MAP511 - Project Report

Reinforcement learning for the game of Snake

Hugo Artigas[1], Reda Belhaj-Soullami[1], and Mohamed Mimouna[1]

[1]Ecole Polytechnique

Abstract

The goal of this project is to apply Reinforcement Learning (RL) methods to solve the game of Snake. We propose two different algorithms and different approaches for modeling the underlying Markov Decision Process (MDP). We also use various reward-shaping approaches to enhance the performance of the algorithm.

1 Introduction

Snake is a well-known video game where the player controls a snake moving in a grid. The goal is to eat the apples, and the snake dies if it collides with a wall or runs into itself. In this project, we develop an AI using Reinforcement Learning (RL) to play this game. First of all, it is insightful to note that there are other approaches that solve the game of snake better than RL. Indeed, one can see the snake problem as a path-finding problem and use corresponding methods (e.g. Dijkstra or A* algorithms). Nevertheless, we are interested in seeing how well an AI based on RL can do at this game.

2 The Snake environment

The game environment is a 12*12 square grid. The snake starts with a fixed length, and goes straight, right or left at each step. There is always at least one apple on the grid. Each time the snake eats an apple, another one spawns randomly in one of the available spots on the grid. In this project, we used the gym implementation Sneks [1]. A screenshot of the environment is displayed in Fig. 1.

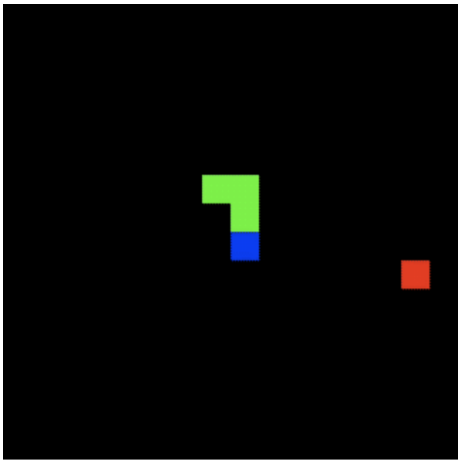

Figure 1: A screenshot of the game environment

3 MDP models

3.1 The reward mechanism

In order to apply RL methods to the game we must specify the Markov Decision Process we are working with. As we will discuss reward shaping in section 4, we will consider the following reward mechanism :

- If the snake eats an apple it gets a reward of $+10$.
- If the snake touches the walls or collides with itself it gets a reward of -40.
- Otherwise it gets a reward of 0.

3.2 State and action spaces

Now let's discuss the state and action spaces. We used three types of state spaces in this project.

- **Raw image** : the state space is the RGB image of the game. The action space is $\mathcal{A}_1 = \{U, R, D, L\}$ for each of the directions. This means that $\mathcal{S} \subset \mathbb{R}^{144}$.
- **Simple features** : instead of the raw image, the state space is formed by only five numbers s_i : (s_1, s_2) are the coordinates of the head of the snake, (s_3, s_4) are the coordinates of the apple, and s_5 represents the current direction of the snake. This time $\mathcal{S} \subset \mathbb{R}^5$.

- **Rotated image** : the state space is an image deduced by the raw image with a rotation, such that the snake's direction is always north. This allows to reduce the state space's size, as the agent will only see situations where it faces north. This time, the action space can also be reduced to three actions, as the snake can never go south when it faces north.

3.3 The hunger parameter

In order to accelerate training, we used the following method : the game ends if the snake does not eat a single apple in a fixed number of steps. This number of steps is called the *hunger* parameter. This enables to avoid situations where the snake eternally goes into a loop, gaining no reward but never dying. These situations slow down the training a lot as the corresponding episodes are extremely long. Using a hunger parameter allows to avoid very long and boring episodes. However, it is important to note that it makes the problem partially observable. From a state-action pair, it is now impossible to predict accurately the next state (or the true distribution of the next state). Indeed, if the state is a state with maximal hunger, it is in fact a final state.

4 Reward Shaping methods

The simple reward mechanism presented in section 3.1 is an example of a *sparse* reward mechanism. Indeed, during an arbitrary game, the snake only receives immediate feedback when it eats an apple or when it dies. Environments with sparse rewards are harder to solve. The approach we decided to use is reward shaping, i.e. using new rewards that will eventually help the agent learn the original task. To decide on what rewards need to be used, we rely on problem-specific engineering. We came up with two different reward shaping methods, both based on the distance between the agent and the fruit. These methods are inspired from [2].

- **Closer bonus** : The agent gets a positive reward when it gets closer to the apple and a negative one when it gets further from the apple. An interesting property of this method is that it can also be used to avoid loops. For example is the positive reward is +1 and the negative one is -2, the sum of rewards the agent will get when going through a loop will be negative. This way, we can hope the agent will be discouraged to go into loops forever.

- **Differential Distance** Denote by d_{old} the old distance from the agent to the fruit and d_{new} the new distance from the agent to the fruit. Let $\alpha > 0$. We define the reward to be $r_{\text{RS}} = -\alpha(d_{\text{new}} - d_{\text{old}})$. This way, the agent gets rewarded when it gets closer to the apple and punished when it gets further. The parameter α is called the distance bonus.

Denote by r_{RS} the reward defined above. The reward seen by the agent is then $r + r_{\text{RS}}$ with r the original reward defined in section 3.1

5 Algorithms

For training the agents, we decided to try two algorithms. The first one is A2C (Advantage Actor-Critic) algorithm [3]. The second one is PPO (Proximal Policy Optimization) [4] and can be viewed as an ameliorated version of A2C.

5.1 Actor-Critic methods

The two algorithms use two models :

- The actor maps states to a probability distribution over actions. We will denote by $\pi(a|s)$ the probability of taking action a in state s.

- The critic maps states to their value. We will denote by $v(s)$ the estimated value of state s.

The goal of the two algorithms is, given an initial probability distribution over states ρ, is to find a policy π that maximizes the expected discounted return

$$\mathcal{J}(\pi) = \mathbb{E}_{\tau\sim\pi, s_0\sim\rho} R(\tau)$$

Where, for a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$

$$R(\tau) = \sum_{i=0}^{\infty} \gamma^i r(s_i)$$

and $r(s)$ is the reward obtained when in state s. The notation $s_0 \sim \rho$ means that the initial state is drawn from the distribution ρ and the notation $\tau \sim \pi$ means that the trajectory is the one obtained when playing with policy π : in a state s_i, an action a_i is chosen according to the distribution $\pi(.|s_i)$. The next state is then drawn using the transition kernel of the MDP.

The actor and critic are approximated using a neural network that takes as an input the state s and returns two outputs : the value and the probabilities for each action. Before explaining

the difference between the two algorithm let's give more details about the neural network architecture that we used.

5.2 Neural network architecture

- For the "simple feature" state space, the network is a fully connected network with two hidden layers and two heads (one for the actor, with four components, one for the critic with one component).
- In all other cases the state space corresponds to an RGB image. We used a convolutional neural network with two convolutional layers. The final layers (corresponding to the actor and the critic) are fully connected.

In all cases we used sigmoid activations. We did not investigate deeply the role of the network architecture (number and type of hidden layers, activation functions) in this project.

5.2.1 A2C

Denote by π_θ the policy of the network with parameters θ. The Policy Gradients theorem shows that the function $\mathcal{J}(\pi_\theta)$ defined above is differentiable w.r.t θ and that :

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi,\sim_\nu\sim\rho}\left(\sum_{i=0}^{T}\log(\pi(a_i|s_i))R(\tau)\right)$$

where T is the length of the episode. The idea behind A2C is to use a variance reduction technique when estimating the expectation. First, one can replace $R(\tau)$ with the discounted reward from state i defined by

$$G_i = \sum_{j=i}^{T}\gamma^j r_j.$$

Then, for reducing variance, denoting by $V^\pi(s)$ the (real) value of state s, defined by

$$V^\pi(s) = \mathbb{E}_{s_0=s,\tau\sim\pi}(R(\tau)),$$

the idea is to substract $V^\pi(s_i)$ to G_i. However as we do not have access to $V^\pi(s_i)$ we have to estimate it using the critic. A2C uses the following policy gradient estimate when a batch of N episode was played under policy π_θ :

$$\widehat{\nabla\mathcal{J}(\pi_\theta)} =$$

$$\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{T_j}\nabla_\theta\log(\pi_\theta(a_i^{(j)}|s_i^{(j)}))(G_i - v_\theta(s_i))$$

The critic minimizes the L^2 loss

$$l_c = \frac{1}{N}\sum_{i=1}^{T_j}(G_i - v_\theta(s_i)^2$$

The optimization process follows immediately : until convergence, the following steps are repeated.

1. Play a batch of N episodes with policy π_θ.
2. Compute the estimated policy gradient as well as the critic loss.
3. Do one step of gradient ascent (for the actor) and gradient descent (for the critic).

A2C is a performing algorithm that works in a wide variety of situations. However, it has the property of being *sample-inefficient*. Indeed, for each batch of episodes played, only one step of gradient descent is made. Once the parameters are updated, the actor loss does not correspond anymore to the policy gradient. This problem is tackled by PPO with an approach relying on using a surrogate objective.

5.2.2 PPO

The PPO algorithm differs from A2C as it enables, for each batch of episodes played, to do multiple steps of gradient descent. The idea is to use importance sampling. As this technique comes at the cost of high variance, a surrogate objective is used instead of the actor objective. We used the "PPO-clip" variant that uses a clipped objective. Denote by $\hat{\mathbb{E}}_t$ the empirical expectation operator. The actor loss used by PPO is defined as

$$L_C(\theta) = \hat{\mathbb{E}}_t(\min(r_t(\theta)\hat{A}_t, \text{clip}(1-\epsilon, 1+\epsilon, r_t(\theta))\hat{A}_t))$$

Where :

- $\hat{A}_t$ is the estimated advantage at time t : $\hat{A}_t = G_t - v_{\theta_{old}}(s_t)$.
- $r_t(\theta)$ is the ratio of probability of actions between new and old policy :

  $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$
- clip means the ratio is clipped in the interval $[1-\epsilon, 1+\epsilon]$ where ϵ is a parameter.

Here is the corresponding repeated training process.

1. Collect a batch of N episodes playing with policy $\pi_\theta = \pi_{old}$.

2. Do multiple steps of gradient descent using the clipped loss mentioned above.

The idea behind the clipped objective is to avoid encouraging the new policy to be too far from the old one (this happens when the ratio of probabilities is outside of the range $[1-\epsilon, 1+\epsilon]$).

5.2.3 Specificities of the implementation

In addition to the algorithms described above we made some adaptations.

- Regarding the network implementation, weights are partially shared between the actor and the critic. We could have used two different networks for each one but using only one has the benefit of having to learn low-level image features only once.
- KL divergence early stopping (PPO) : Although the algorithm avoids encouraging the new policy to be too far from the old one, it is possible that this happens nonetheless. To enforce this to never happen, we compute the KL divergence between the old and new policy and stop training (and move to the next step) if it goes beyond a fixed threshold.
- Entropy bonus (PPO and A2C): In order to encourage exploration, an entropy bonus can be added to the loss, it is defined by $H = -\hat{\mathbb{E}}_t(\sum_{j=1}^{4} \pi_\theta(a_j|s_t) \log(\pi_\theta(a_j|s_t))$.

6 Experiments

6.1 Choice of algorithm

In this experiment (Fig. 2, 3) we compare the two algorithms PPO and A2C. The state space is the raw RGB image, and no reward shaping was used. The hunger parameter is set to 25. Both A2C and PPO were trained using the same parameters. No entropy bonus is used.

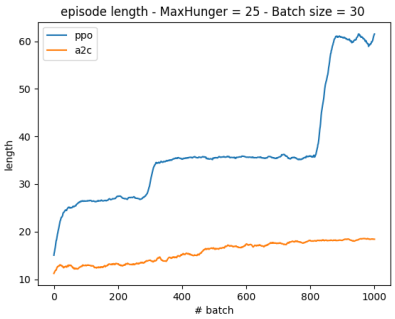

Figure 2: Mean Episode Length during traning for PPO and A2C

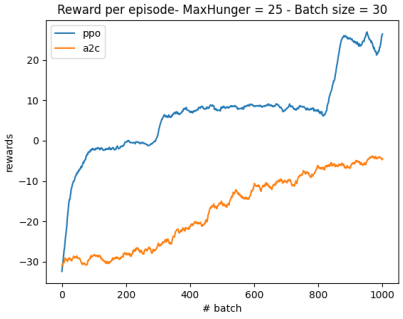

Figure 3: Mean Episode Rewards during traning for PPO and A2C

6.2 Choice of state space

In this experiment (Fig. 4, 5), we compare, using PPO and without reward shaping, the different approaches for the state space as described in Section 3.2.

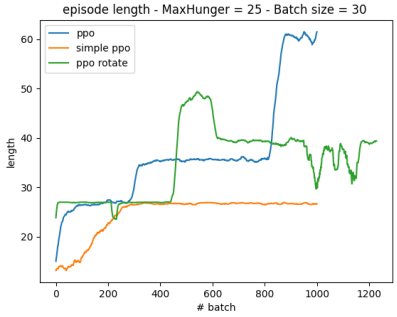

Figure 4: Mean Episode Length during traning for different state spaces

Figure 5: Mean Episode Rewards during traning for different state spaces

6.3 Reward Shaping

In this experiment (Fig. 6, 7), using PPO with the original RGB state, we compare the different reward shaping options described in section 4.

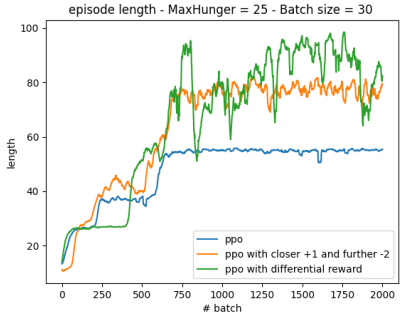

Figure 6: Mean Episode Length during traning for different reward shaping methods

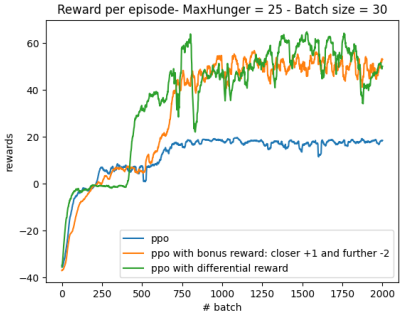

Figure 7: Mean Episode Rewards during traning for different reward shaping methods

7 Discussion

7.1 Algorithms comparison

As shown in figures 2, 3, PPO performs significantly better than A2C, both on the task of maximizing rewards and on the task of maximizing length (surviving). The first plateau, corresponding to a reward of zero (the agent has learned to avoid walls but not to look for the fruit : the episode stops when it reaches the maximum hunger), is reached after 80 iterations by PPO, and only after 1000 iterations by A2C.

7.1.1 State space

The results of figures 4, 5 show several information:

- The simple state space does not allow to make enough progress. The information in this space might be insufficient to derive a good policy.
- The state space with rotations appears to learn faster at first, but seems to suffer from instability. This might be due to the fact that, in this MDP, the transitions between states are highly non-continuous : when the Snake turns left or right, the next state it sees is very far (in terms for example of L^2 norm) from the original state.

7.1.2 Reward shaping

Figures 6, 7 show the comparison between our reward shaping methods. It seems that the reward based on punishing going further from the apple and rewarding getting closer to the apple and the one based on the differential distance to the fruit do improve the performance of the agent. However, the reward shaping based on the differential distance seems to be less stable than the bonus reward.

Conclusion and future work

In this project, we successfully applied Reinforcement Learning to play the game Snake. Our algorithm, based on PPO, manages to learn in the environment. A reward shaping method also enables faster training. However, with our limited computational resources, we could not manage to get the AI to play optimally. The main issues are :

- *Partial observability* induced by the hunger parameter. This parameter is indeed useful for training efficiently but makes the MDP a POMDP. We could investigate methods that produce a similar effect while keeping a MDP (e.g. incorporating the hunger information in the state).
- *Unsafe exploration.* The agent often gets stuck in a situation where the fruit is at the edge of the board. As it gets closer to the fruit (exploration) it often finds a wall, which gives the agent a highly negative reward. In order to make progress, the exploration need to be safe, in the sense that it should avoid at all costs states with a very low reward. This problem has been widely discussed in the literature [5] and could be investigated in this case.

References

[1] Sneks gym environment. https://github.com/nicomon24/Sneks.

[2] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[3] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[5] Javier García, Fern, and o Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015.