Lecture 14a: Perceptron Learning

CSCI 360

Introduction to Artificial Intelligence

USC

Here is where we are…

	3/1		Project 2 Out	
9	3/4 3/6	3/5 3/7	Quantifying Uncertainty Bayesian Networks	[Ch 13.1-13.6] [Ch 14.1-14.2]
10	3/11 3/13	3/12 3/14	*(spring break, no class)* *(spring break, no class)*	
11	3/18 3/20	3/19 3/21	Inference in Bayesian Networks Decision Theory	[Ch 14.3-14.4] [Ch 16.1-16.3 and 16.5]
	3/23		Project 2 Due	
12	3/25 3/27	3/26 3/28	***Advanced topics*** *(Chao traveling to National Science Foundation)* ***Advanced topics*** *(Chao traveling to National Science Foundation)*	
	3/29		Homework 2 Out	
13	4/1 4/3	4/2 4/4	Markov Decision Processes Decision Tree Learning	[Ch 17.1-17.2] [Ch 18.1-18.3]
	4/5 4/5		Homework 2 Due Project 3 Out	
14	4/8 4/10	4/9 4/11	Perceptron Learning Neural Network Learning	[Ch 18.6] [Ch 18.7]
15	4/15 4/17	4/16 4/18	Statistical Learning Reinforcement Learning	[Ch 20.2.1-20.2.2] [Ch 21.1-21.2]
16	4/22 4/24	4/23 4/25	Artificial Intelligence Ethics Wrap-Up and Final Review	
	4/26		Project 3 Due	
	5/3	5/2	**Final Exam** (2pm-4pm)	

Outline

- What is AI?
- Part I: Search
- Part II: Logical reasoning
- Part III: Probabilistic reasoning
- **Part IV: Machine learning**
  - Decision Tree Learning
  - **Perceptron Learning**
  - Neural Network Learning
  - Statistical Learning
  - Reinforcement Learning

Recap: *Forms of learning* *-- Prior knowledge*

- **Inductive** learning
  - Learning a general function, or a general rule, from specific input-output pairs

$$\boxed{\mathcal{D} = \left\{\mathbf{x}(n), y(n)\right\}_{n=1\ldots N} \Rightarrow \left(A \Rightarrow C\right)}$$

- **Deductive** learning
  - Going from a known general rule to a new rule that is logically entailed, but is useful because it allows more efficient processing

$$\boxed{\left(A \Rightarrow B \Rightarrow C\right) \Rightarrow \left(A \Rightarrow C\right)}$$

Recap: *Forms of learning – Feedback to learn from*

- **Unsupervised** learning
  - Learn "patterns in the input" without explicit feedback
- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

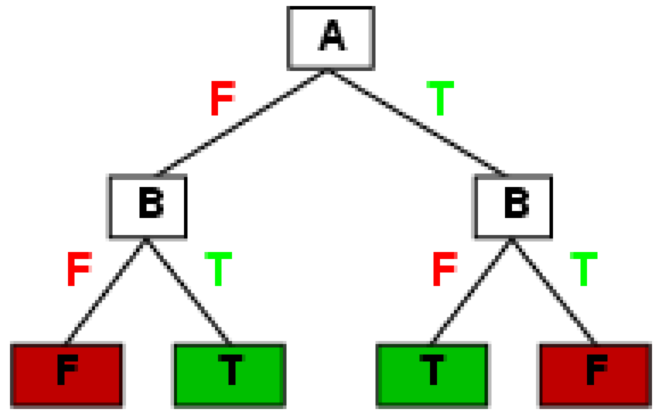
Recap: *Forms of learning – Feedback to learn from*

- **Unsupervised** learning
  - Learn “patterns in the input” without explicit feedback
- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification
- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Recap: *Decision tree -- Expressiveness*

- Decision trees can express any function of the input attributes.
  - e.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- In general, if there is a path to leaf for each example in the training set, it probably **won't generalize well** to new examples

- Prefer to find more compact decision trees

Recap: *Decision tree learning – input/output pairs*

- Examples described by attribute values (Boolean, discrete, continuous)
  - E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	Bar	Fri	Hun	Pat	$Price$	$Rain$	Res	$Type$	Est	$Wait$
X_1											T
X_2											F
X_3											T
X_4											T
X_5											F
X_6											T
X_7											F
X_8											T
X_9											F
X_{10}											F
X_{11}											F
X_{12}											T

- Classification of examples is positive (T) or negative (F)

Recap: *Decision tree learning – input/output pairs*

- Examples described by attribute values (Boolean, discrete, continuous)
  - E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

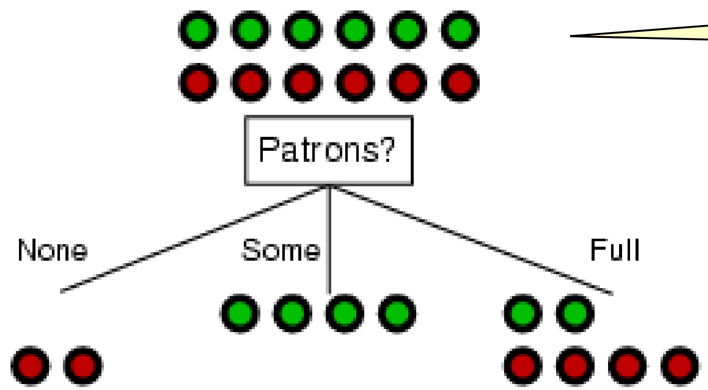
- Classification of examples is positive (T) or negative (F)

Recap: *Decision tree learning – Greedy algorithm*

- Top-down construction of a decision tree by recursively selecting the "**best attribute**" to use at the current node in tree *(with the largest information gain*)
  - Once attribute is **selected** for current node, generate child nodes: one for each possible value of selected attribute
  - **Partition** examples using the possible values of this attribute, and assign these subsets of the examples to the appropriate child node
  - **Repeat** for each child node, until all examples associated with a node are either all positive or all negative

Recap: *Decision tree learning* – *Choosing an attribute*

- **Idea:** a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



Before splitting, there are 6/12 **positive** examples and 6/12 **negative** examples

After splitting, the 1st subgroup has 2/2 **negative** examples

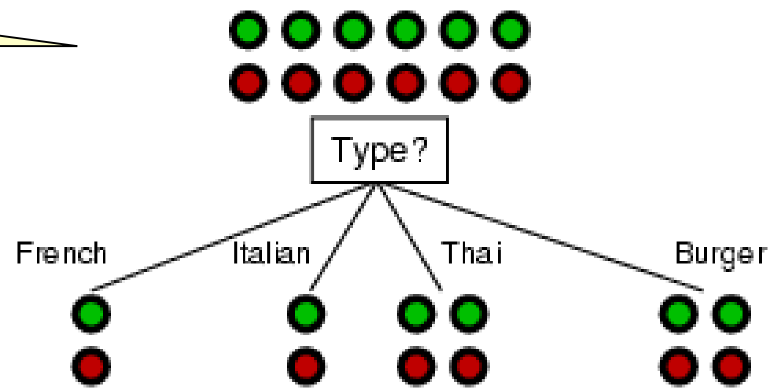
the 2nd subgroup has 4/4 **positive** examples

the 3rd subgroup has 2/6 **positive** examples and 4/6 **negative** examples

Recap: *Decision tree learning – Choosing an attribute*

- **Idea:** a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"

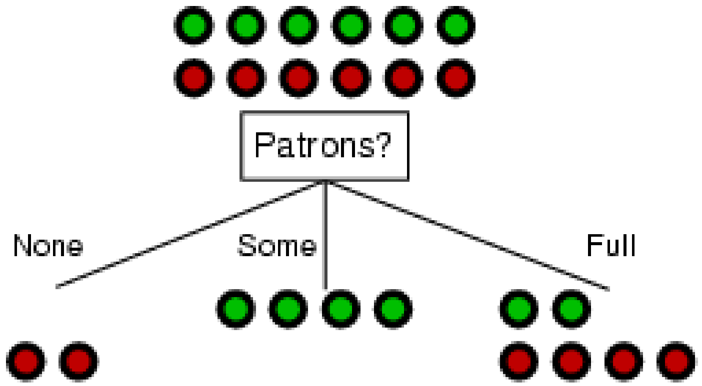
Before splitting, there are 6/12 **positive** examples and 6/12 **negative** examples

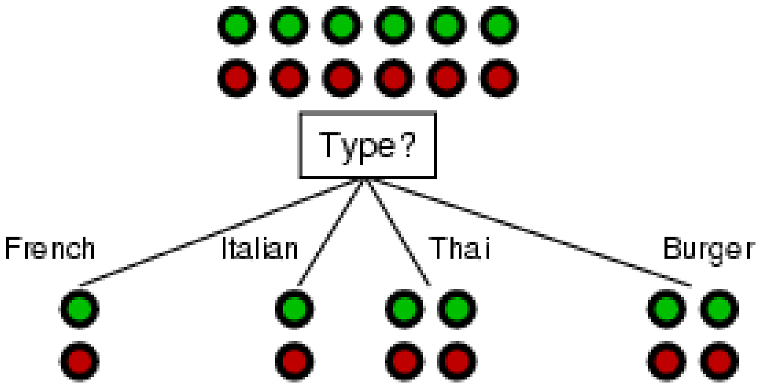


After splitting, the subgroups still have 50% **positive** examples and 50% **negative** examples

Recap: *Decision tree learning – Choosing an attribute*

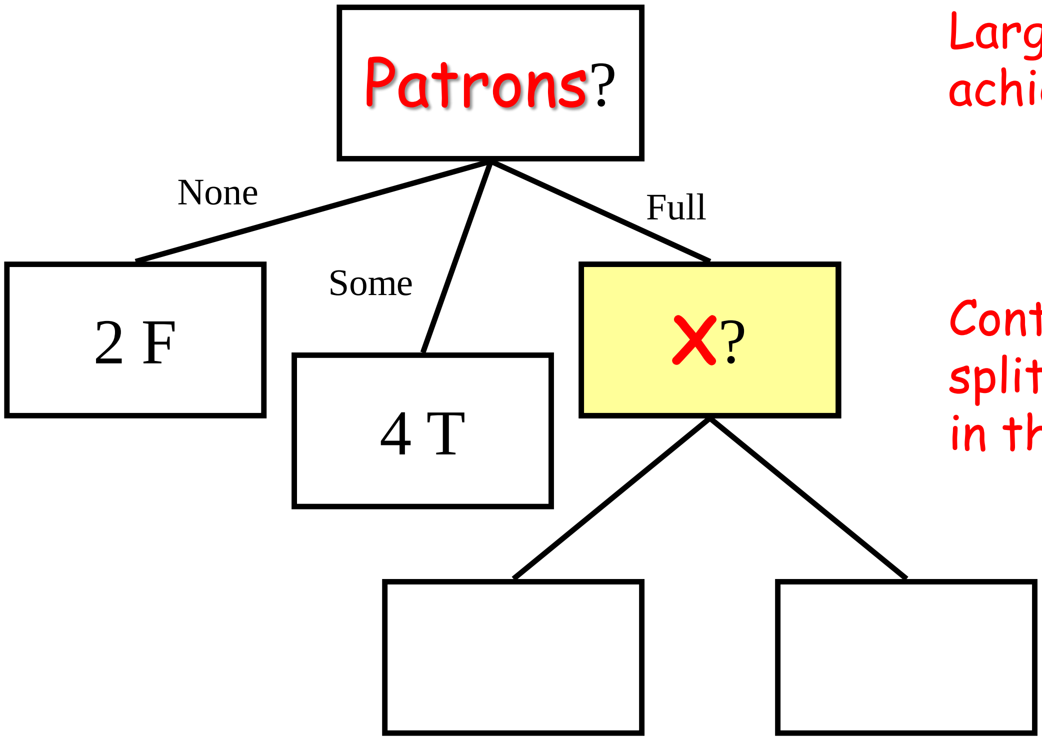
- **Idea:** a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"





- ***Patrons?*** is a better choice

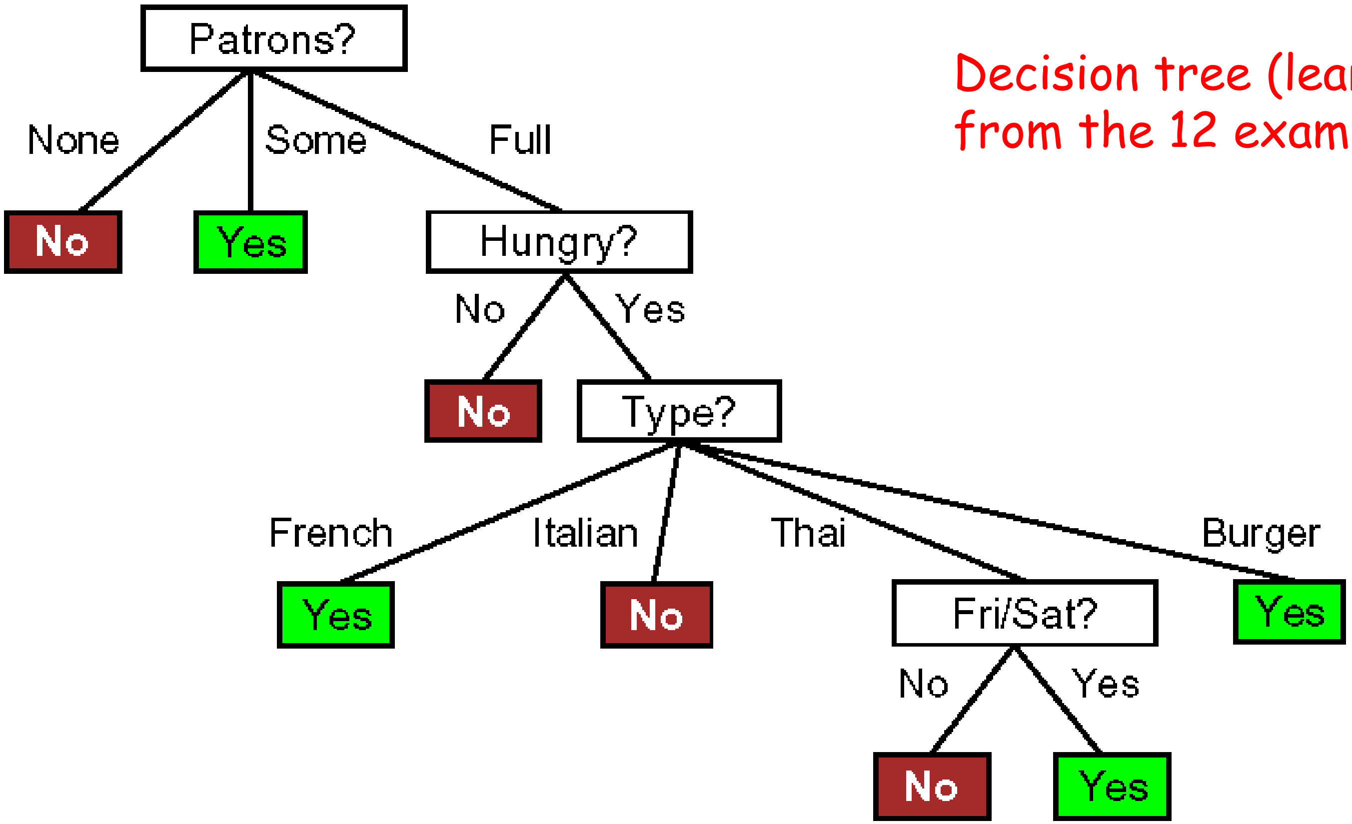
Recap: *Decision tree learning* – *Iterative process*



Largest entropy decrease (0.16) achieved by splitting on Patrons.

Continue like this, making a new split, always trying to purify nodes in the subgroups.

Recap: *Decision tree learning – Final result*



Decision tree (learned from the 12 examples)

Recap: *Forms of learning – Feedback to learn from*

- **Unsupervised** learning
  - Learn "patterns in the input" without explicit feedback

- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification

- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Recap: *Forms of learning – Feedback to learn from*

- **Unsupervised** learning
  - Learn “patterns in the input” without explicit feedback

- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification

- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Regression with linear models

- Used for **hundred of years**: linear functions of continuous-valued inputs

$$h_{\mathbf{w}}(x) = w_1\, x + w_0$$

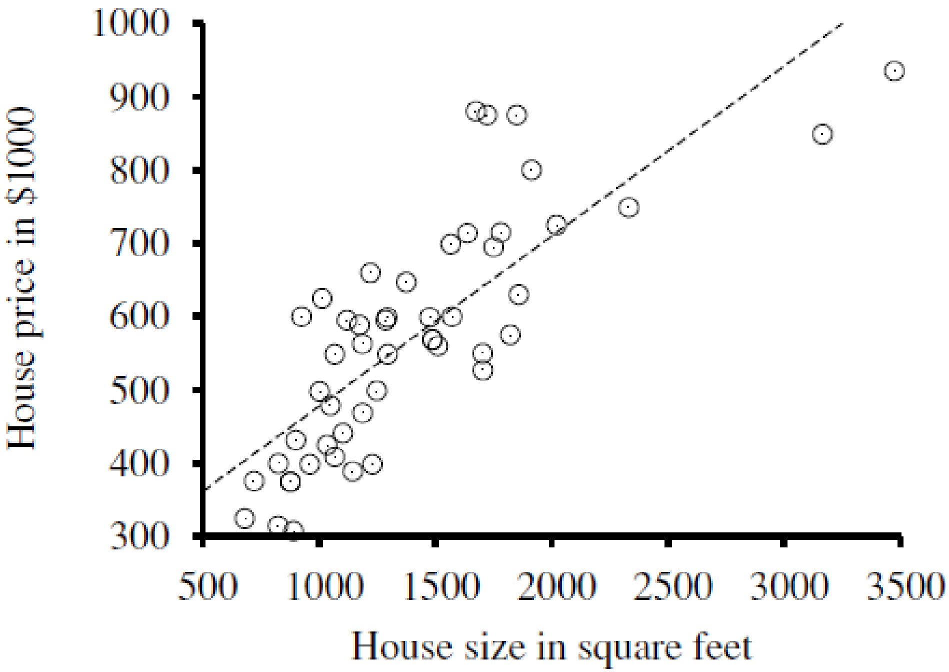
weight $\mathbf{w} = [\, w_0 , w_1 \,]$

Linear regression: finding $h_w(x)$ that best fits training data $\{ (x, f(x)) \}$

Linear regression

- Finding $h_w(x)$ that best fits the training data { (x, $f(x)$) }

$$\boxed{h_{\mathbf{w}}(x) = w_1\, x + w_0}$$



Find the values of $[w_0,w_1]$ that minimize empirical loss

$$\boxed{\mathbf{w}^* = \mathrm{argmin}_{\mathbf{w}}\, Loss(h_{\mathbf{w}}).}$$

Question: How to define the "<u>empirical loss</u>"?

Square loss function (L_2)

- **Gauss**: If the *f(x)* values have normally distributed noise, the most likely values of w_1 and w_0 can be obtained by minimizing the **sum of the squares of the errors**

$$\boxed{Loss(h_{\mathbf{w}})} = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j))$$

$$= \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2$$

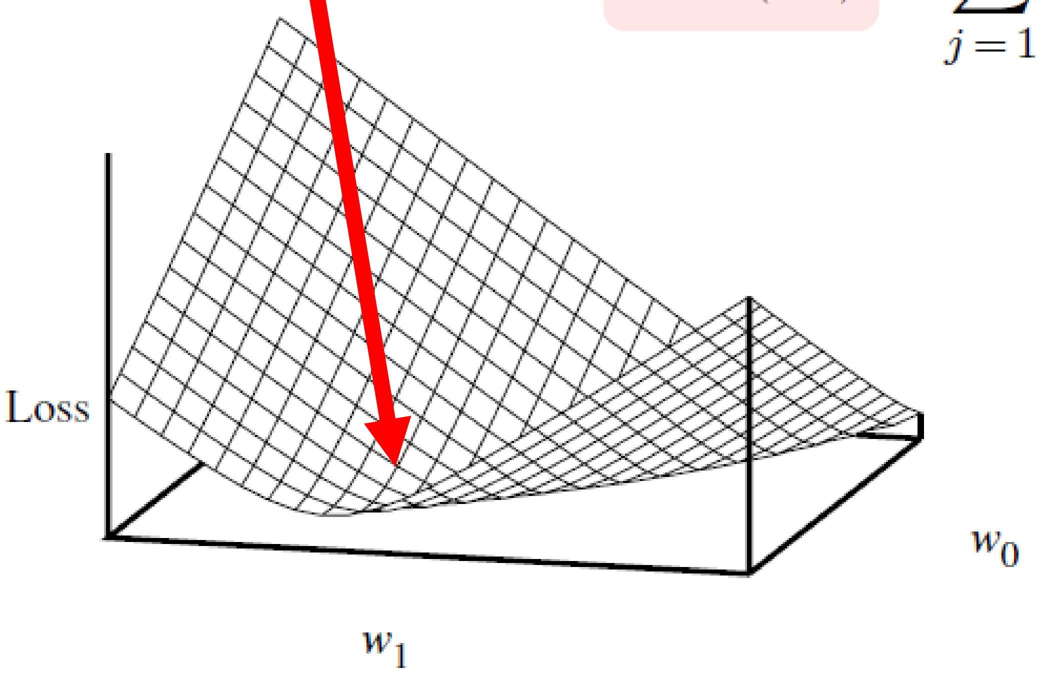
$$= \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

Portrait of Gauss published in *Astronomische Nachrichten* (1828)

Square loss function (L_2)

- **Gauss**: If the *f(x)* values have normally distributed noise, the most likely values of w_1 and w_0 can be obtained by minimizing the **sum of the squares of the errors**

$$\boxed{Loss(h_{\mathbf{w}})} = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$



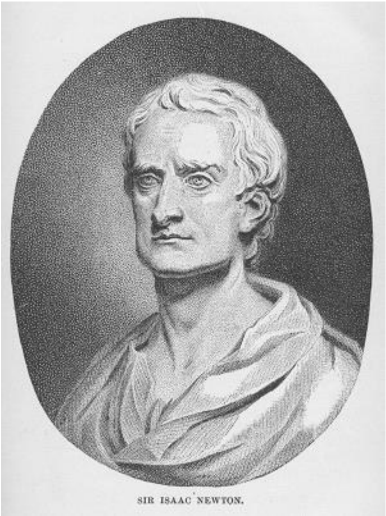
Minimizing L_2

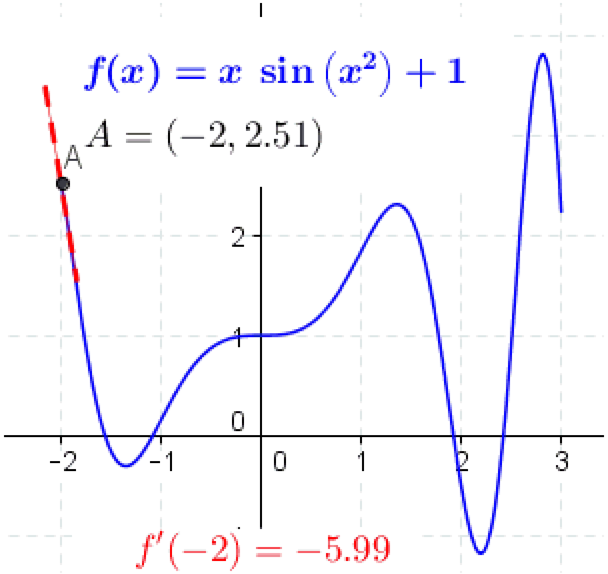
- Two options:
  - **Hand calculation** (closed-form solution)
  - **Hill-climbing search** (gradient descent)

Recap: *Derivative*

- The **derivative** of a function f(x) measures the **sensitivity to change** of the output w.r.t. a change in its input value

- For a univariate function, it's the slope of the tangent line





The derivative at different points of a differentiable function
https://en.wikipedia.org/wiki/Derivative

Recap: *Derivative – Rules for basic functions*

- *Derivatives of powers*: if

$$f(x) = x^r,$$

where r is any real number, then

$$f'(x) = rx^{r-1},$$

- *Exponential and logarithmic functions*:

$$\frac{d}{dx}e^x = e^x.$$

$$\frac{d}{dx}a^x = a^x \ln(a).$$

- *Sum rule*:

$$(\alpha f + \beta g)' = \alpha f' + \beta g'$$

- *Trigonometric functions*:

$$\frac{d}{dx}\sin(x) = \cos(x).$$

$$\frac{d}{dx}\cos(x) = -\sin(x).$$

- *Product rule*:

$$(fg)' = f'g + fg'$$

- *Chain rule* for composite functions: If $f(x) = h(g(x))$, then

$$f'(x) = h'(g(x)) \cdot g'(x).$$

Minimizing L_2 – *closed-form solution*

- The sum of the squares of the errors (L2) is minimized when its partial derivatives w.r.t. w_0 and w_1 are zero.

$$\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0$$

$$\frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0$$

?

?

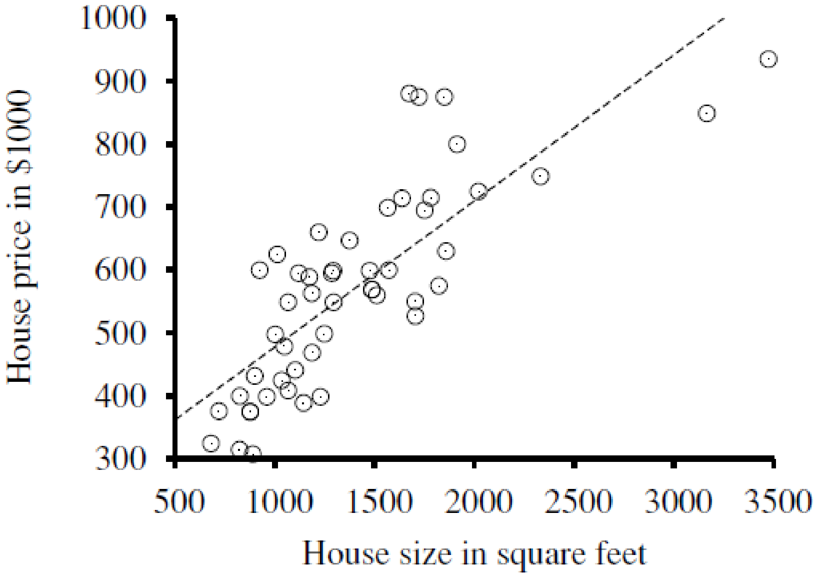
$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$$

$$w_0 = (\sum y_j - w_1(\sum x_j))/N$$

Minimizing L_2 – *closed-form solution*

- The sum of the squares of the errors (L2) is minimized when its partial derivatives w.r.t. w_0 and w_1 are zero.

$$\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$



$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$$

$$w_0 = (\sum y_j - w_1(\sum x_j))/N$$

$$\boxed{w_1 = 0.232,\ w_0 = 246}$$

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0))
\end{aligned}$$

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2$$

$$\frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2$$

Recap: *Computing the partial derivative*

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0))
\end{aligned}$$

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

$$\frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

Minimizing L_2

- Two options:
  - **Hand calculation** (closed-form solution)
  - **Hill-climbing search** (gradient descent)

Minimizing L_2 – *gradient descent*

- General search technique: a **hill-climbing** algorithm that follows the **gradient** of the function to be optimized
  - **Initialization**: choose any starting point (w_0, w_1)
  - **Iteration**: move to a neighboring point that is downhill

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

(α) is **step size** (or **learning rate**)

$$w_0 \leftarrow w_0 + \alpha \left(y - h_{\mathbf{w}}(x)\right);$$

$$w_1 \leftarrow w_1 + \alpha \left(y - h_{\mathbf{w}}(x)\right) \times x$$

Minimizing L_2 – *gradient descent*

- General search technique: a **hill-climbing** algorithm that follows the **gradient** of the function to be optimized
  - **Initialization**: choose any starting point (w_0, w_1)
  - **Iteration**: move to a neighboring point that is downhill

For N training examples, the **batch gradient descent** converges to the unique global minimum, but is very slow

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

Stochastic gradient descent, which considers only a single training point at a time, is often faster, but doesn't guarantee convergence.

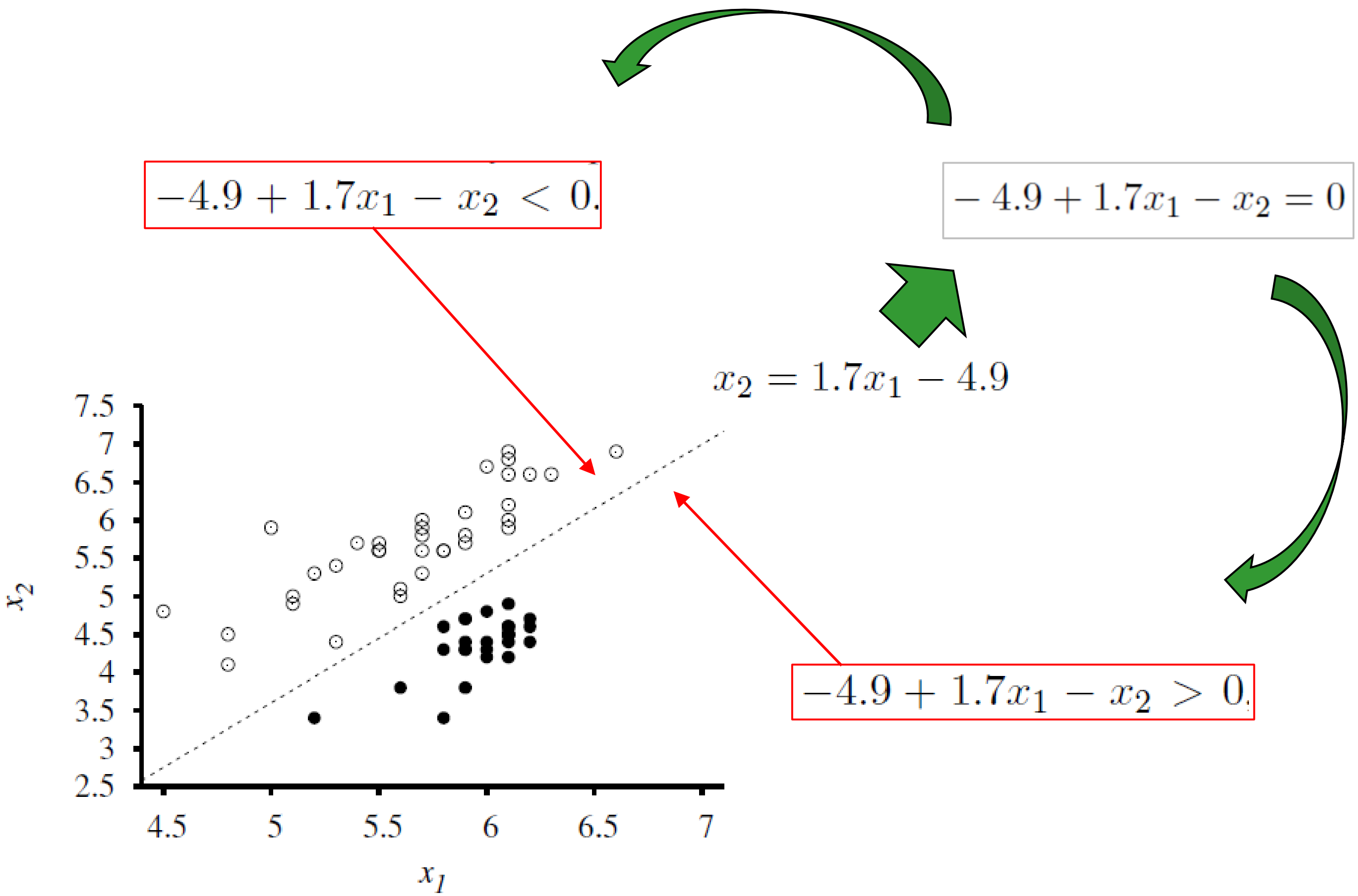
$$\boxed{w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \,; \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j \,.}$$

Recap: *Forms of learning* *– Feedback to learn from*

- **Unsupervised** learning
  - Learn “patterns in the input” without explicit feedback
- **Supervised** learning
  - Given example input-output pairs, learn an input-output function
  - Decision Tree / Regression / Classification
- **Reinforcement** learning
  - Learn from reinforcements (rewards or punishments)

Linear classifiers with a hard threshold

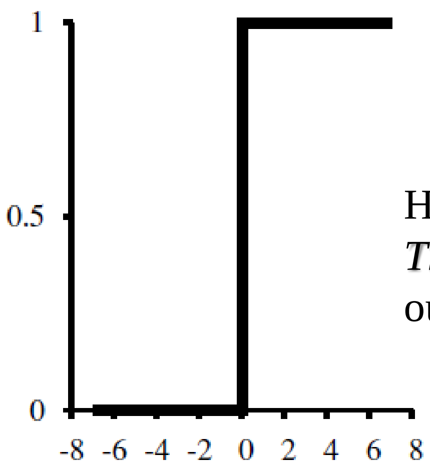
- A **decision boundary** is a line (or surface) that separates the two classes.



Classifier $h_w(x)$

- The classifier is meant to return either **1 (true)** or **0 (false)**
  - Think of it as the result of passing the linear function (w*x) through a threshold function

$$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})}$$ where $Threshold(z) = 1$ if $z \geq 0$ and 0 otherwise.

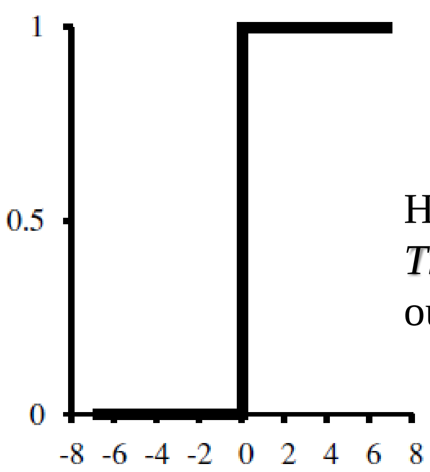


Hard threshold function *Threshold(z)* with 0/1 output

Classifier $h_w(x)$

- The classifier is meant to return either **1 (true)** or **0 (false)**
  - Think of it as the result of passing the linear function (w*x) through a threshold function

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})$$ where $Threshold(z) = 1$ if $z \geq 0$ and 0 otherwise.



Hard threshold function *Threshold(z)* with 0/1 output

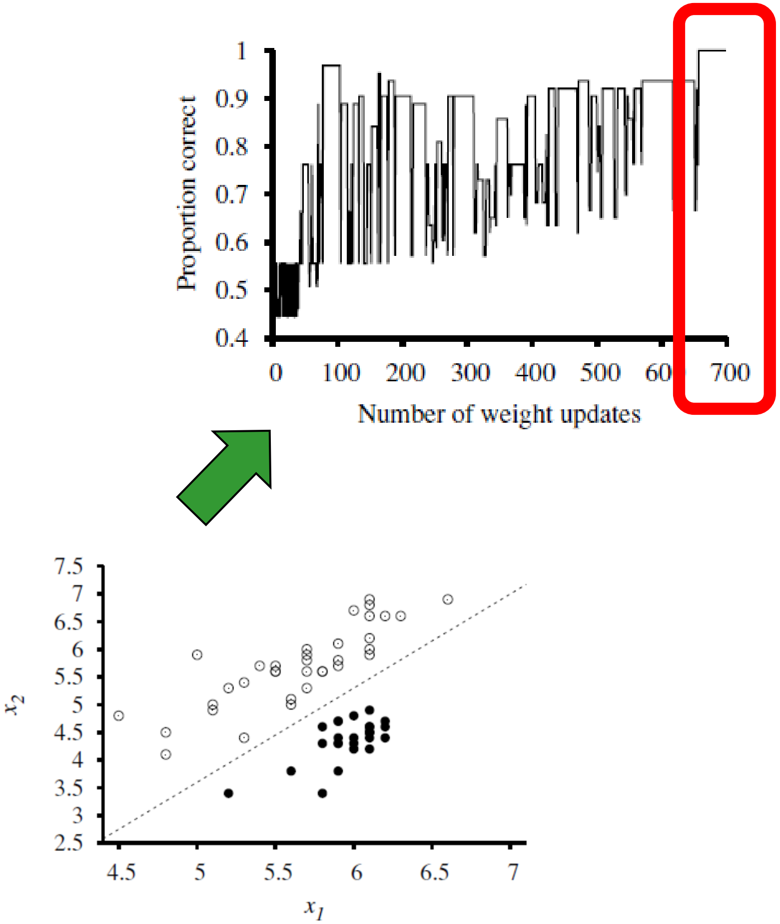
Perceptron learning rule:

$$w_i \leftarrow w_i + \alpha\, (y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

- 0 if the output is correct: no update
- +1 if y is 1 but $h_w(x)$ is 0: increase w_i
- -1 if y is 0 but $h_w(x)$ is 1: decrease w_i

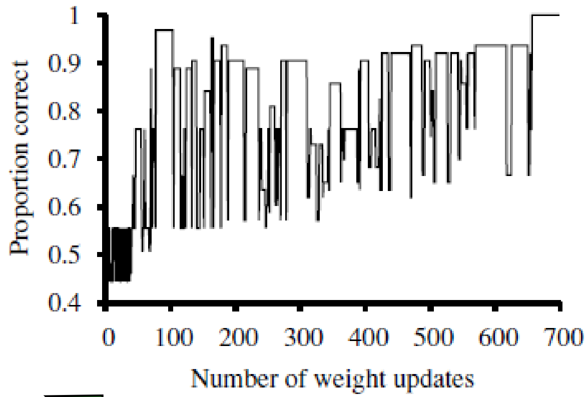
Perceptron learning rule

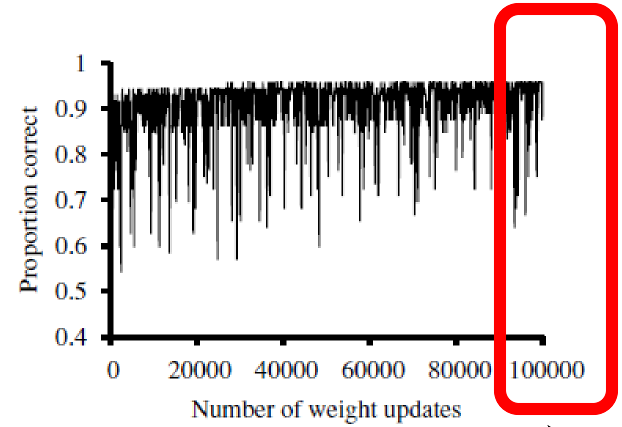
- **Theorem:** Perceptron learning rule converges to a perfect linear separator when data points are linearly separable.

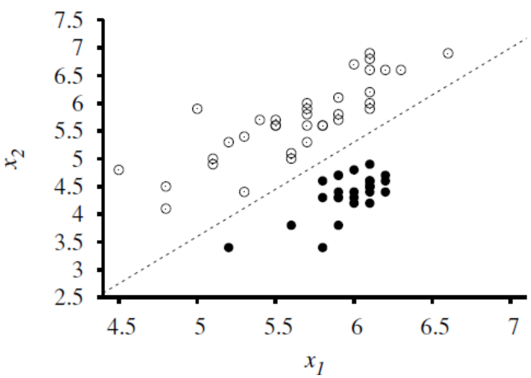


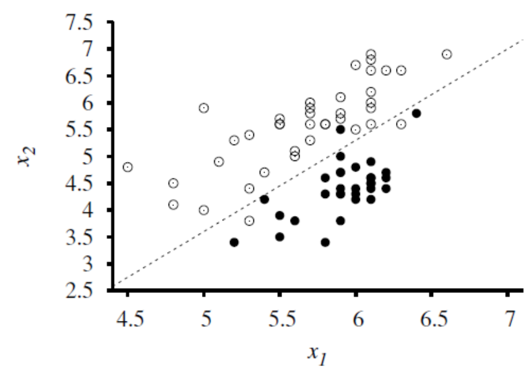
Perceptron learning rule

- **Theorem:** Perceptron learning rule converges to a perfect linear separator when data points are linearly separable.









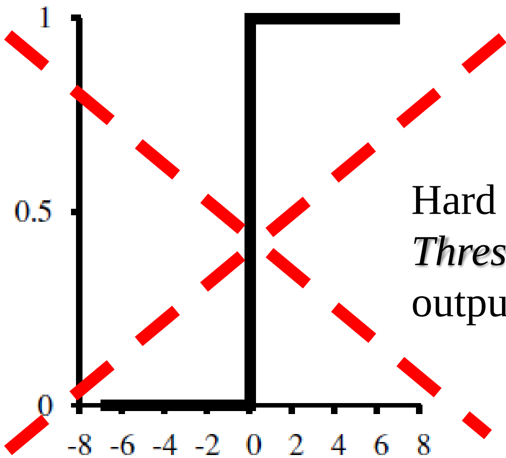
When it's NOT linearly separable, the rule may not coverage

Logistic function as the threshold

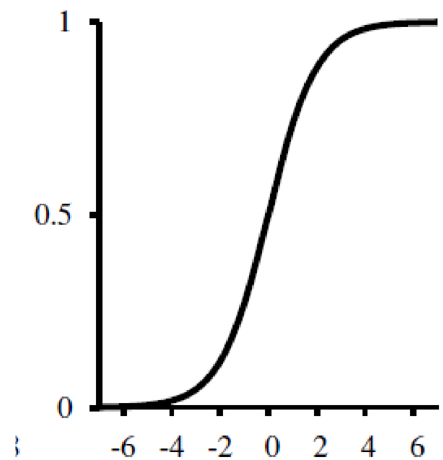
- The classifier is meant to return either **1 (true)** or **0 (false)**
  - Think of it as the result of passing the linear function (w*x) through a threshold function

~~$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})}$ where $Threshold(z) = 1$ if $z \geq 0$~~

$$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})} = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}} \, .$$



Hard threshold function *Threshold(z)* with 0/1 output



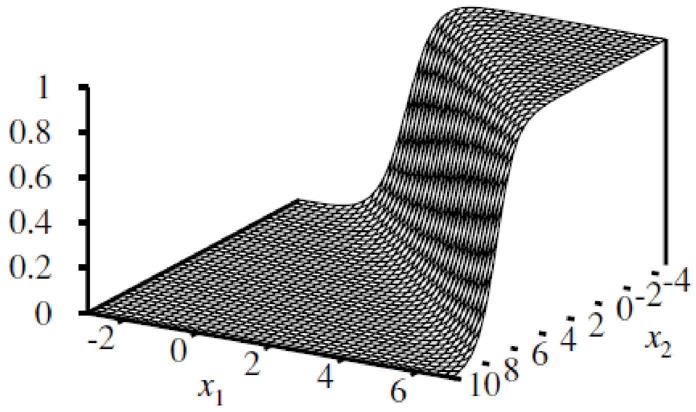
Logistic regression

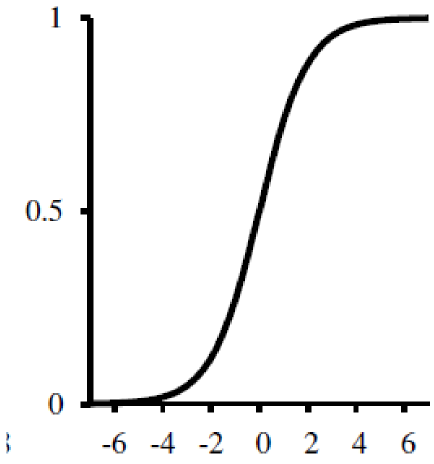
- Derivation of the gradient

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))\end{aligned}$$

$$\boxed{w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i}$$

$$\boxed{h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})} = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}\,.$$



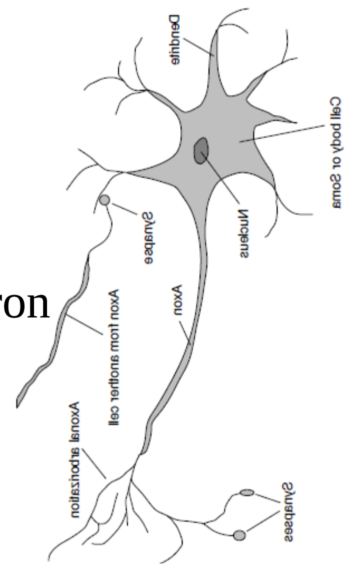


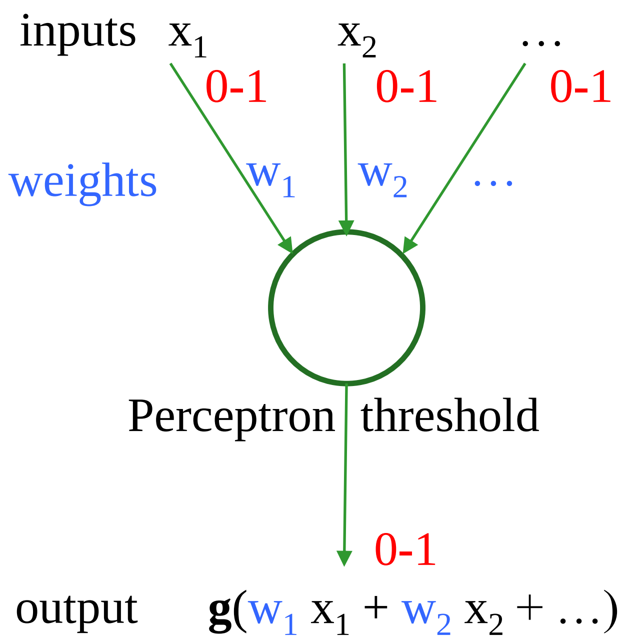
Outline of today's lecture

- Linear regression
- Linear classification
- Logistic regression versus linear classification
  - **Examples**

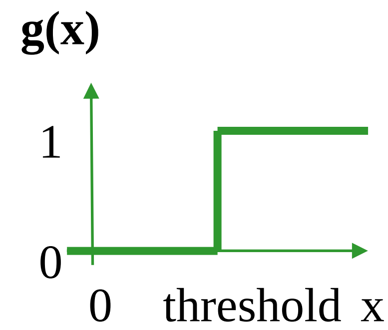
Example: Perceptron Learning

Neuron





threshold
= activation function



- **Objective**: Learn the weights for a given perceptron.
  - For ease of presentation:
    - binary (feature and class) values only (*0*=*false*, *1*=*true*).

Inductive Learning for Classification

- Training examples

Feature_1	Feature_2	Class
true	true	true
true	false	false
false	true	false

Learn ***f (Feature_1, Feature_2) = Class*** from

f(true, true) = true

f(true, false) = false

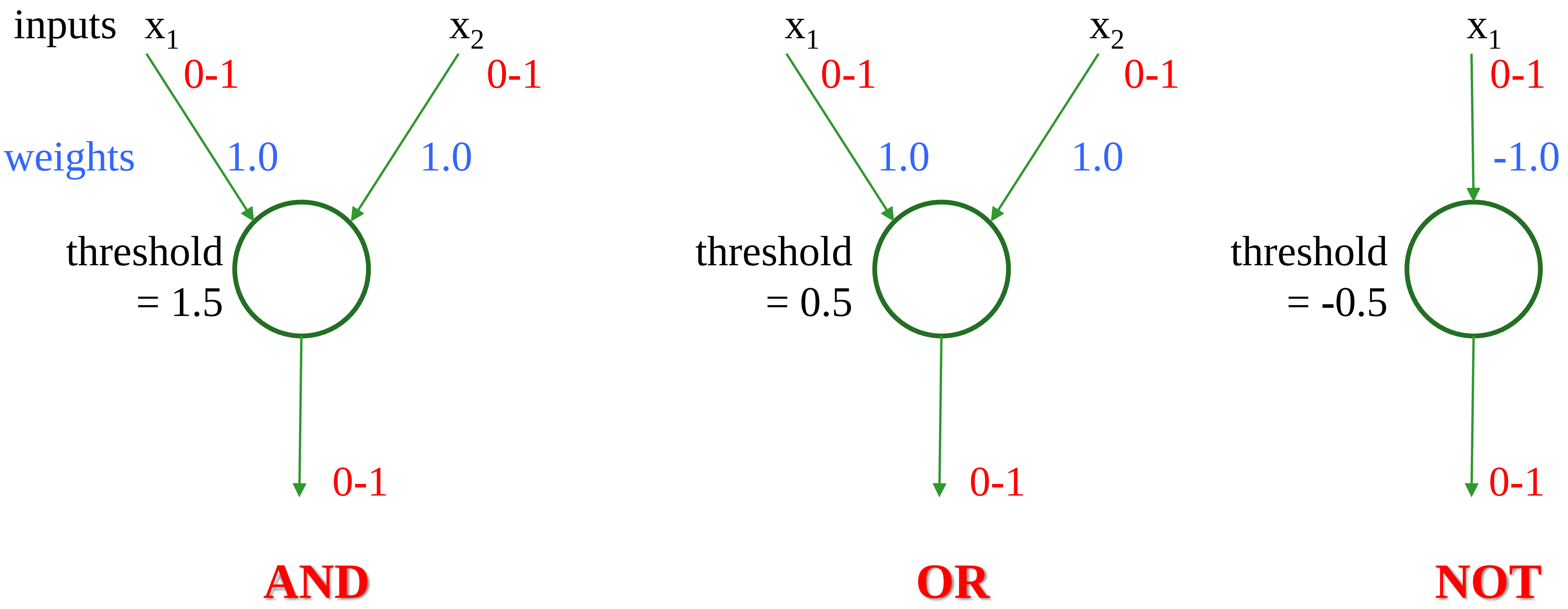
f(false, true) = false

It must be ***consistent*** with all training examples, and make the ***fewest mistakes*** on the test examples.

- Test examples

Feature_1	Feature_2	Class
false	false	?

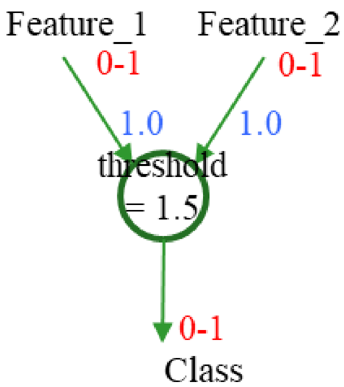
Example: Perceptron Learning



Example: Perceptron Learning

- Labeled examples

Feature_1	Feature_2	Class
true	true	true
true	false	false
false	true	false



- Unlabeled examples (note: classification is very fast)

Feature_1	Feature_2	Class
false	false	? (guess: false)

Example: Perceptron Learning

- Can **perceptrons** represent all Boolean functions?

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

Example: Perceptron Learning

- Can **perceptrons** represent all Boolean functions?

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

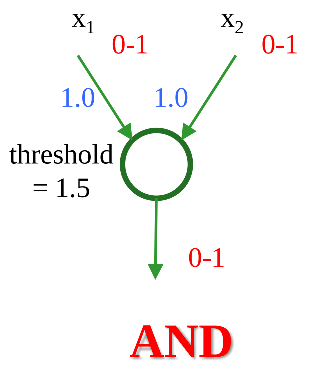
- Linearly separable
  - Need to find an **n-dimensional plane** that separates the labeled examples (true from false)
  - This **plane** determines the weights and the threshold of the perceptron

Example: Perceptron Learning

- Can **perceptrons** represent all Boolean functions?

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- Linearly separable
  - $w_1\, x_1 + w_2\, x_2 = \text{threshold}$
  - $w_1\, x_1 = \text{threshold} - w_2\, x_2$
  - $x_1 = (\text{threshold} / w_1) – (w_2 / w_1)\, x_2 = (1.5 / 1) – (1 / 1)\, x_2 = 1.5 – x_2$

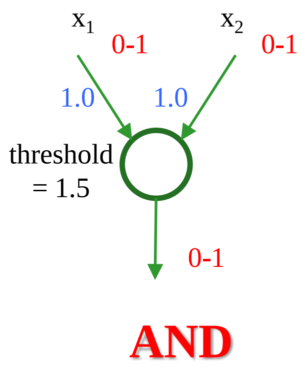


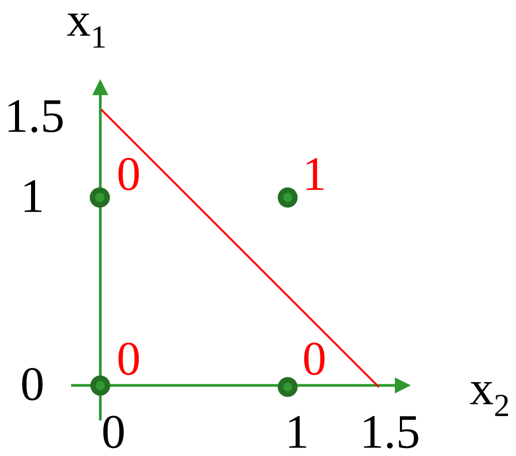
Example: Perceptron Learning

- Can **perceptrons** represent all Boolean functions?

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- Linearly separable
  - $w_1 x_1 + w_2 x_2 = \text{threshold}$
  - $w_1 x_1 = \text{threshold} - w_2 x_2$
  - $x_1 = (\text{threshold} / w_1) - (w_2 / w_1) x_2 = (1.5 / 1) - (1 / 1) x_2 = 1.5 - x_2$





Example: Perceptron Learning

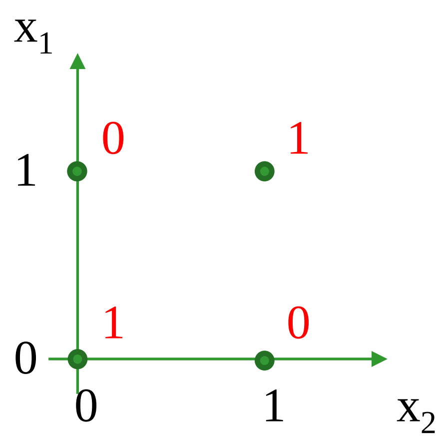
- Can **perceptrons** represent all Boolean functions?

  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- Linearly separable
  - $w_1 x_1 + w_2 x_2 = \text{threshold}$
  - $w_1 x_1 = \text{threshold} - w_2 x_2$
  - $x_1 = (\text{threshold} / w_1) - (w_2 / w_1)\, x_2 = (1.5 / 1) - (1 / 1)\, x_2 = 1.5 - x_2$

?

XOR



Example: Perceptron Learning

- Can **perceptrons** represent all Boolean functions? **NO**

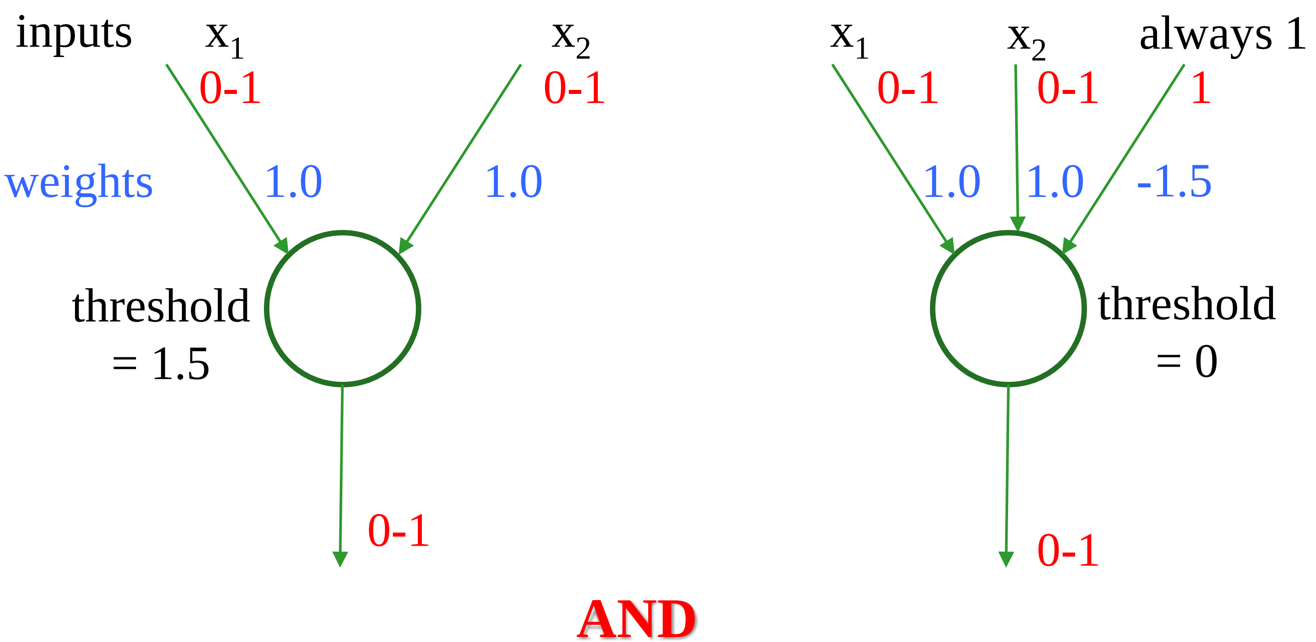
  f (Feature_1, …, Feature_n) ≡ some propositional sentence

- An XOR **cannot** be represented with a single perceptron!

- It does not mean single perceptrons should not be used
  - They will make some mistakes for some Boolean functions but they often work well, that is, make few mistakes on the training and test examples.
  - Of course, you only want to use them if they do not make too many mistakes in the test examples.

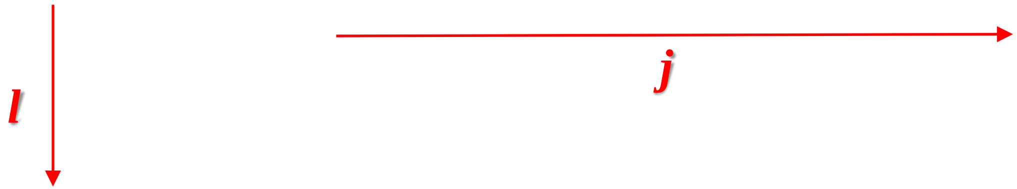
Example: Perceptron Learning

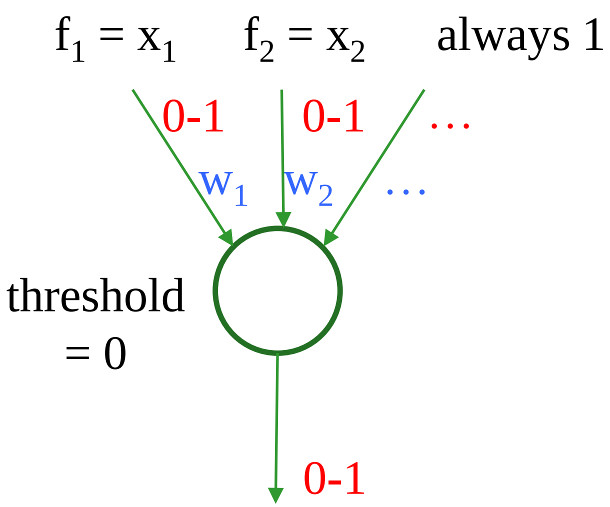
- The threshold can be expressed as a weight too
  - This way, an algorithm only needs to learn weights instead of the threshold and the weights. (*The new threshold is always zero.)*



Example: Perceptron Learning

	Feature f_1	Feature f_2	…	Class
E(xample) 1: l=1	f_{11}	f_{12}	…	c_1
E(xample) 2: l=2	f_{21}	f_{22}	…	c_2
E(xample) 3: l=3	f_{31}	f_{32}	…	c_3
…	…	…	…	…

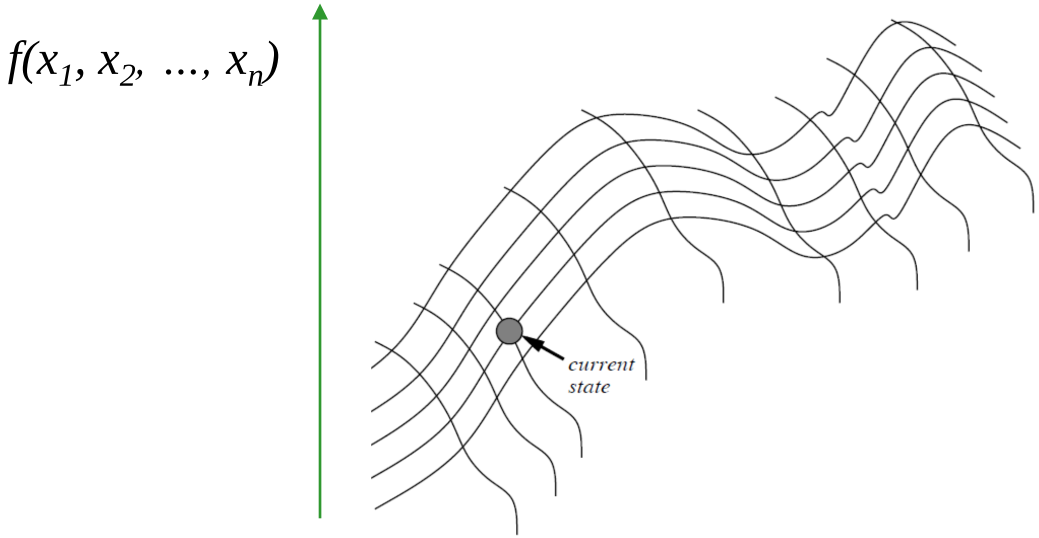




- Learn the weights w_1, w_2, … so that the resulting perceptron is consistent with all training examples

Gradient Descent

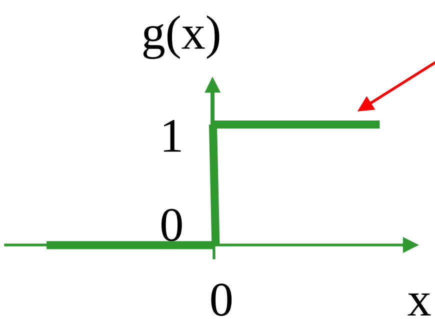
- Finding a local minimum of a differentiable function $f(x_1, x_2, \ldots, x_n)$ with gradient descent



Example: Perceptron Learning

- Learn weights $w_1, w_2, \ldots$ with gradient descent (for a small positive learning rate α) so that perceptron is consistent with all training examples

Threshold function

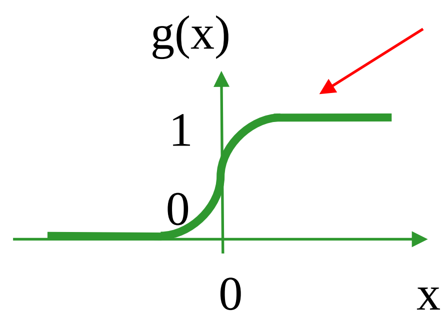


Slope (= 0) does not give gradient descent an indication whether to increase or decrease x to find a local minimum

the output is either 0 or 1

no: not differentiable at x=0

Sigmoid function



Slope (> 0) gives gradient descent an indication to decrease x to find a local minimum

the output is any real value in the range (0,1)

$$g(x) = 1 / (1 + e^{-x})$$
$$g'(x) = e^{-x} / (1 + e^{-x})^2 = g(x)\,(1 - g(x))$$

Recap: *Derivative* *– quotient rule*

Let $f(x) = g(x)/h(x)$,

$$\boxed{f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{[h(x)]^2}.}$$

- $g(x) = 1 / (1 + e^{-x})$
- $g'(x) = e^{-x} / (1 + e^{-x})^2$

  $= g(x)\ (1 – g(x))$

Example: Perceptron Learning

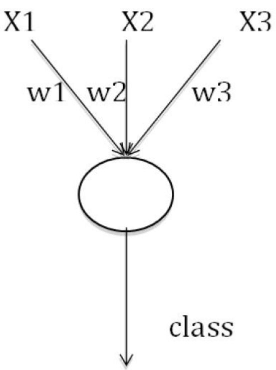
- **Example**: Learn weights w_1, w_2, … with an approximation of gradient descent (for $\alpha = 0.01$) so that the resulting perceptron is consistent with an AND

	Feature f_1	Feature f_2	Feature f_3	Class
E(xample) 1: l=1	0	0	1	0
E(xample) 2: l=2	0	1	1	0
E(xample) 3: l=3	1	0	1	0
E(xample) 4: l=4	1	1	1	1

Implementation

```
#include<stdio.h>
#include<math.h>

#define g(x) ((1.0/(1.0+exp(-x))))
#define gprime(x) ((g(x) * (1-g(x))))

main()
{
  float alpha = 0.01;
  int trainingexamples = 4;
  int features = 3;
  float f[4][3] = {{0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1}};
  float class[4] = {0,0,0,1};
  float w[3] = {1.1, -2.1, 0.3}; /* random values */

  int l, j, epoch;
  float weightedsum;

  for (epoch = 0; 1; ++epoch)
  {
    for (l = 0; l < trainingexamples; ++l)
    {
      weightedsum = 0.0;
      for (j = 0; j < features; ++j)
        weightedsum += w[j]*f[l][j];
      for (j = 0; j < features; ++j)
        w[j] -= alpha*(g(weightedsum) - class[l])*gprime(weightedsum)*f[l][j];
    }
    printf("epoch = %d, weights = ", epoch);
    for(j = 0; j < features; ++j)
      printf("%.2f", w[j]);
    printf(", outputs =");
    for(l = 0; l < trainingexamples; ++l)
    {
      weightedsum = 0.0;
      for(j = 0; j < features; ++j)
        weightedsum += w[j]*f[l][j];
      printf(" %.2f", g(weightedsum));
    }
    printf("\n");
  }
}
```

	f_1	f_2	f_3	class
l=1	0	0	1	0
l=2	0	1	1	0
l=3	1	0	1	0
l=4	1	1	1	1




```
> gcc -lm learning.c
> ./a.out
epoch = 0, weights = 1.10 -2.10 0.30, outputs = 0.57 0.14 0.80 0.33
epoch = 1, weights = 1.10 -2.10 0.30, outputs = 0.57 0.14 0.80 0.33
epoch = 2, weights = 1.10 -2.10 0.30, outputs = 0.57 0.14 0.80 0.33
epoch = 3, weights = 1.10 -2.09 0.29, outputs = 0.57 0.14 0.80 0.33
epoch = 4, weights = 1.10 -2.09 0.29, outputs = 0.57 0.14 0.80 0.33
epoch = 5, weights = 1.10 -2.09 0.29, outputs = 0.57 0.14 0.80 0.33
...
epoch = 100, weights = 1.12 -1.97 0.16, outputs = 0.54 0.14 0.78 0.33
...
epoch = 1000, weights = 1.15 -0.80 -0.84, outputs = 0.30 0.16 0.58 0.38
...
epoch = 10000, weights = 2.56 2.55 -3.96, outputs = 0.02 0.20 0.20 0.76
...
epoch = 100000, weights = 5.47 5.47 -8.30, outputs = 0.00 0.06 0.06 0.93
```

Example: Perceptron Learning

- **Example**: Learn weights w_1, w_2, … with an approximation of gradient descent (for $\alpha = 0.01$) so that the resulting perceptron is consistent with an AND

	Feature f_1	Feature f_2	Feature f_3	Class
E(xample) 1: l=1	0	0	1	0
E(xample) 2: l=2	0	1	1	0
E(xample) 3: l=3	1	0	1	0
E(xample) 4: l=4	1	1	1	1

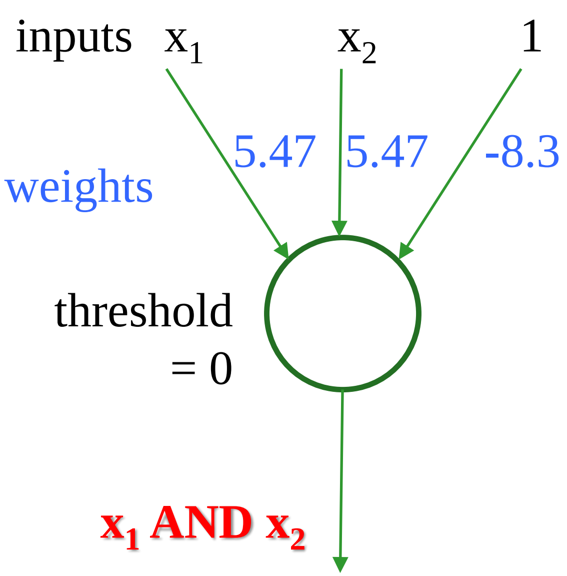
Since the output is now any real value in the range (0,1), we consider a value less than 0.5 to be 0 and a value greater than 0.5 to be 1.
So we indeed learned an AND!

Epoch 0		Epoch 1		Epoch 2		Epoch 100		Epoch 100,000	
Weights	Outputs	Weights	Outputs	Weights	Outputs	Weights	Outputs	Weights	**Outputs**
$w_1 = 1.10$	$o_1 = 0.57$	1.10	0.57	1.10	0.57	1.12	0.54	5.47	0.00
$w_2 = -2.10$	$o_2 = 0.14$	-2.10	0.14	-2.10	0.14	-1.97	0.14	5.47	0.06
$w_3 = 0.30$	$o_3 = 0.80$	0.30	0.80	0.30	0.80	0.16	0.78	-8.30	0.06
	$o_4 = 0.33$		0.33		0.33		0.33		0.93

Example: Perceptron Learning

- **Example**: Learn weights w_1, w_2, … with an approximation of gradient descent (for $\alpha = 0.01$) so that the resulting perceptron is consistent with an AND

- Result:



Example: Perceptron Learning

- **Example**: Learn weights w_1, w_2, … with an approximation of gradient descent (for α = 0.01) so that the resulting perceptron is consistent with an AND

- Result: