RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Machine Learning

Basic Algorithms

Edgar Granados

Classification

- Set of training examples: $X_{train} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Learn: $f(x_i) = y_i$
- Evaluate performance: $X_{test} = \{(x_{n+1}, y_{n+1}), \ldots, (x_m, y_m)\}$

↳ Not known while training

Naive Bayes Algorithm

- Recall Bayes Rule:

$$p(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

- Likelihood:
  - $L(x) = \frac{P(y = true \,|\, x)}{P(y = fasle \,|\, x)} \simeq \frac{P(x|y=true)P(y=true)}{P(x|y=false)P(y=false)}$

if $L(x) > 1$
  y is true
if $L(x) < 1$
  y is false

Naive Bayes Algorithm

- Estimate $P(y)$ from the training examples
  - $P(y = true) = \dfrac{\#\text{ times } y_i = true \text{ in } X_{train}}{n}$
  - $P(y = false) = \dfrac{\#\text{ times } y_i = false \text{ in } X_{train}}{n}$

Naive Bayes Algorithm

X_{train}

- From the training examples

1. Define ϕ, a function that maps each input x into a features vector

$$\phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_p(x))$$

Naive Bayes Algorithm

- From the training examples

1. Define ϕ, a function that maps each input x into a features vector
2. Estimate the probability distribution of each feature ϕ_j

$$P(\phi_j(x) = v \mid y = true) = \frac{\#\ \phi_j(x) = v \text{ AND } y = true \in X_{train}}{\#\ y = true \in X_{train}}$$

$$P(\phi_j(x) = v \mid y = true) = \frac{\#\ \phi_j(x) = v \text{ AND } y = false \in X_{train}}{\#\ y = false \in X_{train}}$$

Naive Bayes Algorithm

- From the training examples

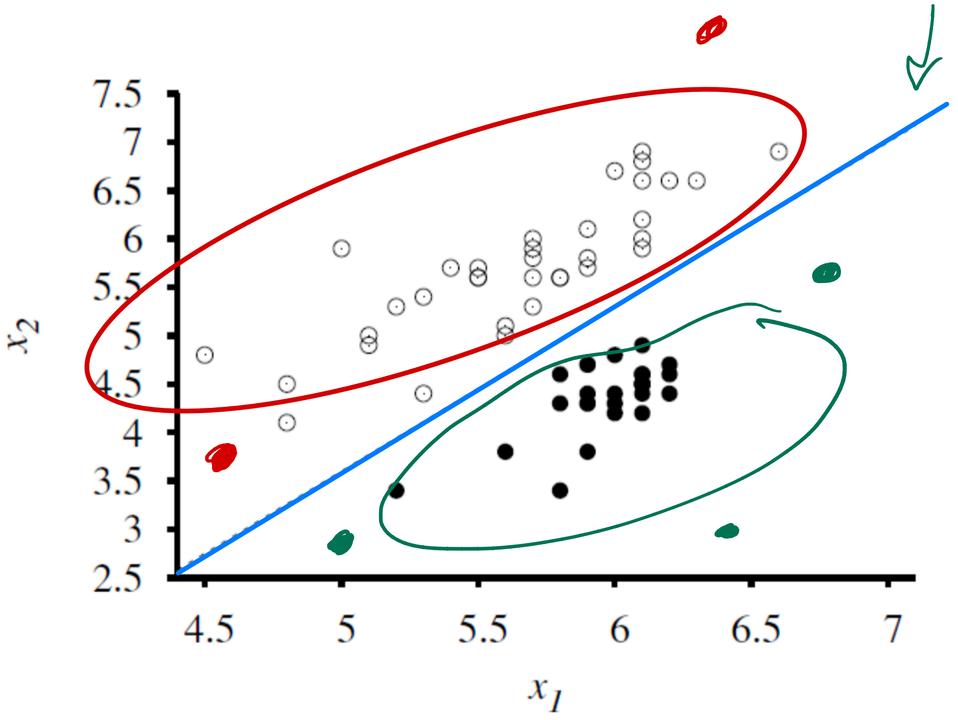
1. Define ϕ, a function that maps each input x into a features vector
2. Estimate the probability distribution of each feature ϕ_j
3. Estimate $P(x \mid y)$ using the naive Bayes assumption

$$p(x|y = true) = \prod_{j=1}^{l} p(\phi_j(x)|y = true)$$

$$p(x|y = false) = \prod_{j=1}^{l} p(\phi_j(x)|y = false)$$

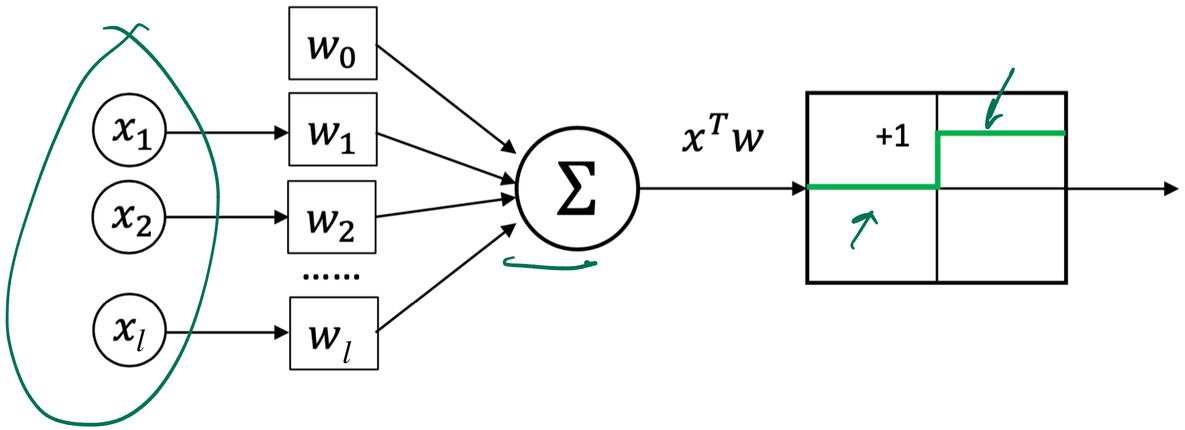
Linear Classifier

- A linear classifier for l features is an $(l-1)$-dimensional plane separating the positive/negative labels



Linear Classifier

Perceptron



- Learning:
  - Find the appropriate weights

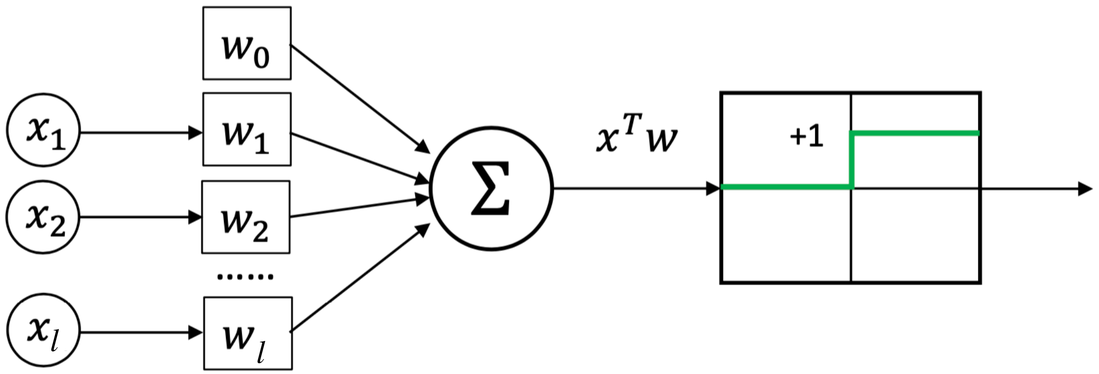
$$y = \begin{cases} 1, & w_0 + w_1\phi_1(x_1) + \ldots + w_l\phi_l(x_l) \geq 0 \\ 0 & w_0 + w_1\phi_1(x_1) + \ldots + w_l\phi_1(x_l) < 0 \end{cases}$$

$y = ax + b$

$y = w_1 x + w_0$

Linear Classifier

Perceptron



- Invented in 1957
- A single-layer neural network.
  - Deep neural nets are nothing but Perceptrons stacked on top of each other

$$y = \begin{cases} 1, & w_0 + w_1\phi_1(x_1) + \ldots + w_l\phi_l(x_l) \geq 0 \\ 0 & w_0 + w_1\phi_1(x_1) + \ldots + w_l\phi_1(x_l) < 0 \end{cases}$$

Perceptron Algorithm

1. Initialize weights

Perceptron Algorithm

1. Initialize weights
2. $\forall (x_i, y_i) \in \mathcal{X}_{train}$:
   - $f(x_i, w) = w_0 + w_1\phi_1(x_i) + \ldots + w_l\phi_l(x_i)$
   - If $(f(x_i, w) \geq 0$ AND $y_i = true)$ OR $(f(x_i, w) < 0$ AND $y_i = false)$
     - Nothing... move to next example

Perceptron Algorithm

1. Initialize weights
2. $\forall (x_i, y_i) \in \mathcal{X}_{train}$:
   - $f(x_i, w) = w_0 + w_1\phi_1(x_1) + \ldots + w_n\phi_n(x_n)$
   - If ($f(x_i, w) \geq 0$ AND $y_i = true$) OR ($f(x_i, w) < 0$ AND $y_i = false$):
     - Nothing... move to next example
   - Else: Update weights

Perceptron Algorithm

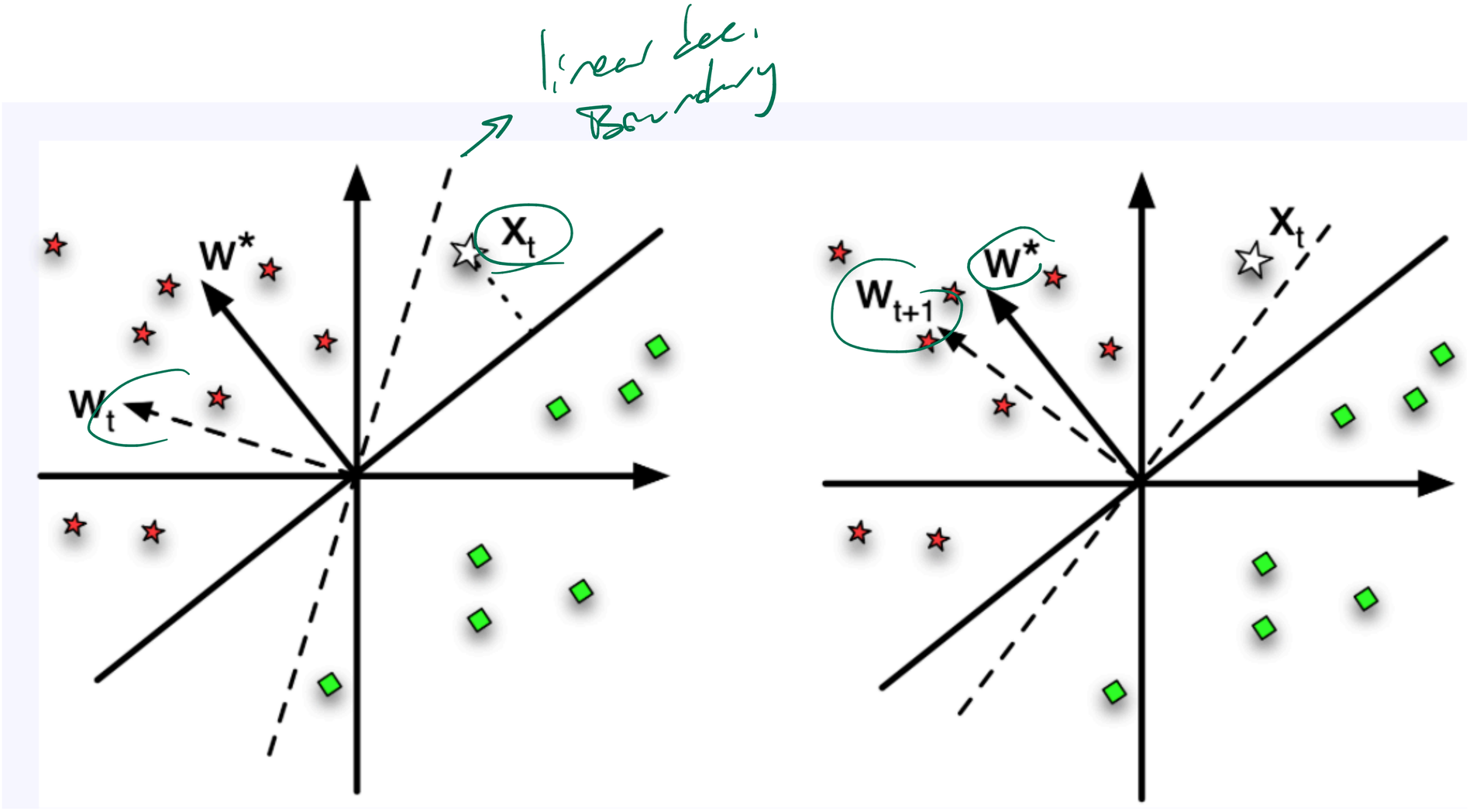
Updating Weights

- if ($f(x_i, w) < 0$ AND $y_i = true$):
  - $w_0 \leftarrow w_0 + 1$
  - $w_j \leftarrow w_j + \phi_j(x_i)$
- if ($f(x_i, w) \geq 0$ AND $y_i = false$):
  - $w_0 \leftarrow w_0 - 1$
  - $w_j \leftarrow w_j - \phi(x_i)$

Perceptron Algorithm

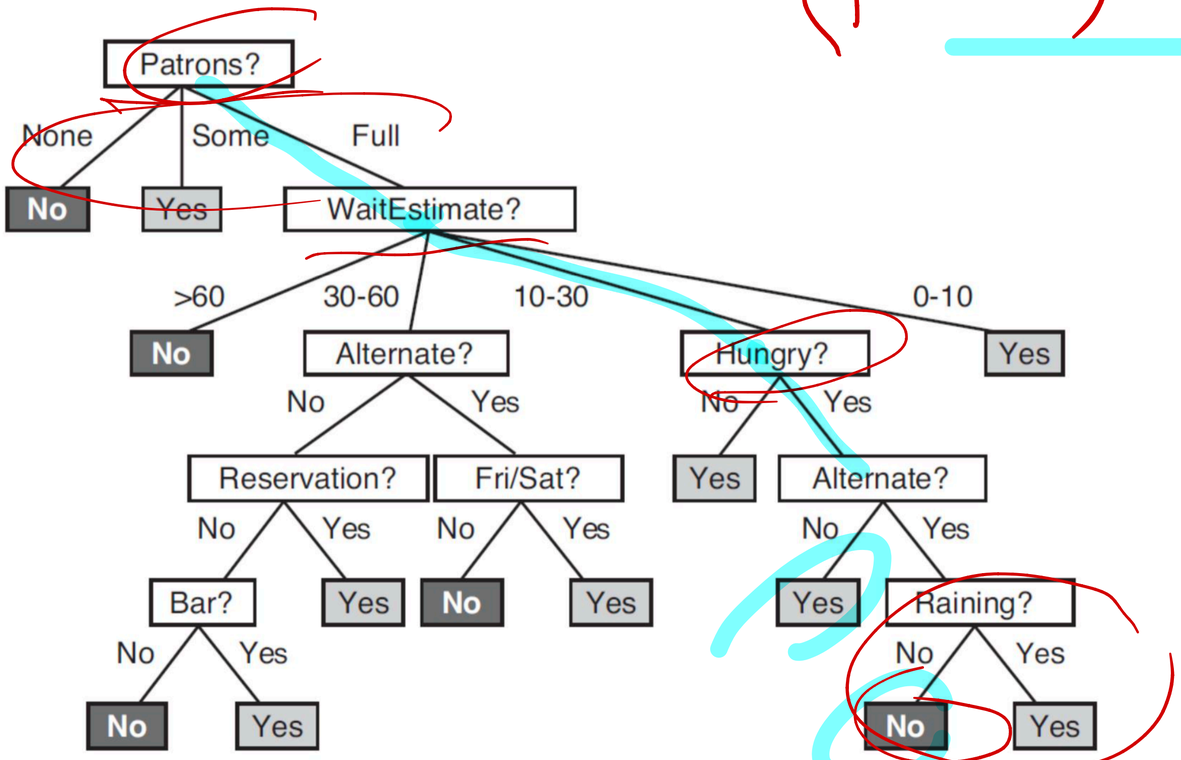
1. Initialize weights
2. $\forall (x_i, y_i) \in \mathcal{X}_{train}$:
   - $f(x_i, w) = w_0 + w_1\phi_1(x_1) + \ldots + w_n\phi_n(x_n)$
   - If ($f(x_i, w) \geq 0$ AND $y_i = true$) OR ($f(x_i, w) < 0$ AND $y_i = false$):
     - Nothing... move to next example
   - Else:
     - if ($f(x_i, w) < 0$ AND $y_i = true$):
       - $w_0 \leftarrow w_0 + 1$
       - $w_j \leftarrow w_j + \phi(x_i)$
     - if ($f(x_i, w) \geq 0$ AND $y_i = false$):
       - $w_0 \leftarrow w_0 - 1$
       - $w_j \leftarrow w_j - \phi(x_i)$
3. Repeat until:
   1. No changes
   2. Time limit reached

Perceptron



Decision Tree

- Decide the outcome based on a set of attributes



Decision Tree

Algorithm

function LEARN-DECISION-TREE($examples$, $attributes$, $parent\_examples$) **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
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v of A **do**
    $exs \leftarrow \{e : e \in examples \text{ and } e.A = v\}$
    $subtree \leftarrow$ LEARN-DECISION-TREE(exs, $attributes - A$, $examples$)
    add a branch to $tree$ with label $(A = v)$ and subtree $subtree$
  return $tree$

Decision Tree

Algorithm: Importance

More entropy → Less info

- Decide over which attribute to split
- **Entropy:**
  - $H(x) = -\sum_{i=1}^{k} P(x_i) log_2 P(x_i)$
  - $B(q) = -(q log_2(q) + (1-q) log_2(1-q))$

X ~ fair coin, X' ~ coin that mostly T

$H(x) = -(.5 \log_2 .5 + .5 \log_2 .5) = 1$

$H(x') = -(.99 \log_2 .99 + .01 \log_2 0.01) = .066$

To RV with prob q

Decision Tree

Algorithm: Importance

- Given attribute A with d distinct values: $a_1, a_2, \ldots, a_d$
  - $\mathcal{X}_{train}$ is divided in d subsets

$n^+ \sim$ Positive labels
$n^- \sim$ Negative labels

$a_k \sim n_k^+ \sim$ Pos. Labels
$n_k^- \sim$ Neg labels

$$Remainder(A) = \sum_{k=1}^{d} \frac{n_k^+ + n_k^-}{n^+ + n^-} B\left(\frac{n_k^+}{n_k^+ + n_k^-}\right)$$

$n^+ + n^-$ → total

Decision Tree

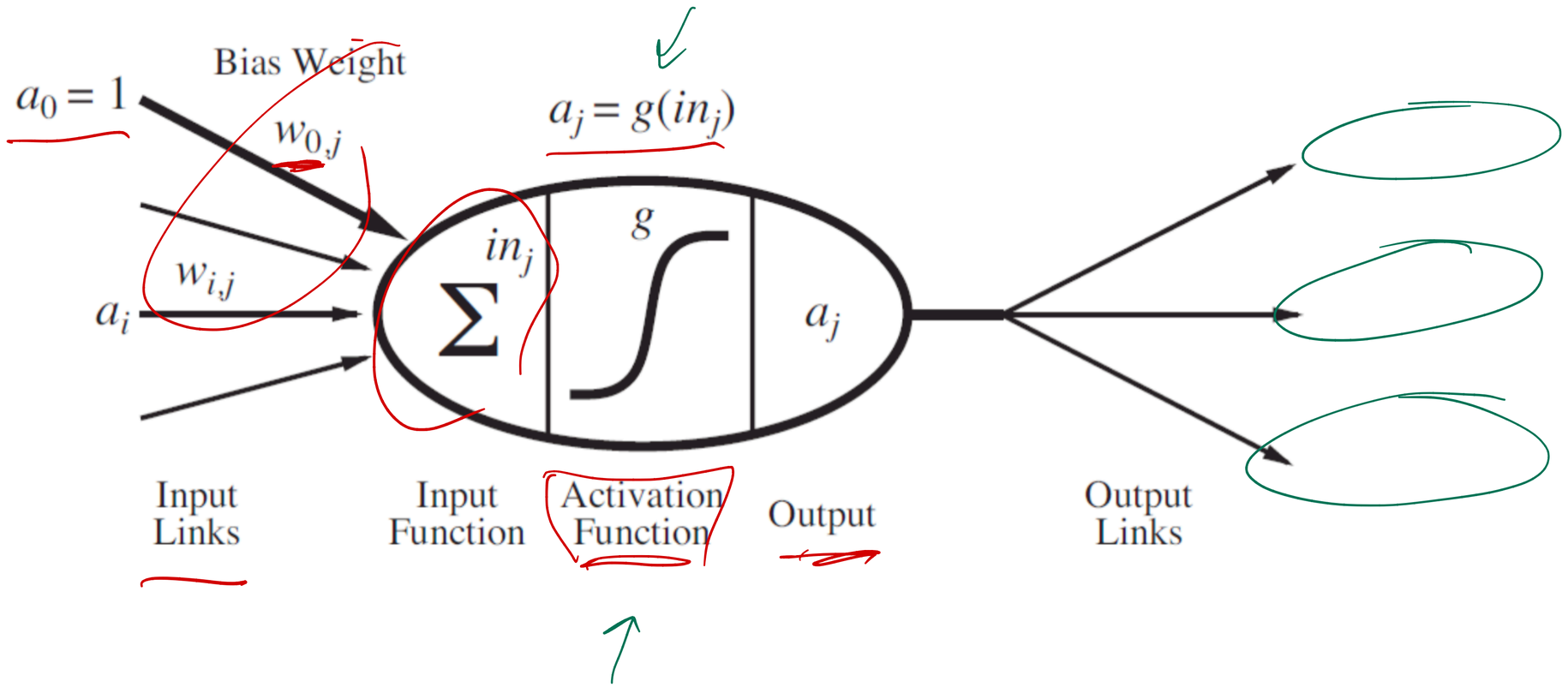
Algorithm: Importance

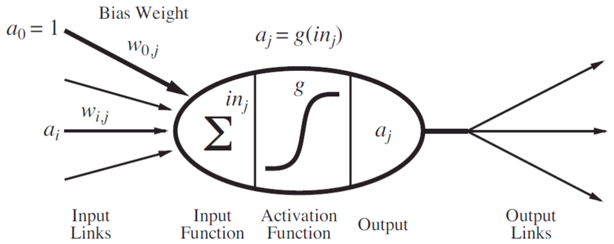
- Information gain: the expected reduction in entropy

$$\text{Importance} = Gain(A) = B\left(\frac{n^+}{n^+ + n^-}\right) - Remainder(A)$$

Neuron

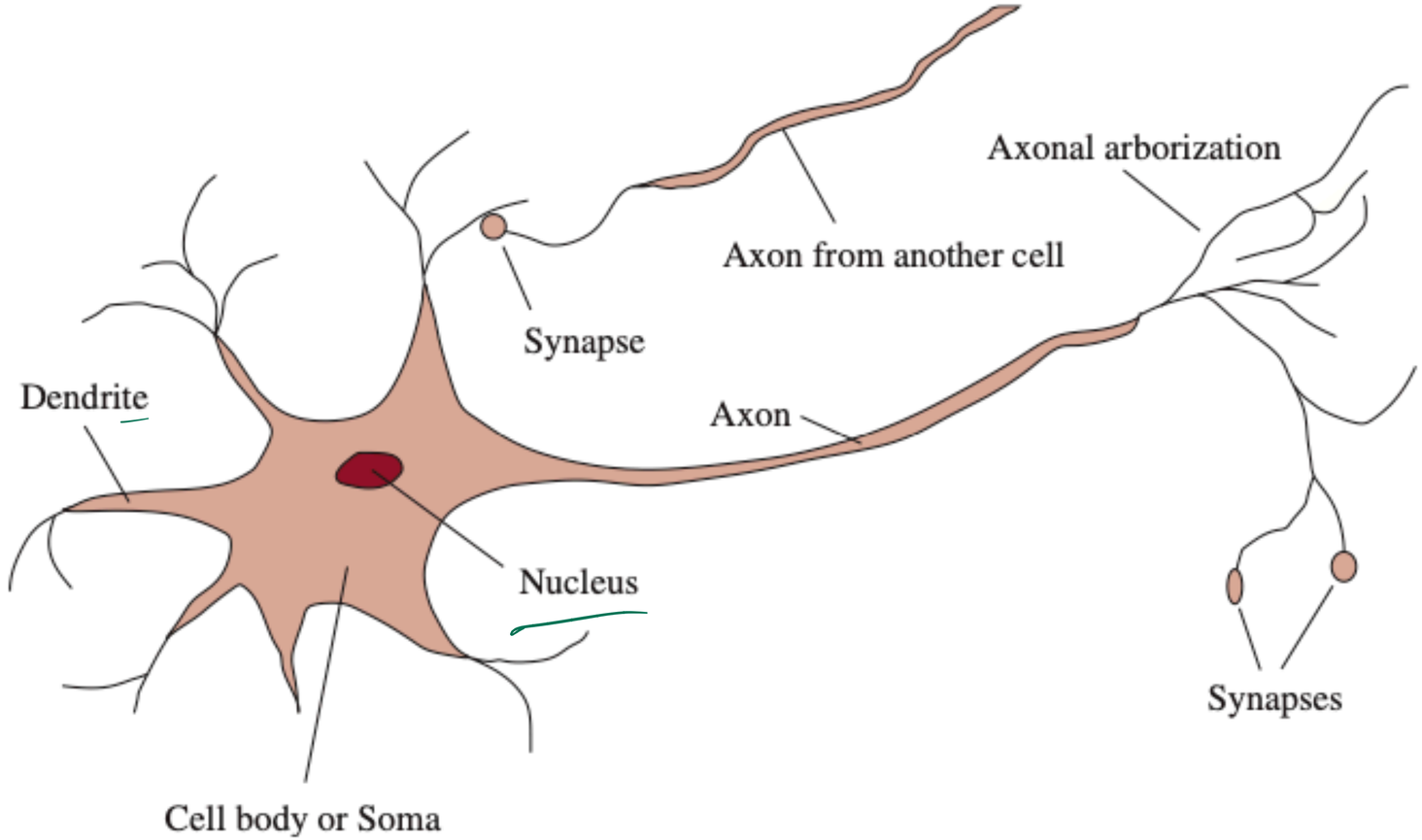
Mathematical Model



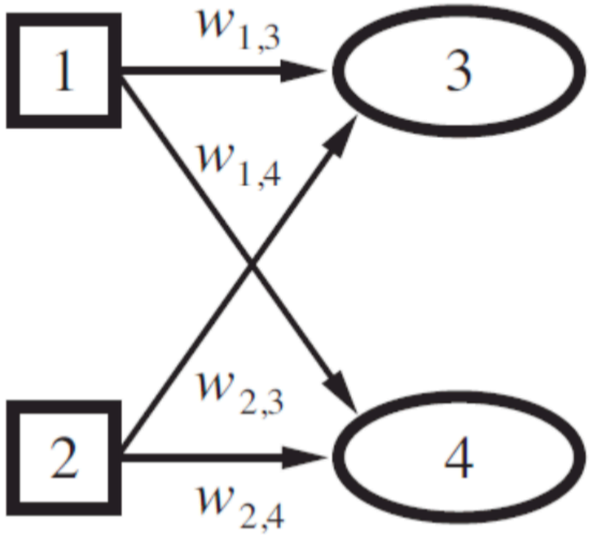


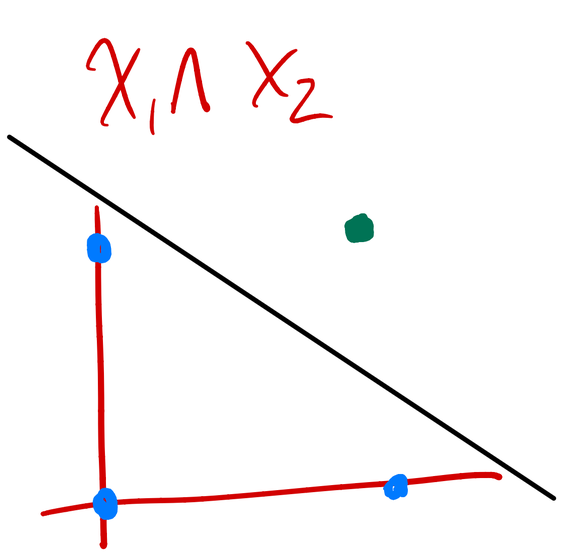
Neuron

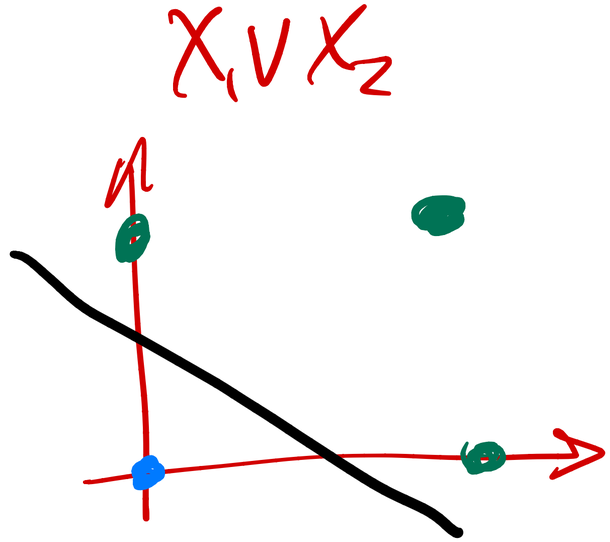
Real Neuron

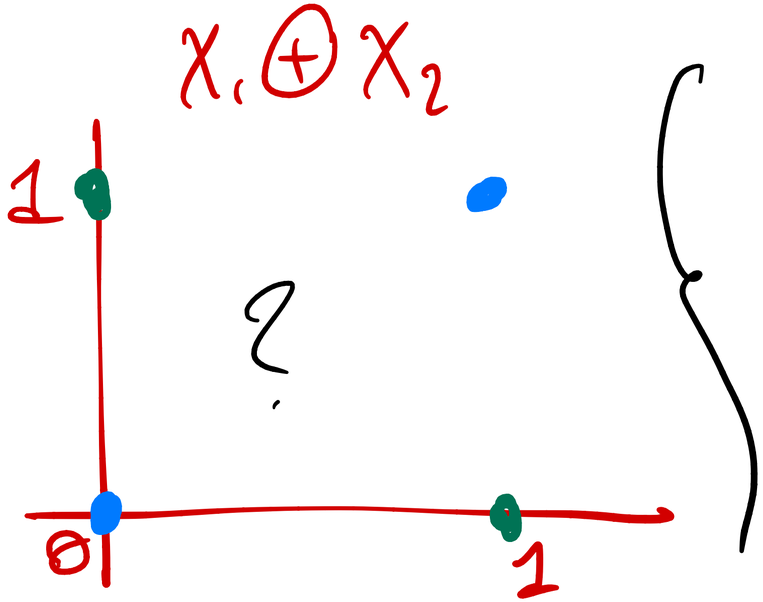


Single Layer Feed-Forward NN

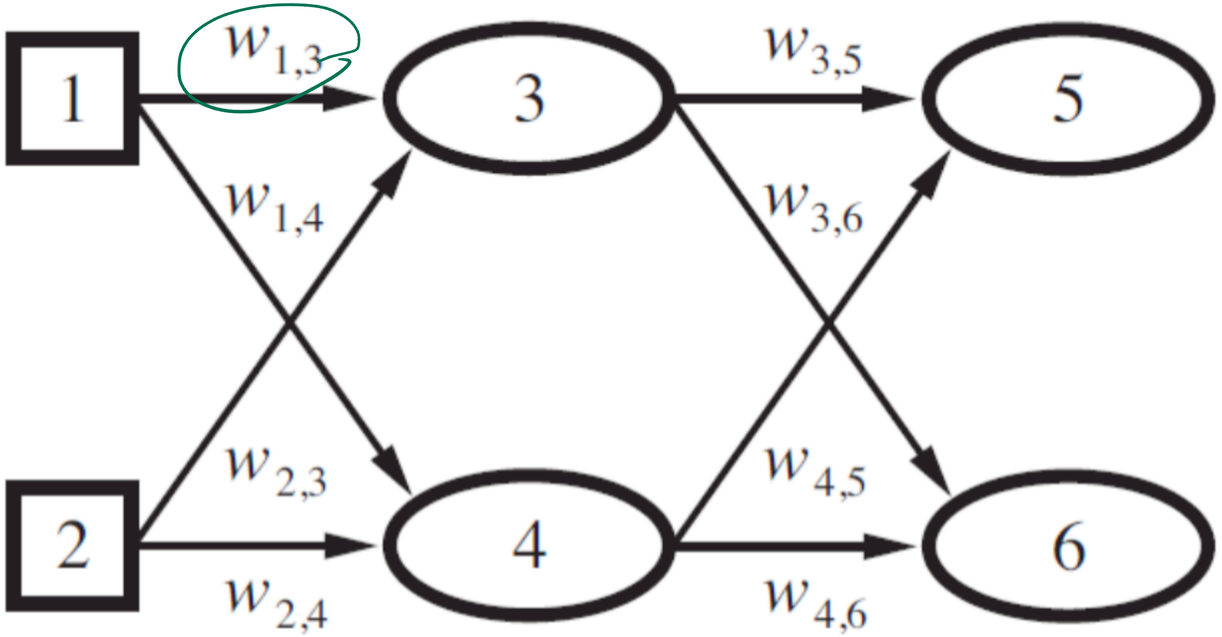


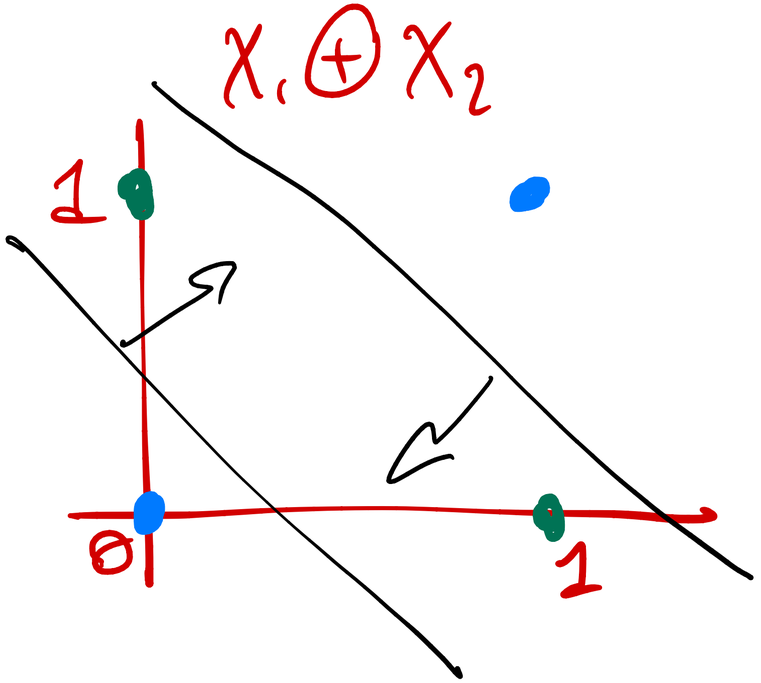




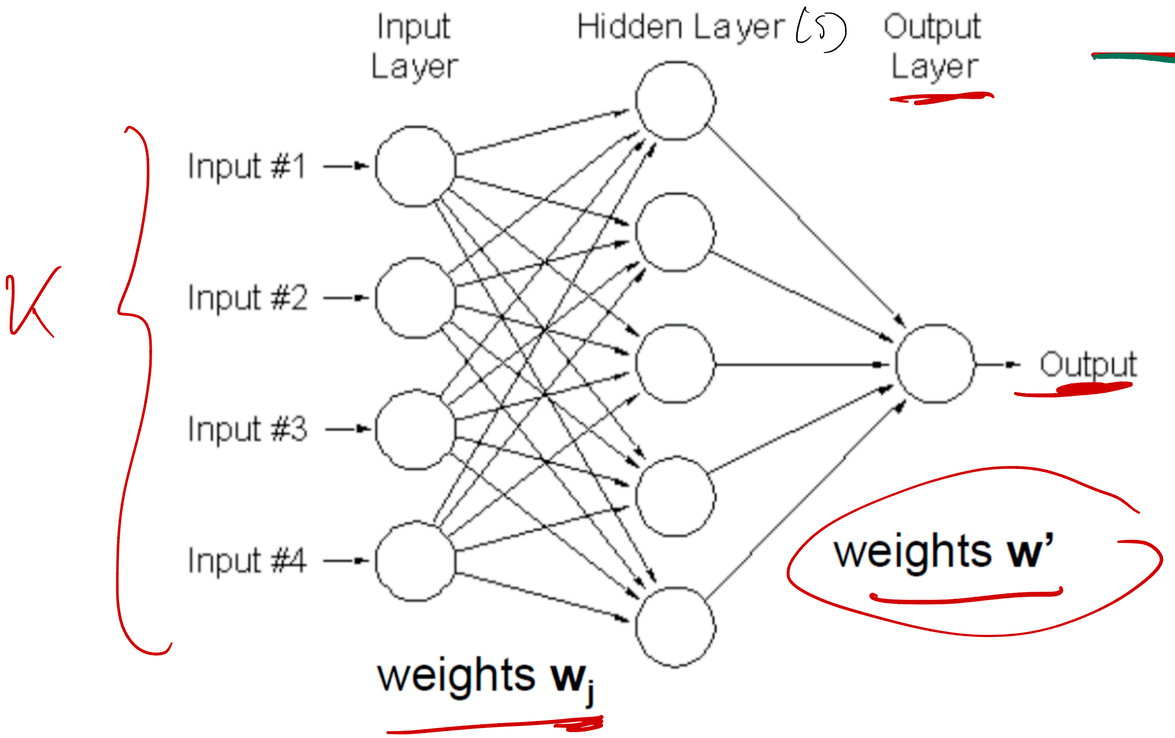


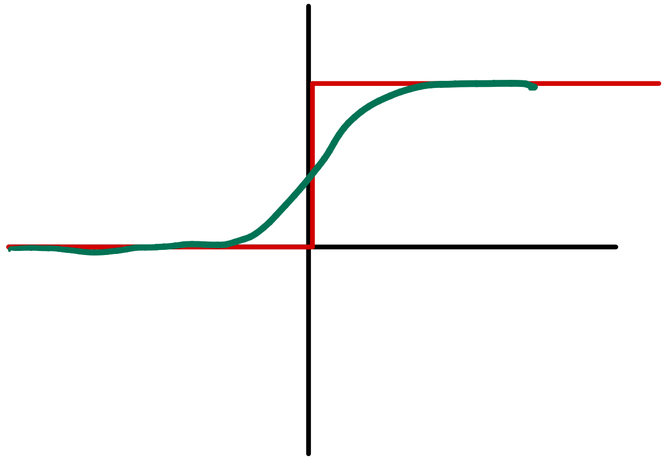
Multi layer





General Structure

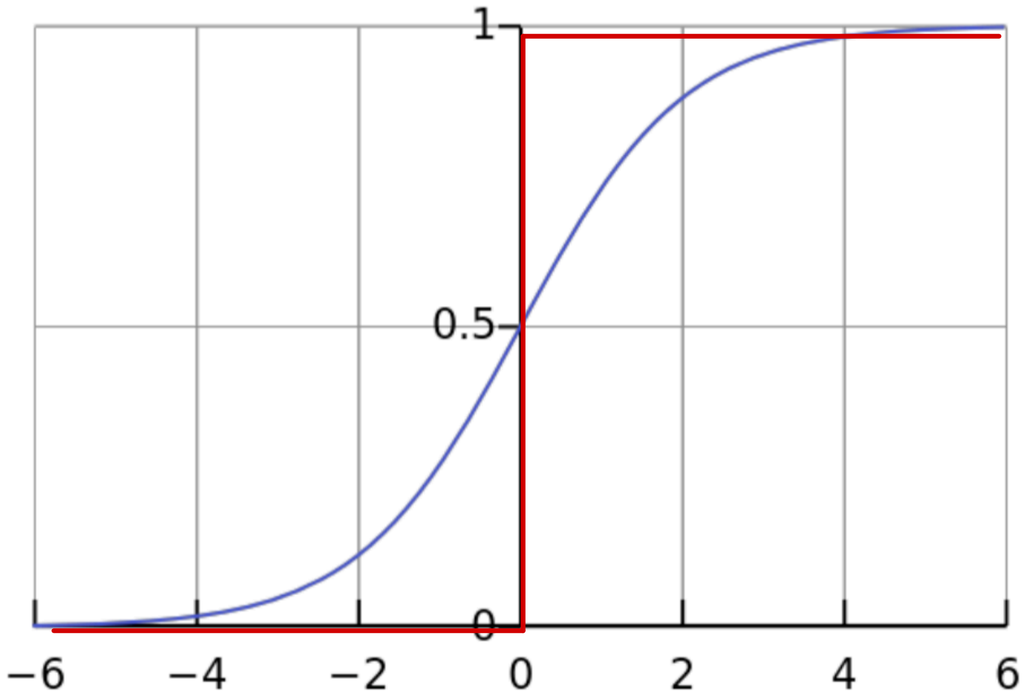




$$h(x) = g\left(\sum_j w_j' \, g\left(\sum_k w_{jk} x_k\right)\right)$$

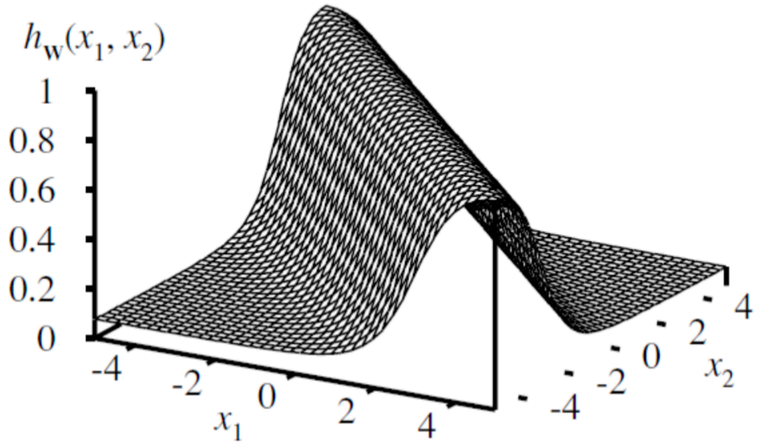
Activation function

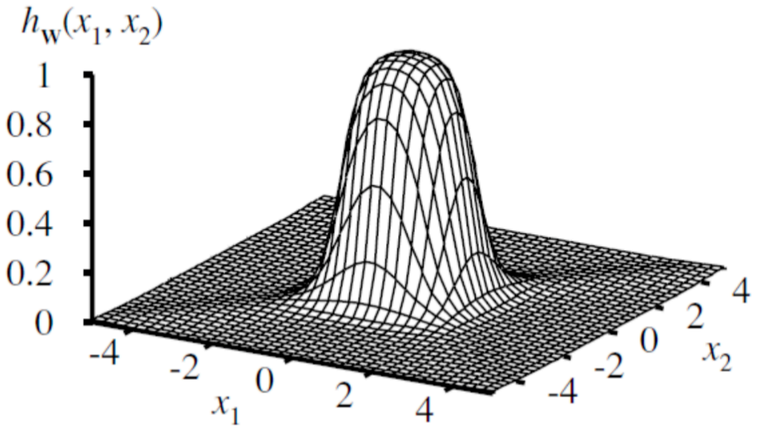
- Sigmoid:
  - $\sigma(x) = \dfrac{1}{1 + e^{-x}}$



Activation function

- Sigmoid:
  - $\sigma(x) = \dfrac{1}{1 + e^{-x}}$





Differentiable Perceptron

- Define the error

$$E(w) = -\frac{1}{2}\sum_j \left(y_j - h_w(x_j)\right)^2, \qquad h_w(x_j) = g(w^T x_j)$$

- Update via gradient descent

$$w \leftarrow w - \alpha \frac{\partial E}{\partial w}$$

$$w \leftarrow w - \alpha \left[y_i - g(w^T x_j)\right]\left[g(w^T x_j)(1 - g(w^T x_j))x_j\right]$$

$$w \leftarrow w + \alpha(y_i - h_w(x_j))x_j$$

Artificial Neural Networks

Comparing

PROS

- Easy to implement (tensorFlow)
- Approxiate Functions
- More layers ⇒ Better Results

CONS

- Depend on X
- Stuck in local optima
- Difficult to Analyze
-

Deep Neural Networks

- end to end training
- Experience needed for Net Structure
- Requires
  - ↑ Data
  - ↑ Computing