Reinforcement Learning Algorithms and Equations

Robert J. Moss MOSSR@CS.STANFORD.EDU
Stanford University, Stanford, CA 94305

This write-up is intended as a collection of common algorithms and equations in reinforcement learning, deep reinforcement learning, decision making under uncertainty, approximate dynamic programming, and stochastic optimization methods.

1 Reinforcement Learning

Reinforcement learning is a form of machine learning that maximizes the cumulative discounted reward an agent receives from taking actions in an environment.[1]

[1] R. S. Sutton, ''Dyna, an Integrated Architecture for Learning, Planning, and Reacting,'' *SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

1.1 Bellman Equation

The *Bellman equation*[2] recursively computes the value of a decision problem.[3] The next state s' comes from taking action a in state s using the transition model T, namely $s' = T(s, a)$.

[2] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[3] https://en.wikipedia.org/wiki/Bellman_equation

$$\underbrace{U(s)}_{\text{state value}} \leftarrow \max_a \overbrace{\left(\underbrace{R(s,a)}_{\text{reward}} + \underbrace{\gamma U(s')}_{\substack{\text{discounted value} \\ \text{of next state}}}\right)}^{\text{estimate of optimal discounted value}} \tag{1}$$

1.2 Q-learning

The core of the *Q-learning* algorithm[4] is the Bellman equation.[5] Q-learning is model-free and does not rely on the transition model T and the reward model R, but use samples of reward r and the next state s'.

[4] C. J. C. H. Watkins, ''Learning from Delayed Rewards,'' PhD thesis, University of Cambridge, 1989.

[5] https://en.wikipedia.org/wiki/Q-learning#Algorithm

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning} \\ \text{rate}}} \cdot \overbrace{\underbrace{\left(\underbrace{r}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount} \\ \text{factor}}} \cdot \underbrace{\max_{a'} Q(s',a')}_{\substack{\text{estimate of optimal} \\ \text{future values}}}\right)}_{\text{new value (temporal difference target)}} - \underbrace{Q(s,a)}_{\text{old value}}}^{\text{temporal difference}} \tag{2}$$

1.3 Sarsa

The *Sarsa* algorithm[6] is a modification of Q-learning that uses (s, a, r, s', a'). It uses the actual next action a' to update Q instead of maximizing over all actions.

[6] G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*. University of Cambridge, 1994, vol. 37.

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning} \\ \text{rate}}} \cdot \overbrace{\underbrace{\left(\underbrace{r}_{\text{reward}} + \underbrace{\gamma Q(s',a')}_{\substack{\text{discounted} \\ \text{next value}}}\right)}_{\substack{\text{new value} \\ \text{(temporal difference target)}}} - \underbrace{Q(s,a)}_{\text{old value}}}^{\text{temporal difference}} \tag{3}$$

2 *Dynamic Programming*

Optimal strategies can be found using a computation technique called *dynamic programming*.[7] Dynamic programming relies on optimal substructures of the problem to reuse already computed information.

[7] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

2.1 *Policy Evaluation*

Policy evaluation computes the expected utility U obtained from executing a policy π for n steps (in the finite horizon case with discount γ).[8]

[8] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

$$U_t^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U_{t-1}^\pi(s') \tag{4}$$

function PolicyEvaluation(π, n)
  $U_0^\pi(s) \leftarrow 0$ for all states s
  for $t \leftarrow 1$ **to** n
    $U_t(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U_{t-1}^\pi(s')$ for all s
  return U_n

Algorithm 2.1. Policy evaluation.

2.2 *Policy Iteration*

Policy iteration computes the optimal policy π^* using policy evaluation.[9] Starting with any policy π_0, policy iteration performs two steps:

[9] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

1. *Policy evaluation*: given the current policy, compute the value.
2. *Policy improvement*: using the value, compute a new policy.

Policy improvement uses the value function returned from policy evaluation.

$$\pi_{k+1}(s) = \arg\max_a \left(R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) U^{\pi_k}(s') \right) \tag{5}$$

function PolicyIteration($\pi_0, k \leftarrow 0$)
  repeat
    Compute U^{π_k} through PolicyEvaluation
    $\pi_{k+1}(s) = \arg\max_a \left(R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) U^{\pi_k}(s') \right)$ for all states s
    $k \leftarrow k + 1$
  until $\pi_k = \pi_{k-1}$
  return π_k

Algorithm 2.2. Policy iteration.

2.3 *Value Iteration*

Value iteration uses dynamic programming and the Bellman equation to compute the optimal *value* (or *utility*) U iteratively.[10]

[10] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

$$U^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) U^*(s') \right) \tag{6}$$

The optimal policy π can be extracted using the value function.

$$\pi^*(s) \leftarrow \arg\max_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) U^*(s') \right) \tag{7}$$

function ValueIteration
  $k \leftarrow 0$
  $U_0(s) \leftarrow 0$ for all states s
  repeat
    $U_{k+1}(s) \leftarrow \max_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) U_k(s') \right)$ for all states s
    $k \leftarrow k + 1$
  until convergence
  return U_k

Algorithm 2.3. Value iteration.

2.4 *Asynchronous Value Iteration (Gauss-Seidel Value Iteration)*

Asynchronous value iteration updates the value for only a subset of states per iteration.[11] An example would be *Gauss-Seidel value iteration* that iterates through some ordering of the states and updates the values in place.[12]

[11] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[12] D. P. Bertsekas, "A New Value Iteration Method for the Average Cost Dynamic Programming Problem," *SIAM Journal on Control and Optimization*, vol. 36, no. 2, pp. 742–759, 1998.

$$U(s) \leftarrow \max_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) U(s') \right) \tag{8}$$

2.5 *Local Approximation Value Iteration*

For large or continuous state spaces, *approximate dynamic programming* is useful in finding approximately optimal polices for the substructure of the problem.[13] For a finite set of states, a function $\beta(s)$ is used to weigh "nearby" states, where $\sum_{i=1}^{n} \beta_i(s) = 1$. The value λ_i of each state s_i is used to approximate the value of an arbitrary state as $U(s) = \sum_{i=1}^{n} \lambda_i \beta_i(s) = \boldsymbol{\lambda}^\top \beta(s)$.

[13] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

function LocalApproximationValueIteration
  $\boldsymbol{\lambda} \leftarrow \mathbf{0}$
  loop
    $u_i \leftarrow \max_a \left(R(s_i,a) + \gamma \sum_{s'} T(s' \mid s_i,a) \boldsymbol{\lambda}^\top \beta(s') \right)$ for $i \leftarrow 1$ to n
    $\boldsymbol{\lambda} \leftarrow \mathbf{u}$
  return $\boldsymbol{\lambda}$

Algorithm 2.4. Local approximation value iteration.

3 *Deep Reinforcement Learning*

Deep reinforcement learning combines deep learning (i.e. neural networks) with reinforcement learning algorithms (e.g., Q-learning) and can scale to larger problems.[14]

[14] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

3.1 *Trust Region Policy Optimization*

Trust region policy optimization (TRPO)[15] is a policy gradient method that guarantees policy improvement.[16]

$$\underbrace{\mathcal{L}_\pi(\tilde{\pi})}_{\text{loss}} = \overbrace{\eta(\pi)}^{\substack{\text{expected}\\ \text{discounted reward}}}_{\substack{\uparrow\\ \text{stochastic}\\ \text{policy}}} + \sum_s \overbrace{\rho_\pi(s)}^{\substack{\text{visitation}\\ \text{frequency}}} \underbrace{\sum_a \tilde{\pi}(a \mid s)}_{\substack{\text{advantage}\\ \text{improving policy}}} \overbrace{A_\pi(s,a)}^{\text{advantage}} \tag{9}$$

[15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *CoRR*, vol. abs/1502.05477, 2015. arXiv: 1502.05477.

[16] Advantage function:

$$A(s,a) = Q(s,a) - U(s)$$

3.2 *Actor-Critic with Experience Replay*

Actor-critic with experience replay (ACER)[17] uses batch off-policy updates to improve stability with optional trust region updates.[18]

$$\underbrace{Q^{\text{ret}}(s,a)}_{\substack{Q\text{-value}\\ \text{retrace}}} = \overbrace{\underbrace{r}_{\substack{\downarrow\\ \text{reward}}} + \underbrace{\gamma\bar{\rho}'}_{\substack{\text{discounted}\\ \text{next truncated}\\ \text{importance weight}}} \left[Q^{\text{ret}}(s',a') - \underbrace{Q(s',a')}_{\substack{\text{estimate}\\ \text{of } Q^\pi}} \right]}^{\text{temporal difference}} + \underbrace{\gamma U(s')}_{\substack{\text{discounted}\\ \text{state value}}} \text{ where } \overbrace{\bar{\rho} = \min(c,\rho)}^{\substack{\text{truncated}\\ \text{importance weight}}}, \underbrace{\rho = \frac{\pi(a \mid s)}{\mu(a \mid s)}}_{\substack{\text{importance weight}\\ (\text{behavior policy } \mu)}} \tag{10}$$

$$\underbrace{\hat{g}^{\text{policy}}}_{\substack{\text{policy gradient}\\ (\text{w/ bias correction})}} = \overbrace{\bar{\rho}\nabla \log \pi(a \mid s)}^{\substack{\text{importance weighted}\\ \text{gradient of log-policy}}} \left[\overbrace{\underbrace{Q^{\text{ret}}(s,a)}_{\substack{Q\text{-value}\\ \text{retrace}}} - \underbrace{U(s)}_{\text{state value}}}^{\text{retrace advantage}} \right] + \underbrace{\mathop{\mathbb{E}}_{\tilde{a}\sim\pi} \left(\overbrace{\left[\frac{\rho(a) - c}{\rho(a)} \right]}^{\substack{\text{correction}\\ \text{weight}}} \overbrace{\nabla \log \pi(\tilde{a} \mid s)}^{\substack{\text{log-policy}\\ \text{gradient}}} \overbrace{\left[Q(s,\tilde{a}) - U(s) \right]}^{\text{advantage estimate}} \right)}_{\text{bias correction}} \tag{11}$$

[17] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, et al., "Sample Efficient Actor-Critic with Experience Replay," *CoRR*, vol. abs/1611.01224, 2016. arXiv: 1611.01224.

[18] https://nervanasystems.github.io/coach/components/agents/policy_optimization/acer

3.3 *Proximal Policy Optimization*

Proximal policy optimization (PPO)[19] is a deep reinforcement learning algorithm that implements a trust region[20] update compatible with stochastic gradient descent.[21]

$$\underbrace{\mathcal{L}^{\text{clip}}(\underbrace{\theta}_{\substack{\downarrow\\ \text{policy}\\ \text{parameters}}})}_{\text{loss}} = \underbrace{\hat{\mathbb{E}}_t \left[\min(\underbrace{r_t(\theta)}_{\substack{\text{probability ratio of}\\ \text{new and old policy}}} \hat{A}_t, \underbrace{\text{clip}}_{\substack{\downarrow\\ \text{gradient}\\ \text{clipping}}}(r_t(\theta), 1-\underbrace{\epsilon}_{\substack{\uparrow\\ \text{hyperparameter}}}, 1+\epsilon) \overbrace{\hat{A}_t}^{\substack{\downarrow\\ \text{advantage}\\ \text{estimate}}}) \right]}_{\text{empirical expectation over time}} \tag{12}$$

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347.

[20] D. C. Sorensen, "Newton's Method with a Model Trust Region Modification," *SIAM Journal on Numerical Analysis*, vol. 19, no. 2, pp. 409–426, 1982.

[21] https://openai.com/blog/openai-baselines-ppo

4 Online Search

4.1 Monte Carlo Tree Search

Monte Carlo tree search (MCTS)[22] is an anytime algorithm that uses rollouts of a random policy to estimate the value of each state-action node in the tree.

[22] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *International Conference on Computers and Games*, 2006.

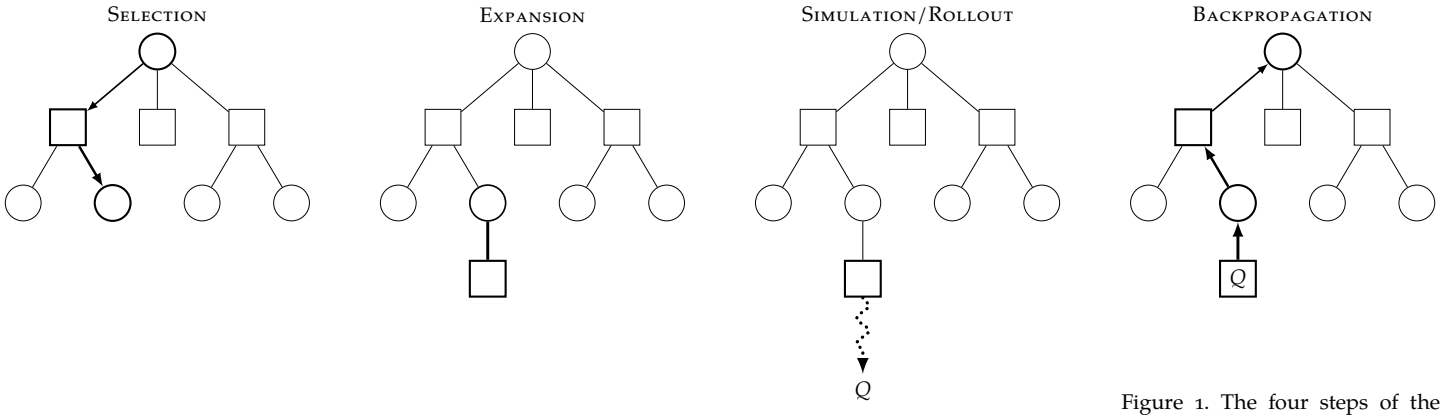

Figure 1. The four steps of the Monte Carlo tree search algorithm.

function MonteCarloTreeSearch(s, d)
  loop Simulate(s, d, π_0)
  return $\arg\max_{a \in A(s)} Q(s,a)$ ▷ select action

Algorithm 4.1. Top-level Monte Carlo tree search algorithm.

function Simulate(s, d, π_0)
  if $d = 0$ **return** 0
  if $s \notin \mathcal{T}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{s\}$
    $(N(s,a), Q(s,a)) \leftarrow (N_0(s,a), Q_0(s,a))$ for all $a \in A(s)$
    return Rollout(s, d, π_0)
  $a \leftarrow \arg\max_a Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$ ▷ selection (UCT)
  $(s', r) \sim G(s,a)$ ▷ expansion
  $q \leftarrow r + \gamma$Simulate($s', d-1, \pi_0$) ▷ simulation/rollout
  $N(s,a) \leftarrow N(s,a) + 1$
  $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$ ▷ backpropagation
  return q

Algorithm 4.2. Monte Carlo tree search simulation.

function Rollout(s, d, π_0)
  if $d = 0$ **return** 0
  $a \sim \pi_0(s)$
  $(s', r) \sim G(s,a)$
  return $r + \gamma$ Rollout($s', d-1, \pi_0$)

Algorithm 4.3. Rollout evaluation.

5 Stochastic Optimization

Stochastic optimization methods use randomness for minimization or maximization.[23] They are often useful when dealing with black-box functions where no gradient information is available.

[23] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.

5.1 Cross-Entropy Method

The *cross-entropy method* (CEM)[24] aims to minimize the cross-entropy between the unknown true distribution f and a proposal distribution g parameterized by θ. This technique reformulates the minimization problem as a probability estimation problem, and uses adaptive importance sampling to estimate the unknown expectation.

[24] R. Rubinstein, "The Cross-Entropy Method for Combinatorial and Continuous Optimization," *Methodology and Computing in Applied Probability*, vol. 1, no. 2, pp. 127–190, 1999.

Algorithm 5.1. The cross-entropy method.

function CrossEntropyMethod($S, g, m, m_{\text{elite}}, k_{\max}$)
  for $k \leftarrow 1$ **to** $k_{\max}$
    $\mathbf{X} \sim g(\cdot \mid \theta_k)$ where $\mathbf{X} \in \mathbb{R}^{|g| \times m}$ ▷ draw m samples from g
    $\mathbf{Y} \leftarrow S(\mathbf{x})$ for $\mathbf{x} \in \mathbf{X}$ ▷ evaluate samples $\mathbf{X}$ using objective S
    $\mathbf{e} \leftarrow$ store top m_{elite} from $\mathbf{Y}$ ▷ select elite samples output from objective
    $\theta_{k'} \leftarrow \text{Fit}(g(\cdot \mid \theta_k), \mathbf{e})$ ▷ re-fit distribution g using elite samples
  return $g(\cdot \mid \theta_{k_{\max}})$

References

1. K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
2. R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
3. D. P. Bertsekas, "A New Value Iteration Method for the Average Cost Dynamic Programming Problem," *SIAM Journal on Control and Optimization*, vol. 36, no. 2, pp. 742–759, 1998.
4. R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *International Conference on Computers and Games*, 2006.
5. M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.
6. M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.
7. R. Rubinstein, "The Cross-Entropy Method for Combinatorial and Continuous Optimization," *Methodology and Computing in Applied Probability*, vol. 1, no. 2, pp. 127–190, 1999.
8. G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*. University of Cambridge, 1994, vol. 37.
9. J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *CoRR*, vol. abs/1502.05477, 2015. arXiv: 1502.05477.
10. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347.
11. D. C. Sorensen, "Newton's Method with a Model Trust Region Modification," *SIAM Journal on Numerical Analysis*, vol. 19, no. 2, pp. 409–426, 1982.
12. R. S. Sutton, "Dyna, an Integrated Architecture for Learning, Planning, and Reacting," *SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.
13. Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample Efficient Actor-Critic with Experience Replay," *CoRR*, vol. abs/1611.01224, 2016. arXiv: 1611.01224.
14. C. J. C. H. Watkins, "Learning from Delayed Rewards," PhD thesis, University of Cambridge, 1989.