Practical Deep Reinforcement Learning

Nikolaos Passalis

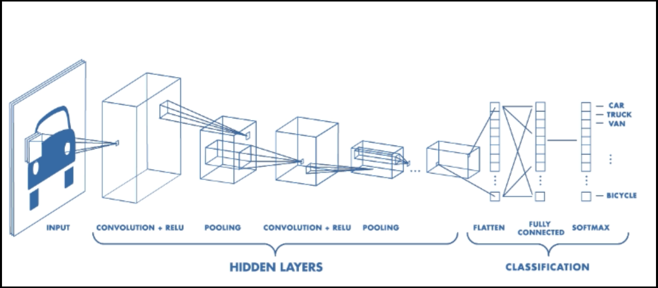
Postdoctoral researcher
Department of Signal Processing
Tampere University of Technology, Finland

Plan

1. Introduction to Deep Reinforcement Learning
2. Quick Theory Recap/Intro
   - 2.1 (Deep) Policy Gradients
   - 2.2 (Deep) Q-learning
   - 2.3 ~~(Deep) Actor-Critic~~
3. Getting Practical
   - 3.1 Introduction to PyTorch
   - 3.2 Implementing Deep Policy Gradient and variants
   - 3.3 Implementing Deep Q-learning and variants
   - 3.4 ~~Implementing Actor-Critic models~~
4. ~~Using Deep RL agents to do my homework~~

Deep Learning

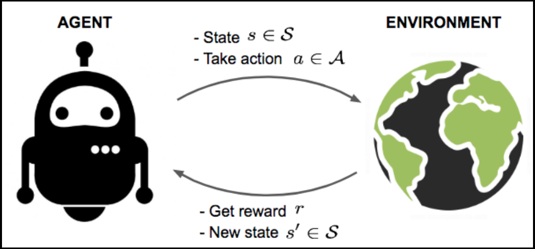
- Provides useful tools (neural networks) for classifying and **extracting representations** from *unstructured* data



- Does not provide a way to take intelligent decisions (this must be coded on top of DL)
- DL: “A car is headed toward me!” (do not know that to do, so this is probably fine)

Reinforcement Learning

- Provides useful tools for **learning how to behave** to achieve certain goals that will maximize a reward


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

- Cannot deal with unstructured environments
- RL: “If I knew that a car is headed toward me (but I don’t), I could try to avoid it!”

Deep Learning and Reinforcement Learning

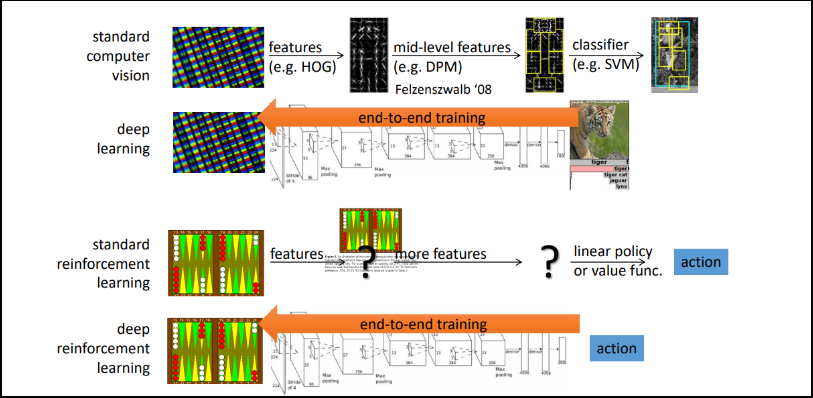
- Two separate approaches?

Deep Reinforcement Learning

- … or one unified approach combining the strengths of both of them?

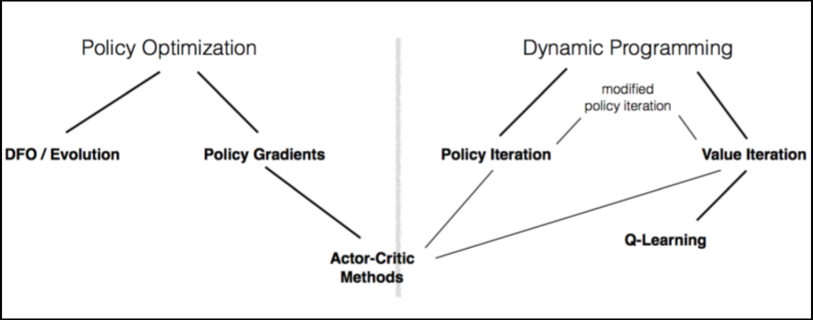
Deep Reinforcement Learning

- History is repeating itself!



Quick introduction to RL theory

RL landscape



Deep Reinforcement Learning

1. Deep Q Network (DQN)
2. Deep Double Q-Learning (DDQN)
3. Dueling Networks for Deep RL
4. Priotized DDQN
5. Distributional Q-learning
6. Noisy DQN
7. Rainbow
8. Advantage Actor-Critic (A2C)
9. Asynchronous Actor-Critic Agents (A3C)
10. Anticipatory Asynchronous Advantage Actor-Critic (A4C)
11. Deep Deterministic Policy Gradient (DDPG)
12. Distributed Distributional DDPG (D4PG)
13. Multi-agent DDPG (MADDPG)
14. Actor-Critic with Experience Replay (ACER)
15. Proximal Policy Optimization (PPO)
16. Soft Actor-Critic (SAC)
17. Twin Delayed Deep Deterministic (TD3)

Policy-gradient based methods

- Advantages
    - Directly optimize a policy (network) according to our goal
    - Can easily deal with continuous action spaces
    - Easier to work with more complex architectures
- … but ..
    - (usually) **on-policy**
    - less sample-efficient (not the same as wall time)
    - the vanilla versions are usually quite unstable

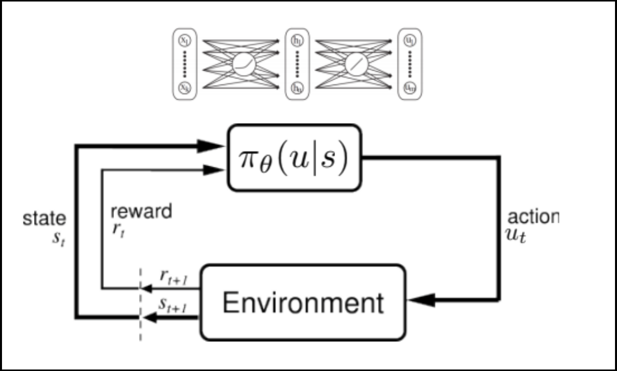
Q-learning based methods

- Advantages
    - More sample-efficient
    - Can easily work **off-policy**
    - More compatible with exploration
- … but …
    - Unstable when combined with non-linear approximation functions
    - May require many tricks to work with DL models

Policy Gradient

Policy Gradients

- Most intuitive approach!
- Main idea: Directly learn a *policy* that allows for taking decisions that maximize the obtained reward



Policy Gradient

- Let τ be a state-action sequence (path or trajectory) $s_0, u_0, \ldots, s_H, u_H$
- The total reward obtained in this sequence is denoted by

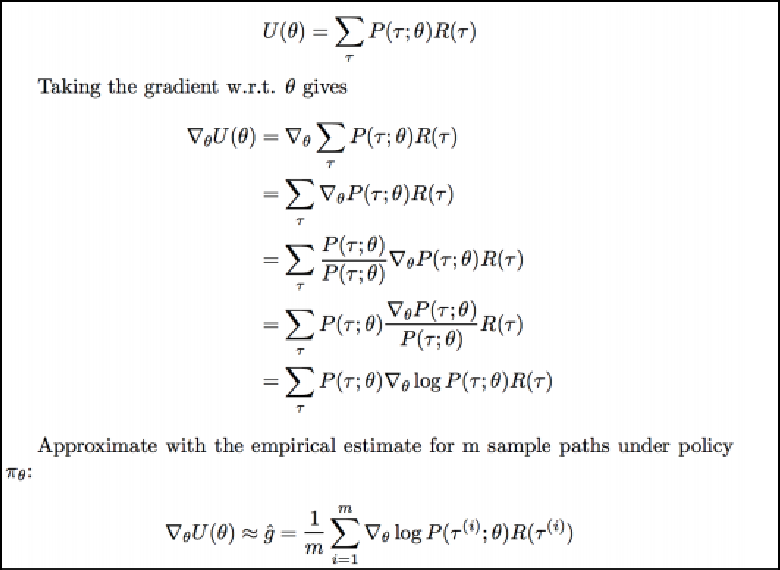
$$R(\tau) = \sum_{t=0}^{H} R(s_t, u_t),$$

where $R(s_t, u_t)$ is the reward obtained when performing the action u_t from the state s_t
- How useful is this policy?

$$U(\theta) = E[R(\tau)|\pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

where π_θ is the policy parameterized by θ

The nice thing



$$U(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau)$$

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

The nice thing

- The gradient of $U(\theta)$ has a really convenient form
- Can be directly used to learn a policy

The nice thing

- The gradient of $U(\theta)$ has a really convenient form
- Can be directly used to learn a policy
- Is it clear why?

The nice thing

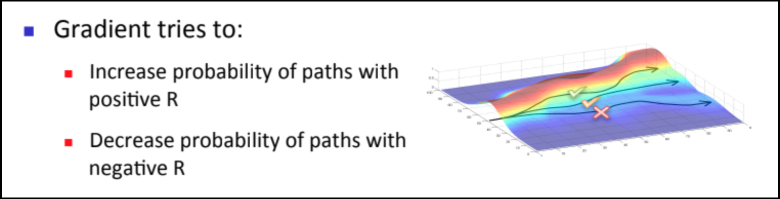
- The gradient of $U(\theta)$ has a really convenient form
- Can be directly used to learn a policy
- Is it clear why?

$$\nabla U(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla \log P(\tau^{(i)};\theta) R(\tau^{i}) \tag{1}$$

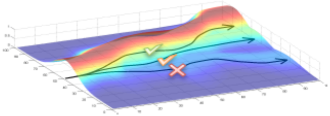
where we sampled m paths and τ^{i} is the i-th path

The nice thing

- Intuition: Change the policy to increase the probability of paths that lead to higher rewards



- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R



The REALLY nice thing

- How do we compute $\nabla \log P(\tau^{(i)}; \theta)$?

The REALLY nice thing

- How do we compute $\nabla \log P(\tau^{(i)}; \theta)$?

$$
\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}
\end{aligned}
$$

What do we implement?

$$\nabla U(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{H} \nabla log \left(\pi_\theta(u_t^{(i)} | s_t^{(i)}) \right) \right) R(\tau^i), \qquad (2)$$

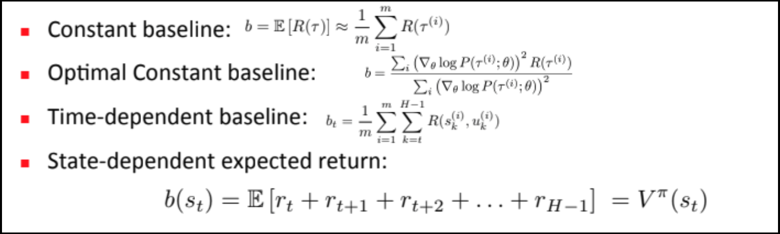
where $\pi_\theta(u_t^{(i)}|s_t^{(i)})$ is just the output of the neural network (and the employed DL framework takes care of the gradient $\nabla\pi_\theta$)

Reducing variance and improving convergence

- Introducing a baseline b (and removing terms that do not depend on the current action) can lower the variance

$$\nabla U(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{H}\nabla log\left(\pi_\theta(u_t^{(i)}|s_t^{(i)})\right)\right)\left(R(\tau^{i}) - b\right), \qquad (3)$$

- Some baseline choices:



- Constant baseline: $b = \mathbb{E}[R(\tau)] \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)})$
- Optimal Constant baseline: $b = \frac{\sum_i \left(\nabla_\theta \log P(\tau^{(i)};\theta)\right)^2 R(\tau^{(i)})}{\sum_i \left(\nabla_\theta \log P(\tau^{(i)};\theta)\right)^2}$
- Time-dependent baseline: $b_t = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)})$
- State-dependent expected return:

$$b(s_t) = \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \ldots + r_{H-1}\right] = V^{\pi}(s_t)$$

Q-learning

Q-learning

- A completely different approach
- Does not directly learn the policy
- Learn to estimate the (expected) *future total reward* that we can earn after performing a specific action and then continuing optimally (Q-value)
- **If such estimation exists, then the optimal policy is to select the action with the highest Q-value**

Q-learning

- A completely different approach
- Does not directly learn the policy
- Learn to estimate the (expected) *future total reward* that we can earn after performing a specific action and then continuing optimally (Q-value)
- **If such estimation exists, then the optimal policy is to select the action with the highest Q-value**
- Is it clear why?

How to find the Q-values?

- They can be iteratively estimated by observing the behavior of the agent
- Let $Q(s_t, u_t)$ is the current estimation of the Q-value when selecting the action u_t from state s_t
- After performing the action we can refine the estimation using the reward r_t obtained by the selected action:

$$Q(s_t, a_t) \leftarrow (1 - a)Q(s_t, a_t) + a * (r_t + \gamma max_a Q(s_{t+1}, a)) \tag{4}$$

where a is the learning rate, γ the discount factor and $max_a Q(s_{t+1}, a)$ the estimation of the optimal future value

How to find the Q-values?

- By repeatedly applying the previous equation we can successfully learn the correct Q-value (this is guaranteed!!!)
- All problems solved?
    - Requires a table of size $N_{actions} \times N_{states}$

How to find the Q-values?

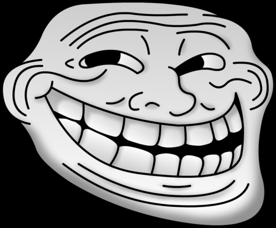
- By repeatedly applying the previous equation we can successfully learn the correct Q-value (this is guaranteed!!!)
- All problems solved?
    - Requires a table of size $N_{actions} \times N_{states}$
    - For an RGB image input (state) of size 4×4 there are more than 10^{100} possible states (there are $\approx 10^{86}$ particles in the visible universe)
    - Cannot easily handle continuous action spaces (many control tasks)

Solution?

- Use deep neural networks to estimate the Q-value of each action for a given state
- This allows also for exploiting the *generalization* abilities of neural networks, i.e., generalizing the encoded knowledge for states that were not seen before
- So, all problems solved?

Solution?

- Use deep neural networks to estimate the Q-value of each action for a given state
- This allows also for exploiting the *generalization* abilities of neural networks, i.e., generalizing the encoded knowledge for states that were not seen before
- So, all problems solved?
- Q-learning does not (always) work with non-linear approximators



Why?

- We are chasing a non-stationary target ($Q(s, u)$ drifts)!
- The data are correlated (successive states are strongly correlated)
- Two useful tricks allowed for training Deep Q-networks
    - Experience replay
    - Use an older copy of the network to compute the target Q-values

Deep Q-learning

- How to we train the network?
- Imitating supervised learning:

$$min||(r_t + \gamma max_a Q(s_{t+1}, a)) - Q(s_t, a_t)||_2^2 \tag{5}$$

- The data are correlated!!!
- Supervised learning analogous: Sort the data by their labels and try training a network!
- Experience Replay
  - Store the previous paths and use them when updating the network
  - The data in each batch are no longer correlated
  - Significantly improves the stability

Deep Q-learning

- Target Network
  - Use the Q-value obtained by an older copy of the Q-network
  - Allows for breaking feedback loops
  - Keeps the network from changing too quickly

$$min||(r_t + \gamma max_a Q_{target}(s_{t+1}, a)) - Q(s_t, a_t)||_2^2 \tag{6}$$

Thank you!
Questions?

Acknowledgments

No image used in this presentation belongs to me (thanks to Google image search) and used resources from various internet sources. Excellent course material (on which this presentation was based) are provided at

- https://sites.google.com/view/deep-rl-bootcamp/lectures
- http://rail.eecs.berkeley.edu/deeprlcourse/