Variational Dropout Sparsification for Particle Identification speed-up

A. Ryzhikov[1,2]

on behalf of LHCb collaboration

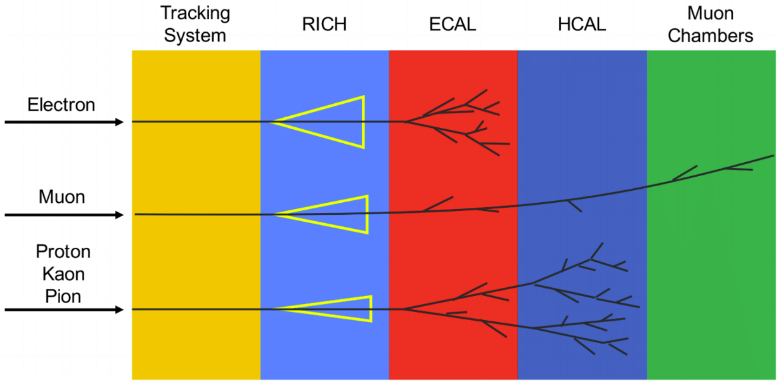
[1]Department of Computer Science
NRU Higher School of Economics, LAMBDA

[2]Yandex School of Data Analysis

QFTHEP, 2019

Original problem. PID

Original problem: Particle Identification (PID)



$\{$Electron, Proton, Muon, Kaon, Pion$\}$ + "Ghost" - **6 classes**

Baseline

6x shallow DNNs (TMVA based baseline):

- 6 binary classifiers (trained in one-vs-all mode)
- 32-34 input features for each binary classifier
- each classifier is dense neural network with 1 hidden layer
- complexity for each one is following "Number of neurons in hidden layer" = 1.4 * "input features count"
- $\sim$ 9200 trainable parameters in total

Single 6 outputs DNN (initial proposal):

- single multiclass classifier as alternative for 6 binary classifiers of baseline
- 1 hidden layer with 150 neurons
- same complexity ($\sim$ 9200 parameters) and speed as baseline provides
- ($\star$) 59 input features

Problem: **Reach maximum** neural network's prediction **speed at** the given **quality** (ROC AUC)

Speed-up 1DNN. Approach 1 (the worse solution)

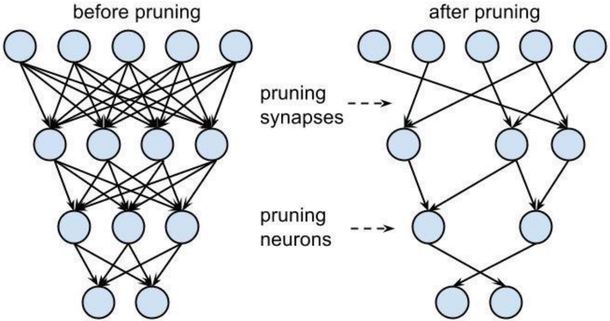
Speed-Up Idea 1 Try different NN architecture configurations and evaluate speed and quality for each one

Drawbacks

- Pointwise estimation. Random walk in hope to find best configuration. Not so precise as it could be using narrow optimization algorithms
- Too long. ~ 12 hours to train each NN configuration. Usually you have at least 10 "candidates" for best configuration role!
- lots of redundant code

Speed-up baseline. More advanced techniques

Idea 2 Train DNN only once and drop all the redundant connections after



Speed-up 1DNN. Approach 2 (better solution)

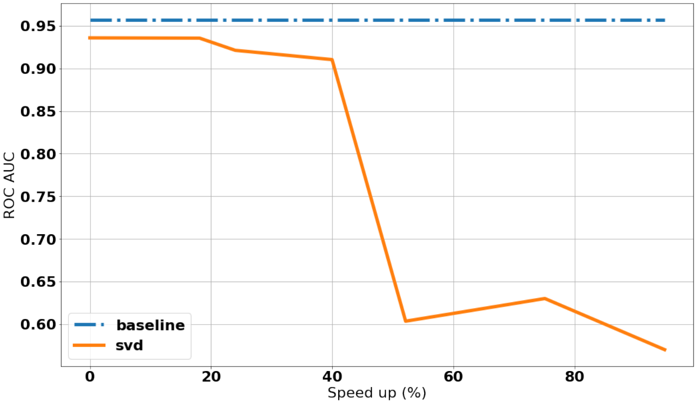
Speed-Up Idea 2 Try to use more advanced techniques

- L1-pruning
  - **Idea** train NN with L1-regularization term and drop connections with small weights from time to time
- SVD
  - **Idea** k-rank approximation using Singular Value Decomposition (SVD): $\theta \approx U^T \Lambda V$ (θ - trainable weights)
- Ternary trainable quantization
  - **Idea** Transform each layer's weight to 3 possible values: $\{\theta^+, 0, \theta^-\}$

Common pros

- much faster than bruteforce

Idea 2. Post-pruning

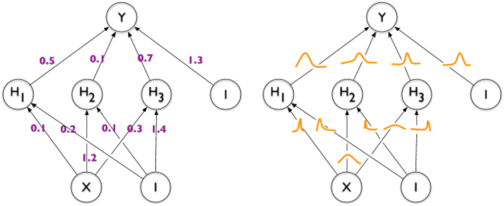


Common problem

- loss of quality and information
- still lot's of code
- small speed-up (up to 2-5 times)

Idea 3. Variational dropout

Idea: **Find useless connections variating its weights** at the specified range/distribution and look how the quality changes



Alternative idea: Drop all the connections with wide weight's distribution, if such distribution is **proper** one!
Problem: How to fit proper weight's distribution for each connection?
Simplest solution: Let each connection's distribution to be gaussian with specific trainable μ and σ (*variational parameters*).

Idea 3. Technical details

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 $\mathcal{F}$)

Bayes ML General idea - estimate the **whole distribution** $p(\theta|X_{train}, \mathcal{F})$ for parameters θ instead of pointwise estimation $\theta^{\mathcal{F}}_{train}$ of them

Bayesian inference $p(\theta|X, \mathcal{F}) = \frac{p(X|\theta,\mathcal{F})p(\theta|\mathcal{F})}{\int p(X|\theta,\mathcal{F})p(\theta|\mathcal{F})d\theta} = \frac{p(X|\theta,\mathcal{F})p(\theta|\mathcal{F})}{p(X|\mathcal{F})}$

$p(X|\mathcal{F})$ - probability to observe the given data X with the given NN configuration $\mathcal{F}$ of neural network!
Idea - the higher $p(X|\mathcal{F})$ (*evidence*) the better NN configuration $\mathcal{F}$ is!
Problem - how to optimize evidence $p(X|\mathcal{F})$ over $\mathcal{F}$? $\mathcal{F}$ is discrete!

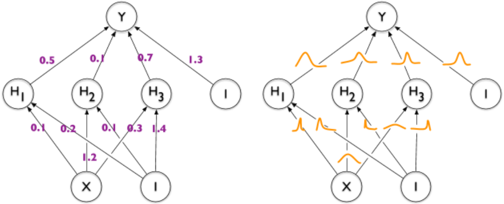
Idea 3. Technical details. ELBO

$log(p(X|\mathcal{F})) = L(q_\phi) + KL[q_\phi(\theta|\mathcal{F})||p(\theta|X,\mathcal{F})]$
$L(q_\phi) = \mathbb{E}_{q_\phi} log(p(X|\theta,\mathcal{F})) - KL[q_\phi(\theta|\mathcal{F})||p(\theta)]$ - *evidence lower bound*
Notation:

- KL - *Kullback-Leibler divergence*
- $q_\phi(\theta)$ - auxilary parametrized distribution over trainable weights (θ)

Interesting fact In discrete case $L(q_\phi)$ is -(cross entropy + regularizer)!
Idea: Instead of estimating and maximizing $log(p(X|\mathcal{F}))$ over discrete $\mathcal{F}$ directly let's maximize the lower bound over continuous ϕ!

Illustration of q_ϕ



Results

Evaluation criterium maximum speed with no significant quality reduction (*Python 3.6*)

Method	# Neurons	Electron	Ghost	Kaon	Muon	Pion	Proton	Speed-Up
6xDNN	45-48	0.9855	0.9485	0.9148	0.9844	0.9346	0.9178	x1
1xDNN	150	0.9863	0.9570	0.9145	0.9889	0.9463	0.9167	x1
1xDNN	30	0.9871	0.9557	0.9158	0.9893	0.9427	0.9125	x5
Ternary	Auto	0.9843	0.9435	0.9154	0.9834	0.9352	0.9110	x5
BDNN	Auto	**0.9881**	0.9548	**0.9244**	**0.9896**	**0.9509**	**0.9228**	**x16**

Pre-conclusion

- best NN configuration (in terms of ROC AUC and speed) is automatically found!
- x16 speed-up (*Python vs. Python*), x7.5 speed-up (*C++ vs. C++*)
- ... moreover, the quality is getting slightly better! (Besides the Ghost, where the quality is comparable)

Usage

```python
import torch
from torch import nn

model = nn.Sequential([
    nn.Linear(59,150),
    nn.Tanh(),
    nn.Linear(150, 6)
    ])


obj = nn.CrossEntropy(...)
```

Baseline implementation

```python
import torch
from torch import nn
import torch_ard as nn_ard

model = nn.Sequential([
    nn_ard.LinearARD(59, 150),
    nn.ReLU(),
    nn_ard.LinearARD(150,6)
    ])

obj = nn.CrossEntropy(...) + \
    nn_ard.get_ard_reg(model)
```

Bayesian NN implementation

Source code: https://github.com/HolyBayes/pytorch_ard
Installation: pip install pytorch-ard

Conclusion

- Leading methods for NN's sparsification and speed-up were tested
- Bayesian Sparsification is the best: x16 (Python), x7.5 (C++)
- Can be applied to almost any problem
- Finds the best NN configuration with no overfitting
- Uncertainty estimation for free! [4], [5]
- Integrated with LHCb software

D Molchanov, A Ashukha, D Vetrov
Variational Dropout Sparsifies Deep Neural Networks.
arXiv:1701.05369, 2017.

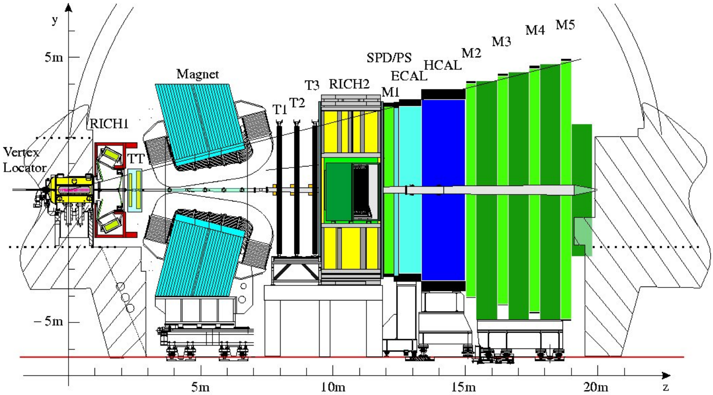
A Ryzhikov
Variatonal Dropout Sparsifies (Pytorch).
https://github.com/HolyBayes/pytorch_ard

J Duarte and Co.
Deep learning on FPGAs for L1 trigger and Data Acquisition
https://indico.cern.ch/event/587955/contributions/2937529/

T Pearce, M Zaki, A Brintrup, A Neely
Uncertainty in Neural Networks: Bayesian Ensembling
arXiv:1810.05546, 2018

C Guo, G Pleiss, Y Sun, K Q. Weinberger
On Calibration of Modern Neural Networks
arXiv:1706.04599, 2017

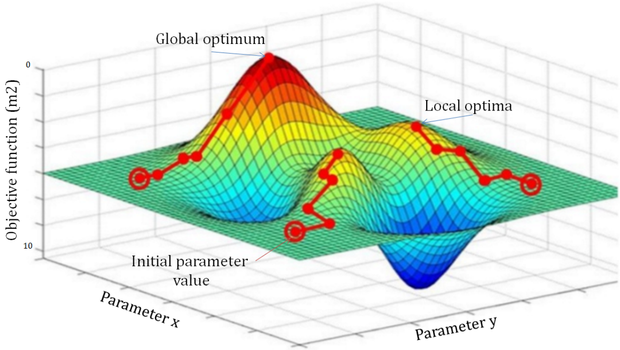
LHCb detector



Authors benchmarks

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

Intuition



Intuition of the results - Finding global optimum with complex model (containing both "x" and "y" parameters) with further dropout of some parameters ("x" for instance) is better that finding global optimum with initially simplified model with "y" parameter only!