A Review of: Human-Level Control through deep Reinforcement Learning

Seminar Paper

Artificial Intelligence for Games

Author: Carsten Lüth

Supervisor: Associate Prof. Dr. Ullrich Köthe

Department of Computer Science

Faculty of Physics

University of Heidelberg

June 25, 2019

Contents

List of Figures

1 Introduction

In 2015 Deepmind presented the DQN agent [1] which was able to play ATARI2600 games on a human-level.
This represented a huge leap in performance of reinforcement learning algorithms since it outperformed many previous methods in these games while working with minimal distilled human knowledge of its environment (or the game).
It was also a big step for general artificial intelligence, since this algorithm learned to solve complex tasks and overcome obstacles without supervision or prior human knowledge.
The DQN agent learned to play the ATARI2600 games or interact with the environment based on the visual input, instead of human defined features, as most prior algorithms did. By only receiving the visual input it learned to extract useful information itself, this was achieved by using a Convolutional Neural Network. A Convolutional Neural Network uses layers of hierarchical convolutions to extract features, which is inspired by Hubel and Wiesel's seminar work on feed-forward processing in the early visual cortex [2].
The concepts of the DQN algorithem, as introduced by Mnih et al. [1], will be explored in this seminar paper, as well as its significance in the search for artificial general intelligence.

1.1 Artificial General Intelligence

Artificial general intelligence is the intelligence of a machine that could successfully perform any intellectual task that a human being can. So artificial general intelligence is closely connected to human intelligence or intelligence in general terms. To this day there is no definition of intelligence that satisfies everyone.
According to John Mc Carthy [3]:
"Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people."
Based on this quote intelligence is mainly used to achieve goals in a specific environment and there are various different ways in which intelligence can be expressed.
Currently there is a wide agreement in this field of research that intelligence is required for the following tasks:

1. reasoning, using strategy, making judgements under uncertainty
2. representing knowledge
3. planning

4. learning
5. communicating
6. integrating all of the above skills towards a goal

In many cases something is called intelligent when it expresses intelligent behaviour during interactions with its environment - by achieving its goals. So a machine with artificial general intelligence has to be able to sense its environment to interact with it in a meaningful way and thereby express its intelligence.

1.2 Reinforcement Learning

Reinforcement learning is an area of machine learning wants to answer the question, how software agents ought to take actions in an environment so as to maximize a pre-defined cumulative reward.
Often the goal is that the agent learns to solve tasks without prior instructions. This is achieved by using two disciplines [4]:

Dynamic Programming is a field of mathematics that has traditionally been used to solve problems of optimization and control.

Supervised Learning is a general method for training a parametrized function approximator to represent functions - such as a neural network.

Together these two methods allow the agent to learn tasks by trial and error through interactions with its environment. This trial and error leads a duality between exploitation where the agent uses already learned knowledge and exploration where the agent learns new things.
In the standard reinforcement learning model an agent interacts with its environment. This interaction takes the form of the agent sensing the environment s_t and based on this sensory input it chooses an action a_t to perform in the environment. The action then changes the environment resulting in a scalar reward r_t for the agent.
The environment is typically formulated as a Markov decision process (MDP). A MDP is a model to describe transitions between states s_t and s_{t+1}, given some action a_t in some environment. A base assumption when using a MDP is that the transition from one state s_t to the next state s_{t+1} is independent of earlier states ($s_{t'<t}$), also known as the Markov-Assumption. The main difference between the classical dynamic programming methods and reinforcement learning algorithms

is that the latter does not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

1.3 Video Games as an Environment for AI Research

The task of creating an agent to play video games in a similar fashion to humans based on sensory inputs is an active field of research. This is because video games are ideally suited for testing and observing agents.
Generally video games are designed to challenge the human mind by making players focus on gameplay-elements including, but not limited to: overcoming obstacles and following sub-stories. This follows a core idea of game design that players should be learning all the time, which is part of the reason many enjoy playing.
When teaching an AI agent to play a video game, the goal is not to develop AI that will create more interesting, dynamic, and realistic game experiences. Instead the virtual world of a video game is of key interest, since it is well suited for developing, training and testing AI agents. These virtual worlds can offer strict rules and even reward systems, making them useful to train and benchmark the intelligence of an agent. So the video game acts as a playground for different types of algorithms where they can test and try new things. By teaching different agents to play games, human researchers can hopefully understand how to train and design agents that are able to perform more complicated tasks in the future and apply them to real world applications. To quote Oriol Vinyals, co-lead on the Google-owned AI lab's StarCraft 2 project [5]: "First and foremost, the mission at DeepMind is to build an artificial general intelligence,...".
Another advantage of a virtual world is that the agent is able to learn in many of those worlds in parallel or by playing against itself, which allows faster accumulation of training data leading to faster training.

2 Methods

The fundamental idea behind the DQN is a Convolutional Neural Network which is used to compute the Q-Values for action state pairs. This section briefly introduces the concepts and methods used to achieve this.

2.1 Convolutional Neural Network

Convolutional neural networks (CNNs) are most commonly used in the field of computer vision for visual inspired tasks. Probably the first successful CNN architecture is the LeNet by Lecun et al. [6], which was initially used to classify images ranging from 0 to 9 of the MNIST dataset. As mentioned earlier, the design of a CNN with multiple hierarchical levels of convolutions represented by filters is inspired by the visual cortex [2]. They use the same knowledge (filters) to extract features from all parts of the image.
For classification these filters learn to extract the features that differentiate the different class to detect them as shown by Zeiler et al. [7]. A visualization of these features for a VGG-network [8] - a subtype of CNN - trained on ImageNet can be seen in figure 2.
From a mathematical standpoint a CNN is a general, fully differentiable function approximator. For classification tasks it typically consists of 4 main components or operations:

1. Convolution
2. Non-Linearity (ReLU)
3. Pooling or Sub-Sampling
4. Classification (Fully-connected-layer)

The structure of a convolutional neural network consists of several blocks with sub-sampling operations in between them and a fully connected network at the very end. Each of those blocks uses a convolution followed by a non-linearity. A typical setup can be seen in figure 1 .

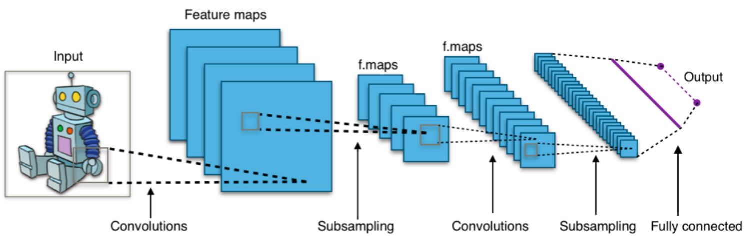


Figure 1: Typical Convolutional Neural Network architecture used for classification tasks where the input is used to create more and more abstract feature-maps via convolutions and non-linearities followed by a fully connected classifier. From: [9]

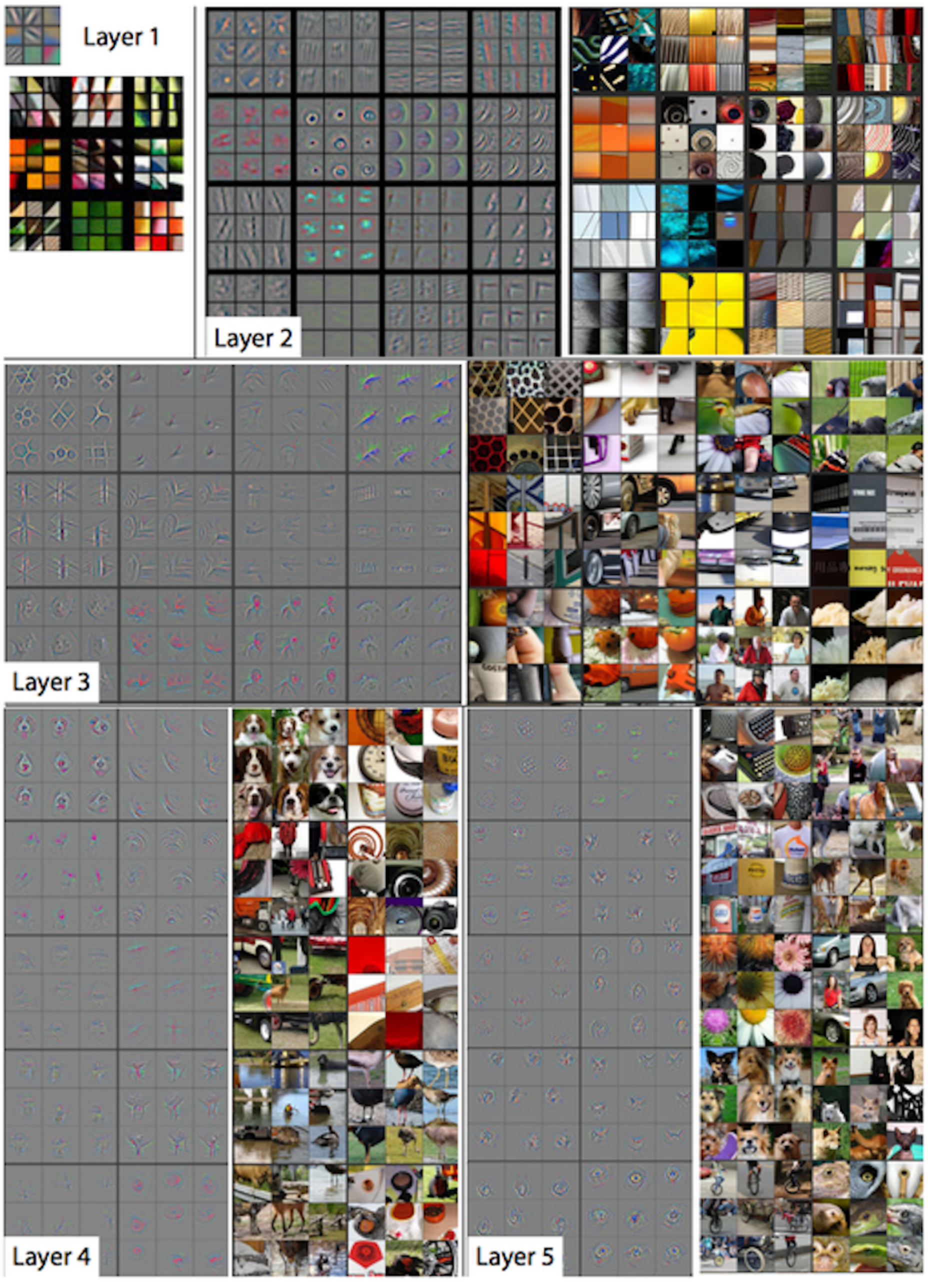


Figure 2: Visualization of the kernels (grey) and features (images) of a VGG network trained to classify the 1000 classes of the IMAGE-Net dataset. The complexity of the features the CNN extracts increases hierarchical for every layer. It becomes apparent especially for the higher level features, that the features are useful to detect humans, animals, cars etc. which are a subgroup of the 1000 classes. The network learns to detect useful features for the classification task by itself.
From: [7]

2.2 Q-Learning

The goal of Q-learning is to learn a policy, which tells an agent what is the most beneficial action to take under what circumstances.
This policy π is defined by the optimal-value-function $Q^*(s, a)$ as defined in equation 1.

$$Q^*(s,a) = \max_{\pi} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi] \tag{1}$$

The optimal-value-function returns when given state s and action a the highest expected reward following policy π. The highest expected reward uses a discount factor $\gamma \in [0, 1]$ with which feature rewards are weighted.
In other words $Q^*(s, a)$ returns the best possible reward at the end of the game after performing action a in state s. By following the policy of taking the action with the maximal Q-Value the agent maximizes the expected reward.
To learn this policy we need to approximate or learn the generally unknown Q^*. This is done by iteratively updating a predefined and known value-function Q. These updates with learning rate α are defined as follows 2

$$Q^{i+1}(s_t, a_t) \longleftarrow Q^i(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q^i(s_{t+1}, a_{t+1}) - Q^i(s_t, a_t)) \tag{2}$$

Q-learning is model free, hence it does not require a model of the environment. It can also handle problems with stochastic transitions and rewards, without requiring adaptions. This makes it a good approach when playing video-games with discrete actions.

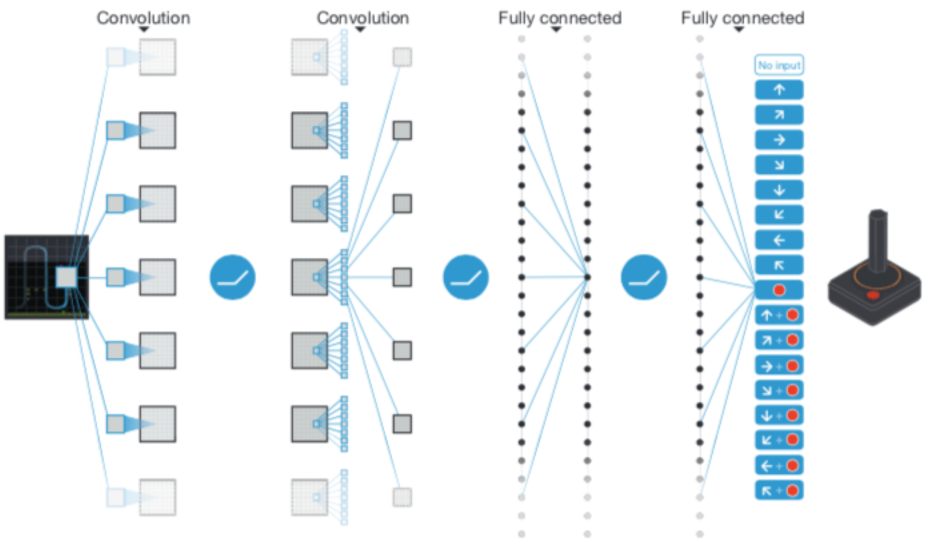


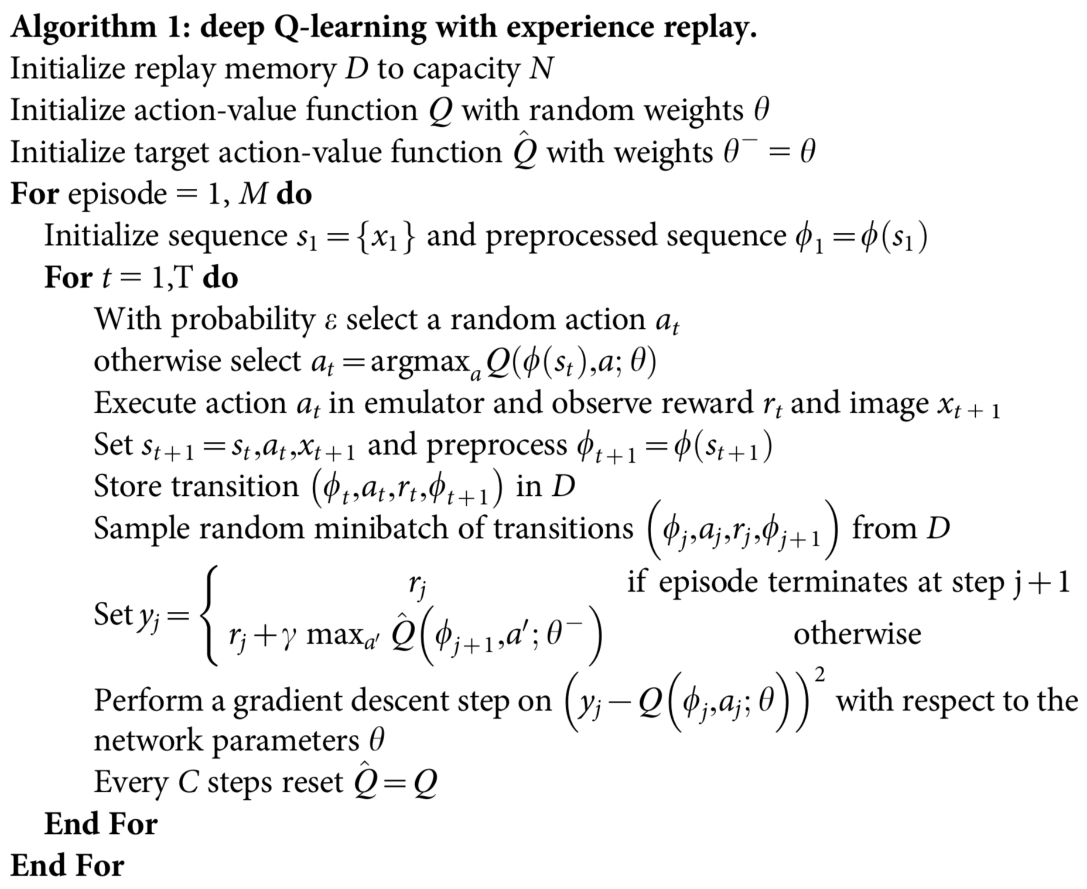
Figure 3: A conceptual visualization of the DQN. The input of the CNN is the state s_t consisting of a few consecutive frames of the video-game from which features are then extracted with the convolutional part of the network. These features are then used to compute the Q values for every possible action.
From: [1]

2.3 Deep Q-Network

The DQN uses a deep convolutional neural network to approximate the optimal-value-function Q^*. It receives the visual input as state s and outputs for every action a one Q-action value $Q(s, a; \theta)$, with θ being the parameters of the deep convolutional network. This setup can be seen in figure 3.
The algorithm to train the DQN can be seen in figure 4. The following methods were used to allow successful training of this network.

2.3.1 Designing State and Rewards with the Concept of Time

Time and Movement play an important role in video games. A conventional CNN has no a way to represent or learn the concept of time and movement.
To circumvent this a trick was implemented, instead of defining every state to be one frame, every state s_t consists of a number of consecutive frames. This allows movement to be captured in a similar fashion to a comic.
To lower the amount of computations needed for each forward pass of the network, the frames that the network receives are grayscaled.
The reward r_t for each state s_t is the increment in score of the game during the time in which the frames are collected.



```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, $M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t$ = 1,T do
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

Figure 4: The algorithm that is used to train the DQN. The different steps of this algorithms are explained in chapter 2.
From: [1]

2.3.2 Experience Replay

The core idea of experience replay is to enable the agent to learn from its past experiences.
The experience replay algorithm stores the experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ of the agent in a dataset $D_t = \{e_1, e_2, e_3, ..., e_t\}$. Then experiences are sampled by randomly drawing from the dataset

$$e = (s, a, r, s') \sim U(D)$$

and used to perform updates of the Q-Learning function.
Thus enabling the agent to learn from the same experience multiple times.
By randomly drawing the experiences the correlation from consecutive experiences is reduced, which stabilizes the training.
Another advantage of the dataset is the agent remembering past experiences through policy updates that are performed based on them.

2.3.3 Updating the Q-function

Due to the many parameters of the DQN agent, training can be unstable with the standard Q-learning update policy 2.
To counteract this an alternative updating policy was implemented, which is defined by gradient descent of the loss in eq. 3. It uses a target-value-function that gets periodically updated $Q(s, a; \theta^-)$ and a value-function that gets updated iteratively $Q(s, a; \theta)$.
Both these networks are identical in their architecture and all hyperparameters, the only difference are their parameters represented byθ.
The intuition for this training is that the value-function $Q(s, a; \theta)$ gets updated based on the outputs of the at least momentarily fixed target value-function $Q(s, a; \theta^-)$ rather than it's own.

$$\mathcal{L}(\theta_i) = E_{(s_t, a_t, r_t, s_{t+1}) \sim U(D)}[(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-) - Q(s_t, a_t; \theta_i))^2] \tag{3}$$

The parameters of the value function θ_i are updated every iteration by taking the gradient of equation 3 whereas the target-value function is updated every C iterations $\theta_C^- = \theta_C$.
This again makes the Training more stable and counteracts oscillations of the value-function.

2.3.4 Exploration vs. Exploitation with ϵ-Greedy

The problem of balance between exploiting the already known and exploring the unknown for reinforcement learning or learning in general is approached by using the ϵ-Greedy algorithm.
Exploiting the already known would be taking the action that maximized the Q-learning function w.r.t. the current state, whereas exploring would be taking a random action.
The ϵ-Greedy algorithm exploits with probability $1 - \epsilon$ and explores with probability ϵ with $\epsilon \in [0, 1]$, this can be seen in equation 4.

$$\begin{aligned} p(a|s) =& 1 - \epsilon + \frac{\epsilon}{k} \\ a = \operatorname{argmax}_a(Q(s, a)) & \\ =& \frac{\epsilon}{k} \\ else & \end{aligned} \tag{4}$$

3 Experiments

Setup The DQN algorithm is tested on the ATARI2600 games. For each game a seperate DQN agent was trained with the same network architecture and hyper-parameters, all agents only received the game score and pixels as input. The input frequency of the agents is set to 10Hz, thus lowering the reaction time of the agent to a human level. Each of those agents then plays the game after the training is converged to get a high-score.
The high-scores of the best performing methods from the reinforcement learning literature in 2015 ([10],[11]), were gathered for comparision with the DQN. As baselines the high-scores of a human video-game test and a player executing random actions are are taken as well. This procedure resulted in scores for 49 different ATARI2600 games.

Performance The high-scores of the agents were used to compute performance scores, which are defined in eq. 5.

$$\text{performance(agent)} = \frac{\text{agent\_score} - \text{randomplay\_score}}{\text{human\_score} - \text{randomplay\_score}} \cdot 100 \tag{5}$$

The performance is setting the scores of random actions 0% and the actions of a human game-tester 100% in a relation, making the agents comparable.
The resulting performance scores for the 49 different ATARI2600 games can be seen in figure 5.

Evaluation The DQN algorithm outperforms the best existing reinforcement learning methods in 43 of the games without incorporating any of the additional prior knowledge about Atari 2600 games used by the other approaches ([10] [11]). Also the performance of the DQN agents suggests that human level control ($> 75\%$) is reached in 29 of the games.
The games in which the DQN excels are very different in their nature and span different genres, from boxing games (Boxing) to side-scrolling shooter (River Raid) and car-racing games (Enduro). In Breakout, the algorithm learns the optimal strategy, where it digs a tunnel around the side of the wall to allow a pass behind the wall. There the ball is trapped for a while and destroys a large number of blocks. This can be seen in the following video: [12].
This strategy shows that the DQN is able to discover relatively long-term strategies given the correct circumstances.
The games with a heavy focus on long-term planning and strategies, such as Montezumas Revenge a Metroidvania, were generally a major challenge for the DQN just as for most existing algorithms.

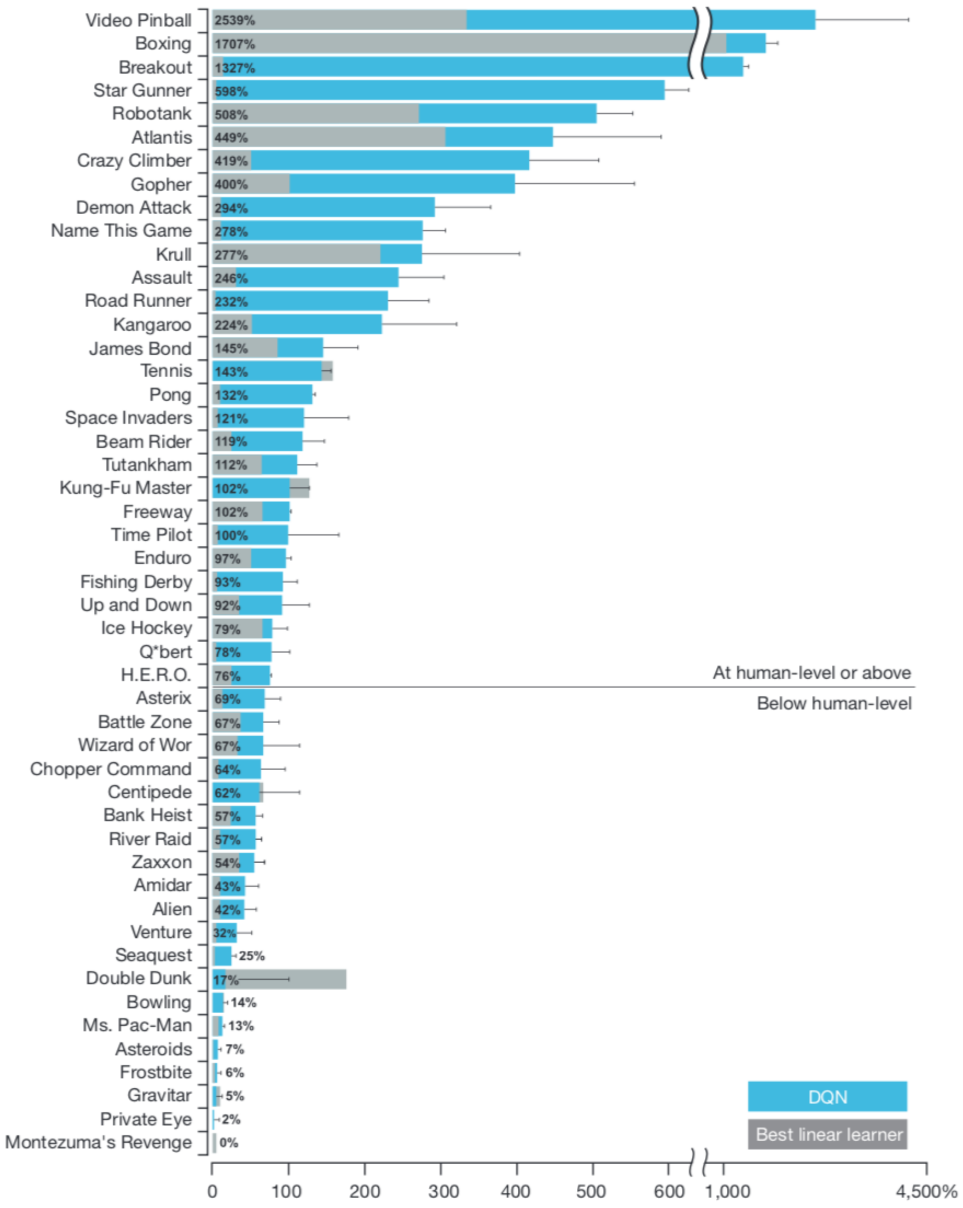


Figure 5: Comparison of the DQN agent with the reinforcement learning methods from gt15 gt12. The performance value is derived with equation 5.
From: [1]

4 Discussion

The performance of the DQN algorithm was a breakthrough for Reinforcement Learning at the time and many current Reinforcement Learning algorithms are based on it.
It is especially ground breaking since it was the first algorithm that performed on a human level on a variety of different video-games - with radically different tasks - without inducing human knowledge into the algorithm.
Meaning the algorithm learned itself to interact with its environment - the video game - to solve specific tasks with the same visual input and with the score a human player receives when playing or learning the game, making the processes comparable. So the algorithm had to learn useful features upon which to base its decisions leading to the most beneficial outcome.
Arguably this means that the DQN expresses intelligent behaviour through its interactions with many different gaming environments.
Meaning that the versatility of the DQN algorithm could make it a major contribution in the search for artificial general intelligence.
The basis of the DQN algorithm which uses the convolutional neural network allows it to be used in very complex tasks. The biggest problems of the DQN are its slow learning and its problems with long-term planning and strategy.
These problems have been worked on extensively in the last few years with different additions to the general DQN algorithm [13] with faster learning and better performance. Also different algorithms with a convolutional neural network for feature extraction as an approximated value function have been proposed with great results in the field of video-games such as DOTA 2 with Proximal Policy Optimization [14] and Starcraft 2.
The main contribution of the authors [1] was to prove the ability of CNNs being used effectively on large-scale reinforcement learning problems, more exactly a wide variety of ATARI2600 video-games.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Nature **518**, 529 EP (2015).

[2] D. Hubel and T. Wiesel, J. Physiol. (Lond.) **165**, 559 (1962).

[3] J. McCarthy, "The bitter lesson," (2007), `http://www-formal.stanford.edu/jmc/whatisai/node1.html`, Sidst set 25.6.2019.

[4] M. E. Harmon and S. S. Harmon, "Reinforcement learning: A tutorial," (1996).

[5] Verge, "The verge games," (2018), `https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning`, Sidst set 25.6.2019.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Proceedings of the IEEE **86**, 2278 (1998).

[7] M. D. Zeiler and R. Fergus, CoRR **abs/1311.2901** (2013), arXiv:1311.2901 .

[8] K. Simonyan and A. Zisserman, in *International Conference on Learning Representations* (2015).

[9] Wikipedia, "Cnn visualization," `https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png`, Sidst set 25.6.2019.

[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, CoRR **abs/1207.4708** (2012), arXiv:1207.4708 .

[11] M. Bellemare, J. Veness, and M. Bowling, "Investigating contingency awareness using atari 2600 games," (2012).

[12] Deepmind, "Dqn playing breakthrough," (2016), `https://www.youtube.com/watch?v=TmPfTpjtdgg`, Sidst set 25.6.2019.

[13] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, CoRR **abs/1710.02298** (2017), arXiv:1710.02298 .

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, CoRR **abs/1707.06347** (2017), arXiv:1707.06347 .