Learning to Play Computer Games with Deep Learning and Reinforcement Learning

Project Plan

Student	Supervisor
Mak Jeffrey Kelvin	Dr. Dirk Schneiders

October 1, 2017

Contents

1 Introduction

The history of reinforcement learning (RL) is lengthy, and originated from behavioural psychology in the 1950's, where it was demonstrated that animals could be trained to perform certain tasks by providing them with positive rewards after task completion [1]. With such a concept in mind, the field began as a sub-discipline of artificial intelligence (AI) with the intention of synthesizing responsive AI systems in the early 2000's. However, many attempts at the time were only limited to simple low-dimensional problems, since the algorithms failed to handle high-dimensional problems [2]. Nevertheless, with the advent of technology, the computational power of computers have increased over the recent years. This indirectly contributed to the realization of deep learning in the form of neural networks, which allowed large high-dimensional datasets to be represented with small quantities of low-dimensional parameters [3].

One of the first ideas in reinforcement learning was to use it for defeating a wide variety of games. It first began by tackling simple conventional board games such as TD-Gammon in 1995 cite(td-gammon)). Next, the field progressed to playing Atari 2600 games in 2013 [4, 5, 6] and 3D world games, such as Doom [7]. Recently, the reinforcement learning community has also seen the defeat of the Chinese game Go by AlphaGo in 2015 [8], whose complexity is demonstrated by the enormous number of game states. Since reinforcement learning algorithms contain very little game-specific information, they can be easily used to play any game, albeit with performance differences in some cases. As a result, it is not uncommon to easily find open-source RL implementations of popular modern games from the community, eg. the first level of Super Mario World [9] and Flappy Bird [10].

One may question the significance of playing games using AI in general, as there are many existing computer game bots implementations already. However, the idea of playing games using reinforcement learning connotes a more important application, namely to develop human-like intelligence that is applicable to the physical world. In order to achieve this, reinforcement learning methods only accept visual inputs in the form of a series of images. In other words, no internal state variable is directly observed by the AI agent. One of the benefits of such an approach is that the AI agent receives the same information that a human would perceive when playing the game, thus encouraging RL methods to achieve human-like behaviours. In addition, games provide a medium to explore algorithms in a safe and cheap manner, in contrast to in the real world where training would become expensive. Lastly, most games were designed as a simplified simulation of the real world, thereby allowing direct application of RL methods to tasks of similar nature in the physical world, such as obstacle navigation.

In terms of computational power, deep learning in the form of neural networks enabled classifiers to become much more versatile in recent years. Since machine learning often deals with high dimensionality data with complex features, it is important to reduce the dimensionality in order to extract, process and visualize such nontrivial features. In particular, according to Hinton and Salakhutdinov (2006), neural networks in the form of deep auto-encoder networks outperform traditional principal component analysis in the ability to compress data [11]. Convolutional neural networks are robust in image pattern recognition, and recurrent neural networks, especially LSTMs [12], are useful for providing persistent memory to neural networks [13].

2 Background

2.1 Reinforcement Learning

In computer science, reinforcement learning can be broadly classified into two types, namely model-based and model-free RL. As the name suggests, model-based RL relies on learning in the presence of a pre-existing model of the agent's controller, whereas model-free RL involves direct trial-and-error by the agent controller in order to achieve learning. Since the system dynamics in games are often convoluted and difficult to model, this project will adopt mostly the model-free approach.

In general, the problem statement solved by reinforcement learning can often be defined as follows: to learn an optimal policy π^* based on the agent's interaction with the environment by maximizing the expected sum of discounted rewards $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where a policy π is a mapping of states to actions. In addition, the environment is usually modelled as a Markov Decision Process (MDP), which is described by a set of possible states S, a set of possible actions A, a transition matrix $T(s'|s,a)$, a reward function R, and a discount variable $\gamma \in [0,1]$. In particular, $T(s'|s,a)$ denotes transition probabilities between states via actions, ie. is a function with mapping $S \times A \to A$, and R is a function with mapping $S \times A \to R$.

Contrary to conventional techniques, agents in games often do not have direct access to the current state. Instead, they receive the current state's observation, which embeds partial information of the current state. Since the series of observations do not form a MDP, a slightly modified process, namely Partially Observable Markov Decision Process (POMDP), is used to describe the agent's interaction with the environment instead [14]. For example, since all state information can be inferred from the last four frames of the game for any Atari game [5], a MDP can be used to model the interaction [4]. On the contrary, the in-game player in the first-player shooting (FPS) game Doom only has a limited view of the map, and so playing this would require POMDP as an approximation to the interaction between the player and the map [7].

2.2 Q-Learning

A popular reinforcement learning technique for constructing an optimal policy is Q-learning. Q-learning is an online model-free method. It is also an off-policy learner, meaning that it can learn the optimal policy regardless of the agent's actions. In terms of the algorithm, it uses a value-iterative approach that is based upon the Q function and Bellman equation. In particular, the Q function of a policy π maps state-action tuples (s,a) to a real number, and is used as an indication of the expected reward by taking action a from state s:

$$Q^\pi(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a]$$

The higher the Q value, the more likely better results will be achieved in the long term. Moreover, a policy pi is said to be an optimal policy π^* if the Bellman equation is satisfied:

$$Q^*(s_t,a_t) = \mathbb{E}[r_t + \gamma \max_{a'} Q^*(s_{t+1},a')]$$

Based on the above equations, a value iterative approach can be used to construct the following update equation in order to attain Q*:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

However, in practice, using only the above update equation would not allow the agent to learn, as the update equation only exploits information that the agent knows. Furthermore, the update equation does not provide an opportunity for the agent to explore other potential states that would lead to an optimal policy. To mitigate this issue, an ε-greedy algorithm is used in practice, where the agent performs a random action with probability ε, and takes the action resulting in maximum Q with probability $1-\varepsilon$. Additionally, the parameter ε is usually set to 1 initially and decayed gradually to zero as more states are explored. This is because the agent has little information on which action to take in the initial iterations, thereby leading to the need to explore the environment by taking random actions. Conversely, taking random actions in late iterations would prevent the agent from achieving the optimal policy, as the average Q value of the next state after taking a random action is less than that of taking the optimal action as suggested by the policy [1].

2.3 Deep Q-Networks

In the context of games, it is often the case that the size of the state-action space is too large for a tabular Q function to be constructed, which renders traditional Q-learning impractical in such situations. To mitigate the issue, Deep Q-Networks (DQN), also known as deep Q-learning (DQL), approximates the Q function by using a neural network with parameters θ instead. For the neural network, θ is trained so that at time t the loss

$$E_{s_t,a_t,r_t,s_{t+1}} ||Q_\theta(s_t,a_t) - y_t||$$

is minimized, where

$$y = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_{a'} Q^*_{\theta_{target}}(s_{t+1},a') & \text{otherwise} \end{cases}$$

[15]. The advantage to such an approach is that it greatly reduces the number of training parameters, and also allows the Q function to generalize training data to unknown scenarios [14].

Apart from functional approximation of the Q function by a neural network, Deep Q-Networks also utilize experience replay in order to facilitate training of the network. Specifically, the network is trained with previous memory tuples $(s_t, a_t, s_{t+1}, r_{t+1})$ in additional to the new state-action tuple obtained after performing an action in each step. Including experience replay in Deep Q-Networks confers more advantages as compared to vanilla Q-learning, with the main reason being that experience replay allows for higher data efficiency as well as reduced fluctuations in parameter values during training [1]. In practice, only the last N memory tuples are stored in the algorithm to reduce memory costs [4]. Additionally, only a subset of the stored tuples are used in backpropagation by stochastic gradient descent, thus ensuring that the computation is scalable to large datasets.

2.4 Deep Recurrent Q-Networks

Some games violate the Markov property of POMDP, ie. the optimal action taken relies on information from multiple observations. For example, in the context of Atari 2600 games, if the game requires information from a previous state more than three frames away from the current frame, then the environment's behaviour appears to be non-Markovian if only the most recent four frames are considered [16]. In order to better approximate the agent-environment interaction, Deep Recurrent Q-Networks (DRQN), also known as Deep Recurrent Q-Learning (DRQL), can be used, which add a Long Short Term Memory (LSTM) [12] on top of DQN's neural network architecture in order to provide the agent with memory of previous states. With this modified architecture, the Q function being estimated would be $Q(h_t, a_t)$, where h_t is the input obtained from the LSTM in the last step [16].

3 Related Works

Several games were played previously using a combination of reinforcement learning and deep learning techniques. The first game in which reinforcement learning was able to play was Backgammon . The algorithm that played the game, named TD-Gammon [2], involved a fully-connected multilayer perceptron architecture for its neural network and used a temporal difference algorithm similar to Q-learning to update the network edges. The results obtained from training TD-Gammon was quite substantial at the time, where it was able to learn basic techniques within the first few thousand training games, and could even learn to use sophisticated techniques after tens of thousands of training games. With success of TD-gammon, similar techniques were applied to other board games such as chess and checkers in an attempt for advancement in the field. Unfortunately, the results were not ideal, and as a result the next breakthrough was only after the development of Q-learning and its combination with a nonlinear function approximator [4].

In 2013, Google DeepMind developed the concept of Deep Q-Networks [4], and experimented with several Atari 2600 games, namely Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest and Space Invaders. The experiment results showed that the DQN superseded human performance for Breakout, Enduro and Pong, and that it also exceeded the performance of other previous algorithms. To reduce the computational cost, the raw RGB image frames were first converted to grayscale, followed by downscaling from 210×160 pixels to 110×84 image. The DQN also utilizes the frame skip technique, where the program only receives a frame for every four frames. As mentioned previously, the neural network accepts the last four pre-processed frames as input in order to mitigate the partial observability of the problem, thus leading to the use of MDP as opposed to POMDP [4].

Shortly after, Deep Recurrent Q Networks [16] was developed in 2015, which allowed agents to maintain a LSTM memory, thus providing an opportunity to explore games such as Doom [7], a FPS game. In particular, the trained agent in Doom only focused on the deathmatch mode. In terms of algorithm design, a DRQN was used for combat with other players, and a DQN was used for map navigation. The idea of a divide-and-conquer approach by separating item collection and enemy targeting network training and combat network training was due to the deathmatch's structure. Specifically, the deathmatch was split into two phases, namely the navigation and action phases. It was also noted that joint training of the DRQN's partially observed state as well as the presence of enemies and locations on the current frame on the same neural network greatly improved training performance [7]. In terms of image processing, a 16/9 color image resolution with size 455×225 was used to maximize the field of view in the game. Results showed that the agent had a high kill to death ratio, and outperformed human players.

Nevertheless, the next breakthrough in reinforcement learning was the creation of AlphaGo by Google DeepMind in early 2016 [8]. Unlike most of the previously described work, AlphaGo was a MCTS-based algorithm that relied on a policy network to evaluate board positions and a value network to determine move selections. Since Go is a board game, it does not use any image preprocessing to feed into a network. Similar to Q-Learning, MCTS converges to optimal play. Results showed that AlphaGo wins 99.9% of the time, and has even beaten a human Go expert [8].

There was a recent surge in the community to build deep reinforcement learning software with the objective of playing many other popular games that were unexplored by academia, such as Super Mario World [9] and Flappy Bird [10]. Apart from community implementations of game-playing deep reinforcement agents, there has also been efforts to open-source various games to promote development in the field of reinforcement learning. In particular, Project Malmo [17], an open-source API for AI implementations in Minecraft, was released by Microsoft in mid-2015. Furthermore, SC2LE [18], a similar API for StarCraft II, was released recently by Google DeepMind and Blizzard in mid-2017. Unlike the previous work, these games aim to solve many more difficult and unsolved problems, such as multi-task learning and how to encourage cooperation between RL agents [17]. Lastly, OpenAI has released OpenAI Gym [19] in mid-2016, thus providing an API for running the same RL implementations across different games with ease.

4 Prerequisites

4.1 Hardware Requirements

In general, the software used in this project does not require any special hardware configuration. A GPU on a Linux machine will be required for training neural networks for reinforcement learning if the training time is long. Otherwise, a Macbook with Intel GPU will be sufficient for training and testing preliminary implementations.

4.2 Software Requirements

Various reinforcement learning methods will be implemented in Python, and Keras with Tensorflow backend will be used in order to train neural networks. Moreover, this project will only deal with open-source games, as it requires direct control of in-game characters by either the game's code or an API that can control the game's in-game characters. In the case of the game Minecraft, Project Malmo [17] from Microsoft will be used. Similarly, OpenAI Gym [19] provides game APIs for many well studied games such as Pacman and Atari Games, and SC2LE [18] provides an API for Starcraft II. Lastly, University of California Berkeley CS170's course project also provides an API to control the Pacman game [20].

5 Scope and Objective

5.1 Scope

There are no strict limitations to which games this project will be dealing with. At the current state, however, the project will likely favor video games limited number of controller inputs, as this configuration resembles interaction between humans and the real world. At this stage, games that fall into this category include, but are not limited to Pacman, Atari 2600 games, Flappy Bird and Minecraft. In terms of algorithmic techniques, this project focuses on mainly using reinforcement learning and deep learning techniques. Despite of a heavy bias in existing reinforcement learning game implementations towards using Deep Q-Networks for playing games, this project will also try to explore other approaches, such as actor-critic methods, depending on the nature of the games examined.

5.2 Objective

The project's main objective is to use reinforcement learning and deep learning in order to play a game. Specifically this can be attained by either defeating the game, if the game has a clear sequence of objectives like in Pacman, or by solving one or more pre-defined tasks in the game, if the game has no clear objectives like in Minecraft. The project will be broadly divided into two stages. Specifically, the first stage involves exploration of various reinforcement algorithms and neural network architectures by using Pacman as a benchmark. The second stage would then involve picking a particular game, defining clear goals and implementing an algorithm that uses both reinforcement learning and deep learning.

6 Methodology

Since this project focuses more on algorithms than specific games, the project will be separated into two stages. However, since this project is rather open-ended, stage two may be subject to change depending on the results of stage one.

6.1 Stage 1: Algorithm Exploration

In this stage, various reinforcement learning and deep-learning methods will be explored, including but not limited to DQN, DQRN, policy-gradient methods and MCTS-based methods. Additionally, this stage will only focus on Pacman as the game of interest. As a result, effects of changing the map and ghost behaviour on the above RL methods, such as ghosts with A* search, will also be examined. The goal of this stage is to compare and contrast between various

RL algorithms in terms of various evaluation metrics, such as average number of dots eaten per death and completion rate, and to better understand which algorithms to favor.

6.2 Stage 2: RL Implementation and Optimization

In this stage, one or more game of the same type is to be chosen, and the goal is to construct and optimize one or more RL implementations that can either defeat or complete certain tasks in the game with high accuracy. At this point, any produced results will be compared against random actions to determine whether training was successful. Such results will also be contrasted with existing published results, if any. In terms of testing, a series of environments will be systematically setup to measure the accuracy and/or time taken to complete the task. Once again, the details in this stage are subject to change depending on the game chosen, and should be refined in the Interim Report.

7 Risks and Mitigation

The main risk thus far in the project is risk of being unable to successfully construct a deep reinforcement learning agent that can complete the goals mentioned in this project. In order to mitigate this, the project is split into two stages. Since Pacman is likely to have been played by others previously using similar techniques, this stage is unlikely to fail with high probability, and thus guarantees a deliverable for this project.

Another risk is the possibility of falling behind schedule during the course of this project due to unexpected obstacles. This can be avoided by setting a firm timeline on completing small deliverables so as to ensure progress is made during the course of the year.

8 Deliverables

Documents

- Project Plan (this document)
- Interim Report
- Final Report
- Project Poster

Presentations

- First Presentation
- Final Presentation

Software Deliverables

- Pacman RL implementation
- RL implementation of another game

All of the aforementioned deliverables are to be completed before the deadlines as described in the section "Project Schedule and Milestones". Additionally, all created software will be placed on a public github link, which is to be made eventually.

9 Project Schedule and Milestones

The internal and official deadlines for tasks for the entire project are detailed in the table on the next page, where official deadlines are bolded.

Date	Task
Early October 2017	Preliminary Research • Perform preliminary literature research on existing games that use reinforcement learning and deep learning techniques. • Experiment with various classic RL methods, including policy iteration, value iteration and tabular Q-Learning
October 1, 2017	**Phase 1 Deliverables (Inception): Project Scheme, Detailed Project Plan**
Mid October 2017	Stage 1: Algorithmic Exploration • Implement and evaluate the performance of DQN in Pacman • Read up on policy-gradient methods and MCTS-based methods
October 31 2017	• Implement and evaluate the performance of DRQN in Pacman • Compare results between DQN and DRQN implementations
Mid November 2017	• If applicable, Implement and evaluate the performances of policy-gradient methods and MCTS-based methods • Compare and contrast between the four implementations
January 8 – 12, 2018	**First Presentation**
January 21, 2018	**Phase 2 Deliverables (Elaboration): Pacman RL Implementations, Interim Report**
Jan –April 2018	Stage 2: RL implementation and Optimization • Pick a game to play with • Implement several RL algorithms • Optimize RL algorithms as necessary
April 15, 2018	**Phase 3 (Construction): Game RL Implementation, Final Report**
April 16 – 20, 2018	**Final Presentation**
May 2, 2018	**Project Exhibition**

10 Conclusion

Despite of the long history of using reinforcement learning to play games, the recent surge in using deep learning as a tool led to the recent development of state-of-the-art techniques such as DQN and DRQNs, which can be applied to video games. This project aims to help further explore the application of deep RL techniques on video games, as well as to contribute to the field of reinforcement learning hopefully.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction.* MIT Press, 1998.

[2] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995. [Online]. Available: http://doi.acm.org/10.1145/203330.203343

[3] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012. [Online]. Available: http://arxiv.org/abs/1206.5538

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[6] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 3338–3346. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969199

[7] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," *CoRR*, vol. abs/1609.05521, 2016. [Online]. Available: http://arxiv.org/abs/1609.05521

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, January 2016.

[9] A. Jung, "Playing mario with deep reinforcement learning," https://github.com/aleju/mario-ai, May 2016.

[10] Y.-C. Lin, "Flappy bird hack using deep reinforcement learning," https://github.com/yenchenlin/DeepLearningFlappyBird, March 2016.

[11] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed\&uid=16873662\&cmd=showdetailview\&indexed=google

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[14] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *CoRR*, vol. abs/1106.0234, 2011. [Online]. Available: http://arxiv.org/abs/1106.0234

[15] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," *CoRR*, vol. abs/1604.07255, 2016. [Online]. Available: http://arxiv.org/abs/1604.07255

[16] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, vol. abs/1507.06527, 2015. [Online]. Available: http://arxiv.org/abs/1507.06527

[17] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The malmo platform for artificial intelligence experimentation," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 4246–4247. [Online]. Available: http://www.ijcai.org/Abstract/16/643

[18] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "Starcraft II: A new challenge for reinforcement learning," *CoRR*, vol. abs/1708.04782, 2017. [Online]. Available: http://arxiv.org/abs/1708.04782

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[20] (2017, Oct). [Online]. Available: http://ai.berkeley.edu/reinforcement.html