Machine Learning

A. M. Sadeghzadeh, Ph.D.

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

September 24, 2024

Most slides have been adapted from Bhiksha Raj, 11-785, CMU 2020, Fei Fei Li, cs231n, Stanford 2017, and Soleymani, CE40717, Sharif 2020.

Schedule

Machine Learning

Definition of machine learning

Tom Mitchell (1998)

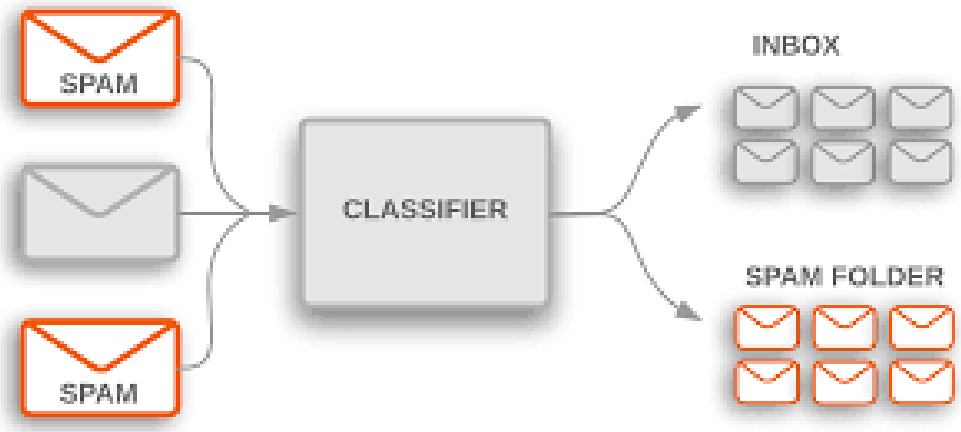
A computer program is said to **learn** from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

Example

Task: Classifying emails as spam or not spam
Experience: Watching label of emails as spam or not spam
Performance metric: The number (or fraction) of emails correctly classified as spam

The essence of machine learning

- A pattern exist
- We do not know it mathematically
- We have data on it



Paradigms of ML

- **Supervised learning**
  Predicting a target variable for which we get to see examples

Paradigms of ML

- **Supervised learning**
  Predicting a target variable for which we get to see examples
- **Unsupervised learning**
  Revealing structure in the observed data

Paradigms of ML

- **Supervised learning**
  Predicting a target variable for which we get to see examples
- **Unsupervised learning**
  Revealing structure in the observed data
- **Reinforcement learning**
  Partial (indirect) feedback, no explicit guidance
  Given rewards for a sequence of moves to learn a policy and utility functions

Supervised learning

Regression

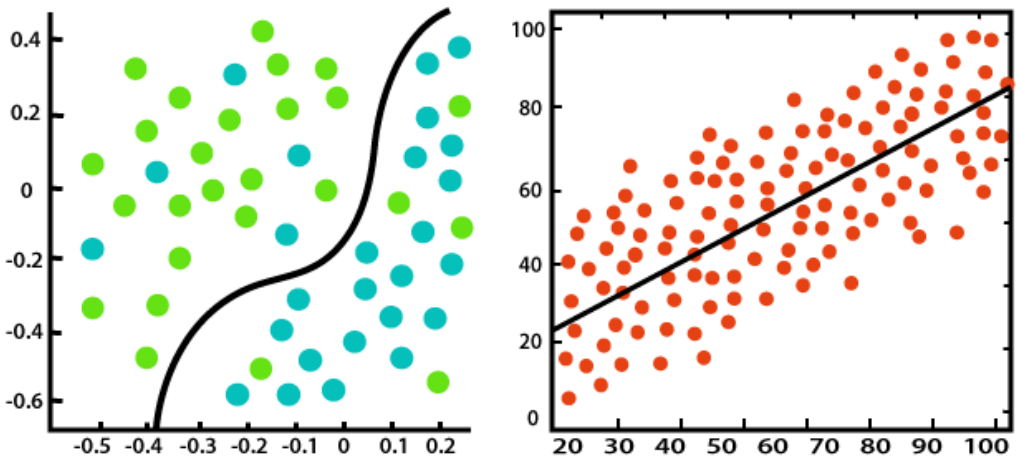
predict a continuous target variable

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

Classification

predict a discrete (unordered) target variable

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



javatpoint.com

Supervised learning: housing price prediction

Given: a dataset that contains N samples

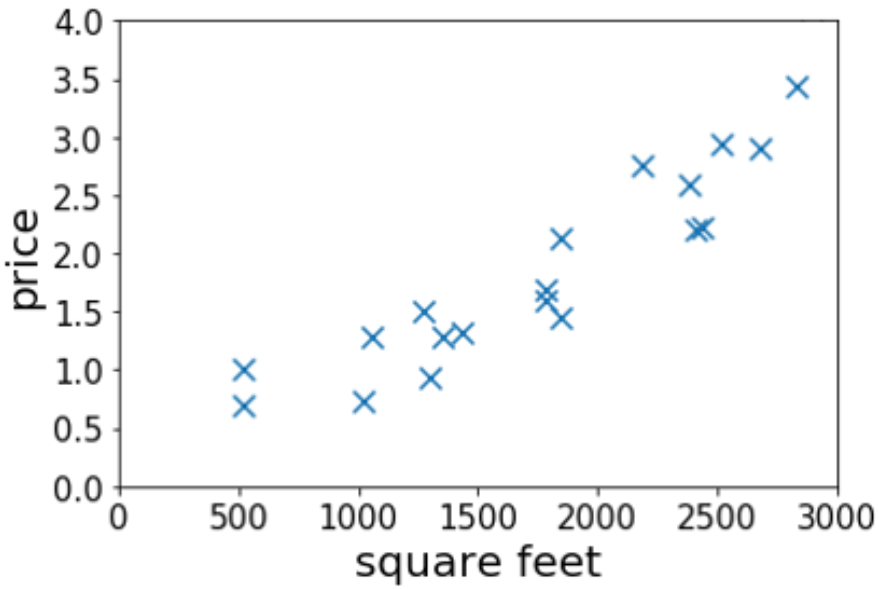
$$\{(x^{(1)}, y^{(1)}), ..., (x^{(N)}, y^{(N)})\} \tag{1}$$

Supervised learning: housing price prediction

Given: a dataset that contains N samples

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

Task: if a residence has x square feet, predict its price?

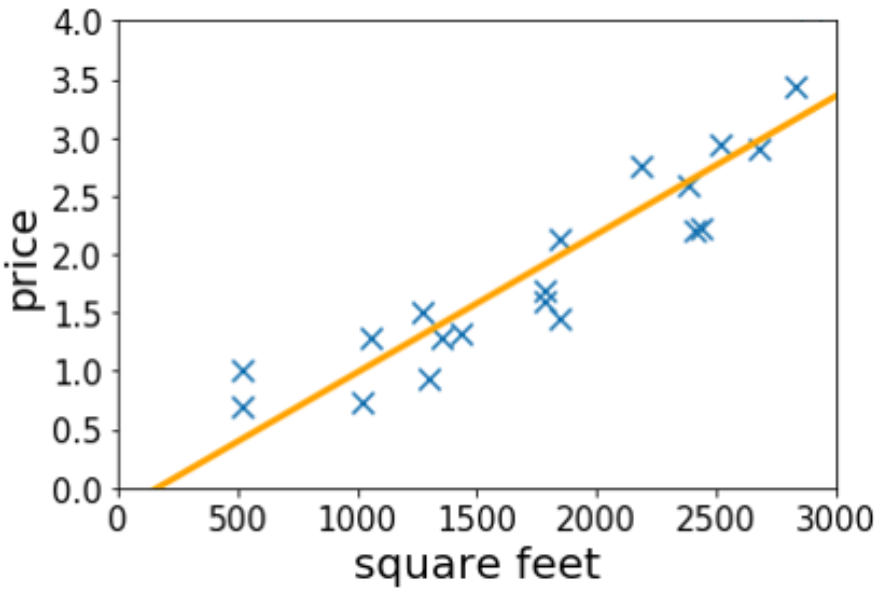


Supervised learning: housing price prediction

Given: a dataset that contains N samples

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

Task: if a residence has x square feet, predict its price?

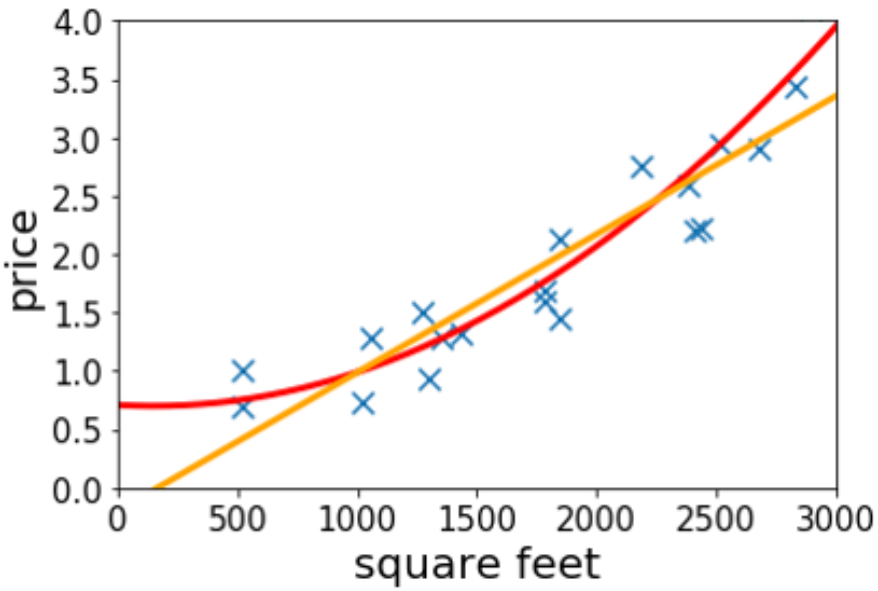


Supervised learning: housing price prediction

Given: a dataset that contains N samples

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

Task: if a residence has x square feet, predict its price?



High-dimensional features

$$\mathbf{x^{(i)}} \in \mathbb{R}^d$$

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_d \end{bmatrix}$$

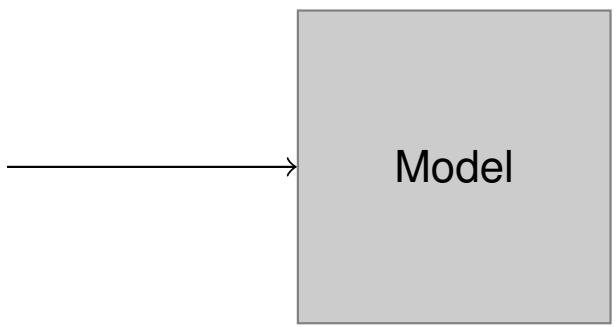
– Size
– Region
– # floors
– Age
.
.
.

High-dimensional features

$$\mathbf{x}^{(i)} \in \mathbb{R}^d$$

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_d \end{bmatrix}$$

– Size
– Region
– # floors
– Age
.
.
.



High-dimensional features

$\mathbf{x^{(i)}} \in \mathbb{R}^d$

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_d \end{bmatrix}$$

– Size
– Region
– # floors
– Age
.
.
.

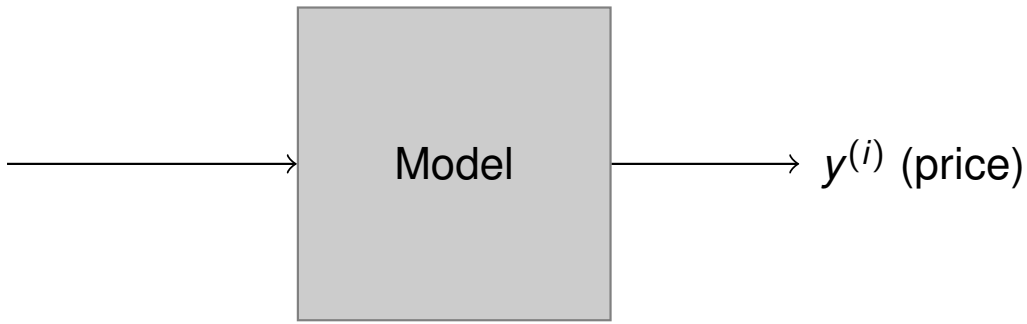
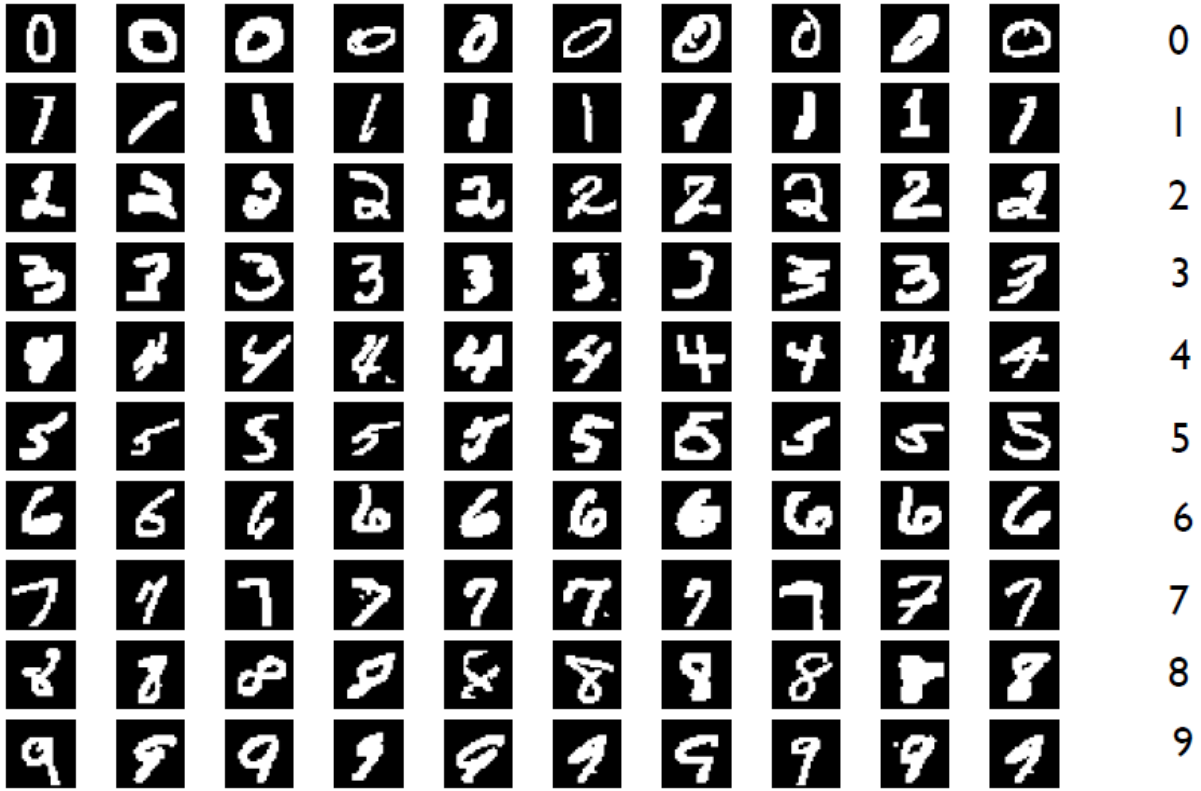
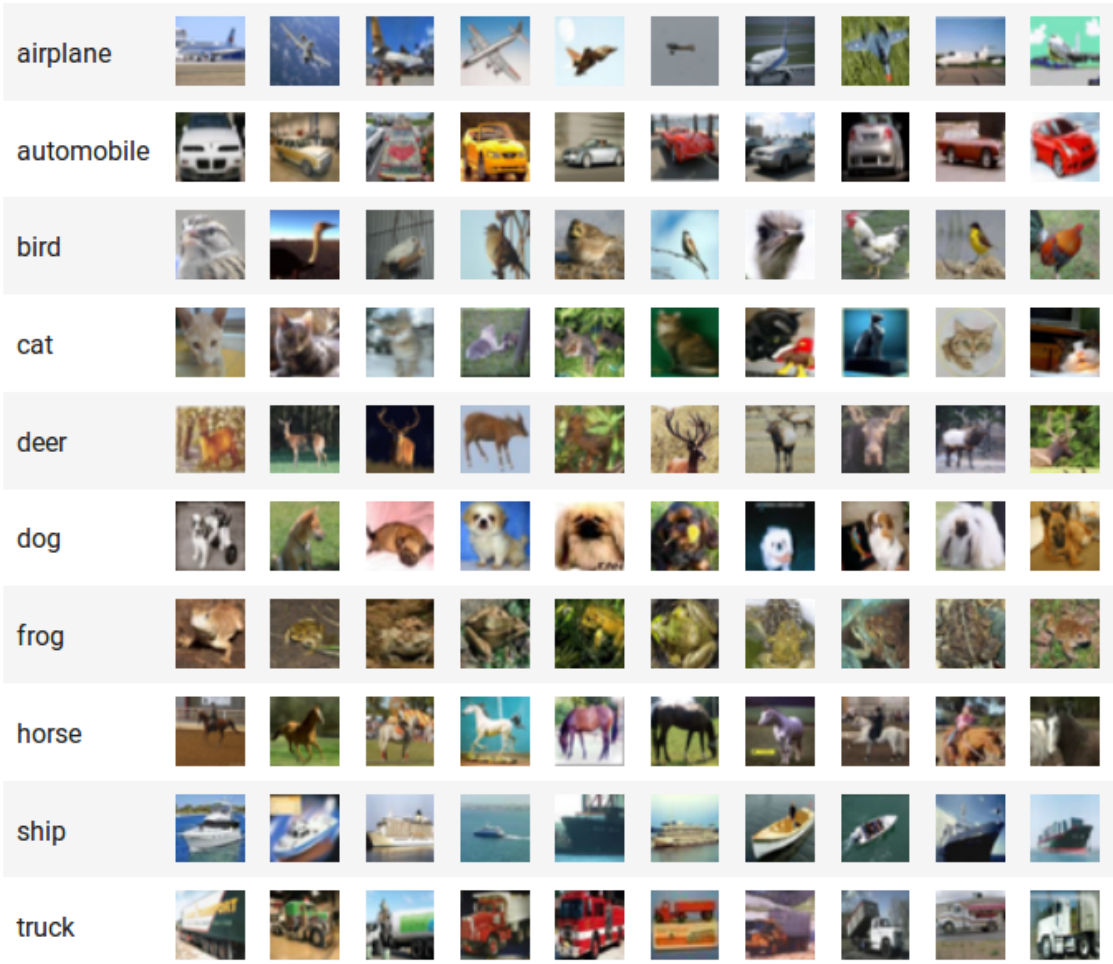


Image classification



Example images from the MNIST dataset.

Image classification



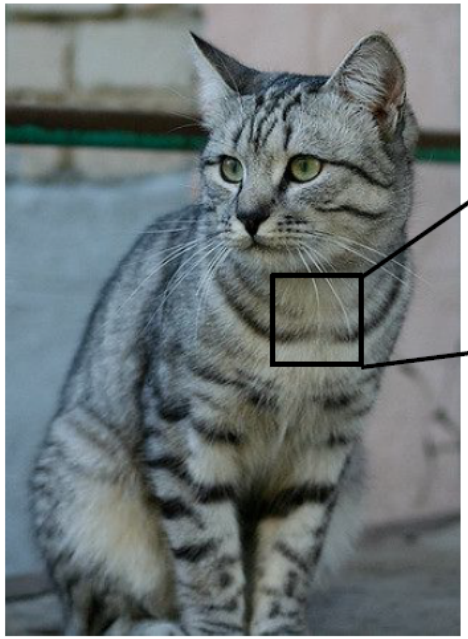
Example images from the CIFAR-10 dataset.

Image classification

(assume given a set of labels)
{dog, cat, truck, plane, ...}

cat

Image classification




```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  95  95  99 115 112 105 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 105  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  90  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  95  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a tensor of integers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Unsupervised learning

Goal

Discover the intrinsic structure in the data

Data

Dataset contains no labels: $\{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$

Unsupervised learning

Goal

Discover the intrinsic structure in the data

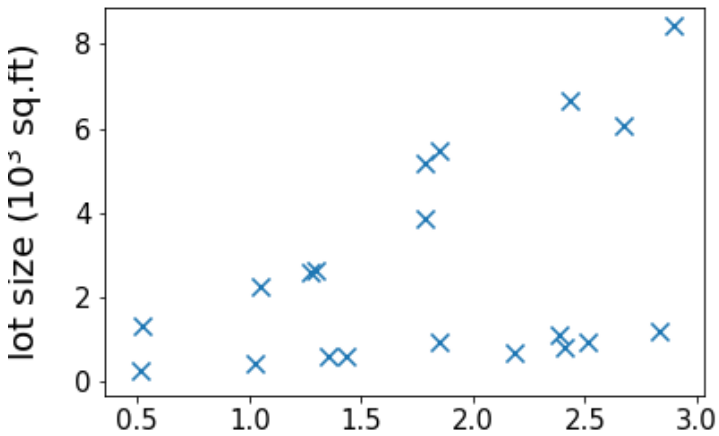
Data

Dataset contains no labels: $\{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$

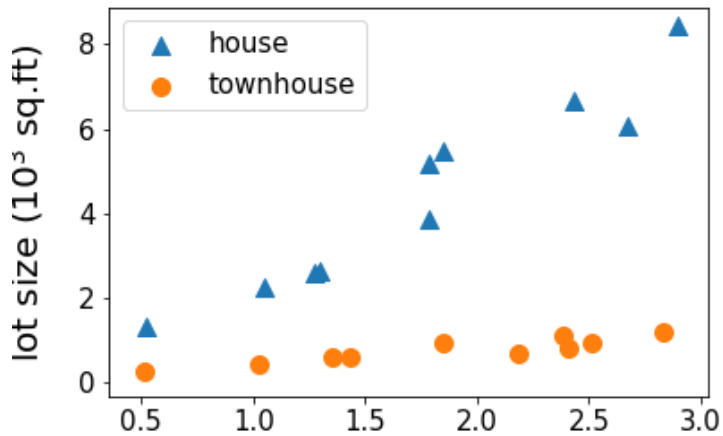
Clustering

Based on a similarity metric

Unsupervised



Supervised



Unsupervised learning

Goal

Discover the intrinsic structure in the data

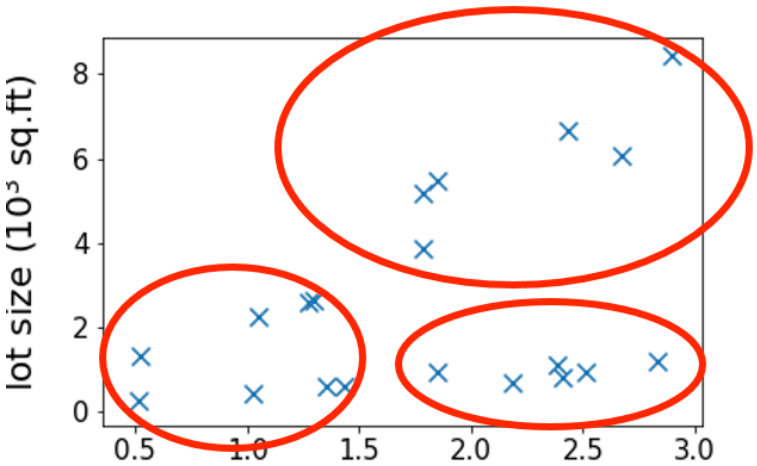
Data

Dataset contains no labels: $\{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$

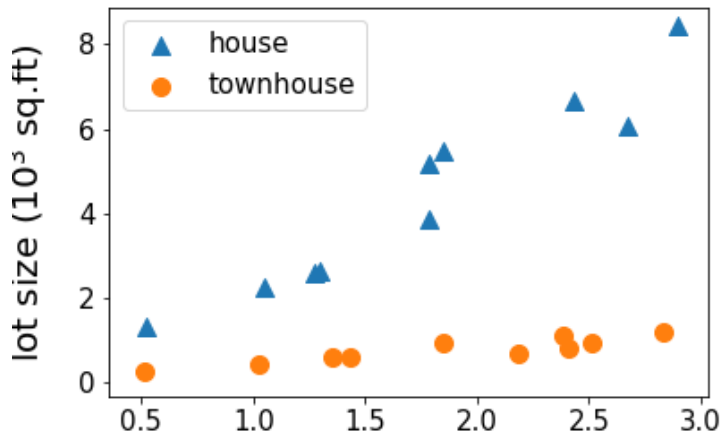
Clustering

Based on a similarity metric

Unsupervised



Supervised



Unsupervised learning

Goal

Discover the intrinsic structure in the data

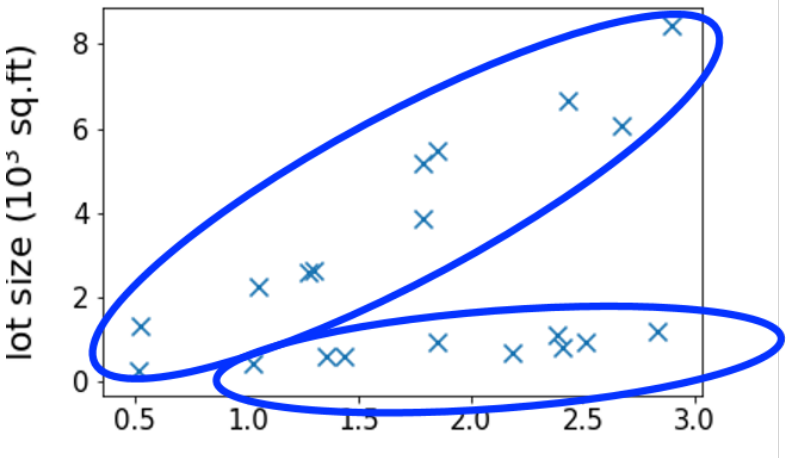
Data

Dataset contains no labels: $\{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$

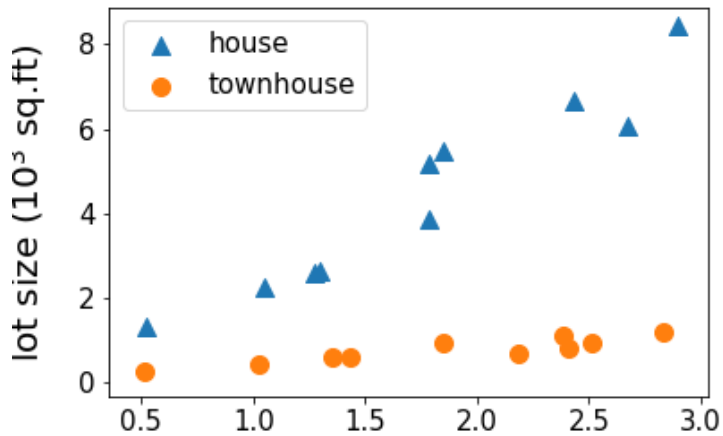
Clustering

Based on a similarity metric

Unsupervised



Supervised



Unsupervised learning

Clustering

- Partitioning of data into groups of similar data points

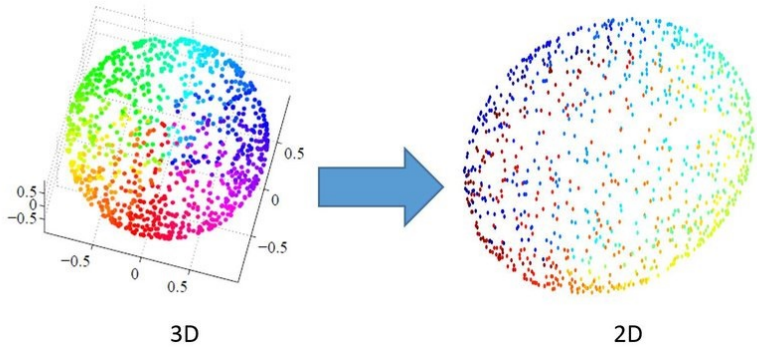
Dimensionality reduction

- Data representation using a smaller number of dimensions while preserving (perhaps approximately) some properties of the data

Synthetic data generation

- e.g., Variational AutoEncoders (VAE) and Generative Adversarial Networks (GANs),

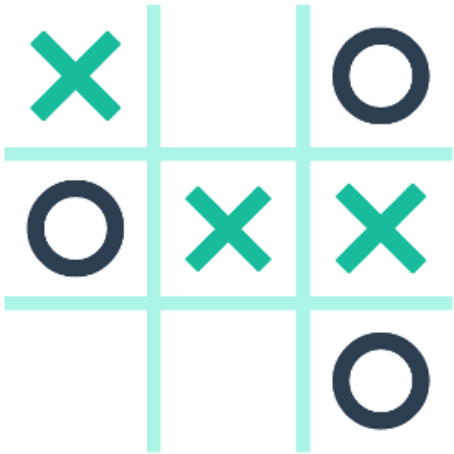
(Karras, 2019)

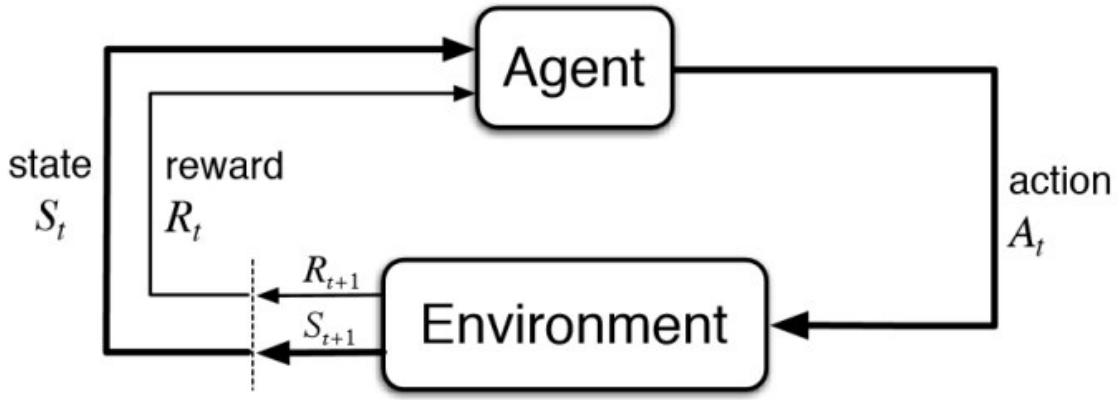


Reinforcement learning

Provides only an indication as to whether an action is correct or not

Data in supervised learning: (input, correct output)
Data in Reinforcement Learning: (input, some output, a reward for this output)





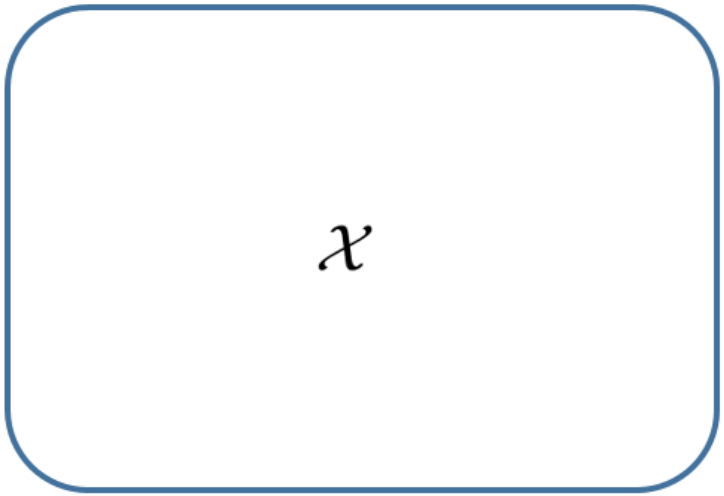
Supervised Learning

Components of supervised learning (function approximation)

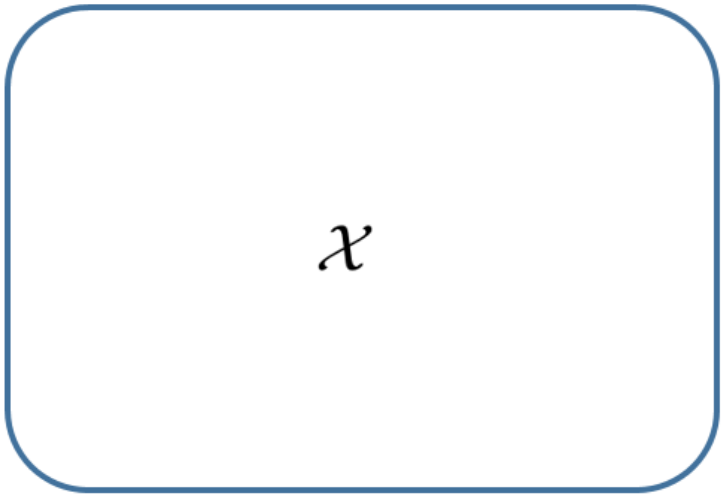
Concept

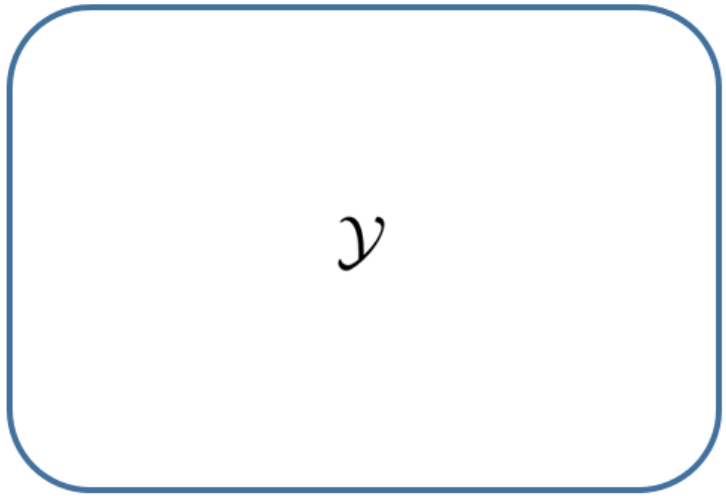
- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Components of supervised learning (function approximation)

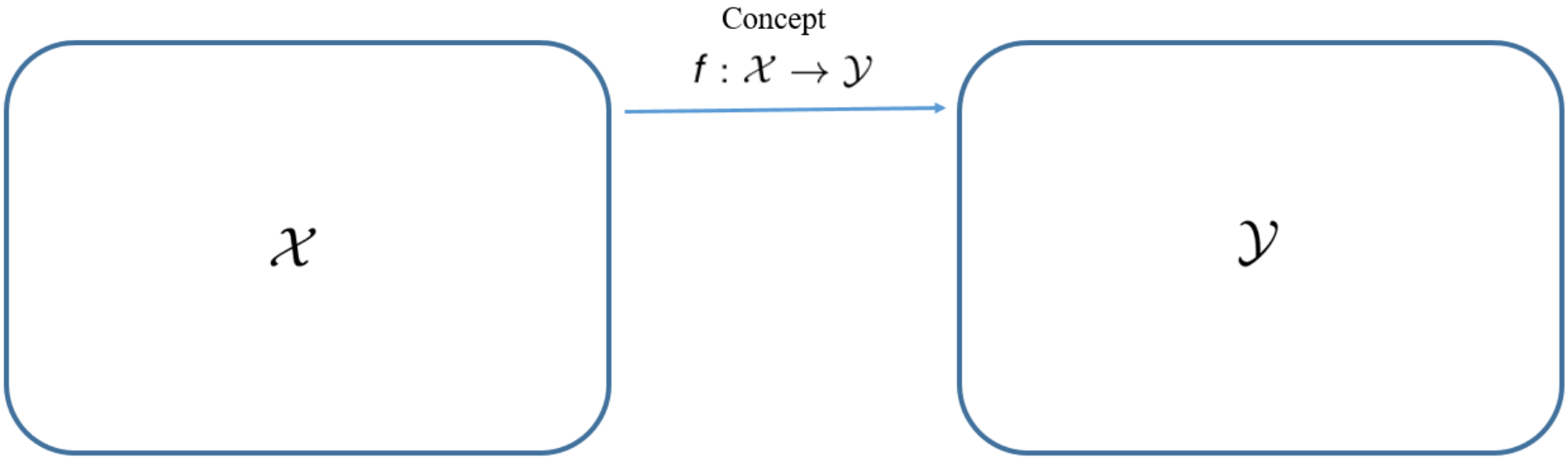


Components of supervised learning (function approximation)

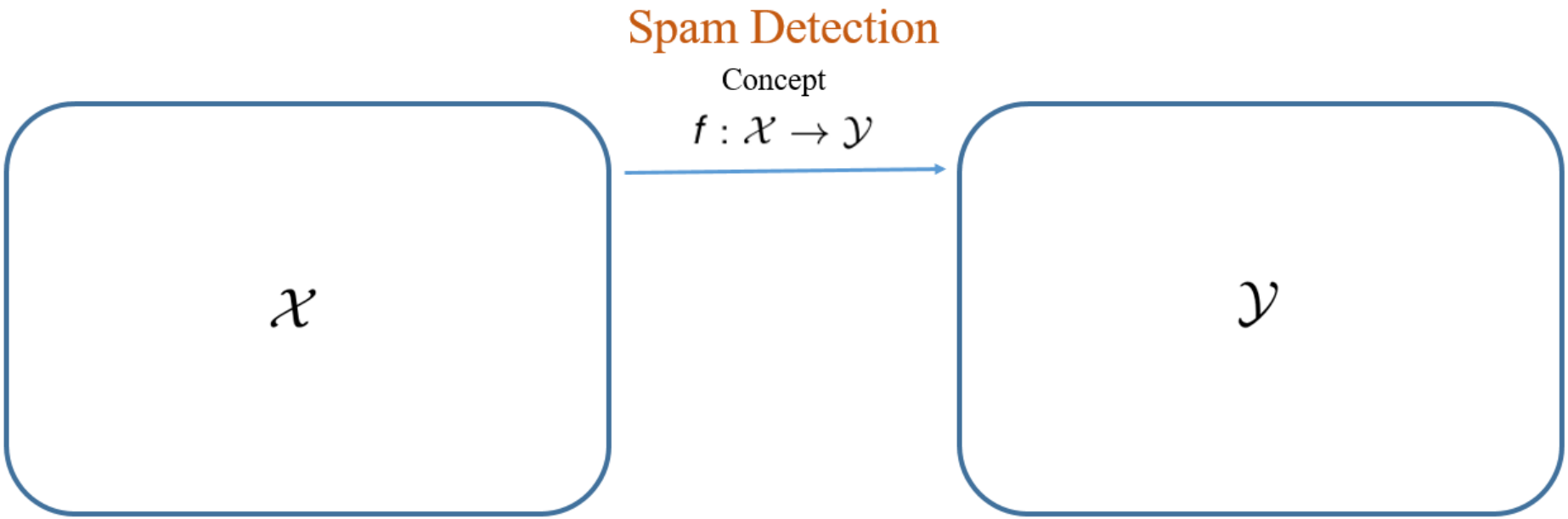




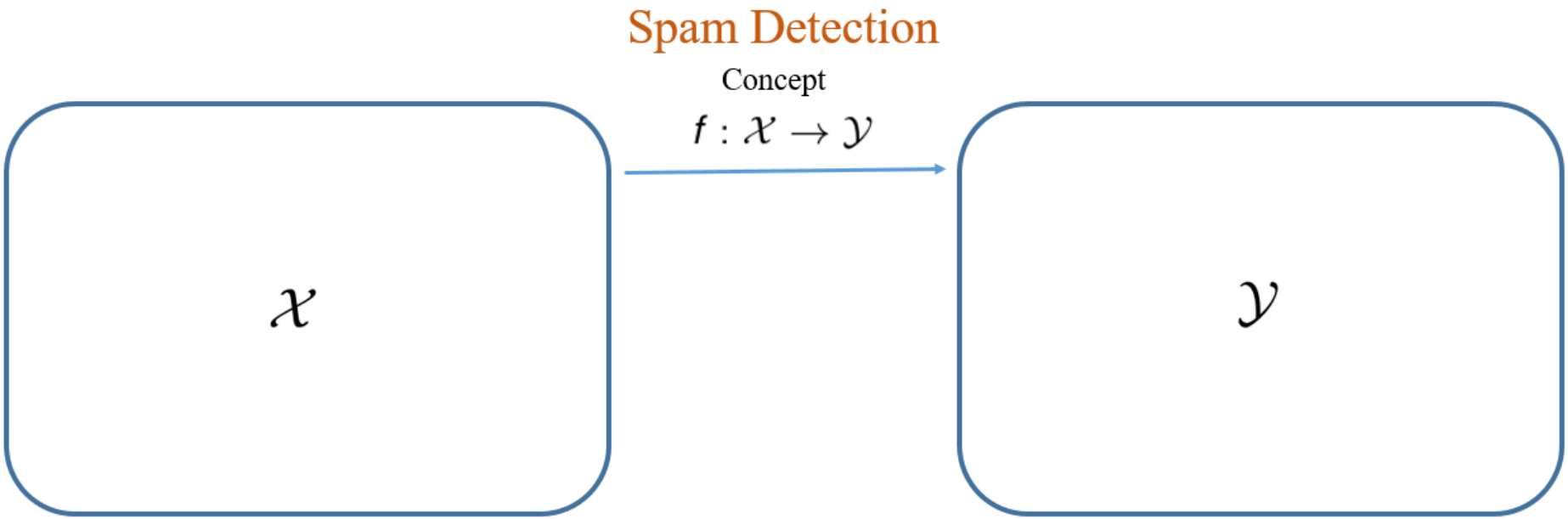
Components of supervised learning (function approximation)



Components of supervised learning (function approximation)

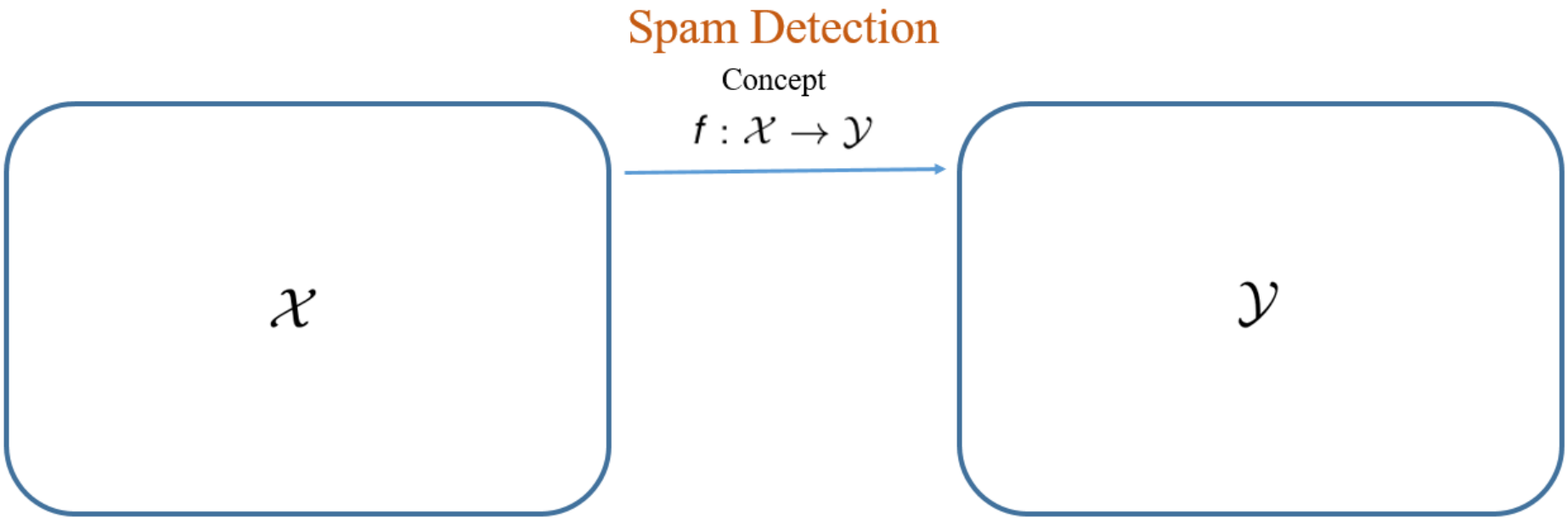


Components of supervised learning (function approximation)



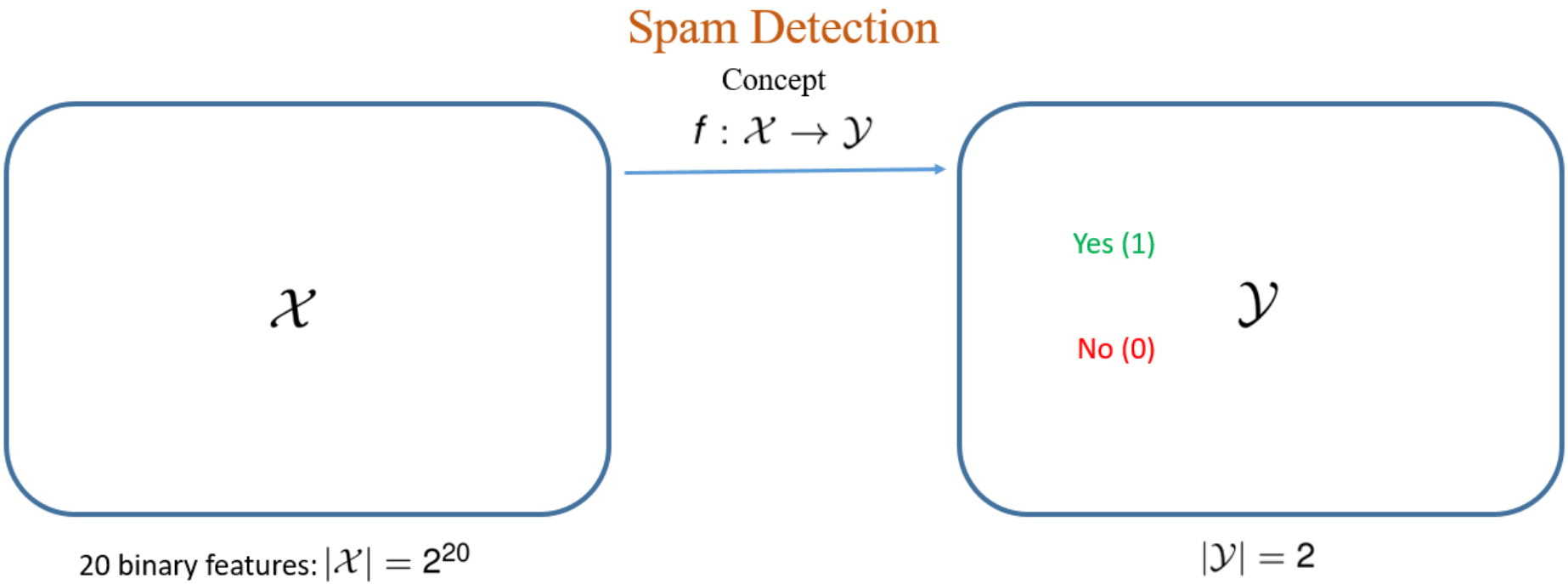
20 binary features:

Components of supervised learning (function approximation)

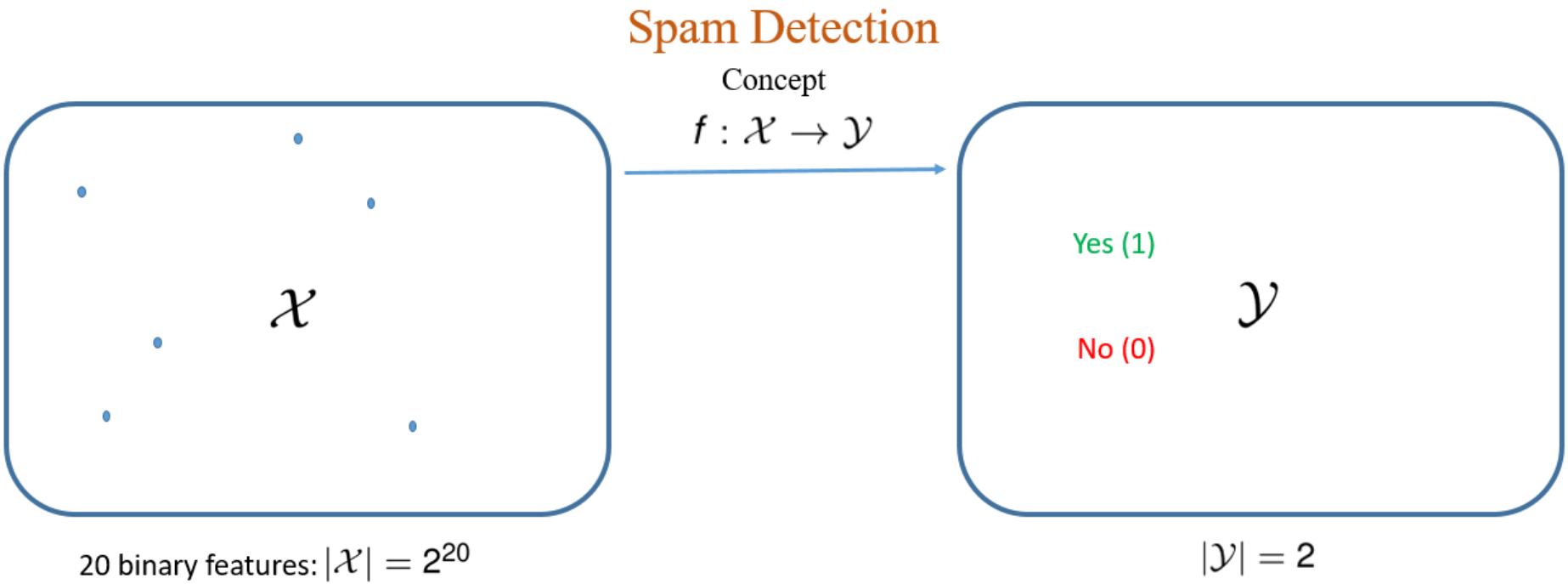


20 binary features: $|\mathcal{X}| = 2^{20}$

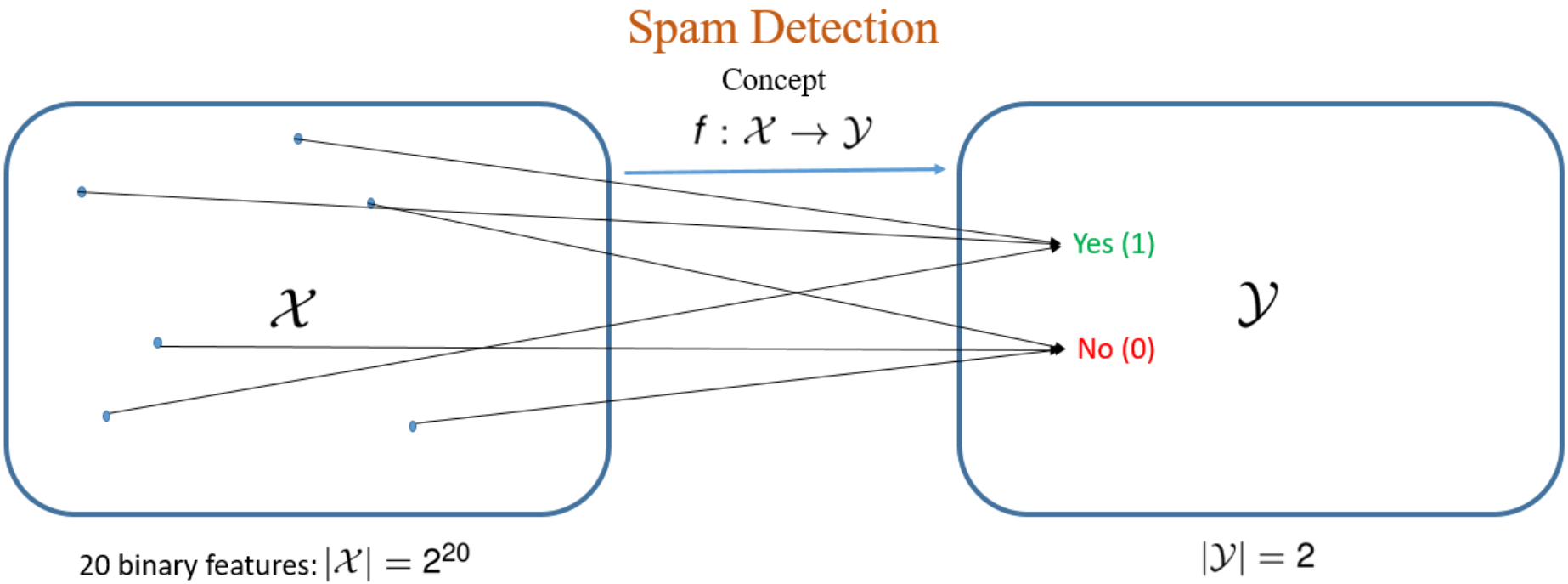
Components of supervised learning (function approximation)



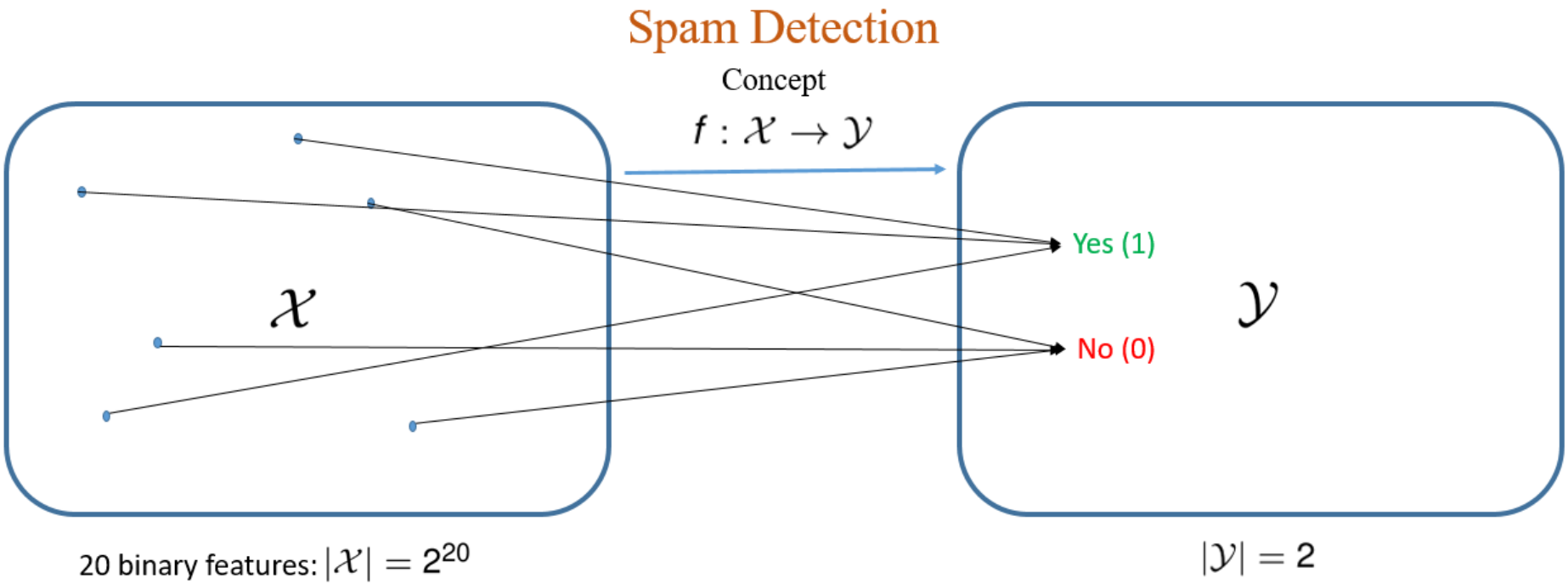
Components of supervised learning (function approximation)



Components of supervised learning (function approximation)

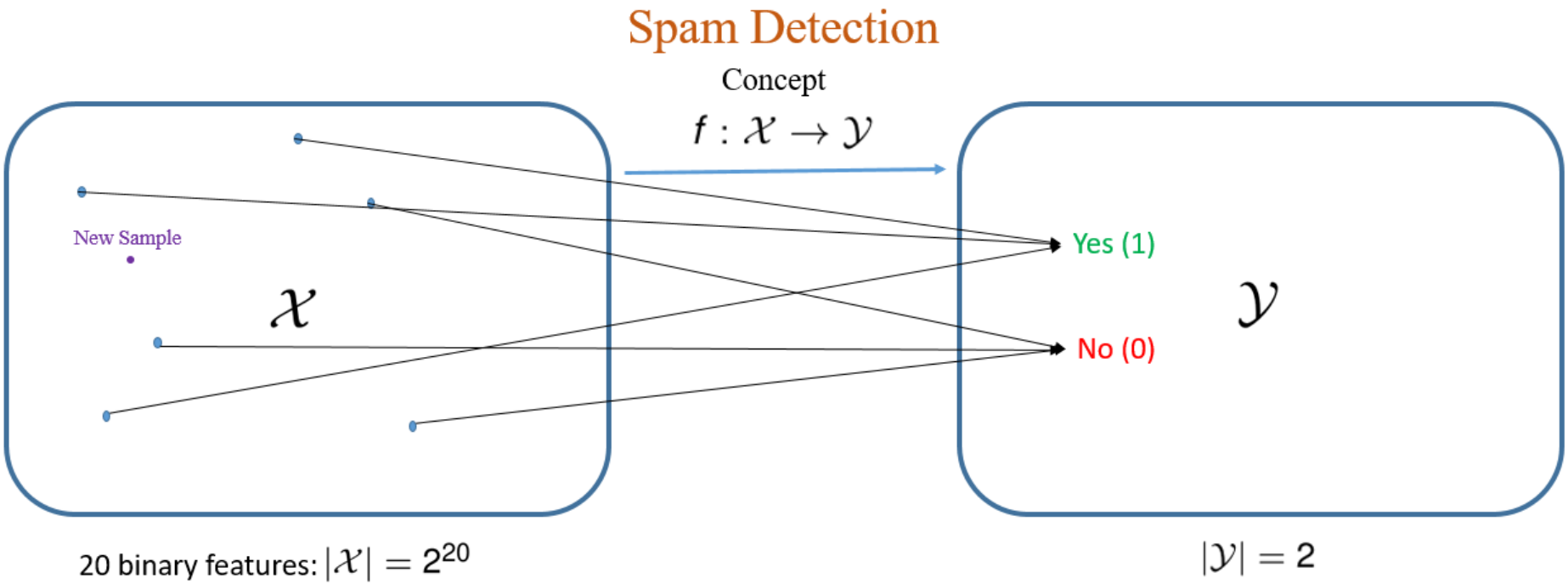


Components of supervised learning (function approximation)



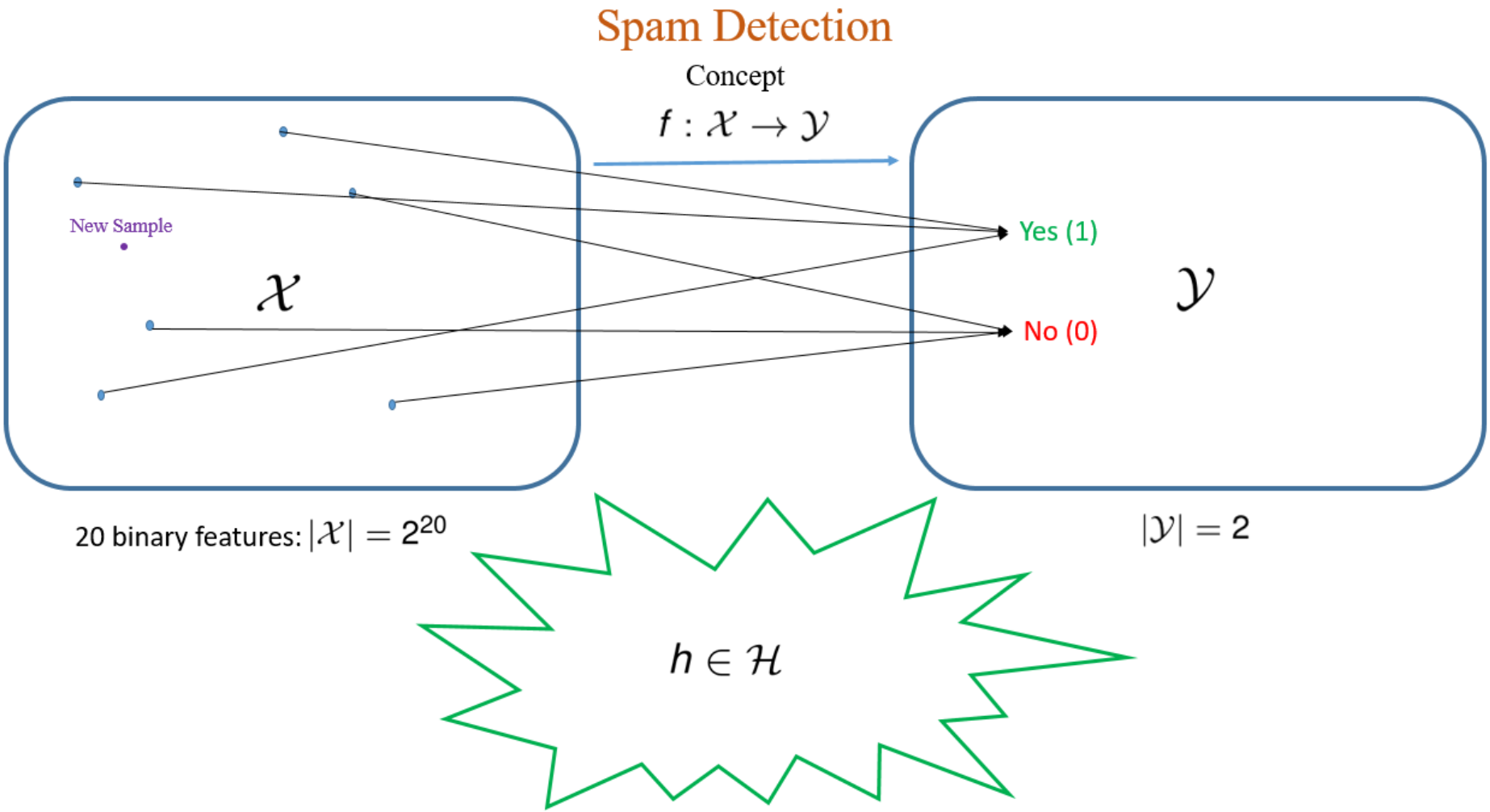
Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f

Components of supervised learning (function approximation)



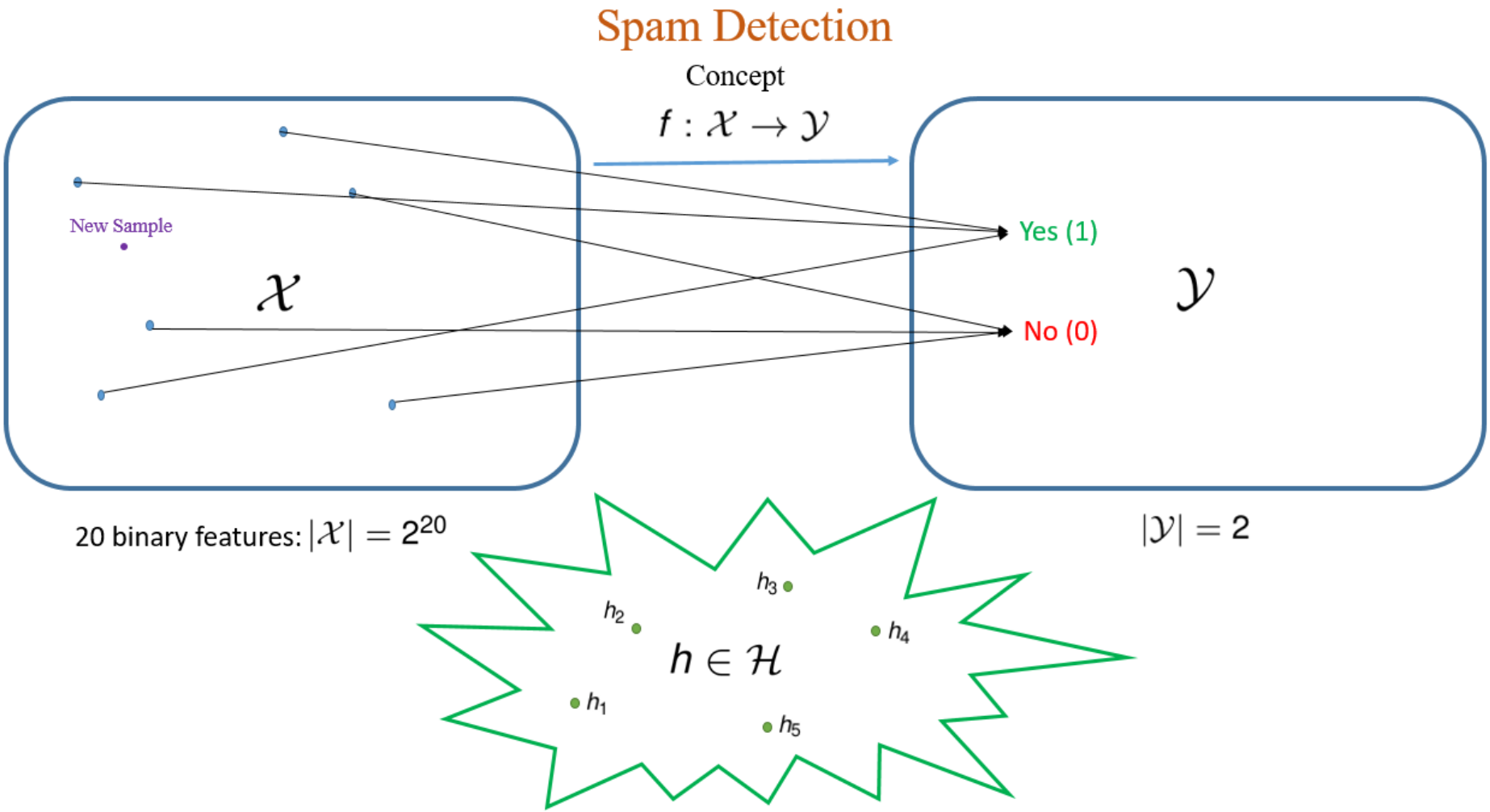
Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)



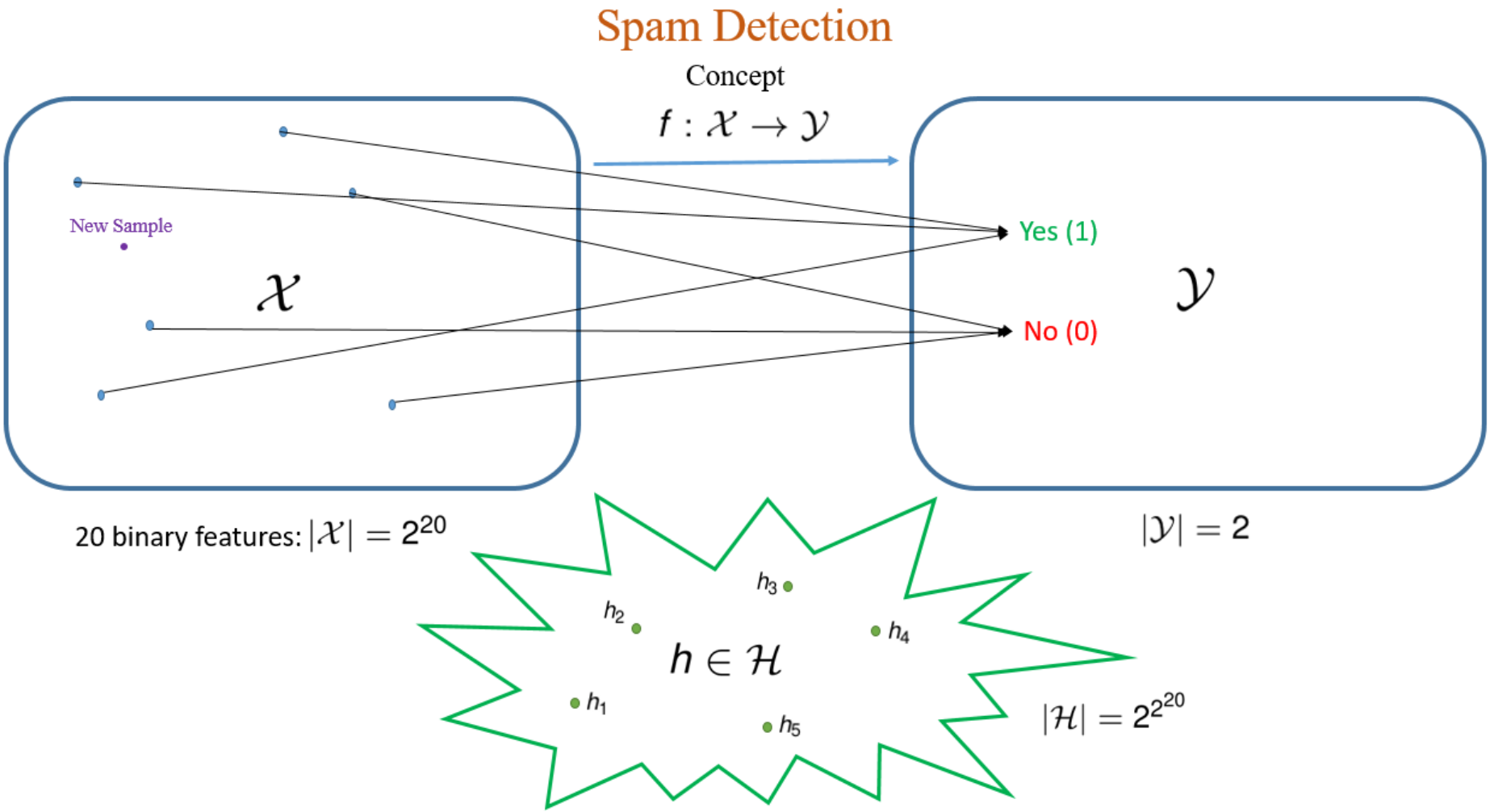
Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)



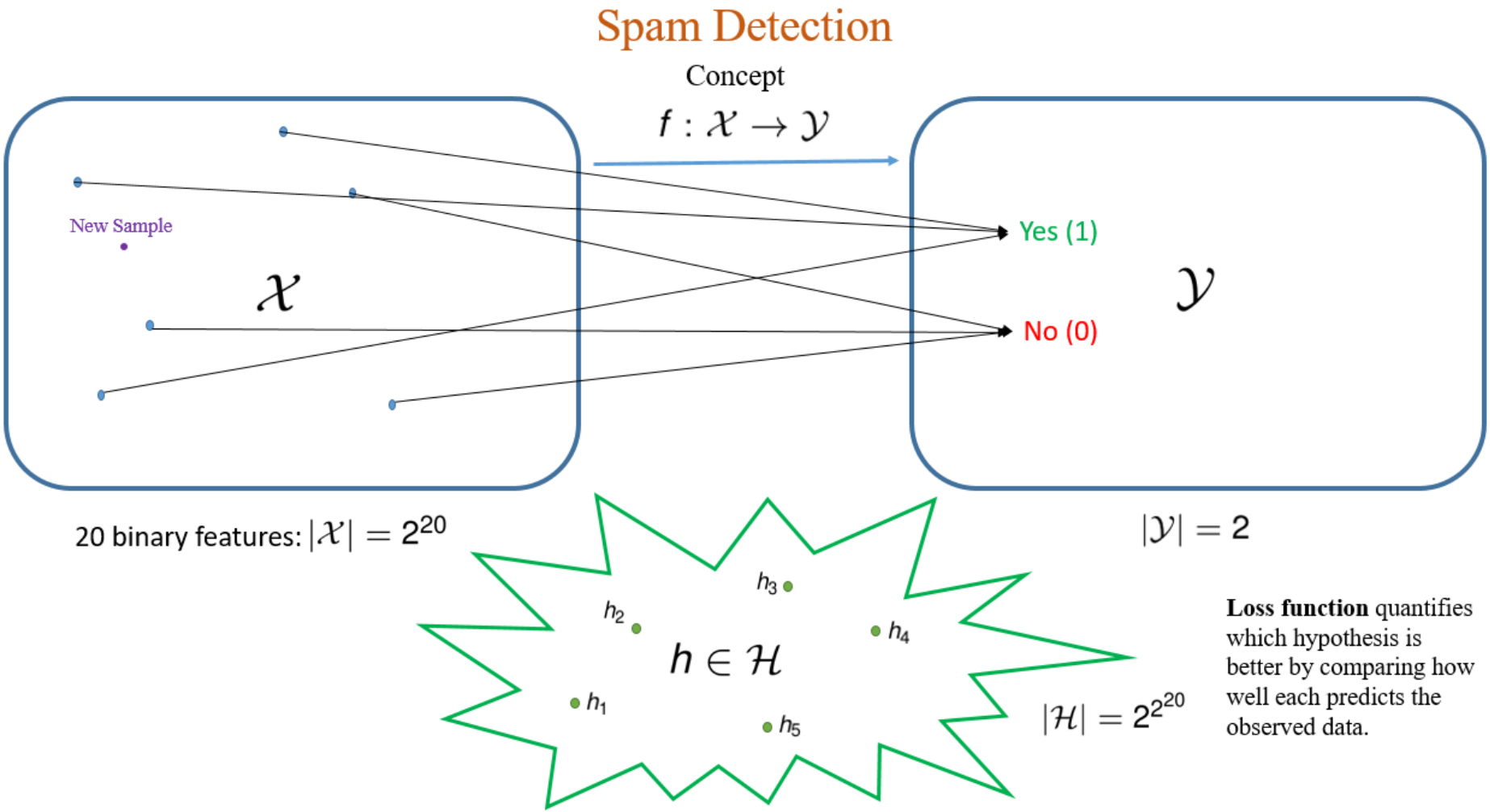
Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)

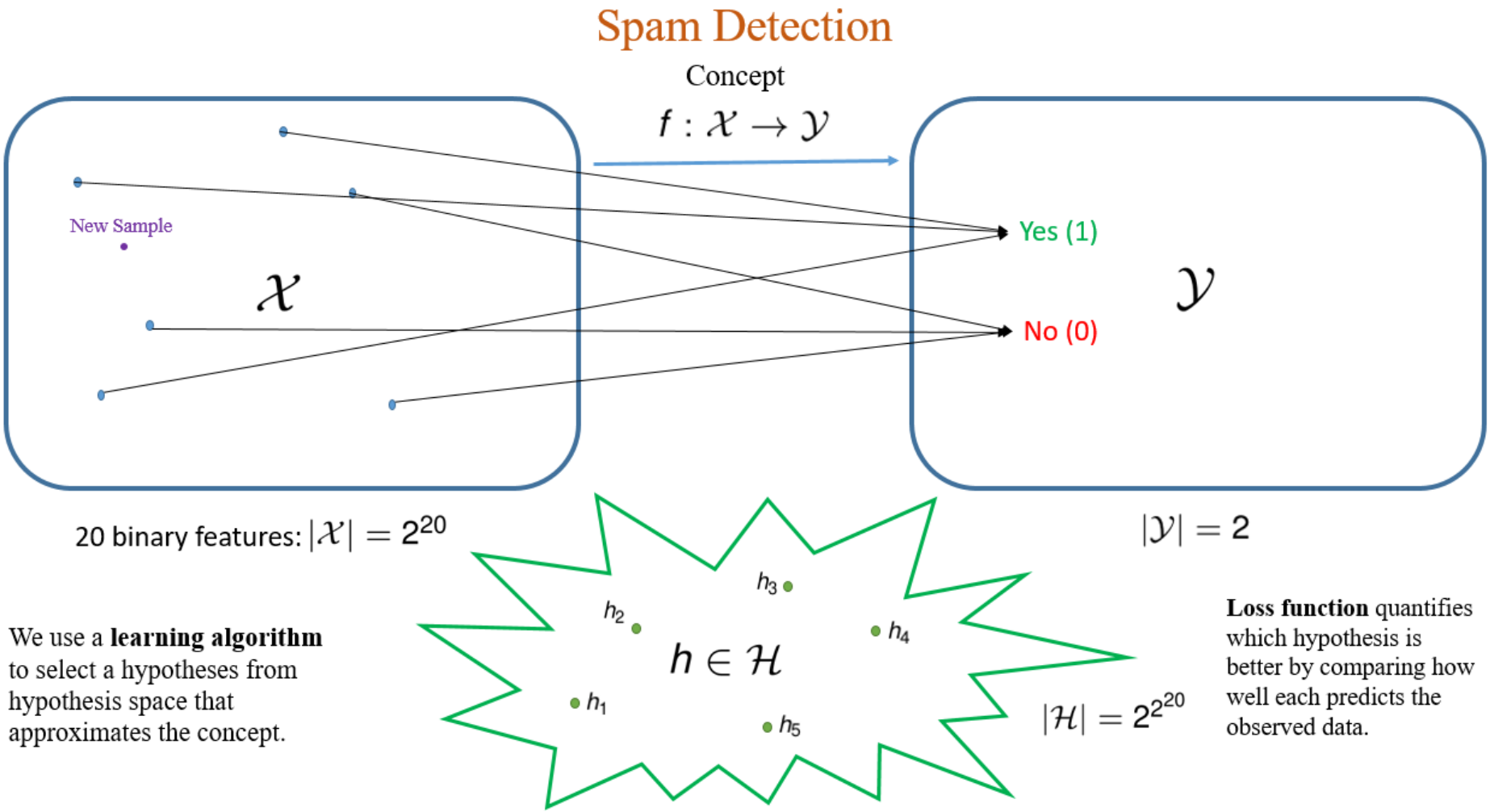


Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)



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)

Components of supervised learning (function approximation)

Concept

- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Training data

- Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Components of supervised learning (function approximation)

Concept

- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Training data

- Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Hypotheses space

- Pick a hypotheses $h : \mathcal{X} \to \mathcal{Y}$ from hypotheses space $h \in \mathcal{H}$ (e.g., linear functions) that approximates the target function f

Components of supervised learning (function approximation)

Concept

- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Training data

- Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Hypotheses space

- Pick a hypotheses $h : \mathcal{X} \to \mathcal{Y}$ from hypotheses space $h \in \mathcal{H}$ (e.g., linear functions) that approximates the target function f

Learning algorithm

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

Components of supervised learning

Learning model composed of

- A hypothesis set
- A learning algorithm

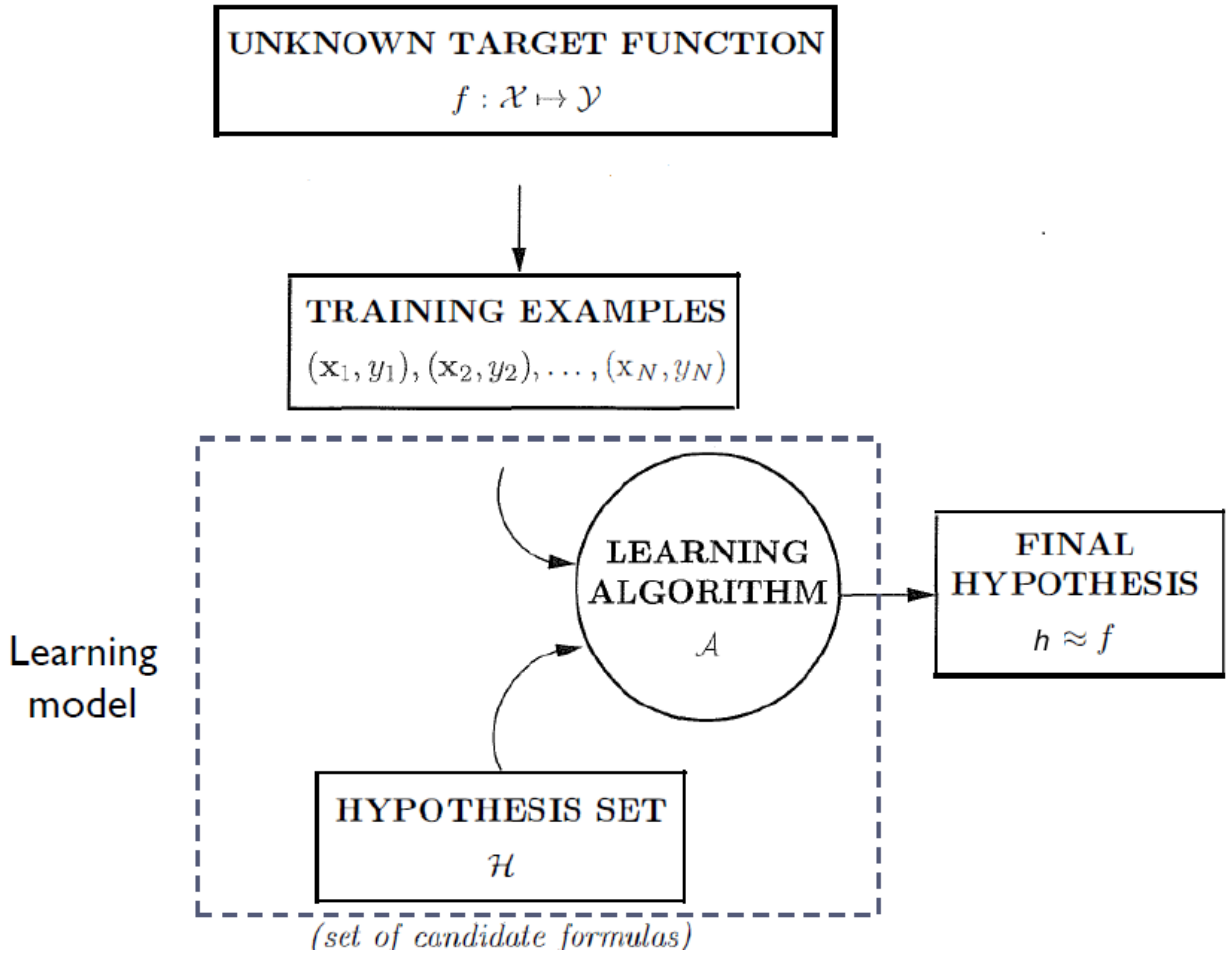


Figure: https://work.caltech.edu/telecourse. html

Generalization

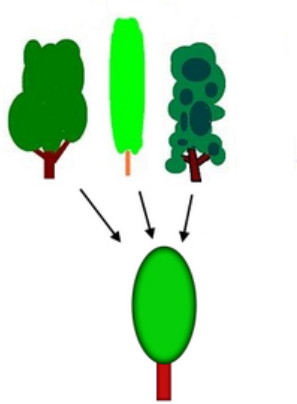
Hypotheses evaluation

How well h generalizes to unseen examples.

Test data

Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Do not exist in the training data



Generalization

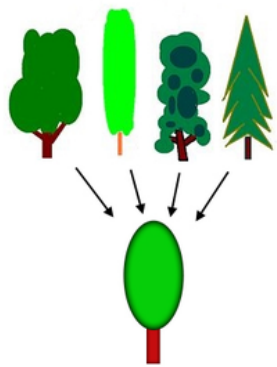
Hypotheses evaluation

How well h generalizes to unseen examples.

Test data

Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Do not exist in the training data



Independent and identically distributed (i.i.d.)

A collection of random variables is independent and identically distributed if each random variable has the same probability distribution as the others and all are mutually independent.

- $\forall i, x^{(i)} \sim \mathcal{D}$ (Identically Distributed)
- $\forall i \neq j, \mathcal{P}(x^{(i)}, x^{(j)}) = \mathcal{P}(x^{(i)})\mathcal{P}(x^{(j)})$ (Independently Distributed)

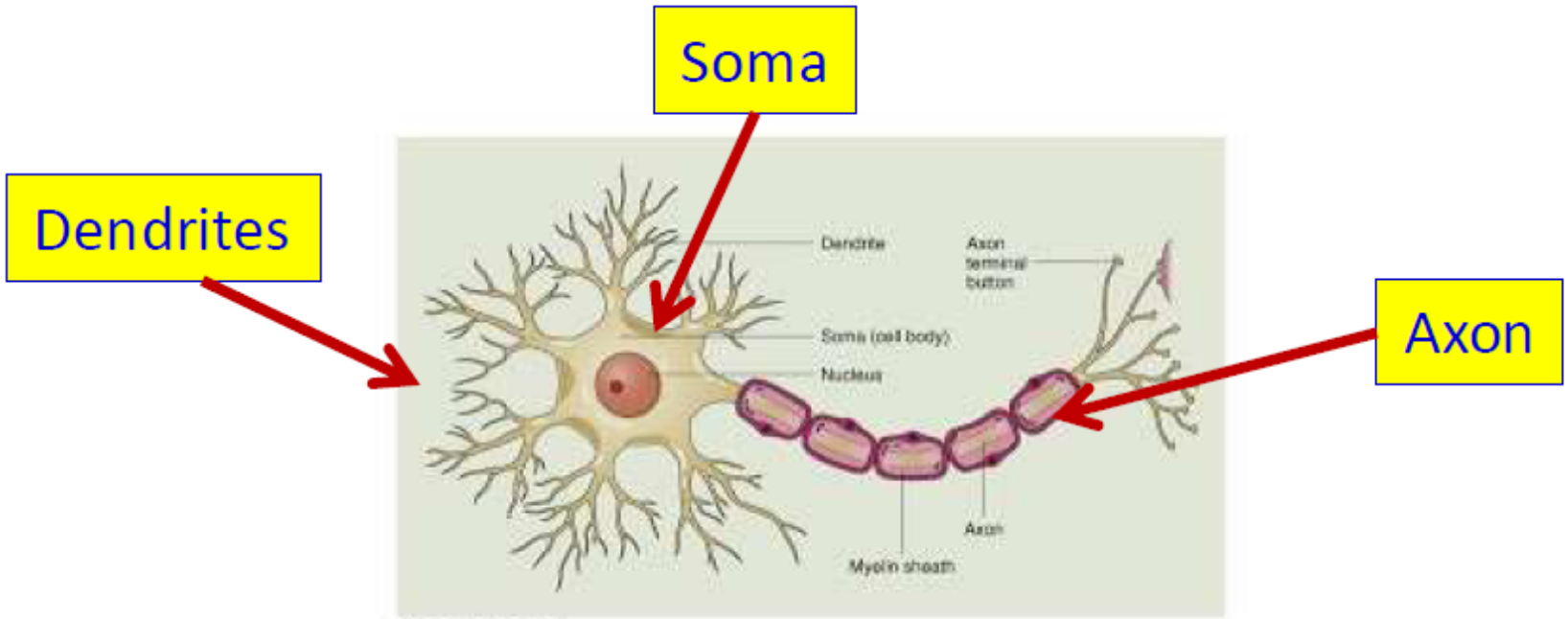
Machine learning algorithms have focused primarily on sets of data points that were assumed to be independent and identically distributed (i.i.d.).

Neural Networks

Modeling the brain

A neuron

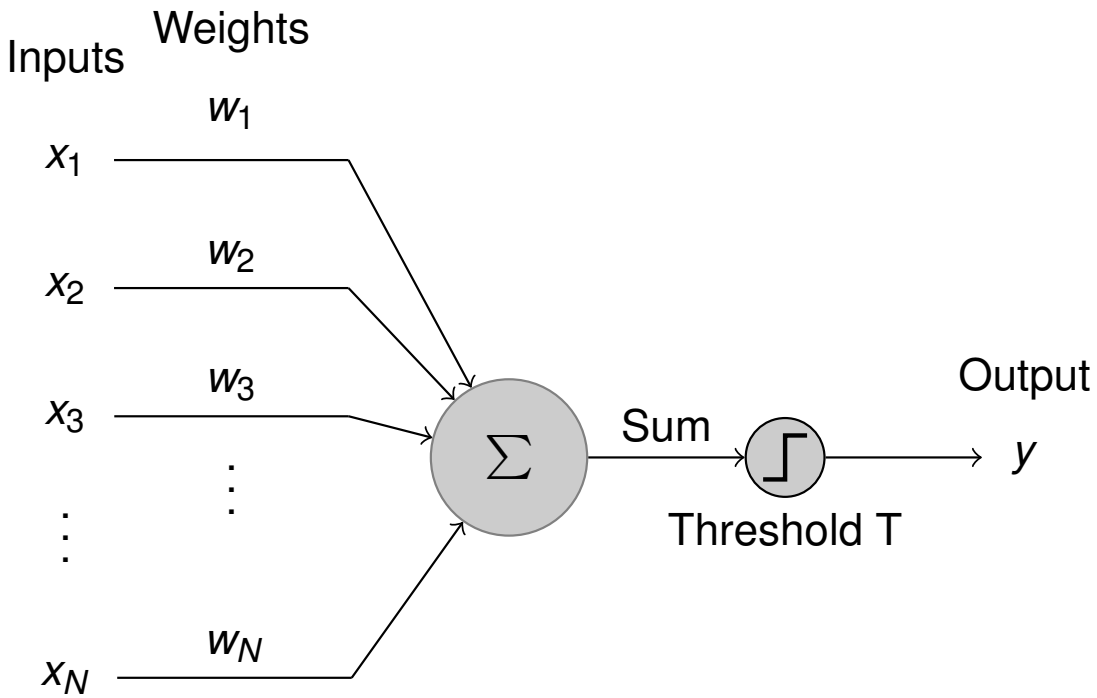
- Signals come in through the dendrites into the Soma
- A signal goes out via the axon to other neurons
  - Only one axon per neuron



Perceptron

Number of inputs combine linearly

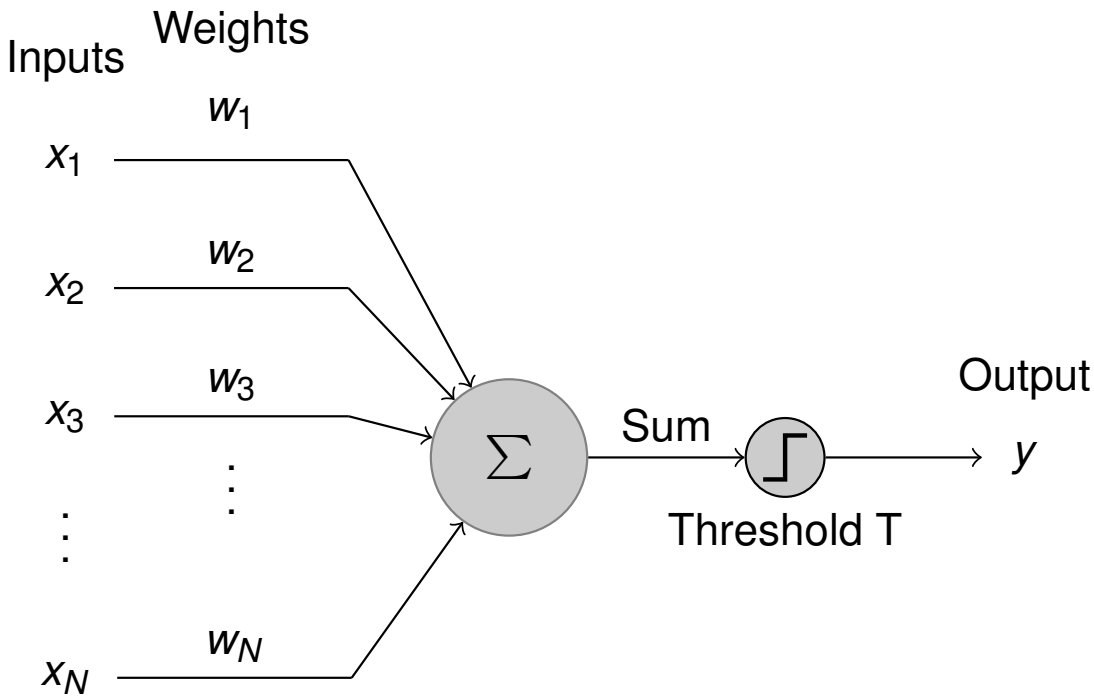
- Threshold logic: Fire if combined input exceeds threshold



Perceptron

Number of inputs combine linearly

- Threshold logic: Fire if combined input exceeds threshold

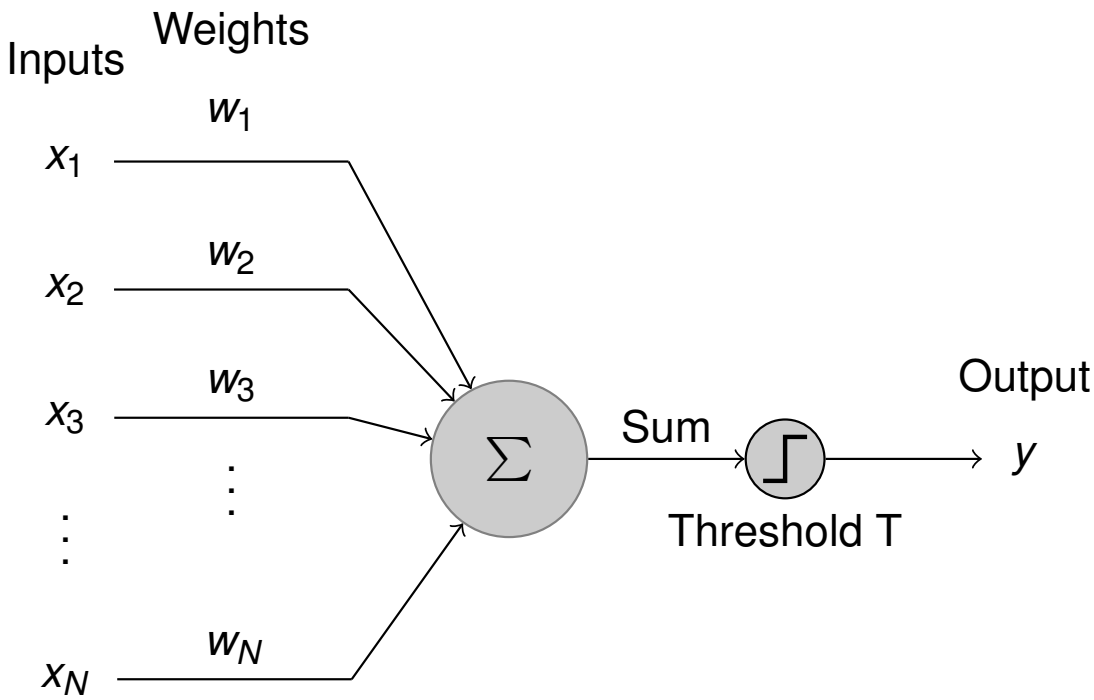


$$y = \begin{cases} 1 & \text{if} \quad \sum_i w_i x_i - T \geq 0 \\ 0 & \text{else} \end{cases}$$

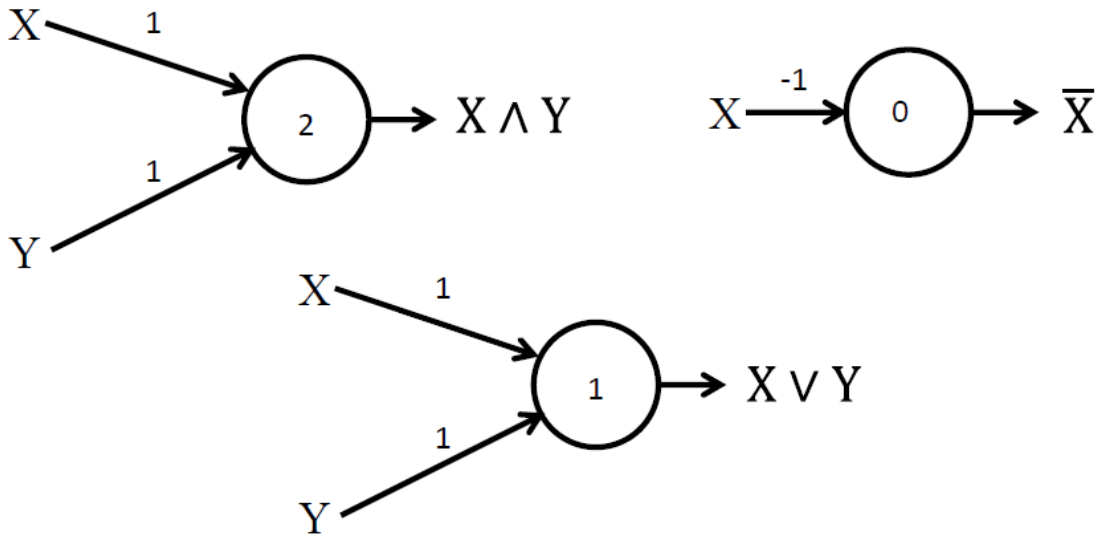
The universal model

Originally assumed could represent any Boolean circuit and perform any logic

- The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence, New York Times (8 July) 1958
- Frankenstein Monster Designed by Navy That Thinks, Tulsa, Oklahoma Times 1958



Perceptron

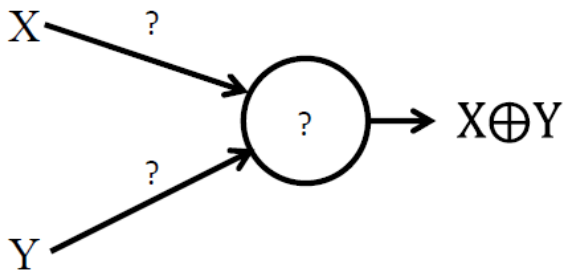


Values shown on edges are weights, numbers in the circles are thresholds ($X, Y \in \{0, 1\}$)

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

Perceptron

No solution for XOR!
Not universal!

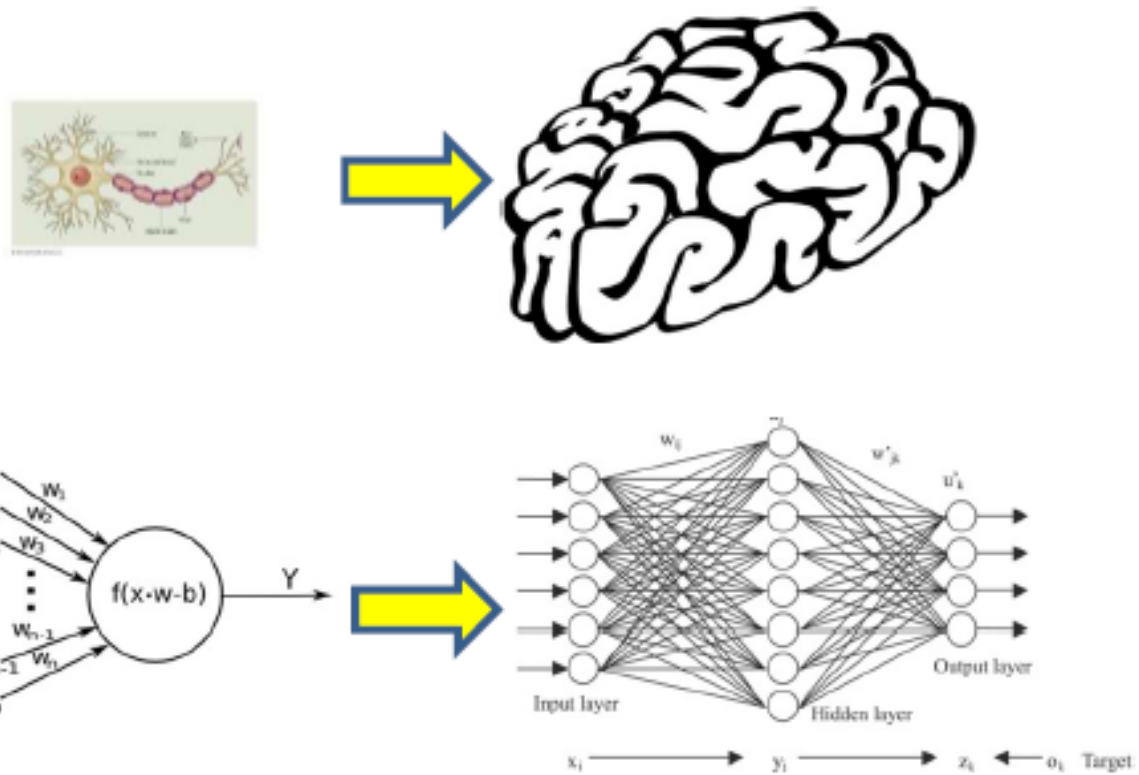


Values shown on edges are weights, numbers in the circles are thresholds ($X, Y \in \{0, 1\}$)

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

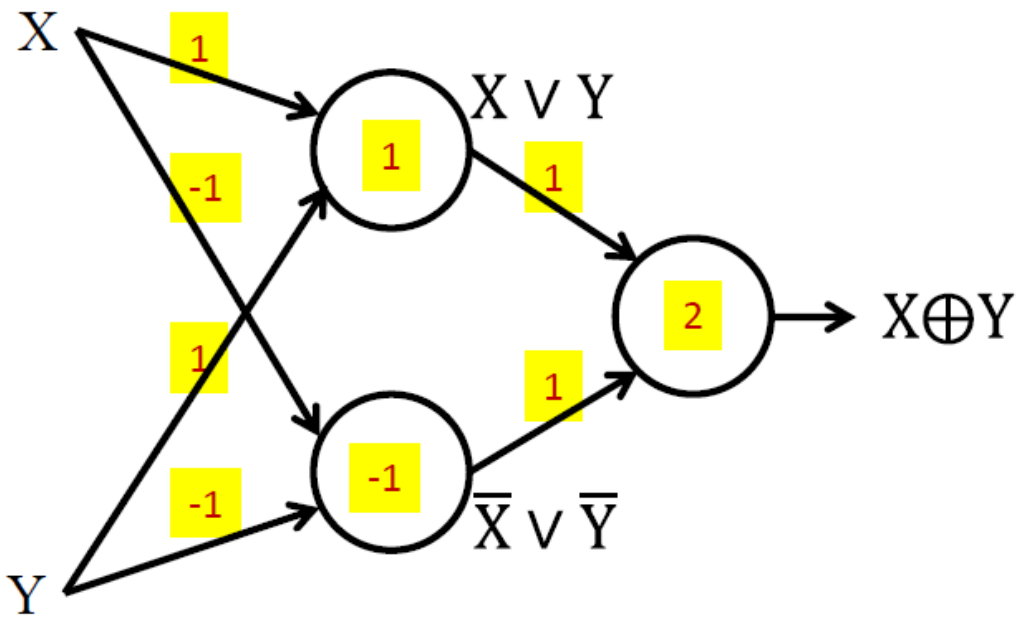
A single neuron is not enough

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



Multi-layer perceptron!

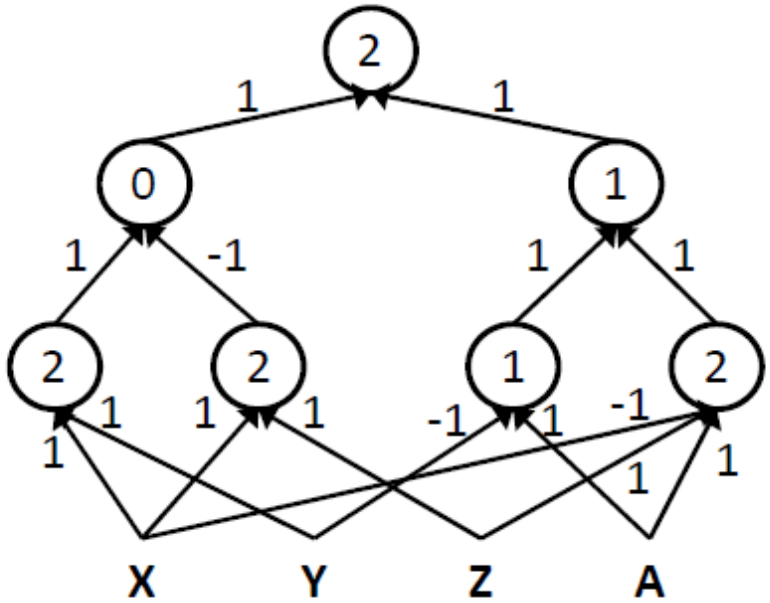
XOR



Multi-layer perceptrons (MLPs)

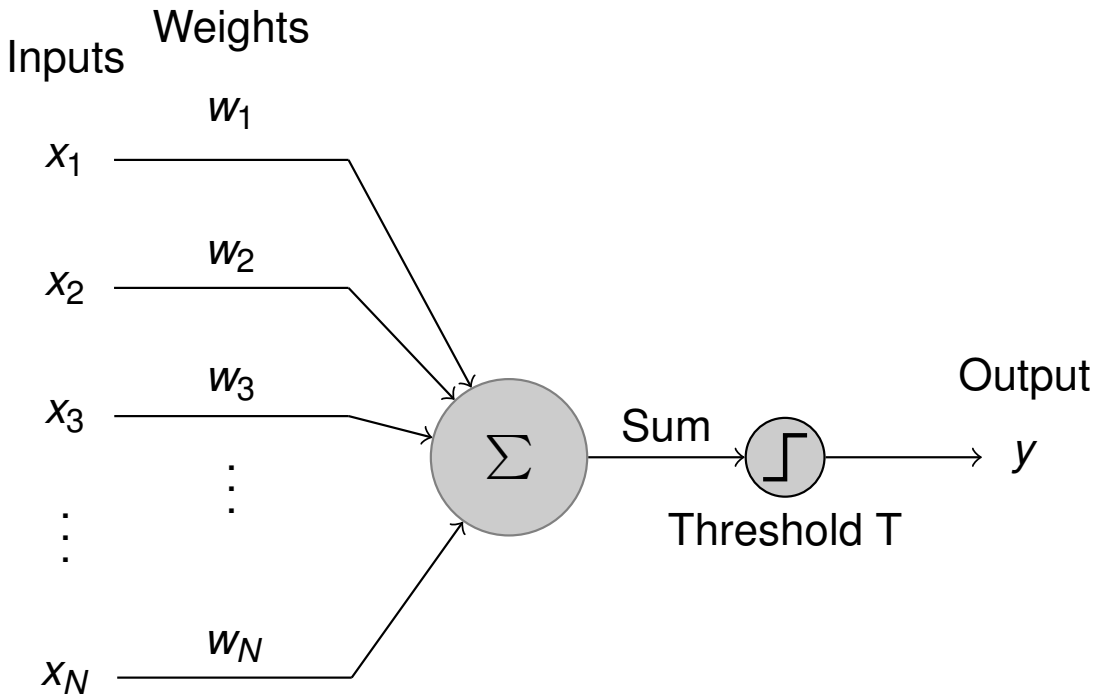
- MLPs can compute *any* Boolean function
  - Any function over any number of inputs and any number of outputs

$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$



The perceptron with real inputs

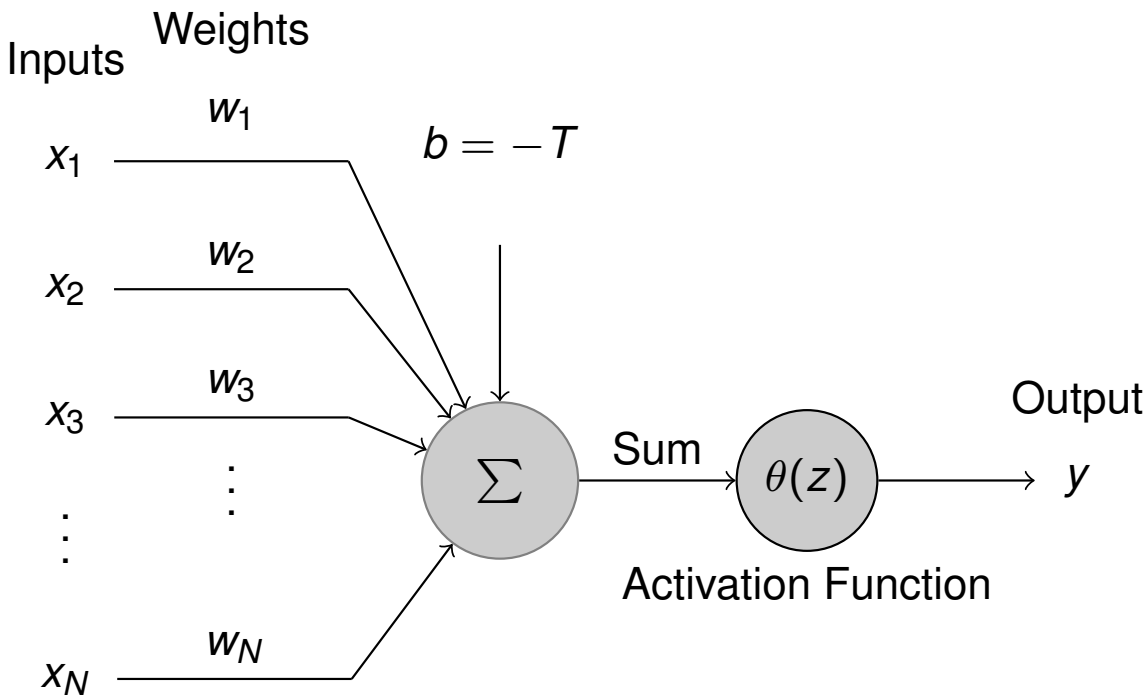
- $x_1, x_2, \ldots, x_n$ are real valued
- $w_1, w_2, \ldots, w_n$ are real valued
- Unit fires if weighted input exceeds a threshold



$$y = \begin{cases} 1 & \text{if} \quad \sum_i w_i x_i - T \geq 0 \\ 0 & \text{else} \end{cases}$$

The perceptron with real inputs

- An alternate view:
  - A threshold activation $\theta(z)$ operates on the weighted sum of inputs plus a bias
  - $\theta(z)$ outputs 1 if z is non-negative, 0 otherwise
- Unit fires if weighted input exceeds a threshold



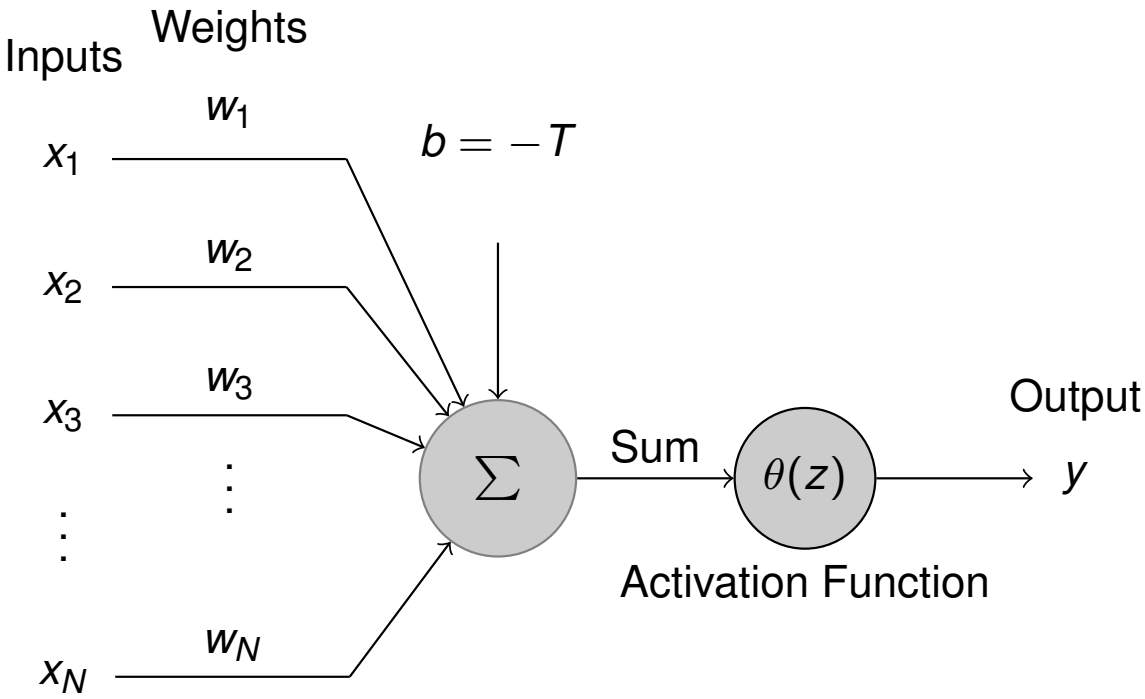
$$z = \sum_i w_i x_i + b$$

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

Perceptron

The bias can also be viewed as the weight of another input component that is always set to 1

- If the bias is not explicitly mentioned, we will implicitly be assuming that every perceptron has an additional input that is always fixed at 1



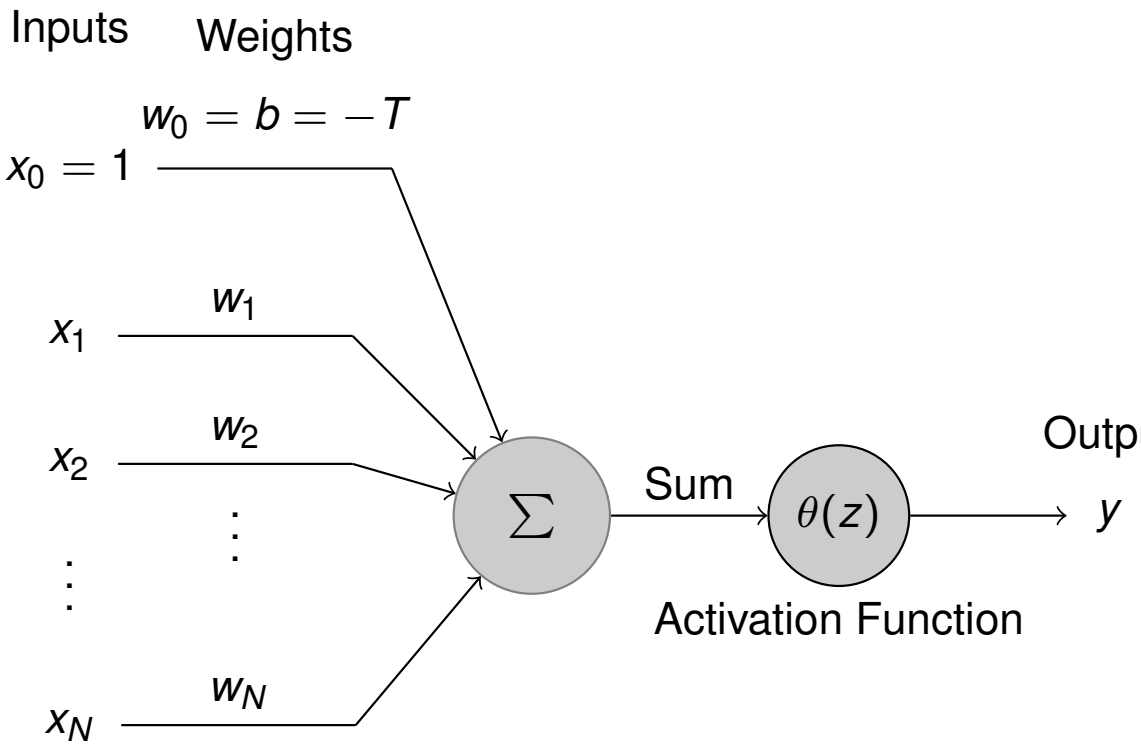
$$z = \sum_i w_i x_i + b$$

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

Perceptron

The bias can also be viewed as the weight of another input component that is always set to 1

- If the bias is not explicitly mentioned, we will implicitly be assuming that every perceptron has an additional input that is always fixed at 1

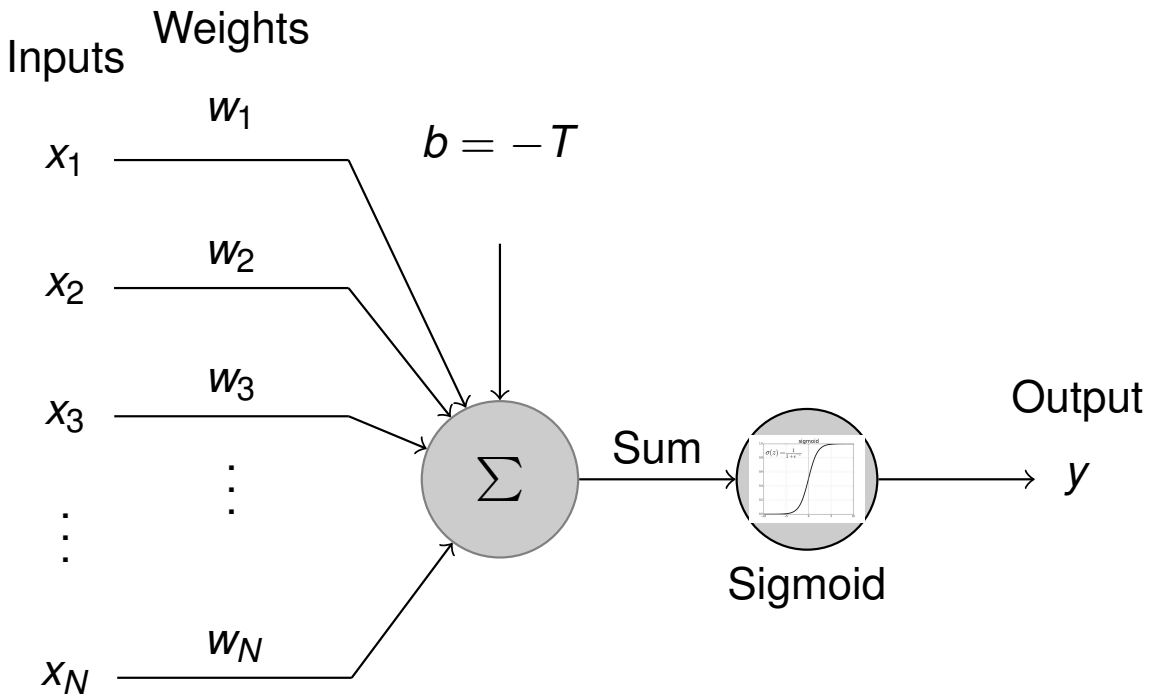


$$z = \sum_{i=1}^{N} w_i x_i + b \text{ or } z = \sum_{i=0}^{N} w_i x_i$$

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

A real output

- An alternate view:
  - A threshold activation $\theta(z)$ operates on the weighted sum of inputs plus a bias
  - $\theta(z)$ outputs 1 if z is non-negative, 0 otherwise
- The output y can also be real valued
  - Sometimes viewed as the probability of firing
- Logistic Regression



$$y = \sigma(\sum_i w_i x_i + b)$$

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

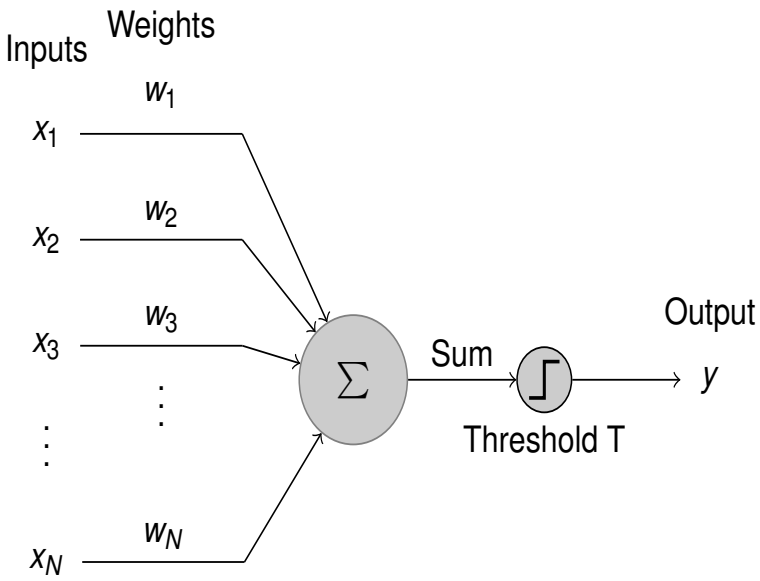
Vectorization

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

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

Perceptron

A perceptron is a linear classifier

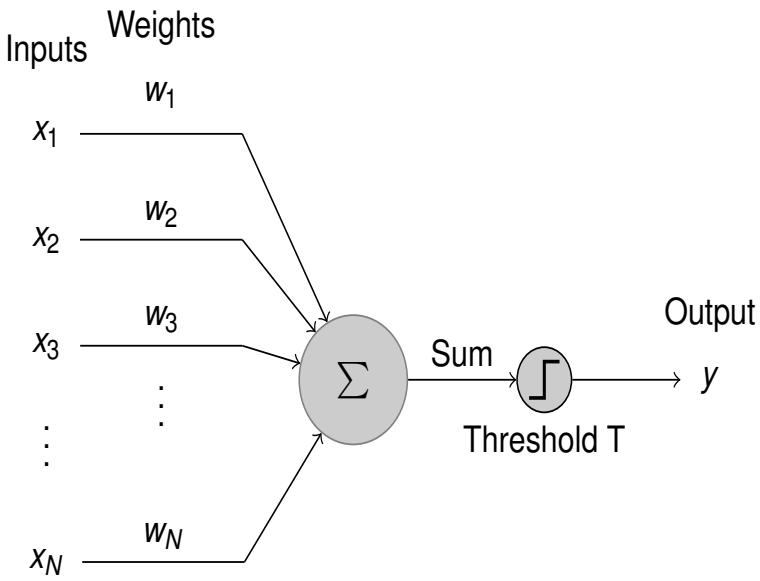


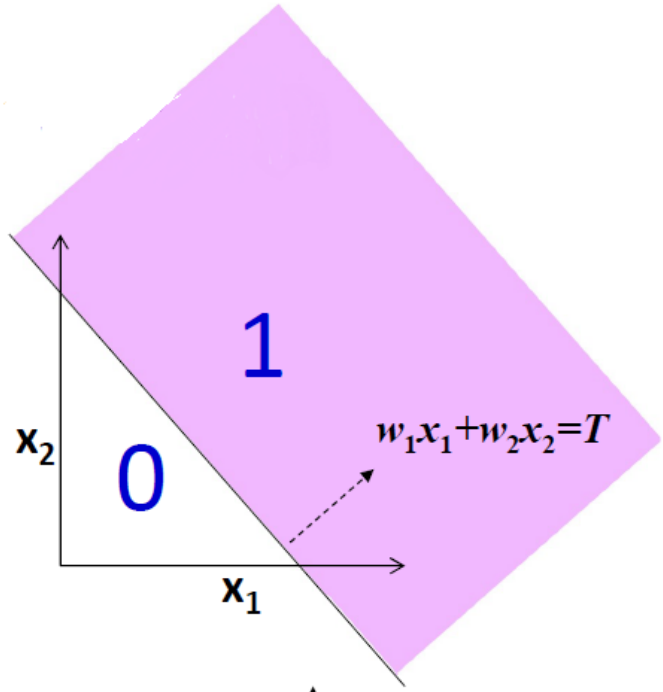
$$y = \theta(\sum_i w_i x_i - T)$$

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

Perceptron

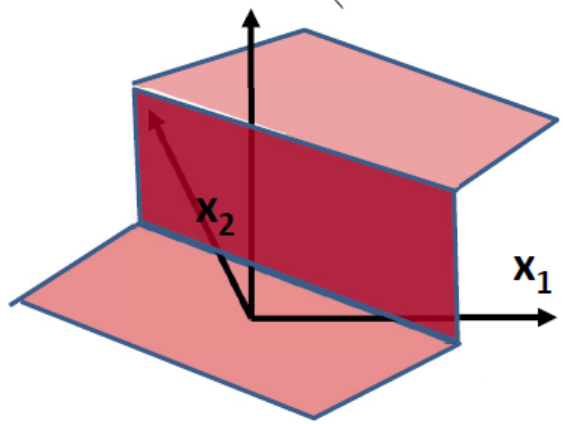
A perceptron is a linear classifier





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

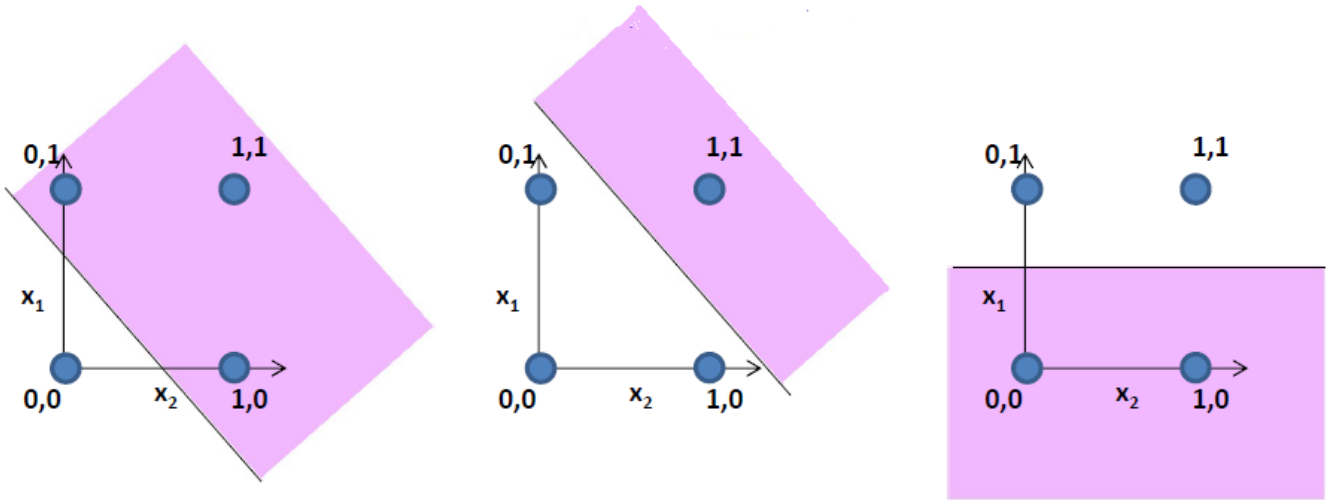
$$\theta(z) = \begin{cases} 1 & \text{if} \quad z \geq 0 \\ 0 & \text{else} \end{cases}$$



Boolean functions with a real perceptron

Boolean perceptrons are also linear classifiers

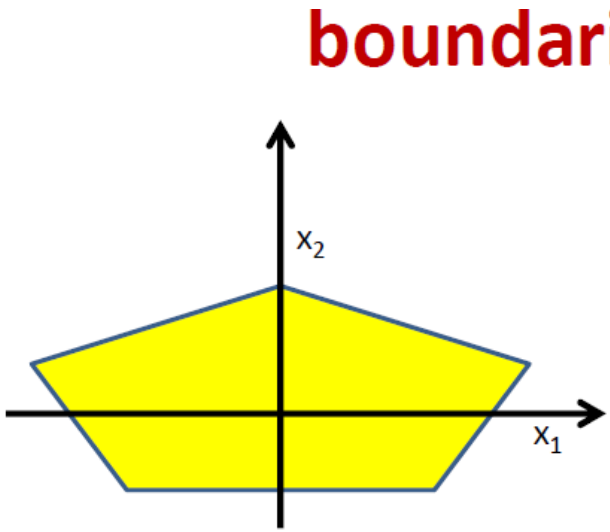
- Purple regions have output 1 in the figures
- What are these functions?
- Why can we not compose an XOR?



A convex region

Build a network of units with a single output that fires if the input is in the coloured area

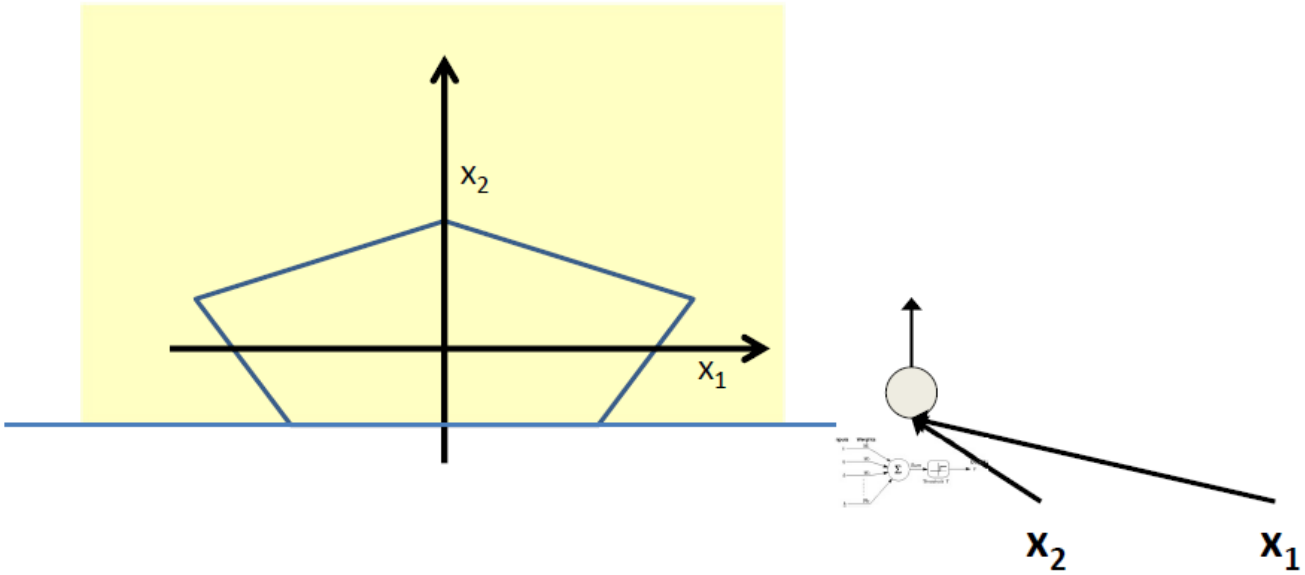
- Neurons can now be composed into networks to compute arbitrary classification boundaries



A convex region

Build a network of units with a single output that fires if the input is in the coloured area

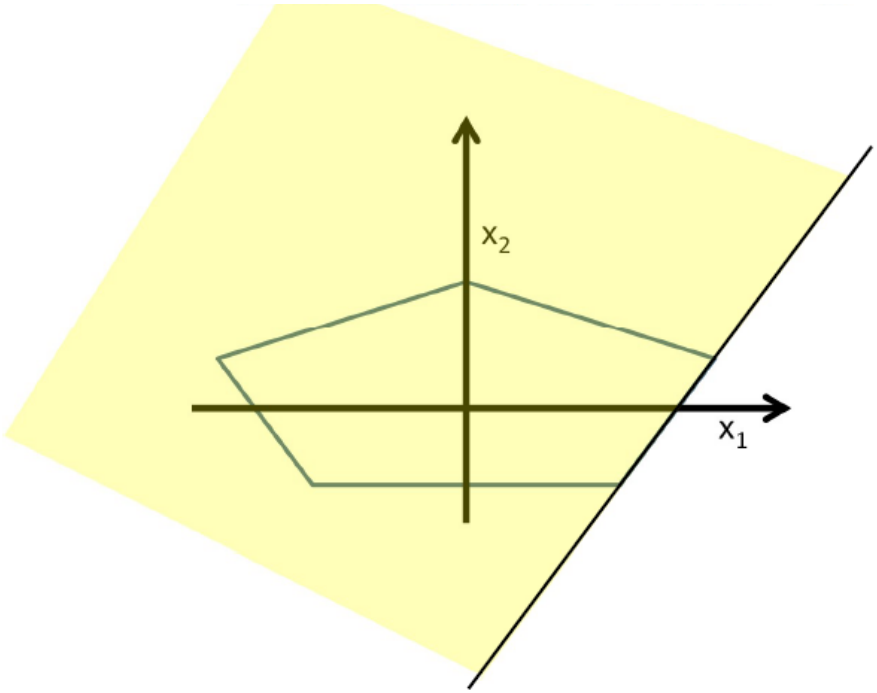
- Neurons can now be composed into networks to compute arbitrary classification boundaries

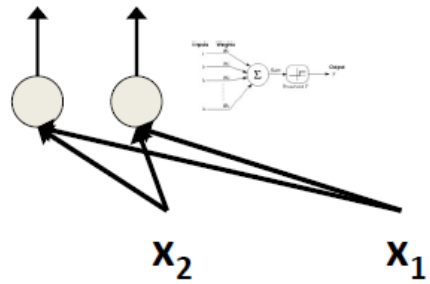


A convex region

Build a network of units with a single output that fires if the input is in the coloured area

- Neurons can now be composed into networks to compute arbitrary classification boundaries

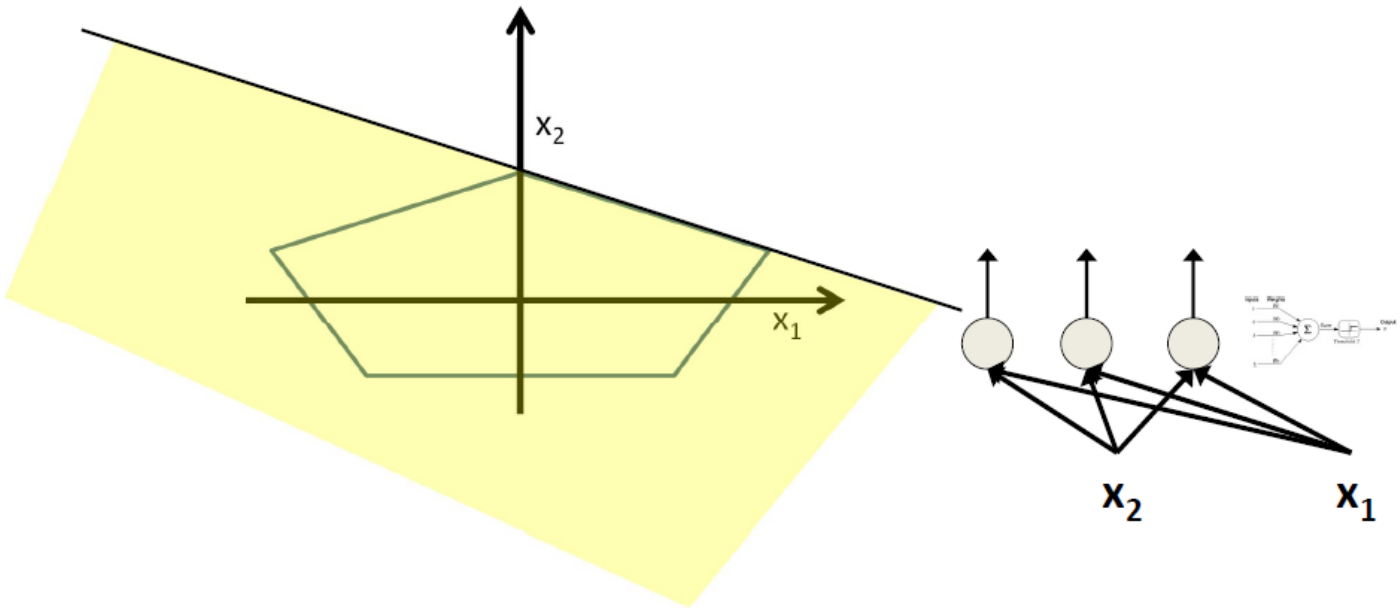




A convex region

Build a network of units with a single output that fires if the input is in the coloured area

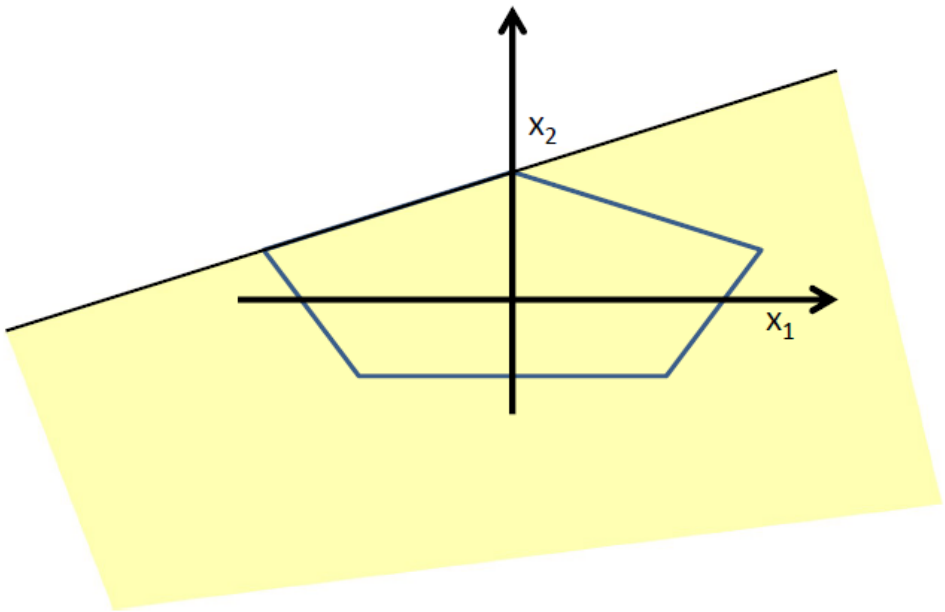
- Neurons can now be composed into networks to compute arbitrary classification boundaries

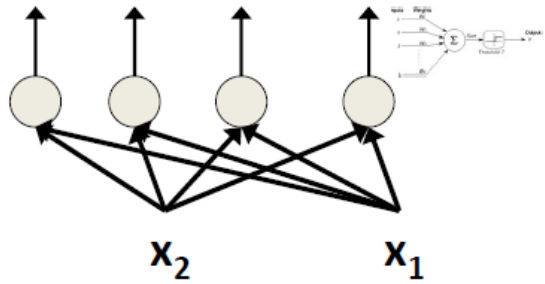


A convex region

Build a network of units with a single output that fires if the input is in the coloured area

- Neurons can now be composed into networks to compute arbitrary classification boundaries

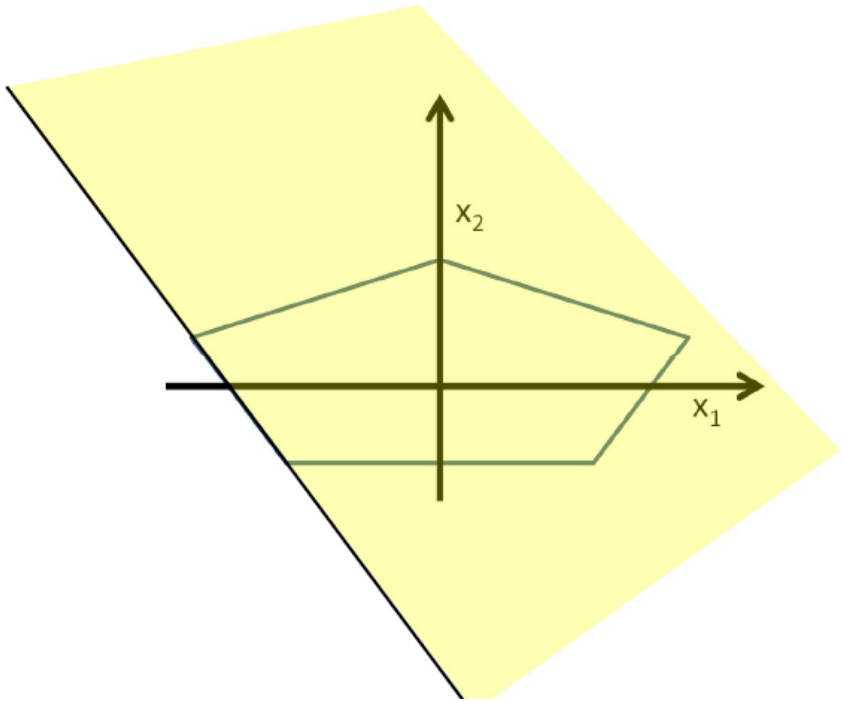


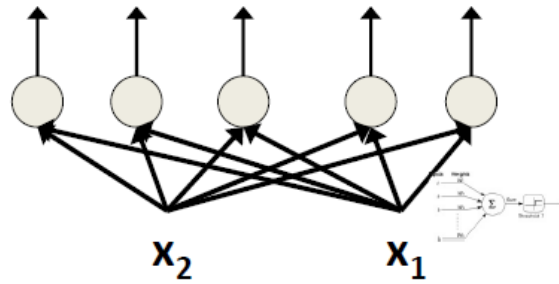


A convex region

Build a network of units with a single output that fires if the input is in the coloured area

- Neurons can now be composed into networks to compute arbitrary classification boundaries

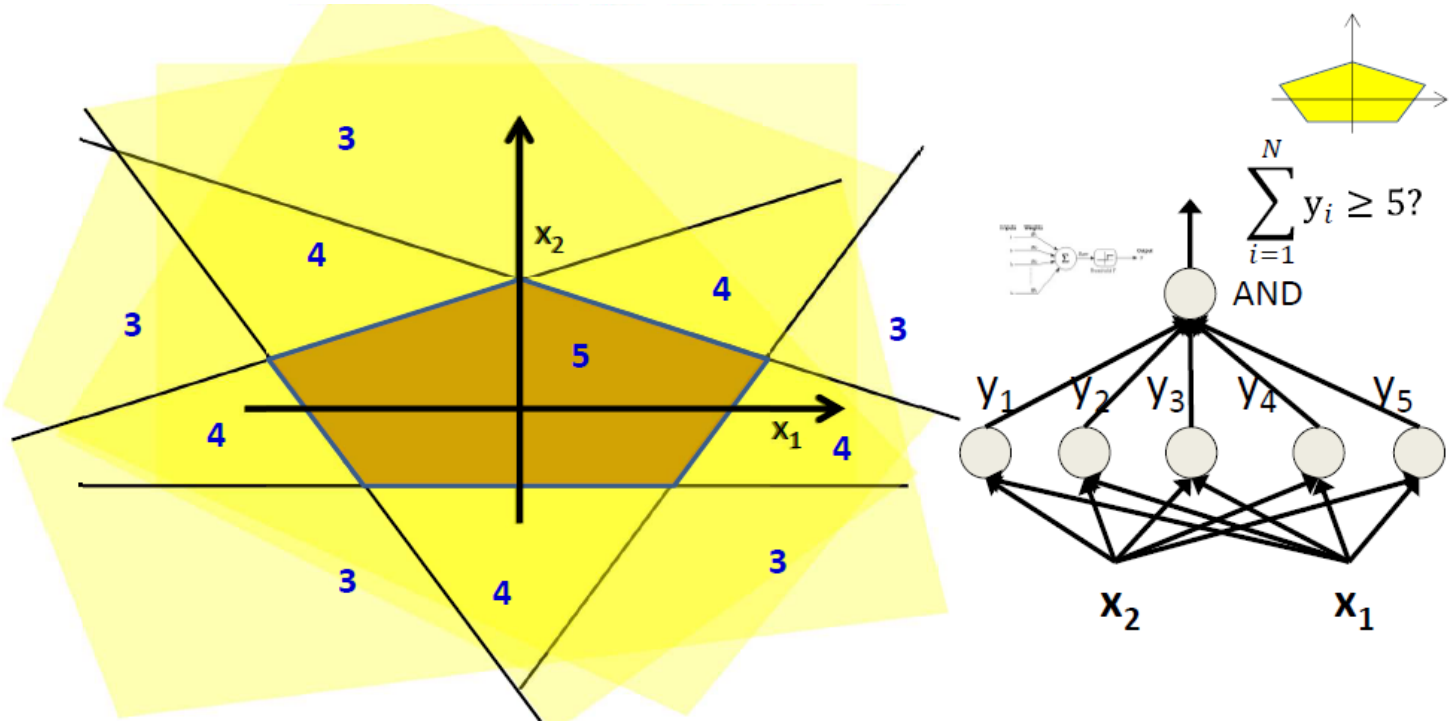




A convex region

Build a network of units with a single output that fires if the input is in the coloured area

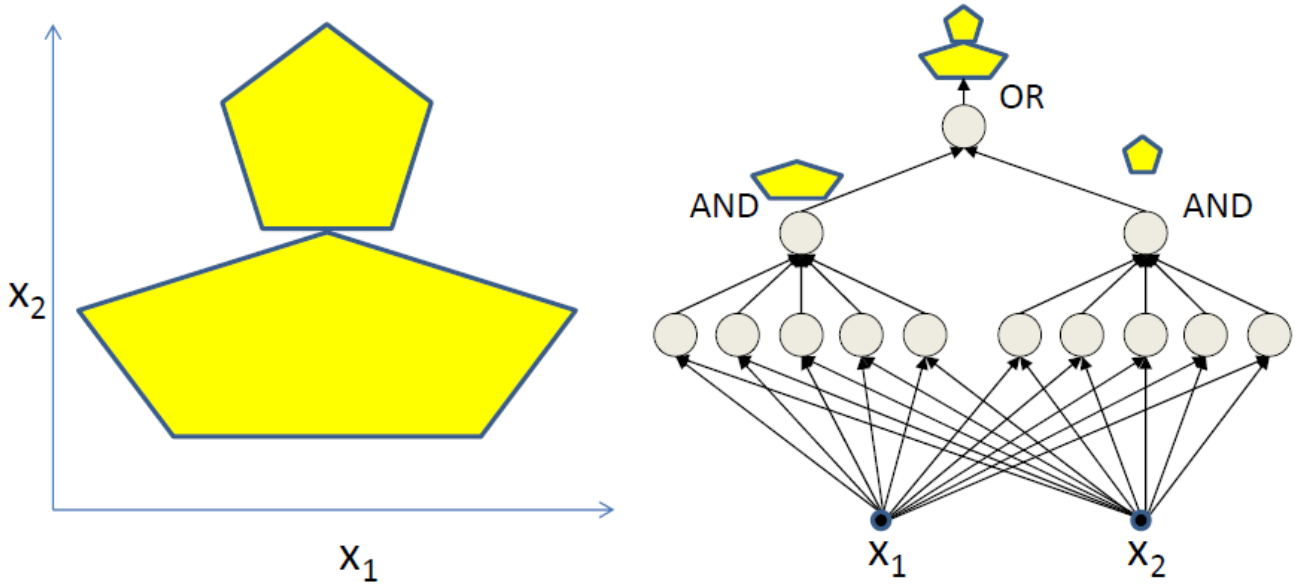
- Neurons can now be composed into networks to compute arbitrary classification boundaries



A convex region

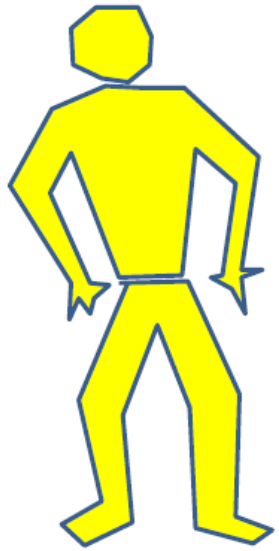
Build a network of units with a single output that fires if the input is in the coloured area

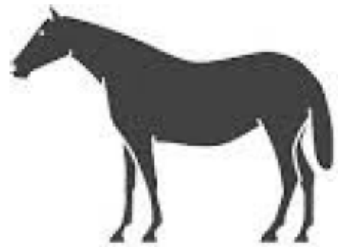
- Neurons can now be composed into networks to compute arbitrary classification boundaries



Complex decision boundaries

Can compose very complex decision boundaries

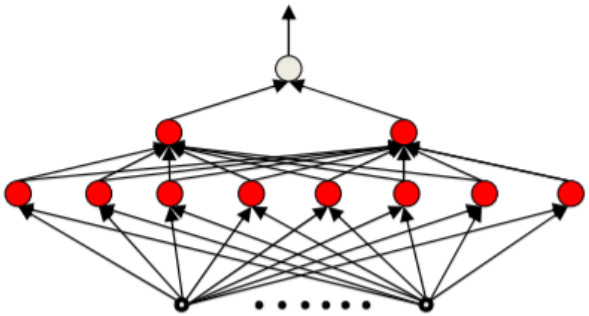


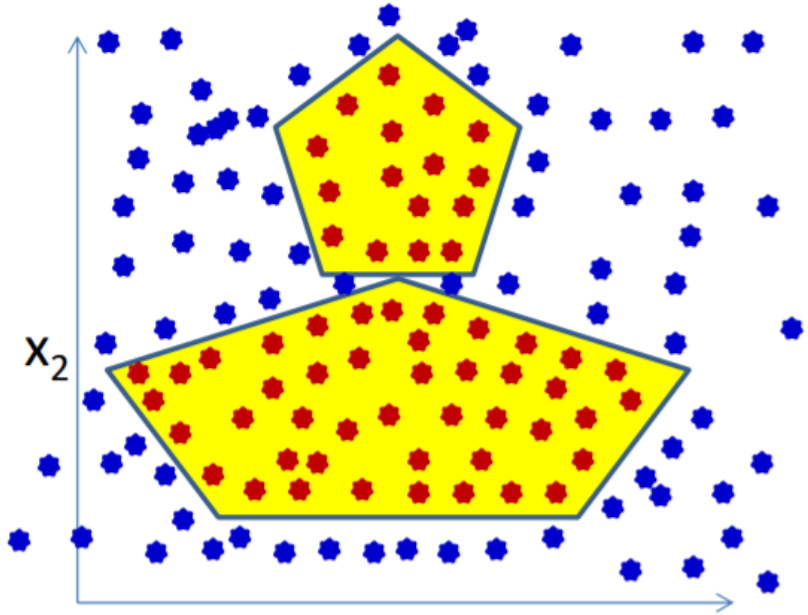


A visual proof that neural nets can compute any function

A real problem

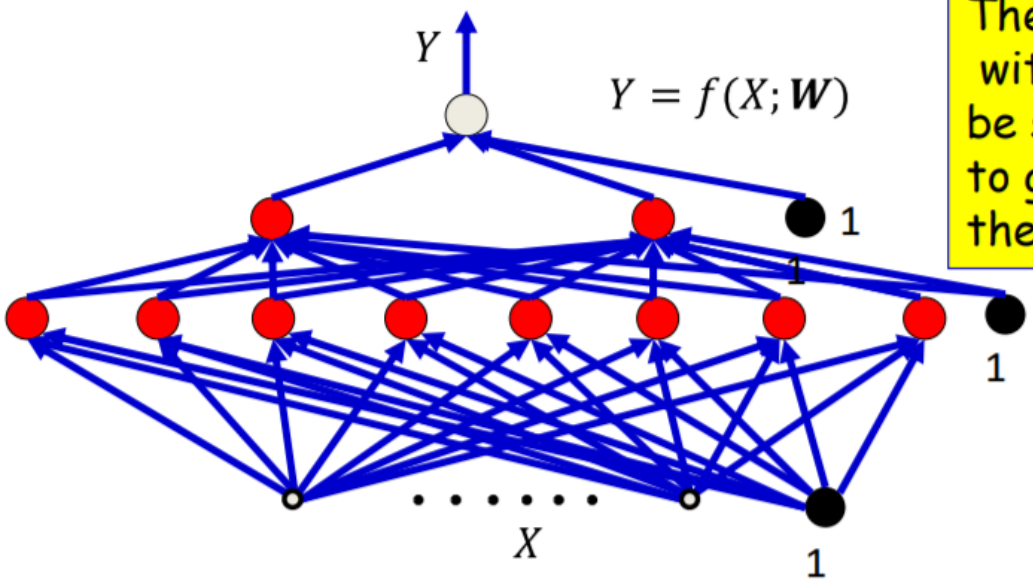
- Learn an MLP for this function
  - 1 in the yellow regions, 0 outside
- Using just the samples
- We know this can be perfectly represented using an MLP





What we learn: The parameters of the network

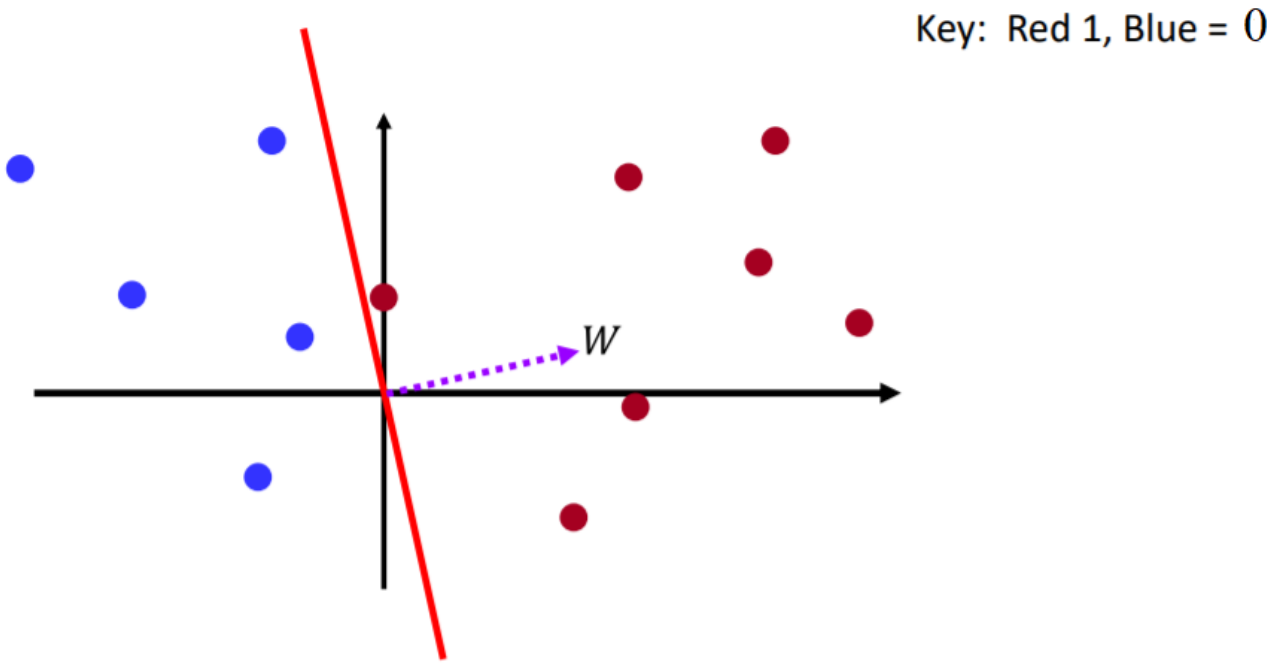
- **Given:** the architecture of the network
- **The parameters of the network:** The weights and biases
  - The weights associated with the blue arrows in the picture
- **Learning the network:** Determining the values of these parameters such that the network computes the desired function



The network is a function f() with parameters W which must be set to the appropriate values to get the desired behavior from the net

The Perceptron Problem

- Find the hyperplane $\boldsymbol{w}^T\boldsymbol{x} = 0$ that perfectly separates the two groups of points
  - $\boldsymbol{w}^T\boldsymbol{x} = 0$ is the hyperplane comprising all $\boldsymbol{x}$'s orthogonal to vector $\boldsymbol{w}$
  - Learning the perceptron $=$ finding the weight vector $\boldsymbol{w}$ for the separating hyperplane
  - $\boldsymbol{w}$ points in the direction of the positive class

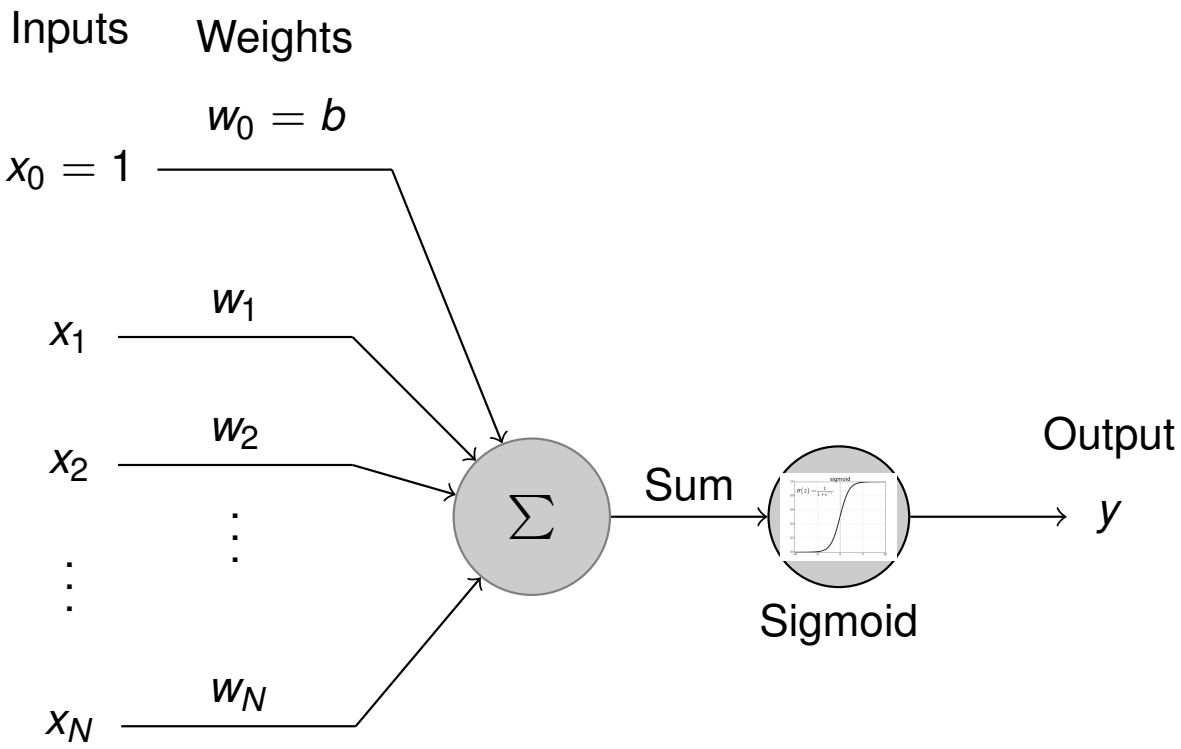


Logistic Regression

Logistic regression

It is the perceptron with a sigmoid activation

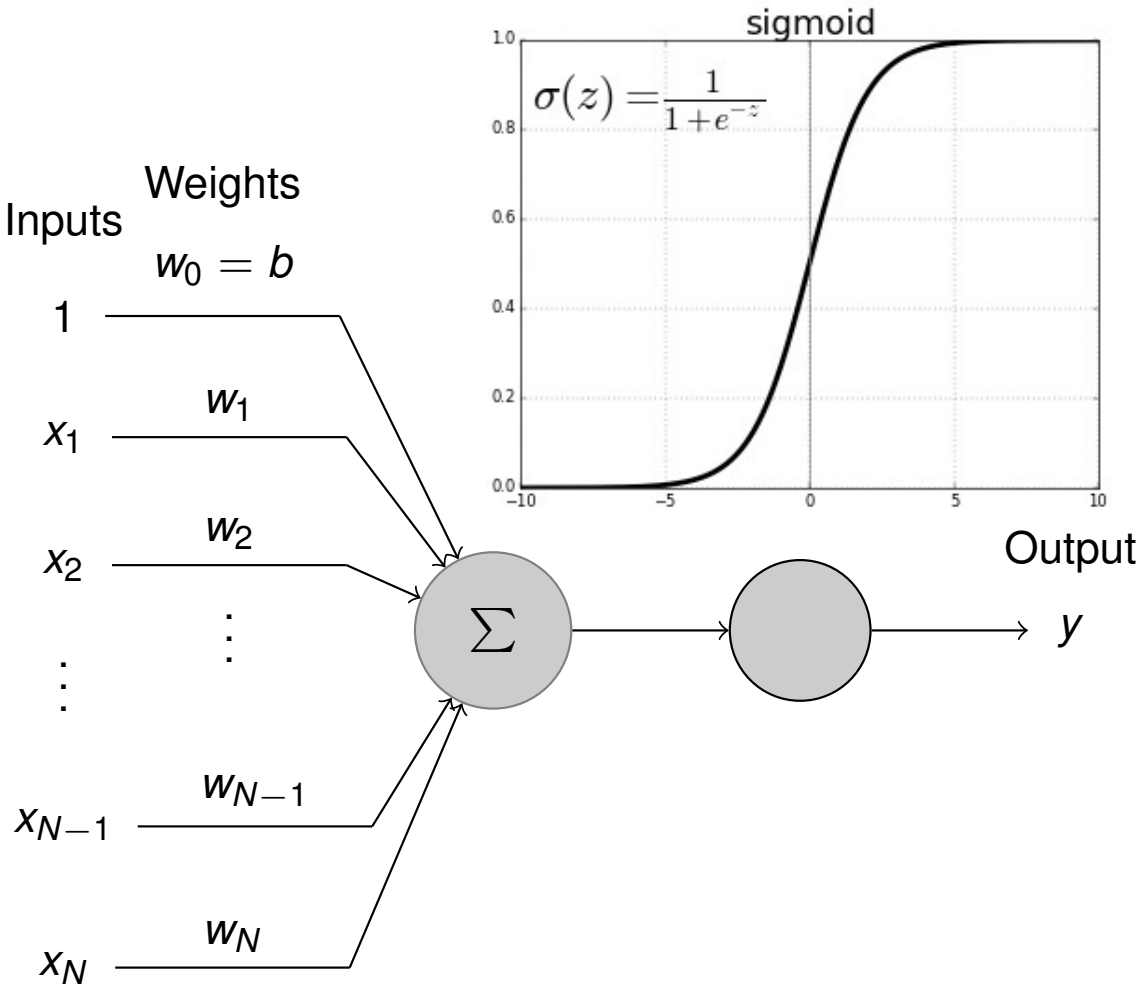
- It actually computes the probability that the input belongs to class 1



Logistic regression

It is the perceptron with a sigmoid activation

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



$$z = \sum_i w_i x_i$$

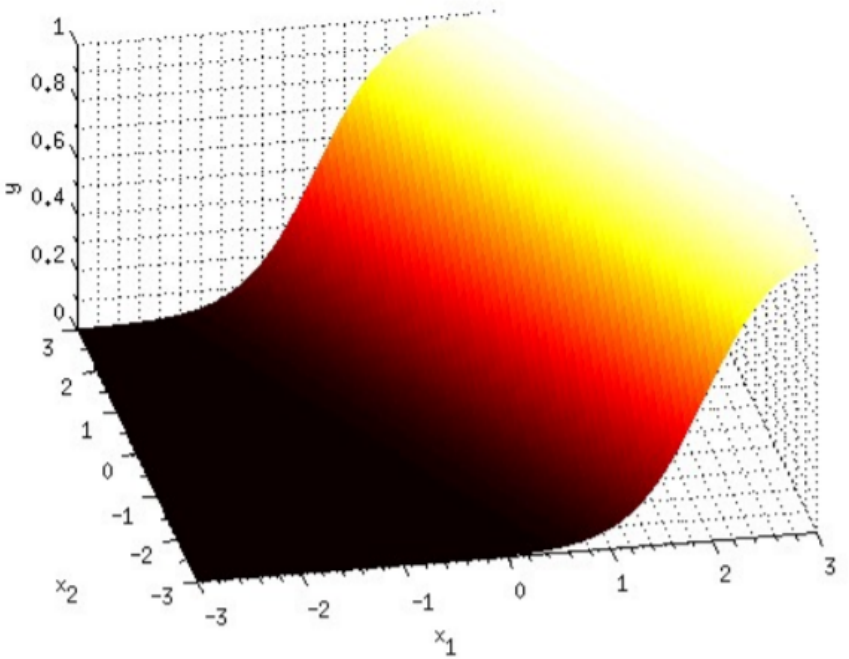
$$p(y = 1|x) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

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

Logistic regression

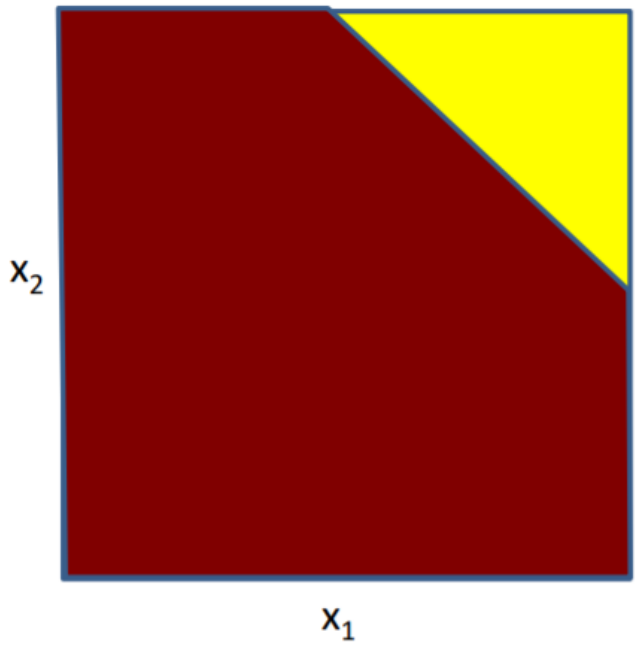
This the perceptron with a sigmoid activation

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



When X is a 2-D variable

Decision: y > 0.5?



$$P(Y = 1|X) = \frac{1}{1 + \exp\left(-\sum_i w_i x_i - b\right)}$$

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

- Loss function quantifies our unhappiness with the scores across the training data.

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 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:

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

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

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

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

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

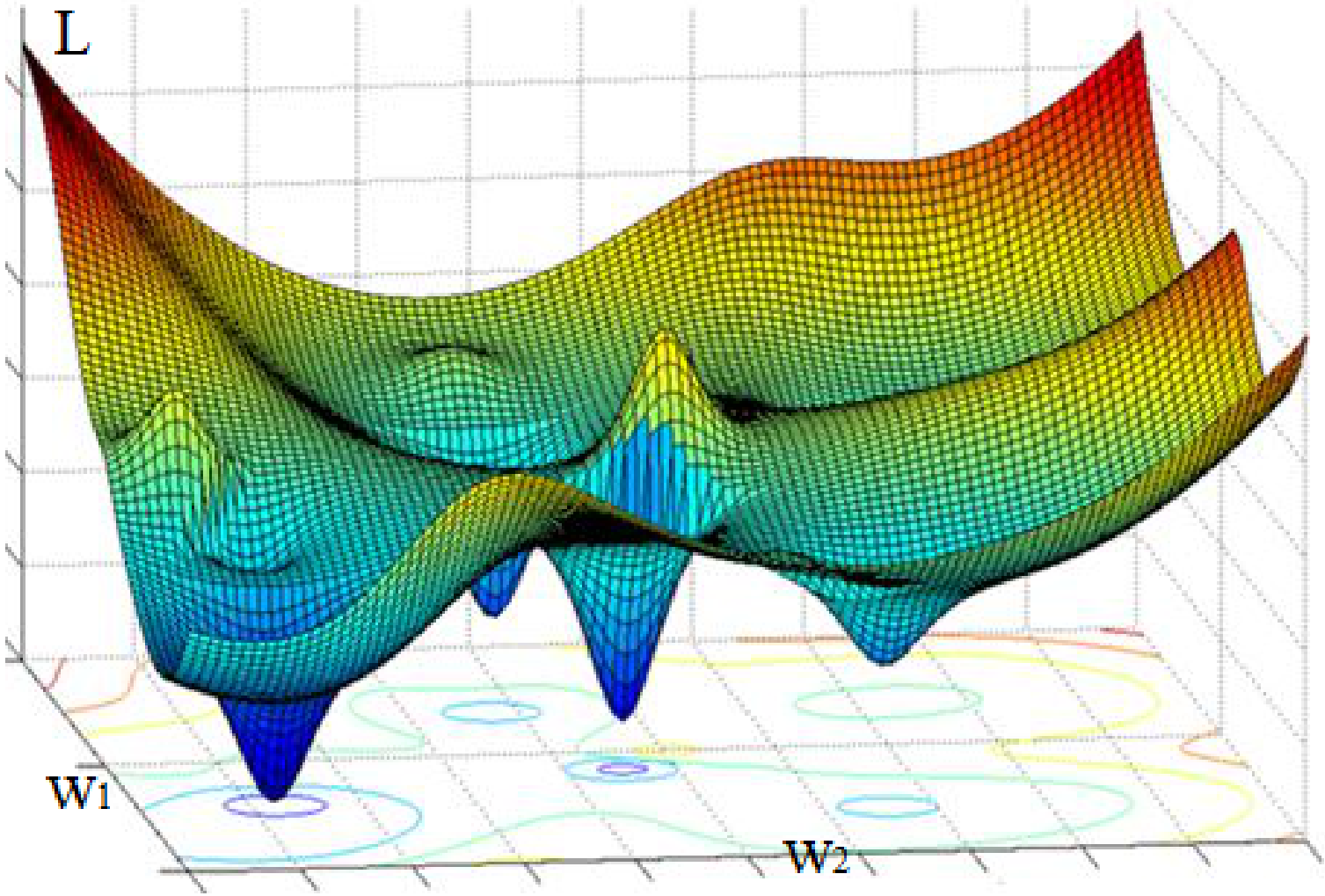
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

How do we find the best w?

Optimization

Optimization

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



Finding the minimum of a function

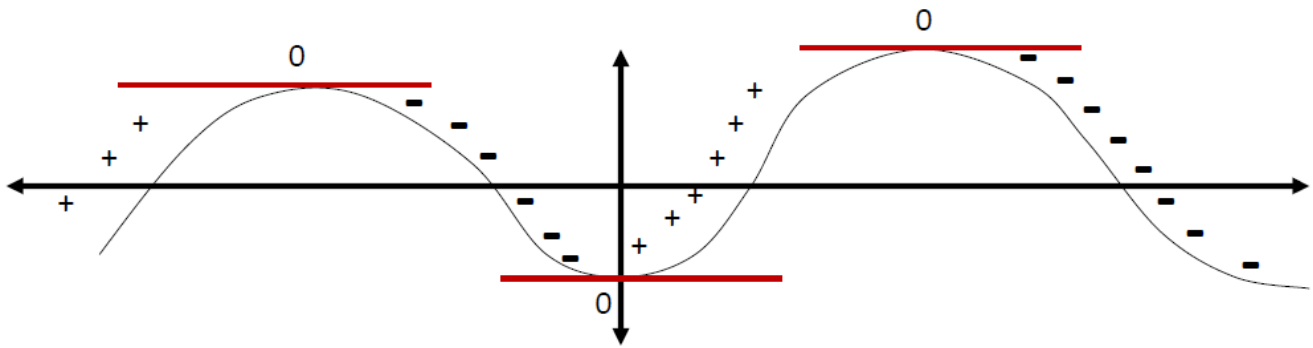
- Solve:

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

- 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

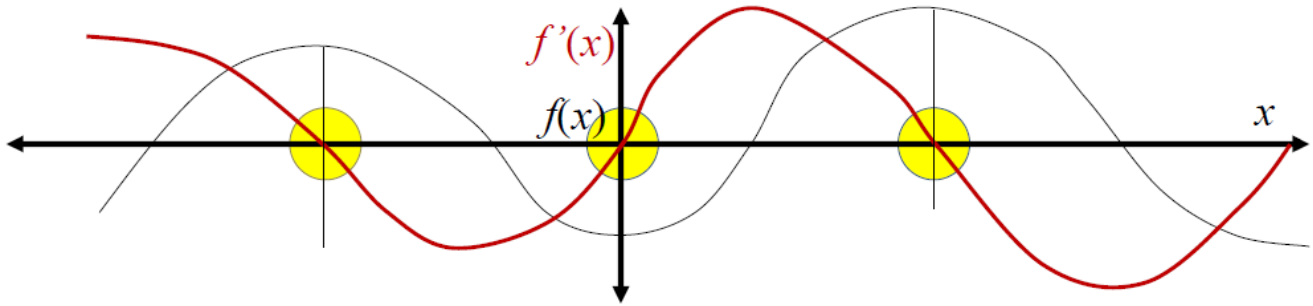
- Solve:

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

- 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

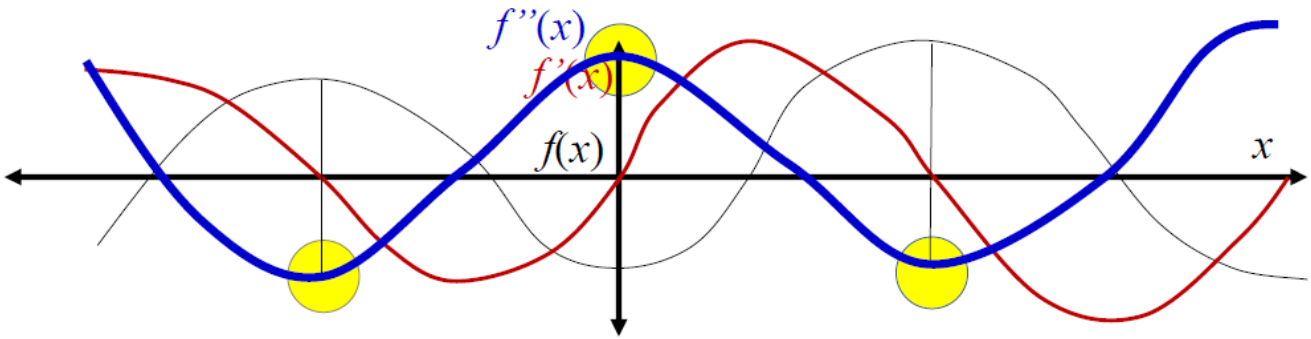
- Solve:

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

- 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

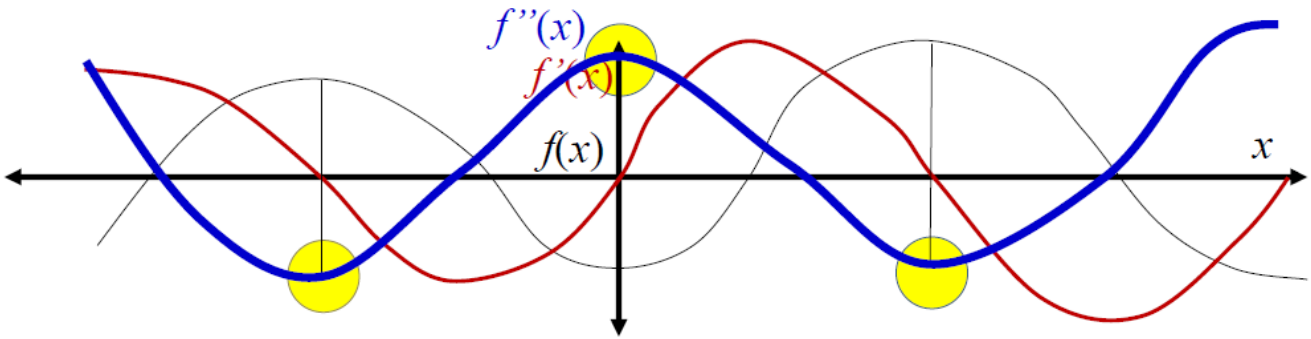
- Solve:

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

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

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

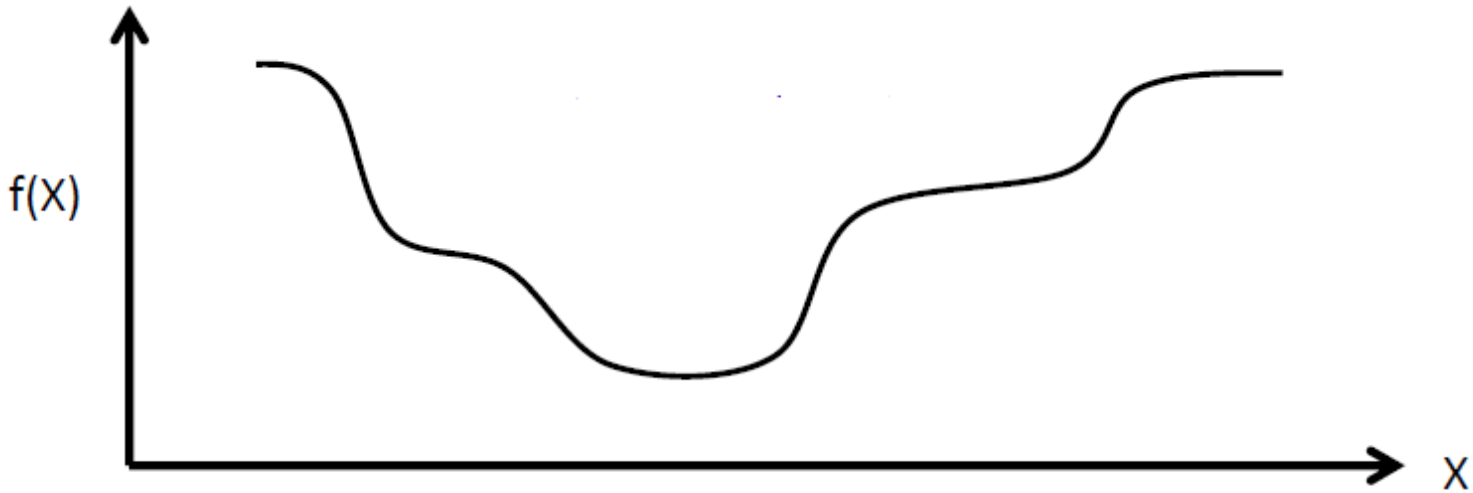
  - If it is positive x^* is a minimum, otherwise it is a maximum



It does not work!!!

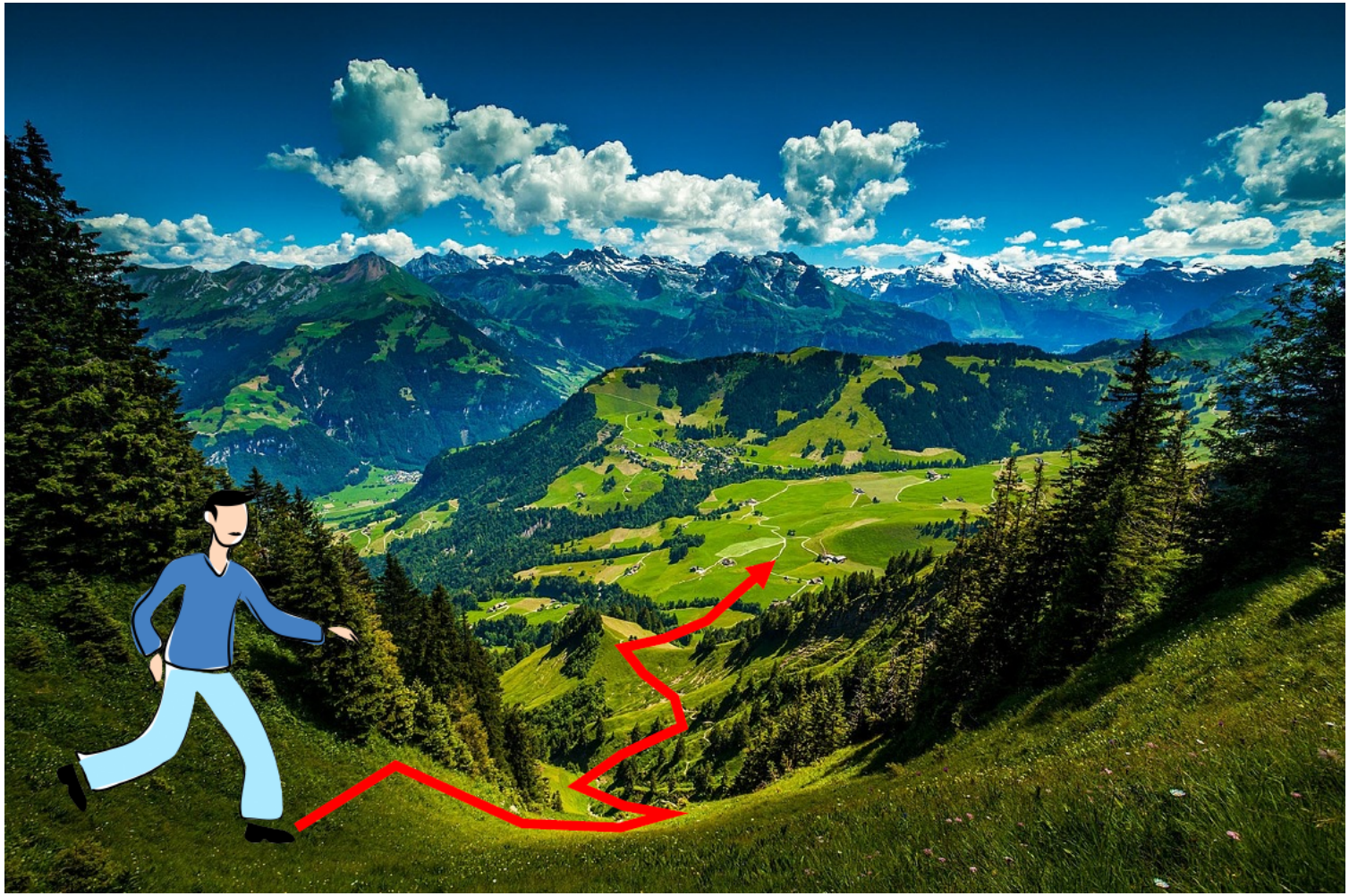
Closed form solutions are not always available

- 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:

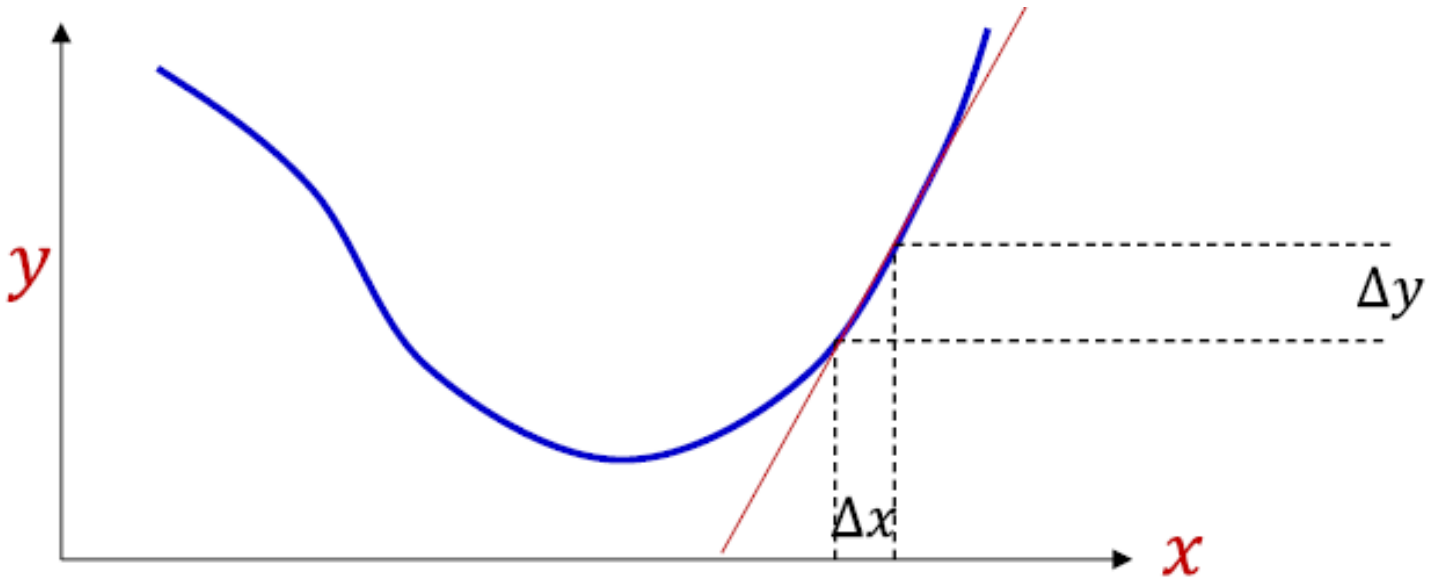
$$if \ \frac{df(x)}{dx} < 0 \Rightarrow f(x+h) < f(x) \Rightarrow f \text{ is decreasing in } x \Rightarrow \text{take a step in } \textbf{positive} \text{ direction}$$

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

  - 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
  $y = f(\boldsymbol{x})$ ($f : \mathbb{R}^d \to \mathbb{R}$)

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

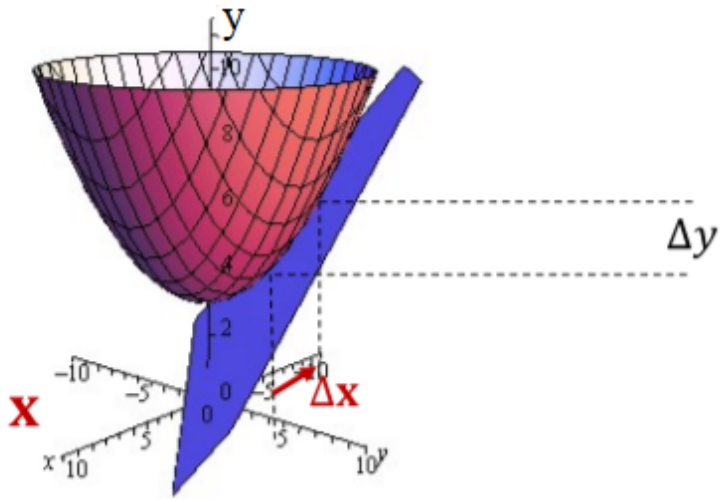
Multivariate scalar function

- 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

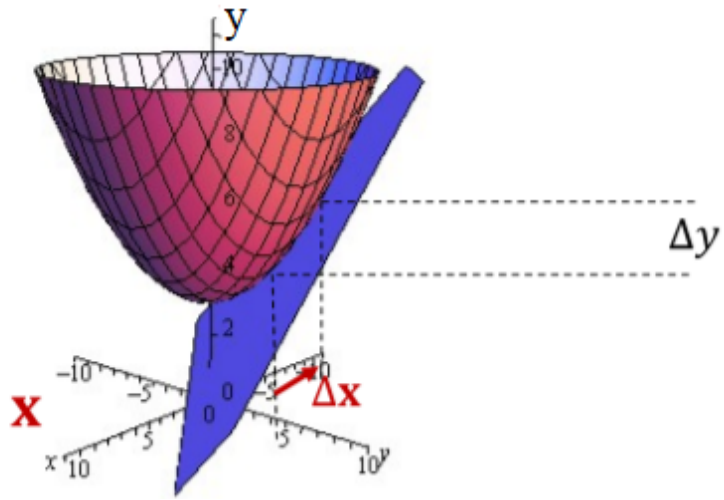
$$\Delta \boldsymbol{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_d \end{bmatrix}$$



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

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

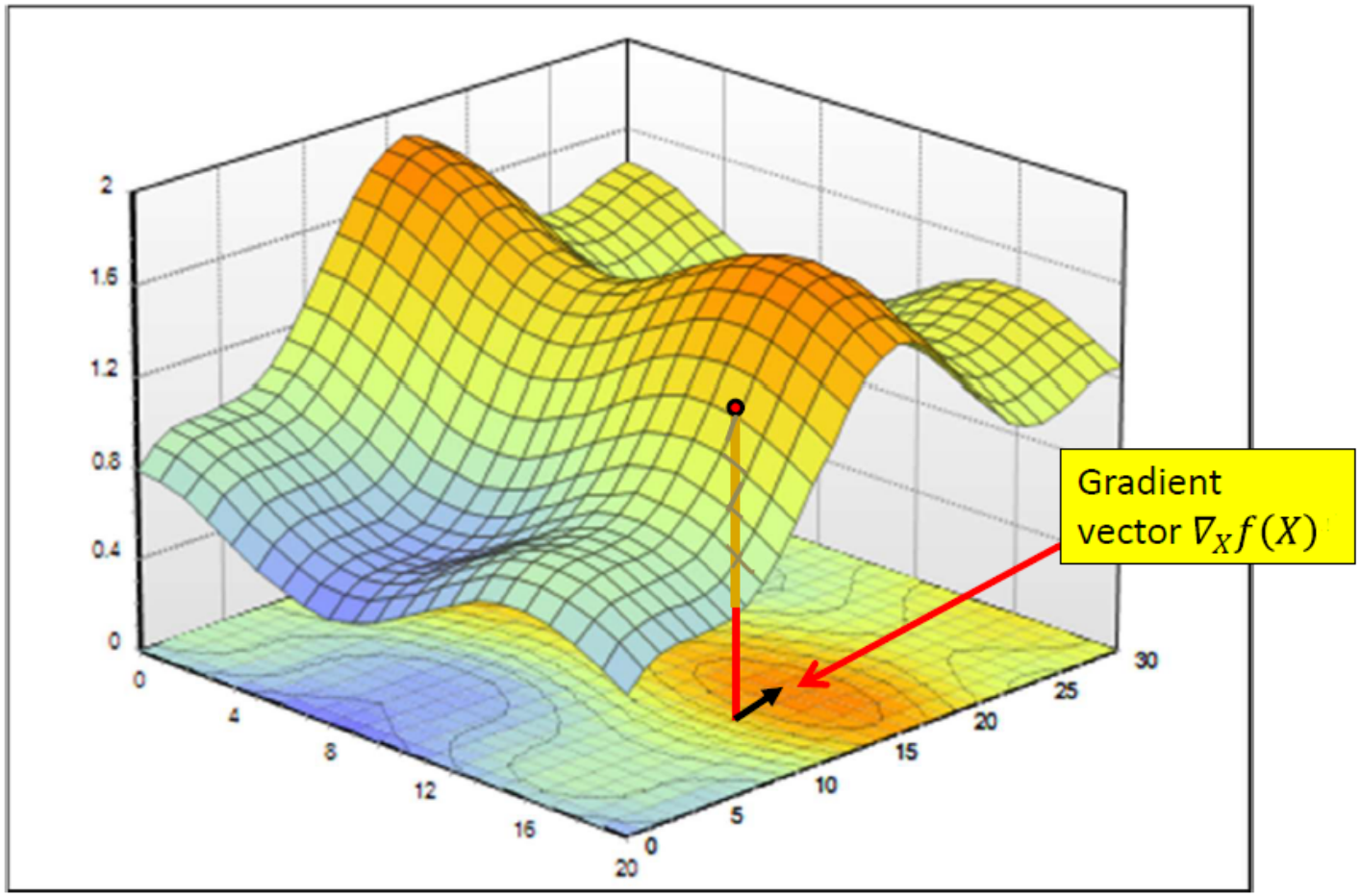
  - Giving us that α is a row vector: $\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_d \end{bmatrix}$

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

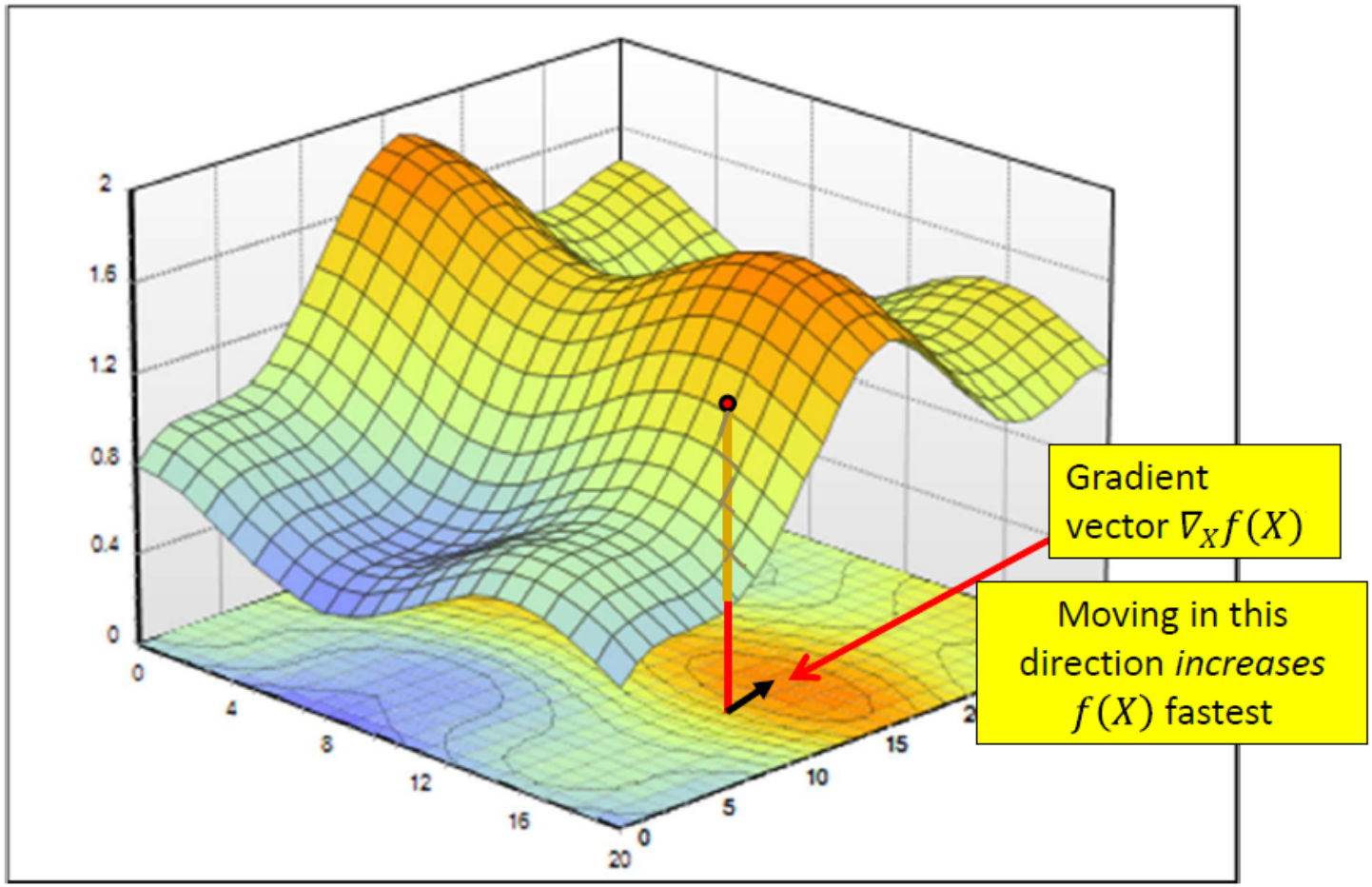
  - The partial derivative α_i gives us how y increments when only x_i is incremented.
  - Often represented as $\frac{\partial y}{\partial x_i}$

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

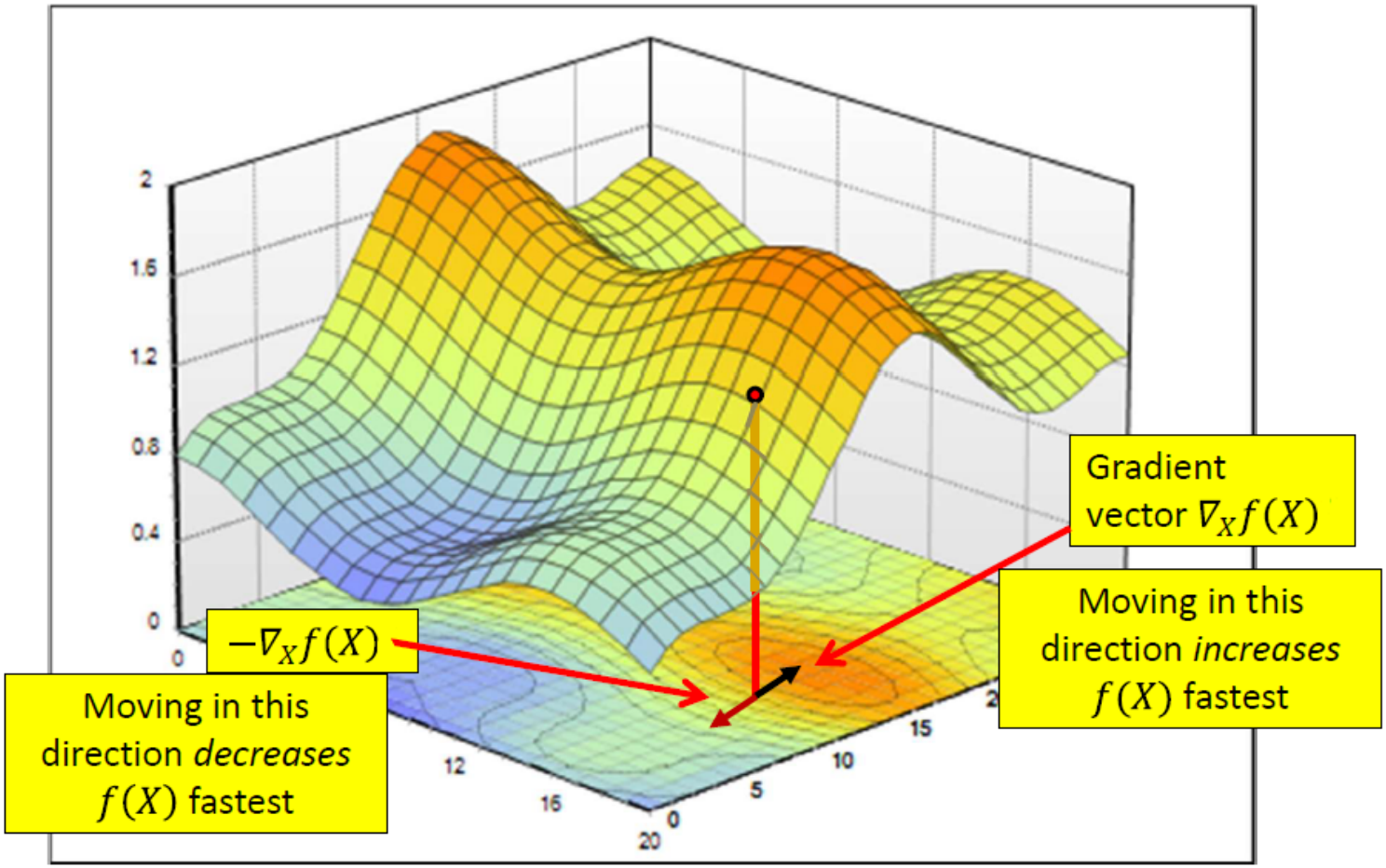
Gradient


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 $\nabla_X f(X)$

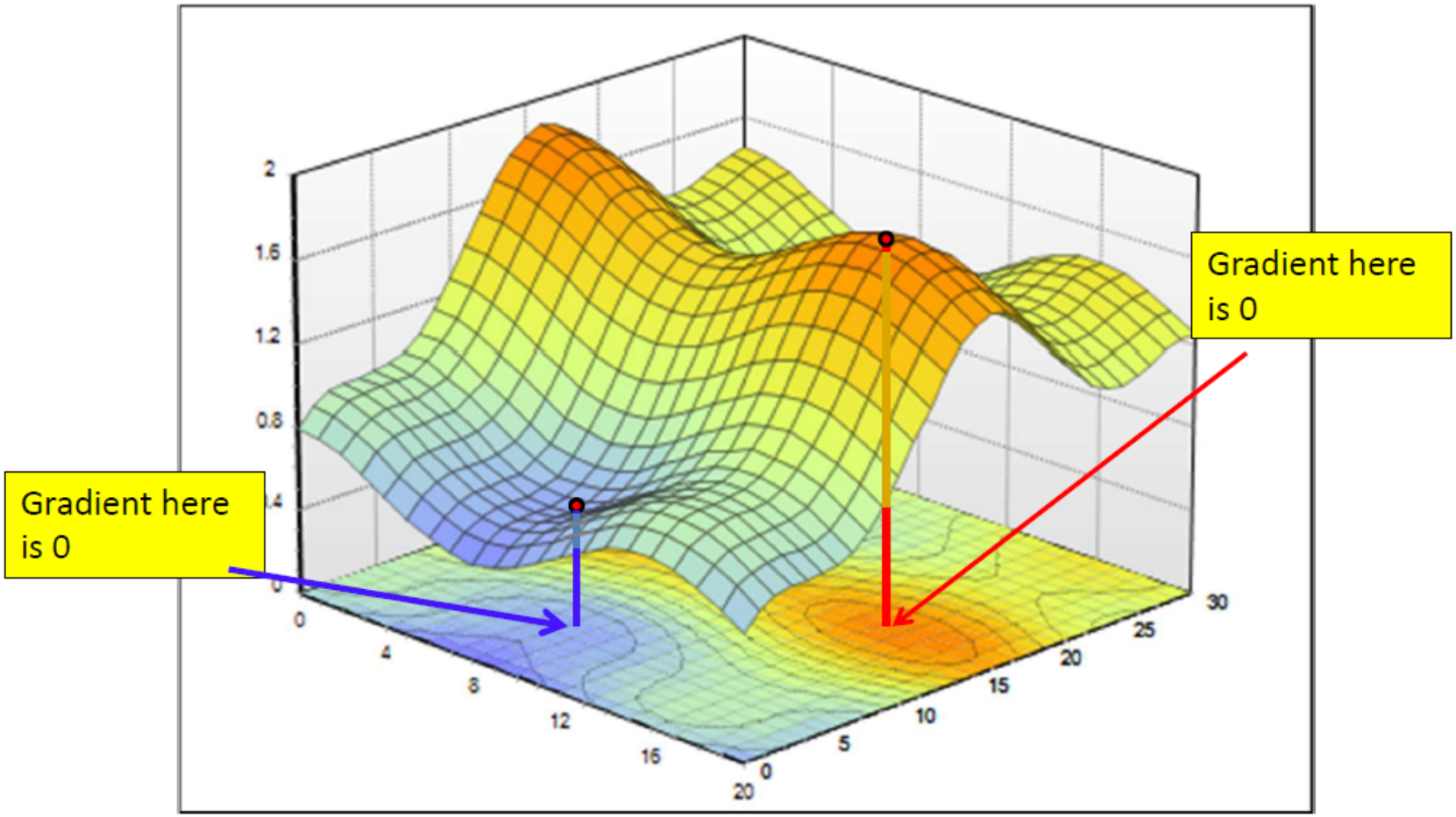
Gradient


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

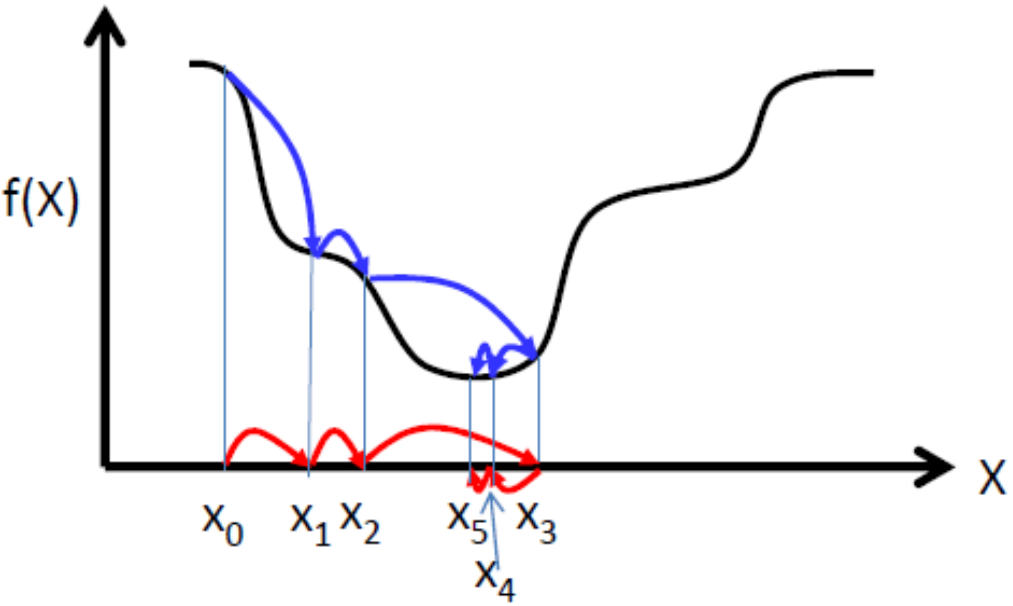

Gradient vector $\nabla_X f(X)$
Moving in this direction *increases* $f(X)$ fastest
$-\nabla_X f(X)$
Moving in this direction *decreases* $f(X)$ fastest
2
1.6
1.2
0.8
0.4
0
0
12
16
20
0
5
10
15

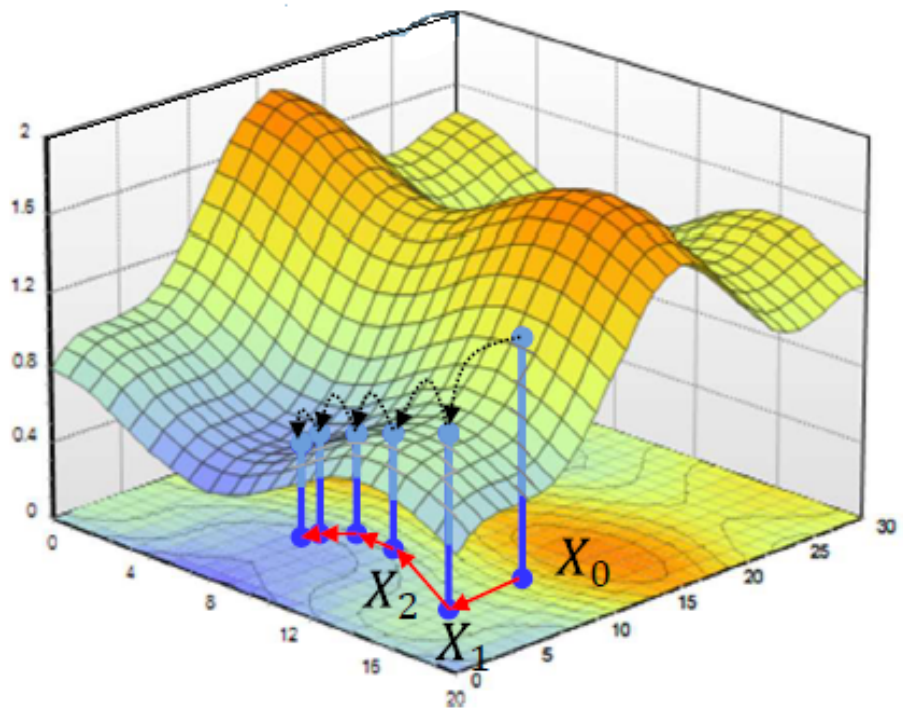
Gradient


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

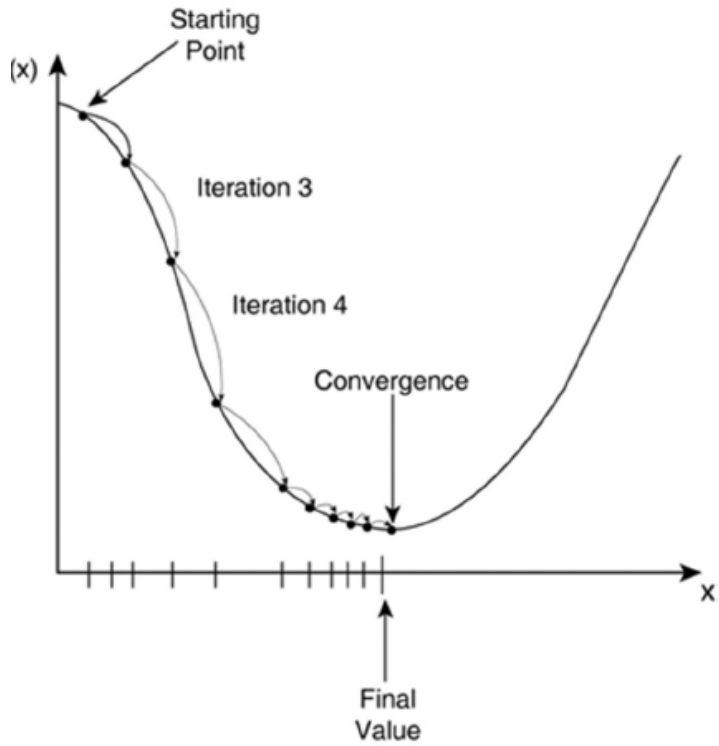
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$$

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

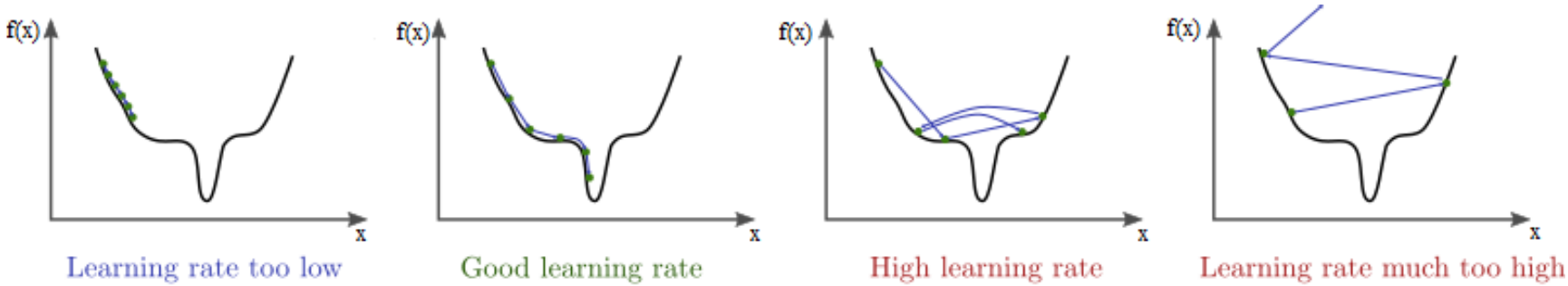
- The gradient descent algorithm converges when the following criterion is satisfied.

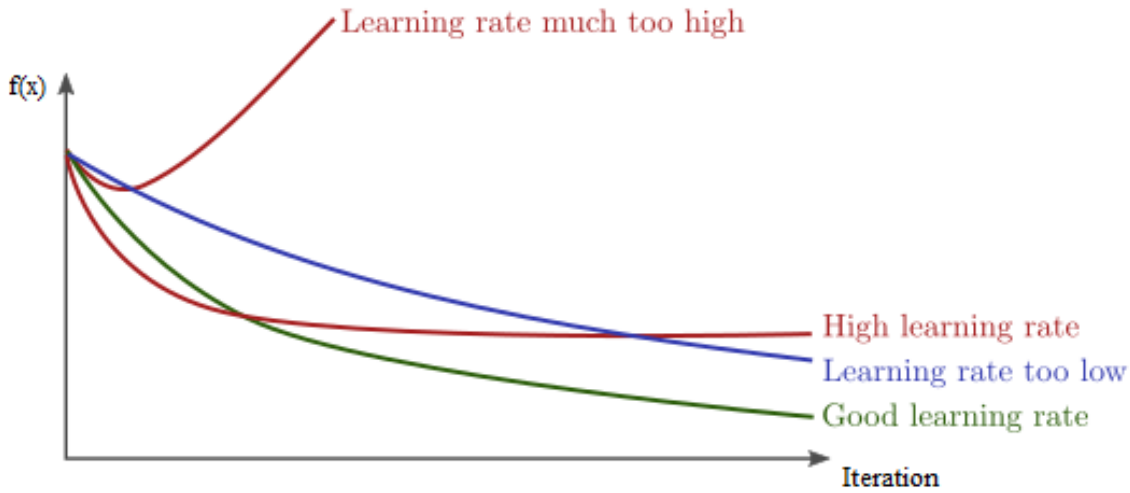
$$|f(\boldsymbol{x}^{t+k}) - f(\boldsymbol{x}^t)| < \epsilon$$

  - k is a hyperparameter.

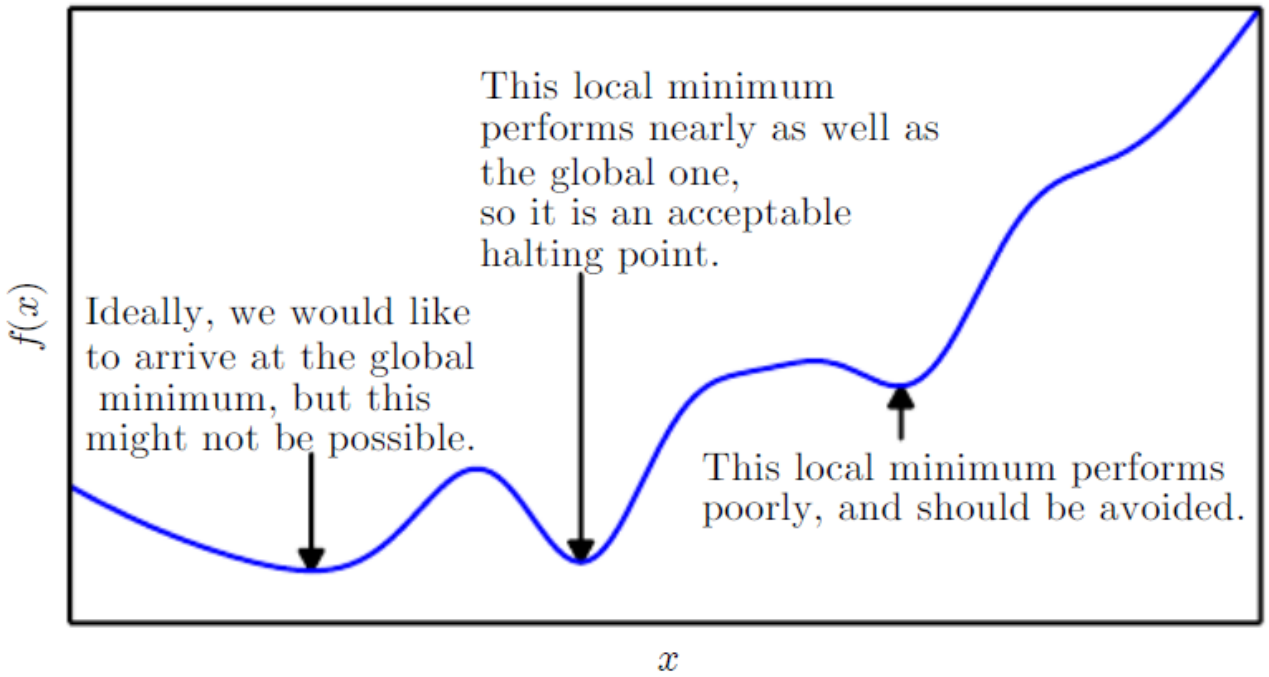


Influence of step size





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

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

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

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

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

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 $\boldsymbol{x}^{(i)}$:

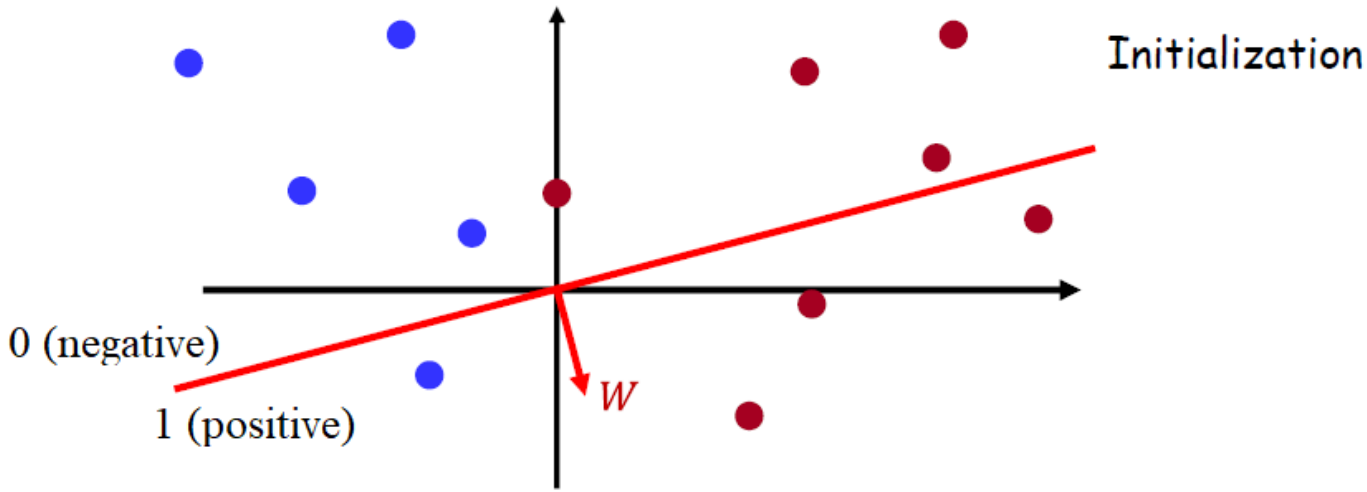
$$\text{If } y^{(i)} = 0: \quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \boldsymbol{x}^{(i)}$$

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

Perceptron learning algorithm

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

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

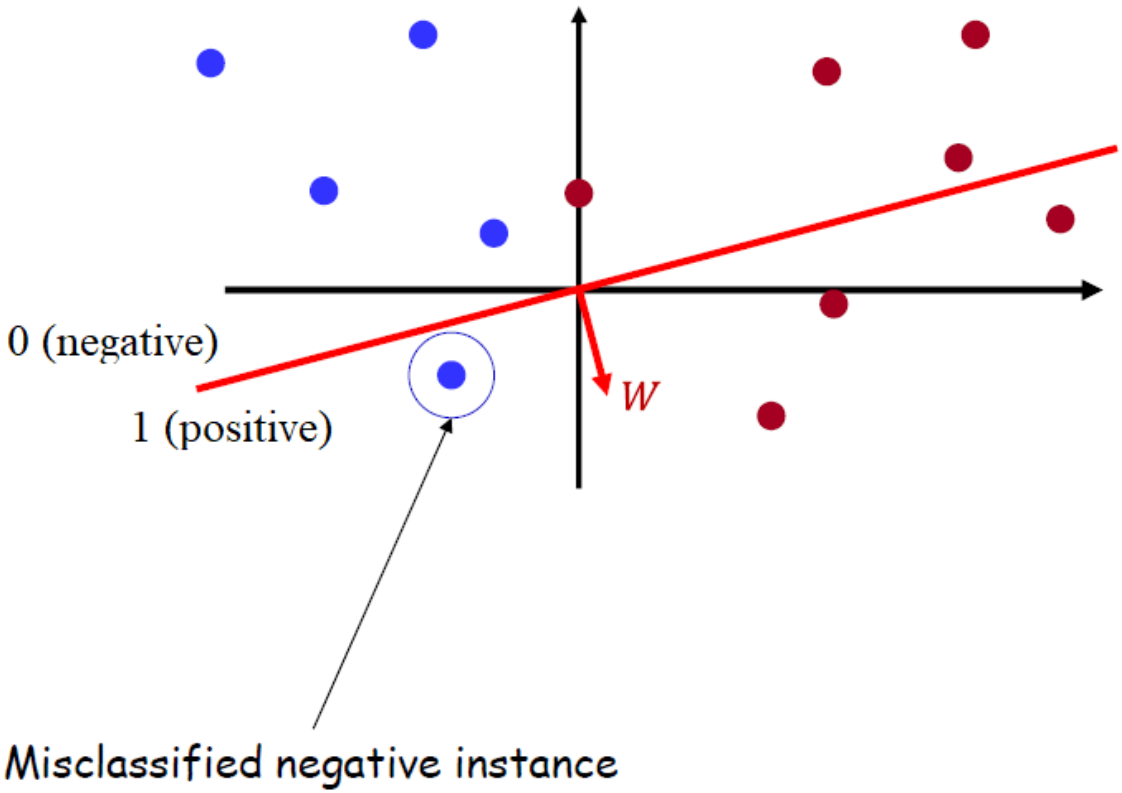


Perceptron learning algorithm

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

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

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

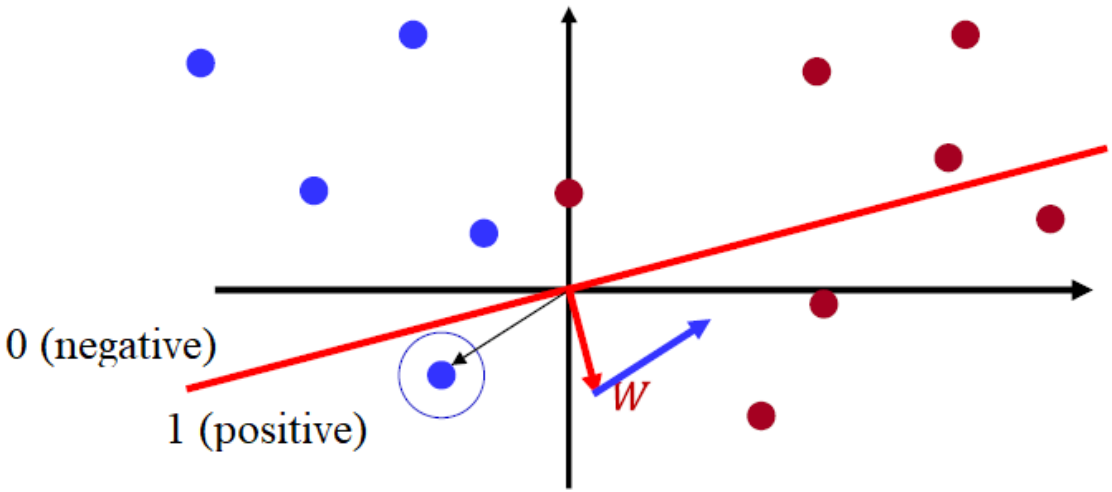


Perceptron learning algorithm

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

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

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



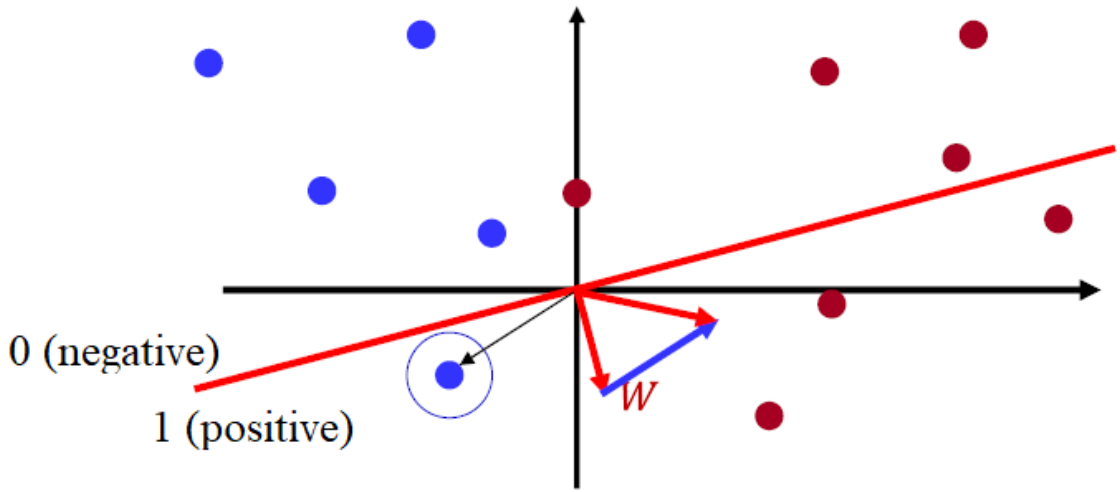
Misclassified *negative* instance, *subtract* it from W

Perceptron learning algorithm

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

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

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

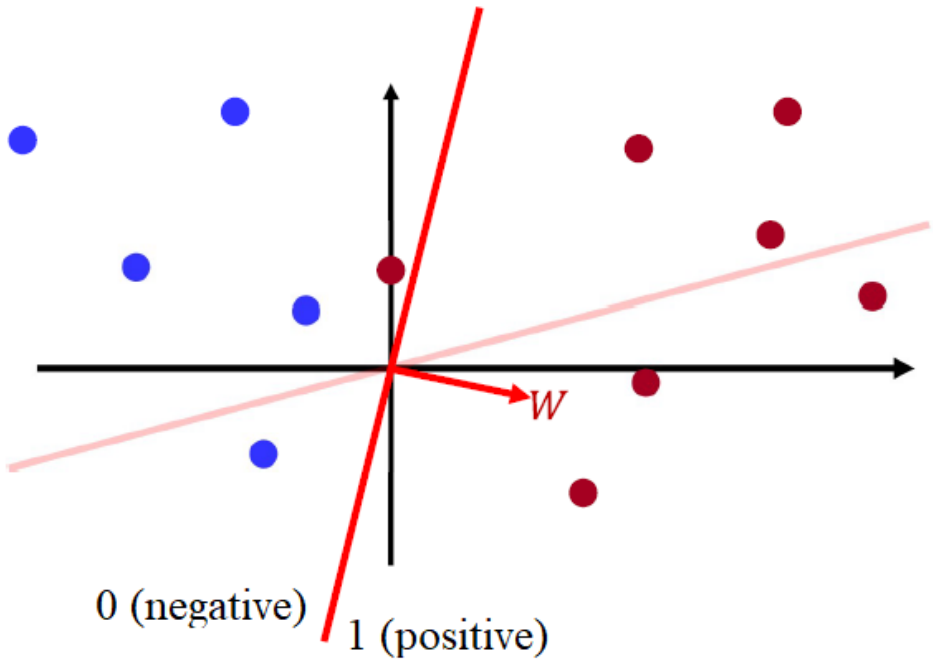


The new weight

Perceptron learning algorithm

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

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



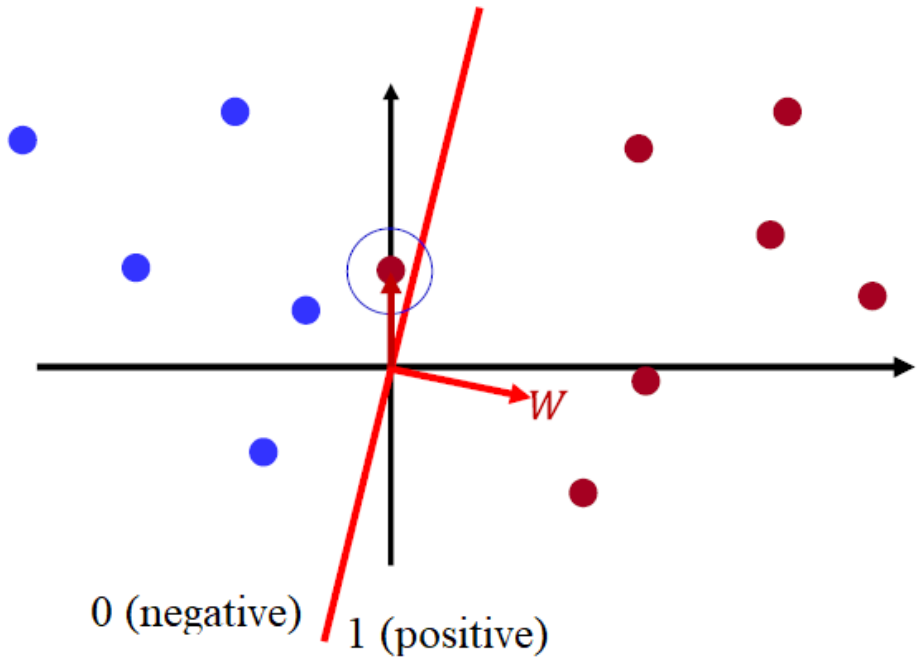
The new weight (and boundary)

Perceptron learning algorithm

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

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

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



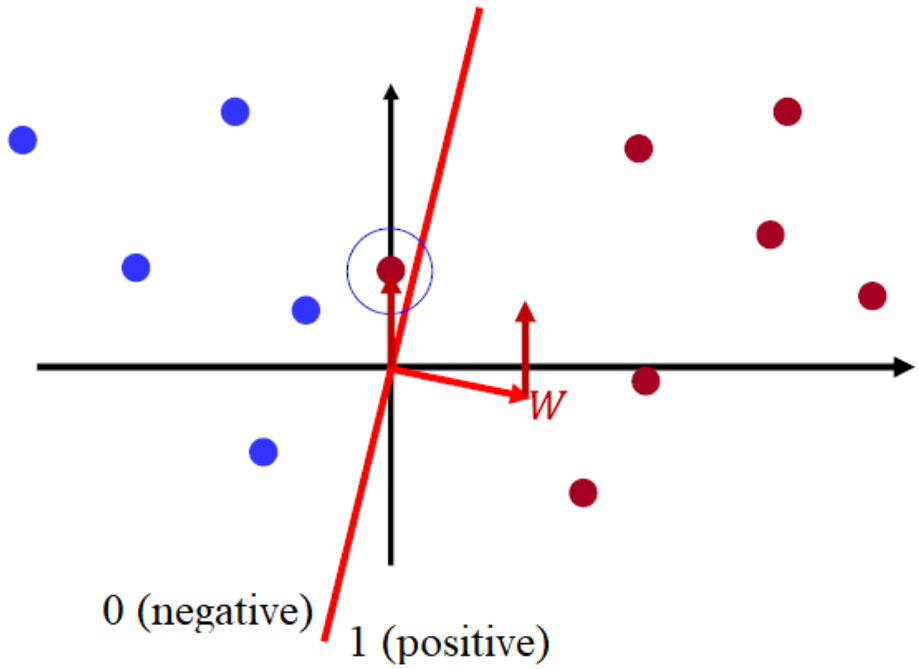
Misclassified *positive* instance

Perceptron learning algorithm

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

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

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



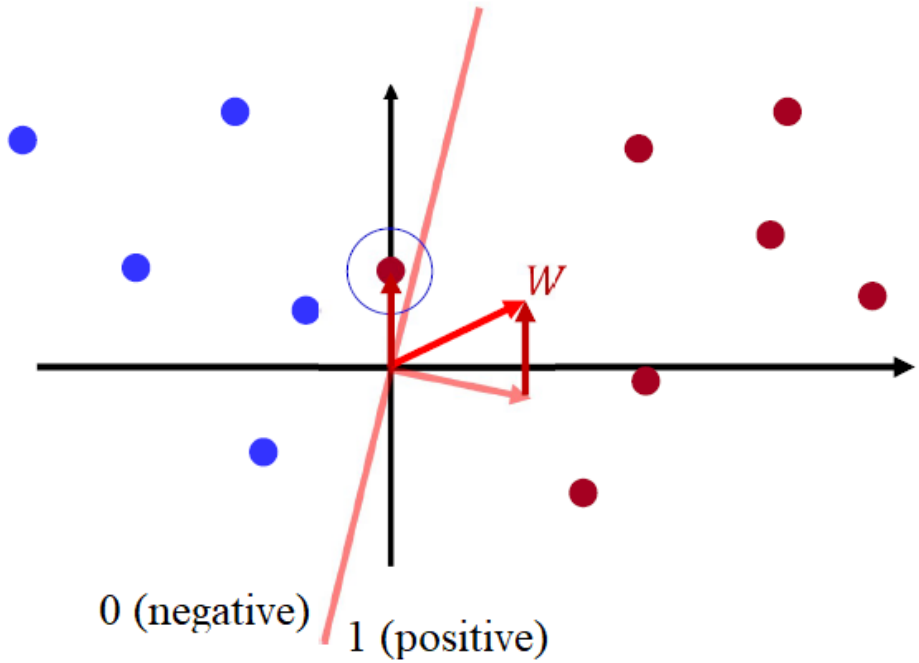
Misclassified *positive* instance, *add* it to W

Perceptron learning algorithm

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

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

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



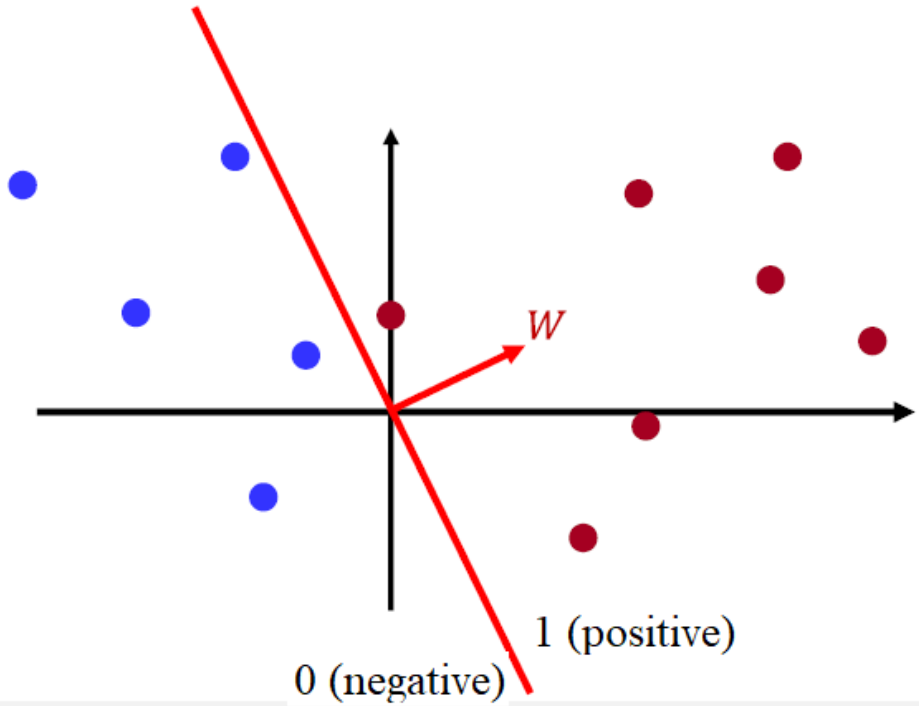
The new weight vector

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

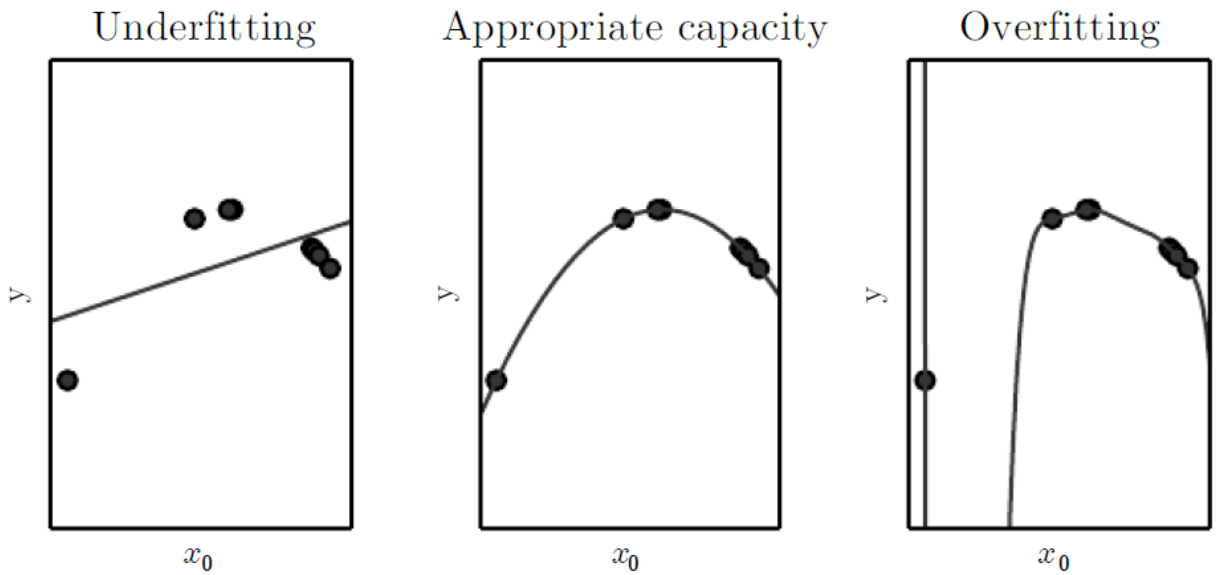
- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. 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

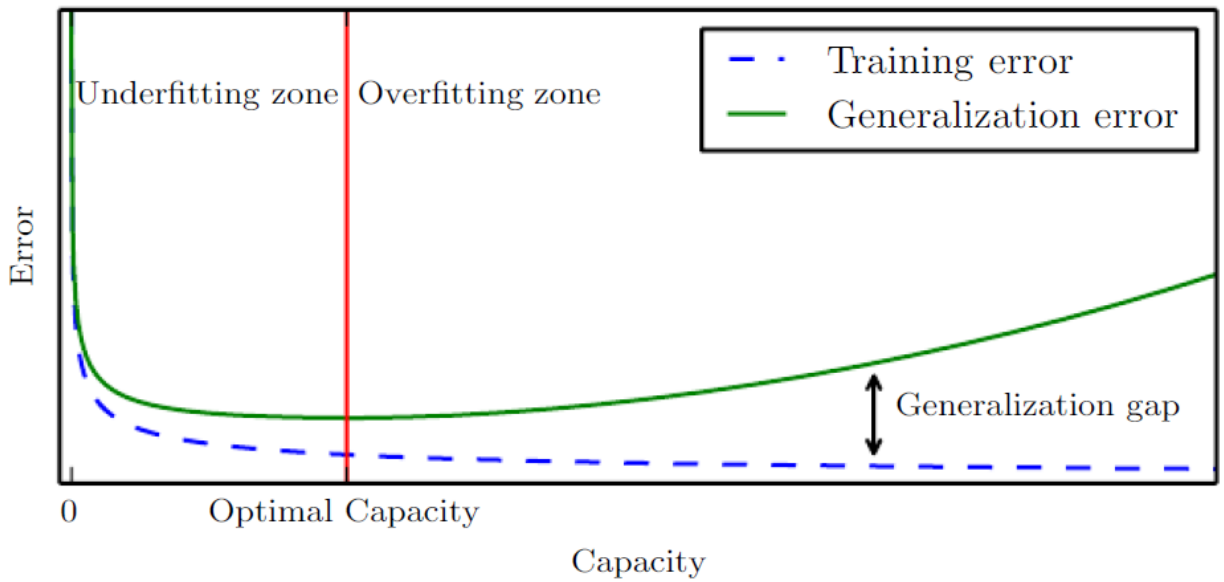
- Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with.



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

Capacity

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



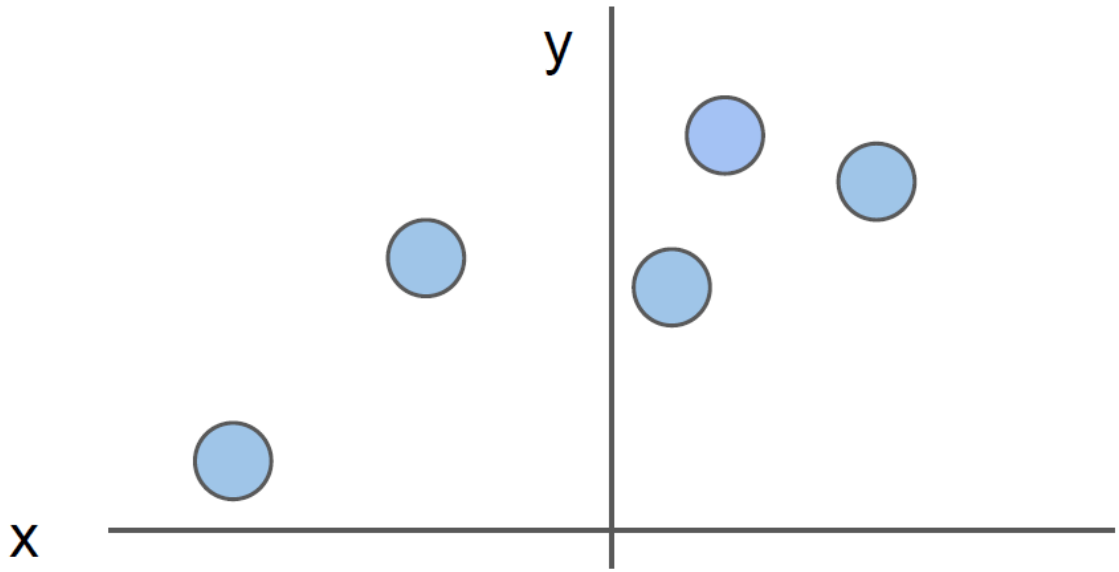
Regularization

- **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?
  - Occams Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
  - Avoid overfitting: prefer simple models that generalize better

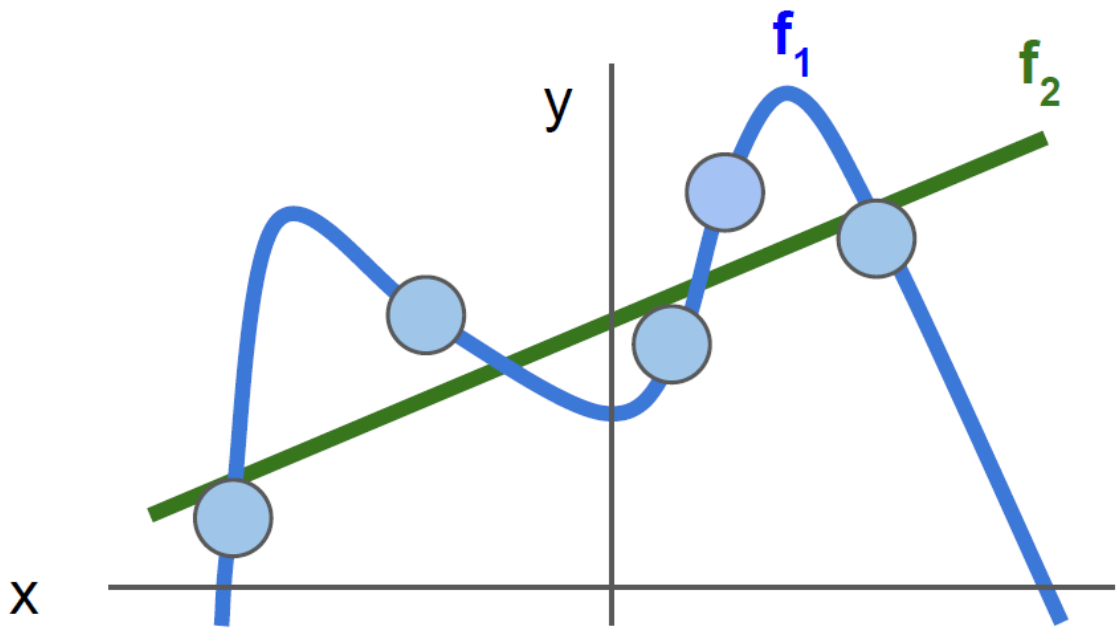
- Regularization intuition: toy example training data



Regularization

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

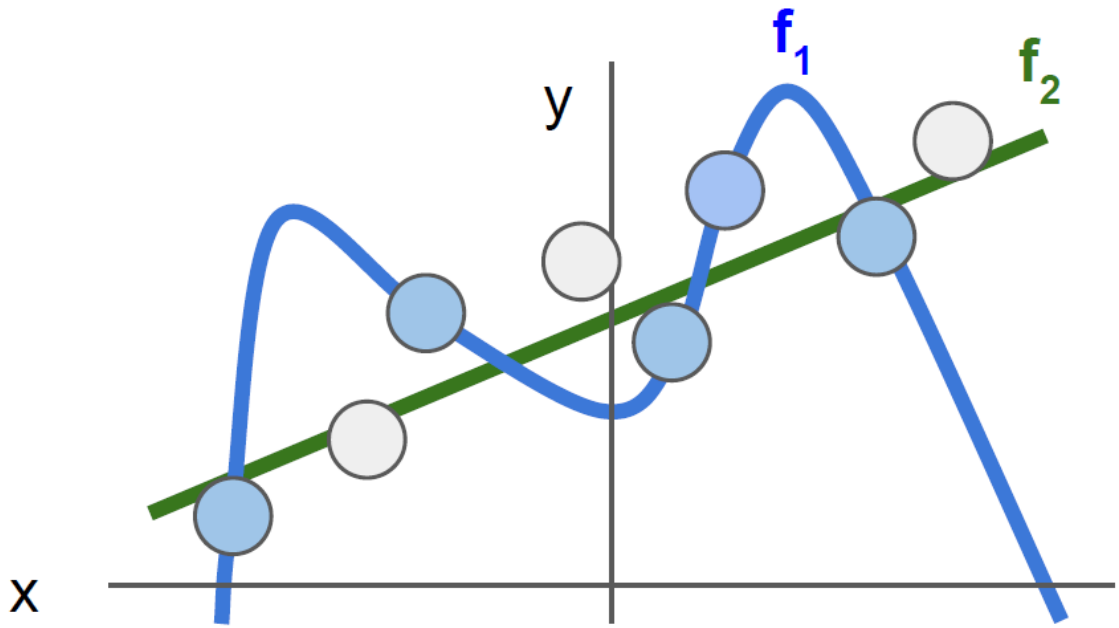
- Regularization intuition: Prefer Simpler Models



Regularization

- How do we choose between hypotheses?
  - Occams 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



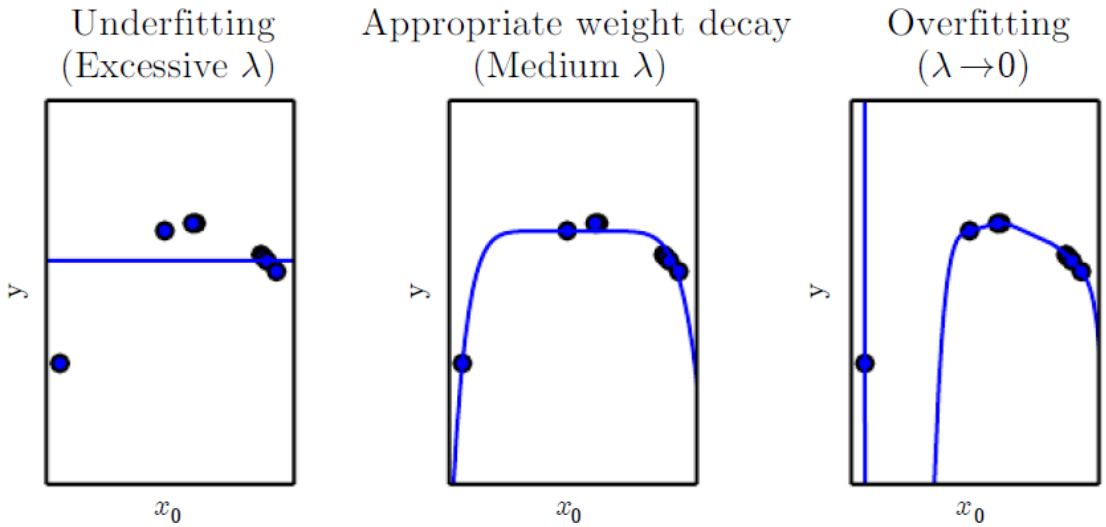
- Regularization pushes against fitting the data too well so we dont fit noise in the data

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