Fakulta elektrotechniky a informatiky

Faculty of Electrotechnics and Informatics

Department of Cybernetics and Artificial Intelligence

Subject : **Basics of Deep Learning**
course 2018 / 2019

Project number # 1 (essay) :

Deep Q-Learning in Video games

Key words : Deep Q-Learning, Reinforcement learning, Video games

---

Processed by :
Juraj Kontuľ, Peter Ličko, Tomáš Lichanec

KKUI
Katedra kybernetiky a umelej inteligencie

Contents

1 Introduction

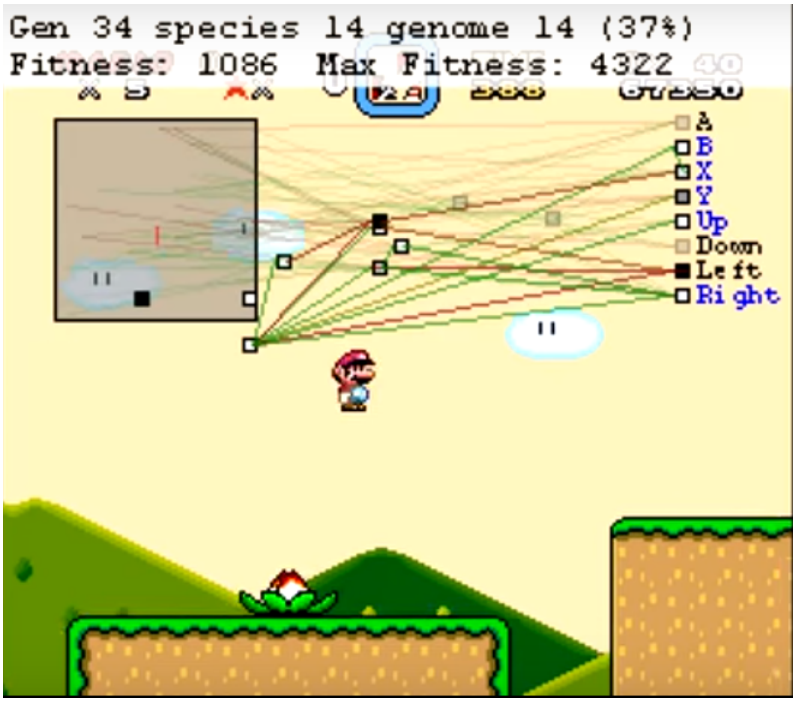
This essay will describe basics of reinforcement learning and changes in order to become deep reinforcement learning and why is it important for a result of higher complexity. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations.Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems, the former evidenced by a wealth of neural data revealing notable parallels between the neurons and temporal difference reinforcement learning algorithms.While reinforcement learning agents have achieved some successes in a variety of domains,their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. This essay describes recent advances in training deep neural networks to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. Stanford University tested this agent on the challenging domain of classic Atari 2600 games [1].

2 Specification of application and description of a problem

In video games such as Super Mario brothers screen capture of game sessions of expert players provide video frames that can be used as input to a model and the output would be directions that Mario could move. That would be supervised classification problem, requiring hours and hours of expert player gameplay and huge requirements for GPU. That is not a elegant solutions, furthermore games do not have static dataset, but dynamic one. New data are continuously streaming, new frames are emerging in this game world, this environment.

Humans learn best by interacting with an environment, not by watching others interact in it. Environments are stochastic, any number of events can occur. It seems best to learn by trying out different possibilities. So rather than framing this problem as solvable by pattern recognition, frame it as solvable through a process of trial and error, which is basic principle of reinforcement learning.

Formalizing this process mathematically, starting with an environment where an AI or agent can perform a number of actions in. Since environments are unpredictable it needs to keep track of its current state. The agent has a state in any given time and based on its actions it may or may not receive a reward - increase in a score. That represents one full episode of this process (episode can be a full game from start to finish) as a sequence of states, actions and rewards for the agent which can be seen on a picture below.



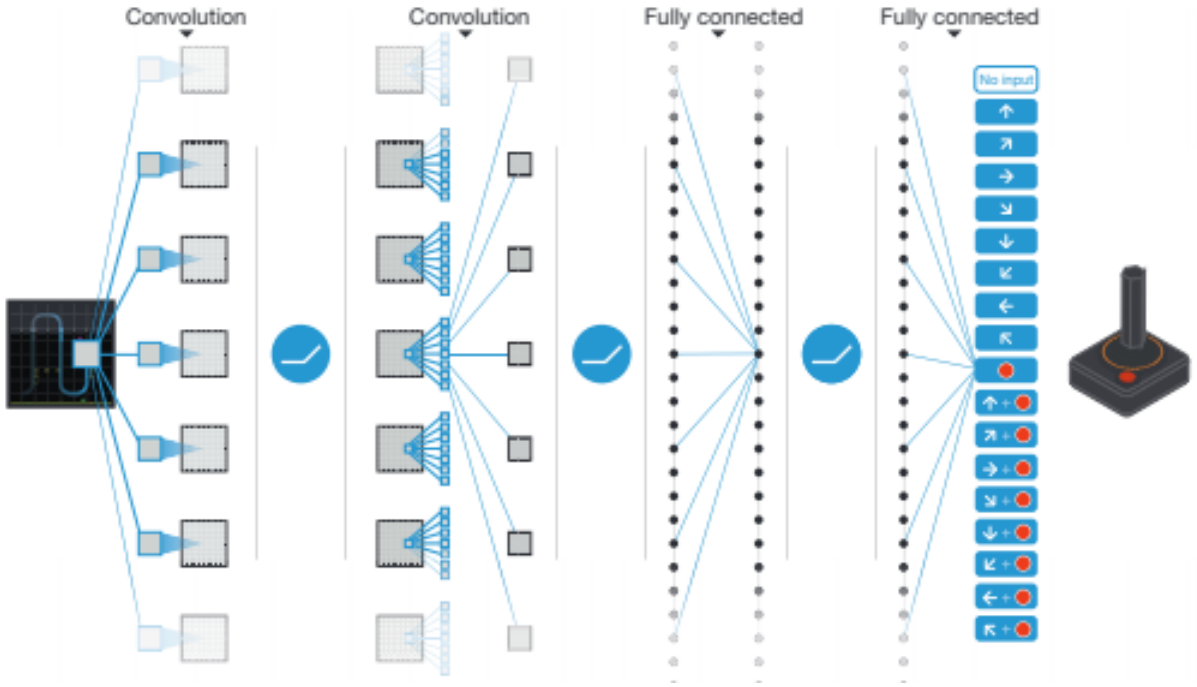
Super Mario played by Deep Q Learning [3].

Probability of each state depends only on immediately previous states in a performed action, but not on a state or action before that, that is called Markov Property. Since the agent is making decisions based on this property, this process is considerate a Markov Decision Process. Agent must plan not only for short-term reward, but also for long-term reward, for a Super Mario example stepping on a Koopa will increase its score in a short-term, but only that. However consuming a star would increase our short-term reward, but also in long-term. That can represent a total future reward for a single episode from a time point "t" onward like this:
G(t):= R(t+1) + R(t+2) + R(t+3) + ... + R(T)
Although it is important that environment is unpredictable, so it is not certain it will get the same rewards in its another episode. The further into the future it goes, the further the rewards could diverge, so add a discount factor between zero and one to the equation.
*G(t):= R(t+1) + Y*R(t+2) +* $Y^2 * R(t+3) + ... + Y^{T-1} * R(T) + ...$
Which means more into the future the reward is the less taken it into an account.

3 Type of a Neural Network and topology

Best is to have balanced value ideally to choose an action that maximizes the discounted future reward. It is possible to represent this discounted future reward when agent performs an action in a state and continue optimally from that point on as a function, this function represents the best possible score at the end of a game after performing a given action in a given state. It is a measure of the quality of an action in that state. That is why call this function Q for quality. Whenever Mario is deciding between several possible actions, the solution is picking the action

that has the highest Q value.
Initialize Q - Choose action from Q - Perform action - Measure reward - Update Q - Choose action from Q - Perform action - ...
Computing this Q function is where the learning process comes in. The maximum future reward for this state and action is the immediate reward plus the maximum future reward for the next state, this is also called the Bellman equation.
$Q^*(s,a) = \left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| s_t = s, a_t = a\right]$
$= \sum_{s'} {}^{a}_{ss'}({}^{a}_{ss'} + \gamma \max_{a'} Q^*(s', a'))$
Q-function can be represented as a matrix where the states are the rows and the action are the columns. Q-matrix is initialized randomly, observing the initial state it can be through a training process where first is chosen an action, later executed and then observed whether or not it received a reward because of it and a new state it got by that chosen action. This is principle of Q-learning and it is used to find the optimal policy for any finite Markov Decision Process.
For Super Mario the state would be presented by the location of enemies on the screen and the obstacles as well as some other factors, but that is specific for this one game. But for wide variety games it must be something generalized, one thing that all the games have in common are pixels. If pixels could be converted into actions, it could fed any game and it would learn the optimal policy for that game. Using a Convolutional Neural Network(CNN) it could use game screens as input and help to put the Q values for each possible actions. Deepmind used the CNN with three convolutional layers and two fully connected layers as can be seen on a picture below.



Schematic illustration of the convolutional neural network [1].

Normally it would also want to use pooling layers which makes the network insensitive to the

location of an object in an image which is perfect for classification tasks like detecting if a picture has a certain object in it, but for video games the location of the player and the enemies is crucial, so pooling layers cannot be used. Input for different game screens as input so that it has a way of representing speed and direction of the game characters and the outputs are the Q-values for each possible action, since it uses a deep network to help approximate the Q-function it is Deep Q-Learning.

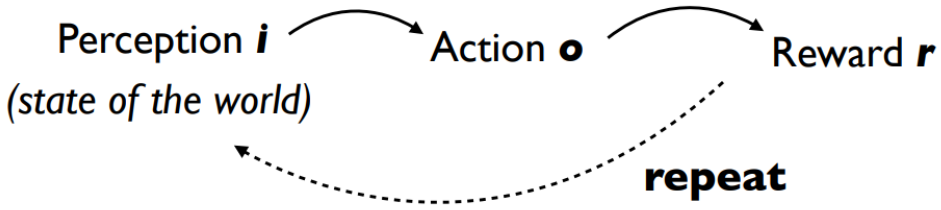
4 Hyperparameters and parameters of a network

The values of all the hyperparameters were selected by performing an informal search on the games Pong, Breakout, Seaquest, Space Invaders and Beam Rider. It is conceivable that even better results could be obtained by systematically tuning the hyperparameter values.

Minibatch size - Number of training cases over which each stochastic gradient descent (SGD) update is computed.
Replay memory size - SGD updates are sampled from this number of most recent frames.
Agent history length - The number of most recent frames experienced by the agent that are given as input to the Q-Network.
Target network update frequency - The frequency (measured in the number of parameter updates) with which the target network is updated.
Discount factor - Discount factor gamma used in the Q-Learning update.
Action repeat - Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
Update frequency - The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
Learning rate - The learning rate used by RMSProp.
Gradient momentum - Gradient momentum used by RMSProp.
Squared gradient momentum - Squared gradient momentum used by RMSProp.
Min squared gradient - Constant added to the squared gradient in the denominator of the RMSProp update.
Initial exploration - Initial value of epsilon in epsilon-greedy exploration.
Final exploration - Final value of epsilon in epsilon-greedy exploration
Final exploration frame - The number of frames over which the initial value of epsilon is linearly annealed to its final value.
Replay start size - A uniform random policy is run for this number of frames before learning start and the resulting experience is used to populate the replay memory.
No-op max - Maximum number of "do nothing" actions to be performed by the agent at the start of an episode. [1]

5 Q-function

This learning framework is used when direct access to the correct output "O" for an input "I" is missing. But it can get a measure of "how good/ bad" an output is. Often called the reward (can be negative or positive). An agent interacts with its environment as shown in picture below.



Agent interaction with its environment [2].

Although description of Q-function is mainly explained in third section(Type of a Neural Network and its topology, the maximum future reward for this state and action is the immediate reward plus the maximum future reward for the next state, this is also called the Bellman equation, which is also already written above. Q-function can be represented as a matrix where the states are the rows and the action are the columns. Q-matrix is initialized randomly, observing the initial state it can be through a training process where first is chosen an action, later executed and then observed whether or not it received a reward because of it and a new state it got by that chosen action. This is principle of Q-learning and it is used to find the optimal policy for any finite Markov Decision Process.

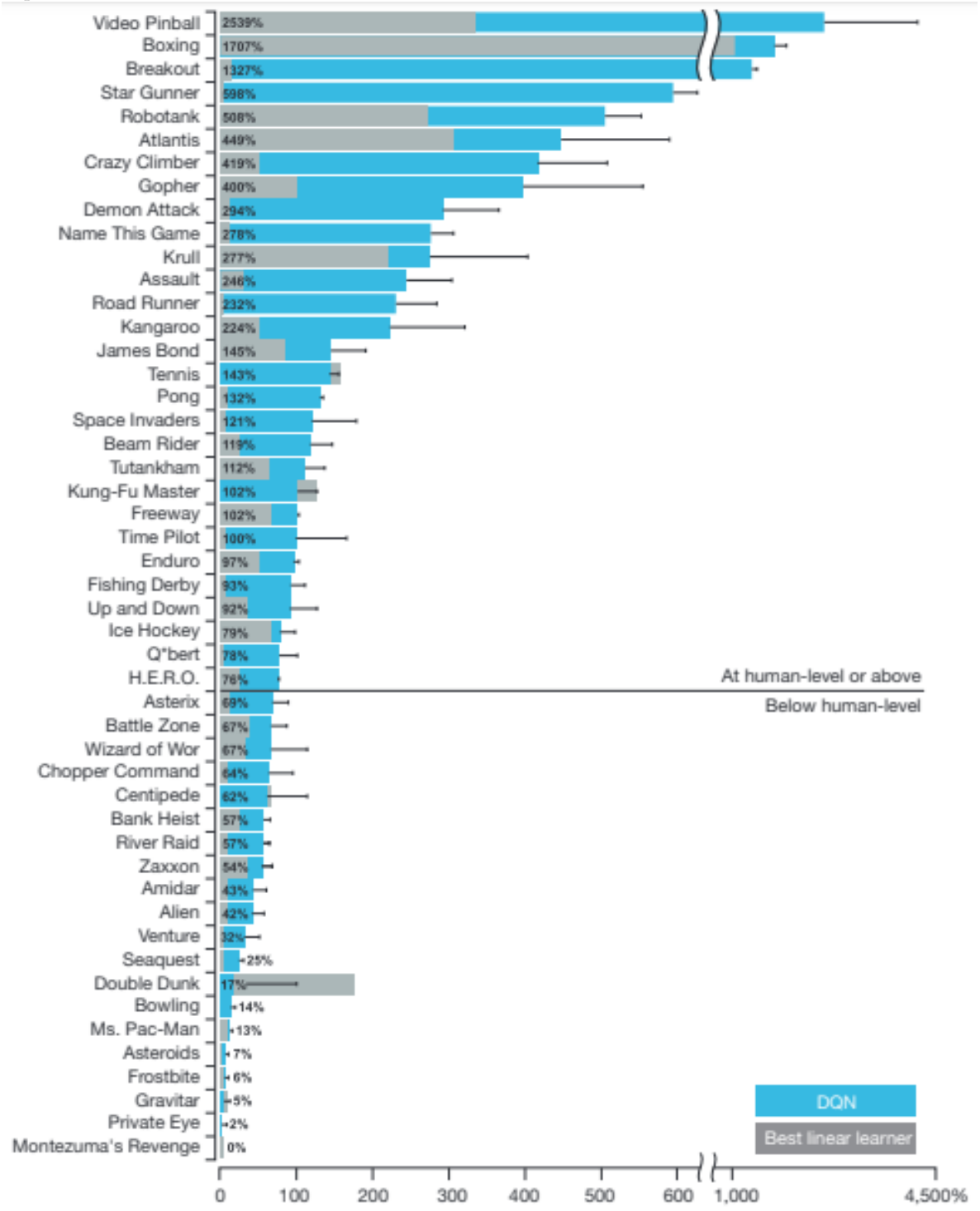
6 Mathematical description of forward transition of a Network

The exact architecture is that the input to the neural network consists of an 84 84* 4 image produced by the preprocessing map Phi. The first hidden layer convolves 32 filters of 8 * 8 with stride 4 with the input image and applies a rectifier nonlinearity[4, 5]. The second hidden layer convolves 64 filters of 4 * 4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64 filters of 3 * 3 with stride 1 followed by a rectifier. The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied between 4 and 18 on the games used for this experiment.*

7 Evaluation of experiment, time of processing, experiences, accuracy

It can be seen that DQN outperforms competing methods in almost all the games, and performs at a level that is broadly comparable with or superior to a professional human games tester (that

is, operationalized as a level of 75percent or above) in the majority of games. Audio output was disabled for both human players and agents. Error bars indicate s.d. across the 30 evaluation episodes, starting with different initial conditions. Comparision for exact games can be seen on a picture below.



The trained agents were evaluated by playing each game 30 times for up to 5 min each time

with different initial random conditions and an 0.05-greedy policy. Accuracy can be seen on a picture above how well it fared against reinforcement learning or expert player.

8 Summary

It is possible to use reinforcement learning if you optimize an agent for sparse, time-delayed labels called rewards in an environment. Markov Decision Processes are a mathematical framework for modeling decisions using states, actions and rewards. Q-learning is a strategy that finds the optimal action selection policy for any MDP and shows that through Deep Q-Learning it is possible to train model for wide variety of games without any major similar trait, but pixels. That is a first step towards unified artificial intelligence, not only for playing video games but also helping with everyday tasks.

9 References

[1]- web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
[2]- folk.uio.no/plison/pdfs/talks/machinelearning.pdf
[3]- ww.youtube.com/watch?v=qv6UVOQ0F44t=0s
[4]- Jarrett, K., Kavukcuoglu, K., Ranzato, M. A. LeCun, Y.What is the bestmulti-stage architecture for object recognition? Proc. IEEE. Int. Conf. Comput. Vis. 2146–2153 (2009).
[5]- Nair, V. Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. Proc. Int. Conf. Mach. Learn. 807–814 (2010).