Analysis of Monte-Carlo propagator data with deep neural networks

Dániel Berényi (Wigner RCP)
Antal Jakovác (Eötvös University)

Balaton Workshop 2019, 20 June 2019, Tihany

The problem

One would like to access the particle content of a strongly interacting theory from lattice calculations

- What is measurable on the lattice is the propagator, G
- Particle mass information is encoded in the spectral function, ρ
- They are related by the integral transform:

$$G(\tau, k) = \int \frac{\mathrm{d}\omega}{2\pi} \frac{\cosh\left(\frac{\beta}{2} - \tau\right)\omega}{\sinh\frac{\beta}{2}\omega} \, \rho(\omega, k)$$

The problem

$$G(\tau, k) = \int \frac{\mathrm{d}\omega}{2\pi} \frac{\cosh\left(\frac{\beta}{2} - \tau\right)\omega}{\sinh\frac{\beta}{2}\omega} \rho(\omega, k)$$

- We need to invert this relation

- G is obtained by numerics, contains noise
  we assume $G = G_0 + z\xi\sqrt{G_0}$,
  where ξ is gaussian noise, z is the amplitude $(0, 10^{-3} \ldots 10^{-1})$

- For the purpose if this study we assume the following form of the spectral function:

$$\rho(\omega) = \sum_k c_k \delta(\omega - m_k)$$

The problem

For the purpose if this study we assume the following form of the spectral function:

$$\rho(\omega) = \sum_k c_k \delta(\omega - m_k)$$

The discretized propagator then:

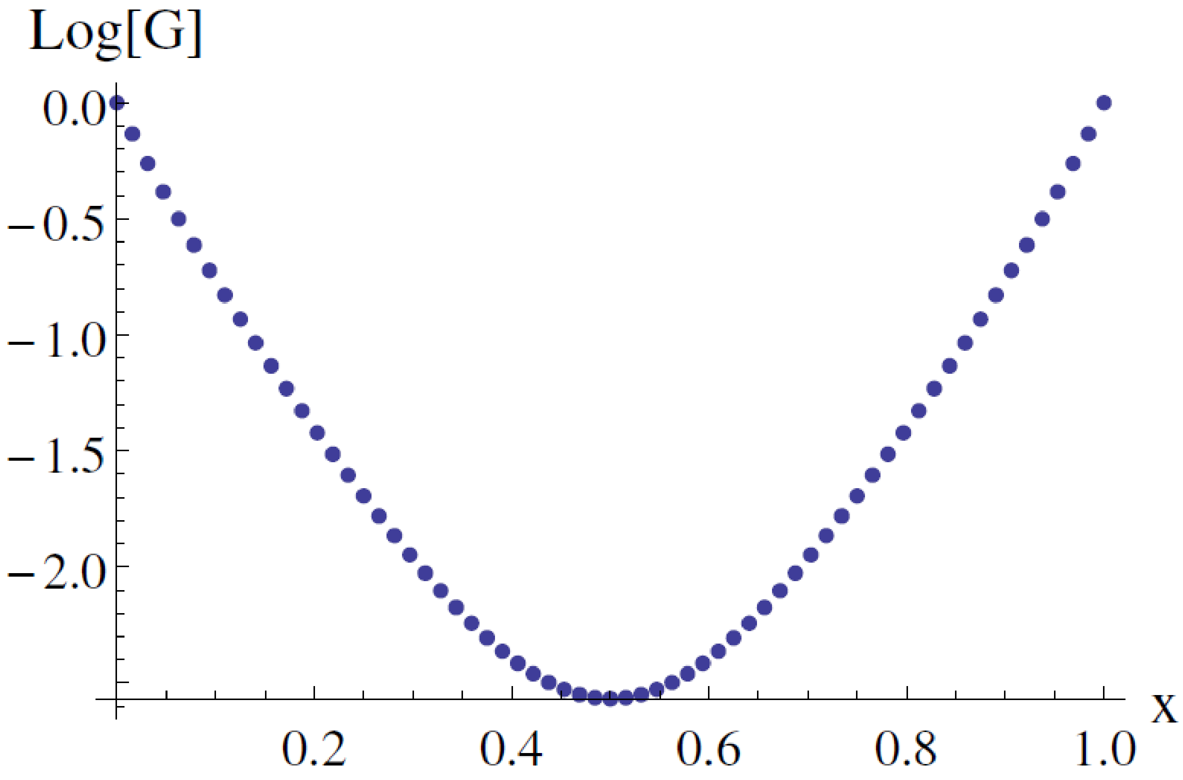
$$G_n = \sum_k \tilde{c}_k \cosh\left(\frac{\beta}{2} - \tau_n\right) m_k$$

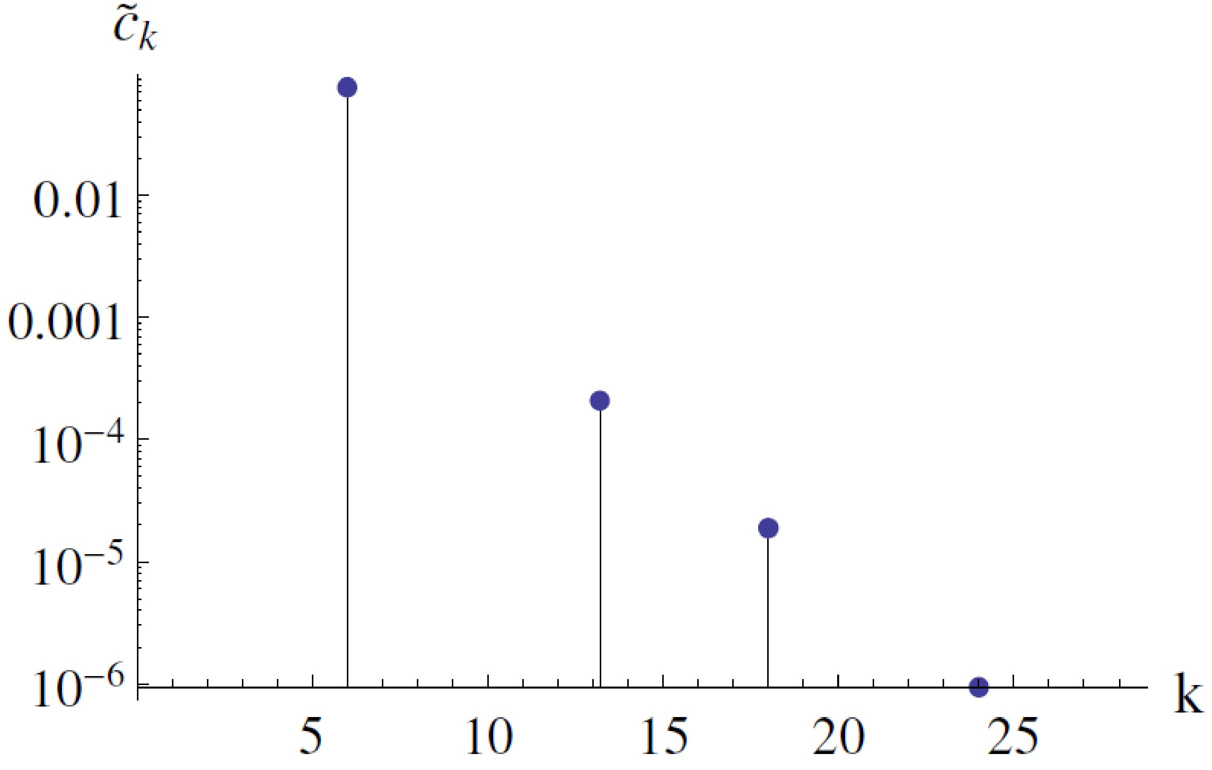
Where:

$$\tilde{c}_k = \sinh\left(\frac{\beta m_0}{2}\right) / \sinh\left(\frac{\beta m_k}{2}\right)$$

$$\tau_n = \frac{\beta n}{N_\tau}, \qquad n = \{0, 1, \dots, N_\tau - 1\}$$

Example case





$$\beta m_0 = 6, \qquad c = \{1, 0.1, 0.1, 0.1\}, \qquad \frac{m_i}{m_0} = (1, 2.2, 3, 4), \qquad N_\tau = 64$$

Methods

In this contribution we compare the following methods for reconstructing the peaks in the spectral function:

- Direct χ^2 fit
- Maximum Entropy Method
- Deep Neural network

Direct χ^2 fit

We assume the trial function:

$$G(A,m,x) = \sum_i A_i \cosh(m_i x)$$

And minimize the following functional:

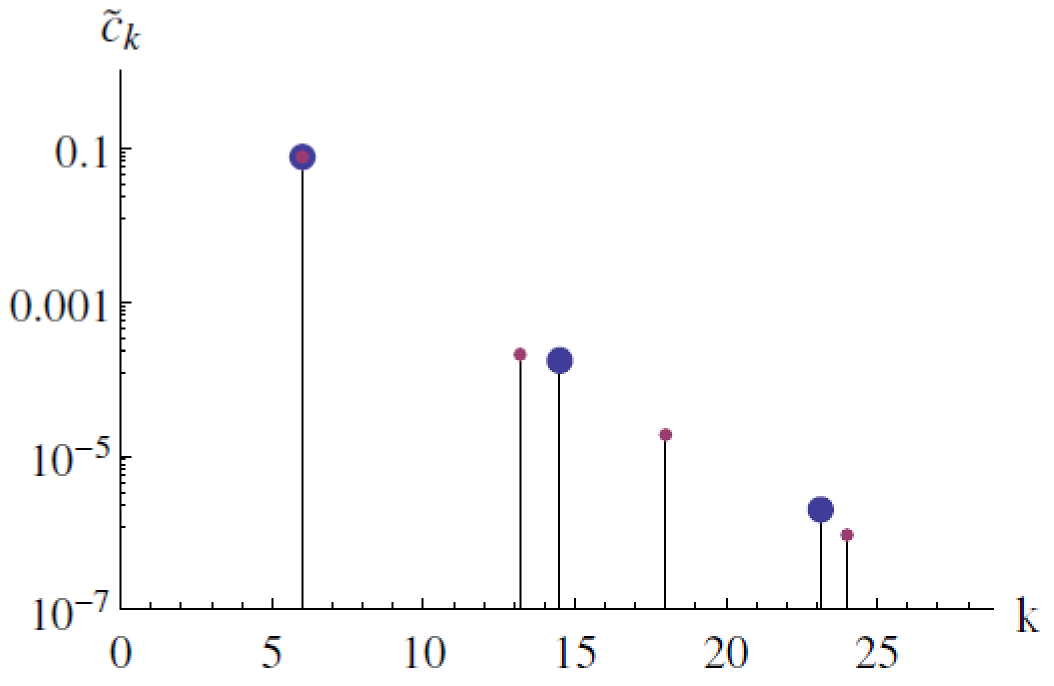
$$\chi^2 = \sum_{n=0}^{N_\tau} \frac{(G(A,m,\tau_n) - G_n)^2}{\sigma_n^2}$$

Where $\sigma_n^2 = z^2 G_n^{(0)}$

Direct χ^2 fit

Problems of this approach:

- Very sensitive to the initial guess
- Can reproduce the datapoints very precisely, with very wrong or unphysical parameters (like negative amplitude!)
- By adding noise, we start to loose peaks



$$ansatz\colon\ m_n = \begin{cases} m_0(1+1.0n) \\ m_0(1+0.9n) \end{cases}$$

Direct χ^2 fit

To sum up:

- Cannot reliably predict the parameters even for a exactly given data
- Very wide distribution of errors
- Near peaks cannot be resolved because numerical sensitivity

Maximum Entropy Method

The spectral function is constrained by physics:

$$\rho(\omega > 0, k) > 0$$

This translates to a monotonity requirement for G_n

We can enforce this, by extending the χ^2 cost function:

$$\overline{\chi^2} = \chi^2(A, m) + \alpha S_{SJ}(A)$$

where, the Shannon entropy term is:

$$S_{SJ} = \sum_m \rho_m \left(\ln \frac{\rho_m}{\rho_m^{(0)}} - 1 \right), \qquad \rho_m = \rho(\omega_m)$$

Maximum Entropy Method

$$\overline{\chi^2} = \chi^2(A, m) + \alpha S_{SJ}(A)$$

After algebraic transformations we arrive at:

$$\chi^2 = \frac{\alpha}{2}\sigma_n^2 Z_n^2 - G_n Z_n + \rho_k^{(0)} e^{Z_n K_{nk}}$$

A. Jakovác, P. Petreczky, K. Petrov, A. Velytsky,
Phys. Rev. D75, 014506.

Where:

$$\rho_k = \rho_k^{(0)} e^{Z_n K_{nk}}$$

$$K_{nk} = \cosh\left[\left(\frac{1}{2} - \frac{n}{N_\tau}\right)\frac{k}{N_\omega}\omega_{max}\right]$$

Instead of N_ω size we need to solve N_τ

Maximum Entropy Method

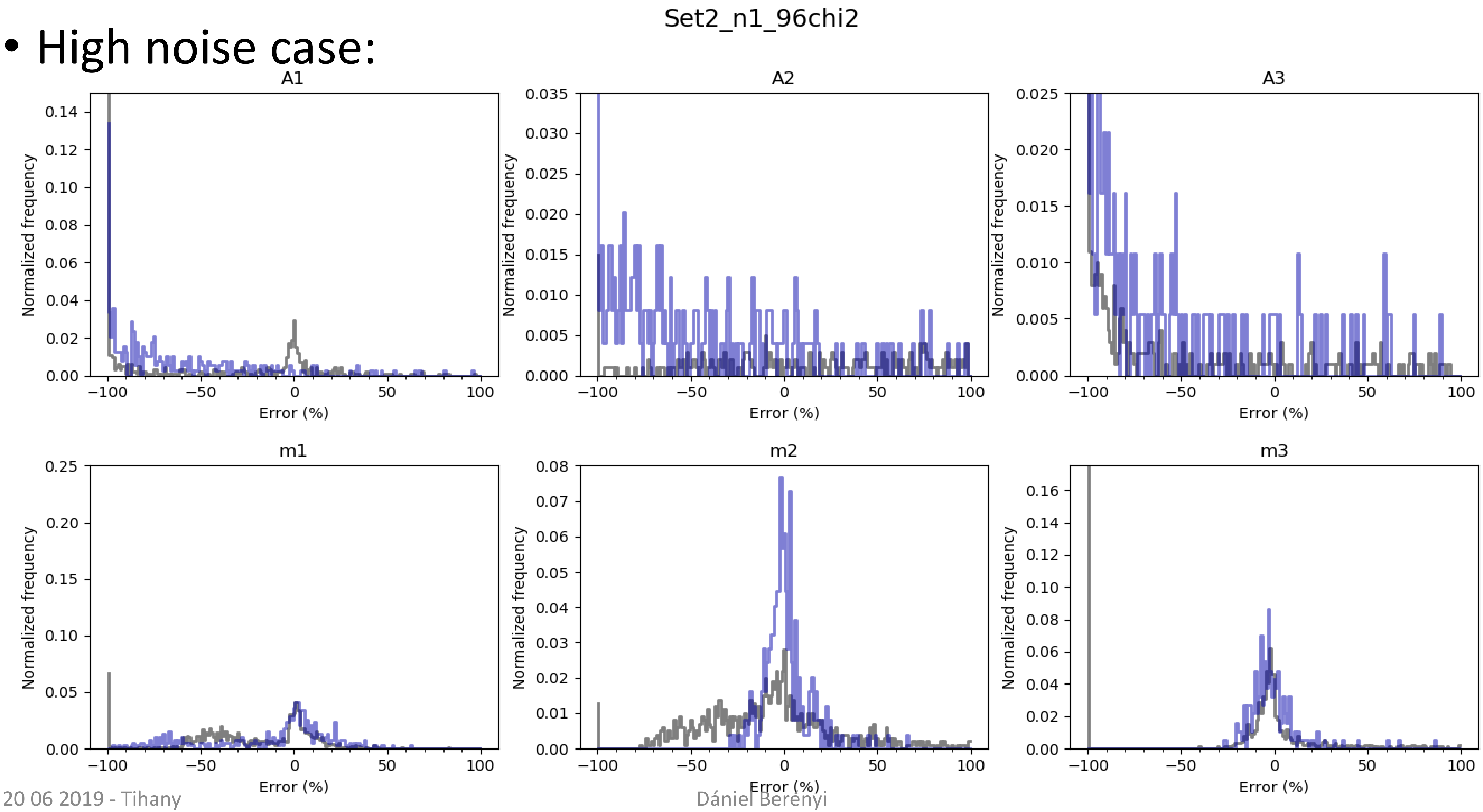
Results:

- Somewhat smaller errors
- Non-negative peak amplitudes
- 3rd peak is missing frequently
- Very precise arithmetic is needed (α need to be tuned)

But, it works even if the precise form of the spectral function is not known (e.g. number of peaks)

Mem and χ^2

- High noise case:



Deep Neural Networks

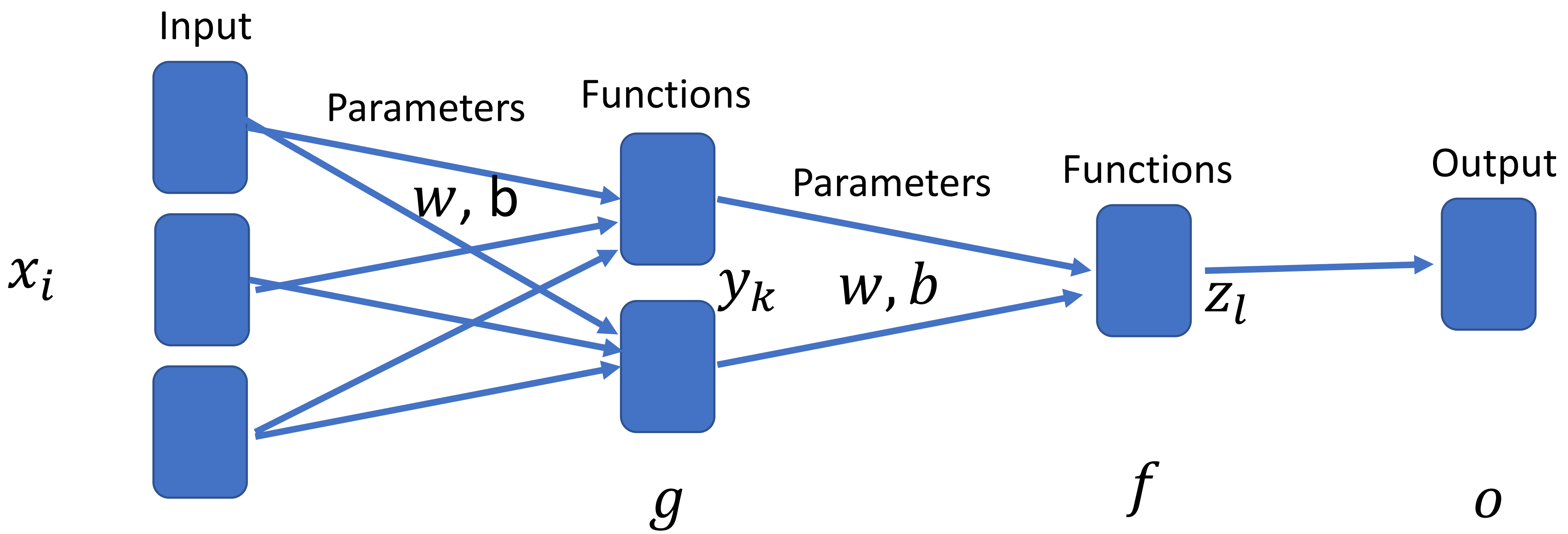
Neural Networks are a new tool for prediction/fitting tasks

- A Neural Network is just a differentiable function composition sequence, that have parameters inside, that we minimize against some cost function

- Due to its universal approximator properties, they can be optimized to fit large classes of functions

- In this method we do NOT assume any functional form, we construct a network, and feed in input-output data pairs

Deep Neural Networks



$$o = (f \circ g)(x_i)$$

$$net = f \circ g$$

$$y_k = g\left(\sum_i w_{ik} x_i + b_k\right) \quad z_l = f\left(\sum_k w_{kl} y_k + b_l\right)$$

f, g is usually a non-linear function

Deep Neural Networks

- At the end we use some loss function, like:

$$L_0(u, v) = \frac{1}{2}(u - v)^2$$

- If we fix the expected output (o) in the cost we have:

$$L(v) = L_0(o, v)$$

- So now we have the composition:

$$e(x) = (L \circ f \circ g)(x)$$

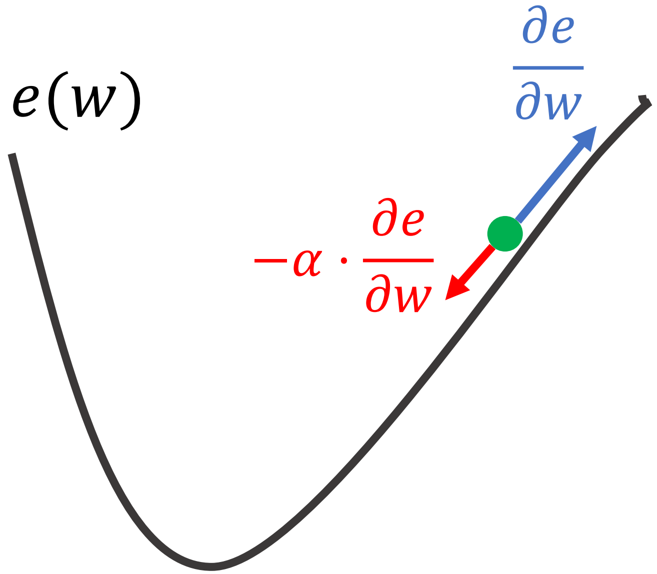
- And we would like to minimize it:

$$\frac{\partial e}{\partial w_{ij}} = 0 \quad \frac{\partial e}{\partial b_i} = 0$$

Deep Neural Networks

Most of the methods are variations of the Gradient Descent:

$$w^{(n+1)} = w^{(n)} - \alpha \cdot \frac{\partial e}{\partial w}$$



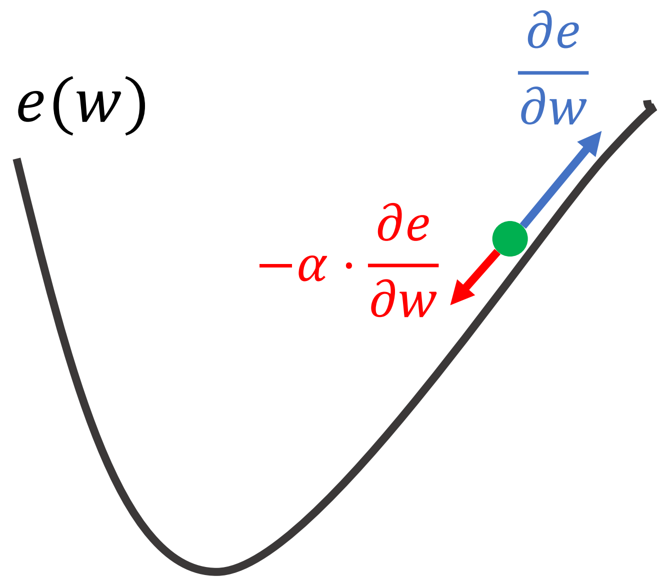
where α is the „learning rate”

Automatic differentiation can be used to produce the required gradients at each function in the net (“back propagation”)

Deep Neural Networks

When training, we take some examples from the “training set”

$$s_q = (x_{i_q}, o_q), q = 1 \dots N$$



and perform an update of the weights:

$$w^{(n+1)} = w^{(n)} - \alpha \cdot \sum_q^n \frac{\partial e(x_{i_q})}{\partial w}$$

And repeat until we reach some low enough cost.

Deep Neural Networks

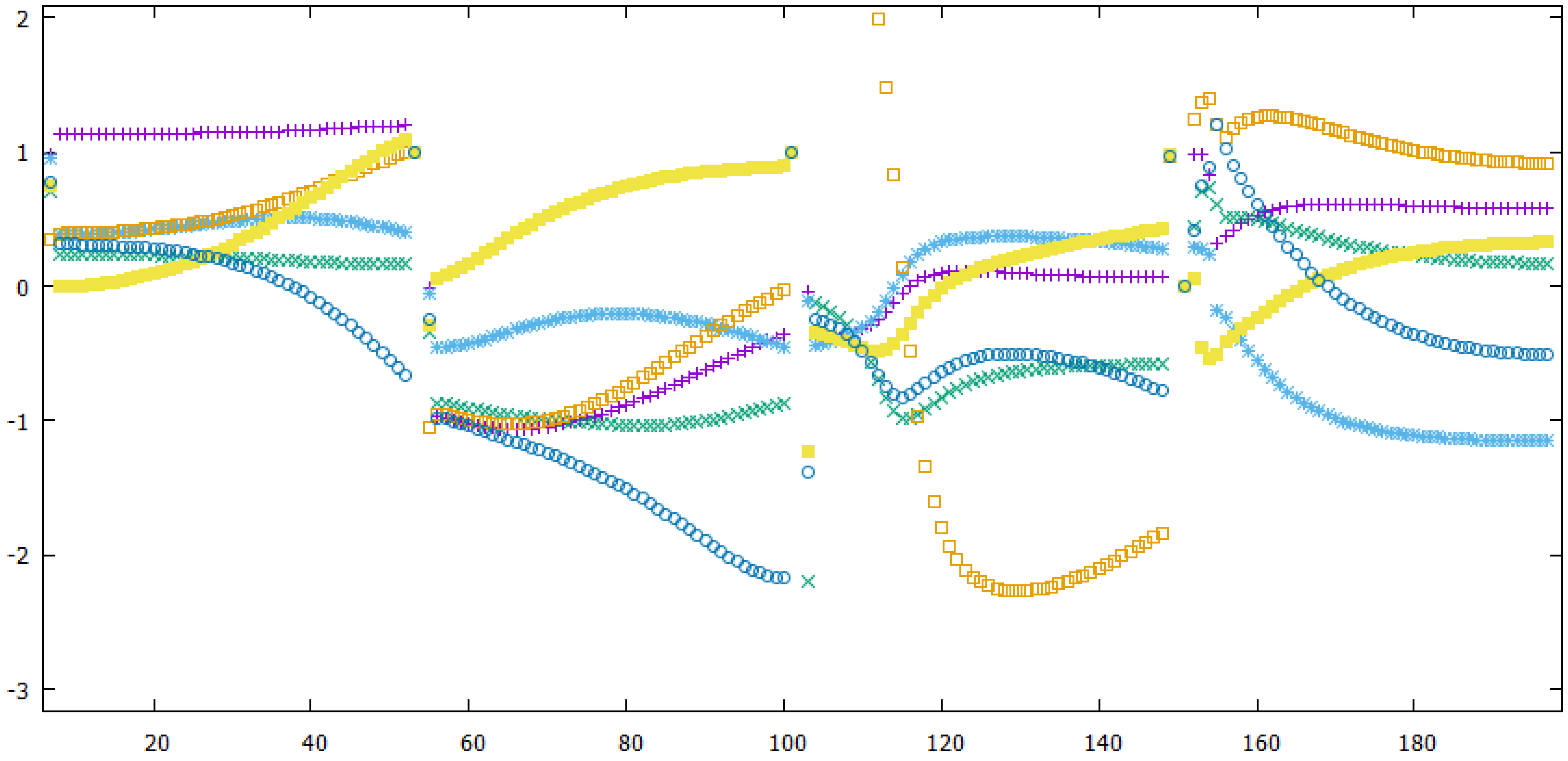
In our case:

- we generated large number (400k) of 3 component propagator realizations with different noise
- Took some network of composed affine transformations (4-6 layers)
- Trained to predict the $\{A_k, m_k\}$ parameters

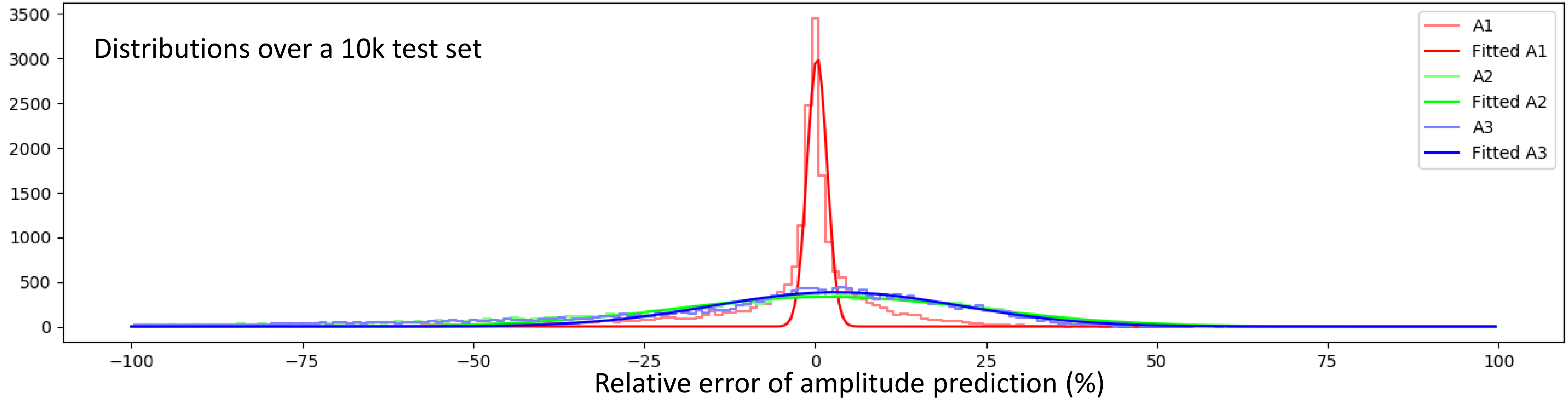
- We did some preprocessing (folding G in half, augmentimg the dataset with logG, integral of logG and the fourier transform of logG)

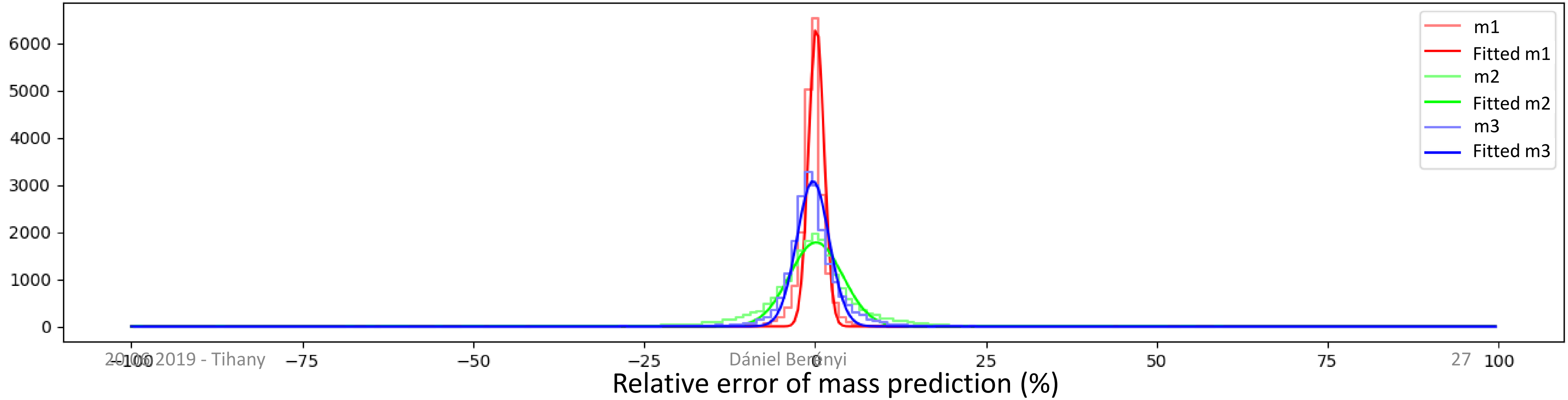
Deep Neural Networks

Sample training data:



Deep Neural Networks

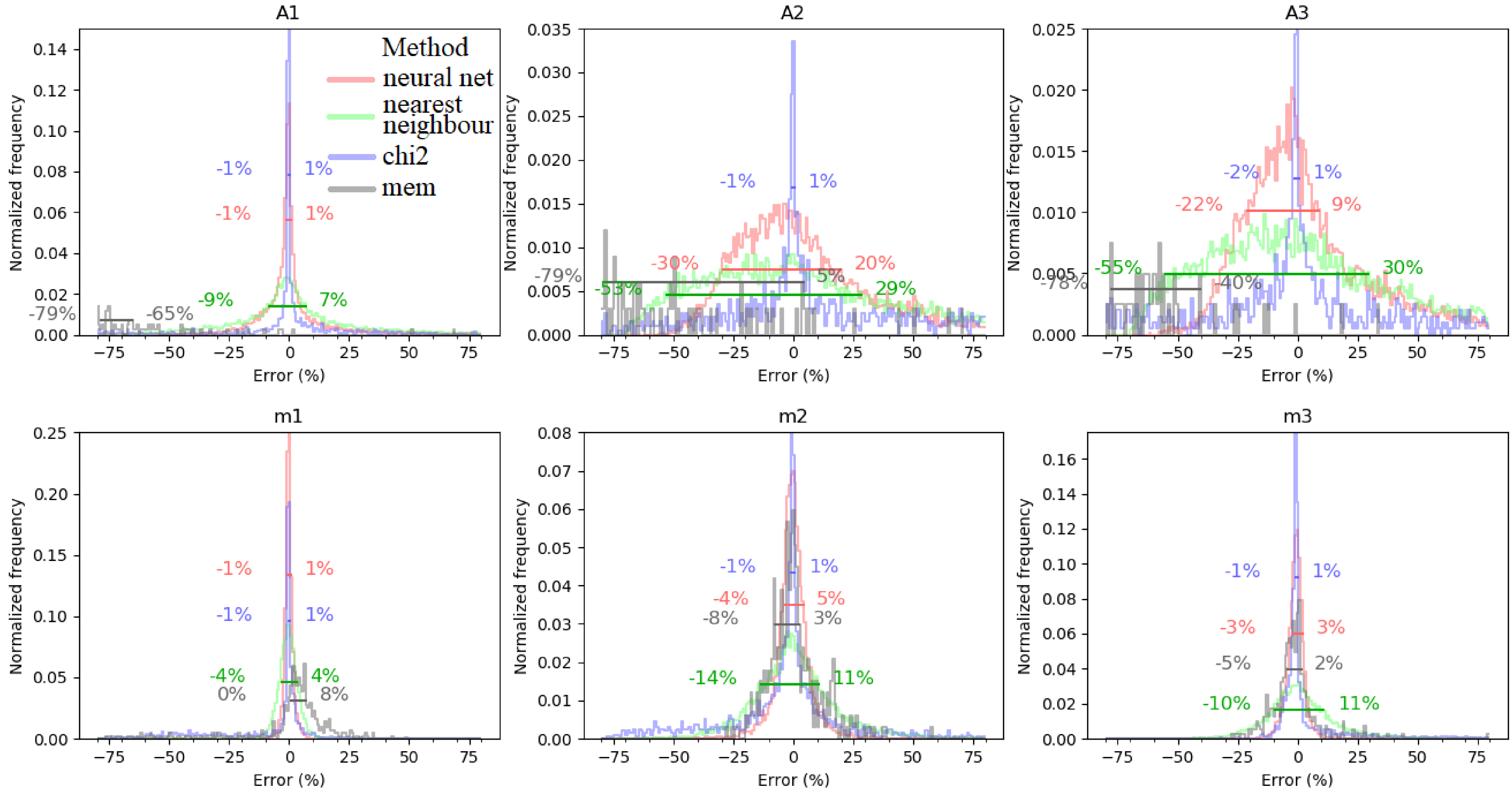




Deep Neural Networks

Selected subset of 4 propagators with different noise

	A1	A1 pred		A2	A2 pred		A3	A3 pred		a1	a1 pred		a2	a2 pred		a3	a3 pred		Average
	1	0,9895	1,1%	1	1,0138	1,4%	1	1,013	1,3%	6	6,8008	13,3%	13,2	15,4341	16,9%	18	18,899	5,0%	
noise	1	0,7999	20,0%	0,3	0,311	3,7%	0,3	0,34	13,3%	6	5,9381	1,0%	13,2	11,0478	16,3%	18	16,8765	6,2%	8,0%
= 0	1	0,8745	12,6%	1	0,796	20,4%	1	0,9315	6,9%	6	6,2046	3,4%	7,2	6,8453	4,9%	18	17,9648	0,2%	
	1	0,986	1,4%	1	1,0406	4,1%	1	1,0452	4,5%	6	6,8882	14,8%	16,8	18,0386	7,4%	18	20,2149	12,3%	
	1	0,9895	1,1%	1	1,0252	2,5%	1	1,0226	2,3%	6	6,8653	14,4%	13,2	15,3065	16,0%	18	19,1216	6,2%	
noise	1	0,8101	19,0%	0,3	0,3108	3,6%	0,3	0,3368	12,3%	6	5,9673	0,5%	13,2	10,8972	17,4%	18	17,0525	5,3%	8,1%
= 1e-3	1	0,8742	12,6%	1	0,7945	20,6%	1	0,927	7,3%	6	6,2042	3,4%	7,2	6,8348	5,1%	18	17,9997	0,0%	
	1	0,986	1,4%	1	1,0472	4,7%	1	1,052	5,2%	6	6,8822	14,7%	16,8	18,1159	7,8%	18	20,1366	11,9%	
	1	0,9977	0,2%	1	0,9875	1,3%	1	0,9821	1,8%	6	6,0197	0,3%	13,2	14,668	11,1%	18	17,9642	0,2%	
noise	1	0,8526	14,7%	0,3	0,3265	8,8%	0,3	0,3333	11,1%	6	5,9245	1,3%	13,2	11,7558	10,9%	18	18,1214	0,7%	6,4%
= 1e-2	1	0,8655	13,5%	1	0,7829	21,7%	1	0,9206	7,9%	6	6,2114	3,5%	7,2	6,7798	5,8%	18	17,7411	1,4%	
	1	0,9933	0,7%	1	1,0728	7,3%	1	1,0735	7,3%	6	6,3868	6,4%	16,8	17,9681	7,0%	18	19,7295	9,6%	
	1	0,9356	6,4%	1	0,7416	25,8%	1	0,6763	32,4%	6	6,2627	4,4%	13,2	13,9988	6,1%	18	19,7976	10,0%	
noise	1	0,6721	32,8%	0,3	0,494	64,7%	0,3	0,3602	20,1%	6	5,869	2,2%	13,2	7,5179	43,0%	18	20,5297	14,1%	13,1%
= 1e-1	1	0,9605	4,0%	1	1,0384	3,8%	1	0,865	13,5%	6	6,0351	0,6%	7,2	7,0552	2,0%	18	17,2971	3,9%	
	1	0,9529	4,7%	1	0,9339	6,6%	1	0,9298	7,0%	6	6,049	0,8%	16,8	17,3584	3,3%	18	18,5556	3,1%	
Average			9,1%			12,6%			9,6%			5,3%			11,3%			5,6%	



A1
A2
A3
m1
m2
m3
Method
neural net
nearest neighbour
chi2
mem
Normalized frequency
Error (%)
-1%
1%
-1%
1%
-9%
7%
-79%
-65%
-30%
20%
-79%
5%
-53%
29%
-2%
1%
-22%
9%
-55%
30%
-78%
-40%
-4%
4%
0%
8%
-4%
5%
-8%
3%
-14%
11%
-3%
3%
-5%
2%
-10%
11%

Deep Neural Networks

Important problem of Neural Networks:

- Hard to characterize / estimate prediction errors

Possible solution:

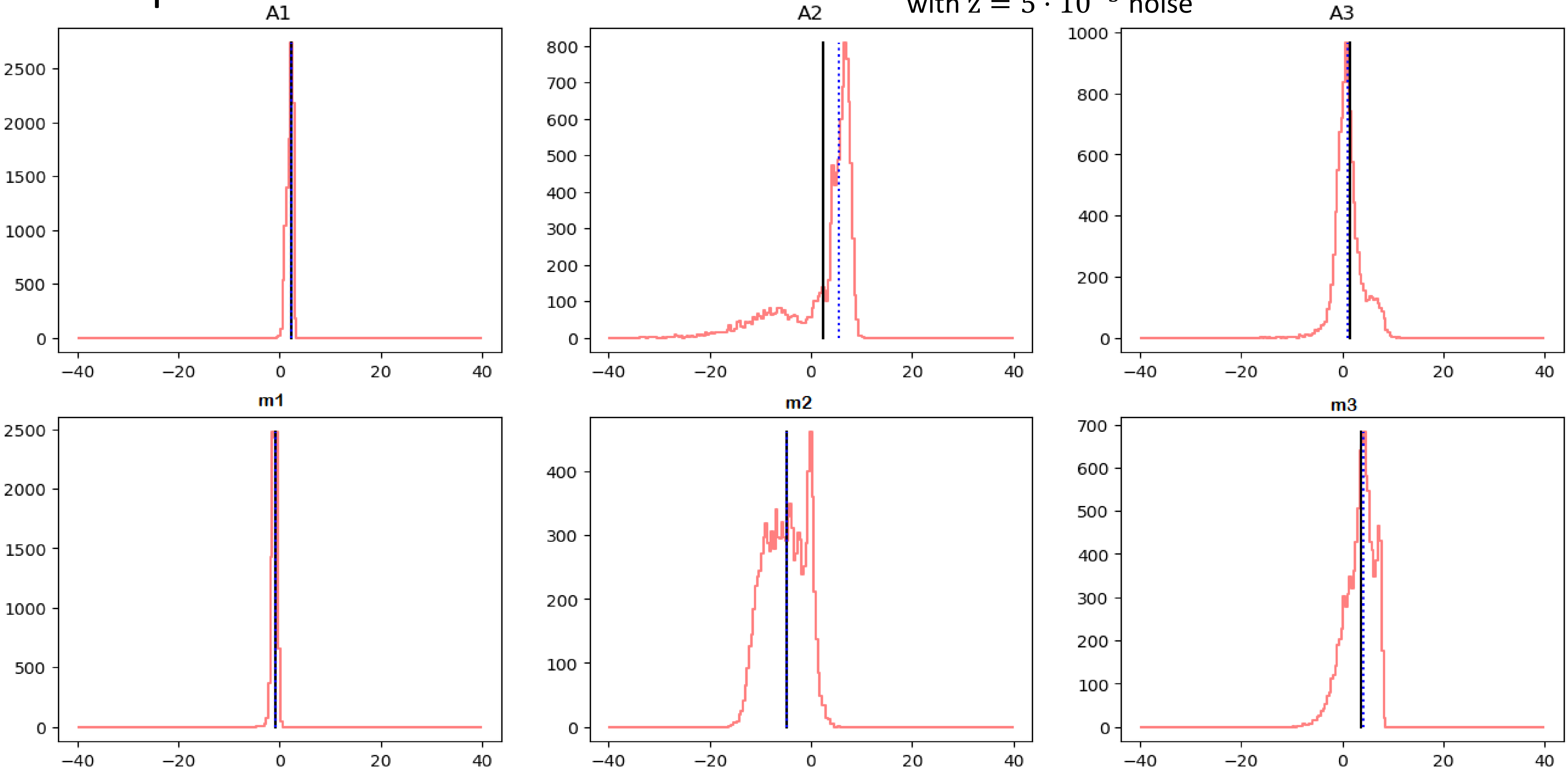
- Monte-Carlo estimation

Question:

- Does small changes in input translate to small changes in the predicted parameters?

Deep Neural Networks

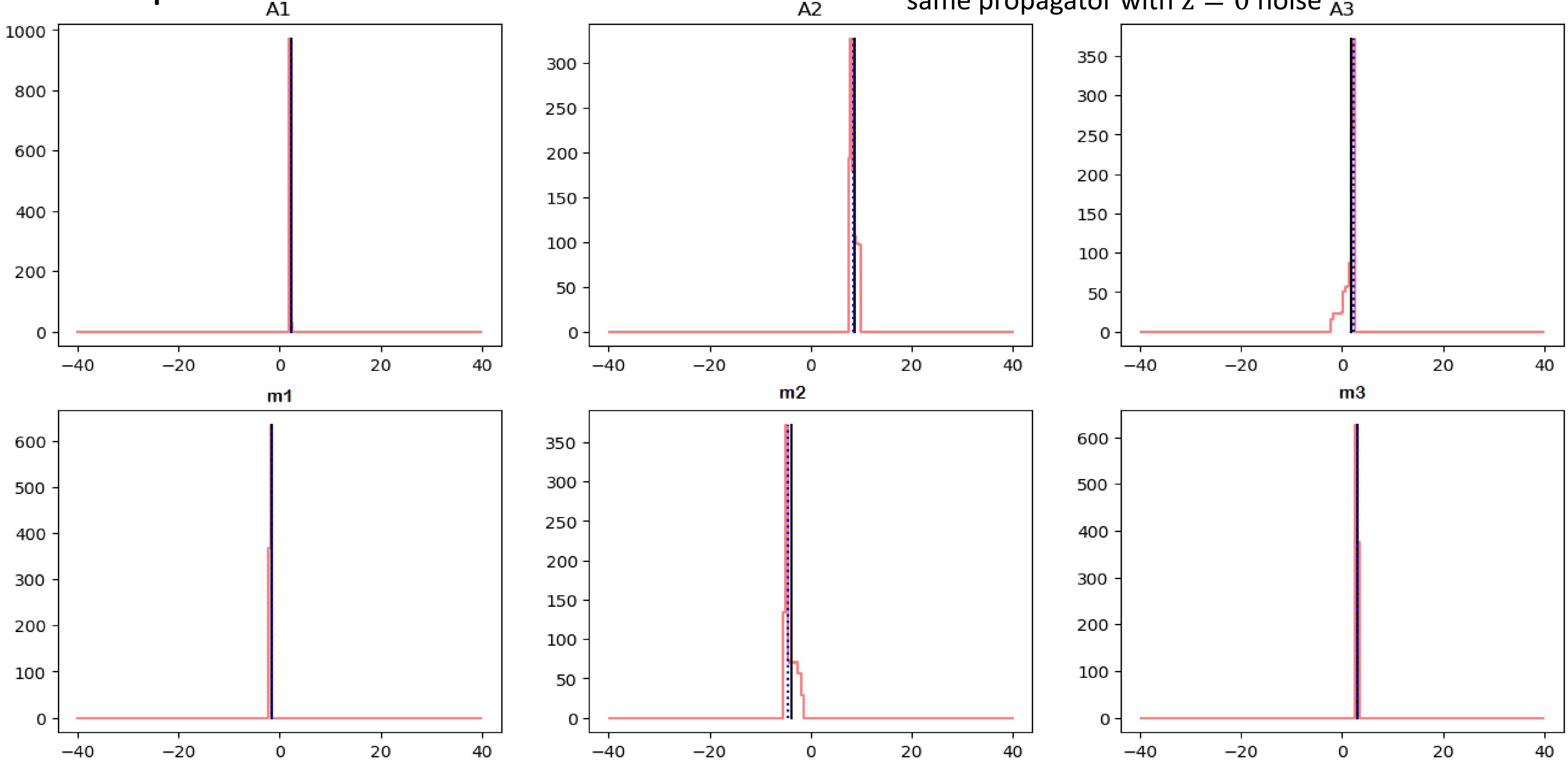
10k realization of the same propagator
with $z = 5 \cdot 10^{-3}$ noise



Relative error of parameters (%)

Deep Neural Networks

1k +/-1% perturbation of m1 in the same propagator with $z = 0$ noise



Relative error of parameters (%)

Deep Neural Networks

Observations:

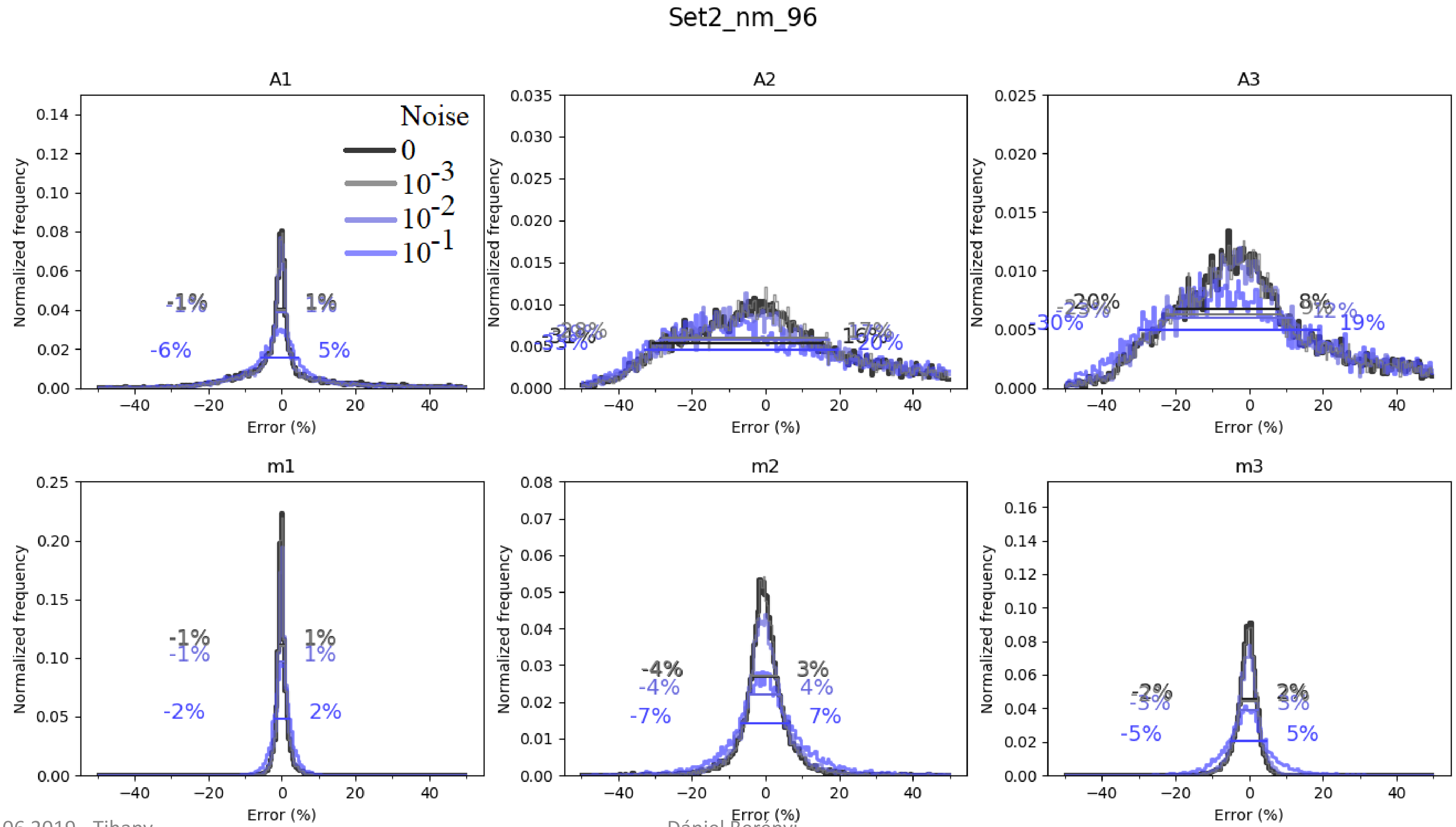
- Neural Networks predict the parameters of the spectral function in the range of 5-15% even for unrealistically large noise ($z = 10^{-1}$)
- Noise sensitivity can be mitigated by ensemble statistics
- Mispredictions seems to be systematic

Further observations

- The method scales moderately with lattice size:

Lattice size	Average error
8	10.493
16	8.735
32	8.558
64	8.374
128	8.063

Deep Neural Networks – noise tolerance



Deep Neural Networks - exrapolation

We are also interested in resonances…

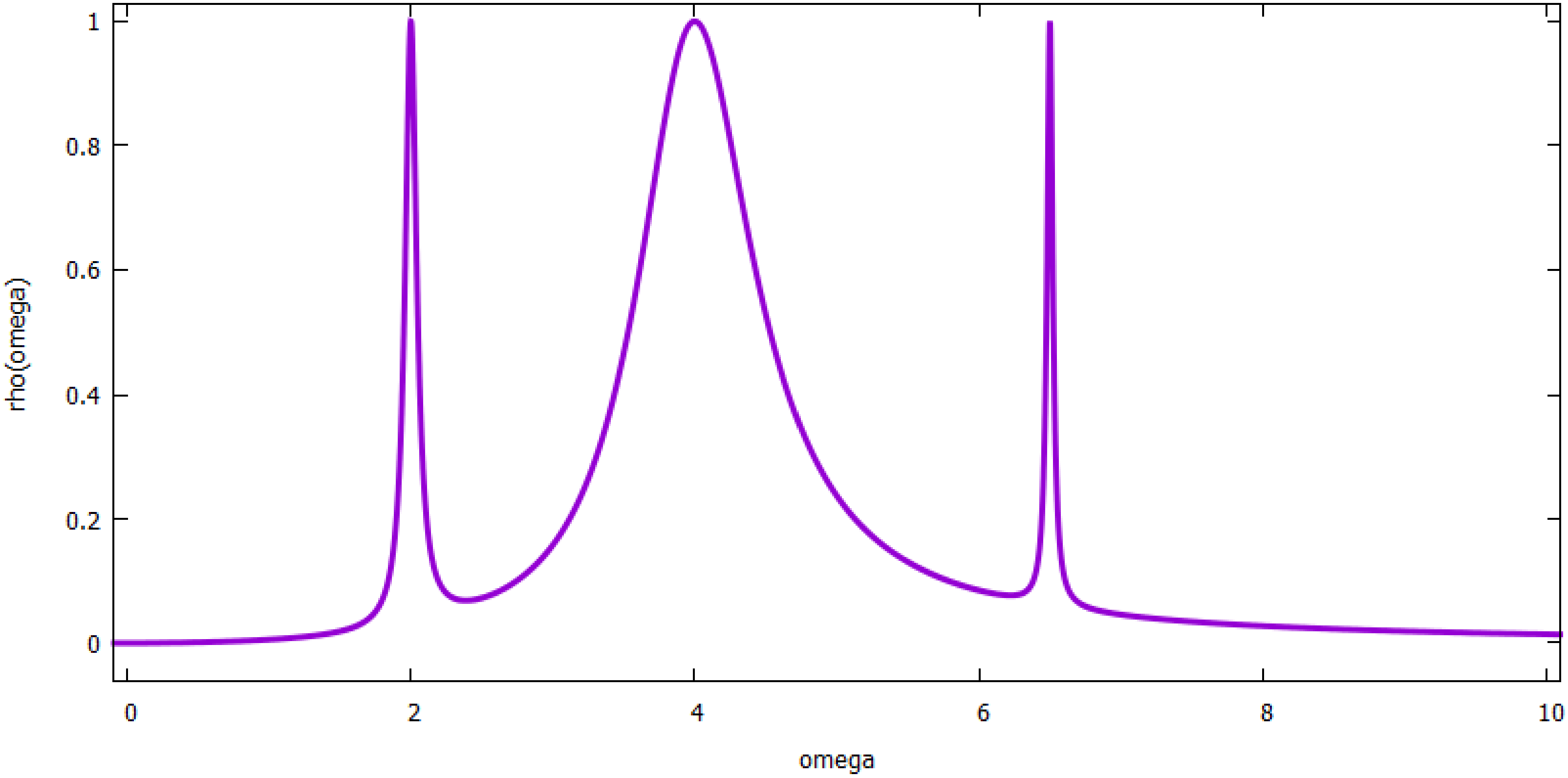
So instead of:

$$\rho(\omega) = \sum_k c_k \delta(\omega - m_k)$$

We might consider:

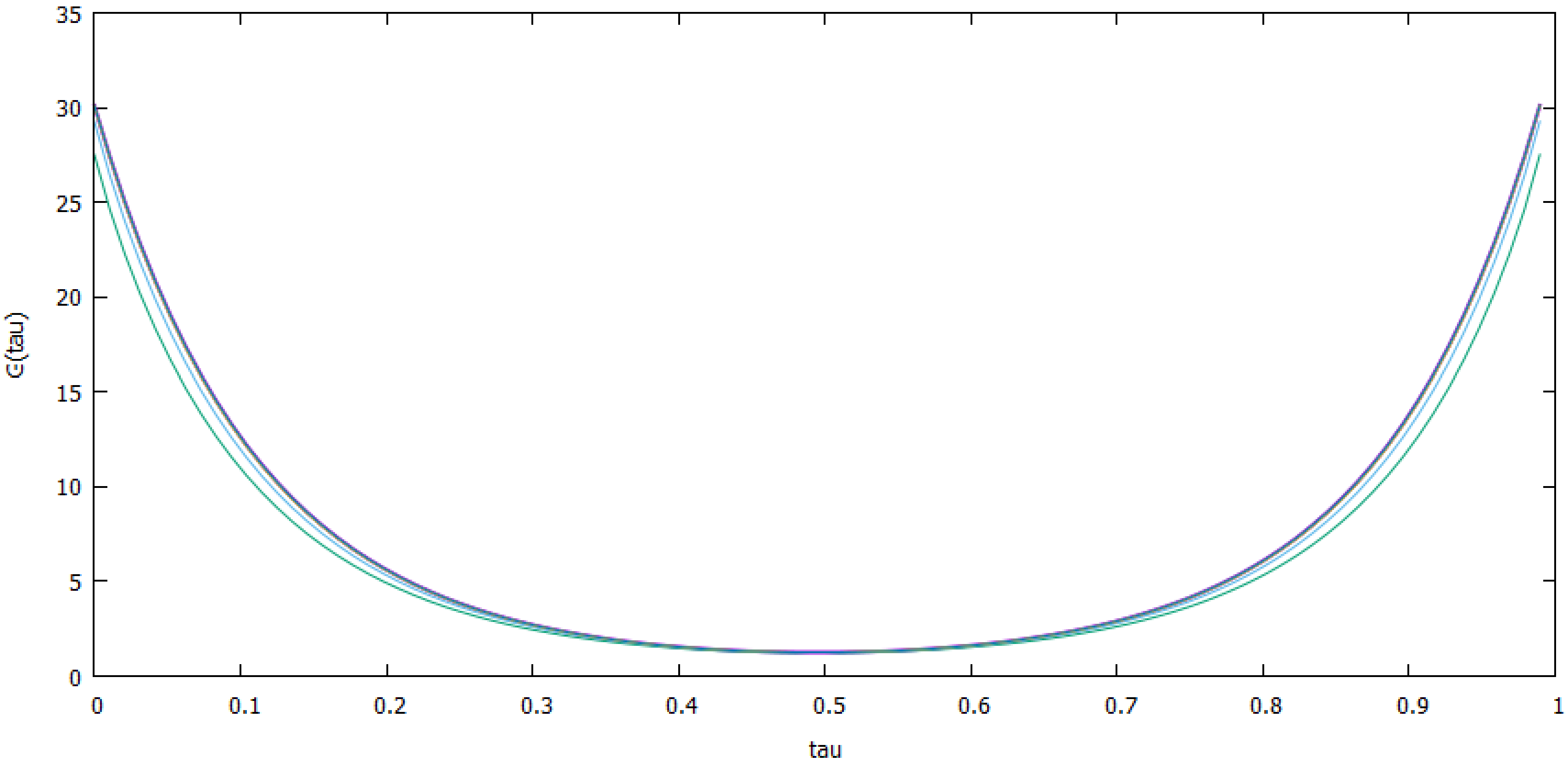
$$\rho(\omega) = \frac{\omega^2}{\left(\sum_k \left(\frac{w_k}{\omega^2 - m_k^2}\right)^2\right)^{-1} + \omega^2}$$

Deep Neural Networks - extrapolation

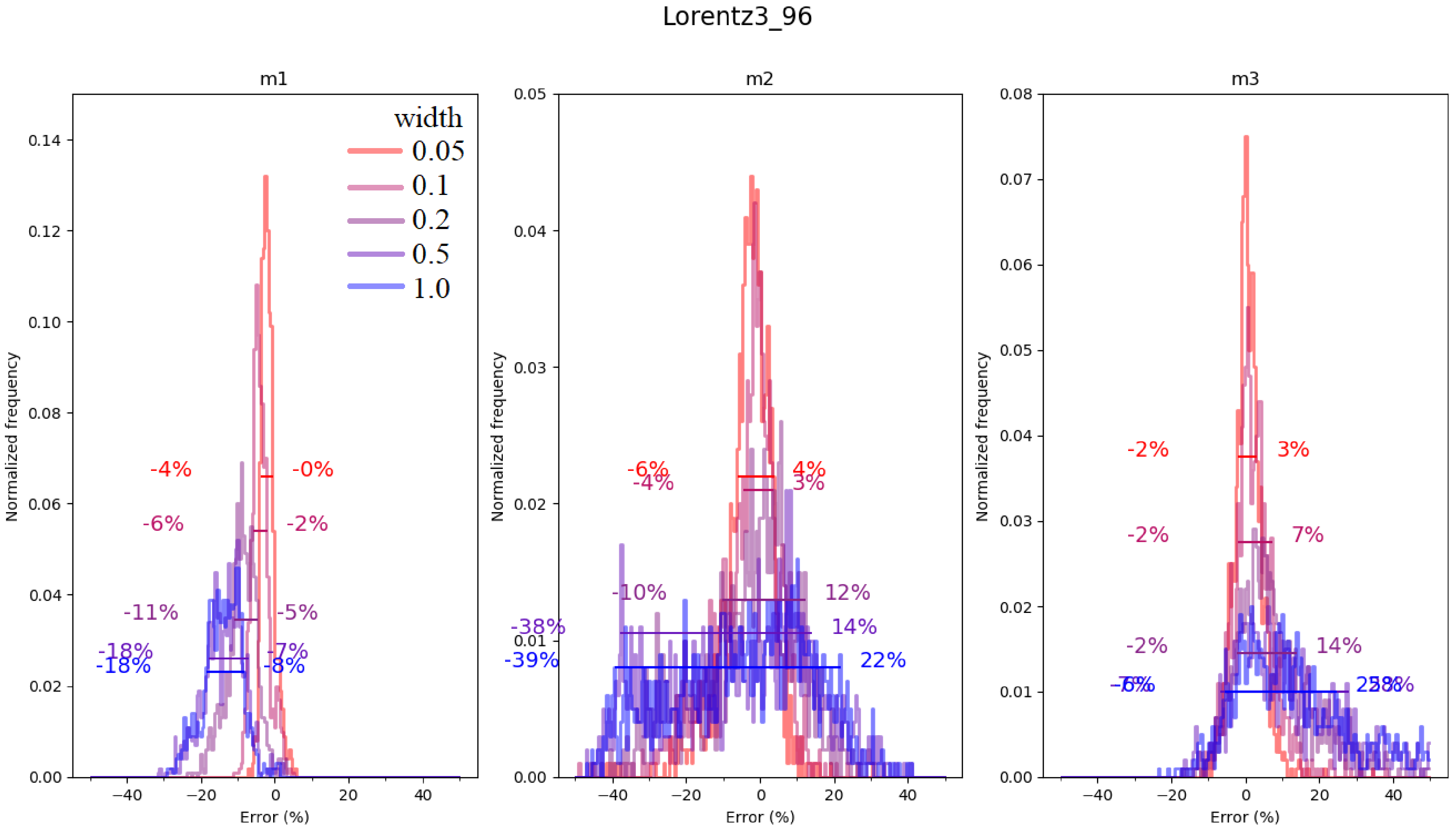


Deep Neural Networks - extrapolation

As the width goes to zero, we recover the distinct peak structure, also, the propagators are similar in this limit:



Deep Neural Networks - extrapolation



Conclusion

- Neural Networks can be considered as a tool for approximating unknown functions, e.g. inverting integral transformations by observing input-output pairs
- While learning directly from data, *a priori* information is still present, it is now encoded in the training set and the network architecture (topology)
- However, if the training set can be set up to reasonably span the expected set of values of interest, the method seems to be reliable and outperform other methods especially for noisy data

This research was supported by: NKFIH grants No. K120660 and K123815