Reinforcement Learning and its Applications

Sunghee Yun

Mobile Shopping Team
Amazon.com, Inc.

About the speaker

- Sunghee Yun
  - B.S., Electrical Engineering @ Seoul National University
  - M.S. & Ph.D., Electrical Engineering @ Stanford University
  - Samsung Electronics
    - CAE Team @ Semiconductor R&D Center of Samsung Electronics
    - Design Technology Team @ DRAM Development Lab. of Samsung Electronics
    - Memory Sales & Marketing Team @ Memory Business Unit of Samsung Electronics
    - Software R&D Center @ Samsung Electronics
  - Senior Applied Scientist @ Amazon
- Specialties
  - convex optimization
  - decentralized machine learning
  - deep reinforcement learning
  - recommendation systems

Supervised learning

- Data: $(x^{(i)}, y^{(i)}) \in \mathbf{R}^m \times \mathbf{R}^l$ $(i = 1, \ldots, N)$
- Goal: learn a function to predict y from x with parameters $\theta \in \mathbf{R}^n$

$$f(x; \theta) \sim y$$

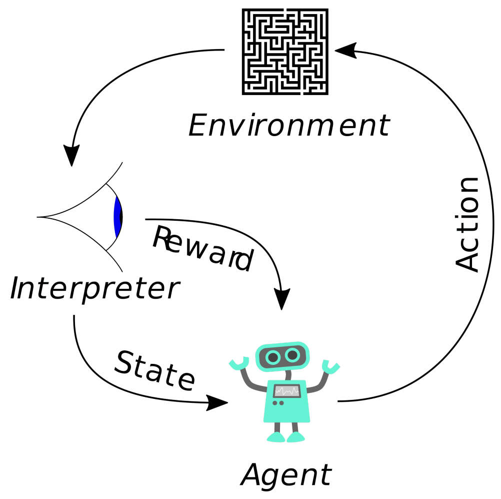
  where $f : \mathbf{R}^m \times \mathbf{R}^n \to \mathbf{R}^l$
- Applications
  - classification
  - regression
  - object detection
  - semantic segmentation

Unsupervised learning

- Data: $x^{(i)} \in \mathbf{R}^n$ $(i = 1, \ldots, N)$
- Goal: learn underlying hidden structure of x
- Applications
  - clustering,
  - dimensionality reduction (matrix factorization)
  - featuring learning
  - density estimation
  - autoencoder

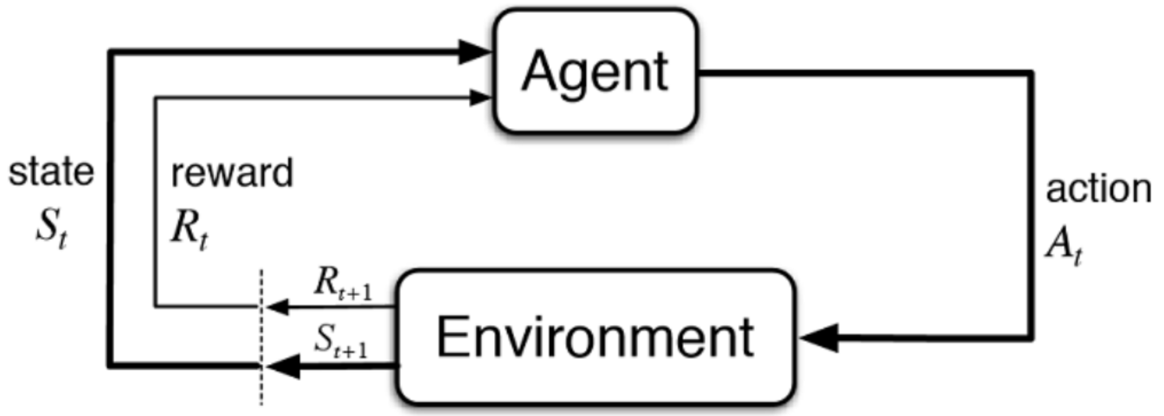
Reinforcement learning

- Agent actively interacts with environment to learn
  - unlike passive ways of learning, $e.g.$, supervised and unsupervised learnings
- Agent decides which actions to take based on history of actions and rewards
- Assumes that the environment reacts with uncertainty $\rightarrow$ stochastic formulation



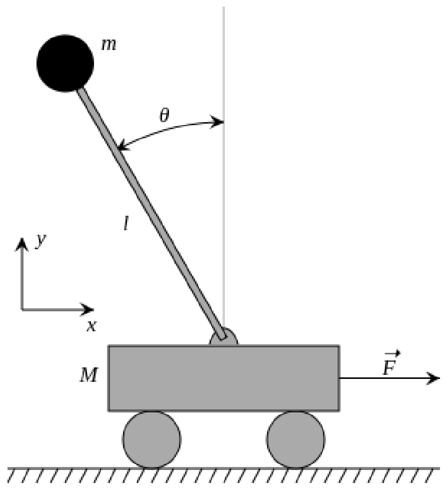
Reinforcement learning

- explicitly considers whole problem of a goal-directed agent
- in contrast to many other machine learning algorithms that consider subproblems without addressing how they may fit into a larger picture



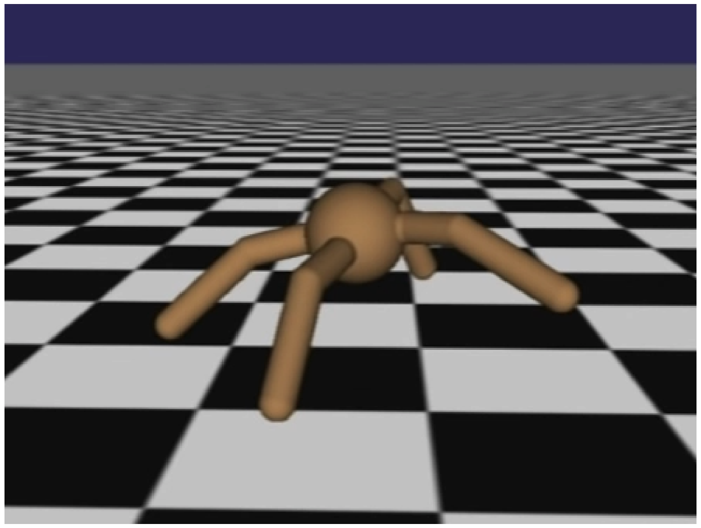
Reinforcement learning example: cartpole problem

- Cartpole: pendulum with center of gravity above pivot point (a.k.a. inverted pendulum)
- Objective: keep cartpole balanced by applying appropriate forces to pivot point
- State: position, (horizontal) velocity, angle, angular speed
- Action: horizontal force applied on cart
- Reward: 1 if the pole is upright @ each time step, 0 otherwise



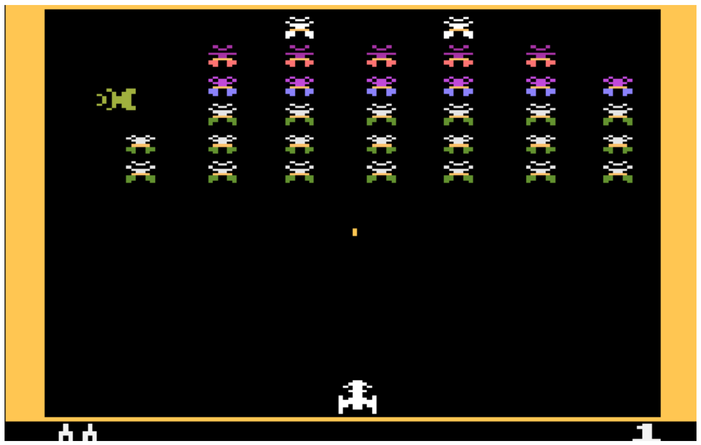
Reinforcement learning example: robot locomotion problem

- Objective: make the robot move forward
- State: angle and position of joints
- Action: torques applied to joints
- Reward: 1 if it's upright and moves forward @ each time step, 0 otherwise
- Sim-to-Real: Learning Agile Locomotion For Quadruped Robots



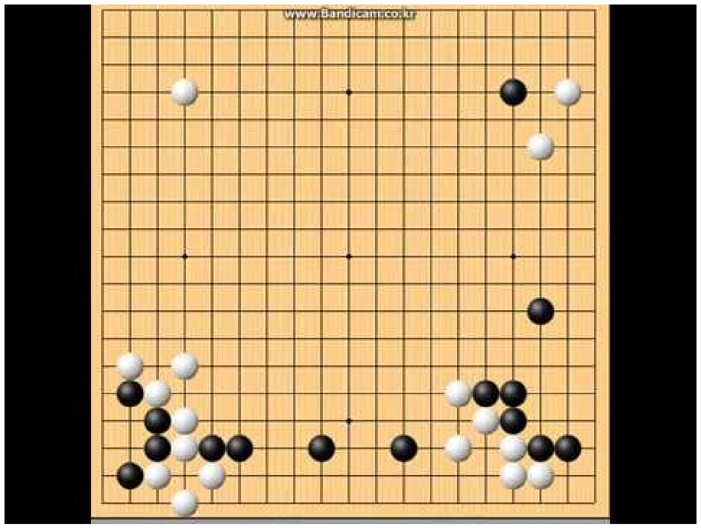
Reinforcement learning example: Atari games

- Objective: maximize score upon completion
- State: raw pixel inputs
- Action: game controls ($e.g.$, left, right, up, down)
- Reward: score increase or decrease @ each time step
- Google DeepMind's Deep Q-learning playing Atari Breakout



Reinforcement learning example: Go

- Objective: surround more territory (than the opponent)
- State: position of all pieces
- Action: where to put th next piece
- Reward: 1 if win at the end of the game, 0 otherwise



Genetic algorithm: learning to swing

- simulation example: Learn to swing by genetic algorithm

Reinforcement learning formulation

- Assume Markov decision process defined by $(\mathcal{S}, \mathcal{A}, R, P)$
  - $\mathcal{S}$: (finite) set of all states
  - $\mathcal{A}$: (finite) set of all actions
  - P: transition probability (given state and action)
  - R: distribution of reward (given state and action)
- Environment and agent are modeled as stochastic finite state machines
- Markov property: conditional probability distribution of future states depends only upon present state

$$\begin{aligned}&\mathbf{Prob}\{s_{t+1} = s, r_{t+1} = r \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0\}\\ &= \mathbf{Prob}\{s_{t+1} = s, r_{t+1} = r \mid s_t, a_t\}\end{aligned}$$

Policy and discount factor

- $\pi : \mathcal{S} \times \mathcal{A} \to \mathbf{R}_+$ is called policy
  - mapping from each state and action to the probability
  - samples trajectories can be drawn from π; $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$
- objective of reinforcement learning is to maximize cumulative reward

$$\sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

- $\gamma \in [0, 1]$: discount factor
  - $\gamma = 1$ means we want to maximize sum of all rewards
  - $\gamma = 0$ means we want to maximize immediate reward (myopic strategy)
  - $0 < \gamma < 1$: trade-off between short-term and long-term rewards

Value functions

- Value function (or state value function): expected cumulative reward from s when following π

$$\begin{aligned} V^{\pi}(s) &= \mathop{\mathbf{E}}_{\pi}\left(\sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau} \middle| s_t = s\right) \\ &= \mathop{\mathbf{E}}_{\pi}\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \middle| s_t = s\right) \end{aligned}$$

- Q-value function (or action value function): expected cumulative reward from taking action a in state s

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{\mathbf{E}}_{\pi}\left(\sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau} \middle| s_t = s, a_t = a\right) \\ &= \mathop{\mathbf{E}}_{\pi}\left(r_t + \gamma r_{t+1} + \cdots \middle| s_t = s, a_t = a\right) \end{aligned}$$

Optimal policy

- want to maximize *expected* sum of rewards

$$\begin{array}{ll} \text{maximize} & \mathbf{E}_\pi \left(\sum_{t=0}^{\infty} \gamma^t r_t \middle|\, s_0 \right) \\ \text{subject to} & s_0 \sim p(s_0) \\ & a_t \sim \pi(\cdot|s_t) \\ & s_{t+1} \sim p(\cdot|s_t, a_t) \end{array}$$

- deals with all uncertainty (or randomness), $e.g.$, initial state, transition probability, reward distribution, $etc.$
- The optimal policy is the optimal solution, $i.e.$,

$$\pi^* = \operatorname*{argmax}_{\pi} \mathop{\mathbf{E}}_{\pi} \left(\sum_{t=0}^{\infty} \gamma^t r_t \middle|\, s_0 \right)$$

Optimal value functions

- optimal value function

$$V^*(s) = \max_{\pi} V^{\pi}(s) = V^{\pi^*}(s)$$

- optimal Q-function

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

Bellman optimality equation

- Bellman optimality equation for $V^*(s)$

$$
\begin{aligned}
V^*(s) &= \max_{a\in\mathcal{A}} Q^*(s,a) \\
&= \max_{a\in\mathcal{A}} Q^{\pi^*}(s,a) \\
&= \max_{a\in\mathcal{A}} \mathop{\mathbf{E}}_{\pi^*}\left(\sum_{\tau=0}^{\infty}\gamma^\tau r_{t+\tau}\middle|\, s_t=s, a_t=a\right) \\
&= \max_{a\in\mathcal{A}} \mathop{\mathbf{E}}_{\pi^*}\left(r_t+\gamma\sum_{\tau=0}^{\infty}\gamma^\tau r_{t+\tau+1}\middle|\, s_t=s, a_t=a\right) \\
&= \max_{a\in\mathcal{A}} \mathbf{E}\left(r_t+\gamma V^*(s_{t+1})\middle|\, s_t=s, a_t=a\right)
\end{aligned}
$$

Bellman optimality equation

- Bellman optimality equation for $Q^*(s, a)$

$$\begin{aligned} Q^*(s,a) &= \mathbf{E}\left(r_t + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a\right) \\ &= \mathbf{E}\left(r_t + \gamma \max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a\right) \end{aligned}$$

Solution candidate: dynamic programming

- dynamic programming methods
  - policy evaluation
  - policy improvement
  - policy iteration
  - value iteration

Value iteration algorithm

- use Bellman equation as itertive update

$$Q^{k+1}(s,a) = \mathbf{E}\left(r + \gamma \max_{a' \in \mathcal{A}} Q^k(s', a') \,\middle|\, s, a \right)$$

- $Q^{k+1}(s,a)$ converges to $Q^*(s,a)$ as $k \to \infty$
- problems
  - must compute $Q^k(s,a)$ for every (s,a) pair
  - intractable when state space is huge ($e.g.$, game state pixels)

Q-learning

- use function approximator to estimate Q-function

- find θ such that
$$Q(s, a; \theta) \approx Q^*(s, a)$$

- called *deep Q-learning* when using deep neural network for function approximator

Q-learning

- find Q-function that satisfies Bellman optimality equation:

$$Q^*(s,a) = \mathop{\mathbf{E}}_{s'}\left(r + \gamma \max_{a'} Q^*(s',a')\,\middle|\, s,a\right)$$

- loss function

$$L^k(\theta^k) = \mathop{\mathbf{E}}_{s,a}(y^k - Q(s,a;\theta^k))^2$$

  where

$$y^k = \mathop{\mathbf{E}}_{s'}\left(r + \gamma \max_{a'} Q(s',a';\theta^{k-1})\,\middle|\, s,a\right)$$

Q-learning

- gradient with respect to θ^k

$$\nabla_{\theta_k} L^k(\theta^k) = -2 \underset{a}{\mathbf{E}}(y^k - Q(s,a;\theta^k))\nabla_{\theta^k} Q(s,a;\theta^k)$$

- gradient update

$$\theta_k^+ = \theta_k - \alpha_k \nabla_{\theta_k} L^k(\theta^k)$$

Training Q-network: experience play

- batches of consecutive samples are problematic
  - samples are correlated $\rightarrow$ inefficient learning
  - current Q-network parameters determine next training samples $\rightarrow$ can lead to bad feedback loops
- solution: experience play
  - continually update *reply memory* table of transitions (s_t, a_t, r_t, s_{t+1}) as game episodes are played
  - train Q-network on random minibatches of transitions from replay memory

Problems with Q-learning

- Q-function can be very complicated
  - for example, robot grasping an object has very high-dimensional state $\rightarrow$ hard to learn exact value of every (s, a) pair
- However,
  - policy can be much simpler, $e.g.$, close your hand!
  - can we learn policy directly, $e.g.$, finding best policy from collection of candidate policies?

Policy gradients

- define class of parameterized policies

$$\Pi = \{\pi_\theta \mid \theta \in \mathbf{R}^n\}$$

- define merit function

$$f(\theta) = \mathbf{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi_\theta\right)$$

- solve maximization problem

$$\begin{array}{ll} \text{maximize} & f(\theta) \\ \text{subject to} & \pi_\theta \in \Pi \end{array}$$

- solution method
  - gradient ascent on θ

$$\theta^+ = \theta + \alpha_k \nabla f(\theta)$$

REINFORCE algorithm

- merit function can be written as

$$f(\theta) = \mathop{\mathbf{E}}_{\tau \sim p(\tau;\theta)} r(\tau) = \int_\tau r(\tau) p(\tau;\theta) d\tau$$

  – $r(\tau)$ is reward of trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$

- with this expression, it is intractable!

$$\nabla f(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) d\tau$$

REINFORCE algorithm

- however, we can use trick

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

- then, gradient of merit function can be written as

$$\begin{aligned}\nabla_\theta f(\theta) &= \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)d\tau \\ &= \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)d\tau \\ &= \mathop{\mathbf{E}}_{\tau\sim p(\tau;\theta)} \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right)\end{aligned}$$

- can we evaluate this without knowing transition probabilities?

REINFORCE algorithm

- express $p(\tau;\theta)$ using transition probabilities

$$p(\tau;\theta) = \prod_{t=0}^{\infty} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

  - thus, $\log p(\tau;\theta) = \sum_{t=0}^{\infty} \log p(s_{t+1}|s_t, a_t) + \sum_{t=0}^{\infty} \log \pi_\theta(a_t|s_t)$
  - and $\nabla_\theta \log p(\tau;\theta) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)$

- therefore, when sampling trajectory, τ, can estimate the gradient as

$$\nabla_\theta f(\theta) \approx \sum_{t=0}^{\infty} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Summary of algorithms

- Policy gradients
  - very general, but suffers from high variance
  - requires lots of samples
  - converges to local minima of $f(\theta)$
  - challenge: sample-efficiency
- Q-learning
  - does not always work
  - when it works, usually more sample-efficient
  - challenge: exploration
  - no guarantee for convergence

Why (Deep) Reinforcement Learning?

- Koray Kavukcuoglu (director of research at Deepmind) says
  - If one of the goals that we work for here is AI then it is at the core of that. Reinforcement Learning is a very general framework for learning sequential decision making tasks. And Deep Learning, on the other hand, is of course the best set of algorithms we have to learn representations. And combinations of these two different models is the best answer so far we have in terms of learning very good state representations of very challenging tasks that are not just for solving toy domains but actually to solve challenging real world problems.

Applications of reinforcement learning

- resource management
- traffic light control
- robotics
- personalized recommendations
- web system configuration
- chemistry
- bidding and advertising
- games ($e.g.$, Atari games, AlphaGo)
- finance
- health care

Resource management in computer clusters

- goal: designing algorithms to allocate limited resources to different tasks
  - very challenging
  - requires human-generated heuristics
- reinforcement learning formulation
  - objective: minimize average job slowdown
  - states: current resources allocation and resources profile of jobs
  - actions: allocate and schedule computer resources to waiting jobs
  - reward: sum of $-1/(\text{duration of job})$
  - optimal policy: probability distribution of actions

Traffic light control

- goal: design a traffic light controller to solve the congestion problem
- reinforcement learning formulation
  - agents in intersection traffic network to control traffic signaling
  - states: traffic flow of each lane
  - actions: traffic light control
  - reward: reduction in delay

Personalized recommendations

- news recommendations faces several challenges
  - rapid changing dynamic of news
  - users get bored easily
- reinforcement learning formulation
  - objective: click-through-rate (CTR), customer engagement
  - states: user features, context features
  - actions: list of news to recommend
  - reward: click on news

Pre-requisites for using RL to your problems

- domain knowledge
- simulated environment
- Markov decision process (MDP)
- algorithms

Conclusion

- Reinforcement Learning (RL) actively interacts with environment to learn
  - closer to the way humans learn
- When combined with Deep Learning, it can do amazing things
  - play Atari games, beat Go world champion
- Efficient algorithms developed
  - Q-learning, Policy gradient, Actor-critic method
- Potential applications in many areas
  - games, robotics, resource management, bidding and advertising, finance, *etc.*

References

- R.S. Sutton, A.G. Barto, "An Introduction to Reinforcement Learning", 2018
- H. Mao, M. Alizadeh, I. Menache, S. Kandula, "Resource Management with Deep Reinforcement Learning", 2016
- I. Arel, C. Liu, T. Urbanik, A.G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control", 2010
- V. Mnih, et al. "Playing Atari with Deep Reinforcement Learning", 2013
- G. Zheng, et al. "DRN: A Deep Reinforcement Learning Framework for News Recommendation", 2018