University of Iceland 2018
REI101M - Reinforcement learning

Report for computational assignment 2

Ásta Lára Magnúsdóttir
Hinrik Már Rögnvaldsson
Marín Ingibjörg McGinley
Sigrún Dís Hauksdóttir

Abstract

This is a project in reinforcement learning for creating an agent that learns the name of backgammon both with a neural network and Dyna-2.

1. Code source

https://github.com/astamagg/reikniverkefni2-3

2. Introduction

***************To be continued when the code is working properly.***************

3. Backgammon

Backgammon is a game played by two players. Each player has 15 checkers to move between 24 points. The objective is to be the first to move the 15 checkers of the board. A point occupied by more than one opponent's checker is blocked, preventing the opponent from placing a checker on that particular point. However, if a point is occupied by exactly one of the players checker it can be taken by the opponent and placed on the middle bar of the board. As long as a player has checkers on the bar, he cannot move any other checkers. To free them, he must play a die which allows him to reach a point of the opponent's home board that is not blocked.[1] Therefore to play the game well the player needs to know the state of the board and what moves would increase his chances of winning.

Because Backgammon has 24 points and 30 checkers all in all the number of states for the game are astronomical. It would therefore not be feasible to store all the states for table lookup purposes. Knowing the number of states and the complexity of the game creates a need for a more efficient method. Creating a neural networks representing the policy for the after-state would enable the player to learn faster by updating the policy as the agent travels through the game, increasing the likelihood of choosing actions that return a favorable result.

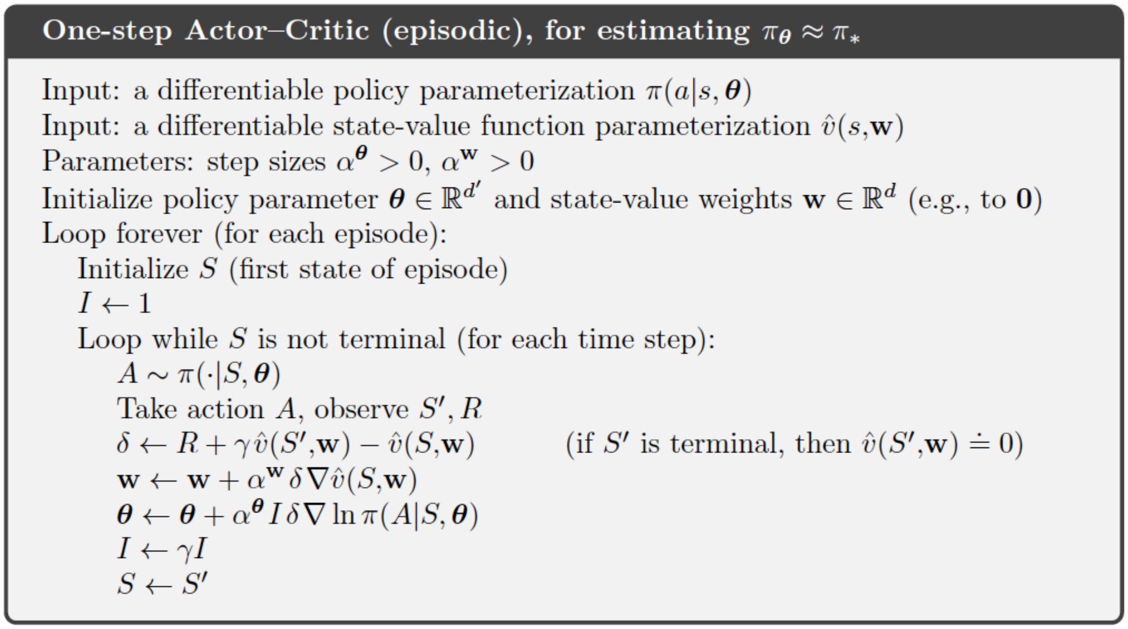
4. Methodology

4.1. Policy Iteration

The neural network has an input of one-hot-encoding of all the possible boards after the agent has rolled the dice. The neural network has one hidden layer. The neural network will have a single input, that is the state of the game of every possible board after the rolling of the dice for the agent. The board will be encoded with one-hot-encoding to reduce the calculation time of the program. The board has 28x28 hidden layers and two outputs. The first one is to upgrade the value function. The output enables us to calculate the loss, in this case the mean square error

[1] Assignment instructions

between the expected output value and actual output value. The other output returns the the likelihood of a certain move in the game. By backpropagation when the game end it is possible to update the gradient toward better odds of choosing a move that lead to victory again next time. We want to use the actor critic method displayed in the pseudo code below as our policy iterator. This pseudo code is taken from the books.



One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\,\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\,\delta\nabla\ln\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

The actor would use the values computed by the part of the neural network responsible for the likelihood of choosing actors and the critic would update using the value-function.

4.2. Dyna-2

***************To be continued after second hand-in.***************

5. Case study

We are having problems computing the actor backpropagation and therefore we weren’t able to test that part of the program. The code does not learn the value function properly, it isn‘t learning in a way that we would like it to. However the code runs and the value function, w1, w2, b1 and b2 updates its values, but obviously their values are incorrect since the agent is not learning and winning the random player as frequently as we suspect. Furthermore we have an error in our policy gradient function, we have a problem with calculating the dot product for the backpropagation because we have a 3D array but the method we found only allows one dimensional calculations.

6. Discussion and conclusion

***************To be continued when the code is working properly.***************

Acknowledgement