Policy Gradient Methods and the Frozen Lake Problem

Sai Krishnan Chandrasekar
Graduate Student, Robotics
University of Pennsylvania
Email: saikrish@seas.upenn.edu

***Abstract**: Reinforcement learning is an area of machine learning, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is used to build systems that improve with experience. The systems are varied in nature. Chess playing systems, humanoid robots etc., are all systems that can be improved with the use of reinforcement learning. In this project, the Frozen Lake problem was modeled as a stochastic Markov Decision Process (MDP) and was solved using Value Iteration, Policy Iteration and Policy Gradient Optimization Methods.*

1 Introduction

The goal of this project was to solve the Frozen Lake Problem by modeling it as an MDP. Reinforcement Learning techniques were used to solve this problem. The three methods used were: Value Iteration, Policy Iteration and Policy Gradient Optimization.

1.1 The Frozen Lake Problem

The problem is as follows: The world is a Frozen Lake. It consists of a $4*4$ or $8*8$ discretized grid. The objective is to get from a start position to the goal position. The catch however is that not all grid cells are frozen. Some of them have melted and have formed holes, which must be avoided. The objective is to get from the start state to the goal state by avoiding the holes.

This is achieved by assigning a reward of $+1$ to the goal state and a reward of 0 everywhere else. The objective of the agent is to maximize the reward. The possible actions the agent can take are: Left, Down, Right and Up. However, the world is stochastic (The agent slips on the ice). There's a 0.8 probability of heading in the direction of the action. There's a 0.1 probability of heading in the two directions perpendicular to the action. For example: If the action was UP, there's a 0.8 probability that the agent will move UP, a 0.1 probability that the agent will move to the LEFT and a 0.1 probability that the agent will move to the RIGHT. If the agent cannot move in a direction due to the wall, it will continue to exist in the same state. If the agent falls into a hole, the game ends. If the agent reaches the goal, the game ends.

A $4*4$ version of the world is indicated below. S indicates the Start location, F indicates Frozen (safe to traverse), H indicates Hole (must be avoided) and G indicates the Goal location.

$$\begin{matrix} S & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & F & G \end{matrix}$$

1.2 Terminology:

Some terminology that will used in this paper:

1.2.1 State:

State are tokens that represent all possible states, for lack of a better word, that the agent could be in. In our case, states are the grid cells the agent can possibly be in. For a $4*4$ map, the possible states run from 0 to 15. For a $8*8$ map, the possible states run from 0 to 63. Denoted by **S**.

The states could also be something like, whether the goal has been attained or not, but in this case, the locations were chosen as it simplified the problem.

1.2.2 Action:

Actions are simply the possible things the agent at a given state can do. In our case, the actions are the directions of motion: LEFT, DOWN, RIGHT and UP. Denoted by **A**.

1.2.3 Transition Matrix or Model:

The transition Matrix or Model is a function of three variables, a state **S**, an action **A** and another state **S'**. The transition matrix consists of the probability of being in a state **S'**, given the previous state was **S** and an action **A** was performed. Denoted by **T**.

$$T(S,A,S') = Pr(S'|S,A)$$

1.2.4 Markov Decision Process (MDP):

A Markov Chain is one where each state is conditioned only upon the previous state. In other words, the current state depends only upon the previous state.

A Markov Decision Process is a Markov Chain whose transitions from one state to the next are controlled.

1.2.5 Reward:

Reward is simply a scalar value that the agent receives for being in that particular state. Rewards are instantaneous and are confined to that particular state. Contrary to common english, in the domain of Reinforcement Learning, a reward can be both positive and negative. In our world, the rewards are only non negative. The agent receives a reward of +1 if it reaches the goal and a reward of 0 everywhere. The goal of any RL problem is to maximize the reward. Denoted by **R**.

Rewards can be given based on the actions or the next states as well. A different reward could be assigned for each **S** and **A** pair. A different reward could be assigned for each **S**, **A** and **S'** trio. These are all, however, mathematically equivalent. In our case, a reward is confined to the current state **S**.

1.2.6 Policy:

A solution to an MDP is a policy. Policy is defined as a mapping from a state **S** to an action **A**. In other words, a policy tells the agent what action to take at a given state. Denoted by π.

The optimal policy π^*, denotes the policy that maximizes the long term expected reward.

1.2.7 Value:

This is a function of the state **S** that estimates how good (in terms of expected return) it is to be in **S** at time t and follow actions from a given policy π. Denoted by $V_t^{\pi}(S)$.

1.2.8 Horizon:

This is essentially the duration within which the agent should reach the goal. A finite horizon specifies the number of timesteps or iterations after which the game will end. An infinite horizon indicates the game goes on forever. It is interesting to think about how the actions might change if we increase the number of horizons.

1.2.9 Discount Factor:

The discount factor is introduced to solve the infinite horizon problem with a finite reward. Denoted by γ. If $\gamma = 0$, the agent is myopic, i.e., maximizes only immediate rewards. If $\gamma < 1$ and $R(S,A)$ is bounded, then the infinite horizon problem has a finite reward.

2 Problem Formulation

The problem at hand was divided into 2 parts:

1. Value Iteration
2. Policy Iteration
3. Policy Gradient Optimization

3 Value Iteration:

The objective of value iteration is to compute an ideal Value function V^* that maximizes the long term expected reward. This is achieved by following the Bellman equation:

$$V(S) = R(S) + \gamma \max_a \sum_{S'} (T(S,A,S')U(S')$$

This process is repeated until convergence of Values. This essentially improves the value by choosing those (S,A) pairs that maximize the reward and hence the value. The discount factor is added to solve the infinite horizon problem. This results in the agent performing the ideal policy (π^*). The results are shown below.

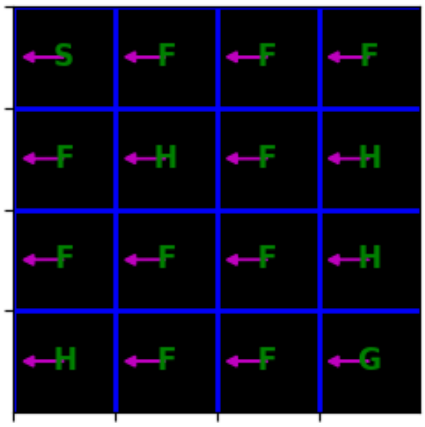

Fig. 1. VI output at iteration 1

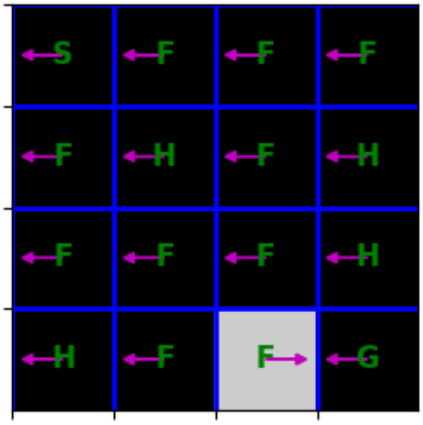

Fig. 2. VI output at iteration 2

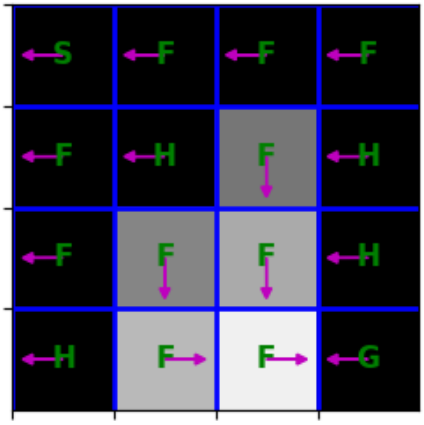

Fig. 4. VI output at iteration 4

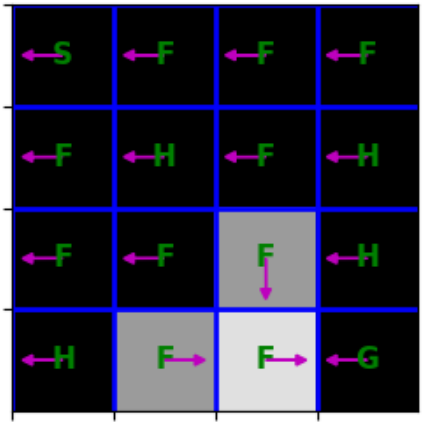

Fig. 3. VI output at iteration 3

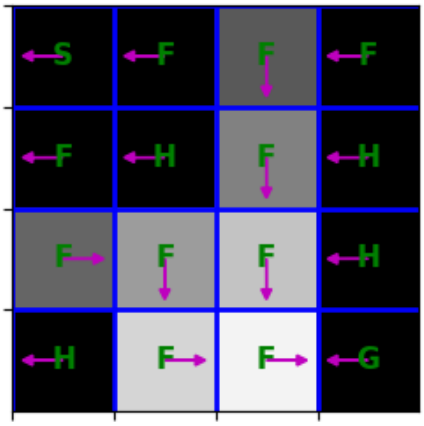

Fig. 5. VI output at iteration 5

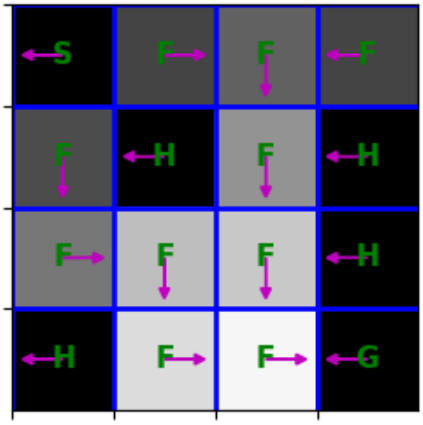

Fig. 6. VI output at iteration 6

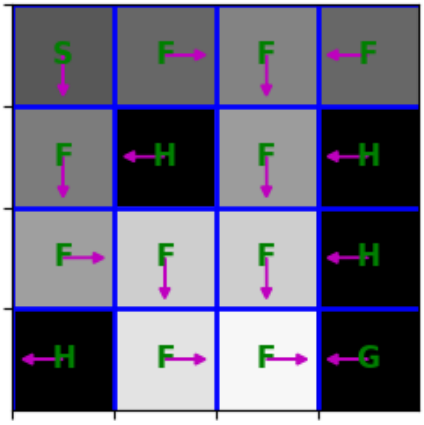

Fig. 8. VI output at iteration 8

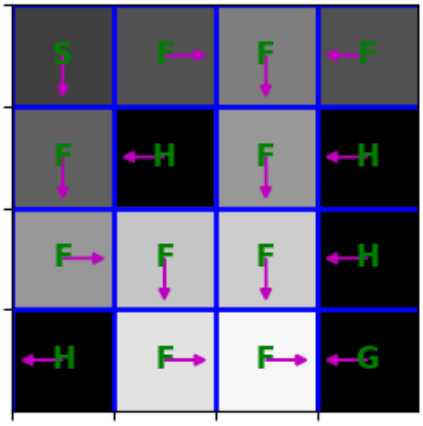

Fig. 7. VI output at iteration 7

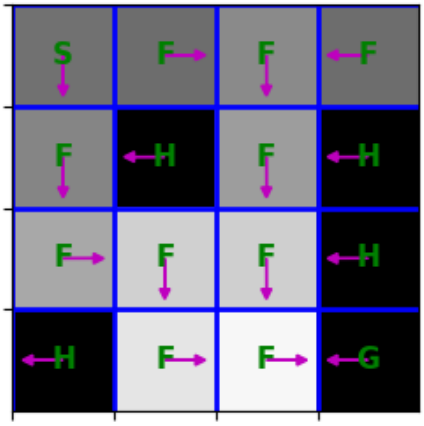

Fig. 9. VI output at iteration 9

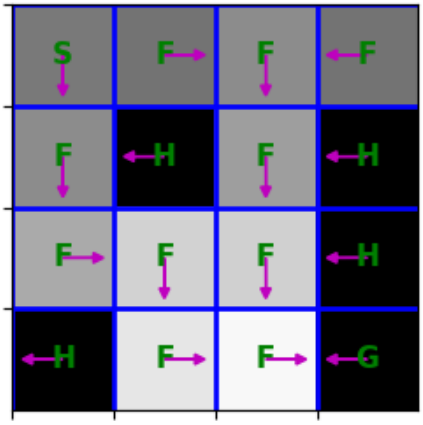

Fig. 10. VI output at iteration 10

The following graph indicates the value at each state at each iteration:

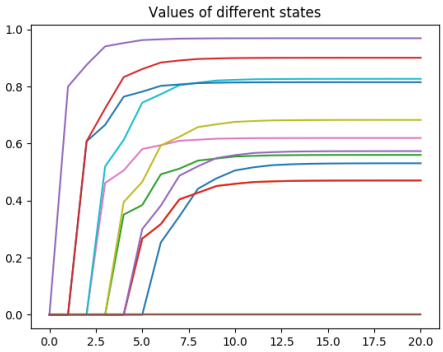

Fig. 11. Values for each state

4 Policy Iteration:

Value iteration deals with a non linear Bellman equation which makes it harder to solve. Even though we have n unknowns and n equations, it's not possible to solve it in linear time, which is why we sort of do a value update step until convergence. There is however, a faster way to get to the optimal policy π^*. That method is known as policy iteration.

Let's take a look at the modified equation:

$$V(S) = R(S) + \gamma \sum_{S'} T(S, \pi_t(S), S') V_t(S')$$

By changing the transition function from $T(S,A,S')$ to $T(S,\pi_t(S),S')$, we eliminate the need for the max function, thus eliminating the non linearity. This new equation is now just a linear system of n equations and n unknowns in the form of $Ax = B$, with x being our values V. Thus, we compute the state-value function for the policy π.

The graphs drawn below indicate the comparison between Value Iteration and Policy Iteration. As can be clearly seen, Policy Iteration converges to the optimal policy much faster than Value Iteration.

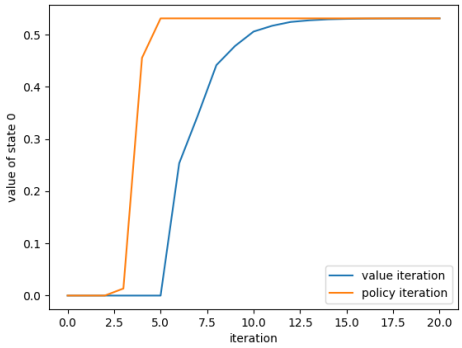

Fig. 12. Comparison between VI and PI

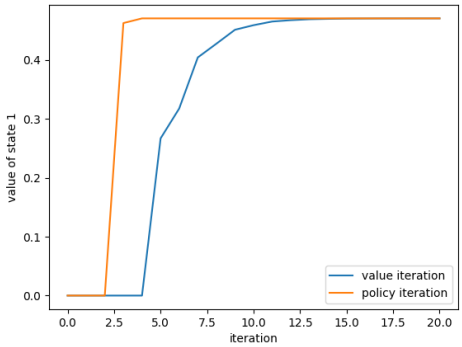

Fig. 13. Comparison between VI and PI

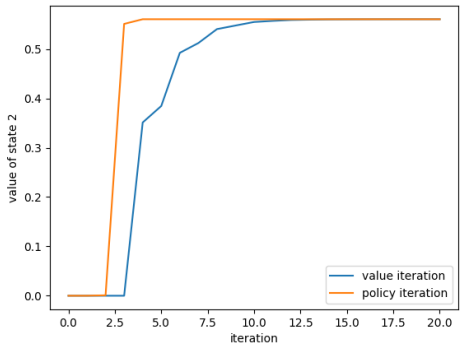

Fig. 14. Comparison between VI and PI

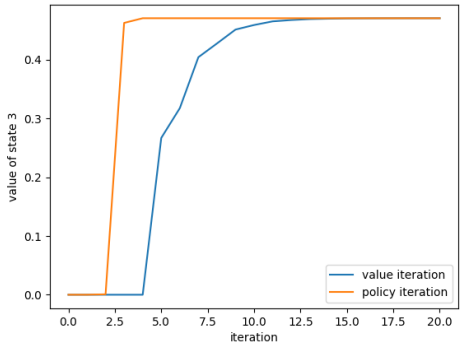

Fig. 15. Comparison between VI and PI

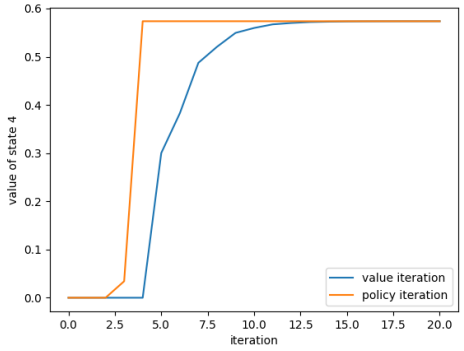

Fig. 16. Comparison between VI and PI

This graph shows the values at different states.

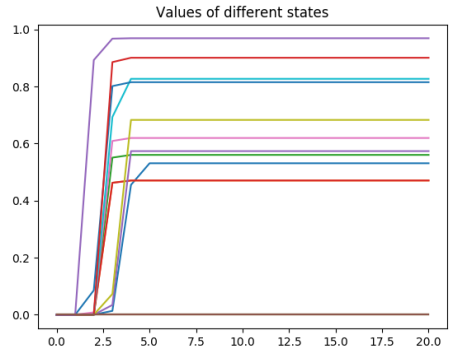

Fig. 17. Values at different states

5 Policy Gradient Optimization:

A vanilla policy gradient method that parametrizes the policy π_θ via parameters θ was implemented. The algorithm computes a gradient estimate $\hat{g} \approx \nabla_\theta \mathbf{E}_t R_t(S_t, \pi(S_t))$.

The algorithm takes gradient ascent steps:

$$\theta \leftarrow \theta + \varepsilon \hat{g}$$

.

The policy for our discrete Frozen Lake MDP was encoded by a matrix parameter $\theta = f_{sa}$. The action probabilities were defined by exponentiating this matrix and then normalizing across the action dimension so that the probabilities add to , i.e. the softmax function.

The policy gradient step function then simply updates the policy parameters. The perplexity of the computed policy was also computed. It was seen that the perplexity went down to 1, corresponding to a deterministic policy. This was attempted on the larger $8 * 8$ map as well. The perplexity went down to 1, but after 2500 iterations of 50 horizons each.

The results are shown below:

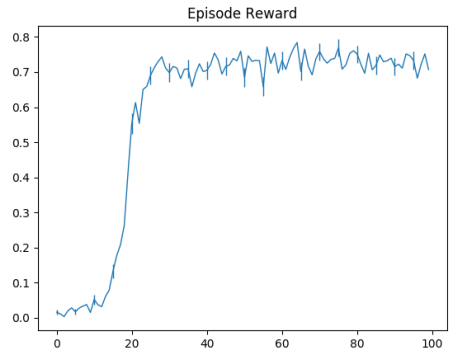

Fig. 18. Episode Reward

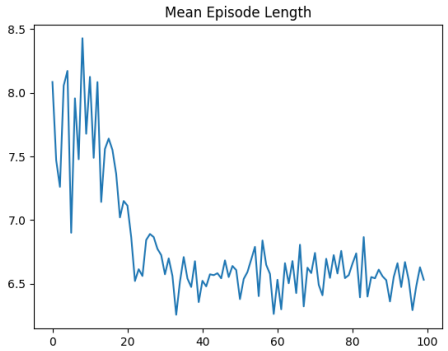

Fig. 19. Mean Episode Length

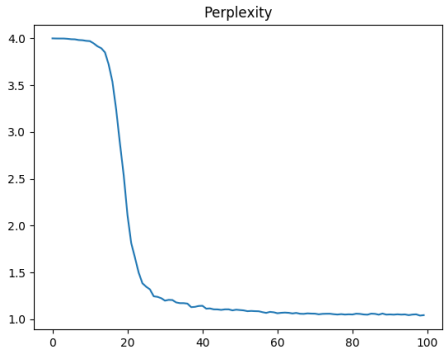

Fig. 20. Perplexity

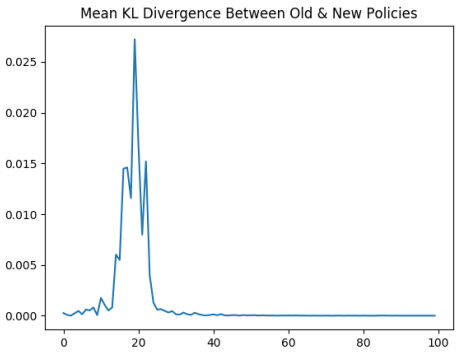

Fig. 21. Mean KL Divergence between old and new Policies

6 Optimal Path:

The optimal path chosen by the agent is indicated below. S indicates the Start location, F indicates Frozen (safe to traverse), H indicates Hole (must be avoided) and G indicates the Goal location. The highlighted element indicates the current location.

$$\begin{matrix}[S] & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & F & G\end{matrix}$$

The agent went down

$$\begin{matrix} S & F & F & F \\ [F] & H & F & H \\ F & F & F & H \\ H & F & F & G\end{matrix}$$

The agent went down

$$\begin{matrix} S & F & F & F \\ F & H & F & H \\ [F] & F & F & H \\ H & F & F & G\end{matrix}$$

The agent went to the right

$$\begin{matrix} S & F & F & F \\ F & H & F & H \\ F & [F] & F & H \\ H & F & F & G\end{matrix}$$

The agent went down

$$\begin{matrix} S & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & [F] & F & G\end{matrix}$$

The agent went to the right

$$\begin{matrix} S & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & [F] & G\end{matrix}$$

The agent went to the right

$$\begin{matrix} S & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & F & [G]\end{matrix}$$

The agent reached the goal in 6 timesteps.

7 Things to improve:

Due to a lack of time, I wasn't able to experiment with the ATARI games or implement a PGO for continuous action space as opposed to the discrete environment.

Experimenting with rewards might be a good way to understand the importance of rewards. Similarly, changing the discount factor will be a good indicator of things too.

8 Conclusion:

The Frozen Lake problem was modeled as a discrete world and as a stochastic stationary MDP. The ideal policy was computed using Policy iteration and Value Iteration. Their results were compared. The Policy Gradient was computed and optimized. The results were observed and documented.

This project was a really good way to understand the basics of Reinforcement Learning. I had absolutely no knowledge of this vast field (including how vast it is) before this project. This was a good way to familiarize myself with the basics and I feel I'll definitely learn more about RL on my own.

References

[1] Dr. Nikolai Atanasov *ESE 650 slides*

[2] Udacity *Machine Learning and Reinforcement Learning courses*