# Grid Path Planning with Deep Reinforcement Learning: Preliminary Results

Aleksandr I. Panov, Konstantin S. Yakovlev, and Roman Suvorov

Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia
{pan,yakovlev,rsuvorov}@isa.ru

**Abstract**
Single-shot grid-based path finding is an important problem with the applications in robotics, video games etc. Typically in AI community heuristic search methods (based on A* and its variations) are used to solve it. In this work we present the results of preliminary studies on how neural networks can be utilized to path planning on square grids, e.g. how well they can cope with path finding tasks by themselves within the well-known reinforcement problem statement. Conducted experiments show that the agent using neural Q-learning algorithm robustly learns to achieve the goal on small maps and demonstrate promising results on the maps have ben never seen by him before.

*Keywords:* path planning, reinforcement learning, neural networks, Q-learning, convolution networks, Q-network

## 1 Introduction

Finding a single path on a given static grid is a well-known and well-studied problem in AI, planning and robotics communities [16] with a large variety of methods and algorithms proposed so far. Most of these algorithms rely on heuristic search in the state-space induced by the grid cells and are based on the well-known A* algorithm proposed in 1968 [4]. Among the most wide-spread algorithms that solve the task on-line, e.g. without any preprocessing, one can name IDA* [7], ARA* [8], JPS [3], Theta* [10] etc., all of which are apparently heuristic search algorithms.

At the same time recently we have witnessed a huge break-through in applying neural networks to all sorts of tasks within AI domain, including playing the game of Go [11], recognizing objects from images [13], generating images and representation learning [2], machine translation [17], speech recognition [5], etc. One common thing that can be noticed is that neural networks cope well with all sorts of tasks when sensory or image data (e.g. "raw pixels") is used as an input. Definitely square grids containing only two types of cells, i.e. traversable and untraversable, look like a perfect input to modern artificial neural network (NN) architectures,

such as convolutional NN and residual learning NN. Path finding problem in that case can be viewed as a problem of learning which image pixels (grid cells) to identify as being part of the sought path.

In this paper preliminary results of applying reinforcement learning machinery to 2D grid-based path finding are presented. First, provided with the most commonly-used problem statement we define the appropriate machine learning problem (Section 2). Second, we suggest a variety of tools, e.g. various deep neural network architectures, to solve it (Section 3). And, finally, we evaluate their performance running a series of experiments in simulated environments (Section 4). The results of the experiments are twofold. On the one hand it is undoubtful that the proposed NN learns to plan paths, e.g. after the learning phase it is capable to solve previously unseen instances. On the other hand the NN sometimes fail to find a path. In case a path is found it is likely to be longer than the path found by A* algorithm, e.g. the shortest one. Obtained results provide an opportunity to claim that although the NN cannot be seen as an alternative to well-establishes path finding techniques so far, the application of various deep neural network architectures to path finding tasks is a perspective line of research.

## 2 Problem statement

Consider an agent operating on the 8-connected grid composed of blocked and unblocked cells. Blocked cells are considered to be ones (white pixels of the image) and unblocked zeroes (black pixels of the image). Given the start and goal locations, which are tied to the centers of distinct unblocked cells s and g, the task is to find a path $\pi$ which is a sequence of adjacent traversable cells starting with $s$ and ending with $g$, $\pi = \pi(s, g) = (s, succ(s), succ(succ(s)), \ldots, g)$. Here $succ(x)$ stands for the successor of cell $x$ and we enforce successors to be neighboring (adjacent) cells. Cost of the move between $x$ and $succ(x)$ equals 1 in case its a cardinal move and $\sqrt{2}$ in case its a diagonal move. The cost of the path is sum of all moves comprising $\pi$. In this work we are not limiting ourselves to finding least cost paths only but paths with smaller costs are preferable.

To transform the abovementioned path finding problem formalization, widely used in AI community, to reinforcement learning problem we utilize a framework of agent-centered search which interleaves planning and execution. Planning here means deciding which movement action (going left, right, up, down, etc.) $a_t = p_t \rightarrow p_{t+1}$ to perform next and execution stands to performing this action as well as receiving feedback from the environment, e.g. a reward that is a real number. We denote the position of the agent at time $t$ as $p_t = (x_t, y_t)$, where $x$ and $y$ are grid cell coordinates. We consider only finite sequences of actions until the agent reaches the destination $g$ or the maximal amount of steps $T$ is achieved.

Agent has a limited field of view, denoted $s_t \in S, S = \mathbb{R}^{(2d+1)^2}$. In this work we presume that its a square window of the predefined size $(2d+1) \times (2d+1)$ (say $11 \times 11$, $d = 5$) with the agent placed in its center (so the agent is capable of seeing in all four directions). When the agent approaches a grid edge the portion of the $s_t$ which is out of bounds is filled with ones (so the outer map is considered to be an obstacle).

The reward at time $t$, e.g. $r_t$, is calculated using the unknown to the agent function $G(s, g, t)$, where $s$ and $g$ are start and goal locations. This function as well as grid map, $M$, comprise the model of the environment $E = (M, G)$. The task of the agent operating in this environment is twofold. First agent has to learn. During the learning phase agent aims at maximizing its overall reward while acting in the environment, e.g. relocating on a grid map. Second agent has to navigate to goal location on any previously unseen grid map he is put into without having access to reward function $G$.

## 2.1 Agent learning process

The goal of the agent is relocating on the map with receiving the maximal sum of rewards from the environment generated according the algorithm $G$. We consider all feature rewards for the moment $t$ with discounted coefficient $\gamma$ and summarized reward is calculated as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$. As it is customary in reinforcement learning [6, 12] we define the optimal action-value function $Q^*(s, a)$ as a maximum of expectation of overall rewards gathered according the strategy of action choosing $\pi$:

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}[R_t | s_t, a_t, \pi].$$

If we decide to choose the action with the maximal value it is well-known that the such problem statement boils down to the Bellman equation:

$$Q^*(s, a) = \mathbb{E}_{s_t \sim S} \left[ r_t + \gamma \max_{a_t} Q^*(s_t, a_t) | s, a \right].$$

To find the optimal action-value function we use Q-learning algorithm [15, 9] and approximate the $Q^*(s, a)$ with parametrization $\theta$ given by weights of a neural network: $Q^*(s, a) \approx Q(s, a; \theta)$. A neural network learns minimizing the loss-function, e.g. mean squired error:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right]$$

where $\rho(\cdot)$ is a probability distribution over a training batch of observations and actions, $y_i$ is a summarized reward predicted by a network for the next observation $s_{t+1}$:

$$y_i = \mathbb{E}_{s_t \sim S} \left[ r_t + \gamma \max_{a_t} Q(s_{t+1}, a_{t+1}; \theta_{i-1}) | s, a \right].$$

The problem statement described is model free (in contrast to value iteration approaches [14]) and off-policy due to and don't learn environment's state transition function explicitly and use fixed greedy strategy where $a_t = \arg\max_a Q(s, a; \theta)$.

It is known that neural networks are affected by correlations in the reinforcement data and to decrease learning degradation we use technique of experience replay [9]. We remember all steps $m_t = \{(s_t, a_t, r_t, s_{t+1})\}$ during $N_e$ episodes into memory $D = \{m_1, m_2, \dots\}$. The agent life circle consists of two phase: acting and learning. During acting phase the agent remember the steps and update memory only. During the learning phase it shuffles the memory and uniformly selects a batch for learning until number of learning epochs will not be reached.

## 2.2 Environment reward policy

The algorithm of reward generation is the main dynamical part of the environment $E = (G, M)$. The state of the environment presented by the map $M$ doesn't change after application of the agent's action. But the algorithm $G$ gives different results depending on the current agent's position. To force the agent reach the final position with the shortest path a function $G$ should use knowledge about optimal path.

We consider the following implementation of the $G$ algorithm:

$$G(s, g, t) = \begin{cases} \alpha_{opt} r_t^{opt} + \alpha_{rat} r_t^{rat} + \alpha_{euq} r_t^{euq}, & p_t \leftarrow 0, \\ r^{obs}, & p_t \leftarrow 1, \\ r^{tar}, & p_t = g, \end{cases}$$

where $\sum \alpha_i = 1$. Here to define a usual reward when current position $p_t$ is unblocked we use three parameters. The parameter $r_t^{opt} = l_t - l_{t-1}$ defines the change of optimal distance $l$ to the final position in view of obstacles. The parameter $r_t^{rat} = e^{-l_t/l_0}$ gives bigger rewards for cells that are close to the final position. And the parameter $r_t^{euq} = |p_t - g| - |p_{t-1} - g|$ plays role of regularizator forcing the agent to go directly to the final position and don't pay attention to obstacles. If in the current position $p_t$ the obstacle is located the agent receives punishment $r^{obs}$. And obviously if it achieves the destination it receives the main part of the overall reward $r^{tar}$.

# 3 Neural architectures

In the article we use several deep neural implementation of Q-network. All of them contain dense output layer with linear activation function. Dimension of the output is equal to number of available actions for the agent (8 in our case). Linear activation function supports both positive $Q$ values and negative. The first layers of the network were convolution layers (2 or 3 layers) with sigmoid or ReLU activation function. The main purpose of using them is to construct invariant representations of patterns occurring in agent's visible area. Convolution cores have size of $3 \times 3$ and have been shifted with 1 stride. After each convolution layers we use dropout for regularization and overfitting avoidance. The second type of inner layers was full connected network (from 3 to 5 layers) accepting flatten output after convolution layers. Dense layers use tanh or ReLU activation function and 50 neurons in hidden part. Full schema of the neural architecture is presented on the Fig.1.

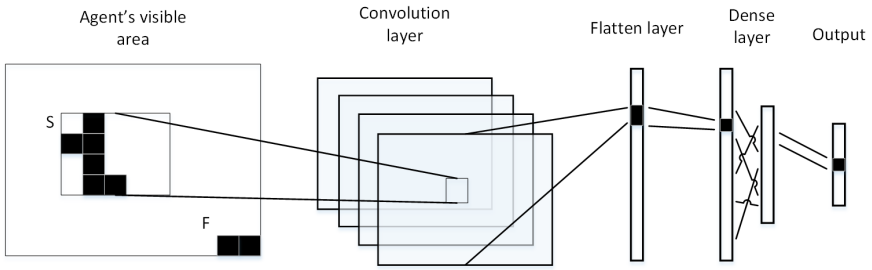


Figure 1: An example of the map used in experiments

Learning process utilizes common parameters in deep learning: batches of samples, ADADelta algorithm for gradient descent, mean squared error as a loss function, several epochs (passes) over training data. We fit the network each 20 episodes.

# 4 Experiments

Experimental evaluation of the proposed method of learning for path finding was based on the OpenAI Gym framework [1][1]. To quickly assess performance, we created a special set of small ($10 \times 10$ and $20 \times 20$) maps with obstacles of "inconvenient for A*" shape. We call obstacle to be of "inconvenient shape", if A* with greedy euclidean heuristic will most probably "get lost in the maze" when trying to build a path around it. In other words, such obstacles increase search

[1]Source code of the algorithm can be found in the repository `https://github.com/cog-isa/deep-path`

space size of "best-first" algorithms (see Fig.2). For each map from the set a task consisting of a pair of valid starting and final positions are generated. Each training episode for the agent includes one of the maps and one of corresponding tasks. After training the agent performance was validated on the set of maps and tasks which are not viewed by the agent.

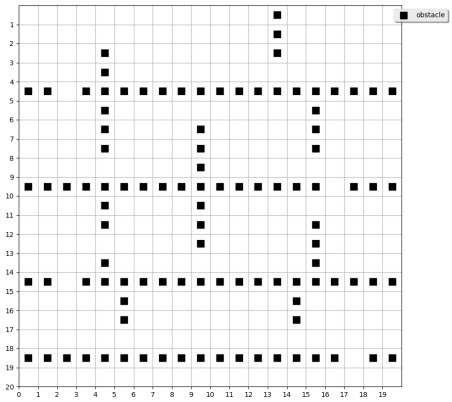


Figure 2: An example of the map used in experiments

To solve the problem of combination of exploitation and exploration we use $\epsilon$-greedy strategy when the agent chooses the random action with probability $\epsilon$ and chooses the predicted action with $1 - \epsilon$ probability. We use annealed strategy of $\epsilon$ predicted decreasing the value of $\epsilon$ during the agent learning to decrease the degree of exploration behavior.

In our experiments we train the agent over the 3000 episodes or considered pairs of a map and a task ($N_{ep} \in [1000, 5000]$). The limit of steps for the agent on the map was 100 ($N_a = 100$) and we define the size of the observed area in such way that agent can view all map in any position ($d = 20$). Another values of parameters were as follows:

- $\alpha_{opt} = 0.8, alpha_{rat} = 0.1$ - parameters of usual reward calculation,
- $r^{obs} = -4, r^{tar} = 10$ - values of obstacle and target rewards,
- $N_e = 10$ - length of the memory,
- $\gamma = 4$ - descending parameter of overall reward.

To evaluate the agent's performance we measured the following metrics during each episode:

- ratio of the path length built by the agent to the optimal path length as a ratio of unique grid cells number in paths (we will refer to it as $M_p$),
- ratio of the sum reward during episode to optimal ($M_r$).

Fig.3,4 present results of some experiments. As one can see, the agent robustly learns to achieve the destination. Best result was achieved using the neural network with 2 convolutional and 5 fully-connected layers. Also, from these figures one can see that neural networks seem to be not fully converged or tend to over fit on partial experience.

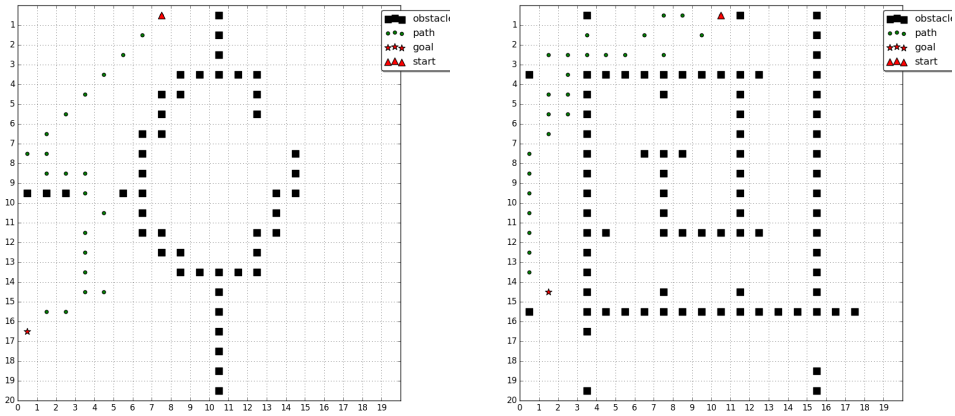


Figure 3: Examples of paths found by the agent during learning

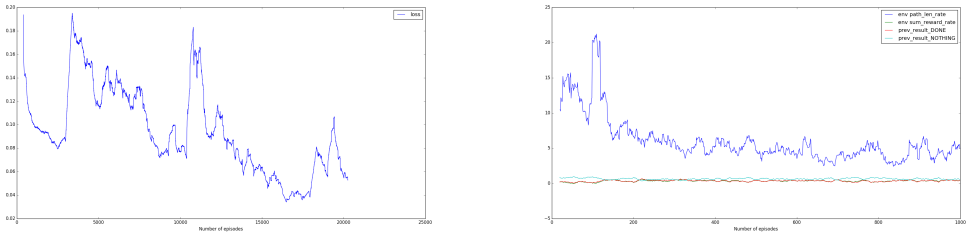


Figure 4: Convergence of learning process: value of loss function is minimized (left part) and paths become shorter

# 5 Summary

In this paper we presented and evaluated a new approach to neural network based path planning using modern deep learning techniques. Experiments show that despite the results are promising, there is much space to improve. Obtained results provide an opportunity to claim that although the NN cannot be seen as an alternative to well-establishes path finding techniques so far, the application of various deep neural network architectures to path finding tasks is a perspective line of research.

# References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[2] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016.

[3] D. Harabor and A. Grastien. An optimal any-angle pathfinding algorithm. In *ICAPS 2013*, 2013.

[4] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[5] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[6] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[7] R. E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[8] M. Likhachev, G. J. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*, 2003.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[10] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *Proceedings of the National Conference on Artificial Intelligence*, page 1177, 2007.

[11] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An Introduction*. The MIT Press, London, 2012.

[13] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[14] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. *arXiv*, pages 1–14, feb 2016.

[15] Christopher J. C. H. Watkins and Peter Dayan. Q-Learning. *Machine Learning*, 8(3):279–292, 1992.

[16] P. Yap. Grid-based path-finding. In *Proceedings of 15th Conference of the Canadian Society for Computational Studies of Intelligence*, pages 44–55, 2002.

[17] Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. Transfer learning for low-resource neural machine translation. *CoRR*, abs/1604.02201, 2016.