Active Learning Policies for Solving Inverse Problems

Tim Bakker* T.B.BAKKER@UVA.NL
AMLab, University of Amsterdam,
Science Park 900,
1098 XH, Amsterdam, Netherlands.

Thomas Hehn THEHN@QTI.QUALCOMM.COM
Tribhuvanesh Orekondy TORENKOND@QTI.QUALCOMM.COM
Arash Behboodi BEHBOODI@QTI.QUALCOMM.COM
Fabio Valerio Massoli FMASSOLI@QTI.QUALCOMM.COM
Qualcomm AI Research,
Science Park 301,
1098 XH, Amsterdam, Netherlands.

Abstract

In recent years, solving inverse problems for black-box simulators has become a point of focus for the machine learning community due to their ubiquity in science and engineering scenarios. In such settings, the simulator describes a forward process $f : (\boldsymbol{\psi}, \boldsymbol{x}) \to \boldsymbol{y}$ from simulator parameters $\boldsymbol{\psi}$ and input data $\boldsymbol{x}$ to observations $\boldsymbol{y}$, and the goal of the inverse problem is to optimise $\boldsymbol{\psi}$ to minimise some observation loss. Simulator gradients are often unavailable or prohibitively expensive to obtain, making optimisation of these simulators particularly challenging. Moreover, in many applications, the goal is to solve a family of related inverse problems. Thus, starting optimisation ab-initio/from-scratch may be infeasible if the forward model is expensive to evaluate. In this paper, we propose a novel method for solving classes of similar inverse problems. We learn an active learning policy that guides the training of a surrogate and use the gradients of this surrogate to optimise the simulator parameters with gradient descent. After training the policy, downstream inverse problem optimisations require up to 90% fewer forward model evaluations than the baseline.

1. Introduction and related work

Across many science and engineering fields, practitioners are required to deduce unknown properties or parameters of a system from observed data (Cranmer et al., 2020). Such problem settings are known by the name of *inverse problems*. They are encountered in fields such as particle physics (Stakia, 2021), wireless applications (Orekondy et al., 2023; Pezeshki et al., 2022), medical imaging (Pineda et al., 2020; Bakker et al., 2020, 2022), molecular dynamics (Jonas, 2019) and design (Schwalbe-Koda et al., 2021), and they even find applications in sustainability (Bliek, 2022).

Typically, such settings involve a forward simulator model $f_s : (\boldsymbol{\psi}, \boldsymbol{x}) \to \boldsymbol{y}$ that maps continuous simulator parameters $\boldsymbol{\psi}$ and input data $\boldsymbol{x}$ to observations $\boldsymbol{y}$. For instance, in a

*. Corresponding author. Work done during internship at Qualcomm AI research.

particle physics setting, f_s may simulate the detection of muons $\boldsymbol{y}$, given properties of particles entering the detector $\boldsymbol{x}$ and detector settings $\boldsymbol{\psi}$. Performing optimisations in simulation can be an important step in designing real-life experiments, potentially strongly accelerating experimental research. However, as simulators often require expensive computation, it is often necessary to perform such optimisations with as few simulator calls as possible.

When objective functions are (approximately) differentiable, gradients can be used to guide the optimisation process. For appropriate loss landscapes, notably those that are convex, this can achieve strong optimisation performance (Williams, 1992; Grathwohl et al., 2018; de Avila Belbute-Peres et al., 2018; Hu et al., 2019; Degrave et al., 2019; Mohamed et al., 2020). However, many applications involve non-differentiable simulators or simulators that are complete black-boxes from the perspective of the practitioner (Lei et al., 2017; Cranmer et al., 2020; Omidvar et al., 2022). In this work, we focus on the subset of inverse problems where the goal is to optimise the parameters $\boldsymbol{\psi}$ of some black-box simulator f_s under some loss function $\mathcal{L}(\boldsymbol{y})$. Continuing the particle physics example, such an optimisation could correspond to minimising the number of muon detection events (considered noise) (Shirobokov et al., 2020; Baydin et al., 2021).

In the absence of differentiability or (known) tractable likelihoods, optimisation may be performed using numerical differentiation (Alarie et al., 2021; Shi et al., 2023), evolutionary strategies (Banzhaf et al., 1998; Maheswaranathan et al., 2019), or Bayesian Optimisation (Oh et al., 2018; Daxberger et al., 2020). An alternative is to do gradient-based optimisation using a surrogate model. After training a differentiable surrogate to approximate the black-box simulator, the gradients of the surrogate may be used for optimisation (Shirobokov et al., 2020). These methods start each inverse problem optimisation ab initio/from scratch, which is expensive if the goal is to solve multiple *related* inverse problems. For instance, we may want to efficiently solve the particle physics inverse problem for many different potential input distributions over muon properties $\boldsymbol{x}$. In this work, we aim to solve efficiently inverse problems for such settings. In particular:

1. We propose a simple heuristic for local surrogate optimisation that performs ab-initio optimisation in 50%-75% fewer simulator calls.
2. We train active learning policies to guide the local surrogate optimisation, leading to a further reduction in the number of simulator calls at the cost of an increased fraction of failed optimisations.

2. Background

We aim to optimise the parameters of a stochastic black-box simulator by stochastic gradient descent. Since black-box simulators are not amenable to automatic differentiation methods, we build on the idea to train surrogate neural networks to locally (in $\boldsymbol{\psi}$) mimic the simulator (Shirobokov et al., 2020). Gradients of these local surrogates may then be used to perform the optimisation over $\boldsymbol{\psi}$. We write $\boldsymbol{y} = f_s(\boldsymbol{\psi}, \boldsymbol{x})$ for a stochastic simulator, where $\boldsymbol{y} \sim p(\boldsymbol{y}|\boldsymbol{\psi}, \boldsymbol{x})$ is a random variable and $\boldsymbol{x} \sim q(\boldsymbol{x})$ is a stochastic input. The goal is now to minimise an expected observation loss $\mathcal{L}$ as a function of the simulator parameters $\boldsymbol{\psi}$. As the functional form of the simulator is generally unknown, this expectation cannot be evaluated exactly,

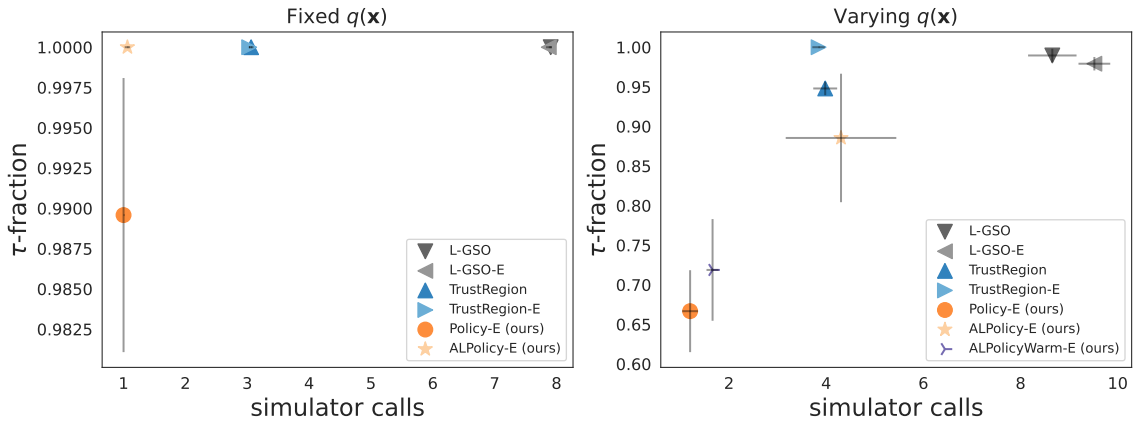


Figure 1: Pareto front for Rosenbrock problem on simulator calls and τ-fraction. Upper-left corners of plots show the best-performing models; lower-right corners show the worst models. $q(\boldsymbol{x})$ is fixed for the left plot and changes every episode on the right. "-E" denotes runs where a surrogate ensemble is used to capture uncertainty. Evaluation is performed over 32 episodes per seed. Error bars denote the standard error of the mean (SEM) over 3 seeds (B.5 for seed details).

and is instead estimated using N Monte Carlo samples:

$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \mathbb{E}\left[\mathcal{L}(\boldsymbol{y})\right] = \arg\min_{\boldsymbol{\psi}} \int \mathcal{L}(\boldsymbol{y})\, p(\boldsymbol{y}|\boldsymbol{\psi}, \boldsymbol{x})\, q(\boldsymbol{x})\, d\boldsymbol{x}\, d\boldsymbol{y}, \tag{1}$$

$$\approx \arg\min_{\boldsymbol{\psi}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_s(\boldsymbol{\psi}, \boldsymbol{x}_i)). \tag{2}$$

After training a neural network surrogate $f_\phi : (\boldsymbol{\psi}, \boldsymbol{x}, \boldsymbol{z}) \to \boldsymbol{y}$ on data generated with f_s, the optimisation may be performed following gradients of this surrogate. Here $\boldsymbol{z}$ is a randomly sampled latent variable that accounts for the stochasticity of the simulator. Gradients are then computed as:

$$\nabla_{\boldsymbol{\psi}} \mathbb{E}\left[\mathcal{L}(\boldsymbol{y})\right] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\psi}} \mathcal{L}\left(f_\phi(\boldsymbol{\psi}, \boldsymbol{x}_i, \boldsymbol{z}_i)\right). \tag{3}$$

Running a simulator forward is often an expensive procedure, so the goal is to minimise the number of simulator calls required (Shirobokov et al., 2020). Local surrogate methods have been shown to outperform optimisation based on numerical gradient estimation (Shi et al., 2023), guided evolutionary strategies (Maheswaranathan et al., 2019), Learning to Simulate (Ruiz et al., 2019), LAX gradients (Grathwohl et al., 2018), and Cylindrical Kernel Bayesian Optimisation (Oh et al., 2018) on this metric.

3. Method

Following Shirobokov et al. (2020), we perform an iterative optimisation based on gradients of equation 3. At each point in the optimisation, Shirobokov et al. (2020) sample $\boldsymbol{\psi}_j$ values in a fixed-size box $U_\epsilon^{\boldsymbol{\psi}}$ – with sides 2ϵ – around the current $\boldsymbol{\psi}$, obtain $\boldsymbol{x}_i$ samples, run the simulator forward to obtain $\boldsymbol{y}$ samples and store these samples in a history H. They train the surrogate from scratch on data extracted from H using a trust-region approach: obtain all samples $(\boldsymbol{\psi}_j, \boldsymbol{x}_i, \boldsymbol{y}_{ji})$ for which $\boldsymbol{\psi}_j$ lies inside the box $U_\epsilon^{\boldsymbol{\psi}}$ centred on the current parameter value. In the following, we will name this procedure a *simulator call*.

We propose to reduce the number of simulator calls required for an optimisation run further with two separate methods that control when data is gathered from the simulator and the local surrogate is retrained. The first method is a simple heuristic, which we name TrustRegion, that only performs a simulator call once the current $\boldsymbol{\psi}$ value is outside of the trust-region box $U_\epsilon^{\boldsymbol{\psi}'}$. Here $\boldsymbol{\psi}'$ is the parameter value at the last-performed simulator call. This heuristic is based on the intuition that the local surrogate should approximate the simulator well for values inside $U_\epsilon^{\boldsymbol{\psi}'}$. Occasionally, repeated gradient steps using the same surrogate lead to loops in the optimisation path. To prevent this issue, TrustRegion also performs a simulator call after υ consecutive steps (30 in our experiments) of not performing one.

The second method uses a learned policy π_θ, parameterised by θ, to decide whether a simulator call is performed or not. The policy is trained as an online actor-critic reinforcement learning agent with Proximal Policy Optimisation (PPO) (Schulman et al., 2017), using Generalised Advantage Estimation (GAE) (Schulman et al., 2016). We formalise the sequential optimisation as an episodic Markov Decision Process (MDP). The state s_t (at timestep t) is given by a tuple $(\boldsymbol{\psi}_t, t, l_t, \sigma_t)$, where $\boldsymbol{\psi}_t$ is the current parameter value, l_t is the number of simulator calls already performed this episode, and σ_t is some notion of uncertainty produced by the surrogate. Actions a_t consist of a binary random variable $b \in \{0, 1\}$, where 1 represents the decision to do a simulator call. Actions optionally include a value ϵ_t, which determines the size of the trust region for sampling new training values $\boldsymbol{\psi}$. Transitions $T(s_t, a_t, s_{t+1})$ consist of a single step of the Adam optimiser (Kingma and Ba, 2014) with learning rate 0.1 using local surrogate gradients computed by equation 3 with $N = 10^4$.

Episodes end when A: the optimisation reaches a parameter for which $\mathbb{E}\left[\mathcal{L}(\boldsymbol{y})\right]$ is below a target value τ (we call this *termination*), B: the maximum number of timesteps T has been reached, or C: the maximum number of simulator calls L has been reached. To incentivise reducing simulator calls, rewards $r(s_t, a_t, s_{t+1})$ are 0 if $b = 0$ and -1 if $b = 1$. To incentivise termination, a reward penalty is added when B or C holds: the penalty is $-(L - l_t) - 1$ when B occurs and -1 when C occurs. This ensures the sum-of-rewards for non-terminating episodes is $-L - 1$. We have observed that primarily using reward penalties based on l_t, rather than t, improves training stability.

As discussed in the introduction, we are often interested in solving multiple *related* inverse problems together. We therefore train our approach on a family of related inverse problems, such that it generalises effectively to test-time problems. In particular, we vary the input distribution $q(\boldsymbol{x})$ between episodes, which corresponds to, e.g., different potential input distributions over muon properties $\boldsymbol{x}$, such as incident angle and energy. The decision to

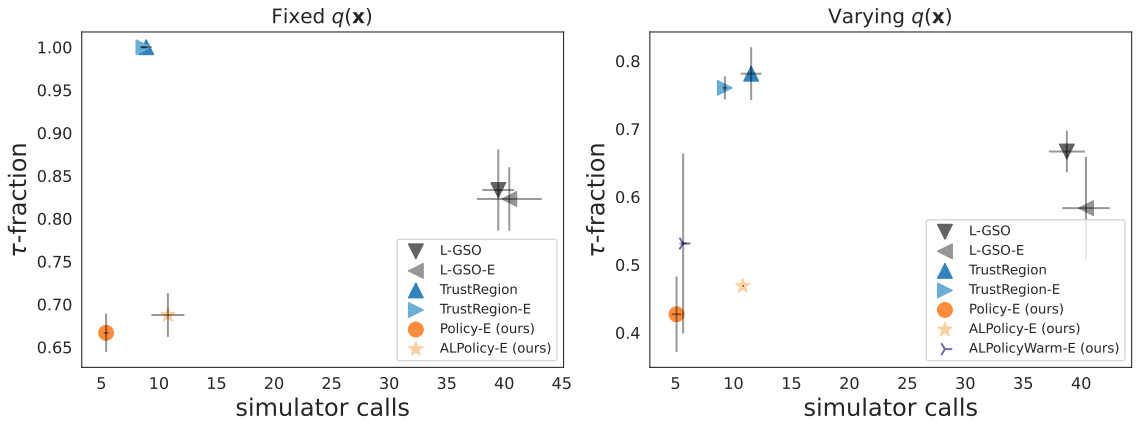


Figure 2: Pareto front for ThreeHump problem on simulator calls and τ-fraction. Upper-left corners of plots show the best-performing models; lower-right corners show the worst models. $q(\boldsymbol{x})$ is fixed for the left plot and changes every episode on the right. "-E" denotes runs where a surrogate ensemble is used to capture uncertainty. Evaluation is performed over 32 episodes per seed. Error bars denote the standard error of the mean (SEM) over 3 seeds (B.5 for seed details).

perform a simulator call or not should depend on the quality of the local surrogate. A surrogate that is well-fitted to the simulator at the current parameter value will presumably provide useful gradients, so gathering additional data and retraining is unnecessary. Vice-versa, a badly fitted surrogate will likely not provide useful gradients and may be worth retraining, even if a simulator call is expensive. We use the uncertainty feature σ to provide this information. To construct σ, we replace the local surrogate with an ensemble of local surrogates, all trained on and applied to the same input data. We compute mean predictions per surrogate on D samples as $\bar{\boldsymbol{y}} = \frac{1}{D}\sum_{i=1}^{D}[f_\phi(\boldsymbol{\psi}, \boldsymbol{x}_i)]$, and construct σ as the standard deviation over these mean predictions. We use three surrogates and $D = 100$ for all experiments that employ the ensemble. See Supplementary B for details.

We have described methods for deciding *when* to do a simulator call, but not *how* to do one: e.g., how to sample data from the simulator for surrogate retraining. To investigate this question, we train policies to additionally output the ϵ for constructing the trust-region $U_\epsilon^{\boldsymbol{\psi}}$, which serves as our data acquisition function. As ϵ parameterises this acquisition function, such policies are an example of active learning (Settles, 2009). In particular, these policies are instances of learning active learning (Hsu and Lin, 2015; Konyushkova et al., 2017; Fang et al., 2017; Ravi and Larochelle, 2018; Pang et al., 2018; Liu et al., 2018; Konyushkova et al., 2018; Bakker et al., 2023), as they learn a distribution over ϵ: see Supplementary B.3.

4. Experiments

We perform experiments on two stochastic simulator functions: the two-dimensional Three-Hump problem, and the ten-dimensional Rosenbrock problem. See Supplementary B.1 for

details. Results are shown in Figures 1 and 2. Here we have plotted the Pareto front of L-GSO (Shirobokov et al., 2020), TrustRegion, and various policy methods on two performance metrics. Our primary goal is to reduce the number of simulator calls for optimisation. However, due to the stochastic nature of the optimisation, some optimisation episodes do not reach the target loss value within the allotted budget and have to be manually ended. We call the fraction of runs that reach the target loss value τ the 'τ-fraction' (plotted on the y-axis). On the x-axis, we plot the number of simulator calls; as such, points further towards the top-left correspond to better overall performance.

We observe that TrustRegion outperforms L-GSO across settings, achieving fewer simulation calls in all cases, and higher τ-fraction in almost all cases. Policy methods often require fewer simulation calls than TrustRegion at the cost of τ-fraction. Both of these effects are more pronounced when $q(\boldsymbol{x})$ varies between episodes, i.e., when the input distribution varies. In the running example of muon detection, this would correspond to doing simultaneous optimisation for various input distributions over muon properties $\boldsymbol{x}$, such as incident angle and energy. In that setting, performance of most methods drops slightly in simulator calls and more significantly in τ-fraction.

Policy methods are split into those that only output *when* to do a simulator call (Policy), and those that also output *how* to do one by providing a trust-region ϵ that parameterises the acquisition function (ALPolicy). L-GSO, TrustRegion and Policy use a fixed value for ϵ that depends on the problem (Supplementary B.1). The active learning policies typically achieve higher τ-fraction than the non-AL policies, but require a few more simulator calls. For fair comparison, we also show results for L-GSO and TrustRegion using the ensemble as their local surrogate model.

Finally, ALPolicyWarm is a version of ALPolicy where the surrogate ensemble is always warm-started from the previous training step, such that the surrogate is continuously improved along the observed trajectories through $\boldsymbol{\psi}$-space (Supplementary B.3). This leads to a Pareto-improvement over ALPolicy on the ThreeHump problem. The improvement on Rosenbrock is less definitive, but ALPolicyWarm lies on the Pareto front here as well. Supplementary C provides some visualisations of resulting optimisation paths.

5. Conclusion and discussion

We have proposed two novel methods for local surrogate-based inverse problem optimisation of black-box simulators that aim to minimise the number of *simulator calls*. Our first contribution – TrustRegion – is a simple heuristic that reduces the required number of simulator calls during optimisation by 50%-75% compared to the L-GSO baseline. Our second proposal is to reinforcement learn an active learning policy that controls when the simulator is used and how to sample data. These policy methods often require even fewer simulator calls than TrustRegion – up to 90% – at the cost of τ-fraction. Our results suggest that inverse problem optimisation may benefit from guidance by both simple policies and learned acquisition functions. We refer to Supplementary A for a discussion of limitations.

References

S. Agostinelli et al. GEANT4 – a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A*, 2003.

Stéphane Alarie, Charles Audet, Aïmen E. Gheribi, Michael Kokkolaras, and Sébastien Le Digabel. Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, 2021.

T. Bakker, M. Muckley, A. Romero-Soriano, M. Drozdzal, and L. Pineda. On learning adaptive acquisition policies for undersampled multi-coil MRI reconstruction. In *Proceedings of Machine Learning Research*, 2022.

Tim Bakker, Herke van Hoof, and Max Welling. Experimental design for MRI by greedy policy search. In *Advances in Neural Information Processing Systems*, 2020.

Tim Bakker, Herke van Hoof, and Max Welling. Learning objective-specific active learning strategies with attentive neural processes. *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery*, 2023.

Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., 1998.

Atılım Güneş Baydin, Kyle Cranmer, Pablo de Castro Manzano, Christophe Delaere, Denis Derkach, Julien Donini, Tommaso Dorigo, Andrea Giammanco, Jan Kieseler, Lukas Layer, Gilles Louppe, Fedor Ratnikov, Giles C. Strong, Mia Tosi, Andrey Ustyuzhanin, Pietro Vischia, and Hevjin Yarar. Toward machine learning optimization of experimental design. *Nuclear Physics News*, 2021.

Laurens Bliek. A survey on sustainable surrogate-based optimisation. *Sustainability*, 2022.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 2020.

Erik Daxberger, Anastasia Makarova, Matteo Turchetta, and Andreas Krause. Mixed-variable bayesian optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2020.

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, 2018.

Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 2019.

Meng Fang, Yuan Li, and Trevor Cohn. Learning how to active learn: A deep reinforcement learning approach. *Empirical Methods in Natural Language Processing*, 2017.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *Proceedings of the International Conference on Learning Representations*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Learning Representations, Workshop*, 2018.

Wei-Ning Hsu and Hsuan-Tien Lin. Active learning by learning. *Association for the Advancement of Artificial Intelligence*, 2015.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *International Conference on Robotics and Automation*, 2019.

Eric Jonas. Deep imitation learning for molecular inverse problems. In *Advances in Neural Information Processing Systems*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014. doi: 10.48550/arXiv.1412.6980.

Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning Active Learning from Data. *Advances in Neural Information Processing Systems*, 2017.

Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Discovering General-Purpose Active Learning Strategies. *arXiv preprint*, 2018. doi: 10.48550/arXiv.1810.04114.

Gang Lei, Jianguo Zhu, Youguang Guo, Chengcheng Liu, and Bo Ma. A review of design optimization methods for electrical machines. *Energies*, 2017.

Ming Liu, Wray Buntine, and Gholamreza Haffari. Learning how to actively learn: A deep imitation learning approach. *Association for Computational Linguistics*, 2018.

Niru Maheswaranathan, Luke Metz, George Tucker, Dami Choi, and Jascha Sohl-Dickstein. Guided evolutionary strategies: augmenting random search with surrogate gradients. In *Proceedings of the International Conference on Machine Learning*, 2019.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 2020.

Samuel Neumann, Sungsu Lim, Ajin George Joseph, Yangchen Pan, Adam White, and Martha White. Greedy actor-critic: A new conditional cross-entropy method for policy improvement. In *Proceedings of the International Conference on Learning Representations*, 2023.

ChangYong Oh, Efstratios Gavves, and Max Welling. BOCK : Bayesian optimization with cylindrical kernels. In *Proceedings of the International Conference on Machine Learning*, 2018.

Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. A review of population-based metaheuristics for large-scale black-box global optimization — part II. *IEEE Transactions on Evolutionary Computation*, 2022.

Tribhuvanesh Orekondy, Pratik Kumar, Shreya Kadambi, Hao Ye, Joseph Soriaga, and Arash Behboodi. WineRT: Towards neural ray tracing for wireless channel modelling and differentiable simulations. In *Proceedings of the International Conference on Learning Representations*, 2023.

Kunkun Pang, Mingzhi Dong, Yang Wu, and Timothy Hospedales. Meta-Learning Transferable Active Learning Policies by Deep Reinforcement Learning. *arXiv preprint*, 2018. doi: 10.48550/arXiv.1806.04798.

Hamed Pezeshki, Fabio Valerio Massoli, Arash Behboodi, Taesang Yoo, Arumugam Kannan, Mahmoud Taherzadeh Boroujeni, Qiaoyu Li, Tao Luo, and Joseph B Soriaga. Beyond codebook-based analog beamforming at mmwave: Compressed sensing and machine learning methods. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 776–781. IEEE, 2022.

Luis Pineda, Sumana Basu, Adriana Romero, Roberto Calandra, and Michal Drozdzal. Active MR k-space sampling with reinforcement learning. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2020.

Sachin Ravi and Hugo Larochelle. Meta-learning for batch mode active learning. *Proceedings of the International Conference on Learning Representations*, 2018.

Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. Learning to simulate. In *Proceedings of the International Conference on Learning Representations*, 2019.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint*, 2017. doi: 10.48550/arXiv.1707.06347.

Daniel Schwalbe-Koda, Aik Rui Tan, and Rafael Gómez-Bombarelli. Differentiable sampling of molecular geometries with uncertainty-based adversarial attacks. In *Nature Communications*, 2021.

Burr Settles. Active learning literature survey. Technical report, University of Wisconsin–Madison, 2009.

Hao-Jun Michael Shi, Melody Qiming Xuan, Figen Oztoprak, and Jorge Nocedal. On the numerical performance of finite-difference-based methods for derivative-free optimization. *Optimization Methods and Software*, 2023.

Sergey Shirobokov, Vladislav Belavin, Michael Kagan, Andrei Ustyuzhanin, and Atilim Gunes Baydin. Black-box optimization with local generative surrogates. In *Advances in Neural Information Processing Systems*, 2020.

Anna Stakia. Advanced multivariate analysis methods for use by the experiments at the large hadron collider. *Physica Scripta*, 2021.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

Appendix A. Limitations and future work

The primary limitation of our current work is the lack of evaluation on real-life simulators. We are currently aiming to apply our work to the muon detection particle physics problem mentioned in the main text, using the GEANT4 simulator (Agostinelli et al., 2003).

Gradient-based optimisation may get stuck in local optima of the loss surface $\mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{\psi},\boldsymbol{x})}\left[\mathcal{L}(\boldsymbol{y})\right]$. Investigating whether introducing a policy into the optimisation can help avoid such local minima, is an interesting direction of future research. The ThreeHump problem has no local minima but does contain a few flat regions, where gradient-based optimisation is more challenging. Exploratory experience have provided weak evidence that the policy may learn to avoid such regions.

Hyperparameter tuning has mostly involved reducing training variance through tuning the number of episodes used for a PPO iteration, as well as setting learning rates and the KL-threshold. Little effort has been spent optimising the policy or surrogate architectures; we expect doing so to further improve performance. Similarly, while PPO with a value function critic is a widely used algorithm, more recent algorithms may offer additional advantages, such as improved planning and off-policy learning for more data-efficient training (Haarnoja et al., 2018; Neumann et al., 2023).

Appendix B. Implementation details

B.1 Simulators

Our experiments use two stochastic simulator functions: ThreeHump and Rosenbrock. ThreeHump is a two-dimensional problem that lends itself well to visualisation, while the N-dimensional Rosenbrock problem is used to test our method in a higher-dimensional setting. We choose $N = 10$ in our implementation. Both of these simulators have scalar output values $\boldsymbol{y} = y$.

ThreeHump: Find the 2-dimensional $\boldsymbol{\psi}$ that optimises:[1]

$$\begin{gathered}\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \mathbb{E}\left[\mathcal{L}(\boldsymbol{y}\right] = \arg\min_{\boldsymbol{\psi}} \mathbb{E}\left[\sigma(y-10) - \sigma(y)\right], \text{s.t.} \\ y \sim \mathcal{N}(y|\mu_i, 1),\ i \in \{1,2\},\ \mu_i \sim \mathcal{N}(x_i\, h(\boldsymbol{\psi}, 1),\ x_1 \sim U[-2,2],\ x_2 \sim U[0,5], \\ P(i=1) = \frac{\psi_1}{||\boldsymbol{\psi}||_2} = 1 - P(i=2),\ h(\boldsymbol{\psi} = 2\psi_1^2 - 1.05\psi_1^4 + \psi_1^6/6 + \psi_1\psi_2 + \psi_2^2.\end{gathered} \tag{4}$$

We consider an episode terminated when $\mathbb{E}\left[\mathcal{L}(\boldsymbol{y}\right] = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f_s(\boldsymbol{\psi}, \boldsymbol{x}_i)) \leq \tau = -0.8$, which we evaluate after every optimisation step using $N = 10^4$ samples. Following (Shirobokov et al., 2020), we use $\epsilon = 0.5$ as the trust-region size. The optimisation is initialised at $\boldsymbol{\psi}_0 = [2.0, 0.0]$; this is a symmetry point in the ThreeHump function such that optimisation with stochastic gradients can fall into either of the two wells around the two minima of the

1. Here the upper bound of x_1 and lower bound of x_2 are switched compared to the notation in Equation (3) of (Shirobokov et al., 2020). These bounds match the official implementation of L-GSO as of August 2023. Private correspondence with the authors has confirmed that this is the intended implementation.

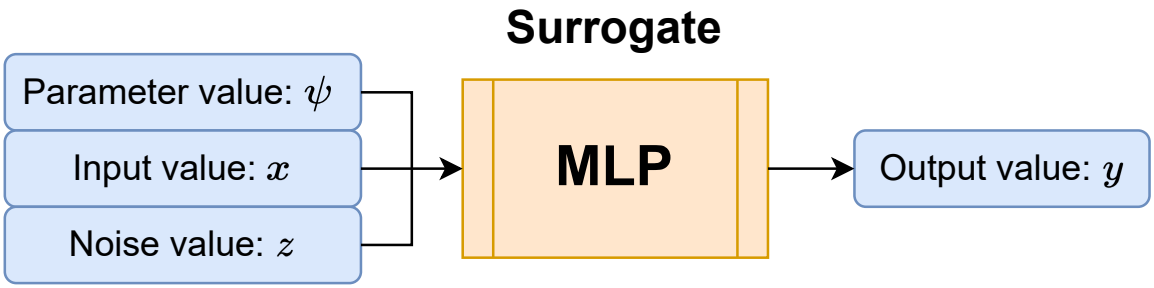


Figure 3: Schematic for the surrogate architecture. The surrogate is an MLP trained to mimic the simulator. It takes $(\boldsymbol{\psi}, \boldsymbol{x}, \boldsymbol{z})$ as input and outputs $\boldsymbol{y}$. The ensemble consists of 3 of these, trained on the same data with different seeds.

function. This requires our methods to learn good paths to both of these optima, making the task more interesting. In principle, optimisation could be initialised at any $\boldsymbol{\psi}_0$.

To test the generalisation behaviour of our method, we parameterise this target function further, by placing distributions on the bounds of the Uniform distributions that x_1 and x_2 are sampled from. We sample new bounds every episode, such that the policy sees multiple related – but different – simulators during training (and evaluation).

In particular, we sample lower and upper bounds of x_1 from respectively $\mathcal{N}(-2, 0.5)$ and $\mathcal{N}(2, 0.5)$. For x_2 we instead use $\mathcal{N}(0, 1)$ and $\mathcal{N}(5, 1)$. Note that this occasionally means episodes cannot terminate, as the specified termination value τ is below the minimum loss value for some samplings.

Rosenbrock: Find the N-dimensional $\boldsymbol{\psi}$ that optimises:

$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \mathbb{E}\left[\mathcal{L}(\boldsymbol{y}\right] = \arg\min_{\boldsymbol{\psi}} \mathbb{E}\left[y\right], \text{s.t.}$$

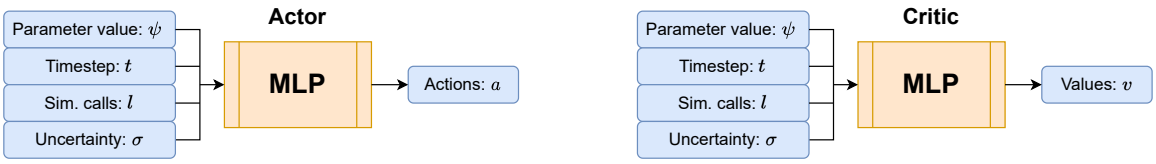
$$y \sim \mathcal{N}\left(y \middle| \sum_{i=1}^{N-1} \left[(\psi_i - \psi_{i+1})^2 + (1 - \psi_i)^2\right] + \mu, 1\right),\ \mu \sim N(\mu|x, 1),\ x \sim U[-10, 10]. \quad (5)$$

We consider an episode terminated when $\mathbb{E}\left[\mathcal{L}(\boldsymbol{y}\right] = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_s(\boldsymbol{\psi}, \boldsymbol{x}_i)) \leq \tau = 3.0$, which we evaluate after every optimisation step using $N = 10^4$ samples. Following (Shirobokov et al., 2020), we use $\epsilon = 0.2$ as the trust-region size. The optimisation is initialised at $\overrightarrow{2.0} \in \mathbb{R}^{10}$. When testing generalisation behaviour, we sample lower and upper bounds of x from respectively $\mathcal{N}(0, 2)$ and $\mathcal{N}(10, 2)$.

B.2 Surrogate

The surrogate consists of ReLU MLP with 2 hidden layers of 256 neurons that takes as input $(\boldsymbol{\psi}, \boldsymbol{x}, \boldsymbol{z})$ and outputs $\boldsymbol{y}$. $\boldsymbol{z}$ is sampled from a 100-dimensional diagonal unit Normal distribution. The surrogate architecture is schematically depicted in Figure 3.

Surrogates are trained on data generated from f_s. Following Shirobokov et al. (2020), we sample M values $\boldsymbol{\psi}_j$ inside the box $U_\epsilon^{\boldsymbol{\psi}}$ around the current parameter value using an adapted



(a) Actor. (b) Critic.

Figure 4: Schematic for the policy architecture. The policy consists of a separate Actor and Critic, which are both MLPs. They take $(\boldsymbol{\psi}, t, l, \sigma)$ as input and output the actions and value function estimates. Actions always contain the decision b to do a simulator call or not, and may optionally also contain a value ϵ used for surrogate training data sampling.

Latin Hypercube sampling algorithm. For each of those $\boldsymbol{\psi}_j$, we then sample $N = 3 \cdot 10^3$ x-values. We use $M = 5$ for the ThreeHump problem and $M = 16$ for the Rosenbrock problem. As in Shirobokov et al. (2020), this means a single 'simulator call' consists of $1.5 \cdot 10^4$ function evaluations for ThreeHump, and $4.8 \cdot 10^4$ for Rosenbrock.

Surrogates are trained with the Adam optimiser for 2 epochs with a learning rate of 10^{-3} and batch size 512. The ensembles consist of 3 surrogates, each trained on the same data, but a different random seed.

B.3 Policy

The policy π_θ consists of a separate Actor and Critic neural network. Both are ReLU MLPs with a single hidden layer of 256 neurons, schematically depicted in Figure 4. Both networks take as input a tuple $(\boldsymbol{\psi}_t, t, l_t, \sigma_t)$, where $\boldsymbol{\psi}_t$ is the current parameter value (at timestep t), l_t is the number of simulator calls already performed this episode, and σ_t is the standard deviation over average surrogate predictions in the ensemble.

The Actor outputs either one or three values. The first is passed through a sigmoid activation and treated as a Bernoulli random variable from which b (the decision to do a simulator call or not) is sampled. If the policy outputs three values, the second and third are treated as the mean and standard deviation of a lognormal distribution from which we sample ϵ for the current timestep. The value corresponding to the standard deviation is first passed through a softplus activation, to ensure it is positive.

The critic outputs a value-function estimate $V_\theta(s)$, where θ are policy parameters, that we use for computing advantage estimates in PPO. See section B.4 for details. Since rewards have unity order of magnitude, we expect return values to be anywhere in $[-T, 0]$. To prevent scaling issues, we multiply the critic output values by T before using them for advantage estimation.

The ALPolicyWarm method: When optimising a policy for downstream optimisation of many related inverse problems, it may be useful to simultaneously train a global surrogate for such a problem setting. Such a global surrogate may provide better gradients for inverse problem optimisation, especially if it has been jointly optimised with the policy. To test this, we have implemented the ALPolicyWarm method. Here the policy outputs both the decision

to do a simulator call and the trust-region ϵ, just as in ALPolicy. Here however, the surrogate ensemble is warm-started from the previous training step every time a retraining decision is made. This results in a surrogate ensemble that is continuously optimised for trajectories seen during training. In order for the surrogate to not forget old experience too quickly, we employ a replay buffer that undersamples data from earlier iterations geometrically. I.e., when training the surrogate with trust-region $U_\epsilon^{\boldsymbol{\psi}}$, we include all data inside $U_\epsilon^{\boldsymbol{\psi}}$ for the current episode, half the data inside $U_\epsilon^{\boldsymbol{\psi}}$ from the previous episode, a quarter of the data seen two episodes ago, etc. We train this model only for settings with varying $q(\boldsymbol{x})$, as only these provide information about the generalisation of the learned policies.

B.4 Training

We train our policy in episodic fashion by accumulating sequential optimisation episodes and updating the policy using PPO (Schulman et al., 2017) with GAE advantages (Schulman et al., 2016) (discount factor $\gamma = 1.0$, GAE $\lambda = 0.95$. Episodes terminate once A: the target loss value τ has been reached, B: the number of timesteps $T = 1000$ has been reached, or C: the number of simulation calls $L = 50$ has been reached. Every training iteration we accumulate 4 episodes before doing PPO updates. We train for a total of 100 iterations.

Actor updates are performed using the PPO-clip objective (with clip value 0.2) on full trajectories with no entropy regularisation. We perform multiple Actor updates with the same experience until either the empirical KL-divergence between old and new policy reaches a threshold ($3 \cdot 10^{-3}$ for simulator-call decision actions, 10^{-2} for trust-region size ϵ actions), or 20 updates have been performed. In practice, we rarely perform the full 20 updates. Updates use Adam with learning rate $3 \cdot 10^{-4}$.

Similarly, we perform multiple Critic updates using the Mean-Squared Error (MSE) between the estimated values $V_\theta(s_t)$ and the observed return (sum of rewards, as $\gamma = 1.0$) R_t at every timestep. We keep updating until either MSE ≤ 30.0 or 10 updates have been done. This helps the critic learn quickly initially and after seeing very surprising episodes, but prevents it from over-updating on very similar experience (as MSE will be low for those iterations). Updates use Adam with learning rate 10^{-4}.

See algorithm 1 for training pseudo-code. Evaluation is performed using algorithm 2 on 32 optimisation episodes.

B.5 Crashed seeds

Results depicted in Figures 1 and 2 are generally the average of 3 seeds. However, some runs crashed, resulting in fewer seeds averaged. In Figure 2 on the right, ALPolicy has 1 seed and ALPolicyWarm has 2 seeds. In Figure 1 on the right, ALPolicy has 2 seeds.

Appendix C. Additional analyses

Figures 5 and 6 depict examples of learned optimisation trajectories by the ALPolicy method on the ThreeHump problem. Example learning curves are depicted in Figure 7.

Algorithm 1: Training the (active learning) policy.

Data: Simulator $f_s(\boldsymbol{\psi}, \boldsymbol{x})$; surrogate $f_\phi(\boldsymbol{\psi}, \boldsymbol{x})$; observation loss function $\mathcal{L}$; policy π_θ; number N of $\boldsymbol{\psi}$ to sample when training the surrogate; number M of of $\boldsymbol{x}$ to sample for each $\boldsymbol{\psi}$; distributions $Q(q)$ over distributions $q(\boldsymbol{x})$ to sample $\boldsymbol{x}$ from; initial value $\boldsymbol{\psi}_0$; target function value τ; number T of timesteps to run each simulation for (episode-length); maximum number of simulator calls L; $\boldsymbol{\psi}$ optimiser OPTIM_ψ with learning rate λ; number of policy training iterations K; number of episodes to accumulate for a PPO step G; policy optimiser OPTIM_π; reward function $\mathcal{R}$; experience buffer B; discount factor γ.

for $k \in (1, ..., K)$ **do**
- Empty experience buffer B.
- **for** $\_ \in (1, ..., G)$ **do**
  - Initialise number of simulator calls done: $l \leftarrow 0$.
  - Set return: $R \leftarrow 0$.
  - Sample $\boldsymbol{x}$-distribution $q \sim Q$.
  - **for** $t \in (1, ..., T)$ **do**
    - Sample $\boldsymbol{x} \sim q(x)$.
    - Obtain surrogate features σ (e.g., ensemble uncertainty) from surrogate $f_\phi(\boldsymbol{\psi}_t, \boldsymbol{x})$.
    - Construct state: $s \leftarrow (\boldsymbol{\psi_t}, t, l, \sigma)$.
    - Obtain action: $a = (\texttt{do\_retrain}, \texttt{trust\_region\_size}) \leftarrow \pi_\theta(s)$.
    - **if** *do_retrain* **then**
      - Obtain N samples $\boldsymbol{\psi}_n$ from trust region with size `trust_region_size`.
      - Obtain M samples $\boldsymbol{x}_m$ for each of these $\boldsymbol{\psi}_n$.
      - Combine into dataset $\{\boldsymbol{\psi}, \{\boldsymbol{x}\}^M\}^N$ and optionally filter or include data from previous timesteps.
      - Retrain surrogate: f_ϕ on this dataset.
      - Increment number of simulator calls: $l \leftarrow l + 1$.
    - **end**
    - Obtain surrogate gradients: $\boldsymbol{g}_t \leftarrow \nabla_\psi f_\phi(\boldsymbol{\psi}, \boldsymbol{x})|_{\boldsymbol{\psi}_t}$.
    - Do optimisation step: $\boldsymbol{\psi}_{t+1} \leftarrow \text{OPTIM}_\psi(\boldsymbol{\psi}_t, \boldsymbol{g}_t, \lambda)$.
    - `terminated` $\leftarrow \mathbb{E}\left[\mathcal{L}(f_s(\boldsymbol{\psi}_t, \boldsymbol{x})\right] \leq \tau$
    - Obtain reward: $r \leftarrow \mathcal{R}(s, a, \boldsymbol{\psi}_{t+1})$.
    - Store (s, a, r) and any other relevant information in buffer B.
    - **if** *terminated* **then**
      - `break`
    - **end**
    - **if** l *equals* L **then**
      - `break`
    - **end**
  - **end**
- **end**
- Update policy $\pi_\theta \leftarrow \text{OPTIM}(\pi_\theta, B, \gamma)$.

end

Algorithm 2: Inference with the (active learning) policy.

Data: Simulator $f_s(\boldsymbol{\psi}, \boldsymbol{x})$; surrogate $f_\phi(\boldsymbol{\psi}, \boldsymbol{x})$; observation loss function $\mathcal{L}$; trained policy π_θ; number N of $\boldsymbol{\psi}$ to sample when training the surrogate; number M of of $\boldsymbol{x}$ to sample for each $\boldsymbol{\psi}$; distributions $q(\boldsymbol{x})$ to sample $\boldsymbol{x}$ from; initial value $\boldsymbol{\psi}_0$; target function value τ; number T of timesteps to run each simulation for (episode-length); maximum number of simulator calls L; $\boldsymbol{\psi}$ optimiser OPTIM_ψ with learning rate λ.

for $t \in (1, ..., T)$ **do**

Initialise number of simulator calls done: $l \leftarrow 0$.

Sample $\boldsymbol{x} \sim q(x)$.

Obtain surrogate features σ (e.g., ensemble uncertainty) from surrogate $f_\phi(\boldsymbol{\psi}_t, \boldsymbol{x})$.

Construct state: $s \leftarrow (\boldsymbol{\psi_t}, t, l, \sigma)$.

Obtain action: $a = (\texttt{do\_retrain}, \texttt{trust\_region\_size}) \leftarrow \pi_\theta(s)$.

if *do_retrain* **then**

Obtain N samples $\boldsymbol{\psi}_n$ from trust region with size `trust_region_size`.

Obtain M samples $\boldsymbol{x}_m$ for each of these $\boldsymbol{\psi}_n$.

Combine into dataset $\{\boldsymbol{\psi}, \{\boldsymbol{x}\}^M\}^N$ and optionally filter or include data from previous timesteps.

Retrain surrogate: f_ϕ on this dataset.

Increment number of simulator calls: $l \leftarrow l + 1$.

end

Obtain surrogate gradients: $\boldsymbol{g}_t \leftarrow \nabla_\psi f_\phi(\boldsymbol{\psi}, \boldsymbol{x})|_{\boldsymbol{\psi}_t}$.

Do optimisation step: $\boldsymbol{\psi}_{t+1} \leftarrow \text{OPTIM}_\psi(\boldsymbol{\psi}_t, \boldsymbol{g}_t, \lambda)$.

`terminated` $\leftarrow \mathbb{E}\left[\mathcal{L}(f_s(\boldsymbol{\psi}_t, \boldsymbol{x})\right] \leq \tau$

if *terminated* **then**

`break`

end

if *l equals L* **then**

`break`

end

end

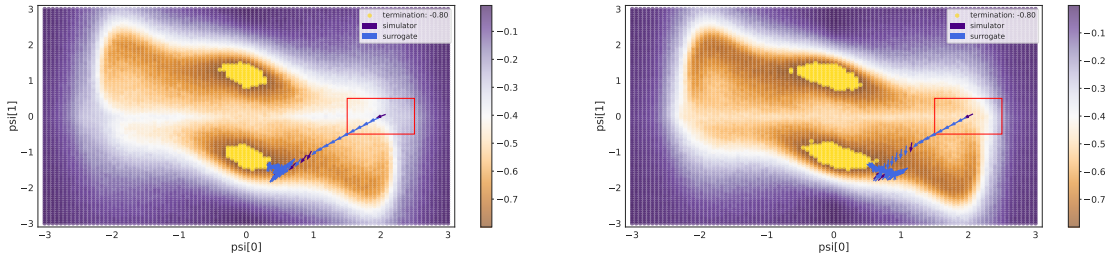


Figure 5: Examples of learned optimisation trajectories on a heatmap of the ThreeHump problem. Shown is $\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f_s(\boldsymbol{\psi}, \boldsymbol{x}_i))$ for a grid of $\boldsymbol{\psi}$ values ($N = 100$). The yellow region denotes values of $\boldsymbol{\psi}$ that lead to termination. The $\epsilon = 0.5$ trust-region box around $\boldsymbol{\psi}_0$ is visualised by the red box. Trajectories are sampled from the final evaluation iteration of ALPolicy: shown here are two trajectories that terminate.

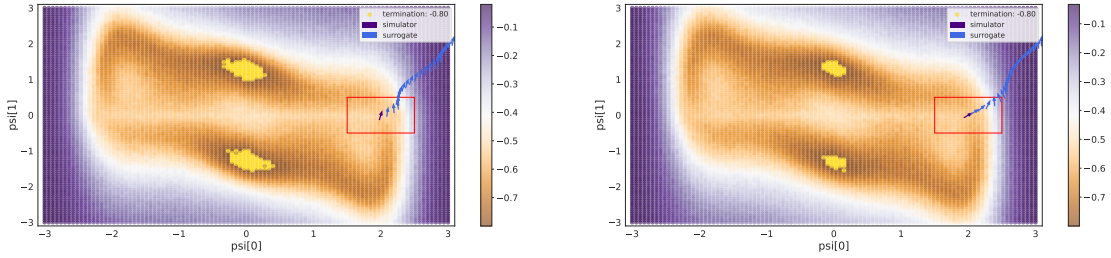
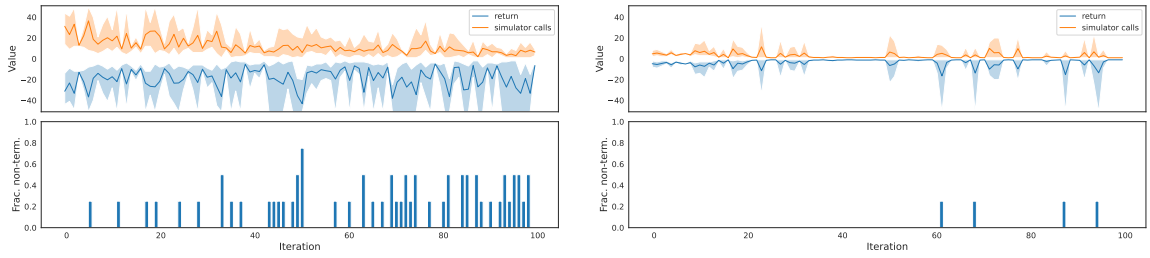


Figure 6: Examples of learned optimisation trajectories on a heatmap of the ThreeHump problem. Shown is $\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f_s(\boldsymbol{\psi}, \boldsymbol{x}_i))$ for a grid of $\boldsymbol{\psi}$ values ($N = 100$). The yellow region denotes values of $\boldsymbol{\psi}$ that lead to termination. The $\epsilon = 0.5$ trust-region box around $\boldsymbol{\psi}_0$ is visualised by the red box. Trajectories are sampled from the final evaluation iteration of ALPolicy: shown here are two trajectories that do not terminate.



(a) ThreeHump. (b) Rosenbrock.

Figure 7: Learning curves for ALPolicy on a) ThreeHump and b) Rosenbrock with fixed $q(\boldsymbol{x})$. The upper plots show average return and number of simulator calls per iteration (which consists of multiple episodes). Shaded regions are the min-max of all the episodes for an iteration. Bottom plots show the fraction of episodes that do *not* terminate in an iteration. Note that Rosenbrock seems much easier to learn than ThreeHump.