Apprentissage, réseaux de neurones et modèles graphiques
(RCP209)
Neural Networks and Deep Learning

Nicolas Thome
Prenom.Nom@cnam.fr
http://cedric.cnam.fr/vertigo/Cours/ml2/

Département Informatique
Conservatoire National des Arts et Métiers (Cnam)
8 Weeks on Deep Learning and Structured Prediction

- Week 1-5: Deep Learning
- Week 6-8: Structured Prediction and Applications
Outline

1. Context
2. Neural Networks
3. Training Deep Neural Networks
Big Data

- Superabundance of data: images, videos, audio, text, use traces, etc

- Obvious need to access, search, or classify these data: **Recognition**

- Huge number of applications: mobile visual search, robotics, autonomous driving, augmented reality, medical imaging etc

- Leading track in major ML/CV conferences during the last decade
Recognition and classification

- Classification: assign a given data to a given set of pre-defined classes
- Recognition much more general than classification, e.g.
  - Object Localization in images
  - Ranking for document indexing
  - Sequence prediction for text, speech, audio, etc
- Many tasks can be cast as classification problems
  \( \Rightarrow \) importance of classification
Focus on Visual Recognition: Perceiving Visual World

- Visual Recognition: archetype of low-level signal understanding
- Supposed to be a master class problem in the early 80’s
- Certainly the most impacted topic by deep learning

- Scene categorization
- Object localization
- Context & Attribute recognition
- Rough 3D layout, depth ordering
- Rich description of scene, e.g. sentences
Recognition of low-level signals

Challenge: filling the semantic gap

What we perceive vs What a computer sees

- Illumination variations
- View-point variations
- Deformable objects
- Intra-class variance
- etc

⇒ How to design "good" intermediate representation?
Deep Learning (DL) & Recognition of low-level signals

- DL: breakthrough for the recognition of low-level signal data
- Before DL: handcrafted intermediate representations for each task
  - Needs expertise (PhD level) in each field
  - Weak level of semantics in the representation

VISION

SPEECH

@Kokkinos
Deep Learning (DL) & Recognition of low-level signals

- DL: breakthrough for the recognition of low-level signal data
- Since DL: automatically **learning intermediate representations**
  - ⊕ Outstanding experimental performances >> handcrafted features
  - ⊕ Able to learn high level intermediate representations
  - ⊕ Common learning methodology ⇒ field independent, no expertise

@Kokkinos
Deep Learning (DL) & Representation Learning

- DL: breakthrough for representation learning
  - Automatically learning intermediate levels of representation

- Ex: Natural language Processing (NLP)

@Kokkinos

@Socher
Outline

1. Context
2. Neural Networks
3. Training Deep Neural Networks
The Formal Neuron: 1943 [MP43]

- Basis of Neural Networks
- Input: vector $x \in \mathbb{R}^m$, i.e. $x = \{x_i\}_{i \in \{1, 2, \ldots, m\}}$
- Neuron output $\hat{y} \in \mathbb{R}$: scalar
The Formal Neuron: 1943 [MP43]

- Mapping from \( x \) to \( \hat{y} \):
  1. Linear (affine) mapping: \( s = w^T x + b \)
  2. Non-linear activation function: \( f: \hat{y} = f(s) \)
The Formal Neuron: Linear Mapping

- Linear (affine) mapping: $s = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{m} w_i x_i + b$
  - $\mathbf{w}$: normal vector to an hyperplane in $\mathbb{R}^m \Rightarrow$ linear boundary
  - $b$: bias, shift the hyperplane position

2D hyperplane: line

$w^\top x + b = 0$

3D hyperplane: plane
The Formal Neuron: Activation Function

- $\hat{y} = f(w^T x + b)$, $f$ activation function
  - Popular $f$ choices: step, sigmoid, tanh

- Step (Heaviside) function: $H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$
Step function: Connection to Biological Neurons

- Formal neuron, step activation $H$: $\hat{y} = H(w^Tx + b)$
  - $\hat{y} = 1$ (activated) $\iff w^Tx \geq -b$
  - $\hat{y} = 0$ (unactivated) $\iff w^Tx < -b$

- Biological Neurons: output activated
  $\iff$ input weighted by synaptic weight $\geq$ threshold
Sigmoid Activation Function

- Neuron output \( \hat{y} = f(\mathbf{w}^\top \mathbf{x} + b) \), \( f \) activation function
- Sigmoid: \( \sigma(z) = \frac{1}{1 + e^{-az}} \)

- \( a \uparrow \): more similar to step function (step: \( a \to \infty \))
- Sigmoid: linear and saturating regimes
The Formal neuron: Application to Binary Classification

- Binary Classification: label input $x$ as belonging to class 1 or 0
- Neuron output with sigmoid: $\hat{y} = \frac{1}{1 + e^{-a(w^T x + b)}}$
- Sigmoid: probabilistic interpretation $\Rightarrow \hat{y} \sim P(1/x)$
  - Input $x$ classified as 1 if $P(1/x) > 0.5 \iff w^T x + b > 0$
  - Input $x$ classified as 0 if $P(1/x) < 0.5 \iff w^T x + b < 0$
  $\Rightarrow \text{sign}(w^T x + b)$: linear boundary decision in input space!

bias $b$ only changes the position of the riff
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2 \), \( x = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( w = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = w^T x + b \)
The Formal neuron: Toy Example for Binary Classification

- 2d example: $m = 2$, $x = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]
- Linear mapping: $w = [1; 1]$ and $b = -2$
- Result of linear mapping: $s = w^T x + b$
- Sigmoid activation function: $\hat{y} = \left(1 + e^{-a(w^T x + b)}\right)^{-1}$, $a = 10$
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2, \ x = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( w = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = w^T x + b \)
- Sigmoid activation function: \( \hat{y} = \left(1 + e^{-a(w^T x + b)}\right)^{-1}, \ a = 1 \)
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2, \ x = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( \mathbf{w} = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = \mathbf{w}^T \mathbf{x} + b \)
- Sigmoid activation function: \( \hat{y} = \left(1 + e^{-a(\mathbf{w}^T \mathbf{x} + b)}\right)^{-1}, \ a = 0.1 \)
From Formal Neuron to Neural Networks

- **Formal Neuron:**
  1. A single scalar output
  2. Linear decision boundary for binary classification

- **Single scalar output:** limited for several tasks
  - Ex: multi-class classification, e.g. MNIST or CIFAR
Perceptron and Multi-Class Classification

- **Formal Neuron**: limited to binary classification
- **Multi-Class Classification**: use several output neurons instead of a single one ! ⇒ **Perceptron**
- **Input** $x$ in $\mathbb{R}^m$
- **Output neuron** $\hat{y}_1$ is a formal neuron:
  - Linear (affine) mapping: $s_1 = w_1^T x + b_1$
  - Non-linear activation function: $f$: $\hat{y}_1 = f(s_1)$
- **Linear mapping parameters**:
  - $w_1 = \{w_{11}, \ldots, w_{m1}\} \in \mathbb{R}^m$
  - $b_1 \in \mathbb{R}$
Perceptron and Multi-Class Classification

- Input $x$ in $\mathbb{R}^m$
- Output neuron $\hat{y}_k$ is a formal neuron:
  - Linear (affine) mapping: $s_k = w_k^T x + b_k$
  - Non-linear activation function: $f$:
    $\hat{y}_k = f(s_k)$
- Linear mapping parameters:
  - $w_k = \{w_{1k}, ..., w_{mk}\} \in \mathbb{R}^m$
  - $b_k \in \mathbb{R}$
Perceptron and Multi-Class Classification

- Input $x$ in $\mathbb{R}^m$ ($1 \times m$), output $\hat{y}$: concatenation of $K$ formal neurons
- Linear (affine) mapping $\sim$ matrix multiplication: $s = xW + b$
  - $W$ matrix of size $m \times K$ - columns are $w_k$
  - $b$: bias vector - size $1 \times K$
- Element-wise non-linear activation: $\hat{y} = f(s)$
Perceptron and Multi-Class Classification

- **Soft-max Activation:**
  \[
  \hat{y}_k = f(s_k) = \frac{e^{s_k}}{\sum_{k'=1}^{K} e^{s_{k'}}}
  \]

- **Probabilistic interpretation for multi-class classification:**
  - Each output neuron \(\Leftrightarrow\) class
  - \(\hat{y}_k \sim P(k|x, w)\)

\[\Rightarrow\] **Logistic Regression (LR) Model!**
2d Toy Example for Multi-Class Classification

- \( x = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \), \( \hat{y} \): 3 outputs (classes)

Linear mapping for each class:
\[
s_k = w_k^\top x + b_k
\]

Soft-max output:
\[
P(k/x, W)
\]

- \( w_1 = [1; 1], \ b_1 = -2 \)
- \( w_2 = [0; -1], \ b_2 = 1 \)
- \( w_3 = [1; -0.5], \ b_3 = 10 \)
2d Toy Example for Multi-Class Classification

- $x = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y}$: 3 outputs (classes)

Soft-max output: $P(k|x, W)$

Class Prediction:
$$k^* = \arg \max_k P(k|x, W)$$
Beyond Linear Classification

X-OR Problem

- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- X-OR: NOT 1 and 2 OR NOT 2 AND 1
  - X-OR: Non linear decision function
Beyond Linear Classification

- LR: limited to linear boundaries
- **Solution**: add a layer!

- Input $x$ in $\mathbb{R}^m$, e.g. $m = 4$
- Output $\hat{y}$ in $\mathbb{R}^K$ ($K$ # classes), e.g. $K = 2$
- **Hidden layer** $h$ in $\mathbb{R}^L$
Multi-Layer Perceptron

- **Hidden layer** $h$: $x$ projection to a new space $\mathbb{R}^L$

- Neural Net with $\geq 1$ hidden layer: Multi-Layer Perceptron (MLP)

- $h$: intermediate representations of $x$ for classification $\hat{y}$: $h = f(xW + b)$

- Mapping from $x$ to $\hat{y}$: non-linear boundary $\Rightarrow$ activation $f$ crucial!
Deep Neural Networks

- Adding more hidden layers: Deep Neural Networks (DNN) ⇒ Basis of Deep Learning
- Each layer $h^l$ projects layer $h^{l-1}$ into a new space
- Gradually learning intermediate representations useful for the task
Conclusion

- Deep Neural Networks: applicable to classification problems with non-linear decision boundaries

- Visualize prediction from fixed model parameters
- Reverse problem: **Supervised Learning**
Outline

1. Context
2. Neural Networks
3. Training Deep Neural Networks
Training Multi-Layer Perceptron (MLP)

- Input $x$, output $y$
- A parametrized model $x \Rightarrow y$: $f_w(x_i) = \hat{y}_i$
- Supervised context: training set $A = \{(x_i, y^*_i)\}_{i \in \{1,2,...,N\}}$
  - A loss function $\ell(\hat{y}_i, y^*_i)$ for each annotated pair $(x_i, y^*_i)$
- Assumptions: parameters $w \in \mathbb{R}^d$ continuous, $\mathcal{L}$ differentiable
- Gradient $\nabla_w = \frac{\partial \mathcal{L}}{\partial w}$: steepest direction to decrease loss $\mathcal{L}$
MLP Training

- Gradient descent algorithm:
  - Initialize parameters \( w \)
  - Update: \( w^{(t+1)} = w^{(t)} - \eta \frac{\partial L}{\partial w} \)
  - Until convergence, e.g. \( \| \nabla w \|^2 \approx 0 \)
Gradient Descent

Update rule: \[ w^{(t+1)} = w^{(t)} - \eta \frac{\partial L}{\partial w} \] \( \eta \) learning rate

- **Convergence ensured?** \( \Rightarrow \) provided a "well chosen" learning rate \( \eta \)
Gradient Descent

Update rule: \[ w^{(t+1)} = w^{(t)} - \eta \frac{\partial L}{\partial w} \]

- **Global minimum ?**
  \[ \Rightarrow \text{convex a) vs non convex b) loss } L(w) \]

**Convex**

\[ w^* \]

- **Global cost minimum**

**Non convex**

\[ \text{Local minima} \]

\[ \text{Starting pt.} \]

\[ \text{Global minima} \]
Supervised Learning: Multi-Class Classification

- Logistic Regression for multi-class classification
  \[ s_i = x_i W + b \]
- Soft-Max (SM): \( \hat{y}_k \sim P(k/x_i, W, b) = \frac{e^{s_k}}{\sum_{k'=1}^{K} e^{s_{k'}}} \)
- Supervised loss function: \( \mathcal{L}(W, b) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{y}_i, y_i^*) \)

1. \( y \in \{1; 2; \ldots; K\} \)
2. \( \hat{y}_i = \arg \max_k P(k/x_i, W, b) \)
3. \( \ell_{0/1}(\hat{y}_i, y_i^*) = \begin{cases} 
1 & \text{if } \hat{y}_i \neq y_i^* \\
0 & \text{otherwise} 
\end{cases} : 0/1 \text{ loss} \)
Logistic Regression Training Formulation

- Input $x_i$, ground truth output supervision $y_i^*$
- One hot-encoding for $y_i^*$:

$$y_{c,i}^* = \begin{cases} 1 & \text{if } c \text{ is the ground truth class for } x_i \\ 0 & \text{otherwise} \end{cases}$$
Logistic Regression Training Formulation

- Loss function: multi-class Cross-Entropy (CE) \( \ell_{CE} \)
- \( \ell_{CE} \): Kullback-Leiber divergence between \( y_i^* \) and \( \hat{y}_i \)

\[
\ell_{CE}(y_i, y_i^*) = KL(y_i^*, y_i) = - \sum_{c=1}^{K} y_{c,i}^* \log(\hat{y}_{c,i}) = -\log(\hat{y}_{c^*,i})
\]

- \( B \) KL asymmetric: \( KL(y_i, y_i^*) \neq KL(y_i^*, y_i) \) \( B \)

\[
KL(y_i^*, \hat{y}_i) = -\log(\hat{y}_{c^*,i}) = -\log(0.8) \approx 0.22
\]
Logistic Regression Training

- \( \mathcal{L}_{CE}(W, b) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{y}_i, y_i^*) = -\frac{1}{N} \sum_{i=1}^{N} \log(\hat{y}^*_i) \)

- \( \ell_{CE} \) smooth convex upper bound of \( \ell_{0/1} \)
  \( \Rightarrow \) gradient descent optimization

- Gradient descent: \( W^{(t+1)} = W^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial W} \)  
  \( b^{(t+1)} = b^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial b} \)

- **MAIN CHALLENGE:** computing \( \frac{\partial \mathcal{L}_{CE}}{\partial W} \) ?

  \( \Rightarrow \) Key Property: chain rule \( \frac{\partial x}{\partial z} = \frac{\partial x}{\partial y} \frac{\partial y}{\partial z} \)

  \( \Rightarrow \) Backpropagation of gradient error!
Chain Rule

\[
\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} \cdot \frac{\partial y}{\partial x}
\]

- Logistic regression:
  \[
  \frac{\partial l_{CE}}{\partial W} = \frac{\partial l_{CE}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial s_i} \cdot \frac{\partial s_i}{\partial W}
  \]
Logistic Regression Training: Backpropagation

\[ \frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}, \ell_{CE}(\hat{y}_i, y_i^*) = -\log(\hat{y}_{c^*}, i) \Rightarrow \text{Update for 1 example:} \]

1. \[ \frac{\partial \ell_{CE}}{\partial \hat{y}_i} = \frac{-1}{\hat{y}_{c^*}, i} = \frac{-1}{\hat{y}_i} \odot \delta_{c,c^*} \]

2. \[ \frac{\partial \ell_{CE}}{\partial s_i} = \hat{y}_i - y_i^* = \delta_i^y \]

3. \[ \frac{\partial \ell_{CE}}{\partial W} = x_i^T \delta_i^y \]
Logistic Regression Training: Backpropagation

- Whole dataset: data matrix $\mathbf{X} (N \times m)$, label matrix $\mathbf{Y}, \mathbf{Y}^* (N \times K)$

- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} \log(\hat{y}_{c*,i}), \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$

- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}} = \mathbf{Y} - \mathbf{Y}^* = \Delta y$

- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \mathbf{X}^T \Delta y$
Perceptron Training: Backpropagation

- Perceptron vs Logistic Regression: adding hidden layer (sigmoid)
- **Goal:** Train parameters $W^y$ and $W^h$ (+bias) with Backpropagation

$\Rightarrow$ computing

$$\frac{\partial L_{CE}}{\partial W^y} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^y}$$

and

$$\frac{\partial L_{CE}}{\partial W^h} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^h}$$

- Last hidden layer $\sim$ Logistic Regression
- First hidden layer: $\frac{\partial L_{CE}}{\partial W^h} = x_i^T \frac{\partial \ell_{CE}}{\partial u_i} \Rightarrow$ computing $\frac{\partial \ell_{CE}}{\partial u_i} = \delta^h_i$
Perceptron Training: Backpropagation

- Computing $\frac{\partial \ell_{CE}}{\partial u_i} = \delta^h_i \Rightarrow$ use chain rule: $\frac{\partial \ell_{CE}}{\partial u_i} = \frac{\partial \ell_{CE}}{\partial v_i} \frac{\partial v_i}{\partial h_i} \frac{\partial h_i}{\partial u_i}$
- ... Leading to: $\frac{\partial \ell_{CE}}{\partial u_i} = \delta^h_i = \delta^y_i \cdot \sigma'(h_i) = \delta^y_i \cdot \mathbf{W}^y \odot (h_i \odot (1 - h_i))$
Deep Neural Network Training: Backpropagation

- Multi-Layer Perceptron (MLP): adding more hidden layers
- Backpropagation update ~ Perceptron: assuming $\frac{\partial L}{\partial u_{l+1}} = \Delta^{l+1}$ known

\[
\frac{\partial L}{\partial w^{l+1}} = H_l^T \Delta^{l+1}
\]

Computing $\frac{\partial L}{\partial u_l} = \Delta^l$ ($= \Delta^{l+1}^T w^{l+1} \odot H_l \odot (1 - H_l)$ sigmoid)

\[
\frac{\partial L}{\partial w^l} = H_{l-1}^T \Delta^h_l
\]
Neural Network Training: Optimization Issues

- Classification loss over training set (vectorized $\mathbf{w}$, $\mathbf{b}$ ignored):

$$
\mathcal{L}_{CE}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{y}_i, y_i^*) = -\frac{1}{N} \sum_{i=1}^{N} \log(\hat{y}_{c^*}, i)
$$

- Gradient descent optimization:

$$
\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{w}}(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}}^{(t)}
$$

- Gradient $\nabla_{\mathbf{w}}^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial \mathbf{w}}(\mathbf{w}^{(t)})$ linearly scales wrt:
  - $\mathbf{w}$ dimension
  - Training set size

$\Rightarrow$ Too slow even for moderate dimensionality & dataset size!
Stochastic Gradient Descent

- **Solution**: approximate \( \nabla_w^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left( w^{(t)} \right) \) with subset of examples

\[ \Rightarrow \text{Stochastic Gradient Descent (SGD)} \]

- Use a single example (online):

\[ \nabla_w^{(t)} \approx \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left( w^{(t)} \right) \]

- Mini-batch: use \( B < N \) examples:

\[ \nabla_w^{(t)} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left( w^{(t)} \right) \]
Stochastic Gradient Descent

- **SGD**: approximation of the true Gradient $\nabla_w$!
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT**: much more parameter updates: online $\times N$, mini-batch $\frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets

Full gradient  SGD (online)  SGD (mini-batch)
Optimization: Learning Rate Decay

- Gradient descent optimization: $w^{(t+1)} = w^{(t)} - \eta \nabla w^{(t)}$
- $\eta$ setup ? $\Rightarrow$ open question
- Learning Rate Decay: decrease $\eta$ during training progress
  - Inverse (time-based) decay: $\eta_t = \frac{\eta_0}{1 + r \cdot t}$, $r$ decay rate
  - Exponential decay: $\eta_t = \eta_0 \cdot e^{-\lambda t}$
  - Step Decay $\eta_t = \eta_0 \cdot r^{t/t_u}$ ...

Exponential Decay ($\eta_0 = 0.1$, $\lambda = 0.1s$)

Step Decay ($\eta_0 = 0.1$, $r = 0.5$, $t_u = 10$)
Generalization and Overfitting

- **Learning:** minimizing classification loss $\mathcal{L}_{CE}$ over training set
  - Training set: sample representing data vs labels distributions
  - **Ultimate goal:** train a prediction function with low prediction error on the \textbf{true} (unknown) data distribution

\[
\mathcal{L}_{\text{train}} = 4, \quad \mathcal{L}_{\text{train}} = 9 \\
\mathcal{L}_{\text{test}} = 15, \quad \mathcal{L}_{\text{test}} = 13
\]

⇒ **Optimization ≠ Machine Learning!**  
⇒ **Generalization / Overfitting!**
Regularization

- **Regularization**: improving generalization, *i.e.* test (≠ *train*) performances
- Structural regularization: add *Prior* $R(w)$ in training objective:

$$L(w) = L_{CE}(w) + \alpha R(w)$$

- $L^2$ regularization: *weight decay*, $R(w) = ||w||^2$
  - Commonly used in neural networks
  - Theoretical justifications, generalization bounds (SVM)
- Other possible $R(w)$: $L^1$ regularization, dropout, *etc*
Regularization and hyper-parameters

- **Neural networks**: hyper-parameters to tune:
  - **Training parameters**: learning rate, weight decay, learning rate decay, # epochs, *etc*
  - **Architectural parameters**: number of layers, number neurones, non-linearity type, *etc*

- **Hyper-parameters tuning**: ⇒ improve generalization: estimate performances on a validation set
Neural networks: Conclusion

- Training issues at several levels: optimization, generalization, cross-validation
- Limits of fully connected layers and Convolutional Neural Nets? ⇒ next course!
Warren S McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics 5 (1943), no. 4, 115–133.