Reinforcement learning is one of the branches of machine learning, in which the system is trained to perform a desired input-output mapping by applying a reward signal that acts to reinforce or weaken the system’s association between a given input and output.

Reinforcement learning is of particular interest because of it allows for the creation of control systems in novel environments, where the details of the optimal policy are not known, or where the data necessary to apply supervised learning is not available.

Reinforcement learning problems are typically viewed as Markov decision processes, these are a special case of decision processes that possess the Markov property. That is, the future evolution of the process depends only on the current state.

There exists a set of states , and a set of actions.

Time progresses in discrete steps. At each time-step t, the agent receives a state and reward , then takes an action at ∈ A. The process then transitions to a new , chosen stochastically with a probability dependent on and . In the context of reinforcement learning, the manner in which an MDP transitions between states is often called the “transition model”.

The problem of training an agent with the best possible performance on a control problem can be framed as finding a policy that maximises the accumulated reward as t → ∞

Stationary problems VS non-stationary problems (harder).

A non-stationary problem is one in which the transition model changes as the process progresses (significantly harder).

There are a variety of ways to classify reinforcement learning methods.

Consider first, the structure of the output of the model. Some of the popular methods:

**Policy-Based.** The model learns to estimate a probability spread of actions for a given state. This can be sampled from to enact a policy.

**State-Value-Based.** The model learns to estimate the value of a given state. This alone is not useful in constructing a policy. However if the transition model if the process is also known, state-values can be used to construct policies i.e. by calculating the weighted average value of the successor states reached by each action. The value of a state is often called the “V-value”, and expressed as a function of the form .

**State-Action-Value-Based.** The model learns to estimate the value of each action for a given state. Unlike a state-value-based approach, this can be used directly to construct a policy by selecting actions with high values. The value of state-action pairs is often called the “Q-value”, and expressed as a function of the form .

**Actor-Critic.** Two model has two components; the actor, and the critic. The actor is policy-based. The critic is value based, either V-values or Q-values can be used. The critic is not used to construct a policy, its output is used to direct the actor during the learning process. Only the actor’s output is necessary to enact the learned policy.

The method by which training data is gathered is also an important factor in the final behaviour of agents. Note that these classifications are orthogonal to the output-structural methods, any combination of the two is possible.

**Monte-Carlo.**

The agent is allowed to interact with the environment until it reaches a terminal state (a full episode) without any learning occurring. Once the episode is over, the model parameters are updated according to the total reward received during the episode. This means that Monte-Carlo methods can only be applied to problems with episodes of finite length.

**Temporal Difference.**

The agent is allowed to interact with the environment. Model parameters are updated according to the reward received during each transition between states. This allows for learning over the course of a single episode, which can quicken convergence.

**Dynamic Programming.**

In contrast to the two prior strategies, in dynamic programming the agent does not interact with the environment. Instead data is gathered by iterating the state space (in an arbitrary order), using the transition model to explore all possible transitions from that state, and updating the model parameters according to the reward. DP requires a finite state and action space, as well as knowledge of the transition model, it cannot be applied to many real-world problems.

There are a number of non-drl approaches; tabular methods, linear regression.

Tabular approaches achieve the desired input-output mapping by constructing a look-up table containing every possible input state, which is held in memory. Tabular methods are backed by theoretical proofs of their ability to converge to optimal policies [3, 4]. Despite this useful property, such methods are not typically practical due to the large state-spaces of real-world machine-learning problems. The associated table would be so large that storing it and updating it during training would be computationally infeasible.

Modern approaches typically use neural networks to approximate TODO. While perceptrons have been proven to converge to the optimal policy over time [5], neural networks in general are not guaranteed to converge, or even to improve. Though in practice they often do.

Methods based on gradient descent can find only the local optimum.

The use of neural networks with hidden layers to solve reinforcement learning problems is known as deep reinforcement learning, and in the last decade it has been applied to achieve human-level performance in complex control problems [6, 7].

Modern RL encompasses a diverse variety of approaches, and there are a variety of overlapping categories by which reinforcement learning algorithms can be classified. The taxonomy of those approaches is of interest if they are to be meaningfully compared.

**Entropy**

The entropy of a policy captures the variance in the actions it prescribes. The entropy of a policy w/r to a particular state is given by:

[1]

Where is the action space, and is the function that gives the probability of the network taking action in state . The policy entropy can be added to the reward at each step of the training process to produce a new target for optimization. Doing so can discourage models from reaching locally optimal policies that prescribe the same action in all, or nearly all states.

*Table 1: Summary of RL Methods*

|  |  |  |
| --- | --- | --- |
| Class | Technique | Notes |
| Q-Value | Deep Q-Learning. | These two are very similar. |
| SARSA |
| V-Value |  |  |
| Policy-Based | REINFORCE |  |
| Proximal Policy Optimisation |  |
| Trust Region Policy Optimisation |  |
| Actor-Critic | Basic Actor-Critic | Original algorithm described in the 1999 paper, with no optimisations. |
| Advantage Actor-Critic (A2C) |  |
| Adversarial Advantage Actor-Critic |  |

**REINFORCE.**

This is a policy-based Monte-Carlo method, one of the earliest policy-based methods to be discussed. At the end of each epoch, the network parameters are updated according to the rule.

[2]

Where and are hyperparameters of the network, is the length of the episode, and is the cumulative reward received during the episode.

If the reward is positive, it is desirable to reinforce this behaviour, so the weights are updated to maximize the probability mass function, making the action more likely to be taken in the future. If it’s negative, they are updated in the opposite direction.

**REINFORCE-MENT**

This is a variant of REINFORCE. Instead of maximising reward, it targets the sum of reward & policy entropy, discouraging the adoption of low entropy policies.

**SARSA/Q-Learning**

These are closely related temporal-difference methods.

Let  be the function that outputs the estimated sum of future rewards from , parameterized by θ.

A transition is a tuple

By adding the observed reward to the system’s predicted reward from , we can produce a more accurate prediction of the value of which can be compared to the predicted value for to produce an error that can be minimized by gradient descent.

**Q-Learning Loss Function**

**SARSA Loss Function**

The methods differ in that SARSA is on-policy, whereas Q-learning is off-policy. That is, under SARSA is the action prescribed by the current policy, whereas under Q-Learning is the action with the greatest predicted future reward. These could differ i.e. under an ε-greedy policy, the action with the maximal Q-value would be chosen only of the time.

**Actor-Critic.**

There are two components of the model; the actor & the critic.

The critic learns a value function using the same method as the value-based techniques discussed earlier. Note that as the transitions are being produced according to the policy of the actor, any learning must be off-policy.

The actor learns the greedy policy for the critic’s value function, actor weight updates are made according to the rule.

Where is some estimate of the of the quality of the action chosen by the actor. There are multiple ways to produce such an estimate. In the regular Basic Actor-Critic, the network learns the Q-values of each action, which are used as C. In A2C, the critic learns the V-value function, and this is used to calculate the advantage of each action, which is used as C.

The actor & critic can be two separate networks, or a single network. In the latter, the network has outputs for both action probabilities and C.

**Advantage Actor-Critic (A2C).**

This is a variant of actor-critic that uses the advantage function. The advantage function captures the value of an action relative to other possible actions.

As advantage is calculated using the state/action history, this is a Monte-Carlo method.

This is a synchronous variant of an earlier algorithm (Asynchronous Advantage Actor-Critic).

**PPO**

Works by going through a single trajectory/epoch, trains in an n-step manner every n steps,

REFERENCES

# References

|  |  |
| --- | --- |
| [1] | R. J. Williams and J. Peng, “Function Optimization Using Connectionist Reinforcement Learning Algorithms,” *Connection Science,* no. 3, pp. 241-268, 1991. |
| [2] | R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning,* vol. 8, p. 229–256, 1992. |
| [3] | V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint arXiv:1312.5602,* 2013. |
| [4] | A. Novikoff, “On Convergence Proofs For Perceptrons,” *Proceedings of the Symposium on the Mathematical Theory of Automata,* vol. 12, no. 1, pp. 615-622, 1963. |
| [5] | P. Dayan and C. J. Watkins, “Q-Learning,” *Machine Learning,* vol. 8, pp. 279-292, 1992. |
| [6] | M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” *Proceedings of the AAAI conference on artificial intelligence,* vol. 32, no. 1, 2017. |
| [7] | V. Konda and J. Tsitsiklis, “Actor-Critic Algorithms,” *Advances in neural information processing systems,* vol. 12, 1999. |
| [8] | R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Second Edition, MIT Press, 2018. |
| [9] | M. Volodymyr, A. P. Badia, M. Mirza, A. Graves, T. Harley, L. P. Timothy, S. David and K. Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning,” *arXiv preprint arXiv:1602.01783,* 2016. |