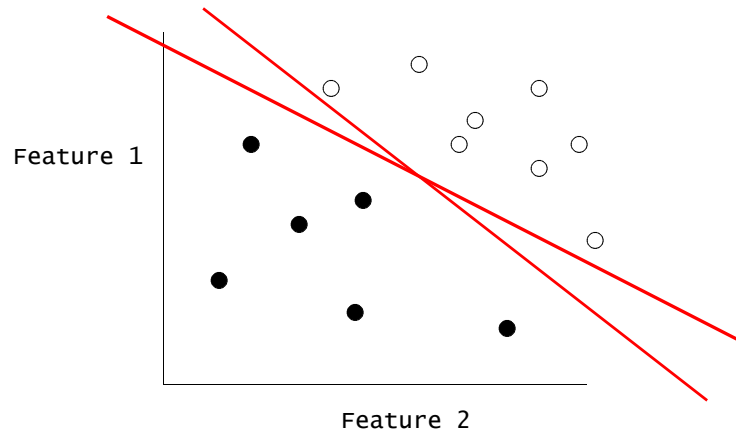


Example where line won't work?



Linear Discriminant Functions

- Example: Boolean functions in 2 dimensions, 3 dimensions, 4 dimensions,
- Notion of “decision surfaces”
- Linear discriminant function: Assuming data is D -dimensional, define $(D-1)$ -dimensional hyperplanes to classify the data into classes.

Perceptrons

- Precursors to neural networks.
- Discriminant function:

$$y(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x} + w_0)$$
$$= f(w_0 + w_1 x_1 + \dots + w_D x_D)$$

w_0 is called the “bias”. (Equivalently, $-w_0$ is called the “threshold”.)

Classification:

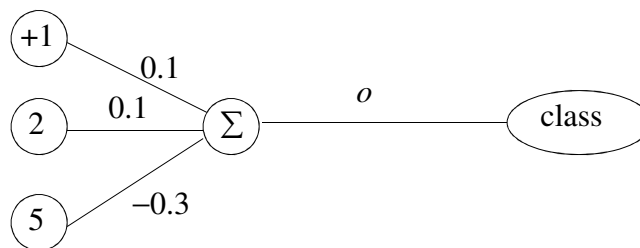
$$y(\mathbf{x}) = o(w_1 x_1 + w_2 x_2 + \dots + w_D x_D + w_0)$$

$$\text{where } o(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

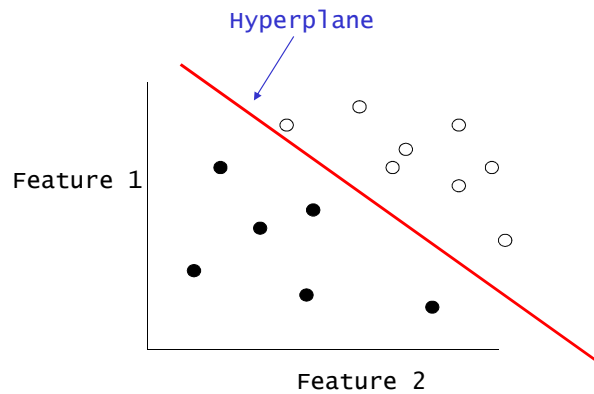
o is called the
“activation function”

Example

What is the class?



Geometry of the perceptron



In 2d:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

In-class exercise

(a) Find weights for a perceptron that separates “true” and “false” in $x_1 \wedge x_2$. Sketch the separation line defined by this discriminant.

(b) Do the same, but for $x_1 \vee x_2$.

- To simplify notation, assume a “dummy” coordinate (or attribute) $x_0 = 1$. Then we can write:

$$y(\mathbf{x}) = o(\mathbf{w}^T \mathbf{x})$$

$$= o(w_0 x_0 + w_1 x_1 + \dots + w_D x_D)$$

- We can generalize the perceptron to cases where we project data points \mathbf{x}_n into “feature space”, $\phi(\mathbf{x}_n)$:

$$y(\mathbf{x}) = o(\mathbf{w}^T \phi(\mathbf{x}))$$

Notation

- Let $S = \{(\mathbf{x}^n, t^n) : n = 1, 2, \dots, m\}$ be a training set.

Note: \mathbf{x}^n is a vector of inputs, and t^n is $\in \{+1, -1\}$ for binary classification, $t^n \in \mathfrak{R}$ for regression.

- Output y :

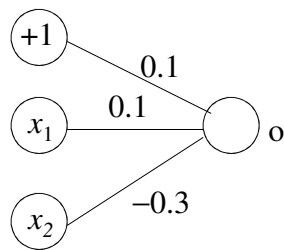
$$y = o\left(\sum_{j=0}^d w_j x_j = \mathbf{w}^T \cdot \mathbf{x}\right)$$

- Error of perceptron on training example n :

$$E^n = \frac{1}{2} (t^n - y^n)^2$$

Example

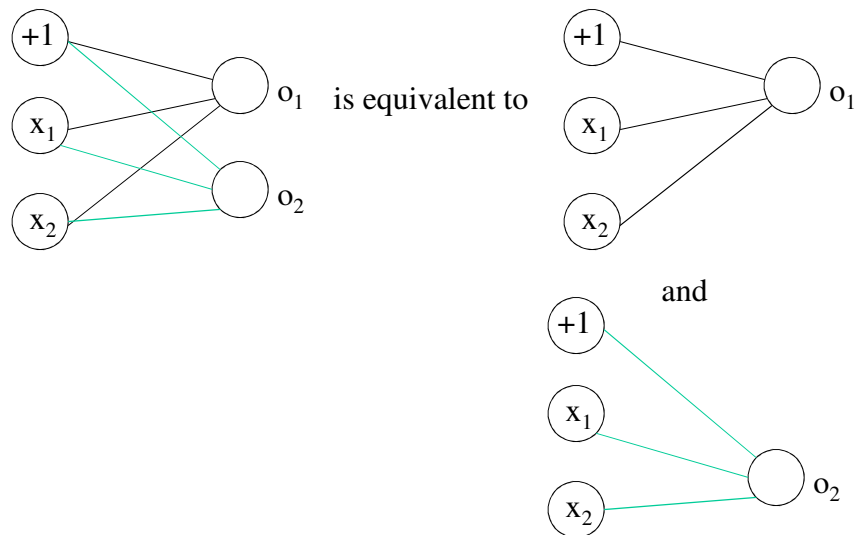
- $S = \{((0,0), -1), ((0,1), 1)\}$
- Let $\mathbf{w} = \{w_0, w_1, w_2\} = \{0.1, 0.1, -0.3\}$



What is E^1 ?

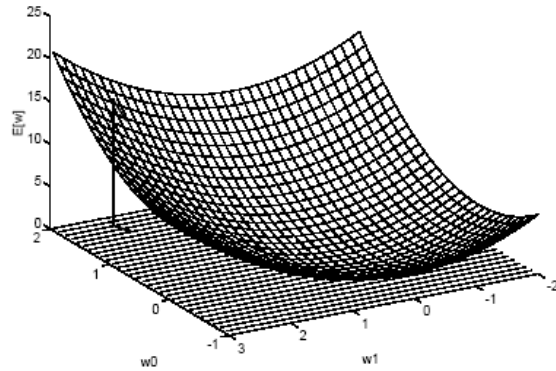
What is E^2 ?

Perceptrons with multiple outputs



How do we train a perceptron?

Gradient descent in weight space



From T. M. Mitchell, *Machine Learning*

Perceptron learning algorithm

- Start with small random (positive and negative) weights $\mathbf{w} = (w_1, w_2, \dots, w_d)$.
- Do gradient descent in weight space, in order to minimize error E :
 - Given error E , want to modify weights \mathbf{w} so as to take a step in direction of steepest descent.

Gradient descent

- We want to find \mathbf{w} so as to minimize *sum-squared error*:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=0}^m (t^n - y^n)^2$$

- To minimize, take the derivative of $E(\mathbf{w})$ with respect to \mathbf{w} .
- A vector derivative is called a “gradient”: $\nabla E(\mathbf{w})$

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Here is how we change each weight:

$$w_j \leftarrow w_j + \Delta w_j$$

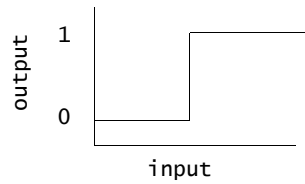
where

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

and η is the *learning rate*.

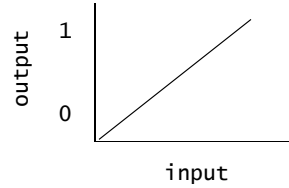
But...what is the problem with calculating $\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$?

Activation functions



$$y = o\left(\sum_j w_j x_j + w_0\right)$$

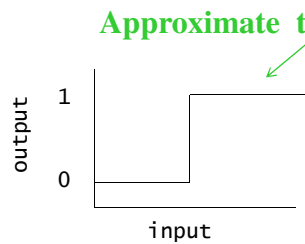
Not differentiable



$$y = \sum_j w_j x_j + w_0$$

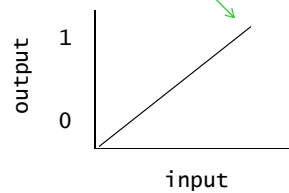
Differentiable

Activation functions



$$y = o\left(\sum_j w_j x_j + w_0\right)$$

Not differentiable



$$y = \sum_j w_j x_j + w_0$$

Differentiable

Approximate this with this

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_n (t^n - y^n)^2 \quad (1)$$

$$= \frac{1}{2} \sum_n \frac{\partial}{\partial w_i} (t^n - y^n)^2 \quad (2)$$

$$= \frac{1}{2} \sum_n 2(t^n - y^n) \frac{\partial}{\partial w_i} (t^n - y^n) \quad (3)$$

$$= \sum_n (t^n - y^n) \frac{\partial}{\partial w_i} (t^n - \mathbf{w} \cdot \mathbf{x}^n) \quad (4)$$

$$= \sum_n (t^n - y^n)(-x_i^n) \quad (5)$$

So,

$$\Delta w_i = \eta \sum_n (t^n - y^n) x_i^n \quad (6)$$

This is called the *perceptron learning rule*.

- Problem with *true gradient descent*:

Search process will land in local optimum.

- Common approach to this: use *stochastic gradient descent*:
 - Instead of doing weight update after all training examples have been processed, do weight update after each training example has been processed (i.e., perceptron output has been calculated).
 - Stochastic gradient descent approximates true gradient descent increasingly well as $\eta \rightarrow 1/\infty$.

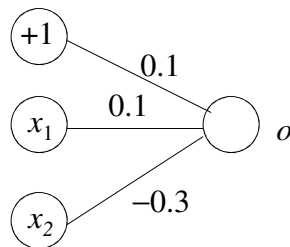
Training a perceptron

1. Start with random weights, $\mathbf{w} = (w_1, w_2, \dots, w_d)$.
2. Select training example (\mathbf{x}^n, t^n) .
3. Run the perceptron with input \mathbf{x}^n and weights \mathbf{w} to obtain $y = o(\mathbf{x}^n)$.
4. Let η be the training rate (a user-set parameter).
Now,
$$w_i \leftarrow w_i + \Delta w_i$$
where
$$\Delta w_i = \eta (t^n - y^n) x_i^n$$
5. Go to 2.

Example (slightly modified)

- $S = \{((0,0), -1), ((0,1), 1)\}$
- Let $\mathbf{w} = \{w_0, w_1, w_2\} = \{0.1, 0.1, -0.3\}$

Perceptron weight updates:



New perceptron weights:

Homework 1

Checked History of the Perceptron

- 1960s: Rosenblatt proved that the perceptron learning rule converges to correct weights in a finite number of steps, provided the training examples are linearly separable.
- 1969: Minsky and Papert proved that perceptrons cannot represent non-linearly separable target functions.

- **Good news:** Adding hidden layer allows more target functions to be represented.
- **Bad news:** No algorithm for learning in multi-layered networks, and no convergence theorem!
- Quote from Minsky and Papert's book, *Perceptrons* (1969):

"[The perceptron] has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgment that the extension is sterile."

- Two major problems they saw were:
 1. How can the learning algorithm apportion credit (or blame) to individual weights for incorrect classifications depending on a (sometimes) large number of weights?
 2. How can such a network learn useful higher-order features?
- **Good news:** Successful credit-apportionment learning algorithms developed soon afterwards (e.g., back-propagation). Still successful, in spite of lack of convergence theorem.

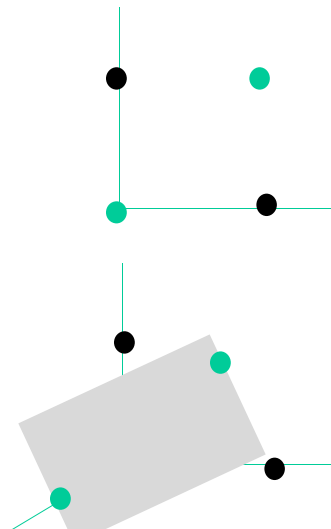
Limitations of perceptrons

- Perceptrons only be 100% accurate only on linearly separable problems.
- Multi-layer networks (often called *multi-layer perceptrons*, or *MLPs*) can represent any target function.
- However, in multi-layer networks, there is no guarantee of convergence to minimal error weight vector.

XOR in 2D and in 3D

x1	x2	Output
0	0	0
0	1	1
1	0	1
1	1	0

x1	x2	x3	Output
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0



Overfitting

- Adding too many features (i.e., making problem higher-dimensional) decreases performance of classifier.
- Also, training for too long.
- How to avoid overfitting?
- How to use validation set

Linear Regression

- Difference between classification and regression
- Least squares regression:

Equation for line: $y = \sum_j w_j x_j$

Error function:

$$E = \left(\sum_j t^j - \sum_i w_i x_i^j \right)^2$$

Gradient descent: Find weights w_j that minimize E