CS4649-7649 PS7
Experimental results

Arnaud Klipfel

October 24, 2020

Contents

Methodology

1.1 General remarks

In all three algorithms only one hidden layer was considered. It is a first good try in most deep RL applications, and here since the pong game is not really complicated it works well.

The architectures of the neural networks have been chosen identical in AC and A2C, for the actor, and the critic(s).

The size of the τ buffer has always been 100 for the tests and the replay buffer for the Q network had also a size of 100. The gradient updates have been normalized by the size of the data.

Biases have been set to 0 at the creation of the neural networks for AC and A2C, and not for Reinforce.

The learning rate of the policy network has always been set to 0 once the training was considered over, to see if the neural network had overfitted, and how well it adapted to new data. As the results show, the given models did not overfit.

1.2 Reinforce

The optimal hyperparameters found for the Reinforce algorithm are presented in figure 1. The parameters were chosen to have a sharp increase right from the start. The actor has to learn and to adapt to its environment. It is also possible to use 8, 16, 32 nodes in the hidden layer. With 32 nodes the neural network reached a higher score in fewer iterations.

1.3 AC

The hardest part was to find the learning rates for the different neural networks. The basic idea was to have a high learning rate for the policy network, for the actor, such that it could learn fast and adapt to its environment, without wanting to learn too fast and crashing to a win/loss ratio of 0. The learning rates of the critic was then chosen to avoid any crashing of the win/loss ratio to 0. The optimal hyperparameters found for the AC algorithm are presented in figure 2.

At the start the win/loss ratio drops and then increases again. This is due to the unreliable representation of the environment, through the Q function that has to be learned first. In the Reinforce algorithm (figure 1) this initial decrease does not appear since the estimation of the Q-function is already an acceptable one.

Once the policy had reached its highest score, it has always happened that the win/loss ratio started to decrease sharply. Once the policy has learned to play the game it is not necessary to keep the initial learning rate, since it may in fact have the opposite effect, to ”un-learn”. This is due also to the Q network learning simultaneously the environment, which introduces instability in the policy update.

A decay of the policy learning rate was used, as described in figure 2, and was shown to have good results. α_Q was not decayed. A decay is done after the policy reaches its highest score, since it would have dropped a few iterations after. The learning rate is then set to 0 progressively.

1.4 A2C

The optimal hyperparameters found for A2C are presented in figure 3. Adding another critic does not change the process presented in part 1.3 , and the observations made for AC are actually also valid for A2C.

Results

The plots show the changes in the parameters (green) and win/loss ratio or score for the policy network (red) in function of the number of iterations. All gradient updates have been normalized by the size of the data.

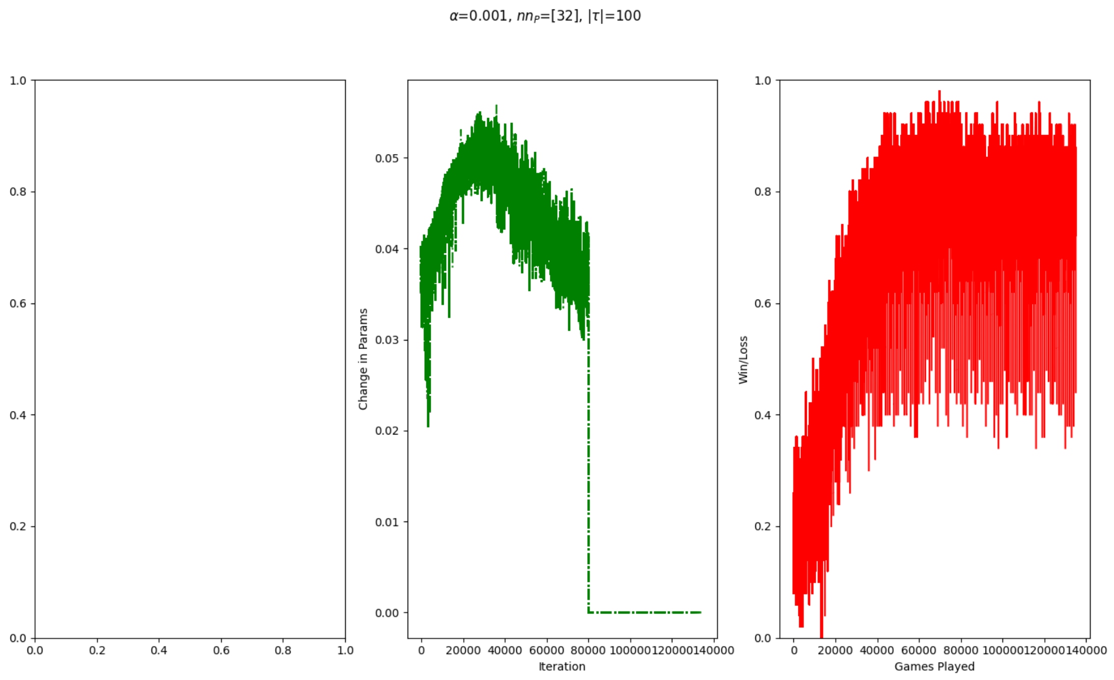

Figure 1: Reinforce for $\alpha = 10^{-3}$, 1 hidden layer, 32 nodes and a buffer of size 100 for one episode. The learning rate was set to 0 after 80,000 iterations.

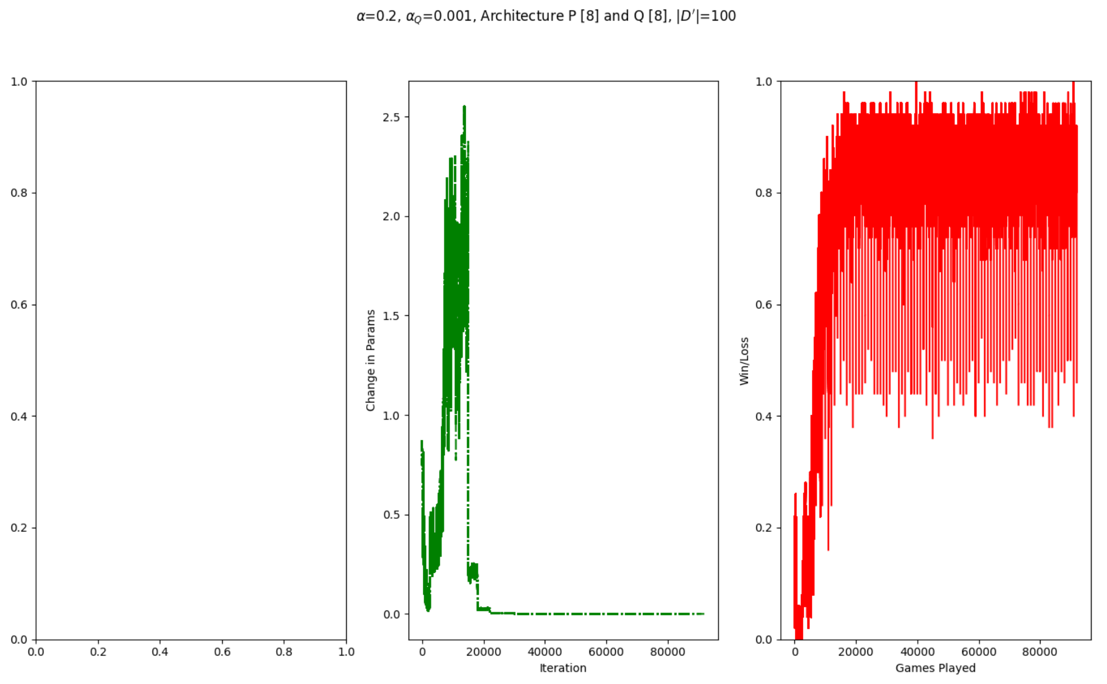

Figure 2: AC for $\alpha = 2e-1, \alpha_Q = 1e-3$, 1 hidden layer, 8 nodes for all networks, a τ buffer of size 100 for one episode and for the policy network, and $|D'| = 100$ for the Q network. α was divided by 10 at 15,000 iterations, by 10 at 18,000, and 22,000, and set to 0 at 30,000 iterations.

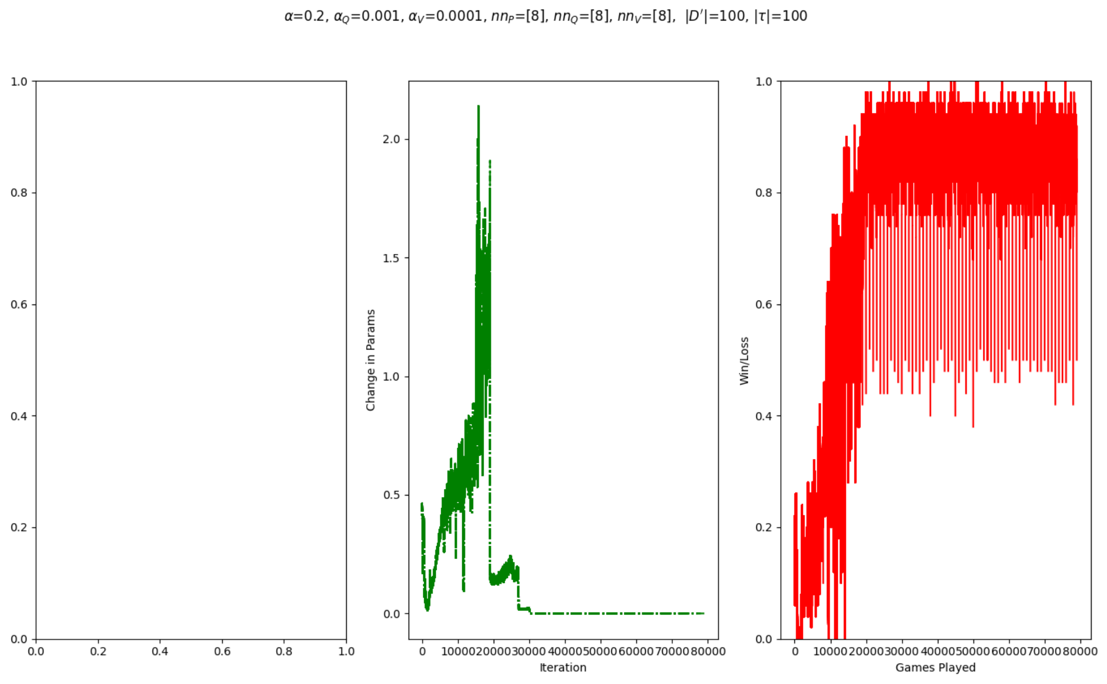

Figure 3: A2C for $\alpha = 2e-1, \alpha_Q = 1e-3, \alpha_V = 1e-4$, 1 hidden layer, 8 nodes for all networks, a τ buffer of size 100 for one episode and for the policy network and the value network, and $|D'| = 100$ for the Q network. α was divided by 10 at 19,000 iterations, by 10 at 27,000, and set to 0 at 30,000 iterations.

Comparison

The numerical values shown in the following table are experimental and have some uncertainty.

Algorithm	speed evaluation (in number of iterations)	final interval score (after $\alpha = 0$)	mean of the final score (after $\alpha = 0$)
Reinforce	score of 0.8 around 36,000	$[0.5, 0.95]$ drops to 0.4 occasionally	0.79
AC	score of 1 around 40,000	$[0.66, 0.99]$ drops to 0.4 occasionally	0.83
A2C	score of 1 around 26,000	$[0.72, 0.99]$ drops to 0.5 occasionally	0.88

In conclusion, A2C and AC performed better than Reinforce, based on the expreimental results presented here. A2C performed better than AC. AC and A2C can learn the environment with more accuracy, since they can approximate the Q-function and the value function better due to the neural network structure, in comparison to reinforce.