1710.09767 - Meta Learning Shared Hierarchies

- **Yunqiu Xu**
- Other reference:
  - OpenAI Blog: Learning a Hierarchy
  - Meta Learning Shared Hierarchies | Two Minute Papers
  - Videos for "Meta Learning Shared Hierarchies"
  - Paper Notes : Meta Learning Shared Hierarchies

---

1. Introduction

- Challenge: different tasks have different optimal policies $\rightarrow$ hard to generalize (or "transfer learning")
- Our work: MLSH
  - Hiarchical model similar to "options framework"
  - Contain a set of shared sub-policies (primitives) $\rightarrow$ these primitives are shared within a distribution of tasks
  - How to switch these sub-tasks : by using a task-specific master policy
  - For new tasks, we can just **learn master policy only** about how to switch the sub-policies correctly
- My brief understanding
  - Learn sub-policies as base models
  - For a new task, we can just learn to choose them correctly

2. Related Work

- Hierarchical RL:
  - Sutton et al. Between mdps and semi-mdps: A framework for temporal

abstraction in reinforcement learning : option framework, assumes that the options are given, some recent work seeks to learn them automatically
  - Florensa et al. Stochastic Neural Networks for Hierarchical Reinforcement Learning : Use stochastic NN to learn the span of skills, the sub-policies are defined according to information-maximizing statistics
  - Bacon et al. The option-critic architecture : end-to-end learning of hierarchy through the options framework
  - Feudal Network Dayan & Hinton 1993 , DeepMind 2017 : learn a decomposition of complicated tasks into sub-goals (Manager & Workers)
  - Their limitation: focus on single-task setting, doesn't account for multi-task structure
  - Our work uses **multi-task setting** to learn temporally extended primitives
- Meta learning: learning to learn
  - Dual et al. 2016 , Wang et al. 2016 : RNN as the entire learning process
  - Mishra et al. Meta-Learning with Temporal Convolutions : Utilize temporal convolutions rather than recurrency
  - MAML Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks : Treat the test error on sampled tasks as the training error of meta-learning process, fine-tune a shared policy by optimizing through a second gradient step
  - Difference:
    - Prior work : try to **learn as much as possible** in a few gradient updates
    - Our work : number of gradient updates can be large, but try to **learn as fast as possible**

3. Problem Statement

- Suppose there are a distribution of tasks, we aim to learn a policy that if we sample a new task, the policy can be easily adapted to it

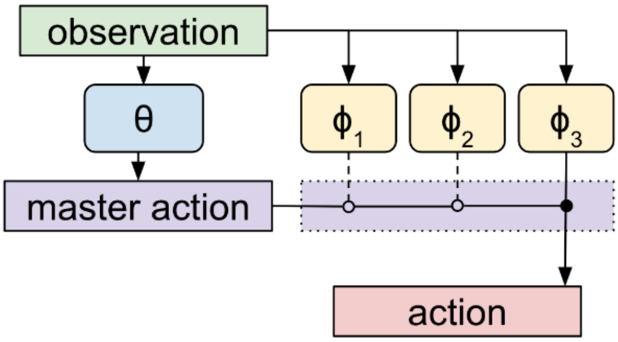


Figure 1: Structure of a hierarchical sub-policy agent. θ represents the master policy, which selects a sub-policy to be active. In the diagram, ϕ_3 is the active sub-policy, and actions are taken according to its output.

- Define a policy $\pi_{\phi,\theta}(a|s)$
  - ϕ :
    - A set of parameters **shared between all tasks**
    - $\phi = \{\phi_1, \phi_2, \ldots, \phi_K\}$
    - Each $\phi_k \rightarrow$ the parameters of a sub-policy $\pi_{\phi_k}(a|s)$
  - θ :
    - The parameters of master policy
    - Task-specific $\rightarrow$ zero or random initialized at the beginning
    - Choose a sub-task to activate for given timestep
- For a task M sampled from P_M
  - Randomly initialized θ and shared ϕ
  - Goal: learn θ , note that this is just the **objective for current task**
- The objective of meta-learning:
  - By learning training tasks, try to **find shared parameter psi** which can be generalize to a new MDP
  - Then for a new task, only learn θ

$$maximize_{\phi} E_{M \sim P_M, t=0,\ldots,T-1}[R]$$

- Why this is faster:
  - For a new task we only learn θ
  - As π_θ is only for choosing π_ϕ every N timesteps, while we do not need to learn

π_{ϕ}

- **It sees a problem as $1/N$ times as long**

4. Algorithm

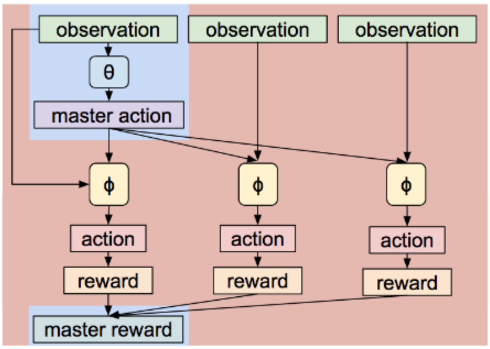
Algorithm 1 Meta Learning Shared Hierarchies

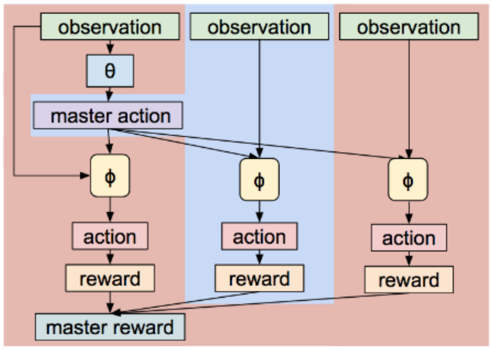
Initialize ϕ
repeat
  Initialize θ
  Sample task $M \sim P_M$
  for $w = 0, 1, ...W$ (warmup period) **do**
    Collect D timesteps of experience using $\pi_{\phi,\theta}$
    Update θ to maximize expected return from $1/N$ timescale viewpoint
  end for
  for $u = 0, 1, ...U$ (joint update period) **do**
    Collect D timesteps of experience using $\pi_{\phi,\theta}$
    Update θ to maximize expected return from $1/N$ timescale viewpoint
    Update ϕ to maximize expected return from full timescale viewpoint
  end for
until convergence

- The goal of meta-training (policy update) is to learn ϕ which can be shared for all tasks
- The goal of learning a single task is to learn $\theta \rightarrow$ choose ϕ correctly
- At the very beginning we random intialize ϕ, and for each new task, we random initialize $\theta \rightarrow$ 对每个新任务都要重设$\boldsymbol{\theta}$
- Warm-up period:
  - **Goal : try to optimize θ to nearly optimal**
  - In this period, we hold phi fixed
  - For each iteration sample D timesteps of experience
  - For each 1/N timescale, consider a sub-policy as an "action"
  - 注意这里 1/N timescale 的意思就是每间隔 $\frac{1}{N} * total\_time$ 选一个动作
- Joint update period:
  - **Both θ and ϕ are updated**
  - For each iteration, collect experience and optimize $\theta \rightarrow$ **same as warm-up**
  - Update ϕ : reuse these D samples, but viewed via sub-policy

- Treat the master policy as an extension of the environment $\rightarrow$ a discrete portion of observation
- For each N-timestep slice of experience, we only update the parameters of the sub-policy that had been activated by master policy

- Example 1:
  - Left : warm-up, update master policy
    - Here we hold sub-policies fixed, the available action is to choose one of them at each time slice
  - Right : train a sub-policy
    - During joint-updating we also need to update θ first
    - Then we use the same data to update ϕ
    - Note that currently only the blue sub-policy is chosen by master $\rightarrow$ only update this sub-policy





- Exemple 2: cooking
  - 我们可以认为 θ 是做某一道菜的基本流程, 而 ϕ 为烹饪技巧如清洗, 切菜, 腌制等
  - 为什么 warm-up : 对于一道菜我们需要在大致了解其制作流程 (nearly optimal θ)的前提下才能着手优化子步骤
  - 因此在 warm-up 环节, 我们努力用自己已有的烹饪技巧来学习烹饪流程, 这里的动作即子任务:
    - 0-5 min : 选择 '清洗'
    - 5-10 min : 选择 '切菜'
    - 10-15 min : 选择 '腌制'
  - 大致了解这道菜做法之后, 我们进入 joint-update 环节, 对每个循环, 首先还是优化制作流程 θ, 然后优化子任务 ϕ
  - 注意只有当前被选择的子任务会被优化, 例如在 timestamp 0-5 min π_{theta} 选择的子

任务是'清洗', 在这5分钟里会被优化的只有清洗, 其他子任务的 ϕ 保持不变
  - meta-training 的目标为通过学习一系列菜品的制作方法, 训练完备的烹饪技巧 $\rightarrow$ 对于一道新菜 (meta-testing), 我们需要学习的只是如何正确组合这些烹饪技巧
- Why our method is faster:
  - Traditional meta-learning : optimize reward (master policy θ) over an entire inner loop
  - MLSH :
    - **注意这里引入假设 1 : 经过warm-up可以学习到 optimal θ**
    - θ is updated per N-timesteps, only a much smaller task ϕ to update over entire inner loop
    - θ is learned to nearly optimal in warm-up, so it will not take much time to reach optimal in joint-update
  - 为什么限制joint-update的次数:
    - **这里引入假设 2: 在 joint-update 时因为 θ 已经是优化的了, 即使更新也和原来区别不大**
    - 因此 joint-update 的主要作用是更新 ϕ, 而子任务相对容易学习, 在参数上微调就好
    - 因此我们不需要在joint-update上花费太多时间, 只需要固定训练循环数就好
    - 这里我的理解是因为一开始 ϕ 并不够robust, 因此在warm-up过程中得到的 θ 只能是近似优化的, 然后在 joint-update 过程中进一步优化, 并优化 ϕ

5. Experiment

- More results can be seen in https://sites.google.com/site/mlshsupplementals/
- Task 1: 2D moving bandits

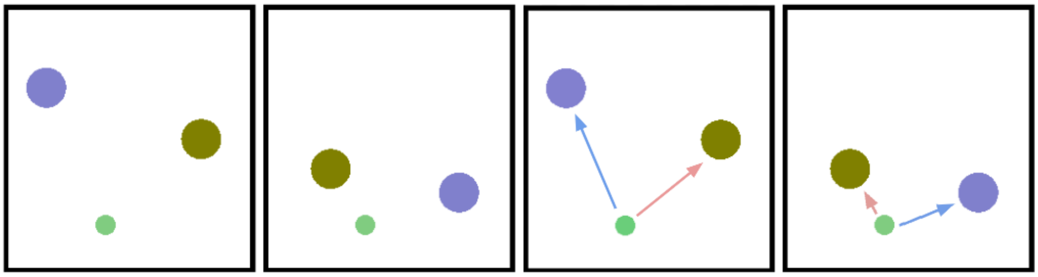


Figure 3: Sampled tasks from 2D moving bandits. Small green dot represents the agent, while blue and yellow dots represent potential goal points. Right: Blue/red arrows correspond to movements when taking sub-policies 1 and 2 respectively.

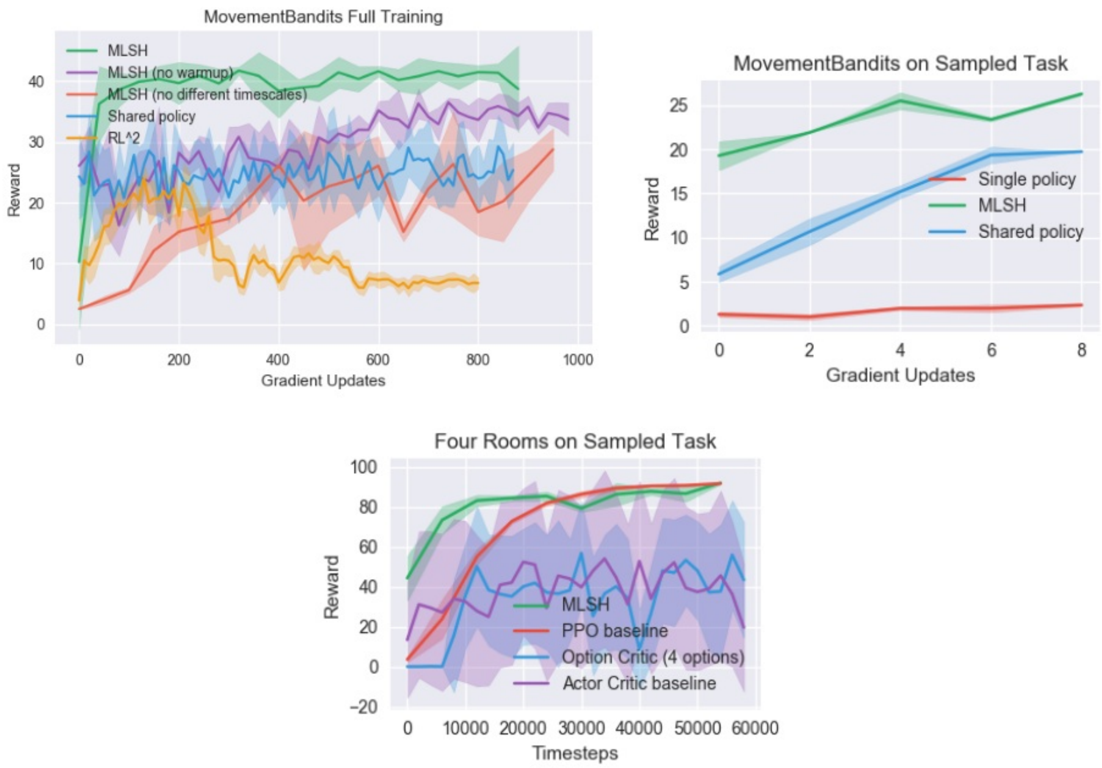


Figure 4: Learning curves for 2D Moving Bandits and Four Rooms

- Task 2: simulated walk

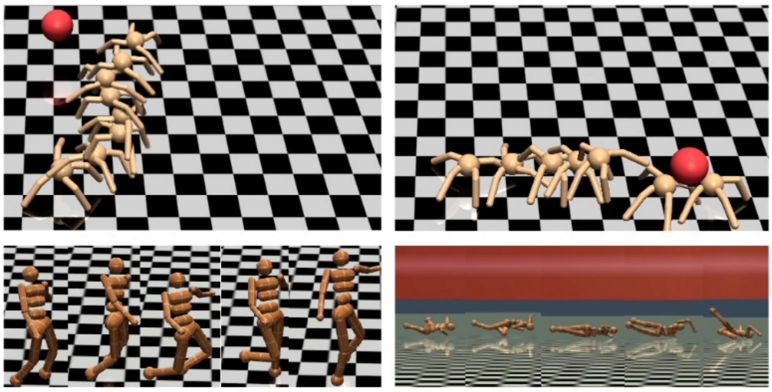


Figure 5: Top: Ant Twowalk. Ant must maneuver towards red goal point, either towards the top or towards the right. Bottom Left: Walking. Humanoid must move horizontally while maintaining an upright stance. Bottom Right: Crawling. Humanoid must move horizontally while a height-limiting obstacle is present.

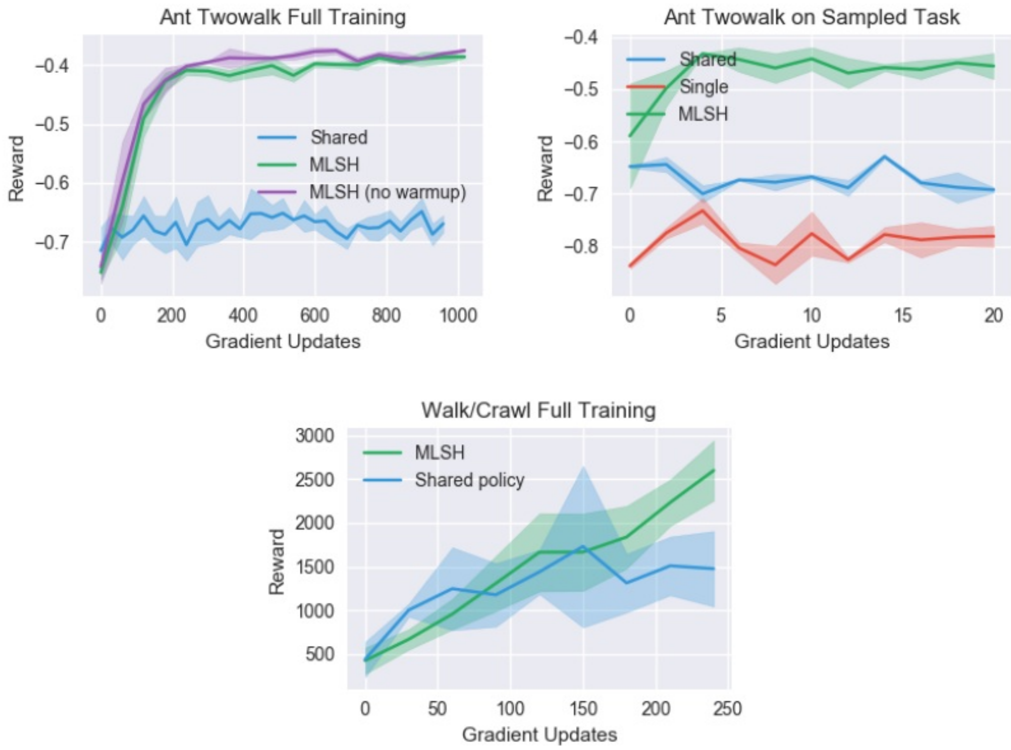


Figure 7: Learning curves for Twowalk and Walk/Crawl tasks

6. Conclusion

- This work combines meta-learning with hierarchical RL, try to learn faster over a large number of gradient updates
- Questions:

- 1. 对比前人工作, 怎样理解其创新点 "multi-task" ?
    - 我的理解为学到的这些子任务可以加速master task的学习
    - 因此这里的multi-task是指快速适应新任务?
  - 1. 在meta-testing的时候, 还需要再更新 ϕ 么, 还是只要更新 θ ?
- For each task, the master policy θ is reset and learned to be optimal with fixed sub-policies ϕ, then nearly-optimal master policy will be treated as an extension of environment and ϕ will be updated