Progressive Nets for Simulation to Robot Transfer

Raia Hadsell

Google DeepMind
Skepticism

Let’s acknowledge a few difficulties with deep learning and robotics:

1. Robot-domain data does not present itself in this form:
Deep RL to the rescue?

**Continuous Deep Q-Learning with Model-based Acceleration.**

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

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

*However, deep RL is very data inefficient*
Skepticism

Let's acknowledge a few difficulties with deep learning and robotics:

2. Robot-domain data does not present itself in this quantity:
Simulation to the rescue?

https://www.youtube.com/watch?v=3WXd4vC3lbQ
Simulation to the rescue?

*Deep learning and deep RL likes simulators:*

- Training
- Algorithms
- Hyperparameters
- Speed

*However…*

There is a Reality Gap!

We aren’t interested in simulation unless learning can transfer to target domain, and transfer is hard, especially for deep learning.
Transfer + continual learning

- Continual + Transfer learning can bridge reality gap and ameliorate data inefficiency
- Unfortunately, neural networks are not well-suited to continual learning
  - Catastrophic forgetting from fine-tuning
  - Policy interference from multi-task learning
Progressive Neural Networks

In collaboration with:

Andrei Rusu  Neil C. Rabinowitz  Guillaume Desjardins  Hubert Soyer  James Kirkpatrick  Koray Kavukcuoglu  Razvan Pascanu

arxiv.org/abs/1606.04671
Progressive Neural Networks
Progressive Neural Networks
Progressive Neural Networks
Progressive Neural Networks
Progressive Neural Networks
Progressive Neural Networks
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
Progressive Neural Networks

Disadvantages

1. Requires knowledge of task boundaries
2. Quadratic parameter growth!

However, sensitivity analysis shows that successive columns use much less capacity.
Experimental setup

All training is with Asynchronous Advantage Actor-Critic (A3C) [mnih et al., 2016]

Progressive Net: column 1 trained on A, column 2 on task B

Baseline 1: column trained on task B

Baseline 2: column trained on A, top layer fine-tuned on B

Baseline 3: column trained on A, all layers fine-tuned on B

Baseline 4: column 1 random, column 2 trained on task B
Pong Soup

- Pong → white Pong
- Pong → horiz-flip Pong

(a) Diagram showing the results of different baseline and progressive scenarios, with a color-coded grid indicating the score for each condition.

(b) Graphs showing the performance of Pong to White and Pong to H-flip, with curves for baseline and progressive scenarios.
Analysis, 2 methods

1. Average **Perturbation** Sensitivity

   Inject Gaussian noise and measure drop in performance

---

**Pong to Noisy Pong**
Noise injected at column 1 (blue) or column 2 (green)
Analysis

2. **Average Fisher Sensitivity**

   - Compute modified diagonal Fisher matrix: network policy with respect to normalized activations of each layer
   - AFS is computed for layer $i$, column $k$, and feature $m$.

\[
\hat{F}_i^{(k)} = \mathbb{E}_{\rho(s,a)} \left[ \frac{\partial \log \pi}{\partial \hat{h}_i^{(k)}} \frac{\partial \log \pi}{\partial \hat{h}_i^{(k)}}^T \right] \quad \text{AFS}(i, k, m) = \frac{\hat{F}_i^{(k)}(m, m)}{\sum_k \hat{F}_i^{(k)}(m, m)}
\]
Pong Soup - Analysis

- **Insensitive**: pong → h-flip
- **Sensitive**: pong → zoom

Network layers:
- **fc**
- **conv 2**
- **conv 1**
Pong Soup - Analysis

pong -> noisy

fc
conv 2
conv 1

noisy -> pong

fc
conv 2
conv 1
Progressive nets from simulation to robot

Column 1: Reacher task with random start, fixed target, trained with Mujoco model of Jaco arm.

Input: RGB only

Output: joint velocities (6 DOF)

Network: ConvNet + LSTM + softmax output

Learning: Asynchronous advantage actor-critic (A3C); 16 threads
Progressive nets from simulation to robot
Progressive nets from simulation to robot

Reacher task: random start, fixed target
Input: RGB images
Output: joint velocities (6 DOF)
Progressive nets from simulation to robot

**Column 2:** Reacher task with random start, random target, trained with real Jaco arm.

**Input:** proprioception + target XYZ

**Output:** joint velocities (6 DOF)

**Network:** MLP + LSTM + softmax output

**Learning:** Asynchronous advantage actor-critic (A3C); 1 thread
Progressive nets from simulation to robot

https://www.youtube.com/watch?v=tXISbTOesMY
Progressive nets from simulation to robot
Progressive nets from simulation to robot

[Diagram showing two nets labeled $\pi_1$ and $\pi_2$, with nodes labeled 128, 16, and connecting to robot images.]

https://www.youtube.com/watch?v=YZz5lo_iPi8
Progressive nets from simulation to robot

**Column 3:** ‘Catch’, trained with real Jaco arm.

https://www.youtube.com/watch?v=qzMTPzbPV0c
Progressive nets from simulation to robot

Column 4: ‘Catch the bee’, trained with real Jaco arm.

https://www.youtube.com/watch?v=JkXhIIWsUA0
What’s next?

- Scaling up Progressive Networks
  - Compression / Brain Damage / Complementary Learning
  - Limiting Model Growth with Sharing of Lateral Connections

- Automating the progression
  - Eliminating the need for manual switch points while keeping model growth in check

- Meta-controller making use old policies in new situations
  - Fast adaptation to new tasks using the fact that old policies are NOT forgotten.

Thank you