Project for RL

Written by Andrei Matraguna

Table of Contents

1.Environment: Private eye – V0

Private eye is a video game produced and published by Activision for the Atari 2600 video game system. In Private eye players assume the role of a private investigator who has been assigned the task of capturing the criminal mastermind. In this game you need to navigate the city streets, dead-ends and one-ways in search of the criminal(Henri Le Fiend and his gang) [1][2].

In "Private Eye" (Figure 1), players are rewarded with points for each successful capture of a criminal and for returning the stolen loot to the detective's car. Additionally, players can earn bonus points for completing a level within a specific time limit or for uncovering bonus items hidden within the game world[2].

The game also features various power-ups that can aid the detective in their quest. These power-ups include a bulletproof vest, which grants temporary invincibility, and a jetpack, which allows the detective to quickly traverse the city blocks[2][3].

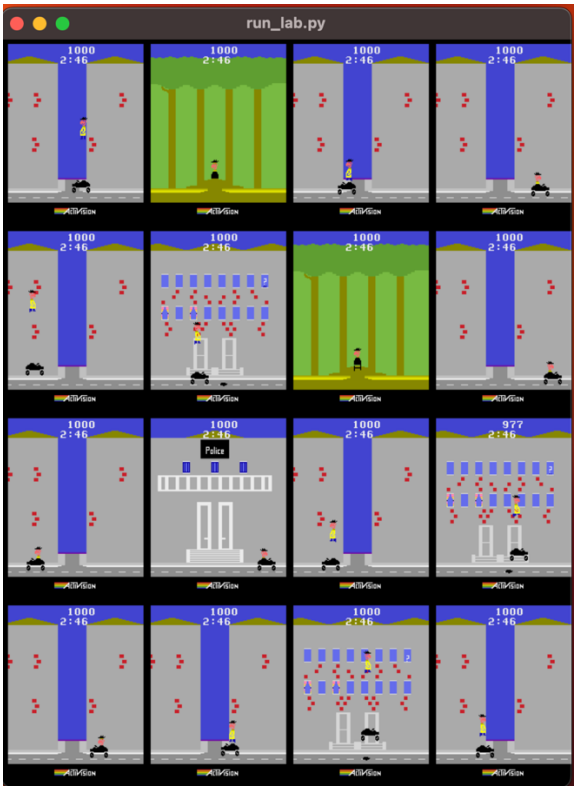

Figure 1. Atari game Private Eye

The game features multiple levels, each with an increasing level of difficulty. As players progress, the neighborhoods become more complex, requiring a combination of observation, strategy, and quick reflexes to succeed.

"Private Eye" is known for its challenging gameplay, engaging detective theme, and distinctive visual style. It provided players with a unique and immersive experience for its time and remains a beloved title among fans of classic Atari games.

In this papper all algorithms were trained in 15 processes of this game (etch algorithm run for 500 000 steps).

2.Algorithms

2.1 Reinforce

The Reinforce algorithm utilizes policy gradients to optimize the agent's policy directly (Figure2). It samples trajectories, computes the total rewards, and updates the policy parameters to maximize expected rewards.

$$Q^{\pi_\theta}(s_t, a_t) = v_t$$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
Initialise θ arbitrarily
for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
for $t = 1$ to $T - 1$ **do**
$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
end for
end for
return θ
end function

Figure 2.1.1 Reinforce algorithm

Network: The network architecture is a ConvNet. It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration:
1. 32 filters, 8 kernelsize, 4 stride, 0 padding, 1 dilation
2. 64 filters, 4 kernelsize, 2 stride, 0 padding, 1 dilation
3. 32 filters, 3 kernelsize, 1 stride, 0 padding, 1 dilation

The fully connected layer has 512 units. The activation function used is ReLU. Normalization is enabled and batch normalization is disabled. The weights are initialized using orthogonal initialization. Gradient clipping is set to 0.5. The loss function used is Mean Squared Error. The optimizer is RMSprop with a learning rate of 0.002. The learning rate is constant throughout, specified by the null learning rate scheduler.

Results:

This algorithm remained in a very good range obtaining a minimum score of 1581 and a maximum score of 3034 Figure(2.1.2). He was defeated on the last hundred meters by the SARSA algorithm.

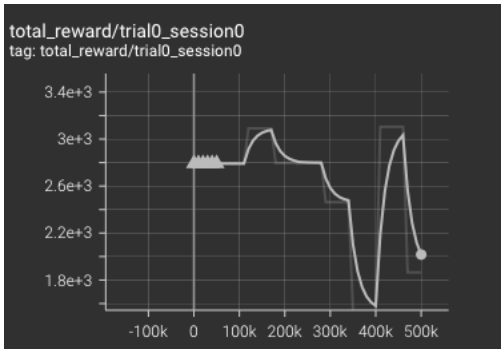

Figure 2.1.2 Results for Reinforce algorithm total reward

2.2 SARSA

The algorithm is SARSA (Figure 2.2.1). The action probability type is argmax and the action policy is epsilon-greedy with linear decay of the exploration variable specification (epsilon is initialized to 0.8 and annealed to 0.1 between time steps 5,000 and 100,000) Gamma (discount factor) is set to 0.99. The network will be trained every 32 steps because of the training_frequency parameter.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 2.2.1 Sarsa algorithm

Network: The network architecture is a ConvNet. It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration:
1. 32filters,8kernelsize,4stride,0padding,1dilation
2. 64filters,4kernelsize,2stride,0padding,1dilation
3. 32filters,3kernelsize,1stride,0padding,1dilation

The fully connected layer has 512 units. The activation function used is ReLU. The weights are initialized using orthogonal initialization. Normalization is enabled and batch normalization is disabled. The same optimizer is used throughout training. Gradient clipping is set to 0.5. The loss function used is Mean Squared Error. The optimizer is RMSprop with a learning rate of 0.01. The learning rate is constant throughout, specified by the null learning rate scheduler.

Results

At first this algorithm didn't seem like it would have a promising result, but apparently this one did the best (Figure 2.2.2).

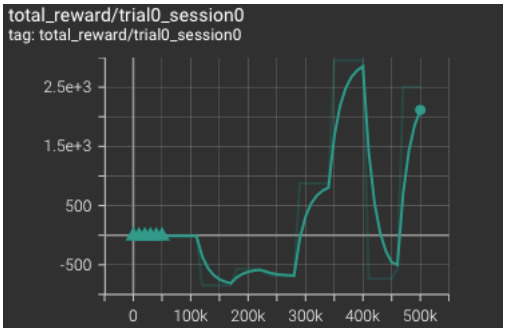

Figure 2.2.2 Results for SARSA (total reward)

2.3 PPO

The algorithm is Proximal Policy Optimization(figure 2.3.1). The action policy is the default policy for discrete action space (Categorical distribution). Gamma (discount factor) is set to 0.99. GAE is used to estimate advantages, with lam value set to 0.70. The clipping hyperparameter epsilon and its decay are specified, and the entropy coefficient is similarly specified. The value loss coefficient is set to 0.5.

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

Figure 2.3.1 PPO algorithm

The network architecture is a ConvNet (line: 35). It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration (line: 37-41):
1. 32filters,8kernelsize,4stride,0padding,1dilation
2. 64filters,4kernelsize,2stride,0padding,1dilation
3. 32filters,3kernelsize,1stride,0padding,1dilation

The fully connected layer has 512 units. The activation function between layers is ReLU. The weights are initialized using orthogonal initialization. Normalization is enabled and batch normalization is disabled. Gradient clipping is set to 0.5. The same optimizer is used differently during training. The loss function used is Mean Squared Error. The actor and critic use a shared network. The optimizer is Adam, with a learning rate of 0.00025. The learning rate has decayed to 0 over 10 million frames.

Results

Although it had a promising start, the algorithm failed to learn to play this game (Figure 2.3.2).

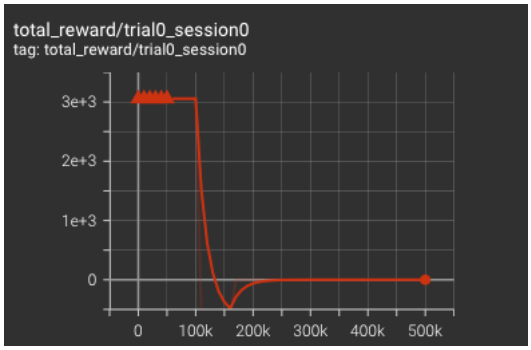

Figure 2.3.2 Results from PPO algorithm

2.4 A2C

The algorithm is Actor-Critic (figure 2.4.1). The action policy is the default policy for discrete action space (Categorical distribution). Gamma (discount factor) is set to 0.99. The lambda parameter value is set to 0.95. The entropy coefficient and its decay during training are also specified. The value loss coefficient is set to 0.5.

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Figure 2.4.1 A2C algorithm

Network

The network architecture is a ConvNet. It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration (line: 29-33):
1. 32filters,8kernelsize,4stride,0padding,1dilation
2. 64filters,4kernelsize,2stride,0padding,1dilation
3. 32filters,3kernelsize,1stride,0padding,1dilation

The fully connected layer has 512 units. The activation function between layers is ReLU. The weights are initialized using orthogonal initialization. Normalization is enabled and batch normalization is disabled. Gradient clipping is set to 0.5. The same optimizer is used differently during training. The loss function used is Mean Squared Error. The optimizer is RMSprop with a learning rate of 0.0007. If separated networks are used instead, it is possible to specify a different optimizer setting for the critic network, by setting use_same_optim to false. Since the network is shared in this case, it is not used. There is no learning rate decay.

Results:

Although at the beginning this algorithm managed to obtain a positive score, it failed to "understand" how to play this game, the final score being the lowest (figure 2.4.2).

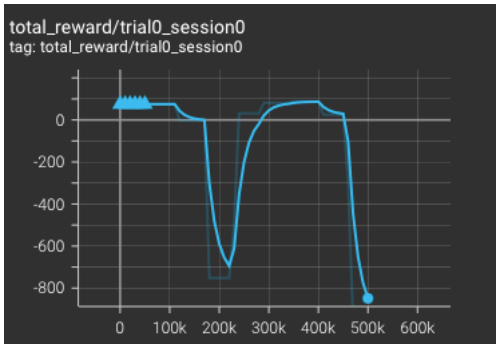

Figure 2.4.2 A2C total reward

2.5 A3C

The algorithm used is Asynchronous Advantage Actor-Critic. The action policy and exploration variance specifications are set to default. Gamma (discount factor) is set to 0.99. The lambda parameter (lam) for GAE is set to 0.95. The entropy coefficient is specified. The value loss coefficient is set to 0.5.

Algorithm 1 A3C

Initialize actor and critic networks with random weights
for each $episode \in [1, n]$ **do**
    Download weights from the headquarters to each AC
    for each AC **do**
        Initialize the random state s_0
        for each $t \in [1, k]$ **do**
            Select actiona_t from the actor network
            Execute action a_t and observe reward r_t from the critic network and next state s_t
            Update the actor network parameters
        end for
        Update the critic network parameters
    end for
    Upload weights of each AC network to the headquarters
end for

Figure 2.5.1 A3C algorithm

Network:

The network architecture is a ConvNet. It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration:

1. 32filters,8kernelsize,4stride,0padding,1dilation
2. 64filters,4kernelsize,2stride,0padding,1dilation
3. 32filters,3kernelsize,1stride,0padding,1dilation

The fully connected layer has 512 units. The activation function between layers is ReLU. The weights are initialized using orthogonal initialization. Normalization is enabled and batch normalization is disabled. Gradient clipping is set to 0.5. The same optimizer is used differently during training. The loss function used is Mean Squared Error. The optimizer is GlobalAdam with a learning rate of 0.0001. If separated networks are used instead, it is possible to specify a different optimizer setting for the critic network. There is no learning rate deca

Results:

Although it started with a score close to 0, this algorithm had 2 learning curves. In the first, he managed to go from a negative score to a positive score. And in the second he managed to triple his score from 900 to 2700 (figure 2.6.2).

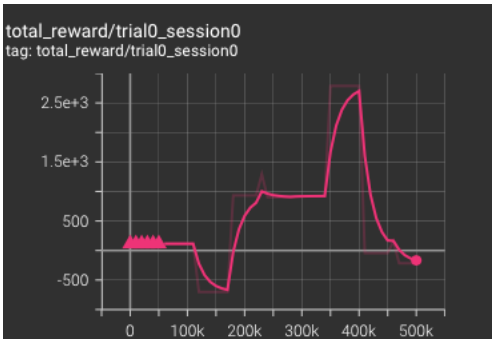

Figure 2.5.2 A3C total reward

2.6 DQN

The algorithm is DQN(figure 2.6.1) . The action probability type is argmax and the action policy is epsilon-greedy with linear decay of the exploration variable specification (epsilon is initialized to 1.0 and annealed to 0.01 between time steps 10,000 and 1,000,000). Gamma (discount factor) is set to 0.99.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 2.6.1 DQN Algorithm

Network:

The network architecture is a ConvNet. It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration:

1. 32filters,8kernelsize,4stride,0padding,1dilation
2. 64filters,4kernelsize,2stride,0padding,1dilation
3. 32filters,3kernelsize,1stride,0padding,1dilation

The fully connected layer has 512 units. The activation function used is ReLU. The weights are initialized using orthogonal initialization. Normalization is enabled and batch normalization is disabled. Gradient clipping is set to 10.0. The loss function used is Smooth L1, a combination of MSE and MAE. The optimizer is RMSprop with a learning rate of 0.0001. The learning rate is constant throughout, specified by the null learning rate scheduler. The parameters are completely replaced with the updated ones after 1,000 steps.

Results:

This algorithm has not learned how to play this game. However, he managed to stay at a positive score.

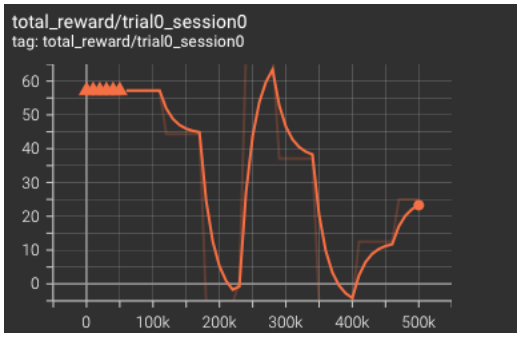

Figure 2.6.2 DQN total reward

2.7 DDQN

The algorithm is DDQN (figure 2.7.1). The action probability type is argmax and the action policy is epsilon-greedy with linear decay of the exploration variable specification (epsilon is initialized to 1.0 and annealed to 0.01 between time steps 10,000 and 1,000,000). Gamma (discount factor) is set to 0.99.

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\, Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

Figure 2.7.1 DDQN Algorithm

The network architecture is a ConvNet. It consists of convolutional layers and a fully connected layer. The convolutional layers have the following configuration:
1. 32filters,8kernelsize,4stride,0padding,1dilation
2. 64filters,4kernelsize,2stride,0padding,1dilation
3. 32filters,3kernelsize,1stride,0padding,1dilation

The fully connected layer has 512 units. The activation function used is ReLU. The weights are initialized using orthogonal initialization. batch normalization is disabled. Gradient clipping is set to 10.0. The loss function used is Mean Squared Error. The optimizer is RMSprop with a learning rate of 0.0001. The learning rate is constant throughout, specified by the null learning rate scheduler. The parameters are completely replaced with the updated ones after 1,000 steps.

Results

This algorithm had a fairly good learning curve, but eventually it started to get lower scores (figure 2.7.2).

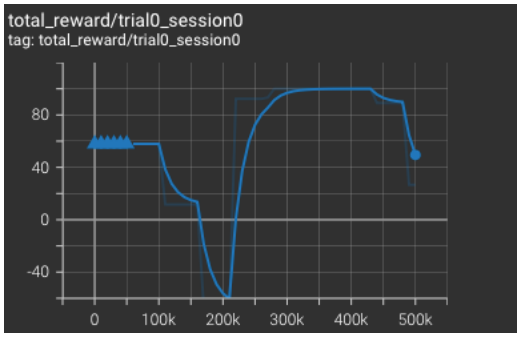

Figure 2.7.2 DDQN total reward

3. Conclusion

With the same type of network, I can say that the reinforce algorithm (2.1) did the best but was defeated by SARSA. This statement can be confirmed by figure 3.1 and the attached ".json" configuration files. All graphs and ".json" files can be seen on this link

Figure 3.1 All results, total reward, in the same picture

4.Bibliography

Huang, Shengyi; Dossa, Rousslan Fernand Julien; Raffin, Antonin; Kanervisto, Anssi; Wang, Weixun "**The 37 Implementation Details of Proximal Policy Optimization"**

AtariAge Private Eye - Activision - Atari 2600

Wikipedia Private Eye