



# APPRENTISSAGE MACHINE & DEEP LEARNING

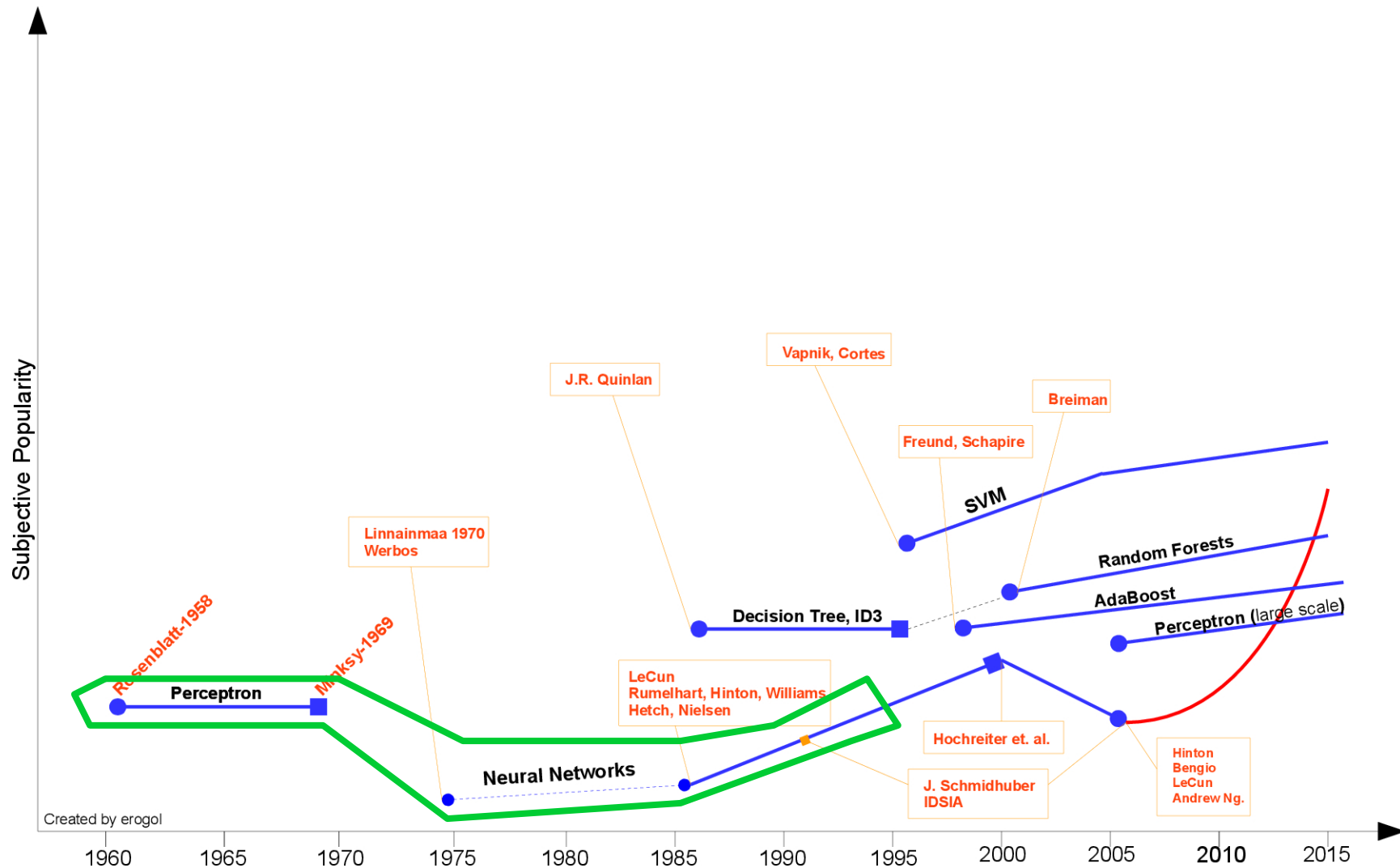
## Deep Learning

**A. Boulch**, A. Chan Hon Tong, S. Herbin, B. Le Saux



retour sur innovation

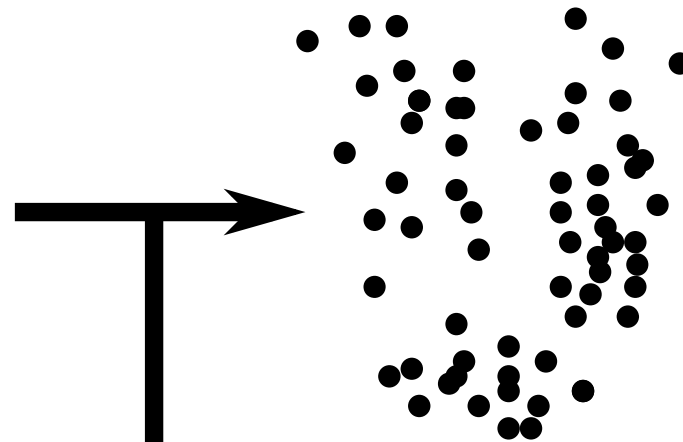
# Réseaux de neurones



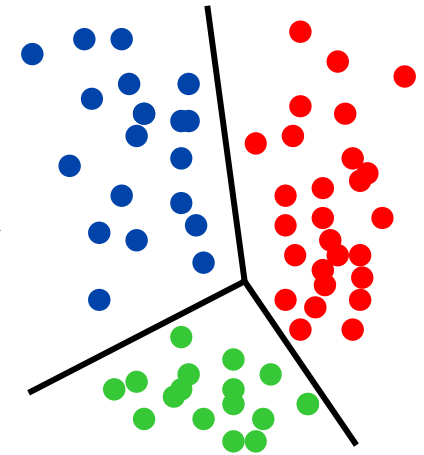
# Réseaux de neurones



Données



Espace  
adapté

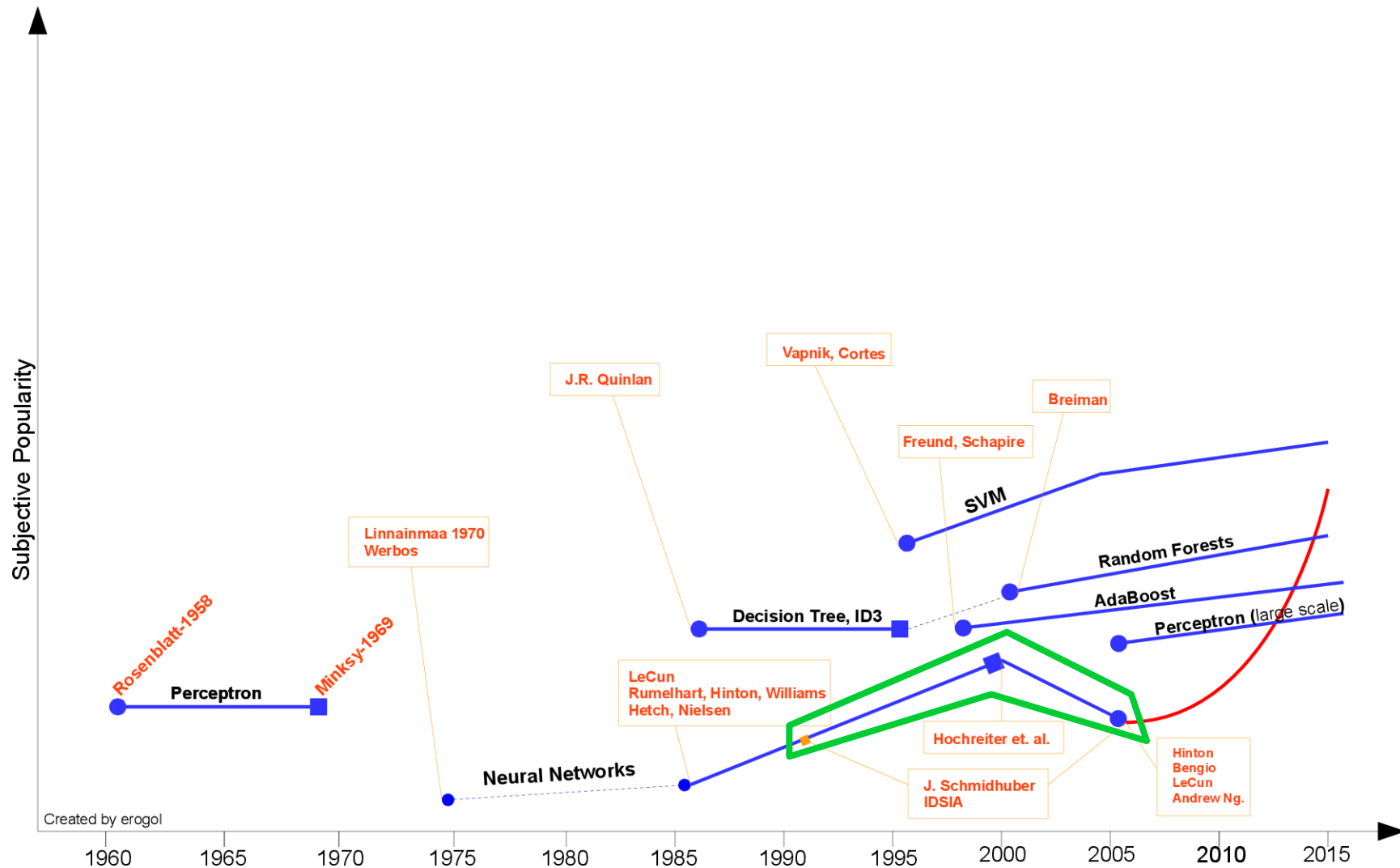


Objectif

Caractéristiques  
bien pensées

Apprentissage  
Expertise

# Réseaux de neurones



Created by erogol

# Deep learning 90's → 2005

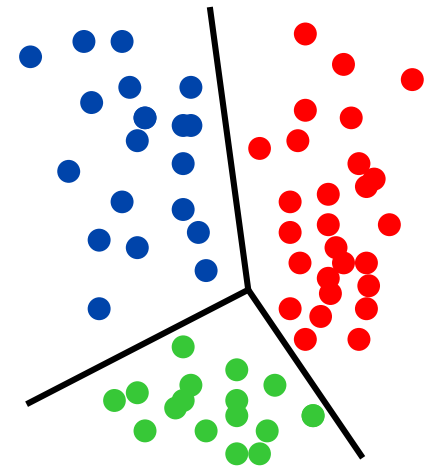
- Mise en place des premiers réseaux convolutifs



Données

Réseaux de  
neurones profonds

Recherche sur les  
architectures, les stratégies,  
la mise en forme des données



Objectif

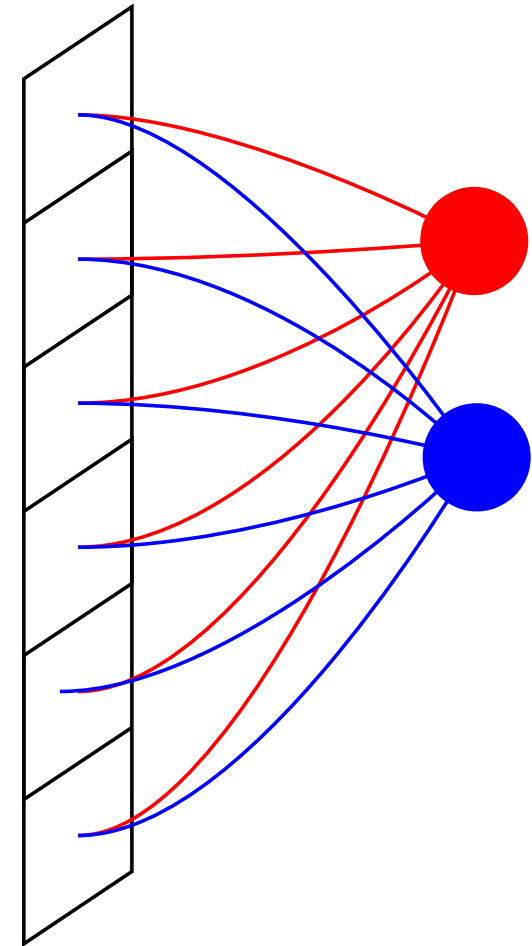
# Fully connected

Perceptron

(présentation précédente)

Un neurone est connecté à toutes les entrées

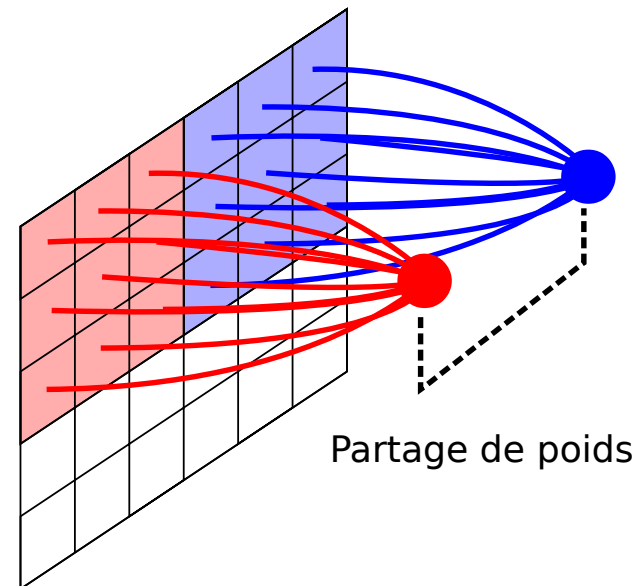
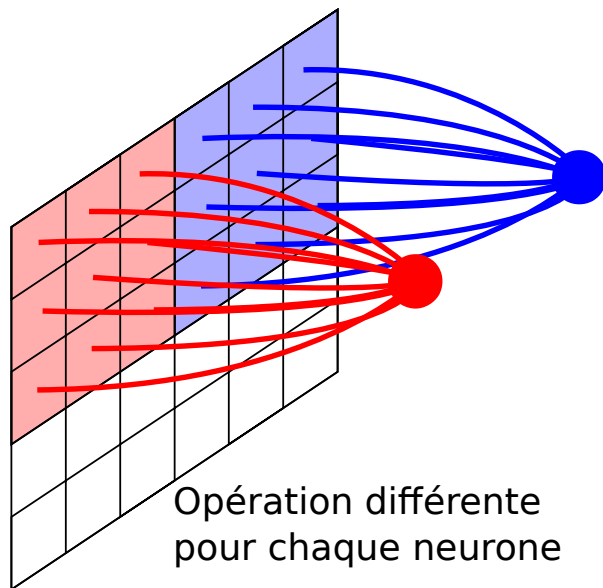
Si dimension d'entrée grande,  
beaucoup de paramètres



- MLP de plus en plus profonds (avant 2005)
  - Très grosses difficultés d'optimisation
  - Convergence difficile
  - Peu de données
  - Entraînement très long
- ⇒ Abandon progressif au profit des SVMs
  - Simples à utiliser
  - Preuves de convergence
  - Rapides

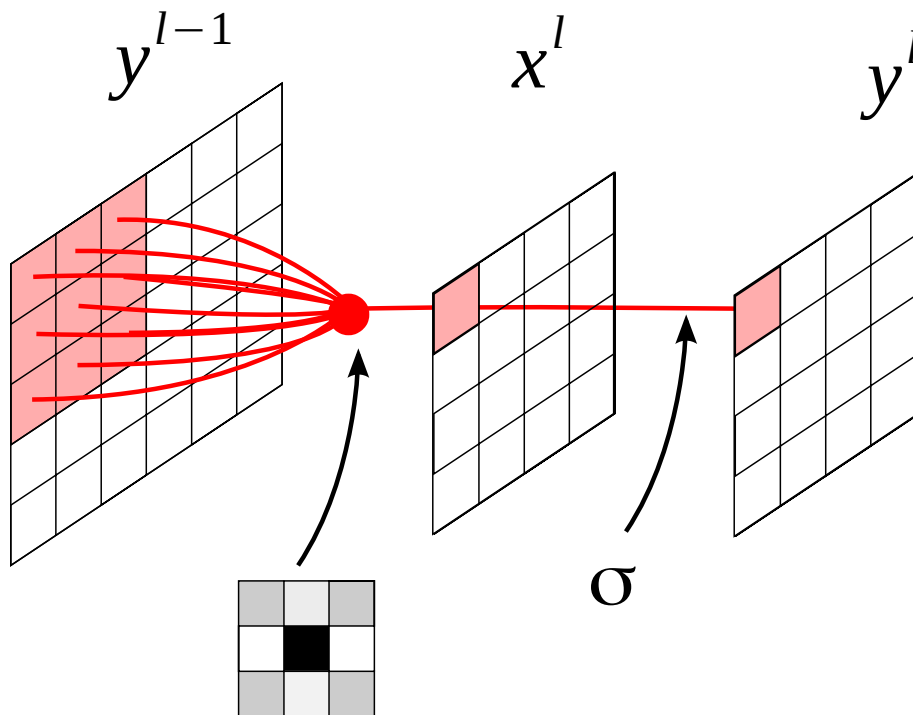
# Réseaux convolutifs

Adapté aux données structurées





# Convolution



Forward

$$x_{i,j}^l = \sum_a \sum_b \omega_{a,b} y_{i+a,j+b}^{l-1}$$

$$y_{i,j}^l = \sigma(x_{i,j}^l)$$

# Convolution

Backward, mise à jour des poids de la convolution

$$\frac{\partial E}{\partial \omega_{a,b}} = \sum_i \sum_j \frac{\partial E}{\partial x_{i,j}^l} \frac{\partial x_{i,j}^l}{\partial \omega_{a,b}} = \sum_i \sum_j \frac{\partial E}{\partial x_{i,j}^l} y_{i,j}^{l-1}$$

or

$$\frac{\partial E}{\partial x_{i,j}^l} = \frac{\partial E}{\partial y_{i,j}^l} \frac{\partial y_{i,j}^l}{\partial x_{i,j}^l} = \frac{\partial E}{\partial y_{i,j}^l} \sigma'(x_{i,j}^l)$$

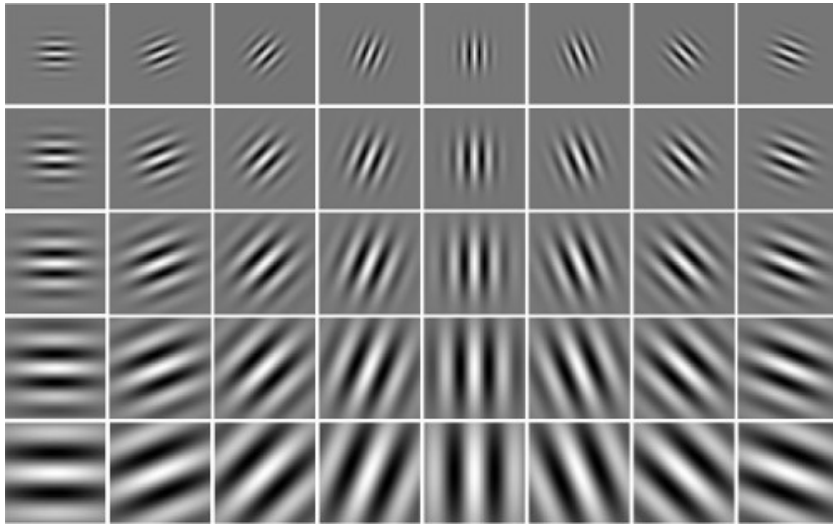
$$\frac{\partial E}{\partial \omega_{a,b}} = \sum_i \sum_j \frac{\partial E}{\partial y_{i,j}^l} \sigma'(x_{i,j}^l) y_{i,j}^{l-1}$$

# Convolution

Backward, erreur

$$\begin{aligned}\frac{\partial E}{\partial y_{i,j}^{l-1}} &= \sum_a \sum_b \frac{\partial E}{\partial x_{i-a,j-b}^l} \frac{\partial x_{i-a,j-b}^l}{\partial y_{i,j}^{l-1}} \\ &= \sum_a \sum_b \frac{\partial E}{\partial x_{i-a,j-b}^l} \omega_{a,b}\end{aligned}$$

# Couches : convolution



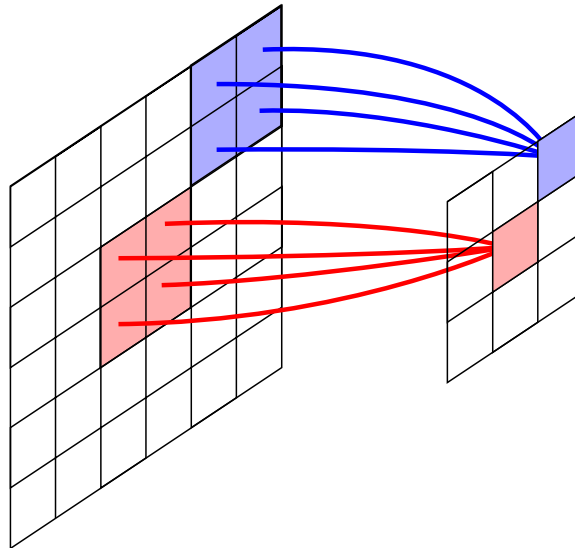
Filtres de Gabor

Exemple poids appris  
par la première couche  
de convolution  
d'AlexNet

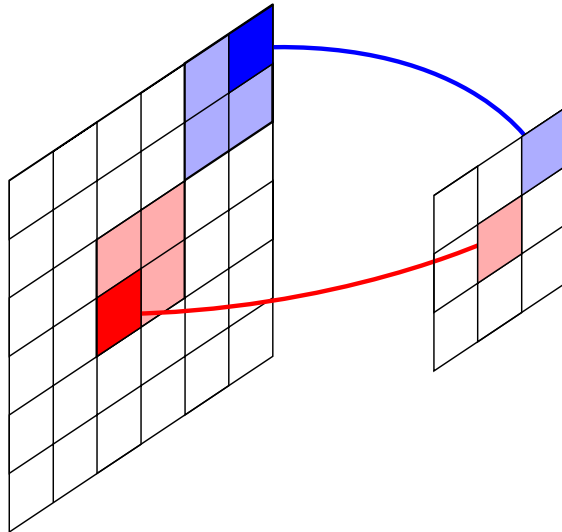


# Couches : Pooling

- Réduction de dimension
- Invariabilité en translation
- Mise en correspondance des pixels voisins
  - Réduction de la profondeur

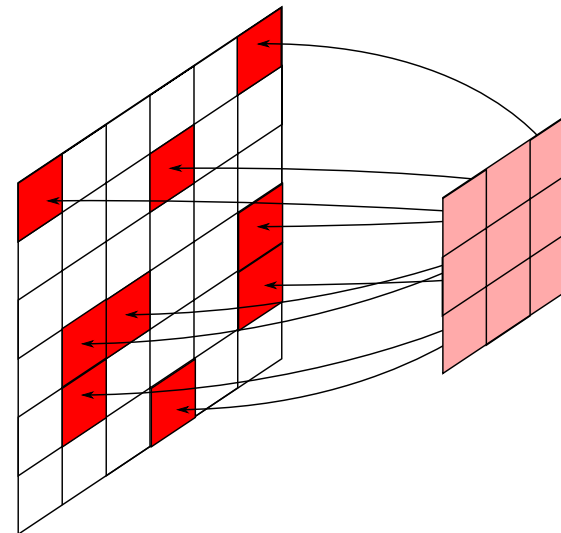


# Couches : Max Pooling



## Forward

Transmission du maximum sur une fenêtre donnée

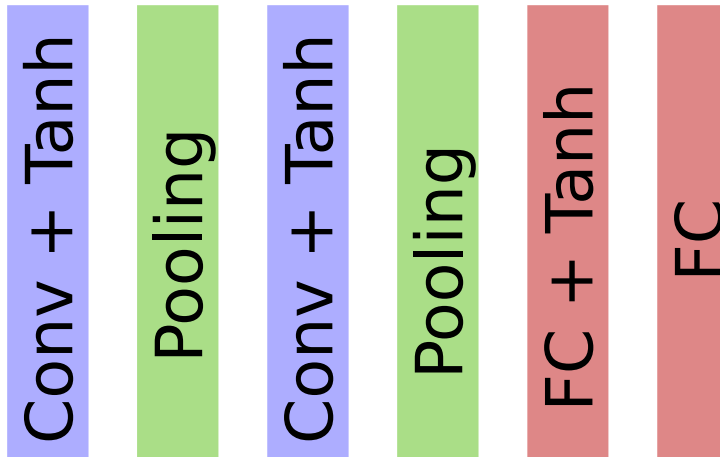


## Backward

Transmission du gradient au maximum identifié, gradient nul sinon

# LeNet

LeNet (1990)  
Images 28x28



# Principaux problèmes

Vitesse d'apprentissage

Gradients exponentiels ou évanescents

Structure du réseau

Matériel

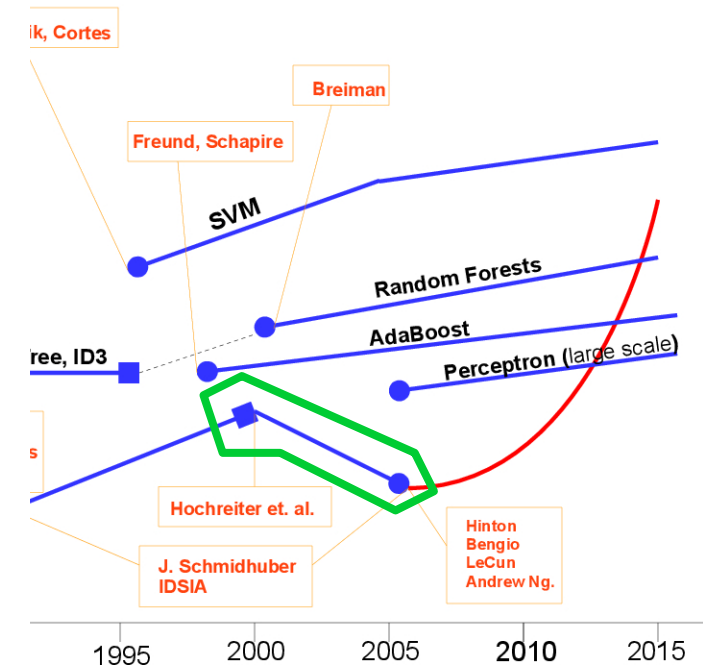
Optimiseur

Données

Initialisation des poids

Overfitting

Minima locaux





# Deep learning : massively data driven approaches



Calcul parallèle



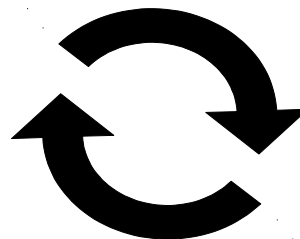
Moyens d'acquisition



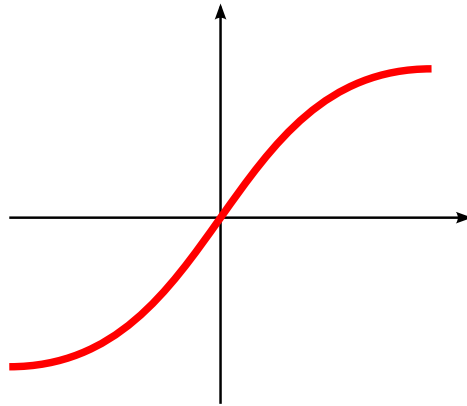
Partage

Performances

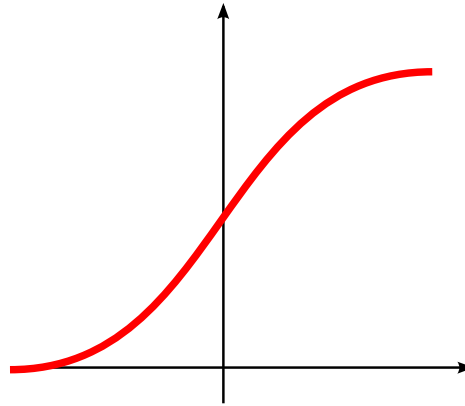
Intérêt



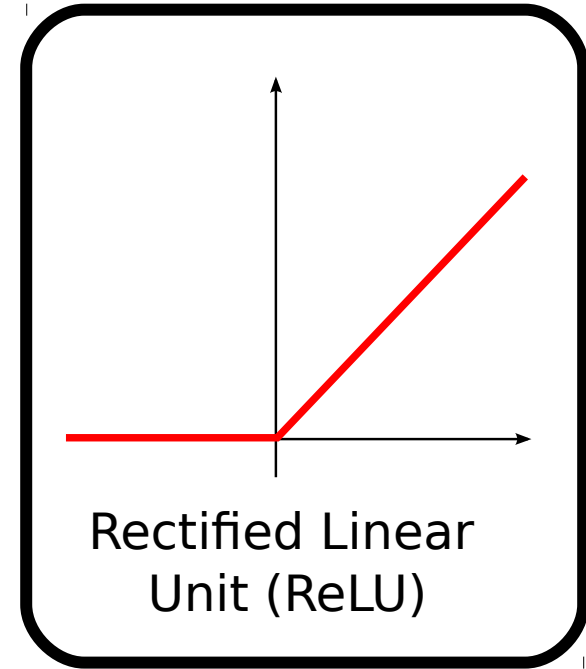
# Activations



Tangente  
hyperbolique



Sigmoïde

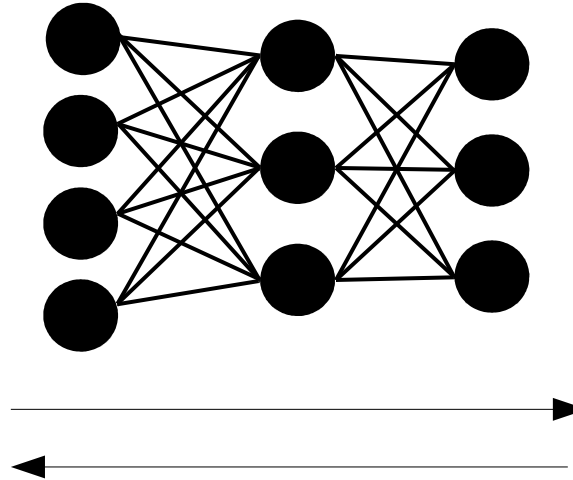


Rectified Linear  
Unit (ReLU)

Gradients plus  
rapide à calculer

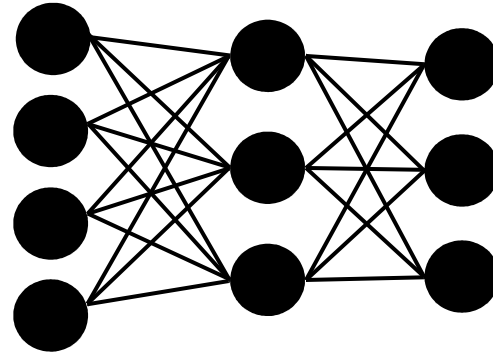
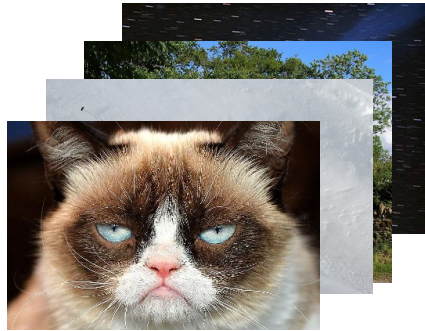
Convergence  
identique

# Mini-batch

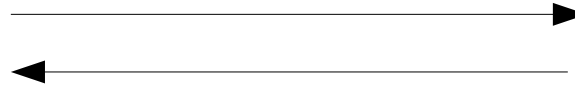


Erreur image 1

# Mini-batch

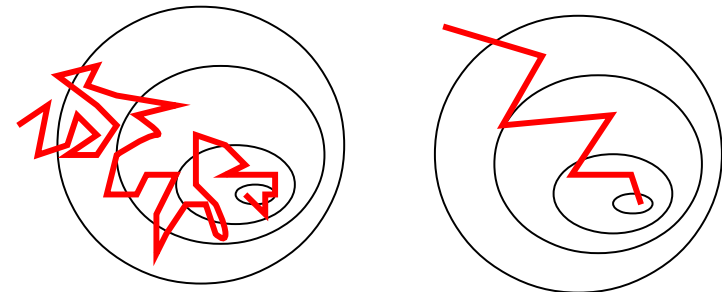


Erreur image 1  
Erreur image 2  
Erreur image 3  
Erreur image 4  
...

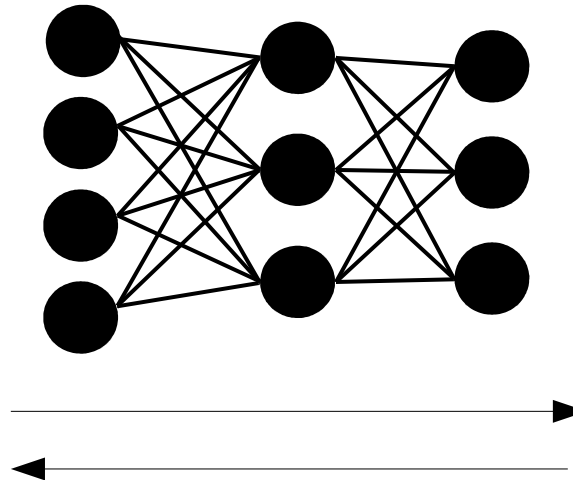
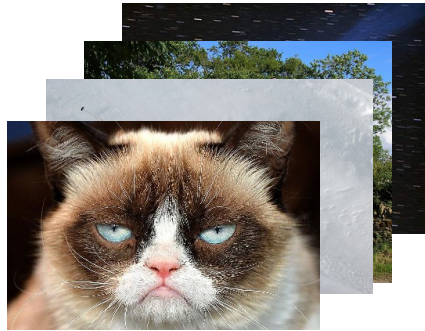


- Forward
  - Prédictions indépendantes
- Backward
  - Moyennes sur les gradients

- **Lissage des gradients**
  - Réduction de la variance
  - Possibilité d'utilisation d'un learning rate plus grand



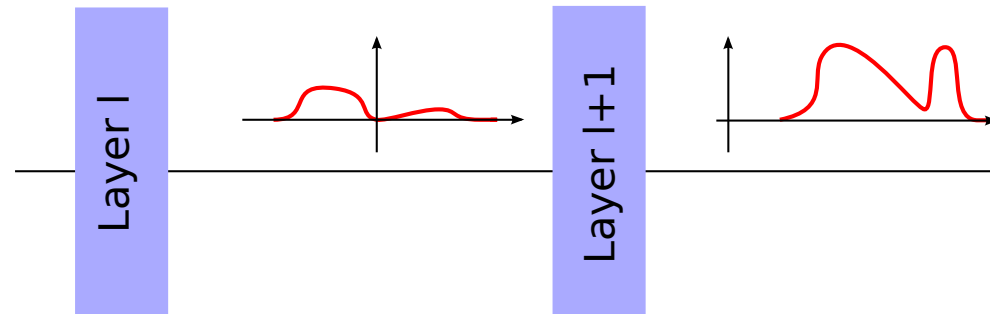
# Mini-batch



Erreur image 1  
Erreur image 2  
Erreur image 3  
Erreur image 4  
...

- **Forward**
  - Prédictions indépendantes
- **Backward**
  - Moyennes sur les gradients
- **Lissage des gradients**
  - Réduction de la variance
  - Possibilité d'utilisation d'un learning rate plus grand
- **Accélération matérielle (GPU)**
  - Appliquer les même opérations

# Couches : Batch Norm



Changements dans la distribution  $\Rightarrow$  variance et moyenne  
Apprentissage plus difficile

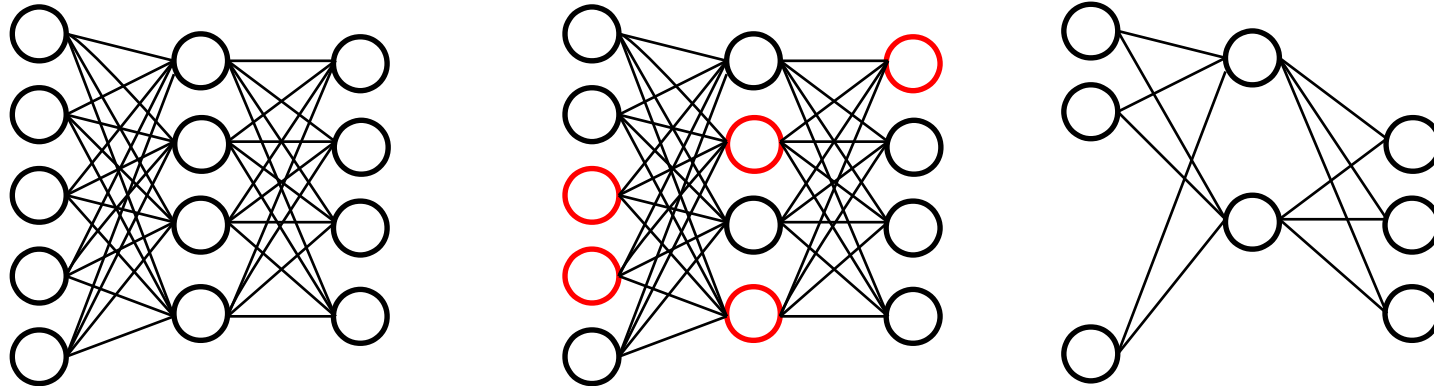
Contrôler la distribution des données avec une couche pour maîtriser la distribution

$$y^{*l} = \frac{y^l - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \beta$$

Les  $\beta$  et  $\gamma$  sont appris.  
 $\mu$  et  $\sigma$  sont calculés sur les batches.

Apprentissage plus rapide (en nombre d'itération), mais  
en général plus lent (calcul des statistiques)

# Dropout



## Entraînement :

- Éteindre certains neurones avec probabilité  $1-p$   
⇒ Entraîner une multitude de sous-réseaux

## Test :

- Utiliser tous les neurones pondérés par  $p$

Les fully connected concentrent les poids des réseaux  $\Rightarrow$  overfitting  
Le dropout apporte plus de robustesse et une meilleure généralisation

Moins de nœuds  $\Rightarrow$  plus rapide

# Initialisation des poids

Les poids des réseaux sont initialisés aléatoirement.  
L'initialisation a une grande influence sur la convergence et sur la vitesse d'apprentissage.

- Poids trop faibles  $\Rightarrow$  le signal peut décroître jusqu'à s'annuler
- Poids trop grands  $\Rightarrow$  le signal peut devenir trop gros pour être utilisable

## Exemple : Initialisation Xavier

$X$  de dimension  $n$ , un neurone linéaire avec poids  $W$ , la sortie est  $Y$

$$Y = W_1 X_1 + W_2 X_2 + \dots + W_n X_n$$

Or

$$\text{Var}(W_i X_i) = E[X_i]^2 \text{Var}(W_i) + \text{Var}(W_i) \text{Var}(X_i) + \text{Var}(X_i) E[W_i]^2 = \text{Var}(W_i) \text{Var}(X_i)$$

Ainsi  $\text{Var}(Y) = n \text{Var}(W_i) \text{Var}(X_i)$  et  $\text{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{in}}$

Glorot et Bengio () :

idem avec la back prop.  $\text{Var}(W_i) = \frac{1}{n_{out}}$  Moyenne des deux  $\text{Var}(W_i) = \frac{1}{n_{in} + n_{out}}$

He, Rang, Zen et Sun () :  $\text{Var}(W_i) = \frac{2}{n_{in}}$  (un ReLU aura la moitié de ses activations nulles)



- **Variation du learning rate**
  - Constant par morceau
  - Décroissance exponentielle
- **Lissage du gradient**
  - Momentum

$$\mathbf{w} = \mathbf{w} - \alpha * \Delta \mathbf{w}$$

$$\begin{aligned} \mathbf{v}_t &= \gamma \mathbf{v}_{t-1} + \alpha * \Delta \mathbf{w} \\ \mathbf{w} &= \mathbf{w} - \mathbf{v}_t \end{aligned}$$

- **Méthodes adaptatives**

- Adagrad : adaptation du learning rate à la fréquence

$$\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} \Delta \mathbf{w}_{t,i}$$

la somme des  
gradients au carré pour  
 $G_{t,i}$

$\mathbf{w}_{t,i}$

- Adam (adaptive momentum estimation)

Adaptation du learning rate et utilisation du momentum

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \Delta \mathbf{w}_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) \Delta \mathbf{w}_t^2 \end{aligned}$$

Estimation de la  
moyenne et de la  
variance

$$\begin{aligned} \hat{m}_t &= m_t / (1 - \beta_1^t) \\ \hat{v}_t &= v_t / (1 - \beta_2^t) \end{aligned}$$

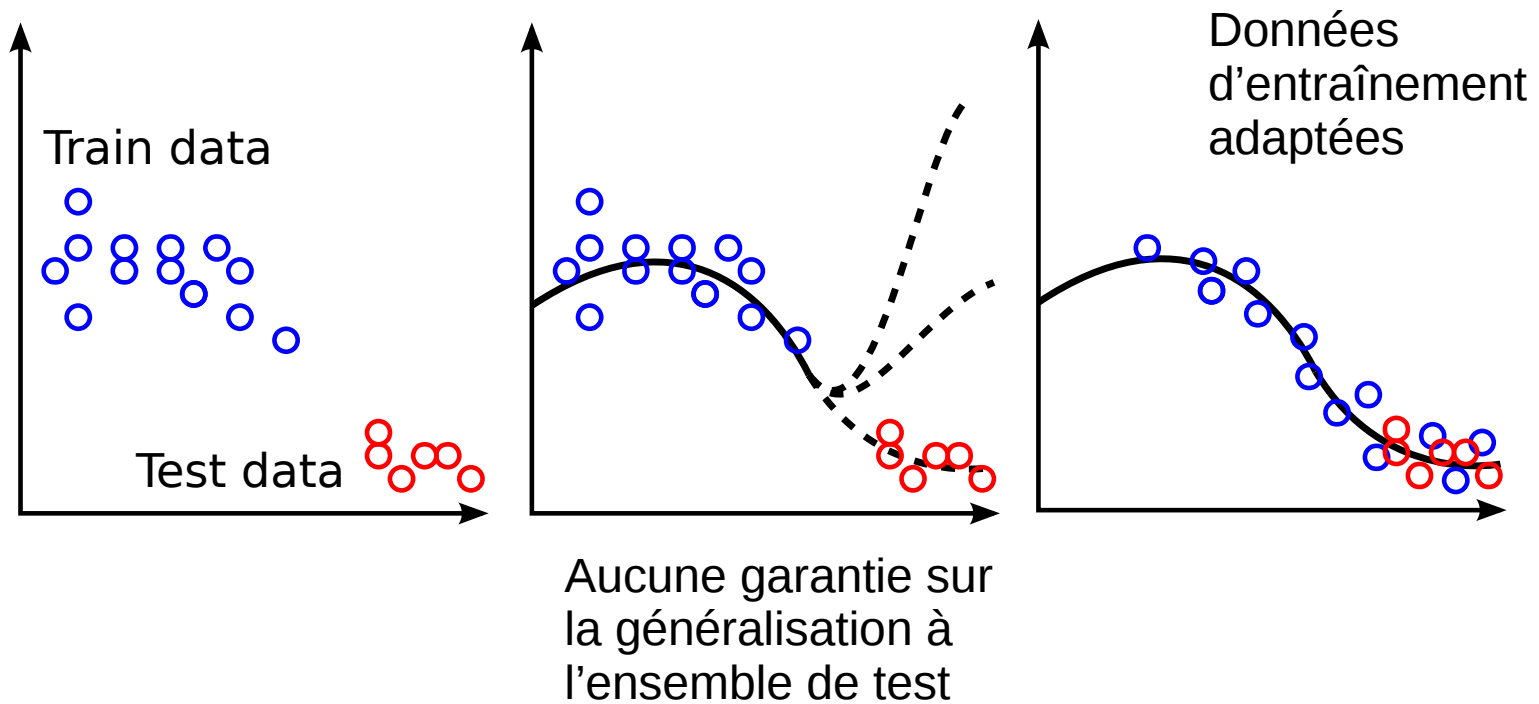
Moments biaisés  
vers 0.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Règle de mise à jour

# Les données

- Données adaptées



# Les données

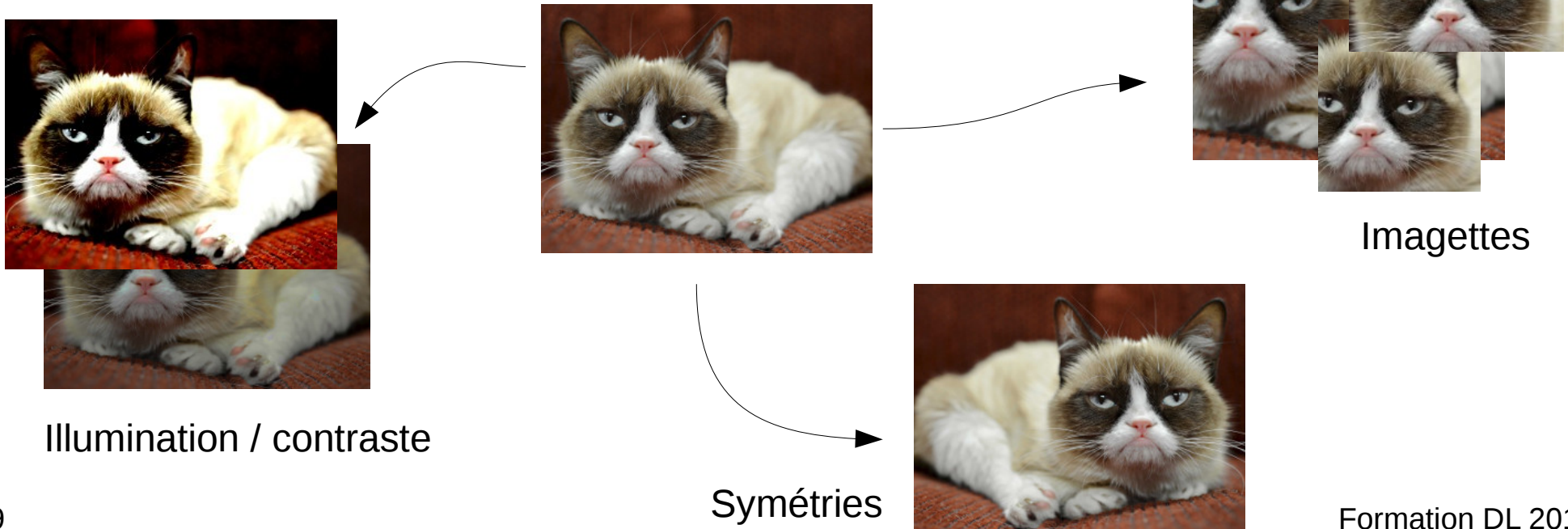
- Données adaptées
- Normalisation

Moyenne nulle, variance 1

⇒ Éviter les problème de dynamique

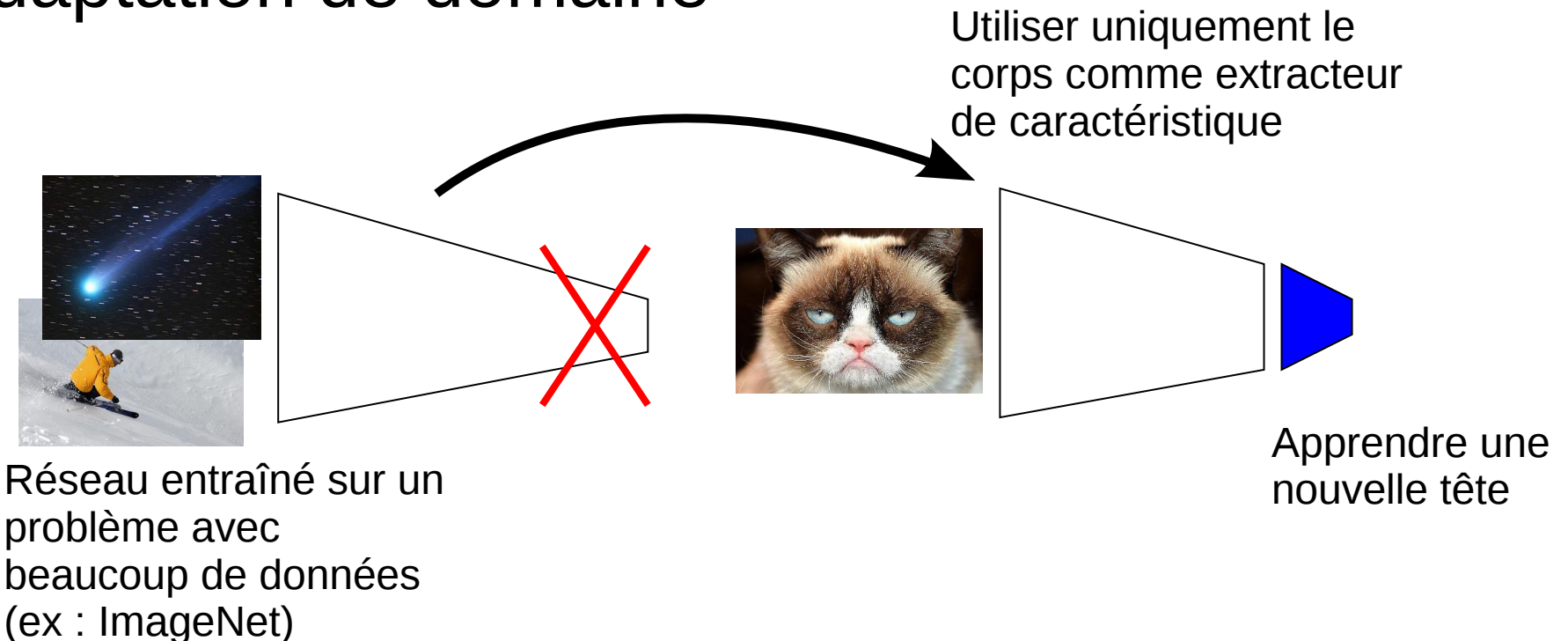
# Les données

- Données adaptées
- Normalisation
- Augmentation de données
  - Faire varier les paramètres qu'on ne veut pas apprendre.



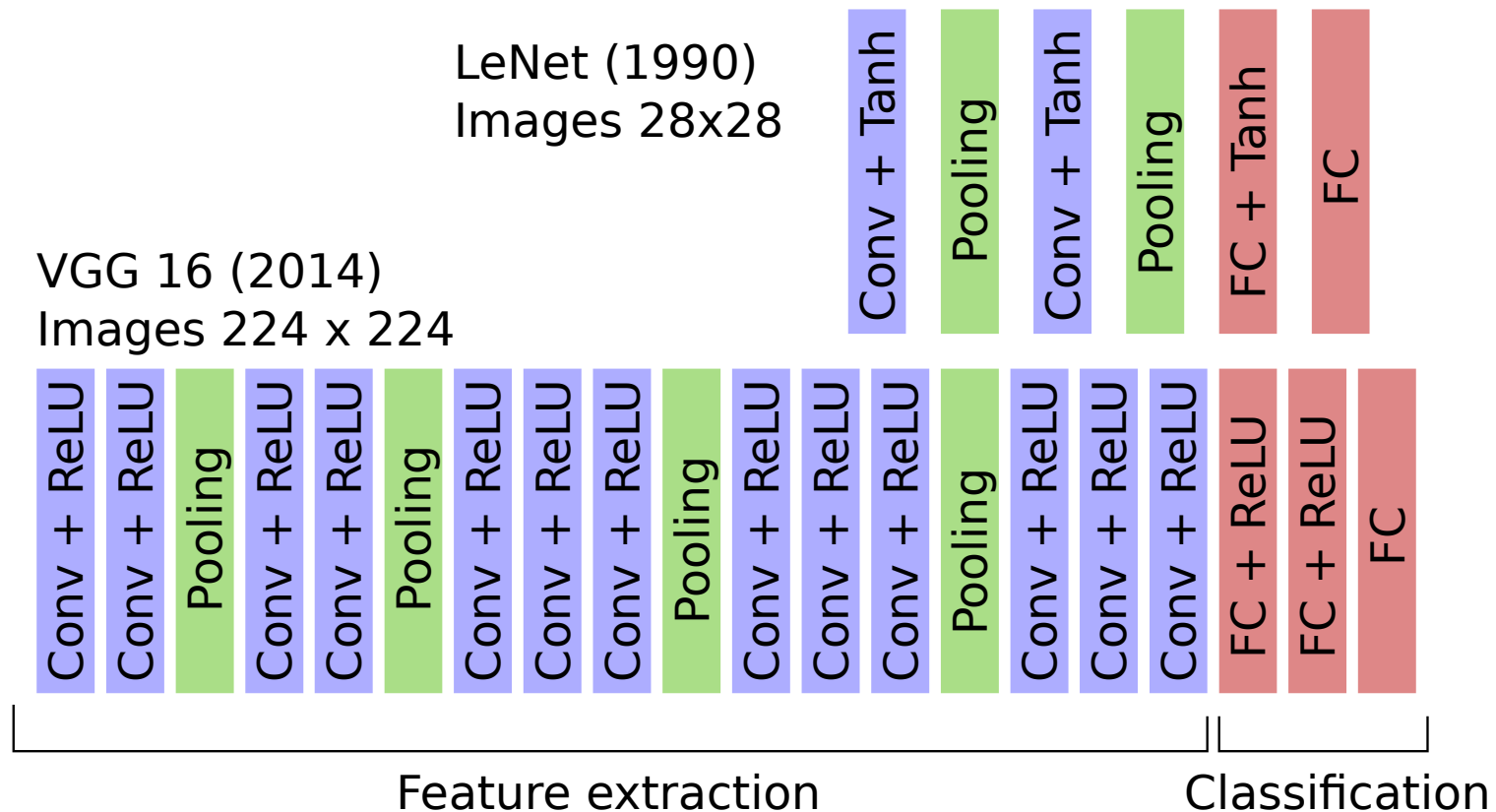
# Les données

- Données adaptées
- Normalisation
- Augmentation de données
- Adaptation de domaine

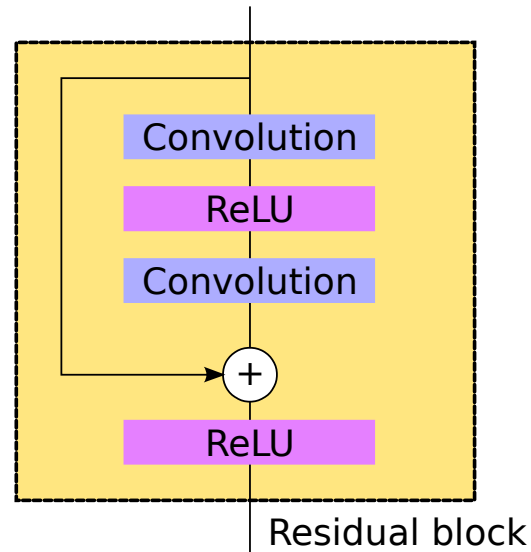


# MNIST / VGG 16

- Architecture classique en vision



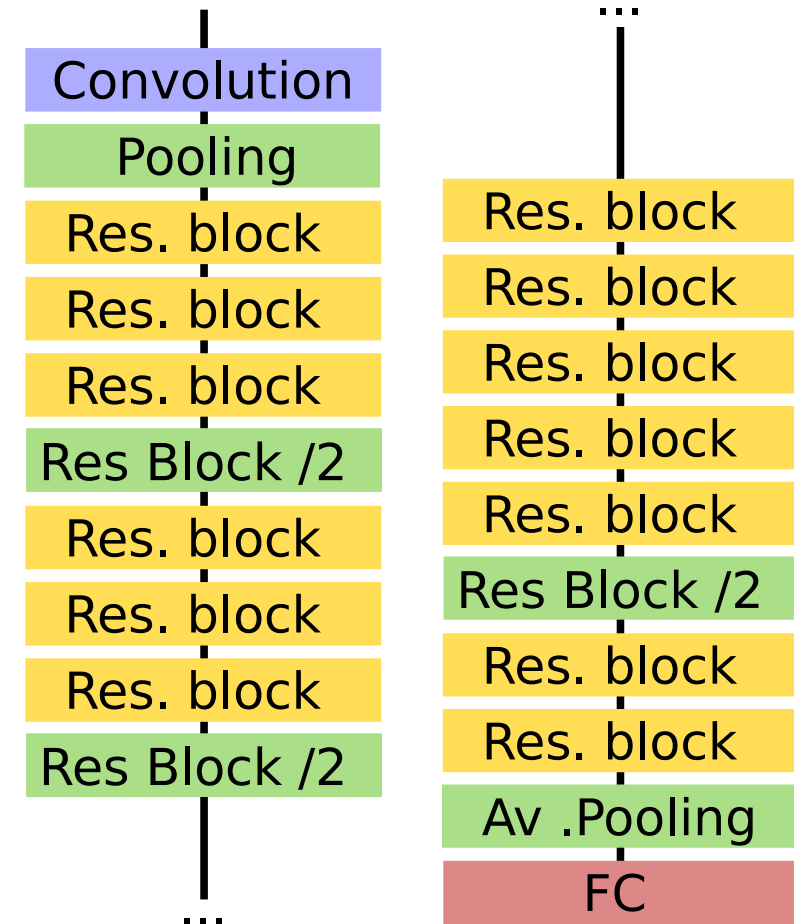
# ResNet



Court circuit sur les convolutions.

⇒ impose aux nouvelles convolution d'apprendre quelque chose de différent de la couche précédente

⇒ Meilleure circulation des gradients. Possibilité de réseaux plus profonds.





- Segmentation sémantique
  - Le unpooling layer pour retrouver la localité en mémorisant les emplacement des activations

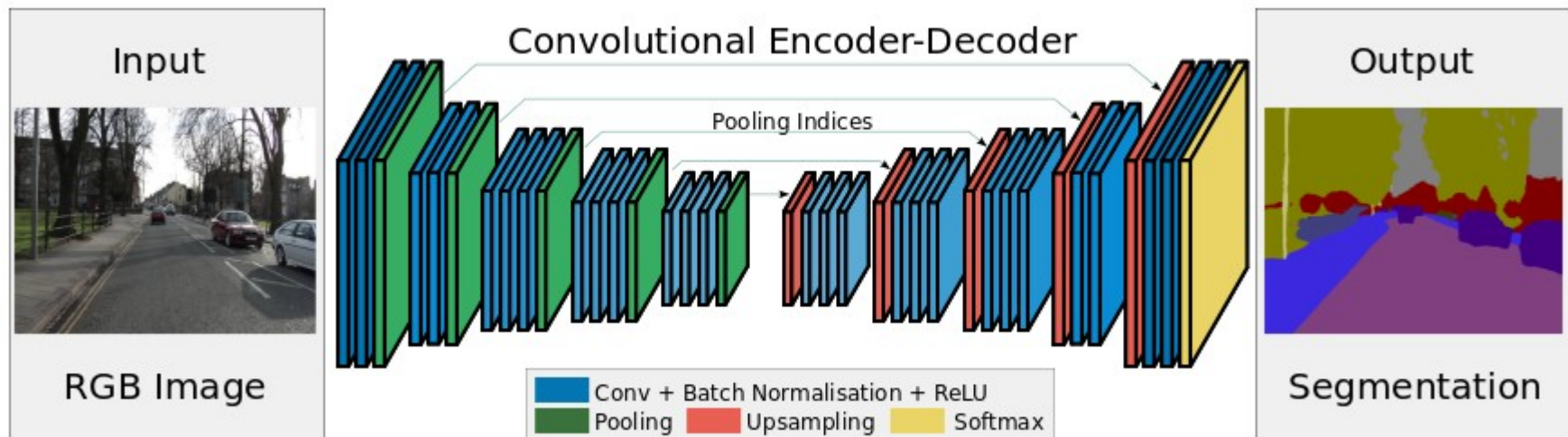


Image : <http://mi.eng.cam.ac.uk/projects/segnet/>

# FasterRCNN

- Détection d'objets

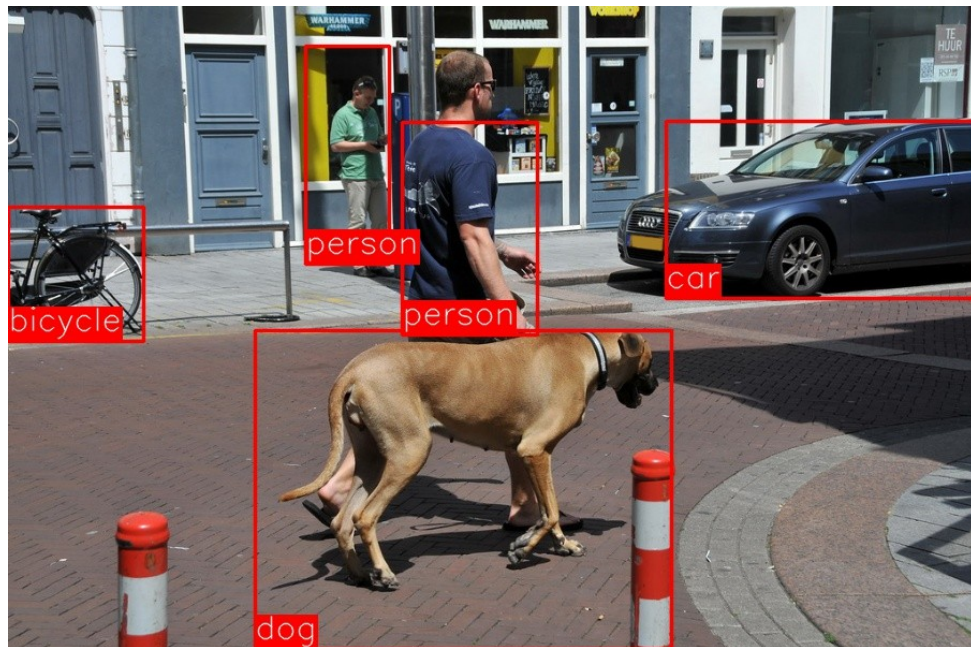


Image : <https://github.com/mitmul/chainer-fast-rcnn>

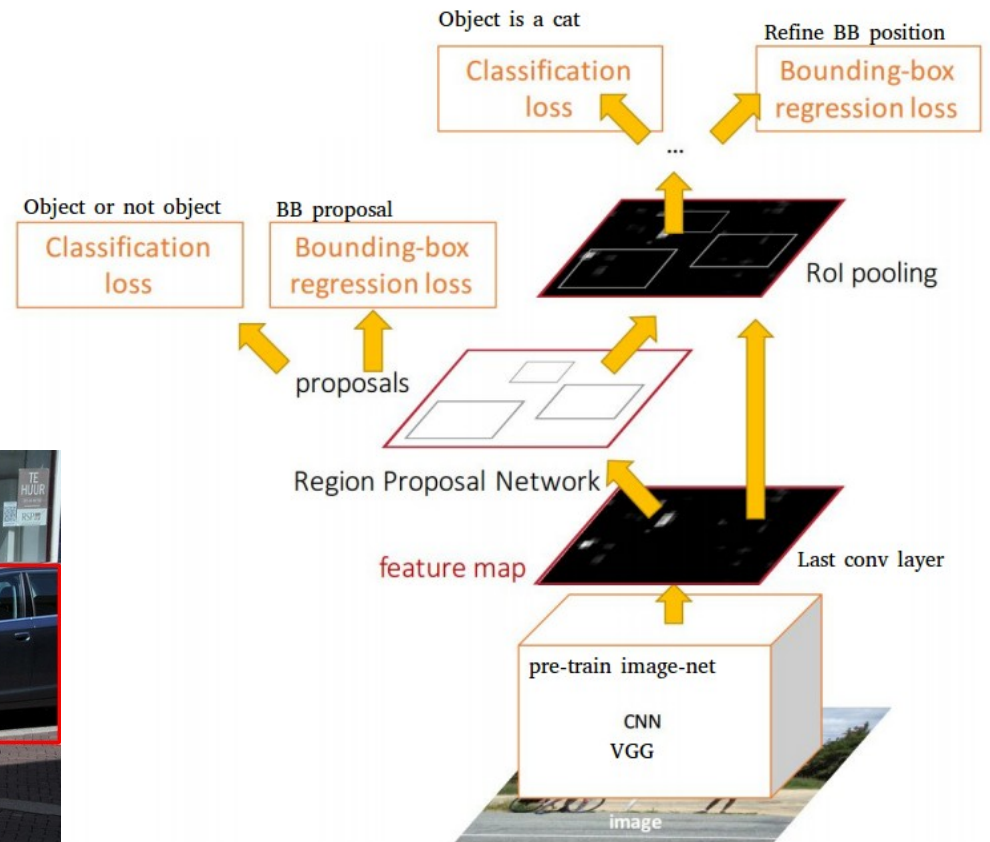
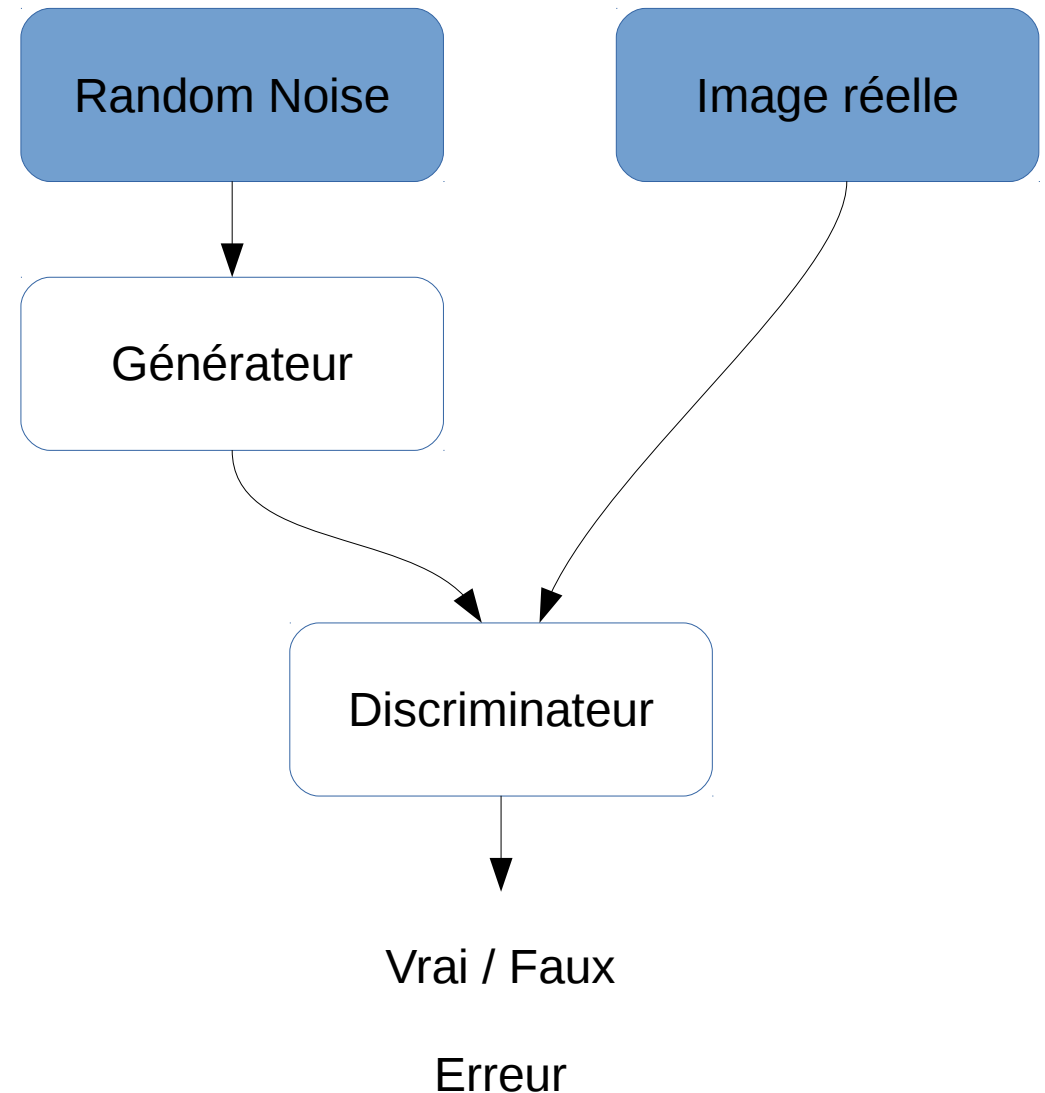
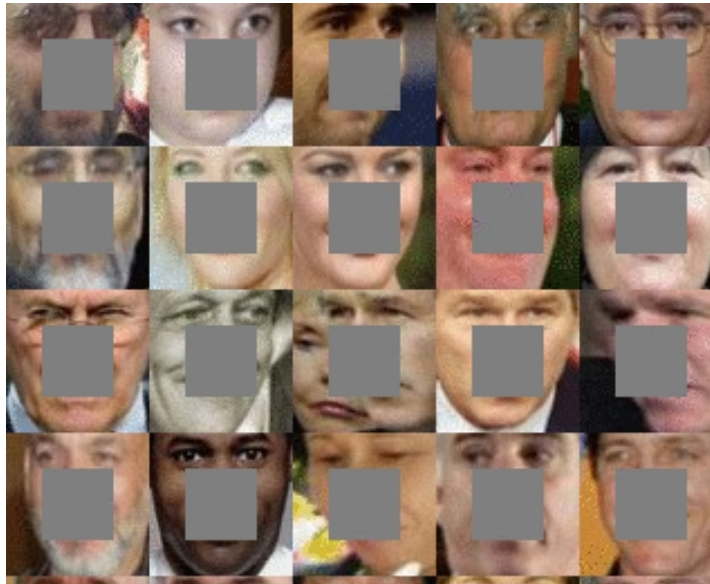
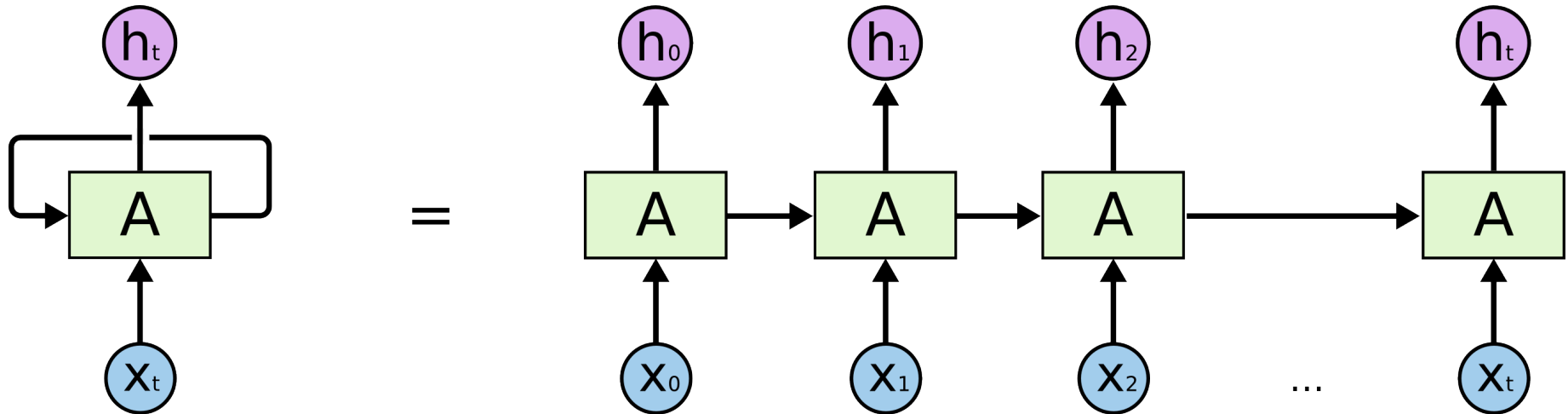


Image :  
[https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/object\\_localization\\_and\\_detection.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/object_localization_and_detection.html)

# Generative Adversarial networks



# Réseaux récurrents



Traitement de données temporelles :  
La sortie de l'état  $T+1$  dépend de  $T$

Analyse de textes, de vidéos, captioning (description d'images avec du texte) ...

- 

Formation DL 2017



# Frameworks



Interface principale :  
**Python**

Autres interfaces :  
C++  
Java, Go (début)

GPU Nvidia ou CPU

Google, Deep Mind



Interface principale :  
**Python, Lua**

Autres interfaces :  
C++

GPU Nvidia ou CPU

Facebook AI, Inria,  
ENS Ulm

Caffe

Interface principale :  
**C++**

Autres interfaces :  
Python

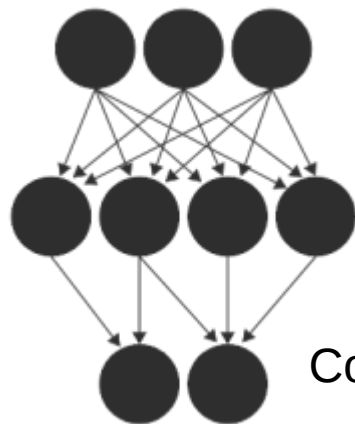
GPU Nvidia ou CPU

Académique

# Frameworks

MatConvNet

*dmlc*  
***mxnet***



ConvNetJs



Keras

Microsoft  
CNTK

theano

DEEPLEARNING4J

# Exemple : LeNet

## Définition du réseau

```
conv1 = conv2d(image, 64 , [3,3] , [1,1])
relu1 = relu(conv1)
pool1 = max_pool_2x2(relu1)
conv2 = conv2d(pool1, 128, [3,3] , [1,1])
relu2 = relu(conv2)
pool2 = max_pool_2x2(relu2)
flat   = flatten(pool2)
fc1    = fully_connected(flat, 256)
reluf1 = relu(fc1)
fc2    = fully_connected(reluf1, 10)
```

## Fonction de perte et optimiseur

```
loss =softmax_cross_entropy(fc2,labels)
optimizer = AdamOptimizer(learning_rate)
train_step = optimizer.minimize(loss)
```



# Exemple : LeNet

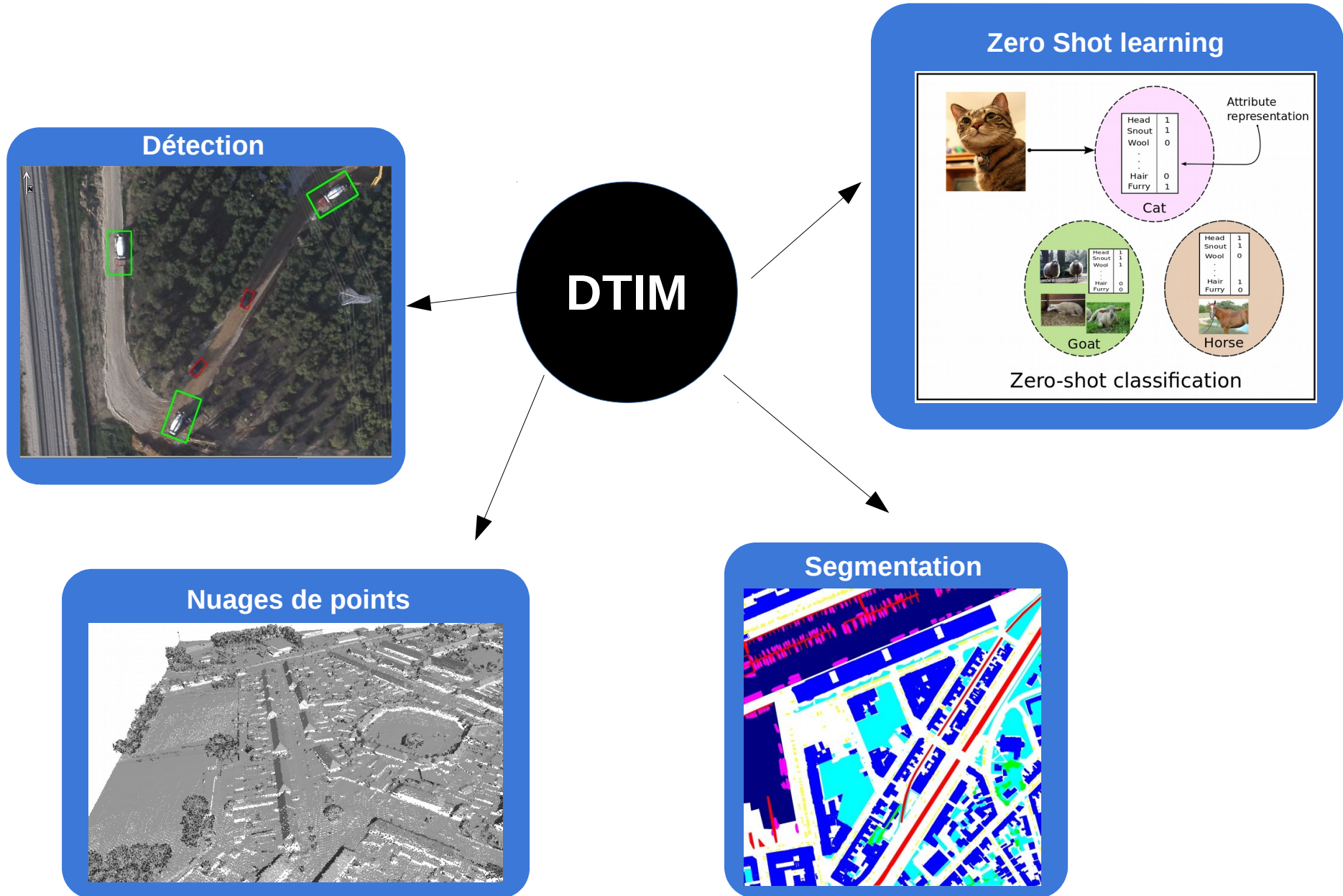
## Entraînement

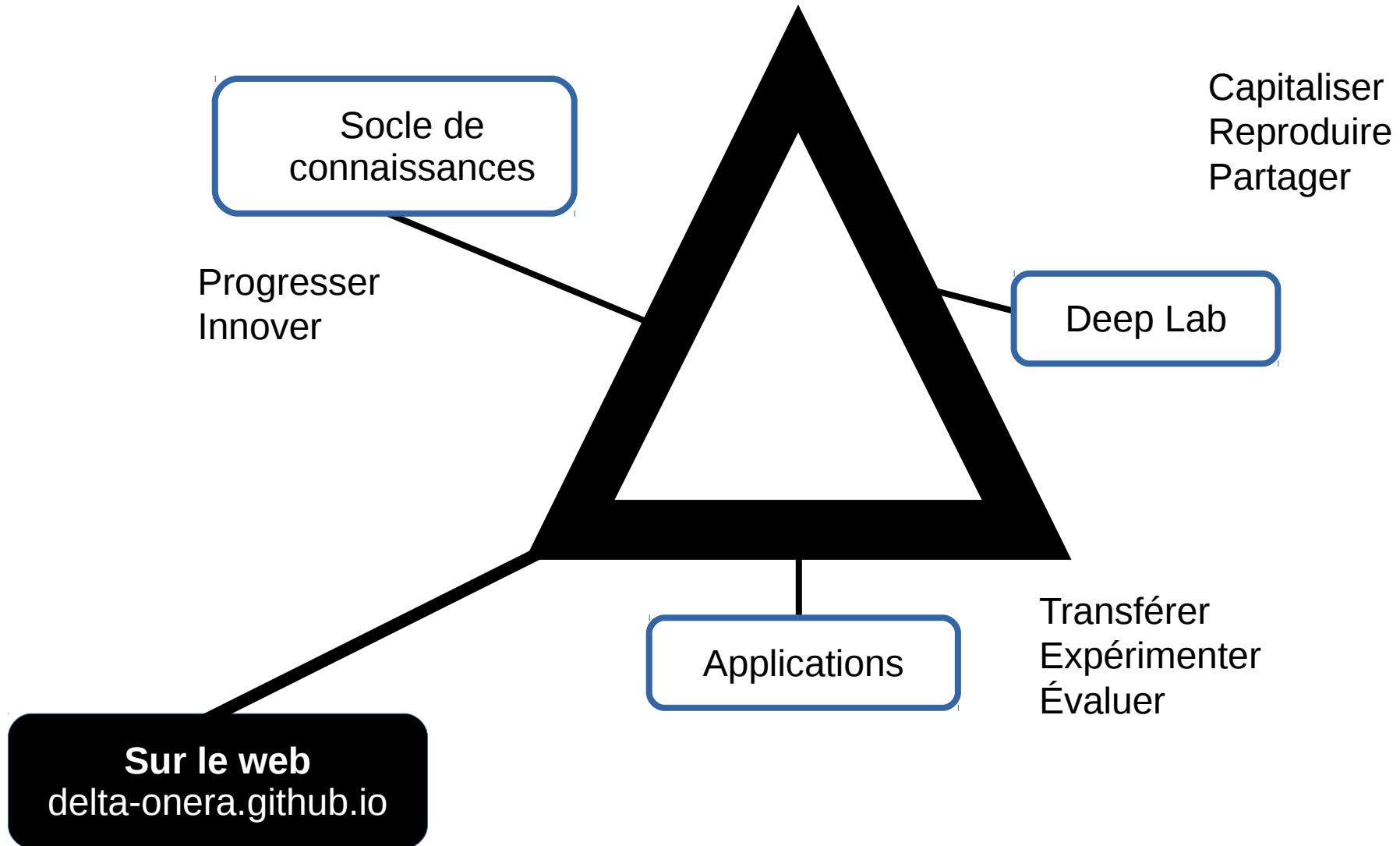
```
for epoch in range(epoch_max) :  
  
    for batch in batches :  
        # numpy tabs  
        fd = {images:batch.images, labels:batch.labels}  
        sess.run(train_step, feed_dict=fd)
```

## Test

```
for test_images in test_set :  
    fd = {images:test_images}  
    predictions = sess.run(fc2, fd)
```

# Le Deep learning à l'ONERA





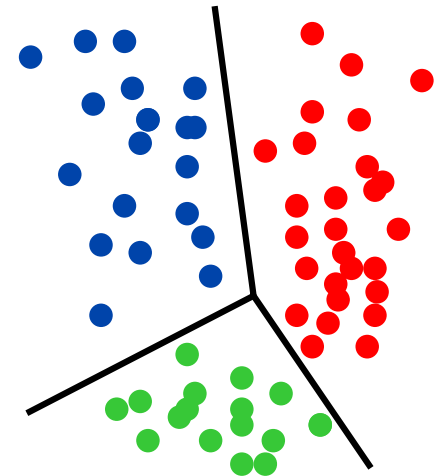
# Conclusion



Données

Réseaux de  
neurones profonds

Recherche sur les  
architectures, les stratégies,  
la mise en forme des données



Objectif

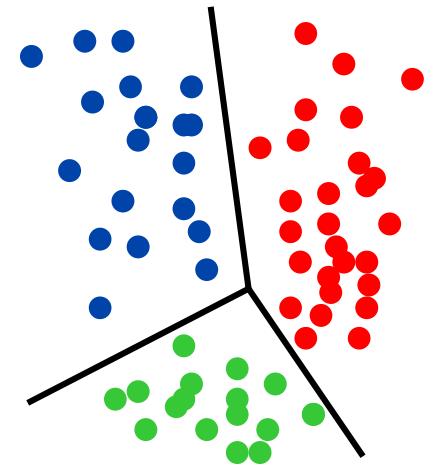
# Conclusion



Données

Réseaux de  
neurones profonds

Recherche sur les  
architectures, les stratégies,  
la mise en forme des données



Objectif

# Ressources

**Gradient-based learning applied to document recognition**, LeCun, Bottou, Bengio and Haffner, 1998

**Batch normalization: Accelerating deep network training by reducing internal covariate shift**, Ioffe and Szegedy, 2015

**Dropout: a simple way to prevent neural networks from overfitting**, Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov, 2014

**Understanding the difficulty of training deep feedforward neural networks**, Glorot and Bengio

**Adaptive Subgradient Methods for Online Learning and Stochastic Optimization**, Duchi, Hazan and Singer, 2011

**Adam: a Method for Stochastic Optimization**, Kingma and Ba, 2015

**Very Deep Convolutional Networks for Large-Scale Image Recognition**, Simonyan and Zisserman

**Deep residual learning for image recognition**, He, Zhang, Ren, and Sun, 2016

**Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding**, Kendall, Badrinarayanan and Cipolla, 2015

**Faster r-cnn: Towards real-time object detection with region proposal networks**, Ren, He, Girshick and Sun, 2015

**Improving semantic embedding consistency by metric learning for zero-shot classification**, Bucher and Herbin and Jurie, 2016

**Processing of Extremely High-Resolution LiDAR and RGB Data: Outcome of the 2015 IEEE GRSS Data Fusion Contest--Part A: 2-D Contest**, multiple authors, 2016

**Semantic Image Inpainting with Perceptual and Contextual Losses**, Yeh and al

# Ressources

## Principaux frameworks

Tensorflow : [tensorflow.org](https://tensorflow.org)

PyTorch et Torch : [pytorch.org](https://pytorch.org) [torch.ch](https://torch.ch)

Caffe : [caffe.berkeleyvision.org/](https://caffe.berkeleyvision.org/)

## Tutoriaux

Classification de chiffres sur MNIST :

[tensorflow.org/tutorials/mnist/pros/](https://tensorflow.org/tutorials/mnist/pros/)

Classification d'images sur CIFAR 10

<https://github.com/pytorch/tutorials>