Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

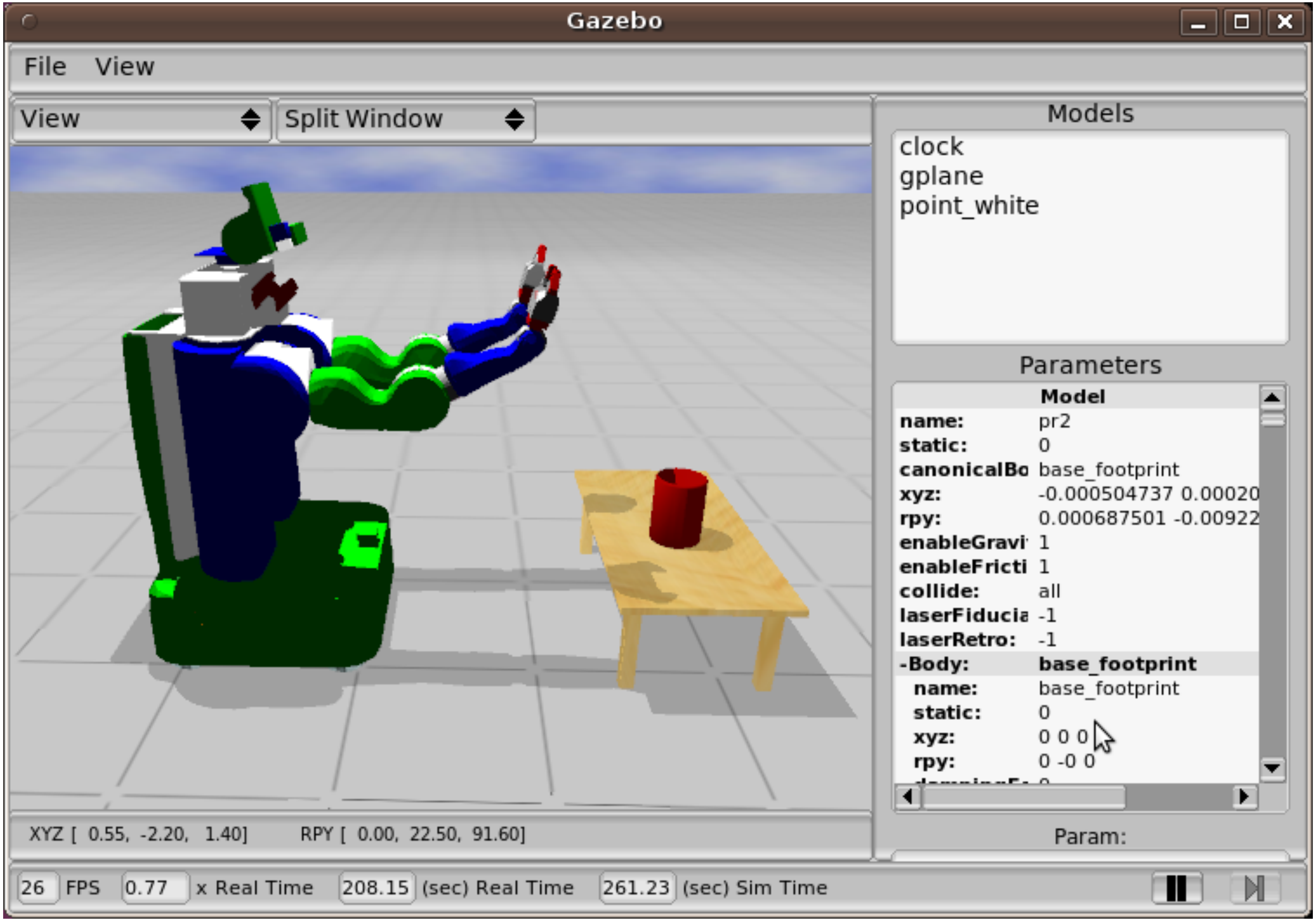
Sim2Real

Katerina Fragkiadaki

So far

The requirement of large number of samples for RL, only possible in simulation, renders RL a model-based framework, we can't really rely (solely) on interaction in the real world (as of today)

Simulation



Gazebo
File View
View
Split Window
Models
clock
gplane
point_white
Parameters
Model
name: pr2
static: 0
canonicalBo base_footprint
xyz: -0.000504737 0.00020
rpy: 0.000687501 -0.00922
enableGravi 1
enableFricti 1
collide: all
laserFiducia -1
laserRetro: -1
-Body: base_footprint
name: base_footprint
static: 0
xyz: 0 0 0
rpy: 0 -0 0
XYZ [0.55, -2.20, 1.40] RPY [0.00, 22.50, 91.60]
Param:
26 FPS 0.77 x Real Time 208.15 (sec) Real Time 261.23 (sec) Sim Time

Pros of Simulation

- We can afford many more samples!
- Safety
- Avoids wear and tear of the robot
- We do not need to rely on demonstrations (often too many are needed)
- Good at rigid multibody dynamics

Cons of Simulation

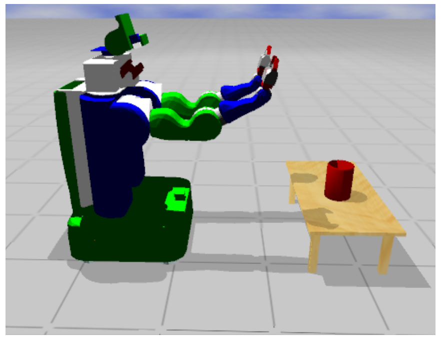
- Under-modeling: many physical events are not modeled.
- Wrong parameters. Even if our physical equations were correct, we would need to estimate the right parameters, e.g., inertia, frictions (system identification).
- Systematic discrepancy w.r.t. the real world regarding:
  - observations
  - dynamics

as a result, policies that learnt in simulation do not transfer to the real world

- Hard to simulate deformable objects (finite element methods are very computational intensive)

Cons of Simulation

- Under-modeling: many physical events are not modeled.
- Wrong parameters. Even if our physical equations were correct, we would need to estimate the right parameters, e.g., inertia, frictions (system identification).
- Systematic discrepancy w.r.t. the real world regarding:
  - observations
  - dynamics



as a result, policies that learnt in simulation do not transfer to the real world

- Hard to simulate deformable objects (finite element methods are very computational intensive)

This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - synthetic data randomization
  - feature fine-tuning
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - synthetic data randomization
  - feature fine-tuning
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors

Justin Fu, Sergey Levine, Pieter Abbeel

Algorithm 1 Model-based reinforcement learning with online adaptation

```
for time step t = 1 to T do
    Observe state x_t
    Update μ̂_t and Δ_t via Equations (3) and (4)
    Compute Σ̂_t = Δ_t − μ̂_t μ̂_t^T
    Evaluate prior to obtain Φ, μ_0, m, and n_0 (see Section V)
    Update β and N as described in Equation (5)
    Compute μ and Σ via Equation (1)
    Compute f_xt, f_ut, f_ct, and F_t from μ and Σ via Equation (2)
    Run LQR to compute K_t, k_t, and Q_u,ut
    Sample u_t from N(û_t + k_t + K_t(x_t − x̂_t), Q_u,ut^{-1})
    Take action u_t
end for
```

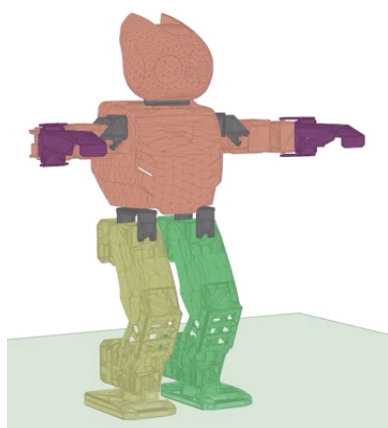
Combining Model-Based Policy Search with Online Model Learning for Control of Physical Humanoids

Igor Mordatch, Nikhil Mishra, Clemens Eppner, Pieter Abbeel
Department of Computer Science and Engineering, University of California, Berkeley, CA, USA.

- Hierarchical control for better sim2real transfer: high-level controllers determine the trajectory, low-level controllers produce the required torques.
- Adapt a dynamics model online during actual task execution for the low level controllers.

Combining Model-Based Policy Search with Online Model Learning for Control of Physical Humanoids

Igor Mordatch, Nikhil Mishra, Clemens Eppner, Pieter Abbeel
Department of Computer Science and Engineering, University of California, Berkeley, CA, USA.



Offline:

Train policy to output *desired* next state:

$\bar{\mathbf{x}}^{t+1}$ Joint Angles, IMU, Forces

At every timestep:

Learn robot dynamics on the fly from past observations

$$\mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Query policy for $\bar{\mathbf{x}}^{t+1}$

Solve for robot torques $\mathbf{u}^*$ such that

$$\bar{\mathbf{x}}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^*)$$

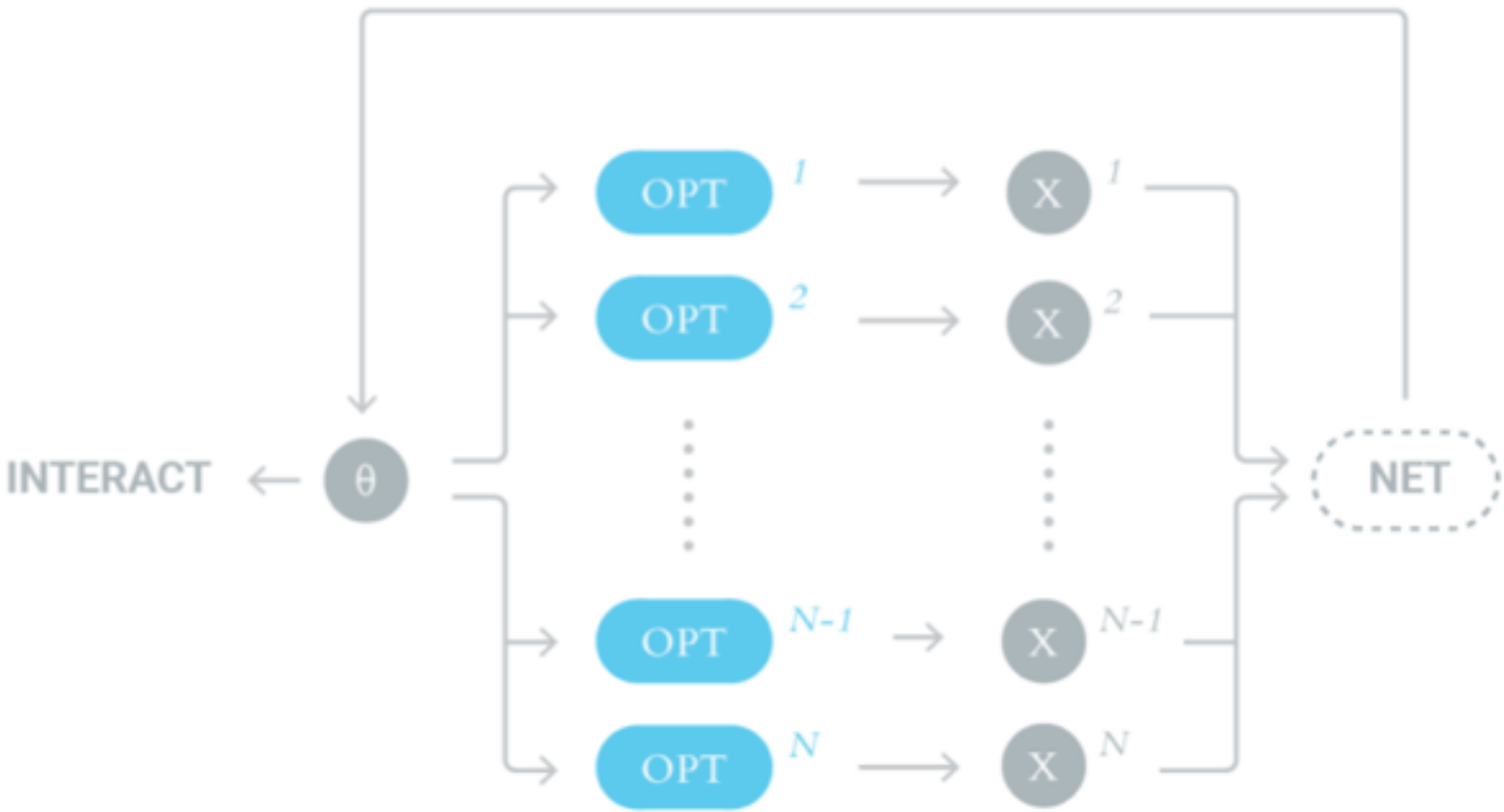
Training the high level policy

- Sensory state: accelerations measured by IMUs, joint angles and their velocities
- High level policy outputs joint angles and their velocities instead of torques
- Learn policy in simulation using guidance from trajectory optimization:

$$\underset{\theta\ \mathbf{X}^1\ \ldots\ \mathbf{X}^N}{\text{minimize}} \quad \sum_i C_i(\mathbf{X}^i)$$

$$\text{subject to} \quad \forall\, i,t:\ \mathbf{a}^t(\mathbf{X}^i) = \boldsymbol{\pi}_\theta(\mathbf{s}^t(\mathbf{X}^i))$$

Training the high level policy

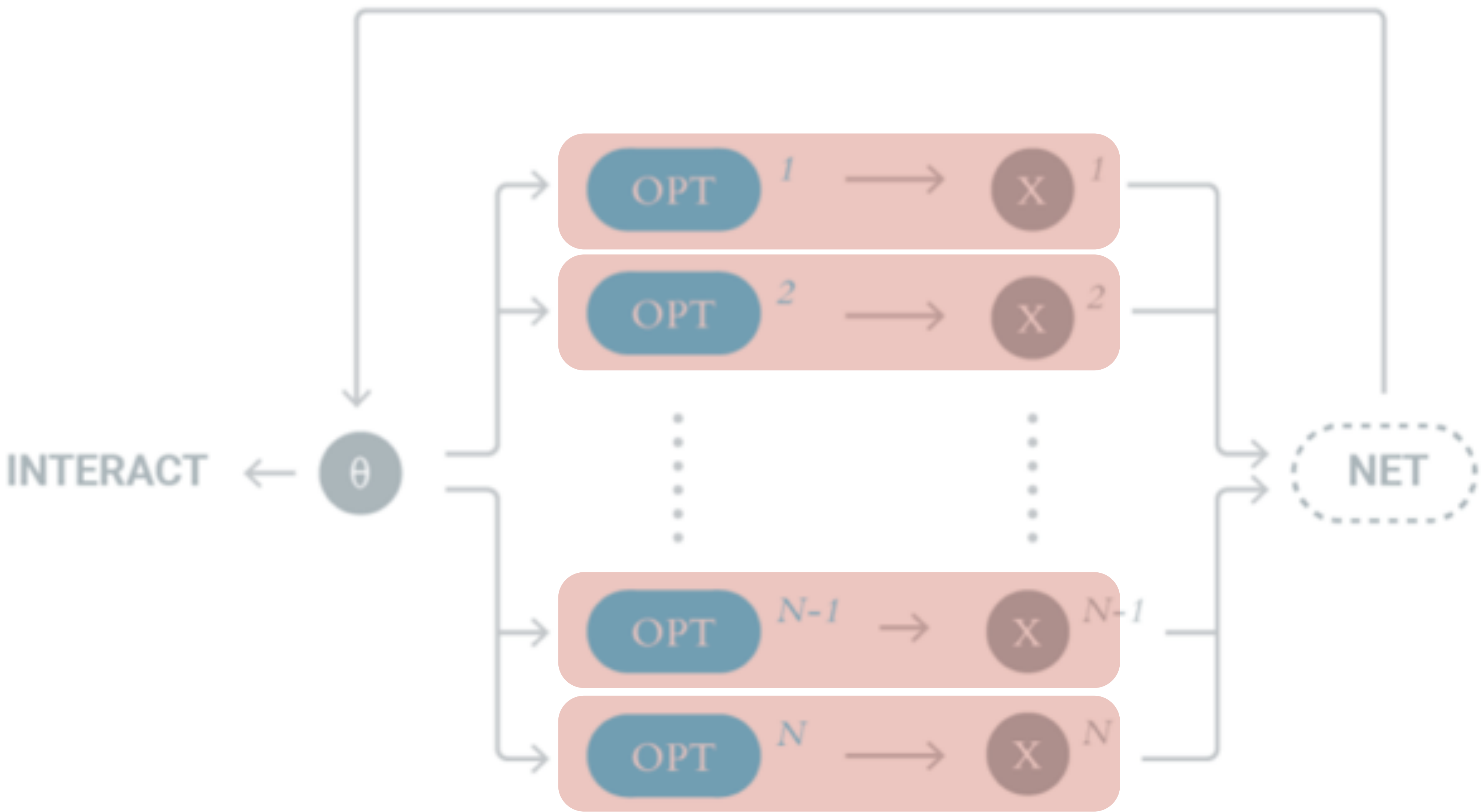


Decompose into:

- trajectory optimizations
- regression

slides: Igor Mordatch

Training the high level policy



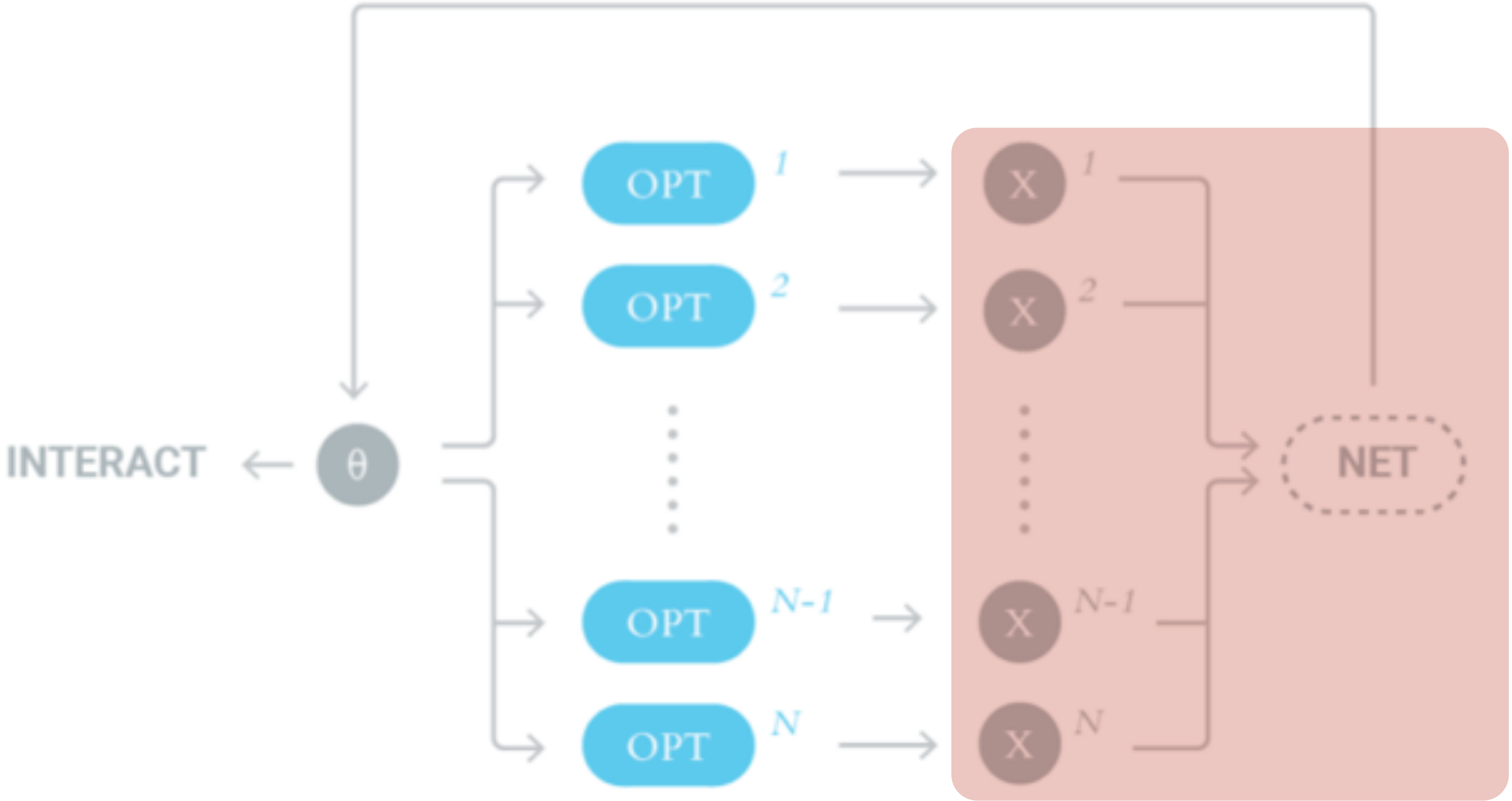
Decompose into:

- trajectory optimizations
- regression

"stay close to policy"

$$\min_{\mathbf{X}} \sum_t C(\mathbf{x}^t) + ||\boldsymbol{\pi}_\theta(\mathbf{x}^t) - \mathbf{u}^t||^2$$

Training the high level policy

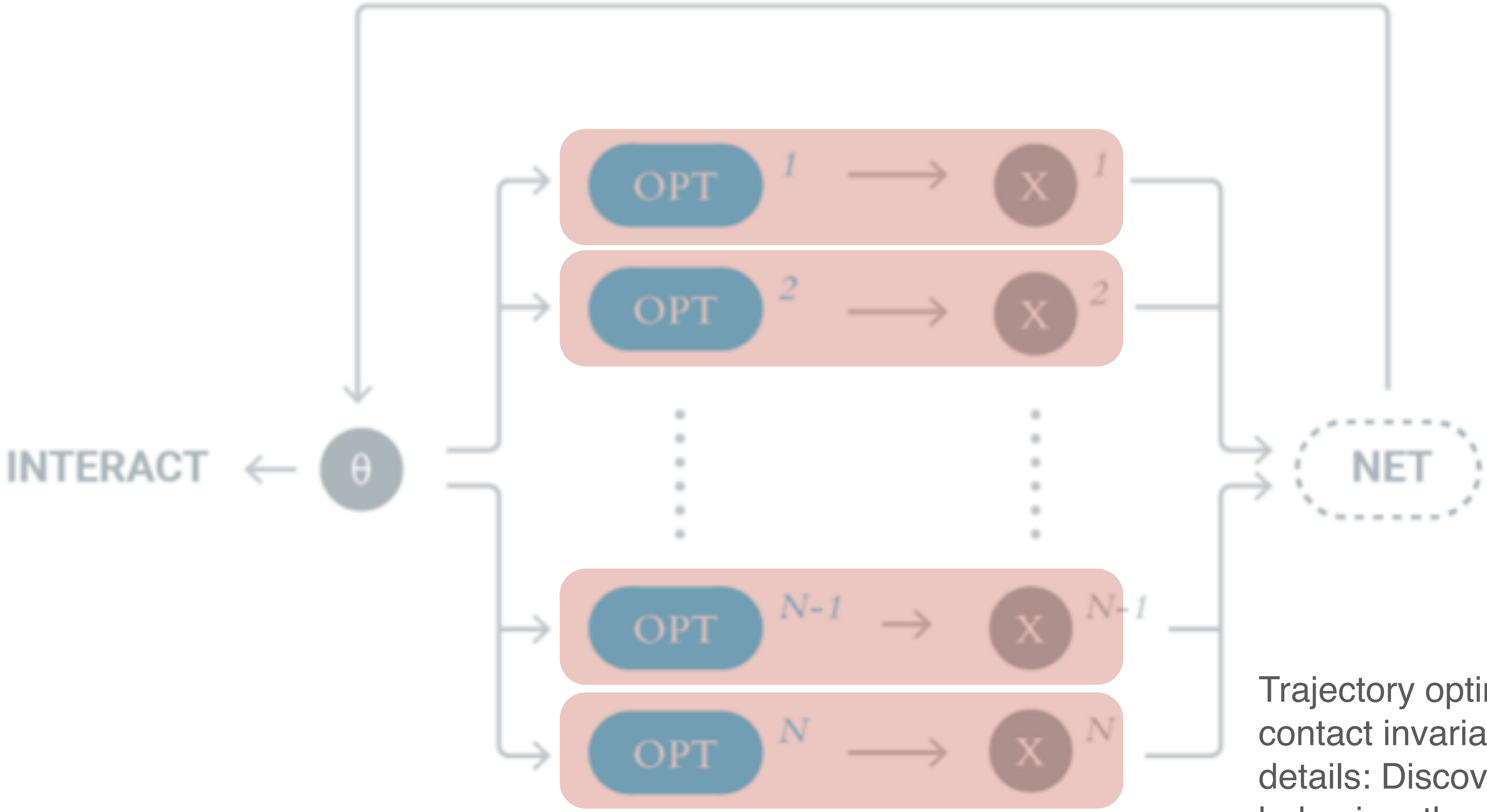


Decompose into:

- trajectory optimizations
- regression

$$\min_{\theta} \sum_{i,t} ||\boldsymbol{\pi}_{\theta}(\mathbf{x}^{i,t}) - \mathbf{u}^{i,t}||^2$$

Training the high level policy



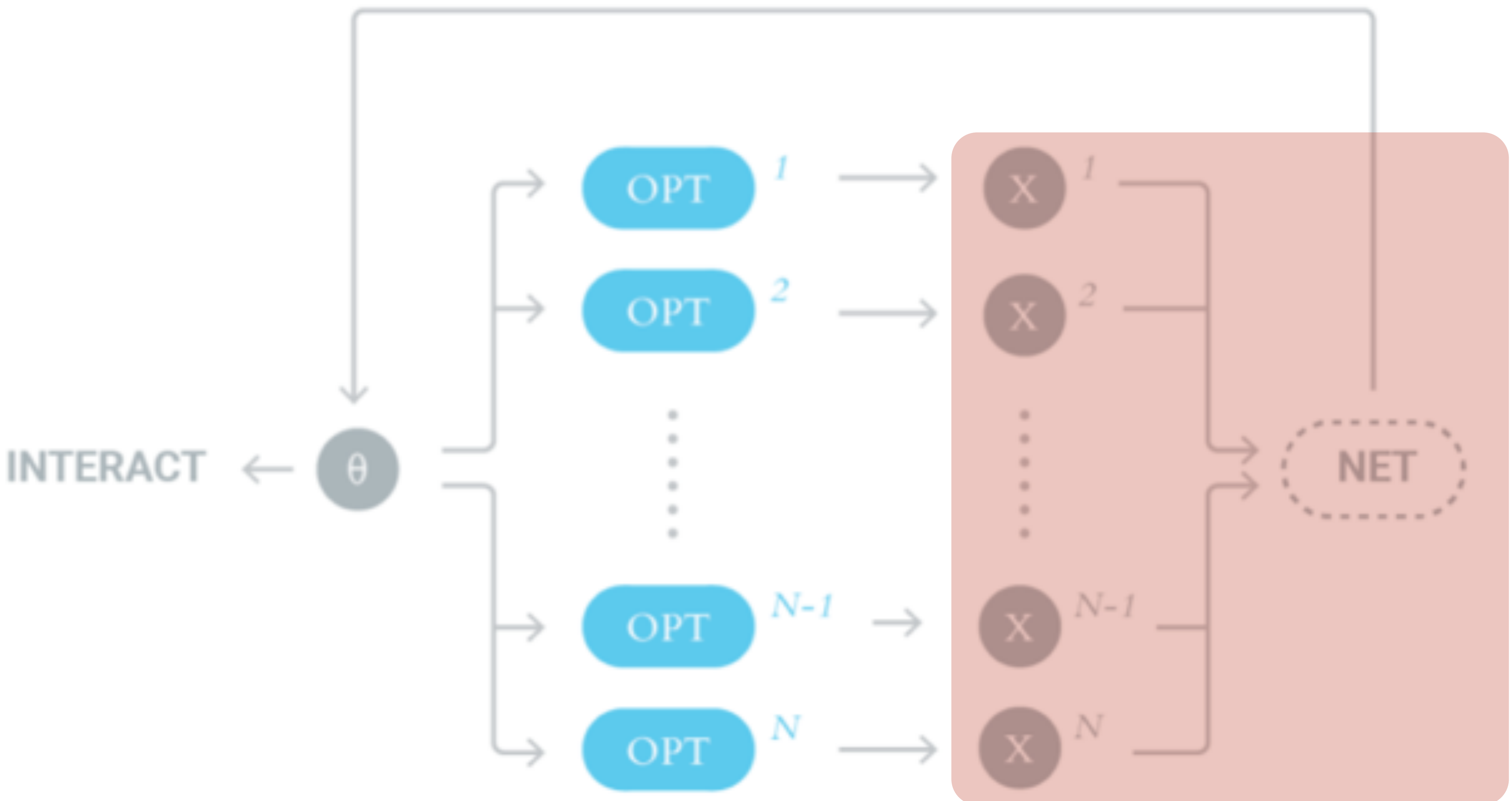
Trajectory optimization used: contact invariant optimization, for details: Discovery of complex behaviors through contact invariant optimization, Mordatch et al. 2012

Decompose into:

- trajectory optimizations
- regression

$$\min_{\mathbf{X}} \sum_t C(\mathbf{x}^t) + ||\boldsymbol{\pi}_\theta(\mathbf{x}^t) - \mathbf{u}^t||^2$$

Training the high level policy



Decompose into:

- trajectory optimizations
- regression

$$\min_{\theta} \sum_{i,t} ||\boldsymbol{\pi}_{\theta}(\mathbf{x}^{i,t}) - \mathbf{u}^{i,t}||^2$$

Low-level controllers

Learn local forward model:

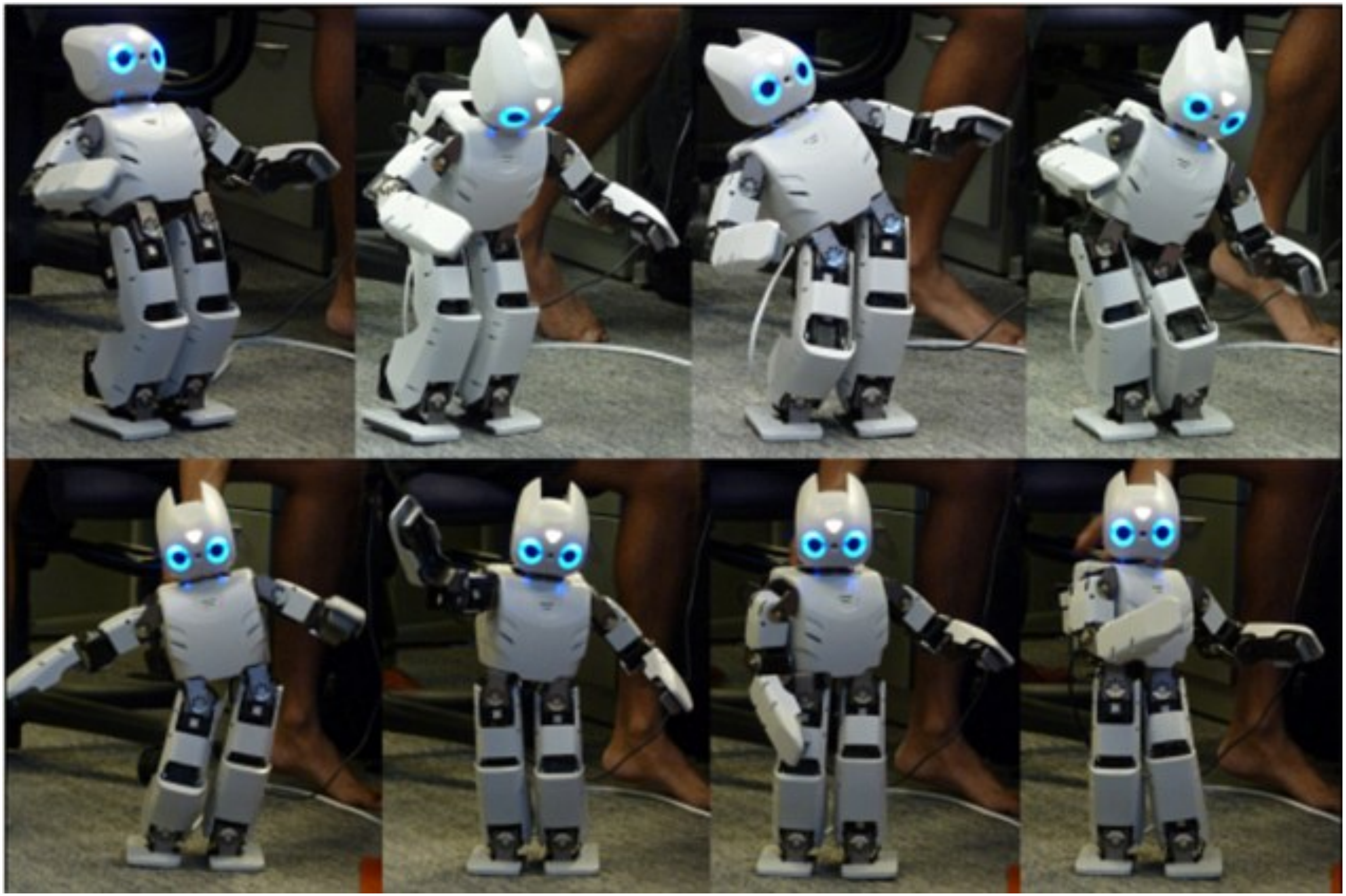
$$\mathbf{s}^{t+1} = \mathbf{g}(\mathbf{s}^t, \mathbf{u}^t)$$

$$\mathbf{J}^t = \underset{\mathbf{J}}{\text{minimize}}\ \frac{1}{2}||\mathbf{J}\,[\ \mathbf{s}\ \mathbf{u}\ ]^{t-1} - \mathbf{s}^t||^2 + \frac{\alpha}{2}||\mathbf{J} - \mathbf{J}^{t-1}||^2$$

$$\mathbf{g}(\mathbf{s}^t, \mathbf{u}^t) = \mathbf{J}_{\mathbf{s}}\mathbf{s}^t + \mathbf{J}_{\mathbf{u}}\mathbf{u}^t$$

Given desired \bar{s}^{t+1} by the high level policy, estimate control u^t:

$$\mathbf{u}^t(\bar{\mathbf{s}}^{t+1}) = \underset{\mathbf{u}}{\text{minimize}}\ \frac{1}{2}\left\|\mathbf{g}(\mathbf{s}^t, \mathbf{u}) - \bar{\mathbf{s}}^{t+1}\right\|^2$$



This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - synthetic data randomization
  - feature fine-tuning
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

Grounded Action Transformation
for Robot Learning in Simulation

Josiah P. Hanna, Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{jphanna,pstone}@cs.utexas.edu

- Idea: bring simulation closer to real world by learning parametrized actions whose execution (in simulation) brings simulation state close to real world state.

Grounded Action Transformation
for Robot Learning in Simulation

Josiah P. Hanna, Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{jphanna,pstone}@cs.utexas.edu

- Idea: bring simulation closer to real world by learning parametrized actions whose execution (in simulation) brings simulation state close to real world state.

Assumes:

- a modifiable simulator with a parametrized transition probabilities $P_{\mathrm{sim}}(\cdot|s, a; \phi)$ where the vector ϕ can be changed to produce in effect a different simulator
- a policy learning procedure (`optimize`) in simulation
- we can evaluate the policy in the real world (physical robot)

Grounded Action Transformation for Robot Learning in Simulation

Josiah P. Hanna, Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{jphanna,pstone}@cs.utexas.edu

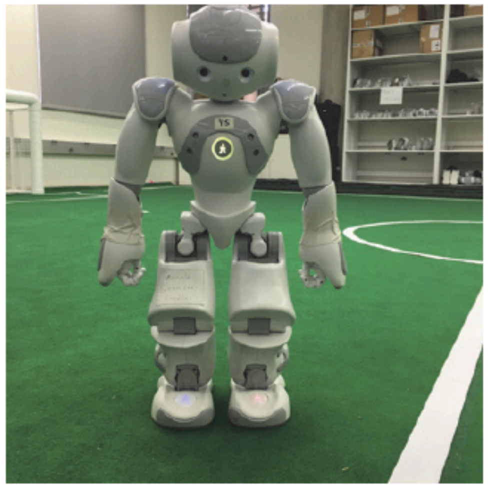
- Idea: bring simulation closer to real world by learning parametrized actions whose execution (in simulation) brings simulation state close to real world state.

Assumes:

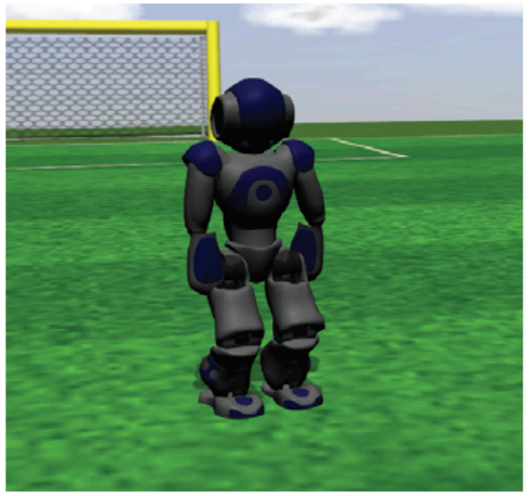
- a modifiable simulator with a parametrized transition probabilities $P_{\text{sim}}(\cdot|s, a; \phi)$ where the vector ϕ can be changed to produce in effect a different simulator
- a policy learning procedure (`optimize`) in simulation
- we can evaluate the policy in the real world (physical robot)

Grounded Simulation learning

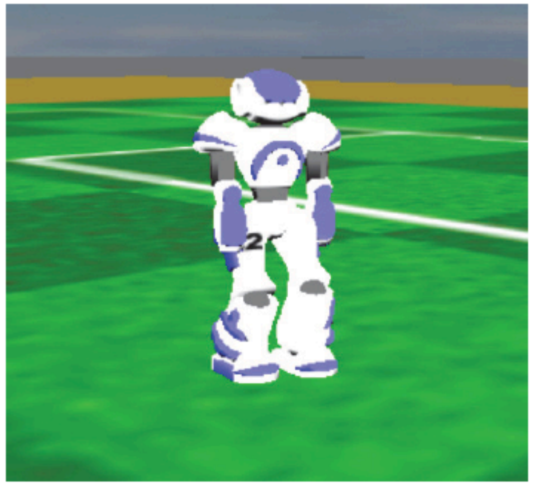
- NAO: humanoid robot with 25 degrees of freedom
- Uses an open source walk engine with 15 parameters (e.g. step height, pendulum model height etc.)
- Simulators used:
  - SimSpark http://simspark.sourceforge.net
  - Gazebo http://gazebosim.org/



(a) A Softbank NAO Robot



(b) A simulated NAO in Gazebo



(c) A simulated NAO in SimSpark

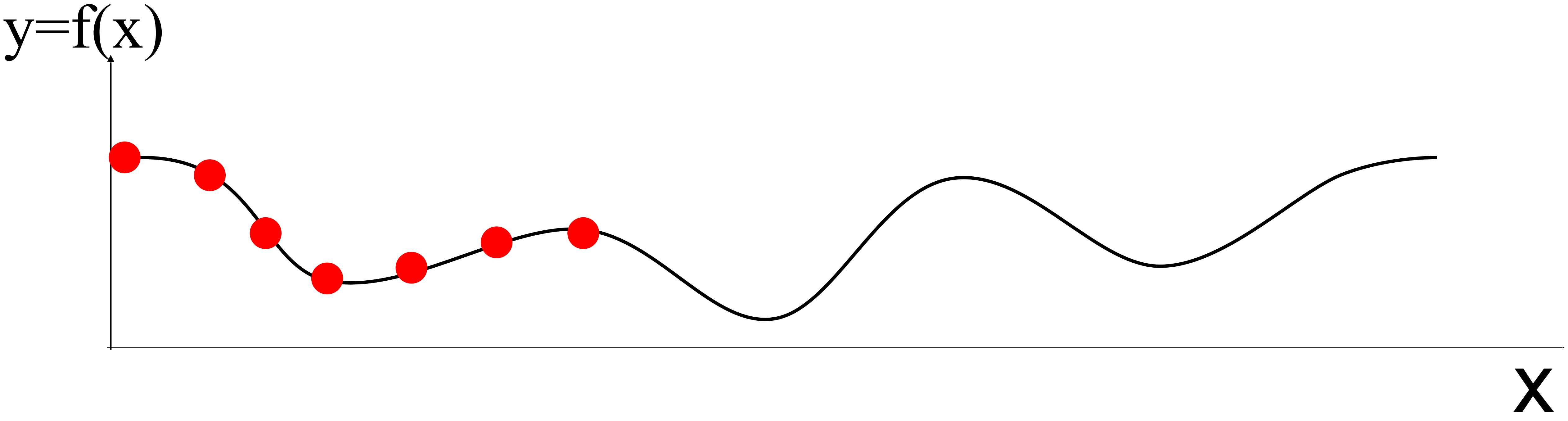
Digression: evolutionary methods for policy search

- Optimization methods that searches for the **optimum** solution in a **search-space without using gradients**

- Evolution strategy steps:
  1. **Generate** a population of candidate solutions
  2. **Evaluate** every individual in the population
  3. **Select** parents from the fittest individuals
  4. **Reproduce** offspring of the next generation (Recombination & mutation)
  5. **Repeat** until a *termination criterion* is met

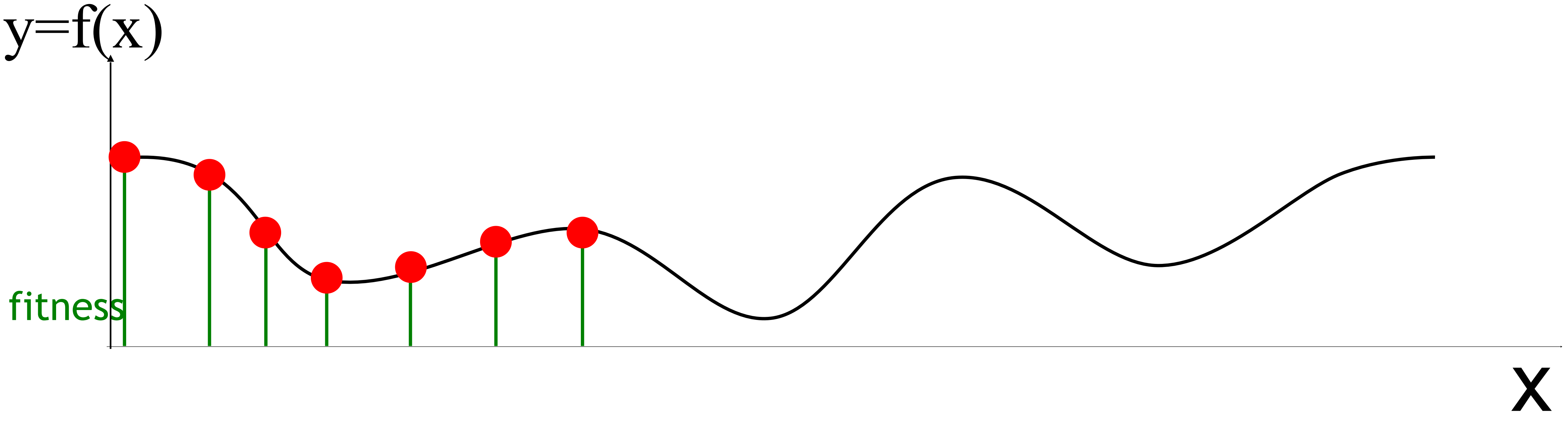
Evolutionary Methods

1. **Generate** a population of candidate solutions



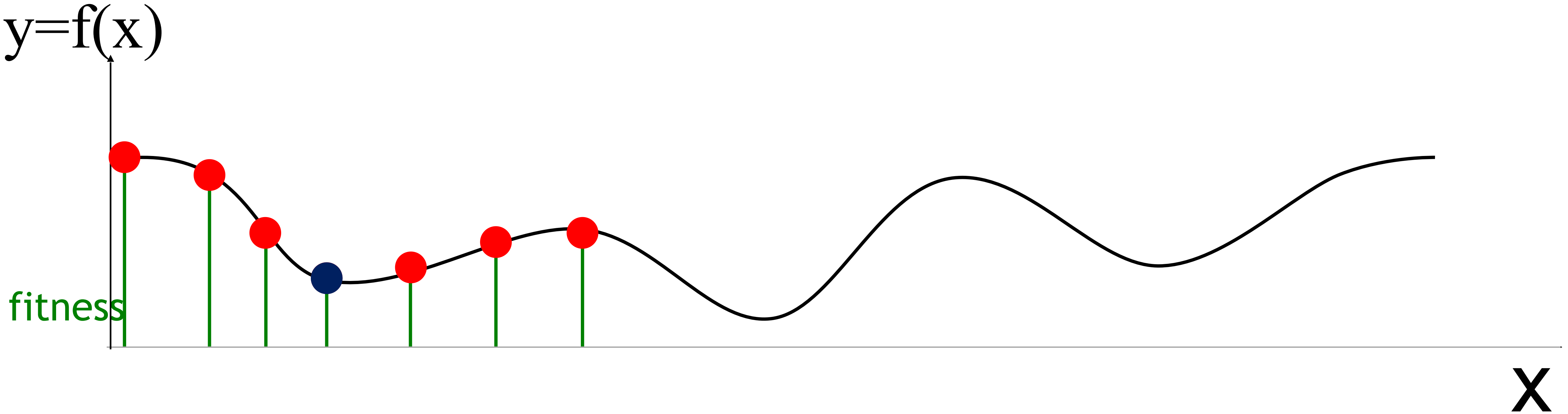
Evolutionary Methods

2. **Evaluate** every individual in the population



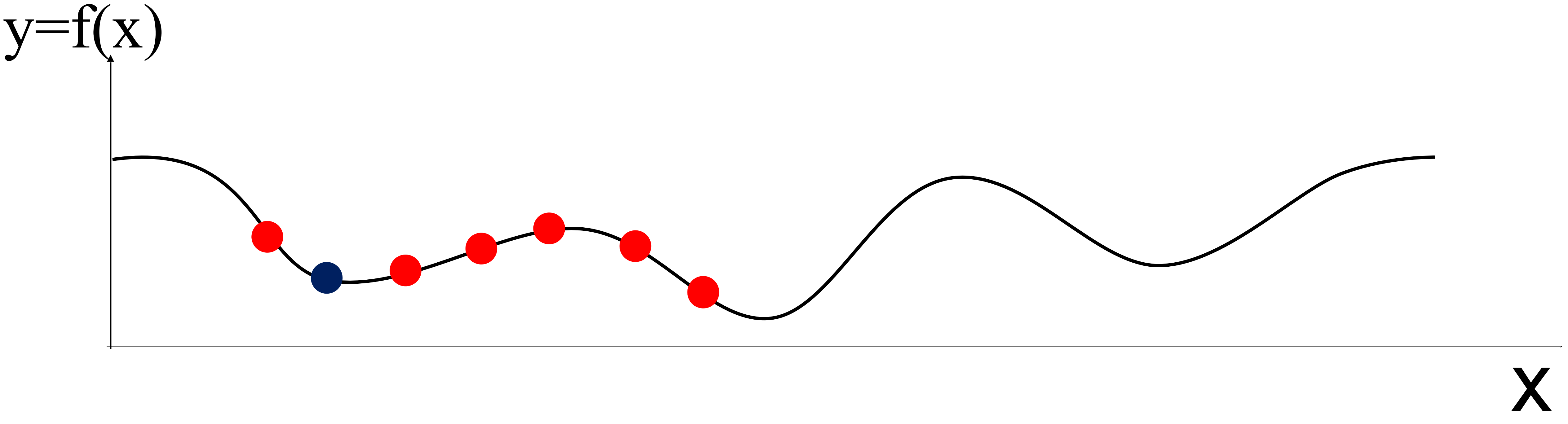
Evolutionary Methods

3. **Select** parents from the fittest individuals



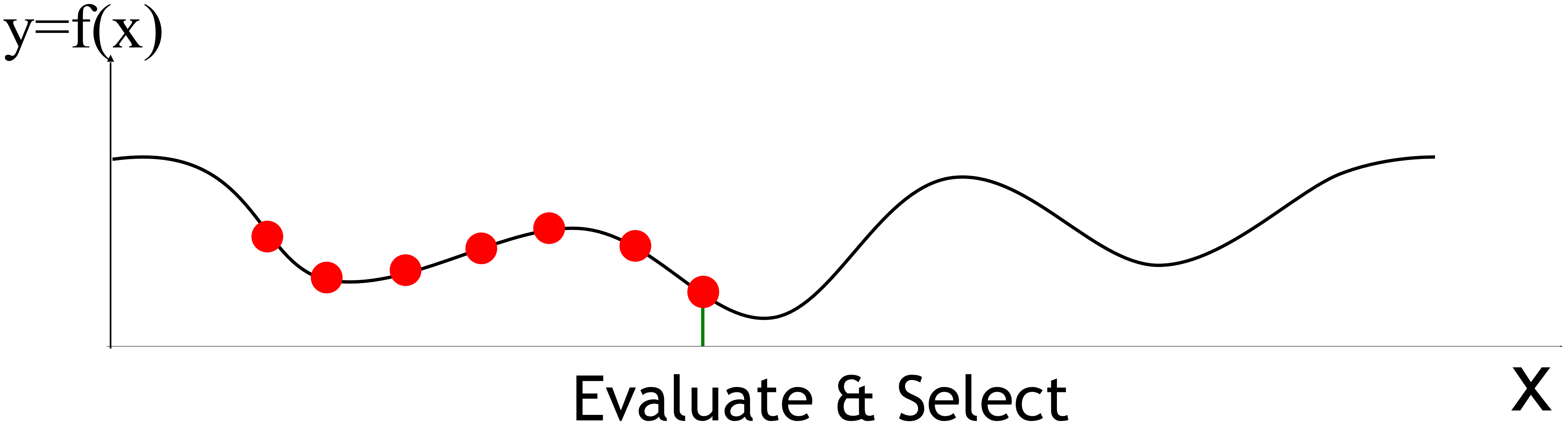
Evolutionary Methods

4. **Reproduce** offspring of the next generation (Recombination & mutation)



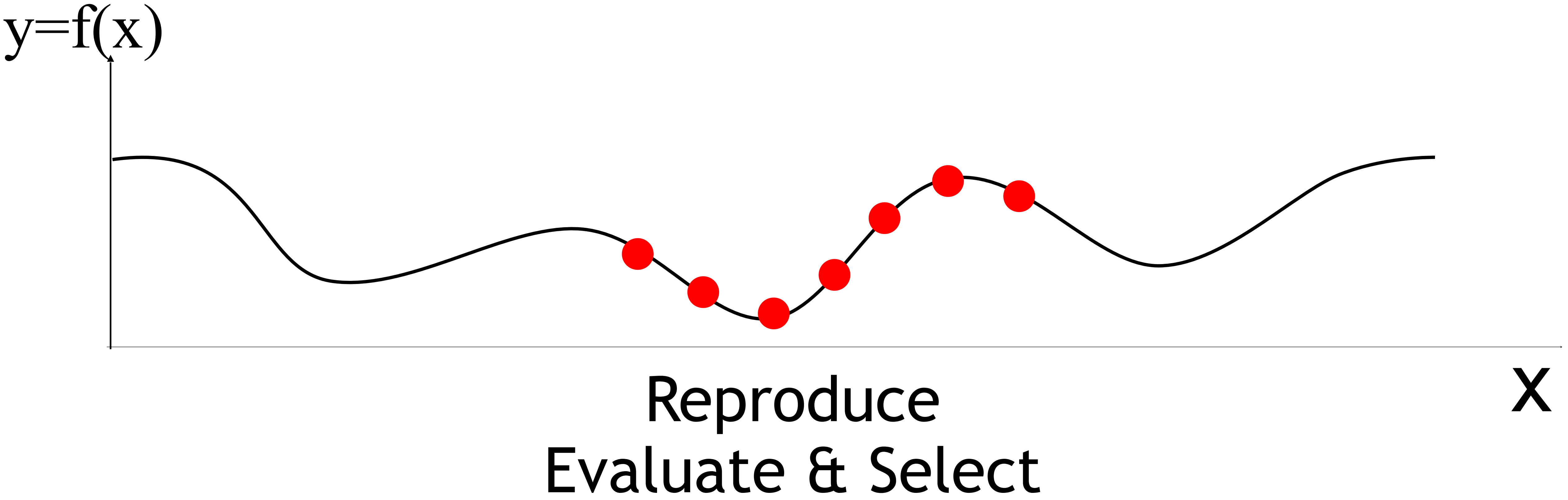
Evolutionary Methods

5. **Repeat** until a *termination criterion* is met



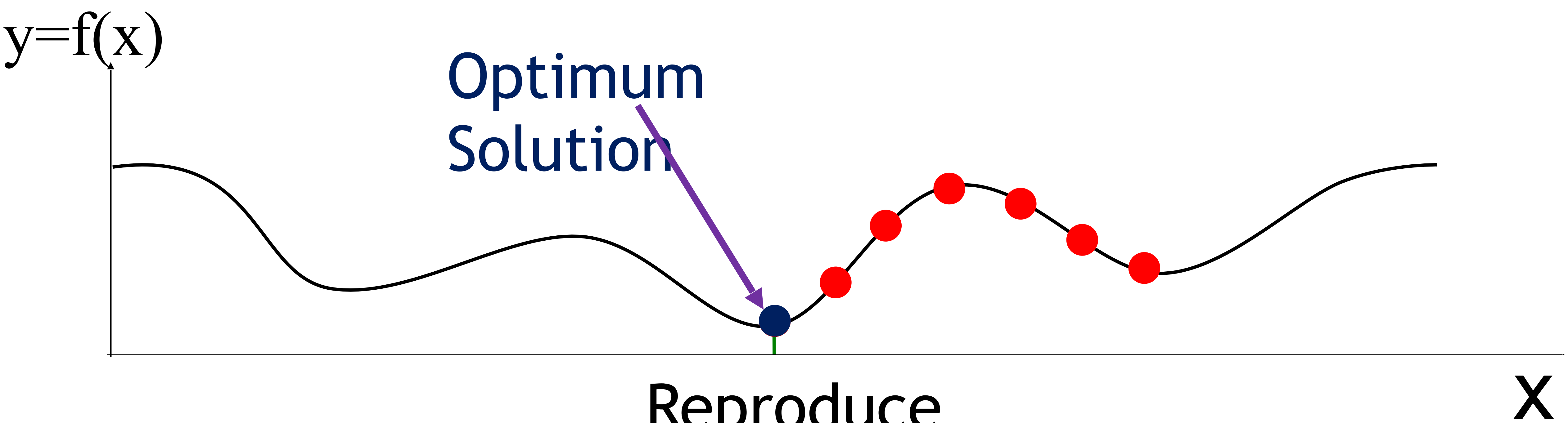
Evolutionary Methods

5. **Repeat** until a *termination criterion* is met



Evolutionary Methods

5. **Repeat** until a *termination criterion* is met



Reproduce

Evaluate – Select – Reproduce

Terminate

Cross-Entropy Method

$$\max_\theta U(\theta) = \max_\theta E[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

- Views U as a black box
- Ignores all other information other than U collected during episode

= evolutionary algorithm

Population: $P_{\mu^{(i)}}(\theta)$

CEM:

```
for iter i = 1, 2, ...
    for population member e = 1, 2, ...
        sample θ^(e) ~ P_{μ^(i)}(θ)
        execute roll-outs under π_{θ^(e)}
        store (θ^(e), U(e))
    endfor
    μ^(i+1) = arg max_μ Σ_ē log P_μ(θ^(ē))
        where ē indexes over top p %
endfor
```

for iter i = 1, 2, ...

for population member e = 1, 2, ...

sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$

execute roll-outs under $\pi_{\theta^{(e)}}$

store $(\theta^{(e)}, U(e))$

endfor

$\mu^{(i+1)} = \arg\max_\mu \sum_{\bar{e}} \log P_\mu(\theta^{(\bar{e})})$

where $\bar{e}$ indexes over top p %

endfor

slides from John Schulam and Pieter Abbeel

Cross-Entropy Method

- Can work embarrassingly well

Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon
INRIA Lille - Nord Europe,
Team SequeL, FRANCE
victor.gabillon@inria.fr

Mohammad Ghavamzadeh*
INRIA Lille - Team SequeL
& Adobe Research
mohammad.ghavamzadeh@inria.fr

Bruno Scherrer
INRIA Nancy - Grand Est,
Team Maia, FRANCE
bruno.scherrer@inria.fr

[NIPS 2013]

Closely Related Approaches

CEM:

for iter i = 1, 2, ...
  for population member e = 1, 2, ...
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
    execute roll-outs under $\pi_{\theta^{(e)}}$
    store $(\theta^{(e)}, U(e))$
  endfor
  $$\boxed{\mu^{(i+1)} = \arg\max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{(\bar{e})})}$$
  where $\bar{e}$ indexes over top p %
endfor

- Reward Weighted Regression (RWR)
  - Dayan & Hinton, NC 1997; Peters & Schaal, ICML 2007

$$\boxed{\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} q(U(e), P_{\mu}(\theta^{(e)})) \log P_{\mu}(\theta^{(e)})}$$

- Policy Improvement with Path Integrals (PI^2)
  - PI2: Theodorou, Buchli, Schaal JMLR2010; Kappen, 2007; (PI2-CMA: Stulp & Sigaud ICML2012)

$$\boxed{\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} \exp(\lambda U(e)) \log P_{\mu}(\theta^{(e)})}$$

- Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)
  - CMA: Hansen & Ostermeier 1996; (CMA-ES: Hansen, Muller, Koumoutsakos 2003)

$$\boxed{(\mu^{(i+1)}, \Sigma^{(i+1)}) = \arg\max_{\mu, \Sigma} \sum_{\bar{e}} w(U(\bar{e})) \log \mathcal{N}(\theta^{(\bar{e})}; \mu, \Sigma)}$$

- PoWER
  - Kober & Peters, NIPS 2007 (also applies importance sampling for sample re-use)

$$\boxed{\mu^{(i+1)} = \mu^{(i)} + \left(\sum_{e} (\theta^{(e)} - \mu^{(i)}) U(e)\right) / \left(\sum_{e} U(e)\right)}$$

Grounded Action Transformation for Robot Learning in Simulation

Josiah P. Hanna, Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{jphanna,pstone}@cs.utexas.edu

- Idea: bring simulation closer to real world by learning parametrized actions whose execution (in simulation) brings simulation state close to real world state.

Assumes:

- a modifiable simulator with a parametrized transition probabilities $P_{\text{sim}}(\cdot|s, a; \phi)$ where the vector ϕ can be changed to produce in effect a different simulator
- a policy learning procedure (`optimize`) in simulation
- we can evaluate the policy in the real world (physical robot)

Grounded Simulation Learning

- Let $d(p, q)$ be a measure of similarity between probabilities p and q. GSL grounds E_{sim} by finding $\phi^{\star}$ such that:

$$\phi^{\star} = \arg\min_{\phi} \sum_{\tau \in \mathcal{D}} d(Pr(\tau|\theta), Pr_{\text{sim}}(\tau|\theta, \phi)) \qquad (1)$$

Humanoid Robots Learning to Walk Faster: From the Real World to Simulation and Back, Farchy et al. 2013

Grounded Simulation Learning

- Let $d(p, q)$ be a measure of similarity between probabilities p and q. GSL grounds E_{sim} by finding $\phi^\star$ such that:

$$\phi^\star = \arg\min_{\phi} \sum_{\tau \in \mathcal{D}} d(Pr(\tau|\theta), Pr_{\text{sim}}(\tau|\theta, \phi)) \qquad (1)$$

$$\phi^\star = \arg\min_{\phi} \sum_{\tau_i \in \mathcal{D}} \sum_{t=0}^{L} d(P(s_{t+1}^i|s_t^i, a_t^i), P_\phi(s_{t+1}^i|s_t^i, a_t^i))$$

Humanoid Robots Learning to Walk Faster: From the Real World to Simulation and Back, Farchy et al. 2013

Grounded Simulation Learning

- Let $d(p, q)$ be a measure of similarity between probabilities p and q. GSL grounds E_{sim} by finding $\phi^\star$ such that:

$$\phi^\star = \arg\min_{\phi} \sum_{\tau \in \mathcal{D}} d(Pr(\tau|\theta), Pr_{\text{sim}}(\tau|\theta, \phi))$$

1. Execute policy θ_0 on the physical robot to collect a data set of trajectories , $\mathcal{D}$.

2. Use $\mathcal{D}$ to find $\phi^\star$ that satisfies Equation 1

3. Use *optimize* with J_{sim} and $P_{\phi^\star}$ to learn a set of candidate policies Π_c in simulation which are expected to perform well on the physical robot

4. Evaluate each proposed $\theta_c \in \Pi_c$ on the physical robot and return the policy, θ_1, with minimal J

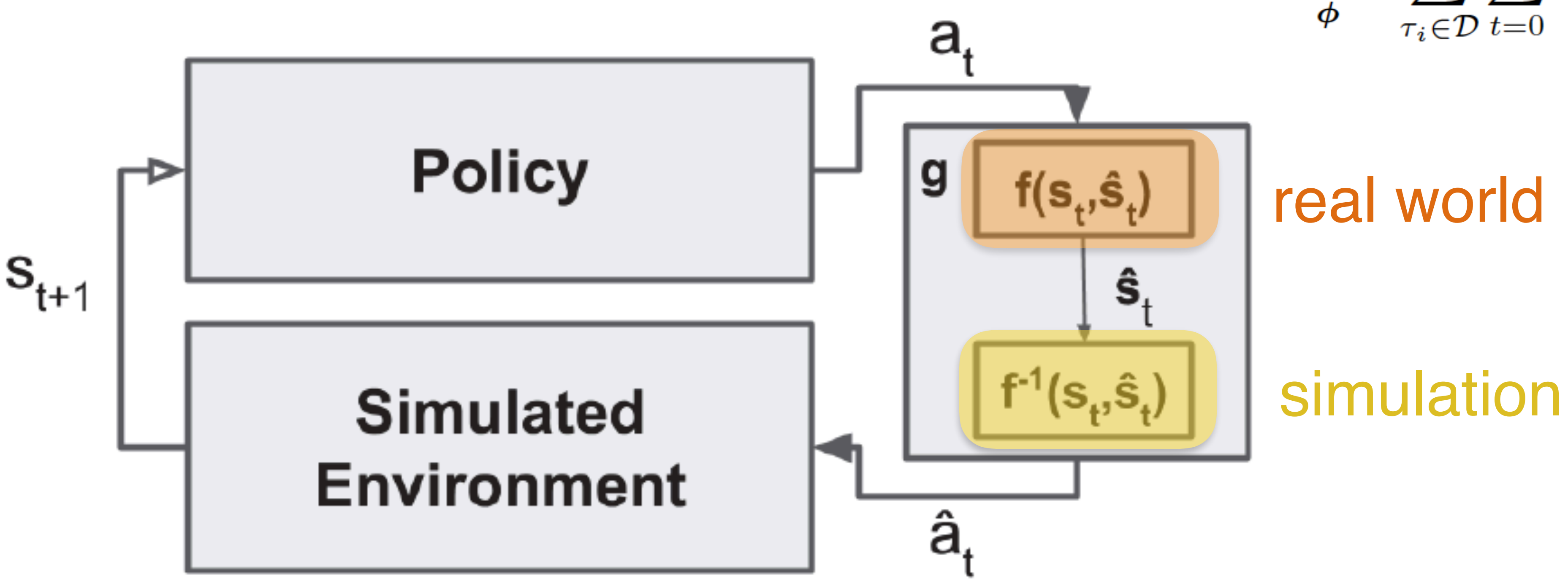
5. GOTO 1

Humanoid Robots Learning to Walk Faster: From the Real World to Simulation and Back, Farchy et al. 2013

Grounded Simulation Learning

ϕ instead of parametrizing physical parameters of the simulator, parametrizes actions transformations!

$$\phi^\star = \operatorname*{argmin}_{\phi} \sum_{\tau_i \in \mathcal{D}} \sum_{t=0}^{L} d(P(\mathbf{s}_{t+1}^i | \mathbf{s}_t^i, \mathbf{a}_t^i), P_{\boldsymbol{\phi}}(\mathbf{s}_{t+1}^i | \mathbf{s}_t^i, \mathbf{a}_t^i))$$



Reminder: Froward and backward models

- Forward model: maps state and action to next state. With forward models you can solve for action that leads to desired next state.

$$s_t, a_t \rightarrow s_{t+1}$$

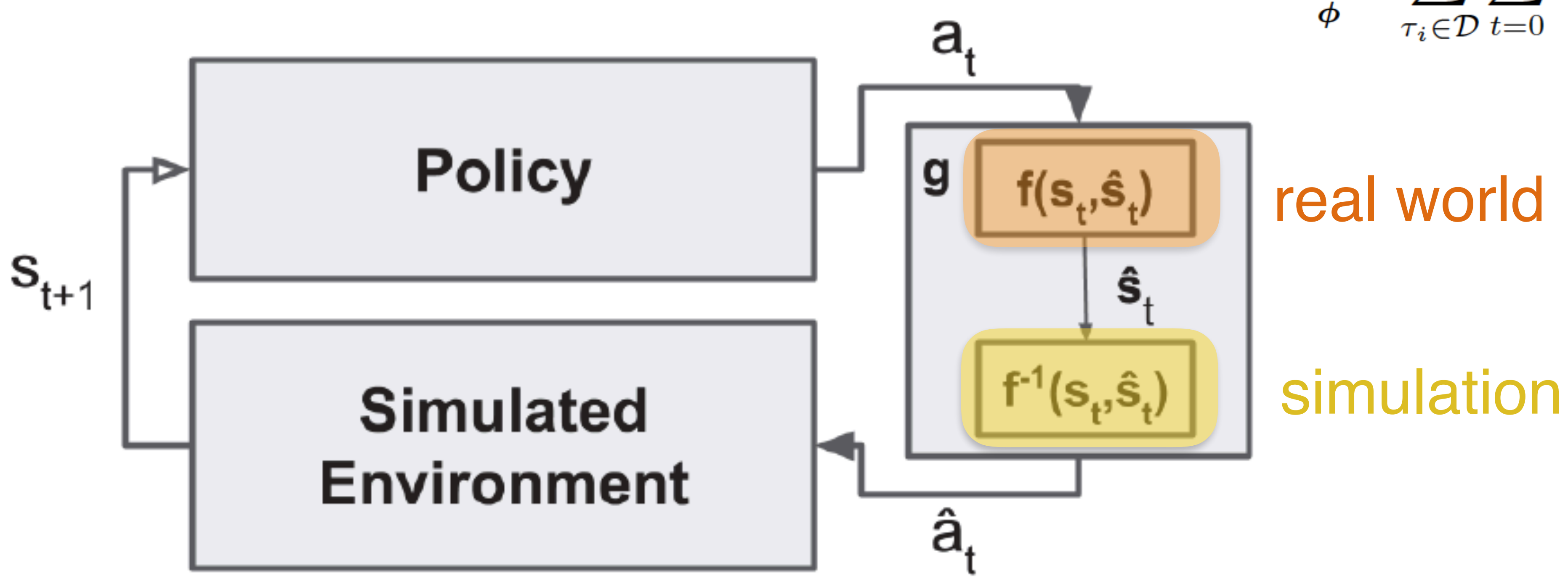
- Backward model: maps state and next state to the action that achieves the transition. Its output is used directly for control.

$$s_t, s_{t+1} \rightarrow a_t$$

Grounded Simulation Learning

ϕ instead of parametrizing physical parameters of the simulator, parametrizes actions transformations!

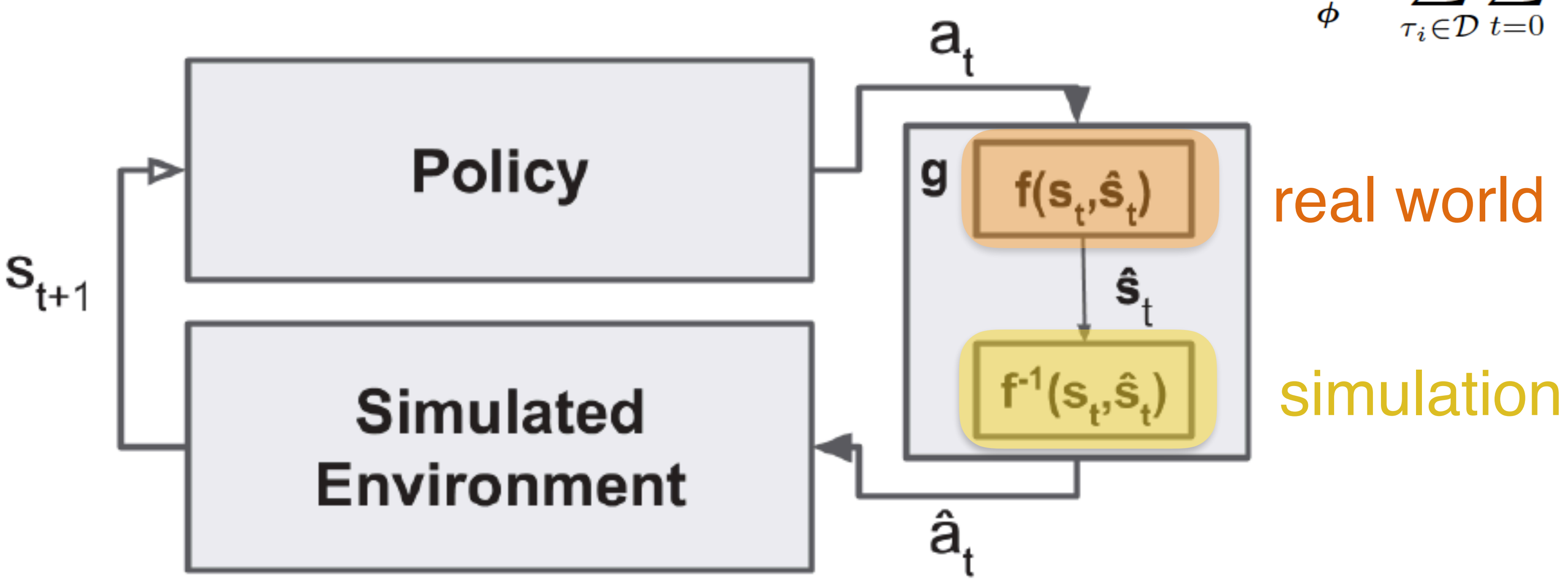
$$\phi^\star = \underset{\phi}{\operatorname{argmin}} \sum_{\tau_i \in \mathcal{D}} \sum_{t=0}^{L} d(P(\mathbf{s}^i_{t+1}|\mathbf{s}^i_t, \mathbf{a}^i_t), P_{\phi}(\mathbf{s}^i_{t+1}|\mathbf{s}^i_t, \mathbf{a}^i_t))$$



Grounded Simulation Learning

ϕ instead of parametrizing physical parameters of the simulator, parametrizes actions transformations!

$$\phi^\star = \underset{\phi}{\operatorname{argmin}} \sum_{\tau_i \in \mathcal{D}} \sum_{t=0}^{L} d(P(\mathbf{s}_{t+1}^i | \mathbf{s}_t^i, \mathbf{a}_t^i), P_\phi(\mathbf{s}_{t+1}^i | \mathbf{s}_t^i, \mathbf{a}_t^i))$$



- A deterministic forward model of the real robot's dynamics, f predicts the effect of executing a_t on the physical robot.
- An inverse dynamics model of the simulated robot uses the prediction , $\hat{s}$, to predict the action $\hat{a}_t$ which will achieve $\hat{x}_t$ in simulation.
- $\hat{a}_t$ is executed in simulation. The resulting state transition will be similar to the transition that would have occurred in the real world.

Algorithm 1 Grounded Action Transformation (GAT) Pseudo code. Input: An initial policy, $\boldsymbol{\theta}$, the environment, E, a simulator, $E_{\texttt{sim}}$, smoothing parameter α, and a policy improvement method, `optimize`. The function `rolloutN(`$\boldsymbol{\theta}$`, N)` executes N trajectories with $\boldsymbol{\theta}$ and returns the observed state transition data. The functions `trainForwardModel` and `trainInverseModel` estimate models of the forward and inverse dynamics respectively.

function GAT
  $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}$
  $\mathcal{D}_{\texttt{robot}} \leftarrow \texttt{RolloutN}(E, \boldsymbol{\theta}_0, N)$
  $\mathcal{D}_{\texttt{sim}} \leftarrow \texttt{RolloutN}(E_{\texttt{sim}}, \boldsymbol{\theta}_0, N)$
  $f \leftarrow \texttt{trainForwardModel}(\mathcal{D}_{\texttt{robot}})$
  $f^{-1}_{\texttt{sim}} \leftarrow \texttt{trainInverseModel}(\mathcal{D}_{\texttt{sim}})$
  $g(s,a) \leftarrow \alpha f^{-1}_{\texttt{sim}}(s, f(s,a)) + (1-\alpha) \cdot a_t$
  $\Pi \leftarrow \texttt{optimize}(E_{\texttt{sim}}, \boldsymbol{\theta}, g)$
  return $\operatorname{argmin}_{\boldsymbol{\theta} \in \Pi} J(\boldsymbol{\theta})$
end function

changed the simulator dynamics

Simulation Grounding Results

Video

Method	% Improve	Failures	Best Gen.
No Ground	11.094	7	1.33
Noise-Envelope	18.93	5	6.6
GAT	**22.48**	**1**	2.67

Method	Velocity (cm/s)	% Improve
$\boldsymbol{\theta}_0$	19.52	0.0
GAT SimSpark $\boldsymbol{\theta}_1$	26.27	34.58
GAT SimSpark $\boldsymbol{\theta}_2$	27.97	43.27
GAT Gazebo $\boldsymbol{\theta}_1$	26.89	37.76

Noise envelope baseline: Add noise to the simulation dynamics to encourage policy learning to find policies robust across environments.

Policies that work under a range of possible models, can be conservative and work worse for that particular world

This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - synthetic data randomization
  - feature fine-tuning
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

EPOpt: Learning Robust Neural Network Policies Using Model Ensembles

Aravind Rajeswaran[1], **Sarvjeet Ghotra**[2], **Balaraman Ravindran**[3], **Sergey Levine**[4]
aravraj@cs.washington.edu, sarvjeet.13it236@nitk.edu.in,
ravi@cse.iitm.ac.in, svlevine@eecs.berkeley.edu
[1] University of Washington Seattle
[2] NITK Surathkal
[3] Indian Institute of Technology Madras
[4] University of California Berkeley

Ideas:

- Consider a distribution over simulation models instead of a single one for learning policies robust to modeling errors that work well under many ``worlds”.
- Progressively bring the simulation model distribution closer to the real world. Bayesian modeling of the dynamics.
- Hard model mining

Source domain distribution over MDPs

- MDPs differ in source and target domains w.r.t.
  - dynamics
  - rewards
  - initial state distributions
- and are identical w.r.t.
  - States
  - Actions

Policy Search under model distribution

Learn a policy that performs best in expectation over MDPs in the source domain distribution:

$$\mathbb{E}_{p\sim\mathcal{P}}\left[\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{T-1}\gamma^t r_t(s_t,a_t)\,\middle|\, p\right]\right]$$

p: simulator parameters

Policy Search under model distribution

Learn a policy that performs best in expectation over MDPs in the source domain distribution:

$$\mathbb{E}_{p\sim\mathcal{P}}\left[\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{T-1}\gamma^t r_t(s_t,a_t)\,\middle|\,p\right]\right]$$

p: simulator parameters

Hard model mining

Learn a policy that performs best in expectation over the worst \epsilon-percentile of MDPs in the source domain distribution

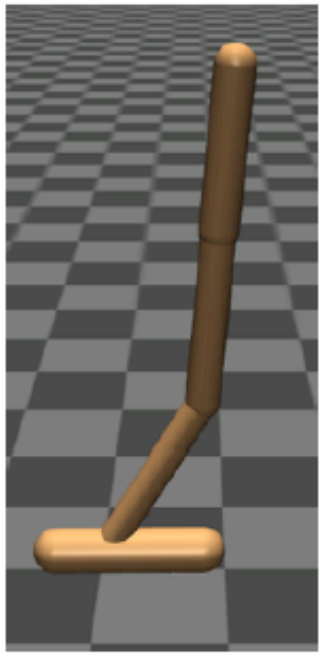
$$\max_{\theta,y}\int_{\mathcal{F}(\theta)}\eta_{\mathcal{M}}(\theta,p)\mathcal{P}(p)dp \qquad s.t.\quad \mathbb{P}\left(\eta_{\mathcal{M}}(\theta,P)\leq y\right)=\epsilon$$

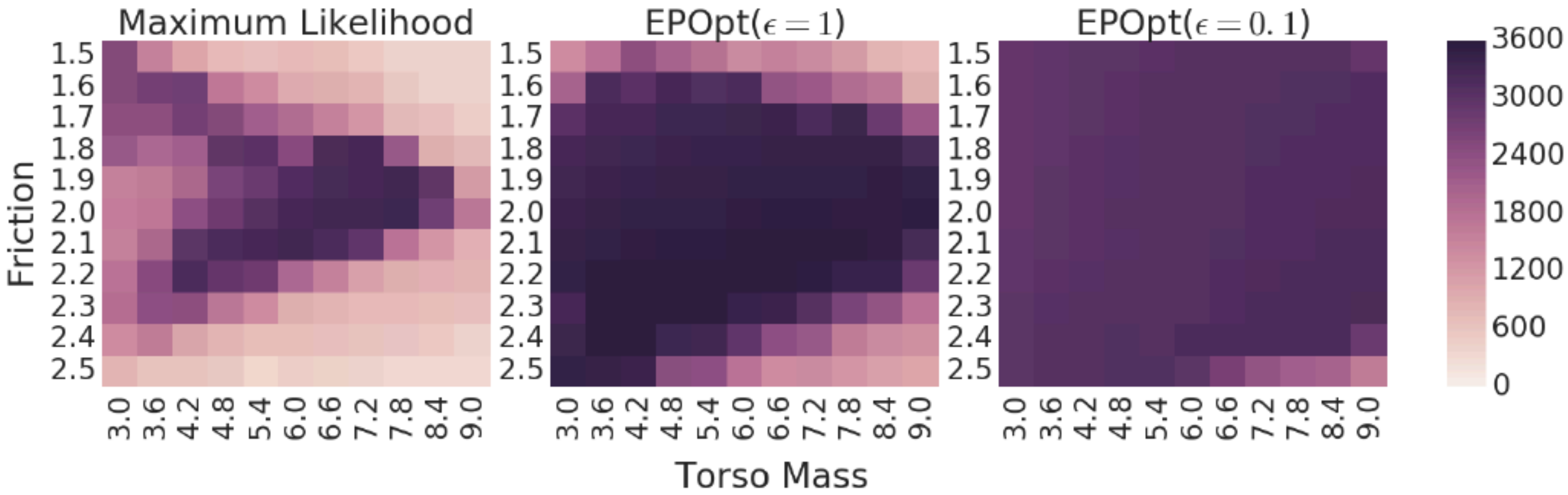
Hard model mining

Algorithm 1: EPOpt–ϵ for Robust Policy Search

Input: ψ, θ_0, $niter$, N, ϵ
for *iteration* $i = 0, 1, 2, \dots niter$ **do**
  for $k = 1, 2, \dots N$ **do**
    sample model parameters $p_k \sim \mathcal{P}_\psi$
    sample a trajectory $\tau_k = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$ from $\mathcal{M}(p_k)$ using policy $\pi(\theta_i)$
  end
  compute $Q_\epsilon = \epsilon$ percentile of $\{R(\tau_k)\}_{k=1}^{N}$
  select sub-set $\mathbb{T} = \{\tau_k : R(\tau_k) \leq Q_\epsilon\}$
  Update policy: $\theta_{i+1} = \text{BatchPolOpt}(\theta_i, \mathbb{T})$
end

Hard model mining results





Adapting the source domain distribution

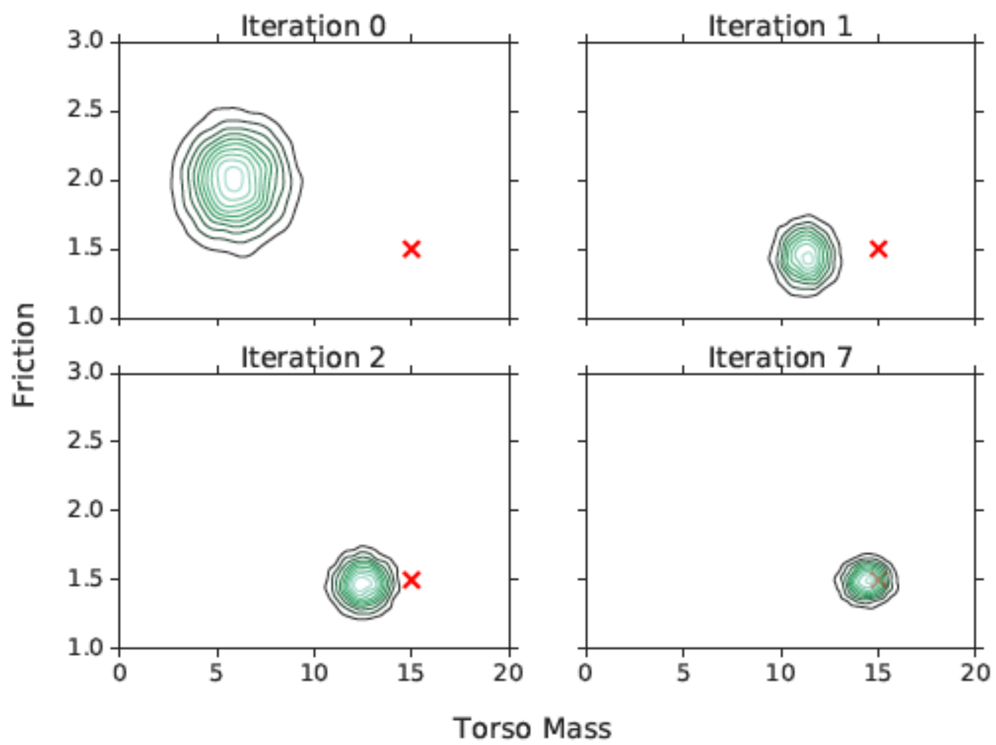
Sample a set of simulation parameters from a sampling distribution S.

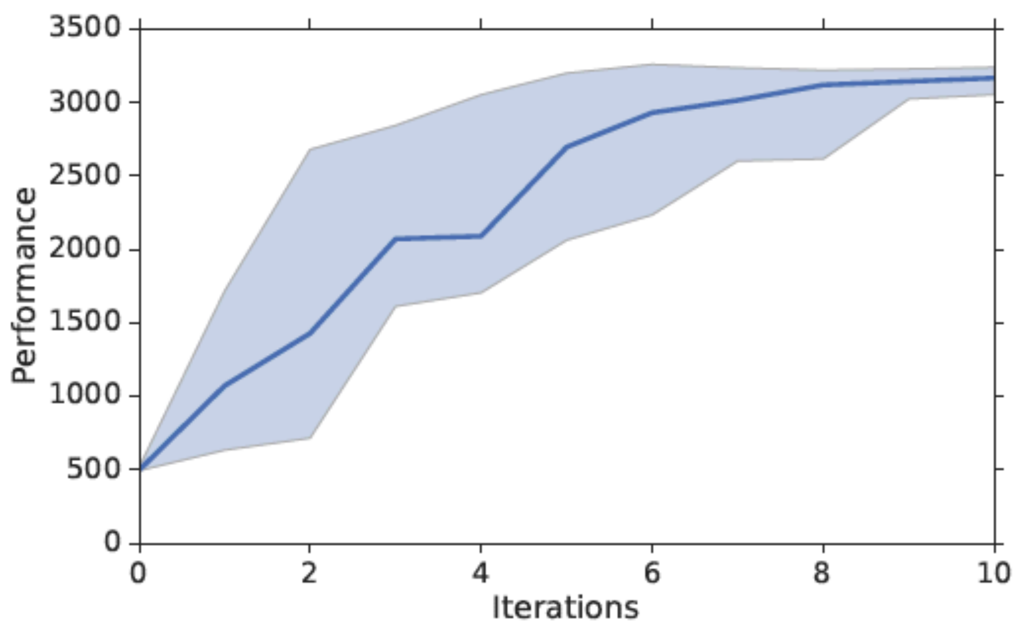
Posterior of parameters p_i:

$$\mathbb{P}(p_i|\tau_k) \propto \prod_t \mathbb{P}(S_{t+1} = s_{t+1}^{(k)}|s_t^{(k)}, a_t^{(k)}, p_i) \times \frac{\mathbb{P}_P(p_i)}{\mathbb{P}_S(p_i)}$$

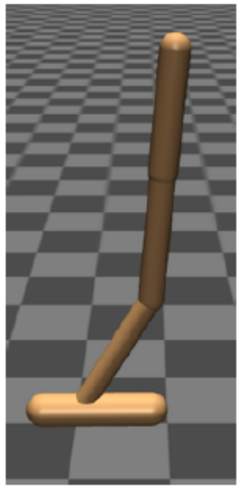
Fit a Gaussian model over simulator parameters based on posterior weights of the samples

Source Distribution Adaptation



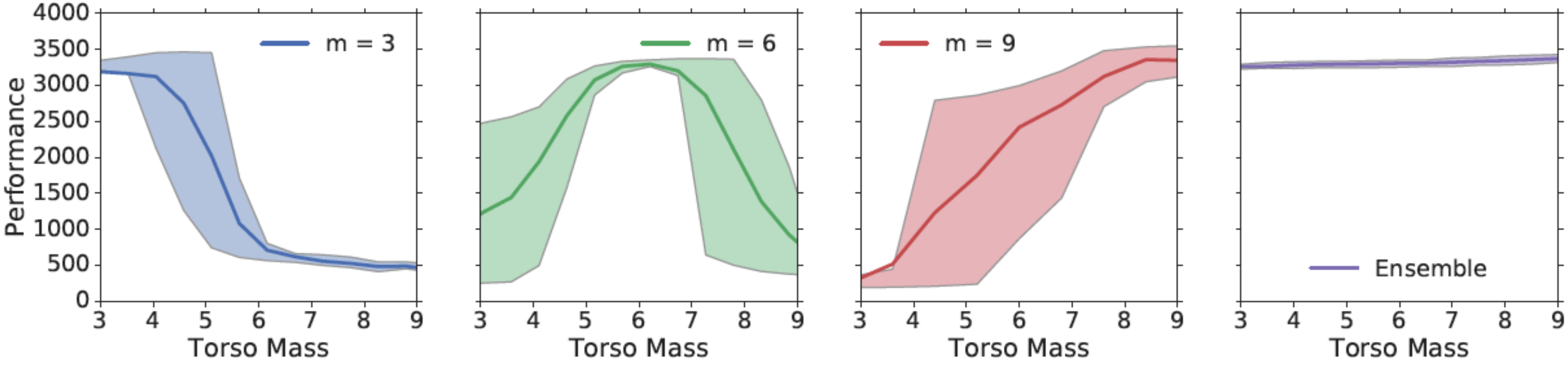


Performance on hopper policies



trained on single source domains

trained on Gaussian distribution of mean mass 6 and standard deviation 1.5



This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - synthetic data randomization
  - feature fine-tuning
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

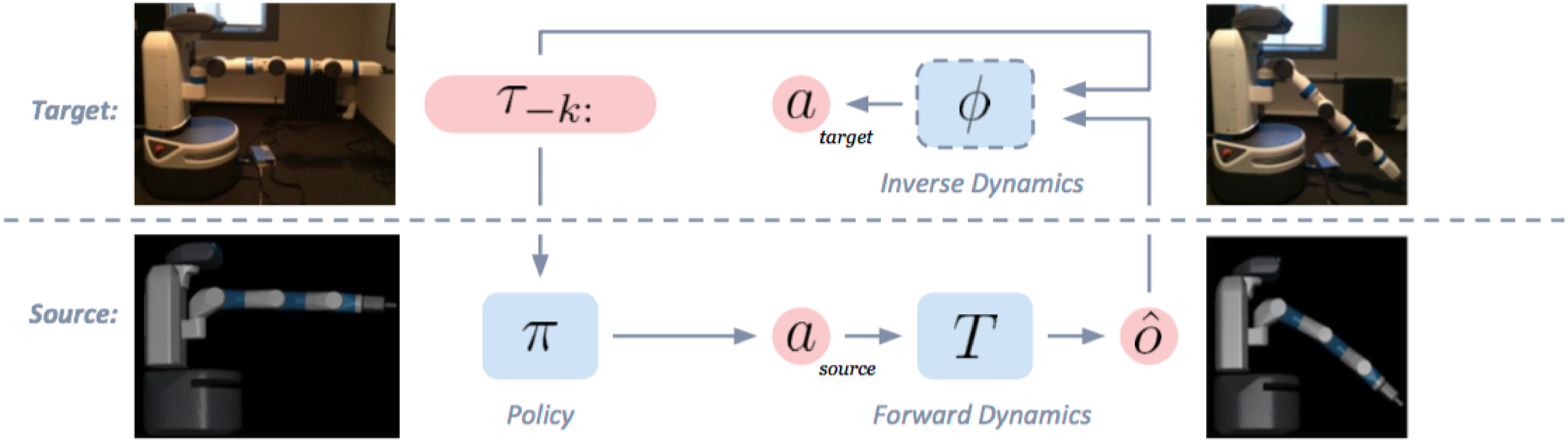
Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model

Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider,
Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba
OpenAI, San Francisco, CA, USA

- Policies search in simulation
- Use neural network to map learned policy in source environment (simulation) to target environment (real world)
- Transfer good policies in one simulation to many other real world environments, where a different inverse model takes care of the transfer to a particular target environment
- Observation in source and target environment are assumed the same, which is not always true

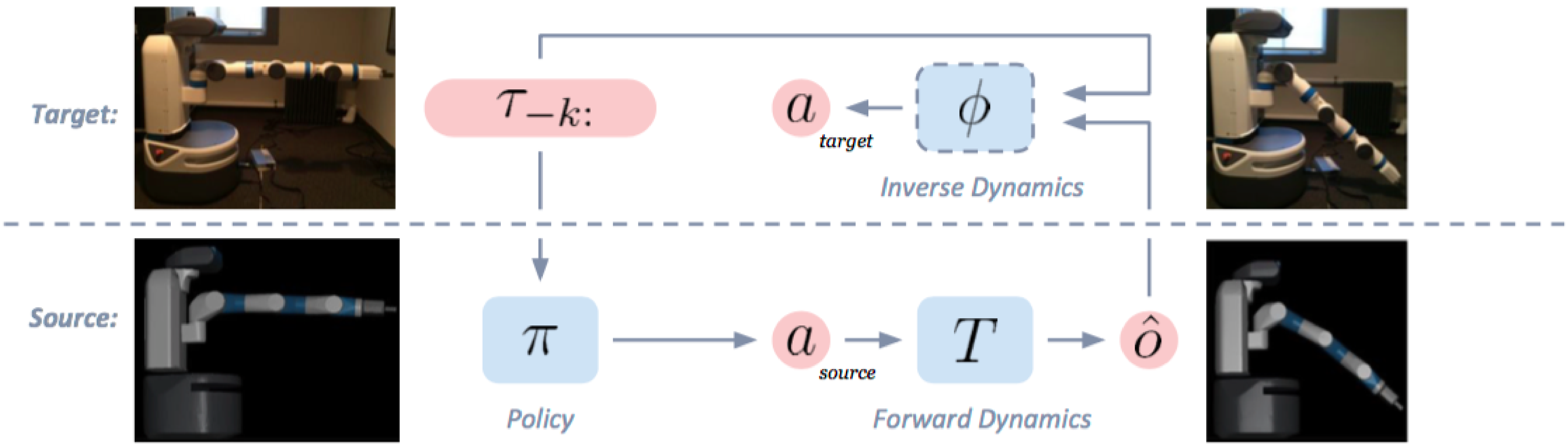
Deep Inverse Dynamic Model

- $\tau_{-k:}$: Trajectory: $\{o\}$ most recent k observations and $k-1$ actions of target environment
- π_{source} : Good enough policy in source environment
- ϕ : Inverse dynamics is a neural network that maps source policy to target policy



Deep Inverse Dynamic Model

1. Compute source action $a_{\text{source}} = \pi_{\text{source}}(\tau_{-k:})$ according to target trajectory
2. Observe the next state given $\tau_{-k:}$ and a_{source}:
3. Feed $\hat{o}_{\text{next}}$ and $\tau_{-k:}$ to inverse dynamics that produce a_{target}

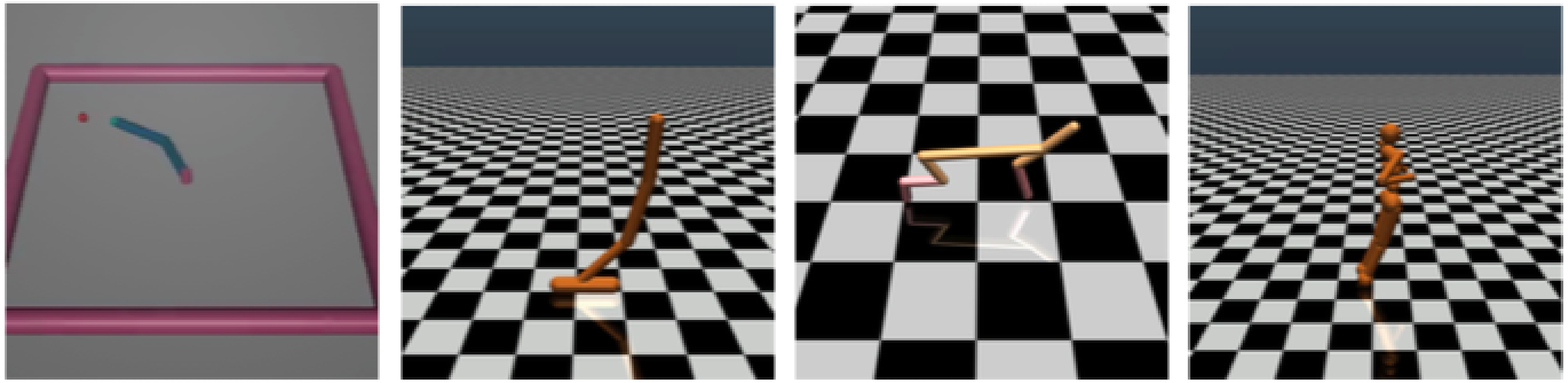


Architecture of Inverse Dynamic Neural Network

- Input: previous k observations, previous k-1 actions, desired observation for next time
- Output: the action that leads to desired observation
- Hidden layer: two fully connected hidden layer with 256 unit followed by ReLU activation function

Simulation 1 to Simulation 2 Transfer I

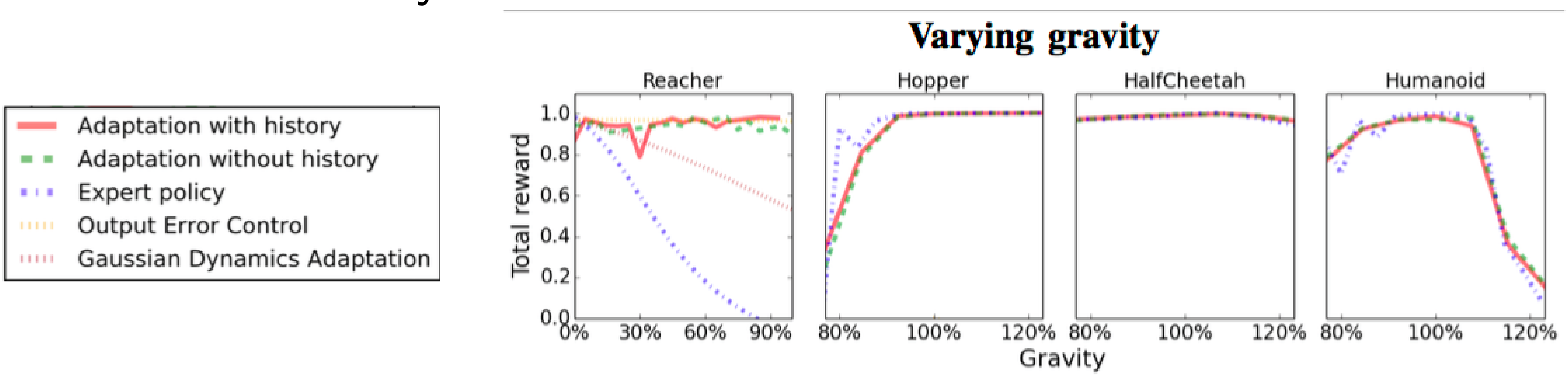
- The experiments are performed on Simulators that can change conditions of it's environment
- The source and target environment are basically the same model except gravity or motor noise
- The following four models are used for simulation



- Figure: From left to right are Reacher, Hopper, Half-cheetah, and Humanoid

Simulation 1 to Simulation 2 Transfer II

Variation of Gravity



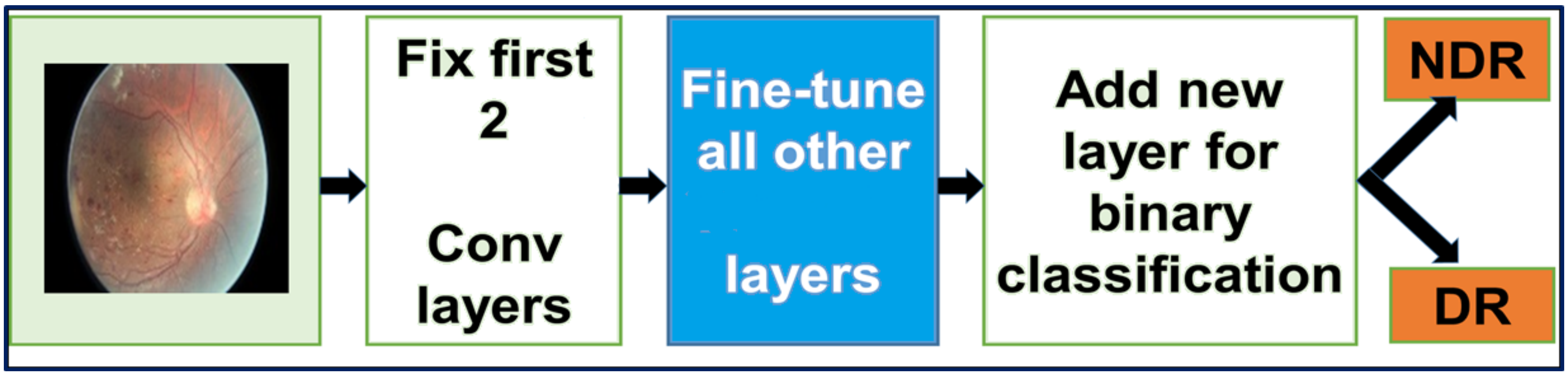
This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - feature fine-tuning
  - feature unsupervised pretraining
  - synthetic data randomization
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

Finetuning Deep (Visual) Features

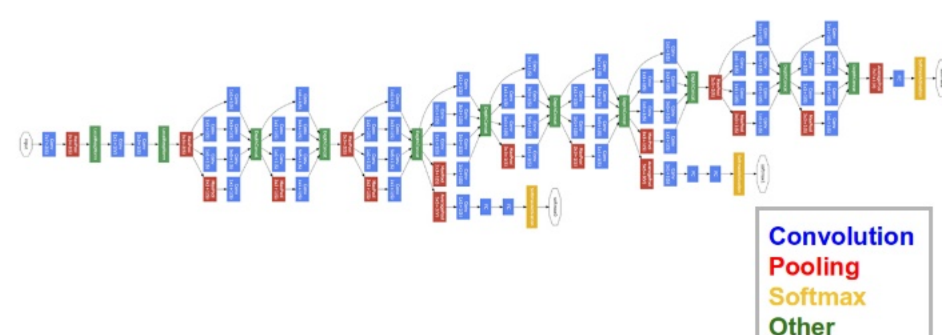
Common practice: we download a pretrained model and adapt it to our task

Finetuning GoogleNet for diabetic retinopathy prediction



Imagenet

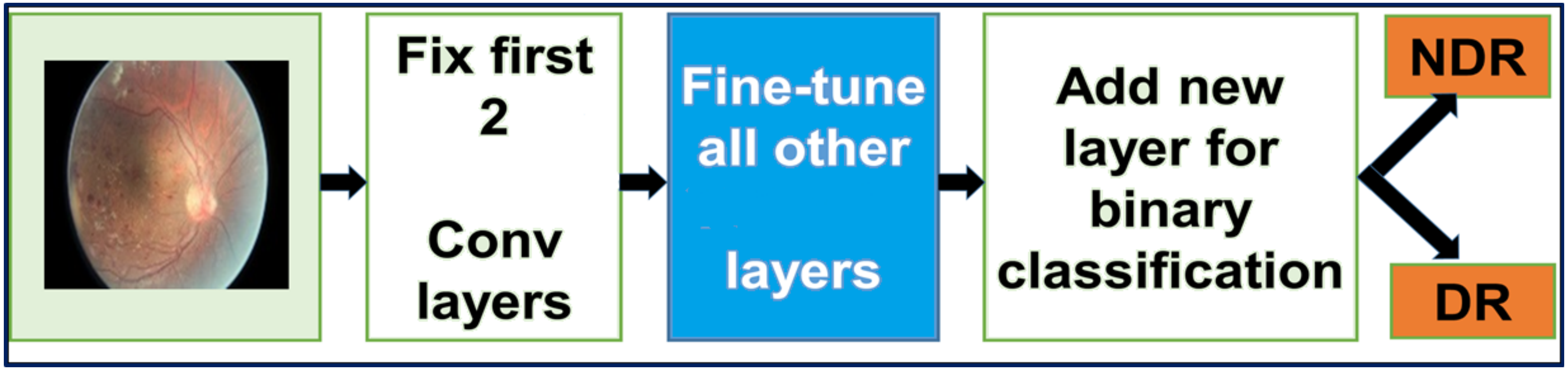
GoogleNet



Finetuning Deep (Visual) Features

Common practice: we download a pretrained model and adapt it to our task

Finetuning GoogleNet for diabetic retinopathy prediction

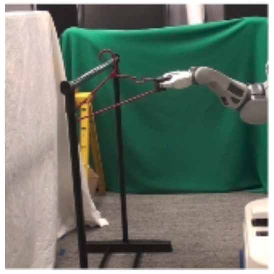


Pros: Straightforward to do

Cons: not great transfer due to difference in image statistics/ dataset biases

Imagenet

Policy Learning



Finetuning Deep (Visual) Features

- How many layers to finetune? For how long to fine-tune (how many iterations)?
- Catastrophic forgetting
- -> Learning to fine-tune (later lecture)

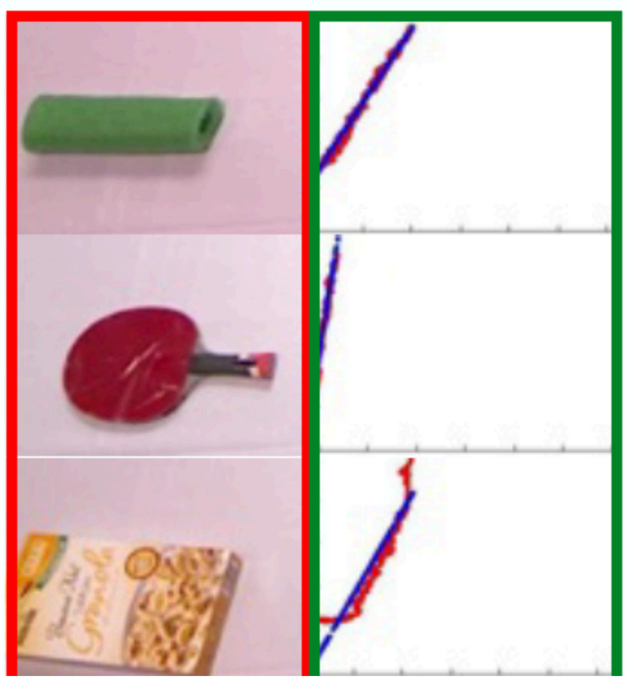
This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - feature fine-tuning
  - feature unsupervised pretraining
  - synthetic data randomization
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

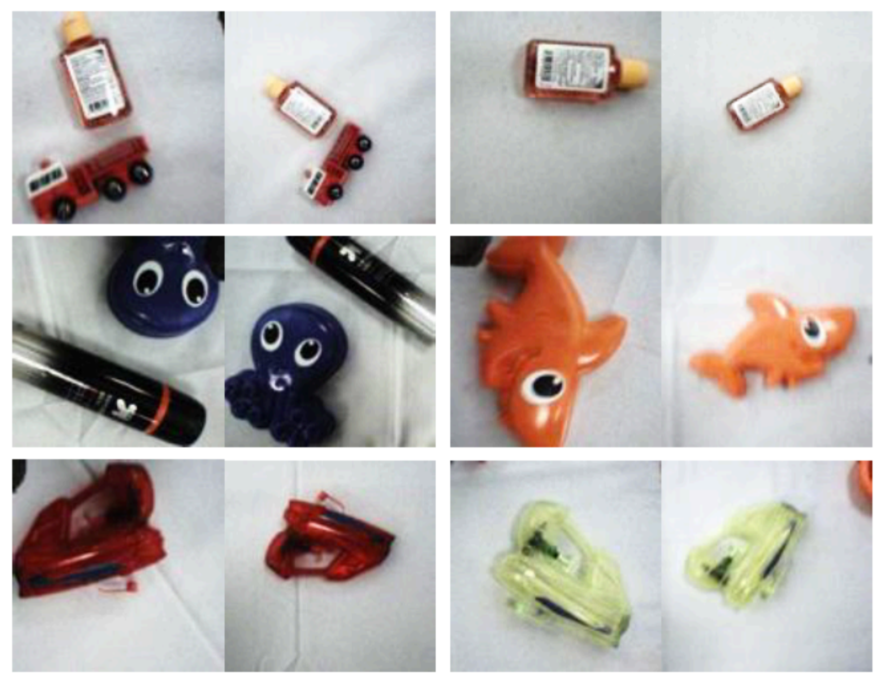
Pretrain Deep (Visual) Features using self-supervision

Less Common practice: use an unsupervised pretaining task to pretrain (or finetune) visual features, e.g., using feature slowness or using inverse dynamics models

predict the tactile profile



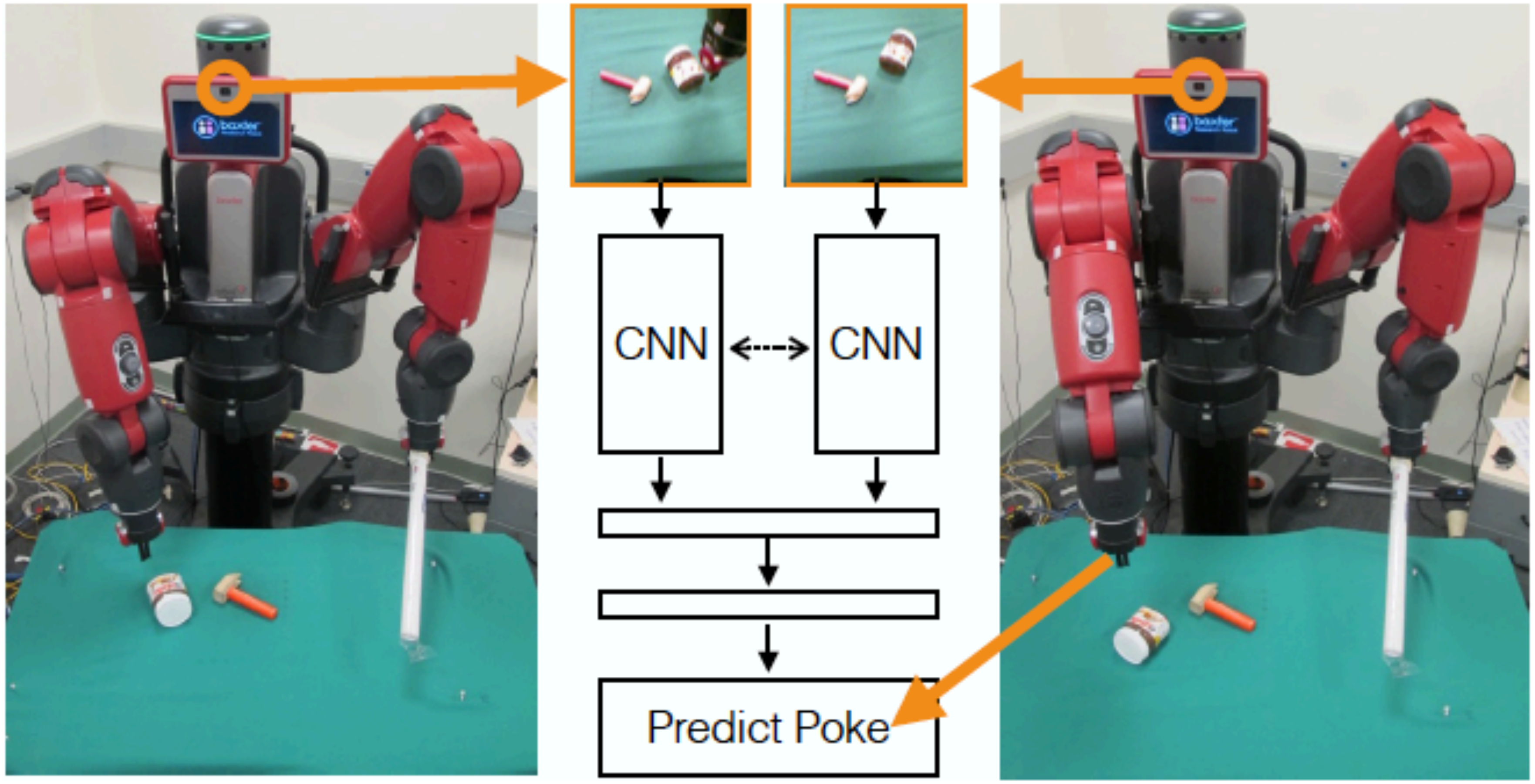
features to be invariant to viewpoint changes, metric learning



predict the push direction

The Curious Robot: Learning Visual Representations via Physical Interactions, Pinto et al.

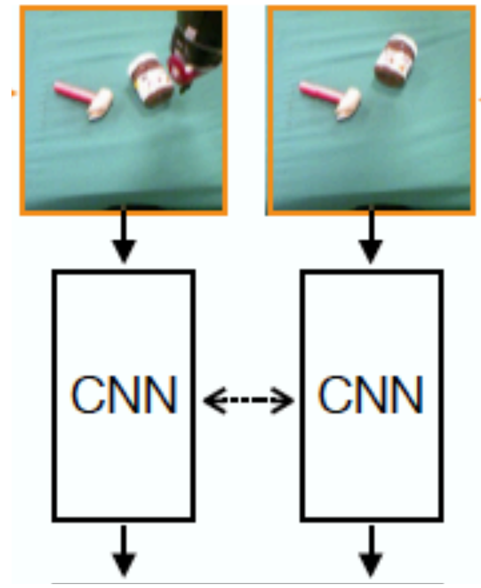
Pretrain Deep (Visual) Features using inverse models



Learning to Poke by Poking: Experiential Learning of Intuitive Physics, Agrawal et al.

The Curious Robot: Learning Visual Representations via Physical Interactions, Pinto et al.

Pretrain Deep (Visual) Features using slowness



feat_i feat_j

min. ||feat_i-feat_j||

The Curious Robot: Learning Visual Representations via Physical Interactions, Pinto et al.
Learning to Poke by Poking: Experiential Learning of Intuitive Physics, Agrawal et al.

This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - feature fine-tuning
  - feature unsupervised pretraining
  - synthetic data randomization
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

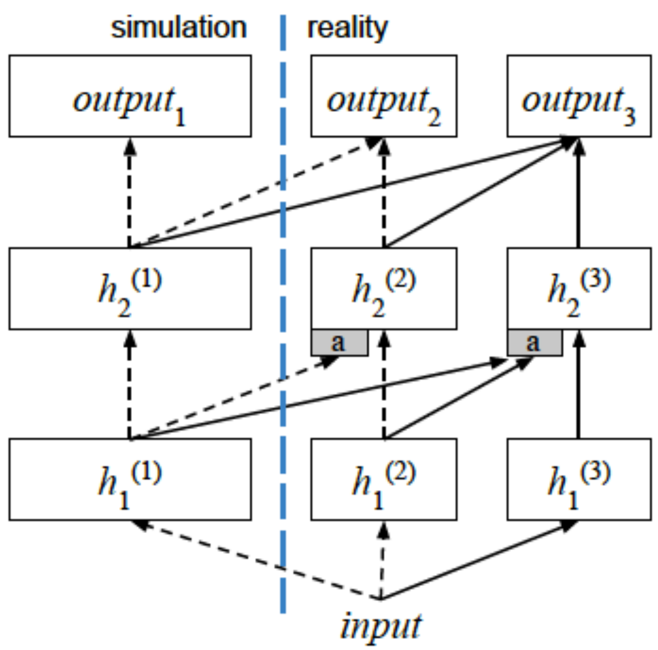
Sim-to-Real Robot Learning from Pixels with Progressive Nets

Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess,
Razvan Pascanu, Raia Hadsell

Google DeepMind
London, UK

- Add new capacity as you see new domains

Progressive Nets (grow a brain)



$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right)$$

$$f(x) = \max(0, x)$$

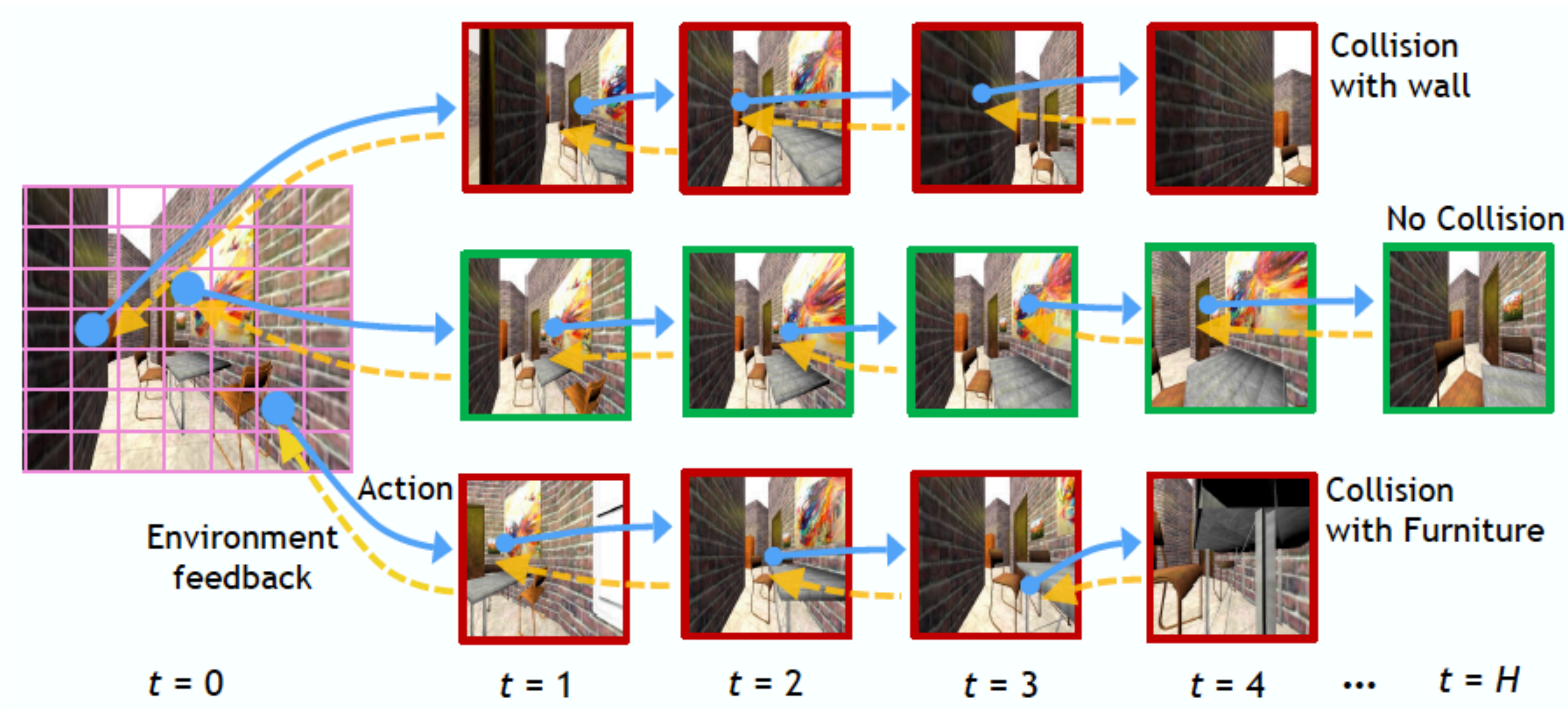
- Freeze previously learnt parameters when new ones are added, else the initia bad gradients will destroy them.
- Each column trains a different policy
- Columns in progressive networks are free to reuse, modify or

ignore previously learned features via the lateral connections.

- Columns of real robot have much smaller capacity that columns trained on simulation

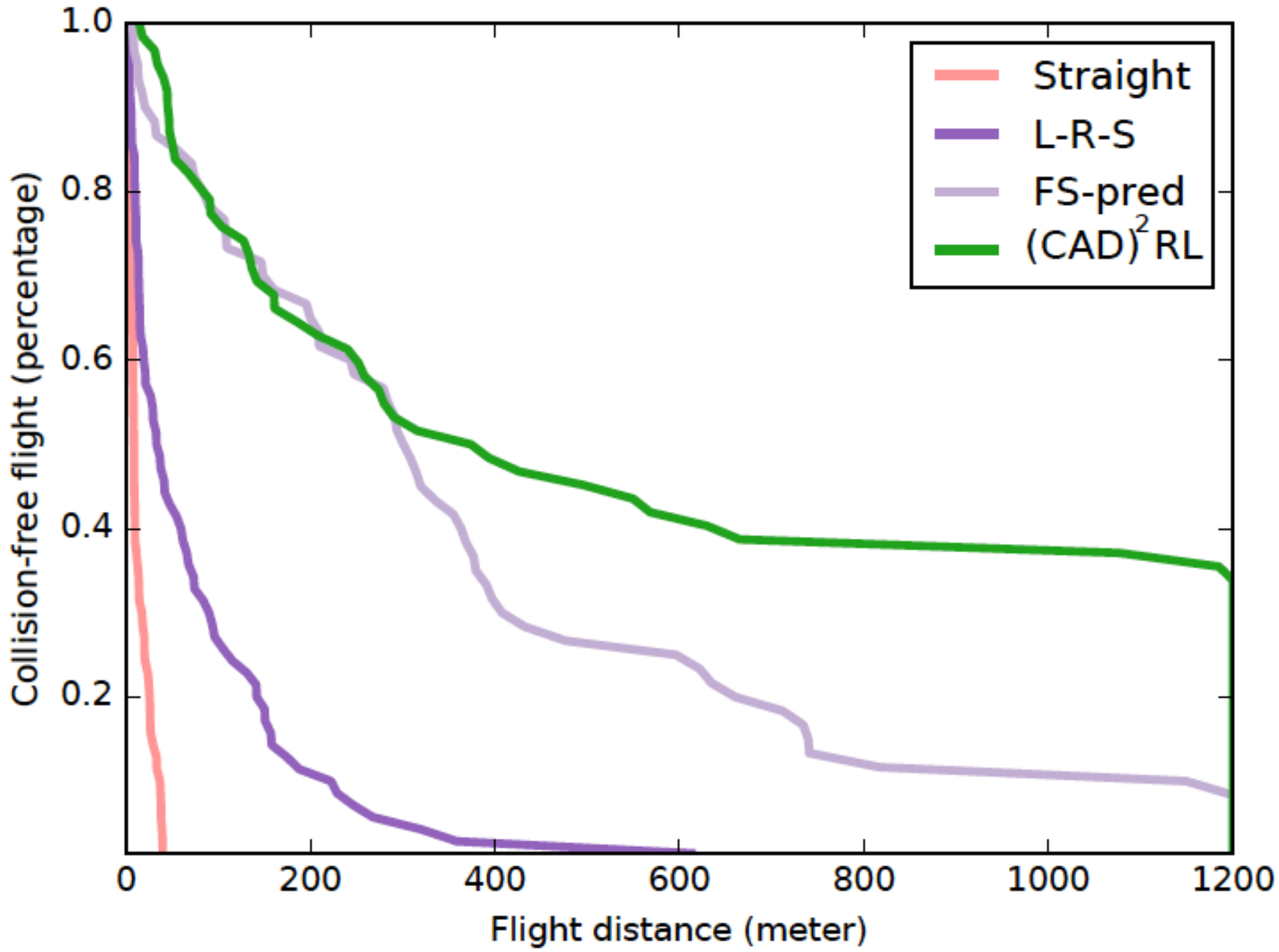
$(CAD)^2$RL: Real Single-Image Flight without a Single Real Image

Fereshteh Sadeghi[1] and Sergey Levine[2]



- reset at any state!
- Exhaustive policy rollouts: all actions are tried out for a horizon of 5 steps, to get the probability of collision in each state and action

Learning control VS learning a forward model



LRS (left-right-straight) baseline: given an image predict direction of motion. Not fine enough around corners, also, you cannot choose your direction of motion.

Instead, here we predict probability of collision given an action, which can be combined with a diverse set of goals, e.g., track the human.

$(CAD)^2RL$:

Real Single-Image Flight without a Single Real Image

Fereshteh Sadeghi

University of Washington

Sergey Levine

University of California, Berkeley

This lecture: Sim2real

- Transfer across different dynamics
  - online dynamics adaptation
  - have a neural network to adapt the policy learnt in simulation to the real world
  - grounding simulators: learning to bring their dynamics closer to real world dynamics using:
    - Parametrized action transformations
    - ensemble of simulators and adapting their distribution to match dynamics in the real world
- Transfer across different observations
  - feature fine-tuning
  - feature unsupervised pretraining
  - synthetic data randomization
  - feature progression
  - Supervised paired alignment between observation in simulation and real world
  - Unsupervised observation distribution matching

Domain Adaptation

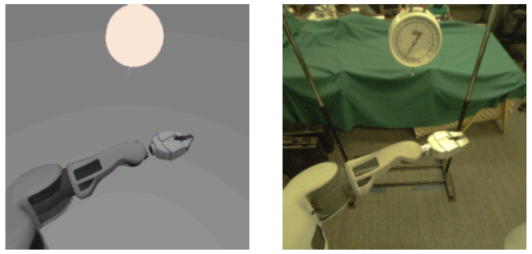
Adapting Deep Visuomotor Representations with Weak Pairwise Constraints

Eric Tzeng*[1], Coline Devin*[1], Judy Hoffman[1], Chelsea Finn[1],
Pieter Abbeel[1], Sergey Levine[1], Kate Saenko[2], Trevor Darrell[1]

[1] University of California, Berkeley
[2] Boston University

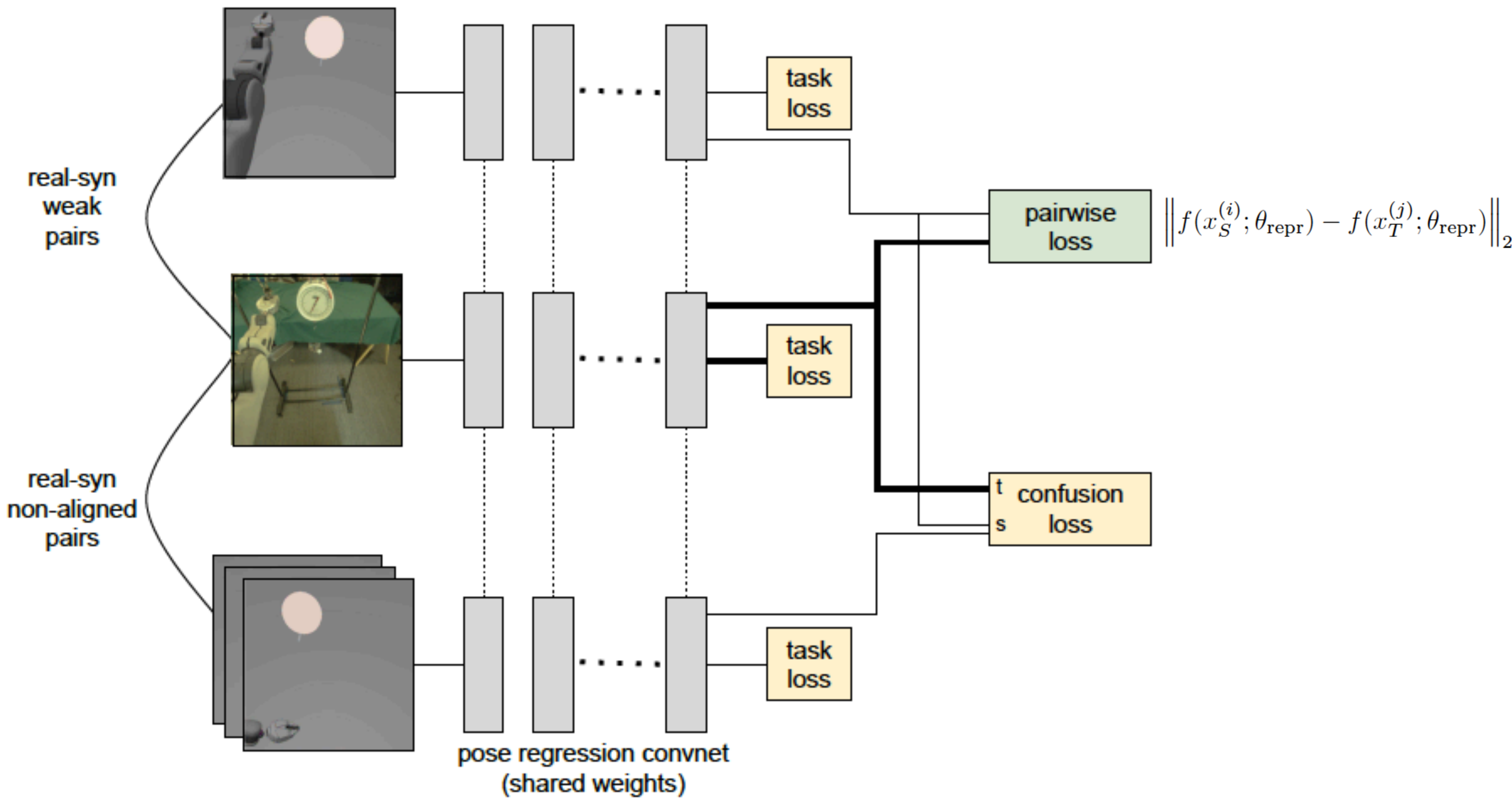
Use both domain confusion (distribution matching) between source and target domains, as well as image pairs, for regularizing the learnt visual feature representation

source target image pairs



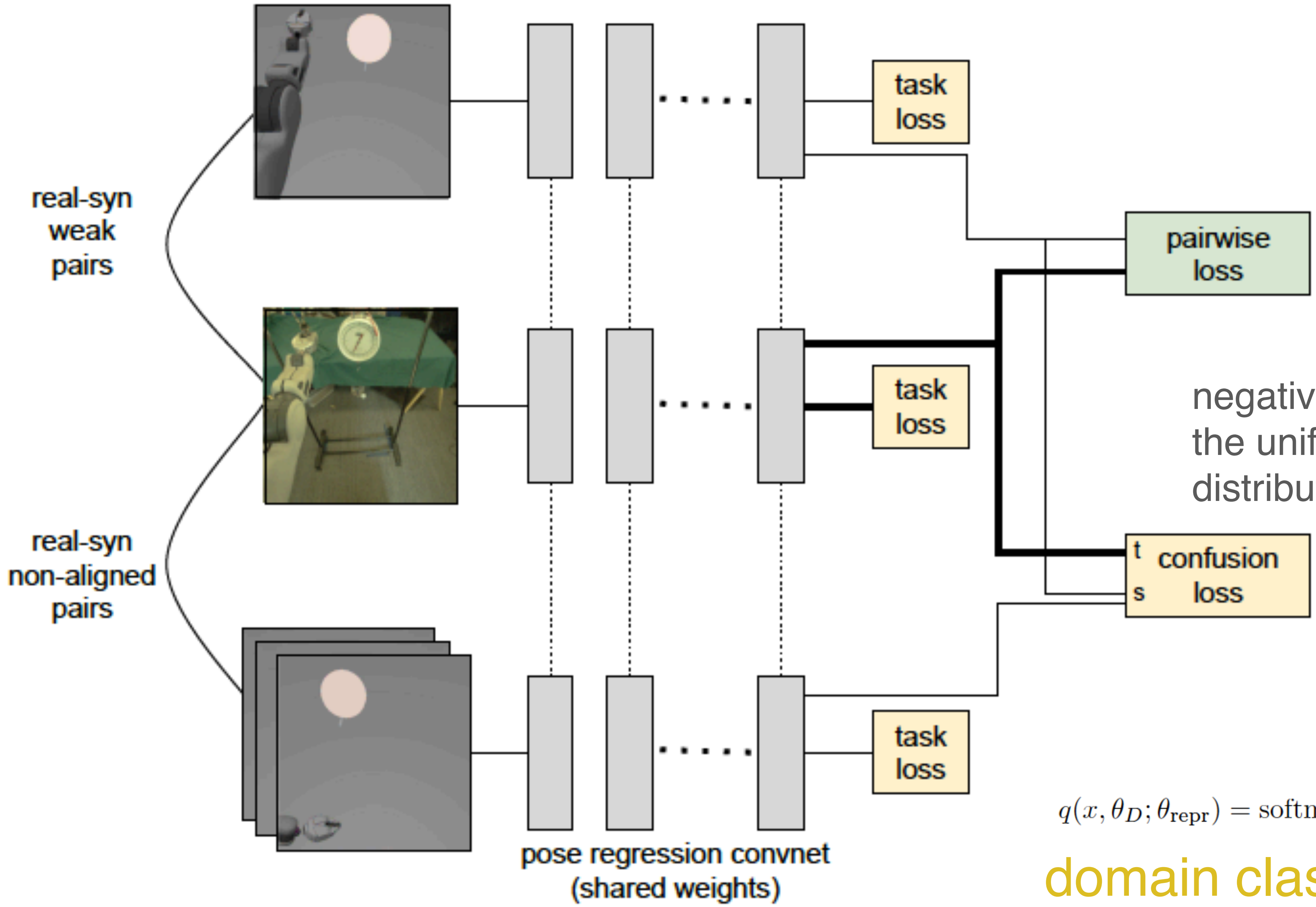
Domain adaptation

task loss: object pose estimation, so that we learn features relevant to objects presents that can be used in GPS



Domain adaptation

task loss: object pose estimation, so that we learn features relevant to objects presents that can be used in GPS



$\left\| f(x_S^{(i)};\theta_{\text{repr}}) - f(x_T^{(j)};\theta_{\text{repr}}) \right\|_2$

negative cross entropy between the uniform distribution and the distribution over domains

$-\sum_{x\in(x_S \cup x_T)} \sum_d \frac{1}{D} \log q_d(x,\theta_D,\theta_{\text{repr}})$

$q(x,\theta_D;\theta_{\text{repr}}) = \text{softmax}(\theta_D^T f(x;\theta_{\text{repr}}))$

domain classifier:

tries to assign each image

to the domain is comes from

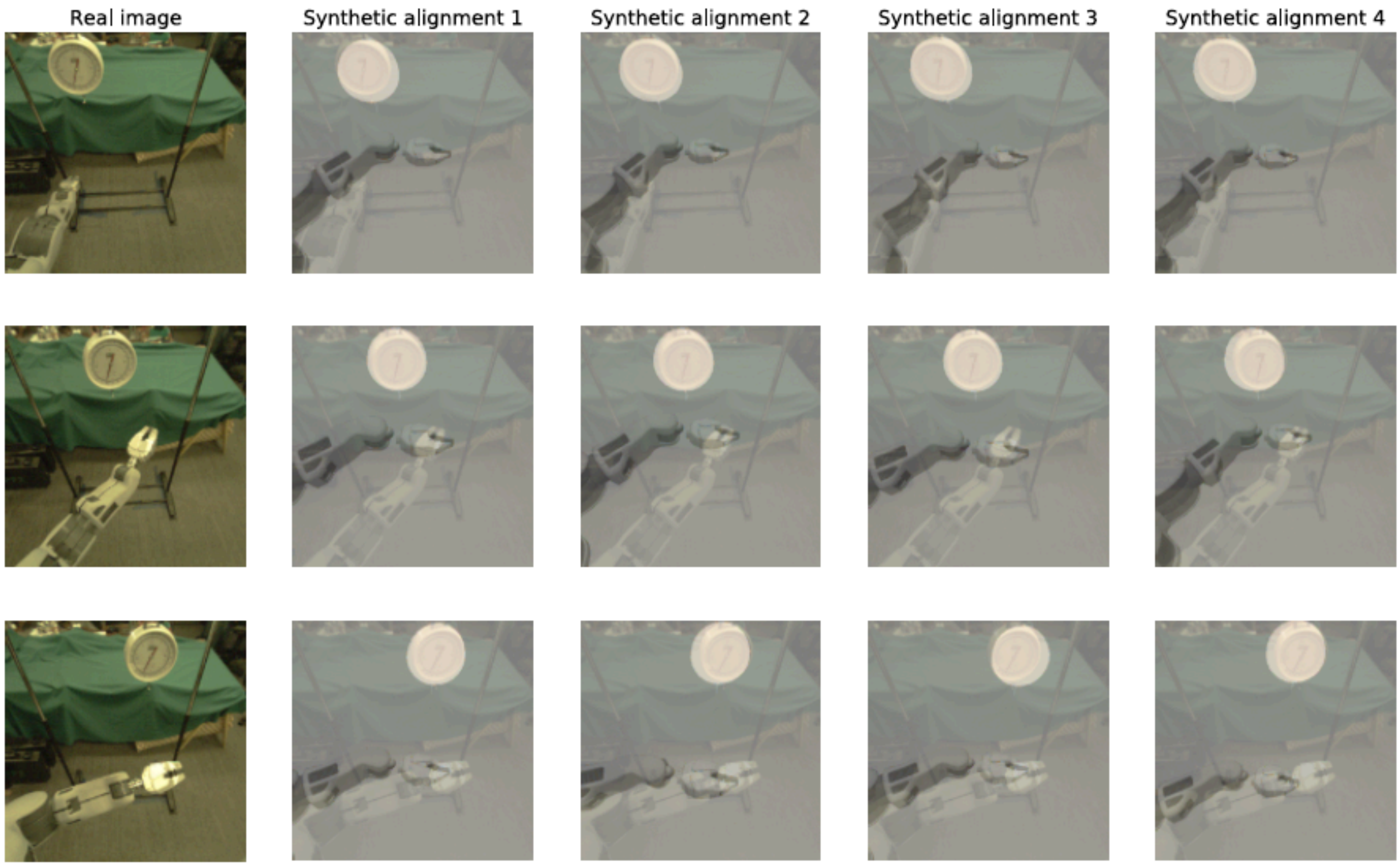
Weakly supervised domain adaptation

Mine image pairs as you go

Algorithm 1 Learning domain-invariant image features

1: Collect x_S source domain images with labeled object pose
2: Collect x_T target domain images
3: Minimize $\mathcal{L}_\phi(x_S, \phi_S; \theta_\phi, \theta_{\text{repr}}) + \lambda \mathcal{L}_{\text{conf}}(x_S, x_T, \theta_D; \theta_{\text{repr}})$ with respect to $\theta_\phi, \theta_{\text{repr}}$
4: **for** $x_T^{(j)}$ in x_T **do**
5: $\quad i^* = \arg\min_i ||f_{\text{conv1}}(x_S^{(i)}; \theta_{\text{repr}}) - f_{\text{conv1}}(x_T^{(j)}; \theta_{\text{repr}})||_2$
6: $\quad$ Add (i^*, j) to P
7: **end for**
8: Minimize $\mathcal{L}(x_S, \phi_S, x_T, \phi_T, P, \theta_D; \theta_\phi, \theta_{\text{repr}})$ with respect to $\theta_\phi, \theta_{\text{repr}}$

Task: match hook position while ignoring the arm



Adaptation results

Table 3. Performance of visuomotor tasks trained using domain alignment with weakly supervised pairwise constraints. We report the percentage of successful attempts at placing a loop of rope on a hook after training with 12 iterations of GPS. Each experiment was repeated 3 times.

Method	**# Sim**	**# Real (unlabeled)**	**Success rate**
Synthetic only	4000	0	38.1% ± 8%
Autoencoder (100)	0	100	28.6% ± 25%
Autoencoder (500)	0	500	33.2% ±15%
Domain alignment with randomly assigned pairs	4000	100	33.3% ±16%
Domain alignment with weakly supervised pairwise constraints	4000	100	**76.2% ± 16%**
Oracle	0	500 (labeled)	71.4% ± 14%

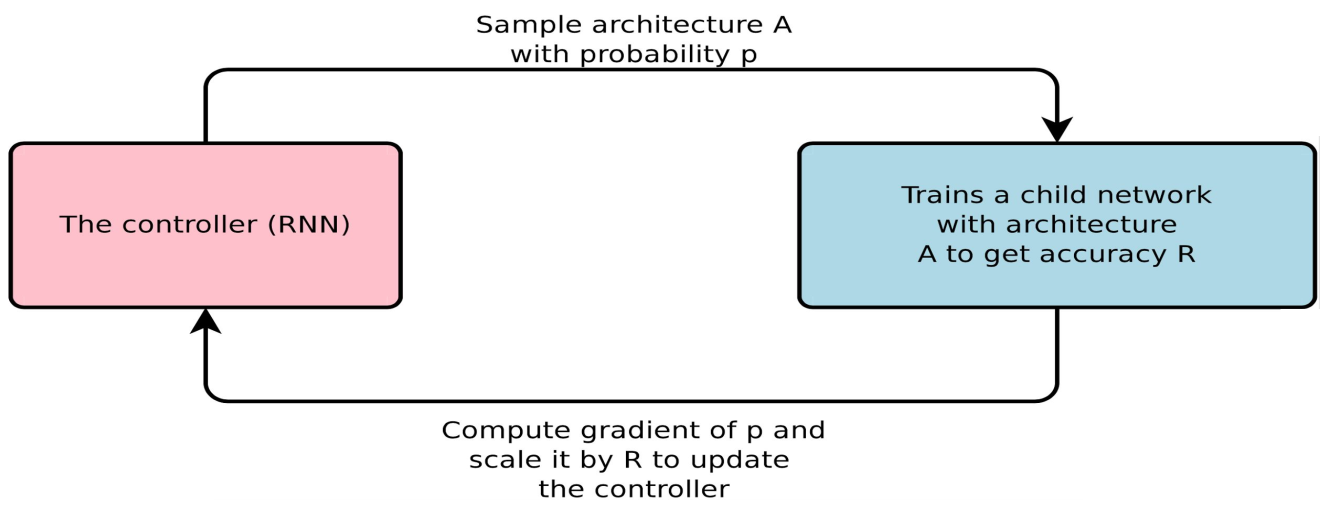
Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Estimating or Propagating Gradients (cont.)

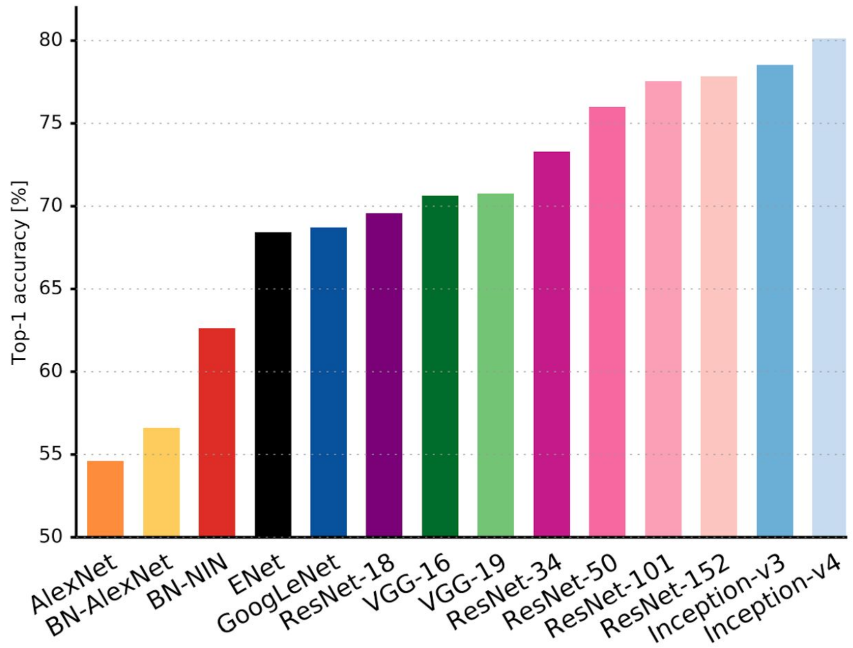
Katerina Fragkiadaki

Architecture search with REINFORCE

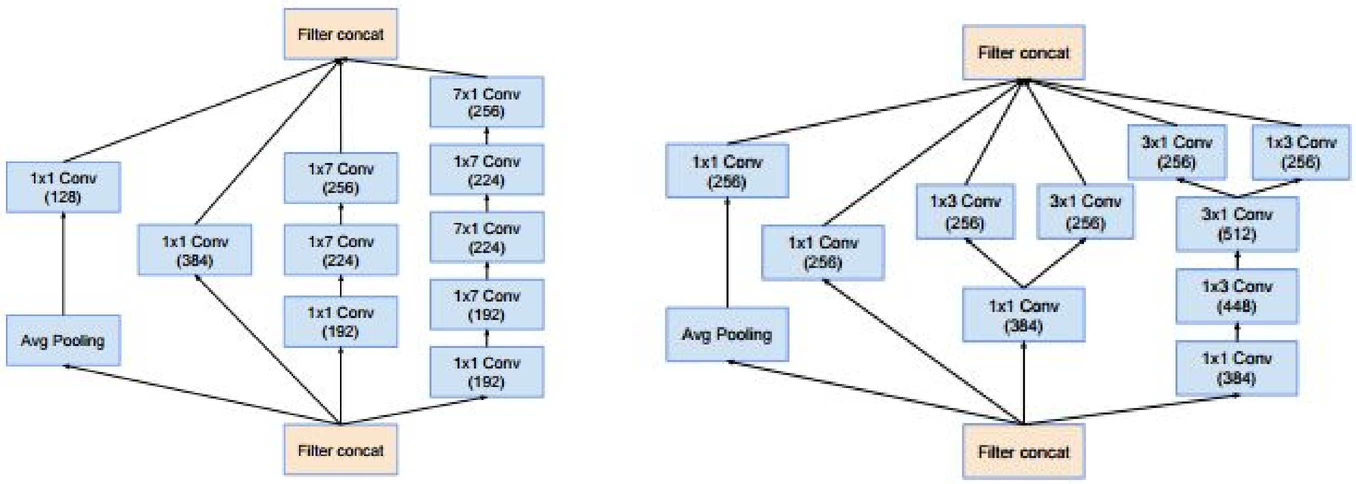


Motivation for Architecture Search

- Can we try and learn good architecture automatically replacing human intuition?



Canziani et al, 2017



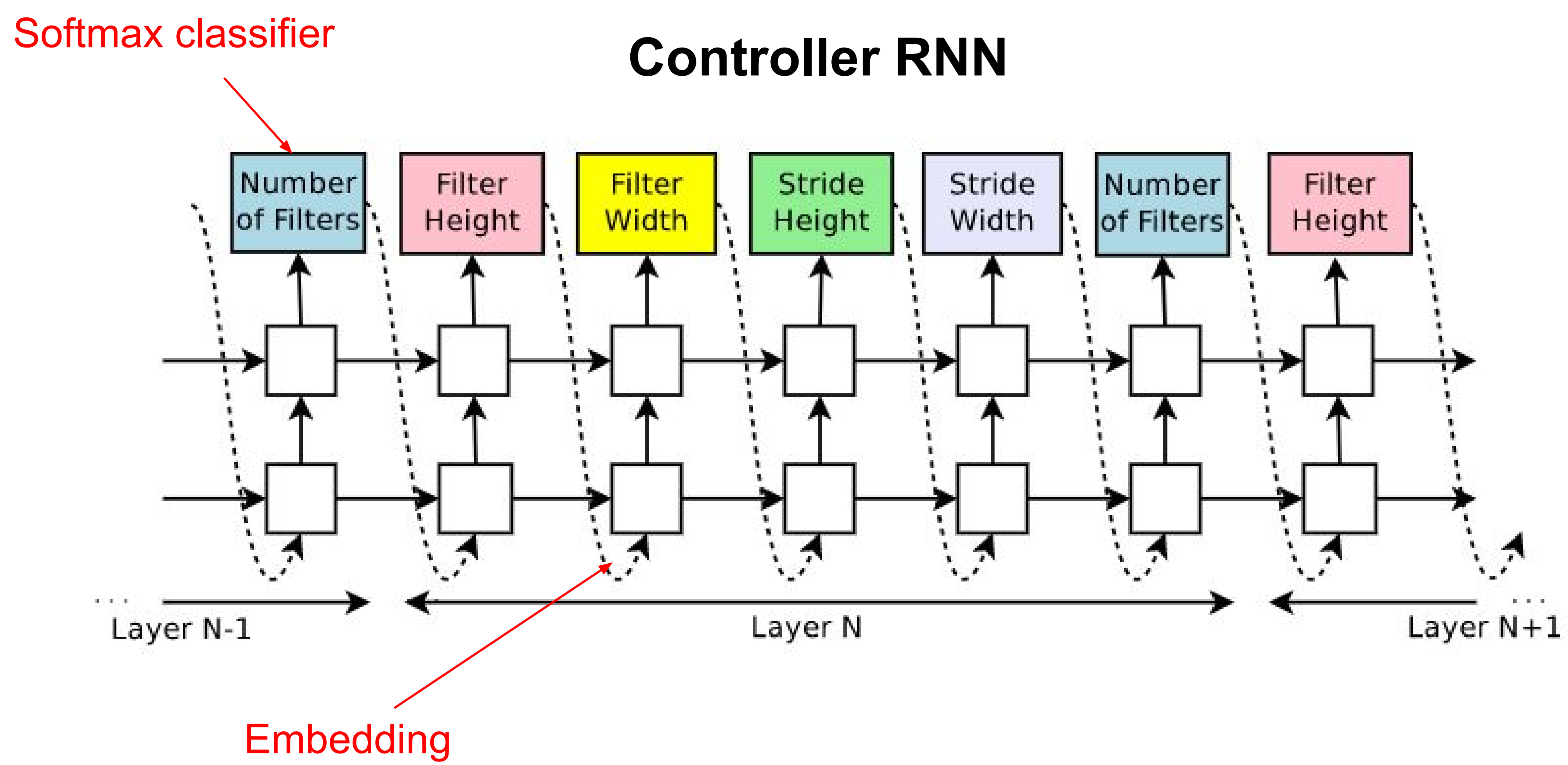
Two layers from the famous Inception V4 computer vision model.
Szegedy et al, 2017

Google

Neural Architecture Search

- Specify the structure and connectivity of a neural network by using a configuration string
    - ["Filter width: 5", "Filter Height: 3", "Num Filters: 24"]
- Use a RNN ("Controller") to generate this string that specifies a neural network architecture
- Train this architecture ("Child Network") to see how well it performs on a validation set
- Use reinforcement learning to update the parameters of the Controller model based on the accuracy of the child model

Neural Architecture Search for Convolutional Networks



Training with REINFORCE

Parameters of Controller RNN

Accuracy of architecture on
held-out dataset

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Architecture predicted by the controller RNN
viewed as a sequence of actions

Training with REINFORCE

Parameters of Controller RNN

Accuracy of architecture on held-out dataset

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Architecture predicted by the controller RNN viewed as a sequence of actions

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\big[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\big]$$

Training with REINFORCE

Parameters of Controller RNN

Accuracy of architecture on held-out dataset

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

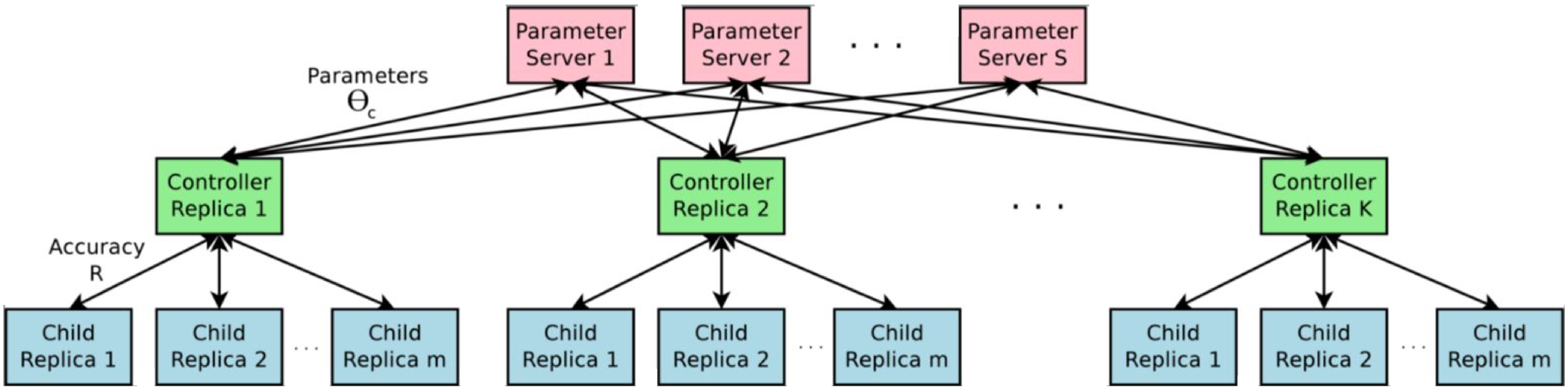
Architecture predicted by the controller RNN viewed as a sequence of actions

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\big[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\big]$$

Number of models in minibatch

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R_k$$

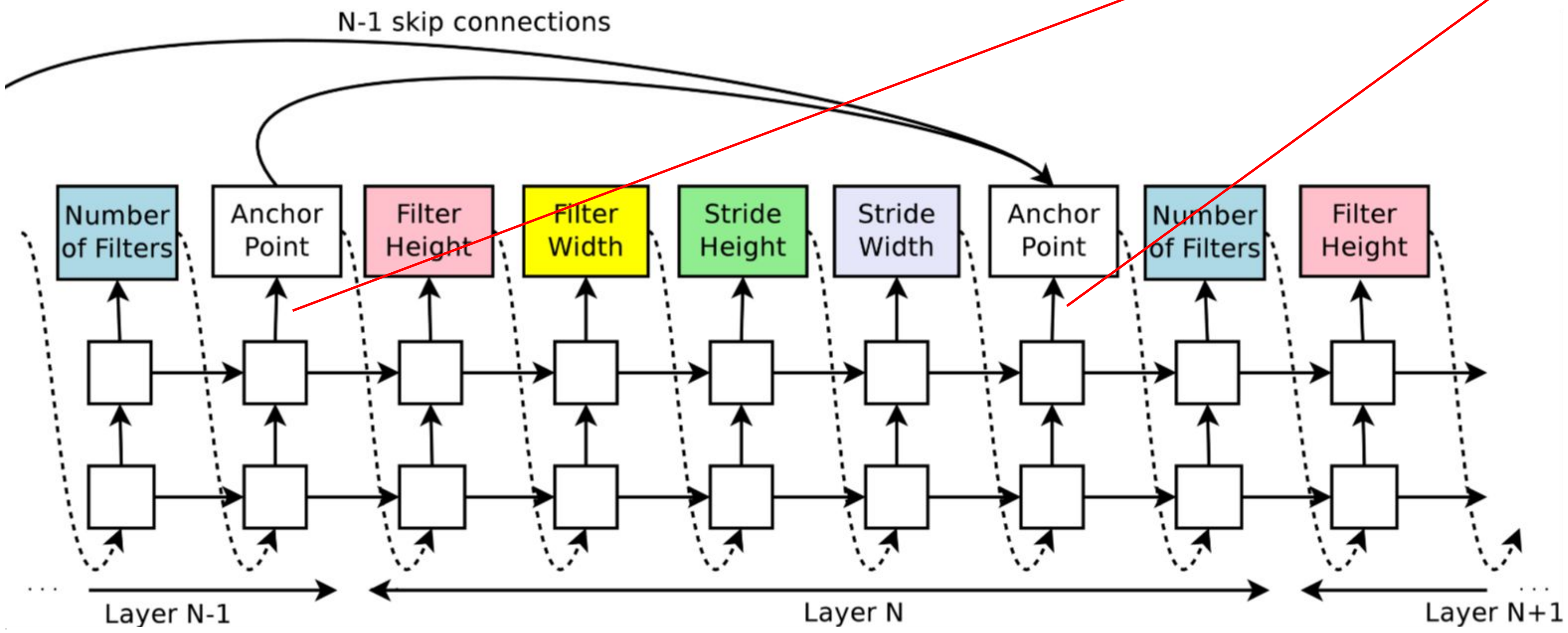
Distributed Training



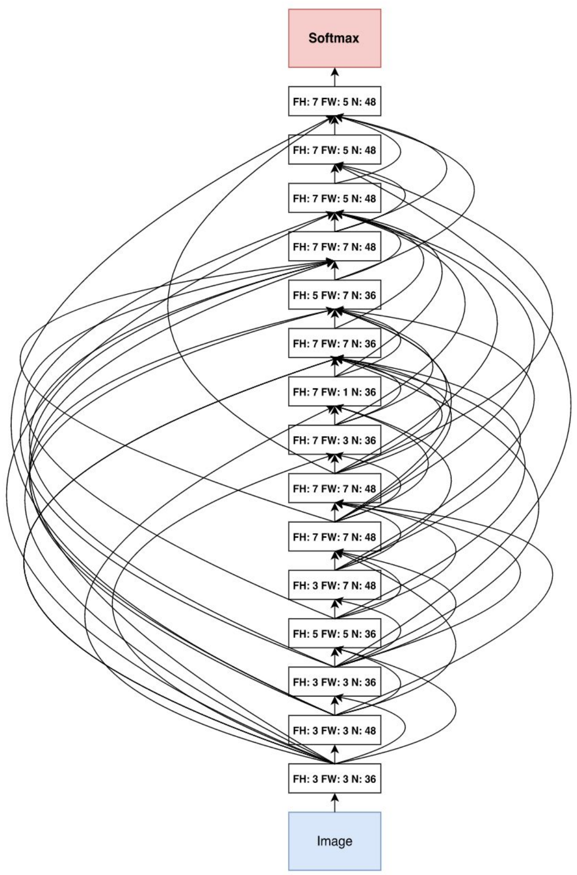
Neural Architecture Search fro CIFAR-10

Weight Matrices

$$\mathrm{P}(\text{Layer j is an input to layer i}) = \mathrm{sigmoid}(v^{\mathrm{T}}\tanh(W_{prev} * h_j + W_{curr} * h_i))$$



Neural Architecture Search fro CIFAR-10



Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

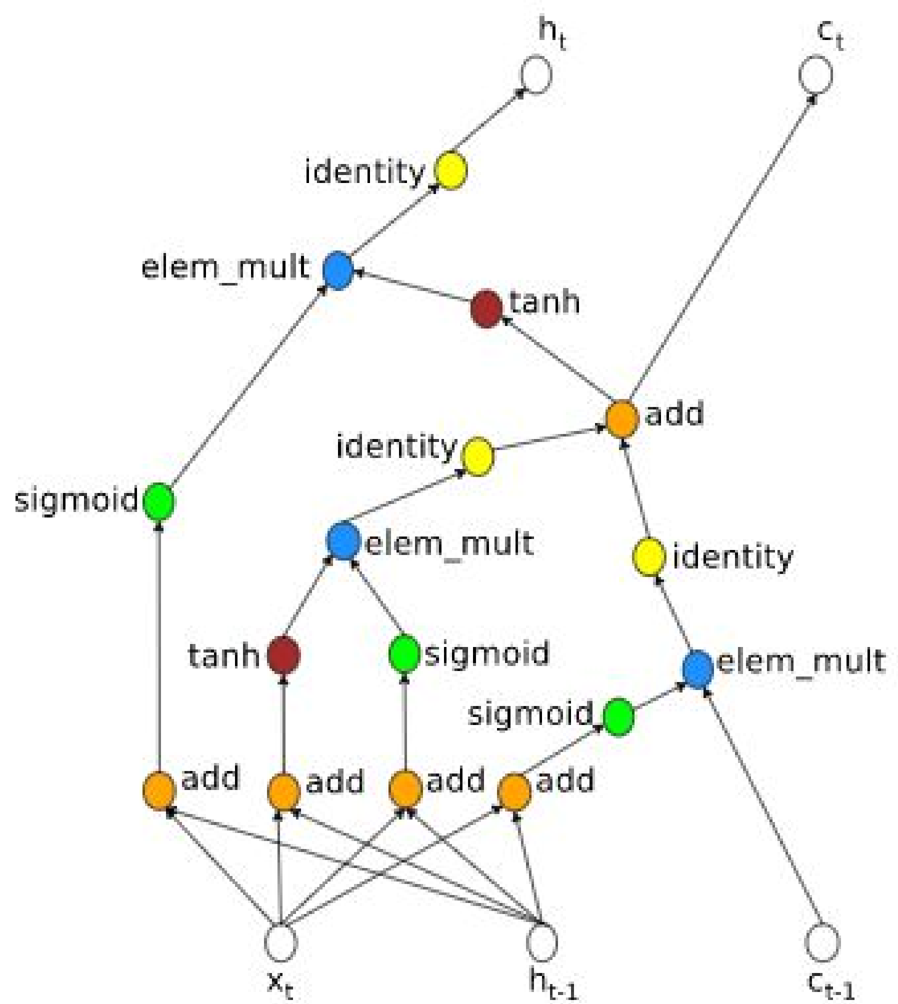
5% faster

Best result of evolution (Real et al, 2017): 5.4%
Best result of Q-learning (Baker et al, 2017): 6.92%

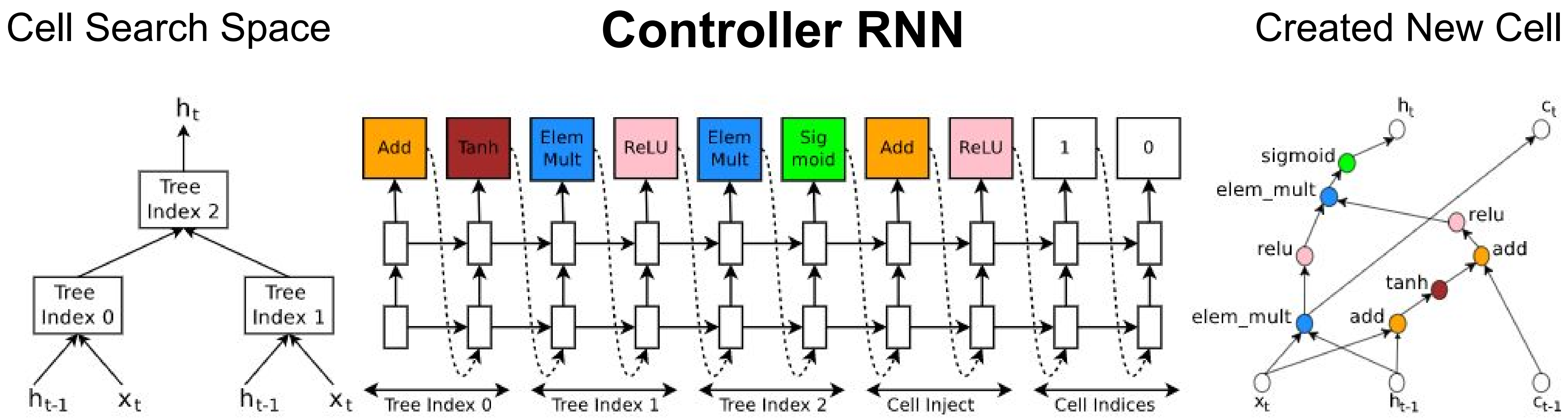
Google

Recurrent Cell Prediction Method

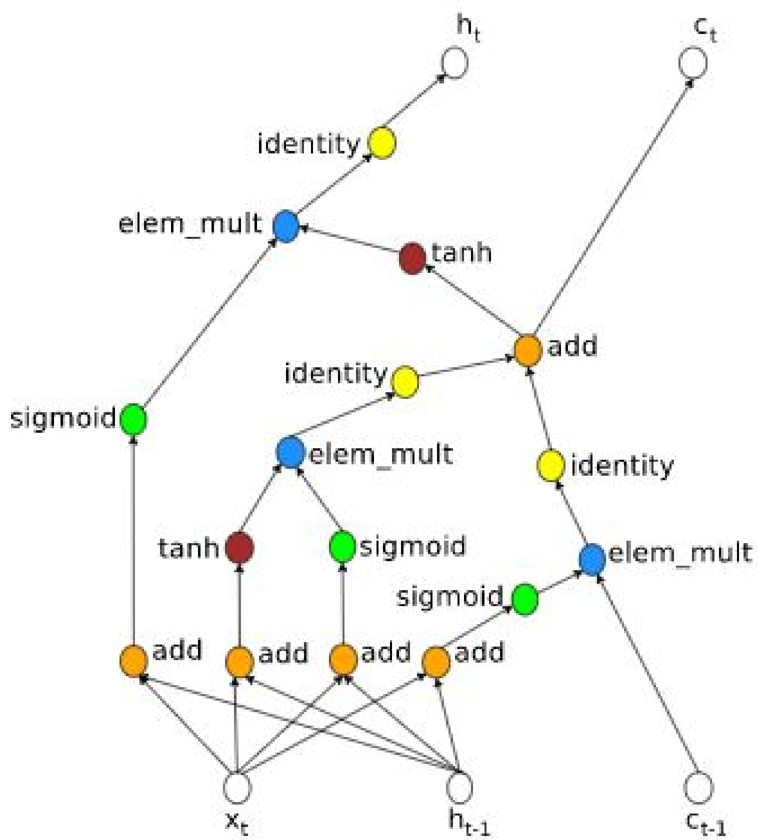
- Created a search space for search over RNN cells like the LSTM or GRU
- Based our search space off the LSTM cell in that we have recurrent state and cell



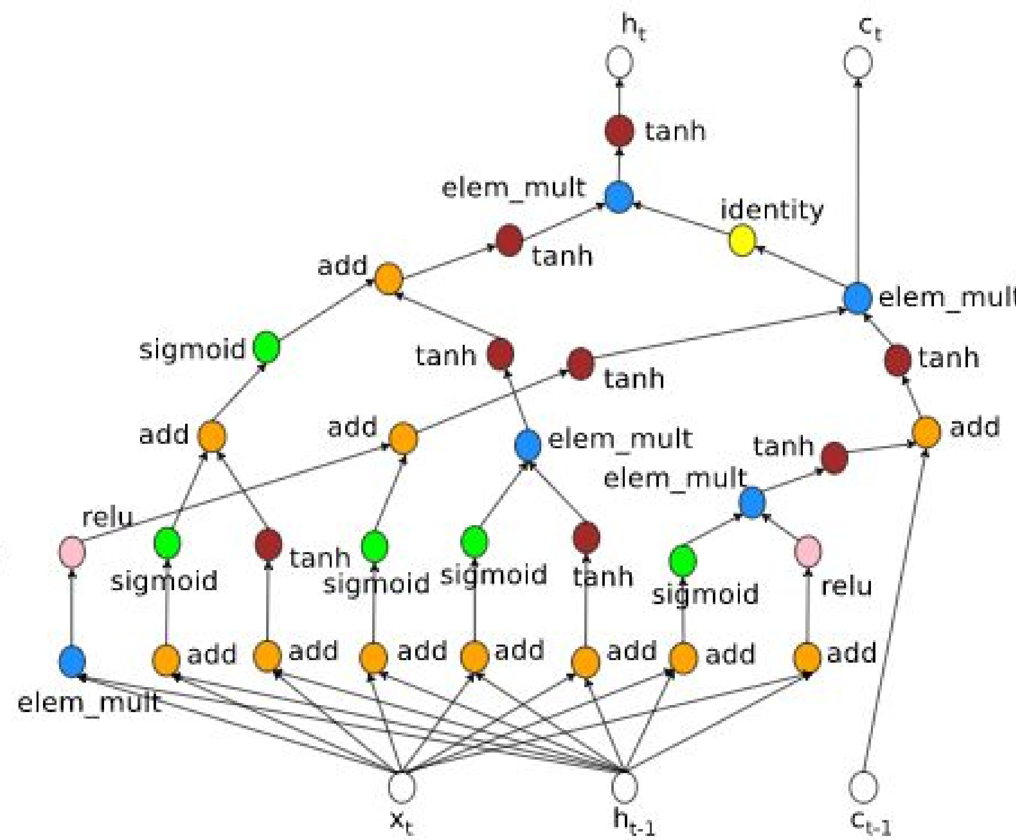
Recurrent Cell Prediction Method



Penn Treebank Results



LSTM Cell



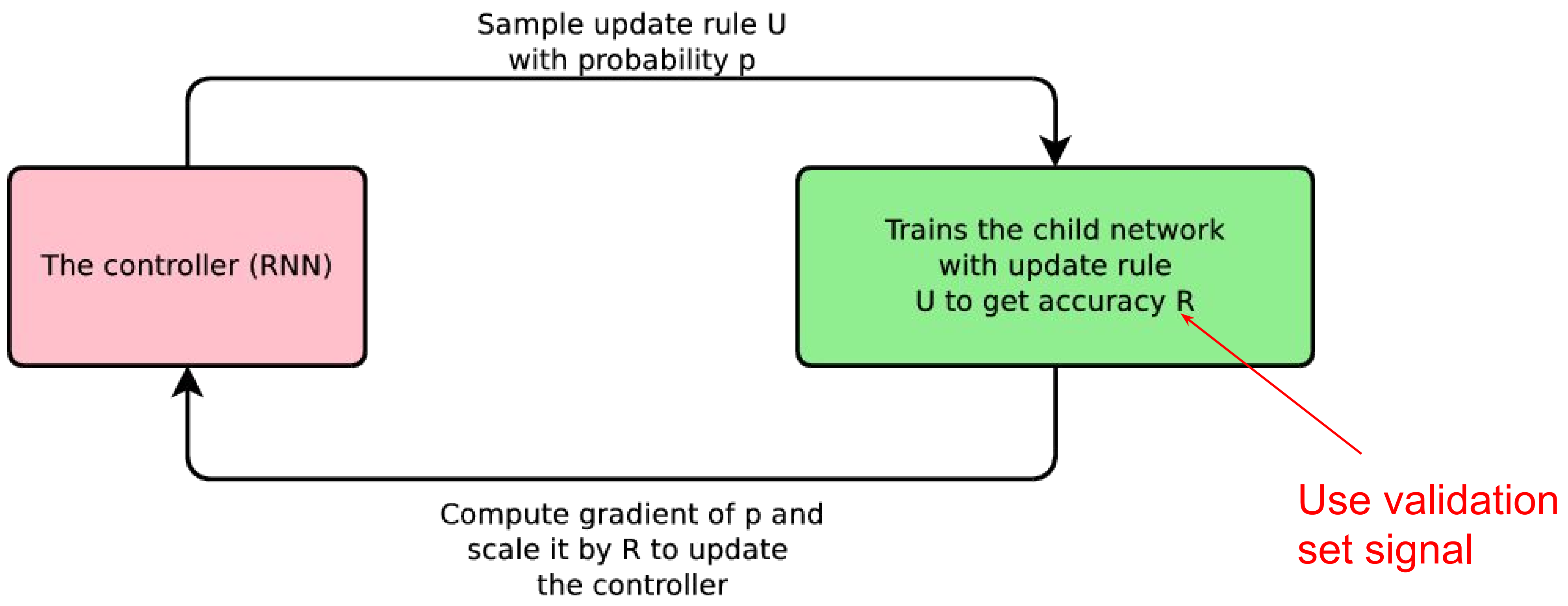
Neural Architecture Search (NAS) Cell

Penn Treebank Results

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

2x as fast

Neural Optimizer Search



Architecture search with Pathwise derivatives

OUTRAGEOUSLY LARGE NEURAL NETWORKS:
THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer[1], Azalia Mirhoseini[*†1], Krzysztof Maziarz[*2], Andy Davis[1], Quoc Le[1], Geoffrey Hinton[1] and Jeff Dean[1]

[1]Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com
[2]Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

As the training data increases, model capacity increases to keep accuracy high.

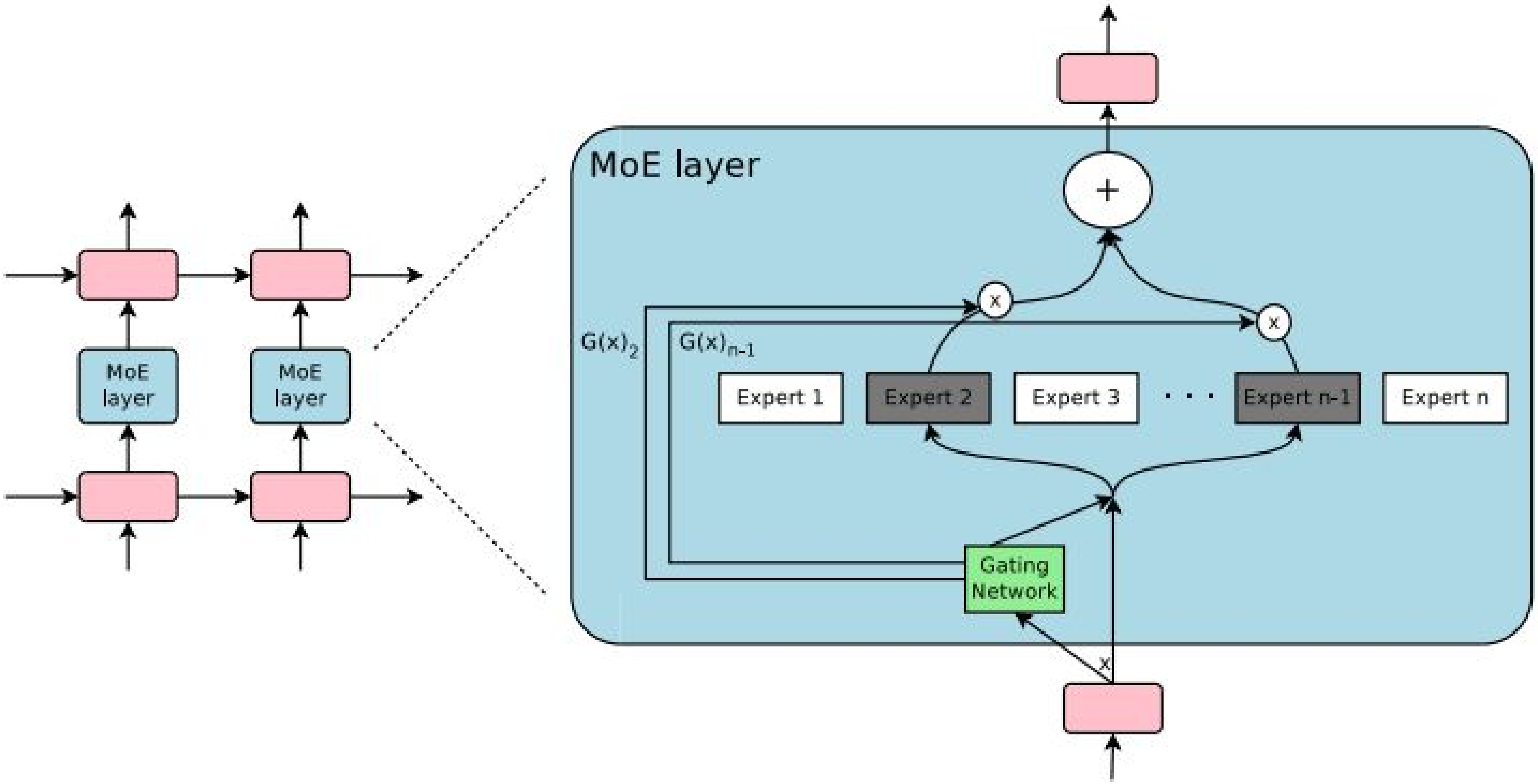
This in turn increases cost of every example, at both train and test time.

Can we do better?

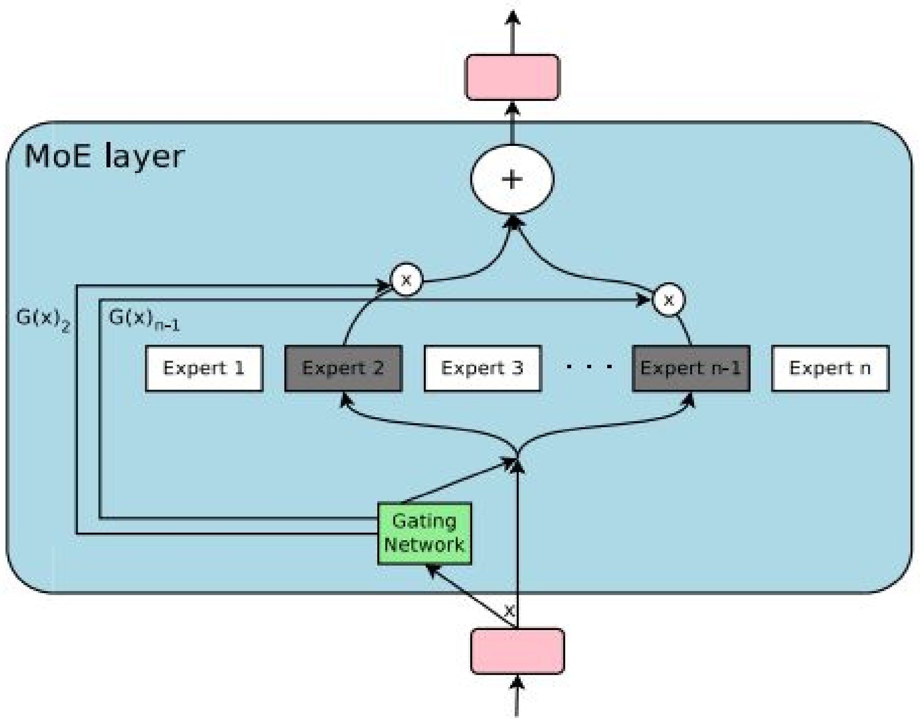
Yes, with conditional computation: specialize the model per example, not all examples will share the same model

We need to learn: 1) how examples will be distributed to models 2) the models themselves

Per example routing



The gating function



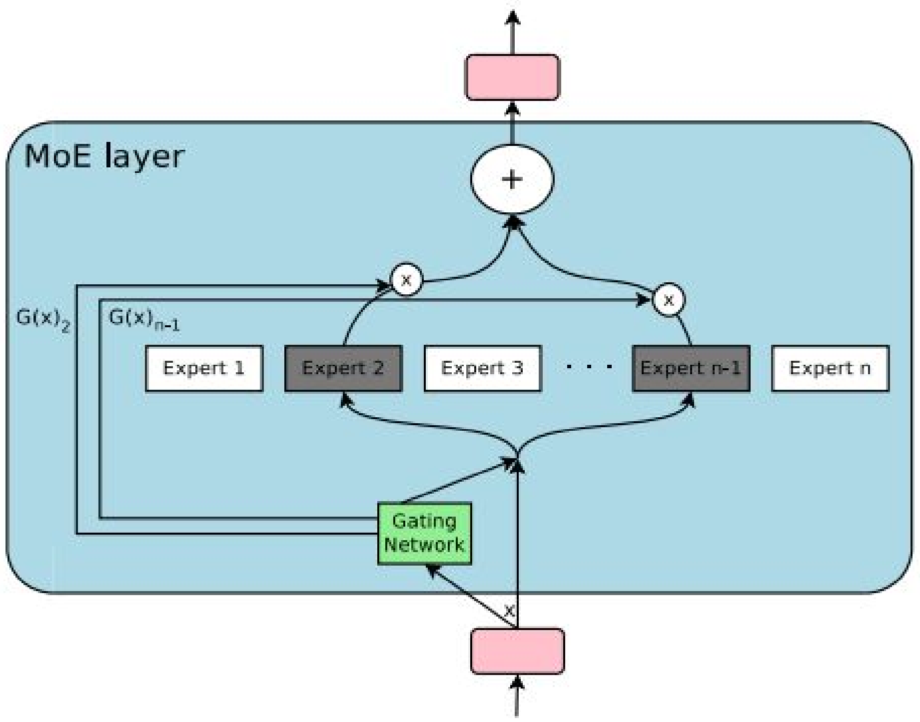
$$G_\sigma(x) = Softmax(x \cdot W_g)$$

$$G(x) = Softmax(KeepTopK(H(x), k)) \tag{3}$$

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i) \tag{4}$$

$$KeepTopK(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases} \tag{5}$$

The gating function



$$G_\sigma(x) = Softmax(x \cdot W_g)$$

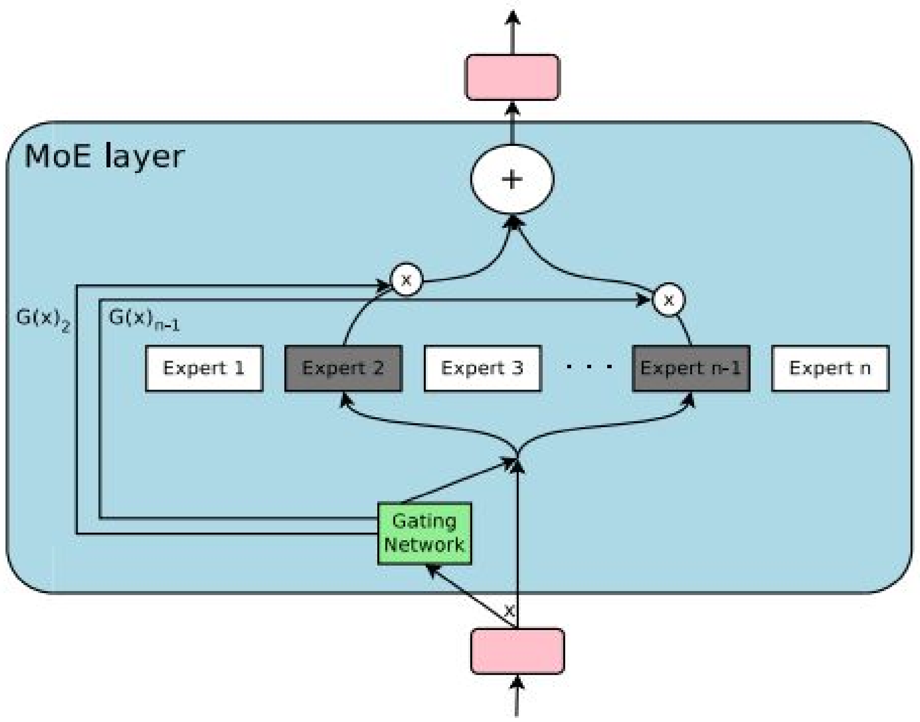
$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$KeepTopK(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

noise from a fixed distribution

The gating function



$$G_\sigma(x) = Softmax(x \cdot W_g)$$

$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$KeepTopK(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

noise from a fixed distribution

this behaves like variance (positive)

Per-Example Routing

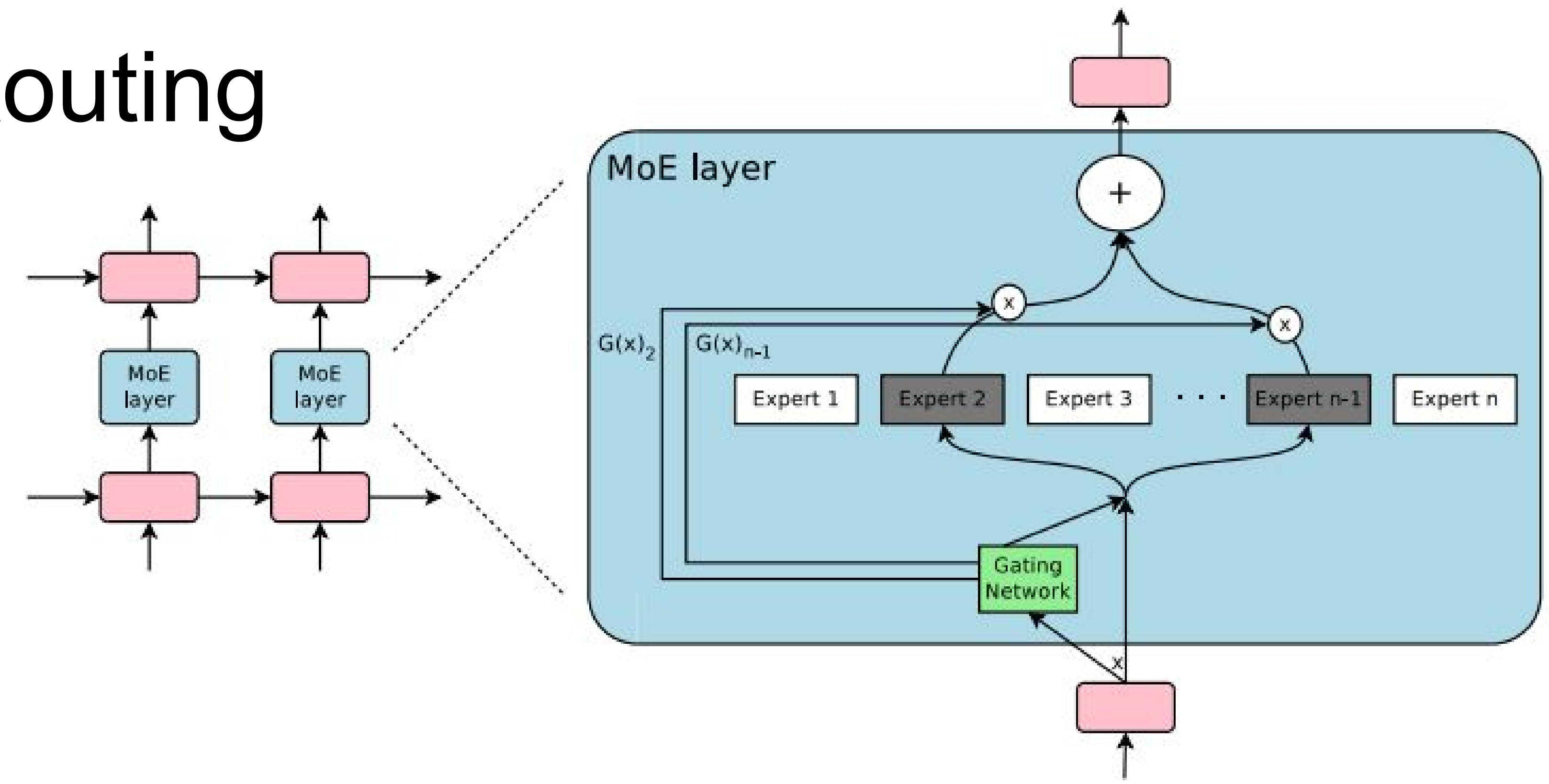


Table 7: Perplexity and BLEU comparison of our method against previous state-of-art methods on the Google Production En→ Fr dataset.

Model	Eval Perplexity	Eval BLEU	Test Perplexity	Test BLEU	Computation per Word	Total #Parameters	Training Time
MoE with 2048 Experts	2.60	37.27	2.69	36.57	100.8M	8.690B	1 day/64 k40s
GNMT (Wu et al., 2016)	2.78	35.80	2.87	35.56	214.2M	246.9M	6 days/96 k80s

Language generation using REINFORCE

Adversarial Learning for Neural Dialogue Generation

Jiwei Li[1], **Will Monroe**[1], **Tianlin Shi**[1],
Sébastien Jean[2], **Alan Ritter**[3] and **Dan Jurafsky**[1]

[1]Stanford University, CA, USA
[2]New York University, NY, USA
[3]Ohio State University, OH, USA
jiweil,wmonroe4,tianlins,jurafsky@stanford.edu
sebastien@cs.nyu.edu
ritter.1492@osu.edu

Actions: words samples (softmax over the vocabulary)

Reward (at each subsequence generated): confusion of a discriminator (likelihood of being real), which is trained to tell apart generated from real sentences

$$J(\theta) = \mathbb{E}_{y \sim p(y|x)}(Q_+(\{x, y\})|\theta)$$

$$\nabla J(\theta) \approx \sum_t (Q_+(x, Y_t) - b(x, Y_t)) \nabla \log p(y_t|x, Y_{1:t-1})$$

Language generation using REINFORCE

For number of training iterations **do**
. **For** i=1,D-steps **do**
. Sample (X,Y) from real data
. Sample $\hat{Y} \sim G(\cdot|X)$
. Update D using (X, Y) as positive examples and $(X, \hat{Y})$ as negative examples.
. **End**
.
. **For** i=1,G-steps **do**
. Sample (X,Y) from real data
. Sample $\hat{Y} \sim G(\cdot|X)$
. Compute Reward r for $(X, \hat{Y})$ using D.
. Update G on $(X, \hat{Y})$ using reward r
. Teacher-Forcing: Update G on (X, Y)
. **End**
End

Language generation using pathwise derivatives

GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution

Matt J. Kusner
Alan Turing Institute
University of Warwick

José Miguel Hernández-Lobato
University of Cambridge

$$[\text{softmax}(\mathbf{h})]_i = \frac{\exp(\mathbf{h}_i)}{\sum_{j=1}^{K} \exp(\mathbf{h}_j)}$$

$$\mathbf{y} = \text{one\_hot}(\arg\max_i (h_i + g_i))$$

$$\mathbf{y} = \text{softmax}(1/\tau(\mathbf{h} + \mathbf{g}))$$

Language generation using pathwise derivatives

Algorithm 1: Generative Adversarial Network [14]

data: $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \sim p(\mathbf{x})$,
Generative LSTM network G_Θ
Discriminative LSTM network D_Φ
while loop until convergence **do**
    Sample mini-batch of inputs $B = \{\mathbf{x}_{B_1}, \ldots, \mathbf{x}_{B_m}\}$
    Sample noise $N = \{\mathbf{z}_{N_1}, \ldots, \mathbf{z}_{N_m}\}$
    Update discriminator $\Phi = \mathrm{argmin}_\Phi -\frac{1}{m}\sum_{\mathbf{x}\in B} \log D_\Phi(\mathbf{x}) - \frac{1}{m}\sum_{\mathbf{z}\in N} \log(1 - D_\Phi(G_\Theta(\mathbf{z})))$
    Update generator $\Theta = \mathrm{argmin}_\Theta -\frac{1}{m}\sum_{\mathbf{z}\in N} \log \frac{D_\Phi(G_\Theta(\mathbf{z}))}{1-D_\Phi(G_\Theta(\mathbf{z}))}$
end while