Review on Optimization-Based Meta Learning

Presenter: Zhe Wang

https://qdata.github.io/deep2Read

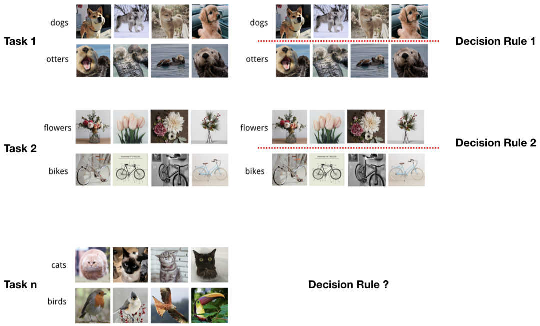
Zhe Wang

201909

Content

Meta-learning

Meta Learning (learning to learn) aims to learn from previous tasks.



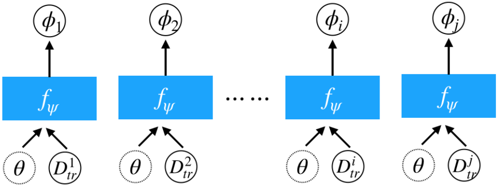
Motivation

Why do we need meta learning?

- For task n, we have limited data (n way k shot), hard to find a classifier without over fitting. (Few-shot learning)
- For task n, learning prior from previous task $1, 2, \cdots k$ boosts the performance of n. (Multi-task learning)
- Fast adaptation, learn shared structure.

Be careful lots of work point out if there is no shared structure for different tasks.

- Black-box based meta learning.



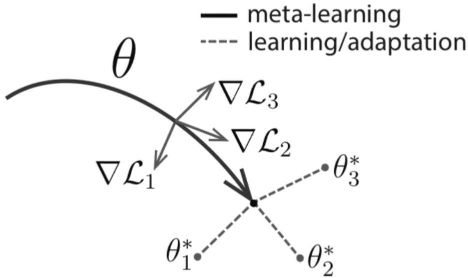
Example + Variants: RNN based, attention based. Modeling $p(\phi_i|D, \theta)$ with neural network.

- Optimization based meta learning (main focus)

Content

Multi-task Learning

MAML, model-agnostic meta learning, optimization framework for meta learning. (supervised learning and reinforcement learning)



Main idea: There is shared initialization for all tasks, the target is to find a good initialization.

MAML

Machine Learning:

- data distribution D
- data sample $\{x^i, y^i\}$ in each batch
- target: find θ, s.t. $f(\theta, D(X)) = D(Y)$

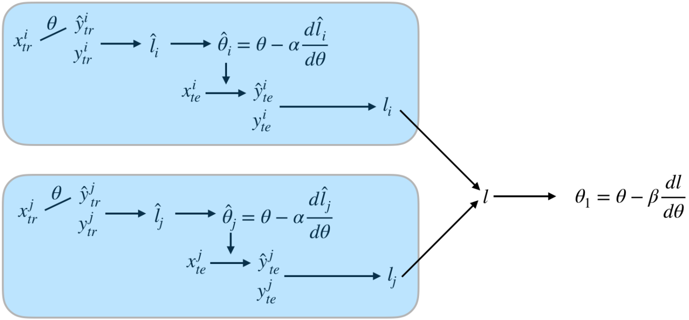
Meta Learning:

- task distribution D_τ
- task sample $\{D^i_{tr}, D^i_{te}\}$ in each batch
- target: find θ, s.t. $f(\theta^i_k, D^i_{te}(X)) = D^i_{te}(Y)$, where $\theta^i_k = SGD_\theta(D^i_{tr}(X), D^i_{tr}(Y))$

MAML

Multi-task Learning:

- inner task: find θ^i_k opt on D^i_{tr}
- out task: find θ, s.t. $\sum_{i=1}^{n} ||D^i_{te}(Y) - f(\theta^i_k, D^i_{te}(X))||$ is opt



MAML

During test time:

$$x_{tr}^{t} \xrightarrow{\theta} \begin{matrix} \hat{y}_{tr}^{t} \\ y_{tr}^{t} \end{matrix} \longrightarrow \hat{l}_t \longrightarrow \hat{\theta}_t = \theta - \alpha \frac{d\hat{l}_t}{d\theta}$$

Fast adaptation, one step fine-tuning. (Learn prior knowledge from multi-tasks, and embedding into initialization)

Limitation:

- a huge computation graph
- expensive memory cost
- deterministic method, disregard ambiguity over the underlying function

RL

MAML for reinforcement learning:

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
end while

Meta-SGD

Not efficient to solve the inner level optimization. (big computational graph, heavy gradient calculation)

Second order opt, the inner solver have a big effect on the final performance.

First order MAML

Vanilla MAML can be formalized as:

$$\arg\min_{\theta} E_{\tau}(L_{\tau,te}(U^{k}_{\tau,tr}(\theta))) \tag{1}$$

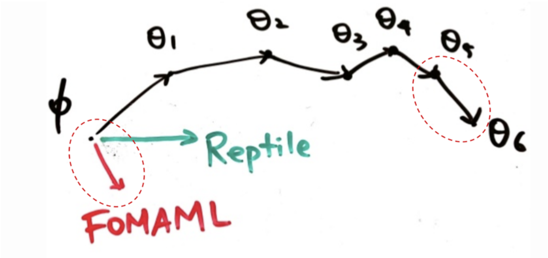
Thus, we have:

$$\begin{aligned} g_{maml} &= \frac{\partial}{\partial\theta}(L_{\tau,te}(U_{\tau,tr}(\theta)) \\ &= L'_{\tau,te}(\hat{\theta})U'_{\tau,tr}(\theta) \end{aligned} \tag{2}$$

where $\hat{\theta} = U_{\tau,tr}(\theta)$. In the experiments, they found if set $U' = I$, surprisingly, you will get almost some results. (Truncated back-propagation)

Reptile

Besides FOMAML, Reptile also aims to do the similar thing.



For each task i, the gradient $\frac{dl_i}{d\theta}$ is approximated as $\hat{\theta}^i - \theta$

Reptile

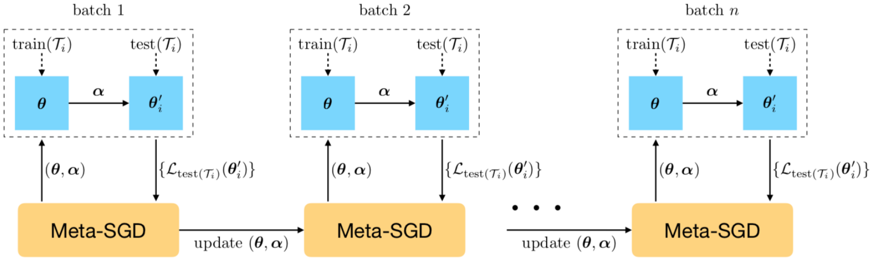
Algorithm 2 Reptile, batched version

Initialize ϕ
for iteration $= 1, 2, \dots$ **do**
    Sample tasks $\tau_1, \tau_2, \dots, \tau_n$
    for $i = 1, 2, \dots, n$ **do**
        Compute $W_i = \mathrm{SGD}(L_{\tau_i}, \phi, k)$
    end for
    Update $\phi \leftarrow \phi + \epsilon \frac{1}{k} \sum_{i=1}^{n} (W_i - \phi)$
end for

Vanilla MAML, FOMAML, Reptile have similar experiments, but the last two is more efficient.

Meta-SGD

In meta-sgd, learning rate for inner tasks are set to be parameters, this trick be helpful.



Tricks for MAML

Summary:

Idea: Automatically learn inner vector learning rate, tune outer learning rate (Li et al. Meta-SGD, Behl et al. AlphaMAML)

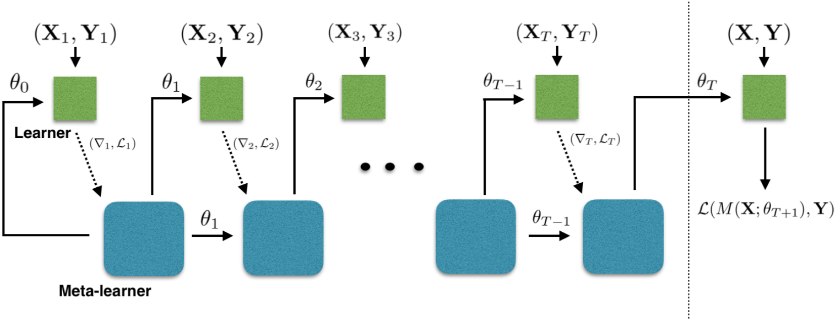
Idea: Optimize only a subset of the parameters in the inner loop (Zhou et al. DEML, Zintgraf et al. CAVIA)

Idea: Decouple inner learning rate, BN statistics per-step (Antoniou et al. MAML++)

Idea: Introduce context variables for increased expressive power. (Finn et al. bias transformation, Zintgraf et al. CAVIA)

Takeaway: a range of simple tricks that can help optimization significantly

Comparing with Black-box meta learning



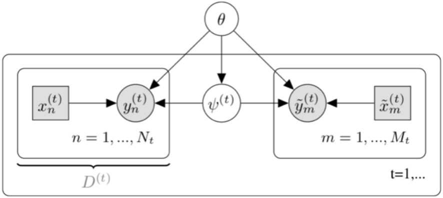
The main difference is modeling the task specific parameters.

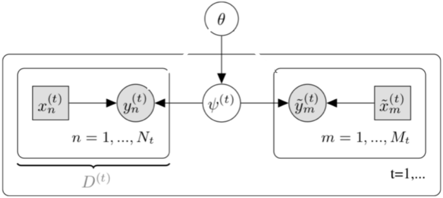
$$\text{where } \phi_i = \theta - \alpha \cancel{\nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}})}$$
$$f(\theta, \mathcal{D}_i^{\text{tr}}, \nabla_\theta \mathcal{L})$$

Content

Bayesian Meta Learning

What does structure even mean?





Two possible share graph. θ is a share prior knowledge for all tasks.

Probabilistic MAML

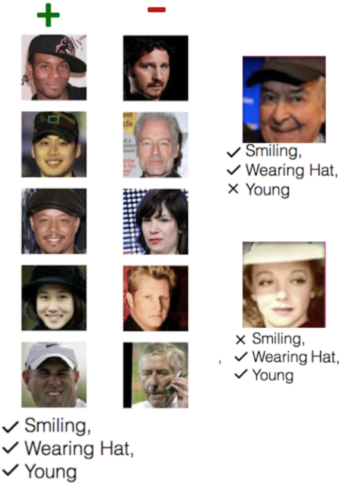
Motivation:
For the new task, you are requiring either no fast adaptation or prior knowledge embedding.

Question:
How to model uncertainty and model ambiguity?

Solution:
Bayesian meta learning

Probabilistic MAML

Task ambiguity example:



Probabilistic MAML

Given a new task, we can sample from $P(\phi^i|\theta, D^i_{tr})$, each sample can give one kind of explanation.
If we directly use standard variational inference to meta learning with multi-task setting, we have:

$$\max_{\phi} E_{\tau}[E_{q(\phi_i|D^i_{tr},\theta)}[\log P(Y_{te}|X_{te},\phi_i)] - KL(q(\phi_i|D^i_{tr},\theta)||p(\phi_i,\theta))] \quad (3)$$

Limitation: Can only represent Gaussian distributions $p(\phi_i|\theta)$

Probabilistic MAML

Perform inference on shared variables θ.

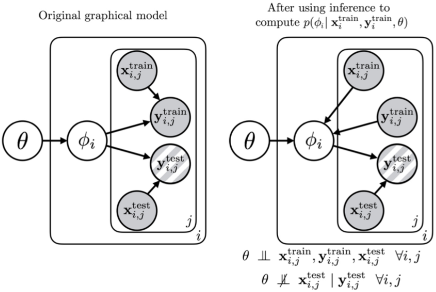
The target is to sample from $P(\Phi|D^i_{tr}(X), D^i_{tr}(Y))$.
Procedure:

- sample from θ,
- use ancestral sampling, we can sample $\phi_i, P(\phi^i|\theta, D^i_{tr}(X), D^i_{tr}(Y))$
- key assumption: $P(\phi^i|\theta, D^i_{tr}(X), D^i_{tr}(Y)) = \delta(\hat{\phi}^i)$, where:

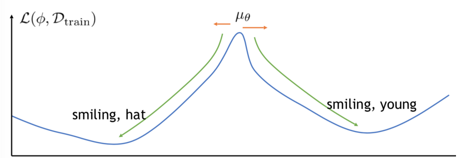
$$\hat{\phi}^i = \theta + \alpha\nabla_\theta log(D^i_{tr}(Y)|D^i_{tr}(Y), \theta)$$

Assumption: sampling from posterior distribution locally.

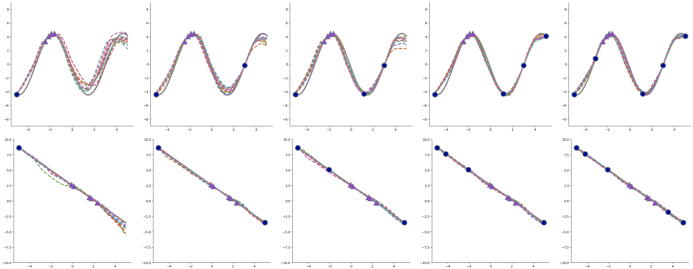
Graph used for generation and inference.



Sampling process:



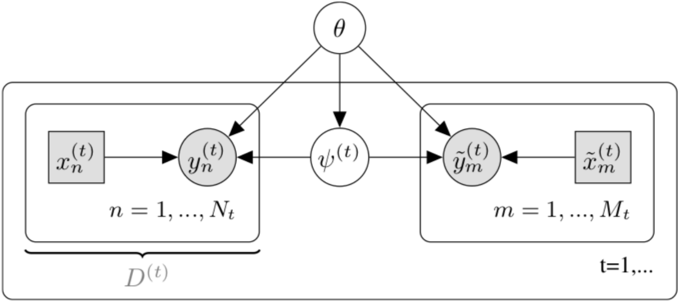
What does that mean?
The whole distribution of parameters works equally well, with each sample corresponding to one explanation.



This provides a perspective for interpretability of a meta learning.

Besides initialization

Besides using initialization as prior knowledge, what else?
For all tasks, using the sharing feature learning structure, whose parameter is θ.



In this case, the goal is to meta-learn accurate approximation to $p(\hat{y}_t|\hat{x}_t, \theta)$

Sharing feature vector

The introduced posterior distribution $q_\phi(\hat{y}|D,\hat{x})$

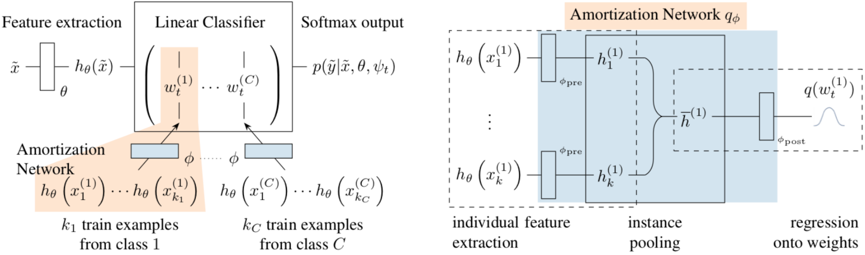
$$\phi^* = \arg\min_\phi \mathbb{E}_{P_D}[KL(p(\hat{y}|\hat{x},\theta)||q_\phi(\hat{y}|D,\hat{x}))] \tag{4}$$

$$= \arg\max_\phi \mathbb{E}_{(\hat{y},\hat{x},P_D)}[\log \int p(\hat{y}|\psi,\hat{x},\theta)q_\phi(\psi|D,\theta)d\phi] \tag{5}$$

The training could proceed as:

- select a task t at random
- sampling training data D^t
- form the posterior predictive $q_\phi(\psi|D^t,\theta)$
- compute the log likelihood on unseen data on D^t.

Diagram



During test time, your number of classes can be different with training.

Content

Bilevel Optimization

Bilevel optimization is common problem in machine learning community. It is defined as:

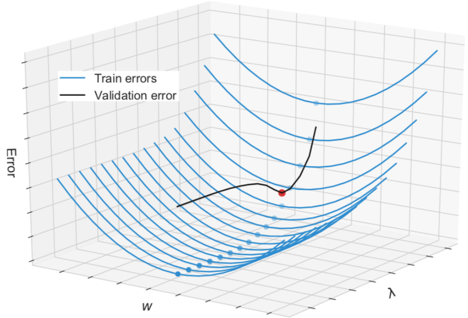
$$\min\{f(\lambda) : \lambda \in \Gamma\} \tag{6}$$

$$f(\lambda) = \inf\{E(w_\lambda, \lambda) : w_\lambda \in \arg\min L_\lambda(u)\}. \tag{7}$$

It is widely used in machine learning task:

- Hyperparameter selection
- Meta Learning

Hyperparameter selection



Meta Learning as bi-opt

For meta learning, this opt target λ is shared prior for across all tasks. The inner target is the optimal of task specific parameters.

Table 1. Links and naming conventions among different fields.

Bilevel programming	Hyperparameter optimization	Meta-learning
Inner variables	Parameters	Parameters of Ground models
Outer variables	Hyperparameters	Parameters of Meta-learner
Inner objective	Training error	Training errors on tasks (Eq. 3)
Outer objective	Validation error	Meta-training error (Eq. 4)

algorithm

Algorithm 1. Reverse-HG for Hyper-representation

Input: λ, current values of the hyperparameter, T number of iteration of GD, η ground learning rate, $\mathcal{B}$ mini-batch of episodes from $\mathcal{D}$
Output: Gradient of meta-training error w.r.t. λ on $\mathcal{B}$
for $j = 1$ **to** $|\mathcal{B}|$ **do**
  $w_0^j = 0$
  for $t = 1$ **to** T **do**
    $w_t^j \leftarrow w_{t-1} - \eta \nabla_w L^j(w_{t-1}^j, \lambda, D_{\text{tr}}^j)$
  $\alpha_T^j \leftarrow \nabla_w L^j(w_T^j, \lambda, D_{\text{val}})$
  $p^j \leftarrow \nabla_\lambda L^j(w_T^j, \lambda, D_{\text{val}})$
  for $t = T - 1$ **downto** 0 **do**
    $p^j \leftarrow p^j - \alpha_{t+1}^j \eta \nabla_\lambda \nabla_w L^j(w_t^j, \lambda, D_{\text{tr}}^j)$
    $\alpha_t^j \leftarrow \alpha_{t+1}^j \left[I - \eta \nabla_w \nabla_w L^j(w_t^j, \lambda, D_{\text{tr}}^j) \right]$
return $\sum_j p^j$