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.

Will now describe...

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).

In value-based methods the agent learns to estimate the value of the actions it can take in a given state, and the associated policy can be retrieved by choosing the action with the highest estimated value. Under a policy-based approach the agent learns to produce a probability distribution of actions at each state, which can be sampled from to enact the learned policy [1]. A third class of agents use actor-critic methods, which combine elements of both approaches [2].

In policy-based approaches, the model learns to estimate a probability spread of actions for a given state. This can be sampled from to enact a policy.

In state-value-based approaches, 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.

In state-action-value-based approaches 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.

In actor-critic methods, two models are used; the actor, and the critic. The actor is policy-based. The critic is value based, either state-value or state-action-value (TODO MAYBE?). The critic is not used to construct a policy, its output is used to train the actor to learn the final policy.

There are a number of primitive 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.

One improtxploration methods.

**Monte-Carlo Methods.**

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 Methods.**

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 Methods.**

The agent does not interact with the environment. Instead the data is gathered by iterating the state space, using the transition model to explore all possible transitions from that state, and updating the model parameters according to the reward. As 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.

*Table 1: Summary of RL Methods*

|  |  |  |
| --- | --- | --- |
| Class | Technique | Notes |
| Value-Based | Deep Q-Learning. | Off-policy temporal difference method. |
| SARSA | On-policy temporal difference method. |
| Policy-Based | REINFORCE | Runs a full episode, uses the future returns of each decision to improve the policy by gradient ascent. Eligibility traces. |
| 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 |  |
| Adversarial Advantage Actor-Critic |  |

**REINFORCE**

This is a policy-based approach. It’s a Monte-Carlo method, this means that weight updates are performed at the end of each episode according to the rule. For each time step in the episode, the network parameters are updated according to the rule.

[1]

Where and are hyperparameters of the network, is the length of the episode, is the reward received during the episode, and is the function giving the probability of the network taking action in state

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.

**SARSA/Q-Learning**

These are closely related temporal-difference methods.

Let  be the function that outputs the 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 networks; the actor & the critic. The critic learns the q-function according to the rule. (this is a TD method as well).

and the actor learn the greedy policy for the actor’s value function.

TODO insert actor learning rule here.

Describe the difference between regular, advantage, A3C.

**PPO**

REFERENCES

# References

|  |  |
| --- | --- |
| [1] | R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning,* vol. 8, p. 229–256, 1992. |
| [2] | 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. |
| [3] | P. Dayan and C. J. Watkins, “Q-Learning,” *Machine Learning,* vol. 8, pp. 279-292, 1992. |
| [4] | 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. |
| [5] | 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. |
| [6] | V. Konda and J. Tsitsiklis, “Actor-Critic Algorithms,” *Advances in neural information processing systems,* vol. 12, 1999. |
| [7] | R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Second Edition, MIT Press, 2018. |