Lecture 8: Advanced topics

Hado van Hasselt

UCL 2018

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

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

Domain: Arcade Learning Environment (Bellemare et al. 2013)

- We use Atari games from the ALE as benchmark
  - Diverse set of games
  - Fun and interesting for humans
  - Good level of difficulty to test algorithms
  - Simulation is easy to work with — good for testing ideas
- The goal is to build a general learning algorithm without game-specific knowledge
- We will allow some Atari-specific knowledge (e.g., size of input screen)
- Can we build an agent that can play all (or most) games well?

Starting point: DQN (Mnih et al. 2013, 2015)

- DQN includes:
  - Convolutional neural network q_θ: $O_t \mapsto \mathbb{R}^m$ for m actions
  - ϵ-greedy policy: π_t
  - Replay buffer for experience replay
  - Target network parameters θ^- (initially $\theta_0^- = \theta_0$)
  - Q-learning loss function on θ (uses replay and target network)

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

  - Optimization method to minimize the loss (e.g., SGD, RMSprop, or Adam)
  - Update $\theta_t^- \leftarrow \theta_t$ occasionally
    (e.g., every 10000 steps — on all other steps $\theta_t^- = \theta_{t-1}^-$)

Double DQN (van Hasselt et al. 2016)

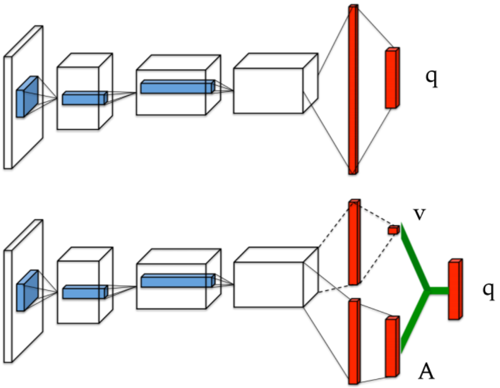
$$I(\theta) = \frac{1}{2}\left(R_{i+1} + \gamma [\![q_{\theta^-}(S_{i+1}, \underset{a}{\operatorname{argmax}}\ q_\theta(S_{i+1}, a))]\!] - q_\theta(S_i, A_i)\right)^2$$

Prioritized replay (Schaul et al. 2016)

- DQN samples uniformly from replay
- Idea: prioritize transitions on which we can learn much
- Basic implementation:
$$\text{priority of sample } i = |\delta_i| ,$$
where δ_i was the TD error on the last this transition was sampled
- Sample according to priority
- Typically involves some additional design choices

Dueling networks (Wang et al. 2016)

- We can decompose $q_\theta(s,a) = v_\xi(s) + A_\chi(s,a)$, where $\theta = \xi \cup \chi$
- Here $A_\chi(s,a)$ is the advantage for taking action a

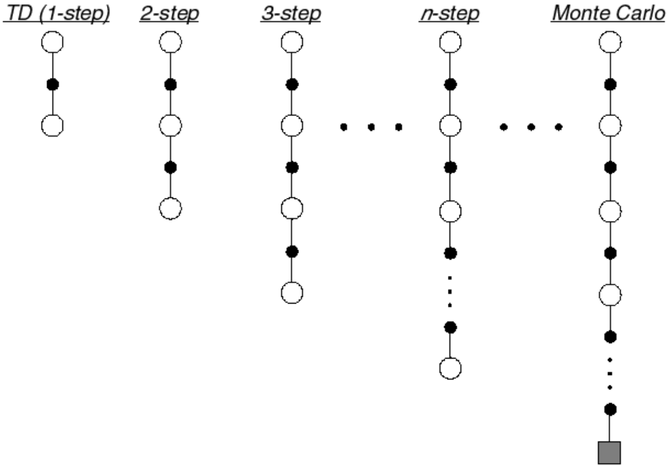


Dueling networks

Video

\yellow{Multi-step updates} (Sutton 1988)

- Let TD target look n steps into the future



\yellow{Multi-step updates}

- Consider the following n-step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll} n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1}) \\ n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\ \vdots & & \vdots \\ n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T \end{array}$$

- Define the n-step return

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

- n-step temporal-difference learning

$$v(S_t) \leftarrow v(S_t) + \alpha \left(G_t^{(n)} - v(S_t) \right)$$

\yellow{Multi-step updates}

- Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \underbrace{q_{\theta^-}(S_{i+1}, \underset{a}{\text{argmax}}\ q_\theta(S_{i+1}, a))}_{\text{Double Q bootstrap target}}$$

- Multi-step Q-learning

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

- Return is partially on-policy, bootstrap is off-policy
- That's okay — less greedy, but still policy improvement
- Still a well-defined prediction target:
  "what if I'm on-policy for n steps, and then take a greedy action?"

Distributional reinforcement learning (Bellemare, Dabney, Munos 2017)

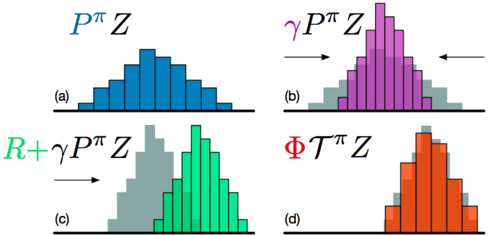
- So far, we've focused on learning expected cumulative rewards
- We can consider learning other things
- One possibility: learning the distribution of returns
- Knowing the distribution may be helpful for some things
- For instance, you can perhaps reason about the probability of termination
- It also means our representation (e.g., deep neural network) is forced to learn more
- This can speed up learning: learning more means potentially fewer samples

Distributional reinforcement learning

- A specific instance is Categorical DQN (Bellemare et al., 2017)
- Consider a 'comb' distribution on $\boldsymbol{z} = (-10, -9.9, \ldots, 9.9, 10)^\top$
- For each point of support, we assign a 'probability' $p_\theta^i(S_t, A_t)$
- The approximate distribution of the return s and a is the tuple $(\boldsymbol{z}, \boldsymbol{p}_\theta(s, a))$
- Our estimate of the expectation is: $\boldsymbol{z}^\top \boldsymbol{p}_\theta(s, a) \approx q(s, a)$ — use this to act
- Goal: learn these probabilities

Distributional reinforcement learning

1. Find max action:
   $$a^* = \operatorname*{argmax}_a \boldsymbol{z}^\top \boldsymbol{p}_\theta(S_{t+1}, a)$$
   (use, e.g., θ^- for double Q)
2. Update support:
   $$\boldsymbol{z}' = R_{t+1} + \gamma \boldsymbol{z}$$
3. Project distribution $(\boldsymbol{z}', \boldsymbol{p}_\theta(S_{t+1}, a^*))$ onto support $\boldsymbol{z}$
   $$d' = (\boldsymbol{z}, \boldsymbol{p}') = \Pi(\boldsymbol{z}', \boldsymbol{p}_\theta(S_{t+1}, a^*))$$
   where Π denotes projection
4. Minimize divergence
   $$\mathrm{KL}(d' \| d) = -\sum_i p'_i \frac{\log p'_i}{\log p^i_\theta(S_t, A_t)}$$



Bottom-right: target distribution $\Pi(R_{t+1} + \gamma \boldsymbol{z}, \boldsymbol{p}_\theta(S_{t+1}, a^*))$
Update $\boldsymbol{p}_\theta(S_t, A_t)$ towards this

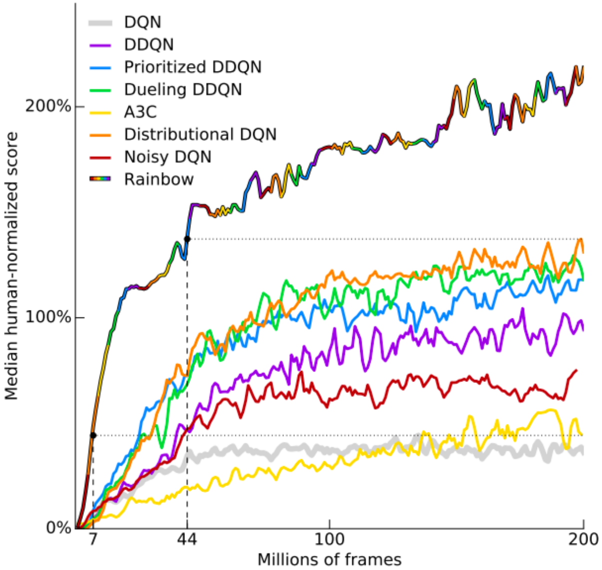
Noisy networks (Fortunato, Azar, Piot, et al. 2017)

- DQN used ϵ-greedy exploration
- We learnt that UCB is better in bandits, but this is hard with function approximation
- Idea: add noise to parameters, replace all linear operations $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ with

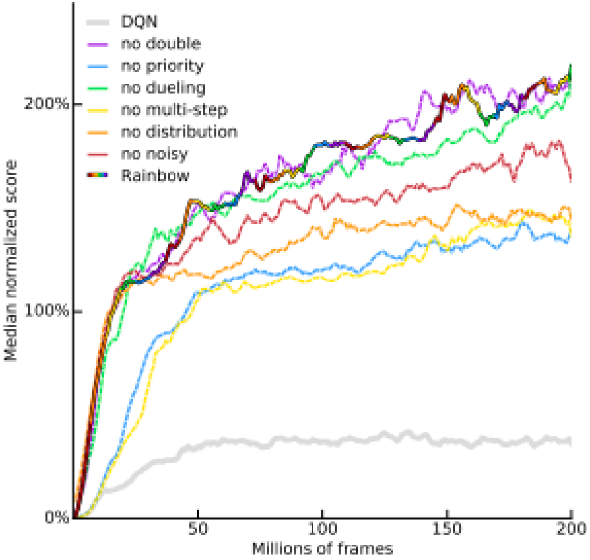
$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} + (\boldsymbol{W}' \times \varepsilon^{\boldsymbol{W}})\boldsymbol{x} + \boldsymbol{b}' \times \varepsilon^{\boldsymbol{b}}$$

  where '$\times$' is element-wise product and $\varepsilon^{\boldsymbol{W}}$ and $\varepsilon^{\boldsymbol{b}}$ are random matrix and vector
- The algorithm can learn to set $\boldsymbol{W}' = \mathbf{0}$ and $\boldsymbol{b}' = \mathbf{0}$, but that takes time
- In the meantime, output is stochastic, and more so for rarely seen inputs $\boldsymbol{x}$
- Results in exploration

Rainbow DQN: results



Rainbow DQN: ablation results



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 target normalization (van Hasselt et al. 2016)

- We can normalize targets before doing an update
- In online RL, we do not have access to the full 'data-set'
- In fact, data will change when our policy changes
- Solution: adaptive normalization of updates

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:

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

   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:

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

- Idea: define

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

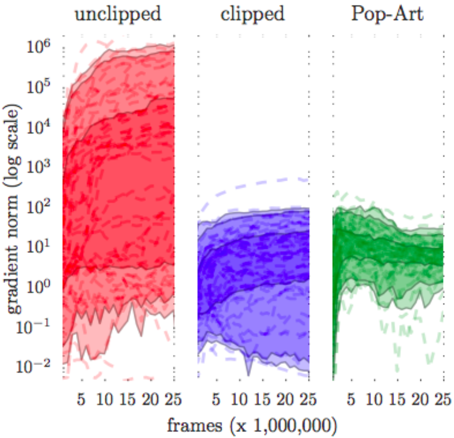
  Then

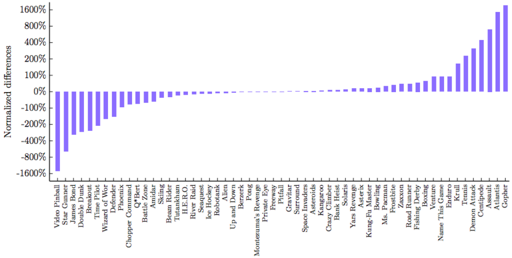
$$\sigma_{t+1}\tilde{\boldsymbol{q}}_{\boldsymbol{W}'_t,\boldsymbol{b}'_t,\theta_t}(s) + \mu_{t+1} \quad = \quad \sigma_t\tilde{\boldsymbol{q}}_{\boldsymbol{W}_t,\boldsymbol{b}_t,\theta_t}(s) + \mu_t$$

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

Preserve outputs

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



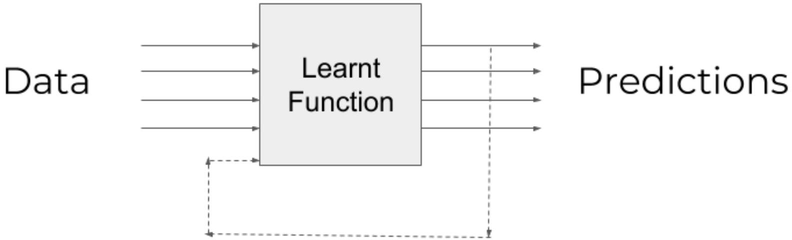


Pop-Art

Video

Obtaining knowledge

- We want our agents to understand and interact with their environment
- A single reward signal can be sparse, and low in information
- But we can use the same prediction methods to learn many things



Horde (Sutton et al. 2011)

General value functions

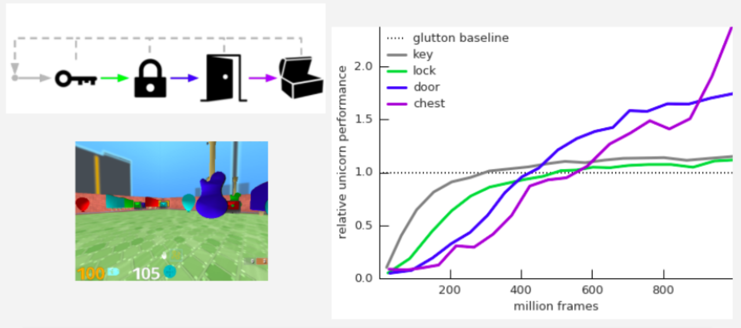
- Key idea is to consider general value functions (GVF)
- A GVF is conditioned on more than just state and actions

  $$q_{c,\gamma,\pi}(s,a) = \mathbb{E}\left[C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + \ldots \mid S_t = s, A_{t+i} \sim \pi(S_{t+i})\right]$$

  where $C_t = c(S_t)$ and $\gamma_t = \gamma(S_t)$ where S_t could be the environment state
- $c : \mathcal{S} \to \mathbb{R}$ is the cumulant
  - Predict many things, including — but not limited to — reward
- $\gamma : \mathcal{S} \to \mathbb{R}$ is the discount or termination
  - Predict for different time horizons γ
- $\pi : \mathcal{S} \to \mathcal{A}$ is the target policy
  - Predict under many different (hypothetical) policies π

Universal value function approximators (Schaul et al. 2015)

- Idea: feed a representation of (c, γ) is as input
- Allows generalization across goals/tasks within an environment



'Unicorn' (Mankowitz et al., 2018)

Learn about many things to learn to do the hard thing

GVF and models

- A transition model is a specific set of GVFs
  - The cumulants are the state components (e.g., pixels)
  - The termination/discount is zero
  - Models are often action-conditional, so we do not need to specify the policy
- Similarly for an expected reward model
  - Cumulant = reward
  - Termination is immediate $\gamma = 0$
- Easily extends to multi-step models
- Rolling forward a model = using predictions as inputs for other predictions

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