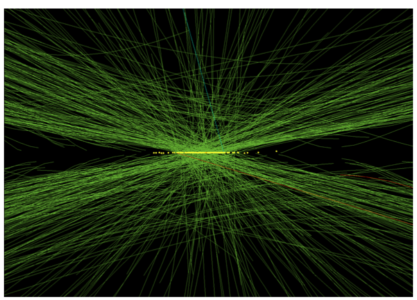
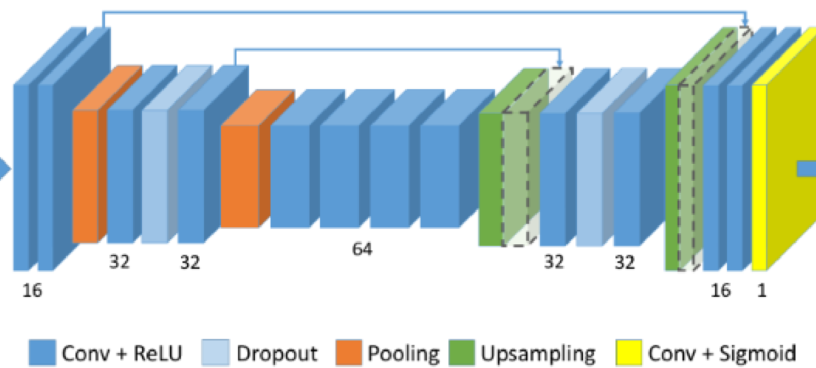
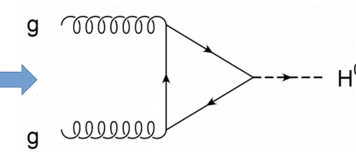
Neural-network Topology Bayesian Optimization for FPGA implementation



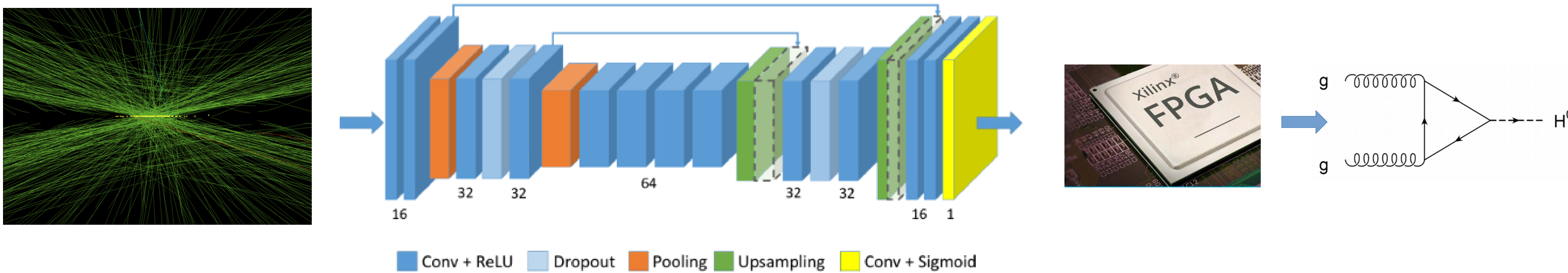




Frédéric Magniette
Journées online inter-réseaux 2019

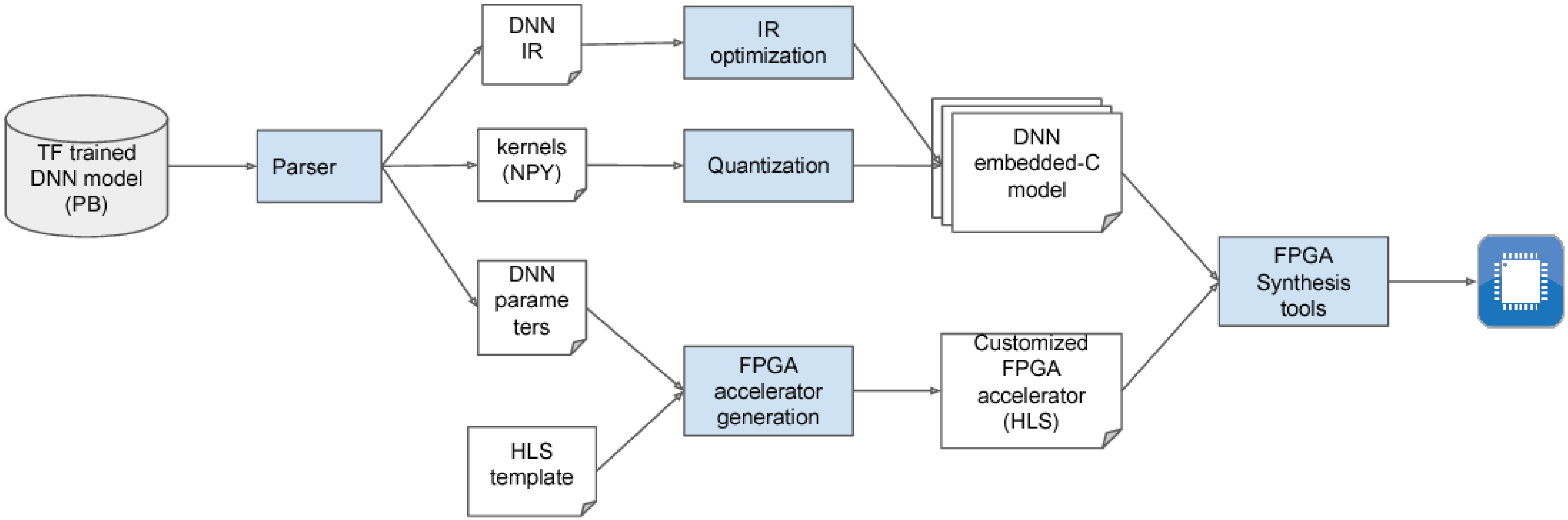
Laboratoire Leprince-Ringuet

Introduction



- Pileup → complicated Trigger algorithm
- Evaluating particle ID and energy
- Hard to implement in FPGA (loops, maths...)
- Complicated algorithms can be replaced by NN
  - Trained on simulations
  - Implemented on FPGA

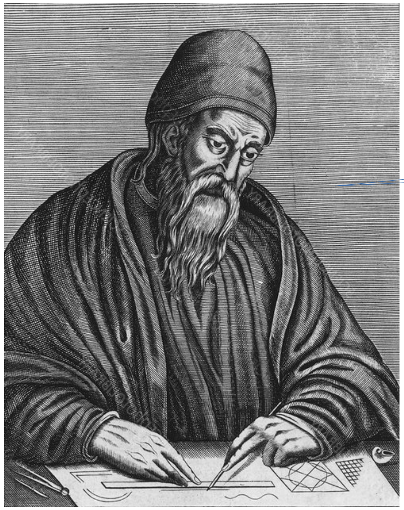
DNN in FPGA



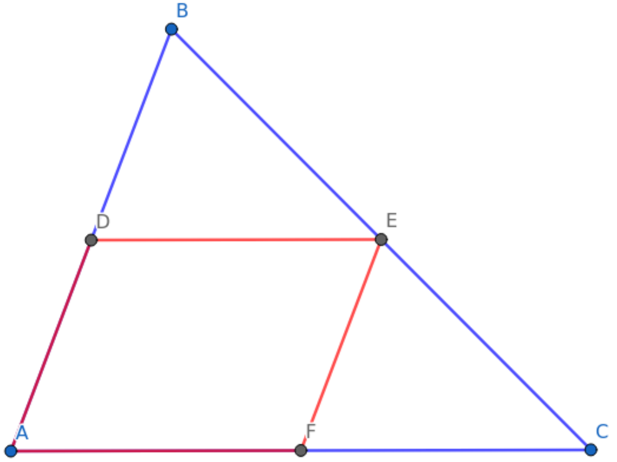
- Easy implementation : dedicated tools
- Conversion software from model to hardware
- Using dedicated functionnal block (DSP, dedicated computation units)
- Key point : precision

How to optimize resources to get the best precision ?

Optimization : an easy question… a hard answer

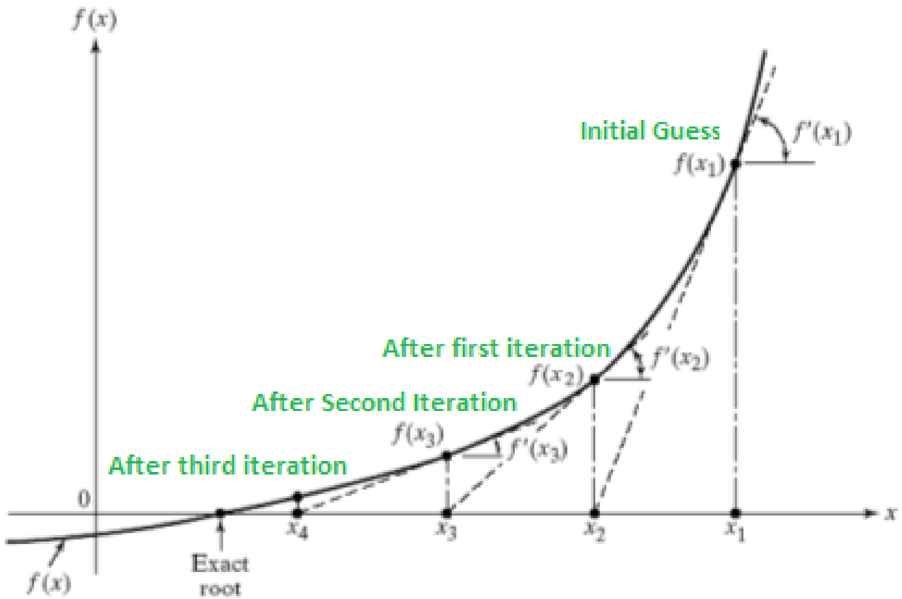


$$argmin(f(\mathbf{x})) = \{\,\mathbf{y}\,|\,\forall \mathbf{x}, f(\mathbf{y}) \leq f(\mathbf{x})\}$$



- First optimization problem in Euclid Elements (300BC) : max surface parallelogram inscribed in triangle
- Easy general formulation
- First general answer with differential calculus 2000 years later
  - 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
- Only local derivatives required
- But : Hessian matrix computationally very expensive

→ need a first order solution

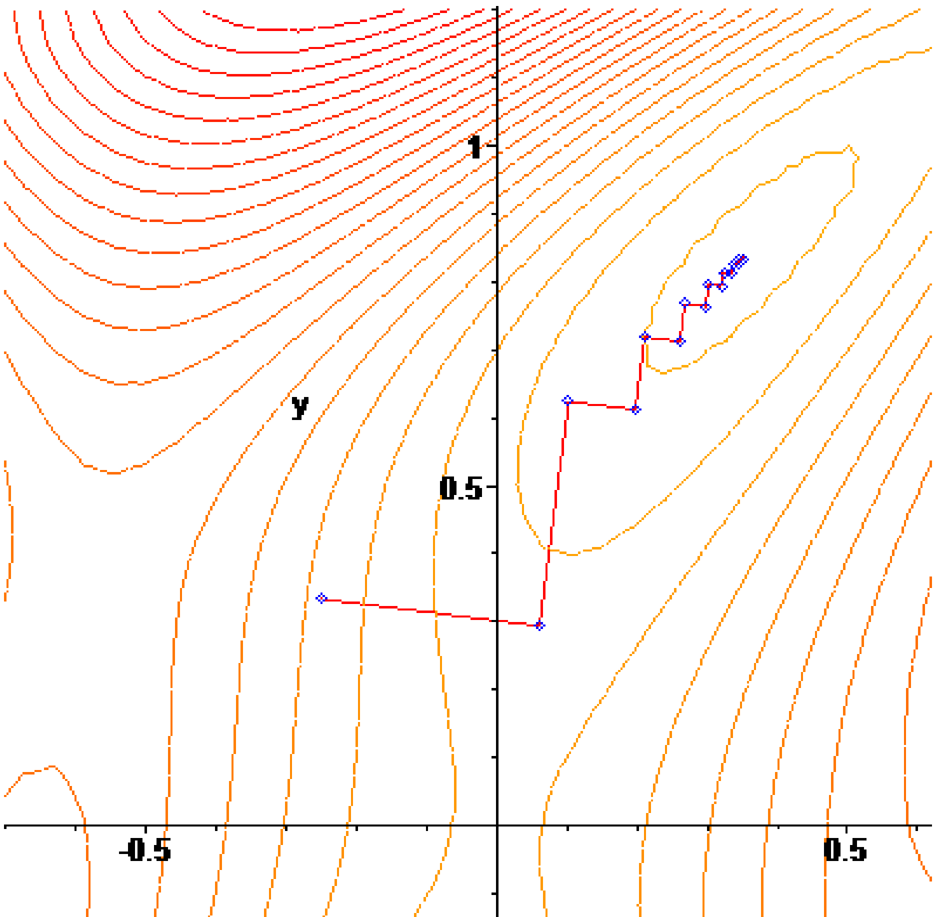
Optimization as a Blind Walk

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

Gradient Descent

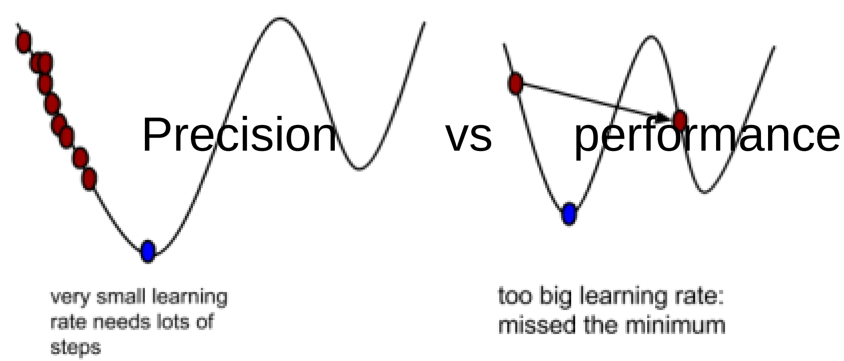
First Idea : following the slope by calculating the gradient vector

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

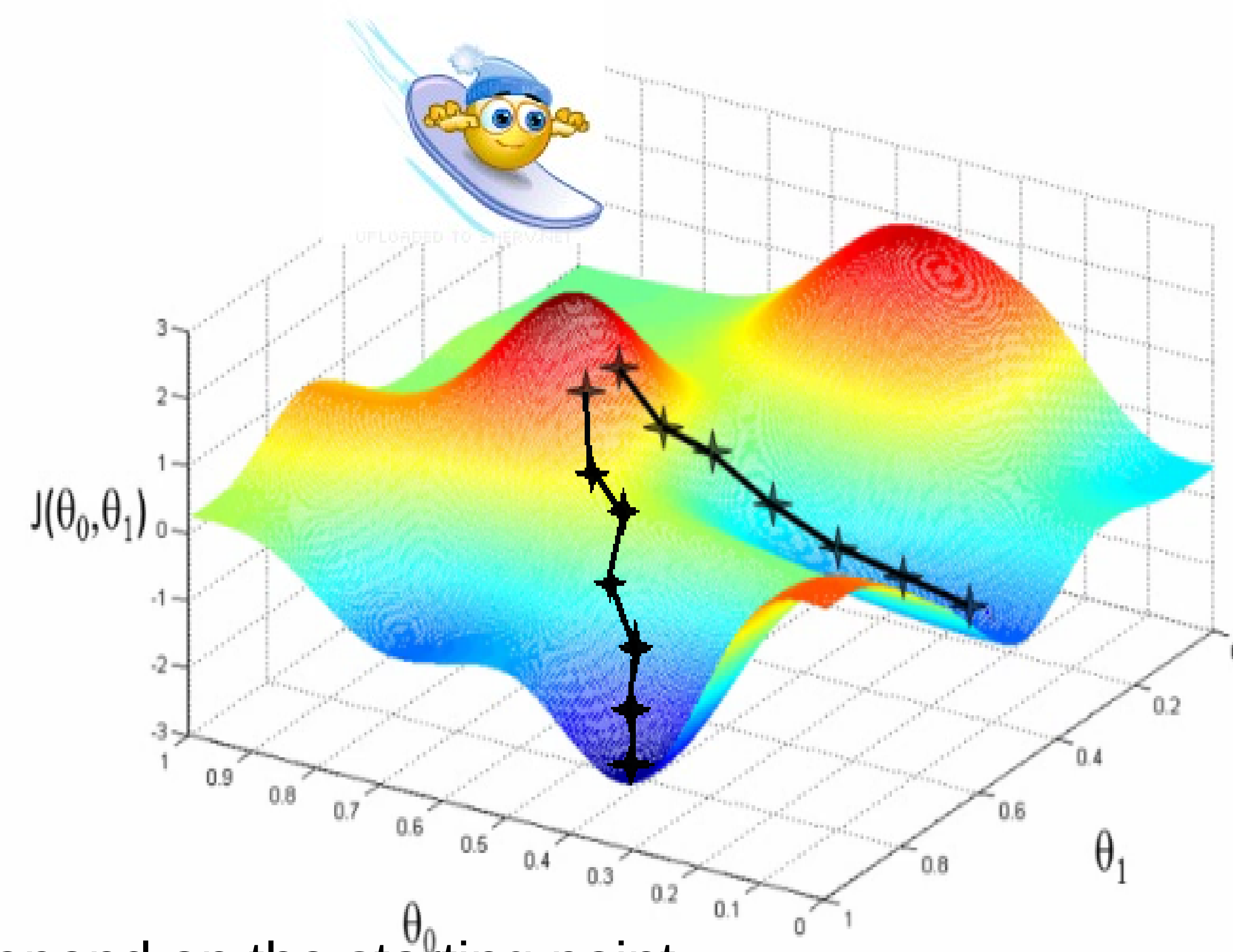


$$\Theta = \Theta - \alpha \nabla J(\Theta)$$

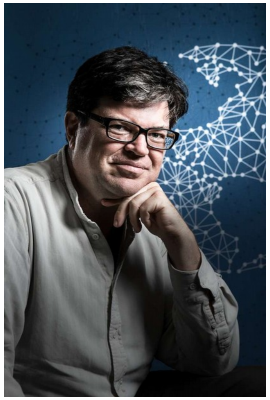
α : step size



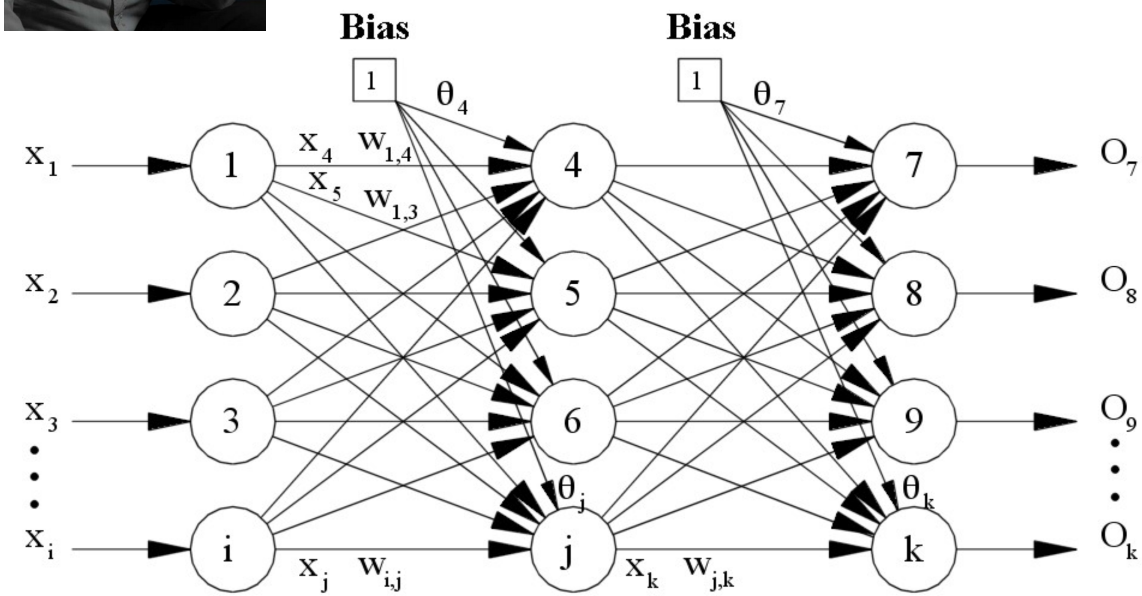
Gradient Descent & Convexity

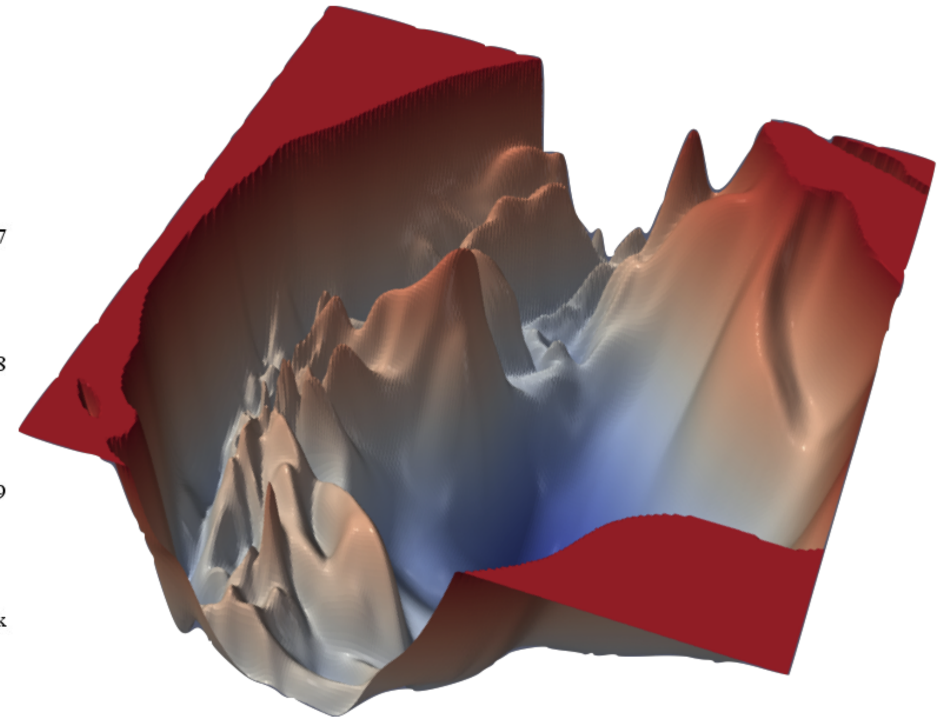


- Depend on the starting point
  → require convexity (unique minima)
- Practical solution : multiple random starts



Neural Networks





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

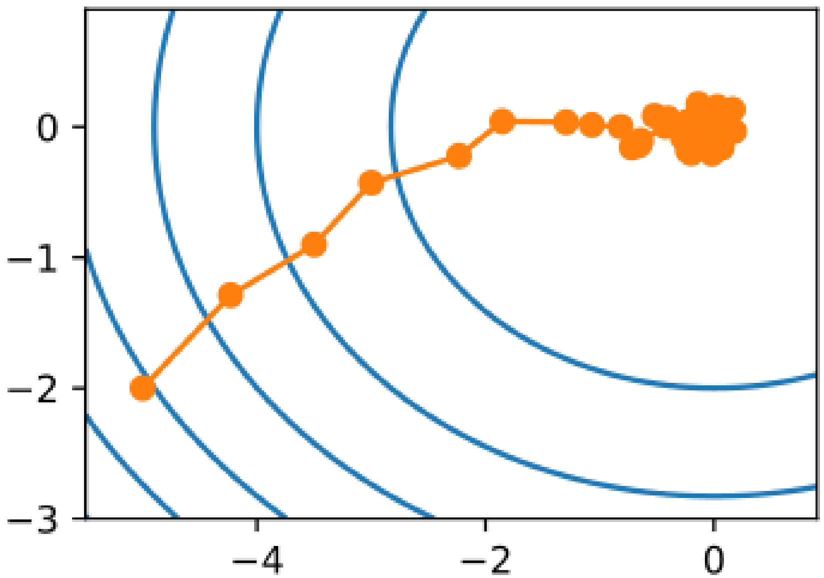
- Learn an algorithm by labelled data
- Invented by Yann Lecun
- Optimization space w_{ij} & θ_i named globally θ
- Function to optimize : loss function $L(\theta)$
- Searching for a good minimum in the loss function

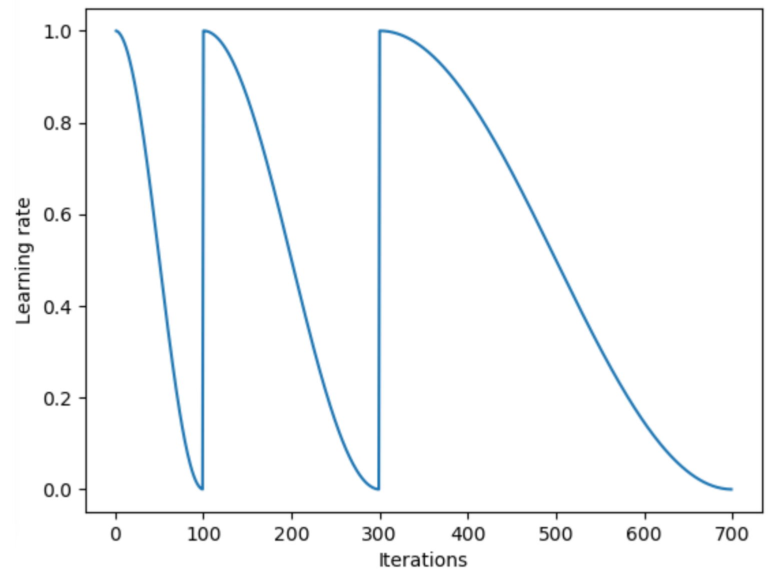
Why does it work ?

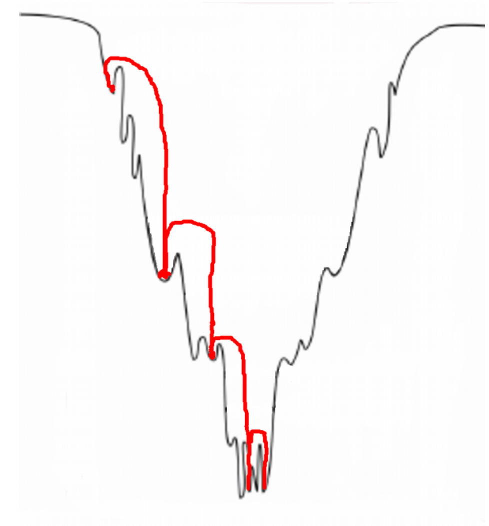
- perceptron ⟷ spherical spin-glass model
- theoritical results reuse
  - $\#min_{loc} \propto e^{dim}$
  - $\#Bad\_min_{loc} \propto e^{-dim}$
  - Good local minimum : $loss(min_{loc}) - loss(min_{glob}) \leq \epsilon$
  - Funnel global shape
- Global minimum is overfitting
- Deep learning (dim is big) gives better results

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

Convergence speed and avoiding local minimas





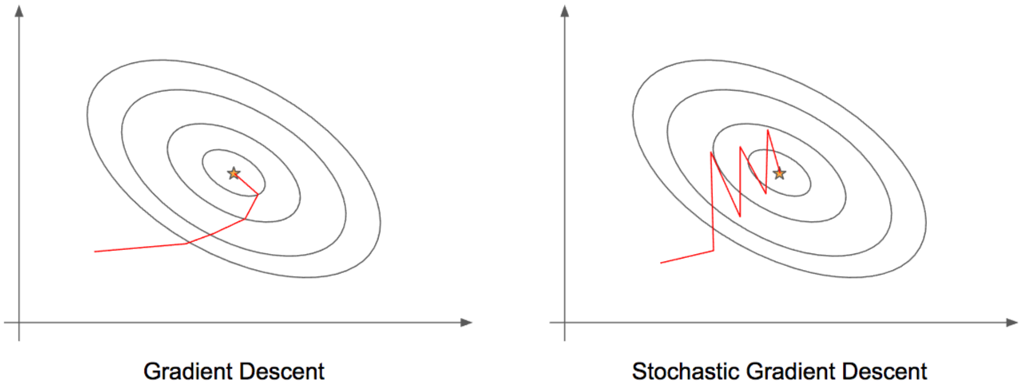


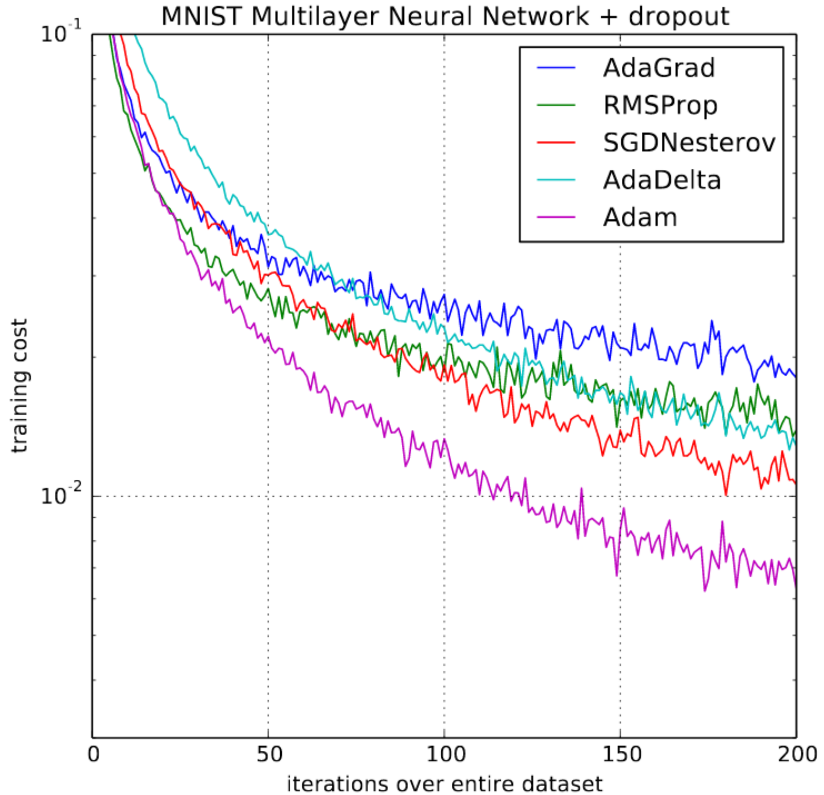
- Adaptive learning rate
  - Big step in big steep → speed up convergence
  - Smaller steps in the hole → increase precision
- Avoid bad local minimas
  - cosine annealing → restarts jump to another local minima

Smith, Cyclical learning rates for training neural networks, 2015, 1506.01186

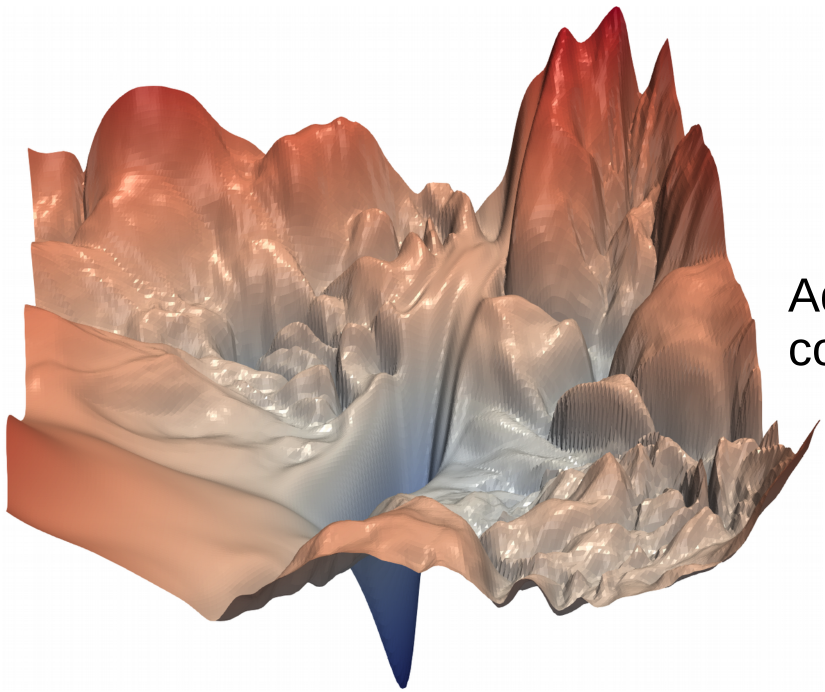
Optimizers for DNN

- Gradient descent implies huge storage of derivatives O(dimension*#inputs) for each update
- SGD slices the problem input by input : slower the convergence and add variance but 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

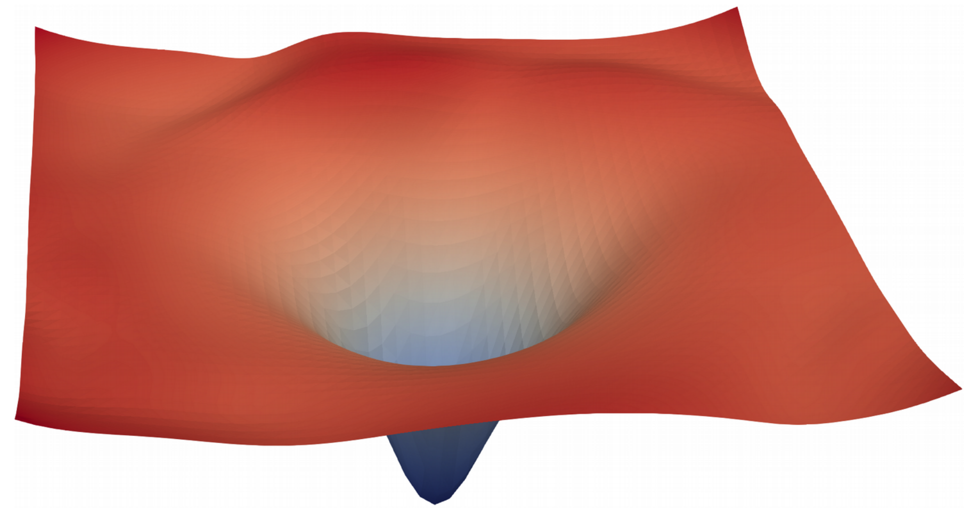




Topology Influence

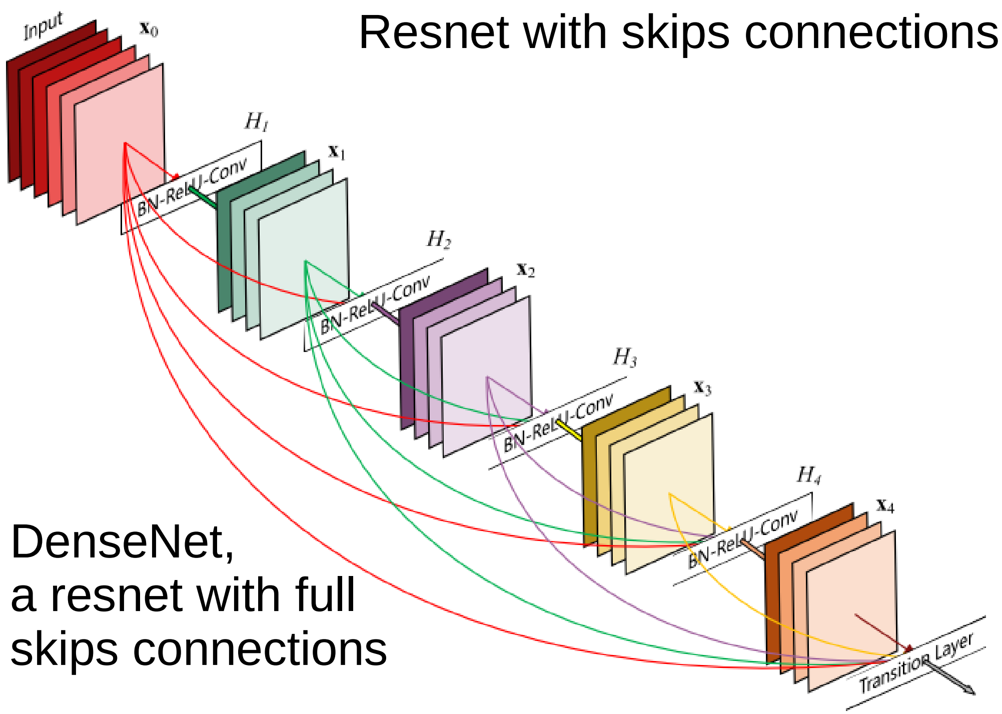


Adding skips → connections



Resnet (very deep convolutional NN)

Resnet with skips connections

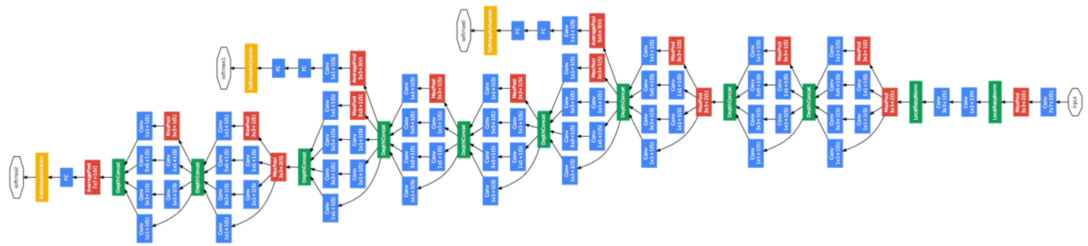


Topology influences dramatically the loss surface shape

DenseNet,
a resnet with full skips connections

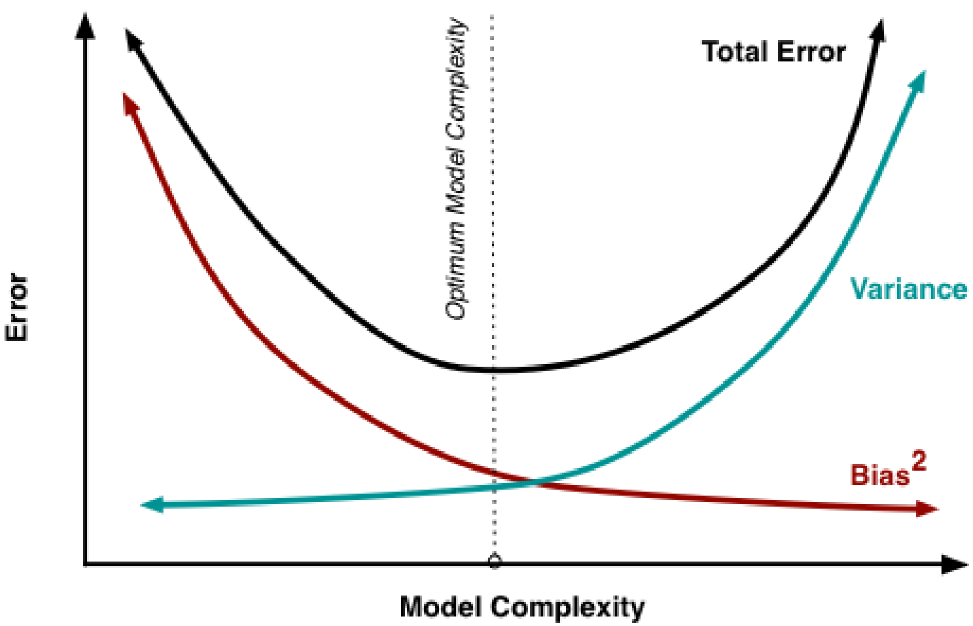
Two reasons to optimize topologies

1. Getting best distribution of neurons / convolutional kernel / pooling / skip connections for fixed resource consumption in FPGA

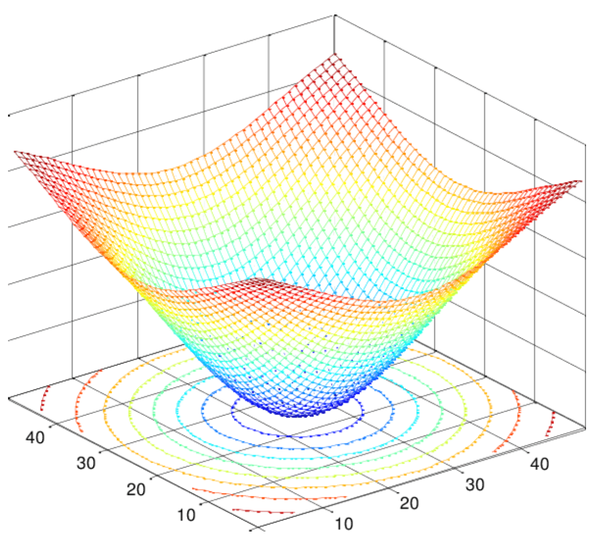


- No thumb-rule
- Often qualified as a dark-art

2. Find the bias-variance tradeoff

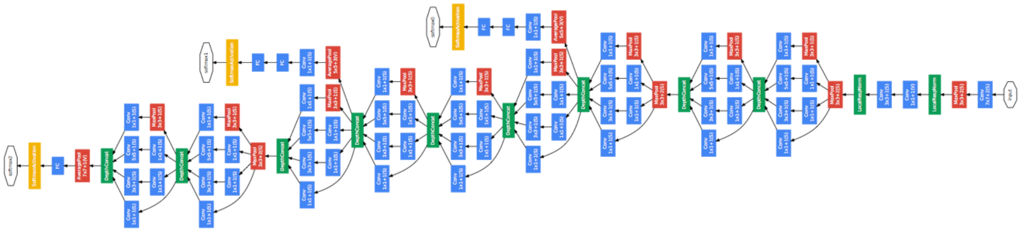


- Too simple model → fit error increased
- Too complicated model → statistical error (variance) increased
- Gives a hope for global convexity
- Help us saving resources



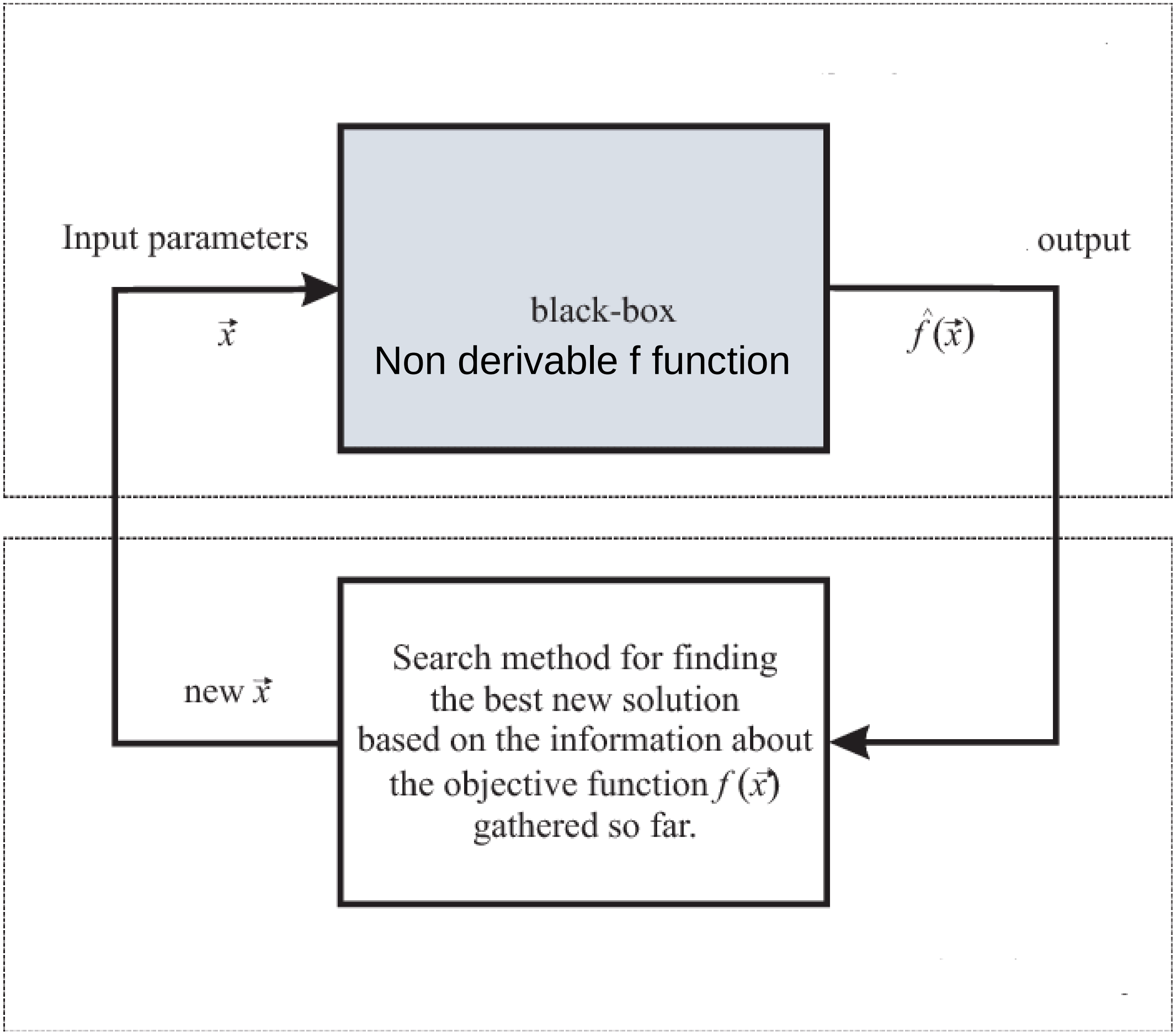
Topology Optimization

- Best topology (in terms of precision) under resource consumption constraint : again an optimization problem
- Parameter space : parametric representation of network
  - #layers #conv-layers #pool-layers
  - #layer1-size #layer2-size …
  - #conv1-size #conv2-size …
  - #pool1-size #pool2-size …

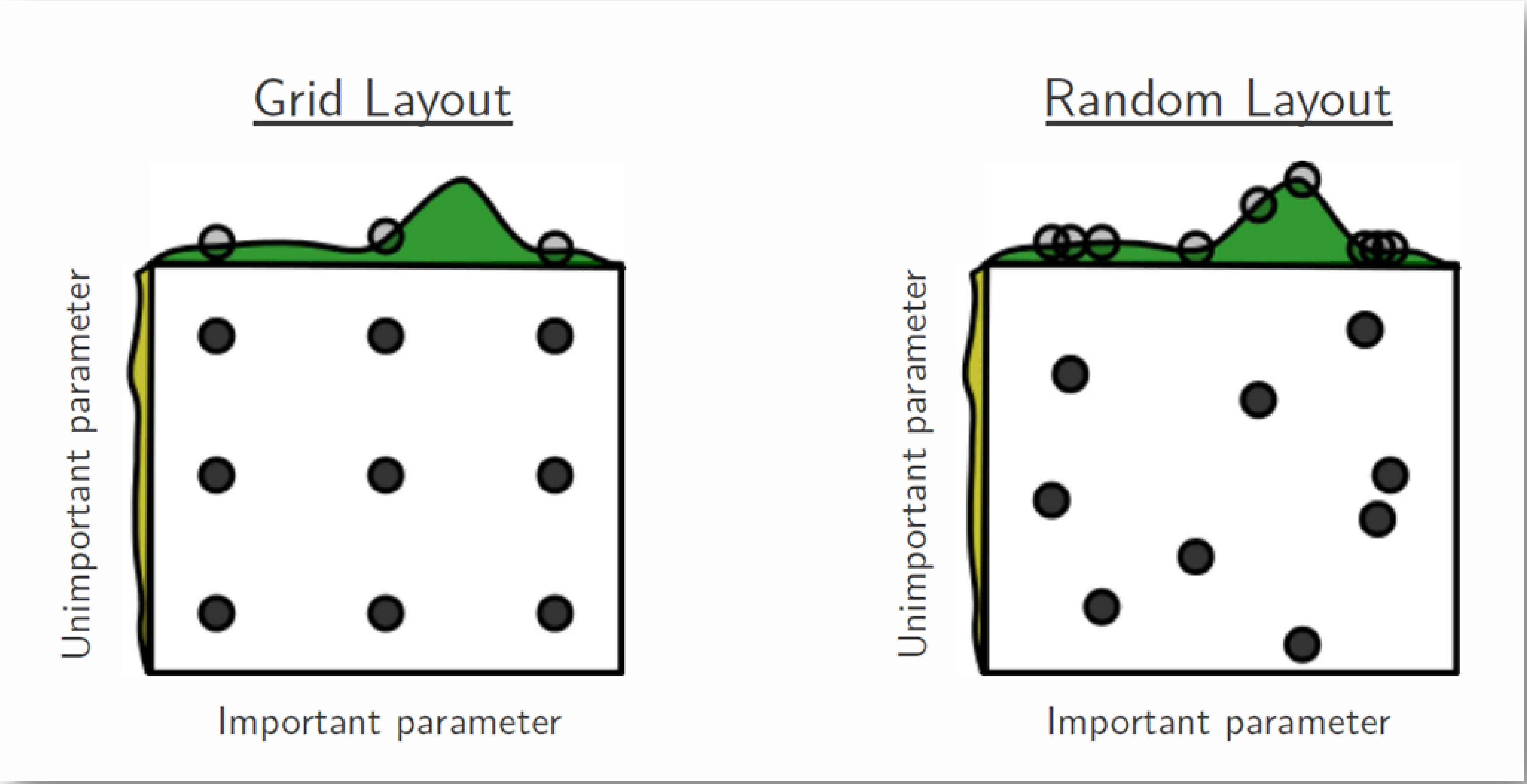


- Loss function : best precision with parametric trained network
- All right, doing gradient descent again ?
- Additionnal constraints
  - Each point is very expensive to calculate (full training)
  - The loss function is not derivable (even numerically)

Black Box / Zero-Order Optimization



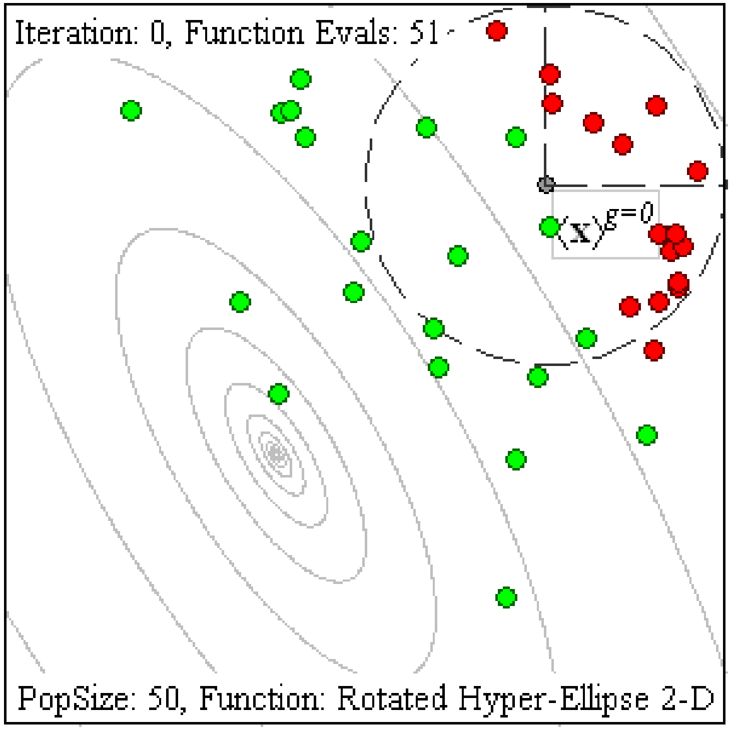
Grid and Random Search



Dimensionality

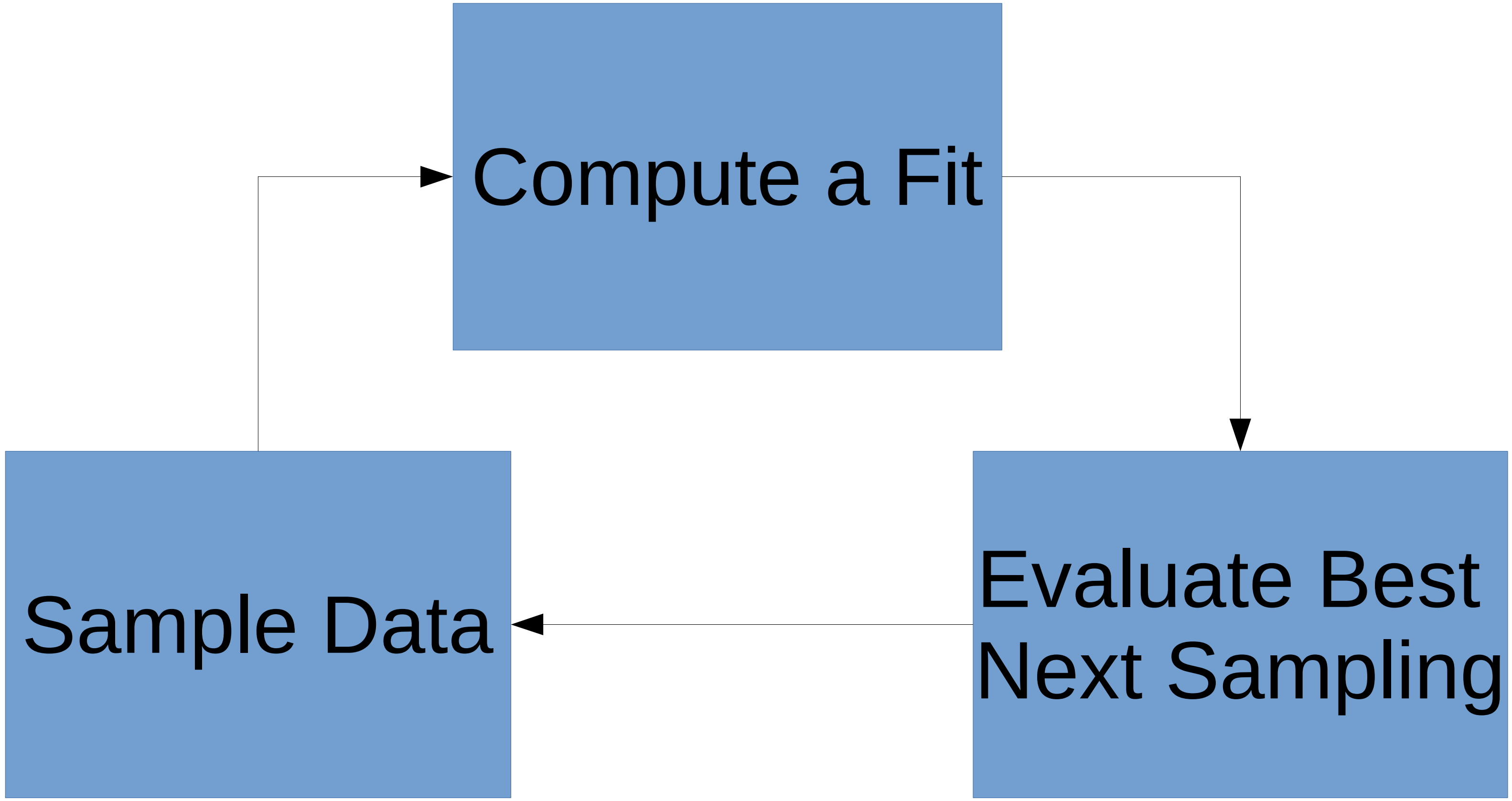
CMA-ES

- Covariance Matrix Adaptation Evolution Strategy
- Stochastic, derivative-free
- Generational adaptation of a population of points
- Elimination of worst point → covariance matrix estimation
- Quasi-newton method (approximation of Hessian)
- Very efficient if function is cheap to compute O(dim^2)



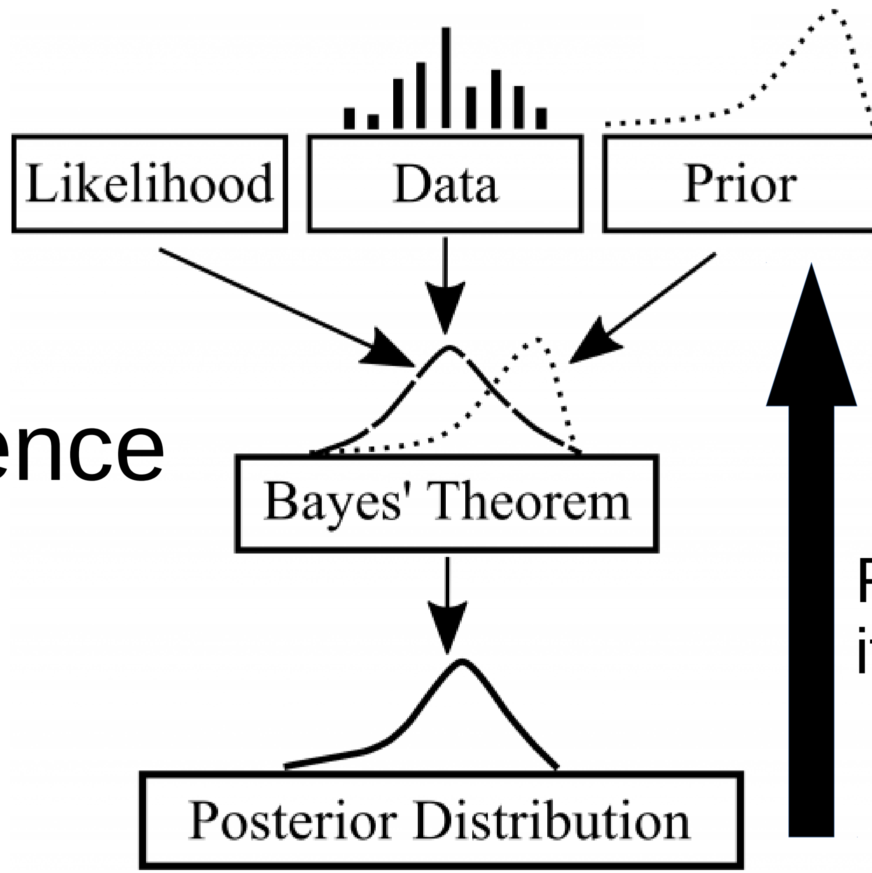
Hansen & Ostermeier, Completely Derandomized Self-Adaptation in Evolution Strategies, 2001

Data-driven Sampling



Best algorithm : Bayesian Optimization

Bayesian Inference



Model inference from data

Recurrence if new data

Model Plausibility → posterior

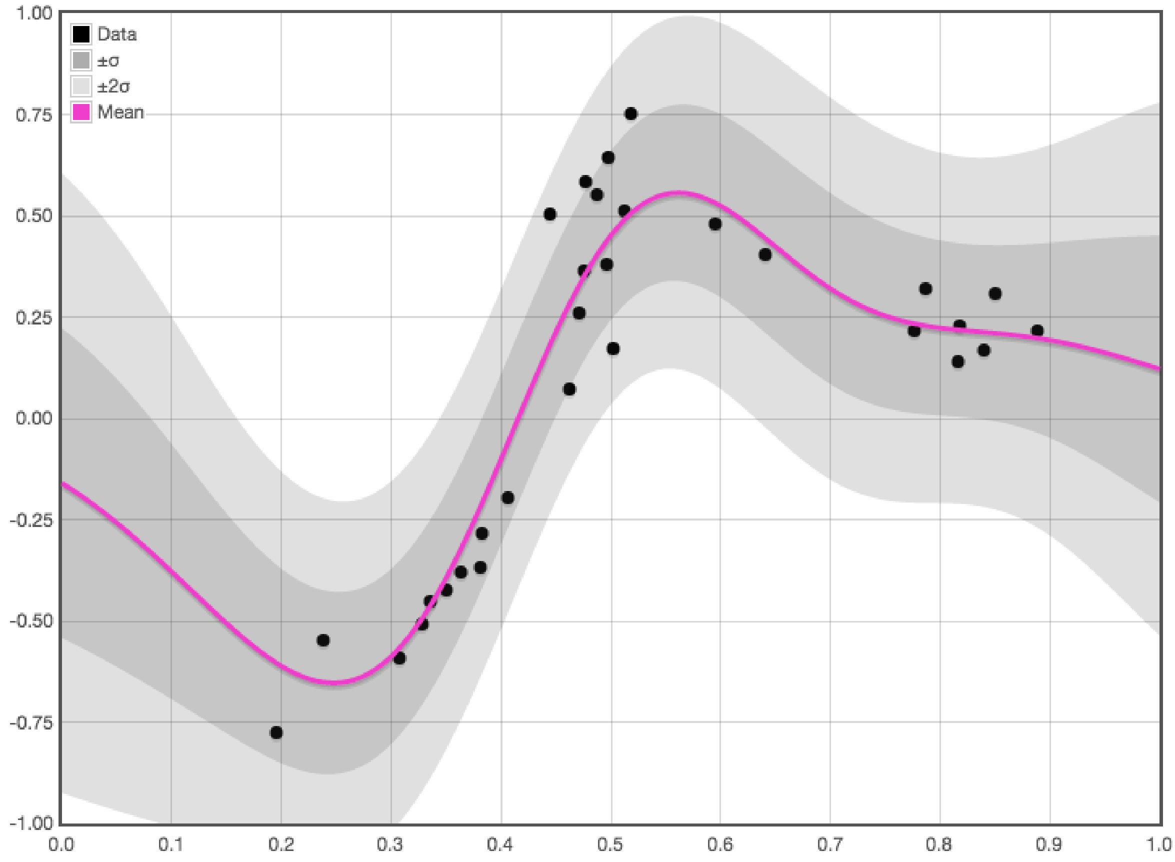
Data Likelihood

Prior

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

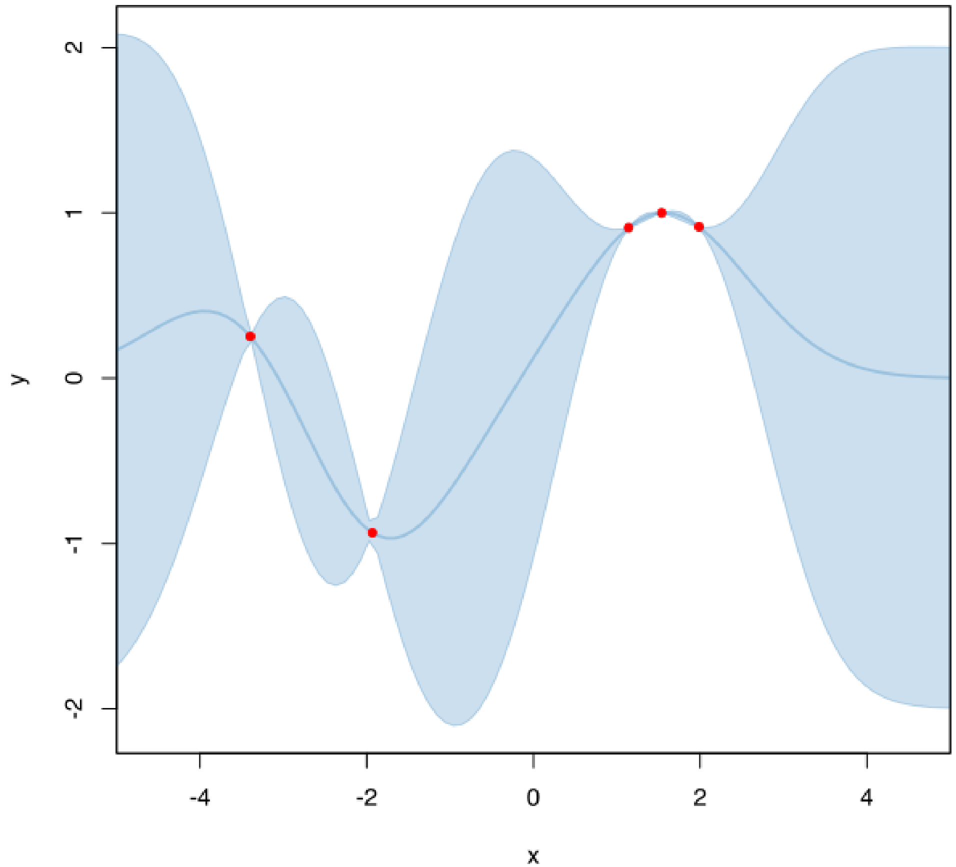
Normalization

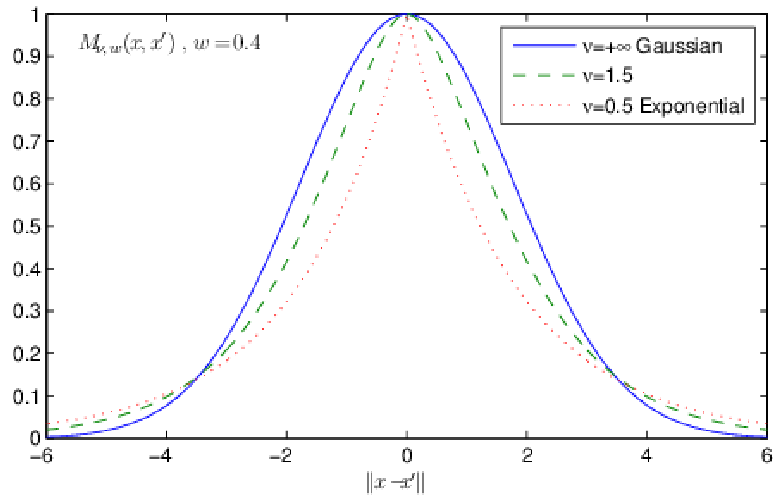
Gaussian Process



- Infinite extension of multi-variate Gaussian
- Arbitrary dimension
- Defined by mean(**x**) and sigma(**x**)

Gaussian Process Regression





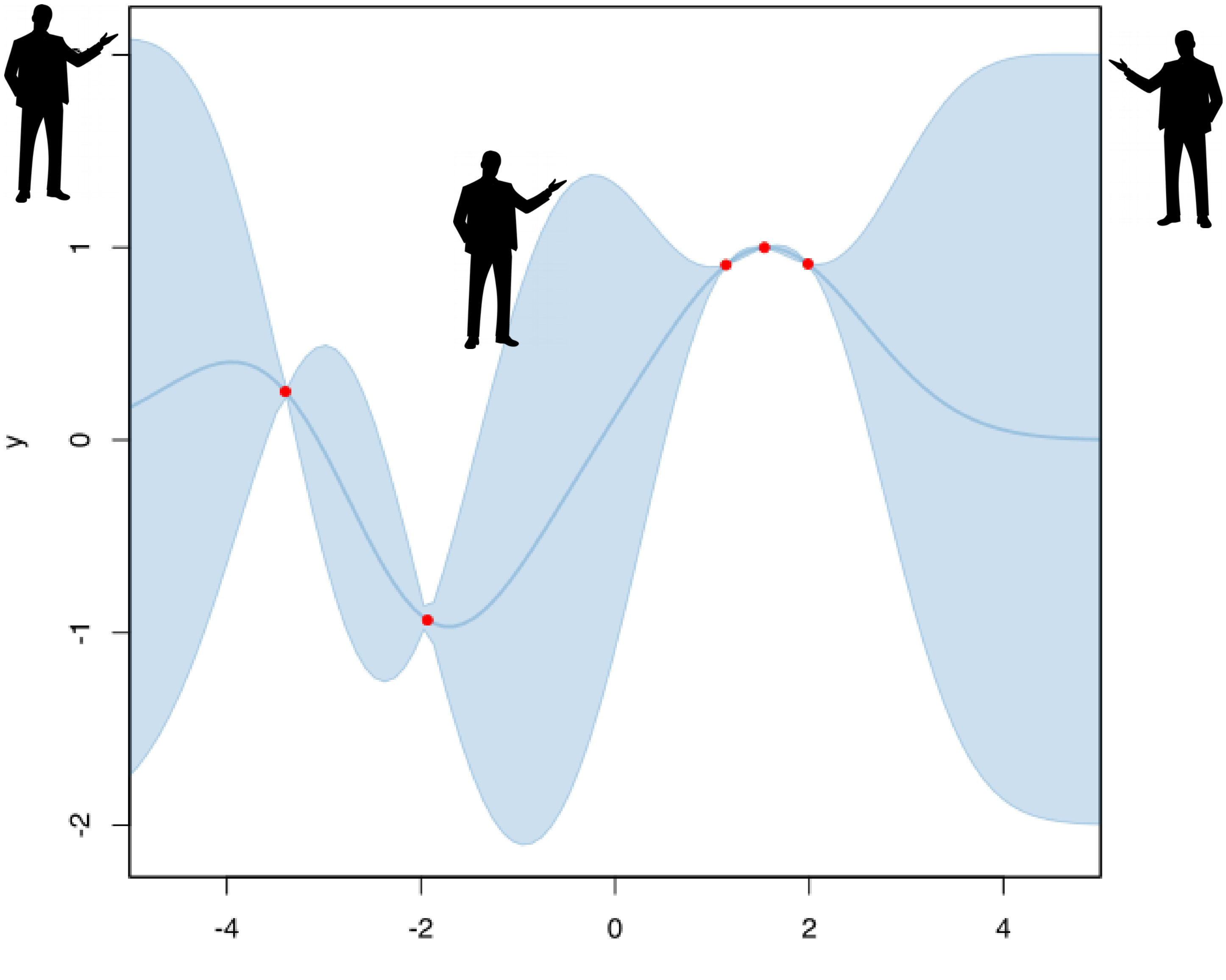
Matérn stationary covariance kernel

$$k(x_i,x_j)=\sigma^2\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\sqrt{2\nu}\frac{d(x_i,x_j)}{l}\right)^{\nu}K_\nu\left(\sqrt{2\nu}\frac{d(x_i,x_j)}{l}\right)$$

Bertil Matérn, Spatial Variation, 1960

- Variance is a function of the distance
- Possible to add noise regression
- Good representation of the so-far collected data

Where to search ? Promising points



Can we express this as a function ?

Acquisition functions

- Upper Confidence Bound (UCB)

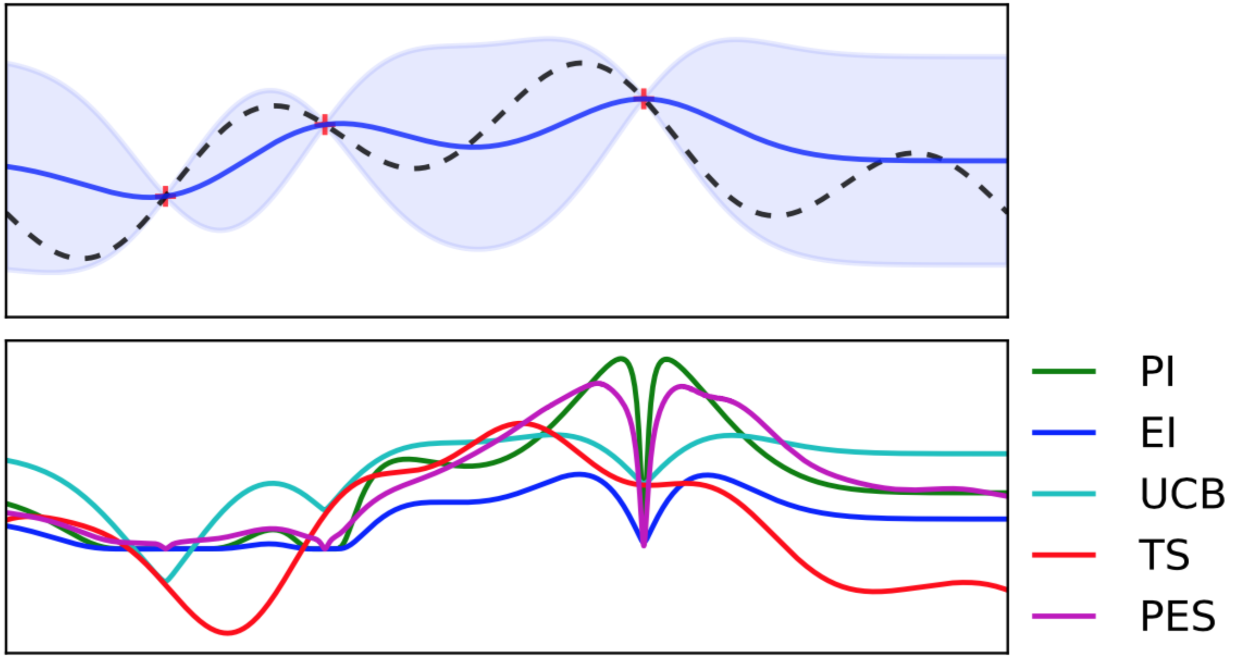
$$A(x) = \pm\mu(x) + \kappa\sigma(x)$$

- Esperance of Improvement (EI or EOI)

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

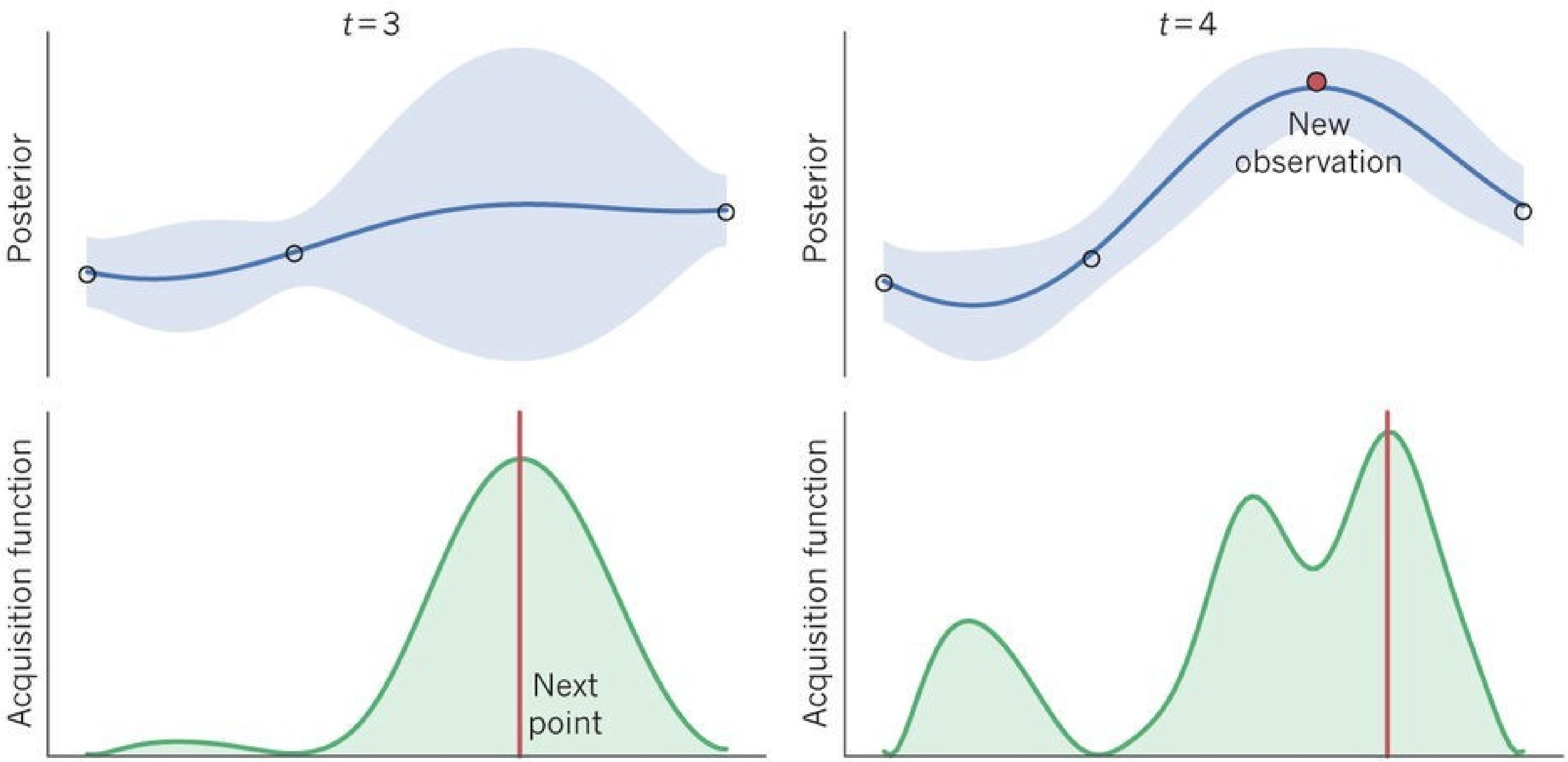
- Probability of Improvment (PI or POI)
- Entropy search (PES)
- Thomson sampling (TS)

- Easy to compute
- Rely only on Gaussian process



Bayesian optimization

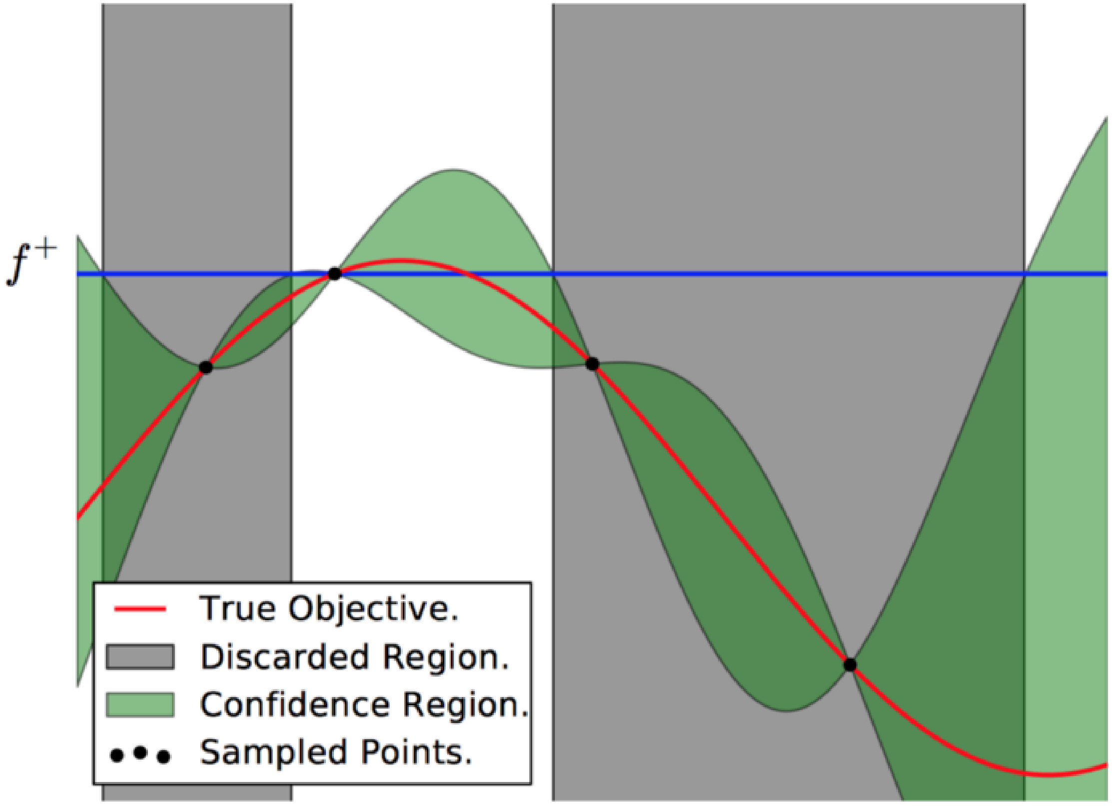
Jonas Mockus, Bayesian Approach to Global Optimization, 1989



$\kappa = 4$

Exploitation vs Exploration

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

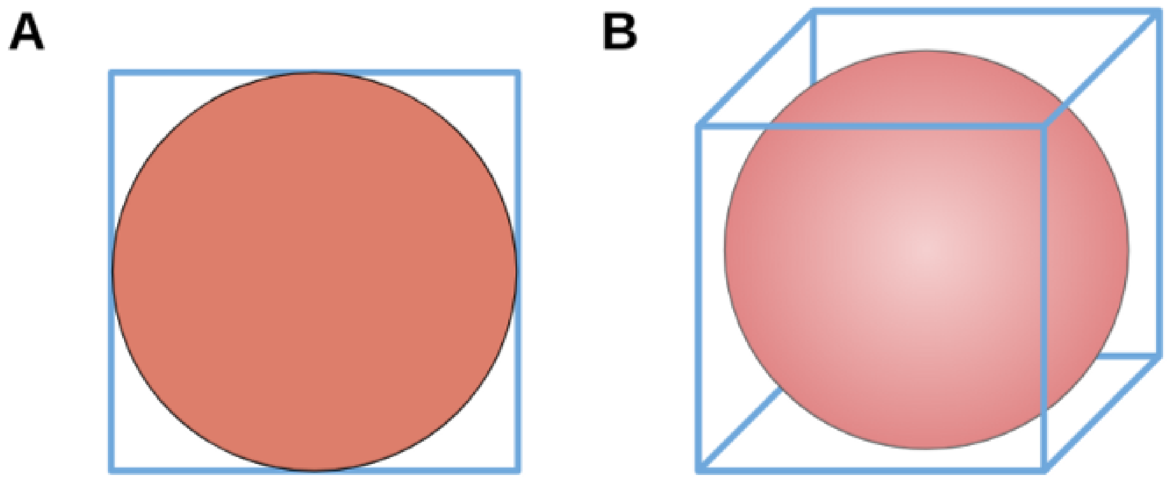


Computational performance
vs
Exhaustivity
(local extremum)

Question : How to optimize hyper-parameters of hyper-parameter optimizer ?

$\kappa = 1$

Limitation: Curse of Dimensionality

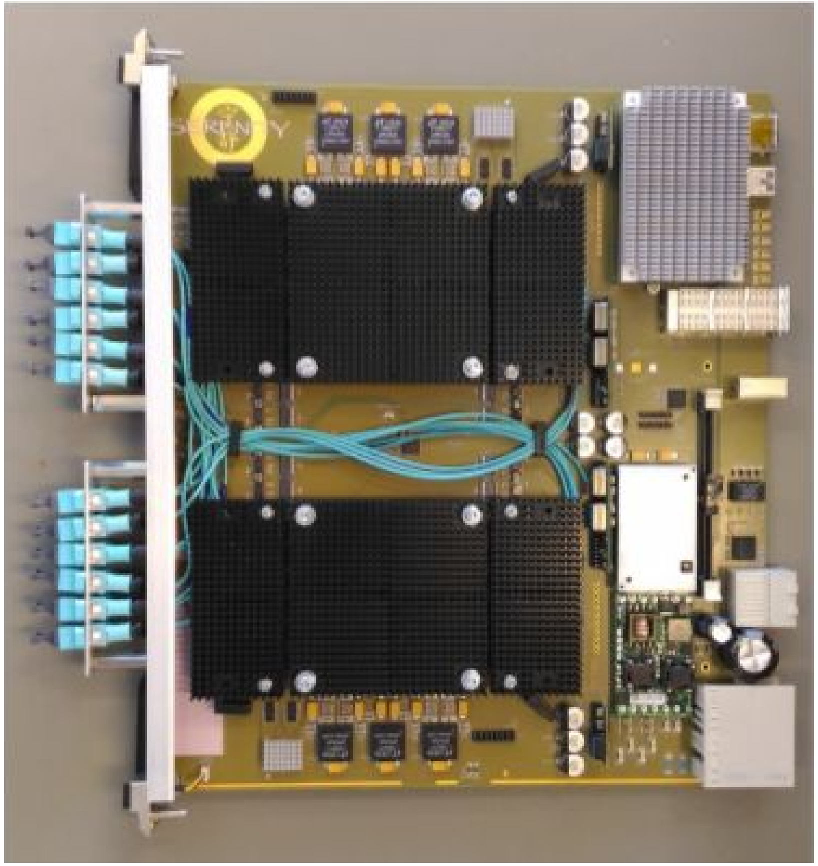


$$\frac{V_{hypersphere}}{V_{hypercube}} = \frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)} \to 0 \text{ when } d \to \infty$$

- 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)

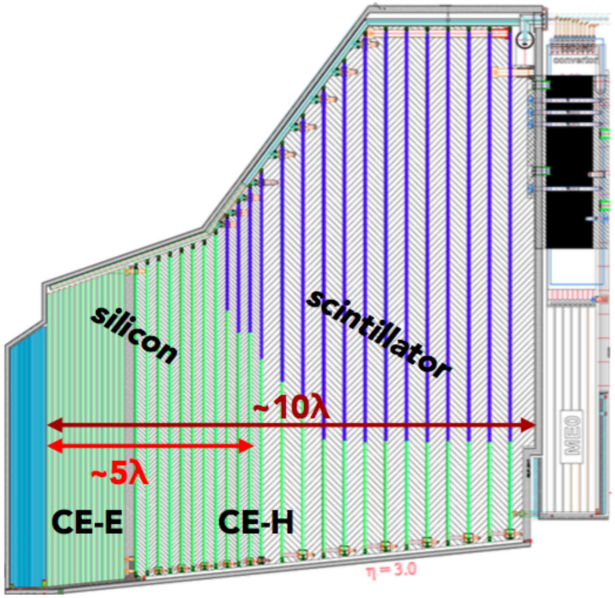
HGCal Trigger

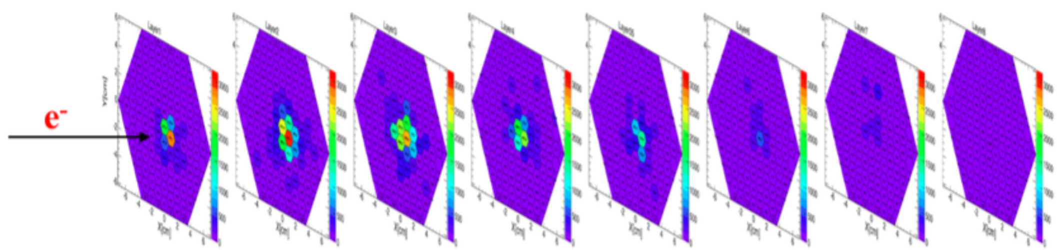
- Serenity platform
  - Generic platform developped by Imperial College
  - Data aggregation on optical links
  - Interconnection between different layers of boards → distributed algorithm
  - Implement clustering algorithm with particle ID and energy evaluation
  - Limited amount of resources and latency → need for good approximation



HGCal Test Case

- Particle ID : pion vs electron shower classification
- Samples simulated by CMSSoftware on HGCal model
- Output : binary choice
- Neural networks
  - Multi-layer perceptrons (max 15 layers)
  - Limited global number of neurons
- Bayesian optimization on #neurons per layer space

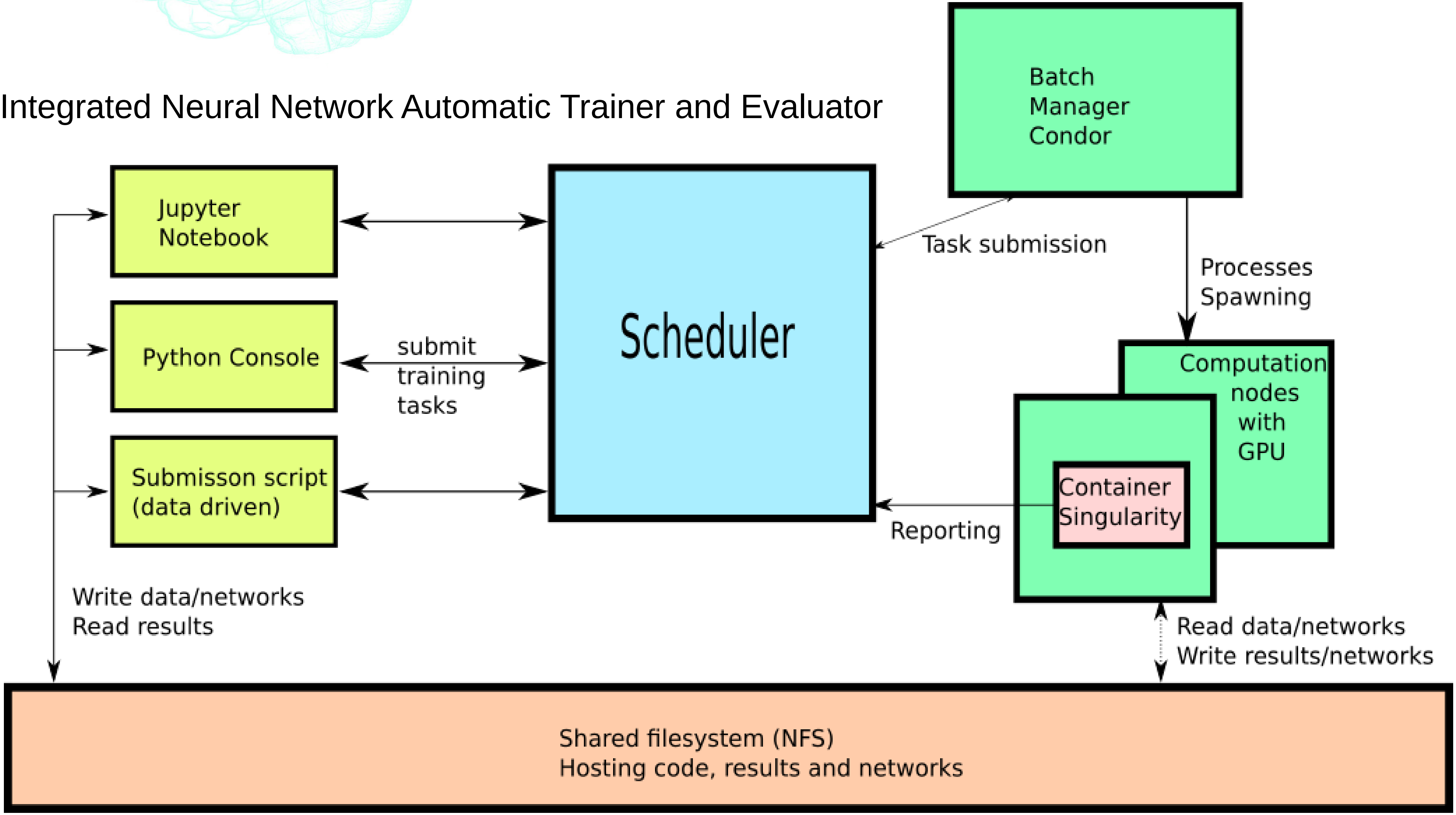




Innate

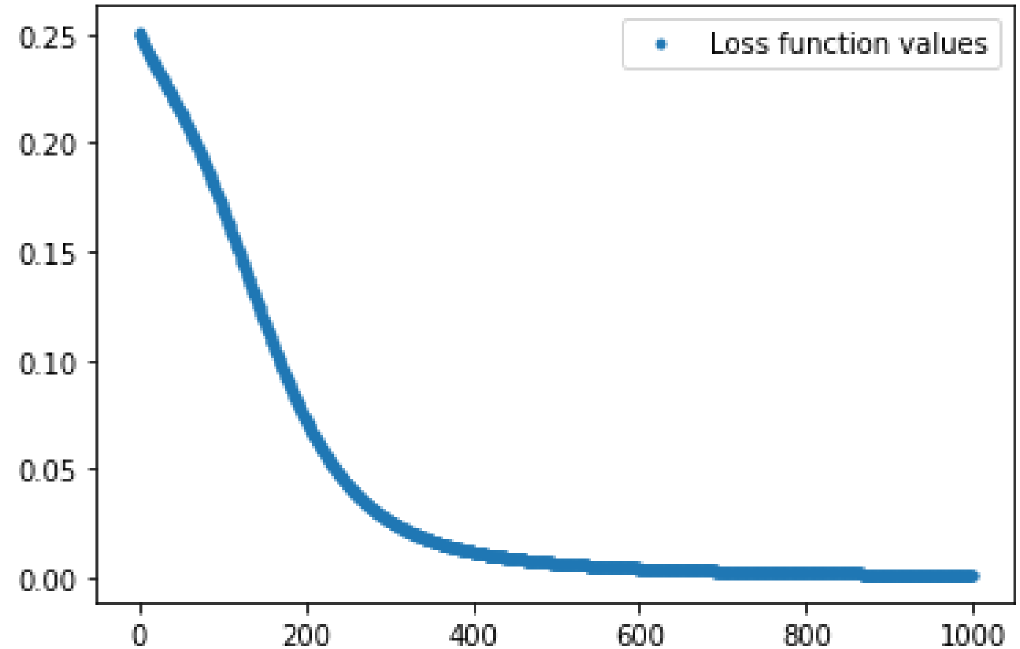
- Runtime encapsulate all algorithmic complexity
→ ease of development
- Based on Keras & Tensorflow

Integrated Neural Network Automatic Trainer and Evaluator



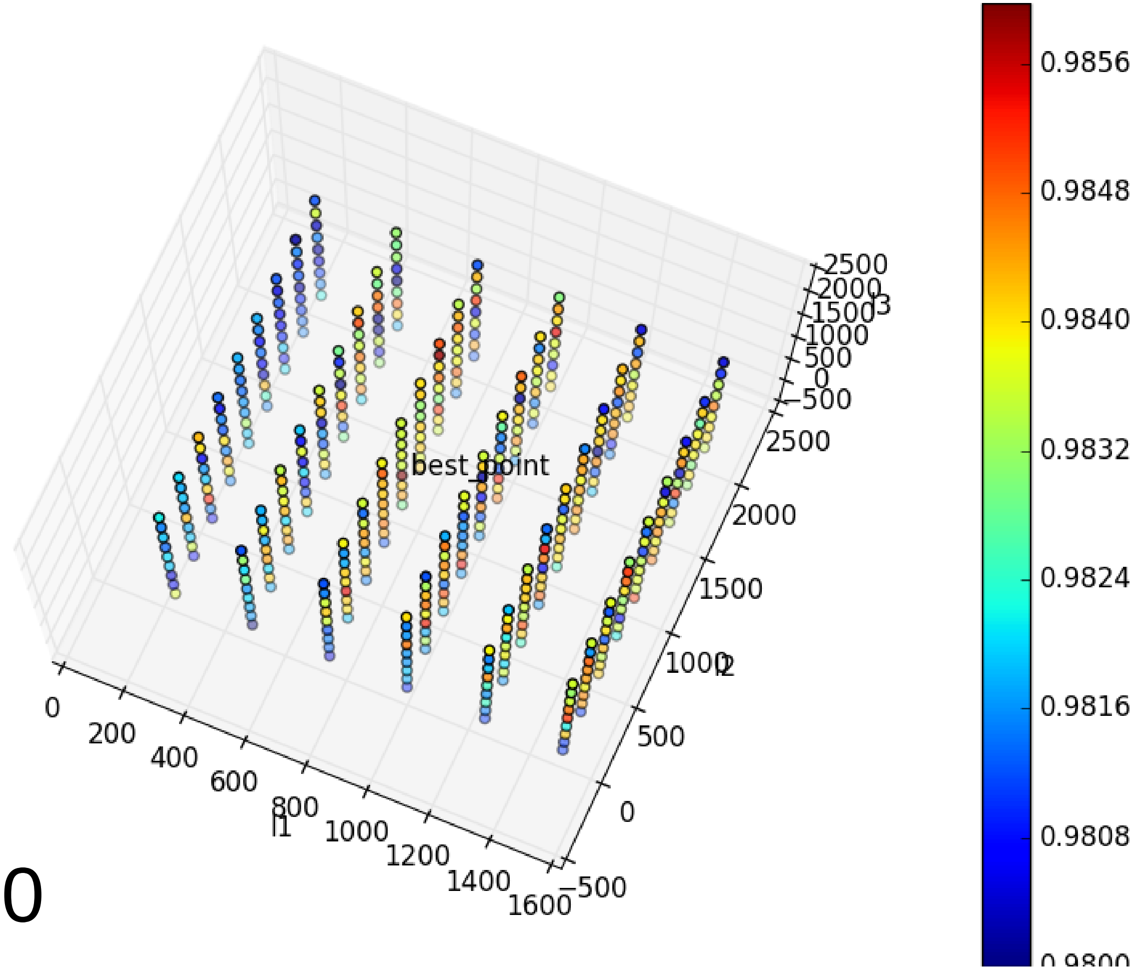
Innate API

```
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)
```



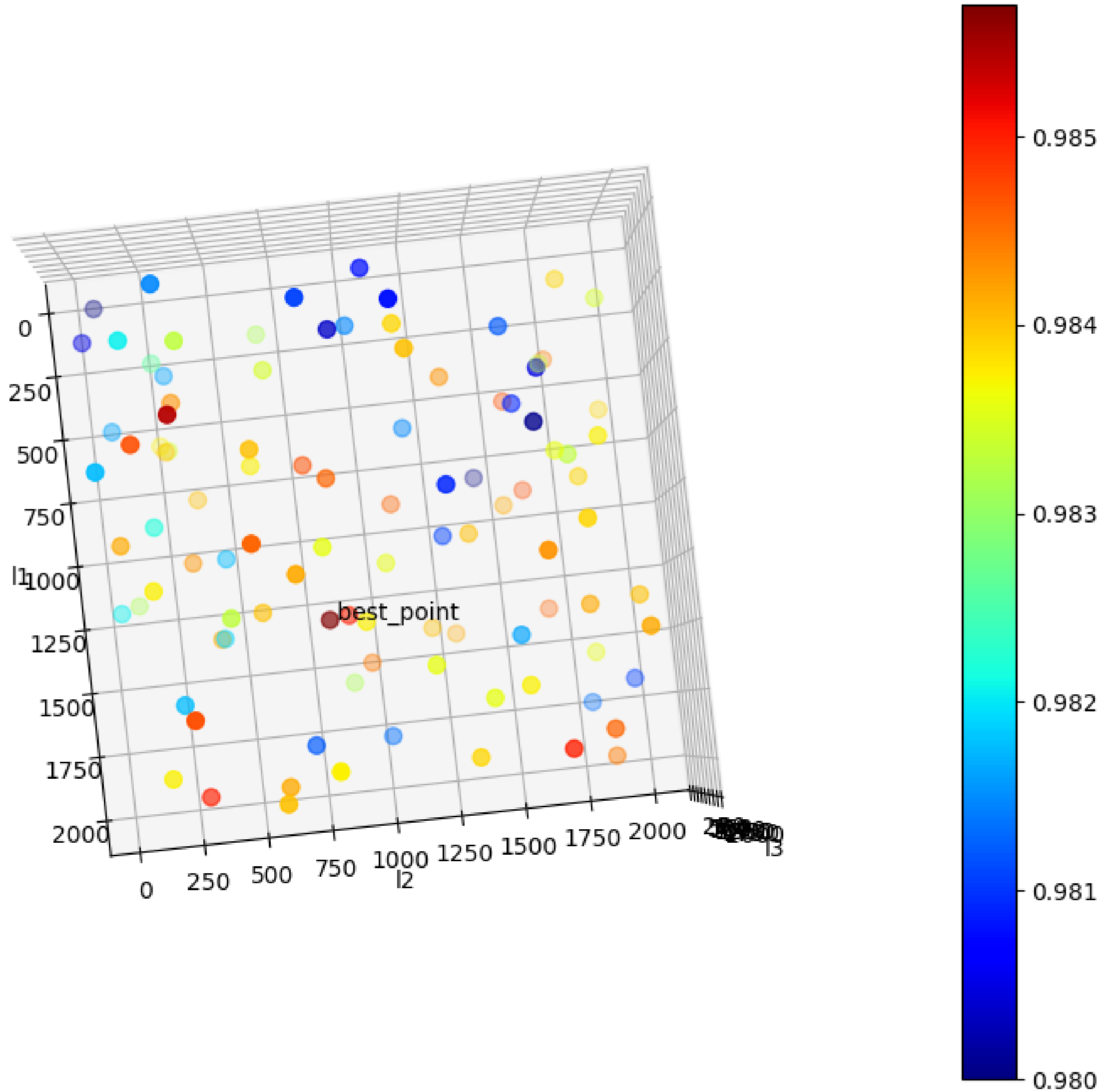
Grid search topology exploration

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



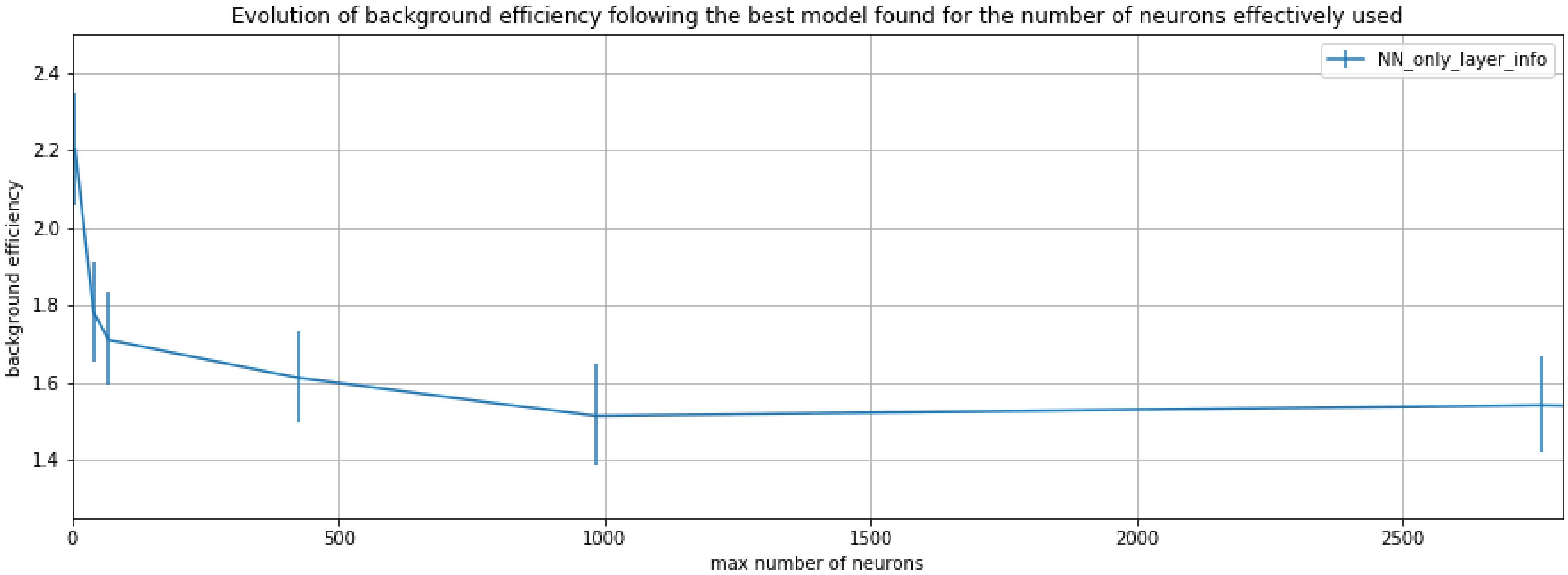
Bayesian Optimization

- 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



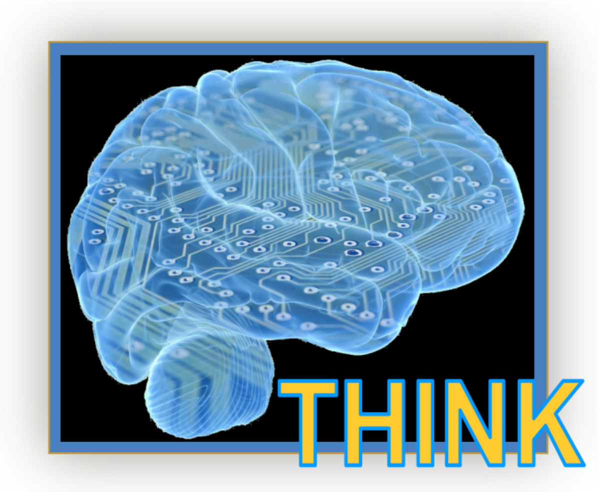
Global Performance over Resource Avaibility

- Taking different max size and searching for best size
- Max 15 layers



Best network : 38x174x302x4x492x11x1

Perspectives



- Add PyTorch to Innate
  - All exciting new technos are there !
- Try different flavour of neural network
  - Graph convolution (non euclidian)
  - Study portability on FPGA
- Implement Parallel Bayesian Optimization
- Participate to the « Think IN2P3 project »
- Implement an optimal DNN for level 1 trigger in CMS HGCal
- Keep the trend in a VERY prolific domain !!