TSIA-203
Introduction to Deep Learning

Geoffroy Peeters, Stéphane Lathuilière

contact: geoffroy.peeters@telecom-paris.fr
Télécom-Paris, IP-Paris, France

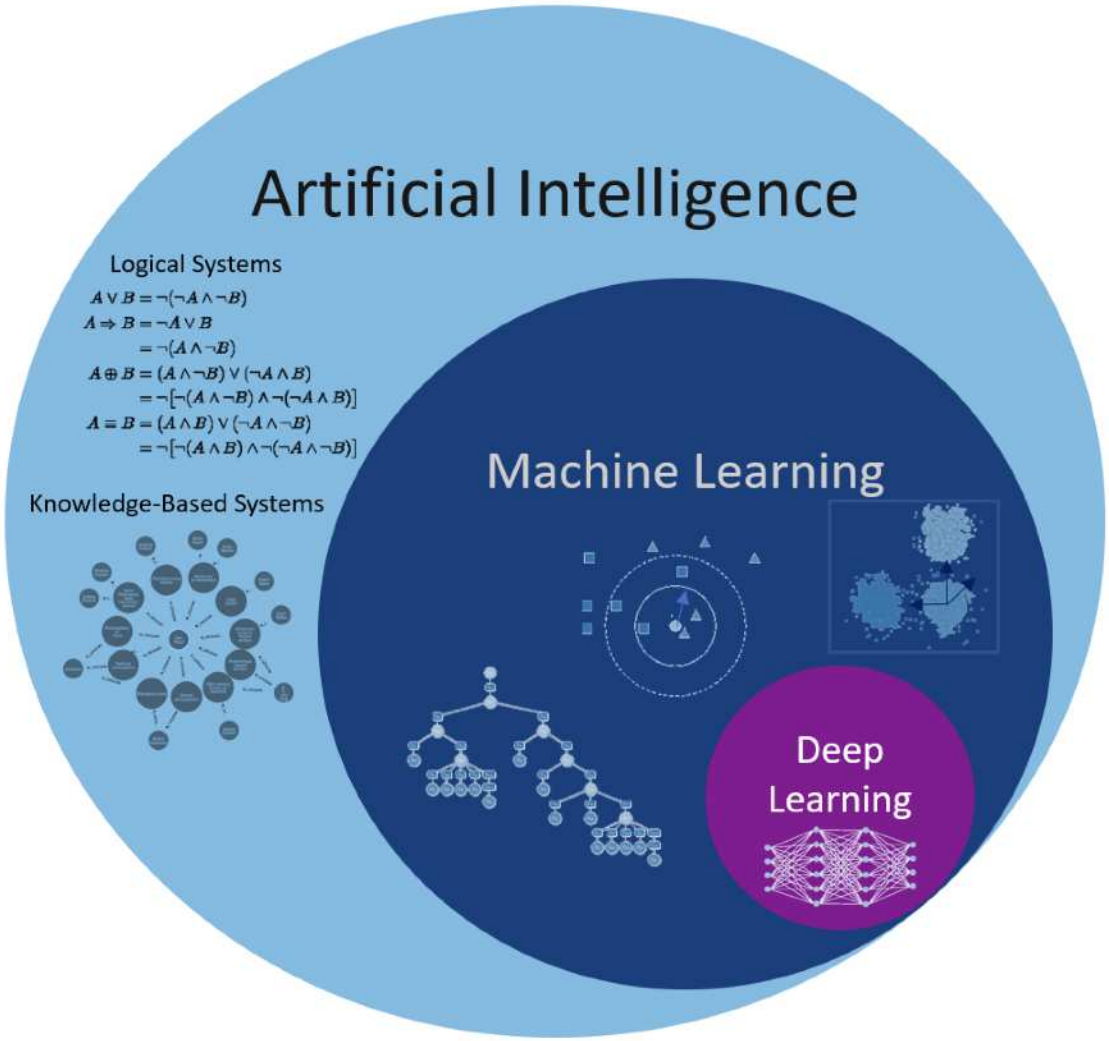
Planning

Date		Topic	Room	Speaker
24/04/2024 AM	Wednesday	Lecture MLP	Thévenin	PEETERS Geoffroy
02/05/2024 AM	Thursday	Lab MLP + Quizz	1A...	PEETERS Geoffroy et al.
15/05/2024 AM	Wednesday	Lecture CNN	Thévenin	Stéphane Lathuilière
29/05/2024 AM	Wednesday	Lab CNN + Quizz	1A...	Stéphane Lathuilière
05/06/2024 AM	Wednesday	Lecture RNN	Thévenin	PEETERS Geoffroy
12/06/2024 AM	Wednesday	Lab RNN + Quizz	1A...	PEETERS Geoffroy
19/06/2022	Wednesday	Lecture VAE	Thévenin	Stéphane Lathuilière
26/06/2022	mercredi	Examen	Thévenin/ Estaunié	

https://ecampus.paris-saclay.fr/course/view.php?id=22120

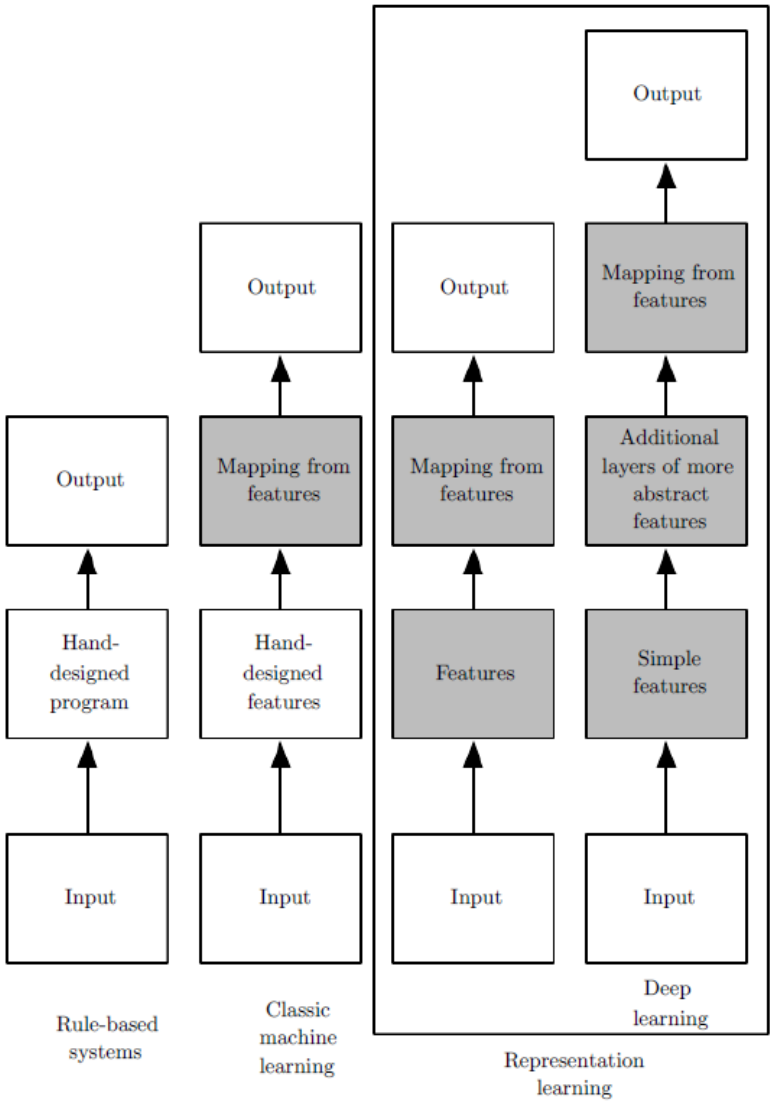
Deep Learning and Neural Networks : History

Deep learning (a subset of machine learning)



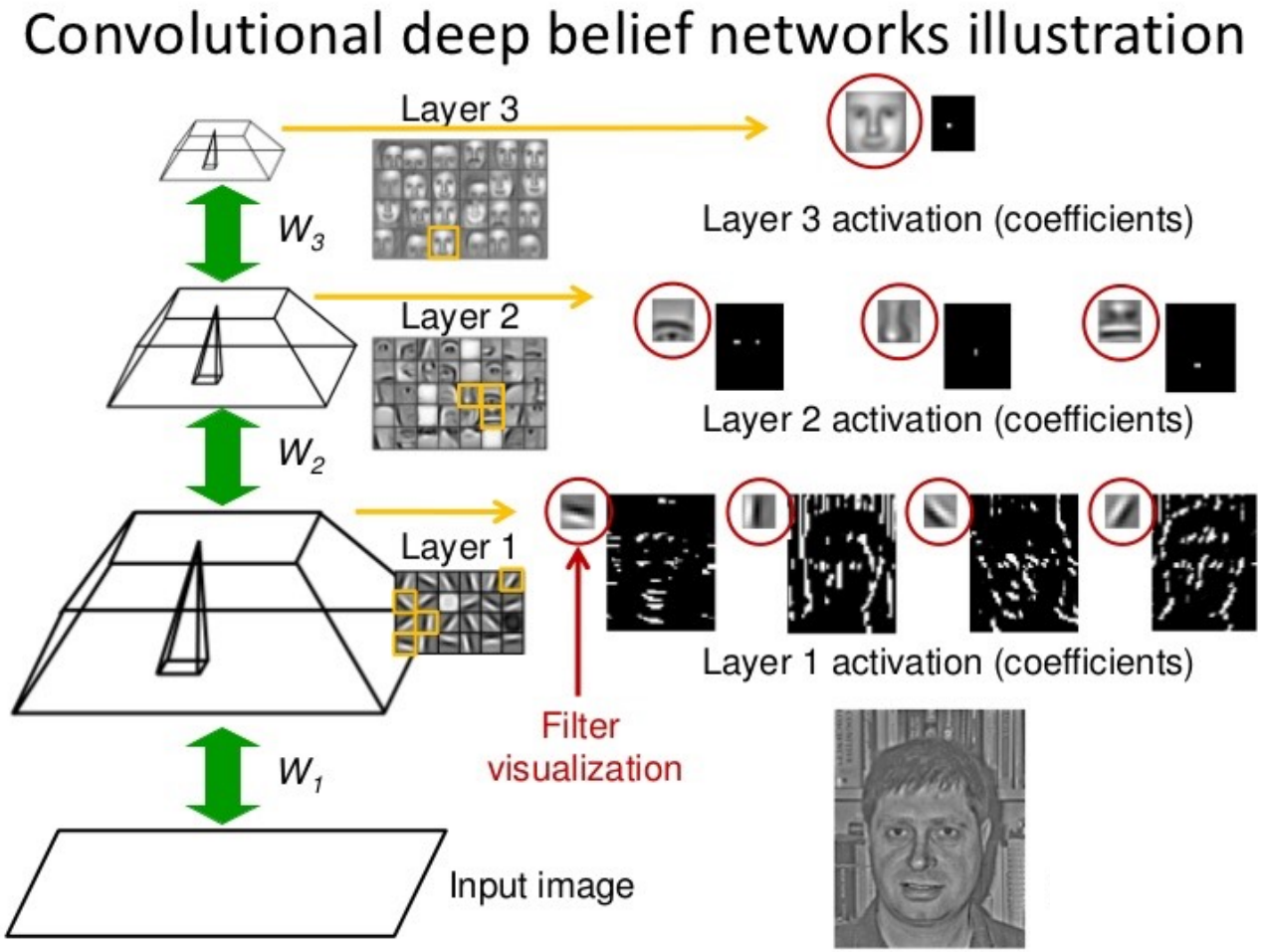
From http://canvart.club/atmose-11_21_13.html

Deep learning : learning hierarchical representations



From Deep Learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville

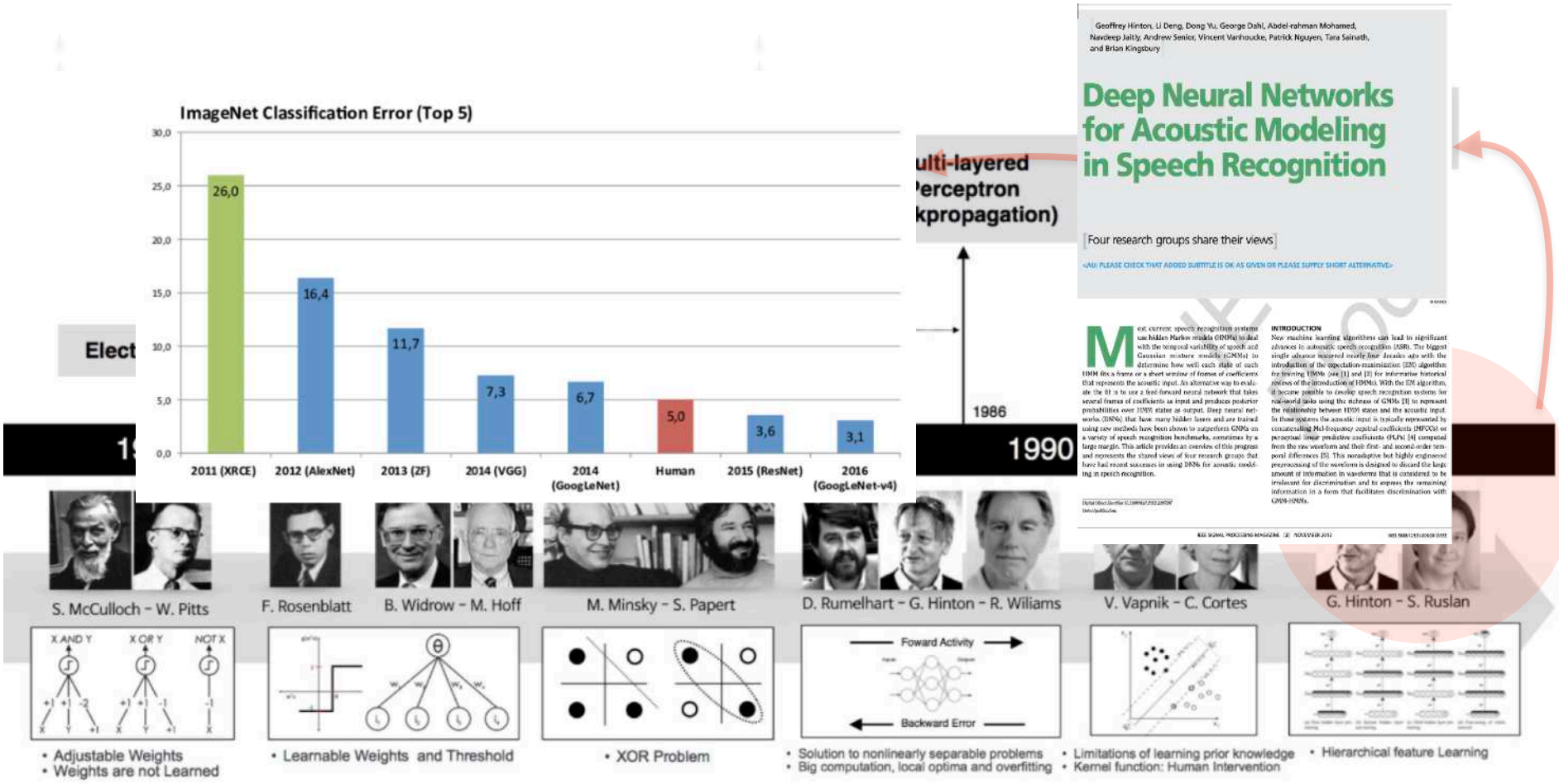
Feature learning examples



From Lee, Grosse, Ranganathn and Ng, "Convolutional Deep Belief Networks", ICML, 2009

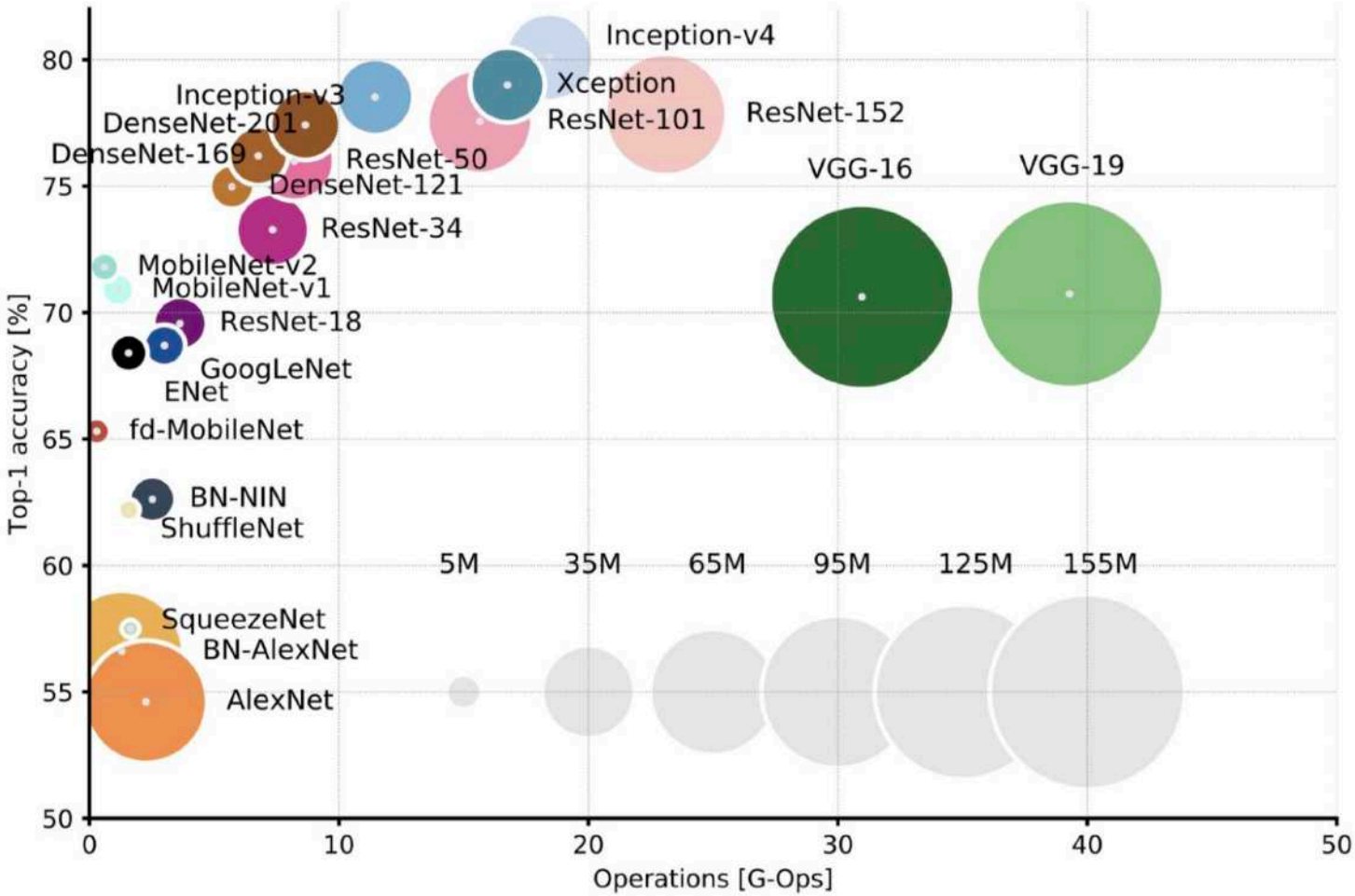
https://github.com/arthurmeyer/Convolutional_Deep_Belief_Network

Deep Learning and Neural Networks : History



https://stateoftther.github.io/post/deep_learning/deep-learning-history.html#2

Deep Learning and Neural Networks : History

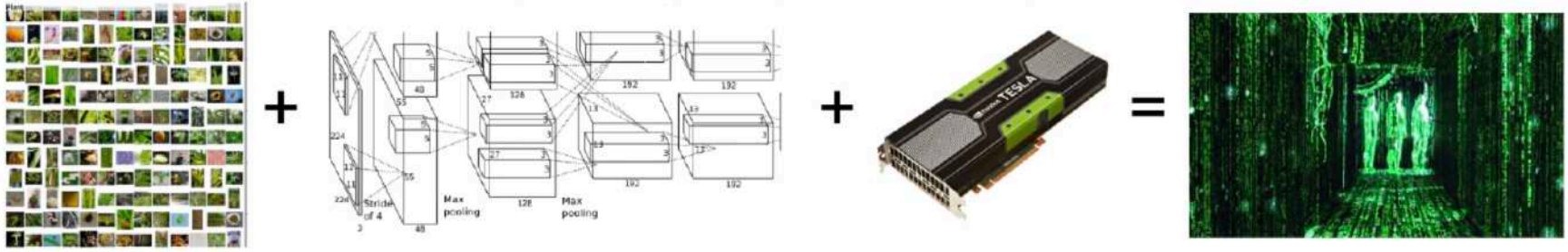


https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

Deep Learning and Neural Networks : History

***Deep Learning* =**
Lots of training data* + *Parallel Computation* + *Scalable, smart algorithms

The Deep Learning "Computer Vision Recipe"

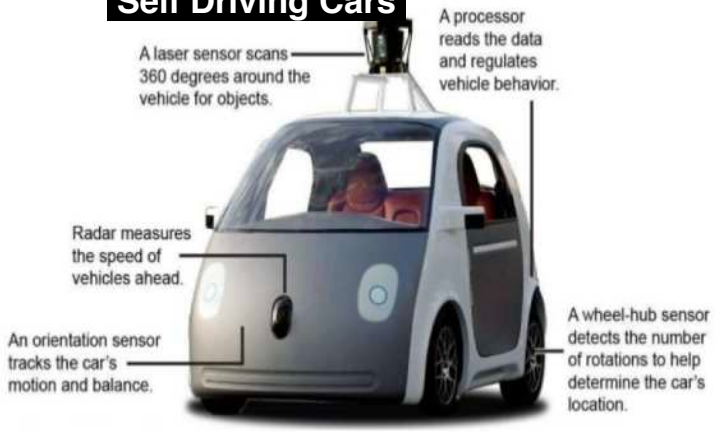


Big Data: ImageNet | Deep Convolutional Neural Network | Backprop on GPU | Learned Weights

Applications

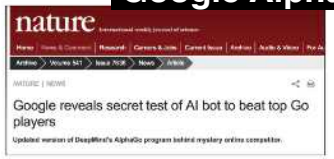
Automatic Speech Recognition

Self Driving Cars

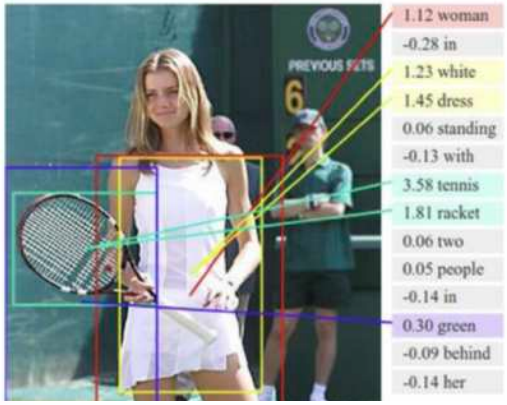


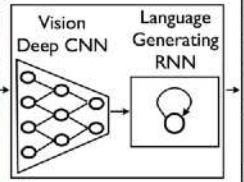
Google Alpha Go

DEEP LEARNING HAS MASTERED GO



Automatic picture captioning

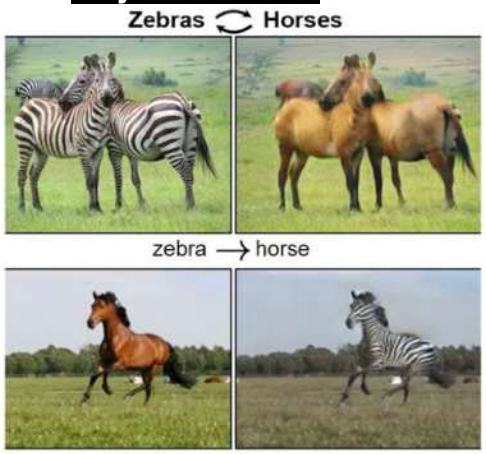


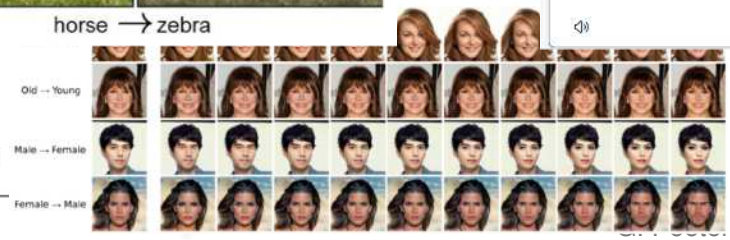


A group shopping at an outdoor market.

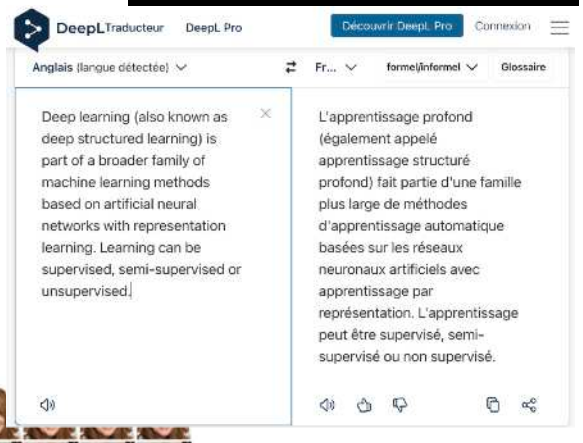
There are many vegetables at the fruit stand.

Style Transfer





Neural Machine Translation



Applications
Generative AI (GenAI)

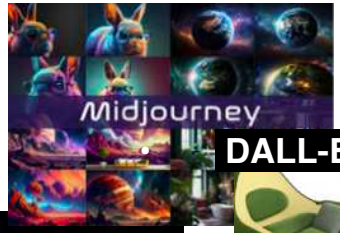
NLP

Video

https://www.youtube.com/watch?v=8LjfxFQ5iNk

Music

Computer Vision



DALL-E

Speech (deep fake)

Voice dubbing

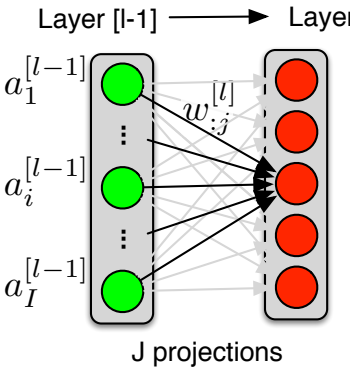
Roadmap

Roadmap

Architectures: *distinguish the way neurons are inter-connected*

Lecture 1

Fully Connected Neural Network

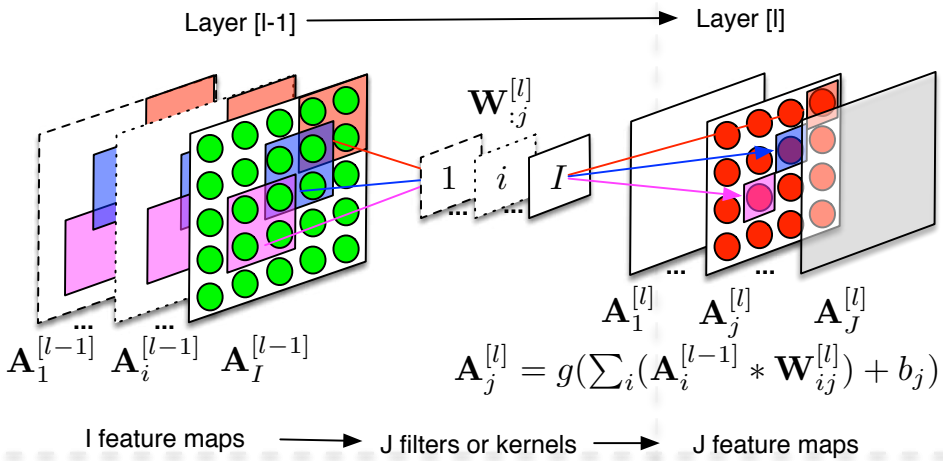


$$a_j^{[l]} = g(\sum_i a_i^{[l-1]} w_{ij}^{[l]} + b_j^{[l]})$$

$$\mathbf{a}^{[l]} = g(\mathbf{a}^{[l-1]}\mathbf{W}^{[l]} + \mathbf{b}^{[l]})$$

Lecture 2

Convolutional Neural Network

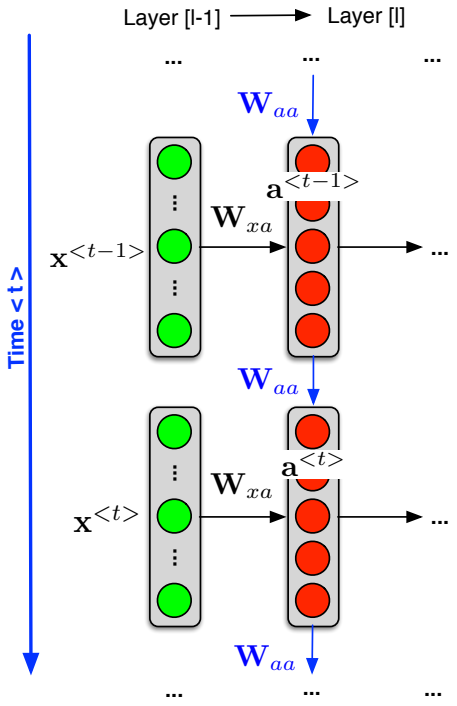


$$\mathbf{A}_j^{[l]} = g(\sum_i (\mathbf{A}_i^{[l-1]} * \mathbf{W}_{ij}^{[l]}) + b_j)$$

Lecture 3

Recurrent Neural Network

$$\mathbf{a}^{<t>} = g(\mathbf{x}^{<t>}\mathbf{W}_{xa} + \mathbf{a}^{<t-1>}\mathbf{W}_{aa} + \mathbf{b}_a)$$

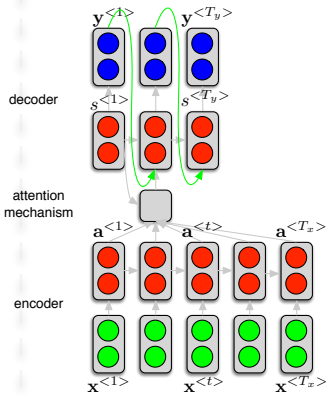


Roadmap

Meta-architectures: *the way architectures are plug to form a system*

Lecture 3

Encoder-Decoder with Attention



Transformer

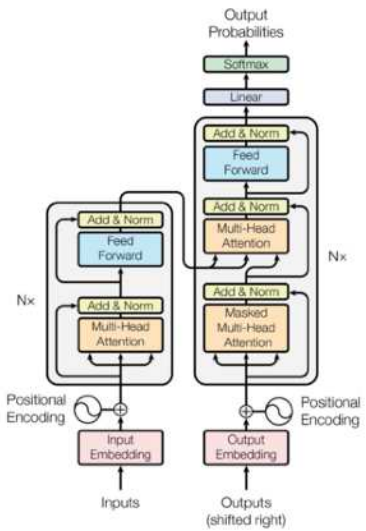
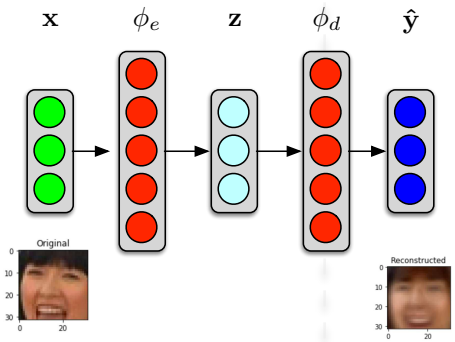


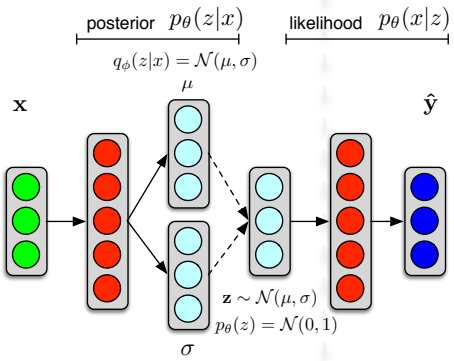
Figure 1: The Transformer - model architecture.

Lecture 4

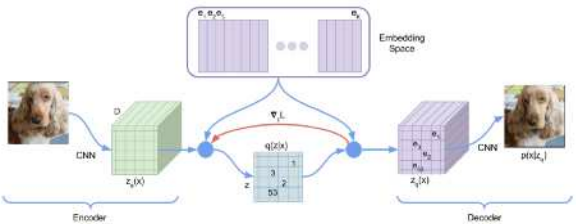
Auto-Encoder



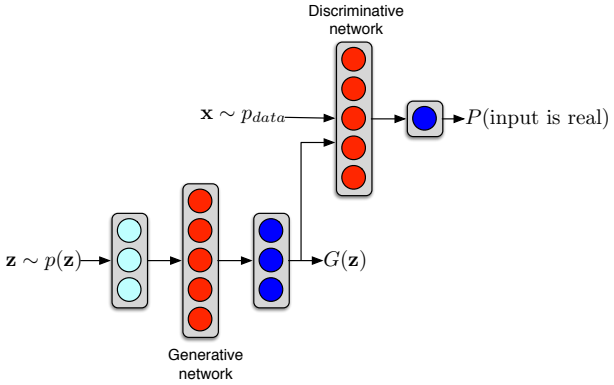
Variational-Auto-Encoder



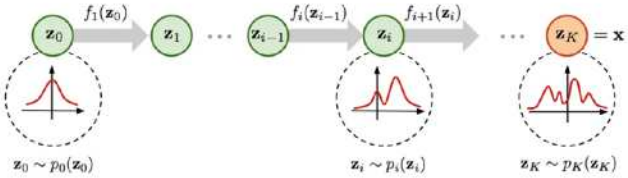
VQ-VAE



Generative Adversarial Network



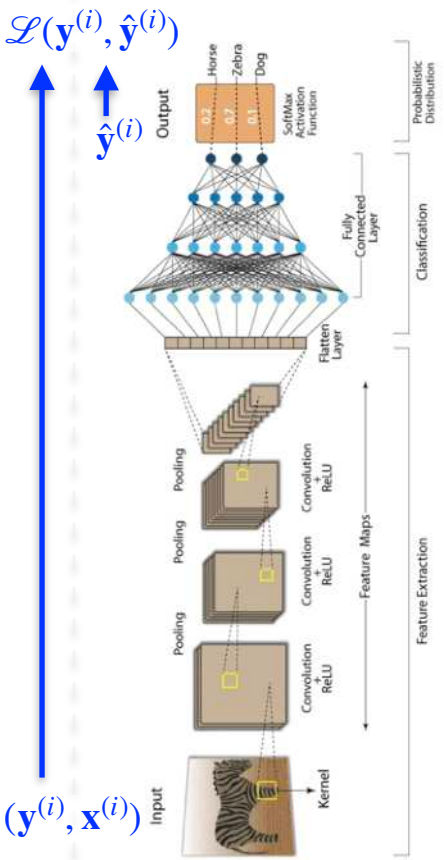
Normalizing FLows



Denoising Diffusion Models

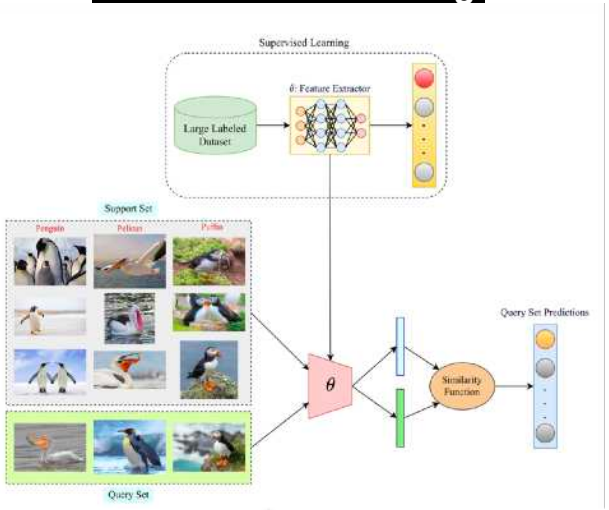
Roadmap
Training-paradigms: *the paradigm we use to train a system*

Supervised learning

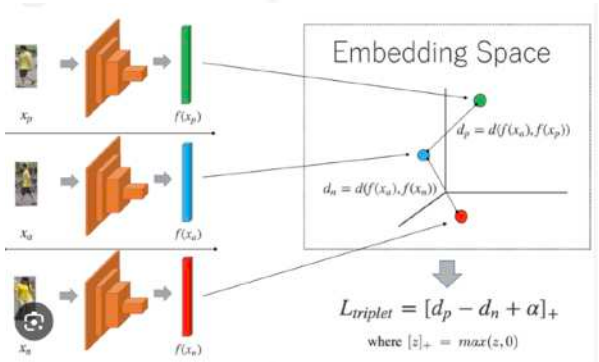


from https://larbifarihi.medium.com/classifying-cats-and-dogs-using-convolutional-neural-networks-cnns-ce6bc7bad64d

Zero/few Shot learning

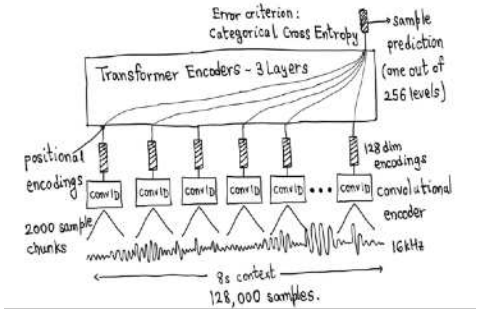


Metric learning

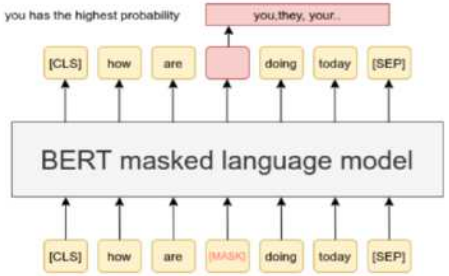


Self Supervised learning

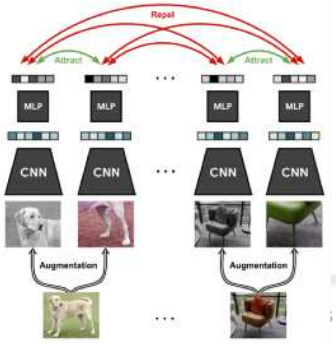
Auto-regressive (WaveNet, ChatGPT-



Masked LLM (BERT)



SimCLR



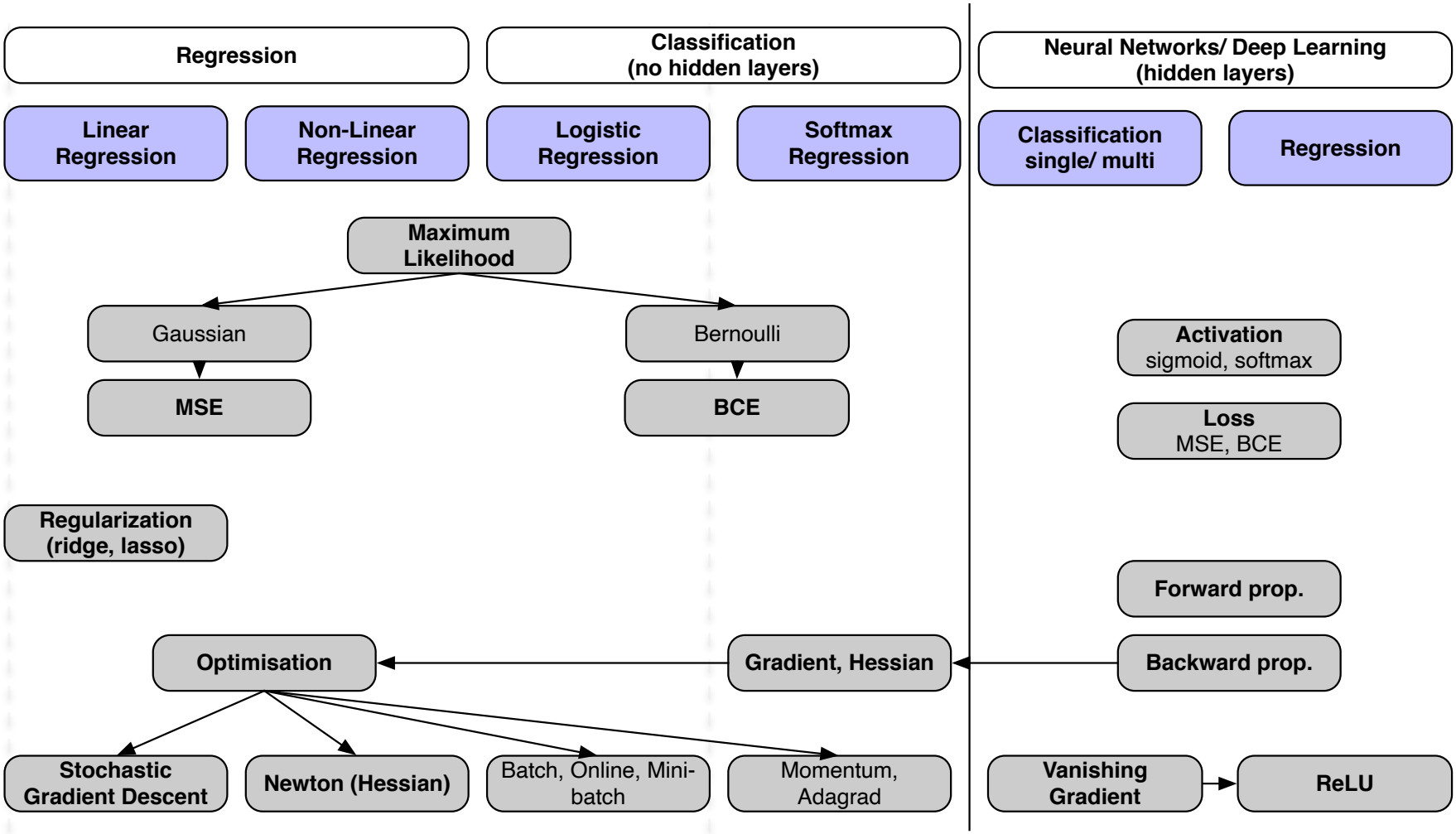
TSIA-203 Introduction to Deep Learning

Lecture 1: Multi-Layer-Perceptron

Geoffroy Peeters

contact: geoffroy.peeters@telecom-paris.fr
Télécom-Paris, IP-Paris, France

Overview

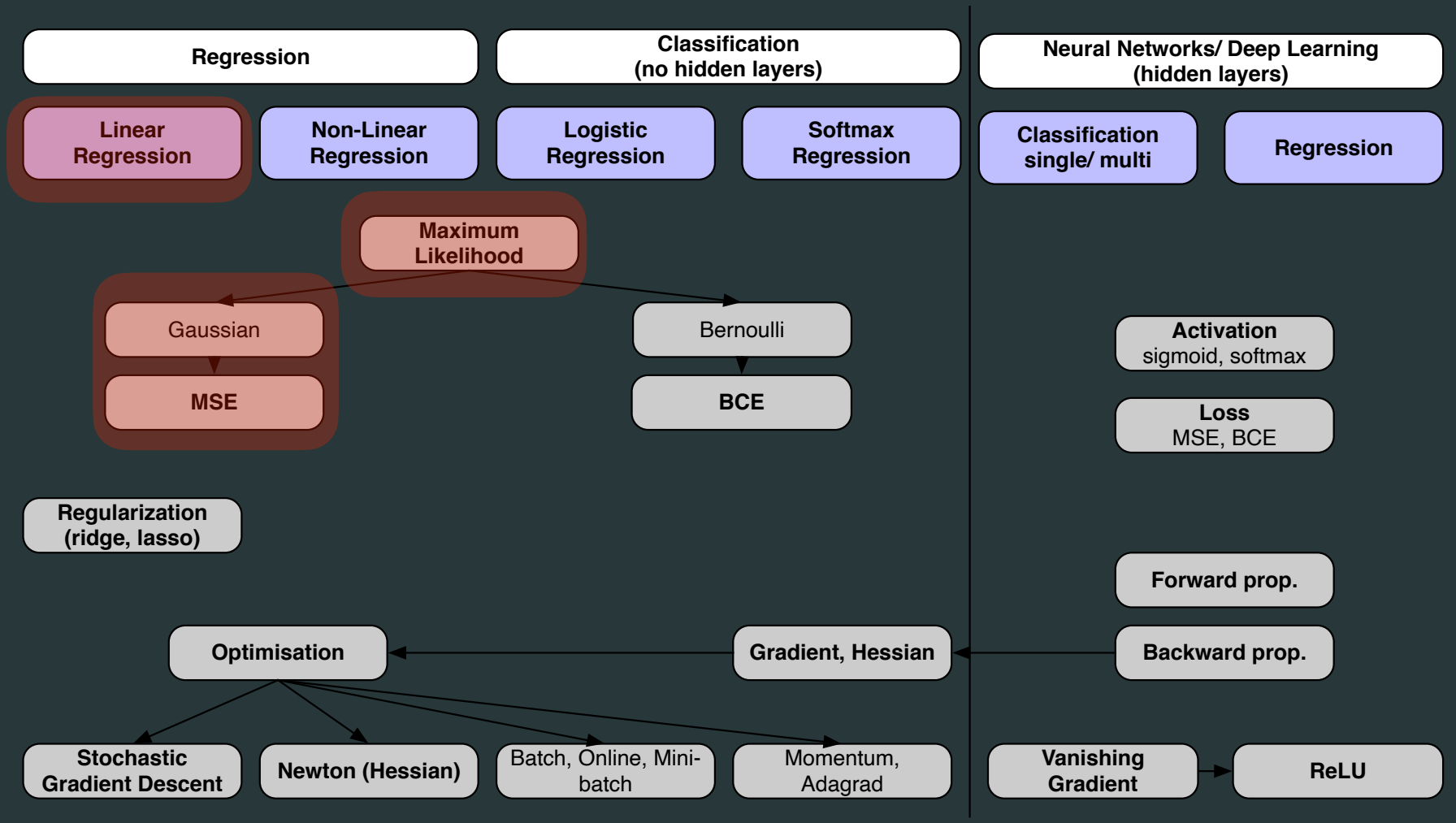

Regression
Classification
(no hidden layers)
Neural Networks/ Deep Learning
(hidden layers)
Linear Regression
Non-Linear Regression
Logistic Regression
Softmax Regression
Classification single/ multi
Regression
Maximum Likelihood
Gaussian
Bernoulli
MSE
BCE
Activation
sigmoid, softmax
Loss
MSE, BCE
Regularization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

Supervised classification

Supervised classification

- **Training/Learning**
  - **Given** a set of m input/output examples $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i\in\{1,\ldots,m\}}$
    - $\mathbf{x}^{(i)} \in \mathbb{R}^d$
    - $\mathbf{y}^{(i)}$
      - $y^{(i)} \in \mathbb{R}$ for regression
      - $y^{(i)} \in \{0,1\}$ for binary classification
      - $\mathbf{y}^{(i)} \in \{0,1\}^C$ for multi-class classification (C classes)
  - **Estimate** the parameters θ of a model f_θ
    - $\{\mathbf{x}^{(1:m)}, \mathbf{y}^{(1:m)}\} \rightarrow$ Learner $\rightarrow \theta$
- **Testing/ Prediction**
  - **Given** a new input $\mathbf{x}^{(m+1)}$ and the parameters θ of the model f_θ
  - **Predict** the output $\mathbf{y}^{(m+1)}$
    - $\{\mathbf{x}^{(m+1)}, \theta\} \rightarrow$ Prediction $\rightarrow \hat{\mathbf{y}}^{(m+1)}$

Linear regression



Linear regression
... 1 dimension

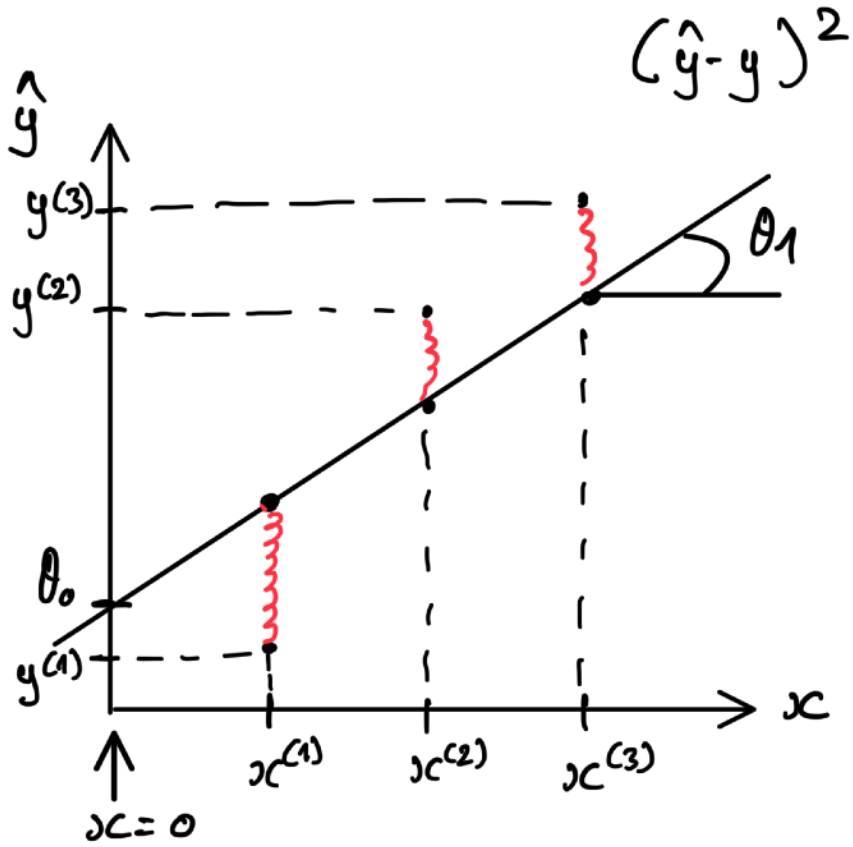
- **Model**

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

- **Objective function (energy, loss):**
  - Mean Square Error (MSE)

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

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



Linear regression
... d dimensions

- **Model**

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

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

- If we note $x_0^{(i)} = 1$ then $\mathbf{y}^{(i)} = \sum_{j=0}^{d} x_j^{(i)} \theta_j$

- In matrix form

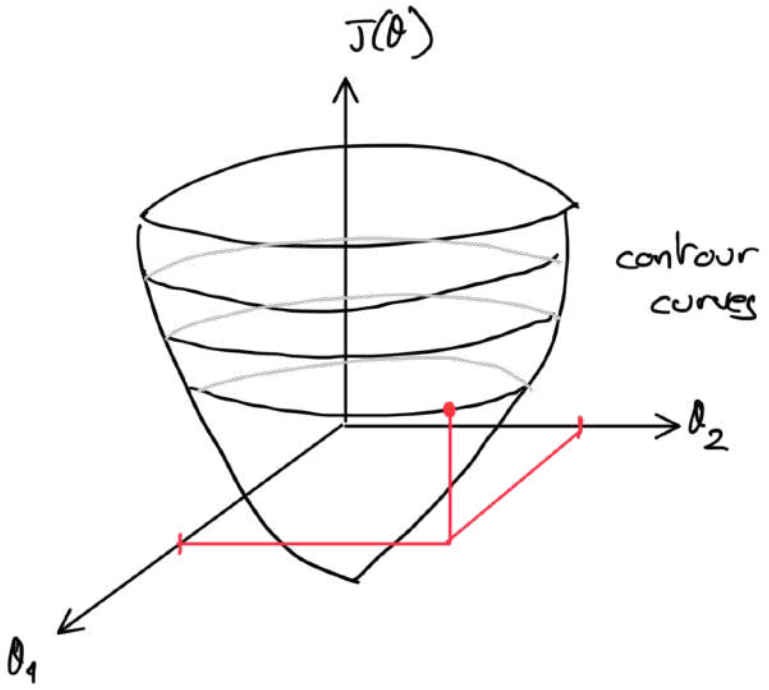
$$\underset{(\mathbb{R}^{(m,1)})}{\hat{\mathbf{y}}} = \underset{(\mathbb{R}^{(m,d+1)})}{\mathbf{X}} \quad \underset{(\mathbb{R}^{(d+1,1)})}{\theta}$$

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

- Objective function

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

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



- Find θ that minimises $J(\theta)$

$$\frac{\partial J(\theta)}{\partial \theta} = 0 = \frac{\partial}{\partial \theta}(\mathbf{y} - \mathbf{X}\,\theta)^T(\mathbf{y} - \mathbf{X}\,\theta)$$
$$= \frac{\partial}{\partial \theta}(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\,\theta + \theta^T\mathbf{X}^T\mathbf{X}\,\theta)$$
$$= 0 - 2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\theta$$
$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Results from matrix differentiation

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

Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation (MLE)
Example

- Suppose we have $n = 3$ data points
  - $y^{(1)} = 1, y^{(2)} = 0.5, y^{(3)} = 1.5$

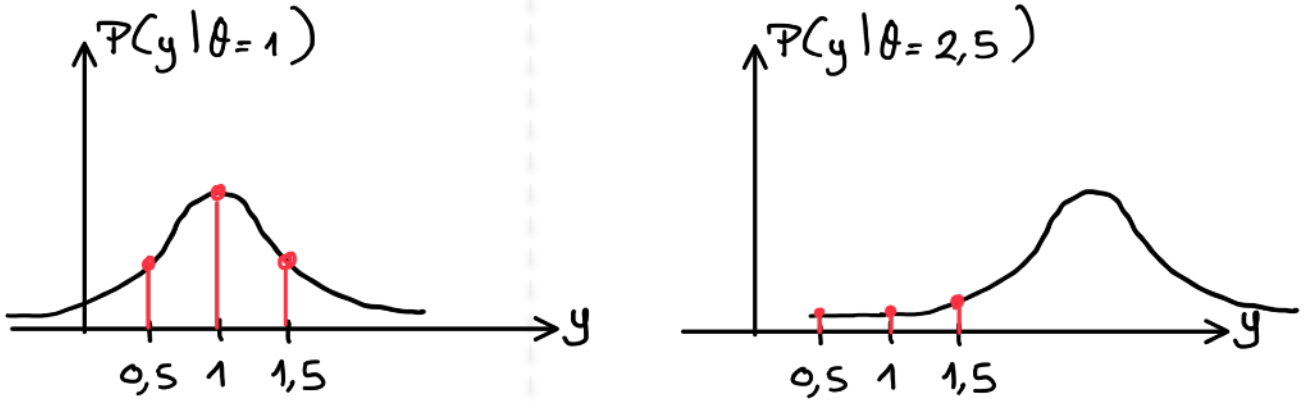
- which are independent samples from a Gaussian with unknown mean θ and variance 1

$$y^{(i)} \sim \mathcal{N}(\theta,1) = \theta + \mathcal{N}(0,1)$$

- Likelihood

$$p(y^{(1)}, y^{(2)}, y^{(3)}) | \theta) = p(y^{(1)} | \theta) \cdot p(y^{(2)} | \theta) \cdot p(y^{(3)} | \theta)$$

- Which guess of θ is more likely ?
  - $\theta = 1$
  - $\theta = 2.5$

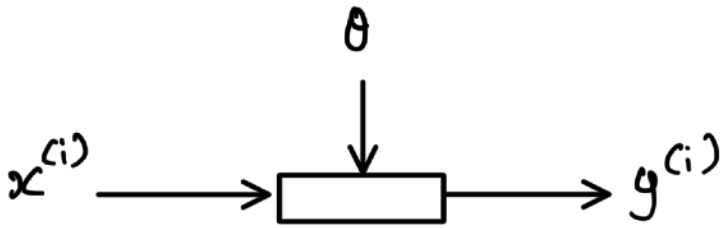


- We take the θ that maximises the likelihood

Maximum Likelihood Estimation (MLE)
Likelihood for linear regression

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

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



- The likelihood is

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

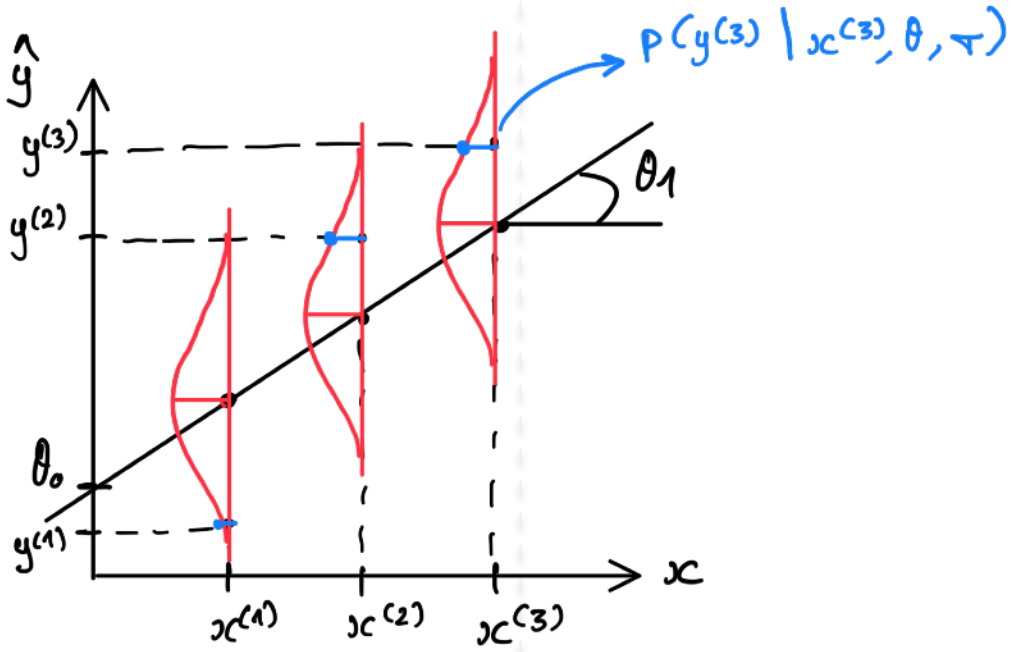
- This is equivalent to $p(\text{data}\,|\,\text{parameters}) = \frac{1}{Z} e^{-Loss(\text{data},\text{parameters})}$
  - with $p(\text{data}\,|\,\text{parameters}) = J(\theta) = \sum_{i=1}^{m} (y^{(i)} - \mathbf{x}^{(i)T}\theta)^2$
  - is the MSE (Mean Square Error) or Linear Regression

- **Maximising the likelihood is equivalent to minimising the MSE Loss !**

Maximum Likelihood Estimation (MLE)

- Example (back to linear regression):

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



Maximum Likelihood Estimation (MLE)
Likelihood for linear regression

- Maximising the **Likelihood** ⇒ minimising the **Negative Log-Likelihood** (NLL)

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

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

- MLE of θ (same solution as MSE)

$$\theta_{ML} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Proof

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

- MLE of σ

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

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

Proof

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

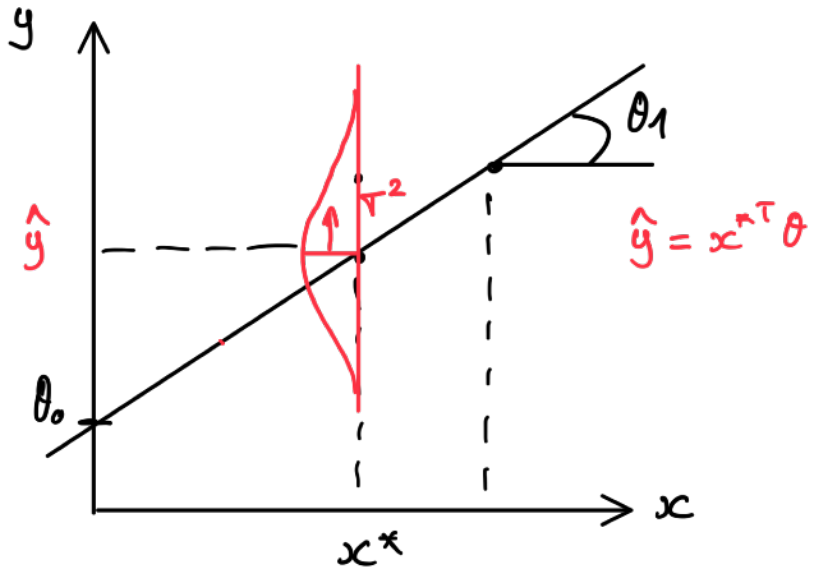
$$= m\frac{1}{\sigma} - \frac{1}{\sigma^3}(\mathbf{y}-\mathbf{X}\,\theta)^T(\mathbf{y}-\mathbf{X}\,\theta)$$

Maximum Likelihood Estimation (MLE)

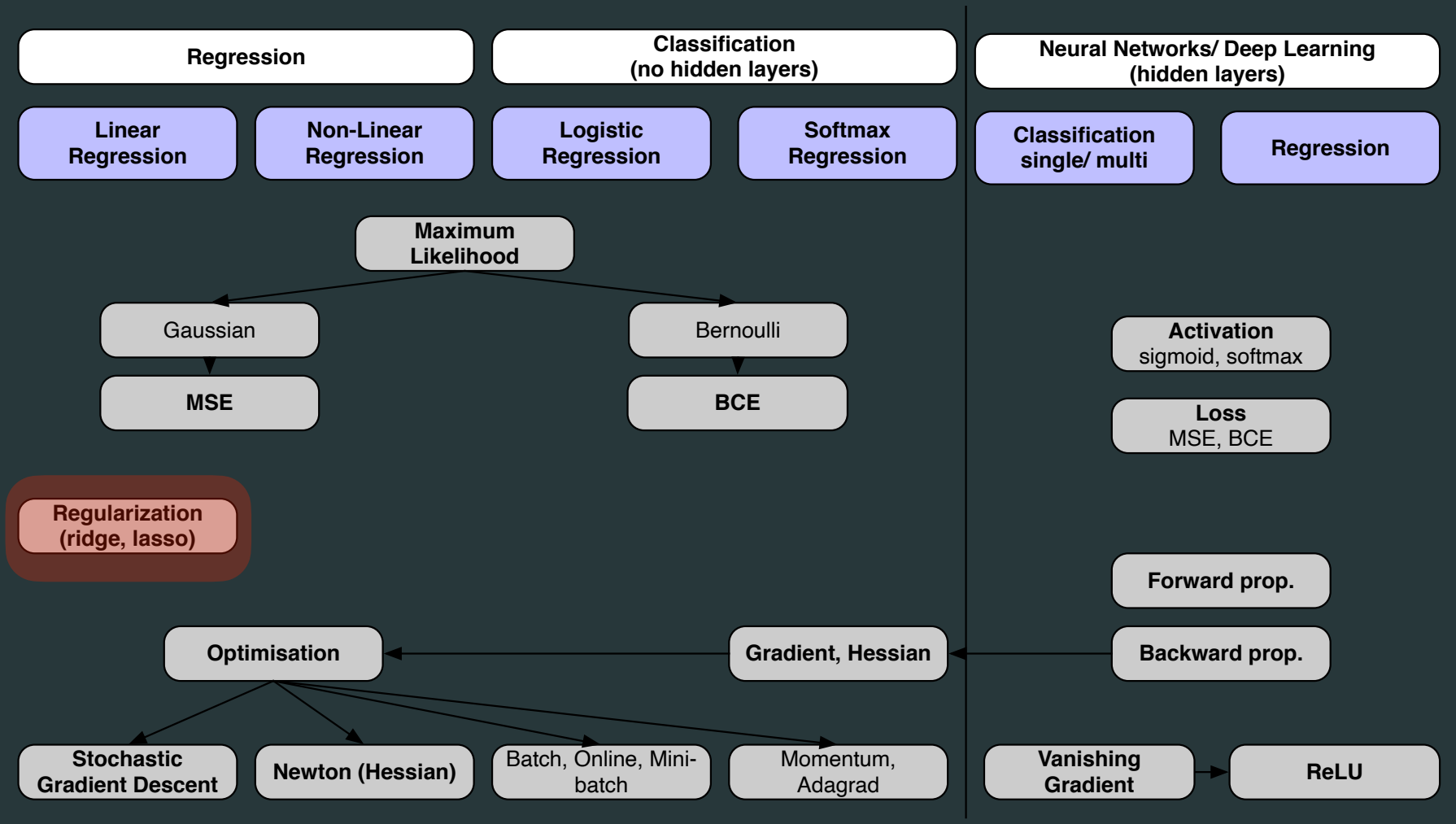
- **Making predictions:**
  - the ML prediction for a new input x^*, given the training data D, and known σ^2 is given by

$$p(y|x^*, D, \sigma^2) = \mathcal{N}(y|x^{*T}\theta_{ML}, \sigma^2)$$

  - it provides a **confidence** in prediction !



Regularization : Ridge, Lasso



Regularization : Ridge, Lasso

- **Least-square regression**

$$J(\theta) = (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta)$$
$$\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

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

- **Ridge regression**

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

  - Corresponds to the regularised quadratic cost function
$$J(\theta) = (\mathbf{y} - \mathbf{X}\,\theta)^T(\mathbf{y} - \mathbf{X}\,\theta) + \delta^2\theta^T\theta$$

  - equivalent to $\min_{\theta:\theta^T\theta \leq t(\delta)} (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta)$
    - where t is an arbitrary function
    - also named weight decay

Proof

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

Regularization : Ridge, Lasso

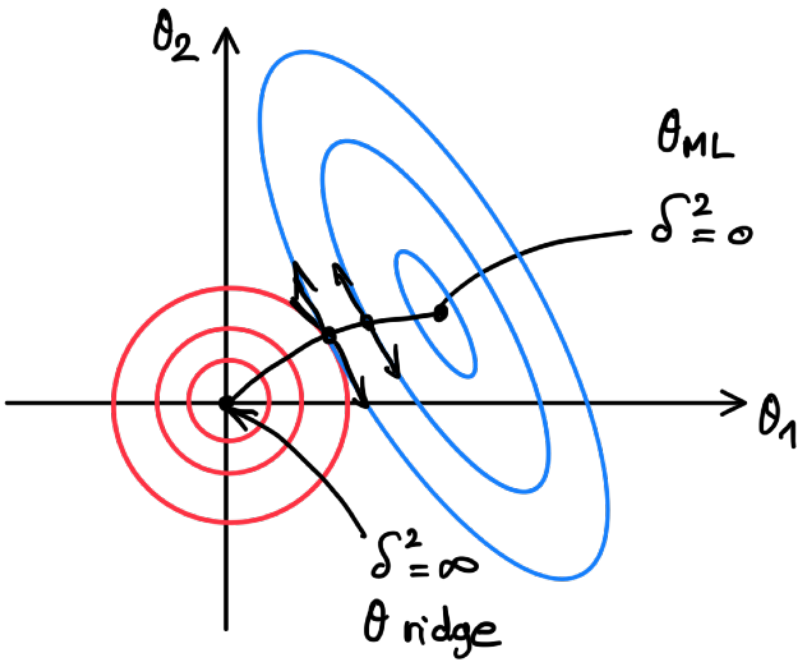
- **Example:**

$$\theta^T = [\theta_1, \theta_2]$$

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

$$= \theta_1^2 + \theta_2^2 = \textbf{const}$$

  - equation of a circle
  - automatically remove irrelevant axes



Regularization : Ridge, Lasso

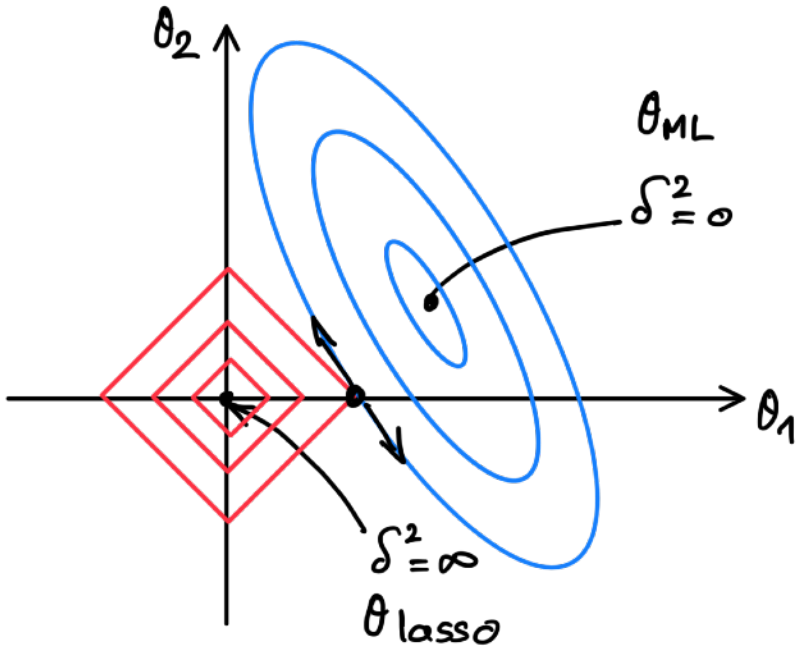
- **Ridge:**

$$\|\theta\|_2^2 = \theta^T\theta$$

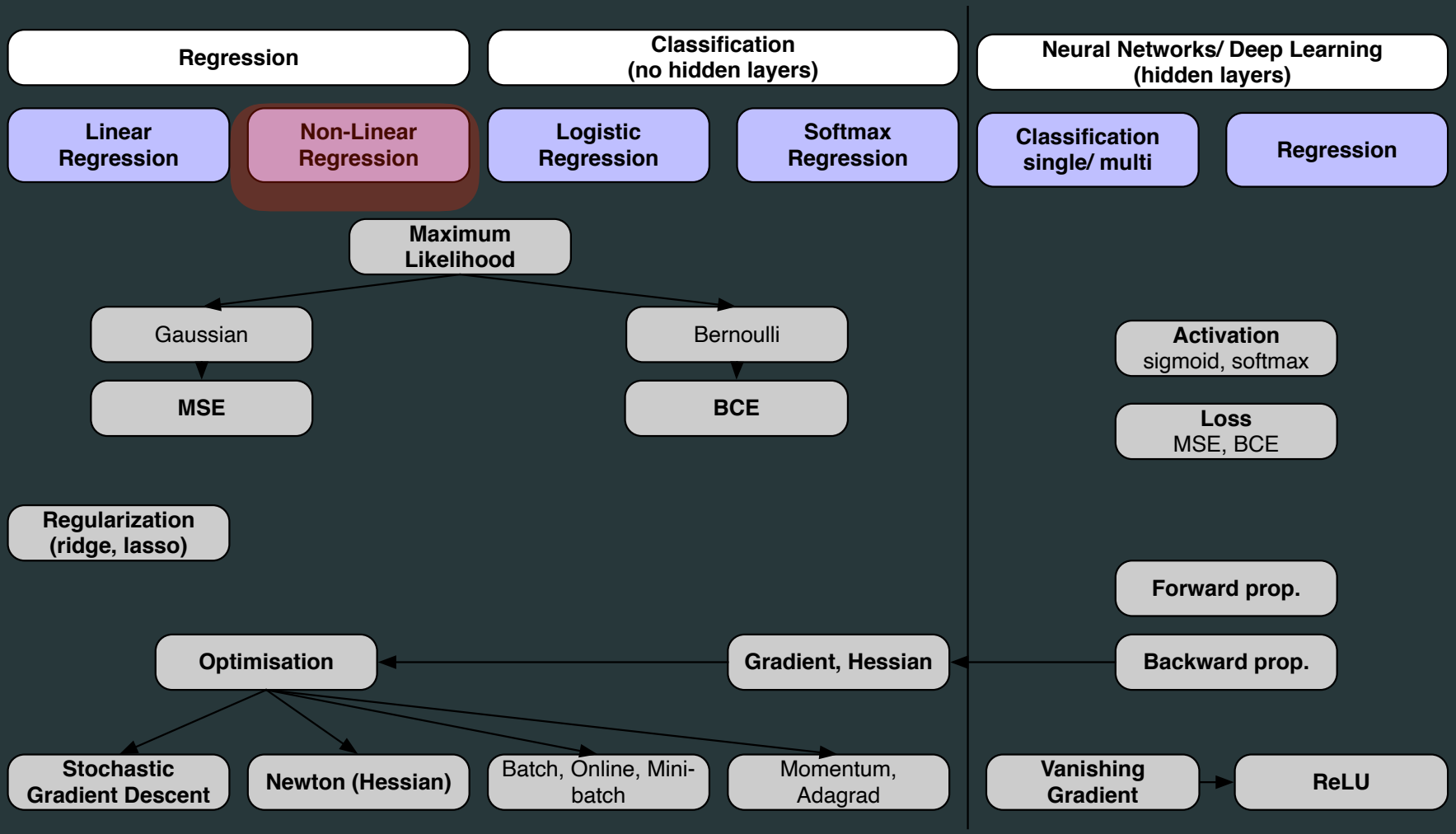
- **Lasso:**

$$\|\theta\|_1 = \sum_d |\theta_d| = \theta_1 + \theta_2$$

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



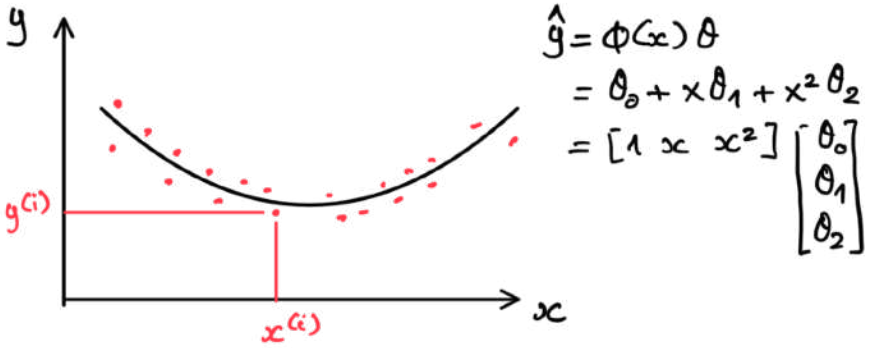
Regression : Non-linear (basis, kernel)



Regression : Non-linear (basis, kernel)

- **Basis functions** $\phi(\,.\,)$

$$\hat{y}(\mathbf{x}) = \phi(\mathbf{x})\,\theta + \epsilon$$



- **Examples:**
  - in 1 dimensions: $\phi(x) = [1, x, x^2]$

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

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

- **Estimation:**

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X} + \delta^2\mathbb{I})^{-1}\mathbf{X}^T\mathbf{y} \quad \rightarrow \quad \hat{\theta} = (\phi^T\phi + \delta^2\mathbb{I})^{-1}\phi^T\mathbf{y}$$

Regression : Non-linear (basis, kernel)

- **RBF (Radial Basis Function) kernel:**

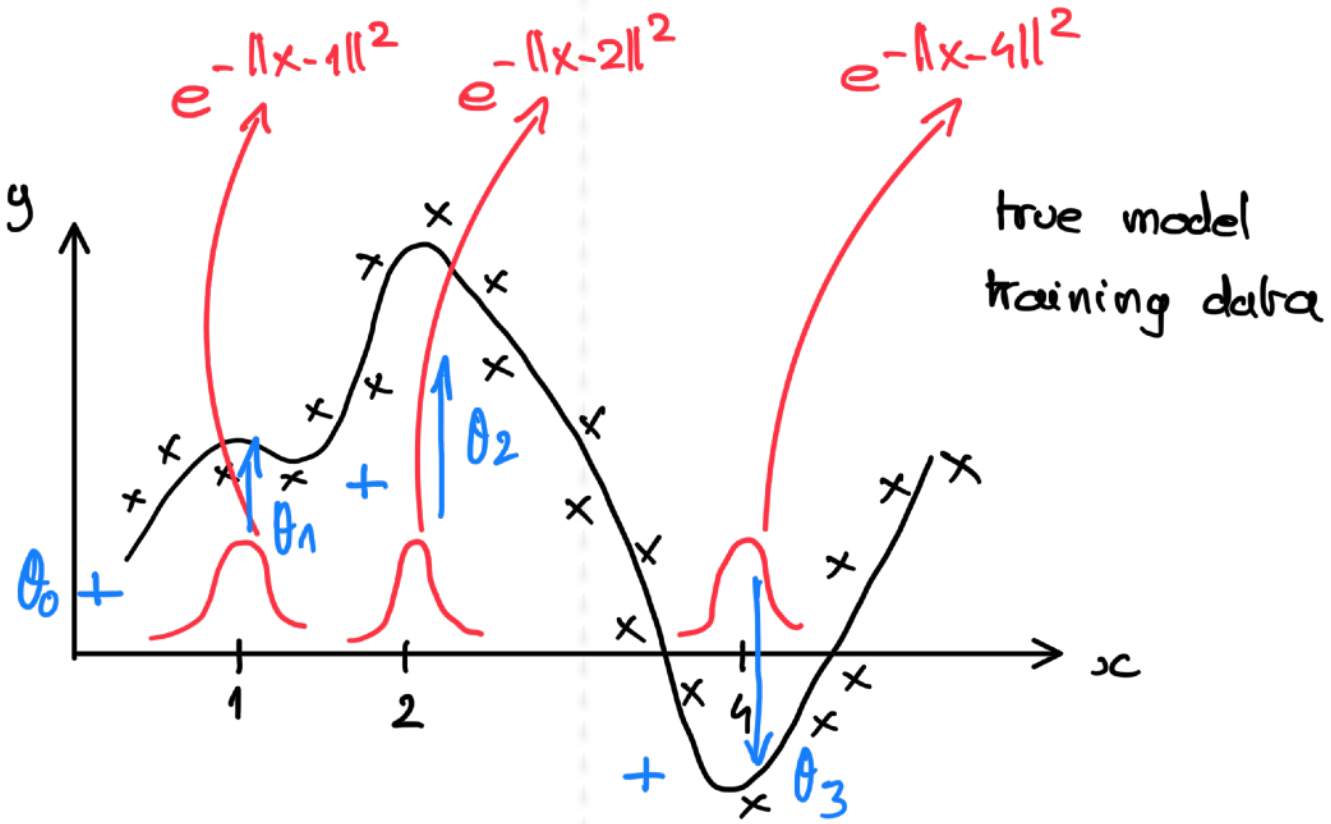
$$\phi(\mathbf{x}) = [k(\mathbf{x}, \mu_1, \lambda), \ldots, k(\mathbf{x}, \mu_d, \lambda)]$$

  - with $k(\mathbf{x}, \mu_d, \lambda) = e^{-\frac{1}{\lambda}||\mathbf{x}-\mu_d||^2}$

$$\hat{y}(\mathbf{x}^{(i)}) = \phi(\mathbf{x}^{(i)}) \ \ \theta$$
$$= 1 \cdot \theta_0 + k(\mathbf{x}^{(i)}, \mu_1, \lambda) \cdot \theta_1 + \ldots + k(\mathbf{x}^{(i)}, \mu_d, \lambda) \cdot \theta_d$$

Regression : Non-linear (basis, kernel)

- **Example:** if $\lambda = 1$ then $\hat{y}(x) = \theta_0 + e^{-\|x-1\|^2}\theta_1 + e^{-\|x-2\|^2}\theta_2 + e^{-\|x-4\|^2}\theta_3$



Regression : Non-linear (basis, kernel)

- We note $\phi(x^{(i)})$ the 4-dimensional (3 bases):
  - For one data:
    $\phi(\mathbf{x}^{(i)}) = [1, k(\mathbf{x}^{(i)}, \mu_1, \lambda), k(\mathbf{x}^{(i)}, \mu_2, \lambda), k(\mathbf{x}, \mu_3, \lambda)]$
  - For m data

$$\hat{\mathbf{y}} = \mathbf{\Phi}\,\theta$$

$$\hat{\theta}_{LS} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{y}$$

$$\hat{\theta}_{Ridge} = (\mathbf{\Phi}^T\mathbf{\Phi} + \delta^2\mathbb{I})^{-1}\mathbf{\Phi}^T\mathbf{y}$$

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

  - $\Rightarrow$It is still a regression !

- **Specific case:**
  - choose the μ_i equal to the training data point $\mathbf{x}^{(i)}$
  - $\Rightarrow$ one kernel on each training data point

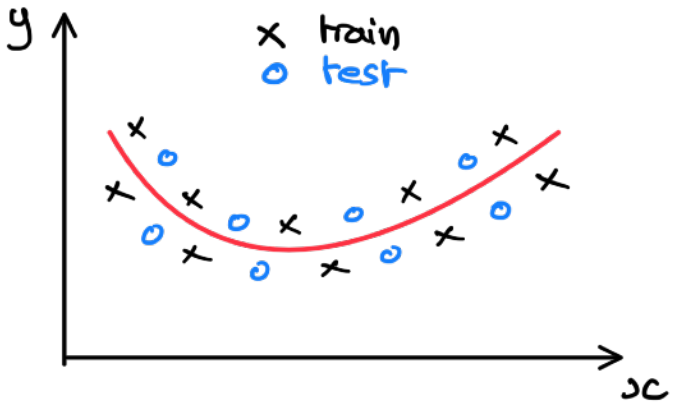
Having the right model (complexity, parameters)
having large number of training

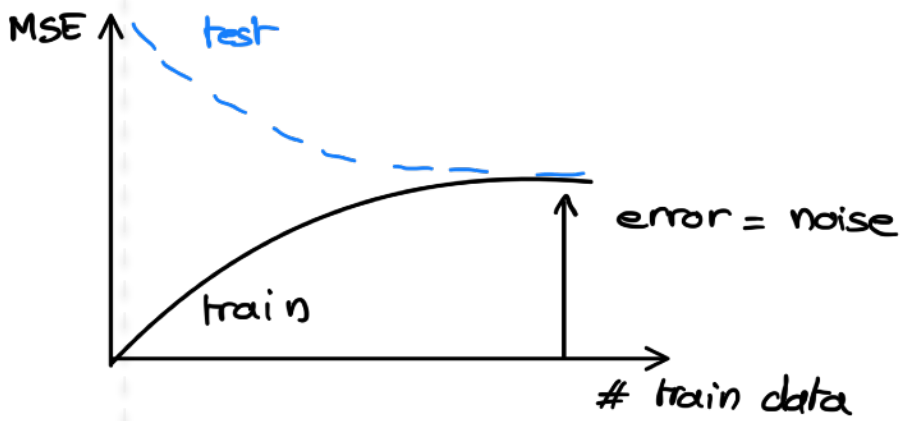
Having the right model (complexity, parameters) having large number of training

- If the model is good → the MSE will converge (with increasing number of data) to the noise level σ^2

True model : $y^{(i)} = \theta_0 + \theta_1 x^{(i)} + \theta_2 x^{2(i)} + \underbrace{\mathcal{N}(0,\sigma^2)}_{\text{noise}}$

Used model : $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$



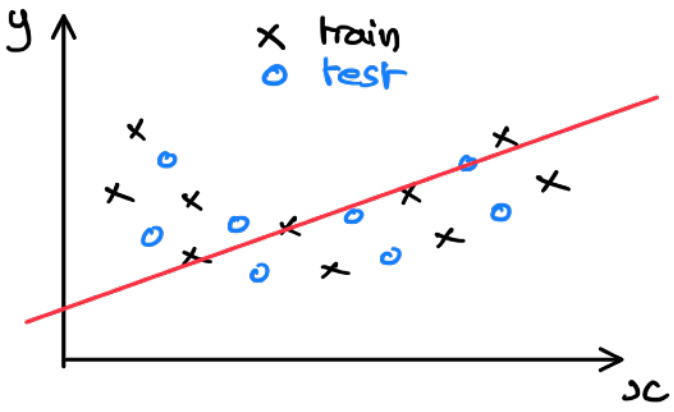


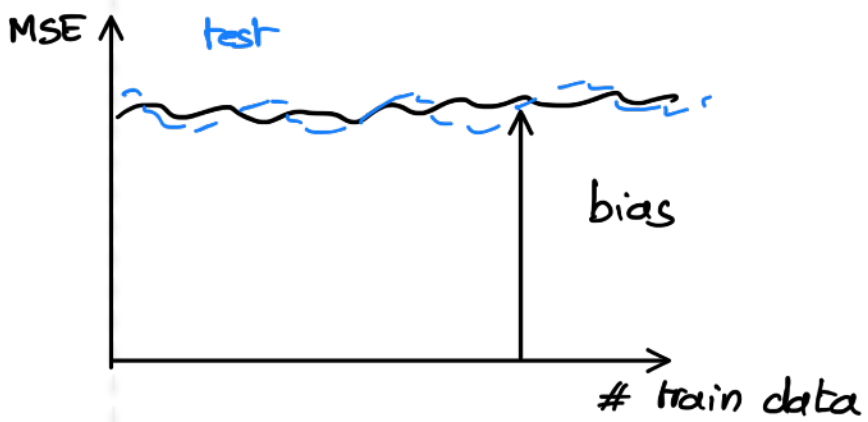
Having the right model (complexity, parameters) having large number of training

- If the model is too weak → the results with be bad whatever the number of data

True model : $y^{(i)} = \theta_0 + \theta_1 x^{(i)} + \theta_2 x^{2(i)} + \underbrace{\mathcal{N}(0, \sigma^2)}_{\text{noise}}$

Used model : $\hat{y} = \theta_0 + \theta_1 x$



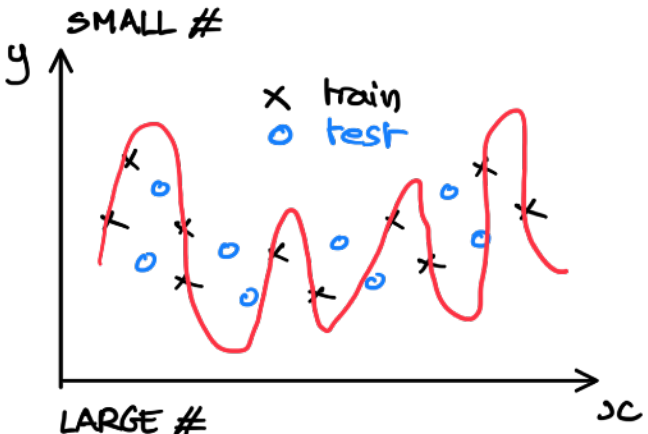


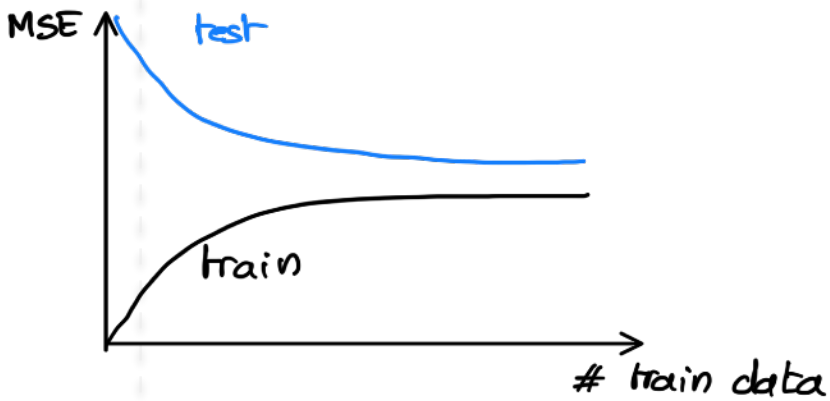
Having the right model (complexity, parameters) having large number of training

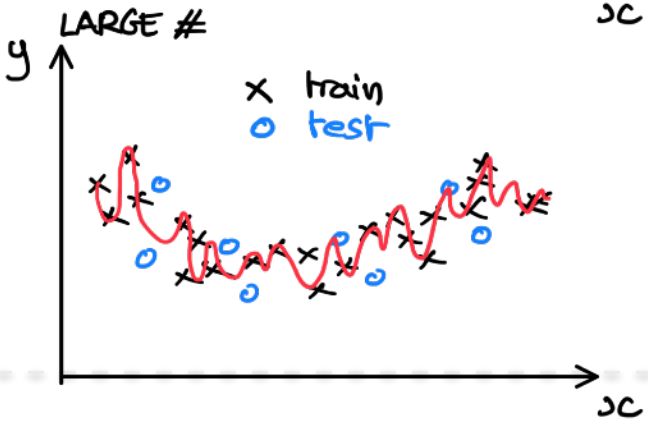
- If the model is too complex → the results will depend on the number of data
  - more data improves the results but only if the model has the right complexity

True model : $y^{(i)} = \theta_0 + \theta_1 x^{(i)} + \theta_2 x^{2(i)} + \mathcal{N}(0, \sigma^2)$

Used model : $\hat{y} = \theta_0 + \theta_1 x + \ldots + \theta_{25} x^{25}$



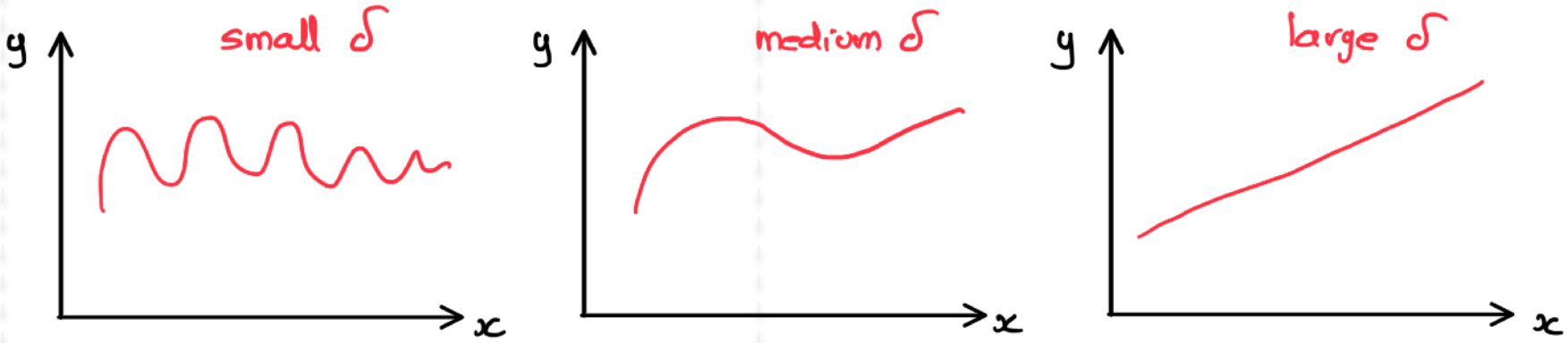




Having the right model (complexity, parameters) having large number of training

- Controlling model complexity with regularisation δ

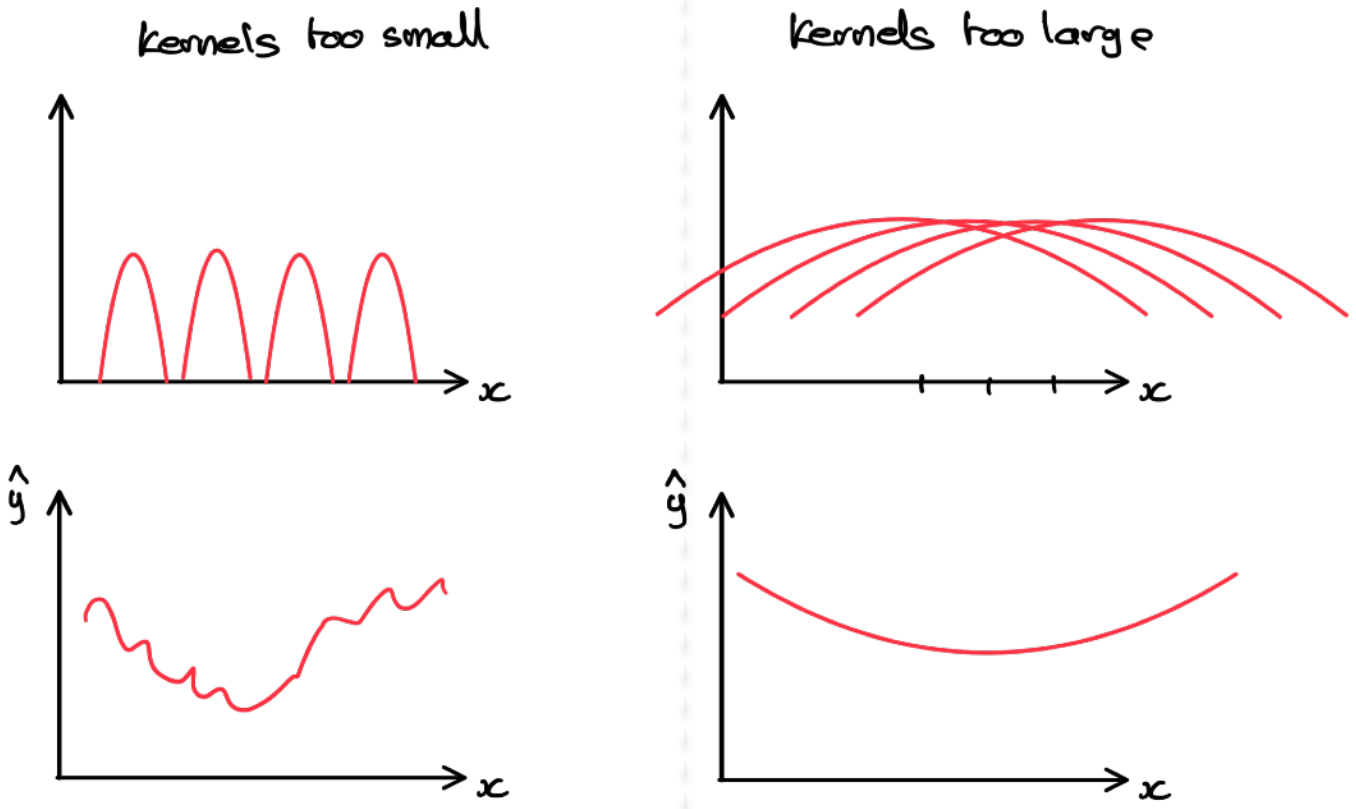
$$J(\theta) = \text{MSE} + \delta \cdot \theta^T \theta$$



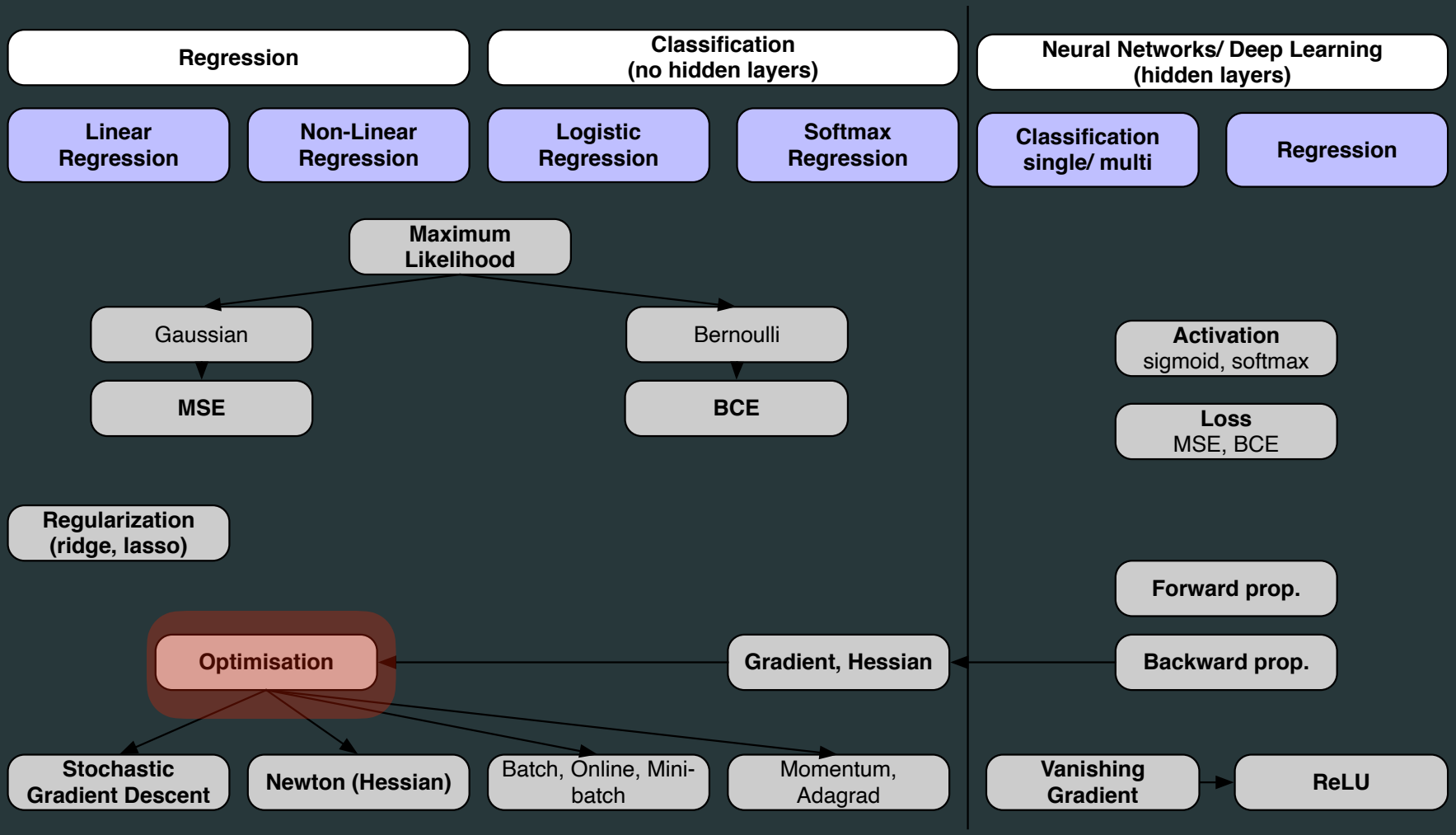
Having the right model (complexity, parameters) having large number of training

- Choice of the RBF kernel size $\lambda \rightarrow$ difficult

$$k(\mathbf{x}, \mu_d, \lambda) = e^{-\frac{1}{\lambda}||\mathbf{x}-\mu_d||^2}$$



Optimisation



Optimisation

- **Gradient**

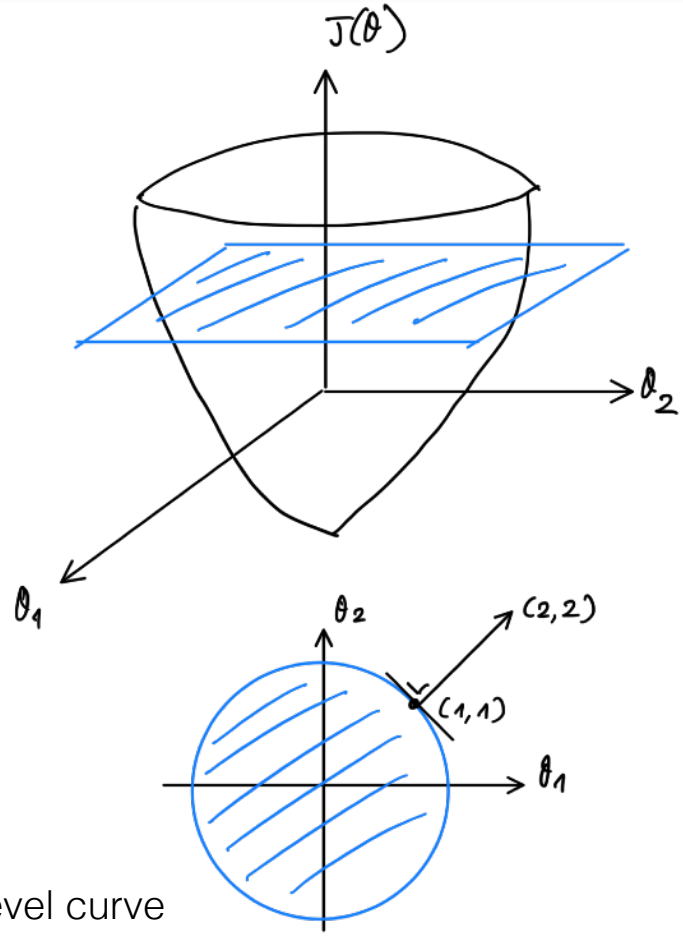
$$\frac{\partial f(\theta_1,\theta_2)}{\partial \theta_1} = \lim_{\Delta\theta_1 \to 0} \frac{f(\theta_1+\Delta\theta_1,\theta_2) - f(\theta_1,\theta_2)}{\Delta\theta_1}$$

- **Example**: consider the function $f(\theta_1,\theta_2) = \theta_1^2 + \theta_2^2$

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

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

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



- **Properties**:
  - the gradient is always orthogonal to the tangent of the level curve
  - the gradient indicates the direction of
    - the maximum change,
    - the steepest ascent

Optimisation

- **Hessian**

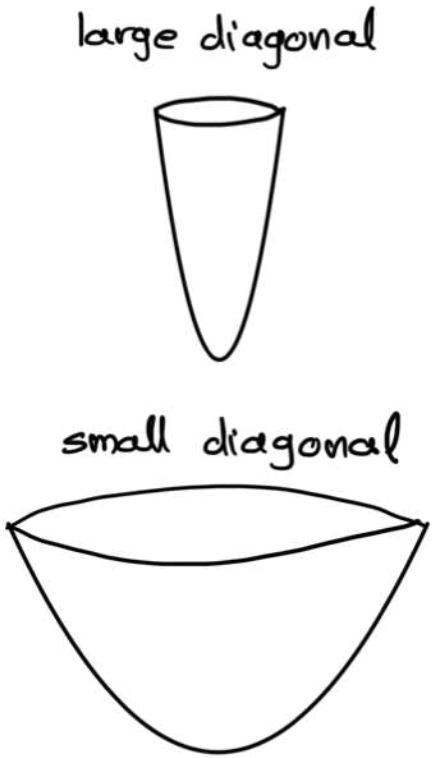
$$\frac{\partial}{\partial\theta_1}\left(\frac{\partial f(\theta)}{\partial\theta_1}\right) = \frac{\partial^2 f}{\partial\theta_1^2} = 2$$

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

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

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

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



Optimisation

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

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

- **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 then ?

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

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

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

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

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

Optimisation
Gradient and Hessian for linear regression

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

- **Scalar form** (applying the chain rule):

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

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

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

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

- **Matrix form**

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

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

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

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

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

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

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

- Notation:
  - θ_k value of θ at iteration k

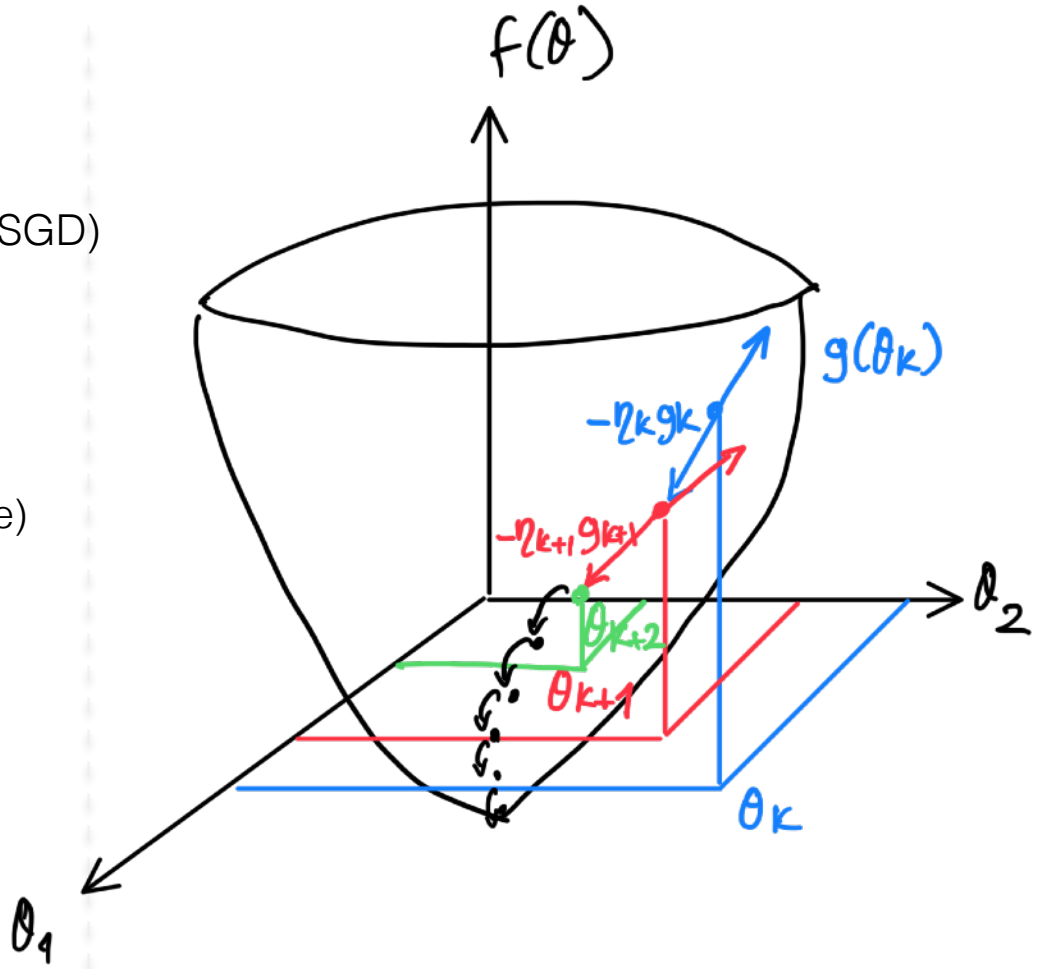
- Steepest/Stochastic Gradient Descent (SGD) algorithm

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

  - where η_k is the learning rate (step size)

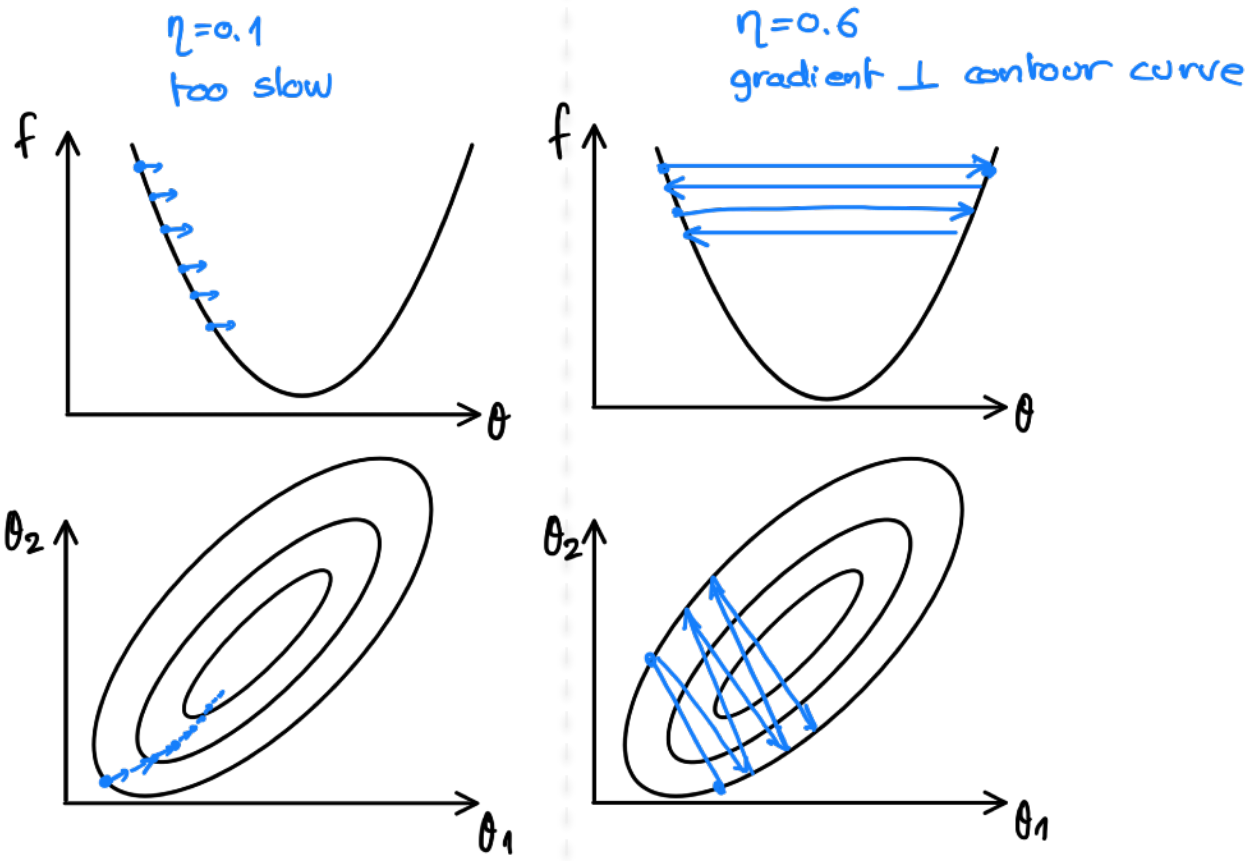
- SGD for linear regression

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



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

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



Optimisation
2) Newton/Hessian algorithm

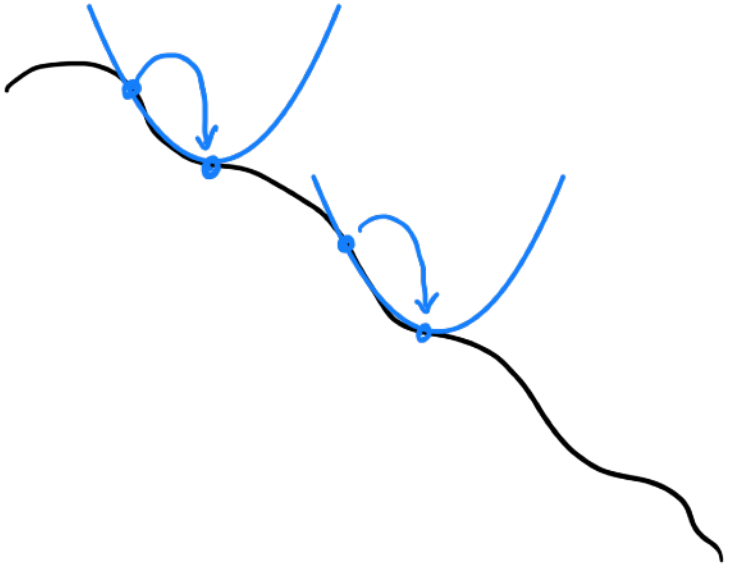
- If we knew the curvature, we could adapt η_k !
- **Taylor series expansion** (approximate $f(\theta)$ around θ_k with a quadratic ball):

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

- Goal: find the minimum of the quadratic ball

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

  - $\Rightarrow$ this is Newton's method



Optimisation
2) Newton/Hessian algorithm

- Newton's for Linear Regression:

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

using (as seen before)

Matrix form

$$\begin{aligned}f(\theta) &= (\mathbf{y} - \mathbf{X}\,\theta)^T(\mathbf{y}\mathbf{X}\,\theta) \\ \nabla_\theta f(\theta) &= \frac{\partial}{\partial\theta}(\mathbf{y}^T\mathbf{y}2\mathbf{y}^T\mathbf{X}\,\theta + \theta^T\mathbf{X}^T\mathbf{X}\,\theta) \\ &= -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\,\theta \\ &= 2\mathbf{X}^T(\mathbf{X}\,\theta - \mathbf{y}) \\ \nabla^2_\theta f(\theta) &= 0 + 2\mathbf{X}^T\mathbf{X} \\ &= 2\,\underset{(d,m)}{\mathbf{X}^T}\,\underset{(m,d)}{\mathbf{X}}\end{aligned}$$

- This is the theoretical solution we found before
  - we get it in just one step !

- However, in practice, Newton/Hessian algorithm is very costly
  - $\Rightarrow$ we need to store $\mathbf{H}$ (if we have 1.000.000 neurons this is a 1.000.000x1.000.000 matrix !)
  - $\Rightarrow$ solution: BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm, L-BFGS (Limited memory) algorithm

Optimisation
Why S stands for "Stochastic" in SGD ?

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

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

- Can be re-written

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

- Therefore

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

Optimisations
Variations

- Rewriting SGD as sum:

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

- Variations of SGD
  - **Batch** (all m data points)

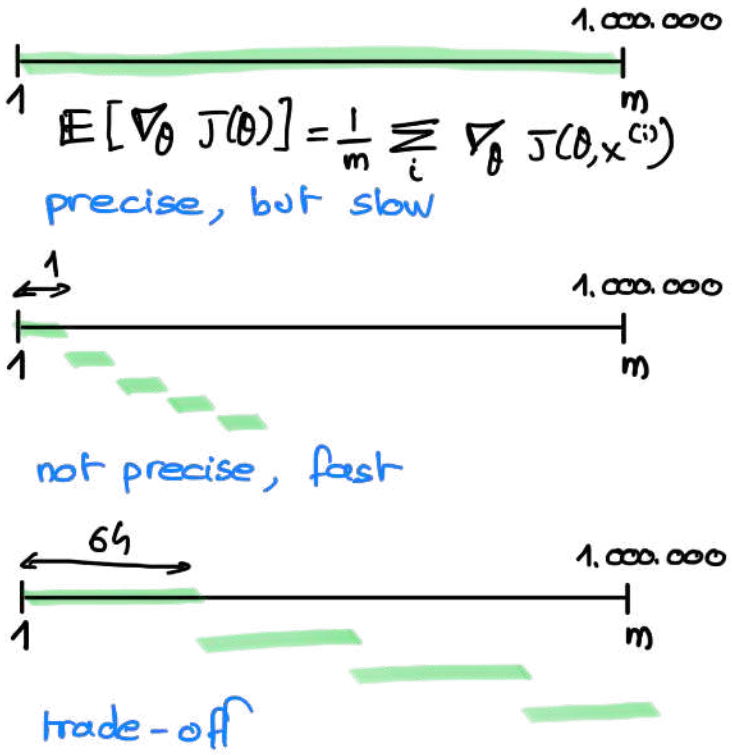
$$\theta_{k+1} = \theta_k + \eta_k \sum_{i=1}^{m} \mathbf{x}^{(i)^T}(\mathbf{y}^{(i)} - \mathbf{x}^{(i)}\theta_k)$$

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

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

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

$$\theta_{k+1} = \theta_k + \eta_k \sum_{j=1}^{64} (\mathbf{x}^{(j)})^T(\mathbf{y}^{(j)} - \mathbf{x}^{(j)}\theta_k)$$

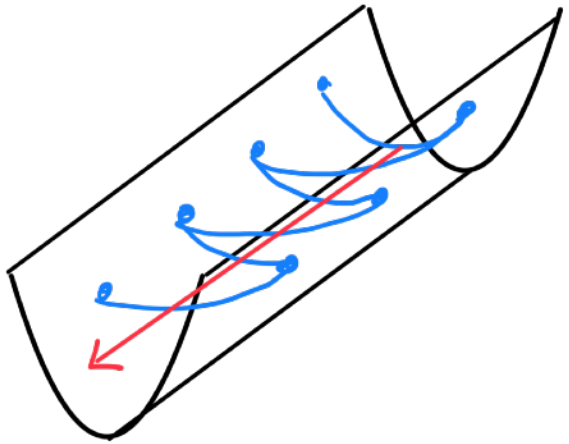


- **Momentum** In this slide we denote by θ^k the value of θ at iteration k
  - Goal: accelerate descent in cumulated direction

Normal SGD:

$$\theta^{k+1} = \theta^k - \eta_k \mathbf{g}_k$$
$$= \theta^k - \eta_k \nabla_\theta f(\theta^k)$$

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



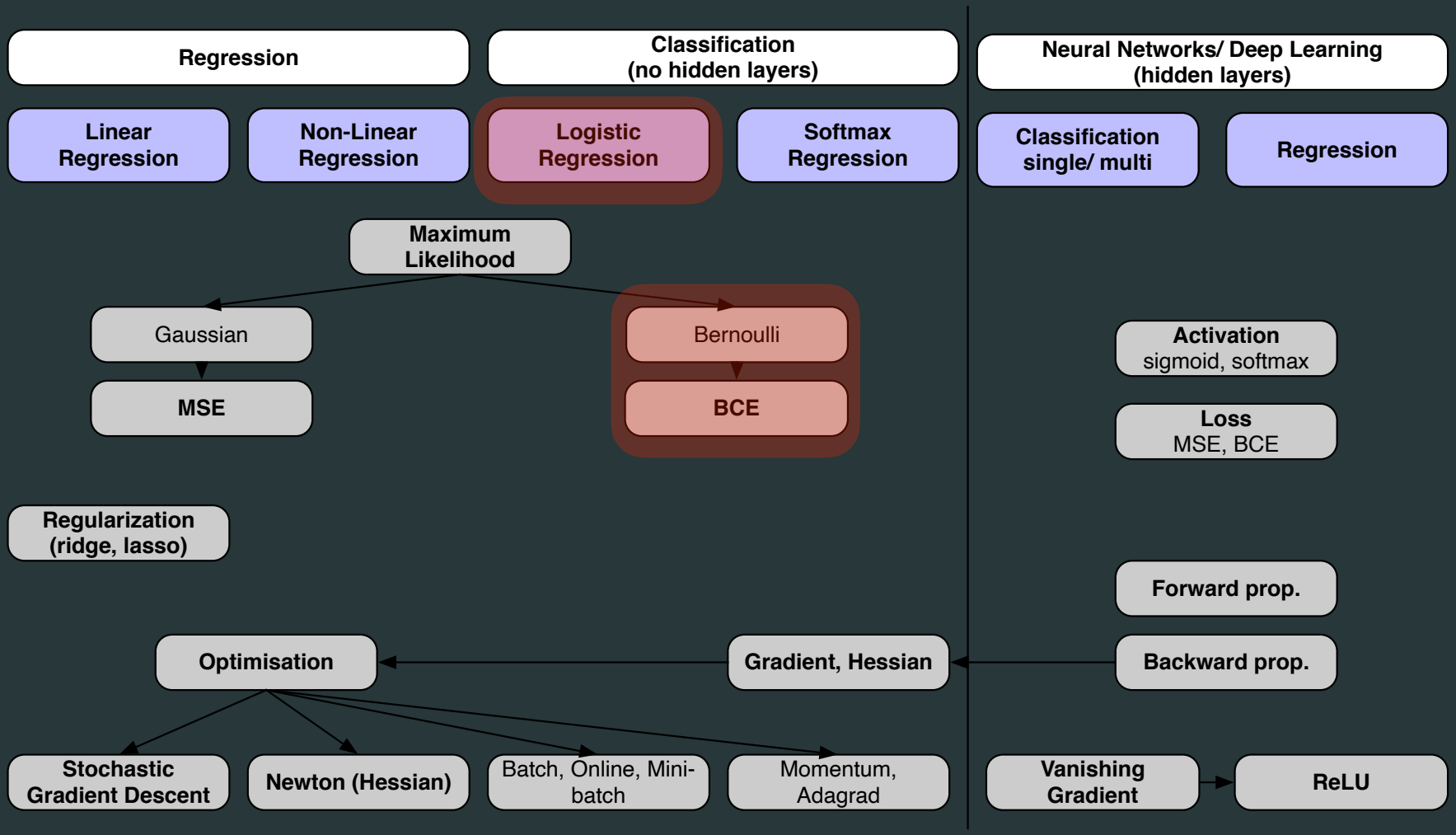
- **AdaGrad**
  - Goal: put more weights on rare features
  - For feature d at step k

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

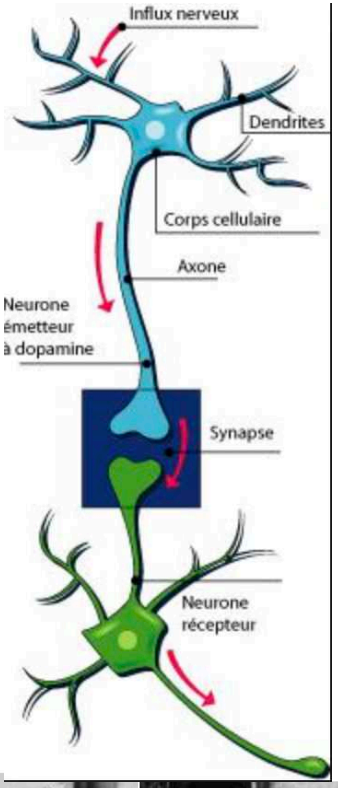
  - $\sum_{\tau=1}^{k} (g_d^\tau)^2$ is the history of how much the feature θ_d has changed in the past
  - if small (has been seen few times in the past) then put more weights

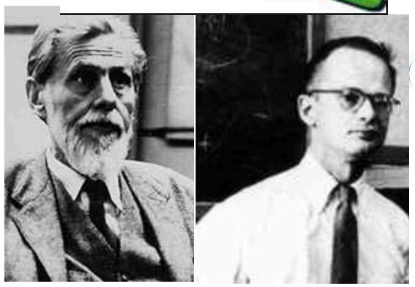
- **ADAM**
  - Momentum + AdaGrad

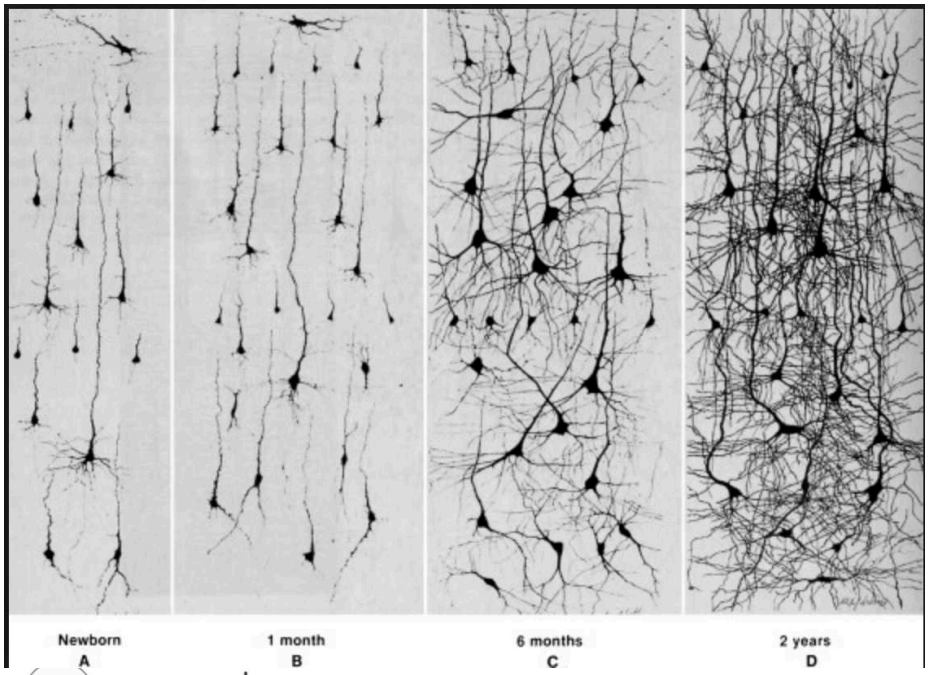
Logistic Regression

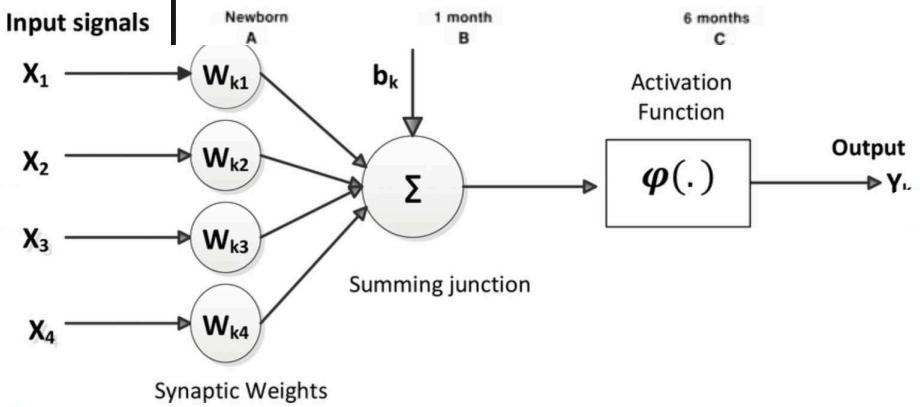


Logistic Regression
[McCulloch - Pitts, 1943] model of a neuron







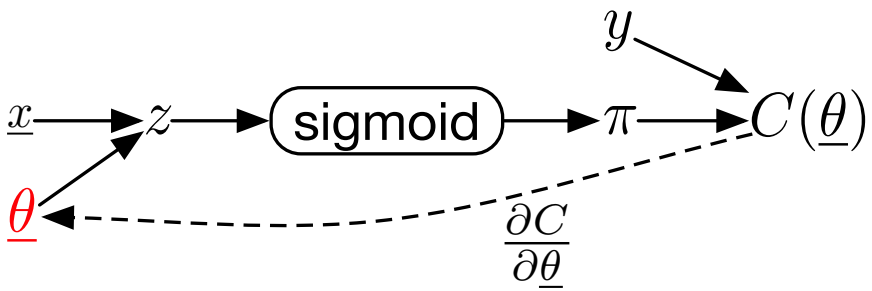


McCulloch-Pitts computational model of a neuron.

Logistic Regression
Activation function

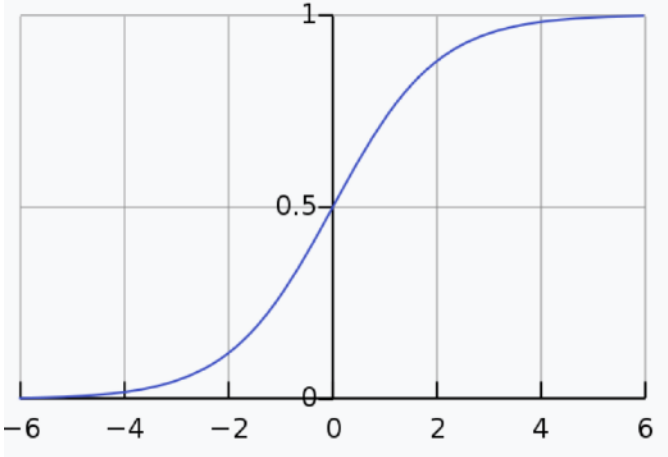
- The logistic regression
  - is a **binary classification** algorithm
  - it outputs the probability of the class 1

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



- **Activation function**
  - **Logistic** (inverse of the logit*)
    - a specific case of sigmoid function (S-shaped)

$$\pi^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \text{ with } z^{(i)} = \mathbf{x}^{(i)}\,\theta$$



  - **Derivative** of the logistic function w.r.t. its input

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

* logit function: $z = \log\left(\frac{p}{1-p}\right)$

Logistic Regression
Linear separating hyper-plane

- The logistic regression defines a linear separating hyper-plane

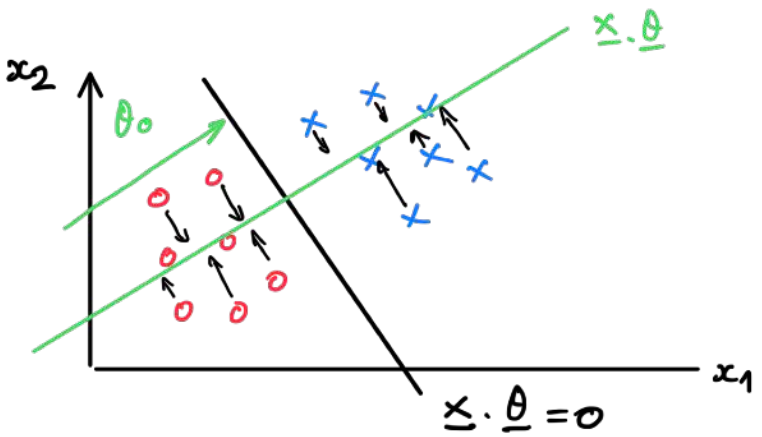
$$P(y^{(i)} = 1 \,|\, \mathbf{x}^{(i)}, \theta) = \frac{1}{2}$$

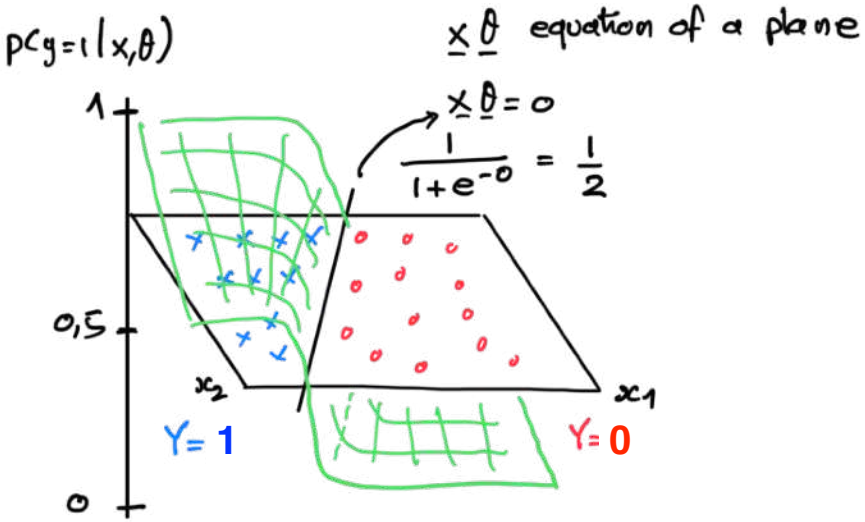
$$\sigma(z) = \frac{1}{2}$$

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

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

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





Logistic Regression
Cost function

- For Linear Regression
  - $\Rightarrow$ uncertainty (random variable) is model by a Gaussian distribution
- For Logistic Regression
  - $\Rightarrow$ uncertainty (random variable) is model by a Bernoulli distribution

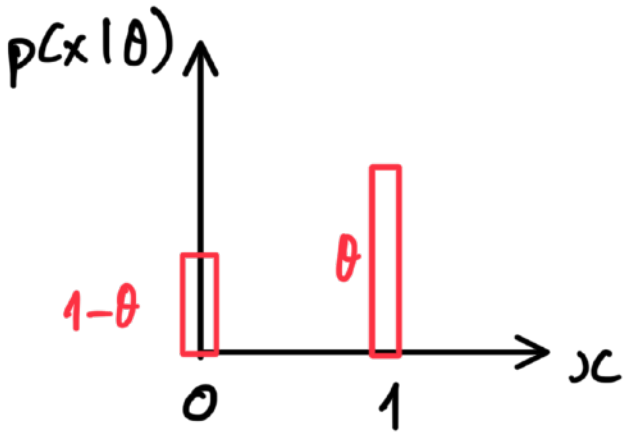
- Learning is about **minimising uncertainty** $\rightarrow$ **entropy**

Maximum Likelihood Estimation (MLE)
Bernoulli distribution

- Bernoulli distribution: X takes value in $\{0,1\}$
  - model of a coin (probability of head or tail)

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

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



Maximum Likelihood Estimation (MLE)
Bernoulli distribution

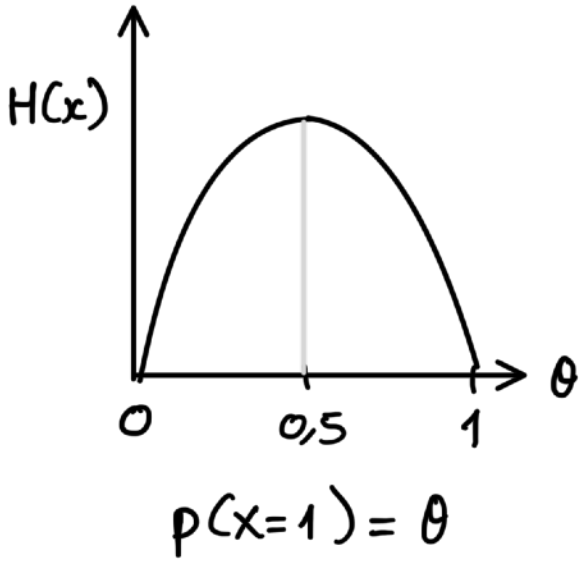
- **Entropy**: is a measure of the uncertainty associated with a random variable:

$$H(X) = -\sum_{x} p(x|\theta) \quad \log(p(x|\theta))$$

- **Entropy of a Bernoulli distribution:**

$$H(X) = -\sum_{x=0}^{1} p^x(1-p)^{(1-x)} \quad \log\left(p^x(1-p)^{(1-x)}\right)$$

$$= -\underbrace{[(1-p)\log(1-p)}_{if\ x=0} + \underbrace{p\log(p)]}_{if\ x=1}$$



- **Entropy of a Gaussian distribution** at D dimensions

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

  - proportional to the determinant of the covariance matrix

Logistic Regression
Cost function

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

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

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

BERNOULLI DISTRIBUTION

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

- **Cost to minimise** = Negative Log-Likelihood (NLL), Binary Cross-Entropy $H(y^{(i)}, \pi^{(i)})$

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

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

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

Logistic Regression
1) Optimisation using Gradient

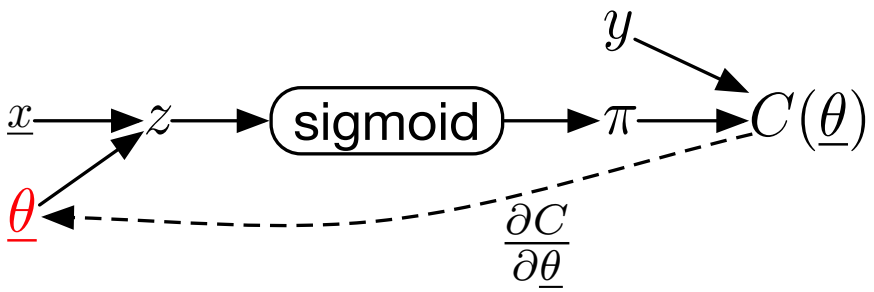
- To minimise the cost $C(\theta)$, we compute the gradient of the $C(\theta)$ w.r.t. to the parameters θ
  - we use the **chain-rule** (* next slide):

$$\mathbf{g} = \frac{\partial C(\theta)}{\partial \theta}$$
$$= \frac{\partial C(\theta)}{\partial \pi}\frac{\partial \pi}{\partial z}\frac{\partial z}{\partial \theta}$$
$$= \sum_{i=1}^{n} \frac{\pi^{(i)} - y^{(i)}}{\pi^{(i)}(1-\pi^{(i)})} \cdot \pi^{(i)}(1-\pi^{(i)}) \cdot \mathbf{x}^{(i)}$$
$$= \sum_{i=1}^{n} (\mathbf{x}^{(i)})^T(\pi^{(i)} - y^{(i)})$$
$$= \mathbf{X}^T(\pi - \mathbf{y})$$

- SGD update at iteration $k+1$

$$\theta_{k+1} = \theta_k - \eta_k \mathbf{g}_k$$
$$= \theta_k - \eta_k \mathbf{X}^T(\pi_k - \mathbf{y})$$

  - where $\pi_k^{(i)} = \sigma(\mathbf{x}^{(i)}\theta_k)$ at iteration k



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

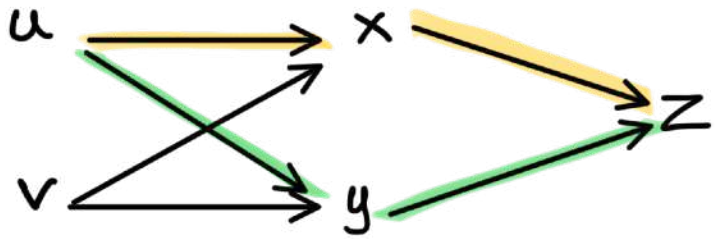
$$\theta_{k+1} = \theta_k - \eta_k \left[2\mathbf{X}^T(\mathbf{X}\,\theta_k - \mathbf{y})\right]$$

Optimisation

- Calculus background: **Chain rule**

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

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



Logistic Regression
2) Optimisation using Hessian (Iteratively Reweighted Least Square

- Newton update:

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

  - with Gradient and Hessian

$$\mathbf{g}_k = \mathbf{X}^T(\pi_k - \mathbf{y})$$
$$\mathbf{H}_k = \mathbf{X}^T\mathbf{S}_k\mathbf{X}$$

- Newton update: at iteration $k+1$

$$\begin{aligned}\theta_{k+1} &= \theta_k - H_k^{-1}g_k \\ &= \theta_k + \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\mathbf{X}^T(\mathbf{y}\pi_k) \\ &= \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\left[(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\theta_k + \mathbf{X}^T(\mathbf{y}-\pi_k)\right] \\ &= \left(\mathbf{X}^T\mathbf{S}_k\mathbf{X}\right)^{-1}\mathbf{X}^T\left[(\mathbf{S}_k\mathbf{X}\theta_k + (\mathbf{y}-\pi_k)\right]\end{aligned}$$

Proof

$$\begin{aligned}H &= \frac{\partial}{\partial\theta}g(\theta)^T \\ &= \sum_{i=1}^{n}\pi^{(i)}(1-\pi^{(i)})\mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T \\ &= \mathbf{X}^T\text{diag}\left(\pi^{(i)}(1-\pi^{(i)})\right)\mathbf{X}\end{aligned}$$

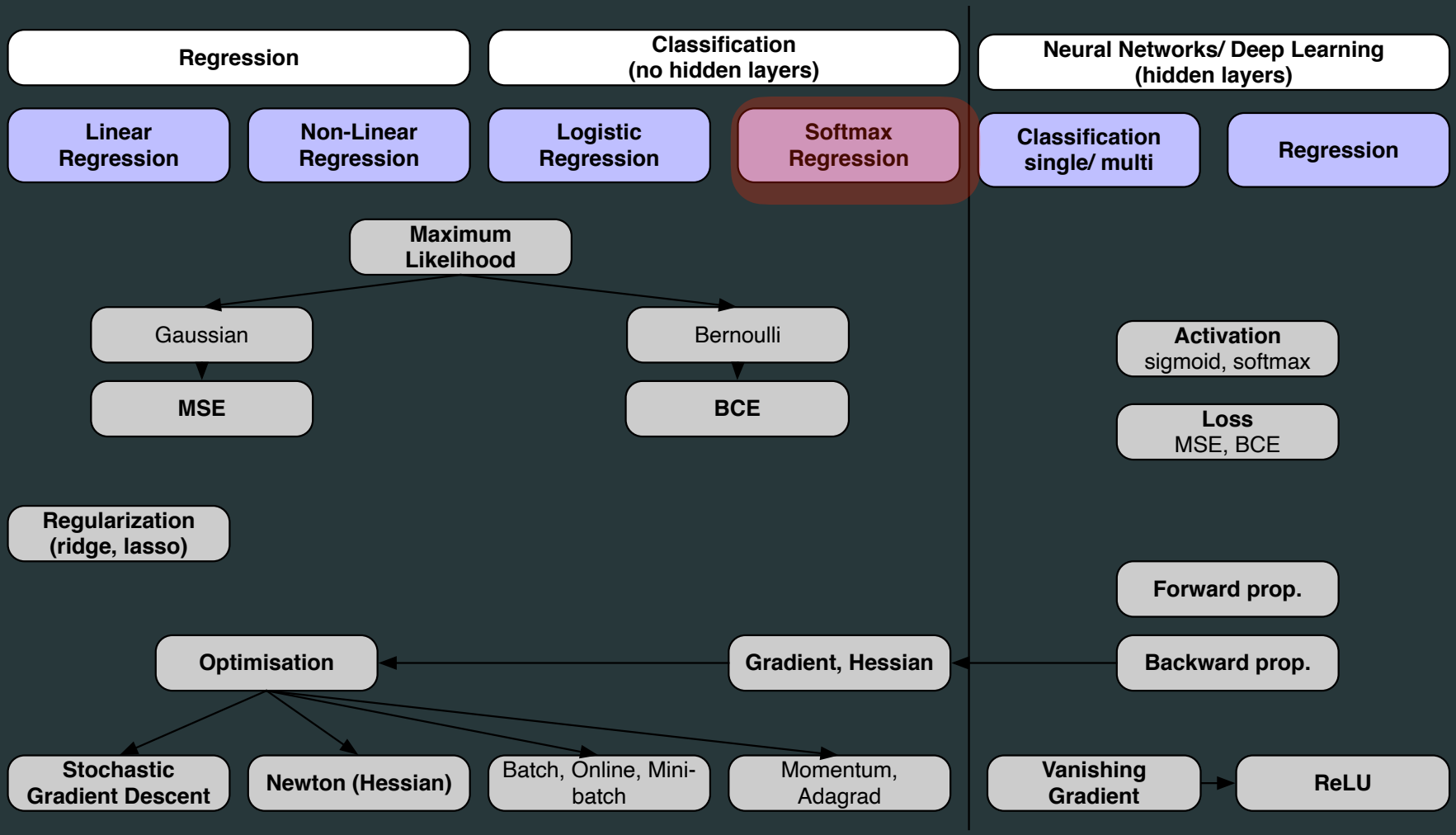
We note $\mathbf{S}_k = \text{diag}\left(\pi_k^{(1)}(1-\pi_k^{(1)})\cdots\pi_k^{(n)}(1-\pi_k^{(n)})\right)$

H is positive definite, hence the NLL is convex and has a unique minimum

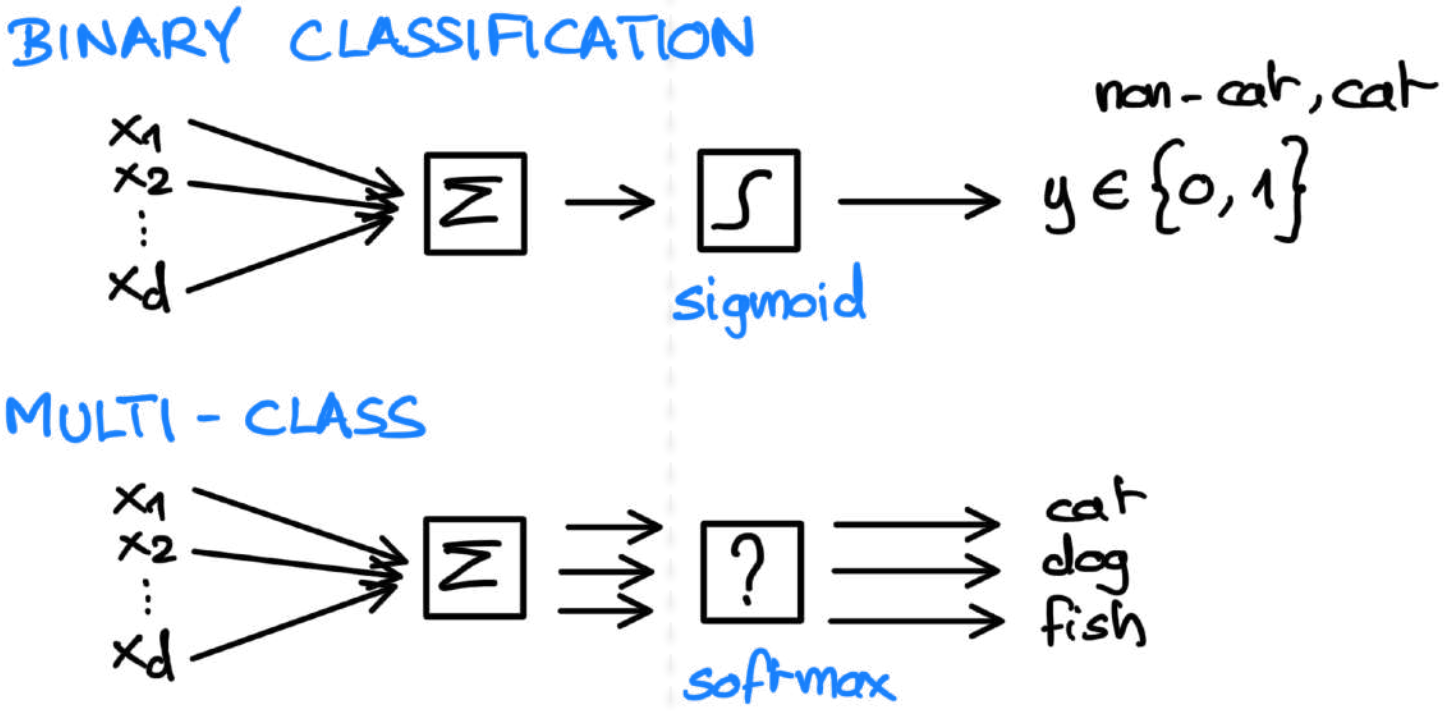
Note : it looks a bit like the least-square solution :

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

Softmax regression



Softmax regression

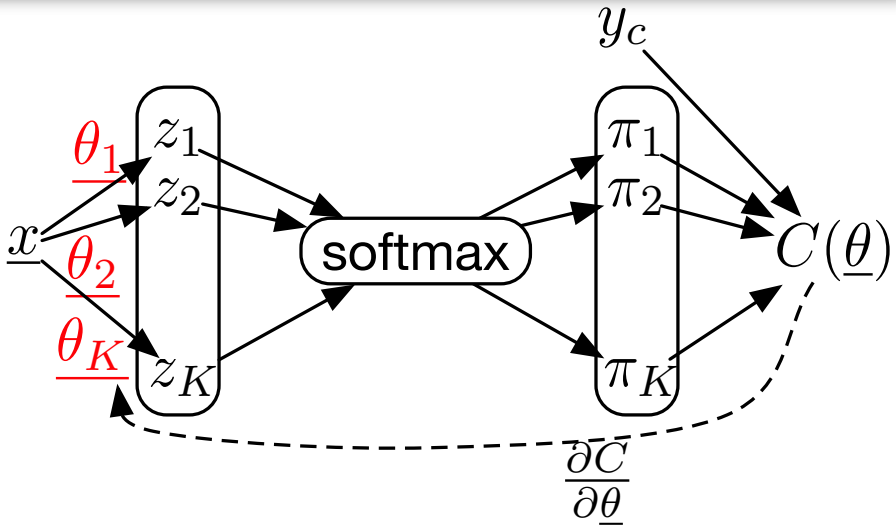


Softmax Regression
Activation function

- The softmax regression
  - is a **multi-class classification** algorithm
  - it outputs the vector of probability for all classes

$$\hat{y}_c^{(i)} = \pi_c^{(i)} = P(y_c^{(i)} = 1 \,|\, \mathbf{x}^{(i)}, \theta)$$
$$= P(y^{(i)} = c \,|\, \mathbf{x}^{(i)}, \theta)$$

$$\hat{y}_c^{(i)} \in [0,1], \quad \sum_c \hat{y}_c^{(i)} = 1$$



- **Activation functions**
  - **Softmax function**

$$\pi_c^{(i)} = \text{softmax}(z_c^{(i)}) = \frac{e^{\mathbf{z}_c^{(i)}}}{\sum_{c'=1}^{K} z_{c'}^{(i)}} \text{ with } \mathbf{z}_c^{(i)} = \mathbf{x}^{(i)}\theta_c$$

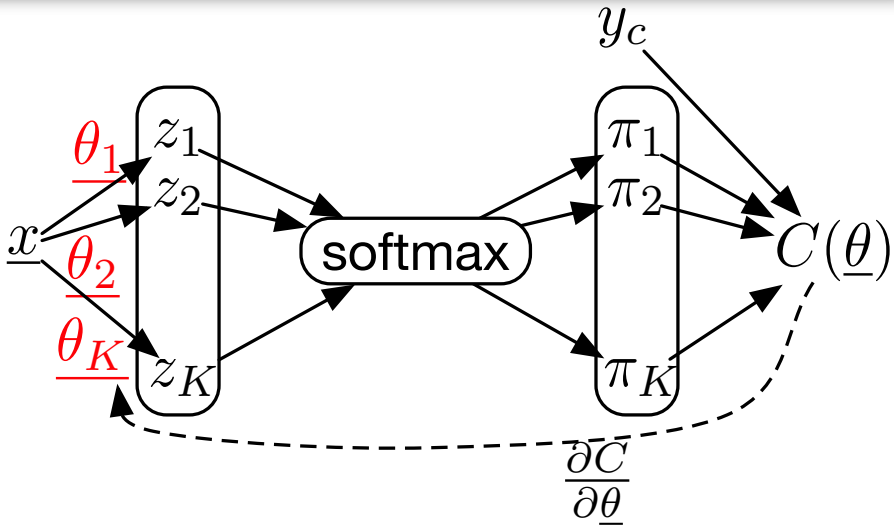
  - It is easy to show that $\pi_1^{(i)} + \cdots + \pi_K^{(i)} = 1$

Softmax Regression
Cost function

- Likelihood function for Softmax
  - Notation: indicator function
    - $\mathbb{I}_c(y^{(i)}) \triangleq \begin{cases} 1 & \text{if } y^{(i)} = c \\ 0 & \text{otherwise} \end{cases}$

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

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



- **Cost to minimise**: Negative Log-Likelihood (NLL) for Softmax, Categorical Cross Entropy

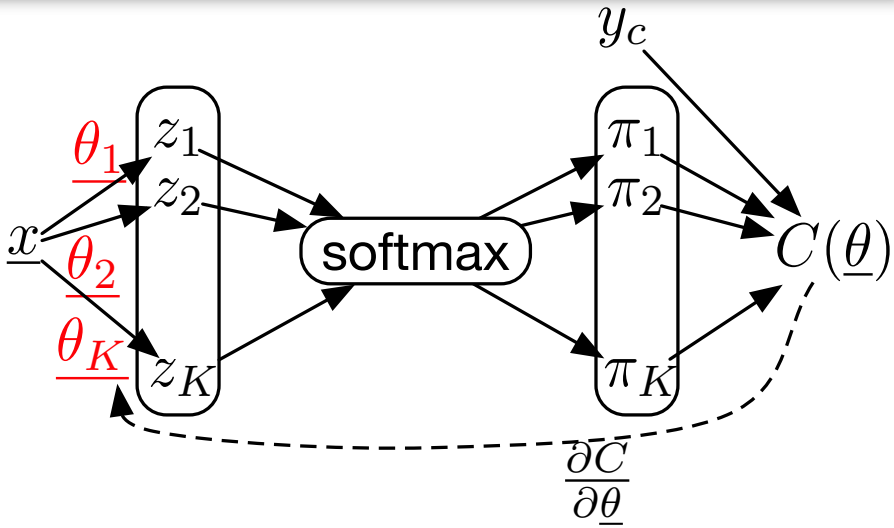
$$C(\theta) = -\log p(\mathbf{y} \mid \mathbf{x}, \theta)$$

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

Softmax Regression
Advanced (for the specific case of 2 classes)

- Likelihood

$$P(y^{(i)}=1\,|\,\mathbf{x}^{(i)},\theta)=(\pi_1^{(i)})^1(\pi_2^{(i)})^0=\pi_1^{(i)}$$
$$=\frac{e^{\mathbf{x}^{(i)}\theta_1}}{e^{\mathbf{x}^{(i)}\theta_1}+e^{\mathbf{x}^{(i)}\theta_2}}$$
$$P(y^{(i)}=2\,|\,\mathbf{x}^{(i)},\theta)=(\pi_1^{(i)})^0(\pi_2^{(i)})^1=\pi_2^{(i)}$$
$$=\frac{e^{\mathbf{x}^{(i)}\theta_2}}{e^{\mathbf{x}^{(i)}\theta_1}+e^{\mathbf{x}^{(i)}\theta_2}}$$



- Negative Log-Likelihood (NLL)

$$C(\theta)=-\log p(\mathbf{y}\,|\,\mathbf{x},\theta)$$
$$=-\sum_{i=1}^{m}\mathbb{I}_1(y^{(i)})\log(\pi_1^{(i)})+\mathbb{I}_2(y^{(i)})\log(\pi_2^{(i)})$$
$$=-\sum_{i=1}^{m}y_1^{(i)}\log(\pi_1^{(i)})+y_2^{(i)}\log(\pi_2^{(i)})$$

Softmax Regression
Advanced (for the specific case of 2 classes)

- Derivative of the cost w.r.t. the parameters

$$\frac{\partial}{\partial \theta_2} C(\theta) = \frac{\partial}{\partial \theta_2} \left(- \sum_i \mathbb{I}_1(y^{(i)}) \log(\pi_1^{(i)}) + \mathbb{I}_2(y^{(i)}) \log(\pi_2^{(i)}) \right)$$

$$= - \sum_i \mathbb{I}_1(y^{(i)}) \frac{\partial}{\partial \theta_2} \log(\pi_1^{(i)}) + \mathbb{I}_2(y^{(i)}) \frac{\partial}{\partial \theta_2} \log(\pi_2^{(i)})$$

$$\frac{\partial}{\partial \theta_2} \log(\pi_1^{(i)}) = \frac{\partial}{\partial \theta_2} \log \frac{e^{\mathbf{x}^{(i)}\theta_1}}{e^{\mathbf{x}^{(i)}\theta_1} + e^{\mathbf{x}^{(i)}\theta_2}}$$

$$= \frac{\partial}{\partial \theta_2} \left(\mathbf{x}^{(i)}\theta_1 - \log(e^{\mathbf{x}^{(i)}\theta_1} + e^{\mathbf{x}^{(i)}\theta_2}) \right)$$

$$= 0 - \frac{\mathbf{x}^{(i)} e^{\mathbf{x}^{(i)}\theta_2}}{e^{\mathbf{x}^{(i)}\theta_1} + e^{\mathbf{x}^{(i)}\theta_2}}$$

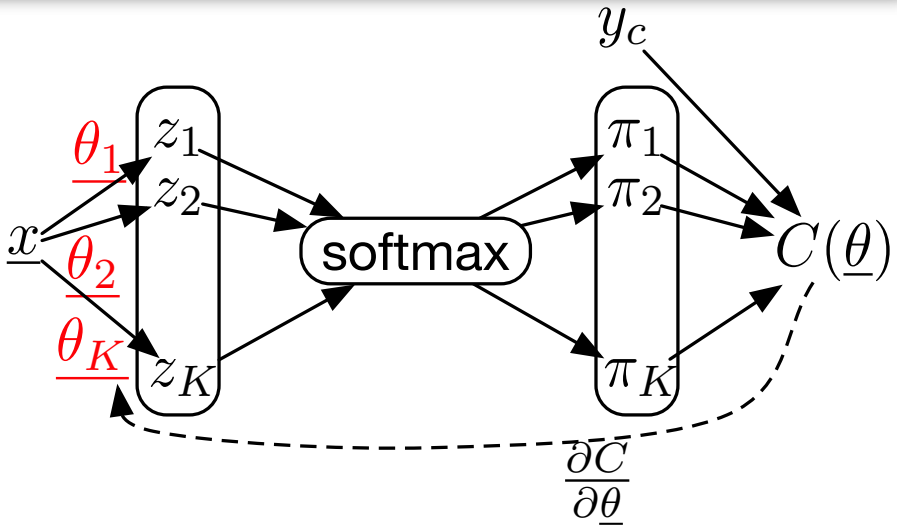
$$= - \mathbf{x}^{(i)} \pi_2^{(i)}$$

$$\frac{\partial}{\partial \theta_2} \log(\pi_2^{(i)}) = \cdots = \mathbf{x}^{(i)}(1 - \pi_2^{(i)})$$

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

Softmax Regression
Advanced (for the specific case of 2 classes)

- **It is equivalent to the logistic regression !!!**



  - Logistic regression $y^{(i)} \in \{0,1\}$ using
    - $\mathbb{I}_1(y^{(i)}) = (1 - y^{(i)})$
    - $\mathbb{I}_2(y^{(i)}) = y^{(i)}$

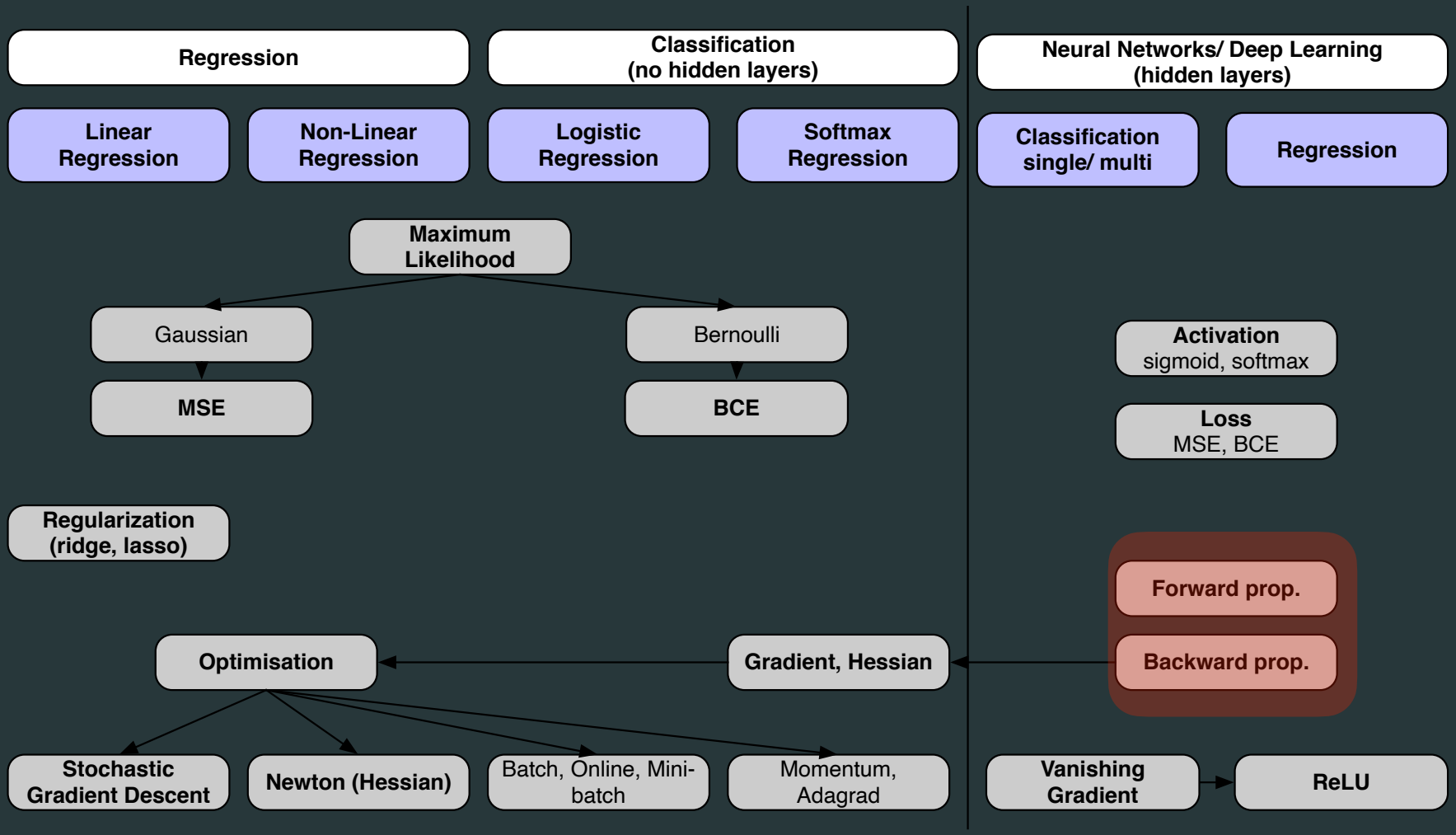
$$\frac{\partial C(\theta)}{\partial \theta_2} = -\sum_i \mathbb{I}_1(y^{(i)})(-\mathbf{x}^{(i)}\pi_2^{(i)}) + \mathbb{I}_2(y^{(i)})\mathbf{x}^{(i)}(1-\pi_2^{(i)})$$

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

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

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

Back-propagation



Back-propagation

- We consider a softmax regression

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

- We consider it as a succession of layers $\mathbf{z}^{[l]}$:

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

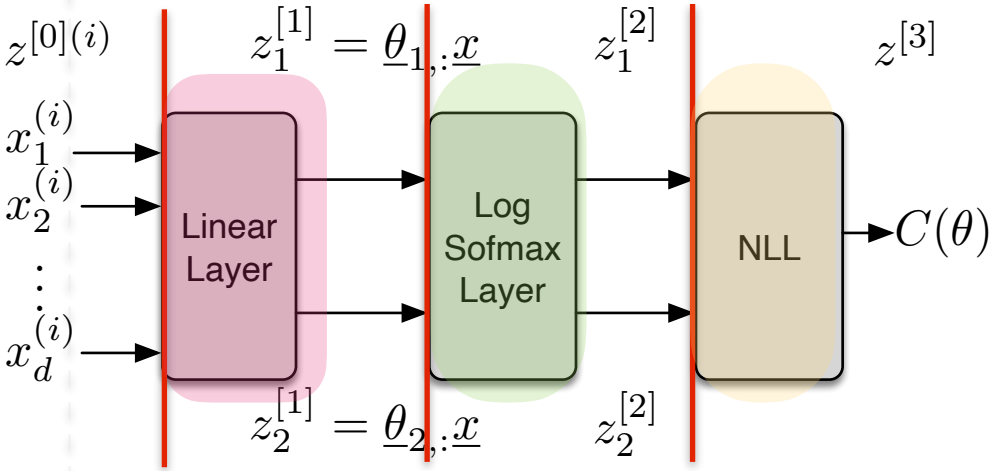
$$z_1^{[1]} = \theta_{1,:}\mathbf{z}^{[0]}$$

$$z_2^{[1]} = \theta_{2,:}\mathbf{z}^{[0]}$$

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

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

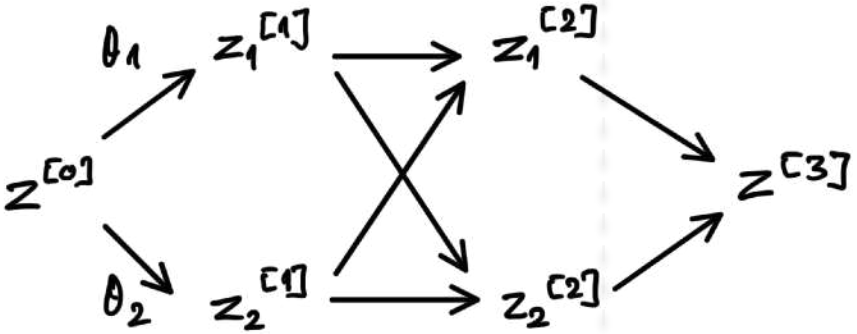
$$z^{[3]} = -\sum_i \mathbb{I}_1(y^{(i)}) z_1^{[2]} + \mathbb{I}_2(y^{(i)}) z_2^{[2]}$$



Back-propagation

- We consider it as a succession of layers $\mathbf{z}^{[l]}$
  - Composite functions

$$C(\theta) = z^{[3]}\left\{ z_1^{[2]}\left[z_1^{[1]}(z^{[0]},\theta_1), z_2^{[1]}(z^{[0]},\theta_2)\right], z_2^{[2]}\left[z_1^{[1]}(z^{[0]},\theta_1), z_2^{[1]}(z^{[0]},\theta_2)\right]\right\}$$



  - Computing derivatives using the **chain rule**

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

    - This is **Inefficient**: we compute several times the same things !!!

Deep Learning
Two ways to implement gradient computation

- Suppose we have the following composition of functions

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

- **Solution 1**

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

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

- **Solution 2**

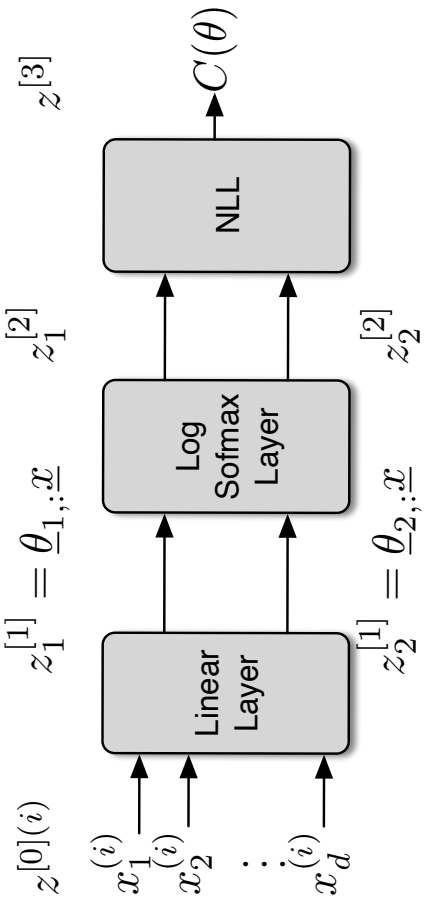
  - start from the end and move back to the start

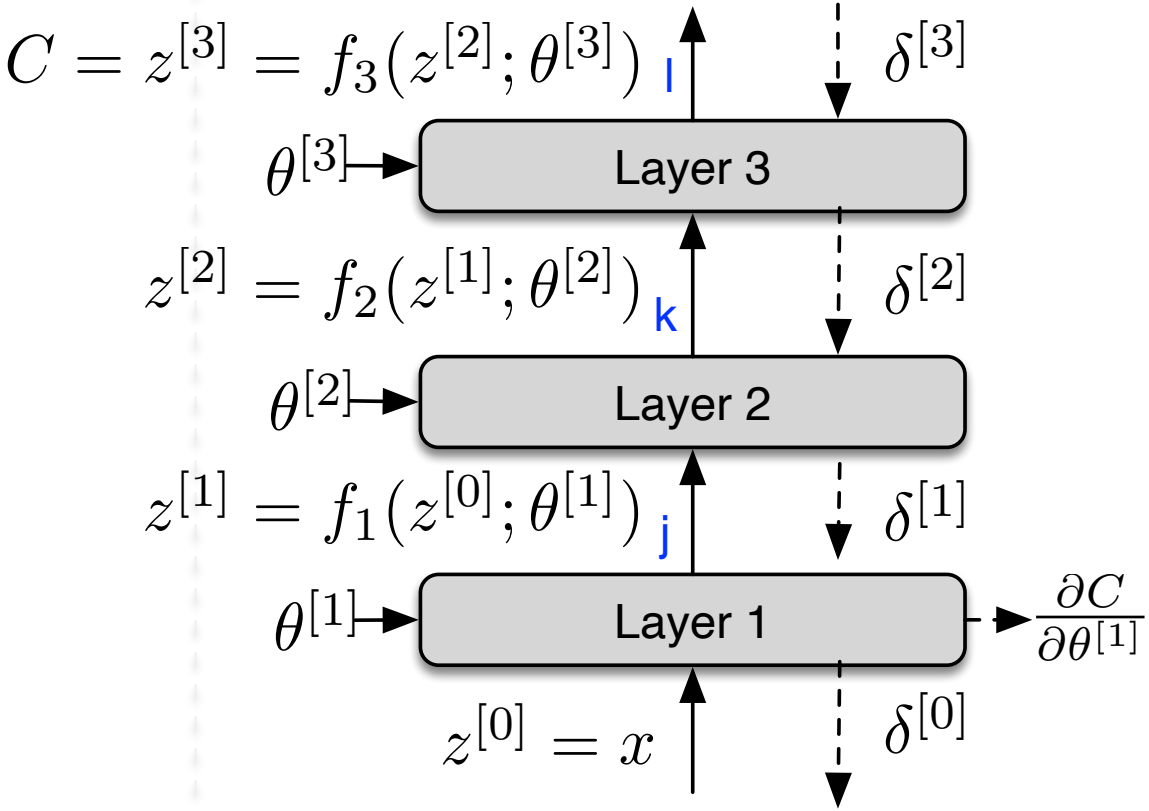
Jacobian

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

  - Compute first (-) then second (-) then third (-)
  - It uses less memory (the intermediate results are always vectors: vector x matrix = vector)
  - ⇒ this is the **back-propagation algorithm**

Back-propagation
Layer specification





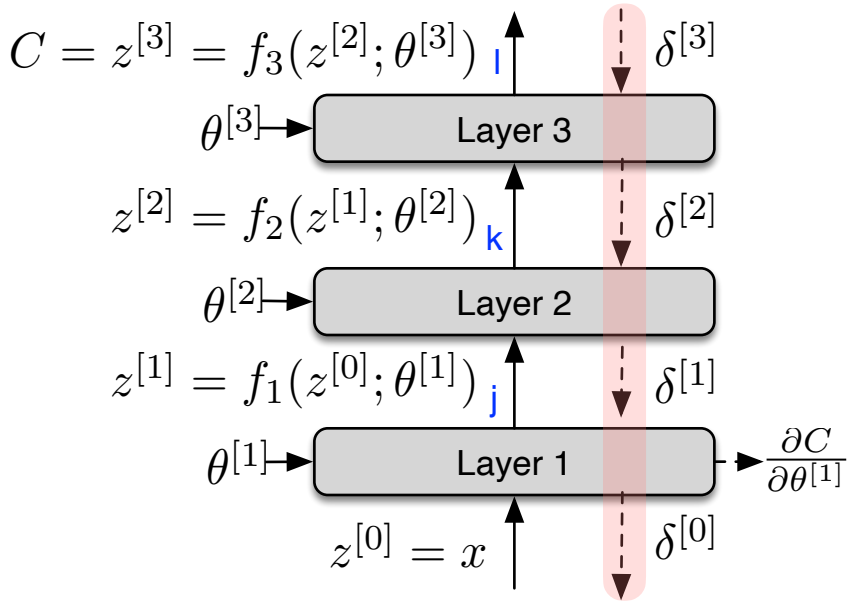
Back-propagation
Derivative via layer-specification

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

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

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

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



- 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 $z^{[0]} = \mathbf{x}^{(i)} \rightarrow z^{[1]}(\mathbf{x}^{(i)}) \rightarrow z^{[2]}(\mathbf{x}^{(i)}) \rightarrow z^{[3]}(\mathbf{x}^{(i)}) = C$
- Backward $\delta^{[0]} \leftarrow \delta^{[1]} \leftarrow \delta^{[2]} \leftarrow \delta^{[3]} = 1$

Back-propagation
Layer specification

- We only have to design a **module** with
  - a **Forward** method:

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

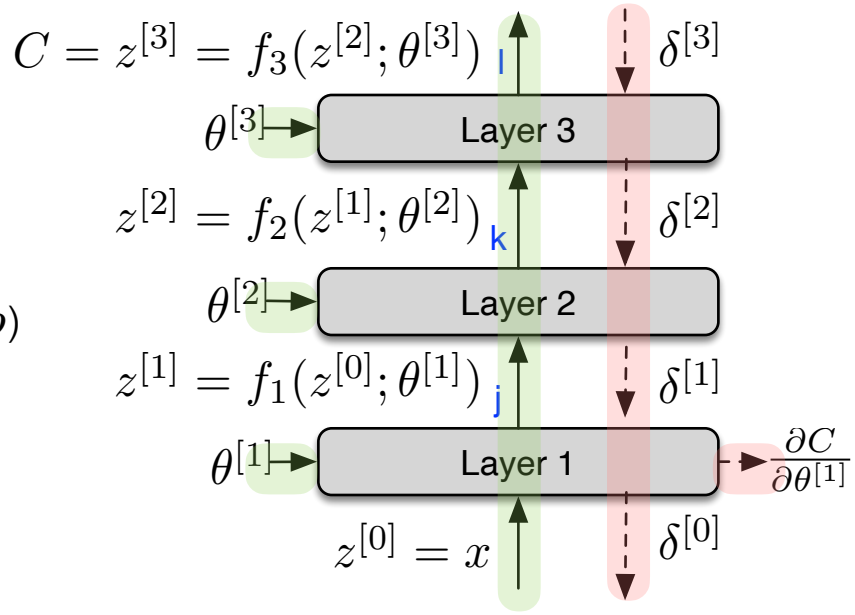
  - a **Backward** method (for inputs i and outputs o)
    - Input

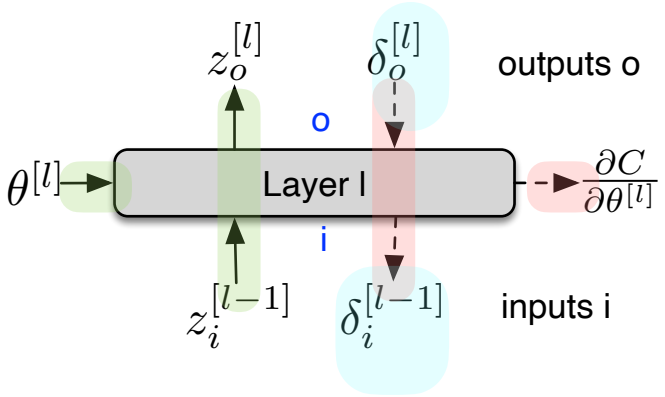
$$\delta_o^{[l]} \triangleq \frac{\partial C}{\partial z_o^{[l]}}$$

    - Computation

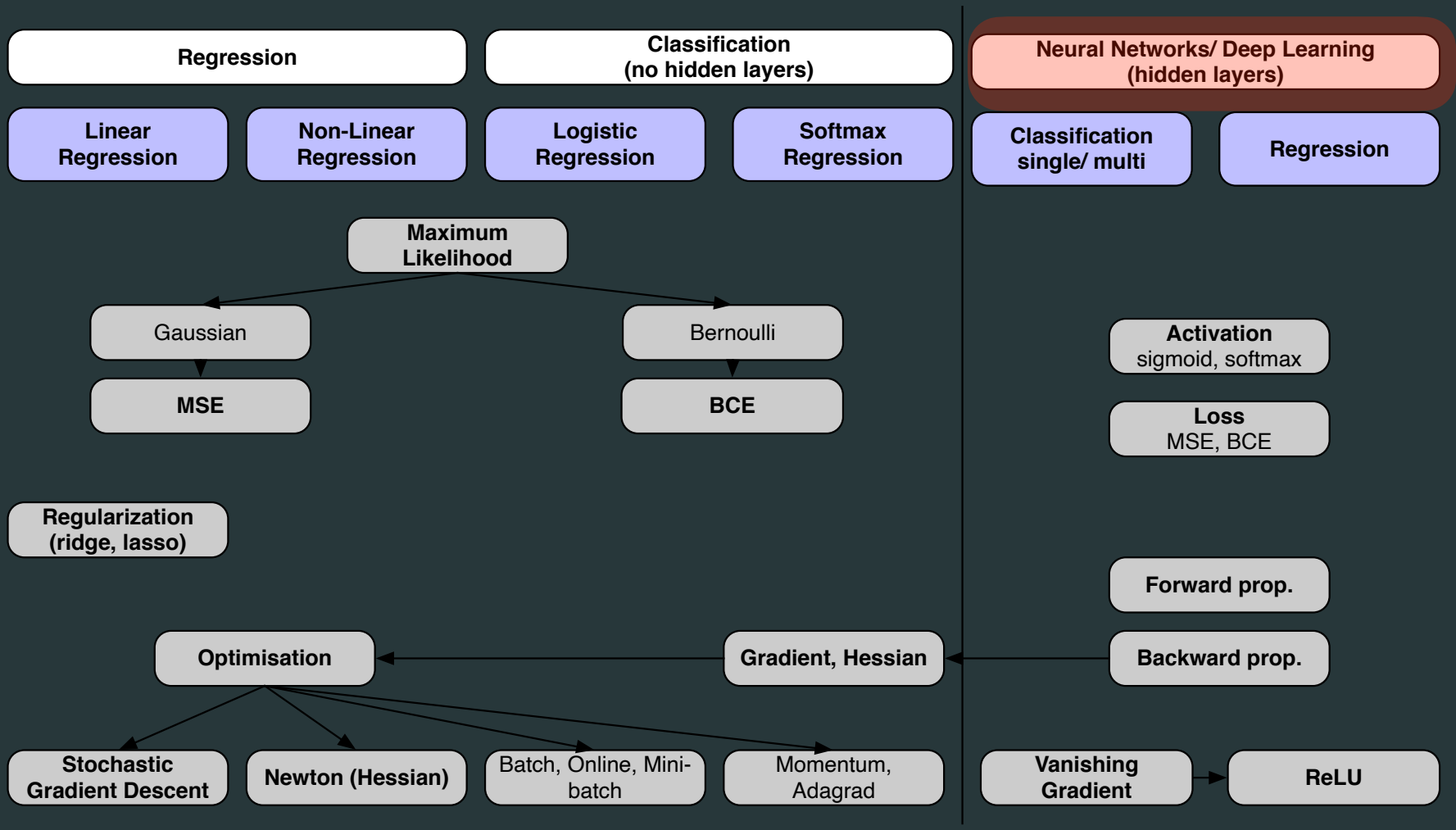
$$\delta_i^{[l-1]} = \sum_o \frac{\partial C}{\partial z_o^{[l]}} \frac{\partial z_o^{[l]}}{\partial z_i^{[l-1]}} = \sum_o \delta_o^{[l]} \frac{\partial z_o^{[l]}}{\partial z_i^{[l-1]}}$$

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



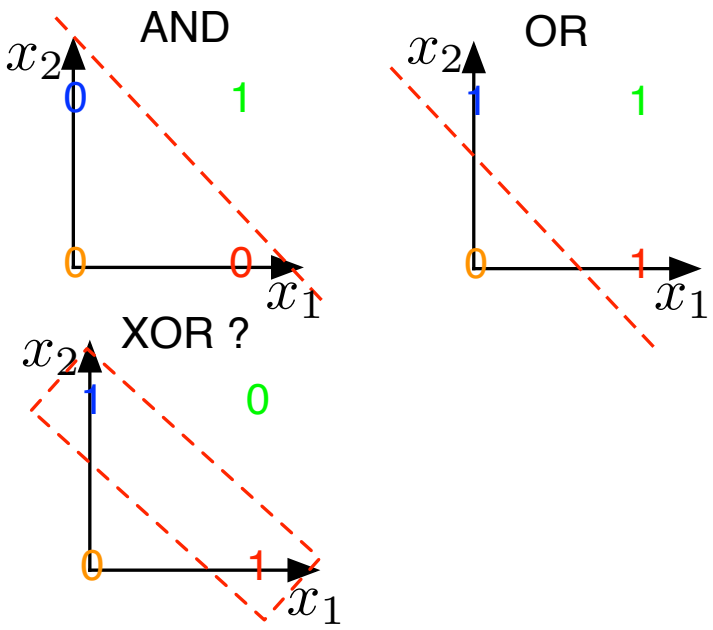


More than one layer

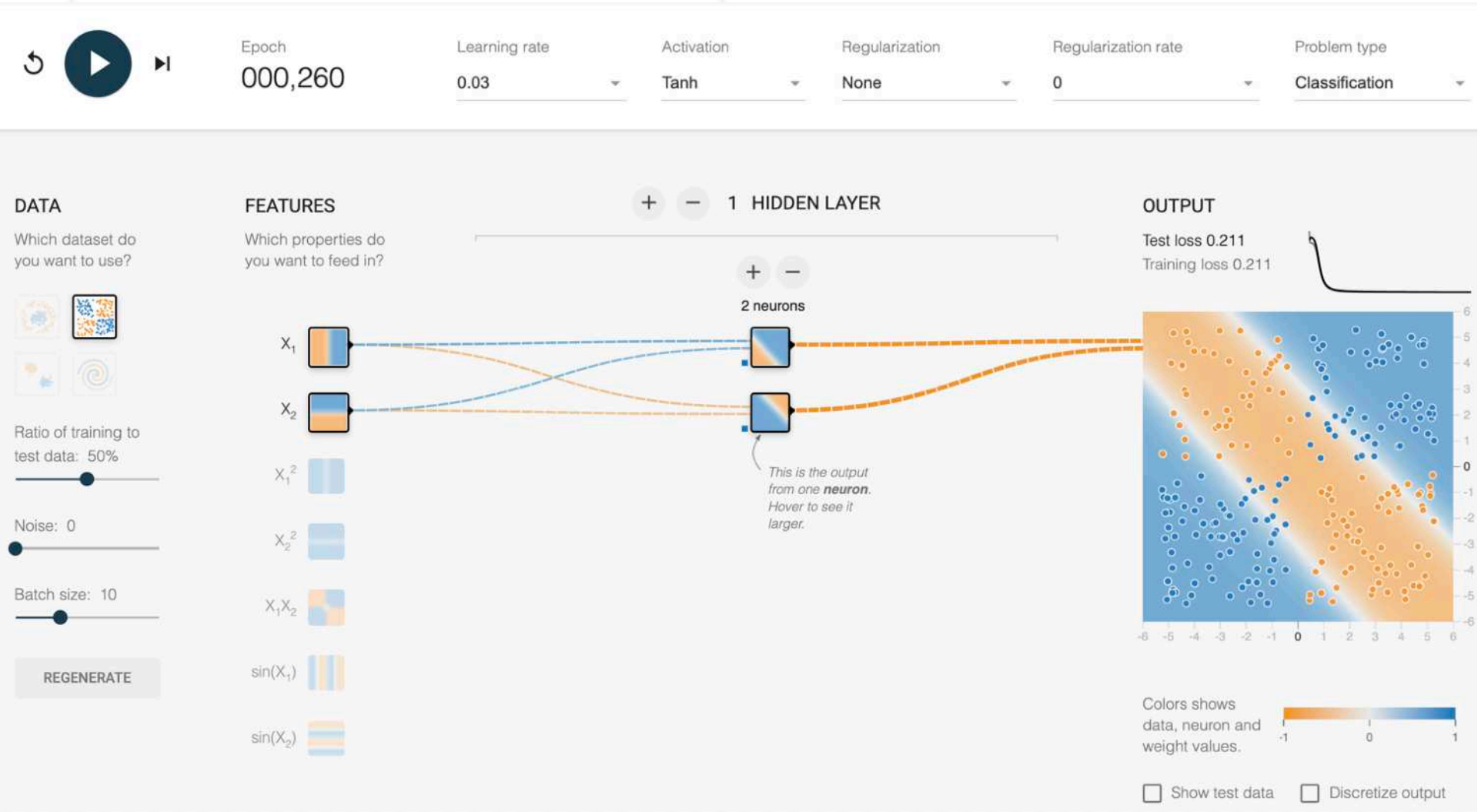


Limitation of linear classifiers

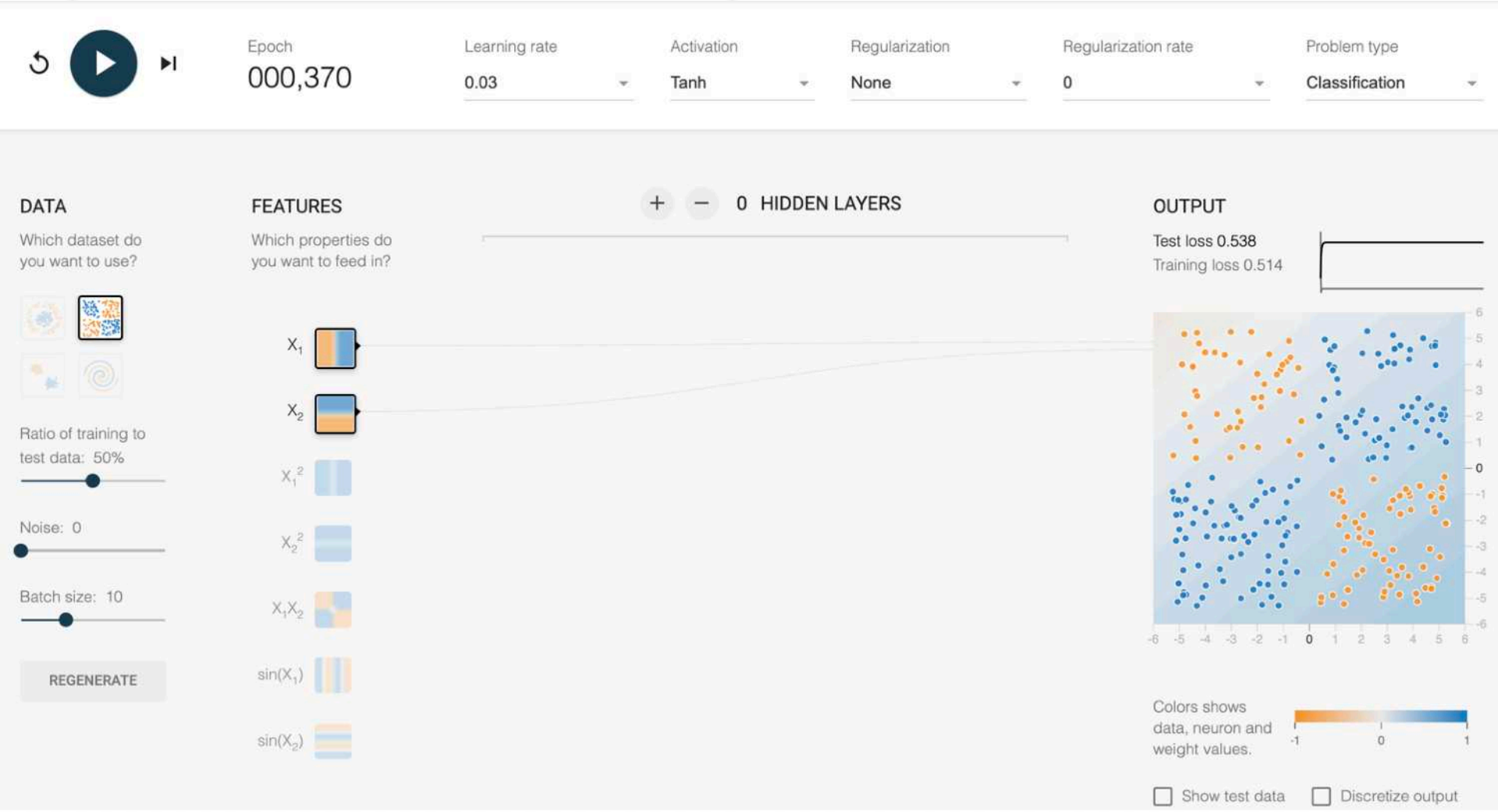
- Perceptron and Logistic Regression are linear classifiers
  - can model AND, OR operators
- What if classes are not linearly separable ?
  - cannot model XOR operator



Logistic Regression (0 hidden layers) illustration in https://playground.tensorflow.org/

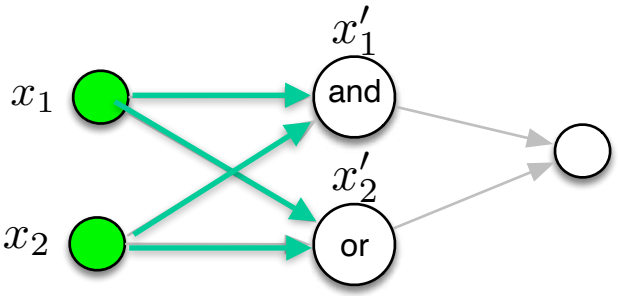


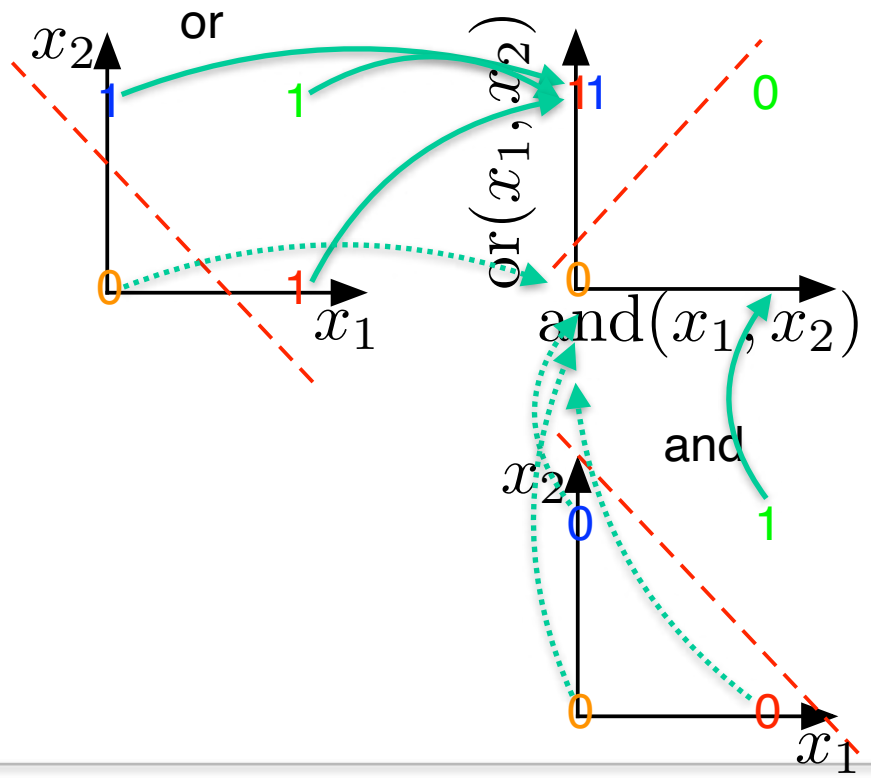
Logistic Regression (0 hidden layers) illustration in https://playground.tensorflow.org/



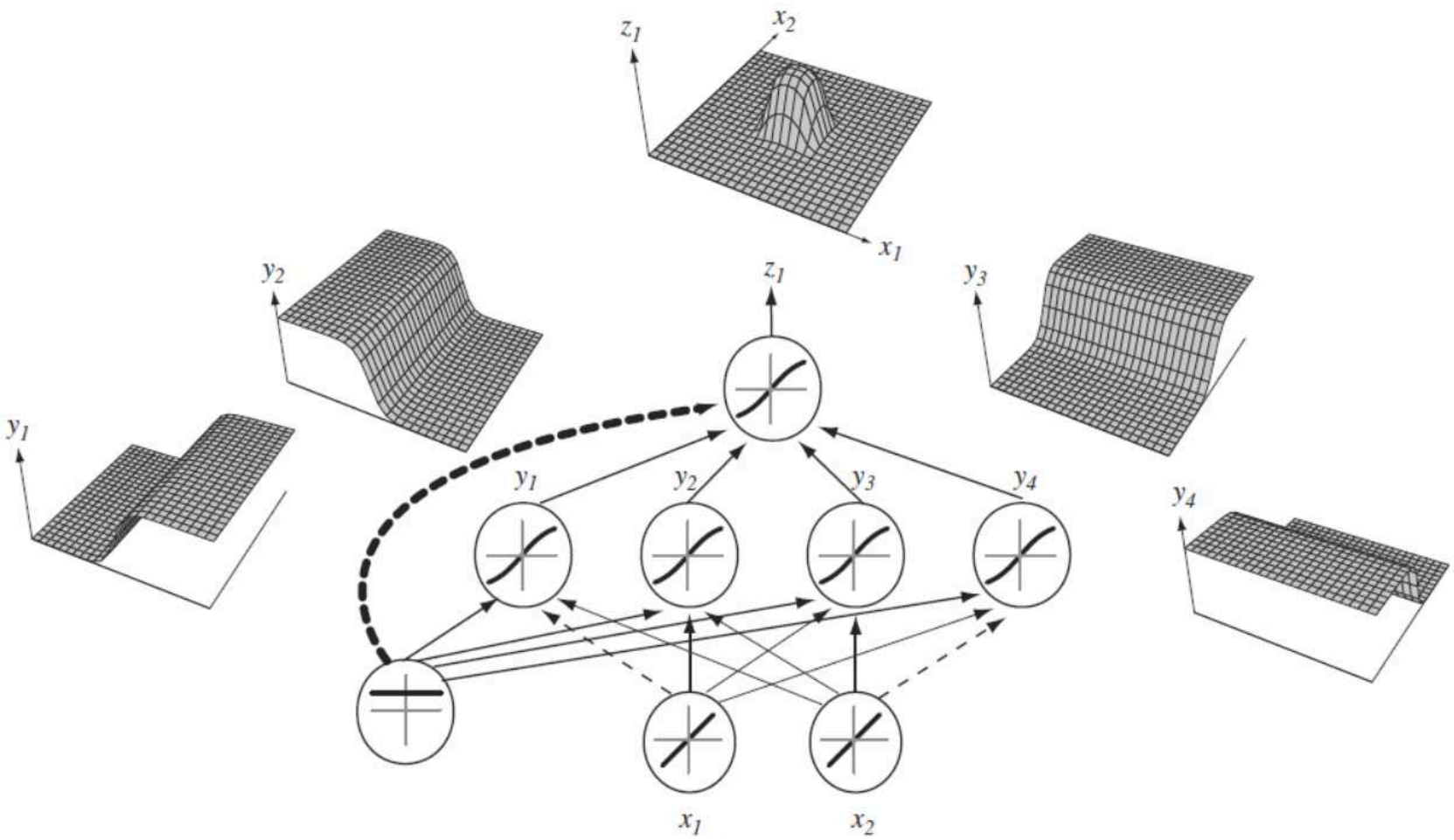
Limitation of linear classifiers

- **Solution to the XOR problem**:
  - project the input data $\mathbf{x}$ in a new space $\mathbf{x}'$
    $x_1^{[1]} = \textbf{AND}(x_1^{[0]}, x_2^{[0]})$
    $x_2^{[1]} = \textbf{OR}(x_1^{[0]}, x_2^{[0]})$
- We therefore need one hidden layer of projection
  - $\Rightarrow$ this is a 2 layers **Neural Network**
    $\Rightarrow$1 hidden projection
- In a Neural Network, f_1 and f_2 and trainable projections; they can be many more of such projections



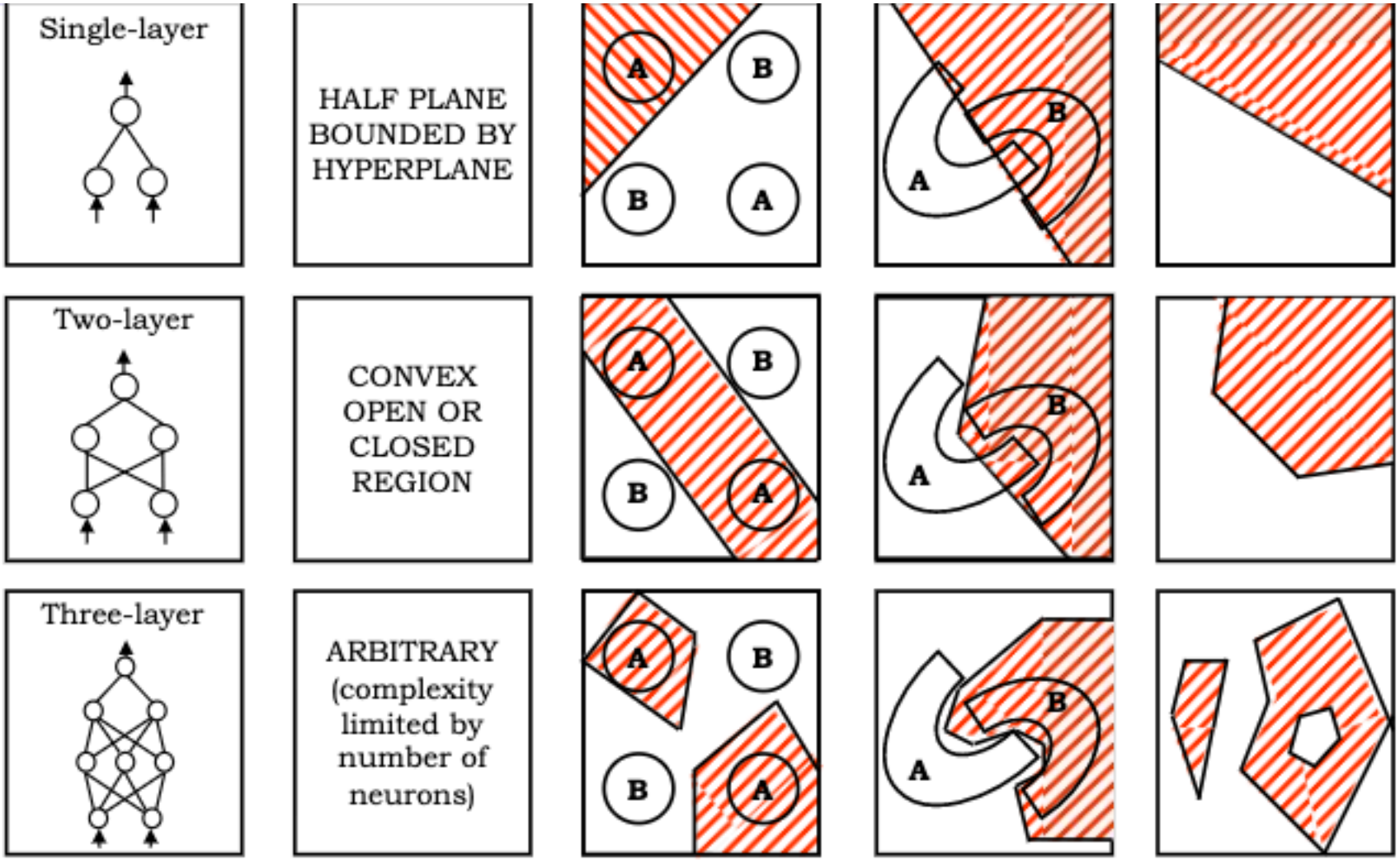


More than one layer



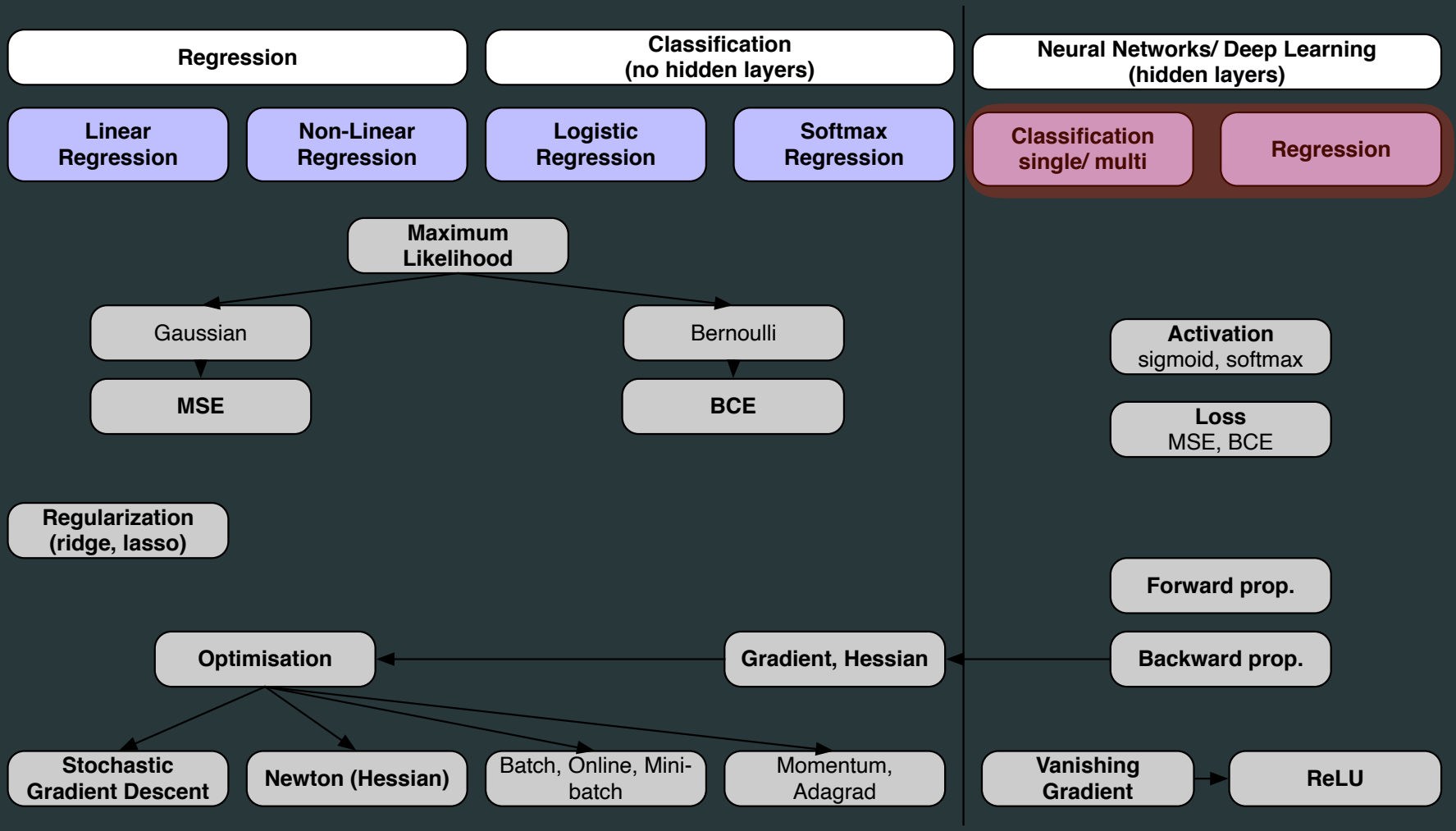
https://ukmie.tistory.com/203

More than one layer



https://ukmie.tistory.com/203

Neural Networks - Multi-Layer-Perceptron (MLP)

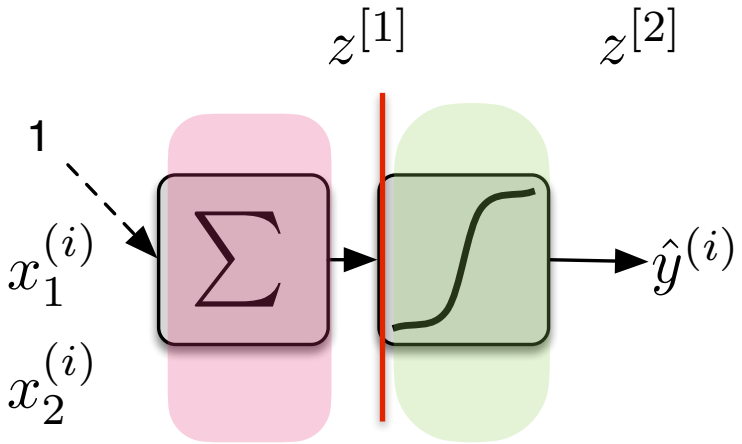


Neural Networks - Multi-Layer-Perceptron (MLP)
... with 1 layer (0 hidden layer) / Binary classification

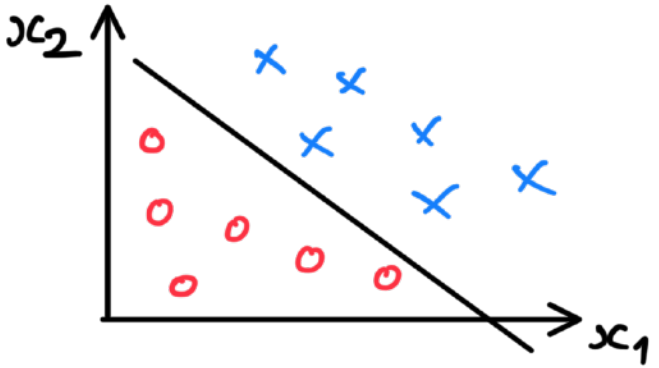
- Layer 1

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

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



Linear separating hyper-plane



Neural Networks - Multi-Layer-Perceptron (MLP)
... with 2 layer (1 hidden layer) / Binary classification

- Layer 1 (hidden)

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

$$z_1^{[2](i)} = \frac{1}{1 + e^{-z_1^{[1](i)}}}$$
$$z_2^{[2](i)} = \frac{1}{1 + e^{-z_2^{[1](i)}}}$$

- Layer 2

$$z_1^{[3](i)} = \theta_{0,1}^{[2]} + \theta_{1,1}^{[2]} z_1^{[2](i)} + \theta_{2,1}^{[1]} z_2^{[2](i)}$$

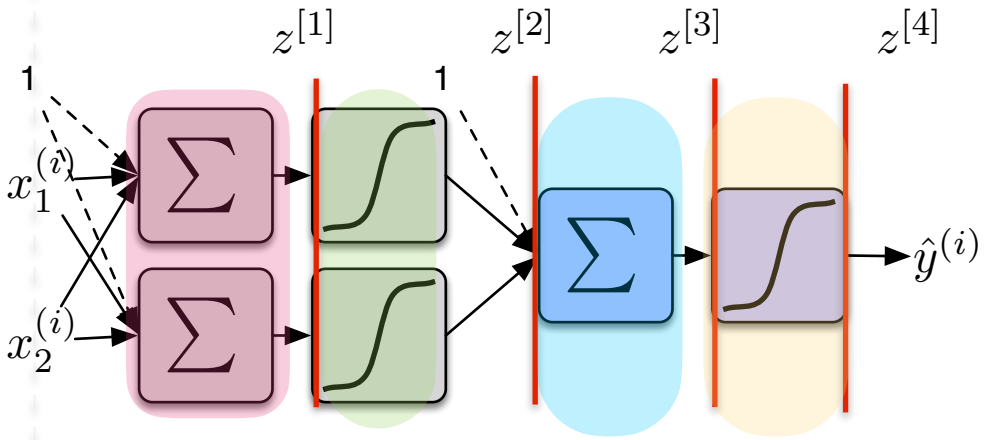
$$z^{[4](i)} = \frac{1}{1 + e^{-z_1^{[3](i)}}}$$

- Output

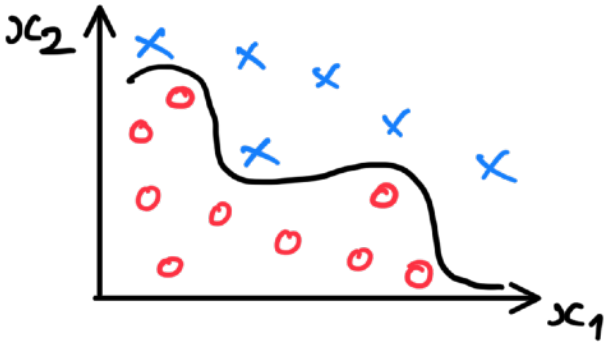
$$\hat{y}^{(i)} = z^{[4]}$$

- Cost:
  - minimise the binary cross-entropy

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



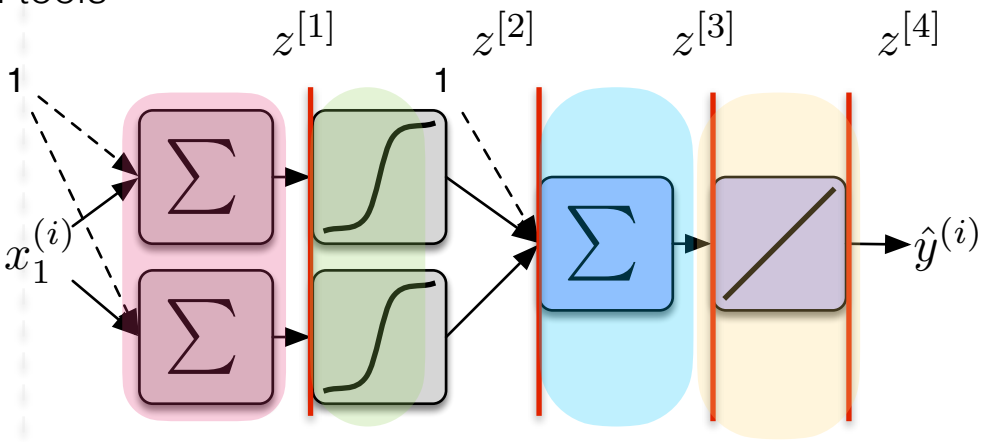
Non-linear separating hyper-plane



Neural Networks - Multi-Layer-Perceptron (MLP)
... with 2 layer (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

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

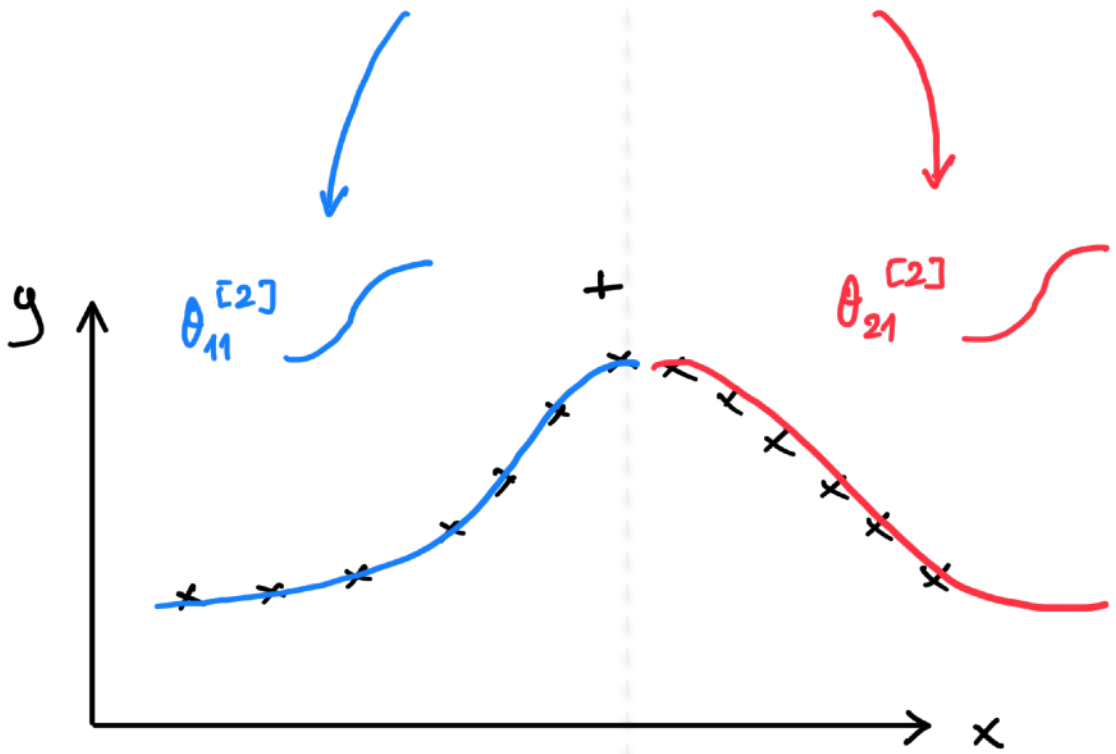
- Likelihood/ Cost

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

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

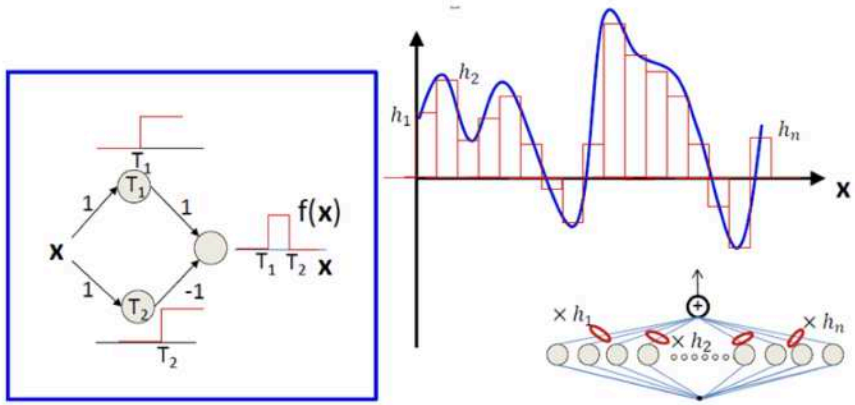
Neural Networks - Multi-Layer-Perceptron (MLP)
... with 2 layer (1 hidden layer) / Regression

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



Neural Network as Universal Function Approximator

- A Neural Network with only one hidden layer can **approximate any function** (with enough hidden neurons)

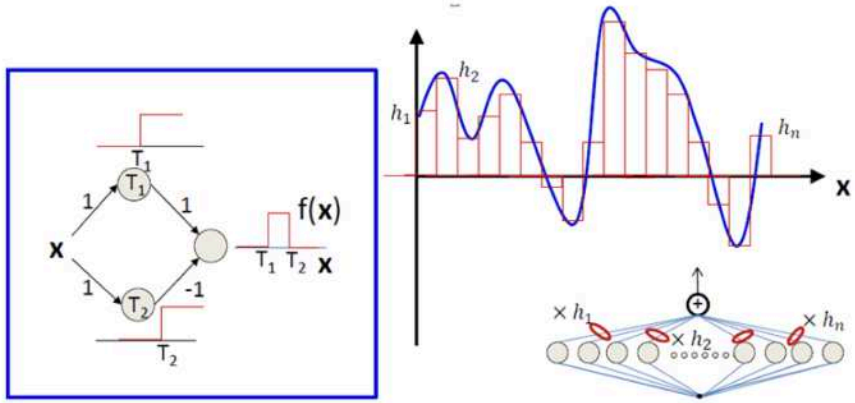


3 unit Multi Layer Perceptron using Step function to approximate a continuous function

https://medium.com/analytics-vidhya/neural-networks-and-the-universal-approximation-theorem-e5c387982eed

Neural Network as Universal Function Approximator

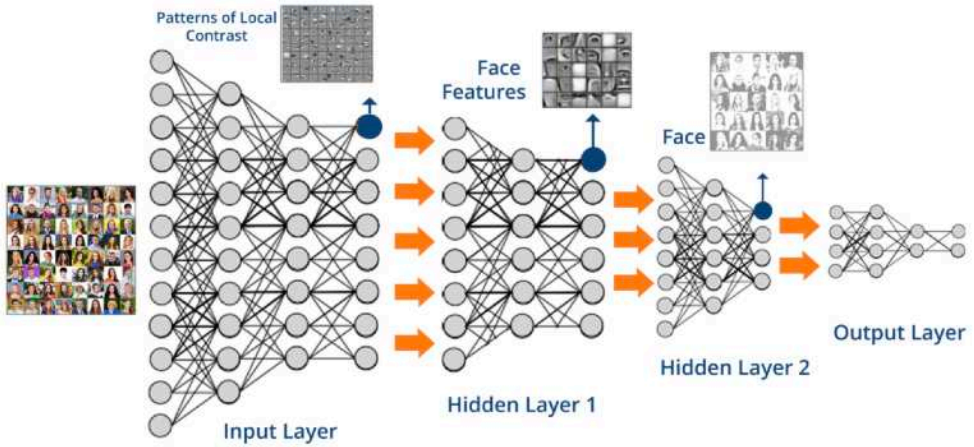
- A Neural Network with only one hidden layer can **approximate any function** (with enough hidden neurons)



3 unit Multi Layer Perceptron using Step function to approximate a continuous function

https://medium.com/analytics-vidhya/neural-networks-and-the-universal-approximation-theorem-e5c387982eed

- Deep Learning is about making the approximation of the function **progressively** (with several hidden layers of fewer neurons)

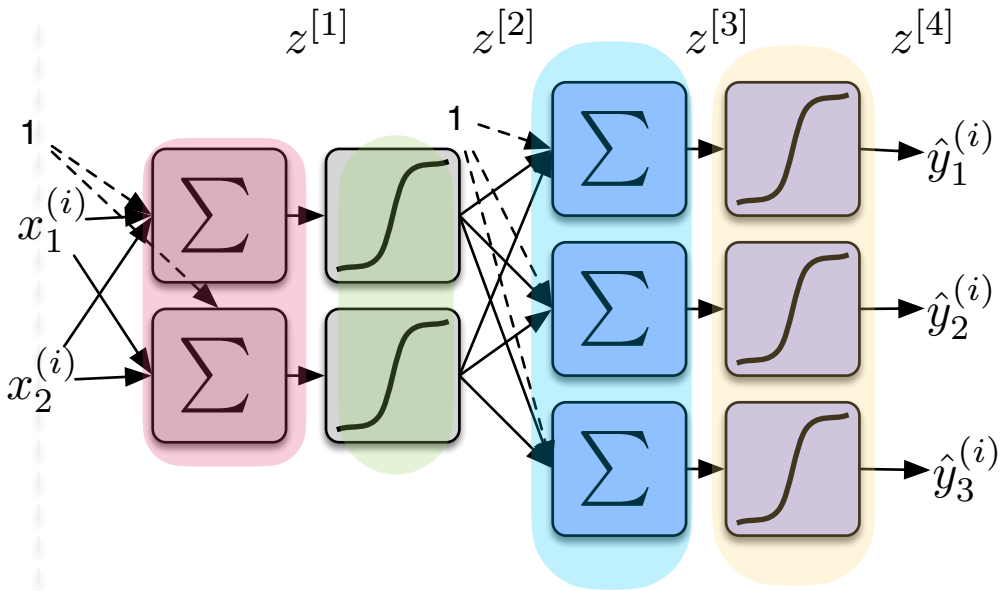


https://thedatascientist.com/what-deep-learning-is-and-isnt/

Neural Networks - Multi-Layer-Perceptron (MLP)
... with 2 layer (1 hidden layer) / Multi-Class classification

- Output

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



- Likelihood

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

- Cost= minimise the cross-entropy

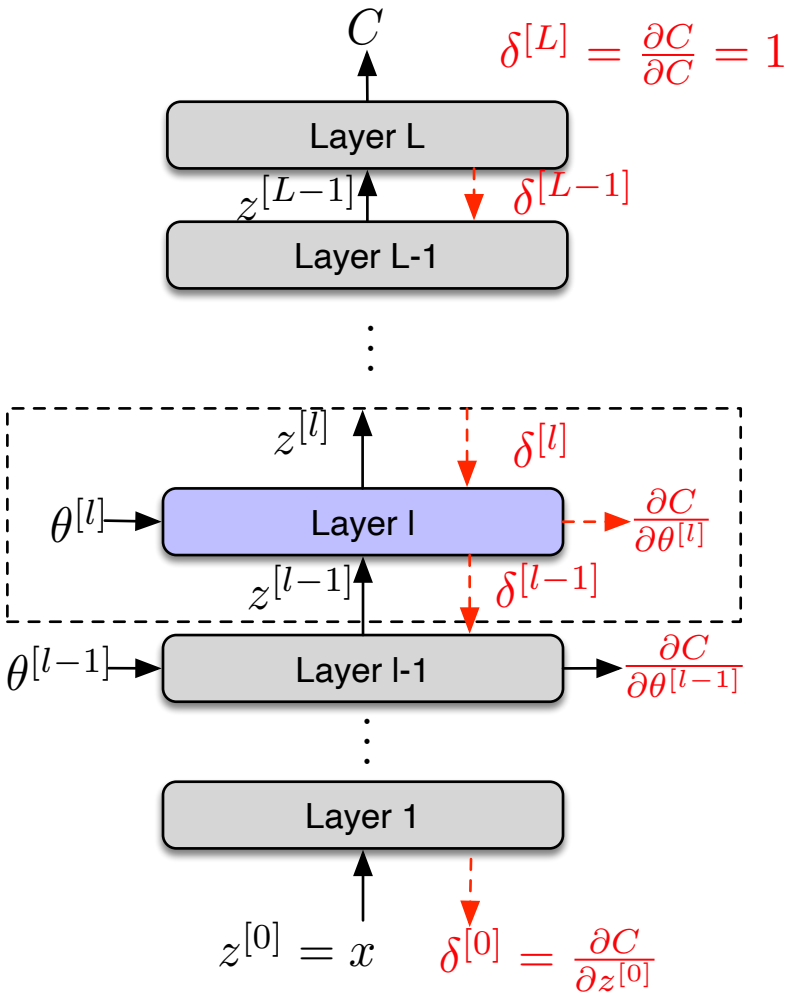
$$C(\theta) = -\log p(y \,|\, x, \theta)$$
$$= -\sum_{i=1}^{n}\sum_{c=1}^{3} \mathbb{I}_c(y^{(i)}) \log \frac{e^{\mathbf{z}c^{(i)}}}{\Sigma}$$

Separating hyper-planes

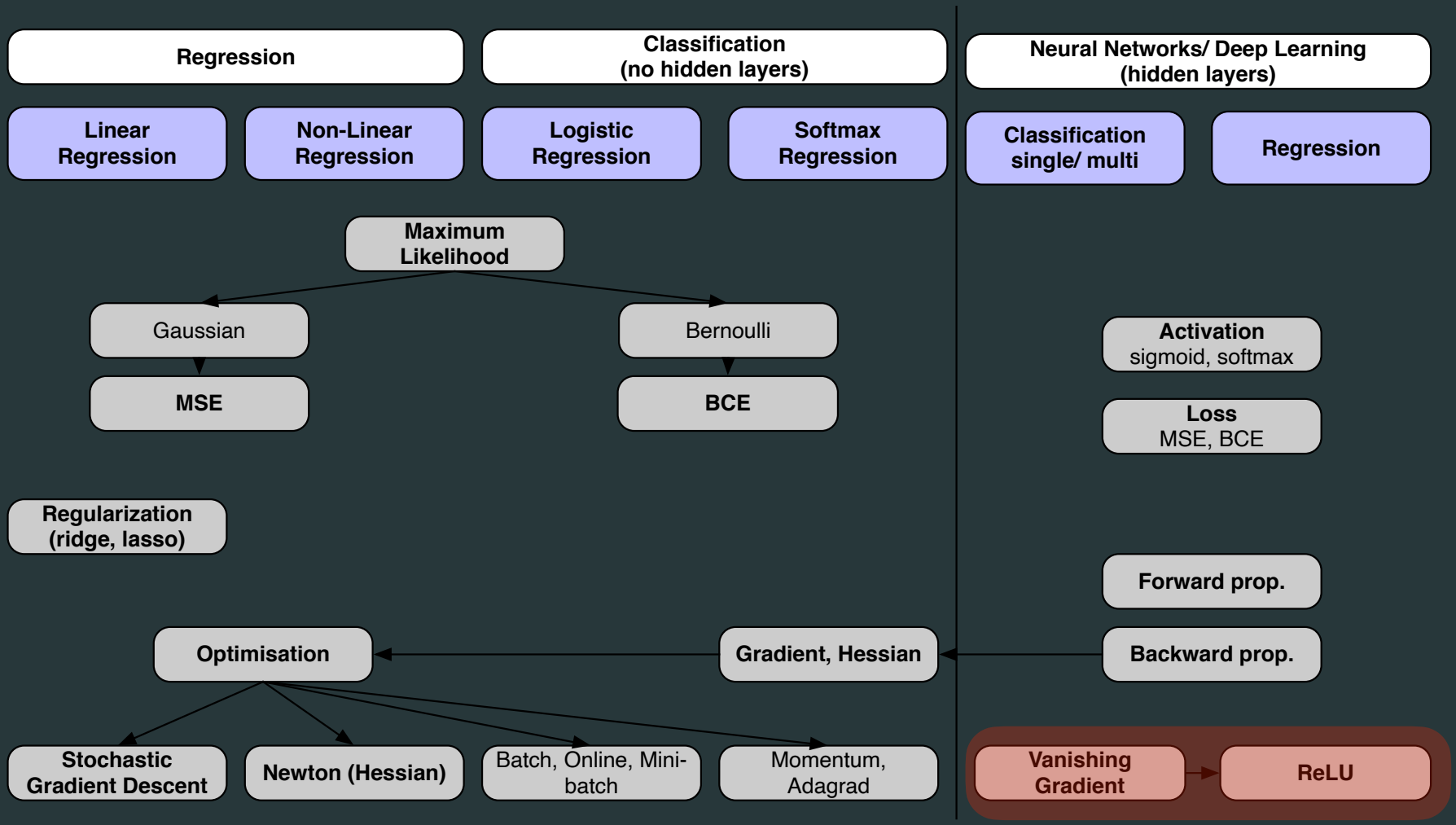
Deep Learning

Deep Learning

- Deep Learning ?
  - neural network with many many layers
  - vanishing gradient problem (sigmoid, ReLu)
  - sequential ≠ recursive



Vanishing gradient



Activation functions

Sigmoid σ .

– **Sigmoid function**

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

– **Derivative:**

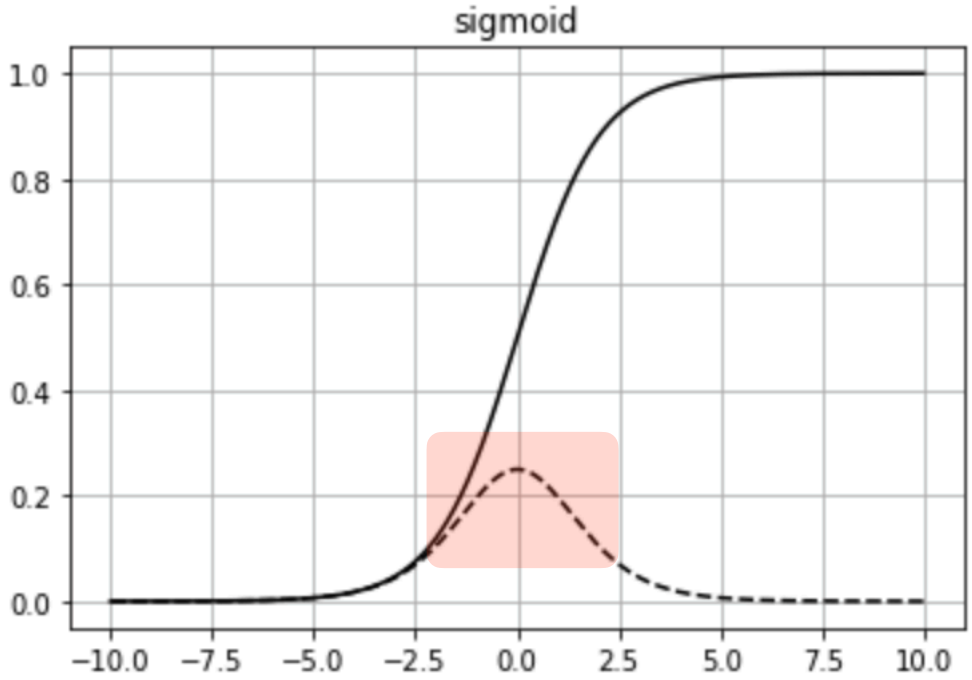
$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$
$$g'(z) = a(1-a)$$

– Proof:

$$\sigma'(z) = --e^{-z}\frac{1}{(1+e^{-z})^2}$$
$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$
$$= \frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right)$$
$$= \sigma(z)(1-\sigma(z))$$

– **Other properties:**

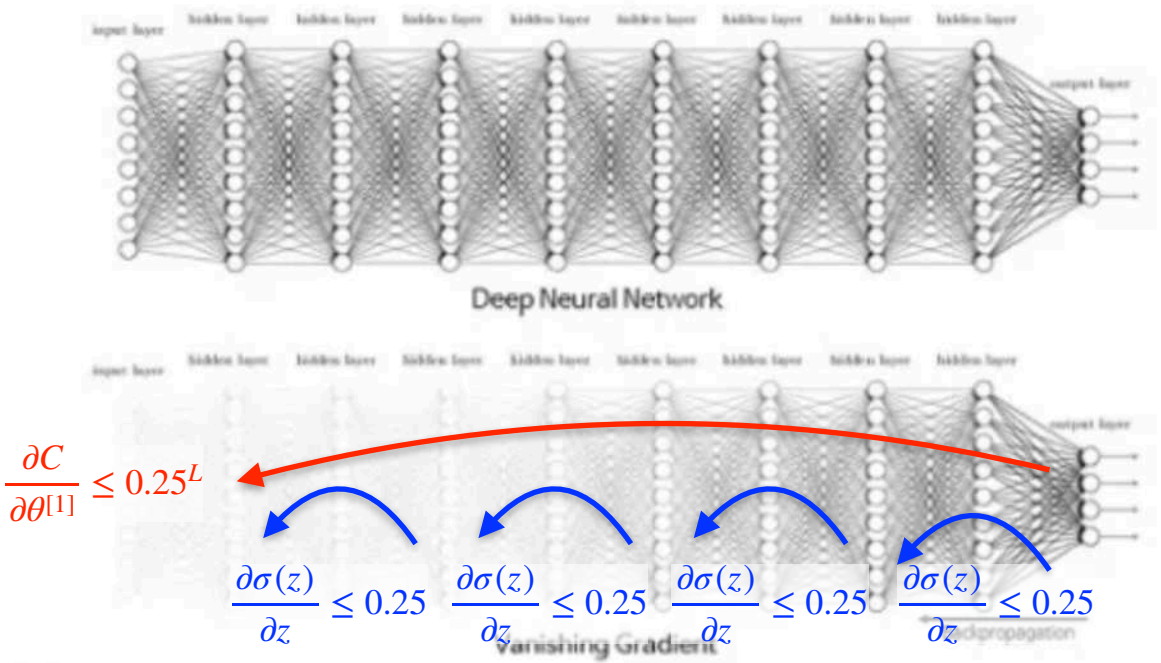
$$\sigma(-z) = 1-\sigma(z)$$



maximum value is 0.25 !!!

Vanishing Gradient

- **Problem:** $\max_z \sigma'(z) = 0.25$!!!
  - The deeper the network, the fastest the gradient diminishes/vanishes during back-propagation
  - For L layers, $\frac{\partial C}{\partial \theta^{[1]}} \propto \frac{\partial \sigma(z^{[L]})}{\partial z^{[L]}} \ldots \frac{\partial \sigma(z^{[L-1]})}{\partial z^{[L-1]}} \ldots \frac{\partial \sigma(z^{[L-2]})}{\partial z^{[L-2]}} \ldots \frac{\partial \sigma(z^{[1]})}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial \theta^{[1]}} \leq 0.25^L \simeq 0,$
  - there is no gradient anymore ⇒we cannot learn $\theta^{[1]}$ using SGD ⇒ **the network stop learning !**



https://miro.medium.com/v2/resize:fit:1400/format:webp/1*-rPcuafUSo-SwZF121y_rQ.png

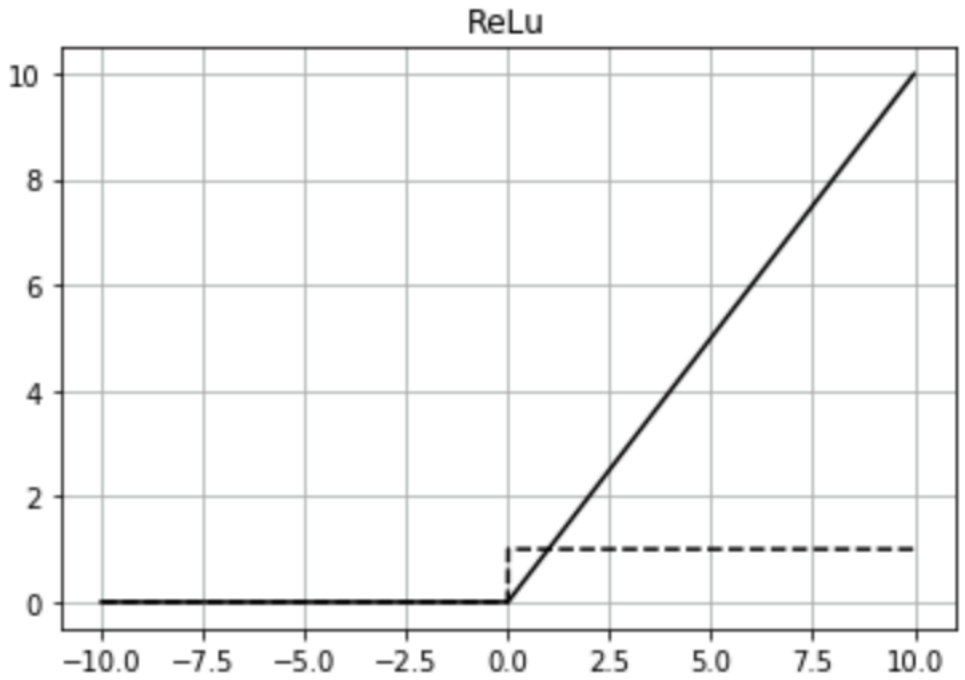
ReLU (Rectified Linear Units)

– **ReLU function**

$$a = g(z) = \max(0,z)$$

– **Derivative**

$$g'(x) = 1 \quad \text{if } z > 0$$
$$= 0 \quad \text{if } z \leq 0$$



_ For L layers, $\frac{\partial C}{\partial \theta^{[1]}} \propto \frac{\partial \text{ReLU}(z^{[L]})}{\partial z^{[L]}} \ldots \frac{\partial \text{ReLU}(z^{[L-1]})}{\partial z^{[L-1]}} \ldots \frac{\partial \text{ReLU}(z^{[L-2]})}{\partial z^{[L-2]}} \ldots \frac{\partial \text{ReLU}(z^{[1]})}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial \theta^{[1]}} \leq 1^L$,

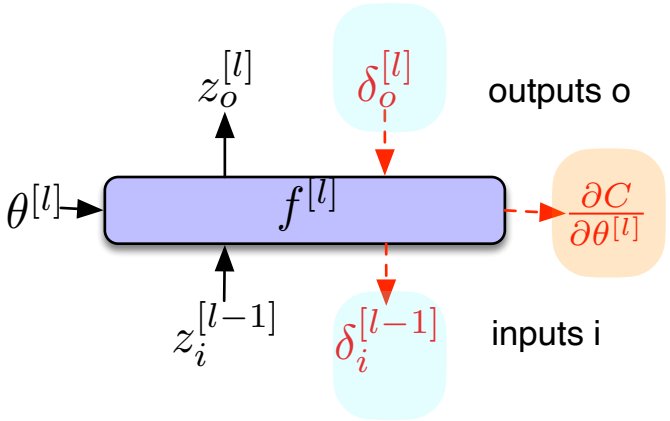
Deep Learning

Deep Learning
Forward/ Backward propagation

- **Layer** l
  - Input: $\mathbf{z}^{[l-1]}$
  - Output: $\mathbf{z}^{[l]}$

- **Forward**

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



- **Backward**
  - Gradients to propagate back
    - matrix form

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

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

    - scalar form

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

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

  - Local gradients for parameters updating
    - matrix form

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

    - scalar form

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

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

Deep Learning
Linear layer

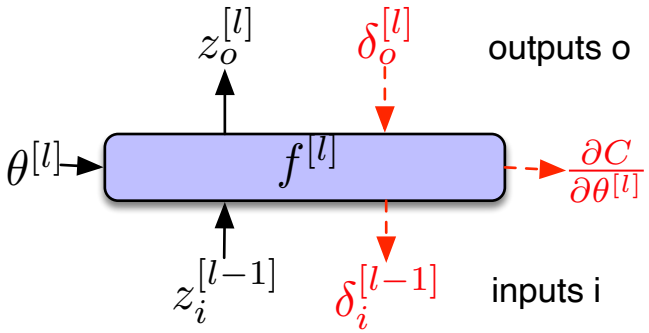
- **Forward**

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

- **Backward**

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

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



Deep Learning
ReLu layer

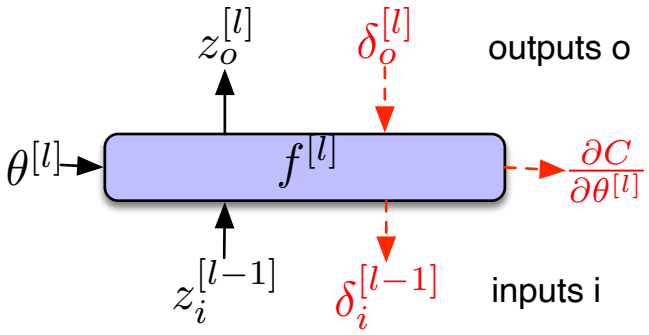
- **Forward**

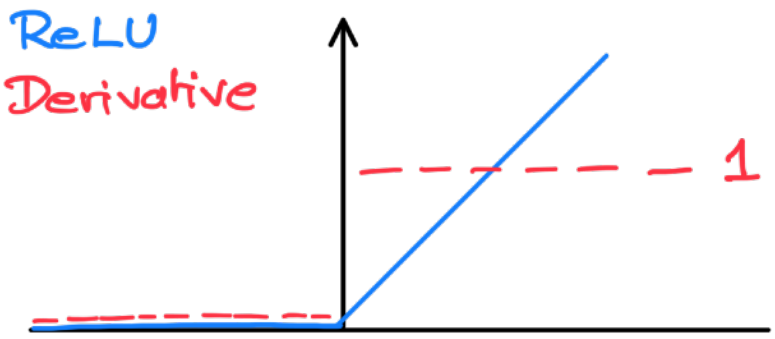
$$z_o = f_o(z_o^{[l-1]}) = \max(0, z_o^{[l-1]})$$

- **Backward**

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

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





End Lecture 1