Artificial Intelligence

Lecture 10：Classification

Xiaojin Gong

2022-05-09

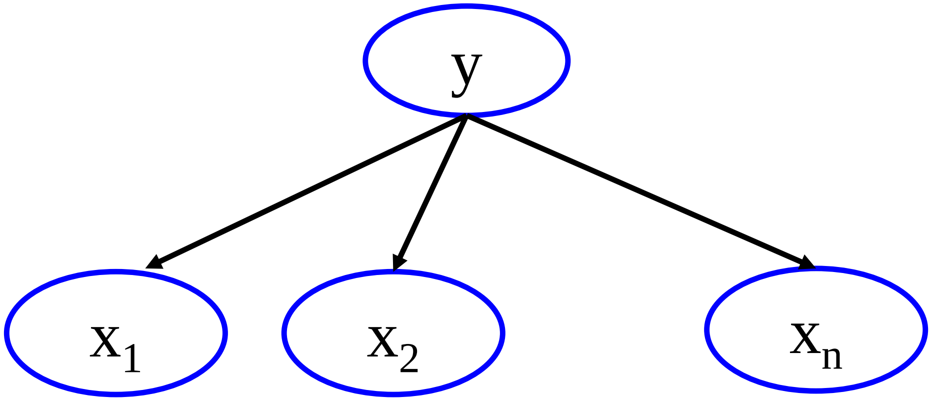
Credits: AI course in Berkeley & MIT

Review

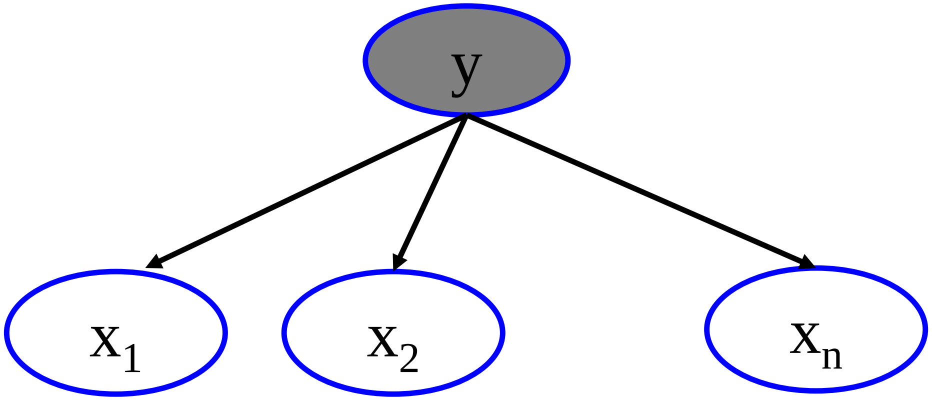
- Supervised learning:
  - Observes example input-output pairs
  - Learns a function that maps from input to output
- Unsupervised learning:
  - Learns patterns in the input even though no explicit feedback is given
- Reinforcement learning:
  - Learns from a series of reinforcements – rewards or punishments

Review

- Supervised classification: naïve Bayes model



- Unsupervised clustering: learning mixtures of Gaussians



Review

- Training & Testing:
  - Training set
  - Validation set
  - Test set
- Generalization & Overfitting:
  - Want a classifier which does well on test data
  - Fitting the training data very closely, but not generalizing well

Outline

- Classification
  - Decision Trees
  - Perceptron
  - Artificial Neural Networks

Supervised Learning

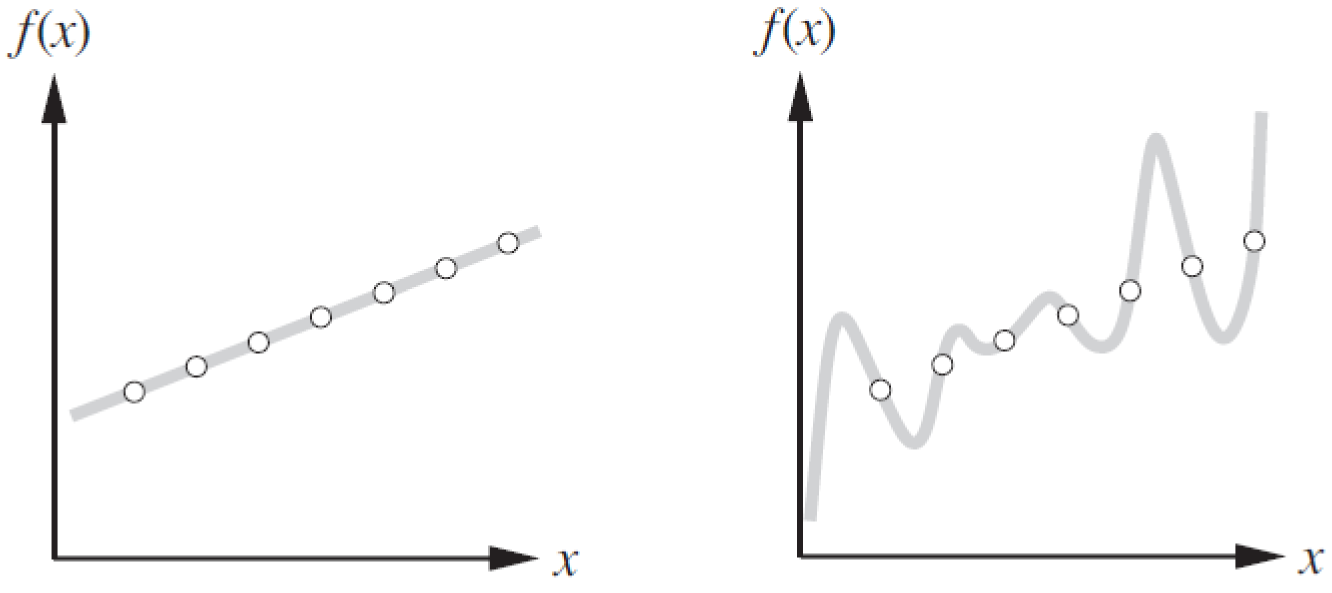
- Given a training set of N example input-output pairs

$$(x_1, y_1),\ (x_2, y_2), \cdots, (x_N, y_N)$$

  - where each y_j was generated by an unknown function $y = f(x)$,
  - discover a function h that approximates the true function f.
- The function h is a hypothesis
- When y is discrete $\Longrightarrow$ classification problem
- When y is continuous $\Longrightarrow$ regression problem

Hypothesis Space

- Ockham's razor:
  - Simpler hypotheses tend to generalize to future data better
  - Prefer the simplest hypothesis consistent with data



Generalization & Overfitting

Hypothesis Space

- Choosing the hypothesis h^* that is most probable given the dat

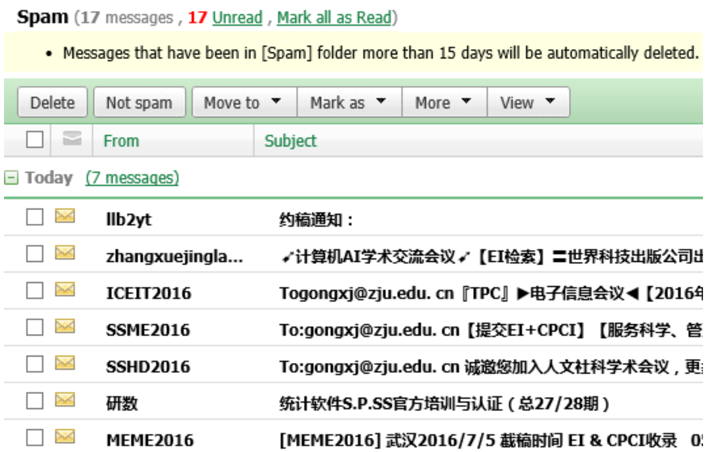
$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, P(h|data)$$

$$= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, P(data|h)\, P(h)$$

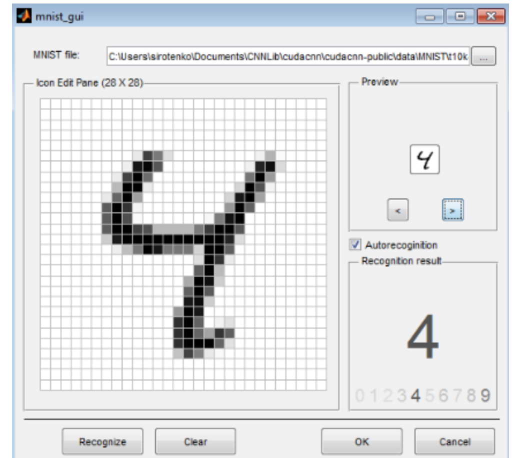
- A tradeoff between the expressiveness of a hypothesis space and the complexity of finding a good hypothesis within that space.

Classification

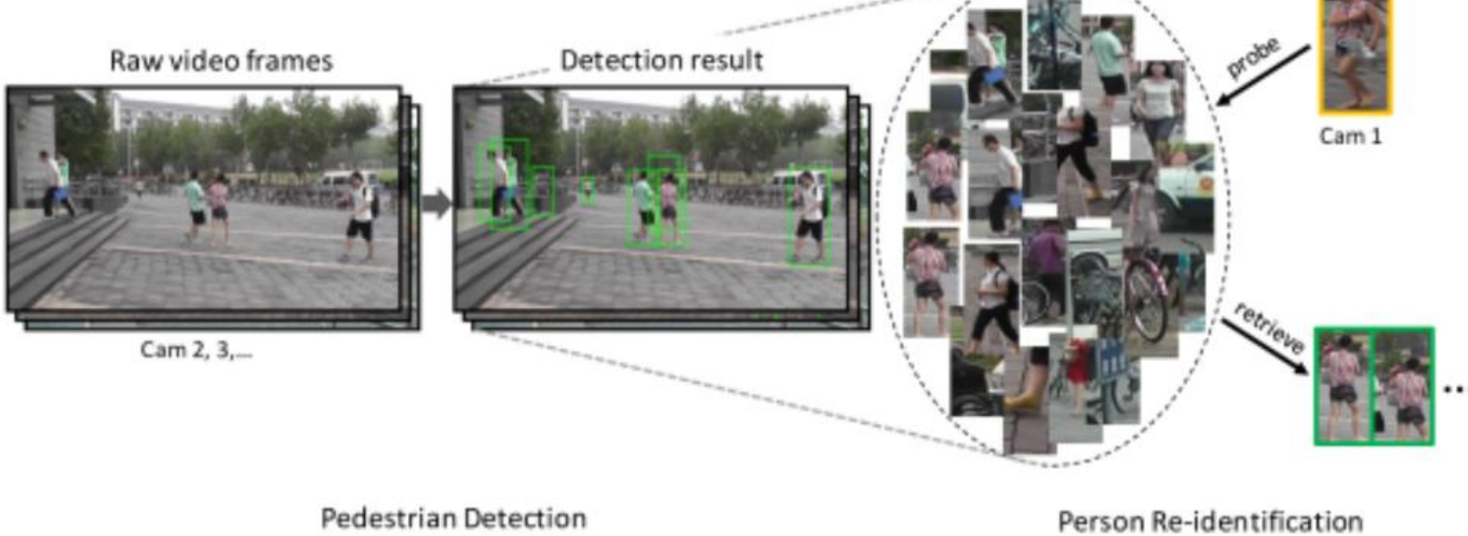
- Binary



- Multi-class

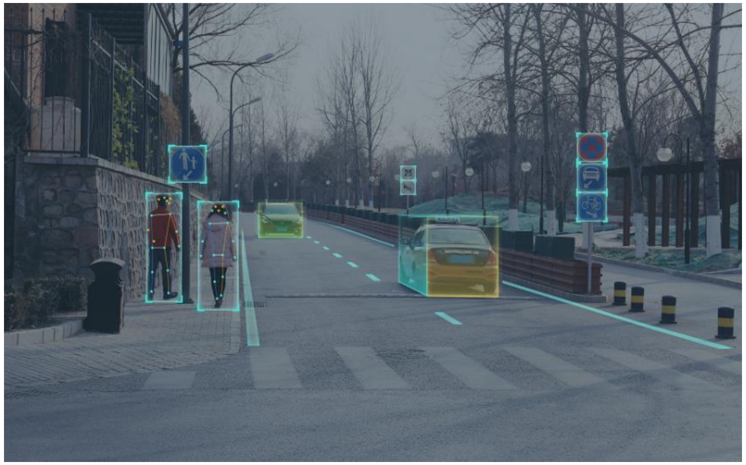


- Person Search / Re-identification



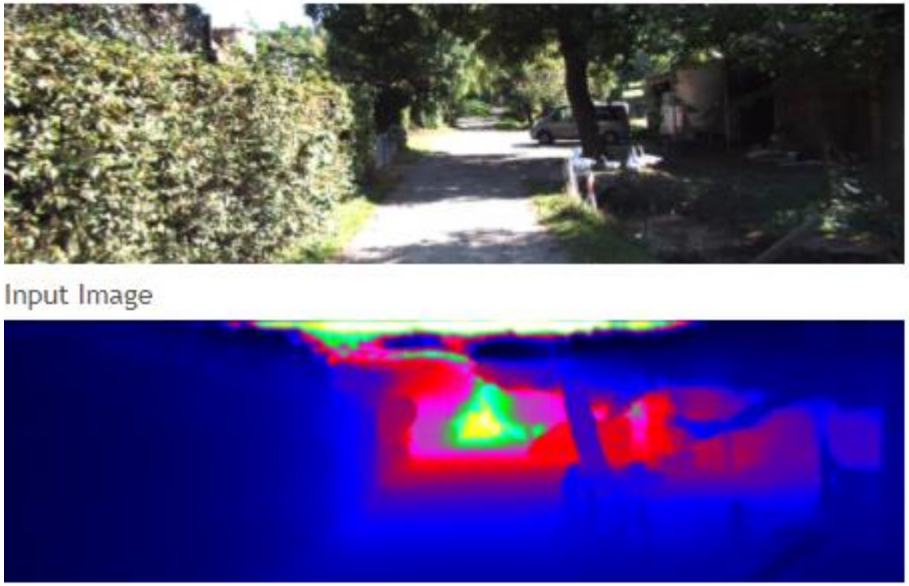
Classification

- Object Detection



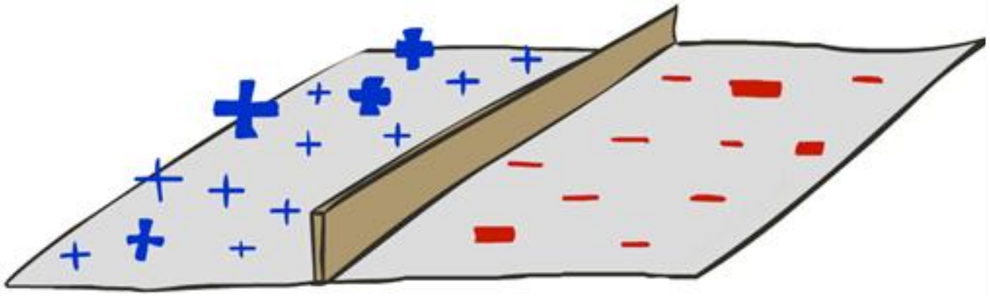
- Semantic Segmentation

- Depth Prediction

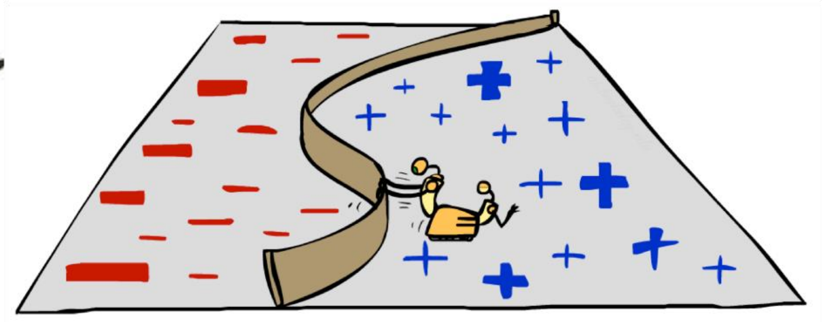


Classification

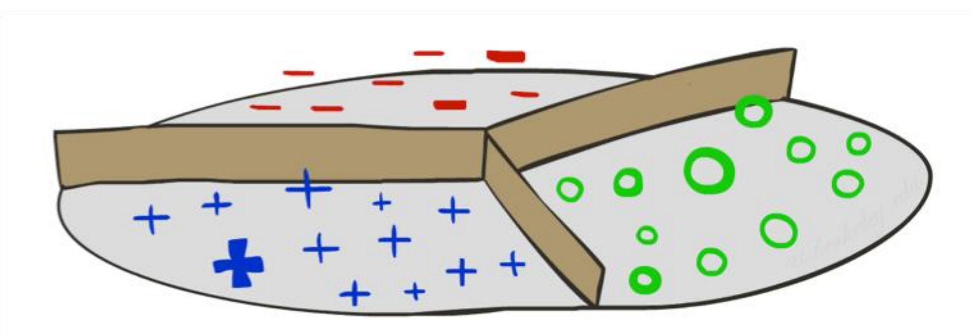
- Linear

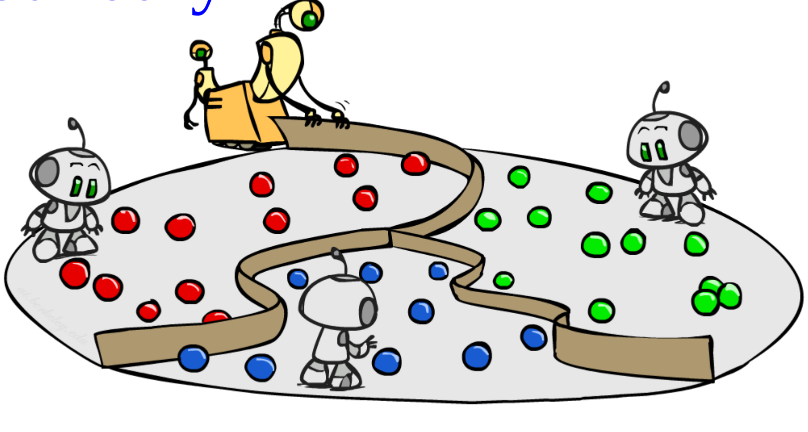


- Nonlinear



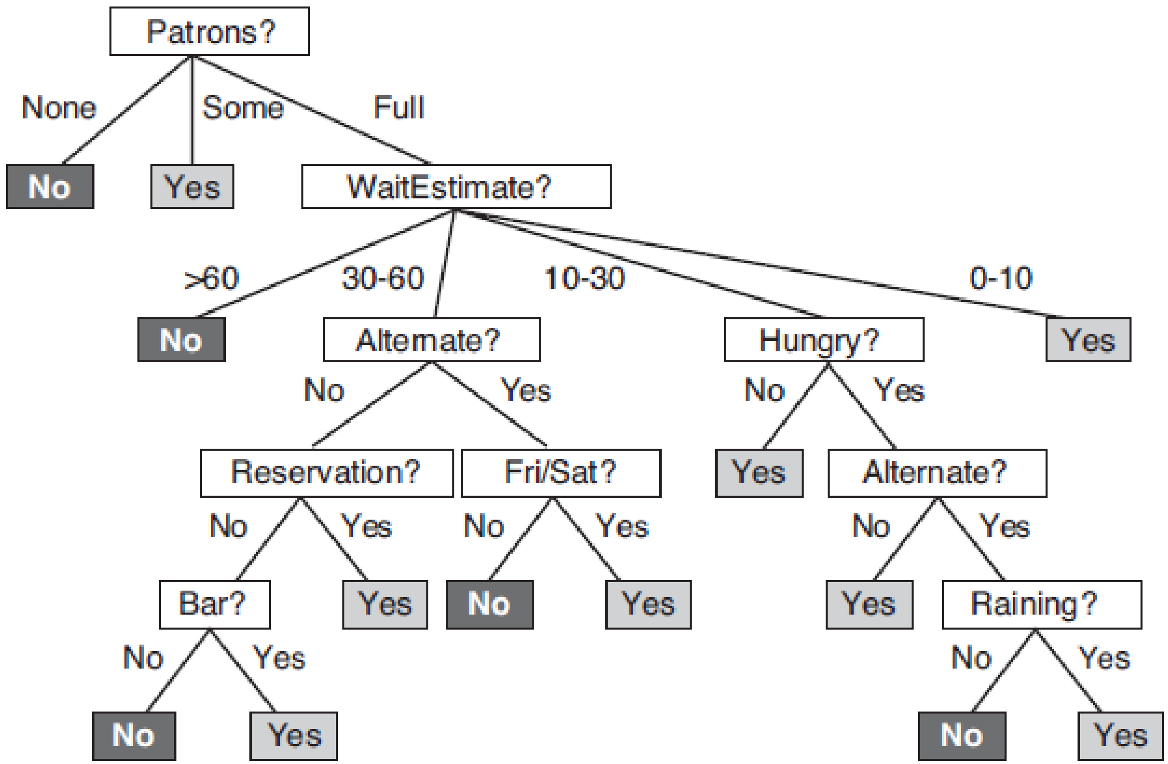
Decision boundary





Decision Trees

- Representation
  - Input: a vector of attribute values
  - Output: a single output value
  - Reaches it decision by performing a sequence of tests
  - Each node corresponds to a test of the value of one of the input attributes



Decision Trees

- Learning: how to learn a good decision tree?

function DECISION-TREE-LEARNING($examples, attributes, parent\_examples$) **returns** a tree

```
if examples is empty then return PLURALITY-VALUE(parent_examples)
else if all examples have the same classification then return the classification
else if attributes is empty then return PLURALITY-VALUE(examples)
else
    A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
    tree ← a new decision tree with root test A
    for each value v_k of A do
        exs ← {e : e ∈ examples and e.A = v_k}
        subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
        add a branch to tree with label (A = v_k) and subtree subtree
    return tree
```

Decision Trees

- Learning: how to learn a good decision tree?

$$\boxed{A \leftarrow \operatorname{argmax}_{a \,\in\, attributes} \ \textsc{Importance}(a, examples)}$$

  - Maximize the information gain: the expected reduction in entropy

$$Gain(A) = B(\tfrac{p}{p+n}) - Remainder(A)$$

    - The entropy of a Boolean random variable:

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$$

    - The expected entropy remaining after testing A

$$Remainder(A) = \sum_{k=1}^{d} \tfrac{p_k+n_k}{p+n} B(\tfrac{p_k}{p_k+n_k})$$

Decision Trees

- Example

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

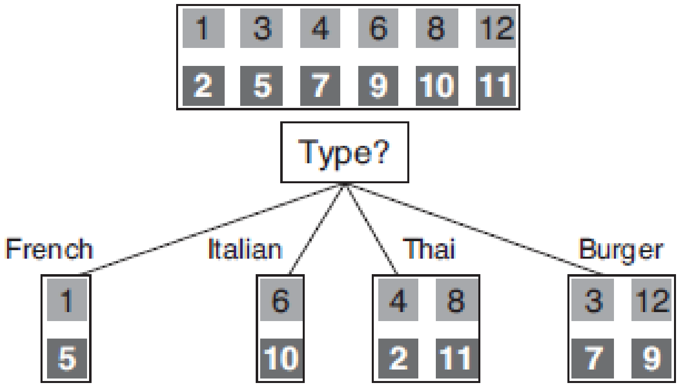
Decision Trees

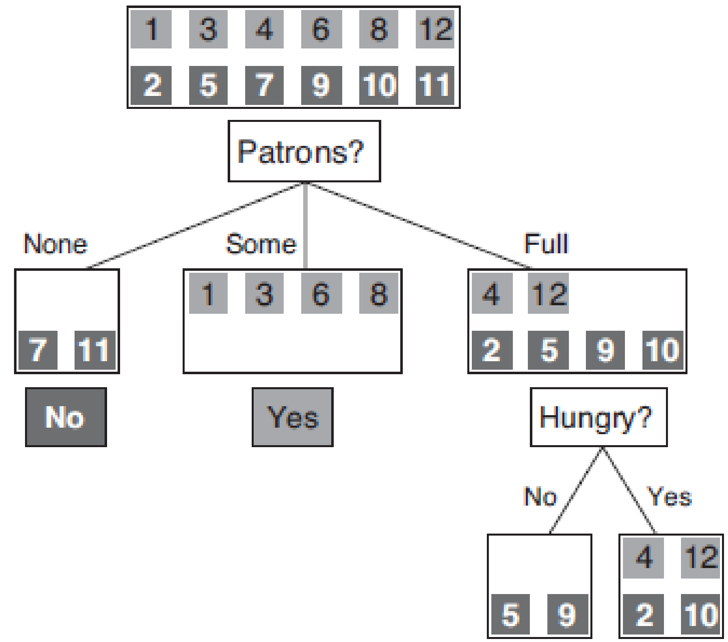
$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

$$B(q) = -(q \log_2 q + (1-q)\log_2(1-q))$$

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B(\frac{p_k}{p_k+n_k})$$

- Example



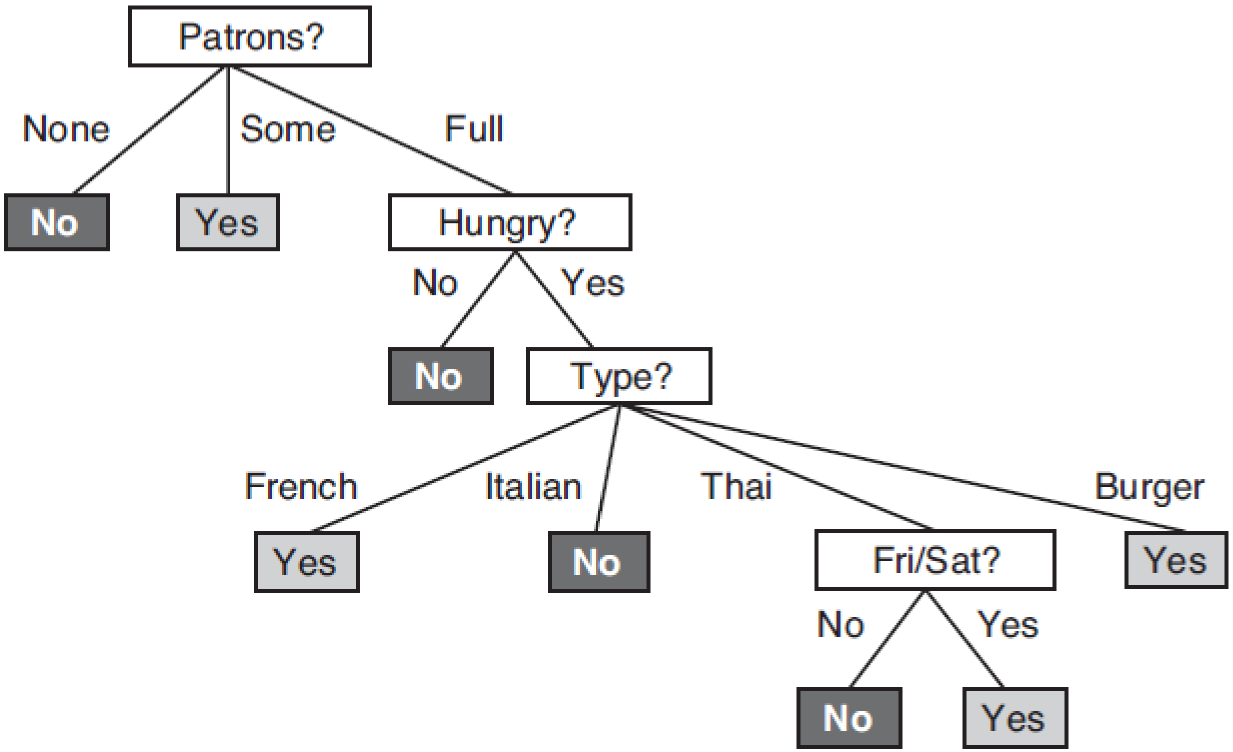


$$Gain(Type)$$
$$= 1 - [\frac{2}{12}B(\frac{1}{2}) + \frac{2}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{2}{4}) + \frac{4}{12}B(\frac{2}{4})]$$
$$= 0 \text{ bits}$$

$$Gain(Patrons)$$
$$= 1 - [\frac{2}{12}B(\frac{0}{2}) + \frac{4}{12}B(\frac{4}{4}) + \frac{6}{12}B(\frac{2}{6})]$$
$$\approx 0.541 \text{ bits}$$

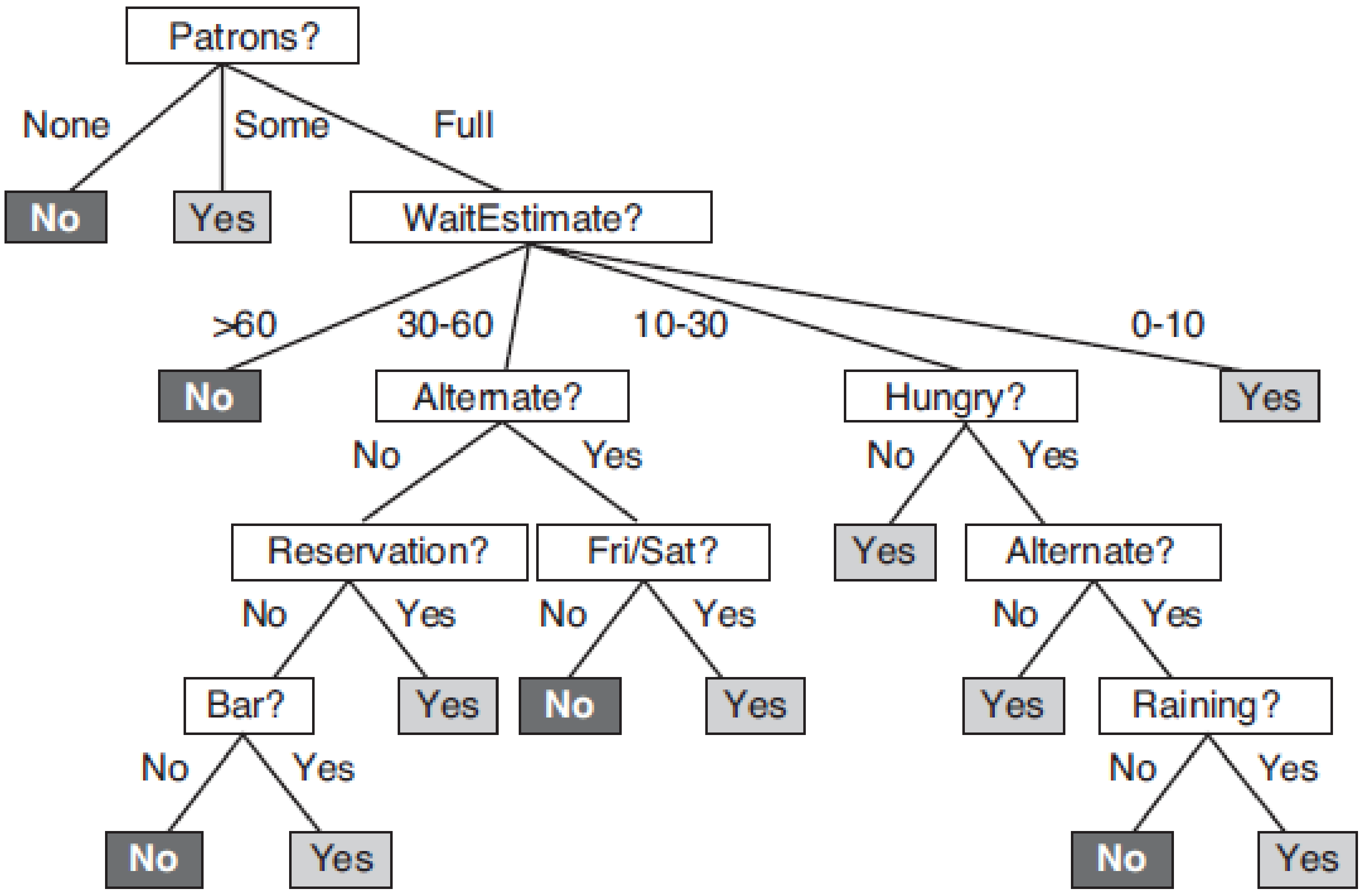
Decision Trees

- Example



Decision Trees

- Example

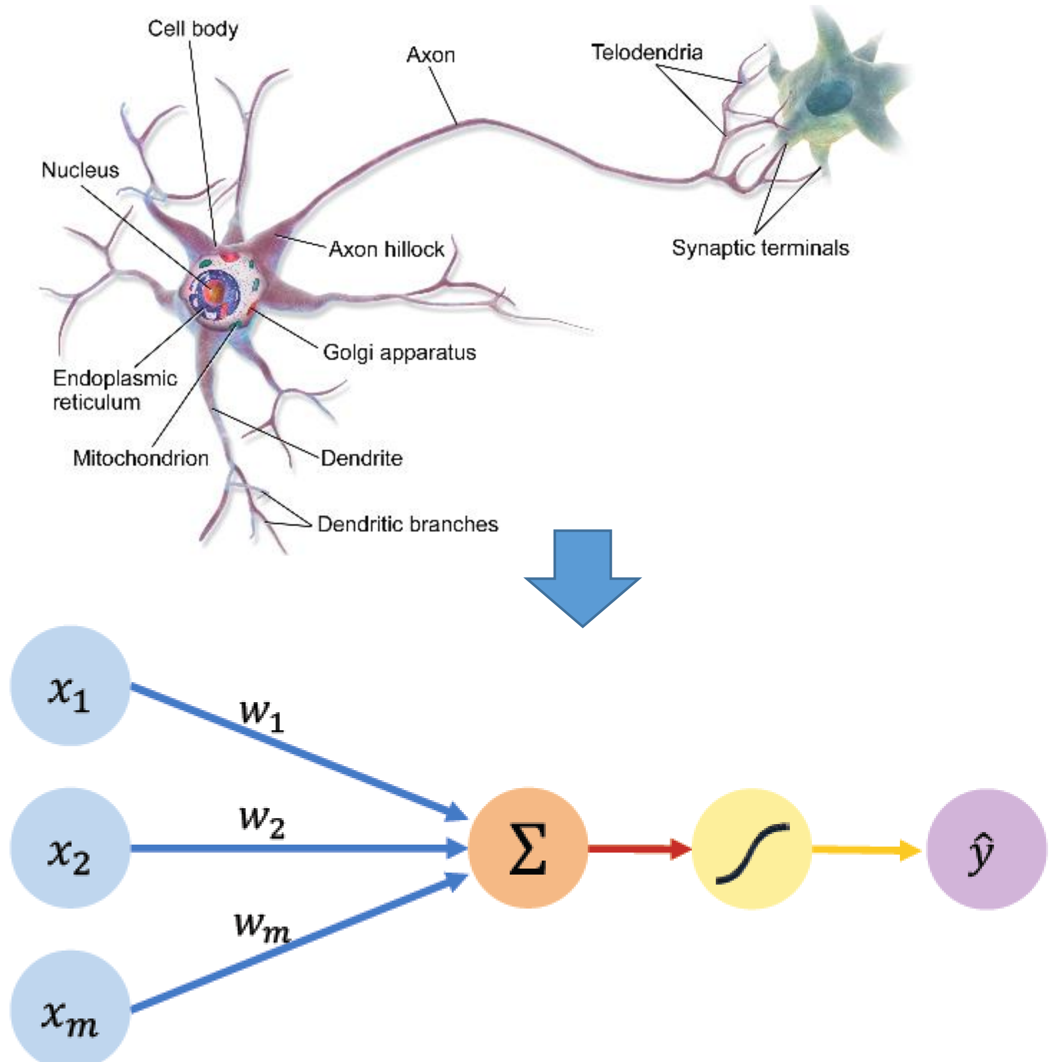


Decision Trees

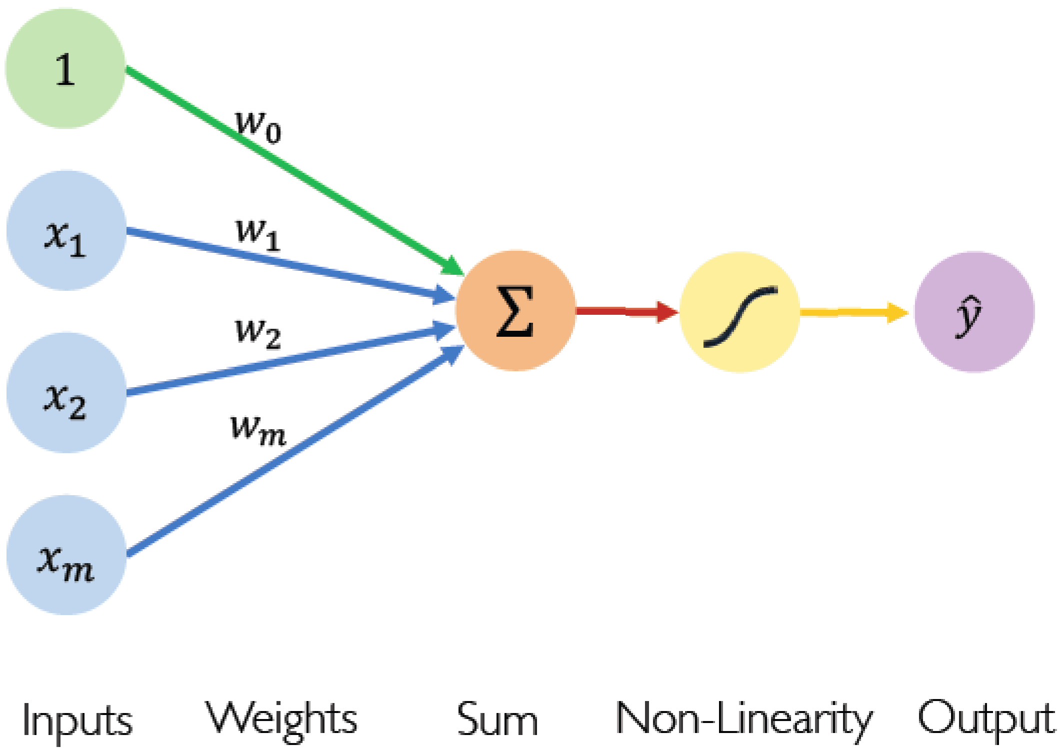
- The decision-tree-learning algorithm adopts a greedy divide-and-conquer strategy.
- The learning algorithm looks at the examples, the set of examples is crucial for constructing the tree.
- For decision trees, decision tree pruning combats overfitting.
- DT is possible for a human to understand the reason for the output of the learning algorithm.

Perceptron

- Inspired by human neuron



Perceptron



Output

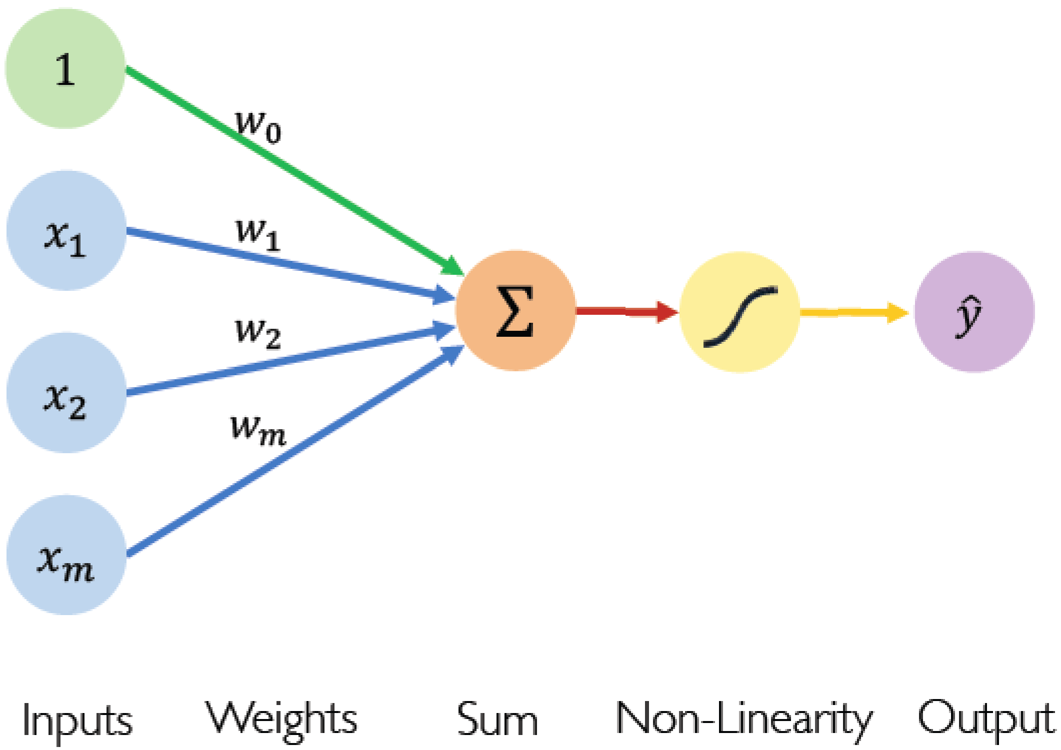
Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

Non-linear activation function

Bias

Perceptron

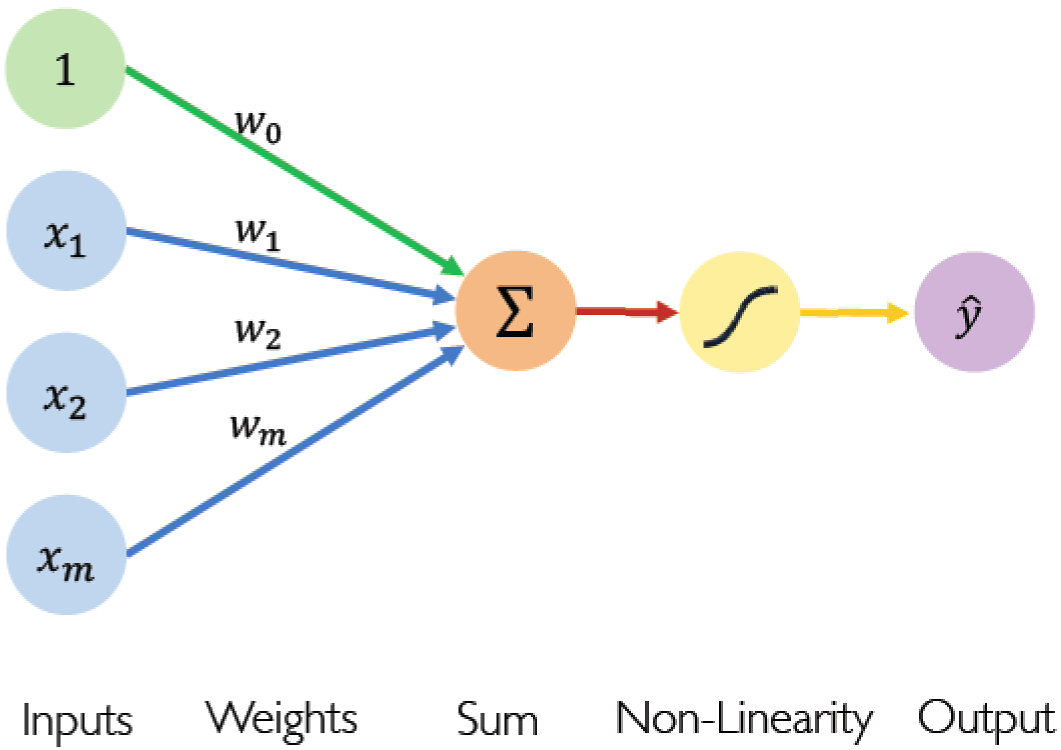


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Perceptron

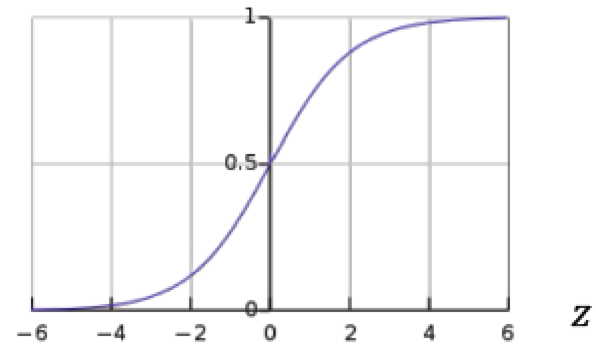


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

- Example: sigmoid function

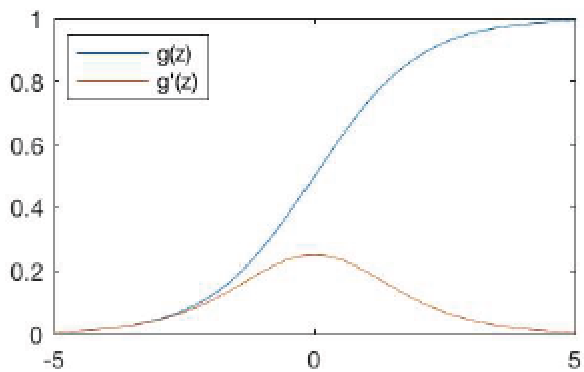
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Perceptron

- Activation function:
  - The purpose of activation functions is to introduce nonlinearities into the network
  - All activation functions are non-linear

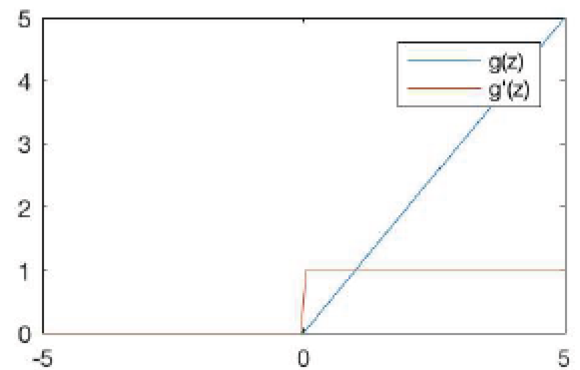
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

Rectified Linear Unit (ReLU)

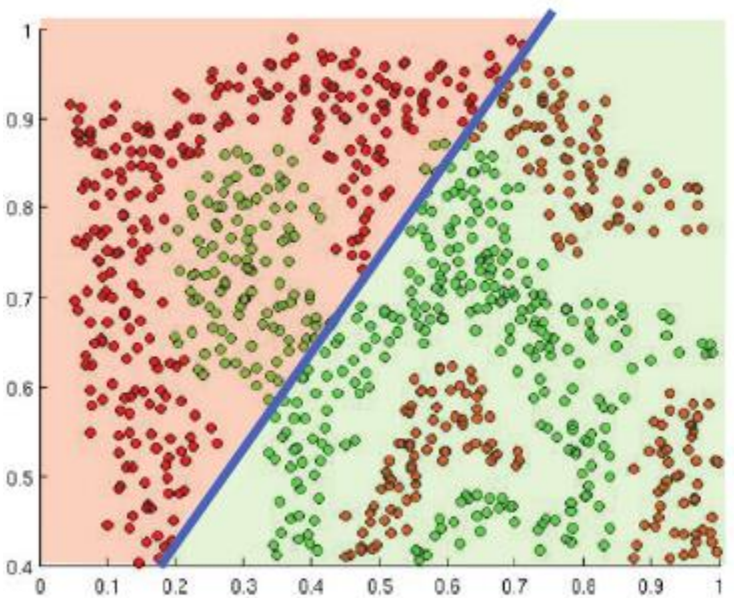


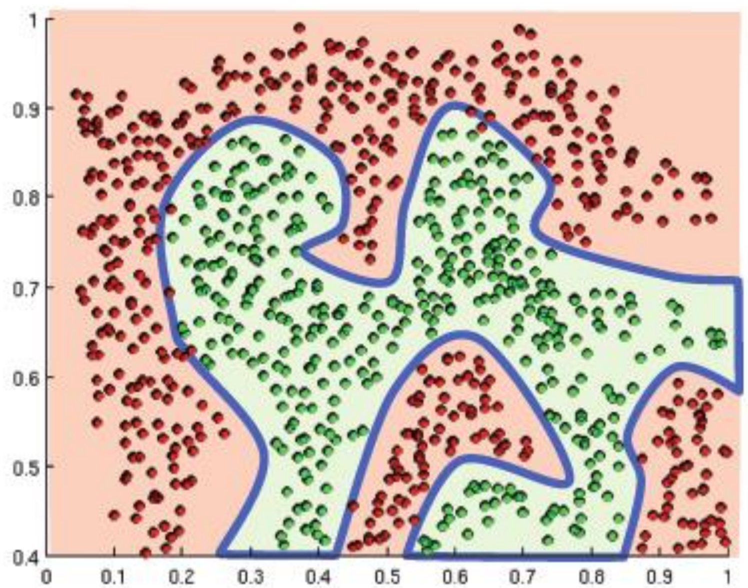
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Perceptron

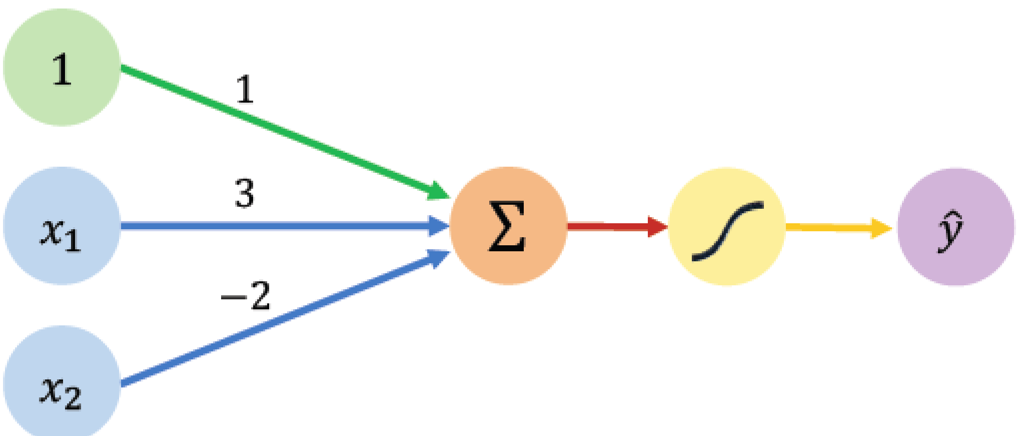
- Activation function:
  - Linear activation functions produce linear decisions no matter the network size
  - Nonlinearities allow us to approximate arbitrarily complex functions





Perceptron

- Example



We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

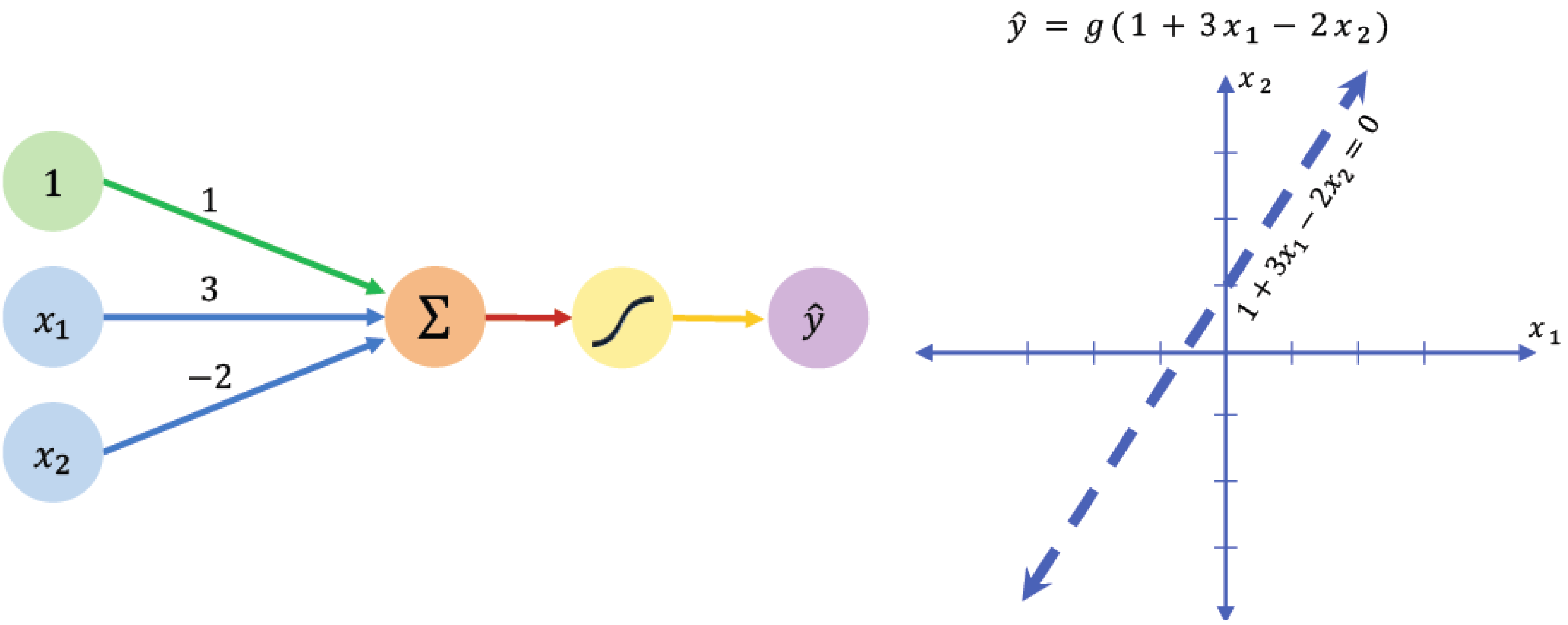
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

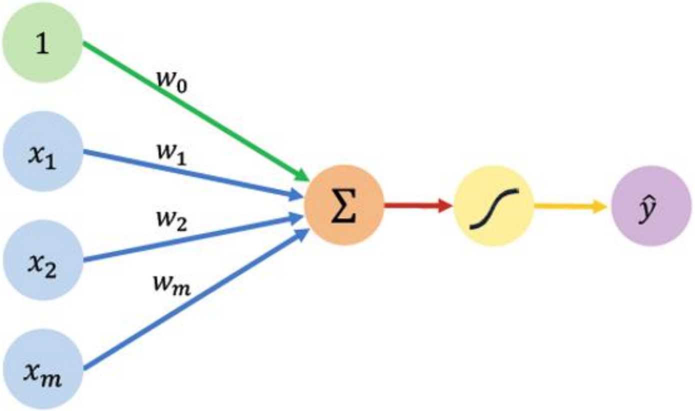
$$\hat{y} = g\left(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}}\right)$$

Perceptron

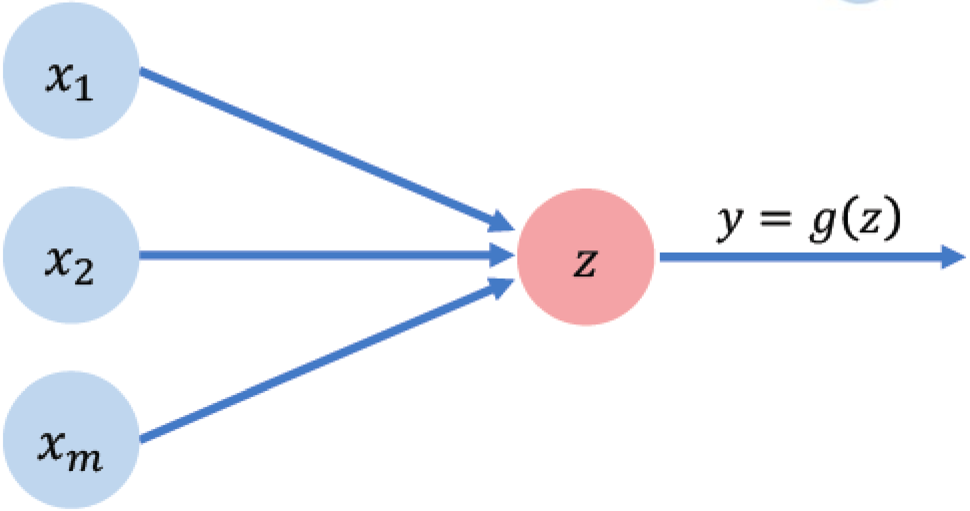
- Example



Perceptron



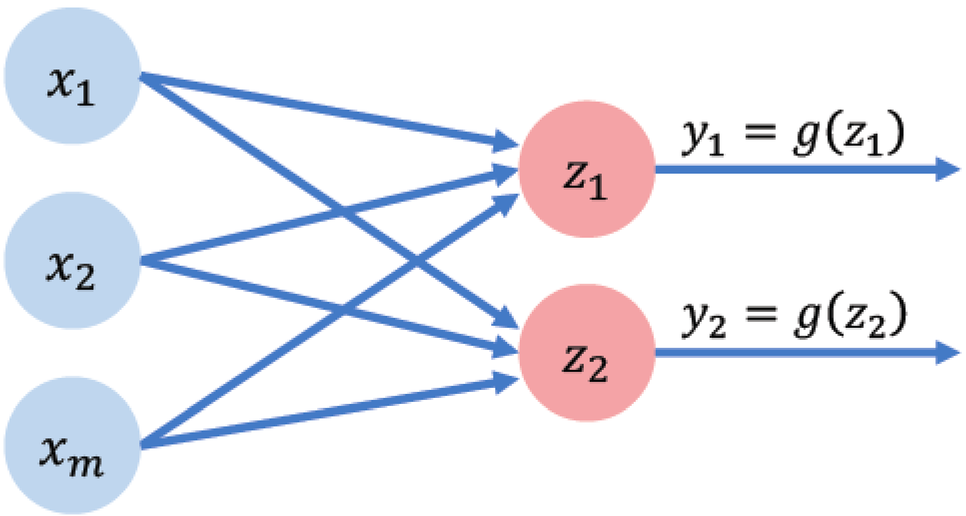
- The Simplified Perceptron



$$z = w_0 + \sum_{j=1}^{m} x_j\, w_j$$

Perceptron

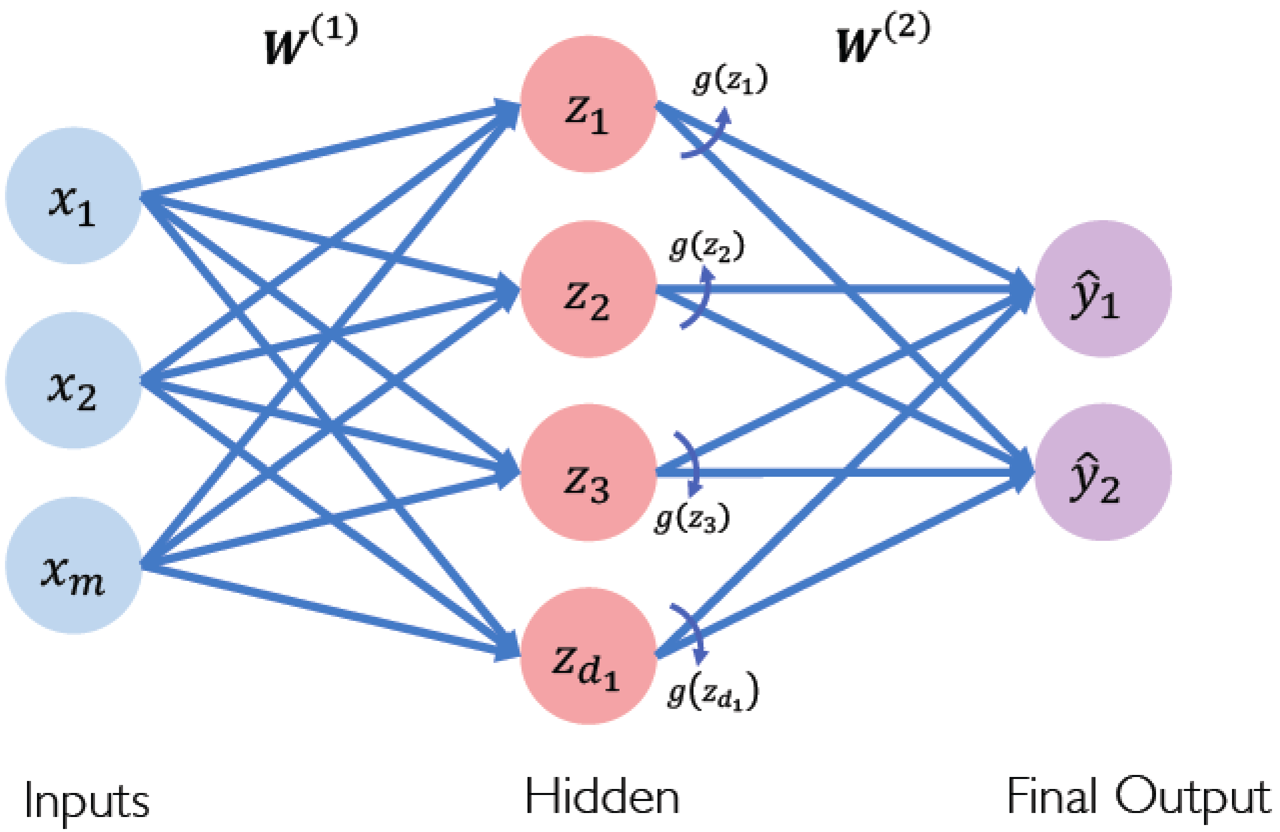
- Multi Output Perceptron



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Artificial Neural Network

- Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j\, w_{j,i}^{(2)}\right)$$

Artificial Neural Network

- Loss Optimization

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:

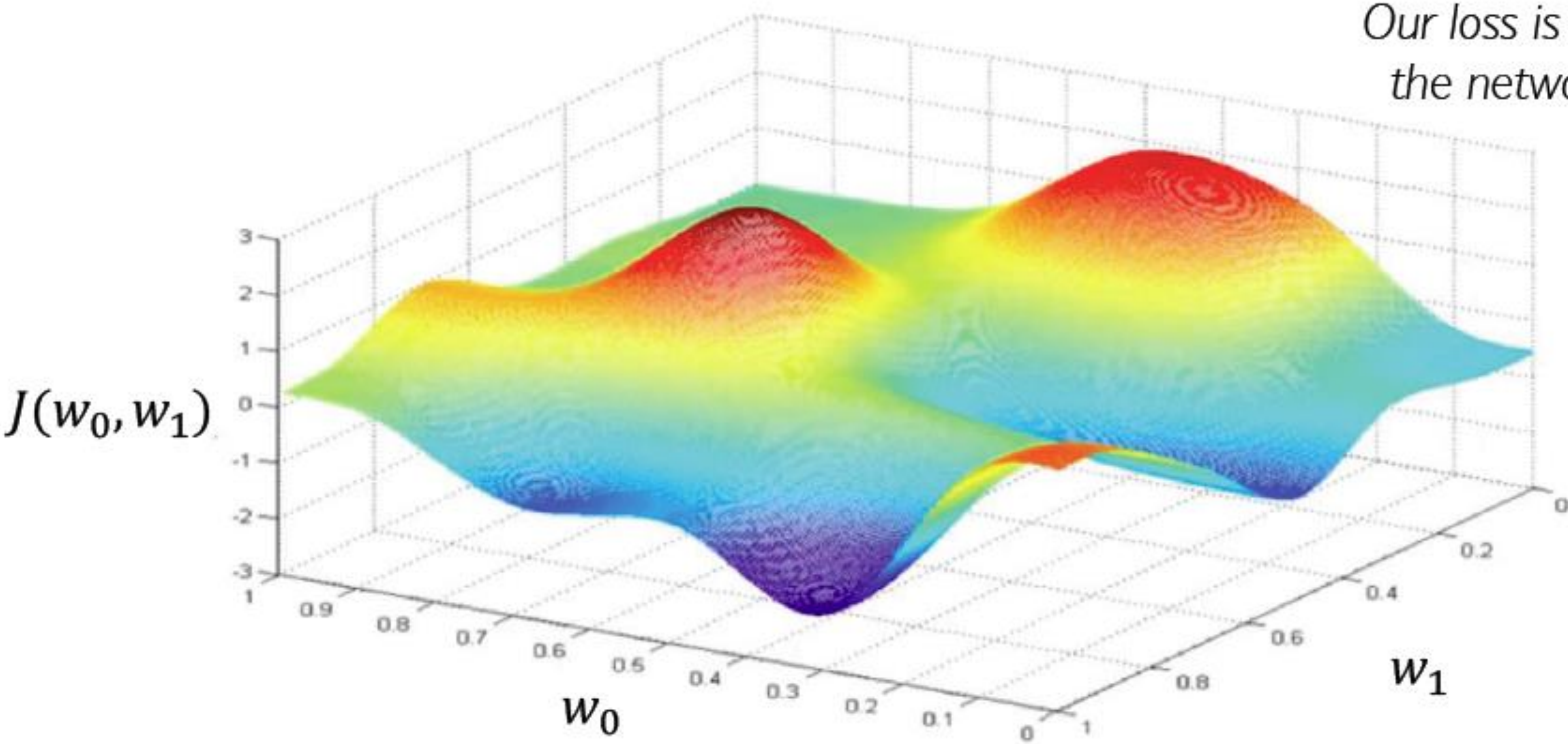
$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\right\}$$

Artificial Neural Network

- Loss Optimization – Gradient Descent

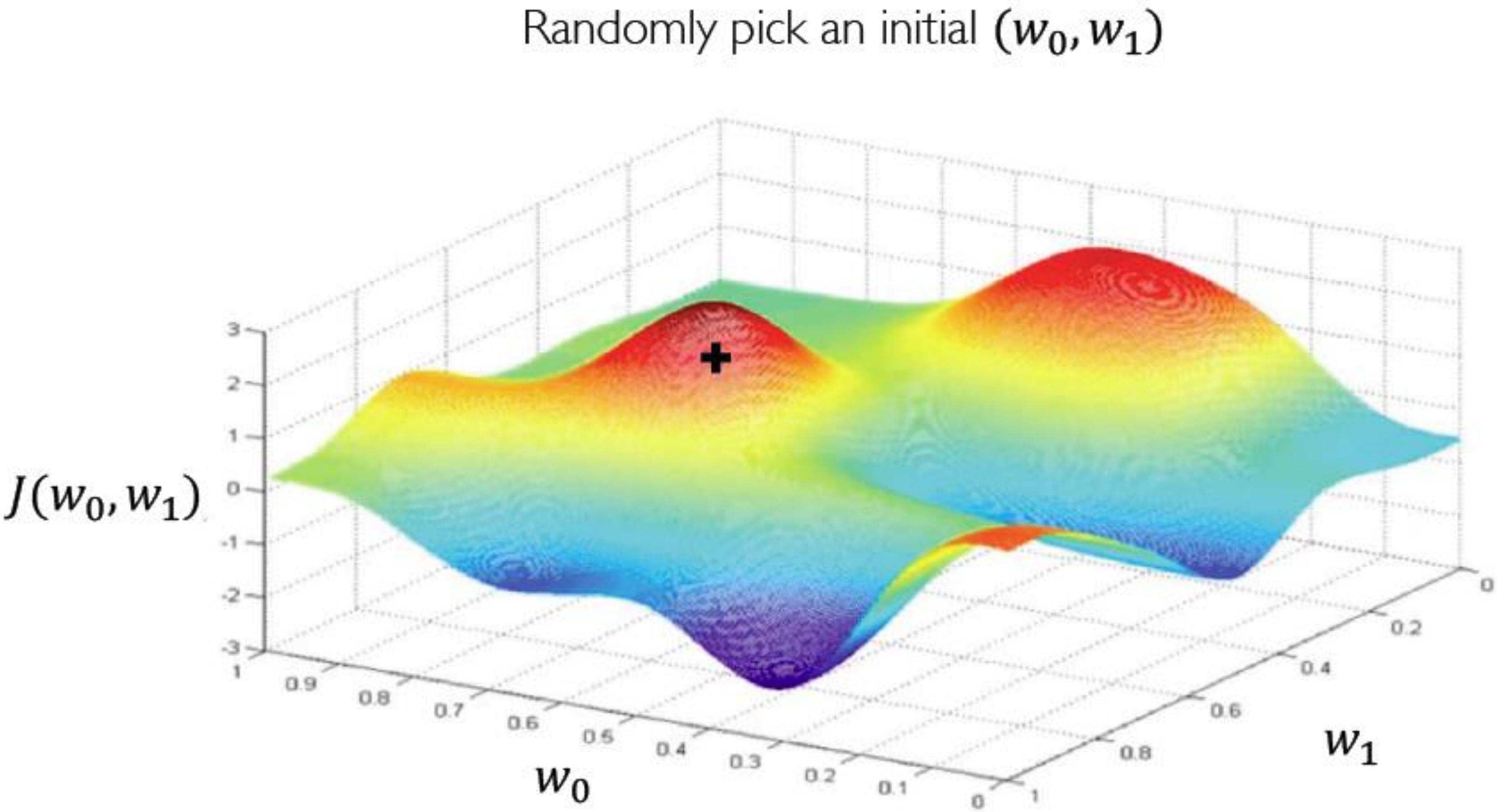
$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

Remember:
Our loss is a function of the network weights!



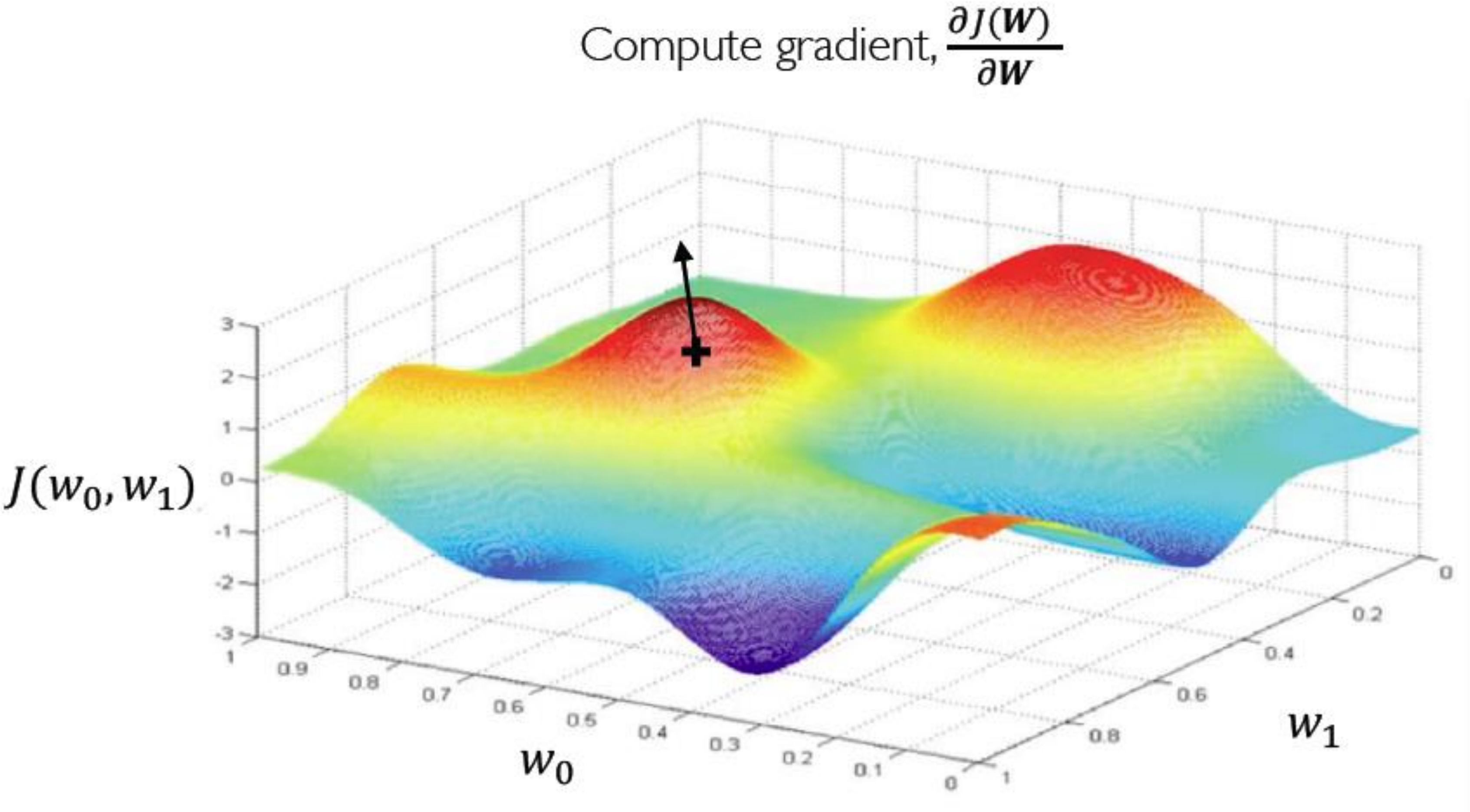
Artificial Neural Network

- Loss Optimization – Gradient Descent



Artificial Neural Network

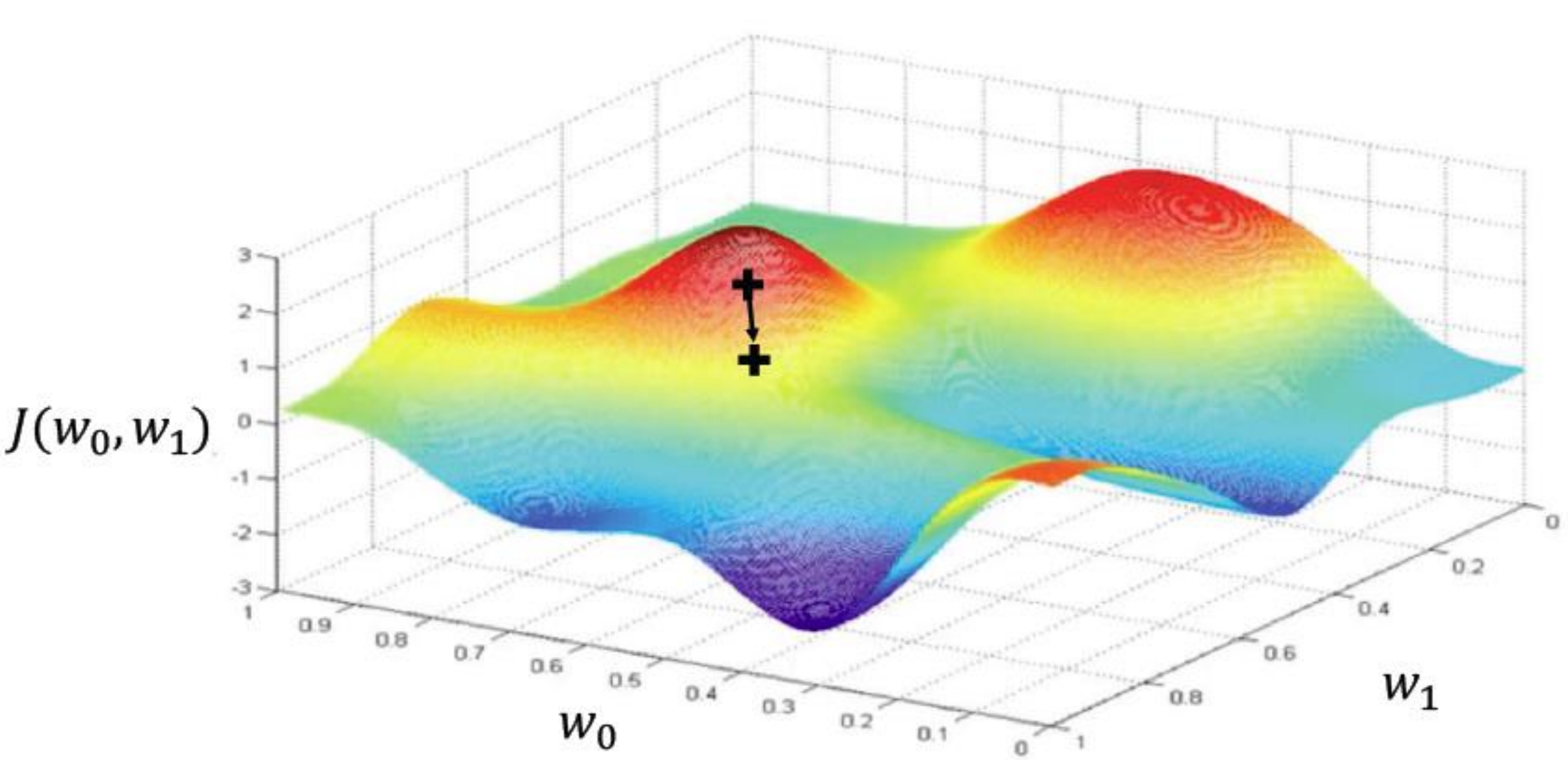
- Loss Optimization – Gradient Descent



Artificial Neural Network

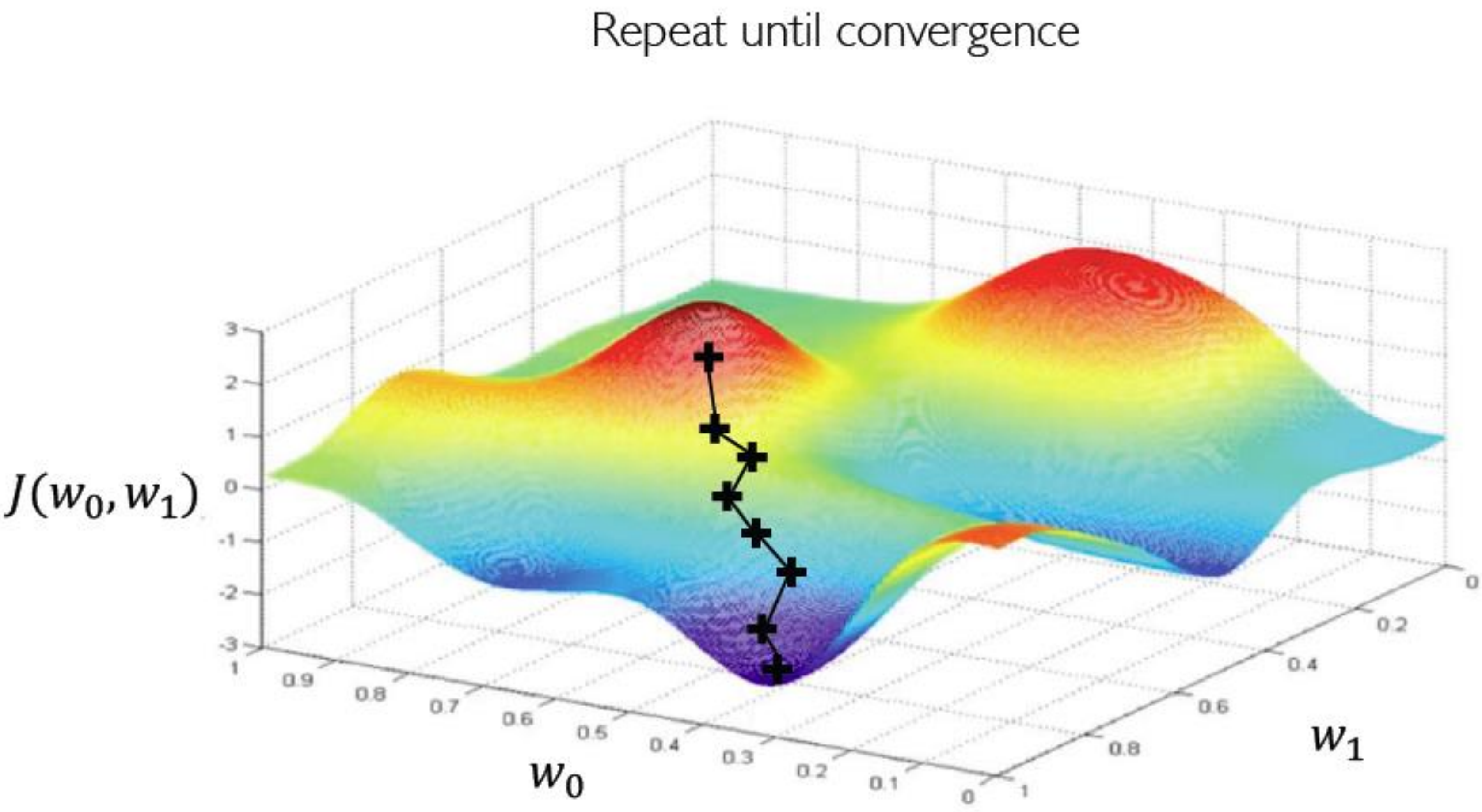
- Loss Optimization – Gradient Descent

Take small step in opposite direction of gradient



Artificial Neural Network

- Loss Optimization – Gradient Descent



Artificial Neural Network

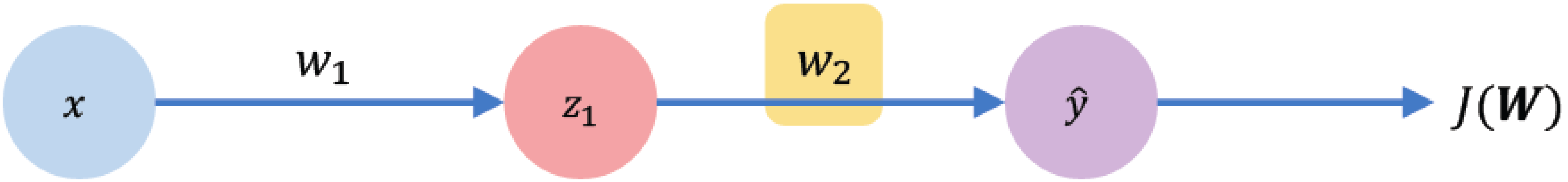
- Loss Optimization – Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Artificial Neural Network

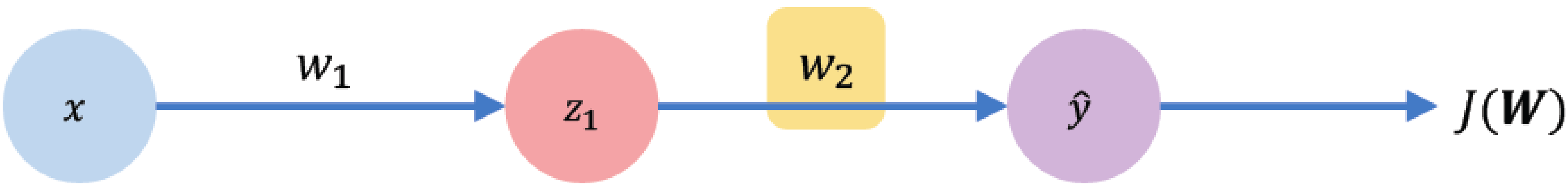
- Computing Gradients: Backpropagation



How does a small change in one weight (ex. $\boldsymbol{w_2}$) affect the final loss $J(\boldsymbol{W})$?

Artificial Neural Network

- Computing Gradients: Backpropagation

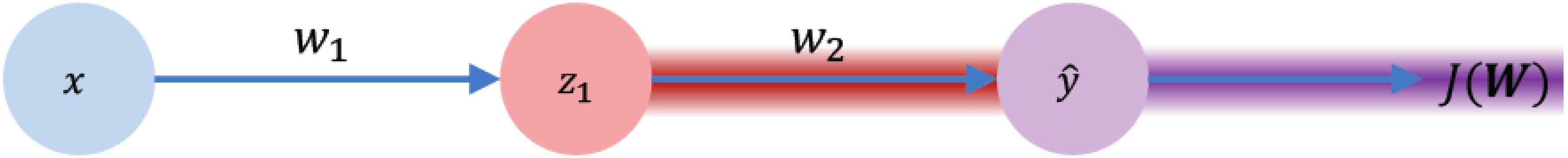


$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

Let's use the chain rule!

Artificial Neural Network

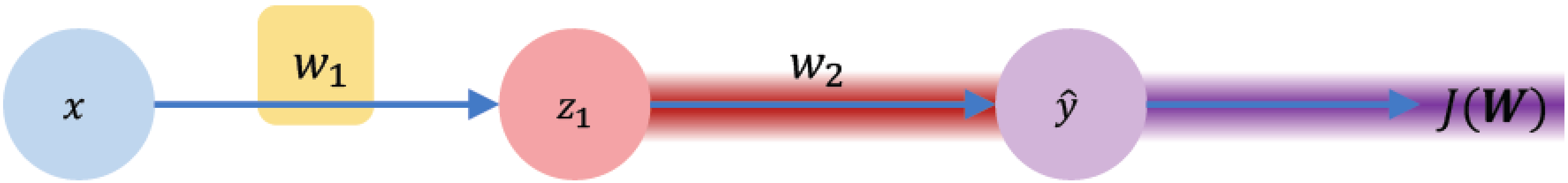
- Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \underbrace{\frac{\partial J(\boldsymbol{W})}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}$$

Artificial Neural Network

- Computing Gradients: Backpropagation

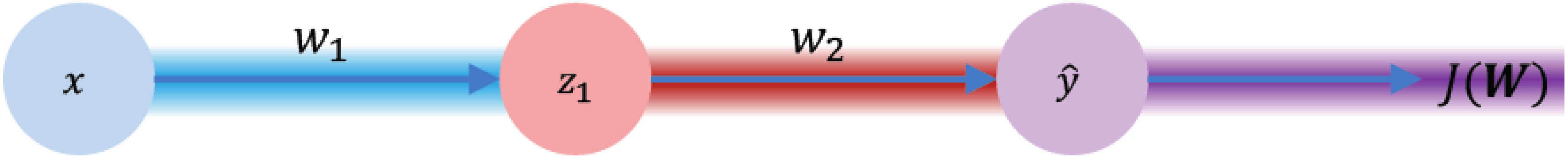


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule! Apply chain rule!

Artificial Neural Network

- Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Artificial Neural Network

```
function BACK-PROP-LEARNING(examples, network) returns a neural network
   inputs: examples, a set of examples, each with input vector x and output vector y
           network, a multilayer network with L layers, weights w_{i,j}, activation function g
   local variables: Δ, a vector of errors, indexed by network node

   repeat
      for each weight w_{i,j} in network do
         w_{i,j} ← a small random number
      for each example (x, y) in examples do
         /* Propagate the inputs forward to compute the outputs */
         for each node i in the input layer do
            a_i ← x_i
         for ℓ = 2 to L do
            for each node j in layer ℓ do
               in_j ← Σ_i w_{i,j} a_i
               a_j ← g(in_j)
         /* Propagate deltas backward from output layer to input layer */
         for each node j in the output layer do
            Δ[j] ← g′(in_j) × (y_j − a_j)
         for ℓ = L − 1 to 1 do
            for each node i in layer ℓ do
               Δ[i] ← g′(in_i) Σ_j w_{i,j} Δ[j]
         /* Update every weight in network using deltas */
         for each weight w_{i,j} in network do
            w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
   until some stopping criterion is satisfied
   return network
```

Artificial Neural Network

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton*, R. R. Salakhutdinov

+ See all authors and affiliations

Science 28 Jul 2006:
Vol. 313, Issue 5786, pp. 504-507
DOI: 10.1126/science.1127647

Readings

- Artificial Intelligence
  - Chapter 18.1 -18.5, 18.7

- Homework 6