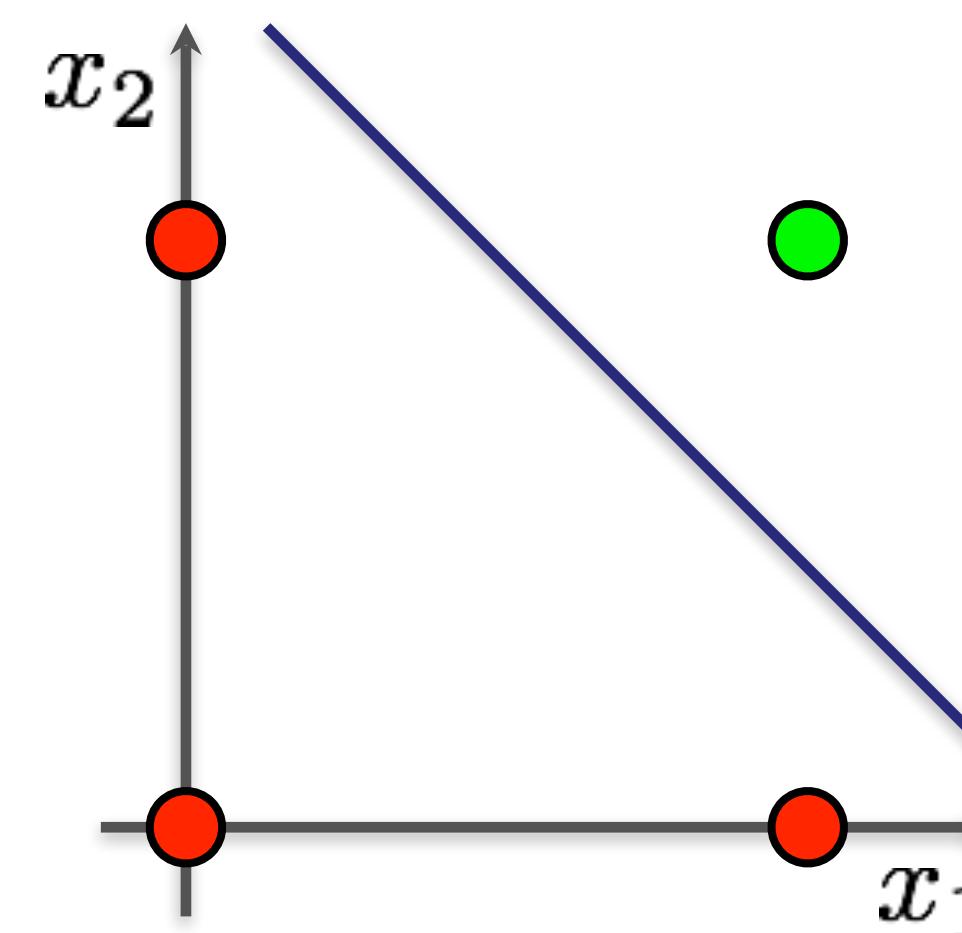
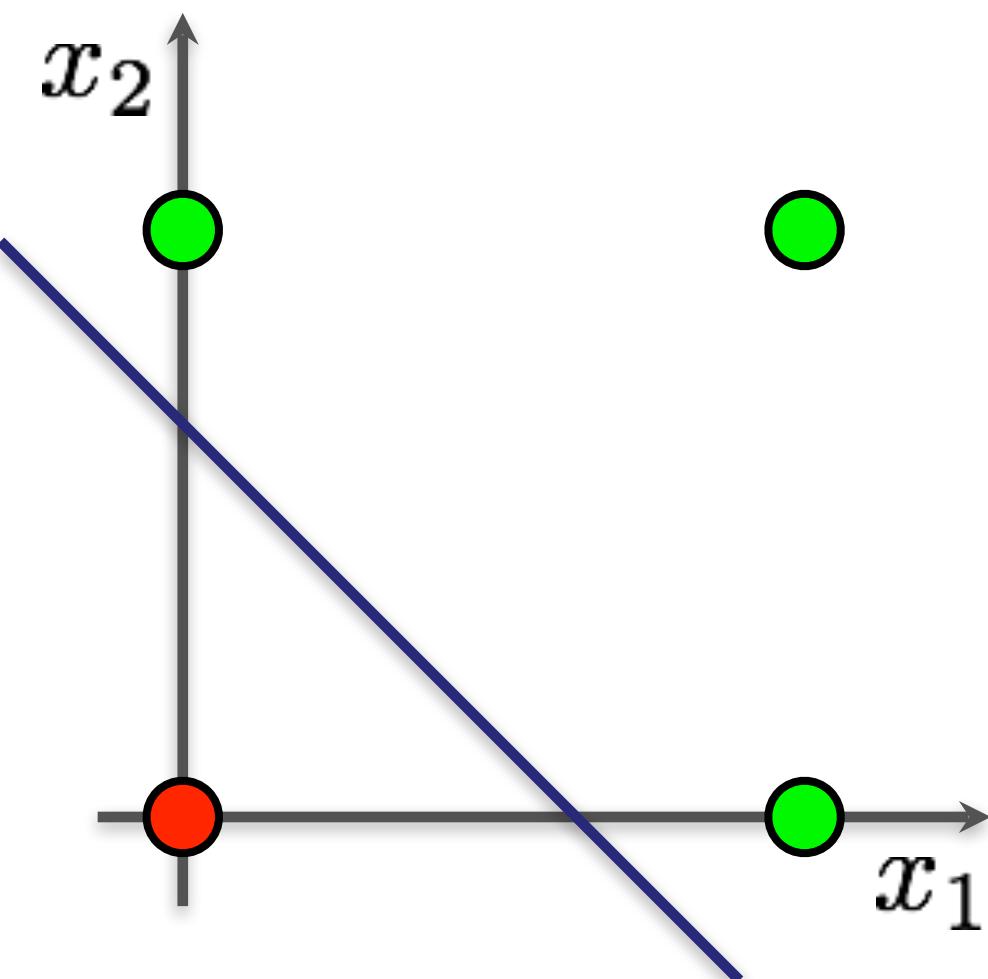
XOR Problem

x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1

手續
去
來
一
切
測

x_1	x_2	z_2
0	0	0
0	1	0
1	0	0
1	1	1

z_1	z_2	y
0	0	0
1	0	1
1	0	1
1	1	0



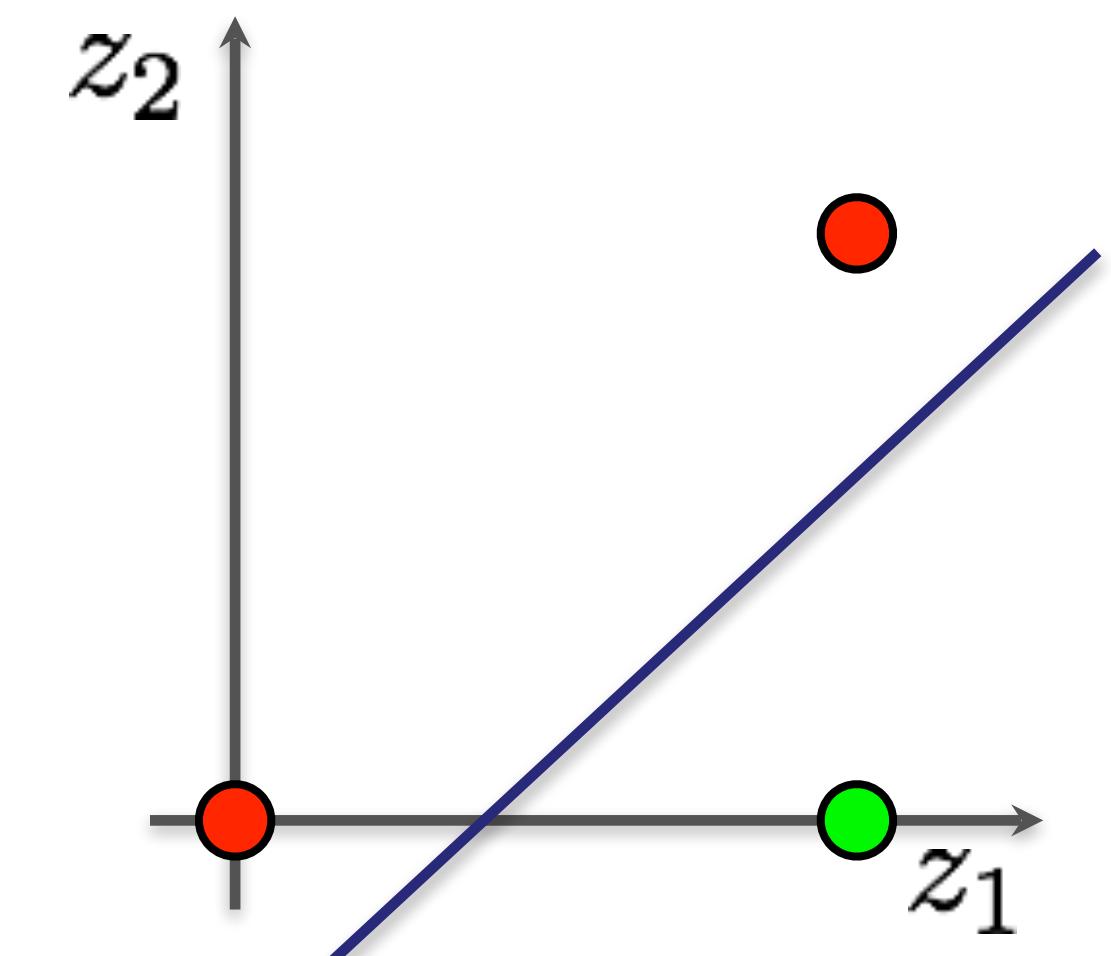
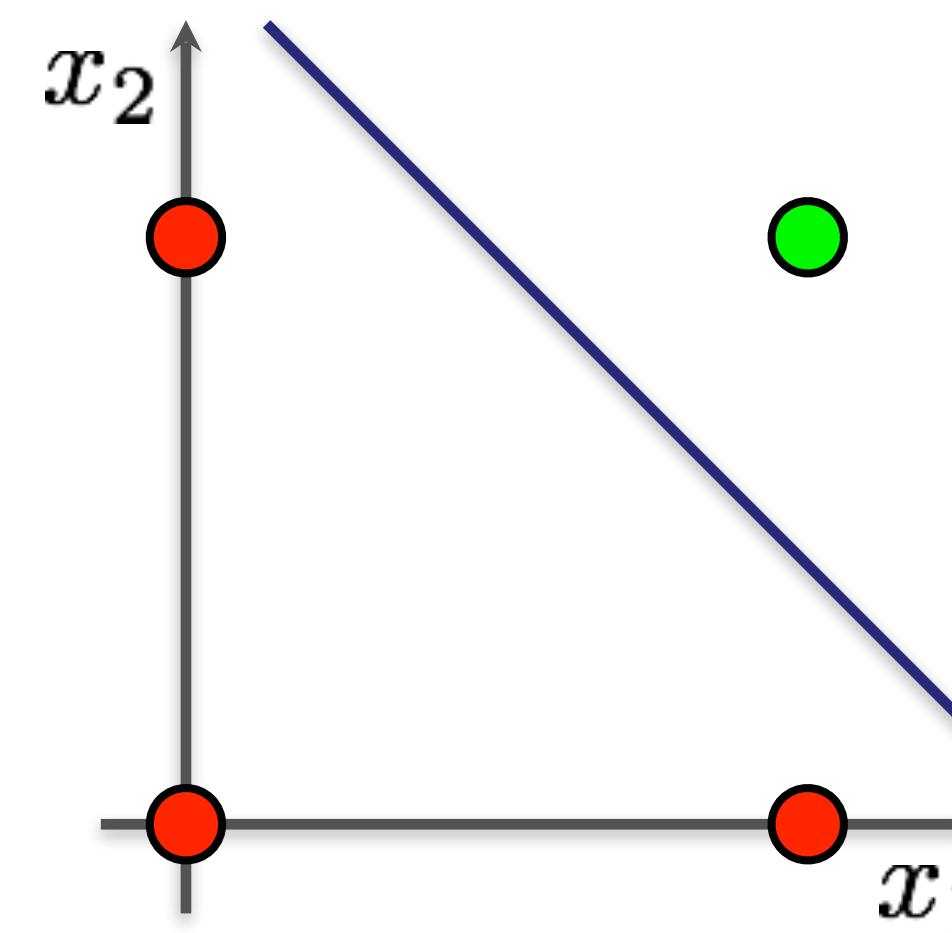
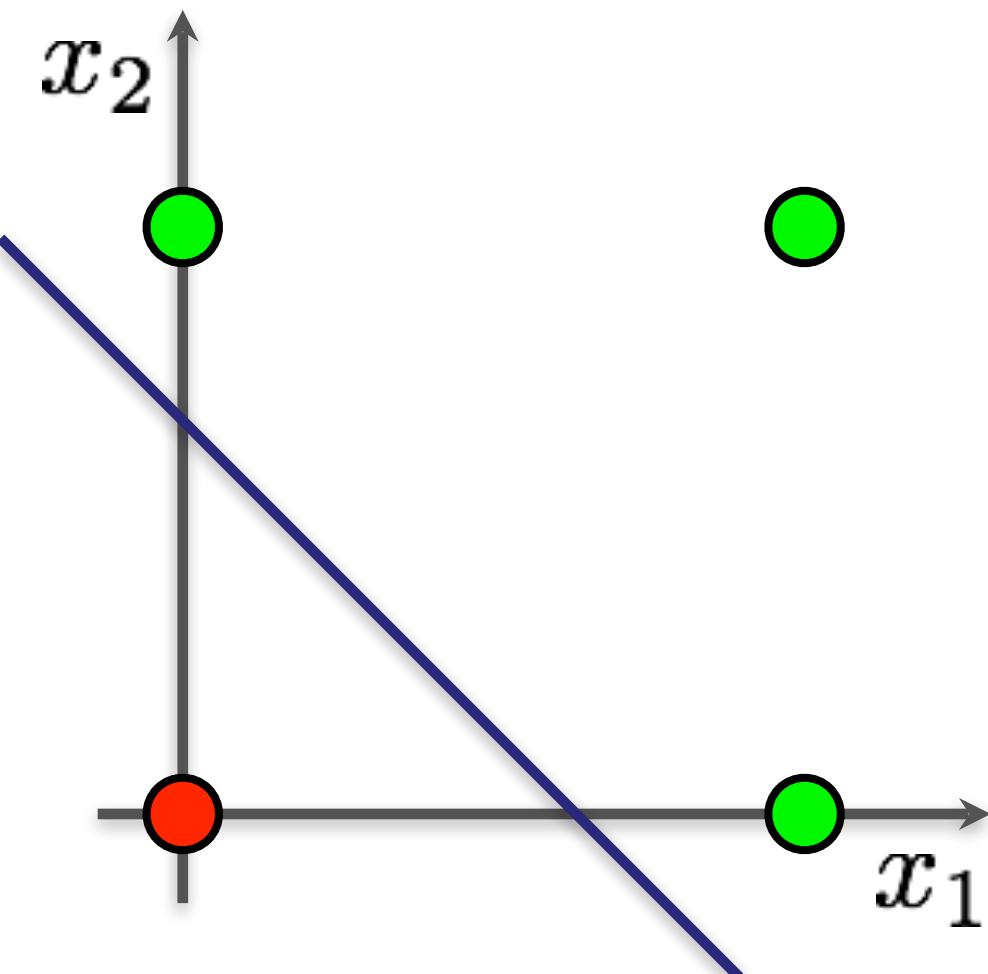
XOR Problem

$$y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2))$$

$$= f(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2))$$

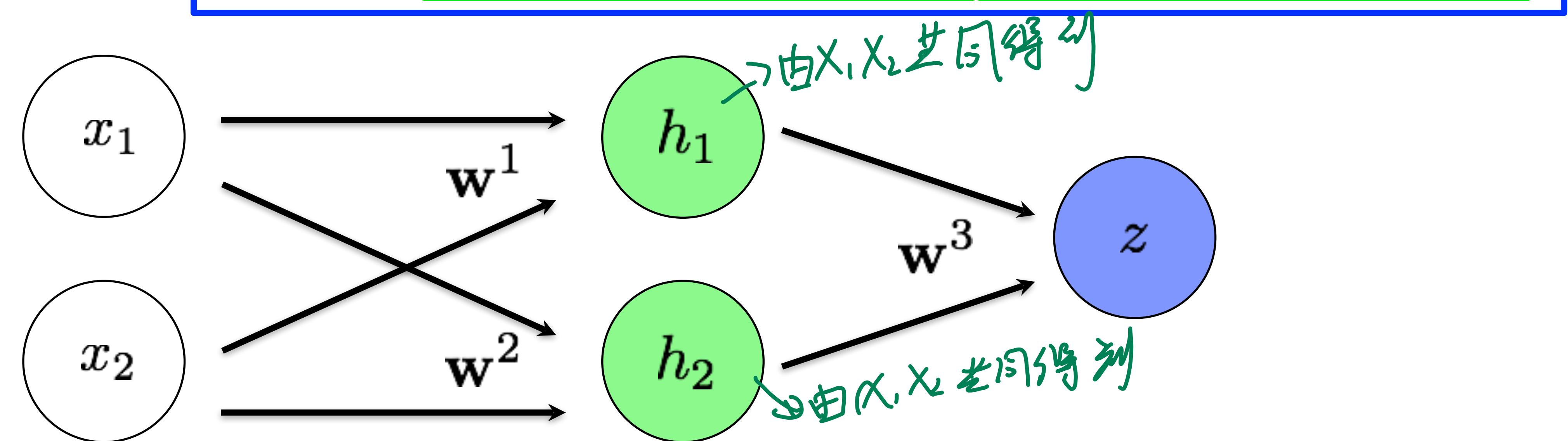
$$= \mathcal{H}(\mathbf{w}^3, \boxed{\mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2))}, \boxed{\mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2))})$$

$\xrightarrow{z_1}$ $\xrightarrow{z_2}$

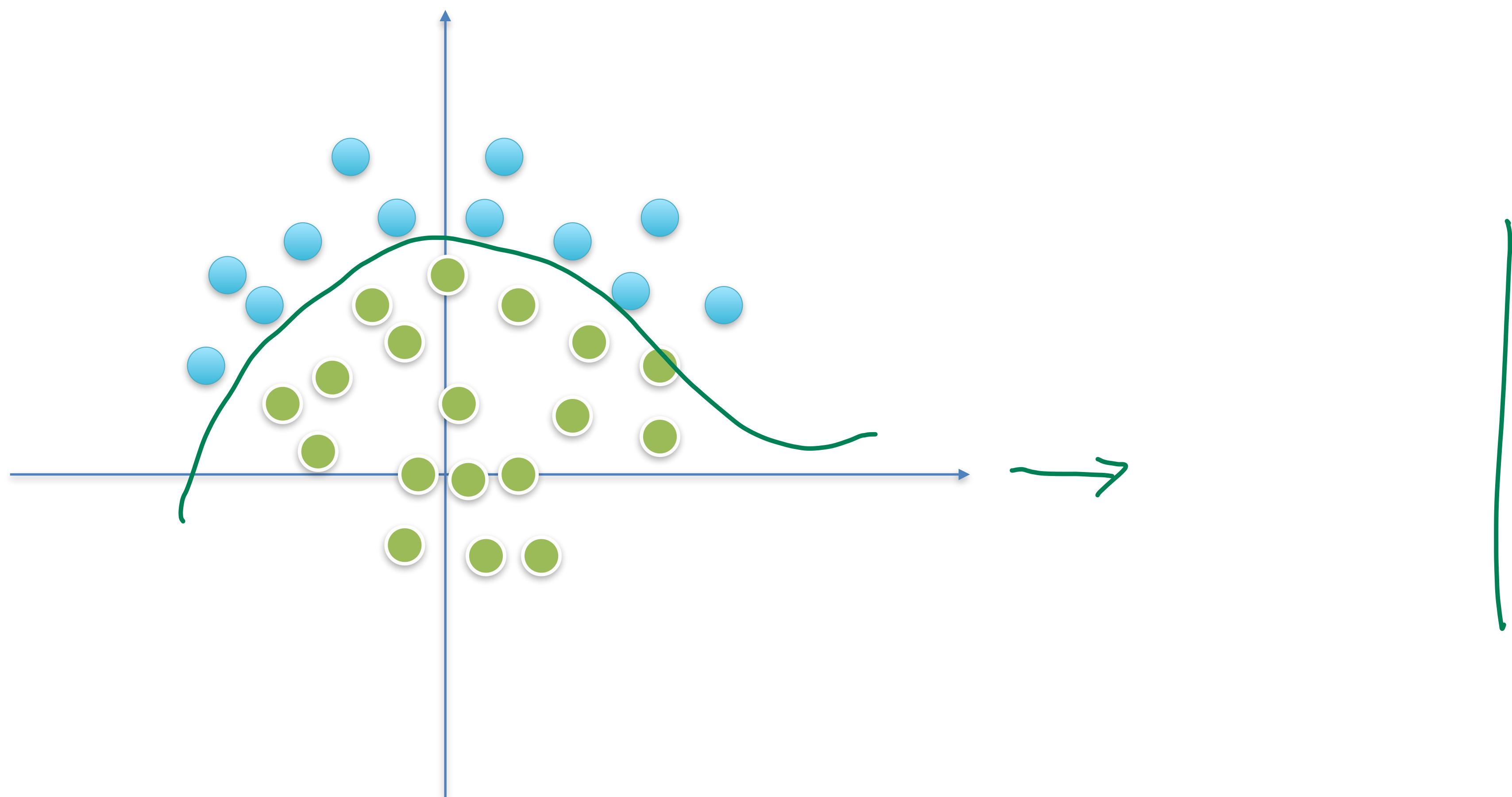


XOR Problem

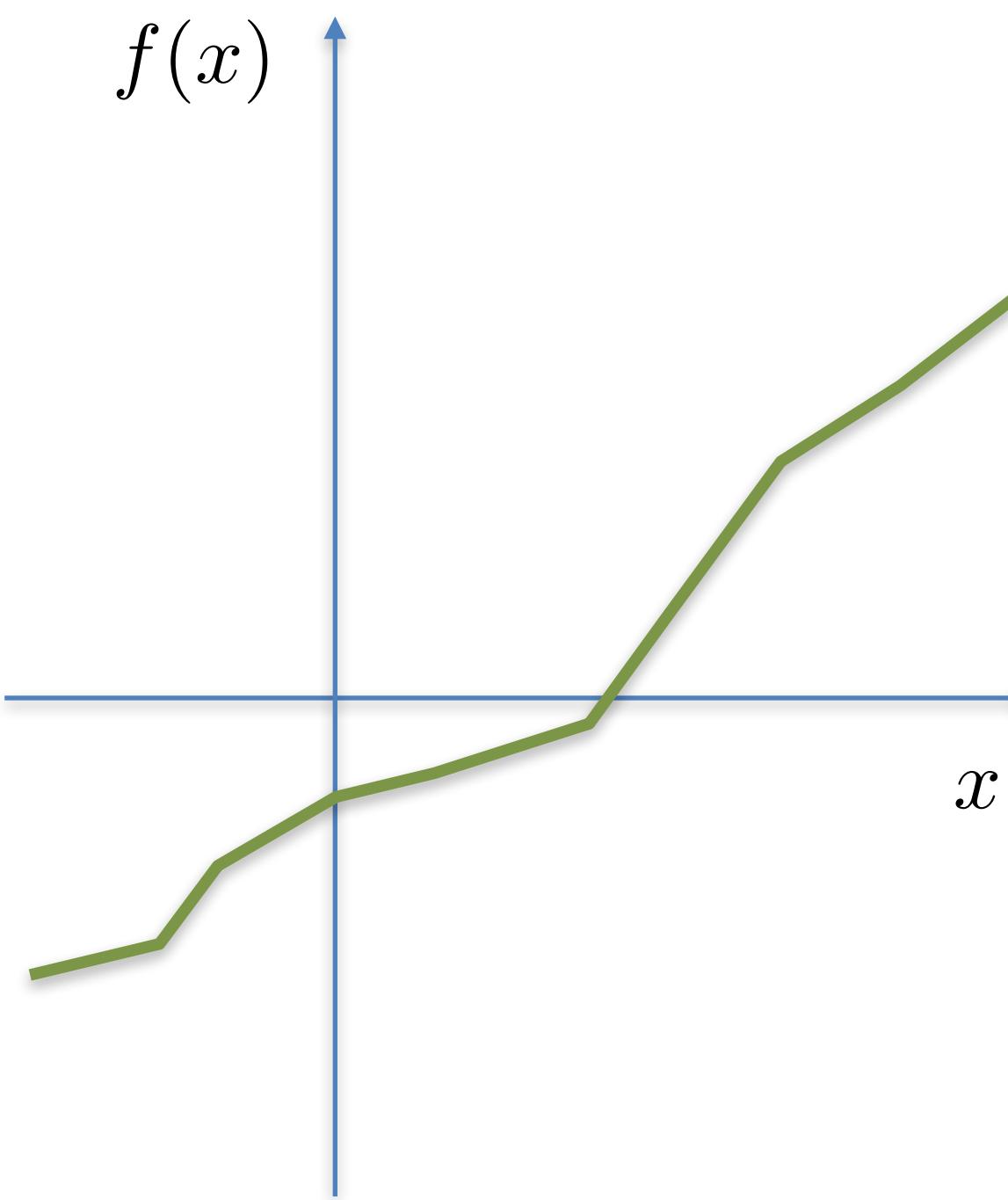
$$\begin{aligned}y &= f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \\&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\&= \mathcal{H}(\mathbf{w}^3, \boxed{\mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2))}, \boxed{\mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2))})\end{aligned}$$



What are Neural Networks Doing?

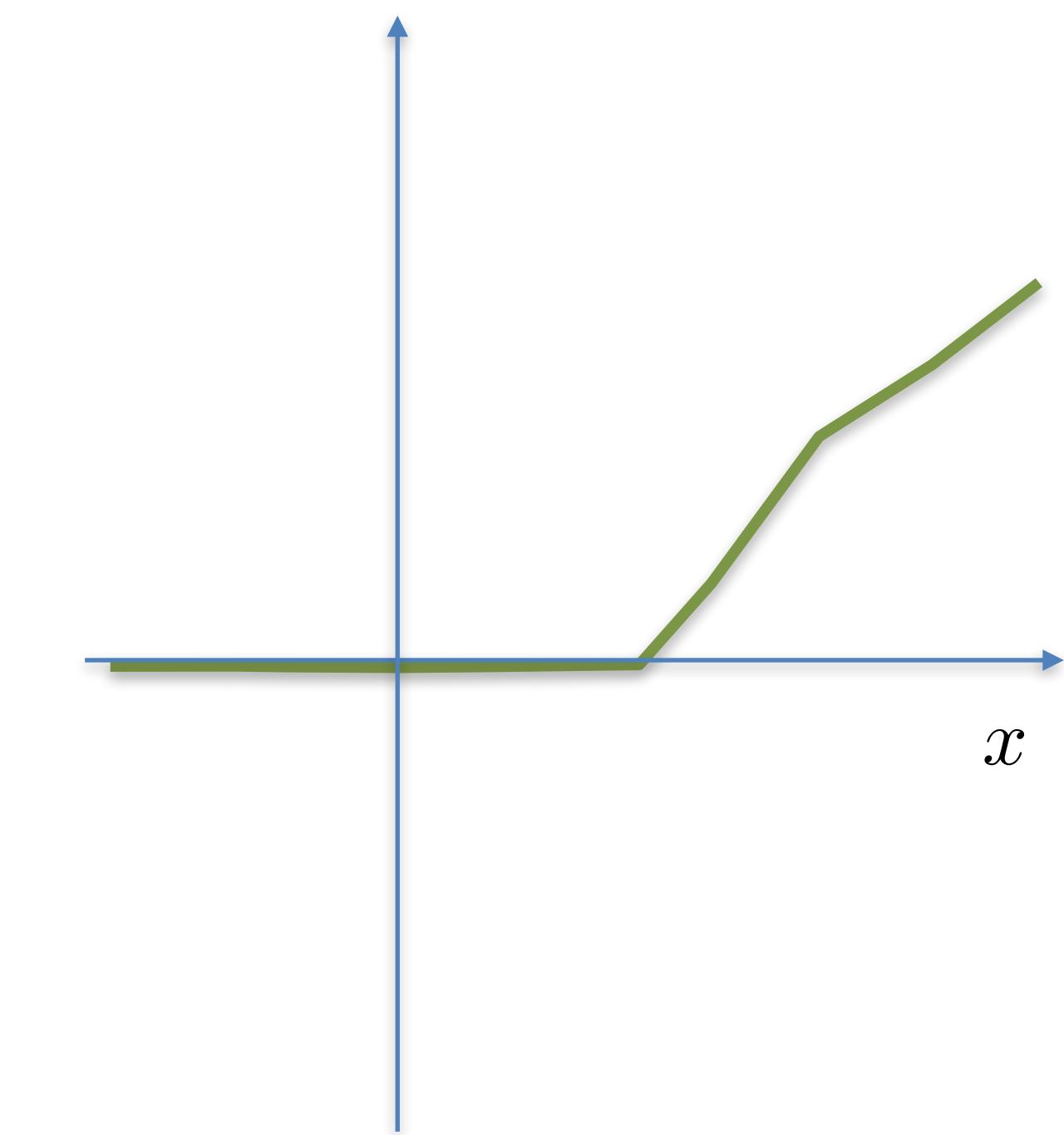


Importance of Non-Linearities



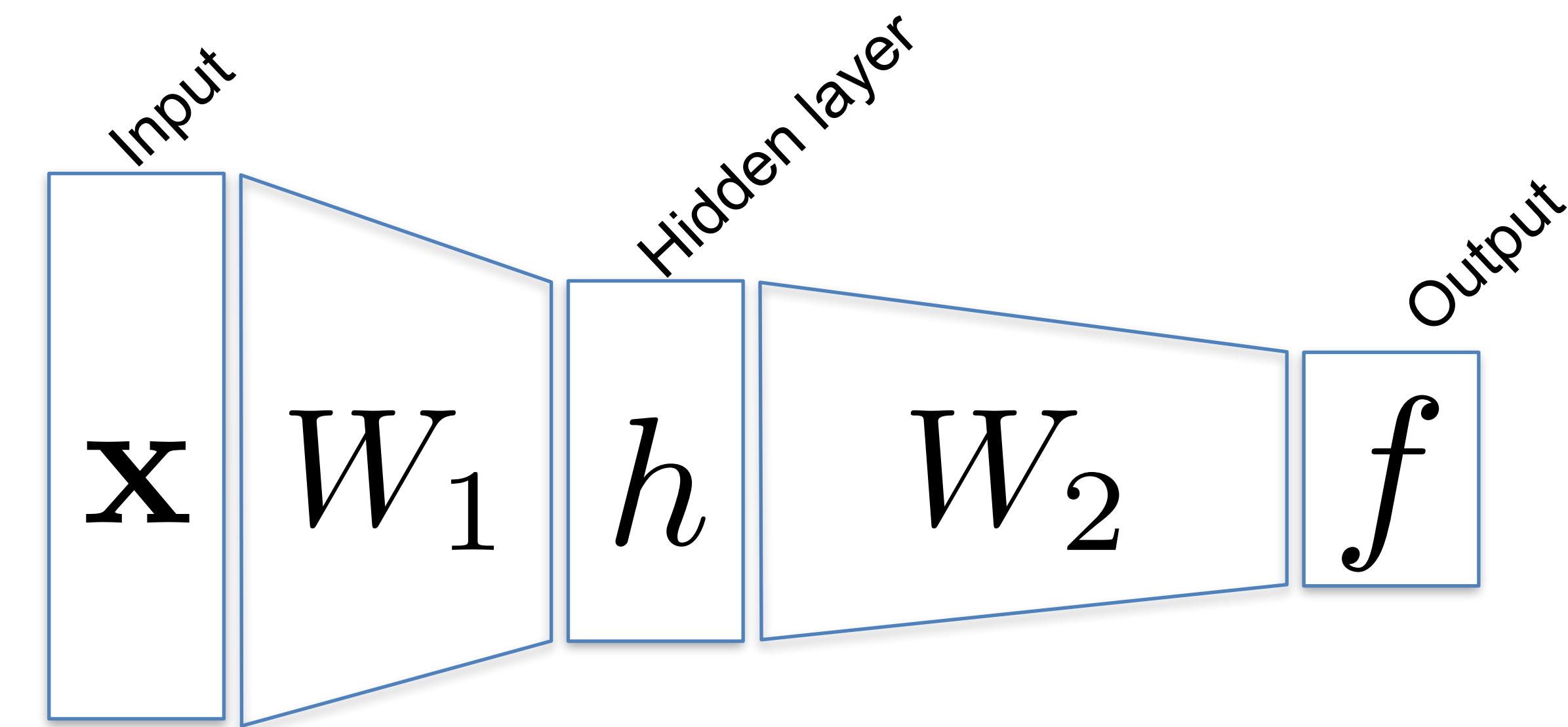
$$g(x) = \max(0, x)$$

$$g(f(x)) = \max(0, f(x))$$



2-layer Network

$h \leftarrow$ ^{non-linear} applied

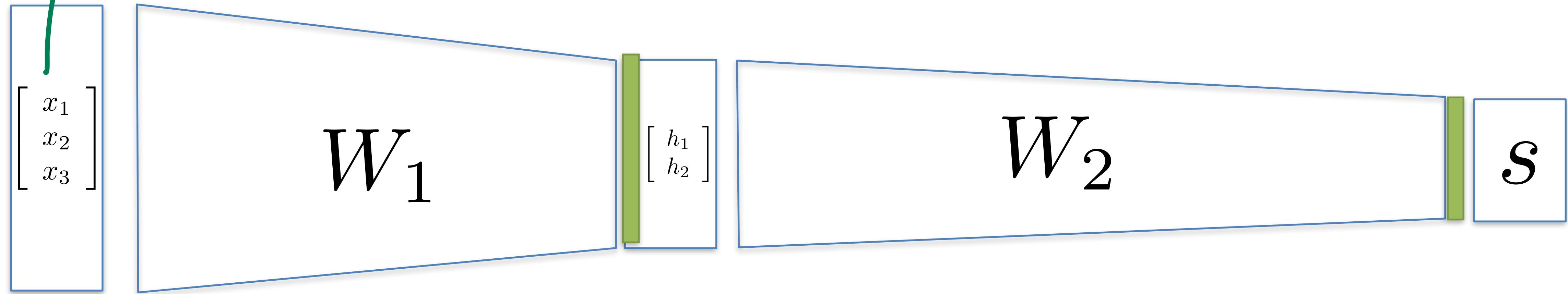


Multi-layer perceptron (MLP)



2-layer Network

一个数据点



$$h_1 = w_1^{11}x_1 + w_1^{12}x_2 + w_1^{13}x_3$$

$$h_2 = w_1^{21}x_1 + w_1^{22}x_2 + w_1^{23}x_3$$

$$\mathbf{W}_1 = \begin{bmatrix} w_1^{11} & w_1^{12} & w_1^{13} \\ w_1^{21} & w_1^{22} & w_1^{23} \end{bmatrix}$$

$$s = w_2^{11}h_1 + w_2^{12}h_2$$

$$\mathbf{W}_2 = \begin{bmatrix} w_2^{11} & w_2^{12} \end{bmatrix}$$

Activation Functions

- **ReLU(x)** $\text{RELU}(x) = \max(0, x)$

ReLU 機械 학습과 텐서플로우

- **Sigmoid** $\sigma(x) = \frac{1}{1 + \exp(-x)}$ $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

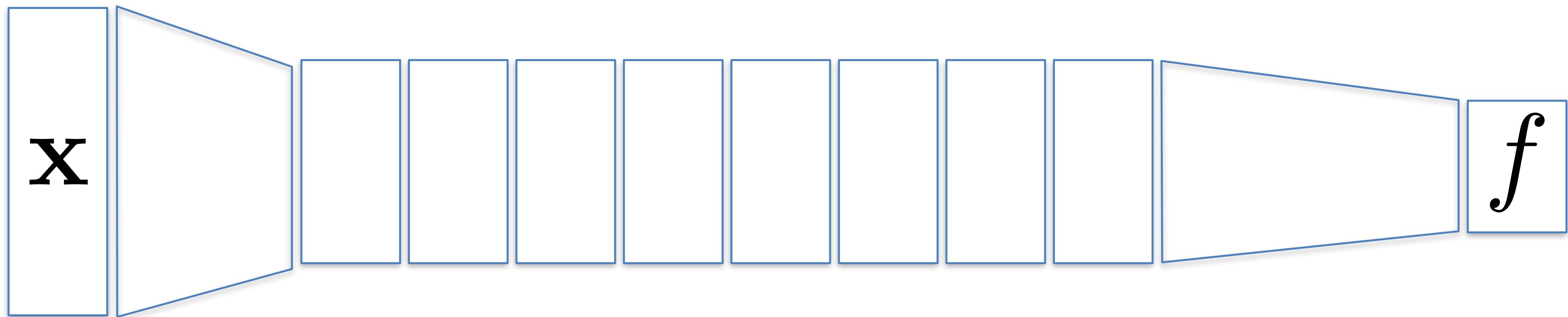
- **Tanh** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

- **Leaky ReLU** $\max(\lambda x, x)$ $\lambda << 1$

- **ELU**

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

Deep Networks



$$f = W_i \max(0, W_{i-1} \max(0, \dots W_0 \mathbf{x}))$$

Loss function and Optimization

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^i - y^i)^2$$

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = 0$$

LS Solution for Linear Regression

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^1)^T \\ (\mathbf{x}^2)^T \\ \vdots \\ (\mathbf{x}^N)^T \end{bmatrix}$$

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Gradient-based Optimisation

Initialize θ

$$\min_{\theta} f_{\theta}(\{\mathbf{x}^{(i)}\})$$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} f_{\theta}$$

$$\max_{\theta} f_{\theta}(\{\mathbf{x}^{(i)}\})$$

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} f_{\theta}$$

gradient
scalar (how far we go)

Gradient Descent

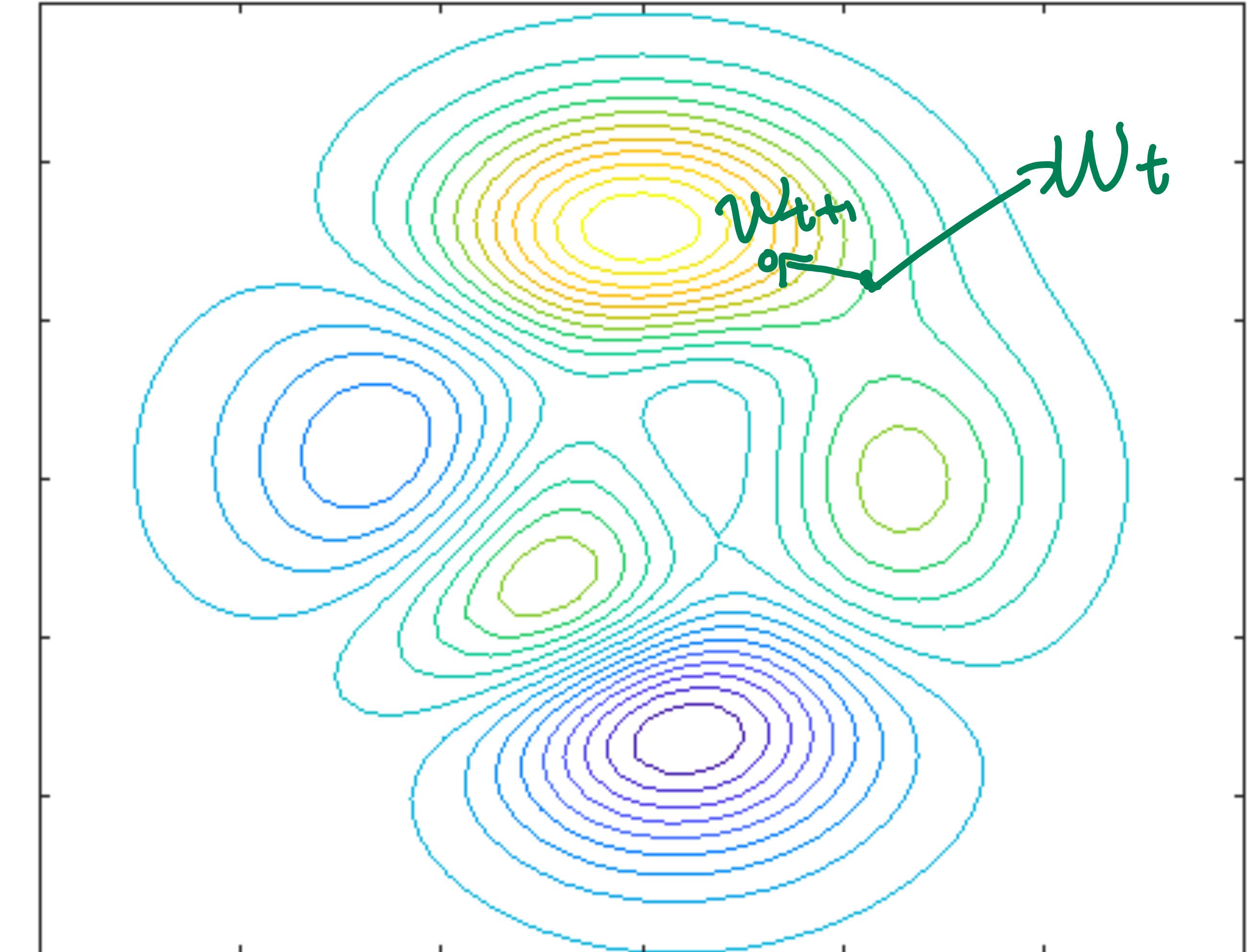
$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

learning rate

```
w = initialize_weights()  
for t in range(num_steps):  
    dw = compute_gradient(loss_fn, data, w)  
    w -= learning_rate * dw
```

- Hyperparameters?
- a) initialize_weights
 - b) learning_rate
 - c) number of steps/truncation condition

more yellow \Rightarrow value \downarrow



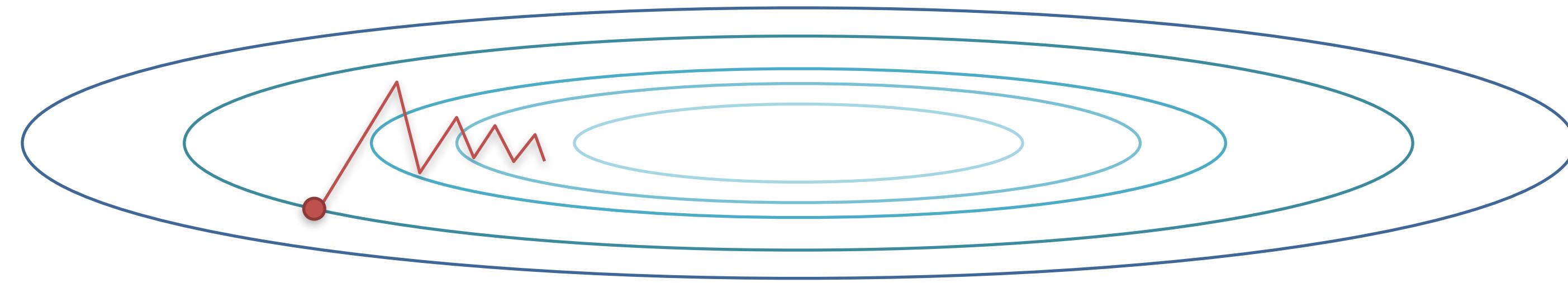
Stochastic Gradient Descent

$$J(\mathbf{w}) = \mathbb{E}_{\mathbf{x}, y \sim p_{\text{data}}} L(\mathbf{x}, y, \mathbf{w}) = \frac{1}{N} \sum_i L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) \approx \frac{1}{m} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

Problems with Gradient Descent



1. Bad condition number \rightarrow slow convergence

go beyond solution
(一大步進太大) $\alpha \cdot \nabla f$)

2. Local minima

3. Saddle point

Stochastic/Batch Gradient Descent

$$J(\mathbf{w}) = \mathbb{E}_{\mathbf{x}, y \sim p_{\text{data}}} L(\mathbf{x}, y, \mathbf{w}) = \frac{1}{N} \sum_i^{\mathcal{N}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) \approx \frac{1}{m} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

- Hyperparameters?
- a) initialize_weights
 - b) learning_rate
 - c) number of steps/truncation condition
 - d) batch size (8, 16, 32, ...)
 - e) sampling

$$m \ll N$$

Interactive Demo

Linear Classification Loss Visualization

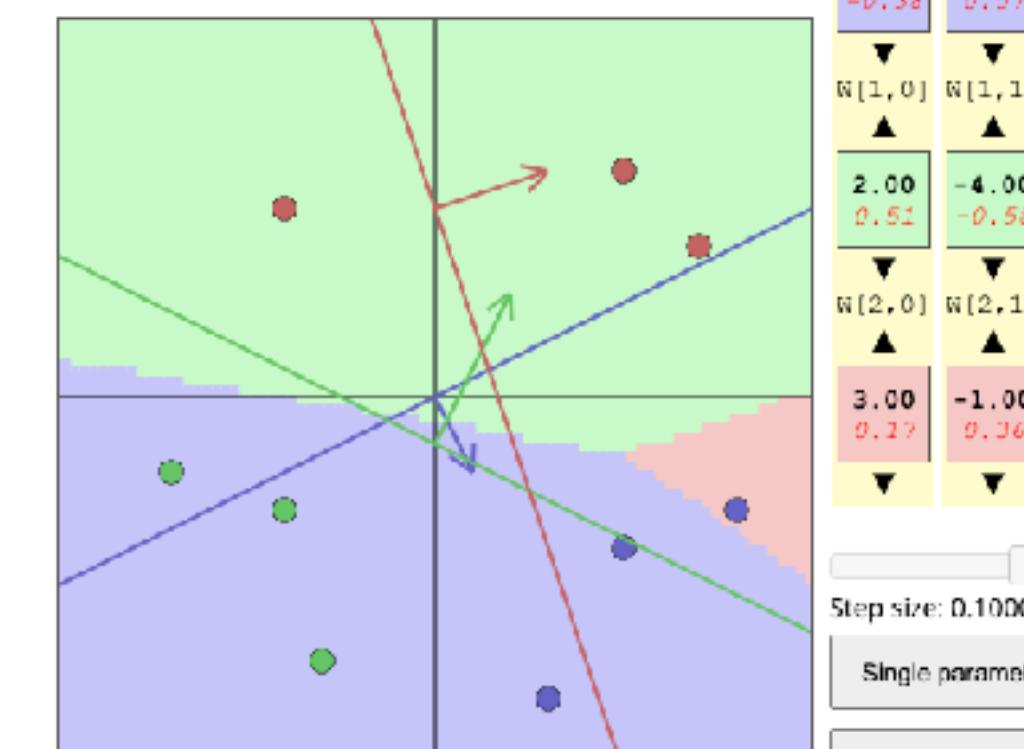
These linear classifiers were written in Javascript for Stanford's CS231n: Convolutional Neural Networks for Visual Recognition.

The class scores for linear classifiers are computed as $f(x_i; W, b) = Wx_i + b$, where the parameters consist of weights W and biases b . The training data is x_i with labels y_i . In this demo, the datapoints x_i are 2-dimensional and there are 3 classes, so the weight matrix is of size $[3 \times 2]$ and the bias vector is of size $[3 \times 1]$. The multiclass loss function can be formulated in many ways. The default in this demo is an SVM that follows [Weston and Watkins 1999]. Denoting f as the $[3 \times 1]$ vector that holds the class scores, the loss has the form:

$$L = \underbrace{\frac{1}{N} \sum_i \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)}_{\text{data loss}} + \lambda \underbrace{\sum_k \sum_l W_{kj}^2}_{\text{regularization loss}}$$

Where N is the number of examples, and λ is a hyperparameter that controls the strength of the L2 regularization penalty $R(W) = \sum_k \sum_l W_{kj}^2$. On the bottom right of this demo you can also flip to different formulations for the Multiclass SVM including One vs All (OVA) where a separate binary SVM is trained for every class independently (vs. other classes all labeled as negatives), and Structured SVM which maximizes the margin between the correct score and the score of the highest runner-up class. You can also choose to use the cross-entropy loss which is used by the Softmax classifier. These losses are explained in the [CS231n notes on Linear Classification](#).

Datapoints are shown as circles colored by their class. Parameters W, b are shown (red/green/blue). The background regions are colored by below. The value is in bold whichever class is most likely at any point according to the current weights. Each classifier is visualized by a line that with backprop) is in red. indicates its zero score level set. For example, the blue *italic* below. Click the classifier computes scores as $W_{0,0}x_0 + W_{0,1}x_1 + b_0$ and the triangles to control the blue line shows the set of points (x_0, x_1) that give score of parameters. zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is. Note: you can drag the datapoints.



Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i . The next three columns are the three class scores from each classifier $f_i(x_i; W, b) = Wx_i + b$ (E.g. $s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]$). The last column is the data loss for a single example, L_i .

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.30	-0.10	0.60	0.30
0.80	0.30	0	1.40	0.90	1.00	1.70
0.30	0.80	0	1.90	-2.10	-0.40	0.00
-0.40	0.30	1	0.20	-1.50	-2.00	3.20
-0.30	0.70	1	1.10	-2.90	-2.10	6.80
-0.70	0.20	1	-0.30	-1.70	-2.80	2.40
0.70	-0.40	2	-0.10	3.50	2.00	2.50
0.50	-0.60	2	-0.70	3.90	1.60	3.30
-0.40	-0.50	2	-1.40	1.70	-1.20	4.70
mean:						2.77
Total data loss: 2.77						
Regularization loss: 3.50						
Total loss: 6.27						

Step size: 0.10000
L2 Regularization strength: 0.10000

Multiclass SVM loss formulation:
 Weston Watkins 1999
 One vs. All
 Structured SVM
 Softmax

Linear Binary Classifier

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C \\ -1 & \text{if } \mathbf{x}_i \notin C \end{cases}$$

Find weights w such that

$$\begin{aligned} \mathbf{x}_i^T \mathbf{w} \geq 0 & \quad \text{if } y_i = 1 \\ \mathbf{x}_i^T \mathbf{w} < 0 & \quad \text{if } y_i = -1 \end{aligned}$$

$$y_i (\mathbf{x}_i^T \mathbf{w}) \geq 0$$

Perceptron Algorithm [Rosenblatt '57]

- Initialize \mathbf{w}
- While $\mathcal{L}(\mathbf{w}) > 0$
 - $V \leftarrow \text{set of indices with } y_i(\mathbf{x}_i^T \mathbf{w}) < 0$
 - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \sum_{i \in V} (-y_i \mathbf{x}_i) = \mathbf{w} + \alpha \sum_{i \in V} y_i \mathbf{x}_i$

Linear Classifier via Regression

Linear Classifier via Regression

(\because 很明显 classify 也能用线性
回归)

$$L(\theta) = \frac{1}{2} \sum_{i=1}^n \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

$$h(\mathbf{x}) = \sum_{i=1}^p \theta_i x_i = \theta^T \mathbf{x}$$

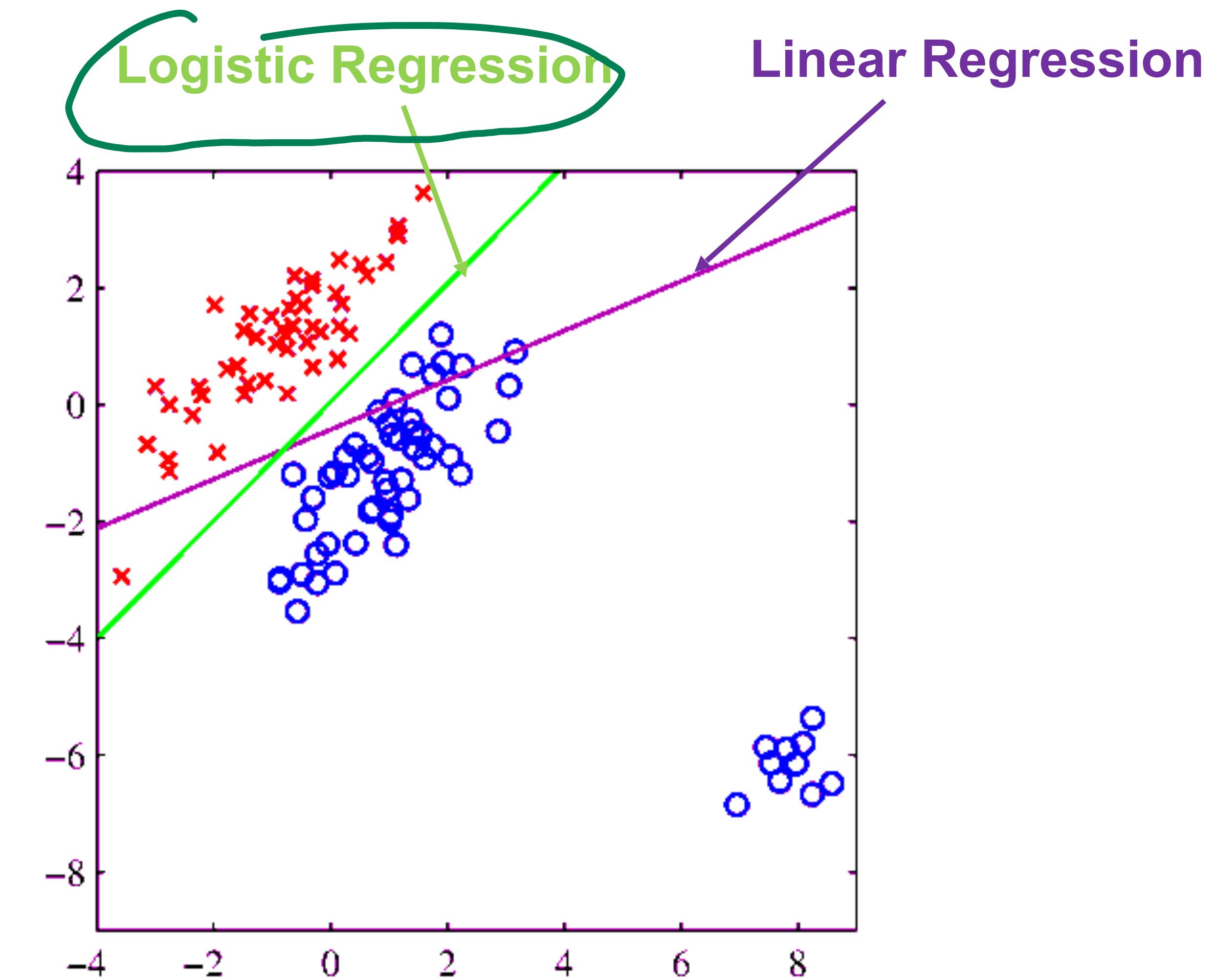
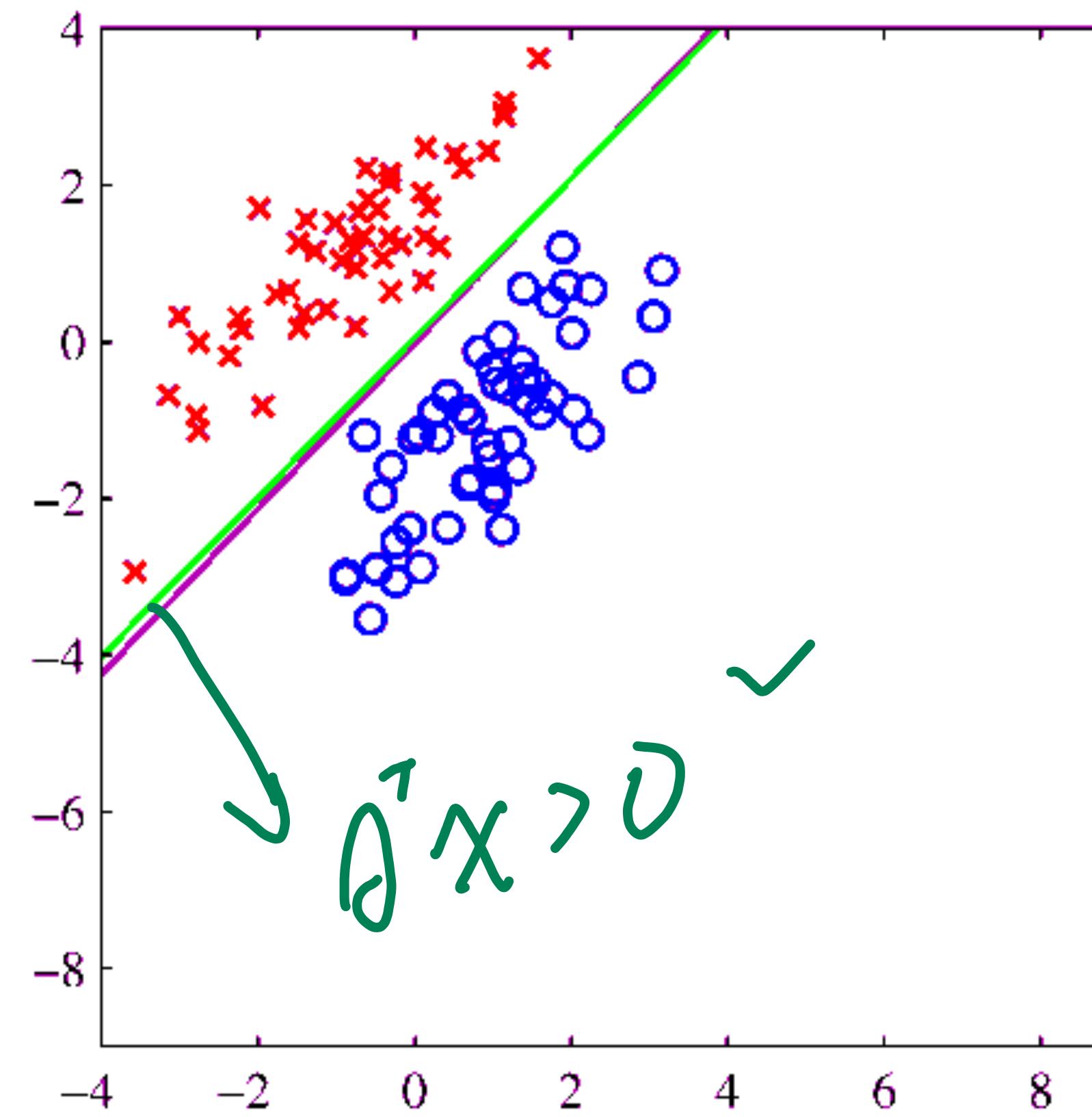
$$\theta = (X^T X)^{-1} X^T \mathbf{y}$$

$\frac{\partial \theta^T \mathbf{x}}{\partial \theta} = \mathbf{x}$

Compute gradient

$$\begin{aligned} \theta &\leftarrow \theta - \frac{\nabla L}{\nabla \theta} \\ \frac{1}{2} \sum_{i=1}^n 2(h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \mathbf{x}^{(i)} &= \sum_{i=1}^n (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \mathbf{x}^{(i)} \end{aligned}$$

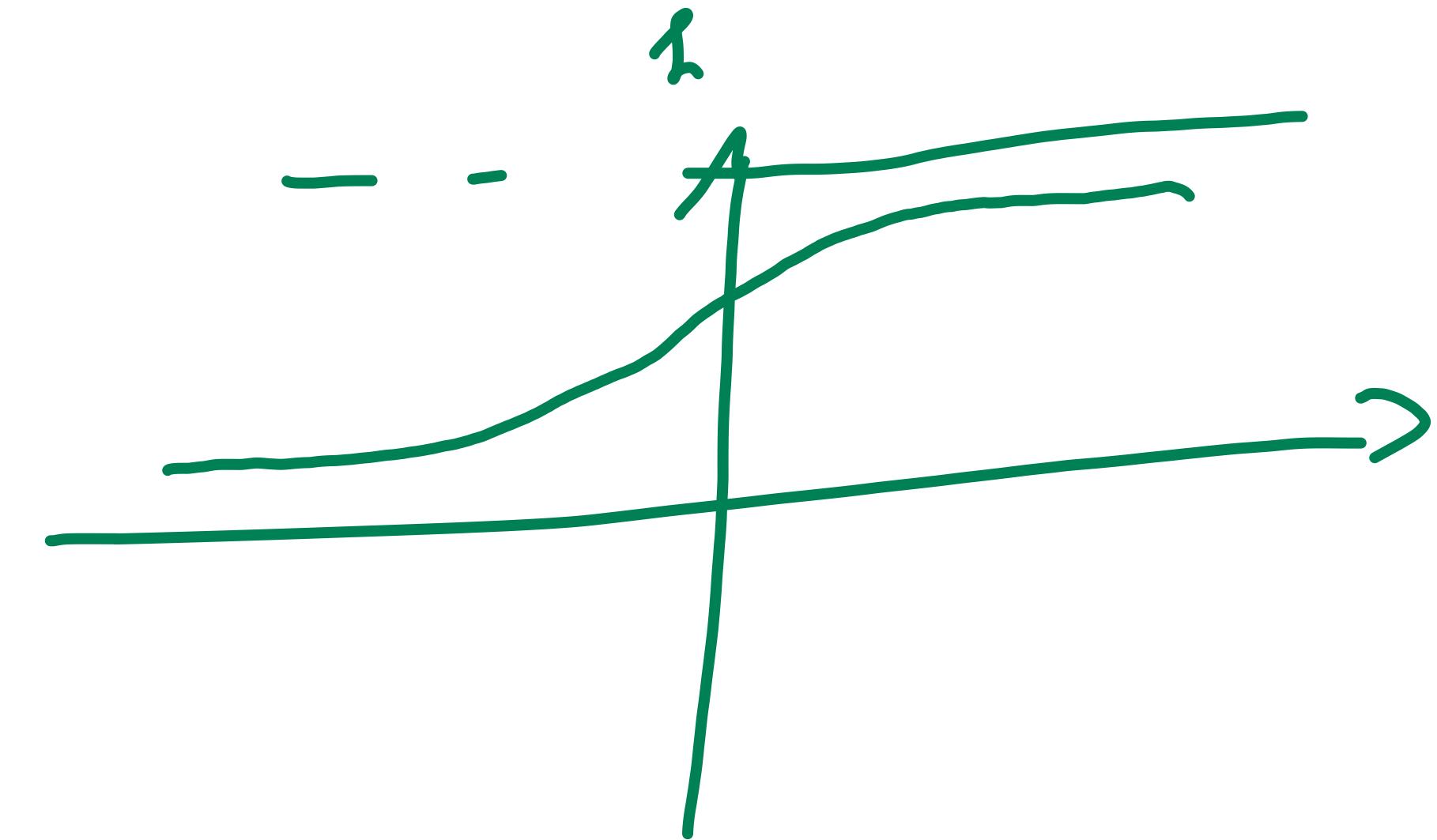
Linear vs Logistic Regression



Logistic Regression

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$\sigma'(z) = \frac{d\sigma(z)}{dz}$$



Logistic Regression

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

~~scalar data~~

$$h(\mathbf{x}) = \sigma(\theta^T \mathbf{x}) = \frac{1}{1 + \exp(\theta^T \mathbf{x})}$$

$$p(y=1|\mathbf{x}; \theta) = h_\theta(\mathbf{x})$$

$$p(y=0|\mathbf{x}; \theta) = 1 - h_\theta(\mathbf{x})$$

$$p(y|\mathbf{x}; \theta) = \sigma(\theta^T \mathbf{x})^y (1 - \sigma(\theta^T \mathbf{x}))^{1-y}$$

Logistic Regression

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$h(\mathbf{x}) = \sigma(\theta^T \mathbf{x}) = \frac{1}{1 + \exp(\theta^T \mathbf{x})}$$

$$p(y|\mathbf{x}; \theta) = \sigma(\theta^T \mathbf{x})^y (1 - \sigma(\theta^T \mathbf{x}))^{1-y}$$

$$p(y|\mathbf{x}; \theta) = \prod_i \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} (1 - \sigma(\theta^T \mathbf{x}^{(i)}))^{1-y^{(i)}}$$

Logistic Regression

$$p(y|\mathbf{x}; \theta) = \prod_i \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} (1 - \sigma(\theta^T \mathbf{x}^{(i)}))^{1-y^{(i)}}$$

$$\log(p(y|\mathbf{x}; \theta))$$

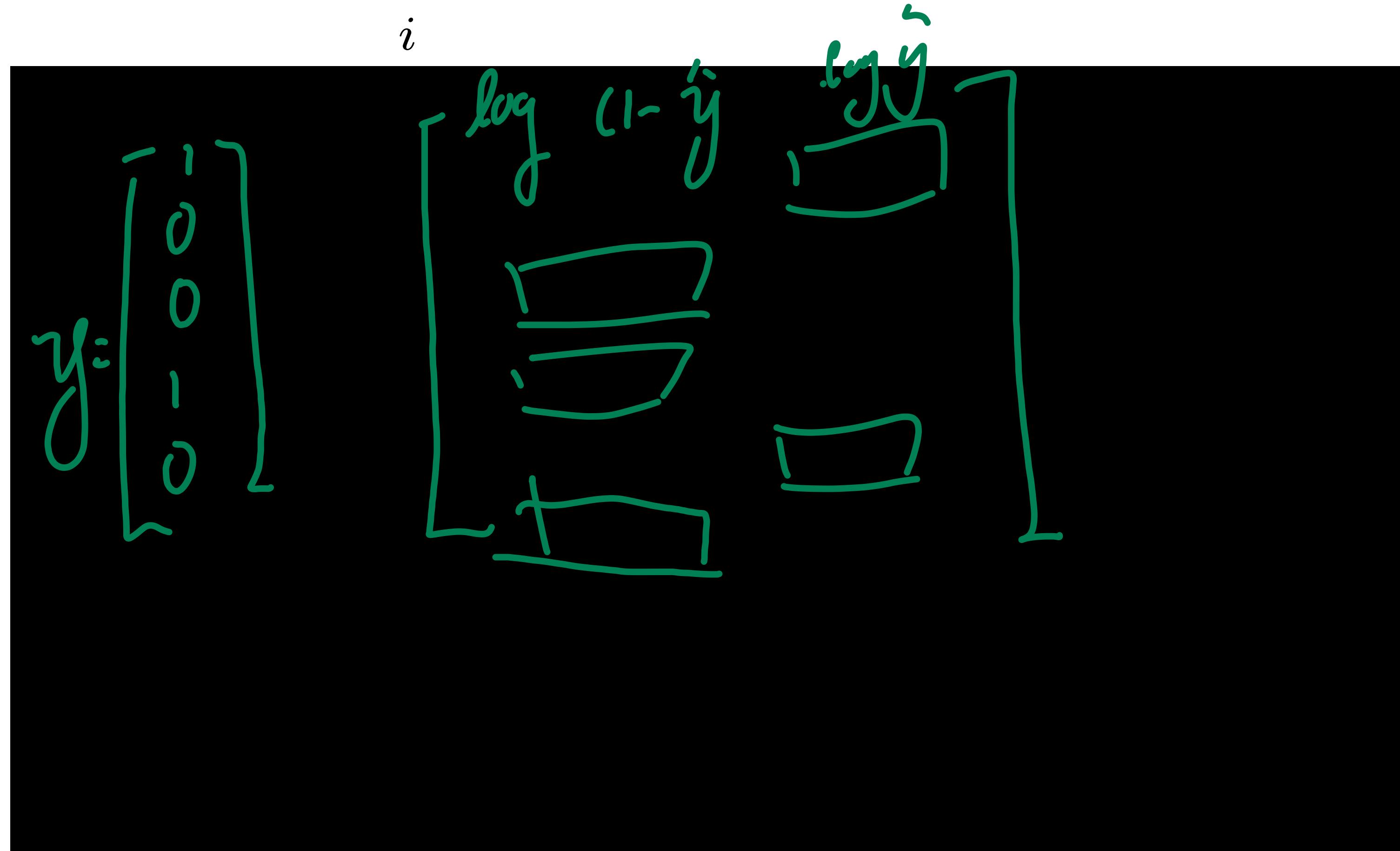
do find max

$$\theta = \theta + \alpha \left(y^{(i)} - h_\theta(\mathbf{x}^{(i)}) \right) \mathbf{x}^{(i)}$$

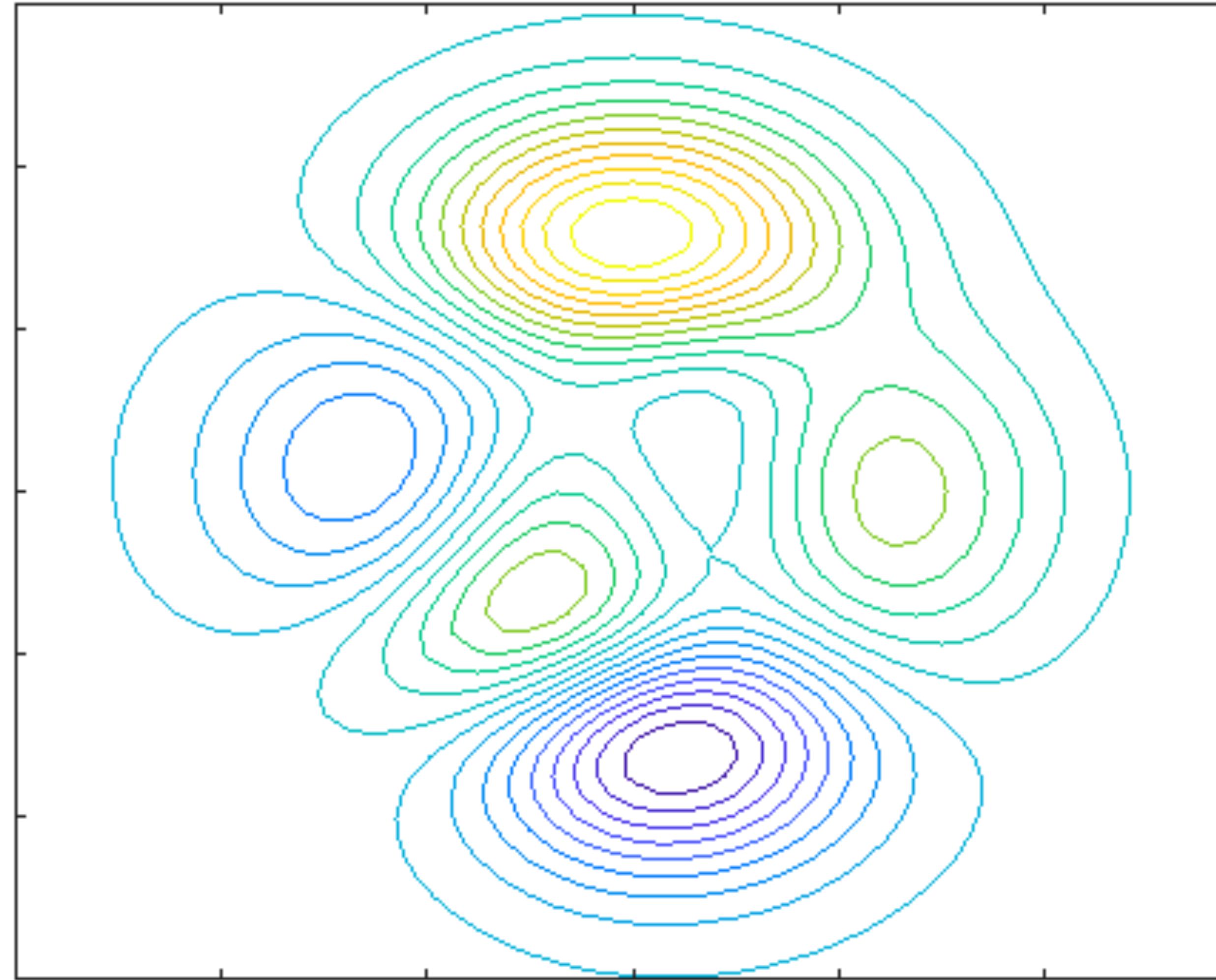
gradient ascend

Negative LogLikelihood

$$\log \mathbb{P}(\mathbf{D}|\theta) = \sum_i (y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log(1 - \hat{y}_{\theta,i}))$$



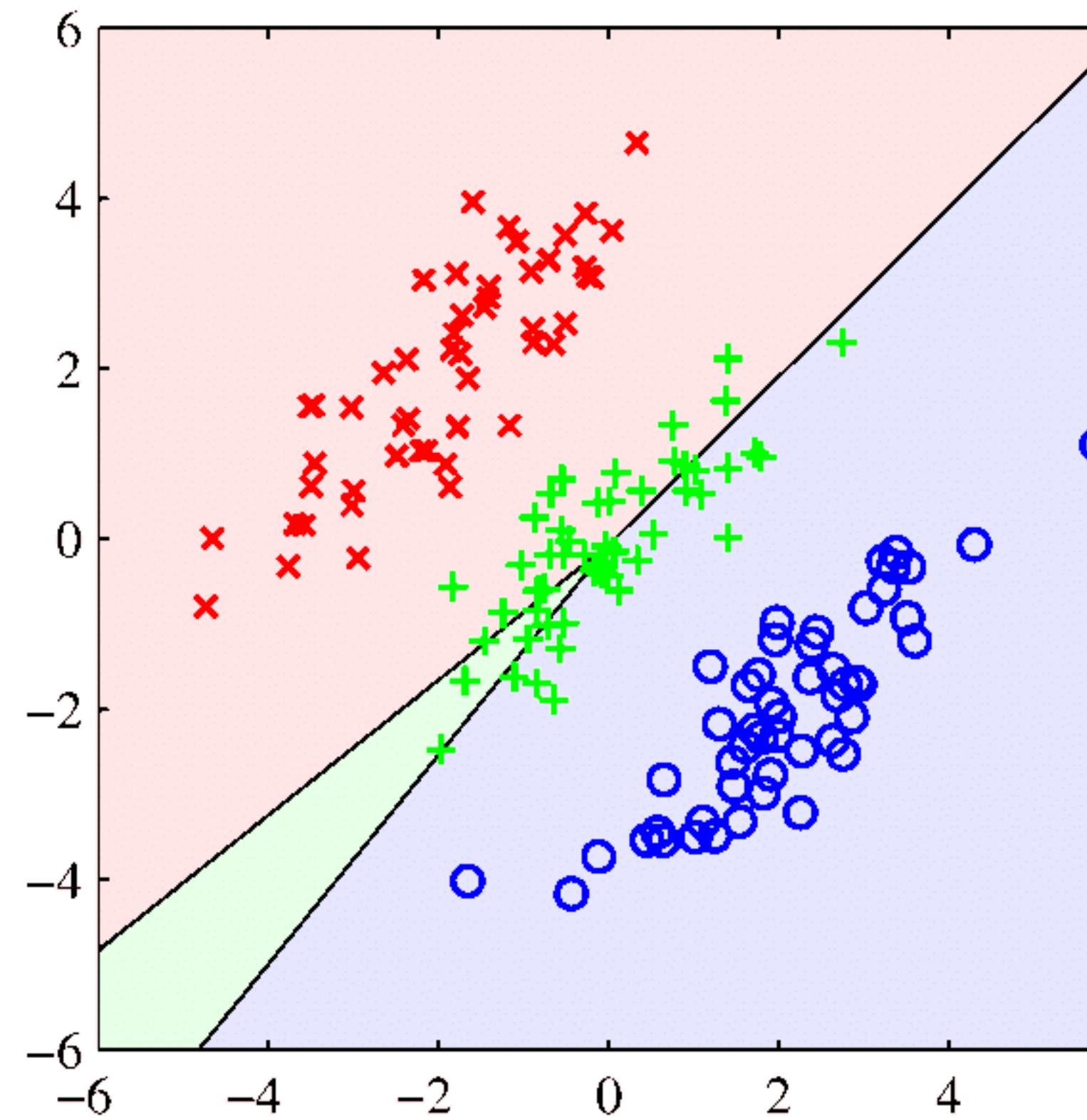
Problems with GD (or SGD)



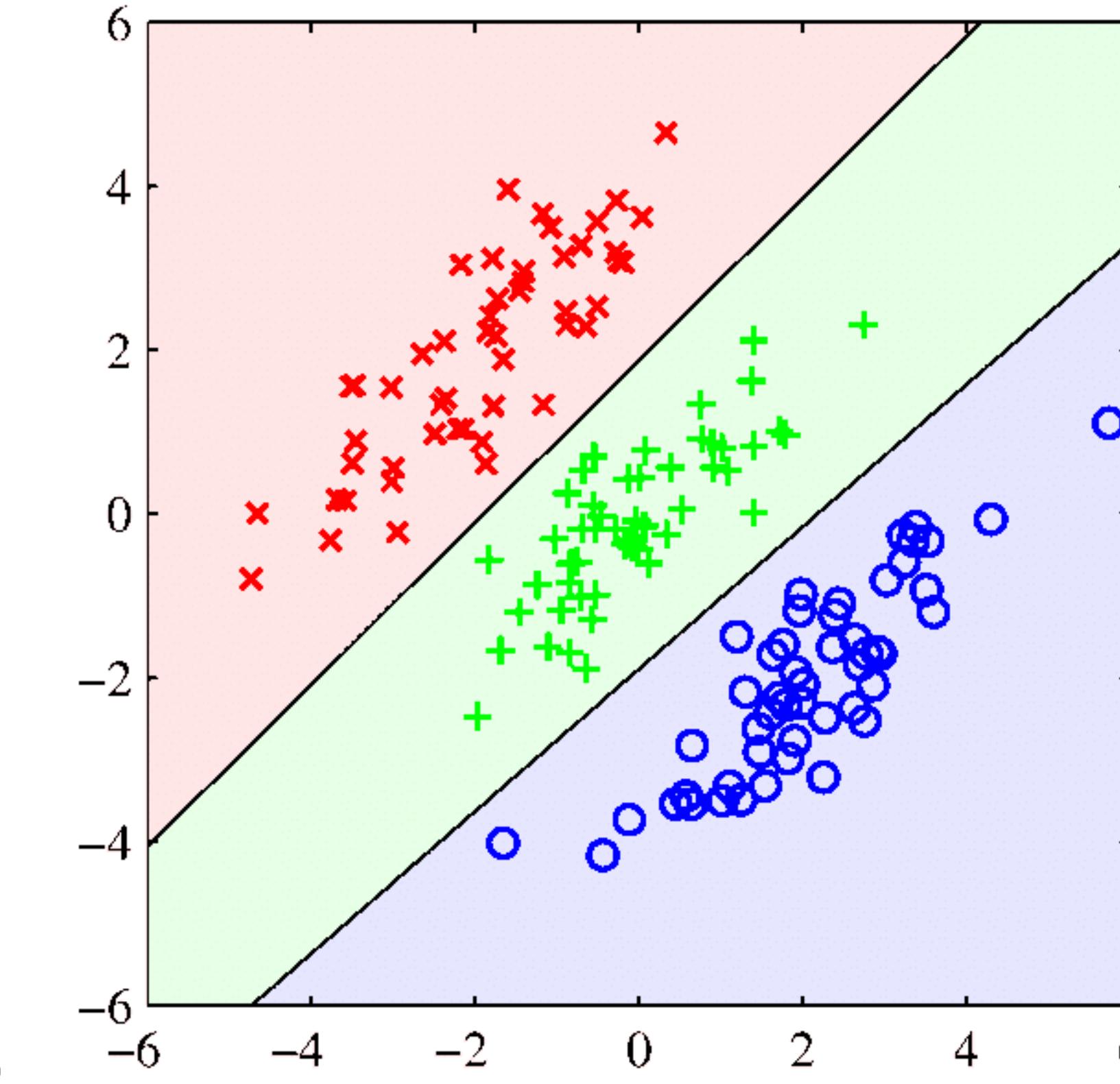
$$\begin{aligned}v_{t+1} &= \rho v_t + \nabla f(w_t) \\w_{t+1} &= w_t - \alpha v_{t+1} \\&= w_t - \alpha \rho v_t - \alpha \nabla f(w_t)\end{aligned}$$

Logistic vs Linear Regression, $n > 2$ classes

Linear regression

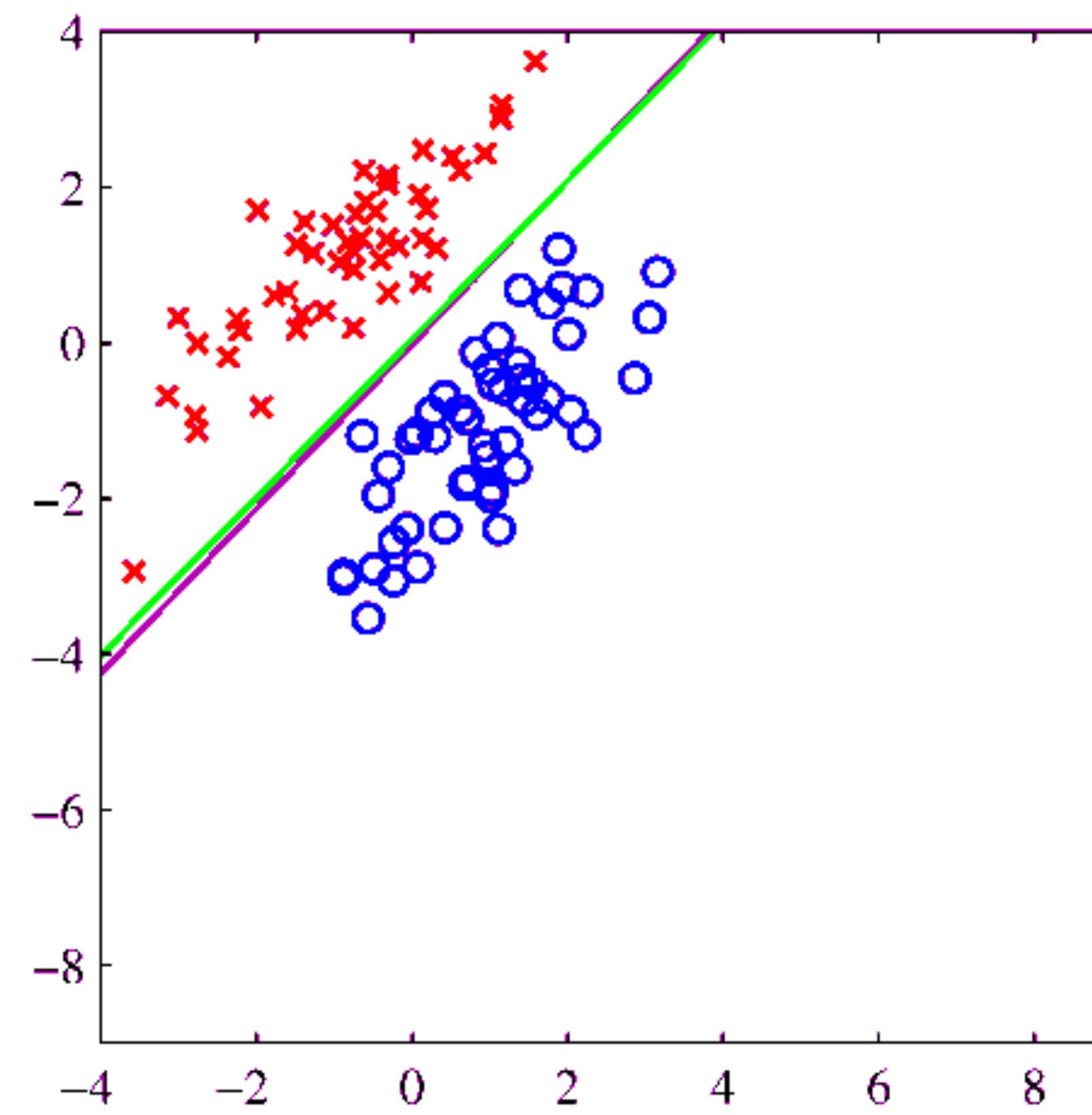


Logistic regression

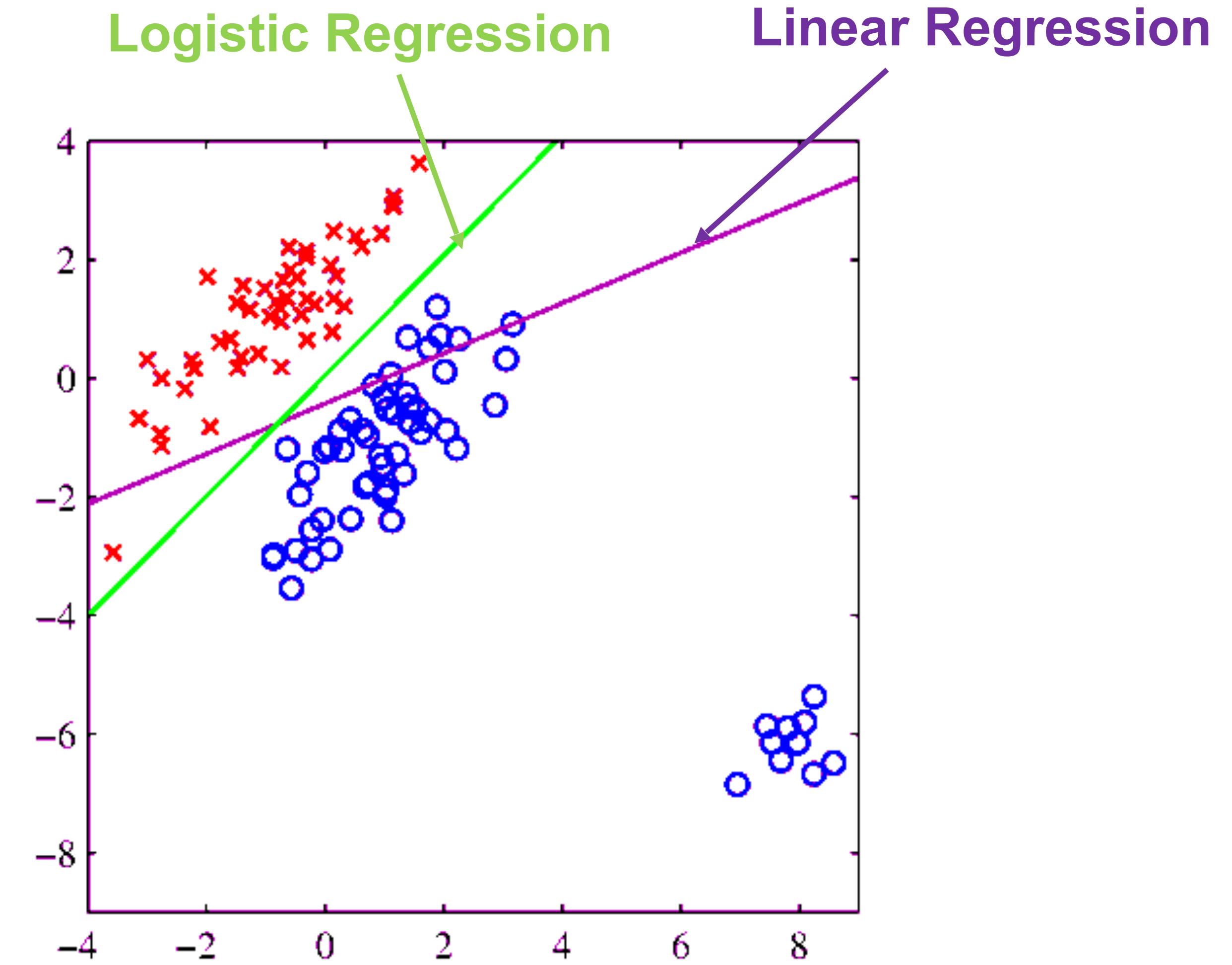


Logistic regression does not exhibit the masking problem

Logistic vs Linear Regression



Logistic Regression



Linear Regression

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:
 $\mathbf{y}^i = (0, 0, 1, 0)$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^N \end{bmatrix} = [\mathbf{y}_1 \mid \dots \mid \mathbf{y}_C] \quad \text{where } \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_C]$$

Loss function:

$$L(\mathbf{W}) = \sum_{c=1}^C (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)^T (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)$$

Least squares fit (decouples per class):

$$\mathbf{w}_c^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}_c$$

Multiple Classes via SoftMax