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

Motivation

- 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 λ.

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

An example in such definition:

Gradient Descent with Momentum

w: weights. J: objective function. λ: 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:

$$\min_{\lambda, s_1, \ldots s_t} E(s_T)$$

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

- Lagrangian:

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

Problem formulation - Optimization

- 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:

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

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

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

$$\frac{\partial L}{\partial \lambda} = \sum_{t=1}^{T} \alpha_t \frac{\partial \Phi_t(s_{t-1}, \lambda)}{\partial \lambda}$$

- 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 the solution is:

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

And we have:

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

Algorithm

Algorithm 1 HO-REVERSE

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

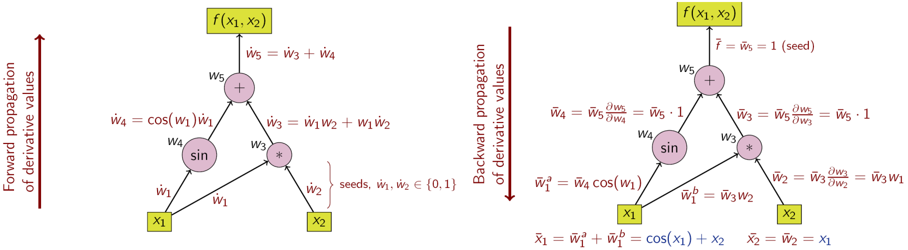
Algorithm2

Algorithm 2 HO-FORWARD

Input: λ current values of the hyperparameters, s_0 initial optimization state
Output: Gradient of validation error w.r.t. λ
$Z_0 = 0$
for $t = 1$ **to** T **do**
  $s_t = \Phi_t(s_{t-1}, \lambda)$
  $Z_t = A_t Z_{t-1} + B_t$
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 $f : R^n \rightarrow R^p$ can be evaluated in time $c(n, p)$ and space $s(n, p)$. We have:
  - For any vector $r \in R^n$, 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 $q \in R^p$, 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.

Complexity

Suppose $s_t = \Phi_t(s_t - 1, \lambda)$ can updated in time $g(d, m)$ and space $h(d, m)$.
For Algorithm 1:

Algorithm 1 HO-Reverse

Input: λ current values of the hyperparameters, s_0 initial optimization state
Output: Gradient of validation error w.r.t. λ
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, s_0 initial optimization state
Output: Gradient of validation error w.r.t. λ
$Z_0 = 0$
for $t = 1$ **to** T **do**
  $s_t = \Phi_t(s_{t-1}, \lambda)$
  $Z_t = A_t Z_{t-1} + B_t$
end for
return $\nabla E(s) Z_T$

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:

$$\min_{\lambda} E_{\text{val}}(W_T, b_T)$$

$$\text{Subject to: } \lambda \in [0,1]^{N_{tr}}, ||\lambda||_1 \le R$$

- 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.

	Accuracy %	F_1
Oracle	90.46	1.0000
Baseline	87.74	-
DH-1000	90.07	0.9137
DH-1500	90.06	0.9244
DH-2000	90.00	0.9211
DH-2500	90.09	0.9217

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.

Experiment 1 result

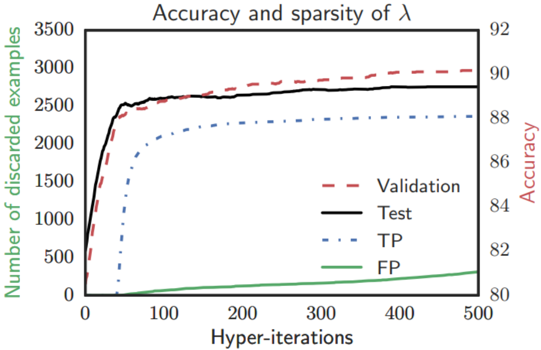


Figure 2: Right vertical axis: accuracies of DH-1000 on validation and test sets. Left vertical axis: number of discarded examples among noisy (True Positive, TP) and clean (False Positive, FP) ones.

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:

$$\min_{\lambda} E_{\text{val}}(W_T, b_T)$$

Subject to: $\rho \geq 0, A \geq 0$

Experiment 2 result

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.

	CIFAR-10	CIFAR-100
STL	67.47±2.78	18.99±1.12
NMTL	69.41±1.90	19.19±0.75
HMTL	70.85±1.87	21.15±0.36
HMTL-S	71.62±1.34	22.09±0.29

HMTL-S algorithm find the following relationship graph:

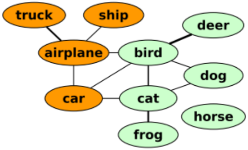


Figure 3: Relationship graph of CIFAR-10 classes. Edges represent interaction strength between classes.

Experiment 3 - Phone classification

- 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 η, momentum μ and ρ, a hyperparameter of the algorithm

Experiment 3 result

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.

	Accuracy %	Time (min)
Vanilla	59.81	12
RS	60.36	300
RTHO	61.97	164
RTHO-NT	61.38	289

Experiment 3 result

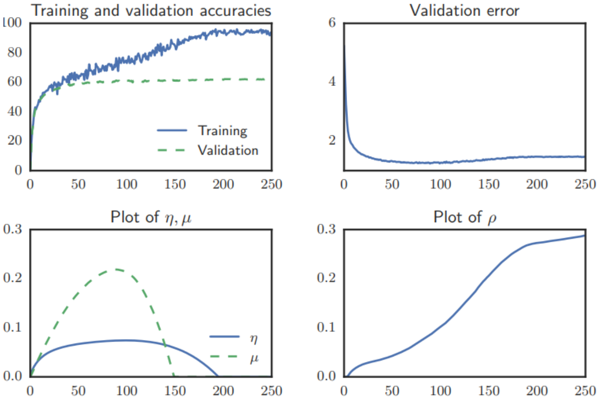


Figure 4: The horizontal axis runs with the hyper-batches. Top-left: frame level accuracy on mini-batches (Training) and on a randomly selected subset of the validation set (Validation). Top-right: validation error E_{val} on the same subset of the validation set. Bottom-left: evolution of optimizer hyperparameters η and μ. Bottom-right: evolution of design hyperparameter ρ.