Neural network optimisation

for accuracy, speed and profit

01 Apr 2020, MISIS

Andrey Ustyuzhanin

NRU HSE

YSDA

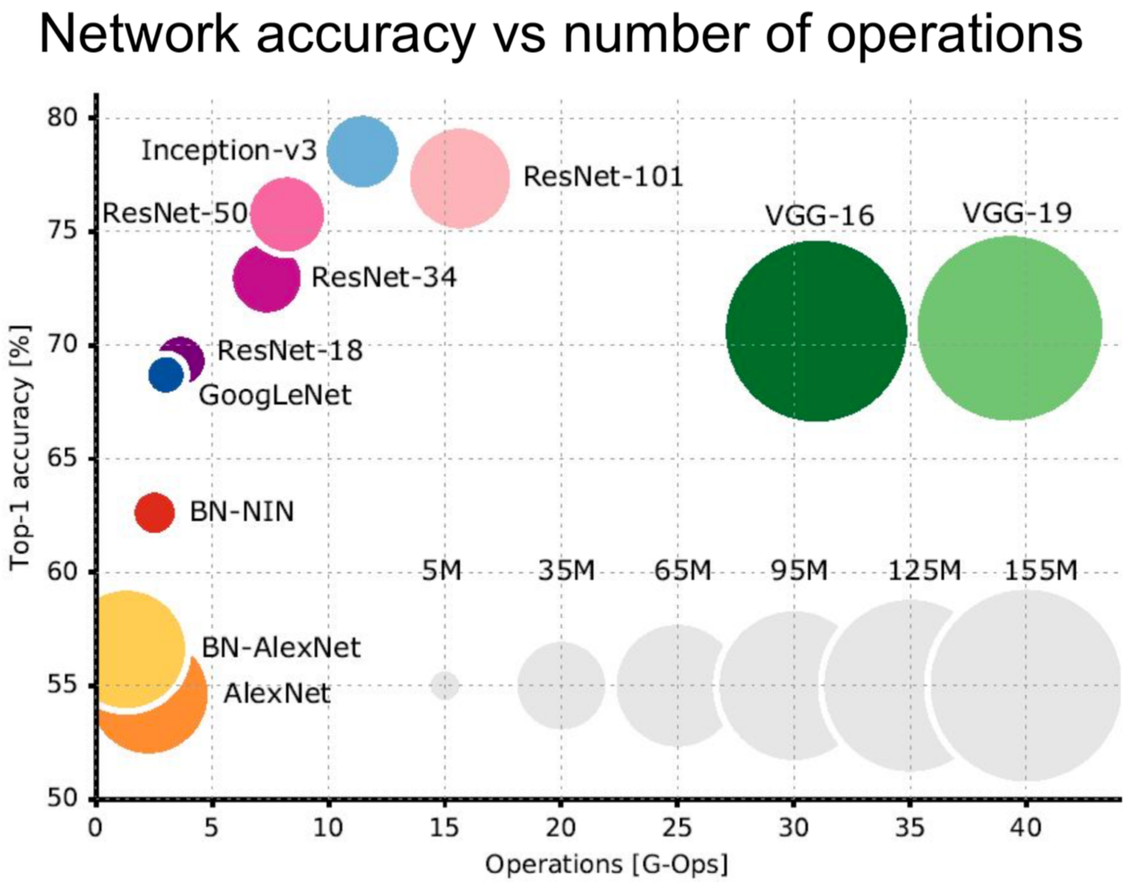
MISIS

Why bother with optimisation?

Accuracy

Compression (space, speed, bandwith, energy):

- Smart architectures (SqueezeNet, MobileNets)
- Low-rank factorization, hashing trick, …
- Pruning
- Reducing numerical precision
- Distillation



Source: Canziani, A., Paszke, A., & Culurciello, E. (2016). An Analysis of Deep Neural Network Models for Practical Applications. arXiv preprint arXiv:1605.07678v2

Outline

I. Hyper parameter optimisation

II. Network architecture search

III. Bayesian NN perspective

IV. Outlook

Quick self-intro

Head of LHCb Yandex School of Data Analysis (YSDA) team

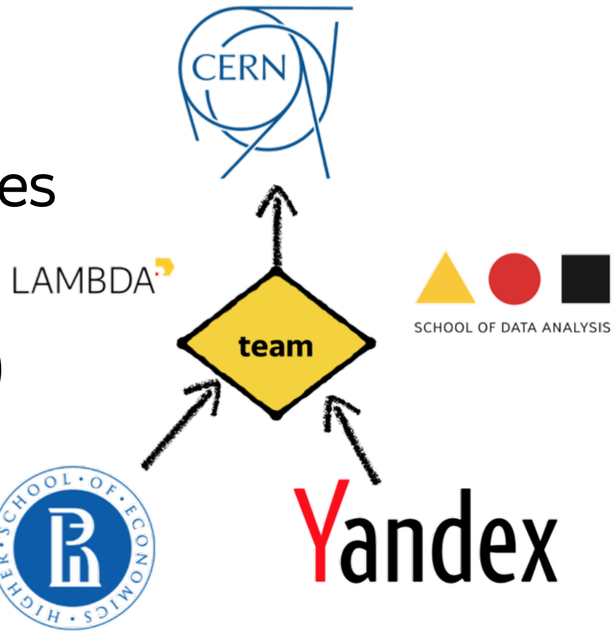
Head of Laboratory (link) of methods for Big Data Analysis at Higher School of Economics (HSE),

- Applications of Machine Learning to **natural science challenges**
- HSE has joined LHCb in 2018!

Co-organizer of Flavours of Physics Kaggle competitions (2015)

Co-organizer of TrackML challenge (2018)

Education activities (MLHEP, ML at ICL, ClermonFerrand, URL Barcelona, Coursera)



Hyper parameter optimisation

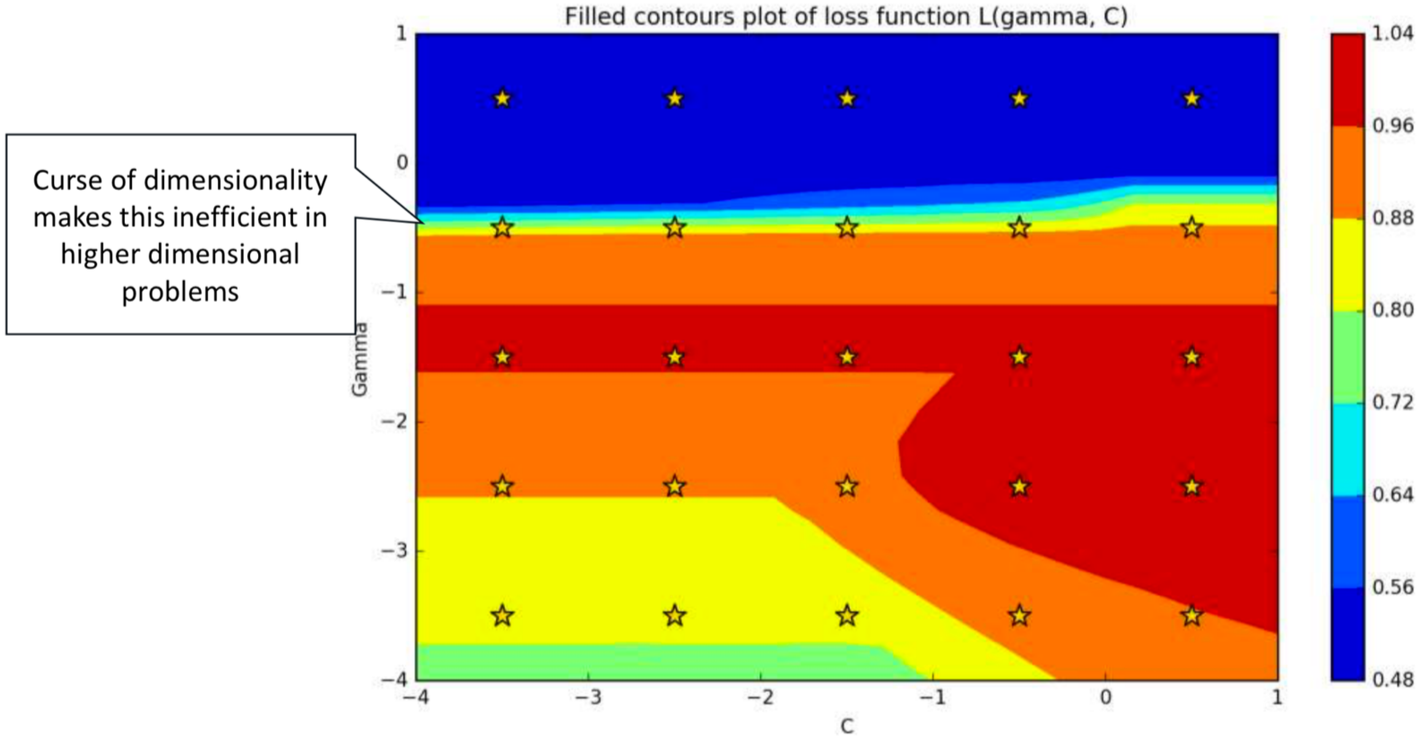
Classic Machine Learning approach

Classic ML General idea - maximum likelihood
$\theta^{\mathcal{F}}_{train} = argmax_{\theta} p(X_{train}|\theta, \mathcal{F})$ (**pointwise estimation** of trainable parameters θ at given configuration $\mathcal{F}$)

There is always set of parameters, that define F, so we can repeat argmax once again...

Scikit-Learn optimization classes

```
class sklearn.model_selection.GridSearchCV(estimator,
param_grid, scoring=None, fit_params=None, n_jobs=1,
iid=True, cv=None, pre_dispatch='2*n_jobs')
```

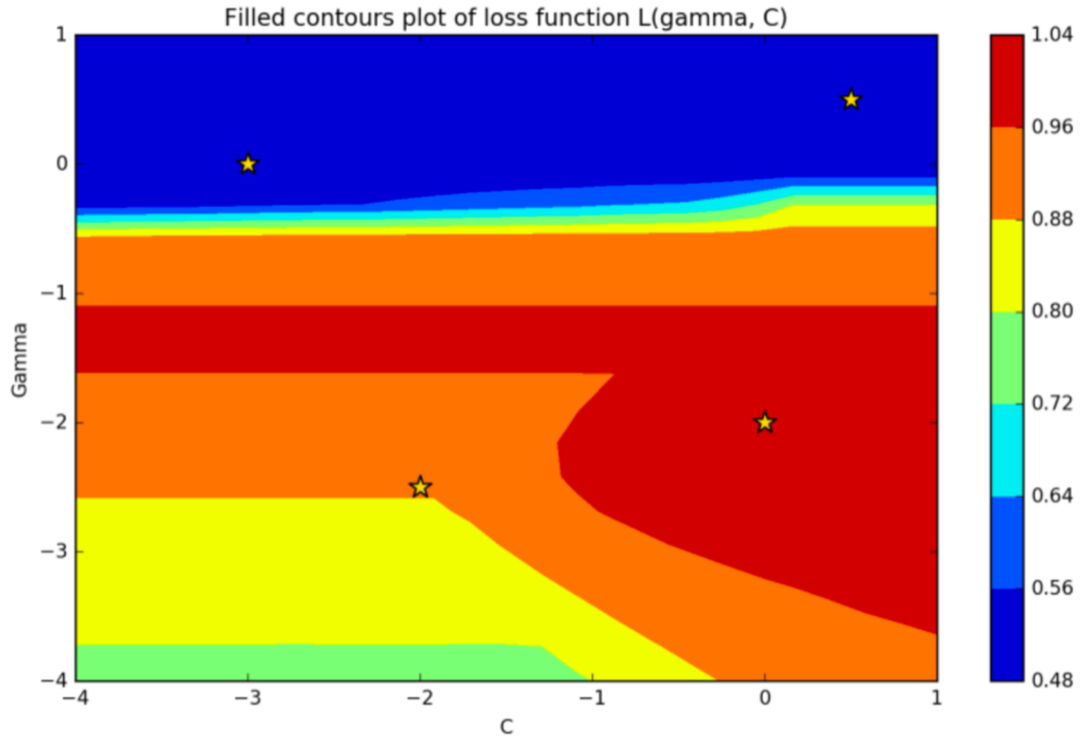


Scikit-Learn optimization classes

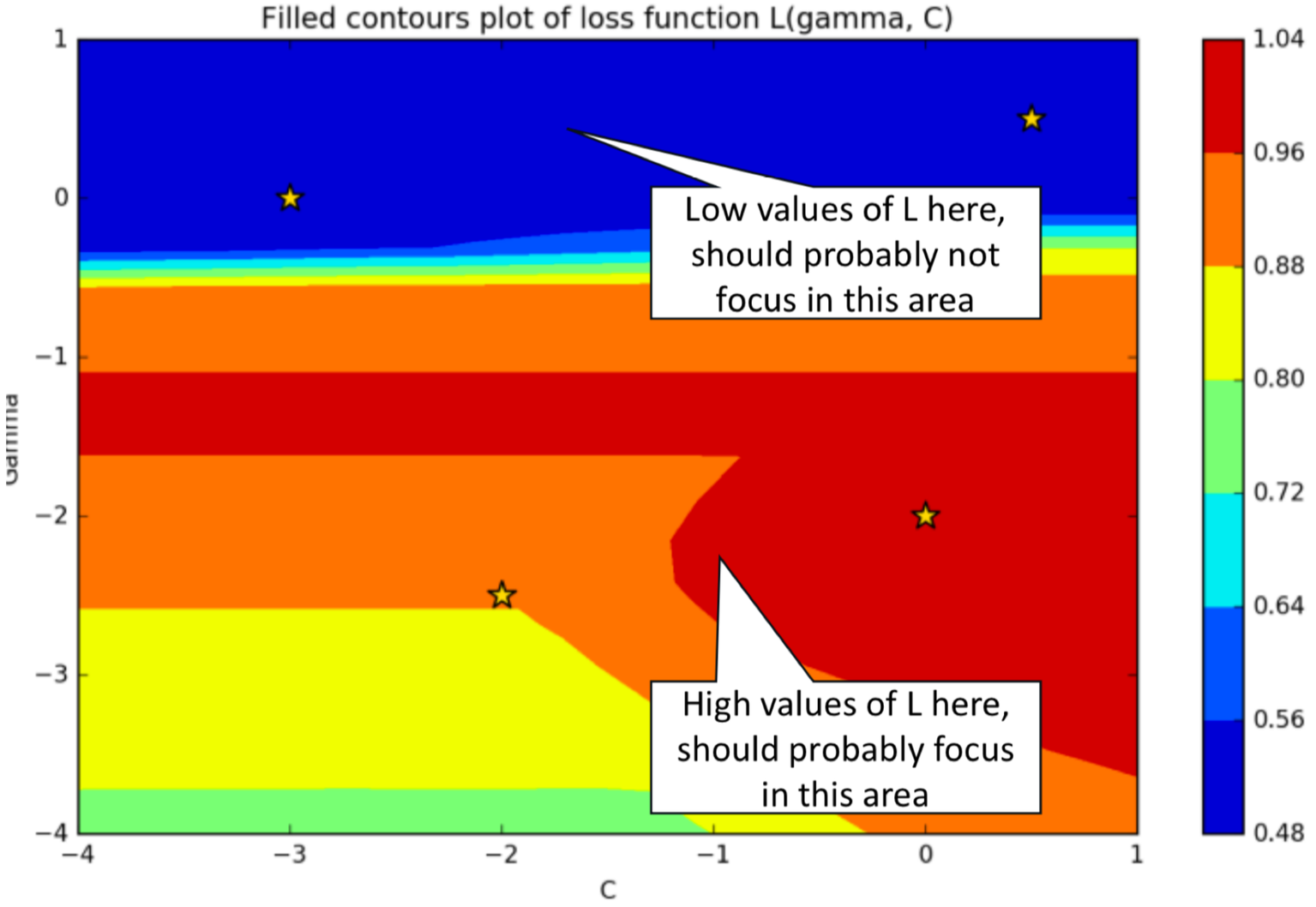
```
class sklearn.model_selection.RandomizedSearchCV(estimator,
param_distributions, n_iter=10, scoring=None,
fit_params=None, n_jobs=1, iid=True, refit=True, cv=None)
```

“If at least 5% of the points on the grid yield a close-to-optimal solution, then random search with 60 trials will find that region with high probability.”
Alice Zheng

https://www.oreilly.com/ideas/evaluating-machine-learning-models/page/5/hyperparameter-tuning



Towards Bayesian optimisation



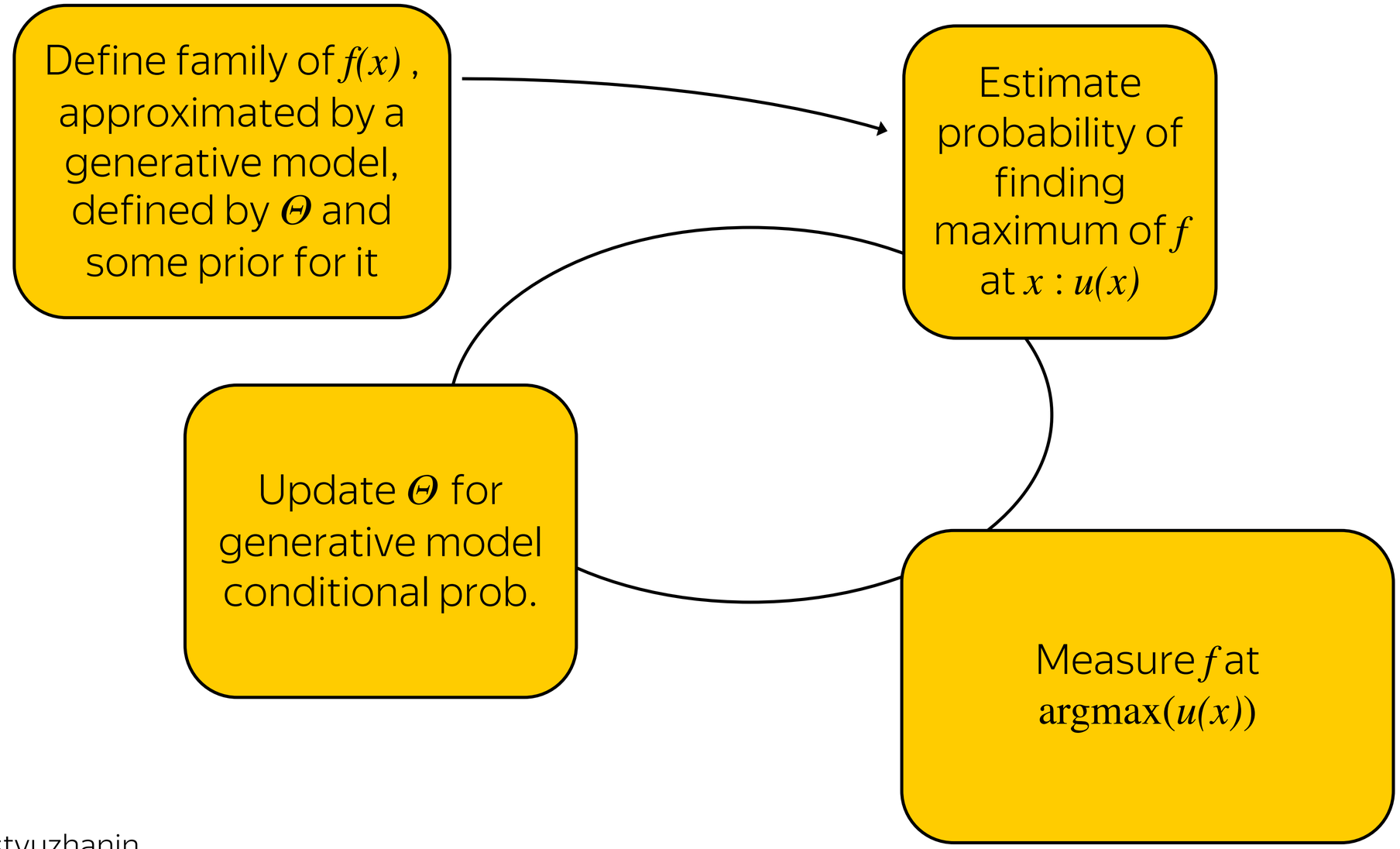
Why ‘Bayesian’ search?

Operates with sample distributions
Bayesian inference
Iteratively updates posterior distribution given prior and observations

Conditions

f is a black box for which no closed form is known (nor its gradients);
f is expensive to evaluate;
and evaluations of $y = f(X)$ may be noisy.

Bayesian optimization cycle



Types of generative models

Gaussian process Regression
Random Forest Regression
GBDT Regression
NN Regression

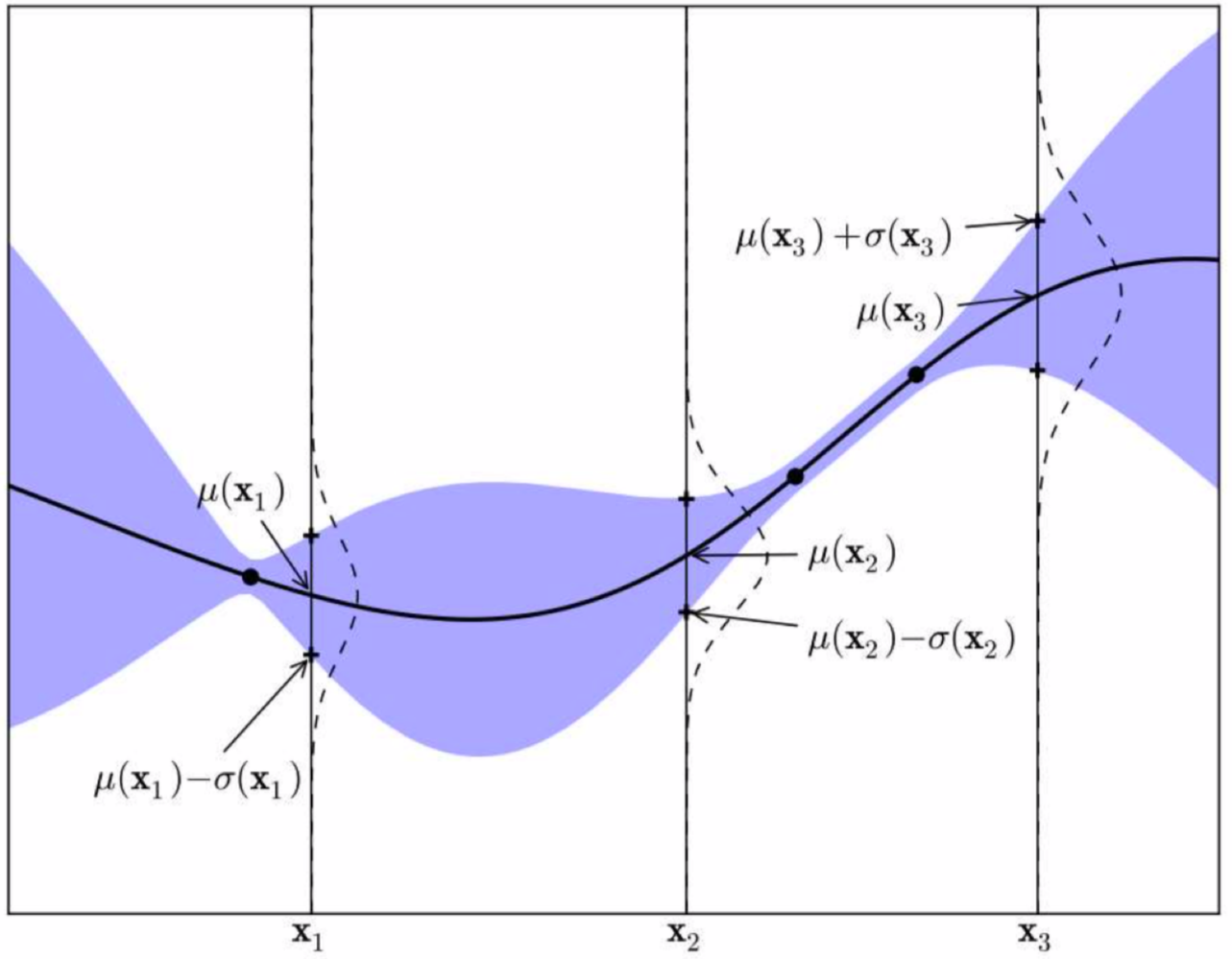
Gaussian Process (GP)

Like a Gaussian defines distribution of variables (tensors), GP defines distribution of functions:

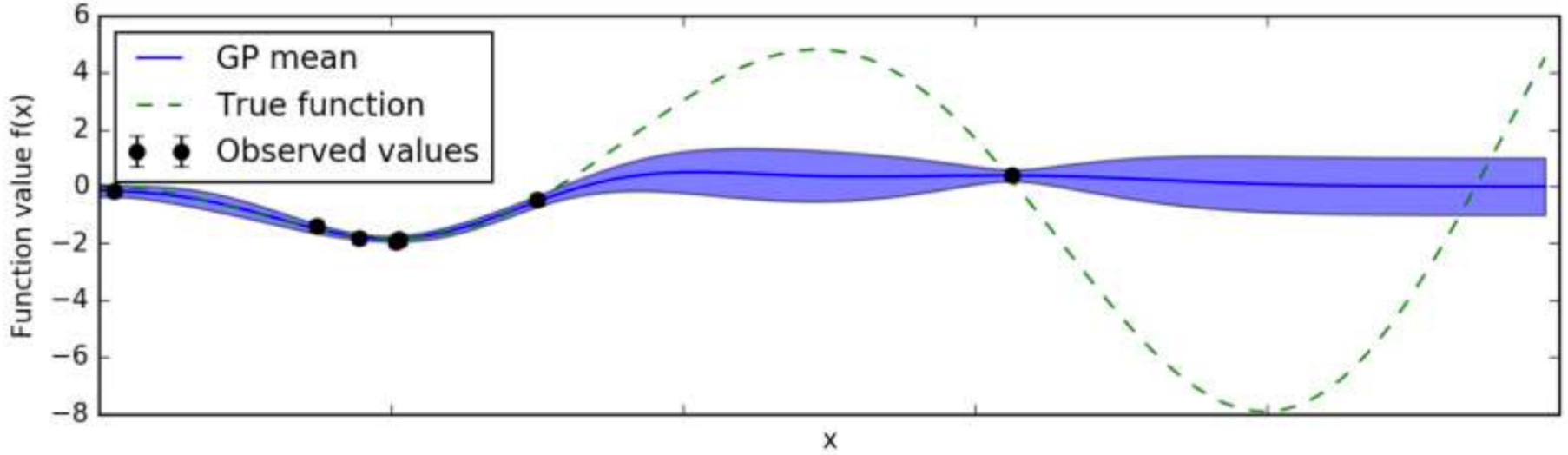
For every x, defines mean $\mu(x)$ and variance $\sigma(x)$

http://www.tmpl.fi/gp/

Determined by Θ:
X, covariance matrix + kernel



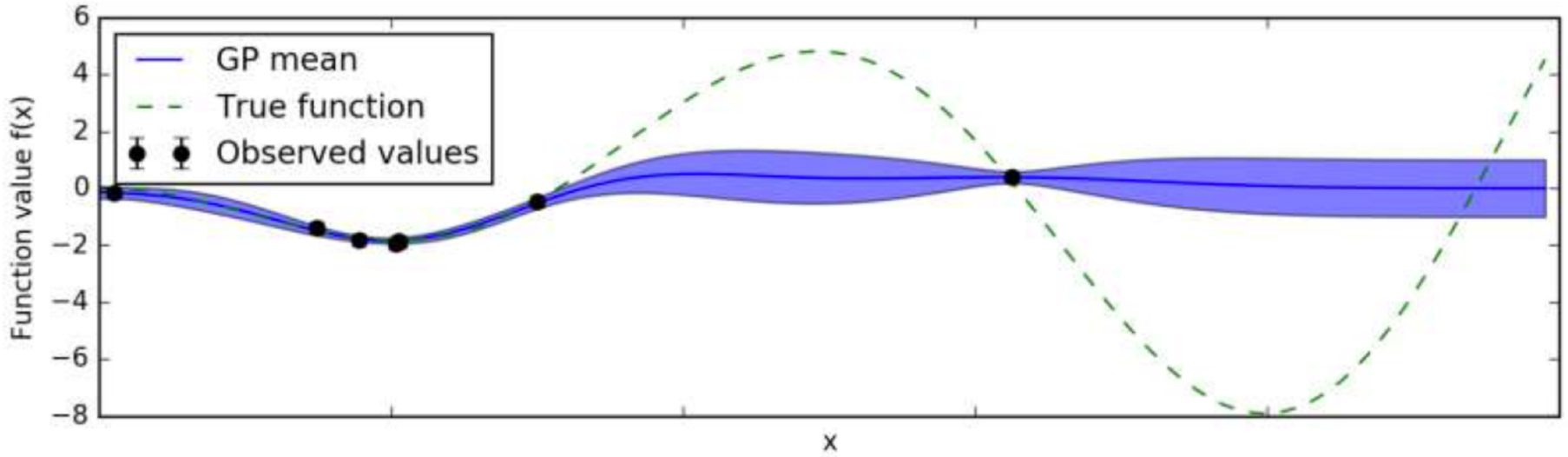
Next best candidate selection



$$I(x) = \max(f^* - Y, 0)$$

$$EI(x) = E_{Y \sim \mathcal{N}(\mu, \sigma^2)}[I(x)]$$

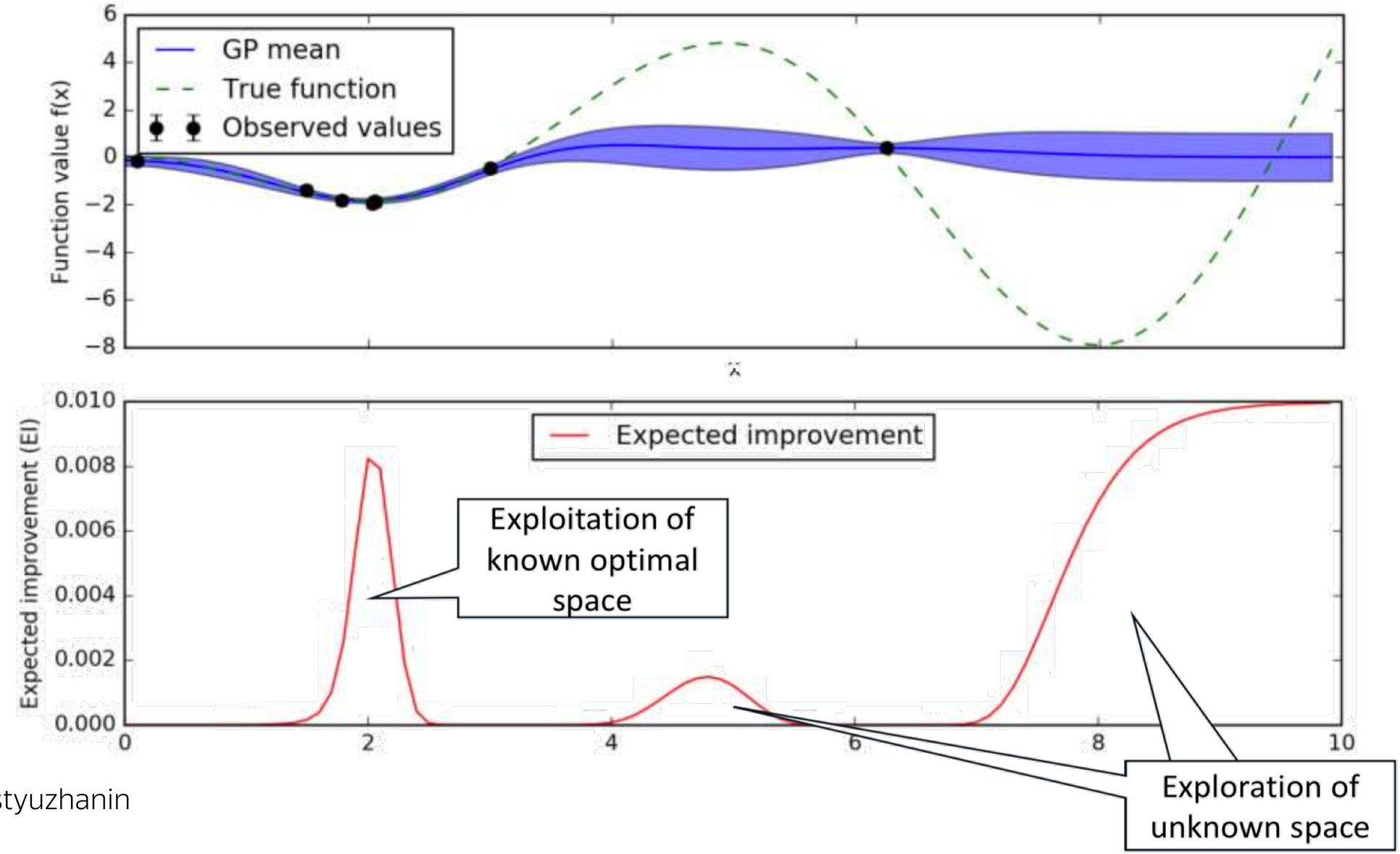
Next best candidate selection



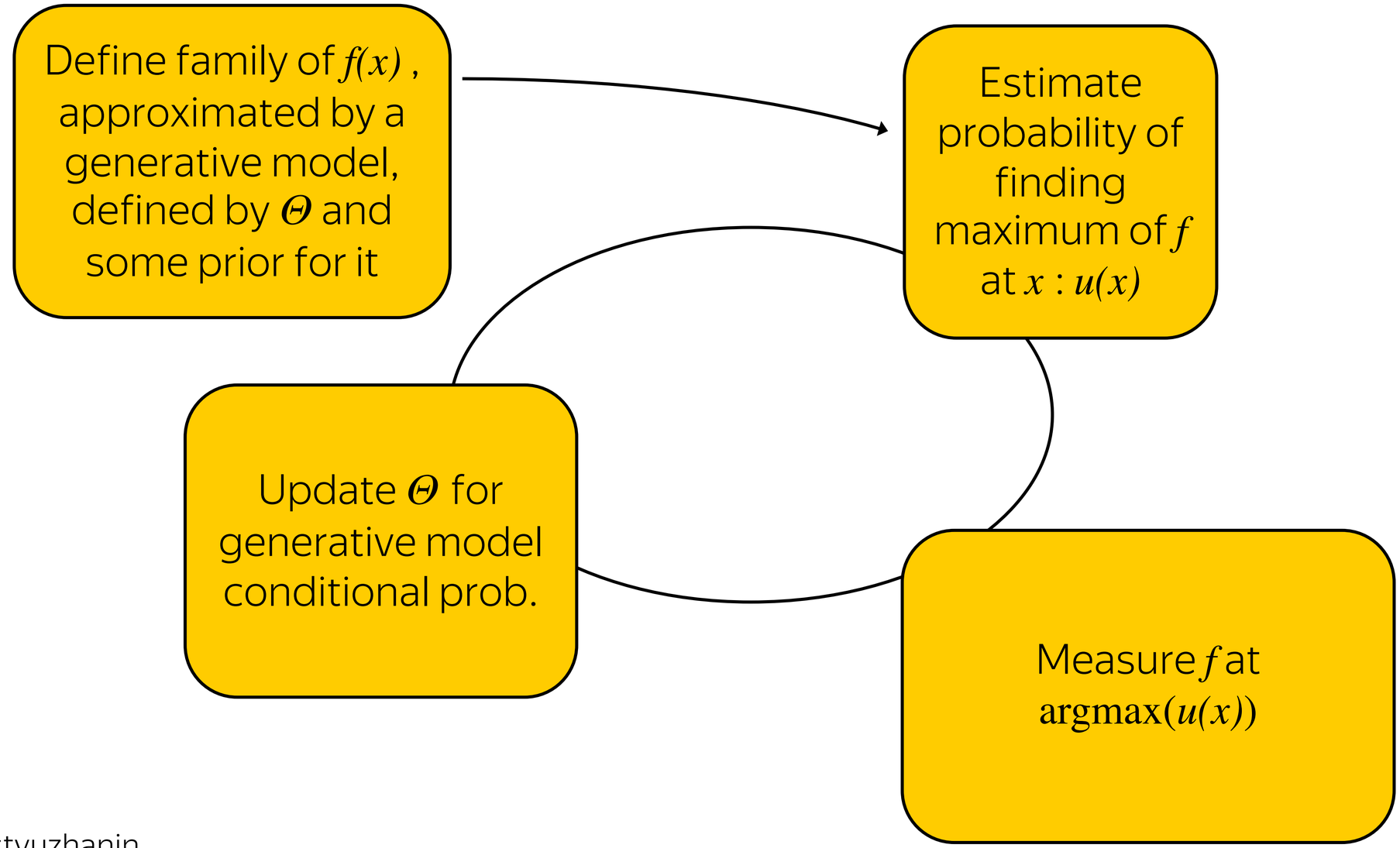
$$EI(x) = (f^* - \mu)\Phi(\frac{f^* - \mu}{\sigma}) + \sigma\phi(\frac{f^* - \mu}{\sigma})$$

where ϕ, Φ are the PDF, CDF of standard normal distribution, respectively http://ash-aldujaili.github.io/blog/2018/02/01/ei/)

Next best candidate selection



Bayesian optimization cycle



<Bayesian optimization demo>

https://github.com/HSE-LAMBDA/MLatMisis-2019

Discussion

- Applicable to network optimisation: number of layers, number of neurons, activation function, etc.
- Hyper parameter search is immediate step towards meta-learning
  - No gradients are available
  - Should account for stochasticity
- **Choose an appropriate prior for the hyperparameters sampling**: For parameters like a learning rate, or regularization term, it makes more sense to sample on the log-uniform domain, instead of the uniform domain
- **Choose the kernel for the Gaussian Process carefully**: each kernel implicitly assumes different properties on the loss function, in terms of differentiability and periodicity
- Libraries: scikit-optimize, Hyperopt, SMAC, ...

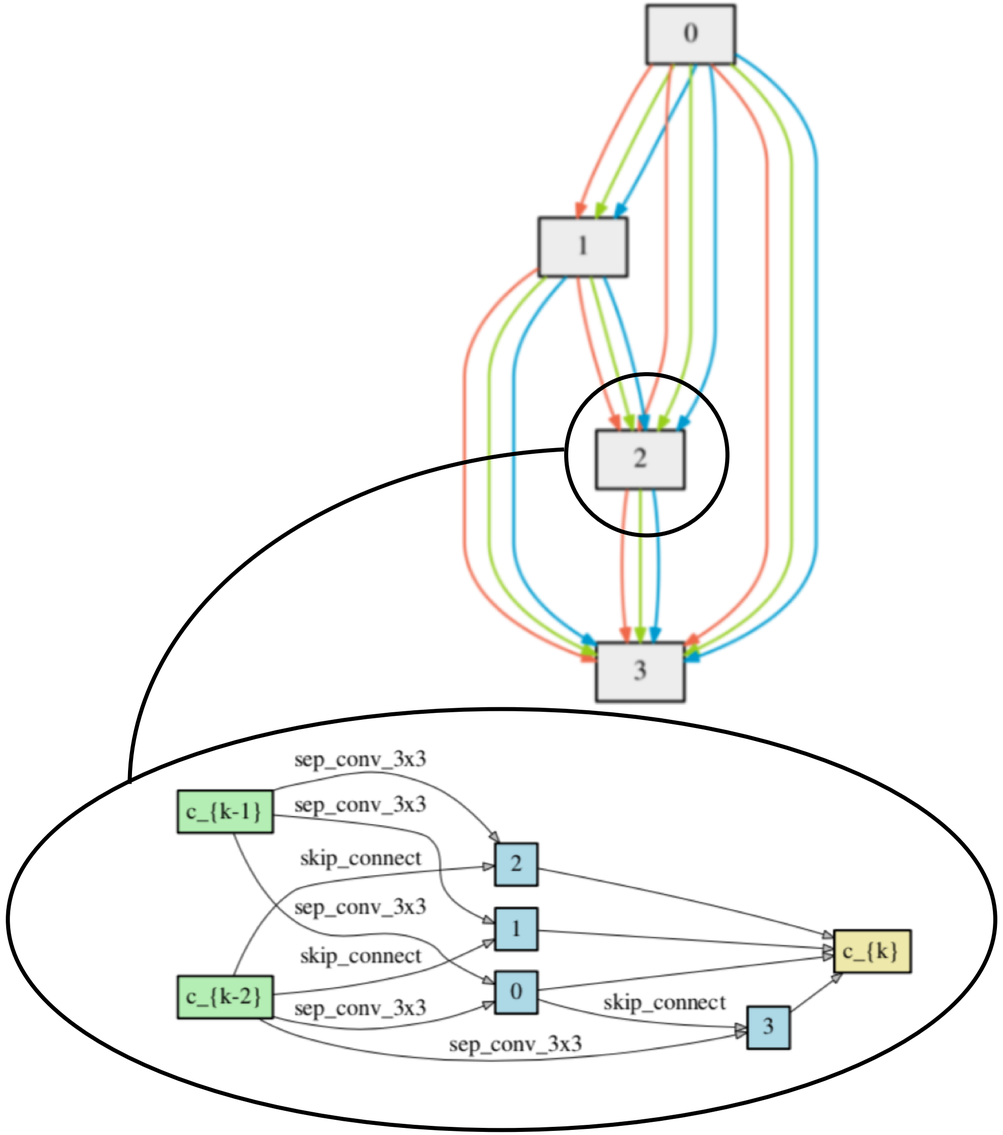
Network Architecture Search

Basic idea

Search for network topology as a graph of nodes connected by operations.

For simplicity the whole network is split into sub-graphs (cells) connected to each other.

Connection between nodes and cells are parametrized (relaxed) by softmax operation



Liu, Hanxiao and Simonyan, Karen and Yang, Yiming, DARTS: Differentiable Architecture Search, 2018

Structure of a cell[i]

0-th node is the output of cell[i-2], **1-st** node is the output of cell[i-1]

All other nodes are connected to all of its predecessors through operations like convolution, pooling, identity, zero:

$$x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)})$$

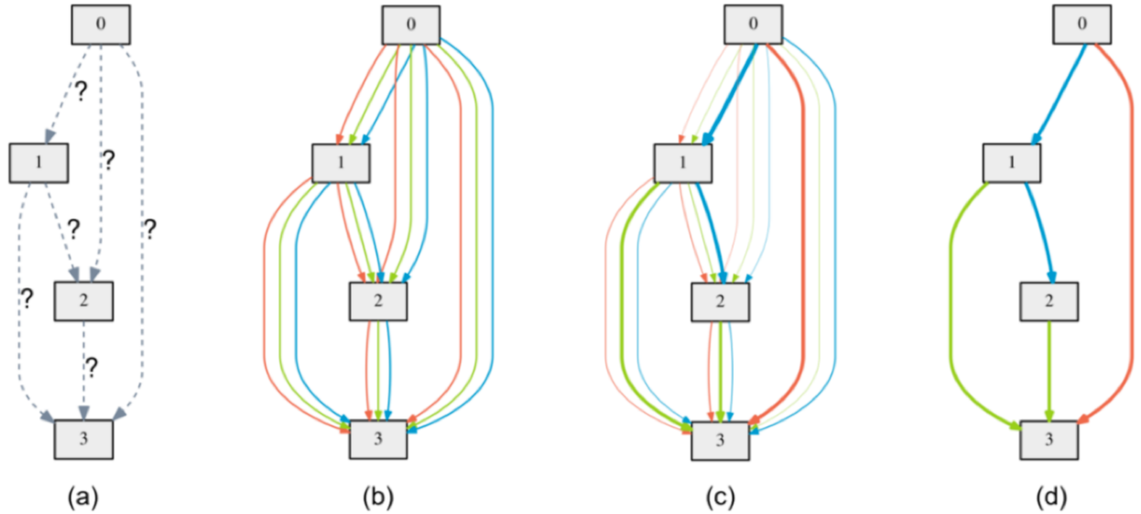
SoftMax of all possible operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

Nodes are parametrized by $\boldsymbol{w}$, connections are parametrized by $\boldsymbol{\alpha}$

Cell optimisation procedure

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \; \mathcal{L}_{train}(w, \alpha)$$



Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)

while *not converged* **do**

1. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \; \alpha_o^{(i,j)}$ for each edge (i, j)

Network structure

Initialise array of cells according to ‘genotype’ (specific cell structure trained before)

Connect each cell to two previous neighbors

Output of the final cell is the output of the network

Learn through backprop

Discussion

Trains reasonable amount of time ~ 1 day on GTX 1080
Works both for CNNs and RNNs
No rigorous guarantees, but:

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10. Results marked with † were obtained by training the corresponding architectures using our setup.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	manual
NASNet-A + cutout (Zoph et al., 2017)	2.65	3.3	1800	RL
NASNet-A + cutout (Zoph et al., 2017)†	2.83	3.1	3150	RL
AmoebaNet-A + cutout (Real et al., 2018)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	evolution
Hierarchical Evo (Liu et al., 2017b)	3.75 ± 0.12	15.7	300	evolution
PNAS (Liu et al., 2017a)	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	RL
Random + cutout	3.49	3.1	–	–
DARTS (first order) + cutout	2.94	2.9	1.5	gradient-based
DARTS (second order) + cutout	2.83 ± 0.06	3.4	4	gradient-based

Play on your own

Original repository: https://github.com/quark0/darts (supports pytorch 0.4)

Updated repository: https://github.com/dragen1860/DARTS-PyTorch

```
git clone <URL> ; cd DARTS-PyTorch
nvidia-smi # check if you have GPU onboard
mkdir exp
python train_search.py
<wait for genotype to evolve, add it to genotype.py>
python visualize.py <NAME_OF_YOUR_GENOTYPE>
python train.py —genotype <NAME_OF_YOUR_GENOTYPE>
```

Other approaches

Reinforcement learning
Neural Architecture Search with Reinforcement Learning (Zoph and Le, 2016)
NASNet (Zoph *et al.*, 2017)
ENAS (Pham *et al.*, 2018)

Evolutionary algorithm
Hierarchical Evo (Liu *et al.*, 2017)
AmoebaNet (Real *et al.*, 2018)

Sequential model-based optimisation (SMBO)
PNAS (Liu *et al.*, 2017)

Bayesian optimisation
Auto-Keras (Jin *et al.*, 2018)
NASBOT (Kandasamy *et al.* 2018)

Gradient-based optimisation
SNAS (Xie *et al.*, 2018)
DARTS (Liu *et al.*, 2018)

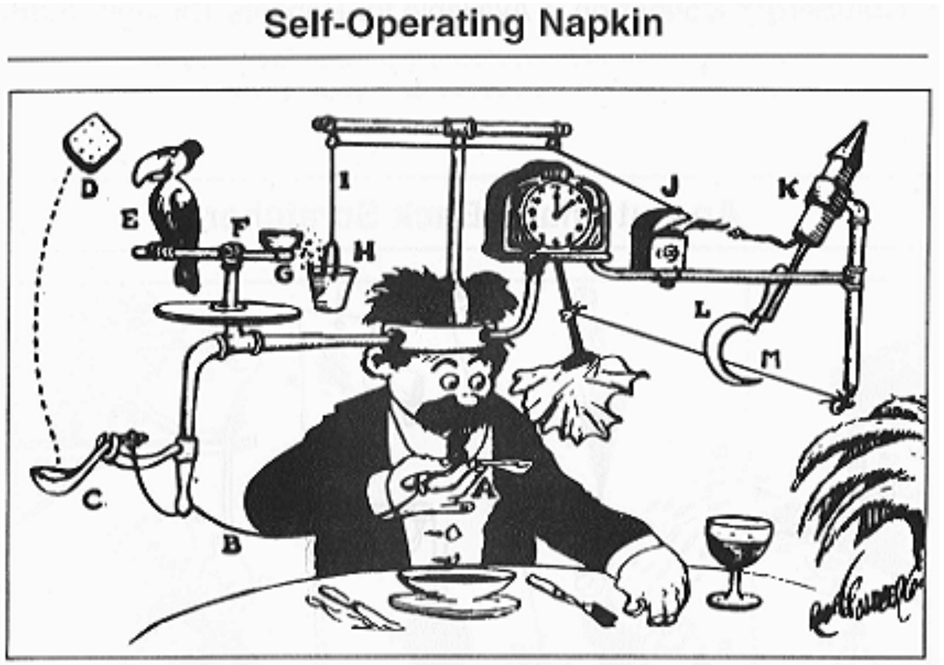
Other approaches

Controller

Child Model

input

DAG



https://www.fast.ai/2018/07/16/auto-ml2/

Bayesian Perspective on Deep Learning

$$P(w|X,Y) = \frac{P(Y|X,w)P(w)}{P(Y|X)}$$

Highlights

What if instead of training a network via Maximum Likelihood (loss-function minimization), we could define a neural net as a *tensor* sampled from some distribution that we can get from the data?

$$p(\theta|X) = \frac{\prod_{i=1}^{n} p(x_i|\theta)p(\theta)}{\int \prod_{i=1}^{n} p(x_i|\theta)p(\theta)d\theta}$$

Encodes uncertainty of the *tensor* in terms of distribution and expressed via data using Bayesian rule (inference).

Bayesian Machine Learning

- Suppose we're given training data (X, T) and a probabilistic classifier $p(t|x, W)$
- Define reasonable prior over the weights $p(W)$
- Training stage:

$$p(W|X, T) = \frac{p(T|X, W)p(W)}{\int p(T|X, W)p(W)dW}$$

- Test stage:

$$p(t^*|x^*, X, T) = \int p(t^*|x^*, W)p(W|X, T)dW$$

- Bayesian learning results in an **ensemble** of classifiers

Variational Trick

- Approximate posterior with a simpler distribution from a restricted parametric family

$$p(W|X,T) \approx q(W|\phi) = \arg\min_{\phi} KL(q(W|\phi)||p(W|X,T))$$

- It can be shown that

$$\arg\min_{\phi} KL(q(W|\phi)||p(W|X,T)) = \arg\max_{\phi} \int q(W|\phi) \log \frac{p(T|X,W)p(W)}{q(W|\phi)} dW$$

- The last expression is usually denoted as $\mathcal{L}(\phi)$ and has special name **evidence lower bound** (ELBO)

ELBO properties

ELBO

$$\mathcal{L}(\phi) = \int q(W|\phi) \log \frac{p(T|X,W)p(W)}{q(W|\phi)} dW \to \max_{\phi}$$

has several nice properties

- We may compute its stochastic gradient by performing **mini-batching** and removing integral with its MC estimate
- We do not overfit - the richer is parametric family the closer we are to the true posterior
- We may rewrite ELBO as follows

Regularizer

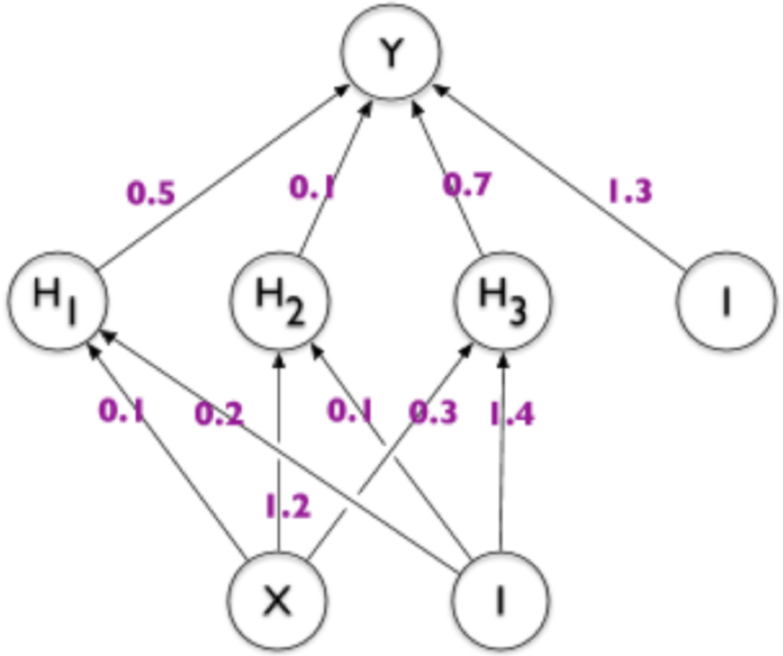
$$\mathcal{L}(\phi) = \int q(W|\phi) \log p(T|X,W) dW - KL(q(W|\phi)||p(W))$$

Data term

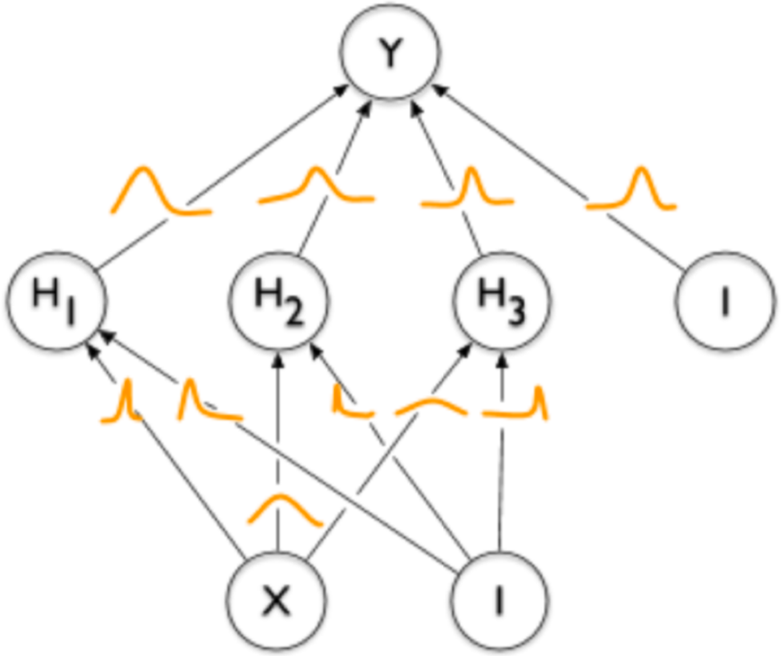
- The second term prevents $q(W|\phi)$ from collapsing to maximum likelihood point

Bayesian Neural Network representation

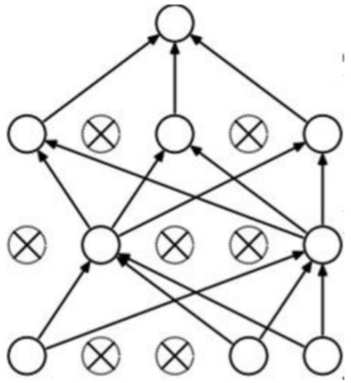
Standard Neural Net



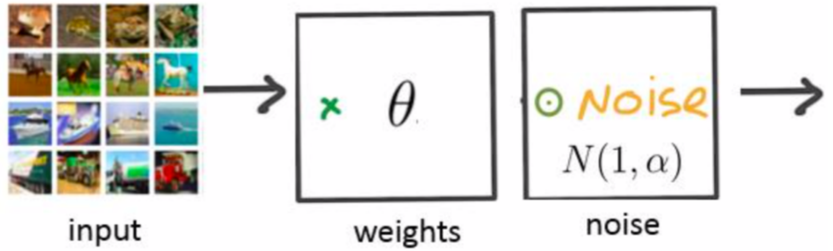
Bayesian Neural Net



Dropout reinvented



Binary dropout



Gaussian droput

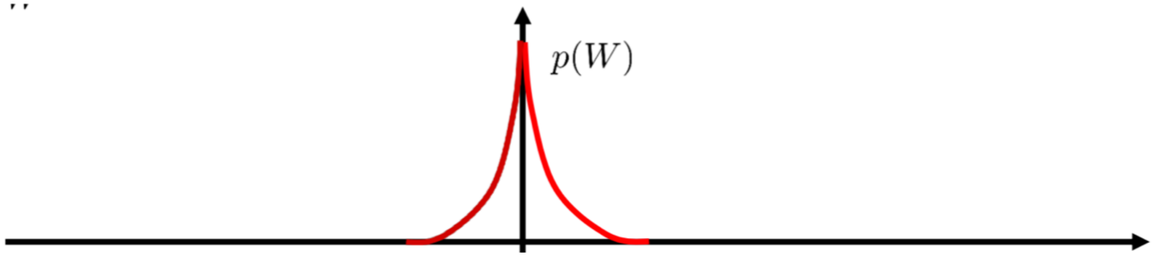
- In 2015 Kingma, Salimans and Welling decided to understand the nature of dropout
- They assumed that gaussian dropout corresponds to Bayesian procedure that optimizes ELBO using SGD with $q(W|\theta, \alpha) = \mathcal{N}(W|\theta, \alpha\theta^2)$

$$\int \mathcal{N}(W|\theta, \alpha\theta^2) \log p(T|X, W) dW - KL(\mathcal{N}(W|\theta, \alpha\theta^2)||p(W)) \to \max_{\theta}$$

- The first term corresponds to the criterion that is really optimized during dropout training... BUT there is no KL-term!

Variational Dropout

$$p(W) \propto \frac{1}{|W|}$$



- KL-term does not depend on θ but still depends on α

$$\mathcal{L}(\theta, \alpha) = \text{DataTerm}(\theta, \alpha) + KL(\alpha) \to \max_{\theta}$$

- Why not trying to optimize ELBO both w.r.t. θ and α?

$$\mathcal{L}(\theta, \alpha) = \text{DataTerm}(\theta, \alpha) + KL(\alpha) \to \max_{\theta, \alpha}$$

Sparse Variational Dropout

- Now we may extend the variational family even further and assign **individual dropout rates** α_{ij} per each weight

$$q(W|\theta,\alpha) = \prod_{i,j} \mathcal{N}(w_{ij}|\theta_{ij}, \alpha_{ij}\theta_{ij}^2)$$

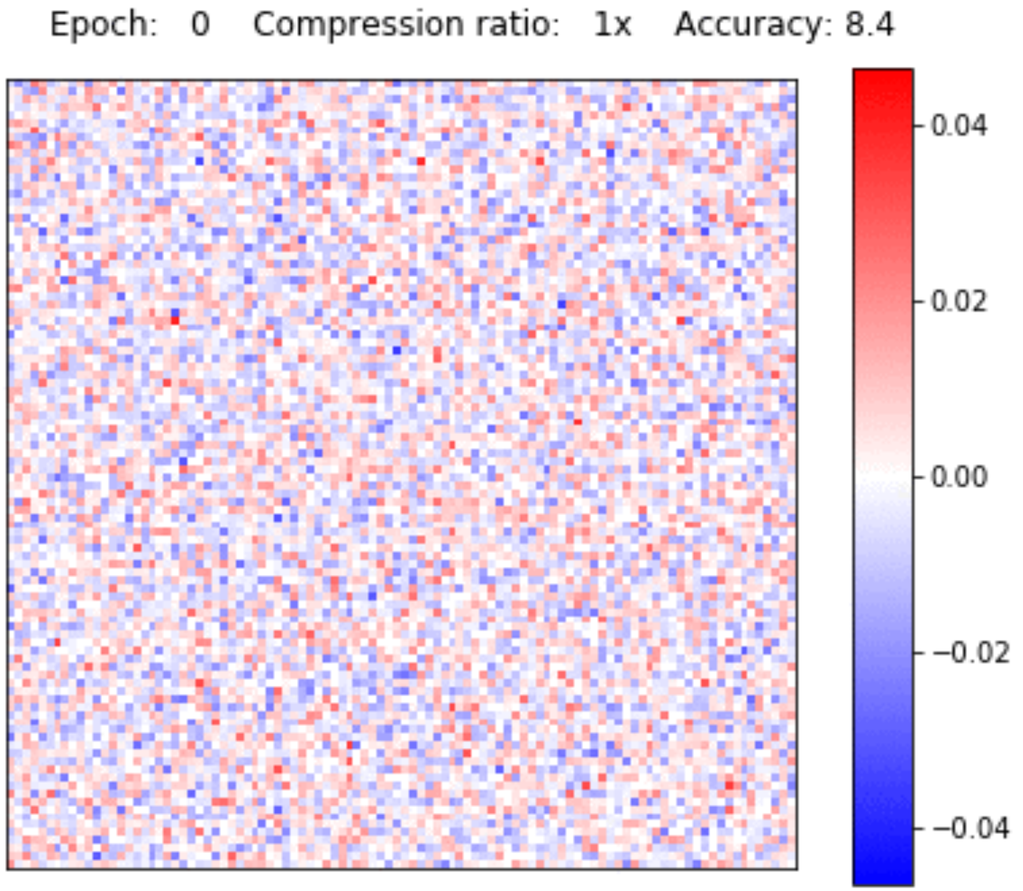
- It can be shown that if $\alpha_{ij} \to +\infty$ then $\theta_{ij} = O\left(\frac{1}{\alpha_{ij}}\right)$ i.e.

$$\lim_{\alpha_{ij}\to+\infty} q(w_{ij}|\theta_{ij},\alpha_{ij}) = \delta(0)$$

- Incredebly efficient way for removing the redundancy of current deep architectures
- Up to 99.9% of the weights in the layer become irrelevant

Dmitry Molchanov, Arsenii Ashukha, Dmitry Vetrov. Variational Dropout Sparsifies Deep Neural Networks. ICML 2017.

Visualisation

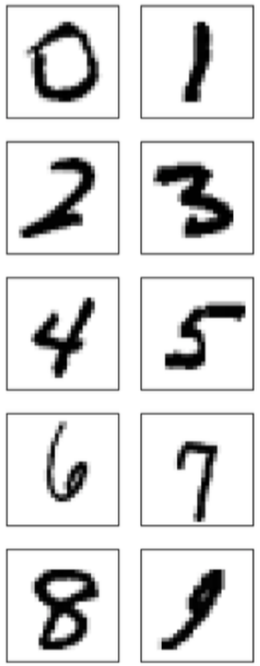


Comparison

Fully Connected network: LeNet-300-100

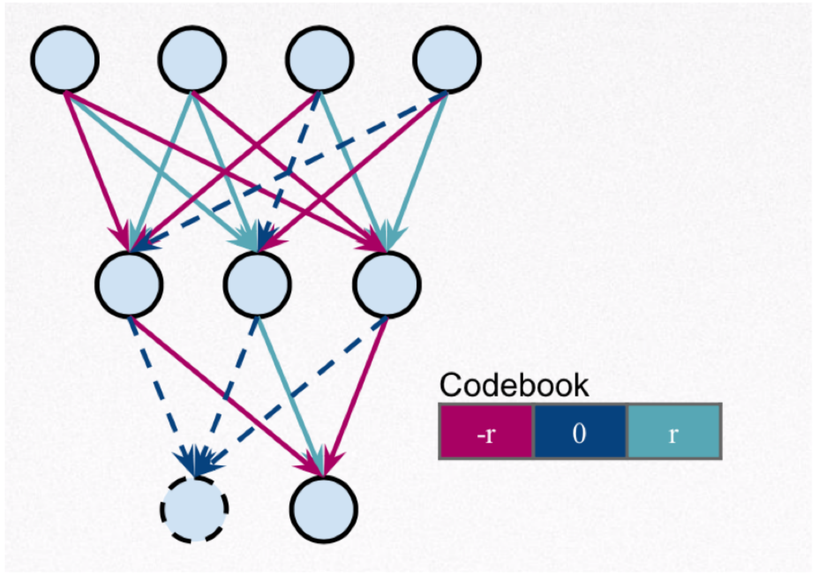
Convolutional network: Lenet-5-Caffe

Network	Method	Error %	Sparsity per Layer %	$\frac{\lvert\mathbf{W}\rvert}{\lvert\mathbf{W}_{\neq 0}\rvert}$
	Original	1.64		1
	Pruning	1.59	92.0 – 91.0 – 74.0	12
LeNet-300-100	DNS	1.99	98.2 – 98.2 – 94.5	56
	SWS	1.94		23
(ours)	Sparse VD	1.92	98.9 – 97.2 – 62.0	**68**
	Original	0.80		1
	Pruning	0.77	34 – 88 – 92.0 – 81	12
LeNet-5-Caffe	DNS	0.91	86 – 97 – 99.3 – 96	111
	SWS	0.97		200
(ours)	Sparse VD	0.75	67 – 98 – 99.8 – 95	**280**

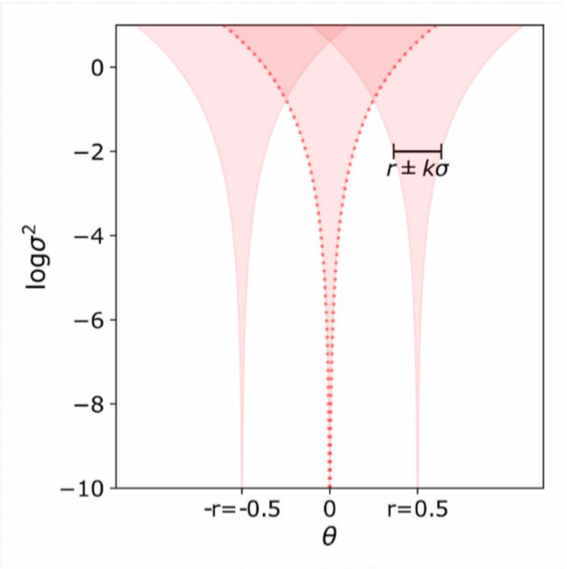


Simultaneous pruning and quantization

Simultaneous pruning and quantization of weights - ternary codebook



Quantizing prior: “multi-spike-and-slab”

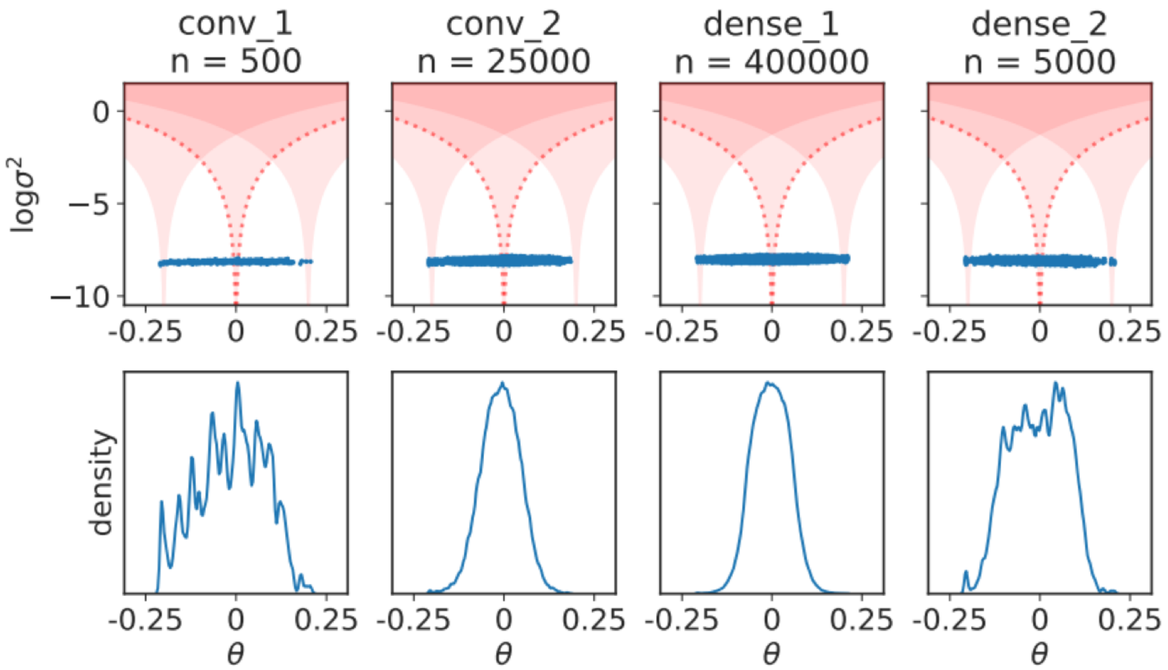


$$p(w_{ij}) \propto \sum_{k=1}^{K} a_k \int \frac{1}{|z|} \mathcal{N}\left(w_{ij}|c_k, z^2\right) dz = \sum_{k=1}^{K} a_k \frac{1}{|w_{ij} - c_k|}$$

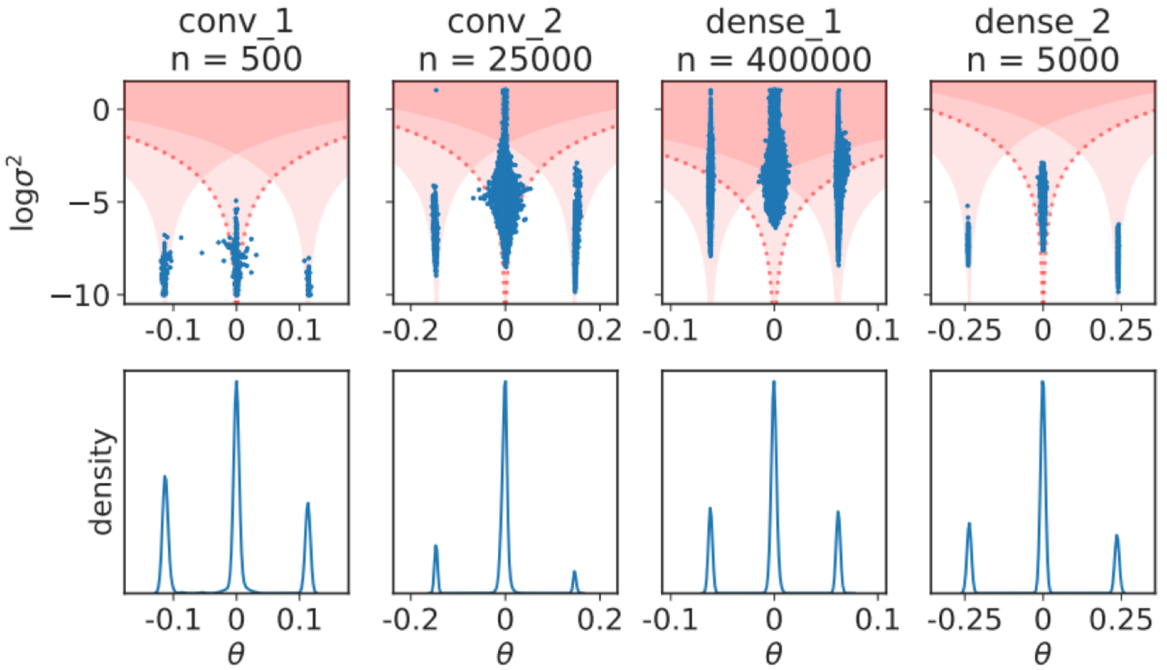
Variational Network Quantization, Achterhold et al., ICLR 2018

Simultaneous pruning and quantization

Standard Training



Variational Network Quantization



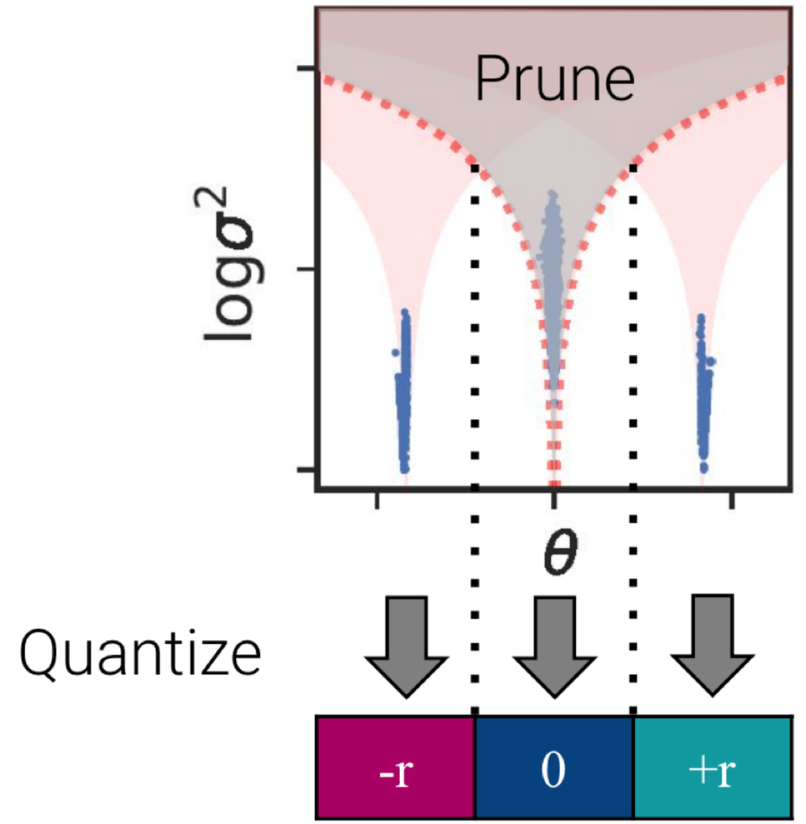
Variational Network Quantization, Achterhold et al., ICLR 2018

Simultaneous pruning and quantization

Prune weights with small expected value or large variance

Quantize by assigning weights to closest quantization level

Method	val error [%]	$\frac{\lvert w \neq 0 \rvert}{\lvert w \rvert}$ [%]	bits
Original	6.81	100	32
VNQ (no P&Q)	8.32	100	32
VNQ + P&Q (w/o 1)	8.78	46	2 (32)
VNQ + P&Q	8.83	46	2



Variational Network Quantization, Achterhold et al., ICLR 2018

More details

https://www.youtube.com/watch?v=TD2PF6TZcx0

https://www.youtube.com/playlist?list=PLe5rNUydzV9Q01vWCP9BV7NhJG3j7mz62

Discussion

Bayesian framework is extremely powerful and extends ML tools
Impossible to overtrain, since more parameters -> better W distribution!
Bayesian NN is an ensemble, hence uncertainty estimation is easier
Scalable algorithms for approximate Bayesian inference are already available
Easily accounts for additional objectives (compression, quantization)
Quite breathtaking mathematical development!

LHCb PID case

Outlook & Conclusion

- Optimisation of a neural networks is not a merely technical procedure
  - Too many techniques, constraints (make it faster, smaller, cheaper?)
- Bayesian approach: allows for complex constraints, fast
- Demand for inclusion of hardware in the optimization loop
  - Specification
  - Simulation
- Practical notebooks: github repo

http://cs.hse.ru/lambda/en
anaderiRu@twitter
austyuzhanin@hse.ru

Moar interesting stuff

- Ternary quantization https://github.com/TropComplique/trained-ternary-quantization/tree/master/ttq_densenet_small
- The State of Sparsity in Deep Neural Networks, https://arxiv.org/pdf/1902.09574.pdf
- Information Bottleneck: https://github.com/ravidziv/IDNNs
- Google vizier (https://github.com/tobegit3hub/advisor)
- https://github.com/ray-project/ray
- https://github.com/keras-team/autokeras , https://arxiv.org/pdf/1806.10282.pdf
- The Lottery Ticket Hypothesis… https://arxiv.org/abs/1803.03635

Frequentist vs Bayesian

	Frequentist	Bayesian
Randomness	Objective indefiniteness	Subjective ignorance
Variables	Random and Deterministic	Everything is random
Inference	Maximal likelihood	Bayes theorem
Estimates	ML-estimates	Posterior or MAP-estimates
Applicability	$n \gg 1$	$\forall n$