Dynamic Programming and Reinforcement Learning

Week 7a: Policy Gradients (2)

Rainer Schlosser und Alexander Kastius

Enterprise Platform and Integration Concepts

27.05.21

REINFORCE

$$\nabla_\phi J(\phi) \propto E\left(\frac{\nabla_\phi \pi_\phi(a_t|s_t)}{\pi_\phi(a_t|s_t)} G_t\right)$$

Inserting this in our gradient ascent update, we get the REINFORCE update mechanism:

$$\phi_{t+1} = \phi_t + \eta \frac{\nabla_\phi \pi_{\phi_t}(a_t|s_t)}{\pi_{\phi_t}(a_t|s_t)} G_t$$

Computation of G_t can only be performed in the episodic case with a guaranteed terminal state.

REINFORCE

Input: Parametric non-deterministic policy $\pi_\phi(a \mid s)$ with initial parameters ϕ_0.

Repeatedly perform:

1. Go through whole episode, following our policy.
2. Compute G_t for all t
3. Apply $\phi_{t+1} = \phi_t + \eta \frac{\nabla_\phi \pi_{\phi_t}(a_t|s_t)}{\pi_{\phi_t}(a_t|s_t)} G_t$ for all tuples (s_t, a_t, G_t) to improve the policy.

There is no real stopping criterion, as there is no measure of performance by definition. Assuming the policy did not change for a long time, it can be seen as "optimal", but this is rarely going to happen.

Variance Issues of REINFORCE

Our current implementation of REINFORCE does suffer from a severe disadvantage: The variance of gradients is directly related to the variance of the rewards, which, depending on the process under assessment might be very high.

$$\nabla_\phi V(s_0) \propto \sum_{x \in S} \mu(x) \sum_{a \in A} \nabla_\phi \pi_\phi(a|x)\, Q^\pi(x,a)$$

The current formulation of the policy gradient theorem allows a small adjustment, which might help. We can introduce a state-dependent baseline to reduce the variance of the gradient:

$$\nabla_\phi V(s_0) \propto \sum_{x \in S} \mu(x) \sum_{a \in A} \nabla_\phi \pi_\phi(a|x)\, (Q^\pi(x,a) - b(x))$$

Introducing Baselines

$$\nabla_\phi V(s_0) \propto \sum_{x \in S} \mu(x) \sum_{a \in A} \nabla_\phi \pi_\phi(a|x) \left(Q^\pi(x,a) - b(x)\right)$$

Does this change the expected value of our gradient?

$$\sum_{a \in A} \nabla_\phi \pi_\phi(a|x) \left(Q^\pi(x,a) - b(x)\right) = \sum_{a \in A} \nabla_\phi \pi_\phi(a|x)\, Q^\pi(x,a) - \sum_{a \in A} \nabla_\phi \pi_\phi(a|x)\, b(x)$$

$$\sum_{a \in A} \nabla_\phi \pi_\phi(a|x)\, b(x) = b(x) \nabla_\phi \sum_{a \in A} \pi_\phi(a|x) = b(x) \nabla_\phi 1 = 0$$

No it doesn't. But in practice, the variance of the gradient will become smaller.

What can we use as $b(x)$?

REINFORCE with Baselines

What can we use as $b(x)$ given a state $x = s_t$?

We want to minimize the variance of the gradient, which means we want to choose a value that does not depend on a but is usually close to G_t.

This makes the value of x a natural choice.

$$\phi_{t+1} = \phi_t + \eta \frac{\nabla_\phi \pi_{\phi_t}(a_t|s_t)}{\pi_{\phi_t}(a_t|s_t)}(G_t - b(s_t)) \text{ with } b(s_t) = V(s_t)$$

Where do we get $V(s_t)$ from? We can't use G_t if we only see states rarely, as we would usually then compute $G_t - G_t = 0$. Instead, we use a parametric estimator for $V(s_t)$.

REINFORCE with Baselines

Required: A value estimator $v(s;\phi_v)$ with parameters ϕ_v, a parametric policy $\pi_{\phi_\pi}(a|s)$ with parameters ϕ_π.

1. Go through an episode, following our policy.
2. Compute G_t for all t in the episode.
3. Update the value estimator by minimizing $\left(G_t - v(s_t;\phi_v)\right)^2$ using gradient descent:

$$\phi_v = \eta_v \nabla_{\phi_v} \frac{1}{2}\left(G_t - v(s_t;\phi_v)\right)^2 + \phi_v$$

4. Update the policy by following the policy gradient theorem:

$$\phi_\pi \leftarrow \phi_\pi + \eta_\pi \frac{\nabla_{\phi_\pi}\pi_{\phi_\pi}(a_t|s_t)}{\pi_{\phi_\pi}(a_t|s_t)}\left(G_t - v(s_t;\phi_v)\right)$$

Are there any disadvantages of REINFORCE that are not solved yet?

Actor-Critic

REINFORCE requires full trajectories, which are not always available.

In value estimation, temporal difference learning allowed omitting this requirement and leads towards faster convergence. The same principle can be reapplied on REINFORCE.

Core ideas:

- Bootstrap $V(s_t; \phi_v)$ by replacing G_t with $r_t + V(s_{t+1}; \phi_v)$
- Bootstrap the target for the policy gradient by replacing G_t with $r_t + V(s_{t+1}; \phi_v)$.

Actor-Critic

Required: An estimator for $v(s;\phi_v)$ with parameters ϕ_v, a parametric policy $\pi_{\phi_\pi}(a|s)$ with parameters ϕ_π.

1. Go through an episode, following our policy.
2. At each step compute $\delta = r_t + v(s_{t+1};\phi_v) - v(s_t;\phi_v)$
3. Update the value estimator by minimizing δ^2 using gradient descent:

$$\phi_v = \eta_v \nabla_{\phi_v} \delta^2 + \phi_v$$

4. Update the policy by following the policy gradient theorem:

$$\phi_\pi \leftarrow \phi_\pi + \eta_\pi \frac{\nabla_{\phi_\pi} \pi_{\phi_\pi}(a_t|s_t)}{\pi_{\phi_\pi}(a_t|s_t)} \delta$$

Reintroducing Discounting

All methods presented here, assumed $\gamma = 1$ for simplicity. The policy gradient as well as the updates change slightly, if this is not the case.

Required: An estimator for $v(s; \phi_v)$ with parameters ϕ_v, a parametric policy $\pi_{\phi_\pi}(a|s)$ with parameters ϕ_π.

1. Go through an episode, following our policy.
2. At each step compute $\delta = r_t + \gamma v(s_{t+1}; \phi_v) - v(s_t; \phi_v)$
3. Update the value estimator by minimizing δ^2 using gradient descent:

$$\phi_v = \eta_v \nabla_{\phi_v} \delta^2 + \phi_v$$

4. Update the policy by following the policy gradient theorem:

$$\phi_\pi \leftarrow \phi_\pi + \eta_\pi \frac{\nabla_{\phi_\pi} \pi_{\phi_\pi}(a_t|s_t)}{\pi_{\phi_\pi}(a_t|s_t)} \delta$$

Review: A bunch of hyper-parameters

All previously mentioned methods do require a noticeable amount of hyper-parameter tuning due to their complex setups, this is the case on for AC-methods as well:

- V and π and their internal setup, which includes the number of layers, their width, internal adjustment mechanisms like batch-normalization and regularization tools like dropout which all might work or not, depending on the problem under assessment. Many times it is also advantageous to share some early layers of the policy and value network.
- Learning rates η_π and η_v/η_q (if there is a Q-estimator as well). Technically more advanced optimizers like ADAM have even more parameters.
- Discount factor γ
- Parameters α and β for PER if applicable
- Exploration rate ϵ if applicable
- The decay rates for all parameters where applicable.

Continuing Development

The perfect AC-algorithm has yet to be found. But: There are many different directions of research right now and at least 5 to 10 prominent alternatives available:

Stable Baselines, a set of reference implementations, incorporates the following set in addition to DQN:

- DDPG (Deep Determinist Policy Gradient)
- A2C (Advantage Actor Critic)
- PPO (Proximal Policy Optimization)
- **SAC (Soft Actor Critic)**
- TD3 (Twin Delayed DDPG)

All of them incorporate one or several additions to naïve actor-critic.

A Final ”Case Study” – Soft Actor Critic

A Final Look at Soft Actor Critic:

SAC implements at least three additions to the AC algorithm as we’ve developed it here:

- It uses an off-policy approach for the baseline instead of policy related value estimates and optimizes the policy based on Q-value estimates, which in turn makes the whole approach work on off-policy generated samples.
- It reintroduces a second network for value estimation, which resembles the double network trick known from DQN and other algorithms.
- It adds entropy regularization.

Check https://arxiv.org/pdf/1801.01290.pdf for original paper

Off-Policy Actor Critic

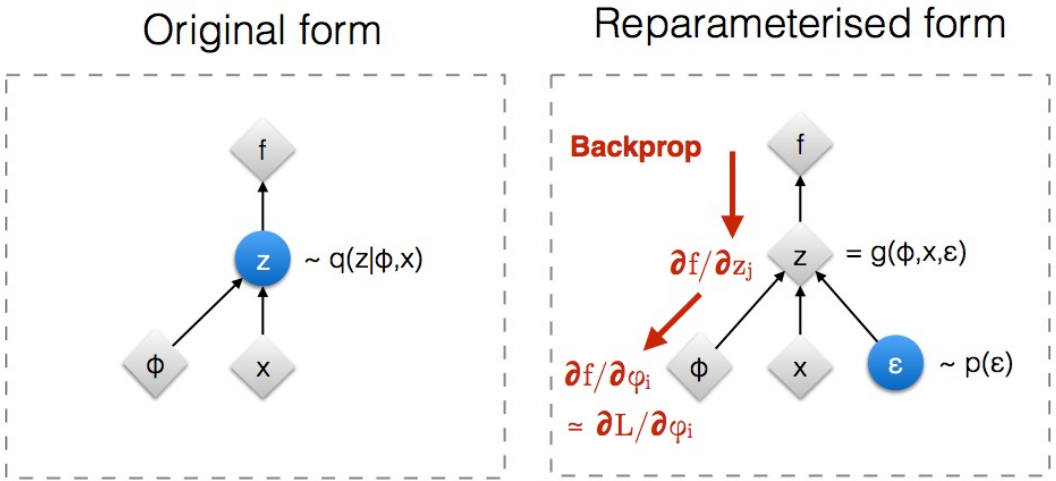
$$J_\pi(\phi_\pi) = E\left(D_{KL}\left(\pi_{\phi_\pi}(a_t|s_t)\middle|\frac{e^{Q(s_t,a_t;\phi_Q)}}{Z(s_t;\phi_Q)}\right)\right)$$

The goal is to minimize the KL-divergence between the policy and the policy induced by the Q-value estimation. This allows off-policy training, as we can compute this difference using only our estimation and the given policy.

Z is the normalizing constant, which would usually be something like the sum over $e^{Q(s,a;\phi_Q)}$ for all a. It is excluded here, because it does not show up in the overall gradient.

There is a problem with this equation: To derive a gradient, during backpropagation we would need to compute a gradient on a random variable a_t, which is sampled from the policy. We can't easily do this.

Reparametrization Trick



[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

https://stats.stackexchange.com/questions/199605/how-does-the-reparameterization-trick-for-vaes-work-and-why-is-it-important

Reparameterization for SAC

$$J_\pi(\phi_\pi) = E\left(D_{KL}\left(\pi_{\phi_\pi}(a_t|s_t)\middle|\frac{e^{Q(s_t,a_t;\phi_Q)}}{Z(s_t;\phi_Q)}\right)\right)$$

Solution: We reparametrize the policy. Instead of sampling from the policy, we determine a by sampling from a gaussian dist. feeding the sample in our neural network as input and use the output as action.

$$a_t = f(\epsilon_t; s_t, \phi_\pi)$$

$$J_\pi(\phi_\pi) = E\left(D_{KL}\left(\pi_{\phi_\pi}(f(\epsilon_t; s_t, \phi_\pi)|s_t)\middle|\frac{e^{Q(s_t,f(\epsilon_t;s_t,\phi_\pi);\phi_Q)}}{Z(s_t;\phi_Q)}\right)\right)$$

This can be backpropagated, and the gradient becomes:

$$\nabla_{\phi_\pi} J_\pi(\phi_\pi) = \nabla_{\phi_\pi} \log \pi_{\phi_\pi}(a_t|s_t) + \left(\nabla_a \log \pi_{\phi_\pi}(a_t|s_t) - \nabla_a Q\left(s_t, a_t; \phi_Q\right)\right)\nabla_{\phi_\pi} f_{\phi_\pi}(\epsilon_t; s_t)$$

Double Q-Network in SAC

- Learning Q in SAC works similar to the way we learned Q before:

$$J_Q(\phi_Q) = E\left(\frac{1}{2}\left(Q(s_t, a_t; \phi_Q) - E(r_t + \gamma V(s_{t+1}; \phi_V))\right)^2\right)$$

- In its first iteration, SAC included a third value estimator $V(s_{t+1}; \phi_V)$, to be used in this equation. In later iterations this was replaced by computing the value from a combination of the policy using f and Q.
- Double network are introduced by keeping to different estimators Q with different parameters and using the minimum of both, whenever a computation requires an estimate.

Entropy Regularization in Values

Last but not least, the probably most important trick of SAC:

We introduce an entropy regularization factor. Regularization usually means that an ML model has a special mechanism to avoid overfitting. In case of RL means: We want to avoid to immediately converge towards near deterministic policies, which do not explore enough.

Entropy can be seen as measure of ”unexpectedness” in a probability distribution, usually relates to information theory. A probability distribution that assigns equal probability to all possible choices of the has very high entropy, a deterministic distribution (that assigns a single option probability 1) has a very low entropy. Given n elements $x_i \in X$, the entropy of a distribution P over support X with probabilities $P(x_i)$ is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i)$$

Entropy Regularization in SAC

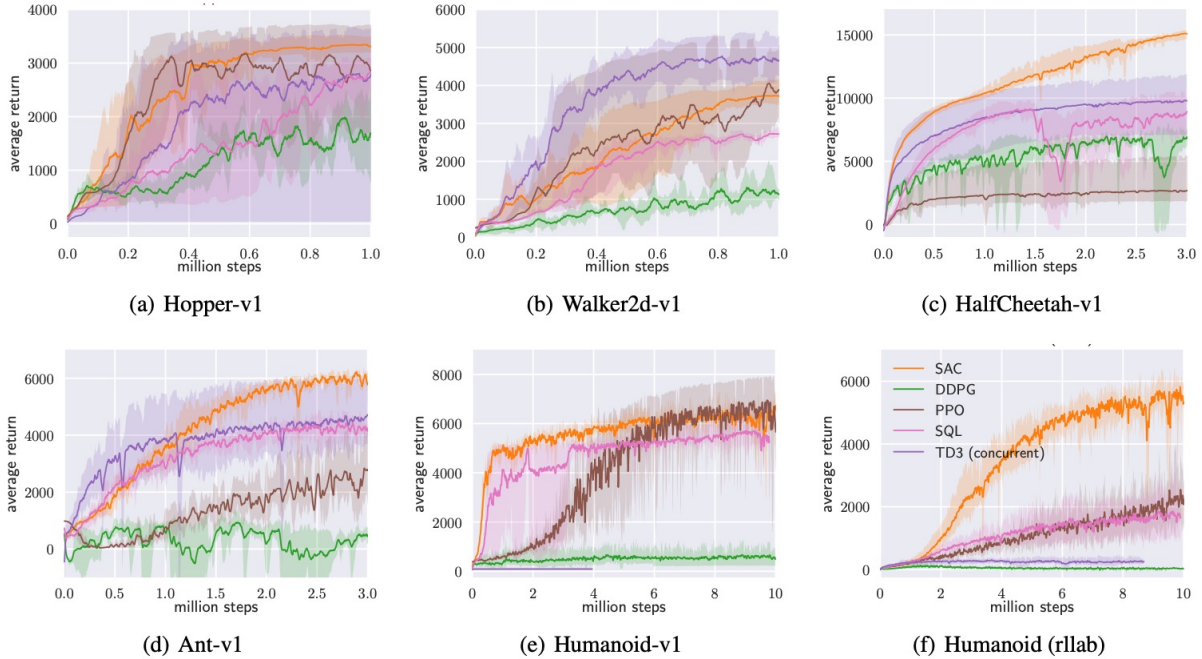
We introduce entropy to SAC, by adding it to the value of a state and by this, also to the values of (s, a) pairs during Q-estimation:

$$J_V(\phi_V) = E\left(\frac{1}{2}\left(V(s_t;\phi_V) - E_{a_t\sim\pi_{\phi_\pi}(a_t|s_t)}\left(Q\left(s_t, a_t;\phi_Q\right) - \log\pi_{\phi_\pi}(a_t|s_t)\right)\right)^2\right)$$

When valuing a state, a bonus is given if the policy is very unpredictable at this point. This leads to two effects:

- The policy itself stays non-deterministic, unless a future state is of very high value and thus, the Q-value is dominated by the value estimation as well
- The value estimation is higher for states which we know less about, e.g. which the policy is not already focused on a single action.
- Both overall massively increase SAC's performance.

Is SAC better than the alternatives?



(a) Hopper-v1 (b) Walker2d-v1 (c) HalfCheetah-v1

(d) Ant-v1 (e) Humanoid-v1 (f) Humanoid (rllab)

https://arxiv.org/pdf/1801.01290.pdf

Schedule

- 3.5.: Problems with Q-Learning
- 10.5.: Deep Q-Networks
- 17.5.: DQN (2) / RAINBOW
-
- **31.5.: Project Assignments**
- 7.6.: Project Work/Support
- 14.6.: Project Work/Support
- 21.6.: Project Work/Support
- 28.6.: Project Work/Support
- 5.7.: Project Work/Support
- 12.7.: Final Presentations
- 31.8.: Final Documentation

6.5.: Artificial Neural Networks

20.5.: Policy Gradients (1)

27.5.: Policy Gradients (2)

3.6.: Project Assignments

10.6.: Project Work/Support

17.6.: Project Work/Support

24.6.: Project Work/Support

31.6.: Project Work/Support

8.7.: Project Work/Support

15.7.: Final Presentations

Project Assignments

- The project aims at understanding the impact of the different design choices in the algorithms to extend the more theoretical knowledge from the first half of the course. So usually, the focus should be put on the algorithm and not on the problem you want to solve with it. Choosing a problem of your choice to solve is still up to you, to allow you to choose one that shows the advantages and disadvantages of the method you're working on.
- We will propose several topics next week.
- But: **Feel free to bring up your own idea.**
- We cannot guarantee, that everything will be possible within half a semester with limited hardware (a full evaluation run of RAINBOW on the whole Atari suite takes about 10 days on specialized hardware…)
- Regarding the assignment: **Please contact me if your DQN setup is not working at all.** There are quite a lot of non-deterministic components involved which are sometimes hard to debug. We can do sanity checks on your codebase :)