Recurrent machines for likelihood-free inference

Arthur Pesah[*]
KTH Royal Institute of Technology
Stockholm, Sweden

Antoine Wehenkel[*]
University of Liège
Liège, Belgium

Gilles Louppe
University of Liège
Liège, Belgium

Abstract

Likelihood-free inference is concerned with the estimation of the parameters of a non-differentiable stochastic simulator that best reproduce real observations. In the absence of a likelihood function, most of the existing inference methods optimize the simulator parameters through a handcrafted iterative procedure that tries to make the simulated data more similar to the observations. In this work, we explore whether meta-learning can be used in the likelihood-free context, for learning automatically from data an iterative optimization procedure that would solve likelihood-free inference problems. We design a recurrent inference machine that learns a sequence of parameter updates leading to good parameter estimates, without ever specifying some explicit notion of divergence between the simulated data and the real data distributions. We demonstrate our approach on toy simulators, showing promising results both in terms of performance and robustness.

1 Introduction

Modern science often relies on the modeling of complex data generation process by means of computer simulators. While the forward generation of observables is often straightforward and well-motivated, inverting generation processes is usually very difficult. In particular, scientific simulators are often stochastic and give rise to intractable likelihood functions that prevent the use of classical inference algorithms. The importance and prevalence of this problem has recently motivated the development of so-called likelihood-free inference methods (LFI) which do not make use of the likelihood function for estimating model parameters. LFI methods (e.g., Beaumont et al., 2002; Gutmann and Corander, 2016; Louppe and Cranmer, 2017) are often based on handcrafted iterative optimization procedures, where a sequence of updates are performed to make the simulated data more similar to the observations.

Driven by the promises of learning to learn, meta-learning has shown that automatically learning neural optimizers from data is possible, achieving results close to the state-of-the-art for the task of training neural networks (Andrychowicz et al., 2016) or solving inverse problems (Putzky and Welling, 2017) when the gradient of the objective function is available. Meanwhile, (Chen et al., 2016) have shown that meta-learning is also capable of learning neural optimizers that rely at each step on the value of the objective function only, without requiring access to its gradient.

In this work, we push the limits of the meta-learning framework further by showing that it can be used even when no explicit objective value is available at each step. More specifically, we focus on likelihood-free inference problems and build a recurrent inference machine that learns an iterative procedure for updating simulator parameters such that they converge (in distance) towards nominal parameter values known at training. In particular, the inference machine is never given access to an explicit objective function that would estimate some divergence between the synthetic distribution and the real data distribution. Rather, both the optimization procedure and the implicit objective to minimize are learned end-to-end from artificial problems.

[*]Both authors contributed equally to this work.

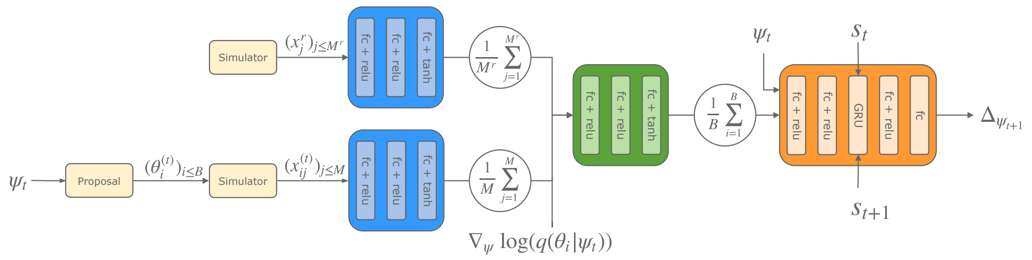

Figure 1: Recurrent machine for likelihood-free inference. All three encoders are fully-connected (fc) neural networks. The RNN is a GRU (Cho et al., 2014) with fully-connected pre-processing and post-processing layers.

2 Problem statement

The goal of likelihood-free inference is the estimation of the parameters of a stochastic generative process with an unknown or intractable likelihood function. Formally, given a set of observations $X^r = \{\boldsymbol{x}_1^r, ..., \boldsymbol{x}_{M^r}^r\}$ drawn i.i.d. from the real data distribution $p_r(\boldsymbol{x})$, we are interested in finding the model parameters

$$\boldsymbol{\theta}^* = \arg\min_{\theta} \rho(p_r(\boldsymbol{x}), p(\boldsymbol{x}|\boldsymbol{\theta}))$$

that minimize some discrepancy ρ between the real data distribution $p_r(\boldsymbol{x})$ and the implicit distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ of the simulated data.

In this work, instead of defining one objective point $\boldsymbol{\theta}^*$, we target a probability distribution over the $\boldsymbol{\theta}$ space. Formally, our goal is to find the optimal parameter $\boldsymbol{\psi}^*$ of a parametric proposal distribution $q(\boldsymbol{\theta}|\boldsymbol{\psi})$ on $\boldsymbol{\theta}$, leading to the following optimization problem

$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \rho(p_r(\boldsymbol{x}), p(\boldsymbol{x}|\boldsymbol{\psi})) \quad \text{where} \quad p(\boldsymbol{x}|\boldsymbol{\psi}) = \int q(\boldsymbol{\theta}|\boldsymbol{\psi})p(\boldsymbol{x}|\boldsymbol{\theta})\boldsymbol{d\theta}.$$

The proposal distribution provides two advantages: i) it leads to a variational formulation of the optimization problem that does not rely on the usage of the gradient of the function to be optimized (Staines and Barber, 2012), ii) by sampling B points from this distribution, $\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_B \sim q(\boldsymbol{\theta}|\boldsymbol{\psi})$, we get distinct parameters that we can give to the simulator. Then, by comparing the generated observations $X_i = (\boldsymbol{x_{i,1}}, ..., \boldsymbol{x_{i,M}})$ to the real ones X^r for each $\boldsymbol{\theta}_i$, it is possible to figure out what the optimal update of $\boldsymbol{\psi}$ is.

Because the likelihood function $p(\boldsymbol{x}|\boldsymbol{\theta})$ cannot be evaluated, strategies must be found to use only simulated data in order to estimate $\boldsymbol{\psi}^*$. Current methods for likelihood-free inference typically rely on a handcrafted iterative update procedure where at each time step $t \in [1, T]$ the next estimate $\boldsymbol{\psi}_{t+1}$ is determined by comparing the simulated data produced from $p(\boldsymbol{x}|\boldsymbol{\psi})$ at the current parameter estimate $\boldsymbol{\psi}_t$ with the real observations X^r. In this work, our goal is to investigate whether meta-learning can be used for learning a parametrized update function $f_{\boldsymbol{\phi}}$ such that

$$\boldsymbol{\psi}_{t+1} = \boldsymbol{\psi}_t + f_{\boldsymbol{\phi}}((\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_t), X^r), \quad \text{with} \quad \boldsymbol{\psi}_t \to \boldsymbol{\psi}^* \quad \text{as} \quad t \to \infty.$$

3 Recurrent machines for likelihood-free inference

In this work, we define $f_{\boldsymbol{\phi}}$ as a recurrent neural network (RNN) whose architecture is given in Figure 1. At each time step t, the RNN takes as input the current proposal parameters $\boldsymbol{\psi}_t$, a memory state s_t, some information about the real observations and those generated with $\boldsymbol{\psi}_t$, and produces as output the parameter update $\Delta_{\boldsymbol{\psi}_{t+1}}$ and s_{t+1}. Observations $(x_j)_{j\le M}$ are ingested through data encoder architectured as a feedforward neural network that takes as input each x_j independently, transforms each of them into a d-dimensional vector, and aggregates them through an averaging of those vectors over j. It should therefore be able to compute any moment (and more general feature averages) of the distribution. If the moments are learned, they can be used to infer the parameters of the distribution by the method of matching moments, whose principle is to check compatibility between the generated data and the observed data by looking at these moments (Ravuri et al., 2018). This also means that moments can capture the relevant information of a set of samples about a parameter of interest.

In our architecture, we use three encoders. The first one is for the real observations $(\boldsymbol{x}_j^r)_{j \leq M^r}$. The second one is for the generated observations at time t. We consider a sampling of B parameters $(\boldsymbol{\theta}_i^{(t)})_{i \leq B}$, and for each $\boldsymbol{\theta}_i^{(t)}$, the corresponding observations $(\boldsymbol{x}_{i,j}^{(t)})_{j \leq M}$. These two encoders share their weights, because the way true and generated data are summarized should be the same in order to be able to compare them. The last encoder takes the results of the first two as well as the log-likelihood value $\nabla_\psi \log q(\boldsymbol{\theta}_i^{(t)}|\boldsymbol{\psi}_t)$, as motivated by Wierstra et al. (2014) and which represents the direction to follow to move the proposal distributions toward $\boldsymbol{\theta}_i^{(t)}$.

The loss used at training should encourage the network to generate updates $\Delta_{\boldsymbol{\psi}_t}$ of the proposal which yield to a final proposal $q(\boldsymbol{\theta}|\boldsymbol{\psi}_T)$ that is close to a delta-function at $\boldsymbol{\theta}^*$. The total loss of the RNN is expressed in a way similar to (Andrychowicz et al., 2016), i.e. as a weighted sum of a local loss evaluated for each $\boldsymbol{\psi}_t$,

$$\mathscr{L}(\boldsymbol{\psi}_T, \boldsymbol{\theta}^*) = \sum_{t=1}^{T} w_t \ell(\boldsymbol{\psi}_t, \boldsymbol{\theta}^*),$$

where ℓ is a loss comparing the proposal parameters $\boldsymbol{\psi}_t$ with the real parameter $\boldsymbol{\theta}^*$, w_t is a weight given to the loss at time t (for instance $w_t = 1$ for all t, or $w_t = \mathbb{1}_{t=T}$). The choice of the weighting and the loss functions are discussed in the experiments section.

We call the architecture ALFI (Automatic Likelihood-Free Inference).

4 Experiments

To illustrate our method and compare it with a simple baseline, we performed experiments on three toy simulators, whose likelihood is known and consequently for which the maximum likelihood estimator (MLE) can be computed. The results for the simplest simulator is shown below and results for the two other toy problems are provided in Appendix C. We also tested our architecture on a simulator from particle physics, whose exact likelihood function is intractable. For this last experiment we assess the model performance by comparing the data generated at the end of the iterative process with the real observations. The full description of each simulator is given in Appendix B.

4.1 Illustrative example

As an illustrative example, we implemented a simulator that generates samples from a Poisson distribution $\mathscr{P}(\lambda = e^\theta)$. The goal is to estimate the parameter θ corresponding to real samples of this distribution.

Results The box plot of Figure 2 compares the performance of our model with the MLE. For our model (ALFI), the root mean-squared error (RMSE) is computed between the mean of the final proposal parametrized by ψ_T and the true value of the parameters: RMSE $= ||\theta^* - \mathbb{E}_{\theta \sim q(\theta|\psi_T)}[\theta]||_2$. We observe that our model achieves similar performance as the MLE whereas it is not given any explicit function to minimize during testing.

The left part of Figure 2 shows the evolution of the average and the standard deviation of the RMSE along the iterative procedure over all test problems. It can be observed from this plot that our model quickly converges to a proposal distribution with an expected value close to θ^*.

4.2 (Simplified) particle physics simulation

We also tested our model on a simplified simulator from particle physics, called Weinberg and introduced as a likelihood-free inference benchmark in (Louppe and Cranmer, 2017). This benchmark comes with two parameters (the beam energy and the Fermi constant) and one observable (the cosine of the scattering angle).

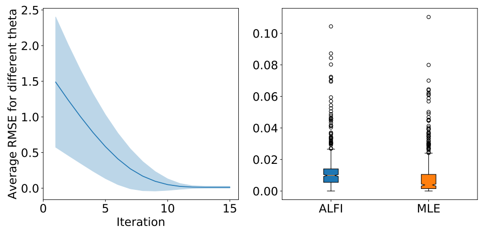

Figure 2: Results for the Poisson simulator. **(Left)** We observe that the RMSE decreases quickly during the 15 first iterations. **(Right)** ALFI and MLE results are very similar.

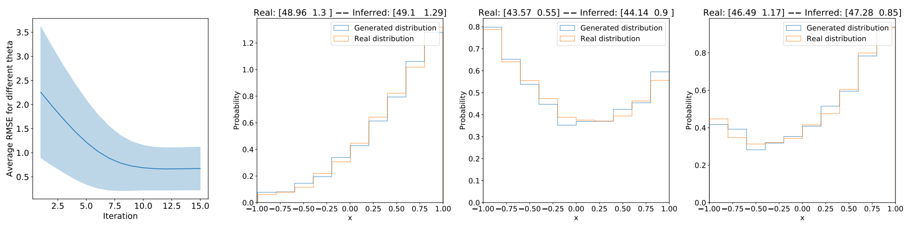

Figure 3: **(Left)** Evolution of the error between the true and predicted parameters for the Weinberg simulator, averaged over several θ^*. **(Right)** Histogram of the real VS. generated data with the inferred parameters, for 3 different $\boldsymbol{\theta}^*$. The vectors on top of each figure represent the real parameters $[\theta_0^*, \theta_1^*]$ (left) and the inferred ones $[\hat{\theta}_0, \hat{\theta}_1]$

Results Figure 3 (left) presents the evolution of the error between the real and the predicted parameters along 15 iterations. We see that it has learned an iterative procedure and converges after 10 iterations. Figure 3 (right) compares the distributions of the simulated data $p(x|\boldsymbol{\psi}_T)$ and the real data $p_r(x)$. We can see that ALFI has managed to infer parameters that simulate a realistic distribution.

4.3 Robustness of the model

In order to evaluate the robustness of the learned optimization procedure, we tested the model on a number of iterations T_{test} greater than the number T_{train} used during training. On the Poisson and the multivariate simulators (Appendix B.3), we observed that increasing the number of iterations improves the performance of ALFI. Therefore, our model seems to learn an update rule $\boldsymbol{\Delta}_\psi$ that is not tied to the specific number of updates used a training, but rather generalizes to a larger horizon. Those results are shown in Appendix C.

5 Conclusions and future works

In this work, we provide a proof-of-concept of a meta-learning architecture designed to solve likelihood-free inference problems. We applied our model on toy simulators and achieved results competitive with maximum likelihood estimation. We finally applied it on a simple particle physics simulator, and showed that it can infer parameters corresponding to samples close to the real ones.

As for future work, we see two paths worth of exploration. First, getting a better understanding of the optimization procedure learned by ALFI: can we interpret the representation learned by each data encoder? Could an update model learned for a simulator be transferred to another simulator? Is the learned procedure comparable to other existing methods? Secondly, evaluating our model on more complex simulators and comparing it to state-of-the-art LFI methods: since our approach is intensive in the number of simulator calls, moderating this complexity would be a necessary step to scale our method to slower simulators.

References

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989.

Beaumont, M. A., Zhang, W., and Balding, D. J. (2002). Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2016). Learning to learn without gradient descent by gradient descent. *arXiv preprint arXiv:1611.03824*.

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Gutmann, M. U. and Corander, J. (2016). Bayesian optimization for likelihood-free inference of simulator-based statistical models. *The Journal of Machine Learning Research*, 17(1):4256–4302.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Louppe, G. and Cranmer, K. (2017). Adversarial variational optimization of non-differentiable simulators. *arXiv preprint arXiv:1707.07113*.

Putzky, P. and Welling, M. (2017). Recurrent inference machines for solving inverse problems. *arXiv preprint arXiv:1706.04008*.

Ravuri, S., Mohamed, S., Rosca, M., and Vinyals, O. (2018). Learning implicit generative models with the method of learned moments. *arXiv preprint arXiv:1806.11006*.

Staines, J. and Barber, D. (2012). Variational optimization. *arXiv preprint arXiv:1212.4507*.

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. (2014). Natural evolution strategies. *Journal of Machine Learning Research*, 15:949–980.

Appendix

A Experimental setup

Optimizer To train our architecture, we used the ADAM optimizer (Kingma and Ba, 2014).

Proposal distribution For all the experiments, we defined $q(\boldsymbol{\theta}|\boldsymbol{\psi})$ as a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$ with parameters $\boldsymbol{\psi} \in \mathbb{R}^d$ where d denotes the size of the parameter space and $\boldsymbol{\psi}_i = [\boldsymbol{\mu}, \boldsymbol{\sigma}] \quad \text{where} \quad \boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^d$. For each experiment the initial proposal $\boldsymbol{\psi}_1$ is made of a random mean vector $\boldsymbol{\mu}_1$ drawn with the same distribution as the parameters $\boldsymbol{\theta}^*$ of the different problems of the training set. The variance vector $\boldsymbol{\sigma}_1$ always starts at $\boldsymbol{\sigma}_1 = [e^{0.5}, ..., e^{0.5}]$.

Partial loss function For the partial loss $\ell(\boldsymbol{\psi}_t, \boldsymbol{\theta}^*)$ at time t, we tried two distinct functions: the mean-squared error between the mean of $\boldsymbol{\psi}_t$ and $\boldsymbol{\theta}^*$, $\ell(\boldsymbol{\psi}_t, \boldsymbol{\theta}^*) = ||\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{\theta}|\boldsymbol{\psi})}[\boldsymbol{z}] - \boldsymbol{\theta}^*||_2^2$ and the negative-log-likelihood of the proposal evaluated on $\boldsymbol{\theta}^*$, $\ell(\boldsymbol{\psi}_t, \boldsymbol{\theta}^*) = -\log(q(\boldsymbol{\theta}^*|\boldsymbol{\psi}))$. The later loss has the advantage of taking more the variance of the proposal into account and lead to better performance experimentally. Therefore, we decided to use the likelihood loss function for all the subsequent experiments.

Weighting function The choice of the weighting function w_t determines the exploration-exploitation trade-off of our iterative algorithm. We tested three weighting schemes:

- $w_t = \mathbb{1}_{t=T}$: only $\boldsymbol{\psi}_T$, the final proposal distribution on the parameters, is taken into account in the total loss function. It means that the algorithm can freely explore the parameter-space during $T-1$ iterations.
- $w_t = 1$ for all t: all the parameters' estimates found during the iterative process are taken into account with the same weight. It encourages the algorithm to converge as fast as possible to a good parameter.
- $w_t = \frac{e^{\beta x}-1}{e^{\beta}-1}$: compromise between the two previous weightings. The first steps are given a low weight, encouraging exploration, while the last steps have a high weight to ensure convergence by the end.

Among those three weighting schemes, the exponential one gave the best performance and we decided to use it for all the subsequent experiments.

Marginalization To avoid overfitting, the initial value of the mean of the proposal parameters is taken randomly. Thus to compute the performance of our model at test time, $\boldsymbol{\psi}_1$ is marginalized out. To do so, we draw 500 $\boldsymbol{\psi}_1$ values and take the average outputs $\boldsymbol{\psi}_T$.

Hyperparameters We provide below a list of all the hyperparameters of ALFI, along with a description if necessary:

- Number of epochs
- Number of iterations T
- Number of $\boldsymbol{\theta}^*$ for meta-training: size of the meta-dataset
- Distribution of the $\boldsymbol{\theta}^*$: how the $\boldsymbol{\theta}^*$ used in the meta-training are generated
- Meta batch-size: number of $\boldsymbol{\theta}^*$ that we use to compute the gradient that we backpropagate in our networks.
- Batch size for $\boldsymbol{\theta}$: number of $\boldsymbol{\theta}$ that we generate from $q(\boldsymbol{\theta}|\boldsymbol{\psi}_t)$ at each time t
- Batch size for $\boldsymbol{x}$: number of $\boldsymbol{x}$ that we generate from each $\boldsymbol{\theta}$ generated from $\boldsymbol{\psi}_t$ at time t
- Learning rate
- Clipping: we force our model to follow an iterative procedure by clipping each component of the output $\boldsymbol{\Delta}_{\psi}$ of the RNN between two values.

B Simulators

B.1 Linear Regression

Forward generation The data generated by this procedure follow a linear law in 2 dimensions, the unknown parameters of the simulator represents the slope and the offset of this line. Formally, let $X = [x, 1, y]$ denote an observation generated by θ, where $x, y \in \mathbb{R}$. Then X satisfies the constraint $y - n = \tan(\theta_0)x + \theta_1$ with $n \sim \mathscr{N}(0, 0.1)$ and the value of x being drawn uniformly between -1 and 1.

Hyperparameters

- Number of epochs: 300
- Number of iterations T: 15
- Number of $\boldsymbol{\theta}^*$ for meta-training: 10000
- Distribution of the $\boldsymbol{\theta}^*$: uniformly in $[0, \frac{\pi}{2}] \times [-1, 1]$
- Meta batch-size: 16
- Batch size for $\boldsymbol{\theta}$: 20
- Batch size for $\boldsymbol{x}$: 20
- Learning rate: $1e - 3$
- Clipping: $[-0.25, 0.25]$

B.2 Poisson distribution

Forward generation The forward generation process is a simple Poisson distribution which depends on the mean parameter $\lambda \in \mathbb{R}^+$ of the distribution. To make the parametrisation real we define $\boldsymbol{\theta} = \theta = \log(\lambda) \in \mathbb{R}$. The generation of an observation x conditionally to the parameter value θ is done by sampling x from $\mathscr{P}(\lambda = e^\theta)$.

Hyperparameters

- Number of epochs: 300
- Number of iterations T: 15
- Number of θ^* for meta-training: 10000
- Distribution of the θ^*: uniformly in $[0.2, 7.0]$
- Meta batch-size: 16
- Batch size for θ: 20
- Batch size for x: 20
- Learning rate: $1e - 3$
- Clipping: $[-0.5, 0.5]$

B.3 Multivariate Distribution

Forward generation The forward generation process for $\boldsymbol{\theta} = [\theta^{(0)}, \theta^{(1)}, \theta^{(2)}]$ can be described as follow:

1. Draw independently $z^{(0)} \sim \mathscr{N}(\theta^{(0)}, 1)$, $z^{(1)} \sim \mathscr{N}(3, e^{\frac{\theta^{(1)}}{3}})$, $z^{(2)} \sim$ GMM $\left(\frac{1}{2}\mathscr{N}(-2, 0.5), \frac{1}{2}\mathscr{N}(2, 1)\right)$, $z^{(3)} \sim \mathscr{U}(-5, \theta^{(2)})$, $z^{(4)} \sim \text{Exp}(0.5)$
2. Compute $\boldsymbol{x} = R\boldsymbol{z}$ with $\boldsymbol{z} = \left[z^{(0)}, z^{(1)}, z^{(2)}, z^{(3)}, z^{(4)}\right]^T$ where $R \in \mathbb{R}^{5\times5}$ is a positive semi-definite matrix.

Hyperparameters

- Number of epochs: 300
- Number of iterations T: 15
- Number of $\boldsymbol{\theta}^*$ for meta-training: 10000
- Distribution of the $\boldsymbol{\theta}^*$: uniformly in $[-3, 3]^3$
- Meta batch-size: 16
- Batch size for $\boldsymbol{\theta}$: 20
- Batch size for $\boldsymbol{x}$: 20
- Learning rate: $1e - 3$
- Clipping: $[-0.2, 0.2]$

B.4 Weinberg Simulator

Introduced in (Louppe and Cranmer, 2017), Weinberg is a simplified simulator from particle physics of electron-positron collisions resulting in muon-antimuon pairs ($e^+e^- \to \mu^+\mu^-$).

Forward generation The simulator takes two parameters, the Fermi constant G^f and the beam energy E^{beam}, and produces the one-dimensional observable $x = \cos(A)$, where A is the angle of the outgoing muon with respect to the originally incoming electron.

Hyperparameters To generate our training dataset we draw 10^3 parameters $[\theta_0, \theta_1]$ uniformly in the square $[40, 50] \times [0.5, 1.5]$. We enforce our model to follow an iterative procedure by clipping the step Δ_ψ output between -0.2 and 0.2.

- Number of epochs: 130
- Number of iterations T: 15
- Number of $\boldsymbol{\theta}^*$ for meta-training: 1000
- Distribution of the $\boldsymbol{\theta}^*$: uniformly in the $[40, 50] \times [0.5, 1.5]$
- Meta batch-size: 16
- Batch size for $\boldsymbol{\theta}$: 8
- Batch size for x: 64
- Learning rate: $2e - 4$
- Clipping: $[-0.2, 0.2]$

C Supplementary results

C.1 Poisson simulator

To check that the procedure learned by our model is robust and really meaningful, we took a number of steps at test time $T_{\text{test}} = 30$ greater than $T_{\text{train}} = 15$. We see on Figure 4 that the performance are slightly better than for $T_{\text{test}} = 15$, which shows that our model is robust to the number of iterations.

C.2 Multivariate Distribution

This simulator is a combination of canonical distributions, aimed at showing that our architecture has enough capacity to learn an optimization procedure valid for parameters with very different impact on the samples. We also took a number of steps at test time $T_{\text{test}} = 30$ greater than $T_{\text{train}} = 15$. It can be observed from the left part of Figure 5 that the RMSE doesn't increase after the iteration 15 which shows that the learned procedure hasn't overfitted on the number of steps. This figure also shows that the procedure converges in both mean and variance.

It shows that the architecture has enough capacity to learn an update procedure which eventually converges to a value close to the MLE, even in the case where the parameters have very different effects on the generated data.

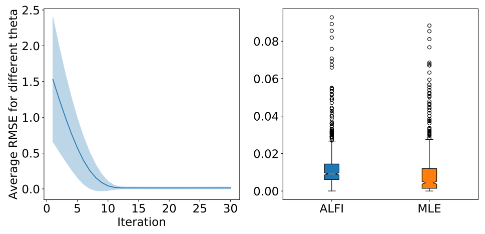

Figure 4: This plot shows that ALFI tested with a number of steps $T_{\text{test}} = 30$ gives even better result than for $T_{\text{train}} = 15$, in particular the upper whisker is smaller for ALFI than for MLE.

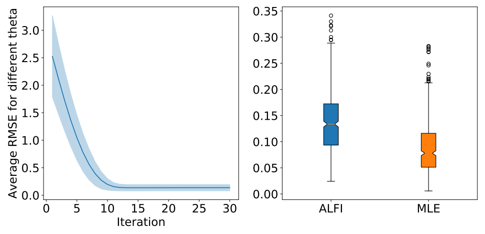

Figure 5: Results for the multivariate simulator. **(Left)** The RMSE quickly decreases during the 15 first iterations and continues to slightly decrease afterward. **(Right)** The performance of our model.

C.3 Linear Regression

Figure 6 presents the results for the linear regression simulator. We also took a number of steps at test time $T_{\text{test}} = 30$ greater than $T_{\text{train}} = 15$. It can be observed from the boxplot that, in average, ALFI has performance comparable with the MLE. However, we can notice the difficulty to estimate precisely the value of parameters for few cases.

The left sub-figure shows that the RMSE quickly converges in 15 iterations and then is stable with a slight variance reduction during the 10 last iteration.

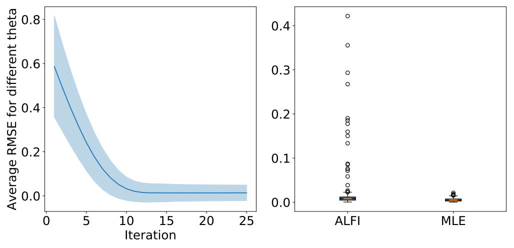

Figure 6: Results for the Linear Regression simulator. **(Left)** Fast convergence to a small average RMSE. **(Right)** ALFI gives numerous outliers but for the other points the performance of ALFI is comparable to the MLE.