Deep Reinforcement Learning (Policy Gradients & Pong)

Deep Learning: Bryan Pardo, Northwestern University, Fall 2020

Policy Gradient Methods

Based on materials from

Andrej Karpathy http://karpathy.github.io/2016/05/31/rl/

John Schulman https://www.youtube.com/watch?v=y4ci8whvS1E&t=2230s

Policy vs Q or V

- A Q(s,a) function estimates the value of action a if you're in state s.
- A V(s) function outputs the expected value of being in state s.
- Neither of these tells you what to do. They tell you what to value.
- A policy $\pi(s)$ outputs what action to take when you're in state s.
- Why not learn the policy directly?

Policy Gradient Methods

- This results in hill-climbing in policy space
  - So, it's subject to all the problems of hill-climbing
  - But…we can also use tricks from search, like random restarts and momentum terms

- This is a good approach if you have a parameterized policy
  - Often faster than value-based methods
  - "Safe" exploration, if you have a good policy
  - Learns **locally-best** parameters ***for that policy***

Problem formulation

- Let θ be the parameters of our policy function π_θ
- We want to maximize the expected return R of the policy.

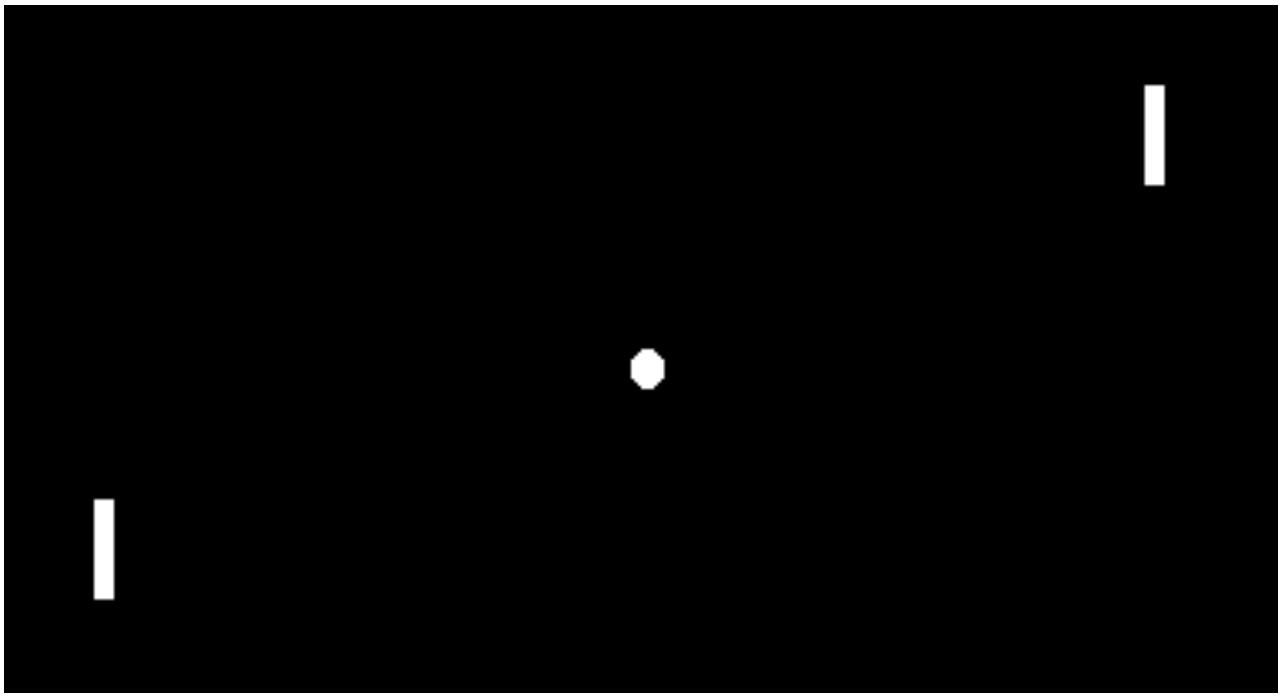
$$\underset{\theta}{\operatorname{argmax}}\, E[R \mid \pi_\theta]$$

- We'll do this by taking the gradient of the policy function with respect to the model parameters and then hill climbing.

Intuitive Approach

- Sample a bunch of action sequences from an existing policy π_θ
- Measure the return of these action sequences
- Make the good action sequences more probable by varying θ

Example: Learning to play Pong



Markov Decision Processes: Pong edition

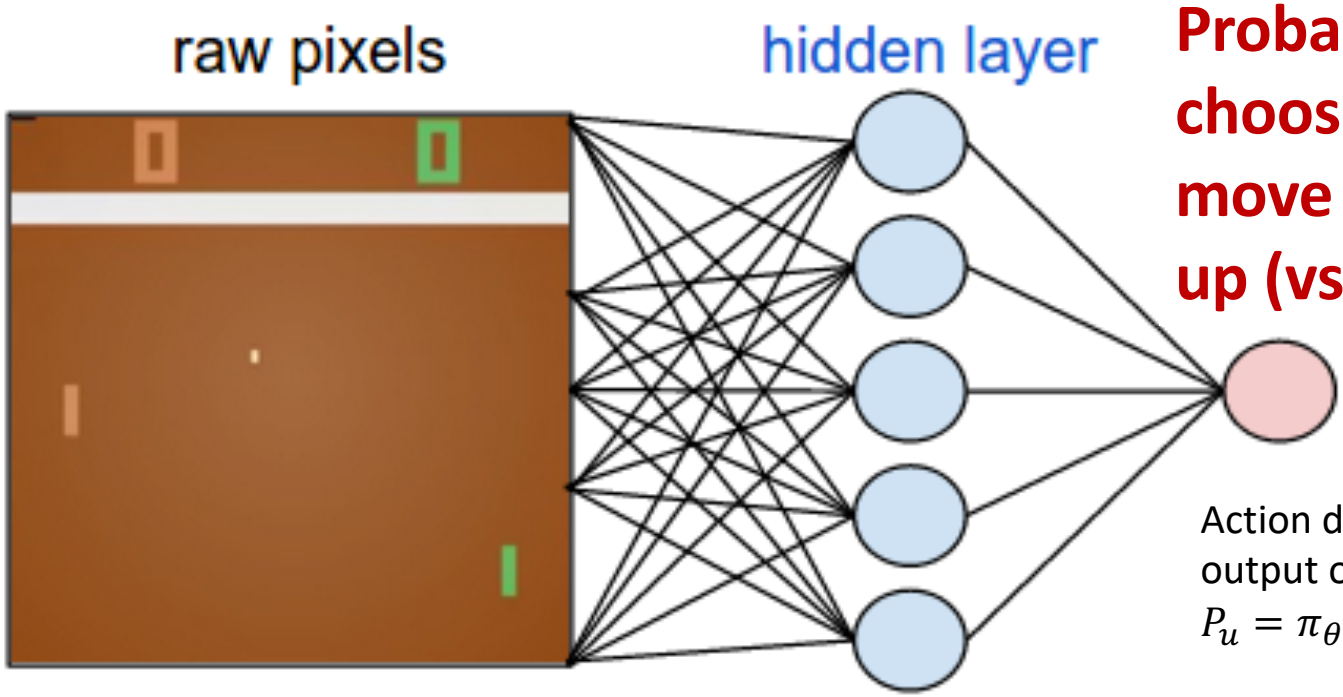
- A set of states, $S = \{s_1, s_2, \ldots, s_n\}$: **Image frames in Pong**
- A set of actions, $A = \{a_1, a_2, \ldots, a_m\}$: **Move paddle up or down**
- A reward function, $R: S \times A \times S \rightarrow \Re$: **+1 for win, -1 for lose, 0 for all other states**
- A transition function, **Paddle action has intended effect**

  $$P_{ij}^{a} = P(s_{t+1} = j \mid s_t = i, a_t = a)$$

- A probability distribution over what the initial state will be: $\mu(s)$

- We want to learn a policy, $\pi: S \rightarrow A$
  - Maximize sum of rewards we see over our lifetime

The policy network



Probability of choosing to move paddle up (vs down)

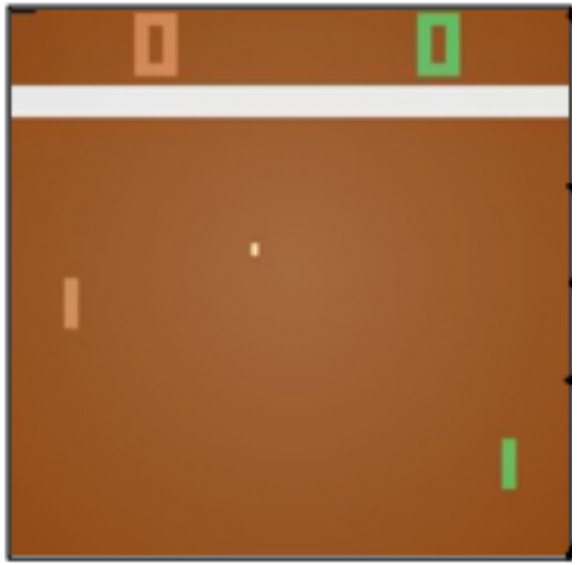
Action distribution is the output of the policy $P_u = \pi_\theta(s_t)$.

State s_t is an image frame of 210 by 160 pixels (that's kind of a lie. It's actually the cropped difference between two time steps)

Policy $\pi_\theta(s_t)$ is the function encoded by a network with tunable parameters θ

What do we need the input state to capture?

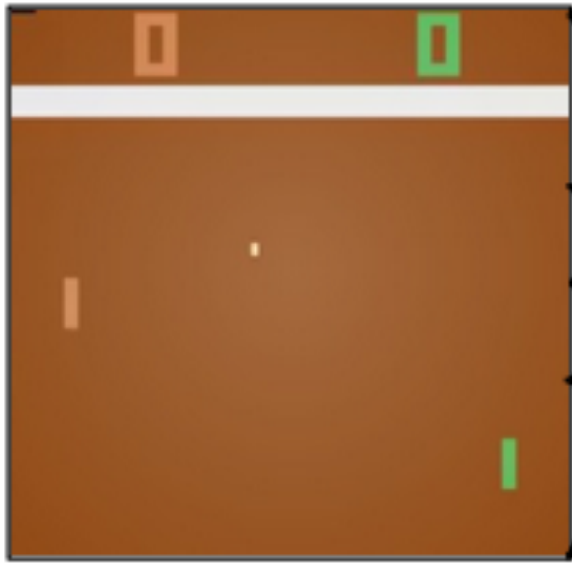
raw pixels



- We need
  - Where the paddles are
  - Where the ball is
  - What direction they're all moving in
- Can we get all that from a single still frame?
- What about if we looked at the difference between 2 sequential frames?

What don’t we need?

raw pixels



- Do we need the score?
- Do we need full resolution?
- How much smaller can we make the state space without these things?

Score Function Gradient Estimator

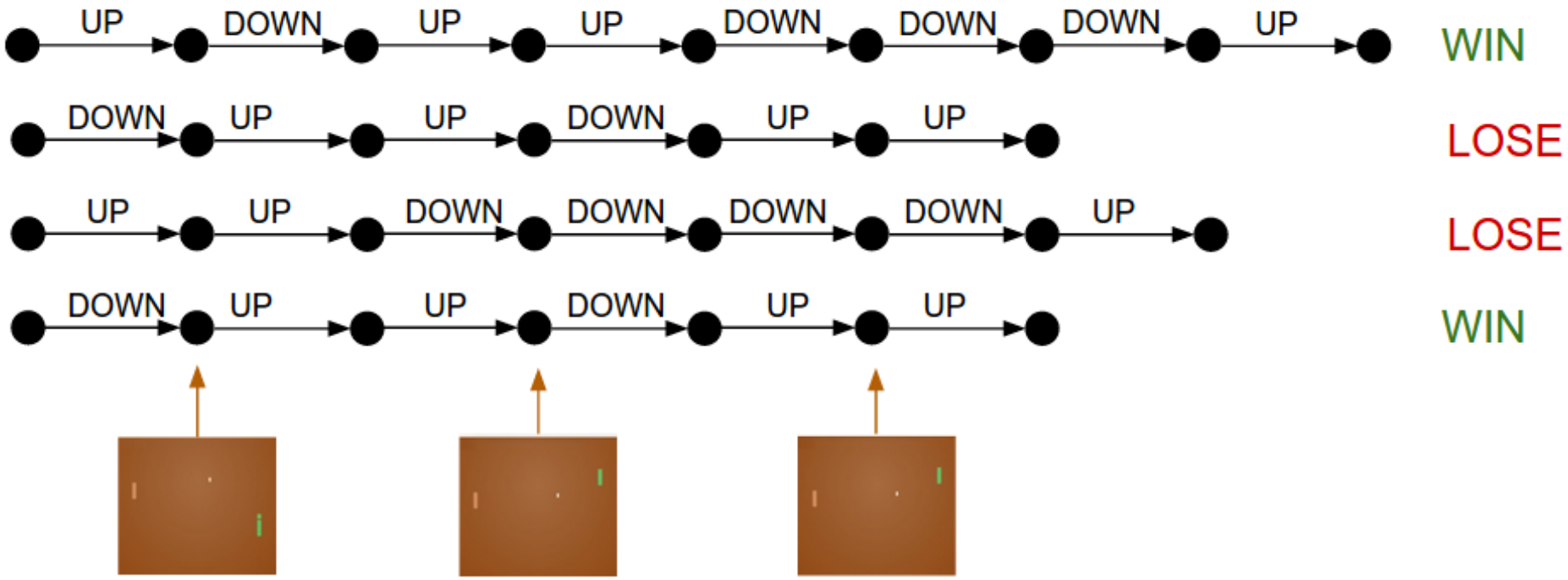
- A trajectory τ is a state, action, reward sequence returned by a policy

$\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$

- Let $R(\tau)$ give the return for the trajectory.

- We want to estimate this gradient: $\nabla_\theta E[R(\tau)]$

Trajectories in Pong



Let's derive the "Score Function" estimator

- We thinking of τ as a random variable…and we want to do this

$$\nabla_\theta E[R(\tau)] = E[\nabla_\theta P(\tau|\theta) R(\tau)]$$

The gradient (with respect to the model parameters θ) of the expected reward for trajectory τ

- To do this, we need to expand and understand $P(\tau|\theta)$

Expanding and explaining $P(\tau|\theta)$

A trajectory

$$\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

$$P(\tau|\theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}, r_t|s_t, a_t)$$

The probability of this trajectory, given our model parameters

Probability of starting a sequence in this state

The probability our policy would return this action, given this state

The probability we'd get this state & reward, given that state & action

Doing some math

$$P(\tau|\theta) = \mu(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)P(s_{t+1}, r_t|s_t, a_t)$$

$$\log(P(\tau|\theta)) = \log\mu(s_0) + \sum_{t=0}^{T}[\log\pi_\theta(a_t|s_t) + \log P(s_{t+1}, r_t|s_t, a_t)]$$

$$\nabla_\theta \log(P(\tau|\theta)) = \nabla_\theta \sum_{t=0}^{T-1} \log\pi_\theta(a_t|s_t)$$

Plugging that back into the original equation

$$\nabla_\theta E[R(\tau)] = E\left[R(\tau)\,\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\right]$$

$$= E\left[\sum_{t=0}^{T-1} r_t\,\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\right]$$

This reorders things so that we consider the value of an action as being the sum of future rewards in the trajectory

$$= E\left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T-1} r_{t'}\right)\right]$$

What are we doing again?

- We want to maximize the expected return of the actions taken by our policy.
- This means estimating the gradient of the reward for taking some action.

$$\nabla_\theta E[R(\tau)] = E\left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'}\right)\right]$$

- What if all actions have positive rewards, but some are much bigger?
- Can we do better?

Comparing to a baseline b(s)

$$\nabla_\theta E[R(\tau)] = E\left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t)\right)\right]$$

This is a baseline expectation for how much reward we might typically see in this state

Q: Why do we introduce a baseline?

A: We would prefer to increase the probability actions that give more future reward than you typically see when you take an action.

Given a set of trajectories, the average reward over the trajectories from time t to the end and call that our baseline value $b(s_t)$

Adding a discount factor for distant rewards

$$\nabla_\theta E[R(\tau)] = E\left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t)\right)\right]$$

If you think that rewards far into the future are unrelated to the current action, we add the discount factor $\gamma^{t'-T}$.

Note: we need to calculate our baseline $b(s_t)$ using the same discount factor.

The advantage function

$$A_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t)$$

We call the difference between the baseline expectation for reward at step t and the observed reward at step t on this particular trajectory the Advantage A_t. This gives the formula below.

$$\nabla_\theta E[R(\tau)] = E\left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) A_t\right]$$

The advantage function: what this does

$$A_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t)$$

Actions that have higher-than-baseline rewards have a positive Advantage.

Actions with lower-than baseline rewards will have negative Advantage

When we optimize to maximize the expected advantage, this means our policy will vary the parameters to make advantageous actions more likely and disadvantageous actions less likely.

The policy gradient algorithm

Initialize policy parameter θ, baseline b

for iteration=$1, 2, \ldots$ **do**

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the *return* $\hat{R}_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the *advantage estimate* $\hat{A}_t = \hat{R}_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,

summed over all trajectories and timesteps.

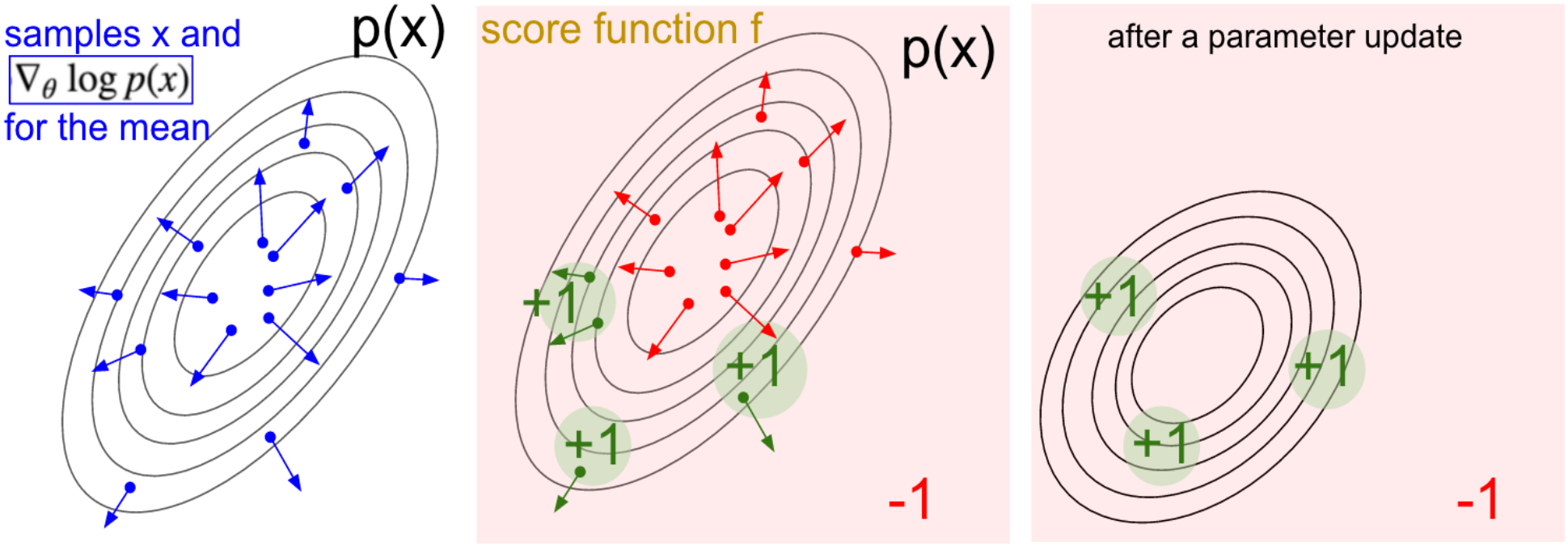
Update the policy, using a policy gradient estimate $\hat{g}$,

which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta) \hat{A}_t$

end for

John Schulman @ Open AI

Visualizing probabilities of taking actions



Proximal Policy Optimization

https://arxiv.org/pdf/1707.06347.pdf

One more tweak: clipping our learning

- Often, our policy gradient estimates give us really steep gradients that cause overstepping the goal.
- This can negatively impact on learning.
- In supervised learning, we can “fix” this with gradient clipping.
- Gradient clipping can be done in RL, too.

In a nutshell:

- Run your RL algorithm on the old policy $\pi_{\theta_{old}}$ to get a new policy π_θ

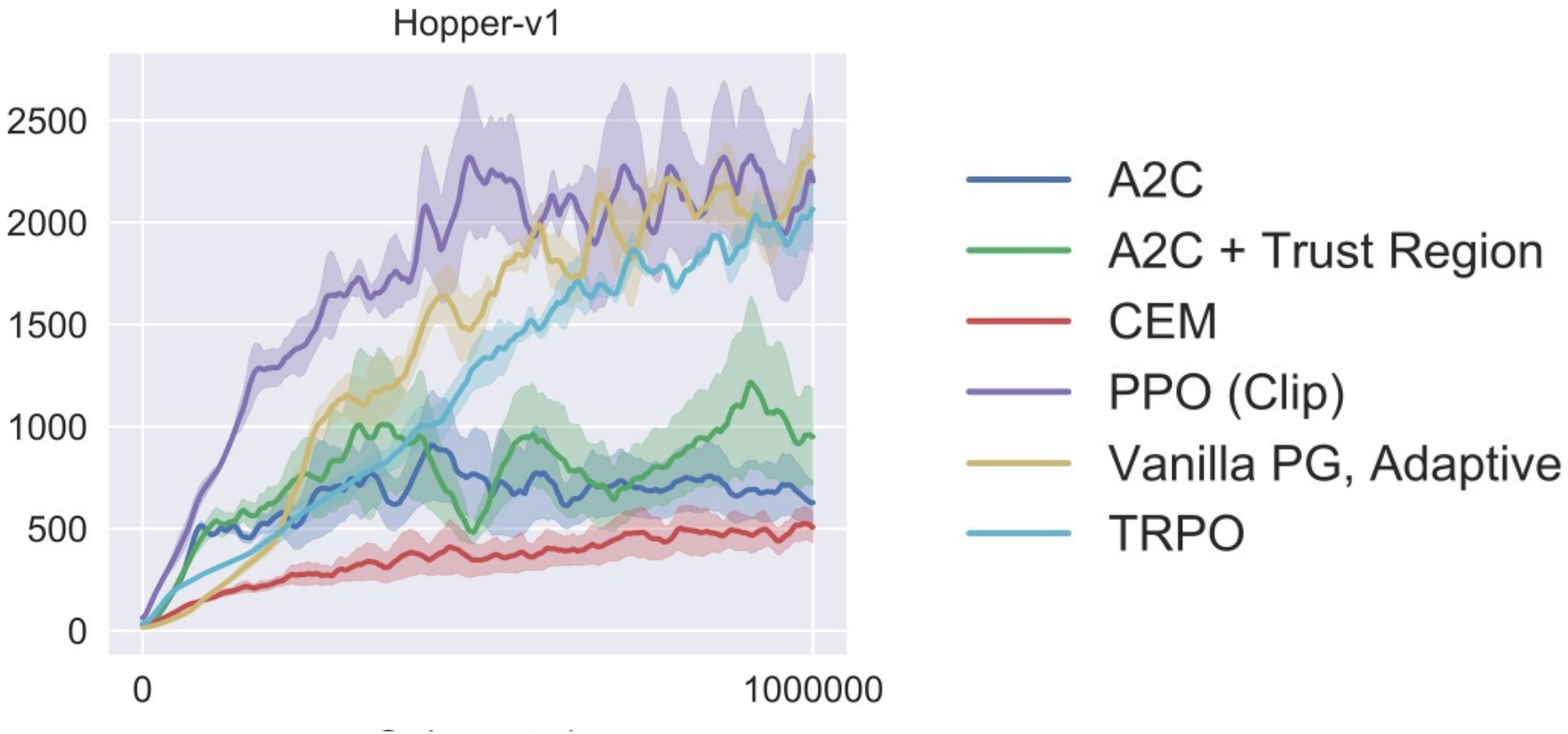
- Calculate how much the policy has changed the probability of an action

$$x = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

- Re-calculate the loss now by clipping it so that the ratio of the probabilities between the old and new policy is never more than.

$$L^{CLIP} = \hat{\mathrm{E}}_t\left[\min(x, clip(x, 1-\epsilon, 1+\epsilon)\hat{A}_t\right]$$

This seems to help a lot.



This seems to help a lot.