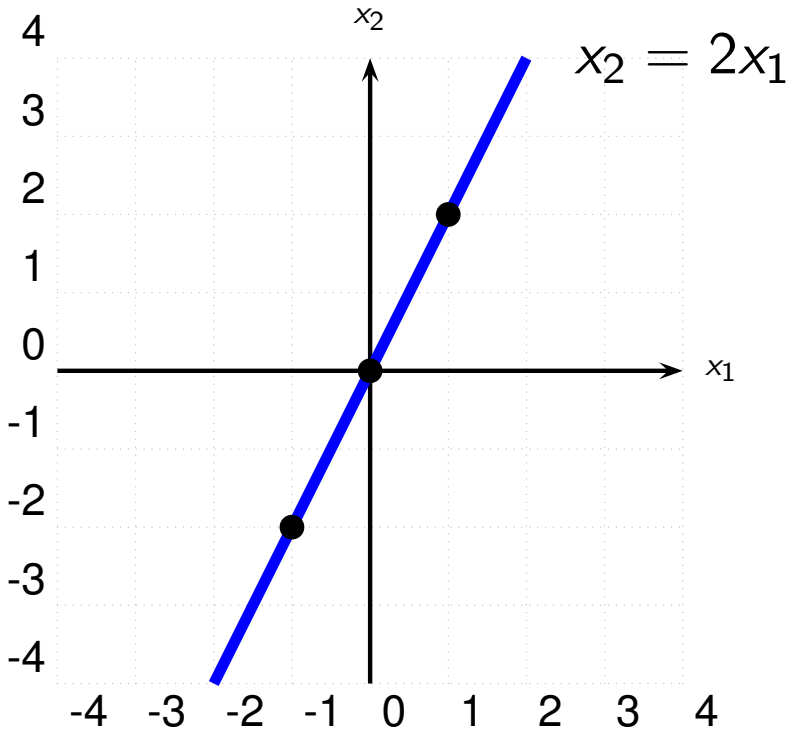
Linear Classifier

- Linear classifiers are single layer neural networks.



Observe, that $x_2 = 2x_1$ can also be expressed as

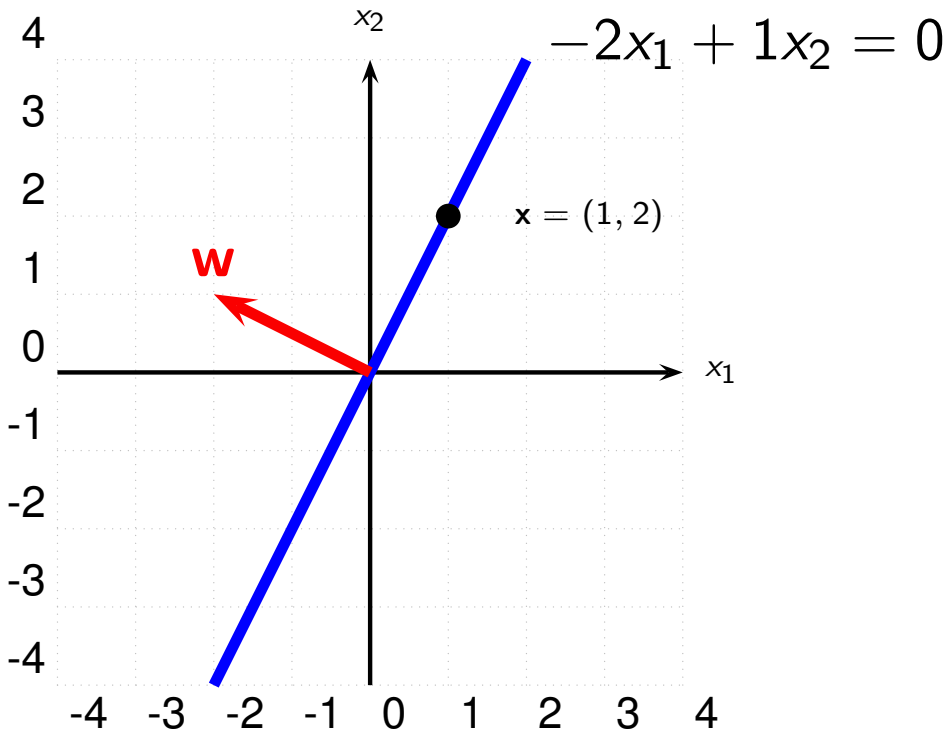
$$w_1 x_1 + w_2 x_2 = 0 \Leftrightarrow x_2 = -\frac{w_1}{w_2} x_1,$$

where for instance

$$w_1 = -2,\ w_2 = 1.$$

Furthermore, observe that all points lying on the line $x_2 = 2x_1$ satisfy $w_1 x_1 + w_2 x_2 = -2x_1 + 1x_2 = 0$.

Linear Classifier & Dot Product



- What about the vector $\mathbf{w} = (w_1, w_2) = (-2, 1)$?
- Vector $\mathbf{w}$ is perpendicular to the line $-2x_1 + 1x_2 = 0$.
- Let us calculate the dot product of $\mathbf{w}$ and $\mathbf{x}$.

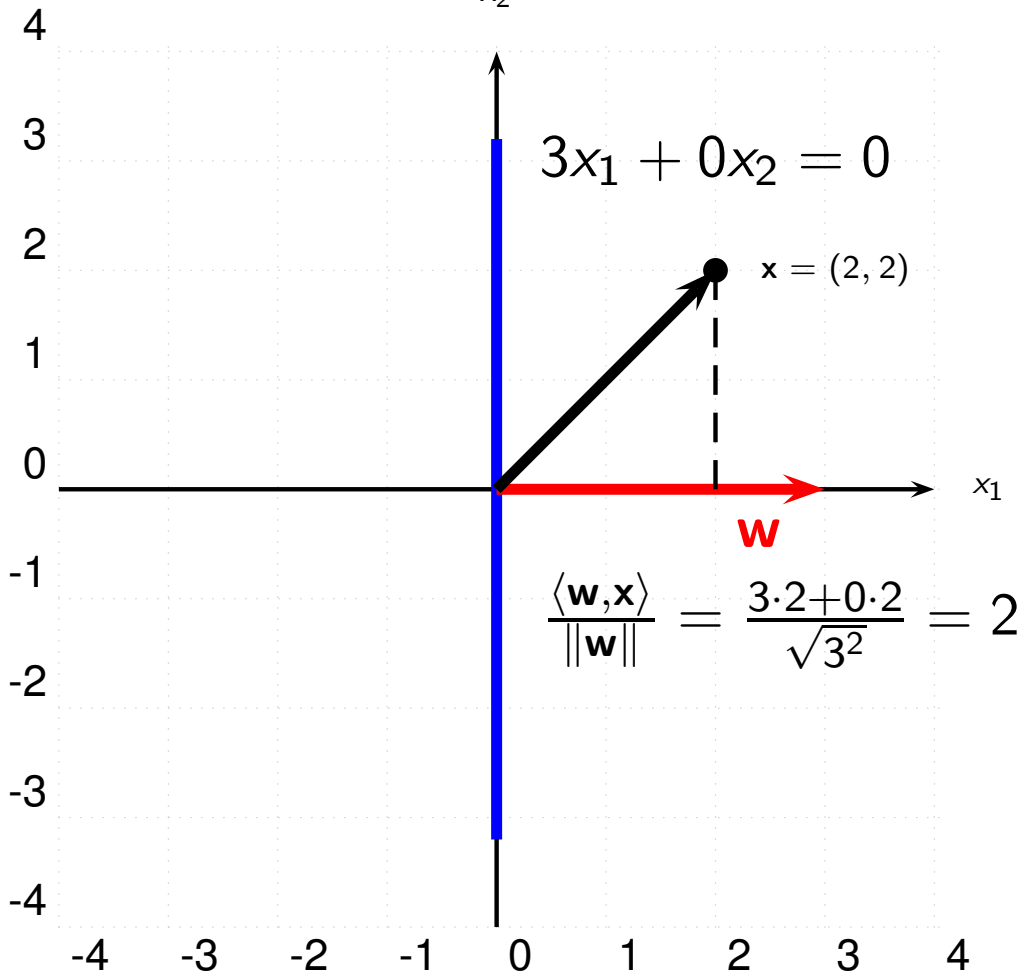
The dot product is defined as

$$w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \mathbf{w}^T \cdot \mathbf{x} \stackrel{def}{=} \langle \mathbf{w}, \mathbf{x} \rangle, \text{ for some } d \in \mathbb{N}.$$

In our example $d = 2$ and we obtain $-2 \cdot 1 + 1 \cdot 2 = 0$.

Linear Classifier & Dot Product (cont.)

Let us consider the *weight* vector $\mathbf{w} = (3, 0)$ and vector $\mathbf{x} = (2, 2)$.



Geometric interpretation of the dot product: Length of the projection of $\mathbf{x}$ onto the unit vector $\mathbf{w}/\|\mathbf{w}\|$.

Dot Product as a Similarity Measure

Dot product allows us to compute: lengths, angles and distances.
Length (norm):

$$\|\mathbf{x}\| = x_1 x_1 + x_2 x_2 + \ldots + x_d x_d = \langle \mathbf{x}, \mathbf{x} \rangle$$

Example: $\mathbf{x} = (1, 1, 1)$ we obtain $\|\mathbf{x}\| = \sqrt{1^2 + 1^2 + 1^2} = \sqrt{3}$
Angle:

$$\cos\alpha = \frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|\|\mathbf{x}\|} = \frac{w_1 x_1 + w_2 x_2 + \ldots + w_d x_d}{\sqrt{w_1^2 + w_2^2 + \ldots + w_d^2}\sqrt{x_1^2 + x_2^2 + \ldots + x_d^2}}$$

Example: $\mathbf{w} = (3, 0)$, $\mathbf{x} = (2, 2)$ we obtain

$$\cos\alpha = \frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|\|\mathbf{x}\|} = \frac{3 \cdot 2 + 0 \cdot 2}{\sqrt{3^2 + 0^2}\sqrt{2^2 + 2^2}} = \frac{2}{\sqrt{8}}$$

and obtain $\alpha = \cos^{-1}\left(\frac{2}{\sqrt{8}}\right) = 0.7853982$ and $0.7853982 \cdot 180/\pi = 45°$

Dot Product as a Similarity Measure (cont.)

Distance (Euclidean):

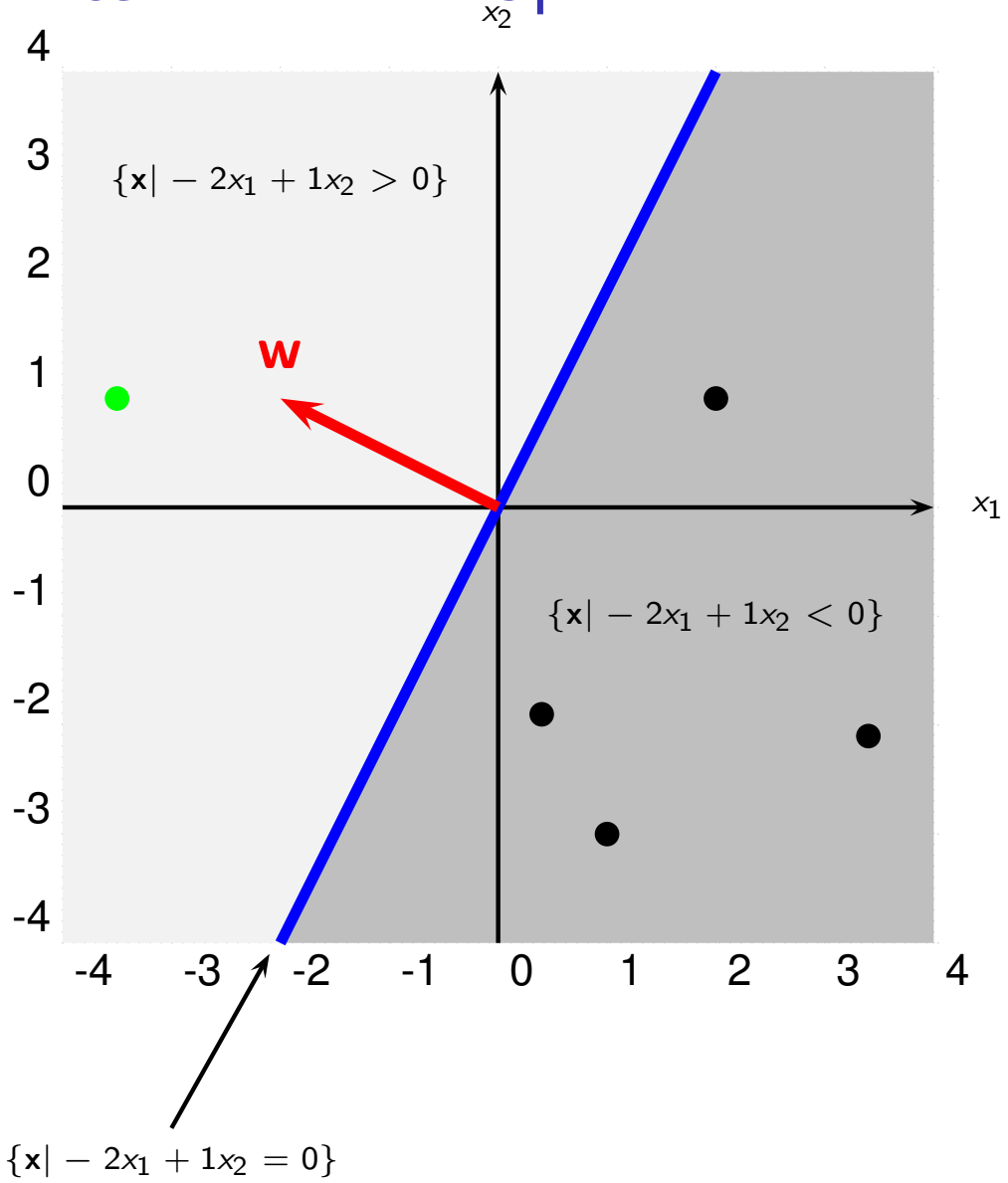
$$dist(\mathbf{w}, \mathbf{x}) = \|\mathbf{w} - \mathbf{x}\| = \sqrt{\langle \mathbf{w} - \mathbf{x}, \mathbf{w} - \mathbf{x} \rangle} = \sqrt{(w_1 - x_1)^2 + (w_2 - x_2)^2}$$

Example: $\mathbf{w} = (3, 0)$, $\mathbf{x} = (2, 2)$ we obtain

$$\|\mathbf{w} - \mathbf{x}\| = \sqrt{\langle \mathbf{w} - \mathbf{x}, \mathbf{w} - \mathbf{x} \rangle} = \sqrt{(3 - 2)^2 + (0 - 2)^2} = \sqrt{5}$$

Popular application in natural language processing: Dot product on text documents, in other words how similar are e.g. two given text documents.

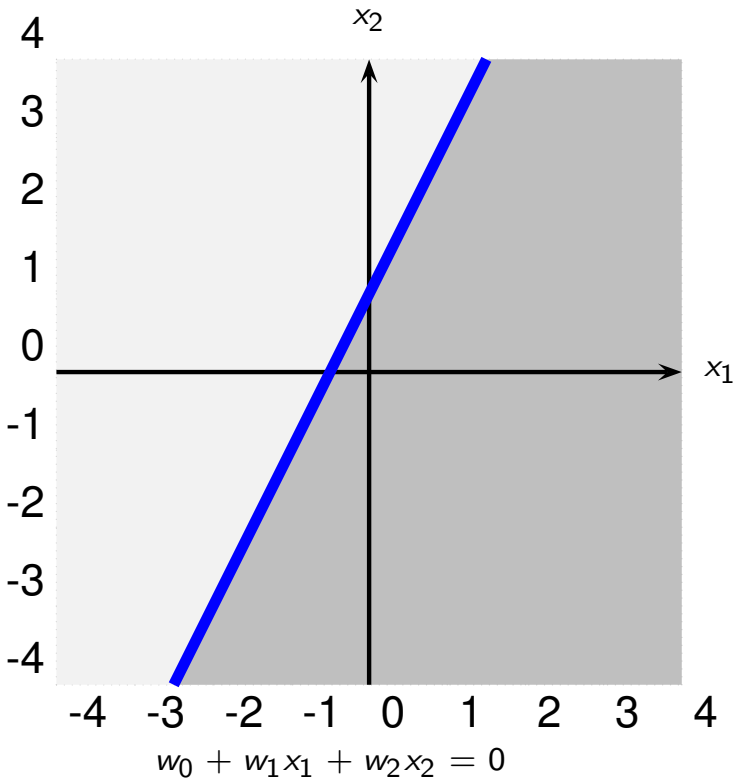
Linear Classifier & Two Half-Spaces

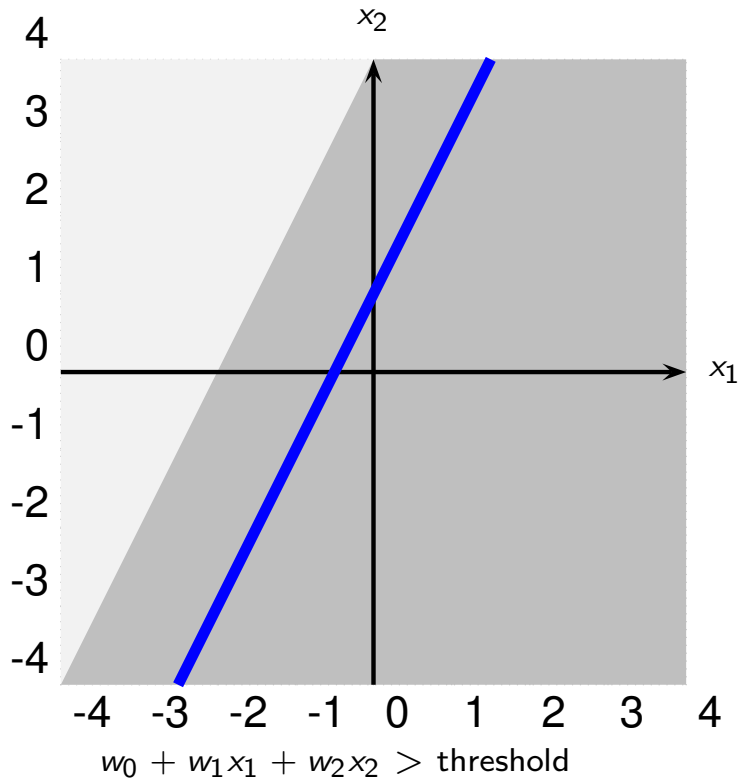


The **x**-space is separated in two half-spaces.

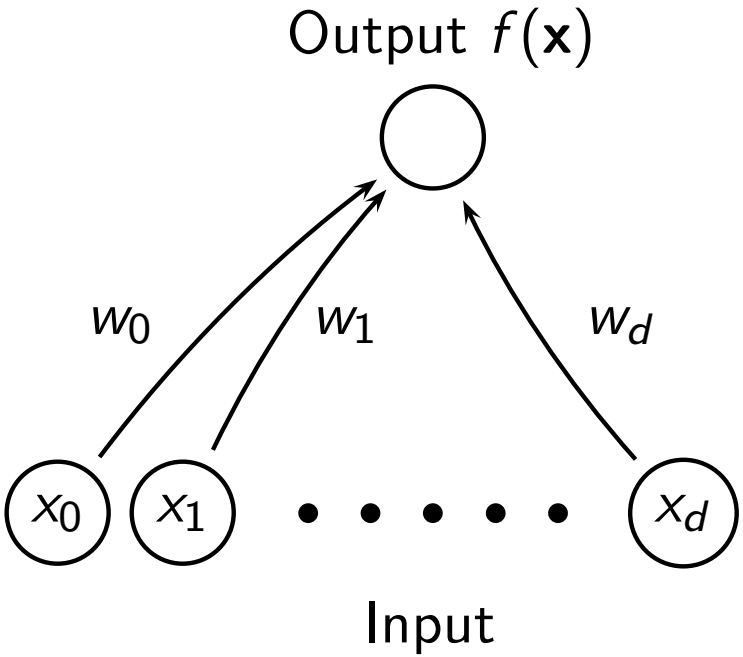
Linear Classifier & Dot Product (cont.)

- Observe, that $w_1 x_1 + w_2 x_2 = 0$ implies, that the separating line always goes through the origin.
- By adding an offset (bias), that is $w_0 + w_1 x_1 + w_2 x_2 = 0 \Leftrightarrow x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} \equiv y = mx + b$, one can shift the line arbitrary.

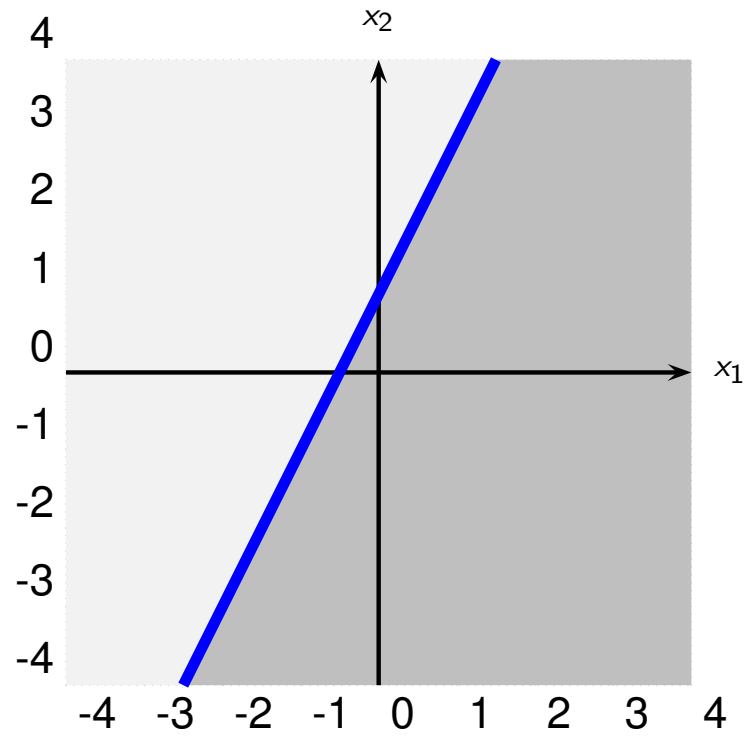




Linear Classifier & Single Layer NN



$\Leftrightarrow$



Note that $x_0 = 1$, $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$.

Given data which we want to separate, that is, a sample
$\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \in \mathbb{R}^{d+1} \times \{-1, +1\}$.

How to determine the proper values of $\mathbf{w}$ such that the "minus" and "plus" points are separated by $f(\mathbf{x})$? Infer the values of $\mathbf{w}$ from the data by some learning algorithm.

Perceptron

Note, so far we have not seen a method for finding the weight vector $\mathbf{w}$ to obtain a linearly separation of the training set.

Let $f(a)$ be (sign) activation function

$$f(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{if } a \geq 0 \end{cases}$$

and decision function

$$f(\langle \mathbf{w}, \mathbf{x} \rangle) = f\left(\sum_{i=0}^{d} w_i x_i \right).$$

Note: x_0 is set to $+1$, that is, $\mathbf{x} = (1, x_1, \ldots, x_d)$. Training pattern consists of $(\mathbf{x}, y) \in \mathbb{R}^{d+1} \times \{-1, +1\}$

Perceptron Learning Algorithm

input : $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^{d+1} \times \{-1, +1\}, \eta \in \mathbb{R}_+$, max.epoch $\in \mathbb{N}$

output: $\mathbf{w}$

begin

  Randomly initialize $\mathbf{w}$;

  epoch $\leftarrow 0$;

  repeat

    for $i \leftarrow 1$ **to** N **do**

      if $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \leq 0$ **then**

        $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i \, y_i$

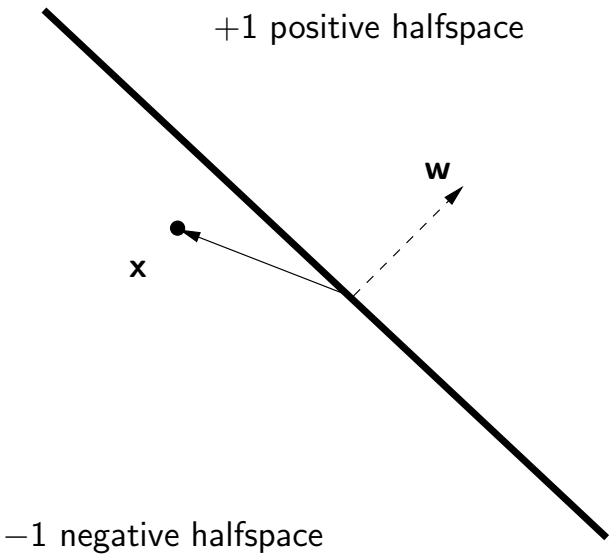
    epoch $\leftarrow$ epoch $+ 1$

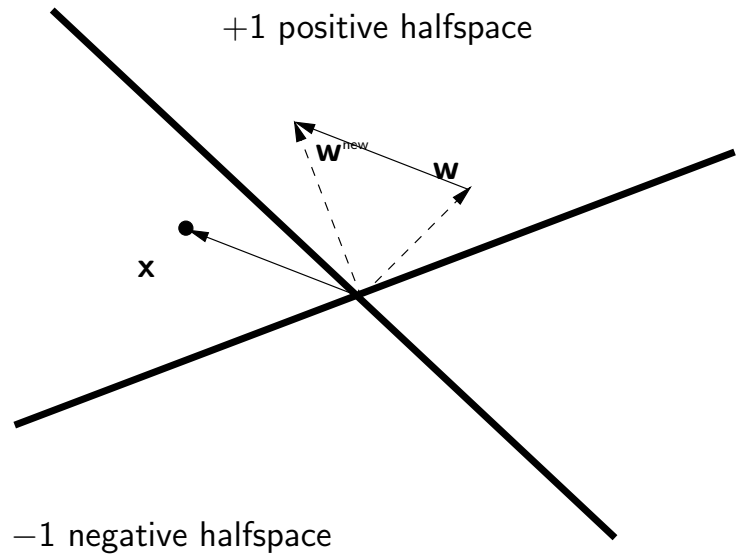
  until (*epoch* $=$ *max.epoch*) *or* (*no change in* $\mathbf{w}$);

  return $\mathbf{w}$

Training the Perceptron (cont.)

Geometrical explanation: If $\mathbf{x}$ belongs to $\{+1\}$ and $\langle \mathbf{w}, \mathbf{x} \rangle < 0 \Rightarrow$ angle between $\mathbf{x}$ and $\mathbf{w}$ is greater than 90°, rotate $\mathbf{w}$ in direction of $\mathbf{x}$ to bring missclassified $\mathbf{x}$ into the positive half space defined by $\mathbf{w}$. Same idea if $\mathbf{x}$ belongs to $\{-1\}$ and $\langle \mathbf{w}, \mathbf{x} \rangle \geq 0$.





Perceptron Error Reduction

Recall: missclassifcation results in:

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \mathbf{x}\, y,$$

this reduces the error since[1]

$$\begin{aligned} -\mathbf{w}_{\text{new}}^T(\mathbf{x}\, y) &= -\mathbf{w}^T(\mathbf{x}\, y) - \underbrace{\eta}_{>0}\underbrace{(\mathbf{x}\, y)^T(\mathbf{x}\, y)}_{\|\mathbf{x}y\|^2>0} \\ &< -\mathbf{w}^T\mathbf{x}y \end{aligned}$$

How often one has to cycle through the patterns in the training set?

- A finite number of steps?

[1] right multiply with $-(\mathbf{x}\, y)$ and transpose term before

Perceptron Convergence Theorem

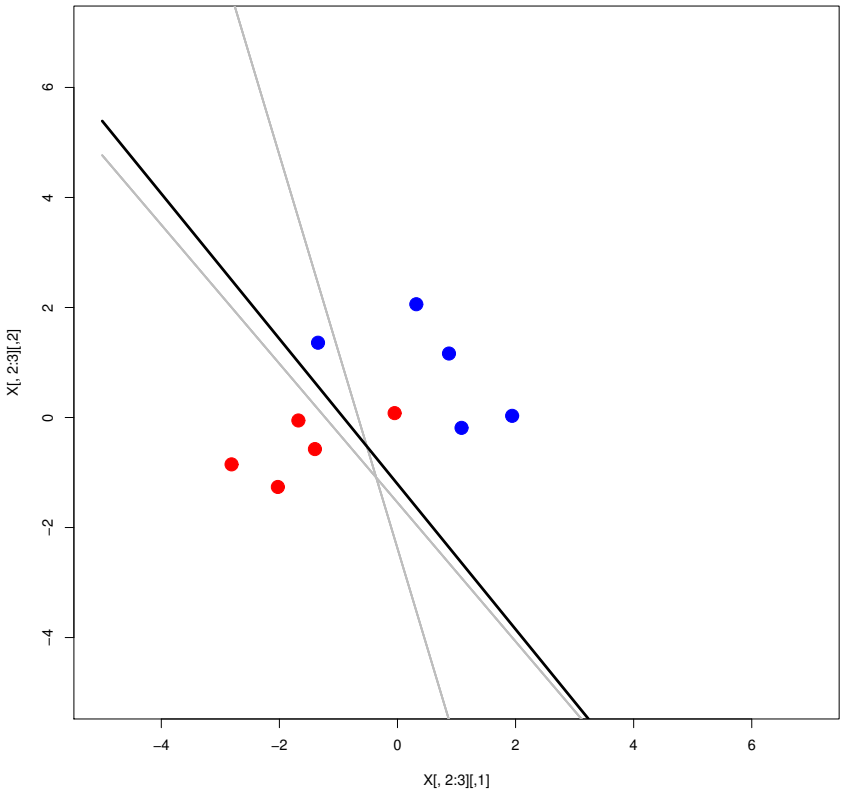
Proposition

Given a finite and linearly separable training set. The perceptron converges after some finite steps [Rosenblatt, 1962].

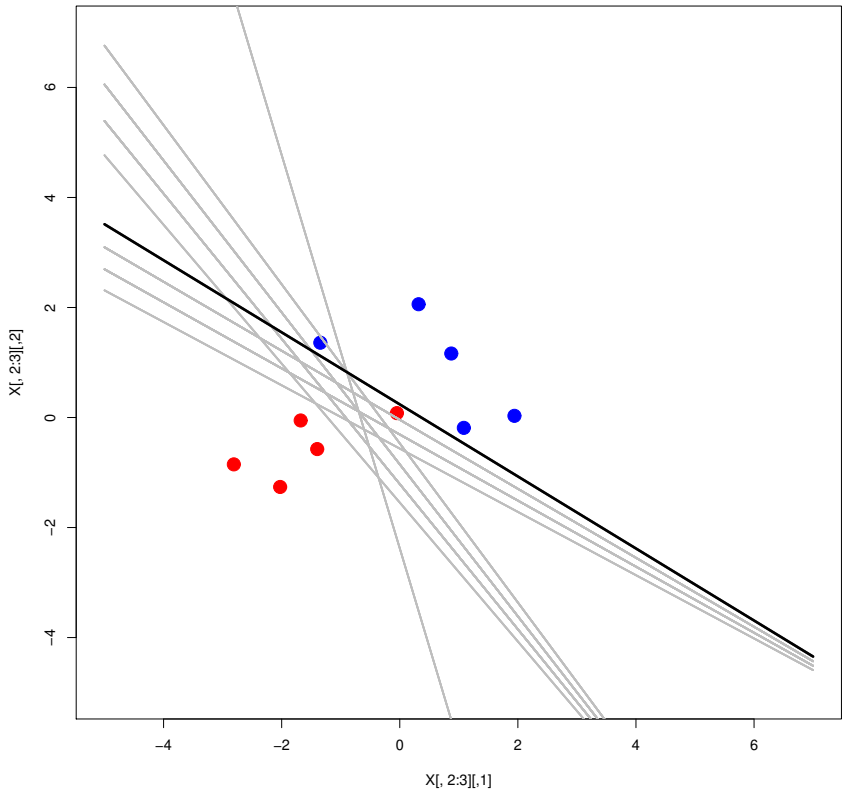
Perceptron Algorithm (R-code)

```
##################################################
perceptron <- function(w,X,y,eta,max.epoch) {
##################################################
  N <- nrow(X);
  epoch <- 0;
  repeat {
    w.old <- w;
    for (i in 1:N) {
      if ( y[i] * (X[i,] %*% w) <= 0 )
        w <- w + eta * y[i] * X[i,];
    }
    epoch <- epoch + 1;
    if ( identical(w.old,w) || epoch = max.epoch ) {
      break; # terminate if no change in weights or max.epoch
    }
  }
  return (w);
```

Perceptron Algorithm Visualization

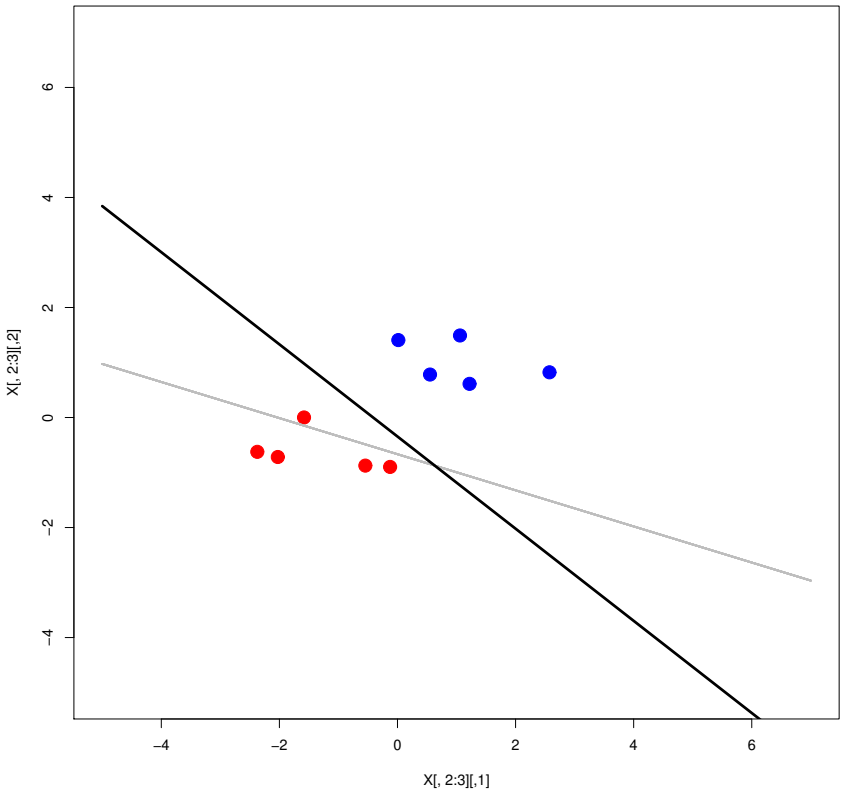


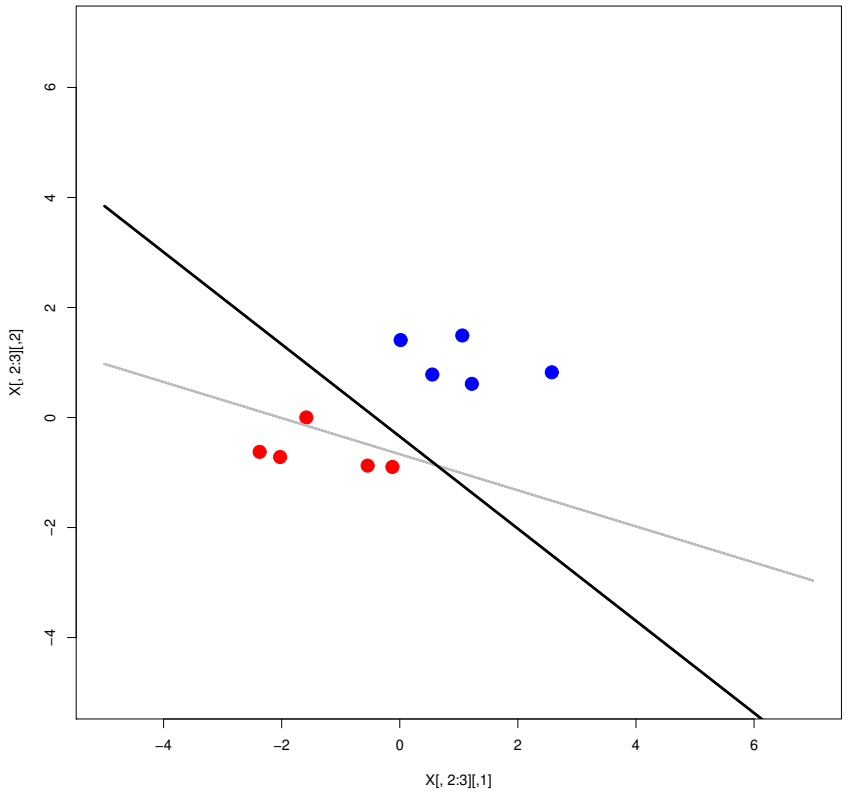
One epoch



terminate if no change in **w**

Perceptron Algorithm Visualization





One epoch

terminate if no change in **w**

From Perceptron Loss_Θ to Gradient Descent

The parameters to learn are: $(w_0, w_1, w_2) = \mathbf{w}$.

- What is our loss function Loss_Θ we would like to minimize?
- Where is term $\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \mathbf{x}\, y$ coming from?

$$\text{Loss}_\Theta \mathrel{\widehat{=}} E(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \langle \mathbf{w}, \mathbf{x}_m \rangle y_m$$

where $\mathcal{M}$ denotes the set of all missclassified patterns. Moreover, Loss_Θ is *continuous* and *piecewise linear* and fits in the spirit iterative *gradient descent* method

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \nabla E(\mathbf{w}) = \mathbf{w} + \eta \mathbf{x}\, y$$

Method of Gradient Descent

Let $E(\mathbf{w})$ be a continuously differentiable function of some unknown (weight) vector $\mathbf{w}$.

Find an optimal solution $\mathbf{w}^\star$ that satisfies the condition

$$E(\mathbf{w}^\star) \leq E(\mathbf{w}).$$

The necessary condition for optimality is

$$\nabla E(\mathbf{w}^\star) = \mathbf{0}.$$

Let us consider the following *iterative* descent:

Start with an initial guess $\mathbf{w}^{(0)}$ and generate sequence of weight vectors $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots$ such that

$$E(\mathbf{w}^{(i+1)}) \leq E(\mathbf{w}^{(i)}).$$

Gradient Descent Algorithm

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla E(\mathbf{w}^{(i)})$$

where η is a positive constant called learning rate.

At each iteration step the algorithm applies the correction

$$\begin{aligned} \Delta \mathbf{w}^{(i)} &= \mathbf{w}^{(i+1)} - \mathbf{w}^{(i)} \\ &= -\eta \nabla E(\mathbf{w}^{(i)}) \end{aligned}$$

Gradient descent algorithm satisfies:

$$E(\mathbf{w}^{(i+1)}) \leq E(\mathbf{w}^{(i)}),$$

to see this, use first-order Taylor expansion around $\mathbf{w}^{(i)}$ to approximate $E(\mathbf{w}^{(i+1)})$ as $E(\mathbf{w}^{(i)}) + (\nabla E(\mathbf{w}^{(i)}))^T \Delta \mathbf{w}^{(i)}$.

Gradient Descent Algorithm (cont.)

$$\begin{aligned} E(\mathbf{w}^{(i+1)}) &\approx E(\mathbf{w}^{(i)}) + (\nabla E(\mathbf{w}^{(i)}))^T \Delta \mathbf{w}^{(i)} \\ &= E(\mathbf{w}^{(i)}) - \eta \|\nabla E(\mathbf{w}^{(i)})\|^2 \end{aligned}$$

For positive learning rate η, $E(\mathbf{w}^{(i)})$ decreases in each iteration step (for small enough learning rates).
At minimum/saddle point gradient vector is $\mathbf{0}$, thus no change in weight.
Example:

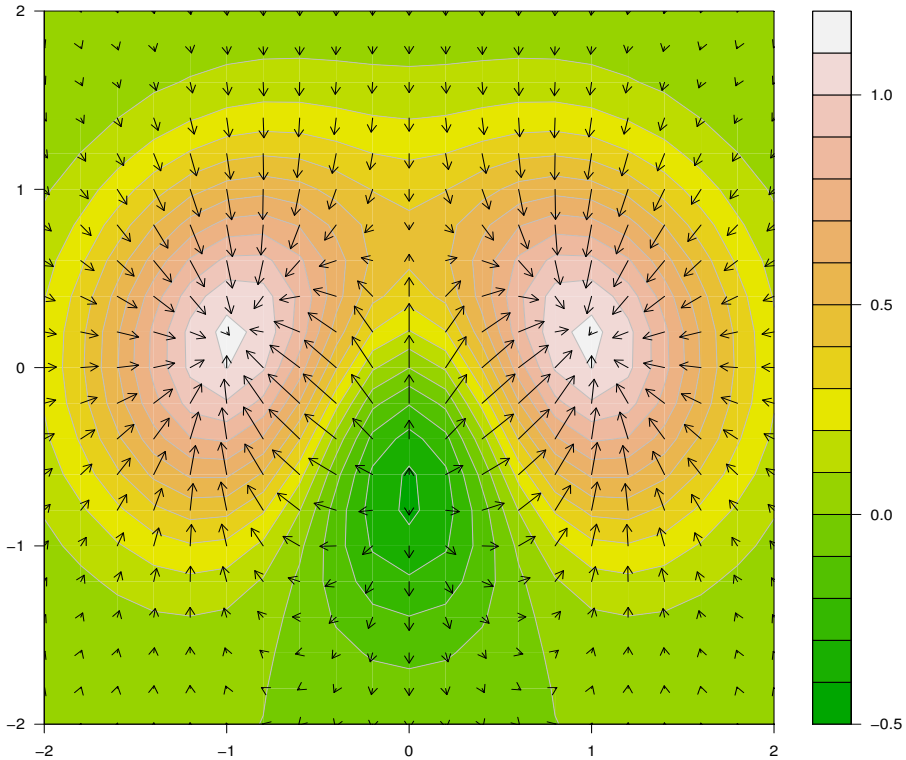
$$f(x, y) = (3x^2 + y)\exp(-x^2 - y^2)$$

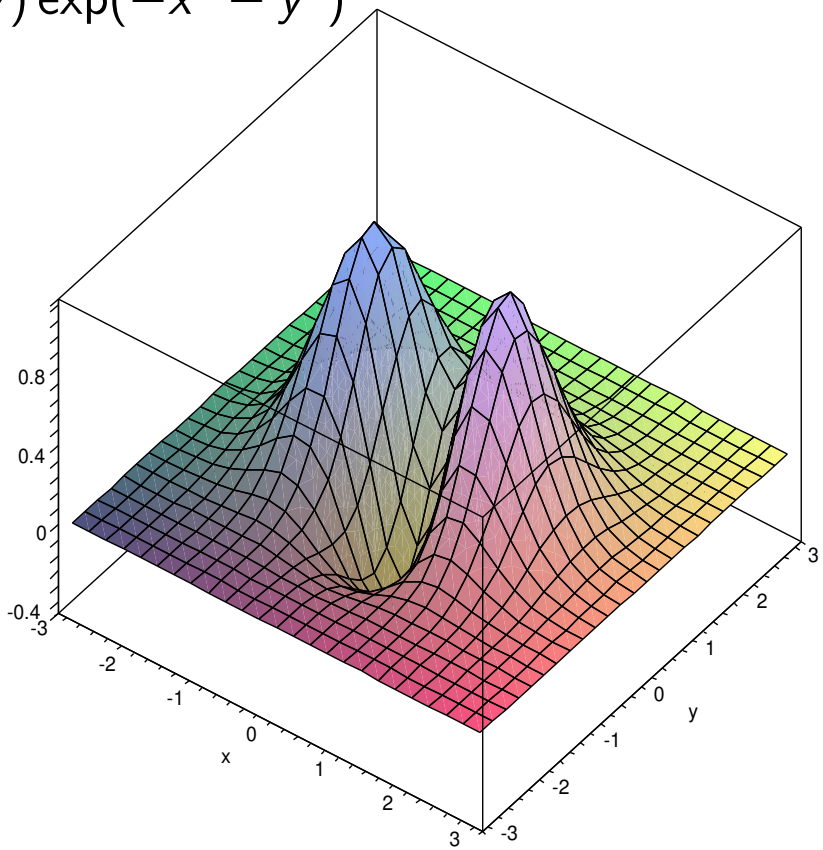
Partial derivatives:

$$\begin{aligned} \frac{\partial f}{\partial x} &= -2x\exp(-x^2 - y^2)(3x^2 + y - 3) \\ \frac{\partial f}{\partial y} &= \exp(-x^2 - y^2)(-6x^2 y - 2y^2 + 1) \end{aligned}$$

Gradient Descent Algorithm (cont.)

$$(3x^2 + y)\exp(-x^2 - y^2)$$

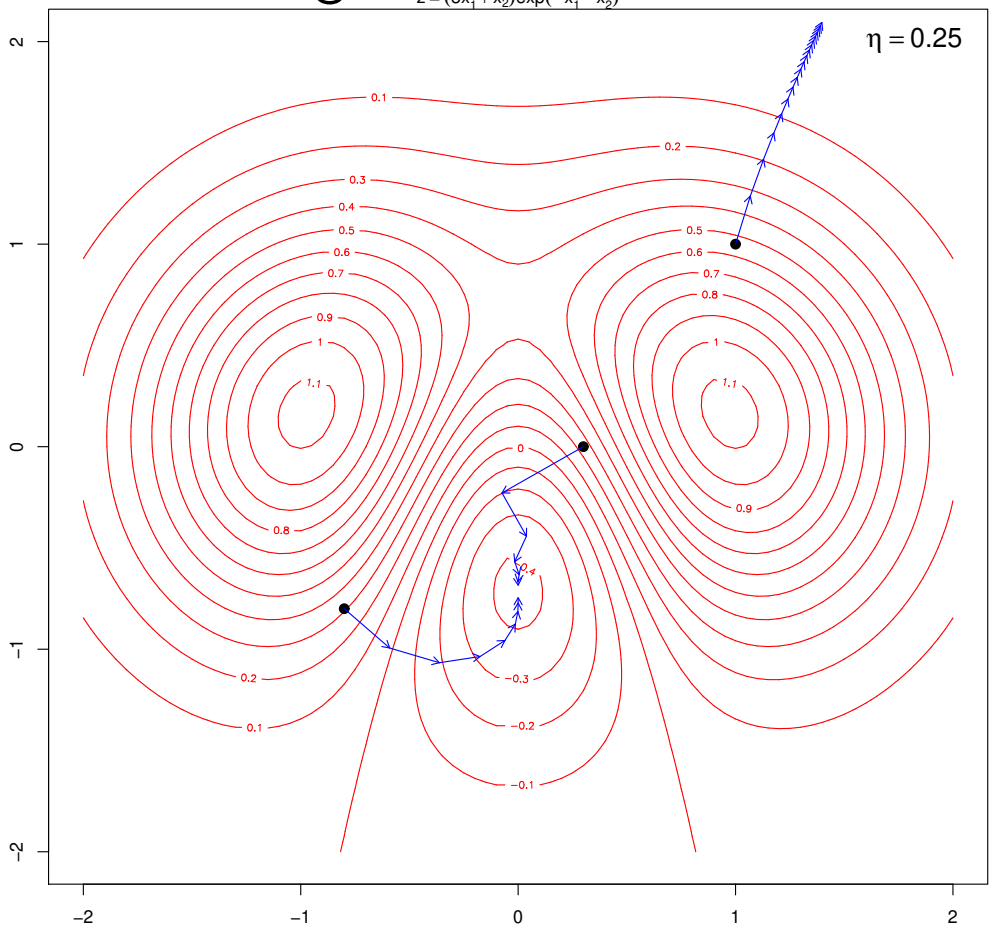




See interactive demo.

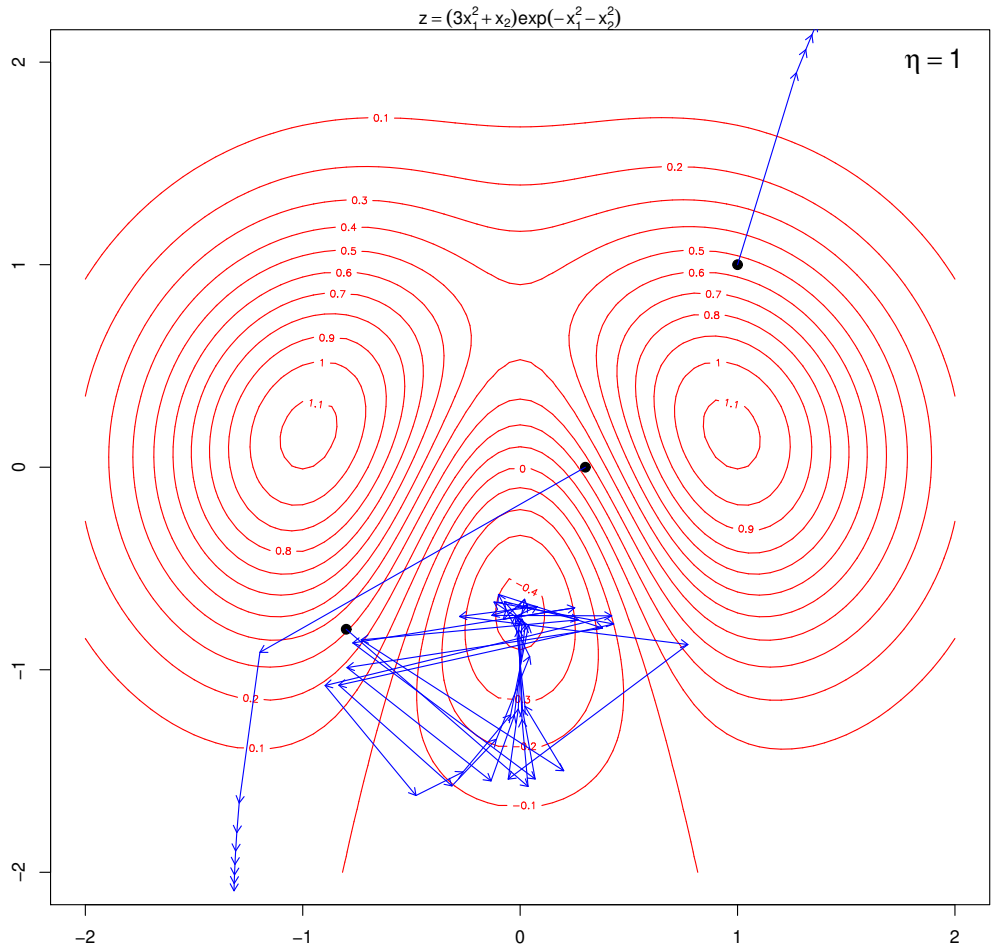
Gradient Descent Algorithm Example

Black points denote different starting values. Learning rate η is properly chosen, however for starting value $(1, 1)$, algorithm converges not to the global minimum. It follows steepest descent in the "wrong direction", in other words, gradient based algorithms are local search algorithms.



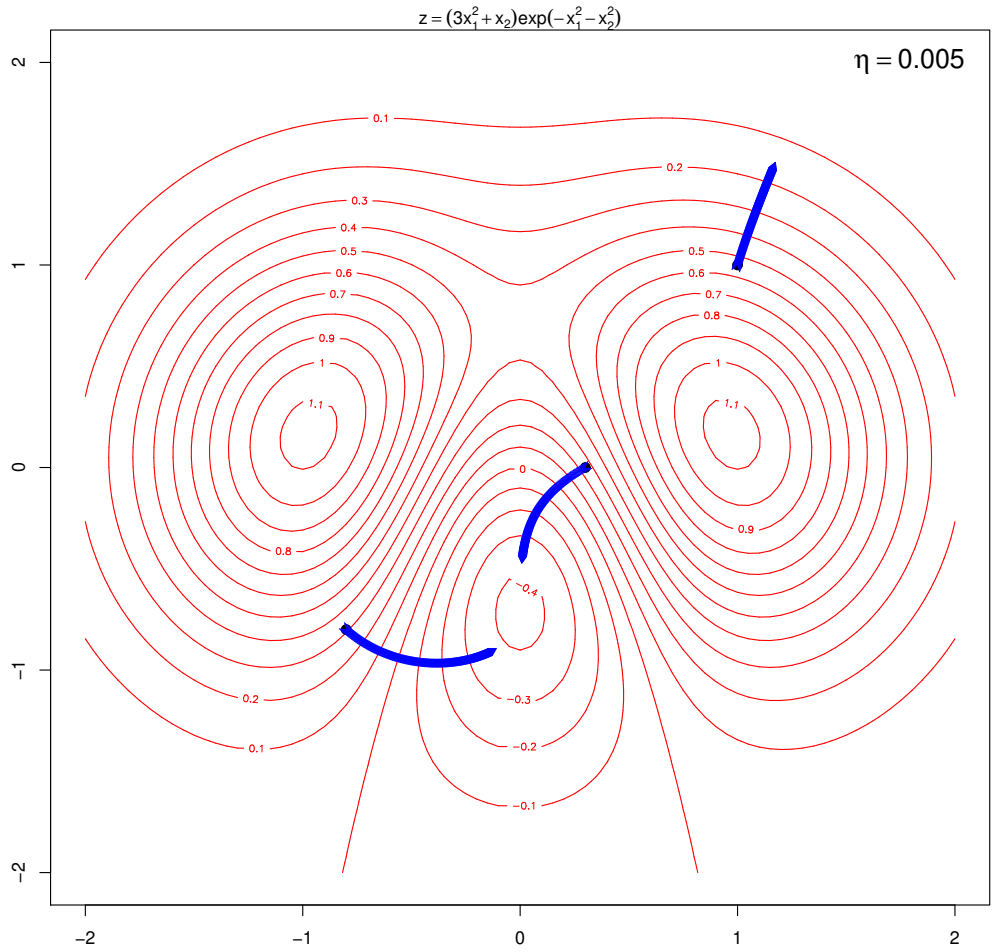
Gradient Descent Algorithm Example (cont.)

Learning rate $\eta = 1.0$ is too large, algorithm oscillates in a "*zig-zag*" manner or "*overleap*" the global minimum.



Gradient Descent Algorithm Example (cont.)

Learning rate $\eta = 0.005$ is too small, algorithm converges "very slowly".



Momentum

- Gradient descent can be very slow if η is too small, and can oscillate widely if η is too large.
- Idea: use fraction of the previous weight change and actual gradient term to control non-radical revisions in the updates.

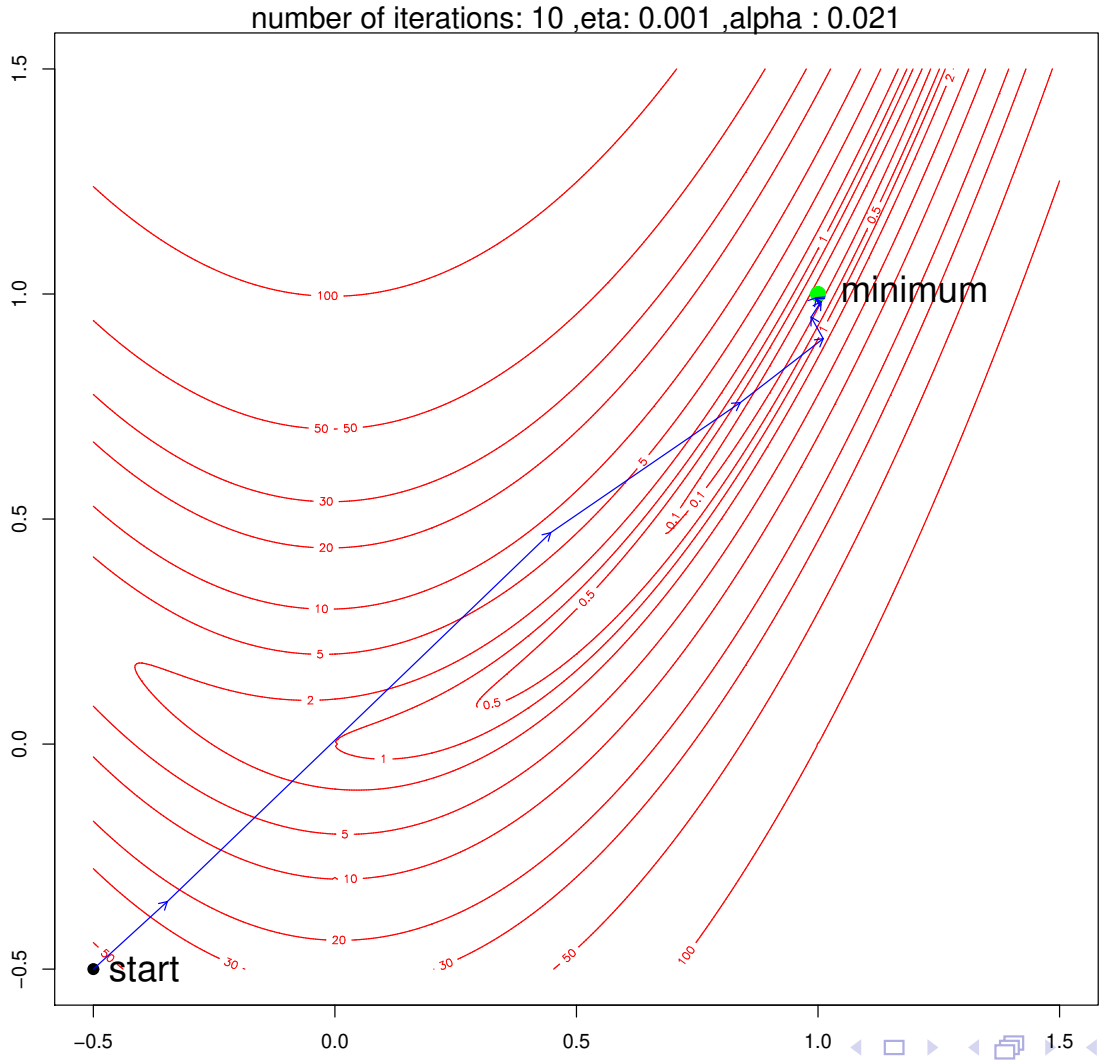
$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla E(\mathbf{w}^{(i)}) + \alpha \mathbf{w}^{(i-1)}, \qquad 0 \leq \alpha \leq 1.$$

Momentum:

- can cancel side-to-side oscillations across the error valley,
- can cause a faster convergence when weight updates are all in the same direction because the learning rate is amplified.

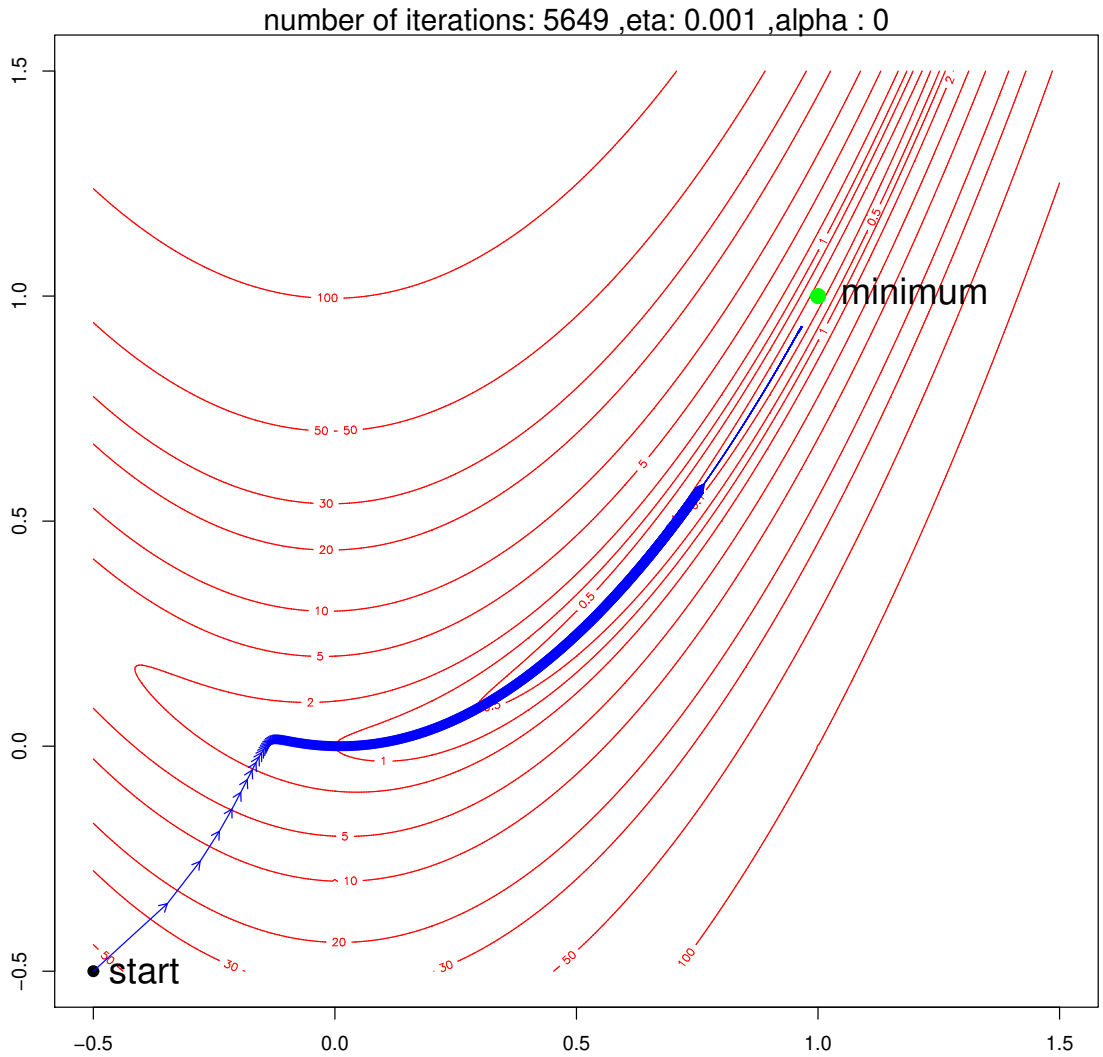
Momentum Example Rosenbrock Function

Rosenbrock function $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$ has global minimum $f(x, y) = (0, 0)$ at $(1, 1)$. Momentum param. $\alpha = 0.021$, learning rate $\eta = 0.001$.



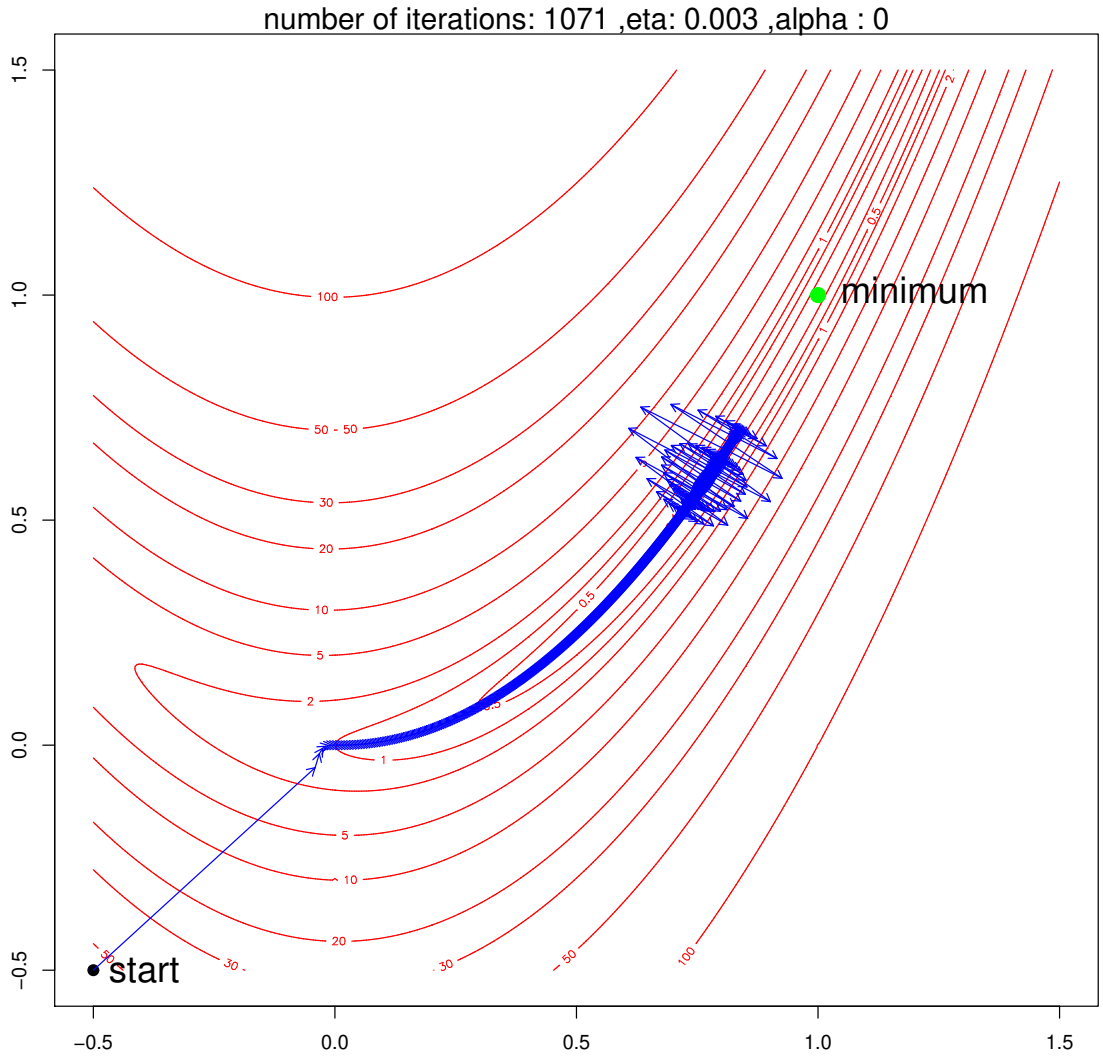
Momentum Example Rosenbrock Func. (cont.)

Setting $\alpha = 0$ (no momentum)



Momentum Example Rosenbrock Func. (cont.)

Setting $\alpha = 0$ (no momentum) and a larger learning rate η



Sophisticated Gradient Descent

Note, gradient descent is the building block for much more sophisticated gradient descent methods such as

- RMSProp
- Adagrad
- Adadelta
- NAG
- Nadam

These are leveraging *adaptive* learning rate η to speedup convergence.

See: An overview of gradient descent optimization algorithms, S. Ruder