Perceptrons

Lecture 15 of “Mathematics and AI”

Outline

1. A neural approach to AI

   Theoretical neuron, synapses, action potential, threshold voltage, plasticity

2. Perceptron

3. Learning algorithms

   Perceptron learning, stochastic gradient descent, mini-batch

4. Variants of stochastic gradient descent

   Momentum, Averaging, RMSProp, Adam

A neural approach to AI

Back to beginnings …

- AI: an interdisciplinary challenge
  - Mathematics,
  - physics,
  - philosophy,
  - psychology,
  - neuroscience

Photo by Margaret Minsky

The human brain as a computer

- Is the brain as a *thinking machine*?
- How could one build such a machine from scratch?

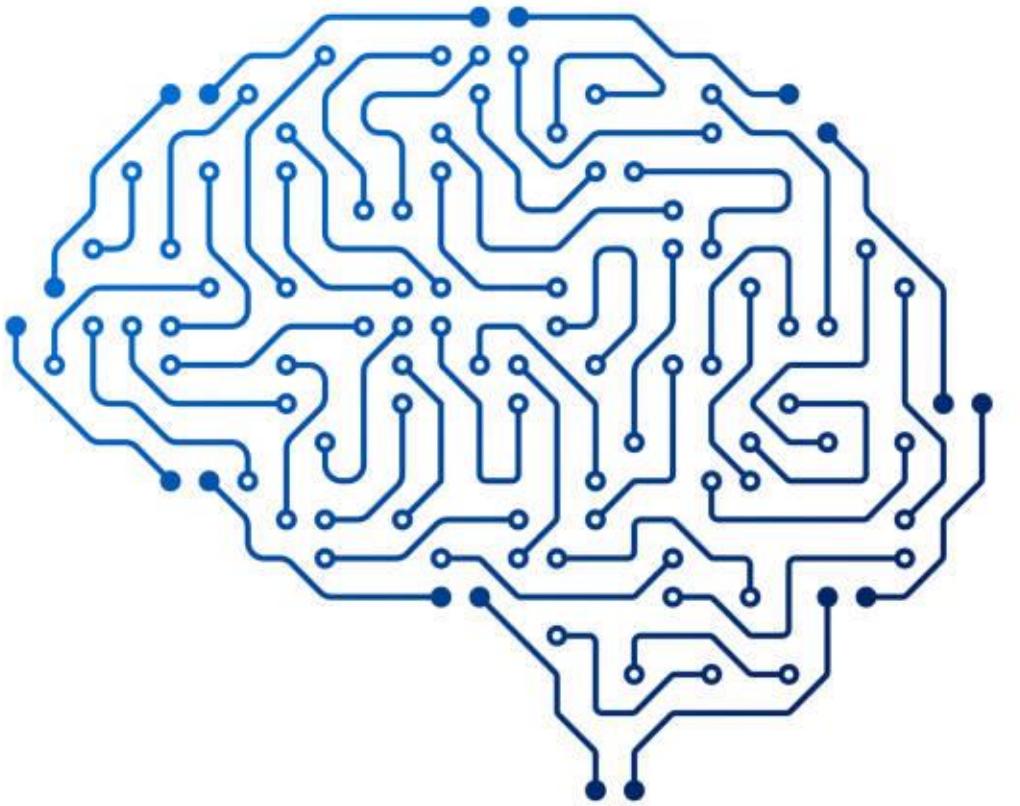


Image source: iStock

The neuron

- Fundamental information processing unit of the brain
- Neuron as I/O system:
  - Input: dendrites
  - Information aggregation: soma
  - Output: axon terminals
  - Communication: synapses

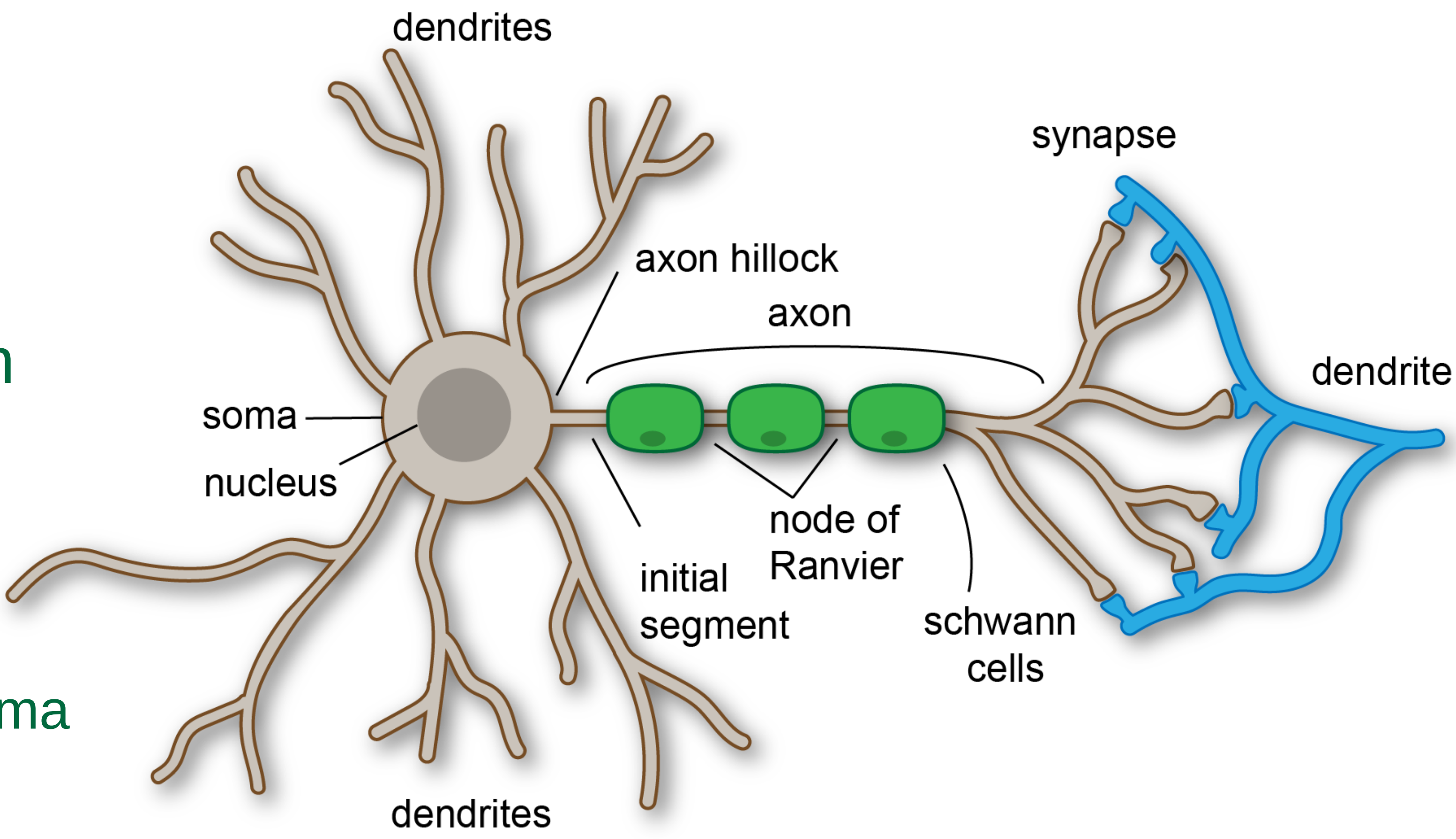


Image source: Wikipedia

Theoretical models of neural dynamics

- Mathematical models of neurons
  - Hodgkin-Huxley model, Integrate-and-fire (IF) model, leaky integrate-and-fire (LIF) model
- Machine learning with spiking neural networks (SNNs)
  - Computer vision, robotics, …

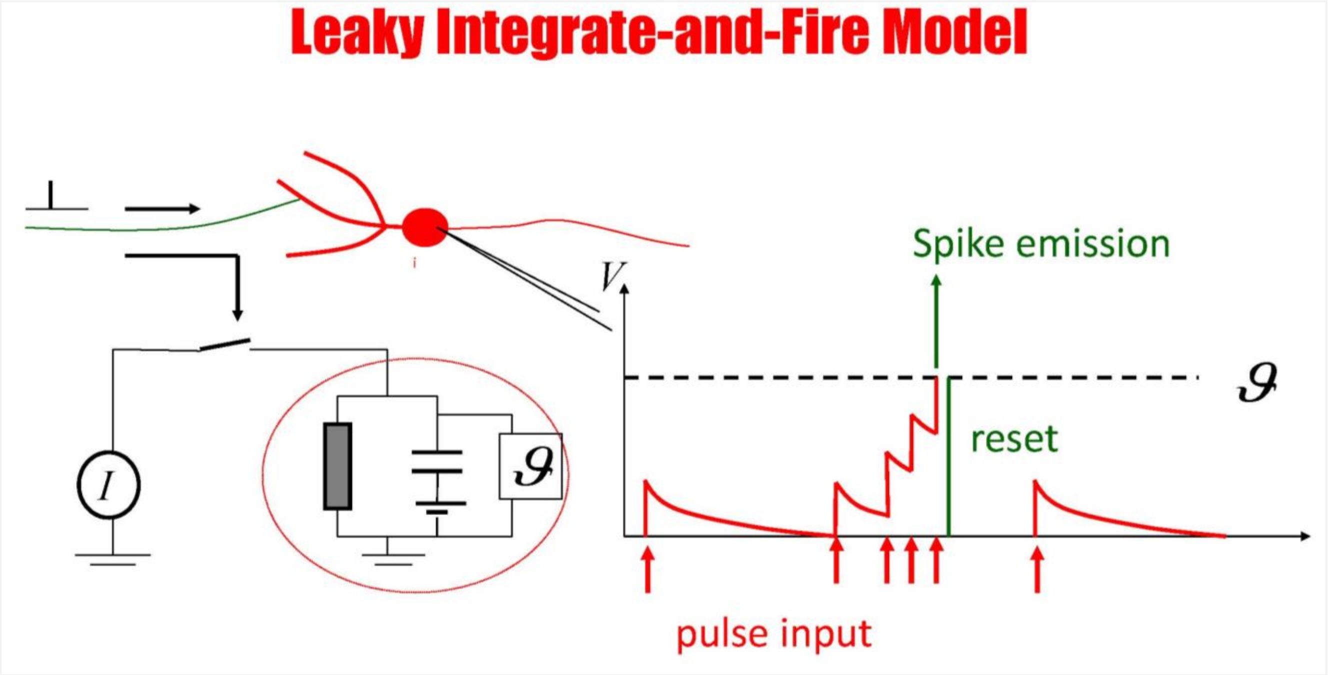


Image source: Wikipedia

Perceptron

Perceptron

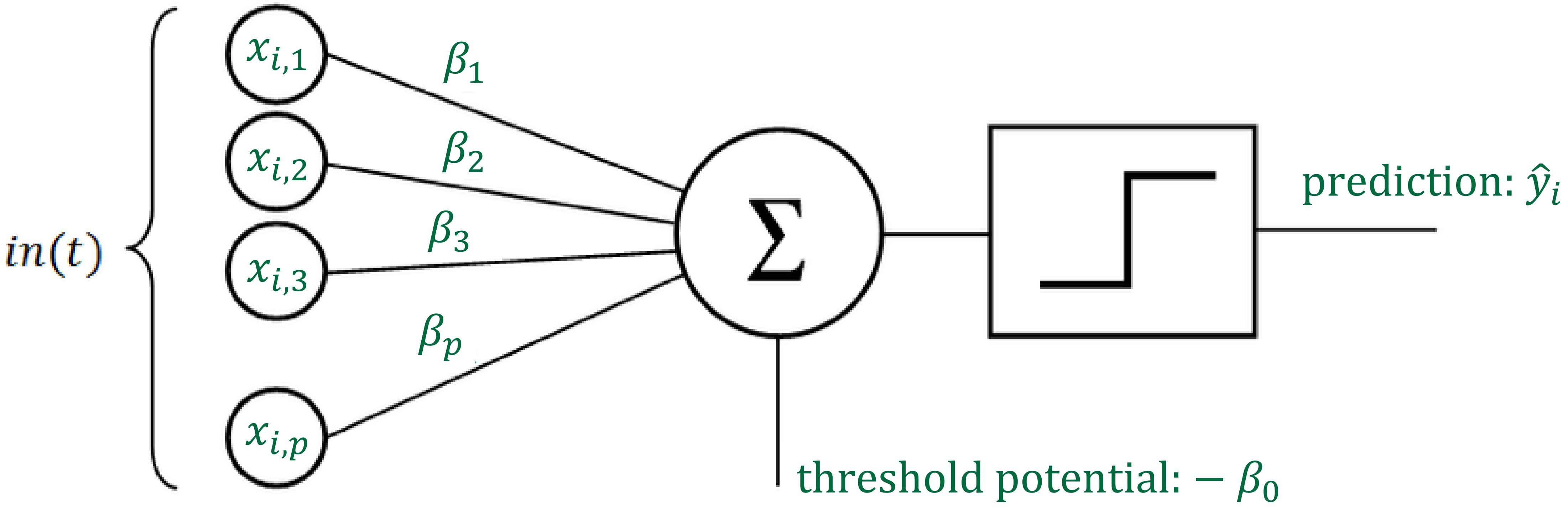
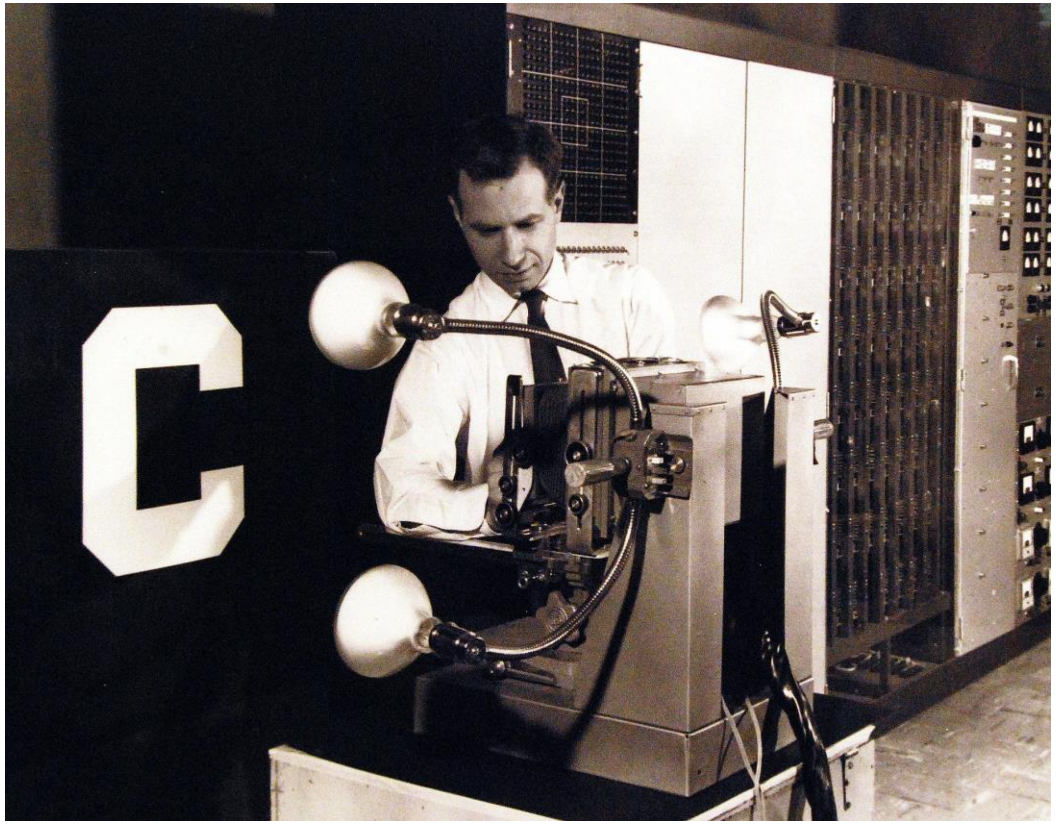


Image source: https://github.com/jg-fisher/perceptron

Perceptron



The Mark I Perceptron

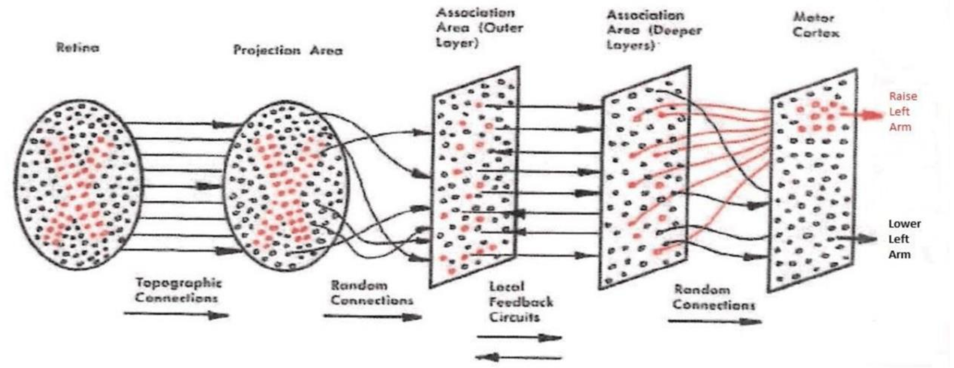


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

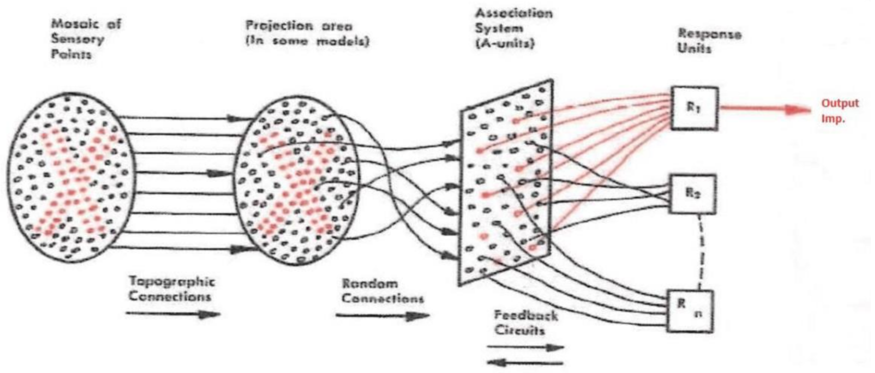
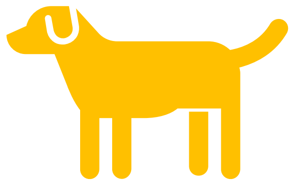


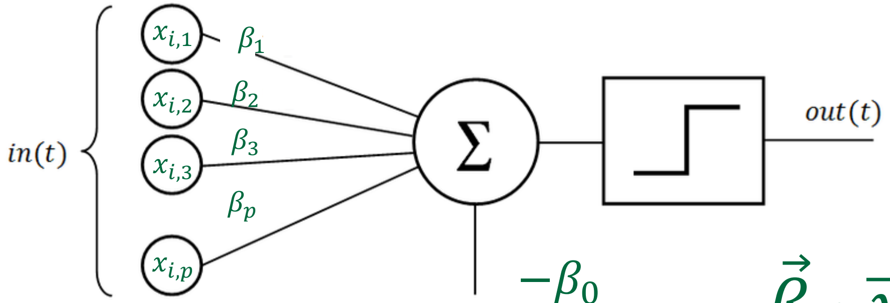
FIG. 2 — Organization of a perceptron.

Image source: National Museum of the U.S. Navy - 330-PSA-80-60 (USN 710739), Rosenblatt, F. - Rosenblatt, F. The Design of an Intelligent Automaton, Research Reviews, Office of Naval Research. Washington, October 1958, 5-13

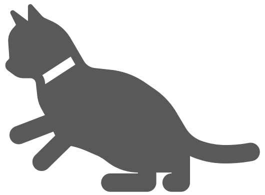
Perceptron as a binary linear classifier



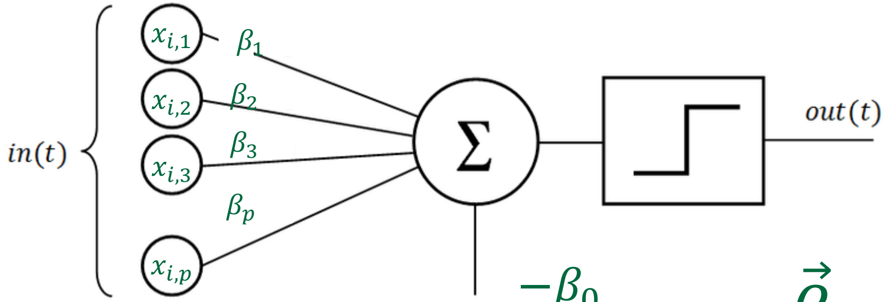
$$y_i = 1$$

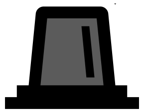


$$\vec{\beta} \cdot \vec{x_i} + \beta_0 \geq 0$$



$$y_i = 0$$

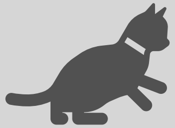




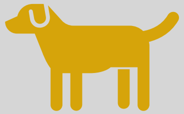
$$\vec{\beta} \cdot \vec{x_i} + \beta_0 < 0$$

Image source: https://github.com/jg-fisher/perceptron

Linear decision boundary for binary classification

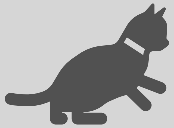
$\{\vec{x_i} \mid \beta_0 + \vec{\beta} \cdot \vec{x_i} < 0\}$ 

- $\vec{x_i}$ on one side of hyperplane
- $(\vec{x_i}, y_i)$ is “cat”
- $y_i = -1$

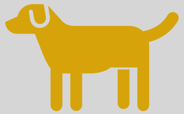
$\{\vec{x_i} \mid \beta_0 + \vec{\beta} \cdot \vec{x_i} > 0\}$ 

- $\vec{x_i}$ on other side of hyperplane
- $(\vec{x_i}, y_i)$ is “dog”
- $y_i = +1$

For **all** training samples:



$$y_i\left(\beta_0 + \vec{\beta} \cdot \vec{x_i}\right) > 0$$



Learning algorithms

Perceptron learning algorithm

- Natural learning is sequential
- Gradient descent does not work with step functions

➢ Perceptron learning algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

Image source: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

Perceptron learning

- Natural learning is sequential
- Gradient descent does not work with step functions

➢ Perceptron learning algorithm

+ Works for with step functions

- Converges slowly or not at all

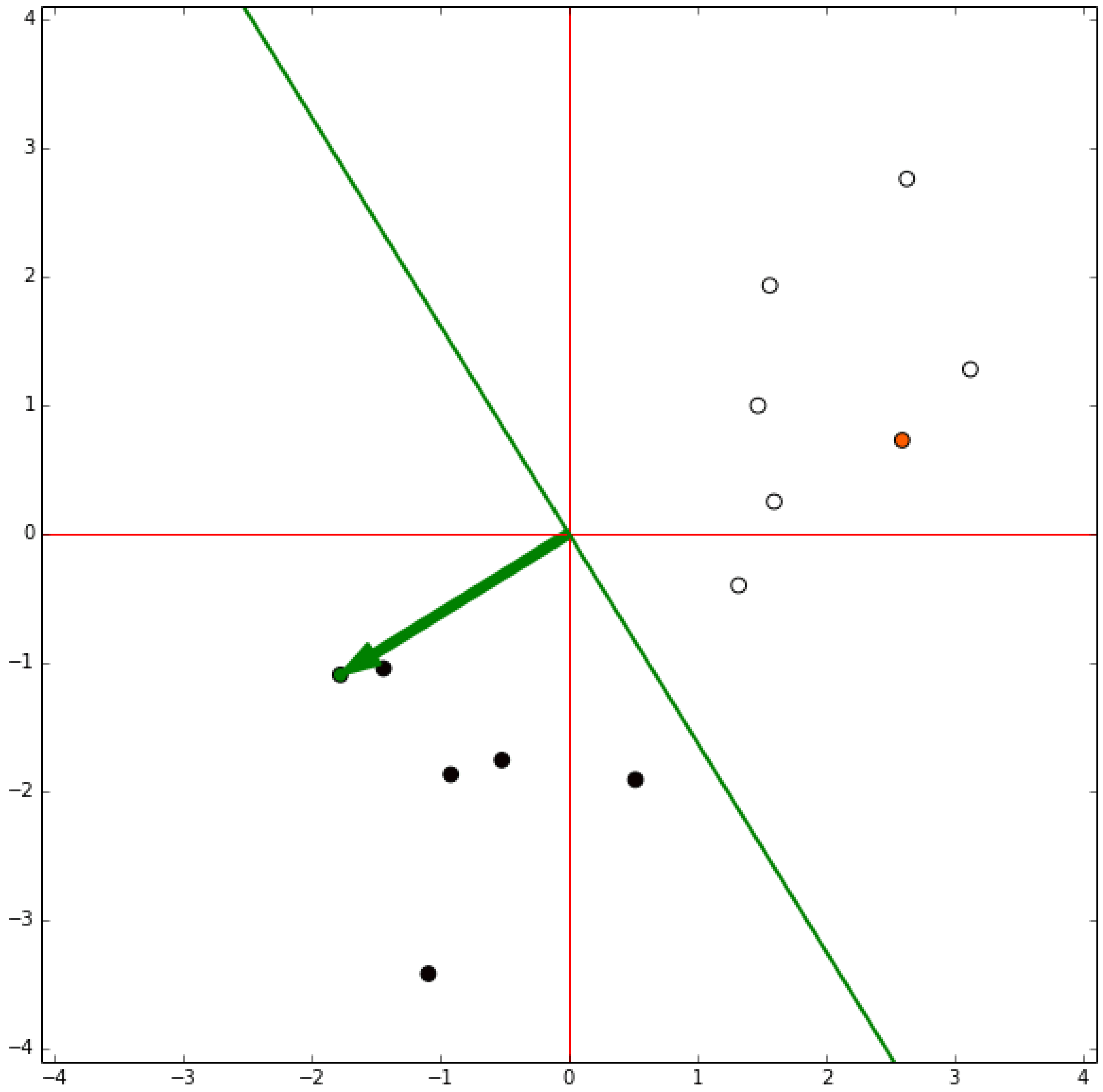


Image source: Wikipedia

Stochastic gradient descent (SGD)

- SGD: Sequential gradient-based learning

Gradient descent (full batch)	Stochastic gradient descent
Loss function: $$L_{GD}(\vec{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\vec{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\vec{\beta}, \vec{x}_i, y_i)$$ Update rule: $$\vec{\beta}' = \vec{\beta} - \gamma \nabla L_{GD}(\vec{\beta}) = \vec{\beta} - \frac{\gamma}{n}\sum_{i=1}^{n} \nabla \ell_i(\vec{\beta})$$	Loss function: $$L_{SGD}(\vec{\beta}) = \ell_i(\vec{\beta}, \vec{x}_i, y_i)$$ Update rule: $$\vec{\beta}' = \vec{\beta} - \nabla L_{SGD}(\vec{\beta}) = \vec{\beta} - \gamma \nabla \ell_i(\vec{\beta})$$

Stochastic gradient descent (SGD)

- Example: OLS with one feature

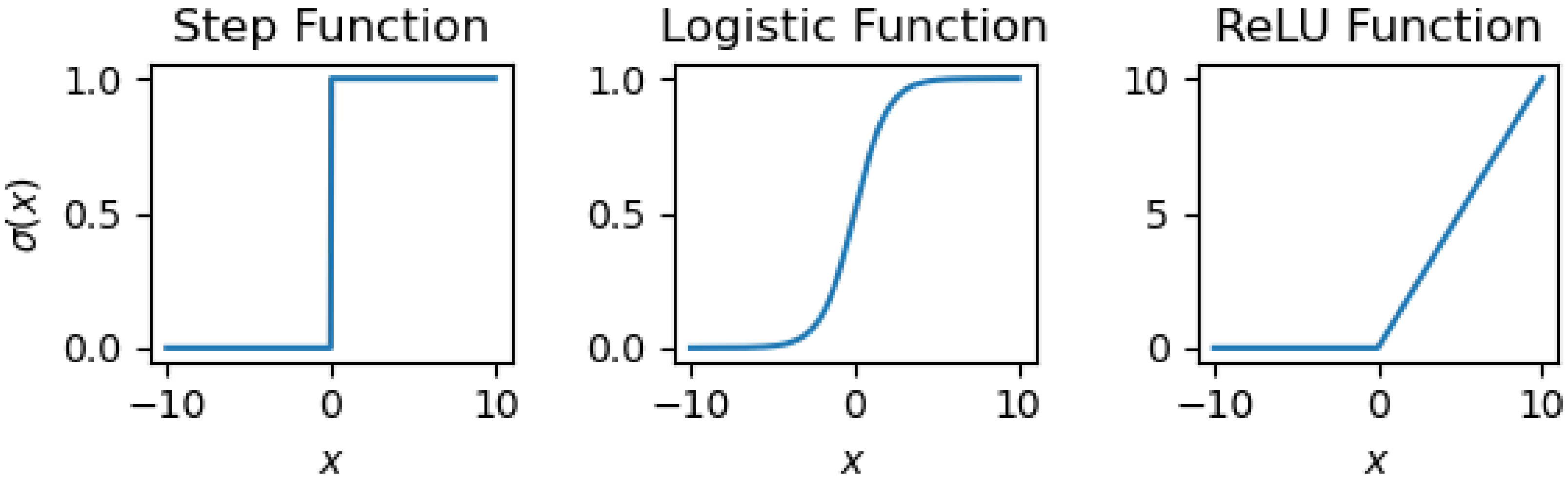
$$L_{GD}(\vec{\beta}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\beta_0 + \vec{\beta}\cdot\vec{x}_i - y_i\right)^2$$

$$L_{SGD}(\vec{\beta}) = \ell_i(\vec{\beta}, \vec{x}_i, y_i) = \left(\beta_0 + \vec{\beta}\cdot\vec{x}_i - y_i\right)^2$$

$$\vec{\beta}' = \vec{\beta} - 2\gamma\left(\beta_0 + \vec{\beta}\cdot\vec{x}_i - y_i\right)\vec{x}_i$$

Activation functions

- SGD does not work with step functions, but with many other naturalistically motivated activation functions
- Examples: logistic/sigmoid function, ReLU (Rectified linear unit)



Linear activation function?

- Unbounded function:
  - Not biologically motivated
  - Not well suited for classification problems
- Linear function:
  - $\vec{\beta} \cdot \vec{x}_i$ is already a linear transformation (LT)
  - LT(LT($\vec{x}_i$)) is just another LT on $\vec{x}_i$
  - Does not change model expressiveness

Variants of SGD

Variants of SGD

GD can get stuck in $\nabla L_{GD}(\vec{\beta}) = 0$ zones

SGD converges slowly and is sensitive to outliers

- Variants of SGD that include smoothening effect
  - Averaging / mini-batch
  - Momentum
  - RMSProp
  - Adam

Averaging/ mini-batch learning

- Split training data into “mini-batches” of size m
- Loss:

$$L_{SGD}(\vec{\beta}) = \frac{1}{m}\sum_{i=1}^{m} \ell_i(\vec{\beta}, \vec{x}_i, y_i)$$

- Update rule:

$$\vec{\beta}' = \vec{\beta} - \gamma \nabla L_{SGD}(\vec{\beta})$$

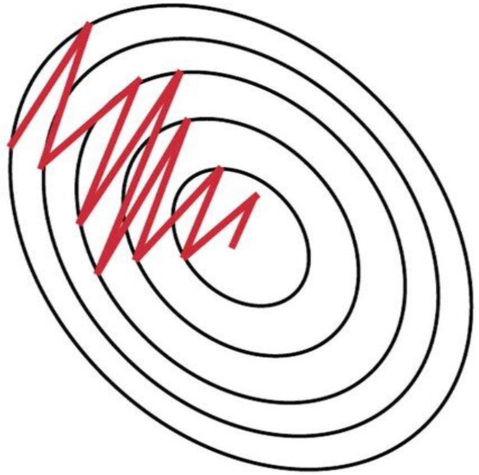
- Improves robustness to outliers

SGD with momentum

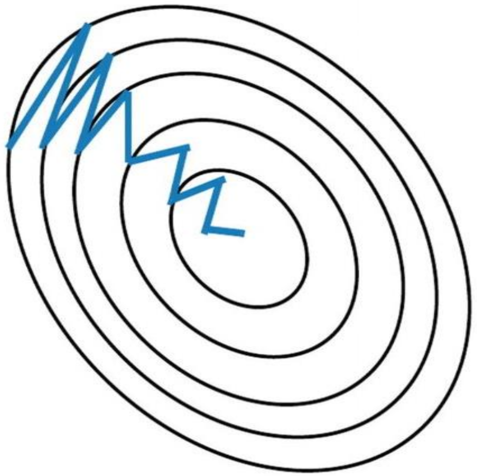
- Add momentum to how $\vec{\beta}$ changes
- Loss:
  - Standard SGD loss or mini-batch
- Update rule:

$$\vec{\beta}' = \vec{\beta} + \Delta\vec{\beta}' \quad \text{with} \quad \Delta\vec{\beta}' = -\gamma \nabla L_{SGD}(\vec{\beta}) + \alpha \Delta\vec{\beta}$$

- Improves robustness for high-variance data



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Image source: https://statusneo.com/accelerate-convergence-mini-batch-momentum-in-deep-learning/

RMSProp, Adam

- RMSProp (Root mean square propagation)
- Adjust learning rate for each parameter
- Update rule:

  $\vec{\beta}' = \vec{\beta} + \Delta\vec{\beta}'$ with $\Delta\vec{\beta}' = -\frac{\gamma}{v}\nabla L_{SGD}(\vec{\beta})$ and

  v moving average of $\Delta\vec{\beta}'$
- Improves robustness to high-variance features
- Adam (Adaptive Moment Estimation)
  - combines RMSProp with momentum