Sim-To-Real Robot Learning with Progressive Nets

Andrei A. Rusu

with contributions from many others

Google DeepMind

Our Mission

1 • Solve intelligence

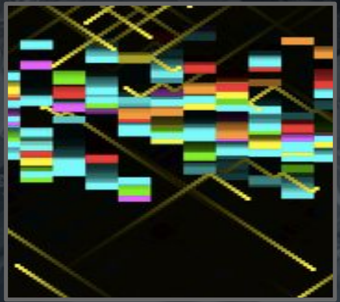
2 • Use it to solve everything else

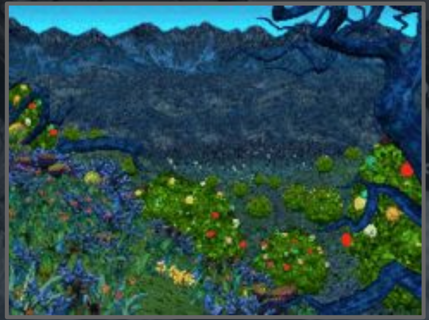
Google DeepMind

DeepMind Research Premise:

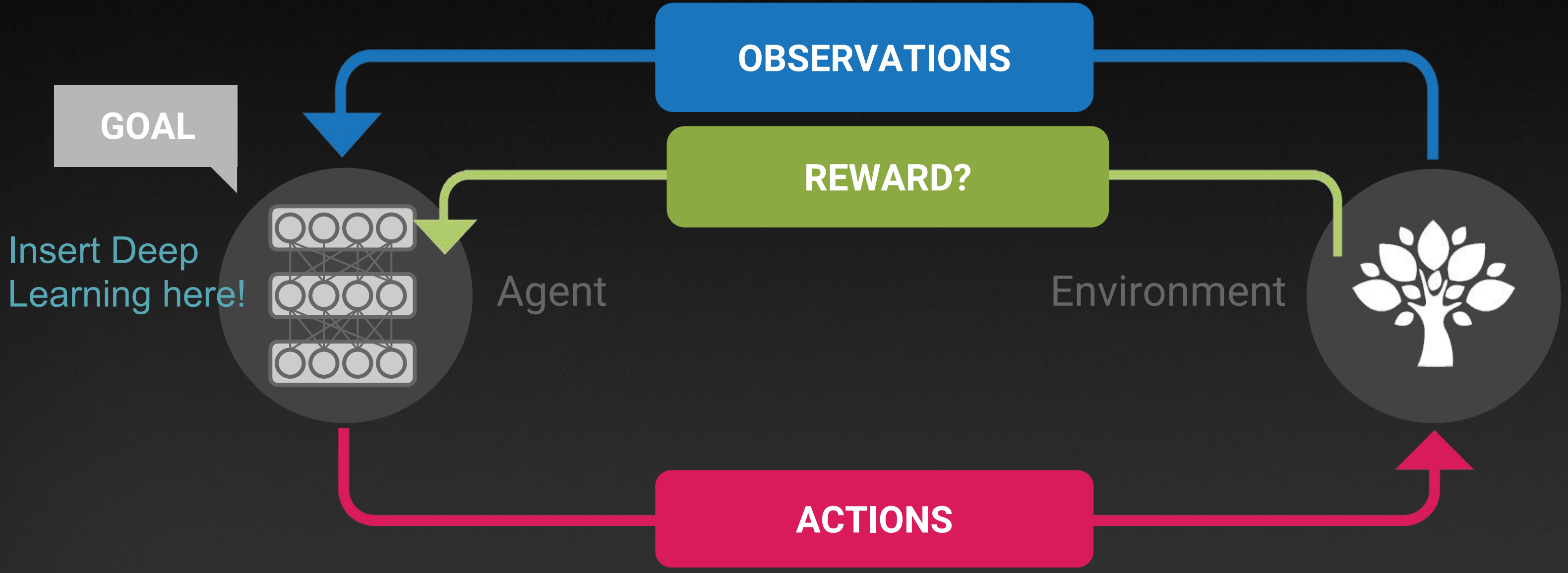
Simulations & Games are perfect platforms for developing AI algorithms! Why?

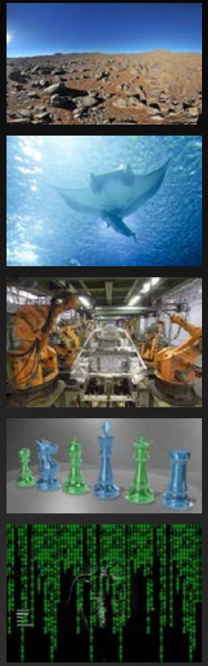
1. Difficult and interesting for humans.
2. Huge variety of games, challenging in many different ways: speed, accuracy, memory, comprehension, logic...
3. Built-in Evaluation criteria and *Reward!*



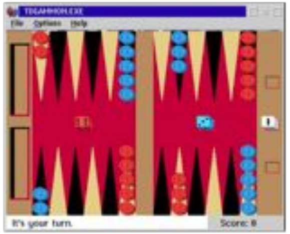
Reinforcement Learning





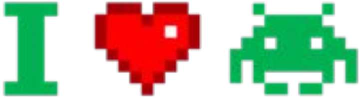
Deep RL (just a few examples)

TD-Gammon (Tesauro, 1989-1995)

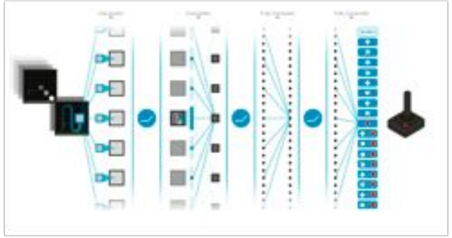


Slot car driving (Lange & Riedmiller, 2012)

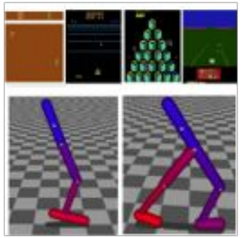
Arcade Learning Environment (Bellemare et al, 2013)



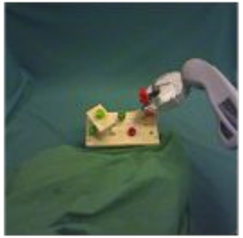
Deep Q-Networks (2013, 2015)



Trust region policy optimization (Schulman et al, 2015)



End-to-end training on real robots (Levine et al, 2015)



DeepMind (2016)

AlphaGo

DeepMind’s first contribution: Atari Agents

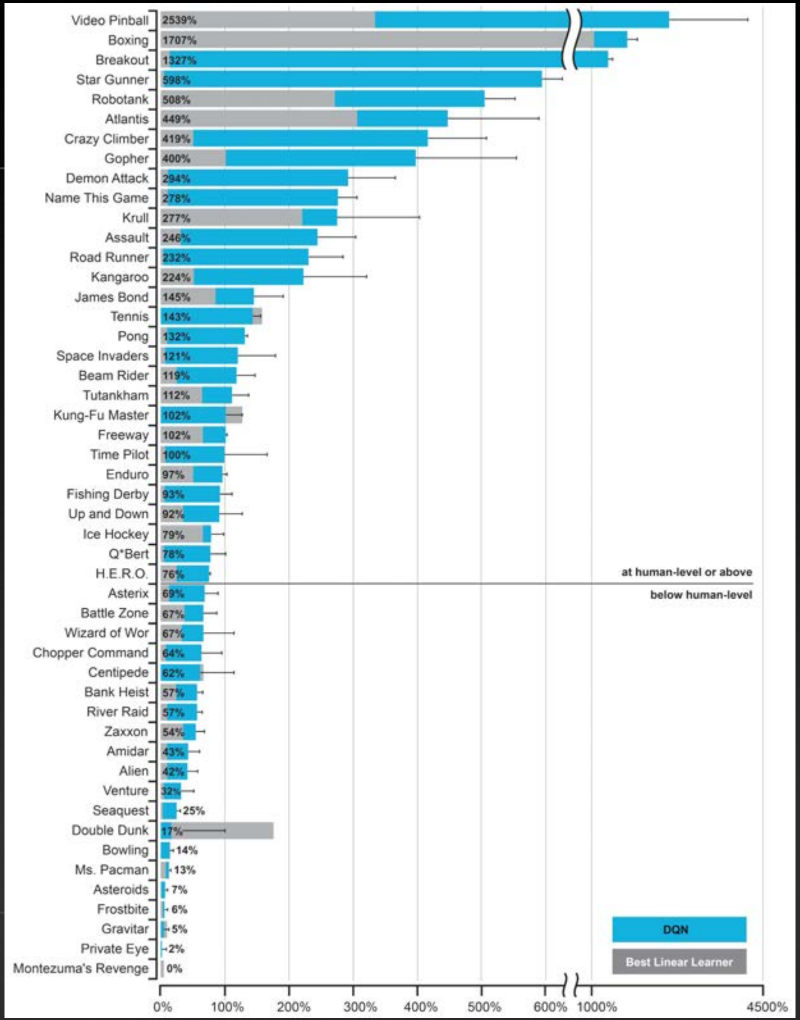
Atari 2600 testbed: 100+ classic 8-bit Atari games from the 80s

- Agents just get the raw pixels as inputs (~33K).
- Goal is simply to maximize score.
- Everything learnt from scratch.
- ***One*** system to play ***all*** the different games.

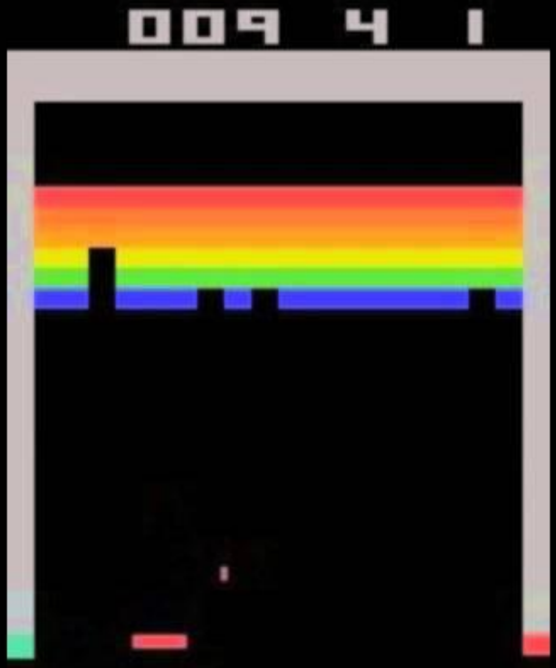
DQN: Deep Q-Learning

- State-of-the-art model-free approach to RL using deep networks.
- Works in environments with discrete action choices.
- Has achieved super-human performance on a variety of Atari 2600 games (Mnih et al., Nature 2015).
- DQN predicts the ***average discounted future return*** of each possible action from raw visual input.
- Uses a replay memory and a target network which stabilize learning over a wide-range of problems.


INNOVATIONS IN
The microbiome
nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
PAGES 486 & 529
EPIDEMIOLOGY
SHARE DATA IN OUTBREAKS
Forge open access to sequences and more
PAGE 477
COSMOLOGY
A GIANT IN THE EARLY UNIVERSE
A supermassive black hole at a redshift of 6.3
PAGES 490 & 512
QUANTUM PHYSICS
TELEPORTATION FOR TWO
Transferring two properties of a single photon
PAGES 491 & 516
NATURE.COM/NATURE
26 February 2015 £10
Vol. 518, No. 7540


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*Bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pacman 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
at human-level or above
below human-level
DQN
Best Linear Learner
0%
100%
200%
300%
400%
500%
600%
1000%
4500%

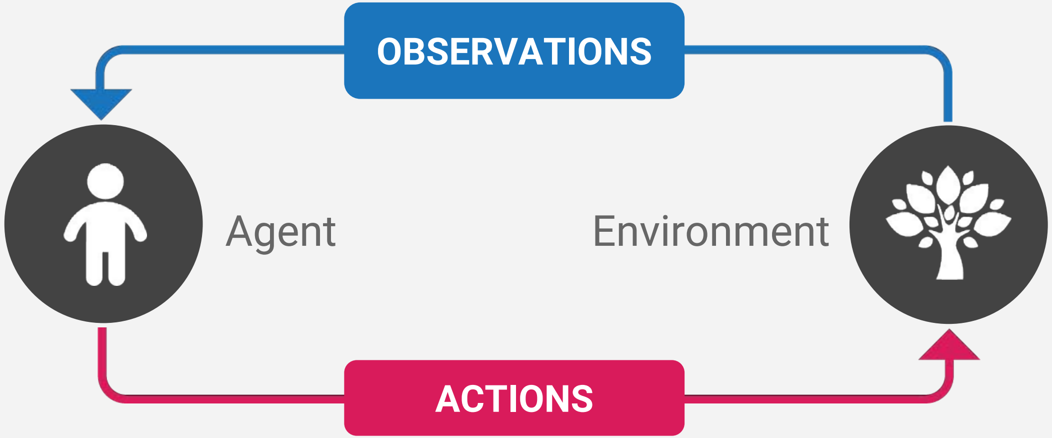
Breakout



Deep Reinforcement Learning

Two state of the art approaches explained… (quickly)

Reinforcement Learning



- General Purpose Framework for AI
- Agent interacts with the environment
- Select actions to maximise reward

Action-Value Function

- Maximise total future reward

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$

- For a policy π the action value function *Q*:

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s, a_t = a] \\ &= \mathbb{E}[r_{t+1} + \gamma Q^{\pi}(s', a') | s_t = s, a_t = a] \end{aligned}$$

- How good is action *a* in state *s*.
  - Greedy - follow the max
  - ε-greedy - follow the max with (1-ε) probability and random o/w.

Value Iteration

- Maximizing $Q^{\pi}(s,a)$ over possible policies gives the optimal action-value function and the Bellman equation:

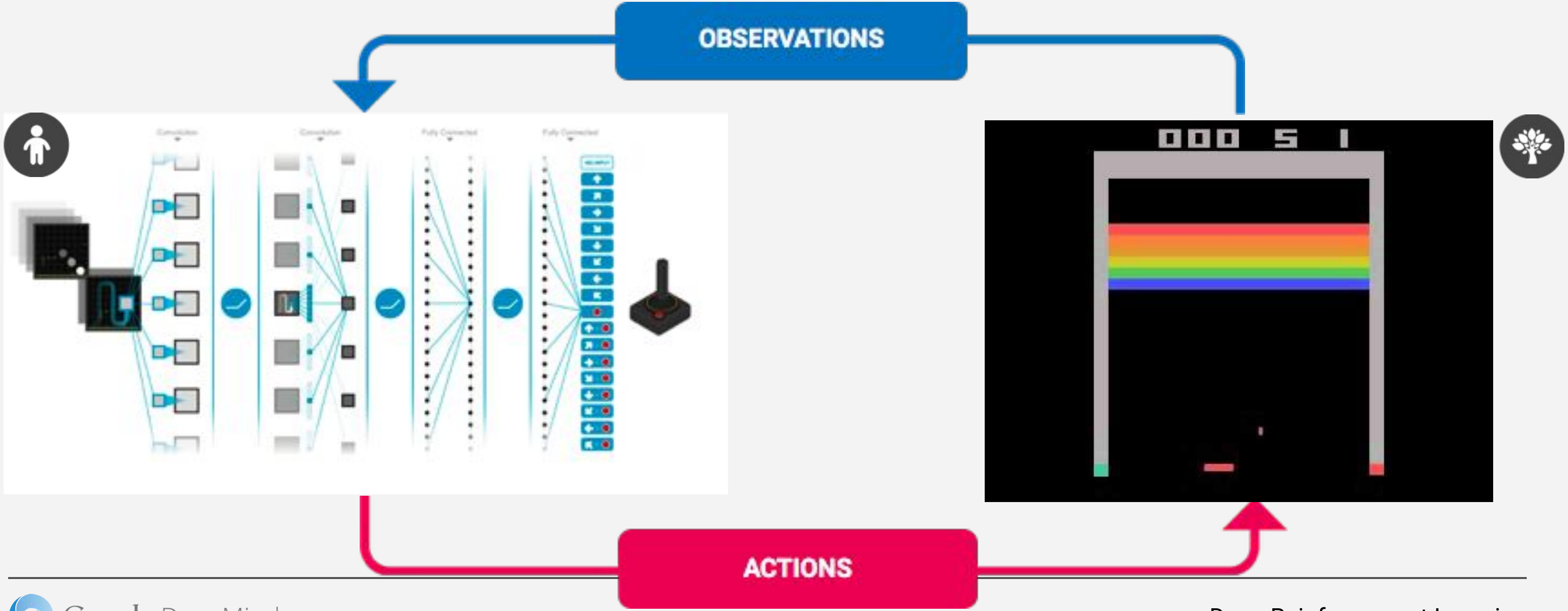
$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi\right]$$
$$= \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\right]$$

- Basic idea:
  - Approximate $\rightarrow Q(s,a;\theta) \approx Q^*(s,a)$
  - Apply the Bellman Equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q_i(s_{t+1}, a') | s_t = s, a_t = a\right]$$

End-to-End Reinforcement Learning

- Use a neural network for *Q(s,a;Θ)*
- Train end-to-end from raw pixels



End-to-End Reinforcement Learning

- We need a loss function to minimize

$$Q_{i+1}(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q_i(s_{t+1}, a') | s_t = s, a_t = a\right]$$

$$L_i(\theta_i) = \mathbb{E}\left(\underbrace{r + \gamma \ \max_{a'} Q(s', a'; \theta_i)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

- So now we can do our good old SGD update

$$\theta \leftarrow \theta - \eta \frac{\partial L(\theta)}{\partial \theta}$$

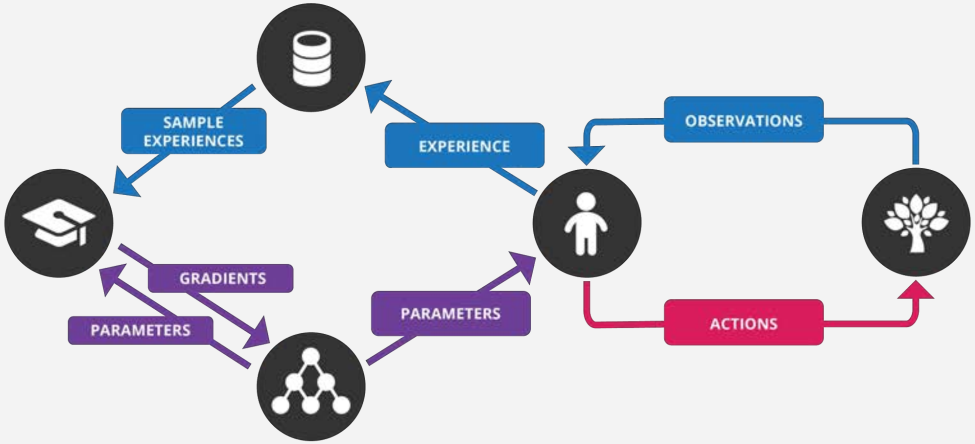
Deep Q-Network (DQN)

- Experiences in a sequence are correlated
  - Do not do online updates, store in replay memory
  - Sample from experience replay memory to apply Q-updates

- Targets can not depend on same θ_i → introduce **target network**

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

DQN



Initialize **target network** $\theta^{-} \leftarrow \theta$

For each time step ***t***

Take action a_t, and observe r_t, s_{t+1}

Sample (s,a,r,s') from **replay memory**

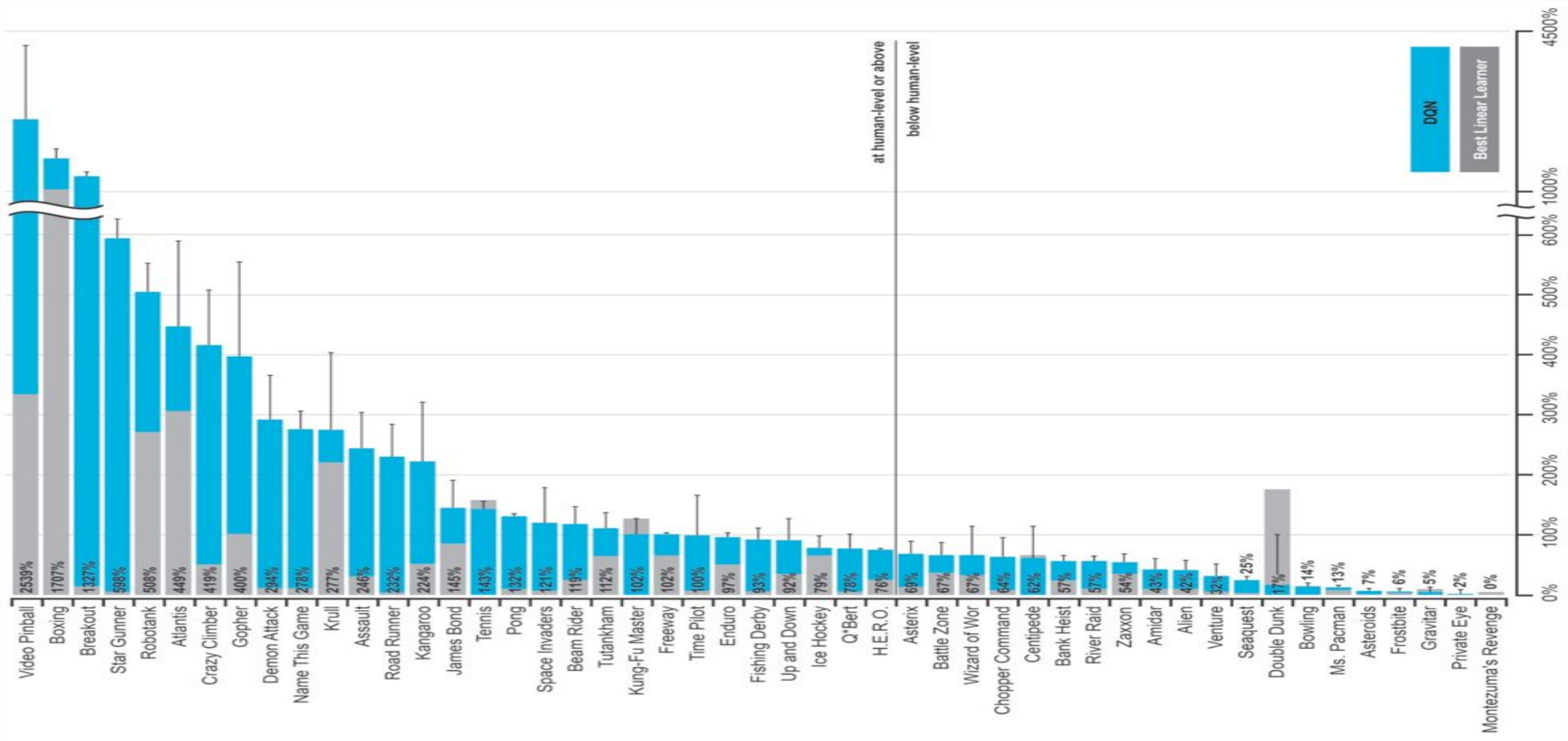
Generate **target** $r + \delta\gamma \max_{a'} Q(s', a'; \theta^{-})$

Take SGD step following $\theta_{t+1} \leftarrow \theta_t - \eta \frac{\partial L(\theta)}{\partial \theta_t}$

Update **target network** if t % k : $\theta^{-} \leftarrow \theta$

Store (s_t, a_t, r_t, s_{t+1}) in **replay memory**

DQN



DQN
Best Linear Learner
at human-level or above
below human-level
Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*Bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pacman
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge

DQN

Asynchronous RL (AsyncRL)

- DQN is very robust, but computationally expensive.
  - ~8 days on a single GPU
- Off-policy Q-Learning
  - We would like a robust system to experiment with both on-policy and off-policy methods
- Discrete action space
  - We want to be able to use the same method on continuous action space too.

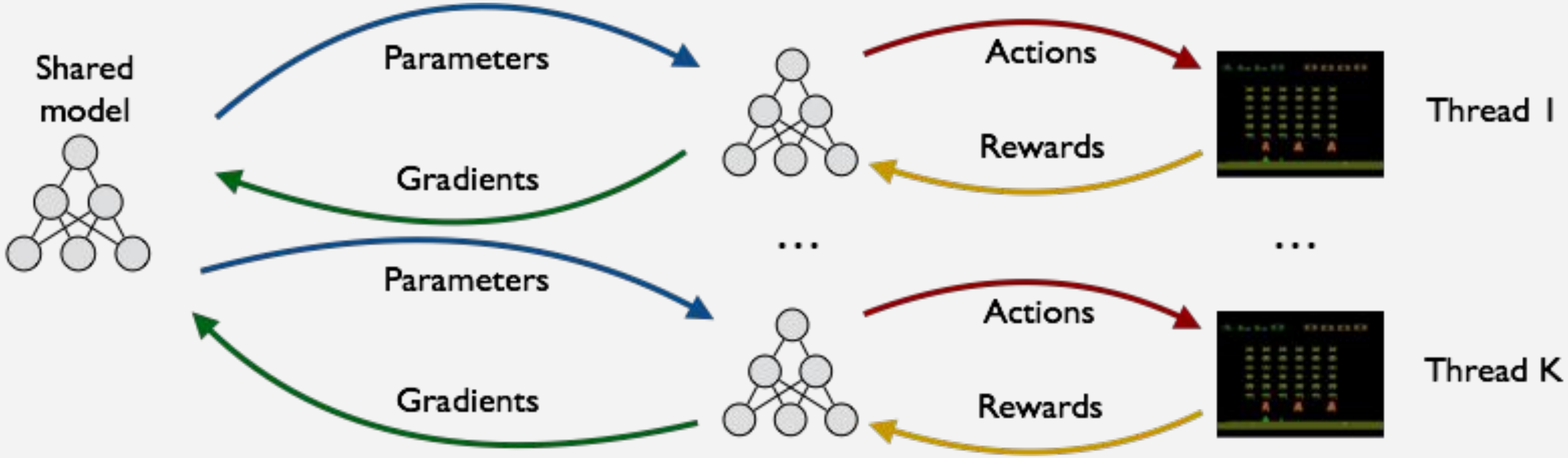
Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu

AsyncRL

- Asynchronous training of RL agents:
- Parallel actor-learners implemented using **CPU threads** and shared parameters.
- Online **Hogwild!**-style asynchronous updates (Recht et al., 2011, Lian et al., 2015).
- No replay? Parallel actor-learners have a similar stabilizing effect.
- Choice of RL algorithm
  - on or off-policy
  - value or policy-based.

AsyncRL


Shared model
Parameters
Gradients
Parameters
Gradients
Actions
Rewards
Actions
Rewards
...
...
Thread 1
Thread K

AsyncRL

- 1-Step Q-learning
  - Parallel actor-learners compute online 1-step update

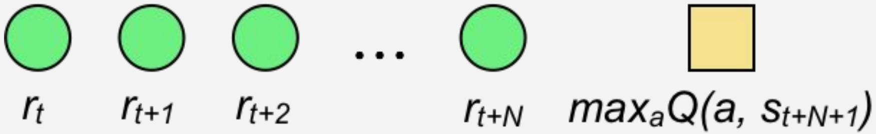
$$y \leftarrow r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

$$\Delta\theta \leftarrow \Delta\theta + \frac{\partial \left(y - Q(s, a; \theta)\right)^2}{\partial\theta}$$

  - Gradients accumulated over minibatch before update

AsyncRL

- n-Step Q-learning
  - Q-learning with a uniform mixture of backups of length 1 through N.



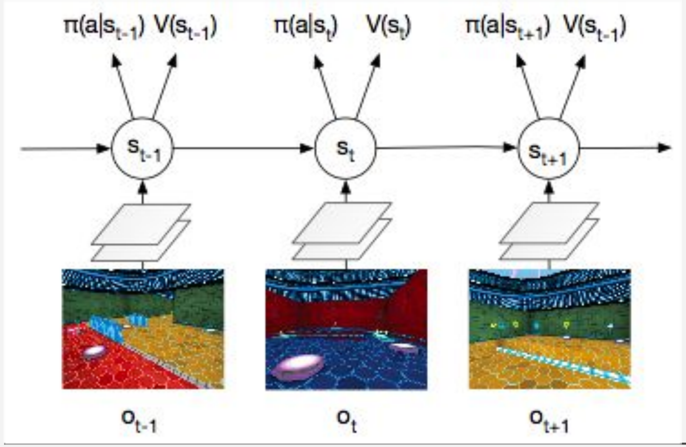
$$y \leftarrow \sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q(s_{t+N}, a'; \theta^-)$$

$$\Delta\theta \leftarrow \Delta\theta + \frac{\partial \left(y - Q(s_t, a_t; \theta)\right)^2}{\partial \theta}$$

  - Variation of “Incremental multi-step Q-learning” (Peng & Williams, 1995).

AsyncRL

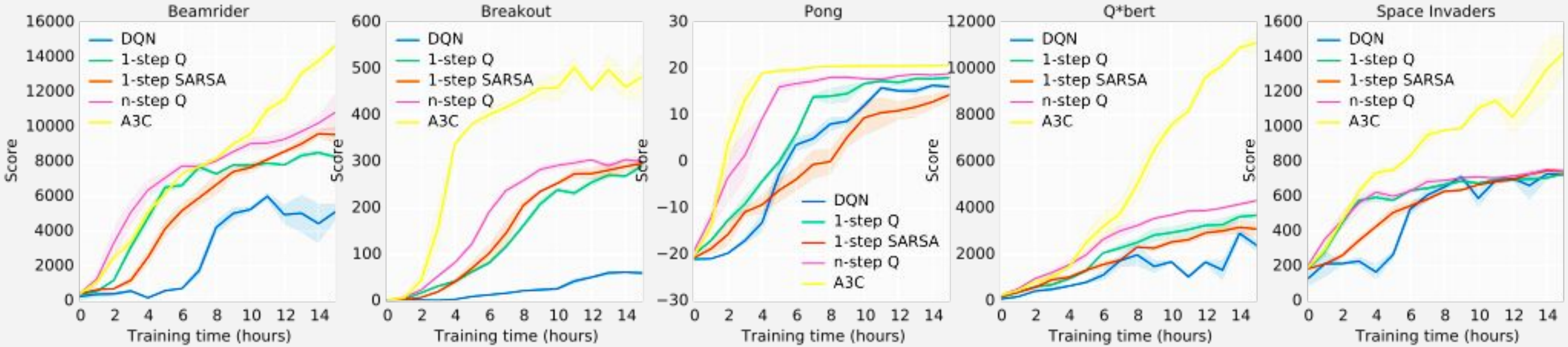
- Async Advantage Actor-Critic (**A3C**)
  - The agent learns a policy and a state value function.
  - Policy gradient multiplied by an estimate of the advantage. Similar to Generalized Advantage Estimation (Schulman et al, 2015).



$$\nabla_\theta \log \pi(a_t|s_t,\theta)\left(\sum_{k=0}^{N}\gamma^k r_{t+k} + \gamma^{N+1}V(s_{t+N+1}) - V(s_t)\right)$$

AsyncRL - Learning Speed

- New asynchronous methods trained on 16 CPU cores compared to DQN (blue) trained on a K40 GPU.
- n-step methods can be much faster than single step methods.
- Async advantage actor-critic tends to dominate the value-based methods.



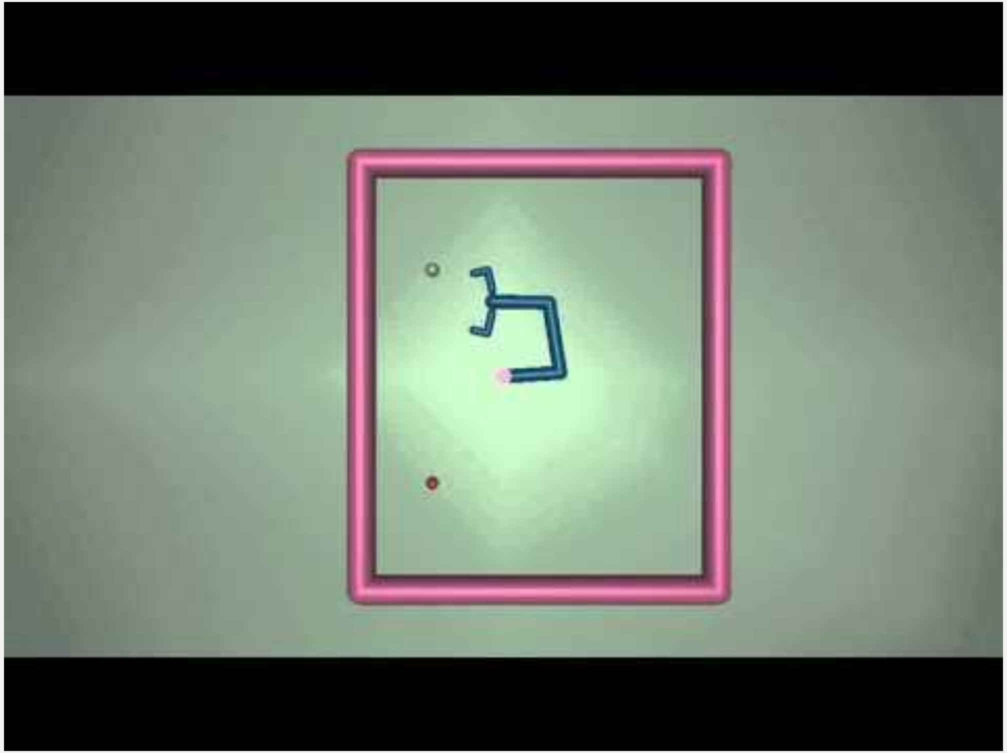
AsyncRL - ATARI Results

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorilla	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

AsyncRL

- Lightweight framework for asynchronous reinforcement learning.
  - Stable training with a variety of standard RL algorithms.
  - State-of-the-art results on a range of domains in hours on a single machine.
- Async advantage actor-critic excels on:
  - Both discrete and continuous actions
  - Feedforward and recurrent agents.
  - 2D and 3D games.
- Upcoming work - drastically improved data efficiency with an async off-policy actor-critic method.

AsyncRL

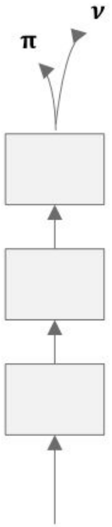


Progressive Neural Networks

Transfer in order to improve real-world sample efficiency

Progressive Neural Networks, arXiv, 2016

A. Rusu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, R. Hadsell



Google DeepMind

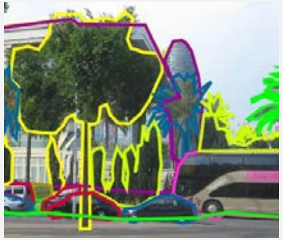
Why Progressive Nets?

Applying end-to-end deep learning to robotics is hard. Why?

1. Pixel-to-action robot data does not have this **form**
2. Pixel-to-action robot data does not have this **quantity**

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

LabelMe

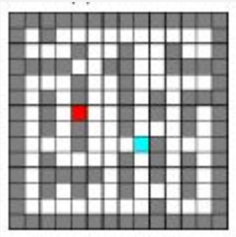


Can Deep RL Help?

Yes, however, (deep) RL is very data inefficient

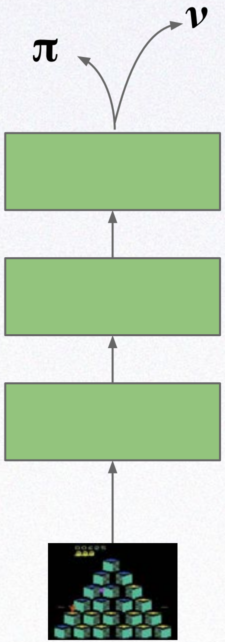
Continuous Deep Q-Learning with Model-based Acceleration.
Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, Sergey Levine. ICML 2016.

Asynchronous Methods for Deep Reinforcement Learning.
Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, ICML 2016

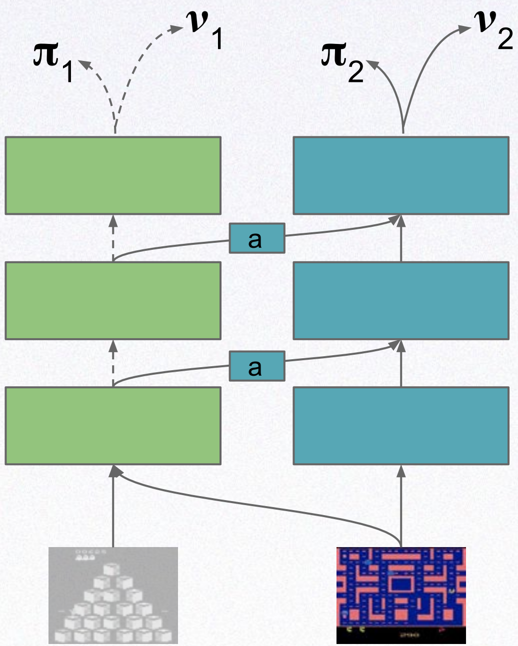


Control of Memory, Active Perception, and Action in Minecraft.
Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee
ICML 2016

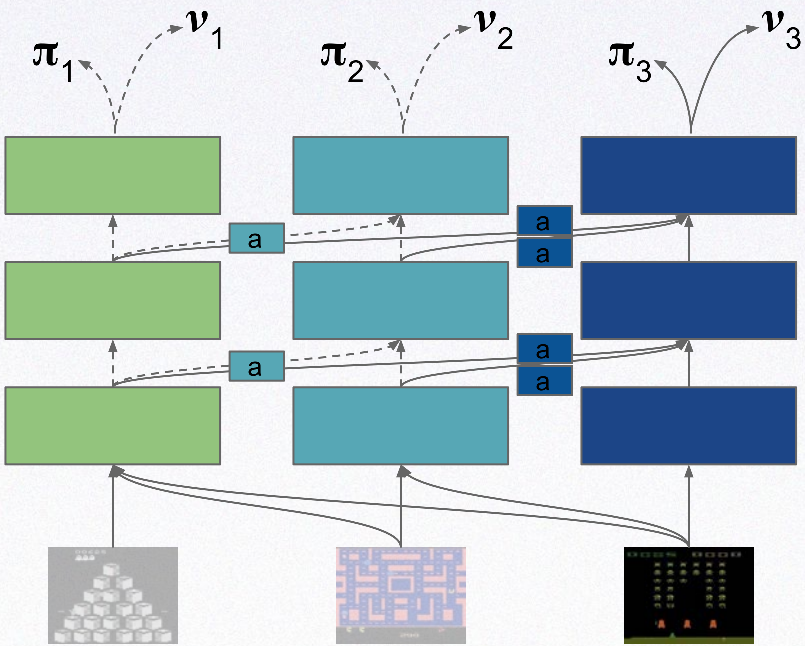
Progressive Neural Networks



Progressive Neural Networks

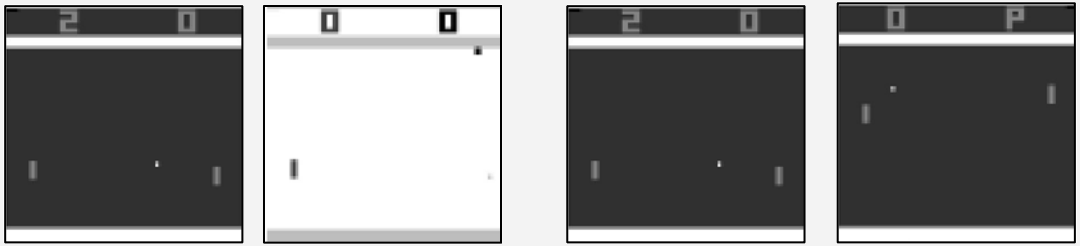


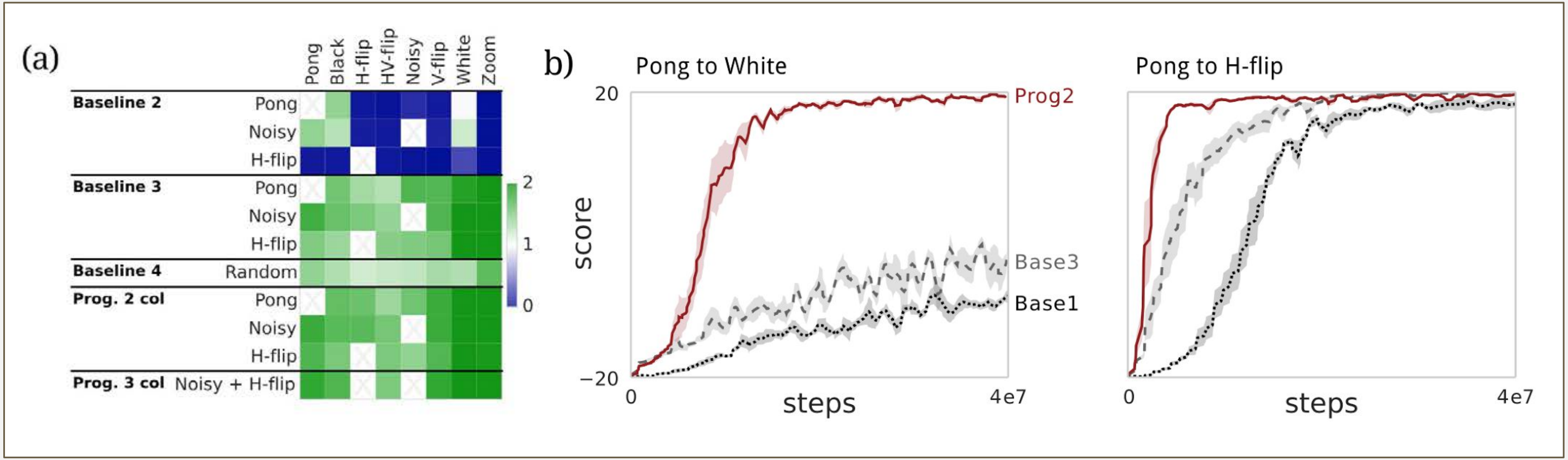
Progressive Neural Networks



Pong Soup

Pong → white Pong   Pong → horiz-flip Pong





arxiv.org/abs/1606.04671

Progressive Neural Networks

Advantages

1. No catastrophic forgetting of previous tasks - by design.
2. Deep, compositional feature transfer from all previous tasks and layers
3. Added capacity for learning task-specific features
4. Provides framework for analysis of transferred features

Disadvantages

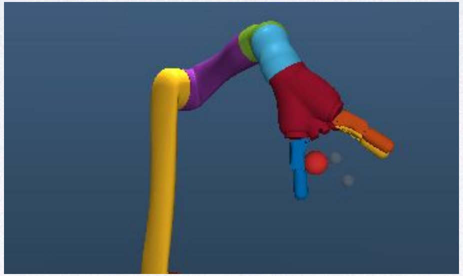
1. Requires knowledge of task boundaries
2. Scaling! Overall parameter growth is quadratic in the number of tasks (backward pass grows linearly).

Deep RL for Robotics

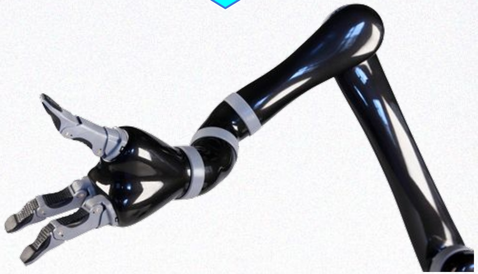
Simulation and Reality

Deep RL for Robotics

- **Deep reinforcement learning** has promise to revolutionise robotics
  - Learning human-level skills directly from raw sensor data
- However, there is a massive data problem.
  - State-of-the-art deep RL requires huge amounts of data in the form of interactive environments.
- *Progressive nets could be used to transfer learned policies from simulation to robot, even when using pixel inputs.*



Simulated Jaco arm

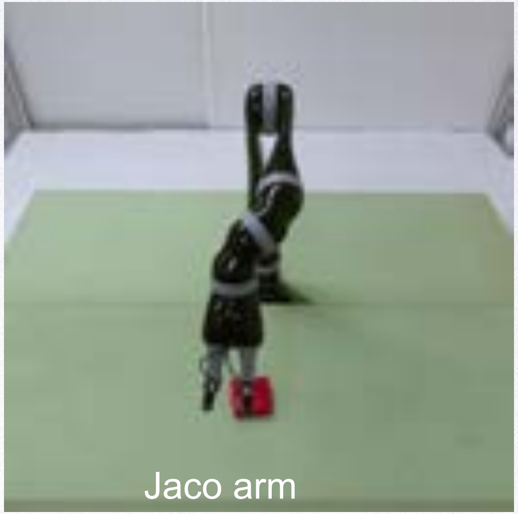


Real Jaco arm

Simulation vs. Reality

Deep learning and deep RL train very well from simulation:

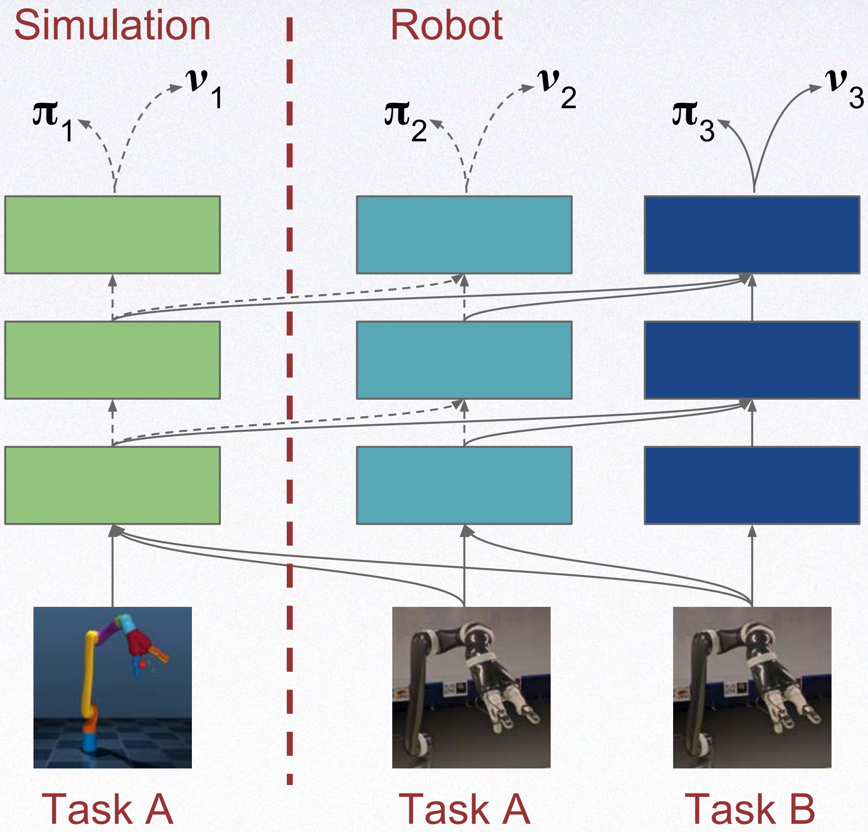
- Training: simulators run 24/7
- Algorithms: multi-threaded
- Hyperparameters: swept
- Speed: faster than real time

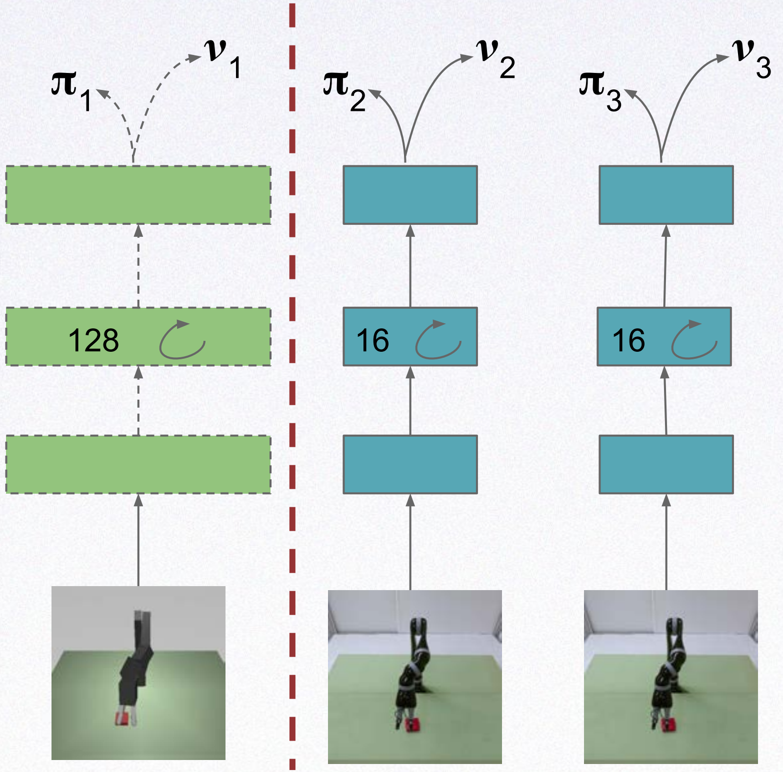


However, simulation is only valuable if whatever is learned can transfer to real robot domain.

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

arxiv.org/abs/1610.04286v1





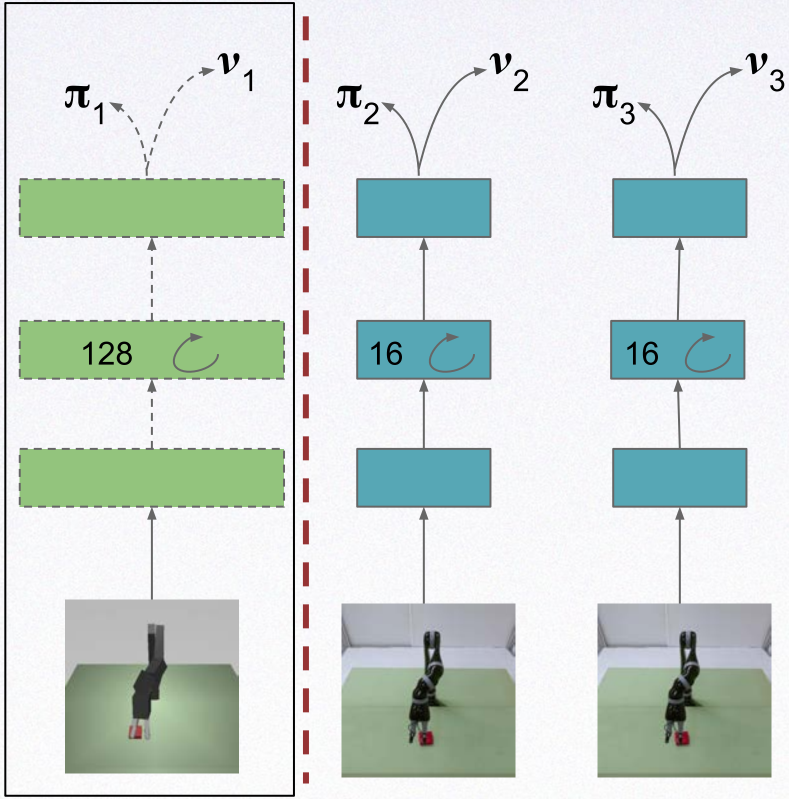
Column 1: Reacher task with random start, random target. Episodes have 50 steps; +1 reward when palm is within 10 cm of target's center.

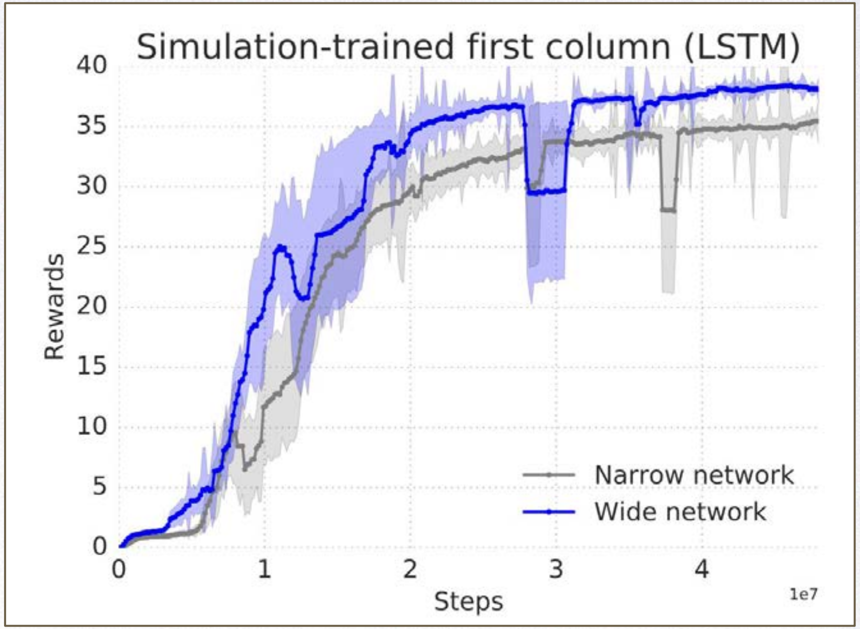
Input: RGB only

Output: joint velocities (9 DOF)

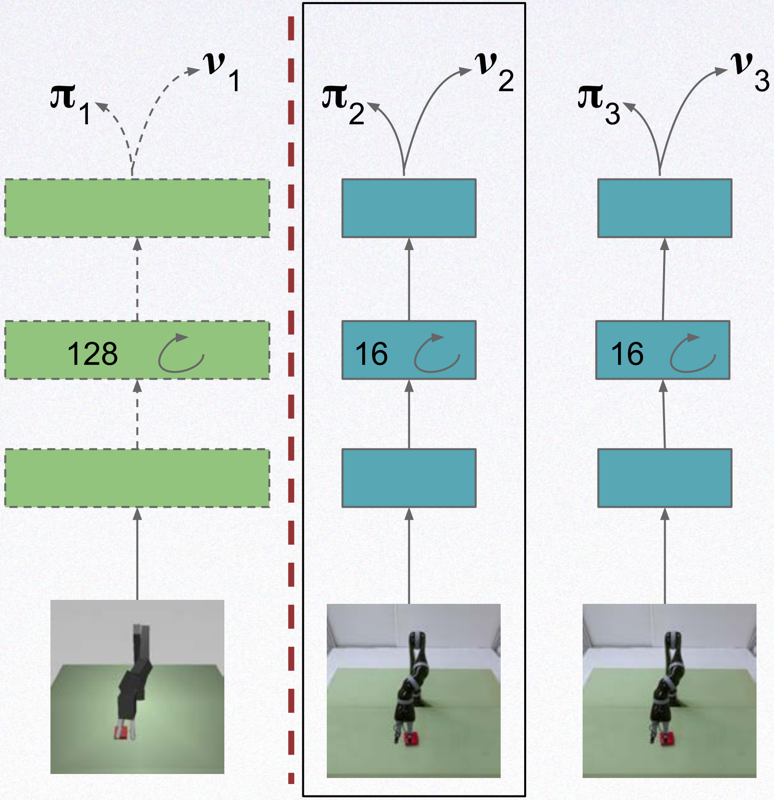
Network: ConvNet + LSTM + softmax output

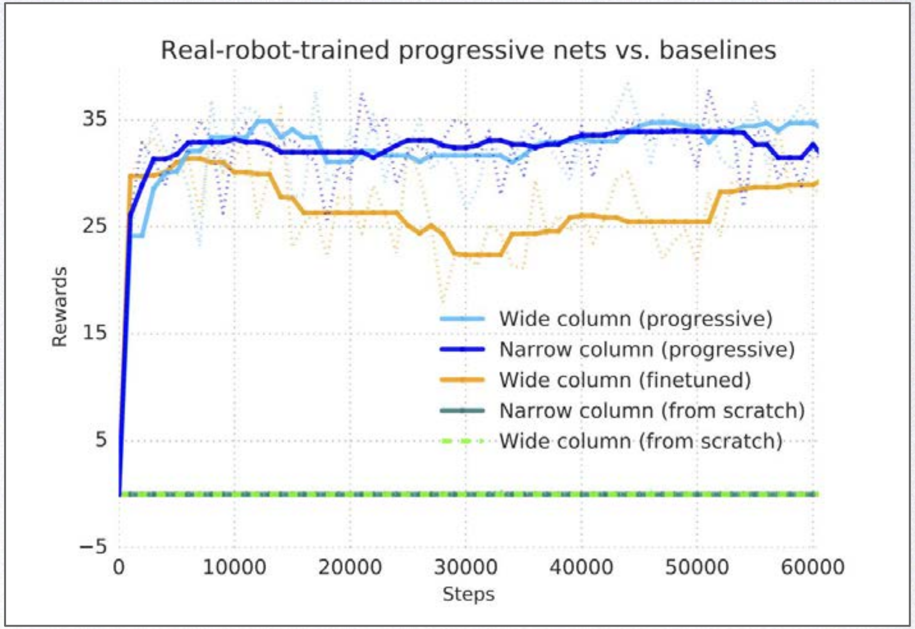
Learning: Asynchronous advantage actor-critic (A3C); 16 threads

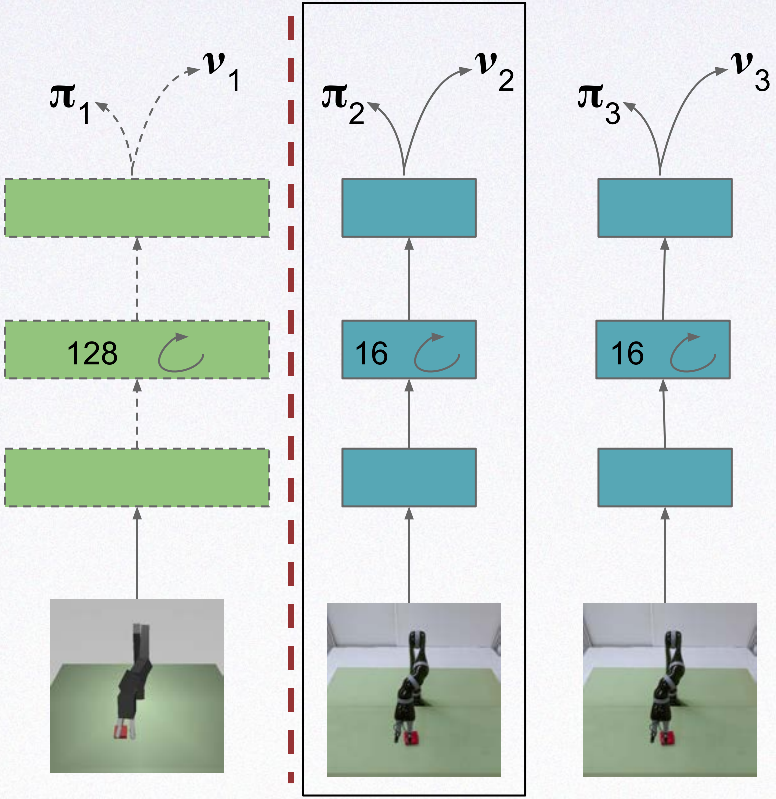




24 hrs of training ⇒ ~55 days real robot time

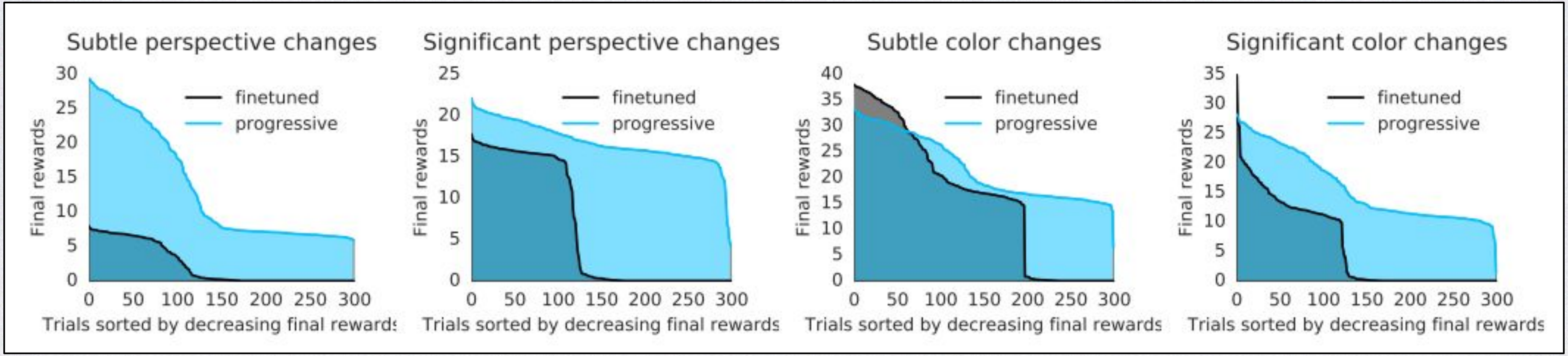

π1
v1
π2
v2
π3
v3
128
16
16

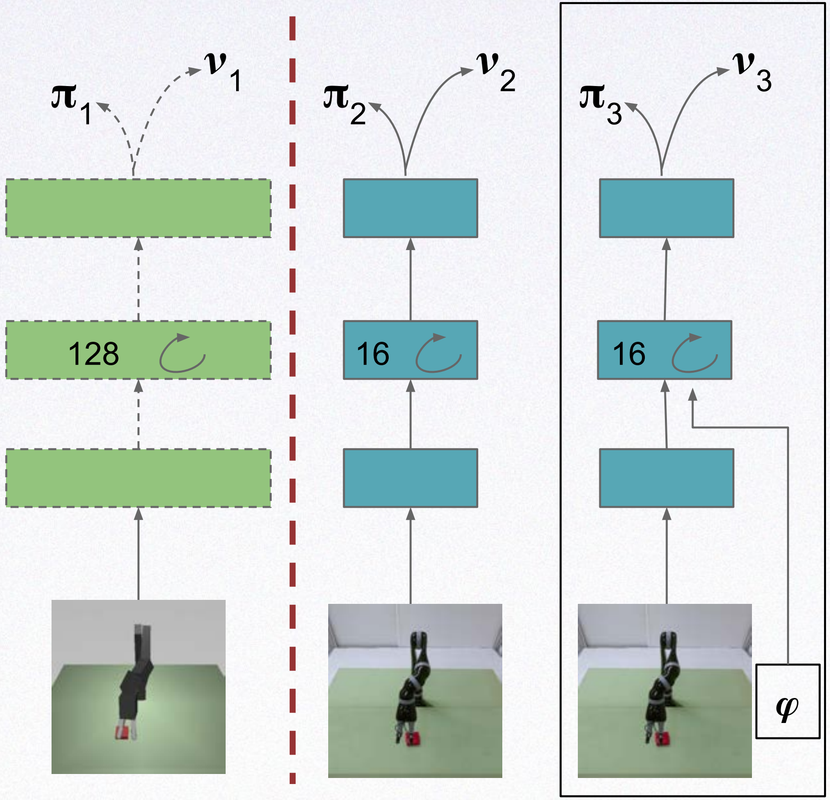

Real-robot-trained progressive nets vs. baselines
Rewards
Steps
35
25
15
5
−5
0
10000
20000
30000
40000
50000
60000
Wide column (progressive)
Narrow column (progressive)
Wide column (finetuned)
Narrow column (from scratch)
Wide column (from scratch)

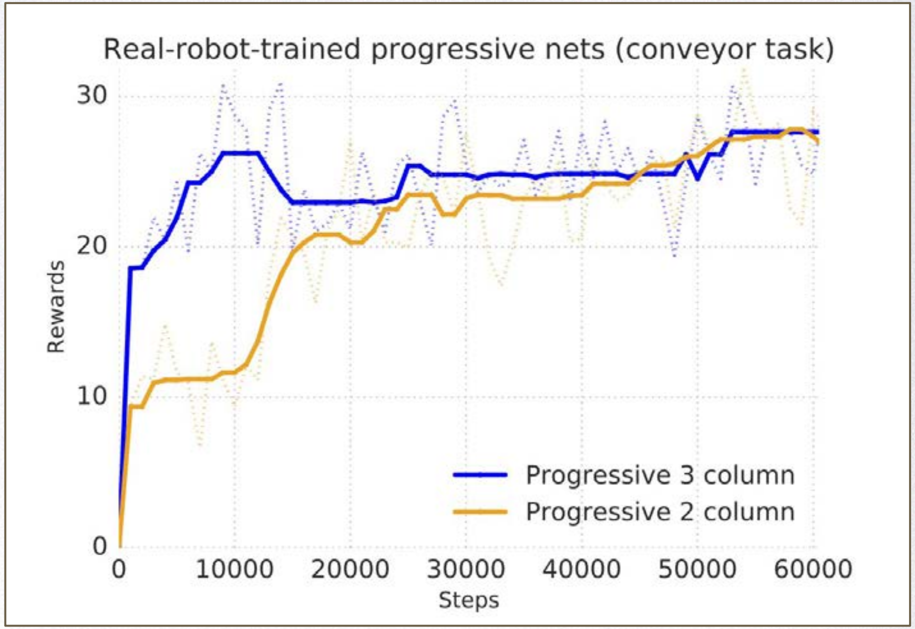


www.youtube.com/watch?v=dpShH7SrQsg

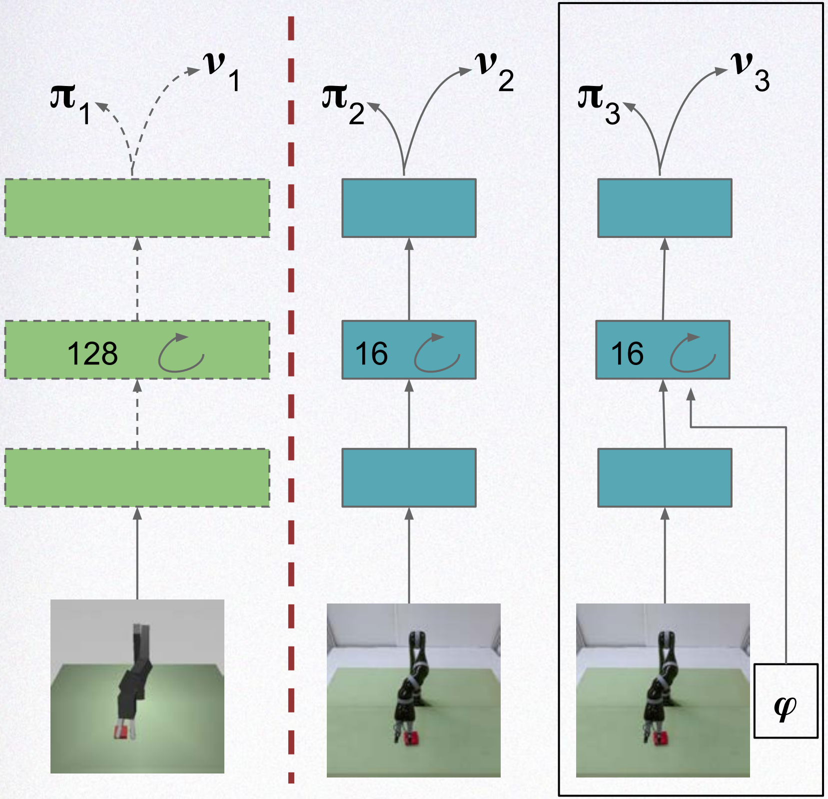
Finetuning or Progressive?

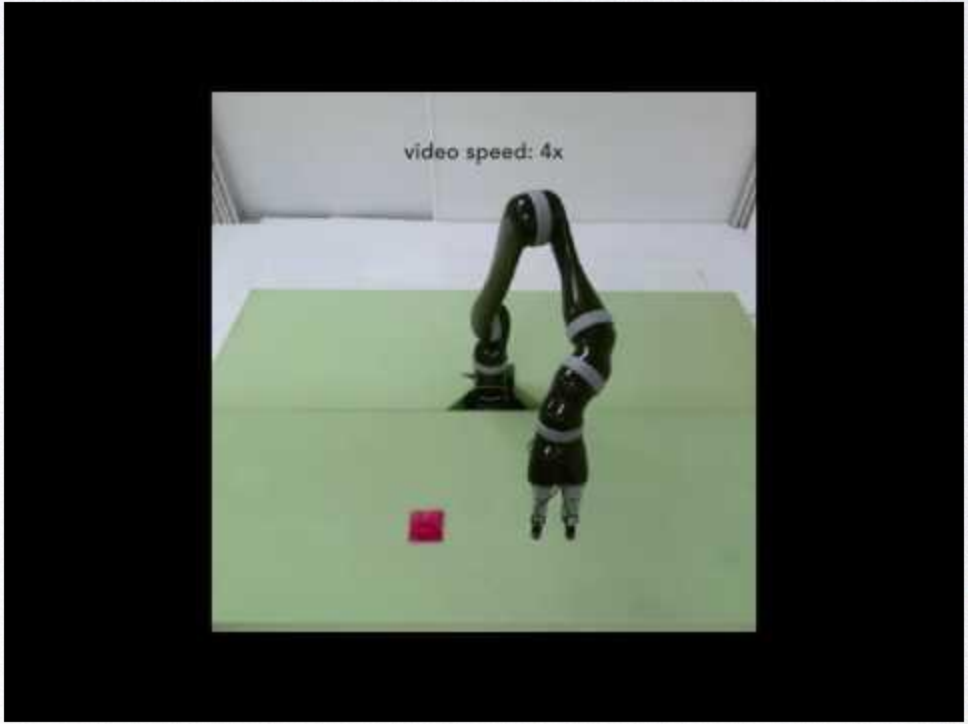






φ proprioception sensor data input to LSTM



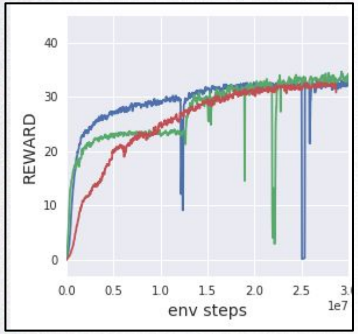


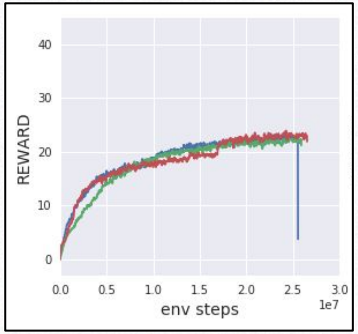
www.youtube.com/watch?v=e78J1K5LKCI

Matching shades of green is a bit of a pain…

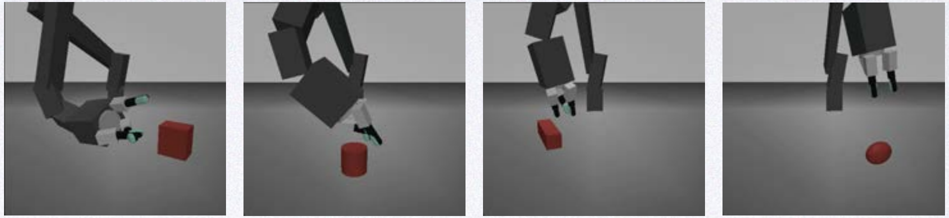
Tried and tested method for improving generalisation:

Data augmentation

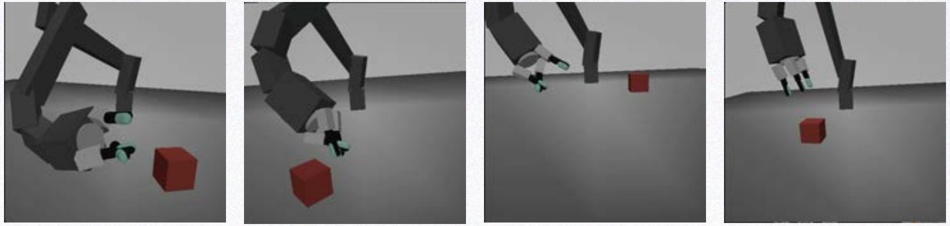




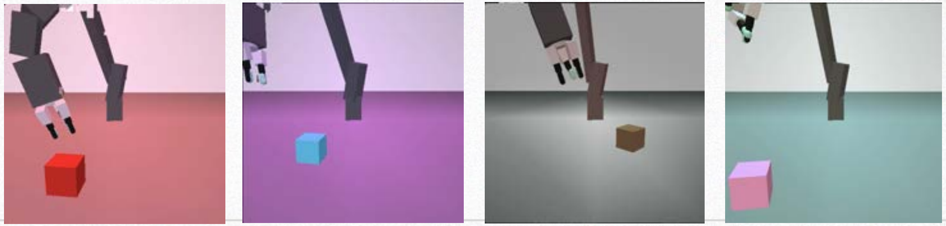
Target shape and size:



Camera position and angle:



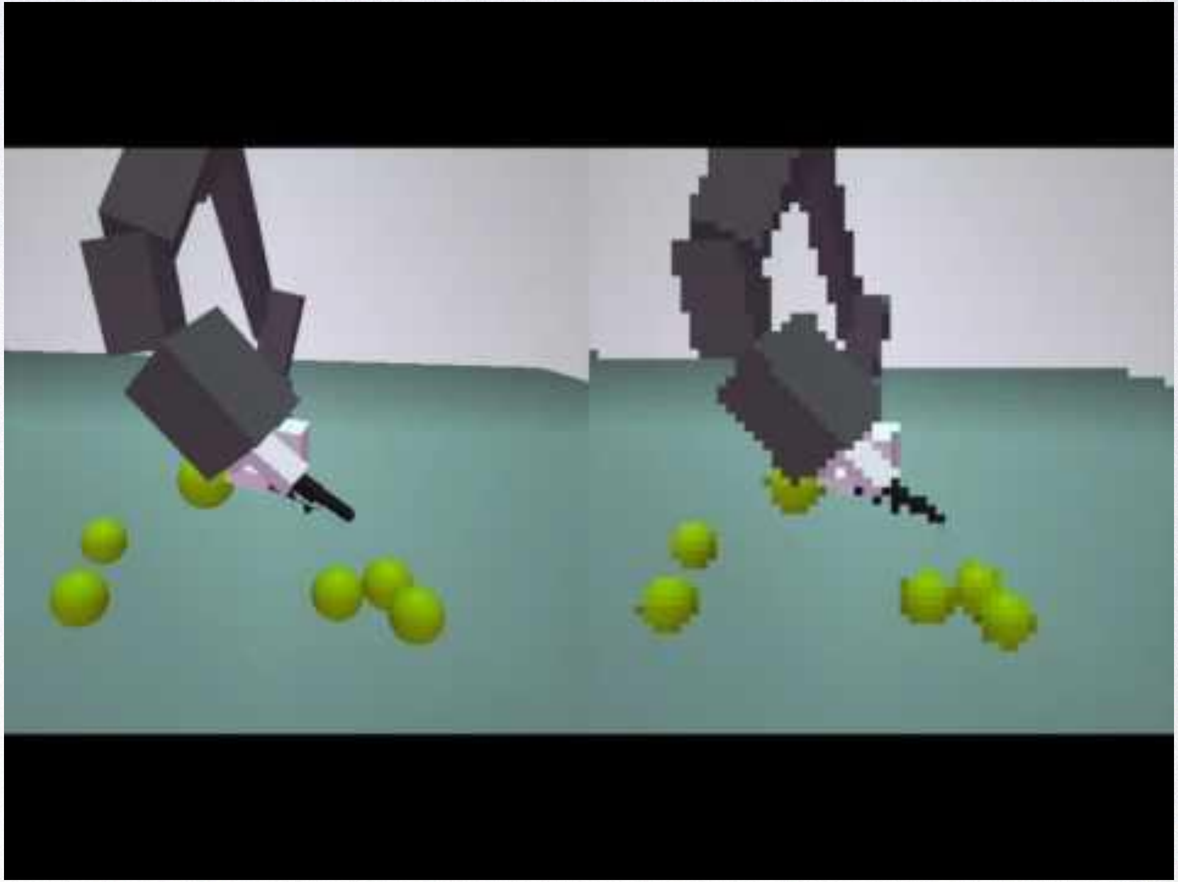
Color and lighting:



Targets are one of 4 geometric shapes with random sizes and randomly placed distractors.

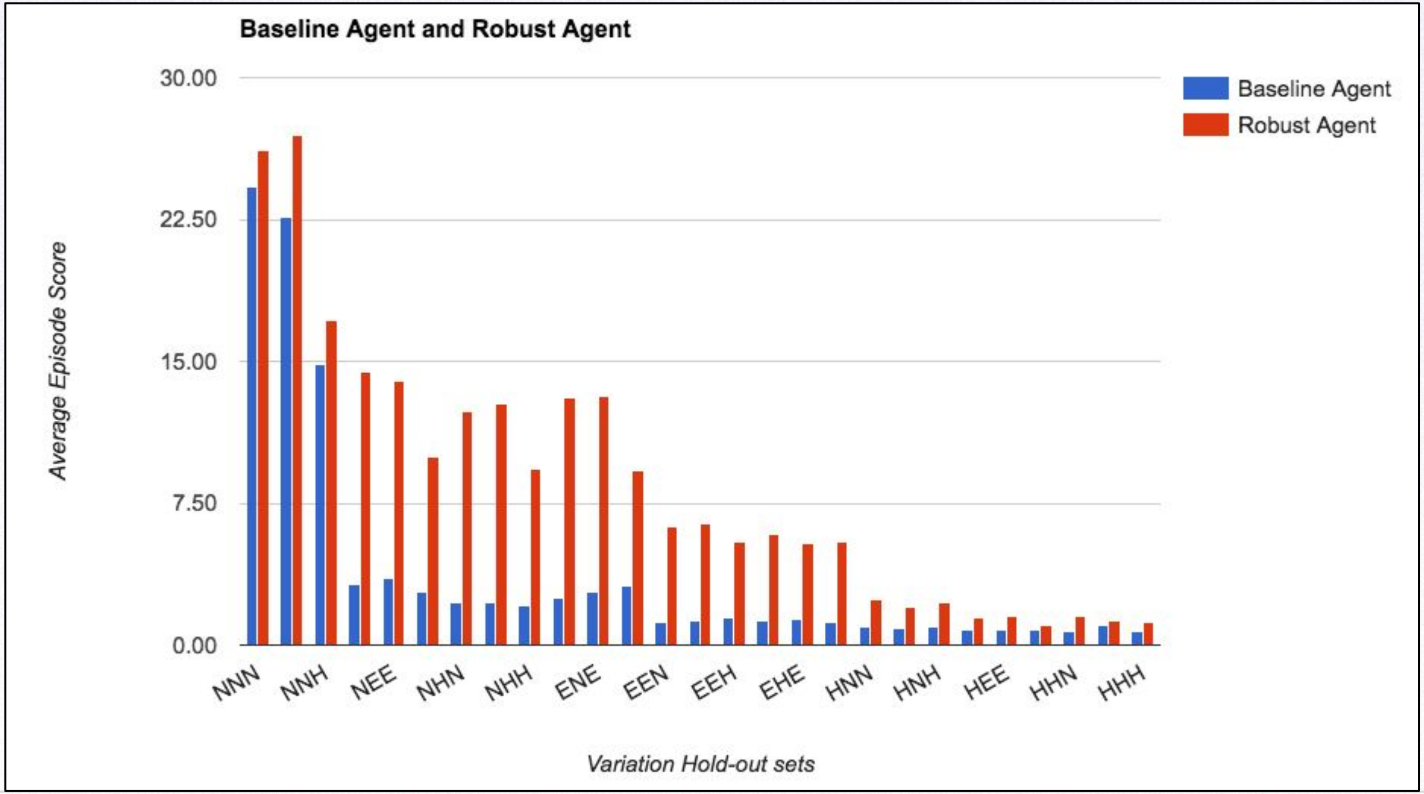
Camera position sampled with gaussian noise in every episode.

Target colors are picked uniformly at random. Random table colors. Random light source height and colors.

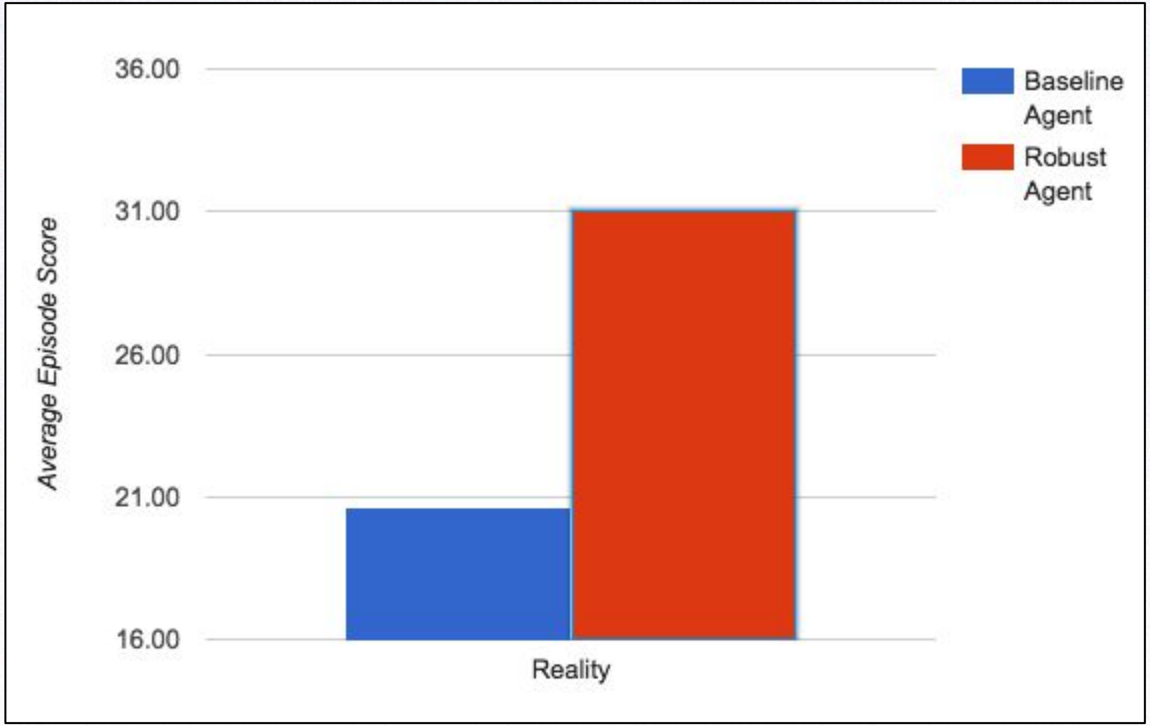


www.youtube.com/watch?v=6-Th424dvvk&feature=player_embedded

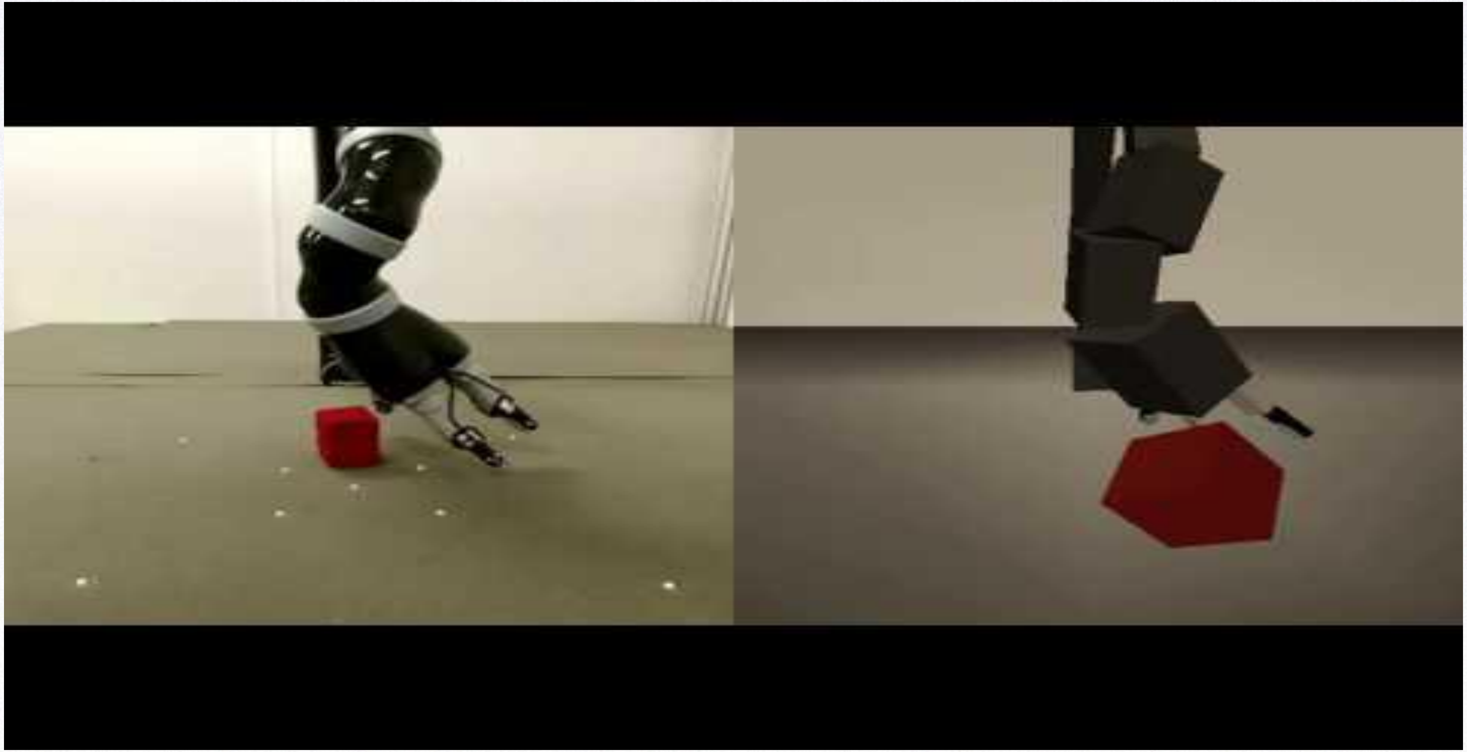
Robustly trained agents are better in novel environments in simulation…



… and, more importantly, in reality:



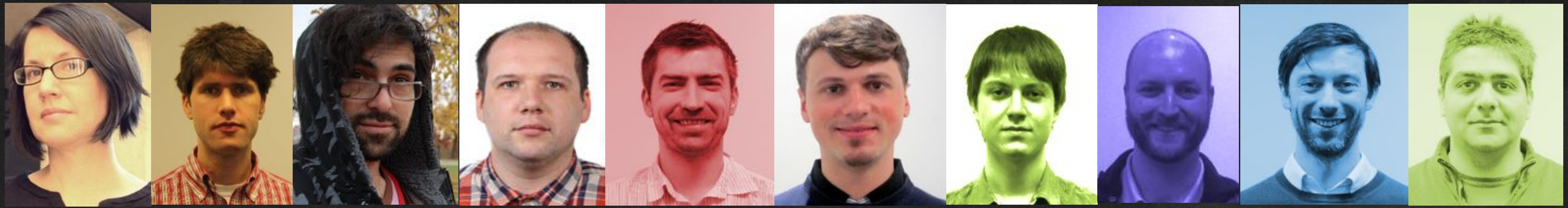
Zero-shot transfer from simulation to reality



Progressive Neural Networks
Sim-to-Real Robot Learning from Pixels

arxiv.org/abs/1606.04671
arxiv.org/abs/1610.04286v1

In collaboration with:



Thank you!