Perceptrons

Sven Koenig, USC

Russell and Norvig, 3rd Edition, Sections 18.7.1-18.7.4

These slides are new and can contain mistakes and typos.
Please report them to Sven (skoenig@usc.edu).

Perceptrons

- We now study how to acquire knowledge with machine learning.

Inductive Learning for Classification

- Labeled examples

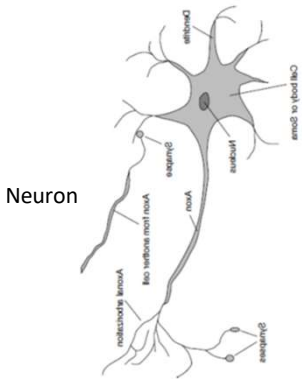
Feature_1	Feature_2	Class
true	true	true
true	false	false
false	true	false

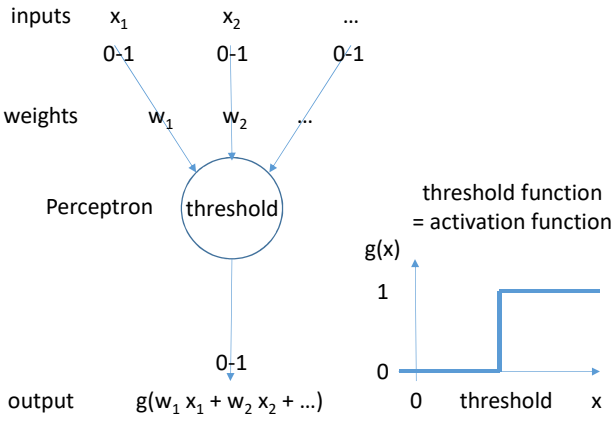
Learn f(Feature_1, Feature_2) = Class from
f(true, true) = true
f(true, false) = false
f(false, true) = false
The function needs to be consistent with all labeled examples
and should make the fewest mistakes on the unlabeled examples.

- Unlabeled examples

Feature_1	Feature_2	Class
false	false	?

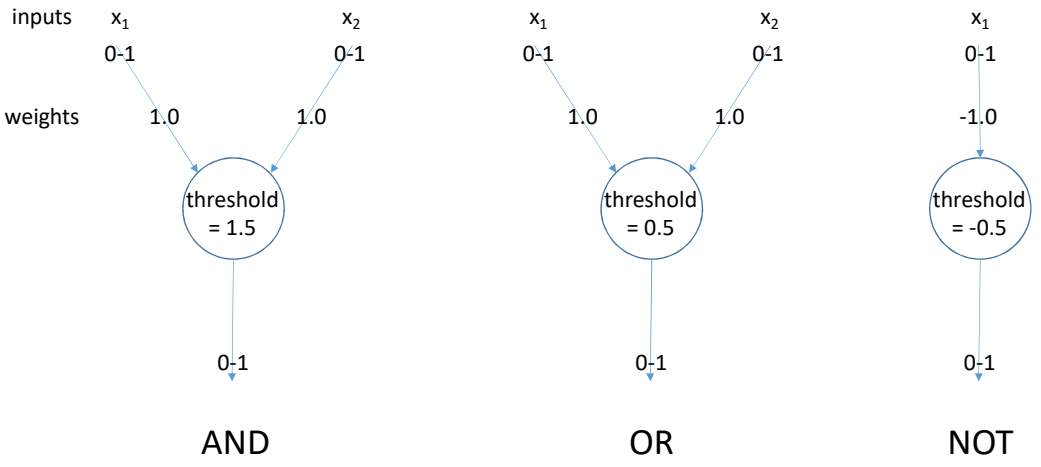
Example: Perceptron Learning





- Objective: Learn the weights for a given perceptron.
- From now on: binary (feature and class) values only (0=false, 1=true).

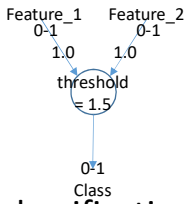
Example: Perceptron Learning



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

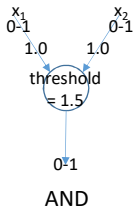
- Can perceptrons represent all Boolean functions?
  f(Feature_1, …, Feature_n) ≡ some propositional sentence

Example: Perceptron Learning

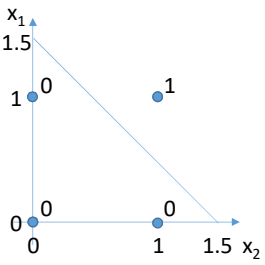
- Can perceptrons represent all Boolean functions?
  f(Feature_1, …, Feature_n) ≡ some propositional sentence
- Linear separability
  - We need to find an n-dimensional plane that separates the labeled examples with class true from the labeled examples with class false.
  - This plane determines the weights and threshold of the perceptron that can then be used to classify the unlabeled examples.

Example: Perceptron Learning

- Can perceptrons represent all Boolean functions?
  f(Feature_1, …, Feature_n) ≡ some propositional sentence
- Linear separability
  - $w_1 x_1 + w_2 x_2$ = threshold
  - $w_1 x_1$ = threshold - $w_2 x_2$
  - x_1 = (threshold / w_1) – (w_2 / w_1) x_2 = (1.5 / 1) – (1 / 1) x_2 = 1.5 – x_2



AND

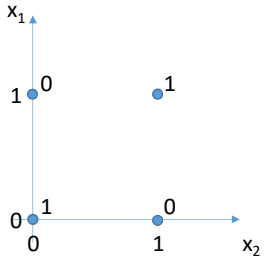


Example: Perceptron Learning

- Can perceptrons represent all Boolean functions?
  f(Feature_1, …, Feature_n) ≡ some propositional sentence
- Linear separability
  - $w_1 x_1 + w_2 x_2$ = threshold
  - $w_1 x_1$ = threshold - $w_2 x_2$
  - x_1 = (threshold / w_1) – (w_2 / w_1) x_2 = (1.5 / 1) – (1 / 1) x_2 = 1.5 – x_2

?

XOR

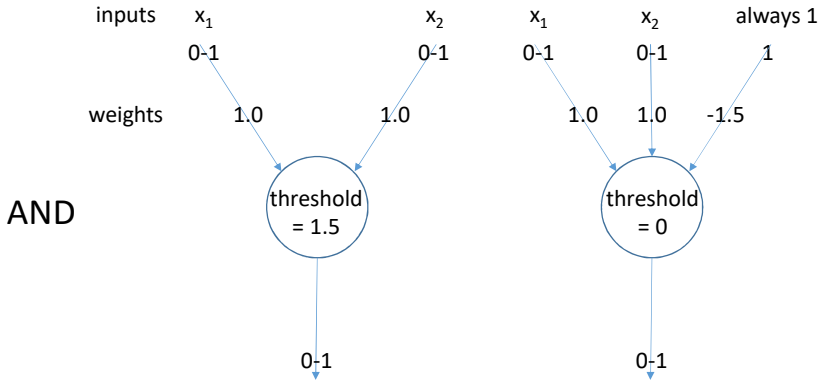


Example: Perceptron Learning

- Can perceptrons represent all Boolean functions? – no!
  f(Feature_1, …, Feature_n) ≡ some propositional sentence
- An XOR cannot be represented with a single perceptron!
- This does not mean that 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 labeled and unlabeled examples.

Example: Perceptron Learning

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

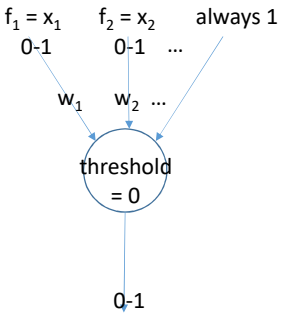


Example: Perceptron Learning

j

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

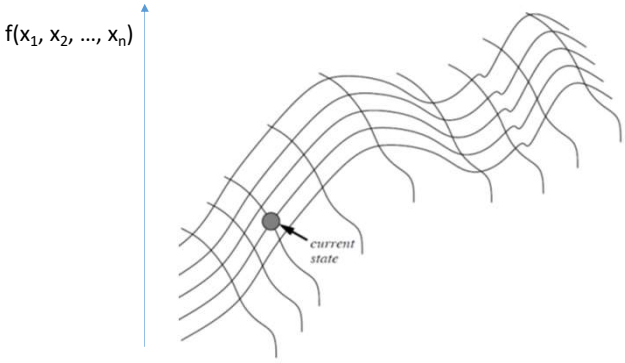
I



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

Gradient Descent

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



Gradient Descent

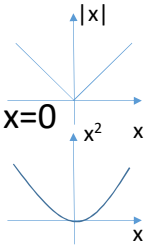
- Finding a local minimum of a differentiable function $f(x_1, x_2, \ldots, x_n)$ with gradient descent
- Initialize $x_1, x_2, \ldots, x_n$ with random values
- Repeat until local minimum reached
  - Update $x_1, x_2, \ldots, x_n$ to correspond to taking a small step against the gradient of $f(x_1, x_2, \ldots, x_n)$ at point $(x_1, x_2, \ldots, x_n)$, where the gradient is $(d\, f(x_1, x_2, \ldots, x_n) / d\, x_1, d\, f(x_1, x_2, \ldots, x_n) / d\, x_2, \ldots, d\, f(x_1, x_2, \ldots, x_n) / d\, x_n)$.

Gradient Descent

- Finding a local minimum of a differentiable function $f(x_1, x_2, \ldots, x_n)$ with gradient descent (for a small positive learning rate α)
- Initialize $x_1, x_2, \ldots, x_n$ with random values
- Repeat until local minimum reached
  - For all x_i in parallel
    - $x_i := x_i - \alpha\, d\, f(x_1, x_2, \ldots, x_n) / d\, x_i$

Example: Perceptron Learning

- We use the number of misclassified labeled examples as error and learn the weights w_1, w_2, … with gradient descent (for a small positive learning rate α) to correspond to a (local) minimum of the error function, that is, so that the resulting perceptron is consistent with all labeled examples:



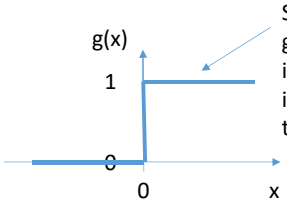
- Minimize Error := 0.5 Σ_l $|o_l - c_l|$ - no: not differentiable at x=0
- Minimize Error := 0.5 Σ_l $(o_l - c_l)^2$

- for $o_l = g(\Sigma_j w_j f_{lj})$, where g() is the activation function.
- The 0.5 is for beauty reasons only (see the slide after the next one).

Example: Perceptron Learning

- Learn the weights w_1, w_2, … with gradient descent (for a small positive learning rate α) so that the resulting perceptron is consistent with all labeled examples:

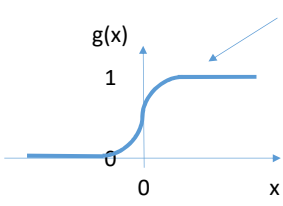
Threshold function



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

no: not differentiable at x=0

Sigmoid function



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

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

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

Example: Perceptron Learning

- Learn the weights w_1, w_2, … with gradient descent (for a small positive learning rate α) so that the resulting perceptron is consistent with all labeled examples:
- Initialize all weights w_j with random values
- Repeat until local minimum reached
  - Let o_l be the output of the perceptron for Example l for the current weights
  - For all weights w_j in parallel
    - $w_j := w_j - \alpha\, d\, Error(w_1, w_2, \ldots) / d\, w_j$

  called one epoch
- Where
  - $d\, Error(w_1, w_2, \ldots) / d\, wj = d\, 0.5\, \Sigma_l\, (o_l - c_l)^2 / d\, w_j = d\, 0.5\, \Sigma_l\, (g(\Sigma_j\, w_j\, f_{lj}) - c_l)^2 / d\, w_j$ $= \Sigma_l\, ((g(\Sigma_j\, w_j\, f_{lj}) - c_l)\, g'(\Sigma_j\, w_j\, f_{lj})\, f_{lj}) = \Sigma_l\, ((o_l - c_l)\, g'(\Sigma_j\, w_j\, f_{lj})\, f_{lj})$

This is the beauty reason!

Example: Perceptron Learning

- Learn the weights w_1, w_2, … with an approximation of gradient descent (for a small positive learning rate α) so that the resulting perceptron is consistent with all labeled examples. Each labeled example is considered individually one after the other:
- Initialize all weights w_j with random values
- Repeat until local minimum reached
  - Let o_l be the output of the perceptron for Example l for the current weights
  - For all labeled examples l
    - For all weights w_j
      - $w_j := w_j - \alpha\, d\, Error(w_1, w_2, \ldots) / d\, w_j$

  called one epoch
- Where
  - $d\, Error(w_1, w_2, \ldots) / d\, wj = (o_l - c_l)\, g'(\Sigma_j\, w_j\, f_{lj})\, f_{lj}$

Example: Perceptron Learning

- Example: Learn the 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 [see the handout for details]

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

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