# Robot Learning : Maximum Entropy Assignment

Junhyeok Ahn

October 9, 2017

## 1 Step Size

Gradient descent algorithm usually require careful choice on step size. In the beginning of the algorithm, large step is required for convergence rate however small step size is required at the end. To choose step size, I used line search algorithm. According to the paper, my object function and gradient could be calculated as

$$\begin{aligned} J(\boldsymbol{\theta}) &= \sum_{\tau \in D} \log P(\tau \mid \boldsymbol{\theta}) \\ \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f}_{s_i}. \end{aligned} \tag{1}$$

Once gradient is calculated, I updated new reward parameters with learning rate $\alpha$ and compute new reward as

$$\begin{aligned} \theta_{\text{new}} &= \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \\ J(\theta_{\text{new}}) &= \sum_{\tau_d \in D} \log P(\tau_d \mid \theta_{\text{new}}). \end{aligned} \tag{2}$$

Then, I compared $J(\boldsymbol{\theta})$ and $J(\theta_{\text{new}})$ and if the reward with new parameters is higher, use step size $\alpha$ but if it is smaller, I reduced step size half, so the reward could always increase. In the attached

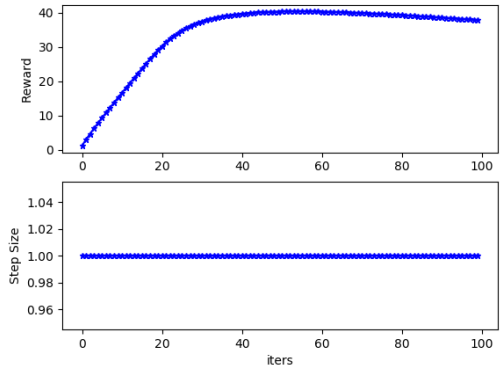


Figure 1: Use same stepsize 0.1 without adjustment

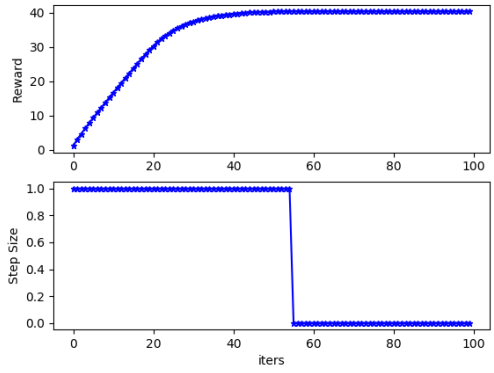


Figure 2: Stepsize adjustment using line search

code, I put line search algorithm right after the gradient computation.

## 2 100 Epochs with a Horizon of 10

I run the Maxent algorithm with 100 epochs and a horizon of 10 with line search algorithm described in the previous problem to adjust step size. The graph of reward function is shown in Figure 3.

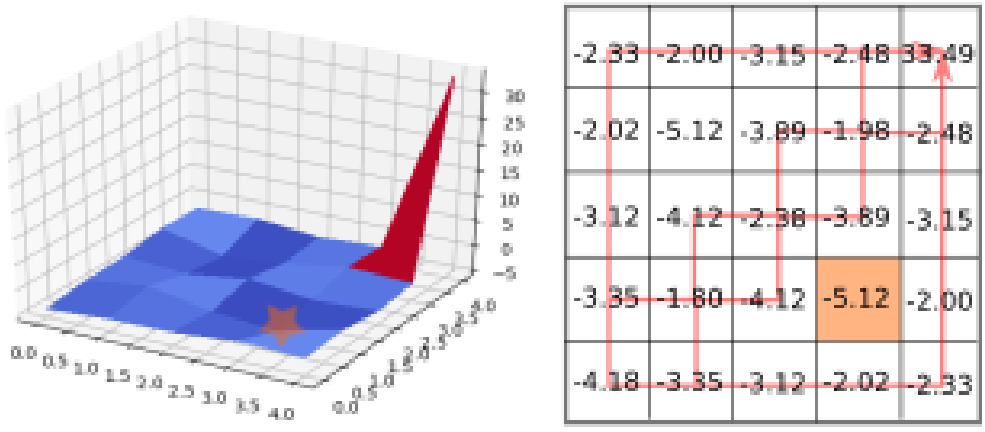


Figure 3: Reward function with 100 epochs and horizon of 10 with line search. Orange cell represent state 8 which is an obstacle.

Overally, it shows significantly high reward at state 24 or $(4, 4)$ and shows low reward at other states relatively. Since the demonstrations gvien was intentionally trying to avoid state 8 or $(1, 3)$, the reward at there is especially low. One thing interesting is that the reward function is symmetry because the given demonstrations are symmetry. That is why state 16 or $(3, 1)$ is as low as state 8 and the reward seems to tell state 16 should be also avoided.

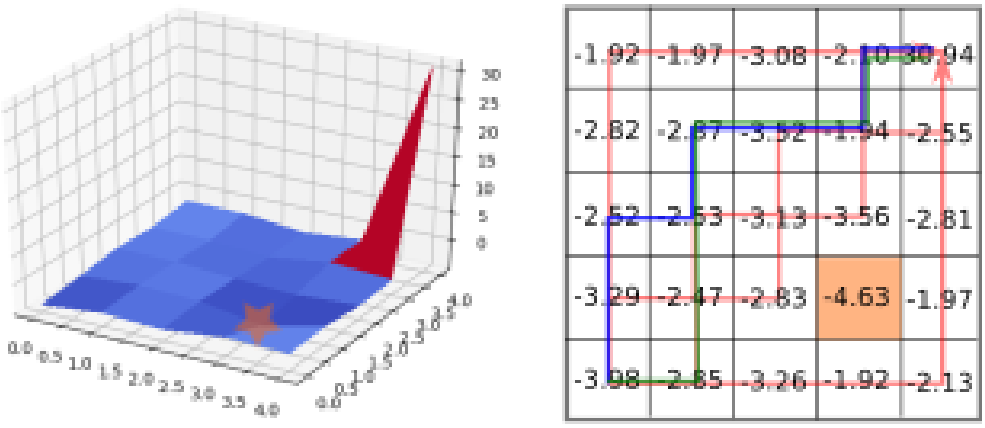


Figure 4: Reward function with 100 epochs and horizon of 10 with line search. Blue [0,5,10,11,16,17,18,23,24,25] and green [0,1,6,11,16,21,22,23,24,25] demonstrations are added to prevent underfitting. Orange cell represent state 8 which is an obstacle.

I think this result is underfitted because in order to represent demonstrations' behavior (try

to avoid state 8), reward function 16 is incorrectly induced. To prevent underfitting, we should provide more demonstration around safe regions. In Figure 4, I added two more green and blue demonstrations to give more information about reward function. In the attached, at the beginning of the algorithm I appended two more demonstration.

# 3 Derivation of Gradient

The purpose of Maxent algorithm is to find reward parameters and could be formulated as below.

$$\begin{aligned}\boldsymbol{\theta}^* &= \text{argmax}_{\boldsymbol{\theta}} \sum_{\tau_\text{d} \in \text{D}} \log \text{P}(\tau_\text{d} \mid \boldsymbol{\theta}), \\ &= \text{argmax}_{\boldsymbol{\theta}} \frac{1}{\text{M}} \sum_{\tau_\text{d} \in \text{D}} \log \frac{\text{e}^{\text{r}(\tau)}}{\text{Z}}, \\ &= \text{argmax}_{\boldsymbol{\theta}} \frac{1}{\text{M}} \sum_{\tau_\text{d} \in \text{D}} \text{r}(\tau) - \log \text{Z},\end{aligned} \tag{3}$$

where M is the number of demonstration trajectories, $r(\tau) = \sum_{s \in \tau} \boldsymbol{\theta}^\top \mathbf{f}(s)$ and $Z = \sum_{\tau \in D} e^{r(\tau)}$. Then,

$$\begin{aligned}\boldsymbol{\theta}^* &= \text{argmax}_{\boldsymbol{\theta}} \frac{1}{\text{M}} \sum_{\tau \in \text{D}} \text{r}(\tau) - \log \sum_{\tau \in \text{D}} \text{e}^{\text{r}(\tau)}, \\ &= \text{argmax}_{\boldsymbol{\theta}} \frac{1}{\text{M}} \sum_{\tau \in \text{D}} \boldsymbol{\theta}^\top \mathbf{f}_\tau - \log \sum_{\tau \in \text{D}} \text{e}^{\boldsymbol{\theta}^\top \mathbf{f}_\tau}.\end{aligned} \tag{4}$$

It is able to see the convexity of object function, which could be taken derivative as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J &= \frac{1}{M} \sum_{\tau \in D} \mathbf{f}_\tau - \frac{1}{\sum_\tau e^{r(\tau)}} \sum_\tau e^{r(\tau)} \frac{dr(\tau)}{d\boldsymbol{\theta}}, \\ &= \frac{1}{M} \sum_{\tau \in D} \mathbf{f}_\tau - \sum_{\tau \in D} \frac{e^{r(\tau)}}{\sum_\tau e^{r(\tau)}} \mathbf{f}_\tau, \\ &= \frac{1}{M} \sum_{\tau \in D} \mathbf{f}_\tau - \sum_{\tau \in D} P(\tau \mid \theta) \mathbf{f}_\tau, \\ &= \tilde{\mathbf{f}} - \sum_s D_s \mathbf{f}_s,\end{aligned} \tag{5}$$

where $\tilde{\mathbf{f}} = \frac{1}{M} \sum_{\tau \in D} \mathbf{f}_\tau$ and $D_s \mathbf{f}_s$ is state visitation frequency for state s.

# 4 Different between Maxent algorithm and Action Based Distribution model

Intuitevely Maxent algorithm (Eq.(5) in the paper) computes probability of an action as the expectation of exponentiated rewards of all path that start with the action. For example in Figure 5, when compute for $P(\uparrow \mid s_\text{start})$, Maxent considers all possible path candidates (e.g. passing through (1) and (2)) after taking action $\uparrow$. As a result, Maxent algorithm regards taking action $\uparrow$ or $\rightarrow$ at starting state in Figure 5 as same probability.

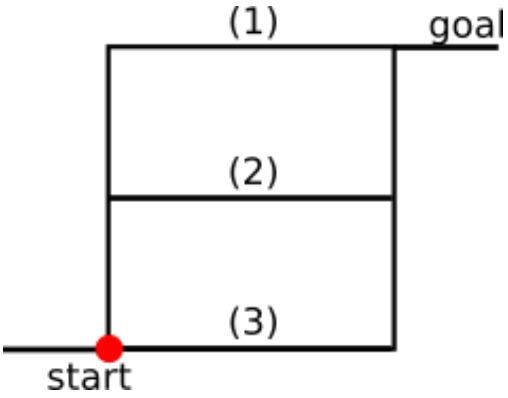


Figure 5: Example of probability distribution over paths

However, action based model (Eq.(7) in the paper) computes action probability as the expectation of exponentiated reward of a optimal path induced from policy after taking the action. For example in Figure 5, $P(\uparrow|\ s_{\text{start}})$ is distributed based on the best reward of the best policy after taking action $\uparrow$ (e.g. passing through (2)). Therefore, although taking $\rightarrow$ or $\uparrow$ should yield same reward and is supposed to be the same probability, action based model shows some bias due to the action level probability mass.

This biased weight is problematic because it does not represent reward function and violates the main assumption in the paper that if the reward of trajectory is same, the probability is also same

## 5 $Z_{a_{i,j}}$ and $Z_{s_i}$

If we only have one terminal state and set the probability of being ended up at terminal state is 1, recursively computed $Z_{a_{i,j}}$ represents the occupancy associated state i when action j is taken (how much action j at state i contributes to arrival of terminal state). It can be seen similar to state-action value in terms of being described for each actions at each states. Moreover, $Z_{s_i}$ is computed adding up all $Z_{a_{i,j}}$'s which are state action values and could be interpreted as in trajectory how much portion $s_i$ is taking charge of given reward parameters. Therefore, the paper calculates the policy by dividing each actions occupancy with total occupancy at state.

## 6 Set $Z_s$ as Zero for Terminal State

### 6.1

As I mentioned in problem 5, $Z_{a_{i,j}}$ and $Z_{s_i}$ are backpropagated every iterations and represent how each states and actions contribute to being ended at the terminal state. In this case, there is only one terminal state and all trajectoried should be terminated at there. In this perspective, $Z_s$ at terminal state should be set as 1. If we have two different equally distributed terminal states, they should be 0.5 each.

### 6.2

Reward at terminal state is always zero and the transition probability is zeros. Therefore, $Z_{ai,j}$ and $Z_{s_i}$ at terminal state ends up being computed as zeros at each iteration. Therefore $Z_s$ at terminal

should reset to one at each iteration to represent the reward of being at terminal state.