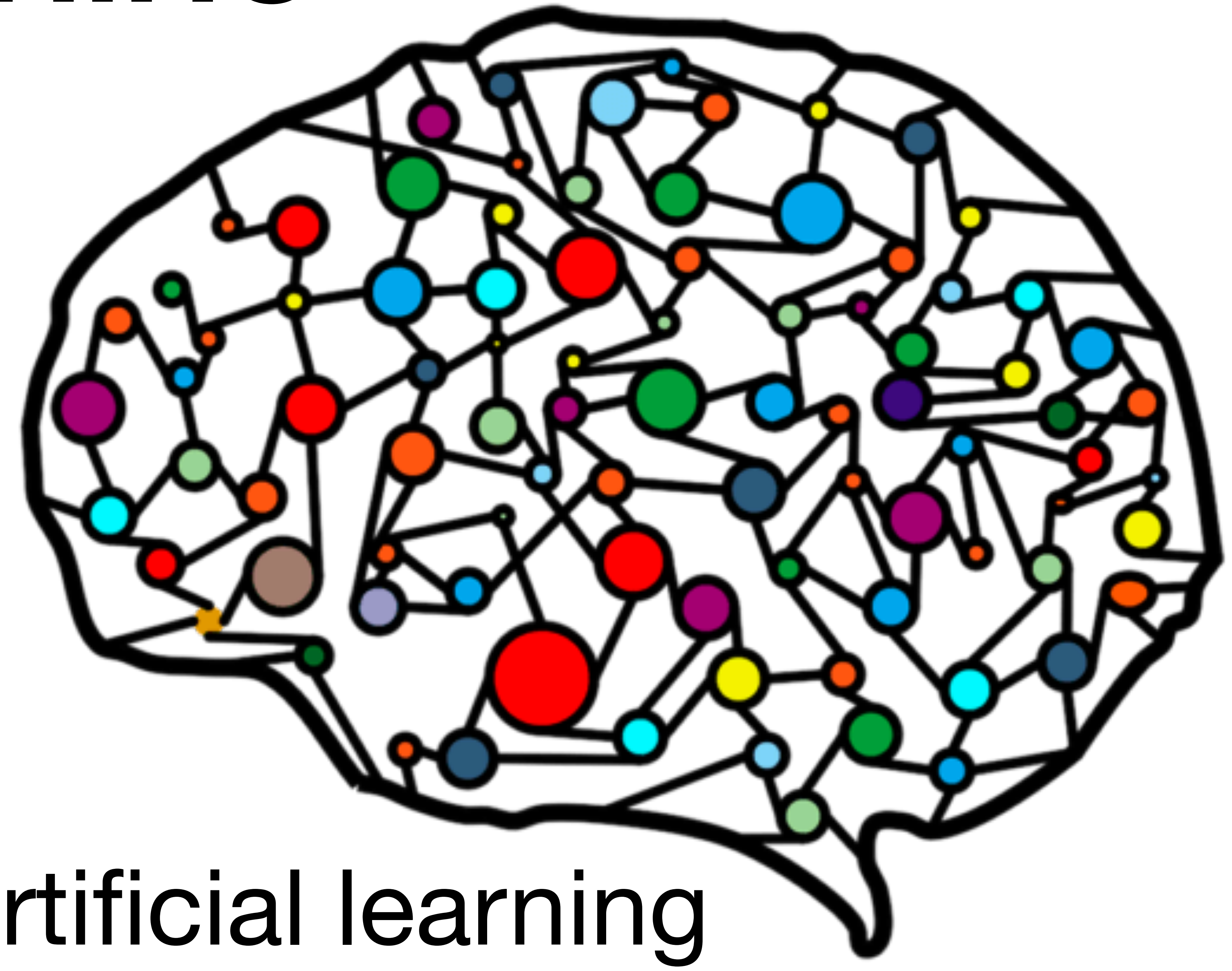
General Principles of Human and Machine Learning



Lecture 2: Origins of biological and artificial learning

Dr. Charley Wu

https://hmc-lab.com/GPHML.html

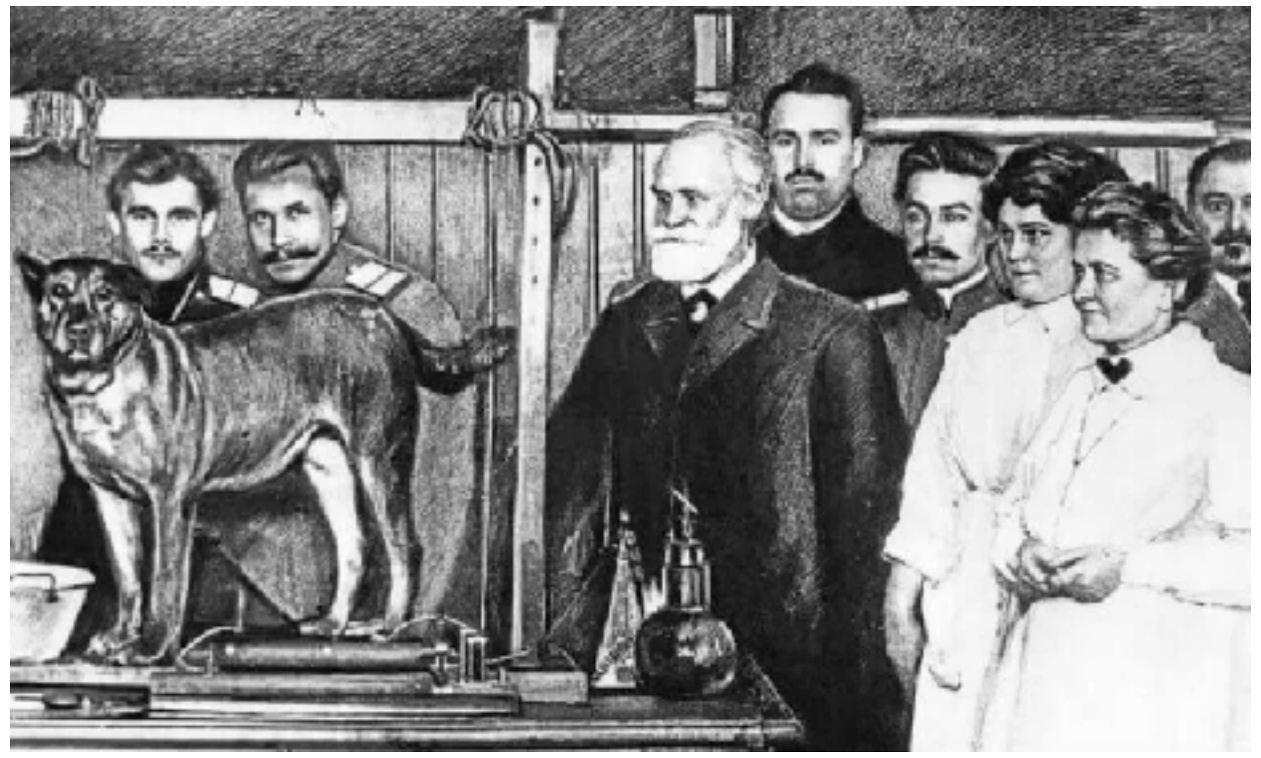
Organization

- To allow time for people to travel between classes
  - Lectures: 10:30 - 12:00 on Thursdays
  - Tutorials: 16:15 - 17:30 on Fridays

Lesson plan

1. Behavioralism

- Understanding intelligence through behavior



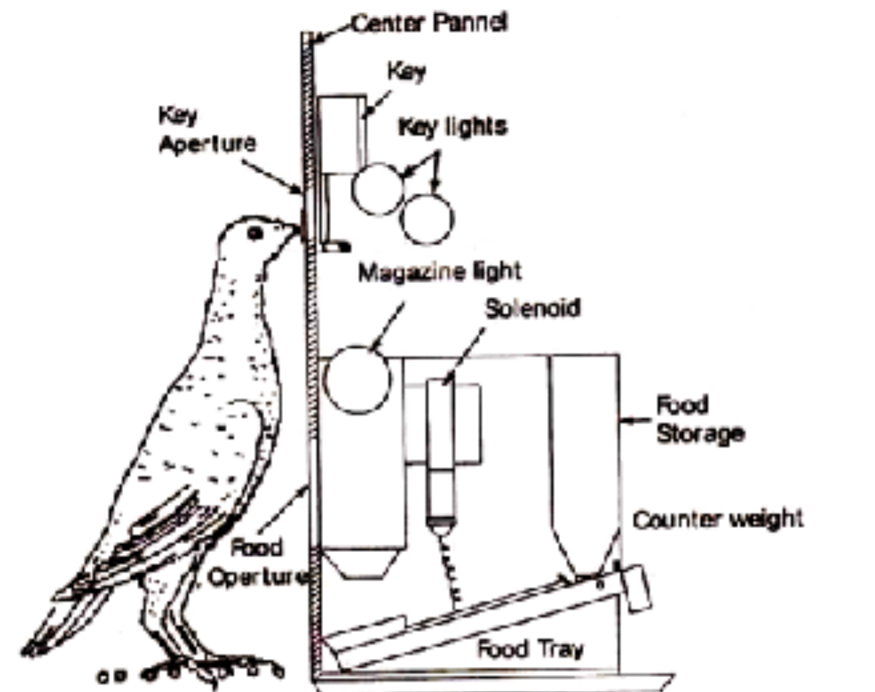
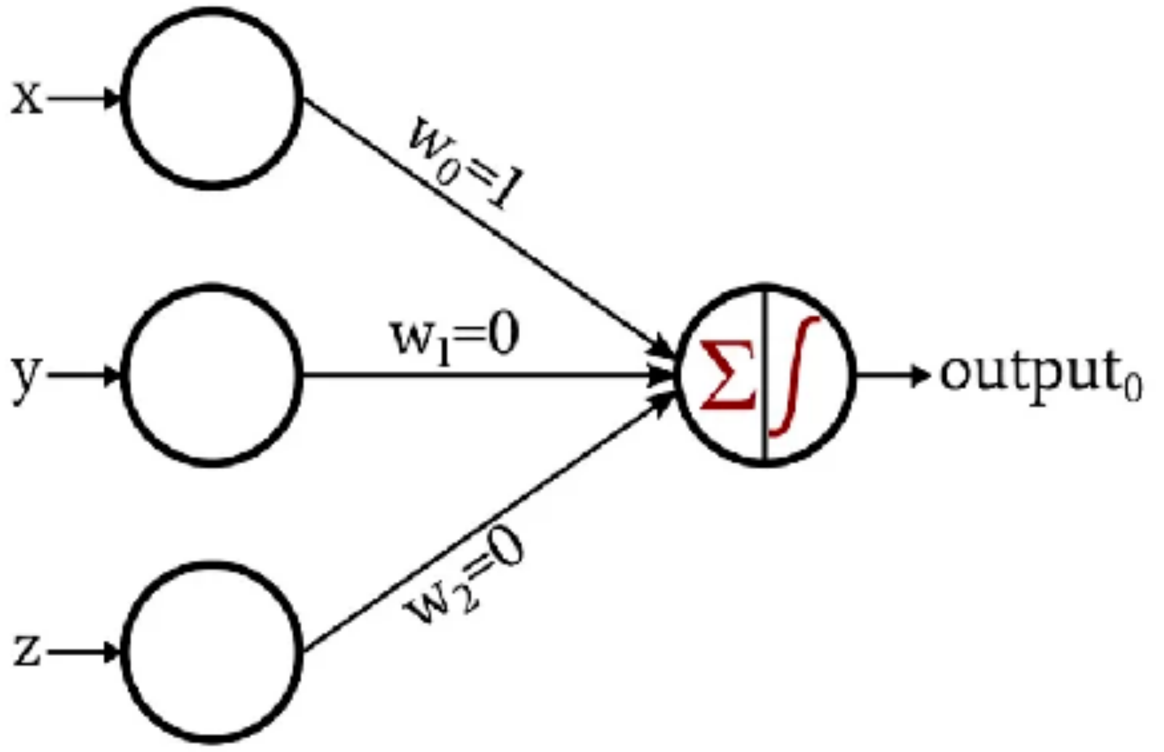


Illustration. Skinner box as adapted for the pigeon.

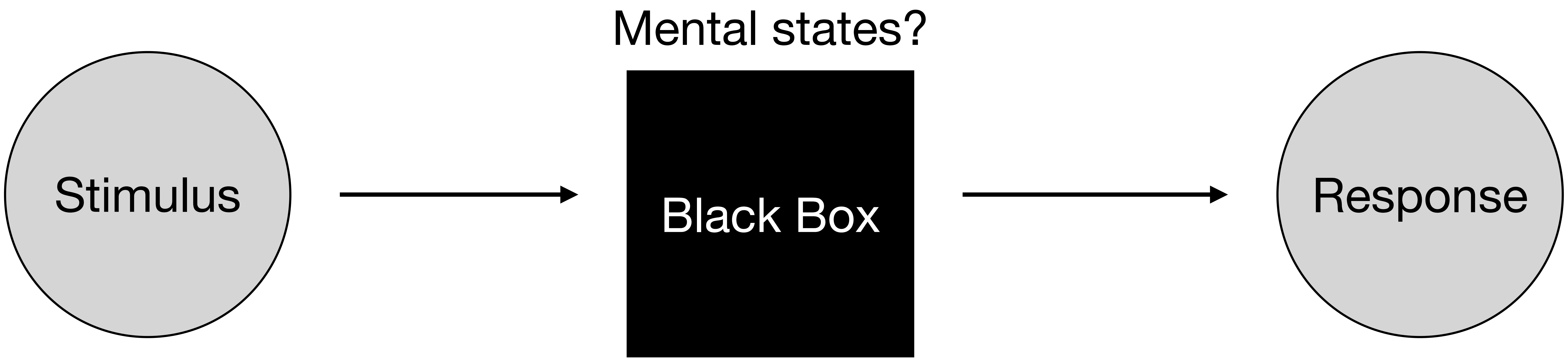
2. Connectionism

- Understanding intelligence through artificial neural networks



Behaviorism

- [*noun* Psychology.] An approach to understanding the behavior of humans and animals that emerged in the early 1900s
  - Generally tries to focus on outward observable behavior rather than hidden inner mental states
  - One of the earliest programs to empirically study biological intelligence and learning



Varieties of Behaviorism

John B. Watson

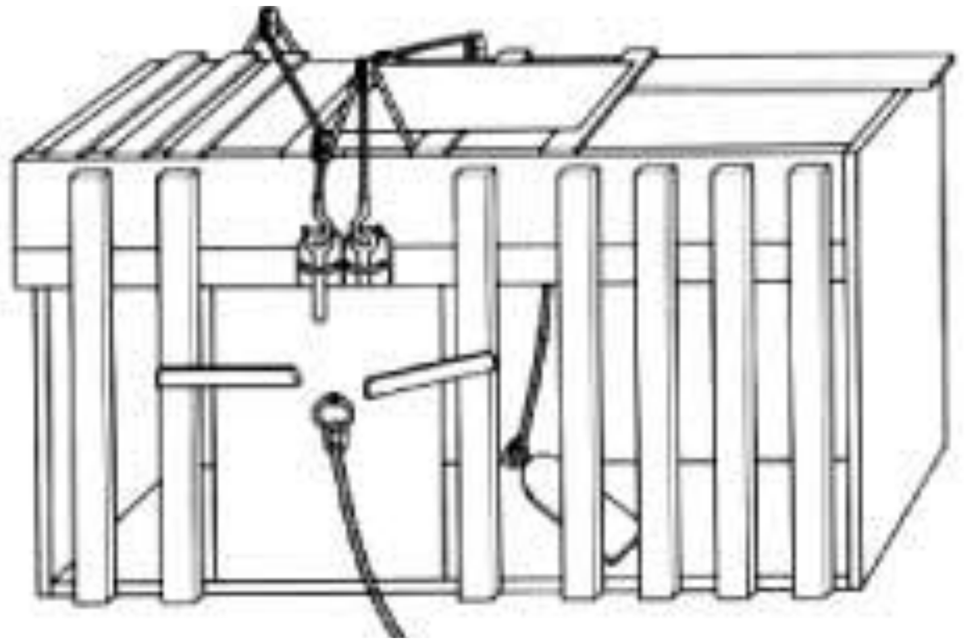
B.F. Skinner

Methodological Behaviorism ⟷ **Radical Behaviorism**

- Thoughts and feelings exist, but cannot be the target of scientific study
- Only public events can be objectively observed and studied scientifically

- Internal processes are also the target of scientific study
- But they are fully controlled by environmental variables just as environmental variables control behavior

A brief timeline of early research on learning

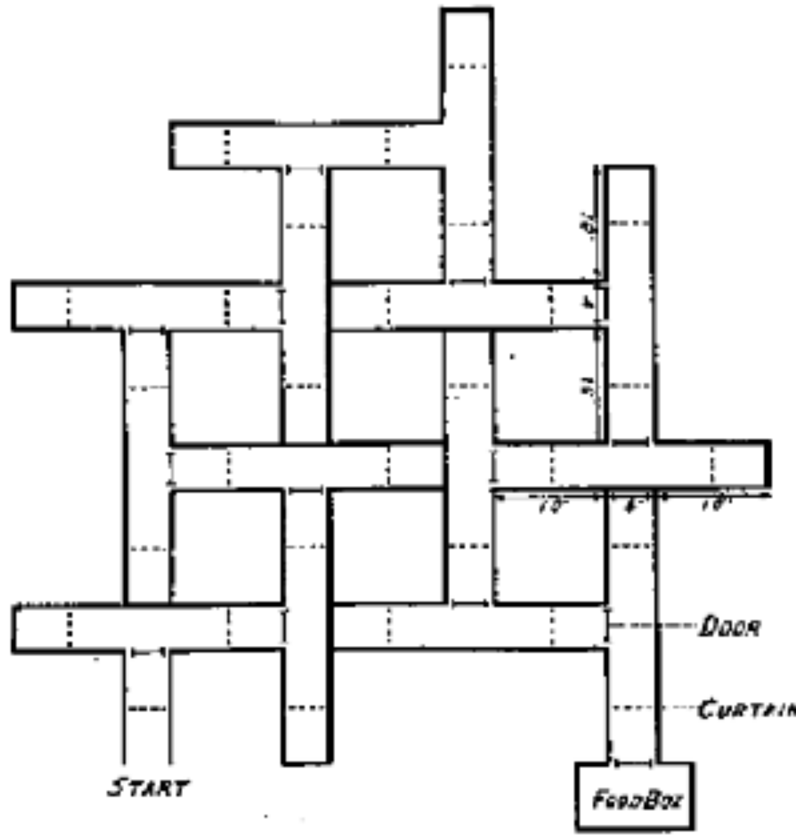


Thorndike (1911)

Pavlov (1927)

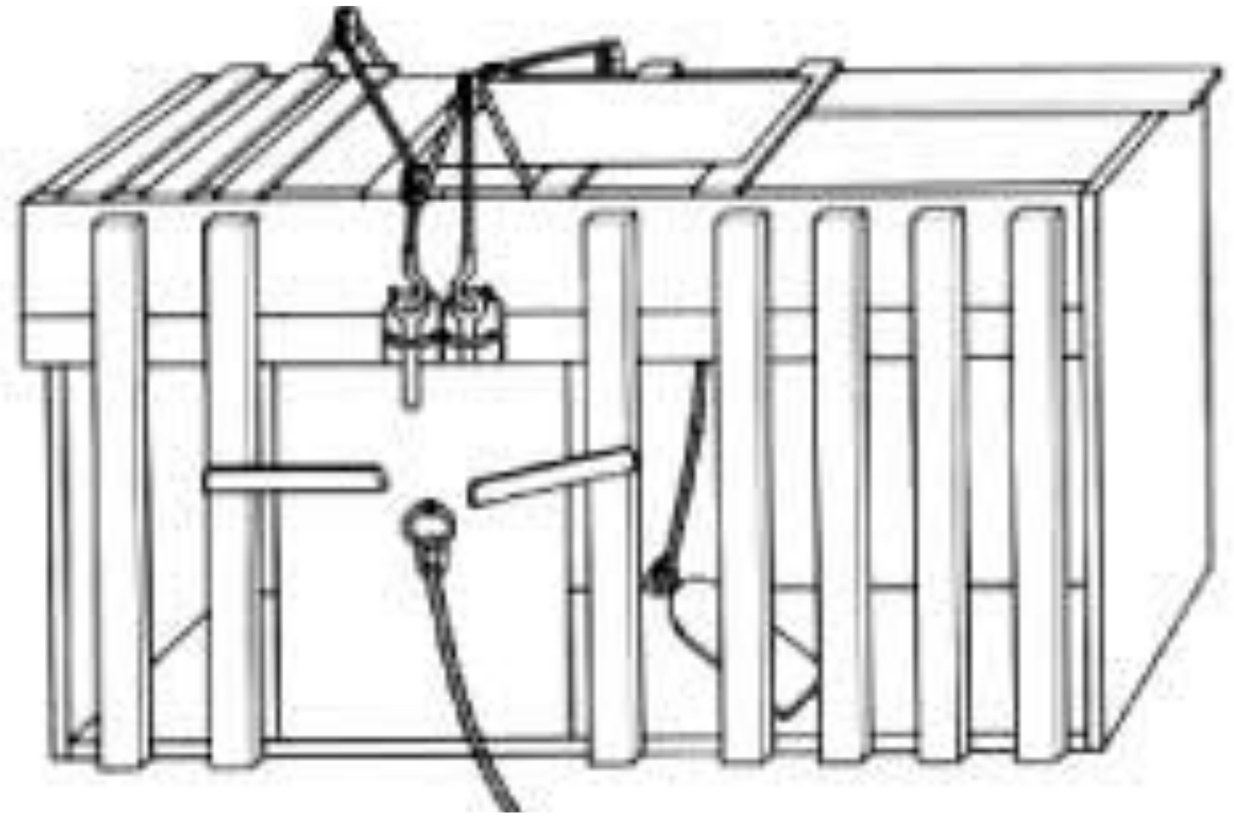
Skinner (1938)

Tolman (1948)



Plan of maze
14-Unit T-Alley Maze
FIG. 1
(From M. H. Elliott, The effect of change of reward on the maze performance of rats. *Univ. Calif. Publ. Psychol.*, 1928, 4, p. 20.)

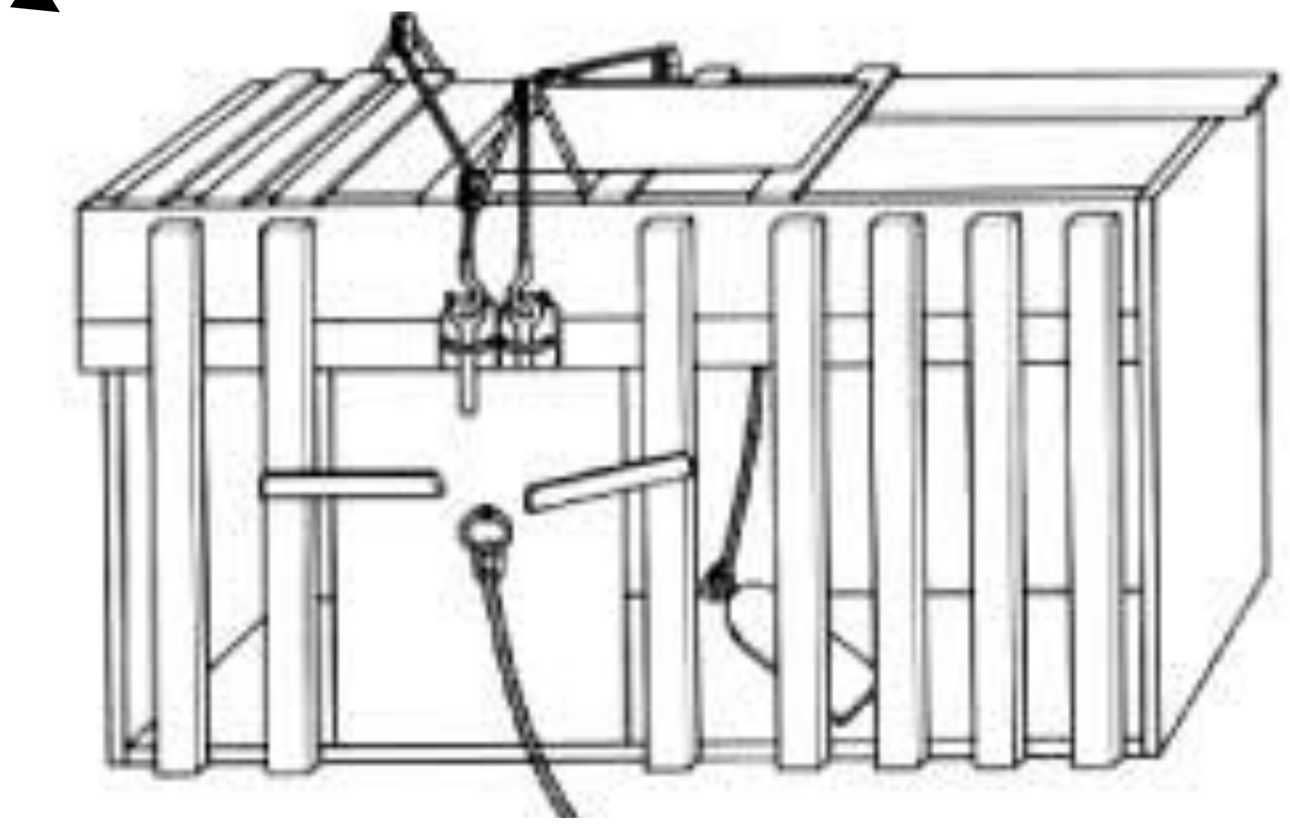
Thorndike’s (1911) Law of Effect



Puzzle Box

Thorndike's (1911) Law of Effect

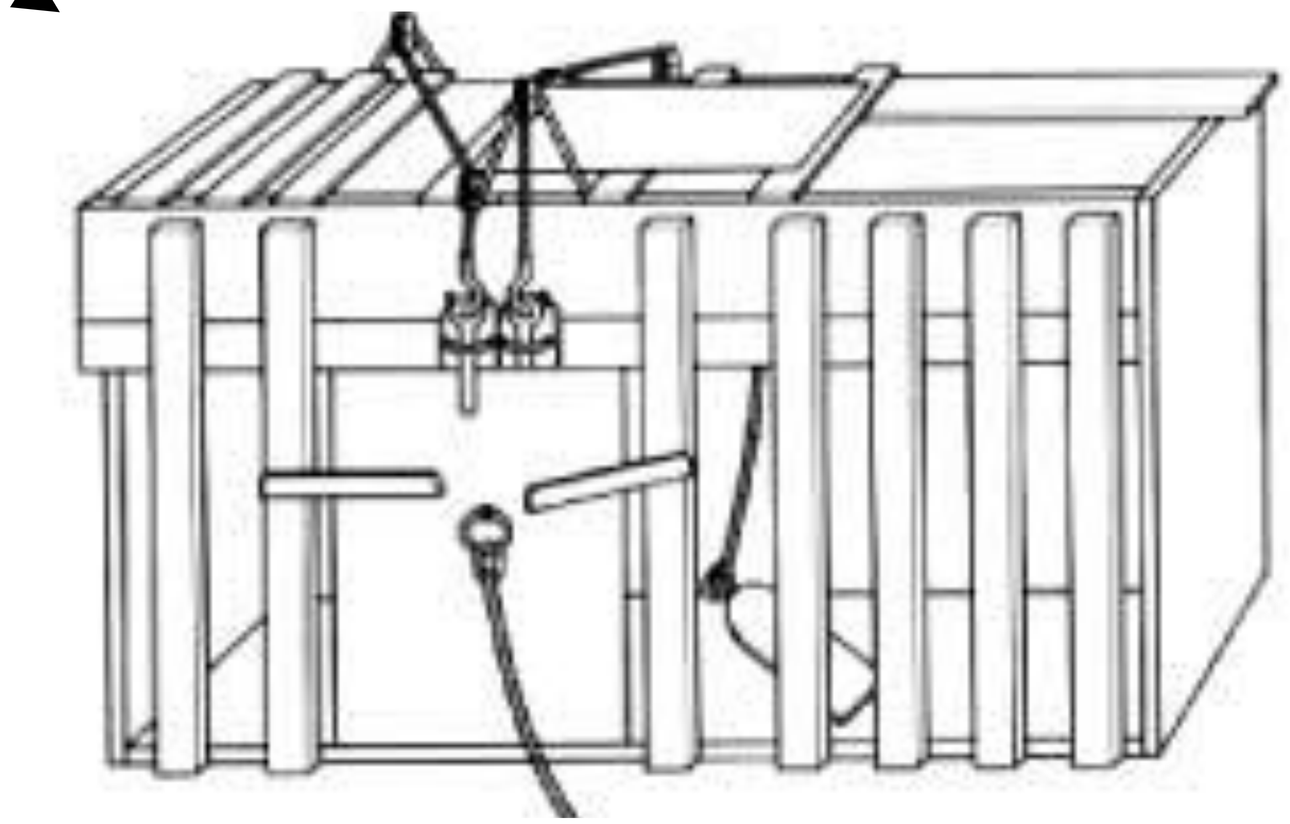
Cat



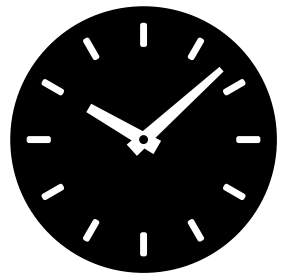
Puzzle Box

Thorndike’s (1911) Law of Effect

Cat



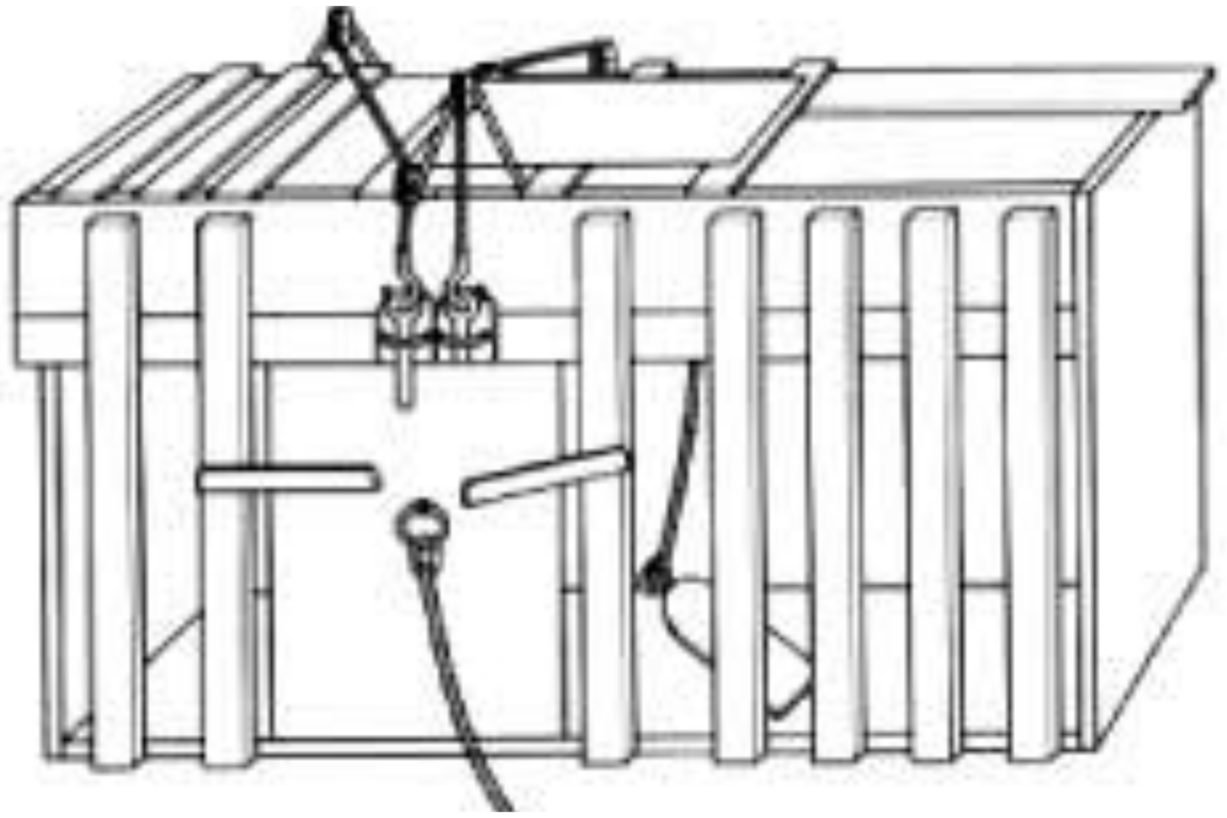
Puzzle Box



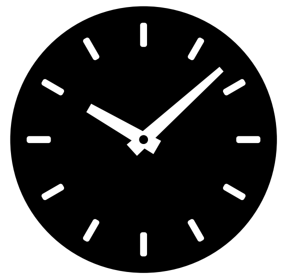
Time to escape

Thorndike’s (1911) Law of Effect

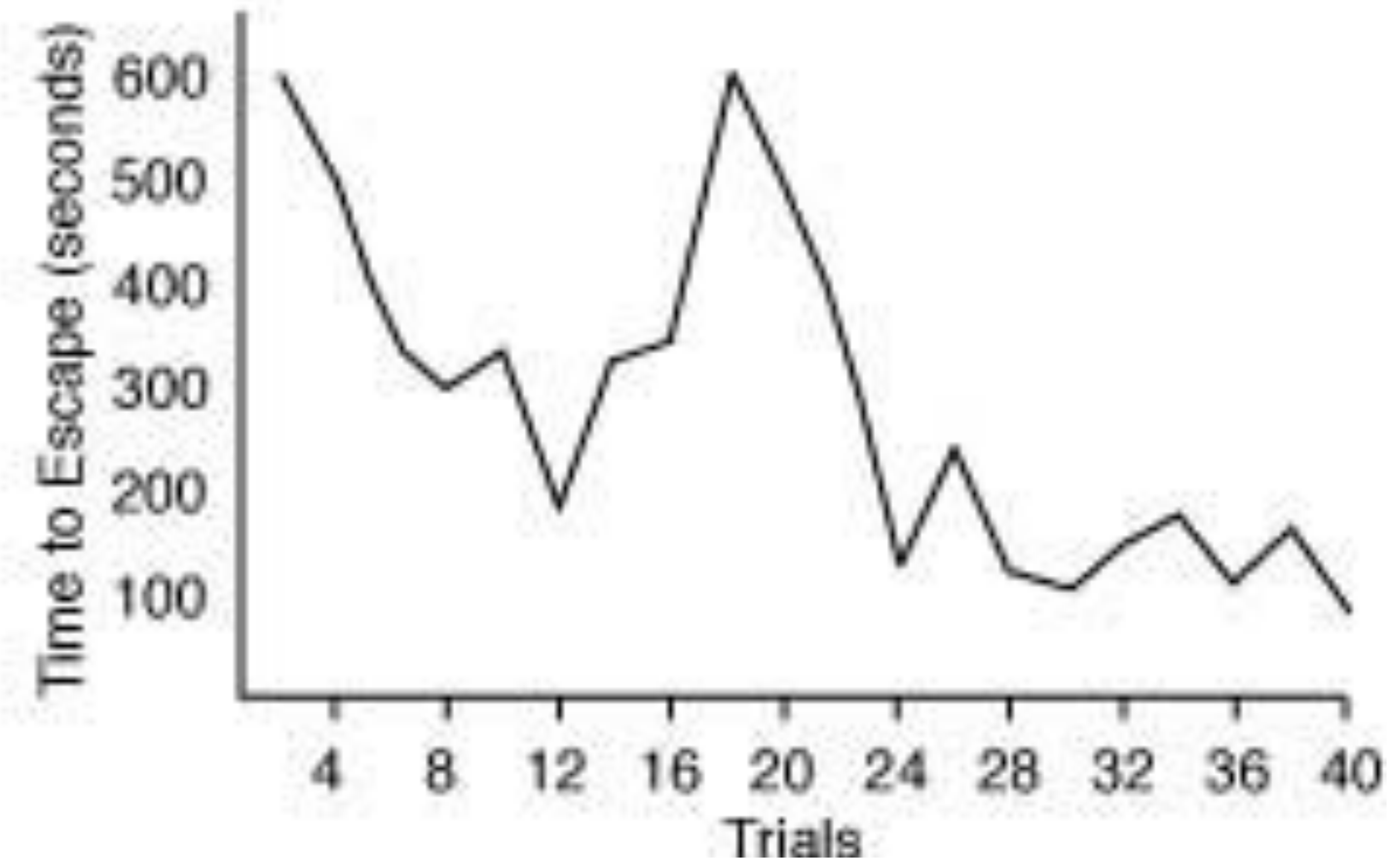
Cat



Puzzle Box



Time to escape



Actions associated with satisfaction are strengthened, while those associated with discomfort become weakened.

Learning as Trial and Error

What are the *benefits*? What are the *limitations*?

Learning as Trial and Error

What are the *benefits*? What are the *limitations*?

Benefits:

- Errors decrease over time
- Openess to trying new solutions
- Basis for all modern reinforcement learning (RL)

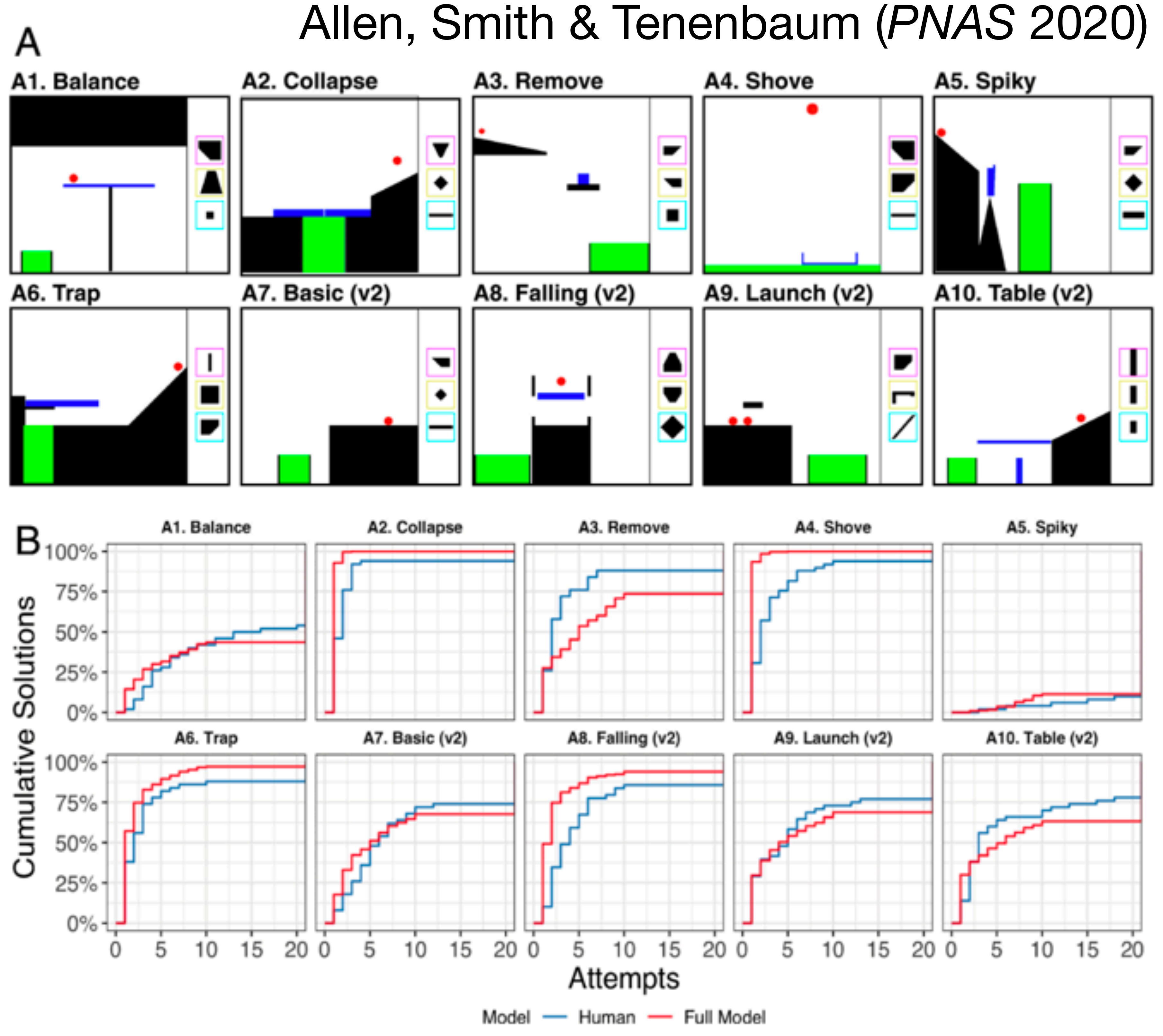
Learning as Trial and Error

What are the *benefits*? What are the *limitations*?

Benefits:

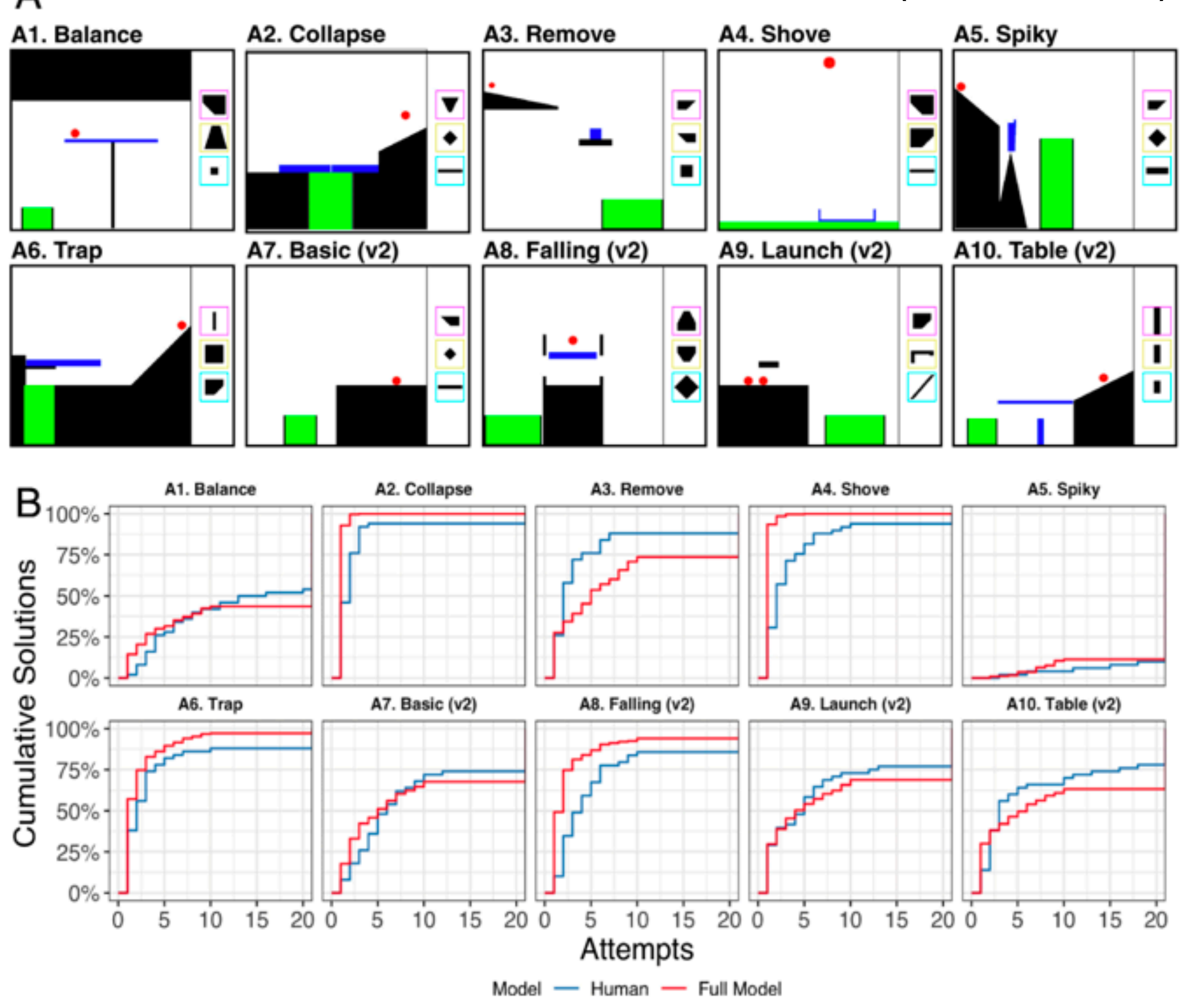
- Errors decrease over time
- Openess to trying new solutions
- Basis for all modern reinforcement learning (RL)

Allen, Smith & Tenenbaum (*PNAS* 2020)



Learning as Trial and Error

Allen, Smith & Tenenbaum (*PNAS* 2020)



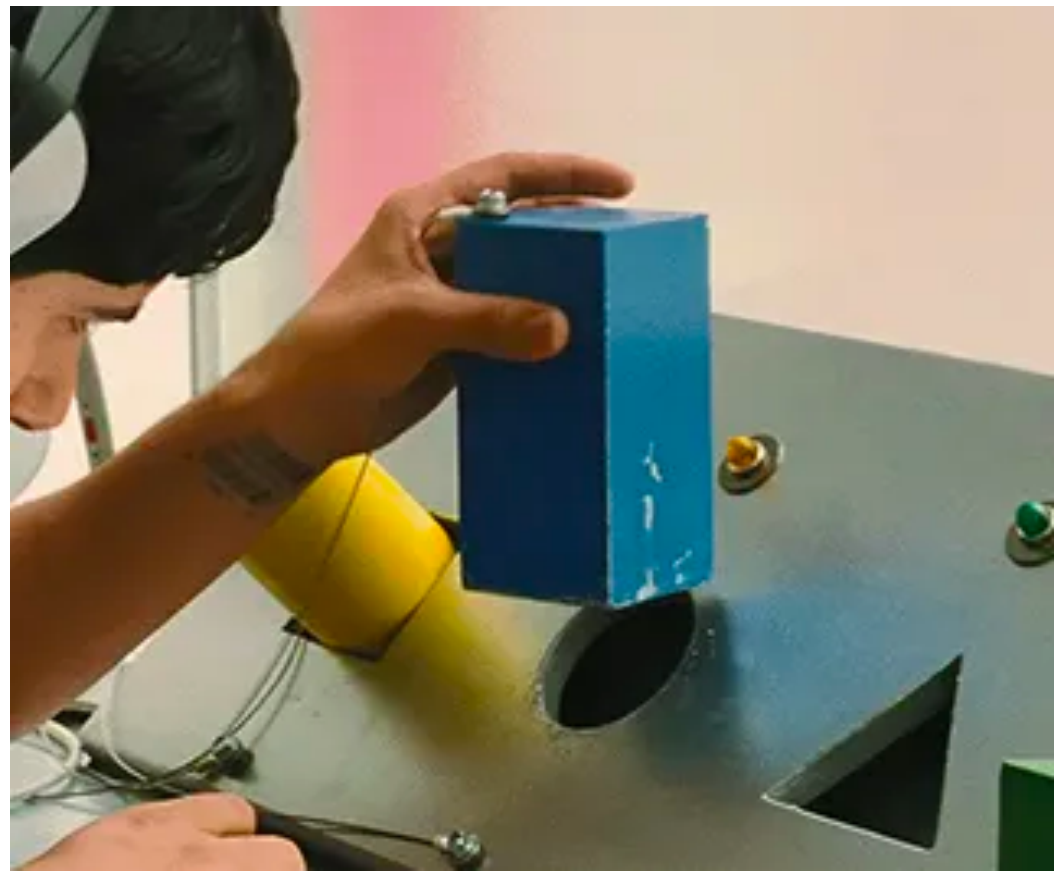
What are the *benefits*? What are the *limitations*?

Benefits:

- Errors decrease over time
- Openess to trying new solutions
- Basis for all modern reinforcement learning (RL)

Limitations:

- Dangerous when some errors are fatal
- Lacks creativity and generalizastion of past solutions
- No formalism between behavior and outcome….

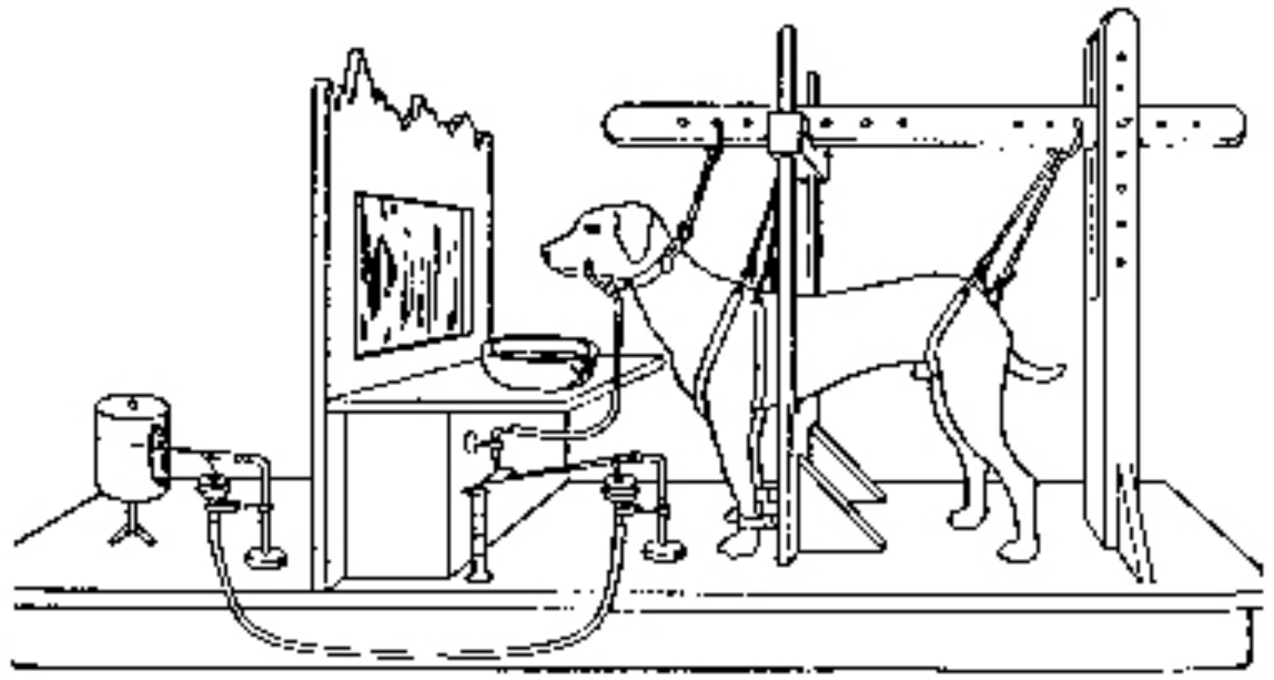


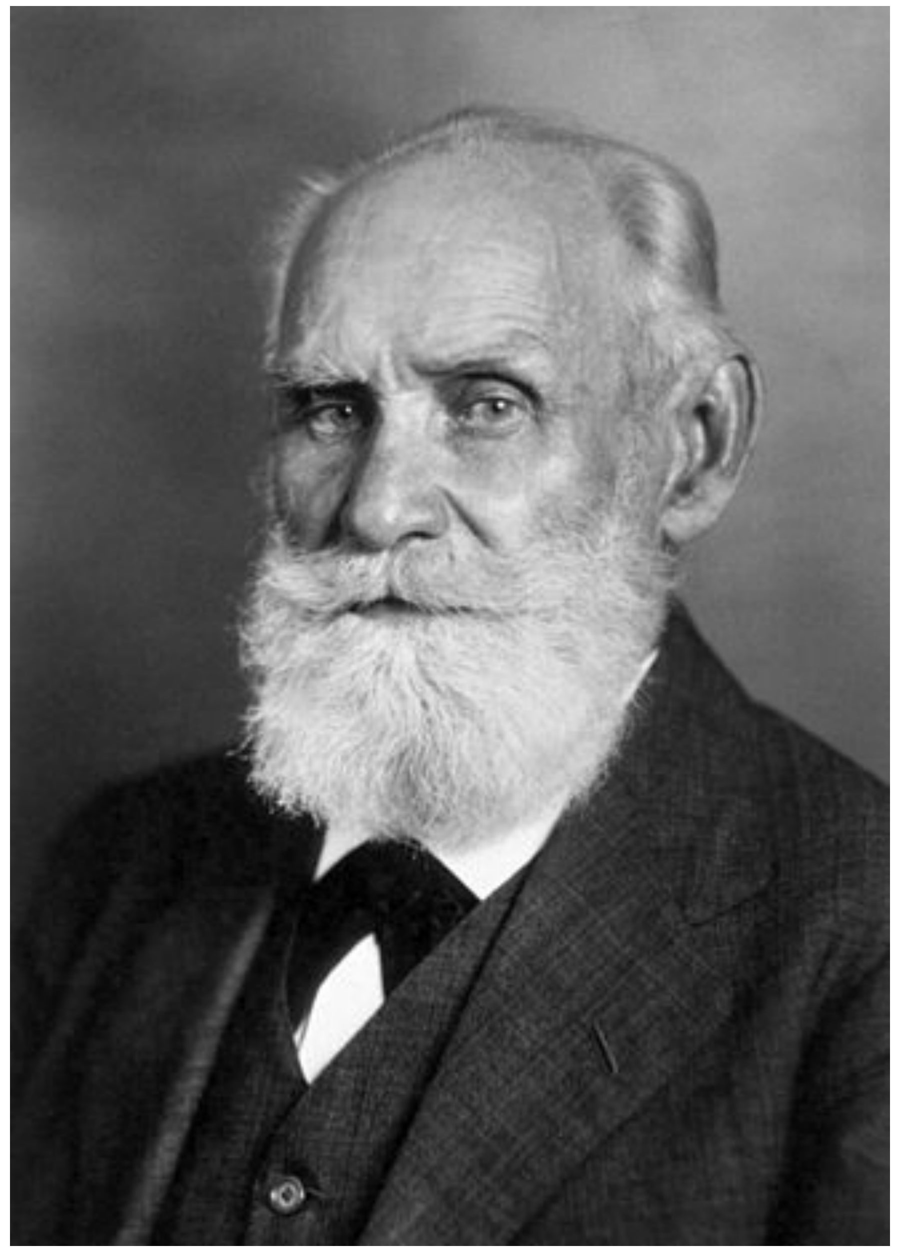
Thorndike’s (1911) Law of Exercise

- In addition to the repeating successful actions, we also repeat actions that we performed in the past
- Habit learning
  - e.g., morning routine, commute to university, studying/exercise routine, etc…
- Behavior is reinforced through frequent connections of stimulus and response

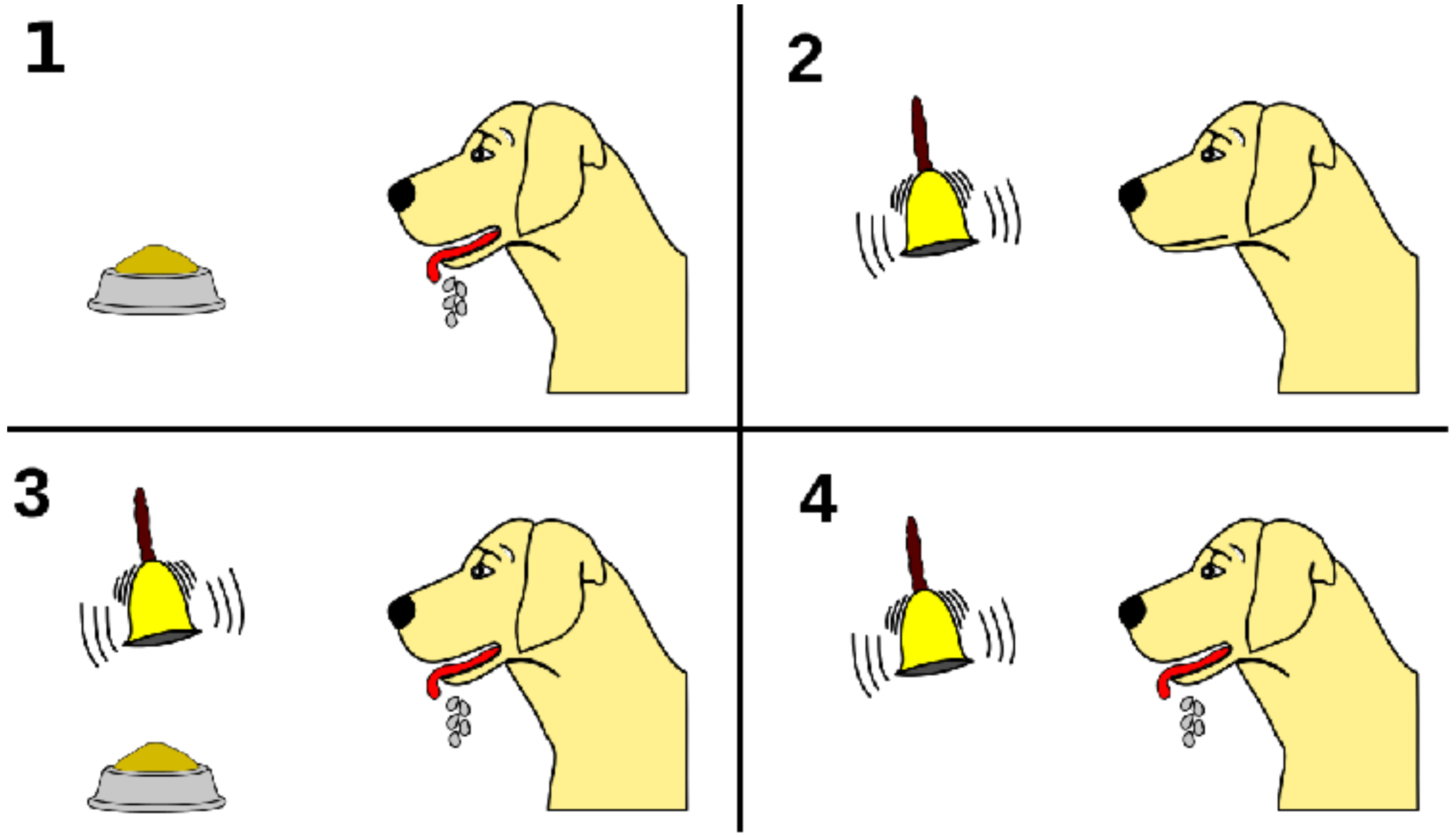
Pavlov’s Dog: Classical conditioning

- Pavlov (1849-1936) was studying digestion in dogs
- The salivation response could be transferred from an unconditioned stimulus (US) —food — to a conditioned stimulus (CS) —the ringing of a bell
  - 1) the dog naturally salivates when presented with food and 2) has no initial response to a bell
  - 3) when the dog is trained to associate a bell with the delivery of food, 4) it learns to anticipate food when a bell rings and begins to salivate





Ivan Pavlov



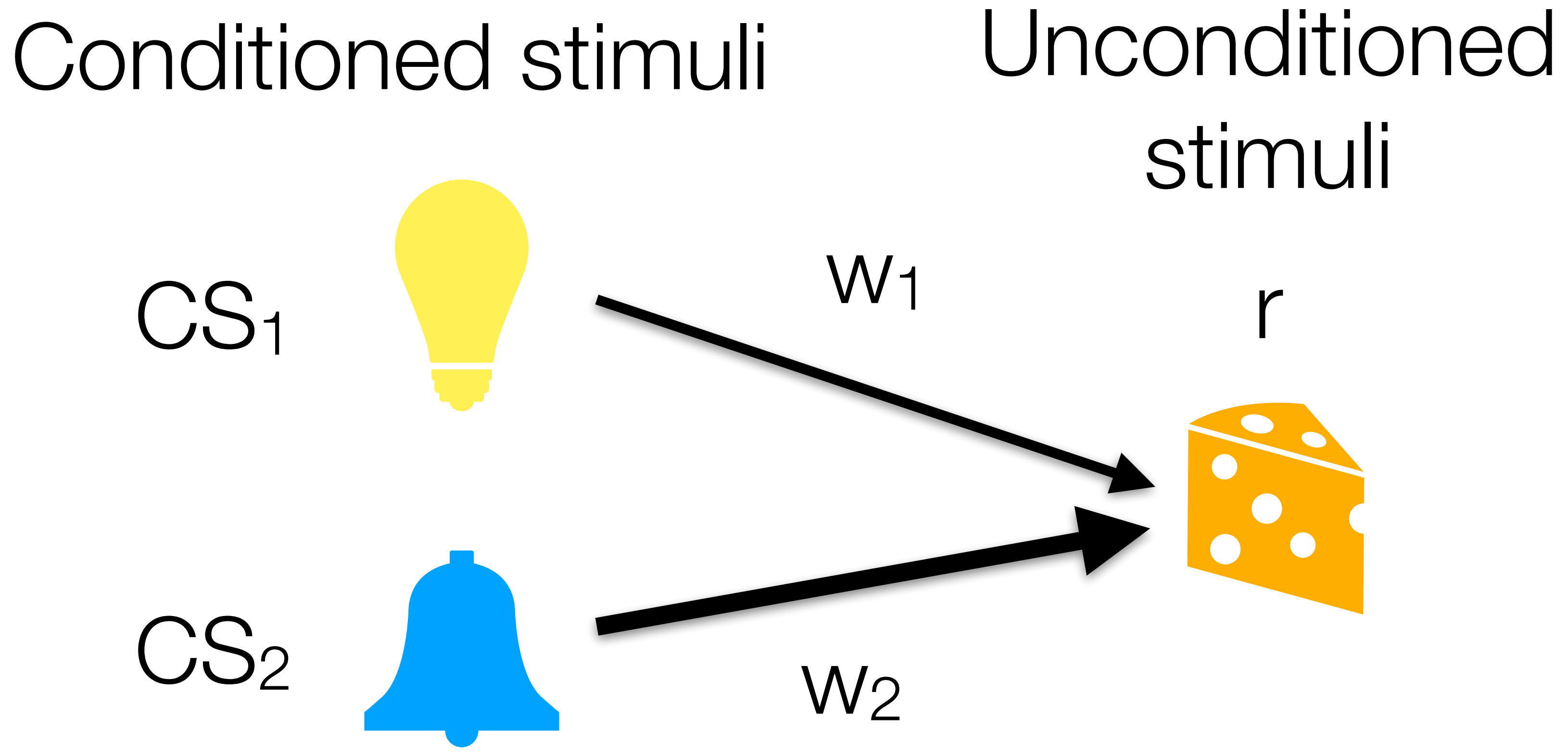
Key ideas: Classical conditioning

Pavlovian responses are driven by outcome expectations

Learning is driven by reward predictions and (as we will see) shaped by prediction error

Cues compete for shared credit in predicting reward outcomes

Rescorla-Wagner



Rescorla-Wagner model

(Bush & Mosteller, 1951; Rescorla & Wagner, 1972)

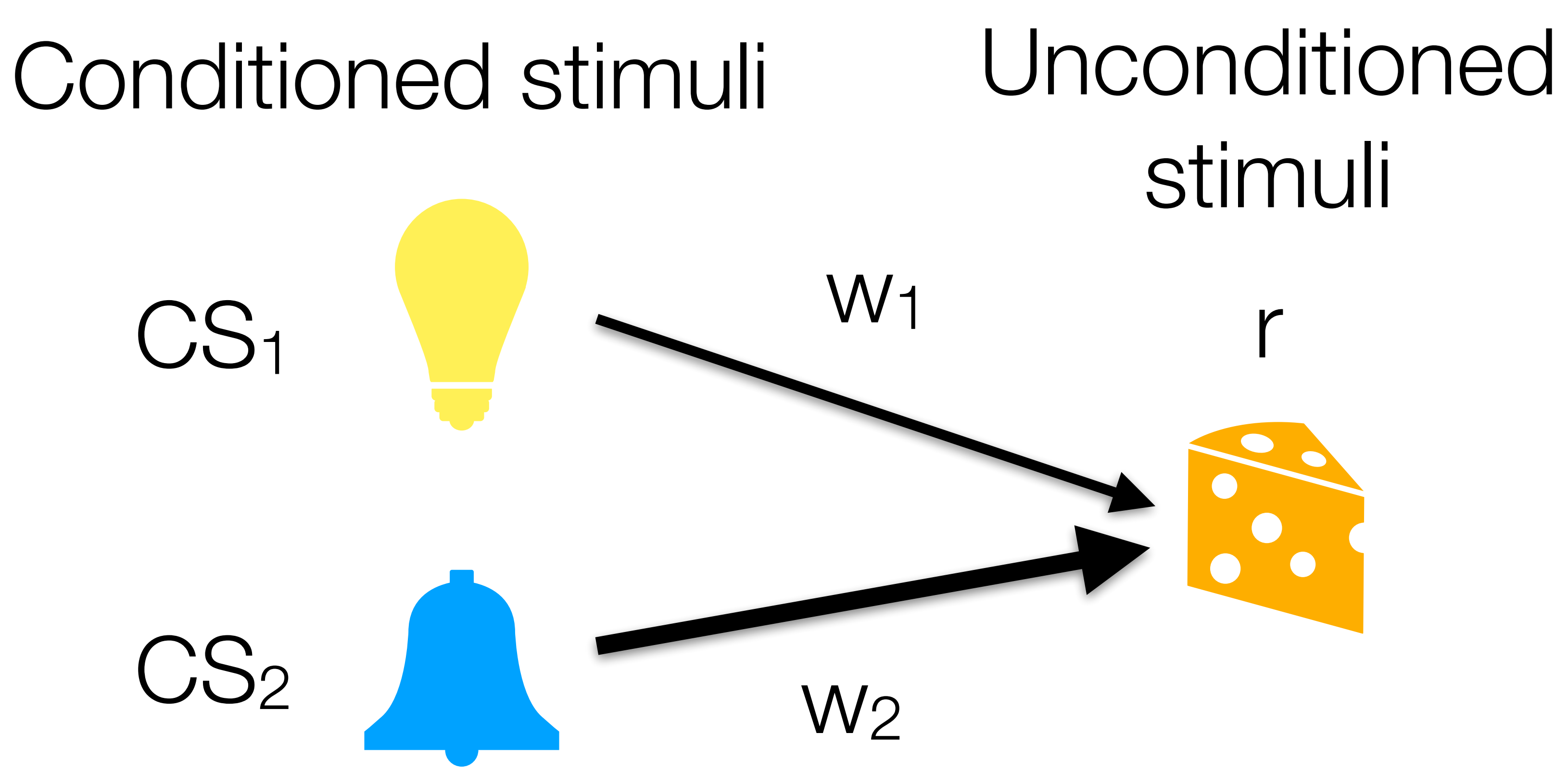
Reward prediction

$$\hat{r}_t = \sum_i \mathrm{CS}_i^t w_i$$

Weight update

$$w_i \leftarrow w_i + \eta(r_t - \hat{r}_t)$$

Rescorla-Wagner



Rescorla-Wagner model

(Bush & Mosteller, 1951; Rescorla & Wagner, 1972)

Reward prediction

$$\hat{r}_t = \sum_i \mathrm{CS}_i^t w_i$$

Reward expectation

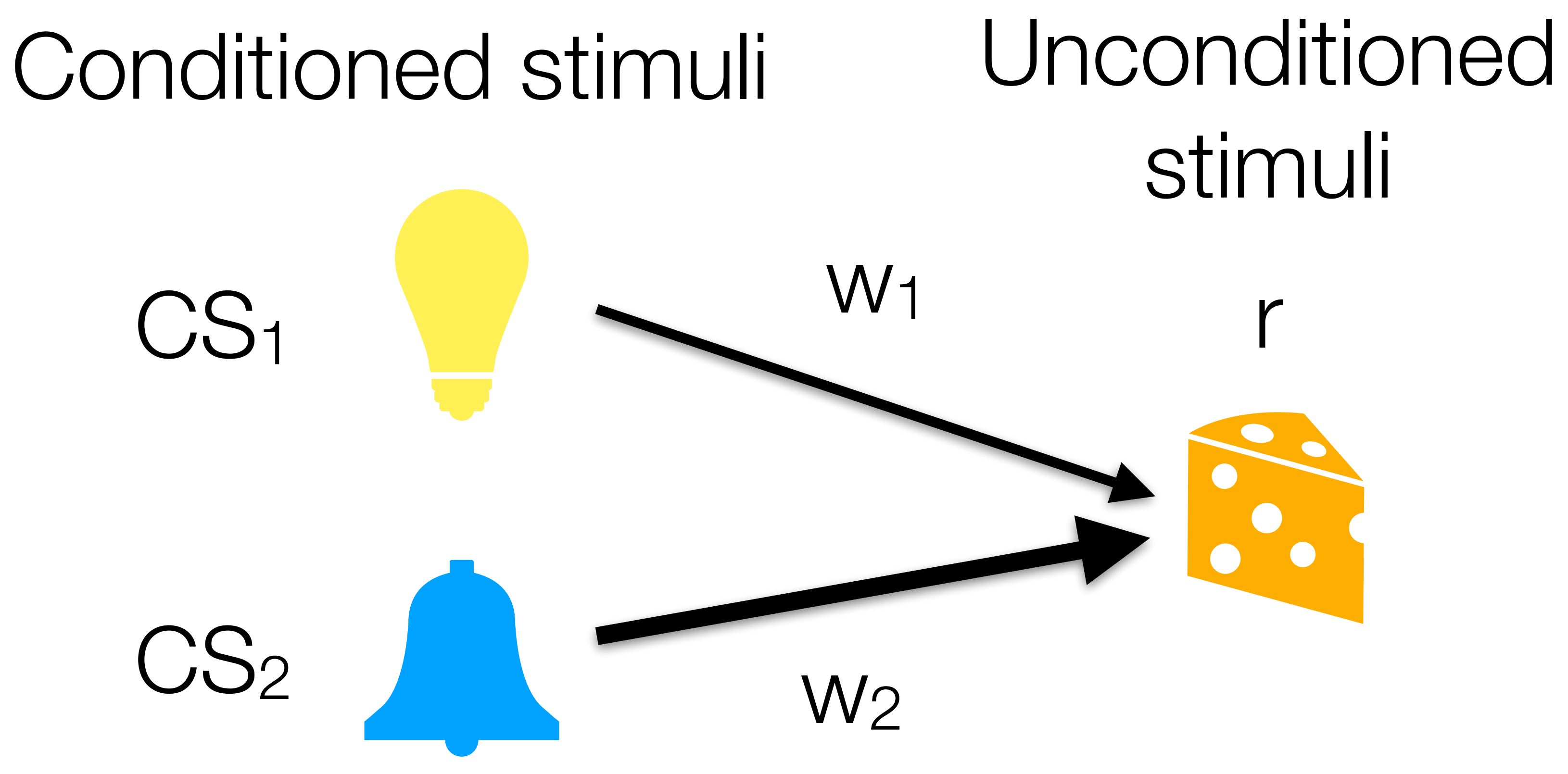
CS i on trial t

Associative strength

Weight update

$$w_i \leftarrow w_i + \eta(r_t - \hat{r}_t)$$

Rescorla-Wagner



Rescorla-Wagner model

(Bush & Mosteller, 1951; Rescorla & Wagner, 1972)

Reward prediction

$$\hat{r}_t = \sum_i \mathrm{CS}_i^t w_i$$

Reward expectation

CS i on trial t

Associative strength

Weight update

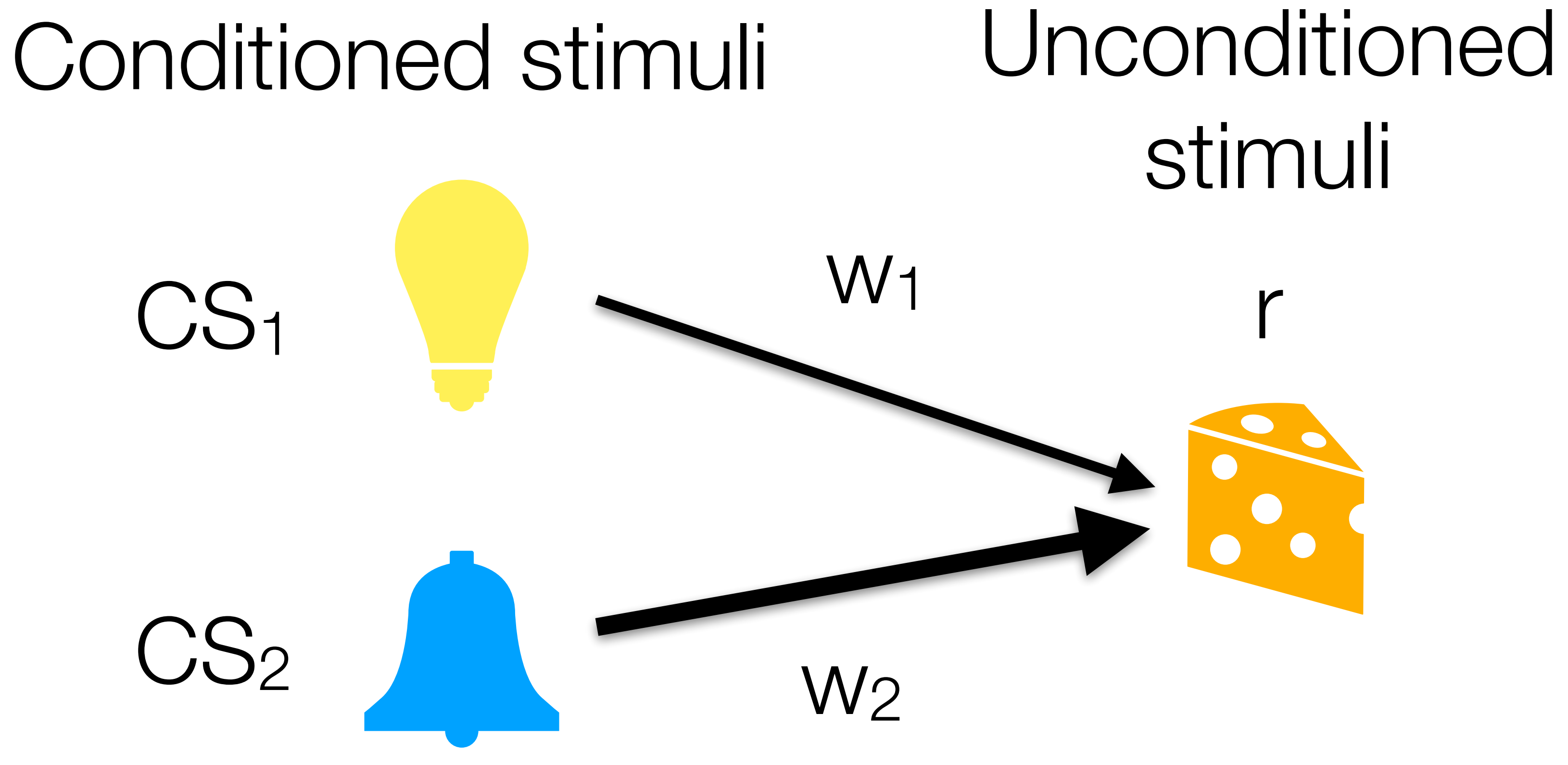
$$w_i \leftarrow w_i + \eta(r_t - \hat{r}_t)$$

Learning rate

Observed outcome

Predicted outcome

Rescorla-Wagner



Rescorla-Wagner model

(Bush & Mosteller, 1951; Rescorla & Wagner, 1972)

Reward prediction

$$\hat{r}_t = \sum_i \mathrm{CS}_i^t w_i$$

Reward expectation

CS i on trial t

Associative strength

Weight update

$$w_i \leftarrow w_i + \eta(r_t - \hat{r}_t)$$

Learning rate

Observed outcome

Predicted outcome

δ

Reward prediction error (RPE)

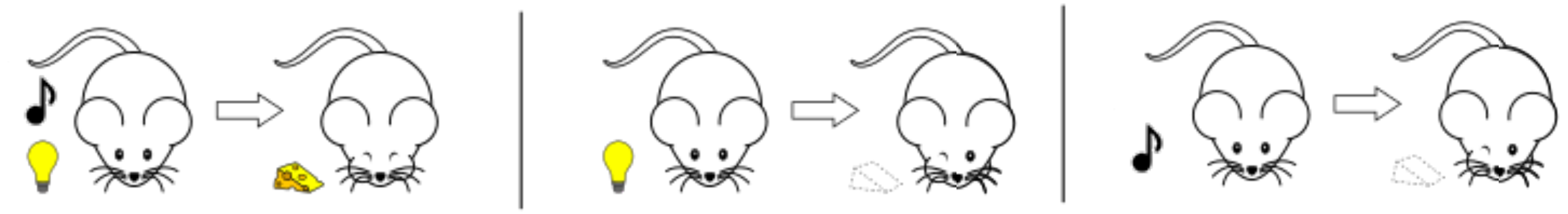
The delta-rule of learning:

- Learning occurs only when events violate expectations ($\delta \neq 0$)
- The magnitude of the error corresponds to how much we update our beliefs

Implications: Cue competition

If multiple stimuli cues predict an outcome, they will share credit

Overshadowing ? ?

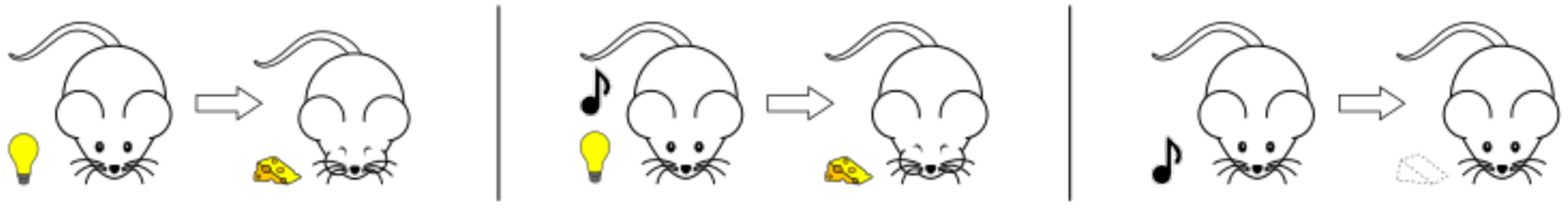


Overshadowing:

- If sound and light are both associated reward, then presenting individual cues will result in weaker responses

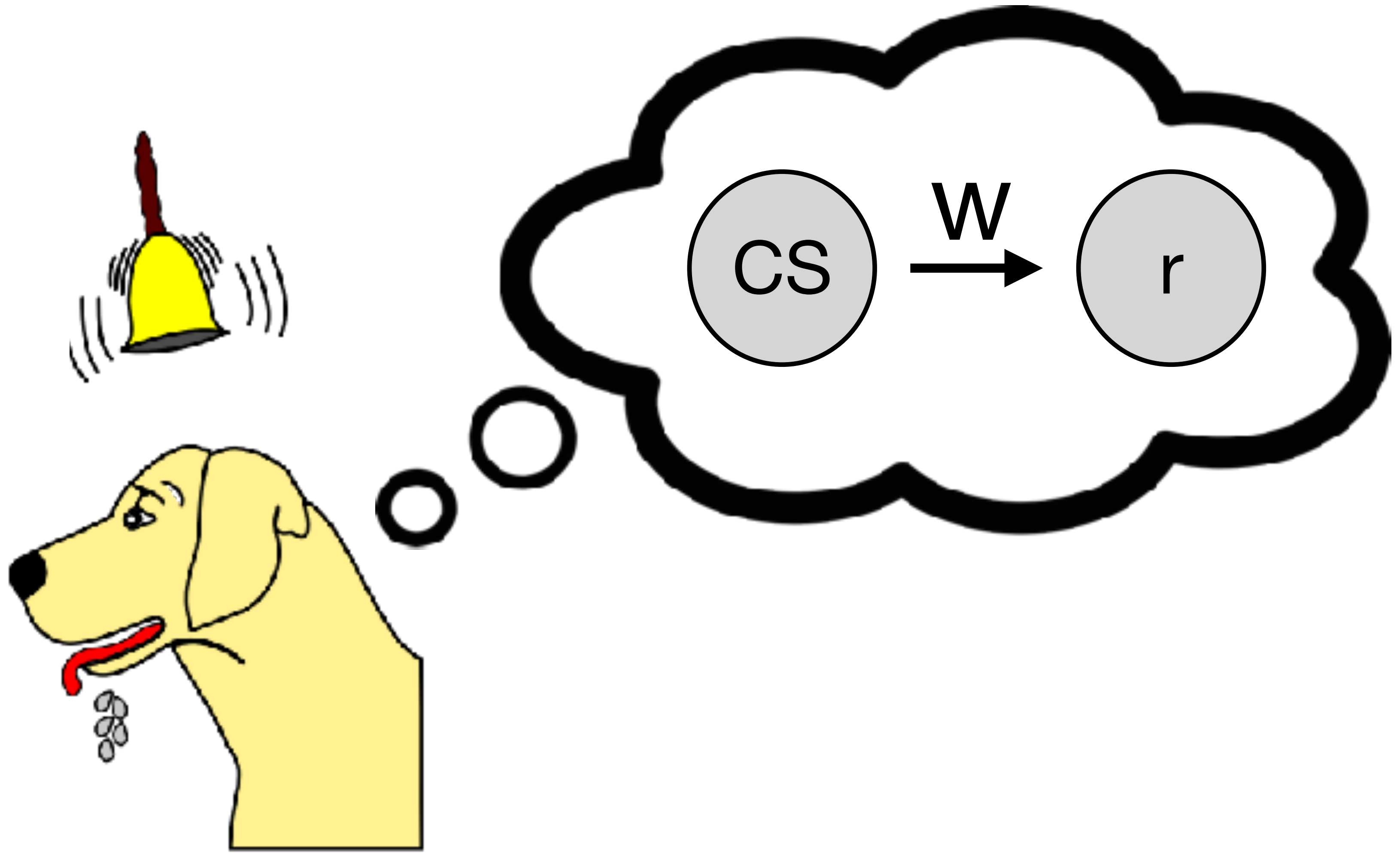
Blocking

- If light is first associated with reward, and then later both light and sound, there will be less associating of sound with reward than if sound were conditioned alone



Reward learning as refining an internal representation of the world

- Internal hypotheses about how sensory data $\mathscr{D}$ were generated
- The parameters w are unknown and must be estimated to maximize the likelihood of the data $P(\mathscr{D} \mid w)$
  - This is known as maximum likelihood estimation (MLE):
    $$\hat{w} = \arg\max_{w} P(\mathscr{D} \mid w)$$
- Under linear Gaussian assumptions, RW implements a MLE through gradient descent



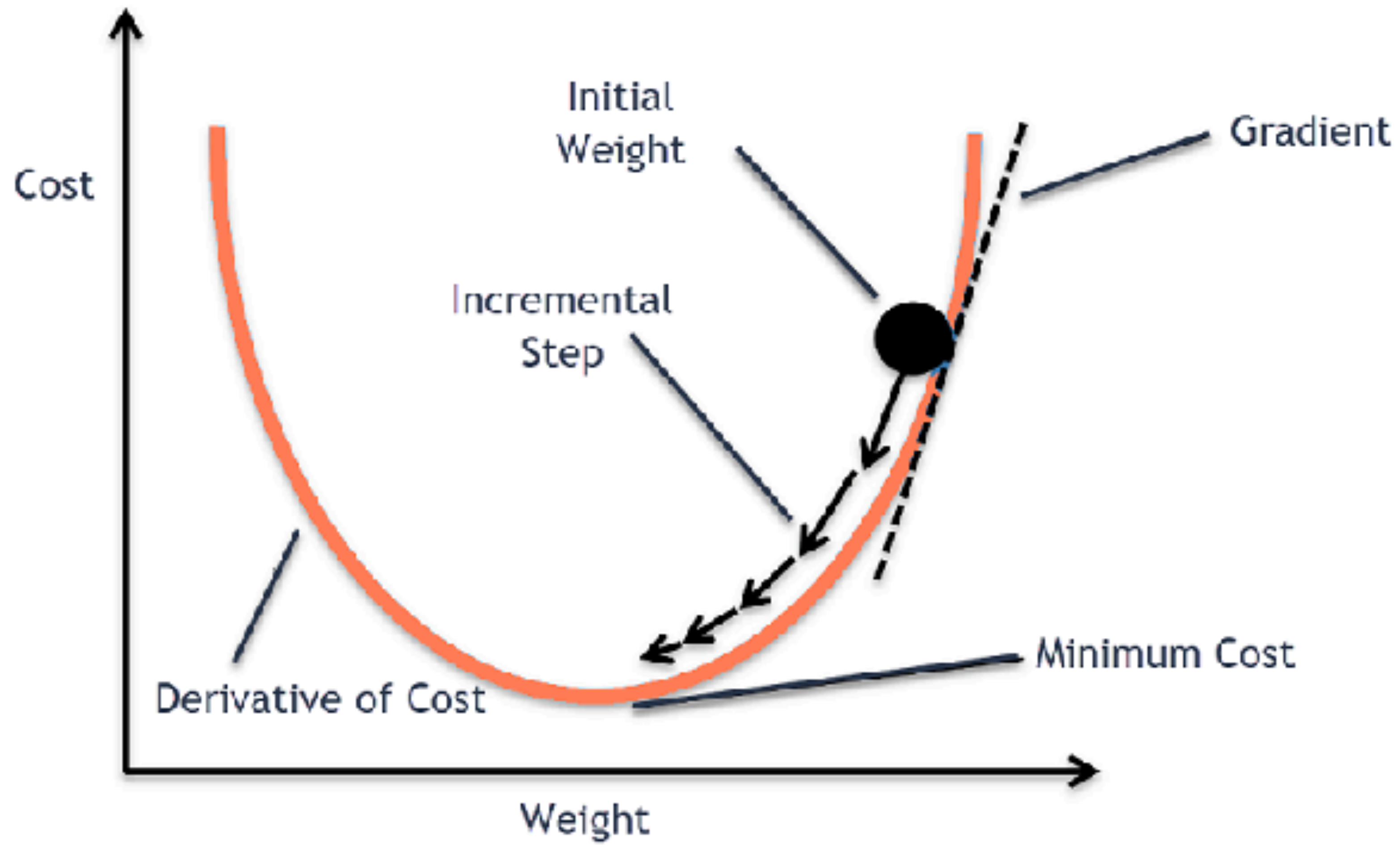
Loss function

$$\mathscr{L}(w) = -\log P(\mathscr{D} \mid w)$$

Gradient update

$$\Delta \hat{w}_i \propto -\nabla_{w_i} \mathscr{L}(w) = \mathrm{CS}_i(r - \hat{r})$$

Gradient descent



Operant conditioning

- Building off of Thorndike's Law of Effect, operant conditioning studies how rewards shape the animal's behavior
- Unlike classical conditioning, operant conditioning describes the *active* selection of actions in response to rewards/punishments, rather than only their *passive* association with stimuli
- This allows us to describe how animals learn to perform *actions* (conditioned on stimuli) that are predictive of reward

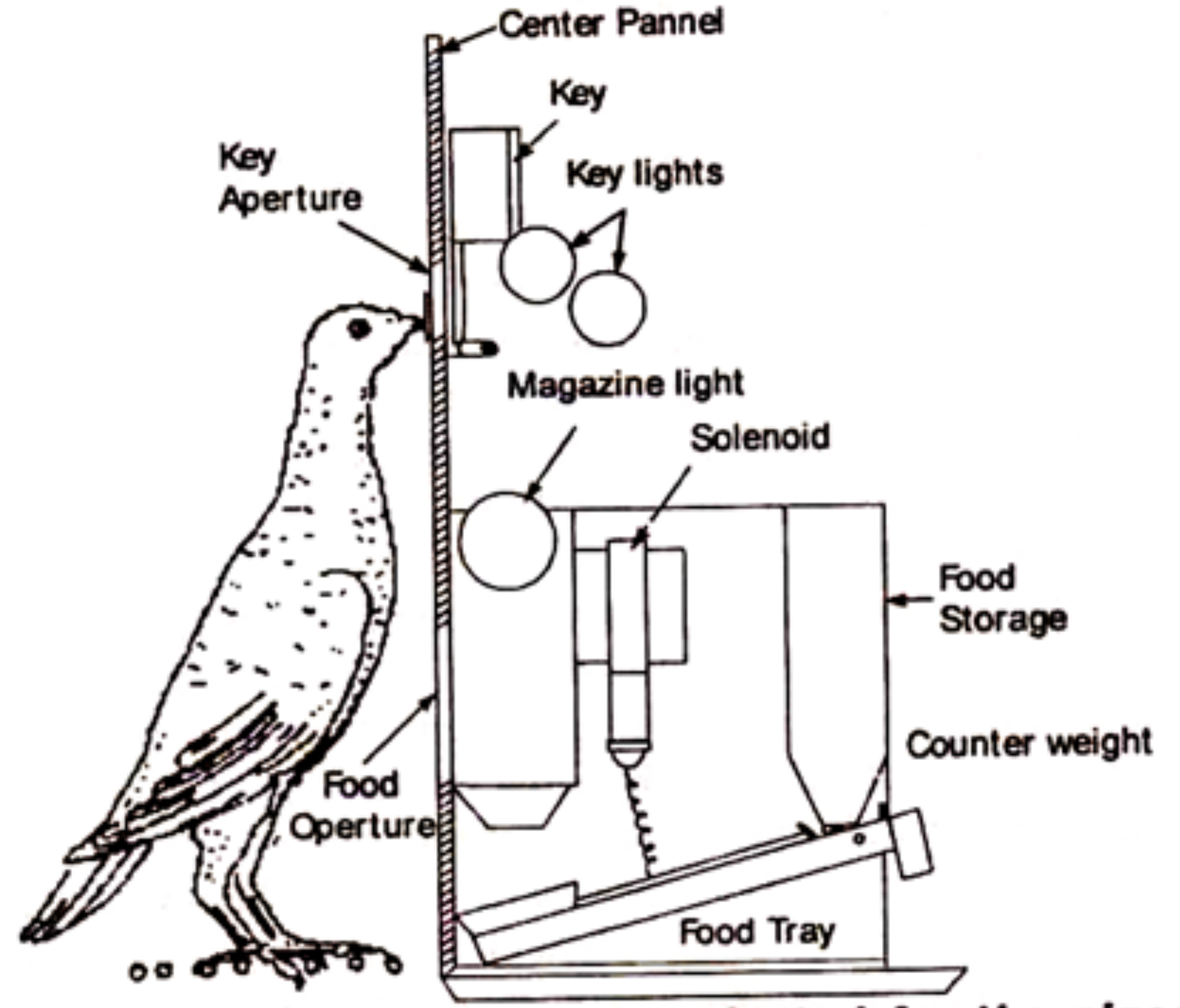
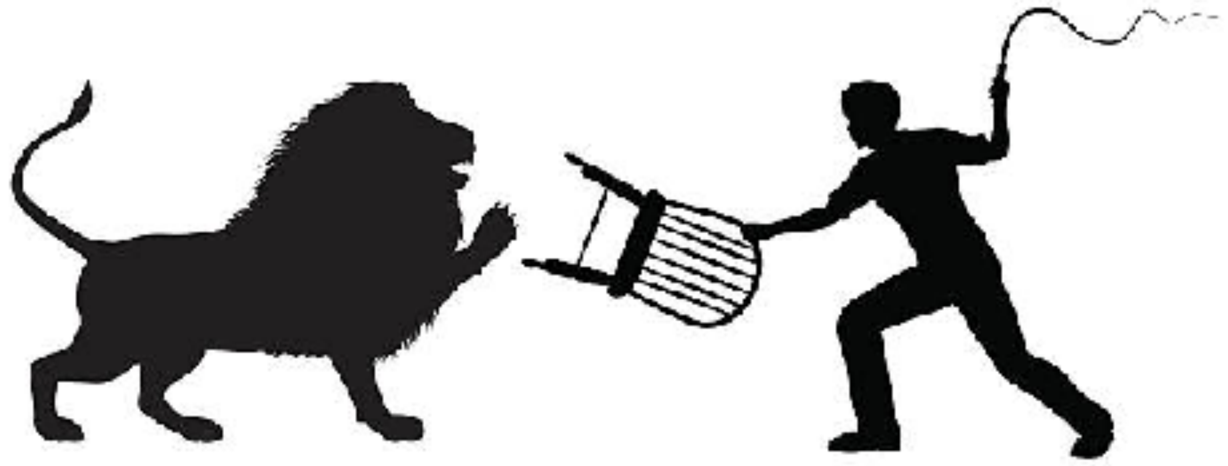


Illustration. Skinner box as adapted for the pigeon.

Behavioral Shaping



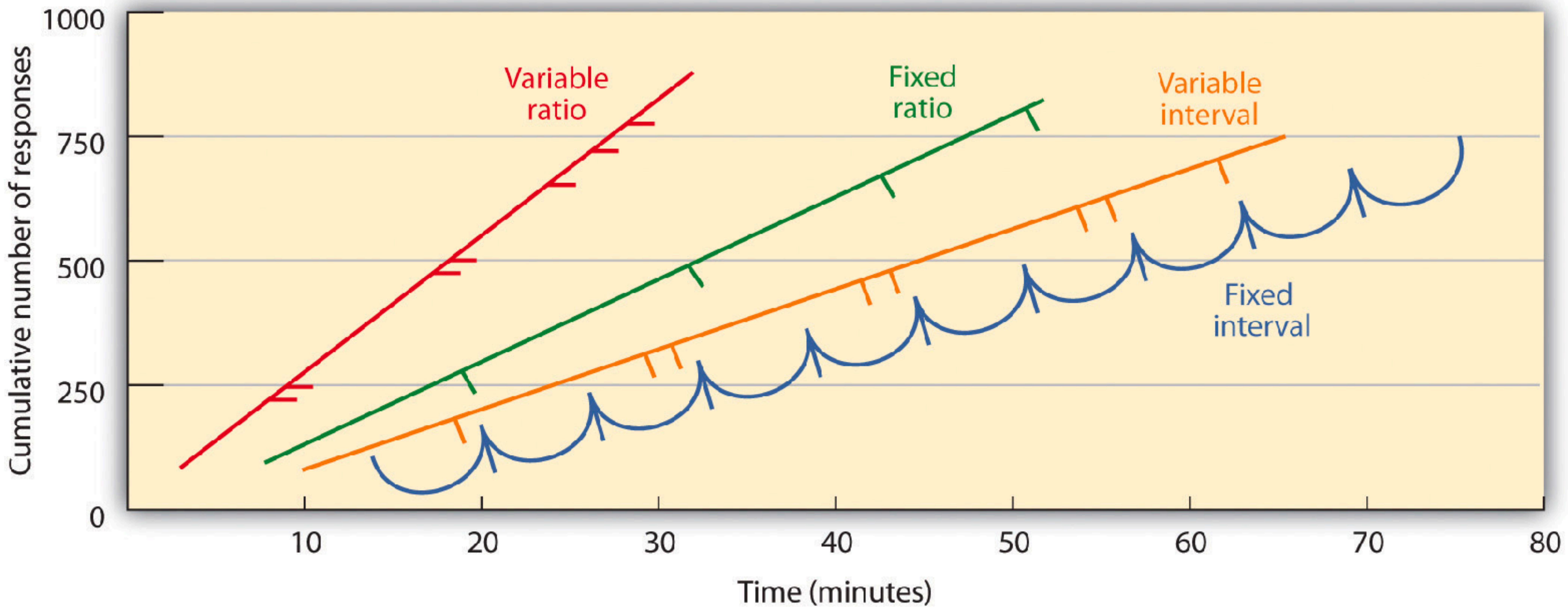
- Reward learning is slow when the space of possible actions is very large
- To encourage exploration towards the target behavior, we can use **shaping** by adding rewards for smaller, intermediate steps,
  - Technique pioneered by Skinner to train a target behavior by rewarding *successive approximations*

1. Reinforce any response that resembles the desired behavior
2. Iteratively reinforce responses that more selectively resemble the target behavior, and remove reinforcement from previously reinforced responses (causing *extinction*)

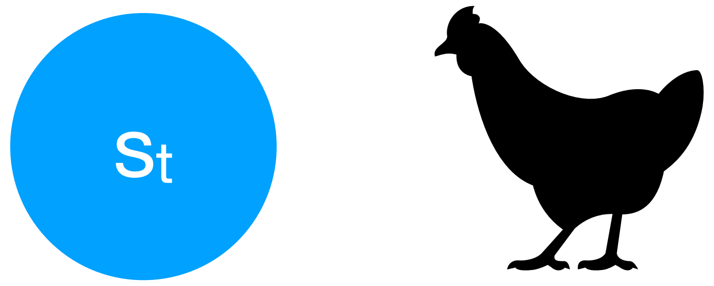
Reinforcement schedules

Different reinforcement schedules yield different response patterns

- **Interval** reward(time) vs. **Ratio** reward(responses)
- **Variable** vs. **Fixed**



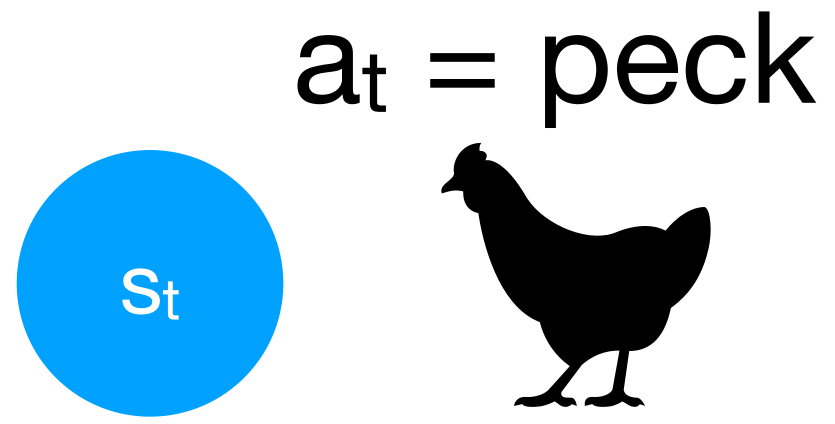
a_t = peck



From Rescorla-Wagner to Q-learning

Rescorla-Wagner model
(Bush & Mosteller, 1951; Rescorla & Wagner, 1972)

Q-learning
(Watkins, 1989)



From Rescorla-Wagner to Q-learning

Rescorla-Wagner model
(Bush & Mosteller, 1951; Rescorla & Wagner, 1972)

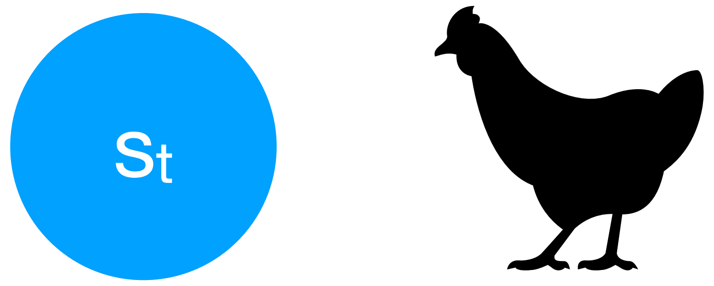
$$\hat{r}_t = \sum_i \mathrm{CS}_i^t w_i$$

Reward estimate

Q-learning
(Watkins, 1989)

$$Q(s_t, a_t)$$

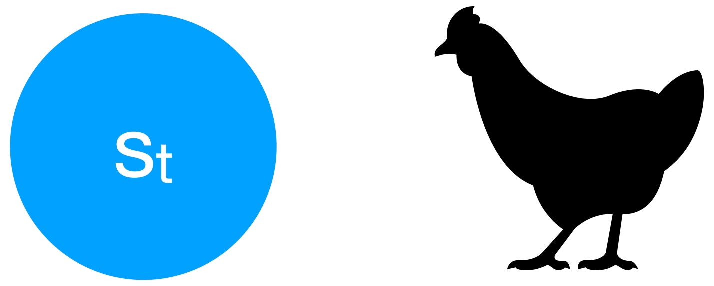
a_t = peck



From Rescorla-Wagner to Q-learning

Rescorla-Wagner model (Bush & Mosteller, 1951; Rescorla & Wagner, 1972)		Q-learning (Watkins, 1989)
$\hat{r}_t = \sum_i \mathrm{CS}_i^t w_i$	Reward estimate	$Q(s_t, a_t)$
$w_i \leftarrow w_i + \eta(r_t - \hat{r}_t)$	Prediction error learning	$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta[r - Q(s_t, a_t)]$

a_t = peck

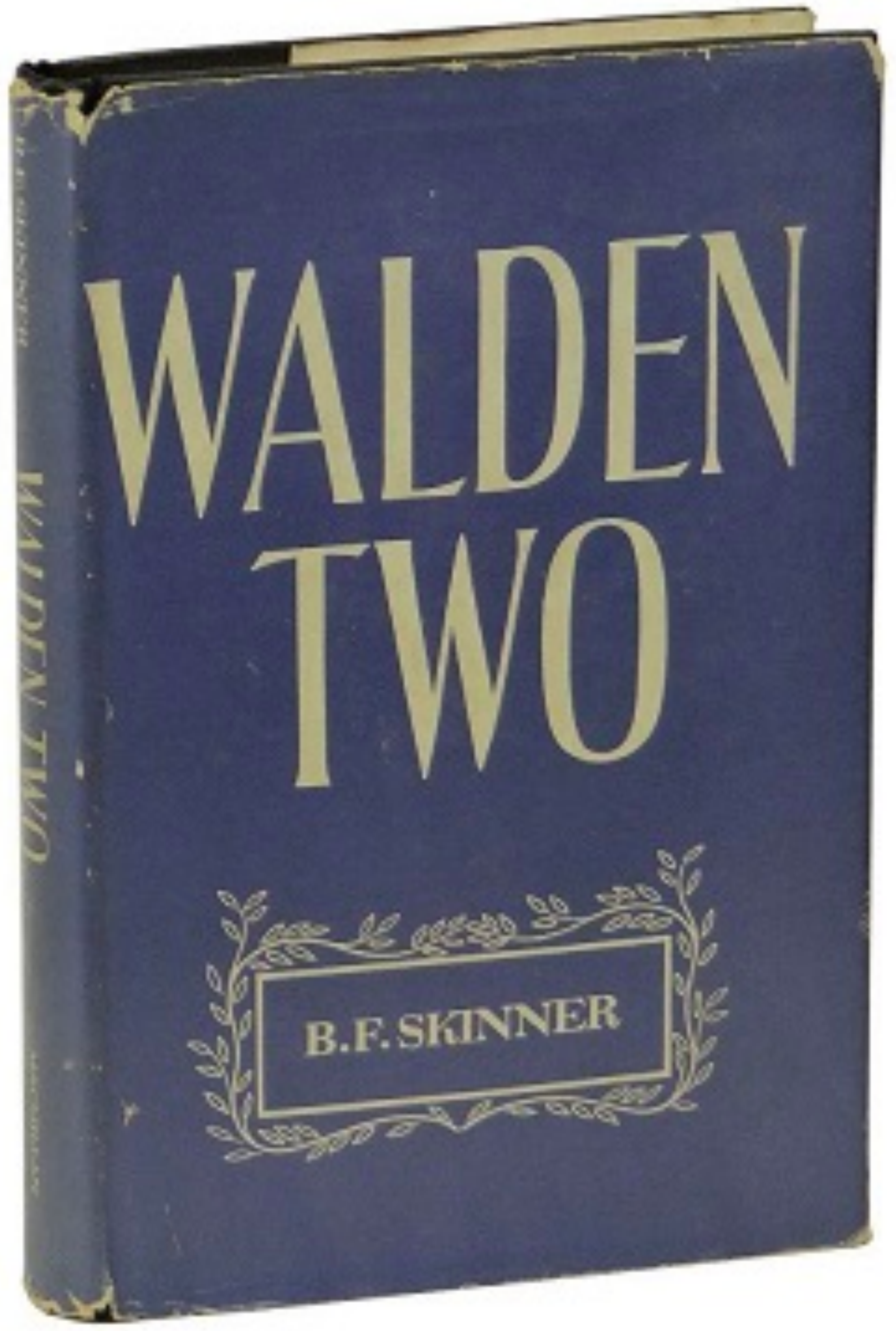


From Rescorla-Wagner to Q-learning

Rescorla-Wagner model (Bush & Mosteller, 1951; Rescorla & Wagner, 1972)		Q-learning (Watkins, 1989)
$\hat{r}_t = \sum_i \text{CS}_i^t w_i$	Reward estimate	$Q(s_t, a_t)$
$w_i \leftarrow w_i + \eta(r_t - \hat{r}_t)$	Prediction error learning	$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta[r - Q(s_t, a_t)]$
?	Behavioral policy	$\pi(a_t \mid s_t) \propto \exp(Q(s_t, a_t)/\tau)$

Dark side of Behavioralism

- Walden Two (1948) describes a Utopia, where behavioral engineering is used to shape a perfect society
  - From childhood, citizens are crafted through rewards and punishment into the ideal citizens and to value benefit for the common good
  - Rejection of free will, and has been criticized as creating a "perfectly efficient anthill"
- Is intelligence just learning to acquire reward and avoiding punishment?



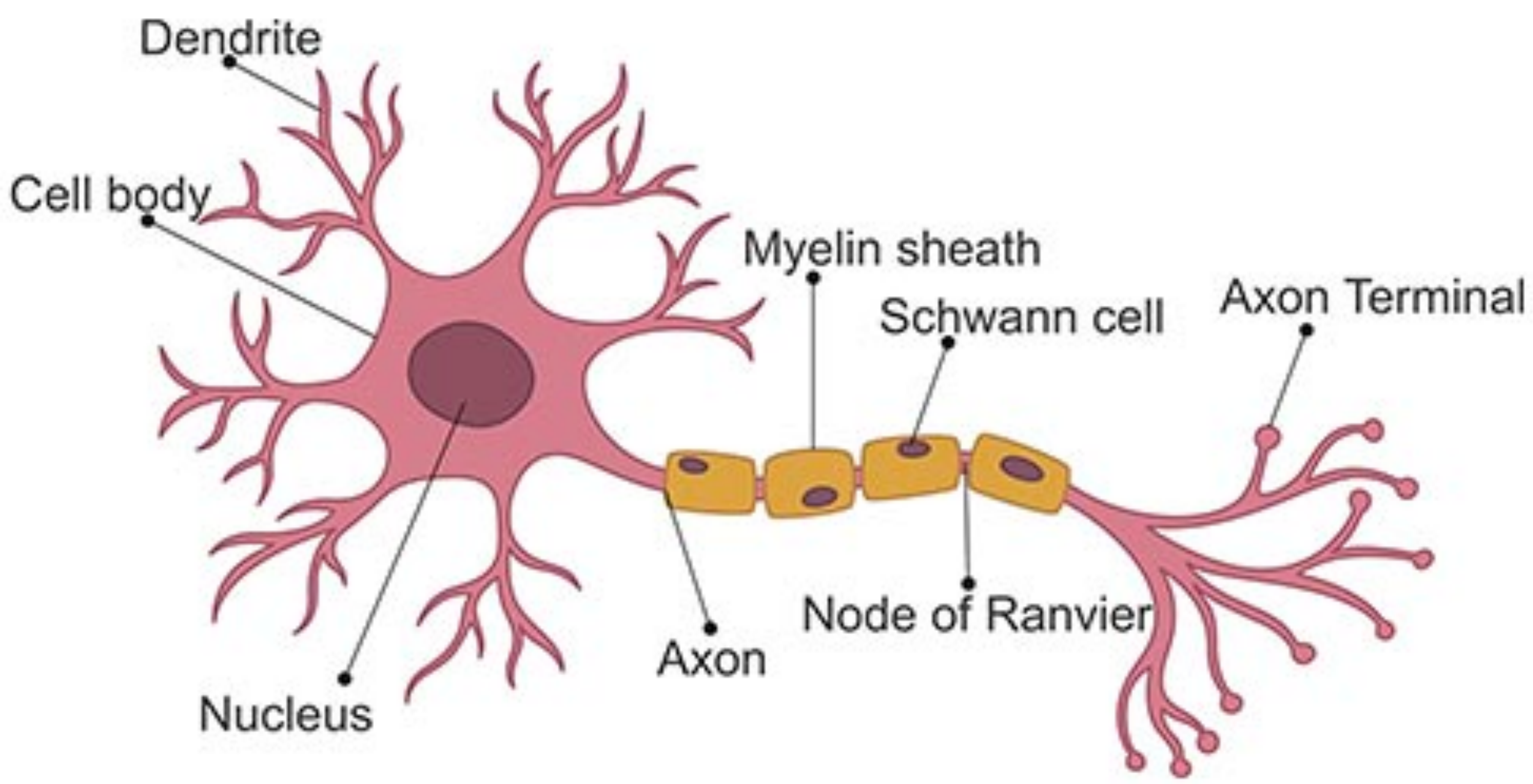
Summary so far

- **Behavioralism** tries to understand intelligence and learning by bracketing out unobservable mental phenomena. How far can we get with this approach?
- **Thorndike's Law of Effect** describes trial and error learning
  - no guidance for what actions we try, but repeat successful actions
- **Pavlovian Classical Conditioning** describes the association between stimuli and rewards based on predictions of reward
  - Rescorla Wagner (RW) model formalizes this theory based on reward prediction error (RPE) updating, which can be related to rational principles of maximum likelihood estimation and gradient descent
- **Operant conditioning** relates stimuli-reward associations to the active shaping of behavior, to acquire rewards and avoid punishment

5 minute break

Neural networks

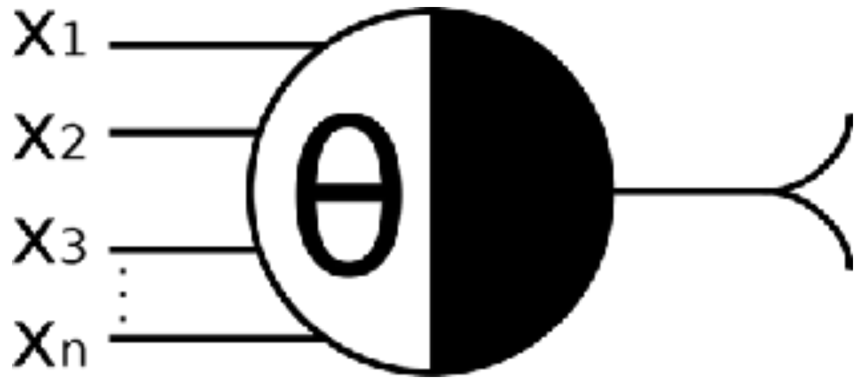
- Neurons are specialized cells that transmit information through electrical impulses
  - Roughly speaking, the dendrites receive information, which is processed in the cell body, and then propogated through the axon and synapses with other neurons
- Human perception, reasoning, emotions, actions, memory, and much more are governed by neural activity
- Whereas behaviorists focused on outward behavior, neuroscientists have been peering into black box for centuries in order to understand how neural activity gives rise to intelligence
- More recently (mid 1900s), artificial neural networks have been developed as computational tool for solving problems



Rosenblatt's Perceptron Mark I

Timeline of Artificial Neural Networks

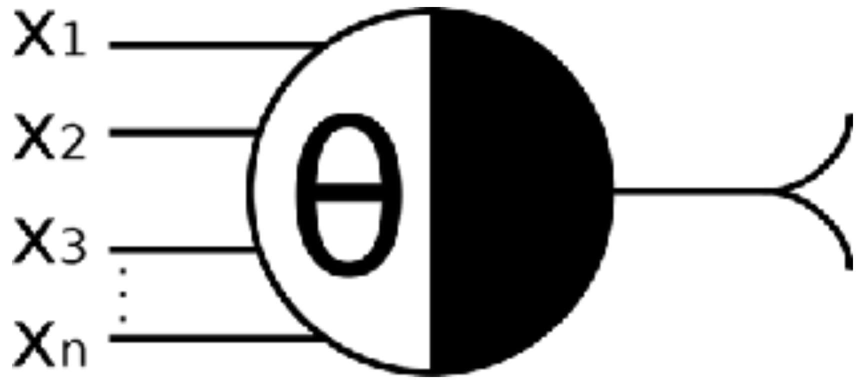
Timeline of Artificial Neural Networks



McCulloch & Pitts
(1943) neuron

Timeline of Artificial Neural Networks

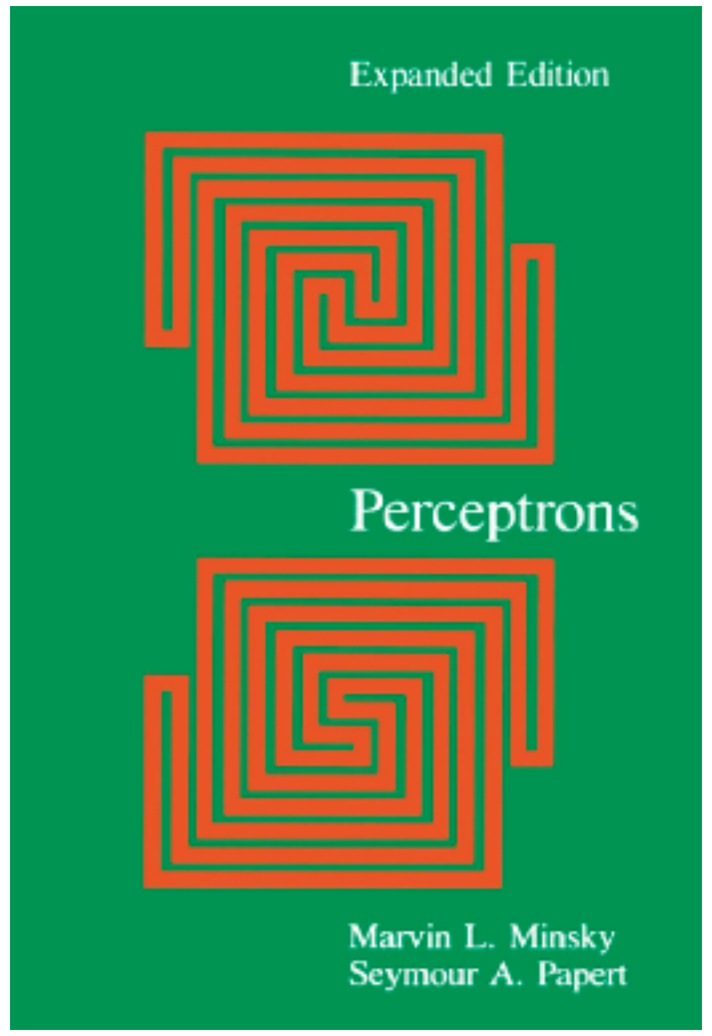
Rosenblatt (1958) Perceptron



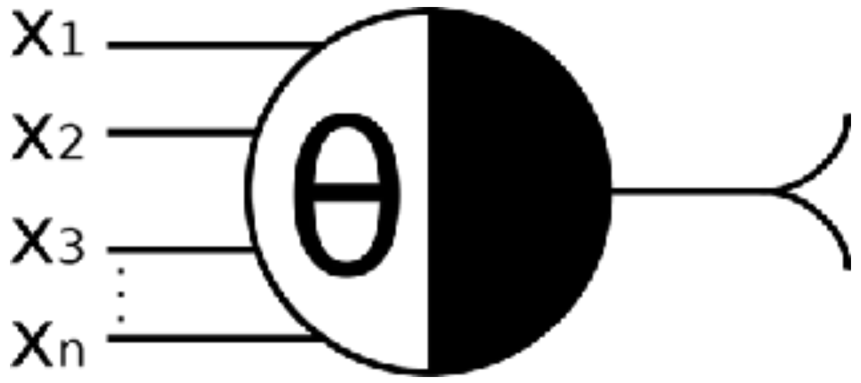
McCulloch & Pitts
(1943) neuron

Timeline of Artificial Neural Networks

Minsky & Parpert (1969)



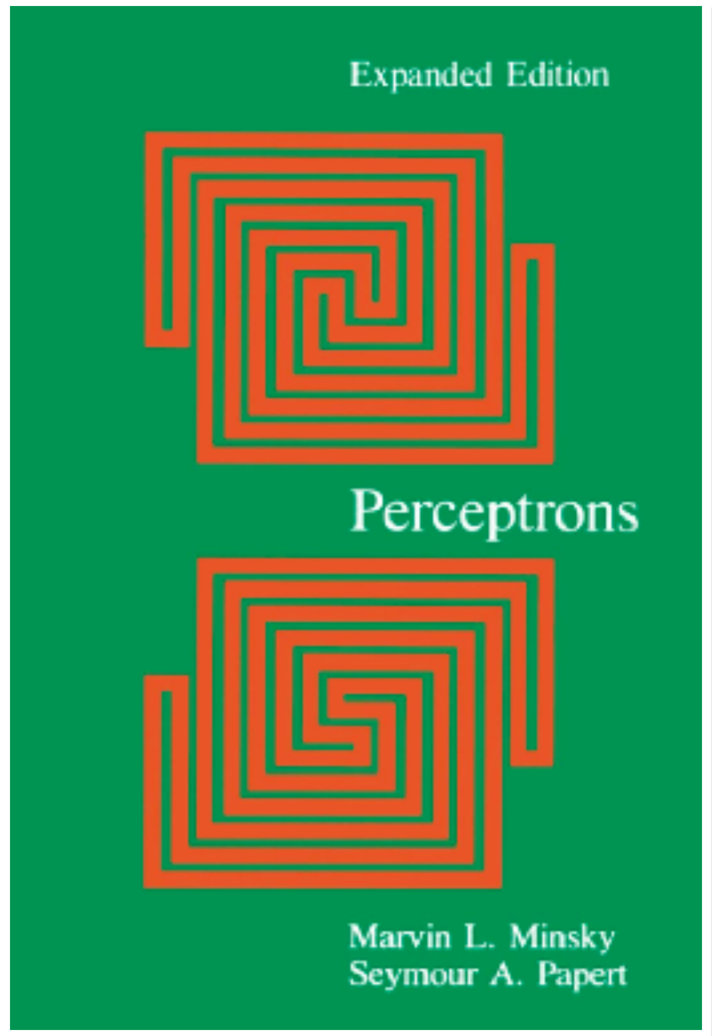
Rosenblatt (1958) Perceptron



McCulloch & Pitts
(1943) neuron

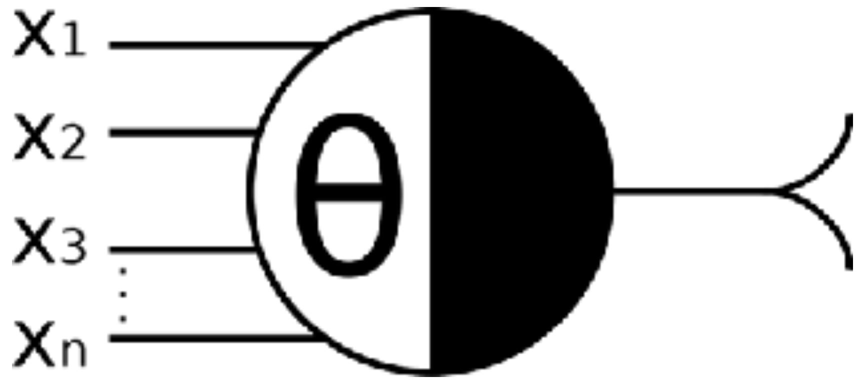
Timeline of Artificial Neural Networks

Minsky & Parpert (1969)



Rosenblatt (1958) Perceptron

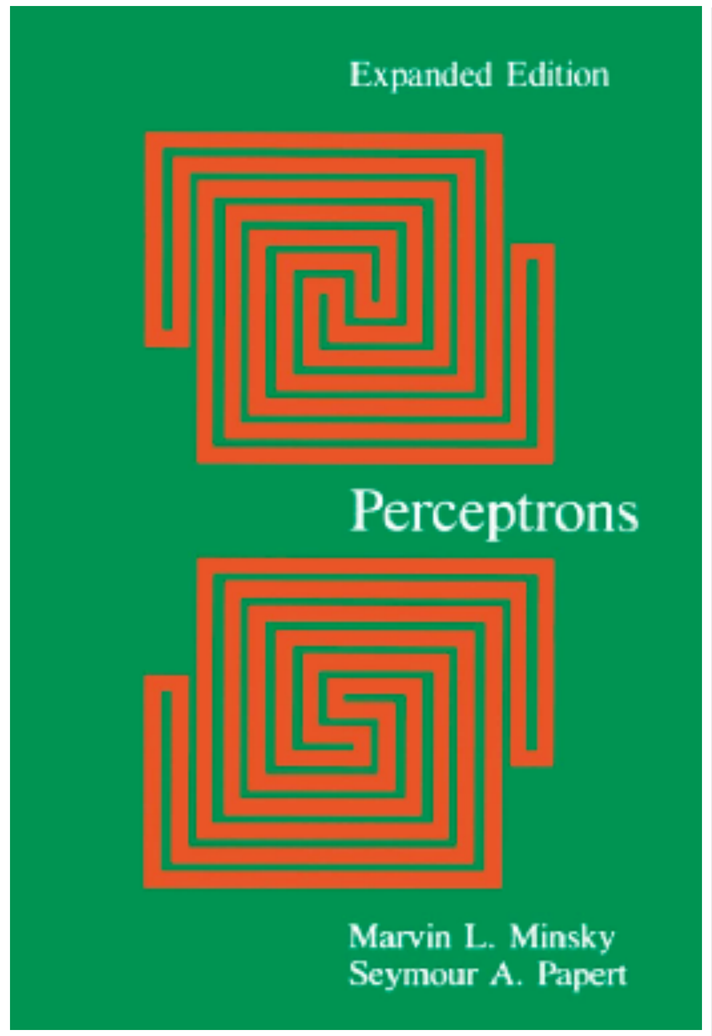
AI Winter



McCulloch & Pitts
(1943) neuron

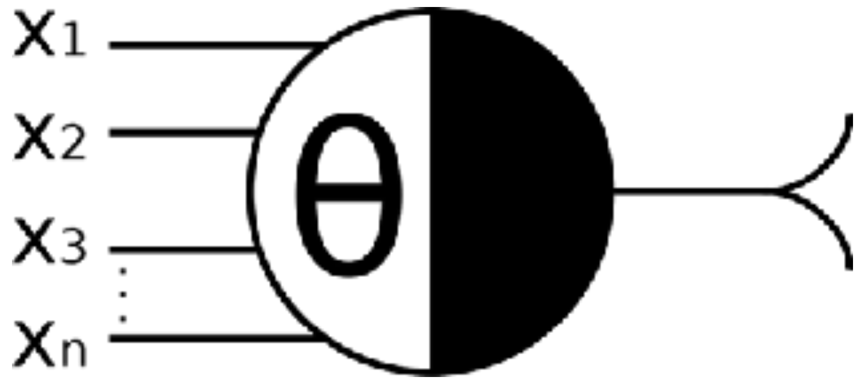
Timeline of Artificial Neural Networks

Minsky & Parpert (1969)

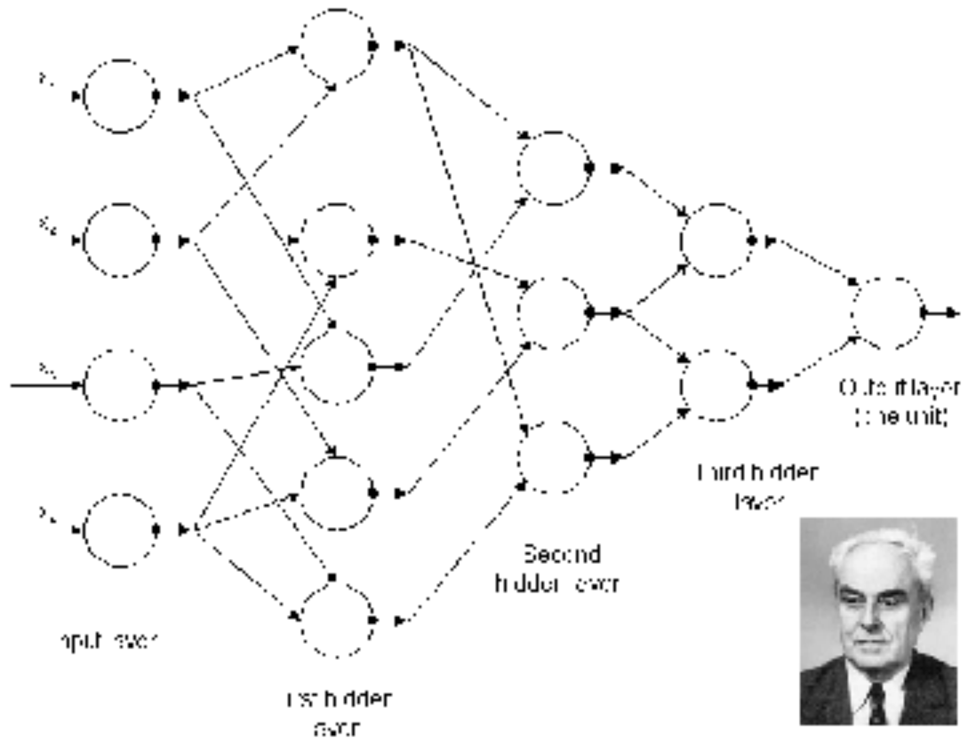


Rosenblatt (1958) Perceptron

AI Winter



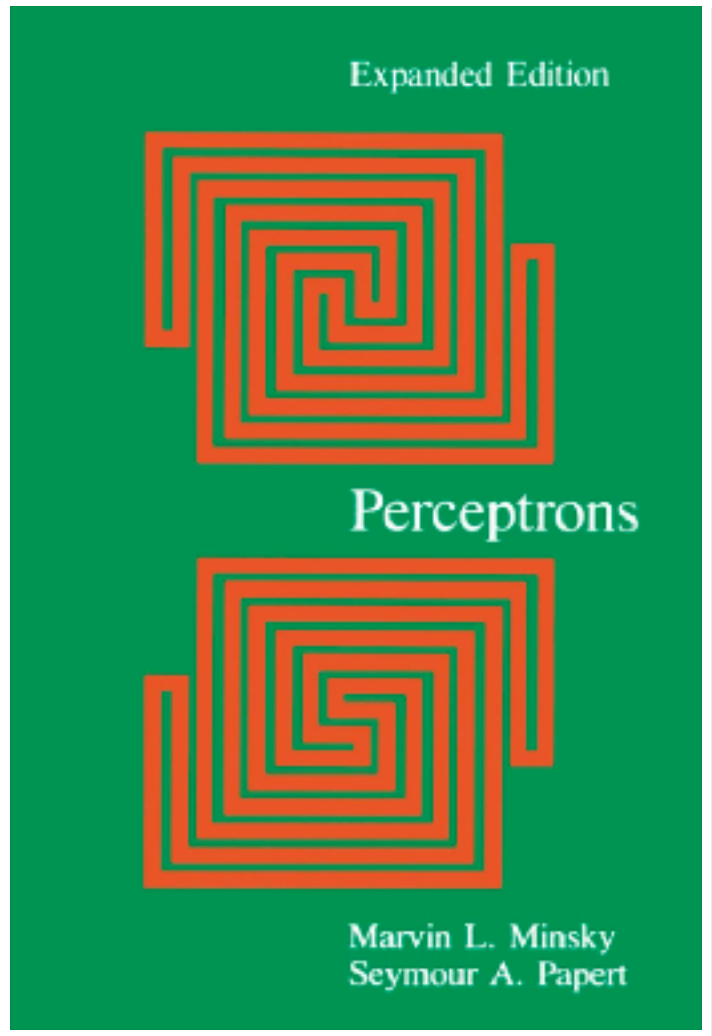
McCulloch & Pitts
(1943) neuron



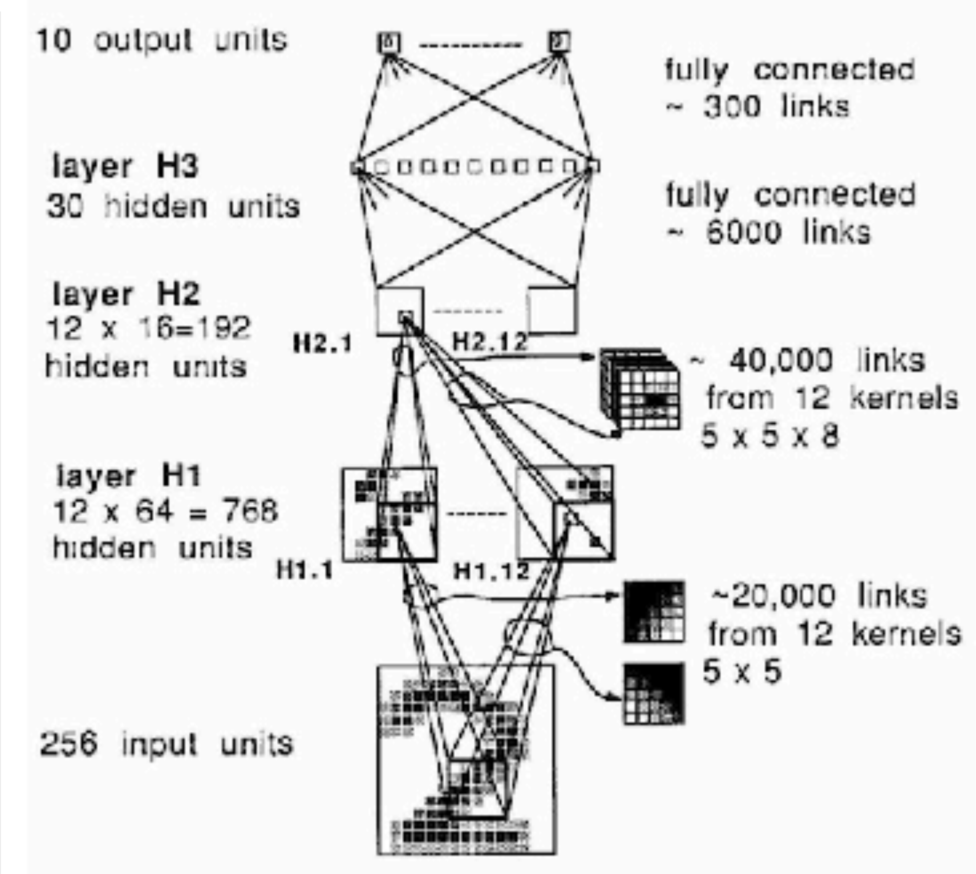
First deep network (Ivakhnenko & Lapa 1965)

Timeline of Artificial Neural Networks

Minsky & Parpert (1969)

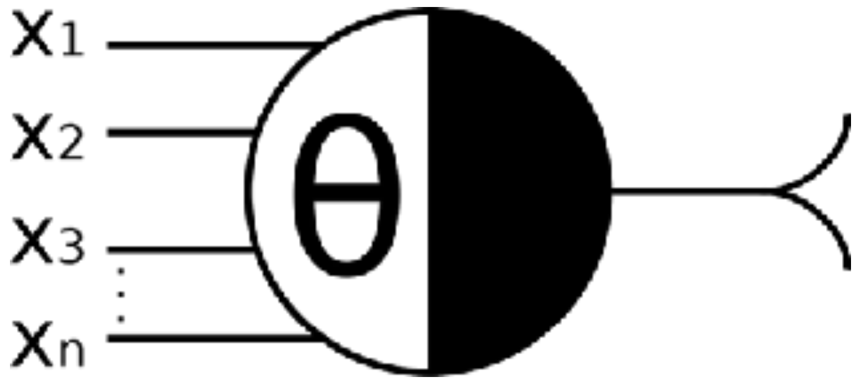


Convnets for MNIST (LeCun et al., 1989)

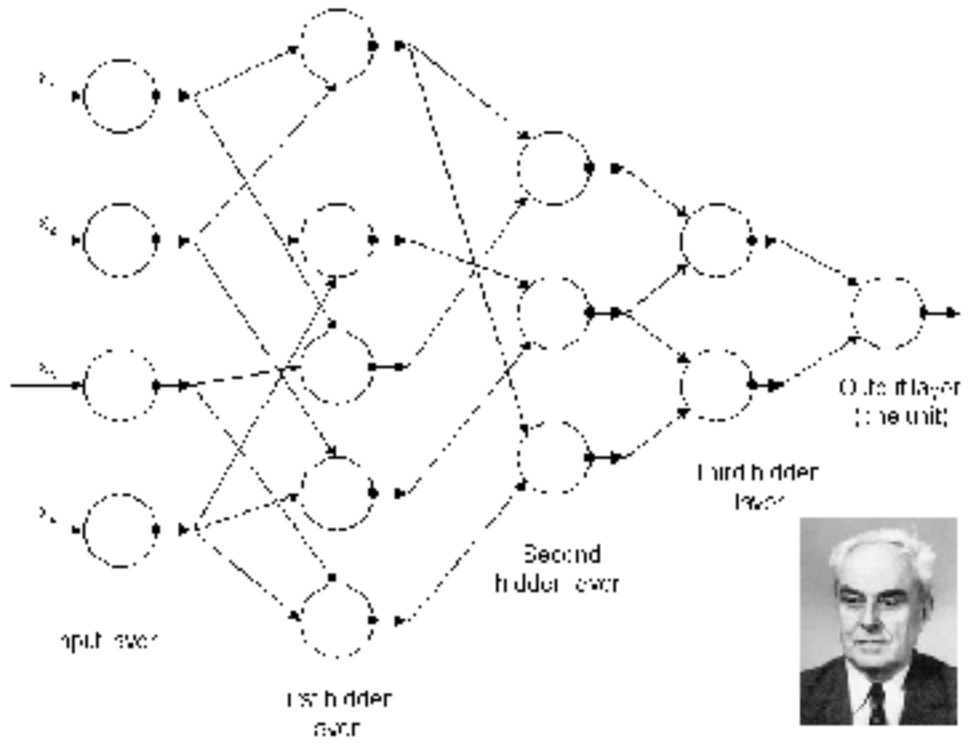


Rosenblatt (1958) Perceptron

AI Winter



McCulloch & Pitts
(1943) neuron



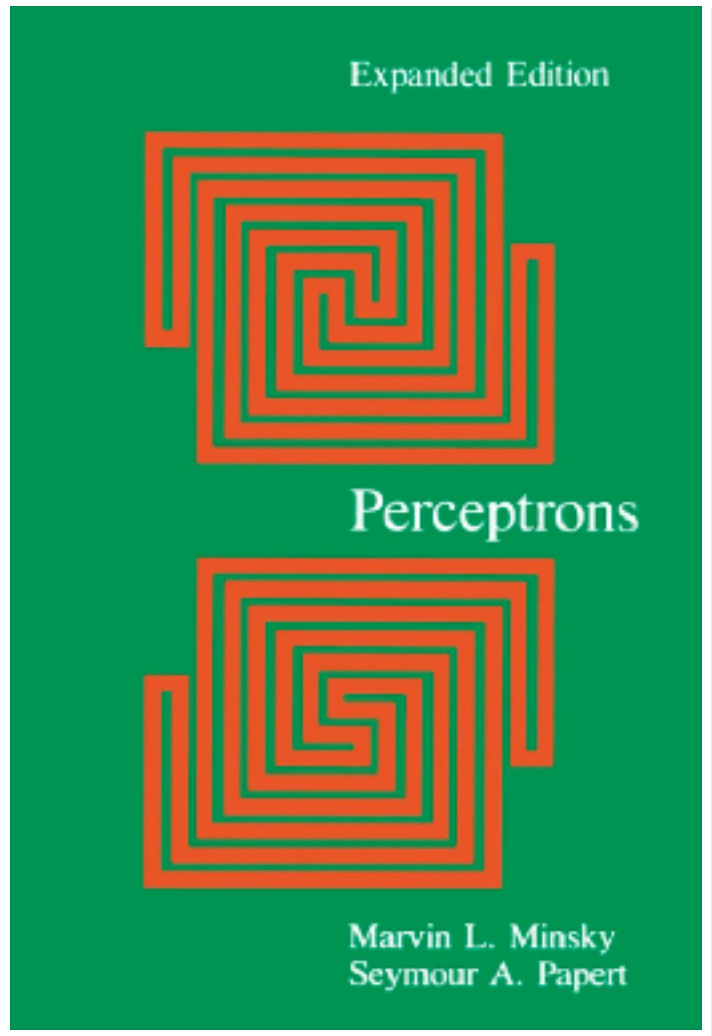
First deep network (Ivakhnenko & Lapa 1965)

Timeline of Artificial Neural Networks

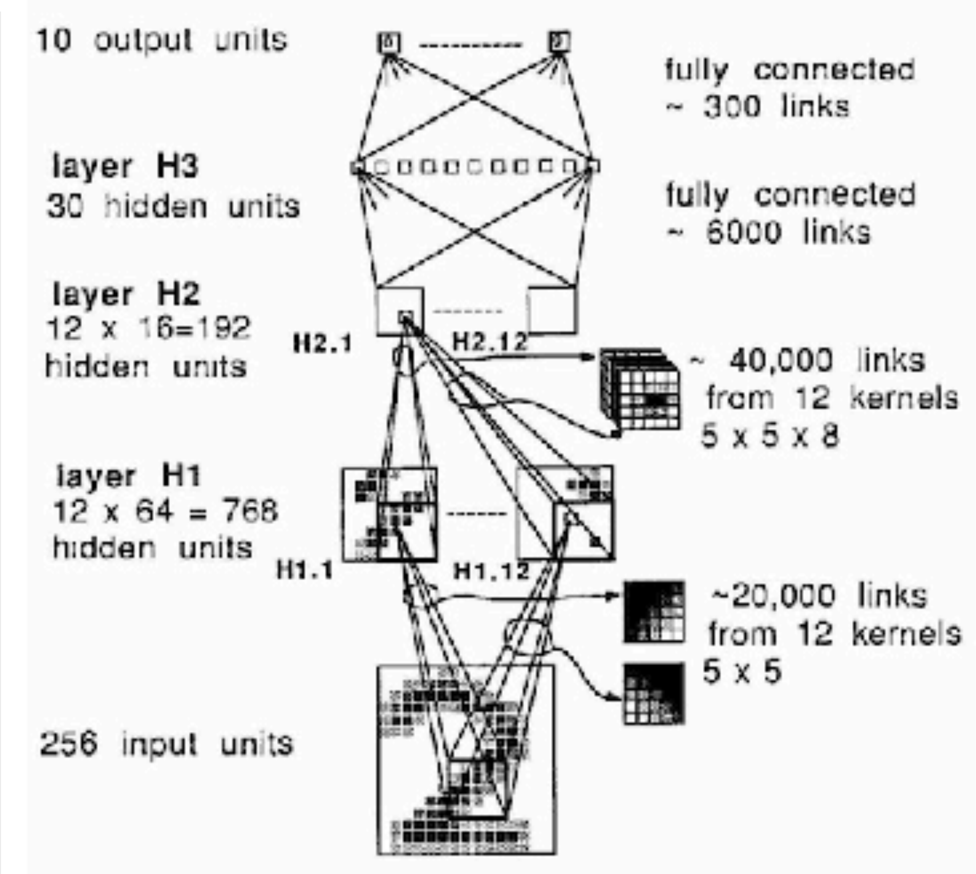
Deep Learning revolution

Rosenblatt (1958) Perceptron

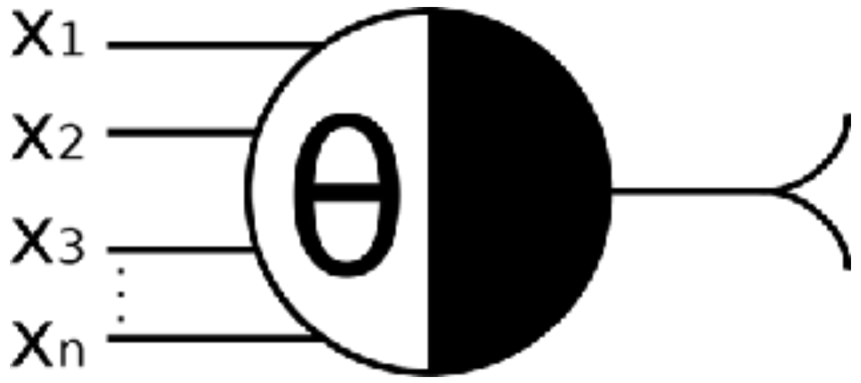
Minsky & Parpert (1969)



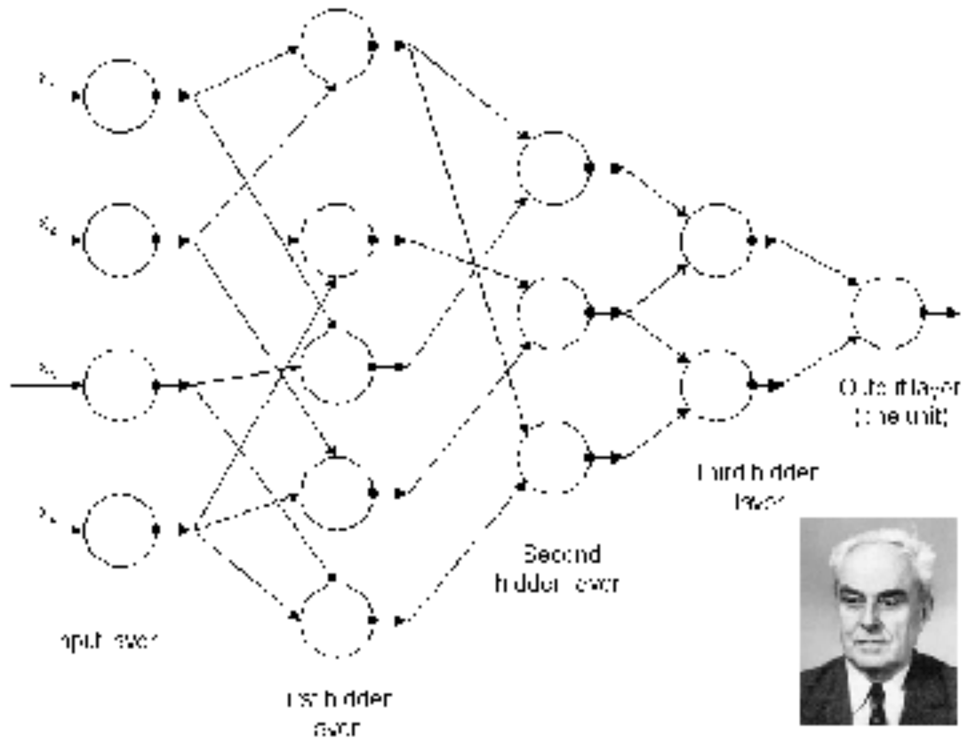
Convnets for MNIST (LeCun et al., 1989)



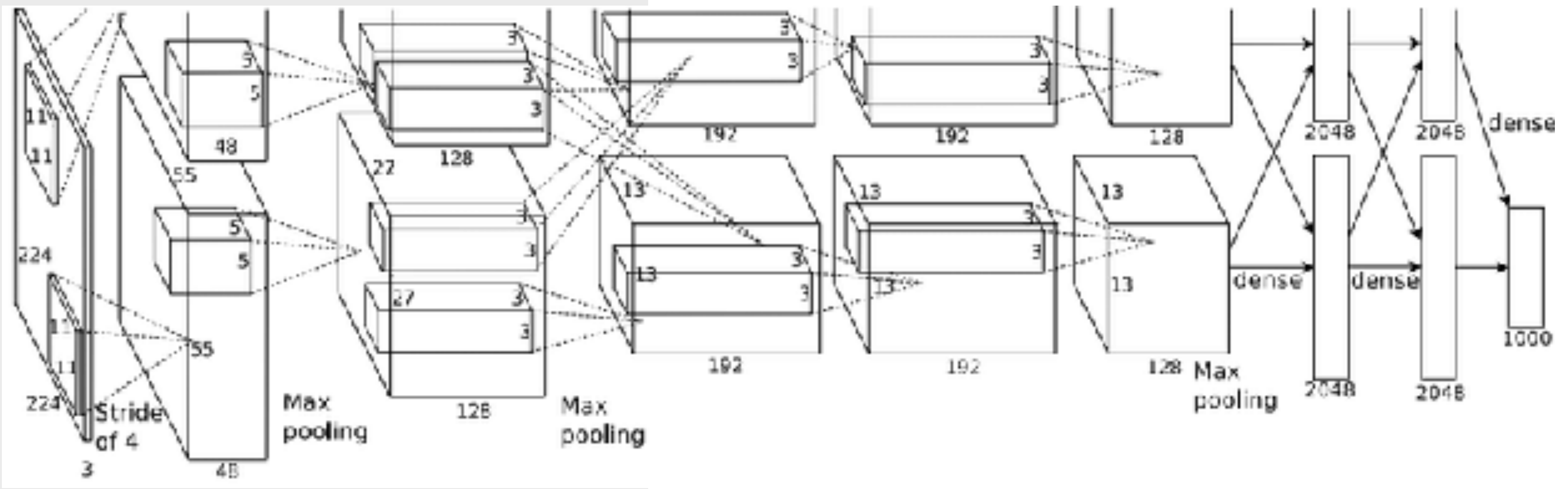
AI Winter



McCulloch & Pitts (1943) neuron

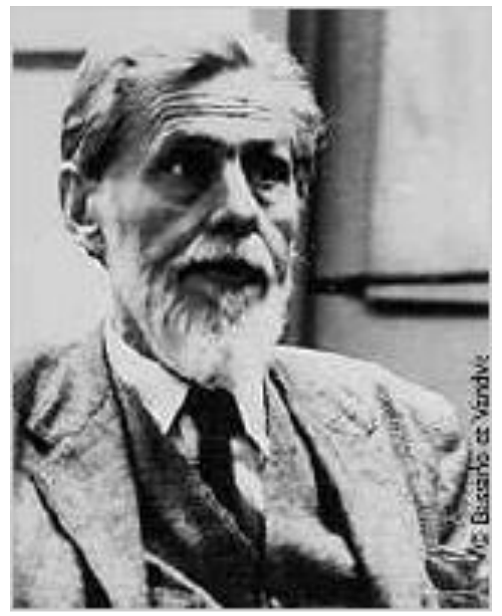


First deep network (Ivakhnenko & Lapa 1965)



ReLU & Dropout (Krizhevsky, Sutskever, & Hinton, 2012)

McCulloch & Pitts (1943)



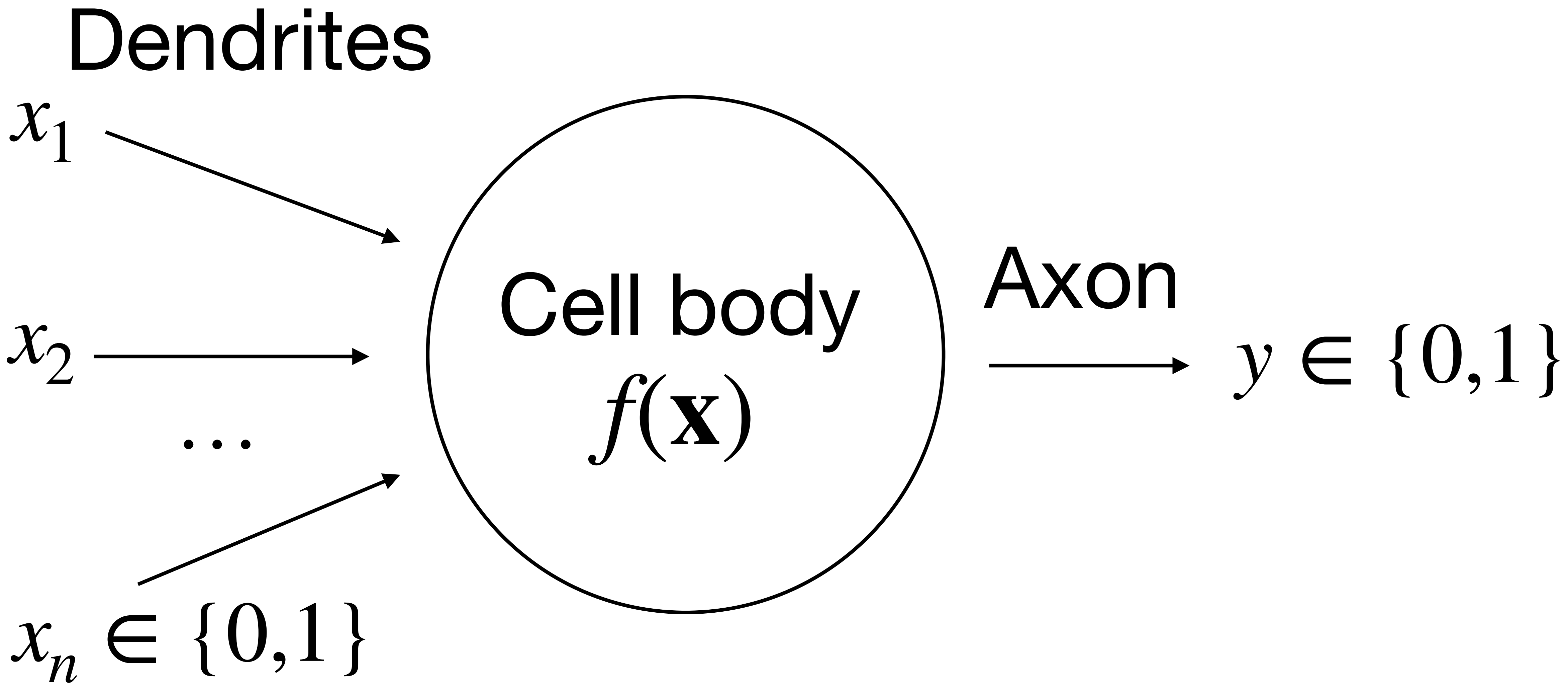
Warren McCulloch

Walter Pitts

- First computational model of a neuron
- The dendritic inputs $\{x_1, \ldots, x_n\}$ provide the input signal
- The cell body processes the signal

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if} \sum x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

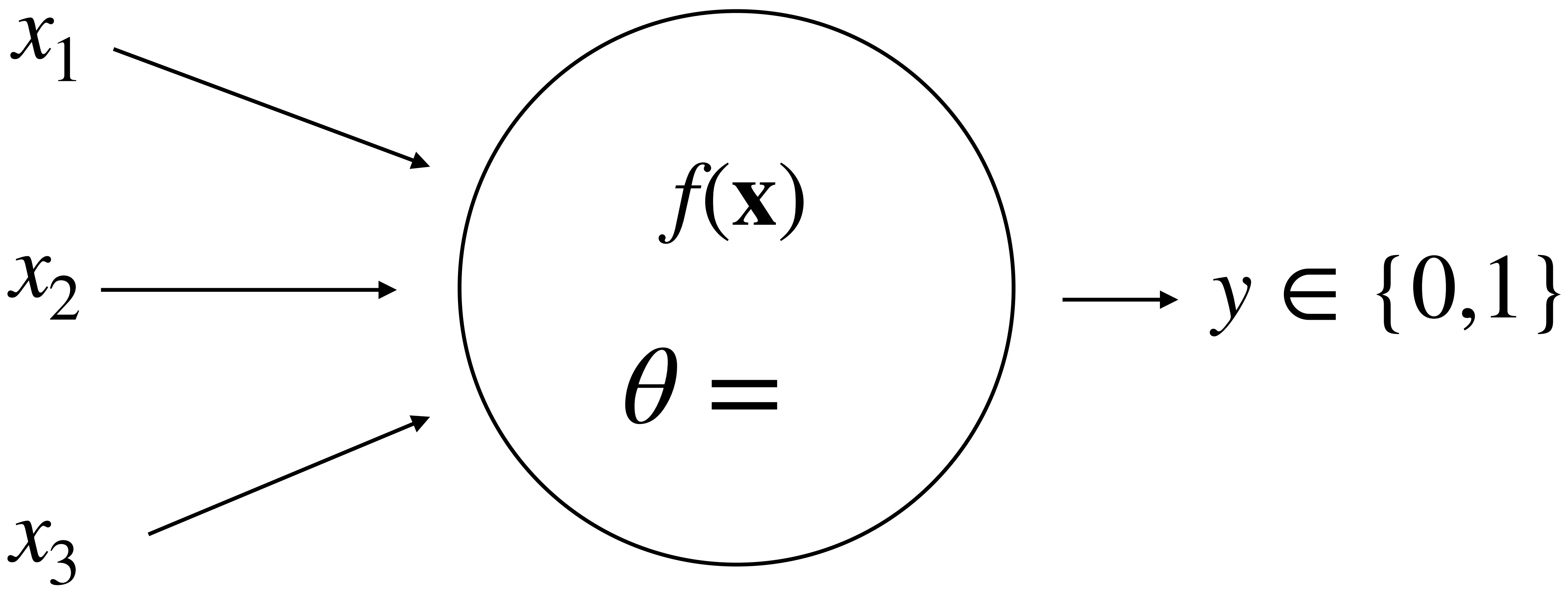
- The axon produces the output



McCulloch & Pitts (1943)

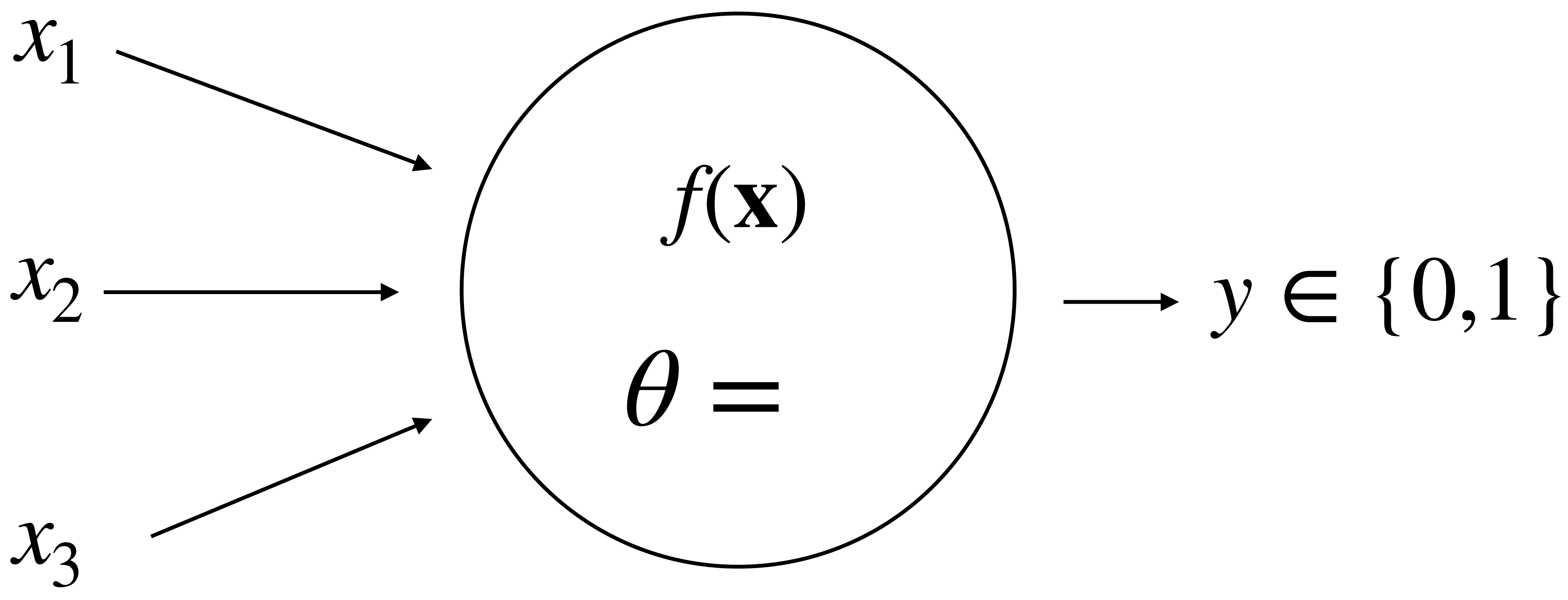
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

AND function



All inputs need to be on for the neuron to fire

OR function

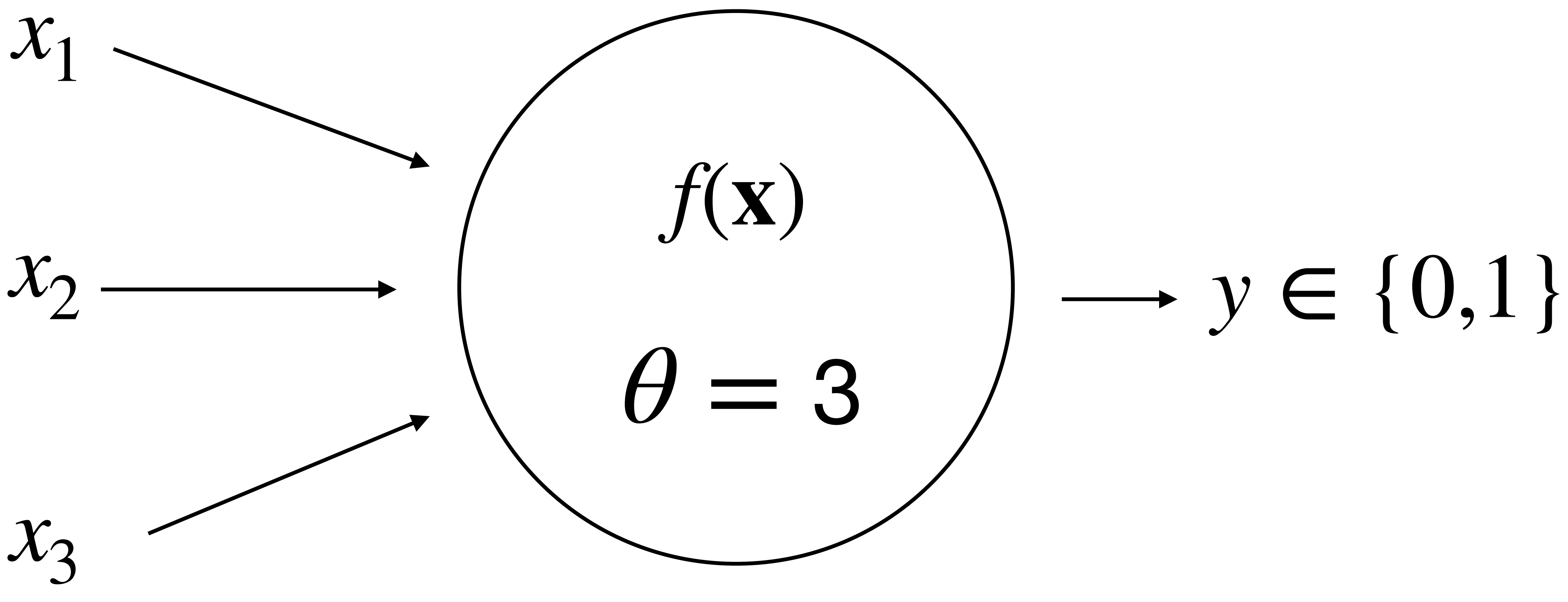


Neuron fires if any input is on

McCulloch & Pitts (1943)

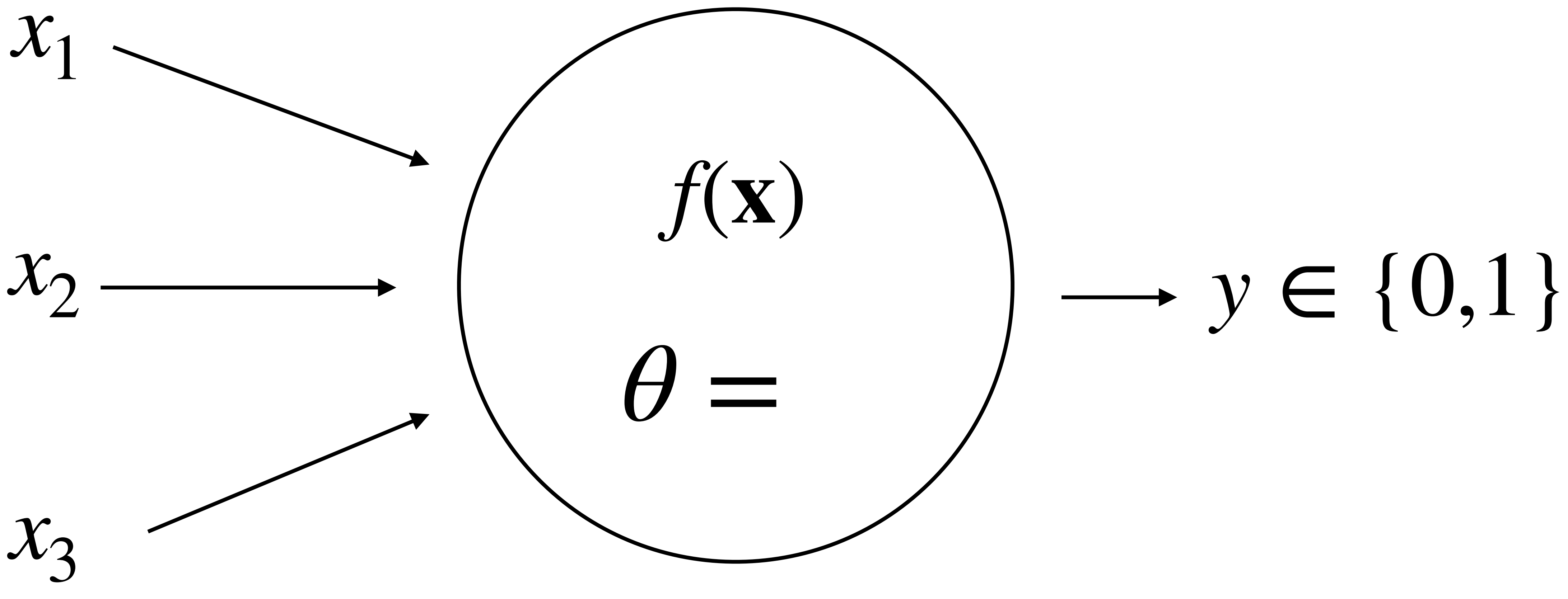
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

AND function



All inputs need to be on for the neuron to fire

OR function

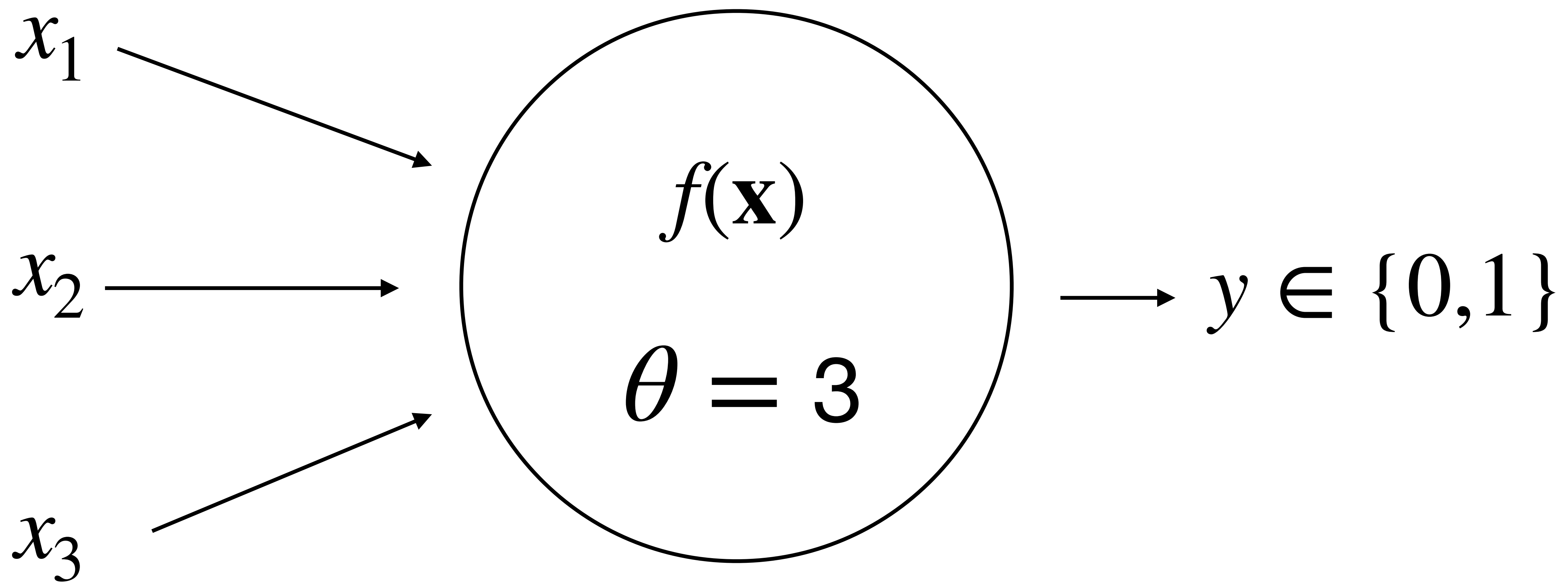


Neuron fires if any input is on

McCulloch & Pitts (1943)

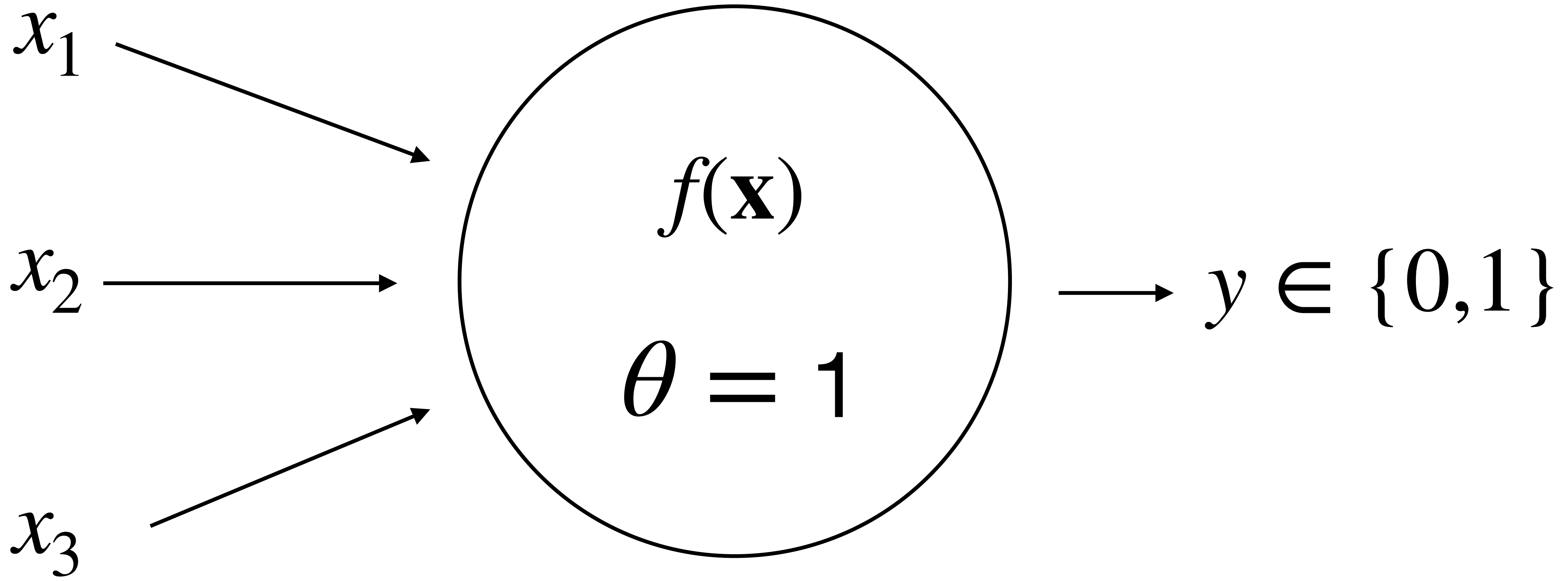
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

AND function



All inputs need to be on for the neuron to fire

OR function

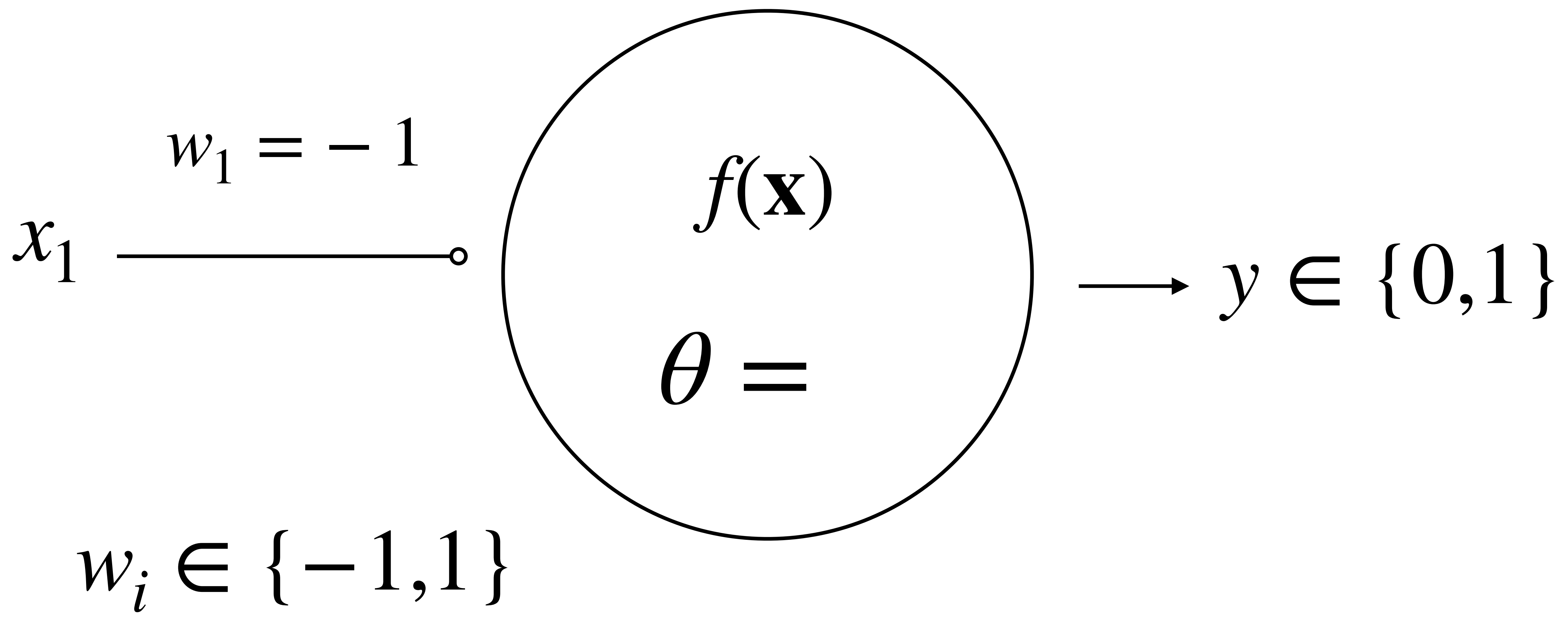


Neuron fires if any input is on

McCulloch & Pitts (1943)

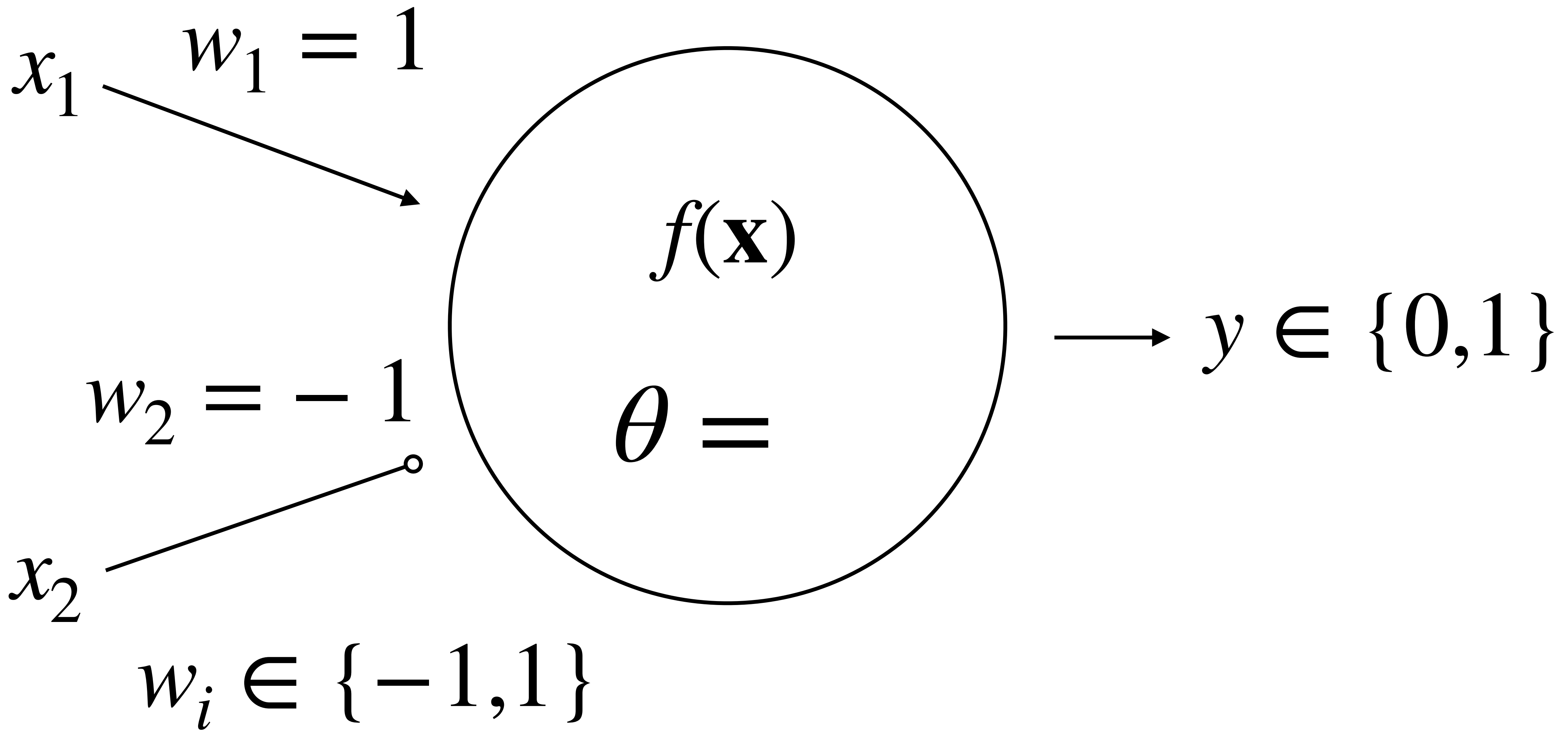
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if} \sum w_i x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

NOT function



Neuron fires if no inputs are on

NAND

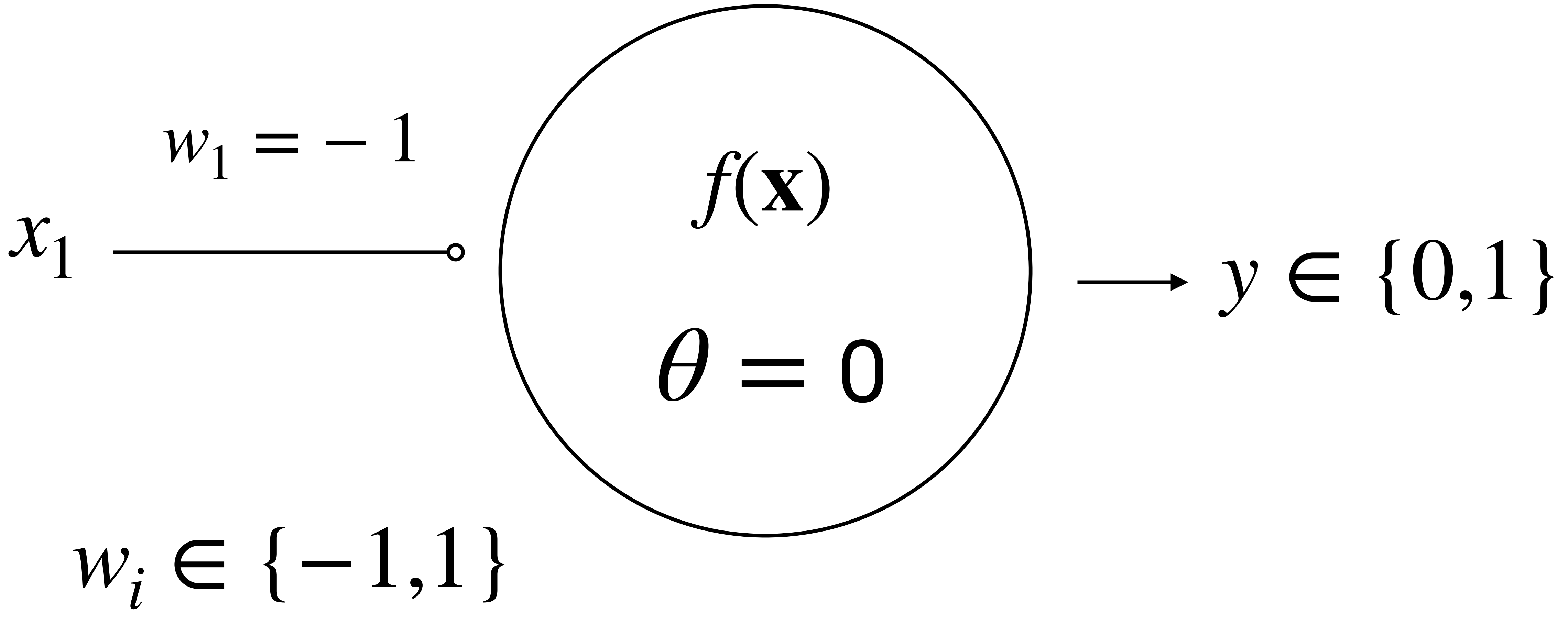


Neuron fires when x_1 is on AND x_2 not on

McCulloch & Pitts (1943)

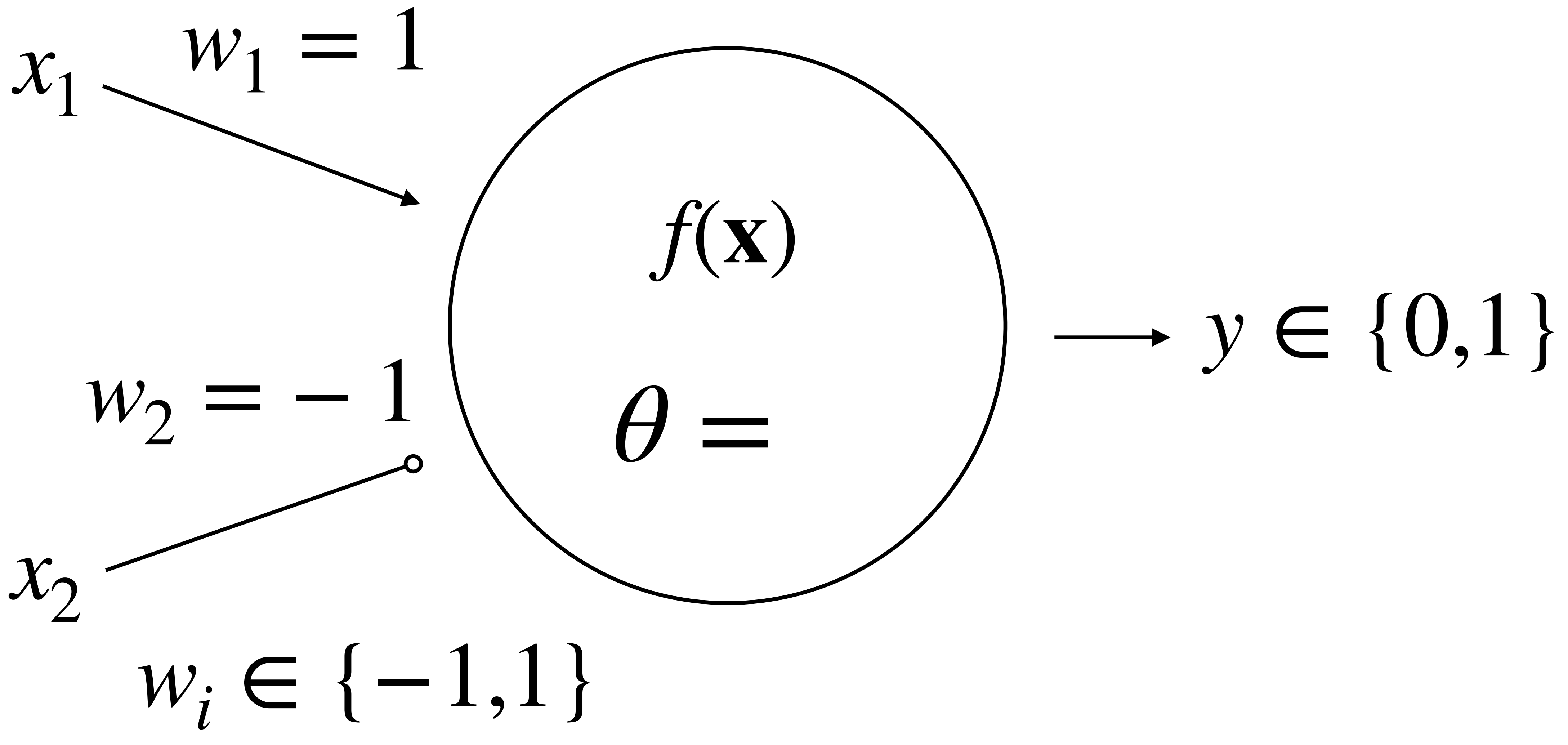
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if} \sum w_i x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

NOT function



Neuron fires if no inputs are on

NAND

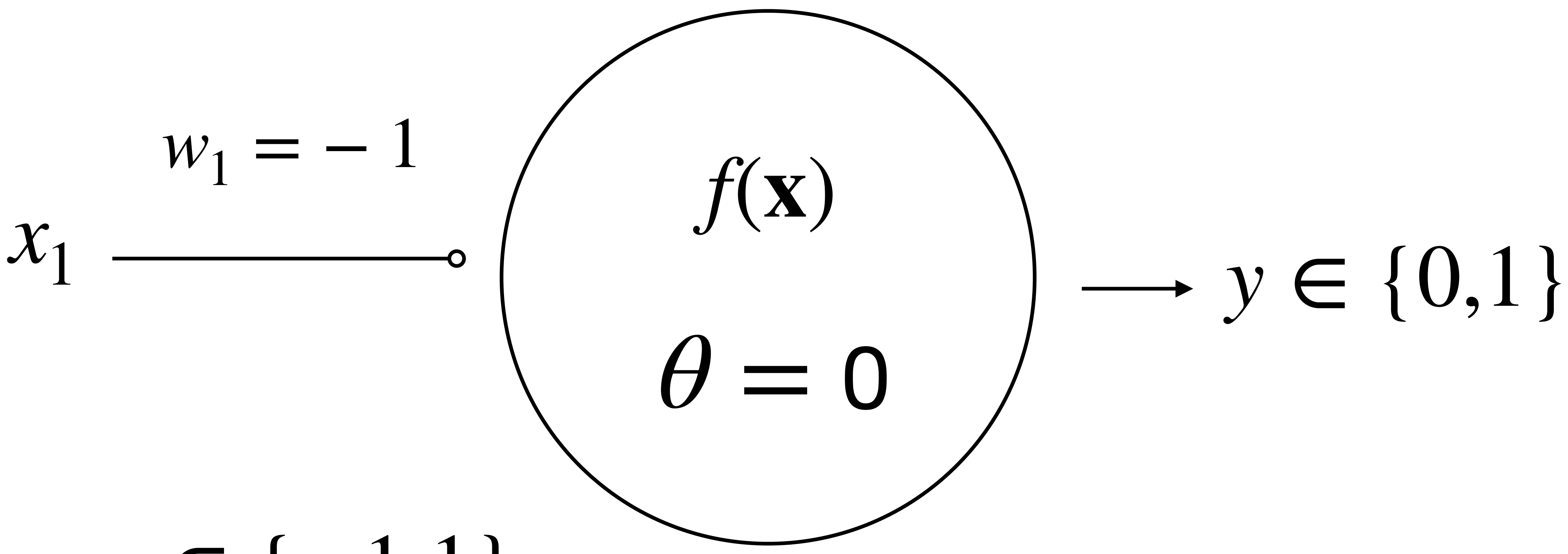


Neuron fires when x_1 is on AND x_2 not on

McCulloch & Pitts (1943)

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum w_i x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

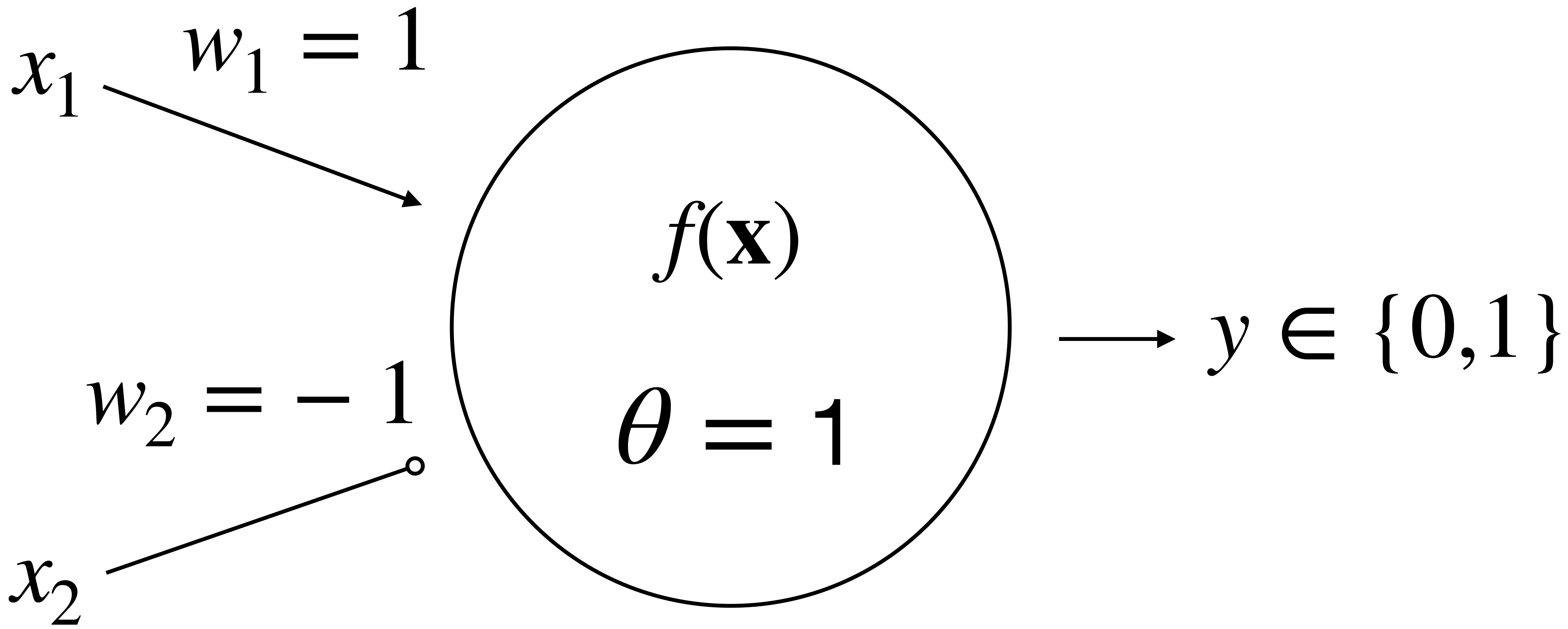
NOT function



$w_i \in \{-1,1\}$

Neuron fires if no inputs are on

NAND



$w_i \in \{-1,1\}$

Neuron fires when x_1 is on AND x_2 not on

Rosenblatt's Perceptron

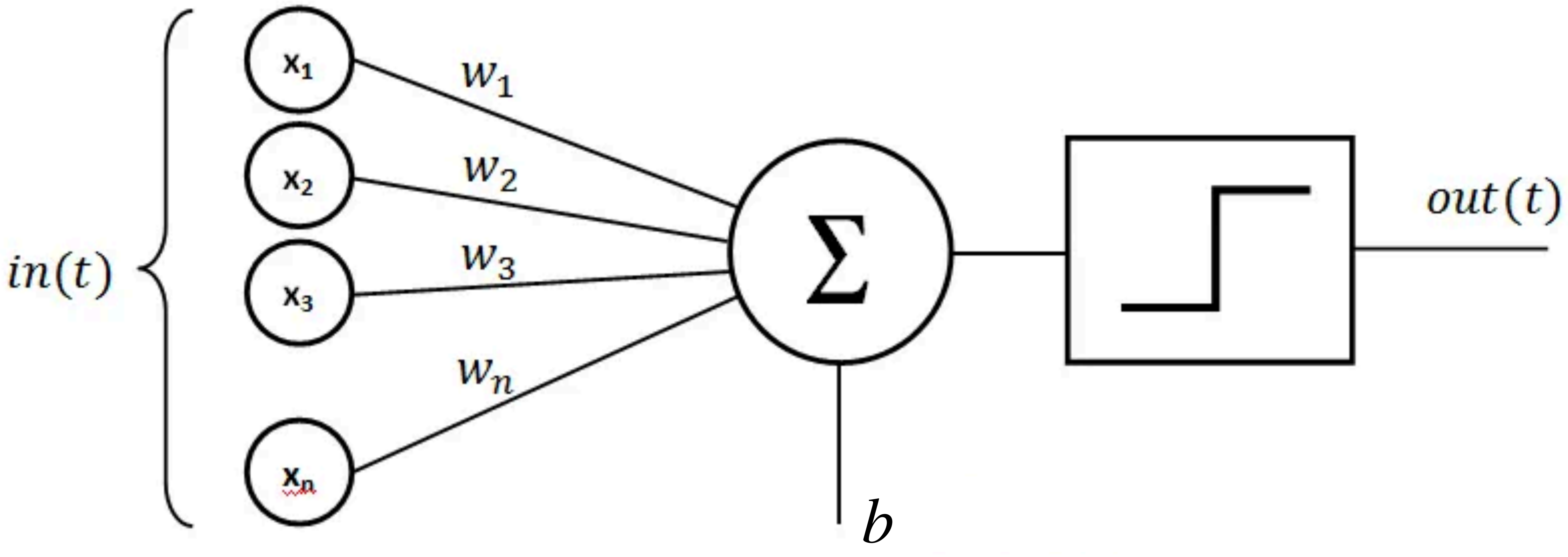
- Added a learning rule, allowing it to learn any binary classification problem *with linear seperability*

- Very similar to McCulloch & Pitts', but with some key differences:

  - A bias term is added b

  - Weights w_i aren't only $\in \{-1,1\}$ but can be any real number

- Weights (and bias) are updated based on error



Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$.

Initialize **w** and b randomly.

while *not converged* **do**

  # # # Loop through the examples.
  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.
    $error = y_j - \sigma(\mathbf{w}^T\mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.
    if *error != 0* **then**

      ### Update the weights.
      $\mathbf{w} = \mathbf{w} + error \times x_j$

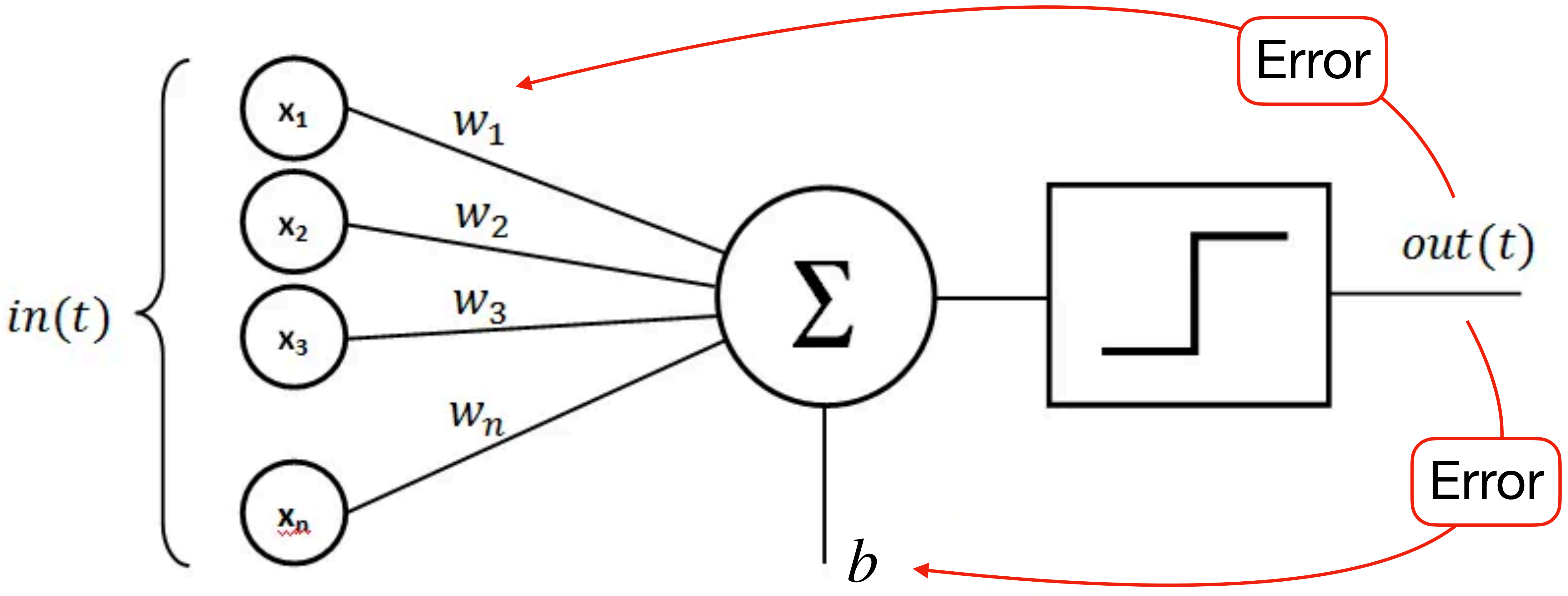
      ### Update the bias.
      $b = b + error$

  Test for convergence

Output: Set of weights **w** and bias b for the perceptron.

Rosenblatt's Perceptron

- Added a learning rule, allowing it to learn any binary classification problem *with linear seperability*
- Very similar to McCulloch & Pitts', but with some key differences:
  - A bias term is added b
  - Weights w_i aren't only $\in \{-1,1\}$ but can be any real number
- Weights (and bias) are updated based on error



Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$.

Initialize $\mathbf{w}$ and b randomly.

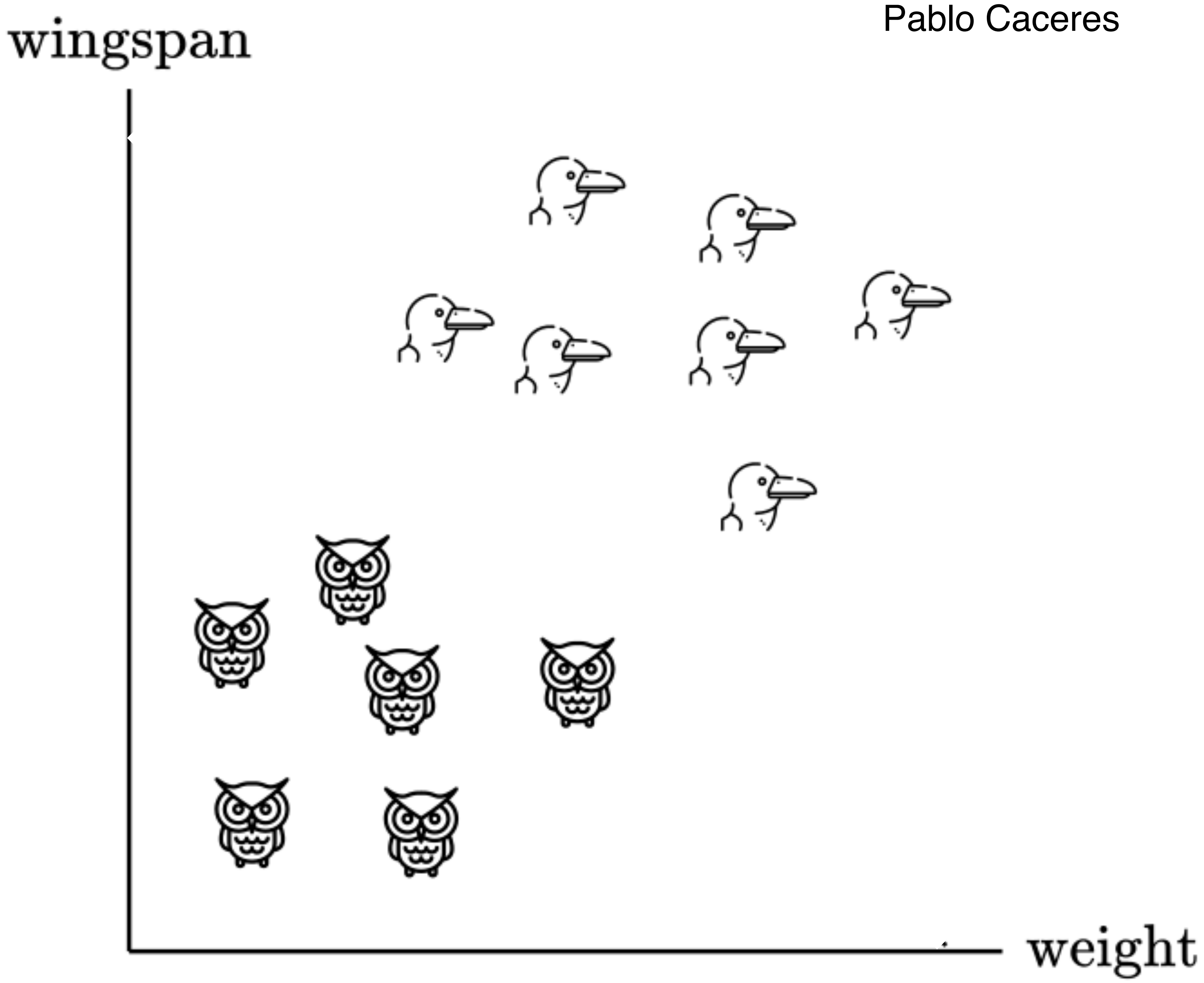
while *not converged* **do**

  # # # Loop through the examples.
  for $j = 1, m$ **do**
    # # # Compare the true label and the prediction.
    $error = y_j - \sigma(\mathbf{w}^T\mathbf{x}_j + b)$
    ### If the model wrongly predicts the class, we update the weights and bias.
    if *error != 0* **then**
      ### Update the weights.
      $\mathbf{w} = \mathbf{w} + error \times x_j$
      ### Update the bias.
      $b = b + error$
  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

Pablo Caceres

Perceptron learning rule



Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$.

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.
  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.
    $error = y_j - \sigma(\mathbf{w}^T \mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.
    if *error != 0* **then**

      ### Update the weights.
      $\mathbf{w} = \mathbf{w} + error \times x_j$

      ### Update the bias.
      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

Pablo Caceres

Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan)

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.
  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.
    $error = y_j - \sigma(\mathbf{w}^T\mathbf{x}_j + b)$

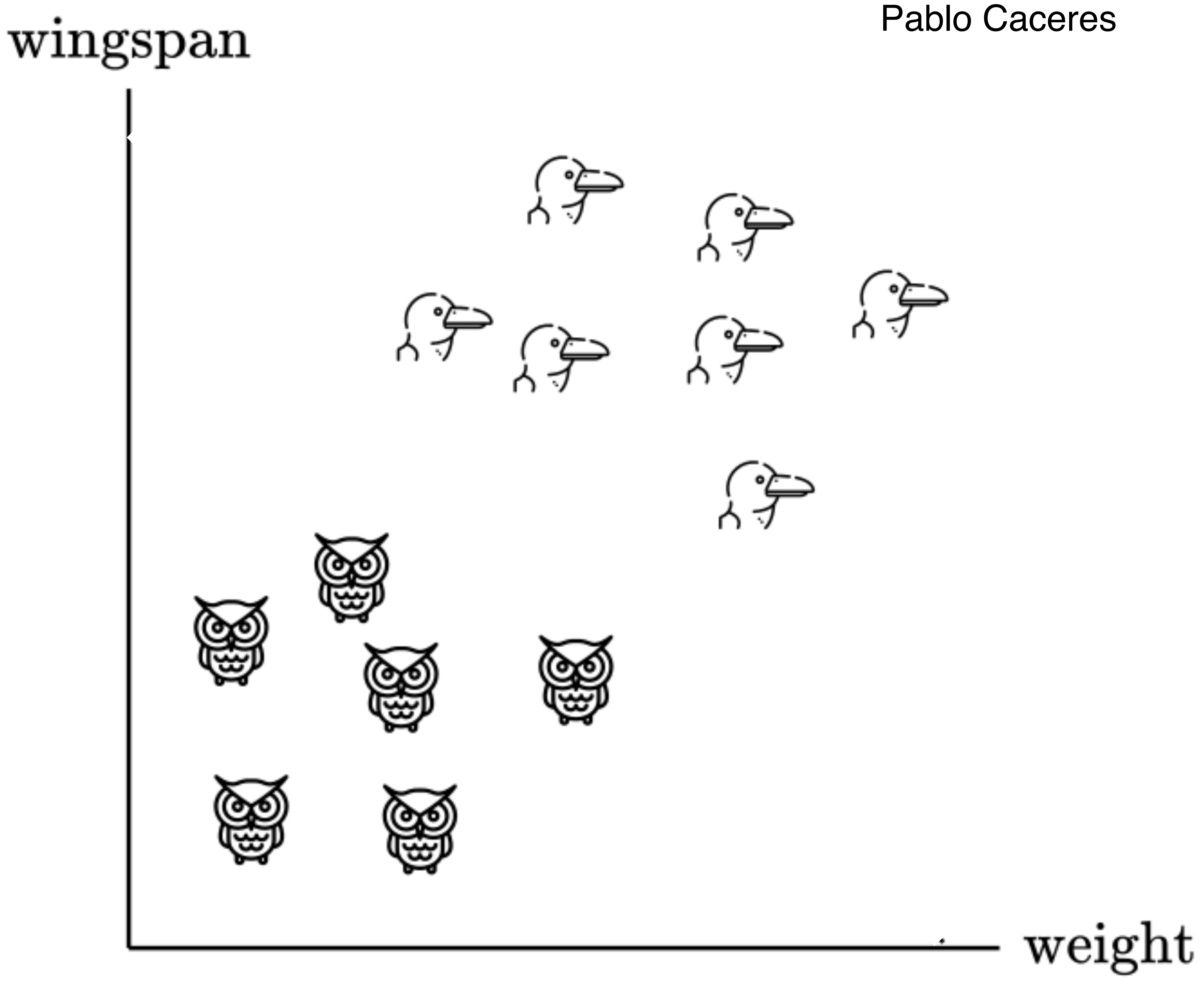
    ### If the model wrongly predicts the class, we update the weights and bias.
    if *error != 0* **then**

      ### Update the weights.
      $\mathbf{w} = \mathbf{w} + error \times x_j$

      ### Update the bias.
      $b = b + error$

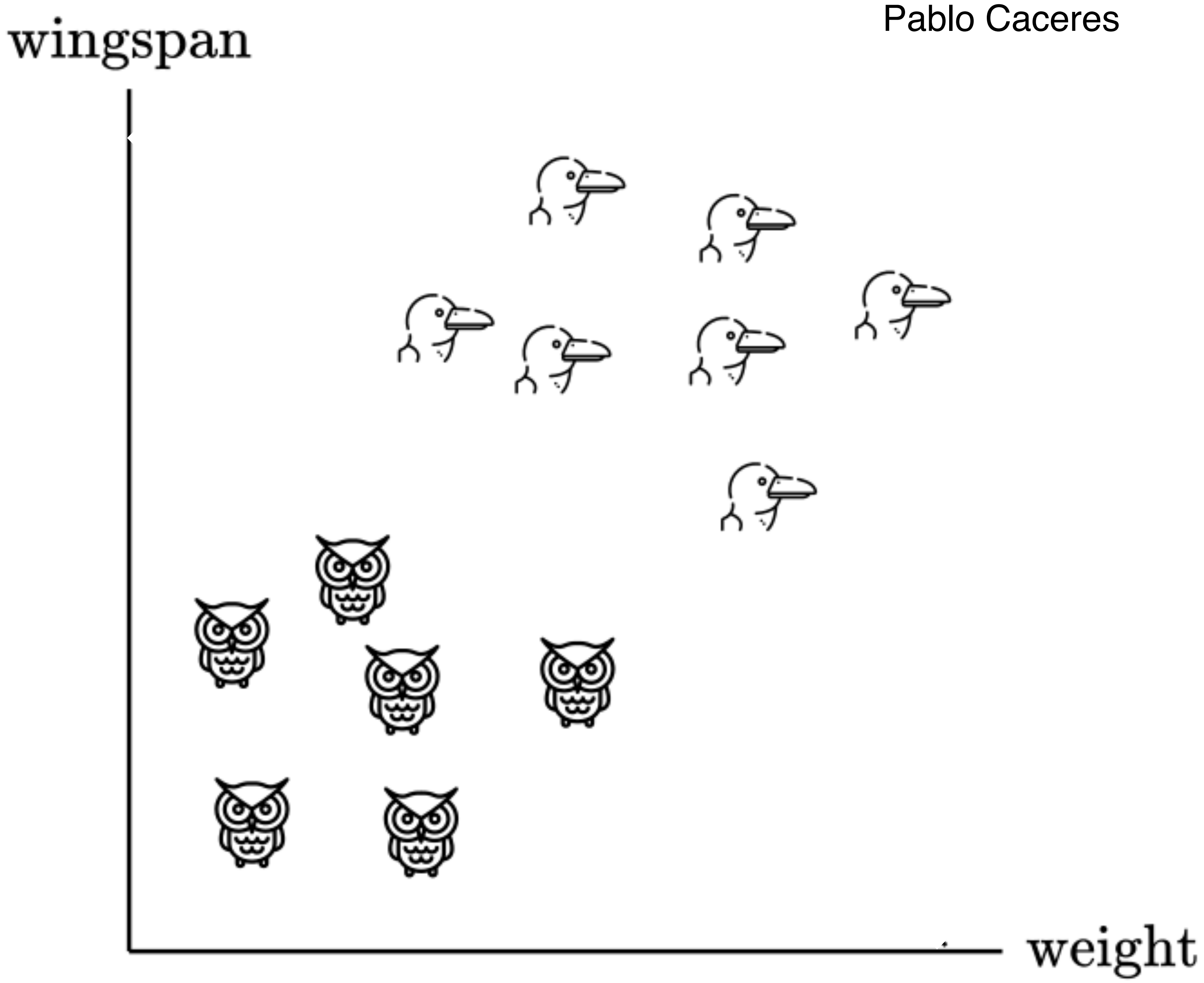
  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.



Pablo Caceres

Perceptron learning rule



Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.
  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.
    $error = y_j - \sigma(\mathbf{w}^T \mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.
    if *error != 0* **then**

      ### Update the weights.
      $\mathbf{w} = \mathbf{w} + error \times x_j$

      ### Update the bias.
      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.
  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.
    $error = y_j - \sigma(\mathbf{w}^T \mathbf{x}_j + b)$

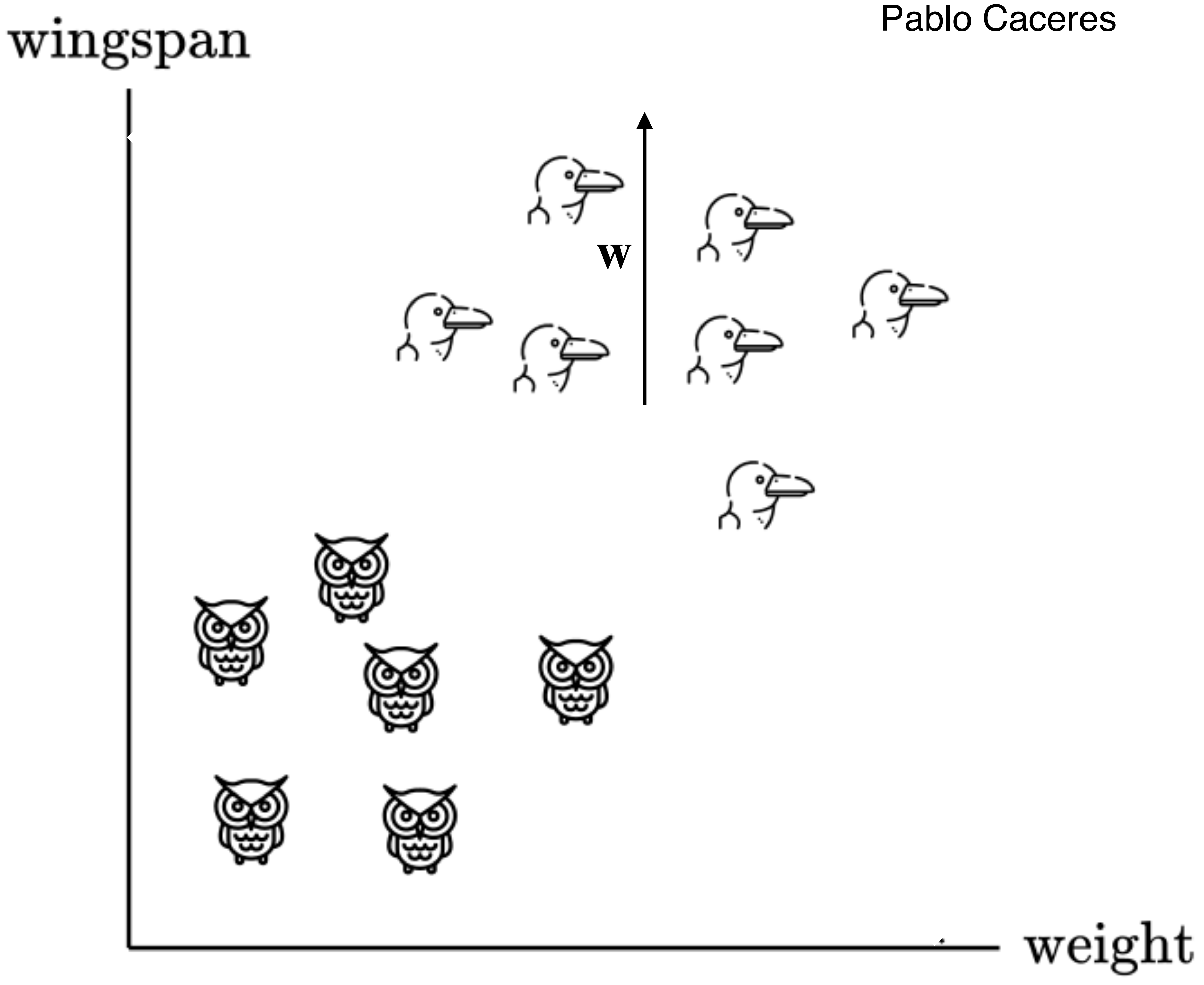
    ### If the model wrongly predicts the class, we update the weights and bias.
    if *error != 0* **then**

      ### Update the weights.
      $\mathbf{w} = \mathbf{w} + error \times x_j$

      ### Update the bias.
      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.



Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

    # # # Loop through the examples.

    for $j = 1, m$ **do**

        # # # Compare the true label and the prediction.

        $error = y_j - \sigma(\mathbf{w}^T \mathbf{x}_j + b)$

        ### If the model wrongly predicts the class, we update the weights and bias.

        if *error != 0* **then**

            ### Update the weights.

            $\mathbf{w} = \mathbf{w} + error \times x_j$

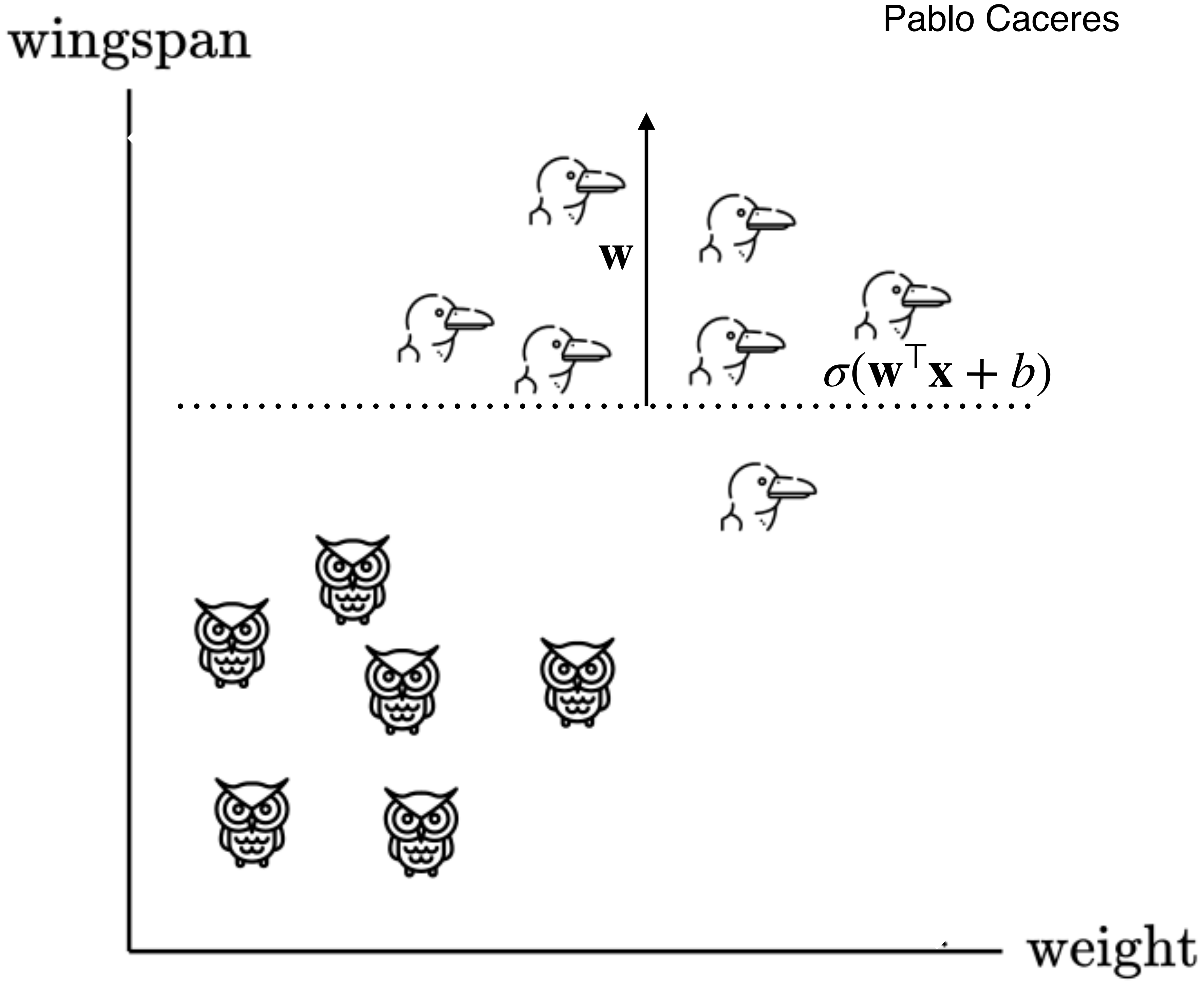
            ### Update the bias.

            $b = b + error$

    Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

$$\sigma(\mathbf{w}^\top \mathbf{x} + b) = \begin{cases} 1 & \text{if} \quad \mathbf{w}^\top \mathbf{x} + b > 0 \\ 0 & \text{else} \end{cases}$$



Pablo Caceres

Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.

  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.

    $error = y_j - \sigma(\mathbf{w}^T\mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.

    if *error != 0* **then**

      ### Update the weights.

      $\mathbf{w} = \mathbf{w} + error \times x_j$

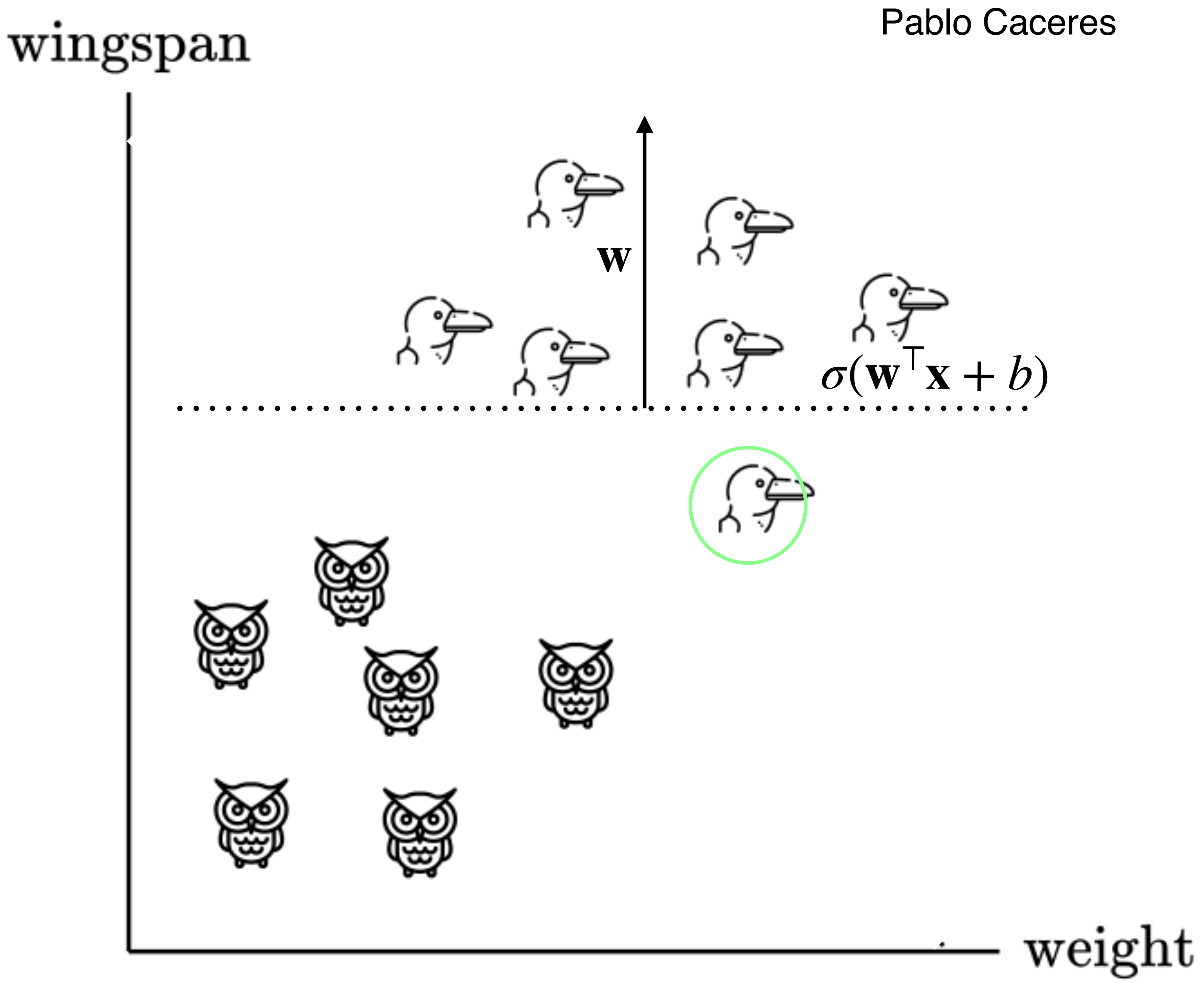
      ### Update the bias.

      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

$$\sigma(\mathbf{w}^\top\mathbf{x} + b) = \begin{cases} 1 & \text{if} \quad \mathbf{w}^\top\mathbf{x} + b > 0 \\ 0 & \text{else} \end{cases}$$



Pablo Caceres

Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.

  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.

    $error = y_j - \sigma(\mathbf{w}^T\mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.

    if *error != 0* **then**

      ### Update the weights.

      $\mathbf{w} = \mathbf{w} + error \times x_j$

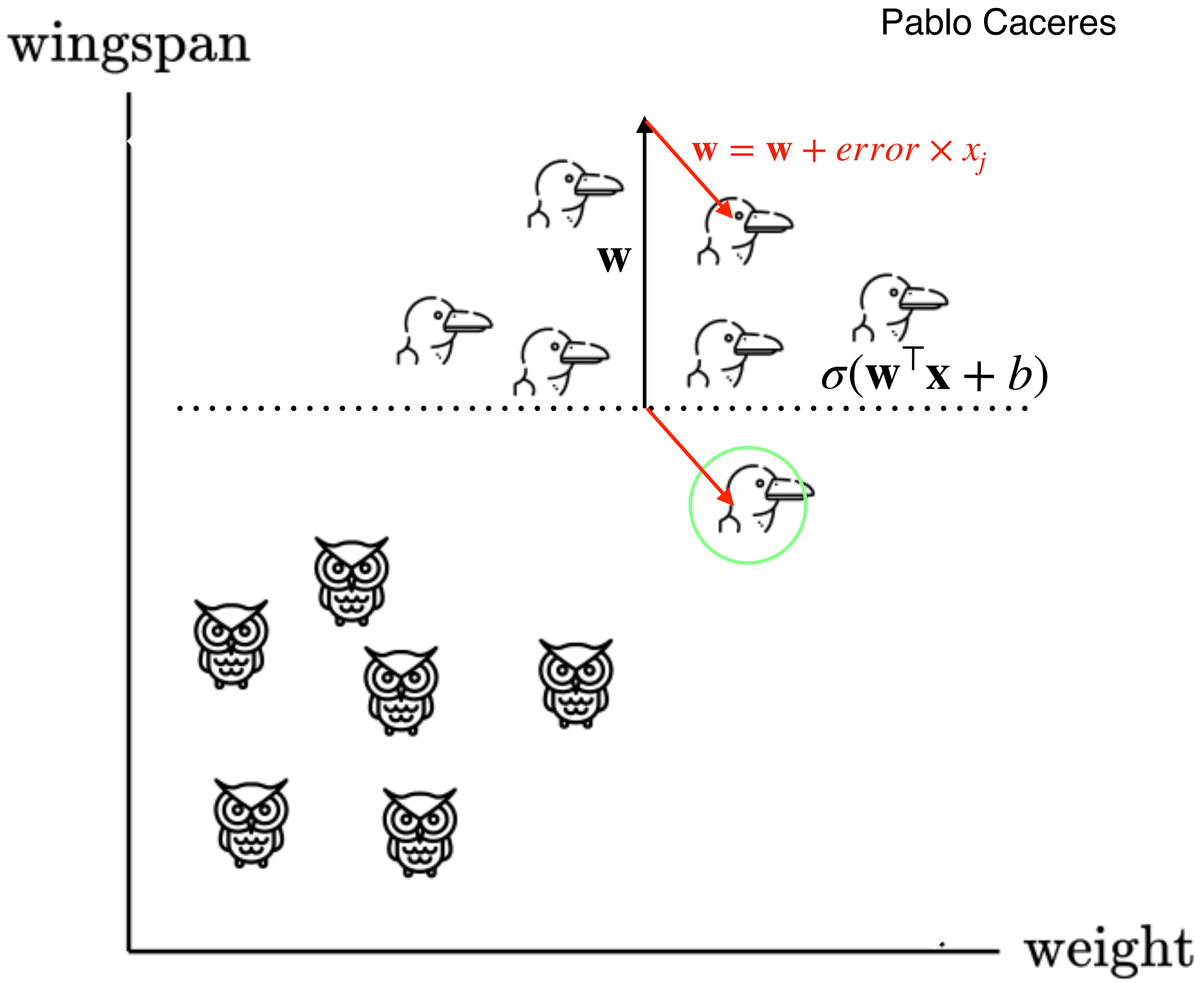
      ### Update the bias.

      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

$$\sigma(\mathbf{w}^\top\mathbf{x} + b) = \begin{cases} 1 & \text{if} \quad \mathbf{w}^\top\mathbf{x} + b > 0 \\ 0 & \text{else} \end{cases}$$



Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.

  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.

    $error = y_j - \sigma(\mathbf{w}^T\mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.

    if *error != 0* **then**

      ### Update the weights.

      $\mathbf{w} = \mathbf{w} + error \times x_j$

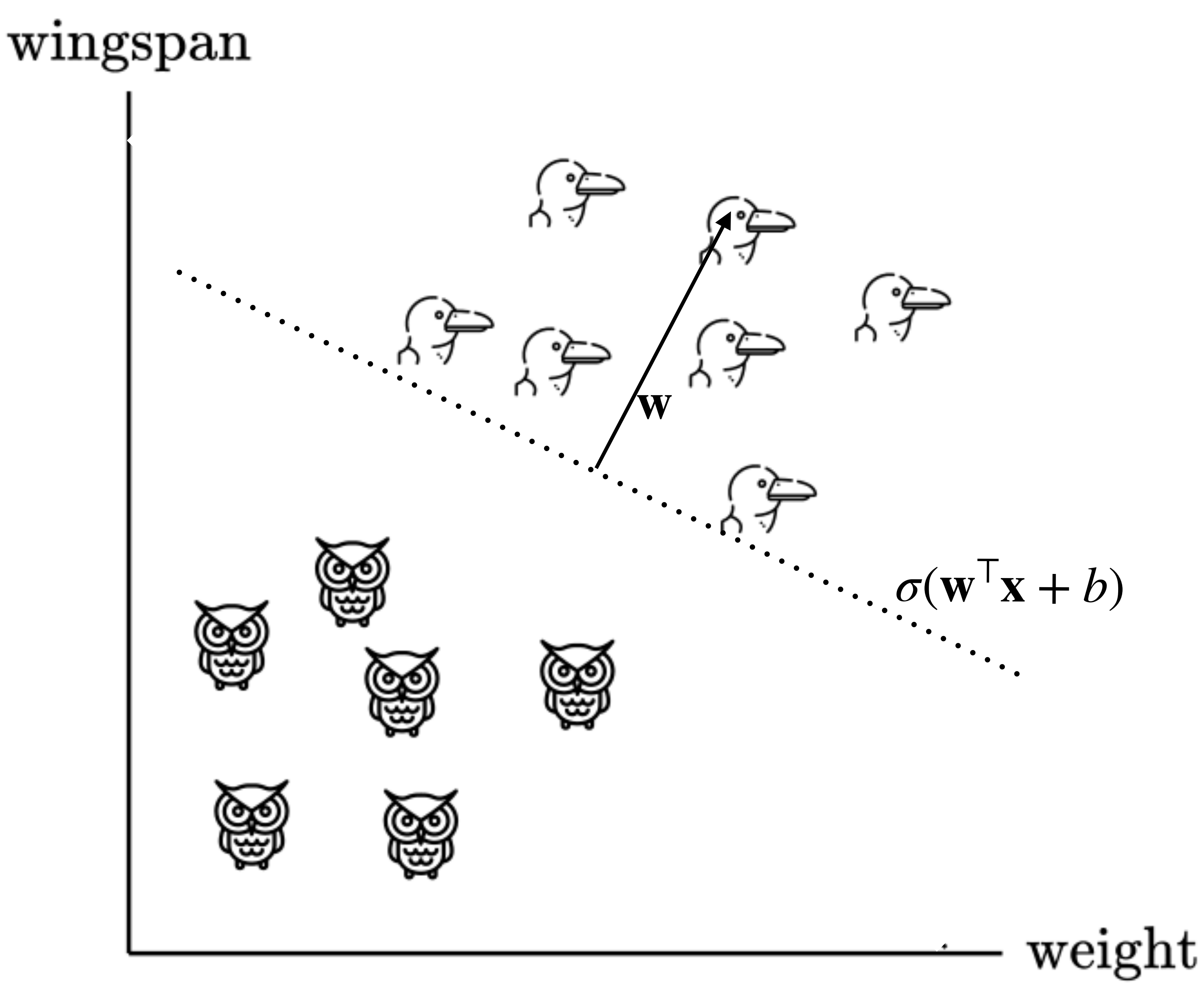
      ### Update the bias.

      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

$$\sigma(\mathbf{w}^\top\mathbf{x} + b) = \begin{cases} 1 & \text{if} \quad \mathbf{w}^\top\mathbf{x} + b > 0 \\ 0 & \text{else} \end{cases}$$



Perceptron learning rule

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$. (weight, wingspan) Owl=0 vs. Albatross=1

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.

  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.

    $error = y_j - \sigma(\mathbf{w}^T \mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.

    if *error != 0* **then**

      ### Update the weights.

      $\mathbf{w} = \mathbf{w} + error \times x_j$

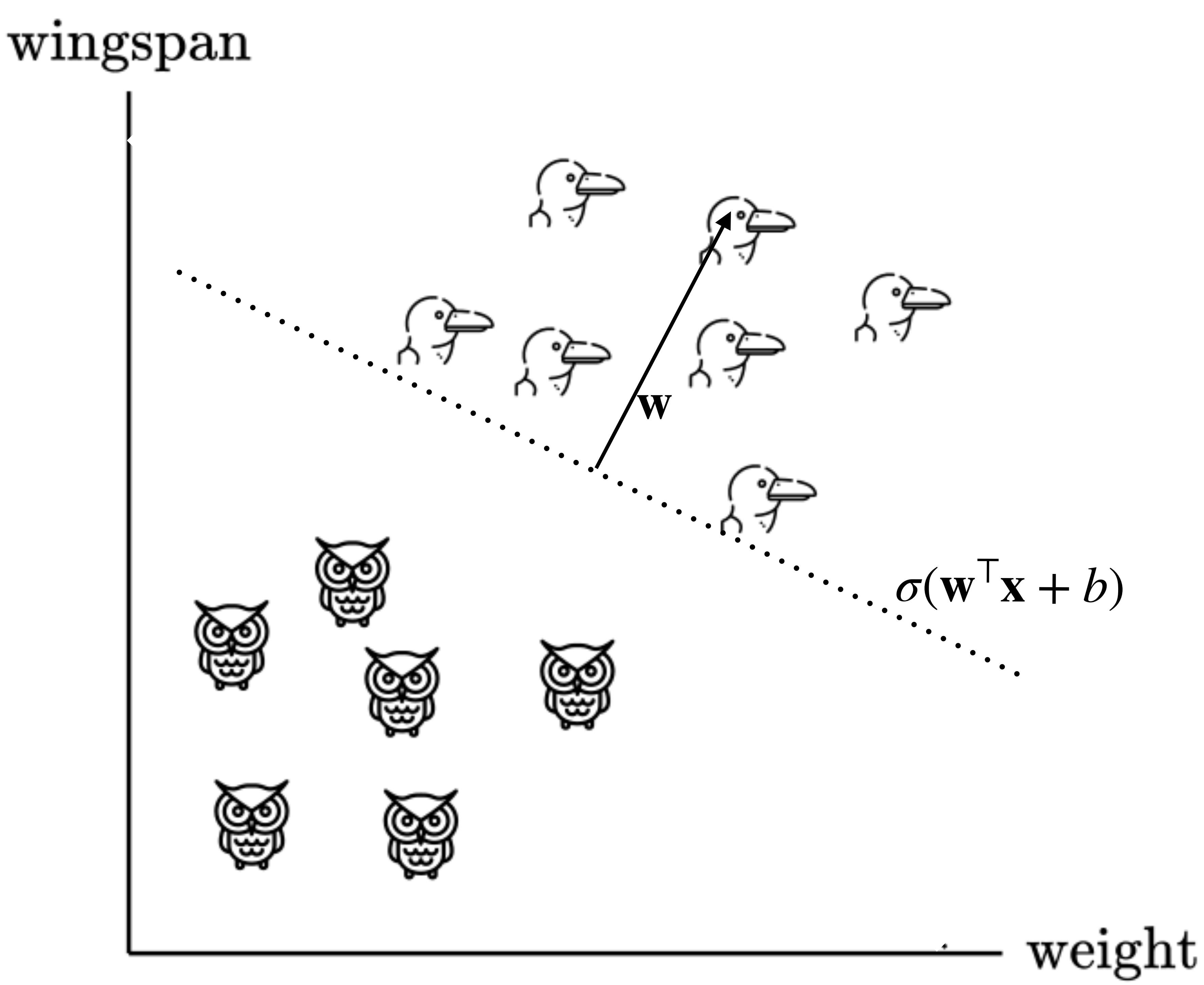
      ### Update the bias.

      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

$$\sigma(\mathbf{w}^\top \mathbf{x} + b) = \begin{cases} 1 & \text{if} \quad \mathbf{w}^\top \mathbf{x} + b > 0 \\ 0 & \text{else} \end{cases}$$

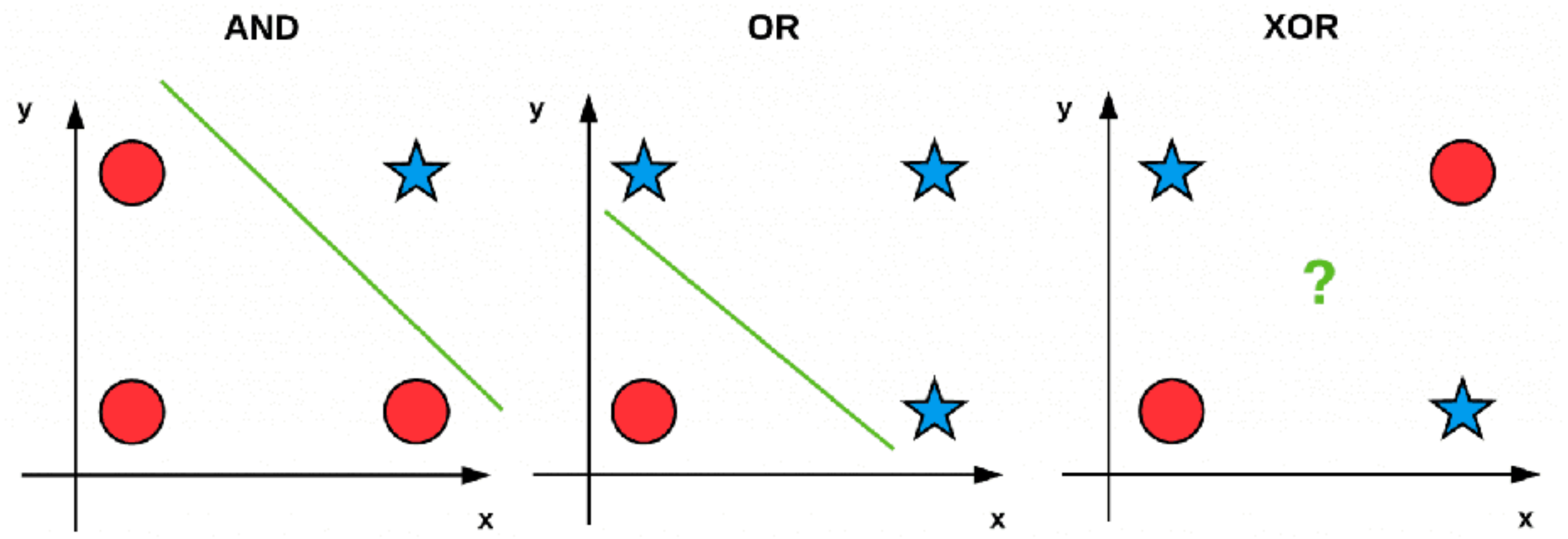


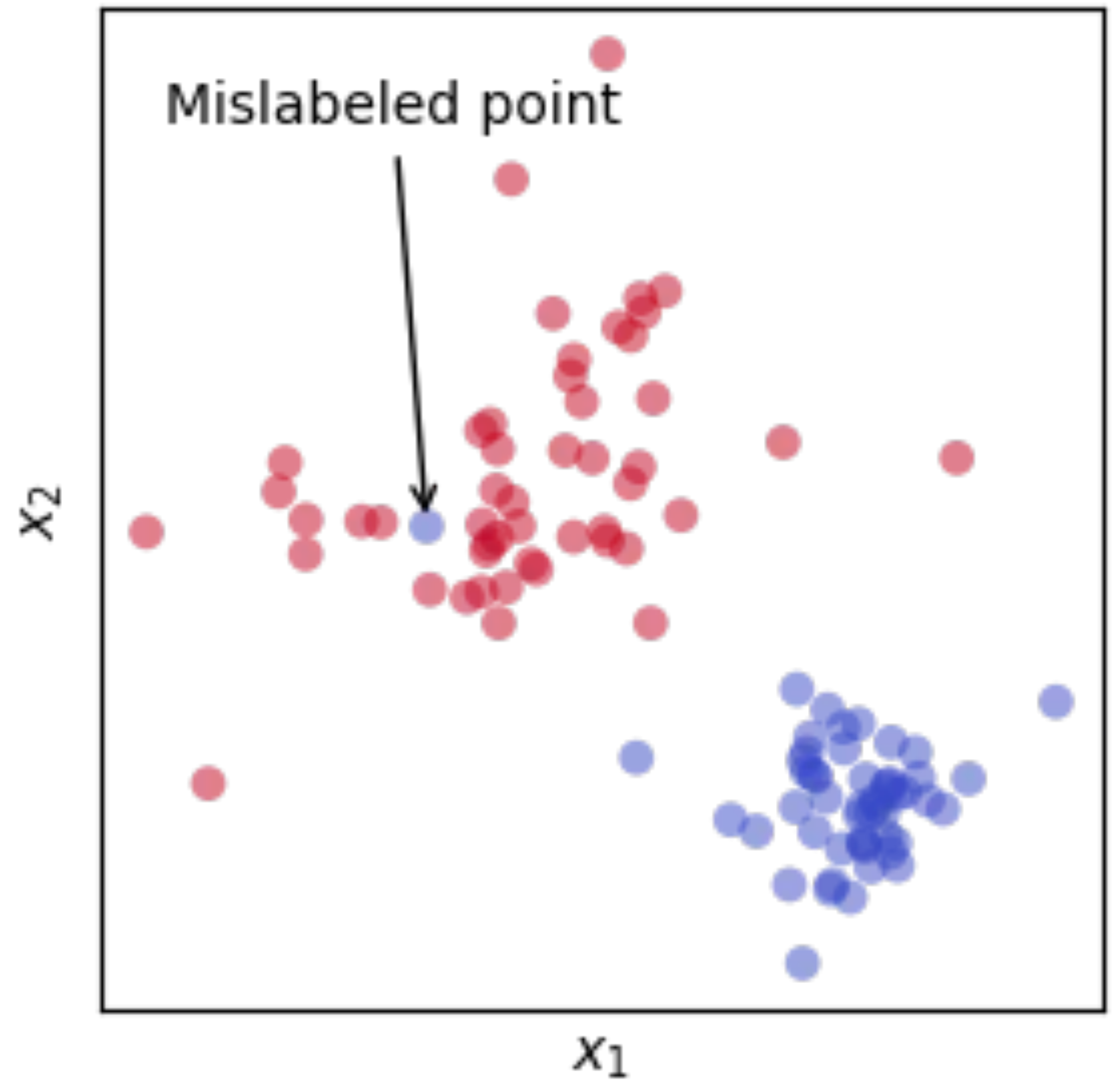
Guaranteed to converge if data is linearly separable

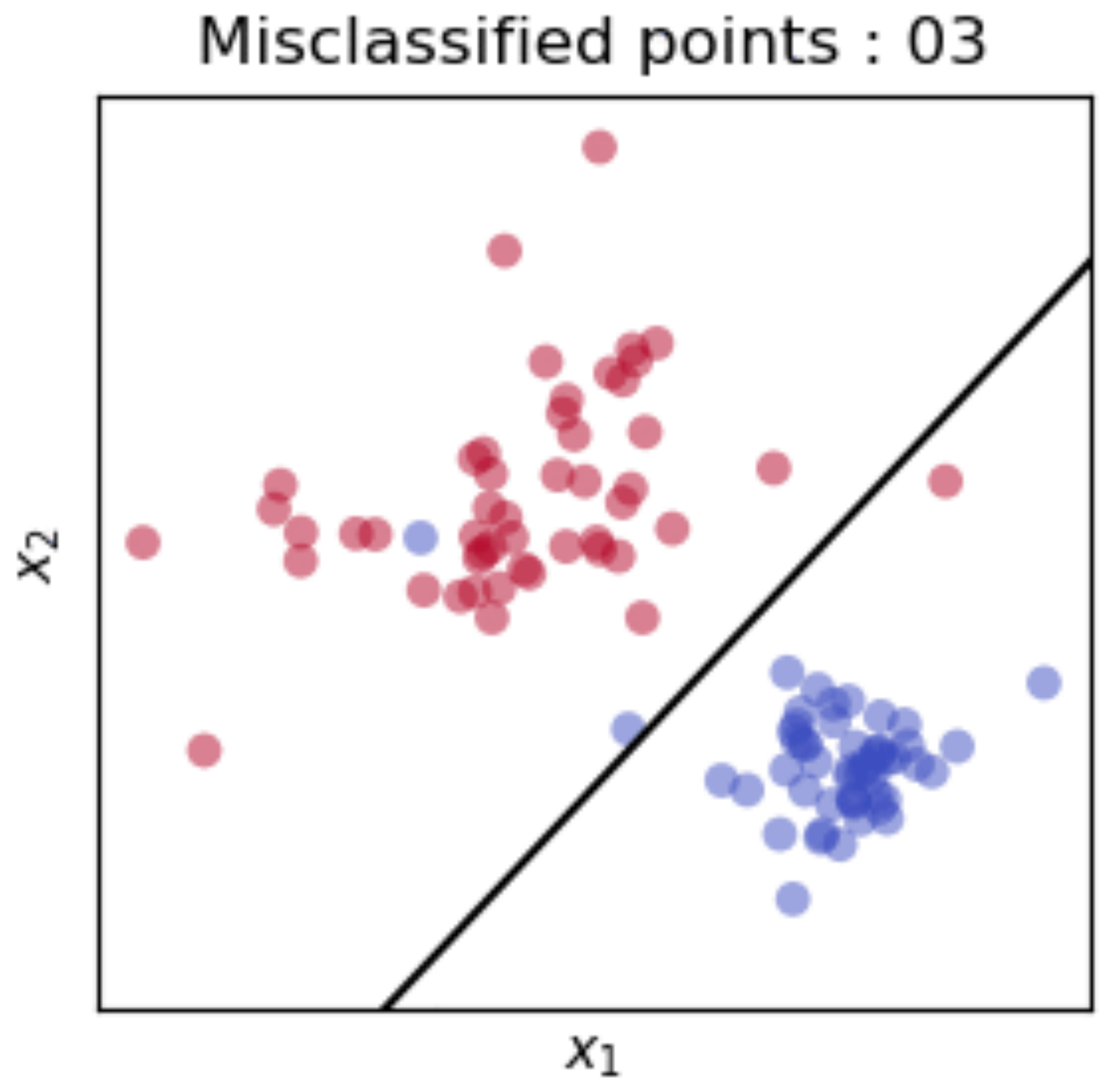
Limitations of linear separability

Adrian Rosebrock

- The perceptron can learn any linearly separable problem
    - But not all problems are lineary separable
- Even a single mislabeled data point in the data will throw the algorithm into chaos
- Enter the XOR problem and Minsky & Parpert (1969) critique
    - Argument: because a single neuron is unable to solve XOR, larger networks will also have similar problems
    - Therefore, the research program should be dropped



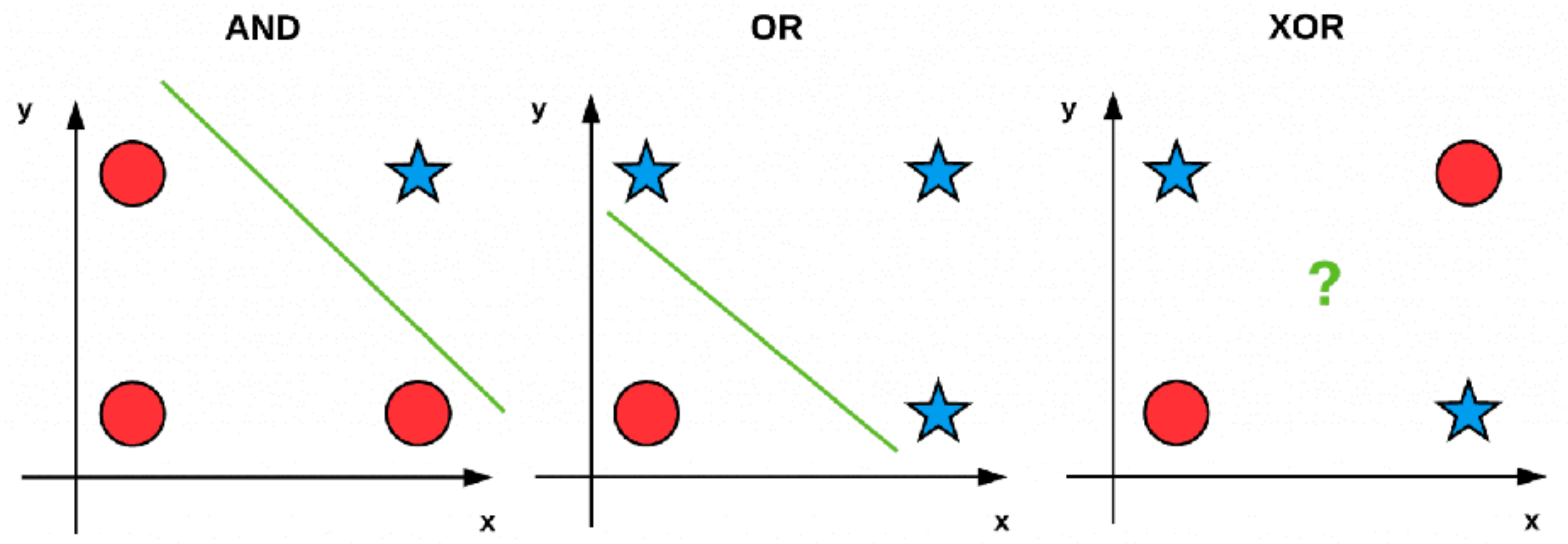


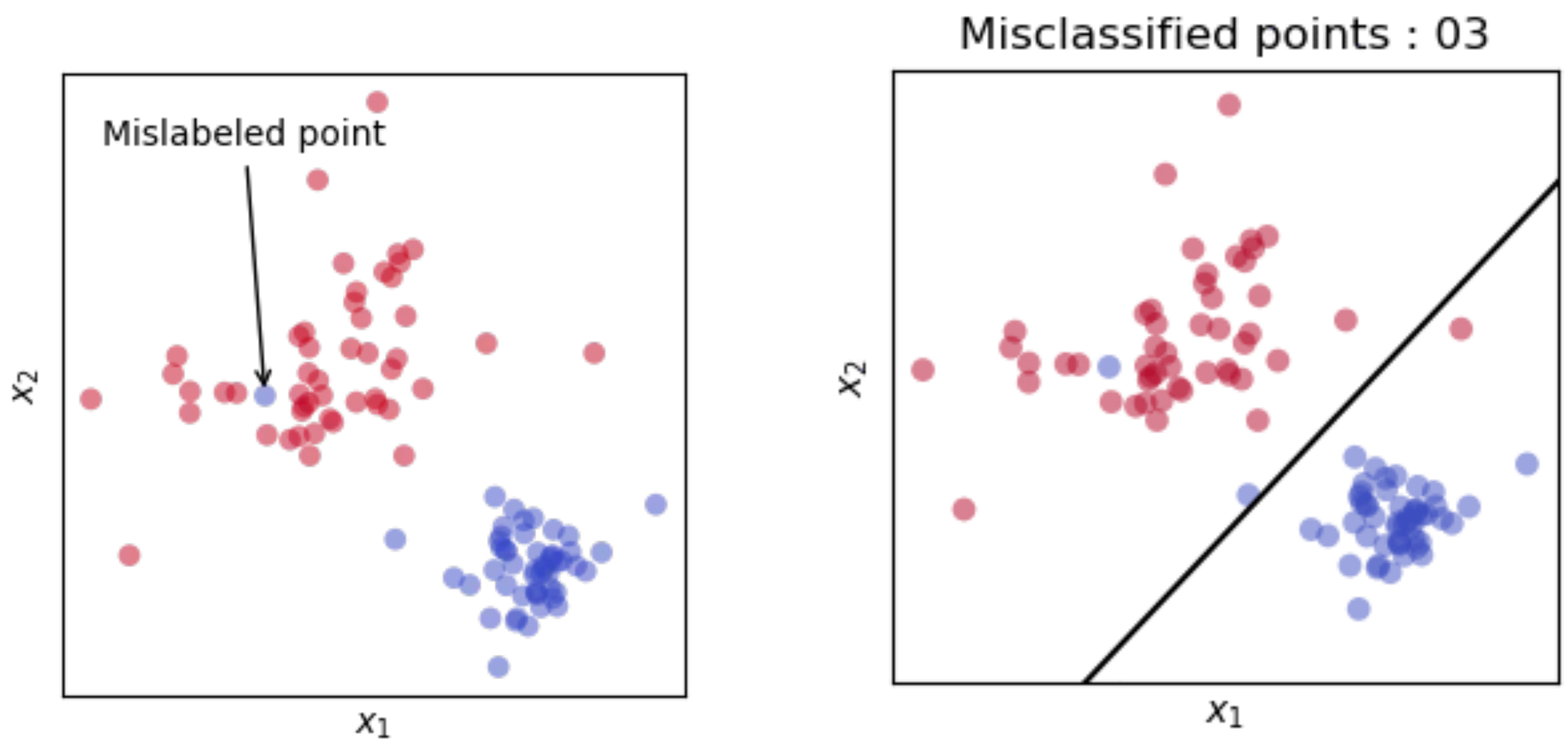


Limitations of linear separability

Adrian Rosebrock

- The perceptron can learn any linearly separable problem
  - But not all problems are lineary separable
- Even a single mislabeled data point in the data will throw the algorithm into chaos
- Enter the XOR problem and Minsky & Parpert (1969) critique
  - Argument: because a single neuron is unable to solve XOR, larger networks will also have similar problems
  - Therefore, the research program should be dropped





Addressing Minsky & Parpert's critiques

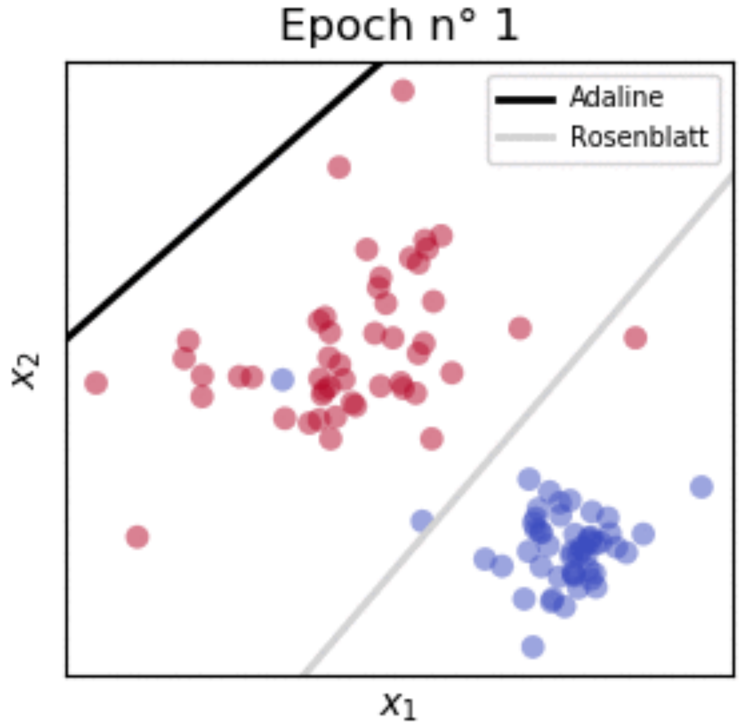
- Changing the learning rule
  - ADALINE adds robustness to training noise
- Adding more layers
  - While single neurons can only compute some logical predicates, *networks* of these neurons can compute any possible boolean function (Rosenblatt, 1962)
  - Multilayer Perceptron can solve XOR
- Changing the activation function
  - Beyond hard thresholds

Improving the Learning Rule

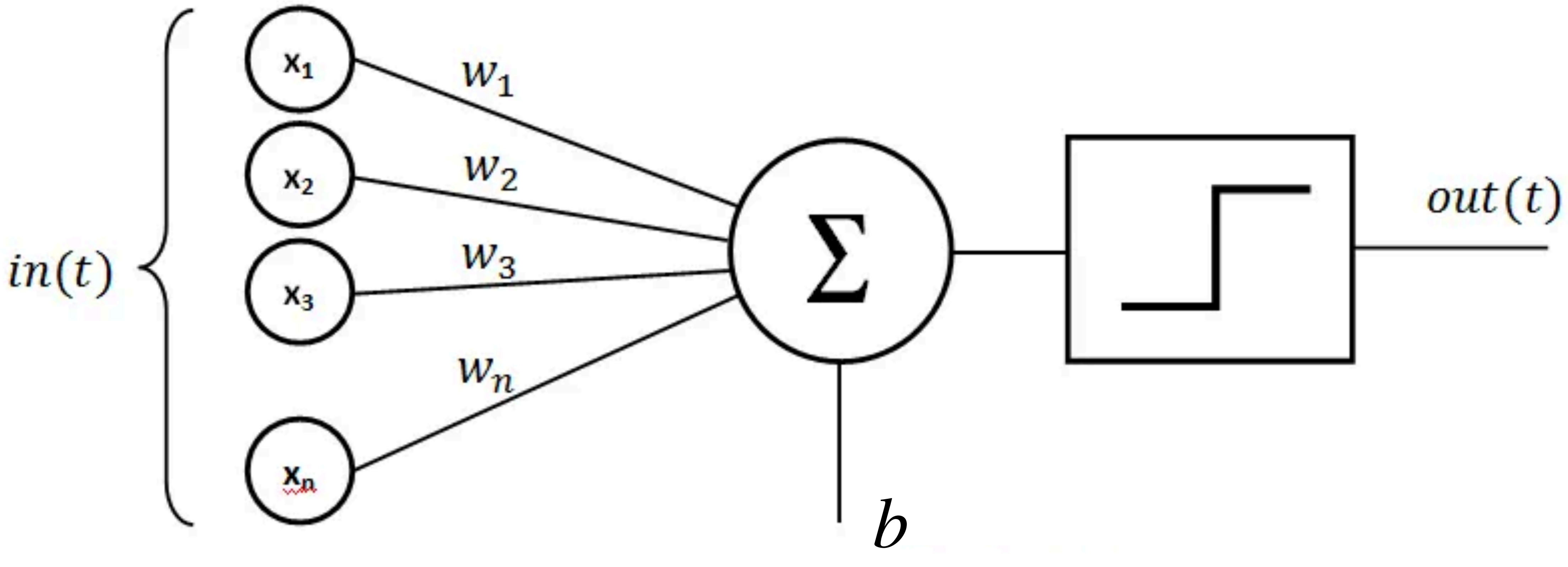
Adaptive Linear Element (ADALINE)

- Weight updates based on a *loss function* rather than the (binary) classification error
    - This uses the activation prior to the sigmoid step, allowing us to compute gradients
- We can use the Delta rule to minimize loss, which is equivalent to stochastic gradient descent for least-squares regression

ADALINE is more robust to training noise:



ADALINE



MSE

$$\mathscr{L}(\mathbf{w}, b) = \frac{1}{2} \sum_{i=1}^{m} \left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i \right)^2$$

Weight update

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \mathbf{w}$$

$$\Delta \mathbf{w} = -\frac{\partial \mathscr{L}}{\partial \mathbf{w}}$$

$$= \sum_{i=1}^{m} \left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i \right) \mathbf{x}_i$$

Bias update

$$b \leftarrow b + \alpha \Delta b$$

$$\Delta b = -\frac{\partial \mathscr{L}}{\partial b}$$

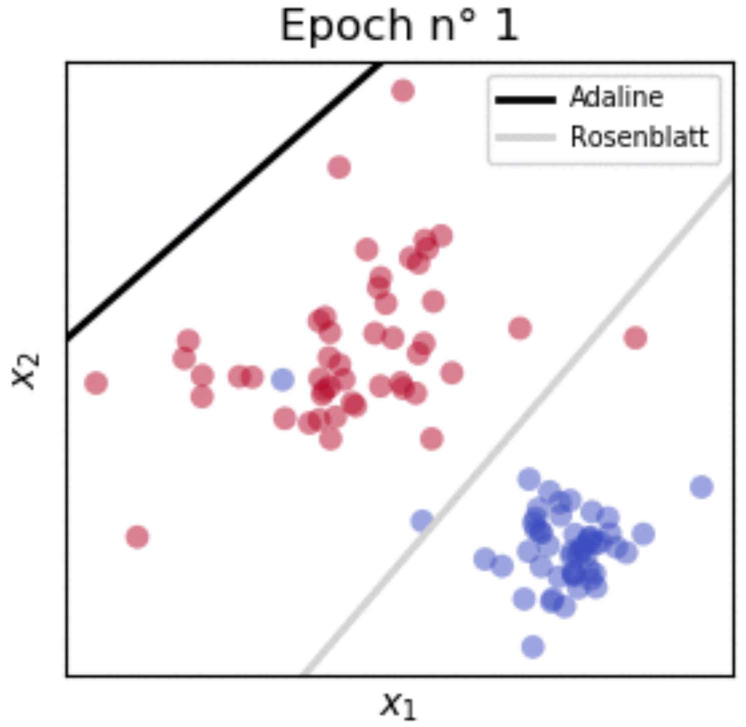
$$= \sum_{i=1}^{m} (\mathbf{w}^\top \mathbf{x}_i + b) - y_i$$

Improving the Learning Rule

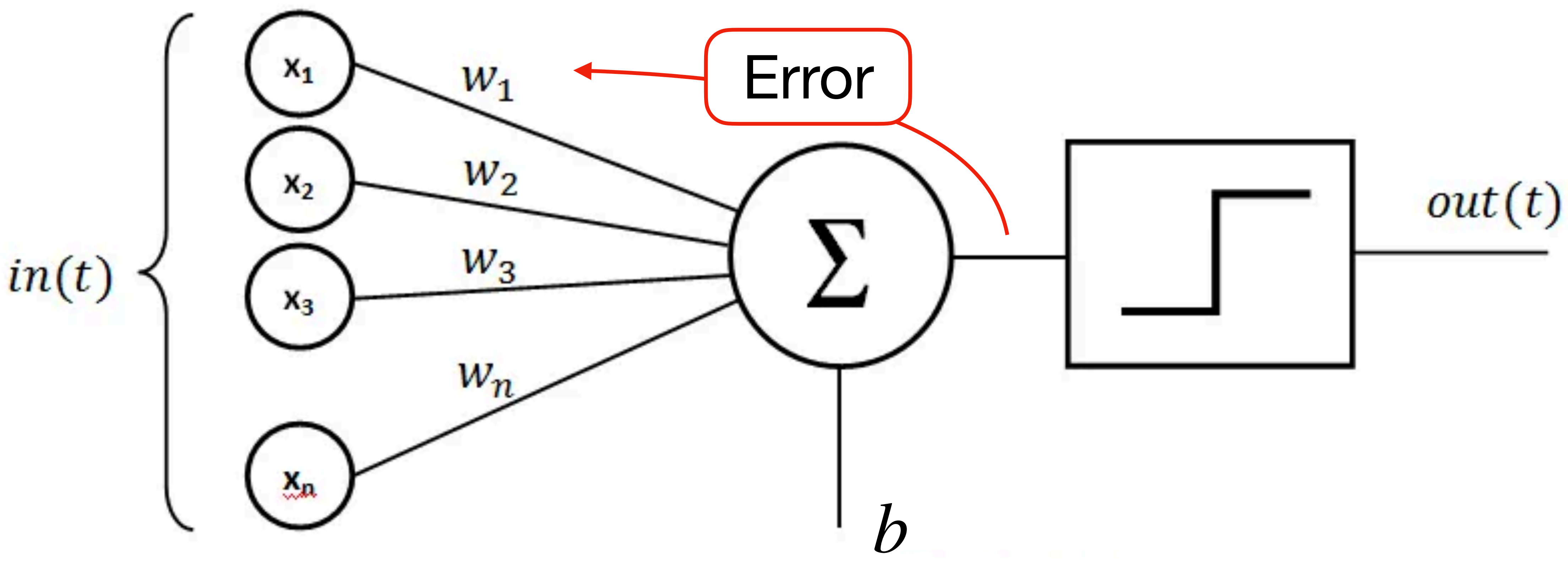
Adaptive Linear Element (ADALINE)

- Weight updates based on a *loss function* rather than the (binary) classification error
  - This uses the activation prior to the sigmoid step, allowing us to compute gradients
- We can use the Delta rule to minimize loss, which is equivalent to stochastic gradient descent for least-squares regression

ADALINE is more robust to training noise:



ADALINE



MSE

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{2}\sum_{i=1}^{m}\left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i\right)^2$$

Weight update

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \mathbf{w}$$

$$\Delta \mathbf{w} = -\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^{m}\left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i\right)\mathbf{x}_i$$

Bias update

$$b \leftarrow b + \alpha \Delta b$$

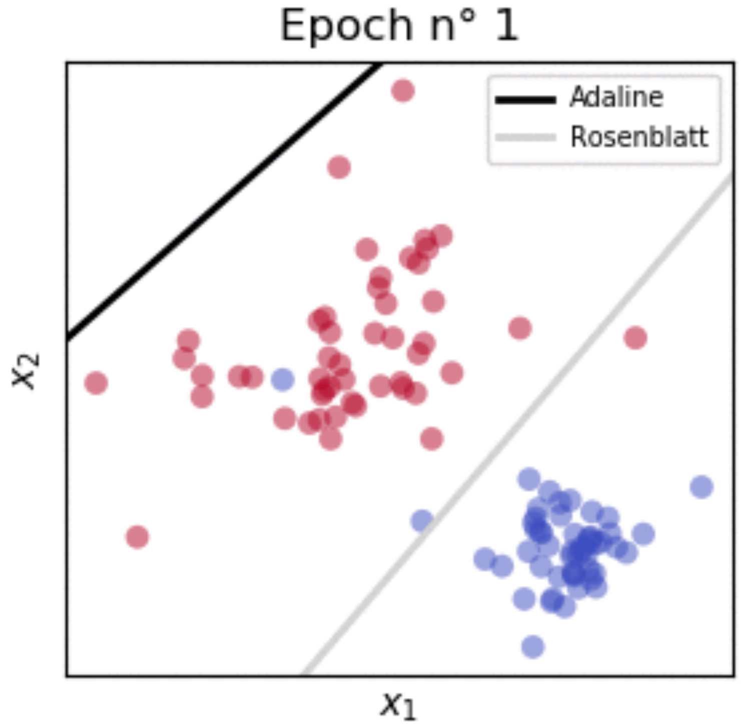
$$\Delta b = -\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{m}(\mathbf{w}^\top \mathbf{x}_i + b) - y_i$$

Improving the Learning Rule

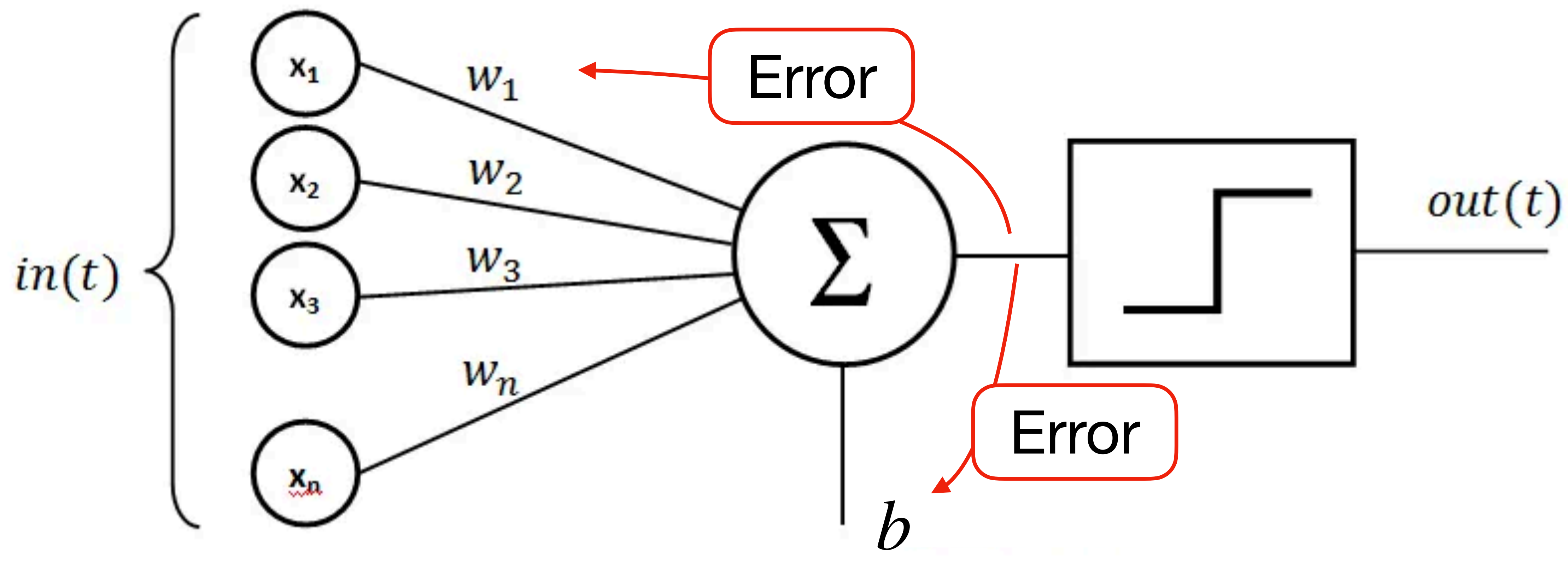
Adaptive Linear Element (ADALINE)

- Weight updates based on a *loss function* rather than the (binary) classification error
    - This uses the activation prior to the sigmoid step, allowing us to compute gradients
- We can use the Delta rule to minimize loss, which is equivalent to stochastic gradient descent for least-squares regression

ADALINE is more robust to training noise:



ADALINE



MSE

$$\mathscr{L}(\mathbf{w}, b) = \frac{1}{2}\sum_{i=1}^{m}\left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i\right)^2$$

Weight update

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \mathbf{w}$$

$$\Delta \mathbf{w} = -\frac{\partial \mathscr{L}}{\partial \mathbf{w}}$$

$$= \sum_{i=1}^{m}\left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i\right)\mathbf{x}_i$$

Bias update

$$b \leftarrow b + \alpha \Delta b$$

$$\Delta b = -\frac{\partial \mathscr{L}}{\partial b}$$

$$= \sum_{i=1}^{m}(\mathbf{w}^\top \mathbf{x}_i + b) - y_i$$

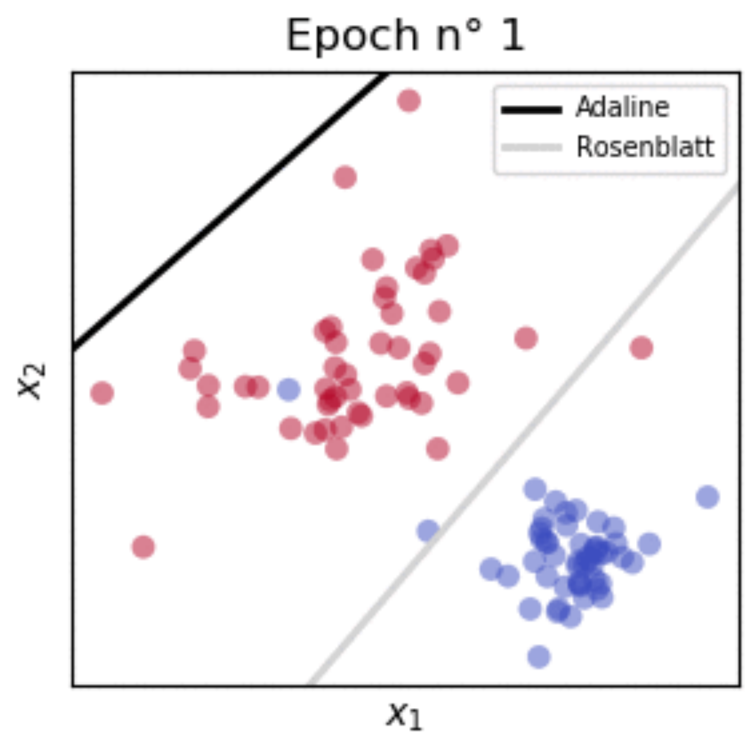
Widrow & Hoff, 1960

Improving the Learning Rule

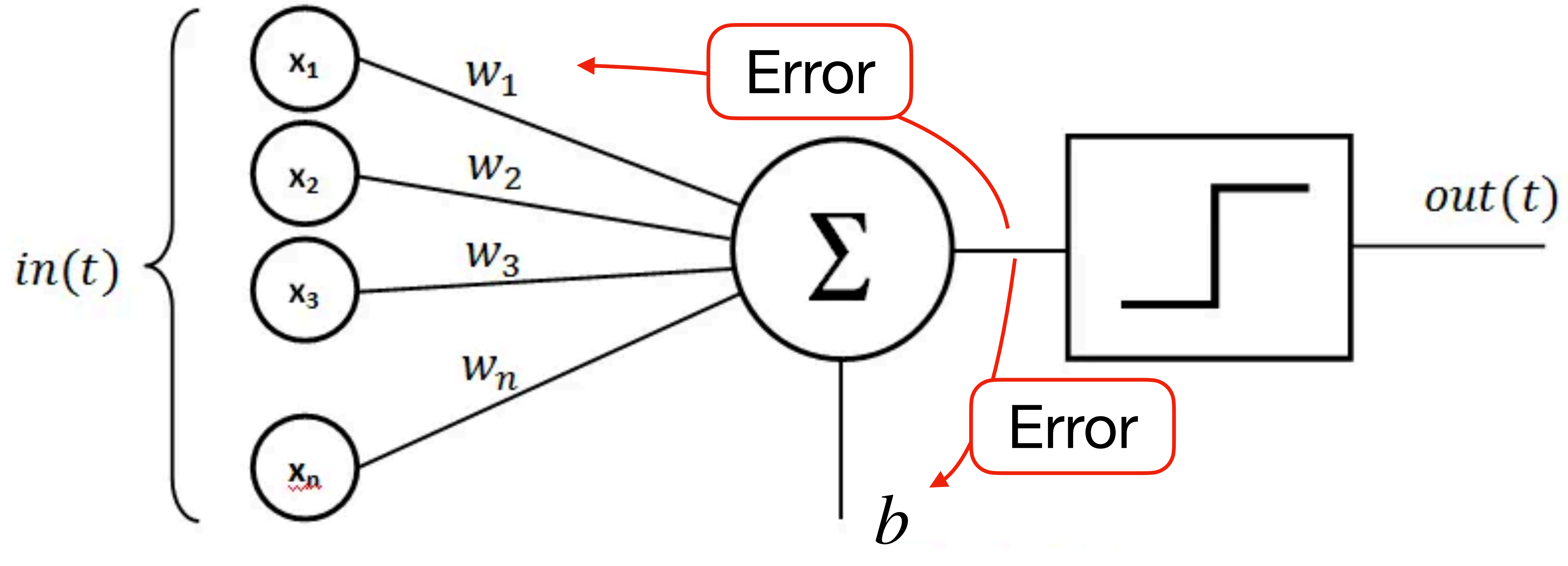
Adaptive Linear Element (ADALINE)

- Weight updates based on a *loss function* rather than the (binary) classification error
  - This uses the activation prior to the sigmoid step, allowing us to compute gradients
- We can use the Delta rule to minimize loss, which is equivalent to stochastic gradient descent for least-squares regression

ADALINE is more robust to training noise:



ADALINE



MSE

$$\mathscr{L}(\mathbf{w}, b) = \frac{1}{2}\sum_{i=1}^{m}\left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i\right)^2$$

Weight update

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \mathbf{w}$$

$$\Delta \mathbf{w} = -\frac{\partial \mathscr{L}}{\partial \mathbf{w}}$$

$$= \sum_{i=1}^{m}\left((\mathbf{w}^\top \mathbf{x}_i + b) - y_i\right)\mathbf{x}_i$$

Bias update

$$b \leftarrow b + \alpha \Delta b$$

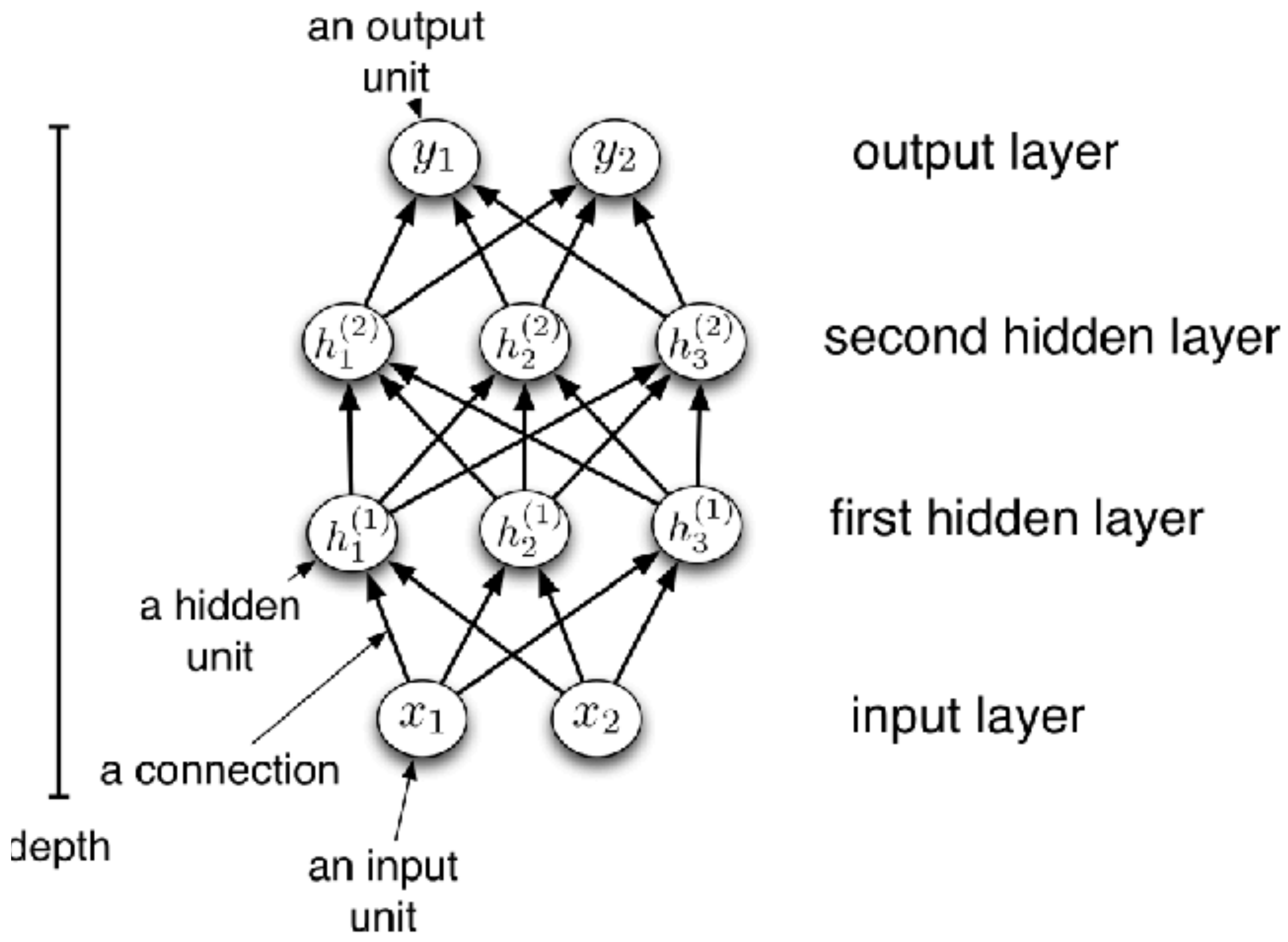
$$\Delta b = -\frac{\partial \mathscr{L}}{\partial b}$$

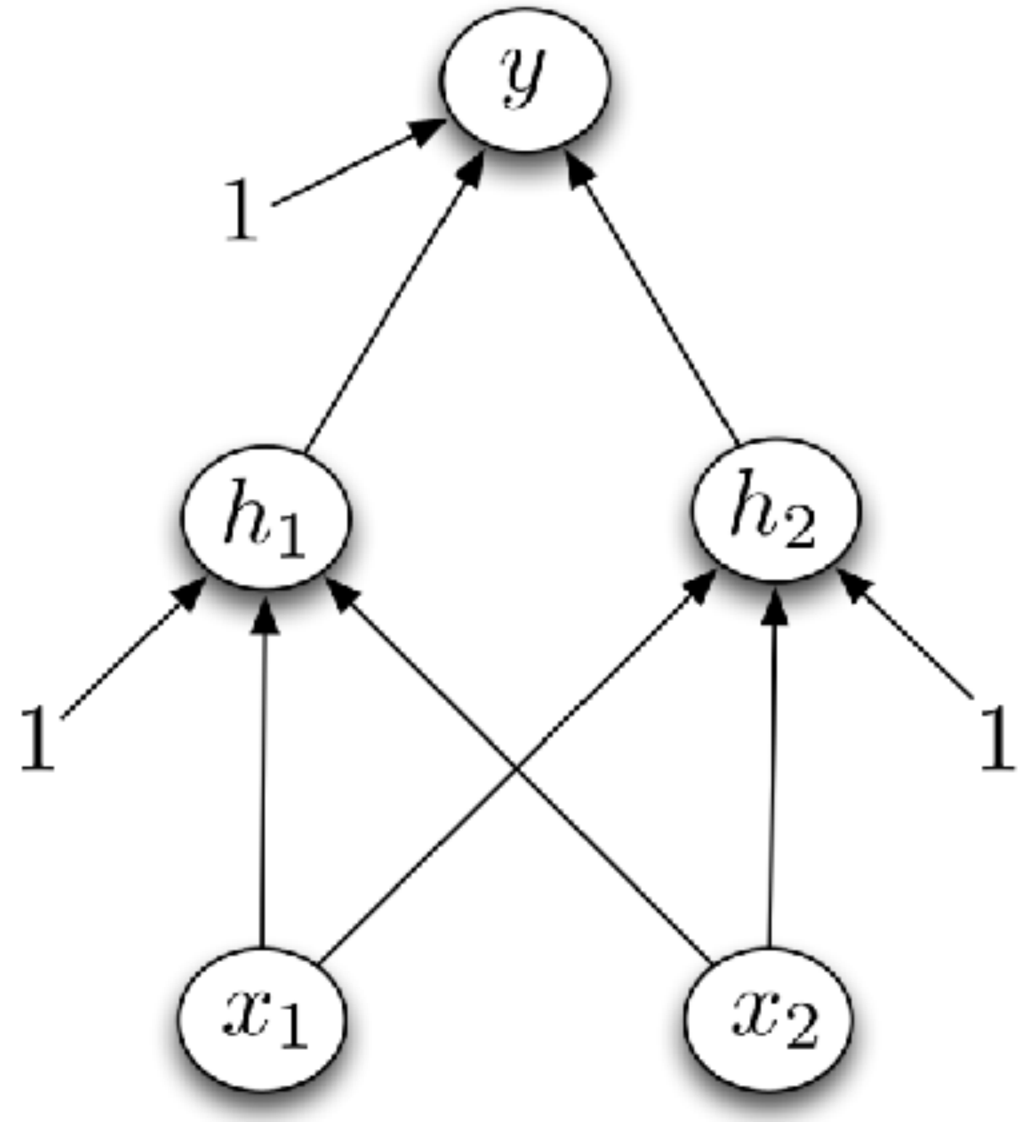
$$= \sum_{i=1}^{m}(\mathbf{w}^\top \mathbf{x}_i + b) - y_i$$

Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0			
1	1			
1	0			
0	1			

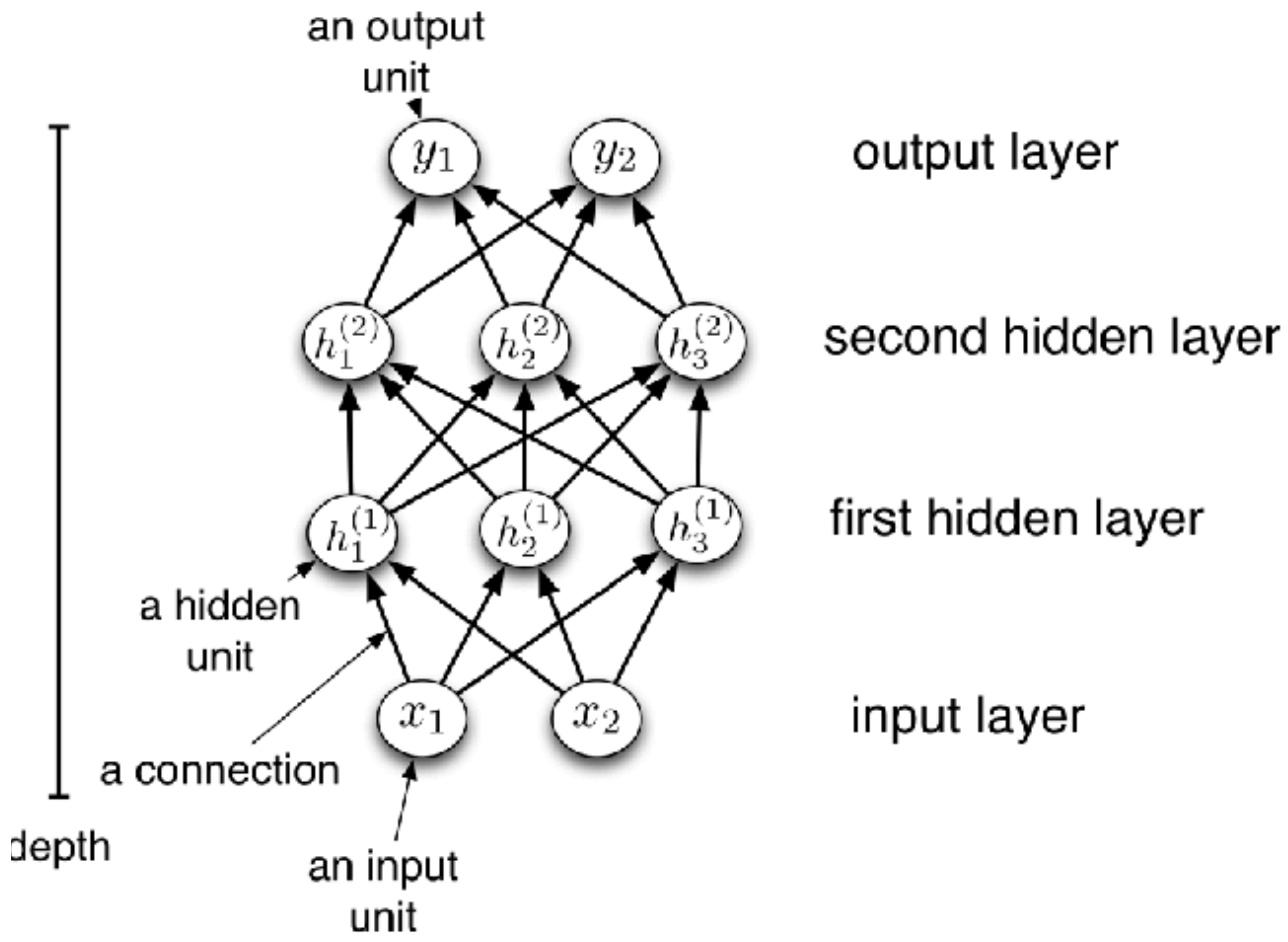


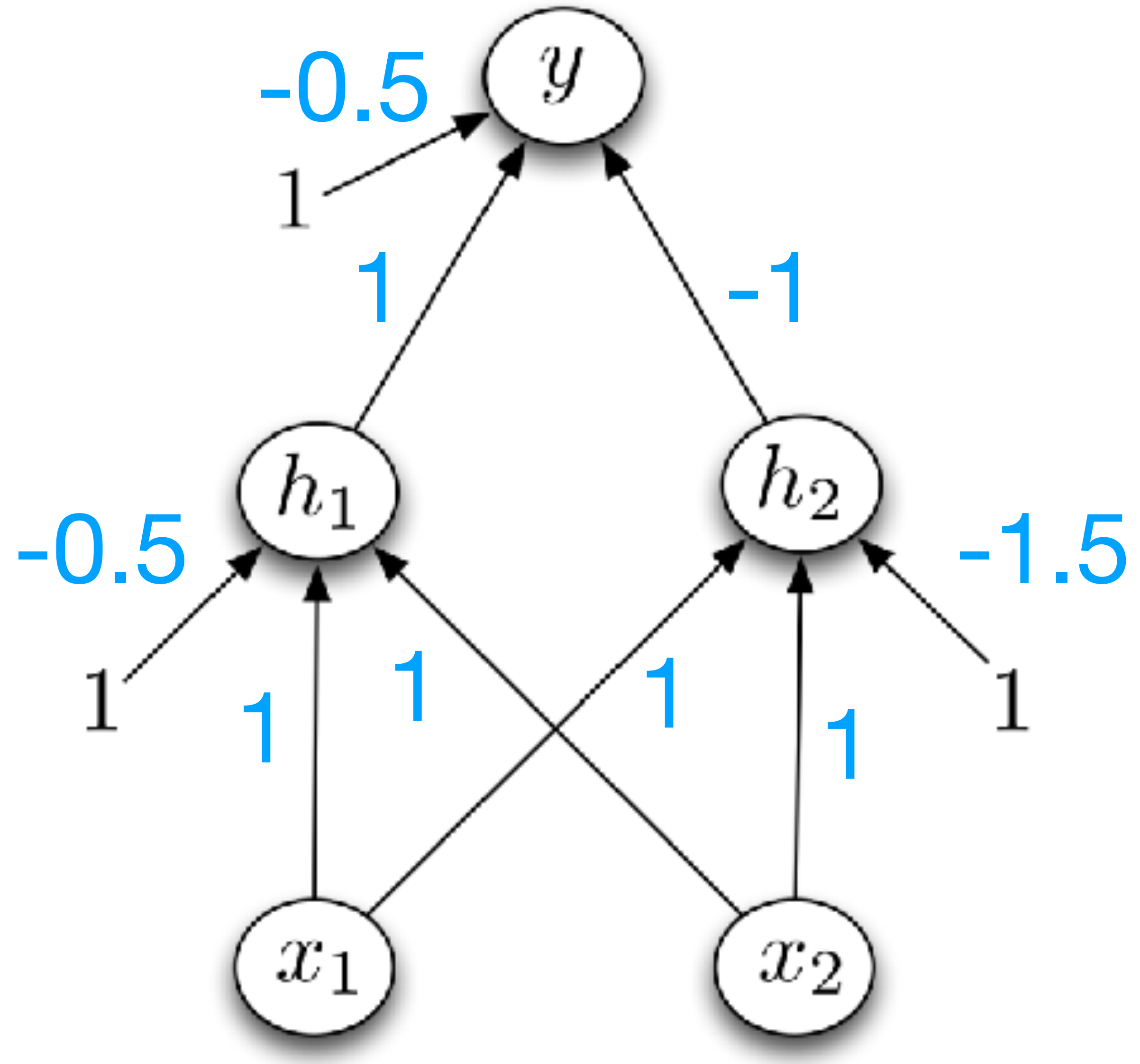


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0			
1	1			
1	0			
0	1			

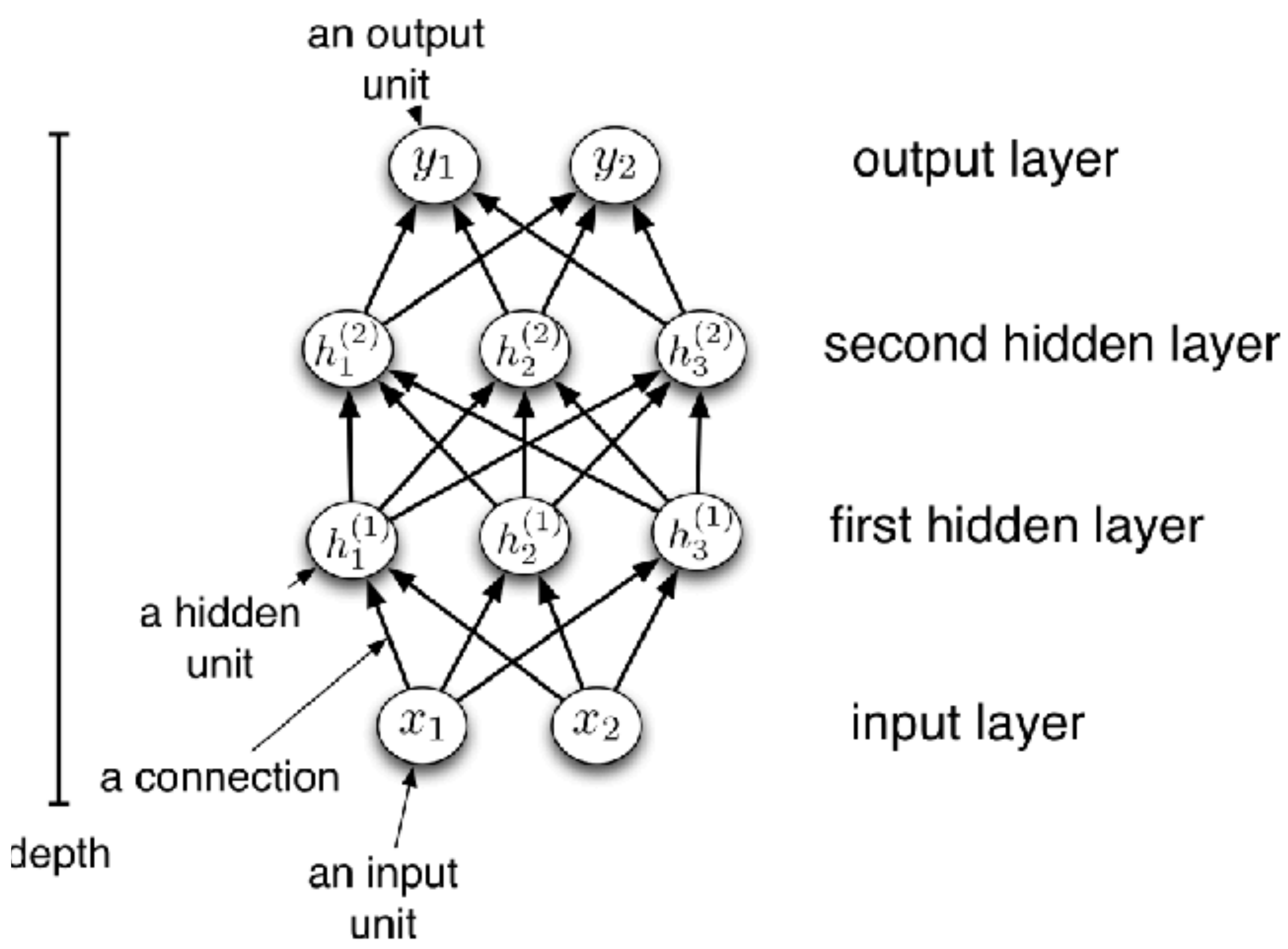


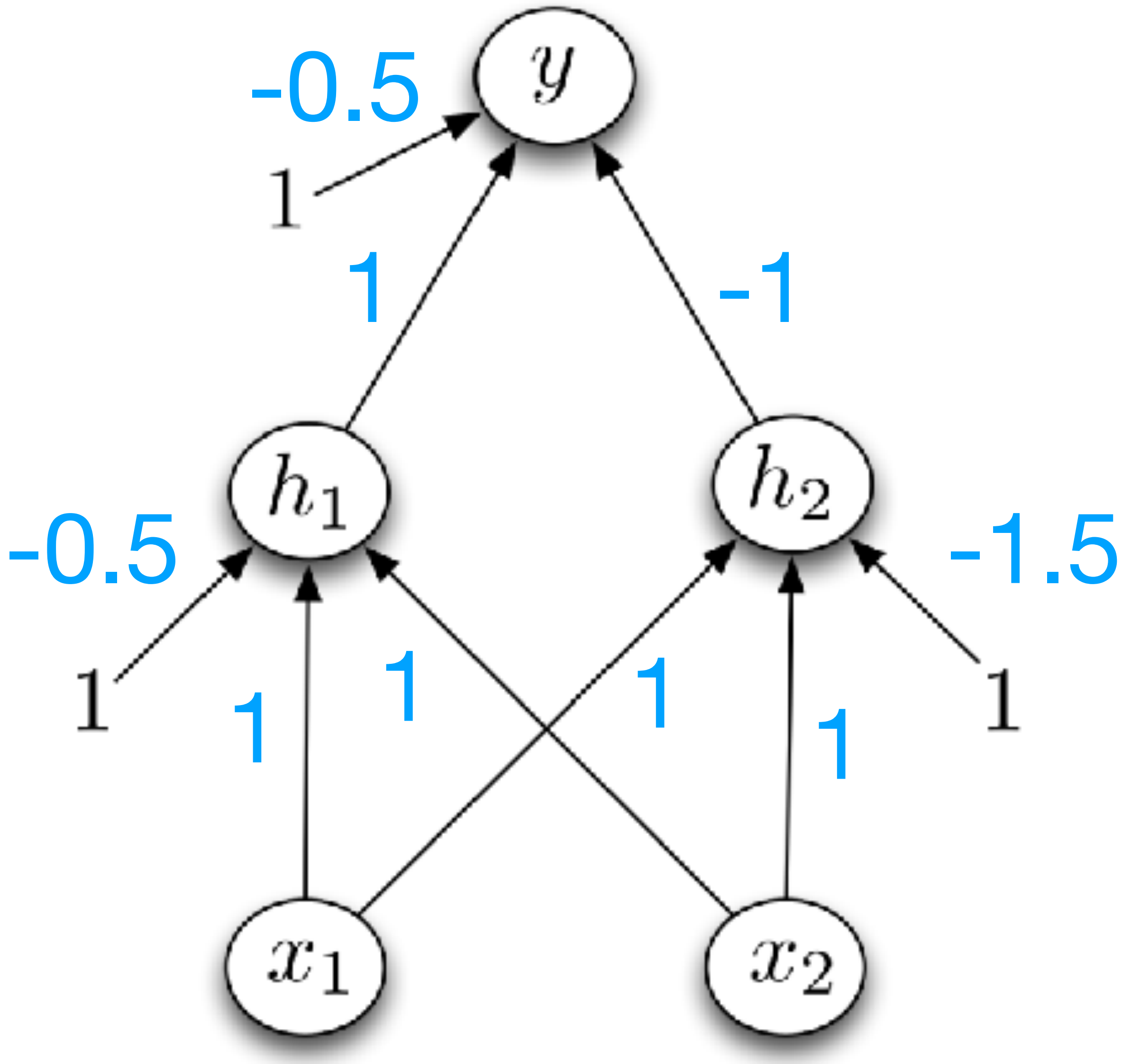


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$		
1	1			
1	0			
0	1			

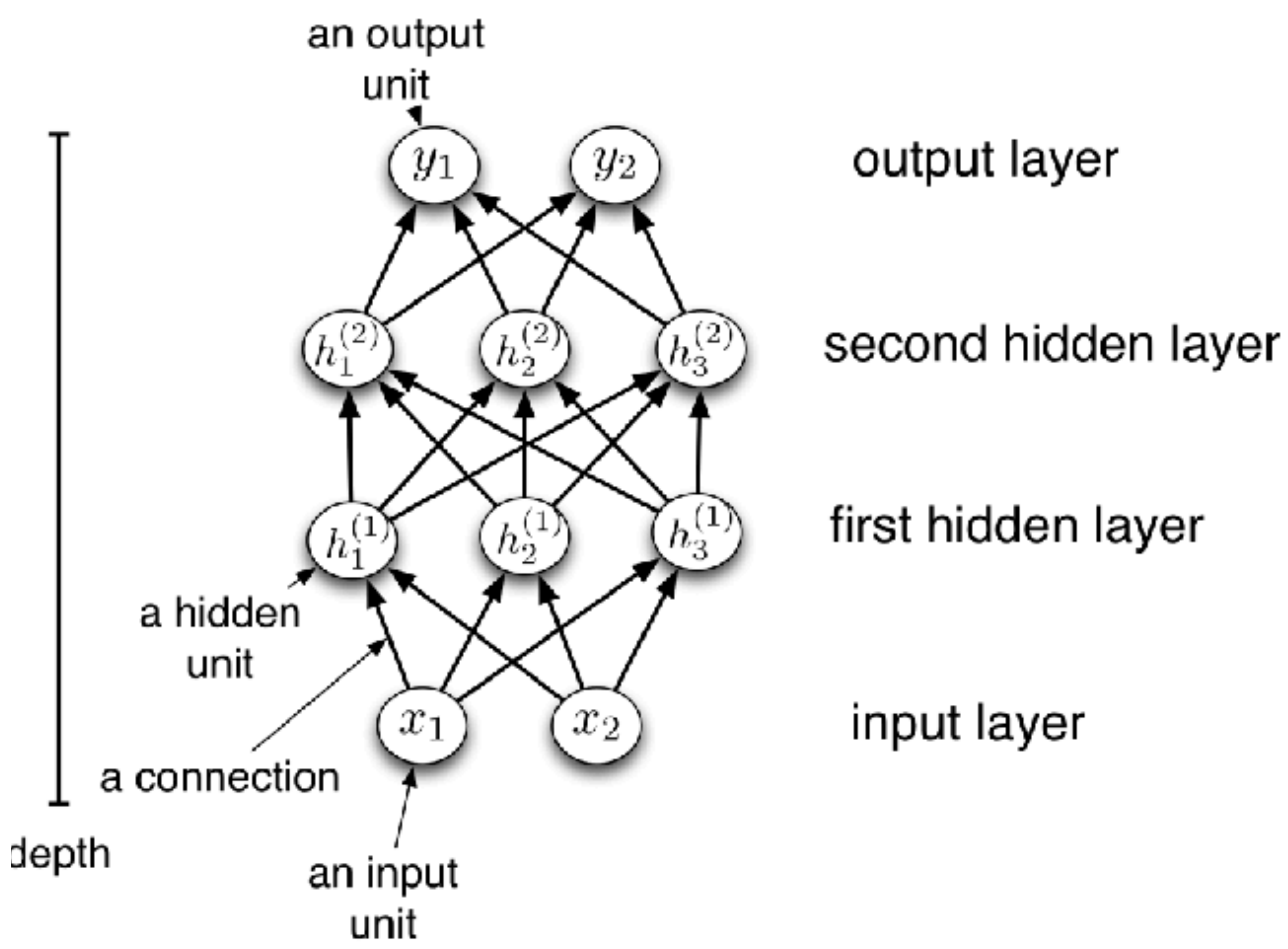


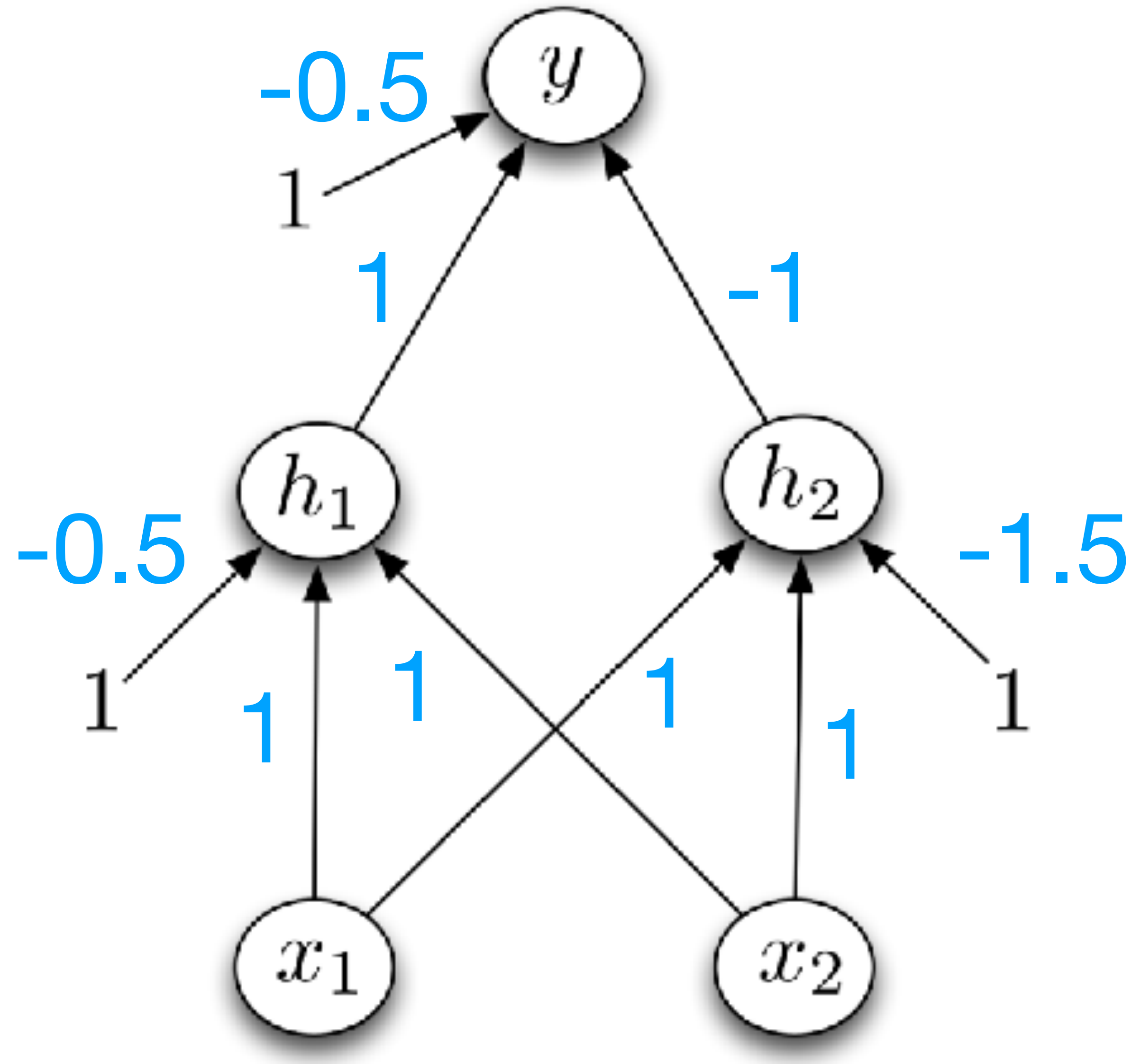


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	
1	1			
1	0			
0	1			

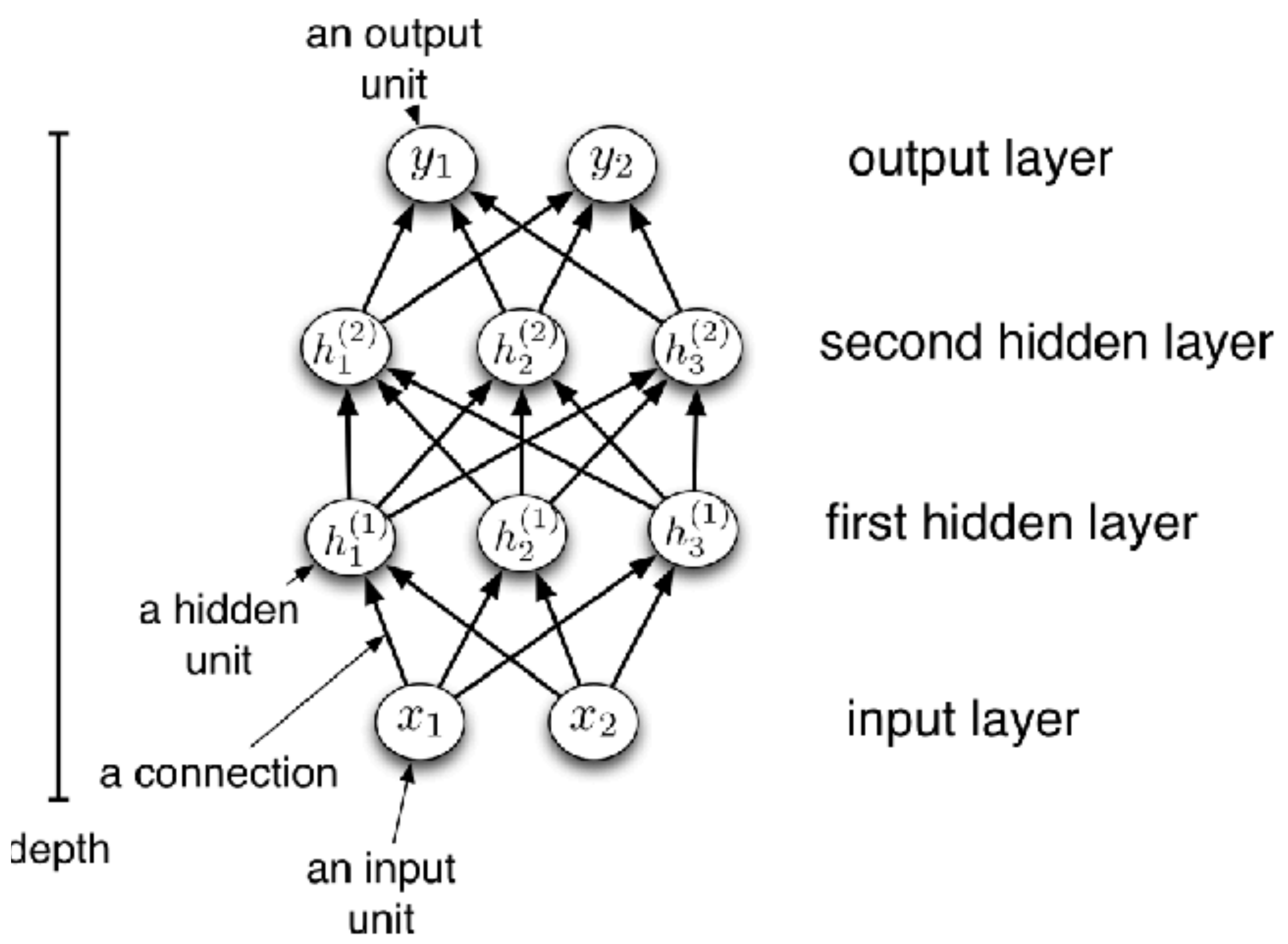


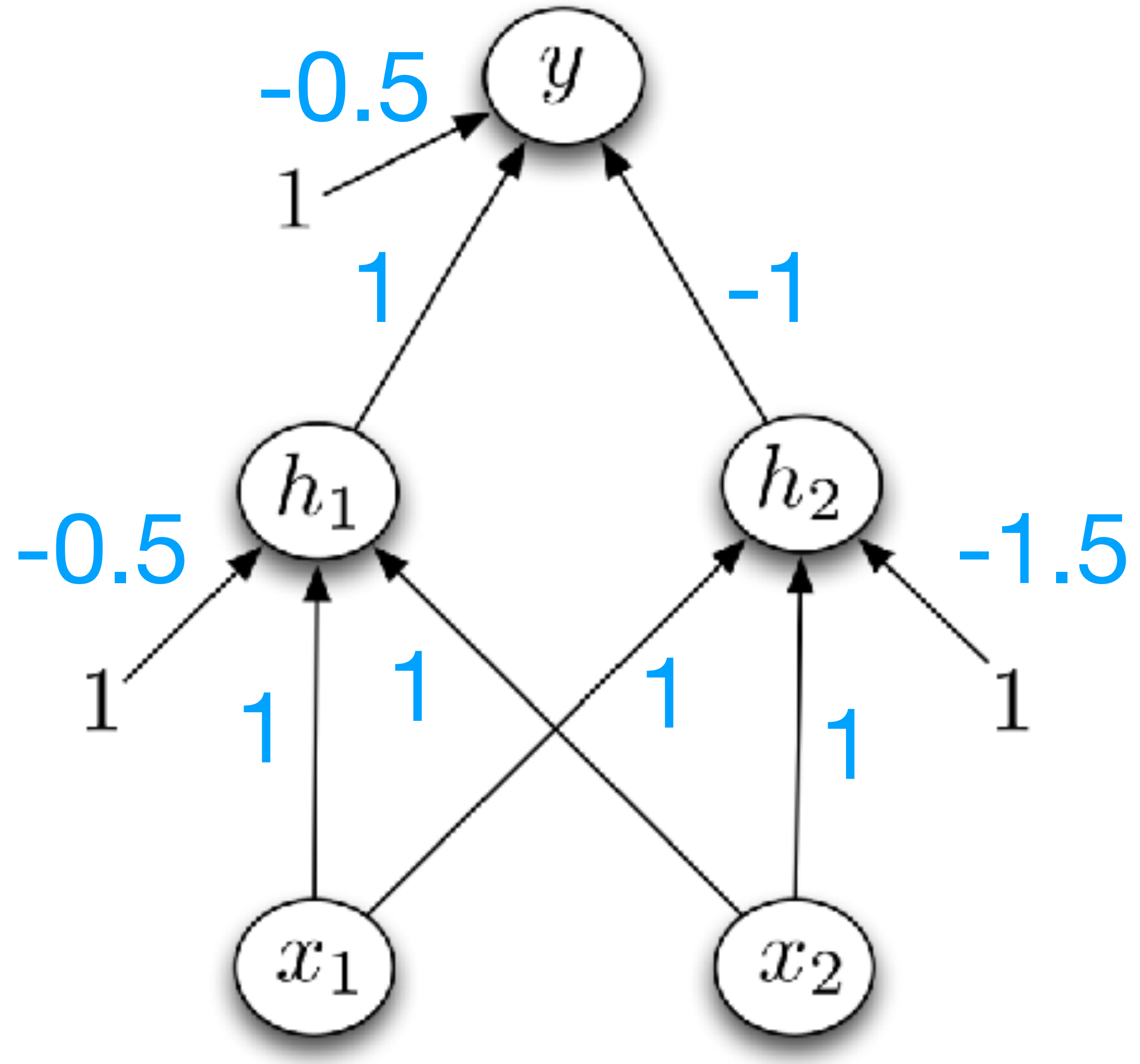


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1, h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1			
1	0			
0	1			

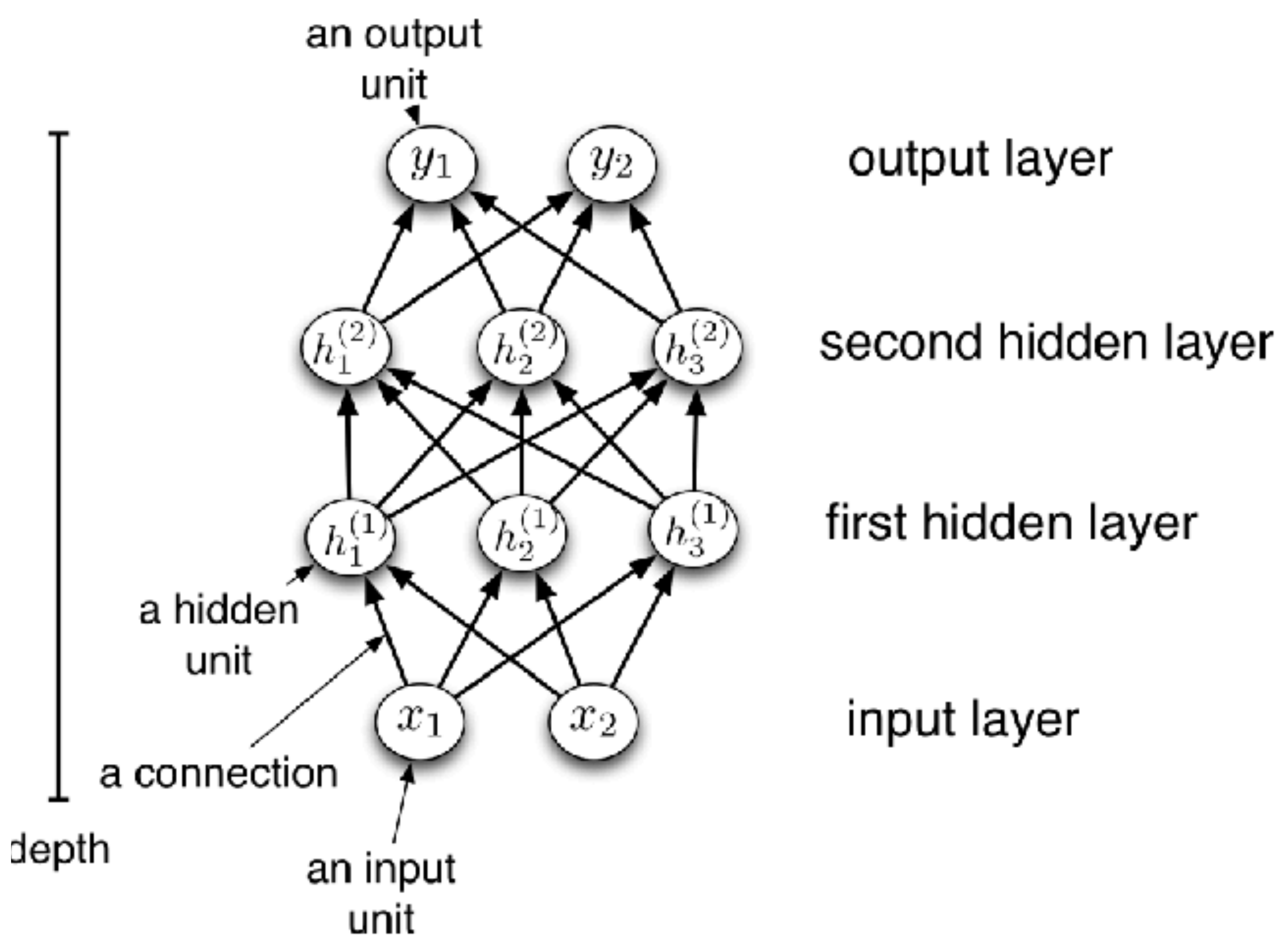


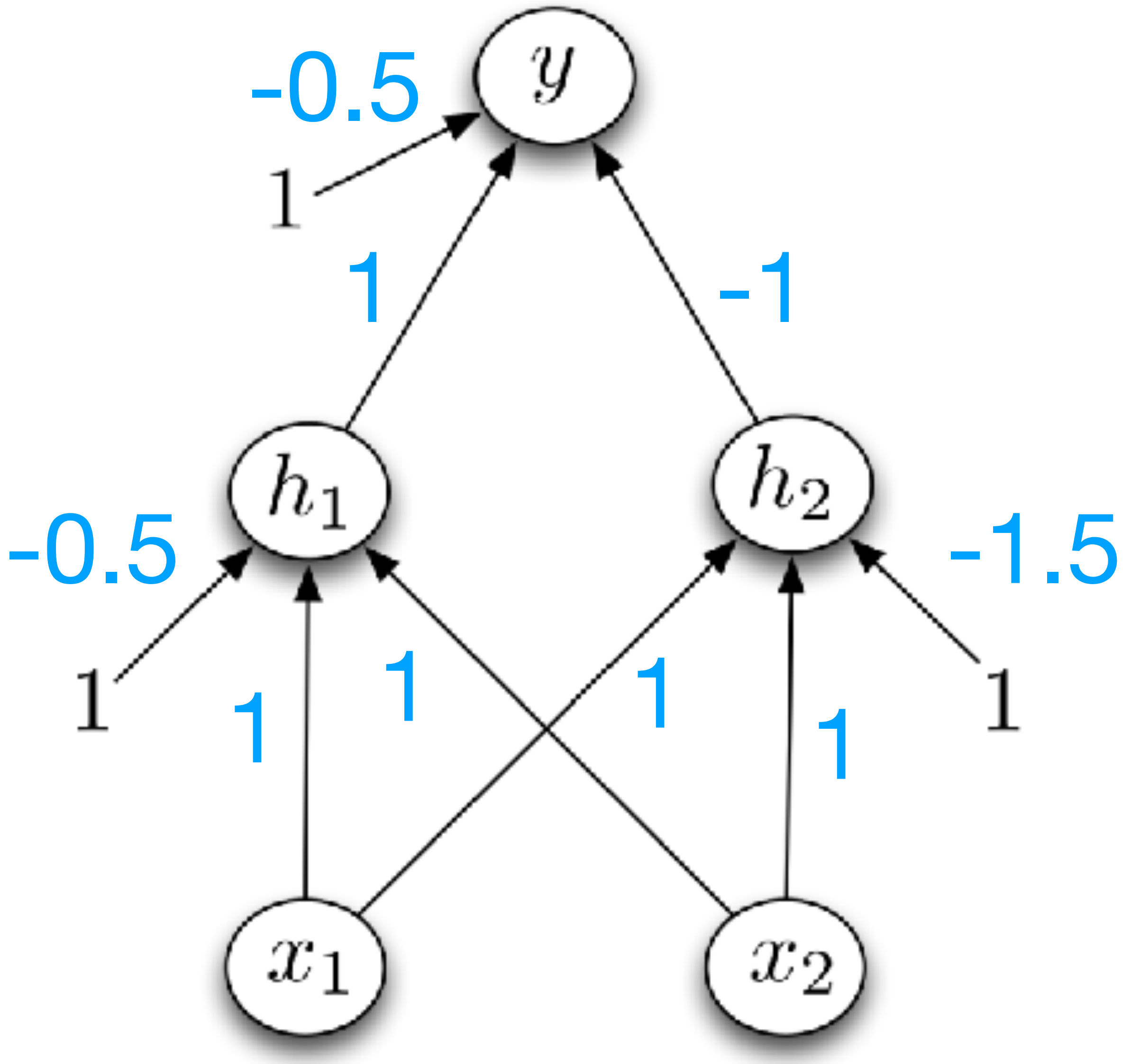


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$		
1	0			
0	1			

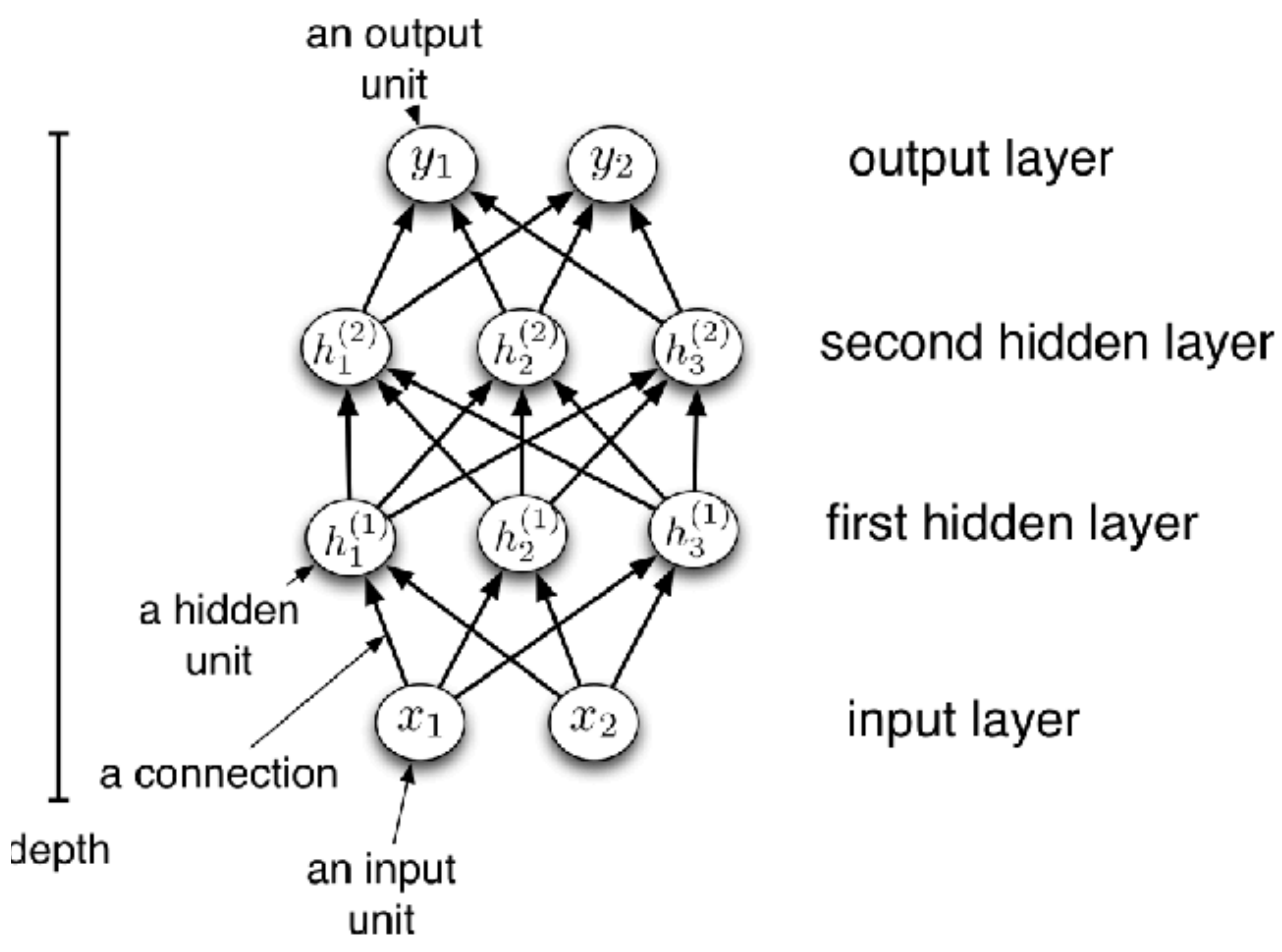


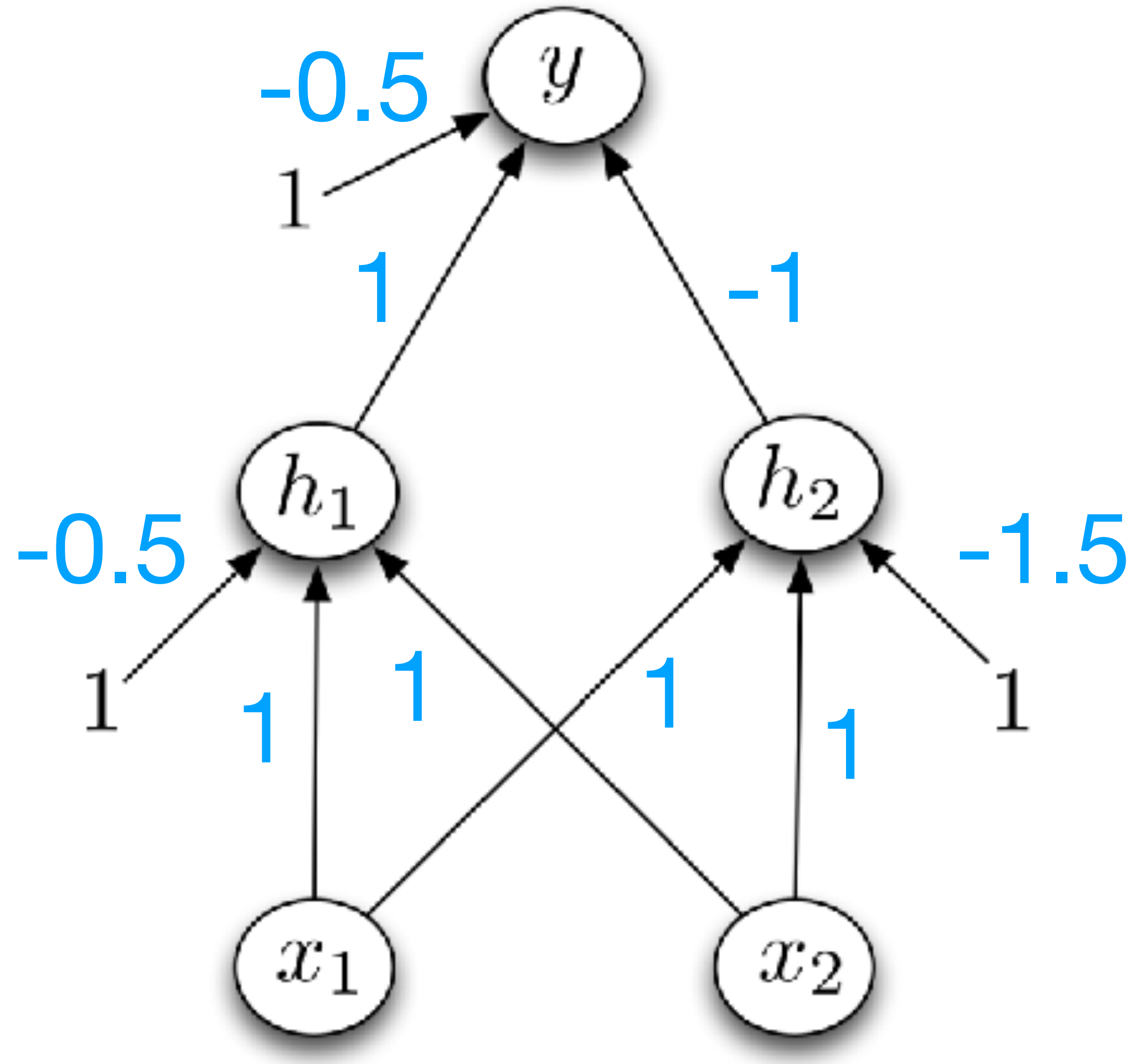


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	
1	0			
0	1			

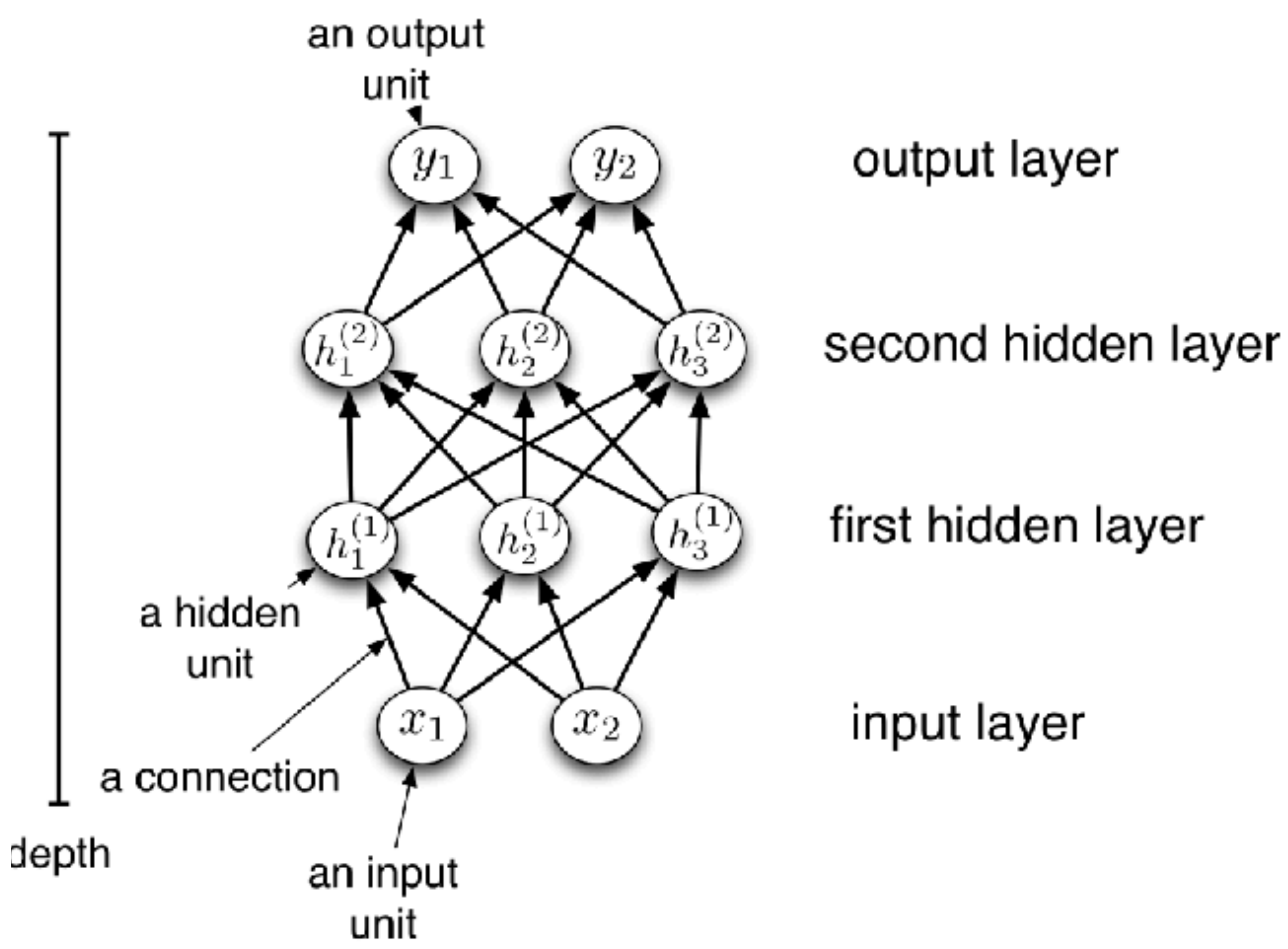


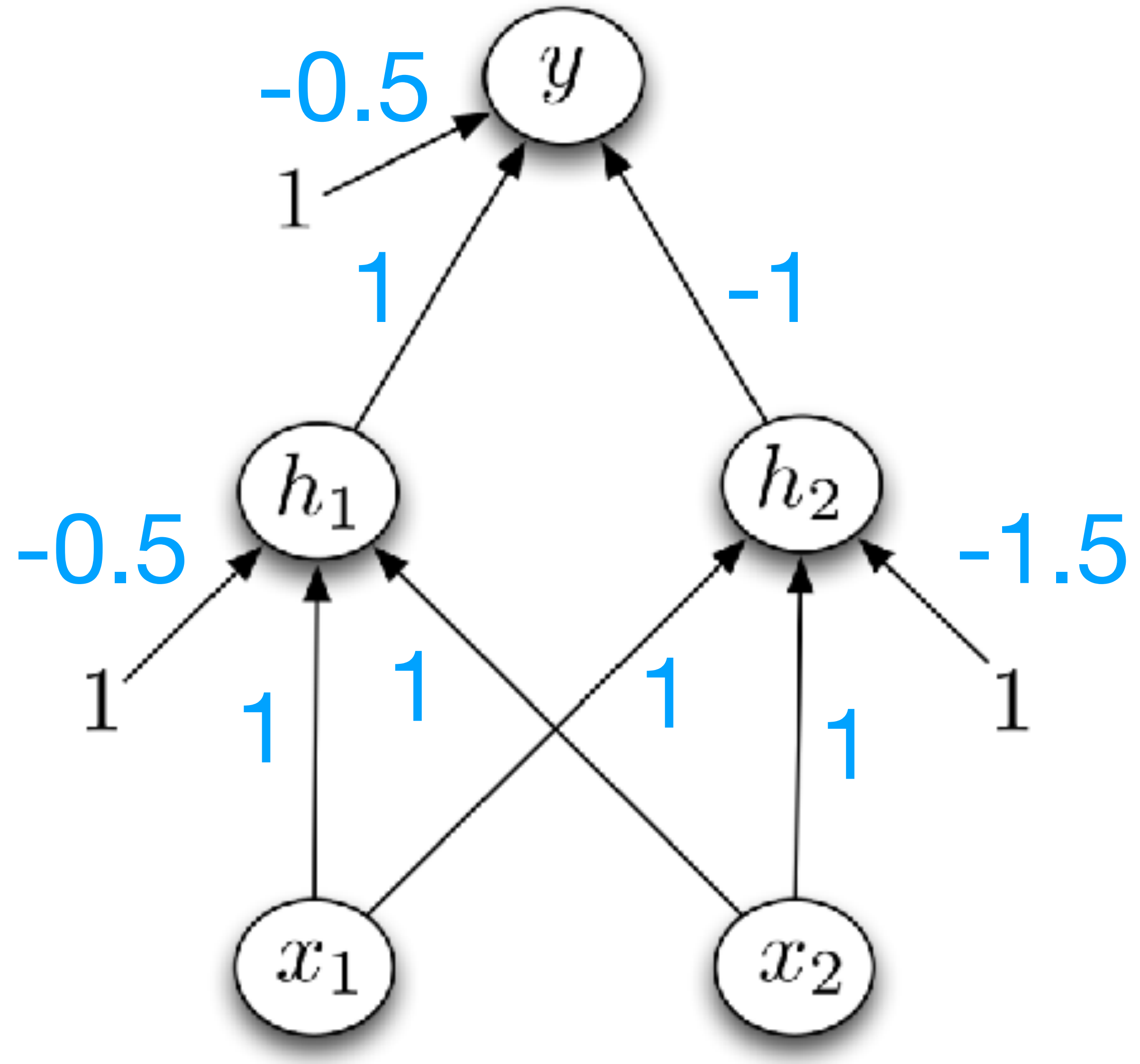


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0			
0	1			

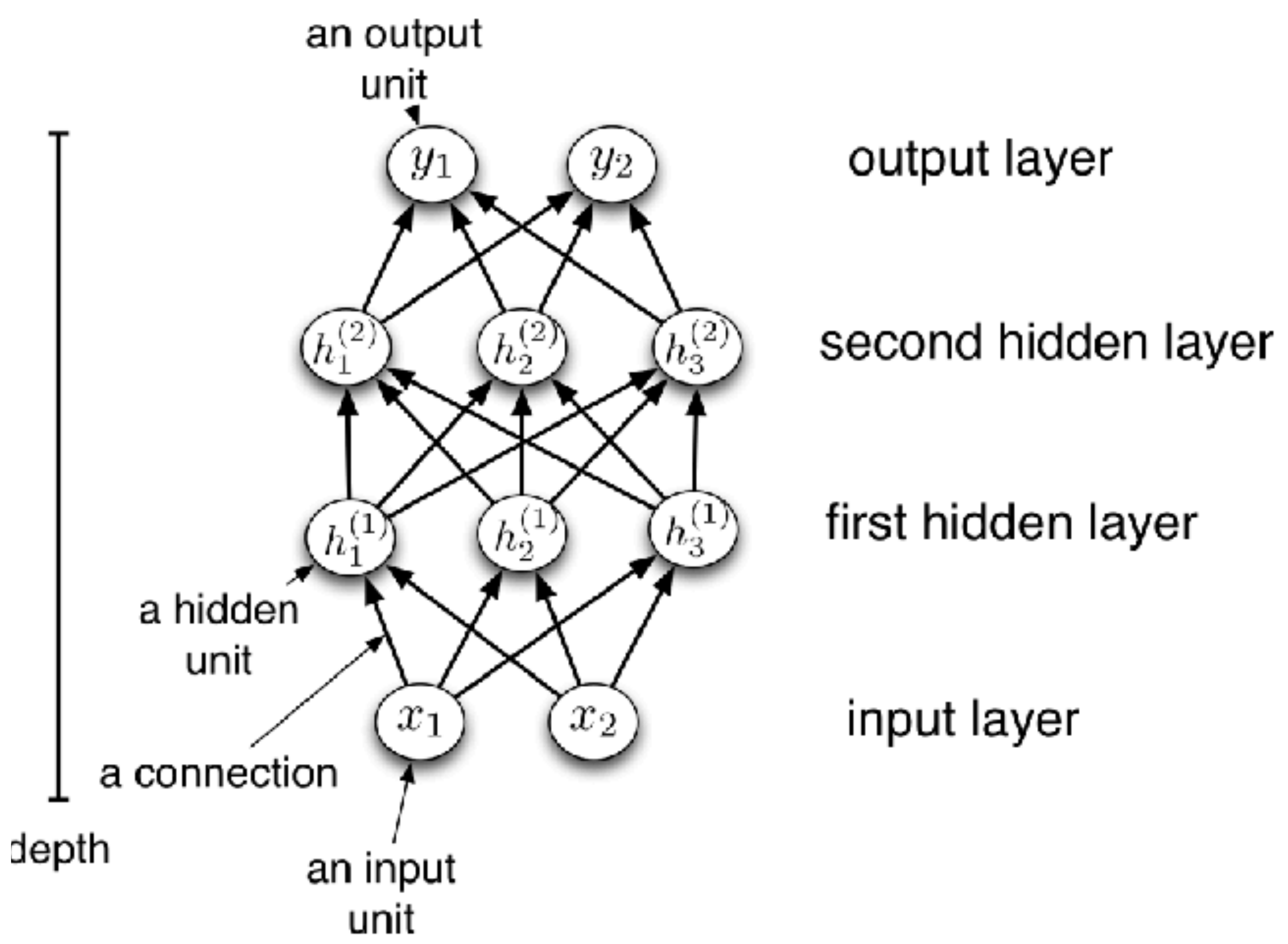


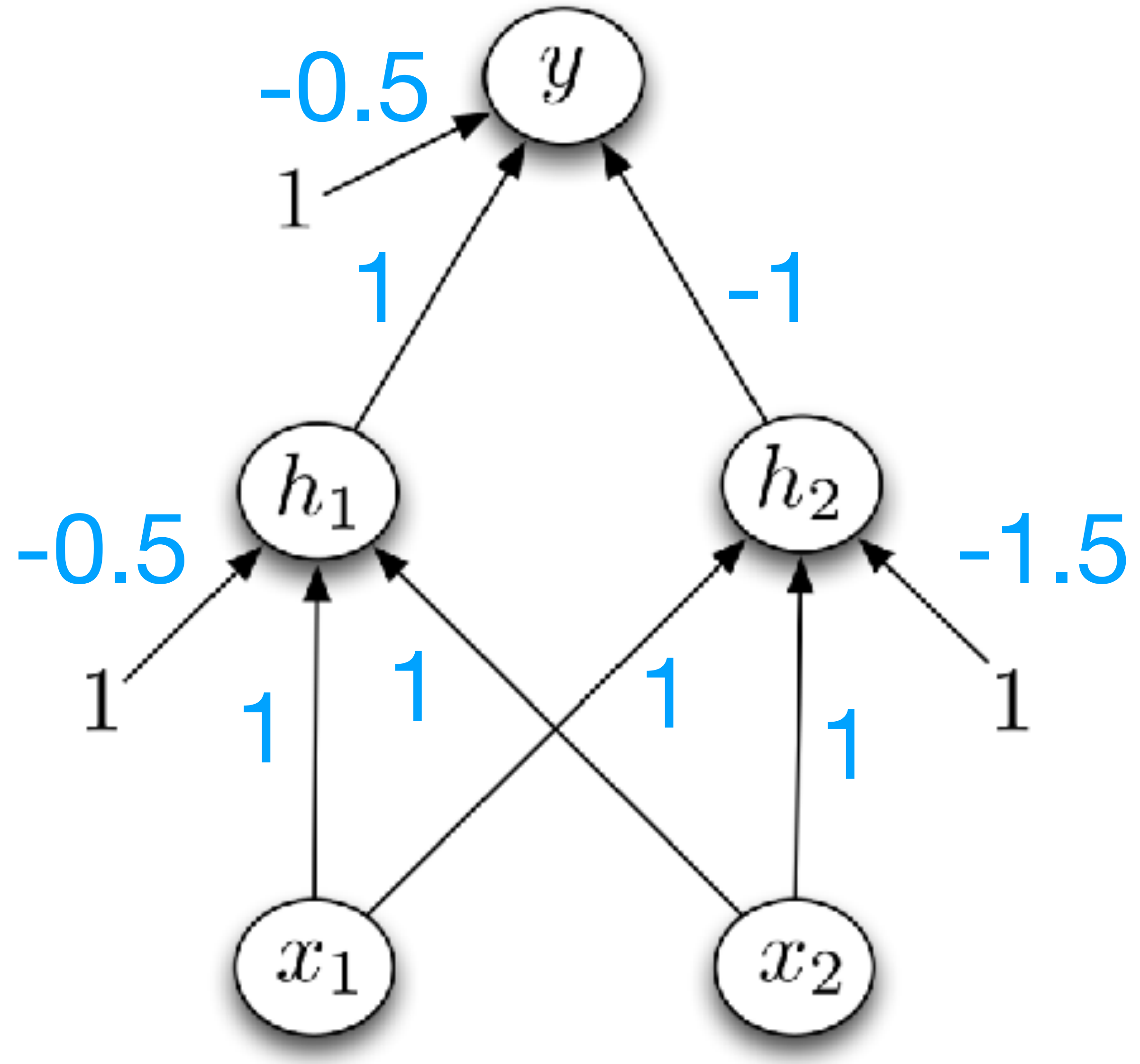


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0	$\sigma(.5) = 1$		
0	1			

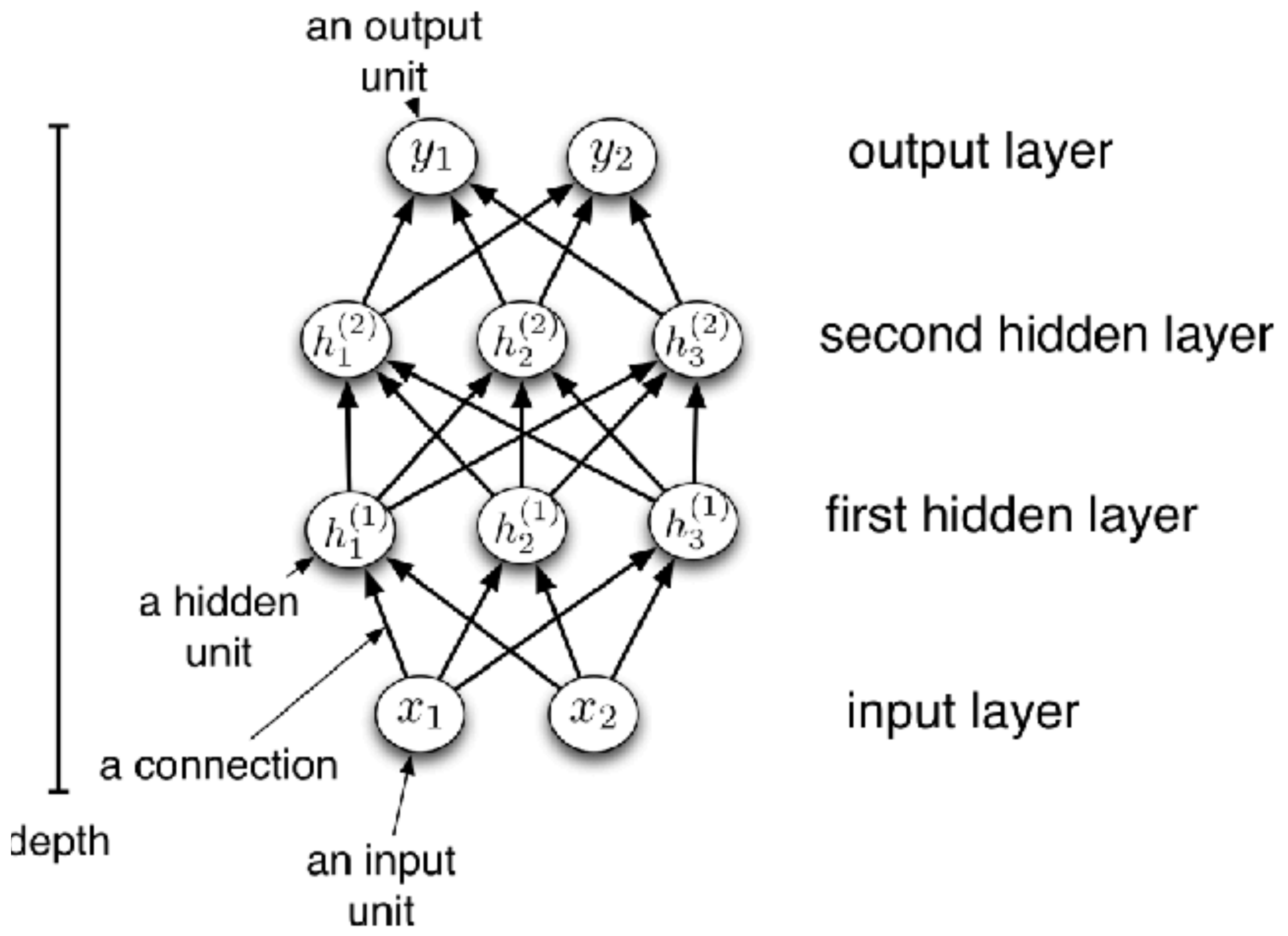


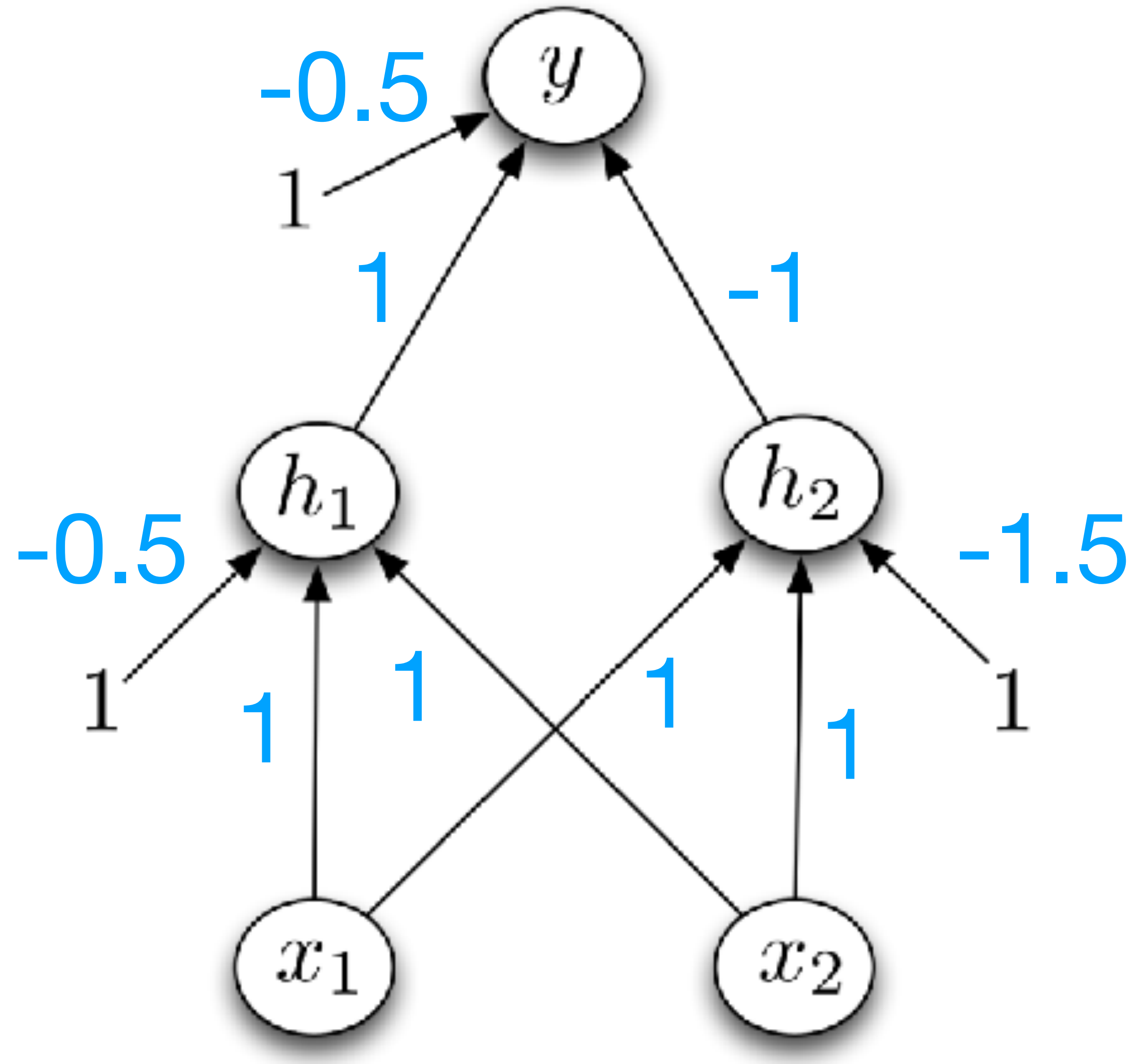


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	
0	1			

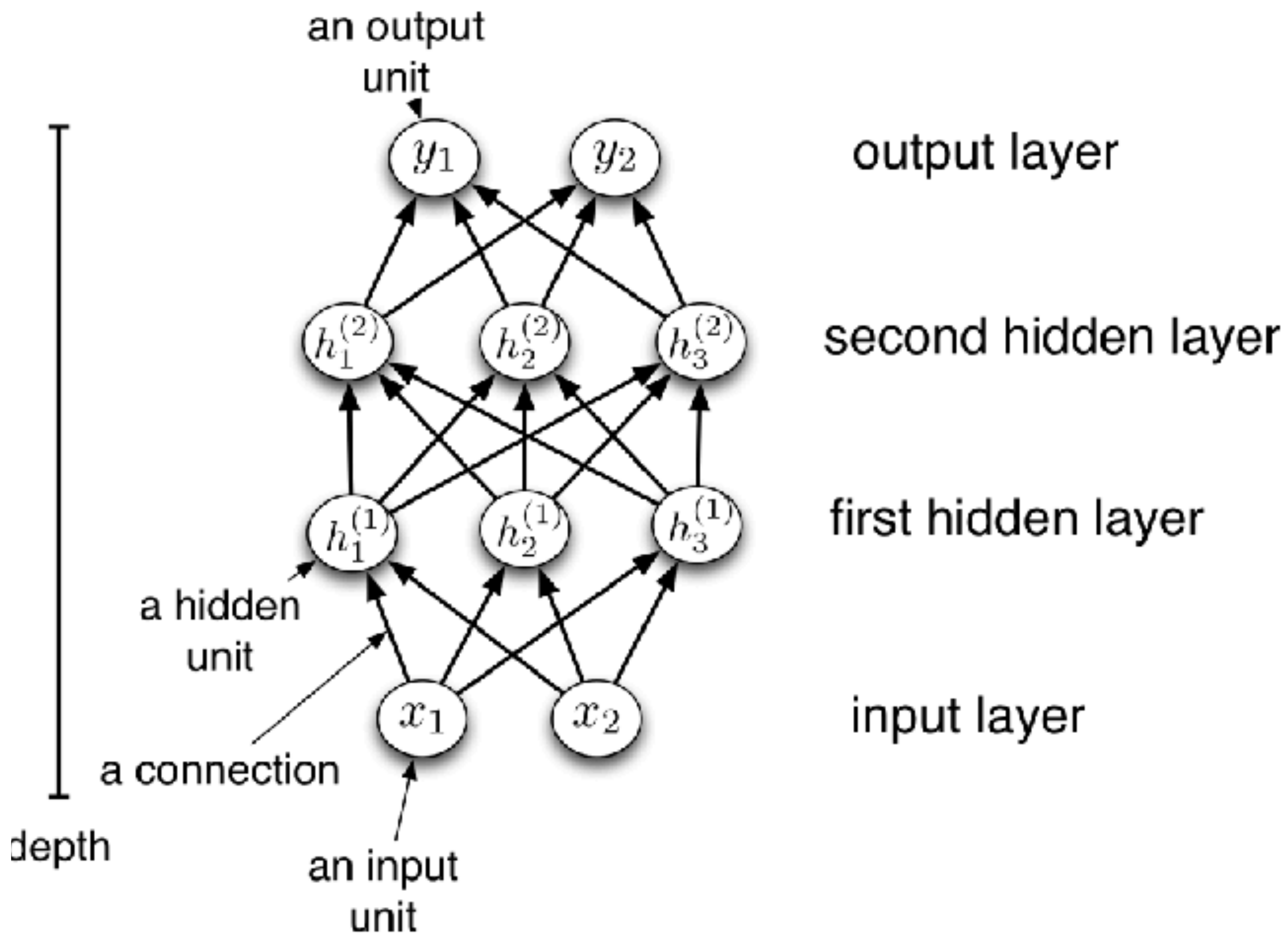


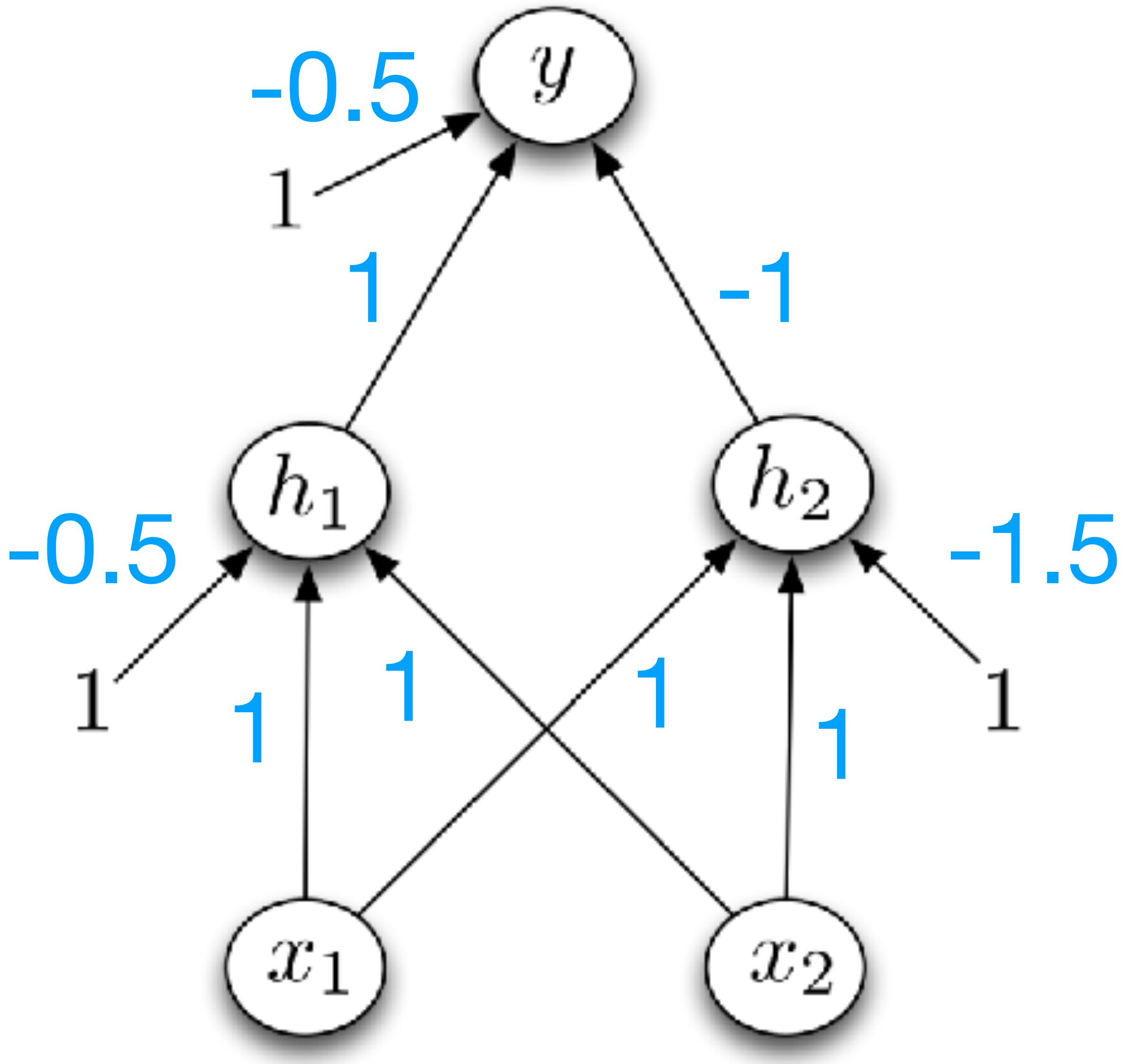


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1, h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	$\sigma(.5) = 1$
0	1			

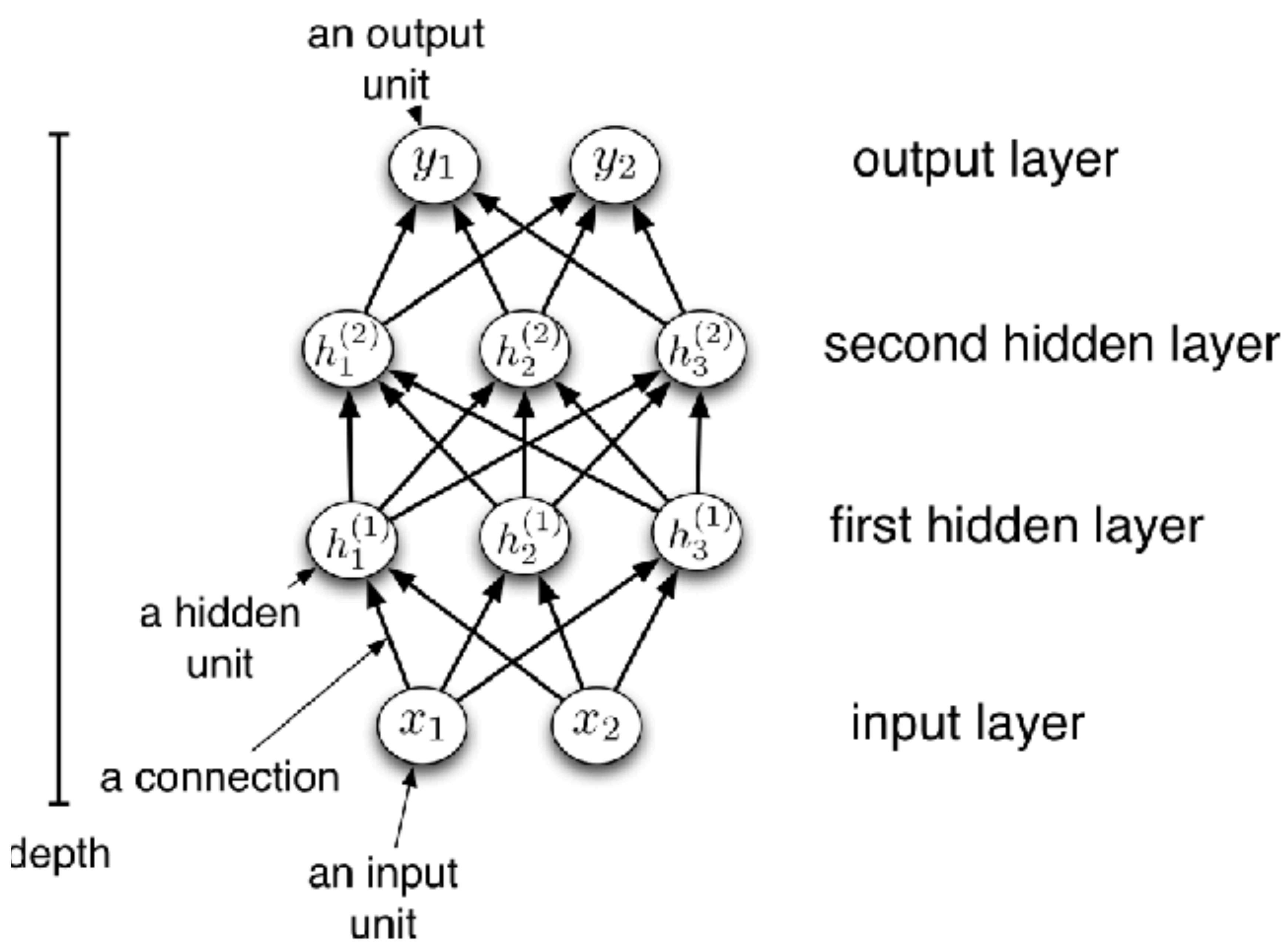


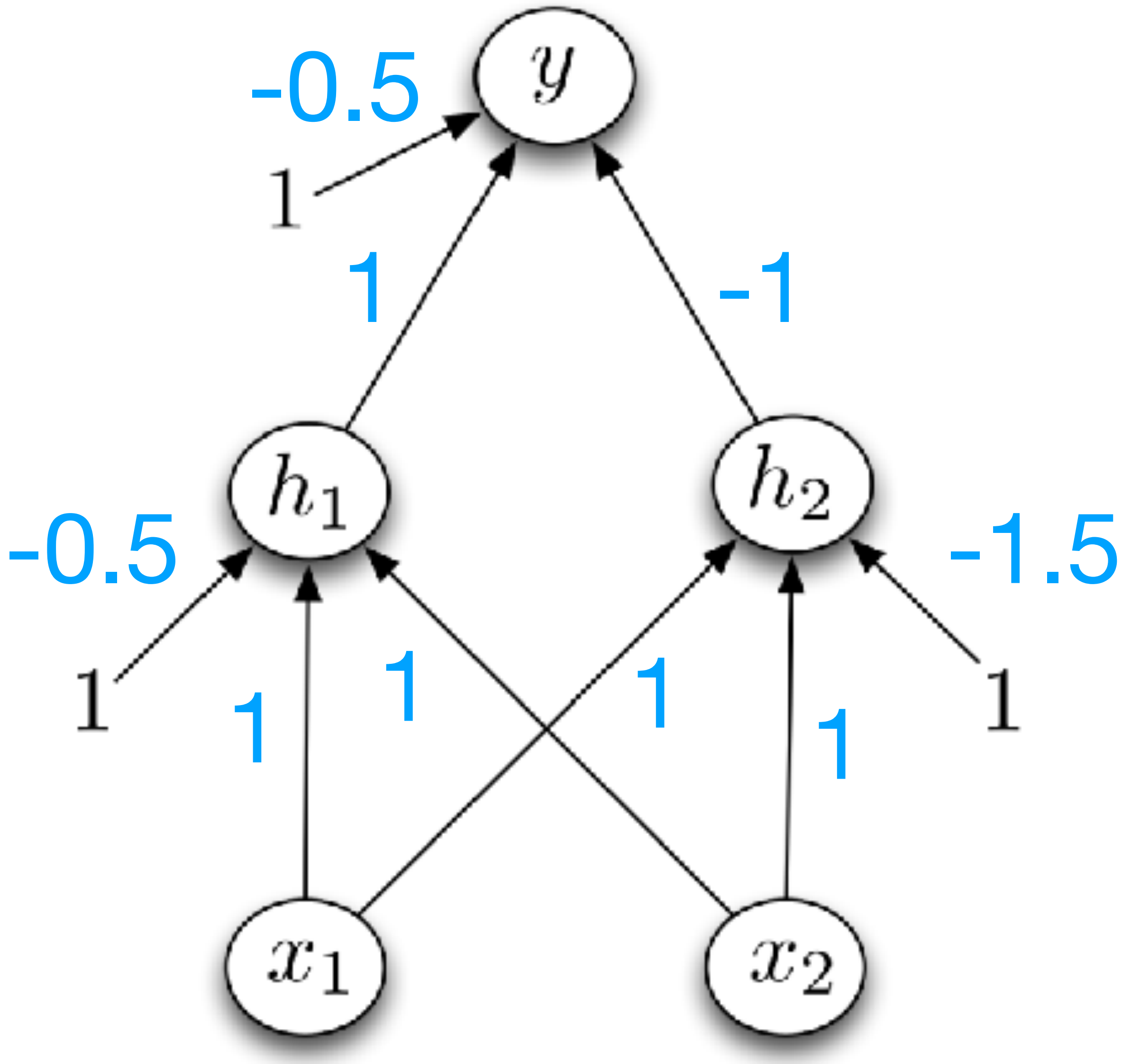


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	$\sigma(.5) = 1$
0	1	$\sigma(.5) = 1$		

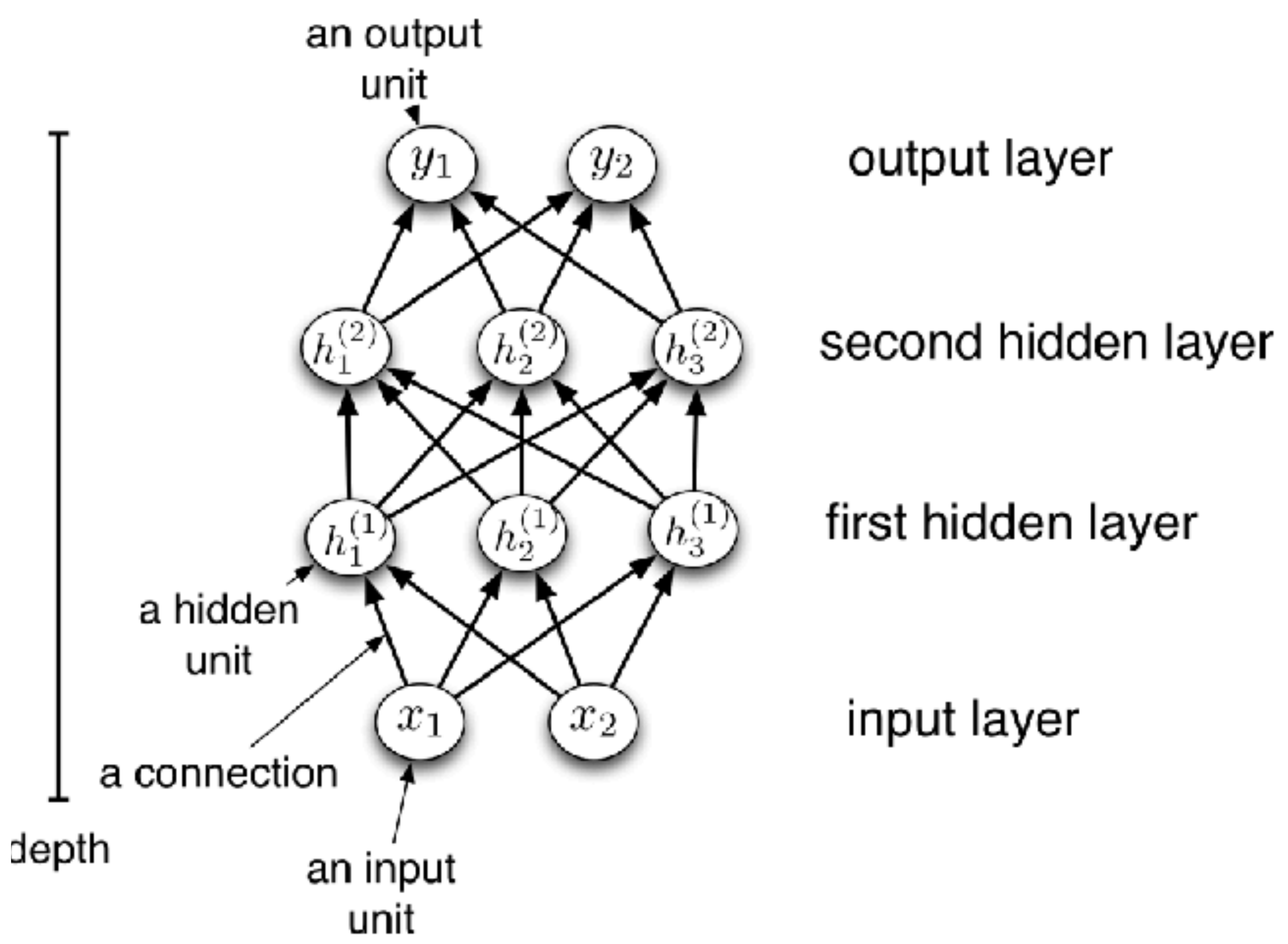


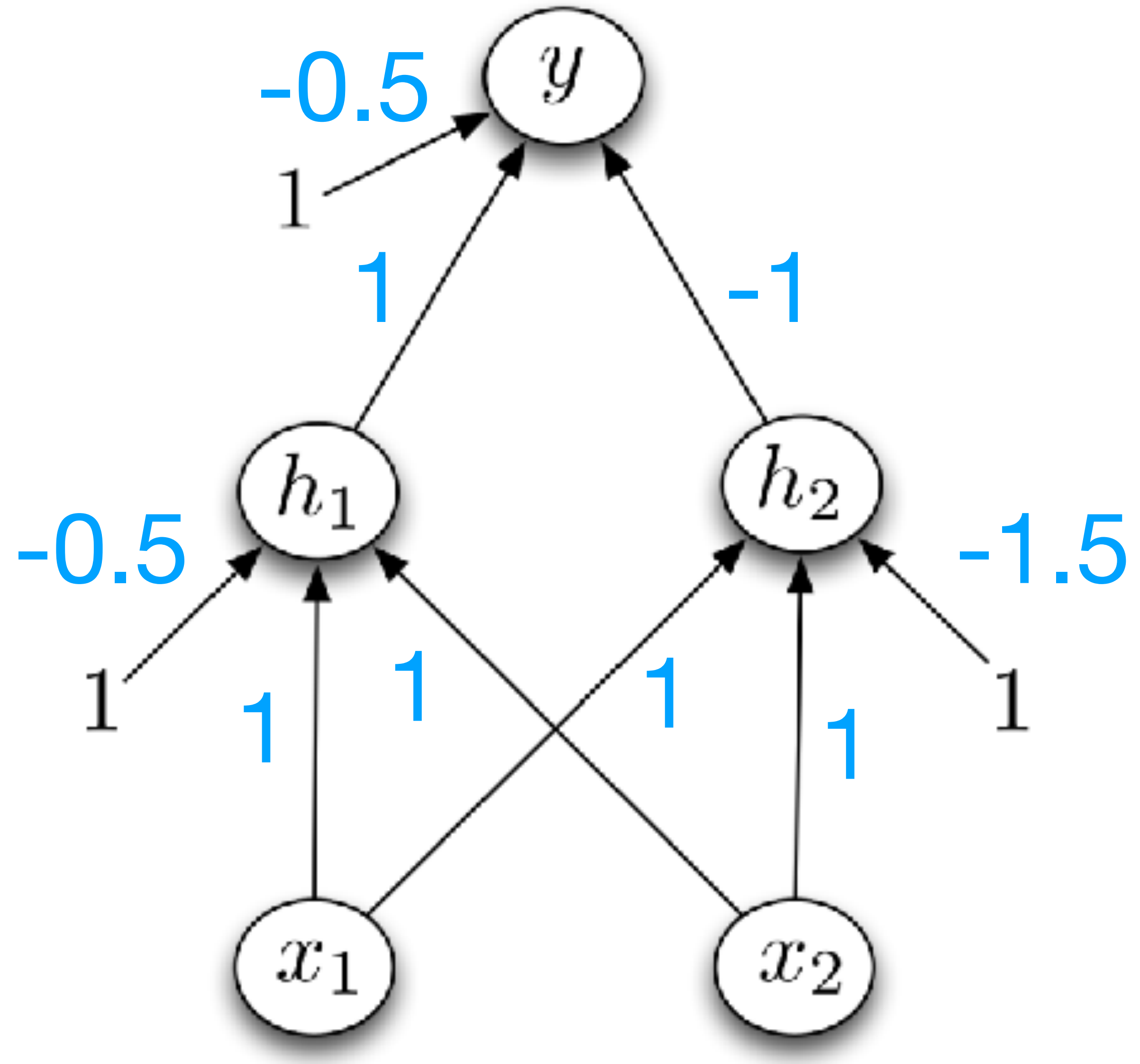


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1, h_2, and y when:

x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	$\sigma(.5) = 1$
0	1	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	

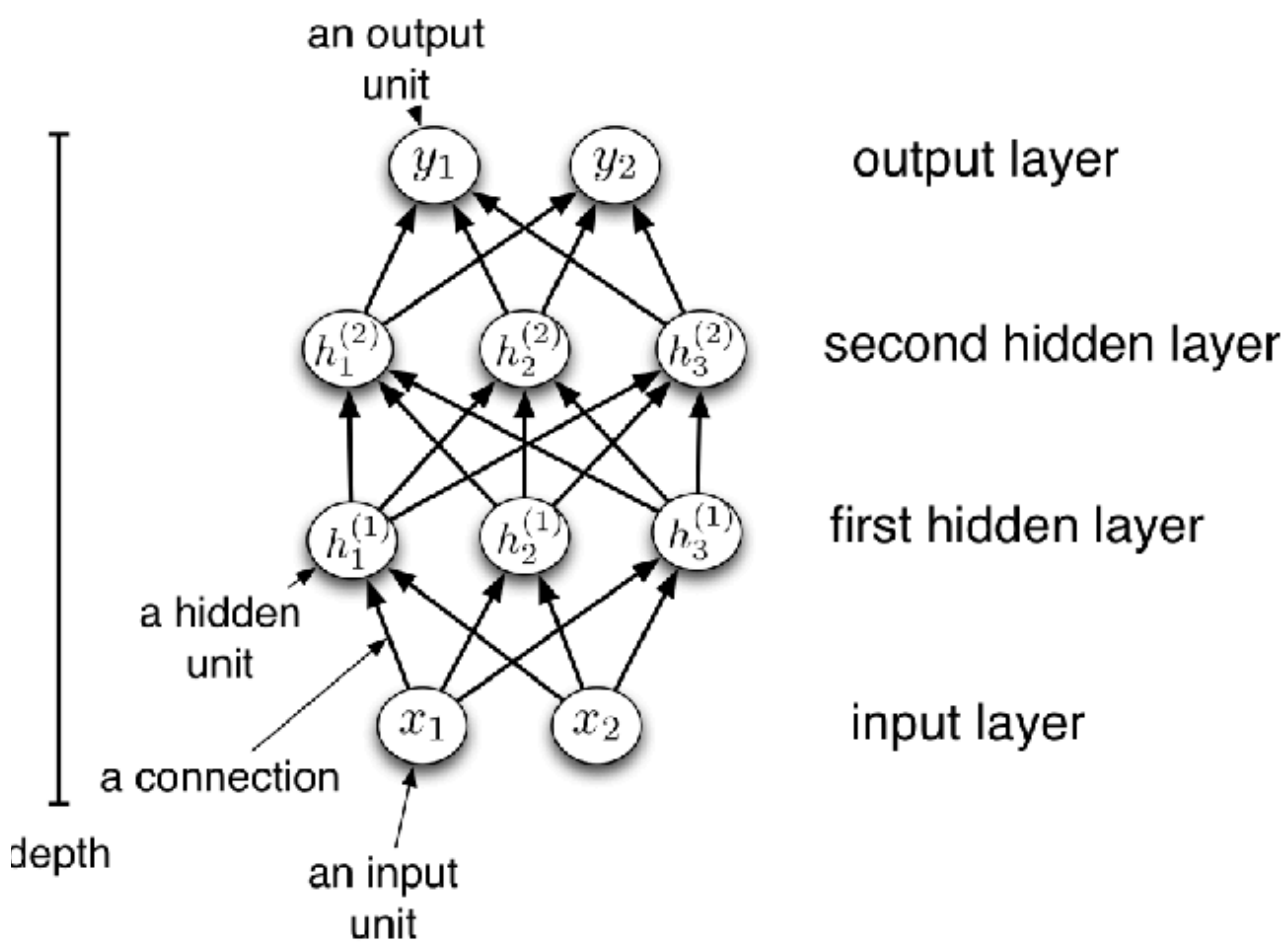


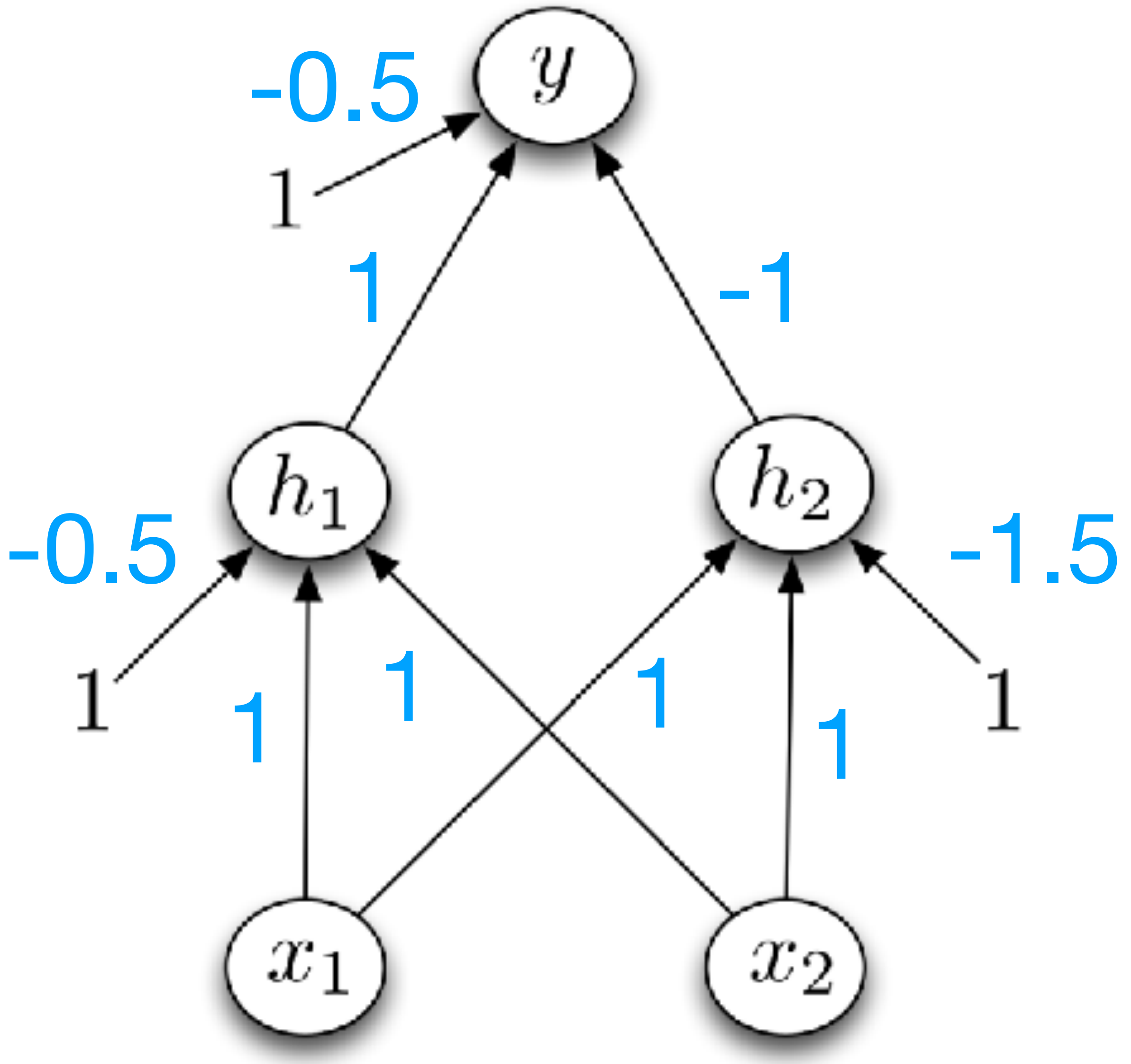


Multilayer Perceptron

- Rosenblatt introduced an MLP with 3 layers in 1962, but only the outer layer had learning connections
- First deep learning MLP by Ivakhenko & Lapa (1965), with stochastic gradient descent added in 1967 by Shun'ichi Amari
- MLPs are feedforward networks with multiple hidden layers, where we apply the same activation function at each layer $h_i = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ and $y = \sigma(\mathbf{w}^\top \mathbf{h} + b)$
- A single hidden layer allows us to solve XOR
- What are h_1,h_2, and y when:

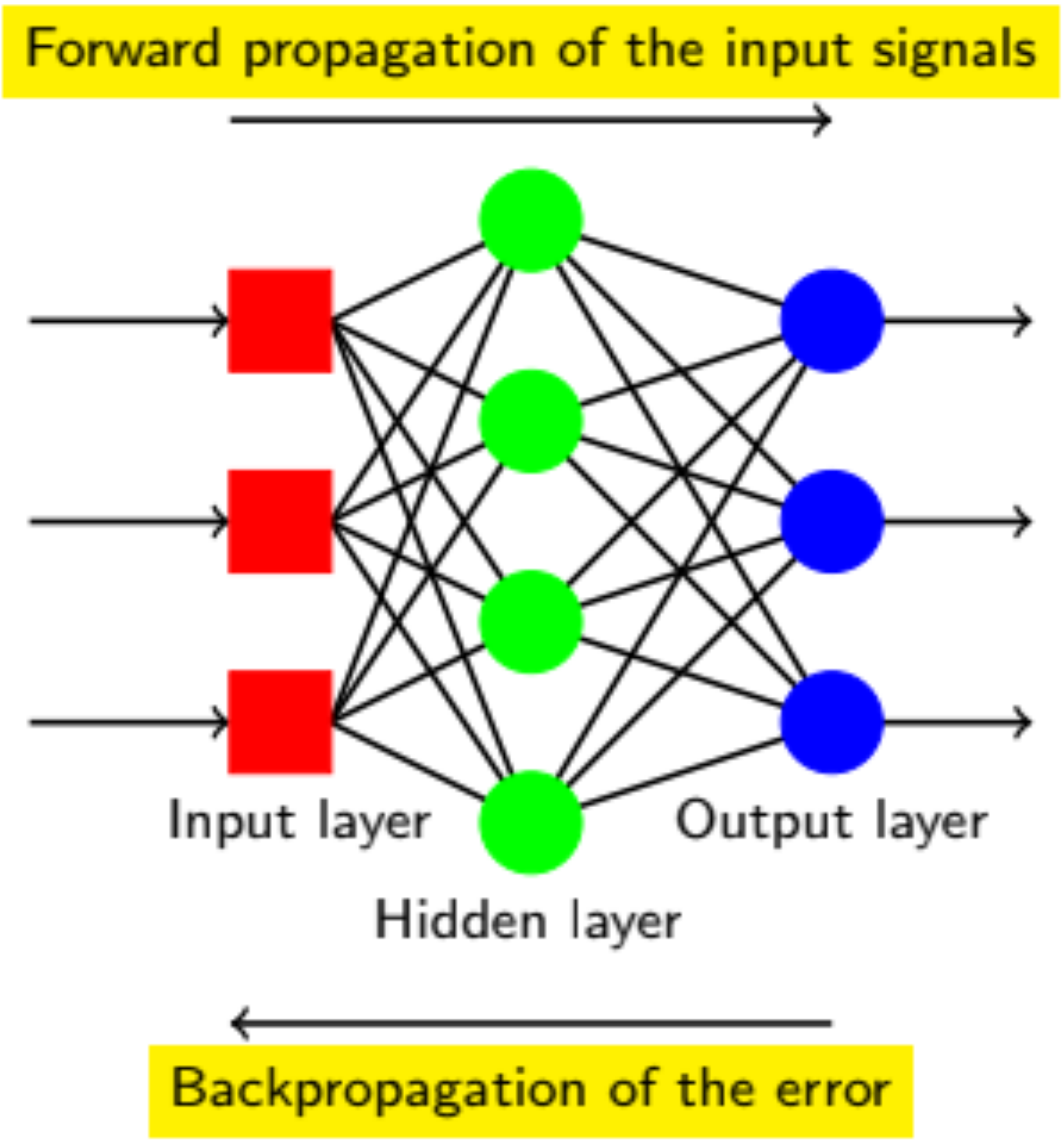
x_1	x_2	h_1	h_2	y
0	0	$\sigma(-.5) = 0$	$\sigma(-1.5) = 0$	$\sigma(-.5) = 0$
1	1	$\sigma(1.5) = 1$	$\sigma(.5) = 1$	$\sigma(-.5) = 0$
1	0	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	$\sigma(.5) = 1$
0	1	$\sigma(.5) = 1$	$\sigma(-.5) = 0$	$\sigma(.5) = 1$





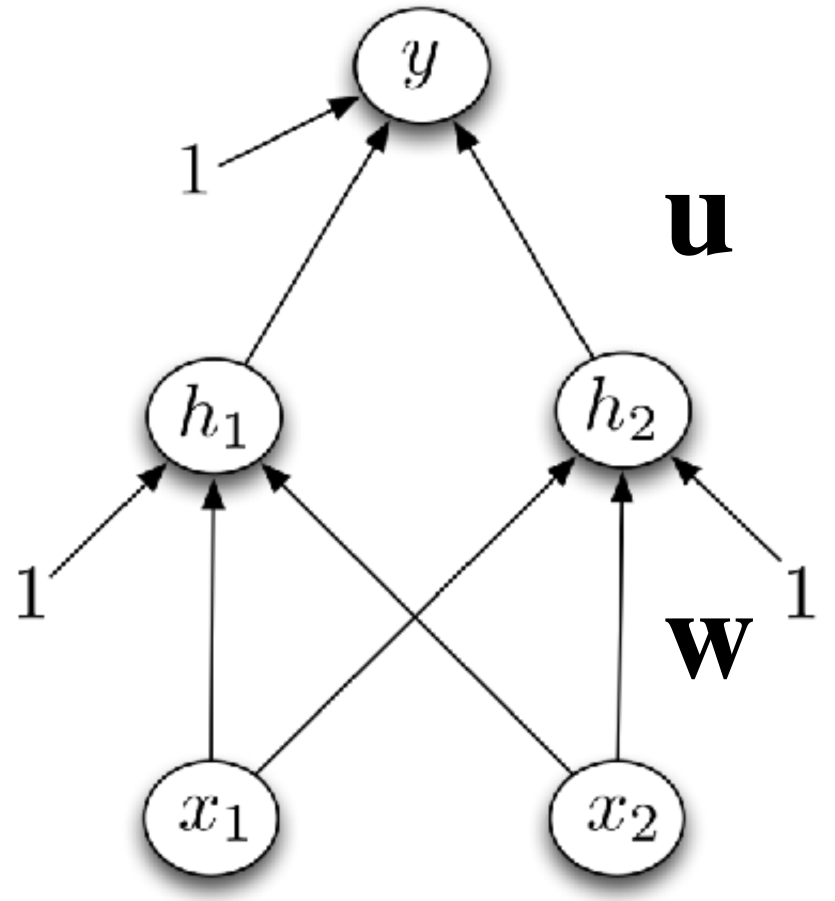
Backpropagation

- Rosenblatt (1962): Forward propagation of signals (for making predictions), and backward propagation of error (for updating weights)
- We can start with the familiar delta-rule weight update and mean-squared error loss
- Backpropogation takes advantage of the fact that the MLP is a function composed of several individual functions (at each layer): $y(x) = h(f(x))$
    - Thus, the loss is also composed of the loss across individual layers
    - This allows us to use the *chain rule* for derivatives: $$\frac{\partial \mathscr{L}}{\partial \mathbf{w}} = \frac{\partial \mathscr{L}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{w}}$$
    - We use the error to first update the $\mathbf{u}$ weights, and then update $\mathbf{w}$ weights w.r.t. how they change $\mathbf{u}$
- For further reading, see Grosse & Ba (CSC421)

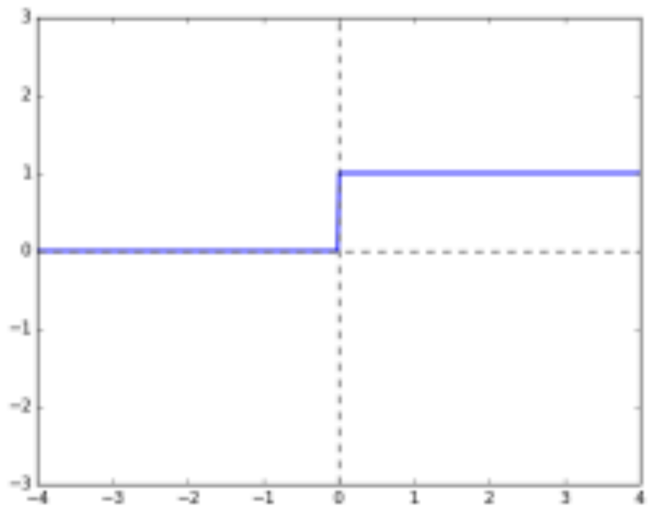


$$w \leftarrow w - \alpha \frac{\partial \mathscr{L}}{\partial w}$$

$$\mathscr{L} = \frac{1}{2} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

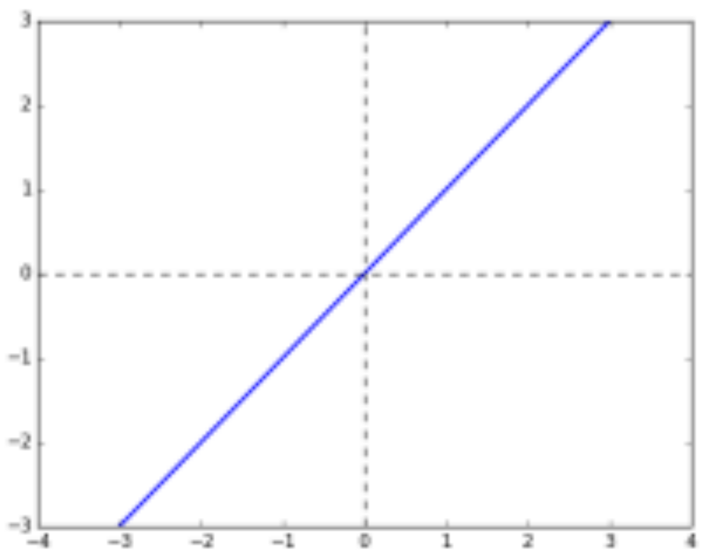


Other activation functions



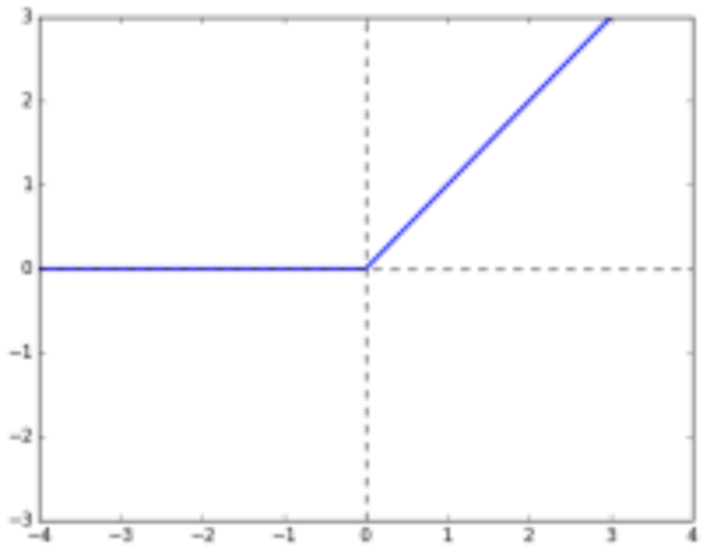
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



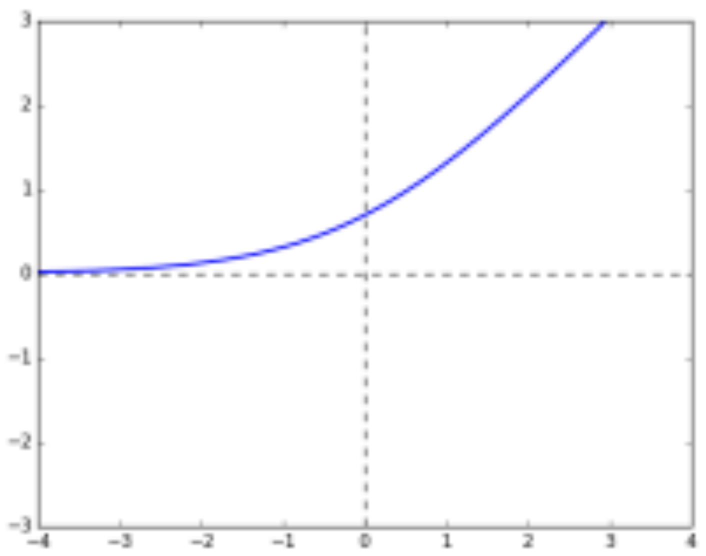
Linear

$$y = z$$



Rectified Linear Unit (ReLU)

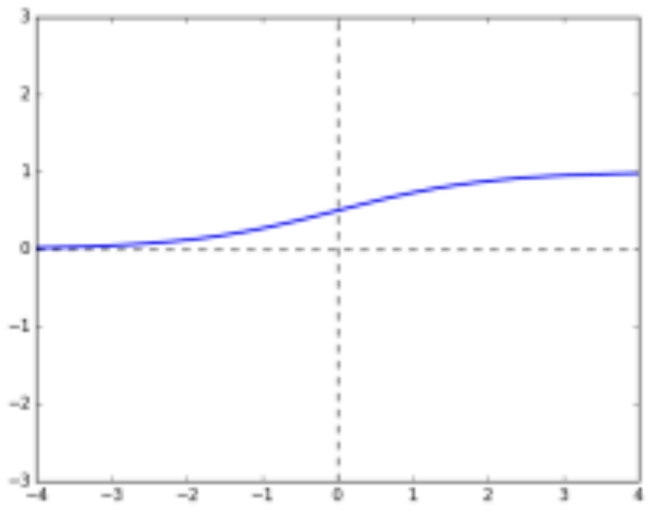
$$y = \max(0, z)$$



Soft ReLU

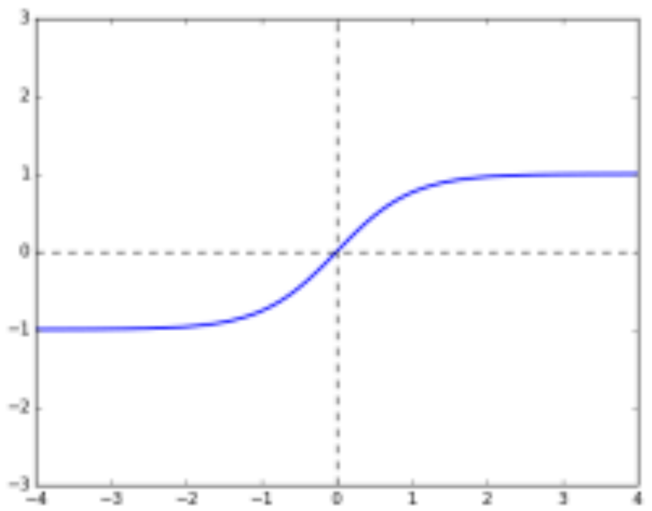
$$y = \log 1 + e^z$$

Universal approximation theorem (Cybenko, 1989): An ANN with a hidden layer with a finite number of units and mild assumptions on the activation function can approximate any function arbitrarily well



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

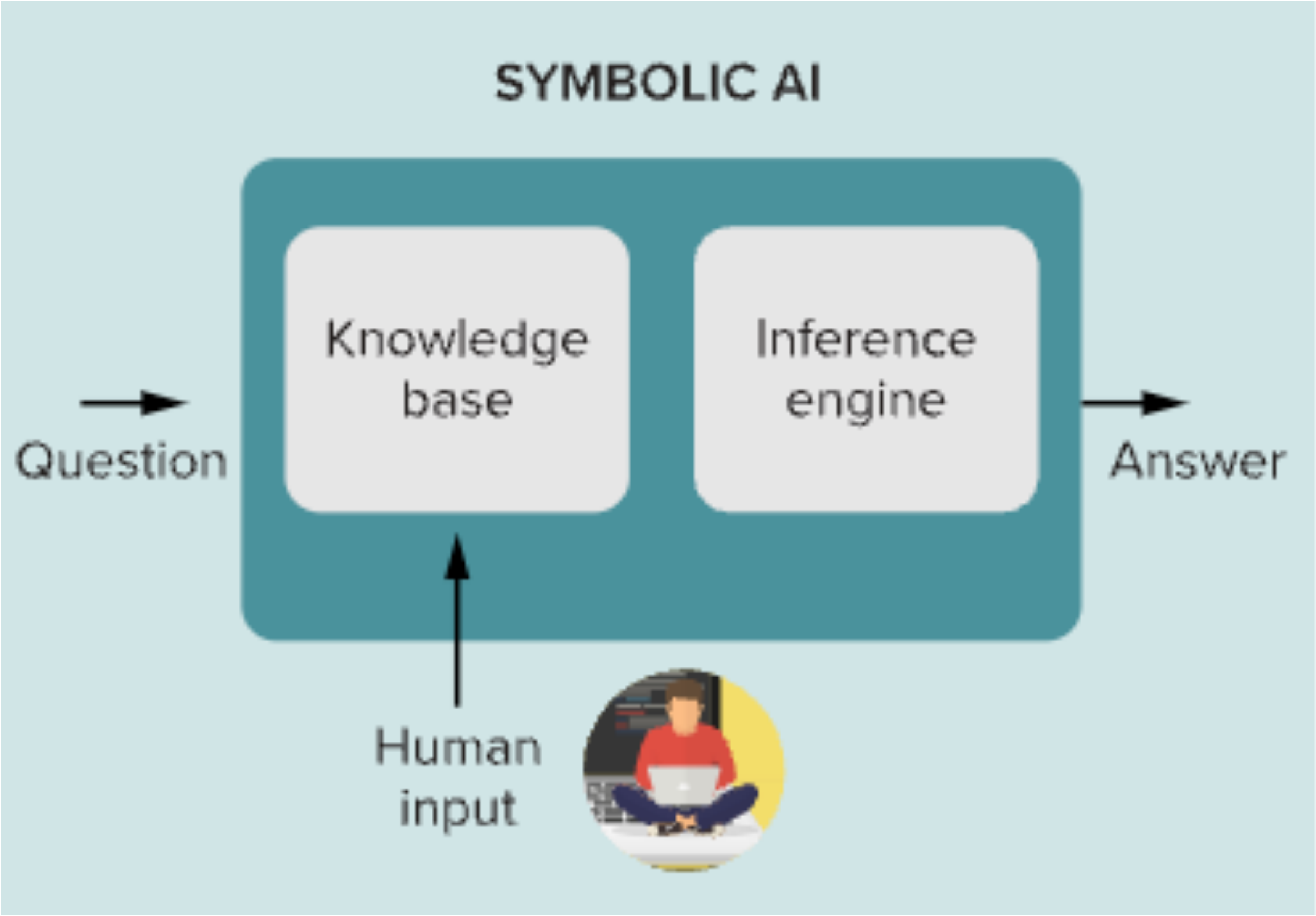
Connectionism: Summary

- **Perceptrons** can learn a number of logical operations, but fail at problems that are not linearly separable (e.g, XOR)
- **Rosenblatt's** learning rule is guaranteed to converge (for linearly separable problems), but is brittle with noisy training data
  - ADALINE offers a more robust learning rule, which is equivalent to stochastic gradient descent
- **Multilayer Perceptrons** are capable of solving XOR and other non-linearly separable problems
- **Backpropogation** is necessary for learning in MLPs, by passing the gradient across multiple layers using the chain rule

General Principles

- Incrementally improve predictions by reducing error
  - The unit of learning is the magnitude of the prediction error (Delta-rule)
    - Rescorla-Wagner model and ADALINE
    - But more generally, stochastic gradient descent, backpropogation, and all modern RL use this principle
- Incremental learning is not always guaranteed to succeed
  - Behavioral shaping and reinforcement schedules help guide learning towards desired outcomes
  - Single layer perceptrons are limited in which types of problems they can solve
    - Adding more layers helps, but it took a long time to develop learning rules
  - Gradient descent can get stuck in local optima
- What other principles have you picked up?

Next week we will look at what happened during the AI winter and explore the limits of stimulus-response learning



Symbolic AI

- What happened during the AI winter?
- Intelligence as manipulating symbols through rules and logical operations
- Learning as search

Cognitive Maps

- From Stimulus-Response learning to Stimulus-Stimulus learning
- Constructing a mental representation of the environment
- Neurological evidence for cognitive maps in the brain