Advanced Imitation Learning
Challenges and Open Problems

CS 294-112: Deep Reinforcement Learning
Sergey Levine
Imitation Learning
Reinforcement Learning
Imitation vs. Reinforcement Learning

**Imitation learning**
- Requires demonstrations
- Must address distributional shift
- Simple, stable supervised learning
- Only as good as the demo

**Reinforcement learning**
- Requires reward function
- Must address exploration
- Potentially non-convergent RL
- Can become arbitrarily good

Can we get the best of both?

E.g., what if we have demonstrations *and* rewards?
Addressing distributional shift with RL?

- Initial policy $\pi$
- Generate policy samples from $\pi$
- Human demonstrations
- Update reward using samples & demos
- Policy $\pi$
- Reward $r$
Addressing distributional shift with RL?

IRL *already* addresses distributional shift via RL

this part is regular “forward” RL

But it doesn’t use a known reward function!
Simplest combination: pretrain & finetune

- Demonstrations can overcome exploration: show us how to do the task
- Reinforcement learning can improve beyond performance of the demonstrator
- Idea: initialize with imitation learning, then finetune with reinforcement learning!

1. collected demonstration data \((s_i, a_i)\)
2. initialize \(\pi_\theta\) as \(\max_\theta \sum_i \log \pi_\theta(a_i|s_i)\)
3. run \(\pi_\theta\) to collect experience
4. improve \(\pi_\theta\) with any RL algorithm
Simplest combination: pretrain & finetune
Simplest combination: pretrain & finetune

Pretrain & finetune

1. collected demonstration data \((s_i, a_i)\)
2. initialize \(\pi_\theta\) as \(\max_\theta \sum_i \log \pi_\theta(a_i | s_i)\)
3. run \(\pi_\theta\) to collect experience
4. improve \(\pi_\theta\) with any RL algorithm

vs. DAgger

1. train \(\pi_\theta(a_t | o_t)\) from human data \(\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}\)
2. run \(\pi_\theta(a_t | o_t)\) to get dataset \(\mathcal{D}_\pi = \{o_1, \ldots, o_M\}\)
3. Ask human to label \(\mathcal{D}_\pi\) with actions \(a_t\)
4. Aggregate: \(\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi\)
What’s the problem?

**Pretrain & finetune**

1. collected demonstration data \((s_i, a_i)\)
2. initialize \(\pi_\theta\) as \(\max_\theta \sum_i \log \pi_\theta(a_i | s_i)\)
3. run \(\pi_\theta\) to collect experience
4. improve \(\pi_\theta\) with any RL algorithm

---

Can we avoid forgetting the demonstrations?

---

Can be very bad (due to distribution shift)

First batch of (very) bad data can destroy initialization
Off-policy reinforcement learning

- Off-policy RL can use any data
- If we let it use demonstrations as off-policy samples, can that mitigate the exploration challenges?
  - Since demonstrations are provided as data in every iteration, they are never forgotten
  - But the policy can still become *better* than the demos, since it is not forced to mimic them

off-policy policy gradient (with importance sampling)

off-policy Q-learning
Policy gradient with demonstrations

\[
\nabla_\theta J(\theta) = \sum_{\tau \in \mathcal{D}} \left[ \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left( \prod_{t'=1}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{q(a_{t'}|s_{t'})} \right) \left( \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right]
\]

includes demonstrations and experience

Why is this a good idea? Don’t we want on-policy samples?

best sampling distribution should have high reward!

optimal importance sampling

say we want \( E_p(x)[f(x)] \)

\[
E_p(x)[f(x)] \approx \frac{1}{N} \sum_i^N \frac{p(x_i)}{q(x_i)} f(x_i)
\]

which \( q(x) \) gives lowest variance?

answer: \( q(x) \propto p(x)|f(x)| \)
Policy gradient with demonstrations

$$\nabla_{\theta} J(\theta) = \sum_{\tau \in \mathcal{D}} \left[ \sum_{t=1}^{T} \nabla_{\theta} \log \pi_\theta(a_t|s_t) \left( \prod_{t'=1}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{q(a_{t'}|s_{t'})} \right) \left( \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right]$$

How do we construct the sampling distribution?

problem 1: which distribution did the demonstrations come from?

option 1: use supervised behavior cloning to approximate $\pi_{\text{demo}}$

option 2: assume Dirac delta: $\pi_{\text{demo}}(\tau) = \frac{1}{N} \delta(\tau \in \mathcal{D})$

this works best with self-normalized importance sampling

problem 2: what to do if we have multiple distributions?

fusion distribution: $q(x) = \frac{1}{M} \sum_i q_i(x)$
Example: importance sampling with demos

\[ \nabla_\theta J(\theta) = \sum_{\tau \in \mathcal{D}} \left[ \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left( \prod_{t'=1}^{t} \frac{\pi_\theta(a_{t'} \mid s_{t'})}{q(a_{t'} \mid s_{t'})} \right) \left( \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right] \]
Q-learning with demonstrations

• Q-learning is already off-policy, no need to bother with importance weights!
• Simple solution: drop demonstrations into the replay buffer

full Q-learning with replay buffer:
initialize $\mathcal{B}$ to contain the demonstration data

1. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$

2. sample a batch $(s_i, a_i, s'_i, r_i)$ from $\mathcal{B}$

$K \times$

3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i) (Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)])$
Q-learning with demonstrations

(a) Peg Insertion Task.  
(b) Hard-drive Task.  
(c) Clip Insertion Task  
(d) Cable Insertion Task.  

Vecerik et al., ’17, “Leveraging Demonstrations for Deep Reinforcement Learning...”
What’s the problem?

Importance sampling: recipe for getting stuck

\[ \nabla_{\theta} J(\theta) = \sum_{\tau \in D} \left[ \sum_{t=1}^{T} \nabla_{\theta} \log \pi_\theta(a_t | s_t) \left( \prod_{t'=1}^{t} \frac{\pi_\theta(a_{t'} | s_{t'})}{q(a_{t'} | s_{t'})} \right) \left( \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right] \]

Q-learning: just good data is not enough
So far...

• Pure imitation learning
  • Easy and stable supervised learning
  • Distributional shift
  • No chance to get better than the demonstrations

• Pure reinforcement learning
  • Unbiased reinforcement learning, can get arbitrarily good
  • Challenging exploration and optimization problem

• Initialize & finetune
  • Almost the best of both worlds
  • ...but can forget demo initialization due to distributional shift

• Pure reinforcement learning, with demos as off-policy data
  • Unbiased reinforcement learning, can get arbitrarily good
  • Demonstrations don’t always help

• Can we strike a compromise? A little bit of supervised, a little bit of RL?
Imitation as an auxiliary loss function

imitation objective: $\sum_{(s,a) \in \mathcal{D}_{demo}} \log \pi_\theta(a|s)$ (or some variant of this)

RL objective: $E_{\pi_\theta}[r(s,a)]$ (or some variant of this)

hybrid objective: $E_{\pi_\theta}[r(s,a)] + \lambda \sum_{(s,a) \in \mathcal{D}_{demo}} \log \pi_\theta(a|s)$

need to be careful in choosing this weight
Example: hybrid policy gradient

\[ g_{aug} = \sum_{(s,a) \in \rho_D} \nabla_{\theta} \ln \pi_\theta(a|s) A_\pi(s,a) + \sum_{(s,a^*) \in \rho_D} \nabla_{\theta} \ln \pi_\theta(a^*|s) w(s,a^*) \]

increase demo likelihood

Rajeswaran et al., ’17, “Learning Complex Dexterous Manipulation...”
Example: hybrid Q-learning

\[ J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q). \]

Q-learning loss

n-step Q-learning loss

regularization loss because why not...

\[ J_E(Q) = \max_{a \in A} [Q(s, a) + l(a_E; a)] - Q(s, a_E) \]

margin-based loss on example action

Hester et al., ‘17, “Learning from Demonstrations...”
What’s the problem?

hybrid objective: $E_{\pi_\theta} [r(s, a)] + \lambda \sum_{(s, a) \in \mathcal{D}_{\text{demo}}} \log \pi_\theta(a | s)$

• Need to tune the weight
• The design of the objective, esp. for imitation, takes a lot of care
• Algorithm becomes problem-dependent
• Pure imitation learning
  • Easy and stable supervised learning
  • Distributional shift
  • No chance to get better than the demonstrations

• Pure reinforcement learning
  • Unbiased reinforcement learning, can get arbitrarily good
  • Challenging exploration and optimization problem

• Initialize & finetune
  • Almost the best of both worlds
  • ...but can forget demo initialization due to distributional shift

• Pure reinforcement learning, with demos as off-policy data
  • Unbiased reinforcement learning, can get arbitrarily good
  • Demonstrations don’t always help

• Hybrid objective, imitation as an “auxiliary loss”
  • Like initialization & finetuning, almost the best of both worlds
  • No forgetting
  • But no longer pure RL, may be biased, may require lots of tuning
Break
Challenges in Deep Reinforcement Learning
Some recent work on deep RL

- Deep Q-Networks
  Mnih et al. 2013

- RL on raw visual input
  Lange et al. 2009

- Guided policy search
  Levine et al. 2013

- Trust region policy optimization
  Schulman et al. 2015

- Deep deterministic policy gradients
  Lillicrap et al. 2015

- End-to-end visuomotor policies
  Levine*, Finn* et al. 2015

- AlphaGo
  Silver et al. 2016

- Supersizing self-supervision
  Pinto & Gupta 2016
Stability and hyperparameter tuning

• Devising stable RL algorithms is very hard
• Q-learning/value function estimation
  • Fitted Q/fitted value methods with deep network function estimators are typically not contractions, hence no guarantee of convergence
  • Lots of parameters for stability: target network delay, replay buffer size, clipping, sensitivity to learning rates, etc.
• Policy gradient/likelihood ratio/REINFORCE
  • Very high variance gradient estimator
  • Lots of samples, complex baselines, etc.
  • Parameters: batch size, learning rate, design of baseline
• Model-based RL algorithms
  • Model class and fitting method
  • Optimizing policy w.r.t. model non-trivial due to backpropagation through time
Tuning hyperparameters

• Get used to running multiple hyperparameters
  • `learning_rate = [0.1, 0.5, 1.0, 5.0, 20.0]`
• Grid layout for hyperparameter sweeps OK when sweeping 1 or 2 parameters
• Random layout generally more optimal, the only viable option in higher dimensions
• Don’t forget the random seed!
  • RL is self-reinforcing, very likely to get local optima
  • Don’t assume it works well until you test a few random seeds
  • Remember that random seed is not a hyperparameter!
The challenge with hyperparameters

• Can’t run hyperparameter sweeps in the real world
  • How representative is your simulator? Usually the answer is “not very”

• Actual sample complexity = time to run algorithm x number of runs to sweep
  • In effect stochastic search + gradient-based optimization

• Can we develop more stable algorithms that are less sensitive to hyperparameters?
What can we do?

• Algorithms with favorable improvement and convergence properties
  • Trust region policy optimization [Schulman et al. ‘16]
  • Safe reinforcement learning, High-confidence policy improvement [Thomas ‘15]

• Algorithms that adaptively adjust parameters
  • Q-Prop [Gu et al. ‘17]: adaptively adjust strength of control variate/baseline

• More research needed here!

• Not great for beating benchmarks, but absolutely essential to make RL a viable tool for real-world problems
Sample Complexity
gradient-free methods (e.g. NES, CMA, etc.)

- 10x

fully online methods (e.g. A3C)

- 10x

policy gradient methods (e.g. TRPO)

- 10x

replay buffer value estimation methods (Q-learning, DDPG, NAF, etc.)

- 10x

model-based deep RL (e.g. guided policy search)

- 10x

model-based “shallow” RL (e.g. PILCO)

- 10x

Wang et al. ‘17

TRPO+GAE (Schulman et al. ‘16)

10,000,000 steps (10,000 episodes) (~ 1.5 days real time)

Chebotar et al. ‘17 (note log scale)

about 20 minutes of experience on a real robot

Chebotar et al. ‘17

10x gap

Gu et al. ‘16

1,000,000 steps (1,000 episodes) (~ 3 hours real time)

100,000,000 steps (100,000 episodes) (~ 15 days real time)
What about more realistic tasks?

- Big cost paid for dimensionality
- Big cost paid for using raw images
- Big cost in the presence of real-world diversity (many tasks, many situations, etc.)
The challenge with sample complexity

• Need to wait for a long time for your homework to finish running
• Real-world learning becomes difficult or impractical
• Precludes the use of expensive, high-fidelity simulators
• Limits applicability to real-world problems
What can we do?

• Better model-based RL algorithms

• Design faster algorithms
  • Q-Prop (Gu et al. ‘17): policy gradient algorithm that is as fast as value estimation
  • Learning to play in a day (He et al. ‘17): Q-learning algorithm that is much faster on Atari than DQN

• Reuse prior knowledge to accelerate reinforcement learning
  • RL2: Fast reinforcement learning via slow reinforcement learning (Duan et al. ‘17)
  • Learning to reinforcement learning (Wang et al. ‘17)
  • Model-agnostic meta-learning (Finn et al. ‘17)
Scaling up deep RL & generalization

- Large-scale
- Emphasizes diversity
- Evaluated on generalization

- Small-scale
- Emphasizes mastery
- Evaluated on performance
- Where is the generalization?
Generalizing from massive experience

Pinto & Gupta, 2015

Levine et al. 2016
Generalizing from multi-task learning

• Train on multiple tasks, then try to generalize or finetune
  • Policy distillation (Rusu et al. ‘15)
  • Actor-mimic (Parisotto et al. ‘15)
  • Model-agnostic meta-learning (Finn et al. ‘17)
  • many others...

• Unsupervised or weakly supervised learning of diverse behaviors
  • Stochastic neural networks (Florensa et al. ‘17)
  • Reinforcement learning with deep energy-based policies (Haarnoja et al. ‘17)
  • many others...
Generalizing from prior knowledge & experience

• Can we get better generalization by leveraging off-policy data?

• Model-based methods: perhaps a good avenue, since the model (e.g. physics) is more task-agnostic

• What does it mean to have a “feature” of decision making, in the same sense that we have “features” in computer vision?
  • Options framework (mini behaviors)
    • Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning (Sutton et al. ’99)
    • The option-critic architecture (Bacon et al. ’16)
  • Muscle synergies & low-dimensional spaces
    • Unsupervised learning of sensorimotor primitives (Todorov & Gahramani ’03)
Reward specification

• If you want to learn from many different tasks, you need to get those tasks somewhere!
• Learn objectives/rewards from demonstration (inverse reinforcement learning)
• Generate objectives automatically?
Learning as the basis of intelligence

- Reinforcement learning = can reason about decision making
- Deep models = allows RL algorithms to learn and represent complex input-output mappings

Deep models are what allow reinforcement learning algorithms to solve complex problems end to end!
What can deep learning & RL do well now?

- Acquire high degree of proficiency in domains governed by simple, known rules
- Learn simple skills with raw sensory inputs, given enough experience
- Learn from imitating enough human-provided expert behavior
What has proven challenging so far?

• Humans can learn incredibly quickly
  • Deep RL methods are usually slow

• Humans can reuse past knowledge
  • Transfer learning in deep RL is an open problem

• Not clear what the reward function should be

• Not clear what the role of prediction should be
What is missing?

How Much Information Does the Machine Need to Predict?

- "Pure" Reinforcement Learning (cherry)
  - The machine predicts a scalar reward given once in a while.
  - A few bits for some samples

- Supervised Learning (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - 10→10,000 bits per sample

- Unsupervised/Predictive Learning (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - Millions of bits per sample

(Yes, I know, this picture is slightly offensive to RL folks. But I’ll make it up)
Where does the supervision come from?

• Yann LeCun’s cake
  • Unsupervised or self-supervised learning
  • Model learning (predict the future)
  • Generative modeling of the world
  • Lots to do even before you accomplish your goal!
• Imitation & understanding other agents
  • We are social animals, and we have culture – for a reason!
• The giant value backup
  • All it takes is one +1
• All of the above
How should we answer these questions?

• Pick the right problems!
• Pay attention to generative models, prediction
• Carefully understand the relationship between RL and other ML fields