Reinforcement Learning

1 Introduction

Own self-test questions:

- What is the „return“?

  The expected cumultative future reward.

- What are possible components of a RL agent?

  The policy, value function and/or model of the environment.

- How is the value or Q-function defined?

  Value Function: The expected future reward if agent is in state s and follows policy π (Quality metric for states).

  Q-Function: The expected future reward if agent takes action a in state s and follows policy π afterwards (Quality metric for state-action pairs).

- The general anatomy of RL algorithms.

  Generate samples $\rightarrow$ fit a model/estimate return $\rightarrow$ improve policy $\rightarrow$ repeat.

- The taxonomy of RL algorithms.

  Value based (fit value function, policy implicit), policy search (estimate return, optimize parameters), actor critic (fit value function, optimize parameters), model-free (fit model/estimate return, improve policy).

- What is the difference between off-policy and on-policy?

  The ability to improve the policy without generating new samples from that policy (reuse old samples).

2 K-armed-bandit

Own self-test questions:

- What is the k-armed bandit problem?

  You can choose between k actions, each action returns stochastic reward whose distribution is unknown.

- How can we estimate action values?

  Monte-Carlo Simulation: $\mathbb{E}_{p(x)}[f(x)] = \int p(x)f(x)dx \approx \frac{1}{N}\sum_{x_i \sim p(x)} f(x_i)$

  For action values: $q^*(a) = \mathbb{E}[r_t|a_t = a] \approx \frac{\sum_{i=1}^{t-1}\mathbb{I}(a_i=a)}{\sum_{i=1}^{t-1}\mathbb{I}(a_i=a)} := q_t(a)$

  Use incremental update rule so that not all rewards need to be saved: $q_{n+1} = q_n + \alpha_n \cdot (r_n - q_n)$

- Why is greedy action selection not sufficient?

  We can easily get stuck in a sub-optimal policy if we have bad estimates for good actions.

- What is exploration and exploitation?

  Exploration: Improve knowledge for long-term benefit / Exploitation: Exploit knowledge for short term benefit.

- What is the exploration/exploitation trade-off?

  We want an optimal policy but need to stay uncertain because our estimates are uncertain.

- How do we solve this trade-off?

  Choose action ε-greedy, add entropy, intitalize the values optimistically or use upper confidence bounds to choose action.

- What is the goal of maximum entropy?

  Find trade-off between exploitation (high q-values) and exploration (high entropy): $\pi_t^* = \text{argmax}_\pi \sum_a \pi(a)q_t(a) + \tau^{-1}H(\pi) = \text{argmax}_\pi \sum_a \pi(a)(q_t(a) - \tau^{-1}\log\pi(a))$

- How does optimistic value intialization work?

  Optimistic Value Intialization adds positive bias to less explored actions and therefore enforces exploration.

- How do we add the upper confidence bound to our action selection?

  UCB Action Selection: $a_t = \text{argmax}_a q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}$ ($\frac{1}{\sqrt{N_t(a)}}$ proportional to standard error, $\sqrt{\log t}$ ensure that we explore sufficiently)

- Why is the maximum entropy policy also called as the soft-max or Boltzmann policy?

  Optimal policy of the rule results in soft-max: $\pi^*(a) = \frac{\exp(\tau q_t(a))}{Z} = \frac{\exp(\tau q_t(a))}{\sum_{a'}\exp(\tau q_t(a'))}$

Wrap up:

- The most simple RL problem: Bandits

- How to estimate action values from data
- Basic update rules for doing so incrementally
- Why greedy action selection is not sufficient
- What the exploration/exploitation trade-off is
- ... and simple ways to solve this trade-off:
  - Max-entropy objective (results in a soft-max policy)
  - Optimistic Value Intialization
  - Optimism in the face of uncertainty

3 MDPs / Optimal Decision Making

- The definition of an MDP.

  A set of states $s \in S$, a set of actions $a \in A$, a transition distribution $p(s'|s,a)$, a reward function $r(s,a)$, a distribution $\mu_0(s)$ over the start state, maybe a terminal state.

  Goal: Find a policy that maximizes expected cumulative future reward $J = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k}]$

- Why do we need discounting?

  Trade-off between optimizing the long-term ($\gamma \to 1$) and short-term ($\gamma \to 0$) reward.

  Helps our algorithms converge.

- The definition of the optimal and the policy based V and Q Functions.

  $V^*(s) = \max_\pi \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$

  $Q^*(s,a) = \max_\pi \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a]$

  $V^\pi(s) = \mathbb{E}_\pi[G_t | s_0 = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s] = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi(s'))$

  $Q^\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a')$

- What is the Bellman principle of optimality?

  The value function of every optimal policy has to satisfy the following condition: $V^*(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^*(s'))$

- How the value iteration algorithm works.

  Use the Bellman principle of optimality to arrive at a converged value function for the optimal policy: $V_k^*(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_{k-1}^*(s'))$ (optimal value if the MDP ends in k more steps)

  Converges to optimal solution.

- How the policy iteration algorithm works.

  Policy evaluation: calculate value function for some fixed policy until convergence.

  Policy improvement: Update policy using one-step look-ahead with value function.

  Converges to optimal solution.

- What are the conceptual differences?

  Policy iteration computes policy explicitly, value iteration implicitly.

  Value iteration is the extreme case of policy iteration where we stop policy evaluation after one update.

- What are the limitations of these approaches?

  Only works in discrete systems or linear systems with quadratic reward and gaussian noise (LQR).

  Solving $\max_a Q^*(s, a)$ is difficult in continuous actions spaces.

  Solving $\mathbb{E}_{\mathcal{P}}[V^*(s')|s, a]$ is difficult for arbitrary functions V and models.

4 Value-Based Methods

- What we mean by Monte Carlo estimates.

  Dynamics model unknown $\rightarrow$ need to estimate from samples.

  First visit Monte Carlo: average return following first visit to state s. Compute moving average: $V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha R_t$

  Limitations: Returns are very noisy, method is sample inefficient and we have to wait until episode end.

- Why TD learning can be seen as a combination of MC and Dynamic Programming.

  TD learning incorperates the target value for sampled action and transition into monte carlo moving average: $y_t = r(s_t, a_t) + \gamma V^\pi(s_{t+1}),\ V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha y_t$

  Less noisy, more sample efficient (only one sample needed to update estimate).

- What is the TD error?

  The difference between the old Q-Value estimate and the new Q-Value estimate.

  Results from TD-Learning for the Q-Function:

  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t$

  $\delta_t = r(s_t, a_t) + \max_{a'} \gamma Q(s_{t+1}, a') - Q(s_t, a_t)$

- How Q-Learning works.

  Tabular Q-Learning: During each iteration, an action is sampled according to an exploration policy. The temporal difference error is calculated and the Q-Function is updated.

  Off policy method: learns optimal Q-function (due to max operator) and not the Q-function of the policy generating the transitions.

- What is Value function approximation?

  Use parametrized Q or V function instead of table (for state/action spaces that are too big for tabular function). Can be linear function in features or complicated neural net.

  Fit with Monte-Carlo returns: Directly minimize regression loss: $L = \sum_t (V_\beta(s_t) - R_t)$ (sample inefficient)

  Q-Learning with neural networks: Minimize squared loss for the targets $y_t = r(s_t, a_t) + \gamma \max_{a'} Q_\beta(s_{t+1}, a')$. This results in the following update rule: $\beta_{\text{new}} = \beta + \alpha\delta_t \frac{d}{d\beta} Q_\beta(s_t, a_t)$

  Fitted Q-Iteration

- Why is Q-Learning not actual gradient ascent?

  Because the targets are always changing: Changing β may also change the target values y_t.

- How to fix Q-learning for deep neural networks?

  Problem: Sequential states are strongly correlated $\rightarrow$ leads to catastrophic forgetting $\rightarrow$ solved by replay buffers.

  Problem: targets are always moving $\rightarrow$ use older set of weights β' to compute targets. No proof of convergence but often works well.

  Alternative: Use moving average: $\beta' \leftarrow \tau\beta' + (1-\tau)\beta$

- Why do we get the overestimation effect and how to fix it?

  We choose the maximum of the possible actions according to our current Q-function. This is a noisy estimate of the real Q-function so the maximum value over all actions is much more likely to be above the true value than to be below it: $\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$.

  Solution: Use two different networks for action selection and value computation: If they are noisy in different ways, then there is no overestimation $\rightarrow$ use current network to evaluate action, use target network to evaluate value.

5 Policy Search

- What are the advantages/disadvantages of policy search vs value based methods?

Obtaining π may be simpler than Q or V (V doesn't prescribe actions and would need a dynamics model, Q needs to efficiently solve $\text{argmax}_a Q(s,a) \rightarrow$ challenging for continuous/high-dimensional action spaces).

Easier to use/tune, works better for continuous action spaces, but needs much more samples.

- What is the main idea of policy gradient algorithms?

  Use a parametric policy (stochastic so that we explore) and update parameters using gradient ascent based on expected return (on-policy).

- What kind of policies are used in discrete action and continuous action domains?

  Discrete: Neural Network softmax-policy: $\pi_\theta(a|s) = \frac{\exp(h_\theta(s,a))}{\sum_{a' \in A} \exp(h_\theta(s,a'))}$ with the DNN output for action a: $h_\theta(s,a)$

  Continuous: Gaussian Neural Network Policy: $\pi_\theta(a|s) = N(a|\mu_\theta(s), \Sigma_\theta(s))$ (mean and variance defined by DNN).

- How can we use the log ratio trick to compute the policy gradient?

  $\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) R(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d\tau$ (intractable integral $\rightarrow$ easy to evaluate expectation)

  Log-ratio trick: $\nabla_\theta \log p_\theta(\tau) = \frac{1}{p_\theta(\tau)} \nabla_\theta p_\theta(\tau) \Leftrightarrow \nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$

- Why can we compute gradients even if the reward or the dynamics are not differentiable?

  $p_\theta(\tau) = p(s_0) \Pi_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$ depends on policy and dynamics model.

  The derivative of the log of $p_\theta(\tau)$ means that the model-dependent terms can be ignored (which means we need no dynamics model) because they do not depend on our parameters: $\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

  We can approximate expectation of return with sampling even if reward is not differentiable.

- Explain the intuition of the REINFORCE update equation.

  Update equation: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i (\sum_t \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))(\sum_t \gamma^t r(s_{i,t}, a_{i,t}))$

  Gradient of log-likelihood of path is scaled by the return of that path $\rightarrow$ increases probability of paths with high returns and decreases probability of paths with low returns.

- Why do we need a baseline in policy gradient algorithms?

  A baseline reduces variance in the update and leaves gradient unbiased. With an appropriate baseline, all gradient vectors should point in a similar direction. Good baseline: Average reward.

  $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log p_\theta(\tau_i)(R_i - b)$

- Why is optimizing the advantage beneficial to optimizing the Q function?

  Advantage tells us how much better an action a is than the expected value of policy π in state s: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. This can be used as the weight in the REINFORCE update equation.

- How can we exploit temporal structure for policy gradients?

  Correlation between action selection strategy and past reward is zero $\rightarrow$ remove past rewards that do not depend on $a_{i,t}$

  Result: Policy gradient Theorem:
  $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i (\sum_t \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))(\sum_{k=t}^{T} \gamma^{k-t} r(s_{i,k}, a_{i,k}))$

- What is the role of the critic in actor critic methods?

  Weighting the gradient of the policy so that good actions are chosen more often (?)

- Which methods do you know for computing the critic?

  Single Sample estimate, n-step returns, generalized advantage estimation (GAE), directly learn the Q-function (?)

6 Trust Regions

- Why are trust regions important for policy gradients?

  We typically also learn the exploration noise of the policy and want to reduce that variance during training. Two problems: 1) Gradient magnitude varies a lot and gets larger with small variance (difficult to pick correct learning rate). 2) Exploration (covariance) decreases too quickly $\rightarrow$ converges to suboptimal solution

  Thus, we want to keep the new policy close to the old policy so that we don't jump to unexplored areas: $D(\pi(a|s), \pi_{\text{old}}(a|s)) \leq \varepsilon, \forall s \in S$

- Which divergence is typically used for the trust region and what are its properties?

  Kullback-Leibler divergence: $\text{KL}(p(x)||q(x)) = \int p(x) \log \frac{p(x)}{q(x)} dx$

  Always positive, 0 iff both distributions are equal, non-symmetric.

- How does Lagrangian optimization work and why is it often easier to solve the dual than solving the primal objective?

  We add the constraints as terms in the optimization problem and maximize over the λ (factor for the constraint). The minimization over x forces the x to behave according to the constraints because otherwise the maximization over λ could create infinitetally large results ($\lambda \geq 0$).

  Maybe easier because we can solve for each constraint seperately?

- How to implement trust-regions with discrete actions using Lagrangian optimization?

  Solution of the dual: $\eta^* = \text{argmax}_\eta \varepsilon\eta + \eta \log(\sum_a q(a) \exp(\frac{r(a)}{\eta}))$ s.t. $\eta > 0$.

  Can be solved with convex optimizers.

  $\pi^*(a) \propto q(a) \exp(\frac{r(a)}{\eta^*})$

- How are natural gradients connected to trust regions?

  They use a Taylor approximation of the trust region problem. 1st order Taylor for objective and 2nd order Taylor for constraint.

- How is the Fisher Information Matrix used in the natural gradient algorithms?

  Appears in 2nd order Taylor for constraint: $\text{KL}(p_{\theta_{\text{old}}+g} || p_{\theta_{\text{old}}}) \approx g^T F g \leq \varepsilon$

  $F = \frac{d\text{KL}(p_\theta || p_{\theta_{\text{old}}})}{d\theta d\theta}|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{p_{\theta_{\text{old}}}}[\nabla_\theta \log p_\theta(x) \nabla_\theta \log p_\theta(x)^T]$ (Zero and first order terms of Taylor expansion are 0 for $\theta = \theta_{\text{old}}$).

- How can the natural gradient be computed (in two ways)?

  Cannot construct KL-constraint for every state $\rightarrow$ introduce relaxed constraint (expectation of KL over state distribution for π_{old}) $\rightarrow$ can still be approximated with FIM (with monte carlo estimate of expectation over state space), which can be computed in closed form for Gaussians (not covered).

  FIM can also be computed from samples $\rightarrow$ Compatible Function Approximation (CFA)

- What is compatible function approximation and how is it connected to natural gradients?

  Compatible Function Approximation is the least squares solution to approximating the advantage using $\nabla_\theta \log \pi_\theta(a|s)$ as features. The function approximator $\tilde{A}_w(s,a) = \nabla_\theta \log \pi(a|s)^T w$ of the advantage function is called compatible function approximator as it uses features that are compatible to the policy representation.

- What does the natural gradient update look like using compatible function approximation?

  Compute FIM from samples: $g_{NG} = \eta^{-1} F^{-1} \nabla_\theta J \approx \eta^{-1} (\Phi^T \Phi)^{-1} \Phi^T A = \eta^{-1} w^*$ with $\Phi_i = \nabla_\theta \log \pi(a_i|s_i)^T$, $A_i = \hat{A}^\pi(s_i, a_i)$

  Update gradient: $\theta_{k+1} = \theta_k + \eta^{-1} w^*$

- When is the natural gradient update implementing the exact trust region update?

  For log-linear policies (policies that are linear in their parameters when taking the log $\rightarrow$ exponential family distributions). Natural gradient for log-linear policies is advantage reweighted old policy $\rightarrow$ equivalent to closed form trust region solution.

- Why are natural gradients difficult to use for bigger networks?

  2nd order algorithms do not scale well (TRPO, CFA).

- What is the main idea of PPO (clipped version) and what are the benefits?

  Proximal Policy Optimization (PPO):

  Dual Descent: No closed form solution exists for Lagrangian of constrained optimization problem. However, can be solved approximately by dual descent: Maximize Lagrangian for fixed η w.r.t θ and minimize for fixed θ w.r.t η. Benefit: needs no second order information (FIM).

  Clipping: $J_{\text{clip}}(\theta) = \sum_i \min(w_i(\theta)\hat{A}_i, \text{clip}(w_i(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_i)$ with $w_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\text{old}}(a_i|s_i)}$: We enforce trust region through clipping of gradient instead of extra constraints. This makes objective pessimistic about performance far away from θ. Benefit: works as well as PPO with KL penalty, but is simpler to implement and can be optimized by standard DNN optimizers (Adam).

  SOTA in on-policy RL.

7 Off-Policy Actor Critic

- Why are off-policy methods more efficient then on-policy methods?

  You can reuse samples of previous policies, even if they are very different from the current policy.

- How can we achieve off-policy RL with continuous actions?

  DDPG, TD3, SAC: Like Q-Learning, but does off-policy learning about the current policy and how to locally improve it (vs. directly learning about the optimal policy)

  Learn the argmax operator as the (deterministic) policy.

  qt-Opt: Simplest solution: random shooting: sample actions $(a_1, \ldots, a_N)$ from some distribution (e.g., uniform). Works ok for up to 40 dimension but needs a lot of computation.

- What are the different options to optimize the actor?

  Stochastic actor update: Traditional methods using critic $Q_\beta(s,a)$ for advantage. Optimize policy gradient objective, arbitrarly many actions can be sampled because we don't need to execute them on the system:

  $\nabla_\theta J_{\text{PG}}(\theta) = \mathbb{E}_{(s,a)\sim\pi(a|s)}[\nabla_\theta \log \pi_\theta(a|s)(Q_\beta(s,a) - V^{\pi_{\text{old}}}(s_i))]$
  $\approx \frac{1}{N}\sum_{s_i}^{N} \mathbb{E}_{a\sim\pi(\cdot|s_i)}[\nabla_\theta \log \pi_\theta(a|s_i)(Q_\beta(s_i,a) - V^{\pi_{\text{old}}}(s_i))]$

  Deterministic actor update: DDPG, TD3

  Deep Deterministic Policy Gradient: Learn deterministic policy that approximates the max operator: $\pi(s) \approx \text{argmax}_a Q_\beta(s,a)$. Actor objective: $J_{\text{DDPG}}(\theta) = \mathbb{E}_{s\sim\mu^\pi(s)}[Q_\beta(s, \pi_\theta(s))]$.

Variational actor update: SAC

- Why do we have a bias in actor critic algorithms and how to fix it?

  Since our approximated policy π_{approx} is trained on the approximate objective Q_β, we will be worse than we think we are: Assuming $\mathbb{E}[Q^\pi(s, \pi_{\text{true}}(s))] \approx \mathbb{E}[Q^\pi(s, \pi_{\text{approx}}(s))]$ then $\mathbb{E}[Q_\beta(s, \pi_{\text{approx}}(s))] \geq \mathbb{E}[Q^\pi(s, \pi_{\text{approx}}(s))]$

  Fix: Maintain two critics β_1, β_2 and two policies θ_1, θ_2. Use θ_1 to get targets for Q-function β_2 and vice versa: $y_{1,i} = r_i + \gamma Q_{\beta_1'}(s_i, \pi_{\theta_2'}(s_i))$

  TD3 only has one policy and uses pessimistic value.

  pp 28

- What type of gradient is DDPG using and why can we not apply the same gradient for the sampled return?

  Uses gradient for policy search. Applies chain rule (use gradient information of Q to do update): $\nabla_\theta J_{\text{DDPG}}(\theta) = \mathbb{E}_{s\sim\mu^\pi(s)}[\frac{\partial Q_\beta}{\partial a}(s, a = \pi(s))\frac{\partial \pi}{\partial \theta}(s)]$

  Policy parameters are not included in Q if we sample.

- What is the objective of max-ent reinforcement learning and why is that useful?
- How to we obtain the policy update in SAC?
- What is the reparametrization trick and when should it be preferred to the likelihood policy gradients?

8 Black Box Optimization

- What are the advantages/disadvantages of an episode-based RL formulation?
- How is episode-based RL connected to black-box optimiatzion/evolutionary strategies (ES)?
- How can we use search distributions for black-box optimization?
- What is the difference between first-order and second-order methods for optimizing search distributions? Pros/Cons?
- What does a simple policy gradient algorithm for the episode-based case look like?
- What are the qualitative differences to "deterministic" gradient descent using finite differences?
- Why do most ES use rankings instead of fitness values?
- How does the cross-entropy method work and what is its connection to evolutionary strategies.

- How to use trust-regions for episode-based RL?
- Why can we use exact trust-regions in this case (as opposed to lecture 6)?
- What does compatible function approximation for Gaussians look like?
- How can we optimize for search distributions that work for multiple task settings/-contexts?

9 Variational Inference (only wrap-up)

1. Variational Inference is a genral tool for approximating distributions:
   - Use the I-projection due to its favourable generative properties
   - Mode-seeking property is essential (compare to mode-averaging property of M-projection)
2. Probabilistic Control can be stated as inference problem...
   - And solved by computing the I-projection to the posterior trajectory distribution
   - The resulting solutions is equivalent to max-ent RL
3. VI can be directly applied to learning latent variable models
   - Learns posterior distribution over latent variables
   - Allows for more efficient (more directed) sampling of the latent variable
4. Variational Inference with GMMs
   - Can represent any target distribution
   - In RL: We can learn multi-modal solutions
   - Mixture of expert models can represent versatile behavior

10 Imitation Learning

None