In: *Volume title,* edited by Jan Editor and Ed Janitor. BeLLS Vol. N (2017), DOI N.

State of the Art on the usage of Deep Learning Neural Networks in Games

P0786

Abstract. The relationship between games and AI goes back a long time. This is no surprise considering the complexity of the problems proposed by most game environments, due to the fact that they are initially designed for humans to play. The complexity has pushed research in AI and more often than not, new techniques which are designed specifically solve a problem within a game, can solve a variety of problems in real world applications (Johannes Heinrich, David Silver, 2016).

Keywords: Games, Neural Networks, Deep, Supervised, Unsupervised, Reinforcement learning, Adaptive, Go, Chess, 3D, Real-Time

1 Introduction

Neural networks have now had the chance to impact games and become a standard AI technique when it comes to simulating realistic human like behaviors in autonomous agents. This is because these algorithms possess the ability to adapt and learn new strategies from game environments in order derive intelligent decisions.

However, games encompass a wide variety of styles, i.e boards games, Atari games, and 3D graphically intensive games. Each style proposes a different kind of game environment that defines a particular set of challenges, which require intelligent and complex decision making. Neural networks have long been sought as the algorithms to choose when modeling tough decision problems with elevated uncertainty such as those proposed in various games.

In this article, we will review how much neural network algorithms have evolved since being used in a game environment for the first time up until now, by examining some of the recent breakthroughs in the field which have come from the

newer set of neural network algorithms called Deep Neural Networks. These networks use a combination of supervised and reinforcement learning techniques to enable them to solve problems traditionally infeasible such as beating the world champion in an ancient and highly complex board game like Go (Silver, et al). We will examine a range of use cases within various types of games.

2 State of the Art Networks

In this section, we will explain in greater detail what deep neural networks are, and how they differ from traditional neural networks, such as the perceptron, in regards to their topologies and learning methodologies.

2.1 Learning Methods

One of the core differences between neural network approaches and the results they produce, is dictated by the way they are trained. There are three main types of training, supervised, unsupervised and reinforcement learning. Supervised training was the first training method to exist. It works by training a network to learn a set of inputs and match them to specific targets. After sufficient training, the network will seek a function so that is can map any new input with a particular target.
Unsupervised training is the opposite, as the network is only trained with a set of inputs. It will then start to learn the structure and relationship between different inputs and group them into specific clusters.

2.2 Popular Network Topologies

All neural networks can be referred to as a class of machine learning architectures. They consist of a network of individual units/neurons, organized in layers, which are connected via weights whose values are adjusted as the network is trained. One of the first neural networks was the single layer perceptron. This network would take a set of input vectors, and try to classify them using a hyperplane defined by a vector of weights (w) and a bias (b). It would do this by taking a weighted sum of these vectors using the hyperplane values and classify the input vectors based on a threshold, i.e 1 if the sum is greater than 0, otherwise 0.
However, this type of network only proved useful for binary classification of linearly separable problems. Multi-Layer feed-forward networks were developed to produce better results when dealing with these problems by introducing addi-

tional layers of abstraction between the input and output neurons. These layers are called hidden layers.
Back-propagation allows for the training of such networks with multiple layers. But training networks with multiple layers proved difficult due to vanishing or exploding gradients. These effects occur when multiplying gradients across many layers, and due to the number of multiplications needed, if weights across the layers have values less than 1, then the gradient ends up vanishing. The vice-versa happens if weights are greater than 1, and a huge number is generated.

This gave rise to a new set of networks called deep neural networks. There are many variations such as:

- Autoencoder
- RNN
- Restricted Boltzmann machine
- SOM
- Convolutional NNs
- Deep Q-Networks and Deep Recurrant Q-Networks.
- Neural Fictitious Self-Play (NFSP)

These networks proposed a new strategy by using a series of single layer networks, like the perceptron, as a way of finding the initial inputs for a multi layered network with various layers such as the Multi-Layer Perceptron.

3 Progress within Games

The field of machine learning has been in constant motion to uncover better, and more effective learning algorithms such as the neural network algorithms in use across many industries today. This section of the article will highlight the progression of these algorithms and their use to tackle problems in a range of game styles over the past decades. We will break down the game styles into 3 distinct categories, board, Atari and 3D GPU intensive games. This will help shed light on current breakthroughs within the field today and provide a basis for evaluating the differences between the algorithms, which is covered in the second part of this section.

3.1 History

3.1.1 Deep Blue

The successful use of artificial intelligence techniques within complex decision-making games goes back a long time. A popular area of AI research is computer chess and one of the most popular projects was IBM's Deep Blue project [Deep Blue reference], a chess playing machine, which ended up beating the world champion, Kasparov in 1997. At the time, the top computer chess programs use more classic search algorithms such as minimax tree search, to sort through all possible moves in each position and their evaluation functions were fined tuned over a long period of time by trial and error from data observed from thousands of expert matches. IBM came out to say that Deep Blue didn't feature any machine learning but others disagreed. (Richard E. Korf, 1997) However, these were very different approaches compared to the machine learning techniques such as the neural networks in use today. Section 3.2.1 contains a more recent approach to the problem of chess.

3.1.2 3D Real-Time Games

3D games of the type that feature various types of game environments and computer controller agents (NPC's), tend to need a different set of algorithms compared to traditional games, in which the sole purpose is to simulate realistic NPC behavior so that the gameplay remains engaging. Traditionally, the methods in these sorts of games could be described as static and relied on methods such as finite state machines and rule based systems, in which the behavior of an agent would be restricted to a set of rules strictly defined by the programmer. Therefore, these systems could not adapt to complex environments or unforeseen circumstances, and overtime players would be able to spot agent patterns which leads to a significant reduction in game play realism and enjoyment. (Di Wang and Ah-Hwee Tan, 2015: 1)

3.2 Recent Trends in Games

3.2.1 DeepChess

Since IBM's deep blue project, chess playing programs have seen big improvements. While none of the new techniques have successfully beaten a world champion, a few have achieved grandmaster level performance. A recent example is a chess program called DeepChess (Omid E. David, Nathan S. Netanyahu and Lior

Wolf, 2016: 1). Unlike its older counterparts and even grandmaster level systems such as FALCON and CRAFTY which need to have extensive prior knowledge of chess rules, piece values, king safety, and various other game details to produce a reasonable evaluation function, DeepChess employs deep neural networks to learn its evaluation function from scratch. To achieve this, the network goes through different stages of training such as, pretraining from a large dataset of thousands of chess games is performed using a deep unsupervised network, then a supervised network is trained to pick the best position from two input positions. The method used in this system claims to obtain grandmaster level playing performance that rivals with better long term tuned evaluation functions such as FALCON and CRAFTY that have competed in World Chess Championship, in which CRAFTY finished in 2nd place. [Omid E. David, Nathan S. Netanyahu and Lior Wolf, 2016: 7]

3.2.2 AlphaGo

One of the biggest AI challenges is the ancient traditional game of Go. The complex, intuition based decision making nature of the game, and the increased board size compared to chess, results in a larger search space rendering traditional techniques useless. A recent breakthrough in tackling this game became the center of attention within the field and the tech industry when Google's DeepMind AlphaGo system beat the world champion Lee Sedol in 2016. Many techniques have been proven successful for tackling the game of Go, such as Monte Carlo tree search (MCTS), minorization-maximization (MM) and Deep Convolutional Neural Networks (DCNN). However, we are interested in the approach used by AlphaGo which was a combination of MCTS and DCNNs which provided a 99.8 percent win rate against other Go programs, and defeated the European Go champion, Fan Hui by 5 games to 0 (Silver et al. 2016: 1) before going on to beat Lee Sedol.

The system consists of two different networks, a policy network for obtaining data on potential moves and a value network for evaluating positions. These networks are put through a pipeline of supervised and reinforcement learning methods (Silver et al. 2016: 3) . Supervised learning is used to train the policy network with data from expert human moves in a large data-set of positions. Reinforcement learning is then used to improve the SL policy network with results from a series of games of self play. The value network is then trained with a prediction of the winner between the games played by the RL policy network.(Silver et al. 2016: 3-7)

The combination of the evaluations of these neural networks with MCTS resulted

in AlphaGo evaluating far fewer positions against Fan Hui (in the regions of thousands) compared to IBM's Deep Blue chess match against Kasparov (Silver et al. 2016: 13) . The techniques presented in AlphaGo resemble bigger similarities to the way an experienced player would make a decision based on past experiences of making similar moves in other matches and their overall outcomes.

3.2.3 3D Games

In recent years, there has been a lot of AI research on 3D real-time computer games, as opposed to just traditional games. The increased consumption of such games due to hardware availability, combined with the desire to be able to interact with a game world which behaves like the real world has lead to this. These games propose a different set of challenges compared to traditional board and atari games. This is because these games usually involve a character navigating larger game environments and handling complex interactions with players, so techniques that rely solely on information from screen states aren't feasible, and neither are traditional state machine and rule-based techniques examined in the last subsection. More recent techniques are needed to achieve an enhanced level of realism in agent behaviour.

An example of a recent technique is FALCON, a self organising neural network based on Adaptive resonance theory (ART). A system that implements this neural network approach is proposed by Di Wang and Ah-Hwee Tan, 2015, to simulate an agent's behaviour in a first-person shooter game called Unreal Tournament. FALCON (Fusion Architecture for learning, cognition and navigation), in this particular implementation, is used to allow an agent to devise strategies and discover the effectiveness of particular weapons in a given situation. The agents can also learn how to play without prior game knowledge and can adapt to new opponents and maps with ease. These networks consist of three input fields. A sensory field that represents states, i.e running around map, fleeing, etc.. A motor field representing behavioural actions and a feedback field for reinforcement values. (Di Wang and Ah-Hwee Tan, 2015: 3) The network is adjusted accordingly, to prefer actions that provide the most rewards from the game environment in a given situation, such as indications of the amount of damage a particular weapon does etc…

Another similar project was proposed by Guillaume Lample and Devendra Singh Chaplot, 2016, which tackled the problem of realistic agent behavior in a 3D environment, but instead uses the Doom game engine. The system's goal is mainly to address the problem of partially observable game states in 3D games. The main difference between the two systems lie in the types of neural networks utilized, in

this case, the author proposed a modification to the Deep Recurrent Q-Network (DQRN) algorithm.

3.2.4 Atari Games

Atari games have also been a popular test bed for advances in new deep reinforcement learning techniques. These new techniques allow autonomous agent to outperform humans playing these games.
A project proposed by Minh. et al, 2013, develops an entirely new deep learning model for reinforcement learning that uses only raw screen pixel information as inputs, and without modification of the learning algorithm for each individual game, produced state of the art results in six of the seven Atari 2600 games in the Arcade Learning Environment it was tested on. A similar project was proposed by Matthew Hausknecht, Joel Lehman and Risto Miikkulainen, 2014, using neuroevolution approaches to tackle Atari Games.

3.2.5 Poker

A new deep reinforcement learning algorithm introduced by Johannes Heinrich, David Silver, 2016, called Neural Fictitious Self Play (NFSP). This method combines fictitious self play with deep reinforcement learning in order to achieve state of the art performance playing Leduc and Limit Texas Hold'Em poker, rivaling algorithms which have pre existing knowledge on the game domain, unlike NFSP which starts with no prior knowledge in a game of imperfect information (Johannes Heinrich, David Silver, 2016: 8). The agent consists of two separate networks, the first network is trained by reinforcement learning of games played against other agents, learning best responses to behaviors of other agents. The second network is trained by supervised learning from the agent's own past behaviors and helps obtain an average of past strategies. (Johannes Heinrich, David Silver, 2016: 2).

4 Conclusion

The aim of this report was to cover the state of the art of machine learning techniques currently being used in the field today. More specifically, we have covered the very recent deep neural networks and their advantages over traditional networks, and supported this by showing recent use cases of these networks within a broad range of game categories such as traditional board and real-time 3D games, and provided a comparison between techniques used in similar styles of games in the past.

5 References

Title	Authors	Year	Link	Criteria
IBM Deep Blue	Murray Campbell, A.Joseph Hoane, Feng-hsiung Hsu.	2001	http://ac.els-cdn.com/S0004370201001291/1-s2.0-S0004370201001291-main.pdf?_tid=1b49564a-f6b2-11e6-a2b0-00000aacb35d&acdnat=1487515730_eb4dcec54bfac6e2a59d448aae3b33a0	Neural Network use in Traditional & Board Games
Does Deep Blue use AI?	Richard E. Korf	1997	http://www.aaai.org/Papers/Workshops/1997/WS-97-04/WS97-04-001.pdf	Neural Network use in Traditional & Board Games
DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess	Omid E. David , Nathan S. Netanyahu, and Lior Wolf	2016	http://www.cs.tau.ac.il/~wolf/papers/deepchess.pdf	Neural Network use in Traditional & Board Games
Mastering the game of Go with deep neural networks and tree search.	David Silver *, Aja Huang *, et al	2016	https://vk.com/doc-44016343_437229031?dl=56ce06e325d42fbc72	Neural Network use in Traditional & Board Games
Deep Reinforcement Learning from Self-Play in Imperfect-Information Games	Johannes Heinrich. David Silver	2016	https://arxiv.org/pdf/1603.01121.pdf	Neural Network use in Traditional & Board Games
Playing Atari with deep reinforcement learning.	Volodymyr Mnih, et al	2013	https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf	Neural Network use in Atari Games
A Neuroevolution Approach to General Atari Game Playing	Matthew Hausknecht, Joel Lehman, Risto Miikkulainen	2014	http://www.cs.utexas.edu/users/pstone/Papers/bib2html-links/TCIAIG13-mhausknn.pdf	Neural Network use in Atari Games
Creating autonomous, adaptive agents in a real-time first person shooter computer games	Di Wang; Ah-Hwee Tan	2015	http://ieeexplore.ieee.org.proxy.library.dmu.ac.uk/stamp/stamp.jsp?arnumber=6849992	Neural Network use in Real-Time 3D Games
Playing FPS Games with Deep Reinforcement Learning	Guillaume Lample* , Devendra Singh Chaplot	2016	https://arxiv.org/pdf/1609.05521.pdf	Neural Network use in Real-Time 3D Games

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$)[1] and especially Go ($b \approx 250$, $d \approx 150$)[1], exhaustive search is infeasible[2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[4], checkers[5] and othello[6], but it was believed to be intractable in Go due to the complexity of the game[7]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[8] and Scrabble[9], and weak amateur level play in Go[10].

Monte Carlo tree search (MCTS)[11,12] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function[12]. The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves[13]. These policies are used to narrow the search to a beam of high-probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play[13–15]. However, prior work has been limited to shallow policies[13–15] or value functions[16] based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification[17], face recognition[18], and playing Atari games[19]. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image[20]. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network.

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1). We begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work[13,15], we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

Supervised learning of policy networks

For the first stage of the training pipeline, we build on prior work on predicting expert moves in the game of Go using supervised learning[13,21–24]. The SL policy network $p_\sigma(a|s)$ alternates between convolutional layers with weights σ, and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a. The input s to the policy network is a simple representation of the board state (see Extended Data Table 2). The policy network is trained on randomly

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK. [2]Google, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA.
*These authors contributed equally to this work.

Creating Autonomous Adaptive Agents in a Real-Time First-Person Shooter Computer Game

Di Wang, *Member, IEEE*, and Ah-Hwee Tan, *Senior Member, IEEE*

***Abstract*—Games are good test-beds to evaluate AI methodologies. In recent years, there has been a vast amount of research dealing with real-time computer games other than the traditional board games or card games. This paper illustrates how we create agents by employing FALCON, a self-organizing neural network that performs reinforcement learning, to play a well-known first-person shooter computer game called Unreal Tournament. Rewards used for learning are either obtained from the game environment or estimated using the temporal difference learning scheme. In this way, the agents are able to acquire proper strategies and discover the effectiveness of different weapons without any guidance or intervention. The experimental results show that our agents learn effectively and appropriately from scratch while playing the game in real-time. Moreover, with the previously learned knowledge retained, our agent is able to adapt to a different opponent in a different map within a relatively short period of time.**

***Index Terms*—Adaptive resonance theory operations, real-time computer game, reinforcement learning, temporal difference learning, unreal tournament.**

I. Introduction

MODERN video games have become a core part of the entertainment world today. The rapidly growing global game industry is valued about 78.5 billion U.S. dollars in 2012, approximately 62% of the global movie industry [1].

Traditionally, game developers tend to utilize scripting techniques, finite state machines, rule-based systems, or other such knowledge intensive approaches to model non-player characters (NPCs) [2]. These approaches often lead to two major limitations. First of all, no matter how skilled the developers are and no matter how long the games have been play-tested before release, the existence of unseen circumstances is unavoidable [3]. In such situations, the decisions of the NPCs are unpredictable and often undesired. As such, it is common nowadays that popular games periodically release series of patches or updates to correct those previously undiscovered loopholes. The second limitation is that these knowledge intensive approaches are static in nature. The behaviors of the NPCs usually follow a few fixed patterns. Once the player learns their patterns and discovers their weaknesses, the game is considered boring, less fun, and less challenging [4].

Furthermore, from the player point of view, invincible NPCs are not preferred [5], [6]. People find it is more enjoyable to play with or against NPCs that try to imitate human players who have flaws but are able to learn from mistakes. If NPCs are able to evolve and dynamically adjust themselves according to the interaction outcomes between different players, the level of player satisfaction increases drastically [7].

Modern video games, especially first-person shooter (FPS) computer games, involve complex and frequent interactions between players and NPCs. Among all the machine learning approaches, FPS game environments are naturally suited for reinforcement learning rather than unsupervised or supervised learning. As reinforcement learning enables the NPCs to be rewarded or be penalized according to different interaction outcomes that are directly derived from the game environment, this trial-and-error philosophy resembles the natural way of human learning and suits complex environments well [8].

We create agents (in this paper, the term NPC is interchangeably used as agent or bot) that employ fusion architecture for learning, cognition, and navigation (FALCON) networks [9] to play an FPS computer game. FALCON is a generalization of the adaptive resonance theory (ART) [10] to perform reinforcement learning. By utilizing respective FALCON networks, our agents learn and apply rules for both behavior modeling and weapon selection during run time.

Our game of choice is Unreal Tournament 2004 (UT2004), which is a well-known commercial FPS computer game. UT2004 has been directly used as the application domain of much research work (elaborated in Section II). The UT2004 game server provides interfaces for two-way communications, such as passing the relevant game information to the user and receiving user commands to control the avatar in the game. To make our implementation work easier, we use Pogamut [11], which is a freeware to facilitate rapid developments of agents embodied in UT2004.

To deal with the partially-observable information (elaborated in Section IV-C) provided by UT2004, in this paper, we propose a set of combinatorial operations for FALCON networks to replace the conventionally applied operations. Our agents employ two respective FALCON networks to learn how to select appropriate behaviors and how to choose effective weapons under different circumstances. In the experiment section, we first show that the proposed combinatorial learning operations enable our agents to learn (from scratch in real-time) more appropriately in terms of behavior modeling and effectively in

Manuscript received July 30, 2013; revised December 20, 2013, March 14, 2014, and June 09, 2014; accepted June 28, 2014. Date of publication July 08, 2014; date of current version June 12, 2015. This work was supported by the National Research Foundation, Prime Minister's Office, Singapore, under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office.

The authors are with the LILY Research Centre and the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: wangdi@ntu.edu.sg; asahtan@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2014.2336702