Machine learning mechanism

- Define a function to be learned: $y^n = f(x^n)$
- Define a loss function $\mathcal{L}(f)$ to describe the error between y^n and $\hat{y}^n$
- Find the optimal parameters that minimize $\mathcal{L}(f)$

Machine learning mechanism – SL, SSL and RL

	Supervised Learning	Self-supervised Learning	Reinforcement Learning
1. Function to be learned	MLP, CNN families	AE/VAE, GAN	Actor
	$y = f(x)$	$\hat{x} = f(x)$	$a = f(s)$
2. Loss function $\mathcal{L}(f)$	MSE, CE	MSE, CE, KLD, JSD	MSE, KLD
3. Minimize $\mathcal{L}(f)$	Gradient decent, Maximum Likelihood		

After learned, what tasks can AI do?

	Supervised Learning	Self-supervised Learning	Reinforcement Learning
Function learned	MLP, CNN families	AE/VAE, GAN	Actor
	$y = f(x)$	$\hat{x} = f(x)$	$a = f(s)$
Intelligence	Recognition		Interact with dynamic, adversial environment
Tasks	• Regression • Classification • CV tasks, image classification, object detection, instance segmentation, subject tracking	• Feature extraction • Image generation • Anomaly detection	• Play chess • Play video game • Mobile robot that can play with elderly • Robot arm that can play with elderly

Challenges in learning $a=f(s)$

- Define a function to be learned: $a = f(s)$
- Define a loss function $\mathcal{L}(f)$ to describe the error between y^n and $\hat{y}^n$

  Time-delayed answer – If at time *t* we perform action a_t under state s_t, we only know the immediate reward r_t and there is a time delay to know the total accumulated reward $\hat{y}$.

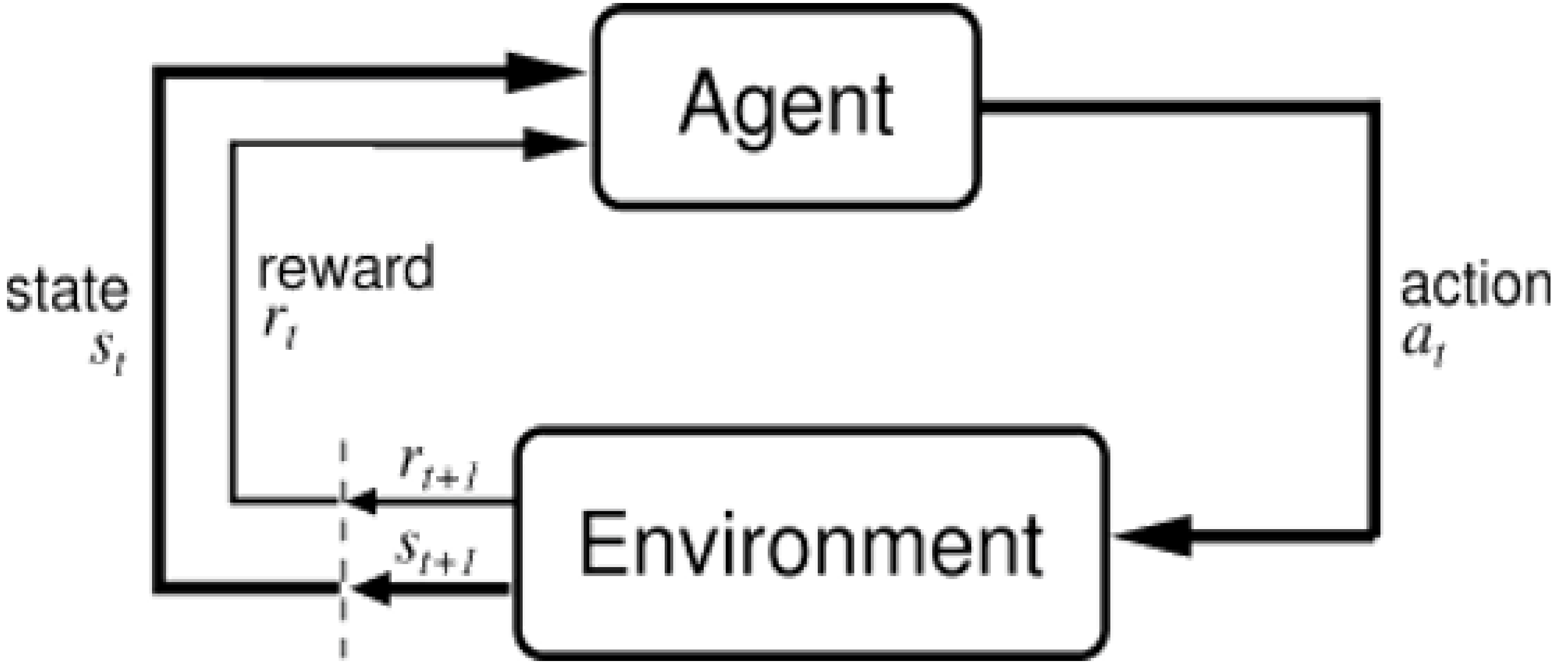
  Adversial interaction – After preforming a_t at state s_t, there are infinite number of possibilities for following state- actions $s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \cdots s_{t+T}, a_{t+T}$. It is difficult to estimate the true answer *y* (the true final reward).

- Find the optimal parameters that minimize $\mathcal{L}(f)$

Development

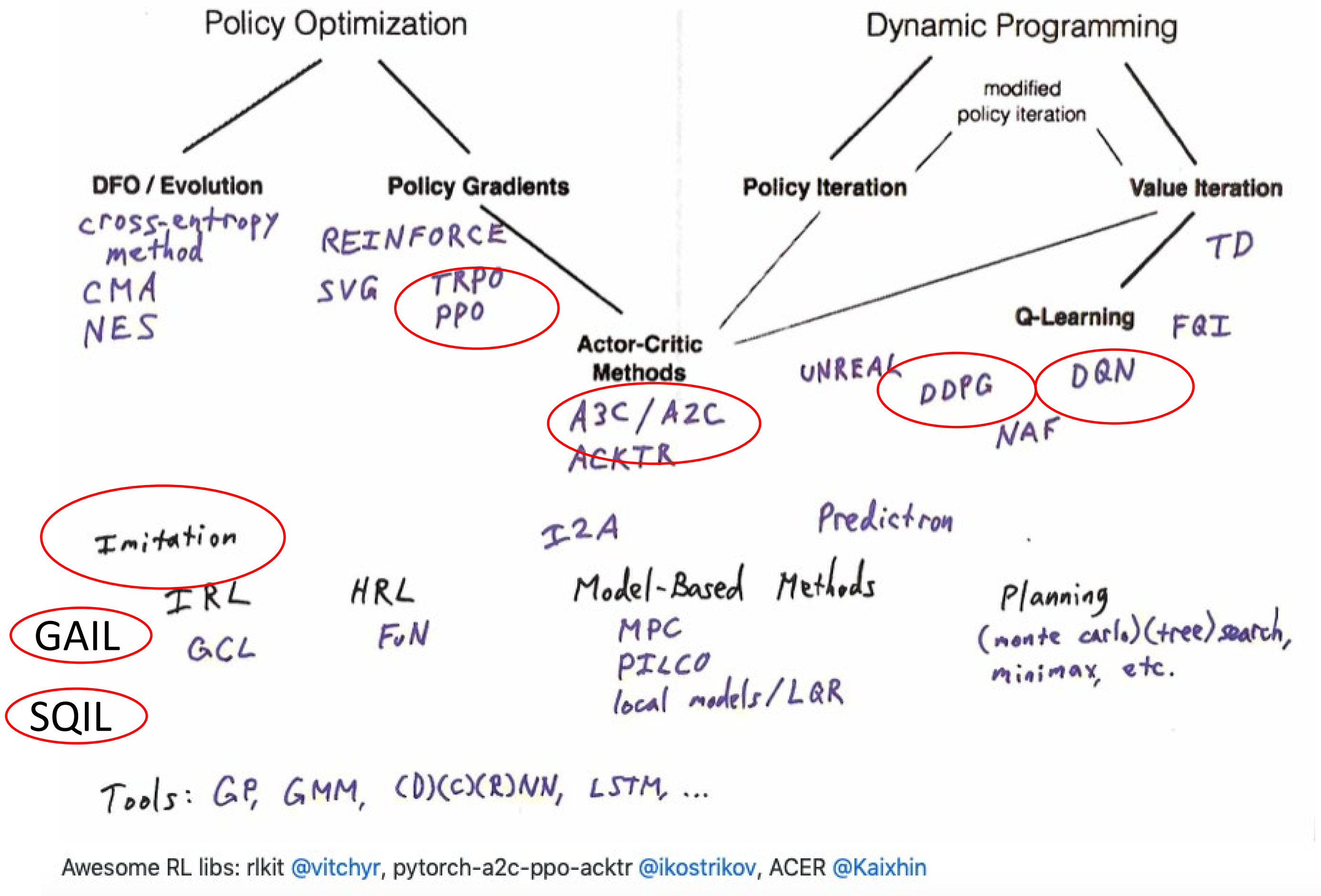
	Supervised Learning	Self-supervised Learning	Reinforcement Learning
Function learned	MLP, CNN families	AE/VAE, GAN	Actor
	$y = f(x)$	$\hat{x} = f(x)$	$a = f(s)$
Training concern	• Labelling cost	• Long training time (? epochs) • Difficult to train	• Long training time (1M ~5M steps) • Difficult to train
Development	• Integrate pre-trained models with application software, e.g., flask, app • Deployed to edge computing devices (Jetson Nano, Xavier)		• Virtual training environment development (Unity, ML Agent) • Reward engineering • Engineering of other training settings • Deploy to edge computing devices (Jetson Nano, Xavier) • Mobile robot and robot arm

Reinforcement learning



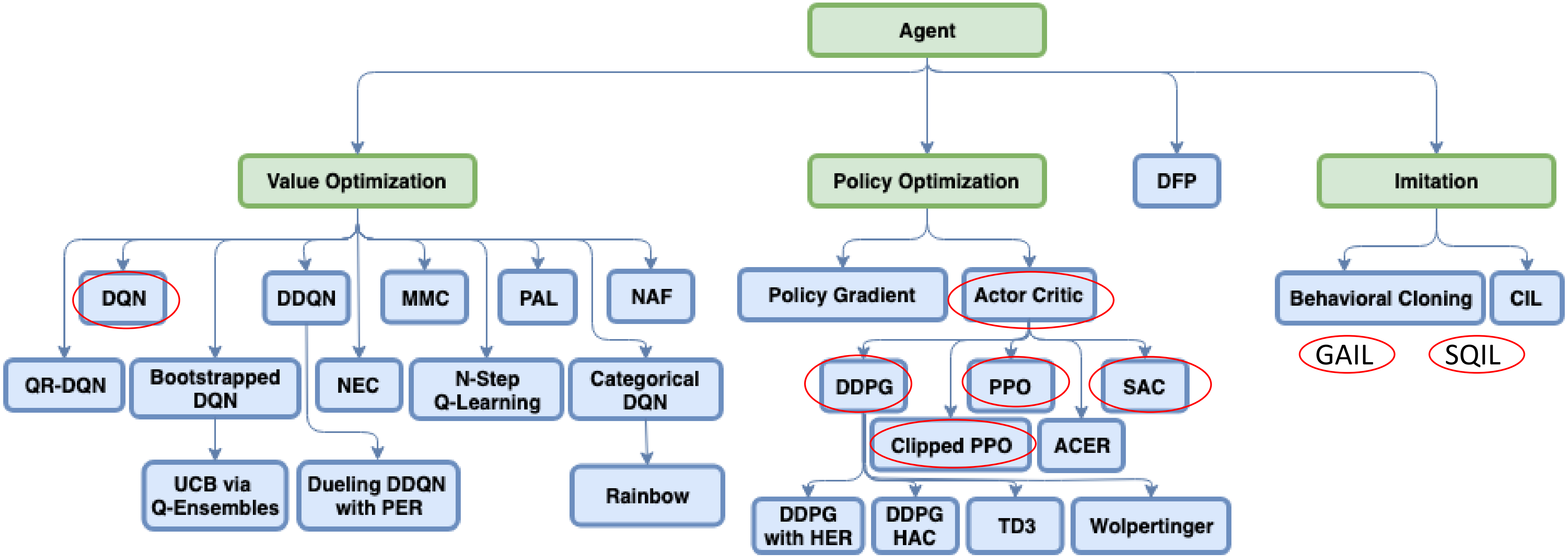
(Sutton and Barto, 1998)

Policy optimization vs dynamic programming approach to learn $a=f(s)$



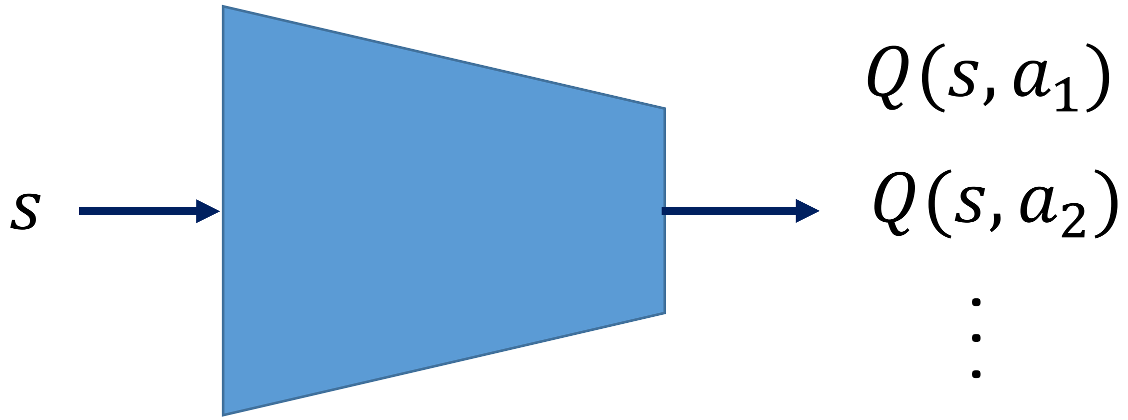
圖片來源: https://images.app.goo.gl/nPJftvi3CChYmyFDA

Policy optimization vs value optimization (DP)



圖片來源: https://nervanasystems.github.io/coach/selecting_an_algorithm.html

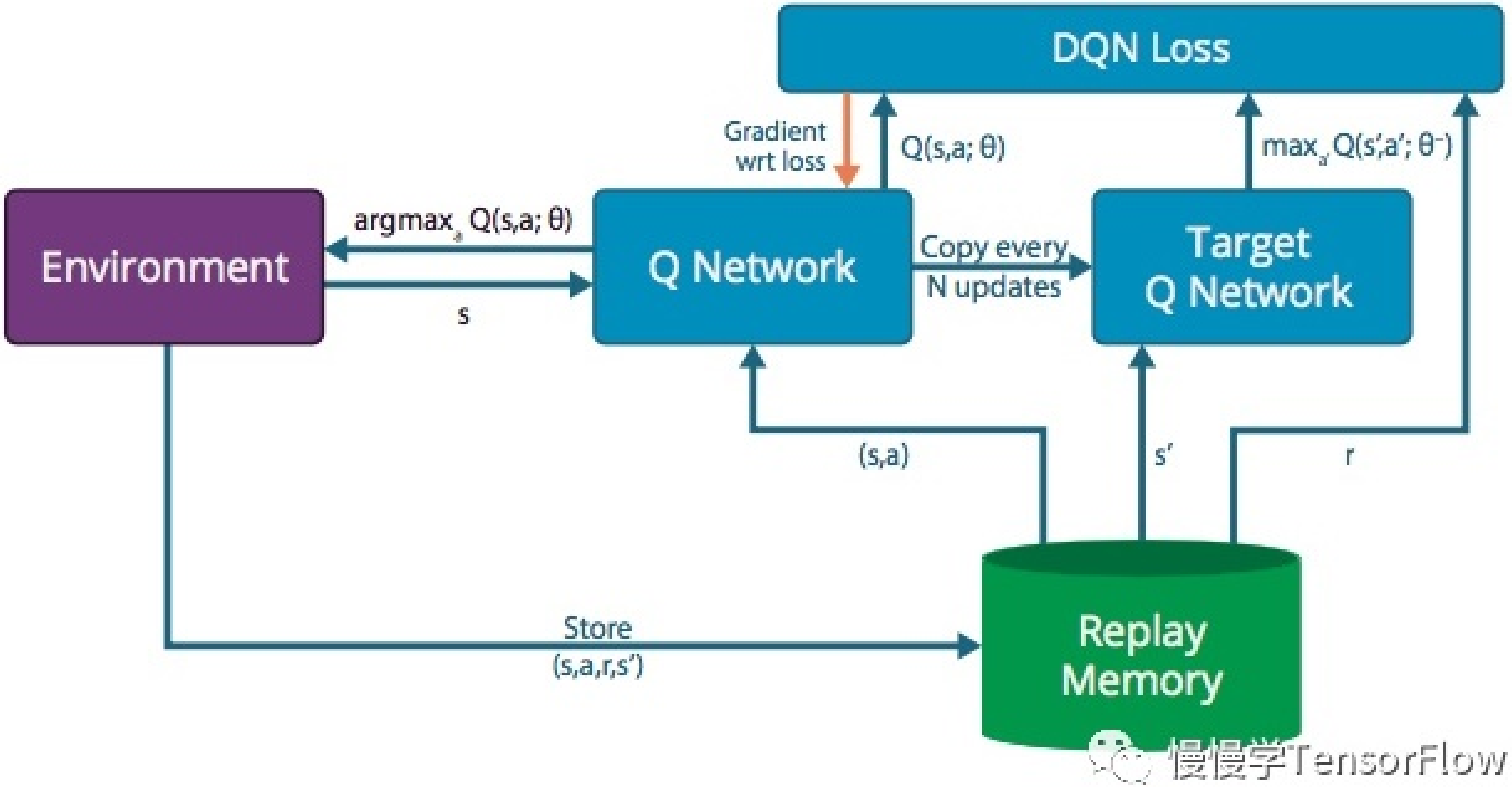
Deep Q-Network (DQN)



Bellman Equation:

$$Q^*(s, a) = \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

Deep Q-Network (DQN)



圖片來源: https://zhuanlan.zhihu.com/p/25546213?from_voters_page=true

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

Policy gradient

$$\tau = \left(s_1, a_1, r_1, s_2, a_2, r_{2,} \cdots s_{\mathrm{T}}, a_{\mathrm{T}}\right)$$

$$p_\theta(\tau) = p(s_1)p_\theta(a_1|s_1)p(s_2|s_1, a_1)p_\theta(a_2|s_2)p(s_3|s_2, a_2)\cdots$$

$$\mathrm{R}(\tau) = \sum_{\mathrm{t}=1}^{\mathrm{T}} \mathrm{r}_{\mathrm{t}}$$

$$\bar{R}_\theta = \sum R(\tau)\, p_\theta(\tau) = E_{\tau \sim p_\theta(\tau)}[R(\tau)]$$

Max $E[\bar{R}_\theta]$

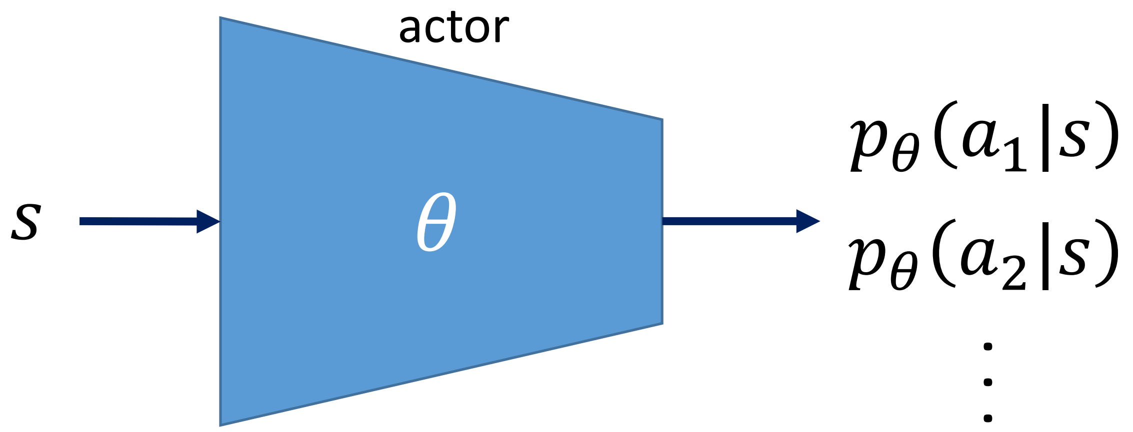
$$\max_\theta E[\bar{R}_\theta]$$

Gradient of the expected value

$$\nabla \bar{R}_\theta = \sum R(\tau)\, \nabla p_\theta(\tau) = E_{\tau \sim p_\theta(\tau)}[R(\tau) \nabla \log p_\theta(\tau)] \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla \log p_\theta(\tau^n)$$

$$= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$$

Use $\nabla \bar{R}_\theta$ to update policy network



$$\theta^{\pi'} \leftarrow \theta^{\pi} + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$$

Tips to improve bias and reduce variance of $\nabla \bar{R}_\theta$

$$\nabla \bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla \log p_\theta(a_t^n|s_t^n)$$

Add a baseline to calculate the reward

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} (R(\tau^n) - b)\nabla \log p_\theta(a_t^n|s_t^n), \qquad b \approx E[R(\tau)]$$

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \left(\sum\nolimits_{t'}^{T_n} r_{t'}^n - b\right)\nabla \log p_\theta(a_t^n|s_t^n)$$

Assign suitable time delayed credit

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \left(\sum\nolimits_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n - b\right)\nabla \log p_\theta(a_t^n|s_t^n), \gamma < 1$$

$$A^\theta(s_t, a_t) = \left(\sum\nolimits_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n - b\right)$$

Off-policy to improve efficiency of calculating $\nabla \bar{R}_\theta$

On-policy

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} A^\theta(s_t, a_t)\nabla \log p_\theta(a_t^n|s_t^n), \gamma < 1 \qquad A^\theta(s_t, a_t) = \left(\sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n - b\right)$$

Importance sampling

$$E_{x\sim \mathrm{p}}[f(x)] = E_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right]$$

$$Var_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right] = E_{x\sim \mathrm{q}}\left[\left(f(x)\frac{p(x)}{q(x)}\right)^2\right] - \left(E_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right]\right)^2$$

$$= E_{x\sim \mathrm{p}}\left[f(x)^2\frac{p(x)}{q(x)}\right] - \left(E_{x\sim \mathrm{p}}[f(x)]\right)^2$$

Off-policy

$$\nabla \bar{R}_\theta = E_{(s_t,a_t)\sim \pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t)\nabla \log p_\theta(a_t^n|s_t^n)\right]$$

From $\nabla \bar{R}_\theta$ to loss function

Off-policy

$$\nabla \bar{R}_\theta = E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\nabla \log p_\theta(a_t^n|s_t^n)\right]$$

Sampling efficiency

Loss function

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

Proximal policy optimization (PPO)

$$J_{PPO}^{\theta'}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta,\theta')$$

$$J_{PPO2}^{\theta'}(\theta) = \sum_{(s_t,a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right)A^{\theta'}(s_t,a_t)\right)$$

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Actor-critic strategy to calculate $\nabla\bar{R}_\theta$

$$\nabla\bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'}^{T_n}\gamma^{t'-t}r_{t'}^n - b\right)\nabla\log p_\theta(a_t^n|s_t^n)$$

$$G_t^n = \sum_{t'}^{T_n}\gamma^{t'-t}r_{t'}^n$$

unstable when sampling amount is not large enough

$$V^{\pi_\theta}(s_t^n)$$

Expected value of b

Use expected value to reduce sampling variance

$$\nabla\bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'}^{T_n}\gamma^{t'-t}r_{t'}^n - b\right)\nabla\log p_\theta(a_t^n|s_t^n)$$

$$E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n)$$

Expected value of G_t^n

Use one neural network that estimates V

$$Q^{\pi_\theta}(s_t^n, a_t^n) = \mathrm{E}[r_t^n + V^{\pi_\theta}(s_{t+1}^n)] = r_t^n + V^{\pi_\theta}(s_{t+1}^n)$$

$$Q^{\pi_\theta}(s_t^n, a_t^n) - V^{\pi_\theta}(s_t^n) = r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)$$

$$\boxed{A^\theta(s_t, a_t) = \left(r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)}$$

Use temporal difference to calculate V

$$A^{\theta}(s_t, a_t) = \left(r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)$$

Monte-Carlo approach

$V^{\pi_\theta}(\, s_a) \leftrightarrow G_a$

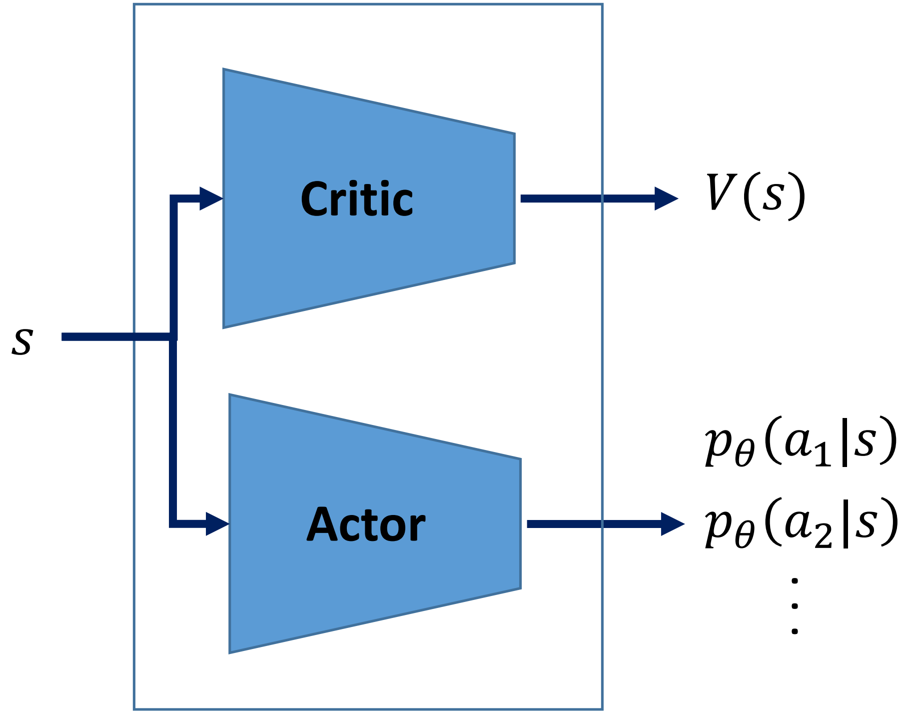
Until the end of the episode, the cumulated reward is G_a

Temporal-difference approach

$V^{\pi_\theta}(\, s_t) + r_t = V^{\pi_\theta}(s_{t+1})$

$V^{\pi_\theta}(\, s_t) - V^{\pi_\theta}(s_{t+1}) \;\leftrightarrow r_t$

Train the network



TD Error

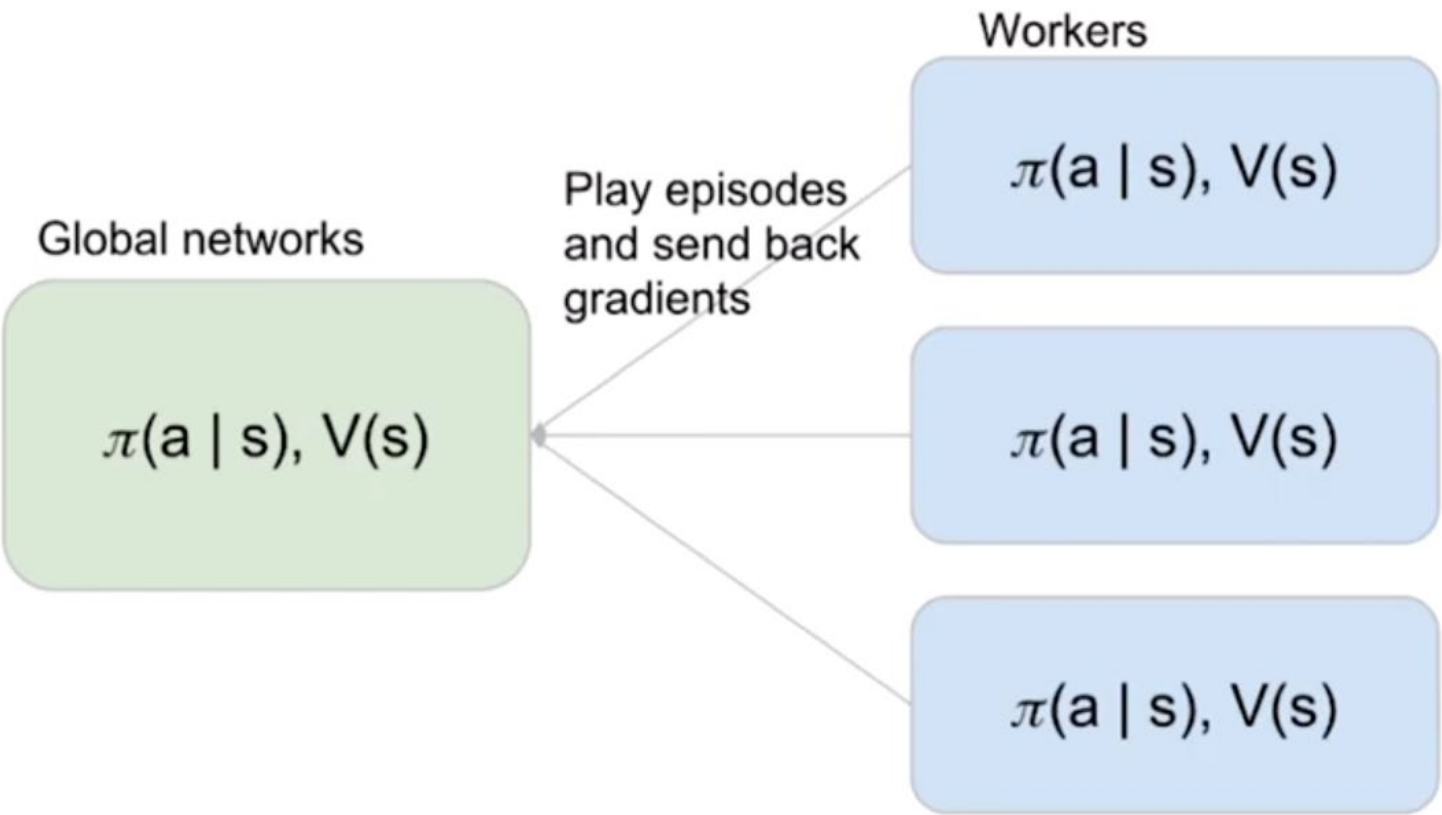
$$L = L_\pi + c_v L_v + c_{reg} L_{reg}$$

$$A^\theta(s_t, a_t) = G_t^n - V^{\pi_\theta}(s_t^n) = Q^{\pi_\theta}(s_t^n, a_t^n) - V^{\pi_\theta}(s_t^n) = r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)$$

$$L_v = (G_t^n - V^{\pi_\theta}(s_t^n))^2 = \left(r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)^2$$

$$L_\pi = \sum_{(s_t, a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta'}(s_t, a_t)\right)$$

A3C



Reference: https://youtu.be/iCV3vOl8IMk

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

A3C

Stability

- Each episode will progress randomly
- Each action is sampled probabilistically
- Occasionally, performance of agent can drop off due to bad update
  - Well, this can still happen with A3C so don't think you are immune

Reference: https://youtu.be/iCV3vOl8IMk

A3C

- DQN is also interested in stabilizing learning
- Techniques:
  - Freezing target network
  - Experience replay buffer
- Use experience replay to look at multiple examples per training step
- A3C simply achieves stability using a different method (parallel agents)
- Both solve the problem: how to make neural networks work as function approximators in classic RL algorithms?

Reference: https://youtu.be/iCV3vOl8IMk

A3C

- Remember: the theory part is not new, just need to create multiple parallel agents and asynchronously update/copy parameters
- 3 files:
  - main.py (master file; global policy and value networks)
    - Create and coordinate workers
  - worker.py (contains local policy and value networks)
    - Copy weights from global nets
    - Play episodes
    - Send gradients back to master
  - nets.py
    - Definition of policy and value networks

Reference: https://youtu.be/iCV3vOl8IMk

A3C

main.py

Instantiate global policy and value networks

Check # CPUs available, create threads and workers

Initialize global thread-safe counter, so every worker knows when to quit (when # of total steps reaches a max.)

Reference: https://youtu.be/iCV3vOl8IMk

A3C

worker.py

```
def run():
  in a loop:
    copy params from global nets to local nets
    run N steps of game (and store the data - s, a, r, s')
    using gradients wrt local net, update the global net
```

Conceptually, it's like:

1) $$g_{local} = \frac{\partial L(\theta_{local})}{\partial \theta_{local}}$$

2) $$\theta_{global} = \theta_{global} - \eta g_{local}$$

But in reality, we'll use RMSprop

Reference: https://youtu.be/iCV3vOl8IMk

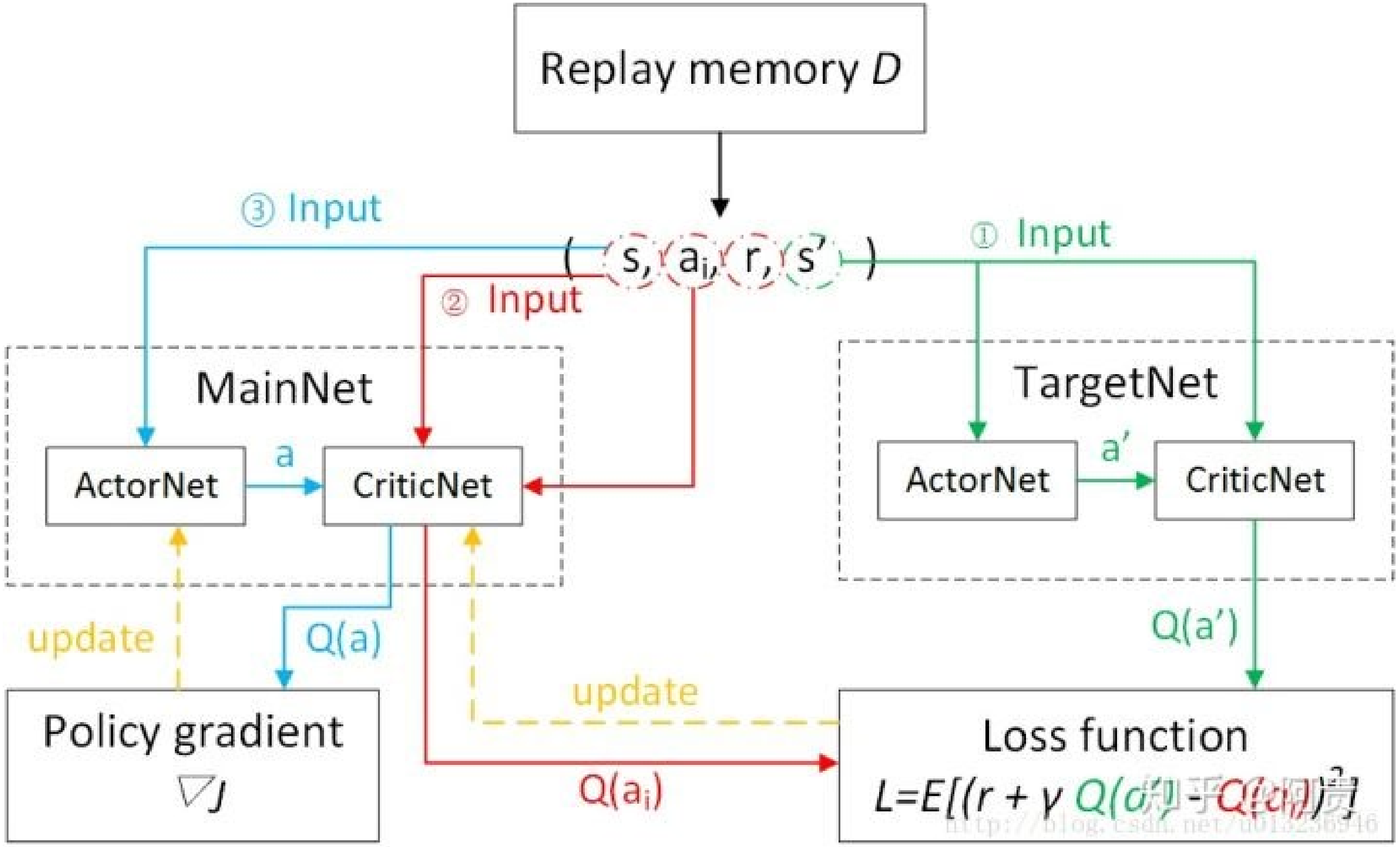
A3C

Multiprocessing in Python

- mp.Queue: a thread-safe FIFO queue for transporting training data
- mp.Process runs a piece of code in a child process
- PyTorch includes its own multiprocessing wrapper, same API

Reference: https://youtu.be/O5BlozCJBSE

Deep deterministic policy gradient (DDPG)



圖片來源: https://zhuanlan.zhihu.com/p/47873624

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Chen, Y. (2017). Mastering the game of go without human knowledge. Nature, 550(7676), 354.

Further study of RL

Chelsea Finn

cbfinn at cs dot stanford dot edu

I am an Assistant Professor in Computer Science and Electrical Engineering at Stanford University. My lab, IRIS, studies intelligence through robotic interaction at scale, and is affiliated with SAIL and the Statistical ML Group. I also spend time at Google as a part of the Google Brain team.

I am interested in the capability of robots and other agents to develop broadly intelligent behavior through learning and interaction.

Previously, I completed my Ph.D. in computer science at UC Berkeley and my B.S. in electrical engineering and computer science at MIT.

Prospective students and post-docs, please read this before contacting me.

CV / Bio / PhD Thesis / Google Scholar / GitHub / Twitter

Sergey Levine

Assistant Professor, UC Berkeley, EECS

Address:
Rm 8056, Berkeley Way West
2121 Berkeley Way
Berkeley, CA 94704

Email:
prospective students: please read this before contacting me.
svlevine AT eecs.berkeley.edu

Follow @svlevine

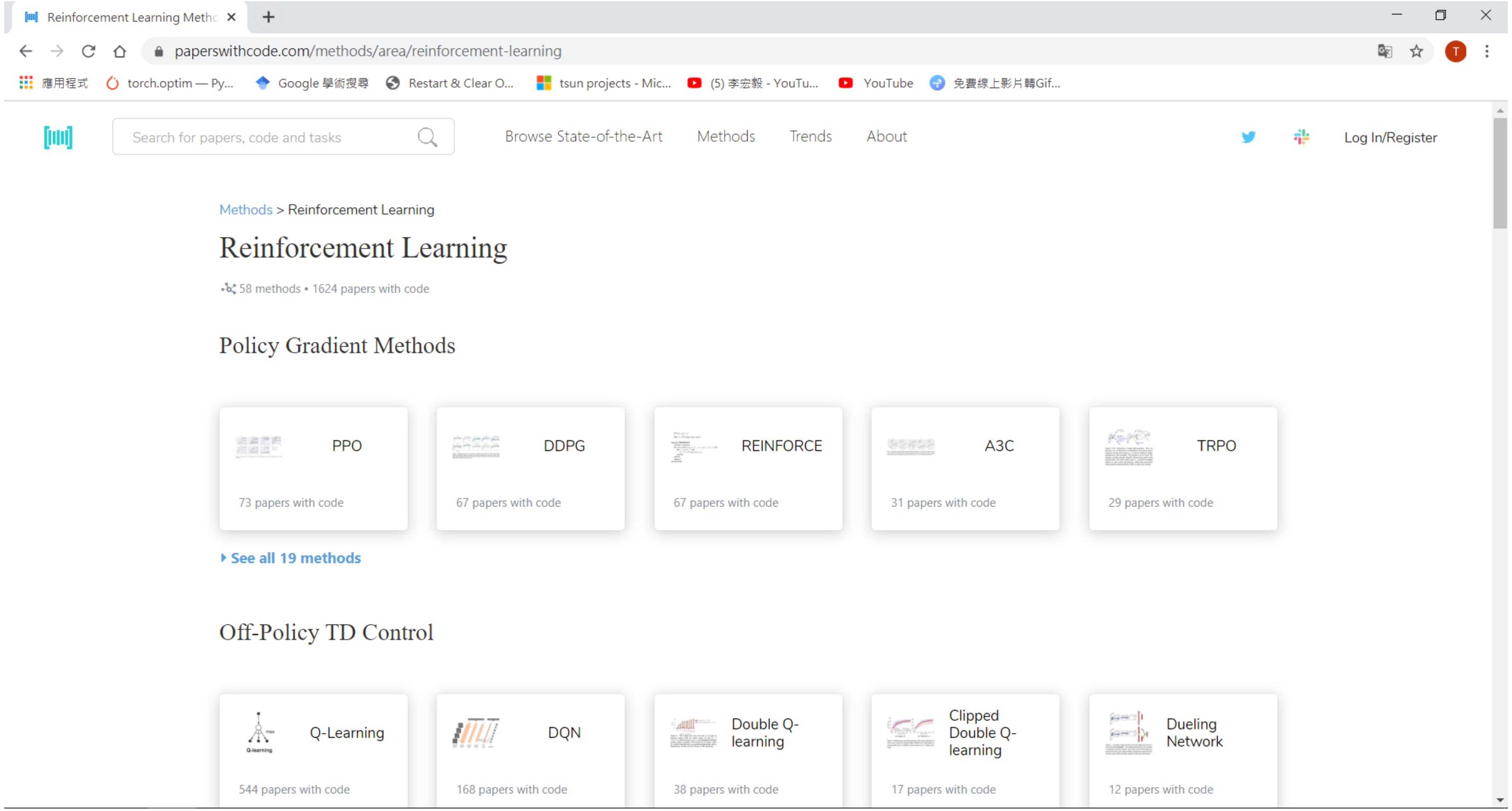
I am an Assistant Professor in the Department of Electrical Engineering and Computer Sciences at UC Berkeley. In my research, I focus on the intersection between control and machine learning, with the aim of developing algorithms and techniques that can endow machines with the ability to autonomously acquire the skills for executing complex tasks. In particular, I am interested in how learning can be used to acquire complex behavioral skills, in order to endow machines with greater autonomy and intelligence. To see a more formal biography, click here.

Research Group: Robotic Artificial Intelligence and Learning Lab

https://ai.stanford.edu/~cbfinn/

http://people.eecs.berkeley.edu/~svlevine/

Further study of RL



Paper with code: https://paperswithcode.com/methods/area/reinforcement-learning