Queen's University | Faculty of Arts and Science

Classroom **PROCEDURES**

- Get vaccinated
- Provide vaccination proof
- Do the daily COVID screen
- Don't attend when ill
- Wear a mask
- Leave room promptly
- Wash hands frequently
- Don't consume drinks/food

QUartsci.com/Fall2021

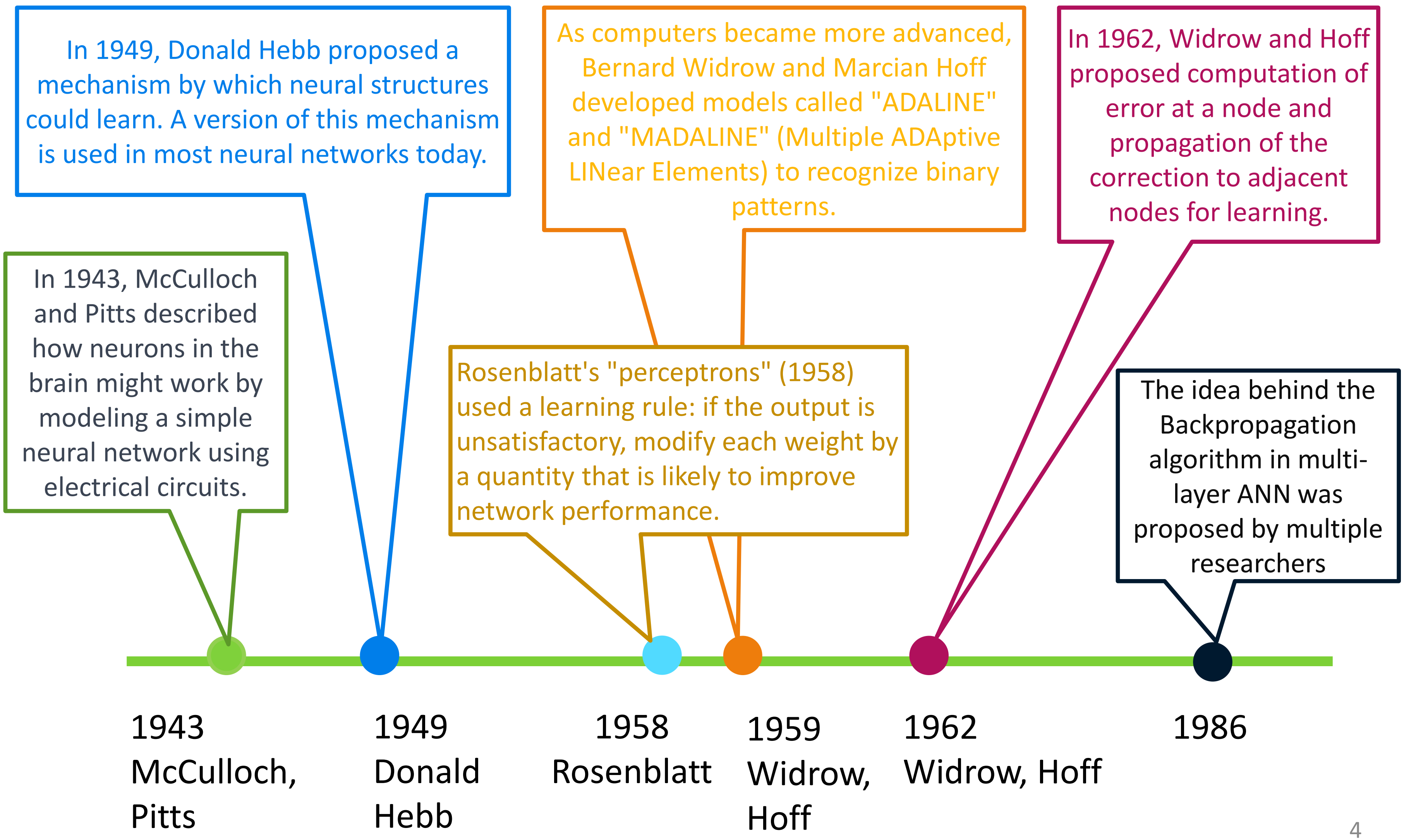
CISC452/CMPE452/COGS400
Perceptron

Farhana Zulkernine

McCulloch and Pitts's Neurons

- McCulloch and Pitts (1943) defined the first mathematical model of a single neuron.
- Early models of ANNs did not demonstrate learning.
- Weights were static and so were the connections.
- Had single layer that could not implement XOR.

History & Evolution of ANN Models

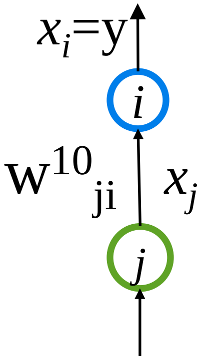


Introduce Learning

- Hebb's learning rule (1949): For each input pattern, increase connection weight between nodes *i* and *j* if both nodes are simultaneously ON or OFF.
- Activation of *j* always causes an activation of *i* where w_{ji} is the weight associated with connection from *j* to *i* and x_i and x_j are inputs to *i* and *j* respectively.
- The strength of connections between neurons eventually comes to represent the correlations between their outputs, e.g.,

$$\Delta w_{ji} = c \, . \, x_i x_j$$

where *c* is a some small constant.



Perceptrons

- Rosenblatt's "perceptrons" (1958) used the following learning rule
  - If the output is unsatisfactory, modify each weight by a quantity that is likely to improve network performance.
- Also introduced the idea of supervised learning.
  - Correct output was known and was used to modify weights to generate better output, and thereby, TRAIN the network.

More Learning Algorithms…

- Widrow and Hoff's learning rule (1960, 1962) was also based on *gradient descent*.
- Then back-propagation algorithms were proposed for training MULTI-LAYER networks.

Perceptron

- Frank Rosenblatt proposed the perceptron learning rule in 1950's based on the idea that the operation of a neuron and its learning could be modeled mathematically, and used as a form of computation.

Perceptron

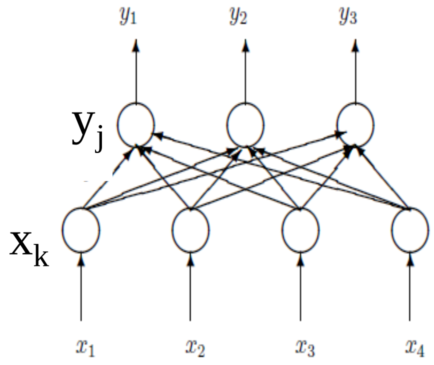
- A Perceptron Network is designed to learn the relationship between an input and output data.
- Input/Desired-output examples – supervised learning: $\{(X_1, D_1), \ldots, (X_p, D_p)\}$

  Vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, $D_i = (d_{i1}, d_{i2}, \ldots, d_{im})$

  $X_i \in \{-1,+1\}^n$ or $[0,1]^n$ or R^n

  $D_i \in \{-1,+1\}^m$ or $\{0,1\}^m$
- For a data point X_i and output node *j*

  $y_j = f(net_j) = f(\sum_{k=1..n} x_{ik} w_{jk})$ if $net >= \theta$,
  and 0 otherwise



- $w_{jk} \in R$
- $(d_j - y_j)$ is the error, θ is threshold or bias

Perceptron for Prediction

- Train the perceptron using **input** and **desired output** vectors.
- Example: Given X_1, we like the perceptron to produce D_1 for output where d_1 is known.
- Predicting two output features.

$$\mathbf{X_1 = (10, 3150, 0.25)} \qquad \mathbf{D_1 = (1, 0)}$$

Acreage of property (x_{13})

Square feet of house (x_{12})

Age of house in years (x_{11})

House will sell within 6 months (1=yes) (d_{12})

Sale is over $300K (1=yes) ($d_{11}$)

Features and Functionality

- Two layer network
- Applies **feedforward processing – all connections go to the next layer**
- **Initially w_i are assigned random values** which results in poor initial performance (high error)
- To improve performance, network is **trained to adjust the weight values** → network **learns**
  - A **Learning Rule** is a strategy by which input/output pairs are used to *incrementally change the weights to gradually improve the performance* of the network

Adjusting both weight and bias

$a = \sum_{i=1}^{n} w_i x_i$

If (a >= θ) then output 1
else output 0

$a = \sum_{i=0}^{n} w_i x_i$

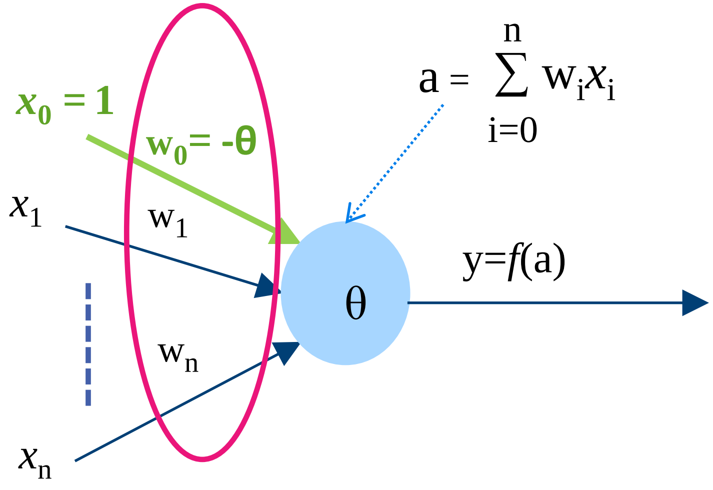
If (a >= 0) then output 1
else output 0

$$\sum_{i=1}^{n} x_i w_i - \theta = 0 \Rightarrow \sum_{i=1}^{n} x_i w_i - x_0 w_0 = 0, \quad \text{with } x_0 = 1,\ w_0 = -\theta$$

Now weight $w_0 = -\theta$ can be learned like the other weights

$$\sum_{i=0}^{n} x_i w_i = 0$$

Allows each neuron to set its own threshold θ.



Perceptron Learning

- Two types of learning:

1. **Simple Feedback learning**

   Uses the correct/incorrect feedback and info about $(y>=d)$ or $(y<d)$ to change weights.

2. **Error Correction Learning**

   Uses an error measure to adapt the weight vector.

Simple Feedback Learning

If $y=1$ and $d=0$ $(y > d)$:

$$w_{ji} \leftarrow w_{ji} - cx_i$$

Use input value in calculation because if input value is high, error will be high and vice versa)

where $(i = 1,..,n)$ and c is a small learning rate

If $y=0$ and $d=1$ $(y < d)$:

$$w_{ji} \leftarrow w_{ji} + cx_i$$

where $(i = 1,..,n)$ and c is a small learning rate

Plotting the line

- For 2-D space, a neuron will represent a straight line

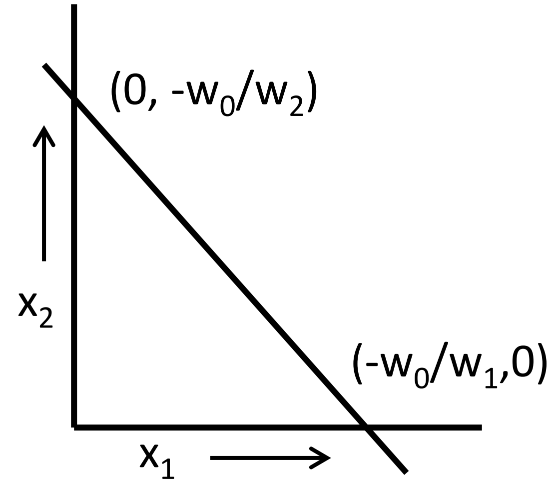
$$w_0+w_1x_1+w_2x_2= 0$$

- Representing it as y=mx+c, (and $y = x_2$)

$$x_2 =(-w_1/w_2)x_1 - w_0/w_2$$

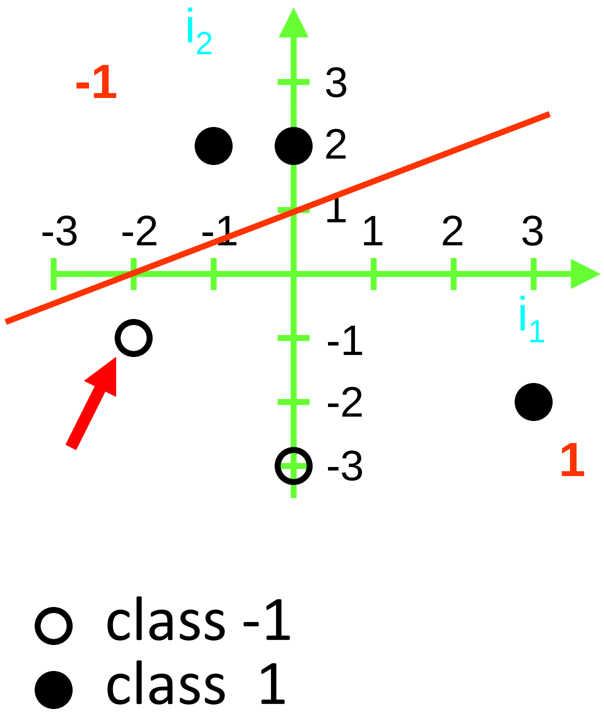
Slope Intercept

- On x_1 axis, x_2=0 and $x_1 = -w_0/w_1$
- On x_2 axis, x_1=0 and $x_2 = -w_0/w_2$
- So, given *w*, we can plot the line.



Perceptron Learning Example

We would like our perceptron to correctly classify the five 2-dimensional data points below.



Let the random initial weight vector $\mathbf{w^0} = (w_0, w_1, w_2) = (2, 1, -2)$.

So, the class separator line or ANN intersects the axes at

$[(-w_0/w_1, 0)$ and $(0, -w_0/w_2)]$

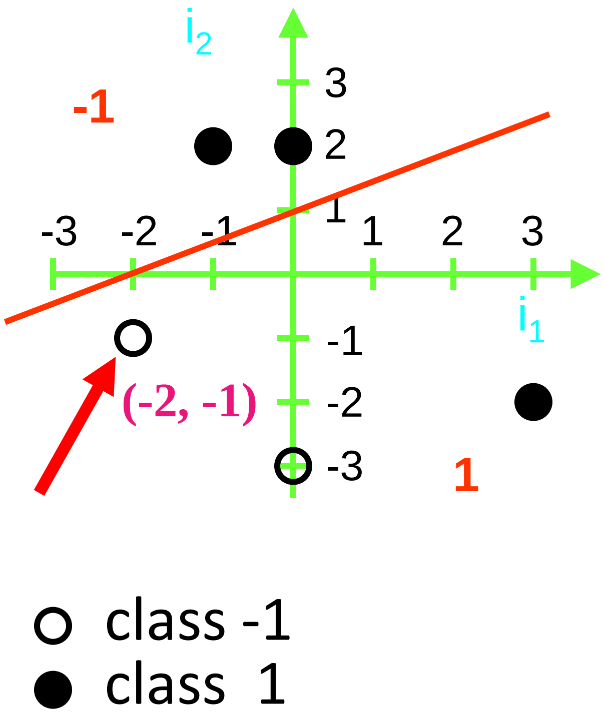
which are $(-2, 0)$ and $(0, 1)$.

Weight adaptation for learning:

$$w_i \leftarrow w_i \pm cx_i$$

Example(cont…)

Let us pick the misclassified point $(x_1 , x_2) = (-2, -1)$



Considering **learning rate c=1,** $x_0 = 1$

$\mathbf{x} = (x_0, x_1, x_2) = (1, -2, -1)$

Since y=1, d= -1,

decrease the weight $\Delta w = -c\mathbf{x}$

$\Delta w = (-1)\cdot(1, -2, -1)$

$\Delta w = (-1, 2, 1)$

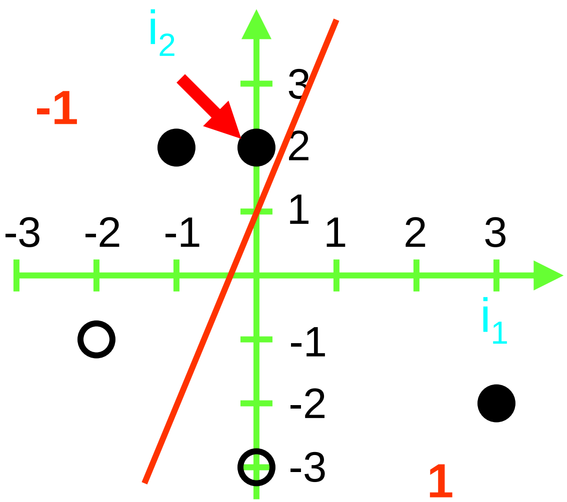
$\mathbf{w^1} = \mathbf{w^0} + \Delta w$

$\mathbf{w^1} = (2, 1, -2) + (-1, 2, 1) = (1, 3, -1)$

Example (cont…)

$\mathbf{w}^1 = (2, 1, -2) + (-1, 2, 1) = (1, 3, -1)$ $[(-w_0/w_1, 0)$ **and** $(0, -w_0/w_2)]$

The new dividing line intersects the axes at (-1/3, 0) and (0, 1).



○ class -1
● class 1

Let us pick the next misclassified point (0, 2) for learning:

$\mathbf{x} = (1, 0, 2)$ (include $x_0 = 1$)

$\Delta w = (1).\ (1, 0, 2)$ ($y = -1$, $d = 1$)

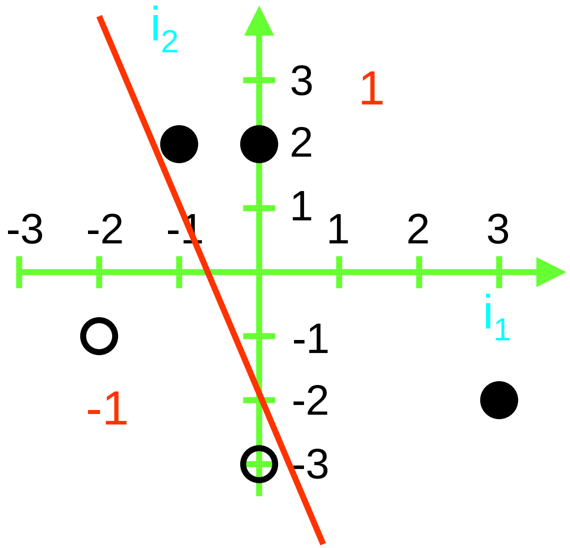
$\mathbf{w}^2 = (1, 3, -1) + \Delta w = (2, 3, 1)$

Why do you think we pick the closest misclassified point?

Example (cont…)

$\mathbf{w}^2 = (2, 3, 1)$ [at $(-w_0/w_1, 0)$ and $(0, -w_0/w_2)$]

Now the line crosses at (-2/3, 0) and (0, -2).



○ class -1

● class 1

- With this weight vector, the perceptron achieves perfect classification!
- The learning process terminates.
- In most cases, many more iterations are necessary than in this example.
  - If *n* data points are given, one iteration through *n* points to adjust weights is called one epoch.
  - Multiple epochs are generally needed to train an ANN.

How do you know the algorithm works?

- Activation $a = \Sigma\, w.x$
- If y=1 and D=0, then $(w - \Delta w).x < w.x$ ---- (1)
- Considering learning rate c=1, $\Delta w = cx = x$
- Therefore, left side of (1) can be written as

  $(w - \Delta w).x \Rightarrow w.x - x.x$
- But $x.x > 0$ (squared values are always +ve)
- So, $(w.x - x.x)$ or w.x reduced by a +ve value must be less than w.x
- Therefore, weight adjustments would eventually lead to a weight value that will correctly classify the input data.
- Same justification can be used for y=0 and D=1.

Perceptron Convergence Theorem

- It can be guaranteed that the Perceptron training algorithm will classify all the data correctly **when they are linearly separable and *c* is sufficiently small**.
- ***See proof in the book in the slides posted on OnQ.

Summary

- First learning algorithm for perceptron
- Simple feedback learning
- Formal notations