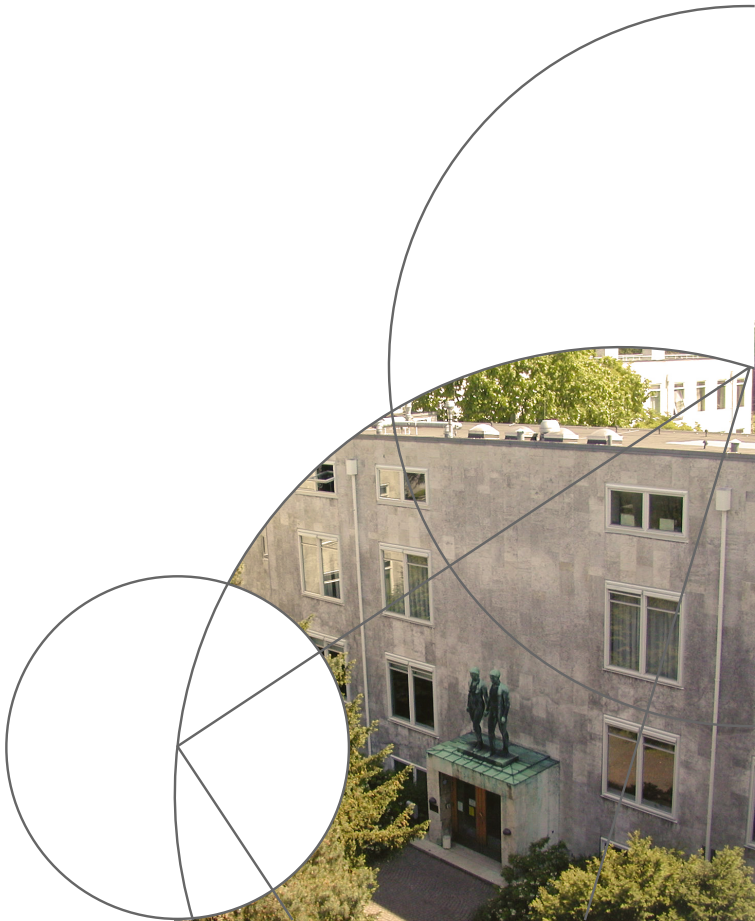
Deep Reinforcement Learning

Reinforcement Learning

Christian Igel
Department of Computer Science



igel@diku.dk

Outline

Outline

Function approximators

- Value functions represented as tables are very limited:
  - Not suitable for continuous state and/or action spaces
  - No generalization (learning about unseen states or state-action pairs from similar states or state-action pairs)
- We can use function approximators with parameters $\boldsymbol{w}$ for the state value function

$$\hat{V} : S \to \mathbb{R}$$

or the state-action value function

$$\hat{Q} : S \times A \to \mathbb{R}$$

Examples of function approximators

- Look-up tables (as in tabular Q-learning) can be viewed as special cases of function approximators that do not generalize.
- For example, assume a feature mapping $\boldsymbol{\phi} : S \to \mathbb{R}^d$ and discrete actions $A = \{a_1, \ldots, a_K\}$, we could have linear approximators

$$\hat{Q}(s, a_i) = \hat{Q}_i(s) = \boldsymbol{w}_i^\mathsf{T} \boldsymbol{\phi}(s)$$

  with $\boldsymbol{w}_i^\mathsf{T} \in \mathbb{R}^d$ for each $i = 1, \ldots, K$.
- If the function approximators are deep neural networks, we talk of Deep RL.

Targets and error for the value function

- Estimate V^π for the current policy π. We have (dropping superscript):

$$V(s_t) = \mathbb{E}[R_t] = \mathbb{E}\left[\sum_{k=1}^{T-t} r_{t+k}\right]$$

- Halved sum of squares error of current value function estimate ($\hat{V}$ parameterized by $\boldsymbol{w}$) for state s: $\frac{1}{2}(\hat{V}(s) - V(s))^2$
- Estimated approximation error of $\hat{V}$ from data $\mathcal{S} = \{(s_{t_1}, R_{t_1}), (s_{t_2}, R_{t_2}), \dots\}$:

$$\hat{L}(\boldsymbol{w}) = \frac{1}{2|\mathcal{S}|} \sum_{(s,R)\in\mathcal{S}} (\hat{V}(s) - R)^2$$

Data could be from a single episode or multiple episodes.

Gradient

Let's consider $\mathcal{S} = \{(s_{t_1}, R_{t_1}), (s_{t_2}, R_{t_2}), \dots\}$ and

$$\hat{L}(\boldsymbol{w}) = \frac{1}{2|\mathcal{S}|} \sum_{(s,R)\in\mathcal{S}} (\hat{V}(s) - R)^2$$

leading to gradient descent learning rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \underbrace{\sum_{(s,R)\in\mathcal{S}} (\overbrace{\hat{V}(s) - \underbrace{R}_{\text{target } y \text{ for } \hat{V}(s)}}^{\delta}) \nabla_{\boldsymbol{w}} \hat{V}(s)}_{\nabla_{\boldsymbol{w}} \hat{L}(\boldsymbol{w})}$$

with learning rate $\eta > 0$.

Note: We could also consider $(R - \hat{V}(s))^2$ instead of $(\hat{V}(s) - R)^2$ leading to a sign change.

Outline

Learning a baseline

The policy gradient theorem can be generalized, in either average-reward or start-state formulations, to include a baseline, e.g.,

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \sum_s \mu^\pi(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(s,a) \left(Q^\pi(s,a) - b(s)\right) \quad ,$$

where $b(s) : S \to \mathbb{R}$ is an arbitrary baseline function.

$\sum_a \nabla_{\boldsymbol{\theta}} \pi(s,a) b(s)$ acts as a control variate. Note $\mathbb{E}[\sum_a \nabla_{\boldsymbol{\theta}} \pi(s,a) b(s)] = 0$.

A possible choice of $b(s)$ would be some estimate of the value function $V(s)$.

REINFORCE with baseline

Algorithm 1: REINFORCE with baseline

Input: differentiable policy π parameterized by $\boldsymbol{\theta}$, differentiable state-value function $\hat{V}$ parameterized by $\boldsymbol{w}$, learning rates $\alpha_{\boldsymbol{w}}, \alpha_{\boldsymbol{\theta}} > 0$, initial $\boldsymbol{\theta}$ and $\boldsymbol{w}$

repeat
- Generate episode $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$
- **foreach** $t = 1, \ldots, T-1$ **do**
  - $R_t = \sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k}$
  - $\delta \leftarrow R_t - \hat{V}(s_t)$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\boldsymbol{w}} \delta \nabla_{\boldsymbol{w}} \hat{V}(s_t)$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \ln \pi(s_t, a_t)$

until *stopping criterion is met*

Outline

Actor-Critic Methods

- Actor-critic method:
  - Policy π with parameters $\boldsymbol{\theta}$: actor
  - Value function with parameters $\boldsymbol{w}$: critic
- REINFORCE with baseline is not fully online
- Introduce bootstrapping: Target not purely based on Monte Carlo return, but on previous estimate(s)
- Temporal difference learning: From

$$V^{\pi}(s_t) = \mathbb{E}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1})\right]$$

we get the new

$$\delta = \underbrace{r_{t+1} + \gamma \hat{V}(s_{t+1})}_{\text{target}} - \hat{V}(s_t)$$

One-step Actor-Critic

Algorithm 2: One-step Actor Critic

differential policy π parameterized by $\boldsymbol{\theta}$, differential state-value function $\hat{V}$ parameterized by $\boldsymbol{w}$, learning rates $\alpha_{\boldsymbol{w}}, \alpha_{\boldsymbol{\theta}} > 0$, initial $\boldsymbol{\theta}$ and $\boldsymbol{w}$

repeat

  $t \leftarrow 0$; $s_0 \sim p_{\text{start}}$

  repeat

    $a_t \sim \pi(s_t, \cdot)$

    take action a_t and observe r_{t+1} and s_{t+1}

    $$\delta \leftarrow \begin{cases} r_{t+1} - \hat{V}(s_t) & \text{if } s_{t+1} \text{ terminal} \\ r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) & \text{else} \end{cases}$$

    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\boldsymbol{w}} \delta \nabla_{\boldsymbol{w}} \hat{V}(s_t)$

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \ln \pi(s_t, a_t)$

    $t \leftarrow t + 1$;

  until *terminal state is reached*

until *stopping criterion is met*

Outline

State-value function targets

In the following, we derive a simple version of Q-learning that works with neural network function approximators.

Recall temporal difference leanring rules for Q^π for the current policy π:

One-step Sarsa (on-policy):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})}_{\text{target } y(s_t, a_t)} - Q(s_t, a_t)]$$

One-step Q-learning (off-policy):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)}_{\text{target } y(s_t, a_t)} - Q(s_t, a_t)]$$

Mean squared error (MSE)

- (Halved) MSE between learnt Q-function with parameters $\boldsymbol{w}$ and target:

$$L(\boldsymbol{w}) = \mathbb{E}_{s,a,r,s',a'}\left[\frac{1}{2}\left(y(s,a) - \hat{Q}(s,a)\right)^2\right]$$

- Estimated from data $\mathcal{S} = \{((s_1,a_1), y(s_1,a_1)), \dots\}$:

$$\hat{L}(\boldsymbol{w}) = \frac{1}{|\mathcal{S}|} \sum_{((s,a),y(s,a))\in\mathcal{S}} \underbrace{\frac{1}{2}\left(y(s,a) - \hat{Q}(s,a)\right)^2}_{\hat{l}(\boldsymbol{w})}$$

Gradient

Assuming

$$\nabla_{\boldsymbol{w}} \hat{l} = -\left(y(s,a) - \hat{Q}(s,a)\right) \nabla_{\boldsymbol{w}} \hat{Q}(s,a)$$

gives learning rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha(y(s,a) - \hat{Q}(s,a)) \nabla_{\boldsymbol{w}} \hat{Q}(s,a)$$

What is the problem? We assume that $y(s,a)$ is independent of $\boldsymbol{w}_t$.

Estimated error from data $\mathcal{S} = \{(s_1, a_1, r_2, s_2), \dots\}$ for Q-learning (note that the current $\hat{Q}$ is used now):

$$\hat{L}(\boldsymbol{w}) = \sum_{(s,a,r,s') \in \mathcal{S}} \frac{1}{2} \left(r + \gamma \max_{a'} \hat{Q}(s,a') - \hat{Q}(s,a)\right)^2$$

Sanity check: Tabular function approximation

- Assume discrete A and S and a table with $|A| \times |S|$ entries corresponding to $|A| \times |S|$-dimensional parameter vector $\boldsymbol{w}$.
- Then in the expression from the previous slide

$$\nabla_{\boldsymbol{w}} \hat{l} = -\left(y(s,a) - \hat{Q}(s,a)\right) \nabla_{\boldsymbol{w}} \hat{Q}(s,a)$$

  $\left[\nabla_{\boldsymbol{w}} \hat{Q}(s,a)\right]_i$ is one for i corresponding to (s,a) and zero otherwise.
- Thus, tabular function approximation fits the presented gradient-based adaptation framework.

Function approximators for discrete actions

- Assume discrete actions $A = \{a_1, \ldots, a_K\}$.
- Instead of a mapping $S \times A \to \mathbb{R}$ we can learn

$$\hat{Q} : S \mapsto \mathbb{R}^{|A|}$$

  approximating $Q(s, a_i)$ by $\left[\hat{Q}(s)\right]_i$.
- In the following, we do not distinguish between $\hat{Q}(s, a_i)$ and $\left[\hat{Q}(s)\right]_i$ and set $\max_a \hat{Q}(s, a) \to 0$ if s is a terminal state.
- If $\hat{Q}$ is a deep neural network, we talk about deep Q-learning.

Variance reduction and experience reply

- Updating a neural network using stochastic gradient step based on single observation is not recommended because the variance is too high
- Solution 1: Accumulate the gradients and perform an update after T_{update} steps; easy to parallelize (Mnih et al., 2016)
- Solution 2: Store experiences $\langle s, a, r, s' \rangle$ in a finite FIFO buffer and sample mini-batches from this buffer for learning (Riedmiller 2005, Mnih et al., 2015),
  if buffer is full, the oldest element will be removed if a new one is added

Sketch of simple mini-batch Q-learning

Algorithm 3: Simple Q-learning

initialize finite experience memory M; initialize $\hat{Q}$
foreach *episode* **do**
    initialize s
    repeat
        choose a based on $\hat{Q}$ `// e.g., ϵ-greedy`
        take action a, observe r, s'
        store $\langle s, a, r, s' \rangle$ in M
        sample mini-batch $\left\{ \langle s_i, a_i, r_i, s'_i \rangle \,\middle|\, i = 1, \ldots \right\}$ from M
        set targets $r_i + \gamma \max_{a'} \left[\hat{Q}(s'_i) \right]_{a'}$ for $\left[\hat{Q}(s_i) \right]_{a_i}$
        update $\hat{Q}$ network weights
        $s \leftarrow s'$
    until *until s is terminal*

Recall: Greedy and soft policy

- Given state-action value function Q and finite action space A, the *greedy* policy is:

$$\pi^Q(s) = \underset{a \in A}{\operatorname{argmax}}\, Q(s,a)$$

- Need for exploration
- Soft policies: $\pi(s,a) > 0$ for all s and a
- Example: ϵ-soft (ϵ-greedy) policy, which follow the greedy policy but take a random action with a probability of ϵ at very step:

$$\text{probability for actions:} \quad \underbrace{\frac{\epsilon}{|A|}}_{\text{non-max}} \quad \underbrace{1 - \epsilon + \frac{\epsilon}{|A|}}_{\text{greedy}}$$

The exploration can be adjusted during learning, in particular by reducing ϵ over time.

Delayed Q update

- Updating the $\hat{Q}$ network changes also the targets for updating the $\hat{Q}$ network, which leads to instabilities
- Idea: Use a different target network for computing the temporal difference errors and update the target network on a slower time scale (Mnih et al., 2015, 2016)
- Same network architecture is used, only the parameters vary: we distinguish between $\boldsymbol{w}$ and $\boldsymbol{w}_{\text{target}}$
- Same for policy parameters: $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_{\text{target}}$
- Two ways for Delayed update:
  - Copy parameters (e.g., $\boldsymbol{w}_{\text{target}} \leftarrow \boldsymbol{w}$) every $T_{\text{target-update}}$ steps (Mnih et al., 2016)
  - Smoothly blend the parameters (e.g., $\boldsymbol{\theta}_{\text{target}} \leftarrow \rho\boldsymbol{\theta}_{\text{target}} + (1-\rho)\boldsymbol{\theta}$ for $\rho \in]0,1[$)

Notation

- A policy we currently optimize is denoted by π or π' with parameters $\boldsymbol{\theta}$ or $\boldsymbol{\theta}'$, respectively.
- A value function we learn is denoted by $\hat{V}$ or $\hat{Q}$ with parameters $\boldsymbol{w}$. The hat is sometimes omitted, it is used to stress that we are dealing with an approximation.
- A policy that may be outdated and used to compute the target values for learning is called π or π_{target}.
- A value function that may be outdated and used to compute the targets for learning is denoted by $\hat{V}_{\text{target}}$ or $\hat{Q}_{\text{target}}$ w/ parameters $\boldsymbol{w}_{\text{target}}$.
- Example: Critic update based on mini-batch $\mathcal{S} = \left\{ \langle s_i, a_i, r_i, s'_i \rangle \,\middle|\, i = 1, \ldots, N \right\}$ follows:

$$\nabla_{\boldsymbol{w}} \frac{1}{N} \sum_{i=1}^{N} \Big(\underbrace{r_i + \gamma \hat{Q}_{\text{target}}(s'_i, \pi_{\text{target}}(s'_i))}_{y_i} - \hat{Q}(s_i, a_i) \Big)^2$$

Outline

Deep deterministic policy gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2016):

- Model-free, off-policy actor critic
- Continuous action space
- Deterministic policy π
- Random exploration modifying π (e.g., additive zero mean Gaussian noise), easy in off-policy framework
- To stabilize learning with neural network:
  - Experience/replay buffer
  - Target network/policy "to give consistent targets during temporal difference backups"

Deterministic policy gradient

Because π is deterministic, we have

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

and consider

$$\nabla_{\boldsymbol{\theta}} V^\pi(s) = \nabla_{\boldsymbol{\theta}} Q^\pi(s, \pi(s)) \overset{\text{chain rule}}{=} \nabla_a \, Q^\pi(s, a)|_{a=\pi(s)} \, \nabla_{\boldsymbol{\theta}} \pi(s) \quad .$$

Changing the policy changes how often different states are visited (i.e., the state distribution). It is not obvious that this can be ignored, but it can (see Silver et al., 2014).

DDPG updates

Mini-batch: $\mathcal{S} = \left\{ \langle s_i, a_i, r_i, s'_i \rangle \,\middle|\, i = 1, \ldots, N \right\}$

Critic update: Gradient step descent following

$$\nabla_{\boldsymbol{w}} \frac{1}{N} \sum_{i=1}^{N} \Big(\underbrace{r_i + \gamma \hat{Q}_{\text{target}}(s'_i, \pi_{\text{target}}(s'_i))}_{y_i} - \hat{Q}(s_i, a_i) \Big)^2 =$$

$$- \frac{1}{2N} \sum_{i=1}^{N} \big(y_i - \hat{Q}(s_i, a_i) \big) \nabla_{\boldsymbol{\theta}} \hat{Q}(s_i, a_i)$$

Actor update: Gradient ascent following:

$$\frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} V^{\pi}(s) = \frac{1}{N} \sum_{i=1}^{N} \nabla_a Q^{\pi}(s, a)\big|_{s=s_i, a=\pi(s_i)} \nabla_{\boldsymbol{\theta}}\, \pi(s)\big|_{s=s_i}$$

Deep deterministic policy gradient (DDPG)

Algorithm 4: Deep Deterministic Policy Gradient

initial policy parameters $\boldsymbol{\theta}$, Q-function parameters $\boldsymbol{w}$, experience buffer M

$\boldsymbol{w}_{\text{target}} \leftarrow \boldsymbol{w}$, $\boldsymbol{\theta}_{\text{target}} \leftarrow \boldsymbol{\theta}$

repeat

  observe s, choose a based on π (plus exploration)

  take action a, observe r, s'

  store $\langle s, a, r, s' \rangle$ in M

  sample mini-batch $\left\{ \langle s_i, a_i, r_i, s'_i \rangle \,\middle|\, i = 1, \ldots \right\}$ from M

  set targets $y_i = r_i + \gamma \hat{Q}_{\text{target}}(s'_i, \pi_{\text{target}}(s'_i)))$

  update $\hat{Q}$ network weights $\boldsymbol{w}$

  update policy parameters $\boldsymbol{\theta}$

  update target networks: $\begin{array}{l} \boldsymbol{w}_{\text{target}} \leftarrow \rho \boldsymbol{w}_{\text{target}} + (1-\rho)\boldsymbol{w} \\ \boldsymbol{\theta}_{\text{target}} \leftarrow \rho \boldsymbol{\theta}_{\text{target}} + (1-\rho)\boldsymbol{\theta} \end{array}$

until *until stopping criterion is met*

Outline

Advantages

Goal is to maximize the expected (discounted) return:

$$J(\pi) = \mathbb{E}\left\{ \sum_{t=1}^{\infty} \gamma^{t-1} r_t \,\middle|\, \underbrace{s_0, \pi}_{\text{actions are generated following } \pi} \right\}$$

Advantage of doing a in state s (and following π afterwards) instead of following π:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

We can express the expected return of a policy π' in terms of its advantage over another policy π and $J(\pi)$:

$$J(\pi') = J(\pi) + \mathbb{E}\left\{ \sum_{t=0}^{\infty} \gamma^{t} \underbrace{A^{\pi}(s_t, a_t)}_{\text{advantage of following } \pi' \text{ instead of } \pi} \,\middle|\, s_0, \pi' \right\}$$

See Kakade & Langford (2002) or Schulman et al. (2017) for a proof.

Advantage estimation

We can estimate the advantage

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

from a sample s_t, a_t, r_{t+1} and approximated value function $\hat{V}^\pi$ by:

$$\hat{A}^\pi(s_t,a_t) = r_{t+1} + \gamma \hat{V}^\pi(s_{t+1}) - \hat{V}^\pi(s_t)$$

Adding more steps incorporates more information and reduces the influence of a badly estimated $\hat{V}^\pi$. Consider a (sub-)sequence $s_0, a_0, r_1, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots, s_T$ we can estimate the advantages:

$$\hat{A}^\pi(s_t,a_t) = -\hat{V}^\pi(s_t) + r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T + \gamma^{T-t} \hat{V}^\pi(s_T)$$

It is a common strategy (PPO, ...) to break down episodes in subepisodes of length T.

(Asynchronous) Advantage Actor-Critic

- Asynchronous Advantage Actor-Critic = A^{3}C
- Delayed critic update: $\hat{Q}$ and $\hat{Q}_{\text{target}}$ parameters $\boldsymbol{w}$ and $\boldsymbol{w}_{\text{target}}$
- Intitialization (not shown in algorithm on next page):
  - initial actor parameters $\boldsymbol{\theta}$
  - step counter $t \leftarrow 0$
  - initial value function weights $\boldsymbol{w} = \boldsymbol{w}_{\text{target}}$
  - accumulated gradient $\delta_{\boldsymbol{w}} \leftarrow 0$
  - observed initial state s
- Algorithms are called asynchronous when designed for parallel threads which update the policy asynchronously. We present the single-thread "synchronous" version A^{2}C.
- In our A^{2}C version, T denotes the maximum number of time steps in a subepisode and we accumulated over T_{update} subepisodes, which could run in different threads.
- We start by looking at one-step Q-learning A^{3}C style.

One-step Q-learning A^{3}C style

Algorithm 5: One-step Q-learning A^{3}C style (w/o initialization)

repeat
  take action a ϵ-greedy based on $\hat{Q}(s,a)$
  observe r, s'
  $$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_a \hat{Q}_{\text{target}}(s,a) & \text{otherwise} \end{cases}$$
  accumulate gradients: $\delta_{\boldsymbol{w}} \leftarrow \delta_{\boldsymbol{w}} + \nabla_{\boldsymbol{w}} (y - \hat{Q}(s,a))^2$
  $s \leftarrow s'$, $t \leftarrow t+1$
  if $t \mod T_{\text{target-update}} = 0$ **then**
    $\boldsymbol{w}_{\text{target}} \leftarrow \boldsymbol{w}$
  if $t \mod T_{\text{update}} = 0$ **then**
    update $\boldsymbol{w}$ based on $\delta_{\boldsymbol{w}}$
    $\delta_{\boldsymbol{w}} \leftarrow 0$
until *until stopping criterion is met*

Advantage Actor-Critic

Algorithm 6: A^{2}C

repeat
  $\delta_{\boldsymbol{w}} \leftarrow 0; \delta_{\boldsymbol{\theta}} \leftarrow 0$
  for $e = 1, \ldots, T_{\text{update}}$ **do**
    $t \leftarrow 0$, observe s_t
    repeat
      perform action a_t according to $\pi(a_t, s_t)$; observe r_{t+1} and s_{t+1}
      $t \leftarrow t + 1$
    until until s_t terminal or $t = T$
    $R \leftarrow 0$ if s_t terminal otherwise $R \leftarrow \hat{V}_{\text{target}}(s_t)$
    for $i = t, \ldots, 1$ **do**
      $R \leftarrow r_i + \gamma R$
      $\delta_{\boldsymbol{\theta}} \leftarrow \delta_{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \log \pi(a_{i-1}, s_{i-1})(R - \hat{V}(s_{i-1}))$ // $\rightarrow$ `REINFORCE`
      $\delta_{\boldsymbol{w}} \leftarrow \delta_{\boldsymbol{w}} + \nabla_{\boldsymbol{w}}(R - \hat{V}(s_{i-1}))^2$
  update $\boldsymbol{w}$ and $\boldsymbol{\theta}$ using $\delta_{\boldsymbol{w}}$ and $\delta_{\boldsymbol{\theta}}$
until *until stopping criterion is met*

Outline

Proximal Policy Optimization

- Proximal Policy Optimization (PPO) is a popular, simple deep RL method
- PPO is rather robust, works for discrete and continuous action spaces
- PPO was used in training ChatGPT
- Ingredients:
  - "Surrogate loss" function CPI loss
  - Clipping
  - Optimize n_{steps} steps using mini-batche drawn from experience buffer

CPI loss function: Rewriting return I

Recall

$$J(\pi') = J(\pi) + \mathbb{E}\left\{ \sum_{t=1}^{\infty} \gamma^{t-1} \underbrace{A^{\pi}(s_t, a_t)}_{\substack{\text{advantage of following} \\ \pi' \text{ instead of } \pi}} \,\middle|\, s_0, \pi' \right\}$$

and

$$\eta_{\gamma}^{\pi}(s) = \mathbb{E}_{s_0 \sim p_{\text{start}}} \left[\sum_{k=0}^{\infty} \gamma^k \Pr\{ s_0 \xrightarrow{k} s \,|\, \pi \} \right]$$

$$= \sum_{t=0}^{\infty} \gamma^t \Pr\{ s_t = s \,|\, \pi \}$$

($\mathbb{E}_{s_0 \sim p_{\text{start}}}[\Pr\{ s_0 \xrightarrow{0} s \,|\, \pi \}]$ is the probability of s being the start state.)

CPI loss function: Rewriting return II

$$
\begin{aligned}
J(\pi') &= J(\pi) + \mathbb{E}\left\{ \sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \,\middle|\, s_0, \pi' \right\} \\
&= J(\pi) + \sum_{t=0}^{\infty} \sum_{s} \mathsf{Pr}\{s_t = s \,|\, \pi'\} \sum_{a} \pi'(a, s) \gamma^t A^{\pi}(s, a) \\
&= J(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t \mathsf{Pr}\{s_t = s \,|\, \pi'\} \sum_{a} \pi'(a, s) A^{\pi}(s, a) \\
&= J(\pi) + \sum_{s} \eta_{\gamma}^{\pi'}(s) \sum_{a} \pi'(a, s) A^{\pi}(s, a)
\end{aligned}
$$

CPI loss function: Approximation I

The dependency on π' on the RHS makes

$$J(\pi') = J(\pi) + \sum_s \eta_\gamma^{\pi'}(s) \sum_a \pi'(a,s) A^\pi(s,a)$$

difficult to optimize. Thus, a local approximation is introduced:

$$J_\pi^{\text{CPI}}(\pi') = \underbrace{J(\pi)}_{\substack{\text{can be dropped} \\ \text{when optimizing}}} + \sum_s \eta_\gamma^{\pi}(s) \sum_a \pi'(a,s) A^\pi(s,a)$$

When optimizing $J_\pi^{\text{CPI}}(\pi')$ w.r.t. to the parameters of π', the $J(\pi)$ term can be dropped as it is independent of π'.

CPI loss function: Approximation II

Let $\boldsymbol{\theta}'$ be the parameter of the policy π'. So far, our objective reads:

$$\max_{\boldsymbol{\theta}'} \sum_s \eta_\gamma^\pi(s) \sum_a \pi'(a,s) A^\pi(s,a)$$

Now we

- Replace $\sum_s \eta_\gamma^\pi(s)[\dots]$ by $(1-\gamma)^{-1}\mathbb{E}_{s\sim\eta_\gamma^\pi}[\dots]$ and drop the constant, and
- Sample actions in a state s from a proposal distribution $q(a \mid S)$ ($\rightarrow$ importance weighting)

and get (Schulman et al., 2017):

$$\max_{\boldsymbol{\theta}'} \mathbb{E}_{s\sim\eta_\gamma^\pi} \mathbb{E}_{a\sim q} \left[\frac{\pi'(a,s)}{q(a \mid s)} A^\pi(s,a) \right]$$

CPI loss function: Approximation III

Now we

- Use $\pi(s, a)$ as proposal distribution $q(a \mid s)$ for the actions, and
- Approximate $\mathbb{E}_{s \sim \eta_\gamma^\pi}$ by states sampled along a (sub-)episode following π.

This gives the CPI (named after *conservative policy iteration*, Kakade & Langford, 2002) objective

$$\left[\frac{\pi'(a_t, s_t)}{\pi(a_t, s_t)} \hat{A}^\pi(s_t, a_t) \right]$$

maximized based on one or several (sub-)episodes $s_0, a_0, r_1, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots, s_T$, see slide 32.

Clipping

- In the iterative optimization of a policy, it is a problem if a single step leads to a too big update, because
  - A single step may be a very noisy approximation of the proper gradient direction; and
  - A too large step may take the parameters outside the region where the first-order approximation underlying a gradient update is reasonable.
- To constrain the change of the policy, the update step (typically the approximated gradient) can be constrained, e.g., by limiting the length of the update vector or clipping the components of the update vector.
- PPO uses a very particular way of clipping the objective.

PPO Clipping

PPO-Clip objective

$$\min\left(\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)}\hat{A}^{\pi}(s_t,a_t), \text{clip}\left(\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi}(s_t,a_t)\right)$$

with $\text{clip}(x,l,u) = \min(\max(x,l),u)$. Clipping active $\rightarrow$ gradient of clipping term w.r.t. $\boldsymbol{\theta}'$ zero.

If $\hat{A}^{\pi}(s_t,a_t) > 0$ optimization wants to increase $\pi'(a_t,s_t)$ and $\min\left(\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)}, 1+\epsilon\right)$ limits increase if already $\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)} > 1+\epsilon$.

If $\hat{A}^{\pi}(s_t,a_t) < 0$ optimization wants to decrease $\pi'(a_t,s_t)$ and $\max\left(\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)}, 1-\epsilon\right)$ limits decrease if already $\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)} < 1-\epsilon$.

The policy is updated if $\frac{\pi'(a_t,s_t)}{\pi(a_t,s_t)} \in [1-\epsilon, 1+\epsilon]$ or if the gradient direction does not point away from that interval.

PPO

Algorithm 7: PPO

init policy and value function approximator parameters $\boldsymbol{\theta}'$ and $\boldsymbol{w}$
repeat
  $M = \emptyset$ // experience buffer
  Gather experience // add samples $\langle s_t^e, a_t^e, p_t^e, \hat{R}_t^e, \hat{A}_t^e \rangle$ to M
  for $i = 1, \ldots, n_{\text{steps}}$ **do**
    Sample mini-batch B from M
    Optimize PPO-Clip objective // update $\boldsymbol{\theta}'$
    Fit value function // update $\boldsymbol{w}$
until *until stopping criterion is met*

PPO sampling

Procedure Gather experience

for $e = 1, \ldots, T_{\text{update}}$ **do**
  $t \leftarrow 0$; observe s_t^e
  repeat
    perform action $a_t^e \sim \pi'(., s_t^e)[= \pi(., s_t^e)]$; observe r_{t+1}^e and s_{t+1}^e
    $p_t^e = \pi'(a_t^e, s_t^e)$ // 'old' probabilities
    $t \leftarrow t + 1$
  until until $t = T$
  for $t = 0, \ldots, T - 1$ **do**
    $\hat{R}_t^e(s_t^e, a_t^e) = r_{t+1}^e + \gamma r_{t+2}^e + + \cdots + \gamma^{T-t-1} r_T^e + \gamma^{T-t} \hat{V}(s_T^e)$
    $\hat{A}_t^e(s_t^e, a_t^e) = \hat{R}_t^e(s_t^e, a_t^e) - \hat{V}(s_t^e)$
  store $\langle s_t^e, a_t^e, p_t^e, \hat{R}_t^e, \hat{A}_t^e \rangle$ in M

We assume each subepisode runs for T steps. Policy and value function use parameters $\boldsymbol{\theta}'$ and $\boldsymbol{w}$.

PPO-Clip optimization

Procedure Optimize PPO-Clip objective

1 Do gradient-based optimization following

$$\nabla_{\boldsymbol{\theta}'} \frac{1}{|B|} \sum_{\langle s_t^e, a_t^e, p_t^e, \hat{R}_t^e, \hat{A}_t^e \rangle \in B} \min \left(\frac{\pi'(a_t^e, s_t^e)}{p_t^e} \hat{A}_t^e, \right.$$
$$\left. \text{clip} \left(\frac{\pi'(a_t^e, s_t^e)}{p_t^e}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t^e \right)$$

where $\boldsymbol{\theta}'$ are the parameters of π'

This is one step of the optimization for one mini-batch B.

Fit value function

Procedure Optimize PPO-Clip objective

1 Do gradient-based optimization following

$$\nabla_{\boldsymbol{w}} \frac{1}{|B|} \sum_{\langle s_t^e, a_t^e, p_t^e, \hat{R}_t^e, \hat{A}_t^e \rangle \in B} \left(\hat{V}(s_t^e) - \hat{R}_t^e \right)^2$$

where $\boldsymbol{w}$ are the parameters of $\hat{V}$

This is one step of the optimization for one mini-batche B.

$\hat{V}$ and π' may share parameters (parts of the network), then the gradients can be weighted, added and optimized jointly.

Regularization

KL-Regularization To prevent too large updates based on one batch of data that make the learning unstable, the KL divergence

$$\mathrm{KL}[\pi(\cdot, s), \pi'(\cdot, s)]$$

can be estimated over the observed states in a batch and can serve as a constraint (Schulman et al., 2015) or can be added as a weighted penalty (to be minimized) to the objective function (PPO)

Entropy regularization To ensure exploration (for discrete action spaces), the weighted entropy of the actions can be added to the objective (to be maximized).

Deep RL: Stabilizing the learning

Deep RL requires mechanisms to stabilize the learning:

- Delayed Q update reduces the influence of a single observed step though averaging
- Advantage learning aims to reduce variance by introducing a covariate
- Clipping and KL regularization reduces the influence of a single step

References

M. Riedmiller. Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. European Conference on Machine Learning (ECML), LNAI 3720:317–328, 2005

S. Kakade, J. Langford. Approximately optimal approximate reinforcement learning. International Conference on Machine Learning (ICML), 2002

V. Mnih, K. Kavukcuoglu, D. Silver A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. Nature 518:529–533, 2015

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning International Conference on Machine Learning (ICML). PMLR 48:1928-1937, 2016

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic Policy Gradient Algorithms. International Conference on Machine Learning (ICML), 2014

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. International Conference on Learning Representations (ICLR), 2016

J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz. Trust region policy optimization. International Conference on Machine Learning (ICML), 2015

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal Policy Optimization Algorithms, 2017