Fundamentals of Machine Learning

Simon BERNARD (simon.bernard@univ-rouen.fr)
LITIS lab. @ University Rouen Normandy

Images and videos generation/manipulation (Dall-E, Midjourney, Stable diffusion, etc.)

Figure 1 – Source : Samsung MegaPortraits (`https://www.youtube.com/watch?app=desktop&v=zUJ1BpZnwQo`)

Natural Language Processing (DeepL, ChatGPT, Gemini, Copilot, etc.)

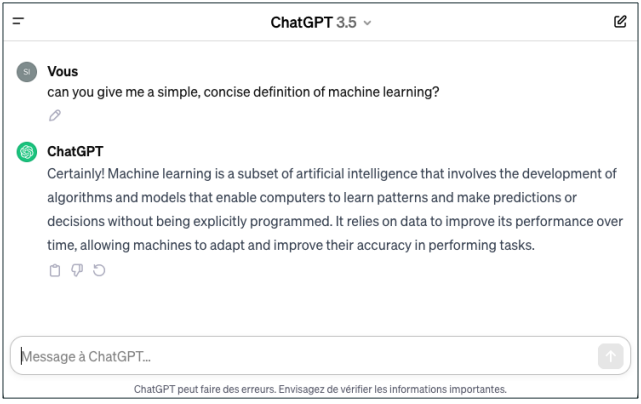


Figure 2 – Source : ChatGPT (`https://chatgpt.com/`)

Autonomous AI (DeepMind AlphaGo, Tesla Autopilot, Boston Dynamics, etc.)

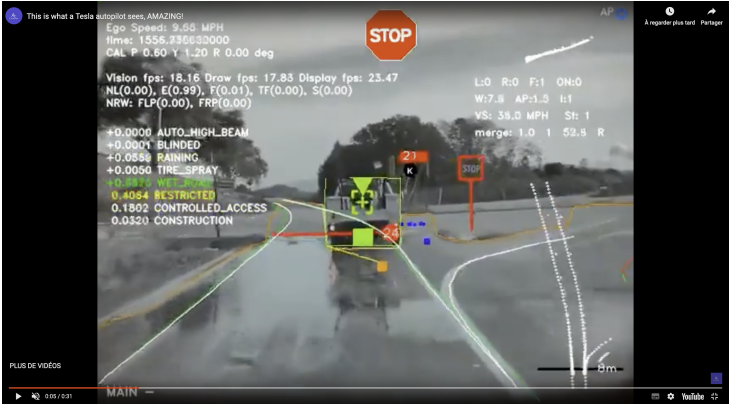
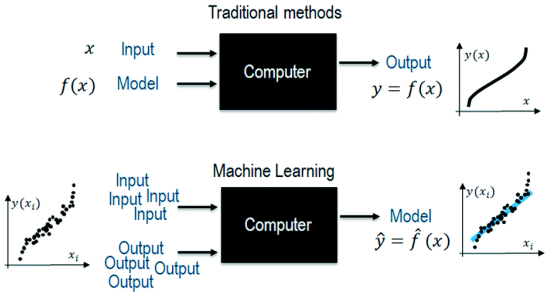


Figure 3 – Source : Tesla Autopilot (`https://www.youtube.com/watch?v=fuCEB_QQDso&t=5s`)

Machine Learning

Making a machine capable of carrying out complex tasks without having been explicitly programmed to do so, but based on experience, using example data.



Machine Learning

Making a machine capable of carrying out complex tasks without having been explicitly programmed to do so, but based on experience, using example data.

cats

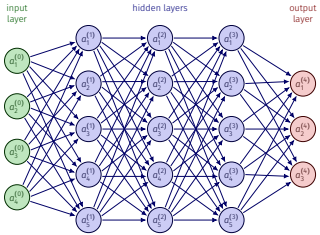



birds



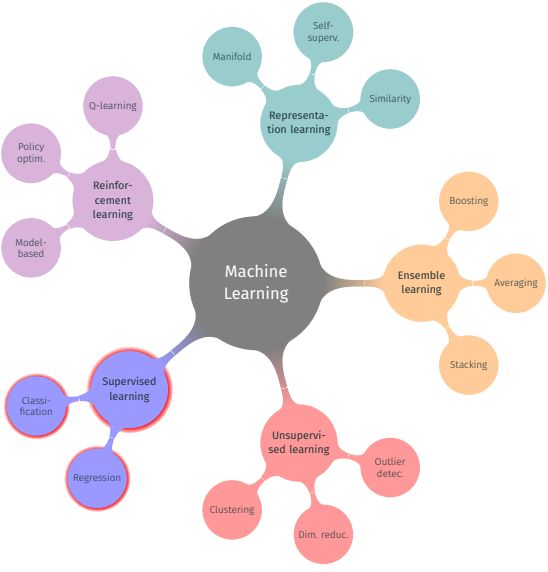

Today's AI ≈ Deep Learning

- Deep Neural Networks
  - Universal approximation theorem
  - "deep" → a LOT of parameters
- Representation learning
  - Convolutional / reccurent Networks
  - Attention mecanism
  - etc.
- Big Data × Computing power



Supervised Learning

Machine Learning



- Machine Learning is based on example data
- These data represent 'objects' of the problem (e.g. patients, images, texts, etc.).
- A data item (instance or sample) is described by d values (features)

$$\mathbf{x} = \{x^{(1)}, x^{(2)}, \ldots, x^{(d)}\}, \mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$$

- The features are a description of objects and often reflect properties of interest
- A training set is a set of n training instances

$$\mathbf{X} = \{\mathbf{x}_i \in \mathcal{X}, i = 1, \ldots, n\}$$

- The data is associated with a target variable that we wish to predict
- It is assumed that there is a true function f (unknown) :

$$f : \mathcal{X} \to \mathcal{Y}$$

where $\mathcal{Y}$ is the domain of the target variable Y

- Learning consists in finding a prediction function (or model) h that best approximate f :

$$\hat{y} = h(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$$

where $\hat{y}$ is the prediction of Y for $\mathbf{x}$

- Supervised learning : the training set is made up with $(\mathbf{x}_i, y_i)$:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, n\}$$

  where y_i is the true value of Y for the corresponding $\mathbf{x}_i$.

- 2 types of task with respect to the nature of $\mathcal{Y}$:

  1. Classification : $\mathcal{Y}$ is a finite set of c classes

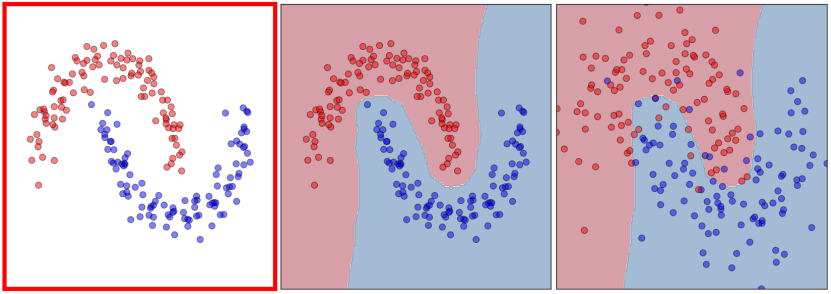
$$\mathcal{Y} = \{\lambda_1, \lambda_2, \ldots, \lambda_c\}, c \geq 2$$

  2. Regression : $\mathcal{Y}$ is a real value

$$\mathcal{Y} \subset \mathbb{R}$$

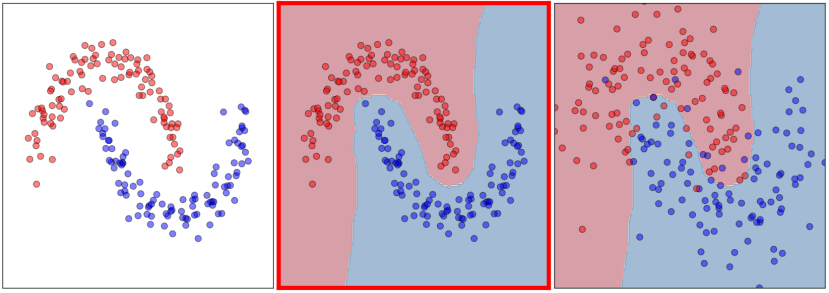
1. Training set, composed of n labeled instances (i.e. we know the *true* labels) :

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, n\}$$



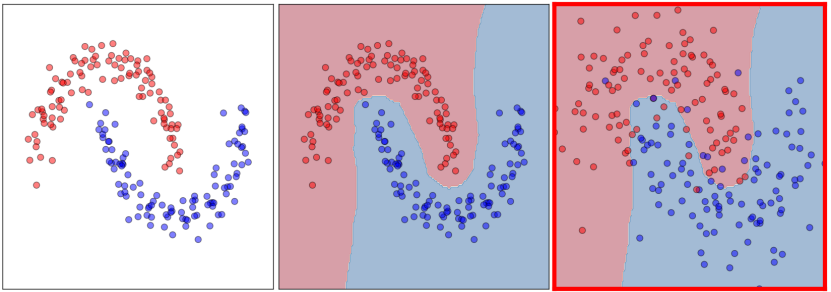
2. Model h learned on these data. Here, $\mathcal{Y} = \{red, blue\}$.

$$h : \mathcal{X} \to \mathcal{Y}$$



3. *h* predict "red" or "blue" for each new instance **x**

$$\hat{y} = h(\mathbf{x}),\ \forall\, \mathbf{x} \notin \mathcal{D}$$



- Which ML method for learning the best h ?

$$h^\star = \underset{h \in \mathcal{H}}{\arg\min}\, P_{\mathcal{X},\mathcal{Y}}\left(h(\mathbf{x}) \neq y\right)$$

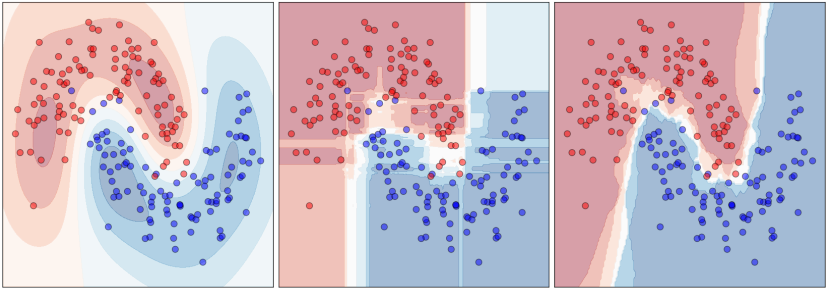


Figure 4 – SVM (RBF kernel), Random Forest (100 trees), K-Nearest Neighbors (K=5)

- The goal of ML methods is to find the best h possible
- For that purpose, we use a loss function

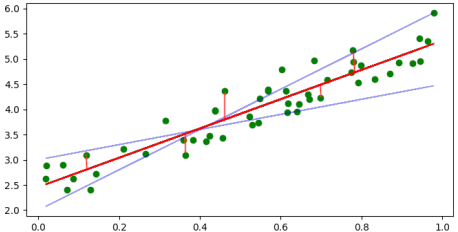
$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$$

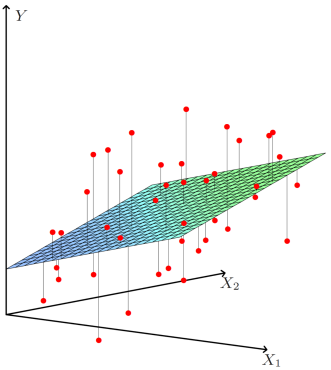
  so that $L(y, h(\mathbf{x})) \geq 0$ measure the extent to which the prediction $h(\mathbf{x})$ is close to y
- The global performance of a model is called true risk or generalization error :

$$R(h) = E_{X,Y}[L(Y, h(X))]$$

- The best model is the one that minimizes the true risk

Example : linear regression with least squares (quadratic loss)





Empirical risk minimization

- Problem : we cannot calculate $R(h)$
- But, we have a training set to calculate an estimation, the empirical risk :

$$R_{\mathcal{D}}(h) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} L(y_i, h(\mathbf{x}_i))$$

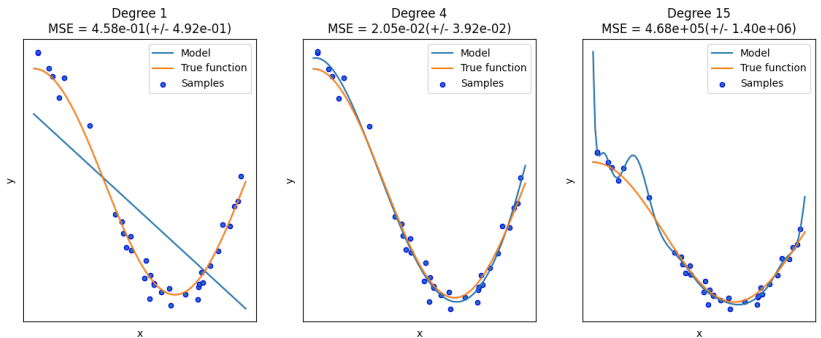
- Empirical risk minimization consists in finding $h^\star_{\mathcal{D}}$ such that :

$$h^\star_{\mathcal{D}} = \underset{h \in \mathcal{H}}{\arg\min}\, R_{\mathcal{D}}(h)$$

- A machine learning technique is a (hopefully effective) way of finding $h^\star_{\mathcal{D}}$ from $\mathcal{D}$

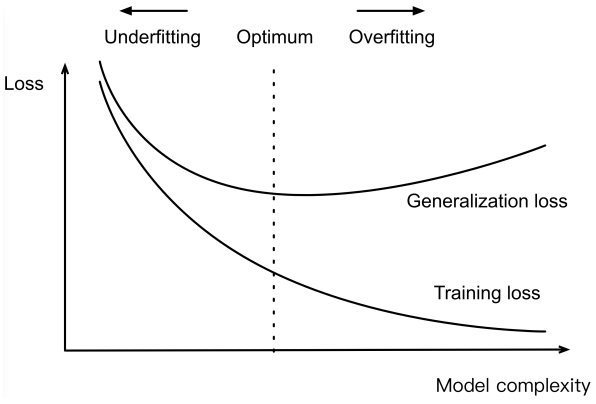
Under-fitting and Over-fitting

- What happens if h is too "simple" (too few parameters) or too "complex" (too many parameters) for the problem ?



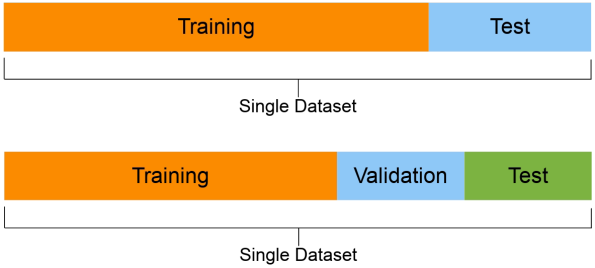
Under-fitting and Over-fitting

- What happens if h is too "simple" (too few parameters) or too "complex" (too many parameters) for the problem ?



Detecting and avoiding overfitting

- Estimate the generalization error by using an independent data subset
- Golden rule : the test set should be used only ONCE and must NOT be used for anything else



Linear models

Example : Predicting the progression of diabetes from clinical data[1]

	AGE	SEX	BMI	BP	⋯	Serum Measurements				⋯	Response
Patient	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	y
1	59	2	32.1	101	157	93.2	38	4	4.9	87	151
2	48	1	21.6	87	183	103.2	70	3	3.9	69	75
3	72	2	30.5	93	156	93.6	41	4	4.7	85	141
4	24	1	25.3	84	198	131.4	40	5	4.9	89	206
5	50	1	23.0	101	192	125.4	52	4	4.3	80	135
6	23	1	22.6	89	139	64.8	61	2	4.2	68	97
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
441	36	1	30.0	95	201	125.2	42	5	5.1	85	220
442	36	1	19.6	71	250	133.2	97	3	4.6	92	57

Table 1. Diabetes study. 442 diabetes patients were measured on 10 baseline variables. A prediction model was desired for the response variable, a measure of disease progression one year after baseline.

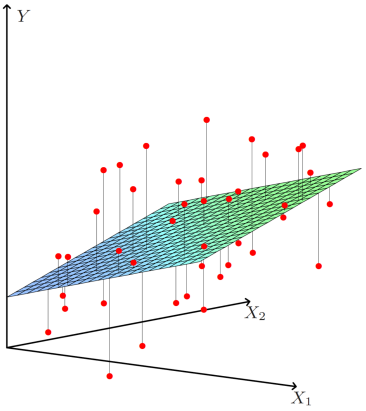
1. Bradley Efron et al., "Least Angle Regression", Annals of Statistics, 2004

- Linear model :

$$h(\mathbf{x}) = \sum_{i=1}^{n} w_i x^{(i)} + b$$
$$= \mathbf{w}^T \mathbf{x}$$

- Learning with least squares :

$$L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$



- Mathematical optimization techniques gives us the (closed-form) solution :

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

where $\mathbf{X}$ is a $n \times (d+1)$ matrix s.t. $\mathbf{X}_{i,j} = x_i^{(j)}$ and $\mathbf{X}_{i,d+1} = 1$

```
import numpy as np
from sklearn import datasets

X, y = datasets.load_diabetes(scaled=False, return_X_y=True)
Xa = np.concatenate((X, np.ones((X.shape[0], 1))), axis=1)
w = np.linalg.inv(Xa.T @ Xa) @ Xa.T @ y
```

- Problem : $(\mathbf{X}^\top \mathbf{X})^{-1}$ can not be calculated if $d > n$
- Solution : regularization, i.e. add a constraint on the parameters in the loss :

$$L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \lambda \Omega(\mathbf{w})$$

- Bonus : help to avoid overfitting

- The most popular regularized linear regression method is the ridge regression :

$$L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \lambda \sum_{j=1}^{d} w_j^2$$

- The closed-form solution is :

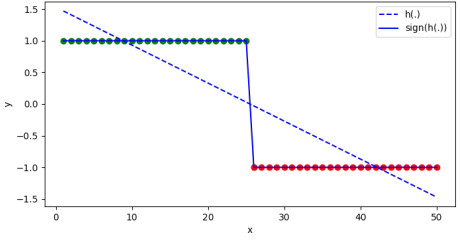
$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{S})^{-1}\mathbf{X}^T\mathbf{y}$$

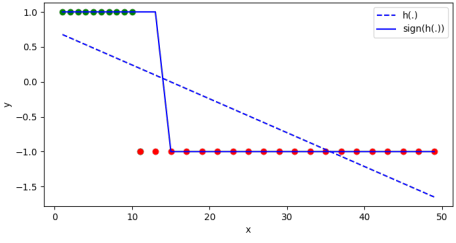
with S a $(d+1) \times (d+1)$ matrix s.t. $S_{i,j} = 1$ if $i = j$ and $i \leq d$, 0 otherwise

```
lam = 0.1   # arbitrary value; should be tuned
d = Xa.shape[1]
S = np.identity(d)
S[d - 1, d - 1] = 0
w = np.linalg.inv(Xa.T @ Xa + lam * S) @ Xa.T @ y
```

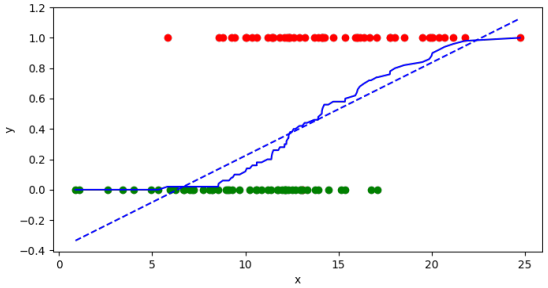
Linear classification

- Linear model on binary classification problems ($y = 1$ or $y = -1$)
- Least squares + the predicted class is given by the sign of $h(.)$
- Loss function becomes meaningless and results not always relevant



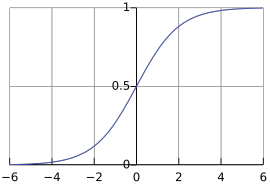


- Alternative : predict the probability of belonging to a class : $p(y = 1|\mathbf{x})$
- Prediction given by $p(y = 1|\mathbf{x}) \underset{0}{\overset{1}{\gtrless}} p(y = 0|\mathbf{x})$
- Linear models are not suitable for modelling probabilities



- Logistic regression : use of the logistic function or sigmoid :

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



- Logistic regression model :

$$h(\mathbf{x}) = p(y = 1|\mathbf{x}) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right)$$

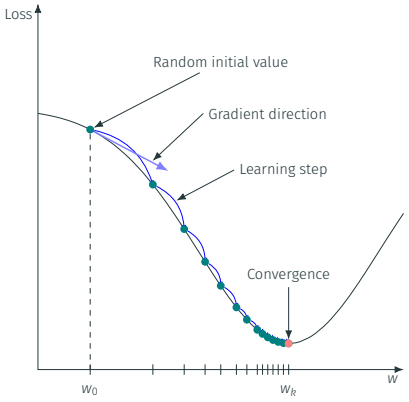
Logistic regression with gradient descent

- Parameter learning is based on the binary cross-entropy loss function :

$$\begin{aligned} LL(\mathbf{w}) &= -\sum_{i=1}^{n} y_i \log(h(\mathbf{x}_i)) + (1-y_i)\log(1-h(\mathbf{x}_i)) \\ &= -\sum_{i=1}^{n} y_i \log(\sigma(\mathbf{x}_i^\top \mathbf{w})) + (1-y_i)\log(1-\sigma(\mathbf{x}_i^\top \mathbf{w})) \end{aligned}$$

  with $y_i \in \{0,1\}$ the class of the i-th sample
- We'll spare you the math - it does not result in a closed-form solution
- Solution : iterative numerical method called gradient descent

Gradient descent



1. initialise $\mathbf{w}_0$ with random values
2. Calculate the gradient $\nabla LL(\mathbf{w}_0)$
3. Update the parameters using the gradient :

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla LL(\mathbf{w}_k)$$

with η the learning rate (hyperparameter)

4. Repeat steps 3 until convergence

- Empirical risk minimization setup :

$$LL(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, h_{\mathbf{w}}(\mathbf{x}_i))$$

$$\nabla LL(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\mathbf{w}} L(y_i, h_{\mathbf{w}}(\mathbf{x}_i))$$

- Therefore, in batch gradient descent, the complexity of an update grows linearly with n
- Bad idea when dealing with large datasets

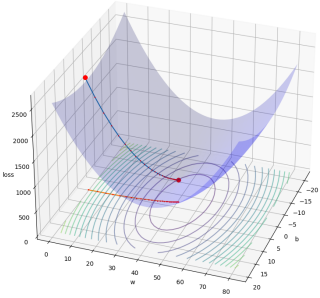
Logistic regression with gradient descent

- Solution : Stochastic Gradient Descent (SGD)

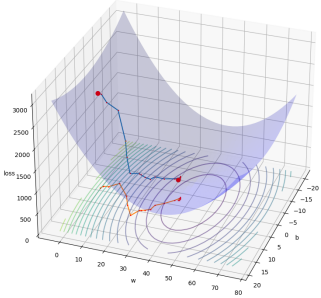
$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla LL(\mathbf{w}_k)$$
$$LL(\mathbf{w}_k) = L(y_{rand}, h_{\mathbf{w}_k}(\mathbf{x}_{rand}))$$

- *rand* is a training instance index picked randomly at each iteration
- Alternative : update on a small subset of the data (Mini-batch Gradient Descent)

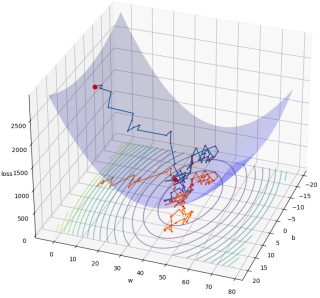
Logistic regression with gradient descent



Batch gradient descent

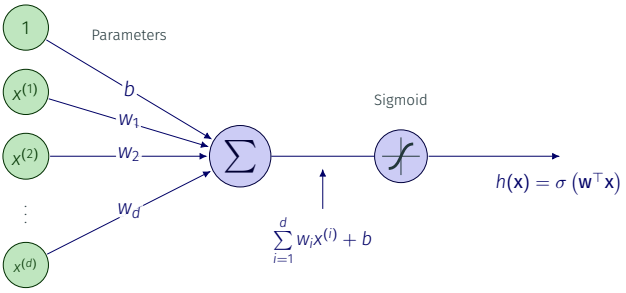


Mini-batch gradient descent

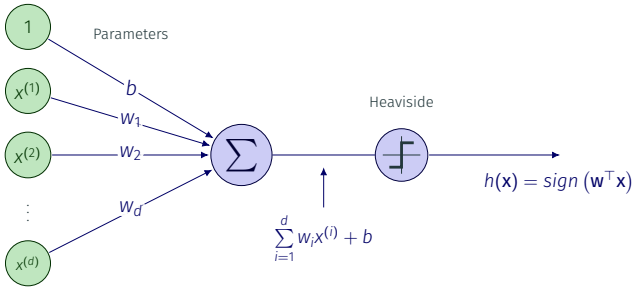


Stochastic gradient descent

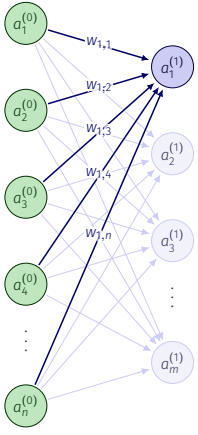
Logistic regression :



The Perceptron (Rosenblatt, 1958) :



Multi-output Perceptron (Neural Network) :



$$= \sigma\left(\sum_{i=1}^{n} w_{1,i} a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{pmatrix} \begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{a}^{(0)} + \mathbf{b}^{(0)}\right)$$

Neural Networks :