Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Chelsea Finn, Pieter Abbeel, Sergey Levine
Presented by: Teymur Azayev

CTU in Prague

17 January 2019
Deep Learning

- Very powerful, expressive differentiable models.
- Flexibility is a double edged sword.
How do we reduce the amount of required samples?

Use Prior knowledge (not in a Bayesian sense). This can be in the form of:

- Model constraint
- Sampling strategy
- Update rule
- Loss function
- etc...
Learning to learn fast.
Essentially learning a prior from a distribution of tasks.
Several recent successful approaches:

- Model based meta-learning [Adam Santoro et al.], [Jx Wang et al.], [Yan Duan et al.]
- Metric meta-learning
  [Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov.],
  [Oriol Vinyals et al.]
- Optimization based meta-learning
  [Sachin Ravi and Hugo Larochelle],
  [Marcin Andrychowicz et al.],
Model Agnostic Metal Learning

Main idea: Learn a parameter initialization for a distribution of tasks, such that given a new task a small amount of examples (gradient updates) suffice.
Task $T_i \sim p(T)$ is defined as a tuple $(H_i, q_i, \mathcal{L}_{T_i})$ consisting of

- time horizon $H_i$ where for supervised learning $H_i = 1$
- initial state distribution $q_i(x_0)$ and state transition distribution $q_i(x_{t+1}|x_t)$
- Task loss function $\mathcal{L}_{T_i} \rightarrow \mathbb{R}$
- Task distribution $p$
Losses

\[ \theta_i^* \] is the optimal parameter for task \( T_i \)

\[ \theta_i' \] is the parameters obtained for task \( T_i \) after a single update

\[ 2) \text{ is the meta objective} \]
Algorithm 1 Model-Agnostic Meta-Learning

Require: \( p(\mathcal{T}) \): distribution over tasks
Require: \( \alpha, \beta \): step size hyperparameters

1: randomly initialize \( \theta \)
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
Reinforcement learning

\[ \mathcal{L}_{T_i}(f_\phi) = -\mathbb{E}_{x_t, a_t \sim f_\phi, q_{T_i}} \left[ \sum_{t=1}^{H} R_i(x_t, a_t) \right] \]
Reinforcement learning adaptation

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha, \beta$: step size hyperparameters

1: randomly initialize $\theta$
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 $\mathcal{D} = \{(x_1, a_1, \ldots, x_H)\}$ using $f_\theta$ 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 $\mathcal{D}'_i = \{(x_1, a_1, \ldots, x_H)\}$ 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
Sin wave regression

Tasks: Regressing randomly generated sin waves

- amplitudes ranging in [0.1, 5]
- phases [0, 2\pi]
- Sampled uniformly in range [−5, 5]
Sin wave regression

MAML, $K=5$

MAML, $K=10$

pretrained, $K=5$, step size=0.01

pretrained, $K=10$, step size=0.02
Classification tasks

Omniglot

- 20 instances of 1623 characters from 50 different alphabets
- Each instance drawn by a different person
- Randomly select 1200 characters for training and the remaining for testing

MiniImagenet

- 64 training classes, 12 validation classes, and 24 test classes
RL experiment

- Rllab benchmark suite, Mujoco simulator
- Gradient update are computed using policy gradient algorithms.
- Tasks are defined by the agents simply having slightly different goals
- Agents are expected to infer new goal from reward after receiving only 1 gradient update.
Conclusion

- Simple effective meta learning method
- Decent amount of follow up work [?], [?]
- Concept extendable to meta learning other parts of the training procedure
Thank you for your attention
References

Marcin Andrychowicz et al.
Learning to learn by gradient descent by gradient descent.
*NIPS 2016*

Yan Duan et al.
*2016*

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov.
Siamese Neural Networks for One-shot Image Recognition
*ICML 2015*

Zhenguo Li et al.
Meta-SGD: Learning to Learn quickly for few shot learning.
*2017*

Matthias Plappert et al.
Meta-SGD: Parameter Space Noise for Exploration
*2017*

Sachin Ravi and Hugo Larochelle
Meta-SGD: Optimization as a Model for Few-shot Learning
*ICLR 2017*

Adam Santoro et al.
One-shot Learning with Memory-Augmented Neural Networks
*ICML 2016*

Oriol Vinyals et al.
Matching Networks for One Shot Learning
*NIPS 2016*

Jx Wang et al.
Learning to Reinforcement Learn
*2016*

Risto Vuorio, Shao-Hua Sun, Hexiang Hu, Joseph J. Lim
Model-Agnostic Meta-Learning for Multimodal Task Distributions
*2017*

Ignasi Clavera, Anusha Nagabandi, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, Chelsea Finn
Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning
*2018*
References I