Learning to Learn for Global Optimization of Black Box Functions

Yutian Chen **Matthew W. Hoffman** **Sergio Gómez Colmenarejo**
Misha Denil **Timothy P. Lillicrap** **Nando de Freitas**
DeepMind
{yutianc,mwhoffman,sergomez,mdenil,countzero,nandodefreitas}@google.com

Abstract

We present a learning to learn approach for training recurrent neural networks to perform black-box global optimization. In the meta-learning phase we use a large set of smooth target functions to learn a recurrent neural network (RNN) optimizer, which is either a long-short term memory network or a differentiable neural computer. After learning, the RNN can be applied to learn policies in reinforcement learning, as well as other black-box learning tasks, including continuous correlated bandits and experimental design. We compare this approach to Bayesian optimization, with emphasis on the issues of computation speed, horizon length, and exploration-exploitation trade-offs.

1 Introduction

Findings in developmental psychology have revealed that infants are endowed with a small number of separable systems of core knowledge for reasoning about objects, actions, number, space, and possibly social interactions [Spelke and Kinzler, 2007]. These core systems enable infants to learn a vast set of skills and knowledge rapidly. The most coherent explanation of this phenomena is that the slow learning (or optimization) process of evolution has led to the emergence of components that enable fast and varied forms of learning. In psychology, learning to learn has a long history full of insights on cognition [Ward, 1937, Harlow, 1949, Kehoe, 1988].

Inspired by this, many researchers have tried to build agents capable of learning to learn [Schmidhuber, 1987, Bengio et al., 1992, Hochreiter et al., 2001, Santoro et al., 2016, Andrychowicz et al., 2016, Zoph and Le, 2016]. The scope of research under the umbrella of learning to learn, or meta-learning, as it is often called in this literature, is remarkably broad, and can be categorized along several dimensions. The learner can implement and be trained by many different algorithms, including gradient descent, genetic algorithms, simulated annealing, and reinforcement learning algorithms, leading to a large space of possibilities.

For instance, one can learn to learn by gradient descent by gradient descent, or learn local Hebbian updates by gradient descent [Andrychowicz et al., 2016, Bengio et al., 1992]. In the former, one uses supervised learning at the meta-level to learn an algorithm for supervised learning, while in the latter, one uses supervised learning at the meta-level to learn an algorithm for unsupervised learning. Other types of learning, including reinforcement learning, can be easily added into this mix.

The product of meta-learning is another valuable dimension for categorizing research efforts in this field. In Andrychowicz et al. [2016] the product of meta-learning is a trained RNN, which is subsequently used as an optimization algorithm to fit other models to data or optimize any differentiable objective function. In contrast, in Zoph and Le [2016] the output of meta-learning may also be an RNN, but in their case the RNN is subsequently used as a model that is fit to data using a classical optimizer. In both cases the output of meta-learning is an RNN, but this RNN is interpreted and applied as a model or as an algorithm. In this sense, learning to learn with neural networks blurs the classical distinction between models and algorithms. For a more thorough review of related works on learning to learn, please consult Appendix A.

In this work, the product of meta-learning is an algorithm for globally optimizing black-box functions. In this case, the meta learner can be trained using gradients and distillation. Specifically, we address the problem of finding a global minimizer of an unknown (black-box) loss function f:

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})\,, \tag{1}$$

where $\mathcal{X}$ is some search space of interest, which in this work is assumed to be a continuous space. The *black-box* function f is not available to the learner in simple closed form at test time, but can be evaluated at a query point $\mathbf{x}$ in the domain. This evaluation produces either deterministic or stochastic outputs $y \in \mathbb{R}$ such that $f(\mathbf{x}) = \mathbb{E}[y \mid f(\mathbf{x})]$. In other words, we can only observe the function f through unbiased noisy point-wise observations y.

Black-box optimization has a long history in machine learning and experimental design. While many tools have been proposed to attack these problems, Bayesian optimization techniques have gained a substantial amount of interest in recent years [Brochu et al., 2009, Shahriari et al., 2016].

Bayesian optimization is a sequential model-based approach to solving problem (1). It has two components. The first component is a probabilistic model, consisting of a prior distribution that captures our beliefs about the behavior of the unknown objective function and an observation model that describes the data generation mechanism. The model can be a Beta-Bernoulli bandit, a random forest with frequentist confidence estimates, a Dirichlet process prior, a Bayesian neural network, or a Gaussian process [Shahriari et al., 2016]. This wide variety of possible models highlights how broad and encompassing this approach can be, despite the fact that it is often associated with Gaussian processes, to the point of receiving the name of Gaussian process bandits in the literature [Srinivas et al., 2010, Kaufmann et al., 2012].

The second component is an acquisition function, which is optimized at each step so as to trade-off exploration and exploitation. Here again we encounter a huge variety of strategies, including Thompson sampling [Thompson, 1933, Scott, 2010, Chapelle and Li, 2011, Agrawal and Goyal, 2013], information gain [Villemonteix et al., 2009, Hennig and Schuler, 2012, Russo and Van Roy, 2014, Shahriari et al., 2014, Hernández-Lobato et al., 2015], probability of improvement [Kushner, 1964, Jones et al., 1998], expected improvement [Močkus et al., 1978, Jones et al., 1998, Jones, 2001, Brochu et al., 2009, Snoek et al., 2012, Wang and de Freitas, 2014], upper confidence bounds [Lai and Robbins, 1985, Srinivas et al., 2010], and so on.

The requirement for optimizing the acquisition function at each step can be a significant cost, as we will illustrate in the empirical section of this paper. It also raises some theoretical concerns [Wang et al., 2014].

In this paper, we will present a learning to learn approach for global optimization of black box functions and contrast it with Bayesian optimization. In the meta-learning phase, we will use a large number of differentiable functions generated with a Gaussian process to train an RNN optimizer. We will make use of two types of RNN: long-short-term memory networks (LSTMs) by Hochreiter and Schmidhuber [1997] and differentiable neural computers (DNCs) by Graves et al. [2016]. In our experiments the RNN uses its memory to store information about previous queries and learns to access its memory to make decisions about which parts of the domain to explore or exploit next. We also consider the use of acquisition functions to guide the process of training the RNN optimisers.

After learning the optimiser, it can be applied to any black-box function. By unrolling the RNN, we generate new candidates for the search process. This process is much faster than applying standard Bayesian optimization, and in particular it does not involve either matrix inversion or optimization of acquisition functions.

During meta-learning, we choose the horizon (number of steps) of the optimization process. We are therefore considering the finite horizon setting that is popular in AB tests Kohavi et al. [2009], Scott [2010] and is often studied under the umbrella of best arm identification in the bandits literature [Bubeck et al., 2009, Gabillon et al., 2011, 2012, Hoffman et al., 2014].

Our experiments will show that the RNN optimizer is faster and tends to achieve better performance within the horizon for which it is trained. It however underperforms against Bayesian optimization for much longer horizons as it has not learned to explore for longer horizons. These experiments also demonstrate a remarkable degree of transfer: We learn an RNN optimizer using synthetic functions, and then successfully apply it to a broad range of benchmarks from the global optimization literature as well as a simple control problem.

2 Learning Black-box Optimization

A black-box optimization algorithm can be summarized by the following loop:

1. Given the current state of knowledge $\mathbf{h}_t$ propose a query point $\mathbf{x}_t$
2. Observe the response y_t
3. Update any internal statistics to produce $\mathbf{h}_{t+1}$

This easily maps onto the classical frameworks presented in the previous section where the update step computes statistics and the query step uses these statistics for exploration. In this work we take this framework as a starting point and define a combined update and query rule using a recurrent neural network (RNN) parameterized by θ such that

$$\mathbf{h}_t, \mathbf{x}_t = \mathrm{RNN}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, y_{t-1}), \tag{2}$$

$$y_t \sim p(y \mid \mathbf{x}_t) \ . \tag{3}$$

Intuitively this rule can be seen to update its hidden state using data from the previous time step and then propose a new query point. In what follows we will apply this RNN, with shared parameters, to many steps of a black-box optimization process. An example of this computation is shown in Figure 1.

Given this rule we now need a way to learn the parameters θ for any given distribution of functions $p(f)$. Perhaps the simplest loss function one could use is the loss of the final iteration: $L_{\text{final}}(\theta) = \mathbb{E}_f[f(\mathbf{x}_T)]$ for some time-horizon T. This loss was considered by Andrychowicz et al. [2016] in the context of learning first-order optimizers, but ultimately rejected in favor of

$$L_{\text{obs}}(\theta) = \mathbb{E}_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} f(\mathbf{x}_t)\right]. \tag{4}$$

A key reason to prefer L_{obs} is that the amount of information conveyed by L_{final} is temporally very sparse. By instead utilizing a sum of losses to train the optimizer we are able to provide information from every step along this trajectory. Note also that optimizing the summed loss is equivalent to finding a strategy which minimizes the expected cumulative regret. Note that although at test time the optimizer typically only has access to the observation y_t, at training time the true loss can be used.

By using the above objective function we will be encouraged to trade off exploration and exploitation and hence globally optimize the function f. This is due to the fact that in expectation, any method that is better able to explore and find small values $f(\mathbf{x})$ will be rewarded for these discoveries. However the actual process of optimizing the above loss can be difficult due to the fact that nothing explicitly encourages the optimizer itself to explore.

We can encourage exploration by encoding an exploration force directly into the loss function for meta learning. Many such examples exist in the bandit and Bayesian optimization communities, for example

$$L_{\text{EI}}(\theta) = -\mathbb{E}_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} \mathrm{EI}(\mathbf{x}_t \mid y_{1:t-1})\right] \tag{5}$$

where $\mathrm{EI}(\cdot)$ is the expected posterior improvement of querying $\mathbf{x}_t$ given observations up to time t. This can encourage exploration by giving an explicit bonus to the optimizer rather than just implicitly doing so by means of function evaluations. Alternatively, it is possible to use the observed improvement (OI)

$$L_{\text{OI}}(\theta) = \mathbb{E}_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} (f(\mathbf{x}_t) - \min_{i<t}(f(\mathbf{x}_i)))\right] \tag{6}$$

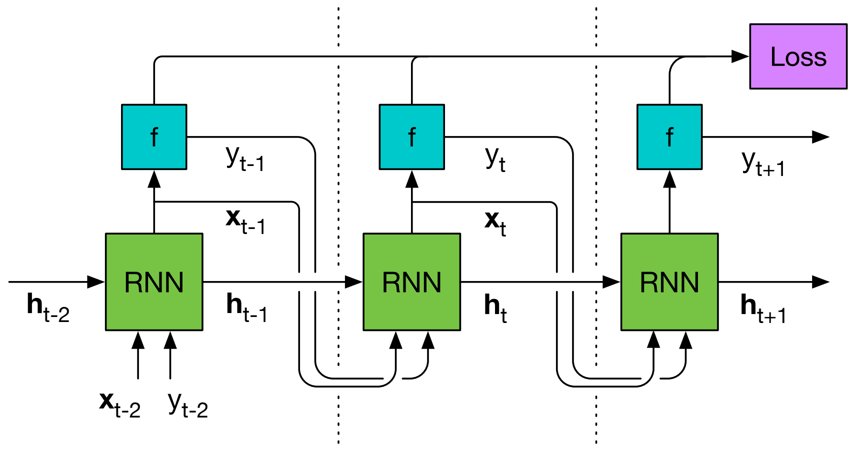

Figure 1: Computational graph of the learned black-box optimizer unrolled over multiple steps. The learning process will consist of differentiating the given loss with respect to the RNN parameters

The illustration of Figure 1 shows the optimizer unrolled over many steps, ultimately culminating in the loss function. To train the optimizer we will simply take derivatives of the loss with respect to the RNN parameters θ and perform stochastic gradient descent (SGD). In order to evaluate these derivatives we assume that derivatives of f can be computed with respect to its inputs. This assumption is made only in order to backpropagate errors from the loss to the parameters, but crucially is *not needed at test time*. If even at training time the derivatives of f are not available then it would be necessary to approximate these derivatives via an algorithm such as REINFORCE [Williams, 1992].

To this point we have made no assumptions about the distribution of training functions $p(f)$. In this work we are interested in learning general-purpose black-box optimizers, and in other words we desire our distribution to be quite broad. As a result we propose the use of Gaussian processes (GPs) as a suitable training distribution. The use of functions sampled from a GP prior also provides functions whose gradients can be easily evaluated and used at training time as noted above. Also the posterior expected improvement used within L_{EI} can be easily computed Močkus [1982] and differentiated as well. The major downside of search strategies which are based on GP inference is their cubic complexity at what we refer to as "test time". However in some sense the strategies learned in this work can be thought of as a distillation of GP-based techniques, and we will show later that the evaluation of our search strategy is negligible compared to that of standard Bayesian optimization techniques.

3 Experiments

We present several experiments that show the breadth of generalization that is achieved by our learned algorithms. We train our algorithm to optimize very simple functions—samples from a GP with a fixed length scale—and show that the learned algorithm is able to generalize from these simple functions to optimize a variety of other test functions that were not seen during training.

Our results are notable in several ways. First, we show that even though the learned algorithms are not competitive with GP-based Bayesian optimization (BO) when the training and test distributions match, our algorithm is competitive with GP-based methods with regards to generalization across several tasks. In addition to achieving competitive performance, our learned algorithms are significantly faster than GP-based methods, because they avoid several costs: inverting the GP covariance matrix, optimizing the acquisition function, and sampling the GP hyper-parameters.

We consider two different recurrent neural network architectures to implement our global optimization algorithms, specifically an LSTM architecture as well as a differentiable neural computer (DNC). We train each algorithm on trajectories of $T = 30$ steps, and update the algorithm parameters using BPTT to optimize the reward function averaged over the trajectory. We repeat this process for each of the reward functions discussed in Section 2 in order to compare these three objectives.

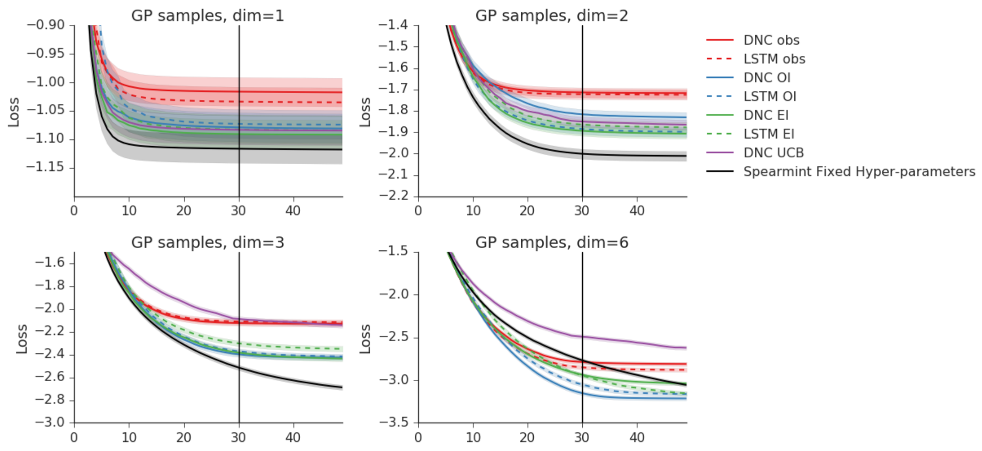

Figure 2: Average test loss with 95% confidence interval on functions sampled from the GP prior. The black vertical line indicates the number of searching steps the models are trained with, $T = 30$.

Hyperparameters for the algorithm (such as number of hidden units, and memory size for the DNC models) are found through grid search.

We compare our model with one of the most commonly used BO packages, Spearmint, and a random search baseline. We evaluate the performance at a given search step $t \leq 50$, according to the test loss function that is defined as the minimum observed function value up to step t, $\text{loss}(t) = \min_{i \leq t} f(\mathbf{x}_i)$. Throughout this set of experiments, as noted earlier, we train using two forms of RNN: the DNC and LSTM, coupled with the losses introduced in Section 2. E.g. in the following experiments, DNC OI refers to the DNC network trained using the observed improvement loss L_{OI}, LSTM EI refers to the LSTM trained using the expected improvement loss L_{EI}, etc. For the first set of experiments we also compare against training with a loss based on GP-UCB [Srinivas et al., 2010], which we refer to as DNC UCB.

3.1 Functions from a GP Prior

In the first set of experiments, we train our model on functions sampled from a GP prior with fixed hyper-parameters. Here for Spearmint we assume a prior distribution equivalent to the training distribution, hence this method knows the ground truth and thus provides a very competitive baseline.

3.1.1 Generalization to Functions Sampled from the Training Distribution

We first evaluate the generalization performance on functions sampled from the same training distribution. Notice, however, that those functions are never observed during training. Figure 2 shows the best observed function values as a function of search step t, averaged over $10,000$ sampled functions. As expected Spearmint proves to be the best model under most settings as it makes use of the ground truth of the prior distribution. When the input dimension is 6, however, neural network models outperform Spearmint and we suspect it is because in higher dimensional space, it global optimization is harder within the time frames at which we apply these algorithms. Among NN models those trained with expected/empirical improvement perform better than the model trained with function observations or GP-UCB.

Figure 3 shows the trajectories of $\mathbf{x}_t$ for different models on a typical function sample. All of the models do exploration initially, and later settle down around one mode and search more locally. The model trained with EI behaves most similarly to the trajectory of Spearmint. Models trained with observations tend to explore less than the other models and therefore easily miss the global modes, while those trained with the observed improvement keep exploring even in the later stage. We do not observe qualitative difference between the DNC and LSTM models trained with the same reward function.

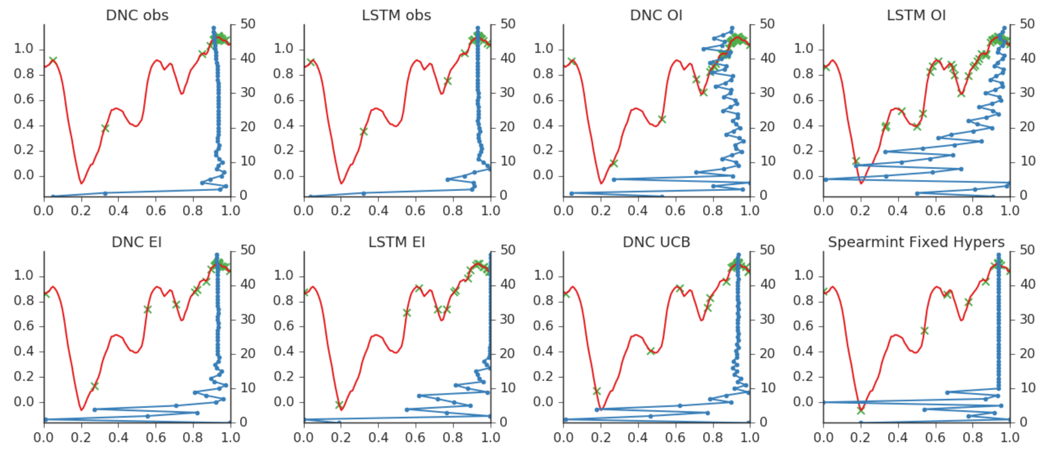

Figure 3: Search trajectories on a 1-dimensional function. Red line: function value vs input. Green cross: function value on query points. Blue line: search iteration vs query locations.

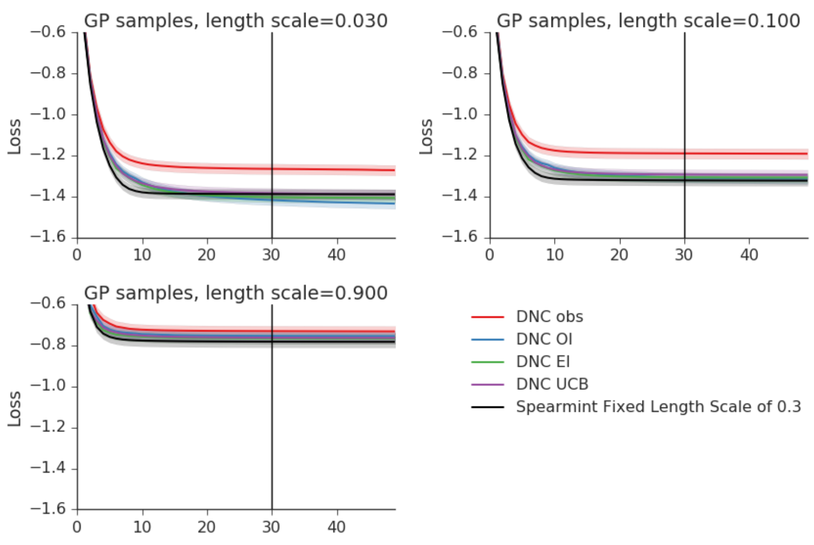

Figure 4: Loss on functions sampled from GP with a different length scale from the training value. LSTM models are not shown as they perform similarly to their DNC counterparts.

3.1.2 Generalization to Functions Sampled with a Different Length-Scale

We then assess how the learned policies generalize to functions that are not sampled from the training distribution. We generate test functions from a 1-dimensional GP prior with a different length-scale from the training length-scale. We compare the neural network models with Spearmint with a length scale fixed at the same value as the training distribution.

In Figure 4 shows the loss of DNC models for functions sampled with length scale $\ell \in \{0.03, 0.1, 0.9\}$ while the training length scale is 0.3. When $\ell < \ell_{\text{train}}$ functions vary more rapidly than the training examples and models assuming a more smooth function class may miss the global minimum between detected modes. We observe that the performance of RNN models changes in the same tendency of Spearmint except for the model trained with observations. For functions with very short length scales, 0.03, DNC OI starts to outperform Spearmint with fixed length scales, thanks to its more exploratory behavior.

3.1.3 Evaluating Transfer of Optimizer on Five Benchmark Functions

We compare the algorithms on five standard benchmark functions for black-box optimization with dimensions ranging from 1 to 6. To have a more robust evaluation of the performance of each model, we generate multiple instances for each benchmark function by applying a random translation ($-0.1 \sim 0.1$), scaling ($0.9 \sim 1.1$), flipping, and dimension permutation in the input domain.

Figure 5 shows the loss compared with a random baseline, Spearmint with length scale fixed to the training length scale, and Spearmint with automatic inference of the GP hyper-parameter values and input warping to deal with non-stationarity [Snoek et al., 2014]. All of the neural network models show competitive performance against Spearmint methods except on the 1D Gramacy & Lee function up to the training horizon, $T = 30$. The Gramacy & Lee function has a much smaller length scale than 0.3 and none of the models is able to find a good mode except Spearmint, which infers the length scale. This problem can be addressed by training models with several length scales sampled from a distribution, in which case the model should learn to infer the underlying length scale and take a good search strategy accordingly.

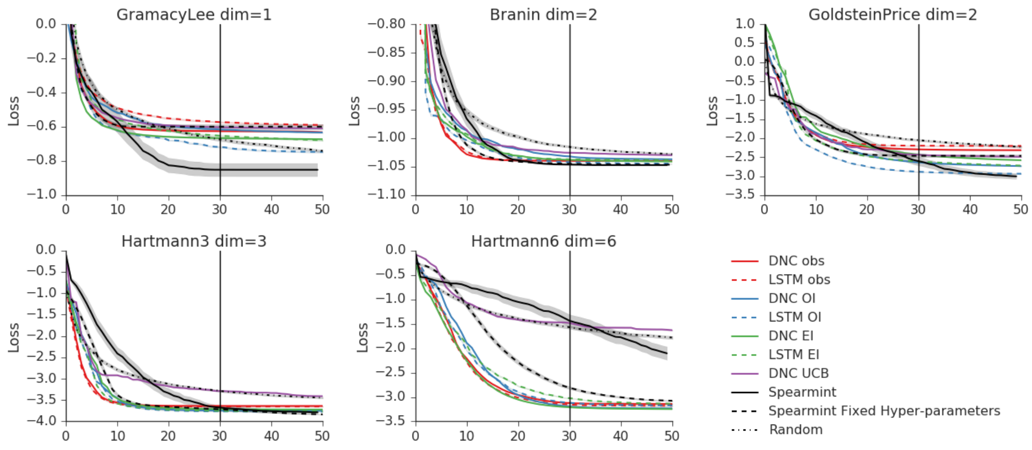

Figure 5: Loss on 5 benchmark functions (Gramacy and Lee, Branin, Goldstein price, Hartmann with dim=3, and Hartmann with dim=6) as a function of the number of steps.

The neural network optimizers run about 10^5 times faster than Spearmint with the LSTM architecture and about 10^4 times faster with the DNC architecture, with detailed numbers in Table 1. This is depicted clearly in Figure 6.

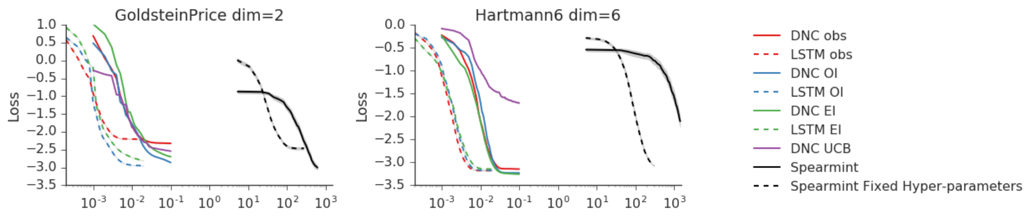

Figure 6: Loss as a function of the optimizer's run-time (seconds) on the 5 benchmark functions, illustrating the superiority in computational time of the RNNs over GP-based Bayesian optimization.

3.1.4 Evaluating Transfer of the Global Learner to a Simple RL Problem

We also consider an application to a simple reinforcement learning task described by Hoffman et al. [2009]. In this problem we simulate a physical system consisting of a number of repellers which affect the fall of particles through a 2D-space. The goal is to direct the path of the particles through high reward regions of the state space and maximize the accumulated discounted reward. The four-dimensional state-space in this problem consists of a particles position and velocity. The path of

Table 1: Run-time (seconds) for 50 iterations excluding the black-box function evaluation time.

	Spearmint	Spearmint-Fixed-Hyperparameters	DNC	LSTM
Gramacy & Lee	472	267	0.05	0.01
Branin	601	275	0.05	0.01
Goldstein Price	608	271	0.05	0.01
Hartmann 3	774	277	0.05	0.01
Hartmann 6	1422	302	0.05	0.01

the particles can be controlled by the placement of repellers which push the particles directly away with a force inversely proportional to their distance from the particle. At each time step the particles position and velocity are updated using simple deterministic physical forward simulation. The control policy for this problem consists of 3 learned parameters for each repeller: 2d location and the strength of the repeller.

In our experiments we consider a problem with 2 repellers, i.e. 6 parameters. An example trajectory along with the reward structure (contours) and repeller positions (circles) is displayed in Figure 7. We apply the same perturbation as in the previous subsection to study the average performance. The loss functions of all models are also shown in Figure 7.

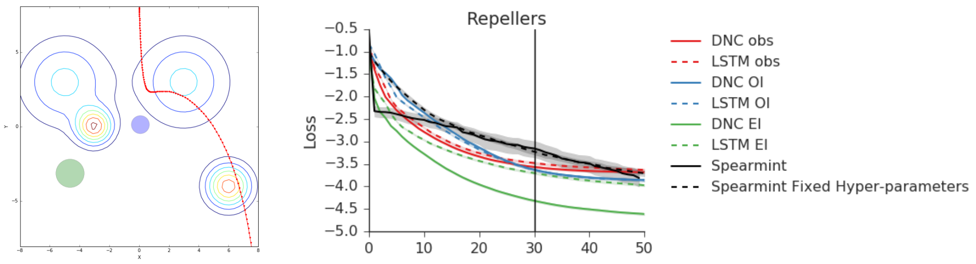

Figure 7: The left plot shows an example trajectory for the repellers problem, where solid circles show the position and strength of the two repellers and contour lines show the reward function. The right plot displays the results of each method on optimizing this problem.

4 Discussion and Future Work

The experiments have shown that up to the training horizon, $T = 30$, the learned RNN global optimizers either match or outperform both Spearmint with fixed hyper-parameters and Spearmint with inferred hyper-parameters and input warping.

Our RNN optimizer was trained on functions genereated using a GP with fixed hyper-parameters. The fact that up to $T = 30$ it can do well on a varied set of tasks illustrates a significant degree of transfer of the RNN learner.

The experiments have also shown that the RNNs are massively faster than Bayesian optimization. Hence, for applications involving a known horizon or where speed matters, we recommend the use of the RNN optimizers.

However, the RNN optimizers also appear to have shortcomings. Training for very long horizons is difficult. This issue was also documented recently in Duan et al. [2016]. It is also clear from Figure 5 (Hartmann6 function) and Figure 7 (control problem) that while the RNNs outperform Spearmint, with DNC doing slightly better than the LSTMs, for a horizon less than $T = 30$, beyond this training horizon Spearmint eventually achieves a lower error than the neural networks. This is because Spearmint has a mechanism for continued exploration that we have not yet incorporated into our neural network models. We are currently exploring extensions to our approach to improve exploration beyond the training horizon.

References

P. Agrawal, A. Nair, P. Abbeel, and J. Malik. Learning to poke by poking: Experiential learning of intuitive physics. In *Neural Information Processing Systems*, 2016.

S. Agrawal and N. Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, 2013.

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

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. Data-efficient learning of feedback policies from image pixels using deep dynamical models. *arXiv preprint arXiv:1510.02173*, 2015.

S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for ANNs. *Neural Processing Letters*, 2(4):26–30, 1995.

Y. Bengio, S. Bengio, and J. Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1990.

Y. Bengio, S. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *in Conference on Optimality in Biological and Artificial Networks*, 1992.

E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical Report UBC TR-2009-23 and arXiv:1012.2599v1, Dept. of Computer Science, University of British Columbia, 2009.

S. Bubeck, R. Munos, and G. Stoltz. Pure exploration in multi-armed bandits problems. In *International Conference on Algorithmic Learning Theory*, 2009.

O. Chapelle and L. Li. An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems*, pages 2249–2257, 2011.

Y. Duan, J. Schulman, X. Chen, P. Bartlett, I. Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. Technical report, UC Berkeley and OpenAI, 2016.

V. Gabillon, M. Ghavamzadeh, A. Lazaric, and S. Bubeck. Multi-bandit best arm identification. In *Advances in Neural Information Processing Systems*, pages 2222–2230, 2011.

V. Gabillon, M. Ghavamzadeh, and A. Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems*, pages 3212–3220, 2012.

A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-BarwiÅska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

H. F. Harlow. The formation of learning sets. *Psychological review*, 56(1):51, 1949.

P. Hennig and C. Schuler. Entropy search for information-efficient global optimization. *The Journal of Machine Learning Research*, 13(1):1809–1837, 2012.

J. M. Hernández-Lobato, M. A. Gelbart, M. W. Hoffman, R. P. Adams, and Z. Ghahramani. Predictive entropy search for Bayesian optimization with unknown constraints. In *International Conference on Machine Learning*, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

M. Hoffman, B. Shahriari, and N. de Freitas. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *AI and Statistics*, pages 365–374, 2014.

M. W. Hoffman, H. Kueck, N. de Freitas, and A. Doucet. New inference strategies for solving Markov decision processes using reversible jump MCMC. In *Uncertainty in Artificial Intelligence*, pages 223–231, 2009.

D. Jones. A taxonomy of global optimization methods based on response surfaces. *J. of Global Optimization*, 21(4):345–383, 2001.

D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black-box functions. *J. of Global optimization*, 13(4):455–492, 1998.

E. Kaufmann, O. Cappé, and A. Garivier. On bayesian upper confidence bounds for bandit problems. In *International Conference on Artificial Intelligence and Statistics*, pages 592–600, 2012.

E. J. Kehoe. A layered network model of associative learning: learning to learn and configuration. *Psychological review*, 95(4):411, 1988.

R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181, 2009.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 86(1):97–106, 1964.

T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. Technical report, DeepMind, 2016.

J. Močkus. The Bayesian approach to global optimization. In *Systems Modeling and Optimization*, volume 38, pages 473–481. Springer, 1982.

J. Močkus, V. Tiesis, and A. Žilinskas. The application of Bayesian methods for seeking the extremum. In L. Dixon and G. Szego, editors, *Toward Global Optimization*, volume 2. Elsevier, 1978.

D. K. Naik and R. Mammone. Meta-neural networks that learn by learning. In *International Joint Conference on Neural Networks*, volume 1, pages 437–442. IEEE, 1992.

T. P. Runarsson and M. T. Jonsson. Evolution and design of distributed learning rules. In *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 59–63. IEEE, 2000.

D. Russo and B. Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, 2016.

J. Schmidhuber. *Evolutionary Principles in Self-Referential Learning. On Learning how to Learn: The Meta-Meta-Meta...-Hook*. PhD thesis, Institut f. Informatik, Tech. Univ. Munich, 1987.

J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

J. Schmidhuber. A neural network that embeds its own meta-levels. In *International Conference on Neural Networks*, pages 407–412. IEEE, 1993.

N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *International Conference on Artificial Neural Networks*, volume 2, pages 569–574, 1999.

S. L. Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

B. Shahriari, Z. Wang, M. W. Hoffman, A. Bouchard-Côté, and N. de Freitas. An entropy search portfolio. In *NIPS workshop on Bayesian Optimization*, 2014.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

J. Snoek, K. Swersky, R. S. Zemel, and R. P. Adams. Input warping for Bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, 2014.

E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, pages 1015–1022, 2010.

R. S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Association for the Advancement of Artificial Intelligence*, pages 171–176, 1992.

K. Swersky, J. Snoek, and R. P. Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 2004–2012, 2013.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 1998.

J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *J. of Global Optimization*, 44(4):509–534, 2009.

Z. Wang and N. de Freitas. Theoretical analysis of Bayesian optimisation with unknown Gaussian process hyper-parameters. *arXiv preprint arXiv:1406.7758*, 2014.

Z. Wang, B. Shakibi, L. Jin, and N. de Freitas. Bayesian multi-scale optimistic optimization. In *AI and Statistics*, pages 1005–1014, 2014.

L. B. Ward. Reminiscence and rote learning. *Psychological Monographs*, 49(4):i, 1937.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. Technical report, submitted to ICLR 2017, 2016.

A Related work

The idea of applying machine learning techniques to optimize the process of learning itself has a long history [Schmidhuber, 1987, Naik and Mammone, 1992, Thrun and Pratt, 1998]. The techniques proposed for solving this *meta-learning* problem are also quite varied, in part because the lack of consensus on how to draw the line between *meta* learning and ordinary learning.

There is somewhat more consensus about the structure of ordinary learning. The target of learning is a *model* that has several *parameters*. The model interacts with an *algorithm* that ingests data and whose purpose is to choose values for the parameters of the model.

The *meta* part of meta learning arises when the target of learning is itself a learning algorithm. In meta learning an algorithm takes the place of the model in an ordinary learning problem, and the algorithm exposes parameters to a *meta algorithm*, whose role mirrors that of the algorithm in ordinary learning.

Frequently the meta algorithm is one that could also play the role of a non-meta algorithm in an ordinary learning problem. In fact the purpose of establishing the meta learning framework is to make this possible. Conceptually one could build increasingly high towers of meta-meta algorithms being tuned by meta-meta-meta algorithms, but in practice this is rarely done.

Different approaches to meta learning can be distinguished by how they handle the interface between model and algorithm in the underlying learning problem, and also by whether the product of meta learning is a model or an algorithm.

Several early works approached the problem of meta learning by coupling the model and algorithm into a single system [Sutton, 1992, Schraudolph, 1999]. The insight here is to observe that the process of learning in a static model can be viewed as the evolution of a dynamical system. Meta learning is achieved by applying an ordinary learning algorithm to the parameters of this dynamical system.

The coupled setting handles the boundary between model and algorithm by removing it. In many ways this is meta learning through force of interpretation alone, since an observer without prior knowledge of the decoupled state of model and algorithm could not distinguish between an ordinary learning in a dynamic model and a coupled meta learning in a static model.

However, since coupled meta learning reduces so readily to an ordinary learning problem it is often the most straightforward setting to work in. A modern technique in this setting is RL^2 [Duan et al., 2016], which lifts an ordinary reinforcement learning agent into the role of an algorithm by reinterpreting the dynamics of the environment. Once the model and algorithm are coupled in this way they are able to apply an ordinary reinforcement learning algorithm to set the parameters of the fused agent-algorithm without further work. The collapse of model and algorithm can be more readily understood by contrasting their work with Mirowski et al. [2016], who attack similar tasks without the meta learning gloss.

Early works that take this approach are Schmidhuber [1992] and Schmidhuber [1993]—building on even earlier work of Schmidhuber [1987]—which consider networks that are able to modify their own weights. This leads to end-to-end differentable systems which allow, in principle, for extremely general update strategies to be learned.

Instead of coupling the model and algorithm the boundary between them can be maintained. Maintaining the boundary allows the model and algorithm to be decoupled post-learning, and maintains a clear separation of concerns between them. Since the model and algorithm decouple in this setting it becomes useful to ask about the product of a meta learning procedure. Throughout the course of meta learning both models and algorithms are trained, but typically only one of them is directly of interest, while the other plays a subordinate role as a tool for realizing the primary target.

Bayesian hyper-parameter optimization is a good example of meta learning where it is the model which is of primary interest. Fitting a response surface used to predict the performance of an underlying model and using that surface to guide selection of different model configurations fits comfortably within the meta learning framework, but the response surface itself is typically not the focus. The product of Bayesian hyper-parameter optimization is a configuration for the model that achieves good performance on the target task.

Zoph and Le [2016] present a "neuralization" of this pattern, carefully engineered for the problem of designing neural network architectures. Their setup trains a controller RNN to produce a string in a custom domain specific language (DSL) for describing neural network architectures. An architecture matching the produced configuration (the "child" network) is instantiated and trained, and its performance is measured. Although this setup avoids building an explicit model (as is done in Gaussian process based Bayesian optimization), we can view the string produced by the controller RNN as playing the same role as the proposal obtained by maximizing the acquisition function in a response surface framework. Representing the query strategy directly allows them avoid the prickly problem of specifying a geometry for strings in their DSL, but makes it more difficult to re-use the information acquired at the meta-level for future tasks.

A series of papers from Bengio et al. [1990, 1992, 1995] present methods for local update rules that avoid back-propogation by using simple parametric rules, and Runarsson and Jonsson [2000] extend this to more complex update models. The result of meta learning is in these cases an algorithm (i.e. the local rule), and the models are used simply as a vehicle for learning the rules. The resulting rules can be applied to new models after meta learning.

Andrychowicz et al. [2016] also learn update rules for optimization, but in a way that is more integrated with the standard optimization framework. Rather than trying to distil a global objective into a local rule, this work focuses on learning how to integrate gradient observations over time in order to achieve fast learning of the model. The resulting algorithm can be applied to new optimization problems, and the component-wise structure of the algorithm allows a single learned algorithm to be applied to new problems of different dimensionality.

A third angle for characterizing meta learning approaches is to look at where information about the learning process is accumulated. This is different than asking about the product of meta learning, which can be seen by examining the standard setup for Bayesian hyper parameter optimization. The product of meta learning in that case is the model, as discussed above, but the information about the learning process itself is encoded in the response surface.

Understanding where information about learning is accumulated allows us to identify where we might expect to see transfer of information between problems at the meta-level. Decoupled approaches that accumulate meta-information into the algorithm can transfer this knowledge trivially by simply applying the algorithm to a new problem. Methods that pack this information into a query policy do not have any straightforward avenue to transfer in this way.

Methods that use an auxiliary model to represent the learning process have the most potential for transfer, but may require additional work to extract the appropriate information from this model [Swersky et al., 2013]. This perspective also allows us to draw a connection between meta learning and model based active learning, where a model of the environment is used to guide an active process for collecting data to improve that model [Assael et al., 2015, Agrawal et al., 2016].