Transfer and Multi-Task Learning

CS 285

Instructor: Sergey Levine
UC Berkeley
What’s the problem?

this is easy (mostly)

this is impossible

Why?
Montezuma’s revenge

- Getting key = reward
- Opening door = reward
- Getting killed by skull = bad
Montezuma’s revenge

- We know what to do because we understand what these sprites mean!
- Key: we know it opens doors!
- Ladders: we know we can climb them!
- Skull: we don’t know what it does, but we know it can’t be good!

- Prior understanding of problem structure can help us solve complex tasks quickly!
Can RL use the same prior knowledge as us?

- If we’ve solved prior tasks, we might acquire useful knowledge for solving a new task

- How is the knowledge stored?
  - Q-function: tells us which actions or states are good
  - Policy: tells us which actions are potentially useful
    - some actions are never useful!
  - Models: what are the laws of physics that govern the world?
  - Features/hidden states: provide us with a good representation
    - Don’t underestimate this!
Aside: the representation bottleneck

To decouple reinforcement learning from representation learning, we decapitate an agent by destroying its policy and value outputs and then re-train end-to-end. The representation remains and the policy is swiftly recovered. The gap between initial optimization and recovery shows a representation learning bottleneck.

slide adapted from E. Schelhamer, “Loss is its own reward”
**transfer learning**: using experience from one set of tasks for faster learning and better performance on a new task

**in RL, task = MDP!**

source domain | target domain

*“shot”*: number of attempts in the target domain

0-shot: just run a policy trained in the source domain

1-shot: try the task once

few shot: try the task a few times
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) Transferring visual representations & domain adaptation
   b) Domain adaptation in reinforcement learning
   c) Randomization

2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Sharing representations and layers across tasks in multi-task learning
   b) Contextual policies
   c) Optimization challenges for multi-task learning
   d) Algorithms

3. Transferring models and value functions
   a) Model-based RL as a mechanism for transfer
   b) Successor features & representations
Forward Transfer
Pretraining + Finetuning

The most popular transfer learning method in (supervised) deep learning!
What issues are we likely to face?

- **Domain shift**: representations learned in the source domain might not work well in the target domain.

- **Difference in the MDP**: some things that are possible to do in the source domain are not possible to do in the target domain.

- **Finetuning issues**: if pretraining & finetuning, the finetuning process may still need to explore, but optimal policy during finetuning may be deterministic!
Domain adaptation in computer vision

Invariance assumption: everything that is different between domains is irrelevant

Formally:

\[ p(x) \text{ is different} \quad \exists z = f(x) \text{ such that } p(y|z) = p(y|x), \text{ but } p(z) \text{ is same} \]
How do we apply this idea in RL?

Tzeng*, Devin*, et al., “Adapting Visuomotor Representations with Weak Pairwise Constraints”
Domain adaptation in RL for dynamics?

Why is invariance not enough when the dynamics don’t match?

\[ r(s, a) = r(s, a) + \Delta r(s, a) \]

\[ \Delta r(s_t, a_t, s_{t+1}) = \log p_{\text{target}}(s_{t+1} | s_t, a_t) - \log p_{\text{source}}(s_{t+1} | s_t, a_t) \]

\[ \Delta r(s_t, a_t, s_{t+1}) = \log p(\text{target} | s_t, a_t, s_{t+1}) - \log p(\text{target} | s_t, a_t) - \log p(\text{source} | s_t, a_t, s_{t+1}) + \log p(\text{source} | s_t, a_t) \]

When might this not work?

Eysenbach et al., “Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers”
What if we can also finetune?

1. RL tasks are generally much less diverse
   • Features are less general
   • Policies & value functions become overly specialized
2. Optimal policies in fully observed MDPs are deterministic
   • Loss of exploration at convergence
   • Low-entropy policies adapt very slowly to new settings
Finetuning with maximum-entropy policies

How can we increase diversity and entropy?

\[ \pi(a|s) = \exp(Q_\phi(s, a) - V(s)) \text{ optimizes } \sum_t E_{\pi(s_t, a_t)} [r(s_t, a_t)] + E_{\pi(s_t)} [\mathcal{H}(\pi(a_t|s_t))] \]

Act as randomly as possible while collecting high rewards!
Example: pre-training for robustness

Learning to solve a task in all possible ways provides for more robust transfer!
Example: pre-training for diversity

Pretraining: reward = speed (any direction)
(one robot per trajectory)

DDPG (policy 1)
25 random seeds; noise added to actions

Soft Q-learning (fixed policy)
random seeds 0 - 24

epoch: 1

random initialization
pretrained with DDPG
pretrained with soft Q-learning

Domain adaptation: suggested readings


...and many many others!
Finetuning: suggested readings


Kumar et al. *One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL.* 2020

...and many many others!
Forward Transfer with Randomization
What if we can manipulate the source domain?

• So far: source domain (e.g., empty room) and target domain (e.g., corridor) are fixed

• What if we can design the source domain, and we have a difficult target domain?
  • Often the case for simulation to real world transfer
EPOpt: randomizing physical parameters

train

unmodeled effects

ensemble adaptation

adapt

Rajeswaran et al., “EPOpt: Learning robust neural network policies...”
Preparing for the unknown: explicit system ID

Yu et al., “Preparing for the Unknown: Learning a Universal Policy with Online System Identification”
Another example
CAD2RL: randomization for real-world control

also called domain randomization

Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”
CAD2RL: randomization for real-world control

Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”
Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”
Randomization for manipulation

Tobin, Fong, Ray, Schneider, Zaremba, Abbeel

James, Davison, Johns
Source domain randomization and domain adaptation suggested readings


Yu et al. (2017). **Preparing for the Unknown: Learning a Universal Policy with Online System Identification.**


Tobin et al. (2017). **Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.**

James et al. (2017). **Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task.**

Methods that **also** incorporate domain adaptation together with randomization:

Bousmalis et al. (2017). **Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping.**

Rao et al. (2017). **RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real.**

... and many many others!
Multi-Task Transfer
Can we learn **faster** by learning multiple tasks?

Multi-task learning can:
- Accelerate learning of all tasks that are learned together
- Provide better pre-training for down-stream tasks
Can we solve multiple tasks at once?

Multi-task RL corresponds to single-task RL in a **joint MDP**

![Diagram showing the process of picking an MDP randomly in the first state and then sampling states with actions and rewards through the MDPs.](image)
What is difficult about this?

- **Gradient interference**: becoming better on one task can make you worse on another

- **Winner-take-all problem**: imagine one task starts getting good – algorithm is likely to prioritize that task (to increase average expected reward) at the expense of others

- In practice, this kind of multi-task RL is very challenging
Actor-mimic and policy distillation

Goal: learn a single policy that can play all Atari games

**Policy Distillation**

Andrei A. Rusu, Sergio Gómez Colmenarejo, Çağlar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu &Raia Hadsel

Google DeepMind

**Actor-Mimic**

**Deep Multitask and Transfer Reinforcement Learning**

Emilio Parisotto, Jimmy Ba, Ruslan Salakhutdinov

Department of Computer Science

University of Toronto
Distillation for Multi-Task Transfer

\[
L = \sum_a \pi_{E_i}(a|s) \log \pi_{AMN}(a|s)
\]

(just supervised learning/distillation)

analogous to guided policy search, but for transfer learning

-> see model-based RL slides

some other details
(e.g., feature regression objective)

– see paper

Combining weak policies into a strong policy

Divide and conquer reinforcement learning algorithm sketch:

1. optimize each local policy $\pi_{\theta_i}(a_t|s_t)$ on initial state $s_{0,i}$ w.r.t. $\tilde{r}_{k,i}(s_t, a_t)$
2. use samples from step (1) to train $\pi(\mathbf{u}_t|\mathbf{x}_t)$ to mimic each $\pi_{\theta_i}(\mathbf{u}_t|\mathbf{x}_t)$
3. update reward function $\tilde{r}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi(\mathbf{u}_t|\mathbf{x}_t)$

For details, see: “Divide and Conquer Reinforcement Learning”
Distillation Transfer Results

How does the model know what to do?

• So far: what to do is apparent from the input (e.g., which game is being played)
• What if the policy can do *multiple* things in the *same* environment?
Contextual policies

standard policy: $\pi_{\theta}(a|s)$

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

\text{e.g., do dishes or laundry}

formally, simply defines augmented state space: \[ \tilde{s} = \begin{bmatrix} s \\ \omega \end{bmatrix} \]

\[ \tilde{S} = S \times \Omega \]

$\omega$: stack location

$\omega$: walking direction

$\omega$: where to hit puck

images: Peng, van de Panne, Peters
Contextual policies

standard policy: \( \pi_\theta(a|s) \)

contextual policy: \( \pi_\theta(a|s, \omega) \)

will discuss more in the context of meta-learning!

\( \omega \): stack location  
\( \omega \): walking direction  
\( \omega \): where to hit puck
Transferring Models and Value Functions
The problem setting

Assumption: the dynamics $p(s_{t+1}|s_t, a_t)$ is the same in both domains but the reward function is different

Common setting:
• Autonomous car learns how to drive to a few destinations, and then has to navigate to a new one
• A kitchen robot learns to cook many different recipes, and then has to cook a new one in the same kitchen
What is the best object to transfer?

**Model:** very simple to transfer, since the model is already (in principle) independent of the reward

**Value function:** not straightforward to transfer by itself, since the value function entangles the dynamics and reward, but possible with a decomposition
  - what kind of “dynamics relevant” information does a value function contain?

**Policy:** possible to do with contextual policies, but otherwise tricky, because the policy contains the *least* dynamics information
Transferring models

\[ \hat{p}(s_{t+1}|s_t, a_t) \]

source domain

\[ \hat{p}(s_{t+1}|s_t, a_t) \]

target domain

why might zero-shot transfer not always work?
Transferring value functions

**Not so fast!** Value functions couple **dynamics, rewards, and policies**!

\[ Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a), a' \sim \pi(a'|s')} [Q^\pi(s', a')] \]

Is this really such a good idea? **Yes, because of linearity**

**Key observation:** the value function is linear in the reward function

- let \( \mathbf{Pv} \) denote a vector \( \mathbf{w} \) of length \(|S||A|\) given by \( \mathbf{w}(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)}[\mathbf{v}(s')] \)
- let \( \mathbf{P}^\pi \mathbf{v} \) denote a vector \( \mathbf{w} \) of length \(|S||A|\) given by \( \mathbf{w}(s, a) = \mathbb{E}_{s' \sim p(s'|s, a), a' \sim \pi(a'|s')}[\mathbf{v}(s', a')] \)

\[ Q^\pi = r + \gamma \mathbf{P}^\pi Q^\pi \quad Q^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1}r \]

vectors with \(|S||A|\) entries
Successor representations & successor features

\[ Q^\pi = (I - \gamma P^\pi)^{-1}r \]

let \( \phi \) be a \( |S||A| \times N \) feature matrix

let \( \psi \) be a \( |S||A| \times N \) matrix such that \( \psi = (I - P^\pi)^{-1} \phi \)

if \( r = \phi w \), then \( Q^\pi = \psi w \)

**Proof:**

\[ Q^\pi = (I - \gamma P^\pi)^{-1}r \]

\[ Q^\pi = (I - \gamma P^\pi)^{-1} \phi w \]

\[ Q^\pi = \psi w \]

\( \psi_i \) is a “successor feature” for \( \phi_i \)
Successor representations & successor features

let $\phi$ be a $|S||A| \times N$ feature matrix

let $\psi$ be a $|S||A| \times N$ matrix such that $\psi = (I - P^\pi)^{-1} \phi$

if $r = \phi w$, then $Q^\pi = \psi w$

For any new reward function, if we can fit $r \approx \phi w$, we get $Q^\pi \approx \psi w$

**Important**: this holds for $Q^\pi$, not $Q^*$! why?

$$Q^*(s, a) = r(s, a) + \gamma E_{s' \sim p(s'|s, a)} \left[ \max_{a'} Q^\pi(s', a') \right]$$

this is no longer linear!
Aside: successor representations

let \( \phi \) be a \(|S| \times |A| \times N\) feature matrix

let \( \psi \) be a \(|S| \times |A| \times N\) matrix such that \( \psi = (I - P^\pi)^{-1}\phi \)

if \( r = \phi w \), then \( Q^\pi = \psi w \)

what if \( \phi = I \)? for each \((s, a)\), there is a \( \phi_{s,a} = \delta(s, a) \)

then we can show that \( \psi_{s',a'}(s, a) \) predicts probability of landing in \((s', a')\) from \((s, a)\) under discount \( \gamma \)

\[
\begin{align*}
S_1 & \rightarrow S_2 & S_3 \\
\gamma = 0.9 & \quad \psi_{s_3}(s_1) = 0.9^2
\end{align*}
\]

Transfer with successor features

**Simplest use**: evaluation

1. get small amount of data \( (s_i, a_i, r_i, s'_i) \) in new MDP
2. fit \( w \) such that \( \phi(s_i, a_i)w \approx r_i \) (linear regression)
3. initialize \( Q^\pi(s, a) = \psi(s, a)w \)
4. finetune \( \pi \) and \( Q^\pi \) with any RL method

**More sophisticated use**: train multiple \( \psi^{\pi_i} \) functions for different \( \pi_i \)

choose initial policy \( \pi(s) = \arg \max_a \max_i \psi^{\pi_i}(s, a)w \)

this provides a *better* initial policy in general

For more details, see: Barreto et al., Successor Features for Transfer in Reinforcement Learning
Recap

No single solution! Survey of various recent research papers

1. Forward transfer: train on one task, transfer to a new task
   a) Transferring visual representations & domain adaptation
   b) Domain adaptation in reinforcement learning
   c) Randomization

2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Sharing representations and layers across tasks in multi-task learning
   b) Contextual policies
   c) Optimization challenges for multi-task learning
   d) Algorithms

3. Transferring models and value functions
   a) Model-based RL as a mechanism for transfer
   b) Successor features & representations