

Deep RL

Multi-agent Learning

Overview of the course

1. **Learning to make decisions** in bandit problems; exploration vs exploitation; learning action values; greedy and ϵ -greedy; policy gradient for bandits; UCB
2. **Sequential decision problems**; MDPs; planning with dynamic programming; policy evaluation + policy improvement = policy iteration
3. **Model-free prediction and control**; Monte Carlo returns; TD learning; on-policy; off-policy; Q-learning; Sarsa; Double Q-learning
4. **Function approximation and deep RL**; tabular vs linear vs non-linear; convergence and divergence; least-squares prediction (LSTD and LSMC); multi-step returns; neural Q-learning; DQN
5. **Policy gradients and actor-critic methods**; REINFORCE; advantage actor-critics (A2C); trust-region methods; continuous actions; CACLA; gradient ascent on the value (DPG)
6. **Learning from a model**; Full models vs expectation models vs stochastic (generative) models; Dyna; parametric vs non-parametric models; experience replay; search; MCTS

Advanced topics and active research

- ▶ The main question is: **how do we maximize future rewards**
- ▶ Some main sub-questions are:
 - ▶ What do we learn? (Predictions, models, policies, ...)
 - ▶ How do we learn it? (TD, planning, ...)
 - ▶ How do we represent the learnt knowledge? (deep networks, sample buffers, ...)
 - ▶ How do we use the learnt knowledge?
- ▶ Specific active research topics include:
 - ▶ **Exploration** in the full sequential, function approximation case
 - ▶ **Credit assignment** with very delayed rewards
 - ▶ **Planning** with partial or **inaccurate models**
 - ▶ **Sample efficient** learning
 - ▶ **Appropriate generalization** (e.g., fast learning in new situations)
 - ▶ Building a useful, general, and information-rich **agent state**

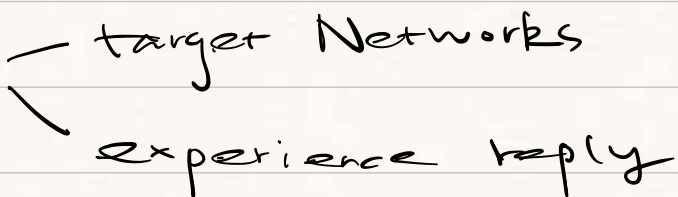
to answer
and for instance we could

Maximize future rewards!

Case study: rainbow DQN (Hessel et al. 2018)

- ▶ Investigation of several algorithm components
- ▶ The starting point was DQN, with **target networks** and **experience replay**
- ▶ The components were:
 - ▶ **Double Q-learning**
 - ▶ **Prioritized replay**
 - ▶ **Splitting values from advantages ('dueling network architectures')**
 - ▶ **Multi-step updates**
 - ▶ **Distributional reinforcement learning**
 - ▶ **Parameter noise for exploration ('noisy networks')**
- ▶ We combined all components, and looked at performance

want us to keep the nice rainbow color
going there so analogies for that a

DQN with 

Double Q-learning

Domain: Arcade Learning Environment

(Bellemare 2013)

Atari games from the ALE as benchmark

"general learning algorithm"

Start Pointing: DQN

1. CNN $Q_\theta : \mathcal{S}_t \rightarrow \mathbb{R}^M$ for M actions
2. ϵ -greedy policy: π_t
3. replay buffer for experience replay
4. target network parameters θ^- ($\theta_0^- = \theta_0$)
5. Q-learning loss function on θ (uses replay and target network)

$$L(\theta) = \frac{1}{2} \left(R_{i+1} + \gamma \left[\max_a Q_{\theta^-}(S_{i+1}, a) \right] - Q_{\theta}(S_i, A_i) \right)^2$$

6. Optimizer (SGD / RMSprop / Adam)
7. Update $\theta_t^- \leftarrow \theta_t$ occasionally
(e.g. every 10000 steps, $\theta_t^- = \theta_{t-1}^-$)

Double Q-learning

Loss function:

$$I(\theta) = \frac{1}{2} \left(R_{i+1} + V[\bar{Q}_\theta - (S_{i+1}, \arg\max_a \bar{Q}_\theta(S_{i+1}, a))] - \bar{Q}_\theta(S_i, A_i) \right)^2$$

Prioritized Replay

DQN samples uniformly from replay

Idea: prioritize transitions on which we can learn much

$$\text{priority of sample } i = |\delta_i|$$

δ_i : TD error on the last this transition was sampled

Dueling Networks (Wang 2016)

$$q_{\theta}(s, a) = V_{\xi}(s) + \underbrace{A_{\chi}(s, a)}$$

$$\theta = \xi \cup \chi$$

↑
advantage
for taking action a

Multi-step updates (Sutton 1988)

TD target look n steps in the future

" n -step" return:

$$n=1 \quad (\text{TD}) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

$$n=2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

⋮

$$n=\infty \quad (\text{MC}) \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$$

n-step return:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

n-step temporal-difference learning:

$$v(S_{t+1}) \leftarrow v(S_t) + \alpha (G_t^{(n)} - v(S_t))$$

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n q_{\theta} - (S_{t+1}, \arg \max_a q_{\theta}(S_{t+1}, a))$$

Double Q bootstrap target

Multi-step Q-learning:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha (G_t^{(n)} - q(S_t, A_t))$$

Distributional RL

expected cumulative rewards



distribution of returns

Categorical DQN (Bellemare 2017)

Distributional reinforcement learning

1. Find max action:

$$a^* = \underset{a}{\operatorname{argmax}} \mathbf{z}^\top \mathbf{p}_\theta(S_{t+1}, a)$$

(use, e.g., θ^- for double Q)

2. Update support:

$$\mathbf{z}' = R_{t+1} + \gamma \mathbf{z}$$

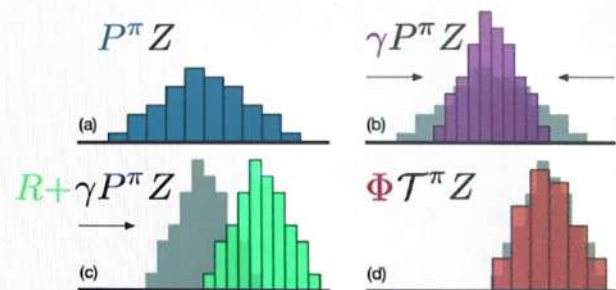
3. Project distribution $(\mathbf{z}', \mathbf{p}_\theta(S_{t+1}, a^*))$ onto support \mathbf{z}

$$d' = (\mathbf{z}, \mathbf{p}') = \Pi(\mathbf{z}', \mathbf{p}_\theta(S_{t+1}, a^*))$$

where Π denotes projection

4. Minimize divergence

$$\text{KL}(d' \| d) = - \sum_i p'_i \frac{\log p'_i}{\log p_\theta^i(S_t, A_t)}$$



Bottom-right: target distribution

$\Pi(R_{t+1} + \gamma \mathbf{z}, \mathbf{p}_\theta(S_{t+1}, a^*))$

Update $\mathbf{p}_\theta(S_t, A_t)$ towards this

Noisy Networks (Furukawa 2017)

DoN \rightarrow ϵ -greedy exploration

Idea:

Add noise to parameters

replace all linear operations

$$y = Wx + b$$

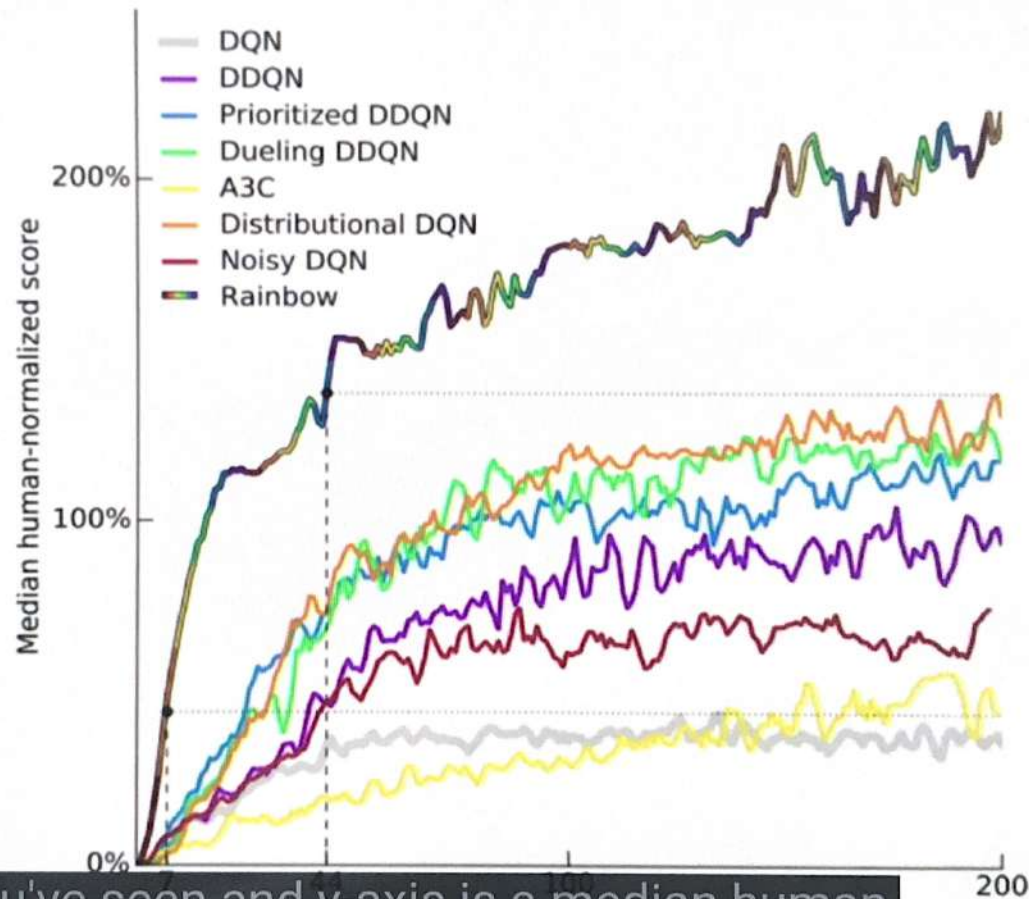
with

$$y = Wx + b + (W' \times \epsilon^W) + b' \times \epsilon^b$$

" \times " : element-wise product

ϵ^W, ϵ^b : random matrix / vector

Rainbow DQN: results



you've seen and y-axis is a median human normalized score which means

Rainbow DQN: conclusions

- ▶ Components work well together
- ▶ Most important: prioritising replay, multi-step returns
- ▶ Least important: double, dueling
- ▶ No wild overestimations due to fixed bounded support of value distribution
- ▶ But this requires knowing appropriate range...
- ▶ ...but different game have different score ranges
- ▶ This is possible due to **reward clipping**: in DQN rewards are clipped to $[-1, 1]$
- ▶ Makes learning easier, but changes the objective...

Adaptive normalization of updates

Normalize
targets \longrightarrow update

Adaptive target normalization (van Hasselt et al. 2016)

1. Observe target, e.g., $T_{t+1} = R_{t+1} + \gamma \max_a q_\theta(S_{t+1}, a)$
2. Update normalization parameters:

$$\begin{aligned}\mu_{t+1} &= \mu_t + \eta(T_{t+1} - \mu_t) && \text{(first moment / mean)} \\ \nu_{t+1} &= \nu_t + \eta(T_{t+1}^2 - \nu_t) && \text{(second moment)} \\ \sigma_{t+1} &= \nu_t - \mu_t^2 && \text{(variance)}\end{aligned}$$

where η is a step size (e.g., $\eta = 0.001$)

3. Network outputs $\tilde{q}_\theta(s, a)$, update with

$$\Delta\theta_t \propto \left(\frac{T_{t+1} - \mu_{t+1}}{\sigma_{t+1}} - \tilde{q}_\theta(S_t, A_t) \right) \nabla_\theta \tilde{q}_\theta(S_t, A_t)$$

4. Recover **unnormalized** value: $q_\theta(s, a) = \sigma_t \tilde{q}_\theta(s, a) + \mu_t$ (used for bootstrapping)

Preserve outputs

- ▶ Naive implementation changes **all outputs** whenever we update the normalization
- ▶ This seems bad: we should avoid updating values of unrelated states
- ▶ We can avoid this. Typically:

$$\tilde{\mathbf{q}}_{\mathbf{W}, \mathbf{b}, \theta}(s) = \mathbf{W} \phi_{\theta}(s) + \mathbf{b}.$$

- ▶ Idea: define

$$\mathbf{W}'_t = \frac{\sigma_t}{\sigma_{t+1}} \mathbf{W} \qquad \mathbf{b}'_t = \frac{\sigma_t \mathbf{b}_t + \mu_t - \mu_{t+1}}{\sigma_{t+1}}$$

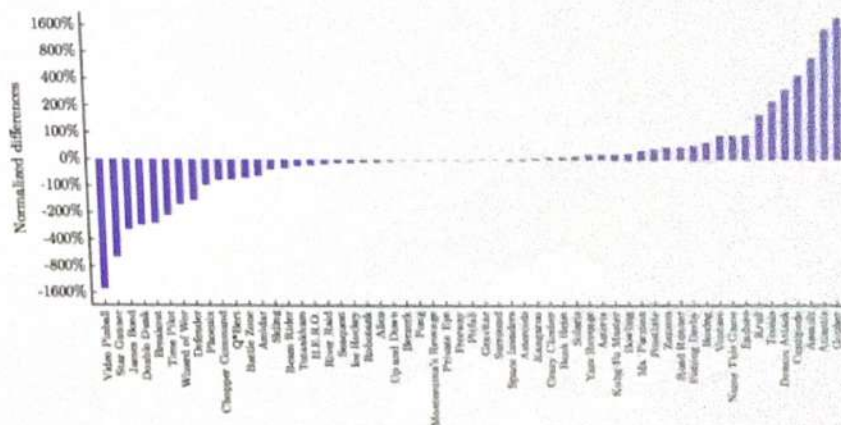
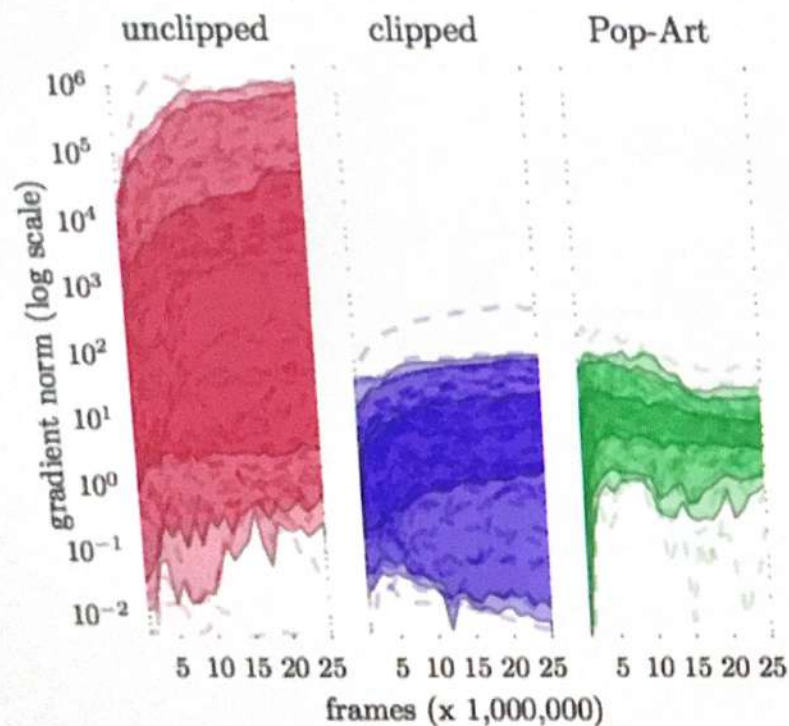
Then

$$\sigma_{t+1} \tilde{\mathbf{q}}_{\mathbf{W}'_t, \mathbf{b}'_t, \theta_t}(s) + \mu_{t+1} = \sigma_t \tilde{\mathbf{q}}_{\mathbf{W}_t, \mathbf{b}_t, \theta_t}(s) + \mu_t$$

- ▶ Then update \mathbf{W}'_t , \mathbf{b}'_t and θ_t as normal (e.g., SGD)

Preserve outputs

- Preserve Outputs Precisely, while Adaptively Rescaling Targets: Pop-Art



Universal value function approximation

Idea: feed a representation of $\langle \mathcal{L}, \mathcal{V} \rangle$ as input

Allows generalization across goals / tasks within an environment.

"Unicorn" (Mankowitz 2018)

GUF and models