Learning to Optimize

Ke Li **Jitendra Malik**
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720
United States
{ke.li,malik}@eecs.berkeley.edu

Abstract

Algorithm design is a laborious process and often requires many iterations of ideation and validation. In this paper, we explore automating algorithm design and present a method to *learn* an optimization algorithm, which we believe to be the first method that can automatically discover an algorithm that is better than known existing algorithms. We approach this problem from a reinforcement learning perspective and represent any particular optimization algorithm as a policy. We learn an optimization algorithm using guided policy search and demonstrate that the resulting algorithm outperforms existing hand-engineered algorithms in terms of convergence speed and/or the final objective value.

1 Introduction

Continuous optimization algorithms are some of the most ubiquitous tools used in virtually all areas of science and engineering. Indeed, they are the workhorse of machine learning and power most learning algorithms. Consequently, optimization difficulties become learning challenges – because their causes are often not well understood, they are one of the most vexing issues that arise in practice. One solution is to design better optimization algorithms that are immune to these failure cases. This requires careful analysis of existing optimization algorithms and clever solutions to overcome their weaknesses; thus, doing so is both laborious and time-consuming. Is there a better way? If the mantra of machine learning is to learn what is traditionally manually designed, why not take it a step further and *learn* the optimization algorithm itself?

Consider the general structure of an algorithm for unconstrained continuous optimization, which is outlined in Algorithm 1. Starting from a random location in the domain of the objective function, the algorithm iteratively updates the current location by a step vector Δx computed from some functional π of the objective function, the current location and past locations.

Different optimization algorithms only differ in the choice of the update formula π. Hence, if we can learn π, we will be able to learn an optimization algorithm. We can model π simply as a function from the objective values and gradients evaluated at current and past iterates to the next step vector. If we represent π using a universal function approximator like a neural net, it is then possible to search over the space of optimization algorithms by learning the parameters of the neural net. We formulate this as a reinforcement learning problem, where any particular optimization algorithm simply corresponds to a policy. Learning an optimization algorithm then reduces to finding an optimal policy. For this purpose, we use an off-the-shelf reinforcement learning algorithm known as guided policy search [17], which has demonstrated success in a variety of robotic control settings [18, 10, 19, 13].

Some details have been omitted from this version due to space constraints. We refer interested readers to https://arxiv.org/abs/1606.01885 for the complete details.

The work was presented at the NIPS workshop on Optimizing the Optimizers. Barcelona, Spain, 2016.

Algorithm 1 General structure of unconstrained optimization algorithms

Require: Objective function f
  $x^{(0)} \leftarrow$ random point in the domain of f
  for $i = 1, 2, \ldots$ **do**
    $\Delta x \leftarrow \pi(f, \{x^{(0)}, \ldots, x^{(i-1)}\})$
    if stopping condition is met **then**
      return $x^{(i-1)}$
    end if
    $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
  end for

Our goal is to learn about regularities in the geometry of the error surface induced by a class of objective functions of interest and exploit this knowledge to optimize the class of objective functions faster. This is potentially advantageous, since the learned optimizer is trained on the actual objective functions that arise in practice, whereas hand-engineered optimizers are often analyzed in the convex setting and applied to the non-convex setting.

2 Related Work

When the objective functions correspond to loss functions for training a model, learning the optimization algorithm can be viewed as learning how to learn. This theme of learning about the learning process itself has been explored and is referred to as "learning to learn" or "meta-learning" [2, 24, 6, 23]. Because there is no consensus on what kinds of meta-knowledge should be learned at the meta-level, meta-learning methods differ in their objectives and the settings under which they operate. One line of work [1] aims to learn about commonalities across a family of related tasks, which is now better known as transfer learning and multi-task learning. A different line of work [7, 22] aims to learn how to select the base-level learner that achieves the best performance for a given task. Under this setting, meta-knowledge takes the form of correlations between properties of tasks and the performance of different base-level learners trained on them. Unlike these approaches, the proposed method learns regularities in the optimization/learning process itself, rather than regularities shared by different tasks or regularities in the mapping between tasks and best-performing base-level learners.

The line of work on program induction [8, 15, 20, 12] considers the problem of learning programs from examples of input and output. In general, the algorithms learned using these methods have not been able to match the performance of simple hand-engineered algorithms. In contrast, our aim is learn an algorithm that is better than known hand-engineered algorithms.

There is also a large body of work on hyperparameter optimization [16, 4, 3], which studies the optimization of hyperparameters used to train a model. When presented with a new objective function at test time, these methods need to conduct multiple trials with different hyperparameter settings. Like the proposed method, online hyperparameter adaption methods [5, 21, 14, 9, 11] follow a single trajectory at test time. However, unlike these methods, the proposed method can search over a larger space of possible optimization algorithms rather than different hyperparameter values.

3 Background on Reinforcement Learning

In the reinforcement learning setting, the learner is given a choice of actions to take in each time step, which changes the state of the environment in an unknown fashion, and receives feedback based on the consequence of the action. The feedback is typically given in the form of a reward or cost, and the objective of the learner is to choose a sequence of actions based on observations of the current environment that maximizes cumulative reward or minimizes cumulative cost over all time steps.

More formally, a finite-horizon reinforcement learning problem can be characterized by the tuple $(\mathcal{S}, \mathcal{A}, p_0, p, c)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p_0 : \mathcal{S} \to \mathbb{R}^+$ is the probability density over initial states, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ is the transition probability density, that is, the conditional probability density over successor states given the current state and action, $c : \mathcal{S} \to \mathbb{R}$ is a function that maps state to cost. A policy $\pi : \mathcal{S} \times \mathcal{A} \times \{0, \ldots, T-1\} \to \mathbb{R}^+$ is a conditional probability density over actions given the state at each time step. The objective is to learn a policy π_t^*

such that the expected cumulative cost is minimized. That is,

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{s_0,a_0,s_1,\ldots,s_T}\left[\sum_{t=0}^{T} c(s_t)\right],$$

where the expectation is taken with respect to the joint distribution over the sequence of states and actions, often referred to as a trajectory, which has the density

$$q\left(s_0, a_0, s_1, \ldots, s_T\right) = p_0\left(s_0\right) \prod_{t=0}^{T-1} \pi\left(a_t \middle| s_t, t\right) p\left(s_{t+1} \middle| s_t, a_t\right).$$

When a policy is identical for all time steps, it is referred to as stationary.

This problem of finding the cost-minimizing policy is known as the policy search problem. To enable generalization to unseen states, the policy is typically parameterized and minimization is performed over representable policies. Solving this problem exactly is intractable in all but selected special cases. Therefore, policy search methods generally tackle this problem by solving it approximately.

We use a policy search method known as guided policy search [17], which can search over expressive non-linear policy classes in continuous state and action spaces. It works by alternating between computing a mixture of target trajectories and training the policy to replicate them. Successive iterations locally improve target trajectories while ensuring proximity to behaviours that are reproducible by the policy. Target trajectories are computed by fitting local approximations to the cost and transition probability density and optimizing over a restricted class of time-varying linear target policies subject to a trust region constraint. The stationary non-linear policy is trained to minimize the squared Mahalanobis distance between the predicted and target actions at each time step.

4 Formulation

We observe that the execution of an optimization algorithm can be viewed as the execution of a particular policy in an MDP: the state consists of the current location and the objective values and gradients evaluated at the current and past locations, the action is the step vector that is used to update the current location, and the transition probability is partially characterized by the location update formula, $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$. The policy that is executed corresponds precisely to the choice of π used by the optimization algorithm. For this reason, we will also use π to denote the policy at hand. Under this formulation, searching over policies corresponds to searching over possible first-order optimization algorithms.

To learn π, we need to define the cost function, which should penalize policies that exhibit undesirable behaviours during their execution. Since the performance metric of interest for optimization algorithms is the speed of convergence, the cost function should penalize policies that converge slowly. To this end, assuming the goal is to minimize the objective function, we define cost at a state to be the objective value at the current location, thereby encouraging the policy to reach the minimum as quickly as possible. We parameterize the mean of π using a neural net.

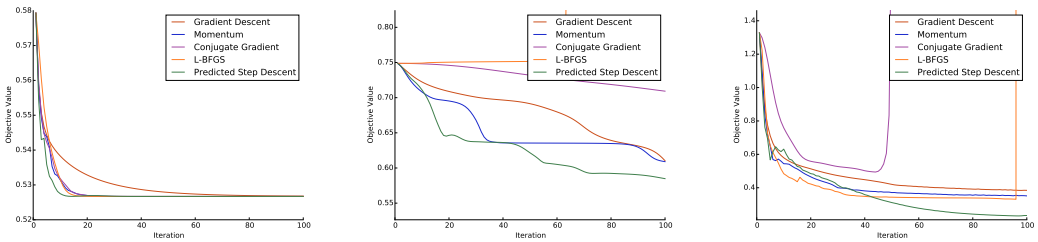
5 Experiments

We learn optimization algorithms for various convex and non-convex classes of objective functions that correspond to loss functions for different machine learning models. Specifically, we learn algorithms for the following optimization problems:

- *Logistic Regression:*

$$\min_{\mathbf{w},b} -\frac{1}{n}\sum_{i=1}^{n} y_i \log\sigma\left(\mathbf{w}^T\mathbf{x}_i + b\right) + \left(1 - y_i\right)\log\left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right) + \frac{\lambda}{2}\left\|\mathbf{w}\right\|_2^2,$$

  where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ denote the weight vector and bias respectively, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$ denote the feature vector and label of the i^{th} instance, λ denotes the coefficient on the regularizer and $\sigma(z) \coloneqq \frac{1}{1+e^{-z}}$. For our experiments, we choose $\lambda = 0.0005$ and $d = 3$. This objective is convex in $\mathbf{w}$ and b.



(a) (b) (c)

Figure 1: Objective values achieved by different algorithms on (a) logistic regression (b) robust linear regression and (c) neural net classification objectives from the test set. Lower objective values indicate better performance. Best viewed in colour.

- *Robust Linear Regression:*

$$\min_{\mathbf{w},b} \frac{1}{n} \sum_{i=1}^{n} \frac{\left(y_i - \mathbf{w}^T \mathbf{x}_i - b\right)^2}{c^2 + \left(y_i - \mathbf{w}^T \mathbf{x}_i - b\right)^2},$$

  where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ denote the weight vector and bias respectively, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ denote the feature vector and label of the i^{th} instance and $c \in \mathbb{R}$ is a constant that modulates the shape of the loss function. For our experiments, we use $c = 1$ and $d = 3$. This loss function is not convex in either $\mathbf{w}$ or b.

- *Neural Net Classifier:*

$$\min_{W,U,\mathbf{b},\mathbf{c}} -\frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{\exp\left(\left(U \max\left(W\mathbf{x}_i + \mathbf{b}, 0 \right) + \mathbf{c} \right)_{y_i} \right)}{\sum_j \exp\left(\left(U \max\left(W\mathbf{x}_i + \mathbf{b}, 0 \right) + \mathbf{c} \right)_j \right)} \right) + \frac{\lambda}{2} \|W\|_F^2 + \frac{\lambda}{2} \|U\|_F^2,$$

  where $W \in \mathbb{R}^{h \times d}, b \in \mathbb{R}^h, U \in \mathbb{R}^{p \times h}, c \in \mathbb{R}^p$ denote the first-layer and second-layer weights and biases, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, \ldots, p\}$ denote the input and target class label of the i^{th} instance, λ denotes the coefficient on regularizers and $(\mathbf{v})_j$ denotes the j^{th} component of $\mathbf{v}$. For our experiments, we use $\lambda = 0.0005$ and $d = h = p = 2$. The error surface is known to have complex geometry and multiple local optima, making this a challenging optimization problem.

In Figure 1, we plot performance of the learned optimization algorithm (labelled as "predicted step descent") and that of existing algorithms on objective functions from the test set. As shown, the learned optimizer outperforms gradient descent, momentum, conjugate gradient and L-BFGS on non-convex objectives in terms of convergence speed and final objective values. The gaps for the non-convex objectives are also larger than that for the convex objective, suggesting that hand-engineered algorithms are more sub-optimal on challenging optimization problems and so the potential for improvement from learning the algorithm is greater in such settings.

6 Conclusion

We presented a method for learning a better optimization algorithm. We formulated this as a reinforcement learning problem, in which any particular optimization algorithm can be represented as a policy. Learning an optimization algorithm then reduces to find the optimal policy. We used guided policy search for this purpose and trained optimizers for different classes of convex and non-convex objective functions. We demonstrated that the learned optimizer converges faster and/or reaches better optima than hand-engineered optimizers. We hope optimizers learned using the proposed approach can be used to solve various common classes of optimization problems more quickly and help accelerate the pace of research in science and engineering.

References

[1] Yaser S Abu-Mostafa. A method for learning from hints. In *Advances in Neural Information Processing Systems*, pages 73–80, 1993.

[2] Jonathan Baxter, Rich Caruana, Tom Mitchell, Lorien Y Pratt, Daniel L Silver, and Sebastian Thrun. NIPS 1995 workshop on learning to learn: Knowledge consolidation and transfer in inductive systems. `https://web.archive.org/web/20000618135816/http://www.cs.cmu.edu/afs/cs.cmu.edu/user/caruana/pub/transfer.html`, 1995. Accessed: 2015-12-05.

[3] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[4] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[5] Matthieu Bray, Esther Koller-meier, Pascal Müller, Luc Van Gool, and Nicol N Schraudolph. 3d hand tracking by rapid stochastic gradient descent using a skinning model. In *1st European Conference on Visual Media Production (CVMP)*. Citeseer, 2004.

[6] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: applications to data mining*. Springer Science & Business Media, 2008.

[7] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[8] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 183–187, 1985.

[9] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning step size controllers for robust neural network training. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[10] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. *arXiv preprint arXiv:1509.06113*, 2015.

[11] Jie Fu, Zichuan Lin, Miao Liu, Nicholas Leonard, Jiashi Feng, and Tat-Seng Chua. Deep q-networks for accelerating the training of deep neural networks. *arXiv preprint arXiv:1606.01467*, 2016.

[12] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[13] Weiqiao Han, Sergey Levine, and Pieter Abbeel. Learning compound multi-step controllers under unknown dynamics. In *International Conference on Intelligent Robots and Systems*, 2015.

[14] Samantha Hansen. Using deep q-learning to control optimization hyperparameters. *arXiv preprint arXiv:1602.04062*, 2016.

[15] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[16] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[17] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[19] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *arXiv preprint arXiv:1501.05611*, 2015.

[20] Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical Bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 639–646, 2010.

[21] Paul L Ruvolo, Ian Fasel, and Javier R Movellan. Optimization on a budget: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*, pages 1385–1392, 2009.

[22] Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004.

[23] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[24] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.