Signal and Image Processing Group, Space Applications Centre, Indian Space Research Organisation, Ahmedabad, India - 380015.

Monte-Carlo Siamese Policy on Actor for Satellite Image Super Resolution

IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshop

Litu Rout*, Saumyaa Shah, S Manthira Moorthi and Debajyoti Dhar
*Correspondence to lr@sac.isro.gov.in

April 17, 2020

Content

Introduction

- Supervised and adversarial learning have been widely adopted in various vision tasks.
- Can another branch of AI, commonly known as Reinforcement Learning (RL) benefit such tasks?
- Explore plausible usage of RL in super resolution of remote sensing imagery

What is reinforcement learning?

Reinforcement learning is a sequential decision making process that focuses on maximizing long-term expected return by interacting with the environment iteratively.

Introduction

- Supervised and adversarial learning have been widely adopted in various vision tasks.
- Can another branch of AI, commonly known as Reinforcement Learning (RL) benefit such tasks?
- Explore plausible usage of RL in super resolution of remote sensing imagery

What is reinforcement learning?

Reinforcement learning is a sequential decision making process that focuses on maximizing long-term expected return by interacting with the environment iteratively.

Introduction

- Supervised and adversarial learning have been widely adopted in various vision tasks.
- Can another branch of AI, commonly known as Reinforcement Learning (RL) benefit such tasks?
- Explore plausible usage of RL in super resolution of remote sensing imagery

What is reinforcement learning?

Reinforcement learning is a sequential decision making process that focuses on maximizing long-term expected return by interacting with the environment iteratively.

Introduction

- Supervised and adversarial learning have been widely adopted in various vision tasks.
- Can another branch of AI, commonly known as Reinforcement Learning (RL) benefit such tasks?
- Explore plausible usage of RL in super resolution of remote sensing imagery

What is reinforcement learning?

Reinforcement learning is a sequential decision making process that focuses on maximizing long-term expected return by interacting with the environment iteratively.

Introduction

- Supervised and adversarial learning have been widely adopted in various vision tasks.
- Can another branch of AI, commonly known as Reinforcement Learning (RL) benefit such tasks?
- Explore plausible usage of RL in super resolution of remote sensing imagery

What is reinforcement learning?

Reinforcement learning is a sequential decision making process that focuses on maximizing long-term expected return by interacting with the environment iteratively.

- A straightforward implementation of RL is *not* adequate.

- Action variables are not fully known in most real-world environments.

- One way is to parameterize action variables by matrices, and train our policy network using Monte-Carlo sampling.

- A straightforward implementation of RL is *not* adequate.
- Action variables are not fully known in most real-world environments.
- One way is to parameterize action variables by matrices, and train our policy network using Monte-Carlo sampling.

- A straightforward implementation of RL is *not* adequate.
- Action variables are not fully known in most real-world environments.
- One way is to parameterize action variables by matrices, and train our policy network using Monte-Carlo sampling.

Preliminaries and Notations

- **Markov Property:** Future is independent of past given present, i.e., $P(s_{t+1}|s_0, s_1, \ldots, s_t) = P(s_{t+1}|s_t)$.

- **Markov Decision Process (MDP):** MDP is defined as a tuple (S, A, R, P, γ), where S is the continuous or discrete state space, A is the continuous or discrete action space, R is the immediate reward function, P is the transition probability, and $\gamma \in (0, 1)$ is the discount factor.

- A sample trajectory, $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1})$ sampled from policy π.

Preliminaries and Notations

- **Markov Property:** Future is independent of past given present, i.e., $P(s_{t+1}|s_0, s_1, \ldots, s_t) = P(s_{t+1}|s_t)$.

- **Markov Decision Process (MDP):** MDP is defined as a tuple (S, A, R, P, γ), where S is the continuous or discrete state space, A is the continuous or discrete action space, R is the immediate reward function, P is the transition probability, and $\gamma \in (0, 1)$ is the discount factor.

- A sample trajectory, $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1})$ sampled from policy π.

Preliminaries and Notations

- **Markov Property:** Future is independent of past given present, i.e., $P(s_{t+1}|s_0, s_1, \ldots, s_t) = P(s_{t+1}|s_t)$.

- **Markov Decision Process (MDP):** MDP is defined as a tuple (S, A, R, P, γ), where S is the continuous or discrete state space, A is the continuous or discrete action space, R is the immediate reward function, P is the transition probability, and $\gamma \in (0, 1)$ is the discount factor.

- A sample trajectory, $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1})$ sampled from policy π.

Preliminaries and Notations

Goal: To find an optimal policy π^* that maximizes its expected reward,

$$\pi^* = \arg\max_{\pi \in \Pi} \mathcal{J}(\pi), \tag{1}$$

where Π is the set of policies and $\mathcal{J}(\pi)$ is the policy evaluation metric defined by,

$$\mathcal{J}(\pi) = \mathbb{E}_{\tau \in \pi}\left[\sum_{t=1}^{T+1} \gamma^{t-1} r_t\right]. \tag{2}$$

Here, T represents the time step of terminal state in each episode and τ is the trajectory.

Preliminaries and Notations

Policy Gradient: Policy, π is parameterized by θ where the objective is to find optimal set of parameters θ^* that maximizes expected reward,

$$\theta^* = \arg\max_{\theta} \mathcal{J}(\theta), \tag{3}$$

$$\mathcal{J}(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \pi_\theta(s, a) R_{s,a}, \tag{4}$$

where $d^{\pi_\theta}(s)$ is a stationary distribution of Markov chain for π_θ and $R_{s,a}$ is the reward function for state s and action a.

Famous Likelihood Trick: Parameters are updated by $\theta \leftarrow \theta + \Delta\theta$, where $\Delta\theta$ is computed as

$$\Delta\theta = \nabla_\theta \mathcal{J}(\theta) = \mathbb{E}\left[R_{s,a} \nabla_\theta \log \pi_\theta(s, a)\right]. \tag{5}$$

Preliminaries and Notations

Policy Gradient: Policy, π is parameterized by θ where the objective is to find optimal set of parameters θ^* that maximizes expected reward,

$$\theta^* = \arg\max_{\theta} \mathcal{J}(\theta), \tag{3}$$

$$\mathcal{J}(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \pi_\theta(s, a) R_{s,a}, \tag{4}$$

where $d^{\pi_\theta}(s)$ is a stationary distribution of Markov chain for π_θ and $R_{s,a}$ is the reward function for state s and action a.

Famous Likelihood Trick: Parameters are updated by $\theta \leftarrow \theta + \Delta\theta$, where $\Delta\theta$ is computed as

$$\Delta\theta = \nabla_\theta \mathcal{J}(\theta) = \mathbb{E}\left[R_{s,a} \nabla_\theta \log \pi_\theta(s, a)\right]. \tag{5}$$

Feature Extractor Network, $\Phi(s)$ parameterized by θ_f operates on each state, $s \in \mathbb{R}^{H\times W\times C}$,

$$\tilde{s} = \Phi(s;\theta_f),\ \tilde{s} \in \mathbb{R}^{H\times W\times \tilde{C}}, \tag{6}$$

where $H,\ W,\ C,$ and $\tilde{C}$ represent height, width, input channels, and number of feature maps, respectively.

The output of neural network, $\Phi(s;\theta_f)$ is computed by,

$$\Phi(s;\theta_f) := FE_n\left(FE_{n-1}\left(\ldots\left(FE_0\left(s\right)\right)\right)\right), \tag{7}$$

where FE represents Feature Extraction block.

Proposed Methodology

Representation Learning

Feature Extractor Network, $\Phi(s)$ parameterized by θ_f operates on each state, $s \in \mathbb{R}^{H \times W \times C}$,

$$\tilde{s} = \Phi(s; \theta_f), \ \tilde{s} \in \mathbb{R}^{H \times W \times \tilde{C}}, \tag{6}$$

where $H, \ W, \ C,$ and $\tilde{C}$ represent height, width, input channels, and number of feature maps, respectively.

The output of neural network, $\Phi(s; \theta_f)$ is computed by,

$$\Phi(s; \theta_f) := FE_n\left(FE_{n-1}\left(\ldots\left(FE_0\left(s\right)\right)\right)\right), \tag{7}$$

where FE represents Feature Extraction block.

Actor Network (AN), $\Omega_{\theta_a}(.)$ parameterized by θ_a performs parametric actions on the latent representation of state space, $\tilde{s}$.

$$RB(x) = x + \lambda h(x), \tag{8}$$

Agent performs sequence of actions, $a_n^{RB}(.)$ and the intermediate states are computed by,

$$\tilde{s}_n = a_n^{RB}(\tilde{s}_{n-1}),\ n = 1, 2, \ldots, N, \tag{9}$$

where N represents total number of action variables in our MDP.

Actor Network (AN), $\Omega_{\theta_a}(.)$ parameterized by θ_a performs parametric actions on the latent representation of state space, $\tilde{s}$.

$$RB(x) = x + \lambda h(x), \tag{8}$$

Agent performs sequence of actions, $a_n^{RB}(.)$ and the intermediate states are computed by,

$$\tilde{s}_n = a_n^{RB}(\tilde{s}_{n-1}),\ n = 1, 2, \ldots, N, \tag{9}$$

where N represents total number of action variables in our MDP.

Transition Blocks (TB) map latent space into state space.

$$\hat{s} = TB_m\left(TB_{m-1}\left(\ldots\left(TB_0\left(\tilde{s}_N\right)\right)\right)\right) \tag{10}$$

NB: Each TB consists of one convolution and one LeakyReLU unit.

Transition Blocks (TB) map latent space into state space.

$$\hat{s} = TB_m\left(TB_{m-1}\left(\ldots\left(TB_0\left(\tilde{s}_N\right)\right)\right)\right) \tag{10}$$

NB: Each TB consists of one convolution and one LeakyReLU unit.

Siamese Policy Network (SPN) estimates their discrepancy,

$$\Psi_{\theta_p}(\hat{s}, s^*) = \Phi_{\theta_p}(\hat{s}) * \Phi_{\theta_p}(s^*) + b, \tag{11}$$

where $b \in \mathbb{R}$ and $\Phi_{\theta_p}(.)$ represents the CNN in each branch with shared parameters θ_p.

Probabilistic confidence:

$$\pi_{\theta_p}(s, a) = \frac{1}{1 + \exp(-\Psi_{\theta_p}(\hat{s}, s^*))}. \tag{12}$$

Siamese Policy Network (SPN) estimates their discrepancy,

$$\Psi_{\theta_p}(\hat{s}, s^*) = \Phi_{\theta_p}(\hat{s}) * \Phi_{\theta_p}(s^*) + b, \tag{11}$$

where $b \in \mathbb{R}$ and $\Phi_{\theta_p}(.)$ represents the CNN in each branch with shared parameters θ_p.

Probabilistic confidence:

$$\pi_{\theta_p}(s, a) = \frac{1}{1 + \exp(-\Psi_{\theta_p}(\hat{s}, s^*))}. \tag{12}$$

***Lemma I**: Training AN*

Let $\theta_{fa} = \{\theta_f, \theta_a\}$ and $\mathcal{J}(\theta_{fa})$ denote the expected return accumulated by the agent with a given policy π_{θ_p},

$$\mathcal{J}(\theta_{fa}) = \mathbb{E}\left[R_{s,a}\right] = \mathbb{E}\left[-(\hat{s} - s^*)^2\right]. \quad (13)$$

The parameters are updated by, $\theta_{fa} \leftarrow \theta_{fa} + \Delta\theta_{fa}$ where,

$$\Delta\theta_{fa} = \mathbb{E}\left[-2\left(\hat{s} - s^*\right)\ \nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right)\right]. \quad (14)$$

Here, $[m]$ represents a set of $\{0, 1, \ldots, m\}$. By stochastic gradient ascent, the update equation becomes

$$\Delta\theta_{fa} = -\alpha\left(\hat{s} - s^*\right)\nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right), \quad (15)$$

where α denotes step size.

***Lemma I**: Training AN*

Let $\theta_{fa} = \{\theta_f, \theta_a\}$ and $\mathcal{J}(\theta_{fa})$ denote the expected return accumulated by the agent with a given policy π_{θ_p},

$$\mathcal{J}(\theta_{fa}) = \mathbb{E}\left[R_{s,a}\right] = \mathbb{E}\left[-(\hat{s} - s^*)^2\right]. \tag{13}$$

The parameters are updated by, $\theta_{fa} \leftarrow \theta_{fa} + \Delta\theta_{fa}$ where,

$$\Delta\theta_{fa} = \mathbb{E}\left[-2\left(\hat{s} - s^*\right)\ \nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right)\right]. \tag{14}$$

Here, $[m]$ represents a set of $\{0, 1, \ldots, m\}$. By stochastic gradient ascent, the update equation becomes

$$\Delta\theta_{fa} = -\alpha\left(\hat{s} - s^*\right)\nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right), \tag{15}$$

where α denotes step size.

***Lemma I**: Training AN*

Let $\theta_{fa} = \{\theta_f, \theta_a\}$ and $\mathcal{J}(\theta_{fa})$ denote the expected return accumulated by the agent with a given policy π_{θ_p},

$$\mathcal{J}(\theta_{fa}) = \mathbb{E}\left[R_{s,a}\right] = \mathbb{E}\left[-(\hat{s} - s^*)^2\right]. \tag{13}$$

The parameters are updated by, $\theta_{fa} \leftarrow \theta_{fa} + \Delta\theta_{fa}$ where,

$$\Delta\theta_{fa} = \mathbb{E}\left[-2\left(\hat{s} - s^*\right)\ \nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right)\right]. \tag{14}$$

Here, $[m]$ represents a set of $\{0, 1, \ldots, m\}$. By stochastic gradient ascent, the update equation becomes

$$\Delta\theta_{fa} = -\alpha\left(\hat{s} - s^*\right)\nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right), \tag{15}$$

where α denotes step size.

***Lemma II**: Training SPN*

Let $\mathcal{J}(\theta_p)$ denotes the expected return accumulated by the agent with fixed set of parameters (θ_{fa}), then

$$\mathcal{J}\left(\theta_p\right) = \mathbb{E}_{\theta_p}\left[r\right] = \sum_{s \in S} d^{\pi_\theta}\left(s\right) \sum_{a \in A} \pi_\theta\left(s, a\right) R_{s,a}. \tag{16}$$

Using stochastic gradient ascent, the parameters are updated using the famous likelihood trick as given by

$$\theta_p \leftarrow \theta_p + \Delta\theta_p,\ \Delta\theta_p = \nabla_{\theta_p}\mathcal{J}(\theta_p) = \beta R_{s,a}\nabla_{\theta_p}\log\pi_\theta\left(s, a\right), \tag{17}$$

where β denotes step size.

***Lemma II**: Training SPN*

Let $\mathcal{J}(\theta_p)$ denotes the expected return accumulated by the agent with fixed set of parameters (θ_{fa}), then

$$\mathcal{J}\left(\theta_p\right) = \mathbb{E}_{\theta_p}\left[r\right] = \sum_{s \in S} d^{\pi_\theta}\left(s\right) \sum_{a \in A} \pi_\theta\left(s, a\right) R_{s,a}. \tag{16}$$

Using stochastic gradient ascent, the parameters are updated using the famous likelihood trick as given by

$$\theta_p \leftarrow \theta_p + \Delta\theta_p,\ \Delta\theta_p = \nabla_{\theta_p}\mathcal{J}(\theta_p) = \beta R_{s,a} \nabla_{\theta_p} \log \pi_\theta\left(s, a\right), \tag{17}$$

where β denotes step size.

***Theorem I**: Training SPOA*
Let $\theta = \{\theta_f, \theta_a, \theta_p\}$ and $\mathcal{J}(\theta)$ denotes the expected return. The parameters of SPOA (θ) are updated by $\theta \leftarrow \theta + \Delta\theta$ where,

$$\Delta\theta = \Delta\theta_p + \Delta\theta_{fa}. \tag{18}$$

Please refer to our paper for a detailed proof of Theorem I.

Result: SPOA parameters, θ
initialize θ;
for *episode = 1,2,...,E* **do**
  initialize empty replay buffer $\mathbb{D}$;
  while $\mathbb{D}$ *not full* **do**
    Sample initial state, $s_0 \sim \mathbb{U}$;
    Sample corresponding goal state, s^*;
  end
  for *actor=1,2,...,A* **do**
    Take parametric sequential actions on $\mathbb{D}$;
    Compute $\Delta\theta_{fa} = -\alpha\left(\hat{s} - s^*\right)\nabla_{\theta_{fa}}\left(TB_{[m]}\left(\Omega_{\theta_a}\left(\Phi_{\theta_f}\left(s\right)\right)\right)\right)$;
    Update $\theta_{fa} \leftarrow \theta_{fa} + \Delta\theta_{fa}$;
  end
  for *policy=1,2,...,P* **do**
    Given actor parameters θ_{fa}, follow parametric policy $\pi_{\theta_p}(s, a)$ on $\mathbb{D}$;
    Compute $\Delta\theta_p = \beta R_{s,a}\nabla_{\theta_p}\log\pi_\theta\left(s, a\right)$;
    Update $\theta_p \leftarrow \theta_p + \Delta\theta_p$;
  end
  for *spoa=1,2,...,S* **do**
    Follow policy with implicit actions on $\mathbb{D}$;
    Compute new $\Delta\theta_p$ and $\Delta\theta_{fa}$;
    Compute $\Delta\theta = \Delta\theta_p + \Delta\theta_{fa}$;
    Update $\theta \leftarrow \theta + \Delta\theta$;
  end
end

Algorithm 1: Monte-Carlo Siamese Policy On Actor

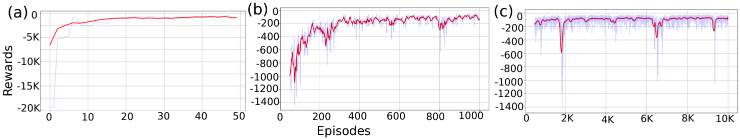

Figure: Learning dynamics. We use a forward window of size 10.

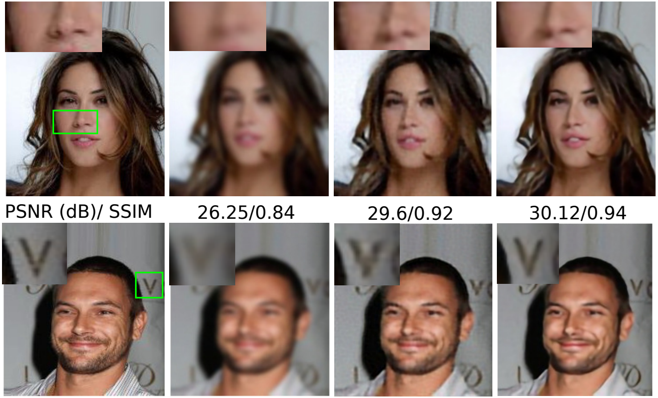

Figure: Qualitative analysis on CelebA. SPOA performs favourably against compared approaches.

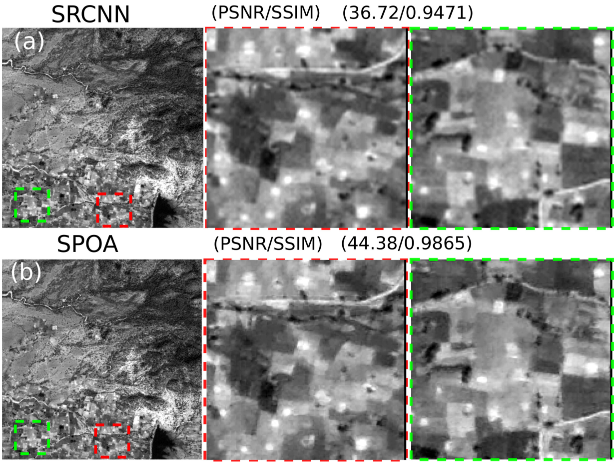

Figure: Qualitative analysis on IRS-1C. SPOA performs reasonably well on IRS-1C imagery.

Metrics	PSNR	SSIM	SRE	SAM	NIQE	Ma's	PI
BiCubic	57.51	0.9939	46.48	17.25	5.50	3.77	5.86
SRCNN [14]	59.15	0.9964	48.10	14.14	5.73	4.88	5.42
LapSRN [28]	59.31	0.9964	48.08	13.98	5.08	5.96	4.56
DRLN [2]	59.32	0.9964	48.10	13.97	4.21	6.03	4.08
SPOA(DRLN)	58.89	0.9960	47.94	14.69	3.65	6.60	3.52
SPOA(DRLN)+SA	59.33	0.9966	48.20	13.81	5.02	5.54	4.74
SPOA(DRLN)+SA+VGG	59.22	0.9963	48.23	14.13	4.30	6.20	4.05
SPOA(DRLN)+VGG	58.98	0.9961	47.94	14.60	4.16	6.56	3.80
GT	-	-	-	-	2.05	7.01	2.52

Table: Comparison with state-of-the-art methods.

Experiments
Analysis on WorldView-2

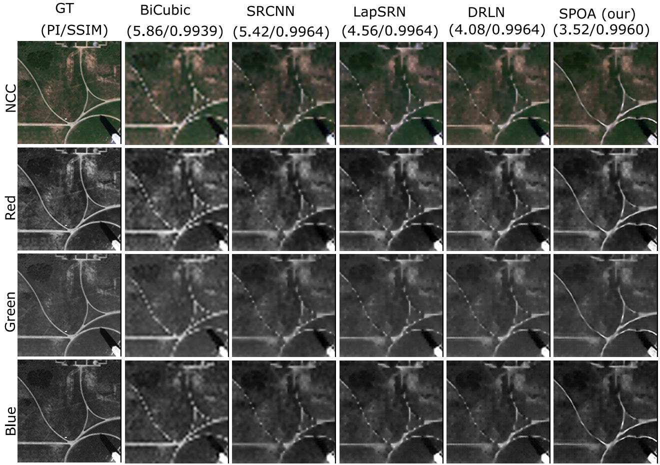

Figure: Qualitative analysis on WorldView-2.

Concluding Remarks

- Explored plausible usage of RL to address complex supervised problems.
- DRL based Monte-Carlo policy gradient approach to solve model-free MDPs.
- Theoretical results of Siamese policy network with implicit action space.
- Demonstrated in a super resolution environment where action variables are not apparent.
- Experimented on remote sensing and non-remote sensing imagery.

Concluding Remarks

- Explored plausible usage of RL to address complex supervised problems.
- DRL based Monte-Carlo policy gradient approach to solve model-free MDPs.
- Theoretical results of Siamese policy network with implicit action space.
- Demonstrated in a super resolution environment where action variables are not apparent.
- Experimented on remote sensing and non-remote sensing imagery.

Concluding Remarks

- Explored plausible usage of RL to address complex supervised problems.
- DRL based Monte-Carlo policy gradient approach to solve model-free MDPs.
- Theoretical results of Siamese policy network with implicit action space.
- Demonstrated in a super resolution environment where action variables are not apparent.
- Experimented on remote sensing and non-remote sensing imagery.

Concluding Remarks

- Explored plausible usage of RL to address complex supervised problems.
- DRL based Monte-Carlo policy gradient approach to solve model-free MDPs.
- Theoretical results of Siamese policy network with implicit action space.
- Demonstrated in a super resolution environment where action variables are not apparent.
- Experimented on remote sensing and non-remote sensing imagery.

Concluding Remarks

- Explored plausible usage of RL to address complex supervised problems.
- DRL based Monte-Carlo policy gradient approach to solve model-free MDPs.
- Theoretical results of Siamese policy network with implicit action space.
- Demonstrated in a super resolution environment where action variables are not apparent.
- Experimented on remote sensing and non-remote sensing imagery.

A Few Noteworthy Extensions

- ***Extension*** of SPOA to wide variety of problems currently solved using supervised learning.
- Explore broad ***spectrum of reinforcement learning*** algorithms in this framework.
- Study how well SPOA figures out matrix representation of actions by ***hiding*** known action variables in RL benchmarks.

A Few Noteworthy Extensions

- ***Extension*** of SPOA to wide variety of problems currently solved using supervised learning.
- Explore broad ***spectrum of reinforcement learning*** algorithms in this framework.
- Study how well SPOA figures out matrix representation of actions by ***hiding*** known action variables in RL benchmarks.

A Few Noteworthy Extensions

- ***Extension*** of SPOA to wide variety of problems currently solved using supervised learning.
- Explore broad ***spectrum of reinforcement learning*** algorithms in this framework.
- Study how well SPOA figures out matrix representation of actions by ***hiding*** known action variables in RL benchmarks.