Meta-learning

Yunqiu Xu

Readings

1. Model-agnostic Meta-learning
2. MAML for One-shot Imitation Learning
3. Meta Learning Shared Hierarchies
4. MAML for Non-stationary Task

MAML : Model-agnostic Meta-learning

MAML : Model-agnostic Meta-learning

+ Meta-learning: train a model with some learning tasks, then it can solve new tasks with only a few samples
+ MAML can be treated as an initialization method to get pretrained “base model” that is easy to fine-tune to new tasks
+ Compare with another recent work (Ravi & Larochelle, 2017) : do not need additional parameters, and “model agnostic”

MAML: Problem Setup

General notion of task $T = \{L(x_1, a_1, \ldots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$:

- $L \to R$: loss function
- $q(x_1)$: distribution over initial observations
- $q(x_{t+1}|x_t, a_t)$: transiion distribution
  - 对于监督学习, 不存在这个分布 : $H = 1$
  - 对于强化学习, $q(x_{t+1}|x_t, a_t)$ 代表某时间点观察值的分布, e.g. 初始观察值后观察值取自$q(x_2|x_1, a_1)$
- H :
  - Episode length, model may generate samples of length H by choosing an output a_t at each time t
  - For supervised learning, $H = 1$ and loss function $L(x_1, a_1)$ could be MSE or cross entropy

MAML : Algorithm

Algorithm 1 Model-Agnostic Meta-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**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while

MAML: K-shot Training and Testing

- Train model f:
  - Sample a new task T_i from $p(T)$ (training taskset)
  - Learn T_i :
    - Train model with K samples drawn from q_i
    - Get feedback L_{T_i} from T_i
  - Test on new samples from T_i and get test error
  - Improve model f : treat the test error on sampled tasks T_i as the training error of meta-learning process
- Test meta-learning:
  - Sample new task from $p(T)$ (testing taskset), try to adapt f to this new task
  - Learn the model with K samples
  - Treat the performance as "meta-performance"

MAML : An Example of Inner-loop

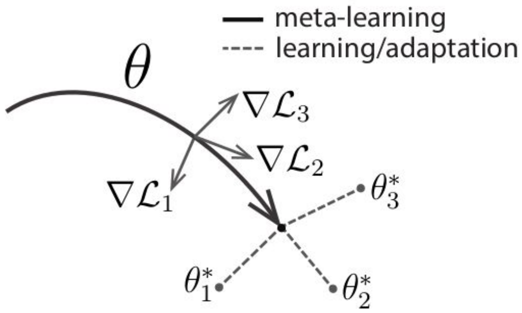


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

+ 假定我们在当前内循环需要训练T2, 更新后的参数 theta_2’ 和原来相比会稍稍 “上移”
+ 同理, 我们训练T1和T3后得到的theta’和原来相比方向也会有一定调整
+ 因此在meta-loop中我们用所有的theta’来更新theta, 得到一个比较general的方向

MAML: SL and RL

Algorithm 2 MAML for Few-Shot Supervised 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 datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
  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 2 or 3
end while

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

MAML: Experiment

Regression task: 模拟sine曲线

Classification task: few-shot image recognition

RL task: 2D Navigation, MuJoCo Simulation

MAML: Regression Result

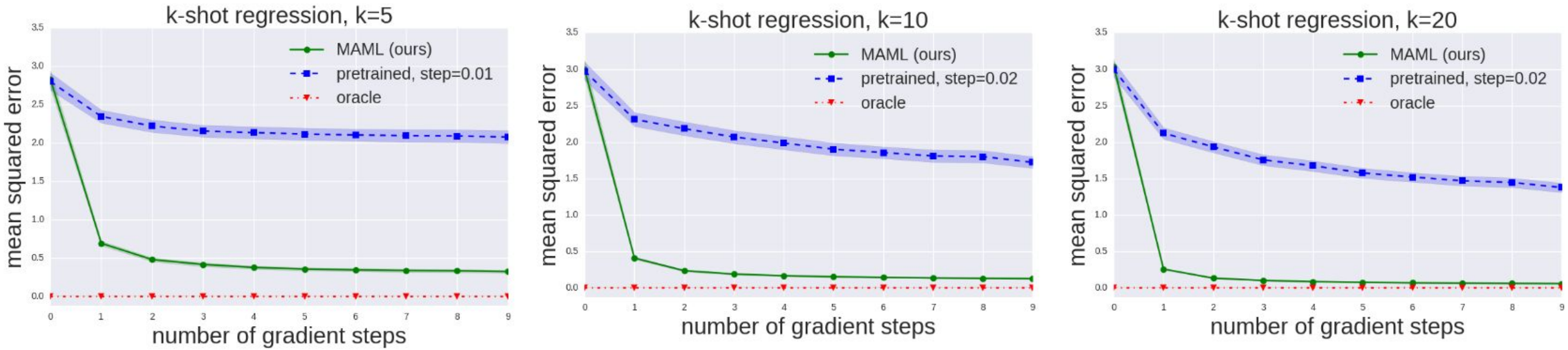


Figure 6. Quantitative sinusoid regression results showing test-time learning curves with varying numbers of K test-time samples. Each gradient step is computed using the same K examples. Note that MAML continues to improve with additional gradient steps without overfitting to the extremely small dataset during meta-testing, and achieves a loss that is substantially lower than the baseline fine-tuning approach.

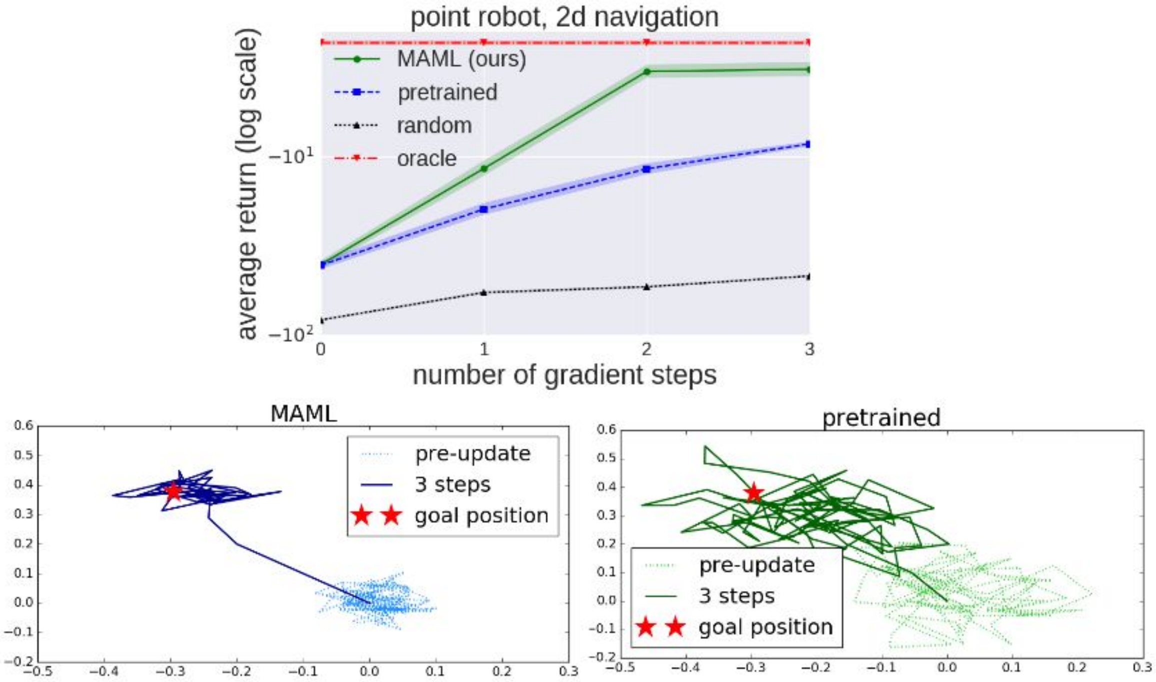
- When all the points are in one half, the model can still infer the shape of the other half ⟶ model the periodic nature
- Regression可以用很少的样本/循环进行finetune, 不会overfitting
- 因为效果本身就很不错了, 增加迭代次数并没有明显提升

MAML: Classification Result

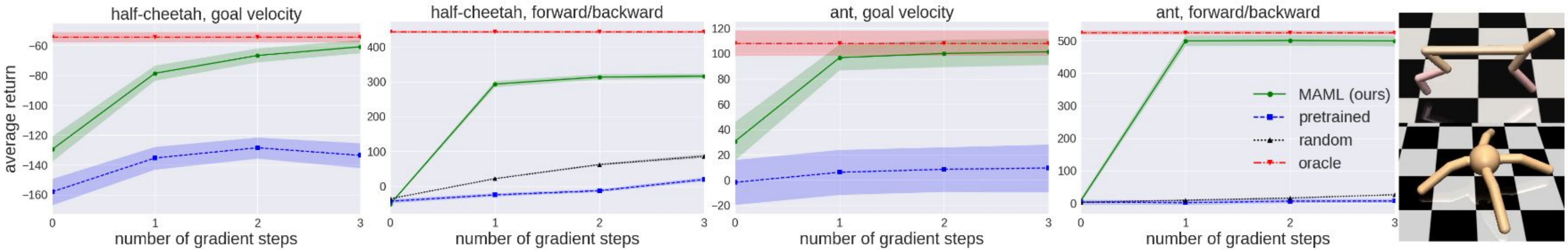
Omniglot (Lake et al., 2011)	5-way Accuracy 1-shot	5-way Accuracy 5-shot	20-way Accuracy 1-shot	20-way Accuracy 5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	**89.7** $\pm$ **1.1%**	**97.5** $\pm$ **0.6%**	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	**98.7** $\pm$ **0.4%**	**99.9** $\pm$ **0.1%**	**95.8** $\pm$ **0.3%**	**98.9** $\pm$ **0.2%**

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy 1-shot	5-way Accuracy 5-shot
fine-tuning baseline	28.86 $\pm$ 0.54%	49.79 $\pm$ 0.79%
nearest neighbor baseline	41.08 $\pm$ 0.70%	51.04 $\pm$ 0.65%
matching nets (Vinyals et al., 2016)	43.56 $\pm$ 0.84%	55.31 $\pm$ 0.73%
meta-learner LSTM (Ravi & Larochelle, 2017)	43.44 $\pm$ 0.77%	60.60 $\pm$ 0.71%
MAML, first order approx. (ours)	**48.07** $\pm$ **1.75%**	**63.15** $\pm$ **0.91%**
MAML (ours)	**48.70** $\pm$ **1.84%**	**63.11** $\pm$ **0.92%**

MAML: RL Result



MAML: RL Result



MIL : MAML for One-shot Imitation Learning

MIL : MAML for One-shot Imitation Learning

+ Generalize meta-learning technique to apply to imitation learning
+ Raw visual inputs
+ Reuse past experience to train the "base model", then adapt it to new task with only a single demonstration

MIL: Problem Setup

- Goal: learn a policy that can quickly adapt to new tasks from a single demonstration of that task
- Task:

Each imitation task $T_i = \{\tau = \{o_1, a_1, \ldots, o_T, a_T\} \sim \pi_i^*, L(a_{1:T}, \hat{a}_{1:T}), T\}$

- τ : a demonstration generated by policy π_i^*
- $L(a_1, \ldots, a_T, \hat{a}_1, \ldots, \hat{a}_T) \to R$: loss function to give feedback

MIL: Algorithm

- o_t is the observation at time t, i.e. an image, while a_t is the action
- For demonstration trajectory τ, we use MSE to compute loss:

$$L_{T_i}(f_\phi) = \sum_{\tau_j \sim T_i} \sum_t ||f_\phi(o_t^{(j)}) - a_t^{(j)}||_2^2 \qquad (2)$$

Algorithm 1 Meta-Imitation Learning with MAML

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

```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        Sample demonstration τ = {o_1, a_1, ...o_T, a_T} from T_i
        Evaluate ∇_θ L_{T_i}(f_θ) using τ and L_{T_i} in Equation (2)
        Compute adapted parameters with gradient descent: θ'_i = θ − α∇_θ L_{T_i}(f_θ)
        Sample demonstration τ'_i = {o'_1, a'_1, ...o'_T, a'_T} from T_i for the meta-update
    end for
    Update θ ← θ − β∇_θ Σ_{T_i~p(T)} L_{T_i}(f_{θ'_i}) using each τ'_i and L_{T_i} in Equation 2
end while
return parameters θ that can be quickly adapted to new tasks through imitation.
```

MIL: Algorithm

- Meta-training:
  - Assume each training task has at least 2 demonstrations, thus we can sample a set of tasks with **two demonstrations per task**
  - For each task T_i, train θ'_i with its one demonstration $\tau_i \rightarrow$ inner loop of meta-learning
  - Use another demonstration τ'_i to "test" θ'_i , i.e. check the mse of predicted actions and demonstration actions
  - Then we can update θ according to the gradient of meta-objective
  - As we get a series of $\theta'_i s$ and their testing error, we can update θ
  - Finally we can get trained parameters θ for meta-learner

- Meta-testing:
  - Sample a new task T and its one demonstration
  - This task can involve new goals or manipulating new, previously unseen objects
  - Then we can adapt θ to this task

MIL: Network Architecture

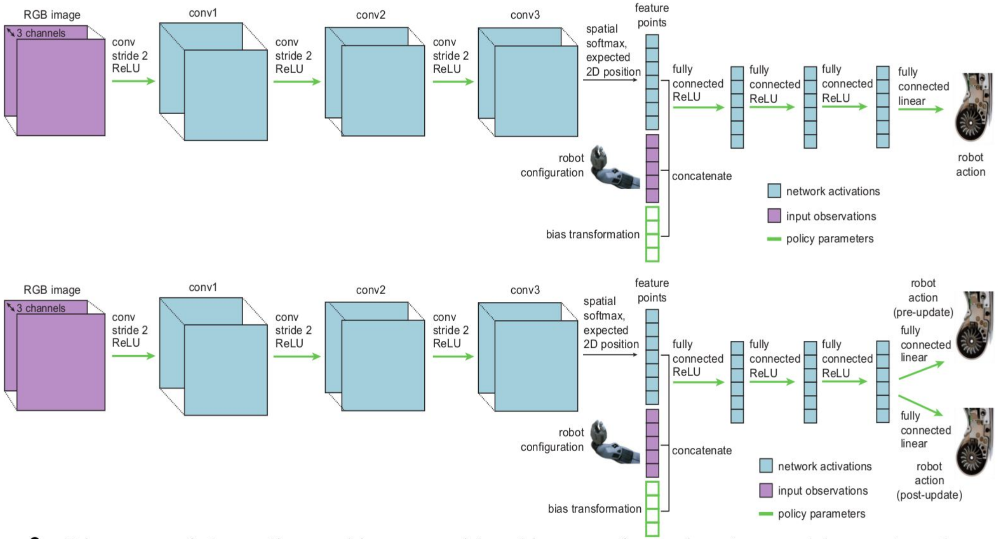


Figure 2: Diagrams of the policy architecture with a bias transformation (top and bottom) and two heads (bottom). The green arrows and boxes indicate weights that are part of the meta-learned policy parameters θ.

Two-head structure : more flexibility during adapting

- Modification : parameters of final layers are not shared, forming two heads
  - Change loss function as:

$$L_{T_i}(f_\phi) = \sum_{\tau_j \sim T_i} \sum_t \|W y_t^{(j)} + b - a_t^{(j)}\|_2^2 \qquad (3)$$

    - $y_t^{(j)}$: post-synamptic activations of the last hidden layer
    - W, b : weights and bias for last layer
  - Then the meta-objective is about θ, W, b

$$\min_{\theta, W, b} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'}) = \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta - \alpha \nabla_\theta L_{T_i}(f_\theta)}) \qquad (4)$$

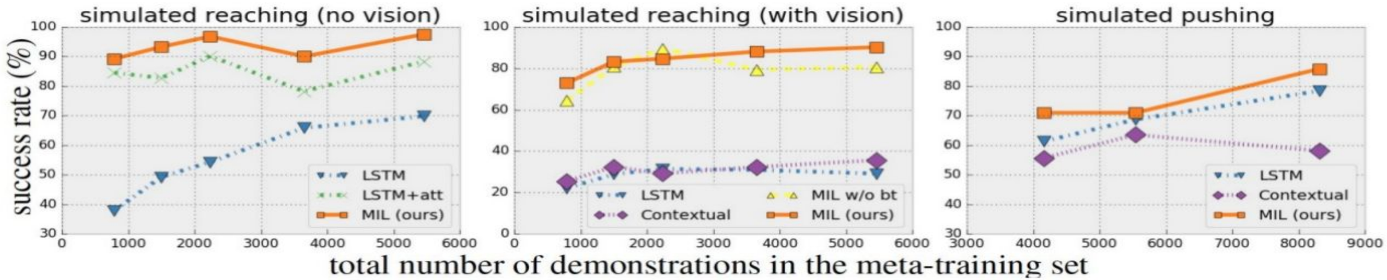
MIL: Network Architecture

- **Layer normalization** after each layer
  - Data within a demonstration trajectory is highly correlated across time
  - Thus BN was not effective
- Bias transformation $\rightarrow$ improve the performance of meta-learning
  - Concatenate a vector of parameters to a hidden layer of post-synaptic activations
  - Thus vector is treated as same as other parameters during meta-learning and final testing

$$y = Wx + b \quad \rightarrow \quad y = W_1 x + W_2 z + b$$

MIL: Experiments

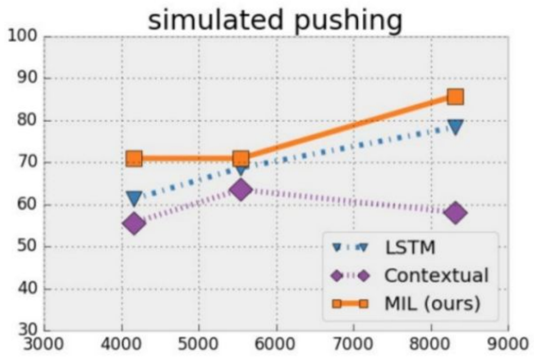
Simulated Reaching



Simulated Pushing

method		video+state +action	video +state	video
LSTM	1-shot	78.38%	37.61%	34.23%
contextual	1-shot	n/a	58.11%	56.98%
MIL (ours)	1-shot	**85.81%**	**72.52%**	**66.44%**
LSTM	5-shot	83.11%	39.64%	31.98%
contextual	5-shot	n/a	64.64%	59.01%
MIL (ours)	5-shot	**88.75%**	**78.15%**	**70.50%**

Table 1: One-shot and 5-shot simulating pushing success rate with varying demonstration information provided at test-time. MIL can more successfully learn from a demonstration without actions and without robot state and actions than LSTM and contextual policies.



MLSH : Meta-learning Shared Hierarchies

MLSH : Meta-learning Shared Hierarchies

- 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

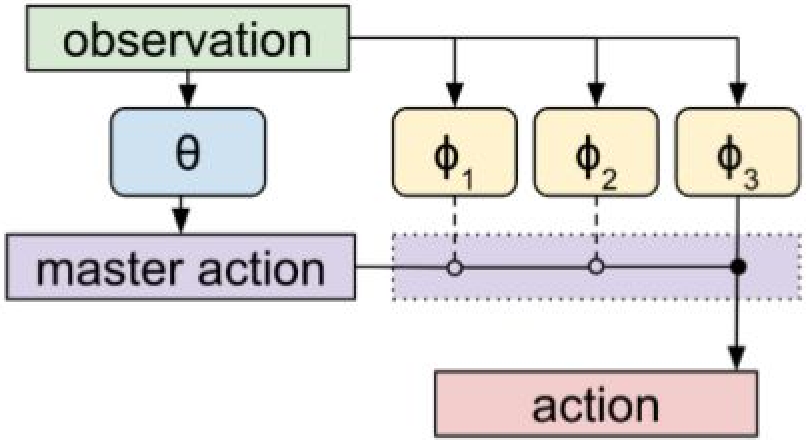
MLSH : Problem Setup

- 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

MLSH : Problem Setup

- For a task $\boldsymbol{M}$ sampled from $\boldsymbol{P_M}$
  - Randomly initialized $\boldsymbol{\theta}$ and shared $\boldsymbol{\phi}$
  - Goal: learn $\boldsymbol{\theta}$, note that this is just the **objective for current task**
- The objective of meta-learning:
  - By learning training tasks, try to **find shared parameter $\boldsymbol{psi}$** which can be generalize to a new MDP
  - Then for a new task, only learn $\boldsymbol{\theta}$

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



MLSH : Algorithm

Algorithm 1 Meta Learning Shared Hierarchies

```
Initialize φ
repeat
  Initialize θ
  Sample task M ~ P_M
  for w = 0, 1, ...W  (warmup period) do
    Collect D timesteps of experience using π_{φ,θ}
    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 π_{φ,θ}
    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 $\boldsymbol{\phi}$ which can be shared for all tasks
- The goal of learning a single task is to learn $\boldsymbol{\theta} \rightarrow$ choose $\boldsymbol{\phi}$ correctly
- At the very beginning we random intialize $\boldsymbol{\phi}$, and for each new task, we random initialize $\boldsymbol{\theta} \rightarrow$ 对每个新任务都要重设$\boldsymbol{\theta}$

MLSH : Warm-up Period

for $w = 0, 1, \ldots W$ (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

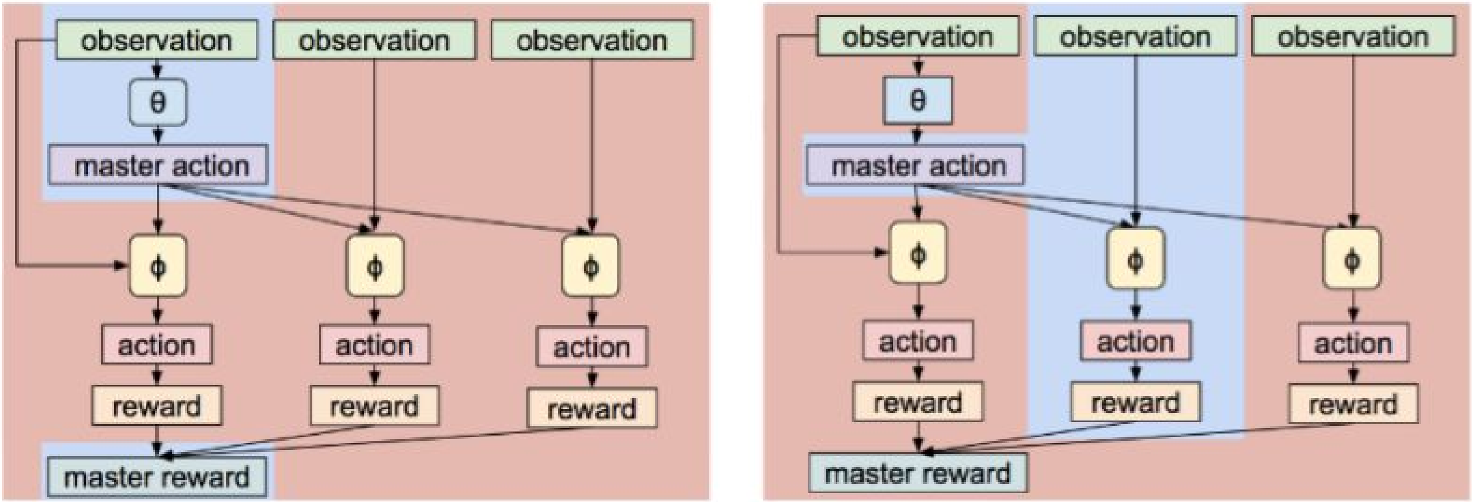
- **Goal : try to optimize $\boldsymbol{\theta}$ to nearly optimal**
- In this period, we hold $\boldsymbol{phi}$ fixed
- For each iteration sample D timesteps of experience
- For each 1/N timescale, consider a sub-policy as an "action"
- 注意这里 1/N timescale 的意思就是每间隔 $\frac{1}{N} * total\_time$ 选一个动作

MLSH : Joint-update Period

for $u = 0, 1, \ldots.U$ (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

- **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

MLSH : An Example

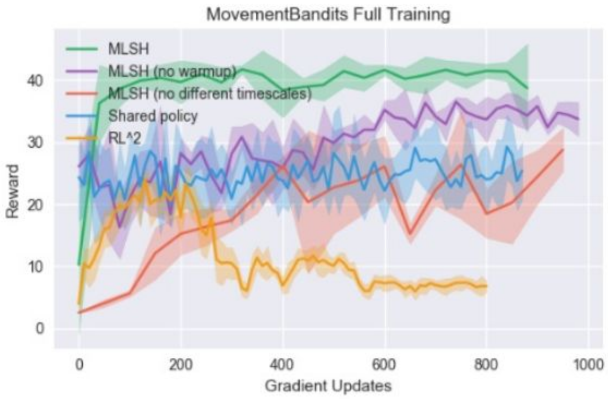


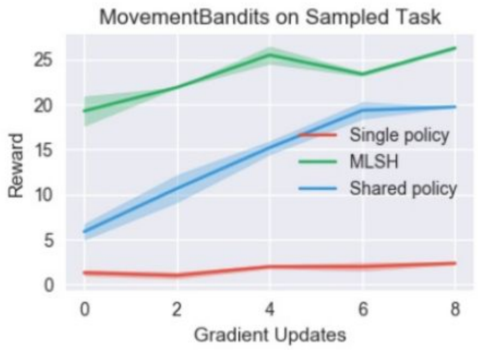
MLSH : Why Faster

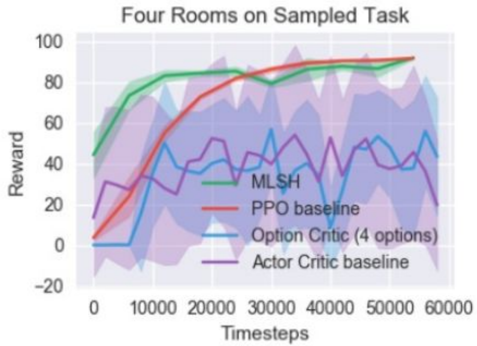
- Traditional meta-learning : optimize reward (master policy $\boldsymbol{\theta}$) over an entire inner loop
- MLSH :
  - **注意这里引入假设 1 : 经过warm-up可以学习到 optimal $\boldsymbol{\theta}$**
  - $\boldsymbol{\theta}$ is updated per N-timesteps, only a much smaller task $\boldsymbol{\phi}$ to update over entire inner loop
  - $\boldsymbol{\theta}$ 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 $\boldsymbol{\theta}$ 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 $\boldsymbol{\phi}$, 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 $\boldsymbol{\phi}$ 并不够robust, 因此在warm-up过程中得到的 $\boldsymbol{\theta}$ 只能是近似优化的, 然后在 joint-update 过程中进一步优化, 并优化 $\boldsymbol{\phi}$

MLSH : Experiments

2D moving bandits

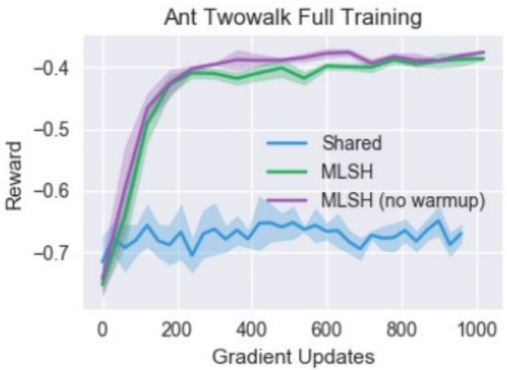


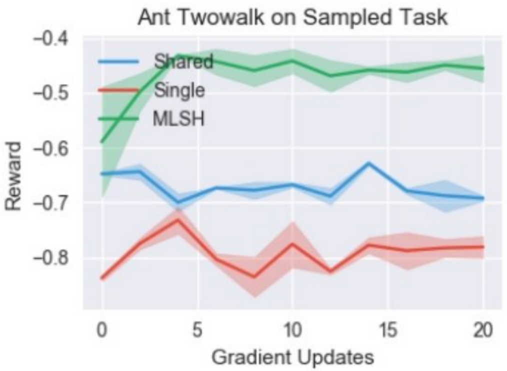


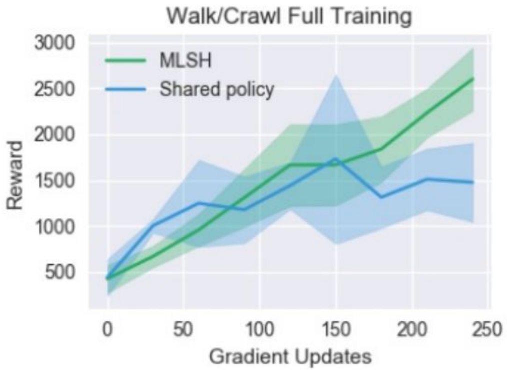


MLSH : Experiments

Simulated Walk







MAML for Non-stationary Task

+ Challenge: non-stationary → changing environment, multiple actors
+ Insight: non-stationary task can be seen as a sequence of stationary tasks → multi-task problem
+ Our work: gradient-based meta-learning algorithm for adaptation in dynamically changing and adversarial scenarios
+ Goal: handle continuous adaptation with few-shot

Probabilistic View of MAML

- Task $\boldsymbol{T}$, trajectories $\boldsymbol{\tau}$ and policies $\boldsymbol{\pi_\theta}$ are random variables, $\boldsymbol{phi}$ is generated from conditional distribution $\boldsymbol{P_T(\phi|\theta, \tau_{1:k})}$
- Inner loop update: equivalent to assuming the delta distribution

$$\boldsymbol{P_T(\phi|\theta, \tau_{1:k}) := \delta(\theta - \alpha \nabla_\theta \frac{1}{K} \sum_{k=1}^{K} L(\tau_k))}$$

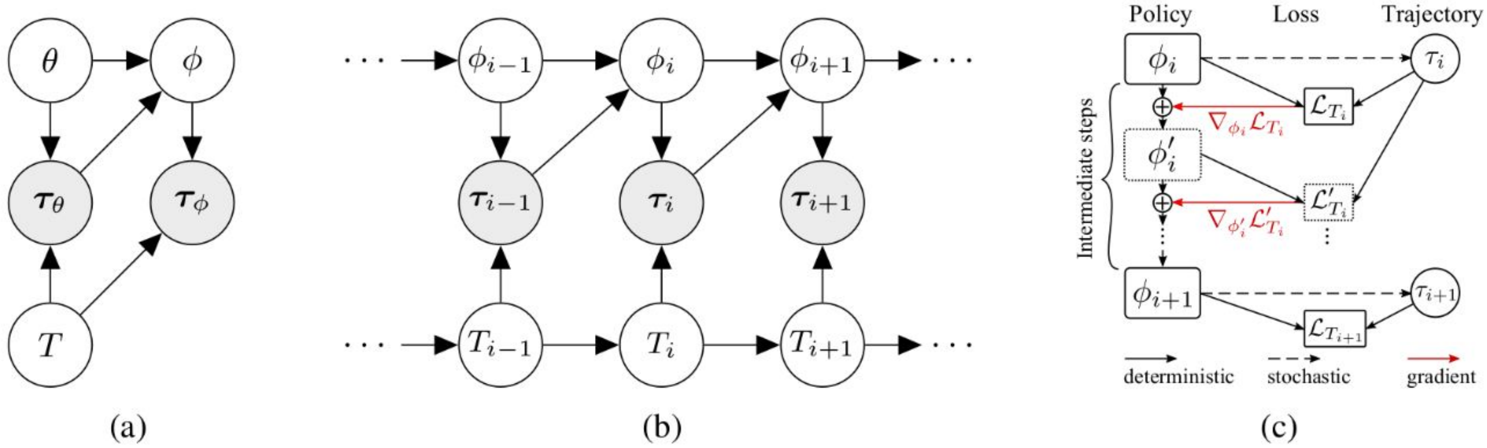
- Optimize meta-objective: PG where the gradient of $\boldsymbol{R_T(\theta)}$

$$\nabla_\theta \mathcal{R}_T(\theta) = \mathbb{E}_{\substack{\boldsymbol{\tau}_\theta^{1:K} \sim P_T(\boldsymbol{\tau}|\theta) \\ \boldsymbol{\tau}_\phi \sim P_T(\boldsymbol{\tau}|\phi)}} \left[\mathcal{L}_T(\boldsymbol{\tau}_\phi) \left[\nabla_\theta \log \pi_\phi(\boldsymbol{\tau}_\phi) + \nabla_\theta \sum_{k=1}^{K} \log \pi_\theta(\boldsymbol{\tau}_\theta^k) \right] \right] \tag{4}$$

Continuous adaptation via meta-learning

- $D(T)$ is defined by the environment changes, and the tasks become sequentially dependent
- Our goal:
  - Find the dependence between consecutive tasks
  - Meta-learn a rule to minimize total expected loss during interacting

Continuous adaptation via meta-learning



- Here is probabilistic graph of MAML
  - (a) MAML in a multi-task RL setting
  - (b) Extend to continuous adaptation
    - Policy and trajectories at a previous step are used to construct a new policy for the current step, i.e. $\phi_i, \tau_i \rightarrow \phi_{i+1}$
  - (c) Computation graph for the meta-update from ϕ_i to ϕ_{i+1}
    - The model is optimized via truncated backpropagation through time starting from $L_{T_{i+1}}$

Continuous adaptation via meta-learning

- From (b) we represent nonstationary as a sequence of stationary tasks, then the goal is try to minimize the expected loss of $\boldsymbol{L}$ tasks, where $\boldsymbol{L}$ is a given number

$$\min_{\theta} \mathbb{E}_{\mathcal{P}(T_0), \mathcal{P}(T_{i+1}|T_i)} \left[\sum_{i=1}^{L} \mathcal{R}_{T_i, T_{i+1}}(\theta) \right] \tag{5}$$

- Expected loss on a pair of consecutive tasks

$$\mathcal{R}_{T_i, T_{i+1}}(\theta) := \mathbb{E}_{\boldsymbol{\tau}_{i,\theta}^{1:K} \sim P_{T_i}(\boldsymbol{\tau}|\theta)} \left[\mathbb{E}_{\boldsymbol{\tau}_{i+1,\phi} \sim P_{T_{i+1}}(\boldsymbol{\tau}|\phi)} \left[\mathcal{L}_{T_{i+1}}(\boldsymbol{\tau}_{i+1,\phi}) \mid \boldsymbol{\tau}_{i,\theta}^{1:K}, \theta \right] \right] \tag{6}$$

The principal difference between the loss in (3) and (6) is that trajectories $\boldsymbol{\tau}_{i,\theta}^{1:K}$ come from the current task, T_i, and are used to construct a policy, π_ϕ, that is good for the upcoming task, T_{i+1}.

Continuous adaptation via meta-learning

- To construct the parameters of policy for task $\boldsymbol{T_{i+1}}$, we start from $\boldsymbol{\theta}$, note that here we can also change inner step size $\boldsymbol{\alpha}$

$$\begin{aligned} \phi_i^0 &:= \theta, \quad \boldsymbol{\tau}_\theta^{1:K} \sim P_{T_i}(\boldsymbol{\tau} \mid \theta), \\ \phi_i^m &:= \phi_i^{m-1} - \alpha_m \nabla_{\phi_i^{m-1}} \mathcal{L}_{T_i}\left(\boldsymbol{\tau}_{i,\phi_i^{m-1}}^{1:K}\right), \quad m = 1, \ldots, M-1, \\ \phi_{i+1} &:= \phi_i^{M-1} - \alpha_M \nabla_{\phi_i^{M-1}} \mathcal{L}_{T_i}\left(\boldsymbol{\tau}_{i,\phi_i^{M-1}}^{1:K}\right) \end{aligned} \tag{7}$$

- PG for (c), as expectation is taken to both $\boldsymbol{T_i}$ and $\boldsymbol{T_{i+1}}$, we change (4) to

$$\nabla_{\theta,\alpha} \mathcal{R}_{T_i,T_{i+1}}(\theta, \alpha) = \mathbb{E}_{\substack{\boldsymbol{\tau}_{i,\theta}^{1:K} \sim P_{T_i}(\boldsymbol{\tau}|\theta) \\ \boldsymbol{\tau}_{i+1,\phi} \sim P_{T_{i+1}}(\boldsymbol{\tau}|\phi)}} \left[\mathcal{L}_{T_{i+1}}(\boldsymbol{\tau}_{i+1,\phi}) \left[\nabla_{\theta,\alpha} \log \pi_\phi(\boldsymbol{\tau}_{i+1,\phi}) + \nabla_\theta \sum_{k=1}^K \log \pi_\theta(\boldsymbol{\tau}_{i,\theta}^k)\right]\right] \tag{8}$$

MAML for Nonstationary Task : Training

Algorithm 1 Meta-learning at training time.

input Distribution over pairs of tasks, $\mathcal{P}(T_i, T_{i+1})$, learning rate, β.

Randomly initialize θ and α.
repeat
    Sample a batch of task pairs, $\{(T_i, T_{i+1})\}_{i=1}^{n}$.
    for all task pairs (T_i, T_{i+1}) in the batch **do**
        Sample traj. $\boldsymbol{\tau}_{1:K}$ from T_i using π_θ.
        Compute $\phi = \phi(\boldsymbol{\tau}_{1:K}, \theta, \alpha)$ as given in ($\boxed{7}$).
        Sample traj. $\boldsymbol{\tau}$ from T_{i+1} using π_ϕ.
    end for
    Construct $\nabla_\theta \mathcal{R}_T(\theta, \alpha)$ and $\nabla_\alpha \mathcal{R}_T(\theta, \alpha)$ using $\boldsymbol{\tau}_{1:K}$ and $\boldsymbol{\tau}$ as given in ($\boxed{8}$).
    Update $\theta \leftarrow \theta + \beta \nabla_\theta \mathcal{R}_T(\theta, \alpha)$.
    Update $\alpha \leftarrow \alpha + \beta \nabla_\alpha \mathcal{R}_T(\theta, \alpha)$.
until Convergence

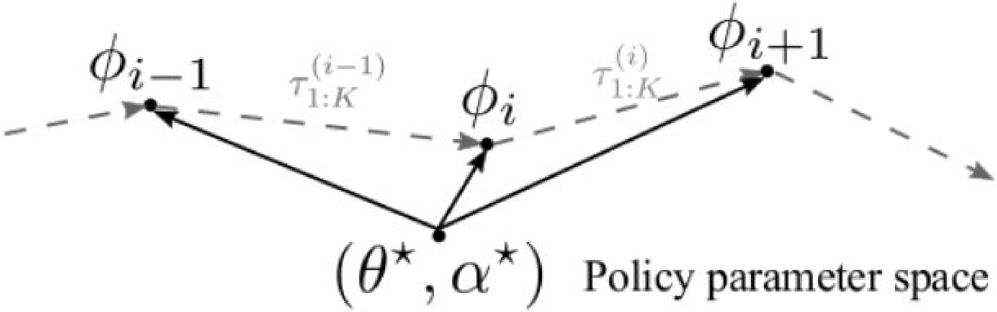
output Optimal θ^* and α^*.

- **Assumption: trajectories of T_i contain some infomation about T_{i+1}**
- Difference from original MAML:
  - Goal: 不仅仅优化 θ 还有一系列步长 α
  - Use a pair of consecutive tasks for training
  - During inner loop:
    - 首先基于原策略 π_θ 从 T_i 中获取一系列 trajectories $\tau_{1:K}$
    - 此处存疑，原文中说获取**trajectories**的过程中还需要与T_{i+1}互动
    - 根据这些 T_i 的 trajectories 计算 ϕ
    - π_ϕ 用于解决 T_{i+1}，基于该策略从 T_{i+1} 中获取一个 trajectory τ
  - Meta-update:
    - 通过一系列子任务获取$\tau_{1:K}$ 以及 τ后，计算$\nabla_\theta R_T(\theta,\alpha)$, $\nabla_\alpha R_T(\theta,\alpha)$
    - 更新 θ 与 α

MAML for Nonstationary Task : Testing

Algorithm 2 Adaptation at execution time.

input A stream of tasks, $T_1, T_2, T_3, \ldots$.
1: Initialize $\phi = \theta$.
2: **while** there are new incoming tasks **do**
3: Get a new task, T_i, from the stream.
4: Solve T_i using π_ϕ policy.
5: While solving T_i, collect trajectories, $\boldsymbol{\tau}^{(i)}_{1:K}$.
6: Update $\phi \leftarrow \phi(\boldsymbol{\tau}^{(i)}_{1:K}, \theta^*, \alpha^*)$ using importance-corrected meta-update as in ($\boxed{9}$).
7: **end while**



- Nonstationary $\rightarrow$ can not access to same task multiple times
- How to handle : keep acting according to π_ϕ and re-use past experience to for computing updates of ϕ for each new incoming task
- π_ϕ 获取的past experience与π_θ会有不同 $\rightarrow$ 使用 importance weight correction 进行调整

$$\phi_i := \theta - \alpha \frac{1}{K} \sum_{k=1}^{K} \left(\frac{\pi_\theta(\boldsymbol{\tau}^k)}{\pi_{\phi_{i-1}}(\boldsymbol{\tau}^k)} \right) \nabla_\theta \mathcal{L}(\boldsymbol{\tau}^k), \quad \boldsymbol{\tau}^{1:K} \sim P_{T_{i-1}}(\boldsymbol{\tau} \mid \phi_{i-1}), \tag{9}$$

Experiment : RoboSumo

Figure 2: (a) The three types of agents used in experiments. The robots differ in the anatomy: the number of legs, their positions, and constraints on the thigh and knee joints. (b) The nonstationary locomotion environment. The torques applied to red-colored legs are scaled by a dynamically changing factor. (c) The `RoboSumo` environment.

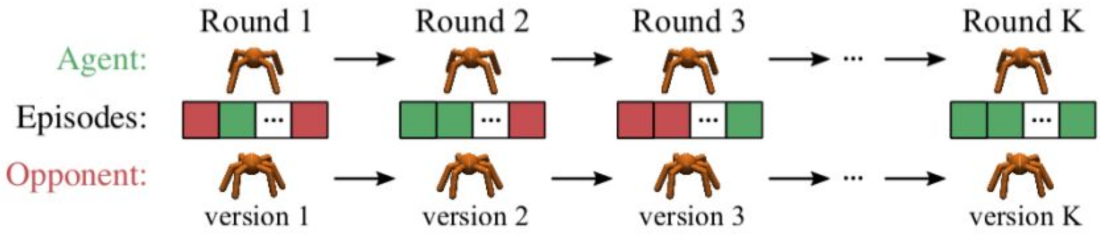


Figure 3: An agent competes with an opponent in an iterated adaptation games that consist of multi-episode rounds. The agent wins a round if it wins the majority of episodes (wins and losses illustrated with color). Both the agent and its opponent may update their policies from round to round (denoted by the version number).

Conclusion and Future Work

- Regard nonstationarity as a sequence of stationary tasks
- Train agents to exploit the dependencies between consecutive tasks
- During testing, the model can handle similar nonstationarities + Problem : 该工作把动态环境看作一系列静态环境的序列, 但真实环境会更复杂
- Future work

Current	**Future**
One-step-ahead update of the policy	Extend to fully recurrent meta-updates → take into full history of interaction
(c) requires second order derivatives at training time	Utilize information provided by the loss but avoid explicit backpropagation through the gradients
Can't handle sparse reward → the meta-updates use policy gradients and heavily rely on the reward signal	Try to introduce auxiliary dense rewards designed to enable meta-learning