Actor-Critic Reinforcement Learning for Classic control problems

Aadesh Neupane
aadeshnpn@byu.edu
Brigham Young University

Abstract

Reinforcement learning (RL) has recently shown great promise in solving difficult learning task, and achieved above human level performance in promising tasks like Atari games (Mnih et al. 2013), Go (Silver et al. 2016) and poker (Revell 2017). RL has two broad classes of algorithms: Actor and Critic, with their own merits and demerits. This paper explores the benefits of combining them into Actor-Critic algorithms and demonstrates their ability in solving classic control problems. We use temporal difference learning class of algorithms as critic and policy gradient as actor. Our implementation of Actor-Critic algorithms was able to solve the classic Cartpole problem (Brockman et al. 2016) in 96 time steps. As these algorithms are model-free, it can be use to solve almost any type of RL problems with very few tweaks.

The intent of this project was to explore RL and apply actor-critic based algorithms to solve classic control problems. RL is inspired by behaviorist psychology and is concerned with how agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The information that the agent receives from the environment is not guaranteed to be complete. But the agent takes action based on the current state of the environment and receives feedback (rewards), which signifies the correctness of the agent's sequence of actions in the past. Thus, the objective of reinforcement learning is to find the best sequence of actions which maximizes the long term reward.

RL is a bit different than standard supervised and unsupervised machine learning approaches as it doesn't have access to correct input/output pair and correct actions are not labeled. Moreover, the environment is stochastic and the agent has only partial or limited observability of the environment. Thus, RL problems are recognized to be more difficult to solve than other supervised learning problems.

Introduction

Most of the environment in RL is modeled as Markov Decision Process (MDP). MDP is extension of Markov chains, which follows Markov property : if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. MDP model time-discrete stochastic state-transition with five tuples (S, P, A, R, π), S is a set of states, P is a set of possible transition functions, A is a set of actions, R is a set of rewards received while transition from one state to another and π is a policy mapping which takes state and output action to take in that state (Bellman 1957). An agent chooses an actions $a_t \in A$ which causes transition from state s_t to some successor state s_{t+1} with probability $P(s, a, s')$. Then the agent receives reward $r \in R$ for taking action a_t in state s_t.

The agent that interacts with the MDP is modeled in terms of a policy π. A deterministic policy $\pi : S \rightarrow A$ is a mapping from the set of states to the set of actions. Applying π means always selecting actions $\pi(s)$ in state s. This is a special case of a stochastic policy, which specifies a probability distribution over A for each state s, where $\pi(s, a)$ denotes that probability to choose action a in state s.

The goal of RL is to find a policy that maximizes the accumulated sum of rewards. One approach to find the optimal policy that maximizes the accumulated sum of rewards is by solving MDP using Bellman equation. Below is a general algorithm to calculate this optimal policy which are defined recursively as follows:

$$\pi(s) := argmax_a\{\sum_x P_a(s, s')(R_a(s, s')+\gamma U(s'))\} \quad (1)$$

$$U(s) := \sum_{s'} P_{\pi(s)}(s, s')(R_{\pi(s)}(s, s') + \gamma U(s')) \quad (2)$$

In Bellman equation, the π function is not used; instead, the value of $\pi(s)$ is calcualted with $U(s)$ whenever needed (Heidrich-Meisner et al. 2007). Substituting the calculation of $\pi(s)$ into the calculation of $U(s)$ gives us the bellman equation.

$$U(s) := max_a\{\sum_{s'} P_a(s, s')(R_a(s, s') + \gamma U(s'))\} \quad (3)$$

Since, we are interested in learning a policy that accumulates as much reward as possible i.e. we are interested in finding a policy π^* that outperforms all other policies. It has been shown that such an optimal policy always exists although it need not be unique.

RL algorithms can be classified into critic-only, actor-only and actor-critic methods where each class can be further divided into model-based and model-free algorithms, depending on whether the algorithm needs or learns explicitly transition probabilities and expected reward for state-action pair.

Critic-only RL

This class of algorithms are based on the idea of first finding the optimal value function and then deriving an optimal policy from this value function (mapping from state to expected reward). A simple approach to learn value function is through dynamic programming. For a finite space, the Bellman equation yields a finite set of linear equations, which can be solved using standard methods. We can modify the Bellman equation to define a learning algorithm called value iteration, which determines the optimal value function. Value iteration is a model based approach, as it makes explicitly use of model of environment, which is given by the transition probabilities P and the expected rewards R. To overcome this disadvantage of value iteration, Temporal Difference (TD) learning comes to the rescue. TD based algorithms estimates the value function and derive an optimal policy from a set of observable quantities: transition from state s to state s' with action a and immediate reward r that was received (Sutton 1988). In TD learning approach, we get the new estimate of the utility by summing up the old estimate for that state with the estimation error. Equation 4 defines the update rule for TD algorithm.

$$U(s_t) \leftarrow (U(s_t) + \alpha[r_{t+1} + \gamma U(s_{t+1}) - U(s_t)] \quad (4)$$

But equation 4 doesn't take past experience into consideration. Although, TD based algorithms converge towards the optimal value function, the rate of convergence is slow due to the fact that only a single state-action pair is updated per time step (Watkins and Dayan 1992) (Tesauro 1992). The concept called eligibility trace is introduce to speed up learning which suggests to update value function not only for the previous state but for the states which occurred earlier in the trajectory (Singh and Sutton 1996) (Främling 2007). The basic TD approach works based on the active scenario with on-policy learning i.e we start to find the optimal policy by using the equation 4 starting with a random policy. The Q-learning algorithm is different than previous algorithms as it uses off-policy learning. In off-policy learning the policy π is updated based on the observation of a second policy μ which is not updated. This strategy allows to learn by observation i.e learn about optimal policy while following an exploratory policy. Equation 5 defines the update rule for Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (5)$$

Most of the algorithms discussed above are for discrete values sets and we assumed that the values can be stored in a table. But in practical problems, state space would become huge or infinite which would be impossible to store in a table based representation. Hence, value function are represented by some function approximators like Neural Networks, Decision Tree, Nearest Neighbor, Linear Combination of features or grid-based function approximators which can also generalize unseen states (Sutton 1996). Some of the popular algorithms that fall into critic-only category are Monte-Carlo methods, TD, Q-Learning, Sarsa and others.

Actor-only RL

This class of algorithms are based on the idea of directly finding the optimal policy function without building value function i.e. these algorithms search directly in policy space. It usages parametrized policy to directly estimate the gradient of the return based on simulation with respect to the parameter of the actor, and updates these parameters in a direction of improvement without using value functions. Mostly used actor-only reinforcement learning algorithm is policy gradient (Sutton et al. 1999). A possible drawback of such methods is that the gradient estimators may have a large variance. Furthermore, as the policy changes, a new gradient is estimated independently of past estimates. Hence, there is no learning in the sense of accumulation and consolidation of older information. Some of the popular algorithms that fall into actor-only category are REINFORCE (Williams 1992) and Evolutionary optimization algorithms.

Actor-Critic RL

Actor-critic methods combine the advantages of actor-only and critic-only methods. While the parameterized actor brings the advantage of computing continuous actions without the need for optimization procedures on a value function, the critics merit is that it supplies the actor with low-variance knowledge of the performance. More specifically, the critics estimate of the expected return allows for the actor to update with gradients that have lower variance, speeding up the learning process. The lower variance is traded for a larger bias at the start of learning when the critics estimates are far from accurate. Actor-critic methods usually have good convergence properties, in contrast to critic-only methods (Kimura and Kobayashi n.d.) (Sutton et al. 1999).

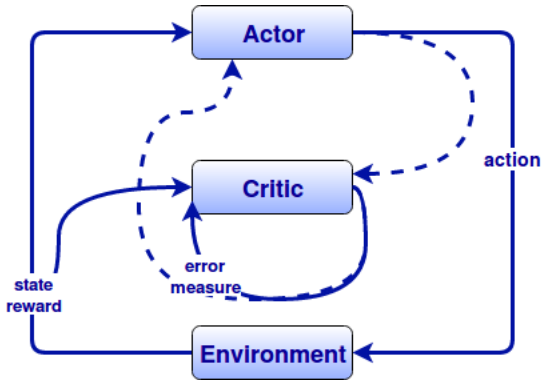

Figure 1: General actor-critic architecture

Problem

The task that we want to solve using our algorithm is known as the CartPole problem. This control problem was described (A. G. Barto, Sutton, and Anderson 1983) in details. In simple terms, this problem can be described as "A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a

force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. CartPole-v0 and CartPole-v1 environment differ only in terms of maintaining average reward over 100 consecutive trails. To solve CartPole-v0 , the agent needs to get average reward of 195 over 100 consecutive trails whereas to solve CartPole-v1, the agent needs get average reward of 475 over 100 consecutive trails.

The remainder of the paper is organized as follows. We start with related work in the first section. Then in the implementation section, we will introduce and discuss our model; in the results section we will visualize the results obtained from different experiments and in the conclusion we will summarize what we gained this project.

Related Work

(Houk, Adams, and A. Barto 1995) explained how doapmine neurons in the basal ganglia might acquire the ability to predict reinforcement, and how outputs from these neurons might then be used to reinforce behaviours. This research paved a path for the further research on actor-critic RL. (Tsitsiklis, Van Roy, et al. 1997) showed that for linear function approximators, convergence is guaranteed if states are sampled according to the steady-state probabilities. (Konda and Tsitsiklis 1999) purposed variations of actor-critic algorithm and provided an overview of convergence proof. (Joel, Niv, and Ruppin 2002) expanded (Houk, Adams, and A. Barto 1995) actor-critic models of the basal ganglia more in terms of computational perspectives. (GOLKHOU, LUCAS, and PARNIANPOUR 2004) used actor-critic RL in real world application for controlling sagittal arm during oscillatory movements. (Bhatnagar et al. 2009) introduce an improvement called Natural actor-critic algorithms to tradition actor-critic algorithms. All actor-critic algorithm use some variant of policy gradient methods. In policy-gradient methods, the policy is taken to be an arbitrary differentiable function of a parameter vector $\in R^d$. Given some performance measure $J : R^d \rightarrow R$, we would like to update the policy parameter in the direction of the gradient.

$$\Delta\theta \propto \Delta_\theta J(\theta) \tag{6}$$

The gradient is not directly available of course, but observed values can be used to construct unbiased estimators of it; estimators that can be used in a stochastic approximation of the actual gradient. This is the basic idea behind all policy-gradient reinforcement learning methods. Specific to natural policy gradient, they follow an intuition that the policy updated should be invariant to bijective transformation i.e. a change in parameterization should not affect the result of the policy update. Moreover, use of Natural gradient leads to simple and computationally effective algorithms. Actor-critic algorithms are being successfully applied for beam setup, a peg-in-hole insertion task, biped locomotion, robot arm to learn motor skills, four legged robot to walk, underwater cable tracking and many more as shown by (Grondman et al. 2012). We were inspired to implement actor-critic RL after we read about its implementation for brain-machine interfaces using neuro-biological feedback (Prins, Sanchez, and Prasad 2014).

Implementation

Since RL doesn't have standard datasets like MNIST (LeCun, Cortes, and Burges 1998) or IMAGENET (Russakovsky et al. 2015), we are using the openAI reinforcement learning platform (Brockman et al. 2016) to quantify our algorithm. At first, we started developing a simple grid world virtual environment to model MDP. In this environment, the agent will move through the grids to accumulate maximum rewards avoiding traps. We started solving the problem by implementing simple Temporal Difference (TD) approach. As Q-learning is critic only RL algorithm, we implemented it as a part of actor-critic RL. After that, we applied the same Q-learning and TD algorithm implementation to the Cartpole control problem. Sadly, it didn't work in this case though both were working perfectly on grid world. After analysis, we found out that our implementation of both algorithms could only handle discrete states but the Cartpole problem has continuous states i.e both of the algorithms were using general table based approach. For those algorithms to work on continuous states, we needed to modify our lookup table based approach to function approximation approach (Doya 2000). There are different function approximation approach like sparse-coarse-coded (CMACs) (Sutton 1996), Tile and Kanerva coding (Wu and Meleis 2009), Radial Basis Function (RBF) (Kretchmar and Anderson 1997), Fourier Basis (Konidaris and Osentoski 2008), Neural Networks (Sutton et al. 1999) and others which can be used instead of the traditional table-based approach. Since both of the implementation didn't work, we tried to solve the Cartpole problem using a simple hill climbing algorithm. To our surprise, it worked quite well.

At first, we started with a MLP as a function approximator and then used RBF as a creative experiment with our algorithm. Implementing MLP with function aproximator made complete sense as MLP are supposed to be universal function approximators (Hornik 1991). Then we implemented policy gradient method based on finite difference. (Peters and Schaal 2006). Policy gradient strategies assume a differentiable structure on a predefined class of stochastic policies and ascent the gradient. After policy gradient method worked for the Cartpole problem, we started combining algorithms to build our model as shown in figure 2. The algorithm that we have used is detailed on figure 3.

We tried implementing the TD algorithm with an eligibility trace for the critic part but it didn't work. After investing significant time, we found out that eligibility trace approach is not straighforward with function approximators (Främling 2007). We found out that instead of eligibility trace approach, experience replay approach is better in case of continuous states (Adam, Busoniu, and Babuska 2012). Our implemented model as of figure 3 was able to solve both versions of Cartpole of the openAI Gym environment. Then, we decided to switch to different function approximators and

observe its performance. First we tried using logistic classifier but it was not able to solve either of the problems. Then we used RBF which was able to solve CartPole-v0 but took large number of episodes to converge. One of the reason for RBF network took large amount to converge was due to how the network is structured to cluster similar things together. We tried our actor-critic model to solve Ping-Pong environment. We preprocessed the input frame and resized the frame to feed into our algorithm. After waiting till 10000 episodes we terminated our program as we thought it would not solve this environment. But latter on we found out that it requires more than 100,000 episodes to show some improvements in that environment.

Model

The model architecture that we have applied is inspired from (Prins, Sanchez, and Prasad 2014).

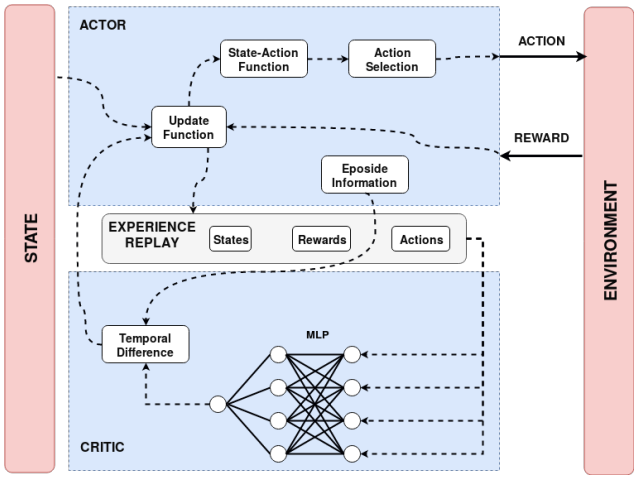

Figure 2: Actor-Critic Model

Algorithm

We first observe the first few episodes recording all the observation and rewards made. During an observation time of 200 time steps, each time step, an actor takes optimal action using the output from simple MLP and saves the next state and reward. Then, total reward of the episode is calculated by summing the individual rewards. If the episode is complete then we move on to next episode whereas we check if the current state is in the replay list or not and add it if it doesn't exist. Then we update replay memory and return the current episode states and rewards to the critic for assessment. Using these values, critic will calculate value function using MLP for each state in episodes. If the observing episodes has passed, actor updates the policy using the value function from critic and also critic updates the value estimates. In case of policy update from actor, the weights of the neural nets is updated using the value function. In case of value estimates in critic, for each batch from replay memory, weights will be optimized using MLP.

```
procedure Actor():
    for each time steps or till episode is over
        Choose action from the action-state function appxomitor given current state
        Observe the next_state, reward by applying the actions
        Accumulate the total rewards
        if state, next_state not already exists in reply
            Store all information current and next state in seperate lists
        else
            update rewards from replay memory
    update experience memory
    return episodes information

procedure Critic(episode information):
    for each states in the episode
        calculate value using TD learning
    return combined value function for the episode

Initialize environment, actor, critic, episodes

For each episodes
    episode_info <- Actor()
    Critic(episode_info)
    update state-action function using the TD value from the critic
    update value function
```

Figure 3: Actor-Critic Algorithm

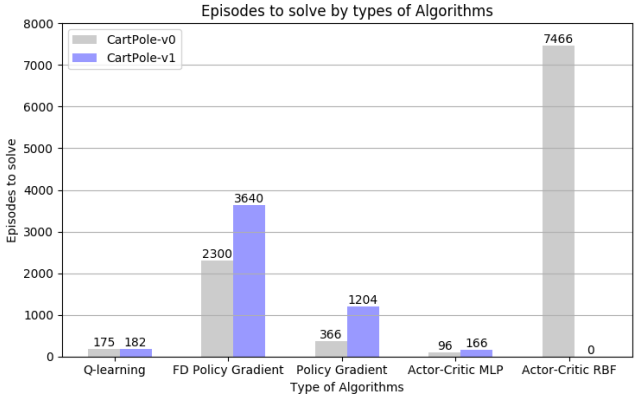

Figure 4: Performance of algorithms on CartPole problems

Results

We ran five variants of RL algorithms for the Cartpole problem. Actor-Critic with MLP did the best among those by solving CartPole-v0 in 96 episodes and CartPole-v1 in 166 episodes. Actor-Critic with RBF only solved CartPole-v0 taking 7466 episodes but didn't solve CartPole-v1 within 10000 episodes. Figure 4 visualizes the results obtained for the experiments.

Conclusion

We studied many Reinforcement Learning algorithms and literature in course of this project. We would like to experiment with Natural policy gradient as future work. We realized the importance of function approximation for continuous state problems and saw the potent of actor-critic reinforcement learning to solve real world complex problems effectively.

References

Adam, Sander, Lucian Busoniu, and Robert Babuska (2012). "Experience replay for real-time reinforcement learning control". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.2, pp. 201–212.

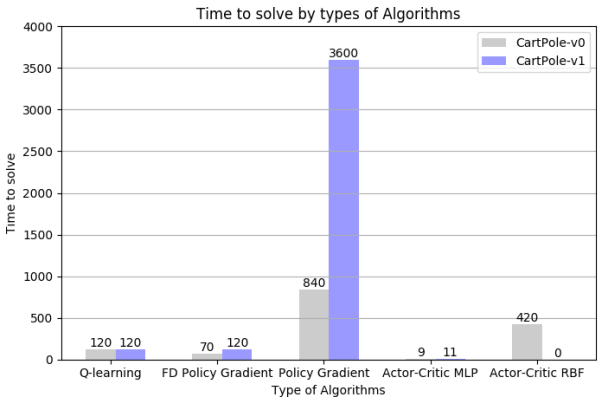

Figure 5: Time taken by algorithms on CartPole problems

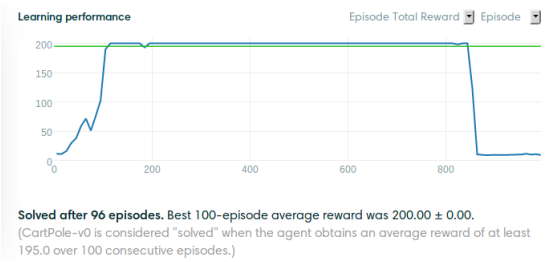

Figure 6: Actor-Critic performance on CartPole-v0 openAI

Barto, Andrew G, Richard S Sutton, and Charles W Anderson (1983). "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5, pp. 834–846.

Bellman, Richard (1957). *A Markovian decision process*. Tech. rep. DTIC Document.

Bhatnagar, Shalabh et al. (2009). "Natural actor-critic algorithms". In: *Automatica* 45.11.

Brockman, Greg et al. (2016). *OpenAI Gym*. eprint: arXiv:1606.01540.

Doya, Kenji (2000). "Reinforcement learning in continuous time and space". In: *Neural computation* 12.1, pp. 219–245.

Främling, Kary (2007). "Replacing eligibility trace for action-value learning with function approximation." In: *ESANN*, pp. 313–318.

GOLKHOU, VAHID, CARO LUCAS, and MOHAMAD PARNIANPOUR (2004). "Application of actor-critic reinforcement learning method for conrol of a sagittal arm during oscillatory movement". In: *Biomedical Engineering: Applications, Basis and Communications* 16.06, pp. 305–312.

Grondman, Ivo et al. (2012). "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6, pp. 1291–1307.

Heidrich-Meisner, Verena et al. (2007). "Reinforcement learning in a nutshell." In: *ESANN*. Citeseer, pp. 277–288.

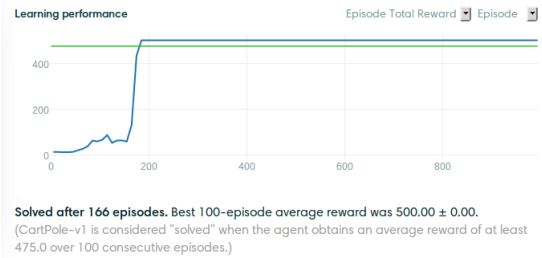

Figure 7: Actor-Critic performance on CartPole-v1 openAI

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2, pp. 251–257.

Houk, JC, JL Adams, and AG Barto (1995). *A Model of How the Basal Ganglia Generate and Use Neural Signals that Predict Reinforcement, Models of Information Processing in the Basal Ganglia (eds. JC Houk, JL Davis and DG Beiser), 249/270*.

Joel, Daphna, Yael Niv, and Eytan Ruppin (2002). "Actor–critic models of the basal ganglia: New anatomical and computational perspectives". In: *Neural networks* 15.4, pp. 535–547.

Kimura, Hajime and Shigenobu Kobayashi. "An Analysis of Actor/Critic Algorithms Using Eligibility Traces: Reinforcement Learning with Imperfect Value Function." In:

Konda, Vijay R and John N Tsitsiklis (1999). "Actor-Critic Algorithms." In: *NIPS*. Vol. 13, pp. 1008–1014.

Konidaris, George and Sarah Osentoski (2008). "Value function approximation in reinforcement learning using the Fourier basis". In:

Kretchmar, R Matthew and Charles W Anderson (1997). "Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning". In: *Neural Networks, 1997., International Conference on*. Vol. 2. IEEE, pp. 834–837.

LeCun, Yann, Corinna Cortes, and Christopher JC Burges (1998). *The MNIST database of handwritten digits*.

Mnih, Volodymyr et al. (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Peters, Jan and Stefan Schaal (2006). "Policy gradient methods for robotics". In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, pp. 2219–2225.

Prins, Noeline W, Justin C Sanchez, and Abhishek Prasad (2014). "A confidence metric for using neurobiological feedback in actor-critic reinforcement learning based brain-machine interfaces". In:

Revell, Timothy (2017). *AI takes on top poker players*.

Russakovsky, Olga et al. (2015). "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3, pp. 211–252.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Singh, Satinder P and Richard S Sutton (1996). “Reinforcement learning with replacing eligibility traces”. In: *Recent Advances in Reinforcement Learning*, pp. 123–158.

Sutton, Richard S (1988). “Learning to predict by the methods of temporal differences”. In: *Machine learning* 3.1, pp. 9–44.

– (1996). “Generalization in reinforcement learning: Successful examples using sparse coarse coding”. In: *Advances in neural information processing systems*, pp. 1038–1044.

Sutton, Richard S et al. (1999). “Policy gradient methods for reinforcement learning with function approximation.” In: *NIPS*. Vol. 99, pp. 1057–1063.

Tesauro, Gerald (1992). “Practical issues in temporal difference learning”. In: *Machine learning* 8.3-4, pp. 257–277.

Tsitsiklis, John N, Benjamin Van Roy, et al. (1997). “An analysis of temporal-difference learning with function approximation”. In: *IEEE transactions on automatic control* 42.5, pp. 674–690.

Watkins, Christopher JCH and Peter Dayan (1992). “Q-learning”. In: *Machine learning* 8.3-4, pp. 279–292.

Williams, Ronald J (1992). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4, pp. 229–256.

Wu, Cheng and Waleed Meleis (2009). “Function approximation using tile and Kanerva coding for multi-agent systems”. In: *Proc. of adaptive learning agents workshop (ala) in aamas*.