Alpha MAML: Adaptive Model-Agnostic Meta-Learning

Harkirat Singh Behel, Atılım Günes Baydin, Philip H.S Torr
Department of Engineering Science, University of Oxford
{harkirat,gunes,phst}@robots.ox.ac.uk

Introduction

- Shortcomings of MAML [1]:
  - training very sensitive to learning rate values [2].
  - requires time-consuming hyperparameter tuning.
  - high training time.

- Aim:
  - make it possible to use MAML without or with significantly less parameter tuning.
  - make the algorithm converge in fewer iterations.

- Contributions:
  - an online hyperparameter adaptation scheme that eliminates the need to tune meta-learning and learning rates.
  - requires no extra gradient computations, only memory cost of storing previous gradient.
  - based on the hypergradient descent (HD) algorithm [3].

MAML: Model-Agnostic Meta-Learning

The MAML algorithm, given model parameters \( \theta \), aims to adapt to a new task \( T \) with SGD:

\[
\theta_{t+1} = \theta - \alpha \nabla_{\theta} L_{\text{train}}(f_{\theta}(x))
\]

where \( t \) is the task number, \( \alpha \) is the learning rate, \( L_{\text{train}}(\cdot) \) and \( L_{\text{test}}(\cdot) \) denote the training and test set within task \( t \). The meta-update equation is:

\[
\theta_{t+1} = \theta - \beta \nabla_{\theta} L_{\text{train}}(f_{\theta}(x))
\]

where \( \beta \) is the meta step size. The full algorithm is shown in Algorithm 1. MAML has two learning rates \( \alpha \) and \( \beta \), which require time-consuming hyperparameter tuning.

Alpha MAML: Adaptive Model-Agnostic Meta-Learning

We would like to derive update rules for the learning rates \( \alpha \) and \( \beta \) as well. We derive the partial gradient of \( L_{\text{train}}(\cdot) \) with respect to the learning rate \( \alpha \):

\[
\frac{\partial L_{\text{train}}}{\partial \alpha}(f_{\theta}(x)) = \frac{\partial}{\partial \alpha} \left( \frac{1}{N} \sum_{i=1}^{N} L_{\text{train}}(f_{\theta}(x_i)) \right)
\]

where \( \alpha_{\text{hyper}} \) is the hyper learning rate for \( \alpha \). Similar to Eq. 1, we derive the partial derivative of \( L_{\text{train}}(\cdot) \) with respect to the meta-learning rate \( \beta \):

\[
\frac{\partial L_{\text{train}}}{\partial \beta}(f_{\theta}(x)) = \frac{\partial}{\partial \beta} \left( \frac{1}{N} \sum_{i=1}^{N} L_{\text{train}}(f_{\theta}(x_i)) \right)
\]

We can estimate \( \alpha \), as follows:

\[
f_{\alpha} = \alpha + \alpha_{\text{hyper}} \nabla_{\alpha} L_{\text{train}}(f_{\theta}(x)) = \alpha_{\text{hyper}} \nabla_{\alpha} L_{\text{train}}(f_{\theta}(x)), \quad \alpha_{\text{hyper}} \]

where \( \alpha_{\text{hyper}} \) is the hyper learning rate for \( \alpha \). The final Alpha MAML algorithm can thus be written down into just 4 update equations:

\[
f_{\alpha} = \alpha + \alpha_{\text{hyper}} \nabla_{\alpha} L_{\text{train}}(f_{\theta}(x)),
\]

\[
f_{\beta} = \beta + \beta_{\text{hyper}} \nabla_{\beta} L_{\text{train}}(f_{\theta}(x)), \quad \beta_{\text{hyper}} \]

\[
\nabla_{\alpha} \frac{\partial L_{\text{train}}}{\partial \alpha}(f_{\theta}(x)) = \nabla_{\alpha} \left( \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_{\text{train}}}{\partial \alpha}(f_{\theta}(x_i)) \right)
\]

\[
\nabla_{\beta} \frac{\partial L_{\text{train}}}{\partial \beta}(f_{\theta}(x)) = \nabla_{\beta} \left( \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_{\text{train}}}{\partial \beta}(f_{\theta}(x_i)) \right)
\]

requiring only the extra memory storage of previous gradient, and no extra gradient needs to be computed. The full algorithm is shown in Algorithm 2.

Algorithm 1 MAML

Input: \( \mu(T) \): distribution over tasks.
Input: \( \alpha, \beta \): learning rates randomly initialize \( \theta \).
while not done do
Sample batch of tasks \( T \sim \mu(T) \)
for all \( T \) do
Evaluate \( \nabla L_{\text{train}}(f_{\theta}) \) with respect to \( \theta \).
Compute adapted parameters with gradient descent:
\n\nend for
Update \( \theta \) = \( \theta - \alpha \nabla L_{\text{train}}(f_{\theta}) \)
end while

Algorithm 2 Alpha MAML

Input: \( \mu(T) \): distribution over tasks.
Input: \( \alpha, \beta \): initial learning rates
Input: \( \alpha_{\text{hyper}}, \beta_{\text{hyper}} \): hypergradient learning rates
Input: \( \alpha, \beta \): randomly initialize \( \theta \).
while not done do
Sample batch of tasks \( T \sim \mu(T) \)
for all \( T \) do
Evaluate \( \nabla L_{\text{train}}(f_{\theta}) \) with respect to \( \theta \).
Compute adapted parameters with gradient descent:
\n\end for
Update \( \theta \) = \( \theta - \alpha \nabla L_{\text{train}}(f_{\theta}) \)
end while

Behaviour of MAML vs Alpha-MAML

By performing online updates with hyperparameters, Alpha-MAML:
- adapts \( \alpha \), \( \beta \) on-the-fly within the main optimization loop, and
- significantly reduces the need to tune the initial learning rates \( \alpha_0 \) and \( \beta_0 \).

Here we choose:
- a good case (where the MAML user picked a good pair of \( \alpha_0 \) and \( \beta_0 \), i.e., the tuned case)
- a bad case (where the user picked a bad pair of \( \alpha_0 \) and \( \beta_0 \), i.e., an untuned case)

In sensitivity with Respect to Hyperparameter Choices

We present experiments to study the effect of initial learning rate values on the number of iterations needed for the algorithms to reach a particular chosen loss threshold, i.e., the grid search for tuning the learning rate hyper-parameters.

We observe:
- MAML shows very slow convergence for a range of initial learning rates. In comparison, Alpha MAML shows comparatively faster convergence for these initial learning rates also, for a wide range of \( \alpha_{\text{hyper}} \) and \( \beta_{\text{hyper}} \).
- For cases where MAML shows fast convergence, Alpha MAML also shows fast convergence for all values of \( \alpha_{\text{hyper}} \) and \( \beta_{\text{hyper}} \).
- No matter which values one chooses for the initial learning and meta-learning rates \( \alpha_0 \) and \( \beta_0 \), Alpha MAML always shows faster, or in the worst case the same, convergence as MAML.

References