Meta-Learning

CS 294-112: Deep Reinforcement Learning

Sergey Levine

Class Notes

1. Two weeks until the project milestone!
2. Guest lectures start next week, be sure to attend!
3. Today: part 1: meta-learning
4. Today: part 2: parallelism

How can we frame transfer learning problems?

No single solution! Survey of various recent research papers

1. "Forward" transfer: train on one task, transfer to a new task
   a) Just try it and hope for the best
   b) Finetune on the new task
   c) Architectures for transfer: progressive networks
   d) Randomize source task domain
2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Model-based reinforcement learning
   b) Model distillation
   c) Contextual policies
   d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
   a) RNN-based meta-learning
   b) Gradient-based meta-learning

So far…

- Forward transfer: source domain to target domain
  - Diversity is good! The more varied the training, the more likely transfer is to succeed
- Multi-task learning: even more variety
  - No longer training on the same kind of task
  - But more variety = more likely to succeed at transfer
- How do we represent transfer knowledge?
  - Model (as in model-based RL): rules of physics are conserved across tasks
  - Policies – requires finetuning, but closer to what we want to accomplish
  - What about *learning methods*?

What is meta-learning?

- If you've learned 100 tasks already, can you figure out how to *learn* more efficiently?
  - Now having multiple tasks is a huge advantage!
- Meta-learning = *learning to learn*
- In practice, very closely related to multi-task learning
- Many formulations
  - Learning an optimizer
  - Learning an RNN that ingests experience
  - Learning a representation

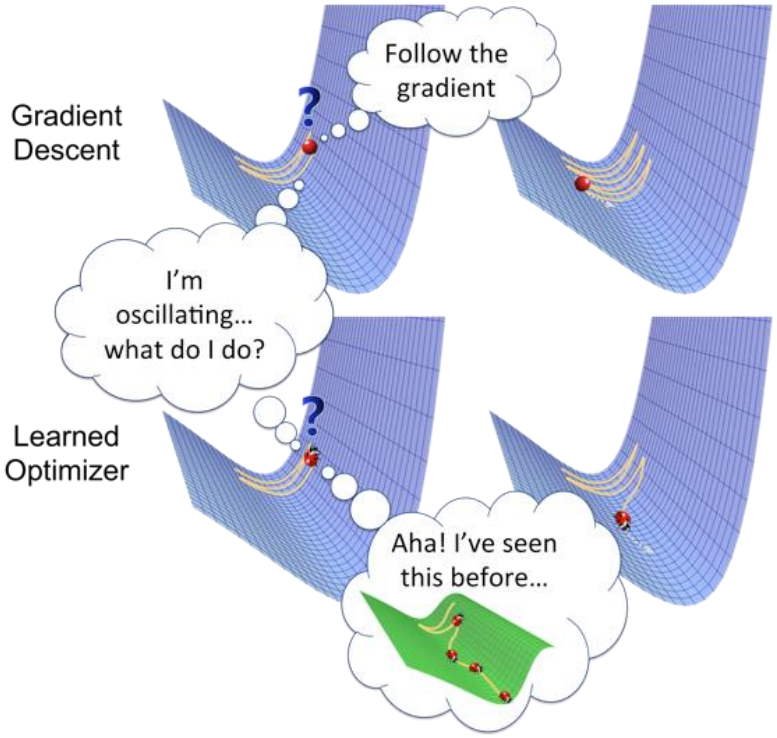


image credit: Ke Li

Why is meta-learning a good idea?

- Deep reinforcement learning, especially model-free, requires a huge number of samples
- If we can *meta-learn* a faster reinforcement learner, we can learn new tasks efficiently!
- What can a *meta-learned* learner do differently?
  - Explore more intelligently
  - Avoid trying actions that are know to be useless
  - Acquire the right features more quickly

Meta-learning with supervised learning

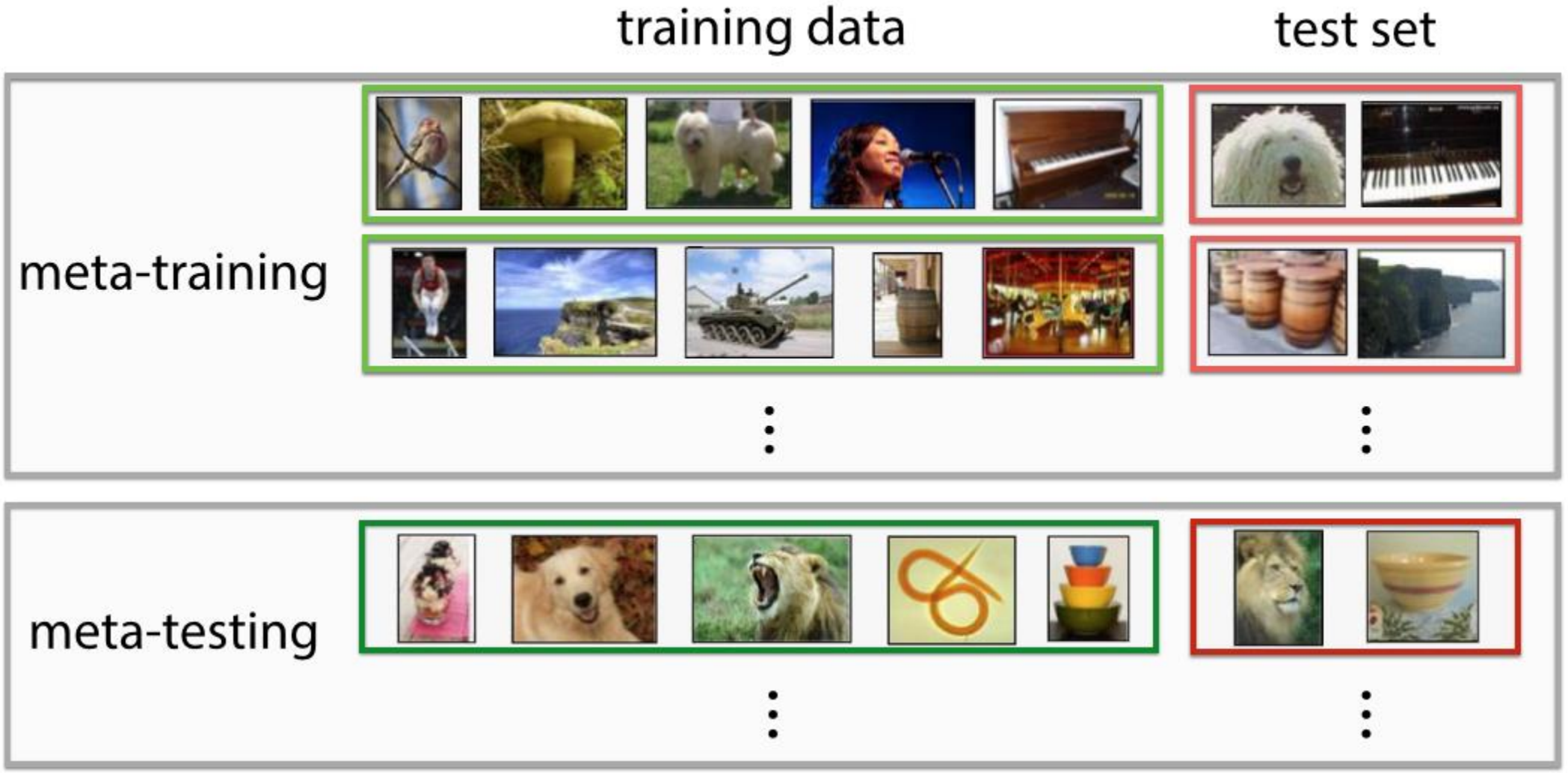
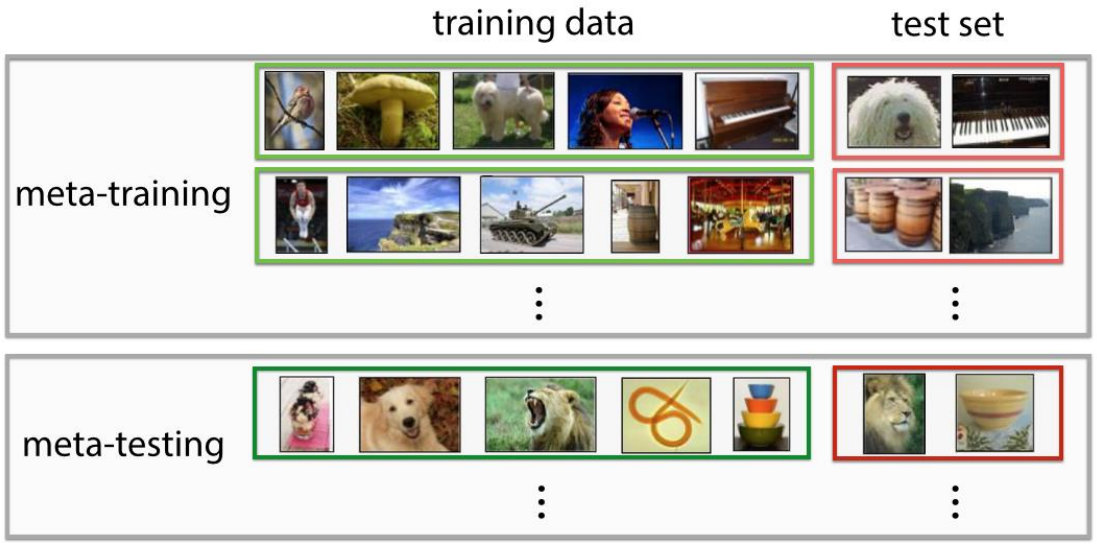


image credit: Ravi & Larochelle ‘17

Meta-learning with supervised learning

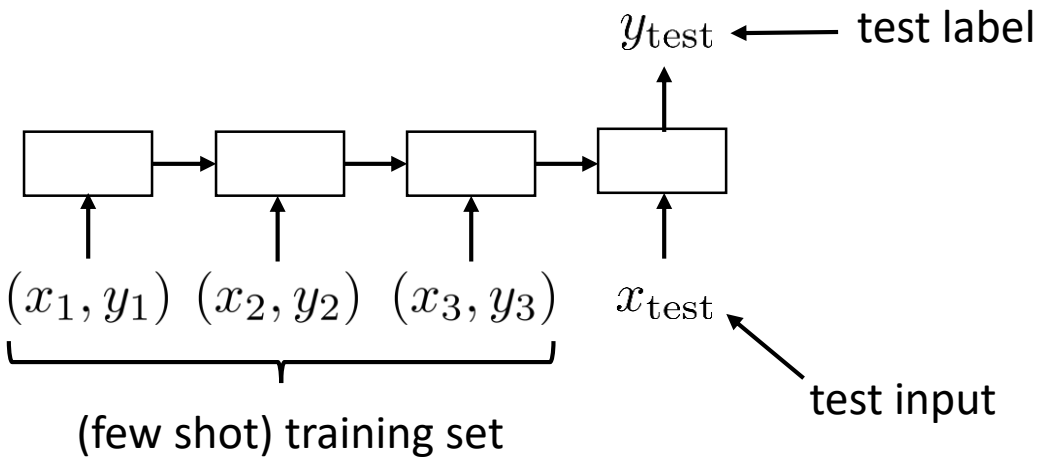


supervised learning: $f(x) \rightarrow y$

input (e.g., image) output (e.g., label)

supervised meta-learning: $f(\mathcal{D}_{\text{train}}, x) \rightarrow y$

training set



- How to read in training set?
  - Many options, RNNs can work
  - More on this later

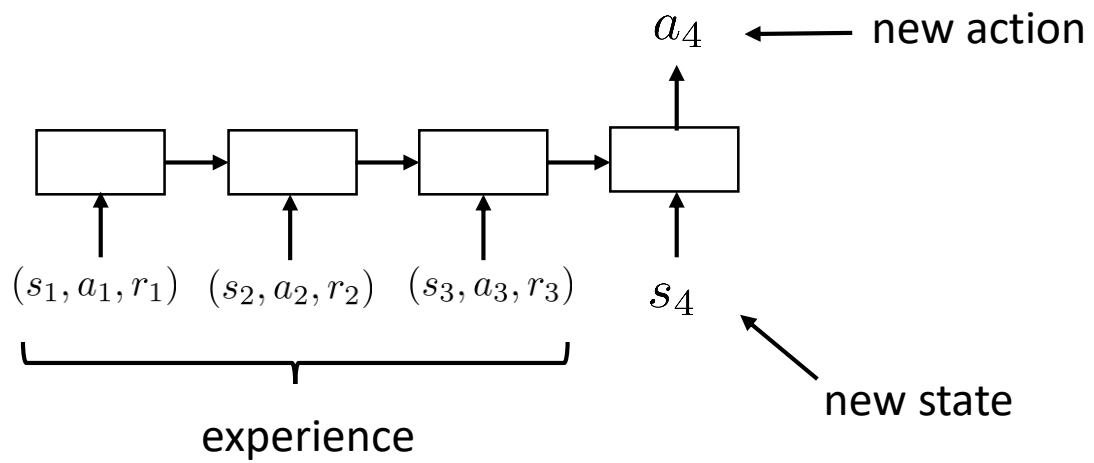
The meta-learning problem in RL

supervised meta-learning: $f(\mathcal{D}_{\text{train}}, x) \rightarrow y$

reinforcement meta-learning (for example...): $f(\mathcal{D}_{\text{train}}, s) \rightarrow a$

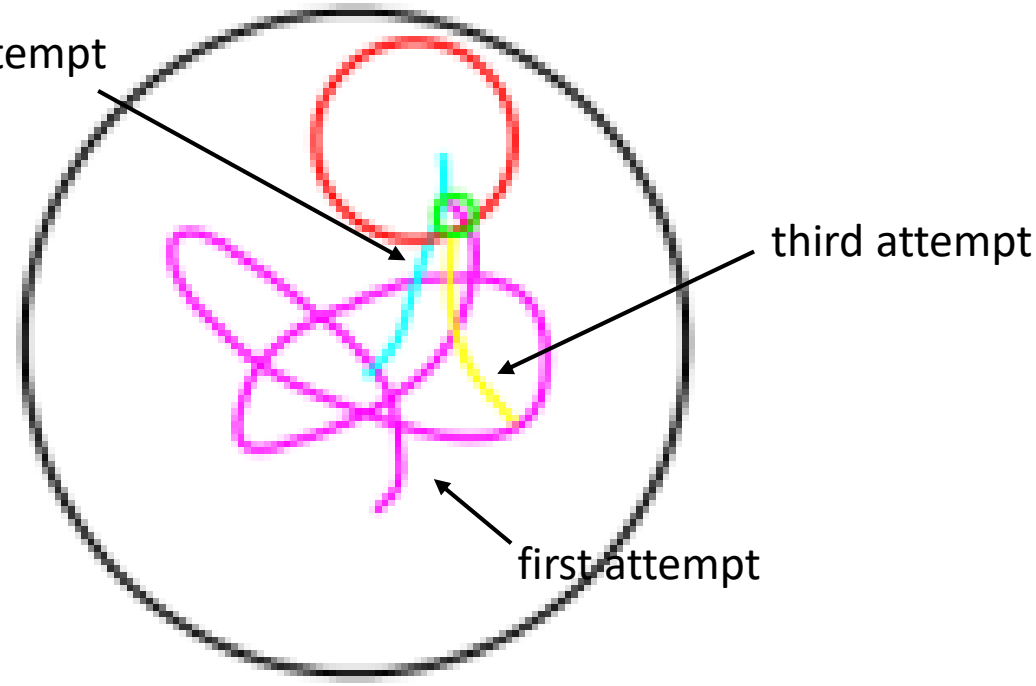
recent experience state output (e.g., action)

$\mathcal{D}_{\text{train}} = \{s_1, a_1, r_1, \ldots, a_N, s_N, r_N\}$

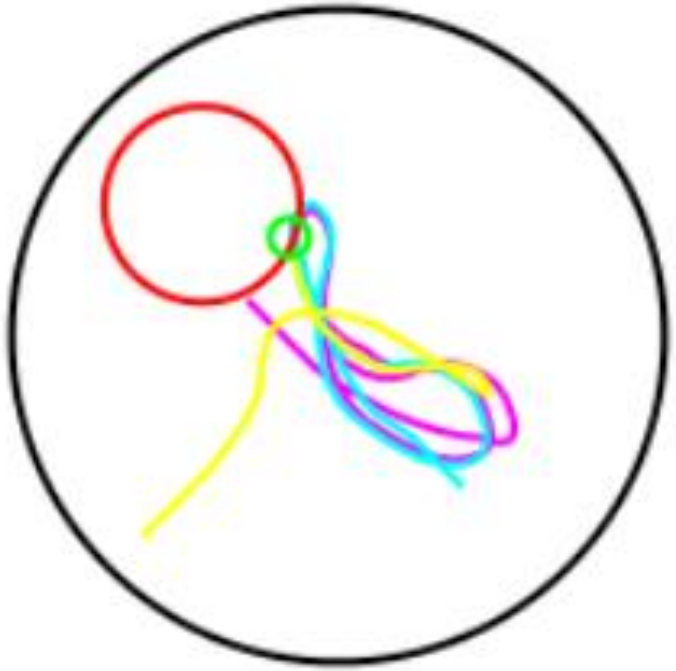


Meta-learning in RL with memory

"water maze" task



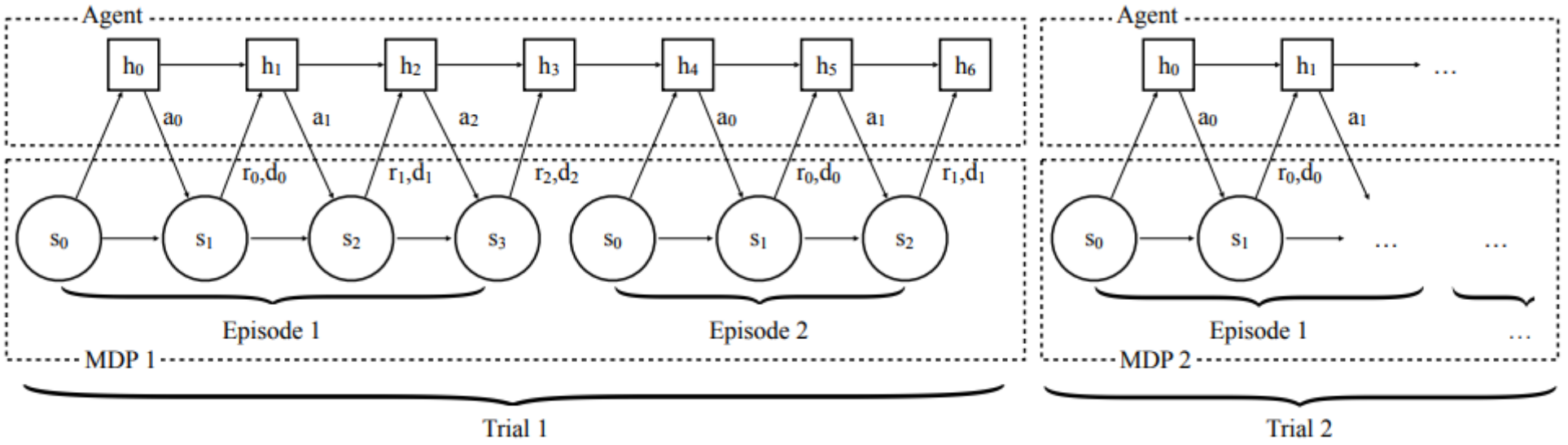
with memory

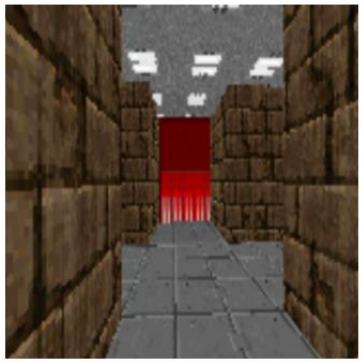


without memory

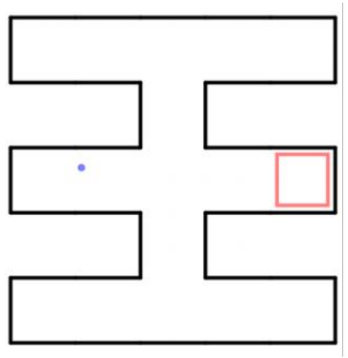
Heess et al., "Memory-based control with recurrent neural networks."

RL2

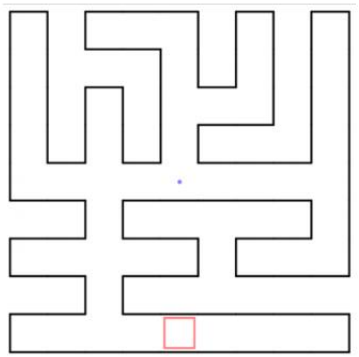




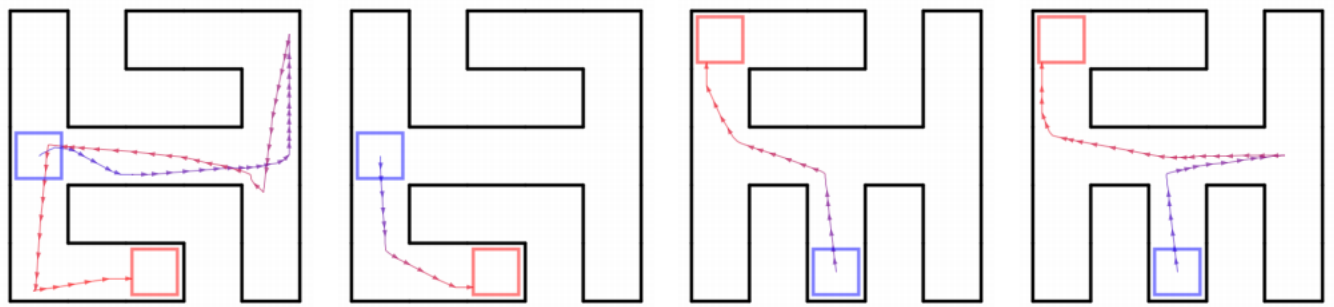
(a) Sample observation



(b) Layout of the 5×5 maze in (a)



(c) Layout of a 9×9 maze



(a) Good behavior, 1st episode

(b) Good behavior, 2nd episode

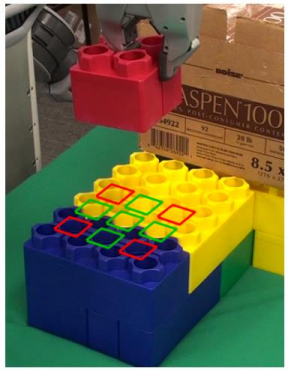
(c) Bad behavior, 1st episode

(d) Bad behavior, 2nd episode

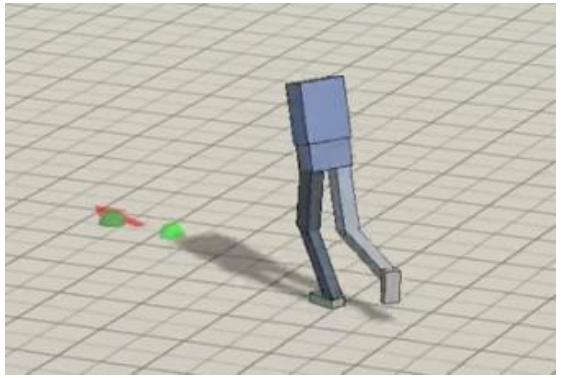
Duan et al., “RL2: Fast Reinforcement Learning via Slow Reinforcement Learning”

Connection to contextual policies

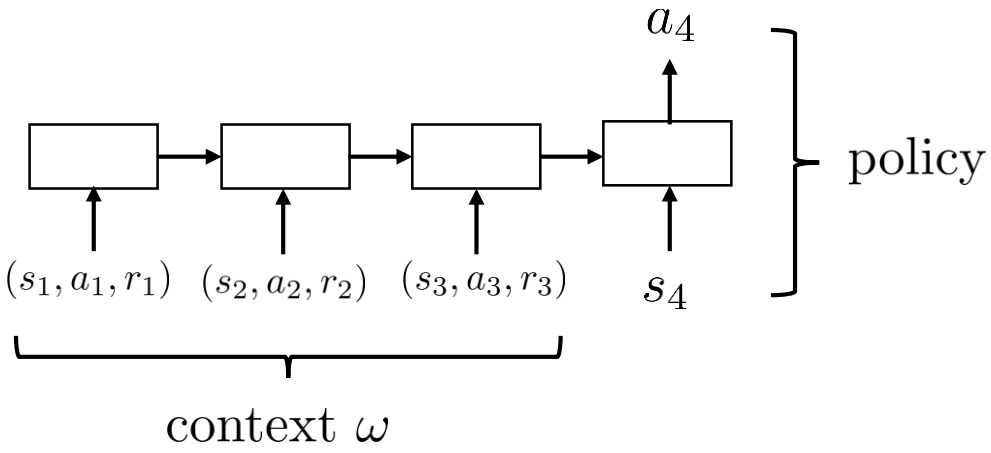
contextual policy: $\pi_\theta(\mathbf{a}|\mathbf{s}, \omega)$



ω: stack location

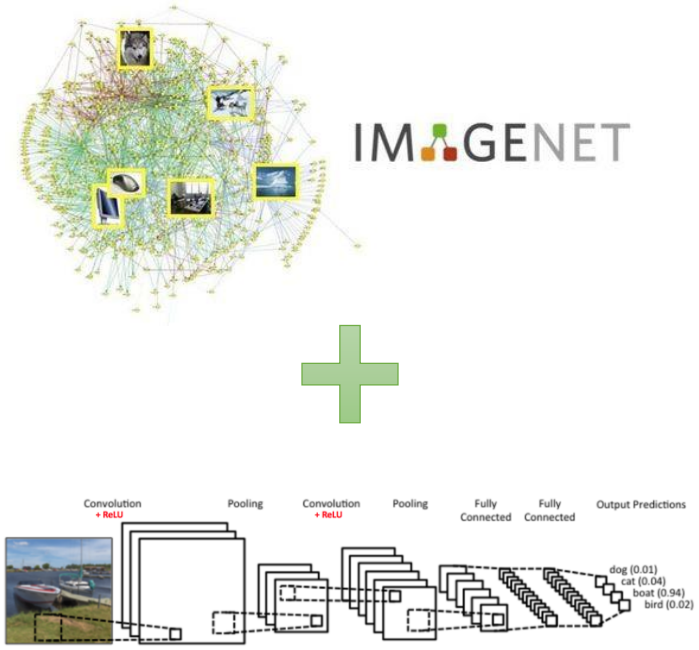


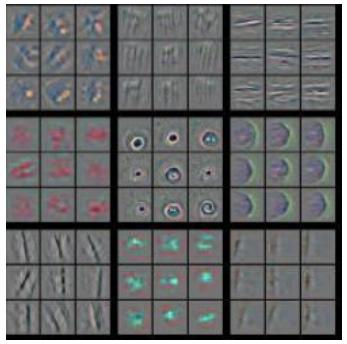
ω: walking direction

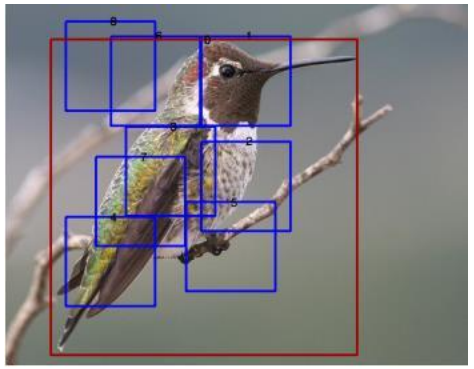


just contextual policies, with experience as context

Back to representations…



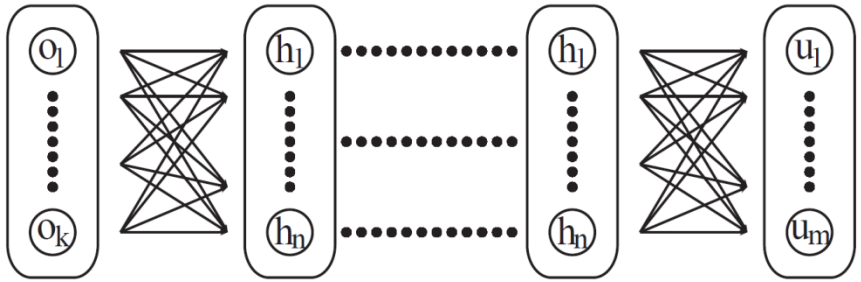


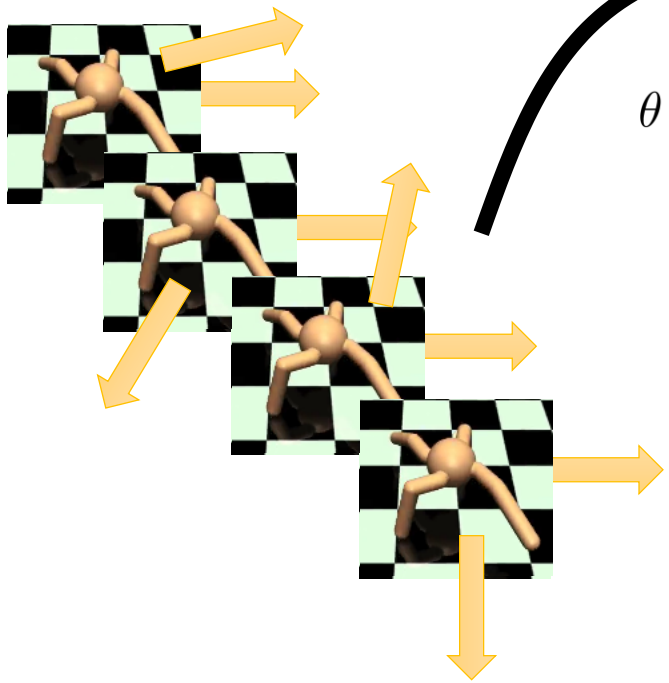


is pretraining a *type* of meta-learning?

better features = faster learning of new task!

Preparing a model for faster learning

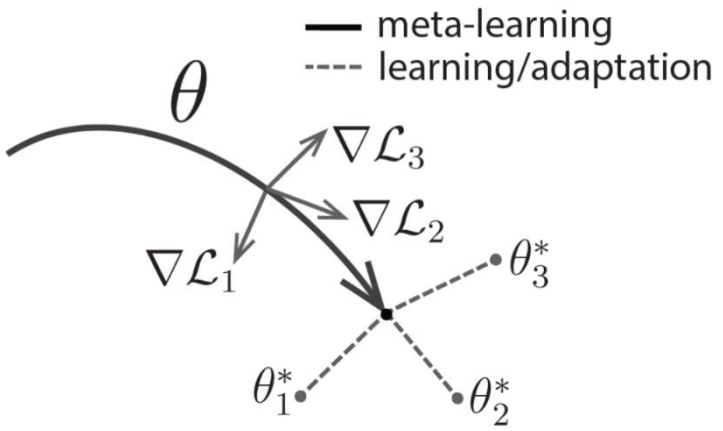




θ

~~$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$~~

$$\theta \leftarrow \theta + \alpha \sum_i \nabla_\theta R_i[\theta + \alpha \nabla_\theta R_i(\theta)]$$



θ'

Finn et al., “Model-Agnostic Meta-Learning”

What did we just do??

supervised learning: $f(x) \rightarrow y$

supervised meta-learning: $f(\mathcal{D}_{\text{train}}, x) \rightarrow y$

model-agnostic meta-learning: $f_{\text{MAML}}(\mathcal{D}_{\text{train}}, x) \rightarrow y$

$$f_{\text{MAML}}(\mathcal{D}_{\text{train}}, x) = f_{\theta'}(x)$$

$$\theta' = \theta - \alpha \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \nabla_\theta \mathcal{L}(f_\theta(x), y)$$

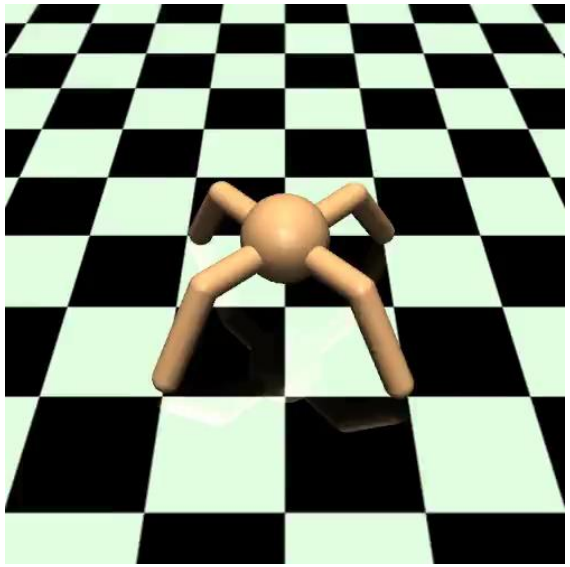
Just another computation graph…

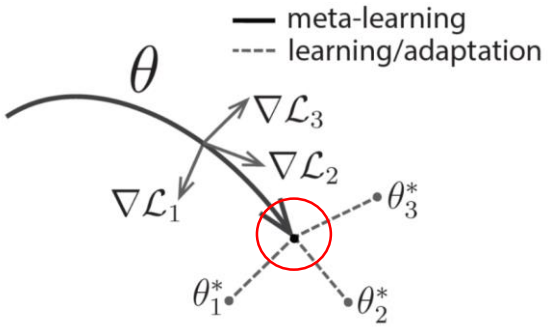
Can implement with any autodiff package (e.g., TensorFlow)

But has favorable inductive bias…

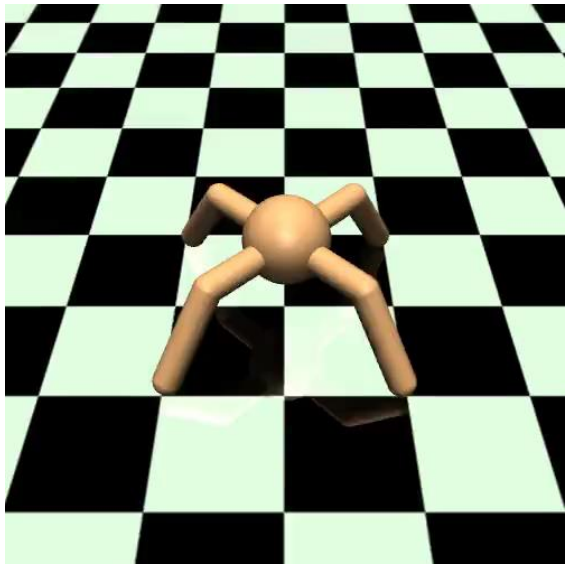
Model-agnostic meta-learning: accelerating PG

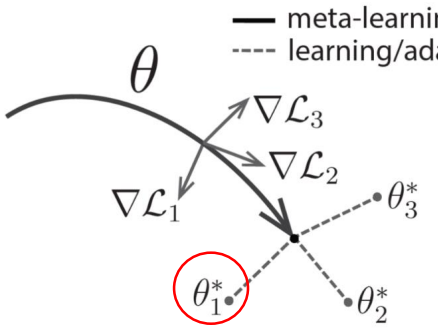
after MAML training



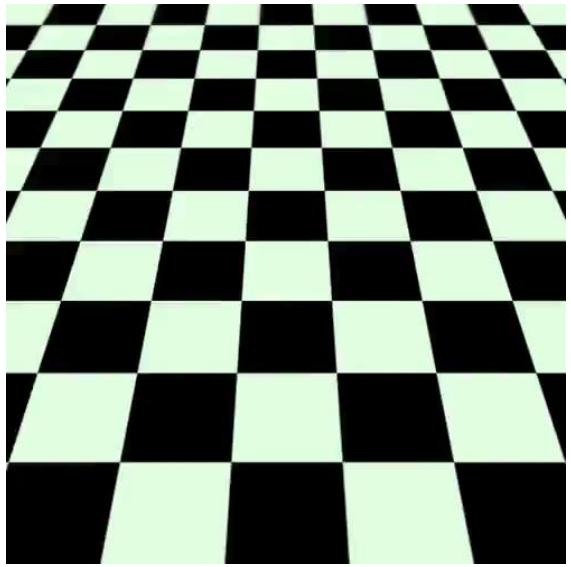


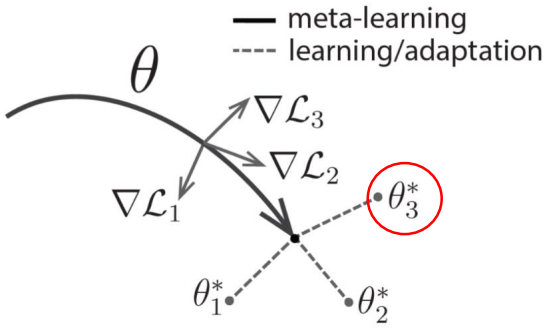
after 1 gradient step (forward reward)





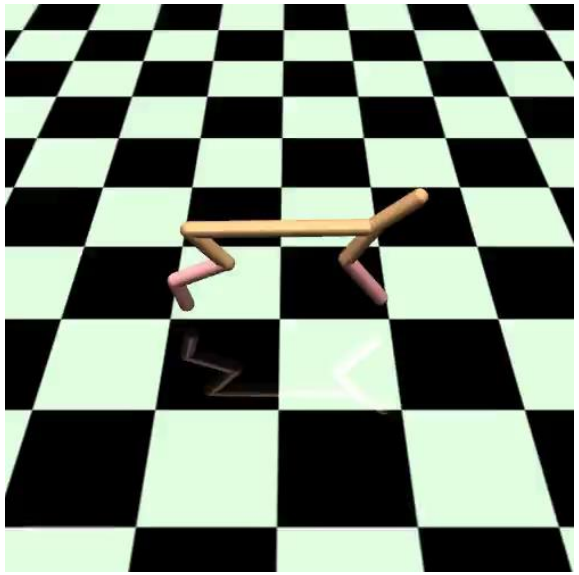
after 1 gradient step (backward reward)

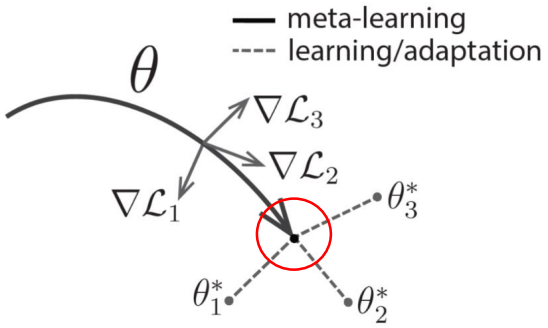




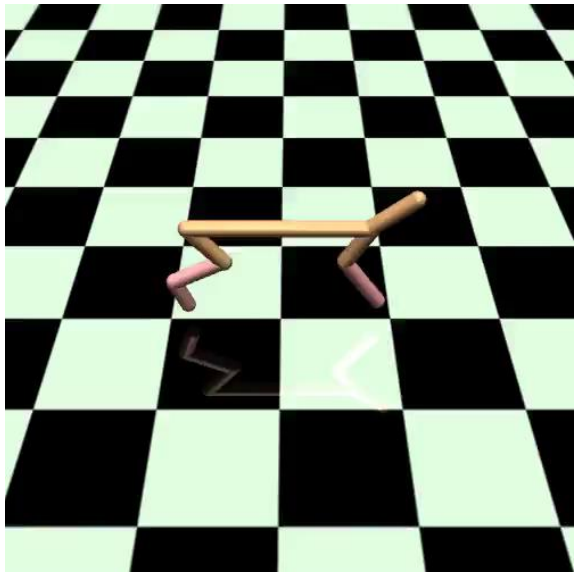
Model-agnostic meta-learning: accelerating PG

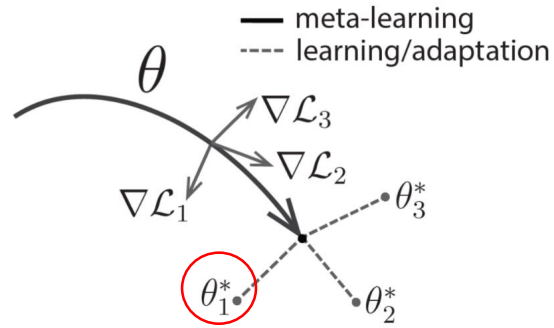
after MAML training



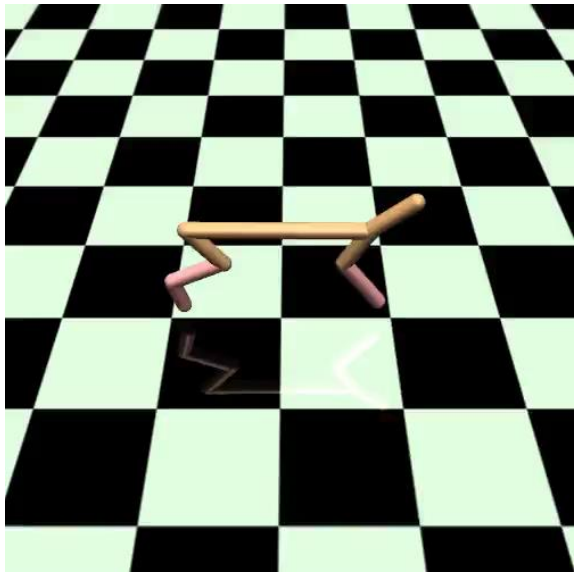


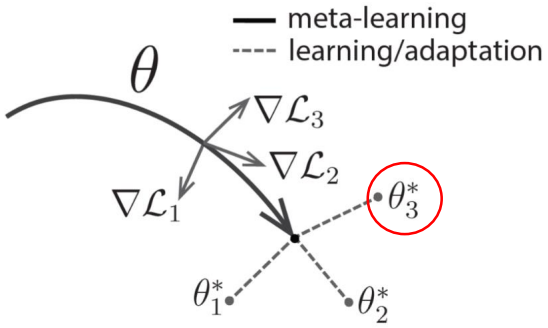
after 1 gradient step (backward reward)





after 1 gradient step (forward reward)





Meta-learning summary & open problems

- Meta-learning = learning to learn
- Supervised meta-learning = supervised learning with datapoints that are entire datasets
- RL meta-learning with RNN policies
  - Ingest past experience with RNN
  - Simply run forward pass at test time to “learn”
  - Just contextual policies (no actual learning)
- Model-agnostic meta-learning
  - Use gradient descent (e.g., policy gradient) learning rule
  - Conceptually not that different
  - …but can accelerate standard RL algorithms (e.g., learn in one iteration of PG)

Meta-learning summary & open problems

- The promise of meta-learning: use past experience to simply acquire a much more efficient deep RL algorithm
- The reality of meta-learning: mostly works well on smaller problems
- …but getting better all the time
- Main limitations
  - RNN policies are extremely hard to train, and likely not scalable
  - Model-agnostic meta-learning presents a tough optimization problem
  - Designing the right task distribution is hard
  - Generally very sensitive to task distribution (meta-overfitting)

Parallelism in RL

Overview

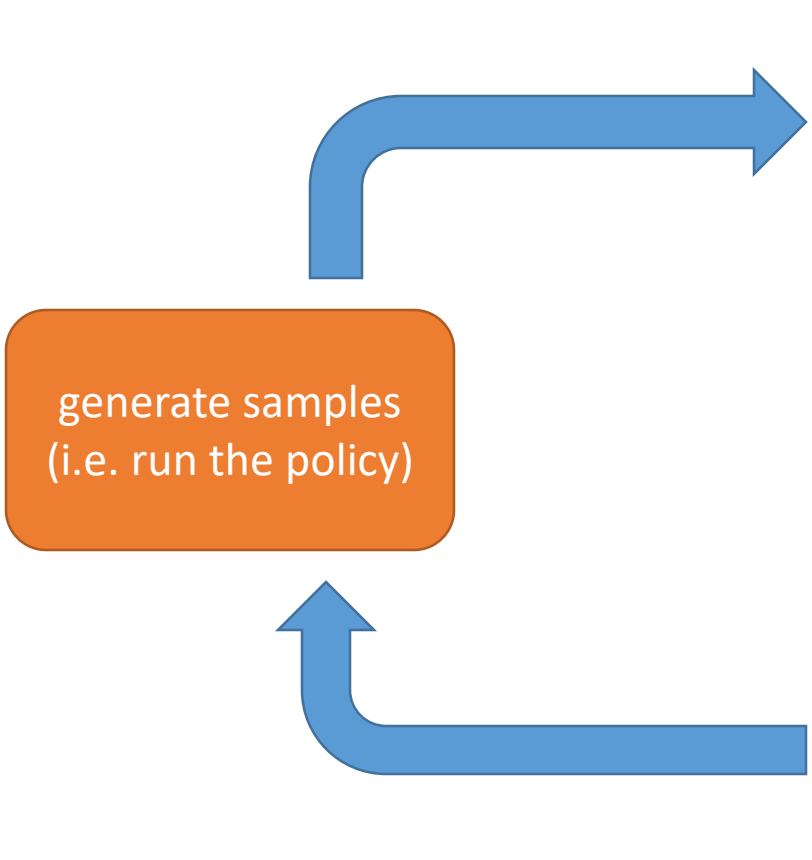
1. We learned about a number of policy search methods
2. These algorithms have all been *sequential*
3. Is there a natural way to parallelize RL algorithms?
   - Experience sampling vs learning
   - Multiple learning threads
   - Multiple experience collection threads

Today's Lecture

1. What can we parallelize?
2. Case studies: specific parallel RL methods
3. Tradeoffs & considerations

- Goals
  - Understand the high-level anatomy of reinforcement learning algorithms
  - Understand standard strategies for parallelization
  - Tradeoffs of different parallel methods

High-level RL schematic



estimate $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ (model-based)

compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

fit $Q_\phi(\mathbf{s},\mathbf{a})$ (actor-critic, Q-learning)

optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

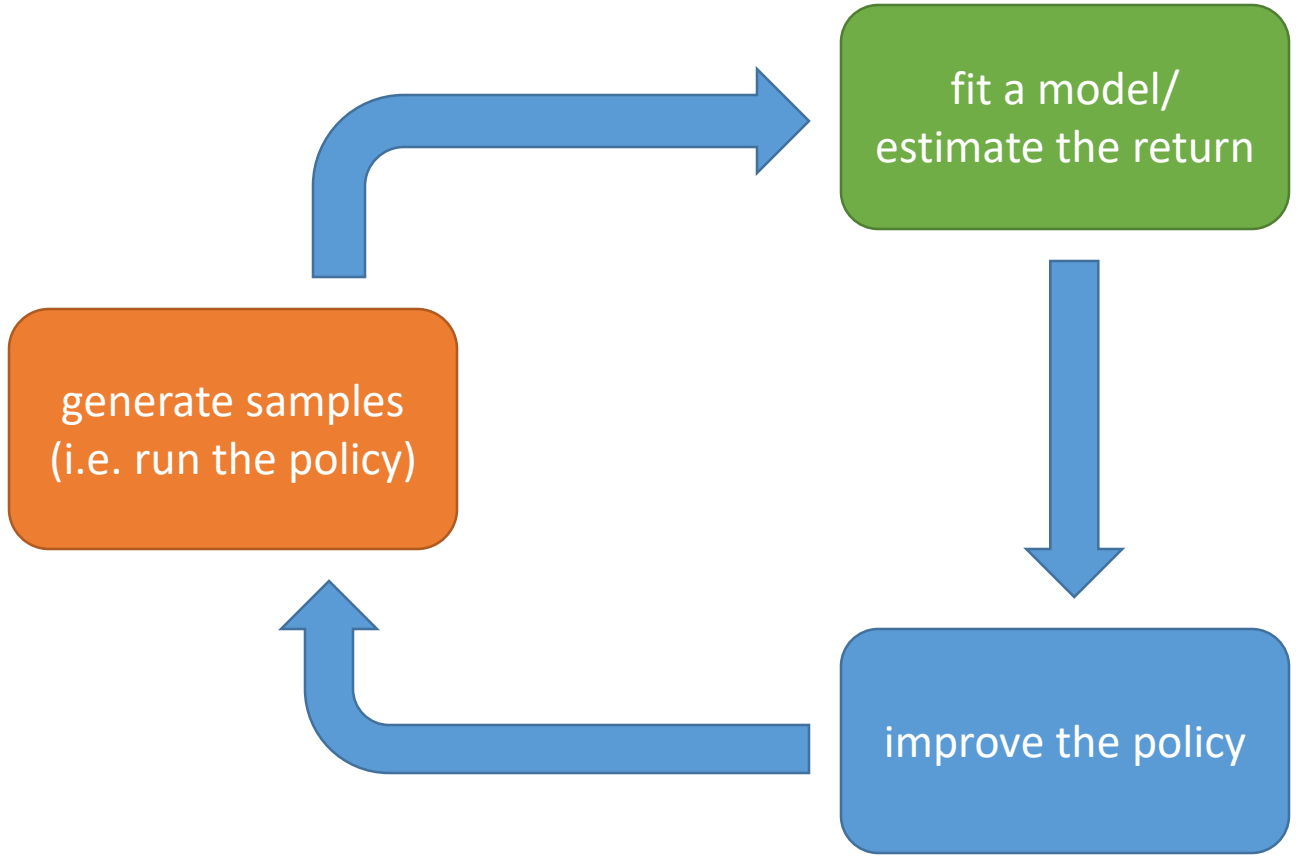
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s},\mathbf{a})$ (Q-learning)

Which parts are slow?

real robot/car/power grid/whatever:
1x real time, until we invent time travel

MuJoCo simulator:
up to 10000x real time



$\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

trivial, fast

$\text{fit } Q_\phi(\mathbf{s}, \mathbf{a})$

expensive, but non-trivial to parallelize

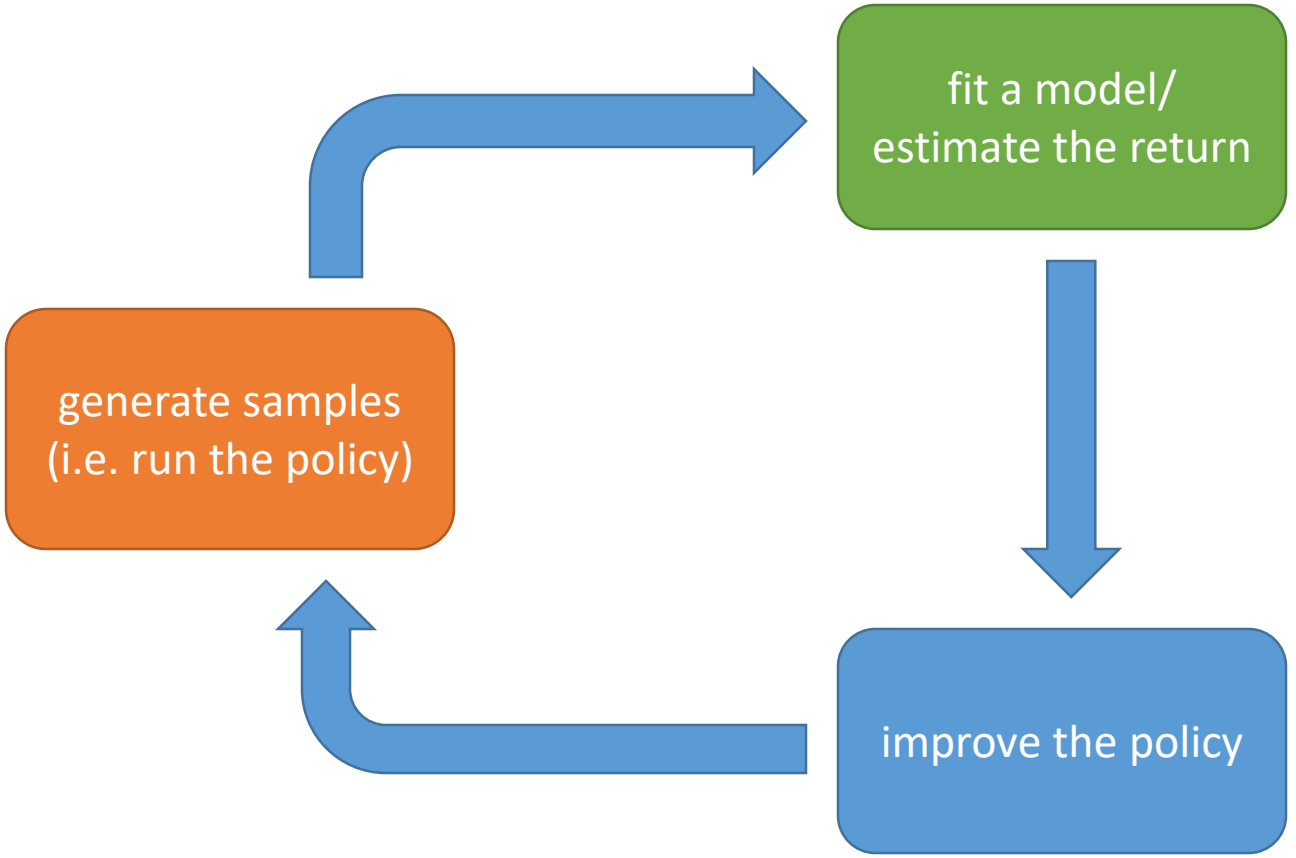
$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$

trivial, nothing to do

$\text{optimize } \pi_\theta(\mathbf{a}|\mathbf{s}) \text{ (model-based)}$

expensive, but non-trivial to parallelize

Which parts can we parallelize?



$\text{fit } Q_\phi(\mathbf{s}, \mathbf{a})$

parallel SGD

$\text{optimize } \pi_\theta(\mathbf{a}|\mathbf{s}) \text{ (model-based)}$

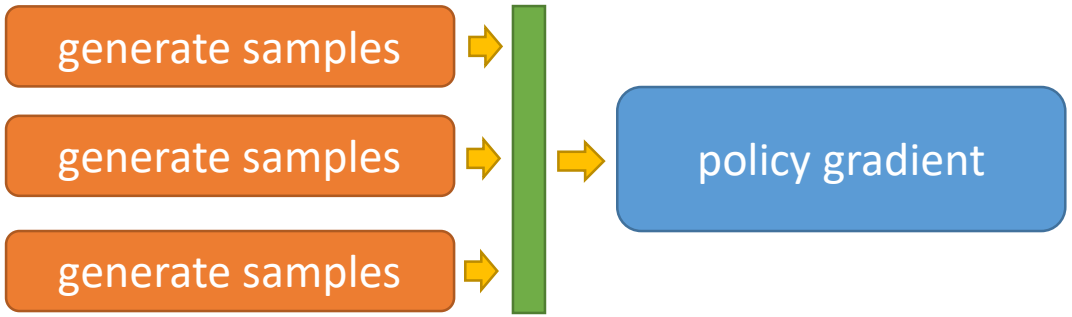
parallel SGD

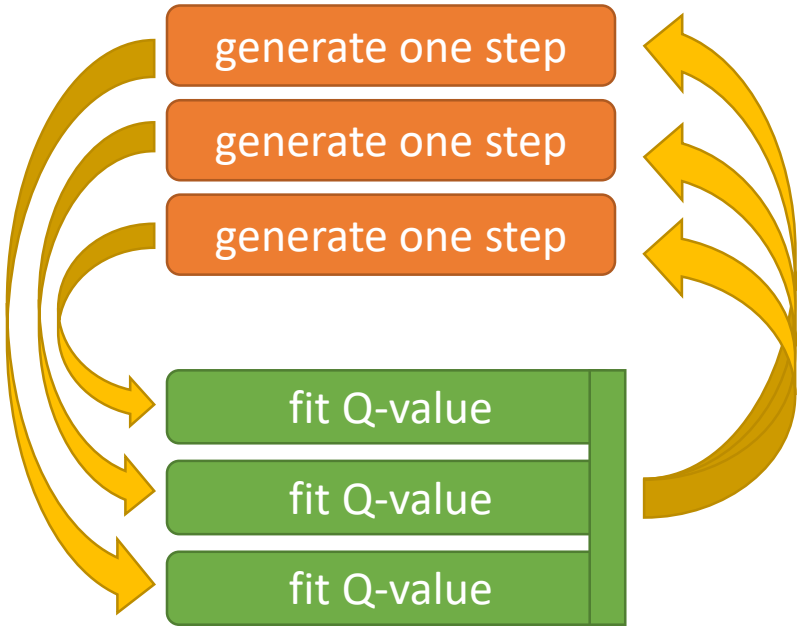
Helps to group data generation and training
(worker generates data, computes gradients, and gradients are pooled)

High-level decisions

1. Online or batch-mode?

2. Synchronous or asynchronous?

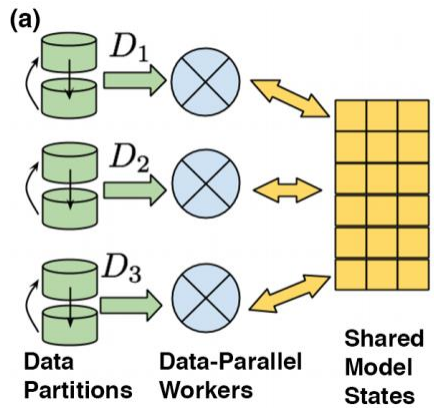




Relationship to parallelized SGD

fit a model/
estimate the return

improve the policy

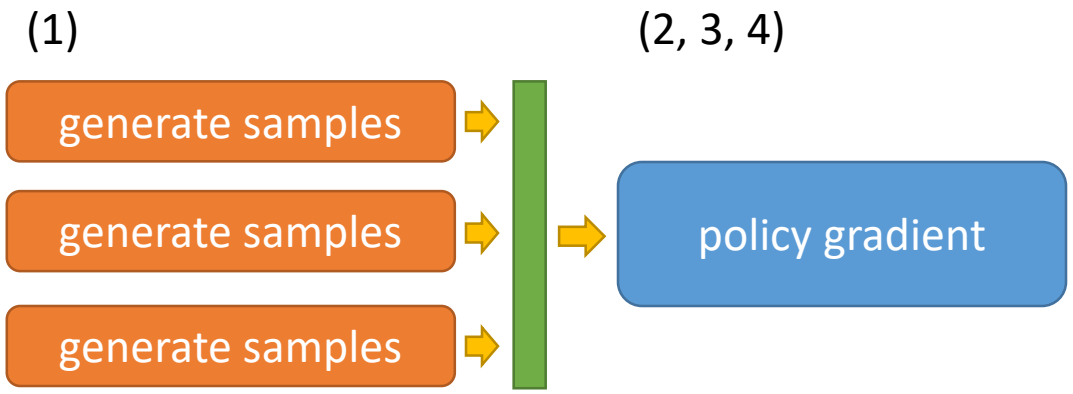


Dai et al. ‘15

1. Parallelizing model/critic/actor training typically involves parallelizing SGD
2. Simple parallel SGD:
    1. Each worker has a different slice of data
    2. Each worker computes gradients, sums them, sends to parameter server
    3. Parameter server sums gradients from all workers and sends back new parameters
3. Mathematically equivalent to SGD, but not asynchronous (communication delays)
4. Async SGD typically does not achieve perfect parallelism, but lack of locks can make it much faster
5. Somewhat problem dependent

Simple example: sample parallelism with PG

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)(r_i - b)$
4. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

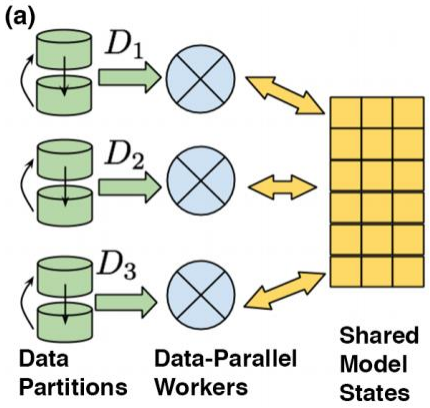


Simple example: sample parallelism with PG

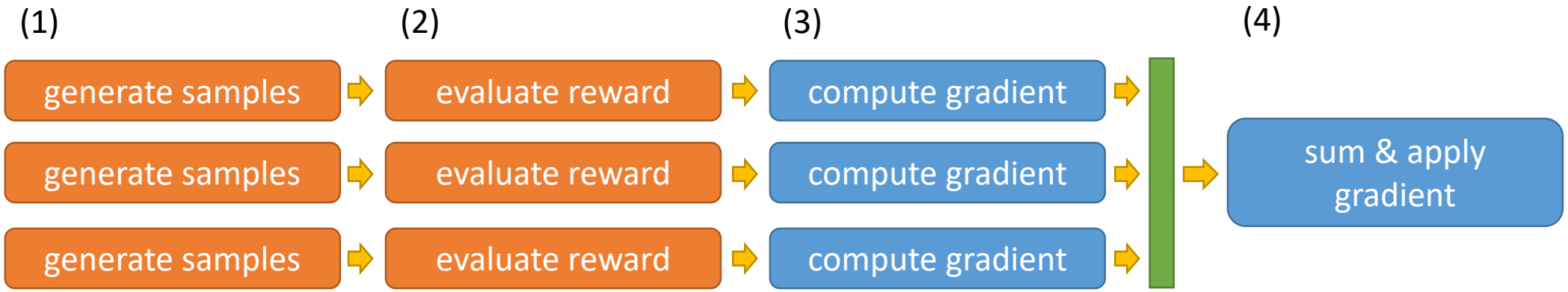
1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)(r_i - b)$
4. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

Simple example: sample parallelism with PG

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)(r_i - b)$
4. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

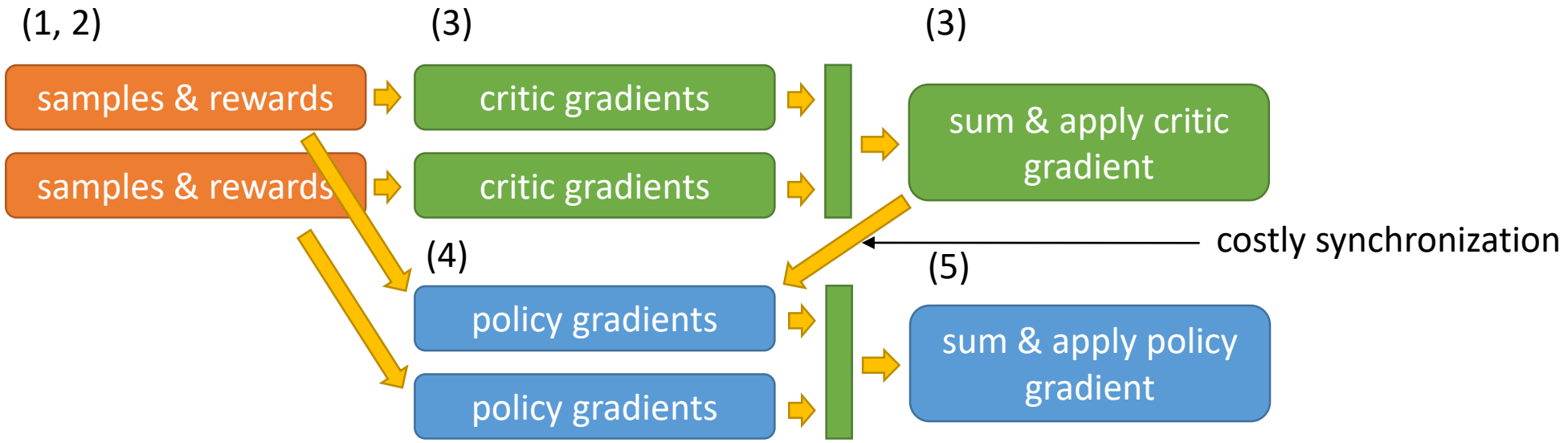


Dai et al. '15



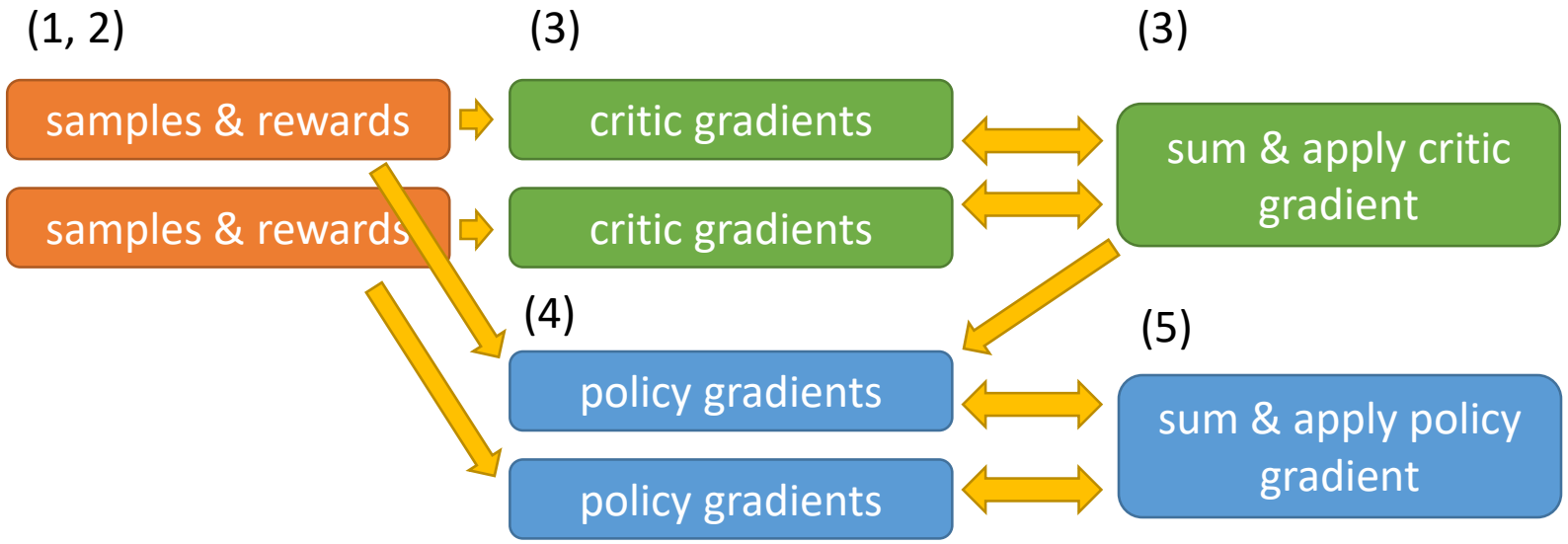
What if we add a critic?

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. update $\hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$ with regression to target values ← see John's actor-critic lecture for what the options here are
4. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$



What if we add a critic?

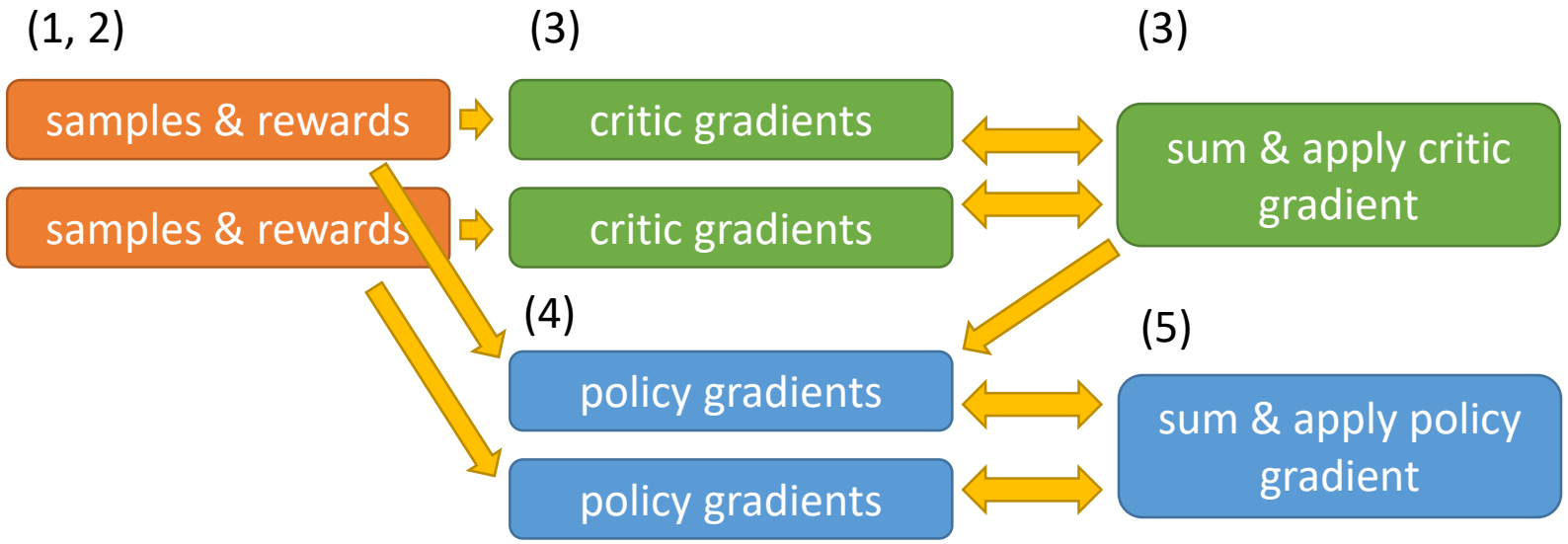
1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. update $\hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$ with regression to target values ← see John's actor-critic lecture for what the options here are
4. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$



What if we run online?

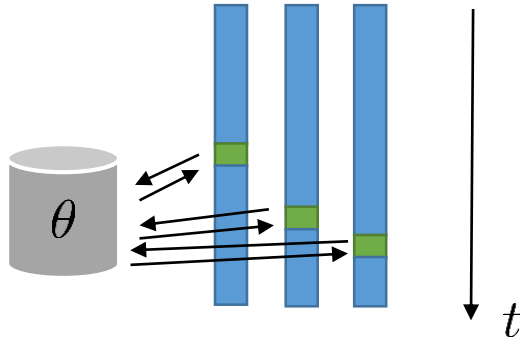
1. collect sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ by running $\pi_\theta(\mathbf{a}|\mathbf{s})$ for 1 step
2. compute $r_i = r(\mathbf{s}_i, \mathbf{a}_i)$
3. update $\hat{A}_\phi(\mathbf{s}^i_t, \mathbf{a}^i_t)$ with regression to target values
4. compute $\nabla_i = \nabla_\theta \log \pi_\theta(\mathbf{a}^i|\mathbf{s}^i)\hat{A}_\phi(\mathbf{s}^i, \mathbf{a}^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

only the parameter update
requires synchronization (actor + critic params)



Actor-critic algorithm: A3C

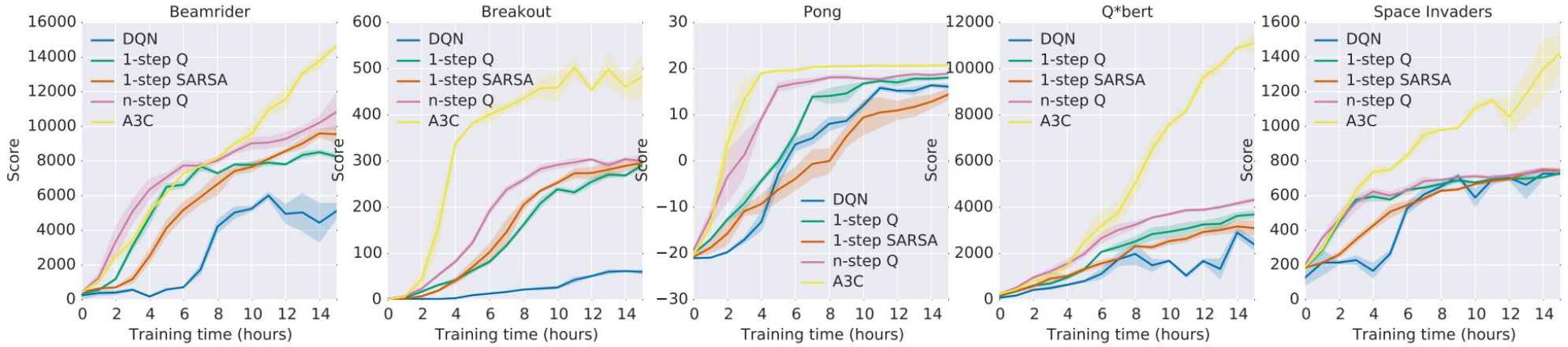
1. collect sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ by running $\pi_\theta(\mathbf{a}|\mathbf{s})$ for 1 step
2. compute $r_i = r(\mathbf{s}_i, \mathbf{a}_i)$
3. update $\hat{A}_\phi(\mathbf{s}^i_t, \mathbf{a}^i_t)$ with regression to target values
4. compute $\nabla_i = \nabla_\theta \log \pi_\theta(\mathbf{a}^i|\mathbf{s}^i)\hat{A}_\phi(\mathbf{s}^i, \mathbf{a}^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$ (only do this every n steps)

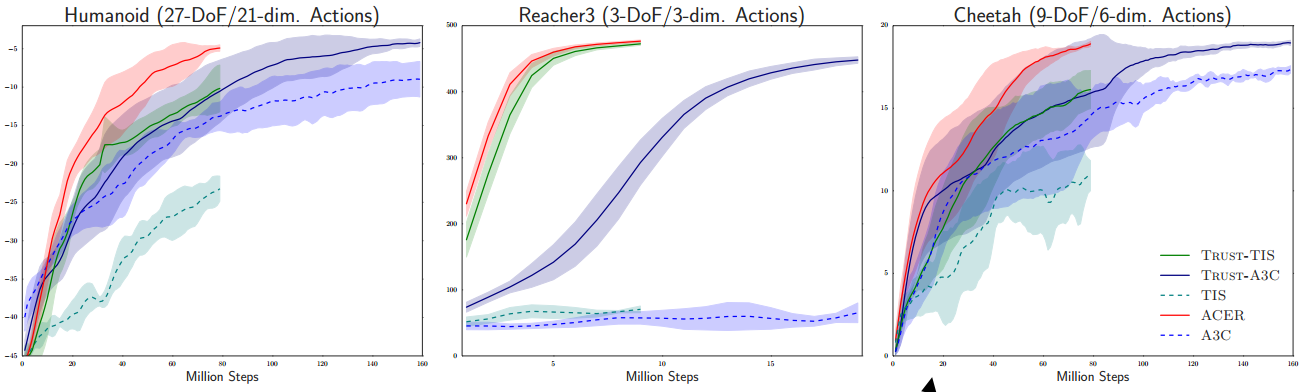


- Some differences vs DQN, DDPG, etc:
  - No replay buffer, instead rely on diversity of samples from different workers to decorrelate
  - Some variability in exploration between workers
- Pro: generally much faster in terms of wall clock
- Con: generally must slower in terms of # of samples (more on this later…)

Mnih et al. ‘16

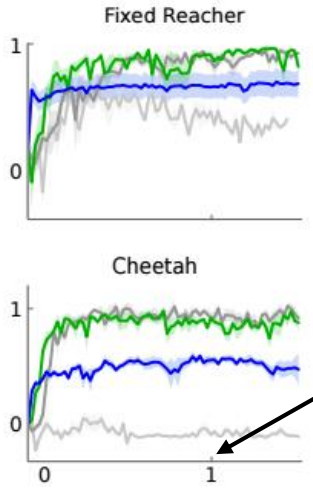
Actor-critic algorithm: A3C





20,000,000 steps

DDPG:

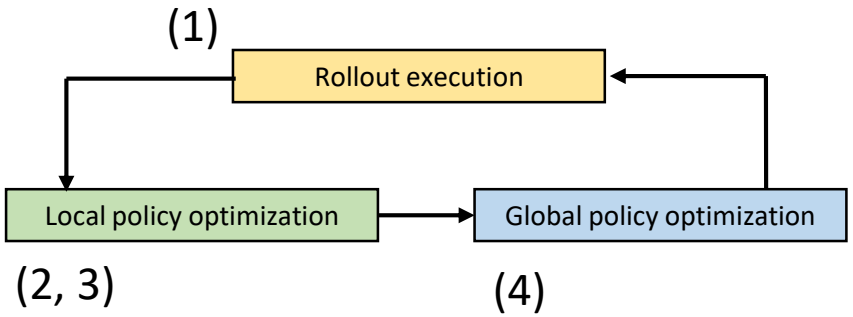


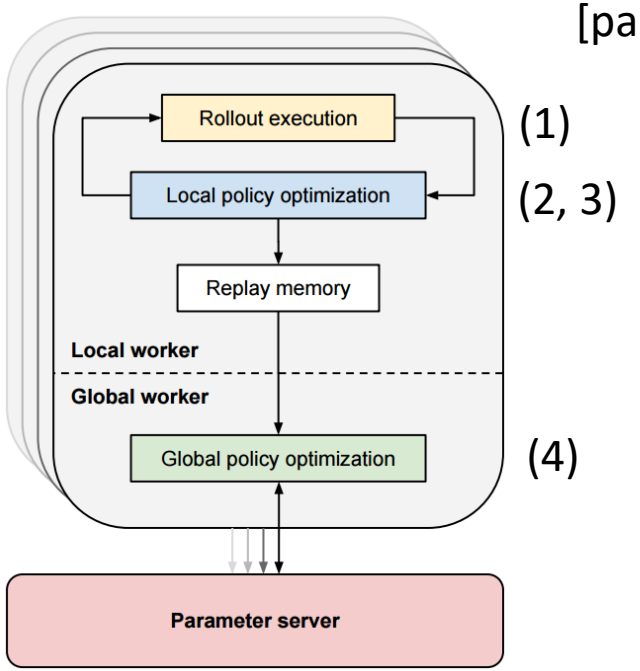
1,000,000 steps

more on this later…

Model-based algorithms: parallel GPS

1. get N samples τ_i by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times for each initial state $\mathbf{s}_0^j$ [parallelize sampling]
2. fit local models for each initial state [parallelize dynamics]
3. use LQR to get updated local policies $p_j(\mathbf{a}_t|\mathbf{s}_t)$ for each initial state $\mathbf{s}_0^j$ [parallelize LQR]
4. update policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ by imitating all $p_j(\mathbf{a}_t|\mathbf{s}_t)$ [parallelize SGD]

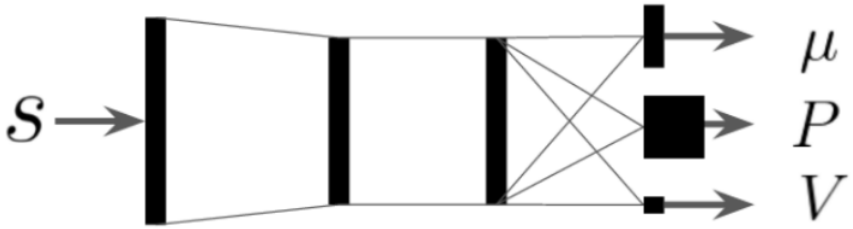




Yahya, Li, Kalakrishnan, Chebotar, L., ‘16

Model-based algorithms: parallel GPS

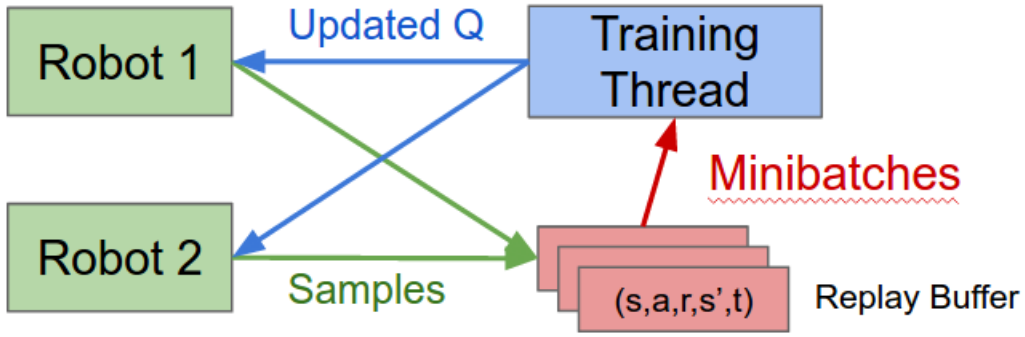
Real-world model-free deep RL: parallel NAF

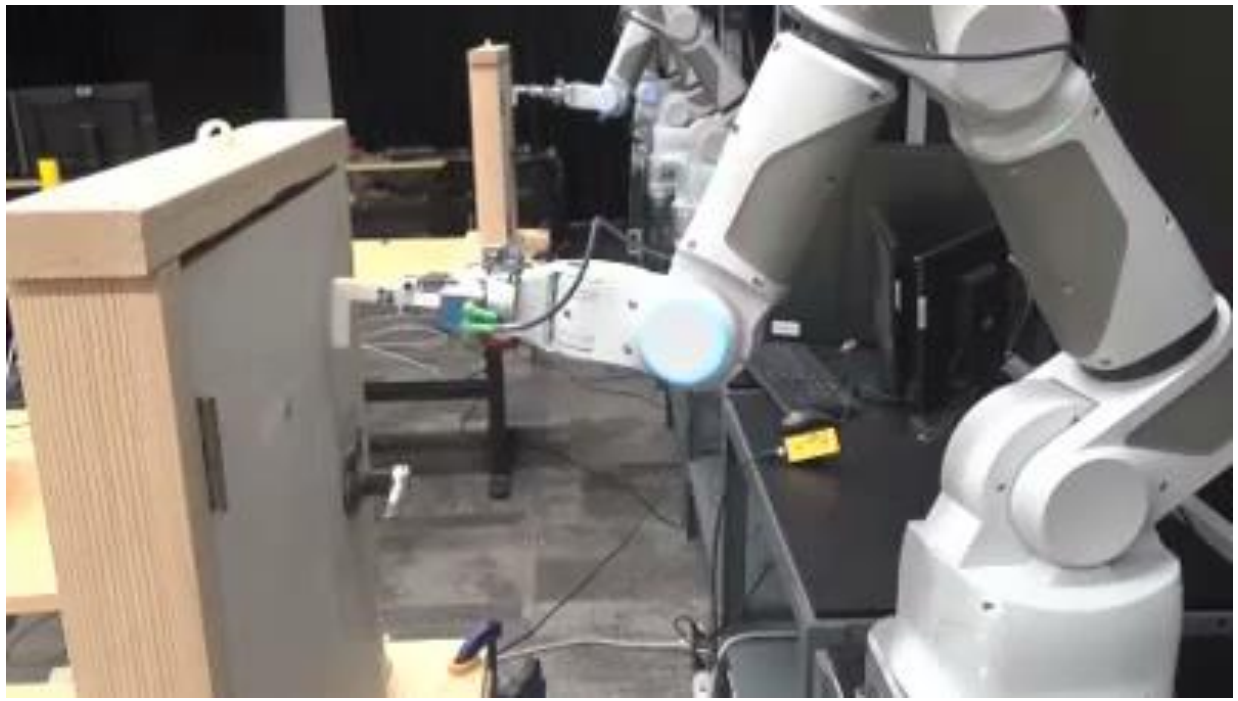


NAF Architecture.

$$Q(\boldsymbol{x},\boldsymbol{u}|\theta^Q) = A(\boldsymbol{x},\boldsymbol{u}|\theta^A) + V(\boldsymbol{x}|\theta^V)$$

$$A(\boldsymbol{x},\boldsymbol{u}|\theta^A) = -\frac{1}{2}(\boldsymbol{u}-\boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu))^T \boldsymbol{P}(\boldsymbol{x}|\theta^P)(\boldsymbol{u}-\boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu))$$





Gu*, Holly*, Lillicrap, L., ‘16

Simplest example: sample parallelism with off-policy algorithms