Novelties in Intelligent Digital Systems
C. Frasson et al. (Eds.)

doi:10.3233/FAIA210070

Forced ε-Greedy, an Expansion to the ε-Greedy Action Selection Method

George ANGELOPOULOS[a] and Dimitris METAFAS[a]
[a] *Department of Electrical and Electronic Engineering, University of West Attica, Athens Greece*

Abstract. Reinforcement Learning methods such as Q Learning, make use of action selection methods, in order to train an agent to perform a task. As the complexity of the task grows, so does the time required to train the agent. In this paper Q Learning is applied onto the board game Dominion, and Forced ε-greedy, an expansion to the ε-greedy action selection method is introduced. As shown in this paper the Forced ε-greedy method achieves to accelerate the training process and optimize its results, especially as the complexity of the task grows.

Keywords. Q Learning, action selection, ε-greedy, forced ε-greedy

1. Introduction

Reinforcement Learning (RL) methods such as Q Learning, Monte Carlo Trees and deep Q Learning have become popular over the past few years. The goal of RL is to train an agent to make the best decisions in order to complete a specific task, which could be as simple as getting out of a maze in the fastest way possible, or as complex as finding the best strategy in order to win a board game. Many researchers use board games to test new methods of RL. Backgammon [1], the game of go [2], [3], Othello [4], [5], the settlers of Catan [6], [7] and other classic board games, were used to create AI agents that evolved strategies through playing the actual game, instead of their choices being controlled by a handwritten deterministic code. Strategies developed by RL agents can be so successful, that they are being adopted worldwide by human players, as demonstrated by G. Tesauro [1].

Not only must the agent be successful in completing the desired task, but the training process must also be as fast as possible. The choice of the action selection method during the training can be crucial in achieving the best possible results, as well as how fast those results are being achieved. The most popular action selection methods are ε-greedy and softmax [8]. Quite a few attempts have been made in order to improve those methods. Some of them are adaptive ε-greedy [9], [10], temporally-extended ε-greedy [11], and ε-BCM [12], which is a Bayesian Ensemble Approach to ε-greedy.

Some different approaches of RL have been tested on the board game Dominion. Researchers used Monte Carlo trees [13], [14], [15], others combinations of artificial

neural networks [16], and even balancing cards sets [17] was researched. In this paper Q Learning is applied on the card game Dominion. Our goal is to train an agent able to oppose and dominate over AI opponents, who use popular strategies developed by human players. We also test a new expansion to ε-greedy method, defined as the Forced ε-greedy, in order to accelerate the training. The agent trained with this new method not only achieves better wining rate, than the one trained with the classic ε-greedy method, but the training process is much faster.

2. Reinforcement Learning

The purpose of reinforcement learning (RL), is to train an agent to perform a task, through interaction with the environment and by using the trial and error method. Firstly we create a virtual episode of the problem, and define the environment as everything else, except for the agent in training. The environment's current state (s) is described by several variables, which are provided to the agent. Then the agent has to make a choice between all possible actions $a(s)$ given the current state of the environment. The choice is made based on a policy $\pi(s,a)$. After the action is performed, the agent receives a numerical reward (r), as well as the environment's new state (s'). The task of the agent is to find an optimal policy $\pi'(s,a)$, in order to maximize the cumulative rewards. To do so the agent, at the beginning of the training, assigns a random value $Q(s,a)$ to every state-action pair, which equals to the expected reward if he takes action a, when the state is s. Every time the agent finds itself to the same state s, taking the same action a, he updates the value of $Q(s,a)$, for that state-action pair. After many iterations, and after passing all possible states, the agent will have an optimal approximation of the true values of the $Q(s,a)$ function. The best policy $\pi'(s,a)$ then is proven [8] to be the greedy policy, in which the agent always selects the action with the highest $Q(s,a)$ value, among all possible actions, in order to maximize the cumulative reward.

2.1. Exploration versus Exploitation

When an agent follows the greedy policy, it is considered an exploitation of current knowledge, in order to maximize the reward. This policy works well when the values of $Q(s,a)$ function are the correct ones. But at the start of the training we assign random values to the $Q(s,a)$ function. If the agent follows the greedy policy during the training, he will never explore some actions, which have lower Q values than others, therefore he may not find the optimal solution to the problem.

During the training process it is better to use a policy that allows the agent to explore alternative actions, even if they have low Q values. It is not wise though to let the agent make random choices, because then the training time would be huge. It would be better for the agent to focus on the actions with higher Q values, in order to get a better approximation of them, but occasionally to explore, in order to find better actions that have been assigned low Q values by chance.

Such action selection methods, which match those criteria, are ε-greedy and softmax. The ε-greedy method states that the agent chooses the greedy action with probability $1-\varepsilon$, and a random action with probability ε, where ε is a number between zero and one. When ε equals zero the agent becomes greedy, and when ε equals one the agent always chooses random actions. According to the softmax method, the agent

chooses an action α with probability $p(a)$ shown in Eq. (1), where τ is a positive parameter called temperature and n the number of possible actions in the current state of the environment. In the limit where $\tau \to 0$ the agent becomes greedy, while higher temperatures make all the actions equiprobable.

$$p(\alpha) = \frac{e^{Q(a)/\tau}}{\sum_{b=1}^{n} e^{Q(b)/\tau}} \quad (1)$$

2.2. Temporal Difference Learning

Temporal Difference learning (TD) is a method of updating the values of $Q(s,a)$ function. Temporal difference learning methods are divided into on policy (S.A.R.S.A.) and off policy (Q Learning). The difference between them is that in S.A.R.S.A. we estimate the Q values for the policy the agent uses to make choices, while in Q Learning we estimate the Q values for the greedy policy, regardless which policy the agent uses. In both methods, a virtual episode of the process is created, and random values $Q(s,a)$ are set, for every state-action pair. The agent finds themselves at the initial state s_0, and selects the action to take between all possible actions, either by the ε-greedy or by the softmax method.

In S.A.R.S.A., at every time step of the episode the agent finds the environment in a state s and selects an action a, based on the $Q(s,a)$ value. Then the agent receives a reward r as well as the environment's new state s'. The agents then selects another action a' with $Q(s',a')$ value and updates $Q(s,a)$ as shown in Eq. (2), where e is the learning rate and γ is the discount factor. This process is repeated until the episodes terminate. After the simulation of a large number of episodes the estimates of the Q values will converge to the true values.

$$Q(s,a) = Q(s,a) + e \cdot [r + \gamma \cdot Q(s',a') - Q(s,a)] \quad (2)$$

During Q Learning the update of $Q(s,a)$ is quite different. Instead of waiting until the next action a', the agent updates $Q(s,a)$ immediately after taking an action, using the maximum Q value of all the possible actions a' at the new state s', as shown in Eq. (3).

$$Q(s,a) = Q(s,a) + e \cdot [r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)] \quad (3)$$

3. Dominion Board Game

Dominion is a card game, consisting of three types of cards, treasure cards, victory cards and kingdom cards. Treasure cards are the currency of the game providing coins when played. There are three kinds of treasure cards, copper that provide one coin, silver that provide two coins and gold that provide three coins. Victory cards cannot be played, but they are worth points, and acquiring as many as possible of them, is the way to win. Estates worth one point, duchies worth two points, provinces worth three points, while curses worth minus one point. There are many different kingdom cards, with each one having a different effect when played.

Each player starts the game with a deck of ten cards, seven coppers and three estates. The remaining treasure and victory cards are placed in piles, and ten piles of kingdom

cards are created, each consisting of ten cards. The players draw five cards from their decks, and take turns, starting with the first player. Each turn has three phases, action phase, buy phase and clean-up. At the action phase the active player can play one kingdom card from his hand. Kingdom cards have various effects, as card drawing, providing additional actions or buys, etc. At the buy phase, the active player can buy one card, or more if he has gained additional buys at the action phase, paying the cost of the bought cards by playing treasure cards from his hand. Finally at the clean-up phase, the active player discards his hand, as well as all the cards he played and bought, and draws five new cards from his deck. If the deck runs out of cards, he shuffles the discarded cards to create a new deck. The players keep taking turns until the provinces pile is empty, or three or more other piles are empty, whichever comes first, and the game ends. The player with the most points is the winner.

3.1. JDominion

A training and testing environment was created, written entirely in Java, called JDominion. The application allows the creation of two, three or four player games, and the option to train using the Q Learning method or test the agent. The ten kingdom cards, which were integrated in the application, as well as a small description and their prices are shown in Table 1**.** An AI opponent was created in order to train and test the agent, Money. The Money policy buys only silver and gold treasure cards, until two gold cards are present in the deck, from that point on it is also possible to buy victory cards. The money policy was also used by Winder [16], to train and test his agent.

Table 1. Kingdom Cards

Kingdom Card	Description	Price
Festival	+2 actions, +1 buy, +2 coins	5
Chapel	Thrash up to four cards from your hand	2
Bazaar	+2 actions, +1 card, +1 coin	5
Adventurer	Reveal cards from your deck, until you draw two treasure cards	6
Conspirator	+2 coins, if you played three or more actions +1 card, +1 action	4
Smithy	+3 cards	4
Moneylender	Thrash a copper from your hand, if you do +3 coins	4
Village	+1 card, +2 actions	3
Woodcutter	+1 buy, +2 coins	3
Workers village	+1 card, +2 actions, +1 buy	4

The JDominion application, not only trains the agent, but also tests the efficiency of the trained agent, alongside with the training. After every 10.000 games, the training process stops temporarily, and 10.000 testing games are conducted, using the same number of players as the training games. During the testing games the agent uses the greedy policy, according to how the values of the Q function have shaped so far. The results of the training and the testing games are stored, as well as various other statistics,

like the average number of each card bought during the game, the maximum and the minimum number of each card bought etc. The application also stores the current values of the Q function, the values of the Q function for which the testing games had the best results, and the number of times the agent has found himself to every specific state-action pair during the training. The JDominion application can be found online at https://sourceforge.net/projects/jdominion/files/.

4. Results

To define the state of the environment several variables were used as shown in Table 2. For every variable we used an upper bound, in order to diminish the total number of different state-actions pairs. Two sets of training games were conducted, the first with the agent against one AI opponent (two players game) and the second with the agent against three AI opponents (four players games).That is because in four player games the provinces pile, tend to exhaust faster, causing the game to end in a lower number of turns than the two players games. So the strategy the agent evolved in the slow two player games, isn't proved to be successful in the fast four player games, and vice versa. Also we chose different upper bound for four players games and for two players games .The upper bound in each case, was chosen as the rounded up average of every variable, which we found during preliminary training games [18].

Table 2 Variables used to describe the environment

	Four players games		**Two players games**	
Variables	**Upper bound**	**Number of states**	**Upper bound**	**Number of states**
Number of turns	*12*	*13*	*19*	*20*
Copper cards in deck	*8*	*2*	*8*	*2*
Silver cards in deck	*4*	*5*	*6*	*7*
Gold cards in deck	*2*	*3*	*4*	*5*
Festivals in deck	*1*	*2*	*1*	*2*
Chapels in deck	*1*	*2*	*1*	*2*
Bazaars in deck	*1*	*2*	*1*	*2*
Adventurers in deck	*1*	*2*	*1*	*2*
Conspirators in deck	*1*	*2*	*1*	*2*
Smithies in deck	*1*	*2*	*2*	*3*
Moneylenders in deck	*1*	*2*	*1*	*2*
Villages in deck	*1*	*2*	*1*	*2*
Woodcutters in deck	*1*	*2*	*1*	*2*
Workers villages in deck	*1*	*2*	*1*	*2*
Possible buys		*18*		*18*
Total number of states		*7.188.480*		*38.707.200*

The purpose of the training was for the agent to learn the best card to buy in every round, in order to win the game. So with 18 different choices in every round, the total number of state-action pairs was 7,188,480 for four player games, and 38,707,200 for two player games.

4.1. Forced Exploration

The action selection method used during both sets of training was ε-greedy, with $\varepsilon = 0.2$, the learning rate $e = 0.2$ and the discount factor $\gamma = 0.95$, which are the recommended values by R. Sutton [3]. Training games with different values for these parameters were conducted, but the results were worst, so we fixed their values as above. Although the total number of different state-action pairs was quite large in both cases, we found that the agent only explored only a small portion of them. In the case of four player games, after 10,000,000 training games, the agent explored 27.2% of the state-action pairs (1,958,822 of 7,188,480) and only in 11.1% (800,087 of 7,188,480) he chose the action, given the state, more than 10 times. In the case of two player games, after 15,000,000 training games, the corresponding numbers are 24.1% (9,358,686 of 38,707,200) and 8.4% (3,265,873 of 38,707,200).

In order to force the agent to explore more, we created an expansion to the ε-greedy method, that we called Forced ε-greedy. To apply this method, we kept track of how many times during the training the agent visited every state-action pair, and we named every visit a "pass". Before the agent select the actions, he checks the passes of every possible action he can take. If there is an action with less than 10 passes, he selects that action. If there are more than one actions with 10 or less passes, he selects the one with the least passes, and if two or more have the least passes he selects one at random. Finally if all the actions have 10 passes or more he selects his next action with the ε-greedy action selection method. The training for both two and four player games was repeated, using the forced ε-greedy method. We slightly modified the Forced ε-greedy method to prefer actions with less than 5 passes instead of 10 in the case of four player games, due to the smaller total number of states. As we can see in Table 3, we forced the agent to explore significantly more in both cases.

Table 3. State-actions pairs explored for classic and forced ε-greedy

	Two players games		**Four players games**	
	Classic ε-greedy	**Forced ε-greedy**	**Classic ε-greedy**	**Forced ε-greedy**
Total Number of states	7,188,480	7,188,480	38,707,200	38,707,200
Number of states explored	1,958,822	2,830,574	9,358,686	15,009,555
Number of states explored (%)	27.2	39.3	24.1	38.7
Number of states with more than 10 passes	800,087	1,213,700	3,265,873	5,702,073
Number of states with more than 10 passes (%)	11.1	16.8	8.4	14.7

In order to verify our results the training for both methods was repeated, for both two and four player games, ten times. In Figure 1 we can see the average winning rate of the agent (5 point moving average) for the four player games, for both classic and forced ε-greedy. The forced ε-greedy method has slightly better results, as the agent reaches the average winning rate, earlier than the agent trained with the classic ε-greedy method.

Figure 1. Average winning of Classic and Forced ε-greedy (4 player games)

In Figure 2 we can see the average winning rate of the agent (5 point moving average) for the two player games this time, for both classic and forced ε-greedy. The agent trained with the forced ε-greedy method, not only is faster, but his average winning rate is better, which suggest that he found a better strategy, than the agent trained with the classic ε-greedy method. The large number of total states, clearly favors the forced ε-greedy method.

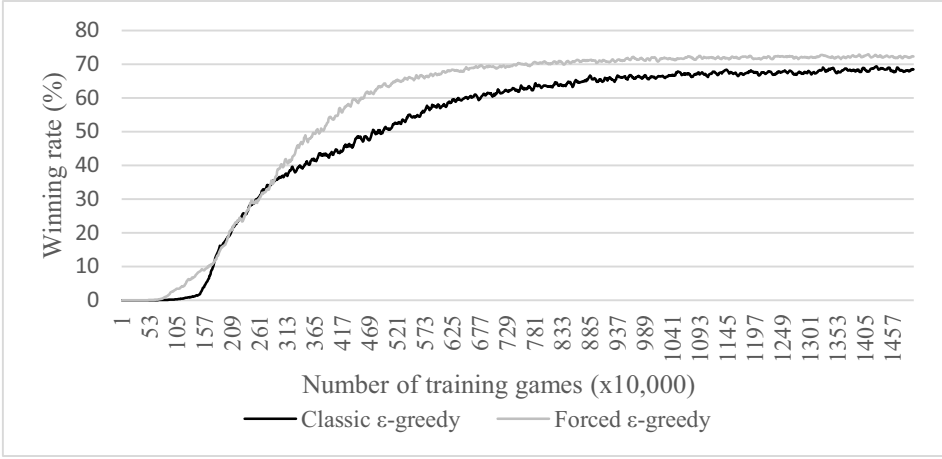

Figure 2. Average winning of Classic and Forced ε-greedy (2 player games)

In the case of the two player games, the success of the trained agent is slightly better than the one trained by Winder [16], who also tested his agent against the money policy, and found that the agent wining rate was 74.7%. The agent trained with the Forced ε-greedy method had average wining rate of 76.9% (the best result found during training).

5. Discussion and Future Work

In Q Learning, it is crucial to give the agent the option to explore new states, in order to find the optimal solution for the task at hand. Given the update algorithm of the Q values for every state-action pair, it is clear that with every update we get a better approximation of the true Q value. With the classic ε-greedy method, the agent only by chance selects low Q value actions. If the number of the total state-action pair is small, ant the number of training games large, we can be quite sure, that most of the possible actions, will get selected, and the agent will find the optimal solution. But in more complex environments, where the number of states is huge, the ε-greedy method can overlook some actions, causing the training process to be slower and possibly miss the optimal solution. In such environments, forcing the agent to explore actions with initial low Q values, which were assigned at random, will lead to better approximations for those values, causing the higher of them to be chosen again and the training process to present faster and better results.

The Forced exploration is not a new action selection method, but merely an expansion to the ε-greedy method. Other actions selection methods, as softmax, can also be expanded the in same way. It is also our opinion that the forced exploration could find an application in Deep Q Learning methods, involving neural networks.

References

[1] Gerald Tesauro. 1994. TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural computation 6, 5 (March 1994), 215-219. https://doi.org/10.1162/neco.1994.6.2.215

[2] Nicol Schraudolph, Peter Dayan, and Terrence Sejnowski. 1994. Temporal difference learning of position evaluation in the game of Go. In Proceedings of the Advances in Neural Information Processing Systems. 6. 817-824

[3] David Silver, Richard Sutton, and Martin Müller. 2007. Reinforcement Learning of Local Shape in the Game of Go. In Proceedings of the 20th international joint conference on Artificial intelligence. 7. 1053-1058

[4] Michiel Van Der Ree, and Marco Wiering. 2013. Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play. In Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. 108-115

[5] Nees Jan van Eck, and Michiel van Wezel. 2008. Application of reinforcement learning to the game of Othello. Computers \& Operations Research 35, 6 (June 2008), 1999-2017. https://doi.org/10.1016/j.cor.2006.10.004

[6] Stván Szita, Guillaume Chaslot, and Pieter Spronck. 2009. Monte-carlo tree search in settlers of catan. In Proceedings of the Advances in Computer Games. 21-32

[7] Michael Pfeiffer. Reinforcement learning of strategies for Settlers of Catan. 2004. in Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education

[8] Richard Sutton, and Andrew Barto. 1998. Reinforcement Learning: An Introduction. Cambridge, UK: MIT Press

[9] Tokic M. (2010) Adaptive ε-Greedy Exploration in Reinforcement Learning Based on Value Differences. In: Dillmann R., Beyerer J., Hanebeck U.D., Schultz T. (eds) KI 2010: Advances in Artificial Intelligence. KI 2010. Lecture Notes in Computer Science, vol 6359. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-16111-7_23

[10] Dos Santos Mignon, Alexandre, and Ricardo Luis de Azevedo da Rocha. "An Adaptive Implementation of ε-Greedy in Reinforcement Learning." Procedia Computer Science 109 (2017): 1146-1151.
[11] Dabney, Will, Georg Ostrovski, and André Barreto. "Temporally-extended {\epsilon}-greedy exploration." arXiv preprint arXiv: 2006.01782 (2020).
[12] Gimelfarb, Michael, Scott Sanner, and Chi-Guhn Lee. "{\epsilon}-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning." arXiv preprint arXiv: 2007.00869 (2020).
[13] Rasmus Bille Fynbo, and Christian Sinding Nellemann. 2010. Developing an agent for dominion using modern ai-approaches. PhD Thesis, M. Sc. IT, Media Technology and Games (MTG-T) Center for Computer Games Research
[14] Robin Tollisen, Jon Vegard Jansen, Morten Goodwin, and Sondre Glimsdal. 2015. AIs for Dominion Using Monte-Carlo Tree Search. In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. 43-52
[15] Sondre Glimsdal. 2015. AIs for Dominion Using Monte-Carlo Tree Search. In Proceedings of the Current Approaches in Applied Artificial Intelligence: 28th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. 43
[16] Ransom Winder. 2014. Methods for approximating value functions for the Dominion card game. Evolutionary Intelligence 6, 4 (March 2014). 195–204. https://doi.org/10.1007/s12065-013-0096-9
[17] Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. 2012. Evolving card sets towards balancing dominion. In Proceedings of the 2012 IEEE Congress on Evolutionary Computation. 1-8
[18] George Angelopoulos and Dimitris Metafas. 2020. Q Learning applied on the Board Game Dominion. In Proceedings of the 2020 24th Pan-Hellenic Conference on Informatics (PCI 2020). 34–37. https://doi.org/10.1145/3437120.3437269