American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)
ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

http://asrjetsjournal.org/

Evaluating the Efficacy of Deep Neural Networks in Reinforcement Learning Problems

Amir Girgis*

Jumeirah College, Sheikh Zayed Road, Dubai, UAE
Email: amir.amgad01@gmail.com

Abstract

The deep learning community has greatly progressed towards integrating deep neural nets with reinforcement learning, in what is termed 'deep reinforcement learning.' This project aims to investigate the importance of deep neural networks in reinforcement learning. It analyzes the role that deep learning plays in tackling a range of different reinforcement learning problems. By analyzing and evaluating different methods (like Monte Carlo Tree Searches and model-based methods), the project refutes the popular claim that deep reinforcement learning is always the best option to tackle certain problems and explores research papers that support this hypothesis. It identifies the current limitations of deep neural nets, such as overfitting, sparse/shaped reward functions, and sample inefficiency. The project also discusses the potential of Deep-Q Networks, and surprising results in various domains. Thus, in an attempt to compare the merits and problems of deep learning, the project determines the degree to which neural networks are useful in reinforcement learning problems, both now and in the future. Taking the AlphaGo algorithm (and how it beat world Go champion Lee Sedol) case study as a starting point, the project unveils the potential of deep reinforcement learning despite the many challenges it faces today. Therefore, it also aims to come to a conclusion about how deep neural nets in reinforcement learning is likely to develop in the future as data becomes increasingly available and hardware becomes cheaper.

Keywords: Deep Neural Networks; Deep Reinforcement Learning; Model-based methods; Monte Carlo Tree Search; AlphaGo; Q-learning; Deep-Q Networks.

1. Introduction

1.1 Neural Networks

Neural networks are a machine learning paradigm that are trained against large sets of data to learn some expected output.

---

* Corresponding author.

They originated alongside the discovery of the perceptron [1] in 1958 which is an algorithm, inspired by the brain, capable of classifying and discriminating. The perceptron takes an input of a matrix, multiplies it by a set of numbers called weights, and then adds a bias. This operation allows the algorithm to discriminate and classify the input by optimizing the set of weights and biases. The perceptron is then activated: an activation function is run to determine whether the perceptron will fire or not (mimicking the structure of the brain) by checking against some threshold – this is used to convert an input signal into an output signal and account for functions with polynomial degrees greater that just one. A neural network is a multi-perceptron computation graph that consists of three components: the input layer, hidden layers, and the output layer. At each hidden layer (the number of hidden layers varies depending on the complexity of the task), the operations described above are carried out and the result is passed through the nodes until it reaches the output layer. For example, to classify hand-written digits, the image would be fed into a neural network in the form of raw red-green-blue pixels. The neural network would then learn a set of weights and biases such that when multiplied by the respective input number, it produces a result that discriminates between each hand-written digit. A deep neural network is a neural network that contains two or more hidden layers in a computation graph. Although in the past neural networks were more popularly used for binary classification tasks (like recognition or diagnosis), they have become an increasingly popular method for attacking common reinforcement learning problems.

1.2 Reinforcement Learning

Reinforcement learning is one of the three subsets of machine learning (including supervised and unsupervised learning) that involves teaching an agent in some environment to reach an optimal policy. It encompasses many broad disciplines such as video-games, game-theory, and genetic algorithms. In contrast to supervised learning, which is normally a discrete task, or unsupervised learning, which is a clustering or association task, reinforcement learning teaches an agent the best actions to take based on some feedback the model receives, in the form of a reward or punishment. Any reinforcement learning problem can be modelled as a Markov Decision Process [2]. This breaks down any environment or problem into several key components.

Agent - An agent is a player who is able to take actions and interact with the environment (for example the snake in the game of snake).

Actions - Actions are the set of all possible actions/events that can be taken in the environment – the agent is free to choose which action to take (for example moving up in the game Snake).

Environment - The environment is the world in which the agent is in - it contains states an agent can be in and the actions that can be taken by the agent.

States - States are positions or situations in an environment (like a square on a chess board). The states and actions are fed in, as inputs, to the agent to make a decision.

Rewards - Rewards are the feedback that is returned for each action or state. It helps the agent identify what is a good or bad move/state.

Policy - A policy is a strategy employed by an agent to decide the next actions to take and optimize a function, which maps states to actions.

Q-learning - Q-learning is a reinforcement learning algorithm where an agent is in an unknown environment and does not know the reward for any actions to take/states to be in. This allows the agent to learn an optimal policy and maximize the expected value of the reward over several permutations. These algorithms are often referred to as Deep-Q Networks (DQNs) which combine neural networks and Q-learning to approximate an accurate reward function of the environment. This is especially important when tackling an unknown problem because it defines the goal of the game for the agent– to maximize a score, to defeat the boss, to finish a race etc. These rewards help an agent choose the actions that can best accomplish the problem by creating an effective strategy - the policy. Algorithms which handle these unknown environments are referred to as model-free methods because they have no previous existing knowledge of the environment.

Although neural networks have revolutionized the way reinforcement learning algorithms are structured, one must investigate the extent to which they are effectively used and whether other alternatives, like domain-specific techniques or model-based methods, can outperform deep neural network architectures. Moreover, with the rapid developments that researchers have made - such as with the latest AlphaGo Zero algorithm - the future of deep reinforcement learning is still uncertain and this paper aims to explore how this is likely to evolve in the future.

The limitations of deep reinforcement learning are most clearly seen in several papers, which yield results that outperform traditional deep neural architectures, with alternatives like Monte Carlo Tree Searches, in tasks like trajectory optimization and Atari games (like Space Invaders). This dissertation will discuss the merits and problems of deep reinforcement learning, allowing the reader to determine the extent to which deep neural networks are effective.

To fully determine an answer, it is vital to compare the results from research papers and notable professors/researchers with my personal opinions and judgements, supported by evidence. By coherently analyzing all of this, one will be able to determine the extent of deep neural networks' importance in reinforcement learning.

2. Literature Review

2.1 Introduction

The theory of reinforcement learning is deeply rooted in psychological perspectives and animal behavior, which discusses how agents can optimize their environment. Although reinforcement learning has had success in the past [3], these model-based methods have often relied on hand-crafted features that are unable to successfully generalize to different reinforcement learning problems. This paper aims to evaluate how deep reinforcement learning models (deep neural networks paired with reinforcement learning) perform on these problems and compare them to alternatives to reach a conclusion about their effectiveness and their potential in the future. As discussed earlier, a modelled Markov Decision Process (MDP) contains states and action pairs. These are used

by the model to evaluate and create an optimal policy. With every state-action pairs, there is an associated reward that provides feedback to the model, allowing it to fine-tune its policy decision making process. When a game becomes complex with many possible state-action pairs, it becomes difficult to store every Q-value (the associated reward with the state-action pairs). This is where deep neural nets are used in reinforcement learning. In model-based algorithms, where the situation has been modelled and the Q-value is known for each state-action pair, deep neural nets are used to store the Q-values given an action [4]. Thus, when a Q-value is needed, it is fetched from the neural net to be used by the model. However, for large, complex environments with high-dimensional sensory inputs like vision/speech (like Atari games), it is computationally expensive to store a table with many state-action pairs and their rewards. This is why deep learning models (combined with Q-learning) are used to approximate functions for rewards that can be used in a given environment.

2.2 Challenges with Reward Functions

In the past, deep neural networks struggled with creating accurate function approximations for rewards for three reasons [5]. Firstly, these models were unstable because of correlations present in the sequence of observation. In order to accurately train a model in a complex environment, the raw pixels of each frame are fed into the model as input. Consider a video game; if you take several screenshots one after the other, the images will be highly correlated. When a deep neural net is trained against these very similar inputs in sequential order, it results in overfitting. Secondly, a small update to a Q-value may change the data distribution of all the other reward values. This may cause the agent to change an optimized policy and, once again, lead to overfitting. Finally, there may be inconsistencies between the target action-values and the predicted values determined by the model. This may lead to unwanted correlations which could result in an unstable model.

2.3 Deep-Q Networks

The Deep-Q Network (DQN) solved these three problems by introducing two features. The first was a biologically-inspired mechanism - termed experience reply - that randomized the input data by shuffling the different frames. This would be equivalent to randomizing the order of the screenshots of the video game. Furthermore, this mechanism allowed the agent to replay past experiences enabling for more efficient learning. By doing this, the DQN removed any correlations between the sequence of frames, thereby avoiding overfitting and preserving the data distribution of Q-values (action values). Secondly, the DQN avoided correlations between target and predicted action values by using a periodic update, which only adjusts the Q-values at regular intervals in each training cycle.

At the time of publishing in 2015, the DQN produced very promising results that could not only outperform the state of the art linear reinforcement learning algorithms in 43 of the 49 Atari games, but was also comparable to human-level performance in 23 of the games. These results are particularly promising because not only did a combination of deep neural nets and Q-learning achieve them, but the algorithm was also model-free and was trained on each game with no existing knowledge of the environment. Based on these results, it’s clear that deep neural nets had potential when attacking reinforcement learning problems.

Since the introduction of Deep-Q Networks by Google DeepMind in 2015, there have been a number of improvements that have addressed data efficiency [6], as well as other concerns, that culminated in project Rainbow [7]. This model attempts to combine the improvements made into one DQN agent by incorporating seven models: the DQN, double DQN, N Step Q-Learning, Prioritized Experience Reply, Dueling Q-Network, Distributional Reinforcement Learning, and the Noisy Network. Each model mentioned was developed to solve a specific problem that Deep-Q Networks faced and thus, project Rainbow aimed to combine these models to see, whether together, these seven models could produce better results than the traditional DQNs and the separate improved models.

The results produced by Rainbow were almost a four-fold improvement. Rainbow achieved better than human-performance using 44 million frames while the next best performing individual model (the Distributional DQN) reached the same level in over 200 million frames. Furthermore, with just 7 million frames, Rainbow reached the same level of performance as the traditional DQN after 200 million frames. Thus, by combining previously independent models, Rainbow was able to address many pressing concerns like sample inefficiency, overestimation bias, and limited exploration to produce promising results.

2.4 Domain-Specific Techniques

2.4.1 Monte-Carlo Tree Search

Although DQNs have shown significant improvements and are continually evolving, there remain both key problems with respect to reinforcement learning, and alternative model-based methods that outperform deep reinforcement learning models. Guo and his colleagues (2014) NIPS [8] showcased how a simple version of Monte-Carlo Tree Search (UCT – Upper Confidence bound applied to Trees) was able to outperform a Deep-Q Network in six Atari games. In many reinforcement learning problems, a probabilistic search algorithm like MCTS would have a significant advantage over a DQN because it's able to look ahead and see the possible moves it could play, as well as the corresponding results it would have. This is very different to a traditional DQN, which is unable to simulate these moves and thus, has significantly less knowledge when choosing the optimal move at any given timestamp. Although DQNs (as model-free algorithms) have shown significant improvements in tackling unknown environments, their generalization capabilities come at a price: because they enter into an unknown environment with no existing knowledge and search capabilities, they can still get easily outperformed by domain-specific algorithms like a MCTS. Thus, even though research continues to develop these agents and improve on previous models, classical domain-specific techniques may still outperform these agents in certain reinforcement learning problems.

2.4.2 Trajectory Optimization

A second example of classical domain-specific techniques performing on-par with deep reinforcement learning is the task of trajectory optimization. In general, this is concerned with a set of mathematical techniques used to find the ideal behavior of a dynamic system [9]. This can include tasks like creating a function to allow a robot to walk, finding trajectories between planets and asteroids, and simulating skeletal agents that attempt to use

their limbs to achieve a certain goal. Kelly goes through a popular example (termed the 'Swing-Up problem) that highlights what underpins any trajectory optimization task. Consider a cart system that travels along a horizontal rail. It is moved by applying a force that pushes the cart forward and attached to the cart is a pendulum suspended vertically downward. The possible task would be to find the force applied as a function of time that will move the cart and in doing so, vertically suspend (upwards) the pendulum. More recently, researchers have explored the potential of deep reinforcement learning in continuous action spaces (as opposed to discrete low-dimensional spaces like Atari games) that involve tasks such as animal/humanoid locomotion.

In the Heess and his colleagues (2017) paper on Emergence of Locomotion Behaviors [10], the DeepMind team used deep reinforcement learning algorithms to test three control tasks: two locomotion tasks (in obstacle free environments) and a planar target-reaching task. The results that the DeepMind team observed [11] were very similar to those Tassa and his colleagues (2012) [12] achieved using online trajectory optimization (which was running on hardware that was five years older). Thus, although the deep reinforcement learning algorithm was able to achieve very promising results, a model running on less advanced hardware - 1 CPU compared to 64 CPUs in Heess and his colleagues (2017) paper - was able to achieve the same results because of a key technique that deep reinforcement learning lacks: model predictive control.

2.4.3 Model Predictive Control

Model predictive control (MPC) is a feature of online trajectory optimization that uses a model to predict a future output and select the optimal action. It is vital for two reasons. Firstly, MPC has preview capabilities that provide information about the environment before selecting the next action to take. In the trajectory optimization papers discussed above, this would allow the agent to see the terrain/obstacles and simulate different actions and the corresponding results that would occur with those actions (using a ground-truth model) before choosing the optimal action to take. Secondly, an MPC model allows constraints to be satisfied. This means a model can trace an optimum trajectory while satisfying certain constraints that, if not met, may otherwise cause an agent to deviate from the desired goal. Combined, these two features give online trajectory optimization models a significant advantage over a deep reinforcement learning model, which can only learn by randomly experimenting with the environment to receive rewards. This is susceptible to overfitting and is less efficient than in the Tassa and his colleagues (2012) model, which explains why five years of research and development in hardware and software could only produce similar, and not better, results with deep neural nets. Therefore, as with MCTS, generalization capabilities of deep reinforcement learning pay an expensive price and compromise their performance on certain reinforcement learning problems. Although deep reinforcement learning is tackling the problem at a disadvantage (because it does not use model predictive control) and is still performing surprisingly well, it cannot (at the moment) compete with a model-specific algorithm and produce better results. While deep reinforcement learning is an avenue for research and can solve many potential problems in the future, it's important to identify when specific problems yield more beneficial results by choosing an alternative path to deep neural networks. This is why AlphaGo received worldwide attention by the media and researchers when it beat world Go champion Lee Sedol (who has won 18 world titles) – because it was an "unadulterated win for deep reinforcement learning" [13] - and can provide useful insights about about how the influence of deep neural nets, in reinforcement learning, is likely to evolve in the future.

3. Discussion

Since the creation of the Deep-Q Network, which galvanized the introduction of deep reinforcement learning (enabling one of the first novel use of deep neural nets in the realm of reinforcement learning), researchers have continued to build on previous models, solving new problems, and establishing new conclusions. In the world of reinforcement learning, problems are of variable complexity and can range from simple two-dimensional games that are based on some key controls, to high-complexity simulation environments. It's important to understand that in the current field, there is no single model that can be applied to any reinforcement learning problem and beat the state-of-the-art results. Rather, the field is trying to move in the direction of versatile model-free agents that can be applied to a number of different problems and achieve good performance. While models like Rainbow and the DQN have had success in the past, these models have often focused on two-dimensional sensory games (like the Atari games) or simple three-dimensional games (like Labyrinth – a three dimensional navigation and puzzle solving environment). Although the Arcade Learning Environment (ALE) provides a stable and reliable benchmark for testing (with methods of interfacing common arcade games, like Atari games, for training and testing models), it also limits the scope of reinforcement problems to a set of definable games. Thus, it's necessary to evaluate deep reinforcement learning against a wide array of problems to determine the extent to which deep neural nets are successful.

This dissertation will discuss each aspect that needs to be considered to reach a final conclusion, such as the key problems underlying deep reinforcement learning - including overfitting and reward systems- as well as the unprecedented successes (like with AlphaGo). By the end of the discussion and in my conclusion, I will have my established my opinion on the degree of importance of deep neural nets in reinforcement learning by drawing on relevant data and evidence. In addition, I will discuss the future of deep reinforcement learning and the challenges it must overcome to truly generalize successfully to a wide-array of different problems.

3.1 Importance of Deep Neural Nets for Q-Learning

Examining previously mentioned successful models (in the realm of deep reinforcement learning) reveals the underlying importance of deep neural nets. They are the fundamental block that allow models to perform so well in a high-dimensional sensory environment where storing billions of Q-values is simply not computationally viable. In trying to approach unknown and new environments, a reinforcement model must be able to develop a system to provide reliable rewards, and with the help of deep neural nets, successful models were able to combine deep learning with Q-learning and create accurate approximations. Unfortunately, as is common with many tasks, not all problems have a clear definable reward system that enable models to a) reach the optimum solutions and b) avoid any environmental nuances. This opens up deep reinforcement learning models to two key problems: overfitting and reward functions. Understanding these problems may provide a possible explanation for why classical domain-specific techniques are still able to compete with and outperform deep reinforcement learning models.

3.2 Overfitting

To be able to develop versatile policies for a number of different environments, an agent must be able to generalize successfully; that is, be able to infer the results from some sample of events, and apply it to unknown and unseen occurrences. Since the introduction of machine learning and statistics, overfitting has always been a significant problem for researchers that often limited models' capabilities to extrapolate data and information. Unlike the human brain that is able to generalize both quickly and successfully, machine learning models still face this problem today. No different from them, deep reinforcement learning is particularly prone to overfitting. In trying to develop versatile solutions, models must be able to adapt to completely different environments, actions, and reward systems. However, as with any game, each environment is heavily nuanced in its design and will often contain potentially unique patterns to it. What this means for deep reinforcement learning models, is that they almost always end up fitting to the unconventional patterns of each environment. Often, the optimized policies are too narrow in scope and rely on highly specific features of the environment that are not always present. In the Lanctot and his colleagues (2017) paper on multi-agent reinforcement learning, agents were trained in a simple game of laser tag over five random seeds. In each respective game, agents learned to move, shoot, and run away from the opponent. Each set of five random seeds is one experiment and the researchers carried out the experiment twice. Although within each experiment, the results shown were beneficial and the agents were successfully shooting and defending, when the agent from one experiment was pitted against the agent of another experiment, the results were starkly different. Not only did the agents not shoot or defend, but they also didn't even try to move closer to the opponent and kept moving in random paths. This occurred because during the five-random seeds, although the agent was successful, it was overfitting to unconventional patterns in the environment (in this case, the presence of the other unchanging opponent). Thus, although the layouts and actions did not change, the model's narrow policy was unable to generalize to a different opponent because it had learned the nuances of its initial environment. This is intensified by the underlying mathematical nature of a deep reinforcement learning algorithm. Since the target reward values are not known and the model is continually updating its reward function to create a more accurate feedback system, the Q-values (the reward values) are constantly changing. Thus, rapidly exploring the environment may bring about a change that causes the network to specifically try to fit to, losing its generalization capabilities.

There is a very popular dilemma, in reinforcement learning, known as the exploration-exploitation trade-off (Sutton, Barto, 1998), which often contributes to the effect of overfitting or underfitting in certain circumstances. Essentially, models (when choosing the next optimal action) have two choices: exploring the environment to collect more information and make a more informed decision or using the current information at hand to take the best action. If the agent chooses to explore, it may develop a highly specialized policy that will not perform well on unseen occurrences (similar to overfitting). However, if the agent chooses to take the best action at hand, it risks getting stuck in a locally optimum strategy that does not yield higher rewards (similar to underfitting). For deep neural nets to achieve an optimal policy and an appropriate balance between exploitation and exploration, they require millions of frames of input data. This raises a pressing concern with data efficiency. For example, in Hessel and his colleagues (2017) Rainbow, despite the model achieving improved results, it took 44 million frames to beat human-level performance which roughly translates to 83 hours of gameplay. This is a task that most humans could pick up in a couple of minutes. It's important to note that, we as humans, have a significant advantage over a Deep-Q Network playing a game because we approach it

knowing what a game is, what the score is, what actions (e.g. shooting ships down) increases our score. A DQN or deep reinforcement learning model has to learn all of this information from scratch. However, if an alternative model-based method can provide better results, should researchers use other deep reinforcement learning model-free algorithms? That is, should a deep reinforcement learning model that is highly susceptive to overfitting, limited in scope to certain problems, and not necessarily able to out-perform the state-of-the art results in a reasonable time frame be prioritized over an alternative model? As deep reinforcement learning algorithms continue to evolve, they will inevitably improve in performance and data efficiency. However, given the nature of approaching an unknown problem, models will always require considerable data to learn enough information for an optimal policy. This is why sample efficiency (although has improved considerably over the past four years) is often a problem for deep neural nets; only when enough data is used can a deep learning model avoid overfitting (to an extent) and find an optimal policy.

3.3 Reward Functions Design

The second key problem with deep reinforcement learning models is reward functions. At the core of any reinforcement learning problem is the concept of rewards – what essentially allows the agent to respond to and interact with different states and actions in the environment. However, this also means that the reward system of a model often defines how successful it is. It's imperative to develop a model that provides easy, progressive rewards that guide an agent to reaching the optimum solutions. For any DQN or deep learning model, the agent is trying to optimize a fundamental score or value it is being given. Unfortunately, not all reinforcement learning problems can be constrained to one defining variable. As games grow in complexity, opening up changing environments, new nuances and obstacles, and a greater number of possible actions to take, reward functions become increasingly difficult to quantify for an agent. It means that there is no one variable the agent is cross-checking against to increase or decrease.

For example, consider a simple car racing game where users race against each other to level up; let's also consider that 'coins' or 'power-ups can be picked up while driving in any game that can be used to upgrade cars and buy other features. The problem then becomes, what should the agent optimize? Should it be collecting as much cash as possible? Would that potentially lead to the car driving slower to collect all the coins and coming last in the race? Or should the variable be the speed of the car – the faster its driving the greater the reward? In essence, reinforcement learning problems cannot always be constrained to one single variable. This can also be applied in locomotion or simulations - such as Heess and his colleagues (2017) paper on Emergence of Locomotion Behaviors, where there is no one defining variable that will lead to the solution. Accordingly, problems like these have led to the introduction of reward engineering, where researchers try to develop reward systems that can model for complex environments. Unfortunately, not only do these deep learning algorithms rely on model-based methods (since one would have to know the problem beforehand), they are also highly complex and time-consuming to design.

There are two common reward systems often adopted by models: shaped rewards and sparse rewards. Shaped rewards are rewards that increase as the agent nears the desired goal while sparse rewards offer no rewards until the agent has achieved the desired goal. Despite shaped rewards being an apparently more effective method,

which provide progressive feedback to agents allowing them to continue moving in the right direction, they can also lead to unwanted behaviors [14]. This is evident in the example described above where the presence of power-ups or coins may increase the score and cause agents to slow down and lose the races. On the other hand, sparse rewards can avoid this problem, but this also means agents are less inclined to explore (since they will receive no reward) and more inclined to develop a locally optimum strategy. As with all aspects of machine learning, it becomes a trade-off and requires careful engineering of a reward system for agents to perform well. However, this is often extremely time-consuming and not that reliable. This is most clearly see in Popov and his colleagues (2017) [15] where the model, instead of picking up the block (in a simulation environment with a robotic arm), pushes it until it is vertical.

In all of the papers released on DQNs and their improvements, the models have always used the Atari games as a benchmark. This provides a reliable comparison between models and using ALE, it also provides a seamless integration between the games and the model. However, all of the Atari games are scored based games. Thus, regardless of how complex or simple the environment is or the number of actions available, the higher the score, the better the player or the model. This means the reward function provides a direct and easy route for the agent to experiment and then optimize. Put simply, whatever the agent does, if the score increases, it should keep doing it. If the score does not change, it should stop and find another action to take. Although it's still impressive that deep learning models are able to perform at such a high level given only the pixels as input, these games represent a small portion of reinforcement learning problems and are not a complete representation of deep neural nets' success in the realm of reinforcement learning (as seen with online trajectory optimization).

3.4 AlphaGo

Yet, with all of its problems, deep reinforcement learning has not failed to surprise us. In 2016, with the power of deep neural nets, AlphaGo beat Lee Sedol – considered to be one of the best players in the past 50 years. For the first time in the artificial intelligence community, deep reinforcement learning had earned a victory. In the past, the game of Go had always been a challenge for the artificial intelligence community because of the number of possibilities – there were more combinations of pieces on a board than atoms in the known universe. The game Go consists of a 19 by 19 board and players fight to gain the most territory. Given the vast network of possible moves, the DeepMind team realized that brute force was not an option and that the power of deep neural nets paired with a MCTS could solve this problem. AlphaGo went through several versions, two of which will be discussed: the version that beat Lee Sedol and AlphaGo Zero, the latest and most impressive version.

The version of AlphaGo that beat Lee Sedol consisted of two main networks: a policy network that was trained on expert moves (using supervised learning) and taught the agent to mimic the playing style of professionals, and a value network that predicted (at each game state) who the winner would most likely be (white or black). These two deep neural networks were paired with a MCTS that provided a look ahead search. Since the number of possible combinations is so vast, the DeepMind team developed the search so that it only focused on a handful (of around 20) of the most promising moves. Because of the design of the network and the deep neural nets used for training, AlphaGo was able to think of strategically unique ways that were fundamentally different to the history of teaching of Go. For example, in the second game, AlphaGo played a surprising move (move

37) that shocked most of the spectators. For professional Go players, this move appeared to be a "strange mistake" as Lee Sedol said. Indeed, this move turned the course of the game as it offered a strategic midpoint to build and expand AlphaGo's territory. With the power of deep neural nets, AlphaGo had developed an artificial intelligence that re-defined many of Go players' key ideologies after the games.

The final version of AlphaGo, titled AlphaGo Zero, consists of only one neural network that only takes the positions of the white and black tiles as input, incorporates tree search within the loop, and most impressively, only uses self-reinforcement learning. This means it's not trained on any professional players or moves, it simply learns by playing millions and millions of games against itself. Furthermore, this model achieves better performance than the last version. To the community at the time, this model was truly a shock because deep reinforcement learning had not made such a giant leap since the introduction of the Deep-Q Network. Many challenges that researchers did not think possible, such as tackling the game of Go, have been proven and continue to be proven. Thus, while there are many challenges that reinforcement learning faces today, it is clear that the influence of deep neural nets has had (and will continue to have) a great influence on the successes of reinforcement learning problems across all fields.

4. Conclusion

In conclusion, although deep reinforcement learning faces many challenges today, it has also achieved many success, tackling problems that were once thought to be impossible and outperforming models that were once thought to be state-of-the art. One can confidently conclude that the success of deep reinforcement learning is largely attributed to the power and influence of deep neural nets, which have allowed for more accurate reward function approximations and more intelligent policies.

However, in order to fully solve the problem of deep reinforcement learning; that is, develop a model that can tackle any unknown environment, game, or simulation, deep reinforcement learning must solve/incorporate three key characteristics.

Firstly, reward functions must be learnable. As discussed earlier, in some games or situations, the community has not figured out a method that allows for a learnable reward function that avoids both overfitting and unwanted behaviors. In order to be able to approach any problem without any initial knowledge, deep reinforcement learning must improve on the current most popular method of Q-learning.

Secondly, models should introduce self-play. This is a very important idea that is one of the main reasons behind the success of AlphaGo. Introducing self-play allows for many more games to be played and as with any machine learning problem, more data yields more accurate results.

Thirdly, hardware needs to become cheaper or models need to become more efficient. All of the research papers discussed in the literature review such as the DQN, Rainbow, or Emergence Locomotion were often using expensive hardware. Over time, technology will become affordable and until hardware reaches a level of affordability where anyone can buy many CPUs and GPUs for a relatively affordable price, or where models become more computationally efficient, deep reinforcement learning will continue to be from a monetary and

computation perspective, expensive.

Combined, these three characteristics will allow deep reinforcement learning to continue to improve on current models and solve new problems. In the future, one can expect research to begin to tackle and solve these challenging problems, allowing deep neural nets to flourish once more within the domain of reinforcement learning and have real-life applications. Thus the future of deep reinforcement learning is optimistic and given the pace at which research occurs in such a dynamic field as machine learning, only time will tell just how fast deep reinforcement learning will improve. However, at the moment, while deep reinforcement learning has not achieved the best results across every task trialed, one can only attribute its consistent performance across these domains to deep neural nets, which have solved many problems in the reinforcement learning community.

References

[1] F. Rosenblatt. “The Perceptron: A probabilistic model for information storage and organization in the brain.” Psychological Review vol. 65, no. 6, pp. 1-6 1958

[2] Skymind. “A Beginner's Guide to Deep Reinforcement Learning.” Internet: https://skymind.ai/wiki/deep-reinforcement-learning [24 August 2018]

[3] M. Riedmiller, T. Gabel, R. Hafner and S. Lange. "Reinforcement learning for robot soccer", Autonomous Robots, vol. 27, no. 1, pp. 55-73, 2009.

[4] A. Gosavi, EMGT 457. Class Lecture, Topic: ‘Neural Networks and Reinforcement Learning’, Rolla, MO 65409, Department of Engineering Management and Systems Engineering, Missouri University of Science and Technology, 2018

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. 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, 2015.

[6] T. Schual, J. Quan,, I. Antonoglou, D. Silver. “Prioritized Experience Replay” in 6th Int. Conf. on Learning Representations, Vancouver, Canada 2016

[7] M. Hessel. et. al. “Rainbow: Combining Improvement in Deep Reinforcement Learning” presented at the AAAI Conf. on Artificial Intelligence, New Orleans, Louisiana, 2018

[8] X. Guo, S. Singh, H. Lee, R. Lewis, X. Wang. “Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning” in Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, 2014

[9] M. Kelly. “Introduction to Trajectory Optimization” Internet:

https://www.youtube.com/watch?v=wlkRYMVUZTs, May 1, 2016 [Sep. 2, 2018]

[10] N. Heess. et. al. "Emergence of Locomotion Behaviours in Rich Environments" in CoRR abs/1707.02286, 2017

[11] N. Heess. et. al. "Emergence of Locomotion Behaviours in Rich Environments" Internet: https://www.youtube.com/watch?v=hx_bgoTF7bs, July 14, 2017 [Sep. 2 2018]

[12] Y. Tassa, T. Erez, E. Todorov. "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization" in IEEE/RSJ Int. Conf. on Intelligent Robot and Systems, 2012, pp: 4906 - 4913

[13] A. Irpan. "Deep Reinforcement Learning Doesn't Work Yet." Internet: https://www.alexirpan.com/2018/02/14/rl-hard.html, June 24, 2018 [Sep. 2 2018].

[14] J. Clark, D. Amodei. "Faulty Reward Functions in the Wild." Internet: https://blog.openai.com/faulty-reward-functions/, Dec. 21 2016 [Sep. 5 2018].

[15] I. Popov. et. al. "Data-efficient Deep Reinforcement Learning for Dexterous Manipulation" in CoRR, abs/1704.03073, 2017