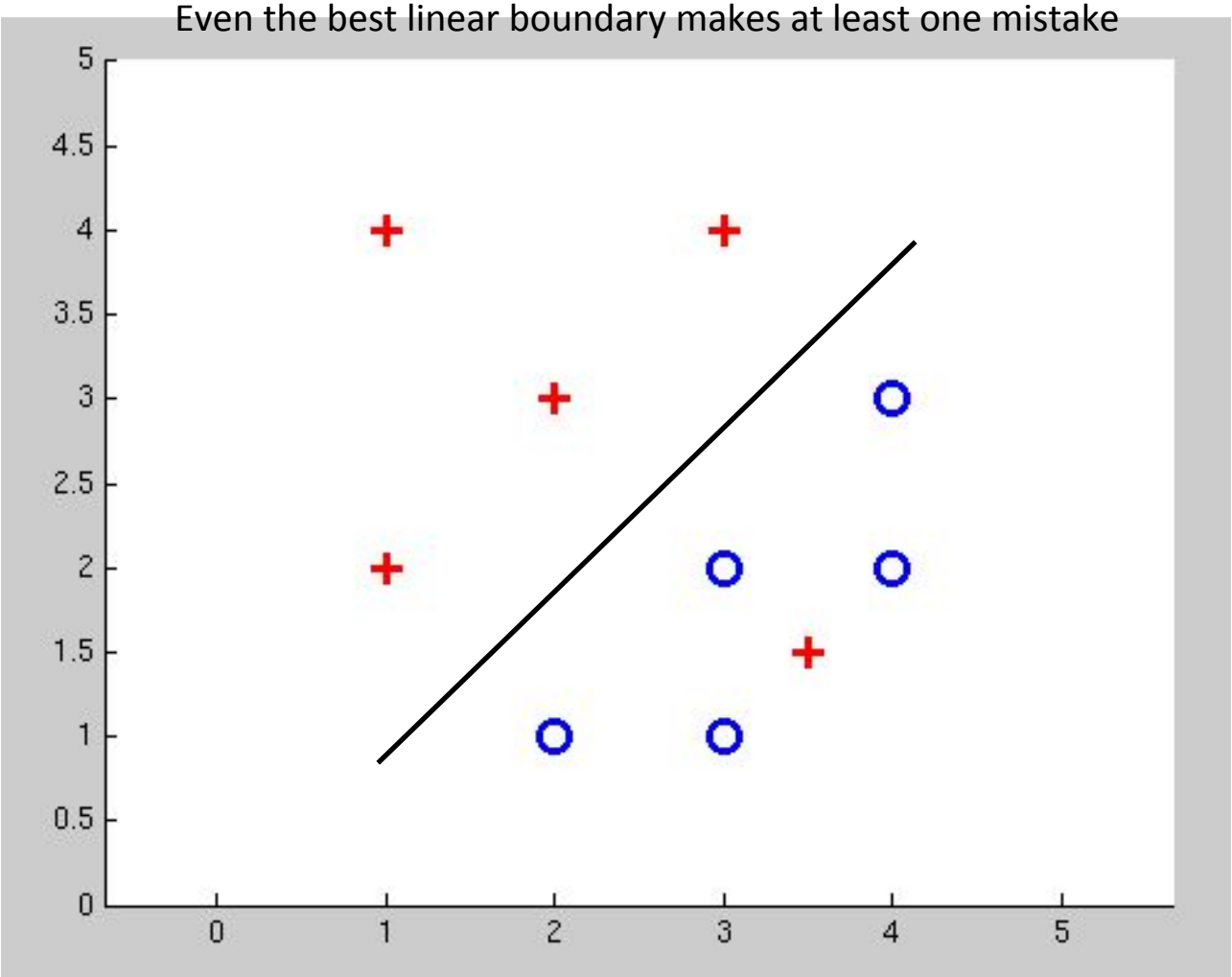
Improving the Perceptron

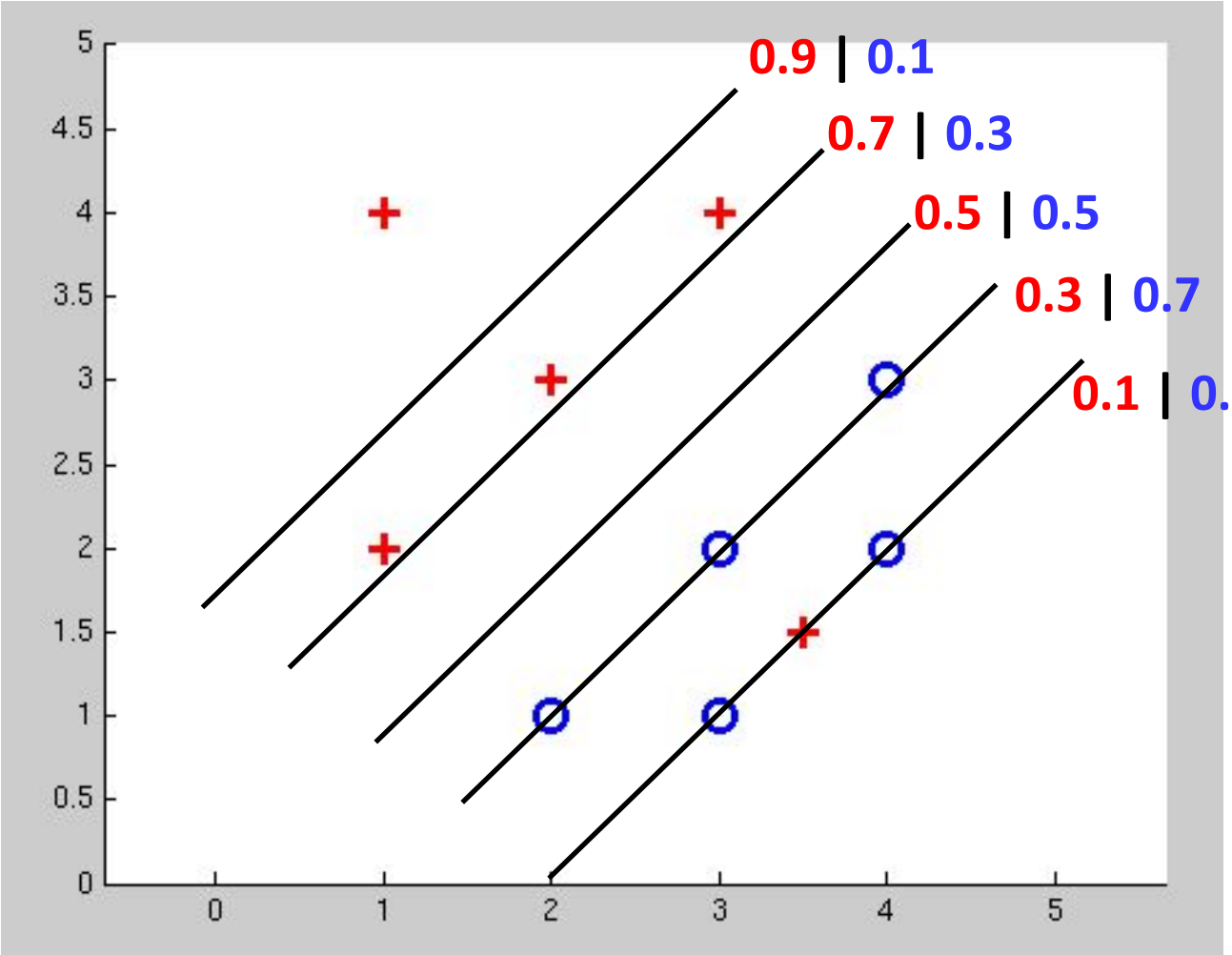
Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



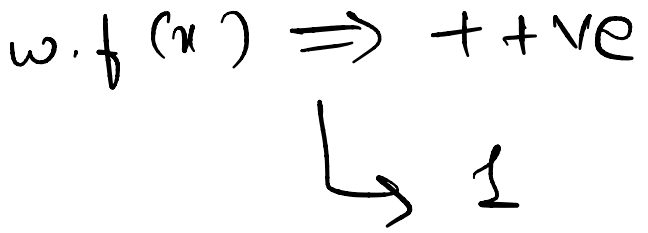
Non-Separable Case: Probabilistic Decision

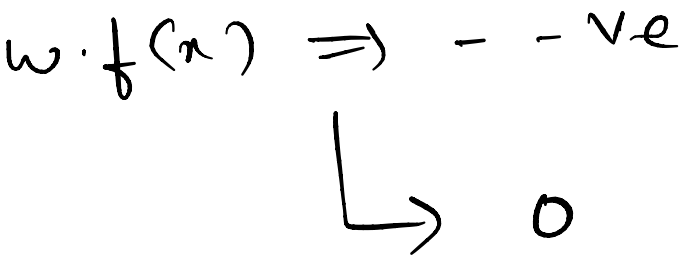
$[0\ 1]$ ← $w \cdot f(x) \rightarrow [-\infty, +\infty]$



How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability going to 1
- If $z = w \cdot f(x)$ very negative → want probability going to 0



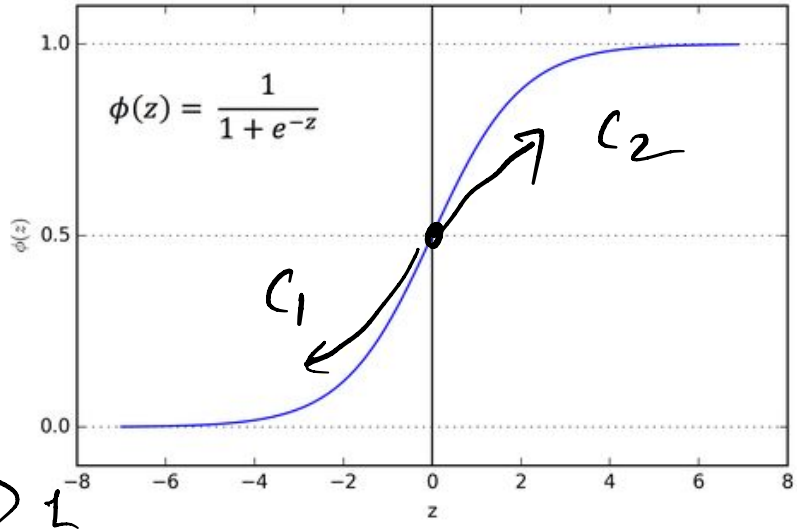


How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive ⇒ want probability going to 1
- If $z = w \cdot f(x)$ very negative ⇒ want probability going to 0

- Sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



$+\infty \to 1$

$-\infty \to 0$

loss funct^n / cost funct^n

Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \left[\sum_i \log P(y^{(i)}|x^{(i)}; w) \right]$$

with:

$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}} \quad \Rightarrow \; p_1$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}} \quad = \; 1 - p_1$$

= Logistic Regression

Percep + sigmoid + loglike → Logistic reg.

simple percept model

100, 1000

(1.) init w

(2) $z = f(x) \cdot w$

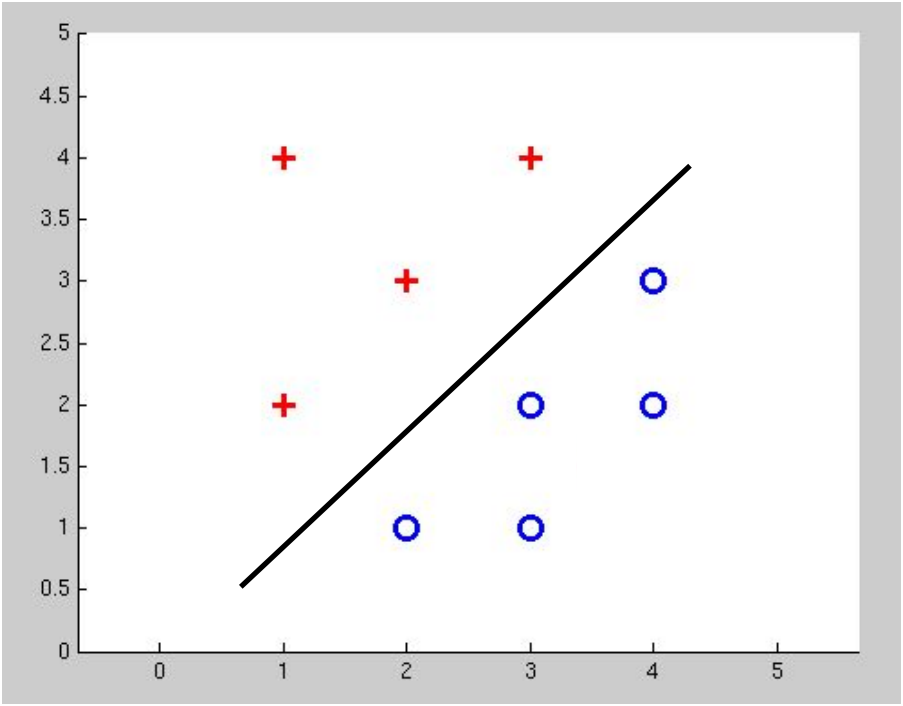
(3) $\frac{1}{1+e^{-z}}$ → p_1

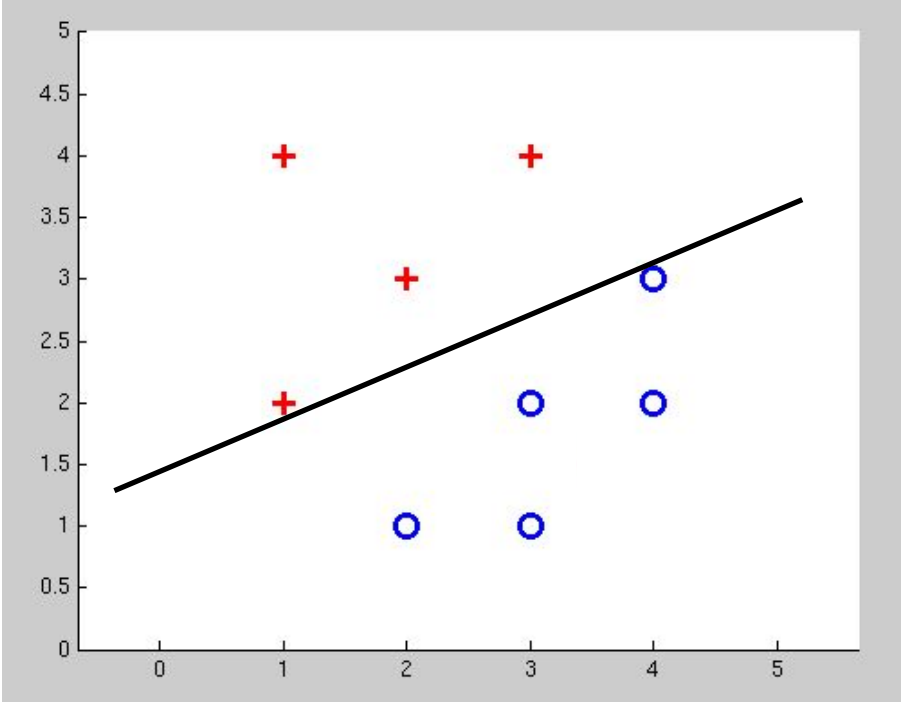
$y = 1 \Rightarrow p_1$

$y = 0 \Rightarrow 1 - p_1$

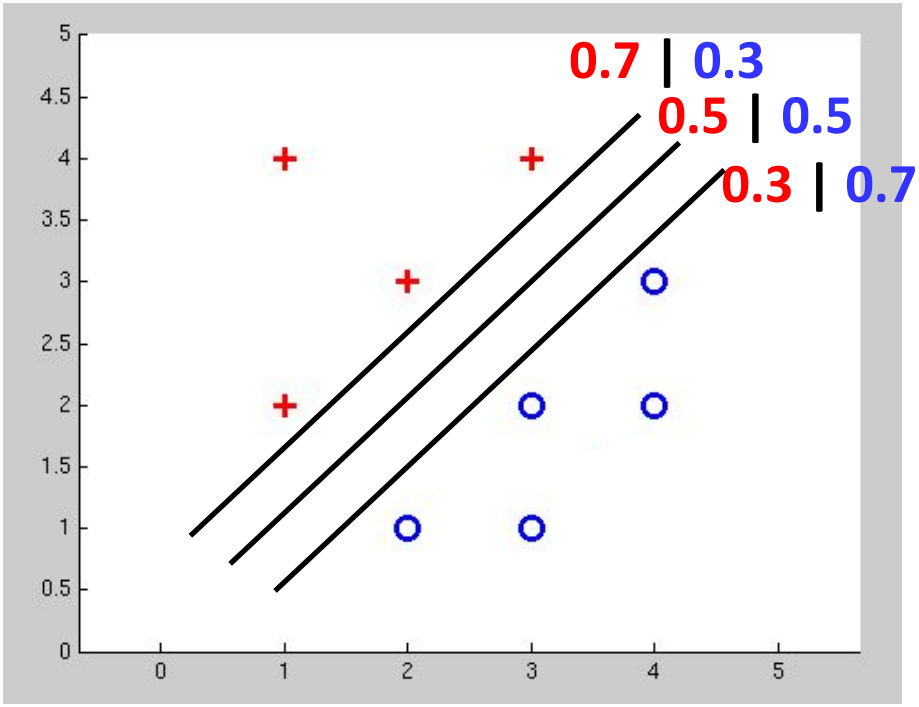
score $= \Sigma p$

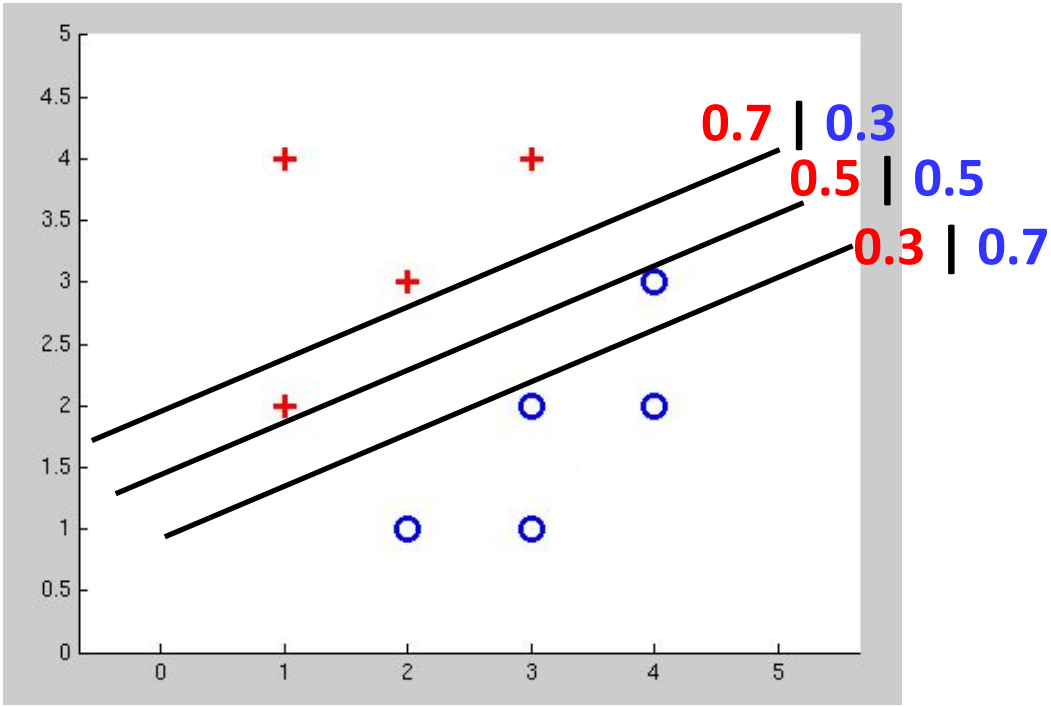
Separable Case: Deterministic Decision – Many Options





Separable Case: Probabilistic Decision – Clear Preference





Multiclass Logistic Regression

- Recall Perceptron:
  - A weight vector for each class: w_y
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins $y = \arg\max_y w_y \cdot f(x)$

- How to make the scores into probabilities?

$$z_1, z_2, z_3 \rightarrow \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

original activations

softmax activations

$\frac{1}{1+e^{-z}}$

Neural networks

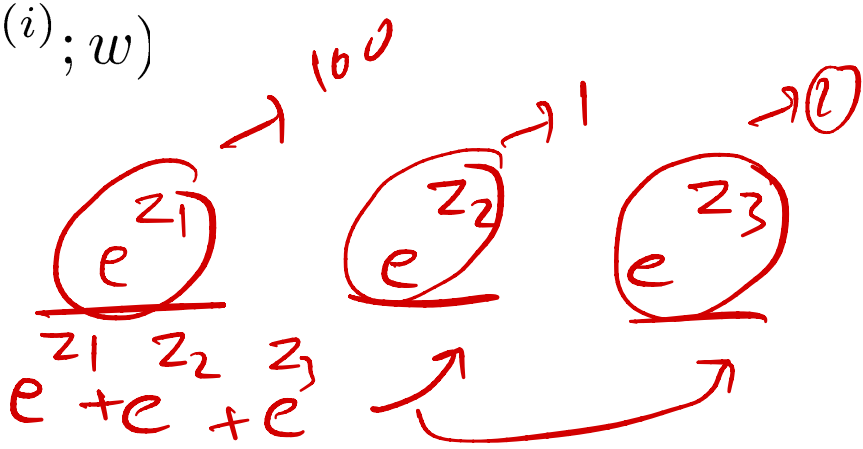
Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)};w)$$

with:

$$P(y^{(i)}|x^{(i)};w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$



= Multi-Class Logistic Regression

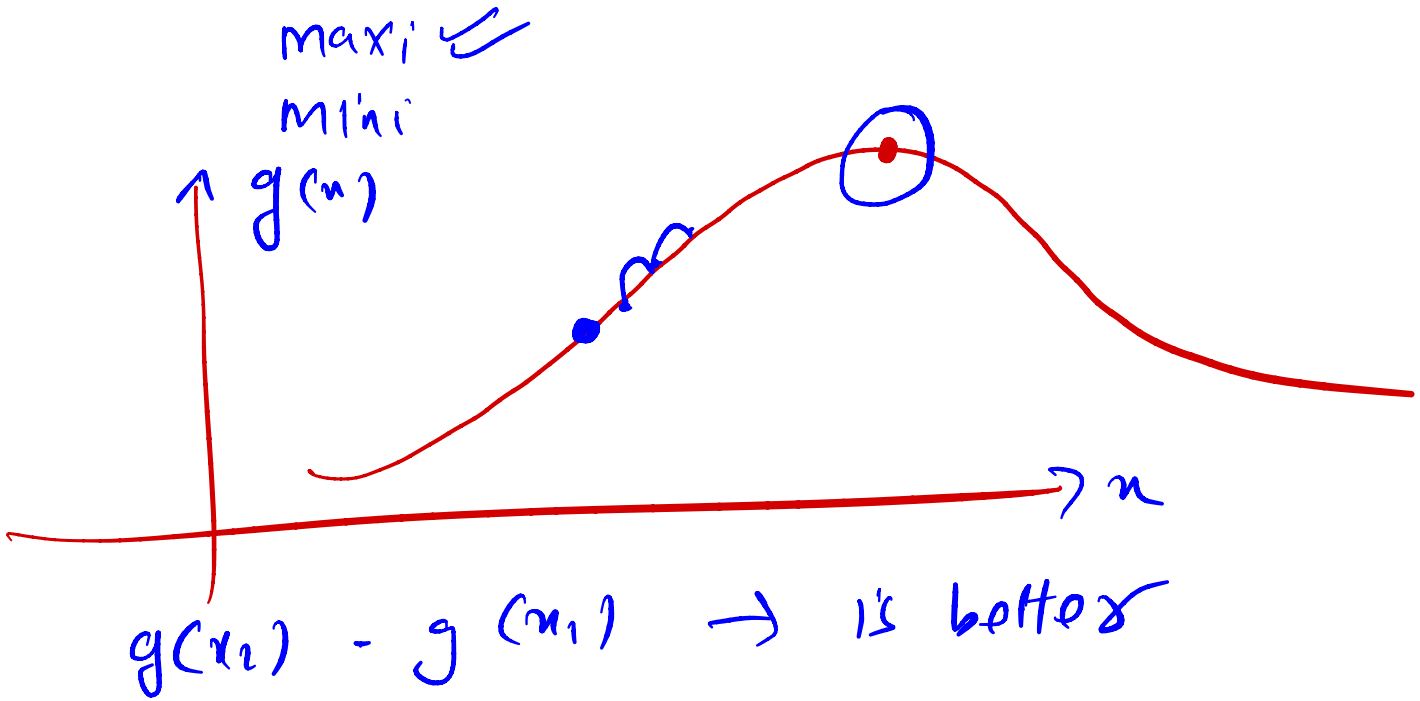
Choosing of w ?

loop → ite

w → 0.1 — 100
0.1

w → -5000 to 5000
0.01

Hill Climbing



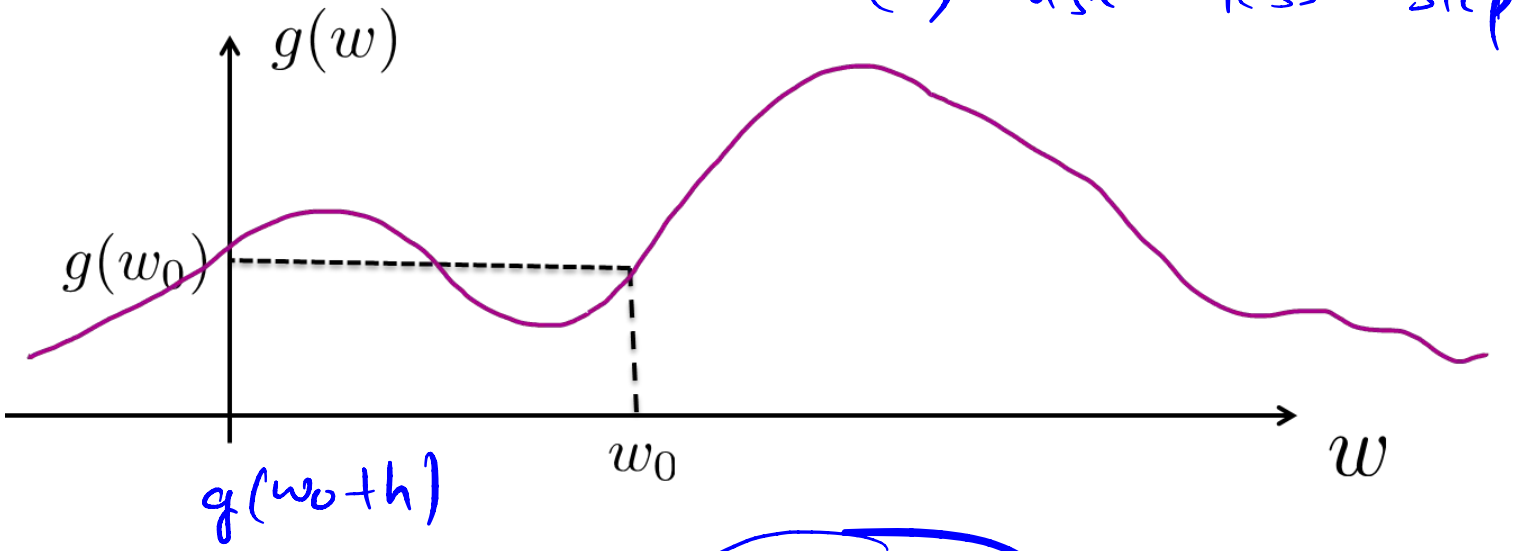
$g(n_2) - g(n_1)$ → is better

gradient

1-D Optimization

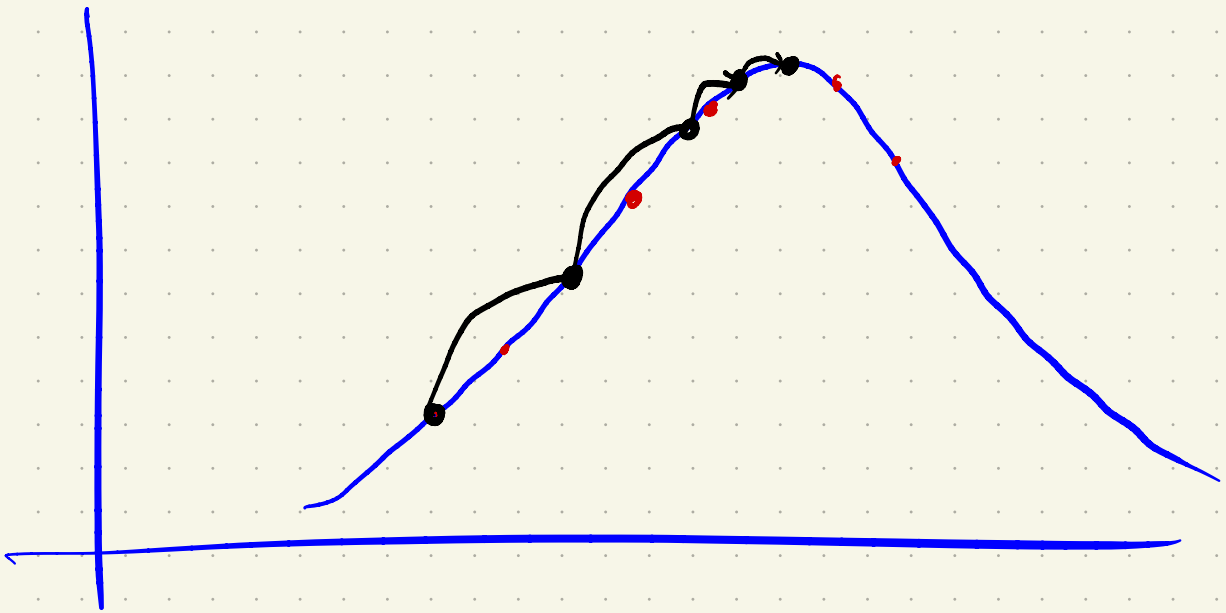
(1) decide to go up or down

(2) use less step



- Could evaluate $g(w_0 - h)$ and $g(w_0 + h)$
  - Then step in best direction

- Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$ = +ve = -ve
  - Tells which direction to step into

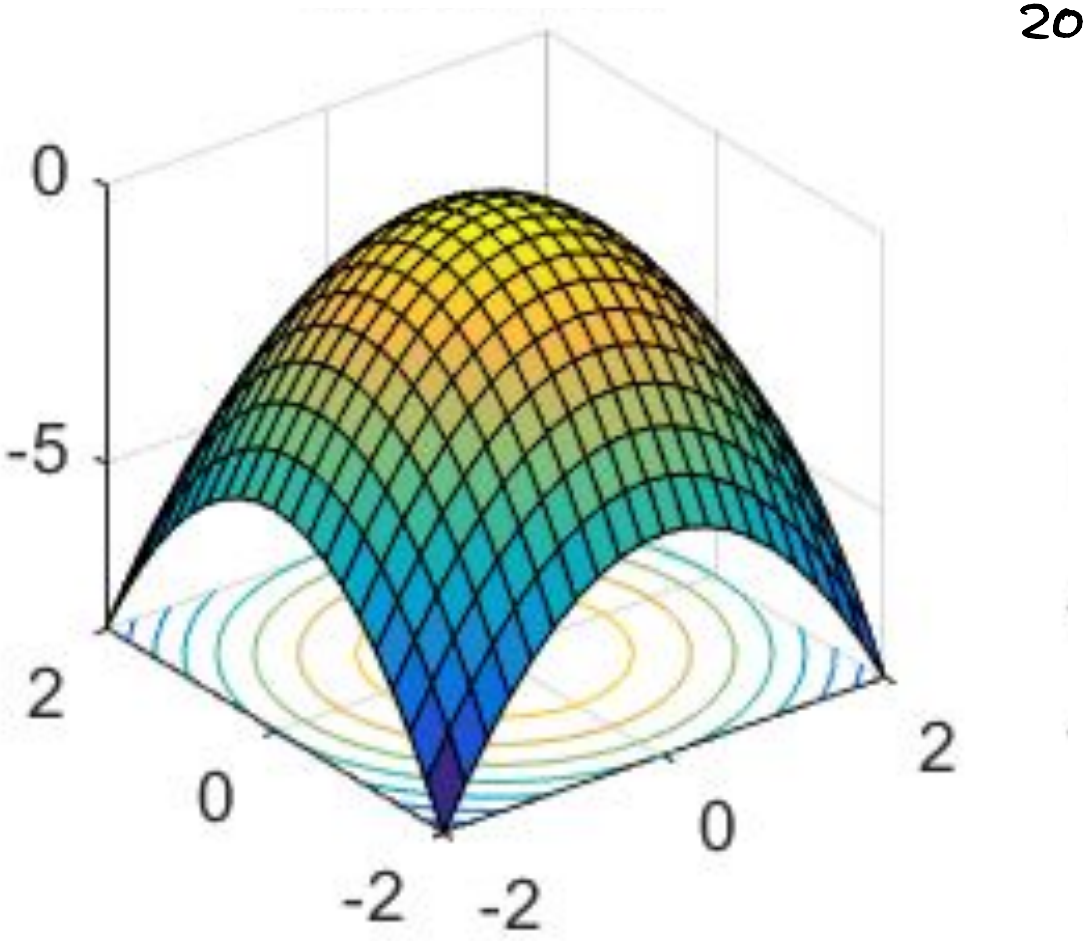


n-D

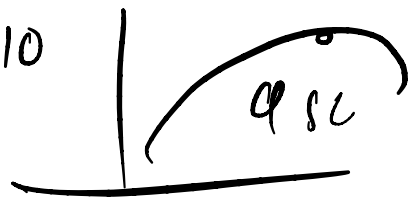
2-D Optimization

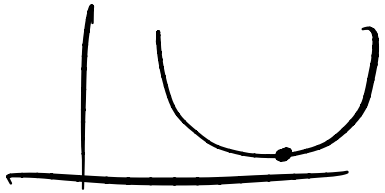
Rishita

20 × 20 ⇒ 400



0.4 0.001
0.2 0.0001
1.2 1.5





init α
↓
10, $\frac{\alpha}{10}$

Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$

$\alpha \rightarrow$ learning rate

$\eta \rightarrow$

  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

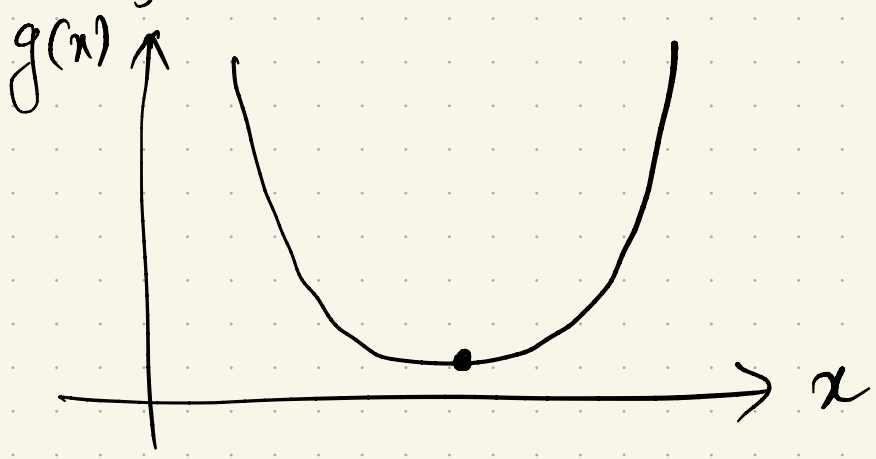
  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

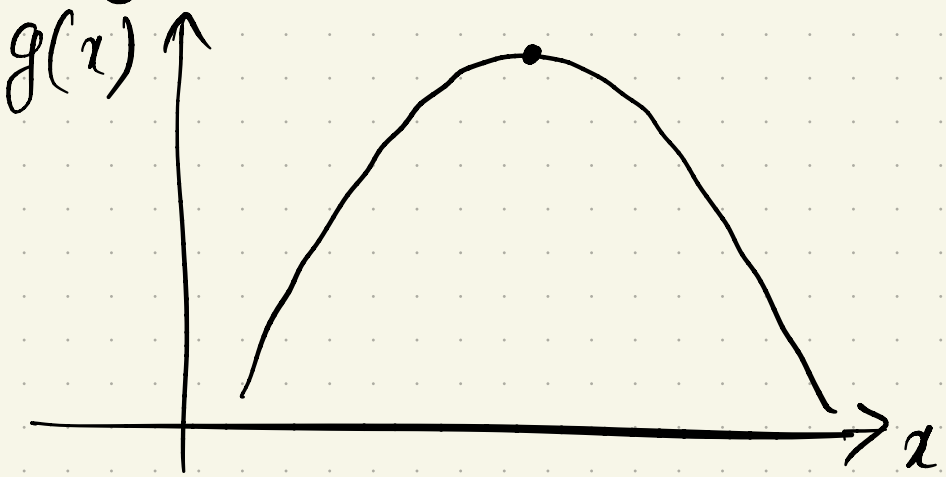
with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ **= gradient**

For **gradient descent**, the sign changes to – in weight update equation.

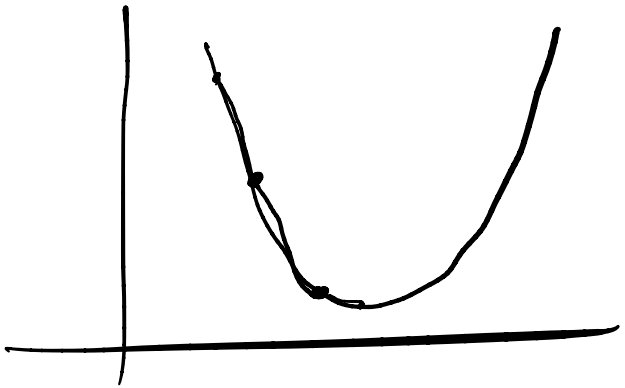
gradient descent



gradient ascent



Gradient Ascent



- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction

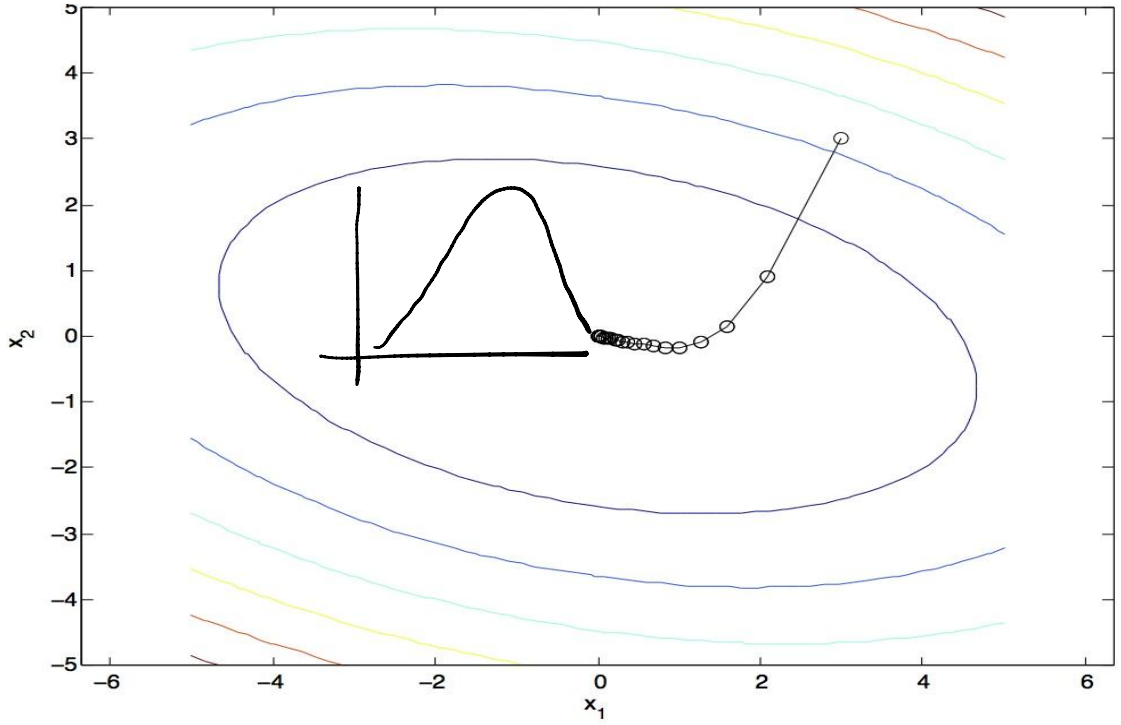


Figure source: Mathworks

Gradient in n dimensions

n features

$$z = x_0 w_0 + x_1 w_1 + x_2 w_2 + \dots x_n w_n$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \dots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

$\frac{\partial g}{\partial w_1} \rightarrow$

$\frac{\partial g}{\partial w_2} \rightarrow$

$\frac{\partial g}{\partial w_n} \rightarrow$

$$w \leftarrow w + \alpha \cdot \frac{\partial g}{\partial w}$$

batch gradient
stochastic gradient
mini-batch gradient

Optimization Procedure: Gradient Ascent

- `init` w
- `for iter = 1, 2, …`

$$w \leftarrow w + \alpha * \nabla g(w)$$

- α : learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

itr = 10

mini batch = 10

samp = 100 → each → sto.

→ once → batch

→ 100 times → mini batch

Batch Gradient Ascent on the Log Likelihood Objective

5000 → 500 × 500 = 250 000

100

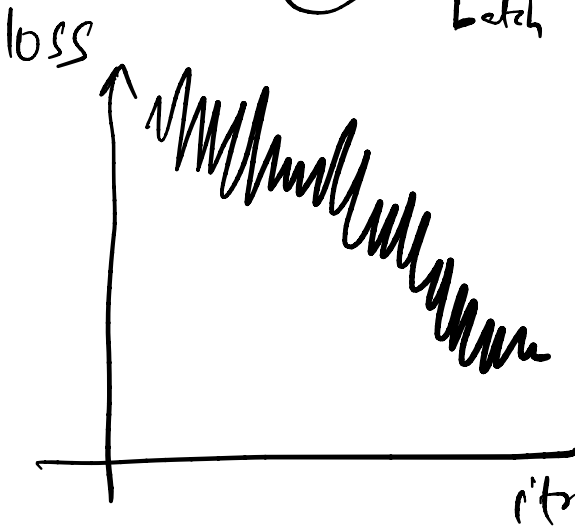
10 40 41

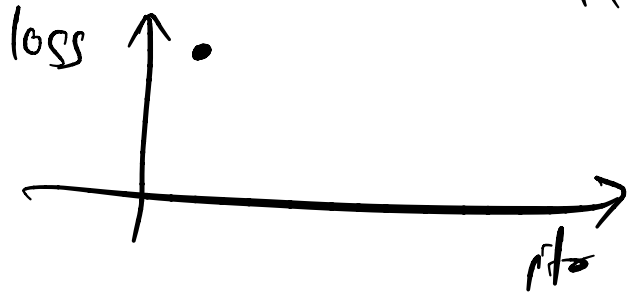
$$\max_w \; ll(w) = \max_w \; \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

100 → 10 → mini-batch

- init w
- for iter = 1, 2, …

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)};w)$$





Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: once gradient on one training example has been computed, might as well incorporate before computing next one

- `init` w
- `for iter = 1, 2, …`
  - `pick random j`

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

neural network sk learn (MLP)

parameter. → w, b
hyperpara. → α, ε, min-batch, epi.

Mini-Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init w
- for iter = 1, 2, …
  - pick random subset of training examples J

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Mini-Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init w
- for iter = 1, 2, …
  - pick random subset of training examples J

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

$x_0 = 1$ $X \longrightarrow x_1, x_2$ {logistic regression} (gradient descent)

$$z = x_0 w_0 + x_1 w_1 + x_2 w_2$$

$$A = \frac{1}{1 + e^{-z}}$$

$$loss = -\left[y \log A + (1-y) \log (1-A) \right] \Rightarrow \text{binary loss}$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial A} \frac{\partial A}{\partial z} \cdot \frac{\partial z}{\partial w}$$

$$w_1 \leftarrow w_1 - \alpha \left(\frac{\partial loss}{\partial w_1} \right)$$

$$w_2 \leftarrow w_2 - \alpha \left(\frac{\partial loss}{\partial w_2} \right)$$

$$w_0 \leftarrow w_0 - \alpha \left(\frac{\partial loss}{\partial w_0} \right)$$