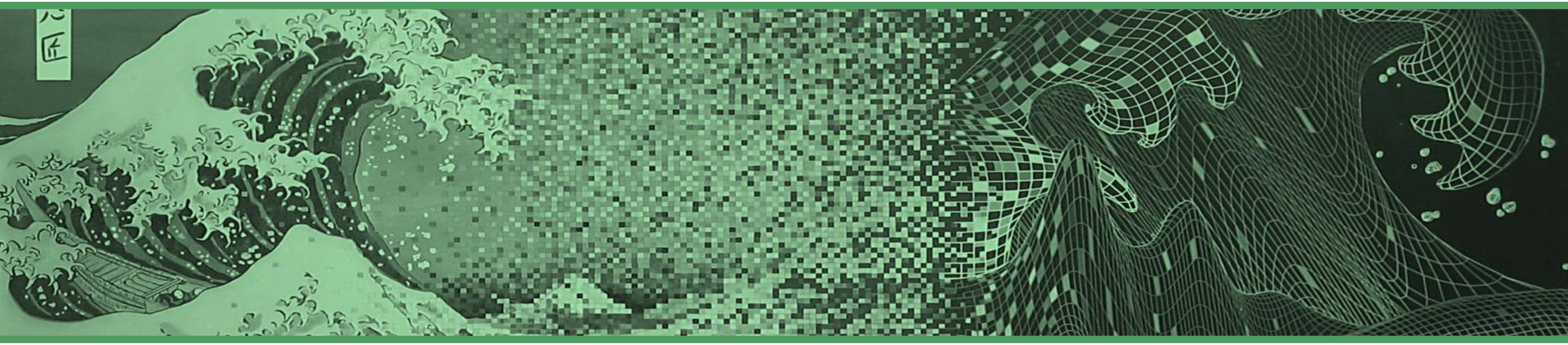
Perceptron

Gradient descent



Aprendizagem 2023

Rui Henriques rmch@tecnico.ulisboa.pt
Andreas Wischert andreas.wichert@tecnico.ulisboa.pt

Outline

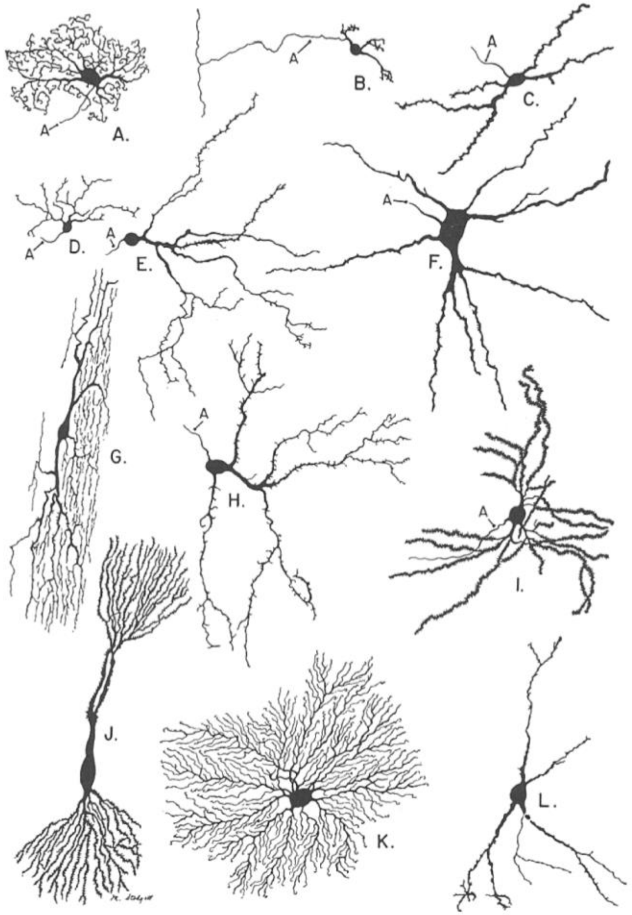
- **Perceptron**
  - origins
  - learning rule
  - limitations
  - activation functions
- **Gradient descent**
- **Perceptron revised:** inferring update rule
- **Logistic regression**
- **Stochastic gradient descent**

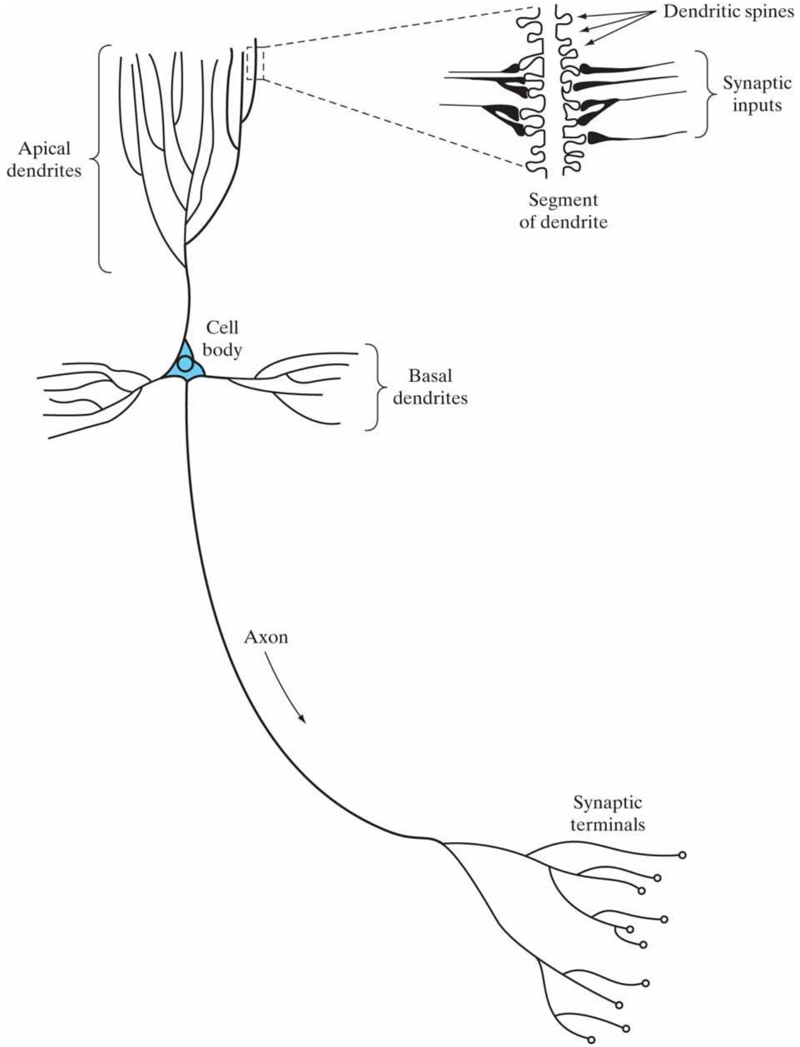
Outline

- **Perceptron**
  - **origins**
  - **learning rule**
  - **limitations**
  - **activation functions**
- Gradient descent
- Perceptron revised: inferring update rule
- Logistic regression
- Stochastic gradient descent

Brain

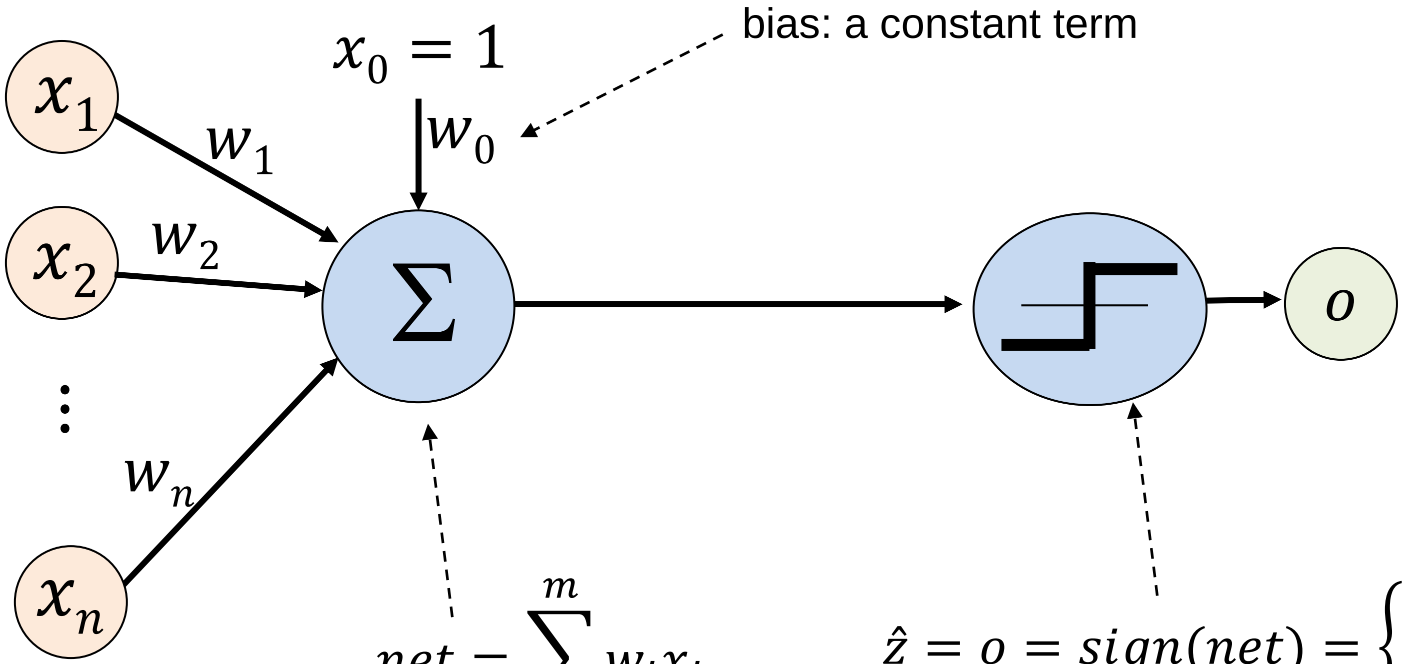
- Neuron
- Complex connectome
  - 10^{40} neurons
  - $10^{4\text{-}5}$ connections per neuron
- Neuroplasticity
  - learning and memory





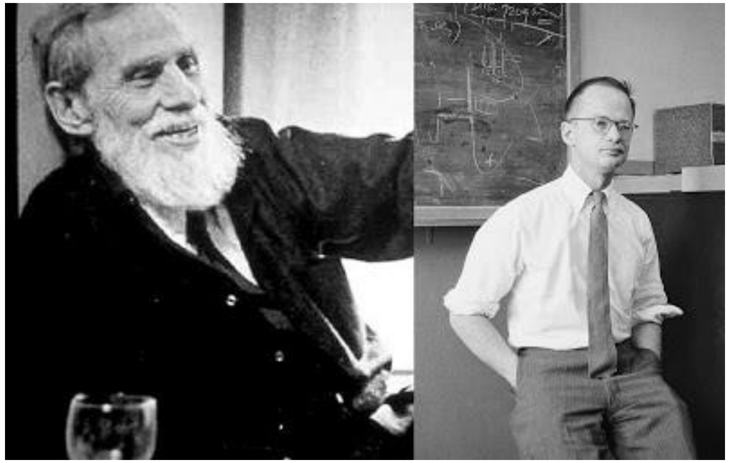
Perceptron (1957)

- Simulates a biological neuron



$$net = \sum_{j=0}^{m} w_j x_j$$

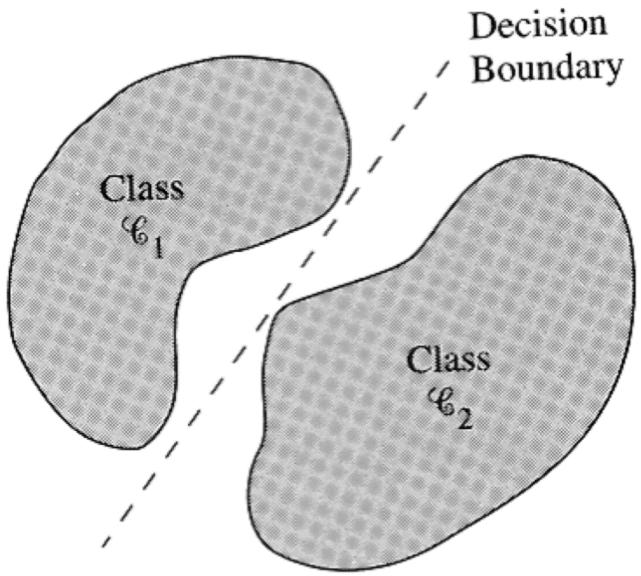
$$\hat{z} = o = sign(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ -1 & \text{if } net < 0 \end{cases}$$



McCulloch-Pitts

Perceptron

- *Perceptron goal*: correctly classify an observation $\mathbf{x} = \{x_1, x_2, ..x_m\}$ into one of the classes c_1 and c_2
  - the output for class c_1 is $t = 1$ and for c_2 is $t = -1$ (hereafter we use t and z interchangeably)
  - example for $m = 2$

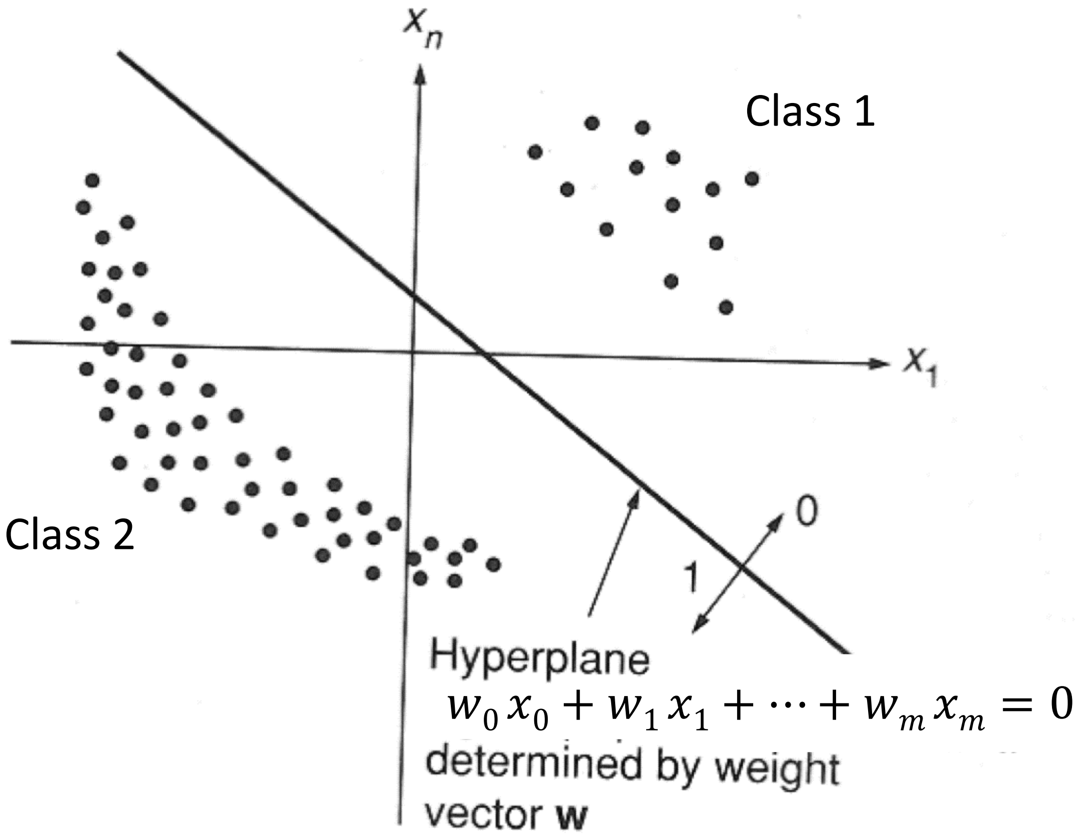


$$o = f(\mathbf{x}) = sign\left(\sum_{j=0}^{m} w_j x_j\right)$$

with $\sum_{j=0}^{m} w_j x_j = \langle \mathbf{x} | \mathbf{w} \rangle = \mathbf{x}^T \mathbf{w} = \mathbf{w}^T \mathbf{x} = \langle \mathbf{w} | \mathbf{x} \rangle$

where $x_0 = 1$ (bias)

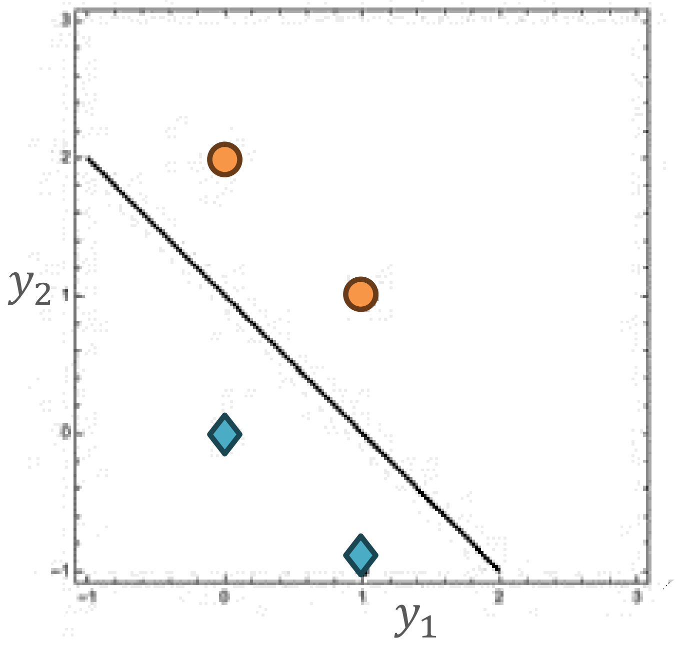
Linearly separable patterns



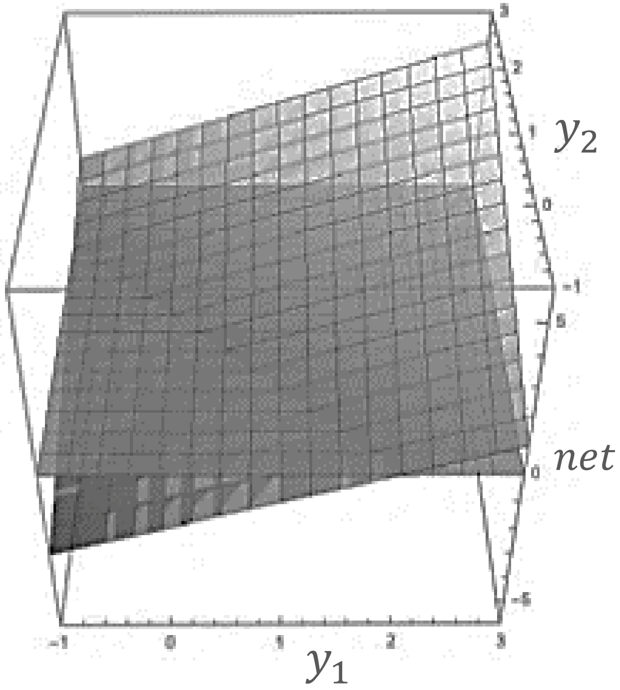
$$\sum_{j=0}^{m} w_j x_j > 0, \qquad \text{for } c_1$$

$$\sum_{j=0}^{m} w_j x_j \leq 0, \qquad \text{for } c_2$$

Linearly separable patterns

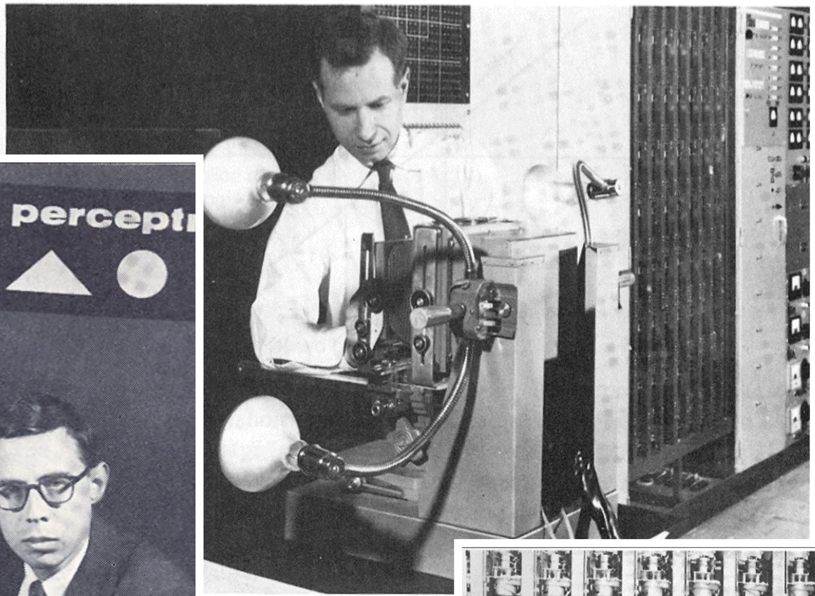


(a) data points separated by the line $-1 + x_1 + x_2 = 0$ (change in $\mathrm{sign}(-1 + x_1 + x_2)$)

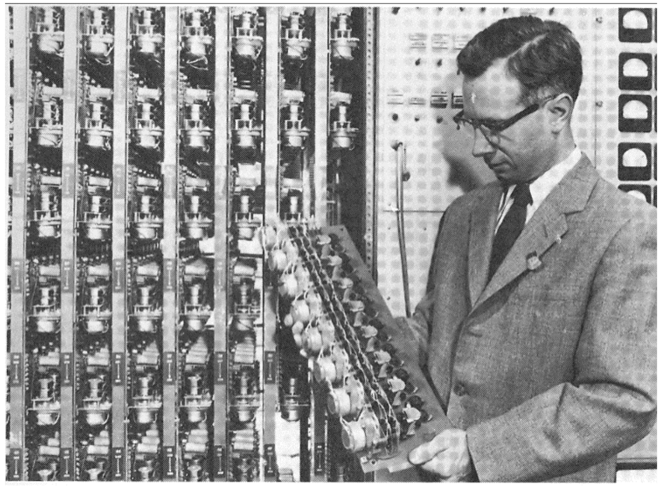


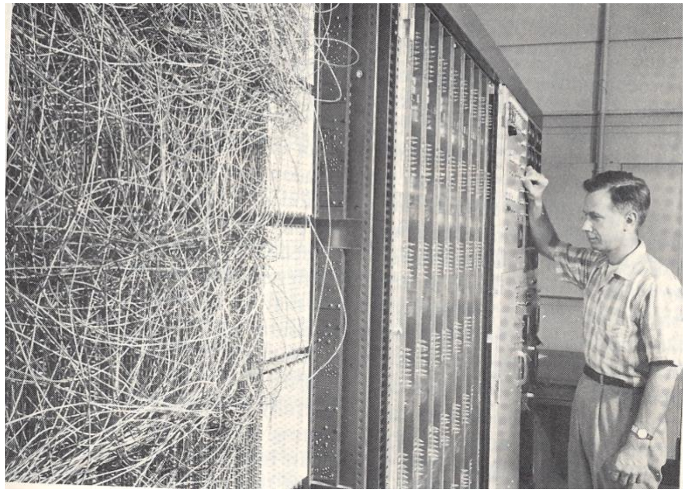
(b) without activation, the hyperplane $-1 + x_1 + x_2 = net$ also separates the input observations

Frank Rosenblatt (1928-1971)



- Rosenblatt's bitter rival and professional nemesis was Marvin Minsky of Carnegie Mellon University
- Minsky despised Rosenblatt, hated the concept of the perceptron, and wrote several polemics against him
- For years Minsky crusaded against Rosenblatt on a very nasty and personal level, denouncing him as a charlatan, hoping to cut all funding for his research in neural nets





Perceptron learning rule

- Consider linearly separable problems
- How to find appropriate weights?
  - supervised learning: assess network against correct answers
  - look if the output o (+1,-1) belongs to the expected target t
    - $(t - o)$ plays the role of the error signal
  - perceptron learning rule

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = \eta \cdot (t - o) \cdot \mathbf{x}$$

  - η is called the learning rate
  - $0 < \eta \leq 1$

Perceptron learning rule

- Rosenblatt's perceptron algorithm
  - Initialize all weights w_i with random values
  - initialize $\eta \in]0,1]$, $\text{iterations} = 0$ and max_iterations
  - Until there is no change in weights or $\text{iterations} > \text{max\_iterations}$
    - choose a observation $\mathbf{x}_i$ out of training data
    - compute $o_i = sign\left(\sum_{j=0}^{m} w_j x_{ij}\right)$
    - compute $\Delta w_j = \eta \cdot (t_i - o_i) \cdot x_{ij}$
    - update the weights $w_j = w_j + \Delta w_j$
    - iterations++

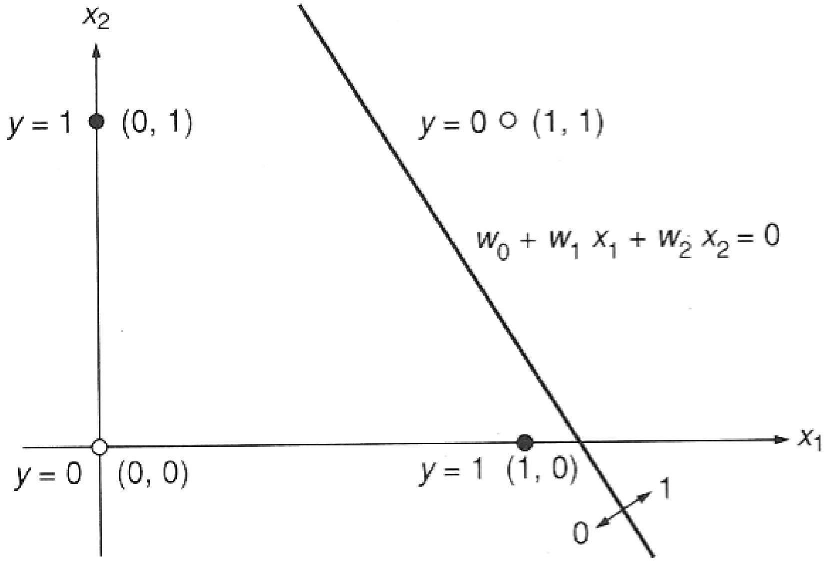
Perceptron learning

- The algorithm converges to the correct classification
  - if the training data is linearly separable
  - η is sufficiently small

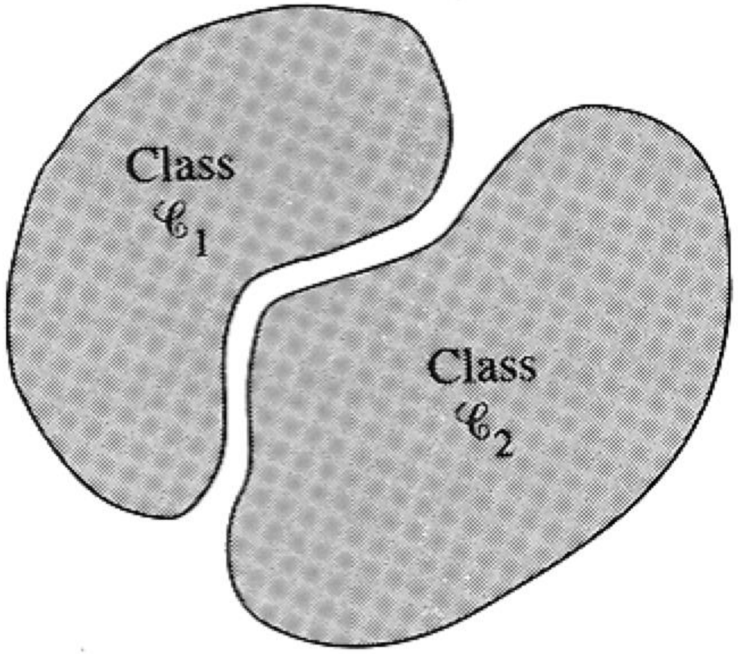
- When assigning a value to η we must keep in mind two conflicting requirements:
  - averaging past inputs to provide stable weights estimates, which requires small η
  - fast adaptation with respect to real changes in the underlying distribution of the process responsible for the generation of the input vector $\mathbf{x}$, which requires large η

Perceptron limitations

- The **XOR** problem!
  - shown by Minsky and Papert, 1960



Linearly separable?



- What about **convergence** of Rosenblatt algorithm?
  - all updates with same strength, $\eta \cdot (t_i - o_i)$, irrespectively of how close or far wrongly classified observations are from the hyperplane – **slow** and **unstable**!

Outline

- Perceptron
  - origins
  - learning rule
  - limitations
  - activation functions
- **Gradient descent**
- Perceptron revised: inferring update rule
- Logistic regression
- Stochastic gradient descent

Gradient

- To understand how can we improve the perceptron learning (*fast convergence* to good weights), we first need to introduce a new concept…
- Consider a continuously differentiable function $f(\mathbf{x})$

$$f : \mathbb{R}^D \rightarrow \mathbb{R} : f(\mathbf{x}) = z$$

  – mapping the observation $\mathbf{x}$ into a real value
- The **gradient** operator for $\mathbf{x}$ in $f(\mathbf{x})$ is

$$\nabla = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \cdots, \frac{\partial}{\partial x_D}\right]^T$$

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_D} \end{pmatrix}$$

Gradient

- Some properties of the gradient:

$$\nabla\big(a \cdot f(\mathbf{x}) + b \cdot g(\mathbf{x})\big) = a \cdot \nabla f(\mathbf{x}) + b \cdot \nabla g(\mathbf{x}), \qquad a, b \in \mathbb{R}$$

$$\nabla_x\left(\mathbf{y}^T\mathbf{x}\right) = \mathbf{y}$$

$$\nabla\left(\mathbf{x}^T \cdot A \cdot \mathbf{x}\right) = (A + A^T) \cdot \mathbf{x}$$

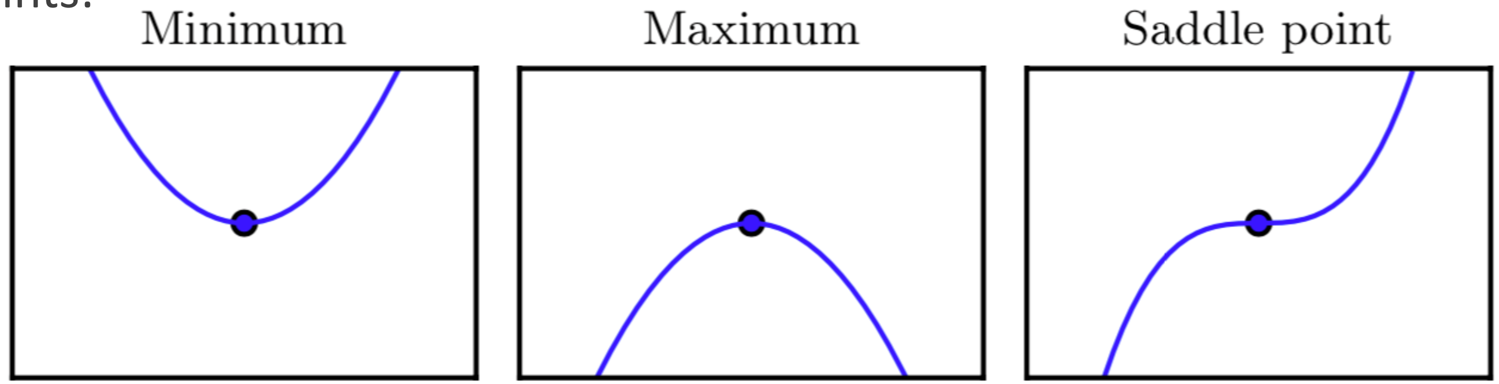
  – if A is a symmetric matrix

$$A^T = A$$

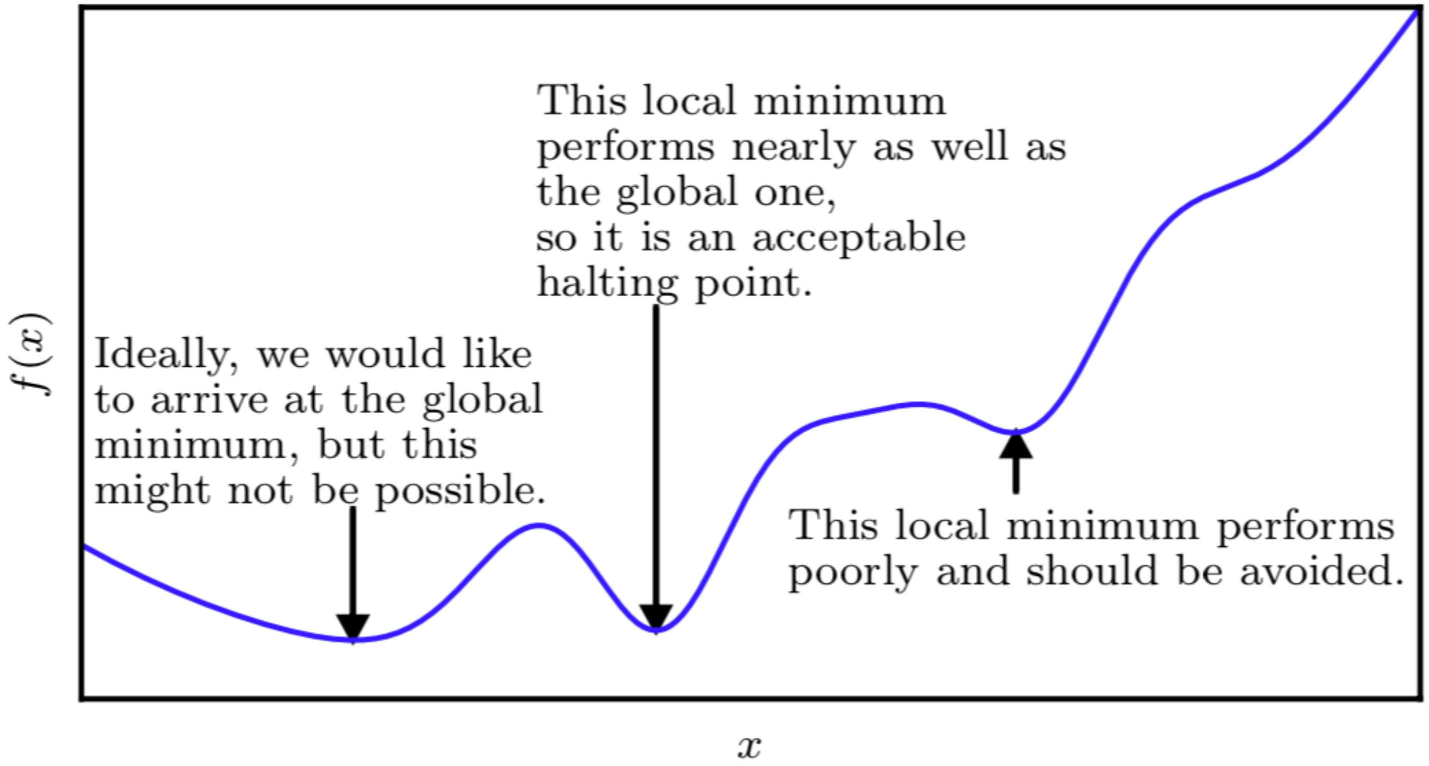
$$\nabla\left(\mathbf{x}^T \cdot A \cdot \mathbf{x}\right) = 2 \cdot A \cdot \mathbf{x}$$

Numerical solution

- Quest: answer **machine learning as an optimization problem**
  - use the gradient to infer the parameters of a descriptive or predictive model, examples:
    - **weights** of a **linear regression** (regression task)
    - **weights** of a **perceptron** (classification)
- Let us consider a univariate case, given a continuously differentiable function $z = f(x)$
  - the function has extreme points for $0 = f'(x)$
  - we can determine solutions numerically by approximating the real zeros $f'(x) = 0$
  - critical points:

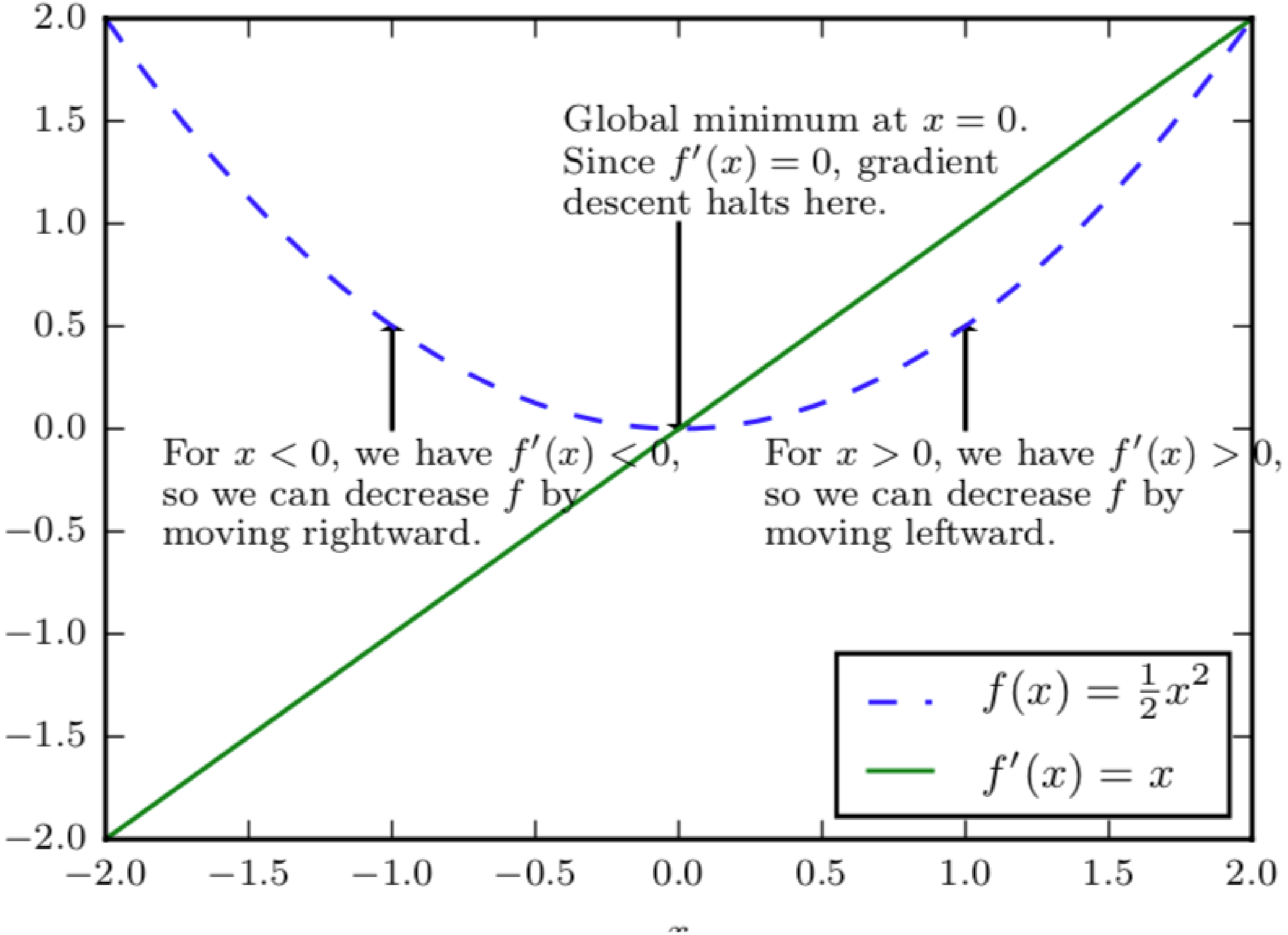


Approximate optimization



- Given a particular point x, how to move to a better state?

Gradient descent for one dimension



Gradient descent for one dimension

- Gradient descent for 1 dimension: $$x_{n+1} = x_n - \eta \cdot f'(x_n)$$

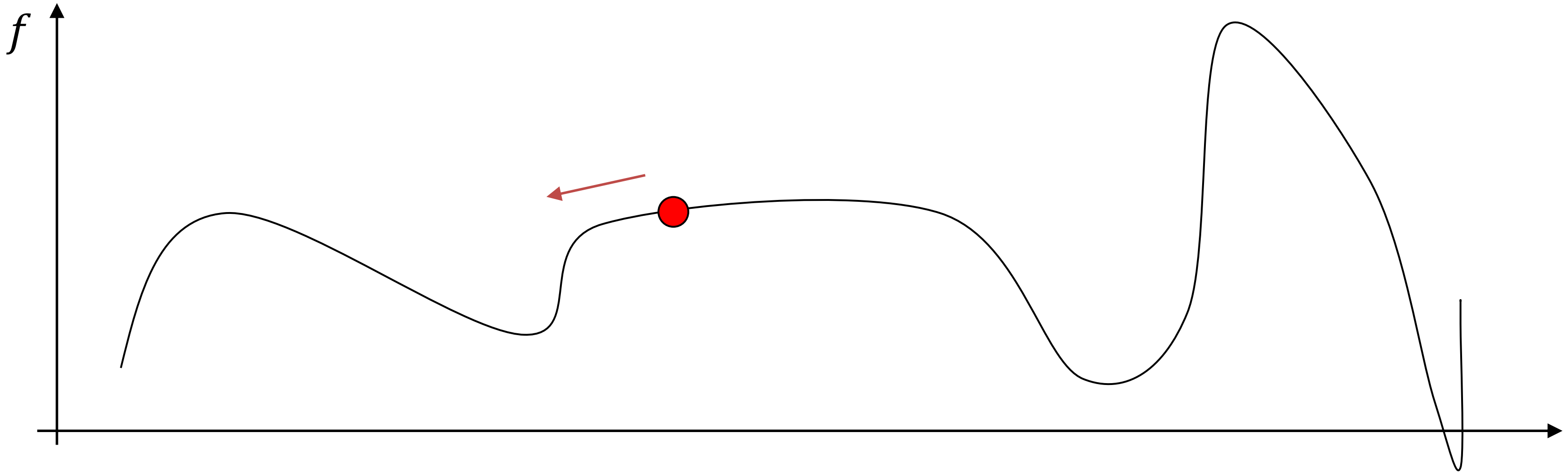
$$-\eta \cdot f'(x_n) = (x_{n+1} - x_n)$$

  – by first-order Taylor series expansion, we have

$$f(x_{n+1}) \approx f(x_n) + f'(x_n) \cdot (x_{n+1} - x_n) = f(x_n) - \eta f'(x_n)^2$$

  – … and it follows

$$f(x_{n+1}) \leq f(x_n)$$



The gradient descent for m dimensions

- Gradient descent for m dimensions:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \cdot \nabla f(\mathbf{x}_n)$$

$$-\eta \cdot \nabla f(\mathbf{x}_n) = (\mathbf{x}_{n+1} - \mathbf{x}_n)$$

  - by first-order Taylor series expansion, we have

$$f(\mathbf{x}_{n+1}) \approx f(\mathbf{x}_n) + (\nabla f(\mathbf{x}_n))^T \cdot (\mathbf{x}_{n+1} - \mathbf{x}_n)$$

$$f(\mathbf{x}_{n+1}) \approx f(\mathbf{x}_n) - \eta\,(\nabla f(\mathbf{x}_n))^T\,(\nabla f(\mathbf{x}_n))$$

$$f(\mathbf{x}_{n+1}) \approx f(\mathbf{x}_n) - \eta \|\nabla f(\mathbf{x}_n)\|^2$$

  - and it follows

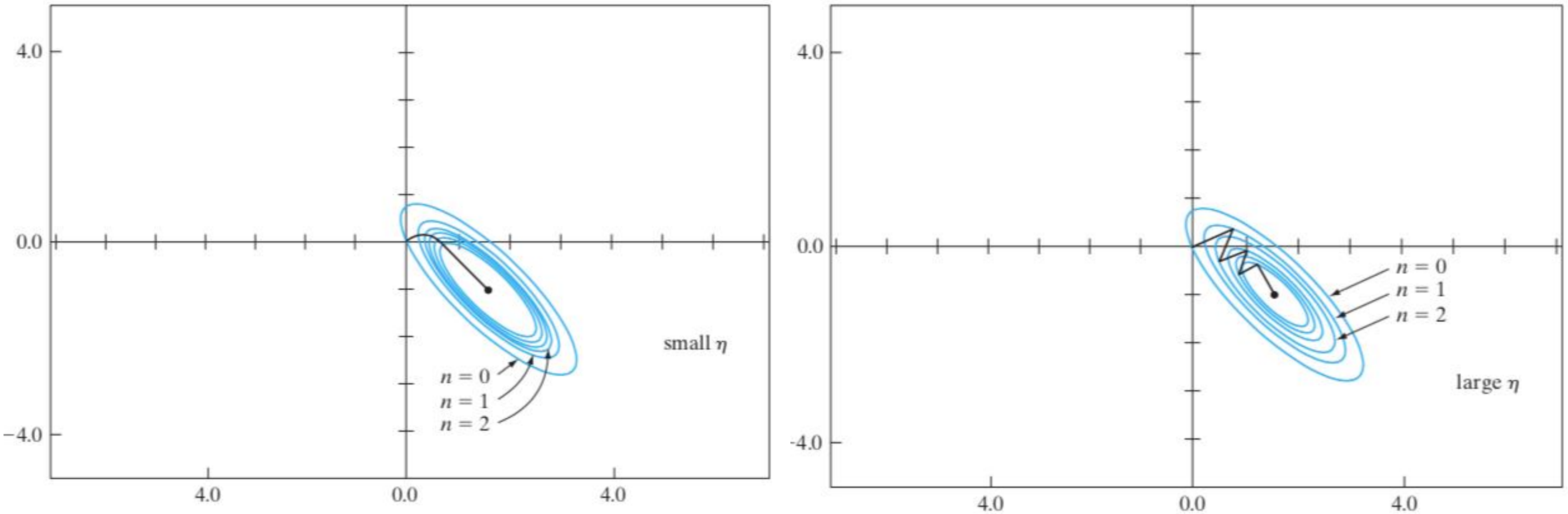
$$f(\mathbf{x}_{n+1}) \leq f(\mathbf{x}_n)$$

Learning parameter η

- The method of steepest descent converges to the optimal solution slowly

- The learning-rate parameter η has a profound influence on its convergence behavior:
  – when η is small: the transient response of the algorithm follows a smooth path (slow)
  – when η is large: the transient response of the algorithm follows a zigzagging (oscillatory) path
  – when η exceeds a certain critical value: the algorithm becomes unstable (i.e., it diverges)

Learning parameter η



Outline

- Perceptron
  - origins
  - learning rule
  - limitations
  - activation functions
- Gradient descent
- **Perceptron revised: inferring update rule**
- Logistic regression
- Stochastic gradient descent

Gradient descent and perceptron learning rule

- Recall that according to the Rosenblatt's rule, all updates have a strength of either 2η or -2η
  - to further guide convergence we should be able to further differentiate weights
  - even correctly classified observations can still provide guidance to update the hyperplane
- Consider a simpler **linear unit** $f(x) = x$ instead of a sign function, i.e., $o = \sum_{j=0}^{m} w_j x_j$
  - the threshold $\theta = 0$ is used to classify observations as positive ($o \geq \theta$) or negative
- Using gradient descent, let's learn w_i that *minimize the **squared errors***, $\{(\mathbf{x}_1, t_1), .., (\mathbf{x}_n, t_n)\}$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=0}^{n}(t_i - o_i)^2$$

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_m}\right]^T$$

Gradient descent and perceptron learning rule

- We want to move the weights $\mathbf{w}$ in the direction that decrease $E(\mathbf{w})$

$$w_j = w_j + \Delta w_j \qquad \mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} \qquad \Delta\mathbf{w} = -\eta \nabla E(\mathbf{w})$$

- Differentiating $E(\mathbf{w})$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=0}^{n}(t_i - o_i)^2 = \frac{1}{2}\sum_{i=0}^{n}\frac{\partial}{\partial w_j}(t_i - o_i)^2$$

$$= \frac{1}{2}\sum_{i=0}^{n} 2(t_i - oi)\frac{\partial}{\partial w_i}(t_i - o_i) = \sum_{i=0}^{n}(t_i - o_i)\frac{\partial}{\partial w_j}\left(t_i - \mathbf{w}^T\mathbf{x}_i\right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{i=0}^{n}(t_i - o_i)\left(-x_j{}^{(i)}\right)$$

Gradient descent and perceptron learning rule

- We reach our target **update rule** for the linear activation

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = \eta \sum_{i=0}^{n} (t_i - o_i) x_j^{(i)}$$

- Quite similar with Rosenblatt's update rule!

  Yet there are two **major differences**:

  1. Rosenblatt's $o_i \in \{0,1\}$, while with linear activation $o_i \in \mathbb{R}$, producing arbitrary $(t_i - oi)$ differences
  2. the inferred update rule first sums the contributions from all observations,
     only then updates the plane (different than Rosenblatt's iterative updates from each observation)

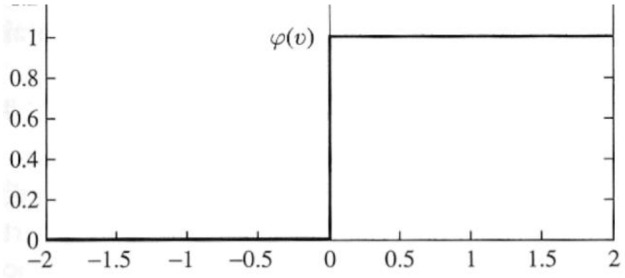
- These differences produce distinct learning settings – convergence can radically differ

Gradient descent and perceptron learning rule

- **Limitations** of gradient descent update using **linear unit**?
  - unbounded activation function (i.e. no lower/upper limits on o_i)
  - radical contributions when $|t_i - o_i|$ is high (superseding other relevant contributions), thus hampering convergence (unstable)
- Solutions?
  - using **alternative activation functions**
  - $sign$ and $step$ units?
    - not differentiable thus unable to apply in the Gradient Descent
    - we need to explore further alternatives

step (sign variant)

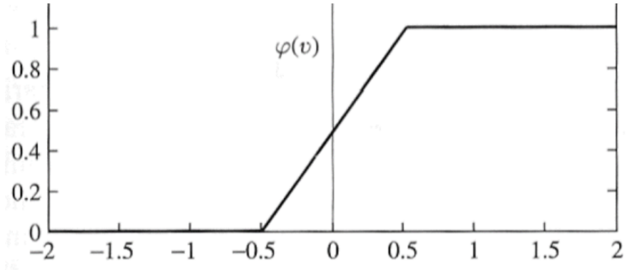
$$f(x) = sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$



Alternative activation functions

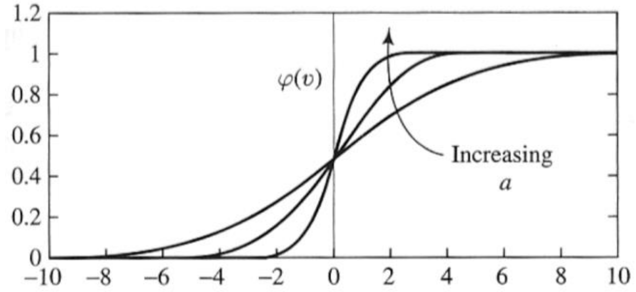
bounded linear

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0.5 \\ x & \text{if } -0.5 > x > 0.5 \\ 0 & \text{if } x \leq -0.5 \end{cases}$$



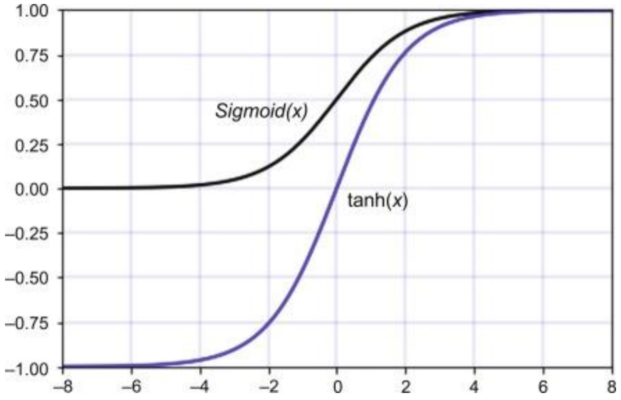
sigmoid

$$f(x) = \sigma(ax) = \frac{1}{1+e^{-ax}}$$



hyperbolic tangent (tanh)

$$f(x) = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



Rule for continuous activation functions

- Let us infer a general rule for differentiable activation functions
  - given a continuous activation ϕ

$$o_i = \phi\left(\sum_{j=0}^{m} w_j x_{ij}\right)$$

  - we can define the gradient descent rule as follows

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=0}^{n}(t_i - o_i)^2$$

$$= \frac{1}{2}\sum_{i=0}^{n} 2(t_i - oi)\frac{\partial}{\partial w_i}(t_i - o_i) = \sum_{i=0}^{n}(t_i - o_i)\frac{\partial}{\partial w_j}\left(-\phi\left(\sum_{j=0}^{m} w_j x_{ij}\right)\right)$$

$$= -\sum_{i=0}^{n}(t_i - o_i)\phi'\left(\sum_{j=0}^{m} w_j x_{ij}\right)\frac{\partial}{\partial w_j}\left(\sum_{j=0}^{m} w_j x_{ij}\right)$$

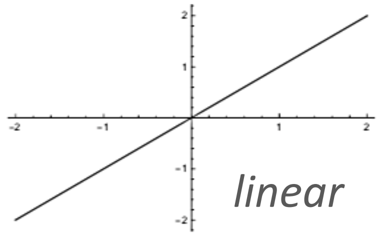
$$= -\sum_{i=0}^{n}(t_i - oi)\phi'\left(\mathbf{w}^T\mathbf{x}_i\right)x_{ij}$$

Outline

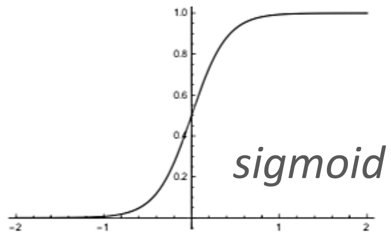
- Perceptron
  - origins
  - learning rule
  - limitations
  - activation functions
- Gradient descent
- Perceptron revised: inferring update rule
- **Logistic regression**
- Stochastic gradient descent

Gradient descent for sigmoid unit

- Let us recall that until now we have two major ways of learning a perceptron
  - classic Rosenblatt's update
  - gradient descent rule with linear unit $\Delta w_j = \eta \sum_{i=1}^{n}(t_i - o_i)x_j^{(i)}$



- Nevertheless we encountered limitations for both options
- Alternative?
  - consider **sigmoid unit**, $\sigma(net) = \frac{1}{1+e^{-\alpha \cdot net}} = \frac{e^{\alpha \cdot net}}{1+e^{\alpha \cdot net}}$
    - $o_i = \sigma\left(\sum_{j=0}^{m} w_j x_j^{(i)}\right)$ for α=1
    - knowing $\sigma'(\mathrm{x}) = \sigma(\mathrm{x}) \cdot (1 - \sigma(\mathrm{x}))$
    - minimizing the **squared error** to compute the update rule:



$$\Delta w_j = -\eta \frac{\partial E(\mathbf{w})}{\partial w_j} = \eta \cdot \alpha \cdot \sum_{i=0}^{n} (t_i - o_i) \cdot o_i \cdot (1 - o_i) \cdot x_j^{(i)}$$

Gradient descent for sigmoid unit

- When considering squared errors, gradient descent with sigmoid differs from previous solutions:

$$\Delta w_j = -\eta \frac{\partial E(\mathbf{w})}{\partial w_j} = \eta \cdot \alpha \cdot \sum_{i=0}^{n} (t_i - o_i) \cdot \boxed{o_i \cdot (1 - o_i)} \cdot x_j^{(i)}$$

  – these novel components in the update ensure a generally more stable convergence

  – however as $o_i \in [0,1[$, the number of required updates can be considerably high

- How to address this last observation?

  – what if instead of the sum of squared errors, we consider an alternative loss function?

  – solution: gradient descent with **cross-entropy error**

$$E(\mathbf{w}) = -\log\big(p(\boldsymbol{t}|\mathbf{w})\big) = -\sum_{i=1}^{n} (t_i \log(o_i) + (1 - \mathrm{t}_i) \log(1 - o_i))$$

  – example $\mathbf{t} = [1,0,1]$, $\boldsymbol{o} = [0.8,0.15,0.9]$,
  then $E(\mathbf{w}) = -(1 \times \log(0.8) + 1 \times \log(0.85) + 1 \times \log(0.9)) = 0.7$

Logistic regression

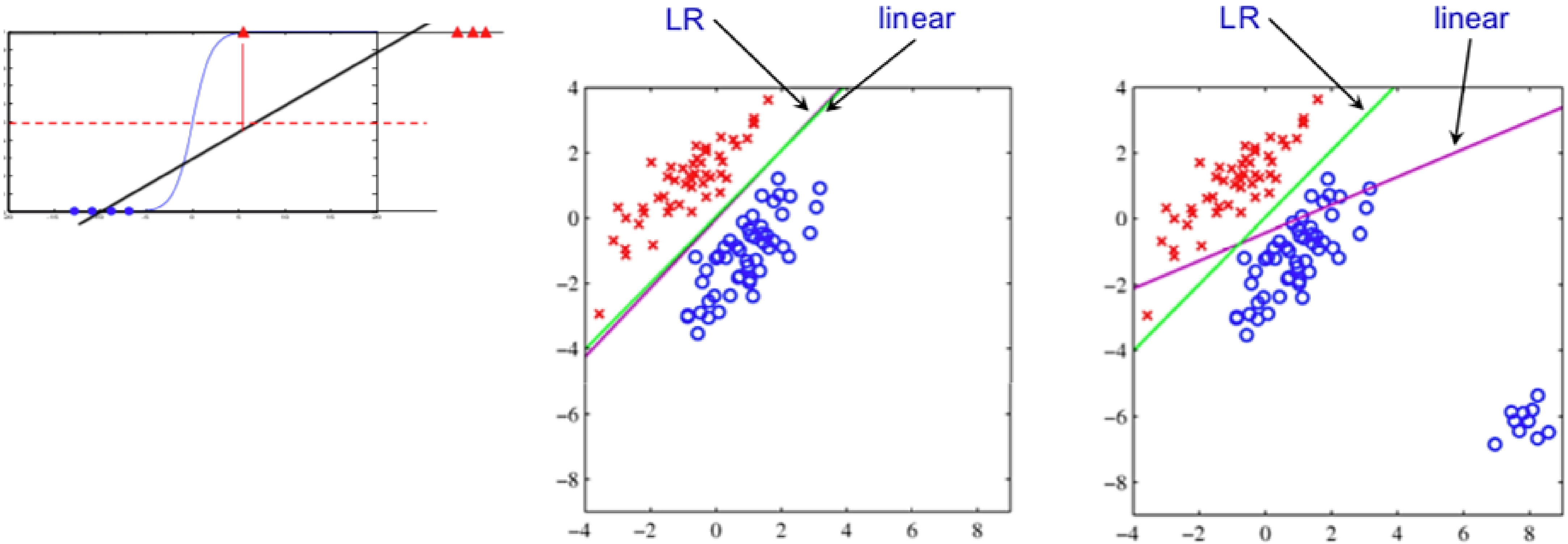
- **Logistic regression** is a perceptron…
  - with a **sigmoid** activation and gradient descent update rule for **cross-entropy** error
  - a *classifier* despite the reference to *regression*
- Exercise: infer the update rule for the logistic regression using gradient descent
  - recall that $\frac{\partial\ \log_a x}{\partial x} = \frac{1}{x}\ln(a)$ and consider $a = e$ for simplicity sake (common assumption)
- Solution: the update rule for the logistic regression is given by:

$$\Delta w_j = \eta \sum_{i=1}^{n} (t_i - o_i) x_j{}^{(i)}$$

  - rule differs from the previous sigmoid unit with SSE: simpler rule, bolder updates
  - although resembling the linear unit rule: o_i is now bounded producing more stable updates

Linear unit versus Logistic regression

- **Logistic regression** and **sigmoid unit** generally give us better boundaries than **linear unit**
  - distant points from the hyperplane have the same impact, while high weight in linear

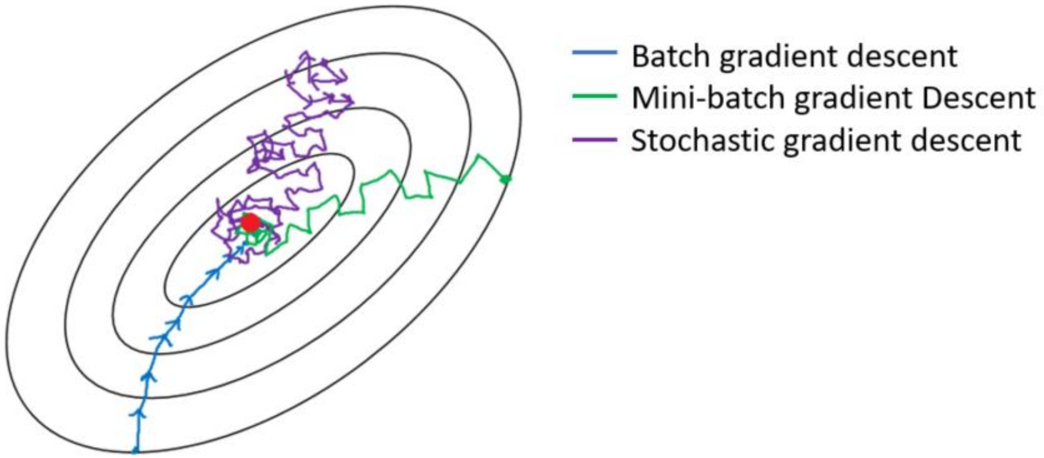


Outline

- Perceptron
  - origins
  - learning rule
  - limitations
  - activation functions
- Gradient descent
- Perceptron revised: inferring update rule
- Logistic regression
- **Stochastic gradient descent**

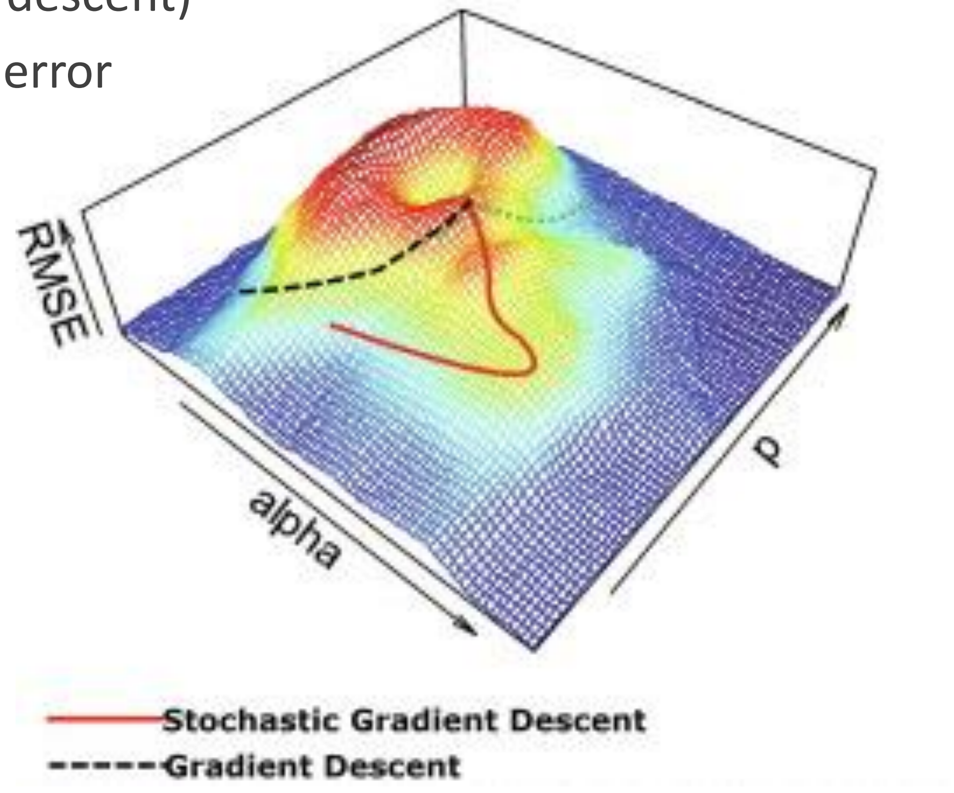
Stochastic gradient descent

- classic/steepest **gradient descent training rule**
  - updates summing all the training examples X (see previous sections)
- **stochastic gradient descent training rule**
  - approximates gradient decent by *updating weights incrementally*
    - calculates error for each observation
      - for instance, $\Delta w_j = \eta(t_i - oi){x_j}^{(i)}$
        instead of $\Delta w_j = \eta \sum_{i=0}^{n}(t_i - oi){x_j}^{(i)}$
    - just like Rosenblatt's observation-specific updates
    - known as delta-rule or LMS weight update



Stochastic gradient descent

- Least Mean Squares (LMS) to estimate weight vector
  - **no well-defined trajectory** in the weight space
    - instead a **random trajectory** (stochastic gradient descent)
  - converge only asymptotically toward the minimum error
    - no steepest descent
  - can approximate gradient descent arbitrarily closely if made small enough η



Gradient descent exercise

a) Derive the training classic gradient descent rule for a perceptron with the activation

$$o = e^{\left(\sum_{j=1}^{m} w_j \times w_j \times x_j\right)}$$

considering the sum of squared errors error function.

b) Given $\eta = 1$ and $\mathbf{w} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, do a *stochastic* update for observation $\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ with target $t = 0$.

Solution notes

a) $\frac{\partial net}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\sum_{j=1}^{m} w_j{}^2 x_j\right) = 2w_j x_j$

$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = \cdots = \eta \sum_{i=0}^{n}(t_i - o_i) \frac{\partial}{\partial w_j}\left(-\exp\left(\sum_{j=1}^{m} w_j{}^2 x_j\right)\right) = \cdots = 2\,\eta\, w_j \sum_{i=0}^{n}(t_i - oi)\, o_i\, x_{ij}$

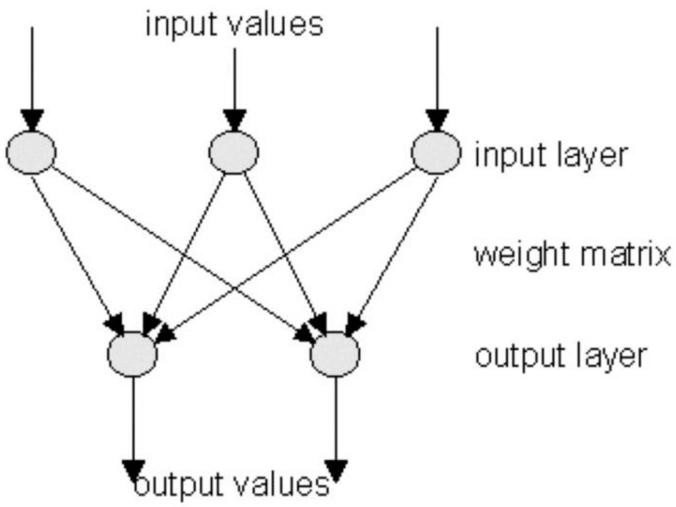
b) $o = \exp(1 \times 1 \times 1 + 0 \times 0 \times 1) = e, \Delta w_1 = 2 \times (0 - e) \times e \times 1 = -2e$

$w_1^{new} = 1 - 2e^2 = 13.78, \Delta w_2 = 0, w_2^{new} = 0$

Going forward…

- Summary: training rules can be inferred for different activation functions
  - gradient or stochastic descent guaranteed to converge to hypothesis with minimum squared error
    - given sufficiently small learning rate η
    - even when training data contains noise
    - even when training data not separable

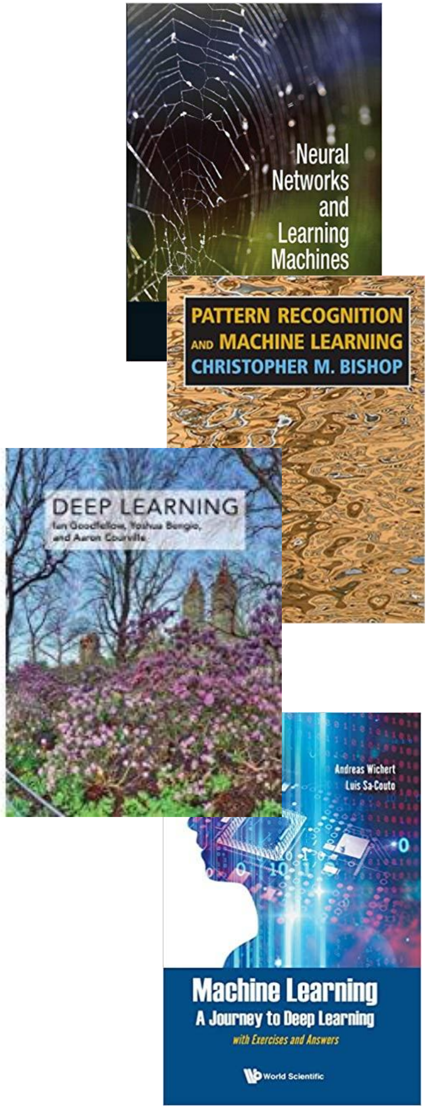
- Next step: solving frontier problems (e.g. XOR)
  - How? **multi-layer preceptrons**



Outline

- **Perceptron**
  - origins
  - learning rule
  - limitations
  - activation functions
- **Gradient descent**
- **Perceptron revised**: inferring update rule
- **Stochastic gradient descent**

Literature



- S. Haykin, Neural Networks and Learning Machine, (3rd Edition), Pearson 2008
  - Chapter 1
- C. Bishop, Pattern Recognition and Machine Learning, Springer 2006
  - Section *1.4*
- Deep Learning, I. Goodfellow, Y. Bengio, A. Courville, MIT Press 2016
  - Chapters 2 and 4
- A. Wichert, L. Sá-Couto, Machine Learning - A Journey to Deep Learning, World Scientific, 2021
  - Chapters 1 and 3

Thank You

rmch@tecnico.ulisboa.pt
andreas.wichert@tecnico.ulisboa.pt