

Lecture 10: Deep Reinforcement Learning

Chong Li

Outline

- Introduction
- Deep Learning to Value Functions
 - DQN
- Deep Learning to Policy Functions
 - Actor-critic methods: DDPG, A3C
 - Optimization methods: TRPO, GPS
- Deep Learning to Model Functions

*some materials are modified from David Silver's RL lecture notes

Outline

- Introduction
- Deep Learning to Value Functions
 - DQN
- Deep Learning to Policy Functions
 - Actor-critic methods: DDPG, A3C
 - Optimization methods: TRPO, GPS
- Deep Learning to Model Functions

Introduction to Deep RL

- The integration of RL and NN has a long history.
- For example, use NN to approximate value functions (1980s – 1990s)
- However, RL algorithms are unstable or even divergent when action value function is approximated with a nonlinear function, e.g. NN.
- What can we do ...

Winter is coming ...



Introduction to Deep RL

- Mnih *et al.* (2015) introduced DQN and opened the door to Deep RL.
- Achievement of DRL benefits from big data, powerful computing capability, mature software packages and architectures, and strong financial support.
- A single agent which can solve human-level task:

AI = Reinforcement Learning + Deep Learning = Deep RL

- Examples:
 - Games: Atari, Go, ...
 - User interaction: recommend, personalize, ...
 - Robotics: walk, swim, ...

Approaches to RL

- Value-based RL
 - Estimate the optimal **value function** $V^*(s)$ or $Q^*(s,a)$
 - *E.g. Q-learning, Sarsa, Monte Carlo, TD, ...*
- Policy-based RL
 - Search directly for the optimal **policy function** π^*
 - An objective function is needed
 - E.g. policy gradient decent
- Model-based RL
 - Build a model (**transition functions**) from samples
 - Plan using model, e.g. value iteration, policy iteration, ...

Approximate **functions**
by deep NN



Deep RL

Outline

- Introduction
- Deep Learning to Value Functions
 - DQN
- Deep Learning to Policy Functions
 - Actor-critic methods: DDPG, A3C
 - Optimization methods: TRPO, GPS
- Deep Learning to Model Functions

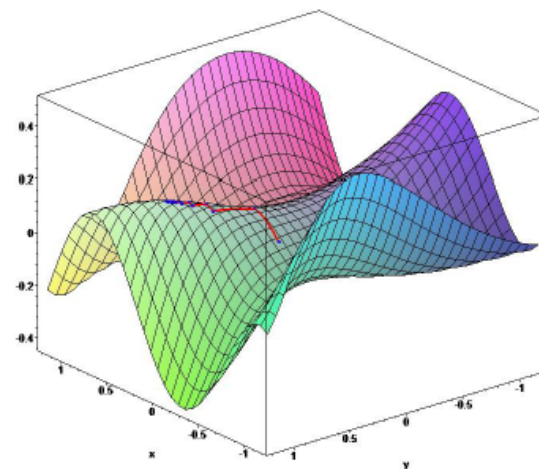
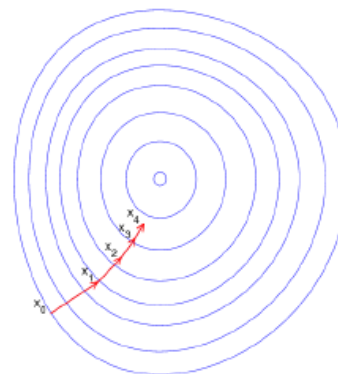
Gradient Descent (refresher)

- Let $J(\mathbf{w})$ be a differentiable function of parameter vector \mathbf{w}
- Define the *gradient* of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust \mathbf{w} in direction of -ve gradient

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$



Stochastic Gradient Descent (refresher)

- Goal: find parameter vector \mathbf{w} minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Deep Q-Network (DQN)

- Represent value function by deep **Q-network** with weights w

$$Q(s, a, w) \approx Q^\pi(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E} \left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w) \right)^2 \right]$$

- Leading to the following **Q-learning** gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E} \left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial \mathcal{L}(w)}{\partial w}$

Stability Issues

- Naïve Q-learning oscillates or diverges with neural nets

Why?

- Data is sequential. i.e. successive samples are correlated, non-iid
- Policy changes rapidly with slight changes to Q-values
- High correlation between Q-values and the target values
- Scale of rewards and Q-values is unknown
 - Naïve Q-learning gradients can be large, unstable when backpropagated

DQN

DQN provides a stable solution to deep value-based RL

- Use **experience replay**
Break correlations in data, bring us back to iid setting
Learn from all past policies
- Freeze **target Q-network**
Avoid oscillations
Break correlations between Q-network and target
- **Clip** rewards or **normalize** network adaptively to sensible range
Robust gradients

DQN - Algorithm

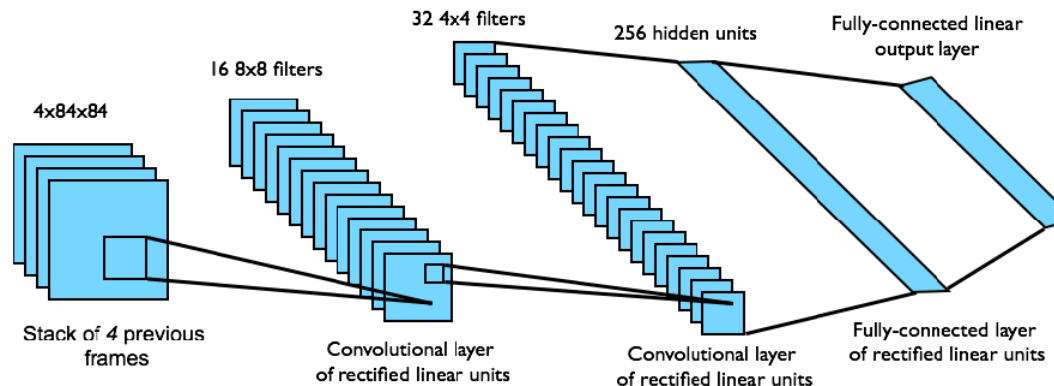
- Take action a_t according to ϵ -greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory \mathcal{D}
- Sample random mini-batch of transitions (s, a, r, s') from \mathcal{D}
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]$$

- Using variant of stochastic gradient descent

DQN in Atari Games

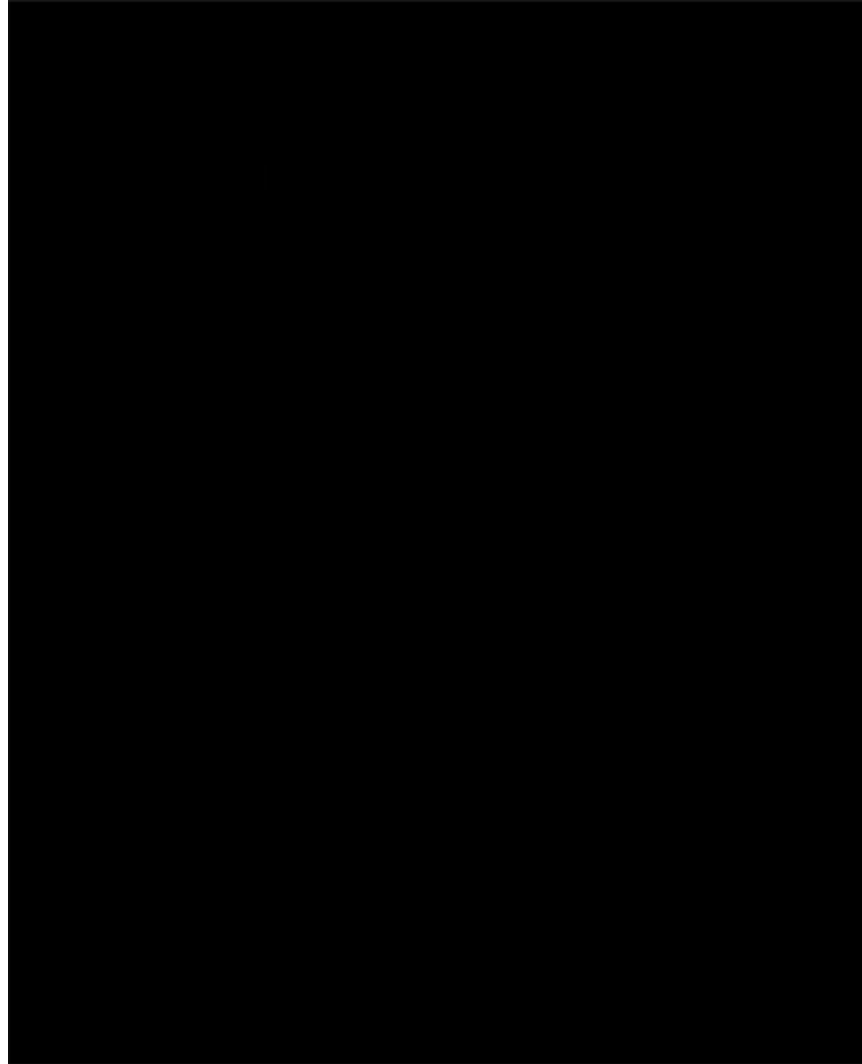
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network **architecture** and **hyperparameters** do not change across games

See paper “Human-level control through deep reinforcement learning” Nature, 2015

Demo- Atari Breakout



Outline

- Introduction
- Deep Learning to Value Functions
 - DQN
- Deep Learning to Policy Functions
 - Actor-critic methods: DDPG, A3C
 - Optimization methods: TRPO, GPS
- Deep Learning to Model Functions

Deterministic Policy Gradient (DPG)

- Policy gradient algorithms are widely used in reinforcement learning problems with continuous action spaces.
- Stochastic policy approximation:

$$\pi_{\theta}(a|s) = \mathbb{P}[a|s; \theta]$$

- Deterministic policy gradient:

$$a = \mu_{\theta}(s)$$

- Computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions, WHY?

Deterministic Policy Gradient (DPG)

- Stochastic policy gradient theorem (lecture 8)

*For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

- Deterministic policy gradient theorem

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s \sim \rho^\mu} \left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} \right] \end{aligned}$$

ρ^π is the discounted state distribution

Deterministic Policy Gradient (DPG) for Continuous Actions

- DPG is the limiting case, as policy variance tends to zero, of the stochastic policy gradient
- DPG can significantly outperform their stochastic counter-parts in high-dimensional action spaces
- DPG may NOT explore full state and action space, how to deal with it?

Deep Deterministic Policy Gradient (DDPG)

- Represent deterministic policy by deep network $a = \pi(s, u)$ with weights u
- Define objective function as total discounted reward

$$J(u) = \mathbb{E} [r_1 + \gamma r_2 + \gamma^2 r_3 + \dots]$$

- Optimise objective end-to-end by SGD

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial Q^\pi(s, a)}{\partial a} \frac{\partial \pi(s, u)}{\partial u} \right]$$

Update policy in the direction that most improves Q

Deep Deterministic Actor-Critic

- Use two networks: an **actor** and a **critic**
 - **Critic** estimates value of current policy by Q-learning

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E} \left[\left(r + \gamma Q(s', \pi(s'), w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]$$

- **Actor** updates policy in direction that improves Q

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial Q(s, a, w)}{\partial a} \frac{\partial \pi(s, u)}{\partial u} \right]$$

Deep Deterministic Policy Gradient (DDPG)

- Naive actor-critic **oscillates** or **diverges** with neural nets
DDAC provides a stable solution

- Use **experience replay** for both actor and critic
Use **target Q-network** to avoid oscillations

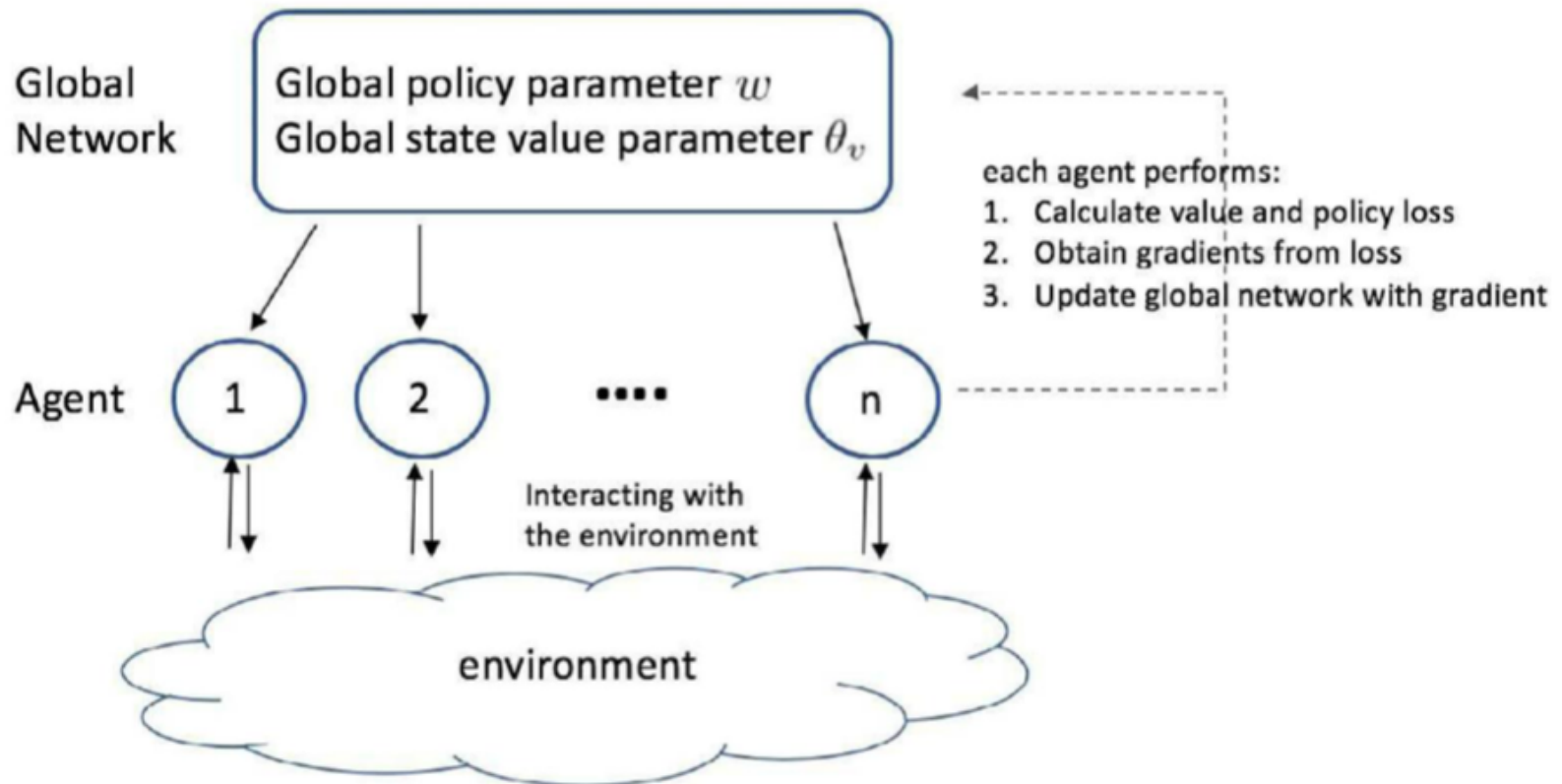
- $$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma Q(s', \pi(s'), w^-) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]$$
$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\frac{\partial Q(s, a, w)}{\partial a} \frac{\partial \pi(s, u)}{\partial u} \right]$$

Asynchronous Advantage Actor-Critic (A3C)

- Asynchronous:
 - A3C uses a global network and multiple agents/workers where each has its own set of network parameters. Each agent interacts with its own local environment and update the global parameter in an asynchronous manner
- Advantage:
 - In policy gradient, we use advantage function to improve the policy update.
- Actor-critic
 - An architecture that we have learnt

Asynchronous Advantage Actor-Critic (A3C)

- A3C architecture



Why A3C?

- The samples an agent gathers are highly correlated, leading to unstable algorithms if nonlinear function approximation (e.g. deep NN) applied.
- In DQN & DDPG, we use experience replay to overcome this issue.
- A3C runs several agents in parallel, each with its own copy of the environment, and use their samples for updates. Different agents will likely experience different states and transitions, thus avoiding the correlation.
- A3C needs much less memory, i.e. no need to store the samples for experience replay.

Outline

- Introduction
- Deep Learning to Value Functions
 - DQN
- Deep Learning to Policy Functions
 - Actor-critic methods: DDPG, A3C
 - Optimization methods: TRPO, GPS
- Deep Learning to Model Functions

Trust Region Policy Optimization (TRPO)

- Schulman *et al.* (2015) introduced an iterative procedure to monotonically improve policies theoretically.

$$\underset{\theta}{\text{maximize}} [L_{\theta_{\text{old}}}(\theta) - CD_{\text{KL}}^{\text{max}}(\theta_{\text{old}}, \theta)]$$

where $D_{\text{KL}}^{\text{max}}(\pi, \tilde{\pi}) = \max_s D_{\text{KL}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s))$

$$D_{\text{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

- A practical algorithm (TRPO) is proposed by making several approximations
 - Introducing a trust region constraint, defined by the KL divergence between the new policy and the old policy, so that at every point in the state space, the KL divergence is bounded. Then approximate the trust region by the average KL divergence constraint

$$\underset{\theta}{\text{maximize}} L_{\theta_{\text{old}}}(\theta)$$

$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta$$

where $\overline{D}_{\text{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s \sim \rho} [D_{\text{KL}}(\pi_{\theta_1}(\cdot|s) \parallel \pi_{\theta_2}(\cdot|s))]$.

- Replacing the expectations and Q value in the optimization problem by sample estimates

Other State-of-the-Arts

- TRPO + GAE
 - Schulman et al [2016] introduced generalized advantage estimation (GAE), proposing more advanced variance reduction baselines for policy gradient methods.
- Guided policy search (GPS) [2016]
 - Searching directly for a policy represented by a neural network with a large number of parameters can be difficult and can suffer from severe local minima.
 - GPS generates suitable guiding samples from differential dynamic programming, and learns from them by using supervised learning in combination with importance sampling.

Outline

- Introduction
- Deep Learning to Value Functions
 - DQN
- Deep Learning to Policy Functions
 - Actor-critic methods: DDPG, A3C
 - Optimization methods: TRPO, GPS
- Deep Learning to Model Functions

Model-based DRL

- Model-based RL: Learn a transition model of the environment

$$p(r, s' \mid s, a)$$

- Model-based DRL:
 - Use **deep NN** to approximate the state and reward transition probability
 - Define objective function measuring how good the model is
 - Optimize objective by SGD
- Not commonly used due to many challenges:
 - Errors in the transition model compound over the trajectory
 - By the end of a long trajectory, rewards can be totally wrong
 - Model-based RL has failed in Atari tests.

AI learns how to walk



Congratulations! You have successfully stepped into the AI world. I wish you all an enjoyable and wonderful journey in the future.

Thank You !