Exact Policy Gradient in a Simulated Environment

Rosemary Nan Ke, Alex Lamb

Policy Gradient Methods

-Directly parameterize a mapping from states to actions - a policy.

-Use an estimator of the gradient of future rewards with respect to the policy.

-Take gradient steps with respect to the policy to improve future rewards.

-Advantages:

- -Can learn a stochastic policy.
- -The policy can have a cleaner parameterization than the value function.
- -Continuous action spaces.

-Disadvantages:

- -High variance. Often have non-stationary objective

Actor-Critic

-This is the most common and successful variant of policy gradient.

-Idea is that a critic network approximates the gradient with respect to the current actor policy.

-The actor is trained to maximize the critic's estimated rewards. Can use TD error.

-What's wrong with Actor-Critic?

  -Hard to do off-policy.

  -Non-stationary objective because what critic learns depends on current actor.

  -May require Temporal Difference method.

  -Each update to actor requires interacting with the environment - bottleneck

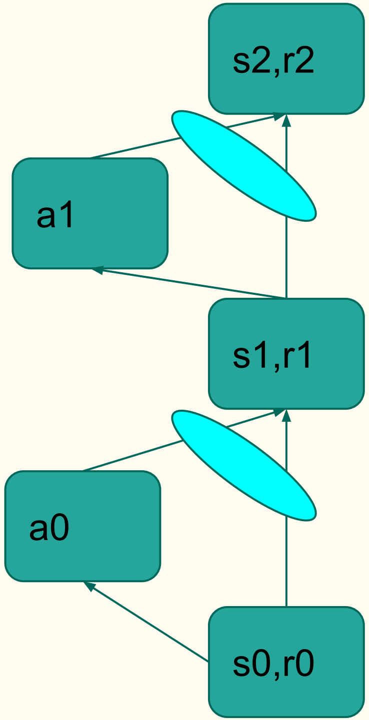
Proposal: Exact Policy Gradient wrt Simulation

-Learn a model which simulates the environment: mapping from the current state and action to the next state.

-Then you can do exact (in the context of the simulated environment) policy gradient on future rewards with respect to the policy network.

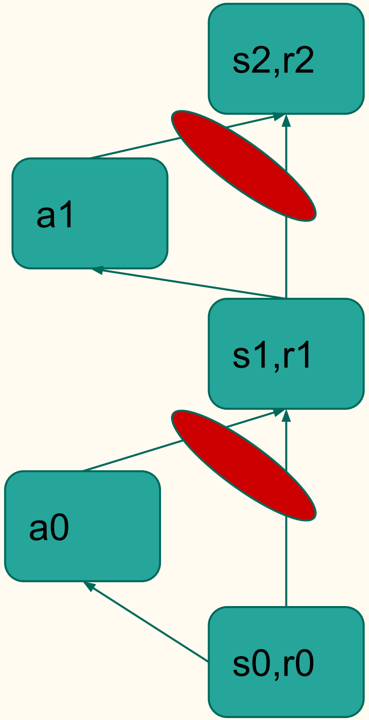
-Need to run with the policy network (or some variant of it) to get the data to train the environment simulator.

Exact Policy Gradient



d[Σ Reward]
----
d[$theta_{policy}$]

Simulation Chain

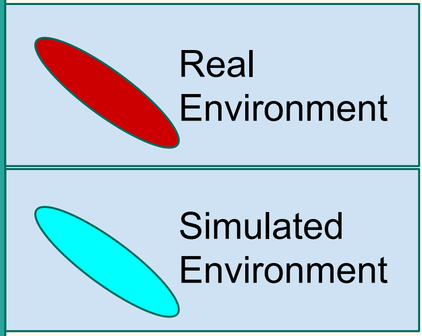


Real Environment Chain

-Both chains use the policy network to pick actions.

-Real Environment Chain runs in the real world and collects data to train the simulator.

-Simulation chain is used to compute exact policy gradients.



Some interesting variants

-In something like a 2-player game, the environment simulator becomes a model of the other player.

-So the environment simulator and the policy network could share most of their parameters (self-play).

Advantages of Exact Policy Gradient

-More stationary objective: the optimal environment simulator doesn’t depend on future policy within an episode (but still depends on past policy - i.e. games).

-On the other hand optimal-critic depends on both past and future policy.

-All of the policy gradient updates are simulated - completely decoupled from actually running the real environment. Could be very appealing in some cases.

-Highly interpretable - can see strengths and weaknesses in model simulator.

-We may already have an environment simulator or have a reason to want to learn one.

-Easy to do off-policy planning?

Disadvantages of Exact Policy Gradient

-Could potentially be much more expensive, especially if there are parts of the environment which have nothing to do with getting rewards.

-Backpropagation through long chains of actions is computationally expensive.

-Hard to handle discrete states.

-What is the right loss for the environment simulator? Potentially easy to mismatch with what we really care about!

-May need massive amounts of data to train a quality environment simulator.

-Very deep gradients - can still be a challenge to do credit assignment this way.

Task - Pendulum

-Selected due to simplicity and because it has continuous state/action space.

-Task is to balance the pendulum.

-For an episode of length 200, a good reward is like -123.

-OpenAI gets there in about 1300 training episodes

Architecture and Model Details

-Environment simulator: 3-layer MLP with 512 units, layer norm.

-Policy network: 3-layer MLP with 512 units, layer norm.

-When training the environment simulator take random steps (works here due to random state initialization).

-Only take policy gradient steps when environment simulator is accurate enough.

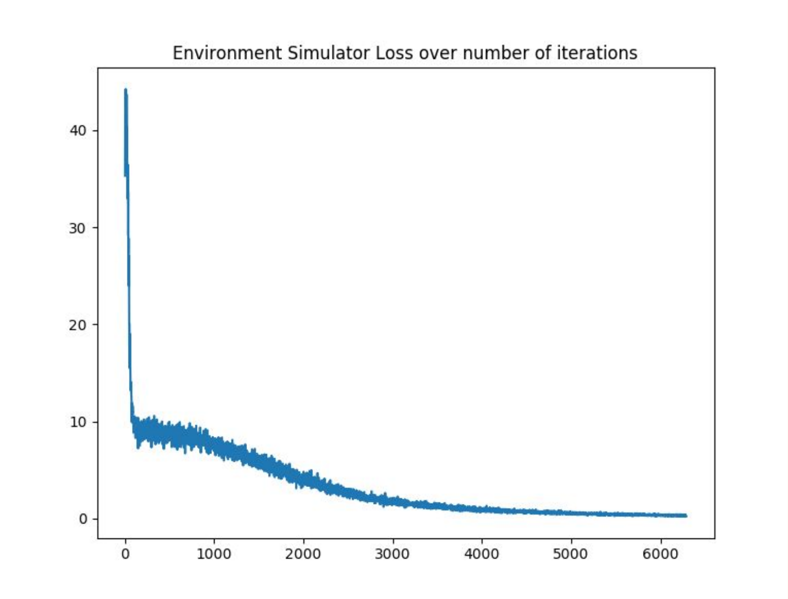
-Optimize the environment simulator using square loss.

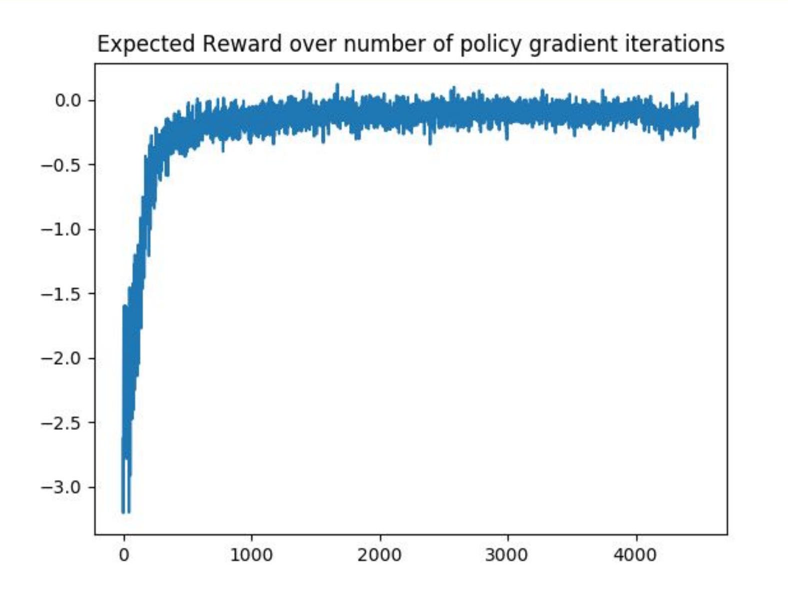
-Run each episode for 200 time steps.

-Epsilon-greedy exploration policy.

Results - Simplified Version

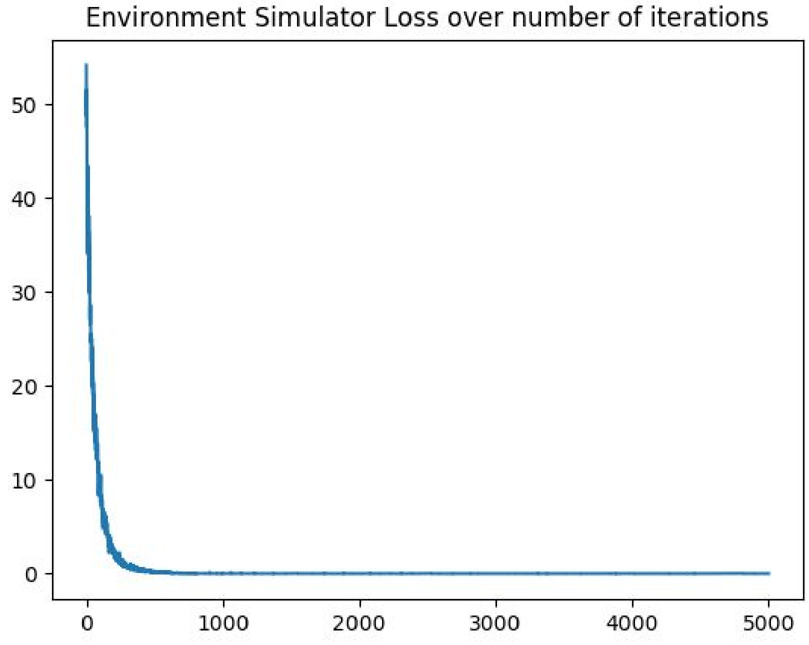
-To test if basic idea works: let the action be the velocity of the pendulum and run for 5 steps. (Removed long term dependencies). Expected reward is average over

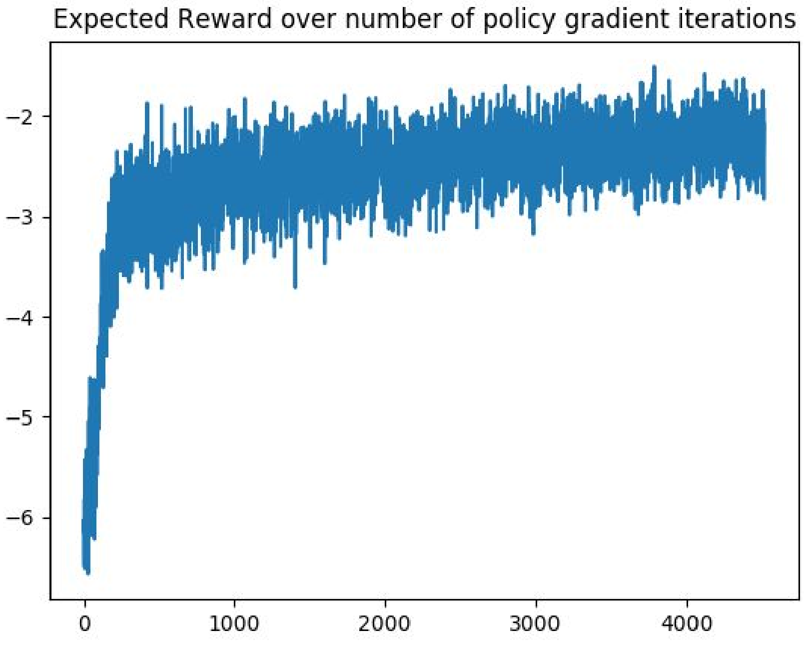




Results - Actual Pendulum Task

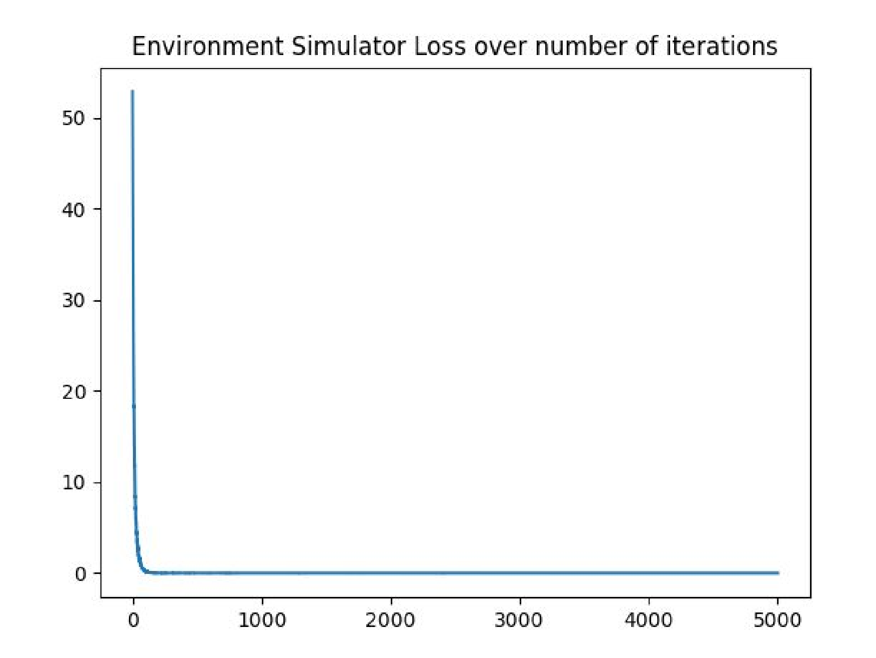
-Ran for 10 steps - hard to get stable gradient with large number of steps.

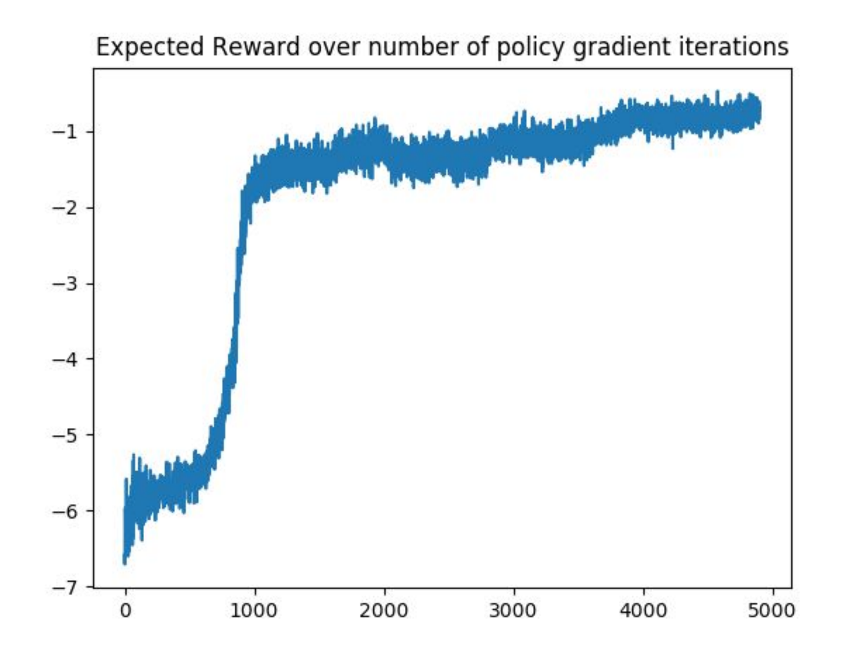




Actual Pendulum Task

-50 steps. Should work *much* better than 10 steps. (eventually does but slow!)





Comments on getting it to work

-Some implementation details that don’t really matter for other RL methods matter a lot here.

-For example, it’s critical that the state, action, and reward sequences are aligned correctly when training the environment simulator.

How to make it tractable?

-Use an online gradient estimator instead of doing full backpropagation.

-One is Synthetic Gradients (Vinyals et. al 2016).

-Another is approximate forward mode differentiation, like UORO (Tallec and Ollivier 2017).

-Is this better or worse than using a value function? How is it different?

-For example, a synthetic gradient module is implicitly saying something about the value of a state action pair.

Backpropagation through discrete states

-We avoided it for now, but it eventually would be an issue.

-Ironically many propose using value-function based RL for getting around discrete states!

-Potential solutions:

- -Just avoid discreteness.
- -Gumbel-Softmax

What is the right loss for the simulator?

-What really matters is if it learns to provide the right gradients for training the policy network?

-But we don’t have “ground truth” gradients.

Amount of data required for training env. sim.

-Training the environment simulator could require massive amounts of data to be accurate enough to be useful.

-But is this necessarily a requirement?

-Could we consider learning a “one shot” generative model, which is able to learn a strong model from just a single episode of training data?

-Could also consider transfer learning, with “one shot” adaptation to a new domain or setting.

Conclusion

-We've explored model-based RL - specifically the idea of exactly passing policy gradients through a neural environment simulator.

-Achieved semi-positive results on the Pendulum toy tasks - seems to require lots of data!

-We've discussed how some new advances in deep learning could make such an approach tractable.

-We've also motivated new areas for deep learning research which could advance this type of model-based RL.