Enhancement of Direct Future Prediction by adding time and memory information

Théo VINCENT
ENS Paris-Saclay & ENPC
theo.vincent@eleves.enpc.fr

1. Introduction

The project focuses on the paper [9] where an algorithm called Direct Future Prediction (DFP) is introduced. This algorithm is part of the Deep Reinforcement Learning (DRL) literature, its goal is to guess the best possible action given a specific state. This algorithm was introduced in the context of a competition of DRL called Visual Doom AI Competition [15] where the participants were asked to submit agents capable of playing the game ViZDoom [16] in the best possible way. DFP won the tournament in 2016.

The authors of the paper released a github repository with the code allowing readers to reproduce the paper [7]. Later, Intel AI Lab integrated this algorithm in their wide publicly available framework for DRL development called Coach-RL [8]. Although it is possible to reuse the code from the authors, I chose to dive into Coach-RL since there has been much more effort to make this framework easy to use and reliable. Indeed, I was even able to spot one mistake myself in the original code. I opened an issue here: [13].

DFP manages to make the agent learn a really effective policy. One major drawback of this algorithm is that the training is slow, it requires millions of steps to be able to learn decent policies. Thus, we will focus on making the training quicker. The contribution of this project is to show that depth information given as an additional input can help decrease the training time. We will explore the case where the depth information is given by the game engine and also the case where it is predicted from a neural network. Another contribution of the project is that memory can also help reduce the number of training steps required to get interesting policies.

The code that I developed for this project is available here [12], I started from a fork of the Coach-RL repository. The RL agents were trained on Google Cloud Platform and the neural network for depth estimation was trained on Google Colab to use a GPU.

2. Environment

I will only focus on the environment called D2 in the paper. In this environment, the agent is left in a maze where its life decreases with time. In order to stay alive the agent has to collect medkits that add 20 health points to a maximum of 100 points. The agent also has to avoid poison that would remove 30 health points. It is important to understand that those items keep spawning on the map when the agent acts. The action space is any combination of the following actions: move forward, turn left, and turn right which makes an action space of size 8.

In order to study the behavior of the learnt agent, I changed the settings of D2 to prevent the items from spawning on the map. This means that there are only 15 medkits and 10 poisons on the map. To make it more interesting, I increased the value of the medkits, now, the medkits regenerate the health points up to 100 if they are collected. This new setting is called D2 No Spawn. It forces the agent to explore the maze rather than waiting for medkits to spawn.

A demonstration of D2 is available here: [2] and one for D2 No Spawn is available here: [3].

3. Architecture

DFP stands out from the traditional DRL algorithms for two reasons. The first one is that not only it takes as input frames from the game but it also takes measurements from the game. The second one is that it does not try to predict the state-action value function or the policy but it predicts for each possible action, the future measurements for 6 following timestamps. Then, the chosen action is the one that would entail the best measurements with respect to a goal. Figure 1 shows the architecture of the neural network. In our case the goal is not important since it is only to maximize health. It is important to have in mind that the neural network takes as input 3 consecutive frames stacked together in one sample.

In some experimentations, depth maps are added to the input. The idea is coming from the work of Dr. Wenzel et al. [14]. It is done in the same way the frames from

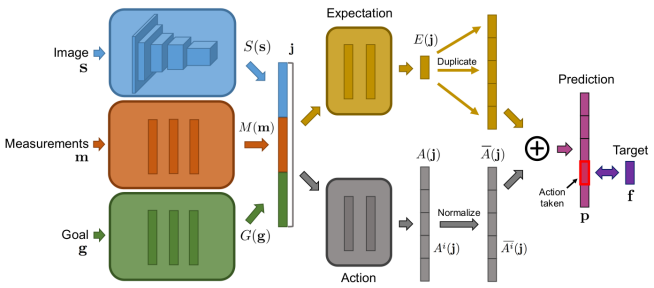

Figure 1. Architecture of the neural network. Graph from the original paper [9].

the game engine are given. All in all, the network receives 3 consecutive frames from the game engine, 3 consecutive depth maps, the health points and the goal (constant here).

In the work of Dr. Wenzel et al. [14], they also propose to generate depth maps from the frames of the game engine. In that case, a conditional GAN (cGAN) is used. The architecture is exactly the one of the paper known as pix2pix [11].

4. Training

I first wanted to have a strong and reliable baseline. With the code from Coach-RL, I tried to reproduce the results on D2. The difference of computational power prevented me from going up to 50 millions steps of training. I was only able to reach 3 millions steps. This is the reason why I did not reach their results. Table 1 shows the average health at the end of an episode on D2. Since the number of training steps are different I also show the average health if only the success are taken into account. The standard deviation of the results are not shown since it would require to train several times the agent. It is worth noticing that in the paper, they only reach 55 points of health after 15 millions steps which is 5 times slower than the implementation of Coach-RL.

	D2
From original paper	84.1
From Coach-RL success only	80.3
From Coach-RL	55.5

Table 1. Average health at the end of an episode on D2

Having that done, I trained on the environment D2 a DFP agent with three different input types. One with 3 consecutive frames, as in the original paper, one with 3 frames and depth maps (taken from the game engine) and one with 10 consecutive frames. The idea of training a DFP agent that sees 10 consecutive frames was taken from a paper investigating memory in DRL [10]. In this paper, they compare the performances of an agent trained with 10 consecutive frames and an agent trained with an LSTM. They show that, in the case of a Partially-Observable Markov Decision Process (POMDP), an agent that sees more frames performs better. They also demonstrate that an agent that has an LSTM in its architecture but that only sees one frame can perform even better. In our case, the two environments are POMDPs. Indeed in both environments, the agent does not see the entire map. Adding an LSTM in the architecture is not an easy task since it entails changing the way the replay buffer is coded so that the back propagation can be done on several samples at a time. This is why before implementing a different architecture with an LSTM, I wanted to see what we can have with an agent that simply sees more frames.

The evaluation reward on the environment D2 is available on the Figure 2. We can see that the training with only 3 frames performs worse than the two other settings. It is important to understand that the difference is not huge here because the evaluation during training is done on D2, we will see how those agent behave on D2 No Spawn.

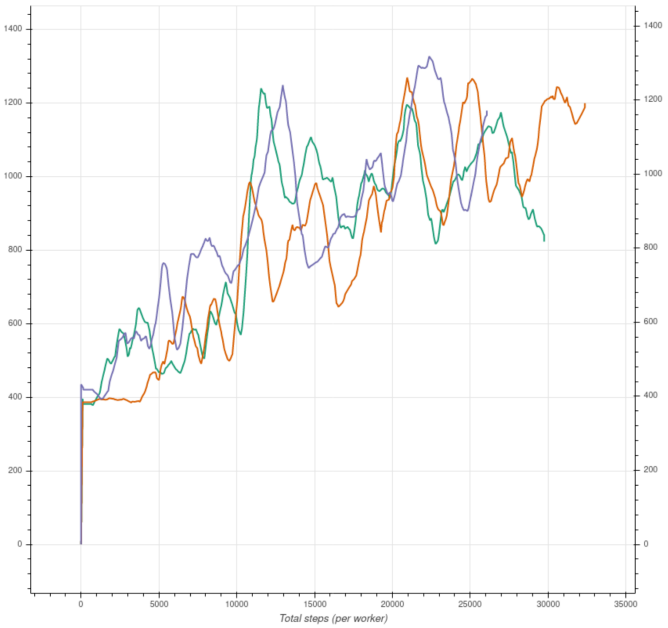

Figure 2. Evaluation reward on D2 during the training on D2 according to the number of steps per worker (7 workers in total). orange: 3 frames, purple: 10 frames, green: 3 frames + depth

To generate depth maps from the frames of the game engine I trained the conditional GAN introduced earlier. The generator of this cGAN has an additional term in its loss which is the L1 reconstruction loss. It is really convenient because it allows us to see how the training behaves. Figure 3 shows the L1 training loss during the training. A demonstration is available here: [4] showing that the cGAN is capable of producing really good predictions of depth maps.

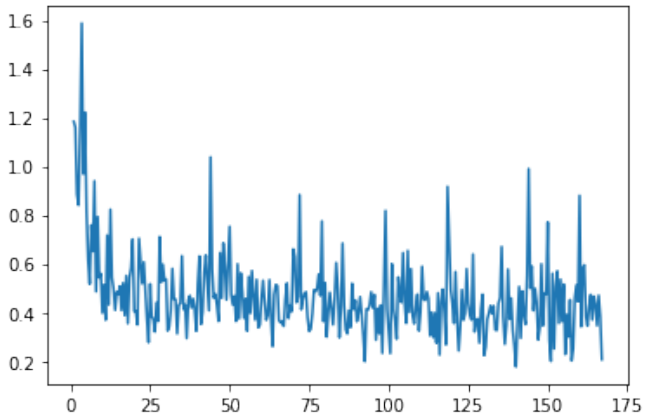

Figure 3. L1 reconstructive loss of the generator according to the epochs during the training of the cGAN.

5. Experiments

To evaluate the ideas that were developed, the different settings were tested on the two environments over 300 episodes. We recall that these agents were only trained on D2. The average reward at the end of the best 20 episodes is reported in table 2. The idea of removing the 280 worse episodes comes from the fact that since the training is stopped very early, there are still some situations where the agent gets stuck. Moreover, the medkits and the poison spawn randomly on the map, by removing the worse episodes we make sure that the items did not spawn in a bad combination that would make the agent blocked. We recall that if an agent reaches 2100 of reward, the episode is considered successful and is stopped. It is worth noticing that the agent that sees 3 frames and has the depth estimation from pix2pix is the only agent that receives as much information from the game engine as the agent that receives 3 frames like in the original paper.

We can first remark that the environment D2 is not interesting since all the agents are capable of reaching 2100 of reward. The results on environment D2 No Spawn are more interesting. Indeed, the agent receiving 3 frames and depth maps and the agent receiving 10 frames performed significantly better than the agent receiving 3 frames. It shows that this additional information helps the training a lot. When tested with depth maps from the cGAN pix2pix, the results are slightly below the ones where the agent had the depth maps directly from the game engine. It is not a surprise since the information is not as accurate as the one from the game engine. It is still interesting to note that the results are better than when the agent only sees 3 frames.

Those results can be understood easily with the video clips showing an episode of each agent on D2 No Spawn. The links of those video clips are available on table 2. In those video clips, we can clearly see that the agent with 3 frames only learnt to wait until a medkit spawn on the map. When tested on D2 No Spawn, the agent is totally lost. The only reason why the agent manages to stay alive is because in some cases, the agent is able to see a medkit and once the agent has reached it, the agent can see another one. Another interesting fact is that the agent never turns right. If it needs to turn left, it does an entire turn on itself. With the depth information, we can see that the agent knows how to explore the maze to find the medkits. Surprisingly, it is also the case when the agent sees 10 frames.

	D2	D2 No Spawn	Video link
3 frames	2100 ± 0	1128 ± 200	[3]
3 frames + depth	2100 ± 0	1676 ± 296	[6]
3 frames + depth from pix2pix	2100 ± 0	1574 ± 250	[5]
10 frames	2100 ± 0	1781 ± 172	[1]

Table 2. Average and standard deviation of the 20 best testing reward at the end of an episode out of 300 episodes on D2 and D2 No Spawn

6. Conclusion

To conclude, I have shown that it is possible to make an agent learn a policy that behaves better than the original network of the paper for a small amount of training steps. This can be done by adding depth information to the inputs of the algorithm or by adding more frames. An interesting fact is that the depth information can be predicted directly from the frames of the game, this makes the algorithm better than the original paper without giving additional information to the agent. It is the way the information is represented that makes the agent learn faster.

Further investigation could focus on making the agent wait before collecting a medkit. Indeed, since the health points are capped at 100 points, if the agent would wait for its life to decrease before taking a medkit, the agent would live longer.

References

[1] 10 frames on D2 No Spawn. https://www.youtube.com/watch?v=Nr91edDl2VY.

[2] 3 frames on D2. https://www.youtube.com/watch?v=QANcd6mH8CU.

[3] 3 frames on D2 No Spawn. https://www.youtube.com/watch?v=YBbic4e8sXU.

[4] 3 frames with depth and pix2pix on D2 No Spawn. https://www.youtube.com/watch?v=Nmet69dG7cA.

[5] 3 frames with depth from pix2pix on D2 No Spawn. https://www.youtube.com/watch?v=9Y5hR894Qd4.

[6] 3 frames with depth on D2 No Spawn. https://www.youtube.com/watch?v=YBbic4e8sXU.

[7] Alexey Dosovitskiy and Vladlen Koltun. Direct Future Prediction GitHub Repository. https://github.com/isl-org/DirectFuturePrediction.

[8] Itai Caspi, Gal Leibovich, Gal Novik, and Shadi Endrawis. Reinforcement Learning Coach. https://github.com/IntelLabs/coach, 2017.

[9] Alexey Dosovitskiy and Vladlen Koltun. Learning to Act by Predicting the Future. In *International Conference on Learning Representations (ICLR)*, 2017.

[10] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *2015 aaai fall symposium series*.

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[12] Théo VINCENT. Code for the project. https://github.com/theovincent/coach.

[13] Théo VINCENT. Issue on original code. https://github.com/isl-org/DirectFuturePrediction/issues/10.

[14] Patrick Wenzel, Torsten Schön, Laura Leal-Taixé, and Daniel Cremers. Vision-based mobile robotics obstacle avoidance with deep reinforcement learning. *CoRR*, 2021.

[15] Marek Wydmuch Wojciech Jaśkowski, Michał Kempka. Visual Doom AI Competition. http://vizdoom.cs.put.edu.pl/competitions/vdaic-2016-cig, 2016.

[16] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018.