

Perceptron -- with "OR" example.
Instructor: Nam Sun Wang

A Threshold Unit (Perceptron)...

- Has either binary (-1 or 1, etc.) or real inputs. x_j . (If the input is binary 0/1, change the notation to -1/1. Otherwise, the weight has absolutely no effect on the 0 input.)
- Sums up weighted inputs. $\sigma = \sum w_j \cdot x_j$.
- Tests to see if $\sum w_j \cdot x_j \geq a$, where a is the threshold. (We can simulate the threshold with $x_0=1$ and $w_0=-a$, an extra dummy input of unity and a weight of $-a$. This is also commonly called a *bias*. This is equivalent to the constant part or intercept in linear regression.)
- Yields a binary output (true or false, 0 or 1, -1 or 1, hit or miss, etc.) If $\sum w_j \cdot x_j > a$, then 1; otherwise, 0.

Training Procedure #1. If classification is correct, do nothing. If we get a false positive, set the new w to $w-x$. On the other hand, if we get a false negative, set the new w to $w+x$. For $y=\{-1, 1\}$, we express this procedure algebraically as:

$$\Delta w_j = y_{\text{true}} \cdot x_j \cdot (y \neq y_{\text{true}})$$

To avoid divergence, we update the weights conservatively by taking a fractional step ε instead of a full step. This is commonly known as the *learning rate*. We can start training with a small value of ε , then later increase it to unity near convergence.

$$\Delta w_j = \varepsilon \cdot y_{\text{true}} \cdot x_j \cdot (y \neq y_{\text{true}})$$

Training Procedure #2 (Steepest Descent Algorithm). We can derive the training procedure by minimizing the error function over all i training patterns.

$$\text{Minimize}_{w} \text{sse}(w) = E(w) = \sum_{i=0}^N (\text{error}_i)^2 = \sum_{i=0}^N (y_i - f(x_i \cdot w))^2$$

$$\frac{d}{dw} \text{sse} = -2 \cdot \left[\sum_{i=0}^N (y_i - f(x_i \cdot w)) \cdot x_i \right] \quad \text{Gradient of error: } \Delta E = 2 \cdot \sum_{i=0}^N (f(x_i \cdot w) - y_i) \cdot x_i$$

The following is the steepest descent algorithm that searches in the negative gradient direction of error for updating the j th weight after sequentially presenting each example i .

$$\Delta w_j = -\varepsilon \cdot (f(x_i \cdot w) - y_i) \cdot x_{ij}$$

where ε is a non-negative scalar learning rate. We can either 1) divide the threshold expression by a constant without changing the net effect of the eventual expression, e.g., $\sum w_j \cdot x_j > a$ for $j=\{1, \dots, m\}$ is transformed to $\sum (w_j/a) \cdot x_j > 1$ or 2) to an un-thresholded form $\sum (w_j/\text{const}) \cdot x_j > 0$ for $j=\{0, 1, \dots, m\}$. In practice, because f is a step function, we can force $f(x \cdot w)$ to be simply $x \cdot w$ in weight calculation. This substituted form is more restrictive and is a special case of $f(x \cdot w) = \pm 1 = x \cdot w$.

$$\Delta w_j = -\varepsilon \cdot (x_i \cdot w - y_i) \cdot x_{ij}$$

We can either update the weights immediately after presenting each sample, or we can wait until the entire set of samples have gone through, then calculate the weight change by summing up the errors in that batch.

Training Procedure #3 (Linear Regression or Pseudo-Inverse Method). We realize that y is simply a linear combination of $x^{<j>}$, each weighted by w_j . This is exactly the same as in the plain linear regression. As opposed to the above algorithms, this is a batch procedure where we present all training examples simultaneously rather than sequentially.

$$w = (x^T \cdot x)^{-1} \cdot x^T \cdot y$$

Of course, as in multiple linear regression, if the independent variables are highly correlated, the inverse of $x^T \cdot x$ is very unstable or nonexistent.

Example: An OR function.

Number of inputs: $j := 0..2$

Maximum number of training patterns: $i := 0..10$

Two-dimensional inputs: $x_{i,1} :=$ $x_{i,2} :=$ One-dimensional output: $y_i :=$

-1	-1
-1	1
1	-1
1	1

-1
1
1
1

Number of training patterns: $N := \text{last}(y)$ $i := 0..N$

Add a dummy input (in lieu of threshold): $x_{i,0} := 1$

Output function: $\sigma(x, w) := x \cdot w$ Element-wise notation: $\sigma(x, w) := \sum_{j=0}^2 w_j \cdot x_j$
 $f(x, w) := \text{if}(\sigma(x, w) \geq 0, 1, -1)$

Assign initial weights:

$$w_j := \text{rnd}(2) - 1$$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k \cdot (N+1) + i + 1>} := w^{<k \cdot (N+1) + i>} + y_i \cdot (x^T)^{<i>} \cdot \left[f \left[(x^T)^{<i>T}, w^{<k \cdot (N+1) + i>} \right] \neq y_i \right]$$

Results after iteration:

	0	1	2	3	4	5	6	7	
$w =$	0	-0.997	-0.997	0.003	1.003	1.003	0.003	0.003	1.003
	1	-0.613	-0.613	-1.613	-0.613	-0.613	0.387	0.387	1.387
	2	0.17	0.17	1.17	0.17	0.17	1.17	1.17	0.17

The last weight: $\omega := w^{<cols(w) - 1>}$

$$\omega = \begin{pmatrix} 1.003 \\ 1.387 \\ 2.17 \end{pmatrix}$$

Prediction for different cases:

$$\begin{aligned} f((1 \ -1 \ -1), \omega) &= -1 \\ f((1 \ -1 \ 1), \omega) &= 1 \\ f((1 \ 1 \ -1), \omega) &= 1 \\ f((1 \ 1 \ 1), \omega) &= 1 \end{aligned} \quad \leftarrow \text{compare} \rightarrow \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

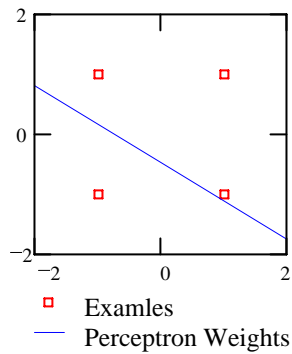
Prediction function defined in terms of the original perceptron without the dummy bias input:

$$y_{\text{pred}}(x) := \text{if} \left(\sum_{i=0}^1 \omega_{i+1} \cdot x_{0,i} \geq -\omega_0, 1, -1 \right)$$

$$\begin{aligned} y_{\text{pred}}((-1 \ -1)) &= -1 \\ y_{\text{pred}}((-1 \ 1)) &= 1 \\ y_{\text{pred}}((1 \ -1)) &= 1 \\ y_{\text{pred}}((1 \ 1)) &= 1 \end{aligned} \quad \leftarrow \text{compare} \rightarrow \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The weights of the perceptron defines a line that divides the $x^{<1>-x^{<2>}$ plane into two sections/regions. For an m-dimensional input, each perceptron defines a hyperplane in an m-dimensional space.

$$x_2(x_1, \omega) := -\frac{1}{\omega_2} \cdot (\omega_0 + \omega_1 \cdot x_1) \quad x_1 := -2, -1.9..2$$



Example. -- Training Method #2 with $x \cdot w$.

Reset weight: $w := 0$ $w_j := \text{rnd}(2) - 1$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$\Delta w_j = -\varepsilon \cdot (x_i \cdot w_j - y_i) \cdot x_i \quad \varepsilon := 0.2$$

$$w^{<k \cdot (N+1) + i + 1>} := w^{<k \cdot (N+1) + i>} - \varepsilon \cdot \begin{bmatrix} (x^T)^{<i>} \\ (x^T)^{<i>T} \end{bmatrix} \cdot w^{<k \cdot (N+1) + i>} - y_i$$

Results after iteration:

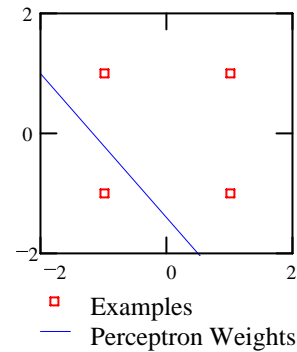
	0	1	2	3	4	5	6	7	
$w =$	0	-0.299	-0.441	0.107	0.245	0.341	0.166	0.429	0.552
	1	0.646	0.787	0.239	0.378	0.473	0.648	0.385	0.509
	2	-0.652	-0.51	0.037	-0.101	-0.005	0.169	0.432	0.308

The last weight: $\omega := w^{<\text{cols}(w) - 1>}$

$$\omega = \begin{pmatrix} 0.5 \\ 0.429 \\ 0.357 \end{pmatrix}$$

Prediction for different cases:

$$\begin{aligned} f((1 \ -1 \ -1), \omega) &= -1 \\ f((1 \ -1 \ 1), \omega) &= 1 \\ f((1 \ 1 \ -1), \omega) &= 1 \\ f((1 \ 1 \ 1), \omega) &= 1 \end{aligned} \quad \leftarrow \text{compare} \rightarrow \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$



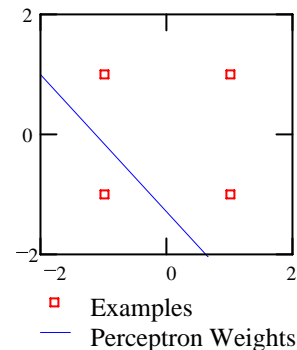
Note that **the answer is not unique**. Many combinations of weights give rise to the same result. Here is another set of weights that work just as well.

The next-to-last weight: $\omega := w^{<\text{cols}(w) - 2>}$

$$\omega = \begin{pmatrix} 0.643 \\ 0.571 \\ 0.5 \end{pmatrix}$$

Prediction for different cases:

$$\begin{aligned} f((1 \ -1 \ -1), \omega) &= -1 \\ f((1 \ -1 \ 1), \omega) &= 1 \\ f((1 \ 1 \ -1), \omega) &= 1 \\ f((1 \ 1 \ 1), \omega) &= 1 \end{aligned} \quad \leftarrow \text{compare} \rightarrow \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$



Example. -- Training Method #2 with $f(x,w)$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k \cdot (N+1) + i + 1>} := w^{<k \cdot (N+1) + i>} - \varepsilon \cdot \left(x^{<i>} \cdot \left[f \left(\left(x^{<i>} \right)^T, w^{<k \cdot (N+1) + i>} \right) - y_i \right] \right)$$

Results after iteration:

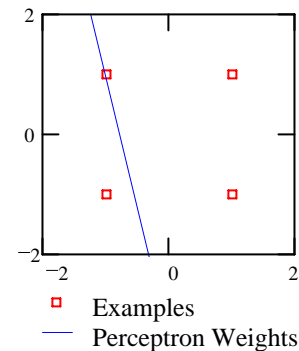
	0	1	2	3	4	5	6	7	
$w =$	0	-0.299	-0.299	0.101	0.101	0.101	-0.299	0.101	0.501
	1	0.646	0.646	0.246	0.246	0.246	0.646	0.246	0.646
	2	-0.652	-0.652	-0.252	-0.252	-0.252	0.148	0.548	0.148

The last weight: $\omega := w^{<\text{cols}(w) - 1>}$

$$\omega = \begin{pmatrix} 0.501 \\ 0.646 \\ 0.148 \end{pmatrix}$$

Prediction for different cases:

$$\begin{aligned} f((1 \ -1 \ -1), \omega) &= -1 \\ f((1 \ -1 \ 1), \omega) &= 1 \\ f((1 \ 1 \ -1), \omega) &= 1 \\ f((1 \ 1 \ 1), \omega) &= 1 \end{aligned} \quad \leftarrow \text{compare} \rightarrow \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

**Example. -- Training Method #2 with $f(x,w)$ but with **batch updating of weights.****

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k+1>} := w^{<k>} - \varepsilon \cdot \sum_{i=0}^N \left(x^{<i>} \cdot \left(f \left(\left(x^{<i>} \right)^T, w^{<k>} \right) - y_i \right) \right)$$

Results after iteration:

	0	1	2	3	4	5	6	7
$w =$	0	-0.299	0.501	0.501	0.501	0.501	0.501	0.501
	1	0.646	0.646	0.646	0.646	0.646	0.646	0.646
	2	-0.652	0.148	0.148	0.148	0.148	0.148	0.148

The last weight: $\omega := w^{<10>}$

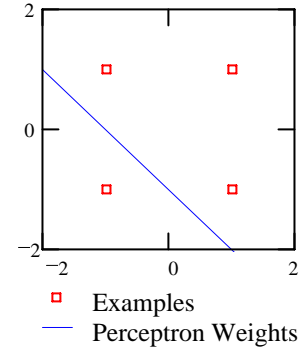
$$\omega = \begin{pmatrix} 0.501 \\ 0.646 \\ 0.148 \end{pmatrix} \quad \leftarrow \text{Same set of weights as in the last problem but converged quickly.}$$

Example. -- Training Method #3 with Pseudo-Inverse.

$$\omega := \left(X^T \cdot X \right)^{-1} \cdot X^T \cdot y \quad \omega = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Prediction for different cases:

$$\begin{aligned} f((1 \ -1 \ -1), \omega) &= -1 \\ f((1 \ -1 \ 1), \omega) &= 1 \\ f((1 \ 1 \ -1), \omega) &= 1 \quad \leftarrow \text{compare} \rightarrow \\ f((1 \ 1 \ 1), \omega) &= 1 \end{aligned} \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$



Example. -- Effect of repeating the same pattern many times.

Expand the number of sample points in the training pattern: $ii := 4 .. 25 \leftarrow$ Try different upper limits.

Repeat the first example many times.

$$x_{ii,0} := 1 \quad x_{ii,1} := -1 \quad x_{ii,2} := -1 \quad y_{ii} := -1$$

Number of training patterns: $N := \text{last}(y) \quad i := 0 .. N$

Present different examples and iterate (hopefully until convergence): $k := 0 .. 10$

$$w^{<k+1>} := w^{<k>} - \epsilon \cdot \sum_{i=0}^N \left(x^{<i>} \cdot \left(f\left(\left(x^{<i>} \right)^T, w^{<k>} \right) - y_i \right) \right)$$

Results after iteration:

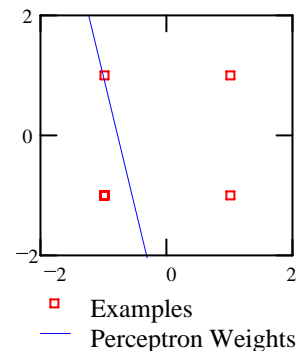
	0	1	2	3	4	5	6	7	
w =	0	-0.299	0.501	0.501	0.501	0.501	0.501	0.501	0.501
	1	0.646	0.646	0.646	0.646	0.646	0.646	0.646	0.646
	2	-0.652	0.148	0.148	0.148	0.148	0.148	0.148	0.148

The last weight: $\omega := w^{<10>} \quad \omega = \begin{pmatrix} 0.501 \\ 0.646 \\ 0.148 \end{pmatrix}$

Prediction for different cases:

$$\begin{aligned} f((1 \ -1 \ -1), \omega) &= -1 \\ f((1 \ -1 \ 1), \omega) &= 1 \\ f((1 \ 1 \ -1), \omega) &= 1 \quad \leftarrow \text{compare} \rightarrow \\ f((1 \ 1 \ 1), \omega) &= 1 \end{aligned} \quad y = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 3 \\ 1 \end{bmatrix}$$

No good!



Because we try to minimize the sum of squared error, presenting the first case many times puts more emphasis on that example. The line moves toward the first case as we increase the number of times we train the perceptron with the first case, eventually leading to erroneous predictions. Thus, we must choose a representative set of samples for training. Otherwise, the model may be skewed.

Example. -- Output y is binary in 0/1 instead of -1/1. Training Method #2 with x·w.

Maximum number of training patterns: $i := 0..10$

Reset input and output variables: $x := 0 \quad y := 0$

Two-dimensional inputs: $x_{i,1} := \quad x_{i,2} :=$ One-dimensional output: $y_i :=$

-1	-1	0
-1	1	1
1	-1	1
1	1	1

Number of training patterns: $N := \text{last}(y) \quad i := 0..N$

Add a dummy input (in lieu of threshold): $x_{i,0} := 1$

Output function: $\sigma(x, w) := x \cdot w$
 $f(x, w) := \text{if}(\sigma(x, w) \geq 0, 1, 0)$

Reset weight: $w := 0 \quad w_j := \text{rnd}(2) - 1$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k \cdot (N+1) + i + 1>} := w^{<k \cdot (N+1) + i>} - \epsilon \cdot (x^T)^{<i>} \cdot \left[(x^T)^{<i>T} \cdot w^{<k \cdot (N+1) + i>} - y_i \right]$$

Results after iteration:

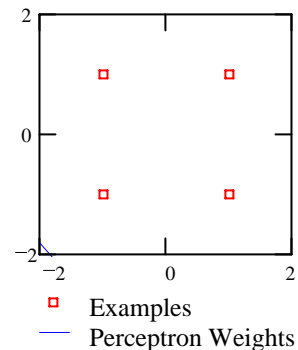
	0	1	2	3	4	5	6	7
0	0.421	0.095	0.361	0.51	0.72	0.548	0.677	0.755
1	-0.392	-0.066	-0.332	-0.183	0.027	0.198	0.069	0.147
2	-0.817	-0.491	-0.225	-0.374	-0.165	0.007	0.135	0.058

The last weight: $\omega := w^{<\text{cols}(w) - 1>} \quad \omega = \begin{pmatrix} 0.75 \\ 0.214 \\ 0.179 \end{pmatrix}$

Prediction for different cases:

$f((1 \ -1 \ -1), \omega) = 1$
 $f((1 \ -1 \ 1), \omega) = 1$
 $f((1 \ 1 \ -1), \omega) = 1$ ← compare → $y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
 $f((1 \ 1 \ 1), \omega) = 1$ **No good!**

No good because the approximation of $f(x \cdot w)$ with $x \cdot w$ is valid only when $f(x \cdot w) = \pm 1 = x \cdot w$. Now, $f = \{0 \text{ or } 1\}$ violates this assumption.



Example. -- Output y is binary in 0/1 instead of -1/1. Training Method #2 with $f(x,w)$

Output function: $\sigma(x, w) := x \cdot w$
 $f(x, w) := \text{if}(\sigma(x, w)_0 \geq 0, 1, 0)$

Reset weight: $w := 0 \quad w_j := \text{rnd}(2) - 1$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k \cdot (N+1) + i + 1>} := w^{<k \cdot (N+1) + i>} - \epsilon \cdot (x^T)^{<i>} \cdot \left[f \left[(x^T)^{<i>}^T, w^{<k \cdot (N+1) + i>} \right] - y_i \right]$$

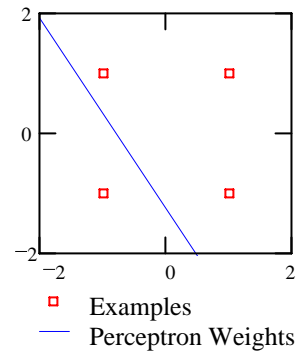
Results after iteration:

	0	1	2	3	4	5	6
0	-0.705	-0.705	-0.505	-0.505	-0.305	-0.305	-0.105
1	0.977	0.977	0.777	0.777	0.977	0.977	0.777
2	-0.762	-0.762	-0.562	-0.562	-0.362	-0.362	-0.162

The last weight: $\omega := w^{<\text{cols}(w) - 1>}$
 $\omega = \begin{pmatrix} 0.295 \\ 0.377 \\ 0.238 \end{pmatrix}$

Prediction for different cases:

$f((1 \ -1 \ -1), \omega) = 0$
 $f((1 \ -1 \ 1), \omega) = 1$
 $f((1 \ 1 \ -1), \omega) = 1$ ← compare → $y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
 $f((1 \ 1 \ 1), \omega) = 1$ Now, the answer is O.K.

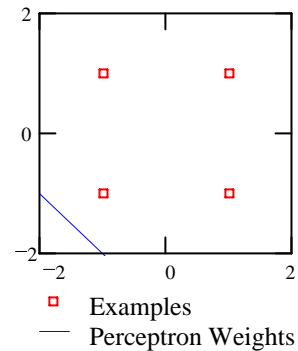


Example. -- Output y is binary in 0/1 instead of -1/1. Training Method #3 with Pseudo-Inverse.

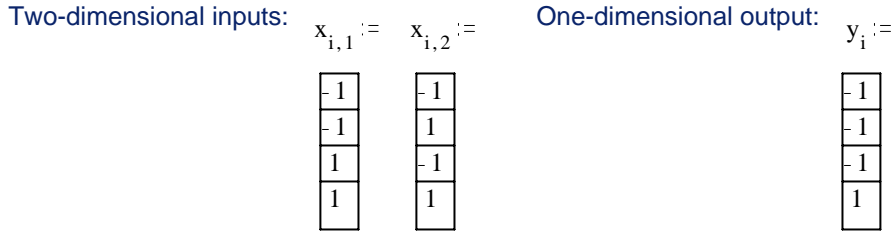
$\omega := 0 \quad \omega := (x^T \cdot x)^{-1} \cdot x^T \cdot y \quad \omega = \begin{pmatrix} 0.75 \\ 0.25 \\ 0.25 \end{pmatrix}$

Prediction for different cases:

$f((1 \ -1 \ -1), \omega) = 1$
 $f((1 \ -1 \ 1), \omega) = 1$
 $f((1 \ 1 \ -1), \omega) = 1$ ← compare → $y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
 $f((1 \ 1 \ 1), \omega) = 1$ **No good!**



Example: An AND function trained with the batch method #2.



Output function: $\sigma(x, w) := x \cdot w$
 $f(x, w) := \text{if}(\sigma(x, w) \geq 0, 1, -1)$

Assign initial weights:

$w := 0$ $w_j := \text{rnd}(2) - 1$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k+1>} := w^{<k>} - \varepsilon \cdot \sum_{i=0}^N \left(x^{<i>} \cdot \left(f\left(\left(x^{<i>} \right)^T, w^{<k>} \right) - y_i \right) \right)$$

Results after iteration:

	0	1	2	3	4	5	6
$w =$	0	-0.982	-0.582	-0.582	-0.582	-0.582	-0.582
1	0.063	0.463	0.463	0.463	0.463	0.463	0.463
2	0.204	0.604	0.604	0.604	0.604	0.604	0.604

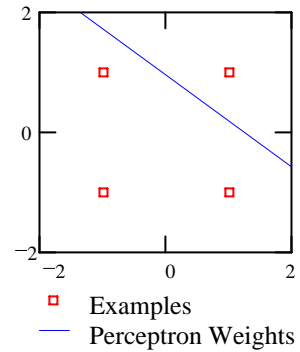
The last weight: $\omega := w^{<\text{cols}(w) - 1>}$

$$\omega = \begin{pmatrix} -0.582 \\ 0.463 \\ 0.604 \end{pmatrix}$$

Prediction for different cases:

$f((1 \ -1 \ -1), \omega) = -1$
 $f((1 \ -1 \ 1), \omega) = -1$
 $f((1 \ 1 \ -1), \omega) = -1$ ← compare → $y = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$
 $f((1 \ 1 \ 1), \omega) = 1$

Note how the line defined by the perceptron weights has shifted from the OR function.



Example: An XOR function trained with the batch method #2.

Two-dimensional inputs: $x_{i,1} :=$ $x_{i,2} :=$ One-dimensional output: $y_i :=$

-1
-1
1
1

-1
1
-1
1

-1
1
1
-1

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k+1>} := w^{<k>} - \epsilon \cdot \sum_{i=0}^N (x^T)^{<i>} \cdot \left(f\left((x^T)^{<i>} \cdot w^{<k>} \right) - y_i \right)$$

Results after iteration:

	0	1	2	3	4	5	6	
$w =$	0	-0.982	-0.182	0.218	-0.182	0.218	-0.182	0.218
	1	0.063	0.063	-0.337	0.063	-0.337	0.063	-0.337
	2	0.204	0.204	-0.196	0.204	-0.196	0.204	-0.196

The last weight: $\omega := w^{<cols(w)-1>}$

$$\omega = \begin{pmatrix} -0.182 \\ 0.063 \\ 0.204 \end{pmatrix}$$

Prediction for different cases:

$f((1 \ -1 \ -1), \omega) = -1$
 $f((1 \ -1 \ 1), \omega) = -1$
 $f((1 \ 1 \ -1), \omega) = -1$ ← compare → $y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$
 $f((1 \ 1 \ 1), \omega) = 1$

No good!

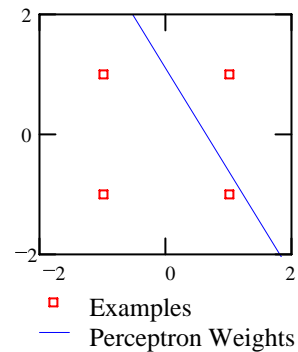
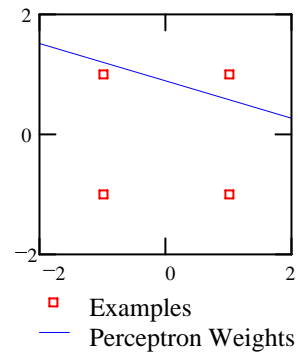
The next-to-last weight: $\omega := w^{<cols(w)-2>}$

$$\omega = \begin{pmatrix} 0.218 \\ -0.337 \\ -0.196 \end{pmatrix}$$

Prediction for different cases:

$f((1 \ -1 \ -1), \omega) = 1$
 $f((1 \ -1 \ 1), \omega) = 1$
 $f((1 \ 1 \ -1), \omega) = 1$ ← compare → $y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$
 $f((1 \ 1 \ 1), \omega) = -1$

No good!



The weights are oscillating. Like the first set of weights above, the second set of weights also fail to yield the correct answers. It is clear that one perceptron with the original unadulterated input variables $x^{<1>}$ and $x^{<2>}$ alone cannot adequately divide the $x^{<1>}$ - $x^{<2>}$ plane into regions where the output is either +1 or -1 such that the patterns match the training data. In other words, one perceptron cannot learn the XOR function, because we need to have at least two straight lines to separate the +1 and -1 outputs, although it maybe possible to do so with a nonlinear curve. Let us generate additional variables consisting of **auto- and cross-products** of $x^{<1>}$ and $x^{<2>}$.

$$x^{<3>} := \overline{(x^{<1>})^2} \quad x^{<4>} := \overline{(x^{<1>} \cdot x^{<2>})} \quad x^{<5>} := \overline{(x^{<2>})^2} \quad j := 0..5$$

Likewise, let us enlarge the weight vector accordingly and assign initial weights.

$$w := 0 \quad w_j := \text{rnd}(2) - 1$$

Present different examples and iterate (hopefully until convergence): $k := 0..10$

$$w^{<k+1>} := w^{<k>} - \varepsilon \cdot \sum_{i=0}^N (x^T)^{<i>} \cdot \left(f\left((x^T)^{<i>T} \cdot w^{<k>} \right) - y_i \right)$$

Results after iteration:

	0	1	2	3	4	5	6
0	-0.668	-0.668	-1.068	-1.068	-1.068	-1.068	-1.068
1	-0.098	-0.098	0.302	0.302	0.302	0.302	0.302
2	-0.886	-0.086	0.314	0.314	0.314	0.314	0.314
3	0.567	0.567	0.167	0.167	0.167	0.167	0.167
4	0.04	-0.76	-1.16	-1.16	-1.16	-1.16	-1.16
5	0.752	0.752	0.352	0.352	0.352	0.352	0.352

The last weight: $\omega := w^{<\text{cols}(w)-1>} \quad \omega^T = (-1.068 \quad 0.302 \quad 0.314 \quad 0.167 \quad -1.16 \quad 0.352)$

The following equation describes the curve in the $x^{<1>}$ - $x^{<2>}$ plane defined by the weights of the perceptron.

$$w_5 \cdot x_2^2 + w_4 \cdot x_1 \cdot x_2 + w_3 \cdot x_1^2 + w_2 \cdot x_2 + w_1 \cdot x_1 + w_0 = \dots \text{ mark } x_2 \text{ and choose [Symbolic]Solve for}$$

$$x_{2a}(x_1, w) := \frac{1}{2 \cdot w_5} \left[- (w_4 \cdot x_1 + w_2) + \sqrt{(w_4 \cdot x_1 + w_2)^2 - 4 \cdot w_5 \cdot (w_3 \cdot x_1^2 + w_1 \cdot x_1 + w_0)} \right]$$

$$x_{2b}(x_1, w) := \frac{1}{2 \cdot w_5} \left[- (w_4 \cdot x_1 + w_2) - \sqrt{(w_4 \cdot x_1 + w_2)^2 - 4 \cdot w_5 \cdot (w_3 \cdot x_1^2 + w_1 \cdot x_1 + w_0)} \right]$$

Prediction for different cases:

$$f\left(\left(\begin{matrix} x \\ 1 \end{matrix}\right)^T \begin{matrix} <0> \\ <1> \\ <2> \\ <3> \end{matrix}^T, \omega\right) = -1$$

$$f\left(\left(\begin{matrix} x \\ 1 \end{matrix}\right)^T \begin{matrix} <1> \\ <2> \\ <3> \end{matrix}^T, \omega\right) = 1$$

$$f\left(\left(\begin{matrix} x \\ 1 \end{matrix}\right)^T \begin{matrix} <2> \\ <3> \end{matrix}^T, \omega\right) = 1$$

$$f\left(\left(\begin{matrix} x \\ 1 \end{matrix}\right)^T \begin{matrix} <3> \end{matrix}^T, \omega\right) = -1$$

$$\leftarrow \text{compare} \rightarrow \quad y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

With the newly added auto- and cross-terms, one perceptron can learn the XOR function.

percept.mcd

