Neural Networks

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

February 22, 2023

Most slides have been adapted from Bhiksha Raj, 11-785, CMU 2020, Justin Johnson, EECS 498-007, University of Michigan 2020, and Fei Fei Li, cs231n, Stanford 2017

Today's agenda

Recap

Supervised learning

Given: a dataset that contains N samples

$$\{(x^{(1)}, y^{(1)}), ..., (x^{(N)}, y^{(N)})\} \tag{1}$$

Regression

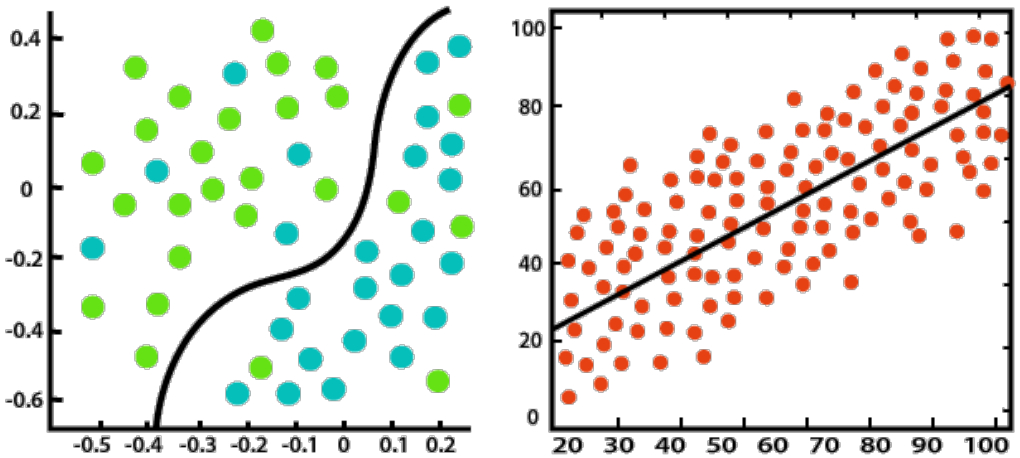
predict a continuous target variable

$$E.g., y \in [0, 1]$$

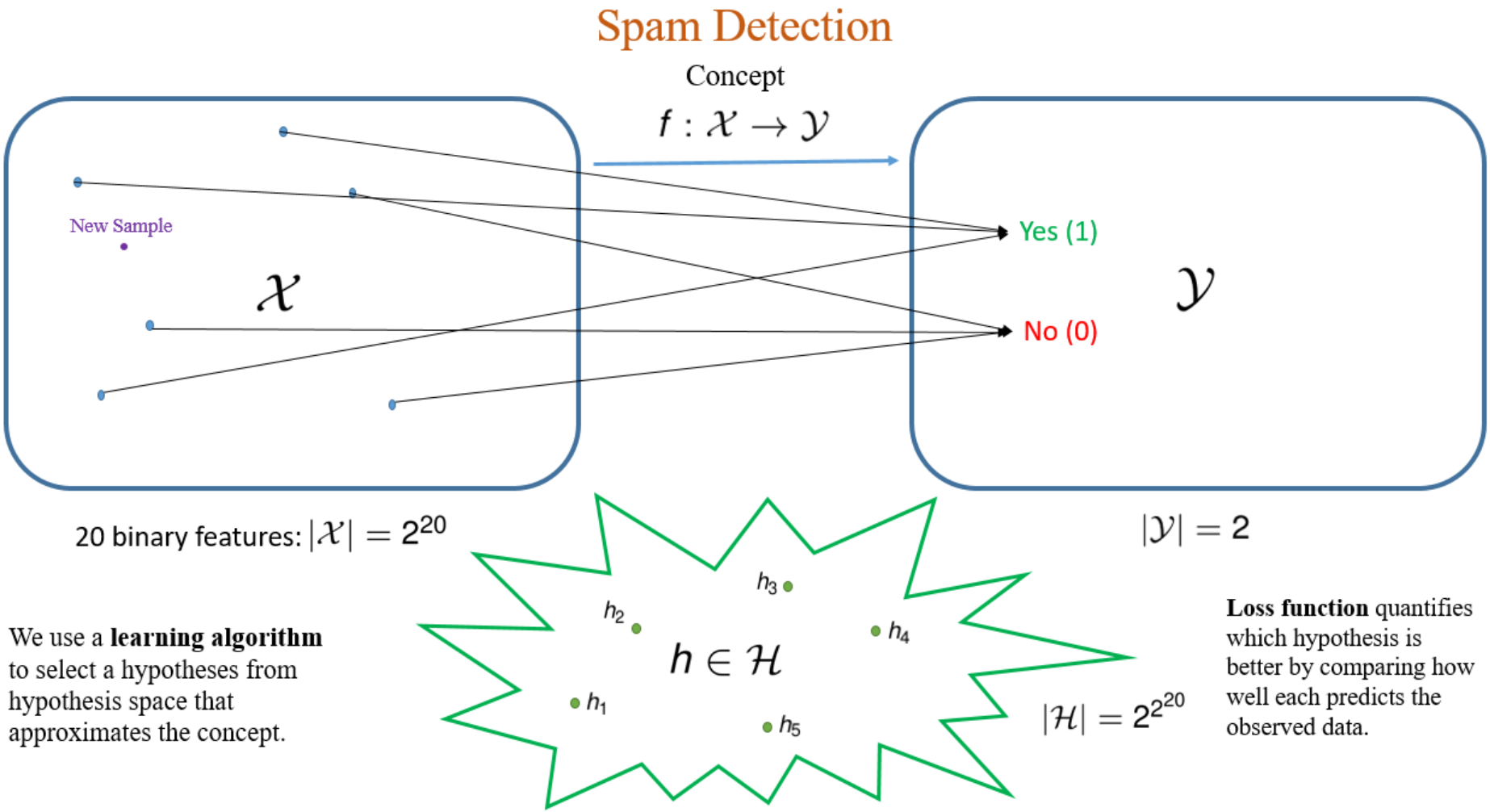
Classification

predict a discrete (unordered) target variable

$$E.g., y \in \{0, 1, 2, ..., 9\}$$



Components of supervised learning (function approximation)



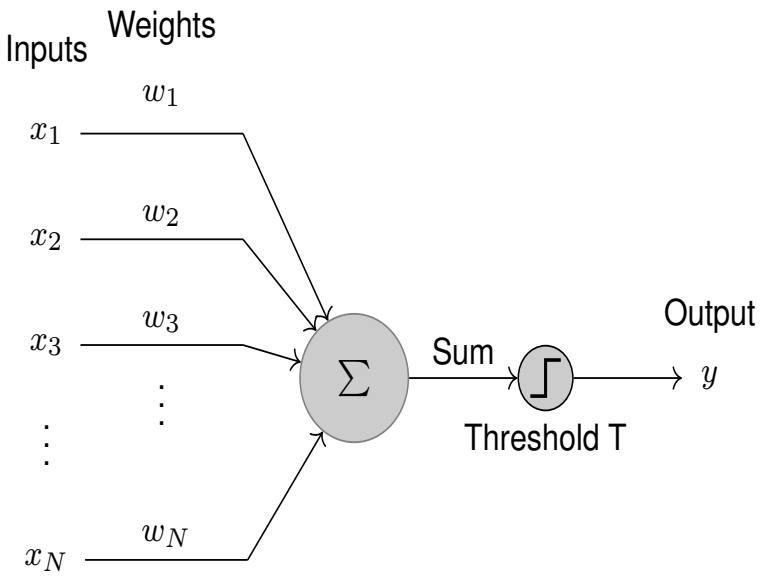
We use a **learning algorithm** to select a hypotheses from hypothesis space that approximates the concept.

Loss function quantifies which hypothesis is better by comparing how well each predicts the observed data.

Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Perceptron

A perceptron is a linear classifier

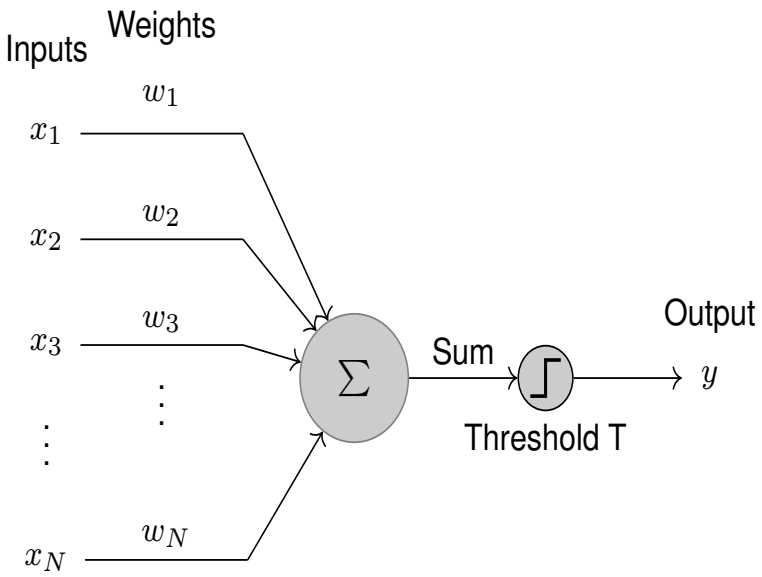


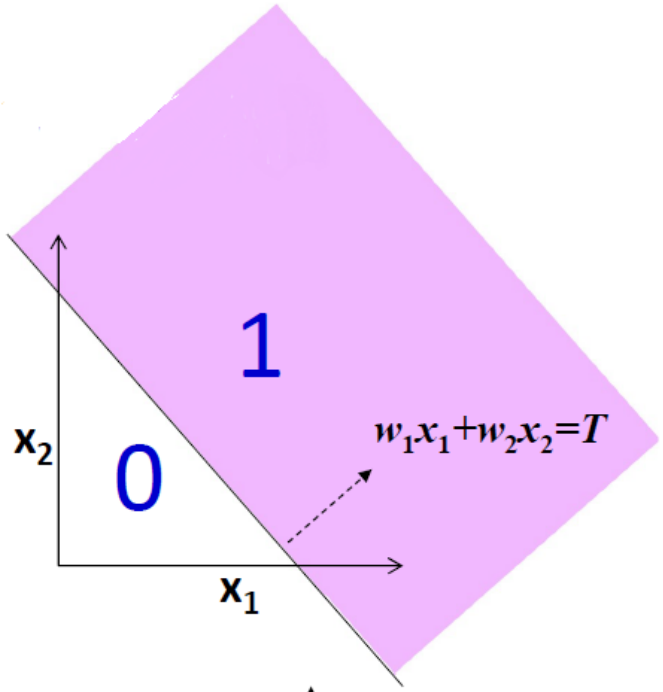
$$y = \theta(\sum_i w_i x_i - T)$$

$$\theta(z) = \begin{cases} 1 & \text{if} \quad z \geq 0 \\ 0 & \text{else} \end{cases}$$

Perceptron

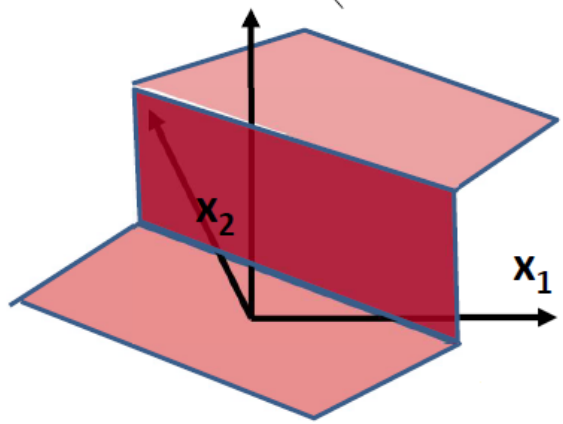
A perceptron is a linear classifier



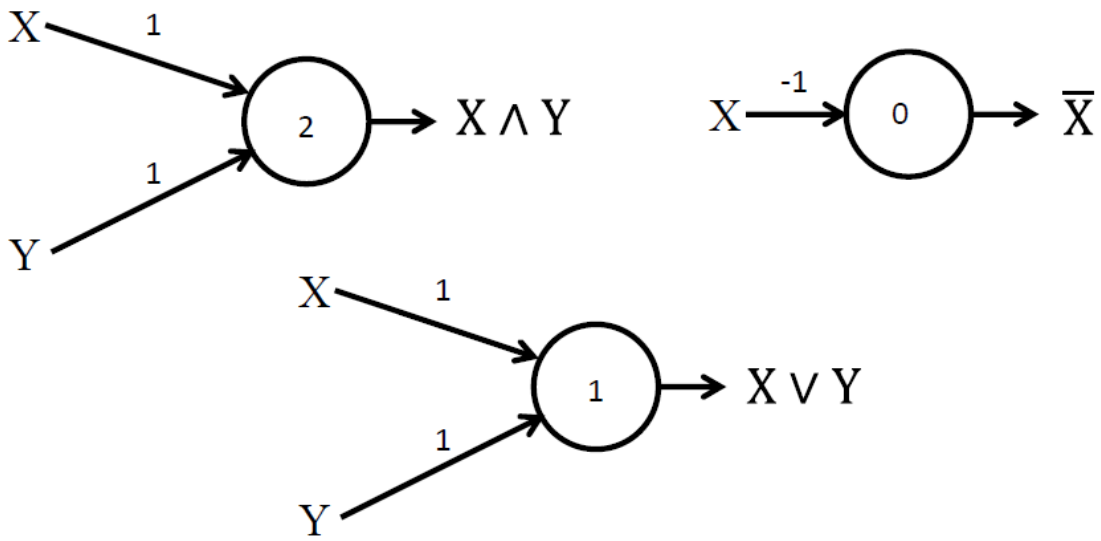


$$y = \theta(\sum_i w_i x_i - T)$$

$$\theta(z) = \begin{cases} 1 & \text{if} \quad z \geq 0 \\ 0 & \text{else} \end{cases}$$



Perceptron

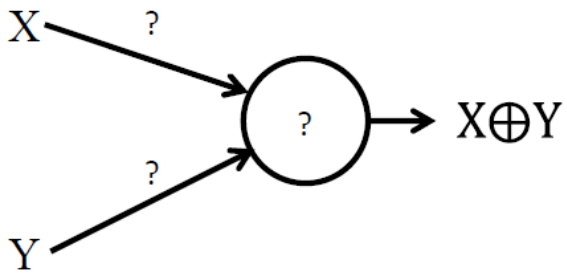


Values shown on edges are weights, numbers in the circles are thresholds ($X, Y \in \{0, 1\}$)

- Easily shown to mimic any Boolean gate
- But ...

Perceptron

No solution for XOR!
Not universal!



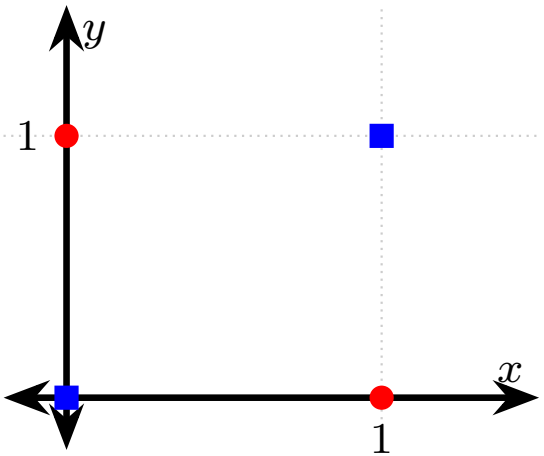
Values shown on edges are weights, numbers in the circles are thresholds ($X, Y \in \{0, 1\}$)

- Easily shown to mimic any Boolean gate
- But ...

Proof

X	Y	Label
0	0	0
0	1	1
1	0	1
1	1	0

XOR



Suppose there is a linear classifier $w_1 x + w_2 y - T = 0$ that can correctly classify this dataset, we have

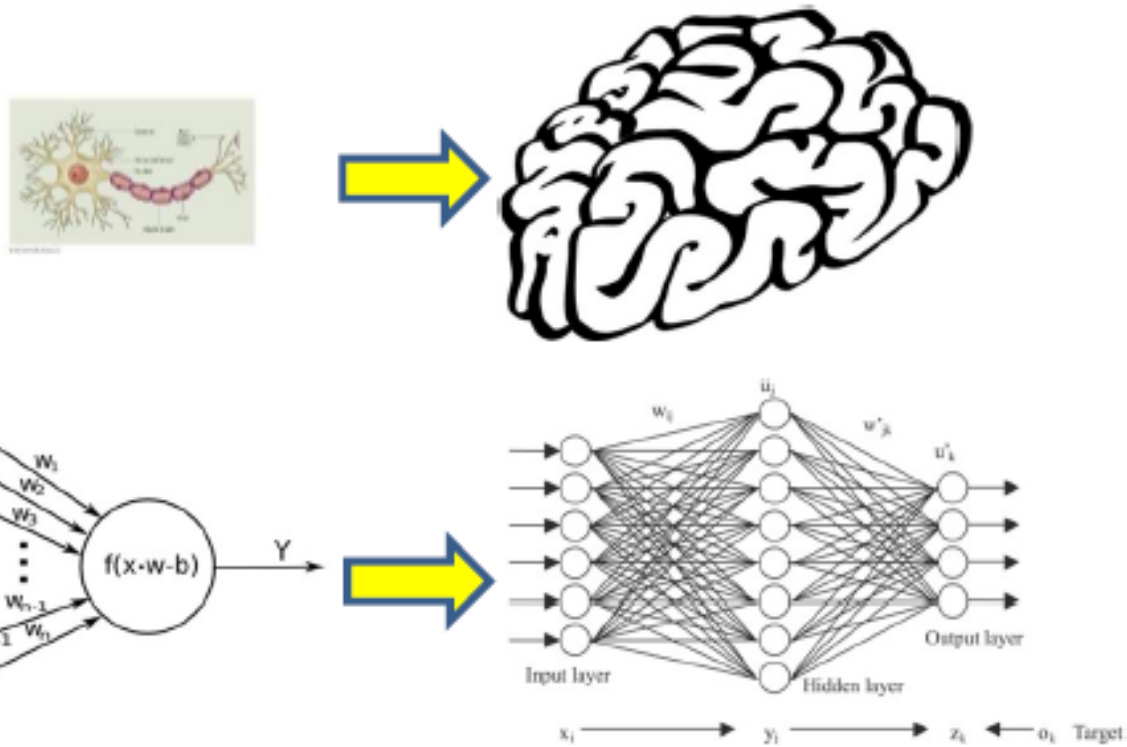
$$(1) \ -T < 0$$
$$(2) \ w_2 - T \geq 0$$
$$(3) \ w_1 - T \geq 0$$
$$(4) \ w_1 + w_2 - T < 0$$

$$(2), (3) \Rightarrow w_1 + w_2 - 2T \geq 0 \ (5)$$
$$(4), (5) \Rightarrow -T > 0 \ (6)$$

But (6) contradicts (1). Therefore our assumption must be incorrect.

A single neuron is not enough

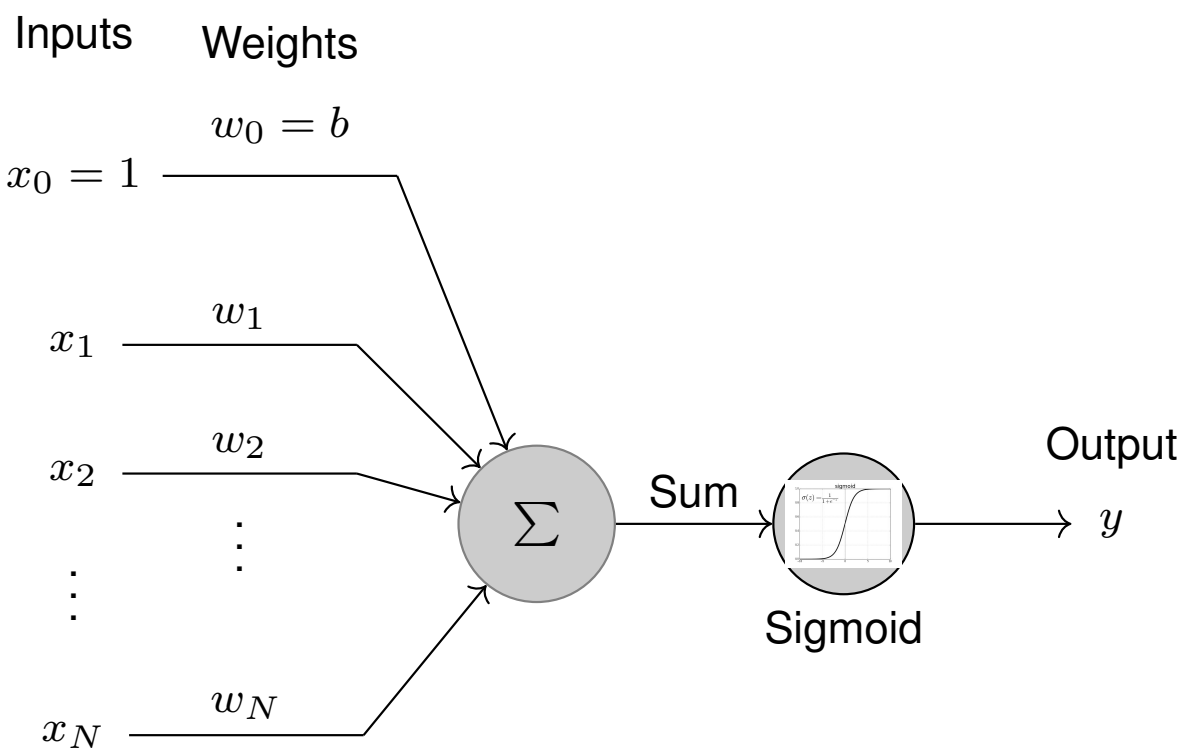
- Individual elements are weak computational elements
  - Marvin Minsky and Seymour Papert, 1969, Perceptrons: An Introduction to Computational Geometry
- Networked elements are required



Logistic regression

It is the perceptron with a sigmoid activation

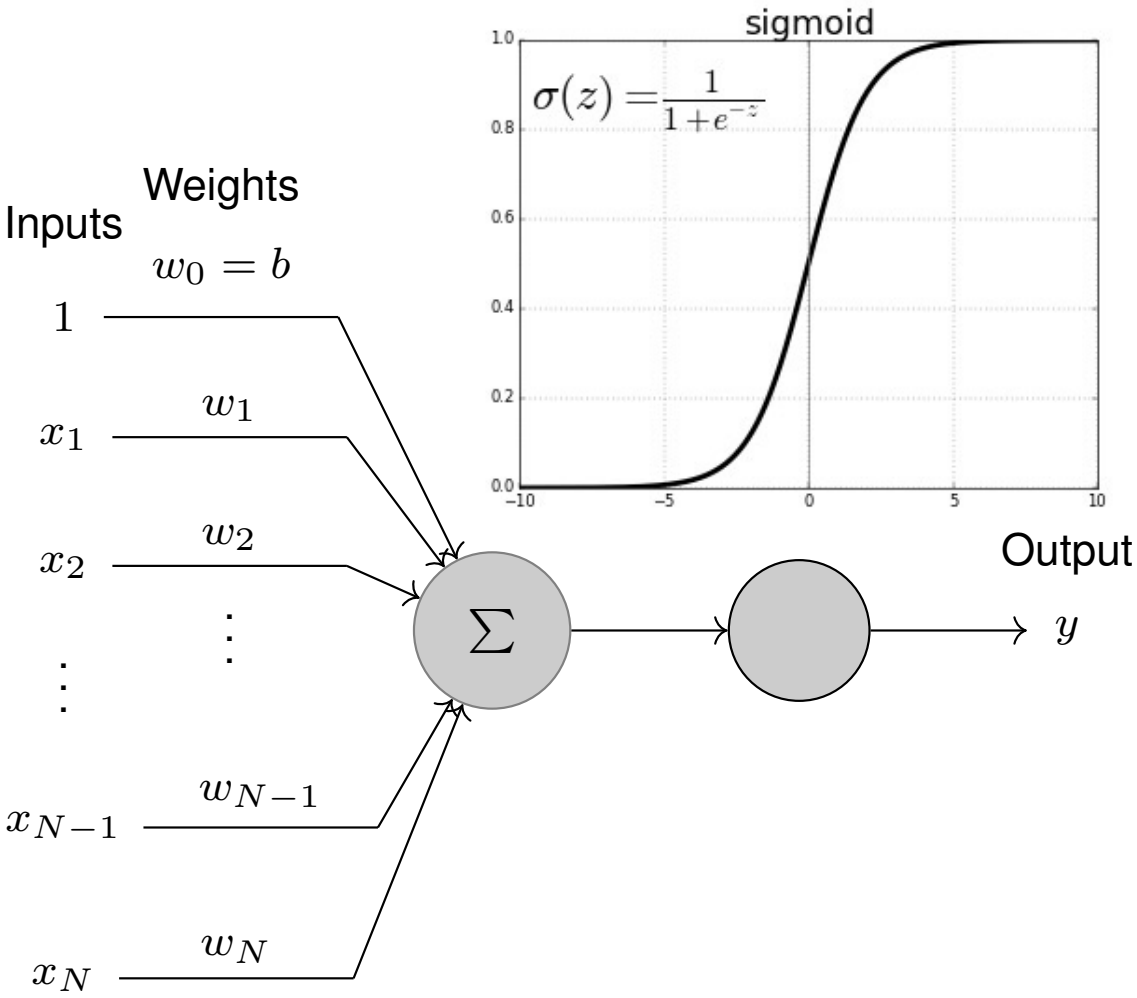
- It actually computes the probability that the input belongs to class 1



Logistic regression

It is the perceptron with a sigmoid activation

- It actually computes the probability that the input belongs to class 1



$$z = \sum_i w_i x_i$$

$$p(y = 1|x) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$y = \begin{cases} 1 & \text{if} \quad \sigma(z) \geq \frac{1}{2} \\ 0 & \text{if} \quad \sigma(z) < \frac{1}{2} \end{cases}$$

Vectorization

$$z = \sum_i w_i x_i = \boldsymbol{w}^T \boldsymbol{x} = \begin{bmatrix} w_0 = b & w_1 & w_2 & \dots & w_{N-1} & w_N \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.
- A loss function tells how good our current classifier is
  - Given a dataset of examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $x^{(i)}$ is data and $y^{(i)}$ is binary label.
- Loss over the dataset is a average of loss over examples:

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_{i} L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

- Empirical loss L is only an empirical approximation of the true loss
- **Learning**
  - Estimate the parameters to minimize the empirical loss L

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ L(f(X, \boldsymbol{w}), \boldsymbol{y})$$

Cross entropy

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x , we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

$$D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x , we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

$$D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- It is the extra amount of information needed to send a message containing symbols drawn from probability distribution P , when we use a code that was designed to minimize the length of messages drawn from probability distribution Q.

Cross entropy

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x , we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

$$D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- It is the extra amount of information needed to send a message containing symbols drawn from probability distribution P , when we use a code that was designed to minimize the length of messages drawn from probability distribution Q.
- KL divergence is non-negative and measures the difference between two distributions, it is often conceptualized as measuring some sort of distance between these distributions. However, it is not a true distance measure because it is not symmetric

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable x and it is fixed.

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable x and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}} - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable x and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}} - \sum_i p_i(x) \log q_i(x)$$

$$CrossEntropy(P(x), Q(x)) = - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable x and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}} - \sum_i p_i(x) \log q_i(x)$$

$$CrossEntropy(P(x), Q(x)) = - \sum_i p_i(x) \log q_i(x)$$

Example

The label of data x is 0. The output of Logistic Regression for x is 0.7

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}}\ D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable x and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}}\ - \sum_i p_i(x) \log q_i(x)$$

$$CrossEntropy(P(x), Q(x)) = - \sum_i p_i(x) \log q_i(x)$$

Example

The label of data x is 0. The output of Logistic Regression for x is 0.7

$$\text{True Dist. } P(x) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad -- \quad \text{Predicted Dist. } Q(x) = \begin{bmatrix} 1 - 0.7 \\ 0.7 \end{bmatrix}$$

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}}\ D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable x and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}}\ - \sum_i p_i(x) \log q_i(x)$$

$$CrossEntropy(P(x), Q(x)) = - \sum_i p_i(x) \log q_i(x)$$

Example

The label of data x is 0. The output of Logistic Regression for x is 0.7

$$\text{True Dist. } P(x) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad -- \quad \text{Predicted Dist. } Q(x) = \begin{bmatrix} 1 - 0.7 \\ 0.7 \end{bmatrix}$$

$$CrossEntropy(P(x), Q(x)) = -1 \times \log(1 - 0.7) - 0 \times \log 0.7 = 1.2$$

Logistic regression loss function

- Logistic Regression

$$p(y = 1|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$p(y = 0|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = 1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$y^{(i)} \in \{0, 1\}$$

- Cross entropy as the loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

Logistic regression loss function

- Logistic Regression

$$p(y = 1|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$p(y = 0|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = 1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$y^{(i)} \in \{0, 1\}$$

- Cross entropy as the loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

How do we find the best w?

Negative Log Likelihood

Suppose we have training dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_N, y_N)\}$. One way of choosing weight vector $\boldsymbol{w}$ is to choose the weights that maximize the likelihood (or minimize the negative log likelihood) of the observed data (training set).

$$\underset{w}{\text{Maximize}} \quad p(D; \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p(\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_N, y_N)\}; \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p((\boldsymbol{x}_1, y_1); \boldsymbol{w}) p((\boldsymbol{x}_2, y_2); \boldsymbol{w}) ... p((\boldsymbol{x}_N, y_N); \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p(y_1|\boldsymbol{x}_1, \boldsymbol{w}) p(\boldsymbol{x}_1; \boldsymbol{w}) p(y_2|\boldsymbol{x}_2, \boldsymbol{w}) p(\boldsymbol{x}_2; \boldsymbol{w}) ... p(y_N|\boldsymbol{x}_N, \boldsymbol{w}) p(\boldsymbol{x}_N; \boldsymbol{w})$$

Since $p(\boldsymbol{x}_i; \boldsymbol{w})$ is constant, it can be removed from the objective function.

Negative Log Likelihood

Suppose we have training dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_N, y_N)\}$. One way of choosing weight vector $\boldsymbol{w}$ is to choose the weights that maximize the likelihood (or minimize the negative log likelihood) of the observed data (training set).

$$\underset{w}{\text{Maximize}} \quad p(D; \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p(\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_N, y_N)\}; \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p((\boldsymbol{x}_1, y_1); \boldsymbol{w}) p((\boldsymbol{x}_2, y_2); \boldsymbol{w}) ... p((\boldsymbol{x}_N, y_N); \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p(y_1|\boldsymbol{x}_1, \boldsymbol{w}) p(\boldsymbol{x}_1; \boldsymbol{w}) p(y_2|\boldsymbol{x}_2, \boldsymbol{w}) p(\boldsymbol{x}_2; \boldsymbol{w}) ... p(y_N|\boldsymbol{x}_N, \boldsymbol{w}) p(\boldsymbol{x}_N; \boldsymbol{w})$$

Since $p(\boldsymbol{x}_i; \boldsymbol{w})$ is constant, it can be removed from the objective function.

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad p(y_1|\boldsymbol{x}_1, \boldsymbol{w}) p(y_2|\boldsymbol{x}_2, \boldsymbol{w}) ... p(y_N|\boldsymbol{x}_N, \boldsymbol{w})$$

$$\underset{\boldsymbol{w}}{\text{Maximize}} \quad \Pi_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = \Pi_{i=1}^{N} (f(\boldsymbol{x}_i, \boldsymbol{w}))^{y_i} (1 - f(\boldsymbol{x}_i, \boldsymbol{w}))^{(1-y_i)}$$

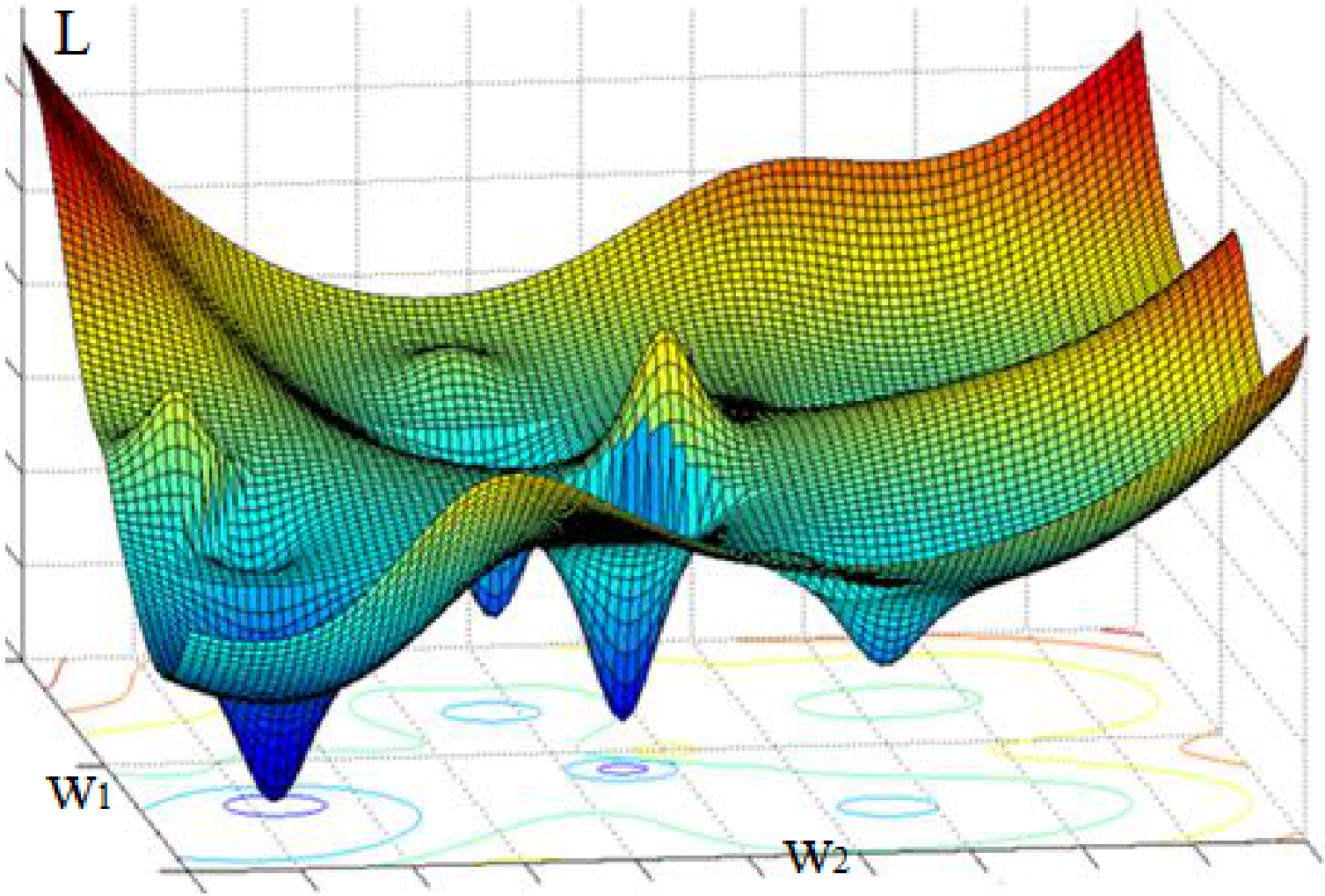
$$\underset{\boldsymbol{w}}{\text{Minimize}} \quad -\log \Pi_{i=1}^{N} (f(\boldsymbol{x}_i, \boldsymbol{w}))^{y_i} (1 - f(\boldsymbol{x}_i, \boldsymbol{w}))^{(1-y_i)}$$

$$\underset{\boldsymbol{w}}{\text{Minimize}} \quad \sum_{i=1}^{N} -y_i \log f(\boldsymbol{x}_i, \boldsymbol{w}) - (1 - y_i) \log(1 - f(\boldsymbol{x}_i, \boldsymbol{w}))$$

Optimization

Optimization

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ L(f(X, \boldsymbol{w}), \boldsymbol{y})$$



Finding the minimum of a function

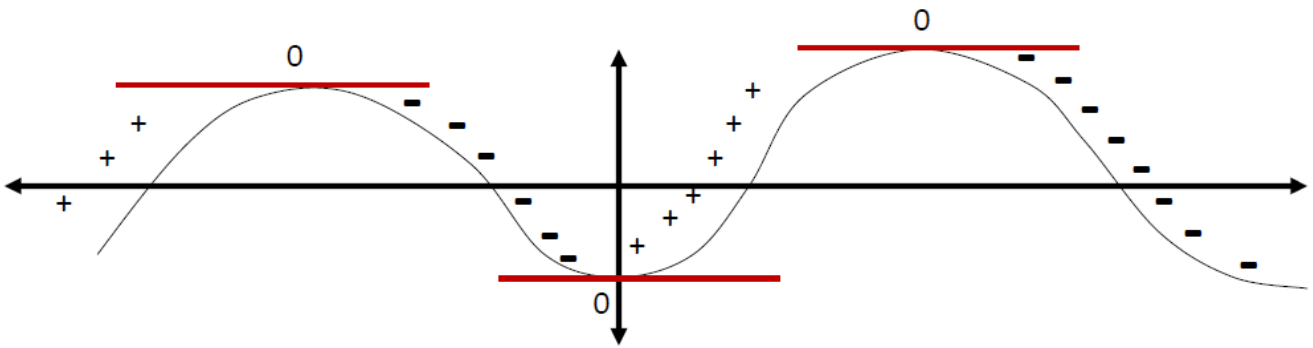
- Solve:

$$\frac{df(x)}{dx} = 0$$

- The solution x^* is a turning point
- Check the double derivative at x^*

$$\frac{d^2 f(x^*)}{dx^{*2}}$$

  - If it is positive x^* is a minimum, otherwise it is a maximum



Finding the minimum of a function

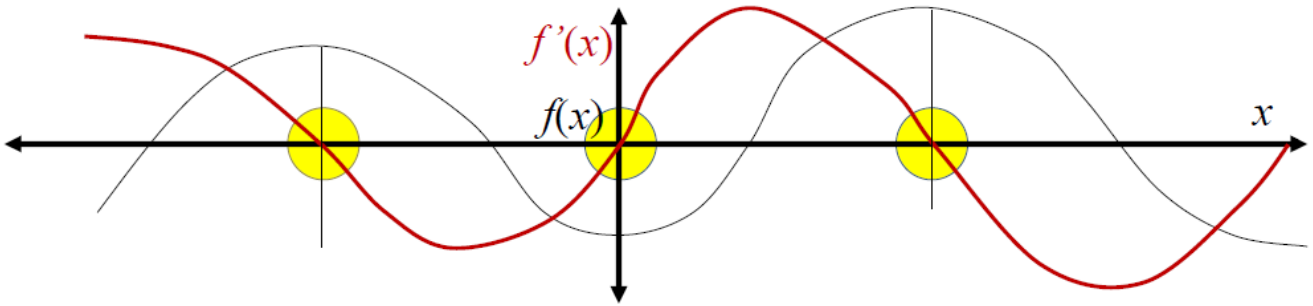
- Solve:

$$\frac{df(x)}{dx} = 0$$

- The solution x^* is a turning point
- Check the double derivative at x^*

$$\frac{d^2 f(x^*)}{dx^{*2}}$$

  - If it is positive x^* is a minimum, otherwise it is a maximum



Finding the minimum of a function

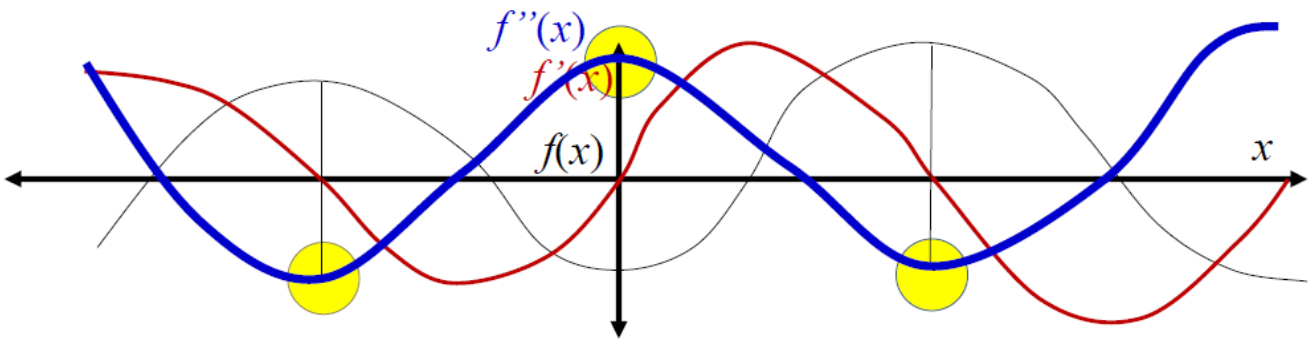
- Solve:

$$\frac{df(x)}{dx} = 0$$

- The solution x^* is a turning point
- Check the double derivative at x^*

$$\frac{d^2 f(x^*)}{dx^{*2}}$$

  - If it is positive x^* is a minimum, otherwise it is a maximum



Finding the minimum of a function

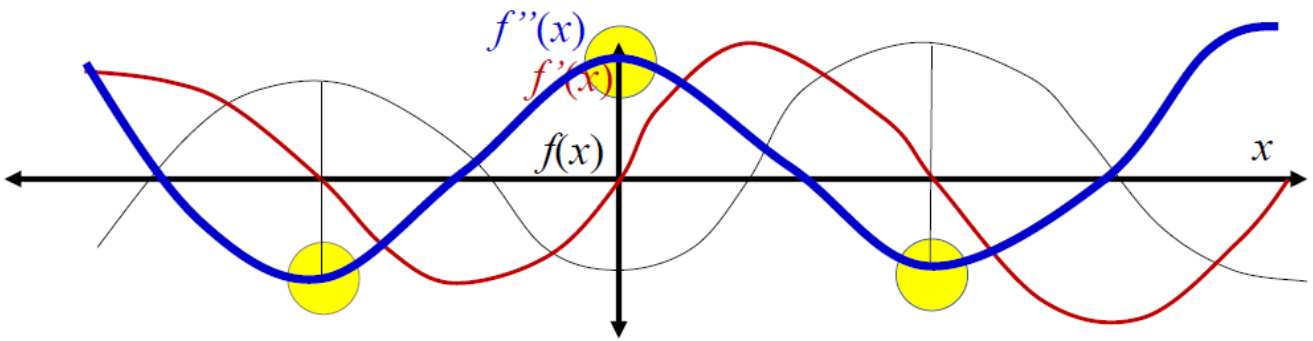
- Solve:

$$\frac{df(x)}{dx} = 0$$

- The solution x^* is a turning point
- Check the double derivative at x^*

$$\frac{d^2 f(x^*)}{dx^{*2}}$$

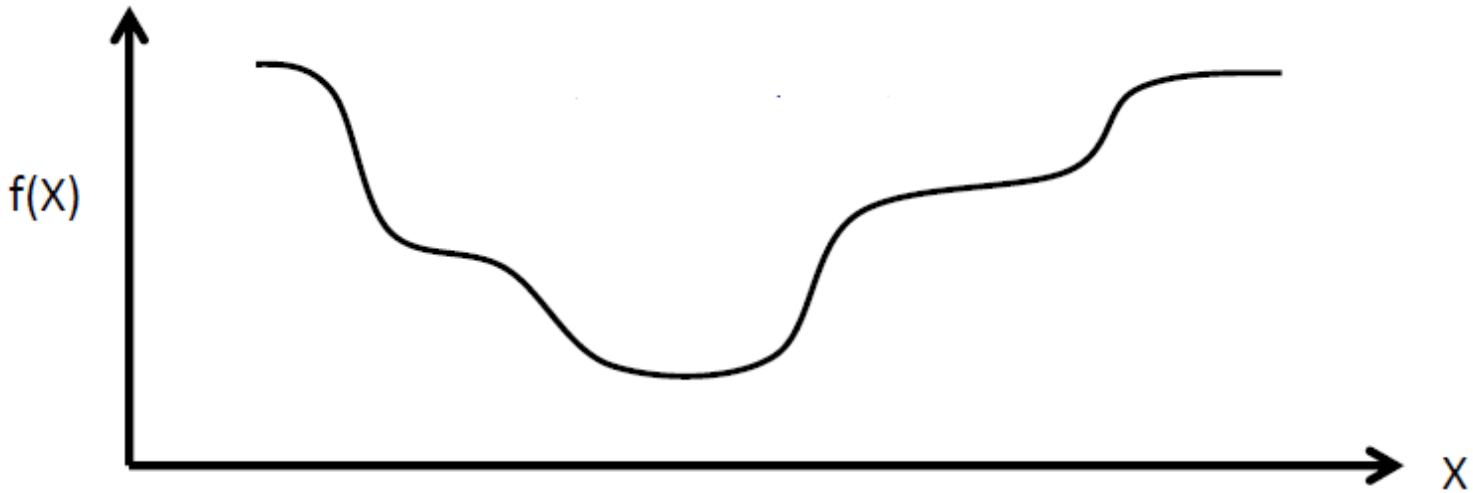
  - If it is positive x^* is a minimum, otherwise it is a maximum



It does not work!!!

Closed form solutions are not always available

- Often it is not possible to simply solve $\nabla_X f(X) = 0$
  - The function to minimize/maximize may have an intractable form
- In these situations, iterative solutions are used
  - Begin with a "guess" for the optimal X and refine it iteratively until the correct value is obtained



Follow the slope

Follow the slope

Follow the slope

Derivative

- In 1-dimension, the derivative of a function gives the slope:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- In multiple dimensions, the gradient is the vector of (partial derivatives) along each dimension
- The direction of steepest descent is the negative gradient

- To decrease f at x:

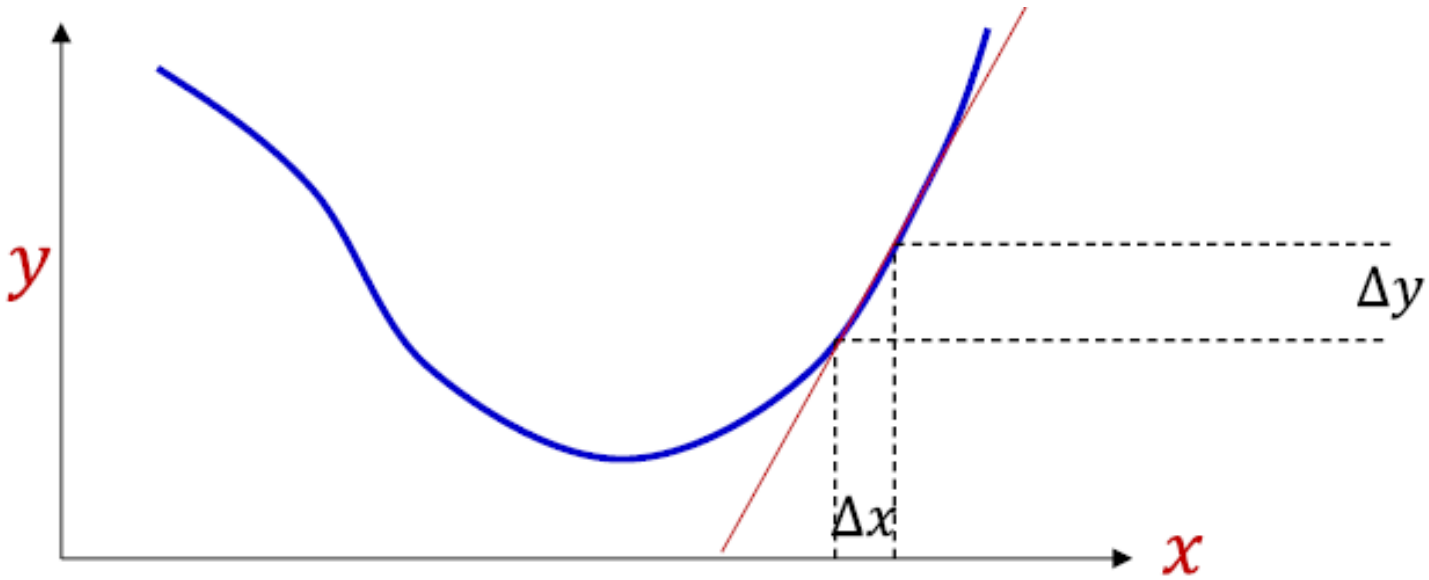
$$if \ \frac{df(x)}{dx} < 0 \Rightarrow f(x+h) < f(x) \Rightarrow f \text{ is decreasing in } x \Rightarrow \text{take a step in } \textbf{positive} \text{ direction}$$

$$if \ \frac{df(x)}{dx} > 0 \Rightarrow f(x+h) > f(x) \Rightarrow f \text{ is increasing in } x \Rightarrow \text{take a step in } \textbf{negative} \text{ direction}$$

  - Take a step in the reverse direction of derivative at x.

A brief note on derivatives...

- A derivative of a function at any point tells us how much a minute increment to the argument of the function will increment the value of the function
  - For any $y = f(x)$, expressed as a multiplier α to a tiny increment Δx to obtain the increments Δy to the output
    $$\Delta y = \alpha \Delta x$$
  - Based on the fact that at a fine enough resolution, any smooth, continuous function is locally linear at any point



Multivariate scalar function

- Scalar function of vector argument
  $y = f(\boldsymbol{x})$ $(f : \mathbb{R}^d \to \mathbb{R})$

$$\Delta y = \alpha \Delta \boldsymbol{x}$$

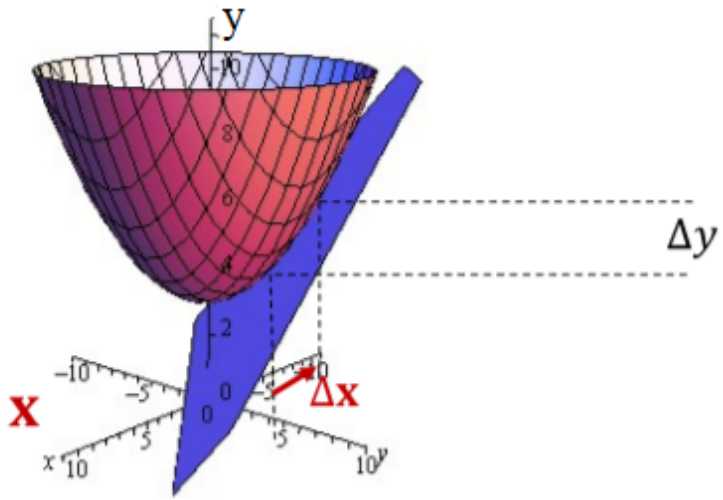
Multivariate scalar function

- Scalar function of vector argument
  $y = f(\boldsymbol{x})$ $(f : \mathbb{R}^d \rightarrow \mathbb{R})$

$$\Delta y = \alpha \Delta \boldsymbol{x}$$

- Δx is now a vector

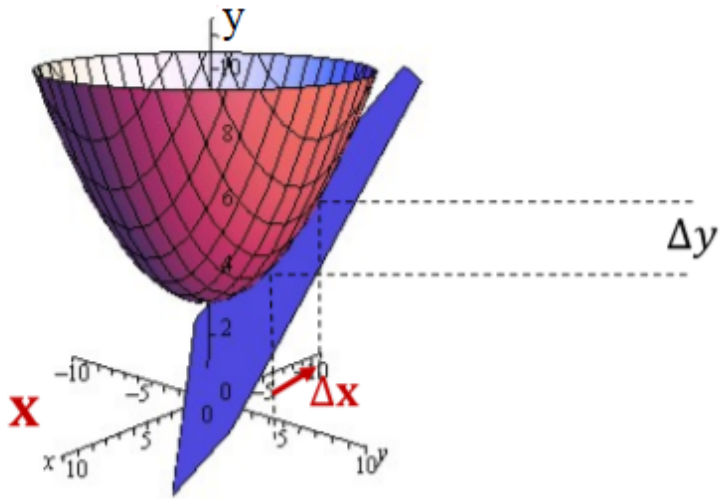
$$\Delta \boldsymbol{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_d \end{bmatrix}$$



Multivariate scalar function

- Scalar function of vector argument
  $y = f(\boldsymbol{x})$ $(f : \mathbb{R}^d \rightarrow \mathbb{R})$

$$\Delta y = \alpha \Delta \boldsymbol{x}$$



- Δx is now a vector

$$\Delta \boldsymbol{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_d \end{bmatrix}$$

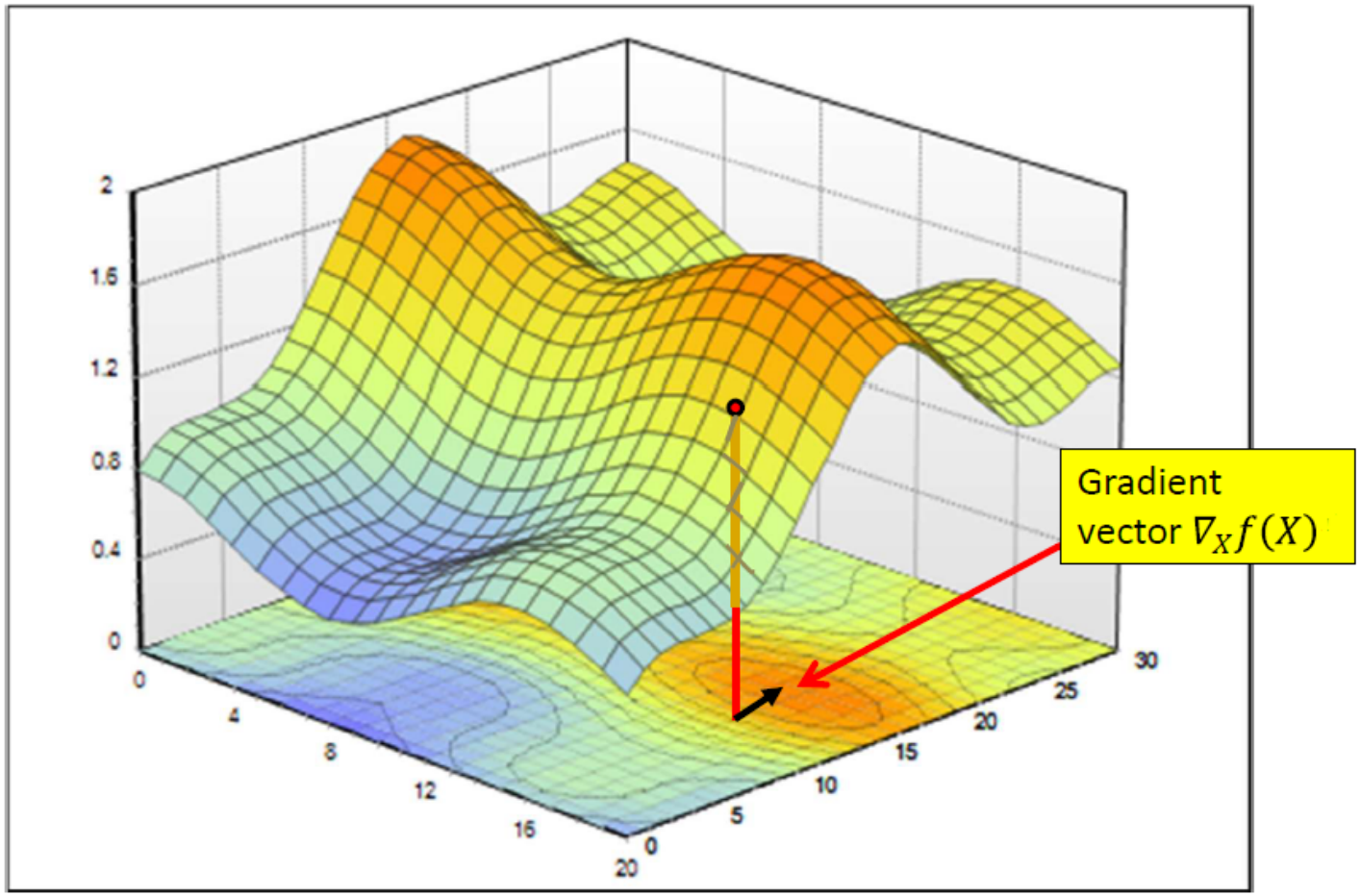
  - Giving us that α is a row vector: $\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_d \end{bmatrix}$

$$\Delta y = \alpha_1 \Delta x_1 + \alpha_2 \Delta x_2 + \dots + \alpha_d \Delta x_d$$

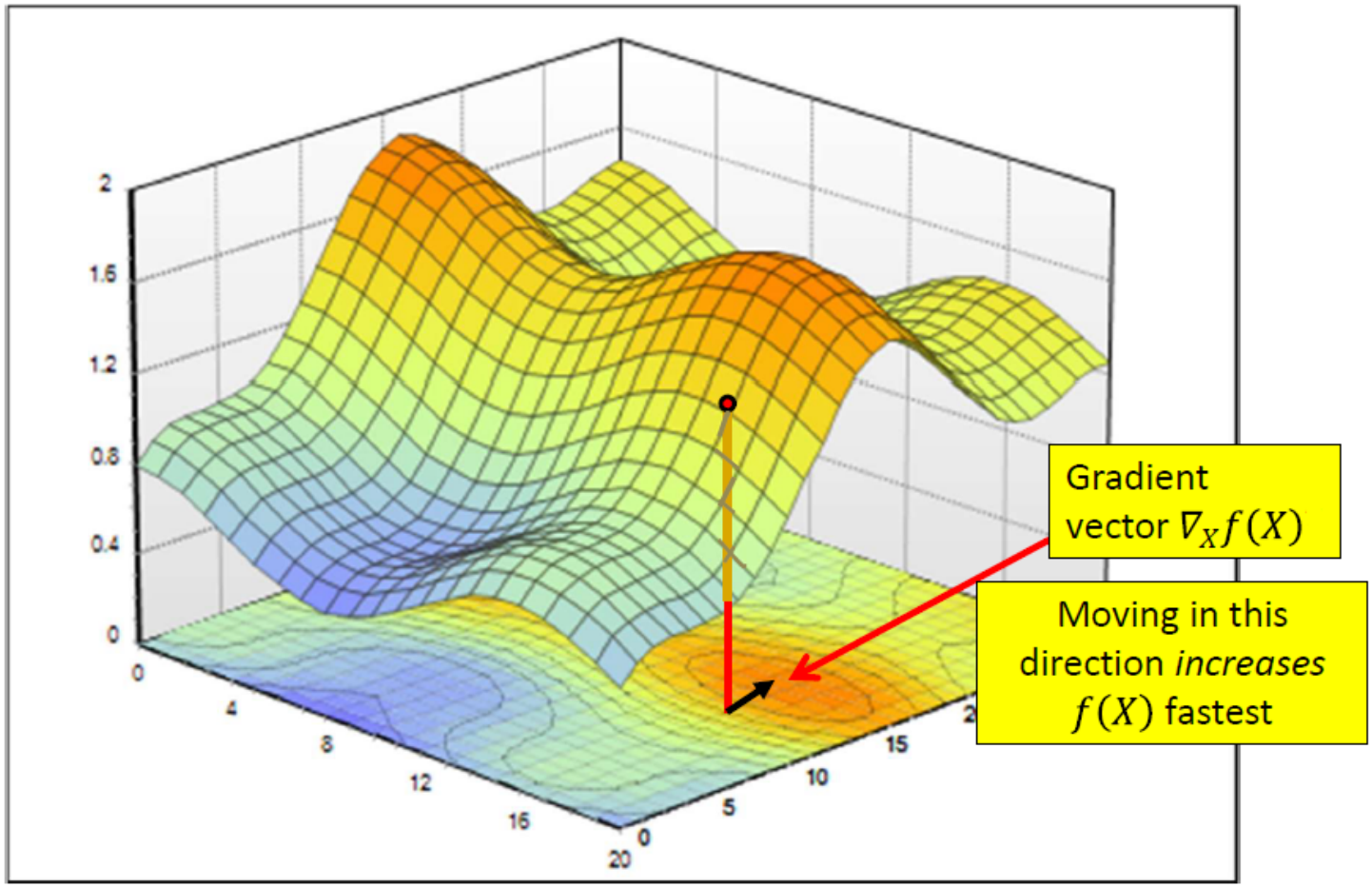
  - The partial derivative α_i gives us how y increments when only x_i is incremented.
  - Often represented as $\frac{\partial y}{\partial x_i}$

$$\Delta y = \frac{\partial y}{\partial x_1} \Delta x_1 + \frac{\partial y}{\partial x_2} \Delta x_2 + \dots + \frac{\partial y}{\partial x_d} \Delta x_d$$

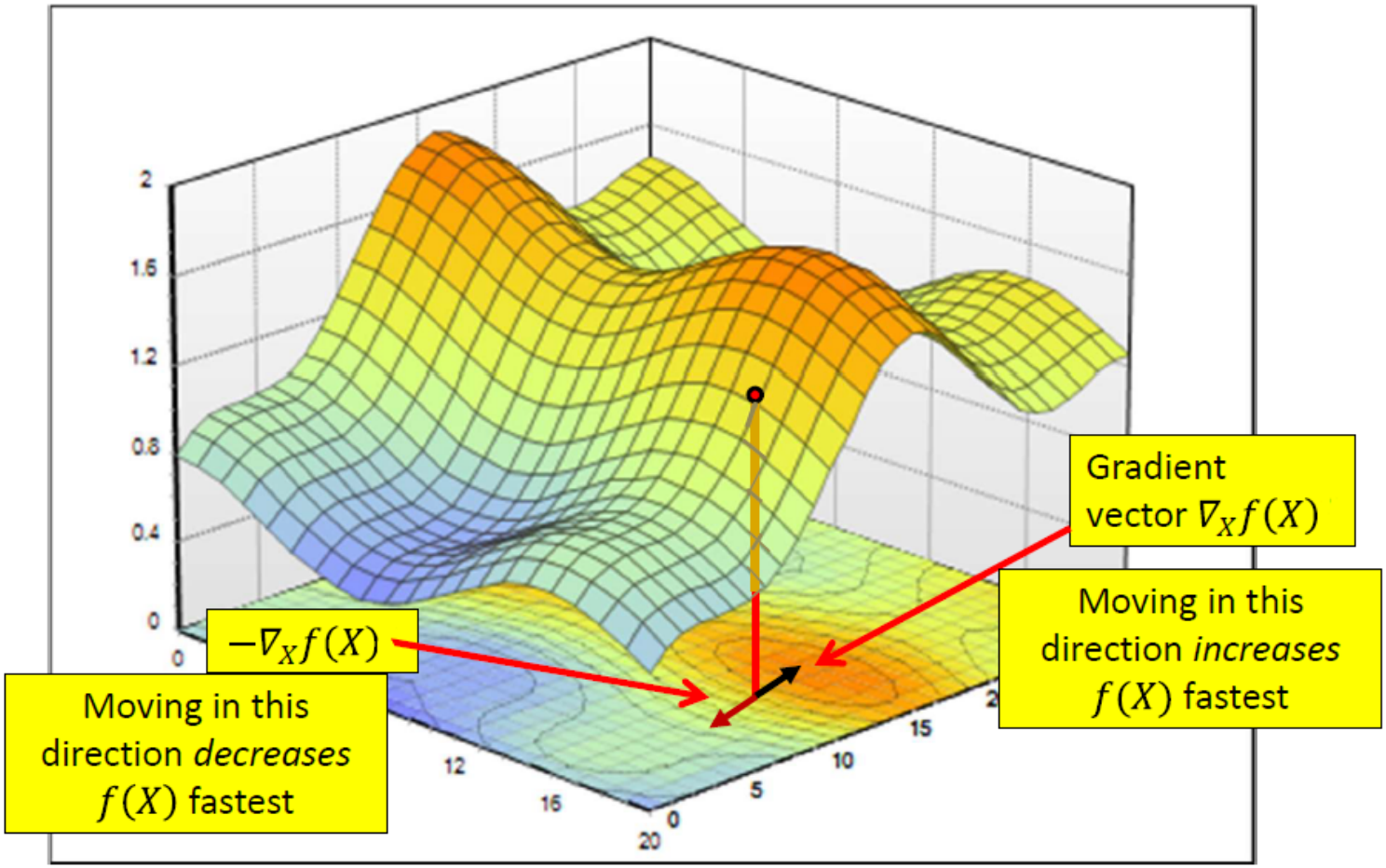
Gradient


2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
20
25
30
Gradient
vector $\nabla_X f(X)$

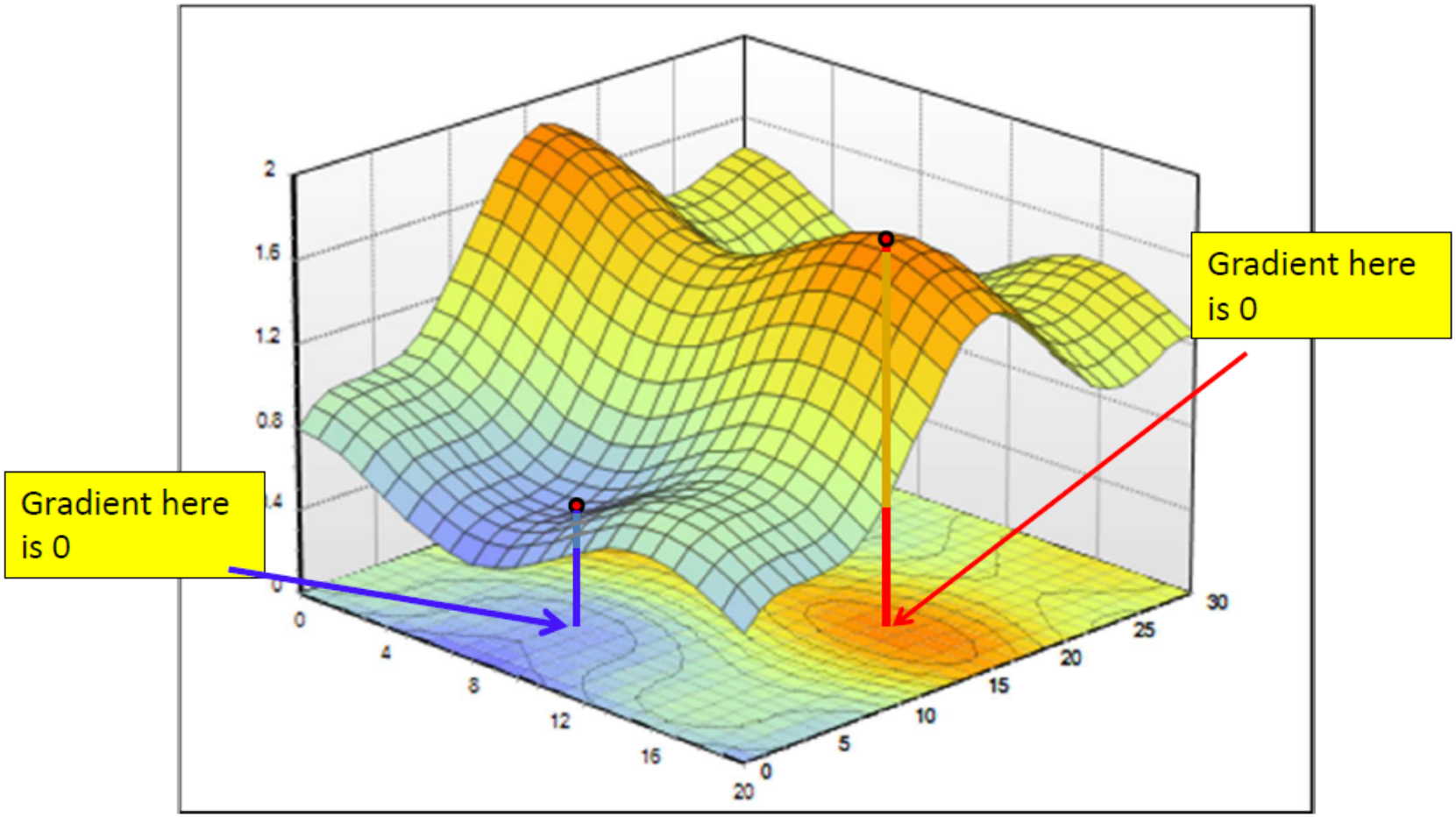
Gradient


2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)$
Moving in this direction *increases* $f(X)$ fastest

Gradient

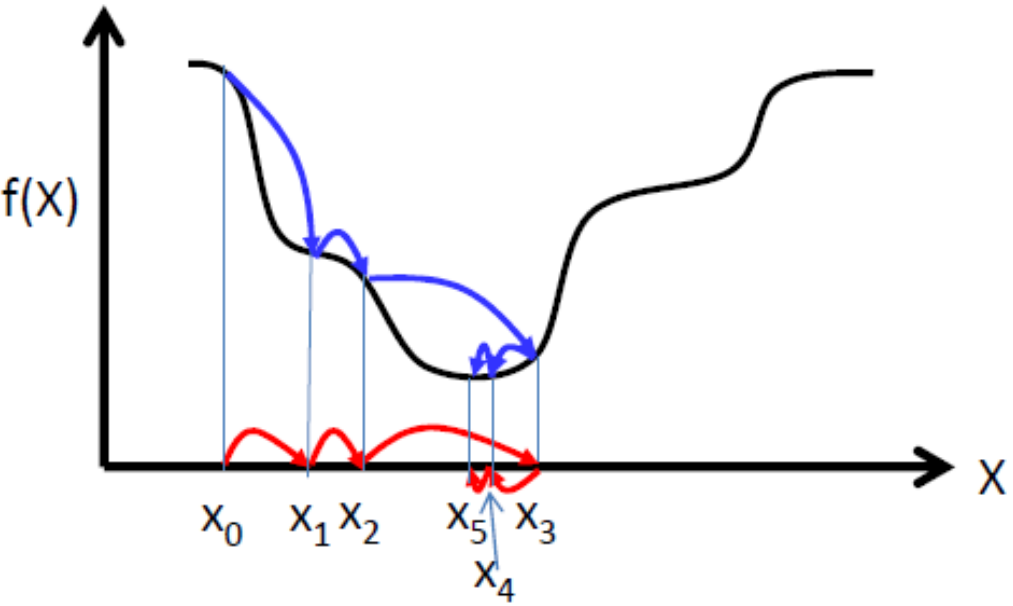

Gradient vector $\nabla_X f(X)$
Moving in this direction *increases* $f(X)$ fastest
$-\nabla_X f(X)$
Moving in this direction *decreases* $f(X)$ fastest
2
1.6
1.2
0.8
0.4
0
0
12
16
20
0
5
10
15

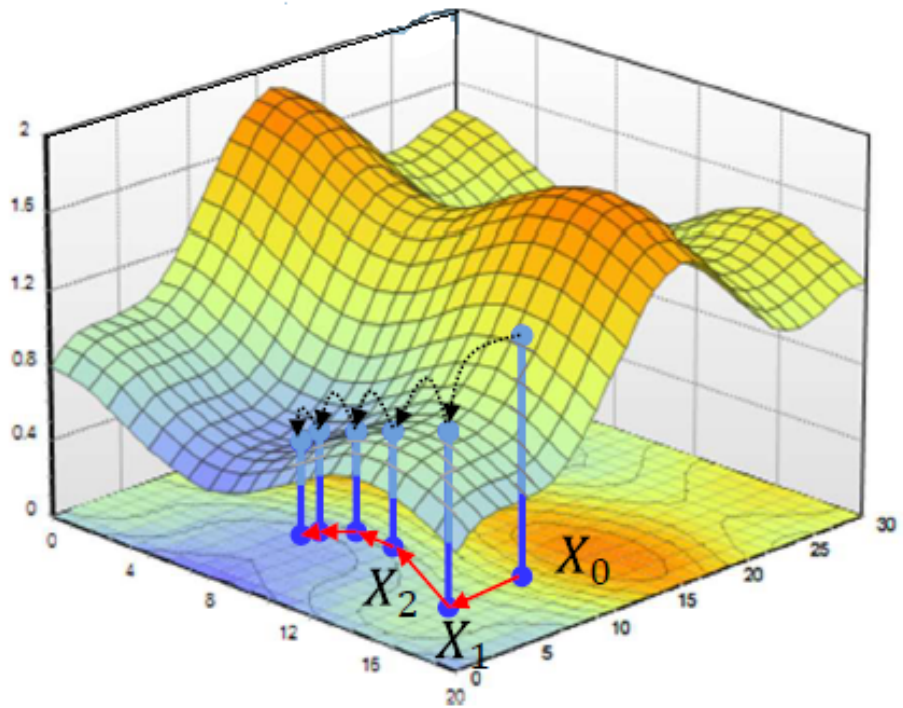
Gradient


Gradient here is 0
Gradient here is 0
2
1.6
1.2
0.8
0
4
8
12
16
20
0
5
10
15
20
25
30

Iterative solutions (Gradient Descent)

- Iterative solutions
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of $f(X)$
  - Stop when $f(X)$ no longer decreases
- Problems
  - Which direction to step in
  - How big must the steps be





Gradient descent

- The gradient descent method to find the minimum of a function iteratively

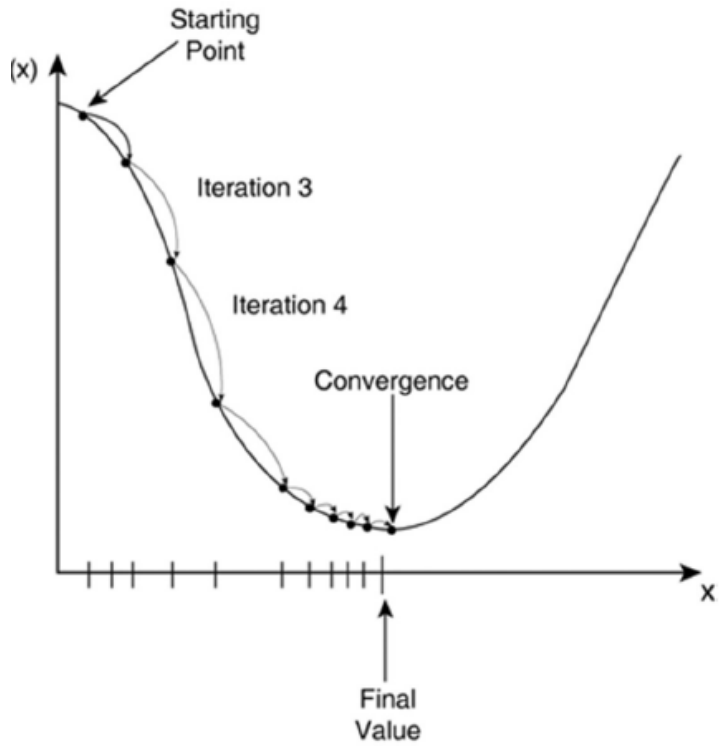
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$$

  - η is the "step size" (Also called "learning rate")

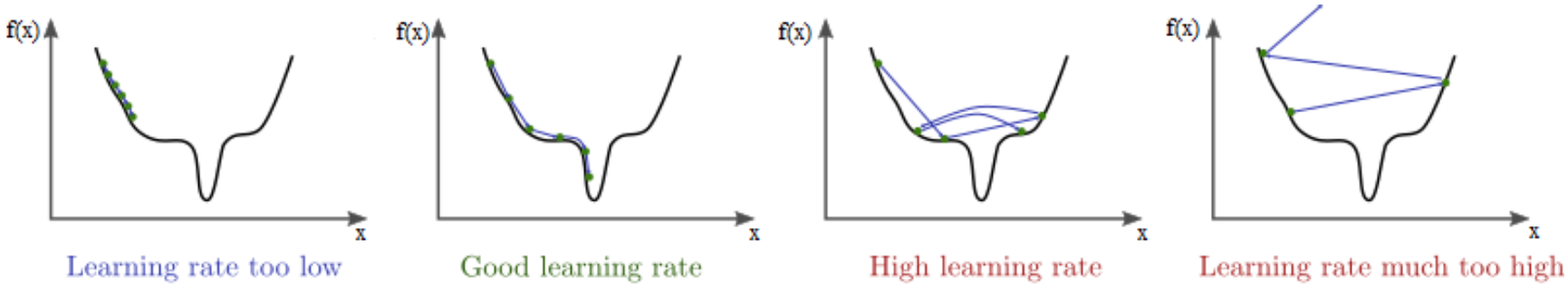
- The gradient descent algorithm converges when the following criterion is satisfied.

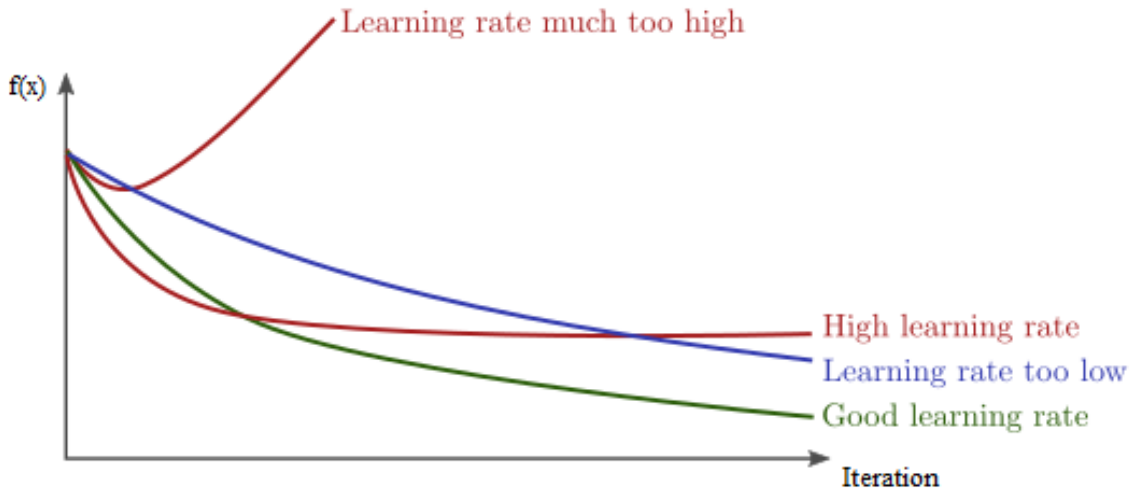
$$|f(\boldsymbol{x}^{t+k}) - f(\boldsymbol{x}^t)| < \epsilon$$

  - k is a hyperparameter.

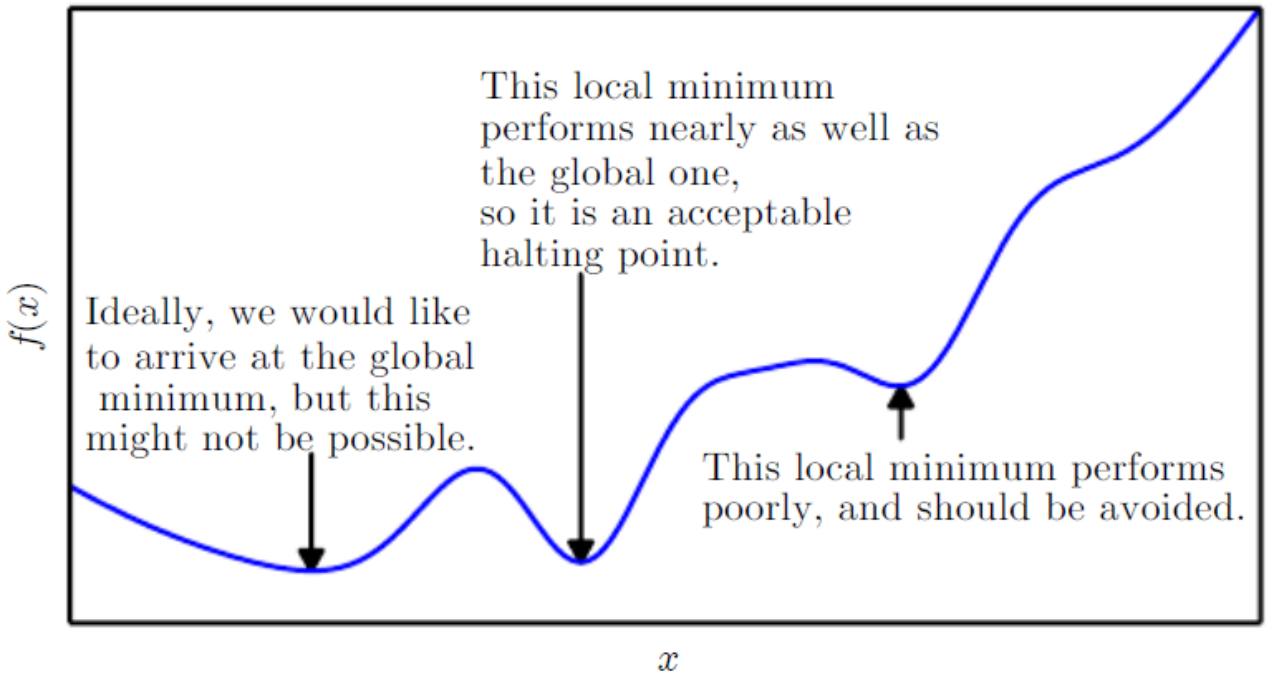


Influence of step size





Local minimum



Logistic regression weights update

- Logistic regression loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) - (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}))$$

Logistic regression weights update

- Logistic regression loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) - (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}))$$

How do we find the best w?

Logistic regression weights update

- Logistic regression loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) - (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}))$$

How do we find the best w? Gradient Descent

$$\frac{\partial L_i}{\partial w_j} = -(y^{(i)} \frac{1}{\sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})} - (1 - y^{(i)}) \frac{1}{1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})}) \frac{\partial}{\partial w_j} \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})$$

$$\frac{\partial L_i}{\partial w_j} = -(y^{(i)} \frac{1}{\sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})} - (1 - y^{(i)}) \frac{1}{1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})}) \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})) \frac{\partial}{\partial w_j} \boldsymbol{w}^T \boldsymbol{x}$$

$$\frac{\partial L_i}{\partial w_j} = -(y^{(i)}(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})) - (1 - y^{(i)}) \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)})) x_j^{(i)}$$

$$\frac{\partial L_i}{\partial w_j} = -(y^{(i)} - f(\boldsymbol{x}^{(i)}, \boldsymbol{w})) x_j^{(i)}$$

Logistic regression weights update

- Gradient descent to train logistic regression

$$w_j^{t+1} = w_j^t - \eta \frac{\partial L_i}{\partial w_j} \quad (*)$$

$$\frac{\partial L_i}{\partial w_j} = -(y^{(i)} - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))x_j^{(i)} \quad (**)$$

$$\underset{(*)\ \&\ (**)}{\Longrightarrow} w_j^{t+1} = w_j^t - \eta \frac{1}{N} \sum_{i=1}^{N} (-(y^{(i)} - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}^t))x_j^{(i)})$$

$$w_j^{t+1} = w_j^t + \eta \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}^t))x_j^{(i)}$$

$$\Longrightarrow \boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}^t))\boldsymbol{x}^{(i)}$$

- Perceptron learning algorithm

  For misclassified sample $\boldsymbol{x}^{(i)}$:

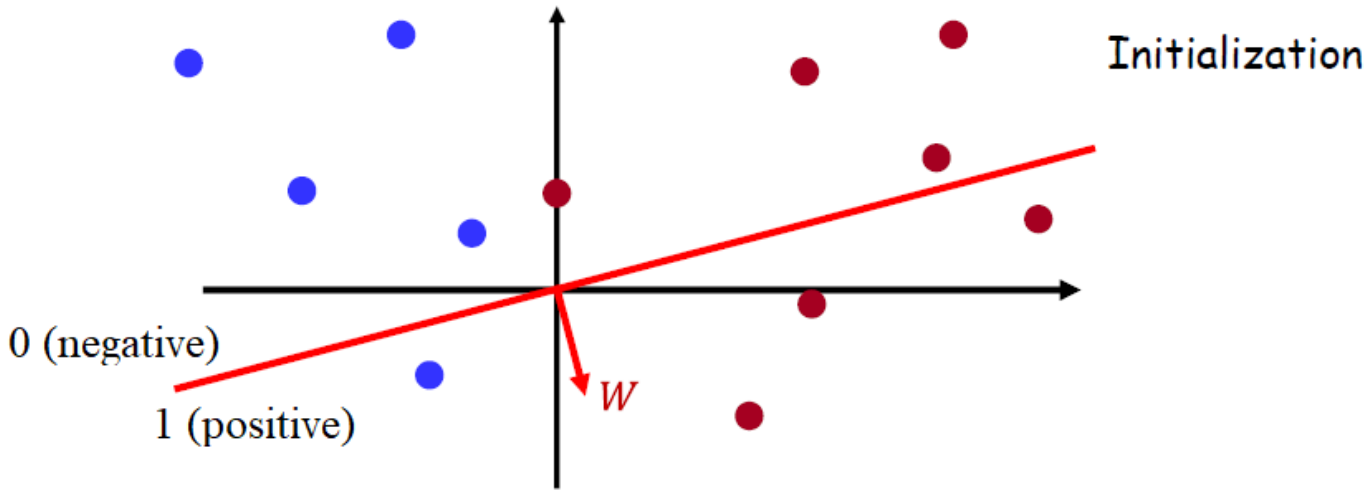
$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \boldsymbol{x}^{(i)}$$

$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \boldsymbol{x}^{(i)}$$

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

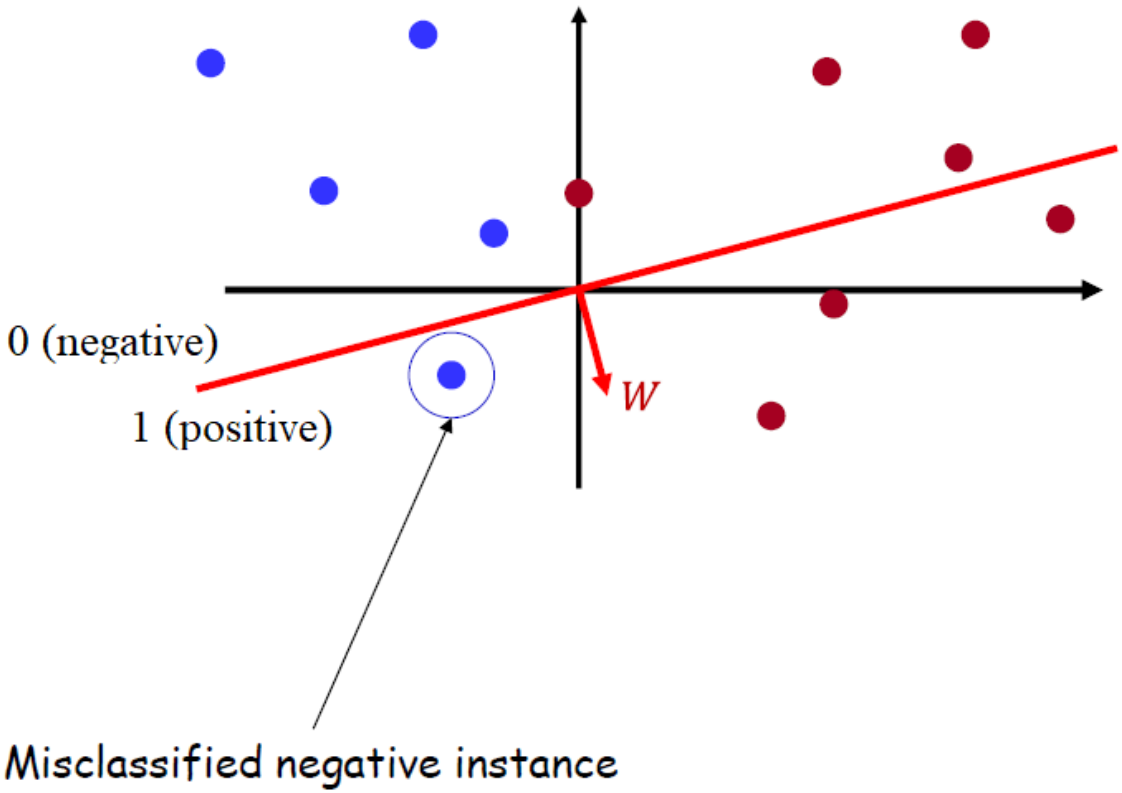
$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$



Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

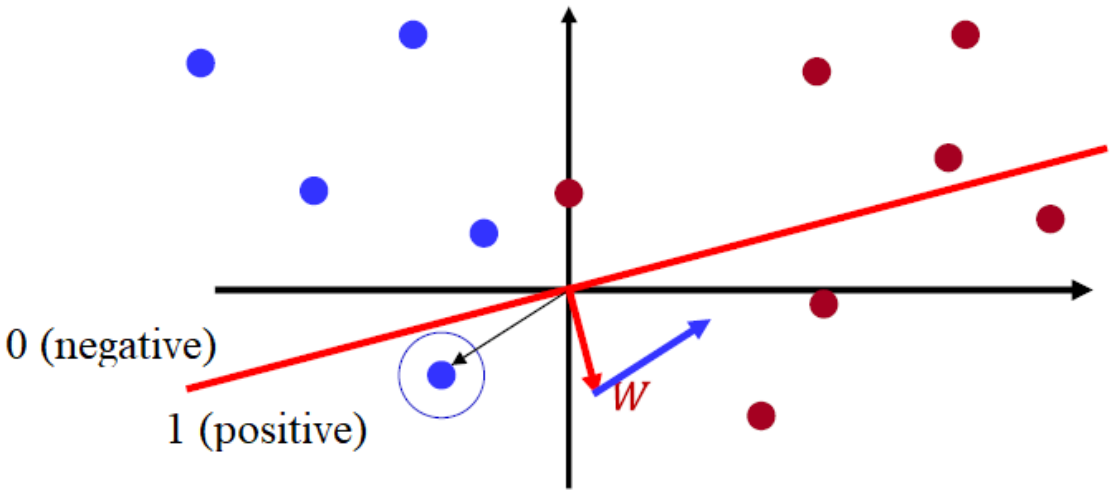
$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$



Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$

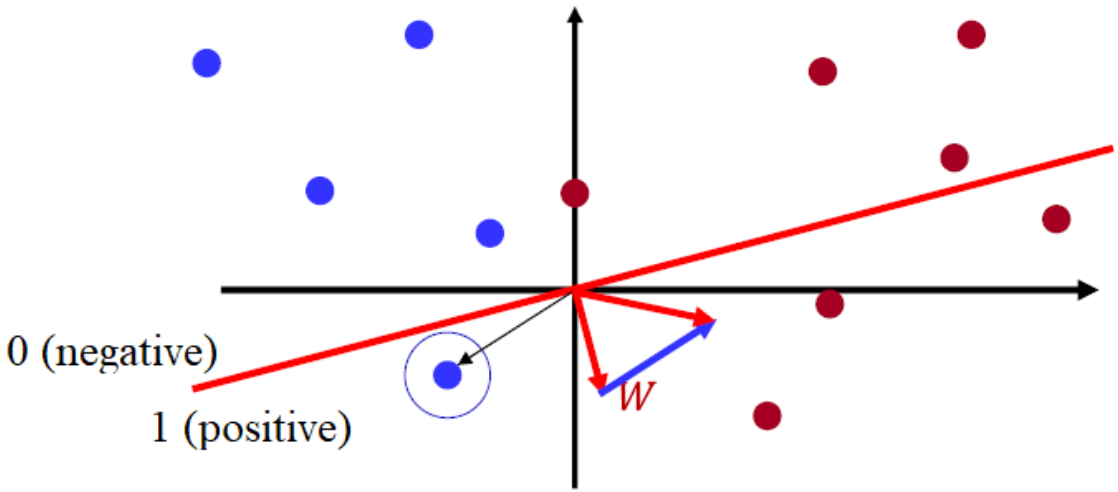


Misclassified *negative* instance, *subtract* it from W

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$

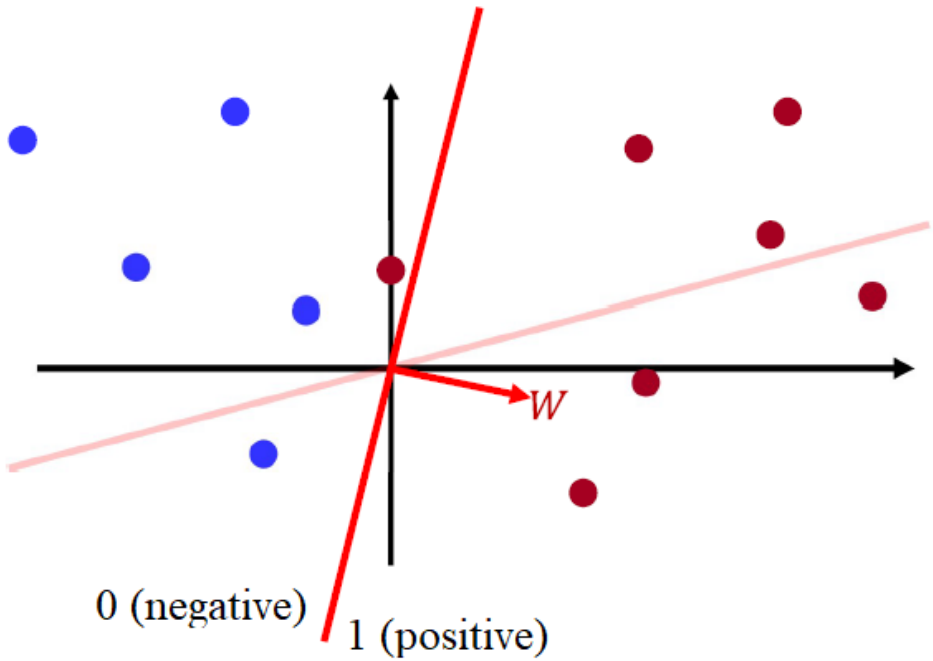


The new weight

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$

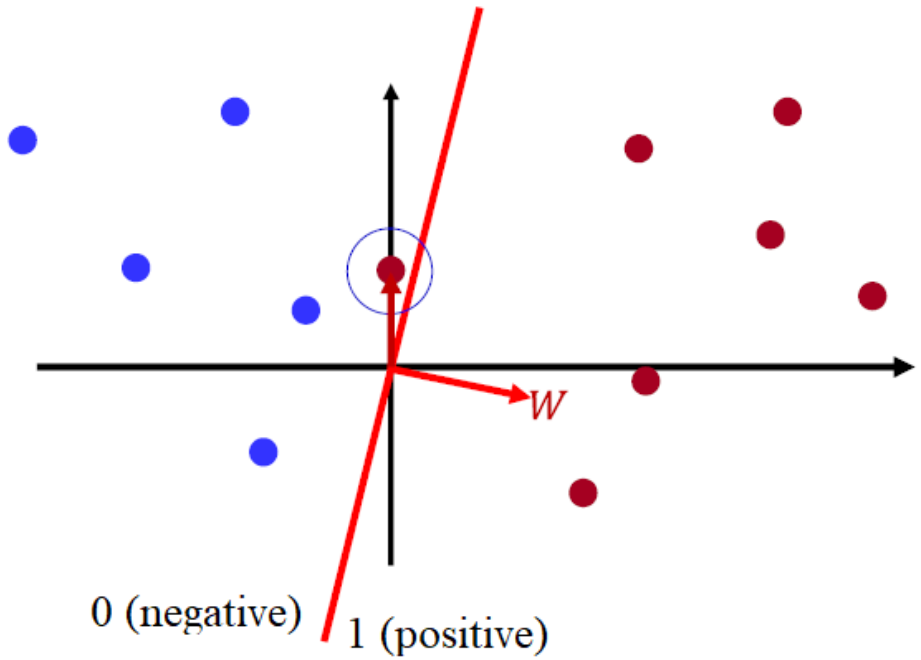


The new weight (and boundary)

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$

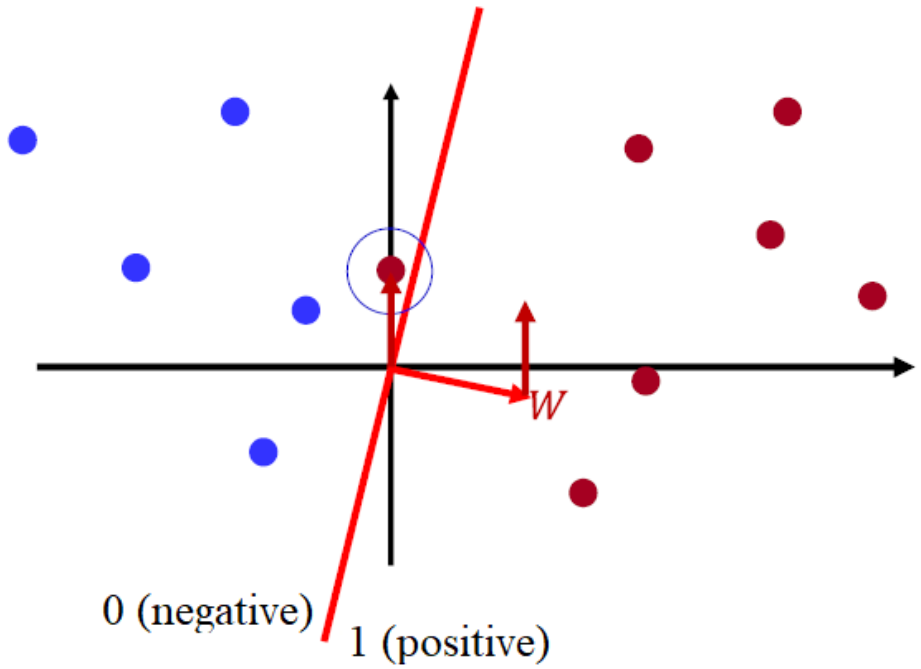


Misclassified *positive* instance

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$

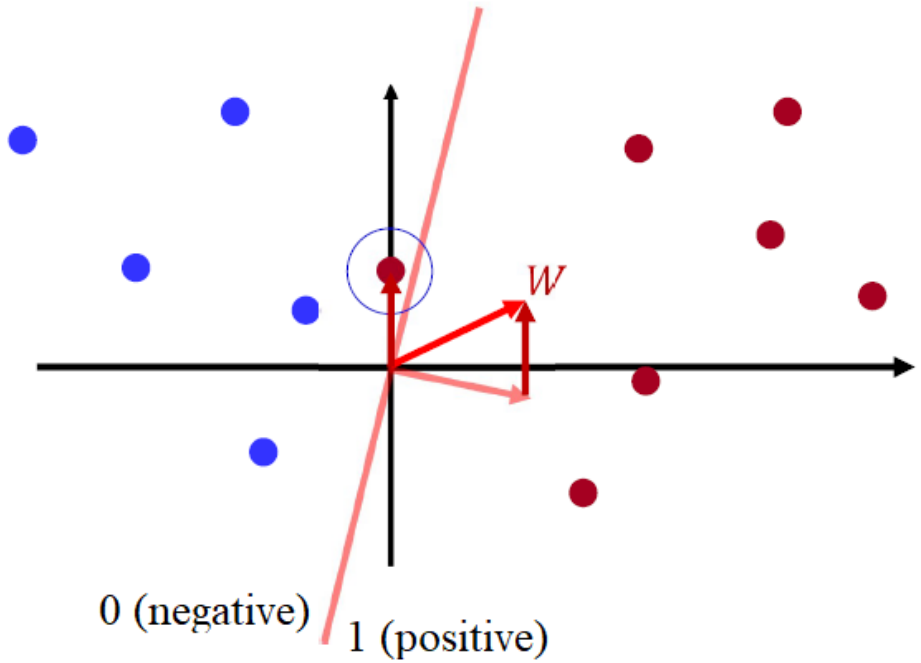


Misclassified *positive* instance, *add* it to W

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$

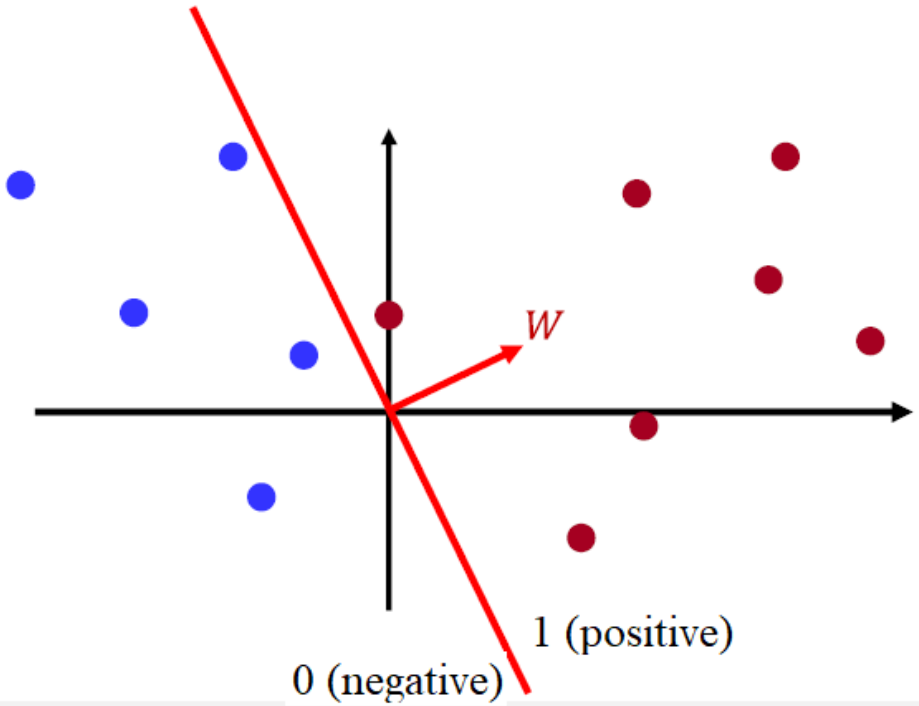


The new weight vector

Perceptron learning algorithm

For misclassified sample $\boldsymbol{x}^{(i)}$:

$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \boldsymbol{x}^{(i)}$$
$$\text{If } y^{(i)} = 1: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)}$$



The new decision boundary

Perfect classification, no more updates, we are done

Regularization

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.
- If the training and the test set are collected arbitrarily, there is indeed little we can do.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.
- If the training and the test set are collected arbitrarily, there is indeed little we can do.
- The train and test data are generated by a probability distribution over datasets called the data generating process. We typically make a set of assumptions known collectively as the **i.i.d.** assumptions.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.
- If the training and the test set are collected arbitrarily, there is indeed little we can do.
- The train and test data are generated by a probability distribution over datasets called the data generating process. We typically make a set of assumptions known collectively as the **i.i.d.** assumptions.
- These assumptions are that the examples in each dataset are independent from each other, and that the train set and test set are identically distributed, drawn from the same probability distribution as each other.

(Deep Learning, Ian Goodfellow, MIT press, 2016)

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.
- Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

Machine learning

underfitting and overfitting

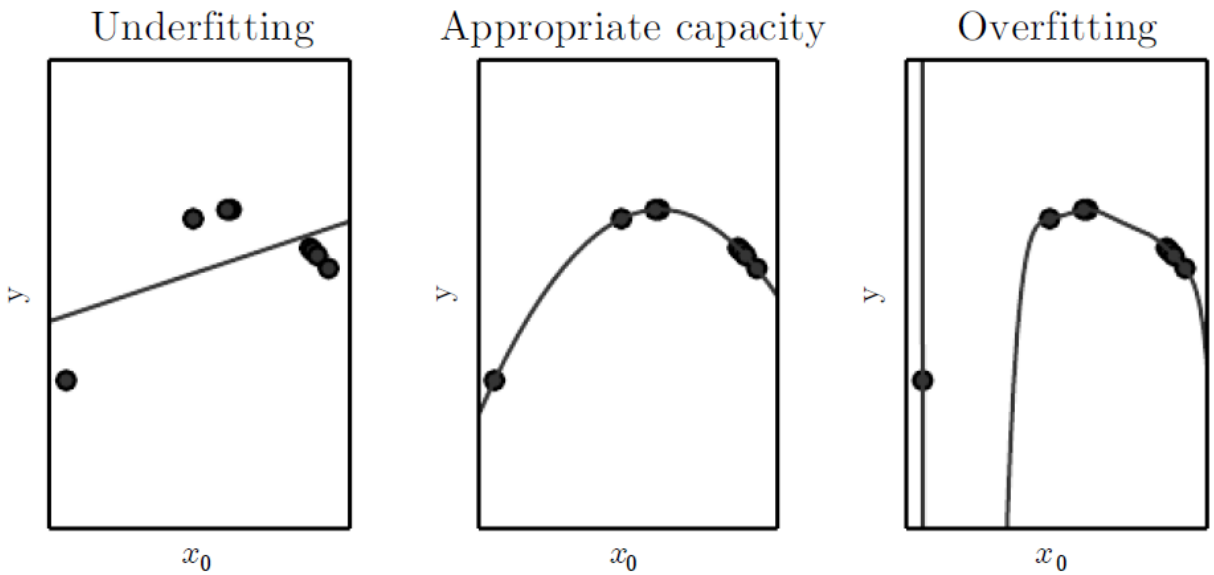
- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.
- Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.
- One way to control the capacity of a learning algorithm is by choosing its **hypothesis space**, the set of functions that the learning algorithm is allowed to select as being the solution.

(Deep Learning, Ian Goodfellow, MIT press, 2016)

Machine learning

underfitting and overfitting

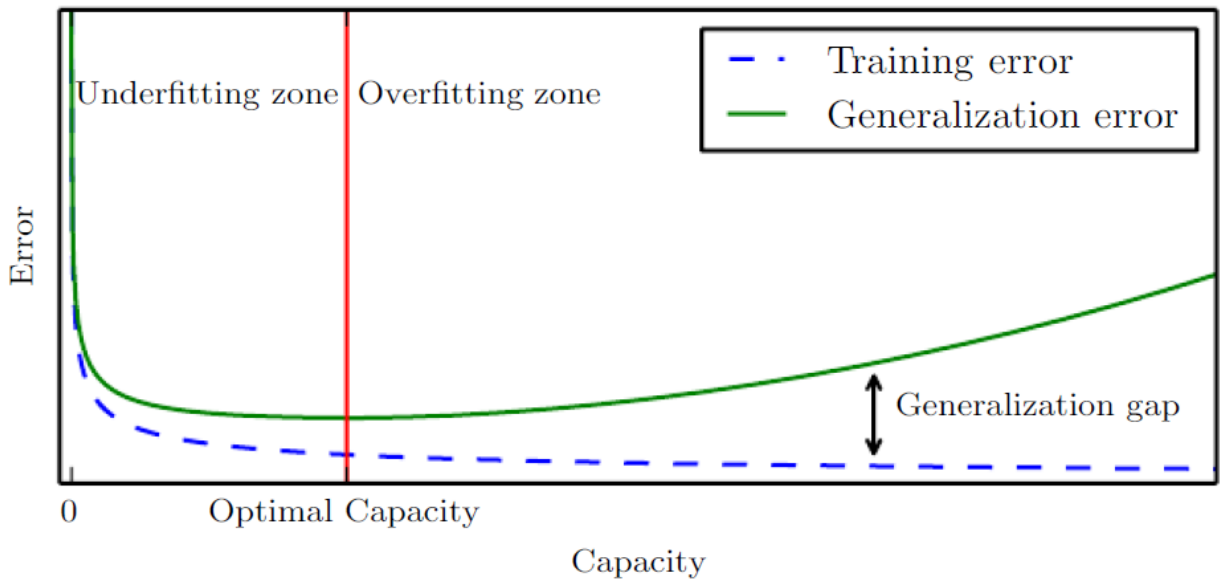
- Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with.



(Deep Learning, Ian Goodfellow, MIT press, 2016)

Capacity

Typically, generalization error has a U-shaped curve as a function of model capacity.



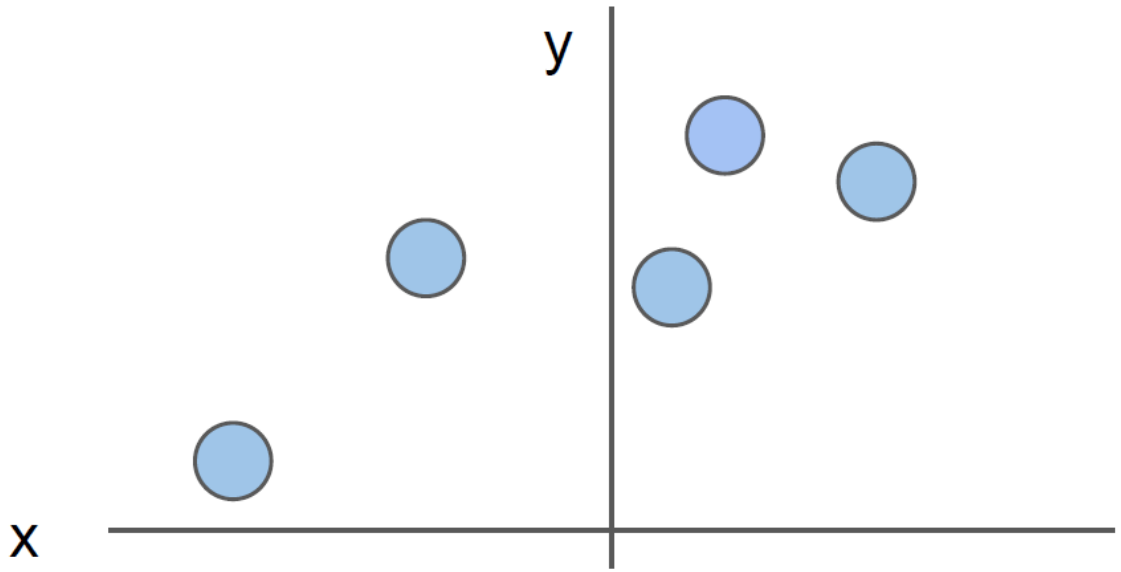
Regularization

- **Regularization** is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
  - We can give a learning algorithm a **preference** for one solution in its hypothesis space to another. This means that both functions are eligible, but one is preferred.

Regularization

- How do we choose between hypotheses?
  - Occam's Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
  - Avoid overfitting: prefer simple models that generalize better

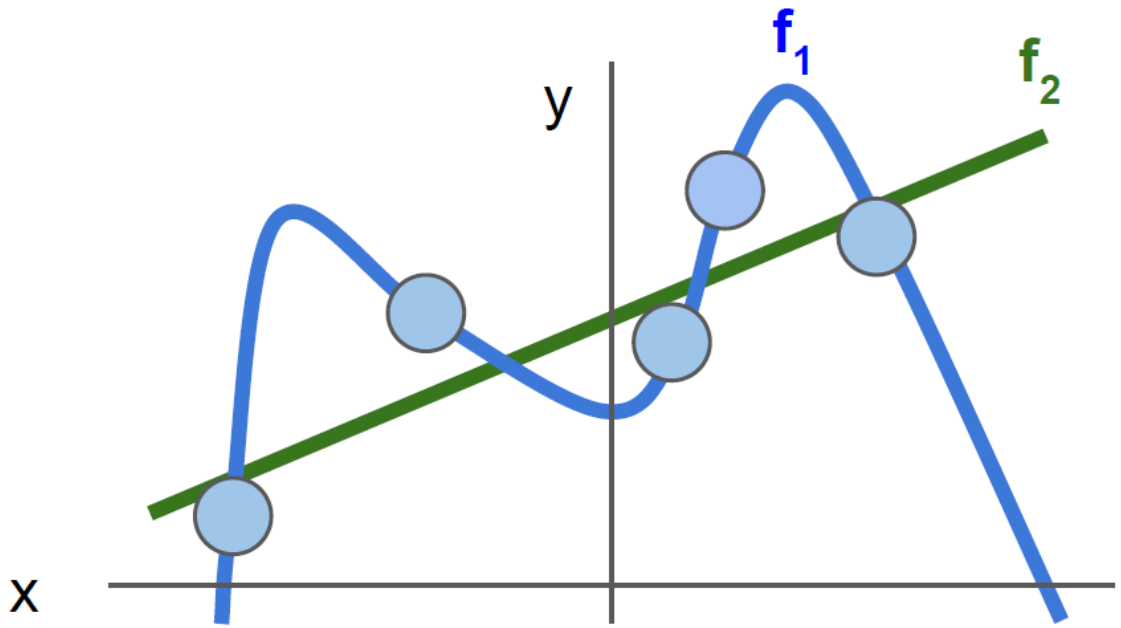
- Regularization intuition: toy example training data



Regularization

- How do we choose between hypotheses?
  - Occam's Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
  - Avoid overfitting: prefer simple models that generalize better

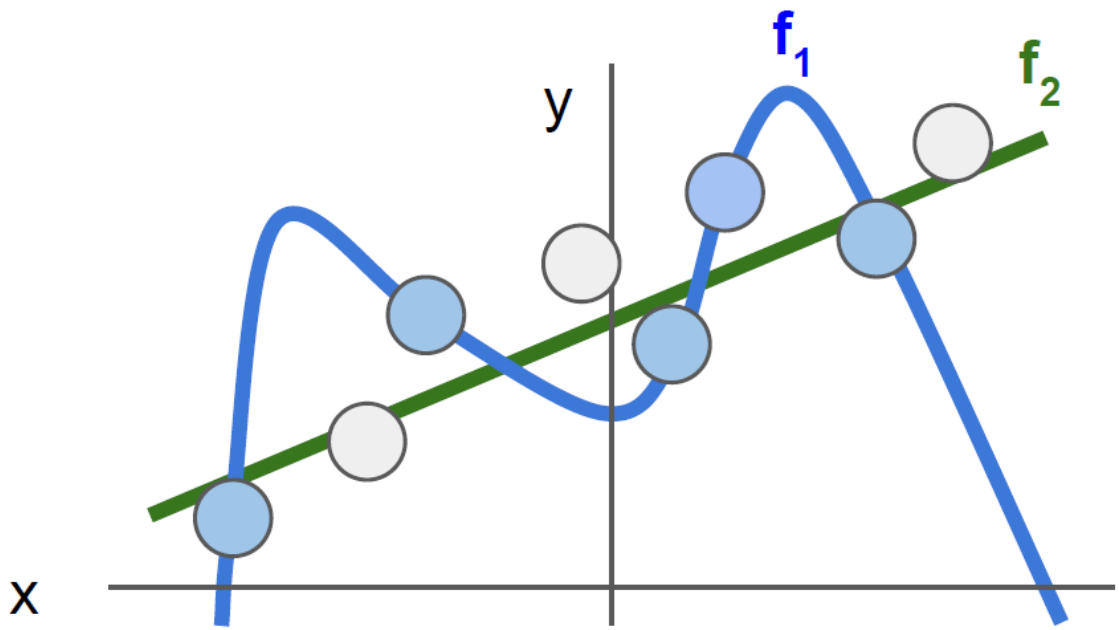
- Regularization intuition: Prefer Simpler Models



Regularization

- How do we choose between hypotheses?
  - Occam's Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
  - Avoid overfitting: prefer simple models that generalize better

- Regularization: Prefer Simpler Models



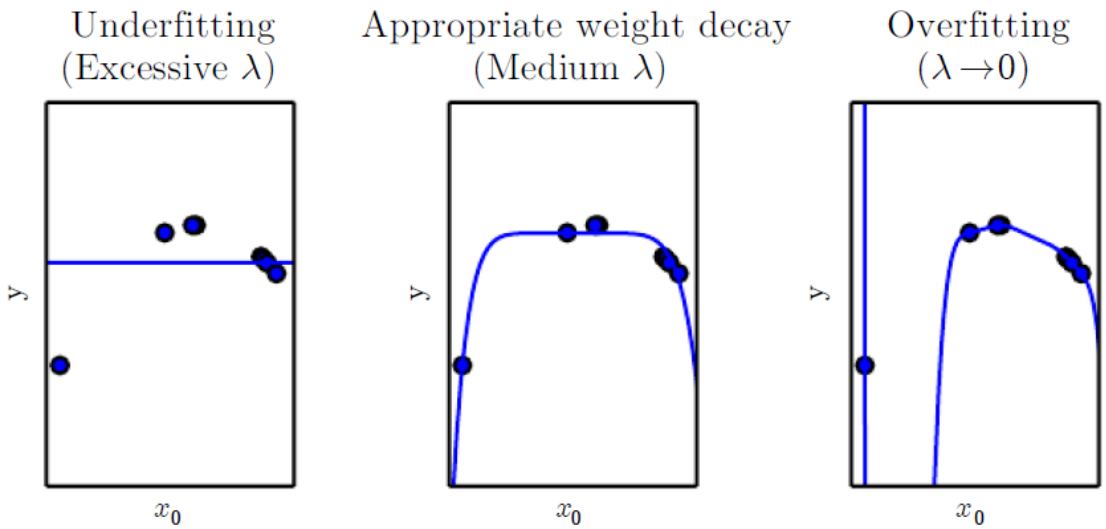
- Regularization pushes against fitting the data too well so we don't fit noise in the data

Regularization

- **Training loss:** Model predictions should match training data
- **Regularization**: Prevent the model from doing too well on training data $R(\boldsymbol{w})$

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)}) + \lambda R(\boldsymbol{w})$$

- λ: regularization strength (hyperparameter)
- Simple examples
  - $L1$ regularization: $R(\boldsymbol{w}) = \sum_i |w_i|$
  - $L2$ regularization: $R(\boldsymbol{w}) = \sum_i w_i^2$



Regularization

- Which of $\boldsymbol{w}_1$ or $\boldsymbol{w}_2$ will the $L2$ regularizer prefer? (Training loss is the same)
  - $L2$ regularization likes to "spread out" the weights

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \boldsymbol{w}_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

$$\boldsymbol{w}_1^T \boldsymbol{x}^{(i)} = \boldsymbol{w}_2^T \boldsymbol{x}^{(i)} = 1$$

Regularization

- Which of $\boldsymbol{w}_1$ or $\boldsymbol{w}_2$ will the $L2$ regularizer prefer? (Training loss is the same)
  - $L2$ regularization likes to "spread out" the weights

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \boldsymbol{w}_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

$$\boldsymbol{w}_1^T \boldsymbol{x}^{(i)} = \boldsymbol{w}_2^T \boldsymbol{x}^{(i)} = 1$$

$$L2(\boldsymbol{w}_1) = 1, \quad L2(\boldsymbol{w}_2) = 0.25$$

Regularization

- Which of $\boldsymbol{w}_1$ or $\boldsymbol{w}_2$ will the $L2$ regularizer prefer? (Training loss is the same)
  - $L2$ regularization likes to "spread out" the weights

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \boldsymbol{w}_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

$$\boldsymbol{w}_1^T \boldsymbol{x}^{(i)} = \boldsymbol{w}_2^T \boldsymbol{x}^{(i)} = 1$$

$$L2(\boldsymbol{w}_1) = 1, \quad L2(\boldsymbol{w}_2) = 0.25$$

- More complex regularization methods
  - Dropout
  - MixUp
  - Stochastic depth

References

- Deep Learning, Ian Goodfellow, MIT Press, Ch.6, 7 ,and 8