SAiDL Assignment -3 [PAPER REVIEW]

Aadetya Jaiswal

5th August 2019

1 Introduction

The paper focuses on implementing an efficient algorithm for **meta learning** (learning to learn).Meta-learning systems are usually trained by being exposed to a large number of tasks and are then tested in their ability to learn new tasks contrary to many standard machine learning techniques which involves sequential training and then testing on held out examples.
The algorithm proposed by author varies substantially from the previous meta learning algorithms. In order to understand the difference here is a brief info about some of them.

1.0.1 Recurrent Models

These approaches train a recurrent model, e.g. an LSTM, to take in the dataset sequentially and then process new inputs from the task. In an image classification setting, this might involve passing in the set of (image, label) pairs of a dataset sequentially, followed by new examples which must be classified.The meta-learner uses gradient descent, whereas the learner simply rolls out the recurrent network.**This method is less meta-efficient as it requires the learner to come up with learning strategy from scratch.**

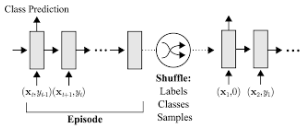


Figure 1: Recurrent model approach for inputs x_t and corresponding labels y_t, figure from Santoro et al. '16

1.0.2 Metric Learning

This approach involves learning a metric space in which learning is particularly efficient. The metric space is learned by training a Siamese network.Like the

previous approach, meta-learning is performed using gradient descent, whereas the learner corresponds to a comparison scheme, e.g. nearest neighbors, in the meta-learned metric space.**These approaches work quite well for few-shot classification, though they have yet to be demonstrated in other meta-learning domains such as regression or reinforcement learning.**

1.0.3 Learning Optimizers

The final approach is to learn an optimizer. In this method, there is one network (the meta-learner) which learns to update another network (the learner) so that the learner effectively learns the task. The meta-learner is typically a recurrent network so that it can remember how it previously updated the learner model. The meta-learner can be trained with reinforcement learning or supervised learning.Basically the algorithm supposes the learner model as an optimization process that should be learned.**This approach is not model and task agnostic and requires learning of initialization and learner model's parameters**

1.1 Model-Agnostic Meta-Learning (MAML)

The algorithm presented by the authors aims at optimizing the initial representation that can be effectively fine-tuned from a small number of examples.The meta-learner seeks to find an initialization that is not only useful for adapting to various problems, but also can be adapted quickly (in a small number of steps) and efficiently (using only a few examples). Like other meta-learning methods, MAML trains over a wide range of tasks. It trains for a representation that can be quickly adapted to a new task, via a few gradient steps.
MAML (unlike other meta learners) rather than trying to learn how to update learner model parameters update them using gradient descent therefore eliminating the need of a particular learner architecture
The biggest plus point of MAML is that it is agnostic to the form of the model and to the particular learning task,therefore making it extremely efficient at adaption for a variety of learning problems.

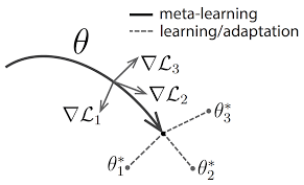


Figure 2: Diagram of the MAML approach.

Above is a visualization – suppose we are seeking to find a set of parameters θ that are highly adaptable. During the course of meta-learning (the bold line), MAML optimizes for a set of parameters such that when a gradient step is taken with respect to a particular task i (the gray lines), the parameters are close to the optimal parameters $\theta_i^\star$ for task i.

1.2 MAML Architecture

To understand the application of the algorithm let us suppose a model f_θ with parameters θ.MAML framework can be applied on a variety of learning problems, from classification to reinforcement learning so authors introduced a generic notion of a learning task given below.
T=(L$(x_1, a_1, ..., x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t)$, H)
This notation will be specialized when we review a variety of learning problems.

1.2.1 Notation

L$(x_1, a_1, ..., x_H, a_H)$ denotes task specific loss function.
q(x_1) denotes the probabilistic distribution of initial (observations/states)
$q(x_{t+1}|x_t, a_t)$ denotes the probability of transition from state x_t to x_{t+1} given action a_t takes place.
H denotes an episode length i.e the duration of interaction between agent and the environment until(observations/states) get initialized to original values.
p(T) denotes the probability distribution over which individual tasks are distributed.

Every model parameters can be broadly classified in two categories one which are broadly applicable to all tasks in p(T) and the other which are more task specific.
MAML algorithm aims at finding those parameters which are sensitive to changes in task,such that small changes will bring large improvements on the loss function.

1.3 MAML Algorithm

First, a task is sampled randomly from p(T). For each task model parameters becomes $\theta_i^{'}$.
Second,Using gradient descent parameters $\theta_i^{'}$ get updated.
$\theta_i^{'} = \theta$ - $\alpha \nabla_\theta L_{T_i}(f_\theta)$
Third,Model parameters are updated by evaluating the model's performance $(f_{\theta^i}^{'})$ on every task sampled across p(T).One way of doing this is updating θ for minimizing the summation of loss accumulated in all the tasks sampled.

$$min_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_\theta) = \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta - \alpha \nabla_\theta L_{T_i}(f_\theta)})$$

Therefore we can see that the training loss computed over all task is the loss function for the model parameters.Therefore meta-optimization is performed using SGD in the following proposed way

$$\theta = \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'})$$

Given below is the full implementation of the algorithm.**Making subtle amends in the loss function makes it ideal for other variety of learning problems**

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Figure 3: Generic form of MAML algorithm

1.4 MAML for K-shot Regression and Classification

Aim of K-shot regression is to predict the outputs of a continuous-valued function from only a few datapoints sampled from that function, after training on many functions with similar statistical properties.
Adapting MAML to this is quite easy:
H=1 as it is a single iteration process.
Replace x_t with x as the model accepts a single input and produces a single output, rather than a sequence of inputs and outputs.
q_i(x) gives K independent and identically distributed observations (K datapoints) x and $q(x_{t+1}|x_t, a_t)$ is not defined as there is no transition from one state to another.
Loss function is Mean squared error for K-shot regression and Cross entropy loss for K-shot classification.

Below is the implementation of the MAML algorithm for K-shot Regression and Classification

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: 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 2 or 3
11: **end while**

1.5 MAML for Reinforcement learning

Aim of meta-learning in RL is basically to quickly learn an optimal policy with only a small test experience. q_i(x) give the initial distribution of states
$q(x_{t+1}|x_t, a_t)$ denotes the probability of transition from state x_t to x_{t+1} given action a_t takes place.
L corresponds to the negative reward function R.
Therefore each task is an MDP(Markov Decision Process) **as given**x_t, a_t **transition to next state is conditionally independent of all the previous states and action(Markov property)**
H denotes the length of sample trajectory$(x_t, a_t.....x_H, a_H)$
Loss is the negative expected value of reward obtained.

$$L = -E_{x_t, a_t \sim f_\theta, q_i}[\sum_{t=1}^{H} R_i(x_t, a_t]$$

Below is the algorithm for RL:
After step 7 in the below given algorithm trajectories are sampled once again using the updated parameters .Therefore the test error accumulated across the sampled trajectories is used as the training loss for meta optimization.
The loss function here is expected reward function which is dynamic in nature and generally is not differentiable therefore we use policy

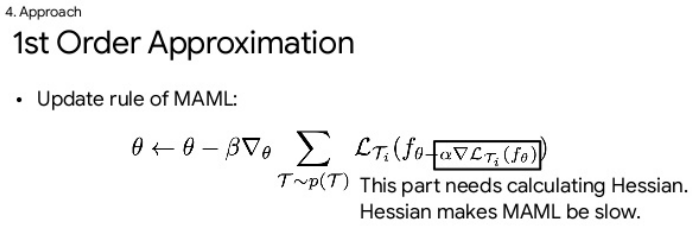
gradient methods to estimate the gradient both for the model gradient update(s) and the meta-optimization. Since policy gradients are an on-policy algorithm, each additional gradient step during the adaptation of f_θ requires new samples from the current policy $f_{\theta_i'}$.

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: 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
11: **end while**

1.6 Short comings of MAML

The MAML meta-gradient update involves a gradient through a gradient. Computationally, this requires an additional backward pass through f to compute Hessian matrix which is computationally very expensive.As a result MAML algorithm are painstakingly slow during model adaption process.



Authors suggest using first order approximation for computing Hessian Matrix with only a little deprecation in accuracy.
Update rule for MAML with First Order Approximation:

Regard δ as constant

$$\nabla_\theta L_{T_i}(f_\theta) \to \delta$$

$$\theta - \beta \nabla_\theta L_{T_i}(f_{\theta - \alpha\delta}) \to \theta$$

Using first order approximation significantly reduces the computation time.