Anomaly detection in low-latency systems : Case study of CERN-LHC superconducting magnets

Maciej Wielgosz
ACC Cyfronet

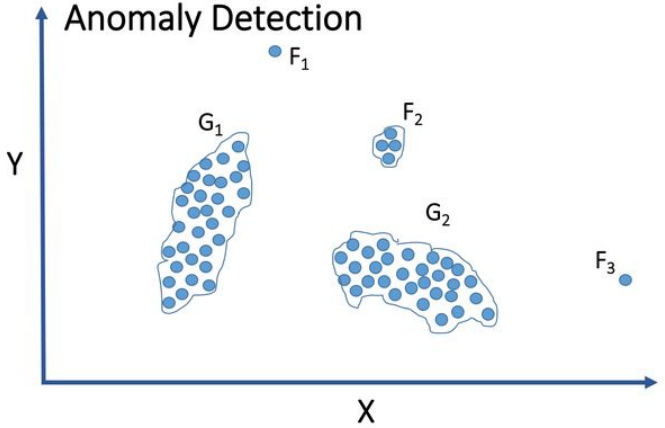
Agenda

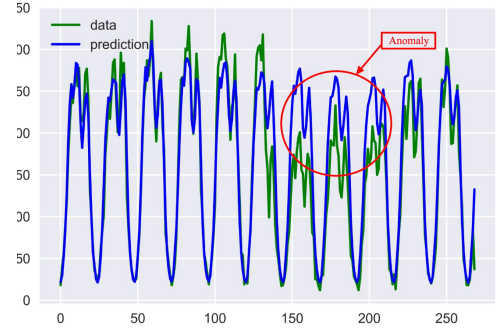
- Introduction
- Time series modelling - why RNNs?
- Architecture of the system
- Experimental setup
- Experimental results
- Compression schemes of DL models
- RNN Deep Learning model compression

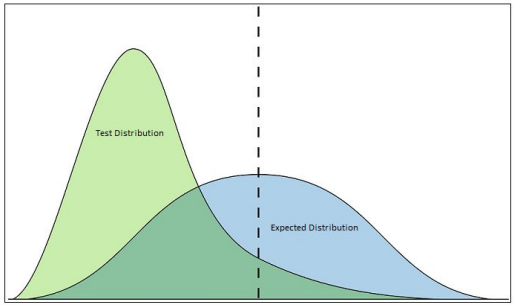
Few words about me ...

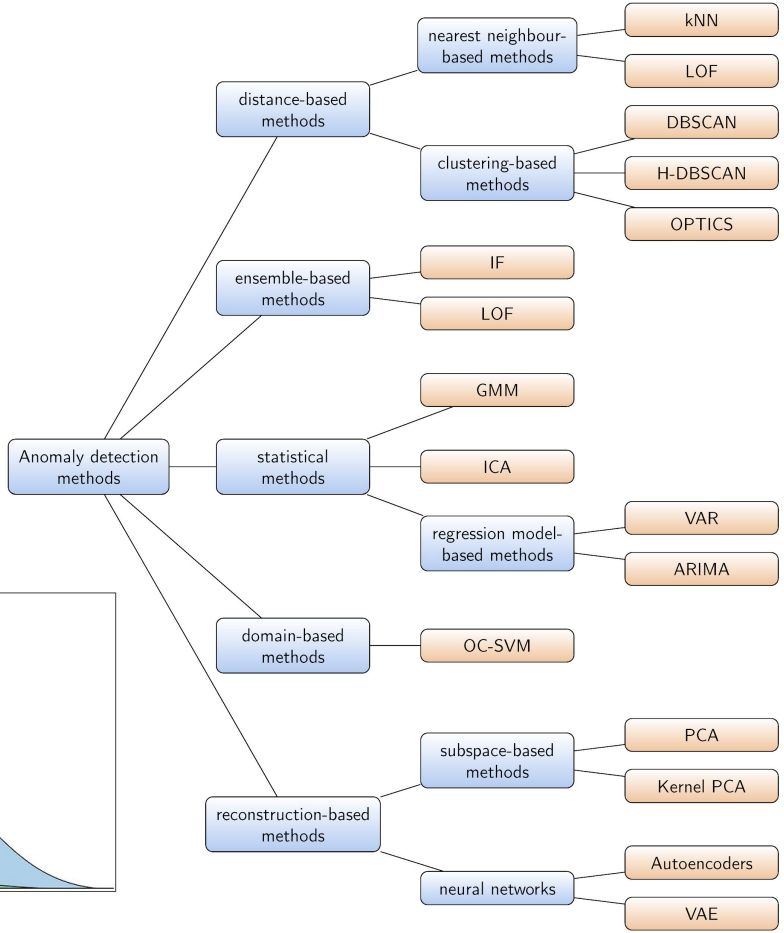
- Ph.D. in HPRC High Performance Reconfigurable Computing,
- AGH University of Science and Technology, Poland, Krakow
- Worked at CERN in 2016/17,
- Cooperation with CERN since then ...

What is anomaly ?

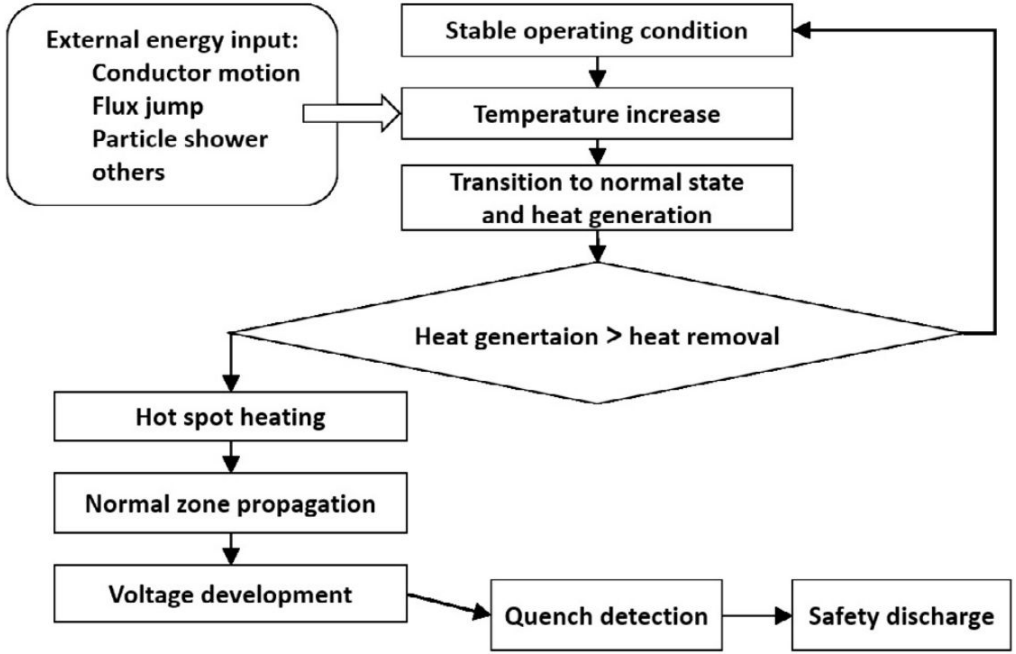




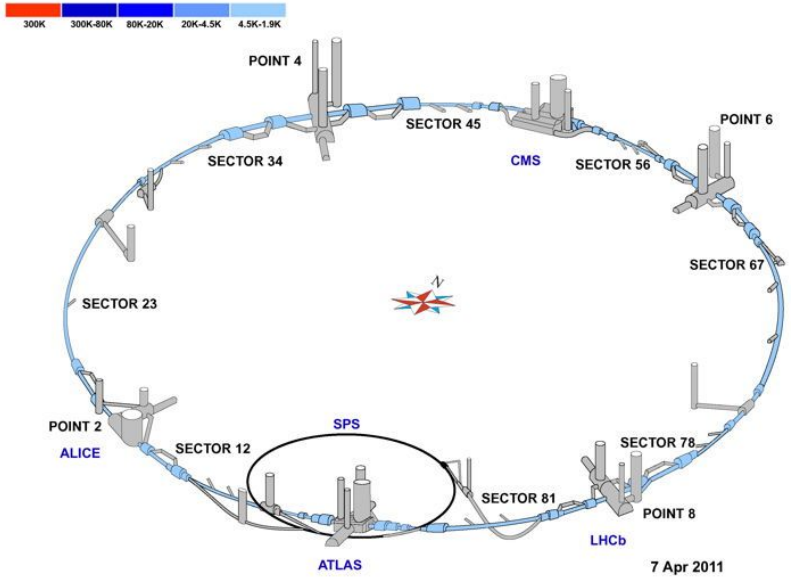


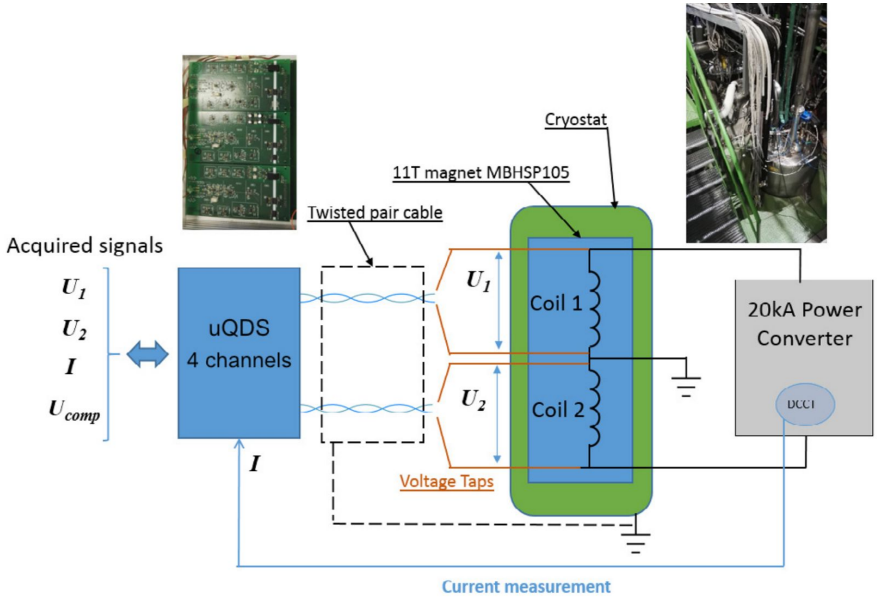


Overview


External energy input:
Conductor motion
Flux jump
Particle shower
others
Stable operating condition
Temperature increase
Transition to normal state and heat generation
Heat genertaion > heat removal
Hot spot heating
Normal zone propagation
Voltage development
Quench detection
Safety discharge

Overview - current approach



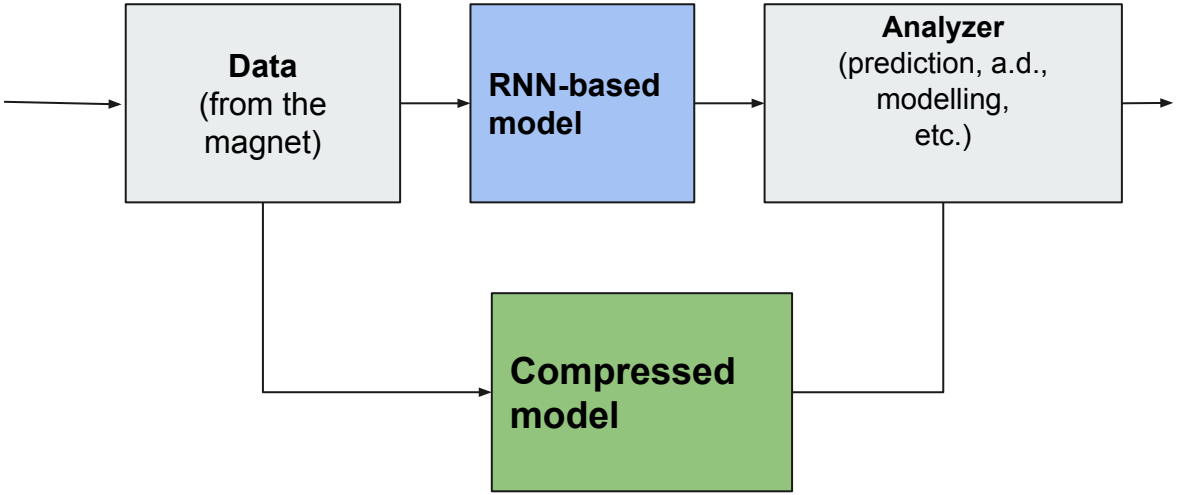


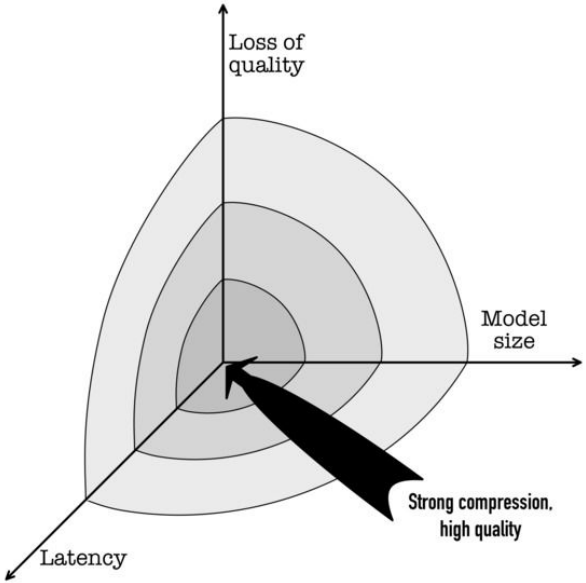
Dataset - challenges (e.g. anomaly detection)

- Asymmetry of the train data (overwhelming amount of regular data),
- very few anomalus cases available,
- very small set of test data - sometimes only few cases,
- noisy input data.

Main challenges

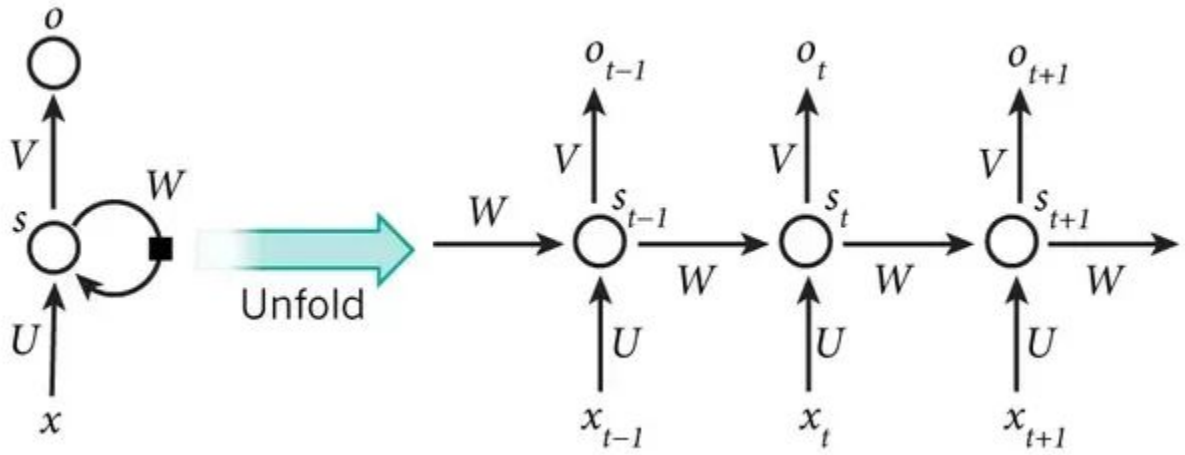
- low latency
- escaped threshold-setting
- scalable
- adjustable to data
- compressible
- easy to modify
- enables precursors detection ?

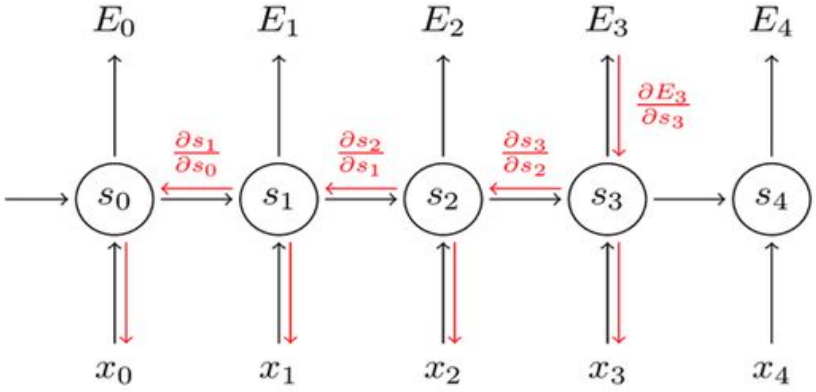




RNNs

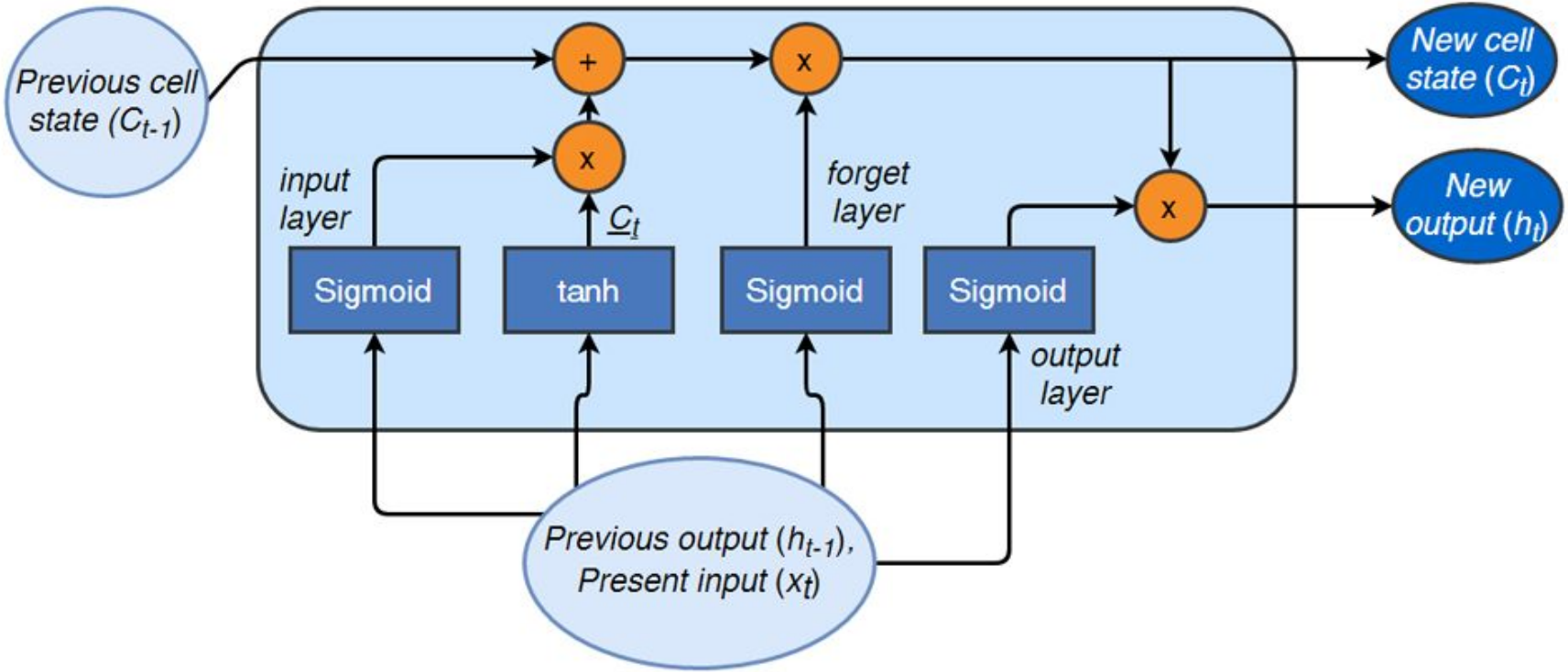
- Vanishing/exploding gradient
- Up to approx. 7 steps



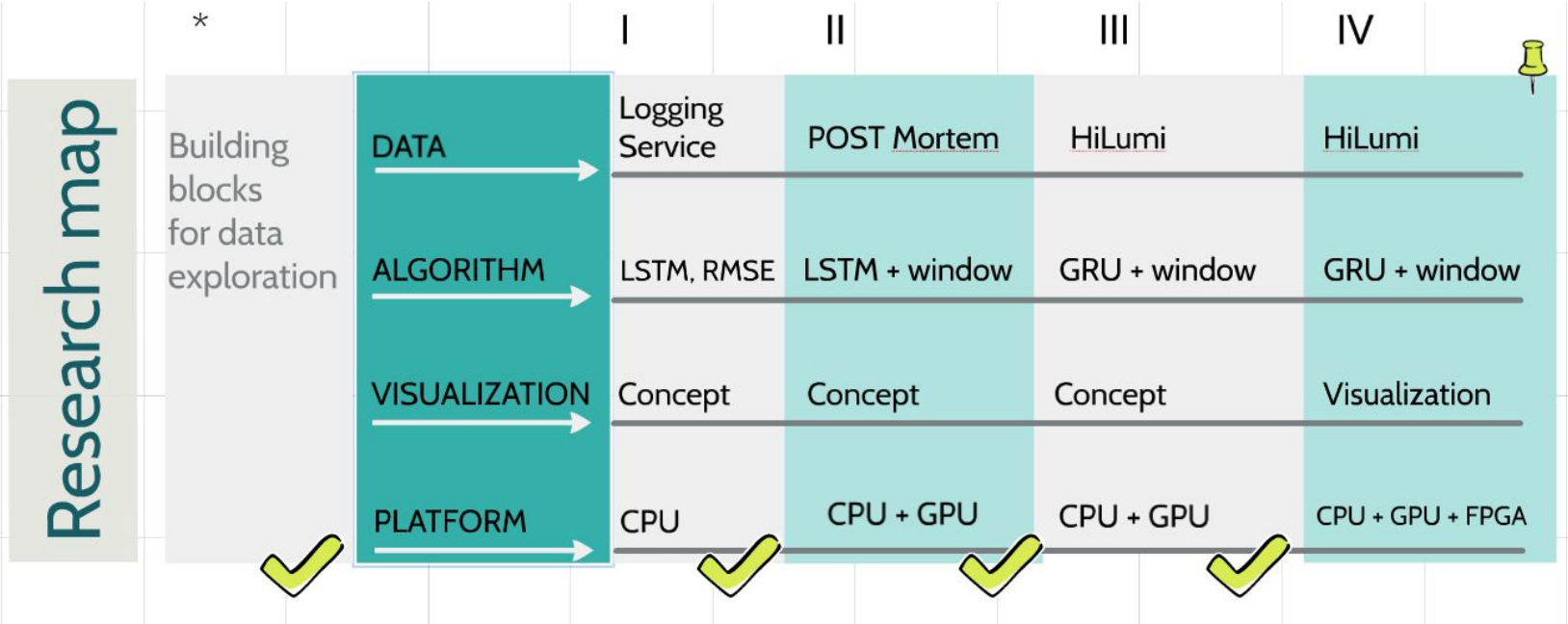


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Long Short-Term Memory (LSTM)



Research steps

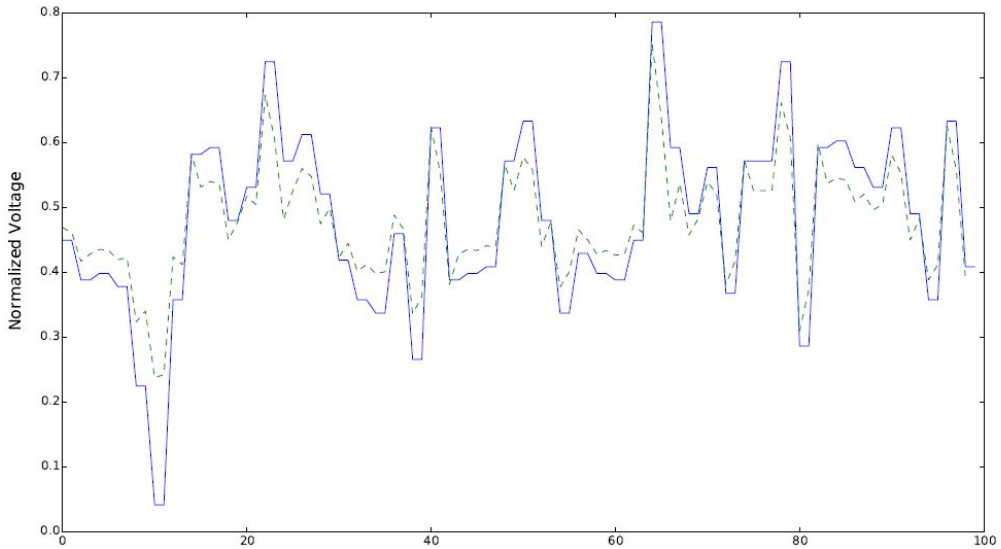


The magnets behaviour modeling

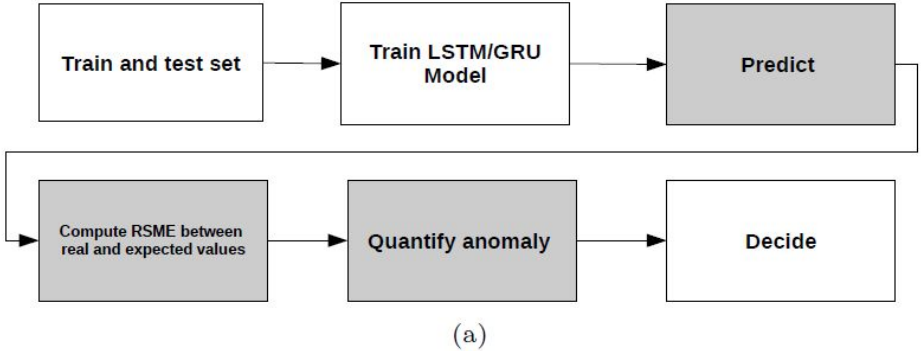
- RNNs can model magnets behaviour: „Using LSTM recurrent neural networks for monitoring the LHC superconducting magnets”, Wielgosz et al.

There are several drawbacks:

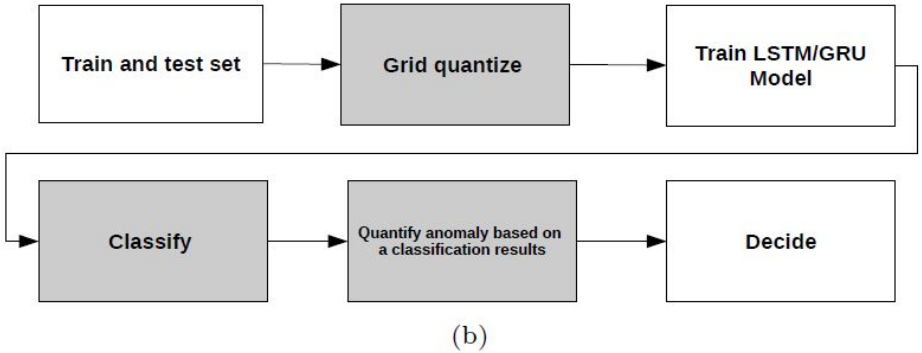
- threshold selection is arbitrary (hard to establish them),
- the resolution of such an anomaly detection depends on a window size,
- choosing too broad of a window would results in an anomaly potentially ’drowning’ in the correct data,
- choosing too small could result in false positives (not enough context),
- precursors can not be easily detected.



Architecture of the system

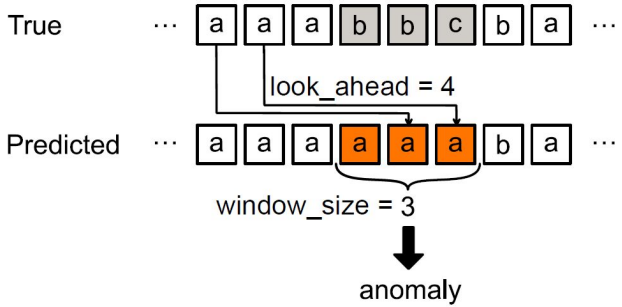


(a)

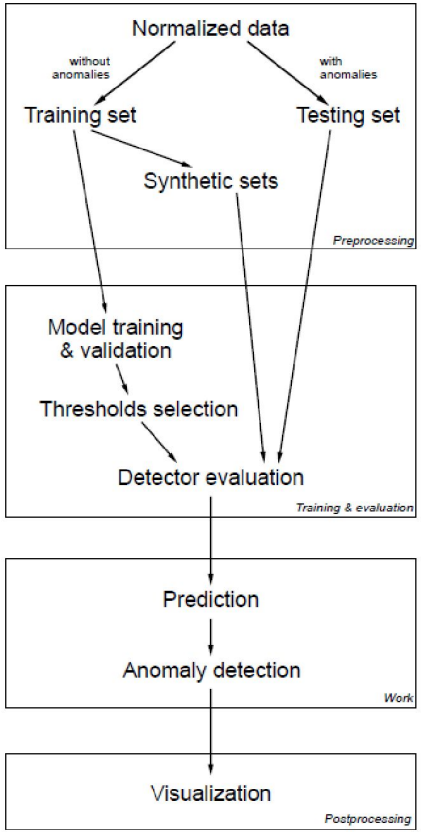


(b)

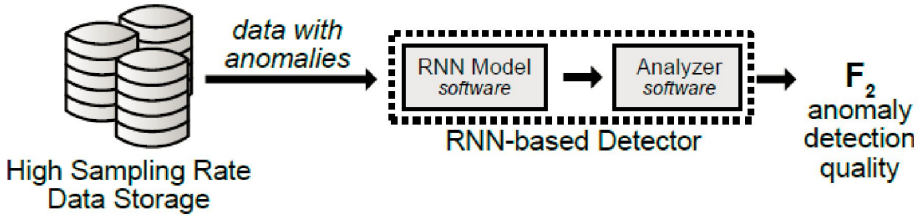
Switch from regression to a form of predictive classification ("grided regression").



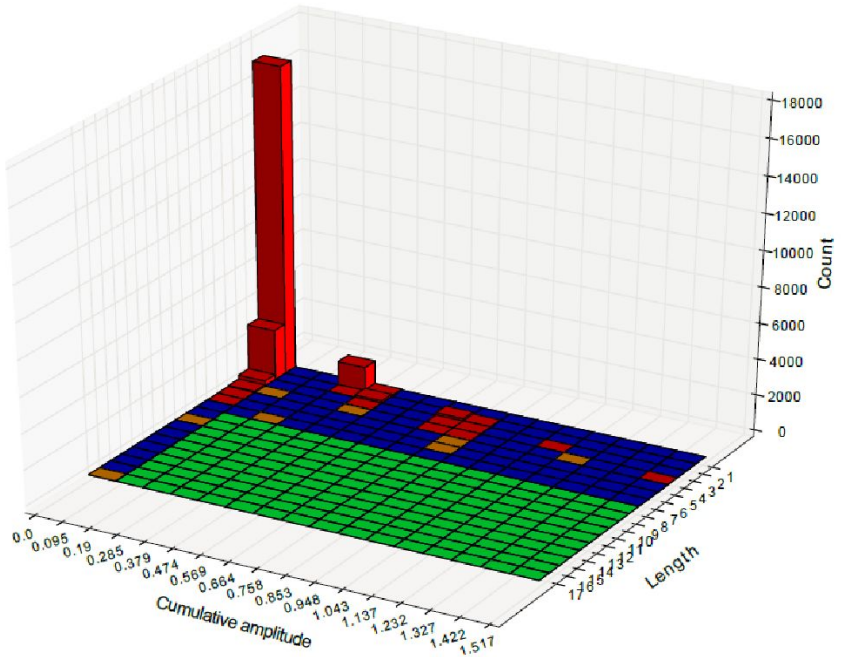
RNN-module architecture



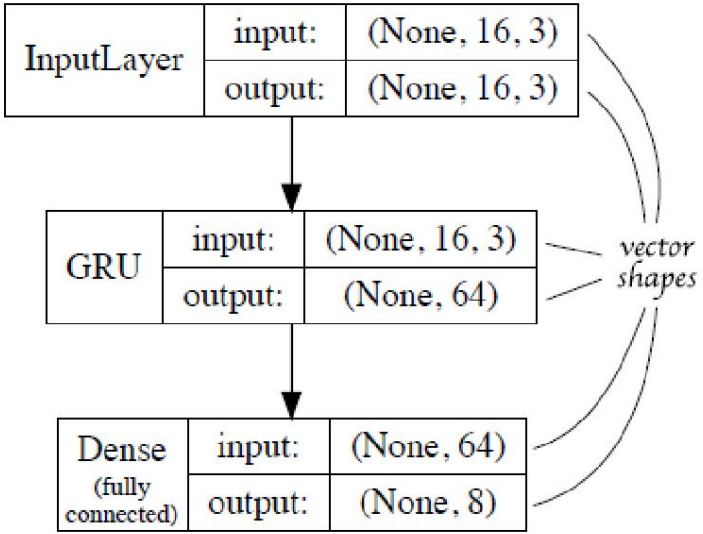
- Module is composed of model and analyzer
- Steps
  - Choice of in_grid and out_grid,
  - training the best possible model - on regular data (without anomaly) should perform well,
  - setting macro-parameters for an analyzer such as anomaly length or cumulative amplitude,
  - iterate the procedure of model training and analyzer macro-parameters tuning till there are no anomalies detected in the training dataset.



Analyzer hyper-parameters adjusting

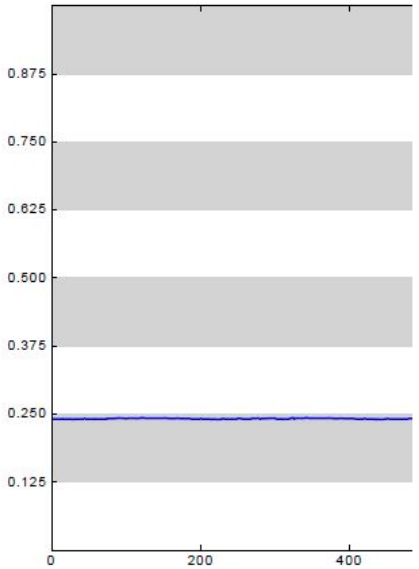


Model

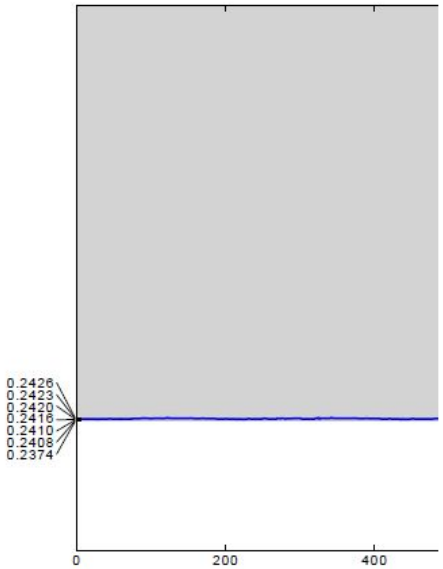


	samples (in millions)		
series	training	testing	total
h1144	~3.8	~1.2	~5
h1011	~5.1	~0.4	~5.5
h1451	~4.5	~0.5	~5
h1819	~5.7	~0.3	~6

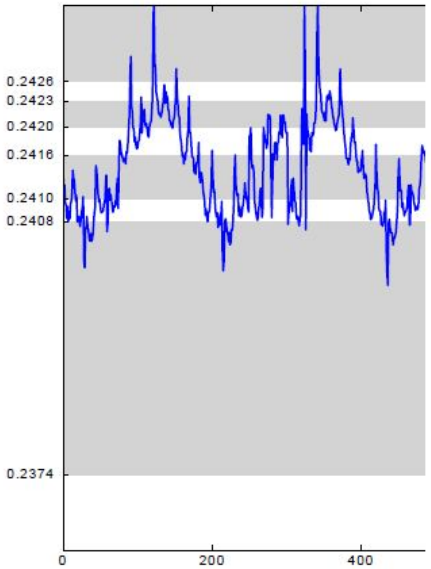
Static and adaptive grid



(a) Static grid

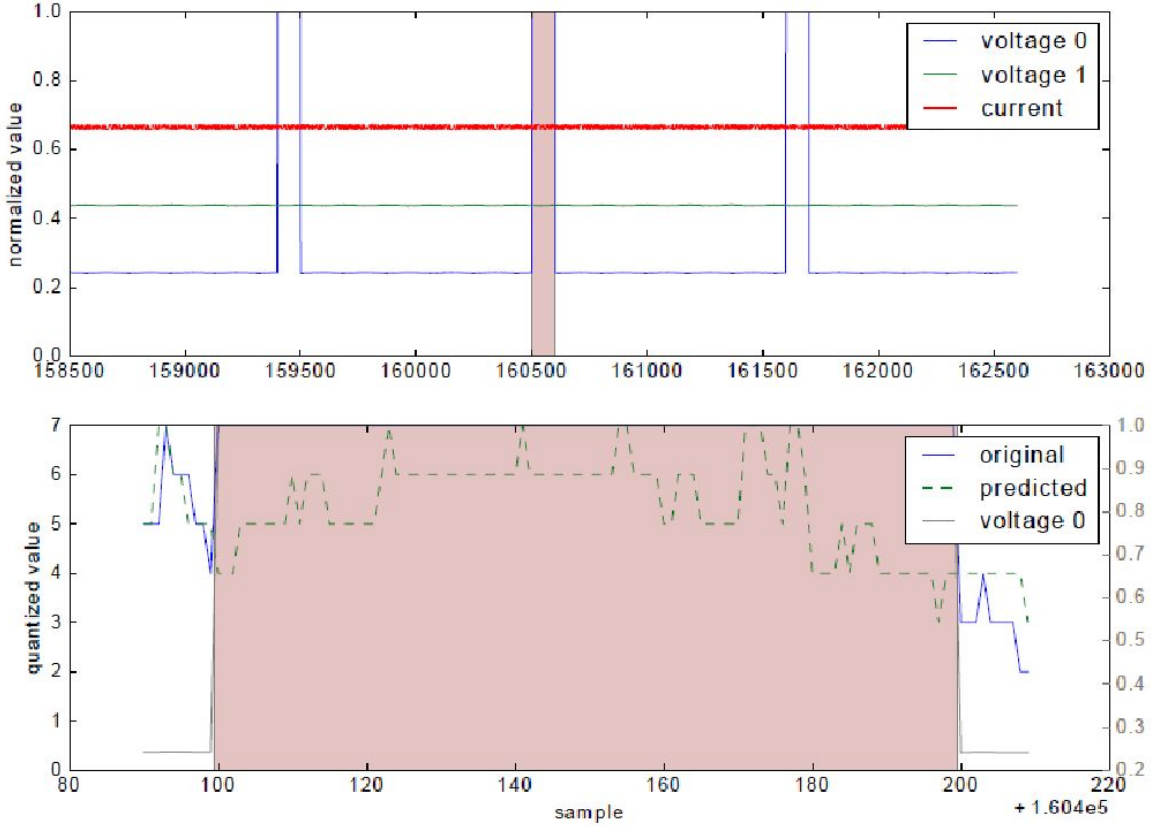


(b) Adaptive grid

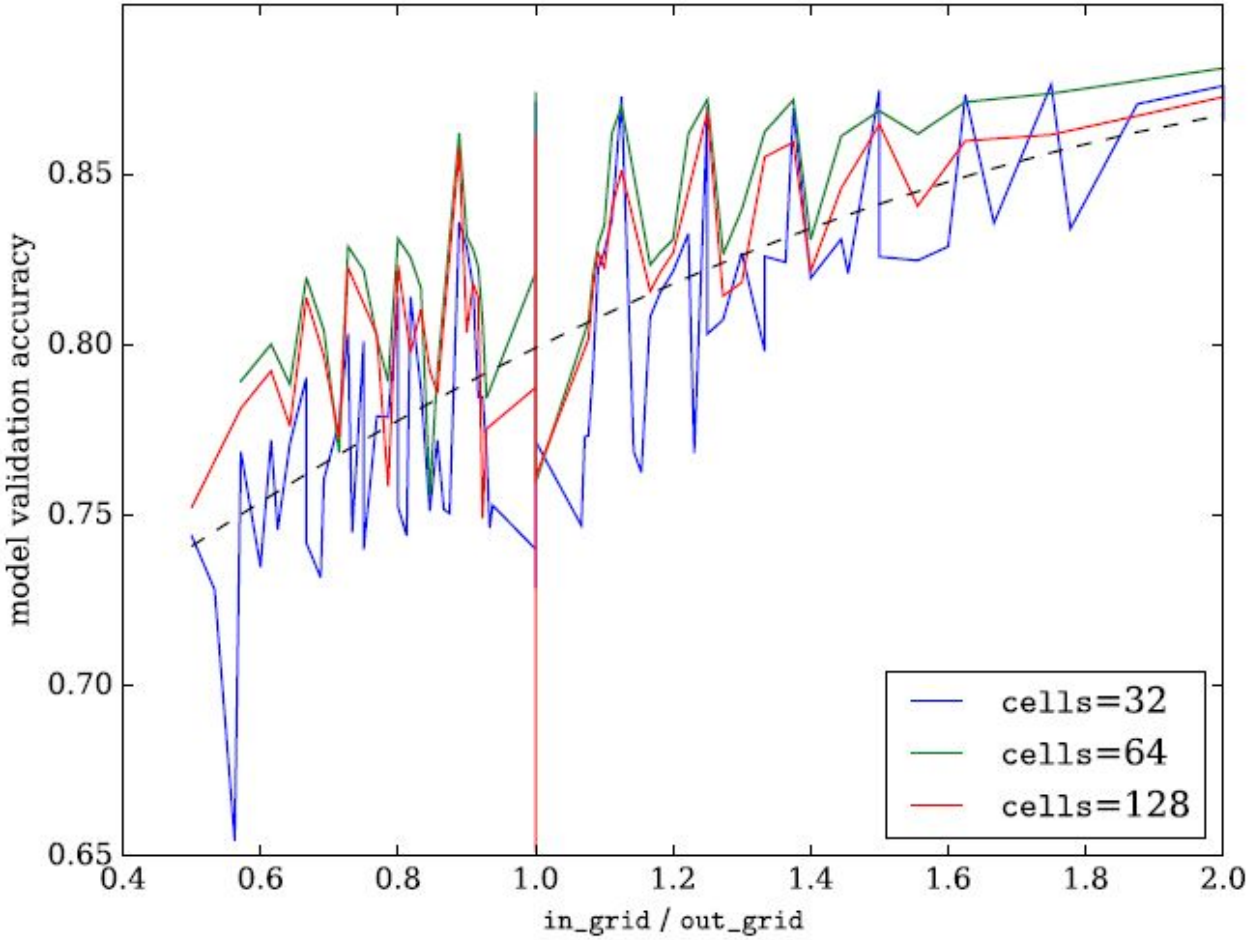


(c) Zoom of adaptive grid

8 bins


voltage 0
voltage 1
current
normalized value
1.0
0.8
0.6
0.4
0.2
0.0
158500
159000
159500
160000
160500
161000
161500
162000
162500
163000
original
predicted
voltage 0
quantized value
normalized value
sample
+ 1.604e5
7
6
5
4
3
2
1
0
80
100
120
140
160
180
200
220
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

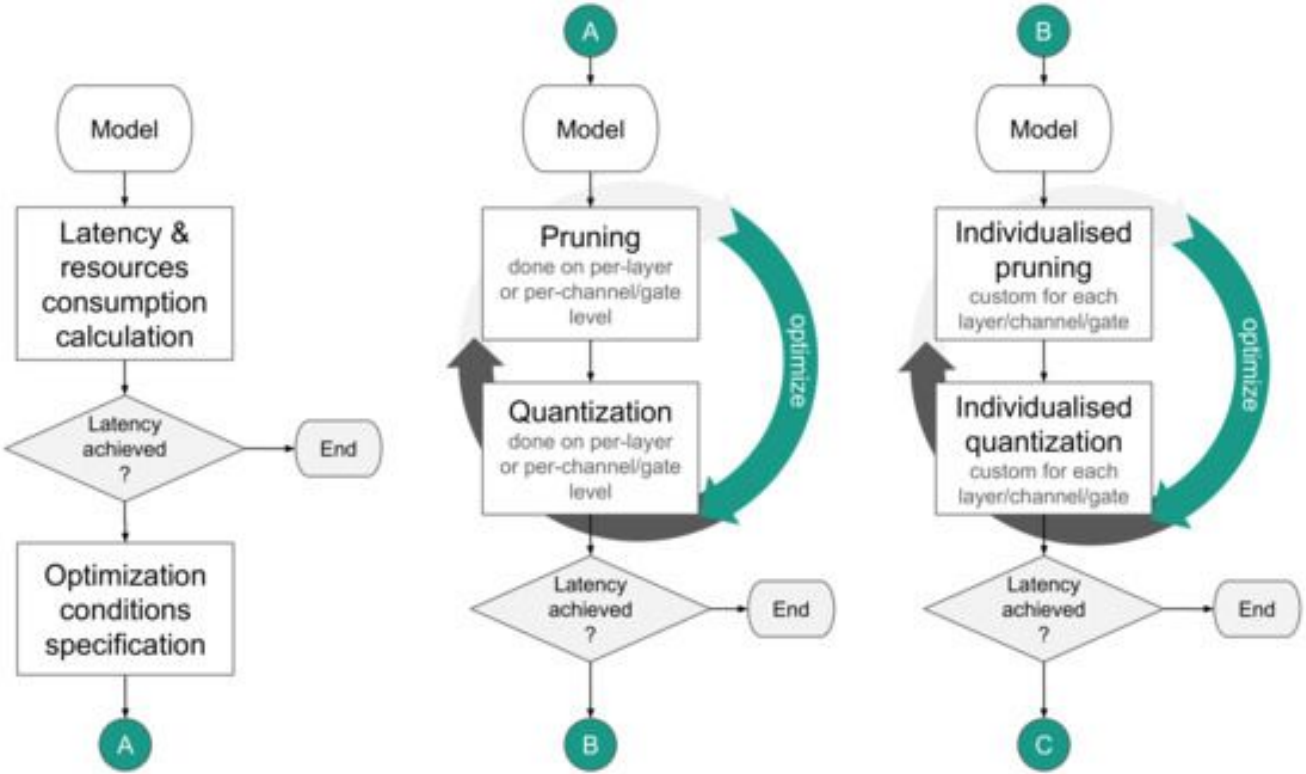
		best_length	best_cum_amp	best_max_amp	best_accuracy	balanced
Parameters	in_grid	16	8	16	16	16
	out_grid	8	8	16	8	8
	look_back	16	16	16	32	8
	cells	64	64	512	64	64
Model accuracy	train	0.8955	0.8828	0.7634	**0.8978**	0.8756
	validation	0.8809	0.8654	0.7553	**0.8812**	0.8551
Saved area thresholds	length	10	**9**	20	14	11
	cum_amp	1.5503	1.5520	1.6474	**0.7755**	**0.7755**
	max_amp	0	0	0	0	0
Real set	recall	1.0	1.0	1.0	1.0	1.0
	precision	**1.0**	0.0046	0.0005	**1.0**	**1.0**
	F_1	**1.0**	0.0091	0.0009	**1.0**	**1.0**
	F_2	**1.0**	0.0225	0.0023	**1.0**	**1.0**
Synthetic set I	recall	0.9974	0.9950	0.6848	**0.9979**	0.8312
	precision	**1.0**	**1.0**	**1.0**	0.9986	**1.0**
	F_1	**0.9987**	0.9975	0.8129	0.9982	0.9078
	F_2	0.9979	0.9960	0.7309	**0.9980**	0.8602
Synthetic set II	recall	**0.8685**	0.8184	0.4340	0.8643	0.5045
	precision	**1.0**	0.9988	**1.0**	0.9977	**1.0**
	F_1	**0.9296**	0.8996	0.6053	0.9262	0.6706
	F_2	**0.8920**	0.8490	0.4894	0.8880	0.5600


model validation accuracy
0.85
0.80
0.75
0.70
0.65
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
in_grid / out_grid
cells=32
cells=64
cells=128

Challenges of hardware neural models implementation

- co-simulation,
- maintenance,
- optimization,
- Reliable measure of performance boost vs accuracy degradation,
- Extraction of the modules for further low-level optimization,
- Fixed-point implementation in efficient and handy fashion.

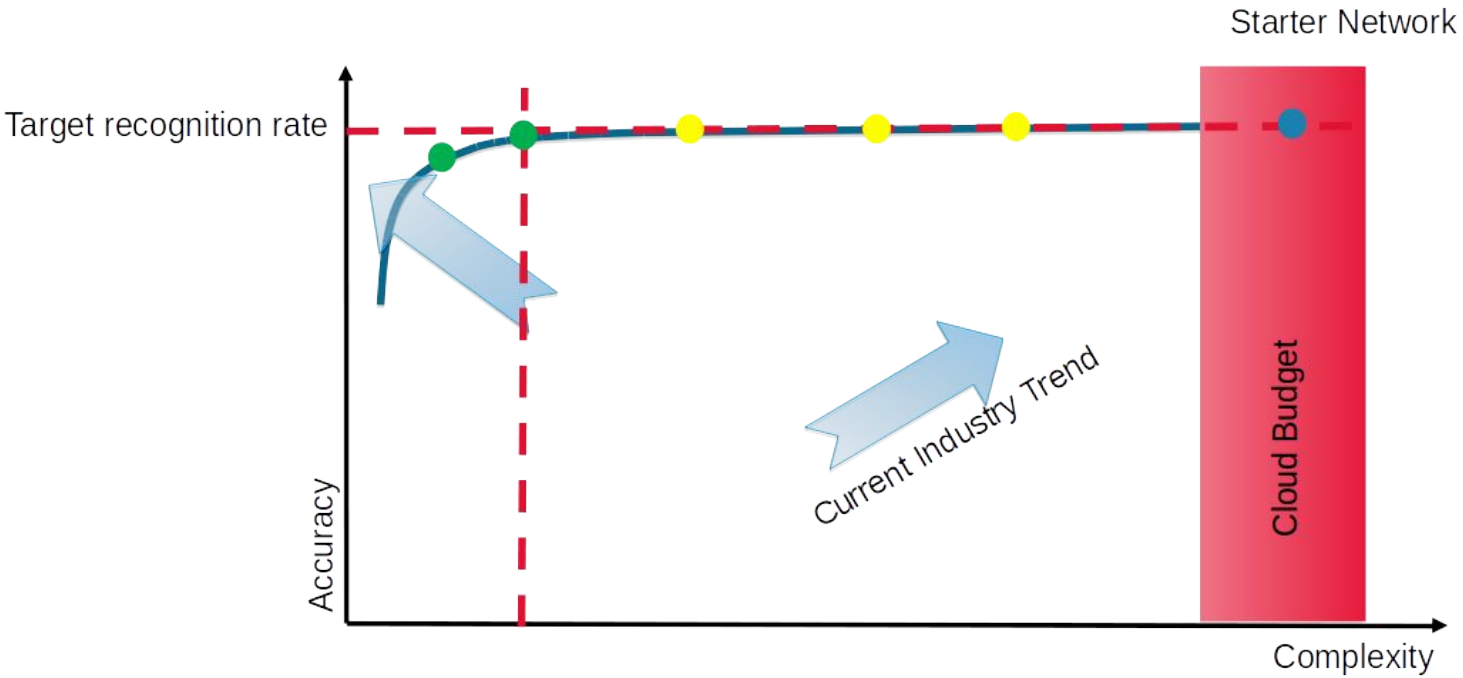
DL models compression



Sample models compression

Network	Original Size	Compressed Size	Compression Ratio	Original Accuracy	Compressed Accuracy
Lenet-300-100	1070KB	27KB	**40x**	98.36%	98.42%
Lenet-5	1720KB	44Kb	**39x**	99.20%	99.26%
AlexNet	240MB	6.9MB	**35x**	80.27%	80.30%
VGGNet	550MB	11.3MB	**49x**	88.68%	89.09%
GoogleNet	28MB	2.8MB	**10x**	88.90%	88.92%
SqueezeNet	4.8MB	0.47MB	**10x**	80.32%	80.35%

Compression of neural models



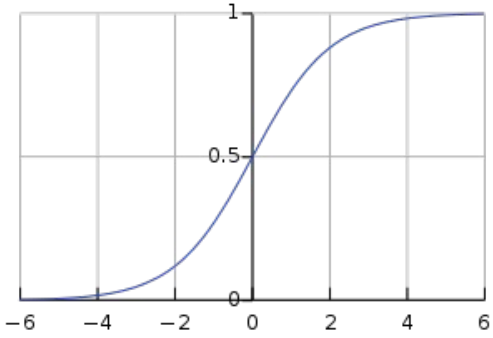
GRU - quantization

Table 5. Coefficients Quantization Results for GRU (two layers, 64 and 32 cells) + Dense, trained on four input channels. Accuracy as a function of bit-width.

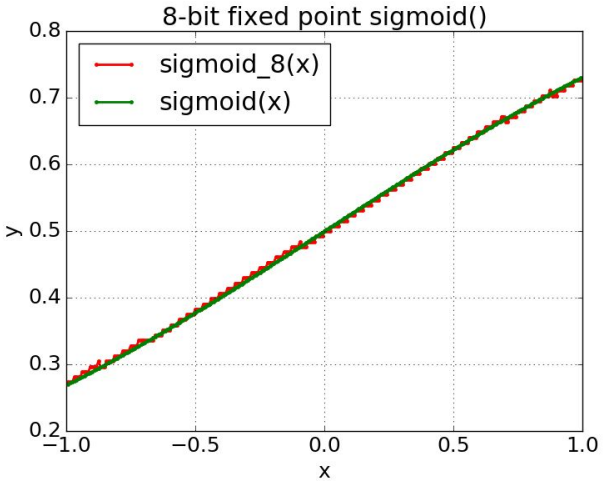
		in_algorithm		
Bits	Method	*adaptive*	*recursive_adaptive*	*cumulative_amplitude*
original model		0.9235	0.9300	0.8842
10	*linear*	0.9236	0.9287	0.8841
	minmax	0.9233	0.9300	0.8841
	log_minmax	0.9235	0.9298	0.8842
	tanh	0.9232	0.9283	0.9232
9	*linear*	0.9236	0.9279	0.8838
	minmax	0.9237	0.9295	0.8842
	log_minmax	0.9231	0.9293	0.8843
	tanh	0.9219	0.9260	0.8842
8	*linear*	0.9206	0.9257	0.8830
	minmax	0.9238	0.9311	0.8838
	log_minmax	0.9207	0.9283	0.8844
	tanh	0.9161	0.9143	0.8836
7	*linear*	0.9177	0.3989	0.8850
	minmax	0.9194	0.9250	0.8841
	log_minmax	0.9218	0.9236	0.8833
	tanh	0.9131	0.9033	0.8851
6	*linear*	0.8952	0.9008	0.8871
	minmax	0.9144	0.8839	0.8842
	log_minmax	0.9111	0.9076	0.8844
	tanh	0.8702	0.8782	0.8788

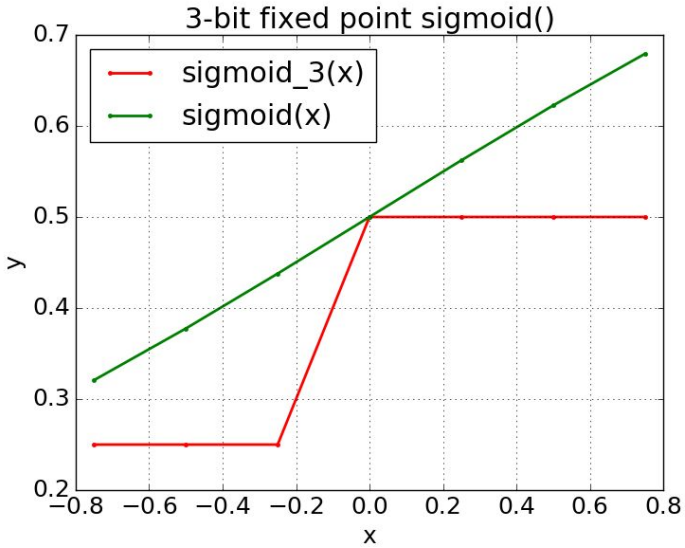
sigmoid(x)

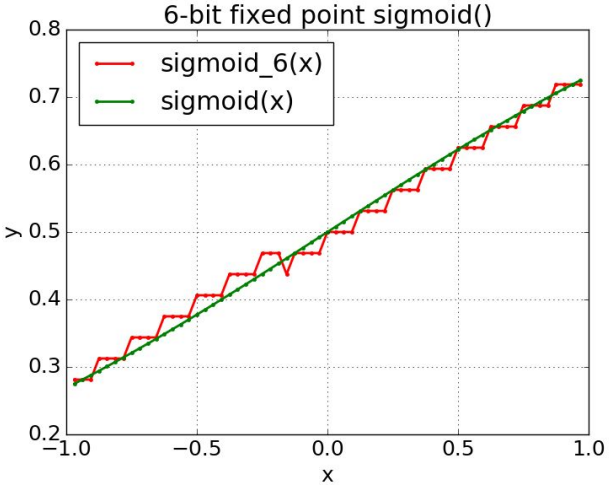
$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$



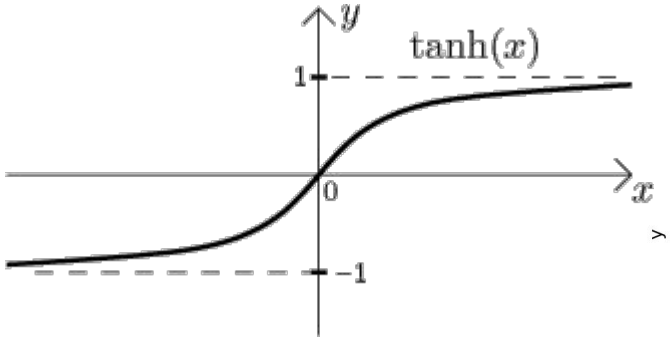

```
sigmoid(x) = (1 / 2.0) + (x / 4.0) - (x ** 3 / 48.0) + (x ** 5 / 480.0)
```

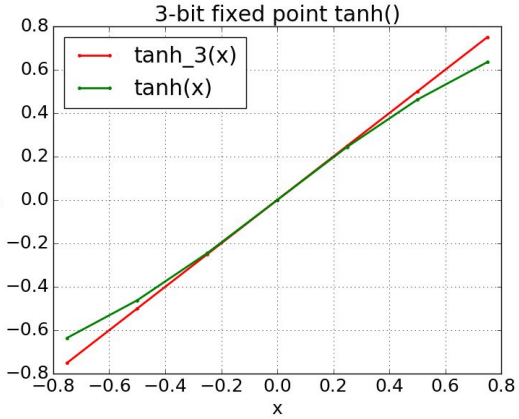




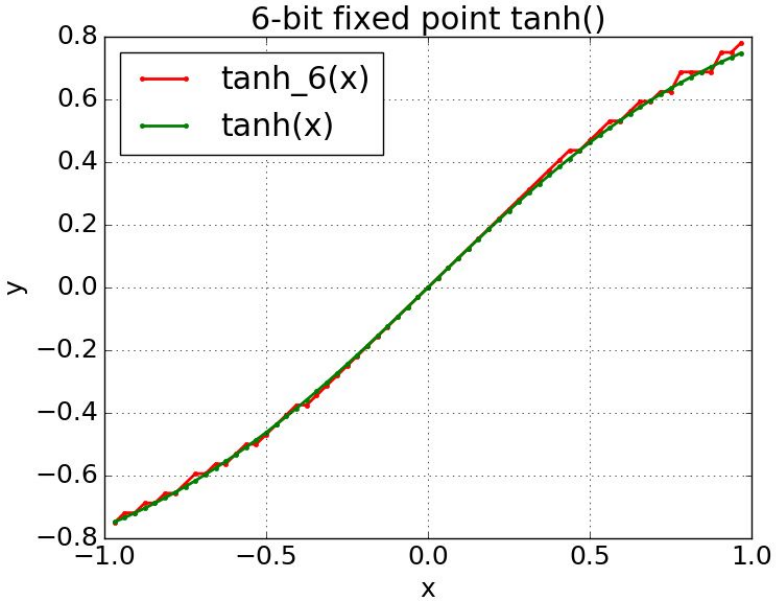


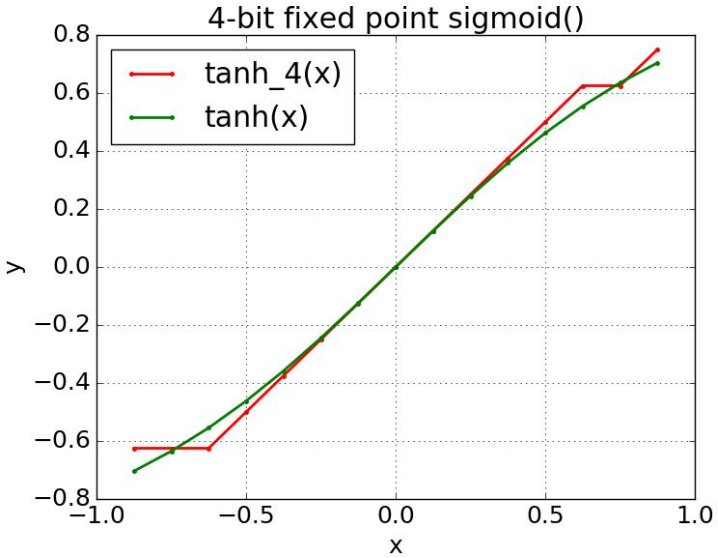
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$





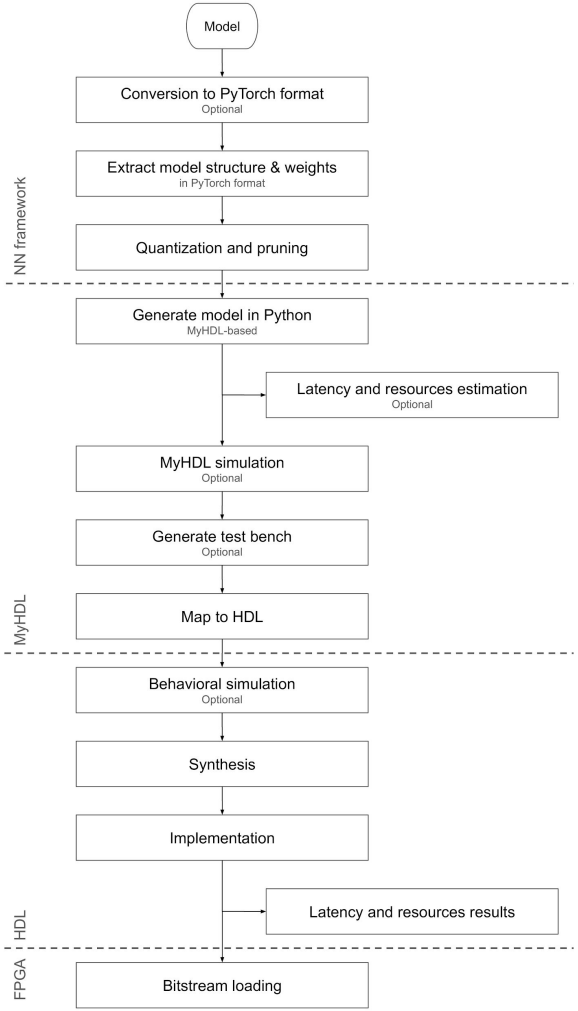
*tanh(x) = x - (1 / 3 * x ** 3) + (2 / 15 * x ** 5)*





tanh(x) FPGA

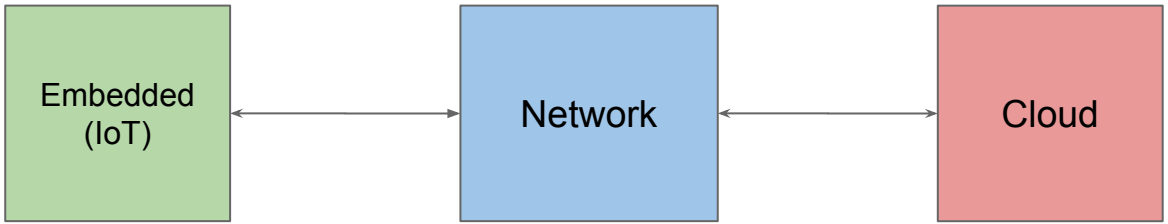
tanh(x)	seq	Zynq-7000, Clg225, Vivado 2017.1		
bits	luts		ff	
#	#	[%]	#	[%]
32	6470	44.9	35	0.12
16	1591	11.05	19	0.07
8	463	3.2	11	0.04
6	298	2.06	9	0.03
4	169	1.17	7	0.02
3	98	0.67	6	0.02


Model
Conversion to PyTorch format
Optional
Extract model structure & weights
in PyTorch format
Quantization and pruning
NN framework
Generate model in Python
MyHDL-based
Latency and resources estimation
Optional
MyHDL simulation
Optional
Generate test bench
Optional
Map to HDL
MyHDL
Behavioral simulation
Optional
Synthesis
Implementation
Latency and resources results
HDL
Bitstream loading
FPGA

Thank you

Trends in ICT

- Moving from cloud to embedded
  - Especially important for latency-critical operations
- Compression of machine learning models
- Data payload reduction



LSTM hardware implementation

description	symbol	LSTM
block input	$\vec{z}_t$	$\tanh\left(\mathbf{W}_z\vec{x}_t + \mathbf{R}_z\vec{y}_{t-1} + \vec{b}_z\right)$
input gate	$\vec{i}_t$	$\sigma\left(\mathbf{W}_i\vec{x}_t + \mathbf{R}_i\vec{y}_{t-1} + \vec{b}_i\right)$
forget gate	$\vec{f}_t$	$\sigma\left(\mathbf{W}_f\vec{x}_t + \mathbf{R}_f\vec{y}_{t-1} + \vec{b}_f\right)$
memory state	$\vec{m}_t$	$\vec{i}_t \odot \vec{z}_t + \vec{f}_t \odot \vec{m}_{t-1}$
output gate	$\vec{o}_t$	$\sigma\left(\mathbf{W}_o\vec{x}_t + \mathbf{R}_o\vec{y}_{t-1} + \vec{b}_o\right)$
hidden state	$\vec{y}_t$	$\vec{o}_t \odot \tanh\left(\vec{c}_t\right)$

Results

Table 3. Testing accuracy (20 % of dataset). All models were run with `in_grid` = 32 and `look_back` = 256, using single input channel (U_{RES}), NNs were trained for 7 epochs.

	`in_algorithm`		
Model	*adaptive*	*recursive_adaptive*	*cumulative_amplitude*
Random (stratified)	0.6334	0.6334	0.6334
Elliptic Envelope	0.6700	0.7775	0.6700
Isolation Forest	0.7947	0.7596	0.8094
One-class SVM (RBF kernel)	0.3300	0.8232	0.3300
One-class SVM (linear kernel)	0.2959	0.7881	0.2528
GRU (two layers, 64 and 32 cells)	0.8928	**0.9005**	0.8842
LSTM (two layers, 64 and 32 cells)	0.8271	0.8552	0.7402