Forward and Reverse Gradient-Based Hyperparameter Optimization

Luca Franceschi $^{1,2}$, Michele Donini $^1$, Paolo Frasconi $^3$, Massimiliano Pontil $^{1,2}$

$^1$Istituto Italiano di Tecnologia, Genoa, Italy
$^2$Dept of Computer Science, University College London, UK
$^3$Dept of Information Engineering, Universit degli Studi di Firenze, Italy

ICML 2017/ Presenter: Ji Gao
Outline

1 Motivation

2 Method
   • Overview
   • Optimization

3 Complexity analysis

4 Experiment
Choose hyperparameters in optimization are hard
Could we automatically select hyperparameters?
Hyperparameter optimization: Construct a response function of the hyperparameters and explore the hyperparameter space to seek an optimum
Related Work

- Grid search: List parameters on a grid and train all of them. Problem: Impractical when number of hyperparameters is large. Even outperform by random search.

- Bayesian optimization: Treat the global process as a random function and place a prior over it. After that, construct an acquisition function (referred to as infill sampling criteria) that determines the next query point.

- Gradient-based methods: Use the gradient method to optimize hyperparameters.
Hyperparameters

s: state in $R^d$, including weights (object) and hyperparameters $\lambda$.

$$s_t = \Phi_t(s_{t-1}, \lambda)$$

An example in such definition:

**Gradient Descent with Momentum**

w: weights. J: objective function. $\lambda$: hyperparameters

$s_t = (v_t, w_t)$:

$$v_t = \mu v_{t-1} + \nabla J_t(w_{t-1})$$

$$w_t = w_{t-1} - \eta(\mu v_{t-1} - \nabla J_t(w_{t-1}))$$

In this case: $\lambda = (\mu, \eta)$
Problem formulation

Goal of hyperparameter optimization

Solve:

$$\min_{\lambda} f(\lambda)$$

Where a response function $f : R^m \rightarrow R$ is defined at $\lambda \in R^m$ as

$$f(\lambda) = E(s_T(\lambda))$$

E: Validation error
Problem formulation - Optimization

Goal of hyperparameter optimization

Solve:

\[
\min_{\lambda, s_1, \ldots, s_T} E(s_T)
\]

Subject to: \( s_t = \Phi_t(s_{t-1}, \lambda) \)

- Lagrangian:

\[
L(s, \lambda, \alpha) = E(s_T) + \sum_{t=1}^{T} \alpha_t (\Phi_t(s_{t-1}, \lambda) - s_t)
\]
Lagrangian:

\[ L(s, \lambda, \alpha) = E(s_T) + \sum_{t=1}^{T} \alpha_t(\Phi_t(s_{t-1}, \lambda) - s_t) \]

Derivatives of Lagrangian:

\[ \frac{\partial L}{\partial a_t} = \Phi_t(s_{t-1}, \lambda) - s_t, \quad t = 1..T \]

\[ \frac{\partial L}{\partial s_t} = a_{t+1} \frac{\partial \Phi_t(s_t, \lambda)}{\partial s_t} - a_t, \quad t = 1..T \]

\[ \frac{\partial L}{\partial s_T} = \nabla E(s_T) - a_T \]

\[ \frac{\partial L}{\partial \lambda} = \sum_{t=1}^{T} \alpha_t \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial \lambda} \]
Solution can be achieved by setting each derivatives to 0.

Solution:

Let $A_t = \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial s_{t-1}}$, $B_t = \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial \lambda}$

The solution is:

$$a_t = \nabla E(s_T) A_{t+1} \ldots A_T$$

And we have:

$$\frac{\partial L}{\partial \lambda} = \nabla E(s_T) \sum_{t=1}^{T} (A_{t+1} \ldots A_T) B_t$$

(1)
Algorithm 1 HO-REVERSE

Input: $\lambda$ current values of the hyperparameters, $s_0$ initial optimization state
Output: Gradient of validation error w.r.t. $\lambda$

for $t = 1$ to $T$ do
    $s_t = \Phi_t(s_{t-1}, \lambda)$
end for

$\alpha_T = \nabla E(s_T)$
$g = 0$

for $t = T - 1$ downto 1 do
    $\alpha_t = \alpha_{t+1} A_{t+1}$
    $g = g + \alpha_t B_t$
end for

return $g$
Another way to Calculate

- We have:
  \[ \nabla f(\lambda) = \nabla E(S_T) \frac{ds_T}{d\lambda} \]

- Let \[ Z_t = \frac{ds_T}{d\lambda}, \]
  \[ Z_t = A_t Z_{t-1} + B_t \]

- Lead to a recursive solution
**Algorithm 2 HO-FORWARD**

**Input:** $\lambda$ current values of the hyperparameters, $s_0$ initial optimization state

**Output:** Gradient of validation error w.r.t. $\lambda$

$Z_0 = 0$

for $t = 1$ to $T$ do

$s_t = \Phi_t(s_{t-1}, \lambda)$

$Z_t = A_tZ_{t-1} + B_t$

end for

return $\nabla E(s)Z_T$

- Can be real-time updated.
Algorithmic Differentiation: Techniques to numerically evaluate the derivative of a function.

Two modes of AD: Forward mode and Reverse mode.
Complexity of Algorithmic Differentiation

- Complexity of calculating the Jacobian matrix (the matrix of all first-order partial derivatives):
  Suppose $f : \mathbb{R}^n \to \mathbb{R}^p$ can be evaluated in time $c(n, p)$ and space $s(n, p)$. We have:
  - For any vector $r \in \mathbb{R}^n$, product of $r$ and Jacobian matrix $J_F r$ can be evaluated in time $O(c(n, p))$ and space $O(s(n, p))$ using forward mode AD.
  - For any vector $q \in \mathbb{R}^p$, product of $q$ and Jacobian matrix $q^T J_F$ can be evaluated in time $O(c(n, p))$ and space $O(s(n, p))$ using reverse mode AD.
  - Jacobian can be calculated in time $O(nc(n, p))$ using forward mode, and $O(pc(n, p))$ using reverse mode.
Suppose $s_t = \Phi_t(s_{t-1}, \lambda)$ can be updated in time $g(d, m)$ and space $h(d, m)$. For Algorithm 1:

```
Algorithm 1 HO-REVERSE

Input: $\lambda$ current values of the hyperparameters, $s_0$ initial optimization state
Output: Gradient of validation error w.r.t. $\lambda$
for $t = 1$ to $T$ do
    $s_t = \Phi_t(s_{t-1}, \lambda)$
end for
$\alpha_T = \nabla E(s_T)$
$g = 0$
for $t = T - 1$ downto 1 do
    $\alpha_t = \alpha_{t+1} A_{t+1}$
    $g = g + \alpha_t B_t$
end for
return $g$
```

Each step of $a_{t+1} A_{t+1}$ and $a_t B_t$ cost $O(g(d, m))$ time. So it’s totally $O(Tg(d, m))$ time. For space, we need to store all $s_t$, which requires $O(Th(d, m))$ space.
Complexity - Algorithm 2

For Algorithm 2:

```
Algorithm 2 HO-FORWARD

Input: λ current values of the hyperparameters, s₀ initial optimization state
Output: Gradient of validation error w.r.t. λ
Z₀ = 0
for t = 1 to T do
  sₜ = Φₜ(sₜ₋₁, λ)
  Zₜ = AₜZₜ₋₁ + Bₜ
end for
return ∇E(s)Zₜ
```

Each step of $A_t Z_{t+1}$ require $m$ Jacobian vector multiplication, so the cost is $O(mg(d, m))$ time. So it’s totally $O(Tmg(d, m))$ time. For space, we only need to store the current $s_t$, which requires $O(h(d, m))$ space.
Experiment 1 - Data Hyper-cleaning

- Task: Have a large dataset with corrupted labels. Can only afford to clean a subset. Train a model.
- Method: Weighting every training sample a hyperparameter in \([0,1]\). Train with a weighted loss on the cleaned validation set.
- Train a plain softmax regression model with weight \(W\) and bias \(b\).
- Optimization problem:

\[
\begin{align*}
\min_{\lambda} & \quad E_{\text{val}}(W_T, b_T) \\
\text{Subject to:} & \quad \lambda \in [0, 1]^{N_{tr}}, \|\lambda\|_1 \leq R
\end{align*}
\]

- Experiment design: 5000 examples from MNIST dataset as the training data, corrupt 2500 of them. Have 5000 more as validation data, and 10000 as test set.
Experiment 1 result

**Table 1:** Test accuracies for the baseline, the oracle, and using data hyper-cleaning with four different values of $R$. The reported $F_1$ measure is the performance of the hyper-cleaner in correctly identifying the corrupted training examples.

<table>
<thead>
<tr>
<th></th>
<th>Accuracy %</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>90.46</td>
<td>1.0000</td>
</tr>
<tr>
<td>Baseline</td>
<td>87.74</td>
<td>-</td>
</tr>
<tr>
<td>DH-1000</td>
<td>90.07</td>
<td>0.9137</td>
</tr>
<tr>
<td>DH-1500</td>
<td>90.06</td>
<td>0.9244</td>
</tr>
<tr>
<td>DH-2000</td>
<td>90.00</td>
<td>0.9211</td>
</tr>
<tr>
<td>DH-2500</td>
<td>90.09</td>
<td>0.9217</td>
</tr>
</tbody>
</table>

Oracle: Train with 2500 correct samples together with validation set.
Baseline: Train with corrupted data and validation set.
DH-R: Optimize and find a cleaned subset $D_c$ with a different R value, and finally train with $D_c$ and the validation set.
Figure 2: Right vertical axis: accuracies of DH-1000 on validation and test sets. Left vertical axis: number of discarded examples among noisy (True Positive, TP) and clean (False Positive, FP) ones.

It can successfully discard corrupted samples.
Experiment 2 - Multiple task learning

- Task: Find simultaneously the model of several different related tasks. For example, few shot learning.

- Experiment design: Try both CIFAR-10 and CIFAR-100. 50 samples on CIFAR-10, 300 samples on CIFAR-100 as training set. Same size of validation set, and all rest for testing. Use pretrained Inception-V3 model to fetch the feature.

- Use a regularizer from [Evgeniou et al., 2005]
  \[ \Omega_{A,\rho}(W) = \sum_{j,k=1}^{K} A_{j,k} ||w_j - w_k||_2^2 + \rho \sum_{k=1}^{K} ||w_k||^2 \]

- Training error \( E_{tr}(W) = \sum_{i} l(Wx_i + b, y_i) + \Omega_{A,\rho}(W) \)

- Optimization problem:
  \[
  \min_{\lambda} E_{val}(W_T, b_T) \\
  \text{Subject to: } \rho \geq 0, A \geq 0
  \]
Table 2: Test accuracy ± standard deviation on CIFAR-10 and CIFAR-100 for single task learning (STL), naive MTL (NMTL) and our approach without (HMTL) and with (HMTL-S) the L1-norm constraint on matrix $A$.

<table>
<thead>
<tr>
<th></th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>STL</td>
<td>67.47±2.78</td>
<td>18.99±1.12</td>
</tr>
<tr>
<td>NMTL</td>
<td>69.41±1.90</td>
<td>19.19±0.75</td>
</tr>
<tr>
<td>HMTL</td>
<td>70.85±1.87</td>
<td>21.15±0.36</td>
</tr>
<tr>
<td>HMTL-S</td>
<td>71.62±1.34</td>
<td>22.09±0.29</td>
</tr>
</tbody>
</table>

HMTL-S algorithm find the following relationship graph:

Figure 3: Relationship graph of CIFAR-10 classes. Edges represent interaction strength between classes.
Task: Phone state classification over 183 classes.

Experiment design: Data: TIMIT phonetic recognition dataset. Model: A previous multi task learning framework [Badino, 2016].

Hyperparameters: learning rate $\eta$, momentum $\mu$ and $\rho$, a hyperparameter of the algorithm
Table 3: Frame level phone-state classification accuracy on standard TIMIT test set and execution time in minutes on one Titan X GPU. For RS, we set a time budget of 300 minutes.

<table>
<thead>
<tr>
<th></th>
<th>Accuracy %</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>59.81</td>
<td>12</td>
</tr>
<tr>
<td>RS</td>
<td>60.36</td>
<td>300</td>
</tr>
<tr>
<td>RTHO</td>
<td>61.97</td>
<td>164</td>
</tr>
<tr>
<td>RTHO-NT</td>
<td>61.38</td>
<td>289</td>
</tr>
</tbody>
</table>
Figure 4: The horizontal axis runs with the hyper-batches. Top-left: frame level accuracy on mini-batches (Training) and on a randomly selected subset of the validation set (Validation). Top-right: validation error $E_{val}$ on the same subset of the validation set. Bottom-left: evolution of optimizer hyperparameters $\eta$ and $\mu$. Bottom-right: evolution of design hyperparameter $\rho$. 