Performance Enhancement of Deep Reinforcement Learning Networks using Feature Extraction

Joaquin Ollero and Christopher Child

City, University of London, United Kingdom

Abstract. The combination of Deep Learning and Reinforcement Learning, termed Deep Reinforcement Learning Networks (DRLN), offers the possibility of using a Deep Learning Neural Network to produce an approximate Reinforcement Learning value table that allows extraction of features from neurons in the hidden layers of the network. This paper presents a two stage technique for training a DRLN on features extracted from a DRLN trained on a identical problem, via the implementation of the Q-Learning algorithm, using TensorFlow. The results show that the extraction of features from the hidden layers of the Deep Q-Network improves the learning process of the agent (4.58 times faster and better) and proves the existence of encoded information about the environment which can be used to select the best action. The research contributes preliminary work in an ongoing research project in modeling features extracted from DRLNs.

Keywords: Reinforcement Learning, Neural Networks, Deep Learning, Feature Extraction, TensorFlow.

1 Introduction

Deep Learning is a class of Machine Learning algorithms based on learning data representations based on Artificial Neural Networks (ANN). By using an architecture such as a Deep Neural Network [1], it is possible to extract information from neurons placed in the hidden layers of the network that automatically encode valuable features from the raw data used as inputs. Reinforcement Learning algorithms, as opposed to supervised and unsupervised learning, present a mechanism to train an artificial agent in the learning process of how to solve a specific problem. Overall, a Reinforcement Learning algorithm maps every state/action combination within a given environment to a specific value. This value will inform the agent how good or bad taking each action is in relation to the next state that the agent will experience straight afterwards. Q-Learning is a popular Reinforcement Learning technique, used to produce optimal selections of actions for any Markov Decision Process [2]. Reinforcement Learning using Deep Learning Neural Networks is currently receiving intense media attention. There is growing interest in this area of artificial intelligence since [3] introduced Deep Q-Networks (DQNs) and the research was published in the multidisciplinary scientific journal Nature. The presented work proved that the proposed model achieved a higher level of expertise than that of a professional human game player in a set of 49 Atari 2600 games [4]. The field of Machine Learning is evolving rapidly in terms of revolutionary applications and groundbreaking research that is leading the area to tremendous popularity. Along with this, open source tools that allow scientists to experiment with algorithms and structures are becoming more accessible and straightforward to learn and use. One of the Machine Learning libraries that is experiencing the biggest growth is TensorFlow [5], an open-source software library for Machine Intelligence developed by Google.

The objective of this paper is the training of a DRLN on features extracted from an DRLN trained to solve a similar problem with the Q-Learning algorithm. Emphasis have been given to the detailed study of this structure by extracting features from its hidden layers in order to use them in an environment modelling algorithm. The state space can be reduced by predicting the future states of these features and using this as a model, replacing the original environment. The system will model features extracted from an initial level of a Deep Neural Network, using them as inputs of a reduced Deep Neural Network (in terms of number of layers) to prove that the information of the inputs is automatically encoded and preserved and can be used to increase the speed of learning of the agent. Performing feature extraction from a later level of a Deep Neural Network will allow the system to classify the codifications and verify if they can be used to predict future states of the features.

The rest of the paper is structured as follows. A literature review that covers research on Deep Reinforcement Learning Networks is presented in the next section. The problem statement, the Q-Learning algorithm and the structures in which the technique is implemented, a Deep Q-Network and a Deep Q-Network using Feature Extraction, are presented on Section 3. Results, introduced on Section 4, are stated in terms of testing the behaviour of the agent and if the overall learning process is improved in terms of time. A thorough analysis of the model, overall considerations and a set of extensions that would improve and contribute to the research are discussed in the last section of this paper.

2 Context

The renaissance of Reinforcement Learning is largely due to the emergence of Deep Q-Networks [3]. The Deep-Q Network framework [3] was motivated due to the limitations of Reinforcement Learning agents when solving real-world complex problems, because they must obtain efficient representations from the inputs and use these to relate past experiences to the next situations the agent will be presented with. The developed framework, a combination of Reinforcement Learning with Deep Neural Networks, was able to effectively learn policies directly from the inputs. More precisely, the agent was tested on 49 classic Atari 2600 games [4], receiving only the pixels of the image and the game score as inputs, performing with a level comparable to the one of a proffesional human player. The research introduced the first intelligent agent that was able to learn how to solve a set of different tasks with groundbreaking results.

Several extensions have been proposed to the work presented in [3]. The adaptation of the Deep Q-Network framework using the Double Q-Learning algorithm has reduced observed overestimations [6]. By prioritizing experience replay, a technique that lets the agent review important transitions more frequently, the learning process of the agent can be improved in terms of efficiency [7]. In contrast to architectures such as Convolutional Neural Networks or Autoencoders, a new structure, termed Dueling Network Architecture, was introduced to prove the generalization of learning accross actions without performing any change to the Reinforcement Learning algorithm [8]. All these works have produced agents that perform better in the Atari 2600 games domain in comparison with the original Deep Q-Network. Finally, the original authors of Deep Q-Networks [3], introduced four different asynchronous methods for Reinforcement Learning: one-step Q-Learning, one-step Sarsa, n-step Q-Learning and advantage actor-critic [9]. Advantage actor-critic was the algorithm that performed best overall.

In an early stage of combining Reinforcement Learning with neural networks, TD-Gammon [10] showed that an agent could learn how to play the board game Backgammon by playing against itself and learning from the results it was obtaining while playing. AlphaGo, a computer program that plays the game of Go, defeated the European Go champion by 5 games to 0 on a full-sized 19x19 board, becoming the first computer program to defeat a human professional player in this game [11]. Later, AlphaGo Zero achieved superhuman performance, mastering Go without human knowledge [12]. DeepMind researchers have recently produced further groundbreaking results. For example, a research has taught digital creatures to learn how to navigate accross complex environments [13]. StarCraft II is a highly technical real-time strategy video game released in 2010. Blizzard Enterntainment, the company that developed the video game, and DeepMind have published a joint paper that presents the StarCraft II Learning Environment, a challenging environment for testing Deep Reinforcement Learning algorithms and architectures on this game [14]. Another work poses the task for an agent to push boxes onto red target squares to successfully complete a level [15]. The challenge for the agent is that some actions might lead to irreversible mistakes resulting in the level being impossible to complete. The agent uses imagination, which is a routine to choose not only one action, but entire plans consisting of several steps to ultimately select the one that has best expected reward.

“Inceptionism” [16] and “DeepDream” [17] proved that after training a Deep Neural Network with a high number of related images and adjusting the network parameters, each layer progressively contains higher-level features of the image, until reaching the output layer, that ultimately makes a decision on what the image shows. Therefore, it was proven that the first hidden layers of a Deep Neural Network trained under these circumnstances, contained low-order features, such as edges or corners. Then, the intermediate hidden layers contained information about simple shapes, such as doors or leaves and the last hidden layers put together this information to form complete figures such as buildings or trees.

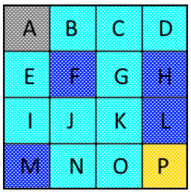
3 Methods

A structured set of methods will be followed to undertake this research. First, the problem to solve and the environment in which the agent will operate will be defined. The cornerstone of the research is the Reinforcement Learning algorithm that will teach the agent how to learn over time the specifics of the environment: the states, actions and reward values. Specifically, the Q-Learning technique will be implemented using a Deep Neural Network, a structure that will ultimately allow the feature extraction to occur. Features are going to be extracted from the initial level of the Deep Q-Network with the objective of improving the overall learning process of the agent and from the later level of the Deep Q-Network to predict which actions will lead to the next best states.

The Deep Reinforcement Learning Networks and the Q-Learning algorithm have been implemented using the programming language Python 3.5 and have been built using TensorFlow 1.2 [5], an open-source software library for Machine Intelligence. The project has been developed using the Python API of TensorFlow with CPU support.

3.1 Problem Statement and Environment

The agent must learn how to solve a pathfinding problem, which is to find the shortest route between two states. It will start in an initial state and its objective will be to learn over time how to arrive at a goal state ($+1$ reward) with the addition of learning how to avoid a set of states (”holes”, -1 reward) that are present in the environment. In order to represent this pathfinding problem, an environment composed by a set of states, actions and rewards in relation to the states is defined. The description of the environment used throughout the implementation is a 4x4 matrix composed by the letters ‘A’ to ‘P’ in alphabetical order, inspired by the Frozen Lake environment [18]. To complete the environment definition, the initial, goal and hole states are designated (Figure 1). Given this environment, a 16x4 matrix (R) that contains, every state, the next state resulting in taking all of the available actions (move up, down, right or left) is automatically generated.



A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Fig. 1: Environment. Initial state: ‘A’, goal state: ‘P’ and hole states: ‘F’, ‘H’, ‘L’, ‘M’.

3.2 Q-Learning

The Q-Learning algorithm [19] works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and consequently following the optimal policy thereafter. The agent will have a maximum of 2000 episodes to learn how to find the shortest path from the initial to the goal state, and in each of these episodes it will have 99 steps to move through the environment in search of the goal and to continuously adjust the values for the weights of the Deep Neural Network. An episode will end if the agent consumes all the steps or if it reaches the goal state. It starts by picking one action from the state it is in at the moment. This action will be either the best one as calculated until that moment or a random one. This decision is taken based on a ϵ-greedy policy ($\epsilon = 0.1$), which keeps a balance between exploration (taking random actions) and exploitation (taking best actions at that moment) in the discovery process of the environment. Once the action has been selected, using the R matrix, the next state that the agent will be in is extracted along with its related reward (neutral, positive or negative). Using the next state, the Q-values associated with that state are generated by feeding the state through the whole neural network. Therefore, it is possible to calculate the maximum Q-value that will be used in the fundamental Q-Learning formula, computed by the multiplication of the discount factor (γ) times the maximum Q-value plus the reward (Equation 1). Overall, for each state, the value that has been reinforced the most, corresponding to the best action, is chosen in order to determine the next state that the agent will be in. The algorithm will ultimately replace all the random initialized values of the weights and these will represent the minimum path from any state to the goal state.

$$Q(s,a) = r + \gamma(max(Q(s',a'))). \tag{1}$$

3.3 Deep Q-Network and Feature Extraction

The Deep Q-Network takes every state encoded in a unique 1x16 vector, and produces a vector of 4 values in its output layer, one for each action. In order to have a Deep Neural Network, it is necessary to include hidden layers between the input and output layers. In relation to the number of inputs and outputs and to have a representative number of hidden layers, a first hidden layer with 12 neurons was added, connected to a second hidden layer composed by 8 neurons, which is finally connected to the last hidden layer composed by 2 neurons ($2^2 = 4$, the number of total actions) (Figure 2).

The method of updating the values of the weights of the Deep Neural Network will be achieved by using backpropagation and a loss function. The loss function is defined as the sum-of-squares loss, where the difference between the output and the predicted output is computed. In this case, the target Q-value for the chosen action is the equivalent to the new Q-value computed in the Q-Learning algorithm. Finally, the agent is trained, using the target and predicted Q-values, with a gradient descent optimizer in order to minimize the loss. With a Deep Neural Network the information is propagated through the weights of the whole network.

The parameters used to configure the neural network are the following. Learning rate $(\alpha) = 0.05$, discount factor $(\gamma) = 0.99$, the hyperbolic tangent (tanh) activation function for hidden and output neurons, the $\sum((Y - Ypredicted)^2)$ loss function and the gradient descent optimizer. The weights of the Deep Q-Network are initialized randomly to normalize the variance of the output of each neuron to 1. This is achieved by scaling its weight vector by the square root of the number of inputs ($\sim \bigcup[-1/\sqrt{numStates}, 1/\sqrt{numStates}]$), where $\bigcup[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and numStates is the number of inputs of the algorithm [20]. The biases are initially set to 0, because the symmetry between hidden units of the same layer is broken by initializing the weights randomly in the indicated range.

Features can be extracted from the the neurons of the hidden layers of a Deep Neural Network trained with a Reinforcement Learning algorithm. A neuron is considered

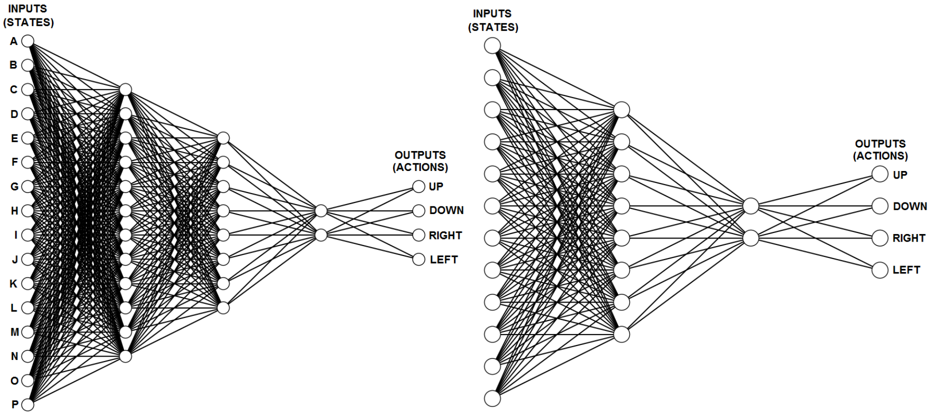


Fig. 2: The DQN with 12 input neurons (right) will be trained using as inputs features extracted from the first hidden layer of the DQN with 16 input neurons (left).

activated if its activation value is greater than 0, and not activated if its activation value is less than 0. More precisely, features are going to be extracted from the first hidden layer of the 16x12x8x2x4 Deep Q-Network, by having each available state encoded in a 1x12 vector, to be used as inputs for a different 12x8x2x4 Deep Q-Network (Figure 2). Features extracted from the last hidden layer of these two Deep Q-Networks, and encoded in 1x2 vectors, are going to be used to demonstrate that these codifications can be used to predict the best action to take in each state.

4 Results

This research was approached using a number of discrete steps. First, a testing phase was undertaken to demonstrate that the trained agent was capable of learning how to behave in the proposed environment. This was achieved by checking whether the agent reaches the goal state in an optimum number of steps from each state. Parallelly, the training time, the accumulated reward over time, the first episode in which the agent receives a reward equal to 1 and the average number of steps per episode were also obtained. These values define the performance of the agent in a specific training process and are used to make observations about the behaviour of the agent following the Q-Learning algorithm in the different Deep Q-Networks. Lastly, by using features extracted from the last hidden layer of a specific Deep Q-Network, we demonstrate that these can be used to predict the best actions to take from each state to reach the goal state. 30 different and independent experiments were run on a MSI GE63VR 7RE Raider laptop (Windows 10 Home 64 bits, Intel Core i7-7700HQ CPU @ 2.80GHz, 16.0GB RAM, GeForce GTX 1060).

4.1 Testing

The experiments demonstrated that the agent was able to find the shortest path from any state to the goal state without traversing through a hole. Because the Q-Learning algorithm does not end an episode if the agent enters a hole state, these states have related values that can lead the agent to the goal state. The agent has great difficulty to find the goal state from either states ’D’ and ’H’. This occurs because these two states are not on the optimal path to the goal state from any state and when exploring the environment, the agent usually does not reach states that are slightly further apart. The Deep Q-Networks that uses features extracted from the original Deep Q-Network as inputs shows marginally better results in finding the optimum path from each starting square (Figure 3).

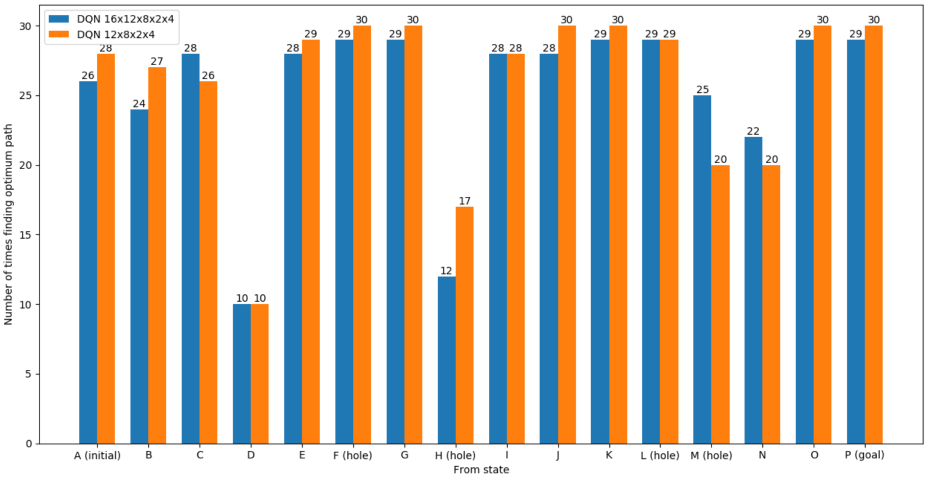


Fig. 3: Number of times that the agent finds an optimum path from each state of the environment to the goal state without entering a hole state.

4.2 Performance

The performance of the agent is tested on both Deep Q-Networks, with the objective of concluding in which one the agent performs the best. The results show that training an agent in the 16x12x8x2x4 Deep Q-Network takes an average training time of 66.73 seconds, with an accumulated reward of -0.09%, that it will receive its first reward equal to 1 in the episode 570, and that it takes 47.93 steps per episode on average. On the other hand, it can be seen that the training process of the agent on the 12x8x2x4 Deep Q-Network is considerably improved by using as inputs features extracted from the first hidden layer of the original Deep Q-Network. In this case, the average training time is 26.06 seconds, the accumulated reward over time is 0.63%, the episode in which it first receives a reward equal to 1 is 117 and that it only takes 16.36 steps per episode to reach the goal state for each episode on average. The performance of the agent using the 12x8x2x4 Deep Q-Network against using the original Deep Q-Network is 2.56 times faster in terms of training time, 8 times better regarding the accumulated reward over time, 4.87 times faster on finding the first episode with reward equal to 1, and 2.92 times better on the average number of steps it consumes per episode (Figure 4). Overall, the agent using 12x8x2x4 Deep Q-Network performs 4.58 times faster and better than using the 16x12x8x2x4 Deep Q-Network.

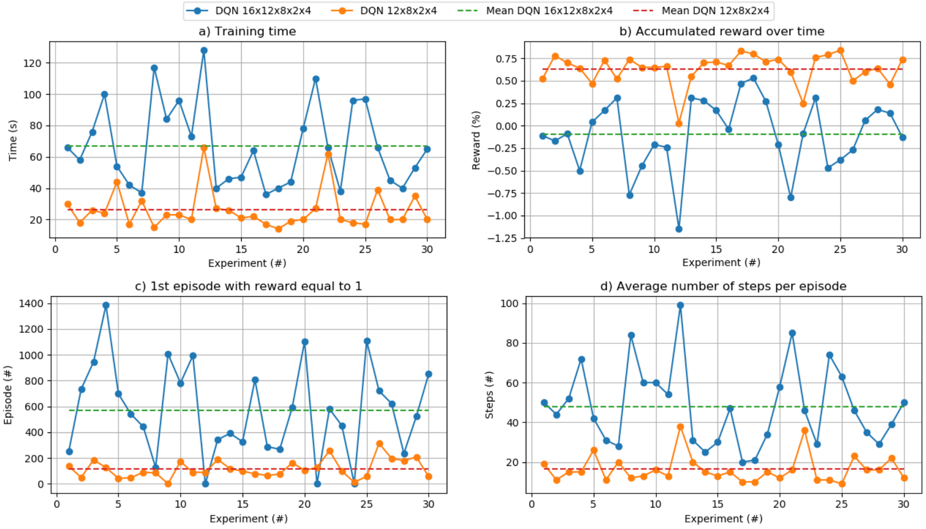


Fig. 4: Comparison of the perfomance of the agent between using 16x12x8x2x4 DQN and the 12x8x2x4 DQN trained on features extracted from the original DQN.

4.3 Prediction of Next States

After training both Deep Q-Networks (Figure 2) on every experiment, features are extracted from the last hidden layers using the activation of neurons. These features, encoded in 1x2 vectors, contain information regarding the best action to take from each state, so they can be used to predict the best state/action combinations without accessing the values contained in the output layers of both Deep Neural Networks. The 16x12x8x2x4 Deep Q-Network has been able to predict the best actions to take from each state 13.33% of times, while the 12x8x2x4 Deep Q-Network has achieved correct predictions 3.33% of times. This is due to several factors. For a prediction to be successful it is fundamental that the training process has firstly been succesful and that from every state, the agent is capable of reaching the goal state in an optimum number of steps (from Figure 3, at maximum 10 times, the number of times that optimum paths were effectively calculated from ‘D’). In addition, the activation threshold is established on 0 and it has been observed that this threshold is sometimes displaced by a factor of

-0.15/$+0.15$ from 0. Due to the training process, the activation values for the neurons are not always centered exactly on 0 because the hyperbolic tangent function is not a step function. When a prediction is correct, different groups of states will share the same codification indicating that all the states that belong to a group will take the same action in order to reach the goal state in an optimum set of steps. This demonstrates that the extracted features can be used to make predictions about the future states in which the agent will be in.

5 Conclusions

This work has presented an analysis of DRLNs focused primarily on the features that are automatically encoded in the hidden layers of this type of neural network. To achieve this, an environment to be discovered by an intelligent agent has been defined and the Q-Learning algorithm has been implemented using a Deep Q-Network. 30 experiments have been run in order to test if the agent is able to solve an established pathfinding problem within the defined environment and to compare the effectiveness of the different Deep Q-Networks. The research has produced two results: features extracted from the initial stage of a Deep Q-Network can be used in a different Deep Q-Network in order to improve the performance of an intelligent agent by a factor of 4.58 on average (Figure 4). And secondly, features extracted from the later stage of a Deep Q-Network contain information regarding the best action to take from a specific state. We have also shown that the information of the states is lost in later levels of the neural network, as the information regarding the states is transformed into decisions about which taken actions will lead to the best next states.

This research could be improved with the implementation of various extensions. First, to determine if the initialization of the weights of the Deep Q-Network and the activation functions and thresholds used are the optimal ones. Second, to implement a decaying ϵ-greedy policy, which would improve the way the agent discovers the environment. Third, to define a metric that would measure how long does it takes to learn the optimal policy. Fourth, to test the agent in different environments. Fifth, to obtain an optimal topology for a Deep Q-Network by consecutively extracting features from the first hidden layers of successive Deep Q-Networks. Lastly, to obtain a model built from a table of features and actions to features and rewards then to unseen states. As it stands, the model cannot work without the original environment, as it is only possible to extract rewards by relating them to the states.

References

1. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786) (2006) 504–507
2. Bellman, R.: A markovian decision process. Journal of Mathematics and Mechanics (1957) 679–684
3. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540) (2015) 529–533

4. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research (2012)
5. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
6. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI. (2016) 2094–2100
7. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. In: International Conference on Learning Representations (ICLR). (2016)
8. Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., de Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
9. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. (2016)
10. Tesauro, G.: Td-gammon, a self-teaching backgammon program, achieves master-level play. Neural computation **6**(2) (1994) 215–219
11. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587) (2016) 484–489
12. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676) (2017) 354
13. Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., et al.: Emergence of locomotion behaviours in rich environments. arXiv preprint arXiv:1707.02286 (2017)
14. Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A.S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al.: Starcraft ii: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 (2017)
15. Weber, T., Racanière, S., Reichert, D.P., Buesing, L., Guez, A., Rezende, D.J., Badia, A.P., Vinyals, O., Heess, N., Li, Y., et al.: Imagination-augmented agents for deep reinforcement learning. arXiv preprint arXiv:1707.06203 (2017)
16. Mordvintsev, A., Olah, C., Tyka, M.: Inceptionism: Going deeper into neural networks. Google Research Blog. Retrieved June **20** (2015) 14
17. Mordvintsev, A., Olah, C., Tyka, M.: Deepdream-a code example for visualizing neural networks. Google Res (2015)
18. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
19. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4) (1992) 279–292
20. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. (2010) 249–256