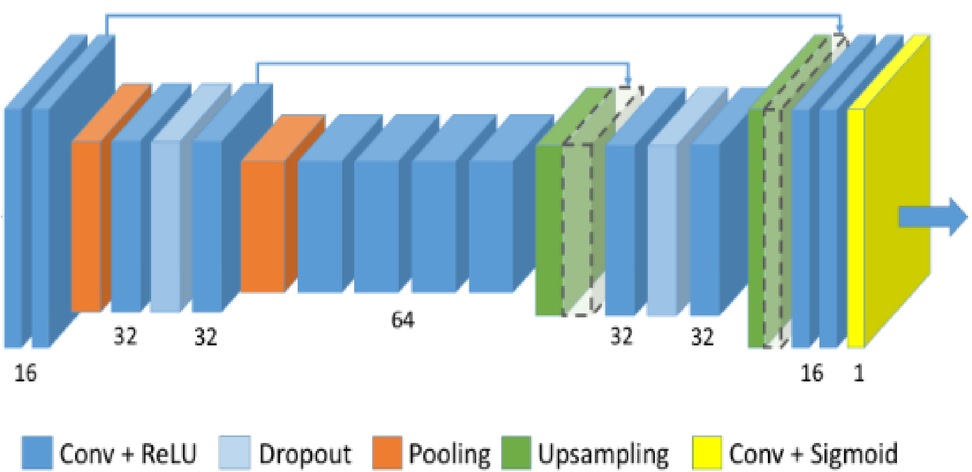
Neural-network Topology Bayesian Optimization for Hardware Implementation

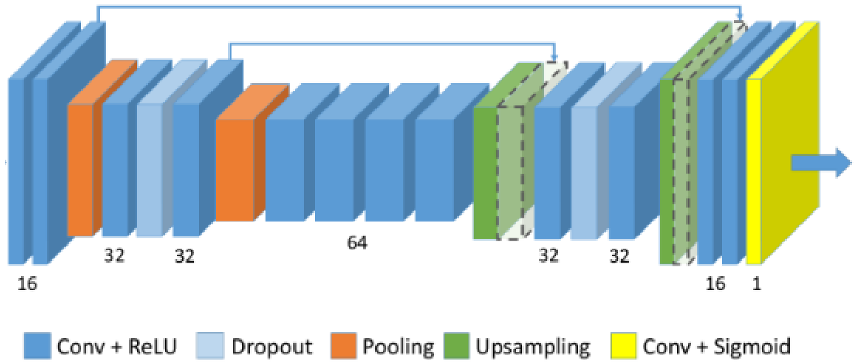


Laboratoire Leprince-Ringuet

Frédéric Magniette
Think Project Kickoff

THINK

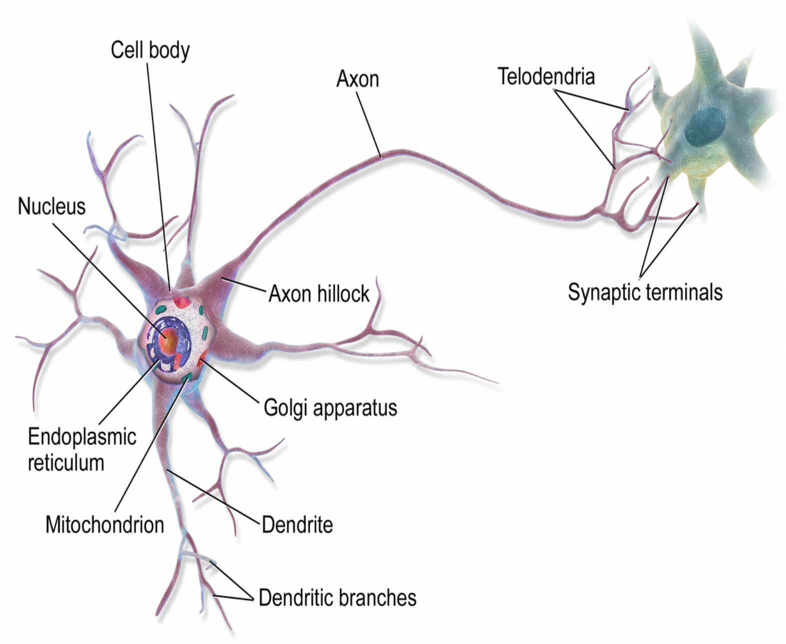
Introduction

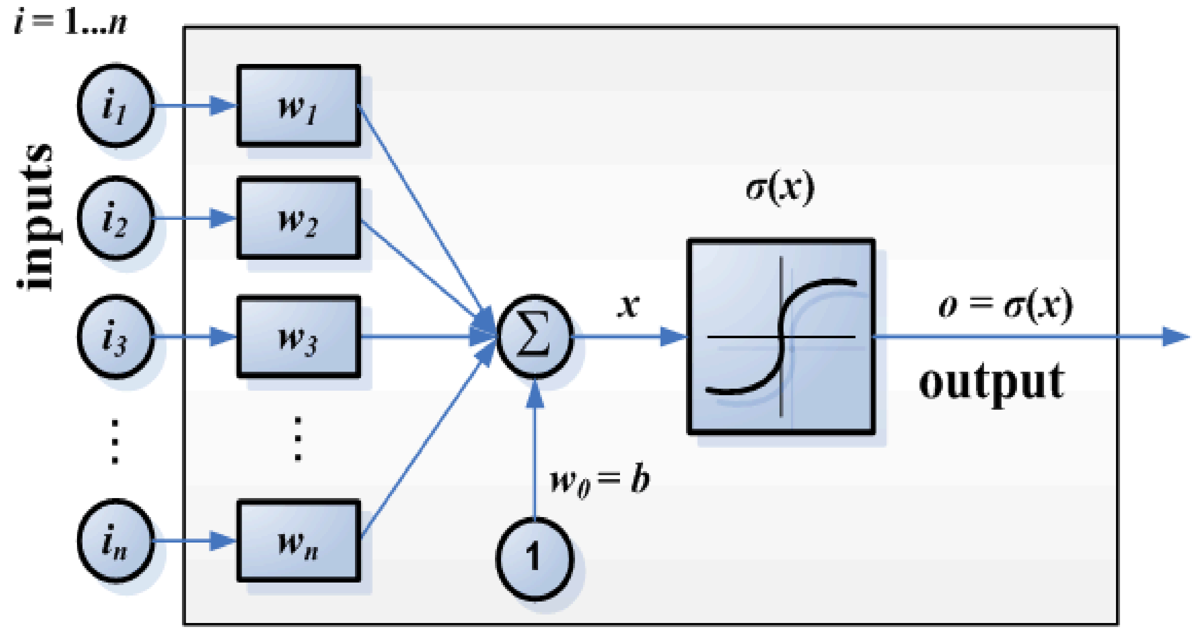


- Hardware implementations
  - Synthetizable models
  - Ressource management
- Need to optimize the network topology
  - Prior implementation
  - With respect to precision
- Two approaches
  - Pruning: reducing size of trained network
  - Optimization: find & train an optimal sized network at the same time

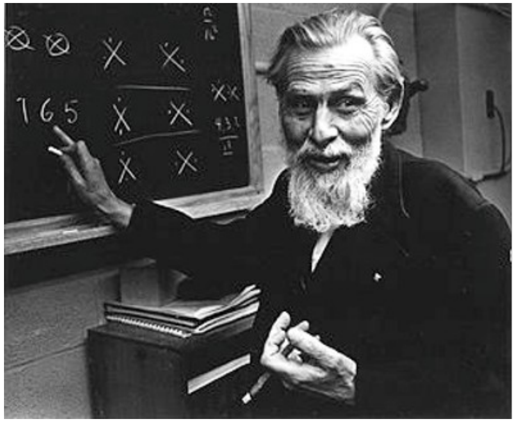
Neural networks

From real to formal neuron

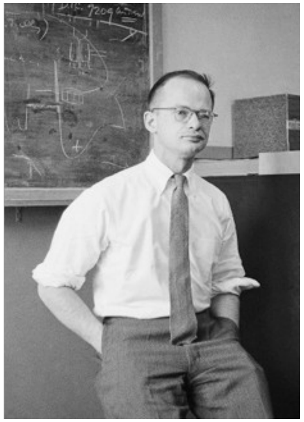




McCullock & Pitts – 1943
A logical calculus of the ideas immanent in nervous activity

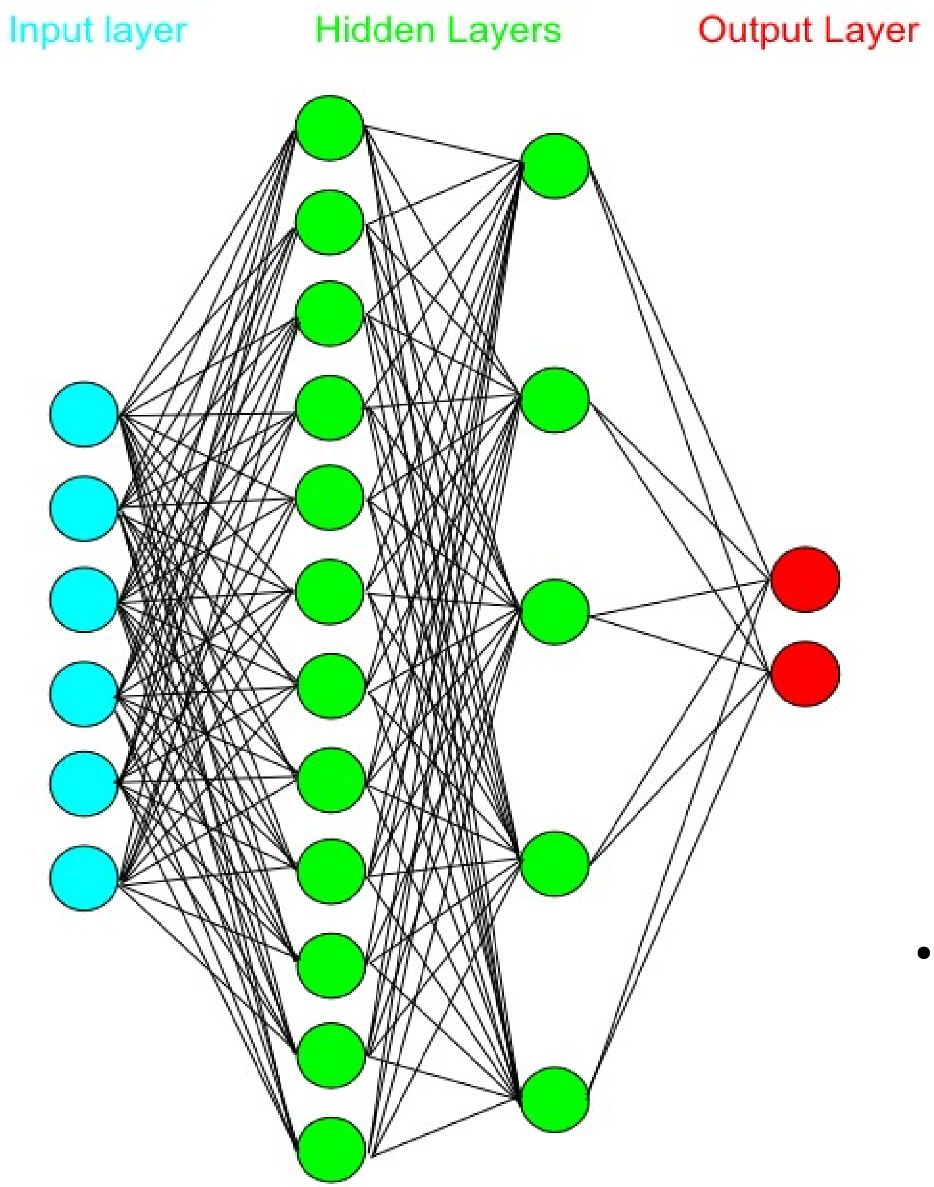


Mc Cullock



Pitts

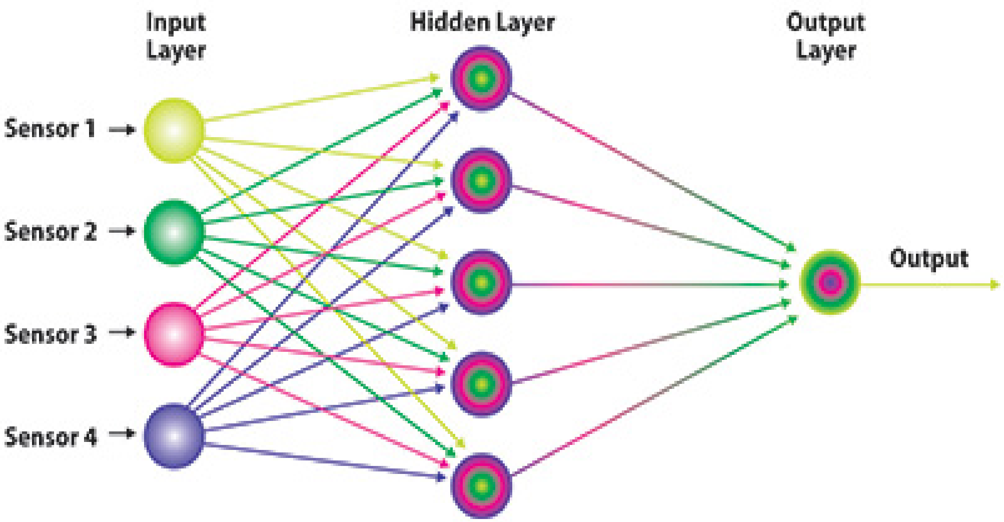
First artificial neural network



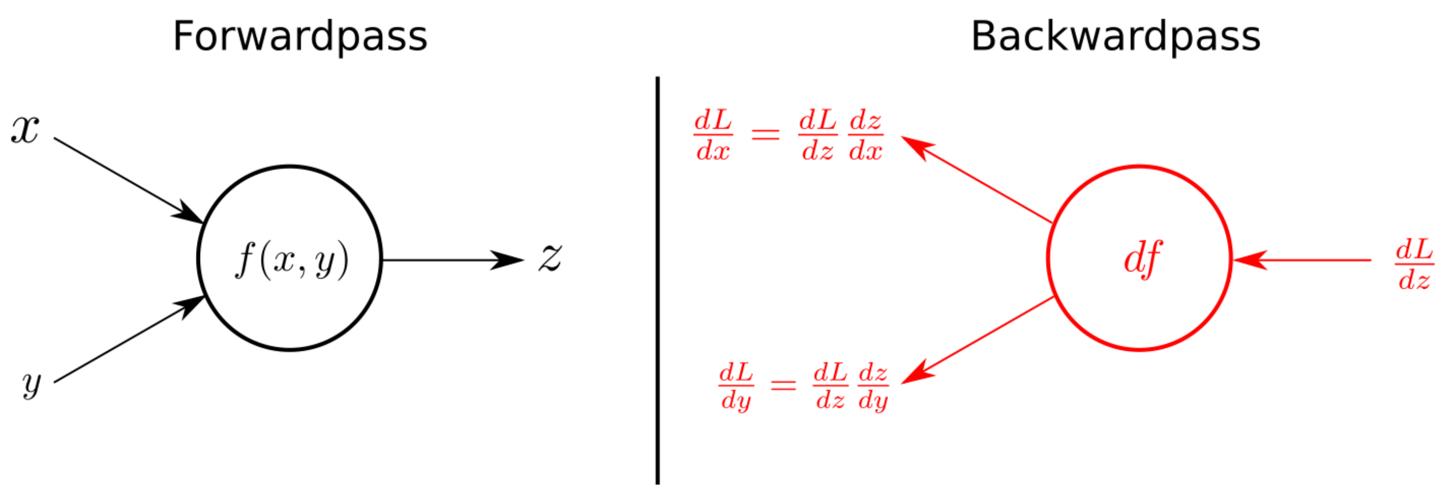
- Multi-layer perceptron
- Fully connected layers
- Deep Network : >6 layers

- Paul Werbos – PhD - 1974
  - multi-layer perceptron
  - gradient retro-propagation algorithm

How does it work

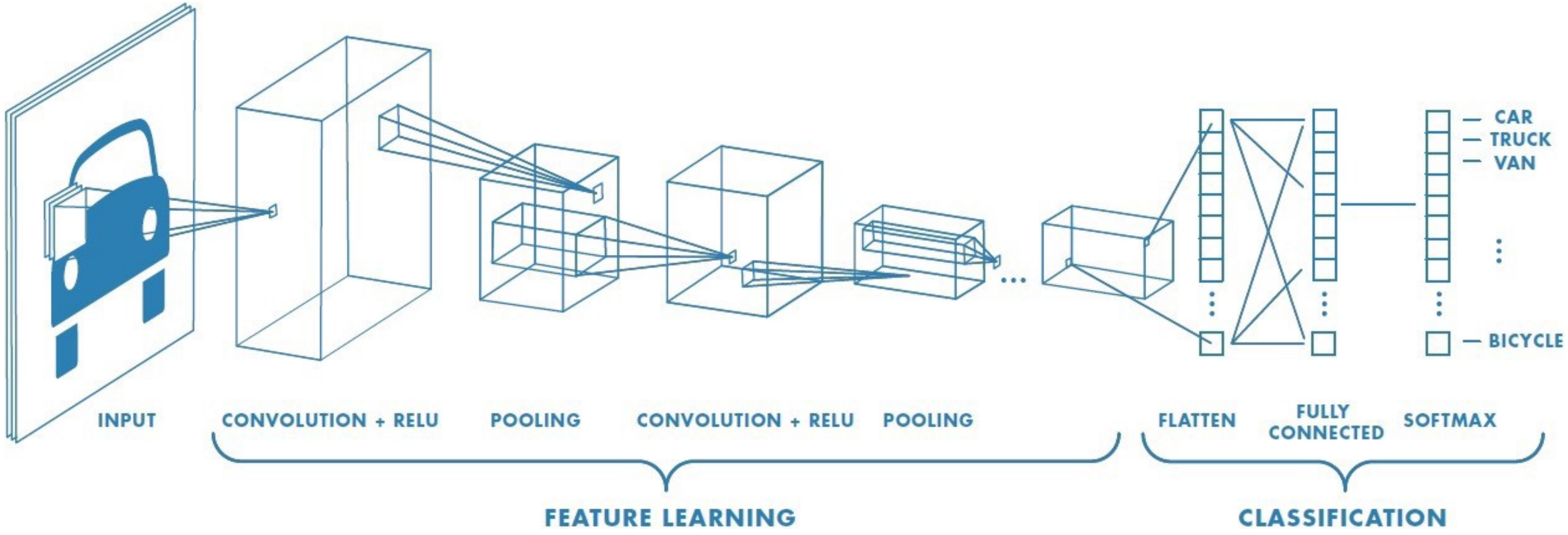


- Transfer function gives non-linearity
- Number of params gives precision
- Similar to power series



Training by back-propagation of error derivative

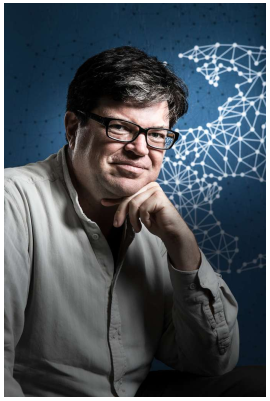
Convolutional network



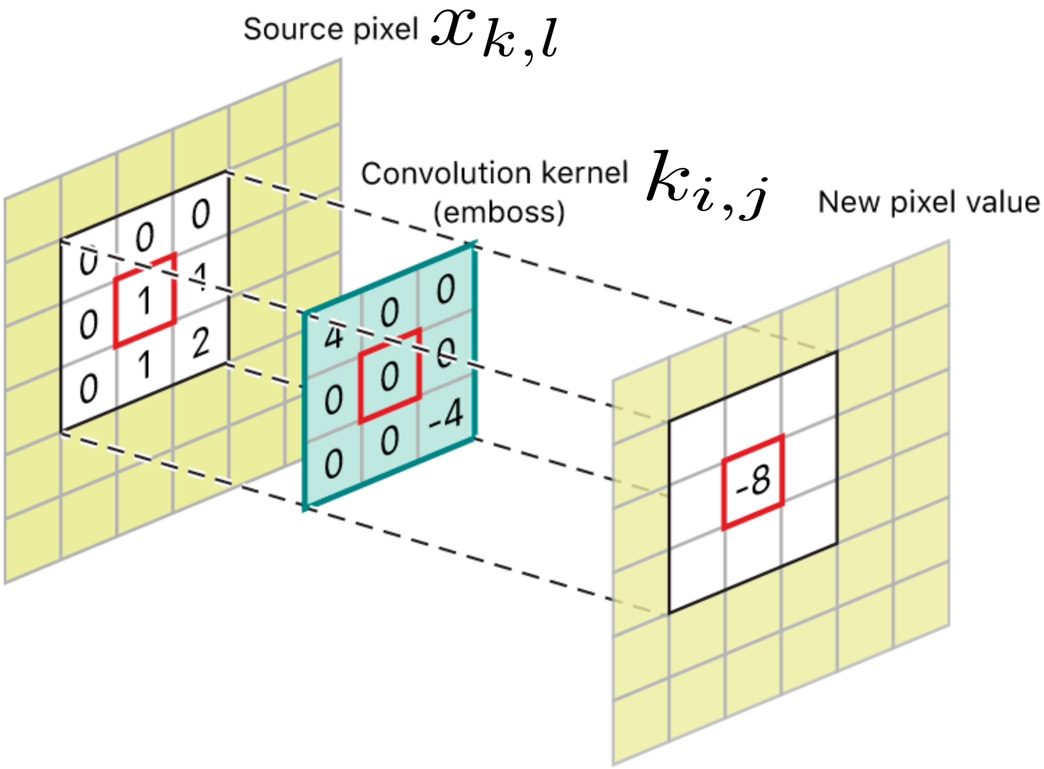
- Network structure :
  - Alternance of convolution & pooling
  - Flattering (sometimes called readout)
  - Multi-layer perceptron

Yann Lecun & al – 1998

Gradient-based learning applied to document recognition.

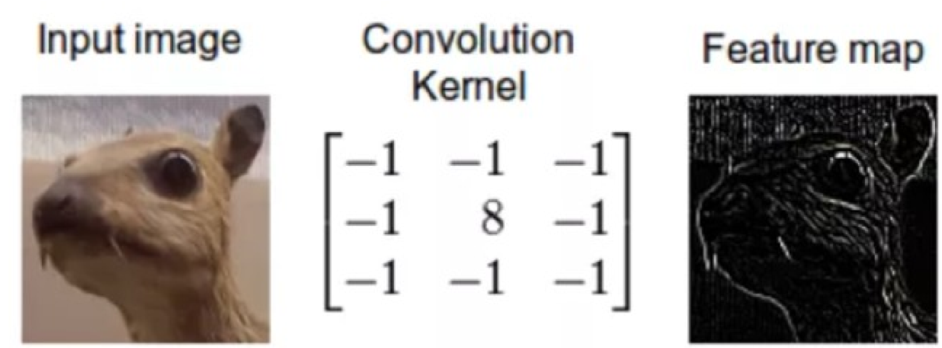


Convolution

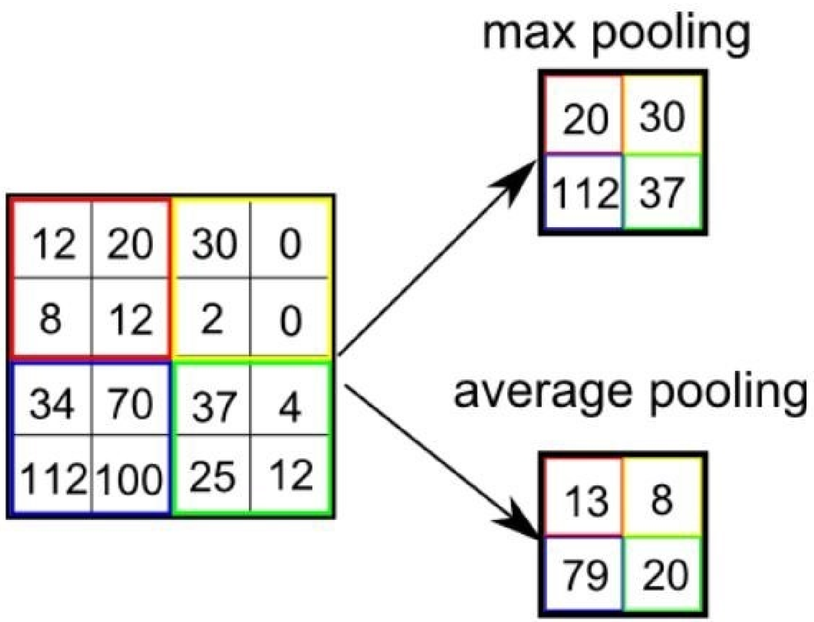


$$x'_{k,l} = \sum_{i,j \in [-s..s]^2} x_{k+i,l+j} * k_{i,j}$$

- Apply kernel on image to create feature maps
- Shared weights over all input space (translation invariance)
- kernel is learnable $k_{i,j}$
- Idea : creating maps of features (one kernel per feature)

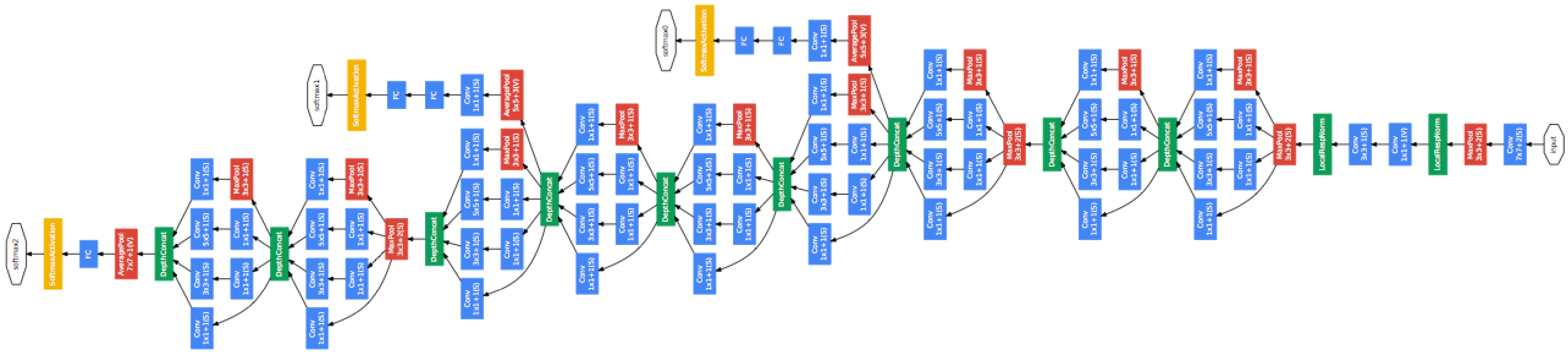


Pooling



- Reduce the dimensionality of the feature maps
- Move to higher level of abstraction
- Max pool is widely used

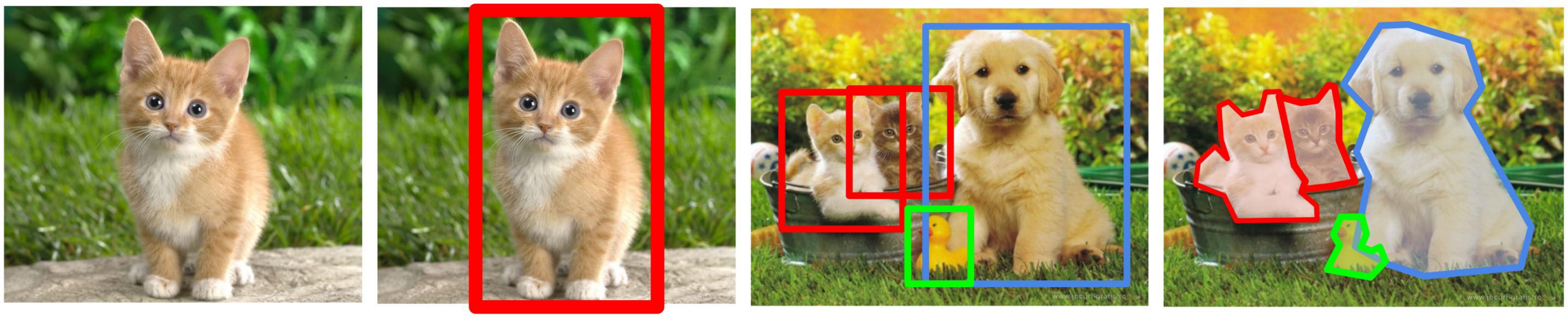
Hybrid Neural network



- Modern neural network : assembly of building blocks : neurons, convolution kernels, poolers
- Parametric : size and number of different layers, choice of transfer functions
- Training : Optimization of the parameters to minimize the loss function

What for ?

Classification	Classification + Localization	Object Detection	Instance Segmentation

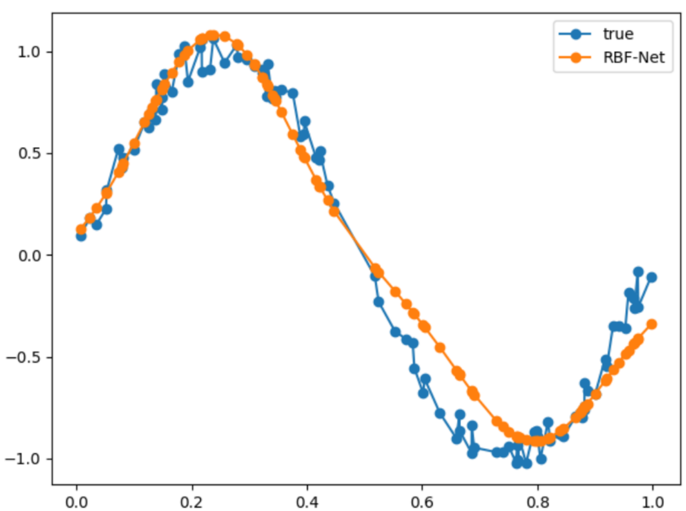


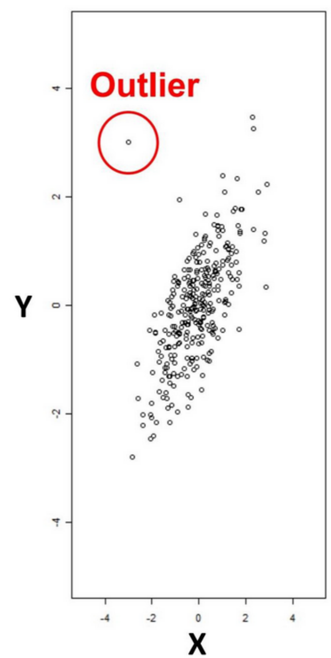
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

Single object

Multiple objects

Regression



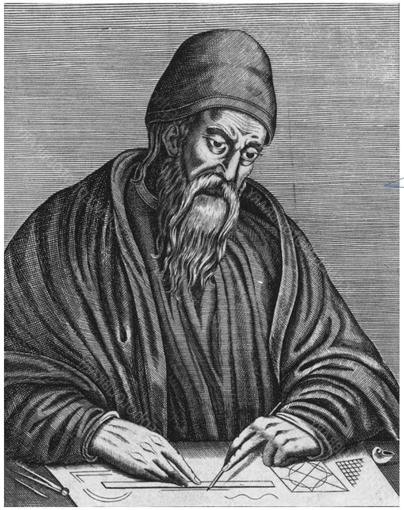


Anomaly Detection

Constrained Data Generation

Optimization

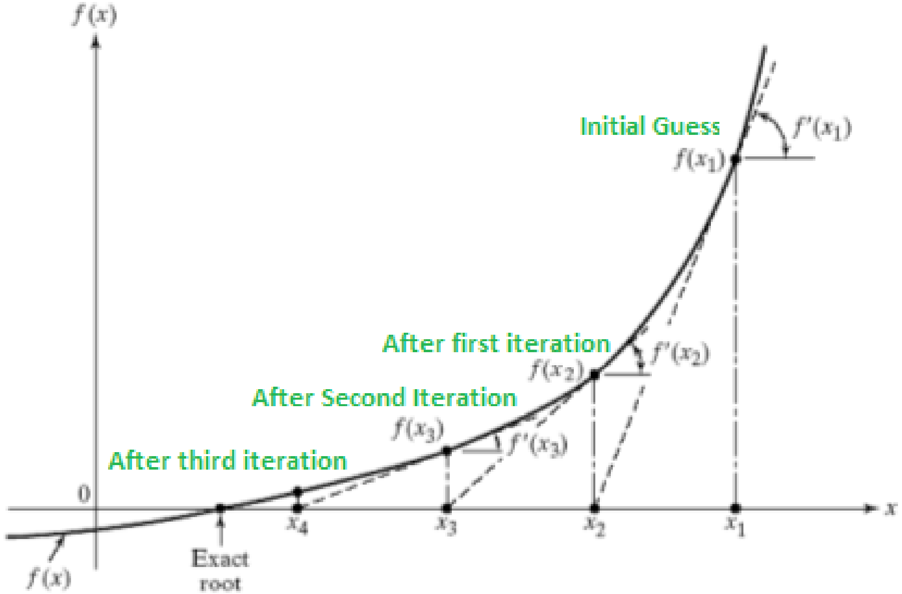
Definition and analytic answer



$$argmin(f(\mathbf{x})) = \{\,\mathbf{y}\,|\,\forall\mathbf{x}, f(\mathbf{y}) \leq f(\mathbf{x})\}$$

- Easy general formulation (Euclid)
- First general answer with differential calculus
  - f'(x)=0   and   f''(x)>0
  - Requires analyticity, derivability and solvability

A first heuristic



$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Crazy ! Coming to me from the sky !

- First heuristic by Newton
  - iterative method to find a zero of the derivative
  - Second order method
- Only local derivatives required
- But : Hessian matrix computationally very expensive

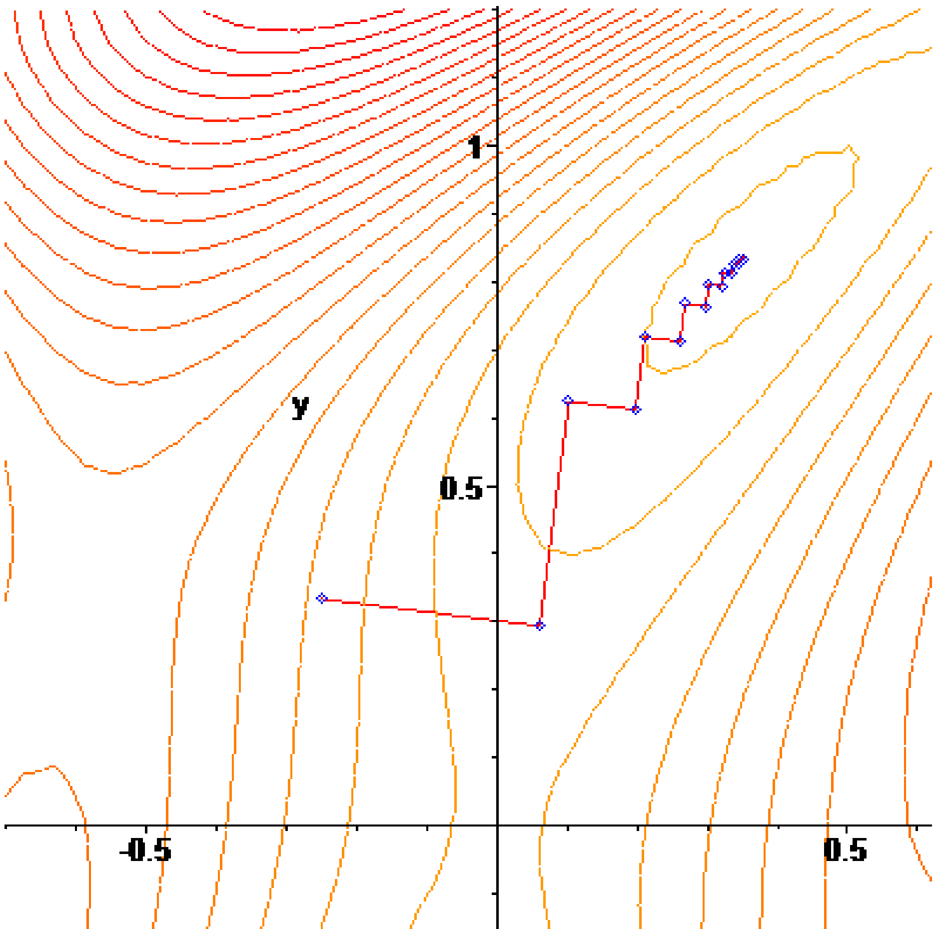
→ need a first order solution

Optimization as a Blind Walk

- « Following the slope » method
- Only local knowledge of the field required
- Known as gradient descent algorithm class
- Proposed by Cauchy in 1847

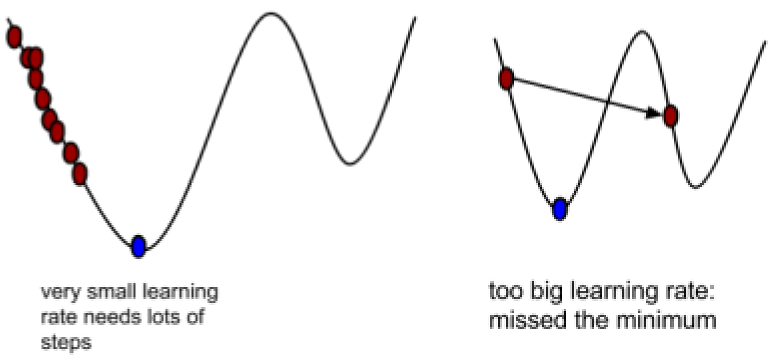
Gradient Descent

$$\nabla J(\Theta) = \left\langle \frac{\partial J}{\partial \Theta_1}, \frac{\partial J}{\partial \Theta_2}, \ldots, \frac{\partial J}{\partial \Theta_n} \right\rangle$$



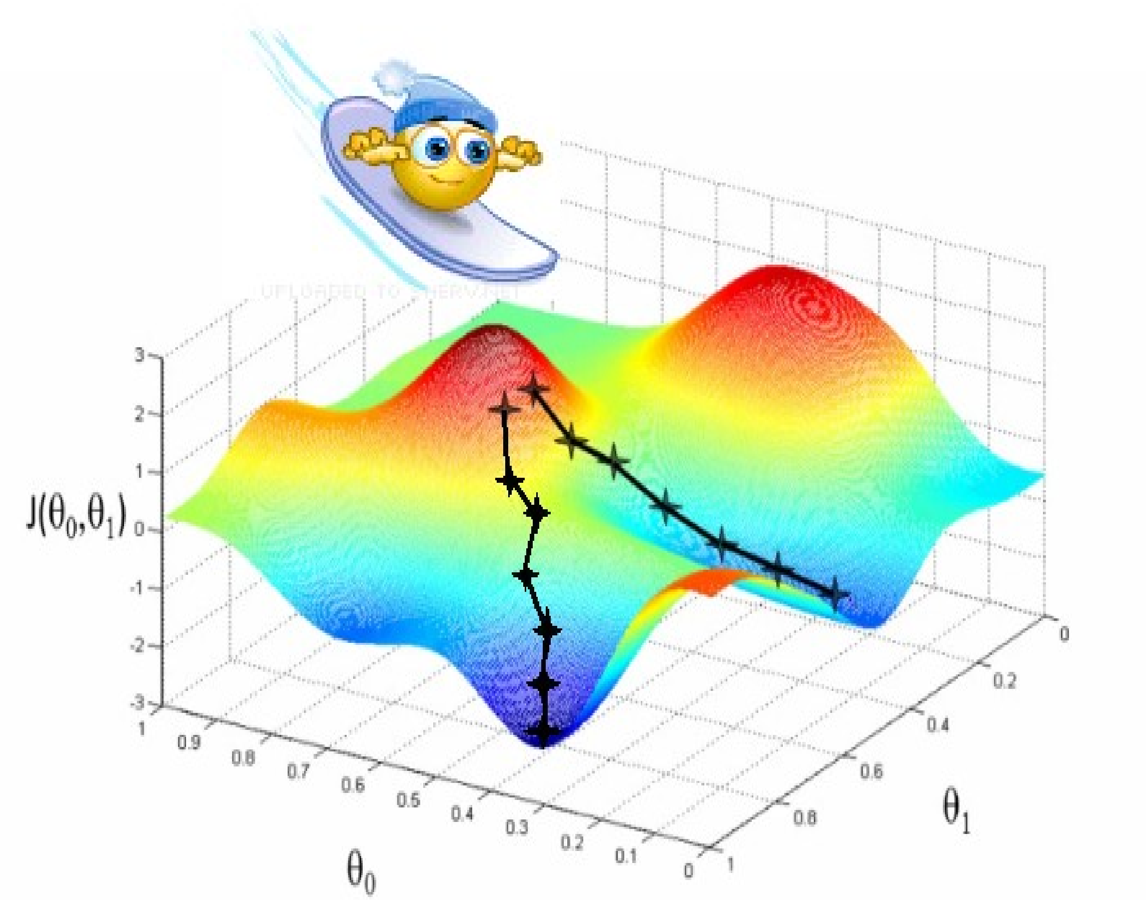
$$\Theta = \Theta - \alpha \nabla J(\Theta)$$

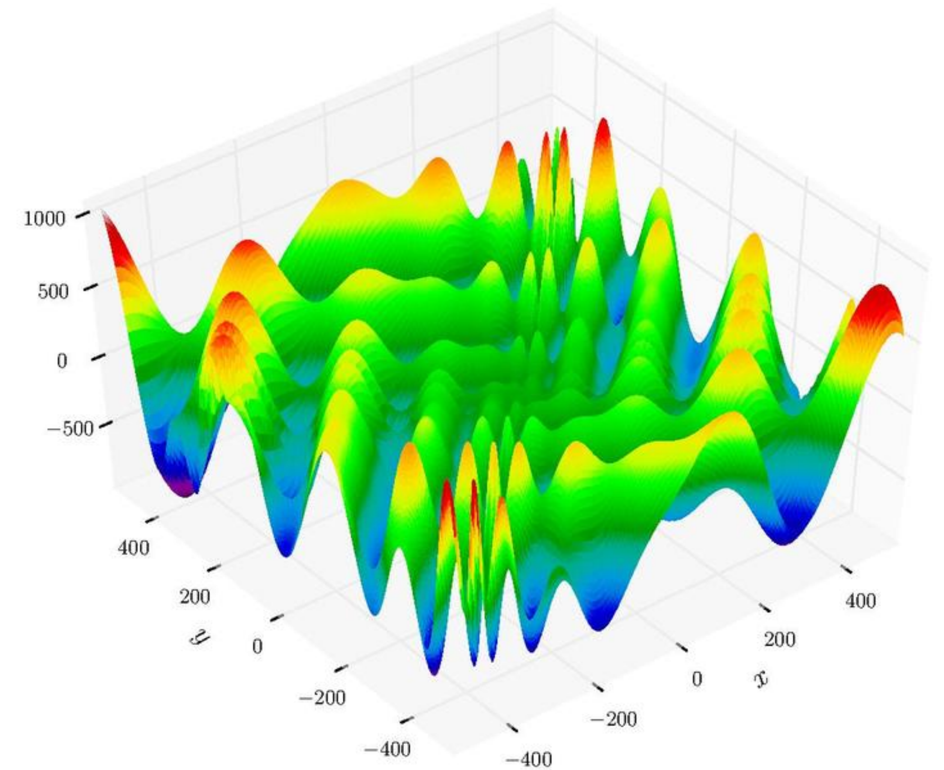
α : step size



Precision vs performance

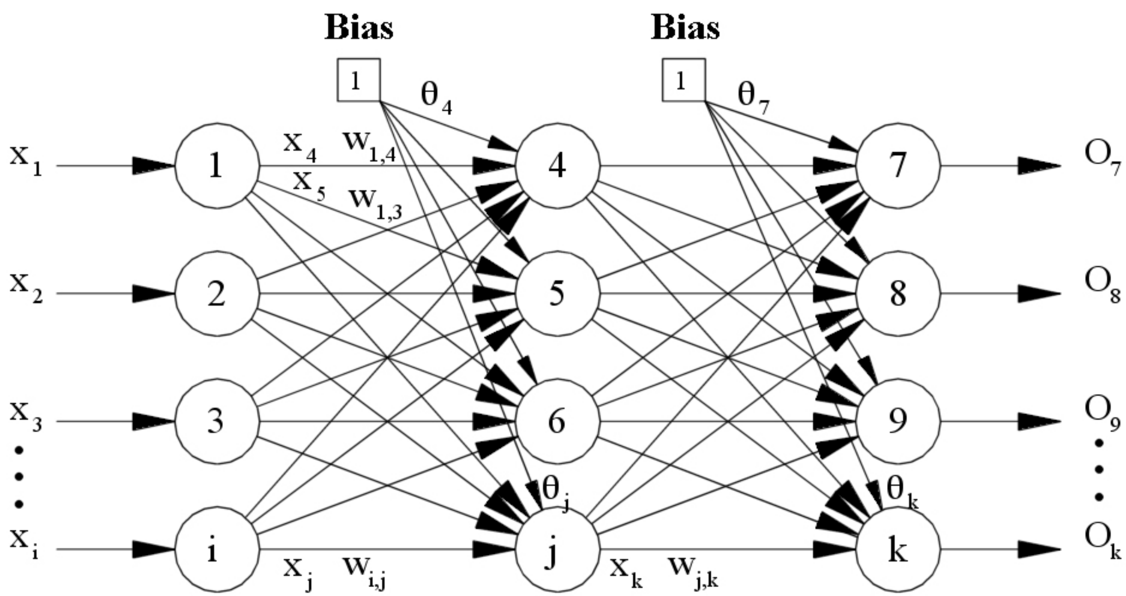
Gradient Descent & Convexity

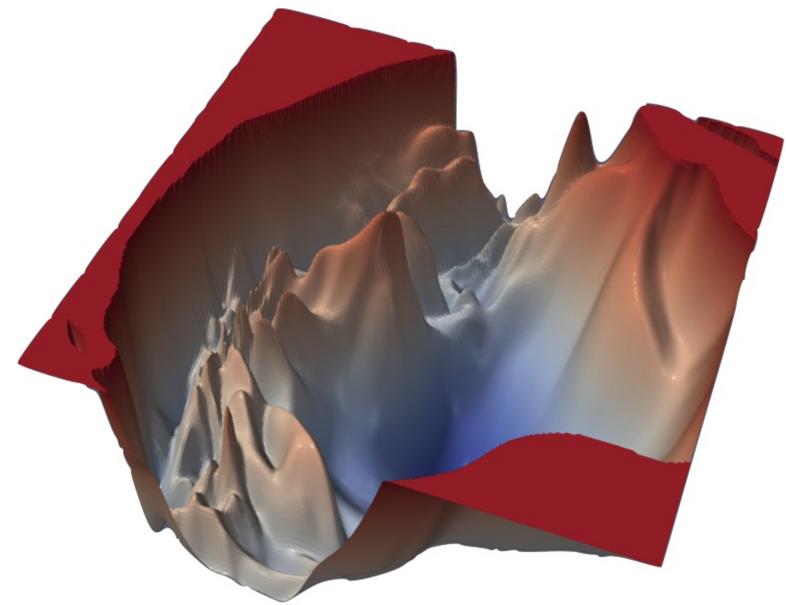




- Depend on the starting point
  → require convexity
- Practical solution : multiple random starts (no guarantee)
- Different class of convexity
- No convexity → no optimization

Optimizing Neural Networks





Li & al, « Visualizing the loss landscape of neural nets, 2018, 1712.09913

- Learn an algorithm by labelled data
- Optimization space : all weights named globally θ
- Function to optimize : loss function L(θ,data)
- Searching for a good minimum in the loss function

Why does it work ?

Lecun & al, The loss surface of multi-layer networks, 2015, 1412.0233

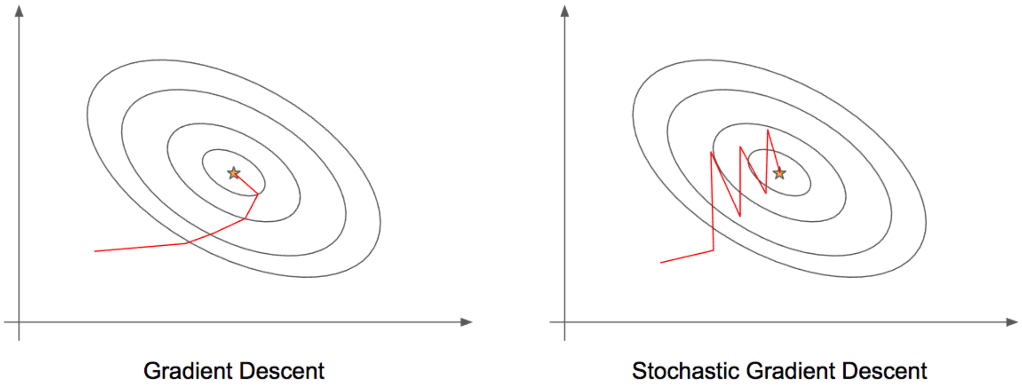
- Perceptron ↔ spherical spin-glass model
- theoritical results
  - Exponential (in dim) number of local minima
  - Minus exponential number of bad local minima
  - Good local minimum :

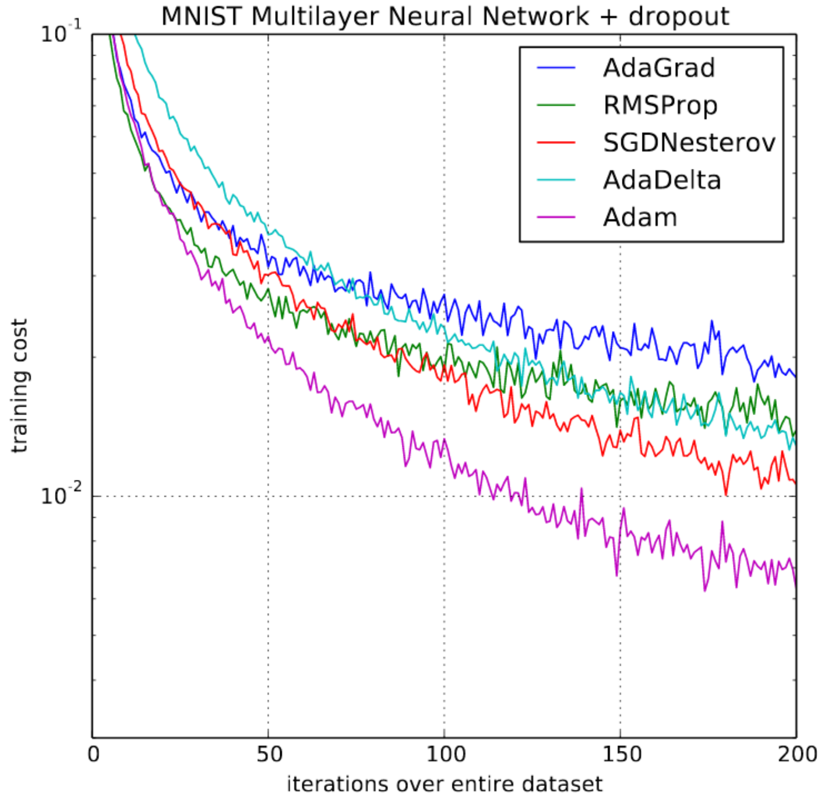
$$loss(min_{loc}) - loss(min_{glob}) \leq \epsilon$$

  - Funnel global shape
- Global minimum is probably overfitting
- Deep learning (dim is big) gives better results

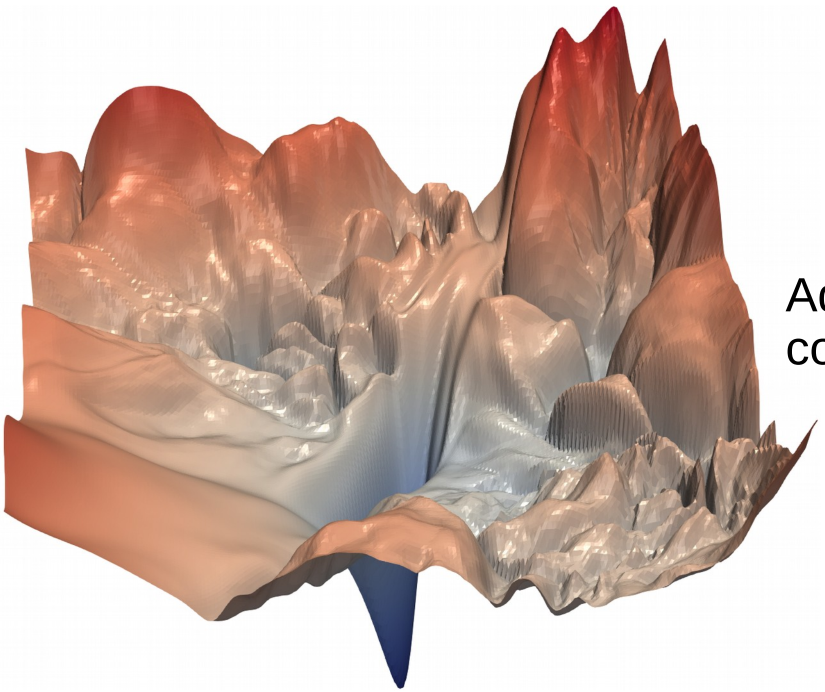
Optimizers for DNN

- Gradient descent implies huge storage of derivatives
- SGD slices the problem input by input :
  - slower the convergence
  - add variance
  - save space
- Big diversity of SGD derived algorithm
- Adam : a method for stochastic optimization, Kingma & Ba, 2017, 1412.6980
  - Automatic adaptative learning rate per parameter
  - Best performance ever → rules the world



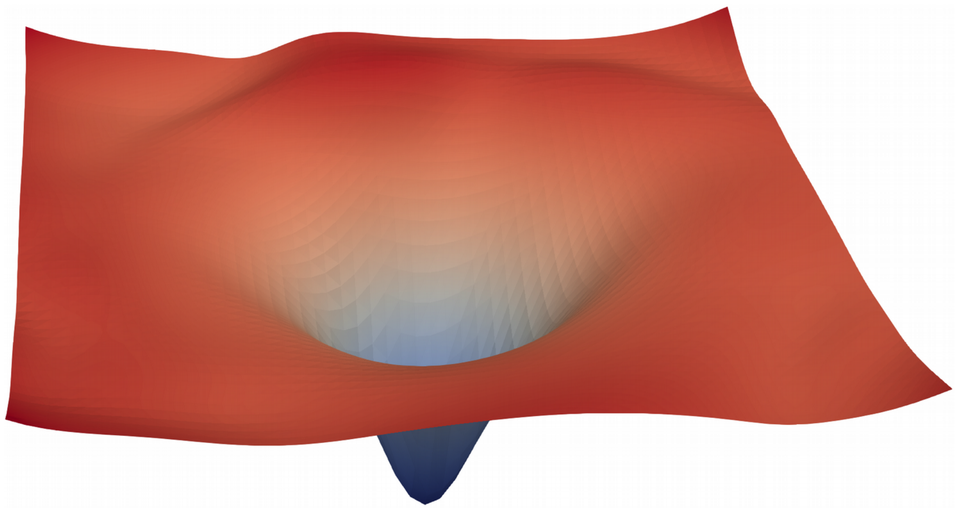


Topology Influence

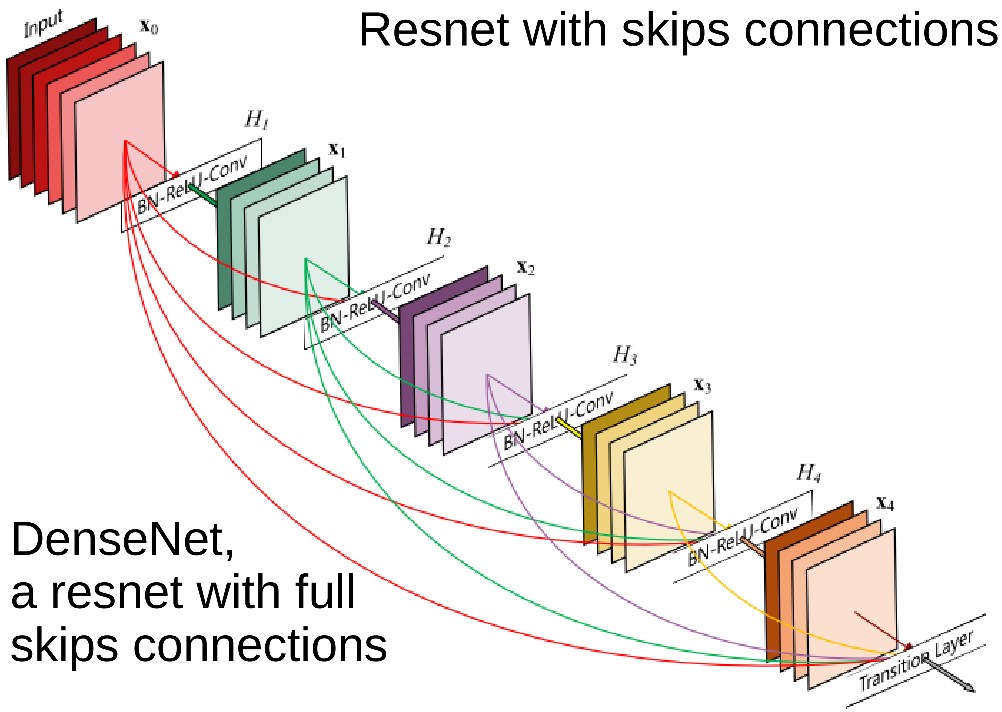


Resnet (very deep convolutional NN)

Adding skips → connections



Resnet with skips connections

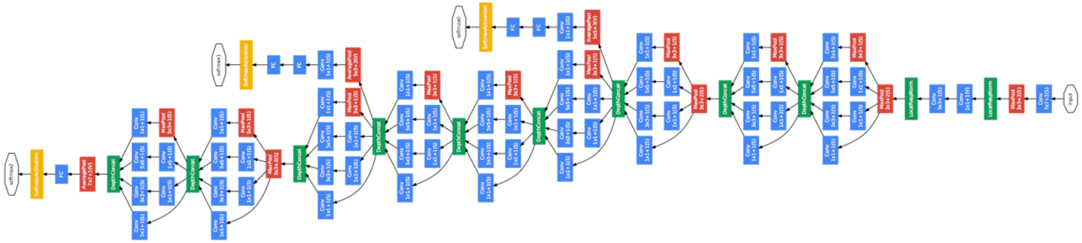


DenseNet,
a resnet with full
skips connections

Topology influences dramatically the loss surface shape

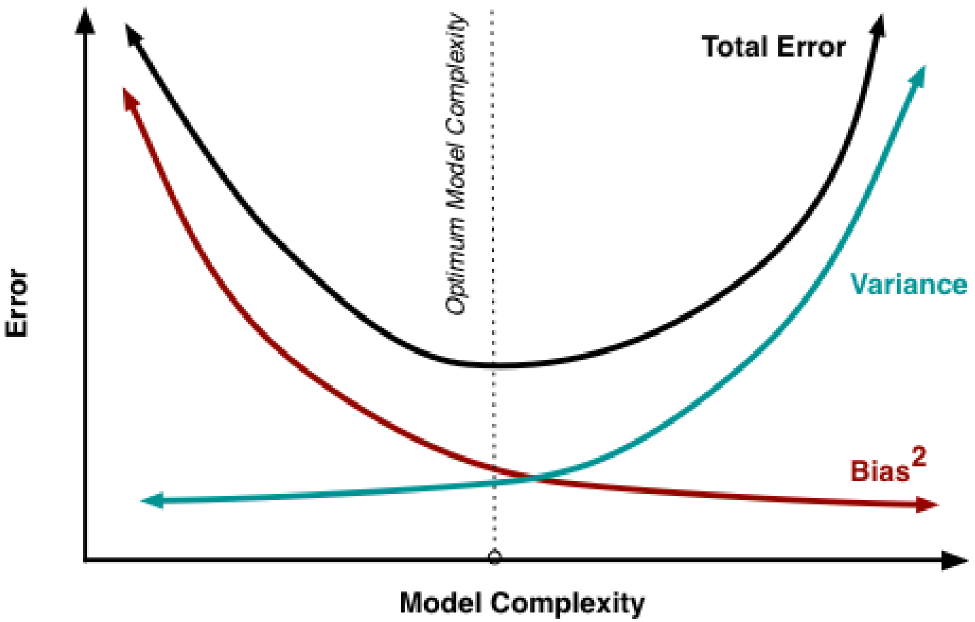
Two reasons to optimize topologies

1. Getting best distribution of building blocks



- No thumb-rule
- Often qualified as a dark-art

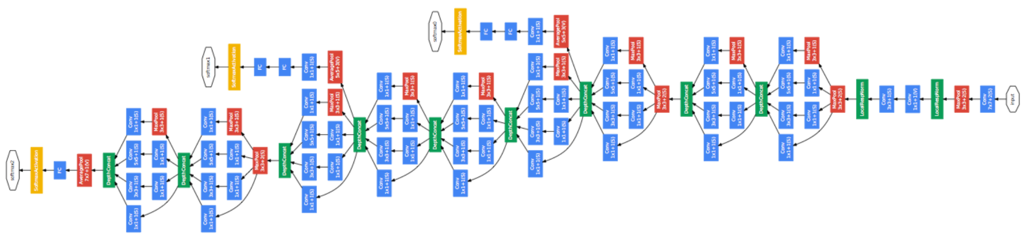
2. Find the bias-variance tradeoff



- Too simple model → fit error increased
- Too complicated model → statistical error (variance) increased
- Gives a hope for global convexity
- Help us saving resources

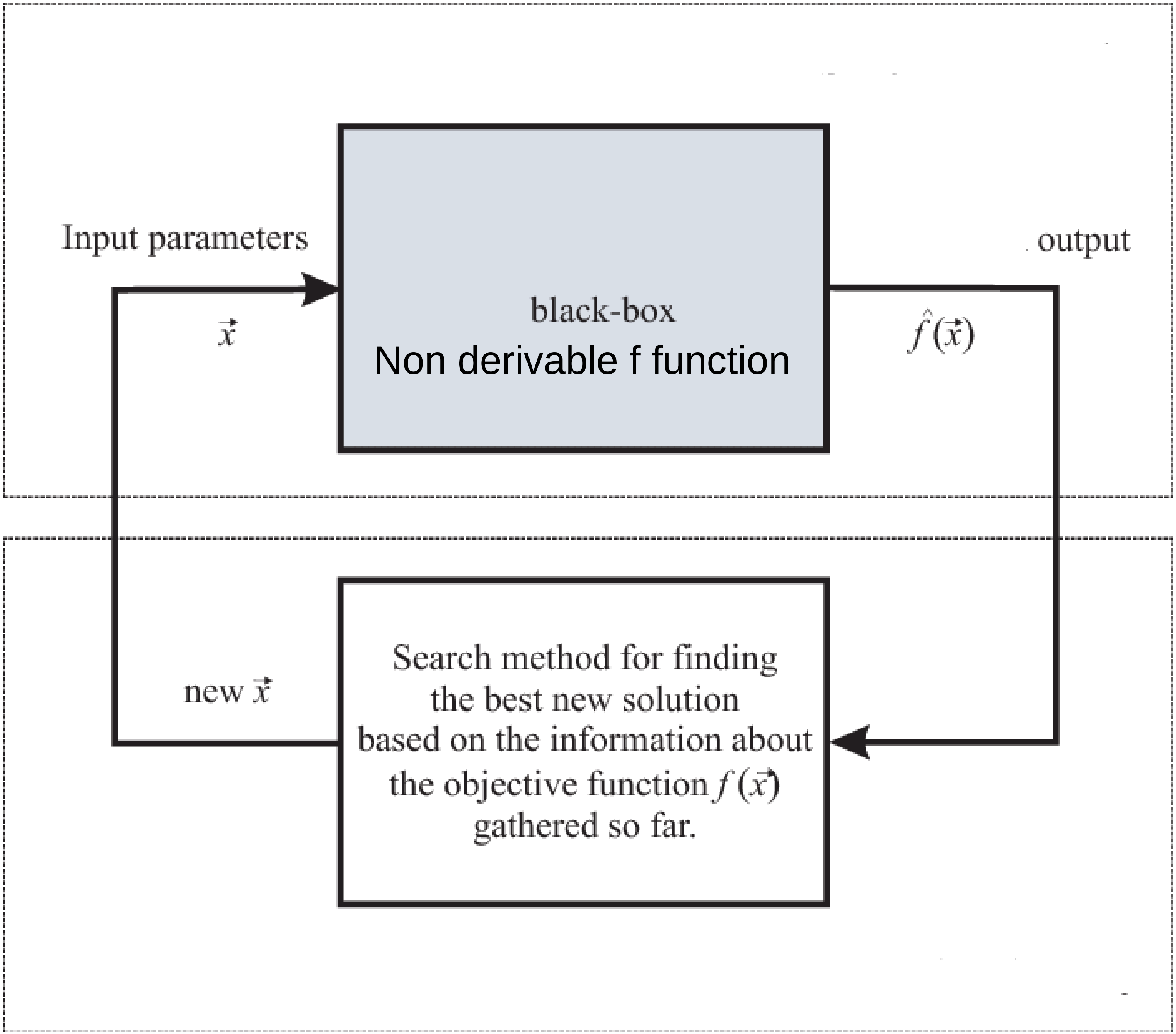
Topology Optimization

Topology Optimization

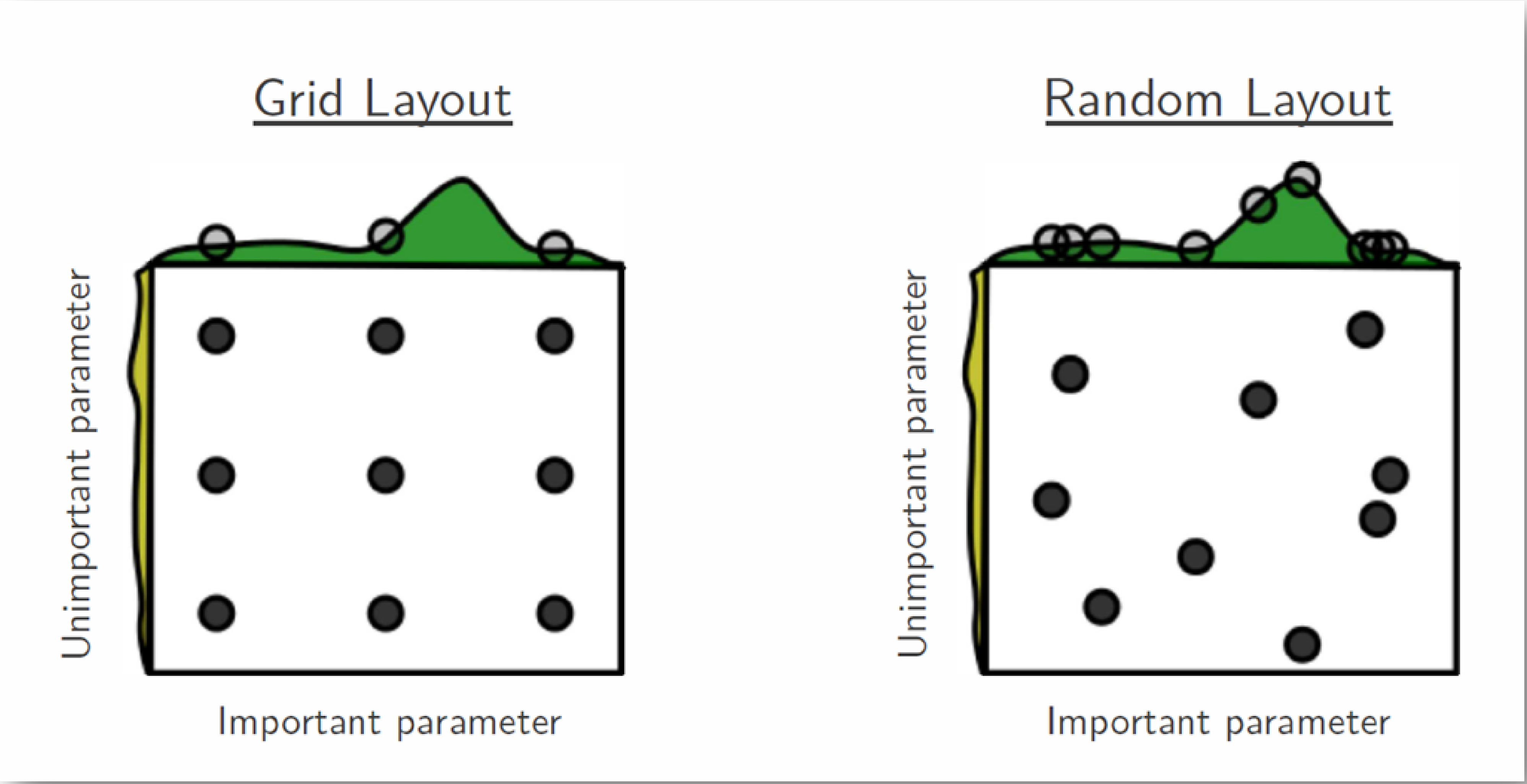


- Best topology under resource consumption constraint → optimization problem
- Parameter space : parametric representation of network
- (sl1=400, ks1=5, ps1=32, … , tf='lrelu', ck1='euclidian')
- Loss function : best precision with parametric trained network
- All right, doing gradient descent again ?
- Additionnal constraints
  - Each point is very expensive to calculate (full training)
  - The loss function is not derivable (even numerically)

Black Box / Zero-Order Optimization

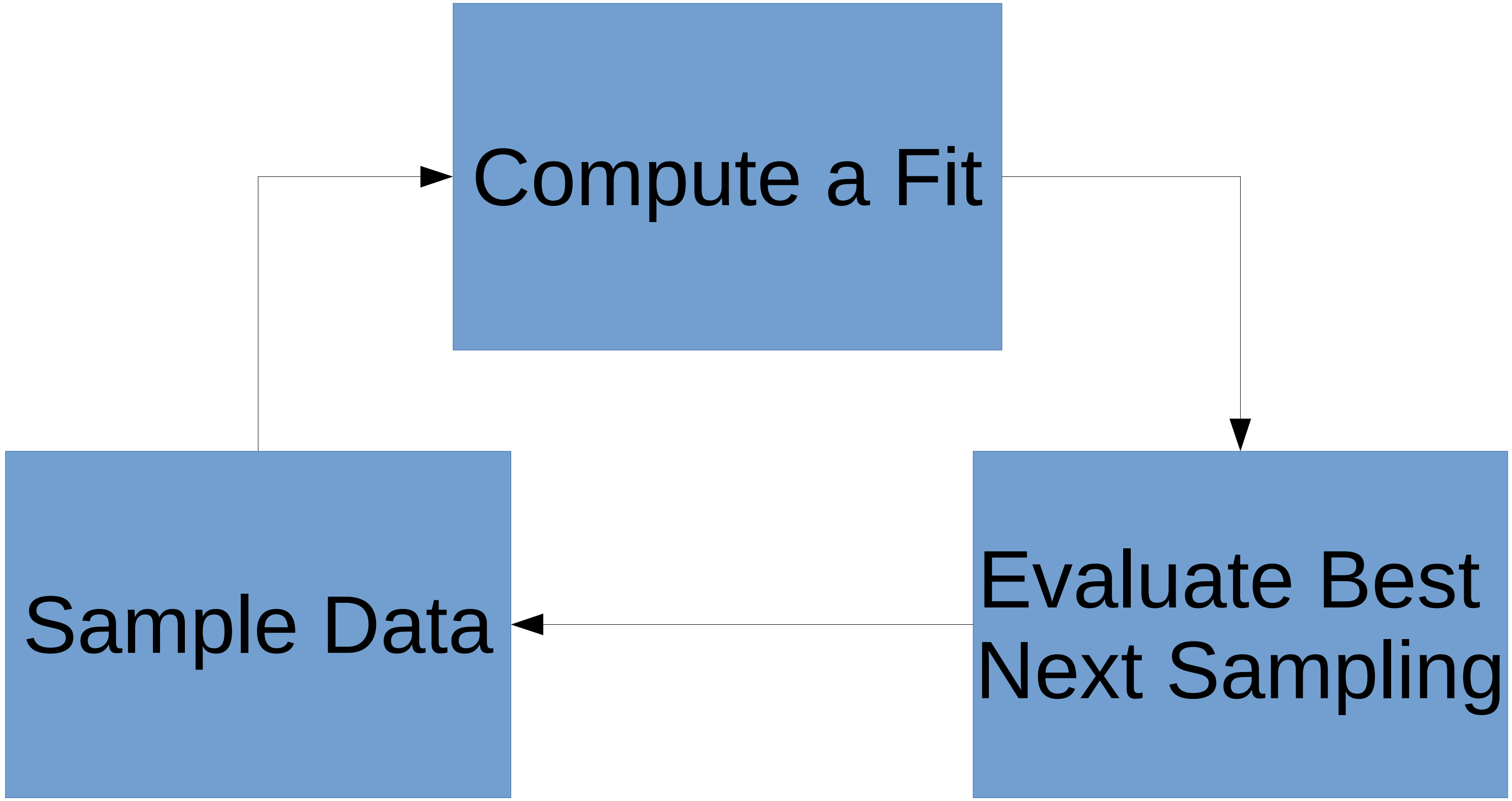


Grid and Random Search



Dimensionality

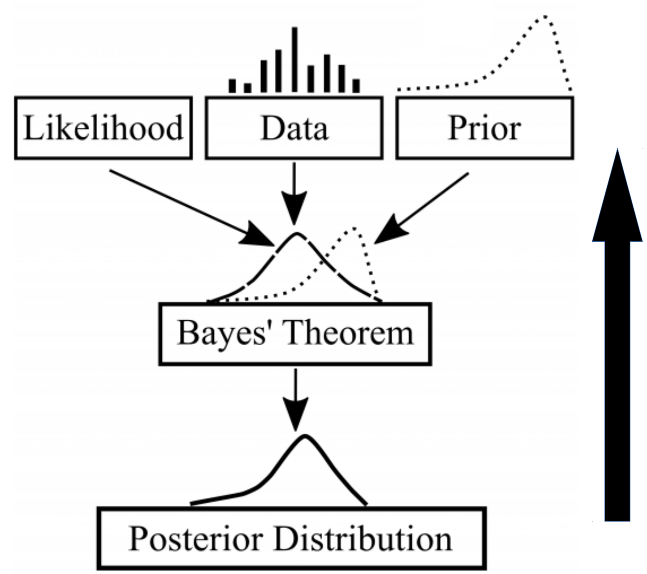
Data-driven Sampling



Best algorithm for expensive loss function
Bayesian Optimization

Bayesian Inference

Statistical method to infer a generative model from data



Recurrence if new data

Model Plausibility
→ posterior

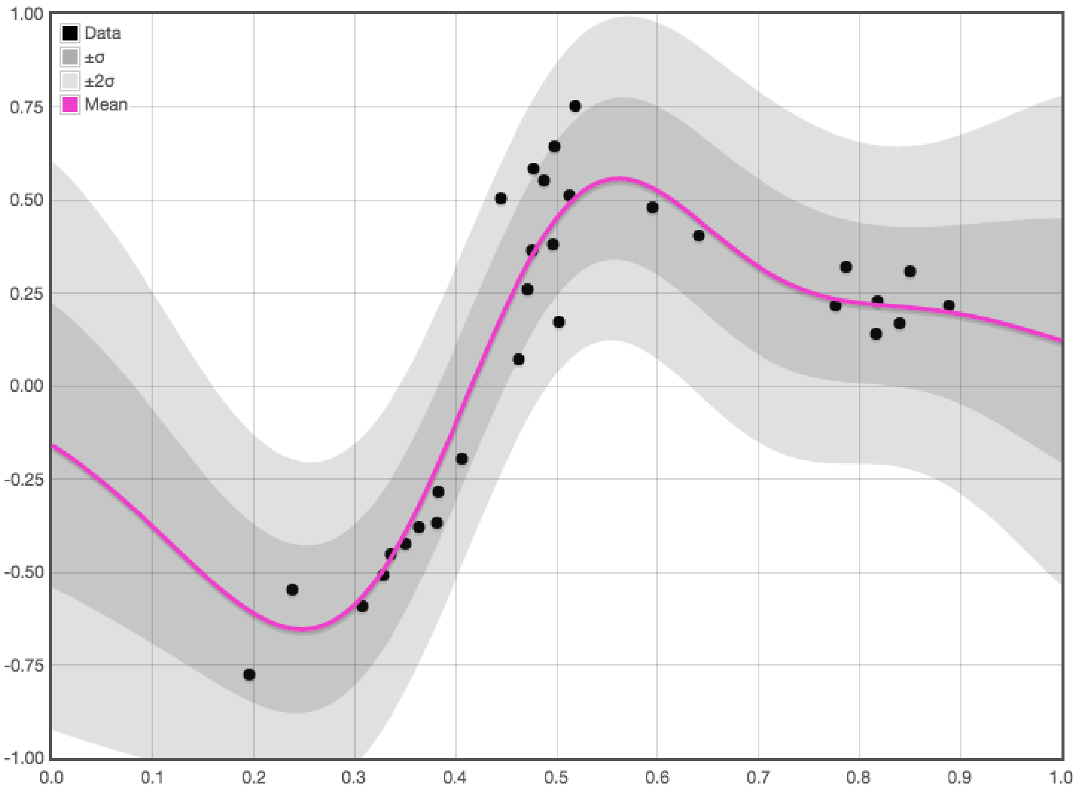
Data Likelihood

Prior

$$p(model|data) = \frac{p(data|model).p(model)}{p(data)}$$

Normalization

The model class : Gaussian Process

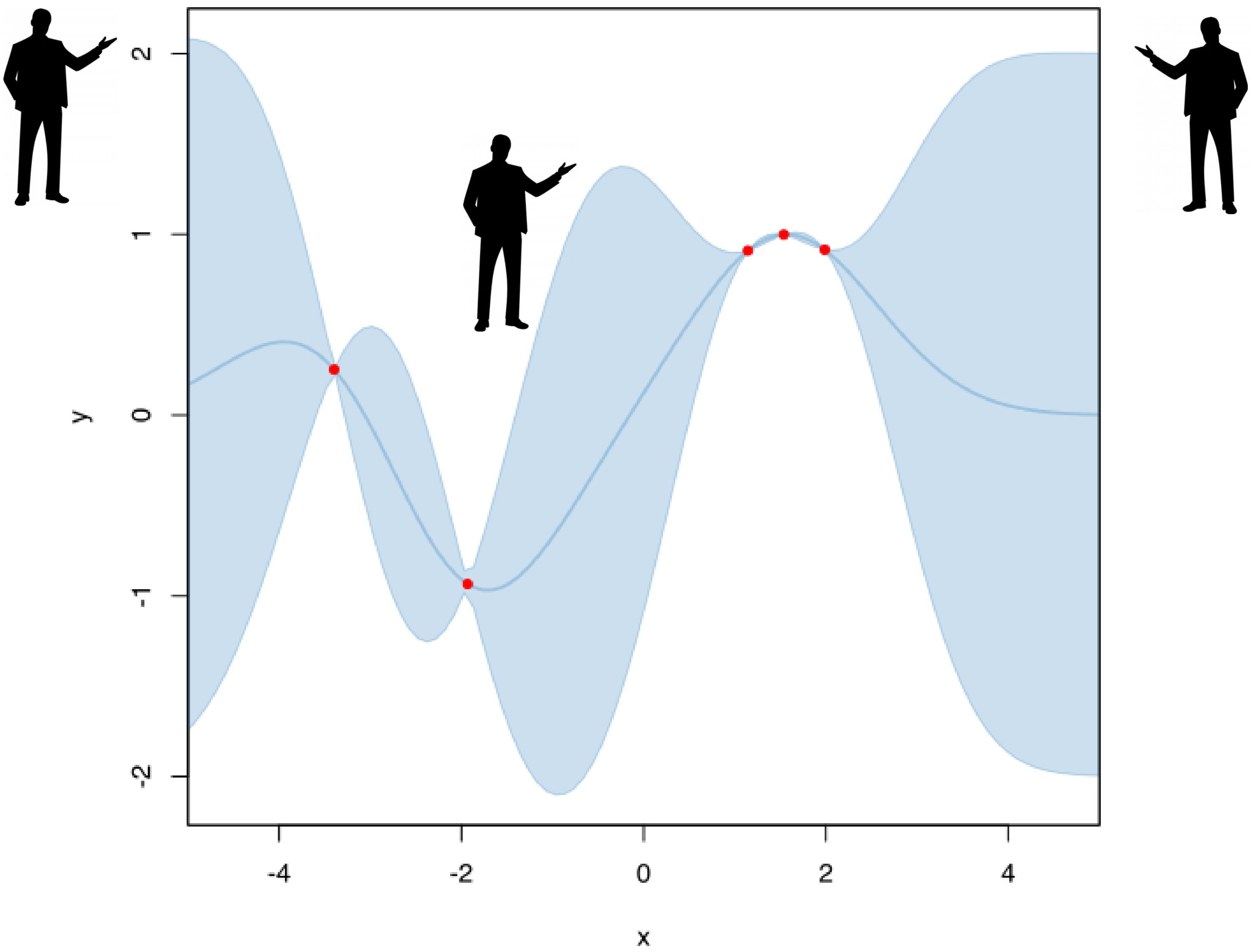


- Probabilistic function
- Arbitrary dimension
- Defined by
  - mean(**x**) function in pink
  - covariance(**x**)

    width of grey bands
  - Good fit for known data
  - Covariance estimated

    as distance function

Where to search ? Promising points



Can we express this as a function ?

Acquisition functions

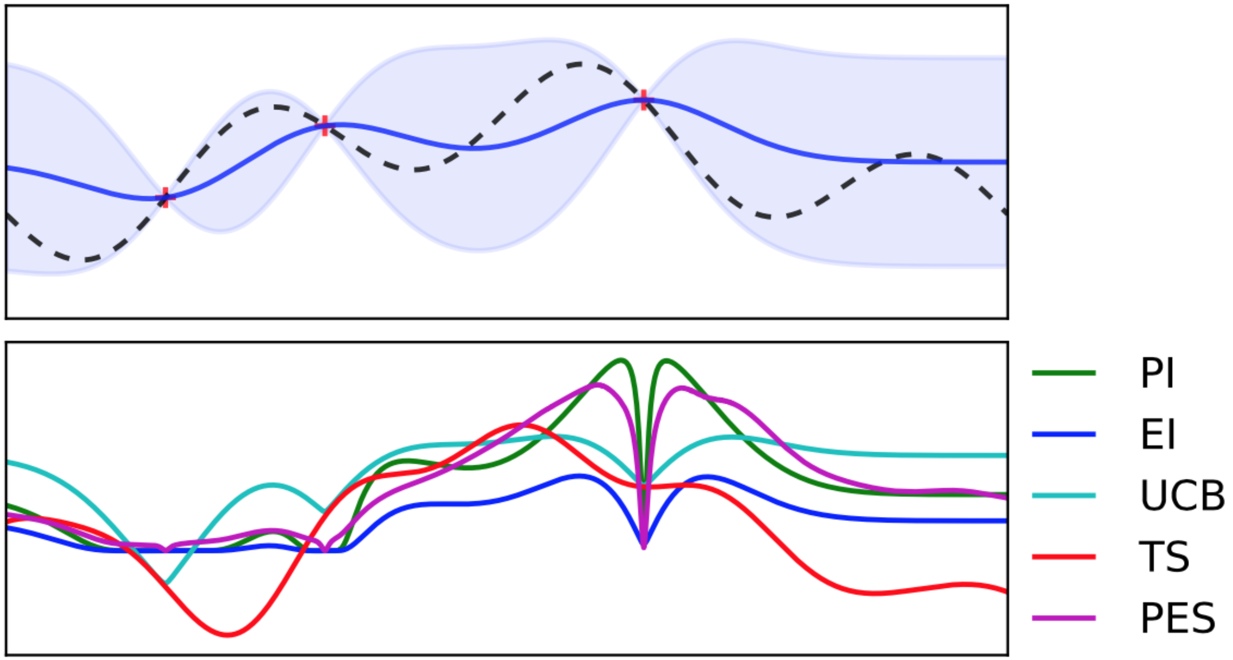
- Upper Confidence Bound (UCB)

$$A(x) = \pm\mu(x) + \kappa\sigma(x)$$

- Esperance of Improvement (EI or EOI)

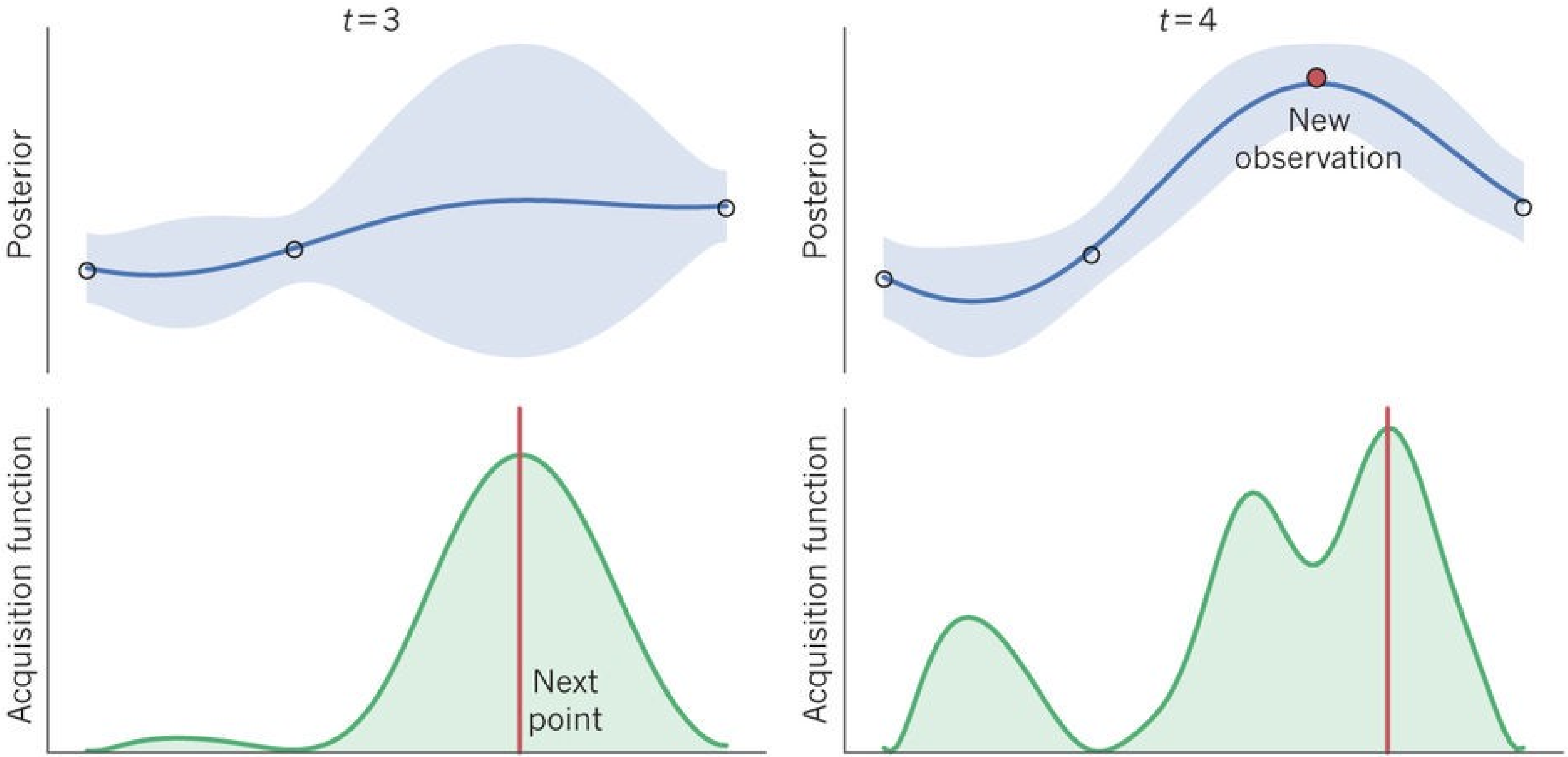
$$EI(x) = \mathbb{E}(max(f(x) - f_{max}, 0))$$

- Easy to compute on the whole space
- Rely only on Gaussian process



Bayesian optimization

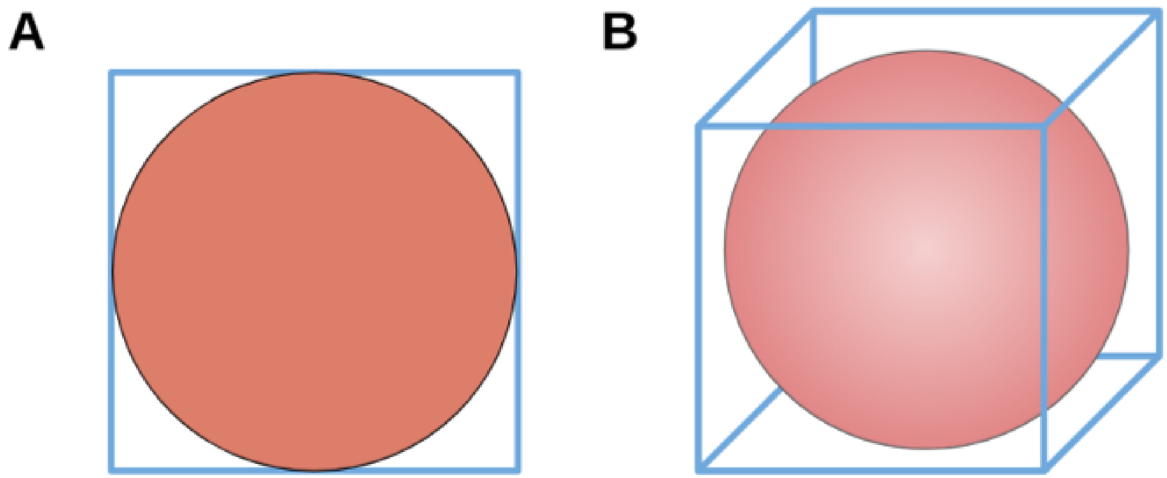
Jonas Mockus, Bayesian Approach to Global Optimization, 1989



$$A(x) = \pm\mu(x) + \kappa\sigma(x)$$

$\kappa = 4$

Limitation: Curse of Dimensionality



$$\frac{V_{hypersphere}}{V_{hypercube}} = \frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)} \to 0 \text{ when } d \to \infty$$

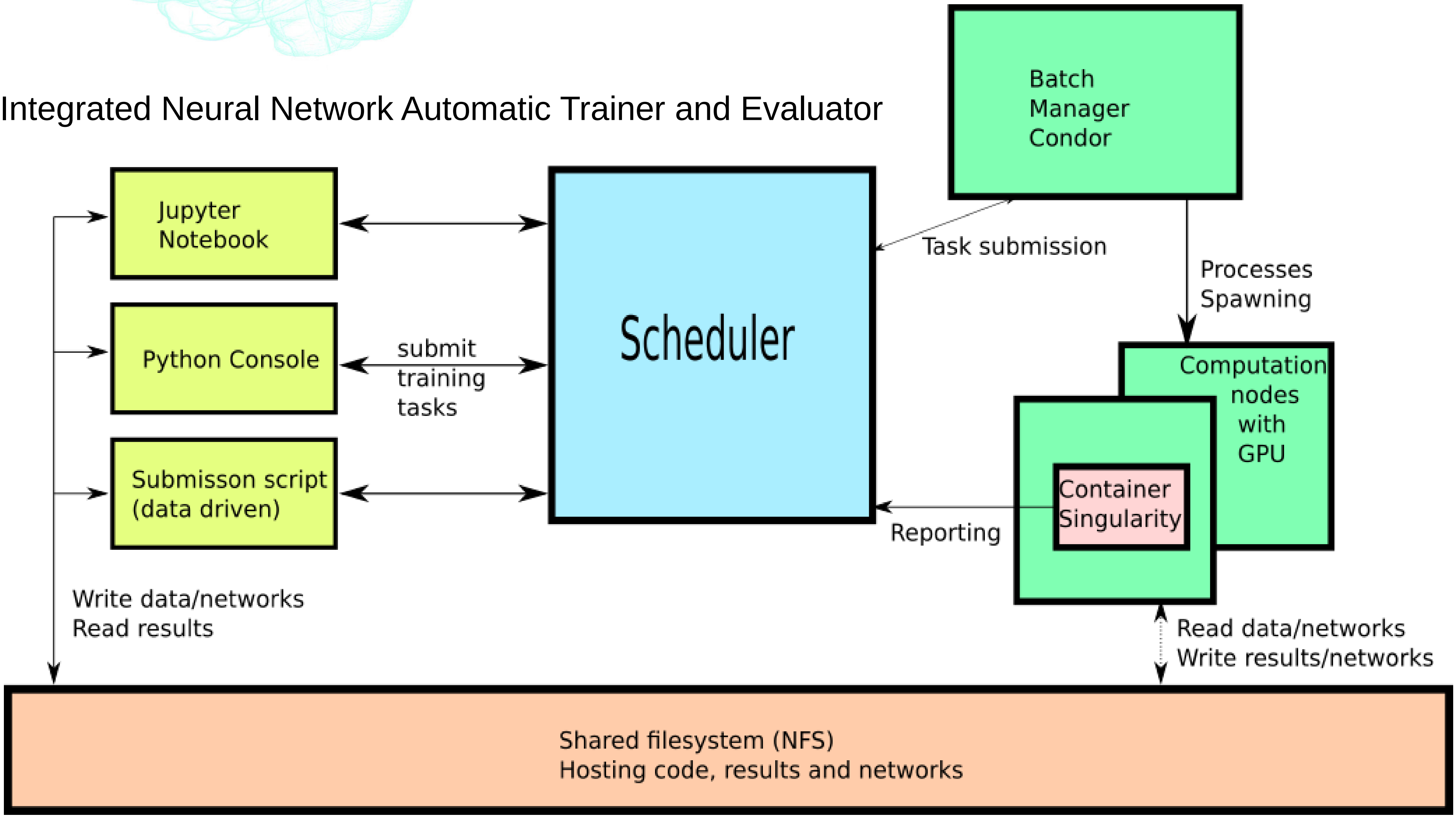
- Necessary data amount grows exponentially with dimension
- Concerns all « neighbouring » fit techniques
- BO is limited in dimension (around 20-30)
- Neural nets are not concerned because their loss function has a special shape (self-regularization)

Implementation & example

Innate

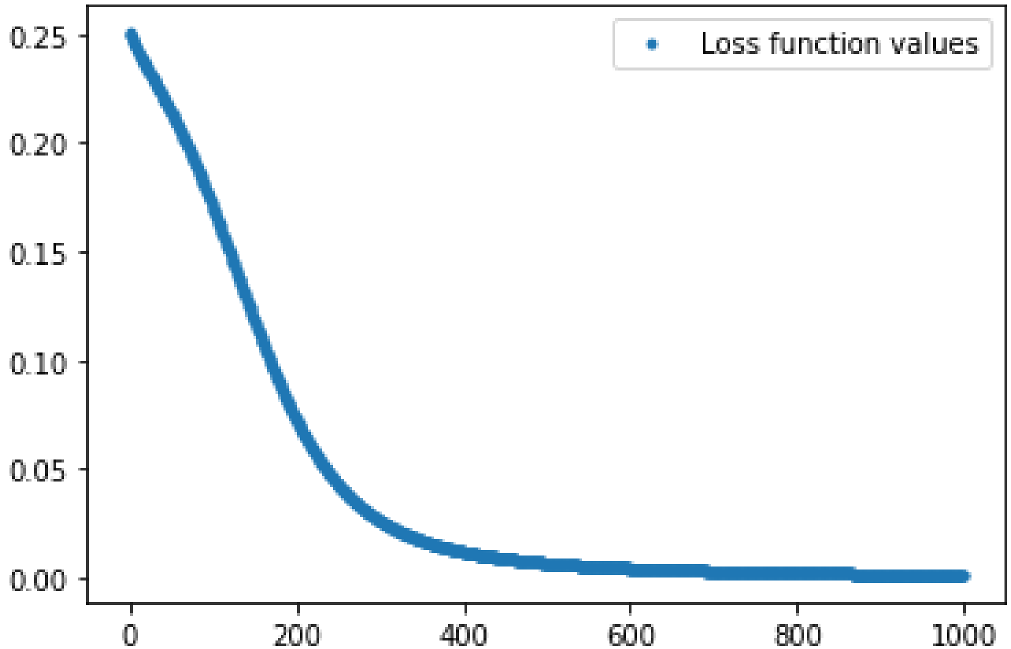
- Runtime encapsulate all algorithmic complexity
→ ease of development
- Based on both Keras & Pytorch

Integrated Neural Network Automatic Trainer and Evaluator



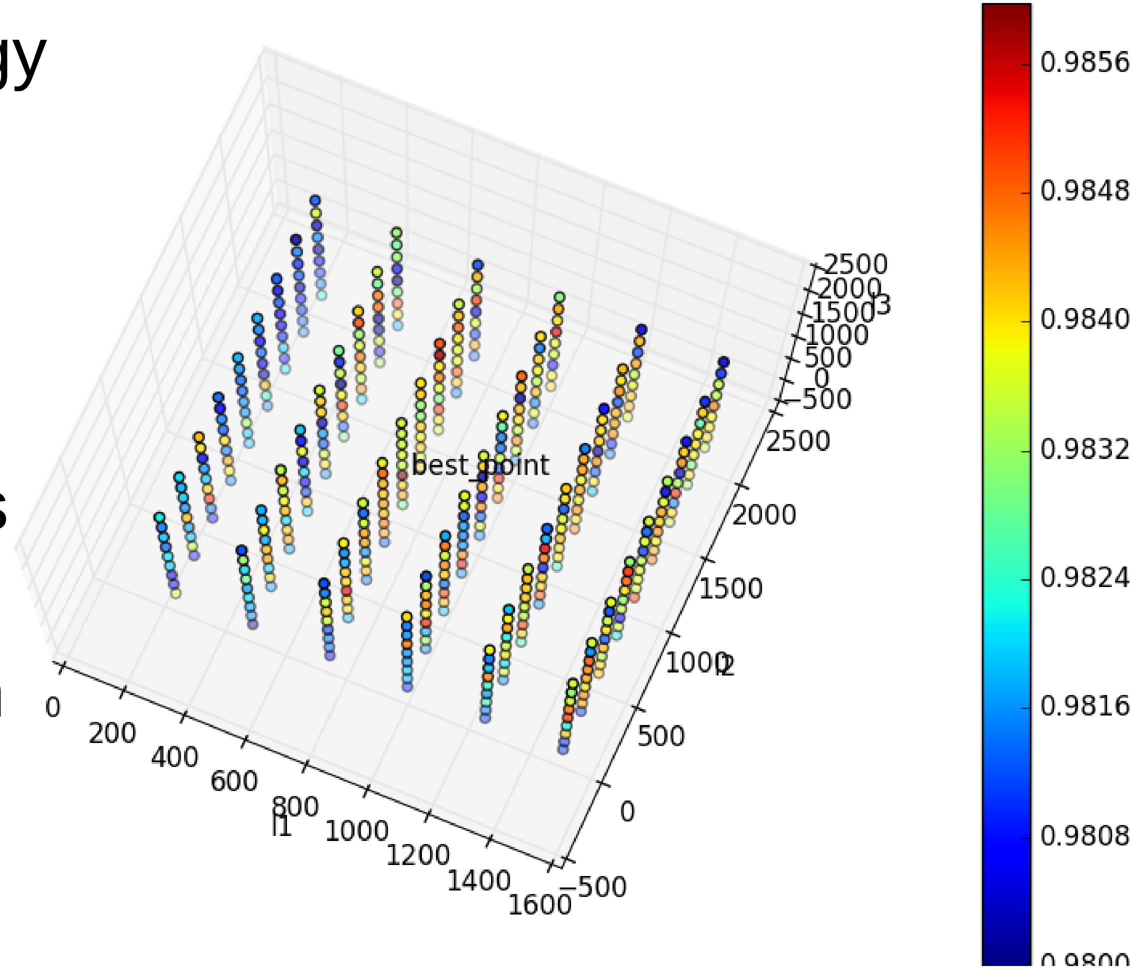
Innate API

```
import innate

#connect to scheduler
ie=innate.init("llrinnate.in2p3.fr")

#launch a simple training (can be asynchronous)
res=innate.train_net(ie,task_name,nn_filename,data_filename,
results_folder,nb_epochs=1000)

#plot result
print("elapsed time :"))
print("%s"%(res["etime"]))
innate.plot_loss(res)
```



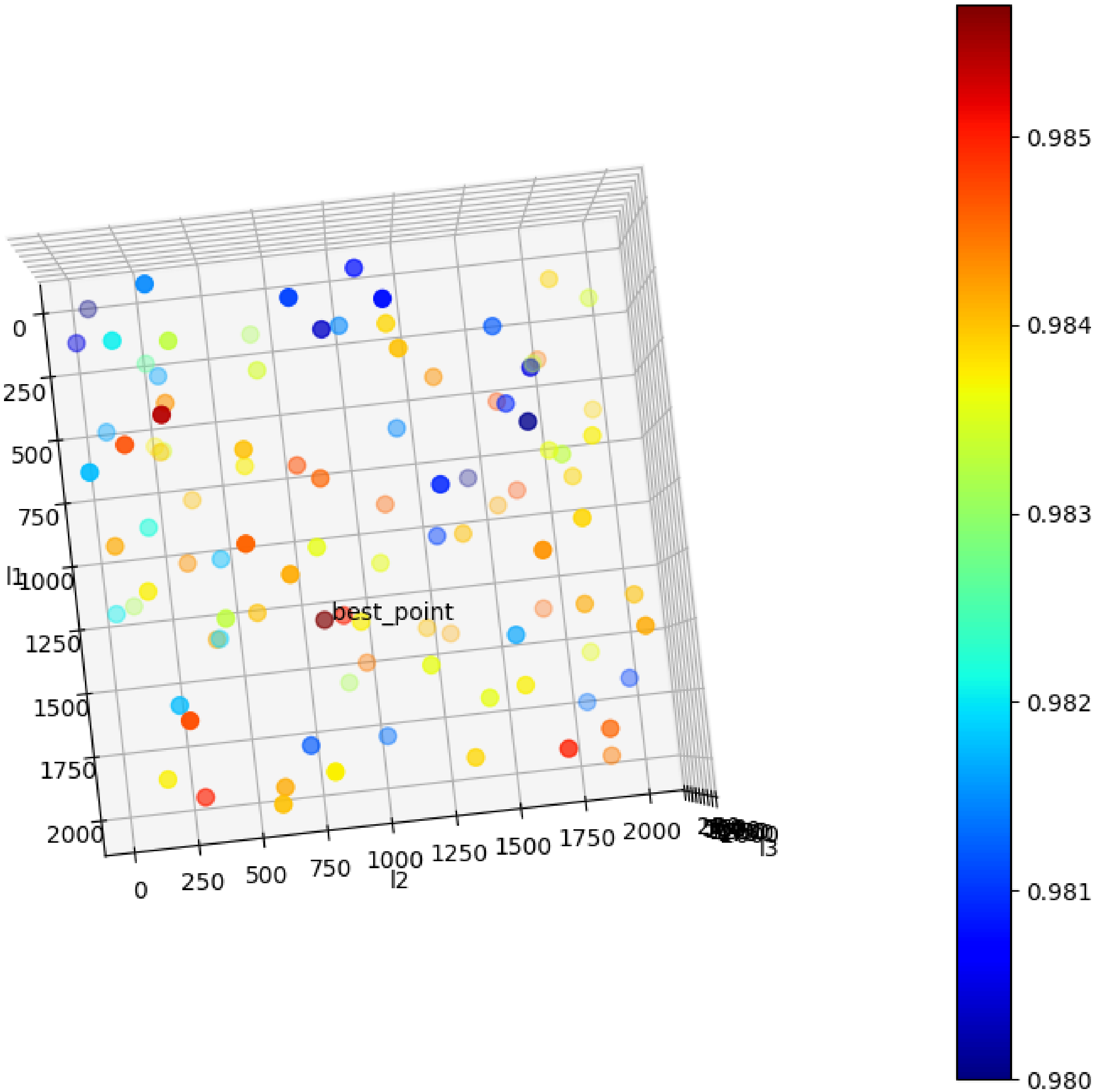
Grid search topology exploration

- Exploring a 3 layers topology between 1 and 2000 neurons
- Inputs : particle clustered energies per layer
- Objective : classifying pions vs electrons
- Precision=1-efficiency (pion seen as electrons)
- 294 points
- Best point : 750 1000 750 with precision 0.985977



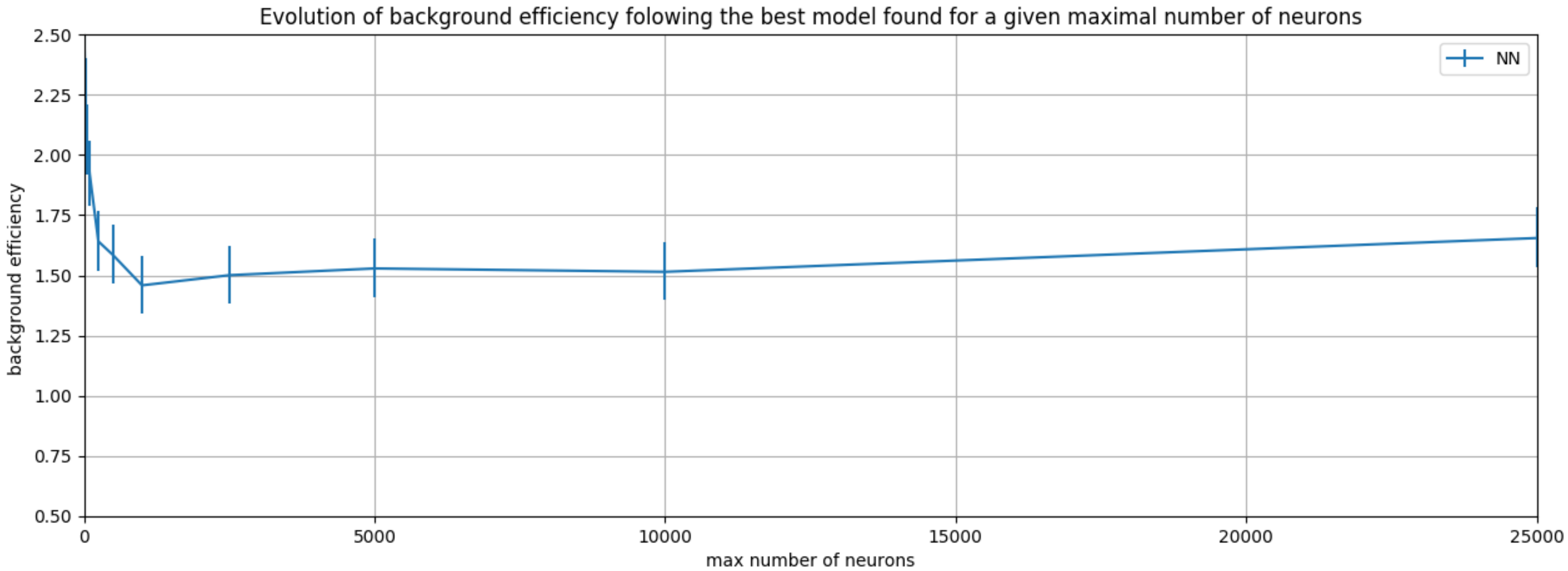
Bayesian Optimization

- Bayes-opt implementation
- Only 100 points
  - 20 random points
  - 80 fit points
  - Could be optimized (50)
- Best point : 1341 835 1117 with precision 0.985696
- Same precision with 1/3 points

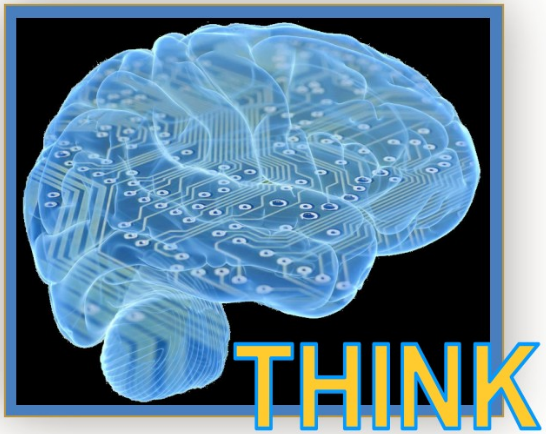


Global Performance over Resource Avaibility

- Taking different max size and searching for best size
- Max 15 layers
- Bias-variance trade-off highlighted



Perspectives



- For the THINK project
  - Create an easily deployable innate package
  - Create a tutorial for Bayesian optimization
  - to be discussed...
- For Innate
  - Implement Parallel Bayesian Optimization
  - Try on alternative architecture
    - Graph convolution networks
    - Auto-encoders
- Keep the trend in a VERY prolific domain !!