

# 1703.03400 - Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

- Yunqiu Xu
- 2nd for this paper, try to understand both paper and code
- Other reference:
  - <https://www.jiqizhixin.com/articles/2017-07-20-4>
  - <http://bair.berkeley.edu/blog/2017/07/18/learning-to-learn/>
  - <https://github.com/cbfinn/maml>

## 1. Introduction

- The goal of meta-learning: train a model with some learning tasks, then it can solve new tasks with only a few samples
- Our work:
  - MAML: pretrain the model with a series of tasks
  - Then this model can be generated to new task with a small number of training samples / gradient steps on that task
- Our advantage: see 6.Discussion and Future Work

## 2. MAML

### 2.1 Problem Setup

- Goal: train a "meta learning" model on a set of tasks, then this model can adapt to a new task with only a few data / iterations → learn **as much as** possible with limited data
- Model  $f : x \rightarrow a$ 
  - $x$  : observations
  - $a$  : outputs

- This model is like a "base model" which will be able to adapt to a lot of new tasks
- General notion of task  $T = \{L(x_1, a_1, \dots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$  :
  - $L \rightarrow R$  : loss function
  - $q(x_1)$  : distribution over initial observations
  - $q(x_{t+1}|x_t, a_t)$  : transition 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

## 2.2 K-shot meta-learning:

- 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"

## 2.3 A MAML Algorithm

- What our model does?

- Be able to learn parameters of any standard model via meta-learning
- Why: some internal representations are more transferrable
- E.G. 我们可以通过一系列任务学到一个神经网络模型, 而非仅仅通过一个任务, 这样这个模型就比较容易迁移到类似的新任务上
- How we learn
  - Learn a model that gradient-based learning rule can make rapid progress on new tasks drawn from  $p(\mathcal{T})$  without overfitting
  - Find model parameters that are **sensitive** to changes of the task  $\rightarrow$  small changes lead to large improvement (direction of gradient)
  - 为什么这样做: 之前的目标就是在样本/更新次数有限的情况下学到的东西尽可能多, 因此我们要尽量让每一点点小改变都能获得较大的提升
  - 此处存疑: 代码里该怎么体现"sensitive"

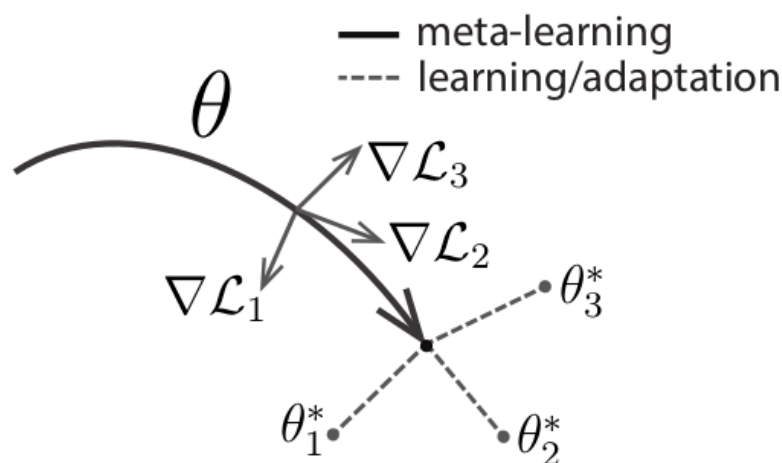


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation  $\theta$  that can quickly adapt to new tasks.

- Algorithm 1:
  - $f_\theta$ , when adapting to a new training task  $\mathcal{T}_i$ , the model's parameter  $\theta$  becomes  $\theta'_i$
  - Then we learn  $\theta'_i$  by gradient update:

$$\theta'_i = \theta - \alpha \nabla_{\theta} L_{\mathcal{T}_i}(f_{\theta})$$

- 这里我理解为调整梯度的方向, 比如为了适应  $\mathcal{T}_2$  我们需要将梯度方向稍微上移

- 经过多次梯度更新(内循环)后, 我们就可以学到比较适合  $T_2$  的参数向量  $\theta'_2$
- Step size  $\alpha$ : fixed as a hyperparameter or meta-learned
- After learning the parameter vectors for all tasks in training task set ( $\theta'_1, \theta'_2, \dots, \theta'_n$ ), compute their test error to update  $\theta$ :
  - Meta objective

$$\min_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'_i}) = \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta - \alpha \nabla_{\theta} L_{T_i}(f_{\theta})})$$

- SGD:

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

---

### 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**
- 

- This process involves a gradient through a gradient  $\rightarrow$  an additional backward pass through  $f$  to compute Hessian-vector products

## 3. Species of MAML

---

**Algorithm 2** MAML for Few-Shot Supervised 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  $K$  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
```

---

---

**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  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, \dots, \mathbf{x}_H)\}$  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  $\mathcal{D}'_i = \{(\mathbf{x}_1, \mathbf{a}_1, \dots, \mathbf{x}_H)\}$  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
```

---

## 3.1 Supervised learning

- Model is to predict output value given input value
- For each training task  $\mathcal{T}_i$ 
  - $H = 1$  : single input and single output
  - $L$  = MSE or corss entropy
  - $q_i(\mathbf{x}_1)$  : 因为不存在时序观察值, 这个分布就是监督学习训练集样本的分布
  - I think there is no  $q_i(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{a}_t)$
  - Then generate K samples  $\mathbf{x}$  from  $q_i(\mathbf{x}_1)$ , compute the error between predicted value  $\mathbf{a}$  and ground truth  $\mathbf{y}$

## 3.2 RL

- Model is to predict action  $\mathbf{a}_t$  given state  $\mathbf{x}_t$
- For each training task  $\mathcal{T}_i$ :
  - timestep  $t \in \{1, \dots, H\}$
  - The initial state distribution  $q_i(\mathbf{x}_1)$ : 为了训练这个子任务, 我们会从这个初始值分布选取K个起始点
  - Transition distribution  $q_i(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{a}_t)$ :
    - 对每个当前观察值及选取的动作, 未来观察值同样构成一个分布
    - 比如我选择吃一口饭, 接下来可能观察到 {饱了, 还饿} 等状态
    - 当然我们训练好  $\mathcal{T}_i$ , 获得的未来状态可能就是选取最优动作后的结果了

- For  $T_i$  and its model parameter  $\phi$ , the loss function is

$$L_{T_i}(f_\psi) = -E_{x_t, a_t \sim f_{\psi, q_{T_i}}} \left[ \sum_{t=1}^H R_i(x_t, a_t) \right] \quad (4)$$

- As reward function is to maximize reward, in loss function we multiply "-1" to minimize the value
- Here  $\phi$  is  $\theta'_i$ , which we mentioned before
- Why in step 8 we sample trajectories using  $f_{\theta'_i}$  instead of  $f_{\theta_i}$ : PG is on-policy, thus each additional gradient updating during the adaption of  $f_\theta$  need to sample from current policy

## 4. Related work

	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Omniglot (Lake et al., 2011)				
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
<b>MAML, no conv (ours)</b>	<b>89.7 ± 1.1%</b>	<b>97.5 ± 0.6%</b>	–	–
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%
<b>MAML (ours)</b>	<b>98.7 ± 0.4%</b>	<b>99.9 ± 0.1%</b>	<b>95.8 ± 0.3%</b>	<b>98.9 ± 0.2%</b>

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

- RNNs as learners: MANN
  - Search space includes all conceivable ML algorithms
  - Moves the burden of innovation to RNNs
  - Ignores advances achieved in ML by humans
  - The results are not good
- Metric learning: Siamese nets, matching nets
  - Learn a metric in input space

- Specialized to one/few-shot classification
- Can't use in other problems
- Optimizer learning: meta-learner LSTM
  - Learn parameter update given gradients (search space includes SGD, RMSProp, Adam etc)
  - Applicable to any architecture / task
  - But we can achieve better performance with MAML

## 5. Experimental evaluation

- Questions need to be answered:
  - Can MAML enable fast learning of new tasks
  - Can MAML be used for meta-learning in multiple different domains: SL / RL ...
  - Can a model learned with MAML continue to improve with additional gradient updates and/or examples
- An oracle work:
  - Receives the identity of the task (which is a problem-dependent representation) as an additional input,
  - Thus oracle will be an upper bound on the performance of the model
  - 我们主要对比用或不用MAML的情形, 越接近oracle代表效果越好

### 5.1 Regression

- 模拟sin曲线
- Each task:
  - The shape of curve varies by amplitude and phase
  - Input and output of a sine wave
  - Data points sampled from  $[-5.0, 5.0]$
  - Loss: MSE
- Model architecture: NN with 2 hidden layers, 40 hidden nodes, ReLU
- Training:
  - $K = 10$

- $\alpha = 0.01$
- Adam
- After all training tasks, we get a pretrained model
- Testing:
  - Fine-tune the pretrained model with  $K$  test samples and a number of GDs
- Result
  - Left is pretrained with MAML, right is without MAML

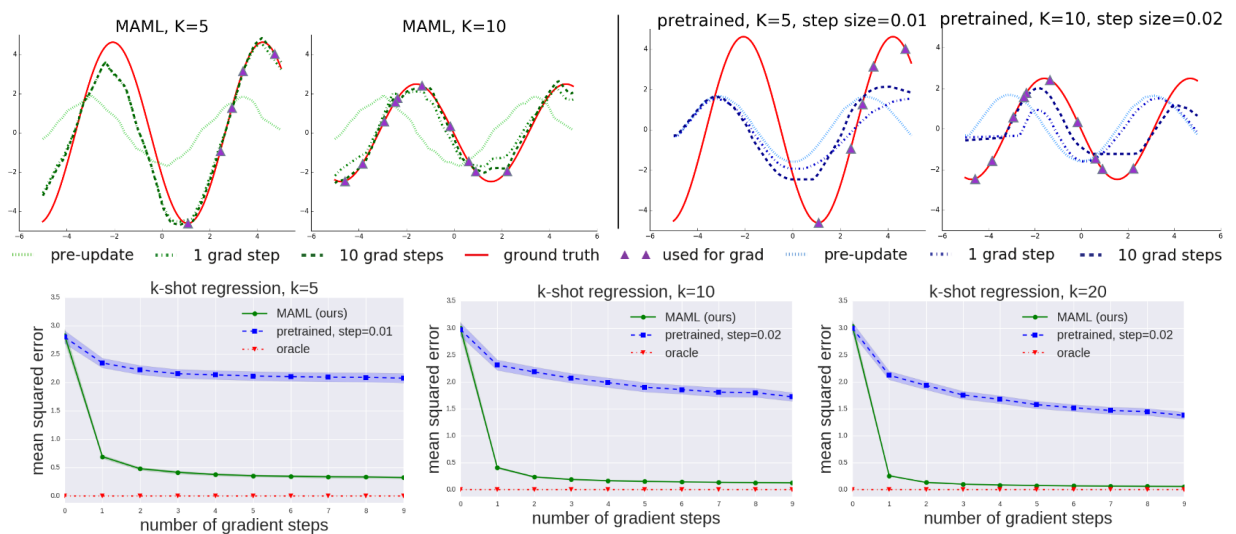


Figure 6. Quantitative sinusoid regression results showing test-time learning curves with varying numbers of  $K$  test-time samples. Each gradient step is computed using the same  $K$  examples. Note that MAML continues to improve with additional gradient steps without overfitting to the extremely small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

- Even with only 5 datapoints the fitting curve is nice
- When all the points are in one half, the model can still infer the shape of the other half  $\rightarrow$  model the periodic nature
- Quantitative results: improve with addition gradient steps
  - 但素, sin曲线这种简单任务好像也就是再多迭代一个循环的事情 :)
- 总之对Regression可以用很少的样本/循环finetune, 不会overfitting

## 5.2 Classification

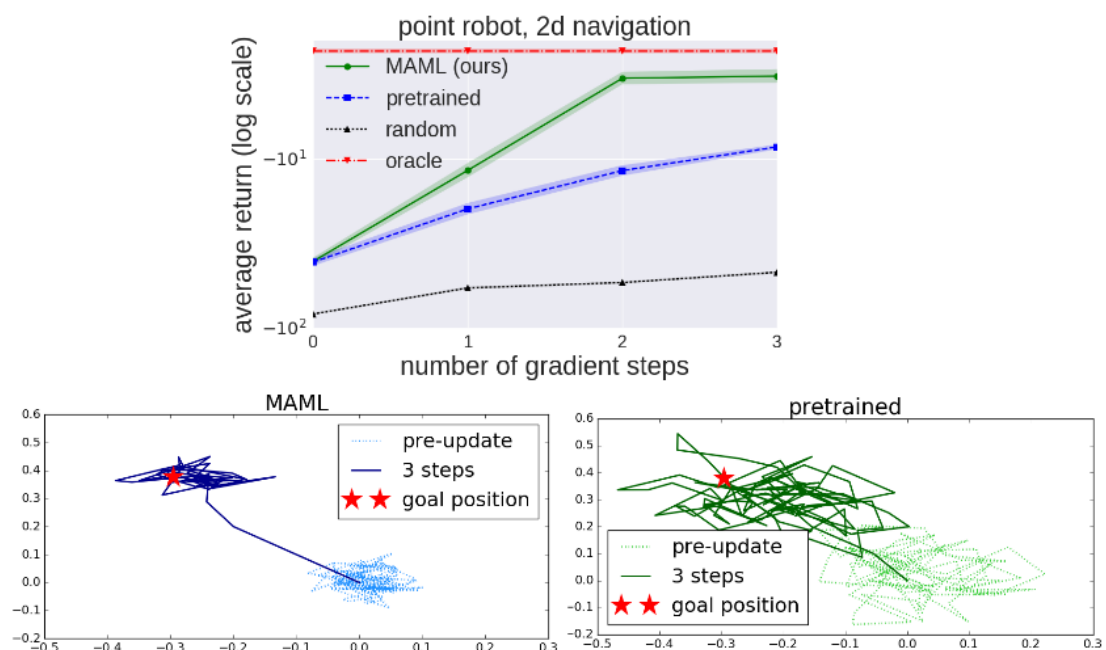
- Vinyals et al. 2016 Matching networks for one shot learning
- Tasks : Few-shot image recognition
- Datasets
  - Omniglot:



- 20 instances of 1623 characters from 50 different alphabets
  - Downsampled to  $28 * 28$
  - 1200 characters for training, remain for testing
  - Augmentation: degree rotations
- MiniImagenet: 64 training classes, 12 validation classes, and 24 test classes
- Evaluation : N-way classification
  - Select N unseen classes
  - Provide the model with K different instances of each of the N classes
  - Evaluate the model' s ability to classify new instances within the N classes
- Model architecture
  - 4 modules, each module:
    - $3*3$  conv, 64 filters (32 filters for MiniImagenet)
    - Batch normalization
    - ReLU nonlinearity
    - Strided convolutions ( $2*2$  max-pooling for MiniImagenet)
  - A non-conv network for comparison: 256-128-64-64, BN, ReLU
  - Loss: cross entropy
- **Comparison result : see 4. related work**
- We also compare the performance between first order and second order derivatives
  - From  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i})$  we can find that there is second order derivatives, which maybe computational expensive
  - Thus we compare it with first-order approximation
    - Compute the meta-gradient at the post-update parameter values  $\theta'_i$
  - Result:
    - **1st order is similar to 2nd order, less need to use 2nd derivatives**
    - Improvement of MAML comes from gradients of the objective at the post-update parameter values
    - Not 2nd derivative for differentiating through the gradient update
    - The use of ReLU make most of 2nd derivatives close to 0

## 5.3 Reinforcement Learning

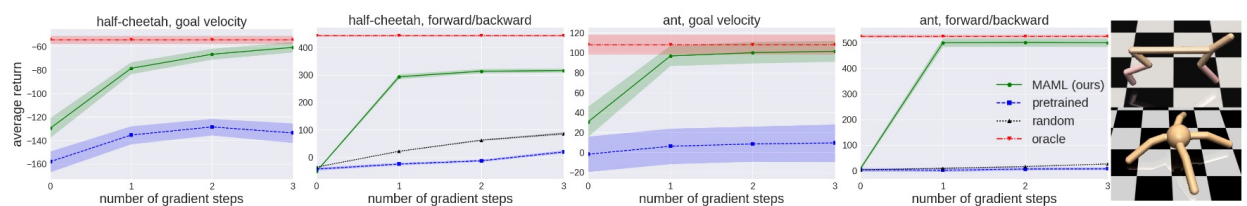
- Dual et al. 2016 Benchmarking deep reinforcement learning for continuous control
- Model architecture
  - 2 hidden layers, 100 hidden nodes, ReLU
  - Vanilla PG + TRPO
  - Use finite differences to compute Hessian-vector products for TRPO: avoid computing third derivatives
- Comparison:
  - policy inited with MAML
  - policy inited with randomly weights
  - oracle policy
- 2D Navigation
  - Goal: move to goal positions in 2D
  - Observation: current 2D position
  - Action: velocity commands
  - Reward: negative squared distance
  - When to terminate: too close to the goal, or  $H = 100$
  - Comparison: adaptation to a new task with up to 4 gradient updates, each with 40 samples



- MuJoCo Simulation:
  - Goal velocity experiment: the reward is the negative absolute value between

the current velocity of the agent and a goal

- Goal direction experiments: the reward is the magnitude of the velocity in either the forward or backward direction
- Result : MAML learns a model that can quickly adapt its velocity and direction with even just a single gradient update, and continues to improve with more gradient steps
- 本文中经过MAML预训练的模型强于随机初始化, 不过在Parisotto et al 2016.的工作中, 也有预训练不如随机初始化的情况
- Note that random baseline curves for game velocity are removed (worse return)



## 6. Discussion and Future Work

- MAML can be treated as an initialization method to get pretrained model that is easy to fine-tune
- Benefits:
  - Simple, does not introduce any learned parameters for meta-learning
  - Can be applied to regression / classification / RL
  - Adaptation on new tasks with few shot / updates
- Future work:
  - Generalize meta-learning technique to apply to any problem and any model → 1709.04905 - One-Shot Visual Imitation Learning via Meta-Learning
  - Apply to multi-task