ml-mipt-advanced

Lecture 11. Exploration strategies. RL outside games.

Nikolay Karpachev
15.04.2024

Outline

- Exploration / exploitation tradeoff
- Exploration strategies
  - Heuristic-based
  - “Optimism in the face of uncertainty”
  - Probability matching

- Structured prediction in NLP -> RL framing
- Optimizing for undifferentiable with policy gradient
- SCST

Exploration strategies

Exploration vs. Exploitation in RL

- Online decision-making involves a fundamental choice
  - **Exploitation:** Make the best decision given the current information
  - **Exploration:** Gather more information
- The best long-term strategy may involve short-term sacrifices
- Agent should gather enough relevant information to make reasonable decisions

Exploration vs. Exploitation: examples

- Restaurant selection
  - **Exploitation**: Go to your favourite restaurant
  - **Exploration**: Try new restaurant
- Online banner advertisements
  - **Exploitation**: Show the most successful advert
  - **Exploration**: Show a different advert
- Game playing
  - **Exploitation**: Play the move you believe is the best
  - **Exploration**: Play a different move

Multi-armed bandit

- What is a bandit?

Multi-armed bandit

Multi-armed bandit

- A single state
- Set of possible actions (decide which slot machine to play)
- Each machine has an unknown probability of success
- <u>Goal: maximize the total number of successful games</u>

Regret

$$Q(a) = \mathbb{E}\left[r|a\right]$$

$$V^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(a)$$

- Regret (Total Regret): opportunity loss for one step (all steps)

$$l_t = \mathbb{E}\left[V^* - Q(a_t)\right]$$

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right]$$

<u>**We want to minimize the total regret**</u>

Exploration strategies so far

- Eps-greedy
  - With p = eps, take random action. Optimal otherwise
- Boltzman (aka softmax)
  - Pick actions proportionately to scaled Q-values

$$P(a) = softmax\left(\frac{Q(s,a)}{std}\right)$$

- Decaying eps-greedy
  - Same as eps-greedy; start with high eps, decrease it during training

Greedy algorithm

- Always selects actions with highest values
- What is the total regret?

Greedy algorithm

- Always selects actions with highest values
- What is the total regret?

- Greedy can lock to a suboptimal action forever
- Hence, **linear total regret**

Epsilon-greedy algorithm

- Explores forever
- Selects suboptimal actions with fixed probability over and over again
- Linear total regret

Epsilon-greedy with decay

- Has a decay schedule for eps
- With properly selected schedule, has a **logarithmic total regret**
- But to design a proper schedule can be tricky

Optimism in the face of uncertainty

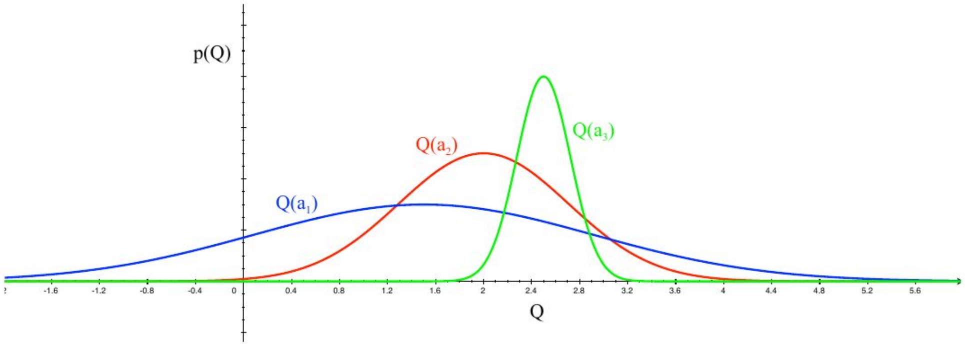
- How do humans explore?
- For example, which of the following questions would you like to investigate?
  - Whether humans can fly by pulling their hair up
  - Whether the new cafe next to the office serves good breakfast

Optimism in the face of uncertainty

- How do humans explore?
- For example, which of the following questions would you like to investigate:
  - Whether humans can fly by pulling their hair up
  - Whether the new cafe next to the office serves good breakfast

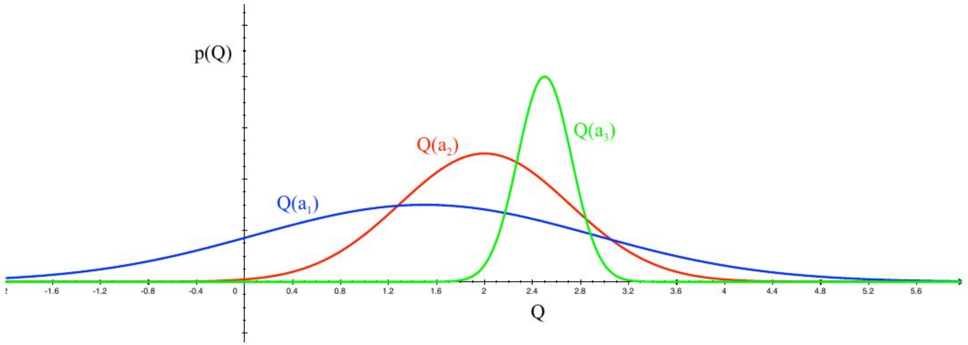
- We want to try actions if we believe there’s a chance they are good

Optimism in the face of uncertainty



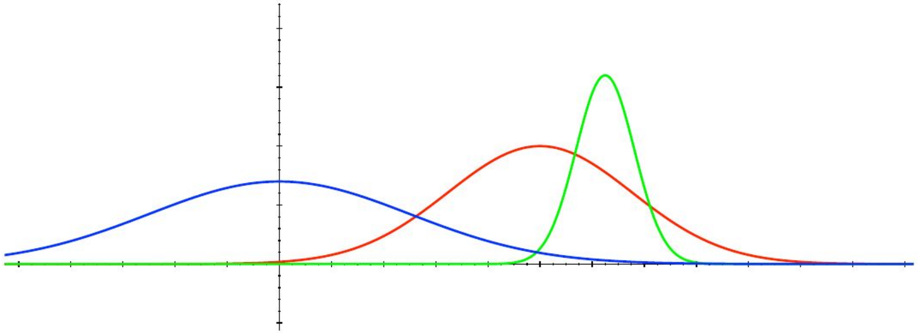
- Which action should we pick?

Optimism in the face of uncertainty



- Which action should we pick?
- The more uncertain we are about an action value
- The more important it is to try that action
- It could turn out to be the best action

Optimism in the face of uncertainty



- After picking blue action
- We are less uncertain about the value
- And more likely to pick another action

Upper confidence bounds

- We want to select
  - Uncertain outcomes
  - With greater expected value

Upper confidence bounds

- We want to select
  - Uncertain outcomes
  - With greater expected value

- Let’s compute 95% upper confidence bound for each action
- Take action with the highest upper confidence bound

Upper confidence bounds

Theorem (Hoeffding's inequality):

Given a sample of a random variable bounded in [0, 1],

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

Upper confidence bounds

- We can apply Hoeffding's inequality to the case of bandits:

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

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

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

Upper confidence bounds

- With fixed p (e.g. 95% UCB)

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

- Possible extension: reduce p during training (ucb converges to 1 in the limit; this guarantees optimal solution)

$$p = t^{-4} \qquad\qquad U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}}$$

UCB-1 algorithm

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

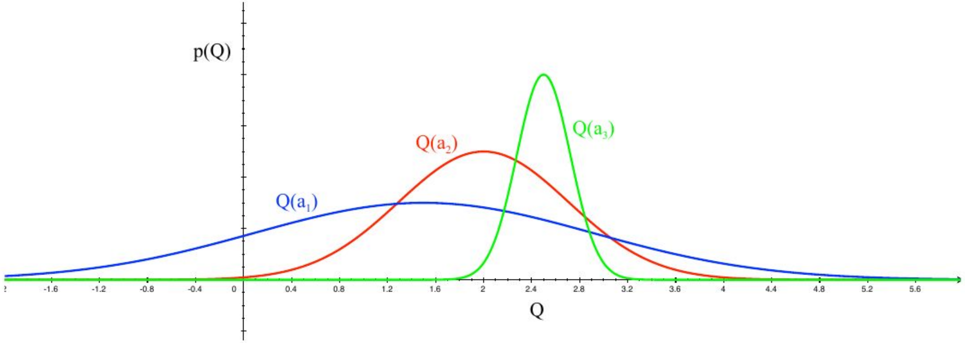
- Achieves **logarithmic total regret**

Bayesian UCB

- Assign prior distribution P(Q(s,a))
- Learn posterior P(Q(s,a)|data)
- Take q-th percentile of P(Q(s,a)) and select the best action

Probability matching

- Select action a according to the probability that **a is the optimal action**



$$\pi(a \mid h_t) = \mathbb{P}\left[Q(a) > Q(a'), \forall a' \neq a \mid h_t\right]$$

Thompson sampling

- Compute posterior distribution for each Q(s,a)
- Sample from each action's posterior
- Select action with max value on sample
- **Thompson sampling will select action proportionately to the probability that this action is optimal**

RL outside games

NLP Training / Inference Regimes

- Supervised seq2seq learning:

$$P(y_{t+1} | x, y_{0:t}), \qquad y_{0:t} \sim reference$$

- Inference

$$P(y_{t+1} | x, \hat{y}_{0:t}), \qquad \hat{y}_{0:t} \sim \textbf{???}$$

NLP Training / Inference Regimes

- Supervised seq2seq learning:

$$P(y_{t+1} | x, y_{0:t}), \qquad y_{0:t} \sim reference$$

- Inference

$$P(y_{t+1} | x, \hat{y}_{0:t}), \qquad \hat{y}_{0:t} \sim \textbf{???}$$

If model ever makes something that isn't in data,
It gets volatile from next time-step!

NLP Training / Inference Regimes

- Supervised seq2seq learning:

$$P\left(y_{t+1}|x, y_{0:t}\right), \qquad y_{0:t} \sim reference$$

- Inference

$$P\left(y_{t+1}|x, \hat{y}_{0:t}\right), \qquad \hat{y}_{0:t} \sim \textbf{???}$$

"Exposure bias"

If model ever makes something that isn't in data,
It gets volatile from next time-step!

Issue 1: Exposure bias

- Need good training data
  - Abundant
  - Few biases and noise
- Need a perfect network to extrapolate training set

Issue 1: Exposure bias

- Need good training data
  - Abundant
  - Few biases and noise
- Need a perfect network to extrapolate training set

Spoiler: Most of the time we don’t. Too bad.

Issue 2: Crossentropy loss

Reality: sometimes there is more than one correct translation (image caption, summary, whatever)

Source: 在 找 给 家里 人 的 礼物.

Versions:
i 'm searching for some gifts for my family.
i want to find something for my family as presents.
i 'm about to buy some presents for my family.
i 'd like to buy my family something as a gift.
i 'm looking for a present for my family.
...

Issue 2: Crossentropy loss

Reality: sometimes there is more than one correct translation (image caption, summary, whatever)

Source: 在 找 给 家里 人 的 礼物.

Versions:	Model 1 **p(y|x)**	Model 2 **p(y|x)**
(version 1)	1e-2	0.99
(version 2)	2e-2	1e-100
(version 3)	1e-2	1e-100
(all rubbish)	0.96	0.01

Question: which model has better Mean log p(y|x) ?

not in data

This one. While it predicts 96% rubbish

Issue 3: Training data

Two kinds of datasets:

Big enough, but suboptimal R(x,y)

- **Large raw data**
  - twitter, open subtitles, books, bulk logs
  - 10^6-8 samples, http://opus.nlpl.eu/OpenSubtitles.php

- **Small clean data**
  - moderated logs, assessor-written conversations
  - 10^2~4 samples

Near-optimal R(x,y), but too small

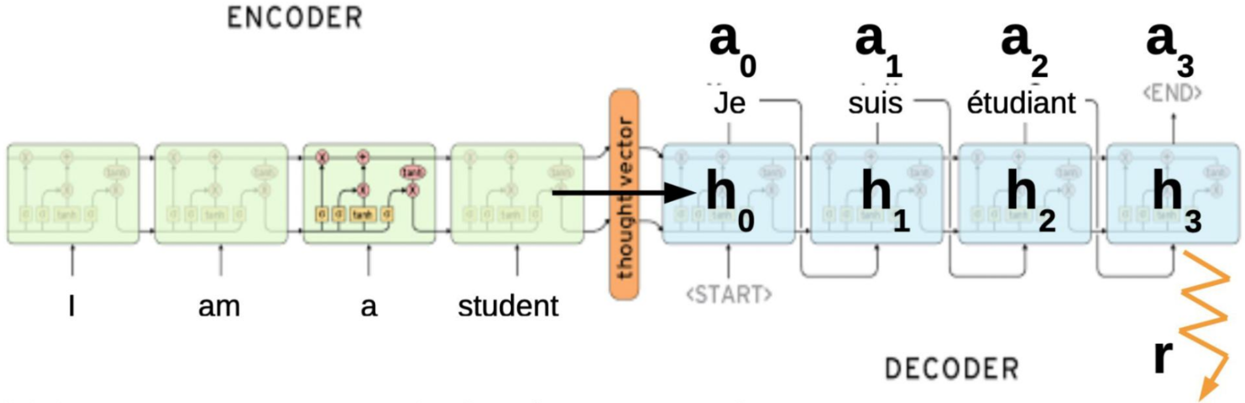
Can we do better?

Key idea: frame structured prediction problem as RL task

Pros:

- Can optimize any objective function directly (not just probability of correct stuff)
  - User feedback (discrete scores)
  - External reward model
  - Practically, any black-box stuff
- Inference data is the new training data!

Seq2Seq as POMDP



Hidden state **s** = translation/conversation state
Initial state **s** = encoder output
Observation **o** = previous words
Action **a** = write next word
Reward **r** = domain-specific reward (e.g. BLEU)

Training with Policy Gradient

Our objective:

Reward
(e.g. BLEU)

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}}{E} R(s,a) = \int_s p(s) \int_a \pi_\theta(a|s) R(s,a)\, da\, ds$$

parameters are
hidden here

We can approximate the expectation with mean:

$$J \approx \frac{1}{N} \sum_{i=0}^{N} R(s,a)$$

Training with Policy Gradient

Our objective:

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}}{E} R(s,a) = \int_s p(s) \int_a \pi_\theta(a|s) R(s,a)\, da\, ds$$

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s,a)\, da\, ds$$

Expectation is lost!

We don’t know how to compute the gradient w.r.t. parameters

Training with Policy Gradient

Problem: we need gradients on parameters

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}}{E} R(s,a) = \int_s p(s) \int_a \pi_\theta(a|s) R(s,a) \, da \, ds$$

Potential solution: Finite differences

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_\theta}{\epsilon}$$

Very noisy, especially if both J are sampled

Training with Policy Gradient

Problem: we need gradients on parameters

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}}{E} R(s,a) = \int_s p(s) \int_a \pi_\theta(a|s) R(s,a)\, da\, ds$$

Wish list:

- Analytical gradient
- Easy/stable approximations

Log-derivative trick

Simple math question:

$$\nabla \log \pi(z) = ???$$

Log-derivative trick

Simple math question:

$$\nabla \log \pi(z) = ???$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$

Policy Gradient Approximation

Algorithm: Monte-Carlo Policy Gradient (REINFORCE)

$$\mathcal{J}(\theta) = \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta) Q_\pi(s,a)$$

$$\begin{aligned}
\nabla\mathcal{J}(\theta) &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \nabla\pi(a|s;\theta) Q_\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta) \frac{\nabla\pi(a|s;\theta)}{\pi(a|s;\theta)} Q_\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta) \nabla\ln\pi(a|s;\theta) Q_\pi(s,a) \\
&= \mathbb{E}_{\pi_\theta}[\nabla\ln\pi(a|s;\theta) Q_\pi(s,a)]
\end{aligned}$$

by log-derivative trick

this expectation can be estimated by samples of episodes

Policy Gradient

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s,a) da\, ds$$

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \nabla \log \pi_\theta(a|s) \cdot R(s,a)$$

Supervised Learning vs. Policy Gradient

Supervised learning:

$$\nabla llh = \underset{s, a_{opt} \sim D}{E} \nabla \log \pi_\theta(a_{opt}|s)$$

Policy gradient:

$$\nabla J = \underset{\substack{s \sim d(s) \\ a \sim \pi(a|obs(s))}}{E} \nabla \log \pi_\theta(a|s) Q(s,a)$$

Question: what is different? (apart from Q(s, a))

Supervised Learning vs. Policy Gradient

Supervised learning:

$$\nabla llh = \underset{s, a_{opt} \sim D}{E} \nabla \log \pi_\theta (a_{opt} | s)$$

reference

Policy gradient:

$$\nabla J = \underset{\substack{s \sim d(s) \\ a \sim \pi(a | obs(s))}}{E} \nabla \log \pi_\theta (a | s) Q(s, a)$$

generated

Supervised Learning vs. Policy Gradient

Supervised learning:

- Need (near-)optimal dataset
- Trains on reference sessions

Policy gradient:

- Need ~some data and reward function
- Trains on its own output

Supervised Learning vs. Policy Gradient

Supervised Learning

Need good reference (y_opt)

If model is *imperfect* [and **it is**],
training:
P(y_next|x,y_prev_ideal)
prediction:
P(y_next|x,y_prev_predicted)

Reinforcement Learning

Need reward function

Model learns to improve current policy. If policy is pure random, local improvements are unlikely to produce good translation.

Supervised Learning vs. Policy Gradient

Supervised Learning

+ Rather simple
+ Small variance

– Need good reference (y_opt)
– **Distribution shift:**
  different **h** distribution
  when training vs generating

Reinforcement learning

+ Cold start problem
+ Large variance (so far)

– Only needs x and r(s,a)
– No **distribution shift**

Supervised Learning vs. Policy Gradient

Supervised Learning

pretraining

+ Rather simple
+ Small variance

– Need good reference (y_opt)
– **Distribution shift:**
  different **h** distribution
  when training vs generating

Reinforcement learning

finetuning

+ **Cold start problem**
+ Large variance (so far)

– Only needs x and r(s,a)
– No **distribution shift**

Outro

This lecture covered:

- Exploration-vs-exploitation tradeoff in RL
- How to compare exploration strategies
- Algorithms:
  - Greedy, eps-greedy, softmax-sampling
  - Upper confidence bound based sampling
  - Probability matching and thompson sampling

Outro

This lecture covered:

- Exploration-vs-exploitation tradeoff in RL
- How to compare exploration strategies
- Algorithms:
  - Greedy, eps-greedy, softmax-sampling
  - Upper confidence bound based sampling
  - Probability matching and thompson sampling
- We can solve **any structured prediction task** as RL problem
- Policy gradient -> optimzie any reward (e.g. BLEU, Human Eval scores, CTR, ranking NDCG, etc.)

Acknowledgements

This lecture uses materials from
(1) RL Lectures by David Siver (licensed CC-BY-NC 4.0)
(2) Practical_RL lectures by Yandex Data School (Unlicense license)

Questions?