Introduction to Deep Learning

Machine Learning basics

Alexandre Allauzen

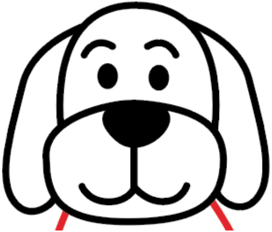
Jan. 2025

Outline

Outline

Starter: a “simple” question

What it is ?

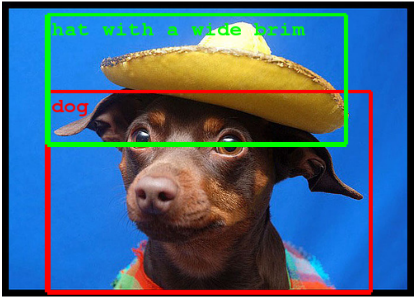


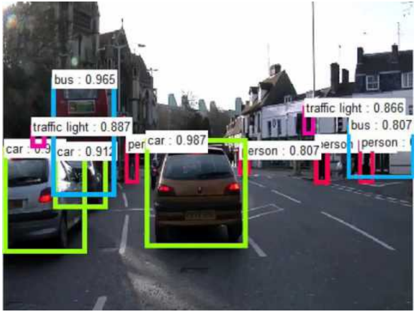
Can you write an algorithm to answer ?

- What is the input space ?
- What do you want to predict ?
- The output space ?

And more questions ?

Object detection



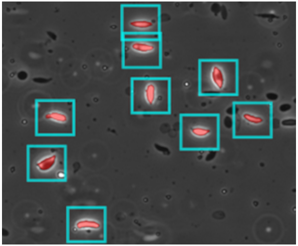


- A review: (Liu et al.2020)
- Real time detection in video, e.g Yolo (Redmon and Farhadi2018)

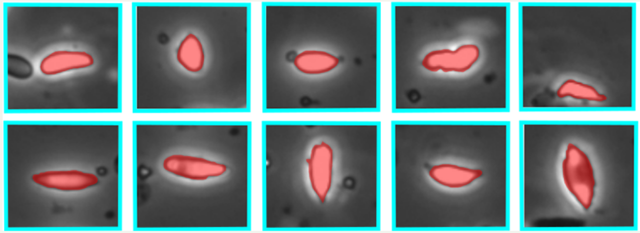
Cells classification / detection (Praljak et al.2020)

Red blood cell detection

- Images from a biochip (microfluidic assay)
- Detect and count sickle cells



Red Blood Cell classification



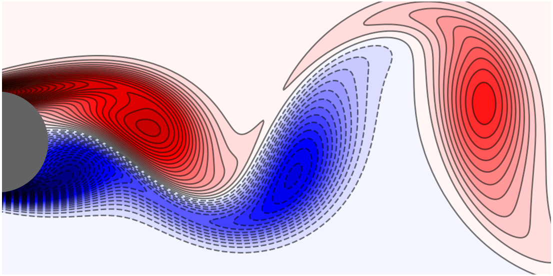
Sequence classification and generation

Classification

$$\boxed{\begin{array}{c} \text{A T C G A T T C} \\ \cdots \text{ G T A A T C G} \end{array}} \;:\; \mathbf{x} \in \mathbb{R}^D \longrightarrow c \in \{0,1\}$$

- Enhancer Identification in DNA Sequences (Min et al.2017; Cohn et al.2018)
- Predicting sequence specificities (Alipanahi et al.2015)

Generation



The "Machine Learning" way

How to make a 'computer' do a specific task?

Traditional approach (old AI)

A program is:

- hand-coded
- specific set of instructions to complete the task
- can be explained and proved, "always" gives the correct answer

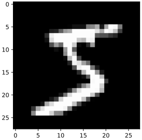
Machine Learning

A program is:

- trained or learnt
- from large amount of annotated data
- algorithm + inductive bias
- it works ... on average

Machine learning: the main tasks

Supervised classification



$$: \quad \mathbf{x} \in \mathbb{R}^{784} \longrightarrow c \in \{0, 1, 2, \ldots, 9\}$$

- $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^{N}$
- **Supervised** learning of a **classification** task

Machine learning: the main tasks

Supervised binary classification or regression

my wonderful friend took me to see this movie for our anniversary. it was terrible.

$: \ \mathbf{x} \in \mathbb{R}^D \ \longrightarrow \ c \in \{0, 1\}$

- $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^{N}$
- Supervised learning of a **binary** classification task

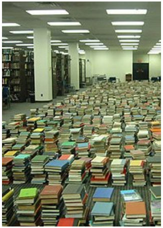
Supervised regression

$$\mathbf{x} \in \mathbb{R}^D \ \longrightarrow \ y \in \mathbb{R} \text{ (e.g } [0, 1]) \text{ or } \mathbf{y} \in \mathbb{R}^m$$

Machine learning: the main tasks

Unsupervised learning / Clustering

$\Longrightarrow$



$$\mathbf{x} \in \mathbb{R}^D \longrightarrow \boldsymbol{z} \in \mathbb{R}^M$$

- A set of observations $\mathcal{D} = (\mathbf{x}_{(i)})_{i=1}^N$ must be assigned to a cluster $\rightarrow z$
- The model infer a hidden/latent structure in the $\mathcal{D}$
- The *guidelines*: the structure of the model, the assumptions (distance, similarty between the $\mathbf{x}$, ...)
- Dimensionality reduction, data mining, ...

Overview in 3 ingredients

The NNet (in one equation)

$$\mathbf{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$$

- $\mathbf{x}$ and $\mathbf{y}$ (structured, in high dimension, ...)
- $f_{\boldsymbol{\theta}}$ is the NNet
- $\boldsymbol{\theta}$ its parameters

The training dataset

$$\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{N}$$

- For supervised learning
- N can (should be) large

Optimisation

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D})$$

- Require time and tricks
- and GPUs !

The structure of f_θ

For image

- Matrices (one per channels) with a spatial structure
- Convolutional NNet

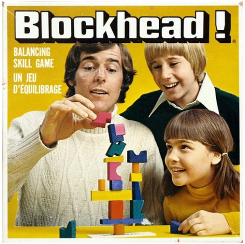
Sequence and time series

- Convolutional NNet (1D)
- Recurrent architectures, Transformers

Graph

- Graph convolutional NNet
- Graph Transformer based attention

A deep NNet:



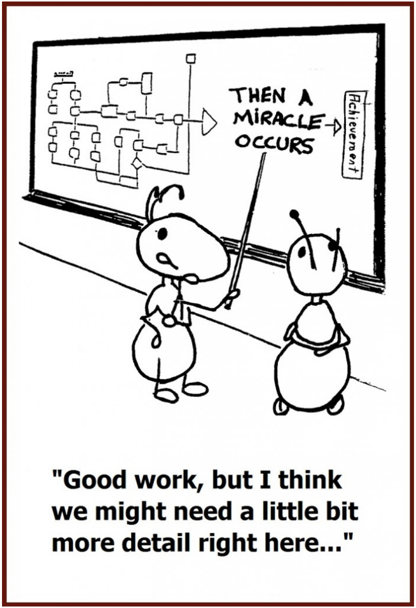
A success story

Since 2009, deep learning approaches won several challenges

- Image classification, object detection, ...
- Handwritting recognition since 2009
- Machine translation, Automatic Speech Recognition
- ...

But it started a bit earlier

- 1943: The artificial neuron by McCulloch & Pitts
- 1949: Hebb's rule for update
- 1958: Rosenblatt and the perceptron
- ...
- 2006: "Deep learning"
- 2018: "Transformers"


THEN A MIRACLE OCCURS
Achievement
"Good work, but I think we might need a little bit more detail right here..."

Outline

Outline of the course

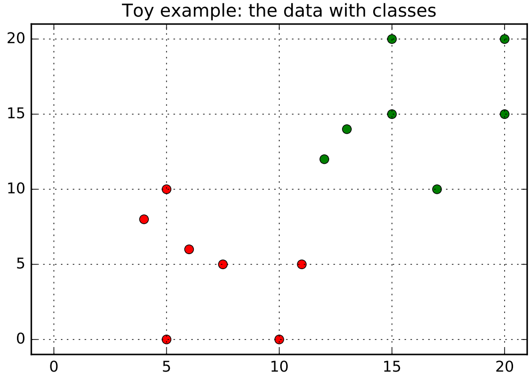
- Machine Learning Basics
- Feed-forward Neural Network: going deep
- Tools for deep-learning
- Convolution and Recurrent Nets
- Transformers

Outline

The A simple example: binary classification in 2D

The dataset $\mathcal{D}$

$$\mathcal{D} = \begin{pmatrix} x_1 = & 17 & 12 & 13 & 15 & 15 & 20 & 20 & 4 & 7.5 & 10 & 11 & 5 & 5 & 6 \\ x_2 = & 10 & 12 & 14 & 15 & 20 & 15 & 20 & 8 & 5 & 0 & 5 & 0 & 10 & 6 \\ c = & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Example of binary classification

Goal:

- Predict whether a candidate is hired ($c = 1$) or not ($c = 0$)
- knowing his results $\mathbf{x}$
- a candidate = 2 grades $\rightarrow \mathbf{x} = (x_1, x_2)$

The simplest model

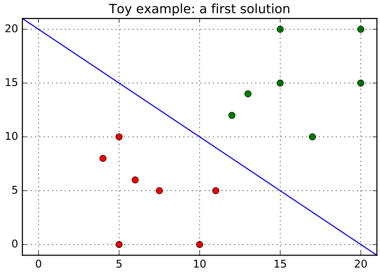
$$c = 1 \text{ if } w_1 x_1 + w_2 x_2 > \alpha \text{ otherwise } c = 0$$
$$c = 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 > 0 \text{ otherwise } c = 0$$
$$c = 1 \text{ if } w_0 + \mathbf{w}^t \mathbf{x} > 0 \text{ otherwise } c = 0 \text{ (with } \mathbf{w} = (w_1, w_2))$$

Learning

- Given $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^{n}$ find $\boldsymbol{\theta} = (w_0, \mathbf{w})$
- $c_{(i)}$ is the good answer (the supervision)

Binary classification or separation

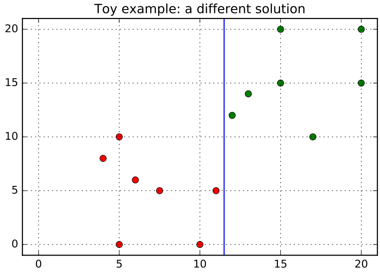
- Given $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^{n}$ find $\boldsymbol{\theta} = (w_0, \mathbf{w})$
- $c_{(i)}$ is the good answer (the supervision)



- $\boldsymbol{\theta}$ defines the separation
- $w_0 + \mathbf{w}^t\mathbf{x} = 0$: on the line
- $w_0 + \mathbf{w}^t\mathbf{x} > \mathbf{0}$ or $< \mathbf{0}$
- $\frac{w_0 + \mathbf{w}^t\mathbf{x}}{||\mathbf{w}||}$: the distance between the line and $\mathbf{x}$

Binary classification or separation

- Given $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^{n}$ find $\boldsymbol{\theta} = (w_0, \mathbf{w})$
- $c_{(i)}$ is the good answer (the supervision)



- $\boldsymbol{\theta}$ defines the separation
- $w_0 + \mathbf{w}^t\mathbf{x} = 0$: on the line
- $w_0 + \mathbf{w}^t\mathbf{x} > \mathbf{0}$ or $< \mathbf{0}$
- $\frac{w_0 + \mathbf{w}^t\mathbf{x}}{||\mathbf{w}||}$: the distance between the line and $\mathbf{x}$

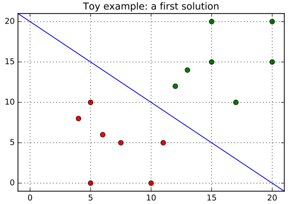
Regression or classification

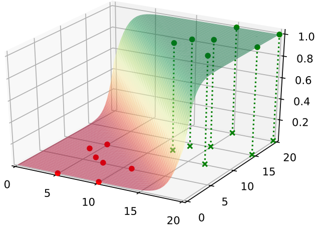
The class c is the outcome of the binary random variable C

$$y = P(C = 1|\mathbf{x}) = \sigma(w_0 + \mathbf{w}^t\mathbf{x})$$

$$\sigma(a) = \frac{e^a}{1 + e^a} = \frac{1}{1 + e^{-a}}$$

$$a = w_0 + \mathbf{w}^t\mathbf{x}, a \in \mathbb{R}$$





Machine Learning

- Learn the parameters $\boldsymbol{\theta} = (w_0, \mathbf{w})$ from $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^n$
- By minimizing a loss function ($a.k.a$ empirical risk)

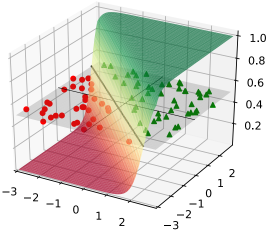
From Maximum-Likelihood to loss minimization

Given $\mathcal{D} = (\mathbf{x}_{(i)}, c_{(i)})_{i=1}^{n} = (\mathcal{X}, \tilde{\mathcal{C}})$ and a logistic regression model, $\boldsymbol{\theta} = (w_0, \mathbf{w})$.

Maximize the Log-Likelihood

$$\log P(\tilde{\mathcal{C}}|\mathcal{X}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \log P(C = c_{(i)}|\mathbf{x}_{(i)}, \boldsymbol{\theta})$$

$$= \sum_{i=1}^{n} \underbrace{c_{(i)} log(y_{(i)})}_{c_{(i)}=1} + \underbrace{(1 - c_{(i)}) log(1 - y_{(i)})}_{c_{(i)}=0}$$



Minimize the loss

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)}), \text{ with}$$

$$l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)}) = -c_{(i)} log(y_{(i)}) - (1 - c_{(i)}) log(1 - y_{(i)})$$

Optimization with Gradient Descent

Setup

$$\operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)})$$

This function is convex.

Gradient Descent

- Start with an initial value $\boldsymbol{\theta}^{(0)}$ and set $k = 1$
- Repeat until convergence:

$$\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k-1)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}), \text{ with}$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{\partial \mathcal{L}(\boldsymbol{\theta}; \mathcal{D})}{\partial \boldsymbol{\theta}}$$

$$\eta > 0, \text{ the } \textit{learning rate} \text{ or } \textit{gradient step}$$

Very efficient (in time) with matrix-matrix operations

Bias or not bias

A matter of notation

The bias can be explicit:

$$w_0 + \mathbf{w}^t\mathbf{x} = w_0 + w_1 x_1 + \cdots + w_D x_D$$

or implicit:

$$\mathbf{w} \cdot \mathbf{x} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

$$= w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$$

Summary

Machine Learning overview

- A model defines a **function**: $\mathbf{x} \rightarrow \boldsymbol{y}$
- The output is then used by a decision rule.
- This function is defined by the parameters $\boldsymbol{\theta}$.
- **Training/Learning** finds the good values for $\boldsymbol{\theta}$
- How ? by minimizing a **loss function** $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$.
- How ? by **Stochastic Gradient Descent**
- The **learning rate** is an **hyper-parameter**.

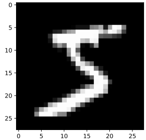
From logistic regression to artificial neurone

- An artificial neurone is a **linear model** followed by a non-linear **activation**
- With the sigmoid activation, it is similar to the **logistic regression**

Outline

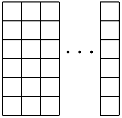
The MNIST dataset

The task



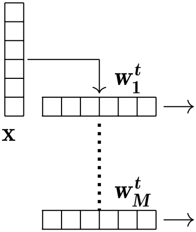
$\longrightarrow$  $\mathbf{x}_{(i)} \in \mathbb{R}^D, D = 784 \longrightarrow c_{(i)} \in \{0, 1, 2, \dots, 9\}$

The dataset

- $\mathcal{D} = (\mathbf{X}, \mathbf{c})$
- $\mathbf{X} =$  is a (D, N) matrix
- The labels $\mathbf{c}$ is $(N, 1)$
- $N = 50\ 000$

The model

From binary classification to a multiclass problem



$$y_1 = f(\mathbf{w_1}^t\mathbf{x})$$

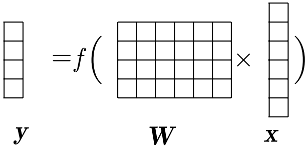
$$y_M = f(\mathbf{w_M}^t\mathbf{x})$$

$$y_k = f(a_k = \mathbf{w_k}^t\mathbf{x}) = P(C = k|\mathbf{x}, \boldsymbol{\theta})$$

$$= \frac{e^{a_k}}{\sum_{k'=1}^{M} e^{a_{k'}}} = \frac{e^{a_k}}{Z(\mathbf{x})}$$

$$\sum_{k=1}^{M} y_k = \sum_{k=1}^{M} P(c = k|\mathbf{x}) = 1$$

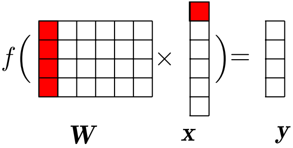
The matrix view



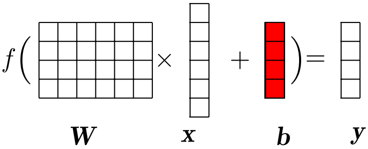
- $\mathbf{x} : (D, 1)$
- $\mathbf{W} : (M, D)$
- $\mathbf{y} : (M, 1)$

Bias or not bias

Implicit Bias



Explicit bias



Loss function

The parameters

$$\boldsymbol{\theta} = \mathbf{W}$$

The negative log-likelihood loss (or Cross-entropy)

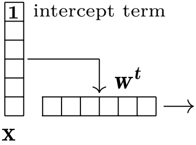
$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = -\sum_{i=1}^{N} \log P(C = c_{(i)} | \mathbf{x}_{(i)}, \boldsymbol{\theta})$$

(Mini-)Batch training

$$\mathbf{Y} = f(\mathbf{W} \times \mathbf{X}) \rightarrow \text{ inference for all the batch in once}$$

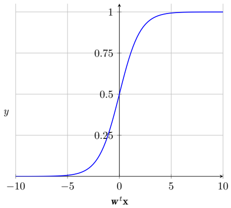
A choice of terminology

Logistic regression (binary classification)

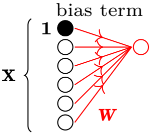


$\longrightarrow \quad y = f(\mathbf{w}^t\mathbf{x})$

$$f(a = \mathbf{w}^t\mathbf{x}) = \frac{e^a}{1+e^a} = \sigma(a)$$



A single artificial neuron

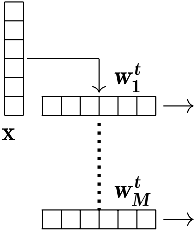


$y = f(\mathbf{w}^t\mathbf{x})$

- pre-activation : $a = \mathbf{w}^t\mathbf{x}$
- f: activation function
- Input values = input “neurones”
- $\mathbf{x}$: a vector of values, a layer

A choice of terminology - 2

From binary classification to M classes (Maxent)



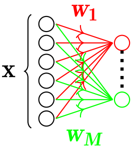
$$y_1 = f(\mathbf{w_1}^t\mathbf{x})$$

$$y_M = f(\mathbf{w_M}^t\mathbf{x})$$

$$y_k = f(a_k = \mathbf{w_k}^t\mathbf{x}) = P(c = k|\mathbf{x})$$
$$= \frac{e^{a_k}}{\sum_{k'=1}^{M} e^{a_{k'}}} = \frac{e^{a_k}}{Z(\mathbf{x})}$$

$$\sum_{k=1}^{M} y_k = \sum_{k=1}^{M} P(c = k|\mathbf{x}) = 1$$

A simple neural network

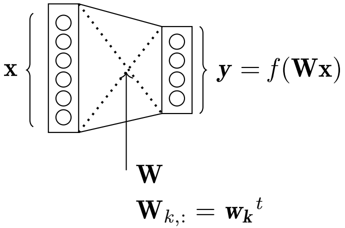


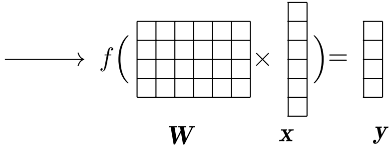
$$y_1 = f(\mathbf{w}_1^t\mathbf{x})$$

$$y_M = f(\mathbf{w}_M^t\mathbf{x})$$

- $\mathbf{x}$: *input layer*
- $\boldsymbol{y}$: *output layer*
- each y_k has its parameters $\boldsymbol{w}_k$
- f is the **softmax** function

Two layers fully connected





Activation f:

- f is usually a non-linear function
- f is a component wise function
- tanh, sigmoid, relu, ...

$e.g$ the softmax function:

Dimensions:

- $\mathbf{x} : D \times 1$
- $\boldsymbol{W} : M \times D$
- $\boldsymbol{y} : (M \times \cancel{D) \times (D} \times 1) = M \times 1$

$$y_k = P(c = k|\mathbf{x}) = \frac{e^{\boldsymbol{w_k}^t\mathbf{x}}}{\sum_{k'} e^{\boldsymbol{w_{k'}}^t\mathbf{x}}} = \frac{e^{\mathbf{W}_{k,:}\mathbf{x}}}{\sum_{k'} e^{\mathbf{W}_{k',:}\mathbf{x}}}$$

Classification with a simple and shallow network

Binary classification

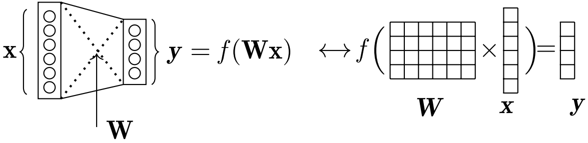
- The input layer is a vector ($\mathbf{x}$), it encodes the data
- A single output neuron transform $\mathbf{x}$, $\boldsymbol{w}$ are its parameters
- With a sigmoïd activation, the loss function is the binary cross entropy, the log-loss, minus the log-likelihood, ...

Multiclass

- The input layer is a vector ($\mathbf{x}$)
- The output layer $\mathbf{y}$ contains one neuron per class
- It transforms $\mathbf{x}$, $\mathbf{W}$ are the parameters of the output layer
- $\mathbf{W}$ gathers the $\boldsymbol{w}_k = \mathbf{W}_{k,:}$
- With a softmax activation, the loss function is the cross entropy, the log-loss, minus the log-likelihood, ...

Other loss functions exist for classification

Two layers fully connected: another view

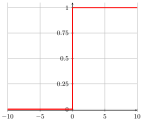


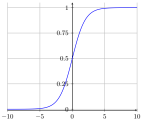
This basic brick implements a transformation of $\mathbf{x}$ in $\boldsymbol{y} = f(\mathbf{Wx})$:

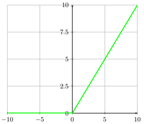
- A linear transformation $\mathbf{Wx}$
- Followed by a non-linear function

Example: a candidate made 6 interviews $\rightarrow \mathbf{x} \in \mathbb{R}^6$

- First compute 4 new scores : $\mathbf{Wx} \in \mathbb{R}^4$, each is a linear combination of $\mathbf{x}$
- Apply a non-linearity to get $\boldsymbol{y}$

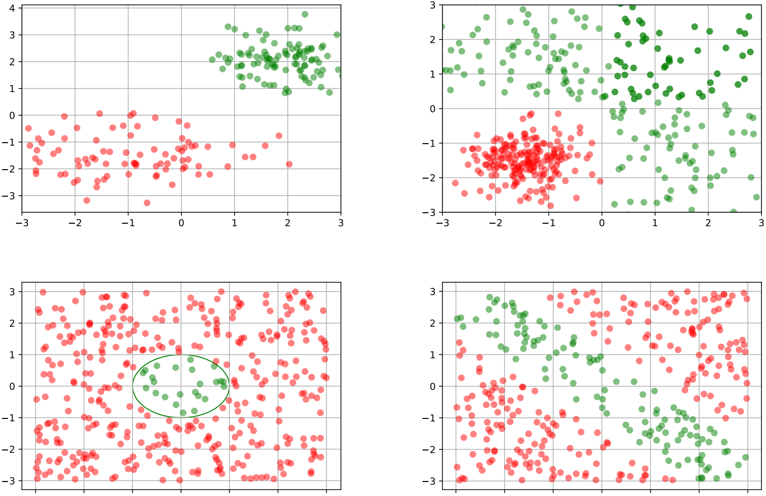






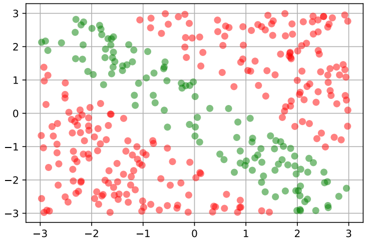
Outline

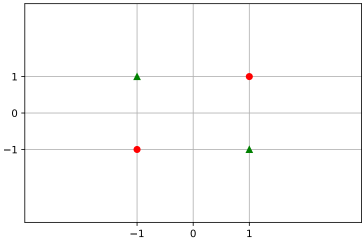
Limits of the linear separation



Case study : X-or

Starting point



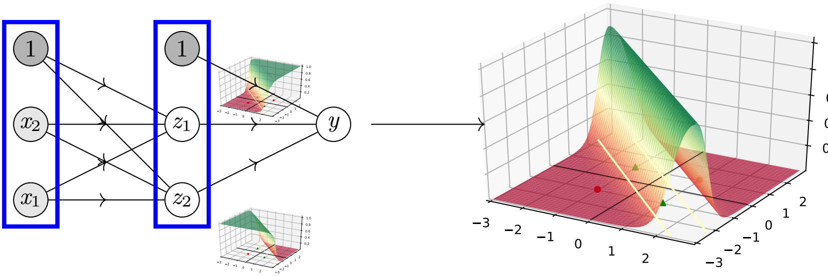


Hint

Try two linear separations instead of one !

A first neural network

The multi-layer architecture



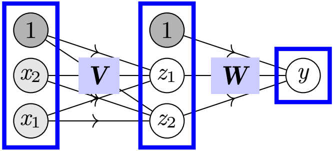
- The network is organized by layers: input ($\mathbf{x}$), hidden ($\mathbf{z}$), and output ($\mathbf{y}$)
- Two layers are fully connected
- The propagation of the input is sequential:

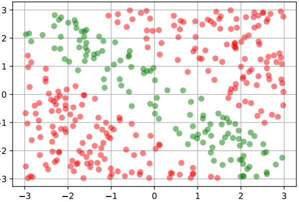
$$z_1 = g(\mathbf{v}_1^t\mathbf{x}) \text{ and } z_2 = g(\mathbf{v}_2^t\mathbf{x})) \qquad \Rightarrow \boldsymbol{z} = g(\boldsymbol{V}\mathbf{x})$$

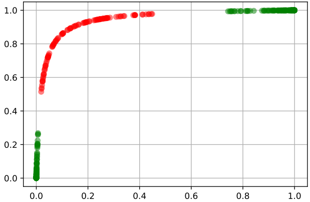
$$y = f(\mathbf{w}^t\boldsymbol{z}) \qquad \Rightarrow y = f(\boldsymbol{W}\boldsymbol{z})$$

Another view of the neural network

Representation learning







- The network learns its own representation $\boldsymbol{z}$.
- The final decision is linear.

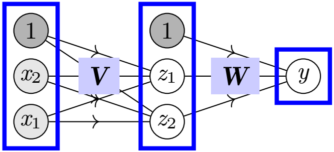
Summary

The multi-layer or feed-forward architecture

- 1 neuron = 1 value; 1 layer = 1 vector
- Two layers $(\boldsymbol{x}, \boldsymbol{z})$ fully connected:

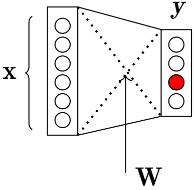
$$\boldsymbol{z} = g(\boldsymbol{V}\boldsymbol{x})$$

- Inference: a sequential propagation
- Hidden layer $(\boldsymbol{z})$: the internal representation



Outline

For a shallow network, with a single layer



$\boldsymbol{y} = f(\mathbf{W}\mathbf{x})$

$y_k = P(c = k|\mathbf{x})$

$\boldsymbol{\theta} = (\mathbf{W})$

Forward (inference) ↑		Backward (gradient) ↓
$l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)})$	$\leftarrow \boldsymbol{y}$	$\dfrac{\partial l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)})}{\partial \mathbf{W}} = \dfrac{\partial l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{y}}$
$\boldsymbol{y}$	$= f(\boldsymbol{a})$	$\times \dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}$
$\boldsymbol{a}$	$= \mathbf{W}\mathbf{x}$	$\times \dfrac{\partial \boldsymbol{a}}{\partial \mathbf{W}}$

Review of the gradient computation

Without hidden layer (shallow net)

l is the shortcut for $l(c_{(i)}, \mathbf{x}_{(i)}, \boldsymbol{\theta}_{(i)})$

$$\frac{\partial l}{\partial \mathbf{W}} = \frac{\partial l}{\partial y_{(i)}} \times \frac{\partial y_{(i)}}{\partial \mathbf{a}} \times \frac{\partial \mathbf{a}}{\partial \mathbf{W}}$$

Supervision

loss *vs* output

activation

gradient of **W**

Error signal from supervision and output

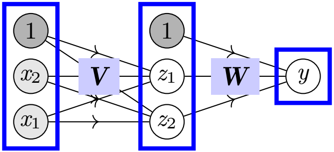
Contribution of **W** $\rightarrow \mathbf{x}$

In the matrix form

$$
\begin{aligned}
\nabla_{\mathbf{W}} = \frac{\partial l}{\partial \mathbf{W}} &= \frac{\partial l}{\partial y_{(i)}} \times \frac{\partial y_{(i)}}{\partial \mathbf{a}} \times \frac{\partial \mathbf{a}}{\partial \mathbf{W}} \\
&= \boldsymbol{\delta}_{\mathbf{W}} \times \mathbf{x}^t \\
&= (M, 1) \times (1, D) \\
&= \text{Gradient up to the pre-activation} \times \text{input of the layer}
\end{aligned}
$$

- $\nabla_{\mathbf{W}}$ is a matrix of size (M, D)
- $\nabla_{\mathbf{W}}[i, j] = \boldsymbol{\delta}[i] \times \mathbf{x}[j]$

Gradient computation with one hidden layer



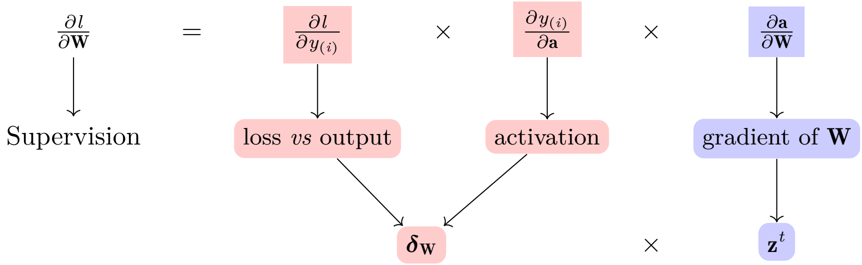
- $\boldsymbol{b} = \boldsymbol{V}\mathbf{x}$
- $\boldsymbol{z} = g(\boldsymbol{b})$
- $\boldsymbol{a} = \boldsymbol{W}\mathbf{z}$
- $\boldsymbol{y} = f(\boldsymbol{a})$

Two questions (or gradients):

- $\frac{\partial l}{\partial \mathbf{W}} = \ ?$
- $\frac{\partial l}{\partial \mathbf{V}} = \ ?$

Gradient computation with one hidden layer

Step 1: **W**



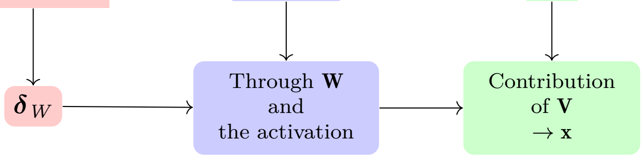
- Very similar to the shallow network, but the “input” is $\mathbf{z}$ instead of $\mathbf{x}$
- But $\mathbf{z}$ is computed by the network (parameters $\mathbf{V}$)

Gradient computation with one hidden layer

Step 2 : $\mathbf{V}$

Denote the pre-activation of the input layer $\mathbf{b} = \mathbf{V}\mathbf{x}$

$$\frac{\partial l}{\partial \mathbf{V}} = \frac{\partial l}{\partial y_{(i)}} \times \frac{\partial y_{(i)}}{\partial \mathbf{a}} \times \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \times \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \times \frac{\partial \mathbf{b}}{\partial \mathbf{V}}$$



- The error signal $\boldsymbol{\delta}_W$ is back-propagated through $\mathbf{W}$,
- to get $\boldsymbol{\delta}_V$
- The update for $\mathbf{V}$ is $\nabla_{\mathbf{V}} = \boldsymbol{\delta}_V \mathbf{x}^t$

The back-propagation algorithm for a feed-forward network with one hidden layer

Introduced in (Rumelhart et al.1986)

One iteration of the online version, for a given value of $\boldsymbol{\theta}$:

1. Foward propagation of $\mathbf{x}_{(i)}$ ($\rightarrow$ $\mathbf{z}$ and $\mathbf{y}_{(i)}$)
2. Compute the loss
3. Back-propagation and collect of the gradients:
   - output layer: $\boldsymbol{\delta}_W$ and $\nabla_{\mathbf{W}} = \boldsymbol{\delta}_W \mathbf{z}^t$
   - input layer: $\boldsymbol{\delta}_V$ and $\nabla_{\mathbf{V}} = \boldsymbol{\delta}_V \mathbf{x}^t$
4. Update parameters:

$$\mathbf{W} = \mathbf{W} - \eta \nabla_{\mathbf{W}}$$
$$\mathbf{V} = \mathbf{V} - \eta \nabla_{\mathbf{V}}$$

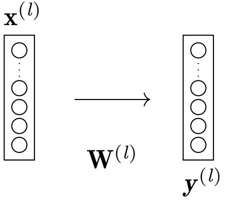
One training iteration: forward and backward propagation

Forward	Backward	
$\boldsymbol{b} = \boldsymbol{V}\mathbf{x}$	$\dfrac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$	$\rightarrow \dfrac{\partial l}{\partial \boldsymbol{b}} \dfrac{\partial \mathbf{b}}{\partial \boldsymbol{V}}$
$\downarrow$	$\uparrow$	
$\boldsymbol{z} = g(\boldsymbol{b})$	$\dfrac{\partial \mathbf{a}}{\partial \boldsymbol{z}}$	
$\downarrow$	$\uparrow$	
$\boldsymbol{a} = \boldsymbol{W}\mathbf{z}$	$\dfrac{\partial y_{(i)}}{\partial \mathbf{a}}$	$\rightarrow \dfrac{\partial l}{\partial \mathbf{a}} \dfrac{\partial \mathbf{a}}{\partial \boldsymbol{W}}$
$\downarrow$	$\uparrow$	
$\boldsymbol{y} = f(\boldsymbol{a})$	$\dfrac{\partial l}{\partial y_{(i)}}$	
$\searrow$	$\nearrow$	

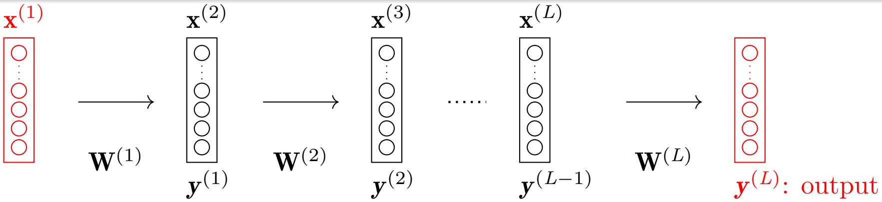
$$l$$

Notations for a multi-layer neural network (feed-forward)

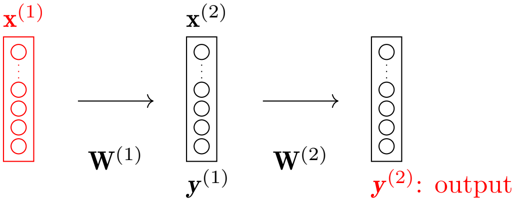
One layer, indexed by l



- $\mathbf{x}^{(l)}$: input of the layer l
- $\boldsymbol{y}^{(l)} = f^{(l)}(\mathbf{W}^{(l)}\ \mathbf{x}^{(l)})$
- stacking layers: $\boldsymbol{y}^{(l)} = \mathbf{x}^{(l+1)}$
- $\mathbf{x}^{(1)}$ = a data example



Example : with one hidden layer



$$\boldsymbol{\theta} = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)})$$

To learn, we need the gradients for:

- the output layer: $\nabla_{\mathbf{W}^{(2)}}$
- the hidden layer: $\nabla_{\mathbf{W}^{(1)}}$

Back-propagation: generalization

For a hidden layer l:

- The gradient at the pre-activation level:

$$\boldsymbol{\delta}^{(l)} = f'^{(l)}(\mathbf{a}^{(l)}) \circ \left(\mathbf{W}^{(l+1)}{}^{t} \boldsymbol{\delta}^{(l+1)}\right)$$

- The update is as follows:

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta_t \boldsymbol{\delta}^{(l)} \mathbf{x}^{(l)}{}^{t}$$

The layer should keep:

- $\mathbf{W}^{(l)}$: the parameters
- $f^{(l)}$: its activation function
- $\mathbf{x}^{(l)}$: its input
- $\mathbf{a}^{(l)}$: its pre-activation associated to the input
- $\boldsymbol{\delta}^{(l)}$: for the update and the back-propagation to the layer $l-1$

Back-propagation: one training step

Pick a training example: $\mathbf{x}^{(1)} = \mathbf{x}_{(i)}$

Forward pass

For $l = 1$ to $(L-1)$

- Compute $\boldsymbol{y}^{(l)} = f^{(l)}(\mathbf{W}^{(l)}\mathbf{x}^{(l)})$
- $\mathbf{x}^{(l+1)} = \boldsymbol{y}^{(l)}$

$\boldsymbol{y}^{(L)} = f^{(L)}(\mathbf{W}^{(L)}\mathbf{x}^{(L)})$

Backward pass

Init: $\boldsymbol{\delta}^{(L)} = \nabla_{\boldsymbol{a}^{(L)}}$
For $l = L$ to 2 // all hidden units

- $\boldsymbol{\delta}^{(l-1)} = f'^{(l-1)}(\boldsymbol{a}^{(l-1)}) \circ \left(\mathbf{W}^{(l)^t}\boldsymbol{\delta}^{(l)}\right)$
- $\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta_t \boldsymbol{\delta}^{(l)}\mathbf{x}^{(l)^t}$

$\mathbf{W}^{(1)} = \mathbf{W}^{(1)} - \eta_t \boldsymbol{\delta}^{(1)}\mathbf{x}^{(1)^t}$

Conclusion on back-propagation for one layer l

Training a NNet relies on forward-backward propagation.

Forward:

- get $\mathbf{x}^{(l)}$ for the previous layer;
- compute and send $\boldsymbol{y}^{(l)} = f^{(l)}(\mathbf{W}^{(l)}\mathbf{x}^{(l)})$.

Backward:

- get $\boldsymbol{\delta}^{(l)}$ as input from the up-coming layer;
- compute and send $\boldsymbol{\delta}^{(l-1)}$ to the previous layer;
- update parameters $\mathbf{W}^{(l)}$

Outline

Summary

Multi-layered Perceptron (MLP) or feed-forward NNet

- Artificial neurons are organizes in **layers**: $\rightarrow$ a vector
- Two layers are in general **fully connected**
  - A linear transformation parametrized by a matrix
  - followed by a pointwise non-linear function, the activation function
- A **feed-forward** architecture is a stack of layers fully connected

$\rightarrow$ Can approximate any functions depending on the number of hidden units.

Training by back-propagation

- After a forward pass (inference from input to the output)
- Backward pass (compute the gradients of each layer from the output to the input)

Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey.
2015.
Predicting the sequence specificities of dna- and rna-binding proteins by deep learning.
Nature Biotechnology, 33(8):831–838.

Dikla Cohn, Or Zuk, and Tommy Kaplan.
2018.
Enhancer identification using transfer and adversarial deep learning of dna sequences.
bioRxiv.

Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen.
2020.
Deep learning for generic object detection: A survey.
International Journal of Computer Vision, 128(2):261–318, Feb.

Xu Min, Wanwen Zeng, Shengquan Chen, Ning Chen, Ting Chen, and Rui Jiang.
2017.
Predicting enhancers with deep convolutional neural networks.
BMC Bioinformatics, 18, 11.

Niksa Praljak, Shamreen Iram, Utku Goreke, Gundeep Singh, Ailis Hill, Umut A. Gurkan, and Michael Hinczewski.
2020.

Integrating deep learning with microfluidics for biophysical classification of sickle red blood cells.
bioRxiv.

Joseph Redmon and Ali Farhadi.
2018.
Yolov3: An incremental improvement.
arXiv.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams.
1986.
Learning representations by back-propagating errors.
Nature, 323(6088):533–536, 10.