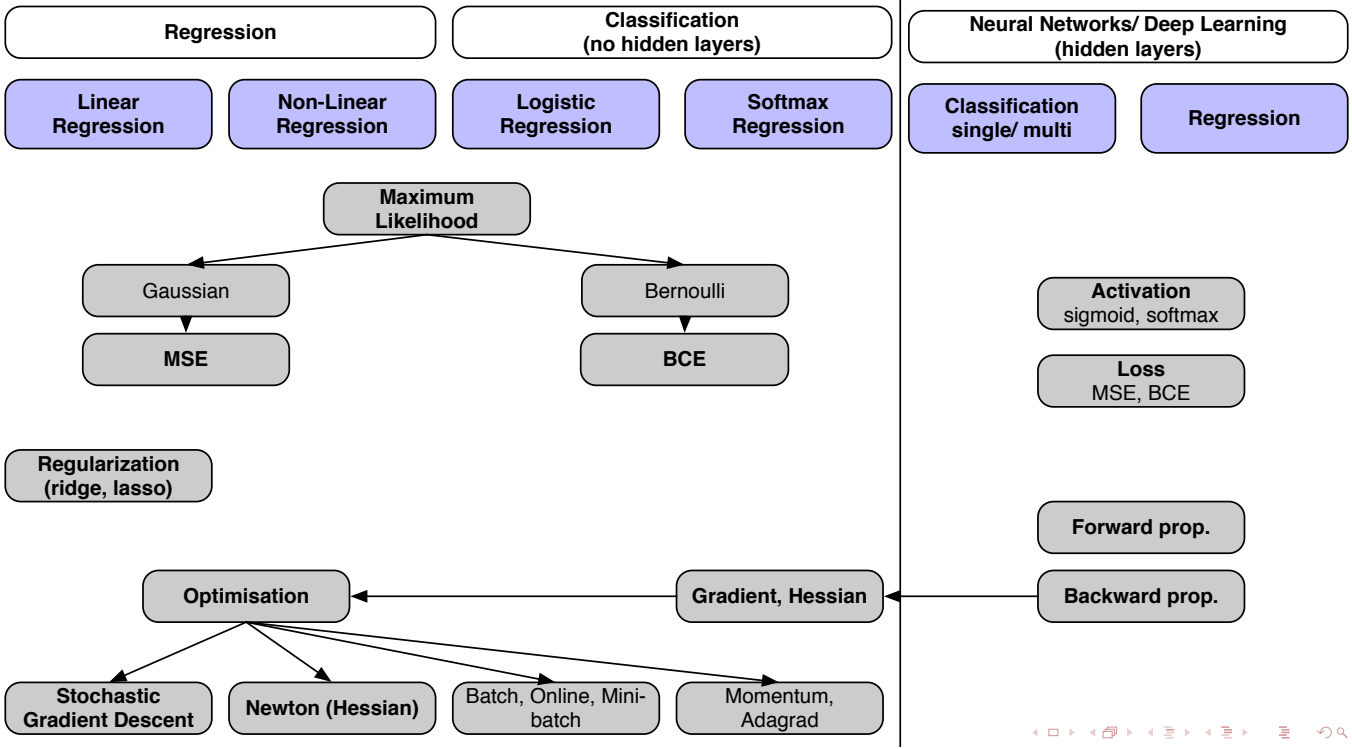
Introduction to Deep Learning
Topics: Multi-layer-Perceptron

Geoffroy Peeters

LTCI, Télécom Paris, IP Paris

-

Overview


Regression
Classification
(no hidden layers)
Neural Networks/ Deep Learning
(hidden layers)
Linear
Regression
Non-Linear
Regression
Logistic
Regression
Softmax
Regression
Classification
single/ multi
Regression
Maximum
Likelihood
Gaussian
Bernoulli
MSE
BCE
Activation
sigmoid, softmax
Loss
MSE, BCE
Regularization
(ridge, lasso)
Forward prop.
Optimisation
Gradient, Hessian
Backward prop.
Stochastic
Gradient Descent
Newton (Hessian)
Batch, Online, Mini-
batch
Momentum,
Adagrad

Supervised classification

Training/Learning

- Given a set of m input/output examples :
  - $\underline{x}^{(1:m)} = \{\underline{x}^{(1)}, \underline{x}^{(2)}, \ldots, \underline{x}^{(m)}\}$
    - with $\underline{x}^{(i)} \in \mathbb{R}^d$
  - $y^{(1:m)}$
    - $y^{(i)} \in \mathbb{R}$ (regression)
    - $y^{(i)} \in \{0, 1\}$ (binary classification)
    - $\underline{y}^{(i)} \in \{0, 1\}^C$ (multi-classes classification)
- Estimate the parameters $\underline{\theta}$ of the model :

$$\{x^{(1:m)}, y^{(1:m)}\} \rightarrow \boxed{Learner} \rightarrow \underline{\theta}$$

Testing/ Prediction

- Given a new input $x^{(m+1)}$ and the parameters $\underline{\theta}$ of the model
- Predict the output $y^{(m+1)}$

$$\{x^{(m+1)}, \underline{\theta}\} \rightarrow \boxed{Prediction} \rightarrow \hat{y}^{(m+1)}$$

... 1 dimension

Model

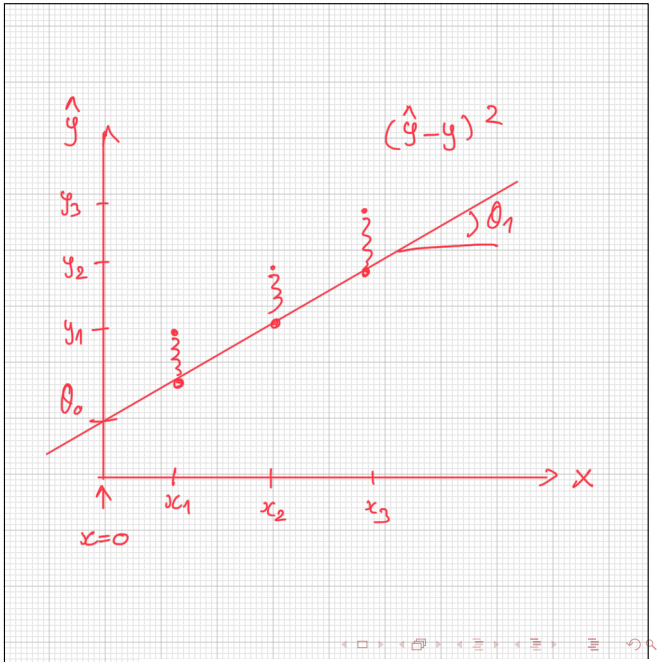
$$\hat{y}^{(i)} = \theta_0 + x^{(i)}\theta_1$$

Objective function (energy, loss) :

- Mean Square Error (MSE)

$$J(\theta) = \sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

$$= \sum_{i=1}^{m}(y^{(i)} - (\theta_0 + x^{(i)}\theta_1))^2$$



... d dimensions

Model :

$$\hat{y}^{(i)} = \theta_0 + \sum_{j=1}^{d} x_j^{(i)} \theta_j$$

$$= \theta_0 + x_1^{(i)} \theta_1 + x_2^{(i)} \theta_2 + \ldots + x_d^{(i)} \theta_d$$

- If we note $x_0^{(i)} = 1$, then

$$\hat{y}^{(i)} = \sum_{j=0}^{d} x_j^{(i)} \theta_j$$

- In matrix form

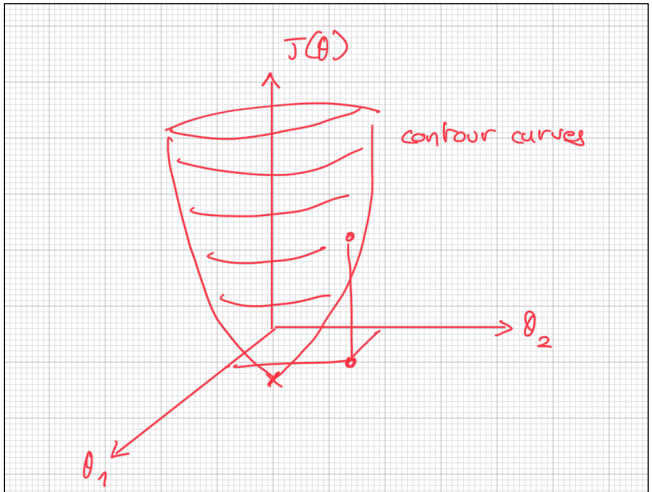
$$\underset{(\mathbb{R}^{(m,\mathbf{1})})}{\underline{\hat{y}}} = \underset{(\mathbb{R}^{(m,d+\mathbf{1})})}{\underline{\underline{X}}} \; \underset{(\mathbb{R}^{(d+\mathbf{1},\mathbf{1})})}{\underline{\theta}}$$

$$\begin{bmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} \cdots x_d^{(1)} \\ \vdots \ddots \vdots \\ x_0^{(m)} \cdots x_d^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_d \end{bmatrix}$$

Objective function

$$J(\underline{\theta}) = \sum_{i=1}^{m} (y^{(i)} - \underline{x}^{(i)\,T} \underline{\theta})^2$$

$$= \boxed{(\underline{y} - \underline{\underline{X}}\,\underline{\theta})^T (\underline{y} - \underline{\underline{X}}\,\underline{\theta})}$$



Parameter estimation

Find $\underline{\theta}$ that minimizes $J(\underline{\theta})$

$$\frac{\partial J(\underline{\theta})}{\partial \underline{\theta}} = 0 = \frac{\partial}{\partial \underline{\theta}}(\underline{y} - \underline{\underline{X}}\,\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\,\underline{\theta})$$

$$= \frac{\partial}{\partial \underline{\theta}}(\underline{y}^T\underline{y} - 2\underline{y}^T\underline{\underline{X}}\,\underline{\theta} + \underline{\theta}^T\underline{\underline{X}}^T\underline{\underline{X}}\,\underline{\theta})$$

$$= 0 - 2\underline{\underline{X}}^T\underline{y} + 2\underline{\underline{X}}^T\underline{\underline{X}}\underline{\theta}$$

$$\underline{\theta} = \boxed{(\underline{\underline{X}}^T\underline{\underline{X}})^{-1}\underline{\underline{X}}^T\underline{y}}$$

Results from matrix differentiation

$$\frac{\partial A\theta}{\partial \theta} = A^T$$

$$\frac{\partial \theta^T A\theta}{\partial \theta} = 2A^T\theta$$

Reminder from probability

Univariate Gaussian ditribution

Probability Density Function (pdf)

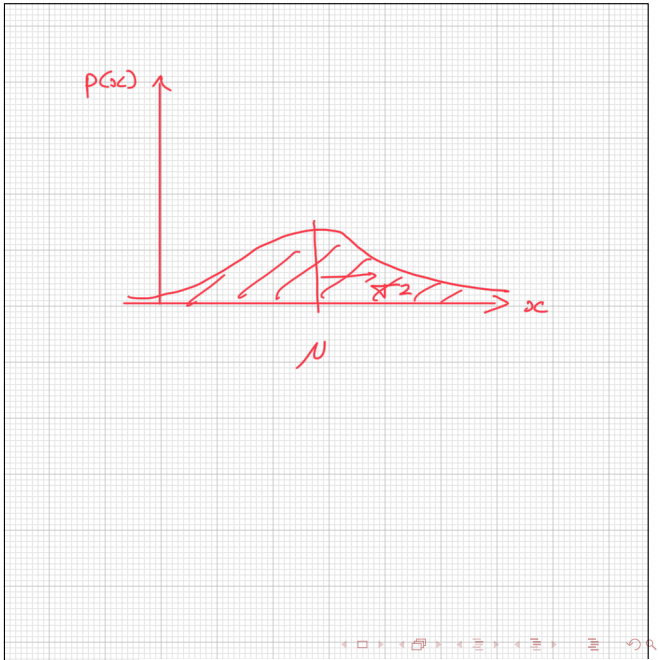
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Parameters (sufficient statistics)

- μ : mean/ center of mass
- σ^2 : variance

Properties

$$\int_{-\infty}^{+\infty} p(x)dx = 1$$



Expectation

Expectation

$$\mathbb{E}(X) = \int x\, p(x) dx = \mu \approx \frac{1}{m} \sum_{i=1}^{m} X^{(i)}$$

- When $\underline{X}$ has d dimensions

$$\underline{\mu} = \mathbb{E}(\underline{X}) = \begin{bmatrix} \mu_{x_1} \\ \mu_{x_2} \\ \vdots \\ \mu_{x_d} \end{bmatrix}$$

$$p(x) = \frac{1}{m} \sum_{i=1}^{m} \delta(x - x^{(i)})$$

$$\int x \frac{1}{m} \sum_{i=1}^{m} \delta(x - x^{(i)})\, dx$$

$$= \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

Covariance

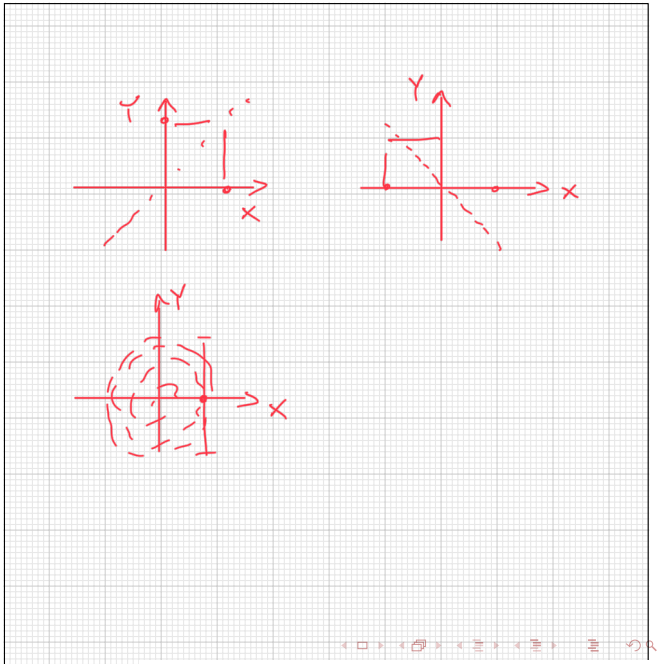
Covariance

$$cov(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$
$$= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$$

- When $\underline{X}$ has d dimensions

$$\underline{\underline{\Sigma}} = cov[\underline{X}]$$
$$= \mathbb{E}[(\underline{X} - \mathbb{E}[\underline{X}])(\underline{X} - \mathbb{E}[\underline{X}])]$$
$$= \begin{pmatrix} var(X_1) & cov(X_1,X_2) & \cdots & cov(X_1,X_d) \\ cov(X_2,X_1) & var(X_2) & \cdots & cov(X_2,X_d) \\ \vdots & \vdots & \ddots & \vdots \\ cov(X_d,X_1) & \cdots & \cdots & var(X_d) \end{pmatrix}$$

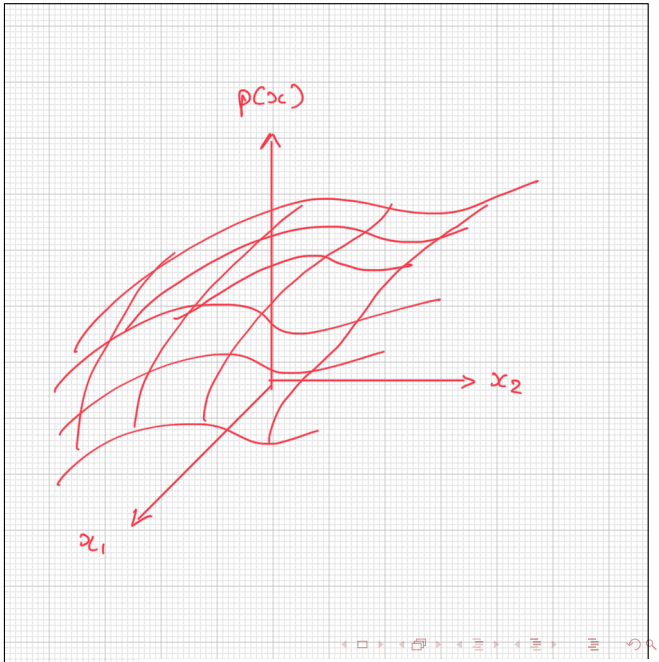
- with $\sigma^2_{X_1} = var(X_1)$



Multi-dimensional Gaussian distribution

Probability Density Function (pdf)

$$\mathcal{N}(\underline{x}|\underline{\mu},\underline{\underline{\Sigma}}) = \frac{1}{(2\pi)^{d/2}|\underline{\underline{\Sigma}}|^{1/2}} e^{-\frac{1}{2}(\underline{x}-\underline{\mu})^T\underline{\underline{\Sigma}}^{-1}(\underline{x}-\underline{\mu})}$$



Bivariate Gaussian distribution

Specific case : we suppose that x_1 and x_2 are two **independent** Gaussian variables :

- $x_1 \sim \mathcal{N}(\mu_1, \sigma^2)$
- $x_2 \sim \mathcal{N}(\mu_2, \sigma^2)$

Their joint distribution is

$$p(x_1, x_2) = |2\pi\Sigma|^{-1/2} \exp\left(-\frac{1}{2}[\underline{x} - \underline{\mu}]^T \underline{\underline{\Sigma}}^{-1}[\underline{x} - \underline{\mu}]\right)$$

Proof

$$\begin{aligned} p(x_1, x_2) &= p(x_2|x_1)p(x_1) \\ &= p(x_2) \cdot p(x_1) \\ &= (2\pi\sigma^2)^{-1/2} e^{-\frac{1}{2\sigma^2}(x_1-\mu_1)^2} \cdot (2\pi\sigma^2)^{-1/2} e^{-\frac{1}{2\sigma^2}(x_2-\mu_2)^2} \\ &= |2\pi\Sigma|^{-1/2} e^{-\frac{1}{2}[x_1-\mu_1, x_2-\mu_2]\begin{pmatrix}\sigma^2 & 0 \\ 0 & \sigma^2\end{pmatrix}^{-1}\begin{pmatrix}x_1-\mu_1 \\ x_2-\mu_2\end{pmatrix}} \end{aligned}$$

with $\Sigma = \begin{bmatrix}\sigma^2 & 0 \\ 0 & \sigma^2\end{bmatrix}$, $|\Sigma| = (\sigma^2)^2$, $a\Sigma \rightarrow a^2|\Sigma|$

Maximum Likelihood Estimation (MLE)

Example

Suppose we have $n = 3$ data points :

- $y^{(1)} = 1$,
- $y^{(2)} = 0.5$,
- $y^{(3)} = 1.5$

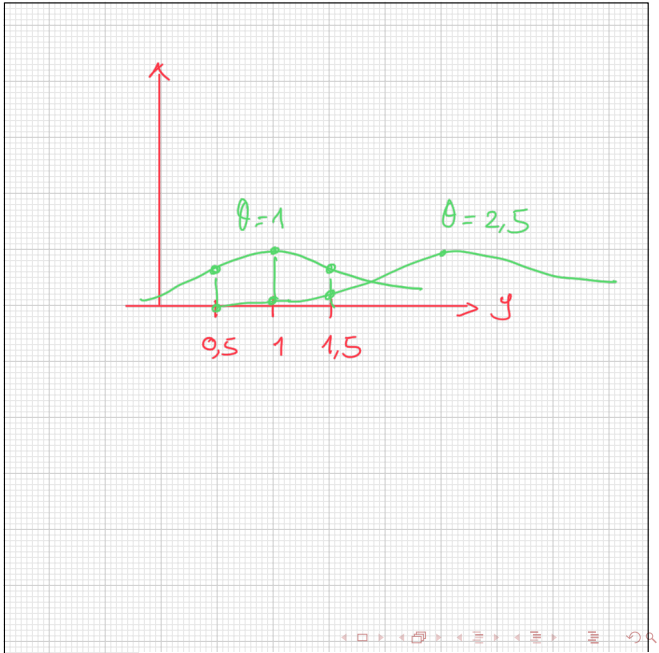
which are independent and Gaussian with unknown mean θ and variance 1.
We can express the likelihood as :

- $y^{(i)} \sim \mathcal{N}(\theta, 1) = \theta + \mathcal{N}(0, 1)$
- $p(y^{(1)}, y^{(2)}, y^{(3)}|\theta) = p(y^{(1)}|\theta)p(y^{(2)}|\theta)p(y^{(3)}|\theta)$

Which guess of θ is more likely ?

- $\theta = 1$
- $\theta = 2.5$

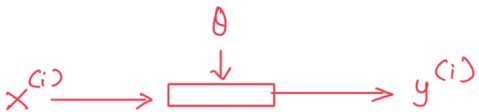
We take the θ the maximizes the likelihood



Likelihood for linear Regression

Suppose $y^{(i)}$ is Gaussian distributed with mean $\underline{x}^{(i)\,T}\underline{\theta}$ and variance σ^2

$$y^{(i)} \sim \mathcal{N}(\underline{x}^{(i)\,T}\underline{\theta}, \sigma^2)$$
$$= \underline{x}^{(i)\,T}\underline{\theta} + \mathcal{N}(0, \sigma^2)$$



The likelihood is

$$p(\underline{y}|\underline{\underline{X}}, \underline{\theta}, \sigma) = \prod_{i=1}^{m} p(y^{(i)}|\underline{x}^{(i)}, \underline{\theta}, \sigma)$$
$$= \prod_{i=1}^{m} (2\pi\sigma)^{-1/2} e^{-\frac{1}{2\sigma^2}(y^{(i)} - \underline{x}^{(i)\,T}\underline{\theta})^2}$$
$$= (2\pi\sigma)^{-m/2} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y^{(i)} - \underline{x}^{(i)\,T}\underline{\theta})^2}$$
$$= (2\pi\sigma)^{-m/2} e^{-\frac{1}{2\sigma^2}(\underline{y} - \underline{\underline{X}}\,\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\,\underline{\theta})}$$

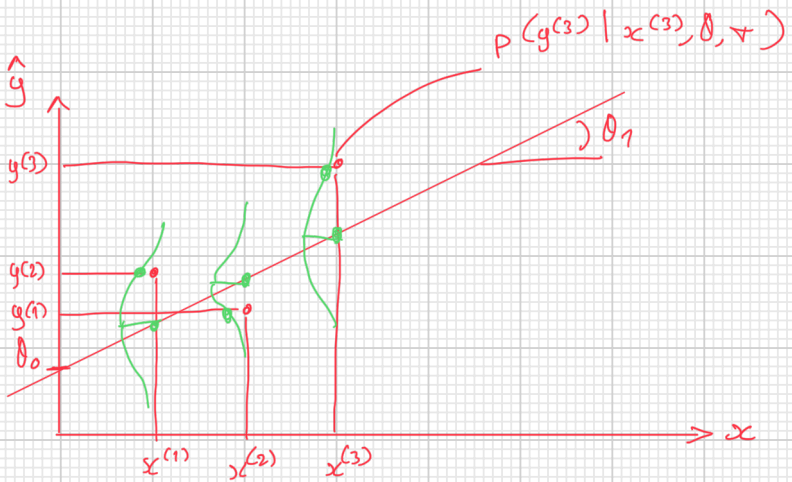
This is equivalent to $p(\text{data}|\text{parameters}) = \frac{1}{Z} e^{-\text{Loss}(\text{data},\text{parameters})}$

- with $\text{Loss}(\text{data}, \text{parameters}) = J(\underline{\theta}) = \sum_{i=1}^{m}(y^{(i)} - \underline{x}^{(i)\,T}\underline{\theta})^2$
  - is the MSE (Mean Square Error) of Linear Regression

Maximazing the likelihood is equivalent to minimizing the Loss

Example (back to linear regression) :

$$p(y^{(i)}|\underline{x}^{(i)}, \underline{\theta}, \sigma) = (2\pi\sigma)^{-1/2} e^{-\frac{1}{2\sigma^2}(y^{(i)} - \theta_0 - x^{(i)}\theta_1)^2}$$



Likelihood for linear Regression

Maximizing the **Likelihood** $\Rightarrow$ minimizing the **Negative Log-Likelihood (NLL)**

$$p(\underline{y}|\underline{\underline{X}},\underline{\theta},\sigma) = (2\pi\sigma^2)^{-m/2} e^{-\frac{1}{2\sigma^2}(\underline{y}-\underline{\underline{X}}\,\underline{\theta})^T(\underline{y}-\underline{\underline{X}}\,\underline{\theta})}$$

$$-\log p(\underline{y}|\underline{\underline{X}},\underline{\theta},\sigma) = \frac{m}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(\underline{y}-\underline{\underline{X}}\,\underline{\theta})^T(\underline{y}-\underline{\underline{X}}\,\underline{\theta})$$

MLE of $\underline{\theta}$ (same solution as MSE) :

$$\underline{\theta}_{ML} = \boxed{(\underline{\underline{X}}^T X)^{-1} X^T \underline{y}}$$

Proof :

$$0 = \frac{\partial}{\partial \underline{\theta}} \left(\frac{1}{2\sigma^2}(\underline{y}-\underline{\underline{X}}\,\underline{\theta})^T(\underline{y}-\underline{\underline{X}}\,\underline{\theta}) \right)$$

MLE of σ :

$$\sigma^2_{ML} = \frac{1}{m}(\underline{y}-\underline{\underline{X}}\,\underline{\theta})^T(\underline{y}-\underline{\underline{X}}\,\underline{\theta})$$

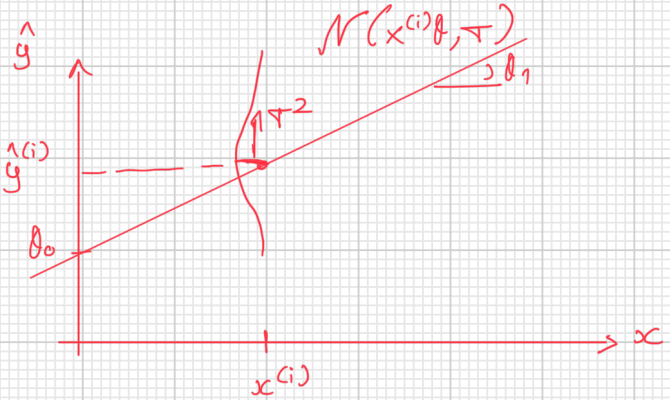
$$= \boxed{\frac{1}{m}\sum_{i=1}^{n}(y^{(i)} - \underline{x}^{(i)^T}\underline{\theta})^2}$$

Proof

$$0 = \frac{\partial}{\partial \sigma} \left(\frac{m}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(\underline{y}-\underline{\underline{X}}\,\underline{\theta})^T(\underline{y}-\underline{\underline{X}}\,\underline{\theta}) \right)$$

$$= m\frac{1}{\sigma} - \frac{1}{\sigma^3}(\underline{y}-\underline{\underline{X}}\,\underline{\theta})^T(\underline{y}-\underline{\underline{X}}\,\underline{\theta})$$

Making predictions (provides a confidence in prediction !)



Bernoulli distribution

Benoulli distribution : X takes value in $\{0, 1\}$

- model of a coin (probability of head or tail)

$$P(X = x|\theta) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases}$$

$$= \boxed{p^x(1-p)^{(1-x)}}$$

- with $p \in [0, 1]$

Bernoulli distribution

Entropy : is a measure of the uncertainty associated with a random variable :

$$H(X) = -\sum_x p(x|\theta)\log(p(x|\theta))$$

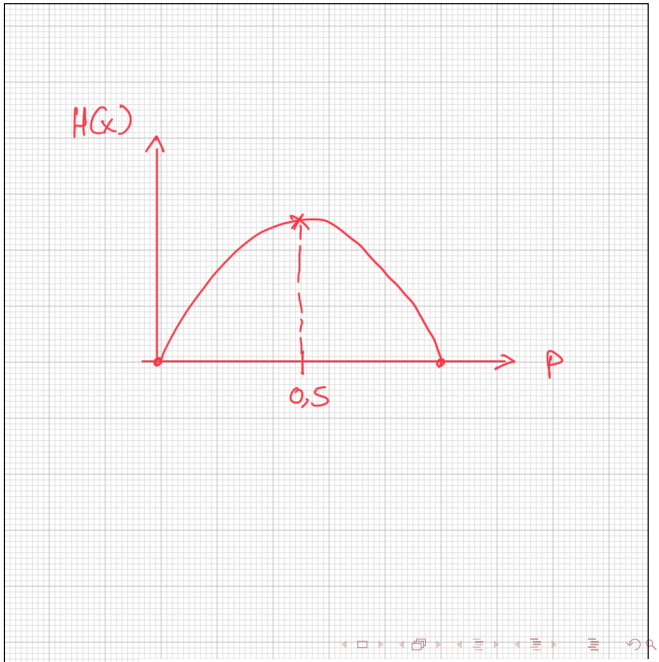
Entropy for a Brenouilli distribution

$$H(X) = -\sum_{x=0}^{1} p^x(1-p)^{(1-x)}\log\left(p^x(1-p)^{(1-x)}\right)$$
$$= -[\underbrace{(1-p)\log(1-p)}_{if\ x=0} + \underbrace{p\log(p)}_{if\ x=1}]$$

Entropy of a Gaussian distribution at D dimensions

$$H(\mathcal{N}(\mu,\Sigma)) = \frac{1}{2}\log((2\pi e)^D|\Sigma|)$$

- proportionnal to the determinant of the covariance matrix



Regression : Ridge, Lasso, Non-linear (basis, kernel)

Ridge regularization

Least-square regression

$$J(\theta) = (\underline{y} - \underline{\underline{X}}\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\underline{\theta})$$

$$\boxed{\hat{\underline{\theta}} = (\underline{\underline{X}}^T\underline{\underline{X}})^{-1}\underline{\underline{X}}^T\underline{y}}$$

- inversion of $(\underline{\underline{X}}^T\underline{\underline{X}})$ can lead to problems (if system poorly conditionned)
- solution : add a small constant δ to the diagonal elements

Ridge regression

$$\boxed{\hat{\underline{\theta}} = (\underline{\underline{X}}^T\underline{\underline{X}} + \delta^2\mathbb{I})^{-1}\underline{\underline{X}}^T\underline{y}} \text{ with } \mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Corresponds to the **regularized quadratic cost function**

$$J(\theta) = (\underline{y} - \underline{\underline{X}}\,\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\,\underline{\theta}) + \delta^2\underline{\theta}^T\underline{\theta}$$

- equivalent to

$$\min_{\underline{\theta}:\underline{\theta}^T\underline{\theta} \leqslant t(\delta)} (\underline{y} - \underline{\underline{X}}\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\underline{\theta})$$

- where t is an arbitrary function
- also named weight decay

Proof :

$$\begin{aligned} 0 &= \frac{\partial}{\partial\underline{\theta}}(\underline{\theta}^T\underline{\underline{X}}^T\underline{\underline{X}}\underline{\theta} - 2\underline{y}^T\underline{\underline{X}}\underline{\theta} + \underline{y}^T\underline{y} + \delta^2\underline{\theta}^T\underline{\theta}) \\ &= 2\underline{\underline{X}}^T\underline{\underline{X}}\underline{\theta} - 2\underline{\underline{X}}^T\underline{y} + 2\delta^2\mathbb{I}\underline{\theta} \\ &= 2(\underline{\underline{X}}^T\underline{\underline{X}} + \delta^2\mathbb{I})\underline{\theta} - 2\underline{\underline{X}}^T\underline{y} \\ \underline{\theta}_{ridge} &= (\underline{\underline{X}}^T\underline{\underline{X}} + \delta^2\mathbb{I})^{-1}\underline{\underline{X}}^T\underline{y} \end{aligned}$$

Ridge regularization

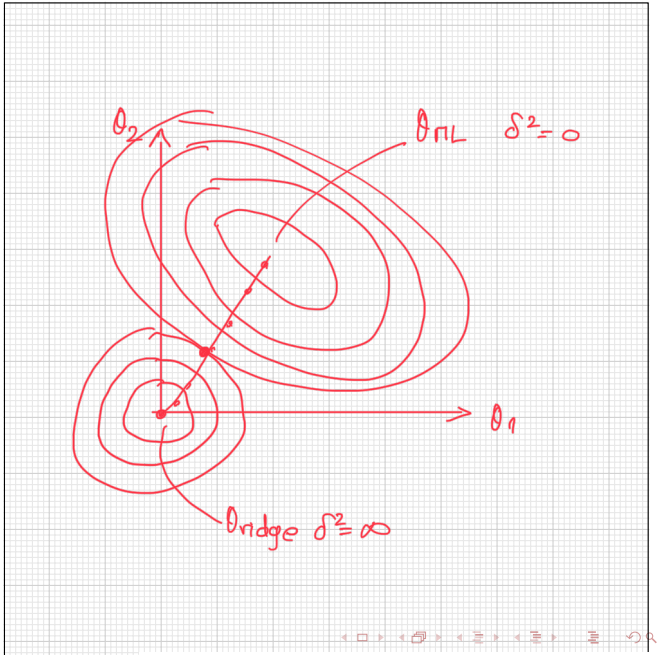
Example :

$$\underline{\theta}^T = [\theta_1, \theta_2]$$

$$\rightarrow \underline{\theta}^T \underline{\theta} = [\theta_1, \theta_2] \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$= \theta_1^2 + \theta_2^2 = const$$

- equation of a circle
- automatically remove irrelevent axes



LASSO regularization

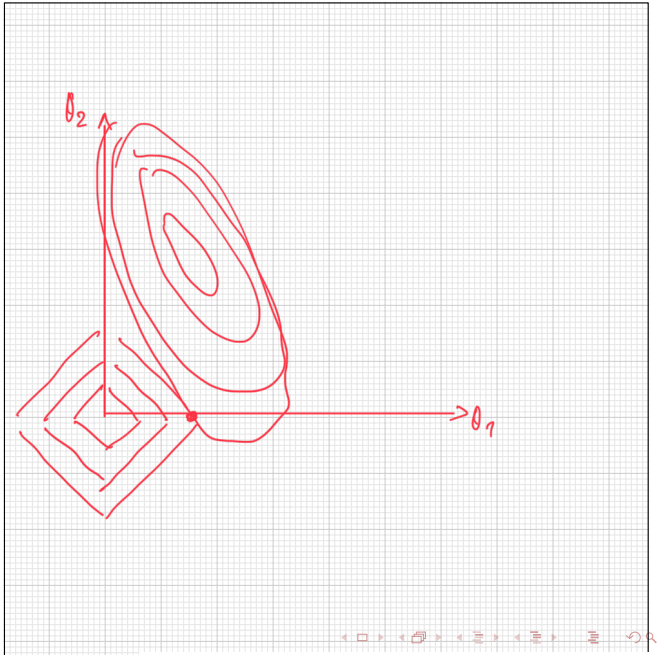
Ridge :

$$||\underline{\theta}||_2^2 = \underline{\theta}^T \underline{\theta}$$

Lasso :

$$||\underline{\theta}||_1 = \sum_d |\theta_d| = \theta_1 + \theta_2$$

- intersections happen at corners ($\theta_2 = 0$)
- sparse, compress-sensing



Non-linear regression (regression with basis functions)

Basis functions $\phi(.)$

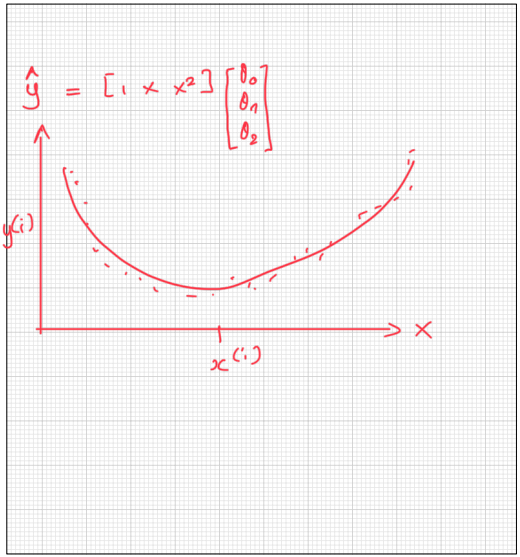
$$\boxed{\hat{y}(\underline{x}) = \underline{\phi}(\underline{x})\ \underline{\theta} + \epsilon}$$

Examples

- in 1 dimensions : $\underline{\phi}(x) = [1, x, x^2]$

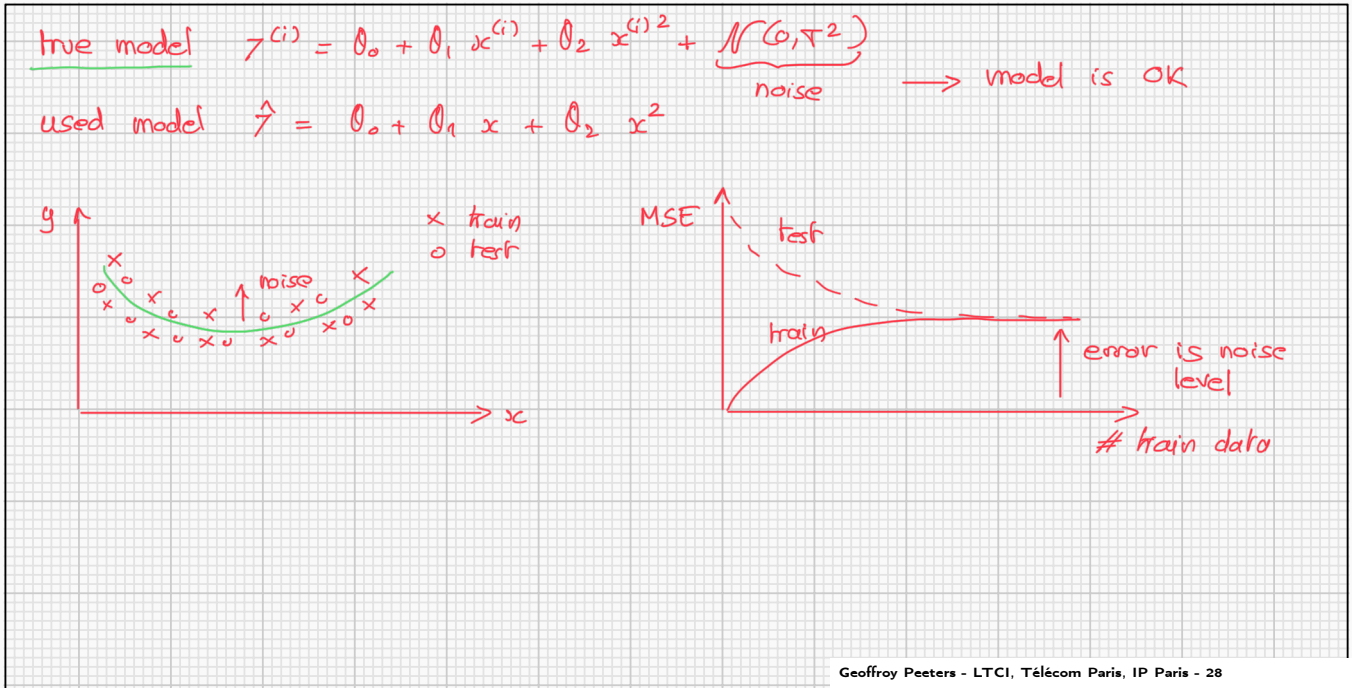
$$\begin{aligned}\hat{y} &= \underline{\phi}(x)\ \underline{\theta} \\ &= \theta_0 + x\theta_1 + x^2\theta_2 \\ &= [1\ x\ x^2] \begin{bmatrix}\theta_0 \\ \theta_1 \\ \theta_2\end{bmatrix}\end{aligned}$$

- in 2 dimensions : $\underline{\phi}(\underline{x}) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$



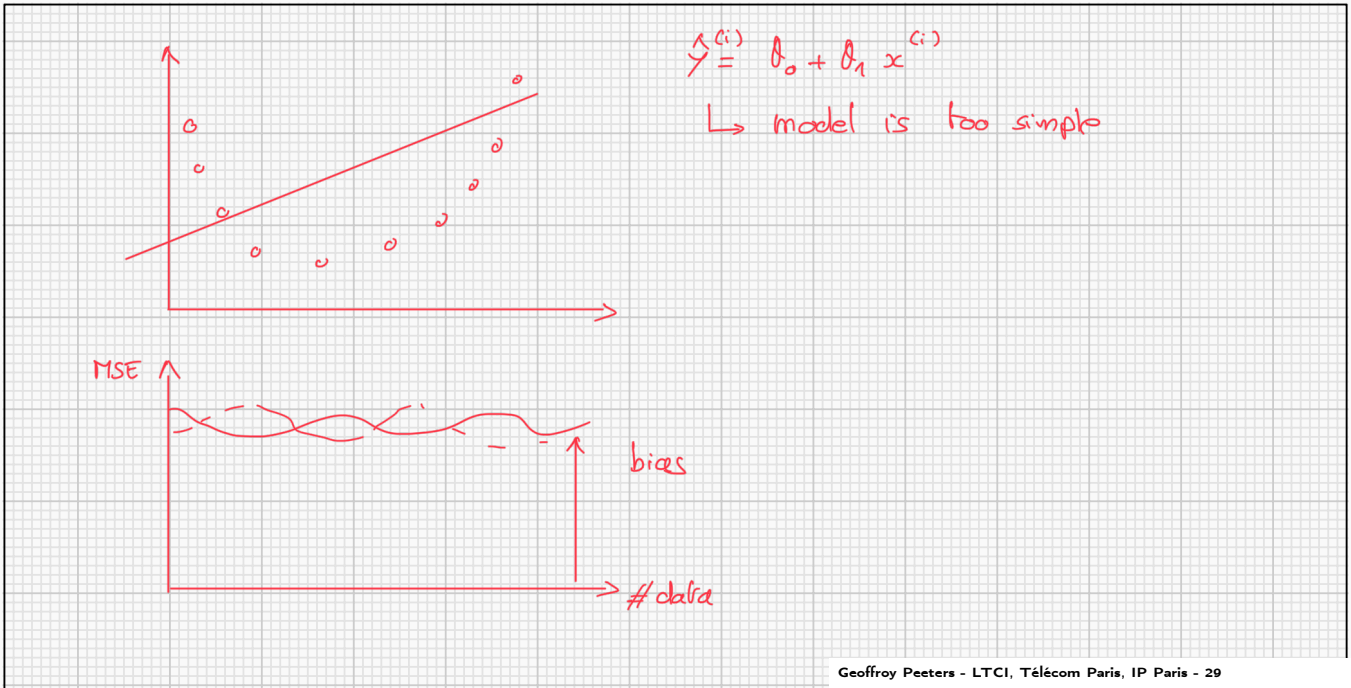
Having the right model complexity / having large number of training data

Good model (bias equal noise level)



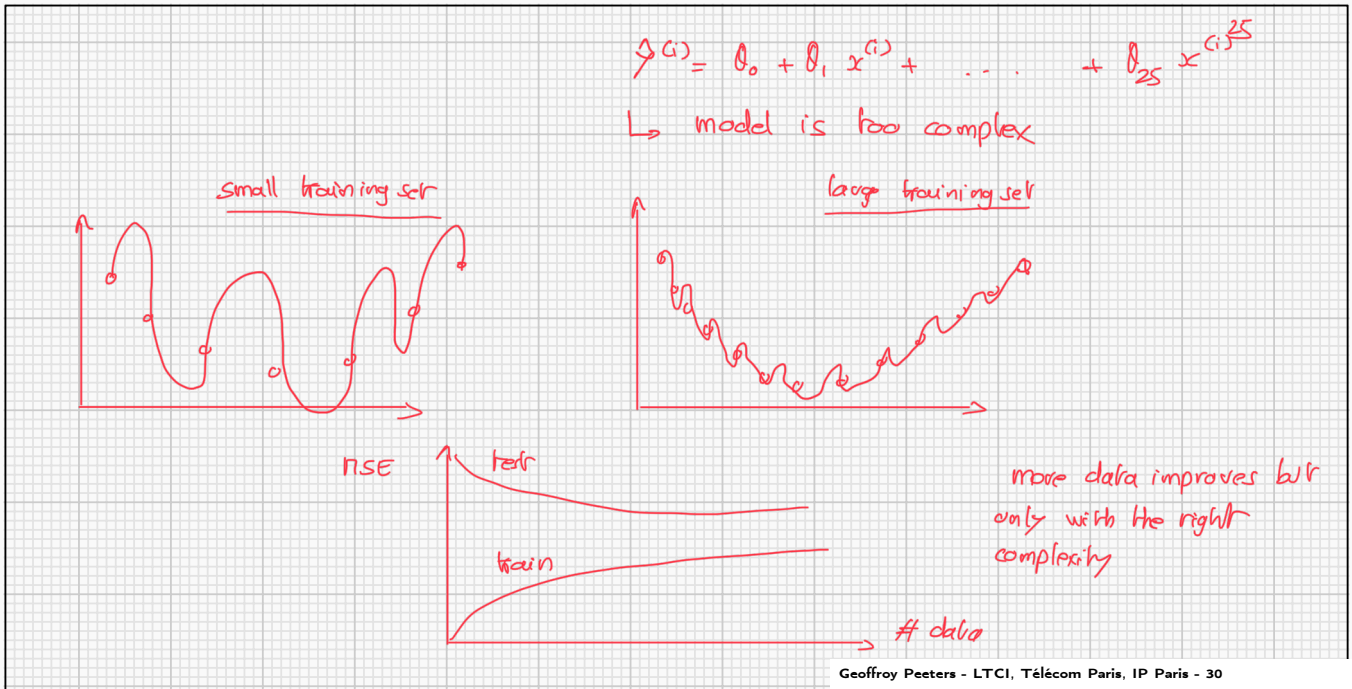
Having the right model complexity / having large number of training data

Too simplest model (bad results whatever number or data)

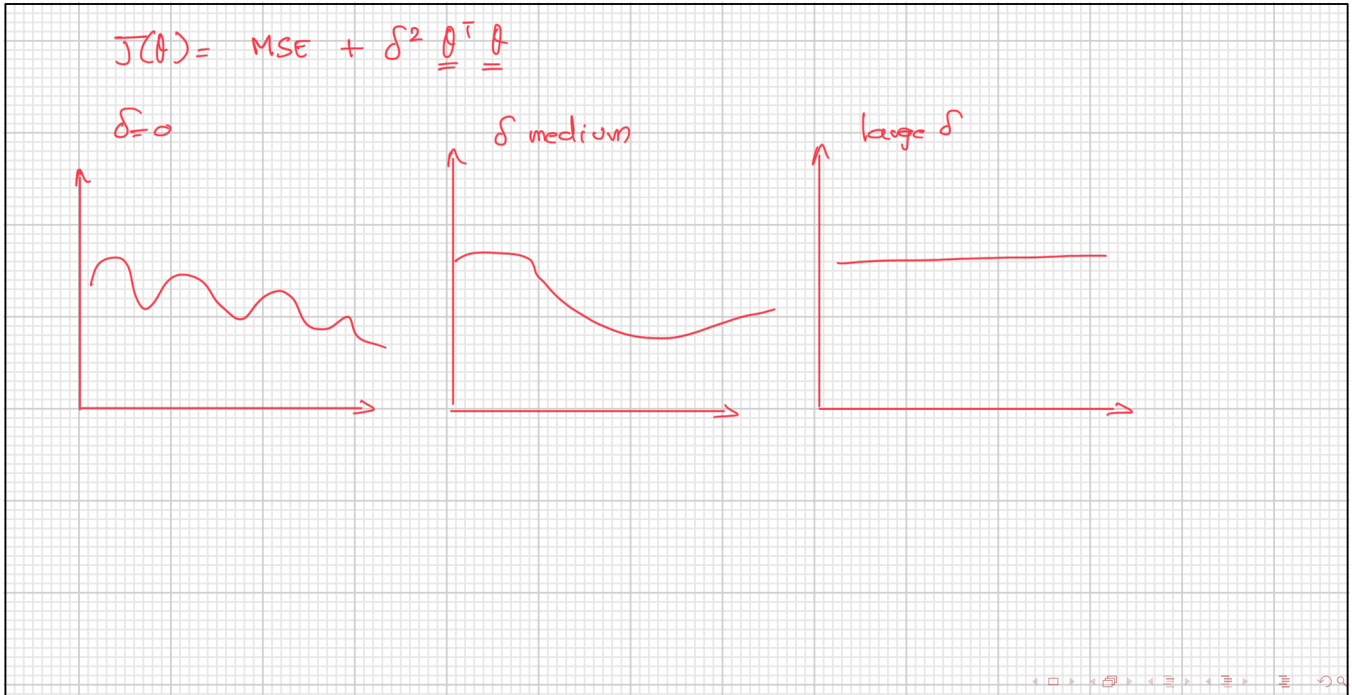


Having the right model complexity / having large number of training data

Very complex model (results on test-set depend on the number of data)



Controlling model complexity with regularzation δ



Non-linear regression (regression with basis functions)

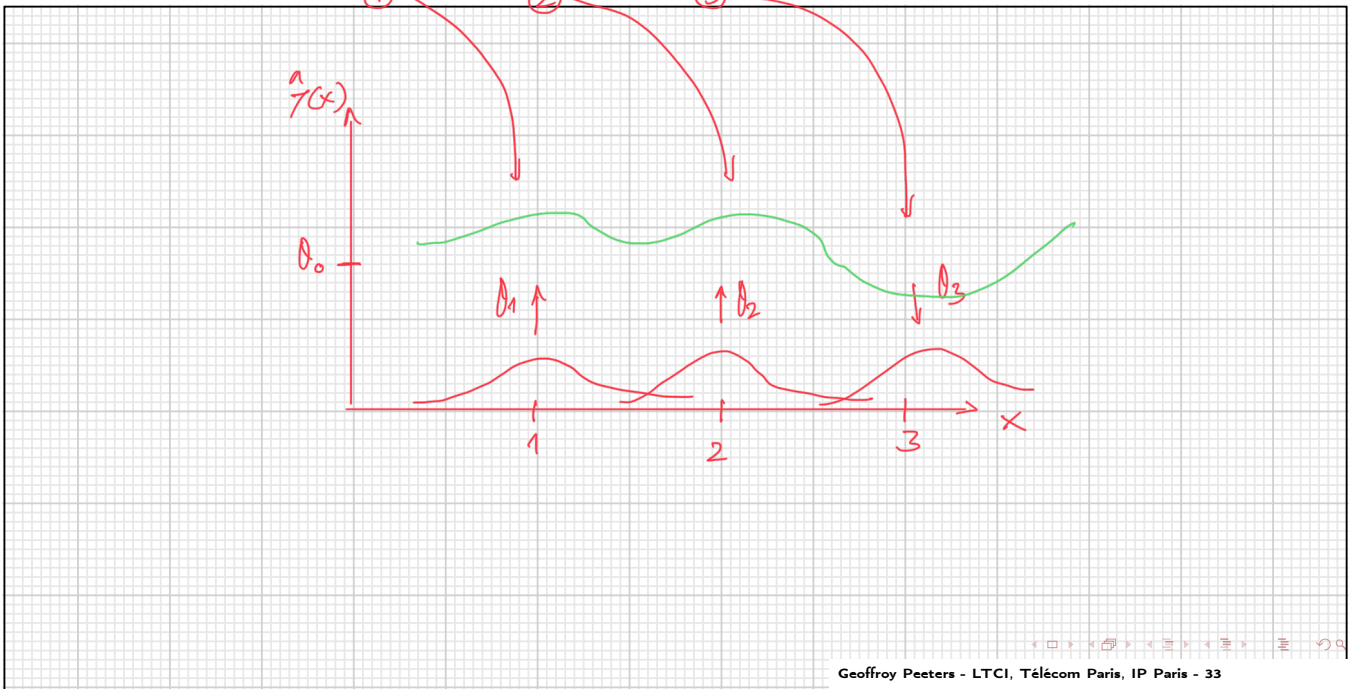
- **RBF kernel :** $\underline{\phi}(\underline{x}) = [k(\underline{x}, \underline{\mu}_1, \lambda), \cdots, k(\underline{x}, \underline{\mu}_d, \lambda)]$
  - with $k(\underline{x}, \underline{\mu}_d, \lambda) = e^{-\frac{1}{\lambda}||\underline{x}-\underline{\mu}_d||^2}$

$$\hat{y}(\underline{x}^{(i)}) = \phi(\underline{x}^{(i)})\underline{\theta}$$
$$= 1\theta_0 + k(\underline{x}^{(i)}, \underline{\mu}_1, \lambda)\theta_1 + \ldots + k(\underline{x}^{(i)}, \underline{\mu}_d, \lambda)\theta_d$$

$\Phi(x) = [1, x, x^2]$
$\theta_0 \ \theta_1 \ \theta_2$
$= [e^{-\frac{\|x-\mu_1\|^2}{\lambda}}, e^{-\frac{\|x-\mu_2\|^2}{\lambda}}, e^{-\frac{\|x-\mu_3\|^2}{\lambda}}]$

Kernel regression with RBFs

- **Example :** $\hat{y}(\underline{x}) = \theta_0 + e^{-||x-1||^2}\theta_1 + e^{-||x-2||^2}\theta_2 + e^{-||x-3||^2}\theta_3$



Kernel regression with RBFs

We note $\underline{\phi}(x^{(i)})$ the 4-dimensional (3 bases) :

- For one data :
$$\underline{\phi}(\underline{x}^{(i)}) = [1, k(\underline{x}^{(i)}, \underline{\mu}_1, \lambda), k(\underline{x}^{(i)}, \underline{\mu}_2, \lambda), k(\underline{x}, \underline{\mu}_3, \lambda)]$$
- For m data

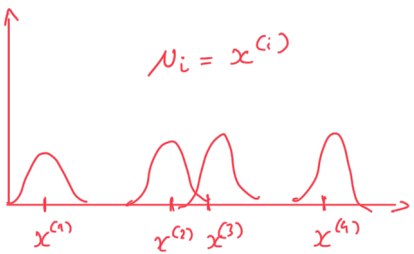
$$\hat{\underline{y}} = \underline{\underline{\phi}}\,\underline{\theta} \quad \rightarrow \times\theta$$
$$\hat{\underline{\theta}}_{LS} = (\underline{\underline{\phi}}^T\underline{\underline{\phi}})^{-1}\underline{\underline{\phi}}^T\underline{y} \quad \rightarrow (X^TX)^{-1}X^Ty$$
$$\hat{\underline{\theta}}_{Ridge} = \boxed{(\underline{\underline{\phi}}^T\underline{\underline{\phi}} + \delta^2\mathbb{I})^{-1}\underline{\underline{\phi}}^T\underline{y}}$$

$$\underline{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \underline{\underline{\phi}} = \begin{bmatrix} \underline{\phi}(x^{(1)}) \\ \vdots \\ \underline{\phi}(x^{(m)}) \end{bmatrix}, \underline{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

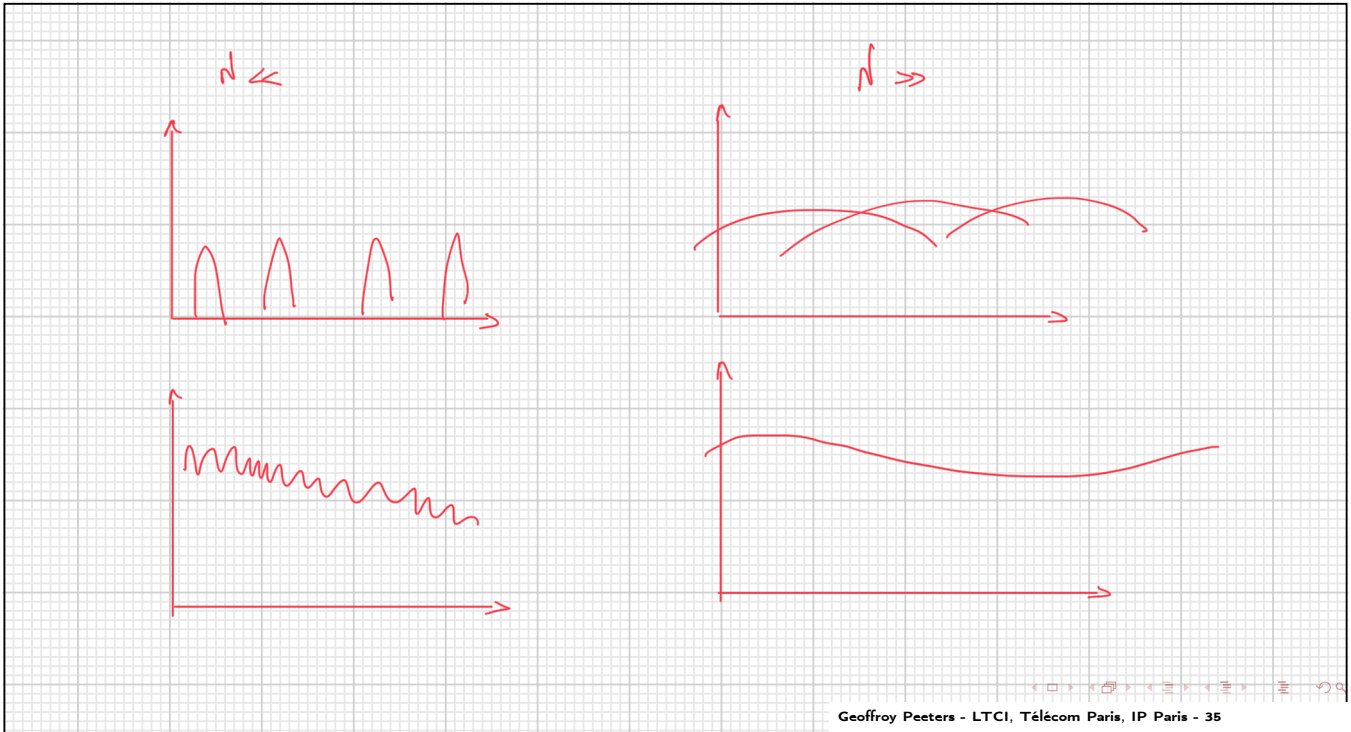
- $\Rightarrow$ It is still a regression !

Specific case :

- choose the $\underline{\mu}_i$ equal to the training data point $\underline{x}^{(i)}$
- $\Rightarrow$ one kernel on each training data point



Choice of the kernel size λ



Optimisation

Gradient

$$\frac{\partial f(\theta_1,\theta_2)}{\partial \theta_1} = \lim_{\Delta\theta_1 \to 0} \frac{f(\theta_1+\Delta\theta_1,\theta_2) - f(\theta_1,\theta_2)}{\Delta\theta_1}$$

- Consider the function $f(\theta_1,\theta_2) = \theta_1^2 + \theta_2^2$

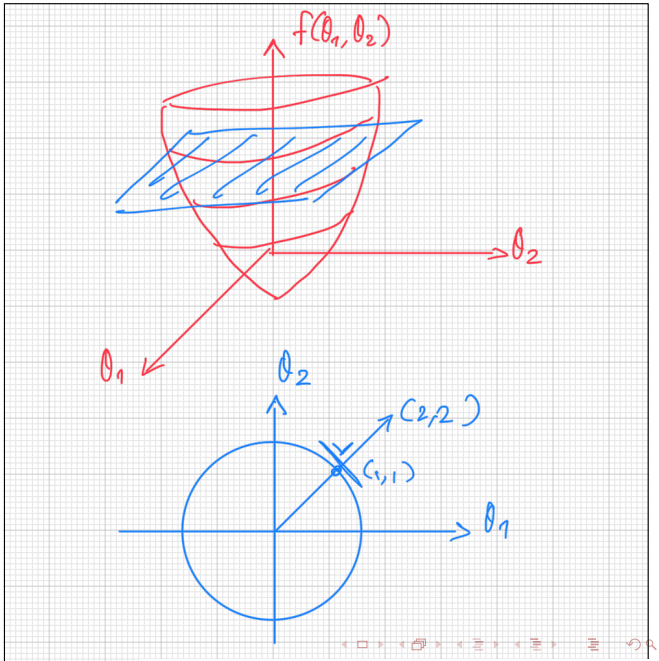
$$\frac{\partial f}{\partial \theta_1} = 2\theta_1$$

$$\frac{\partial f}{\partial \theta_2} = 2\theta_2$$

$$\nabla f(\theta) = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 \end{bmatrix}$$

Gradient :

- always orthogonal to the tangent of the level curve
- indicates the direction of
  - the maximum change,
  - the steepest ascent



Hessian

$$\frac{\partial}{\partial \theta_1}\left(\frac{\partial f(\theta)}{\partial \theta_1}\right) = \frac{\partial^2 f}{\partial \theta_1^2} = 2$$

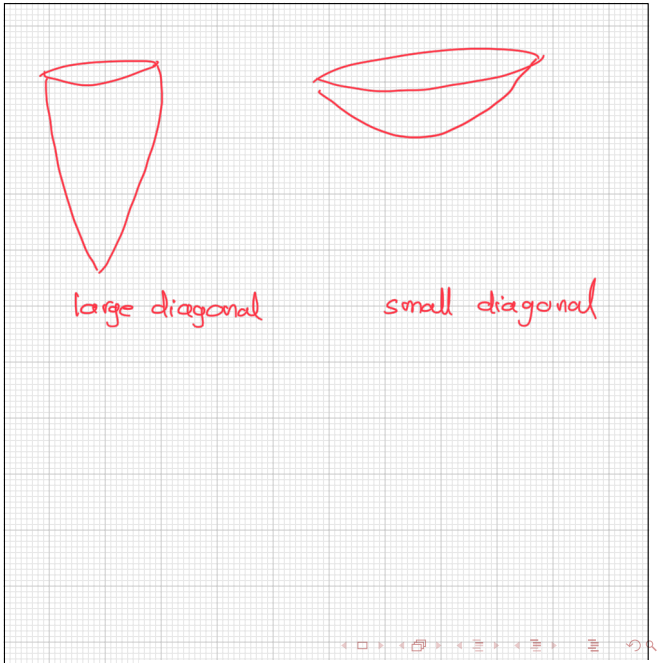
$$\frac{\partial^2 f}{\partial \theta_2^2} = 2$$

$$\frac{\partial^2 f}{\partial \theta_1 \theta_2} = \frac{\partial^2 f}{\partial \theta_2 \theta_1} = 0$$

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Hessian :

- represents the curvature,
- allows to check convexity (saddle point)
- if a minimum exists, all positive eigen values



Gradient : $\mathbb{R}^d \rightarrow \mathbb{R}$

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_d} \end{bmatrix}$$

Jacobian : $\mathbb{R}^d \rightarrow \mathbb{R}^{d'}$

$$J(\theta) = \begin{bmatrix} \frac{\partial^2 f_1(\theta)}{\partial \theta_1} & \cdots & \frac{\partial f_1(\theta)}{\partial \theta_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f_{d'}(\theta)}{\partial \theta_1} & \cdots & \frac{\partial f_{d'}(\theta)}{\partial \theta_d} \end{bmatrix}$$

Hessian $\mathbb{R}^d \rightarrow \mathbb{R}$

$$\nabla^2_\theta f(\theta) = \begin{bmatrix} \frac{\partial^2 f(\theta)}{\partial \theta_1^2} & \cdots & \frac{\partial f(\theta)}{\partial \theta_1 \theta_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\theta)}{\partial \theta_d \theta_1} & \cdots & \frac{\partial f(\theta)}{\partial \theta_d^2} \end{bmatrix}$$

How to estimate them ?

$$f(\theta) = f(\theta, x^{(1:m)}) = \frac{1}{m} \sum_{i=1}^{m} f(\theta, x^{(i)})$$

$$\underline{g}(\theta) = \nabla_\theta f(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta f(\theta, x^{(i)})$$

$$\underline{\underline{H}}(\theta) = \nabla^2_\theta f(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla^2_\theta f(\theta, x^{(i)})$$

Calculus background : Chain rule

$$z = f(x(u,v)), y(u,v))$$

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}$$

Gradient and Hessian for linear regression

- For linear regression : $y^{(i)} = \theta_0 + x^{(i)}\theta_1$

Scalar form

$$f(\theta) = \sum_i \left(y^{(i)} - \theta_0 - x^{(i)}\theta_1\right)^2 + \delta^2\theta_1^2$$

Applying the chain rule :

$$\nabla_\theta f = \begin{bmatrix} \frac{\partial}{\partial \theta_0} \\ \frac{\partial}{\partial \theta_1} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_i 2(y^{(i)} - \theta_0 - x^{(i)}\theta_1)(-1) \\ \sum_i 2(y^{(i)} - \theta_0 - x^{(i)}\theta_1)(-x^{(i)}) + 2\delta^2\theta_1 \end{bmatrix}$$

$$= \begin{bmatrix} 2\sum_i 1 \cdot \left((\theta_0 + x^{(i)}\theta_1) - y^{(i)}\right)(-1) \\ 2\sum_i x^{(i)} \cdot \left((\theta_0 + x^{(i)}\theta_1) - y^{(i)}\right) + 2\delta^2\theta_1 \end{bmatrix}$$

Matrix form

$$f(\theta) = (\underline{y} - \underline{\underline{X}}\,\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\,\underline{\theta})$$

$$\nabla_\theta f(\theta) = \frac{\partial}{\partial \theta}(\underline{y}^T\underline{y} - 2\underline{y}^T\underline{\underline{X}}\,\underline{\theta} + \underline{\theta}^T\underline{\underline{X}}^T\underline{\underline{X}}\,\underline{\theta})$$

$$= -2\underline{\underline{X}}^T\underline{y} + 2\underline{\underline{X}}^T\underline{\underline{X}}\,\underline{\theta}$$

$$= \boxed{2\underline{\underline{X}}^T(\underline{\underline{X}}\,\underline{\theta} - \underline{y})}$$

$$\nabla^2_\theta f(\theta) = 0 + 2\underline{\underline{X}}^T\underline{\underline{X}}$$

$$= \boxed{2\underset{(d,m)}{\underline{\underline{X}}^T}\ \underset{(m,d)}{\underline{\underline{X}}}}$$

Optimisation : 1) Steepest/Stochastic Gradient Descent (SGD) algorithm

Notation : $\underline{\theta}_k$ value of $\underline{\theta}$ at iteration k

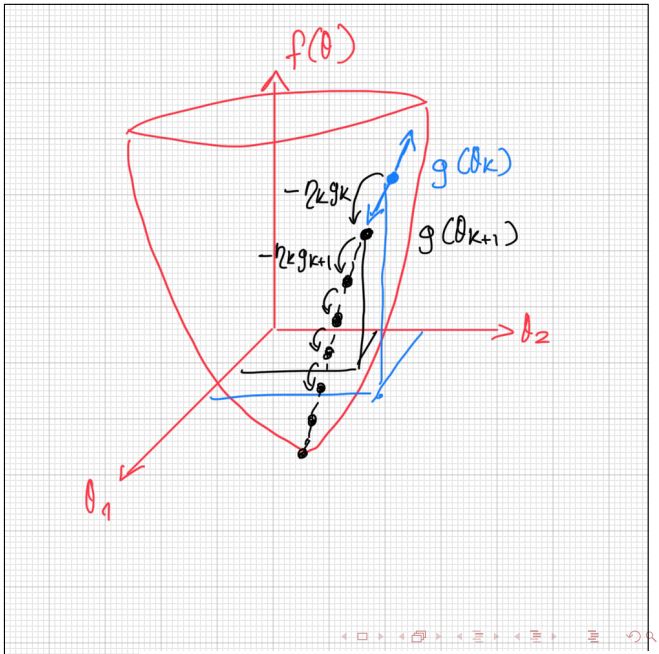
Steepest/Stochastic Gradient Descent (SGD) algorithm

$$\theta_{k+1} = \theta_k - \eta_k \underline{g}_k$$
$$= \theta_k - \eta_k \nabla_\theta f(\theta_k)$$

- where η_k is the learning rate, step size

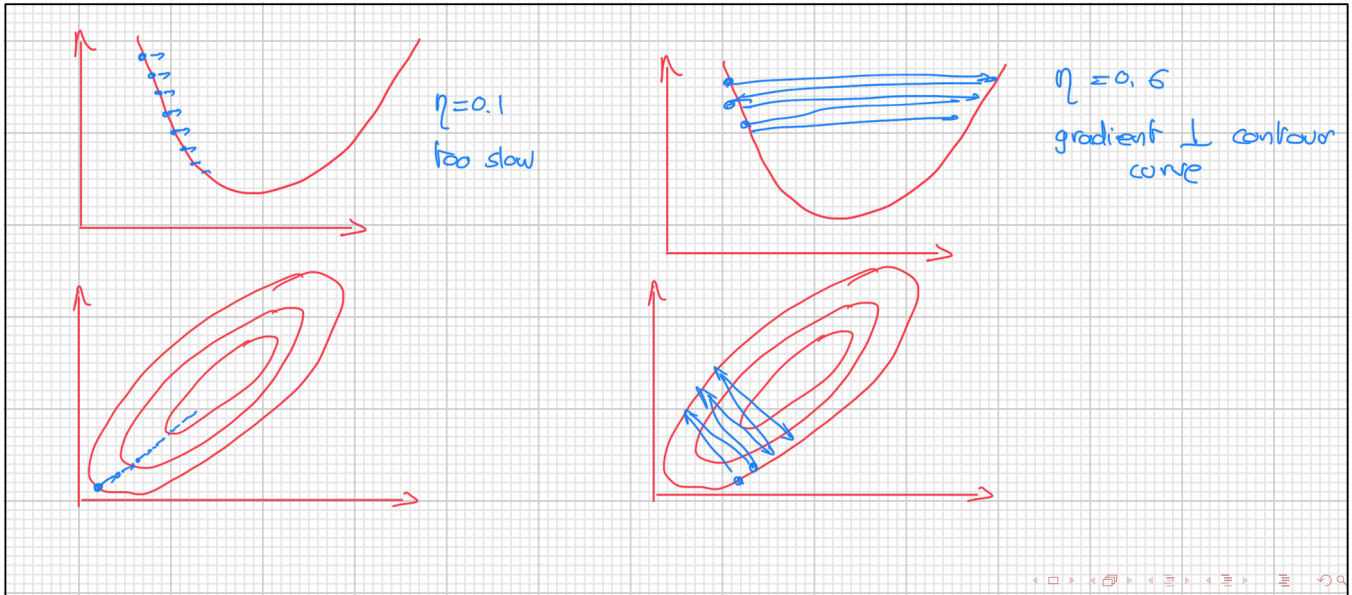
SGD for Linear Regression

$$\theta_{k+1} = \theta_k - \eta_k \left[2\underline{\underline{X}}^T (\underline{\underline{X}}\, \underline{\theta}_k - \underline{y}) \right]$$



Optimisation : 1) Steepest/Stochastic Gradient Descent (SGD) algorithm

- How to choose η_k ?
  - $\eta = 0.1$ too slow
  - $\eta = 0.6$ too large



Optimisation : 2) Newton/Hessian algorithm

If we knew the curvature, we could adpat η_k !

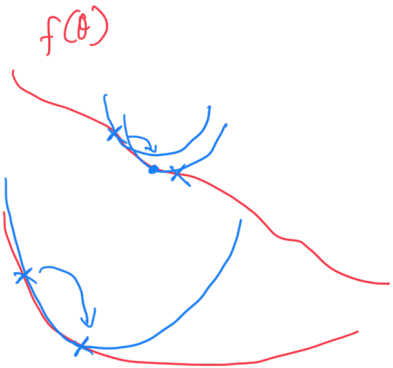
Taylor serie expansion (qapproximate $f(\underline{\theta})$ around $\underline{\theta}_k$ with a quadratic ball) :

$$f_{quad}(\theta) = f(\theta_k) + \underline{g}_k^T(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T \underline{\underline{H}}_k(\theta - \theta_k)$$

find the minimum of the quadratic ball :

$$\begin{aligned} 0 &= \nabla_\theta f_T(\theta) \\ &= 0 + \underline{g}_k + \underline{\underline{H}}_k(\theta - \theta_k) \\ -\underline{g}_k &= \underline{\underline{H}}_k(\theta - \theta_k) \\ \theta_{k+1} &= \boxed{\theta_k - \underline{\underline{H}}_k^{-1}\underline{g}_k} \end{aligned}$$

- $\Rightarrow$ this is Newton's method



Optimisation : 2) Newton/Hessian algorithm

Newton's for Linear Regression :

$$\begin{aligned}\theta_{k+1} &= \theta_k - \underline{\underline{H}}_k^{-1}\underline{g}_k \\ &= \theta_k - (2\underline{\underline{X}}^T\underline{\underline{X}})^{-1}[2\underline{\underline{X}}^T(\underline{\underline{X}}\theta_k - \underline{y})] \\ &= \theta_k - (\underline{\underline{X}}^T\underline{\underline{X}})^{-1}\underline{\underline{X}}^T\underline{\underline{X}}\theta_k + (\underline{\underline{X}}^T\underline{\underline{X}})^{-1}\underline{\underline{X}}^T\underline{y} \\ &= \boxed{(\underline{\underline{X}}^T\underline{\underline{X}})^{-1}\underline{\underline{X}}^T\underline{y}}\end{aligned}$$

using (as seen before)

$$f(\theta) = (\underline{y} - \underline{\underline{X}}\,\underline{\theta})^T(\underline{y} - \underline{\underline{X}}\,\underline{\theta})$$

$$\underline{g} = \nabla_\theta f(\theta) = 2\underline{\underline{X}}^T(\underline{\underline{X}}\underline{\theta} - \underline{y})$$

$$\underline{\underline{H}} = \nabla^2_\theta f(\theta) = 2\underline{\underline{X}}^T\underline{\underline{X}}$$

- This is the theoretical solution (we get it in just one step) !
- In practice, it is very costly
  - $\Rightarrow$ need to store H (if we have 1.000.000 neurons this is a 1.000.000x1.000.000 matrix !)
  - BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm, L-BFGS (Limited memory) algorithm

Why "Stochastic" in SGD ?

Stochastic ?

- because we only have seen a fraction of the data, uncertainty

$$\nabla_{\underline{\theta}} f(\theta) = \underbrace{\int \nabla_{\underline{\theta}} f(x, \underline{\theta}) p(x) dx}_{\mathbb{E}[\nabla_{\underline{\theta}} f(x,\theta)]} \overset{\text{we want}}{=} 0$$

SGD
↑
steepest

Can be re-written

$$\begin{aligned}
\mathbb{E}[\nabla_{\underline{\theta}} f(x, \theta)] &= 0 \\
\eta \mathbb{E}[\nabla_{\underline{\theta}} f(x, \theta)] &= 0 \cdot \eta = 0 \\
\underline{\theta} + \eta \mathbb{E}[\nabla_{\underline{\theta}} f(x, \theta)] &= \underline{\theta} \\
\underline{\theta}_{k+1} &= \underline{\theta}_k - \eta \mathbb{E}[\nabla_{\underline{\theta}} f(x, \theta)] \\
&\simeq \underline{\theta}_k - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\underline{\theta}} f(x^{(i)}, \theta) \text{ with } x^{(i)} \sim p(x) \\
&\simeq \underline{\theta}_k - \eta \nabla_{\underline{\theta}} f(x^{(k)}, \theta_k) \quad \swarrow\ m=1
\end{aligned}$$

Therefore

$$\underbrace{\underline{\theta}_{k+1} = \underline{\theta}_k - \eta \mathbb{E}[\nabla_{\underline{\theta}} f(x, \theta)]}_{\text{what we want}} + \underbrace{\eta \left[\mathbb{E}[\nabla_{\underline{\theta}} f] - \nabla_{\underline{\theta}} f(x^{(k)}, \theta_k) \right]}_{\text{noise, change over time, stochastic, bounded}}$$

Variations

Re-writting SGD as a sum :

$$\begin{aligned}\underline{\theta}_{k+1} &= \underline{\theta}_k - \eta_k \left[2\underline{\underline{X}}^T(\underline{\underline{X}}\,\underline{\theta}_k - \underline{y})\right] \\ &= \underline{\theta}_k + \eta_k \underline{\underline{X}}^T(\underline{y} - \underline{\underline{X}}\,\underline{\theta}_k) \\ &= \underline{\theta}_k + \eta_k \sum_i \underline{x}^{(i)\,T}(y^{(i)} - \underline{x}^{(i)}\,\underline{\theta}_k)\end{aligned}$$

Variations of SGD

- **Batch** (all m data points) :

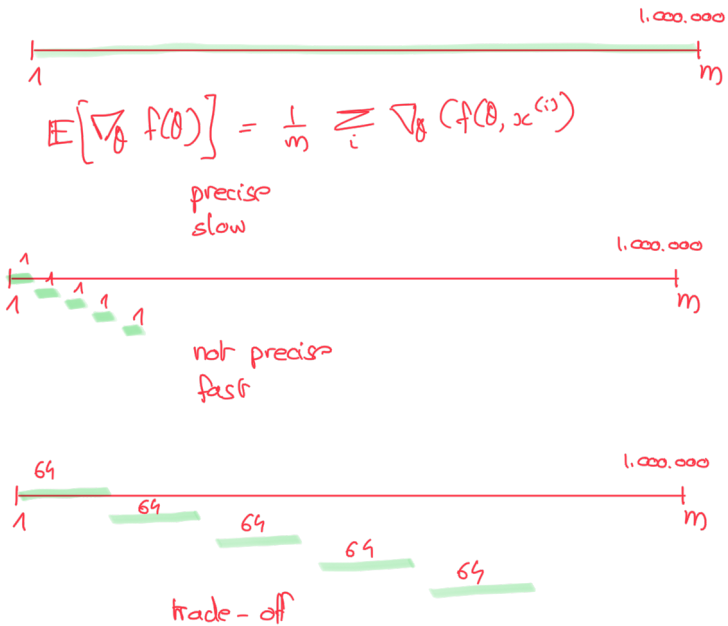
$$\theta_{k+1} = \theta_k + \eta_k \sum_{i=1}^{m} \underline{x}^{(i)\,T}(y^{(i)} - \underline{x}^{(i)}\underline{\theta}_k)$$

- **Online** (a single data point) :

$$\theta_{k+1} = \theta_k + \eta_k (\underline{x}^{(k)})^T (y^{(k)} - \underline{x}^{(k)}\underline{\theta}_k)$$

- **Mini-Batch** (a subset of data points) :

$$\theta_{k+1} = \theta_k + \eta_k \sum_{j=1}^{20} (\underline{x}^{(j)})^T (y^{(j)} - \underline{x}^{(j)}\underline{\theta}_k)$$



Variations

Momentum :

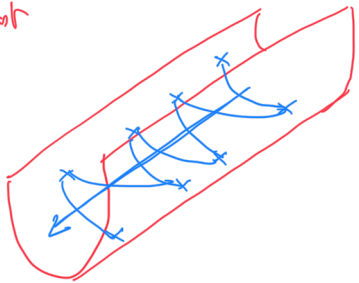
- Goal : accelerate descent in cumulated direction

previous

$$\theta_{k+1} = \theta_k + \alpha(\theta_k - \theta_{k-1}) + (1-\alpha)[-\eta \nabla J(\theta)]$$

$$\underbrace{\theta_{k+1} - \theta_k}_{\Delta\theta_{k+1}} = \alpha \underbrace{(\theta_k - \theta_{k-1})}_{\Delta\theta_k} + (1-\alpha)[-\eta \nabla J(\theta)]$$

gradient



Adagrad :

- Goal : put more weights on rare features ;
  - For feature d at step k

$$w_d^{k+1} = w_d^k - \frac{\eta}{\sum_{\tau=1}^{k} (g_d^\tau)^2} g_d^k$$

d dimension

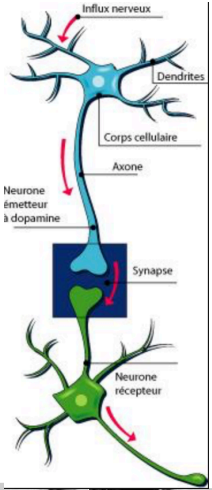
increase

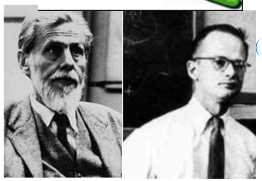
  - $\sum (g_d^\tau)^2$ is the history of how much a feature has changed in the past
  - if small (has been seen few times in the past) then put more weights

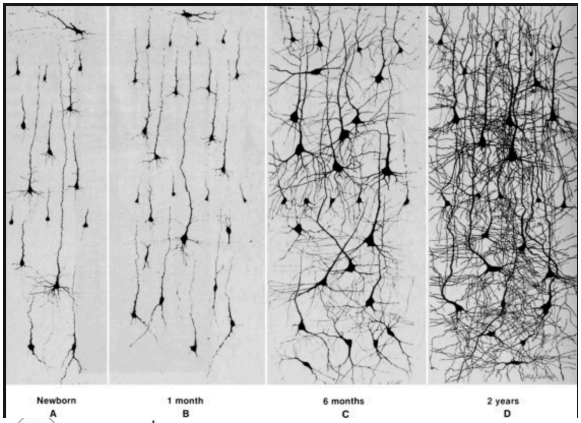
Momentum + Adagrad = ADAM

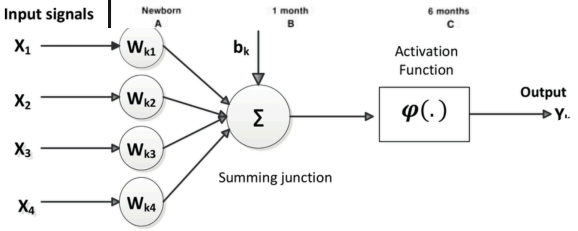
Logisitic Regression

Mc Culloch - Pitts model of a neuron









McCulloch-Pitts computational model of a neuron.

Activation function

The logisitic regression is a **binary classification** algorithm :

- output $\pi^{(i)}$ = the probability that $y^{(i)} \in \{0,1\}$

$$\pi^{(i)} = P(y^{(i)} = 1 | \underline{x}^{(i)}, \underline{\theta}) \in [0,1]$$

Activation function

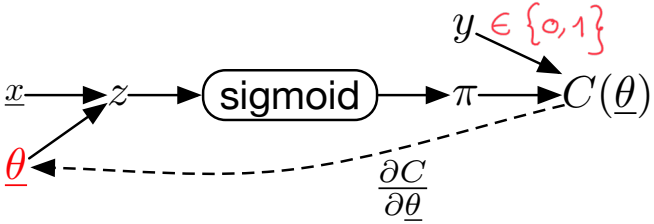
- **Sigmoid function** (Logisitic or logit function)

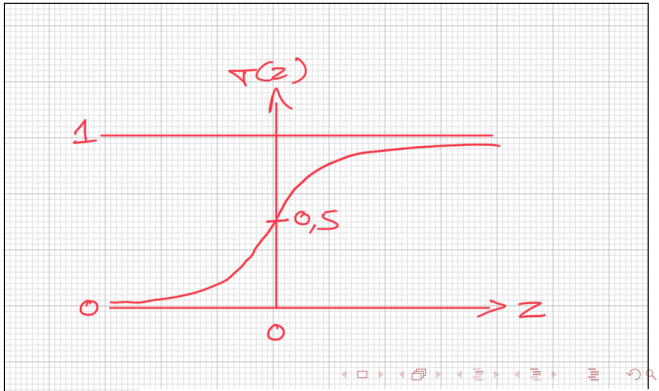
$$\pi = \sigma(z) = \boxed{\frac{1}{1+e^{-z}}} = \frac{e^z}{e^z+1}$$

  - with $z = \underline{x}\,\underline{\theta}$

- Derivative of sigmoid function

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$$
$$= \pi^{(i)}(1-\pi^{(i)})$$





Linear separating hyper-plane

The logisitic regression defines a linear separating hyper-plane

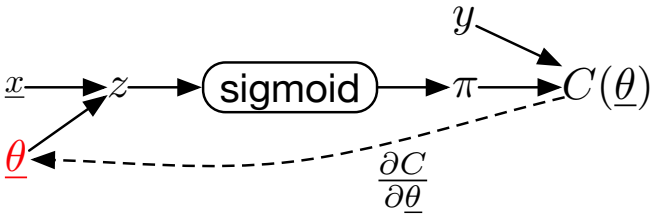
$$P(y^{(i)} = 1|\underline{x}^{(i)}, \underline{\theta}) = \frac{1}{2}$$

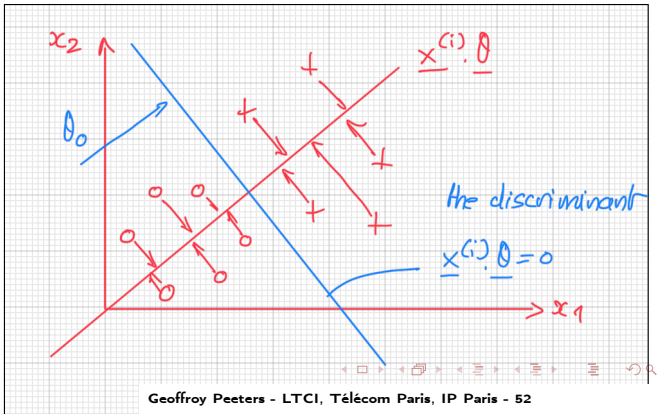
$$\sigma(z) = \frac{1}{2}$$

$$\frac{1}{1+e^{-z}} = \frac{1}{2}$$

$$z = \underline{x}^{(i)}\underline{\theta} = 0$$

$\underline{x}^{(i)}\underline{\theta} = 0$ is the equation of a plane





Cost function

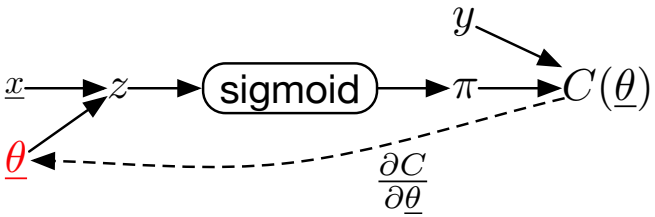
- For Linear Regression $\Rightarrow$ the uncertainty (random variable) is model by a Gaussian distribution
- For Logisitic Regression $\Rightarrow$ the uncertainty (random variable) is model by a Bernouilli distribution
- Learning is about minimizing uncertainty

Logisitic regression specifies the probability of a binary output $y^{(i)} \in \{0,1\}$ given input $\underline{x}^{(i)}$

$$p(y|\underline{x},\underline{\theta}) = \prod_{i=1}^{n} p(y^{(i)}|\underline{x}^{(i)},\underline{\theta}) = \prod_{i=1}^{n} Ber(y^{(i)}|\underline{x}^{(i)},\underline{\theta})$$
$$= \prod_{i=1}^{n} \left[\pi^{(i)}\right]^{y^{(i)}} \left[1-\pi^{(i)}\right]^{1-y^{(i)}}$$

- with $\pi^{(i)} = \frac{1}{1+e^{-z^{(i)}}}$ and $z^{(i)} = \underline{x}^{(i)}\underline{\theta}$

$y=1 \rightarrow \pi$, $y=0 \rightarrow (1-\pi)$ } $P(Y=y) = \pi^{y}(1-\pi)^{(1-y)}$



Cost to optimize (minimize)= Negative Log-Likelihood (NLL)

$$C(\underline{\theta}) = -\log p(y|\underline{x},\underline{\theta}) = \boxed{-\sum_{i=1}^{n} y^{(i)} \log \pi^{(i)} + (1-y^{(i)}) \log(1-\pi^{(i)})}$$

- this is called the binary cross-entropy $H(y^{(i)}, \pi^{(i)})$

Derivative of the cost w.r.t. to the output

$$\frac{\partial C(\underline{\theta})}{\partial \pi} = -\sum_{i=1}^{n} \left(\frac{y^{(i)}}{\pi^{(i)}} - \frac{(1-y^{(i)})}{1-\pi^{(i)}} \right) = \sum_{i=1}^{n} \frac{\pi^{(i)} - y^{(i)}}{\pi^{(i)}(1-\pi^{(i)})}$$

1) Optimization using Gradient

To minimize the cost $C(\underline{\theta})$, we compute the gradient of the $C(\theta)$ w.r.t. to the parameters $\underline{\theta}$

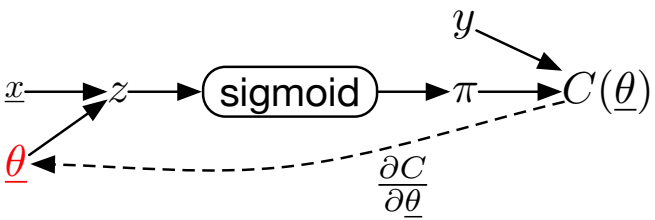
- we use the chain-rule :

$$\begin{aligned}\underline{g} &= \frac{\partial C(\underline{\theta})}{\partial \underline{\theta}} \\ &= \frac{\partial C(\underline{\theta})}{\partial \pi}\frac{\partial \pi}{\partial z}\frac{\partial z}{\partial \theta} \\ &= \sum_{i=1}^{n} \frac{\pi^{(i)} - y^{(i)}}{\pi^{(i)}(1-\pi^{(i)})} \cdot \pi^{(i)}(1-\pi^{(i)}) \cdot \underline{x}^{(i)} \\ &= \sum_{i=1}^{n} (\underline{x}^{(i)})^T (\pi^{(i)} - y^{(i)}) \\ &= \boxed{\underline{\underline{X}}^T (\underline{\pi} - \underline{y})}\end{aligned}$$

SGD update at iteration $k+1$:

$$\begin{aligned}\underline{\theta}_{k+1} &= \underline{\theta}_k - \eta_k \underline{g}_k \\ &= \boxed{\underline{\theta}_k - \eta_k \underline{\underline{X}}^T (\underline{\pi}_k - \underline{y})}\end{aligned}$$

where $\pi_k^{(i)} = \sigma(\underline{x}^{(i)} \underline{\theta}_k)$ at iteration k



Note : it looks a bit like the least-square solution :

$$\underline{\theta}_{k+1} = \underline{\theta}_k - \eta_k \left[2\underline{\underline{X}}^T (\underline{\underline{X}}\, \underline{\theta}_k - \underline{y}) \right]$$

2) Optimization using Hessian (Iteratively Reweighted Least Square - IRLS)

Newton update :

$$\theta_{k+1} = \boxed{\theta_k - \underline{\underline{H}}_k^{-1} \underline{g}_k}$$

- with Gradient, Hessian

$$\underline{g}_k = \underline{\underline{X}}^T (\underline{\pi}_k - \underline{y})$$

$$\underline{\underline{H}}_k = \underline{\underline{X}}^T \underline{\underline{S}}_k \underline{\underline{X}}$$

Proof

$$H = \frac{\partial}{\partial \underline{\theta}} g(\underline{\theta})^T$$

$$= \sum_{i=1}^{n} \pi^{(i)}(1-\pi^{(i)}) \underline{x}^{(i)} (\underline{x}^{(i)})^T$$

$$= \underline{\underline{X}}^T \, \mathrm{diag}(\pi^{(i)}(1-\pi^{(i)})) \underline{\underline{X}}$$

We note $\underline{\underline{S}}_k = \mathrm{diag}(\pi_k^{(1)}(1-\pi_k^{(1)}) \cdots \pi_k^{(n)}(1-\pi_k^{(n)}))$
H is positive definite, hence the NLL is convex and has a unique minimum

Newton update at iteration $k+1$:

$$\theta_{k+1} = \theta_k - H_k^{-1} g_k$$

$$= \theta_k + \left(\underline{\underline{X}}^T \underline{\underline{S}}_k \underline{\underline{X}}\right)^{-1} \underline{\underline{X}}^T (\underline{y} - \underline{\pi}_k)$$

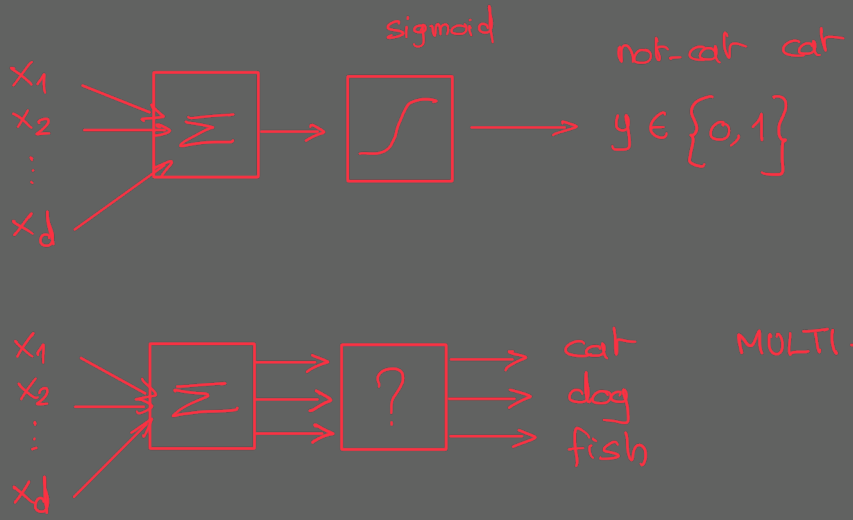
$$= \left(\underline{\underline{X}}^T \underline{\underline{S}}_k \underline{\underline{X}}\right)^{-1} \left[(\underline{\underline{X}}^T \underline{\underline{S}}_k \underline{\underline{X}})\theta_k + \underline{\underline{X}}^T (\underline{y} - \underline{\pi}_k)\right]$$

$$= \boxed{\left(\underline{\underline{X}}^T \underline{\underline{S}}_k \underline{\underline{X}}\right)^{-1} \underline{\underline{X}}^T \left[(\underline{\underline{S}}_k \underline{\underline{X}})\theta_k + (\underline{y} - \underline{\pi}_k)\right]}$$

Note : it looks a bit like the least-square solution :

$$\theta_{k+1} = \left(\underline{\underline{X}}^T \underline{\underline{S}}_k \underline{\underline{X}}\right)^{-1} \underline{\underline{X}}^T$$

Softmax regression



Activation function

The softmax regression is a **multi-class classification** algorithm :

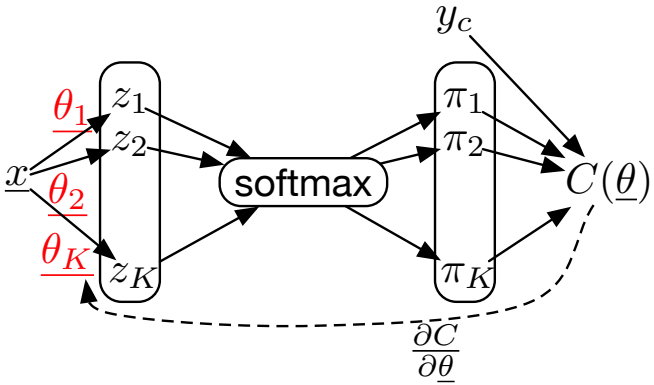
- output $\pi_c^{(i)}$ = probability that $y^{(i)} = c$
  $\pi_c^{(i)} = P(\underline{y}^{(i)} = c | \underline{x}^{(i)}, \underline{\theta}) \in [0,1]$

Acitvation function

- Softmax function

$$\pi_c^{(i)} = softmax(z_c^{(i)}) = \frac{e^{z_c^{(i)}}}{\sum_{c'=1}^{K} z_{c'}^{(i)}} = \frac{e^{\underline{x}^{(i)} \underline{\theta}_c}}{\sum_{c'=1}^{K} e^{\underline{x}^{(i)} \underline{\theta}_{c'}}}$$

- It is easy to show that
  - $\pi_1^{(i)} + \cdots + \pi_K^{(i)} = 1$



Cost function

Indicator function $\mathbb{1}_c(y^{(i)}) \triangleq \begin{cases} 1 & \text{if } y^{(i)} = c \\ 0 & \text{otherwise} \end{cases}$ **Likelihood**

function for Softmax

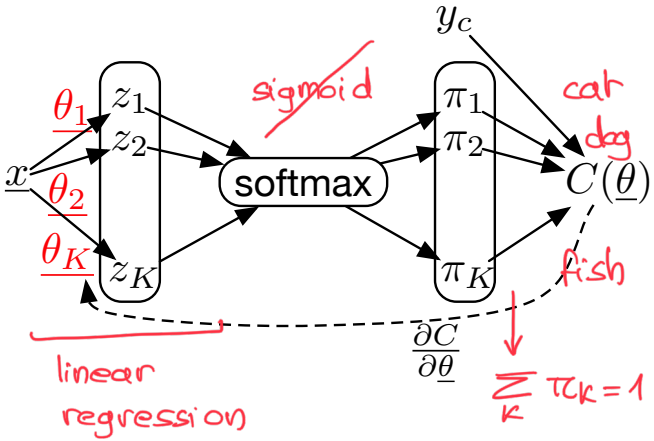
$$P(\underline{y}|\underline{x},\underline{\theta}) = \prod_{i=1}^{m} p(y^{(i)}|\underline{x}^{(i)},\underline{\theta})$$

$$= \prod_{i=1}^{m}\prod_{c=1}^{K} (\pi_c^{(i)})^{\mathbb{1}_c(y^{(i)})}$$

Cost to optimize : Negative Log-Likelihood (NLL) for Softmax

$$C(\theta) = -\log p(\underline{y}|\underline{x},\underline{\theta})$$

$$= -\sum_{i=1}^{m}\sum_{c=1}^{K} \mathbb{1}_c(y^{(i)}) \log(\pi_c^{(i)})$$



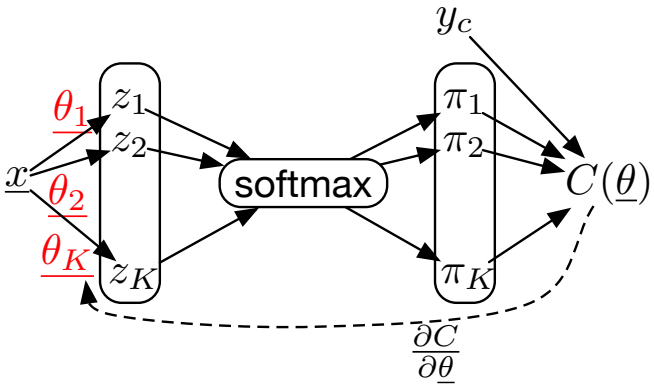
Cost function

For the specific case of 2 classes

- Likelihood

$$P(y^{(i)}=1|\underline{x}^{(i)},\underline{\theta}) = (\pi_1^{(i)})^1(\pi_2^{(i)})^0 = \pi_1^{(i)} = \frac{e^{\underline{x}^{(i)}\underline{\theta}_1}}{e^{\underline{x}^{(i)}\underline{\theta}_1}+e^{\underline{x}^{(i)}\underline{\theta}_2}}$$

$$P(y^{(i)}=2|\underline{x}^{(i)},\underline{\theta}) = (\pi_1^{(i)})^0(\pi_2^{(i)})^1 = \pi_2^{(i)} = \frac{e^{\underline{x}^{(i)}\underline{\theta}_2}}{e^{\underline{x}^{(i)}\underline{\theta}_1}+e^{\underline{x}^{(i)}\underline{\theta}_2}}$$



- Negative Log-Likelihood (NLL)

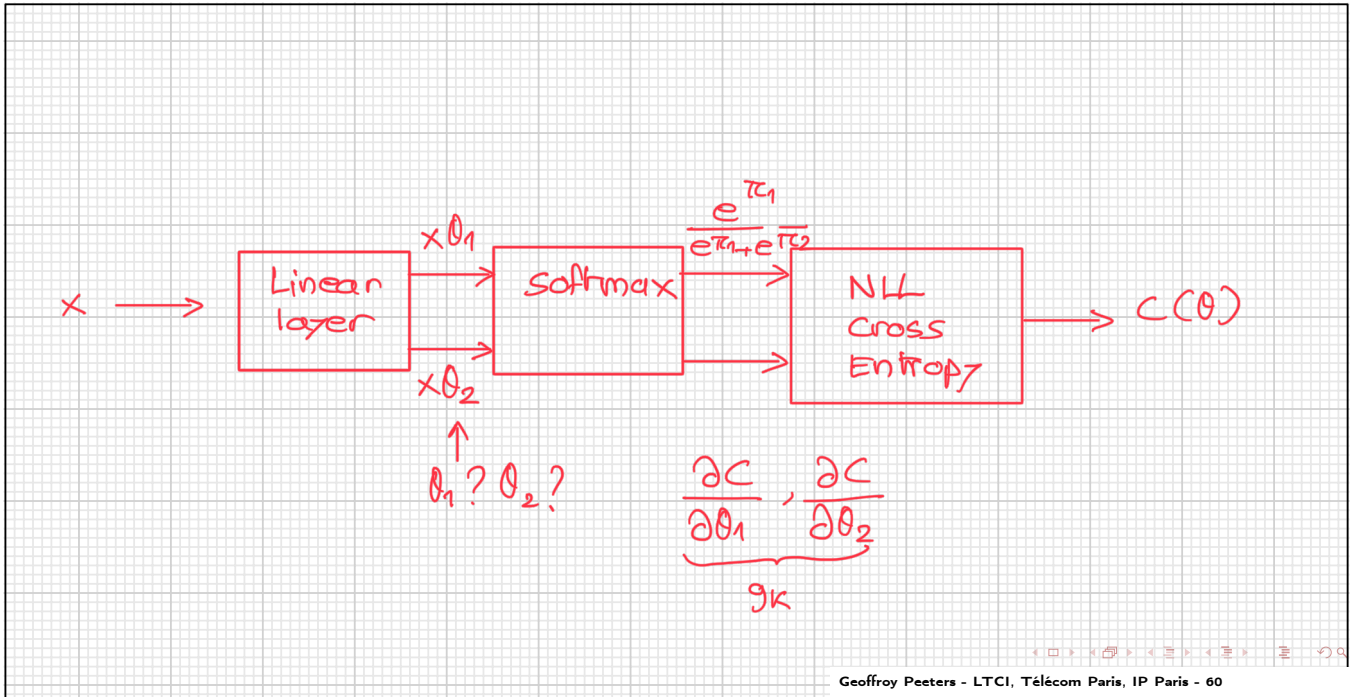
$$C(\theta) = -\log p(\underline{y}|\underline{x},\underline{\theta})$$

$$= -\sum_{i=1}^{m} \mathbb{1}_1(y^{(i)})\log(\pi_1^{(i)}) + \mathbb{1}_2(y^{(i)})\log(\pi_2^{(i)})$$

$$= -\sum_{i=1}^{m} \left[y_1 \log(\pi_1) + y_2 \log(\pi_2) + \dots \right]$$

Cost function

We can consider the Cost as a new layer



Optimization : derivative of the cost w.r.t. the parameters (manual)

For the specific case of 2 classes

$$\frac{\partial}{\partial \underline{\theta}_2} C(\underline{\theta}) = \frac{\partial}{\partial \underline{\theta}_2} \left(-\sum_i \mathbb{1}_1(y^{(i)}) \log(\pi_1^{(i)}) + \mathbb{1}_2(y^{(i)}) \log(\pi_2^{(i)}) \right)$$

$$= -\sum_i \mathbb{1}_1(y^{(i)}) \frac{\partial}{\partial \underline{\theta}_2} \log(\pi_1^{(i)}) + \mathbb{1}_2(y^{(i)}) \frac{\partial}{\partial \underline{\theta}_2} \log(\pi_2^{(i)})$$

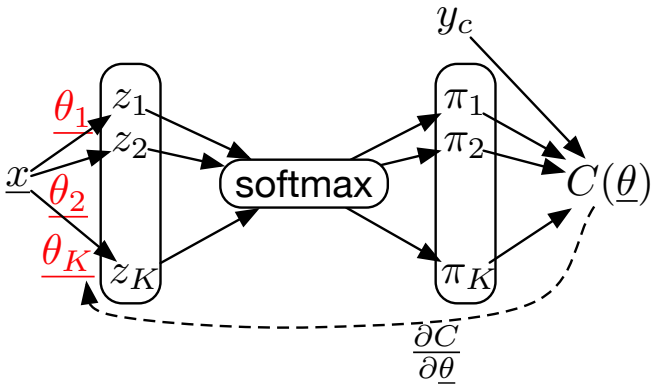
$$\frac{\partial}{\partial \underline{\theta}_2} \log(\pi_1^{(i)}) = \frac{\partial}{\partial \underline{\theta}_2} \log \frac{e^{\underline{x}^{(i)} \underline{\theta}_1}}{e^{\underline{x}^{(i)} \underline{\theta}_1} + e^{\underline{x}^{(i)} \underline{\theta}_2}}$$

$$= \frac{\partial}{\partial \underline{\theta}_2} \left(\underline{x}^{(i)} \underline{\theta}_1 - \log(e^{\underline{x}^{(i)} \underline{\theta}_1} + e^{\underline{x}^{(i)} \underline{\theta}_2}) \right)$$

$$= 0 - \frac{\underline{x}^{(i)} e^{\underline{x}^{(i)} \underline{\theta}_2}}{e^{\underline{x}^{(i)} \underline{\theta}_1} + e^{\underline{x}^{(i)} \underline{\theta}_2}}$$

$$= -\underline{x}^{(i)} \pi_2^{(i)}$$

$$\frac{\partial}{\partial \underline{\theta}_2} \log(\pi_2^{(i)}) = \cdots = \underline{x}^{(i)} (1 - \pi_2^{(i)})$$



- Therefore

$$\frac{\partial C(\underline{\theta})}{\partial \underline{\theta}_2} = -\sum_i \mathbb{1}_1(y^{(i)})(-\underline{x}^{(i)} \pi_2^{(i)}) + \mathbb{1}_2(y^{(i)}) \underline{x}^{(i)} (1 - \pi_2^{(i)})$$

Optimization : derivative of the cost w.r.t. the parameters (manual)

For the specific case of 2 classes

- $\Rightarrow$ equivalence with the logistic regression

Logistic regression $y^{(i)} \in \{0, 1\}$

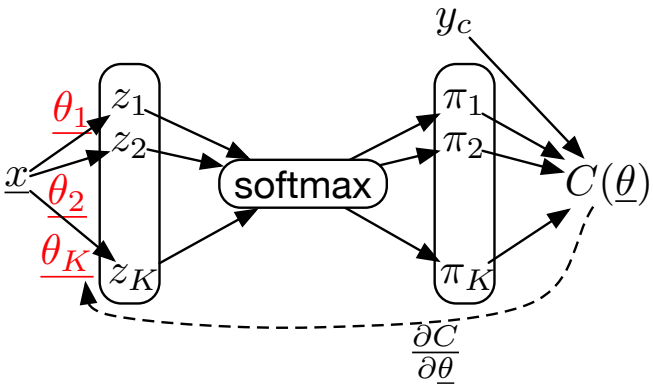
- using
  - $\mathbb{1}_1(y^{(i)}) = (1 - y^{(i)})$
  - $\mathbb{1}_2(y^{(i)}) = y^{(i)}$

$$\frac{\partial C(\underline{\theta})}{\partial \theta_2} = -\sum_i \mathbb{1}_1(y^{(i)})(-\underline{x}^{(i)}\pi_2^{(i)}) + \mathbb{1}_2(y^{(i)})\underline{x}^{(i)}(1 - \pi_2^{(i)})$$

$$= -\sum_i (1 - y^{(i)})(-\underline{x}^{(i)}\pi_2^{(i)}) + y^{(i)}\underline{x}^{(i)}(1 - \pi_2^{(i)})$$

$$= -\sum_i -\underline{x}^{(i)}\pi_2^{(i)} + y^{(i)}\underline{x}^{(i)}\pi_2^{(i)} + y^{(i)}\underline{x}^{(i)} - y^{(i)}\underline{x}^{(i)}\pi_2^{(i)}$$

$$= \sum_i \underline{x}^{(i)}(\pi_2^{(i)} - y^{(i)})$$



Backpropagation

Logistic Regression

Softmax

$$C(\theta) = -\sum_{i=1}^{n} \mathbb{1}_1(y^{(i)}) \log\left(\frac{e^{\theta_{1,:}\underline{x}^{(i)}}}{e^{\theta_{1,:}\underline{x}^{(i)}} + e^{\theta_{2,:}\underline{x}^{(i)}}}\right) + \mathbb{1}_2(y^{(i)}) \log\left(\frac{e^{\theta_{2,:}\underline{x}^{(i)}}}{e^{\theta_{1,:}\underline{x}^{(i)}} + e^{\theta_{2,:}\underline{x}^{(i)}}}\right)$$

We consider the logistic regression as a succession of layers $\underline{z}^{[l]}$

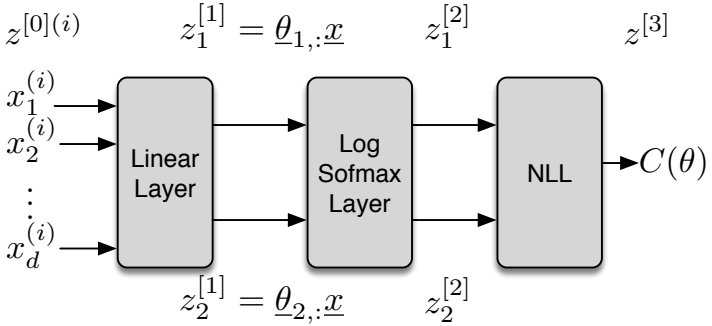
Softmax

$$\underline{z}^{[0]} = \underline{x}^{(i)}$$

$$z_1^{[1]} = \underline{\theta}_{1,:}\underline{z}^{[0]}$$
$$z_2^{[1]} = \underline{\theta}_{2,:}\underline{z}^{[0]}$$

$$z_1^{[2]} = \log\left(\frac{e^{z_1^{[1]}}}{e^{z_1^{[1]}} + e^{z_2^{[1]}}}\right)$$
$$z_2^{[2]} = \log\left(\frac{e^{z_2^{[1]}}}{e^{z_1^{[1]}} + e^{z_2^{[1]}}}\right)$$

$$z^{[3]} = -\sum_i \mathbb{1}_1(y^{(i)}) z_1^{[2]} + \mathbb{1}_2(y^{(i)}) z_2^{[2]}$$



Logistic Regression

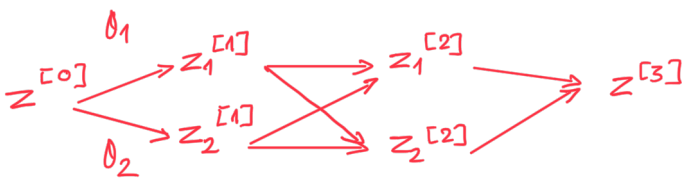
We consider the logistic regression as a succession of layers $\underline{z}^{[l]}$

- **Composition of functions**

$$C(\underline{\theta}) = z^{[3]}\left\{z_1^{[2]}\left[z_1^{[1]}(z^{[0]},\underline{\theta}_1), z_2^{[1]}(z^{[0]},\underline{\theta}_2)\right], z_2^{[2]}\left[z_1^{[1]}(z^{[0]},\underline{\theta}_1), z_2^{[1]}(z^{[0]},\underline{\theta}_2)\right]\right\}$$

- **Computing derivatives using the chain rule**
  - inefficient method : we compute several times the same things

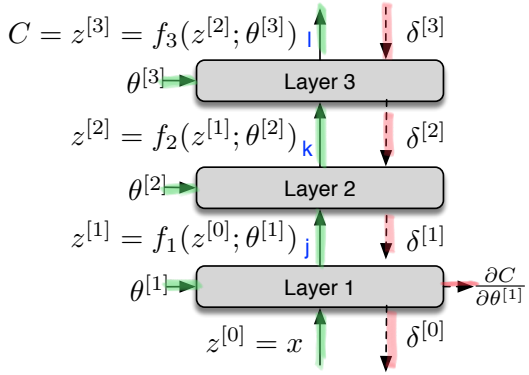
$$\frac{\partial C(\underline{\theta})}{\partial \theta_1} = \frac{\partial z^{[3]}}{\partial z_1^{[2]}}\frac{\partial z_1^{[2]}}{\partial z_1^{[1]}}\frac{\partial z_1^{[1]}}{\partial \theta_1} + \frac{\partial z^{[3]}}{\partial z_1^{[2]}}\frac{\partial z_1^{[2]}}{\partial z_2^{[1]}}\frac{\partial z_2^{[1]}}{\partial \theta_1} + \frac{\partial z^{[3]}}{\partial z_2^{[2]}}\frac{\partial z_2^{[2]}}{\partial z_1^{[1]}}\frac{\partial z_1^{[1]}}{\partial \theta_1} + \frac{\partial z^{[3]}}{\partial z_2^{[2]}}\frac{\partial z_2^{[2]}}{\partial z_2^{[1]}}\frac{\partial z_2^{[1]}}{\partial \theta_1}$$

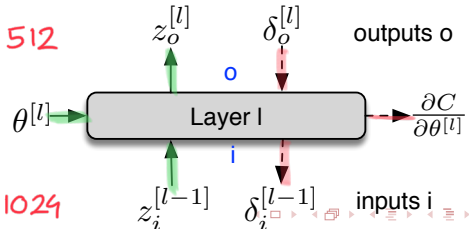


Layer specification

We only have to design a **module** with a

- **Forward method :**
  $z^{[l]} = f_l(z^{[l-1]}, \theta^{[l]})$
- **Backward method :**
  (for i inputs and o outputs)
  - Input :
    $$\boxed{\delta_o^{[l]} \triangleq \frac{\partial C}{\partial z_o^{[l]}}}$$
  - Computation
    $$\delta_i^{[l-1]} = \sum_o \frac{\partial C}{\partial z_o^{[l]}} \frac{\partial z_o^{[l]}}{\partial z_i^{[l-1]}}$$
    $$= \sum_o \delta_o^{[l]} \frac{\partial z_o^{[l]}}{\partial z_i^{[l-1]}}$$
    $$\frac{\partial C}{\partial \theta^{[l]}} = \sum_o \frac{\partial C}{\partial z_o^{[l]}} \frac{\partial z_o^{[l]}}{\partial \theta^{[l]}}$$
    $$= \sum_o \delta_o^{[l]} \frac{\partial z_o^{[l]}}{\partial \theta^{[l]}}$$





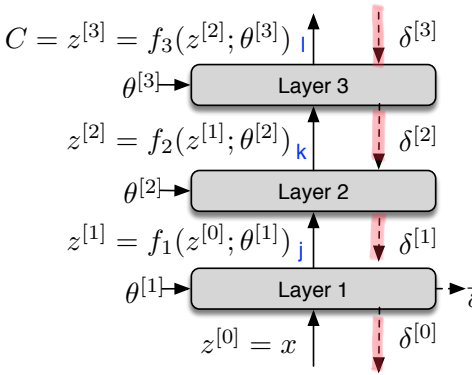
Derivative via layer-specification

$$\frac{\partial C}{\partial \theta^{[1]}} = \sum_j \frac{\partial C}{\partial z_j^{[1]}} \frac{\partial z_j^{[1]}}{\partial \theta^{[1]}}$$

$$= \sum_j \left(\sum_k \frac{\partial C}{\partial z_k^{[2]}} \frac{\partial z_k^{[2]}}{\partial z_j^{[1]}} \right) \frac{\partial z_j^{[1]}}{\partial \theta^{[1]}}$$

$$= \sum_j \left(\sum_k \left(\sum_l \frac{\partial C}{\partial z_l^{[3]}} \frac{\partial z_l^{[3]}}{\partial z_k^{[2]}} \right) \frac{\partial z_k^{[2]}}{\partial z_j^{[1]}} \right) \frac{\partial z_j^{[1]}}{\partial \theta^{[1]}}$$

$$= \sum_j \left(\sum_k \left(\sum_{l=1} 1 \frac{\partial z_l^{[3]}}{\partial z_k^{[2]}} \right) \frac{\partial z_k^{[2]}}{\partial z_j^{[1]}} \right) \frac{\partial z_j^{[1]}}{\partial \theta^{[1]}}$$



- Note : this is equivalent to what we have seen above but without redundant computation

$$\frac{\partial C(\theta)}{\partial \theta_1} = \frac{\partial z^{[3]}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial \theta_1} + \frac{\partial z^{[3]}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial z_2^{[1]}} \frac{\partial z_2^{[1]}}{\partial \theta_1} + \frac{\partial z^{[3]}}{\partial z_2^{[2]}} \frac{\partial z_2^{[2]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial \theta_1} + \frac{\partial z^{[3]}}{\partial z_2^{[2]}} \frac{\partial z_2^{[2]}}{\partial z_2^{[1]}} \frac{\partial z_2^{[1]}}{\partial \theta_1}$$

Forward $z^{[0]} = \underline{x}^{(i)} \rightarrow z^{[1]}(\underline{x}^{(i)}) \rightarrow z^{[2]}(\underline{x}^{(i)}) \rightarrow z^{[3]}(\underline{x}^{(i)}) = C$

Backward $\delta^{[0]} \leftarrow \delta^{[1]} \leftarrow \delta^{[2]} \leftarrow \delta^{[3]} = 1$

Neural Networks - Multi-Layer-Perceptron (MLP)

... with 1 layer (0 hidden layer) / Binary classification

- Layer 1 :

$$z^{[1](i)} = \theta_0^{[1]} + \theta_1^{[1]} x_1^{(i)} + \theta_2^{[1]} x_2^{(i)}$$

$$z^{[2](i)} = \frac{1}{1 + e^{-z^{[1](i)}}}$$

- Output

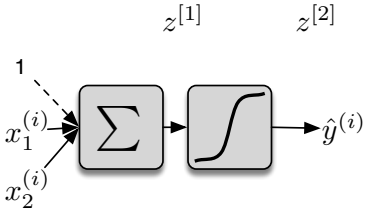
$$\hat{y}^{(i)} = z^{[2](i)}$$

$$\hat{y}^{(i)} = p(y^{(i)} | x^{(i)}, \underline{\theta})$$

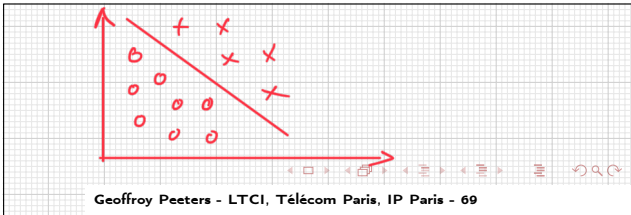
- Binary classification

$$p(y^{(i)} | x^{(i)}, \underline{\theta}) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1 - y^{(i)}}$$

$$= \begin{cases} \hat{y}^{(i)} & \text{when } y^{(i)} = 1 \\ 1 - \hat{y}^{(i)} & \text{when } y^{(i)} = 0 \end{cases}$$



- Linear separating hyper-plane



... with 2 layers (1 hidden layer) / Binary classification

- Layer 1 (hidden)

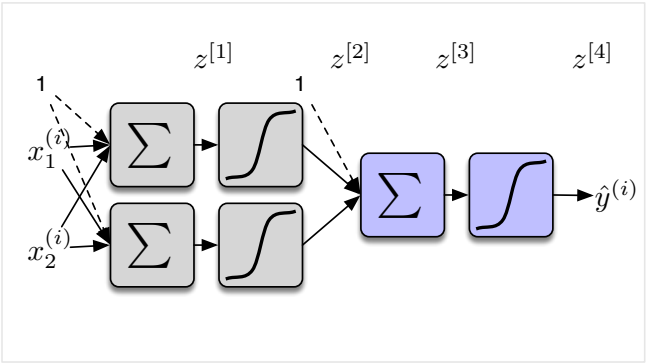
$$z_1^{[1]} = \theta_{0,1}^{[1]} + \theta_{1,1}^{[1]} x_1^{(i)} + \theta_{2,1}^{[1]} x_2^{(i)}$$
$$z_2^{[1]} = \theta_{0,2}^{[1]} + \theta_{1,2}^{[1]} x_1^{(i)} + \theta_{2,2}^{[1]} x_2^{(i)}$$
$$z_1^{[2]} = 1/(1 + e^{-z_1^{[1]}})$$
$$z_2^{[2]} = 1/(1 + e^{-z_2^{[1]}})$$

- Layer 2

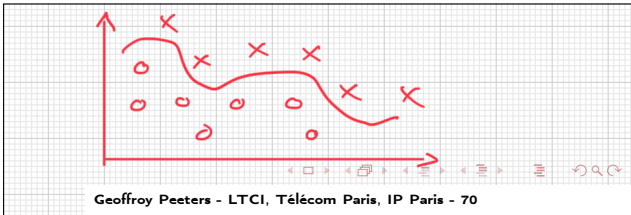
$$z_1^{[3]} = \theta_{0,1}^{[2]} + \theta_{1,1}^{[2]} z_1^{[2]} + \theta_{2,1}^{[1]} z_2^{[2]}$$
$$z^{[4]} = \frac{1}{1 + e^{-z_1^{[3]}}}$$

- Output

$$\hat{y}^{(i)} = z^{[4]}$$



- Non-Linear hyper-plane



- Cost : minimize the binary cross-entropy

$$C(\theta) = -\log p(y|x,\theta)$$
$$= -\sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

... with 2 layers (1 hidden layer) / Regression

Neural networks are function approximation tools

- Model

$$\hat{y}^{(i)} = \theta^{[2]}_{0,1} + \theta^{[2]}_{1,1} \frac{1}{1 - e^{-(\theta^{[1]}_{0,1} + \theta^{[1]}_{1,1} x^{(i)})}} + \theta^{[2]}_{2,1} \frac{1}{1 - e^{-(\theta^{[1]}_{0,2} + \theta^{[1]}_{1,2} x^{(i)})}}$$

- Output

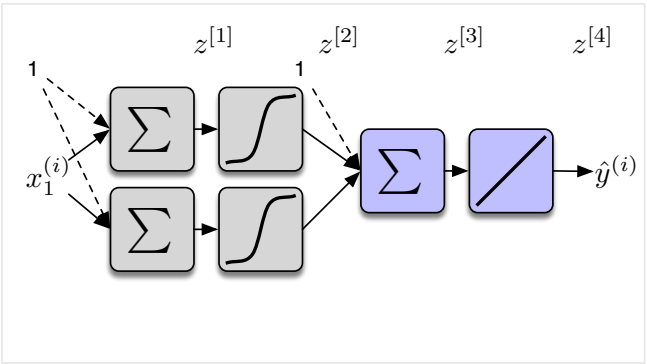
$$\hat{y}^{(i)} = \mathbb{E}(y^{(i)} | x^{(i)}, \theta)$$

- Likelihood/ Cost

$$p(y^{(i)} | x^{(i)}, \theta) = (2\pi\sigma^2)^{-1/2} e^{-\frac{1}{2\sigma^2}(\hat{y}^{(i)} - y^{(i)})^2}$$

$$C(\theta) = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

  - with $\hat{y}^{(i)} = f(x^{(i)}, \theta)$

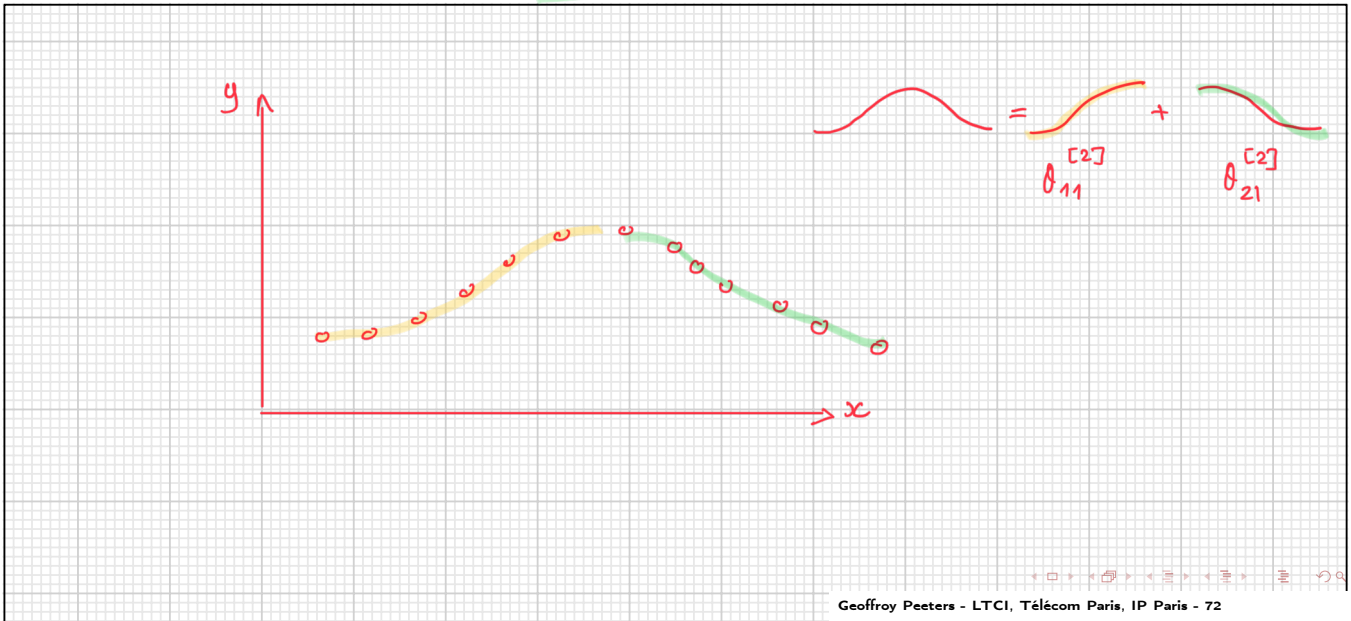


... with 2 layers (1 hidden layer) / Regression

$$\hat{y}^{(i)} = \theta^{[2]}_{0,1} + \theta^{[2]}_{1,1} \frac{1}{1 - e^{-(\theta^{[1]}_{0,1} + \theta^{[1]}_{1,1} x^{(i)})}} + \theta^{[2]}_{2,1} \frac{1}{1 - e^{-(\theta^{[1]}_{0,2} + \theta^{[1]}_{1,2} x^{(i)})}}$$

RBF regression

$$\|x - \mu_d\|^2$$



... with 2 layers (1 hidden layer) / Multi-Class classification

- Output :

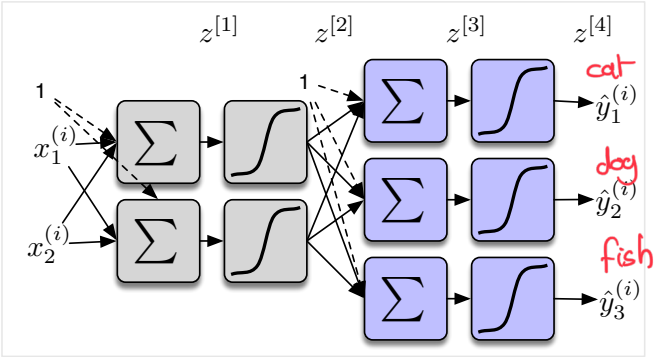
$$p(\underline{y}^{(i)} = [010]|x^{(i)}, \theta) = p(y^{(i)} = 2|x^{(i)}, \theta)$$
$$= softmax(\underline{z}_2^{(i)})$$
$$= \frac{e^{\underline{z}_2^{(i)}}}{e^{\underline{z}_1^{(i)}} + e^{\underline{z}_2^{(i)}} + e^{\underline{z}_3^{(i)}}}$$

- Lilelihood

$$p(y^{(i)}|x^{(i)}, \theta) =$$
$$\left[\frac{e^{\underline{z}_1^{(i)}}}{\sum}\right]^{\mathbb{1}_1(y^{(i)})} + \left[\frac{e^{\underline{z}_2^{(i)}}}{\sum}\right]^{\mathbb{1}_2(y^{(i)})} + \left[\frac{e^{\underline{z}_3^{(i)}}}{\sum}\right]^{\mathbb{1}_3(y^{(i)})}$$

- Cost : minimize the cross-entropy

$$C(\theta) = -\log p(y|x, \theta)$$
$$= -\sum_{i=1}^{n}\sum_{c=1}^{3} \mathbb{1}_c(y^{(i)}) \log \frac{e^{\underline{z}_c^{(i)}}}{\sum}$$

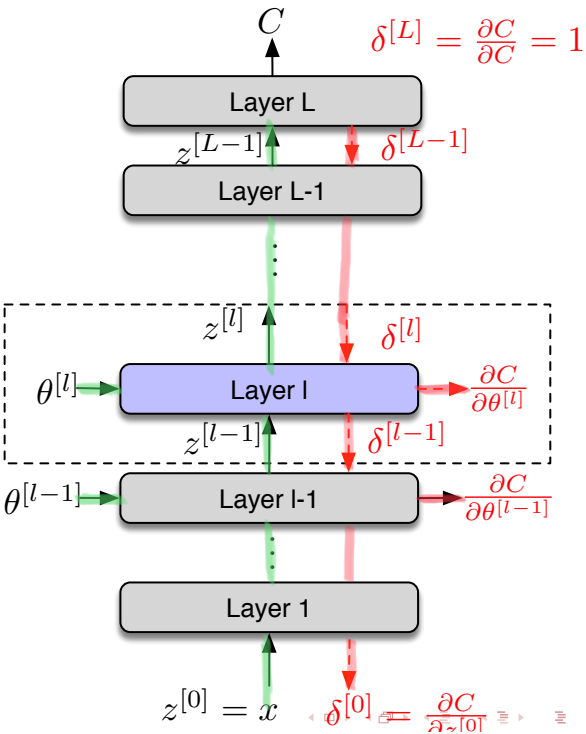


- Separating hyper-planes

Deep Learning

Deep Learning ?

- neural network with many many layers
- vanishing gradient problem (sigmoid, ReLu)
- sequential $\neq$ recursive



Forward/ Backward propagation

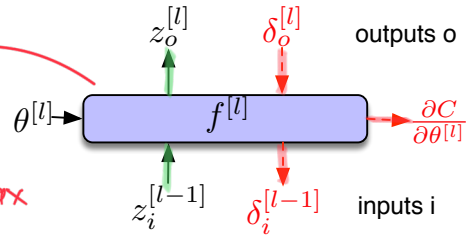
Layer l :

- Input : $\underline{z}^{[l-1]}$ / Output : $\underline{z}^{[l]}$

Forward :

$$z^{[l]} = f^{[l]}(z^{[l-1]}, \theta^{[l]})$$

- Linear $\underline{\theta} \times$
- ReLu
- Sigmoid, Softmax



Backward :

Gradients to propagate back

- matrix form

$$\underline{\delta}^{[l-1]} = \frac{\partial C}{\partial \underline{z}^{[l-1]}} = \frac{\partial C}{\partial \underline{z}^{[l]}} \frac{\partial \underline{z}^{[l]}}{\partial \underline{z}^{[l-1]}} = \underline{\delta}^{[l]} \frac{\partial \underline{z}^{[l]}}{\partial \underline{z}^{[l-1]}}$$

  - where $\frac{\partial \underline{z}^{[l]}}{\partial \underline{z}^{[l-1]}}$ is a Jacobian matrix

- scalar form

$$\delta_i^{[l-1]} = \sum_o \frac{\partial C}{\partial z_o^{[l]}} \frac{\partial z_o^{[l]}}{\partial z_i^{[l-1]}}$$

$$= \sum_o \delta_o^{[l]} \frac{\partial [f_o^{[l]}(\underline{z}^{[l-1]}, \underline{\theta}^{[l]})]}{\partial z_i^{[l-1]}}$$

Local gradients for parameters updating

- matrix form

$$\frac{\partial C}{\partial \underline{\theta}^{[l]}} = \frac{\partial C}{\partial \underline{z}^{[l]}} \frac{\partial \underline{z}^{[l]}}{\partial \underline{\theta}^{[l]}} = \underline{\delta}^{[l]} \frac{\partial \underline{f}^{[l]}(\underline{z}^{[l-1]}, \underline{\theta}^{[l]})}{\partial \underline{\theta}^{[l]}}$$

- scalar form

$$\frac{\partial C}{\partial \theta_i^{[l]}} = \sum_o \frac{\partial C}{\partial z_o^{[l]}} \frac{\partial z_o^{[l]}}{\partial \theta_i^{[l]}}$$

$$= \sum_o \delta_o^{[l]} \frac{\partial [f_o^{[l]}(\underline{z}^{[l-1]}, \underline{\theta}^{[l]})]}{\partial \theta_i^{[l]}}$$

Two ways to implement gradient computation

Suppose we have the following composition of functions

$$f^{[L]}(\underline{f}^{[L-1]}(\cdots \underline{f}^{[2]}(\underline{f}^{[1]}(\underline{x}))))$$

- **Solution 1**

$$\frac{\partial f^{[L]}(\underline{f}^{[L-1]}(\cdots \underline{f}^{[2]}(\underline{f}^{[1]}(\underline{x}))))}{\partial \underline{x}} = \frac{\partial f^{[L]}}{\partial \underline{f}^{[L-1]}} \cdot \frac{\partial \underline{f}^{[L-1]}}{\partial \underline{f}^{[L-2]}} \cdots \frac{\partial \underline{f}^{[2]}}{\partial \underline{f}^{[1]}} \cdot \frac{\partial \underline{f}^{[1]}}{\partial \underline{x}}$$

  - chain rule
  - need a lot of memory (need to store all matrices)

- **Solution 2**

$$\frac{\partial f^{[L]}(\underline{f}^{[L-1]}(\cdots \underline{f}^{[2]}(\underline{f}^{[1]}(\underline{x}))))}{\partial \underline{x}} = \left(\left(\left(\frac{\partial f^{[L]}}{\partial \underline{f}^{[L-1]}} \cdot \frac{\partial \underline{f}^{[L-1]}}{\partial \underline{f}^{[L-2]}}\right)\cdots\right)\frac{\partial \underline{f}^{[2]}}{\partial \underline{f}^{[1]}}\right) \cdot \frac{\partial \underline{f}^{[1]}}{\partial \underline{x}}$$

  - compute first (-) then second (-) then third (-)
  - less memory (the intermediate results are always vectors : vector x matrix = vector)
  - $\Rightarrow$ this is **the back-propagation algorithm**

Linear layer

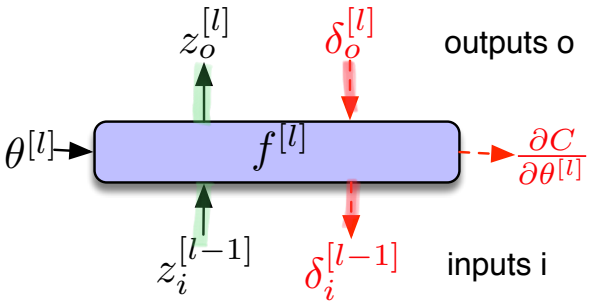
- **Forward**

$$z_o^{[l]} = f_o(\underline{z}^{[l-1]}, \underline{\theta}) = \sum_i z_i^{[l-1]} \theta_{io}$$

- **Backward**

$$\delta_i^{[l-1]} = \sum_o \delta_o^{[l]} \frac{\partial [f_o(\underline{z}^{[l-1]}, \underline{\theta})]}{\partial z_i^{[l-1]}} = \sum_o \delta_o^{[l]} \theta_{io}$$

$$\frac{\partial C}{\partial \theta_{io}} = \sum_o \delta_o^{[l]} \frac{\partial [f_o(\underline{z}^{[l-1]}, \underline{\theta})]}{\partial \theta_{io}} = \delta_o^{[l]} z_i^{[l-1]}$$



ReLu layer

- **Forward**

$$z_o = f_o(z_o^{[l-1]}) = \max(0, z_o^{[l-1]})$$

- **Backward**

$$\delta_i^{[l-1]} = \sum_o \delta_o^{[l]} \frac{\partial[f_o(z_o^{[l-1]})]}{\partial z_i^{[l-1]}}$$

$$= \delta_i^{[l]} \mathbb{1}_{[z_i^{[l-1]}>0]}$$