Stochastic Hyperparameter Optimization through Hypernetworks

Jonathan Lorraine
& David Duvenaud

University of Toronto
Cross-validation nests optimization of network weights inside of optimization of hyperparameters.

Bi-level optimization is a game with a leading player and a following player. Each has their own objective.

The followers best-responding strategy depends on the leaders strategy: \( w^*(\lambda) = \arg\min_w \mathcal{L}_{\text{Train}}(w, \lambda) \)

Maclaurin et. Al. (2015) backprop through a training procedure to get gradients, but requires training from scratch each time.
Learning the best-response function

- Let’s learn the best-response function and amortize optimization!

\[ w^*(\lambda) = \arg\min_w \mathcal{L}_{\text{Train}}(w, \lambda) \]

- New gradient terms:

\[ \frac{\partial \mathcal{L}_{\text{Train}}(w_\phi)}{\partial w_\phi} \frac{\partial w_\phi}{\partial \phi} \text{ or } \frac{\partial \mathcal{L}_{\text{Valid}}(w_\phi(\lambda))}{\partial w_\phi(\lambda)} \frac{\partial w_\phi(\lambda)}{\partial \lambda} \]

\[ i = \arg\min_i \mathcal{L}_{\text{Train}}(w^{(i)}, \lambda^{(i)}, x) \]

Return \( \lambda^{(i)}, w^{(i)} \)

\[ \text{Algorithm 1 Standard cross-validation with stochastic optimization} \]

\[ \text{for } i = 1, \ldots, T_{\text{outer}} \text{ do} \]

\[ \begin{array}{c}
\text{initialize } w \\
\lambda = \text{hyperopt}(\lambda^{(1:i)}, \mathcal{L}_{\text{Valid.}}(w^{(1:i)})) \\
\text{loop} \\
\quad x \sim \text{Training data} \\
\quad w \leftarrow \alpha \nabla_w \mathcal{L}_{\text{Train}}(w, \lambda, x) \\
\quad \lambda^i, w^i = \lambda, w \\
\end{array} \]

Return \( \lambda^{(i)}, w^{(i)} \)

\[ \text{Algorithm 2 Optimization of hyper-network, then hyperparameters} \]

\[ \begin{array}{c}
\text{initialize } \phi \\
\text{initialize } \lambda \\
\text{loop} \\
\quad x \sim \text{Training data, } \lambda \sim p(\lambda) \\
\quad \phi \leftarrow \alpha \nabla_\phi \mathcal{L}_{\text{Train}}(w_\phi(\lambda), \lambda, x) \\
\end{array} \]

\[ \begin{array}{c}
\text{loop} \\
\quad x \sim \text{Validation data} \\
\quad \hat{\lambda} \leftarrow \beta \nabla_{\hat{\lambda}} \mathcal{L}_{\text{Valid.}}(w_\phi(\hat{\lambda}), x) \\
\end{array} \]

Return \( \hat{\lambda}, w_\phi(\hat{\lambda}) \)
Figure 2: The validation loss of a neural net, estimated by cross-validation (crosses) or by a hypernetwork (line), which outputs 7,850-dimensional network weights. Cross-validation requires optimizing from scratch each time. The hypernetwork can be used to evaluate the validation loss cheaply.
**Algorithms**

**Algorithm 2 Optimization of hyper-network, then hyperparameters**

1. Initialize $\phi$
2. Initialize $\lambda$
3. Loop
   - $x \sim$ Training data, $\lambda \sim p(\lambda)$
   - $\phi := \alpha \nabla_\phi \mathcal{L}_{\text{Train}}(w_\phi(\lambda), \lambda, x)$
4. Loop
   - $x \sim$ Validation data
   - $\lambda := \beta \nabla_\lambda \mathcal{L}_{\text{Valid.}}(w_\phi(\lambda), x)$
5. Return $\lambda, w_\phi(\lambda)$

**Algorithm 3 Joint optimization of hyper-network and hyperparameters**

1. Initialize $\phi$
2. Initialize $\lambda$
3. Loop
   - $x \sim$ Training data, $\lambda \sim p(\lambda|\hat{\lambda})$
   - $\phi := \alpha \nabla_\phi \mathcal{L}_{\text{Train}}(w_\phi(\lambda), \lambda, x)$
4. Loop
   - $x \sim$ Validation data
   - $\lambda := \beta \nabla_\lambda \mathcal{L}_{\text{Valid.}}(w_\phi(\lambda), x)$
5. Return $\lambda, w_\phi(\lambda)$

**Algorithm 4 Simplified joint optimization of hyper-network and hyperparameters**

1. Initialize $\phi, \lambda$
2. Loop
   - $x \sim$ Training data, $x' \sim$ Validation data
   - $\phi := \alpha \nabla_\phi \mathcal{L}_{\text{Train}}(w_\phi(\hat{\lambda}), \hat{\lambda}, x)$
   - $\hat{\lambda} := \beta \nabla_\hat{\lambda} \mathcal{L}_{\text{Valid.}}(w_\phi(\hat{\lambda}), x')$
3. Return $\hat{\lambda}, w_\phi(\hat{\lambda})$

- Local Optimization
  - Limited capacity hypernetwork in practice.
  - Learn the best-response in some small neighborhood about our current hyperparameter.
Figure 2: The validation loss of a neural net, estimated by cross-validation (crosses) or by a hypernetwork (line), which outputs 7,850-dimensional network weights. Cross-validation requires optimizing from scratch each time. The hypernetwork can be used to evaluate the validation loss cheaply.

Figure 4: Training and validation losses of a neural network, estimated by cross-validation (crosses) or a linear hypernetwork (lines). The hypernetwork’s limited capacity makes it only accurate where the hyperparameter distribution puts mass.
Figure 3: A visualization of exact (blue) and approximate (red) optimal weights as a function of hyperparameters. The approximately optimal weights $w_{\phi^*}$ are output by a linear model fit at $\hat{\lambda}$. The true optimal hyperparameter is $\lambda^*$, while the hyperparameter estimated using approximately optimal weights is nearby at $\lambda_{\phi^*}$. 
Idea: Hyper-training is effective because it partially optimizes across many hyperparameters.
Limitations

- No inner optimization parameters can be tuned (they don’t exist!).

- Hard to tune discrete hyperparameters with gradients (working on it).

- No uncertainty based exploration.

- Hard to choose the distribution of hyperparameters to train against.
  - Future work: use implicit function theorem instead?
Takeaway and Future Directions

- Currently using a linear hypernet - consider a net with 10,000,000 weights and 10 hyperparameters.

- Try optimizing other hyperparameters: e.g. training data

\[ w^*(\lambda) = \arg\min_w \sum_{x_i, t_i \in D_i} L_p(y_w(x_i), t_i) + L_t(y_w(X_t)) \]

- Main point: Learning the best-response function lets you collapse a nested optimization problem into a joint optimization problem. Can be applied to GANs, and possibly to funding Nash equilibria more generally.
Optimizing weight dropout on MNIST.
Related Work

- MAML
- Efficient Neural Architecture search