Lecture 10: Deep Reinforcement Learning

Chonggang Wang

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

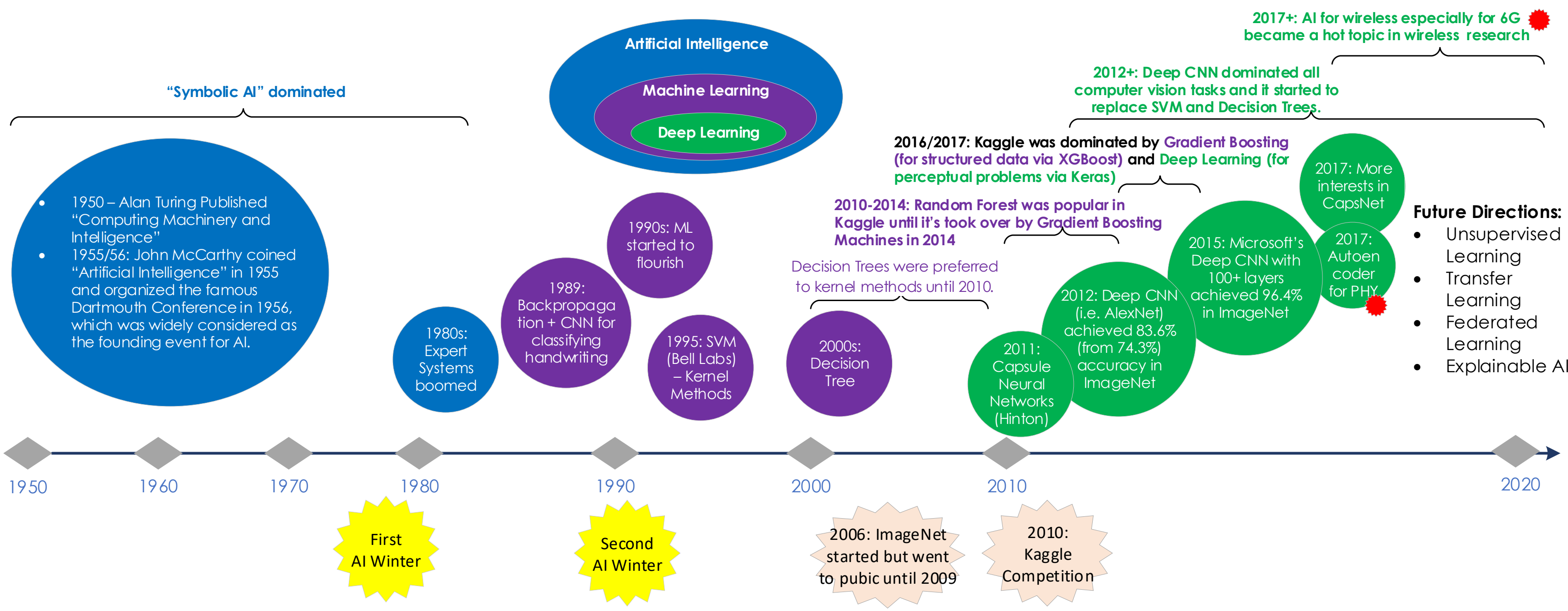
- The integration of RL and NN has a long history.
- For example, use NN to approximate value functions (1980s – 1990s)
  - e.g., TD-Gammon (1995)

- 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 …

A Brief History of AI



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 ascent
- 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

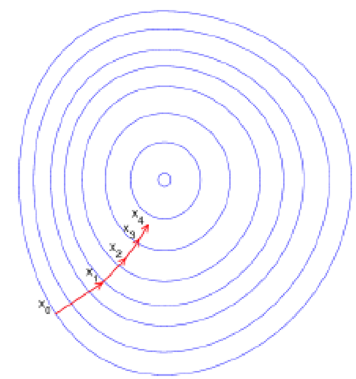
Gradient Descent (refresher)

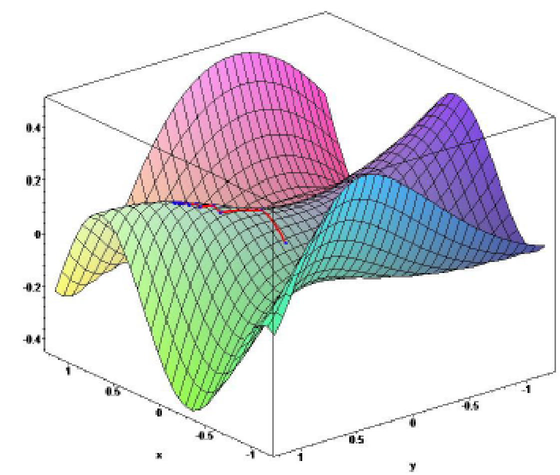
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- 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 $v_\pi(s)$

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

- Gradient descent finds a local minimum

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

- 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 $\frac{\partial L(w)}{\partial w}$

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

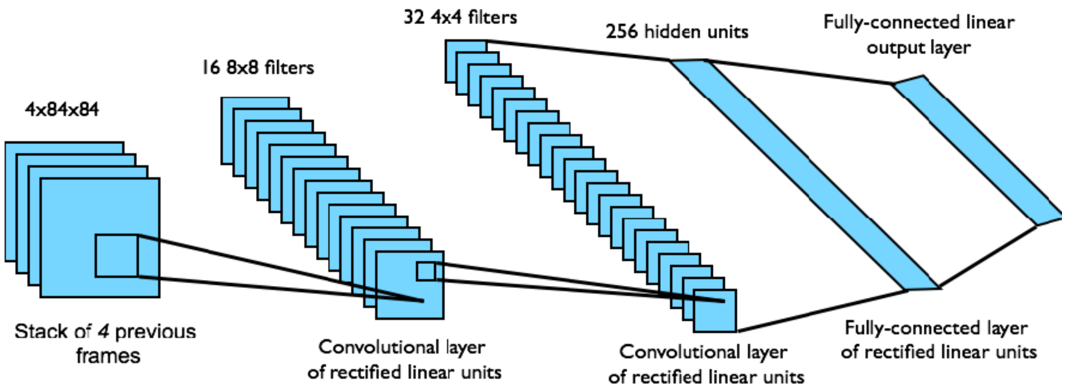
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and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



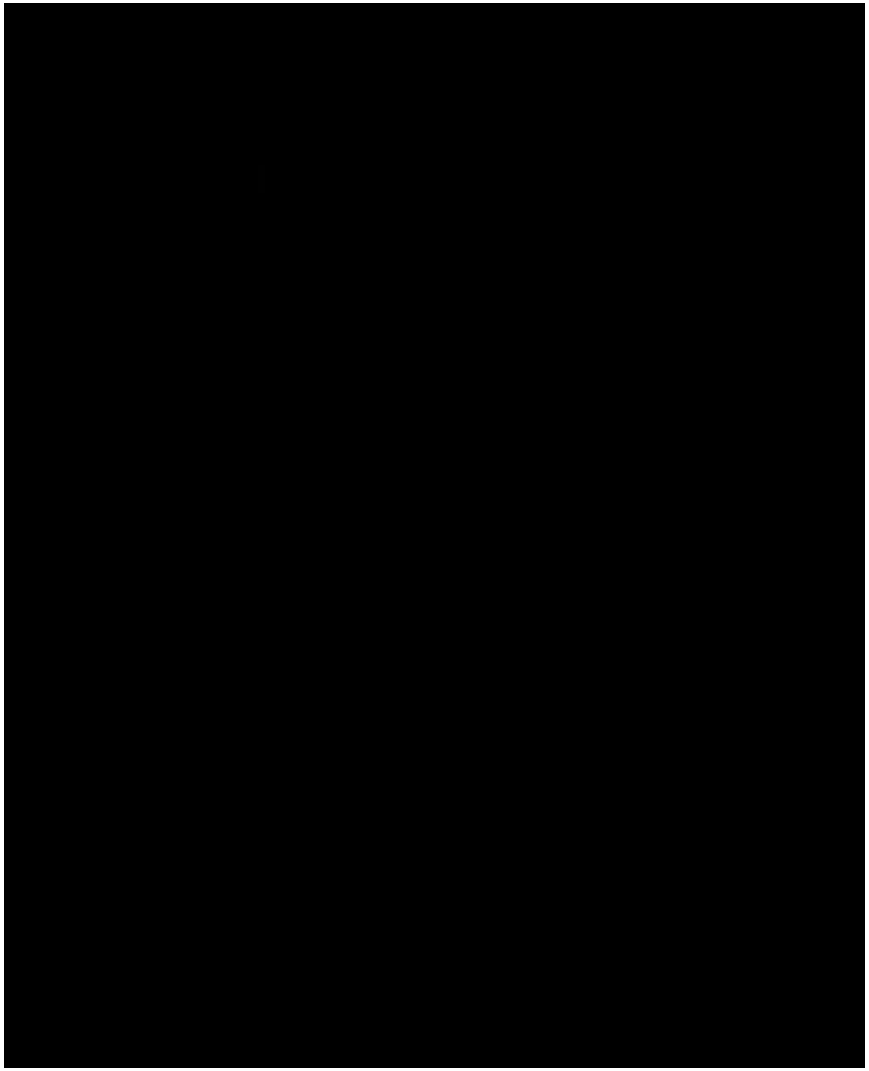
Some Hyperparameters:

- replay memory size = 1M
- minibatch size = 32
- target network update frequency = 10000

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:

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

- 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?
  - *"In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. "*
  - *"As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions" [DPG]*

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

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

- Deterministic policy gradient theorem

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

ρ^μ *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?
  - Off-Policy Learning Algorithm: Choose actions according to a stochastic behavior policy to provide adequate exploration, but to learn about a deterministic target policy

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

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

- 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 (DDAC)

- 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

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

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

"Soft" Target Updates in DDPG (τ<<1):

$$w^{-} = \tau * w + (1-\tau) * w^{-}$$
$$\pi(s') = \tau * \pi(s) + (1-\tau) * \pi(s')$$

Deep Deterministic Policy Gradient (DDPG)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

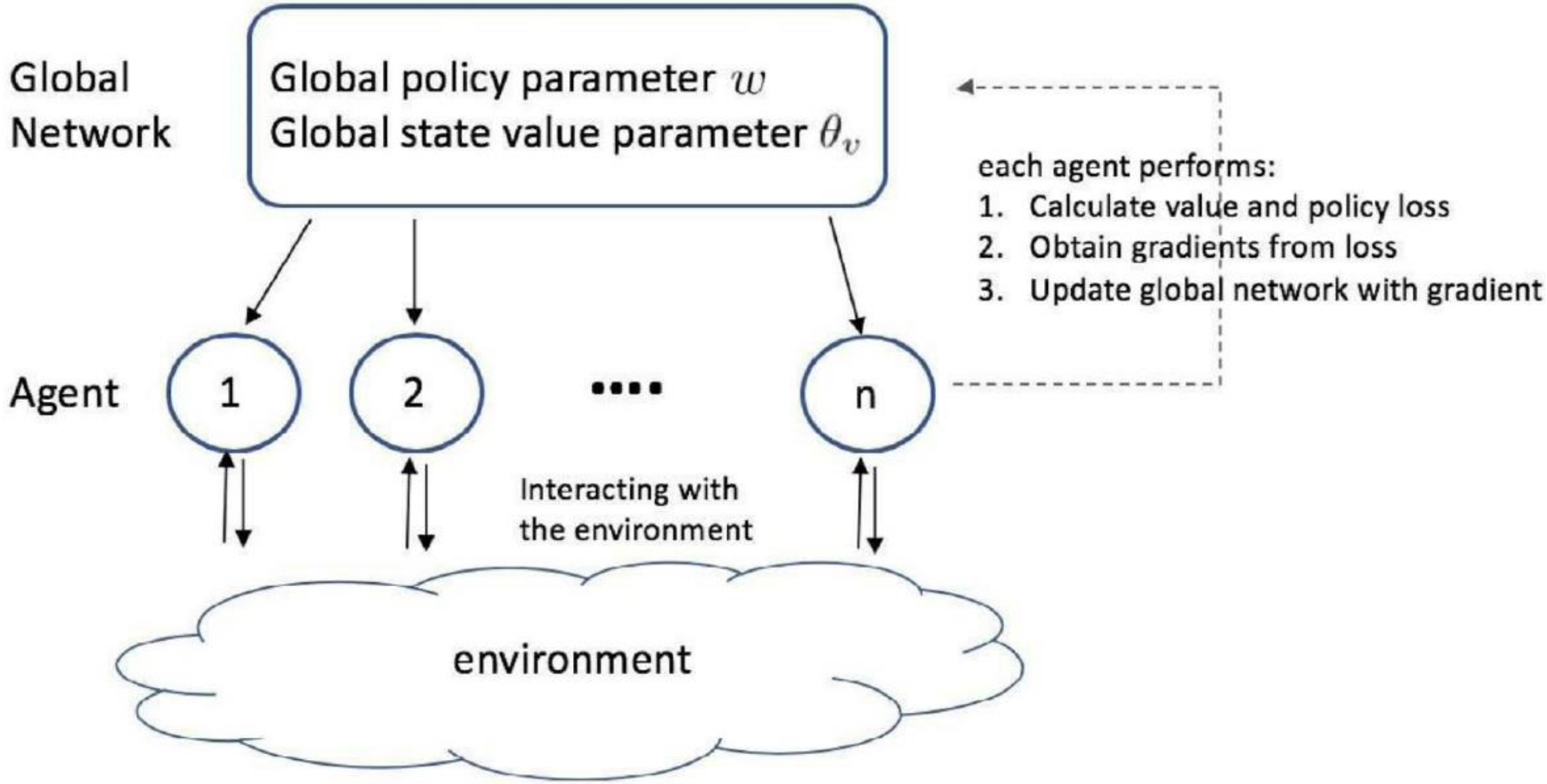
Source: [DDPG]

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 in Lecture 8

Asynchronous Advantage Actor-Critic (A3C)

- A3C architecture



Asynchronous Advantage Actor-Critic (A3C)

Algorithm S2 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

01 → *// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$*
02 → *// Assume thread-specific parameter vectors θ' and θ'_v*
03 → Initialize thread step counter $t \leftarrow 1$
04 → **repeat**
05 → Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
06 → Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
07 → $t_{start} = t$
08 → Get state s_t
09 → **repeat**
10 → Perform a_t according to policy $\pi(a_t|s_t;\theta')$
11 → Receive reward r_t and new state s_{t+1}
12 → $t \leftarrow t + 1$
13 → $T \leftarrow T + 1$
14 → **until** terminal s_t **or** $t - t_{start} == t_{max}$
15 → $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
for $i \in \{t-1, \ldots, t_{start}\}$ **do**
16 → $R \leftarrow r_i + \gamma R$
17 → Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
18 → Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
end for
19 → Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Source: [A3C] Supplementary Material

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, experience replay is used 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.
- Each actor-learner under A3C can use different exploration policies, which can maximize the diversity and reduce correlations compared to a single agent case
- A3C needs much less memory (i.e., no need to store the samples for experience replay).
- A3C runs on multi-core CPU threads on a single machine (i.e., one CPU thread for one actor-learner)

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.

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

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

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

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

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

Other State-of-the-Arts

- TRPO + GAE [2016] ("High-Dimensional Continuous Control using Generalized Advantage Estimation")
    - Schulman et al [2016] introduced generalized advantage estimation (GAE), proposing more advanced variance reduction baselines for policy gradient methods.
- GPS [2013] ("Guided Policy Search")
    - 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.

AlphaGo and AlphaGo Zero

- AlphaGo [Nature 2016]
  - Rollout Policy Network, SL Policy Network, RL of Policy Network, RL of Value Network
  - Modified MCTS: 1) Value Network + Rollout; 2) Modified UCB; 3)Threshold-based Expansion
    - Tree Search: $UCB(s,a) = Q(s,a) + U(s,a\ )$
      - $Q(s,a) = \sum_{s'|s,a \to s'}, V(s')\ /N(s,a)$
      - $U(s,a) \propto P(s,a)/[1 + N(s,a)]$
    - Rollout Policy: A linear softmax policy learned from human knowledge
    - Play Policy: $a_t = \underset{a}{\operatorname{argmax}}(N(s,a))$

N(s, a): Visit Account
P(s, a): Prior Probability
(from RL of Policy Network)
W(s, a): Total Action Value
Q(s, a): Mean Action Value
(=W(s, a)/N(s, a))

- AlphaGo Zero [Nature 2017]
  - Self-Play DRL w/o Human Knowledge, An Integrated Policy & Value Network
  - Modified MCTS: 1) Value Network without Rollout; 2) Modified UCB; 3) Always Leaf Node Expansion
    - Tree Search: $UCB(s,a) = Q(s,a) + U(s,a\ )$
      - $Q(s,a) = \sum_{s'|s,a \to s'} V(s')\ /N(s,a)$
      - $U(s,a) \propto P(s,a)/[1 + N(s,a)]$
    - Play Policy: $\pi(a|s) = N(s,a)^{1/\tau} / \sum_b (N(s,b)^{1/\tau})$

References

- Book Chapter 9.7, “Reinforcement Learning: An Introduction” (2nd Edition), 2018
- [DQN] “Human-Level Control through Deep Reinforcement Learning”, Nature 2015
- [DPG] “Deterministic Policy Gradient Algorithms”, ICML 2014
- [DDPG] “Continuous Control with Deep Reinforcement Learning”, ICLR 2016
- [A3C] “Asynchronous Methods for Deep Reinforcement Learning”, ICML 2016
- [TRPO] “Trust Region Policy Optimization”, ICML 2015
- [GPS] “Guided Policy Search”, ICML 2013
- [AlphaGo] “Mastering the Game of Go with Deep Neural Networks and Tree Search”, Nature 2016
- [AlphaGo Zero] “Mastering the Game of Go without Human Knowledge”, Nature 2017