Off-Policy Learning with Recognizers and Function Approximation
COMP652 - Final Project

Jordan Frank

December 13, 2006

Contents

Motivation

- Why Off-Policy Learning?

Motivation

- Why Off-Policy Learning?
  - Cannot always control the behaviour policy.

Motivation

- Why Off-Policy Learning?
  - Cannot always control the behaviour policy.
  - Sometimes we don't even know what it is.

Motivation

- Why Off-Policy Learning?
  - Cannot always control the behaviour policy.
  - Sometimes we don't even know what it is.
- Why function approximation?

Motivation

- Why Off-Policy Learning?
  - Cannot always control the behaviour policy.
  - Sometimes we don't even know what it is.
- Why function approximation?
  - Continuous state space.

Motivation

- Why Off-Policy Learning?
  - Cannot always control the behaviour policy.
  - Sometimes we don't even know what it is.
- Why function approximation?
  - Continuous state space.
  - Continuous action space.

Motivation

- Why Off-Policy Learning?
  - Cannot always control the behaviour policy.
  - Sometimes we don't even know what it is.
- Why function approximation?
  - Continuous state space.
  - Continuous action space.
  - State space may be huge.

Function Approximation (FA)

- 2D Continuous state space, so we use Tile Coding to represent our state.
- For each state s we get an n dimensional feature vector ϕ_s representing the index of the tile in each layer of the tiling.
- Then we have a weight vector θ, and we compute the state-value function as

$$V(s) = \theta^T \phi_s$$

- Simple gradient for weight updates

$$\nabla_\theta V(s) = \phi_s$$

Function Approximation (FA)

- Logistic Regression (we'll need this later):
  - Hypothesis:

$$h_{\mathbf{w}}(\phi_s) = \sigma(\mathbf{w}^T \phi_s) = \frac{1}{1 + e^{-\mathbf{w}^T \phi_s}}$$

  - Gradient ascent update rule (on-line is what we'll be using)

$$\theta \leftarrow \theta + \alpha(y - h_{\mathbf{w}}(\phi_s))\phi_s$$

    where y is the true class label, $\mathbf{w}$ is the learned weight vector, and α is the learning rate.

Off-Policy TD Learning with FA

- Q-Learning is the most popular off-policy reinforcement learning algorithm, but it was shown in 1996 that it is unsound with linear function approximation.
- First stable algorithm for off-policy temporal-difference learning with function approximation was introduced in 2001 paper by Precup, Sutton, and Dasgupta.
- Combines TD(λ) over state-action pairs with importance sampling. Trained under any ϵ-soft policy, the algorithm converges with probability 1 to a close approximation to the action-value function for an arbitrary target policy.

Off-Policy TD Learning with FA

- Q-Learning is the most popular off-policy reinforcement learning algorithm, but it was shown in 1996 that it is unsound with linear function approximation.
- First stable algorithm for off-policy temporal-difference learning with function approximation was introduced in 2001 paper by Precup, Sutton, and Dasgupta.
- Combines TD(λ) over state-action pairs with importance sampling. Trained under any ϵ-soft policy, the algorithm converges with probability 1 to a close approximation to the action-value function for an arbitrary target policy.
- **But**, behaviour policy must be stationary and known, and even if it is, though we do converge, we may still have high variance and the convergence might be slow.

Recognizers

- Introduced in the paper *Off-policy Learning with Recognizers* by Precup, Sutton, Paduraru, Koop, and Singh earlier this year.
- A recognizer is "a filter on actions that distorts the behavior policy to produce a related target policy with low-variance important-sampling corrections".
- A recognizer is a function $c : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ where $c(s, a)$ tells us to what extent the recognizer allows action a in state s.
- For our purposes we will concern ourselves with binary recognizer functions such that $c(s, a) = 1$ if the action a is recognized in state s, and $c(s, a) = 0$ otherwise.

Recognizers

- It is important to understand that a recognizer does not define a policy, it is merely a filter.
- A recognizer c together with a behaviour policy b generates a *target policy* π, where

$$\pi(s,a) = \frac{b(s,a)c(s,a)}{\mu(s)}$$

  where $\mu(s)$ is the *recognition probability* at s, that is the probability that some action will accepted by our recognizer at s when our behaviour is generated according to b.
- So $\mu(s) = \sum_x b(s,x)c(s,x)$. But this relies on b being known, but what if it isn't?

Recognizers

- "For the case of general linear function approximation, we conjecture that [...] the recognition probability is learned using logistic regression. The development of this part is left for future work."
- And this is what I have attempted to do with this project.

Learning Algorithm

- Goal is to approximate the reward model.
- Problem: We have a fixed unknown behaviour policy that generates our actions, and we want to approximate the expected reward were we to start at some state but act according to a different policy.
- Idea:
  - Use a recognizer to distort the behaviour policy to produce our desired target policy.
  - Use linear function approximation to represent the expected reward $E\{R(s)\} \approx \theta^T \phi_s$, where ϕ_s is generated by tile coding and θ is our learned weight vector.
  - Use logistic regression to estimate our recognition probabilities $\mu(s) = \sigma(\mathbf{w}^T \phi_s)$, and update $\mathbf{w}$ at each step based on whether or not it was accepted by the recognizer.

Learning Algorithm

- The paper provides the algorithm, we just need to supplement it with logistic regression for the recognition probabilities.
- Initialize $k_0 = g_0$, $e_0 = k_0 \phi_{s_0}$, and θ and $\mathbf{w}$ to the zero vectors.
- At every transition $s_t, a_t \rightarrow r_{t+1}, s_{t+1}, a_{t+1}$:

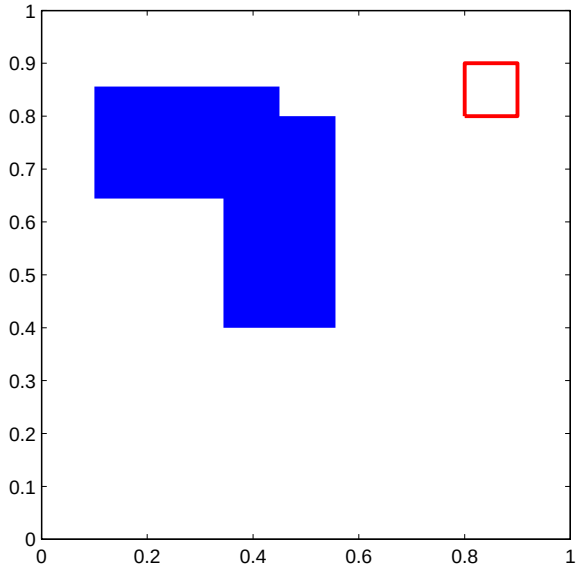
$$\begin{aligned}
\rho_t &= c(s_t, a_t)/\sigma(\mathbf{w}_t^T \phi_{s_t}) \\
\delta_t &= \rho_t(r_{t+1} + (1 - \beta_{t+1})\theta_t^T \phi_{s_{t+1}}) - \theta_t^T \phi_{s_t} \\
\theta_{t+1} &= \theta_t + \alpha \delta_t e_t \\
k_{t+1} &= \rho_t k_t (1 - \beta_{t+1}) + g_{t+1} \\
e_{t+1} &= \lambda \rho_t (1 - \beta_{t+1}) e_t + k_{t+1} \phi_{t+1} \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha'(c(s_t, a_t) - \sigma(\mathbf{w}_t^T \phi_{s_t}))\phi_{s_t}
\end{aligned}$$

Experiment

- Use RL-Glue from RLAI.net, "a standard software protocol for benchmarking and interconnecting reinforcement learning agents and environments".
- Create an continuous gridworld environment called PuddleWorld.
- Create an agent that implements the algorithm with a behaviour policy that generates uniformly random actions, and a recognizer that accepts a certain subset of the actions.
- Let the agent run around the environment for a while, moving according the the behaviour policy, but learning according to our desired target policy, and see what happens.

PuddleWorld

- Two dimensional unit square. One L shaped puddle with radius 0.1 and one small square goal region. Reward is -1 at each time step, 100 for being in the goal region at a time step, and a negative reward between 0 and -400 proportional to how close to the center the agent is if it is in the puddle.

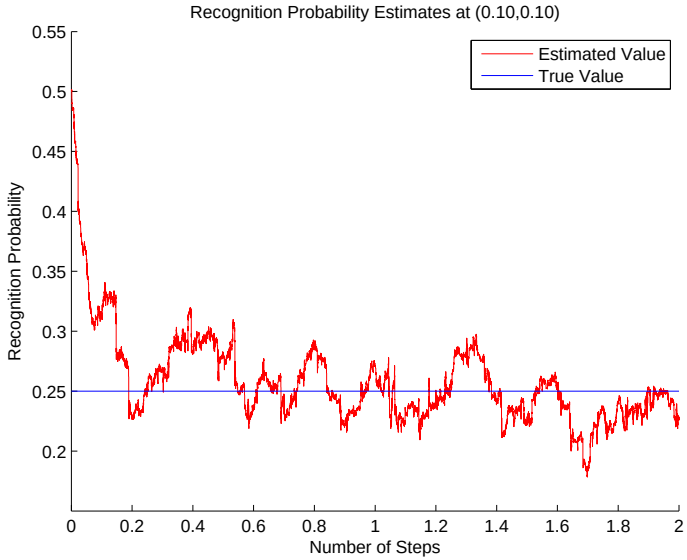


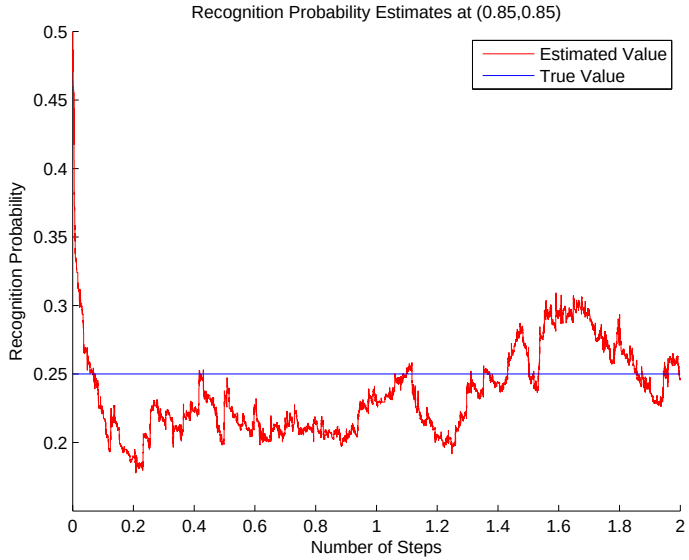
PuddleWorld

- Agents can move in one of 16 directions $\{0, \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}, \ldots, \frac{15\pi}{8}\}$ (where 0 is vertically upwards), and take steps of length 0.05. We will use a recognizer that recognizes 4 of these actions $[0, \frac{3\pi}{8}]$. Therefore the recognition probability should be 0.25.
- Initial parameters:
  - All weight vectors set to 0, since we are using a sigmoid function, that will set the initial estimates for the recognition probabilities to be 0.5.
  - $\lambda = 0.6$, $\beta = g_0 = 0.05$, $\alpha = 0.001$, $\alpha' = 0.005$.
  - 8 layers of tiling, each layer is an 8x8 grid.
- We will just run one continuous episode, updating our parameters at every step.

Results

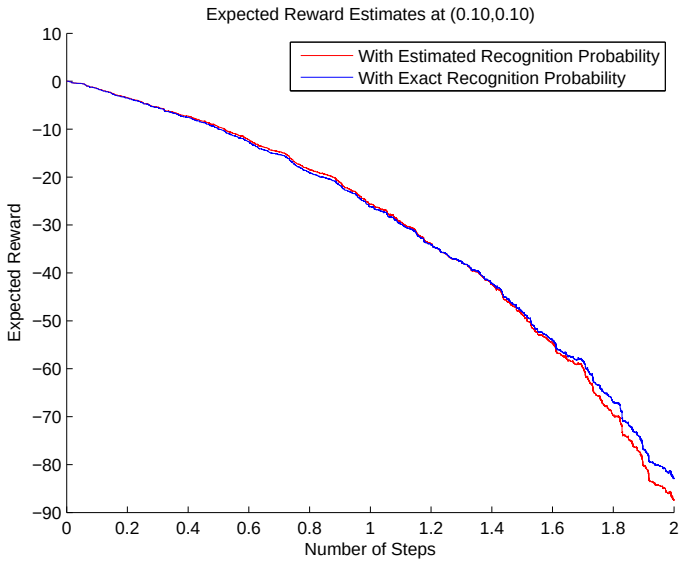
Agent seemed to learn the recognition probabilities fairly well.

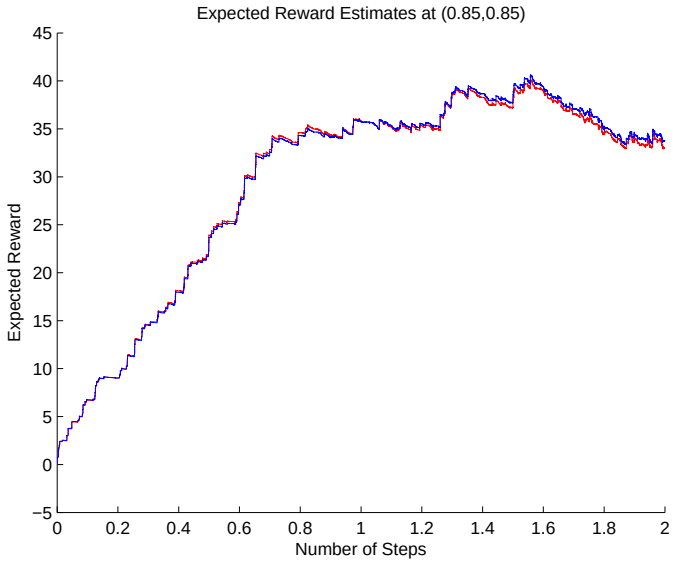




Results

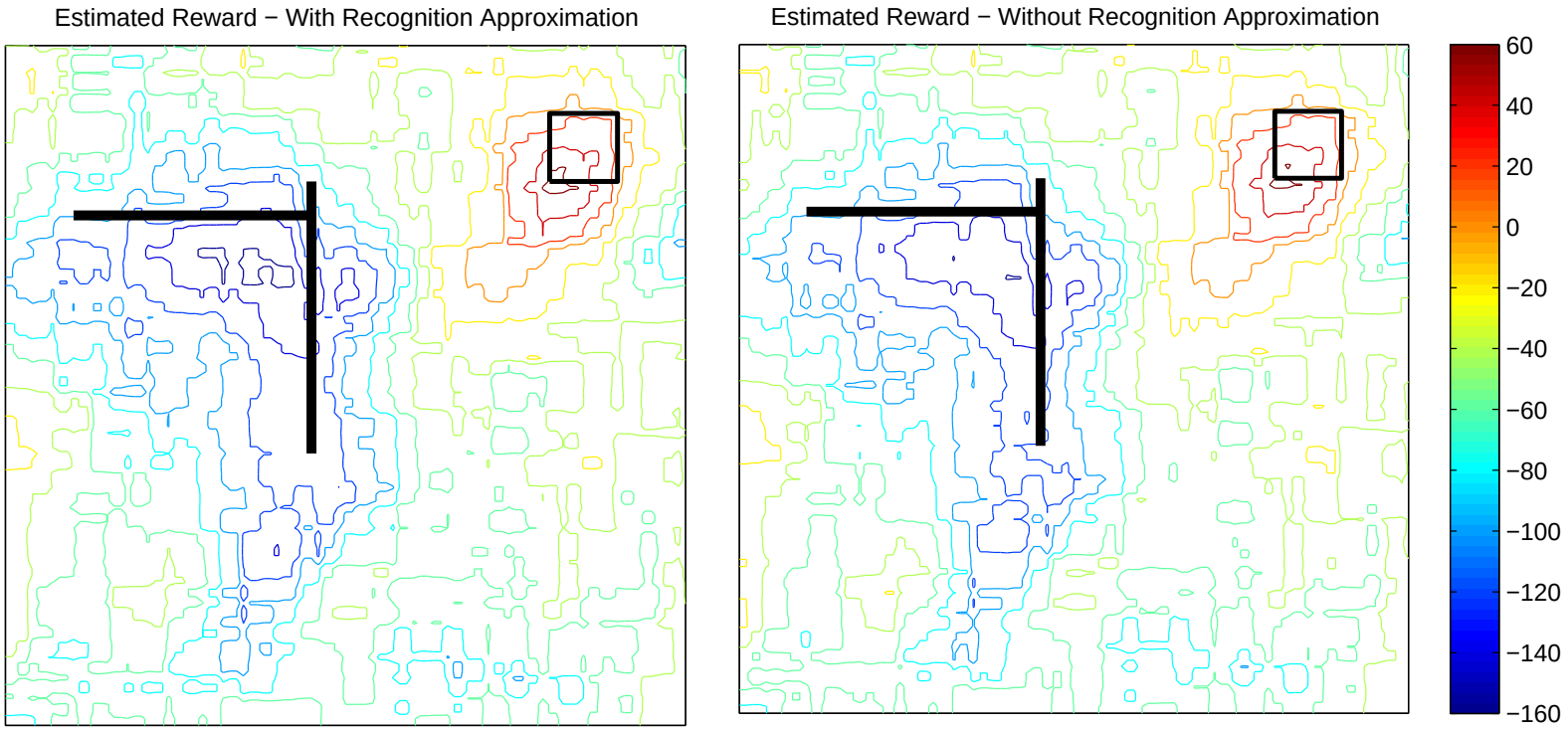
Estimating the recognition probabilities had little effect on the estimated reward model.





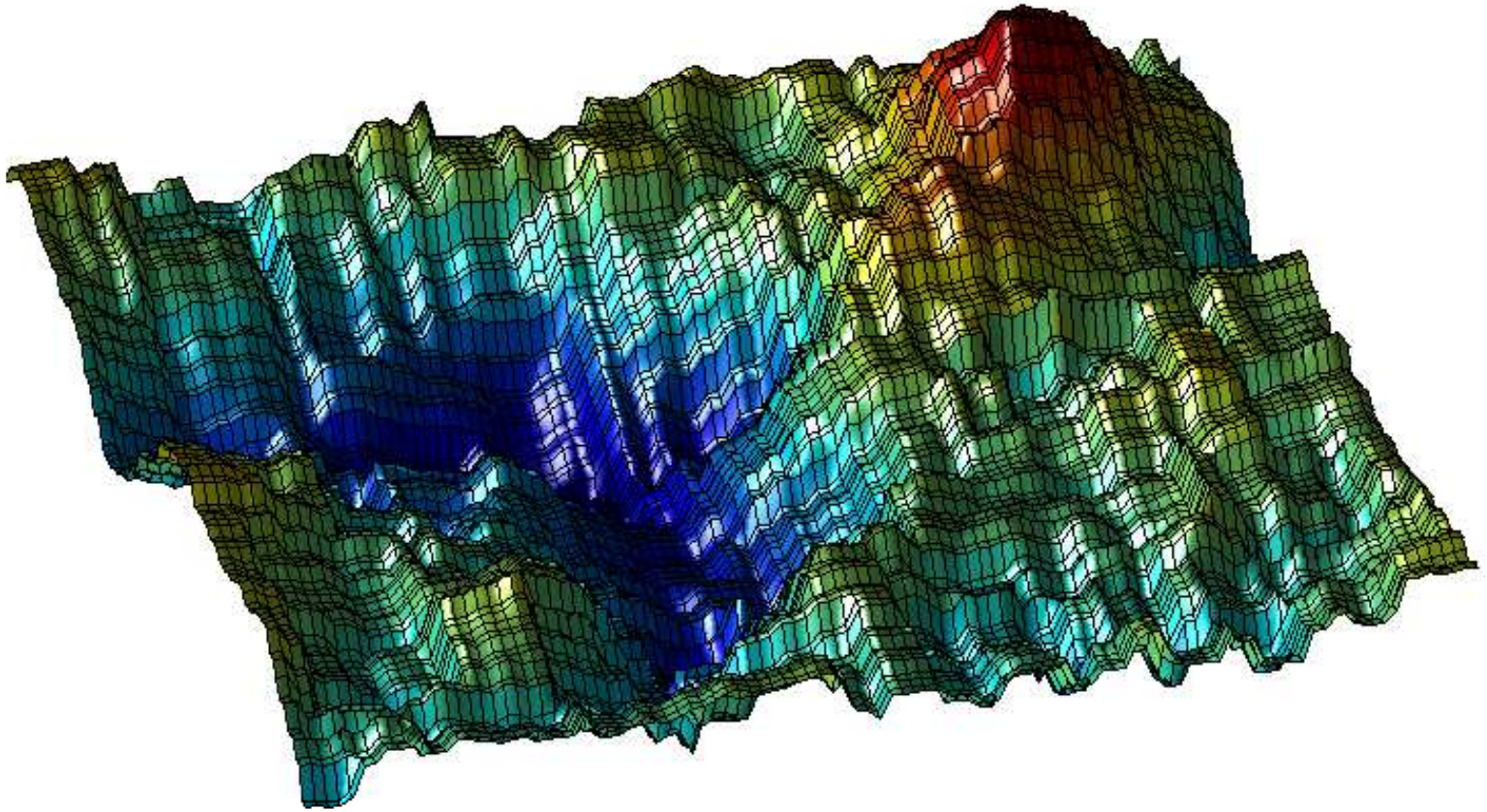
Results

Estimated reward model was what we'd expect.



Results

And some eye candy.



Conclusion and Future Work

- Lots of parameters to tune. One definite improvement would be to decrease the learning rates as we proceed. Tuning the tiling parameters also seem to make a huge difference, which makes sense.
- Longer runs. Current runs take around one minute, so we can easily increase the number of runs by an order of magnitude or two. Decrease learning rates substantially and just let it run.
- Try out more recognizers, and different behaviour policies.
- Theory. It seems like we can estimate the recognition probability using logistic regression and it will converge to the same value as it would if the recognition probability was known, but can we prove that it converges?
- Clean up the code, and maybe try to get it into the RL-Library.

References

- Precup, D., Sutton, R. S., Paduraru, C., Koop, A., Singh, S. (2006). *Off-policy Learning with Recognizers*. Advances in Neural Information Processing Systems 18 (NIPS*05)
- Precup, D., Sutton, R.S., Dasgupta, S. (2001). *Off-policy temporal-difference learning with function approximation*. Proceedings of the 18th International Conference on Machine Learning.
- Sutton, R.S., Precup, D., Singh, S. (1999). *Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning*. Artificial Intelligence 112:181-211.
- RL-Glue and RL-Library: http://rlai.cs.ualberta.ca/RLAI/rlai.html