

Machine Learning in Practice

#1-1: Course Overview

Sang-Hyun Yoon

Summer 2019

Without Machine Learning

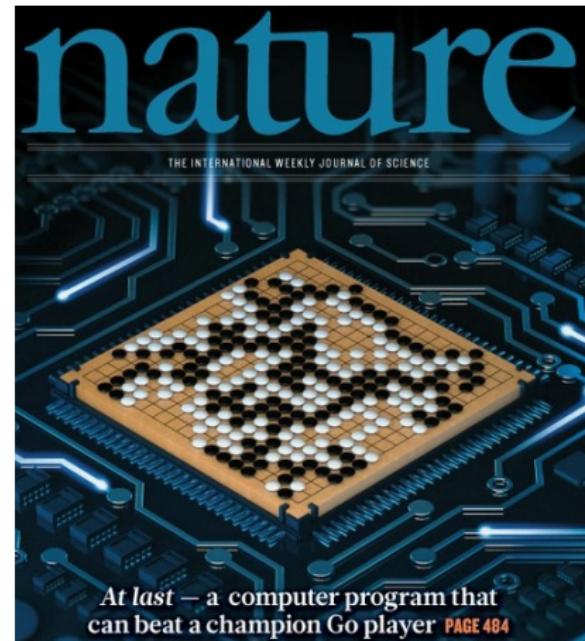
vs.

With Machine Learning

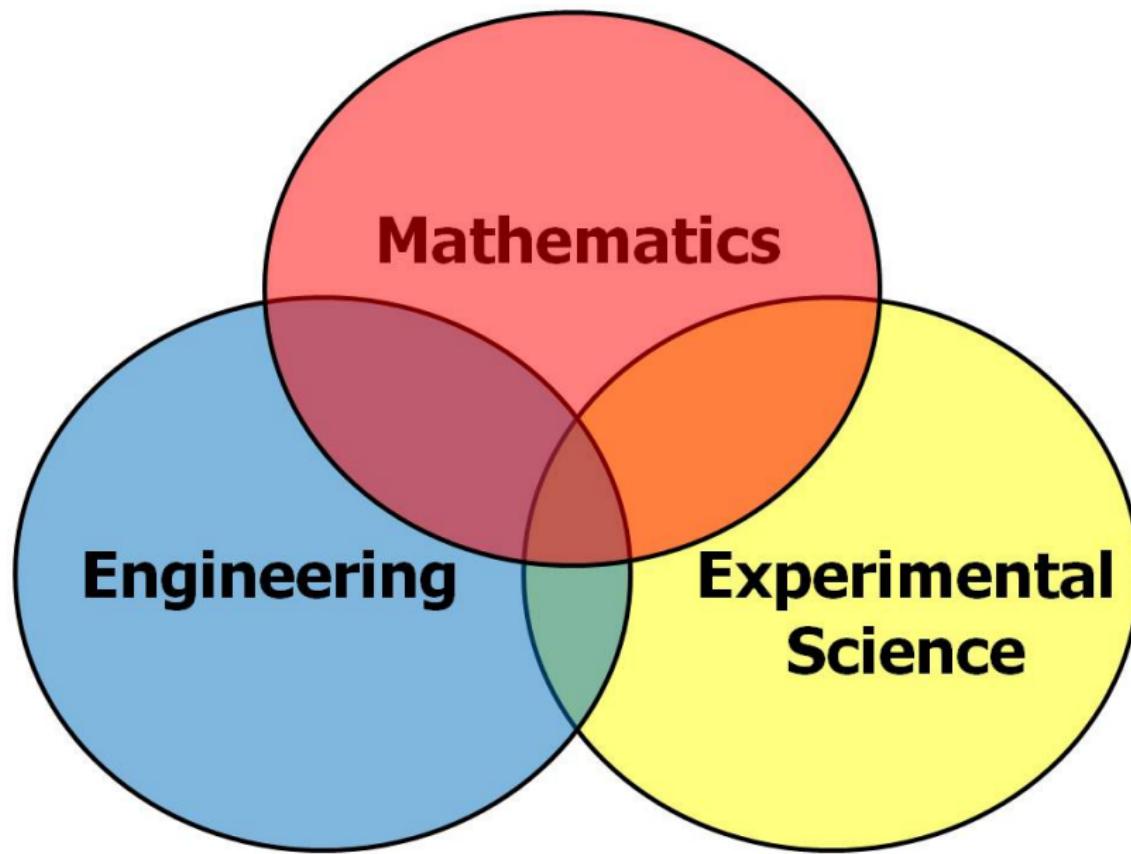
IBM DEEPBLUE (link)



GOOGLE ALPHAGO (link)



Constituents of Computer Science (1/2)



Constituents of Computer Science (2/2)

Mathematical aspects:

- computational complexity, formal languages, PL theory
- computational logic (e.g. automated theorem proving)
- cryptography, coding theory, quantum computing
- design/analysis of algorithms, algorithmic game theory, ...

Engineering aspects:

- computer architecture, operating systems, compilers
- parallel/distributed computing, network, databases, ...

Experimental science aspects:

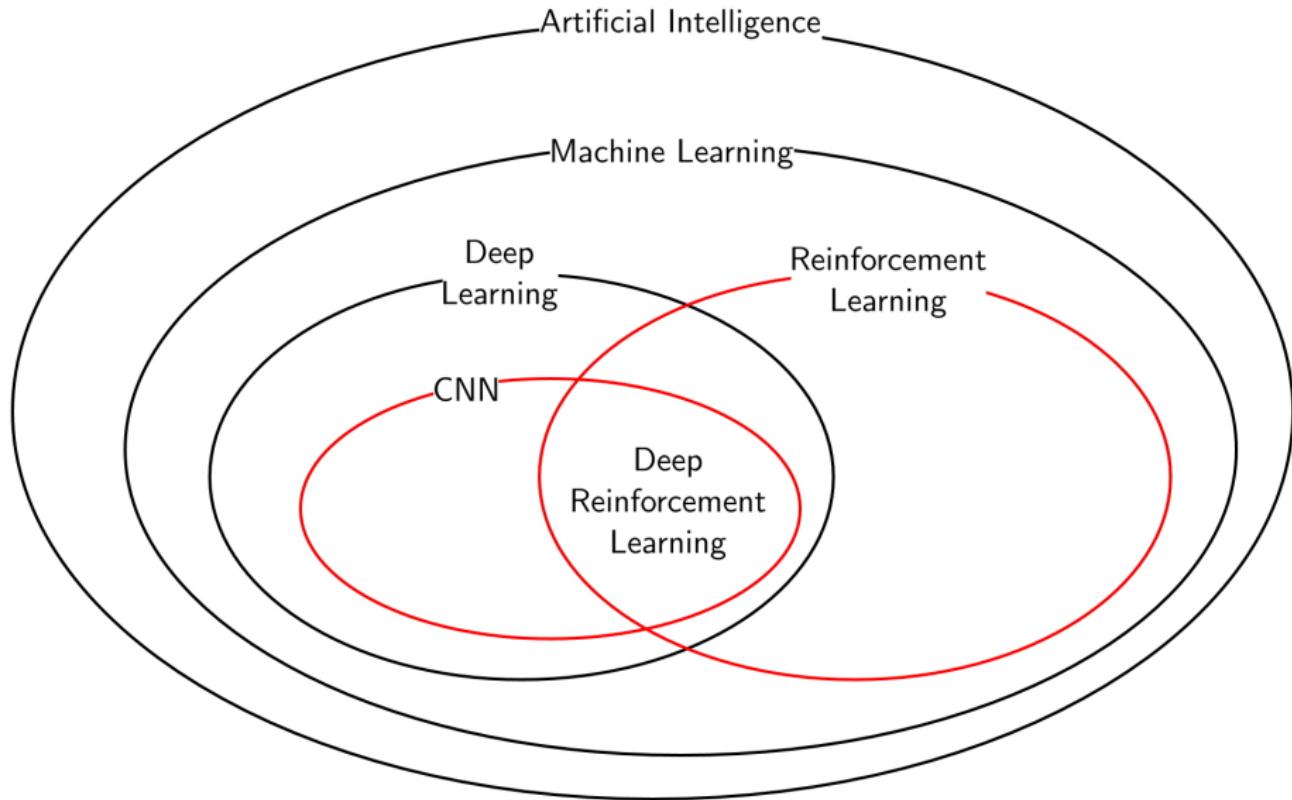
- **artificial intelligence**, computer graphics/vision, ...

Example: Typical Syllabus of Artificial Intelligent (AI) Course

Where are we now?

- Search & Planning
- Inductive Learning
 - ▶ (deep) neural networks, SVM, decision trees, ...
- Knowledge Representation & Reasoning
 - ▶ formal logic, automated theorem proving
 - ▶ symbolic programming, logic programming, ...
- Reasoning under uncertainty
 - ▶ decision theory, game theory
 - ▶ probabilistic reasoning, Bayesian networks, ...
- Applications
 - ▶ expert systems, image understanding, natural language processing, ...

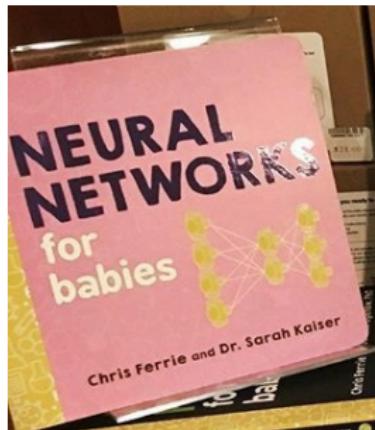
AI/ML Terminologies



요즘 대중매체에 등장하는 AI는 대부분 ML을 가르킴 (특히 DL)

AI/ML: 과열? 회의?

Artificial intelligence pioneer says we need to start over



- 모든 분야가 약간의 실용성을 갑자기 보이면 **과열됨**
- 10년 정도 지나면 **한계**가 보이면서 **회의론**이 일고 소강됨
- AI/ML은 두어번째 cycle을 도는 중
- 본 과목을 통해 ML 기법의 한계를 깨우치는 것이 가장 중요

ALPHAGO Hardware: NOT realistic!



- AlphaGo Fan (2015):
1,202 CPUs / 176 GPUs
- AlphaGo Lee (2016):
48 TPUs (\approx 1,000 GPUs)
 - ▶ even, TENSORFLOW is highly optimized for TPU



- AlphaGo Zero (2017): ...

ALPHAGO Hardware

vs.

Realistic Hardware..



Outline

1 Soft Computing

2 Placement Information

3 Topics/Plan

Hard Computing vs. Soft Computing: What for?

Algorithmic Problems & Algorithms

- Problem: a function $f : X \rightarrow Y$
 - ▶ X : set of inputs / Y : set of outputs
 - ▶ e.g. $f_{\text{bipartite_matching}}(\text{bipartite graph}) = \text{maximum matching}$
 $f_{\text{factoring}}(pq) = (p, q)$
- Algorithm for the problem f : Turing machine to compute f
 - ▶ Turing machine \equiv Python programs

Hard Computing (trained in discrete math/algorithm courses)

- Problem f : precisely defined in terms of math
- Algorithm for f : mathematically rigorous (correctness proof)

Soft Computing (\approx Artificial Intelligence)

- Problem f : usually ill-defined (or well-defined but intractable)
- Algorithm for f : unrigorous, heuristic, imprecise
- Role model = human mind

What problems are NOT suitable to soft computing?

Problem: MINIMUM-SPANNING-TREE (well-defined/tractable)

- input: a graph G
- output: a **minimum spanning tree** of G



- Well-defined combinatorial problem
- Even better, admits fast algorithmic solutions.
 - ▶ e.g. Prim's $O((|V|+|E|)\cdot\log|V|)$, Kruskal's $O(|E|\alpha(|E|,|N|))$
- Every tractable problem should be addressed by a poly-time algorithm with rigorous proof

What problems are NOT suitable to soft computing?

Problem: SUBSET-SUM (well-defined/intractable)

- input: a set $\{w_1, w_2, \dots, w_n\}$ of positive integers and W
- output: a subset $S \subseteq \{w_1, w_2, \dots, w_n\}$ s.t. $\sum_{x \in S} x = T$

1, 4, 16, 64, 256, 1040,
1041, 1093, 1284, 1344

3754



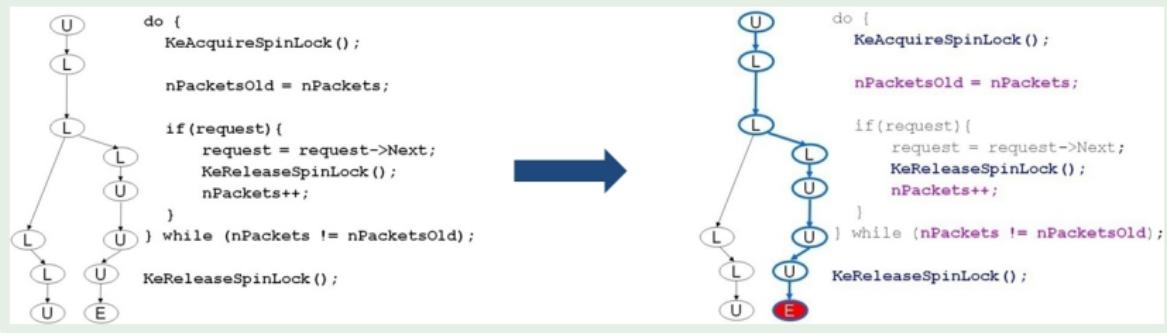
1, 4, 16, 64,
1040, 1093,
1284

- Well-defined combinatorial problem
- NP-hard in general
- But there are pseudo-polynomial-time algorithm and even fully polynomial-time approximation scheme (FPTAS)
- NP-hard 문제일지라도 subproblem analysis, approximation 등의 수학적으로 엄밀한 기법을 최대한 시도해야만 함

What problems are NOT suitable to soft computing?

Problem: FORMAL-VERIFICATION (well-defined/undecidable)

- input: a program P
- output: a correctness proof of P or bugs (e.g. deadlocks) in P



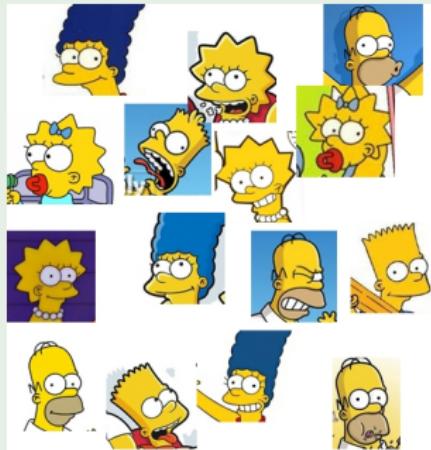
- Well-defined problem in PL theory & computational logic
- Undecidable in general (Rice's theorem)
- Nevertheless, lots of algorithmic solutions for subproblems
 - ▶ temporal-logic model checking, automated theorem proving, abstract interpretation, Hoare logic, separation logic, ...
- Well-defined면 undecidable일지라도 엄밀한 기법 고수해야

What problems are suitable to soft computing?

Problem: THE-SIMPSONS

(ill-defined)

- input: a **photograph** of the Simpsons (in .jpg format)
- output: the **person** in the photograph
 - ▶ e.g. $f_{\text{Simpsons}}(\text{Homer}) = \text{"Homer"}$

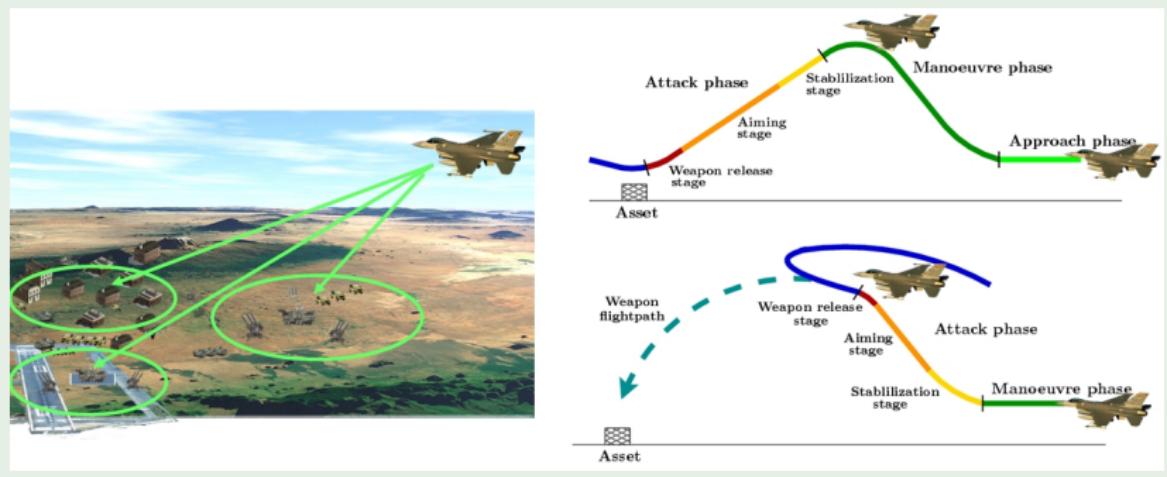


- NOT **well-defined** in terms of mathematical language
- 수학적 정의조차 불가하므로 엄밀한 방식으로 해결불가

Real-world applications of soft computing

Problem: THREAT-EVALUATION (for anti-air missile systems)

- input: a **flight profile** of a threat aircraft
- output: the estimated **asset** to be **attacked** & attack **strategy**



- SPEECH-RECOGNITION: Given a .mp3 file, transcript it down
- BANKRUPTCY: Given a bankbook, issue credit card or not

Soft computing for well-defined problems

Sometimes useful for **well-defined** (but highly intractable) problems

Problem: Go (well-defined / intractable)

- input: a configuration of $C \in \{\text{백, 흑, 무}\}^{[19]^2}$ of Go game
- output: $(i, j) \in [19]^2$ s.t. $C||(i, j)$ is a **winning** configuration



- 바둑은 PSPACE-hard/EXP-hard으로 NP-hard 문제와 달리 subproblem analysis/approximation 기법 등이 잘 안통함
- 물론 soft computing으로도 winning move를 항상 찾을 수는 없으나 “꽤 좋은” move는 찾을 수도 있음
- NP-hard라 꽤 좋은 엄밀한 기법이 존재하는 automated theorem proving 문제에서도 soft computing 적용 논의중

Summary: Places for soft computing

문제 $f : X \rightarrow Y$ 가 ill-defined인 경우

- 수학적으로 엄밀한 기법은 불가능으로 어쩔 수 없이 고려
- 기계학습의 경우 (input,output) pair가 매우 풍부해야 함
 - ▶ 매우 풍부하지 않으면 기계학습 기법 적용 불가
 - ▶ 예: 이세돌과의 5판 기보는 기계학습에는 전혀 도움 안됨 (overfitting). 단 rule-based hard computing 방식에는 도움됨

문제 $f : X \rightarrow Y$ 가 well-defined인 경우

- Poly-time computable: 수학적으로 엄밀한 기법만 사용해야
- NP-easy: 엄밀한 기법만으로 버티되 최후의 경우에나 고려
 - ▶ subproblem analysis, approximation으로 버티면 대부분 해결
 - ▶ 그렇지 않은 경우에도 2-OPT/3-OPT 같이 효율적인 휴리스틱을 사용하면 실험적 성능은 좋음
 - ▶ 자동증명정리 같이 매우 복잡한 경우에 부분적 적용 가능
- PSPACE-hard: NP-easy인 경우보다는 적절한 경우 많음
 - ▶ 엄밀한 기법은커녕 좋은 휴리스틱도 별로 없어서

Outline

1 Soft Computing

2 Placement Information

3 Topics/Plan

Course Objectives

Recall: Places for soft computing

- **III-defined problems** (기계학습의 경우 **입출력 쌍도 풍부**)
- Well-defined, but **highly intractable** (**PSPACE/EXP-hard**)
- 기계학습 기법을 널리 전파하는 것이 과목의 목표가 **아님**
 - ▶ 대학교/대학원에서 깊이/넓게 다루는 수업이 많음
- 기계학습을 **어쩔 수 없이 사용해야만 하는 상황**을 제대로 **판별**하고 **한계**를 인지하고 **조심스럽게 사용**하게 함
 - ▶ NP-completeness를 CS에서 꾸준히 다룬 것과 유사한 맥락
- 기계학습 그 자체가 목적이 아니고 각 학생들이 관심있어 하는 **문제**에 기계학습을 **하나의 도구로서 적용**함을 체험
 - ▶ 기계학습은 알고리즘/이산수학과 달리 **교육적 가치**가 낮음
- 알파고 이벤트 정도에 휘둘리지 않게 할 **교양**을 갖추게 함
 - ▶ c.f. 소수판별용 AKS 알고리즘 등장때 RSA가 깨졌다라는 **crap**

Level of Difficulty/Workload

- Prerequisites: 수학적 언어 구사 능력 관련
 - ▶ a little **discrete math** (e.g. combinatorial games)
 - ▶ a little **probability/statistics, linear algebra, 다변수 calculus**
- Prerequisites: 프로그래밍 능력 관련
 - ▶ combinatorial games의 exact solver와 Monte-Carlo tree search
 - ▶ 5×5 바둑 코드를 이해할 정도
 - ▶ **TENSORFLOW** API를 잘 가져다 쓸 수 있을 정도
 - ▶ 오늘 Python/numpy 요약 수업

인공지능/기계학습 과목의 "교육적" 가치에 대한 "개인적" 의견

- 90년대 대학의 인공지능 과목의 첫 시간 수업 분위기..
 - ▶ 인공지능 분야를 전공하고 평생을 헌신해오신 분들의 입장이 저렇다면 ..
- 실용적 가치는 분명히 충분하고 실제로 체험하고 있음
- 알고리즘/이산수학 교과서는 수십년 후에도 (새로운 내용이 추가가 될지언정) 큰 틀은 변함이 없을 것
- 인공지능 분야는 다루는 문제자체가 잘 정의되지도 않는 것들을 다루다 보니 수학적으로 성능이 완벽히 보장되는 답을 줄 수 없어 **주류 기법이 끊임없이 바뀔 수 밖에 없음**
- 수업에서는 **변화 없이 오래가고 널리 사용되는 기본기** 과목을 혹독하게 훈련시키는 것이 **효과적**
- **현존하는 주류 기법** 위주로 가볍게 공부하고 **활용**에 주안점을 두고, 인공지능을 전공할 경우에만 넓고 깊게

Outline

1 Soft Computing

2 Placement Information

3 Topics/Plan

Topics to be covered

- Machine learning 개요
 - ▶ 구성요소: model architecture, objective func., learning algo.
 - ▶ 학습 방식: supervised vs. unsupervised vs. reinforcement
 - ▶ 현존 기법: NN, SVM, Bayesian network, decision tree, probabilistic graph model, MCMC, k -nearest neighbor
- Software toolkit for machine learning
 - ▶ numpy library, CUDA programming for general purpose GPU
 - ▶ SW package for machine learning: **TENSORFLOW**/KERAS
- Neural networks 개요
 - ▶ multi-layer feed-forward networks, back-propagation learning
- Deep neural networks
 - ▶ convolutional NN, recurrent NN, deep belief network
- Reinforcement learning (SARSA, Q-learning, **deep Q-learning**)
- Case study: **DQN**, **ALPHAGO**

Topics to be covered

요약하면,

- deep NN + reinforcement learning 관련 theory/practice

DNN 위주의 협소적이고 그것마저 가볍게 다루는 이유:

- 시간이 매우 부족
- 주류 기법은 계속 바뀌어왔고 앞으로도 바뀜 (예: SVM)
- DNN도 몇 년 후에는 다른 기법에 의해 사장될..
- DNN을 black box처럼 가져다 쓸 수 있는 software toolkit이 많고, 이것들을 잘 쓰는 것이 더 중요
- 인공지능/기계학습 분야를 세부전공할 것이 아닌 이상 자신의 분야에 활용만 잘 해도 충분하고 그것도 어려움
 - ▶ 기계학습-assisted 자동정리증명
 - ▶ 알파고도 CNN/TORCH7/TENSORFLOW를 그대로 가져다 씀. 실제 재현시 parallelized MCTS 구현이 어려움

Programming Practice

Software toolkit for machine learning

- VMware + Ubuntu 16.04 + GOOGLE TENSORFLOW v1.10
 - ▶ TENSORFLOW가 64비트만 지원. Windows가 64bit여야
- 개발환경이 모두 설치된 Ubuntu image가 준비됨 (8GB 정도)
- MacOS: 부트캠프로 윈도우 설치 또는 VMware for MacOS
- Windows에 TENSORFLOW를 직접 설치하는 것은 권장안함
- GPU(TITAN-X) 장착된 Ubuntu 서버 사용 가능 (telnet/ftp)

TENSORFLOW 프로그래밍 실습/과제

- regression, k-clustering (기본 API)
- 숫자/그림 인식 (NN, CNN)
- 미로, breakout, catch (SARSA, Q-learning, deep Q-learning)
- tic-tac-toe, 5×5 바둑

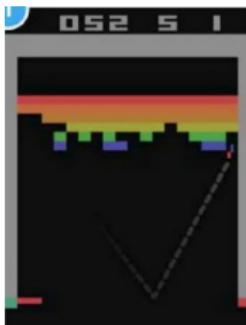
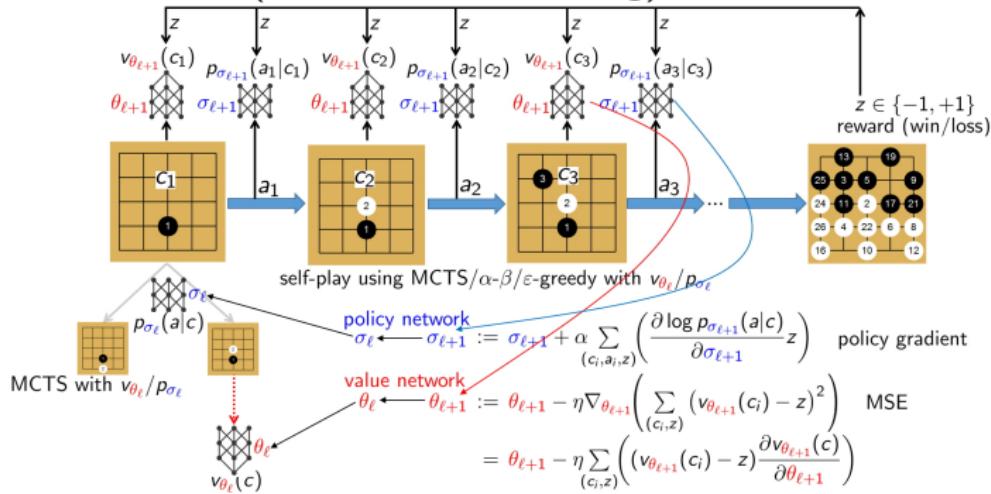
Programming Practice: Examples

지도 학습 (supervised learning)

0	4	1	9	2	1	3	1	4	3
5	3	6	1	7	2	8	6	9	4
0	9	1	1	2	4	3	2	7	3
8	6	9	0	5	6	0	7	6	1
8	7	9	3	9	8	5	9	3	3
0	7	4	9	8	0	9	4	1	4
4	6	0	4	5	6	1	0	0	1
7	1	6	3	0	2	1	7	9	0
0	2	6	7	8	3	9	0	4	6
7	4	6	8	0	7	8	3	1	Σ



강화 학습 (reinforcement learning)



Schedule (tentative)

#	Topics	Practice/Assignment
1	Course Overview Python Language	tool installation numpy library
2	Machine Learning Overview TENSORFLOW Basics	regression, <i>k</i> -clustering
3	Neural Network	digit classifier
4	Convolutional Neural Network	image classifier
5	Reinforcement Learning	maze, breakout, catch
6	Deep Reinforcement Learning	
7	Combinatorial Games Search Heuristics for Games	simple games, tic-tac-toe
8	Monte-Carlo Policy Iteration ALPHAGO	5×5 go