Evolution of Reinforcement Learning in Games or How to Win against Humans with Intelligent Agents

Thomas Pignede
Fachbereich 20 - Informatik
TU Darmstadt
thomas.pignede@stud.tu-darmstadt.de

Abstract

This paper reviews reinforcement learning in games. Games are a popular domain for the design and evaluation of AI systems, and still offer formidable challenges. This paper presents some classic and more recent work from this field. First it starts with Tesauro's TD-Gammon, which was one of the first successes where a game-playing agent learned only from its own experience. Then it talks about Hu and Wellman's approach for modelling multi-agent environments and calculating their Nash-equilibria. Finally it explains Johanson, Zinkevich and Bowling's method on how to deal with stochasticity and computing robust strategies. The conclusion connects the different algorithms together and gives an outlook on current research topics and on the practical application of the presented techniques.

1 Introduction

Since artificial intelligence and machine learning have become more and more popular, the dream of being able to reproduce the capabilities of humans is omnipresent. To demonstrate the efficiency of the learning algorithm, games have often played a big role. Especially in reinforcement learning there have been lots of challenges, where the goal was to create a learning system that can play a game at least at a person's level - or even better to develop one that can beat every possible human being.

The allure of games as test beds is the possibility to explore domains of huge complexity, while still having a well-defined environment and rather simple rules that enable to simulate the game easily. Thanks to their unambiguous outcomes, games provide straightforward performance measures for evaluating the agent. Last but not least they are usually easy to understand and allow to show the key issues of the learning algorithm, without also having to focus on many external factors and influences like non-linearities in robotics or unwanted side-effects in real-world scenarios.

So games have useful properties that allow to design and test learning systems from scratch, without the need of incorporating many other questions that are not very important yet. Of course, after this development and evaluation process, the algorithms can be adapted and used in many other contexts, but it is nice to have such an environment where one is able to explore the strengths and the weaknesses of the agent already at the beginning.

Now the big question is to know how and where to start diving into this huge topic. By giving an overview about the evolution of reinforcement learning in games and presenting three different approaches, this paper first illustrates where it all came from. Then it shows what are the kind of problems the current state of the art has to deal with. At the end it hopefully will have given a good overview about this field and transmitted some ideas about open issues when applying the presented techniques to concrete examples from the real-world.

Because temporal difference methods are one of the oldest in reinforcement learning, the first paper also serves as an example to demonstrate the very basics of this class of algorithms. In comparison to those from the second and third paper, the TD-algorithm is rather easy to understand and helps focus on the essentials of the learning process.

2 Regression from experience: TD-Gammon

2.1 Background

In 1995 Gerald Tesauro published his work on TD-Gammon [1], a game-learning program that achieved master level play in backgammon only by training itself while playing against its own. This probably has been one of the most important milestones in reinforcement learning, because it was one of the first considerable successes in solving large-scale problems with a high complexity. TD-Gammon exerted strong influence over subsequent research in artificial intelligence and had a big impact for the growing interest in further research regarding learning agents.

2.2 Learning-Algorithm

TD-Gammon uses a neural network to predict outcomes of the game from a current state. The states are represented by an input vector X that contains all the board positions of the checkers and the predicted estimate is represented by an output vector Y that stands for the four possible results “White wins”, “White wins with a gammon”, “Black wins”, “Black wins with a gammon”. At every round TD-Gammon will choose the move with the best estimated outcome out of those allowed by the stochastic dice roll.

The learning process consists of improving the approximated expectation calculated by the neural network. For this a temporal difference learning algorithm called TD(λ) is used to update the weights of the neural function in order to be more close to the exact prediction. At each time step, the difference between the next approximation Y_{t+1} and the current approximation Y_t (i.e. the “TD-error”) is used to adapt the weights w towards a value consistent with the observations

The mathematical formula goes as follows:

$$w_{t+1} = w_t + \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

where α is the learning rate, $\nabla_w Y_k$ is the gradient of the network output at time k (i.e. the weight changes) and the parameter λ controls the feedback of the TD-error for correcting previous estimates (the so-called “temporal credit assignment”). A value between the extreme cases 0 and 1 has to be chosen to provide a correction smoothly decaying the farther the time goes back. If $\lambda = 0$ the error does not feed back in time, meaning that only the current time steps plays a role for updating the weights. With $\lambda = 1$ the feedback occurs without any discounting, so even errors from far back in time get accounted to correct previous estimates.

2.3 Successes

One of the most amazing results of the training was that first, even without any initial knowledge TD-Gammon had already learned basic strategies after a few thousand games and second, with a growing number of training games it still continued discovering better and better strategies in a well-scaled behaviour. By adding some extra features it finally reached world-class level and furthermore even found new previously unexplored strategies or revised traditional ones.

The main reason basically is due to the numerical precision of the estimates. Experiments analysing the data discovered that the choices between two candidate positions are estimated very similarly, because they’re coming from very small changes with respect to the absolute context. But it turned out that the relative difference between those almost similar-looking states still resulted to a clear ranking between the optimal actions, while a human would not be able to judge the best strategy in such a situation.

Further, thank to the stochasticity of backgammon, the random dice roll will make the learning agent explore many more states than it would do by taking only deterministic actions. This leads to the discovery of new and untested strategies, which improves the evaluation of the possible actions way farther. Another aspect is that even for random strategies backgammon never falls into infinite loops and always terminates in a clear state, meaning TD(λ) receives a final reward with a determined signal (win or loss).

2.4 Limits

It seems that temporal difference learning works well when learning game strategies for large-scaled complex problems. However, the selection of the best legal move is not always as simple and straight-forward as in this game. There are many cases where the optimal action depends much more on the opponents, especially in multi-agent environments where each agent acts on its own with potentially completely different goals and actions. Therefore it is not possible to consider individual actions anymore, but instead an adaptation to each other is needed. An attempt to handle such scenarios is presented in the next section.

3 Modelling Opponents: Nash Q-Learning

3.1 Background

One of the biggest problem when learning in a multi-agent context is the loss of a stationary environment because all agents are adapting simultaneously. The consequence is that the best action of one agent is also dependent of the other agents' behaviour. In 2003 Junling Hu and Michael P. Wellman published a paper [2] where they adapted classical single-agent Q-learning to work with multi-agent systems, where each agent's reward in the current state now depends on the joint actions of all agents in that state. This results to a Nash-equilibrium where every agent chooses its best strategy accordingly to the expected behaviour of the other agents. To achieve this Nash-equilibrium, in the presented learning algorithm all agents have to iteratively update their so-called "Nash Q-values" relatively to the estimated best strategy of all other agents, such that the optimal actions work as a best response to the other agents' derived model.

3.2 The Nash Q-Learning Algorithm

In order to understand the algorithm, it is useful to start with the standard single-agent Q-learning. The goal is to learn the Q-function $Q^*(s,a)$ from which we can derive the optimal policy

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$$

where s is the state and a is the action that maximizes the Q-function in that state. The Q-function is defined as

$$Q^*(s,a) = r(s,a) + \beta \sum_{s'} p(s'|s,a) v(s', \pi^*)$$

where $r(s,a)$ is the reward for taking the action a in the state s, $\beta \in [0,1)$ is the discount factor, $p(s'|s,a)$ is the probability of resulting in the state s' after taking the action a in the state s and $v(s',\pi^*)$ is the value for taking the optimal policy after being in the state s', so it can be rewriten as $\max_a Q^*(s',a)$.

The iterative Q-learning algorithm starts with initial values of $Q(s,a)$ for every state s and for every action a and then updates the Q-function with the following directive where $\alpha \in [0,1)$ is the learning rate:

$$Q_{t+1}(s,a) = (1-\alpha_t) Q_t(s,a) + \alpha_t \left(r_t + \beta \max_a Q_t(s',a) \right)$$

Now it is possible to extend this algorithm for multi-agent environments to reach the optimal joint strategy where each agent acts as a best response relative to the other agent's behaviour (i.e. the Nash-equilibirum). That means that the goal is to calculate a tuple of strategies $(\pi_1^*, ..., \pi_n^*)$, so that for every agent the value function $v(s; \pi_1^*, ..., \pi_n^*)$ for each state s has its maximum with this tuple of strategies, so any other strategy of an agent i could only be equal to or worse than its strategy π_i^*.

The Nash Q-function of an agent i is quite similar to the single-agent case except that now it depends on the joint action $(a^1, ..., a^n)$. So the idea, that the Q-value in a specific state s is equal to the current reward for the joint action plus the expected future rewards under the assumption of all agents following the optimal joint strategy, remains the same:

$$Q_i^*(s; a^1, ..., a^n) = r^i(s; a^1, ..., a^n) + \beta \sum_{s'} p(s'|s; a^1, ..., a^n) v^i(s'; \pi_1^*, ..., \pi_n^*)$$

Also the Nash Q-learning algorithm works rather analoguously to the classical Q-learning. The important difference when updating the Q-value of the current state s is the question on how to use the Q-values of the next state s'. Instead of using the agent's own maximum payoff $\max_a Q_t(s', a)$, the multi-agent algorithm uses the future Nash-equilibrium payoff $NashQ_t(s')$, for which it is important to consider the rewards of all agents, because the Q-functions of all agents are needed for calculating this Nash-equilibrium (those Q-values have to be learned by the agent too).

Therefore the iterative directive for the i-th Q-function again starts with an initial value of $Q^i(s; a^1, ..., a^n)$ for every state s and every joint action $(a^1, ...a^n)$ and then updates the Q-values like this:

$$Q_{t+1}^i(s; a^1, ..., a^n) = (1 - \alpha_t) Q_t^i(s; a^1, ..., a^n) + \alpha_t \left(r_t^i + \beta NashQ_t^i(s') \right)$$

where the Nash-equilibrium $NashQ_t^i(s')$ is the payoff of the i-th agent for using the current (i.e. at time t) optimal joint strategy $(\pi_1^*, ..., \pi_n^*)$ in the state s'. This equilibirum is calculated with:

$$NashQ_t^i(s') = \pi^1(s')...\pi^n(s') Q_t^i(s')$$

This finally leads to the following algorithm that a learning agent i has to execute:

- for all states s, for all joint actions $(a^1, ..., a^n)$ and for all learning agents j initialize the Q-function with $Q_0^j(s; a^1, ..., a^n) = 0$
- on each time step, choose your own action a^i and observe the joint action $(a^1, ..., a^n)$, all rewards $r_t^1, ..., r_t^n$ and the resulting state s'
- update the Q-values for all agents j with
  $Q_{t+1}^j(s; a^1, ..., a^n) = (1 - \alpha_t) Q_t^j(s; a^1, ..., a^n) + \alpha_t \left(r_t^j + \beta NashQ_t^j(s') \right)$
  where $NashQ_t^j(s') = \pi^1(s')...\pi^n(s') Q_t^j(s')$

3.3 Experimental Runs

The Nash Q-learning algorithm has been successfully applied in simple grid-world games with two agents trying to reach their goal, where they'll earn a positive reward. The first important result to note is that in almost all of the experiments the algorithm has converged towards a Nash-equilibrium that corresponded to the theoretically derived optimal Q-function Q^*, so the learning agents were likely to get a strategy very close to the best strategy π^*. It took about 5000 training episodes until the values of the Q-functions stabilized. Another interesting aspect is that even in an environment where all the other agents act randomly, a learning agent using Nash Q-learning performs better than an agent that uses single-agent Q-learning. The last surprising thing is the fact, that when at least one of the learning agent operates with this multi-agent algorithm, this situation already yielded to a better functioning of all learning agents, even if all the others were still just using classical Q-learning.

Unfortunately the authors did not test this method in more complex games, especially with more than two players. The reason why this would have been interesting is the exponential complexity of this algorithm in the number of agents. In single-agent Q-learning the learner has to retain one Q-function with a total number of entries of $(|states| \cdot |actions|)$. However, in the multi-agent scenario with n actors, each learner has to maintain n Q-functions (one for each agent), whereas every Q-function needs to memorize $\left(|states| \cdot \left|actions^1\right| \cdot ... \cdot |actions^n|\right) \in O\left(|states| \cdot |actions|^n\right)$ entries. So a growing number of actors could prevent the algorithm to stay practicable.

While the performance of this algorithm was perfect in grid-world games where all moves were deterministic, in games with stochastic transitions it did not always attain a convergence to a Nash-equilibrium. Such problems with stochastic environments are kind of common when designing and evaluating learning systems. Nevertheless in many games the stochasticity plays a big role, so developing robust strategies is essential for intelligent agents that have to perform well in a stochastic context. This is what the following part tries to deal with.

4 Dealing with Stochasticity: Robust Planning

4.1 Background

One of the most common techniques for learning agents to make a decision adapted to the other agents' behaviour in multi-agent scenarios is called "best-response strategy". The counter-strategy chosen by the agent tries to maximize its performance with respect to the choices of the other agents. In order to adapt to the multi-agent system, the learning agent ideally knows how the other agents are acting, but at least has to be able to learn and to make assumptions about the expected behavioral model of the counterparts. Unfortunately the calculated strategies often are very poor when the presumed model about the scenario is wrong.

This problem is addressed by Michael Johanson, Martin Zinkevich and Michael Bowling from the University of Alberta, especially famous for its computer poker research group. They published a paper in 2007 that introduces a new approach for calculating robust counter-strategies [3]. By computing their so-called "restricted Nash responses", they are able to provide counter-strategies that give a good balance between performance maximization and reasonable results if the model is wrong. As demonstrated in their case study about Texas Hold'Em, it seems that while still being very effective, those restricted responses are much more robust than a normal best-response strategy.

4.2 Frequentist Best Response

Before coming to the actual algorithm of interest, the authors begin with an examination of an approximate best-response counter-strategy against several poker opponents. The frequentist algorithm tries to learn a model of its opponent by observing it playing many poker games and then computes an appropriate best-response as its counter-strategy. In the paper, first they trained several agents using this method against different opponents by running about 5 million training matches. Then for the evaluation they played the resulting responses against all given strategies and analysed the performance of the agent at exploiting the adversary and not being exploited itself.

The most important fact is that while the frequentist best response works quite well against the strategy from which it learned an opponent's model, it is really bad for exploiting opponents using another strategy. This analysis shows that best responses seem not to be very robust, because they will mostly fail even if the scenario is just using a slightly different model than presumed by the learning agent.

4.3 Restricted Nash Response

The main part of the paper discusses what the authors call "restricted Nash responses": A model that requires the algorithm to be robust with respect to uncertainty in an opponent model.

The basic idea of this approach is the hypothesis that the model of the opponent is not quite strict but allows some freedom, for which the learning agent still has to be robust. This is formalized by considering the opponent's strategy to be a pair of a fixed strategy $\sigma_{fix} \in \Sigma_2$ and an arbitrarilly chosen strategy $\sigma_2' \in \Sigma_2$, where Σ_2 is the set of all possible strategies for the opponent. Further the opponent is supposed to choose the fixed strategy σ_{fix} with a probability of p and to take the unknown strategy σ_2' with probability $(1-p)$, so the set $\Sigma_2^{p,\sigma_{fix}}$ is the set of all possible mixed strategies, where the opponent plays with the strategy σ_{fix} with a probability of p and with the other strategy otherwise.

Now the goal for the learning agent is to have a counter-strategy that is able to exploit the opponent while still being robust for all strategies σ_2'. To formalize this concept, the set of "restricted best responses" $BR(\sigma_2)$ to an opponent's strategy $\sigma_2 \in \Sigma_2^{p,\sigma_{fix}}$ represents the counter-strategies $\sigma_1 \in \Sigma_1$ that are a best response for the learning agent and is defined with

$$BR(\sigma_2) = \operatorname{argmax}_{\sigma_1 \in \Sigma_1} u_1(\sigma_1, \sigma_2)$$

where $u_1(\sigma_1, \sigma_2)$ is the utility of player 1 when using σ_1 while player 2 is using σ_2. So the learning agent has to find the strategy σ_1 that yields to the highest value when the opponent plays with strategy σ_2.

If analogous the set of "restricted best responses" $BR(\sigma_1)$ to the learning agent's strategy is defined with

$$BR(\sigma_1) = \text{argmax}_{\sigma_2 \in \Sigma_2^{p,\sigma_{fix}}} u_2(\sigma_1, \sigma_2)$$

then all pair of strategies (σ_1^*, σ_2^*) are a "restricted Nash equilibrium" when $\sigma_1^* \in BR(\sigma_2^*)$ and $\sigma_2^* \in BR(\sigma_1^*)$ holds. The strategy σ_1^* is the wanted "restricted Nash response" as counter-strategy to σ_{fix}.

4.4 Results

For the evaluation of these "restricted Nash responses" the authors used the same setup as for the "frequentist best responses" shown at the beginning. So they again used about 5 million test matches for training the several learning agents against the different opponents, but this time the agent computed a restricted response to the opponent's model with the algorithm presented before.

The paper claims that those counter-strategies are ideal when playing against the mixed strategy $\{\sigma_{fix}, \sigma_2'\}$, because the probability p provides a good balance between exploiting the opponent and not being exploited itself (i.e. robustness). For example, when p is near to 1, the agent acts more or less with a normal best-response because it assumes that the opponent always plays with the strategy σ_{fix}. So this learning agent is in fact very good in exploiting an opponent that exactly uses that strategy, but just like in the previous evaluation it is very poor when the opponent's strategy varies a bit. In comparison to that, a learning agent with a lower value for the probability p gets strategies that are much closer to the "restricted Nash equilibrium" and much more robust against different opponents, while still performing well against the model it relies on. Experiments have shown that already a value of $p \approx 0.9$ is able to reduce the exploitability of the agent considerably without having an important loss in the exploitation of the opponent's model. Therefore it is suggested to use such "restricted Nash responses" because the learner becomes much more robust against errors in the presumed model.

Nevertheless the question on how to actually generate candidate responses still remains a challenging task. To solve this problem the authors formalized Texas Hold'Em as an abstract game, that enabled the computation of such restricted responses and to check whether they form a Nash equilibrium with the adversarial strategy. The concrete abstraction and calculation is rather only sketched out, but it becomes obvious that modelling games - in a way that one can use those techniques - seems not to be a trivial task at all. So a lot of work still has to be done when applying this algorithm to other games.

5 Conclusions & Outlook

In this paper the evolution of reinforcement learning has been illustrated. The TD-Gammon player elucidates some basic princples of machine learning in adversarial settings. Recent work has addressed complications such as the non-stationarity of multi-agent scenarios and the robustness in stochastic environments, both being essential issues one has to deal with for succeeding in beating a human.

When designing and testing intelligent agents these are the key elements of the development process. The system usually learns iteratively through its growing experience of choosing best moves while playing the game again and again, but since many games have to deal with independent opponents and a non-stationary, stochastic context, adaptation to the adversary and robustness in such an environment is also a crucial part to model.

The approaches presented here are intended to give an idea of possible attempts for solving those challenges. They hopefully will help having a good starting point in this huge and complex domain and provide an outline for a specific exploration of further topics of interest. However it is true that the focus laid more on explaining the general problems and the foundations of the methods trying to handle those issues, than on demonstrating concrete implementations of the algorithms. The goal was not to end up with a sort of cookbook for generating autonomous game-playing agents from a reinforcement learning framework (and without really knowing what it's all about), but rather to give an elementary comprehension over the material.

Nevertheless there are many descriptions of more practical techniques available. For example one work on how to concretely compute Nash-equilibria in games can be found in Martin Zinkevich's, Michael Bowling's and Neil Burch's paper "A New Algorithm for Generating Equilibria in Massive Zero-Sum Games" from 2007 [4]. One of the results was the observation, that while an equilibrium strategy obtained against a strong opponent A is still very safe against a weaker opponent B, it does not become considerably more exploitive against this simpler bot. So it seems that the generated responses are not that adaptive with respect to the adversarial strength. This should be a clear disadvantage to human players because they're able to adapt to the different skill levels of their counterplayers. Therefore the development of a better balance between exploitation and safety should be considered in future approaches.

Another issue in the presented works is the lack of tests in real-world scenarios. The papers did not really spot the practicability of the methods in really complex, non-stationary environments with a huge number of other autonomous systems. It would be interesting to know whether the algorithms could be applied in other domains where a fast decision is also essential during the learning process. For example, in high-frequency trading there is the need of automatically buying or selling stocks respectively to the actions done by other market participants. As already mentioned, the complexity for calculating such strategies is highly correlated to the number of independent learning agents. So trying to find a compromise between computing approximate Nash-equilibria (if any) and being faster than other automated opponents in this setup would be a possible research topic.

Furthermore the learning agents needed a high number of training matches before reaching a reasonable level. In comparison to that, humans normally do not need thousands Backgammon matches until they have understood basic strategies, e.g. to protect their checkers for not being hit. Even if the algorithms performed really well after having been trained enough, there probably exists some situations where one would just not have the time to wait that long, because of fast-changing states, actions and rules of the games (here again the high-frequency trading market could serve as example). It could be exciting to find out, how far human capabilities for understanding basic concepts after very few trials can be extracted and applied to learning algorithms, without loosing too much of the benefits of precise machines.

Acknowledgments

I thank Svenja Stark and Philipp Hennig for helpful discussions, critical comments and providing many suggestions.

References

[1] Gerald Tesauro: Temporal Difference Learning and TD-Gammon. *Communications of the ACM,* March 1995 / Vol. 38, No. 3, 58-68

[2] Junling Hu & Michael P. Wellman: Nash Q-Learning for General-Sum Stochastic Games. *Journal of Machine Learning Research,* November 2003 / Vol. 4, 1039-1069.

[3] Michael Johanson & Martin Zinkevich & Michael Bowling: Computing Robust Counter-Strategies. *Advances in Neural Information Processing Systems,* NIPS 2007

[4] Martin Zinkevich & Michael Bowling & Neil Burch: A New Algorithm for Generating Equilibria in Massive Zero-Sum Games. *Proceedings of the Twenty-Second Conference on Artificial Intelligence,* AAAI 2007