Hyperparameter Optimization with Hypernets

Jonathan Lorraine & David Duvenaud
Department of Computer Science
University of Toronto
{lorraine, duvenaud}@cs.toronto.edu

Abstract

Machine learning models are usually tuned by nesting optimization of model weights inside the optimization of hyperparameters. We give a method to collapse this nested optimization into joint stochastic optimization of both weights and hyperparameters. Our method trains a neural network to output approximately optimal weights as a function of hyperparameters. We show that our method converges to locally optimal weights and hyperparameters for sufficiently large hypernets. We compare this method to standard hyperparameter optimization strategies and demonstrate its effectiveness for tuning thousands of hyperparameters.

1 Introduction

Model selection and hyperparameter tuning is a major bottleneck in designing predictive models. Hyperparameter optimization can be seen as a nested optimization: The inner optimization finds model parameters w which minimizes the training loss $\mathcal{L}_{\text{Train}}$ given hyperparameters λ. The outer optimization chooses λ to minimize a validation loss $\mathcal{L}_{\text{Valid.}}$:

$$\underset{\lambda}{\operatorname{argmin}} \underset{\text{Valid.}}{\mathcal{L}} \left(\underset{\mathrm{w}}{\operatorname{argmin}} \underset{\text{Train}}{\mathcal{L}} (\mathrm{w}, \lambda) \right) \tag{1}$$

Standard practice in machine learning solves (1) by gradient-free optimization of hyperparameters, such as grid search, random search, or Bayesian optimization. We usually estimate the parameters with stochastic optimization, but the true optimal parameters are a deterministic function of the hyperparameters λ:

$$\mathrm{w}^*(\lambda) = \underset{\mathrm{w}}{\operatorname{argmin}} \underset{\text{Train}}{\mathcal{L}} (\mathrm{w}, \lambda) \tag{2}$$

We propose to *learn this function*. Specifically, we train a neural network with inputs of hyperparameters, and outputs of an approximately optimal set of weights given the hyperparameters. This provides two major benefits: First, we can train the hypernet to convergence using stochastic gradient descent, denoted SGD, without training any particular model to completion. Second, differentiating through the hypernet allows us to optimize hyperparameters with gradient-based stochastic optimization.

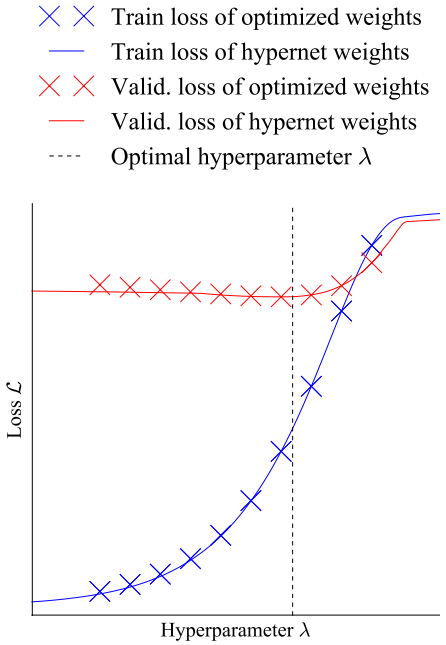

Figure 1: Training and validation loss of a neural net, estimated by cross-validation (crosses) or by a hypernet (lines), which outputs $7,850$-dimensional network weights. The training and validation loss can be cheaply evaluated at any hyperparameter value using a hypernet. Standard cross-validation requires training from scratch each time.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 Training a network to output optimal weights

A neural net which outputs the weights of another neural net is called a *hypernet* (Ha et al., 2016). At each iteration, we ask a hypernet to output a set of weights given some hyperparameters: $\mathrm{w} = \mathrm{w}_\phi(\lambda)$. Instead of updating weights w using the loss gradient $\partial \mathcal{L}(\mathrm{w}) / \partial \mathrm{w}$, we update the hypernet weights ϕ using the chain rule: $\frac{\partial \mathcal{L}(\mathrm{w}_\phi)}{\partial \mathrm{w}_\phi} \frac{\partial \mathrm{w}_\phi}{\partial \phi}$. We call this method *hyper-training*. The function $\mathrm{w}^*(\lambda)$ that outputs optimal weights for hyperparameters is a *best-response function* (Fudenberg & Levine, 1998). At convergence, we want our hypernet $\mathrm{w}_\phi(\lambda)$ to closely match the best-response.

2.1 Advantages and limitations of hypernet-based optimization

We can compare hyper-training to other model-based hyperparameter schemes, such as Bayesian optimization. Bayesian optimization (Snoek et al., 2012) builds a model of the validation loss as a function of hyperparameters, usually using a Gaussian process (Rasmussen & Williams, 2006) to track uncertainty. This approach has several disadvantages compared to hyper-training.

First, obtaining data for standard Bayesian optimization requires optimizing models from initialization for each set of hyperparameters. In contrast, hyper-training never needs to fully optimize any one model. Second, standard Bayesian optimization treats the validation loss as a black-box function: $\hat{\mathcal{L}_{\text{Valid.}}}(\lambda) = f(\lambda)$. In contrast, hyper-training takes advantage of the fact that the validation loss is a known, differentiable function which can be evaluated stochastically: $\hat{\mathcal{L}_{\text{Valid.}}}(\lambda) = \mathcal{L}_{\text{Valid.}}(\mathrm{w}_\phi(\lambda))$ This removes the need to learn a model of the validation loss.

Our approach only proposes making local changes to the hyperparameters and does not do uncertainty-based exploration. It is not obvious how to choose the distribution over hyperparameters $p(\lambda)$. A difficulty of this approach is that training a hypernet typically requires training several times as many parameters as training a single model. Both problems are addressed in section 2.3.

2.2 Asymptotic convergence properties

Algorithm 1 trains a hypernet using SGD, drawing hyperparameters from a fixed distribution $p(\lambda)$. This section proves that Algorithm 1 converges to a local best-response under mild assumptions. In particular, we show that, for a sufficiently large hypernet, the choice of $p(\lambda)$ does not matter as long as it has sufficient support.

Theorem 2.1. *Sufficiently powerful hypernets can learn continuous best-response functions, which minimizes the expected loss for any hyperparameter distribution.*

$$\text{There exists } \phi^*, \text{ such that for all } \lambda \in \text{support}(p(\lambda)),$$

$$\underset{\text{Train}}{\mathcal{L}}\left(\mathrm{w}_{\phi^*}(\lambda), \lambda\right) = \min_{\mathrm{w}} \underset{\text{Train}}{\mathcal{L}}(\mathrm{w}, \lambda) \text{ and } \phi^* = \operatorname*{argmin}_{\phi} \mathop{\mathbb{E}}_{p(\lambda')}\left[\underset{\text{Train}}{\mathcal{L}}(\mathrm{w}_\phi(\lambda'), \lambda')\right]$$

Proof. If w_ϕ is a universal approximator (Hornik, 1991) and the best-response is continuous in λ, then there exists optimal hypernet parameters ϕ^* such that for all hyperparameters λ, $\mathrm{w}_{\phi^*}(\lambda) = \operatorname{argmin}_{\mathrm{w}} \mathcal{L}_{\text{Train}}(\mathrm{w}, \lambda)$. Thus, $\mathcal{L}_{\text{Train}}(\mathrm{w}_{\phi^*}(\lambda), \lambda) = \min_{\mathrm{w}} \mathcal{L}_{\text{Train}}(\mathrm{w}, \lambda)$. In other words, universal approximator hypernets can learn continuous best-responses.

Substituting ϕ^* into the training loss gives $\mathbb{E}_{p(\lambda)}[\mathcal{L}_{\text{Train}}(\mathrm{w}_{\phi^*}(\lambda), \lambda)] = \mathbb{E}_{p(\lambda)}[\min_\phi \mathcal{L}_{\text{Train}}(\mathrm{w}_\phi(\lambda), \lambda)]$. By Jensen's inequality, $\min_\phi \mathbb{E}_{p(\lambda)}[\mathcal{L}_{\text{Train}}(\mathrm{w}_\phi(\lambda), \lambda)] \geq \mathbb{E}_{p(\lambda)}[\min_\phi \mathcal{L}_{\text{Train}}(\mathrm{w}_\phi(\lambda), \lambda)]$. Thus, $\phi^* = \operatorname{argmin}_\phi \mathbb{E}_{p(\lambda)}[\mathcal{L}_{\text{Train}}(\mathrm{w}_\phi(\lambda), \lambda)]$. In other words, if the hypernet learns the best-response it will simultaneously minimize the loss for every point in the $\text{support}(p(\lambda))$. □

Thus, having a universal approximator and a continuous best-response implies for all $\lambda \in \text{support}(p(\lambda))$, $\mathcal{L}_{\text{Valid.}}(\mathrm{w}_{\phi^*}(\lambda)) = \mathcal{L}_{\text{Valid.}}(\mathrm{w}^*(\lambda))$ because $\mathrm{w}_{\phi^*}(\lambda) = \mathrm{w}^*(\lambda)$. Theorem 2.1 holds for any $p(\lambda)$. However in practice, we have a limited-capacity hypernet, and so should choose a $p(\lambda)$ that puts most of its mass on promising hyperparameter values. This motivates the joint optimization of ϕ and $p(\lambda)$. In practice, there are no guarantees about the network being a universal approximator or the finite-time convergence of optimization.

Algorithm 1: Stochastic optimization of hypernet, then hyperparameters

initialize $\phi, \hat{\lambda}$
for T_{hypernet} steps
    $\mathbf{x} \sim$ Training data, $\lambda \sim p(\lambda)$
    $\phi = \phi - \alpha \nabla_\phi \mathcal{L}_{\text{Train}}(\mathbf{x}, \mathrm{w}_\phi(\lambda), \lambda)$
for $T_{\text{hyperparameter}}$ steps
    $\mathbf{x} \sim$ Validation data
    $\hat{\lambda} = \hat{\lambda} - \beta \nabla_{\hat{\lambda}} \mathcal{L}_{\text{Valid.}}(\mathbf{x}, \mathrm{w}_\phi(\hat{\lambda}))$
return $\hat{\lambda}, \mathrm{w}_\phi(\hat{\lambda})$

Algorithm 2: Stochastic optimization of hypernet and hyperparameters jointly

initialize $\phi, \hat{\lambda}$
for T_{joint} steps
    $\mathbf{x} \sim$ Training data, $\lambda \sim p(\lambda|\hat{\lambda})$
    $\phi = \phi - \alpha \nabla_\phi \mathcal{L}_{\text{Train}}(\mathbf{x}, \mathrm{w}_\phi(\hat{\lambda}), \hat{\lambda})$

    $\mathbf{x} \sim$ Validation data
    $\hat{\lambda} = \hat{\lambda} - \beta \nabla_{\hat{\lambda}} \mathcal{L}_{\text{Valid.}}(\mathbf{x}, \mathrm{w}_\phi(\hat{\lambda}))$
return $\hat{\lambda}, \mathrm{w}_\phi(\hat{\lambda})$

Figure 2: A side-by-side comparison of two variants of hyper-training. Algorithm 2 fuses the hypernet training and hyperparameter optimization into a single loop of SGD.

2.3 Jointly training parameters and hyperparameters

We use a limited-capacity hypernet, so it may not be possible to learn a best-response for all hyperparameters. Thus, we propose Algorithm 2, which only tries to learn a best-response locally. We introduce a "current" hyperparameter $\hat{\lambda}$, which is updated each iteration. We define a conditional hyperparameter distribution, $p(\lambda|\hat{\lambda})$, which only puts mass close to $\hat{\lambda}$.

Algorithm 2 combines the two phases of Algorithm 1 into one. Instead of first learning a hypernet that can output weights for any hyperparameter then optimizing the hyperparameters, Algorithm 2 only samples hyperparameters near the current best guess. This means the hypernet only has to be trained to estimate good enough weights for a small set of hyperparameters.

3 Related Work

Our work is closely related to the concurrent work of Brock et al. (2017), whose SMASH algorithm also approximates the optimal weights as a function of model architectures, to perform a gradient-free search over discrete model structures. Their work focuses on efficiently evaluating the performance of a variety of discrete model architectures, while we focus on efficiently exploring continuous spaces of models.

Model-free approaches Model-free approaches only use trial-and-error to explore the hyperparameter space. Simple model-free approaches applied to hyperparameter optimization include grid search and random search (Bergstra & Bengio, 2012). Hyperband (Li et al., 2016) combines bandit approaches with modeling the learning procedure.

Model-based approaches Model-based approaches try to build a surrogate function, which often allows gradient-based optimization or active learning. A common example is Bayesian optimization (Snoek et al., 2012). Freeze-thaw Bayesian optimization (Swersky et al., 2014) can condition on partially-optimized model performance.

Differentiation-based approaches Domke (2012) proposes to differentiate through unrolled optimization to approximate best-responses in nested optimization and Maclaurin et al. (2015) differentiate through entire unrolled learning procedures. A closely-related procedure to our method is the $T1 - T2$ method of Luketina et al. (2016), which also provides an algorithm for stochastic gradient-based optimization of hyperparameters.

Game theory Best-response functions are extensively studied as a solution concept in discrete and continuous multi-agent games (Fudenberg & Levine, 1998).

4 Experiments

In all experiments, Algorithms 1 or 2 are used to optimize weights of a linear regression on MNIST (LeCun et al., 1998) with an L_2 weight decay penalty weighted by $\exp(\lambda)$. The elementary model has $7,850$ weights.

4.1 Learning a global best-response

Our first experiment, shown in Figure 1, demonstrates learning a global approximation to a best-response function using Algorithm 1. To make visualization of the regularization loss easier, we use 10 training data points to exacerbate overfitting. We compare the performance of weights output by the hypernet to those trained by standard cross-validation. Thus, elementary weights were randomly initialized for each hyperparameter setting and optimized using Adam (Kingma & Ba, 2014) for $1,000$ iterations with a step size of 0.0001.

When training the hypernetwork, hyperparameters were sampled from a broad Gaussian distribution: $p(\lambda) = \mathcal{N}(0, 1.5)$. The hypernet has 50 hidden units which results in $400,450$ parameters of the hypernetwork. Each minibatch sampled 10 pairs of hyperparameters and the entire training data. Adam was used for training the hypernet, with a step size of 0.0001. The minimum of the best-response in Figure 1 is close to the true minimum of the validation loss, which shows a hypernet can satisfactorily approximate a global best-response function in small problems.

4.2 Optimizing 10 and $7,850$ Hyperparameters

Next, we optimized models with 10 and $7,850$ hyperparameters. All architectural choices are specified in the appendix. Figure 3, right, shows that our method converges more quickly and to a better optimum than either alternative method, demonstrating that medium-sized hyperparameter optimization problems can be solved with Algorithm 2. The left shows that Algorithm 2 performs better than random optimization. Standard Bayesian optimization cannot be scaled to this many hyperparameters. This experiment shows Algorithm 2 can optimize thousands of hyperparameters.

4.3 Estimating weights versus estimating loss

In this final experiment, we untangle the reason for the better performance of our method. First, we constructed a hyper-training set: We optimized 25 sets of weights to completion, given randomly-sampled hyperparameters. We also constructed a validation set of $10,215$ (optimized weight, hyperparameter) tuples generated in the same manner. We then fit a Gaussian process (GP) regression model on the hyper-training data. A hypernet is fit to the same dataset, optimizing training weights, not optimized validation loss. Finally, we optimize another hypernet using Algorithm 1, for the same amount of time as building the hyper-training set.

Figure 4 shows the distribution of prediction errors of these three models. We can see that the Gaussian process tends to underestimate loss. The hypernet trained with the same small fixed set of examples tends to overestimate loss. We conjecture the hypernet overestimates loss because it produces bad weights in regions where it doesn't have enough training data. Finally, the hypernet trained with Algorithm 1 produces errors tightly centered around 0.

5 Conclusions

In this paper, we:

- Presented algorithms that efficiently learn a differentiable approximation to a best-response without nested optimization.
- Showed empirically that hypernets can provide a better inductive bias for hyperparameter optimization than Gaussian processes fit directly to the validation loss.
- Gave a theoretical justification that sufficiently large networks will learn the best-response for all hyperparameters it is trained against.

We hope that this initial exploration of stochastic hyperparameter optimization will inspire further refinements, such as hyper-regularization methods, or uncertainty-aware exploration using Bayesian hypernetworks.

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Smash: One-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

Drew Fudenberg and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*, pp. 2952–2960, 2016.

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

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

A Experiment Images

Acknowledgments

We thank Dougal Maclaurin, Daniel Flam-Shepard, Daniel Roy, and Jack Klys for helpful discussions.

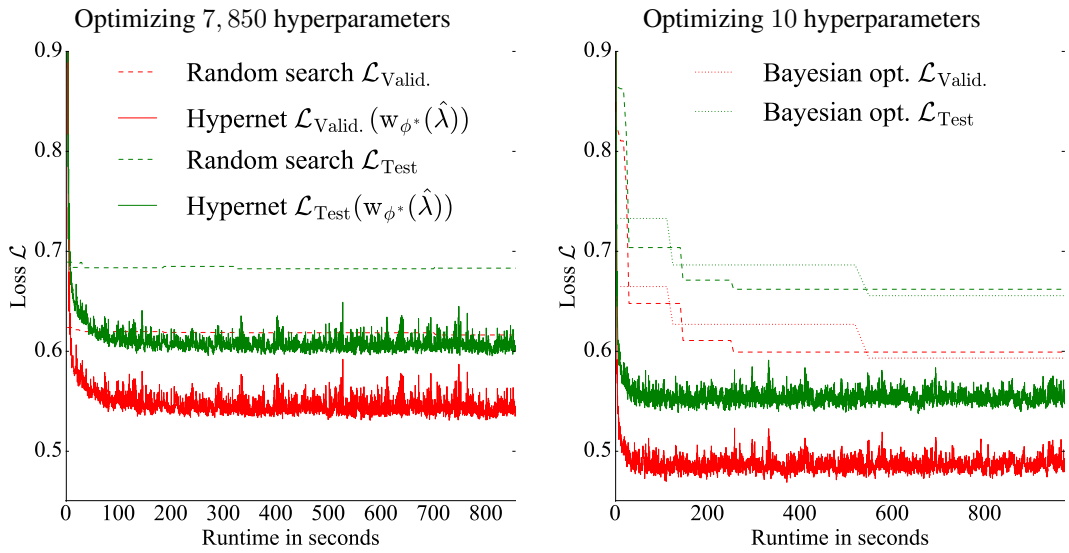

Figure 3: Validation and test losses during hyperparameter optimization. *Left:* A separate L_2 weight decay is applied to each weight in the model, resulting in $7,850$ hyperparameters. *Right:* A separate L_2 weight decay is applied to the weights each digit class, resulting in 10 hyperparameters. Hypernetwork-based optimization converges much more quickly than random search or Bayesian optimization. We also observe significant overfitting on the validation set for all methods. The architectural choices are as follows: The standard $50,000$ training data points and a mini-batch size of 100 for the validation and training sets are used. The conditional hyperparameter distribution is the same the prior experiment. Each iteration samples 10 pairs of hyperparameters and a mini-batch from the training data. Adam is used for training the hypernet, with a step size of 0.0001 for 10 iterations, alternated with 1 iteration of Adam on the hyperparameter with a step size of 0.0001. Algorithm 2 is compared against random search and a standard Bayesian optimization implementation from `sklearn`. A linear hypernet is used, for both sizes.

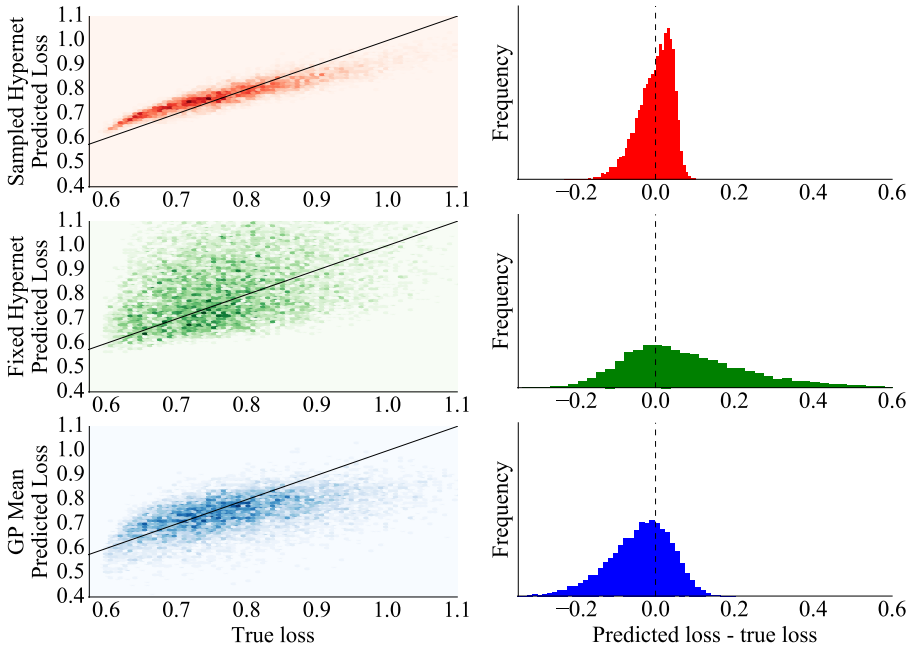

Figure 4: A comparison of a hypernet trained with stochastically sampled hyperparameters, a hypernet trained with a fixed set of hyperparameters, and a Gaussian process fit on a fixed set of hyperparameters and optimized losses. The two hypernets are linear models and are trained with the same optimizer parameters as the 10-dimensional hyperparameter optimization. *Left:* The distribution of predicted and true losses. The diagonal black line is where predicted loss equals true loss. *Right:* The distribution of differences between predicted and true losses.