IMPLEMENTING "LEARNING TO OPIMIZE" - FINAL REPORT

Stewart Slocum
sslocum3@jhu.edu

Siyuan Du
sdu11@jhu.edu

December 10, 2020

ABSTRACT

Training machine learning models is computationally expensive and tricky, and the development of new, improved, training algorithms is laborious and difficult. The 2016 paper *Learning to Optimize* provides a framework for learning new optimization algorithms from a reinforcement learning perspective, representing an algorithm as a policy [1]. It's almost the end 2020 now and there is still no available open-source implementation of the paper. We provide an open-source implementation of this paper (at https://github.com/stewy33/Learning-to-Optimize) and extend its results with modern policy search methods (A2C, PPO, etc). We were able to match performance on two out of the three benchmarks from the paper, approaching or outperforming tuned hand-engineered algorithms in terms of convergence speed and/or final objective value. Siyuan Du contributed to the Introduction and Related work sections of the paper, and investigations into GPS and alternative policy search algorithms as well as a few experiments. Stewart Slocum contributed the RL environment, the creation of various objective functions, and the results and conclusion sections of the paper. Slides at https://slides.com/stewyslocum/deck-295006.

1 Introduction

Machine learning is data-driven and learning patterns automatically from data is often more powerful than meticulously crafting rules by hand. However, the tools that power machine learning itself are still hand-crafted. An important example is that of optimization algorithms. The optimization algorithm is like a black box that takes the inputs like the model we designed and the data that we collected, and outputs the optimal model parameters.

Nonlinear optimization is an important problem in many domains, from control to machine learning. It is a very general framework: the goal is simply to minimize or maximize some function, which could be nonlinear and arbitrarily complex.

$$\min_{x \in \mathbb{R}^n} f(x)$$

Specifically, the nonlinear optimization problem of training large machine learning models is time-consuming and difficult, and is one of the major obstacles towards further progress.

Standard algorithms for training large machine learning models use gradient information to move towards better solutions (SGD, Nesterov's accelerated gradient method, Adam, etc.). But these standard convex optimization algorithms don't take the unique problem structure of optimizing non-convex neural networks into account. So, our approach is to learn a better optimizer.

In *Learning to Optimize*, instead of using a hand engineered optimization algorithm, one is learned instead. It could potentially learn to recognize common pitfalls during the optimization process such as oscillating behavior or stopping at saddle points. We provide an open-source implementation of the paper and extend its results with modern policy search methods like A2C, PPO, etc. The results we got in terms of convergence speed and/or final objective value match the results from the paper.

2 Related Work

Our work is based on the 2016 paper *Learning to Optimize*, where the authors explored learning an optimization algorithm for training logistic regression, linear regression, and shallow neural nets. They developed an extension that was suited to learning optimization algorithms in this setting, and showed that the learned optimization algorithm outperforms other known optimization algorithms.

In the special case where objective functions that the optimization algorithm is trained on are loss functions for training other models, this method can be considered a way of "learning to learn". To place this method in context, consider two other proposed "learning to learn" frameworks.

2.1 Learning What to Learn

This category of methods (Thrun Pratt, 2012) [2] mainly learn what parameter values of the base-level learner are useful across a family of related tasks. This learning to learn framework leverages commonalities in the family of tasks to make learning on a new task from the family quicker and easier. Traditional single-model multi-objective learning fits into this category.

2.2 Learning Which Model to Learn

This category of methods (Brazdil et al., 2008) [3] mainly learn which base-level learner (out of a set of potential models) achieves the best performance on a task. This learning to learn framework takes the correlations between different tasks and the performance of different base-level learners on those tasks to find a model that works well on a new task.

2.3 Learning How to Learn

This category of methods learn a good algorithm for training a base-level learner. The goal is to focus on the process of learning instead the outcome of learning (unlike the other two methods). This learning to learn framework learns commonalities in the behaviours of learning algorithms (optimizers) that achieve good performance to construct a new and better optimizer.

3 Problem Formulation

Consider the general structure of an algorithm for unconstrained continuous optimization, outlined in Algorithm 1. Starting from a random $x^{(0)}$, the algorithm iteratively updates the current location by a step vector computed from some functional π of the objective function, the current location, and past locations.

Algorithm 1 General structure of 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

This general structure includes standard algorithms such as gradient descent:

$$\pi(f, x^{(0)}, ..., x^{(i1)}) = \alpha \nabla f(x^{(i1)})$$

where α denotes the learning rate, and momentum:

$$\pi(f, x^{(0)}, ..., x^{(i1)}) = \alpha \sum_{j=0}^{i-1} \beta^{i-1-j} \nabla f(x^{(j)})$$

where β is the momentum constant.

So by choosing different policies π, we define different optimization algorithms.

Now a reinforcement learning problem is typically formulated as a Markov Decision Process (MDP). In our case, we consider an episodic MDP with continuous state an actions spaces defined by the 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$:

- State $\mathcal{S}$: Since we are specifically interested in learning optimization algorithms π that use first-order information, define the state s to be the history of gradients $\{\nabla f(x^{(i)}), \nabla f(x^{(i-1)}), ..., \nabla f(x^{(i-H)})\}$ and objective value improvements $\{f(x^{(i)}) - f(x^{(i-1)}), f(x^{(i)}) - f(x^{(i-2)}), ..., f(x^{(i)}) - f(x^{(i-H)})\}$ over the past H iterations.
- Action $\mathcal{A}$: The step vector Δx used to update the current iterate $x^{(i)} = x^{(i-1)} + \Delta x$.
- Reward r: The negative value of the objective function $r(s) = -f(x)$ that we are trying to minimize.

It is this framework that we use to learn our autonomous optimizer. We limit episodes to 40 timesteps in order to keep the problem challenging and to study the convergence of methods under few iterations.

4 Contributions

4.1 Implementation Details

We tested the performance of our approach on three types of objective functions included in the original paper: training logistic regression, robust linear regression (uses a Geman-McClure estimator), and a simple 2-layer multilayer perceptron neural network. As a simple baseline and to better understand behavior, we also examined performance on a fourth type of objective function: random convex quadratics.

We defined the optimization environment using the OpenAI gym [4] interface and trained policies with state-of-the-art reinforcement learning algorithms (PPO, A2C) implemented in the stable-baselines3 package [5].

Unfortunately, in the given time-frame, we were unable to additionally implement the reinforcement learning algorithm used in the paper called Guided Policy Search (GPS) [6]. There is no existing open-source implementation of model-free GPS [7] available, and there has been comparatively little discussion around the method since its introduction in 2013.

We compare the new method against several standard machine learning optimization algorithms: standard gradient descent, gradient descent with momentum, Adam, and LFBGS. We hand-tune learning rate and momentum hyperparameters (when applicable). We found that careful per-problem hand-tuning led to better results for the hand-engineering algorithms than Bayesian optimization on a train set of objectives.

Across all experiments, we found that PPO with a small importance sampling history window (1 or 2 updates) with a batch size of 32-64 consistently performed best.

4.2 Results

4.2.1 Convex Quadratics

We train our algorithm on a dataset of 90 convex quadratic functions of the form $f(x) = \frac{1}{2}x^T Ax + b^T x$. We randomly generate two dimensional quadratics with eigenvalues between 1 to 5 (condition number usually around 3). Besides providing a simple and easy benchmark to test our approach on, we added this objective function to our study because the simplicity of the convex quadratic model allows for interpretable trajectory plots (see Figure 1b at the end of the report).

We test our algorithm along with other standard optimization algorithms on a new convex quadratic function that was not present in the training set. While being a bit unstable once it approaches the minimum, the autonomous optimizer reaches the minimum very quickly, faster than accelerated methods like Adam and SGD with momentum (Figure 1a). Figure 1b shows that this is a result of the large initial step sizes taken by the algorithm. The promise of an intelligent optimizer is that it can learn when such large stepsizes are acceptable, and when they need to be shortened (if in a region of high curvature or near the minimum).

4.2.2 Logistic Regression

We train our algorithm on a dataset of 90 logistic regression problems with L2-regularization, a convex objective function. For each logistic regression problem, we generate 100 data points. For each problem we construct two

3-variate Gaussians with random mean and covariance. We draw 50 points each from the two Gaussians and give them labels according to which Gaussian they were drawn from. The logistic regression problem is to classify these correctly.

Again, we observe that the autonomous optimizer performs well, and converges extremely quickly (Figure 1c). Also as before, it remains noisy near the optimum. However, taking a look at the accuracy plot (a metric not relevant to optimization but very relevant to machine learning - Figure 1d), we find that after the initial convergence of the autonomous optimizer, there are only negligible gains in performance amongst all methods. It seems that this fast initial convergence may be enough for most machine learning applications. And if not, a hybrid approach (of a learned optimizer at first, and a traditional one near the minimum) may be a very effective approach.

4.2.3 Robust Linear Regression

We train our algorithm on a dataset of 120 robust linear regression problems. These problems are defined by the loss function:

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

. Here, we let $c = 1$. This is a specific instance of a Geman-McClure estimator, which is less sensitive to outlier data points. This loss function is nonconvex and more difficult to optimize. We generate 100 datapoints from four Gaussians. Each Gaussian is assigned a random vector. We then take all 25 datapoints from each Gaussian, and project them along the same random vector associated with that Gaussian. We then add i.i.d. standard normal noise.

The autonomous optimizer still performs relatively well, converging quickly, although diverging slightly later as on other problems (Figure 1e).

4.2.4 Training a 2-layer Multilayer Perceptron

We train our algorithm on a dataset of 80 MLP binary classification problems. As in the robust linear regression case, we generate four bivariate Gaussians with random mean and covariance. We randomly assign a 0-1 label to each Gaussian, but not all can have the same label. We draw 25 points from each Gaussian and give them the label associated with that Gaussian. We use standard binary cross entropy loss.

This was by far the most difficult optimization problem. Several methods struggled, including the autonomous optimizer, although over the course of the project, we were able to significantly improve performance (from completely diverging, to no improvement, to consistent monotonic improvement, even if it settles into local optima). We think that with further time, we could match the performance of the paper on this benchmark.

5 Conclusion

In this work, we provide an open source implementation of the *Learning to Optimize* paper and made two new contributions. First, besides logistic regression, robust linear regression, and a on 2-layer MLP, we studied performance on a new objective of convex quadratic functions, and created trajectory plots to get a deeper understanding of the qualitative behavior of the algorithm. Our second new contribution was to test with two new state-of-the-art reinforcement learning algorithms (A2C and PPO) that didn't exist in 2016 when *Learning to Optimize* was published. With these, we were able to match performance on two out of the three benchmarks from the original paper and get decent performance on the third. The little time we spent tuning hyperparameters already led to huge gains in performance, and surely with more time and compute resources, performance would continue to improve.

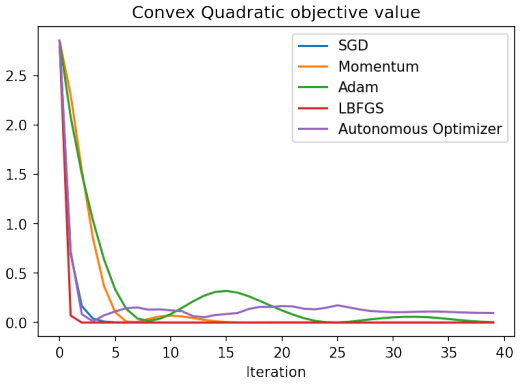
Future directions include implementing the paper *Learning to Optimize Neural Networks* [8] from the same authors, which followed previous work but extended its application to larger scale neural networks and problems like MNIST and CIFAR. Additionally, it would likely be profitable to (even analytically) examine similarities between learned optimizers and hand-engineered ones through saliency maps and other methods to interpret the behavior of the neural network.

References

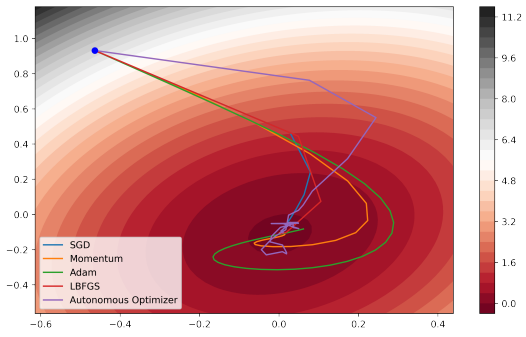
[1] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

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

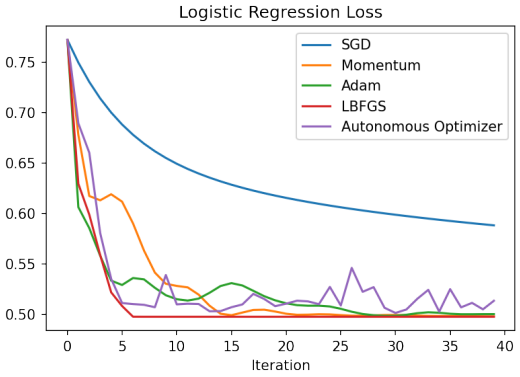
[3] Soares Carlos Brazdil, Pavel B and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 2003.



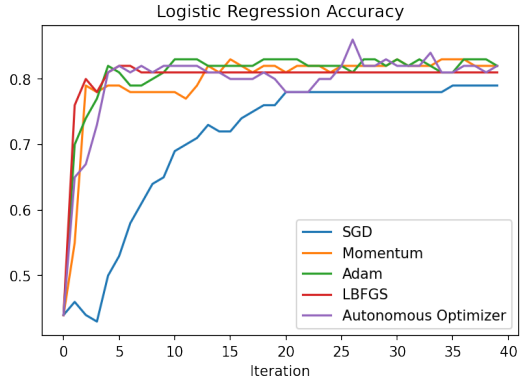
(a)



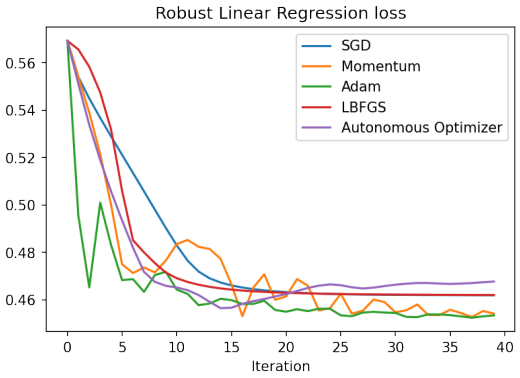
(b)



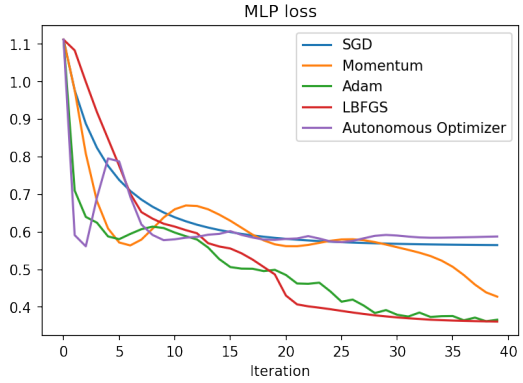
(c)



(d)



(e)



(f)

1a: Objective value with respect to iteration for several standard optimization algorithms as well as the autonomous optimizer. Convex quadratic minimization is a simple and easy baseline and all methods do well, although only LBFGS (an expensive second order method) converges faster than the learned optimizer.
1b: This plot overlays the trajectories of the benchmarked algorithms on top of a contour plot of the convex quadratic function. It is notable that the learned optimizer takes much larger steps than most of the other optimizers initially, allowing it to converge quickly. As it gets closer to the minimum, stepsize decreases.
1c: Objective value with respect to iteration for logistic regression. As in with convex quadratics, the learned optimizer converges extremely quickly, but then is somewhat noisy very close to the minimum.
1d: Logistic regression accuracy with respect to iteration.
1e: Objective value vs iteration for robust linear regression. This nonconvex optimization problem is more difficult for most of the methods, but the autonomous optimizer still converges faster than every method but Adam. After reaching this minimum, it begins to diverge.
1f: Objective value vs iteration for two-layer MLP training. This was the most difficult problem in the set, and the autonomous optimizer gets stuck like SGD. We believe that we could improve performance with more time to do hyperparameter tuning and study the reasons for observed behavior.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[6] Sergey Levine and Vladlen Koltun. Guided policy search. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1–9, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[7] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 1071–1079. Curran Associates, Inc., 2014.

[8] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.