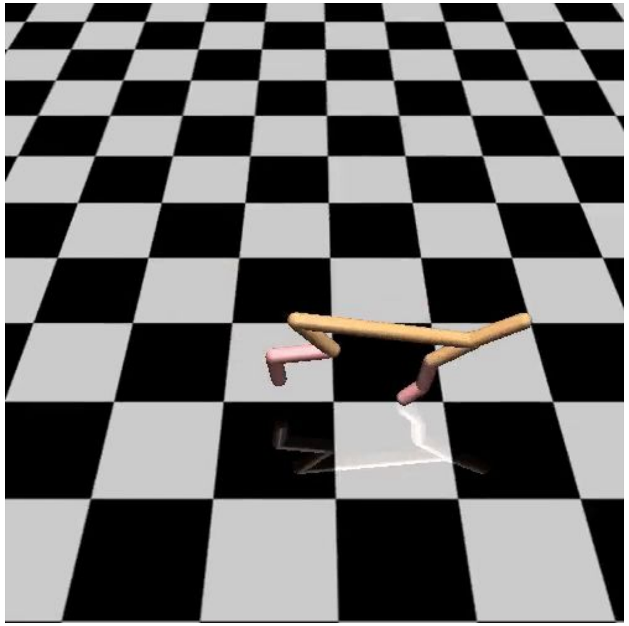
Transfer Learning techniques in Robotics

Melissa Mozifian

Transfer Club Week 2

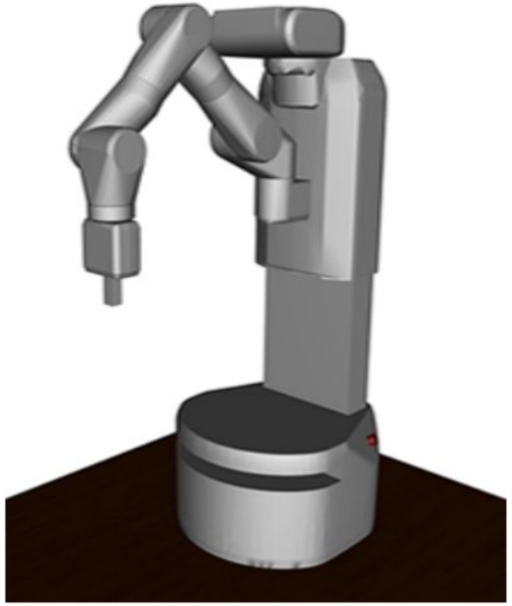
2018-11-02

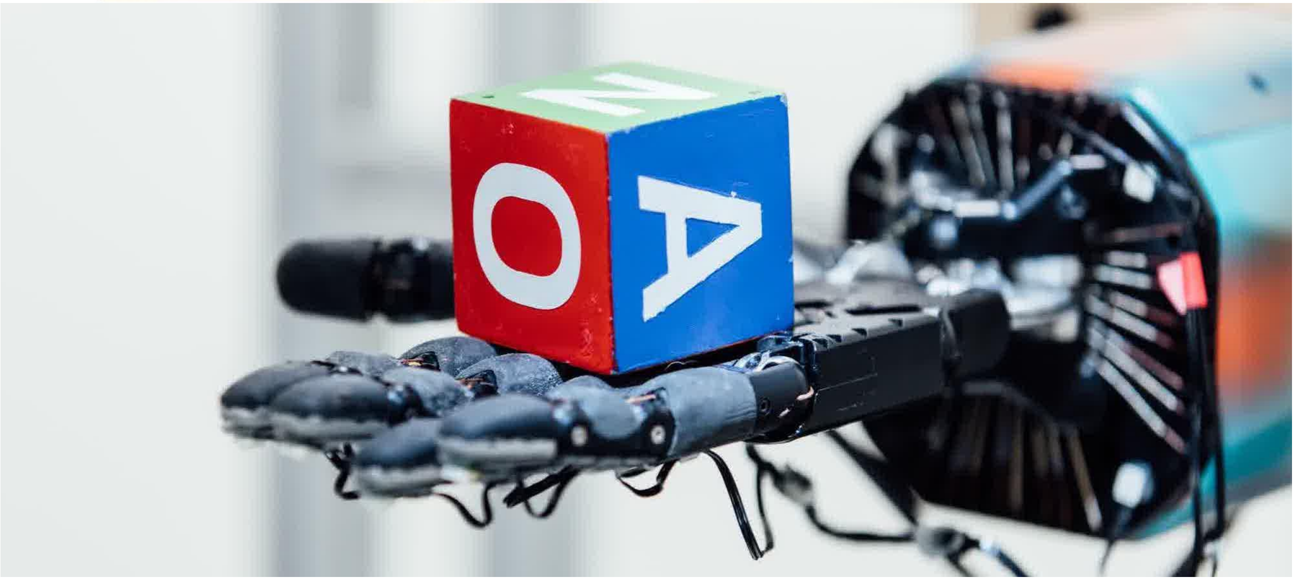
Motivation



Why is it hard?

- Real-world is complicated.
- High-dimensional control.
- Noisy and partial observations.
- Flexibility





Primary concerns

Versatility

Sample Efficiency

How is this related to Transfer?

- Build robots by training **entirely** in simulation, followed by a **small amount of self-calibration** in the real world.
- We want robots to be able to **transfer knowledge** seamlessly, acquire many skills and adapt to many environments.

How do we solve this?

There are *currently* 3 things one can do:

1. Train on huge fleets of physical robots
2. Make simulators increasingly match the real world
3. Randomize the simulator to allow the model to generalize to the real-world.

How do we solve this?

There are *currently* 3 things one can do:

1. Train on huge fleets of physical robots
2. Make simulators increasingly match the real world
3. Randomize the simulator to allow the model to generalize to the real-world.

Domain Randomization

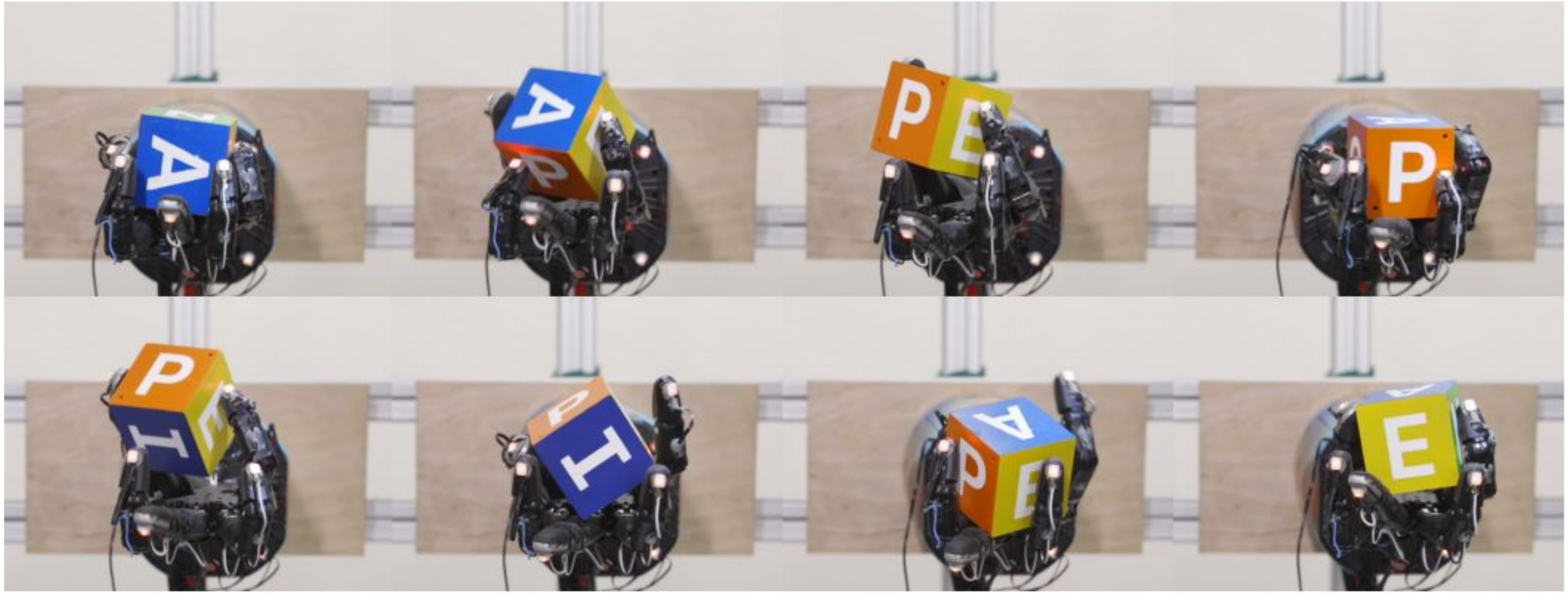
Idea :

- Let’s not try to model the real-world dynamics perfectly.
- Instead *randomize relevant* aspects of the environment.

Domain Randomization

- Randomize a large set of properties that determine the *dynamics* of the environment
- e.g.
  - Mass of each link in the robot’s body
  - Friction & damping of the objects its being trained on
  - Latency between actions
  - Noise in observations
  - ….

Learning Dexterous In-Hand Manipulation

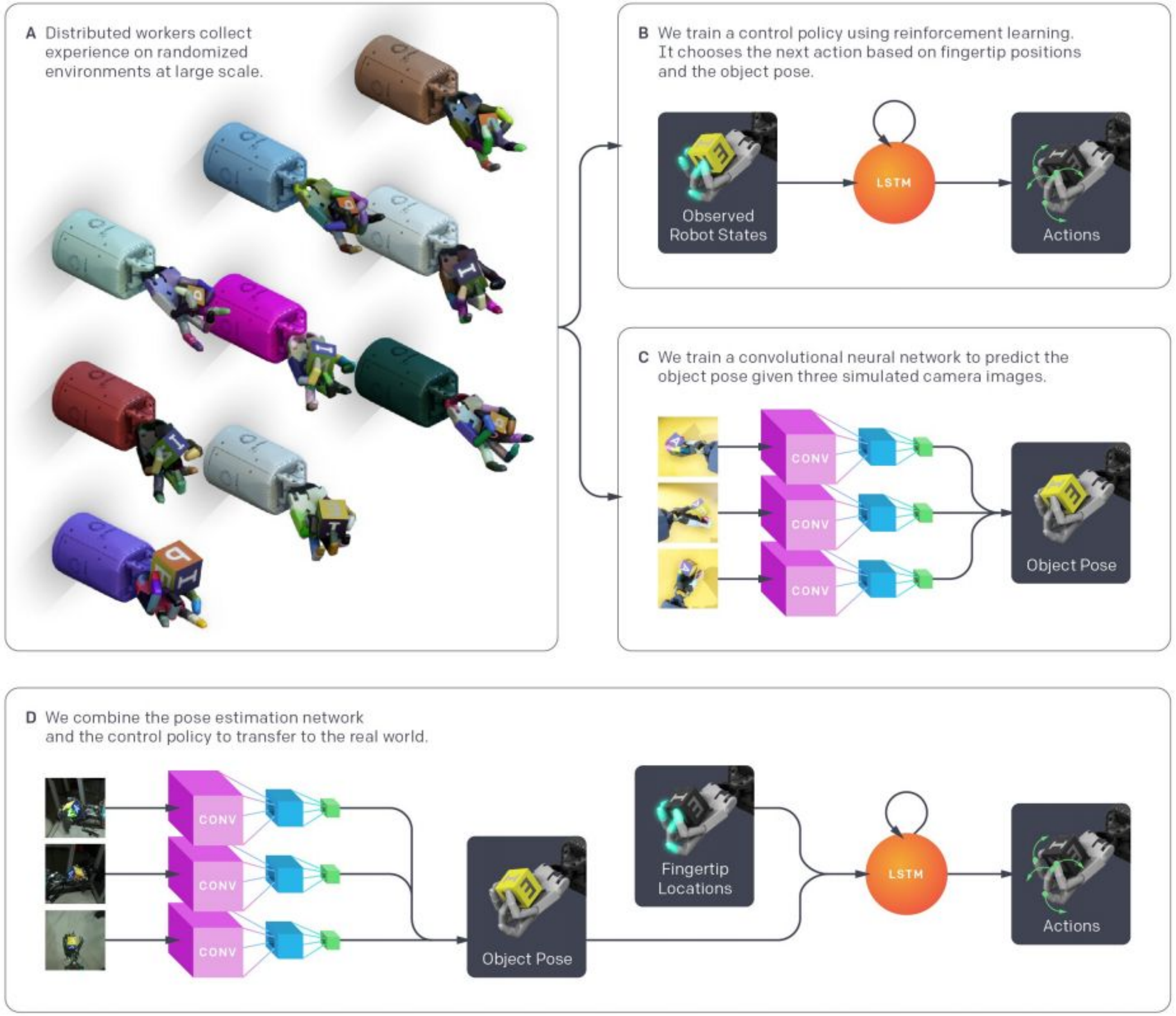


OpenAI*

Learning Dexterity - Dactyl

- Object manipulation task using a robot arm
- The network observes :
  - The coordinates of the fingertips
  - The images from three regular RGB cameras.

Learning Dexterity - Dactyl



Learning Dexterity - Dactyl

- Learning control policy from states
  - Policy as an **LSTM** trained with **PPO** (Proximal Policy Optimization)
- PPO requires training two networks
  - Policy network, maps observations to actions
  - Value network, predicts discounted sum of future rewards.
    - Asymmetric Actor-Critic

Ranges of physics parameter randomizations

Parameter	Scaling factor range	Additive term range
object dimensions	$\text{uniform}([0.95, 1.05])$	
object and robot link masses	$\text{uniform}([0.5, 1.5])$	
surface friction coefficients	$\text{uniform}([0.7, 1.3])$	
robot joint damping coefficients	$\text{loguniform}([0.3, 3.0])$	
actuator force gains (P term)	$\text{loguniform}([0.75, 1.5])$	
joint limits		$\mathcal{N}(0, 0.15)$ rad
gravity vector (each coordinate)		$\mathcal{N}(0, 0.4)$ m/s^2

What did we gain, at what cost?

- Gives the robot a variety of *experiences* rather than maximizing realism.
- Costs/challenges:
    - Dynamics randomization slows down training.
    - How do you select the appropriate randomization?
    - Would this really work with a robot **outside** of a laboratory?
    - Still making assumptions about the real word …
        - Real world experiments were used to tune the randomization.

Meta-learning / Learning-to-learn

"Meta-learning frames the learning problem at **two levels**. The first is quick acquisition of **knowledge within each separate task** presented. This process is guided by the second, which involves slower extraction of **information learned across all the tasks**."

(Ravi & Larochelle '17.)

Meta-learning / Learning-to-learn

"Process of **learning to learn**. Use **experience** to change certain aspects of a learning algorithm, or the learning method itself, such that the **modified learner is better** than the original learner at learning from additional experience."

(Scholarpedia, 5(6):4650.)

What-to-transfer?

- Meta-learner / parameters (of task functions)
  - **Optimization-based**
  - **Initialization-based**
- Embeds in a single function (task functions as meta-learner)
  - **Model-based**
  - **Metric-based**
- Random variables / random variables
  - **graphical models**

One-shot / few-shot learning

- In RL the goal of *few-shot meta-learning* is :
  - Enable an agent to quickly acquire a policy for a new task using a *small* amount of experience in the test setting.
- What is a new task?
  - Achieving a *new goal* or succeeding on the same task in a *new environment*.

One-shot / few-shot learning

	Classification	Regression	RL
Task	Task	Task	MDP
Learner	Classifier	Regressor	Policy
K-shot samples	K examples per class (N-classes) = N-way K-shot Learning	K examples	K roll-outs (episodes)

1. Introduction to meta-learning - Jae Hyun Lim - MILA Reading Groups

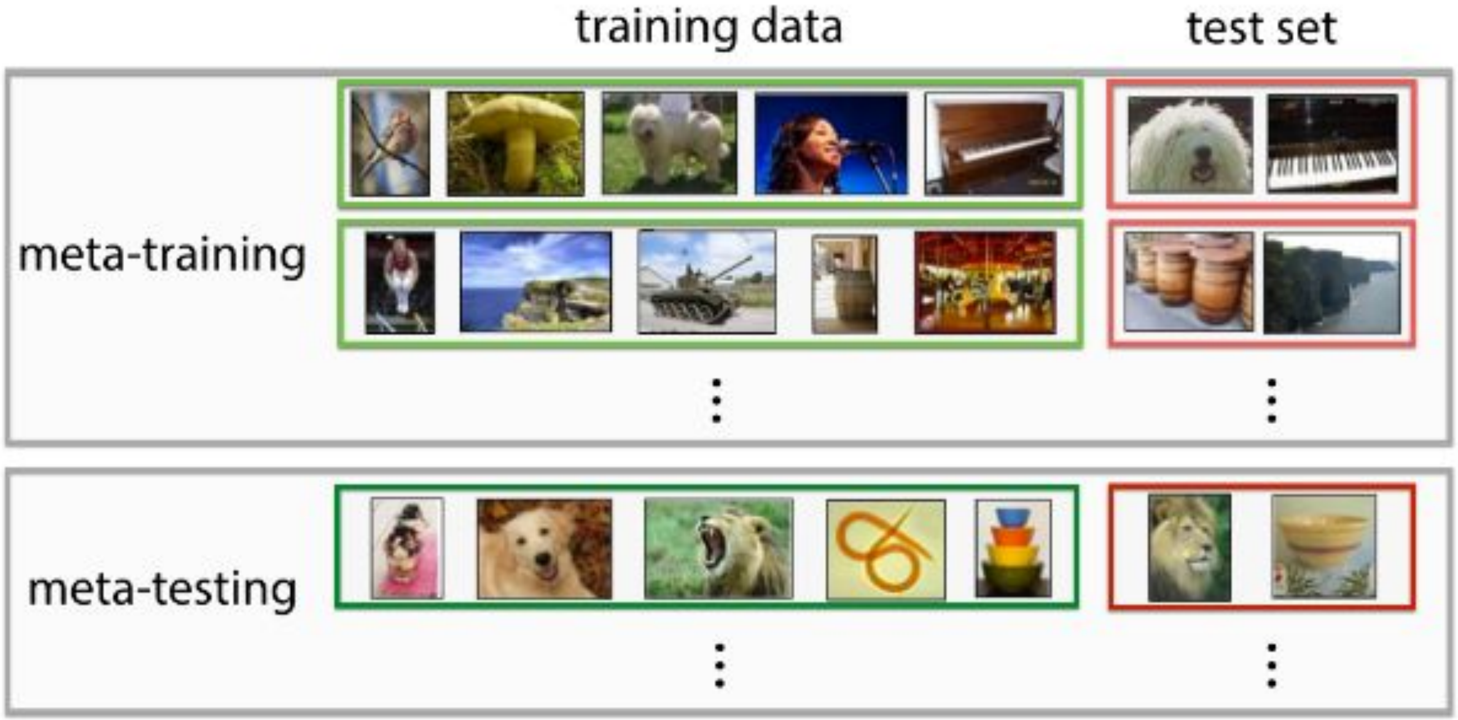
K-shot N-way classification

- Given a **base-training** set:
  - N object classes with K examples
- Given a **base-test** set:
  - Same N object classes but different examples
- Goal :
  - Learn a classifier f_θ that minimizes the validation error with **limited data** available.

K-shot reinforcement learning

- Base-task corresponds to an environment and a reward function $R(s_t|a_t)$
- We are allowed to execute our policy **K** times.
- The task is to **learn a policy** $f_\theta(a_t|s_t)$ that **maximizes reward**, under limited interaction with the environment.

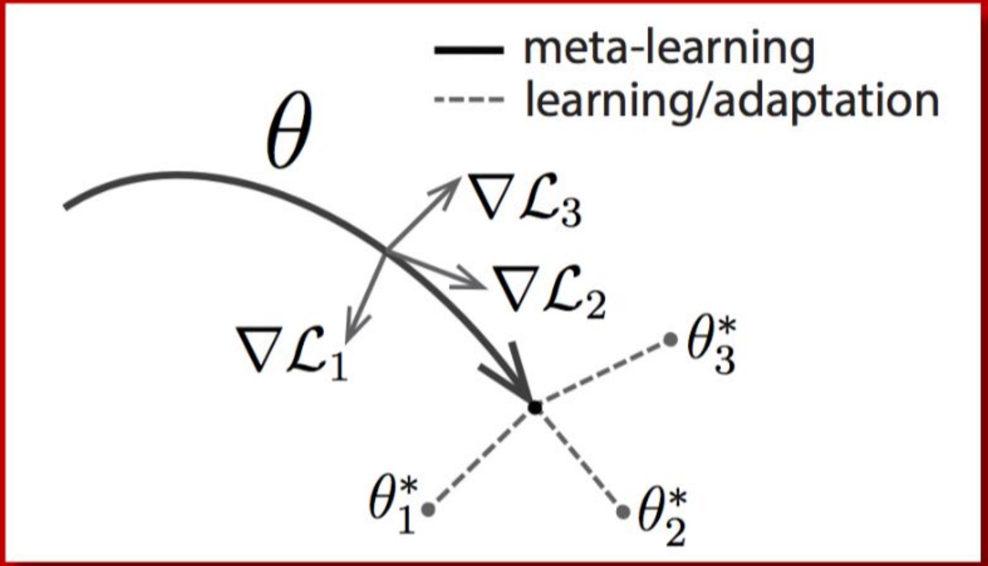
Few-shot image classification



Example meta-learning set-up for few-shot image classification, visual adapted from Ravi & Larochelle '17.

2. Learning to learn - Chelsea Finn - https://bair.berkeley.edu/blog/2017/07/18/learning-to-learn/

Model Agnostic Meta Learning



Chelsea Finn, Pieter Abbeel, Sergey Levine

MAML

- Key idea:
  - Train initial parameters s.t. the model has **maximal** performance on a **new task** after the parameters have been updated through **one** or more **gradient steps**, computed with a **small** amount of data from the new task.

MAML

- Don’t expand the number of learned parameters
- Don’t place constraints on architecture
- Maximize the sensitivity of the loss functions of new tasks w.r.t. the parameters

Formulation

- Formally, each task,

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

- For *each* task T_i model parameters become

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta).$$

- Objective

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

MAML for Supervised Learning

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**

T1: Classify Monkey, Banana, Tree

T2: Classify Pedestrian, Bicycle, Car

{(image, class), (image,class)...}
Base train

Base SGD update using base-train

{(image, class), (image,class)...}
Base test

Meta SGD update using base-train

MAML for RL

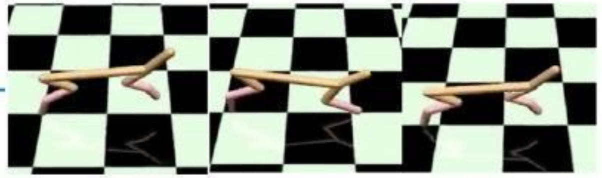
Algorithm 3 MAML for Reinforcement Learning

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

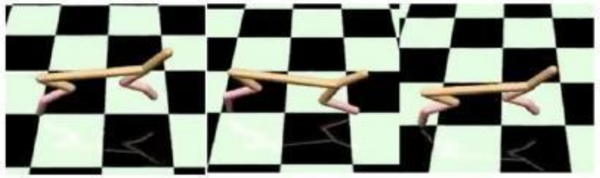
```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        Sample K trajectories D = {(x_1, a_1, ...x_H)} using f_θ in T_i
        Evaluate ∇_θ L_{T_i}(f_θ) using D and L_{T_i} in Equation 4
        Compute adapted parameters with gradient descent: θ'_i = θ − α∇_θ L_{T_i}(f_θ)
        Sample trajectories D'_i = {(x_1, a_1, ...x_H)} using f_{θ'_i} in T_i
    end for
    Update θ ← θ − β∇_θ Σ_{T_i~p(T)} L_{T_i}(f_{θ'_i}) using each D'_i and L_{T_i} in Equation 4
end while
```

T1: Run forward

T2: Run backwards



Base SGD update using base-train



Meta SGD update using base-train

Loss Functions

- Classification Loss

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)},\mathbf{y}^{(j)}\sim\mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1-\mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)}))$$

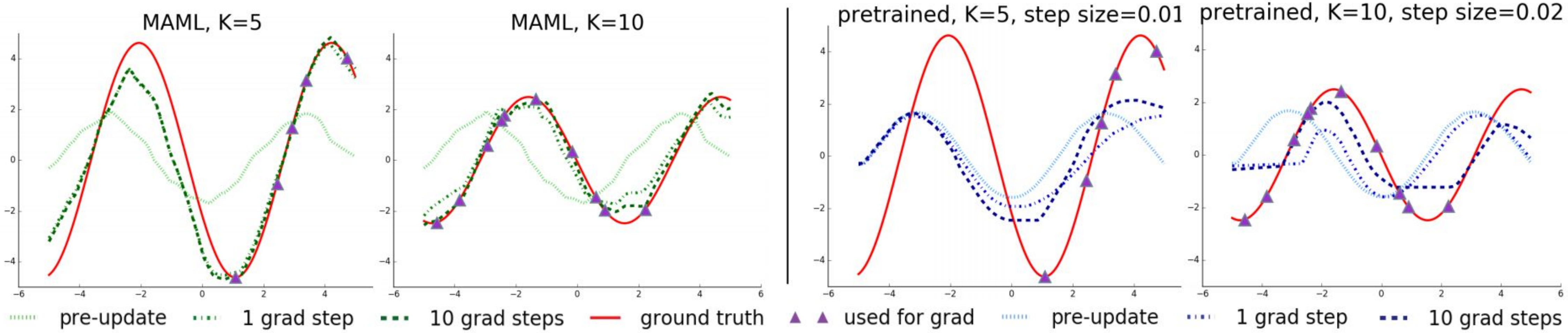
- Regression Loss

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)},\mathbf{y}^{(j)}\sim\mathcal{T}_i} \| f_\phi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)} \|_2^2$$

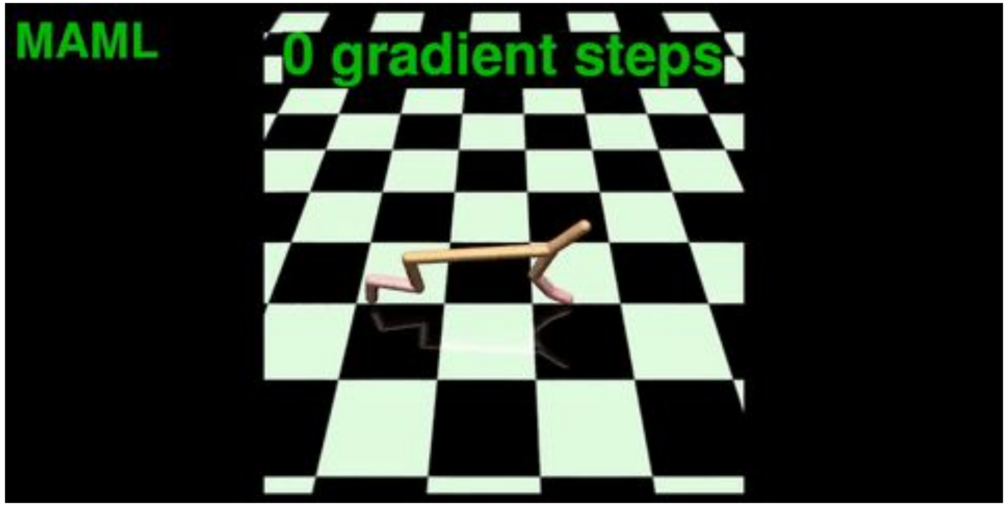
- RL Loss

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = -\mathbb{E}_{\mathbf{x}_t,\mathbf{a}_t\sim f_\phi, q_{\mathcal{T}_i}} \left[\sum_{t=1}^{H} R_i(\mathbf{x}_t, \mathbf{a}_t) \right]$$

MAML for Regression Task



MAML RL Task



Meta-learning overview

- **Learning Unsupervised Learning Rules**

Luke Metz, Niru Maheswaranathan, Brian Cheung, Jascha Sohl-Dickstein

Method	Inner loop updates	Outer loop updates, meta-			Generalizes to
		parameters	objective	optimizer	
Our work — metalearning for unsupervised representation learning	**many applications of an unsupervised update rule**	**parametric update rule**	**few shot classification after unsupervised pre-training**	**SGD**	**new base models (width, depth, nonlinearity), new datasets, new data modalities**

- Treat the creation of the **unsupervised update rule** as a transfer learning problem.

Method	Inner loop updates	Outer loop updates, meta-			Generalizes to
		parameters	objective	optimizer	
Hyper parameter optimization [31, 32, 33, 34]	many steps of optimization	optimization hyper-parameters	training or validation set loss	Baysian methods, random search, etc	nothing, or test set within fixed dataset
Neural architecture search [35, 36, 37, 38, 39]	supervised SGD training using meta-learned architecture	architecture	validation set loss	RL or evolution	test loss within similar datasets
Task-specific optimizer (eg for quadratic function identification) [1]	adjustment of model weights by an LSTM	LSTM weights	task loss	SGD	similar domain tasks
Learned optimizers [31, 40, 41, 42, 43, 44, 45]	many steps of optimization of a fixed loss function	parametric optimizer	average or final loss	SGD or RL	new loss functions (mixed success)
Prototypical networks [29]	apply a feature extractor to a batch of data and use soft nearest neighbors to compute class probabilities	weights of the feature extractor	few shot performance	SGD	new image classes within similar dataset
MAML [28]	one step of SGD on training loss starting from a meta-learned network	initial weights of neural network	reward or training loss	SGD	new goals, similar task regimes with same input domain
Evolved Policy Gradient [46]	performing gradient descent on a learned loss	parameters of a learned loss function	reward	Evolutionary Strategies	new environment configurations, both in and not in meta-training distribution.
Few shot learning [25, 26, 27]	application of a recurrent model, e.g. LSTM, Wavenet.	recurrent model weights	test loss on training tasks	SGD	new image classes within similar dataset.
Meta-unsupervised learning for clustering [30]	run clustering algorithm or evaluate binary similarity function	clustering algorithm + hyperparameters, binary similarity function	empirical risk minimization	varied	new clustering or similarity measurement tasks
Learning synaptic learning rules [22, 23]	run a synapse-local learning rule	parametric learning rule	supervised loss, or similarity to biologically-motivated network	gradient descent, simulated annealing, genetic algorithms	similar domain tasks

Follow-up Questions

- How to **incorporate new information** from **target** task?
- How to **autonomously make decisions** on whether to spend more time learning on the *source* task or the *target* task?
- How to **predict** a successful transfer?
- How to **estimate** how much knowledge is transferred?
- Assumptions about the source and target task **distributions**.