# Lecture 11: The Perceptron

**COMP90049**
**Introduction to Machine Learning**
Semester 2, 2022

Lida Rashidi, CIS

THE UNIVERSITY OF MELBOURNE

# Introduction

**So far... Naive Bayes and Logistic Regression**

- KNN
- Probabilistic models
- Maximum likelihood estimation
- Examples and code

**So far... Naive Bayes and Logistic Regression**

- KNN
- Probabilistic models
- Maximum likelihood estimation
- Examples and code

**Today... The Perceptron**

- Geometric motivation
- Error-based optimization
- ...towards neural networks

**Naive Bayes**

- Generative model of $p(x, y)$
- Find optimal parameter that maximize the log data likelihood
- Unrealistic independence assumption $p(x|y) = \prod_i p(x_i|y)$

**Logistic Regression**

- Discriminative model of $p(y|x)$
- Find optimal parameters that maximize the conditional log data likelihood
- Allows for more complex features (fewer assumptions)

**Naive Bayes**

- Generative model of $p(x, y)$
- Find optimal parameter that maximize the log data likelihood
- Unrealistic independence assumption $p(x|y) = \prod_i p(x_i|y)$

**Logistic Regression**

- Discriminative model of $p(y|x)$
- Find optimal parameters that maximize the conditional log data likelihood
- Allows for more complex features (fewer assumptions)

**Perceptron**

- Biological motivation: imitating neurons in the brain
- No more probabilities
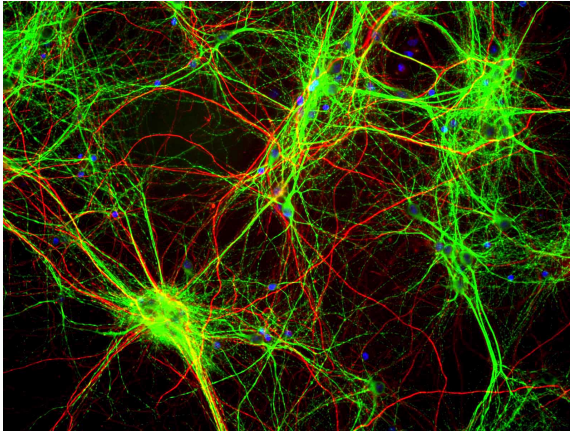- Instead: minimize the classification error directly
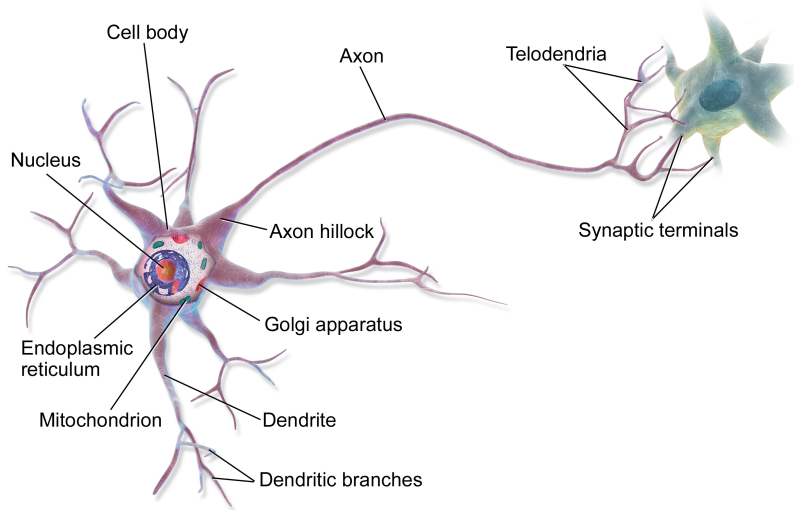
## Introduction: Neurons in the Brain

- Humans are the best learners we know
- Can we take inspiration from human learning
- $\rightarrow$ the brain!

https://vimeo.com/227026686

**The hype**

- 1943 McCulloch and Pitts introduced the first 'artificial neurons'
- If the **weighted sum of inputs** is equal to or greater than a **threshold**, then the **output** is 1. Otherwise the output is 0.
- the **weights** needed to be designed by hand

- In 1958 Rosenblatt invented the **Perceptron**, which can learn the optimal parameters through the **perceptron learning rule**
- The perceptron can be **trained** to learn the correct weights, even if randomly initialized [[ for a limited set of problems ]].



NEW NAVY DEVICE
LEARNS BY DOING

Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

WASHINGTON, July 7 (UPI)
—The Navy revealed the em-
bryo of an electronic computer
today that it expects to be
able to walk, talk, see, write,
reproduce itself and be con-
scious of its existence.
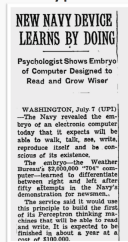
The New York Times, July 8 1958
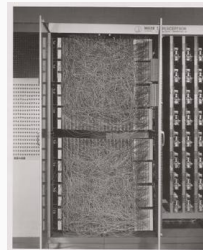
**The AI winter**

- A few years later Misky and Papert (too?) successfully pointed out the fundamental limitations of the perceptron.
- As a result, research on artificial neural networks stopped until the mid-1980s
- But the limitations can be overcome by combining multiple perceptrons into **Artificial Neural Networks**
- The perceptron is the basic component of today's deep learning success!

$$y = f(\theta^\top x)$$
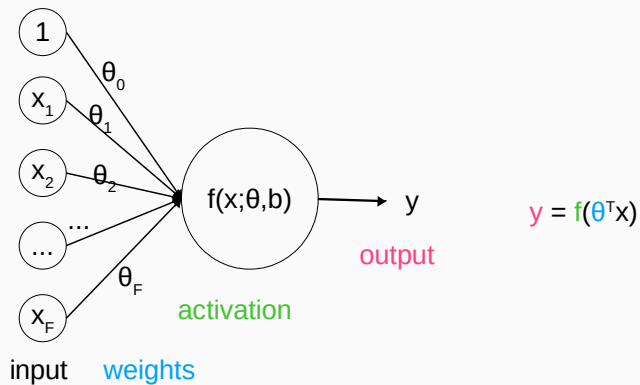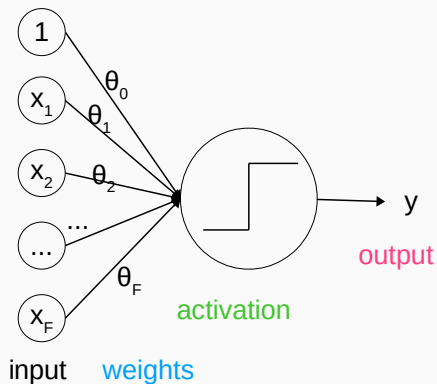
$y = f(\theta x + b)$

$f: y = 1$ if $f(\theta^T x) >= 0$
$\quad y = -1$ if $f(\theta^T x) < 0$

## Perceptron: Definition I

- The Perceptron is a **minimal neural network**

- **neural networks** are composed of **neurons**

- A neuron is defined as follows:
  - input = a vector $x$ of numeric inputs ($\langle 1, x_1, x_2, ... x_n \rangle$)
  - output = a scalar $y_i \in \mathbb{R}$
  - hyper-parameter: an **activation function** $f$
  - parameters: $\theta = \langle \theta_0, \theta_1, \theta_2, ... \theta_n \rangle$

- Mathematically:

$$y^i = f\left(\left[\sum_j \theta_j x_j^i\right]\right) = f(\theta^T x^i)$$

## Perceptron: Definition II

- Task: binary classification of instances into classes 1 and $-1$
- Model: a single-neuron (aka a "perceptron") :

$$f(\theta^T x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- $\theta^T x$ is the decision boundary
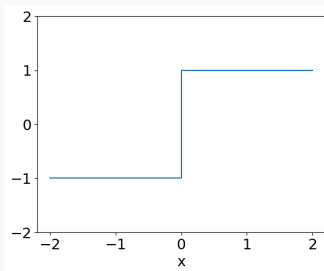- Graphically, $f$ is the **step function**

## Perceptron: Definition II

- Task: binary classification of instances into classes 1 and $-1$
- Model: a single-neuron (aka a "perceptron") :

$$f(\theta^T x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- $\theta^T x$ is the decision boundary
- Example: 2-d case:

## Towards the Perceptron Algorithm I

- As usual, **learning** means to modify the **parameters** (i.e., weights) of the perceptron so that performance is **optimized**

- The perceptron is a **supervised** classification algorithm, so we learn from observations of input-label pairs

$$(x^1, y^1), (x^2, y^2), ... (x^N, y^N)$$

- Simplest way to learn: compare predicted outputs $\hat{y}$ against true outputs $y$ and minimize the number of mis-classifications. Unfortunately, mathematically inconvenient.

- Second simplest idea: Find $\theta$ such that gap between the predicted value $\hat{y}^i \leftarrow f(\theta^T x^i)$ and the true class label $y \in \{-1, 1\}$ is minimized

**Intuition** Iterate over the **training data** and modify weights:

- if the true label $y = 1$ and $\hat{y} = 1$ then **do nothing**
- if the true label $y = -1$ and $\hat{y} = -1$ then **do nothing**
- if the true label $y = 1$ but $\hat{y} = -1$ then **increase** weights
- if the true label $y = -1$ but $\hat{y} = 1$ then **decrease** weights

## Towards the Perceptron Algorithm I

**Intuition** Iterate over the **training data** and modify weights:

- if the true label $y = 1$ and $\hat{y} = 1$ then **do nothing**
- if the true label $y = -1$ and $\hat{y} = -1$ then **do nothing**
- if the true label $y = 1$ but $\hat{y} = -1$ then **increase** weights
- if the true label $y = -1$ but $\hat{y} = 1$ then **decrease** weights

**More formally**

---

Initialize parameters $\theta \leftarrow 0$
**for** training sample $(x, y)$ **do**
    Calculate the output $\hat{y} = f(\theta^T x)$
    **if** $y = 1$ and $\hat{y} = -1$ **then**
        $\theta^{(new)} \leftarrow \theta^{(old)} + x$
    **if** $y = -1$ and $\hat{y} = 1$ **then**
        $\theta^{(new)} \leftarrow \theta^{(old)} - x$
**until** tired

---

- We set a **learning rate** or **step size** $\eta$
- and note that

$$(y^i - \hat{y}^i) = \begin{cases} 0 \text{ if } y^i == \hat{y}^i \\ 2 \text{ if } y^i = 1 \, and = \hat{y}^i = -1 \\ -2 \text{ if } y^i = -1 \, and = \hat{y}^i = 1 \end{cases} \quad (1)$$

- For each individual weight $\theta_j$, we compute an update such that

$$\theta_j \leftarrow \theta_j + \eta(y^i - \hat{y}^i)x_j^i$$

$D = \{(\mathbf{x}^i, y^i) | i = 1, 2, ..., N\}$ the set of training instances
Initialise the weight vector $\theta \leftarrow 0$
$t \leftarrow 0$
**repeat**
   $t \leftarrow t+1$
   **for** each training instance $(x^i, y^i) \in D$ **do**
      compute $\hat{y}^{i,(t)} = f(\theta^T x^i)$
      **if** $\hat{y}^{i,(t)} \neq y^i$ **then**
         **for** each each weight $\theta_j$ **do**
            update $\theta_j^{(t)} \leftarrow \theta_j^{(t-1)} + \eta(y^i - \hat{y}^{i,(t)})x_j^i$
      **else**
         $\theta_j^{(t)} \leftarrow \theta_j^{(t-1)}$
**until** tired
Return $\theta^{(t)}$

**An example**

- Training instances:

| $\langle x_{i1}, x_{i2} \rangle$ | $y_i$ |
|:---:|:---:|
| $\langle 1, 1 \rangle$ | $1$ |
| $\langle 1, 2 \rangle$ | $1$ |
| $\langle 0, 0 \rangle$ | $-1$ |
| $\langle -1, 0 \rangle$ | $-1$ |

- Learning rate $\eta = 1$

## Perceptron Example II

- $\theta = \langle 0, 0, 0 \rangle$
- learning rate: $\eta = 1$
- Epoch 1:

| $\langle x_1, x_2 \rangle$ | $\theta_0 \cdot 1 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$ | $\hat{y}_i^{(1)}$ | $y_i$ |
|---|---|---|---|
| $\langle 1, 1 \rangle$ | $0 + 1 \times 0 + 1 \times 0 = 0$ | 1 | 1 |
| $\langle 1, 2 \rangle$ | $0 + 1 \times 0 + 2 \times 0 = 0$ | 1 | 1 |
| $\langle 0, 0 \rangle$ | $0 + 0 \times 0 + 0 \times 0 = 0$ | 1 | -1 |
| | Update to $\theta = \langle -2, 0, 0 \rangle$ | | |
| $\langle -1, 0 \rangle$ | $-2 + -1 \times 0 + 0 \times 0 = -2$ | -1 | -1 |

## Perceptron Example III

- $\theta = \langle -2, 0, 0 \rangle$
- learning rate: $\eta = 1$
- Epoch 2:

| $\langle x_1, x_2 \rangle$ | $\theta_0 \cdot 1 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$ | $\hat{y}_i^{(2)}$ | $y_i$ |
|---|---|---|---|
| $\langle 1, 1 \rangle$ | $-2 + 1 \times 0 + 1 \times 0 = -2$ | -1 | 1 |
| | Update to $\theta = \langle 0, 2, 2 \rangle$ | | |
| $\langle 1, 2 \rangle$ | $0 + 1 \times 2 + 2 \times 2 = 6$ | 1 | 1 |
| $\langle 0, 0 \rangle$ | $0 + 0 \times 2 + 0 \times 2 = 0$ | 1 | -1 |
| | Update to $\theta = \langle -2, 2, 2 \rangle$ | | |
| $\langle -1, 0 \rangle$ | $-2 + -1 \times 2 + 0 \times 2 = -4$ | -1 | -1 |

- $\theta = \langle -2, 2, 2 \rangle$
- learning rate: $\eta = 1$
- Epoch 3:

| $\langle x_1, x_2 \rangle$ | $\theta_0 \cdot 1 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$ | $\hat{y}_i^{(3)}$ | $y_i$ |
|---|---|---|---|
| $\langle 1, 1 \rangle$ | $-2 + 1 \times 2 + 1 \times 2 = 2$ | 1 | 1 |
| $\langle 1, 2 \rangle$ | $-2 + 1 \times 2 + 2 \times 2 = 4$ | 1 | 1 |
| $\langle 0, 0 \rangle$ | $-2 + 0 \times 2 + 0 \times 2 = -2$ | -1 | -1 |
| $\langle -1, 0 \rangle$ | $-2 + -1 \times 2 + 0 \times 2 = -4$ | -1 | -1 |

- Convergence, as no updates throughout epoch

## Perceptron Convergence

**Perceptron Rule:**

$$\theta_j^{(t+1)} \leftarrow \theta_j^{(t)} + \eta(y_i - \hat{y}^i)x_j^i$$

- So, all we're doing is adding and subtracting constants every time we make a mistake.
- Does this really work!?

**Perceptron Convergence**

- The Perceptron algorithm is guaranteed to **converge** for linearly-separable data

    - the convergence point will depend on the initialisation
    - the convergence point will depend on the learning rate
    - (no guarantee of the margin being maximised)

- No guarantee of convergence over non-linearly separable data

**Perceptron Rule**

$$\theta_j^{(t+1)} \leftarrow \theta_j^{(t)} + \eta(y_i - \hat{y}^i)x_j^i$$

**Gradient Descent**

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \frac{\partial f}{\partial \theta^{(t)}}$$

**Activation Functions**
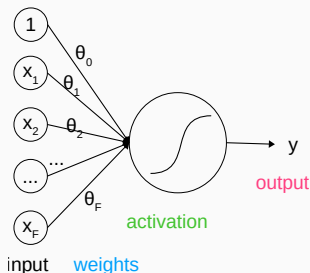
**Perceptron Rule**

$$\theta_j^{(t+1)} \leftarrow \theta_j^{(t)} + \eta(y_i - \hat{y}^i)x_j^i$$

**Gradient Descent**

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta\frac{\partial f}{\partial \theta^{(t)}}$$

**Activation Functions**

A single 'neuron' with a **sigmoid activation** which optimizes the **cross-entropy** loss (negative log likelihood) is equivalent to **logistic regression**



f: y = 1 / (1 + exp(-θᵀx))

16

## Online learning vs. Batch learning

- It is an **online algorithm**: we update the weights after each training example

- In contrast, Naive Bayes and logistic regression (with Gradient Descent) are updated as a **batch** algorithm:
  - compute statistics of the *whole* training data set
  - update all parameters at once

- Online learning can be more efficient for large data sets

- Gradient Descent can be converted into an online version: **stochastic gradient descent**

## Multi-Class Perceptron

**We can generalize the perceptron to more than 2 classes**

- create a weight vector for each class $k \in Y$, $\theta^k$
- score input wrt each class: $\theta_k^T x$ for all $k$
- predict the class with maximum output $\hat{y} = \text{argmax}_{k \in Y} \theta_k^T x$
- learning works as before: if for some $(x^i, y^i)$ we make a wrong prediction $\hat{y}^i \neq y^i$ such that $\theta_{y^i}^T x^i < \theta_{\hat{y}^i}^T x^i$,

$$\theta_{y^i} \leftarrow \theta_{y^i} + \eta x^i \qquad \text{move towards predicting } y^i \text{ for } x^i$$
$$\theta_{\hat{y}^i} \leftarrow \theta_{\hat{y}^i} - \eta x^i \quad \text{move away from predicting } \hat{y}^i \text{ for } x^i$$

## Summary

**This lecture:** The Perceptron

- Biological motivation
- Error-based classifier
- The Perceptron Rule
- Relation to Logistic Regression
- Multi-class perceptron

**Next**

- More on Evaluation
- More on Unsupervised/Semi-supervised Learning
- More powerful machine learning through combining Perceptrons

## References

- Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

- Minsky, Marvin, and Seymour Papert. "Perceptrons: An essay in computational geometry." MIT Press. (1969).

- Bishop, Christopher M. Pattern recognition and machine learning. Springer, 2006. Chapter 4.1.7