Lecture 8: Building agents

Hado van Hasselt

Outline

Recap

- Reinforcement learning is the science of learning to make decisions
- We can do this by learning one or more of:
  - policy
  - value function
  - model
- The general problem involves taking into account time and consequences
- Our decisions affect the reward, our internal knowledge, and the state of the environment

This Lecture

- So far, we mainly discussed learning algorithms
- How do we build a full agent?
- We will talk about which components are needed, and how to combine these
- We will talk about policies and exploration

Interface between agent and environment

- The typical interface to a task is: action goes in, reward and observation come out
- So, our full agent must be a function: $A_t = \text{agent}(R_t, O_t)$
- A policy can be considered a simple agent that only looks at the observation O_t
- We want learning agents, which requires learning from R_t as well
- Simple policies can be embedded inside complex agents

High-level agent components

- Agents may have several components
  - An agent must include a policy $\pi_t : \mathcal{S} \to \mathcal{A}$
    where $\mathcal{S}$ is the set of possible agent states
  - An agent can include a representation $S_t = f(S_{t-1}, O_t)$
  - An agent can include an algorithm to learn the policy
  - An agent can include an algorithm to learn the representation
  - An agent can include an algorithm to learn predictions
  - An agent can include an algorithm to learn a model
  - An agent can include memory of past experiences
  - An agent can include ...
- Different components may or may not share (parts of) the representation

Example: neural Q-learning

- Online neural Q-learning may include:
  - A network q_θ: $O_t \implies (q[1], \ldots, q[m])$ (m actions)
  - An ϵ-greedy exploration policy: $q_t \implies \pi_t \implies A_t$
  - A Q-learning loss function on θ

$$I(\theta) = \frac{1}{2}\left(R_{t+1} + \gamma\left[\max_a q_\theta(S_{t+1}, a)\right] - q_\theta(S_t, A_t)\right)^2$$

  where $[\cdot]$ denotes stopping the gradient, so that the gradient is

$$\nabla_\theta I(\theta) = \left(R_{t+1} + \gamma \max_a q_\theta(S_{t+1}, a) - q_\theta(S_t, A_t)\right) \nabla_\theta q_\theta(S_t, A_t)$$

  - An optimizer to minimize the loss (e.g., SGD, RMSProp, Adam)

Example: TF pseudo-code for Q-learning

```
# Compute Q values Q(S_t, .)
q = q_net(obs)

# Get action A_t
action = epsilon_greedy(q)

# Compute Q(S_t, A_t)
qa = q[action]

# Step in environment
reward, discount, next_obs = env.step(action)

# Get max of values at next state
max_q_next = tf.reduce_max(q_net(next_obs))

# Compute TD-error, do not to propagate into next state value
delta = reward + discount * tf.stop_gradient(max_q_next) - qa

# Define loss
q_loss = tf.square(delta)/2
```

Example: DQN

- DQN includes:
  - A network q_θ: $O_t \implies (q[1], \ldots, q[m])$ (m actions)
  - An ϵ-greedy exploration policy: $q_t \implies \pi_t \implies A_t$
  - A replay buffer to store and retrieve past experiences
  - A target network q_{θ^-}: $O_t \implies (q^-[1], \ldots, q^-[m])$
  - A Q-learning loss function on θ
    (uses replay and target network)

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

  - An optimizer to minimize the loss

Example: (Asynchronous) Advantage actor critic

- Advantage actor critic includes:
  - A representation (e.g., LSTM): $(S_{t-1}, O_t) \implies S_t$
  - A network v_η: $S \implies v$
  - A network π_θ: $S \implies \pi$
  - Copies/variants π^m of π_θ as policies: $S_t^m \implies A_t^m$
  - A n-step TD loss on v_η

$$l(\eta) = \frac{1}{2}\left(G_t^{(n)} - v_\eta(S_t)\right)^2$$

    where $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} v_{\eta^-}(S_{t+n})$
  - A n-step REINFORCE loss on (each) π_η

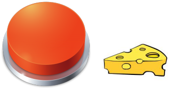
$$l(\theta) = \left[G_t^{(n)} - v_\eta(S_t)\right] \log \pi_\theta(A_t|S_t)$$

  - Optimizers to minimize the losses
- The optimizer can be synchronous (A2C), or asynchronous (A3C)

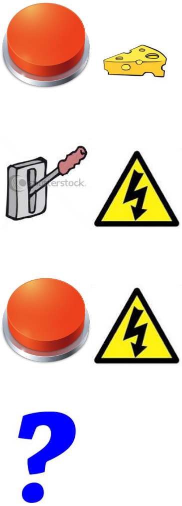
Policies

- We typically agents that learn to improve their behaviour over time
- Good policies trade off exploration and exploitation
- There are many design choices
  - Policies may depend on internal state $S_t \in \mathcal{S}$, or just on observation O_t
  - Policies could be 'blind', e.g., consider uniform random policies
  - Output actions must be compatible with the environment
    e.g., integer for Atari, real-valued vector for continuous control

Rat Example



Rat Example

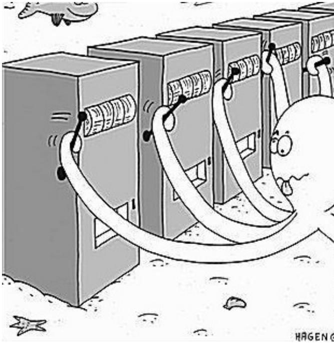


Exploration vs. Exploitation

- Online decision-making involves a fundamental choice:
  - Exploitation: Maximize return given current knowledge
  - Exploration: Increase knowledge
- The best long-term strategy may involve short-term sacrifices
- Gather enough information to make the best overall decisions

Recap: The Multi-Armed Bandit

- A multi-armed bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$
- $\mathcal{A}$ is a known set of actions (or "arms")
- $\mathcal{R}^a(r) = \mathbb{P}\left[R_t = r | A_t = a\right]$ is an unknown probability distribution over rewards
- At each step t the agent selects an action $A_t \in \mathcal{A}$
- The environment generates a reward $R_t \sim \mathcal{R}^{A_t}$
- The goal is to maximize cumulative reward $\sum_{i=1}^{t} R_i$
- Repeated 'game against nature'



Regret

- The *optimal value* v_* is

$$v_* = \max_{a \in \mathcal{A}} q(a) = \max_a \mathbb{E}\left[R_t \mid A_t = a\right]$$

- Regret is the opportunity loss for one step

$$v_* - q(A_t)$$

- E.g., I might regret going to class, although I might learn by going

Regret

- Trade-off exploration and exploitation by minimizing *total regret*:

$$L_t = \sum_{i=1}^{t} v_* - q(a_i)$$

- Maximise cumulative reward $\equiv$ minimise total regret
- Note: cumulation here extends over termination of 'episode'
- Extends over 'lifetime of learning', rather than over 'current episode'

Counting Regret

- The *gap* Δ_a is the difference in value between action a and optimal action a_*, $\Delta_a = v_* - q(a)$
- Total regret depends on gaps and counts

$$
\begin{aligned}
L_t &= \sum_{i=1}^{t} v_* - q(a_i) \\
&= \sum_{a \in \mathcal{A}} N_t(a)(v_* - q(a)) \\
&= \sum_{a \in \mathcal{A}} N_t(a)\Delta_a
\end{aligned}
$$

- A good algorithm ensures small counts for large gaps
- Problem: gaps are not known...

Exploration

- We need to explore to learn about the values of all actions
- What is a good way to explore?
- One common solution: ϵ-greedy
  - Select greedy action (exploit) w.p. $1 - \epsilon$
  - Select random action (explore) w.p. ϵ
- Used in Atari
- Is this enough?
- How to pick ϵ?

ϵ-Greedy Algorithm

- Greedy can lock onto a suboptimal action forever
- $\Rightarrow$ Greedy has linear expected total regret
- The ϵ-*greedy* algorithm continues to explore forever
  - With probability $1 - \epsilon$ select $a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a)$
  - With probability ϵ select a random action
  - Constant ϵ ensures minimum expected regret

$$\mathbb{E}\left[v_* - q(A_t)\right] \geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Delta_a$$

- $\Rightarrow$ ϵ-greedy with constant ϵ has linear total regret

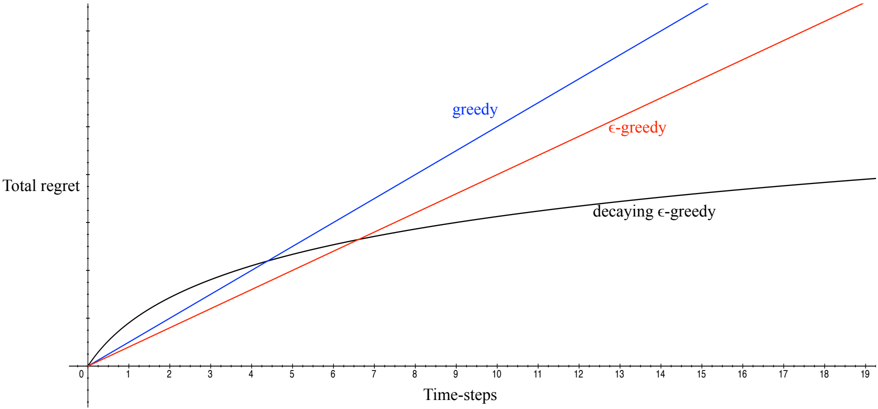
Decaying ϵ_t-Greedy Algorithm

- Pick a decay schedule for $\epsilon_1, \epsilon_2, ...$
- Consider the following schedule

$$\epsilon_t = \frac{c}{t} \qquad \text{where} \qquad c \in (0, 1]$$

- Decaying ϵ_t-greedy has *logarithmic* asymptotic total regret
- Finding proper decay (e.g., c) is not trivial (can depend on problem)

Linear or Sublinear Regret



Lower Bound

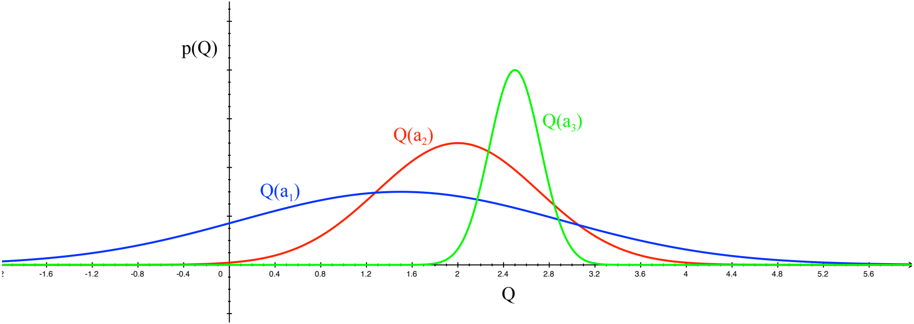
- The performance of any algorithm is determined by similarity between optimal arm and other arms
- Hard problems have arms with similar distributions but different means
- This is described formally by the gap Δ_a and the similarity in distributions $KL(\mathcal{R}^a||\mathcal{R}^a*)$

Theorem (Lai and Robbins)

Asymptotic total regret is at least logarithmic in number of steps

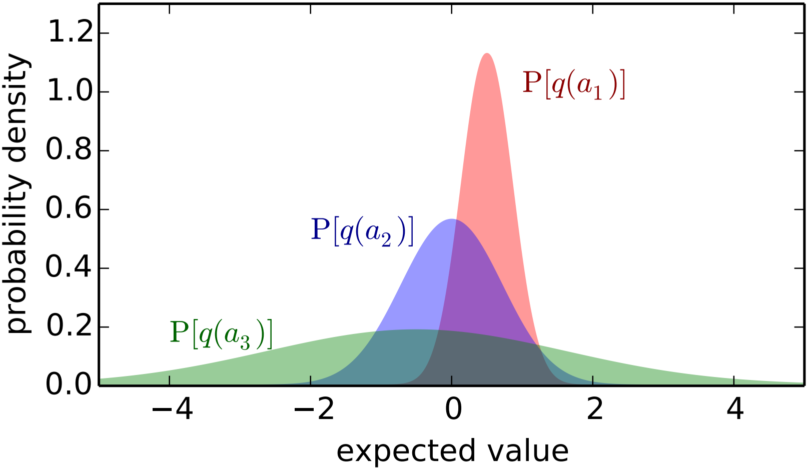
$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\Delta_a>0} \frac{\Delta_a}{KL(\mathcal{R}^a||\mathcal{R}^{a_*})}$$

Optimism in the Face of Uncertainty

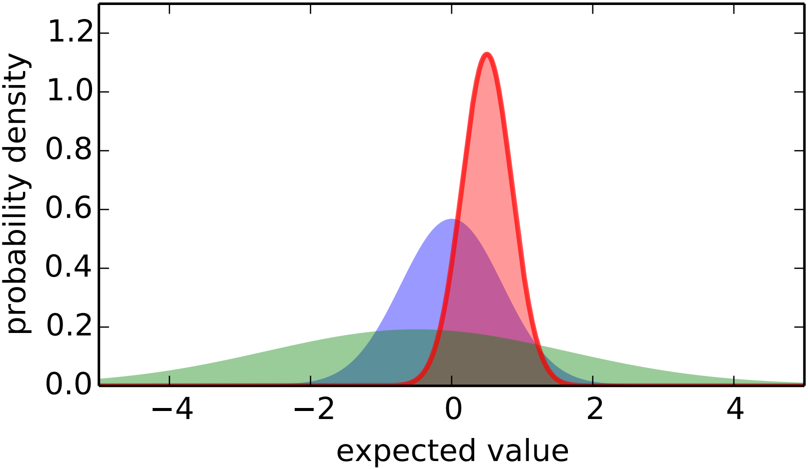


- Which action should we pick?
- More uncertainty: more important to explore that action
- It could turn out to be the best action

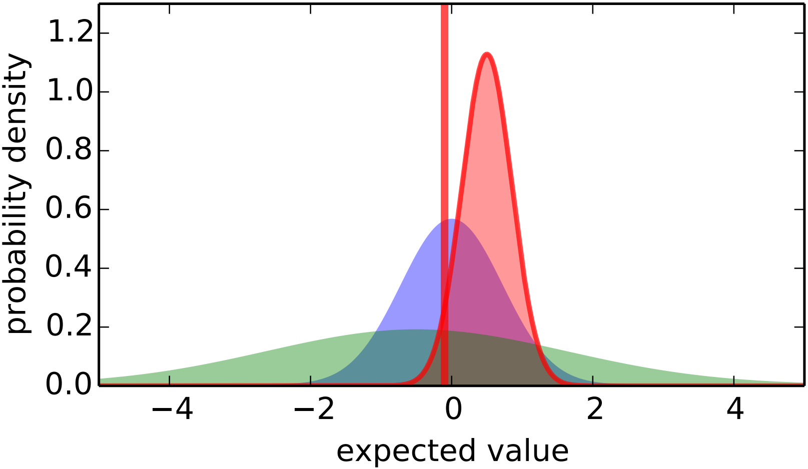
Optimism in the Face of Uncertainty



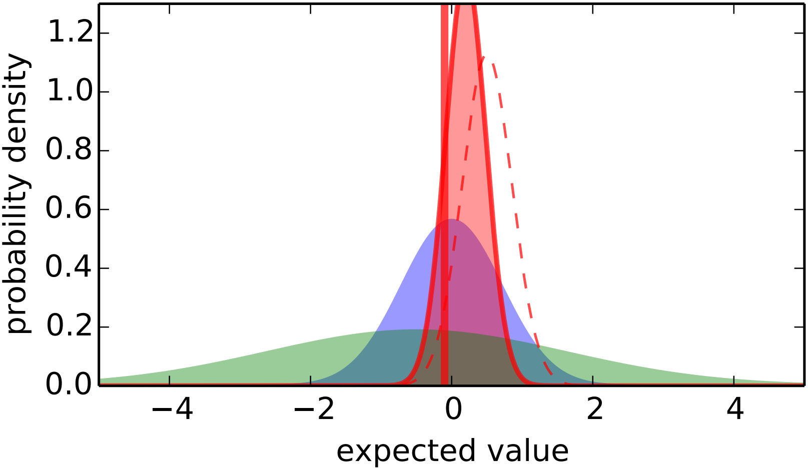
Optimism in the Face of Uncertainty



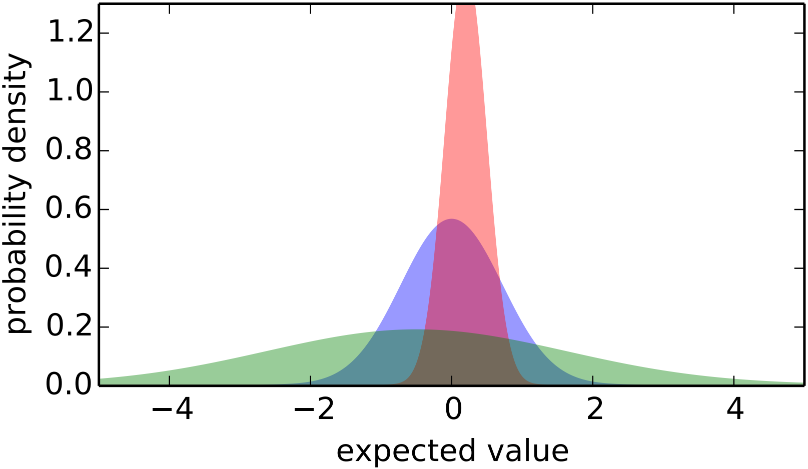
Optimism in the Face of Uncertainty



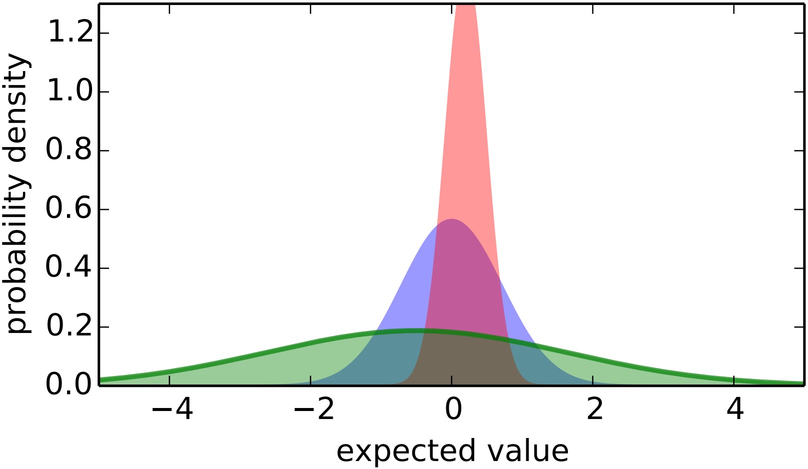
Optimism in the Face of Uncertainty



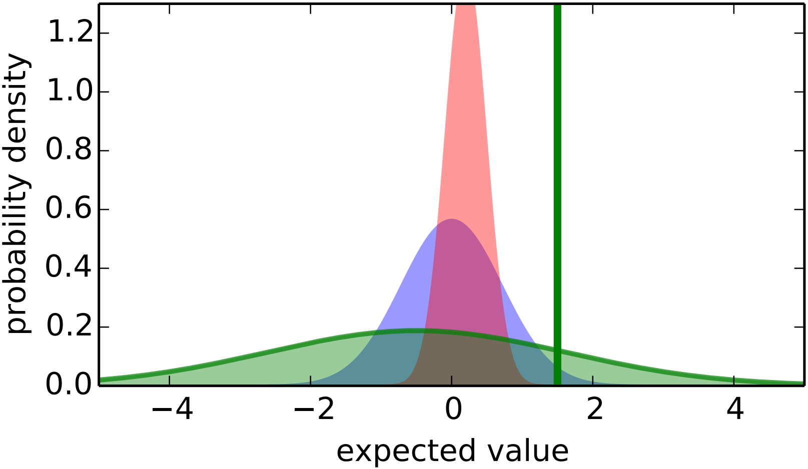
Optimism in the Face of Uncertainty



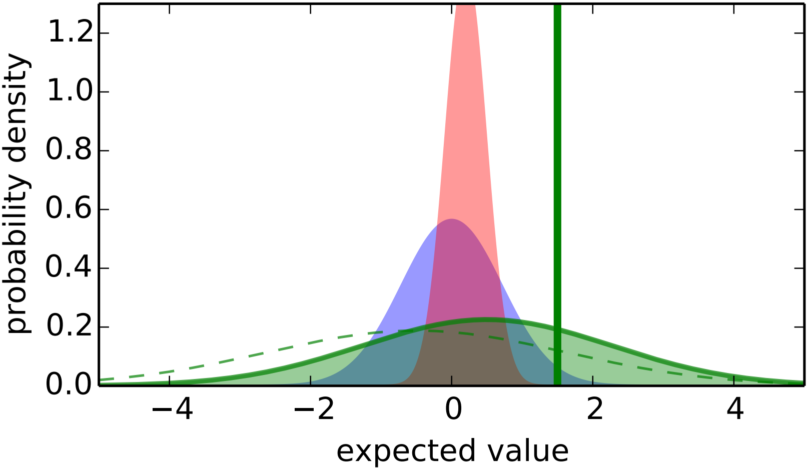
Optimism in the Face of Uncertainty



Optimism in the Face of Uncertainty



Optimism in the Face of Uncertainty



Upper Confidence Bounds

- Estimate an upper confidence $U_t(a)$ for each action value, such that $q(a) \leq Q_t(a) + U_t(a)$ with high probability
- Uncertainty depends on the number of times $N(a)$ has been selected
  - Small $N_t(a) \Rightarrow$ large $U_t(a)$ (estimated value is uncertain)
  - Large $N_t(a) \Rightarrow$ small $U_t(a)$ (estimated value is accurate)
- Select action maximizing Upper Confidence Bound (UCB)

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\; Q_t(a) + U_t(a)$$

Hoeffding's Inequality

Theorem (Hoeffding's Inequality)

Let $\Sigma_1, ..., \Sigma_t$ be i.i.d. random variables in [0,1], and let $\overline{X}_t = \frac{1}{t}\sum_{i=1}^{t}\Sigma_i$ be the sample mean. Then

$$\mathbb{P}\left[\mathbb{E}\left[X\right] > \overline{X}_t + u\right] \leq e^{-2tu^2}$$

- We can apply Hoeffding's Inequality to bandits with bounded rewards
- E.g., if $R_t \in [0, 1]$, then

$$\mathbb{P}\left[q(a) > Q_t(a) + U_t(a)\right] \leq e^{-2N_t(a)U_t(a)^2}$$

Calculating Upper Confidence Bounds

- Pick a probability p that true value exceeds UCB
- Now solve for $U_t(a)$

$$e^{-2N_t(a)U_t(a)^2} = p$$

$$U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

- Reduce p as we observe more rewards, e.g. $p = t^{-1}$
- Ensures we select optimal action as $t \to \infty$

$$U_t(a) = \sqrt{\frac{\log t}{2N_t(a)}}$$

UCB1

- This leads to the UCB algorithm

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}$$

with $c > 0$

Theorem (Auer et al., 2002)

The UCB algorithm (with $c = 1/\sqrt{2}$) achieves logarithmic expected total regret

$$L_t \leq 8 \sum_{a|\Delta_a > 0} \frac{\log t}{\Delta_a} + O(\sum_a \Delta_a)$$

for any t

Value of Information

- Exploration is valuable because information is valuable
- Can we quantify the value of information?
- Information gain is higher in uncertain situations
- Therefore it makes sense to explore uncertain situations more
- If we know value of information, we can trade-off exploration and exploitation optimally (in bandits)

Solving information state space bandits

- We can formulate bandits as an infinite MDP over information states
- 'Information state' includes (sufficient statistic) information of what we have learned (e.g., rewards and actions)
- Can be solved by reinforcement learning
- Model-free reinforcement learning
  - e.g. Q-learning (Duff, 1994)
- Bayesian model-based reinforcement learning
  - e.g. Gittins indices (Gittins, 1979)
- Latter approach is known as *Bayes-adaptive* RL
- Finds Bayes-optimal exploration/exploitation trade-off (with respect to prior distribution)

Towards complex environments

- Bandits are simple, in several senses
  - Tabular states, tabular actions, no generalisation
  - No sequential nature within episodes
- Real problems are much harder
- Key insights:
  - Optimism in the face of uncertainty
  - Seek information
- Key question: can we find good proxy for information?
- We need policies that yield information—in some settings this is easy, in some this is hard

Example: Cart Pole

- Imagine balancing a pole on a cart by hitting the cart
- If each episode starts with an almost-balanced pole, exploration is easy

Jittering

- Simplest approach to exploration: jitter
- Example: ϵ-greedy
- Example: soft max policy $\pi(a|s) = e^{p(s,a)} / \sum_b e^{p(s,b)}$
  for some preference function p (e.g., $p(s,a) = q(s,a)/\tau$)
- Used in much current state of the art!

Intrinsic motivation

- Idea: UCB uses counts of states and actions
- These counts capture a notion of uncertainty
- Can we use (pseudo-) counts in complex environments?
- Can we use motivate the agent to go to novel situations?

Video: Montezuma's revenge

https://www.youtube.com/watch?v=0yI2wJ6F8r0

Agent components

- This agent had an additional a component: a density model
- This component was used to improve the behaviour policy
- This component can be (and recently was) swapped out and improved, while keeping the rest of the agent fixed
- But often components interact, so revisiting other components may be important (e.g., retuning)

Auxiliary predictions

- Sometimes we can improve components in interesting ways
- E.g., we can use a representation to make multiple predictions
- For instance: predict immediate rewards with the representation in A2C
- This help performance, even though the signal is quite indirect!

$$\text{Auxiliary predictions} \implies \text{Better representation} \implies \text{Better policy}$$

Models

- We have seen that models can be useful components
- In last lecture, a model was trained, and then used to update to a value function (Dyna)
- In the past, model-based RL was known to be data efficient, while model-free RL was thought to be more computation efficient
- These days: less clear
  - It is hard to learn good models in messy domains
  - Perhaps we are not using them optimally
  - However, replay is essentially a non-parametric model and is useful
  - We probably do not want to model individual pixels—but what then?
  - More to be done!

Questions

Topics and Terms

- Reward (and the reward hypothesis)
- Observation
- Action
- History
- State, environment state, agent state, Markov state
- Markov decision process (MDP)
- Partially observable MDP (POMDP)
- Discount
- Policy
- Value function
- Model-free / model-based
- Actor critic
- Planning
- Prediction
- Control
- Exploration/exploitation
- Multi-armed bandit
- Return
- Bellman equation
- Bellman optimality equation
- Dynamic programming
- Policy evaluation
- Policy improvement
- Policy iteration
- Generalized policy iteration
- Value iteration
- Contraction mappings
- Monte Carlo
- Temporal difference (TD)
- Bootstrapping
- TD error
- n-step returns
- λ-returns, TD(λ)
- Eligibility traces (forward view, backward view)
- On-policy / off-policy
- Sarsa, Q-learning
- Policy gradient
- REINFORCE (with and without baseline)
- Dyna
- Monte Carlo tree search (MCTS)
- Regret
- Optimism in the face of uncertainty