Neural networks

Cours 1/3

Machine Learning
ModIA 2023-2024

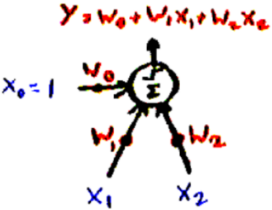
Enseignant: Sixin Zhang
sixin.zhang@toulouse-inp.fr

- *Pattern Recognition and Machine Learning*, Christopher M. Bishop - 2006
- *Deep Learning*, I Goodfellow, Y Bengio, A Courville - 2016
- *Understanding machine learning: From theory to algorithms*, S Shalev-Shwartz, S Ben-David - 2014

Ce cours a été conçu avec Sandrine Mouysset et Axel Carlier.

Perceptron: 1-layer Neural Network (NN)

Historically, Perceptron is a highly simplified neuron model to achieve linear classification (McCulloch et Pitts 1943, Rosenblatt 1957).



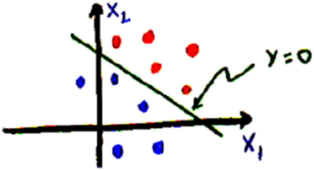
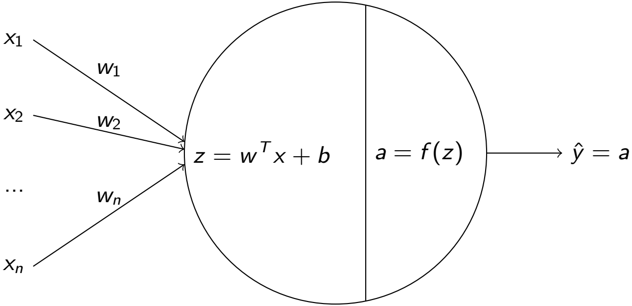


image source: lecture notes of Y. LeCun (NYU)

Representation of Perceptron



1. Inner product between input vector $x \in \mathbb{R}^n$ and the weight w : $w^T x$;
2. Add a bias scalar $(b \in \mathbb{R})$: $z = w^T x + b$
3. Application of an activation function to z : $a = f(z)$
4. Output value $\hat{y} = a$, e.g. $\hat{y} \in \{0, 1\}$ for binary classification.

Activation functions

The **activation functions**, denoted f, are usually non-linear functions. They can play a role of thresholding with 3 regimes,

- non-active: if the input value is under a threshold;
- transition phase: if the input value is close to the threshold;
- active: if the input value is above the threshold;

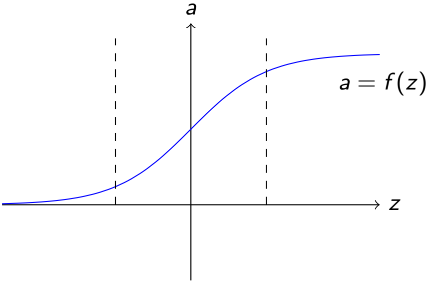
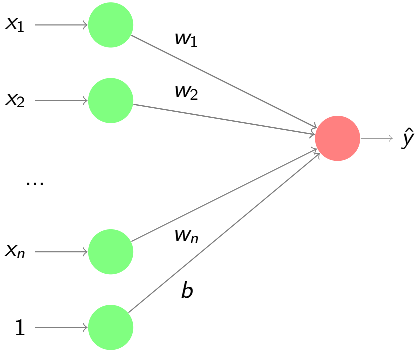


Figure: Sigmoid activation function : $f(z) = \frac{1}{1+e^{-z}}$

Simplify the Perceptron function: convert bias to input weight

Assume input vector: $(x_1, \cdots, x_n, 1)^T \in \mathbb{R}^{n+1}$

$$w^T x + b = (w_1, \cdots, w_n, b)^T (x_1, \cdots, x_n, 1)$$



Therefore: we write $a = f(w^T x)$ instead of $a = f(w^T x + b)$

To achieve binary classification using Perceptron $\hat{y} = f(\sum_i w_i x_i) \in \{0, 1\}$:

- show each training sample (x, y) in sequence repetitively.
- if the output $\hat{y} = y$ is correct, do nothing.
- if the output $\hat{y} = 0$ and the desired output $y = 1$: increase the weights w_i whose inputs x_i are positive, decrease the weights whose inputs are negative.
- if the output $\hat{y} = 1$ and the desired output $y = 0$: decrease the weights whose inputs are positive, increase the weights whose inputs are negative.

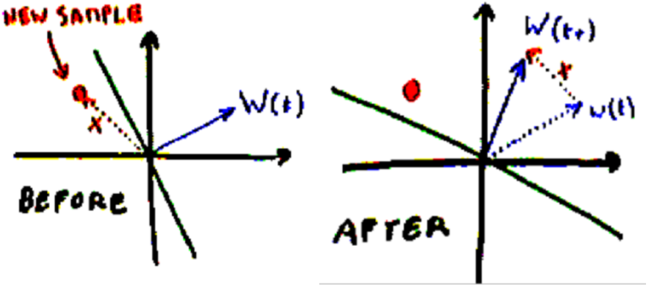


image source: lecture notes of Y. LeCun (NYU)

1. Initial weight $w^{(0)} = (w_i^{(0)})_{i \leq n}$
2. Draw training samples $(x^{\{1\}}, y^{\{1\}}), ..., (x^{\{m\}}, y^{\{m\}})$.
3. Compute the output of Perceptron and a differentiable loss $J(w)$:

$$\hat{y}^{\{j\}}(w) = f\left(\sum_{i=1}^{n} w_i x_i^{\{j\}}\right) \text{ and } J(w) = \frac{1}{m}\sum_{j=1}^{m} \ell(\hat{y}^{\{j\}}(w), y^{\{j\}})$$

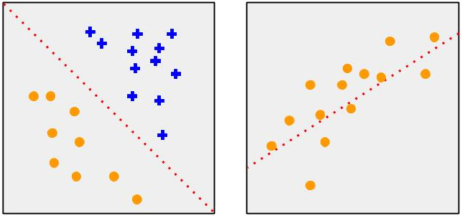
4. Update the weights from $w^{(t)}$ to $w^{(t+1)}$

$$w_i^{(t+1)} = w_i^{(t)} - \alpha^{(t)} \frac{\partial J}{\partial w_i}(w^{(t)})$$

   where $\alpha^{(t)}$ is a step size (learning rate) $\alpha^{(t)} > 0$.
5. Repeat 2-4 until convergence of $w^{(t)}$ or $J(w^{(t)})$.

$\Rightarrow$ How to define the **cost function** ℓ ?

Classification and regression



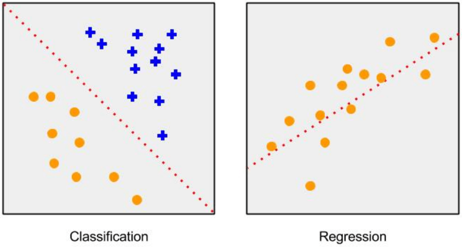
Classification   Regression

Classification (Logistic regression) Assign a category to each observation
Binary case : false/true, $y \in \{0,1\}, \hat{y} \in [0,1]$

- sigmoid activation ($\mathbb{R} \to [0,1]$): $f(z) = (1 + e^{-z})^{-1}$
- Loss function: logistic cost (cross-entropy):

$$\text{loss}(\hat{y}, y) = -y log(\hat{y}) - (1-y) log(1-\hat{y})$$

Classification and regression



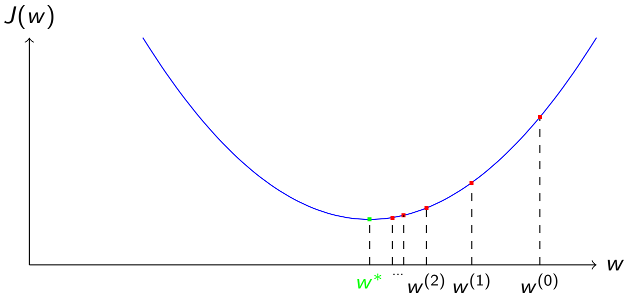
Linear Regression Predict a real value of each observation :

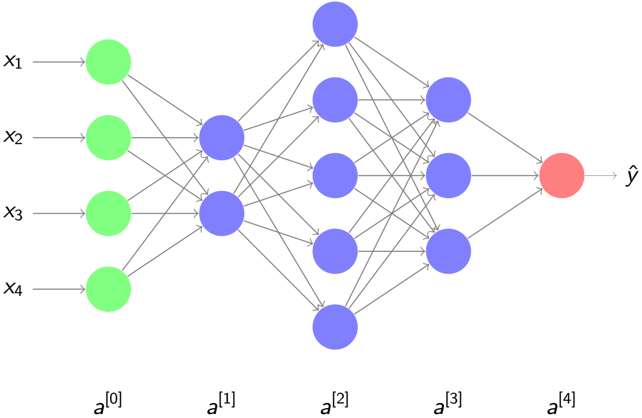
- linear activation : $f(z) = z$
- Mean squared error cost function (MSE):

$$\ell(\hat{y}, y) = (y - \hat{y})^2$$

$\Rightarrow$ How **to solve this type of problem** ?

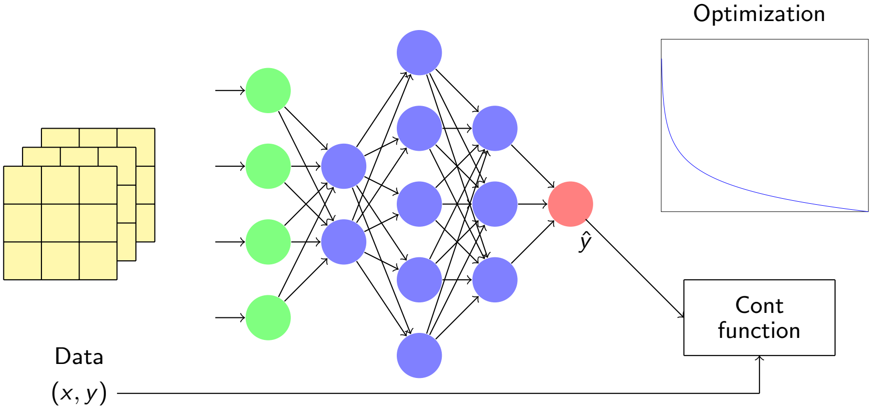
Gradient descent method: iterative method to find an optimal w^*





A multi-layer perceptron (MLP) is composed of an input layer, several hidden layers and an output layer. The hidden layer is usually composed of a linear layer and a non-linear activation function.
The **depth** of the network above is $L = 4$ (3 hidden layers plus one output layer).

Supervised learning framework

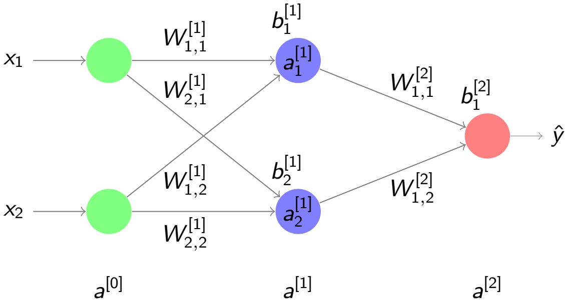


Multi-layer perceptron :

1. Functionality
2. Interpretation
3. Activation function
4. Multi-class classification loss

In order to train a multi-layer perceptron, we need to understand the following computational steps:

1. **Forward propagation** of input data to output;
2. Compute a **loss** from the output;
3. **Back propagation**: compute **gradients** of the loss with respect to the weights of the **output layer** and **hidden layers**;
4. **Update** all the weights based on optimization methods.



The weights of layer k: $W^{[k]}_{i,j}$ and $b^{[k]}_i$, i output index, j input index. For depth L, we denote all the weights by $\theta = (W^{[k]}, b^{[k]})_{k \leq L}$, e.g. $L = 2$

$$\hat{y}(x, \theta) = f \circ f^{[2]} \left(W^{[2]} f^{[1]}(W^{[1]} x + b^{[1]}) + b^{[2]} \right)$$

For an input $x^{\{i\}}$, we write the output $\hat{y}^{\{i\}}(\theta) = \hat{y}(x^{\{i\}}, \theta)$

2) Compute the objective function after the forward-propogation:

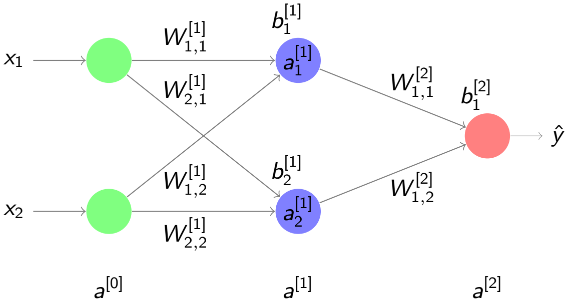
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(y^{\{i\}}, \hat{y}^{\{i\}}(\theta))$$

3) Back-propagation: to compute the gradients $\nabla_\theta J = (\frac{\partial J}{\partial \theta})^\intercal$ from output to input by the *chain rule* in Calculus, e.g.

$$\nabla_\theta J = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{\partial \hat{y}^{\{i\}}}{\partial \theta} \right)^\intercal \nabla_{\hat{y}^{\{i\}}} \ell(y^{\{i\}}, \hat{y}^{\{i\}})$$

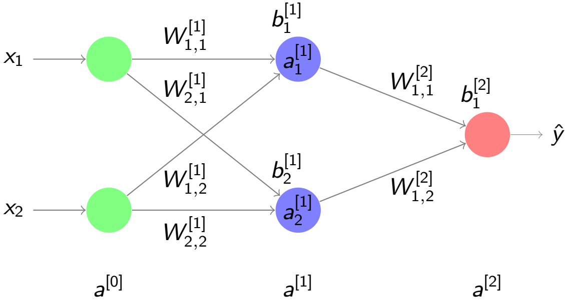
- Step 1: compute $\nabla_{\hat{y}^{\{i\}}} \ell(y^{\{i\}}, \hat{y}^{\{i\}})$ for $1 \leq i \leq m$.
- Step 2: compute $\frac{\partial \hat{y}^{\{i\}}}{\partial \theta}$ for $1 \leq i \leq m$.
- Step 3: compute $\nabla_\theta J$.

Step 1 can be solved analytically when ℓ is differentiable. How about Step 2?



Assume $\hat{y} = f(a^{[2]}) \in \mathbb{R}$, $a^{[2]} = f^{[2]}(W_{1,1}^{[2]} a_1^{[1]} + W_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]}) \in \mathbb{R}$.
Compute $\frac{\partial \hat{y}}{\partial \theta}$:

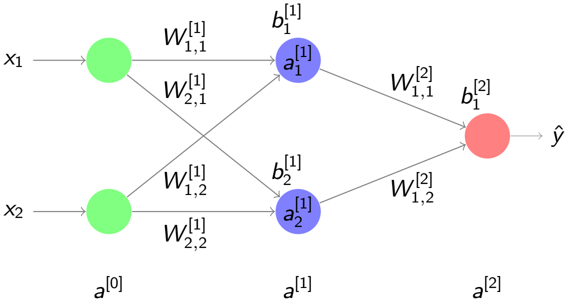
$$\frac{\partial \hat{y}}{\partial b_1^{[2]}} = \frac{\partial \hat{y}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial b_1^{[2]}} = f'(a^{[2]}) f^{[2]'}(W_{1,1}^{[2]} a_1^{[1]} + W_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]})$$



Assume $\hat{y} = f(a^{[2]}) \in \mathbb{R}$, $a^{[2]} = f^{[2]}(W^{[2]}_{1,1}a^{[1]}_1 + W^{[2]}_{1,2}a^{[1]}_2 + b^{[2]}_1) \in \mathbb{R}$.
Compute $\frac{\partial \hat{y}}{\partial \theta}$:

$$\frac{\partial \hat{y}}{\partial W^{[2]}_{1,1}} = \frac{\partial \hat{y}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial W^{[2]}_{1,1}} = f'(a^{[2]}) f^{[2]'}(W^{[2]}_{1,1}a^{[1]}_1 + W^{[2]}_{1,2}a^{[1]}_2 + b^{[2]}_1) a^{[1]}_1$$

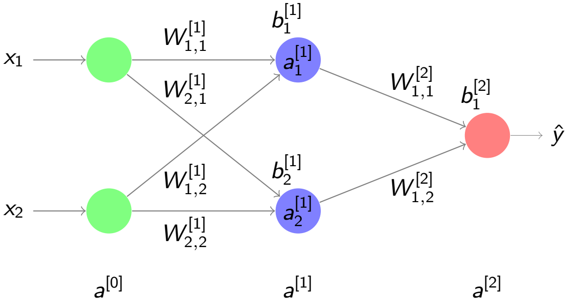
Illustration of back propagation



Assume $\hat{y} = f(a^{[2]}) \in \mathbb{R}$, $a^{[2]} = f^{[2]}(W^{[2]}_{1,1}a^{[1]}_1 + W^{[2]}_{1,2}a^{[1]}_2 + b^{[2]}_1) \in \mathbb{R}$.
Compute $\frac{\partial \hat{y}}{\partial \theta}$: Assume $a^{[1]} = f^{[1]}(W^{[1]}a^{[0]} + b^{[1]})$

$$\frac{\partial \hat{y}}{\partial W^{[1]}_{i,j}} = \frac{\partial \hat{y}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial W^{[1]}_{i,j}}$$

Jacobian matrices: $\frac{\partial a^{[1]}}{\partial W^{[1]}_{i,j}} : \mathbb{R}^1 \to \mathbb{R}^2$ $\frac{\partial a^{[2]}}{\partial a^{[1]}} : \mathbb{R}^2 \to \mathbb{R}^1$, $\frac{\partial \hat{y}}{\partial a^{[2]}} : \mathbb{R} \to \mathbb{R}$,



Assume $\hat{y} = f(a^{[2]}) \in \mathbb{R}$, $a^{[2]} = f^{[2]}(W^{[2]}_{1,1}a^{[1]}_1 + W^{[2]}_{1,2}a^{[1]}_2 + b^{[2]}_1) \in \mathbb{R}$.
Compute $\frac{\partial \hat{y}}{\partial \theta}$: Assume $a^{[1]} = f^{[1]}(W^{[1]}a^{[0]} + b^{[1]})$

$$\frac{\partial a^{[2]}}{\partial a^{[1]}_1} = f^{[2]\prime}(W^{[2]}_{1,1}a^{[1]}_1 + W^{[2]}_{1,2}a^{[1]}_2 + b^{[2]}_1)W^{[2]}_{1,1}$$

$$\frac{\partial {a_i}^{[1]}}{\partial {W_{i,j}}^{[1]}} = f_i^{[1]\prime}(W^{[1]}a^{[0]} + b^{[1]})a^{[0]}_j, \quad i,j \in \{1,2\}$$

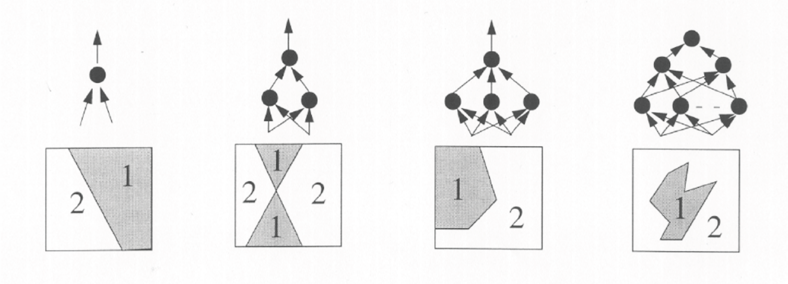
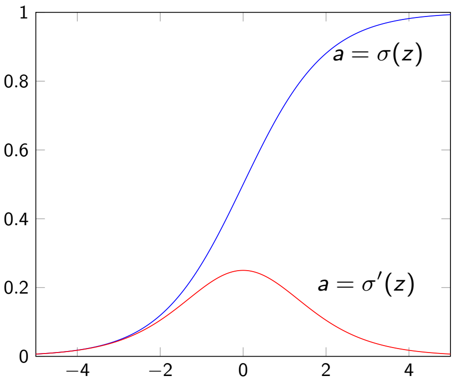


Figure: Linear vs. non-linear separation of training data

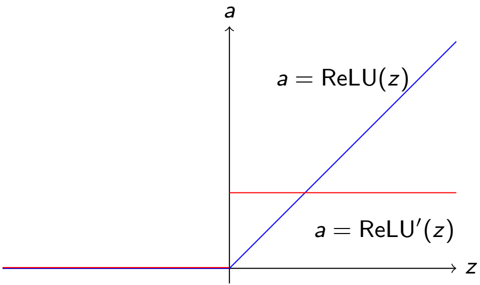
The non-linearity $f^{[1]}, f^{[2]}, \cdots$ in MLP plays a key role for non-linear separation.

Example: `https://playground.tensorflow.org/`

Activation functions: sigmoid

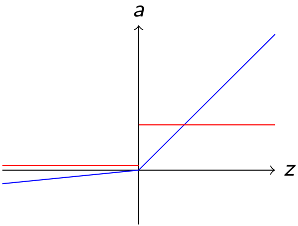


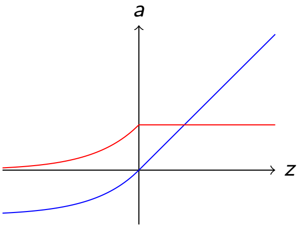
- The gradient function tends to zero when z is away from 0: cause vanishing gradients in the back propagation.



$$\mathrm{ReLU}(z) = \begin{cases} z & \text{if} \quad z > 0 \\ 0 & \text{if} \quad z \leq 0 \end{cases}$$

The gradient is either 0 or 1 (when $z \neq 0$), very different to that of the sigmoid.





$$\text{leakyReLU}(z) = \begin{cases} \alpha z & \text{if } z < 0 \\ z & \text{if not} \end{cases}$$

Leaky Rectified Linear Unit

$$\text{eLU}(z) = \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if not} \end{cases}$$

Exponential Linear Unit

These functions could potentially improve the gradient-descent training, e.g. to achieve a faster convergence.

- Question 1: Assume $y \in \{1, 2, \cdots, C\}$. To classify x into C categories, how to design a **differentiable** loss $\ell(y, \hat{y})$?
- Question 2: Assume $\hat{y}$ is a probability distribution over $\{1, 2, \cdots, C\}$, how to compute it as the output of an MLP?

- Answer 1: Use the cross-entropy loss by representing y and $\hat{y}$ as a probability distribution over $\{1, 2, \cdots, C\}$.
- Answer 2: Use the Softmax non-linear function f so that $\hat{y} = f(a)$.

Cross-entropy loss

- KL divergence between two distributions p and q over $\{1, \cdots, C\}$

$$KL(q||p) = \sum_{i=1}^{C} \log \frac{q_i}{p_i} q_i$$

- Let y be a vector in $\{0, 1\}^C$ such that $y_i = 1$ i.f.f the category of y is i.
- Let $\hat{y}$ be a vector in $(0, 1)^C$ such that $\sum_i \hat{y}_i = 1$.
- The cross-entropy loss is $KL(y||\hat{y})$, which is equivalent to

$$\sum_{i=1}^{C} \log(y_i) y_i - \sum_{i=1}^{C} \log(\hat{y}_i) y_i$$

- In practice, we minimize the second term (to optimize MLP):

$$-\sum_{i=1}^{C} \log(\hat{y}_i) y_i$$

Softmax non-linearity

- Let $a \in \mathbb{R}^C$ and $\hat{y} = f(a)$, such that

$$\hat{y}_i = \frac{e^{a_i}}{\sum_k e^{a_k}},$$

  - We have $\hat{y}_i \geq 0$ and $\sum_i \hat{y}_i = 1$.
- **Property**: for any $c \in \mathbb{R}$ and $a \in \mathbb{R}^C$, $f(a + c) = f(a)$.
- To avoid numerical issues, we compute $f(a + c)$ with $c = -\max_j a_j$,

$$\hat{y}_i = \frac{e^{a_i - \max_j a_j}}{\sum_k e^{a_k - \max_j a_j}}.$$