Simultaneous use of imitation learning and reinforcement learning in artificial intelligence development for video games

Karavaev Vadim
North-Caucasus Federal University
Stavropol, Russia
5665tm@gmail.com

Kiseleva Tatyana
North-Caucasus Federal University
Stavropol, Russia
polet65@mail.ru

Orlinskaya Oksana
North-Caucasus Federal University
Stavropol, Russia
o.orlinskaya@rambler.ru

Abstract

The development of artificial intelligence is one of the most common problems in the video games industry. In most cases, the behavior of computer characters is defined by classical, deterministic algorithms. However, with increasing com-plexity of AI behavior, the complexity of the code describing that behavior also increases. Even more difficult is to create an AI that behaves like a real player. A deterministic algorithm will work efficiently but its behavior may look unnatural and unpleasant. To solve this problem, it is possible to use Machine Learning. Achievements in this field have found application in different niches, and video games have been no exception. This study explores the creation of a convincing and effective AI by simultaneous use of Imitation Learning and Reinforcement Learning. Also this study explores the tools for creating the Learning Environ-ment and learning AI agents, the principles of writing the program code for AI agents, and gives practical recommendations to speed up the learning of AI and improve its efficiency. As a practical example, for the purposes of this study, will be created an agent to control the tank, capable of maneuvering and fighting with several opponents simultaneously.

1 Introduction

Since the creation of video games, the problem of artificial intelligence development, which would make games more interesting to play, has always been relevant. Often artificial intelligence was inferior to players in skills, so video games became too simple for them. In this case, the developers usually resorted to various tricks to adjust

In: Marco Schaerf, Massimo Mecella, Drozdova Viktoria Igorevna, Kalmykov Igor Anatolievich (eds.): Proceedings of REMS 2018 – Russian Federation & Europe Multidisciplinary Symposium on Computer Science and ICT, Stavropol – Dombay, Russia, 15–20 October 2018, published at http://ceur-ws.org

the forces of man and computer opponent. If we are talking about a racing simulator, then the so-called "catch-up" is used. The characteristics of a car belonging to a computer are artificially overstated. Thus, regardless of how skillful the player is, the computer can play with him as an equal. Another example - games of the RTS genre, where the computer takes advantage of the multiplier for the extracted resources. However, such tricks look like a fraud and reject the player. That is why online games are so popular, because playing with a real person is much more interesting. Thus, the problem is the extremely low or unreasonable level of intelligence of computer opponents, which is inferior to the human. Using Machine Learning will solve this problem and greatly expand the capabilities of AI in games. One of the most famous studies on this topic is the paper of DeepMind Technologies employees. With the help of Q-Learning, they managed to implement an algorithm capable of playing simple Atari 2600 video games without knowing anything about them, except for the pixels on the screen. If more information is provided to the neural network (such as, for example, the coordinates of game objects), the application of AI expands. Gathering information about the state of the game world is a fairly simple task if the neural network is intended for in-game AI, which implies the existence of game sources, in contrast to the example with Atari games. In this study the creation and learning within the game agent, which is able to control the tank in a three-dimensional video game, will be considered. The agent's tasks include maneuvering to avoid enemy shells, effective shooting at enemy tanks, turning the turret and using additional game mechanics to gain an advantage on the battlefield.

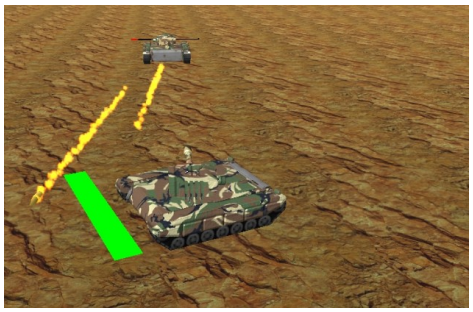

Figure 1: Demonstration of the gameplay of a practical example from the study

2 Background

As a tool for creation of Learning Environment was chosen the Unity game engine. To train game agents was used the Tensorflow. To use trained agents in the game, the project with open source Unity Machine Learning Agents was used. For learning, two approaches will be used simultaneously: Reinforcement Learning and Imitation Learning.

2.1 Reinforcement Learning

The main idea of Reinforcement Learning is that the t-agent exists in a certain S-environment. At any time, the agent may perform an action (or several actions) from the set of A-actions. In response to this action, the environment changes its state and the agent receives the r-reward. Based on this interaction with the environment, the agent must choose the optimal strategy that maximizes his reward.

Reinforcement Learning is especially good for solving problems associated with a choice between long-term and short-term benefits. It has been successfully applied in various fields, such as robotics, telecommunications, elevator management. Also Reinforcement Learning is a good way to develop AI in games. In the case of games, the game character acts as an agent and the world around him and his opponents act as an environment. Every time the character performs an action that approximates him to win, he receives a reinforcing reward. For example, the car agent on a racing track receives a reward over time if the distance to the finish line is reduced. This is also works the other way around - when performing ineffective actions, the agent receives a punishment. In order for the agent to perform effective actions, it is necessary for him to receive an array of data characterizing the state of the environment. The amount of this data should be sufficient to ensure that the agent receives all the necessary information about the environment, but not be too large for the agent to train more effectively. Also, it is necessary to normalize the input data, so that the values of the signals arriving to the agent were within the range of [0; 1] or [-1; 1]. Examples of the input signal for a car include speed and position on the

Figure 2: The reinforcement learning cycle

racetrack. An array with action signals is the result of the agent's work. As well as input signals, they require normalization. Examples of the input signal for a car include the gas pedal [0; 1] and the steering [-1; 1].

2.2 Imitation Learning

As opposed to Reinforcement Learning, which works with a reward/punishment mechanism, Imitation Learning uses a system based on the interaction between a Teacher agent (performing the task) and a Student agent (imitating the teacher). This is very useful in situations where you don't want your AI to have machine-like perfection, but want it to behave like a real person instead. Imitation Learning Support was implemented in ML-Agents v0.3 Beta. This functionality is a powerful mechanism which allows to develop AI with complex behavior using a little effort.

Figure 3: Imitation learning inside Unity Engine

From the outside, the process of AI development looks like this: there are two agents, one is a Teacher and another is a Student. Another neural network, a person or a deterministic algorithm may act as a Teacher. The most effective results are achieved if the Teacher is a real person. Next, the learning process begins. The Teacher plays for a while. The timing varies depending on the task complexity. For simple tasks, it takes about 5 minutes. For complex tasks, it is required up to 2 hours. The learning is that while the Teacher plays, the Student watches his actions and makes attempts to imitate him.

2.3 Simultaneous use of Imitation Learning and Reinforcement Learning

Reinforcement learning and Imitation Learning might be used simultaneously. The Student will watch the Teacher and receive a reward depending on his actions. Simultaneous use of the two approaches makes it possible to achieve much better results than the separate use of them. Initially, only Reinforcement Learning was used in the practical example of this study, and the results were unsatisfactory. After 10 hours of learning, the tank, controlled by an agent, was able to fight a maximum of two opponents. The effectiveness of the tank with more number of opponents, was significantly reduced. However, after Imitation Learning support was added, the tank was able to lead an equal fight with 3-4 opponents after only 40 minutes of learning

2.4 Learning Environment

As a practical example, a battlefield with 1 tank under the agent's control and belonging to the green team and 5 tanks under the simple, deterministic algorithm control and belonging to the red team is used. After completing training of the neural network, the green team's tank must be able to resist the tanks of the red team - this will

be the criterion of a successfully trained agent. As a battlefield, a flat rectangular area that simulates the surface of the earth, is used. In total, 20 battlefields were established, located one above the other at a distance of 50 meters. This approach allows to significantly accelerate the training of agents due to their number and more intensive accumulation of information.

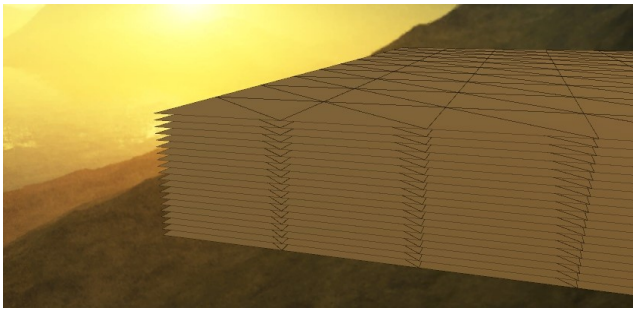

Figure 4: All battlefields located one above the other

Each round is as follows: there are all the necessary tanks on the battlefield and they are placed in random order. Then the battle begins and continues until the tanks of only one team are left. After the battle ends, all the remaining tanks are destroyed, then the round begins again. The game uses the mechanics of "charge accumulation". Its essence is that if you do not shoot for a long time, then the accumulation of "energy" begins. With full accumulation of the scale, it becomes possible to make a shot with a burst of five volleys. The trained neural network should also be able to use this opportunity competently and make the right choice between an immediate single shot or waiting and risk to get five shots.

3 Development

3.1 Neural network architecture

For proper operation of the neural network, an important factor is the correct choice of its architecture and parameters. In this case, based on the existing information on the experience of using different types of neural networks for different tasks, several different configurations were selected and tested. The network parameters that showed the best results are as follows:

- Architecture: feedforward neural network
- Batch size: 64
- Batches per epoch: 5
- Number of layers: 4
- Hidden units: 512
- Activation function: tanh
- Beta: 1.0e-4
- Gamma: 0.995
- Lambda: 0.95
- Learning rate: 3.0e-4
- Time horizon: 512
- Normalize: true

The feedforward neural network architecture and the tanh activation function are the default parameters that are used by the ml-agents' environment for agents training. All other parameters are configured in the file trainerconfig.yaml from the on-line ml-agents and have been selected based on experience.

3.2 Input Signals

The signal is transmitted inside the agent using the AddVec-torObs function built into ML-Agents. To achieve the best results during training, the input signals must be normalized to the range of [-1, 1] or [0, 1]. Due to the value normalization, the neural network finds a solution faster - this is a good practice in the design of neural networks. For the normalization most of the input signals, the satlin transfer function is used.

```
var health = Normalize01(currentData.Health, 0, tankMB.PrototypeData.StartHealth);
AddVectorObs(health);
```

In the final implementation, the tank agent has 27 input signals, 7 of which reflect the state of the tank:

- Physical condition
- Current speed
- Time to recharge
- The angle of rotation of the turret to the shell
- The amount of accumulated energy
- Whether the energy has accumulated completely (Boolean)
- If there is an enemy on the line of fire (Boolean)

Another 18 signals reflect information about the nearest two enemies, per 9 signals each:

- Distance to the enemy
- The angle from the agent tank to the position of the enemy
- Physical condition
- Position along the X axis relative to the agent tank
- Position along the Z axis relative to the agent tank
- The angle between the shell of the enemy tank and the position of the agent
- The angle between the agent turret and the enemy's shell
- Previous value, but converted. This is described in more detail below.
- If there is an enemy on the line of fire

The remaining two signals contain information about the third enemy (the distance and angle relative to the shell). Information about distant enemies is not transmitted, since it is insignificant. Also, some signals were pre-processed, so that the neural network would more effectively perceive them. So, for the angle of rotation of the turret to the enemy, there are two signals that have different shapes (See Fig. 5).

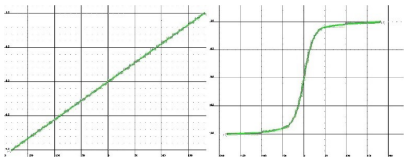

Figure 5: Signals of the angle of the turret to the enemy. Left one-normal, right one-converted in order to improve the shooting accuracy

The left signal has a linear dependence and allows the neural network to quickly direct the turret to the enemy. The right signal has a non-linear dependence and allows to target more accurately. Due to the shape of the curve, the neural network reacts much more strongly to minor rotation angles of the turret. In other words, it is a program analogue of the telescopic sight. In the course of the experiments, it was found that such an approach allows to increase the number of hits of projectiles by 35%. It is recommended to apply this approach to any signals containing small but important ranges where it is necessary to detect the slightest changes.

3.3 Output Signals

The output signals of the neural network allow to control the tank. At the output, the neural network of the practical example has 4 signals:

- Forward-backward motion signal [-1; 1]
- Turret turn signal [-1; 1]
- Shell turn signal [-1; 1]
- Shot signal [0; 1]

Each of the signals is transferred to a secondary low-level module responsible for the control of the tank:

```
ShotSignalAction = Mathf.Clamp(vectorAction[0], 0, 1);
AccelAction = Mathf.Clamp(vectorAction[1], -1, 1);
RotateAction = Mathf.Clamp(vectorAction[2], -1, 1);
RotateTurretAction = Mathf.Clamp(vectorAction[3], -1, 1);
_sublogic.SendCommand(AccelAction, ShotSignalAction, RotateAction, RotateTurretAction);
```

3.4 Rewards

For the correct work of Reinforcement Learning algorithm, it is necessary to use the reward/punishment system at training. This is a very important point; it influences the result behavior of the neural network. Rewards are issued using the AddReward (value) method. The ML-Agents documentation states that rewards should be issued only according to the final result, not the actions, but in this practical example, the issuance of small rewards for certain actions positively affected the results. The final reward system is as follows:

- Damage - +0.1
- Damage taken - -0.3
- Destruction of the enemy tank - +0.4
- Destroying the agent tank -0.8
- Holding the sight on the enemy tank - 0.01 each second
- Shooting in the empty space -0.1 per shot

4 Experiments

4.1 Preparation for learning

After creating the Learning Environments and writing the code for the agents, you can begin to train the neural network. However, before that, it is worth taking several actions. One of the agents should be appointed as a Teacher and it must be managed by a real person. To do this, it is necessary to use the Brain component from the ML-agents project and assign it a Player type. Then it is necessary to appoint the control. Having any of the famous game controllers like the Xbox 360, you may get some advantage. In contrast to the keyboard control, the use of the controller allows to transmit an analog signal instead of a discrete one, because the controller has analog triggers and joysticks. Unfortunately, the CoreBrainPlayer handles signals from the keyboard only. However, the Xbox 360 controller support is a trivial task, all it takes is to add several code lines to the input handler:

```
[System.Serializable]
private class JoyAxisActions
{
public string axis;
public int index;
public float scale = 1f;
}
```

```
[SerializeField]
[Tooltip("The list of axis actions.")]
private JoyAxisActions[] axisActions;
<..>
foreach (JoyAxisActions axisAction in axisActions)
{
var axisValue = Input.GetAxis(axisAction.axis);
axisValue *= axisAction.scale;
if (Mathf.Abs(axisValue) > 0.001)
{
action[axisAction.index] = axisValue;
}
}
```

4.2 Learning

After all the settings have been completed, it is possible to build an executable file that will be used by the Tensorflow environment. In order to start the learning process, it is necessary to insert the following command:

```
python python/learn.py build/build.exe --run-id=0133 --train -slow
```

A window with the game will open, and the person using the previously assigned commands will be able to control the tank. It is very important to prepare, you need to be able to play well - the better the Teacher plays, the better the neural network learns. For a game scene, it is desirable to prepare two cameras in advance. One camera (main) will be aimed at the teacher's tank. The second one (auxiliary) - at the tank of one of the trained agents - it may be placed on a screen quarter in one of the corners. This method will allow to monitor the progress of learning (see Fig. 6)

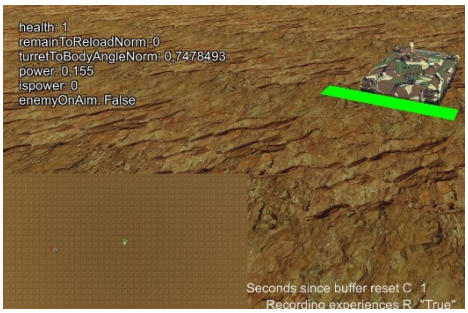

Figure 6: Auxiliary camera placement option

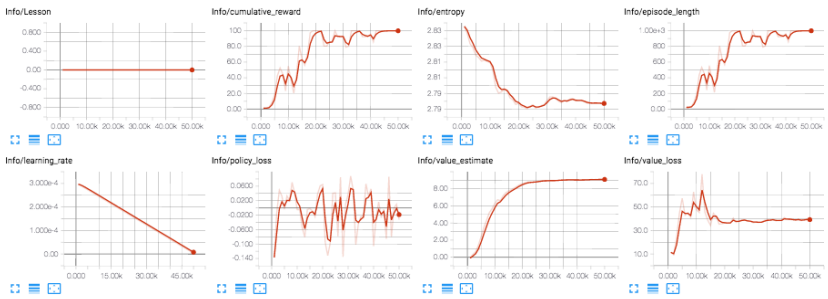

Figure 7: Tensorboard statistics

It's worth displaying all the debug information on the screen, because it's easy to make a mistake in the code and it's much better to learn about it right away and not after a failed learning attempt. It is also very helpful

to use the Tensorboard tool, which displays the progress of learning (see Fig. 7). The most important indication is cumulative_reward - it displays the average reward of each agent per round. Over time, the statistics values should rise.

In this case, after two minutes, the agent, managed by the neural network, learned to ride and make attempts to shoot. After 15 minutes, the counter of the destroyed tanks of both teams showed identical numbers. After 40 minutes the counter of the green team managed by the neural network showed a 3.5 times greater number than the counter of the red team, proving that the agents successfully completed the learning.

5 Conclusion

This study explores the opportunities and benefits of simultaneous use of Reinforcement Learning and Imitation Learning in artificial intelligence development for video games. Tools for creating the Learning Environment and learning AI agents have been considered. Practical recommendations, allowing to optimize the parameters and characteristics of the neural network and to conduct more effective training, were given. In the final result, a video game agent, which controls the tank, effectively uses the available game mechanics and whose behavior is similar to a human, was created and trained.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. *Human-level control through deep reinforcement learning.* Nature, 518(7540):529–533, February 2015.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. *Learning to learn by gradient descent by gradient descent.* In Neural Information Processing Systems (NIPS), 2016.

[3] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. *Using fast weights to attend to the recent past.* In Neural Information Processing Systems (NIPS), 2016.

[4] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. *Learning invariant feature spaces to transfer skills with reinforcement learning.* In Int. Conf. on Learning Representations (ICLR), 2017.

[5] Jonathan Ho and Stefano Ermon. *Generative adversarial imitation learning.* In Advances in Neural Information Processing Systems, 2016.

[6] Ke Li and Jitendra Malik. *Learning to optimize.* arXiv preprint arXiv:1606.01885, 2016.

[7] Sachin Ravi and Hugo Larochelle.*Optimization as a model for few-shot learning.* In Under Review, ICLR, 2017.

[8] Bradlie Stadie, Pieter Abbeel, and Ilya Sutskever. *Third person imitation learning.* In Int. Conf. on Learning Representations (ICLR), 2017.

[9] P. Englert, A. Paraschos, J. Peters, and M.P. Deisenroth. *Model-based Imitation Learning by Probabilistic Trajectory Matching.* Proceedings of the International Conference on Robotics and Automation, 2013.

[10] T. Lens. *Physical Human-Robot Interaction with a Lightweight, Elastic Tendon Driven Robotic Arm: Modeling, Control, and Safety Analysis.* PhD thesis, TU Darmstadt, Department of Computer Science, 2012.

[11] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. *The arcade learning environment: An evaluation platform for general agents* In: Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.

[12] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel *Benchmarking Deep Reinforcement Learning for Continuous Control* In: arXiv preprint arXiv:1604.06778, 2016.

[13] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller *Emergence of Locomotion Behaviours in Rich Environments* In: arXiv preprint arXiv:1707.02286, 2017.