# What can I improve Apprenticeship learning as DNN? (with experiments)

Jeong Gwan Lee

KAIST

(Korea Advanced Institute of Science and Technology)

# Batch, Off-Policy and Model-Free Apprenticeship Learning

Klein, Edouard, Matthieu Geist, and Olivier Pietquin. "Batch, off-policy and model-free apprenticeship learning." *European Workshop on Reinforcement Learning*. Springer, Berlin, Heidelberg, 2011.

# IRL set-up

The true reward function belongs to some hypothesis space

$$\mathcal{H}_\phi = \{\theta^T \phi(s), \theta \in \mathbb{R}^p\}, \ |\phi_i(s)| \leq 1, \forall s \in S, 1 \leq i \leq p.$$

$$R^*(s) = (\theta^*)^T \phi(s)$$

parameters, weights. features; state representation; input.

$$R(s) = f_N(\ldots(f_2(f_1(x, \theta_1), \theta_2), \ldots), \theta_N)$$

$f_i$ : DNN layer with activation function.

*Reward estimator*

$$\phi(s) = f_N(\ldots(f_2(f_1(x, \theta_1), \theta_2), \ldots), \theta_N)$$

$f_i$ : DNN layer with activation function.

*Feature expect. estimator*

# IRL set-up

For any reward function belonging to $\mathcal{H}_\phi = \{\theta^T \phi(s), \theta \in \mathbb{R}^p\}$,
Value function V(s) can be expressed,

$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \theta^T \phi(s_t) | s_0 = s, \pi] = \theta^T E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | s_0 = s, \pi]$$

1.(reward esti.) DNN, Non-linearity, so, might not make feature expectation?

2. (feature extractor) $\phi(s_t)$ might be "the feature output just before softmax" in classification? → Depending on the input.

*Feature expectation* is,

$$\mu^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | s_0 = s, \pi]$$

$$|V^{\pi_E}(s_0) - V^{\tilde{\pi}}(s_0)| = |\theta^T(\mu^{\pi_E}(s_0) - \mu^{\tilde{\pi}}(s_0))| \leq \|\mu^{\pi_E}(s_0) - \mu^{\tilde{\pi}}(s_0)\|_2$$

# IRL Algorithm

1. Starts with some initial policy $\pi^{(0)}$ and compute $\mu^{\pi^{(0)}}(s_0)$. Set $j = 1$;
2. Compute $t^{(j)} = \max_{\theta:\|\theta\|_2 \leq 1} \min_{k \in \{0, j-1\}} \theta^T(\mu^{\pi_E}(s_0) - \mu^{\pi^{(k)}}(s_0))$ and let $\theta^{(j)}$ be the value attaining this maximum. At this step, one searches for the reward function which maximizes the distance between the value of the expert at $s_0$ and the value of *any* policy computed so far (still at $s_0$). This optimization problem can be solved using a quadratic programming approach or a projection algorithm [1];
3. if $t^{(j)} \leq \epsilon$, terminate. The algorithm outputs a set of policies $\{\pi^{(0)}, \ldots, \pi^{(j-1)}\}$ among which the user chooses manually or automatically the closest to the expert (see [1] for details on how to choose this policy). Notice that the last policy is not necessarily the best (as illustrated in Section 4);
4. solve the MDP with the reward function $R^{(j)}(s) = (\theta^{(j)})^T \phi(s)$ and denote $\pi^{(j)}$ the associated optimal policy. Compute $\mu^{\pi^{(j)}}(s_0)$;
5. set $j \leftarrow j + 1$ and go back to step 2.

# IRL Algorithm

1. Starts with some initial policy $\pi^{(0)}$ and compute $\mu^{\pi^{(0)}}(s_0)$. Set $j = 1$;

2. Compute $t^{(j)} = \max_{\theta:\|\theta\|_2 \leq 1} \min_{k \in \{0,j-1\}} \theta^T(\mu^{\pi_E}(s_0) - \mu^{\pi^{(k)}}(s_0))$ and let $\theta^{(j)}$ be the value attaining this maximum. At this step, one searches for the reward function which maximizes the distance between the value of the expert at $s_0$ and the value of *any* policy computed so far (still at $s_0$). This optimization problem can be solved using a quadratic programming approach or a projection algorithm [1];

[1] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

# IRL Algorithm – Projection method

### 3.1. A simpler algorithm

The algorithm described above requires access to a QP (or SVM) solver. It is also possible to change the algorithm so that no QP solver is needed. We will call the previous, QP-based, algorithm the **max-margin** method, and the new algorithm the **projection** method. Briefly, the projection method replaces step 2 of the algorithm with the following:

- Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)}-\bar{\mu}^{(i-2)})^T(\mu_E-\bar{\mu}^{(i-2)})}{(\mu^{(i-1)}-\bar{\mu}^{(i-2)})^T(\mu^{(i-1)}-\bar{\mu}^{(i-2)})}(\mu^{(i-1)}-\bar{\mu}^{(i-2)})$
(This computes the orthogonal projection of $\mu_E$ onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$.)
- Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
- Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$

[1] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

# IRL Algorithm

3. if $t^{(j)} \leq \epsilon$, terminate. The algorithm outputs a set of policies $\{\pi^{(0)}, \ldots, \pi^{(j-1)}\}$ among which the user chooses manually or automatically the closest to the expert (see [1] for details on how to choose this policy). Notice that the last policy is not necessarily the best (as illustrated in Section 4);

**LSPI(Least Square Policy Iteration)[2] : LSTD-Q + Policy Evaluation**

4. solve the MDP with the reward function $R^{(j)}(s) = (\theta^{(j)})^T \phi(s)$ and denote $\pi^{(j)}$ the associated optimal policy. Compute $\mu^{\pi^{(j)}}(s_0)$;

**LSTD-$\mu$**

[2]Lagoudakis, Michail G., and Ronald Parr. "Least-squares policy iteration." *Journal of machine learning research* 4.Dec (2003): 1107-1149.

# IRL Algorithm

1. Initialize $\mu^E(s_0)$ $\pi^{(0)}$ $\mu^{(0)}(s_0)$

2. Projection method

$\bar{\mu}^{(0)} = \mu^{(0)}(s_0)$ $\theta^{(1)} = \mu^E(s_0) - \bar{\mu}^{(0)}$ $t^{(1)} = \left\|\mu^E(s_0) - \bar{\mu}^{(0)}\right\|_2$

3. if $t^{(j)} \leq \epsilon$, terminate.

4. Solve MDP using LSPI, get $\mu^{\pi^{(1)}}(s_0)$ using LSTD-$\mu$.

$$R^{(1)}(s) = (\theta^{(1)})^T \phi(s) \xrightarrow[\text{LSPI}]{} \pi^{(1)} \xrightarrow[\text{LSTD-}\mu]{} \mu^{\pi^{(1)}}(s_0)$$

---

2. Projection method

$\bar{\mu}^{(1)} =$ Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu^{(i-1)} - \bar{\mu}^{(i-2)})}(\mu^{(i-1)} - \bar{\mu}^{(i-2)})$ $\theta^{(2)} = \mu^E(s_0) - \bar{\mu}^{(1)}$ $t^{(2)} = \left\|\mu^E(s_0) - \bar{\mu}^{(1)}\right\|_2$

3. if $t^{(j)} \leq \epsilon$, terminate.

4. Solve MDP using LSPI, get $\mu^{\pi^{(2)}}(s_0)$ using LSTD-mu.

$$R^{(2)}(s) = (\theta^{(2)})^T \phi(s) \xrightarrow[\text{LSPI}]{} \pi^{(2)} \xrightarrow[\text{LSTD-}\mu]{} \mu^{\pi^{(2)}}(s_0)$$

# IRL Algorithm – LSPI(Least Square Policy Iteration)

Approximate Q

$$\widehat{Q}^{\pi} = \mathbf{\Phi} w^{\pi}$$

$$\mathbf{\Phi} = \begin{pmatrix} \phi(s_1, a_1)^{\intercal} \\ \dots \\ \phi(s, a)^{\intercal} \\ \dots \\ \phi(s_{|\mathcal{S}|}, a_{|\mathcal{A}|})^{\intercal} \end{pmatrix}$$

Find $w^{\pi}$

$$w^{\pi} = \left(\mathbf{\Phi}^{\intercal}\Delta_{\mu}(\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Pi}_{\pi}\mathbf{\Phi})\right)^{-1}\mathbf{\Phi}^{\intercal}\Delta_{\mu}\mathcal{R}$$

$$w^{\pi} = A^{-1}b$$

$$\mathbf{A} = \mathbf{\Phi}^{\intercal}\Delta_{\mu}(\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Pi}_{\pi}\mathbf{\Phi}) \qquad \text{and} \qquad b = \mathbf{\Phi}^{\intercal}\Delta_{\mu}\mathcal{R}$$

[2]Lagoudakis, Michail G., and Ronald Parr. "Least-squares policy iteration." *Journal of machine learning research* 4.Dec (2003): 1107-1149.

# IRL Algorithm – LSPI(Least Square Policy Iteration)

$$\begin{aligned}\mathbf{A} &= \mathbf{\Phi}^{\intercal}\Delta_{\mu}(\mathbf{\Phi} - \gamma \mathbf{P\Pi}_{\pi}\mathbf{\Phi}) \\ &= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \phi(s,a)\mu(s,a)\Big(\phi(s,a) - \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\phi(s',\pi(s'))\Big)^{\intercal} \\ &= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mu(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\Big[\phi(s,a)\Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)^{\intercal}\Big]\end{aligned}$$

$$\begin{aligned}b &= \mathbf{\Phi}^{\intercal}\Delta_{\mu}\mathcal{R} \\ &= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \phi(s,a)\mu(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')R(s,a,s') \\ &= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mu(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\Big[\phi(s,a)R(s,a,s')\Big] .\end{aligned}$$

Random action sampling!
not follow policy.

$$D = \Big\{ (s_i, a_i, r_i, s_i') \;\Big|\; i = 1, 2, \ldots, L \Big\}$$

$$\widetilde{\mathbf{A}} = \frac{1}{L}\sum_{i=1}^{L}\Big[\phi(s_i,a_i)\Big(\phi(s_i,a_i) - \gamma\phi(s_i',\pi(s_i'))\Big)^{\intercal}\Big]$$

$$\widetilde{b} = \frac{1}{L}\sum_{i=1}^{L}\Big[\phi(s_i,a_i)r_i\Big] ,$$

```
# Intializaiton
state = env.reset()

# Collect samples
for j in range(TRANSITION):
    if isRender:
        env.render()
    action = env.action_space.sample()
    next_state, reward, done, info = env.step(action)
    memory.add([state, action, reward, next_state, done])
    state = next_state
    if done:
        break
```

# IRL Algorithm – LSPI(Least Square Policy Iteration)

**LSTDQ** $(D,\ k,\ \phi,\ \gamma,\ \pi)$ // Learns $\widehat{Q}^{\pi}$ from samples

// $D$ : Source of samples $(s,a,r,s')$
// $k$ : Number of basis functions
// $\phi$ : Basis functions
// $\gamma$ : Discount factor
// $\pi$ : Policy whose value function is sought

$\widetilde{\mathbf{A}} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$\widetilde{b} \leftarrow \mathbf{0}$ // $(k \times 1)$ vector

**for each** $(s,a,r,s') \in D$
$\widetilde{\mathbf{A}} \leftarrow \widetilde{\mathbf{A}} + \phi(s,a)\Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)^{\intercal}$
$\widetilde{b} \leftarrow \widetilde{b} + \phi(s,a)r$

$\widetilde{w}^{\pi} \leftarrow \widetilde{\mathbf{A}}^{-1}\widetilde{b}$

**return** $\widetilde{w}^{\pi}$

$$\widehat{Q}^{\pi} = \Phi w^{\pi}$$

**LSPI** $(D,\ k,\ \phi,\ \gamma,\ \epsilon,\ \pi_0)$ // Learns a policy from samples

// $D$ : Source of samples $(s,a,r,s')$
// $k$ : Number of basis functions
// $\phi$ : Basis functions
// $\gamma$ : Discount factor
// $\epsilon$ : Stopping criterion
// $\pi_0$ : Initial policy, given as $w_0$ (default: $w_0 = 0$)

$\pi' \leftarrow \pi_0$ // $w' \leftarrow w_0$

**repeat**
$\pi \leftarrow \pi'$ // $w \leftarrow w'$
$\pi' \leftarrow$ **LSTDQ** $(D,\ k,\ \phi,\ \gamma,\ \pi)$ // $w' \leftarrow$ **LSTDQ** $(D,\ k,\ \phi,\ \gamma,\ w)$
**until** $(\pi \approx \pi')$ // **until** $(\|w - w'\| < \epsilon)$

**return** $\pi$ // **return** $w$

[2]Lagoudakis, Michail G., and Ronald Parr. "Least-squares policy iteration." *Journal of machine learning research* 4.Dec (2003): 1107-1149.

# LSTD-$\mu$

Compute $\mu^{\pi^{(j)}}(s_0)$ LSTD-$\mu$ : to estimate feature expectation of intermediate policies

$$\mu_i^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \phi_i(s_t)|s_0 = s, \pi]. \quad \approx \quad V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_0 = s, \pi]$$

$$\mathcal{H}_\psi = \{\hat{V}_\xi(s) = \sum_{i=1}^{q} \xi_i \psi_i(s) = \xi^T \psi(s), \xi \in \mathbb{R}^q\}$$

LSTD-Q

$$\xi_i^* = \left(\sum_{t=1}^{n} \psi(s_t)(\psi(s_t) - \gamma\psi(s'_t))^T\right)^{-1} \sum_{t=1}^{n} \psi(s_t)\phi_i(s_t)$$

$$\widehat{Q}^\pi = \mathbf{\Phi} w^\pi$$

$$w^\pi = \left(\mathbf{\Phi}^\intercal \Delta_\mu (\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi})\right)^{-1} \mathbf{\Phi}^\intercal \Delta_\mu \mathcal{R}$$

$$w_i^* = \left(\sum_{t=1}^{n} \phi(s_t, a_t)\left(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi(s'_t))\right)^T\right)^{-1} \sum_{t=1}^{n} \phi(s_t, a_t) r(s_t, a_t)$$

a set of transitions $\{(s_t, r_t, s_{t+1})_{1 \le t \le n}\}$ sampled according to the policy $\pi$

$$(\hat{\mu}^\pi(s_0))^T = \psi(s_0)^T (\Psi^T \Delta \Psi)^{-1} \Psi^T \Phi$$

**for each** $(s, a, r, s') \in D$
$\widetilde{\mathbf{A}} \leftarrow \widetilde{\mathbf{A}} + \phi(s,a)\left(\phi(s,a) - \gamma\phi(s', \pi(s'))\right)^\intercal$
$\widetilde{b} \leftarrow \widetilde{b} + \phi(s,a) r$

# LSTD-$\mu$

Compute $\mu^{\pi^{(j)}}(s_0)$ LSTD-$\mu$ : to estimate feature expectation of intermediate policies

$$\mu_i^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \phi_i(s_t) | s_0 = s, \pi]. \approx V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s, \pi]$$

$$\mathcal{H}_\psi = \{\hat{V}_\xi(s) = \sum_{i=1}^{q} \xi_i \psi_i(s) = \xi^T \psi(s), \xi \in \mathbb{R}^q\}$$

$$\xi_i^* = \left(\sum_{t=1}^{n} \psi(s_t)(\psi(s_t) - \gamma\psi(s_t'))^T\right)^{-1} \sum_{t=1}^{n} \psi(s_t)\phi_i(s_t)$$

LSTD-Q

$$\widehat{Q}^\pi = \mathbf{\Phi} w^\pi$$

$$w^\pi = \left(\mathbf{\Phi}^\intercal \Delta_\mu (\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi})\right)^{-1} \mathbf{\Phi}^\intercal \Delta_\mu \mathcal{R}$$

$$w_i^* = \left(\sum_{t=1}^{n} \phi(s_t, a_t)\left(\phi(s_t, a_t) - \gamma\phi(s_t', \pi(s_t'))\right)^T\right)^{-1} \sum_{t=1}^{n} \phi(s_t, a_t) r(s_t, a_t)$$

a set of transitions $\{(s_t, r_t, s_{t+1})_{1 \le t \le n}\}$ sampled according to the policy $\pi$

$$(\hat{\mu}^\pi(s_0))^T = \psi(s_0)^T (\Psi^T \Delta \Psi)^{-1} \Psi^T \Phi$$

**for each** $(s, a, r, s') \in D$

$\tilde{\mathbf{A}} \leftarrow \tilde{\mathbf{A}} + \phi(s,a)\left(\phi(s,a) - \gamma\phi(s', \pi(s'))\right)^\intercal$

$\tilde{b} \leftarrow \tilde{b} + \phi(s,a) r$

# LSTD-$\mu$ as off-policy manner

$$
\begin{aligned}
\mathbf{A} &= \mathbf{\Phi}^\intercal \Delta_\mu (\mathbf{\Phi} - \gamma \mathbf{P} \mathbf{\Pi}_\pi \mathbf{\Phi}) \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \phi(s,a)\mu(s,a) \Big(\phi(s,a) - \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\phi(s',\pi(s'))\Big)^\intercal \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \Big[\phi(s,a)\Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)^\intercal\Big]
\end{aligned}
$$

$$
\begin{aligned}
b &= \mathbf{\Phi}^\intercal \Delta_\mu \mathcal{R} \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \phi(s,a)\mu(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') R(s,a,s') \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \Big[\phi(s,a) R(s,a,s')\Big] \ .
\end{aligned}
$$

$$
D = \Big\{ \ (s_i, a_i, r_i, s'_i) \ | \ i = 1, 2, \ldots, L \ \Big\}
$$

$$
\widetilde{\mathbf{A}} = \frac{1}{L} \sum_{i=1}^{L} \Big[\phi(s_i,a_i)\Big(\phi(s_i,a_i) - \gamma\phi(s'_i,\pi(s'_i))\Big)^\intercal\Big]
$$

$$
\widetilde{b} = \frac{1}{L} \sum_{i=1}^{L} \Big[\phi(s_i,a_i) r_i\Big] \ ,
$$

Random action sampling!
not following policy.

```
action = env.action_space.sample()
next_state, reward, done, info = env.step(action)
memory.add([state, action, reward, next_state, done])
```

Additional degree of freedom( $a_0 = a$ ) allows off-policy learning.(LSTD-Q)
LSTD-Q (Q-function) → LSTD-$\mu$ (state-action feature expectation)

# Implementation & Experiment

# Critical Problem 1

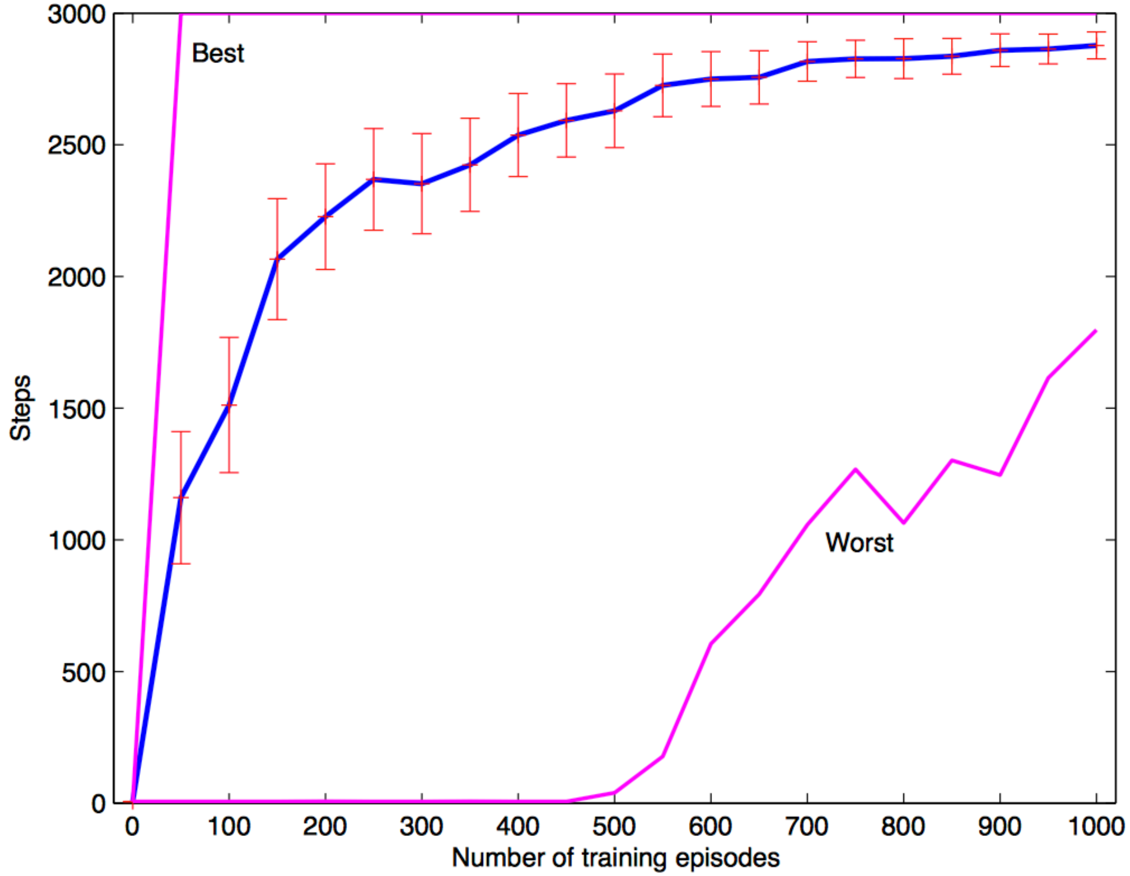
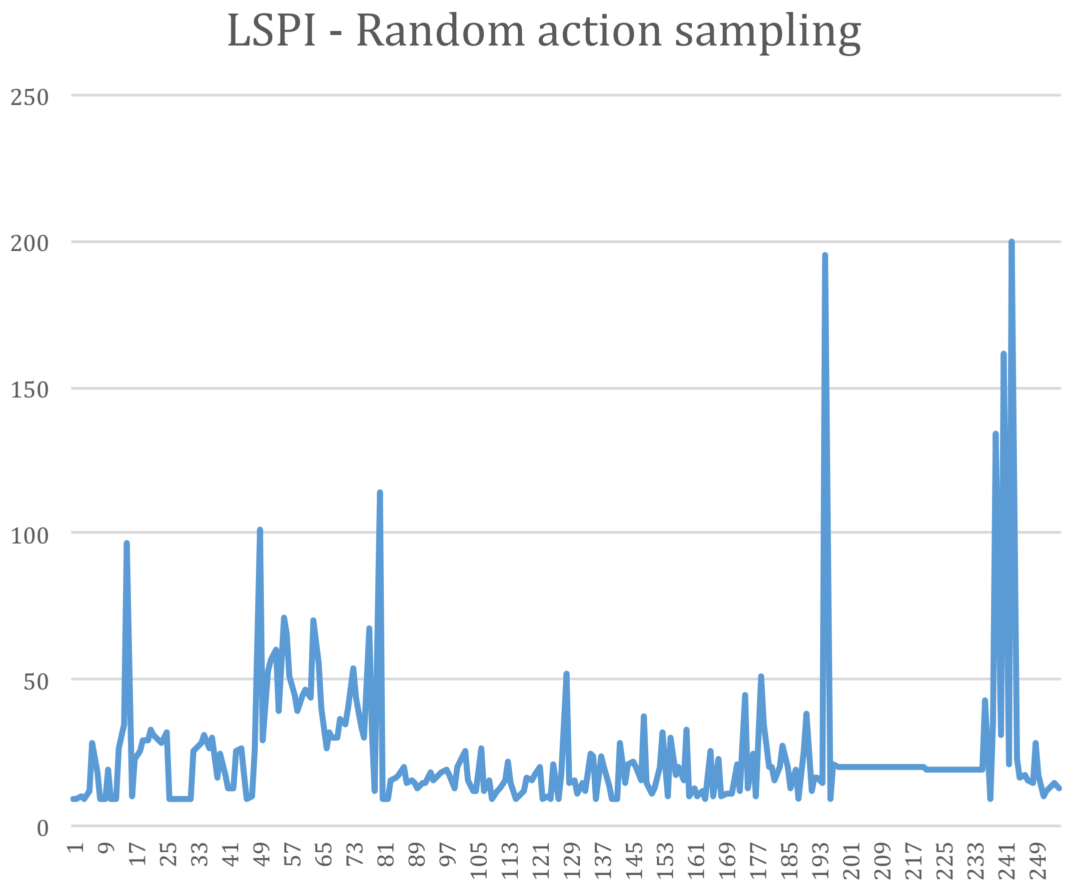



Figure 16: Inverted pendulum (LSPI): Average balancing steps.




[Left figure]Lagoudakis, Michail G., and Ronald Parr. "Least-squares policy iteration." *Journal of machine learning research* 4.Dec (2003): 1107-1149.

# Experiment Setting


CartPole

state dimension : 4

# of actions : 2

# Experiment Setting

1. Initialize $\mu^E(s_0)$ $\pi^{(0)}$ $\mu^{(0)}(s_0)$

2. Projection method

$\bar{\mu}^{(0)} = \mu^{(0)}(s_0)$ $\theta^{(1)} = \mu^E(s_0) - \bar{\mu}^{(0)}$ $t^{(1)} = \left\|\mu^E(s_0) - \bar{\mu}^{(0)}\right\|_2$

**30 iteration for one LSPI**

3. if $t^{(j)} \leq \epsilon$, terminate.

4. Solve MDP using LSPI, get $\mu^{\pi^{(1)}}(s_0)$ using LSTD-$\mu$.

$R^{(1)}(s) = (\theta^{(1)})^T \phi(s) \xrightarrow{\text{LSPI}} \pi^{(1)} \xrightarrow{\text{LSTD-}\mu} \mu^{\pi^{(1)}}(s_0)$

**Baseline 1**

---

$\mu^{\pi^{(1)}}(s_0)$

compute feature expectation by MC approach(like expert mu)

**Baseline 2**

2. Projection method

$\bar{\mu}^{(1)} =$ Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu^{(i-1)} - \bar{\mu}^{(i-2)})}(\mu^{(i-1)} - \bar{\mu}^{(i-2)})$

$\theta^{(2)} = \mu^E(s_0) - \bar{\mu}^{(1)}$ $t^{(2)} = \left\|\mu^E(s_0) - \bar{\mu}^{(1)}\right\|_2$

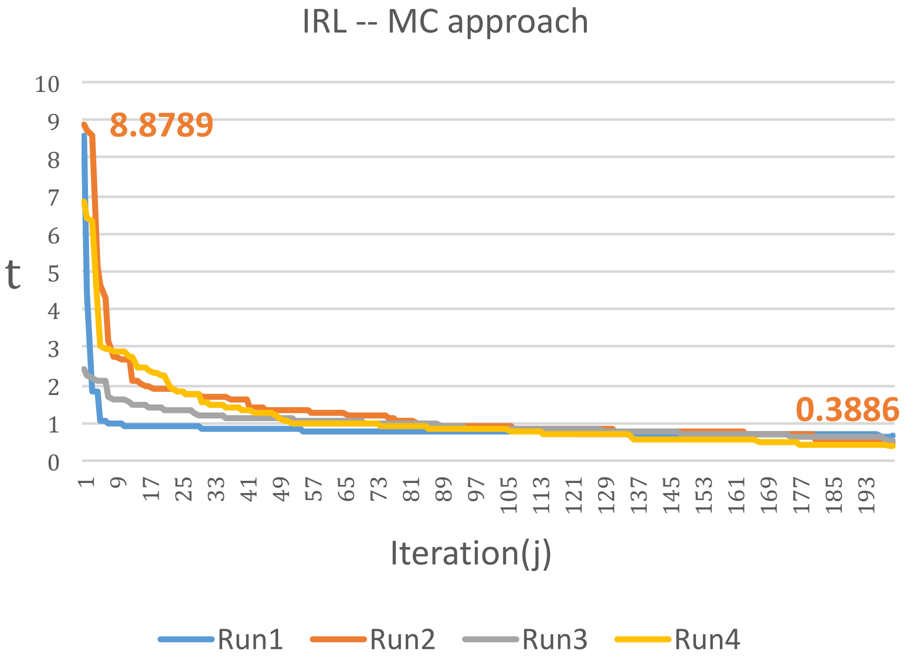
3. if $t^{(j)} \leq \epsilon$, terminate.

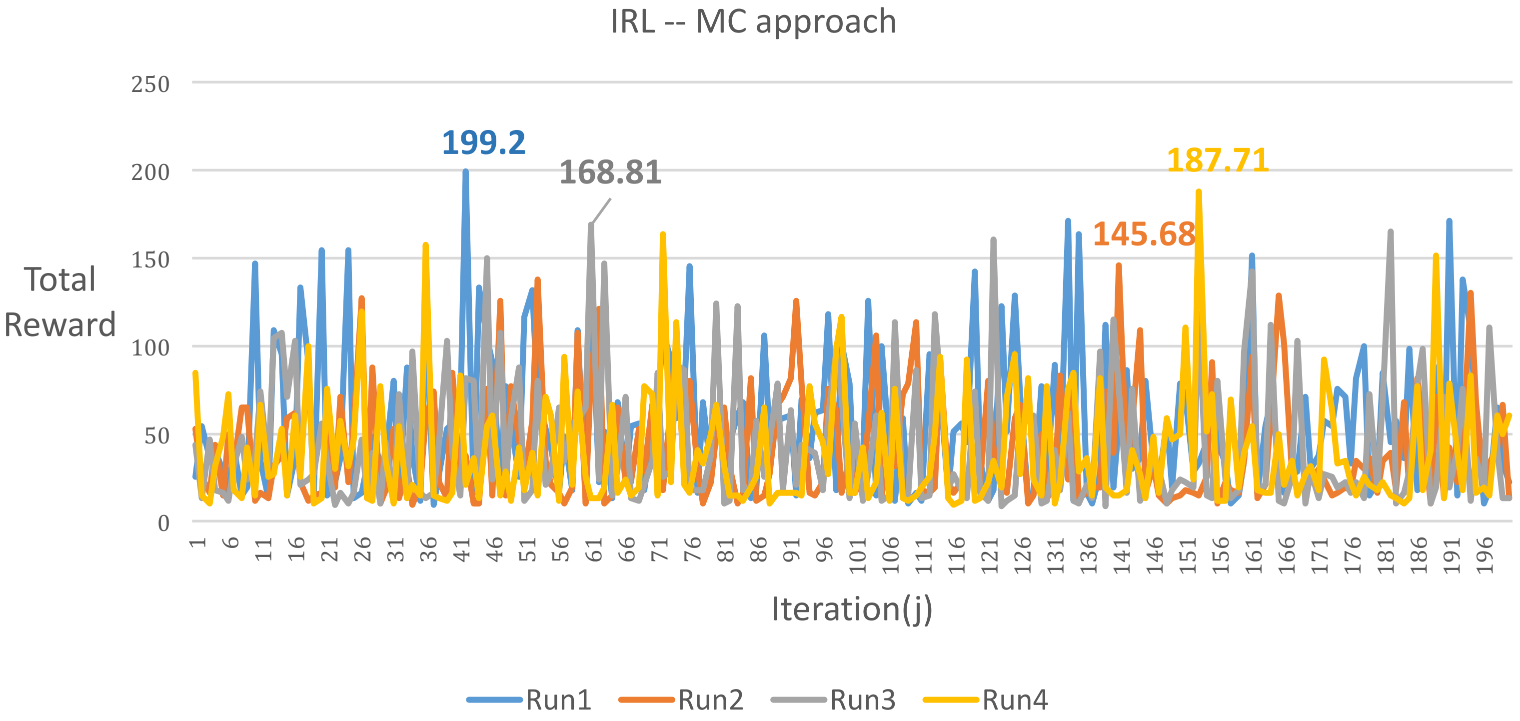
4. Solve MDP using LSPI, get $\mu^{\pi^{(2)}}(s_0)$ using LSTD-mu.

$R^{(2)}(s) = (\theta^{(2)})^T \phi(s) \xrightarrow{\text{LSPI}} \pi^{(2)} \xrightarrow{\text{LSTD-}\mu} \mu^{\pi^{(2)}}(s_0)$

# IRL + MC approach

$$t^{(i)} = \left\|\mu^E(s_0) - \bar{\mu}^{(i-1)}\right\|_2$$



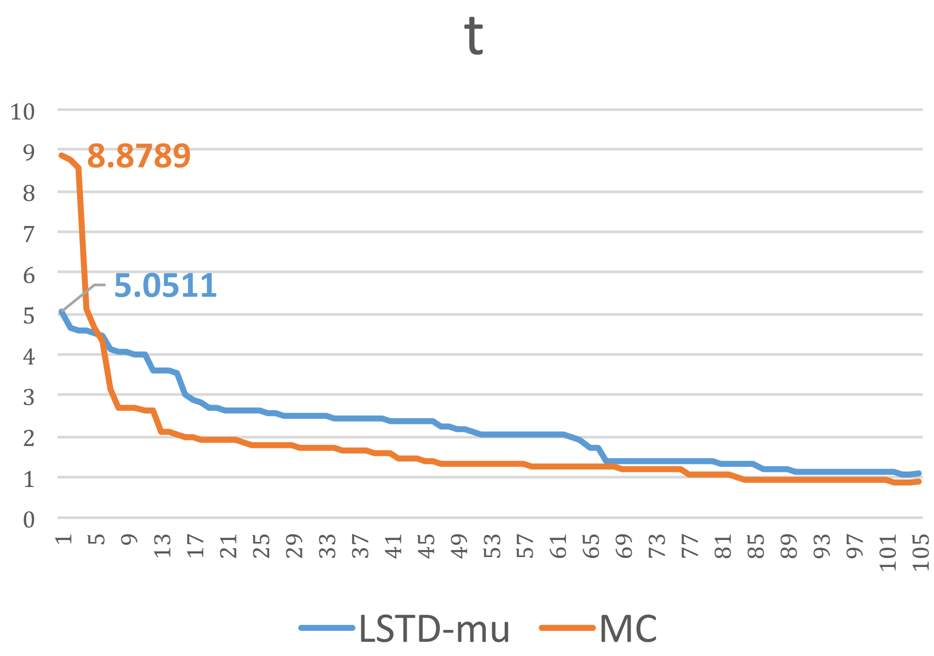

$$\mathcal{H}_\phi = \{\theta^T\phi(s), \theta \in \mathbb{R}^p\},\ |\phi_i(s)| \le 1, \forall s \in S, 1 \le i \le p.$$

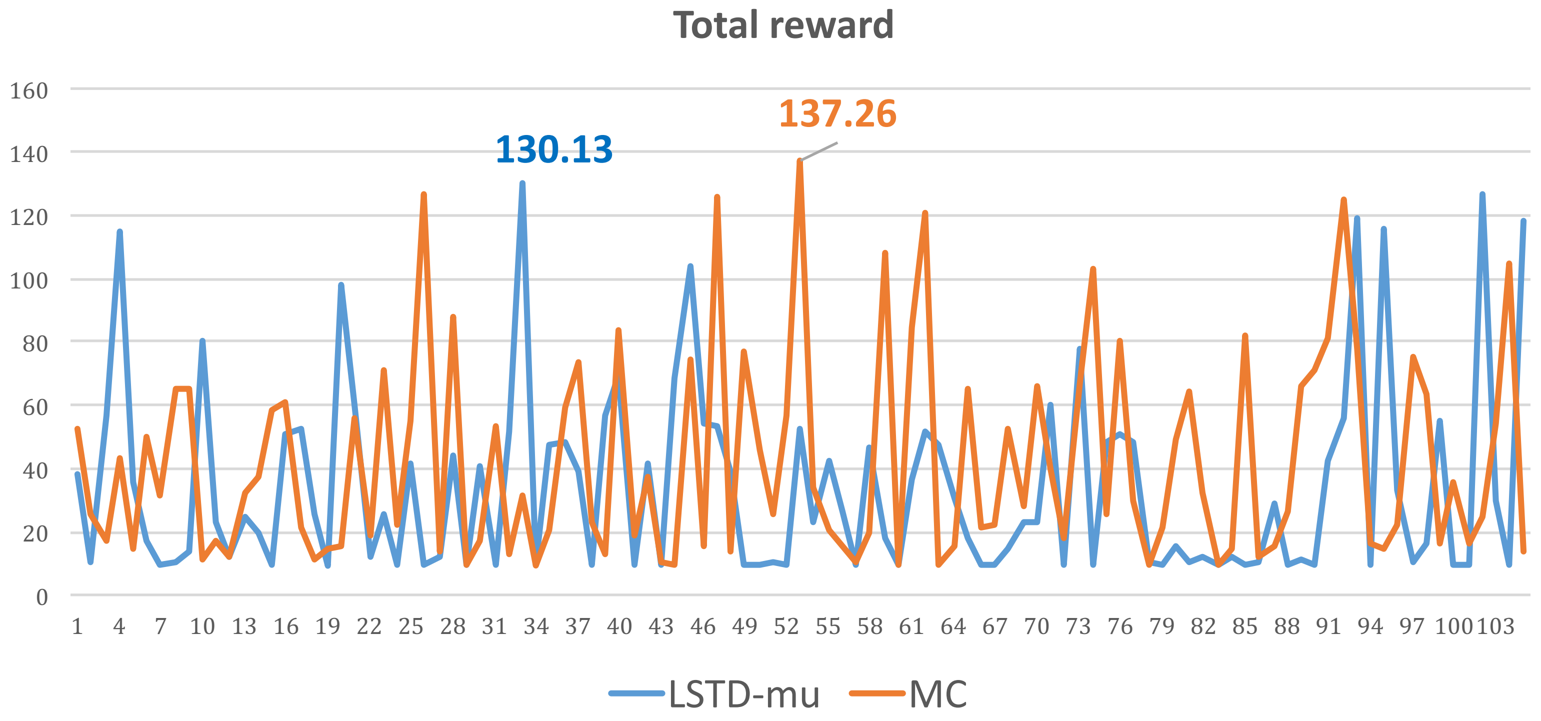
Reward Basis Feature Dimension(p) : 9

#Expert Trajectory : 100

# IRL + LSTD-mu

$$t^{(i)} = \left\| \mu^E(s_0) - \bar{\mu}^{(i-1)} \right\|_2$$




$$\mathcal{H}_\phi = \{\theta^T\phi(s), \theta \in \mathbb{R}^p\},\ |\phi_i(s)| \leq 1, \forall s \in S, 1 \leq i \leq p.$$

Reward Basis Feature Dimension(p) : 9

#Expert Trajectory : 100

Merit : Fast! (x8 ~ x16)

# Given information, Expert Trajectories.

Expert Trajectory,

$$T_i = \{(s_t, a_t, s_{t+1})\ _{1 \leq t \leq Done}\}$$

Expert Trajectories,

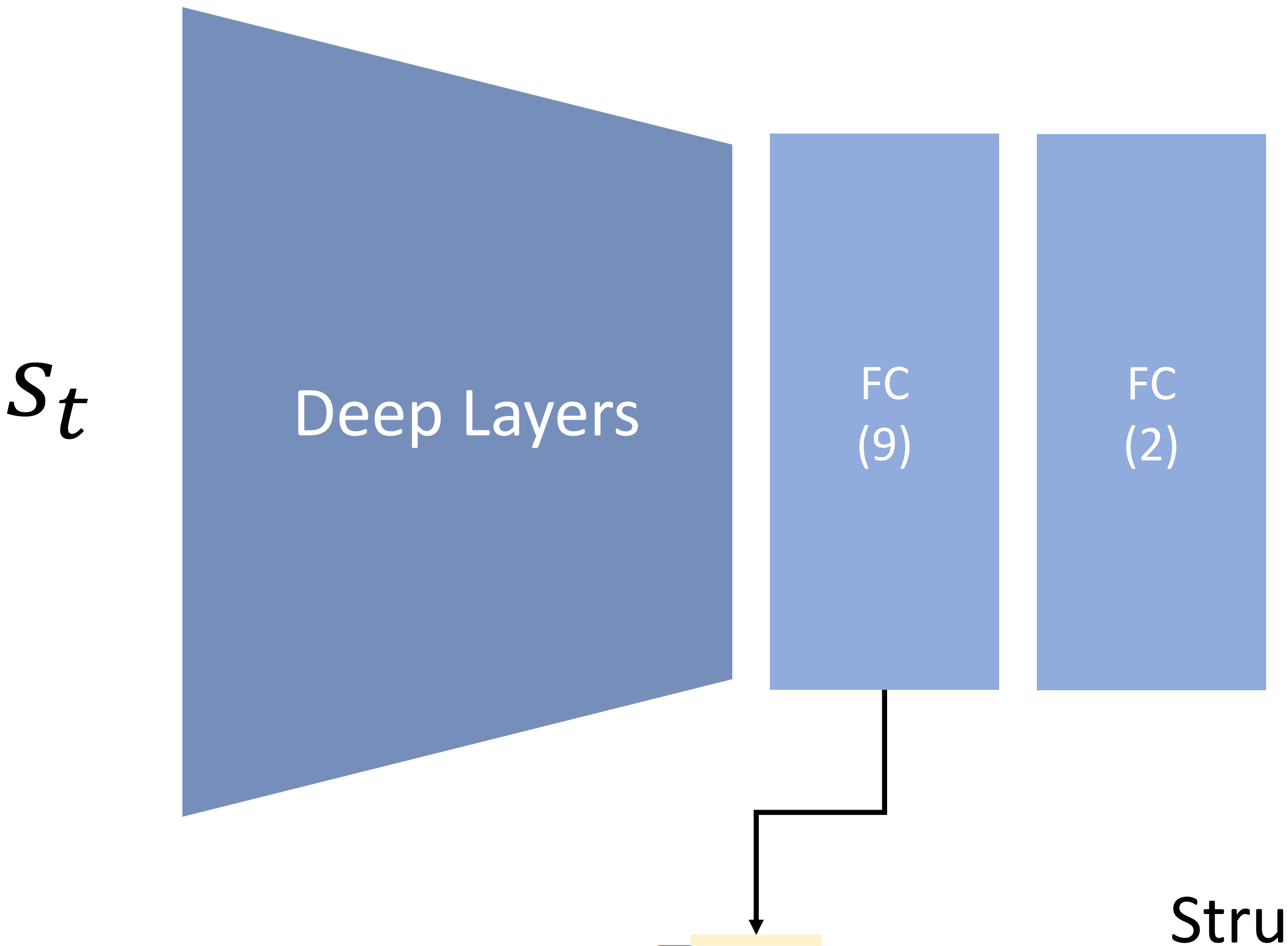
$$T_{collection} = \{T_i, 1 \leq i \leq n\}$$

Supervised Action Estimator? (not DQN but quite similar)

$$s_t(input) \rightarrow a_t(output)?$$

# Deep Action Network(DAN)

$\text{Dataset} = \{(s_{i,t}, a_{i,t})\}\ \text{i} \in \text{Trajectories},\ 1 \leq t \leq Done$




one hot vector

$$\hat{a}_t = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.1 \\ 0 \end{bmatrix} \longleftrightarrow a_t = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$Loss(w) = \sum (\hat{a}_t - a_t)^2$$

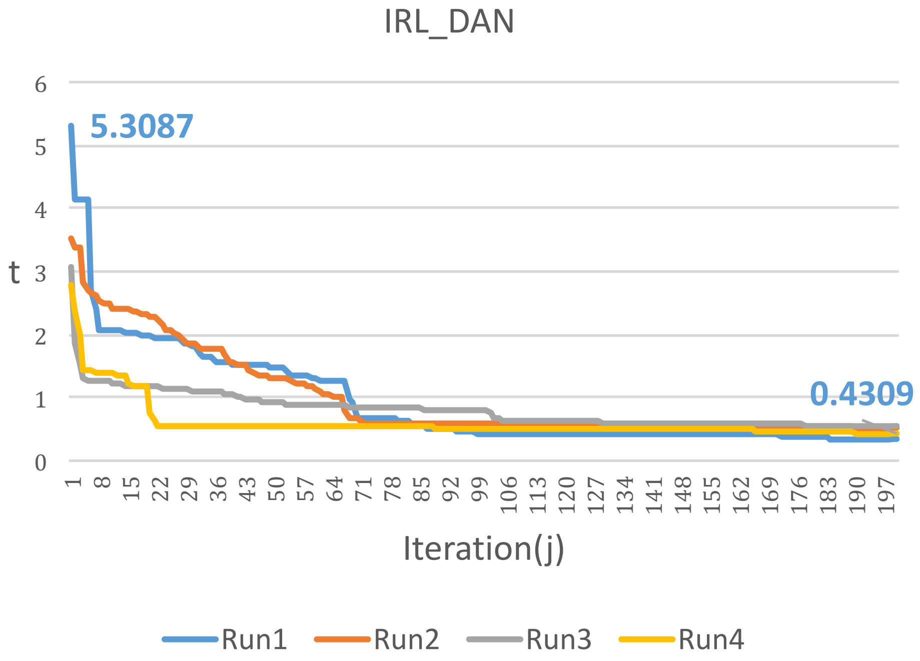
$$R^*(s) = (\theta^*)^T \phi(s)$$

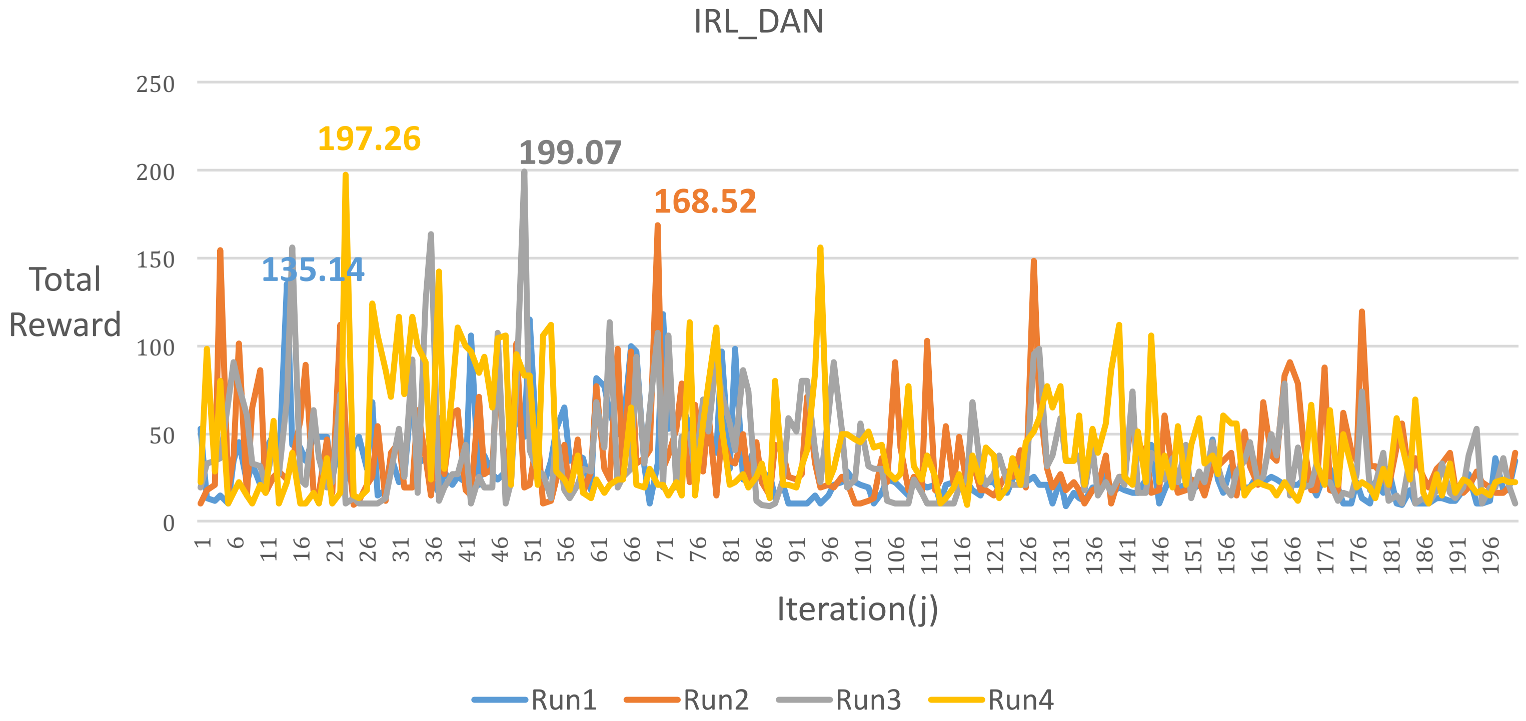
Structure for CartPole

4(input) - 20(fc1) - 9(fc2) - 2(fc3)

# IRL + MC approach + Deep Action Network

$$t^{(i)} = \left\| \mu^E(s_0) - \bar{\mu}^{(i-1)} \right\|_2$$



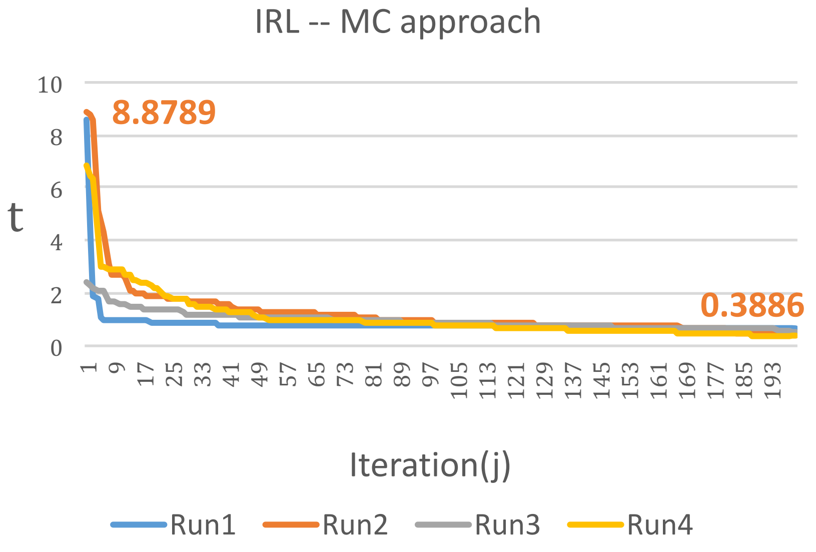

Deep Action Network 4(input) - 20(fc1) - 9(fc2) - 2(output)

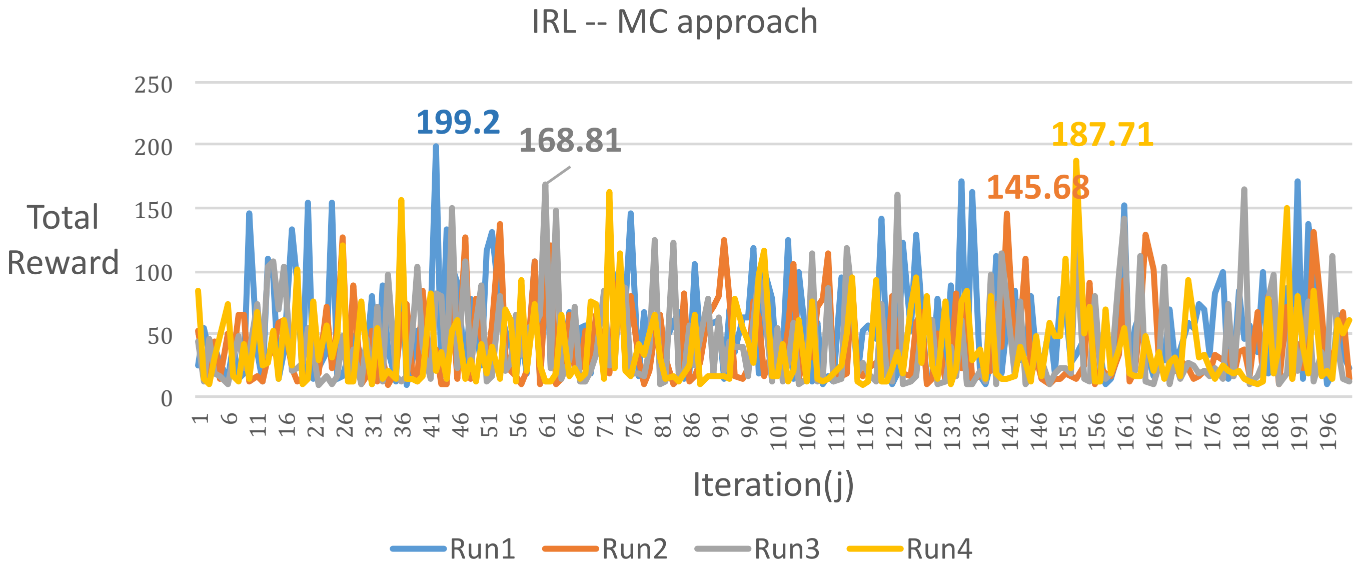
Reward Basis Feature Dimension(p) : 9

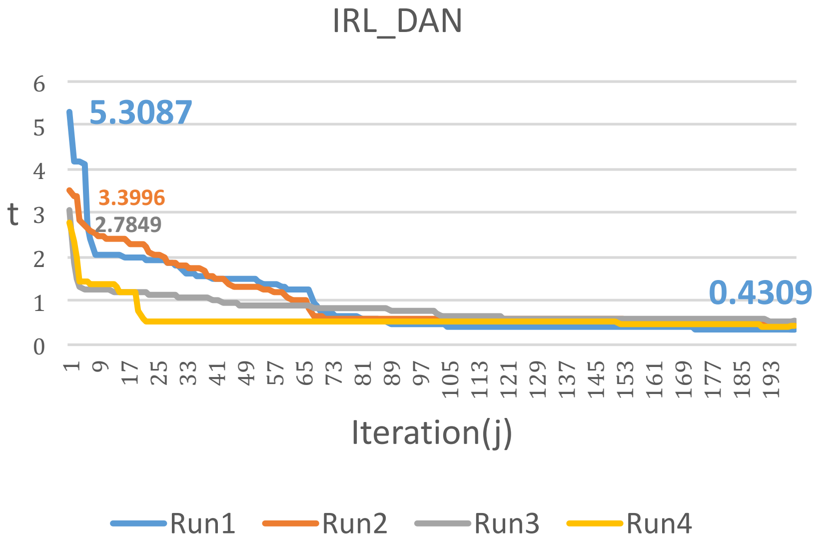
#Expert Trajectory : 100

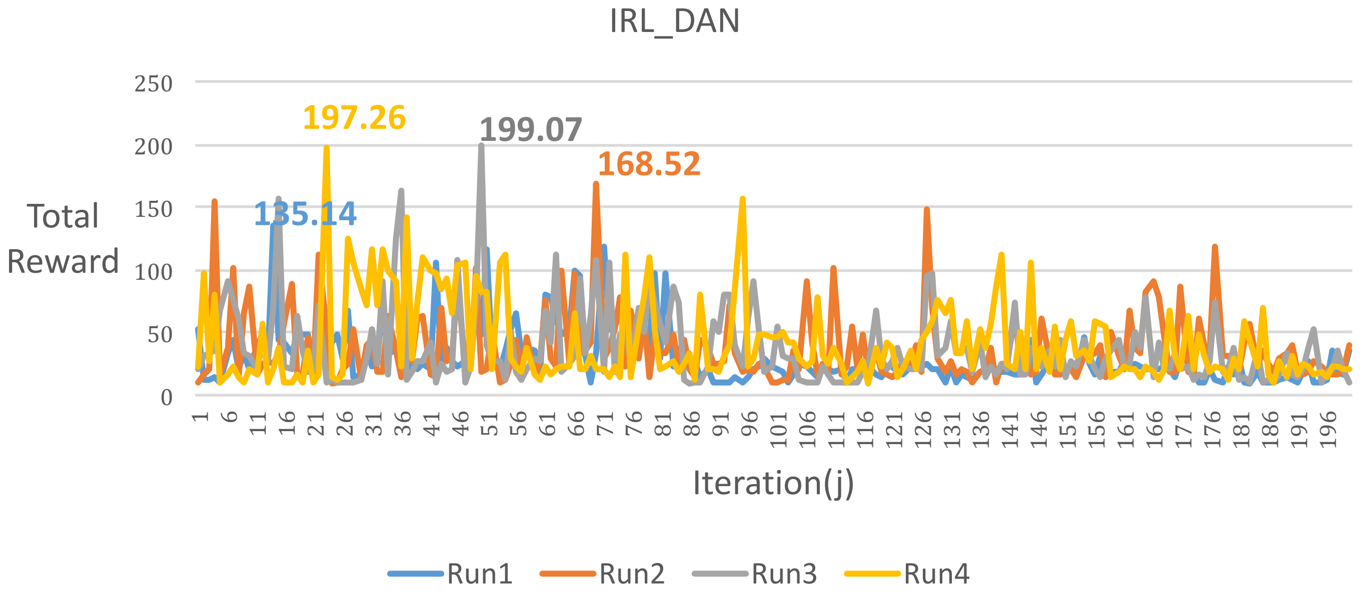
# IRL vs. IRL_DAN

$$t^{(i)} = \left\| \mu^E(s_0) - \bar{\mu}^{(i-1)} \right\|_2$$



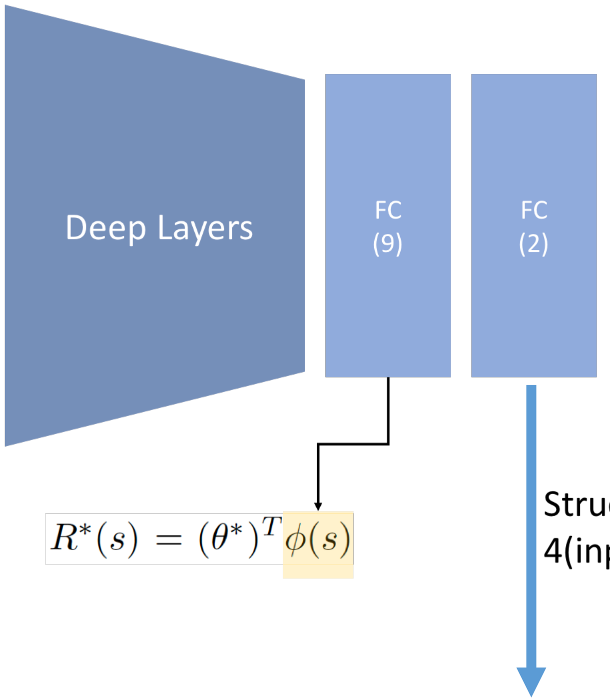



# What I tried, (Deep Reward Network)

DeepActionNetwork

$s_t$




one hot vector

| $\hat{a}_t$ | $a_t$ |
|---|---|
| 0.1 | 0 |
| 0.8 | 1 |
| 0.1 | 0 |
| 0 | 0 |

$Loss(w) = \sum(\hat{a}_t - a_t)^2$

$R^*(s) = (\theta^*)^T \phi(s)$

Structure for CartPole
4(input) - 20(fc1) - 9(fc2) - 2(fc3)

q(s, a=0)

q(s, a=1)

Now, we know $a_t$(only for $s_t$)

$$s_t \rightarrow a_t \rightarrow s_{t+1} \rightarrow a_{t+1} \dots$$

$$Dataset = \{(s_{i,t}, a_{i,t}, s_{i,t+1})\ _{i \in Traj, 1 \le t \le Done}\}$$

$$q(s,a) = r(s,a) + \gamma\ q(s', \pi(s')) \quad \text{(by Bellman equation)}$$

$$r(s,a) = q(s,a) - \gamma\ q(s', \pi(s'))$$

$$r(s,a) = \underbrace{q(s,a)}_{\text{known}} - \gamma\ \max_a \underbrace{q(s',a')}_{\text{known}}$$

$\hat{r}_t$?

Cartpole is a little tricky to apply DRN since all reward is 1 until “done”.

# Next steps, (in order of importance)

1. Fix LSPI problem(oscillation)
2. Analyze Deep Reward Network
3. IRL + DQN (as MDP solver) or apply other Deep RL
4. Other simulator (CartPole might be easy)
5. Fully connected layer → CNN (input : env.render() )
6. Projection method → Quadratic programming
7. Apply deep features for LSTD-mu

The code is available on the my github page( https://github.com/jeonggwanlee/LSTD-mu)

# Backup slide

```
Find Best Agent iteration : 20/30
99's agent ####################9.41
mu_diff  [0.61039731 0.56565781 0.79287856 0.95013813 0.87483152 0.6577403
 0.69860213 0.55918665 1.27960358]
threshold:  1.050349088643595
threshold_gap: -0.004185
iteration:  100
Find Best Agent iteration : 0/30
Find Best Agent iteration : 20/30
100's agent ####################25.49
mu_diff  [-35.94200319 -26.17849604  -5.04567943 -27.03646621 -10.881589
 -15.69293346 -29.36636834 -23.92220604 -21.39544313]
threshold:  1.049606184474631
threshold_gap: -0.000743
iteration:  101
Find Best Agent iteration : 0/30
Find Best Agent iteration : 20/30
101's agent ####################11.3
mu_diff  [  4.99395696   3.37577399 -21.34116135   4.38247295   2.92195305
   2.58690356   2.79950747   1.75200989   3.47019326]
threshold:  0.9351382856972915
threshold_gap: -0.114468
iteration:  102
Find Best Agent iteration : 0/30
Find Best Agent iteration : 20/30
102's agent ####################9.94
mu_diff  [-1.19435403 -0.47307715  0.10268988 -0.54256935 -0.51734355 -1.09139626
 -2.11042581 -2.67381087 -0.23387989]
threshold:  0.8565909493870978
threshold_gap: -0.078547
iteration:  103
Find Best Agent iteration : 0/30
```

# Backup slide

Add # expert trajectory experiment!

Add DRN experiment!
영상 찍기