More Efficient Stochastic Hyperparameter Optimization

Alex Adam, Jonathan Lorraine
Department of Computer Science
University of Toronto
Vector Institute
alex.adam@mail.utoronto.ca, lorraine@cs.toronto.edu

Abstract

Stochastic hyperparameter optimization is a general formulation which includes problems like finding neural network architectures. Current hyperparameter optimization procedures often involve nesting training of the elementary parameters of our network, with our hyperparameters. This allows the approximation of gradient updates for the hyperparameters, but is often has a prohibitive computational cost. We seek to reduce the cost by (1) using reinforcement learning to simultaneously learn a controller for our optimizer and architecture, and (2) including a gradient term missing in most formulations.

1 Introduction

Selecting the hyperparameters of neural networks is often more of an art than a science. These include choices (among many others) for the network architecture, the optimizer, the loss function, or how to augment the data. The recent work of Efficient Neural Architecture Search (ENAS) [Pham et al., 2018] has seen success learning network architectures by training a controller to explore shared weights. This was computationally expensive, with a variety of routes for potential improvement.

Proper tuning and scheduling of factors such as weight decay, learning rate, and weight initialization can determine if learning is possible. Jointly optimizing these parameters with the architecture may allow us to more quickly find solutions from a broader set of candidate models.

Additionally, there have been recent advances in gradient based hyperparameter optimization. These methods often involve computing a gradient of how the optimal elementary network parameters vary as the hyperparameters vary - denoted the response gradient. This term can be important when the validation and training loss want to update in different directions, as is the case with preventing overfitting.

We have two primary contributions. (1) We propose to jointly find optimal combinations of model architecture and optimization hyperparameters using a policy gradient approach where a policy network outputs both architectural and optimizer hyperparameters. This addresses the sub-optimality in approaches like ENAS, which neglect to adapt the optimizer along with the model hyperparameters. (2) We combine methods for computing response gradients with ENAS. This addresses the potential sub-optimality in approaches like ENAS, which treat the response gradient as 0.

2 Related Work

The field of automatic architecture design was revolutionized by Neural Architecture Search [Zoph and Le Google Brain] who decided to use reinforcement learning to probabilistically propose new architectures intended to maximize performance on a validation set. This was done using an RNN

Preprint. Work in progress.

controller that sequentially proposes hyperparameter settings that determine various convolutional or recurrent layer design choices. However, this framework is not practical as it was trained using 450 GPUs, so a more efficient method called ENAS was developed by [Pham et al., 2018]. The main difference between NAS and ENAS is that in ENAS the parameters of the cells (i.e. the nodes and connections within an recurrent cell) are shared by all models thus leveraging the power of transfer learning. First, a set of child models is proposed by an RNN controller, and the shared parameters of those models are trained to minimize average validation loss across all the models. Then, the parameters of the controller are updated using REINFORCE in order to increase the likelihood of producing models that have better validation performance.

Hyperparameter optimization can be done in a fully differentiable way if the dynamics of the training procedure are known ahead of time. [Maclaurin et al., 2015] do this by starting with the final weights of a neural network, then computing the gradient of the validation loss w.r.t. hyperparameters by iteratively determining what the neural network weights were at each step in the training procedure starting at the end. This allows them to not have to store the weights and gradients at every training iteration in memory as that is in-feasible, even for moderately-sized models.

3 Methods

We adopt the framework proposed in ENAS [Pham et al., 2018] for optimizing architectural hyperparameters, but focus on smaller scale experiments as proof of concept.

Suppose that $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ are our training and validation losses respectively. Additionally, suppose that $\mathbf{w}$ and λ are our elementary parameters and hyperparameters. Note that to optimize our hyperparameters - including architectural parameters - we wish to solve:

$$\lambda^* = \arg\min_{\lambda} \mathcal{L}_{\text{val}}(\lambda, \arg\min_{\mathbf{w}} \mathcal{L}_{\text{train}}(\lambda, \mathbf{w})) = \arg\min_{\lambda} \mathcal{L}_{\text{val}}(\lambda, \mathbf{w}^*(\lambda)) \tag{1}$$

If we wish to do gradient descent on the hyperparameters we must compute the following term:

$$\overbrace{\frac{d\mathcal{L}_{\text{val}}}{d\lambda}}^{\text{Outer gradient}} = \overbrace{\frac{\partial \mathcal{L}_{\text{val}}}{\partial \lambda}}^{\text{Direct gradient}} + \overbrace{\frac{\partial \mathcal{L}_{\text{val}}}{\partial \mathbf{w}^*}\frac{\partial \mathbf{w}^*}{\partial \lambda}}^{\text{Response gradient}} \tag{2}$$

Note how the Outer gradient is the sum of the direct gradient and the response gradient. The direct gradient is how the hyperparameters change the validation loss, given the elementary parameters $\mathbf{w}^*$ stay constant. The response gradient is how the validation loss changes as the elementary parameters change with how the optimal elementary parameters change as the hyperparameters vary. In ENAS, the direct gradient is computed using REINFORCE, while the response gradient is assumed to be 0. Here, we approximate the response gradient using either unrolled differentiation as in Maclaurin et al. [2015] or the Implicit Function Theorem as in Pedregosa [2016].

Additionally, we include optimizer hyperparameters in λ, whereas ENAS only includes architectural hyperparameters. Specifically, our controller will be multi-task, and the corresponding weight/hidden state sharing between the two tasks (selecting architecture hyperparameters, and selecting optimizer hyperparameters), is what allows for representing the interactions between the two types of hyperaparameters.

References

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Barret Zoph and Quoc V Le Google Brain. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING. Technical report. URL `https://arxiv.org/pdf/1611.01578.pdf`.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*, 2016.