Using artificial neural networks in reinforcement learning algorithms

Martin Glova, Gabriela Andrejková

Institute of computer science, Faculty of Science
P. J. Šafárik University in Košice
Jesenná 5, 04001 Košice, Slovakia
gabriela.andrejkova@upjs.sk
martin.glova@upjs.sk

Abstract: Reinforcement learning algorithms represent a specific category in the machine learning field precisely because of their unique approach based on a trial-error basis. We introduce a new approach into the Q-algorithm, namely maximizing k-future rewards policy, which decreases learning time and increases maximal and average score value of an optimizing function significantly. Our modified Deep NNQ-learning using feed-forward networks instead of originally proposed convolutional neural networks gives the best results in a tested problem. We implemented the developed algorithm for the Flappy Bird game where significant improvements are achieved by appropriate setting of state space, act policy, and by pretraining neural networks.

Keywords: Artificial intelligence, Reinforcement learning, Q-learning, Deep Q-learning, Flappy Bird game

1 Introduction

Reinforcement learning is a learning method that determines what action will maximize the agent's reward in a state derived from an environment. Introduction to the theory of the reinforcement learning can be found in the book Sutton & Barto [24]. The goal of reinforcement learning algorithms is to build a policy by learning from past good action sequences [2]. For this class of algorithms, two basic properties are significant: *trial and error method* and *possible delayed reward*.

In the developed algorithm, we focus mainly on using of neural networks in a reinforcement learning approach as far as it appears to be a method with a potential to achieve better results. The main reason that neural networks have been introduced in the reinforcement learning is that "representation learning with deep learning enables automatic feature engineering and end-to-end learning through updating weights of neural networks so that reliance on domain knowledge is significantly reduced or even removed" [15]. According to [15] deep learning and reinforcement learning will play an important role in achieving artificial intelligence and the overview describes main principles of both learnings.

In this paper, we describe an algorithm based on a learning algorithm that could learn to play computer games and solve similarly defined problems. Games provide excellent test beds for similar algorithms [15]. Our introduced algorithms could be later used in more real-world applications such as a natural language processing [6, 14, 23, 30], computer vision [19, 29, 31], business management (recommendation, customer management, marketing, etc.) [5, 9, 13, 25, 28], robot navigating [15] in an environment or drones navigation, etc. An autonomous drone has similar problems as are described in this paper. Especially when we try to cope with indoor navigation with lots of obstacles or navigation in caves, etc. More real-world examples of reinforcement learning applications can be found in the overview [10].

The paper is organized as follows: Section 2 contains a description and comparison of related algorithms. Section 3 describes our new algorithms. In Section 4, there are the results and observations of the tested algorithms on the Flappy Bird game. The section 5 contains a summary of our contribution to the area.

2 Reinforcement Learning Algorithms

An overview of deep reinforcement learning algorithms can be found for example in [7]. In the following subsections we describe reinforcement learning algorithms which we modified, tested and compared to our results.

2.1 Q-learning

Q-learning is an algorithm which uses *Q-values*. These values are represented as a function $Q : S \times A \to \mathrm{R}$. Input of the function Q is a couple $\langle s, a \rangle$, where $s \in S$ (S is a state space), $a \in A$ (A is an action space) and R is the set of real numbers. As output of the function Q we expect a future possible reward - a Q-value. As far as the sets S and A are finite the Q function is a discrete function. The pseudocode of the Q-learning algorithm is given in the Algorithm 1 prepared according to [22].

The algorithm requires to define a state space S, an action space A and parameters like a learning rate η and a discount factor γ. The *learning rate* $\eta \in [0, 1]$ means how much of the old Q-value we take into account. The *discount factor* $\gamma \in [0, 1]$ means how much of a next possible

reward we take into account. The key part of the algorithm is the equation (1) which shows how the Q-function is updated. Q is initialized by random values [11]. If all actions from A are chosen in all states infinitely many times, so if the algorithm is executed infinitely many times and parameter η and γ are set properly, then Q-values converge to the optimal values with the probability 1 [17, 27].

Algorithm 1: The pseudocode of the $Q-$learning algorithm.

Initialize all $Q(s,a)$ arbitrarily.
for *all episodes* **do**
  Initialize $s_t \in S$, where $t = 0 \in T$.
  while s_t *is not the terminal state* **do**
    Choose $a \in A$ using policy derived from Q for the state s_t (e.g. the ε-greedy policy).
    Take action a, observe r and a new state s_{t+1}.
    Update $Q(s_t,a)$ by using equations

$$Q(s_t,a) = Q(s_t,a) + \delta \qquad (1)$$

$$\delta = \eta(r + \gamma \max_{a' \in A} Q(s_{t+1},a') - Q(s_t,a)).$$

    Update $t = t+1$.
  end
end

The algorithm *SARSA* (State-Action-Reward-State-Action) is very similar to the Q-learning Algorithm 1. Actually, in the beginning, it was called *modified Q-learning* by its inventors Rummery and Niranjan [24]. The only difference in the pseudocode is in the equation (1). It is changed to the following equation (2):

$$Q(s_t,a) = Q(s_t,a) + \eta(r + \gamma Q(s_{t+1},a') - Q(s_t,a)), \quad (2)$$

where a' is chosen by the same policy as a is chosen in the state s_t (for example by the ε-greedy policy). So we do not maximize future possible reward in the equation but always use the same policy to choose the best action [22, 24]. That is why we also say that SARSA is *on-policy* algorithm and Q-learning is *off-policy* algorithm.

For given policy, the value of Q-function can be computed by Bellman operator which is a contraction with unique fixed point [8] under some conditions on Q-function.

2.2 Deep Q-learning

The algorithm *Deep Q-learning* (DQN) introduced by DeepMind Technologies Company is described in Algorithm 2 and it is highly based on the algorithm from Riedmiller [21] which uses multi-layer perceptrons.

At first, it initializes a replays memory (similar principle as it is "experience replay" in [16]). Such memory is used to store training data. The first training data can be created in some reasonable way, for example as a random playing of the game. The agent should balance between failure data and successful data if he wants to learn faster. Picking of an action in the algorithm is done the same way as it is for Q-learning or SARSA. Only updates are done by performing a gradient step and changing weights of the network. In the algorithm $\gamma \in [0,1]$ is a discount factor and has the same meaning as before in the SARSA and Q-learning algorithms. The original algorithm was introduced in a combination with convolutional neural networks [18, 20].

Algorithm 2: The pseudocode of the algorithm Deep Q-learning.

Initialize replay memory D to capacity n.
Initialize the function Q with random weights.
Remark. Question mark is used for indices of minibatch transitions.
for *all episodes* **do**
  Initialize $s_t \in S$, where $t = 0 \in T$.
  while s_t *is not the terminal state* **do**
    Choose $a \in A$ using policy derived from Q for the state s_t (e.g. the ε-greedy policy).
    Take action a, an observation r and a new state s_{t+1}.
    Store $\langle s_t, a, r, s_{t+1} \rangle$ in D.
    Sample a minibatch of transitions $\langle s_?, a_?, r_?, s'_? \rangle$ from D.
    if $s'_?$ *is the terminal state* **then**
      Set $y = r_?$.
    else
      Set $y = r_? + \gamma \max_{a' \in A} Q(s'_?, a', \theta)$.
    end
    Perform a gradient step on $(y - Q(s_?, a_?, \theta))^2$.
    Update $t = t+1$.
  end
end

3 Modifications of the Algorithms

The following subsections describe a new developed policy: how to choose a new action in the Q-learning and also modified version of the Deep Q-learning algorithm.

3.1 Maximizing k-Future Rewards Policy

The *policy* for selecting actions is one of the key factors of reinforcement learning algorithms.

It is known [18] that the reinforcement learning algorithm should be unstable or to diverge when nonlinear function approximator is applied as the Q-function. But approximations give quite interesting results in many experimental cases.

The classical description of the Q-learning algorithm and also SARSA only takes one next step into account. The main idea of our new approach is not only to maximize the Q-values of the actual state for all the actions but sum up Q-values of all possibilities in a depth k ($k > 0$) and pick up the best action with the greatest summed value (in many games a player analyzes more steps forwards). The complexity of picking a new action grows exponentially with respect to the number of future steps taken into account. If we take k future steps it is $\mathcal{O}(|A|^k)$, where A is the set of all possible actions in the action space.

3.2 Modified Deep Q-learning

In the first step, we need to decide, what type of neural networks will used in the implementation of the Deep Q-learning Algorithm 2. Authors [3, 4, 20] usually use convolutional neural networks and use raw pixel information from the game screen to define the state space. We decided to use feed-forward neural networks (FNN) and we use an ensembling of FNNs in a pre-training of the developed algorithm. Pre-training was done on real data and then we use the best FNN to train. The pseudocode of the modified Deep NNQ-learning algorithm is in Algorithm 3.

The input for FNN is defined as a couple $\langle s, a \rangle$, where $s \in S$ and $a \in A$ and the output is defined as a real value (a future possible reward). An advantage of the approach is that such network is trained as one system. For example, let us have a frequent occurrence of the state $s_i = \langle s_{i1}, \ldots, s_{in} \rangle \in S$, so our algorithm is well trained for the state s_i. Then let us also have a very rare occurrence of the state $s_j = \langle s_{j1}, \ldots, s_{jn} \rangle \in S$, $s_j \neq s_i$, but $s_j = \langle s_{i1} + \varepsilon_1, \ldots, s_{in} + \varepsilon_n \rangle$ and ε_k is very close to zero for k, where $1 \leq k \leq n$. In other words, states s_i and s_j are very similar but no the same. In the previous approach using Algorithm 1 with equation (1) or (2), the agent should act very well for the state s_i but not for the state s_j, which is almost the same state, as far as training is realized by the discrete Q function.

Algorithm 3: The pseudocode of the modified Deep NNQ-learning algorithm. The function Q in the algorithm represents a neural network.

```
Obtain Q-values using Algorithm 1 using policy
  described in Subsection 3.1.
Initialize the function Q with random weights.
Pre-train the function Q using obtained Q-values
  from the first step.
for all episodes do
    Initialize s_t ∈ S, where t = 0 ∈ T.
    while s_t is not the terminal state do
        Choose a ∈ A using policy derived from Q
          for the state s_t (e.g. the ε-greedy policy).
        Take action a, observe r and a new state
          s_{t+1}.
        if s_{t+1} is the terminal state then
            Set y = r.
        else
            Set y = r + γ max_{a'∈A} Q(s_{t+1}, a', θ).
        end
        Perform a gradient step on
          (y − Q(s_t, a, θ))^2.
        Update t = t + 1.
    end
end
```

4 Results

4.1 Flappy Bird Game

In our research, we compare the results of the algorithms applied in the *Flappy Bird* game. The game is a side-scroller where the player controls a bird, attempting to fly between columns of braun pipes without hitting them. The score is evaluated using of the number not hitting columns. Fig. 1 shows an example of the game configuration. The player can see up to two of the following pipes. The bird moves forward (along the horizontal axis) with a constant velocity, but his movement along the vertical axix is more complicated and it is possible to influence it by the player's control. If the bird is not controlled by the player, then its vertical velocity (hereinafter referred to as only velocity) decreases in each step by 1 downwards until his velocity is 10 downwards.

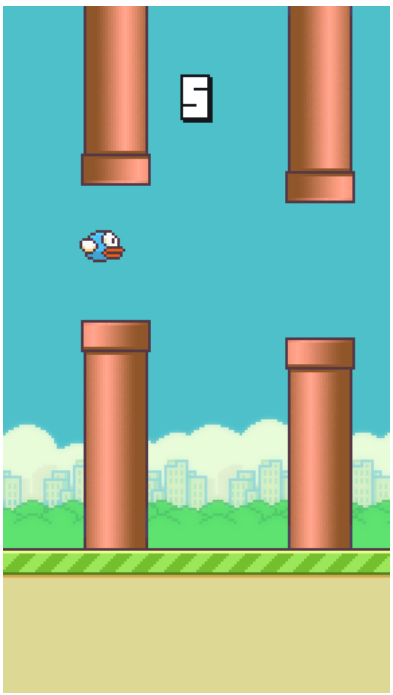

Figure 1: An example of the Flappy Bird game configuration.

Definition 1. *Let $X \subseteq \mathrm{R}$ be the set of all the different distances of the leftmost pixels of the bird to the rightmost pixels of the pipe nearest to the bird, $Y \subseteq \mathrm{R}$ be the set of the different distances of the bottommost pixels of the bird to the topmost pixels of the next bottom pipe, $V \subseteq \mathrm{Z}$ is the bird's velocity where the negative values mean upwards direction and the positive values mean downwards direction. Then the set $S = \{\langle x,y,v\rangle : x \in X, y \in Y, v \in V\}$ is called state space.*

We define the action space as a set of two actions $A = \{FLAP, NOT\_FLAP\}$ and the state space by Definition 1 to use them in reinforcement learning algorithms. In our implementation, we work with the rounding function $rnd : \mathrm{R} \times \mathrm{N}^+ \to \mathrm{Z}$, $x \in \mathrm{R}, \mathrm{r} \in \mathrm{N}^+$ defined by (3).

$$rnd(x,r) = r * \lfloor x/r \rceil. \tag{3}$$

The cardinality of S could be very high and we reduce some states using a states conversion function defined in Definition 2, formula (4).

Definition 2. *Let $X \subseteq \mathrm{R}$ be the set of all the different distances of the leftmost pixels of the bird to the rightmost pixels of the pipe nearest to the bird, $Y \subseteq \mathrm{R}$ be the set of the different distances of the bottommost pixels of the bird to the topmost pixels of the next bottom pipe, $R_X, R_Y \subseteq \mathrm{N}^+$ be the sets of all the rounding values for the horizontal/vertical distances, $V \subseteq \mathrm{Z}$ is the bird's velocity where the negative values mean upwards direction and the positive values mean downwards direction and $S = \{\langle x,y,v\rangle : x \in X_S, y \in Y_S, v \in V\}$ is **state space**, where $X_S = X \cap M_{r_X} \cup \{rnd(\min(X), r_X), rnd(\max(X), r_X)\}$ and $Y_S = Y \cap M_{r_Y} \cup \{rnd(\min(Y), r_Y), rnd(\max(Y), r_Y)\}$, where M_{r_Z} is the set of all multiples of $r_Z \in R_Z$ for $Z \in \{X,Y\}$ and rnd is the rounding function defined by (3). The function $scf : X \times Y \times R_X \times R_Y \times V \to S$ is called the states conversion function and if $x \in X$, $y \in Y$, $r_X \in R_X$, $r_Y \in R_Y$, $v \in V$ and rnd is the rounding function (3), then*

$$scf(x,y,r_X,r_Y,v) = \langle rnd(x,r_X), rnd(y,r_Y), v\rangle. \tag{4}$$

4.2 Simple and Advanced Greedy Algorithms

We tested the simple greedy algorithm described in Algorithm 4. This algorithm does not use any learning and only uses the following simple rule: flap anytime vertical distance of the bird to the next bottom pipe could be in the next state less than zero (so the agent could hit the pipe).

The average score of the algorithm for 5000 games is 143.43 and with individual scores of games (the red dots) are shown in Fig. 2. The algorithm is implemented mostly for comparison with our developed algorithm and in the future it can be used as a part of other algorithms.

So the advanced greedy policy does not change the average score signficantly but still is better than Simple Greedy Algorithm. The pipes positions in played games were the same as it was for the simple greedy algorithm.

Algorithm 4: The pseudocode of the simple greedy algorithm, where y is the vertical distance of the bird to the next bottom pipe.

```
if y < 10 then
    return FLAP
else
    return NOTFLAP
end
```

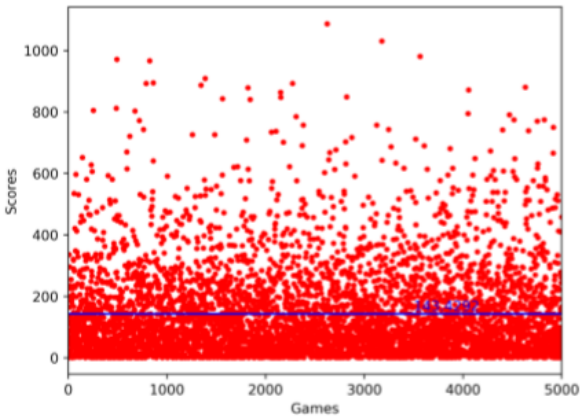

Figure 2: Scores of 5000 games (the red dots) with the highlighted average score 143.43 (the blue line) played in the sequence by Algorithm 4. Higher scores are not so frequent and they are interleaved by many smaller scores.

The advanced greedy algorithm does not use any learning and flaps only if it is necessary - so only if in the next 20 states the bird would hit the ground of the bottom pipe.

The number 20 is chosen because it is the minimal time until the bird is in the same vertical position as it was before it flaps the last time. Or in other words: 20 is the number of the states which are affected by flapping. The last restriction is that the bird would flap only if it does not cause hitting the pipe in any of the twenty next steps. The average score of the algorithm for 5000 games is 144.36 as we can see it in Fig. 3.

4.3 Q-learning and ε-greedy policy

We focused on searching of the optimal parameters for the Q-learning algorithm described in Algorithm 1.

The policy depends on the ε value or better say whether ε-greedy policy is used or not. If ε is not used then it means there is no signficant difference between SARSA and Q-learning as far as during update of Q-value SARSA uses ε-greedy policy and Q-learning maximize the values (we are now comparing the update Equations (1) and (2)). Setting ε to some non-zero values is a mechanism to force the algorithms to be able to converge to optimal Q-values. Otherwise, it would be impossible to prove such statement.

Algorithm 5: The pseudocode of the advanced greedy algorithm.

```
have_to_flap = FALSE
for state ∈ next 20 states do
    x = getX(state)
    y = getYWithoutFlap(state)
    if y ≤ GROUND_BASE or
      (x ≤ PIPE_WIDTH + BIRD_WIDTH and
      y ≤ 0) then
        have_to_flap = true
    end
end
if have_to_flap then
    for state ∈ next 20 states do
        x = getX(state)
        y = getYWithoutFlap(state)
        if x ≤ PIPE_WIDTH + BIRD_WIDTH
          and y ≥ VERTICAL_GAP_SIZE −
          BIRD_HEIGHT then
            return NOTFLAP
        end
    end
    return FLAP
end
return NOTFLAP
```

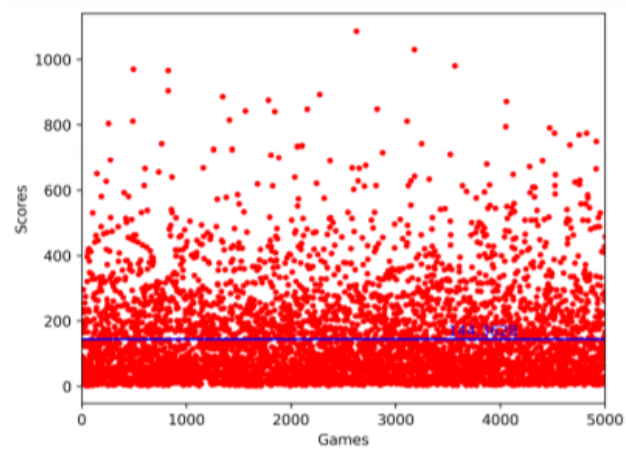

Figure 3: Scores of 5000 games (the red dots) with the highlighted average score 144.36 (the blue line) played in the sequence by Algorithm 5.

This parameter ensures that we are also likely to choose other previously unselected actions to also test whether the future reward is not higher by choosing such action. Fig. 3 4 shows that there is not a big difference in using or not using ε-greedy policy. But using ε-greedy policy generates slightly better results. The value ε was changed by the formula $\varepsilon_k = 1/k$, where k is the number of the iteration.

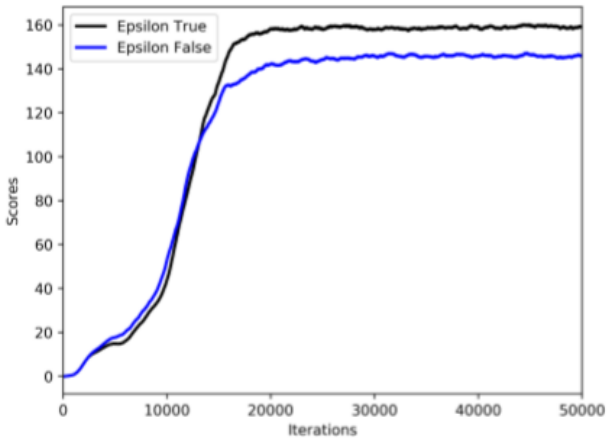

Figure 4: Using and not using ε-greedy policy.

4.4 Maximizing k-Future Rewards Policy

Based on analysis of all parameters we set the parameters to achieve as good results as possible. In all of the simulations in Fig. 5 learning rate is set to 0.7, discount factor is set to 1.0, the reward $r = 1$ for alive states and $r = -1000$ for the three last states, the rounding values are set as follows $r_X = r_Y = 5$ with $n = 1$ with $n = 1$ and the ε policy is set to true. The colored trending lines represent average scores of the last 1,000 games with the parameter k appertains to the values in the legend on the left side of the figure.

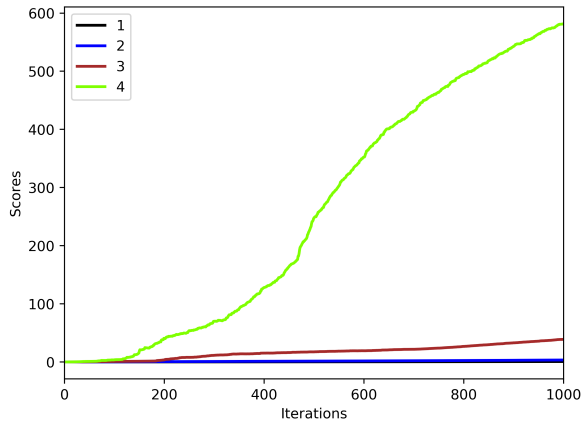

Figure 5: Changing the number of steps taken into account when selecting a new action (for 1,000 iterations and averaged the results of 25 random simulations). {1, 2, 3, 4} represents the number of the future rewards.

4.5 Deep NNQ-learning Algorithm

In the experiments discussed in this subsection we use *feed-forward neural network* with *2 hidden layers* with *600 and 200 neurons*. These values seemed to be the best in our case. We also tested more and fewer layers but the best results are achieved by using only two layers (we also tried 3 to 5 layers with more neurons, like 1000, or less, like 100). The input for the network is a triple consisting of three real values: horizontal and vertical distance of the bird to the next pipe and the velocity. The output of the network is an ordered pair of two real values from interval $[-1,1]$ meaning future reward by choosing flap or not flap action. Three different activation functions were tested: *sigmoid*, *hyperbolic tangent* and *relu*. In the given evaluation we use *tanh*. The network is initialized with random weights and these are updated by using the Adam algorithm [12]. Learning rate is set to 0.01.

We initialize the network by using Q-values generated by Q-learning as described in Algorithm 3. We map for each state s decisions to 1, if the action would be taken or -1, if the action would not be taken by the Q-learning algorithm. So we have for each state $s \in S$ and both actions a_1 and a_2 training data in format $Q(s,a_1) = 1$ and $Q(s,a_2) = -1$ or $Q(s,a_1) = -1$ and $Q(s,a_2) = 1$. When training the model during initialization we use more optimizers like gradient descent, AdaGrad, RMSProp or Adam and train more separated networks. As far as Adam trains the best models we use them and decisions are made by all of them by summing predictions of all models and then picking the action to take by maximizing the values.

Rewards are then generated in a similar way as during initialization - in state s the reward in 1, if the action was taken or -1, if the action was not taken. Only the last 30 states are ignored because we presume that some of them caused the death of the bird. Training is then done only by using new data and discount factor is not used in this case. In Fig. 6 one can see the comparison of two best-implemented instances of Q-learning and Deep NNQ-learning tested on the same 100 games where each game has at maximum 50,000 pipes generated (then the game stops). The *average score* is approximately *36,429.26 for Deep Q-learning* and *2,771.59 for Q-learning*. So the modified Deep NNQ-learning plays significantly better then Q-learning using maximizing k-future rewards policy.

An open-source `Python` implementation of the game is used in the paper [26]. Neural networks are implemented using the library `TensorFlow` [1]. All the source codes used to create the results of this paper are available in the public GitHub repository `https://github.com/martinglova/FlappyBirdBotsAI`.

5 Conclusion

The results described in the paper show some properties of Q-learning and the role of neural networks in reinforcement learning through the Deep Q-learning algorithm. The experiments confirm some hypotheses about the setting parameters of the algorithms like learning rate, discount factor, rewards, the number of penalized states, etc.

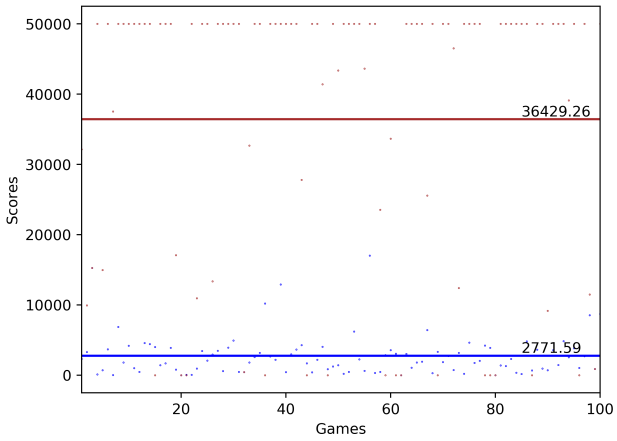

Figure 6: The comparison of the Deep NNQ-learning algorithm (brown color) and Q-learning (blue color) results.

The introduced algorithms tested in the Flappy Bird game show that extracting features from images and using feed-forward neural network instead convolutional can lead to significant results too. The same approach has potential in more real-world applications where raw pixels with lots of noise are used as an input.

In Fig. 5 one can see how introducing of the maximizing k-future rewards policy improve the original Q-learning algorithm and in Fig. 6 one can see that the improvement can be even better using the Deep NNQ-learning algorithm.

Acknowledgements. The research is supported by the Slovak Scientific Grant Agency VEGA, Grant No. 1/0056/18.

References

[1] TensorFlow Library, `https://www.tensorflow.org/`

[2] Alpaydin, E.: Introduction to machine learning. MIT Press, Cambridge (2014)

[3] Appiah, N., Vare, S.: Playing flappybird with deep reinforcement learning (2016), `http://cs231n.stanford.edu/reports/2016/pdfs/111_Report.pdf`

[4] Chen, K.: Deep reinforcement learning for flappy bird (2015), `http://cs229.stanford.edu/proj2015/362_report.pdf`

[5] Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q.: Deep direct reinforcement learning for financial signal representation and trading. IEEE transactions on neural networks and learning systems **28**(3), 653–664 (2017)

[6] Dhingra, B., Li, L., Li, X., et al.: End-to-end reinforcement learning of dialogue agents for information access. arXiv preprint arXiv:1609.00777 (2016)

[7] Fenjiro, Y., Benbrahim, H.: Deep reinforcement learning overview of the state of the art. Journal of automation, mobile robotics & Intelligent Systems **12**(3), 20–38 (2018)

[8] Fujimoto, S., Meger, D., Precup, D.: Off–policy deep reinforcement learning without exploration. In: 36th International Conference on Machine Learning, PMLR 97. pp. 2052–2062 (2019)

[9] Goldberg, Y., Kosorok, M.R.: Q-learning with censored data. Annals of statistics **40**(1), 529 (2012)

[10] Hammond, M.: Deep reinforcement learning in the enterprise: Bridging the gap from games to industry (2017), `https://conferences.oreilly.com/artificial-intelligence/ai-ca-2017/public/schedule/detail/60500`, aI Conference, San Francisco

[11] Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of artificial intelligence research **4**, 237–285 (1996)

[12] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[13] Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th international conference on World wide web. pp. 661–670. ACM (2010)

[14] Li, X., Chen, Y.N., Li, L., Gao, J.: End-to-end task-completion neural dialogue systems. arXiv preprint arXiv:1703.01008 (2017)

[15] Li, Y.: Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274 (2017)

[16] Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning **8**(3-4), 293–321 (1992)

[17] Melo, F.S.: Convergence of q-learning: A simple proof. Institute Of Systems and Robotics, Tech. Rep pp. 1–4 (2001)

[18] Mnih, V., Heess, N., Graves, A., et al.: Human–level control through deep reinforcement learning. Nature **518**, 529–533 (2015)

[19] Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Recurrent models of visual attention. In: Advances in neural information processing systems. pp. 2204–2212 (2014)

[20] Mnih, V., et al.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)

[21] Riedmiller, M.: Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In: European Conference on Machine Learning, LNAI 3720. pp. 317–328. Springer (2005)

[22] Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems, vol. 37. University of Cambridge, Department of Engineering (1994)

[23] Su, P.H., Gasic, M., Mrksic, N., Rojas-Barahona, L., Ultes, S., Vandyke, D., Wen, T.H., Young, S.: On-line active reward learning for policy optimisation in spoken dialogue systems. arXiv preprint arXiv:1605.07669 (2016)

[24] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, London (1998)

[25] Theocharous, G., Thomas, P.S., Ghavamzadeh, M.: Personalized ad recommendation systems for life-time value optimization with guarantees. In: IJCAI. pp. 1806–1812 (2015)

[26] Verma, S.: Flappy bird. `https://github.com/sourabhv/FlapPyBird` (2017)

[27] Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**, 279–292 (1992)

[28] Wen, Z., O'Neill, D., Maei, H.: Optimal demand response using device-based reinforcement learning. IEEE Transactions on Smart Grid **6**(5), 2312–2324 (2015)

[29] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. In: International Conference on ML. pp. 2048–2057 (2015)

[30] Young, S., Gašić, M., Thomson, B., Williams, J.D.: Pomdp-based statistical spoken dialog systems: A review. Proceedings of the IEEE **101**(5), 1160–1179 (2013)

[31] Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)