Deep Pacman

Kyrylo Rudavskyy
Ryerson University
Toronto, Canada

I. Introduction

This work follows in the footsteps of DeepMind's breakthrough in deep reinforcement learning (DRL)[2]. The company introduced a novel way to combine deep and reinforcement learning. Although the use of deep networks for the purpose of value function approximation was previously considered, DeepMind was the first to solve significant issues that hindered the technique. The result was superhuman performance on most Atari games.

However, some games proved challenging to solve with this approach. Ms. Pac-Man is a good example where DeepMind's technique does not produce a considerable advantage over human performance. While in some games, DeepMind reached scores ten times above human capability, in Pacman, it could only reach about a tenth of human performance.

This work adopts the conclusion of Seijen et al.[3] that the objective function is too challenging to learn in the case of Ms. Pac-Man. The reason is that the environment is too dynamic and contains a stochastic component. In Seijin's words: "... if the optimal value function is very complex, then learning an accurate low-dimensional representation can be challenging or even impossible."[3, p. 2] I demonstrate that the author's approach of dividing and conquering the reward function makes it possible to perform in a complex environment such as Ms. Pac-Man.

The remainder of the report is organized as follows. Section two will give a detailed description of Ms. Pacman along with the learning objective and accompanying difficulties. Section three will describe the methods employed to solve the problem. Section four will present and discuss the results of this experiment. Finally, third-party libraries and tools that were used here will be described in section five.

II. Problem Statement

The experiment's main objective is to produce a DRL agent capable of outperforming DeepMind at the Atari game of Ms. Pac-Man. A contingent aim is to demonstrate the possibility of reaching superhuman performance. Ms. Pac-Man is composed of four maps.

Each map contains the following elements: pellets, power pellets, multi-colour ghosts, blue ghosts, fruits and a human-controlled pawn called Pacman. These objects are dispersed in a labyrinth. The objective is to consume pellets and fruits while avoiding ghosts. Contact with a ghost reduces the three initial lives by one. Power-pellets, on the other hand, make the ghosts edible. Fruits give extra points. Finally, consuming all the pellets makes the player progress to the following map. That last element is precisely the implicit objective of the current experiment. A visual representation of the game can be seen in Fig. 1.

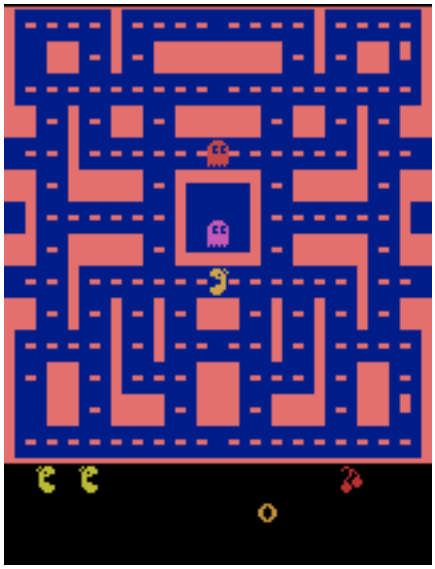

Fig. 1: Source: OpenAI [1]

This experiment is limited to only the first map because it is sufficient to extrapolate experimental objectives. In principle, the maps are very similar environments, and if the agent can outperform DeepMind on the first one, it can do so on the rest. Moreover, the first map contains almost enough points to demonstrate superhuman performance[3].

The experiment scope was further limited by ignoring the fruit artifacts. These are moving artifacts that pseudo-randomly appear on the map and carry bonus points. If Pacman is capable of achieving the pellet collection task, it will be capable of collecting fruits. However, limiting the scope in the two ways described above significantly reduces project complexity in computation and human resources. This is necessary to make the project feasible.

III. Methodology

A. Architecture

Pacman game is composed of two main tasks: pellet collection and ghost avoidance. A 100% success at both of these tasks is required to gain levels and terminate the game. Moreover, ghosts can change velocity, teleport through walls and disappear. When compared to other Atari games, like

Pong, this is significantly more challenging. In Pong, the agent only has to control one non-stochastic task: hit a ball.

Seijen's idea is to divide the reward function into sub-functions. Consequently, the first version of the agent was composed of two agents. One agent was trained to avoid ghosts while ignoring all other game elements. The other agent was trained to collect pellets while also ignoring other game elements. The decisions of the two agents were weighted, and the hope was that a simple greed-vs-fear balance would solve the problem. This, however, turned out to be a failure.

Not only was the agent incapable of potentially reaching superhuman performance, but it also was not even able to outperform DeepMind's agent. Tweaking and tuning the weights, the number of training episodes or network parameters did not significantly improve the result. This approach was abandoned. Moreover, the architecture was horribly inefficient. Each agent employed a convolution network to capture information from the video frames and estimate a value function. Basically, each agent was a clone of DeepMind's single agent.

The second version of the agent was much closer to Seijen's version. Each pellet and ghost became a different head of the deep network. The last layer of DeepMind's network was replaced with 100+ separate layers connected to the base of the deep network. This produced an equivalent number of Q matrices. Finally, a weighted average of the n heads was taken. Therefore, the loss function resemble closely Seijin's version, which is

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\sum_n (y_k - Q_k(s,a;\theta))^2\right]$$

where $y_k = R_k(s,a,s') + \gamma \max_{a'} Q_k(s',a';\theta^-)$.

This is a familiar Q-learning-based DRL setup. The idea of periodically updating the target network is borrowed from DeepMind's approach. This is done to make the neural network's targets stationary. Moreover, a high sample correlation inherent to a video stream is broken using random sampling or, in DeepMind's words, experience replay. The novelty here is the usage of R_k for each of the n heads and taking a sum of the squared term. Moreover, Seijin defines

$$Q_{HRA} = \sum_n w_k Q_k \text{ and } R_{HRA} = \sum_n w_k R_k$$

I modified this slightly to simplify the computation such that

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\sum_n Y_{HRA} - Q_{HRA}(s,a;\theta)\right]^2$$

where $Y_{HRA} = R_{HRA}(s,a,s') + \gamma \max_{a'} Q_{HRA}(s',a';\theta^-)$. Moreover, I redefined Q_{HRA} and R_{HRA} as follows:

$$Q_{HRA} = \frac{1}{n}\sum_n w_k Q_k \text{ and } R_{HRA} = \sum_n R_k$$

I changed the loss function because I wanted to square a sum of terms rather than sum a series of squared terms. This seemed computationally easier when taking a MSE of a batch. The other two changes were a simple mistake that turned out to produce a desired result. A visual depiction of the architecture can be seen in Fig. **??**.

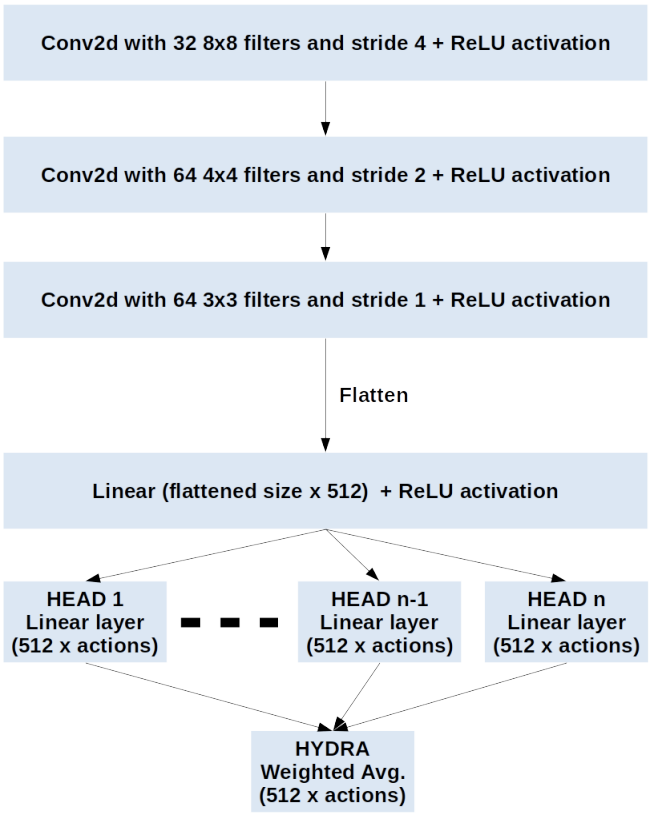

Fig. 2: HydraNet

The final important innovation adopted from Seijin is the use of a diversification head. The author notes that using an ϵ-greedy approach to exploration makes the agent too risk-tolerant. Instead, Seijin proposes to add a diversification head filled with numbers drawn from a normal distribution in the range of [0, 20]. This is the approach that was taken here.

B. Pseudo Rewards and Terminations

On a high level, the above agent does not learn to play Pacman. It learns to find the shortest path to each of the game artifacts. This is the purpose of the heads. Therefore, the actual reward that it receives is a pseudo-reward. Each agent gets a hundred points for reaching its designated artifact and a minus point for loitering on the game board.

Similarly, pseudo rewards are complemented by pseudo terminal states. Each head's terminal state is when the agent reaches the head's designated object. For example, a purple ghost has a corresponding head. If Pacman reaches it, it will receive a positive reward, and the episode will terminate. This also gets multiplied by the number of locations. However, object-location pairs are abstracted away by the network.

The actual game-logic rewards come from the weights. At each time step, the system evaluates the shortest distance to each object and weighs it according to domain knowledge.

This way hitting a ghost, blue ghost or a pellet gives -1000, 1000 and 10 points.

C. Preprocessing

While the previous section demonstrates the general architecture of the agent, the actual input to the network was not mentioned. This, however, is a critical part of the system. Keeping track of more than a hundred game artifacts in a computationally efficient manner is a difficult task.

The first immediate steps are as follows. Four consecutive frames are maxpooled together. The game action area is cropped. The image is downscaled by two. Thus, we arrive at an 80x80x3 image. The next step is to detect all the objects.

First, Pacman and the ghosts are detected. This is done by filtering colours because Pacman and ghosts have unique colours. Their colours are pre-computed and stored. Once detected, pixels containing the object are assigned a value, while other pixels are zeroed out. Each of the five objects goes on a unique channel. All other artifacts will be represented in the same way on an individual, unalterable channel.

Second, the blue ghosts are detected. These are also filtered by a unique colour. However, since there are four ghosts, they must be separated. Segmentation is done using Sobel edge-detection and Watershed basins. This approach was inspired by an example from Scikit-Image[1].

Finally, the pellets are detected using Soble and Watershed method from above. However, because the pellets are stationary objects, their locations were pre-computed and stored. The detection algorithm only checked the presence of each pellet using colour filtering in a pre-computed location.

Once everything is detected, all channels of the frame are downsampled by two again. A network input frame is ready with a size of 40 x 40 x 142. An example of a preprocessed channel can be seen in Fig. 3.

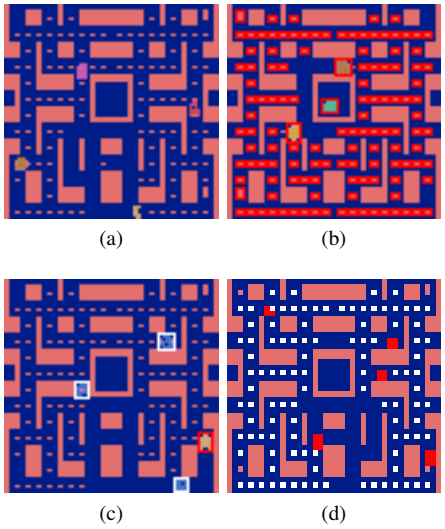

Fig. 3: Preprocessing visualizations: (a) downscaled once, (b) tracking objects, (c) tracking blue ghosts, and (d) final result.

IV. Results

The experiment's results show considerable performance gains over DeepMind's agent. On average, Hydra(name borrowed from Seijin) architecture achieved a 30% higher result in half the runs. This can be seen in the top chart of Fig. 4.

Moreover, video analysis shows that my agent is more interested in eating pellets than blue ghosts. It appears that the agent is using the power pellets to chase away the ghosts from pellet fields. This is a significant result because eating blue ghosts can give higher scores, but the agent might not finish the level without power pellets.

The score achieved here would have been higher had the fruit artifacts been taken into account. This is not an overwhelmingly difficult extension of the current architecture. But, I am more interested in the following finding.

After 500k runs, Hydra shows an asymptotic behaviour while DeepMind keeps learning slowly. Finally, after 3m training rounds, DeepMind's agent catches up, while my agent stays constant. Upon visual examination of my agent's behaviour, it became clear that the agent struggles with the last few pellets. This is the cause of the asymptotic behaviour!

It is already a good achievement that the agent can budget its lives to consume most of the pellets after only 500k runs. Also, if only a handful of pellets remain, that means the agent succeeds at 90+% of its task. This is usually a good result in machine learning. But, we need to get to 100% to gain a level.

If a solution to this "last-mile" problem can be found, then the agent will clearly be on its way to superhuman performance. The reason is that the agent gains about 2000 points from a level. If it passes four levels, it will accumulate 8000 points. Then it only remains to add a fruit collection head.

Each fruit gives about 1000 points, and the agent can be easily expected to eat a fruit per level. This sums up to 12000 points in total. Given that human performance peaks at about 15000 points[3], it is very likely that this architecture can reach superhuman performance.

However, this optimistic prediction rests on the agent's ability to finish the level. Seijin also encountered this problem. His agent was struggling with the last few pellets. The good news is that the author was able to solve this problem by normalizing the score heads to [0, 1] and the ghost multiplier to -10. In essence, tweaking and tuning the system solved his problem. My agent's problem can likely be solved this way as well. It is also worth noting that the last-mile problem will affect DeepMind's agent.

V. Implementation

The agent was is an extension of the code provided in Ryerson RL course[2]. This codebase was built using Python and relied on the OpenAI's Gym[3]. The neural network was

[1]https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_expand_labels.html#sphx-glr-auto-examples-segmentation-plot-expand-labels-py

[2]http://narimanfarsad.com/cps824/syllabus.html

[3]https://gym.openai.com/

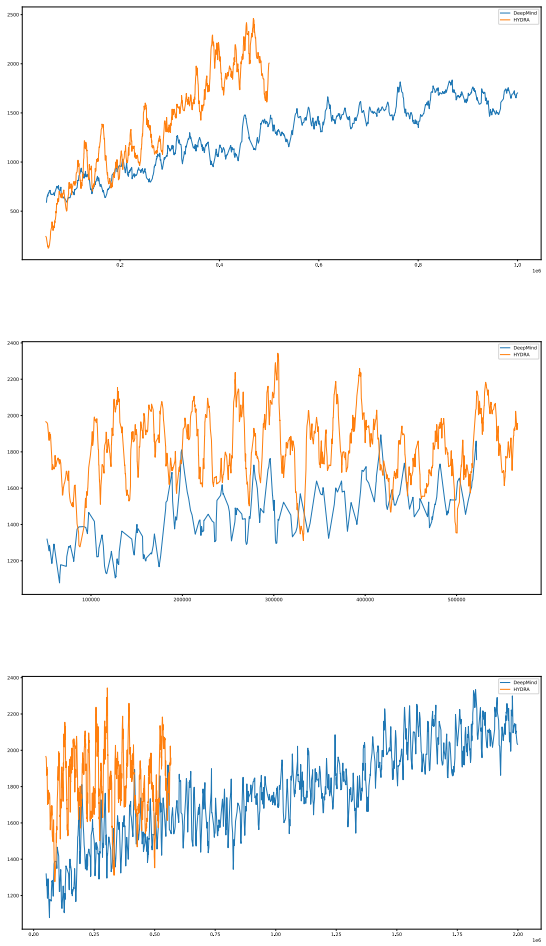

Fig. 4: Top image shows 500k runs of Hydra and 1m runs of DeepMind agents. Middle image shows an additional 500k rounds on top of the ones in the top image. Bottom image is the same the the middle one except that DeepMind reached 3m steps.

built using Pytorch[4]. Computer vision was built using scikit-image[5]. Linear algebra calculations were done using Numpy[6]. Finally, the results were analyzed using Tensorboard[7].

REFERENCES

[1] Greg Brockman et al. "OpenAI Gym". In: *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540 [cs].

[2] Volodymyr Mnih et al. "Human-Level Control through Deep Reinforcement Learning". English. In: *Nature* 518.7540 (Feb. 2015), 529–533H. ISSN: 00280836. DOI: http://dx.doi.org.ezproxy.lib.ryerson.ca/10.1038/nature14236.

[3] Harm van Seijen et al. "Hybrid Reward Architecture for Reinforcement Learning". en-US. In: (July 2017).

[4] https://pytorch.org/
[5] https://scikit-image.org/
[6] https://numpy.org/
[7] https://www.tensorflow.org/tensorboard