ACSIP- Association of Chinese Senior IT Professionals

Post-AlphaGo Era

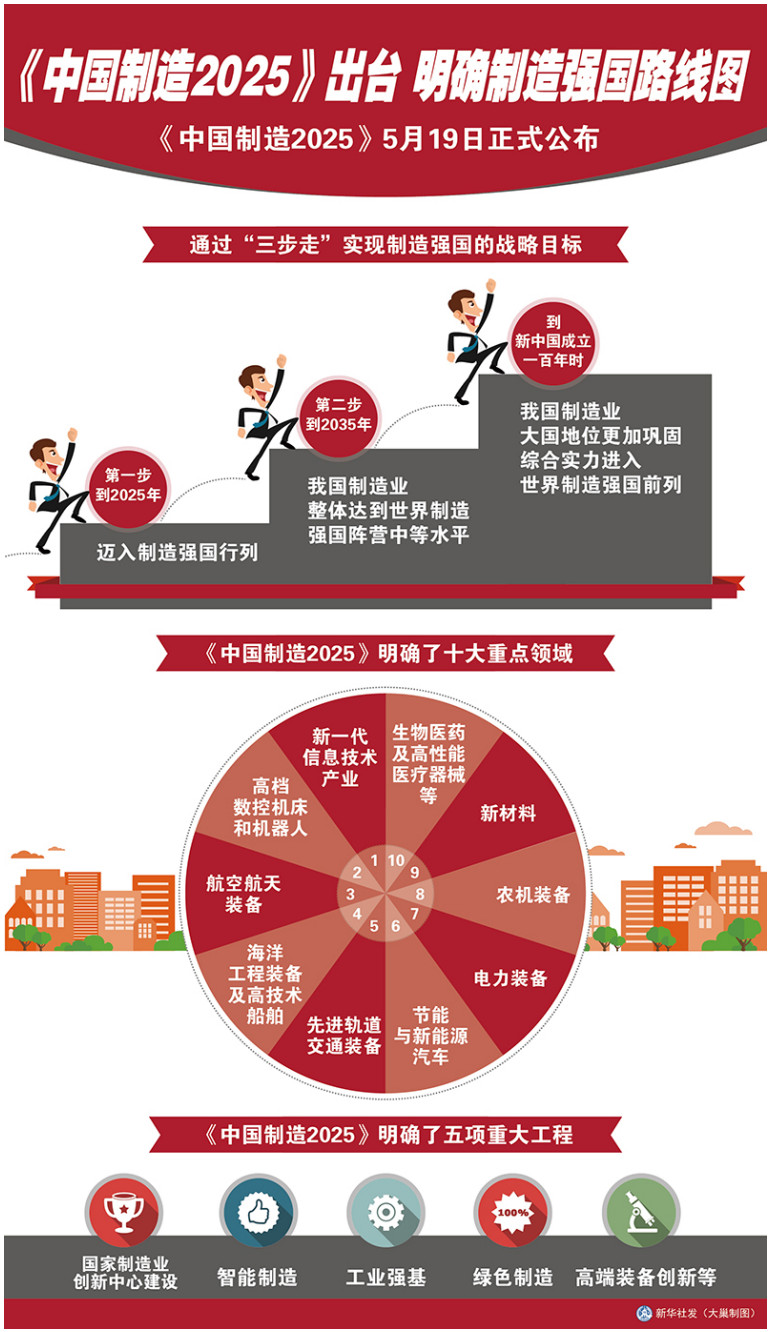
Deep Reinforcement Learning

An Introduction and Applications

Yuxi Li (yuxili@gmail.com)

Investment in AI: Canada

- In Budget 2017, the Government of Canada announced **$125 million** in funding for a Pan-Canadian Artificial Intelligence (AI) Strategy to be led by the Canadian Institute for Advanced Research (CIFAR).
  - Establish interconnected nodes of scientific excellence in Canada's three major centres for artificial intelligence in **Edmonton** ($35M/$125M), **Montreal** and **Toronto**.
- Vector Institute, Toronto
- Element AI, Montreal
- Deepmind Alberta, Edmonton
  - Deepmind's first international research lab outside of UK



3月5日，2017年全国两会在京召开。李克强总理在政府工作报告中表示，要“全面实施战略性新兴产业发展规划，加快新材料、人工智能、集成电路、生物制药、第五代移动通信等技术研发和转化”。这是人工智能第一次被写进了政府工作报告。这也表明人工智能技术已经成为了中国的国家战略。

此外，政府工作报告还提出，“大力改造提升传统产业。深入实施《中国制造2025》，加快大数据、云计算、物联网应用，以新技术新业态新模式，推动传统产业生产、管理和营销模式变革。把发展智能制造作为主攻方向，推进国家智能制造示范区、制造业创新中心建设，深入实施工业强基、重大装备专项工程，大力发展先进制造业，推动中国制造向中高端迈进”。在《中国制造2025》中，明确了机器人技术成为国家战略。机器人作为智能制造的重要抓手，必然也会成为2017年政府扶持重点。

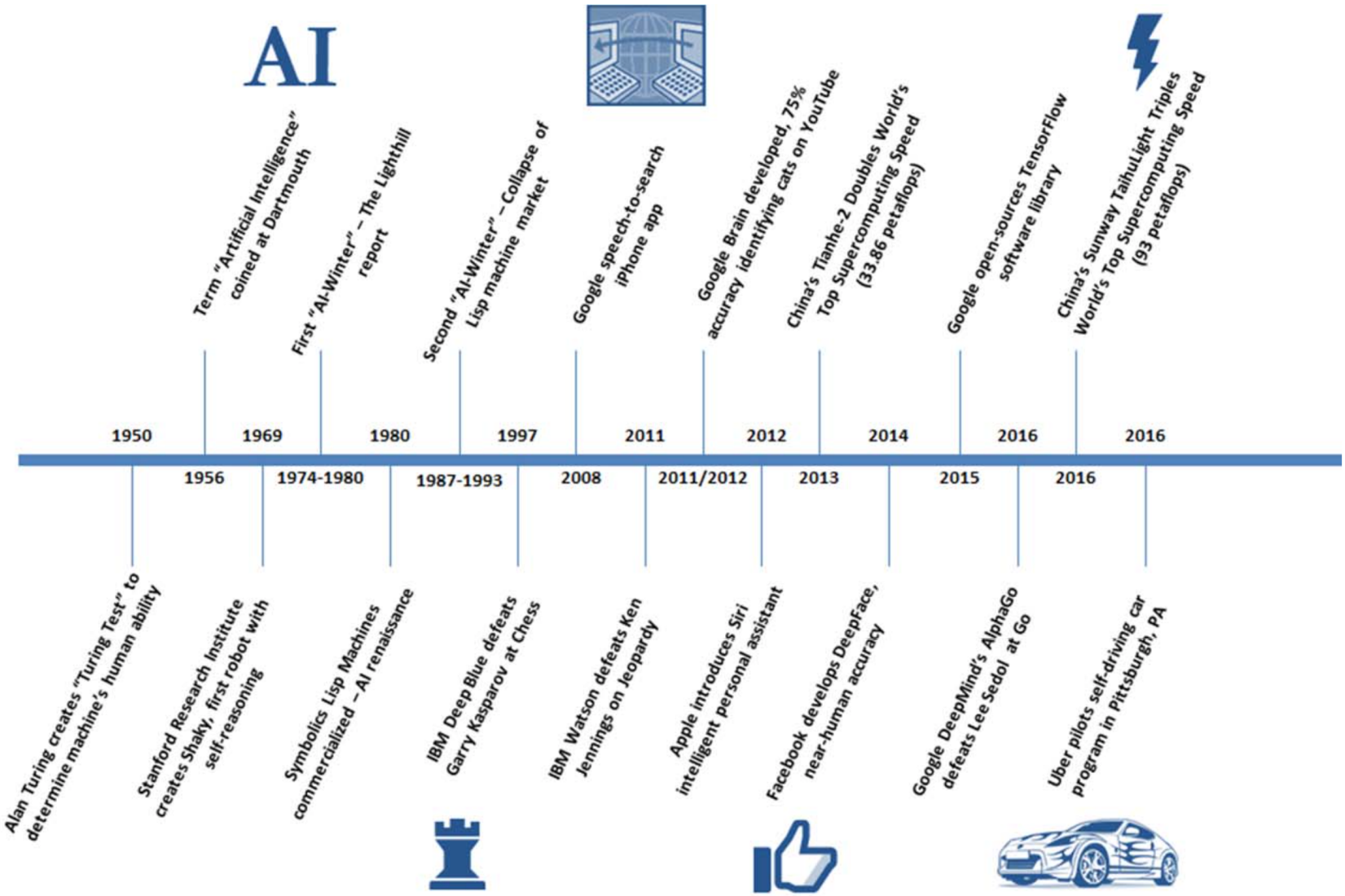
EMBARGOED UNTIL 4:30 PM ET, DECEMBER 20, 2016

Artificial Intelligence, Automation, and the Economy

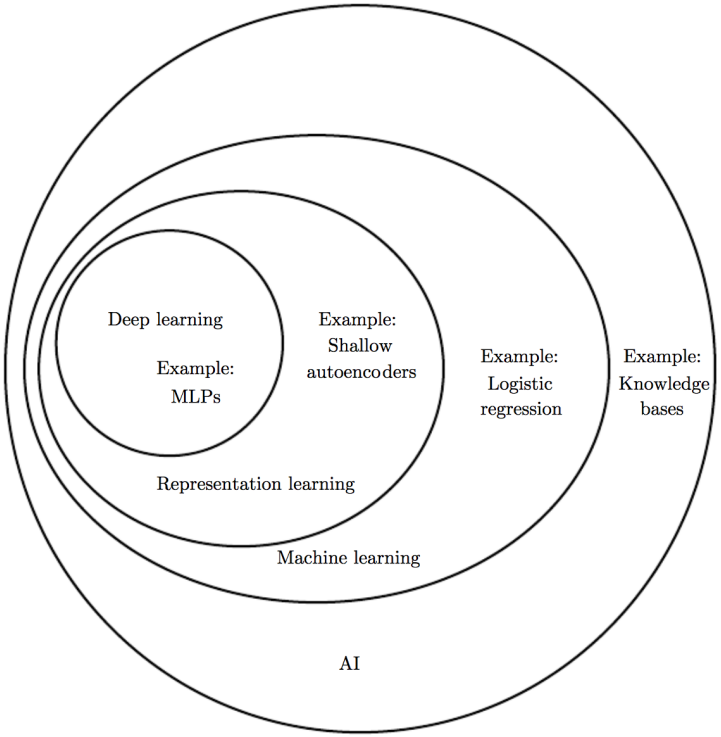
Executive Office of the President

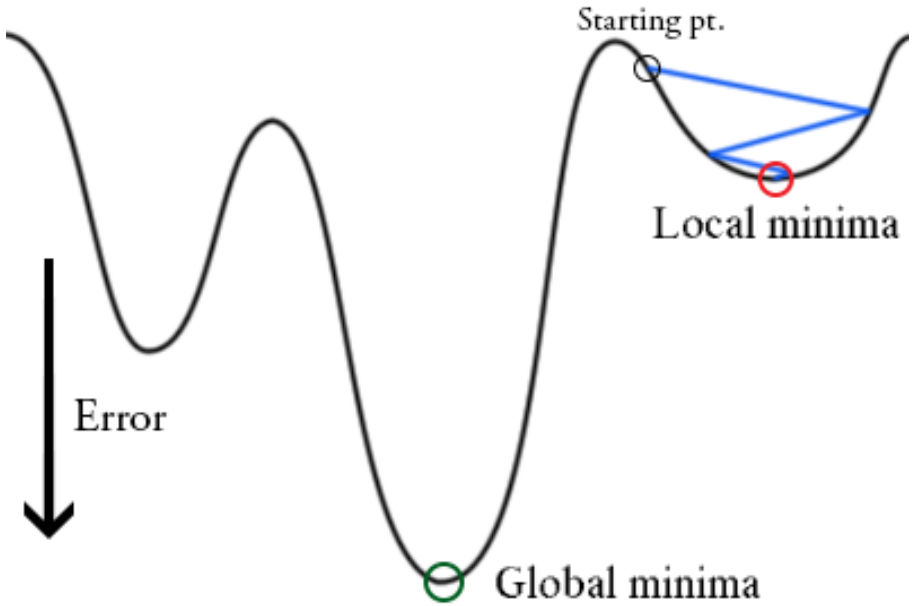
December 2016

Exhibit 8: Evolution of AI: 1950-Present

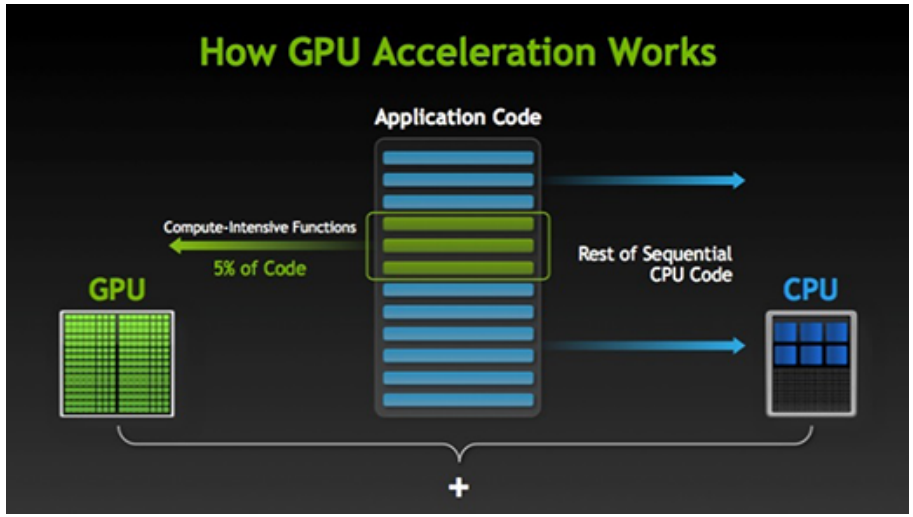


AI, Machine Learning and Data Fuel the Future of Productivity, The Goldman Sachs Group, Inc., Nov. 2016


Deep learning
Example: MLPs
Example: Shallow autoencoders
Example: Logistic regression
Example: Knowledge bases
Representation learning
Machine learning
AI

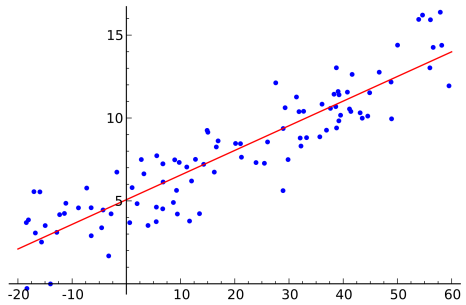

Starting pt.
Local minima
Error
Global minima

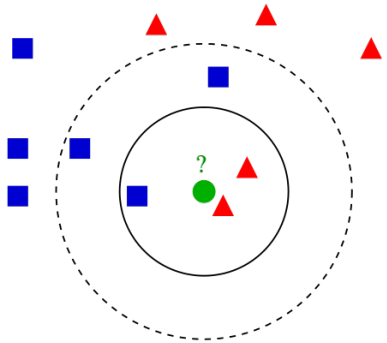

analytics
data
increased
make
companies
easier
production
investment
platform
Text
things
Spark
use
top
digital
Big
Hadoop
algorithms
e.g
security
toward
Real-time
sensor
source
adoption
Apache
learning
Deep
continue
open
marketing
internet
Move
machine
Enterprises
logs
get
processing

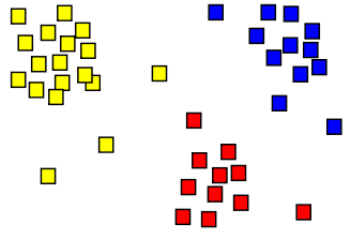

How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
Rest of Sequential CPU Code
GPU
CPU
+

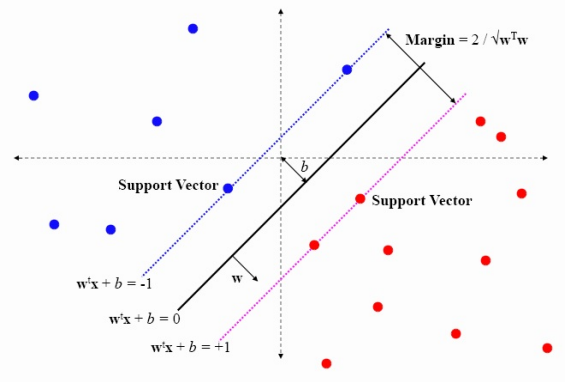
Machine Learning

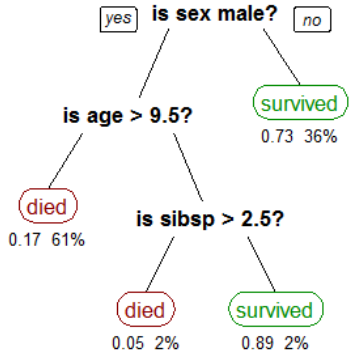
- Machine learning is about learning from data and making predictions and/or decisions.
- In supervised learning, there are labeled data.
  - making predictions
  - classification, regression
- In unsupervised learning, there are no labeled data.
  - extract information from data without labels
  - clustering, PCA
- In reinforcement learning, there are evaluative feedbacks, but no supervised signals.
  - making predictions and/or decisions
  - AlphaGo, shortest path

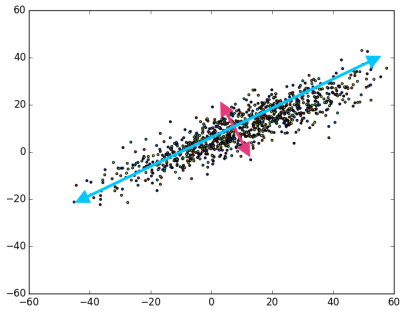




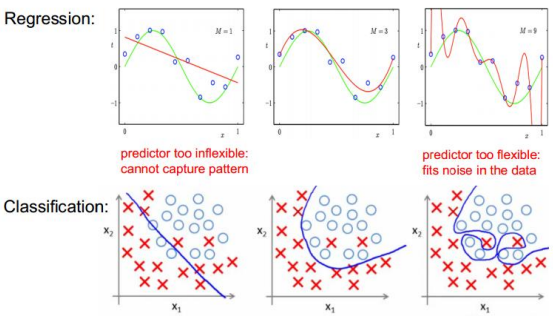








Under- and Over-fitting examples



No Free Lunch Theorem

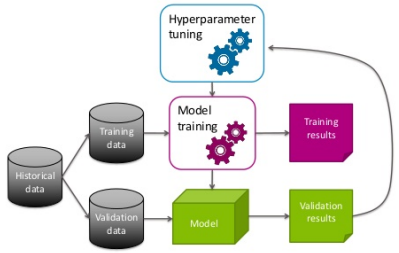
All models are wrong, but some models are useful. - George Box

There are many machine learning models.

There is no single best model that works best for all problems.

Occam's razor: parsimony implies predictive power.

Model Selection and Tuning



Deep Learning in a nutshell

DL is a general-purpose framework for representation learning

- given an objective
- learn representation that is required to achieve objective
- directly from raw inputs
- using minimal domain knowledge

(David Silver, ICML 2016 Tutorial: Deep Reinforcement Learning)

the choice of representation has an enormous effect on the performance of machine learning algorithms

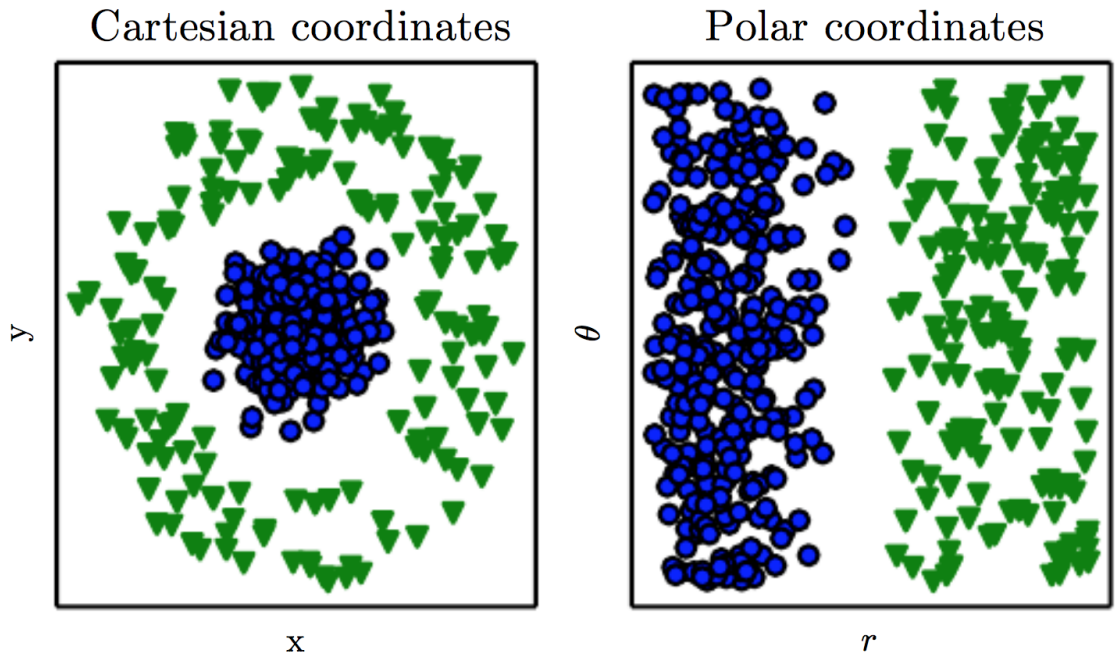
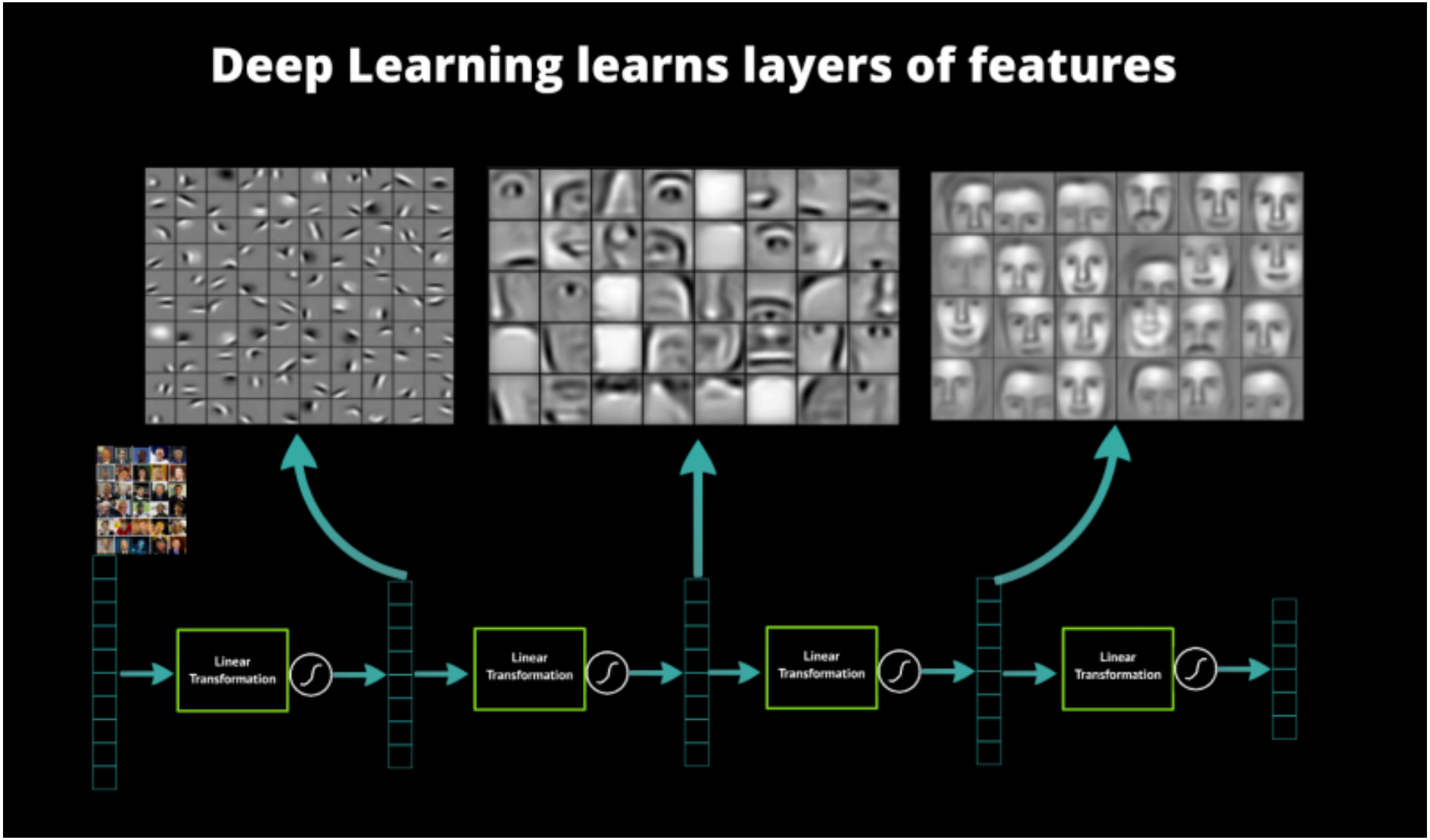


Figure 1.1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line. (Figure produced in collaboration with David Warde-Farley)



(https://www.datarobot.com/blog/a-primer-on-deep-learning/)

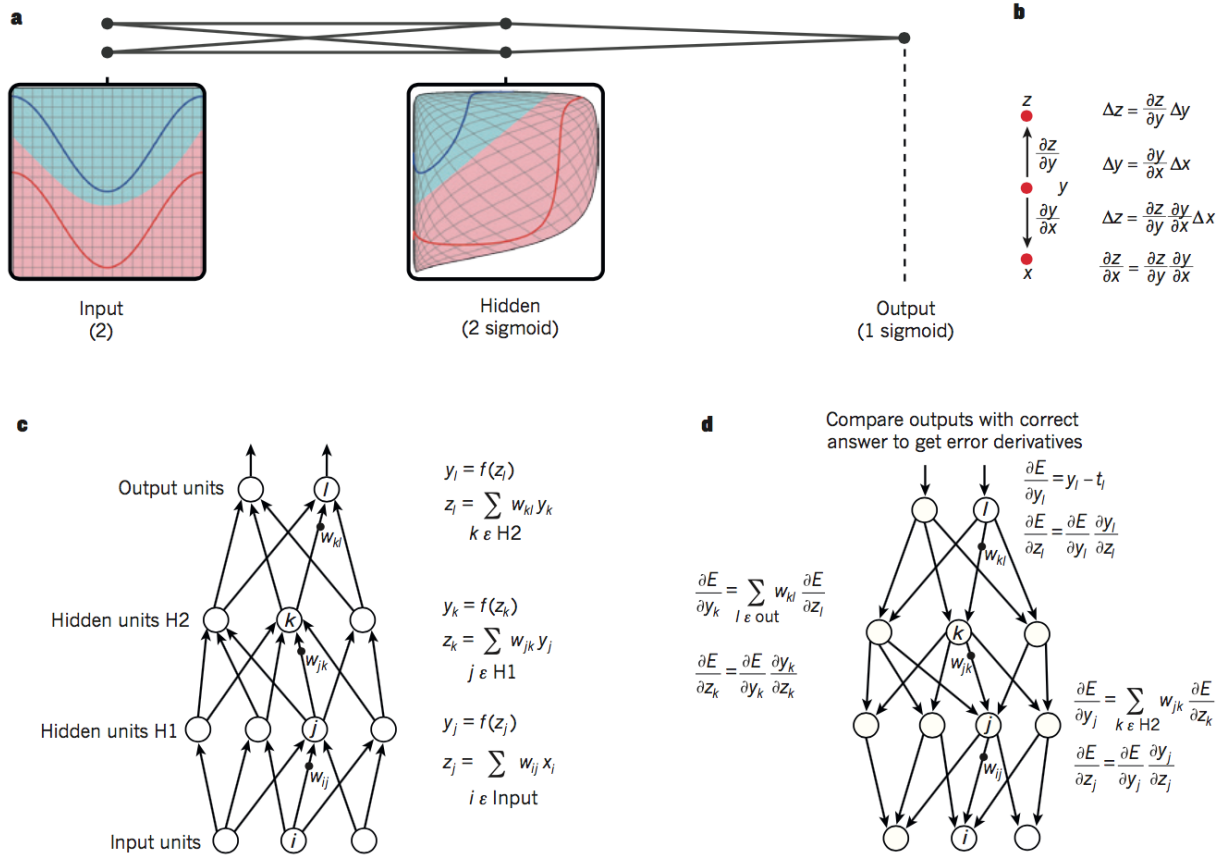


Figure 1 | Multilayer neural networks and backpropagation. **a**, A multilayer neural network (shown by the connected dots) can distort the input space to make the classes of data (examples of which are on the red and blue lines) linearly separable. Note how a regular grid (shown on the left) in input space is also transformed (shown in the middle panel) by hidden units. This is an illustrative example with only two input units, two hidden units and one output unit, but the networks used for object recognition or natural language processing contain tens or hundreds of thousands of units. Reproduced with permission from C. Olah (http://colah.github.io/). **b**, The chain rule of derivatives tells us how two small effects (that of a small change of x on y, and that of y on z) are composed. A small change Δx in x gets transformed first into a small change Δy in y by getting multiplied by $\partial y/\partial x$ (that is, the definition of partial derivative). Similarly, the change Δy creates a change Δz in z. Substituting one equation into the other gives the chain rule of derivatives — how Δx gets turned into Δz through multiplication by the product of $\partial y/\partial x$ and $\partial z/\partial x$. It also works when x, y and z are vectors (and the derivatives are Jacobian matrices). **c**, The equations used for computing the forward pass in a neural net with two hidden layers and one output layer, each constituting a module through which one can backpropagate gradients. At each layer, we first compute the total input z to each unit, which is a weighted sum of the outputs of the units in the layer below. Then a non-linear function $f(.)$ is applied to z to get the output of the unit. For simplicity, we have omitted bias terms. The non-linear functions used in neural networks include the rectified linear unit (ReLU) $f(z) = \max(0,z)$, commonly used in recent years, as well as the more conventional sigmoids, such as the hyberbolic tangent, $f(z) = (\exp(z) - \exp(-z))/(\exp(z) + \exp(-z))$ and logistic function logistic, $f(z) = 1/(1 + \exp(-z))$. **d**, The equations used for computing the backward pass. At each hidden layer we compute the error derivative with respect to the output of each unit, which is a weighted sum of the error derivatives with respect to the total inputs to the units in the layer above. We then convert the error derivative with respect to the output into the error derivative with respect to the input by multiplying it by the gradient of $f(z)$. At the output layer, the error derivative with respect to the output of a unit is computed by differentiating the cost function. This gives $y_l - t_l$ if the cost function for unit l is $0.5(y_l - t_l)^2$, where t_l is the target value. Once the $\partial E/\partial z_k$ is known, the error-derivative for the weight w_{jk} on the connection from unit j in the layer below is just $y_j\, \partial E/\partial z_k$.

Deep Learning Achievements

- Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years.
- It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government.
- image recognition, speech recognition, predicting the activity of potential drug molecules, analyzing particle accelerator data, reconstructing brain circuits, and predicting the effects of mutations in non-coding DNA on gene expression and disease
- natural language understanding, particularly topic classification, sentiment analysis, question answering and language translation
- deep learning will have many more successes in the near future: it requires very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data

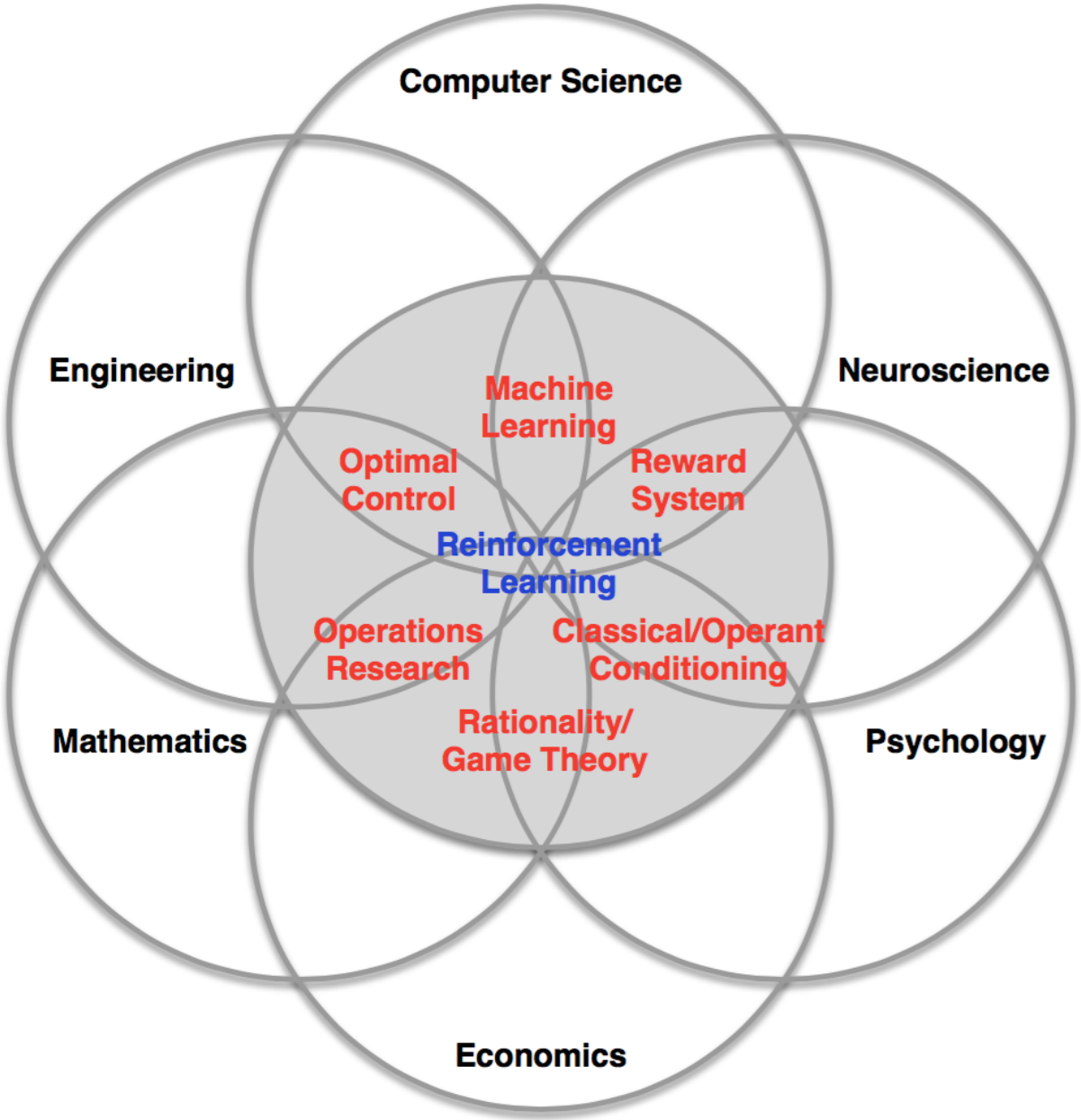
(LeCun, Bengio and Hinton, Deep Learning, Nature, May 2015)

Reinforcement Learning in a nutshell

RL is a general-purpose framework for decision-making

- RL is for an agent with the capacity to act
- each action influences the agent's future state
- success is measured by a scalar reward signal
- goal: select actions to maximise future reward

(David Silver, ICML 2016 Tutorial: Deep Reinforcement Learning)


Computer Science
Engineering
Neuroscience
Machine Learning
Optimal Control
Reward System
Reinforcement Learning
Operations Research
Classical/Operant Conditioning
Rationality/ Game Theory
Mathematics
Psychology
Economics

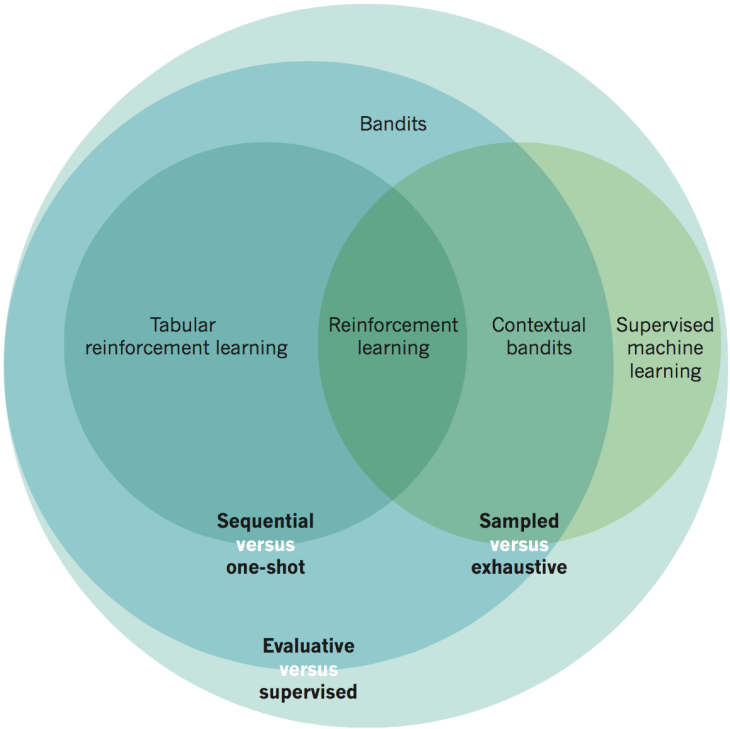


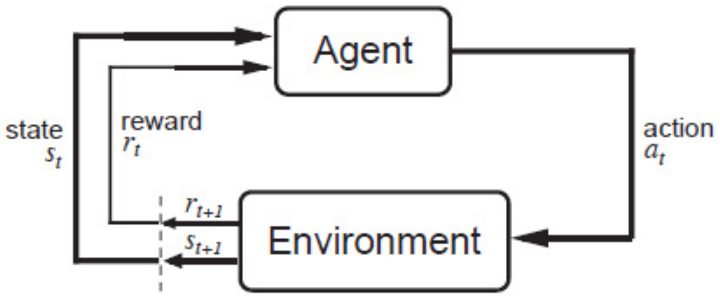
Figure 1 | Decisions of machine-learning feedback. Three kinds of feedback (sequential versus one-shot, sampled versus exhaustive and evaluative versus supervised) in machine learning and examples of learning problems (bandits, tabular reinforcement learning, reinforcement learning, contextual bandits and supervised machine learning) that result from their combination.

- exhaustive versus sampled feedback
  - coverage of training examples
- supervised versus evaluative feedback
  - how the learner is informed of right and wrong answers
  - supervised learning with known optimal decisions
  - evaluative feedback with an assessment of the effectiveness of the decisions; no information on the appropriateness of alternatives
- one-shot versus sequential feedback
  - relative timing of learning signals
  - long term impacts evaluated over a sequence of decisions

RL methods encounter all three forms of weak feedback simultaneously - sampled, evaluative and sequential feedback

(Michael Littman, Reinforcement learning improves behaviour from evaluative feedback, Nature, May 2015)

The Agent-Environment Interaction in RL

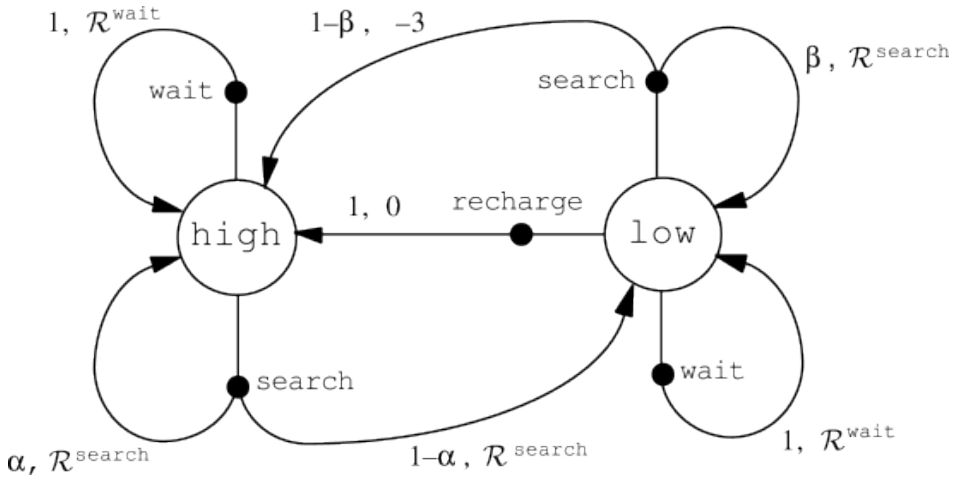


At each time step t, the agent

- receives state s_t
- selects an action a_t
- receives a numerical reward r_{t+1}
- transitions into a new state s_{t+1}

(Richard Sutton and Andrew Barto, Reinforcement Learning: An Introduction, MIT Press, 1998)

An Example: Recycling Robot



- S: a set of states, situations in which a decision can be made
- A: a set of actions, the decisions the decision maker can select
- P: the transition model, $P^a_{ss'} = P\{s_{t+1} = s' | s_t = s, a_t = a\}$
- R: the reward model, $R^a_{ss'} = E\{r_t | s_t = s, a_t = a, s_{t+1} = s'\}$
- γ: the discount factor, $\gamma \in [0, 1]$
- Goal: maximizing the cumulative discounted expected reward

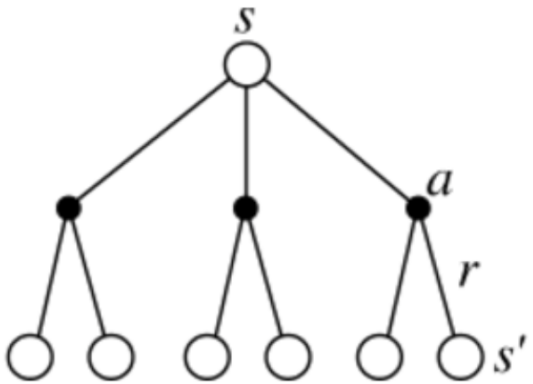
Components of a RL Agent

- policy, the agent's behavior, a map from state to action
  - deterministic policy, $a = \pi(s)$
  - stochastic policy, $\pi(s|a) = P(s|a)$
- value function, how good is each state and/or action
  - a prediction of future reward
- model, agent's representation of the environment
  - P: the transition model
  - R: the reward model
  - usually learned from experience (s, a, r, s')
  - sometimes perfect model, e.g., game rules are known

Value Function

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$$

Bellman equation: $V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')]$



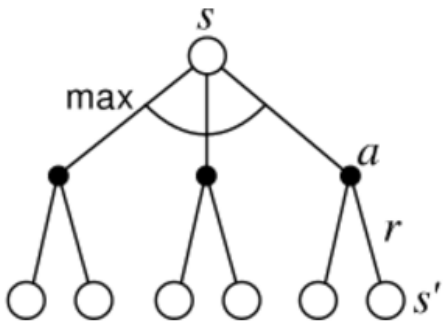
Optimal Value Function

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

Bellman optimal equation: $V^*(s) = \max_{a \in A(s)} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$

Markov decision processes (MDPs) satisfy two requirements applying dynamic programming (DP): optimal substructure by Bellman equation; overlapping subproblems with value function.

Given $V^*(s)$, the actions that are best after a one-step search will be optimal actions.



Temporal Difference Learning

- a central idea in reinforcement learning
- learn directly from experience
- learn from temporal difference error
- learn a guess from a guess - bootstrapping
- model-free, online, fully incremental

Tabular TD(0) for estimating v_π

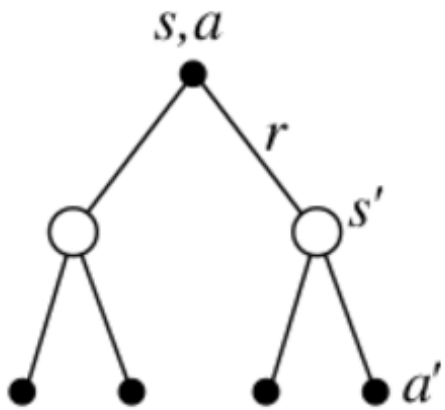
Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Q-Value Function

Q-value function gives expected total reward, from state s and action a, under policy π, with discount factor γ.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} | s_t = s, a_t = a\}$$

Bellman equation: $Q^{\pi}(s,a) = \mathbb{E}_{s',a'}[r + \gamma Q^{\pi}(s',a') | s,a]$



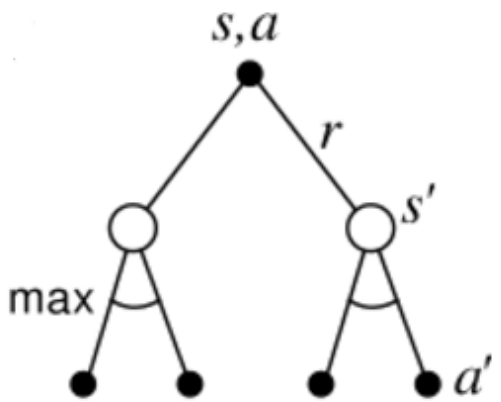
Optimal Q-Value Function

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

Q-learning

Bellman optimal equation: $Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^\pi(s',a')|s,a]$

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$



Q-learning: An off-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

- The **prediction** problem, or policy evaluation, is to compute the state or action value function for a policy.The **control** problem is to find the optimal policy. **Planning** constructs a value function or a policy with a model.
- **On-policy** methods evaluate or improve the behavioural policy.
  ○ SARSA, representing state, action, reward, (next) state, (next) action, is an on-policy control method, with the update rule, $Q(s_t, a_t) \leftarrow Q(S,A) + \alpha[r + \gamma Q(S', A') - Q(S,A)]$.
  ○ SARSA fits the action-value function to the current policy, i.e., SARSA evaluates the policy based on samples from the same policy, then refines the policy greedily with respect to action values.
- In **off-policy** methods, an agent learns an optimal value function/policy, maybe following an unrelated behavioural policy.
  ○ e.g., Q-learning attempts to find action values for the optimal policy directly, not necessarily fitting to the policy generating the data, i.e., the policy Q-learning obtains is usually different from the policy that generates the samples.
- The notion of on-policy and off-policy can be understood as same-policy and different-policy.
- The **exploration-exploitation** dilemma is about the agent needs to exploit the currently best action to obtain rewards, yet it has to explore the environment to find better actions.
- In **model-free** methods, the agent learns with trail-and-error from experience explicitly; the model (state transition function) is not known or learned from experience. RL methods that use models are **model-based** methods.
- In **online** mode, training algorithms are executed on data acquired in sequence. In **batch** mode, models are trained on the entire data set.
- With **bootstrapping**, an estimate of state or action value is updated from subsequent estimates.

Function Approximation (FA)

- tabular
- state aggregation
- linear FA
- non-linear FA (neural/deep networks)

- state value function: $V(s;\theta) \approx V^{\pi}(s)$
- state-action value function: $Q(s,a;\theta) \approx Q^{\pi}(s,a)$
  - (stochastic) gradient descent to find optimal value function
- policy: $\pi(s,a;\theta) \approx \mathbb{P}(a|s;\theta)$
  - policy gradient to find optimal policy
  - e.g. with likelihood ratio in REINFORCE

Approaches to Reinforcement Learning

- value-based RL
  - estimate the optimal value function $V^*(s)$ or $Q^*(s, a)$
  - this is the maximum value achievable under any policy
- policy-based RL
  - search directly for the optimal policy π^*
  - this is the policy achieving maximum future reward
- model-based RL
  - build a model of the environment
  - plan (e.g. by lookahead) using model

(David Silver, ICML 2016 Tutorial: Deep Reinforcement Learning)

The Space of RL Methods

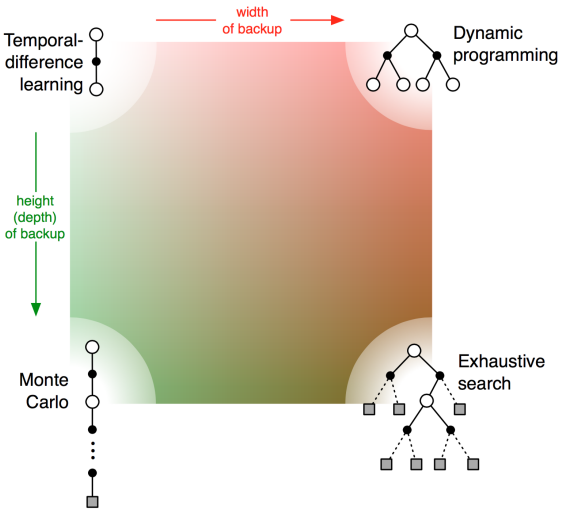


Figure 17.1: A slice of the space of reinforcement learning methods.

search width, depth

- TD, width $= 1$, depth $= 1$
- MC, width $= 1$, depth $= \infty$
- DP, width $= \infty$, depth $= 1$
- ES, width $= \infty$, depth $= \infty$

four dimensions

- sample or full (width of) backup
  - model-based or model-free (raw experience)
- depth of backup
- function approximation (FA)
- on-policy vs off-policy

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s\}$$

- temporal difference (TD) learning
  - model-free
  - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
- Monte Carlo (MC)
  - model-free, full trajectory of experience
- Dynamic Programming (DP)
  - model-based, one full backup

(Richard Sutton and Andrew Barto, Reinforcement Learning: An Introduction, 2nd edition, in progress)

Deep Reinforcement Learning

- use deep neural networks to represent
  - value function
  - policy
  - model
- optimize loss function by stochastic gradient descent

(David Silver, ICML 2016 Tutorial: Deep Reinforcement Learning)

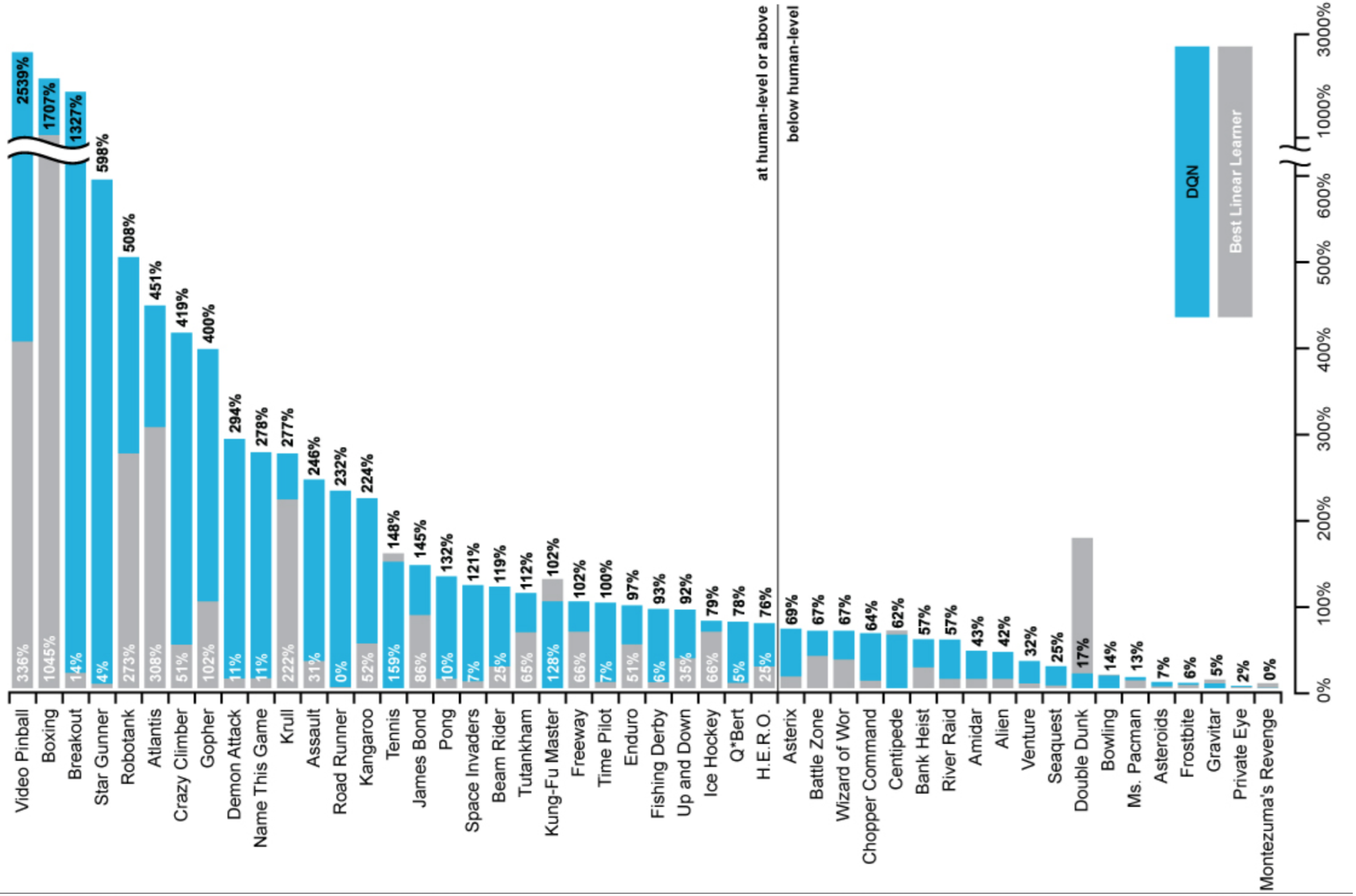
Deep Q-Network

- Q-learning converges to optimal value
  - table lookup representation
  - linear function approximation
- deep Q-networks represent Q-value function with deep neural networks, $Q(s, a; \boldsymbol{w}) \approx Q^*(s, a)$
- Q-learning diverges using neural networks
  - correlation samples
  - non-stationary optimization targets
- DQN uses experience replay to solve these issues
  - stores the agent's experiences in a replay memory
  - samples experiences uniformly to update weights

DQN in Atari games

- end-to-end learning of $Q(s, a)$ from raw 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

(David Silver, ICML 2016 Tutorial: Deep Reinforcement Learning)



a deep Q-network agent, receiving only the pixels and the game score as inputs, was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters

(Mnih, V. et al. Human-level control through deep reinforcement learning. Nature 518, 529-533 (2015).)

Deep Reinforcement Learning: AI = RL + DL

we seek a single agent which can solve any human-level task

- RL defines the objective
- DL gives the mechanism
- RL + DL = general intelligence

(David Silver, ICML 2016 Tutorial: Deep Reinforcement Learning)

Deep Reinforcement Learning is Artificial Intelligence.

DRL Applications Abound

- games
- robotics
- natural langage processing (NLP)
- computer vision
- neural architecture design
- business management
- healthcare
- finance
- Industry 4.0
- smart grid
- intelligent transportation systems
- computer systems

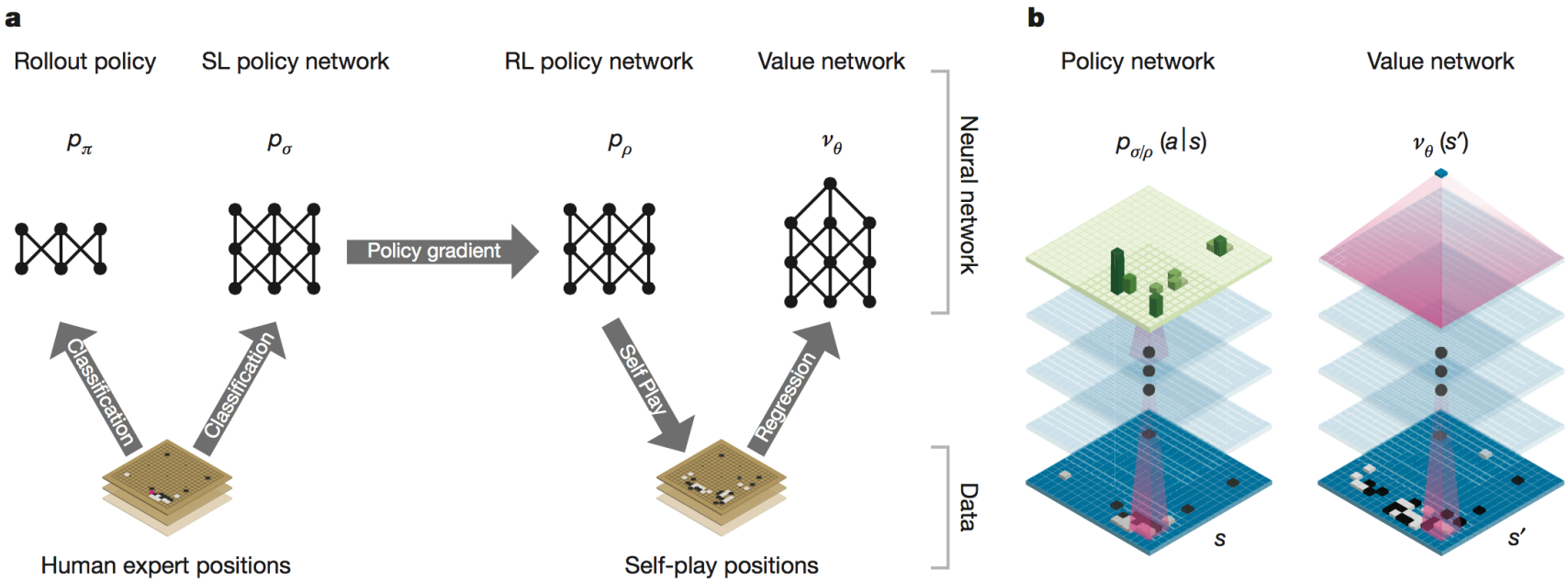


Figure 1 | Neural network training pipeline and architecture. a, A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set. **b,** Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

- why is computer Go so hard? search space - Chess: 35^{80}, Go: 250^{150}
- AlphaGo: deep learning + reinforcement learning + Monte Carlo tree search
- value network reduce depth; policy network reduce width

(David Silver et al., Mastering the game of Go with deep neural networks and tree search, Nature, January 2016.)

AlphaGo: Neural Networks Training Pipline

- supervised learning of policy networks
  - policy network: 12 layer convolutional neural network
  - training data: 30M positions from human expert games
  - training algorithm: maximize likelihood by SGD
  - training time: 4 weeks on 50 GPUs using Google Cloud

- reinforcement learning of policy networks
  - policy network: 12 layer convolutional neural network
  - training data: games of self-play between policy network
  - training algorithm: maximize wins by policy gradient RL
  - training time: 1 week on 50 GPUs using Google Cloud

- reinforcement learning of value networks
  - value network: 12 layer convolutional neural network
  - training data: 30 million games of self-play
  - training algorithm: minimize MSE by SGD
  - training time: 1 week on 50 GPUs using Google Cloud

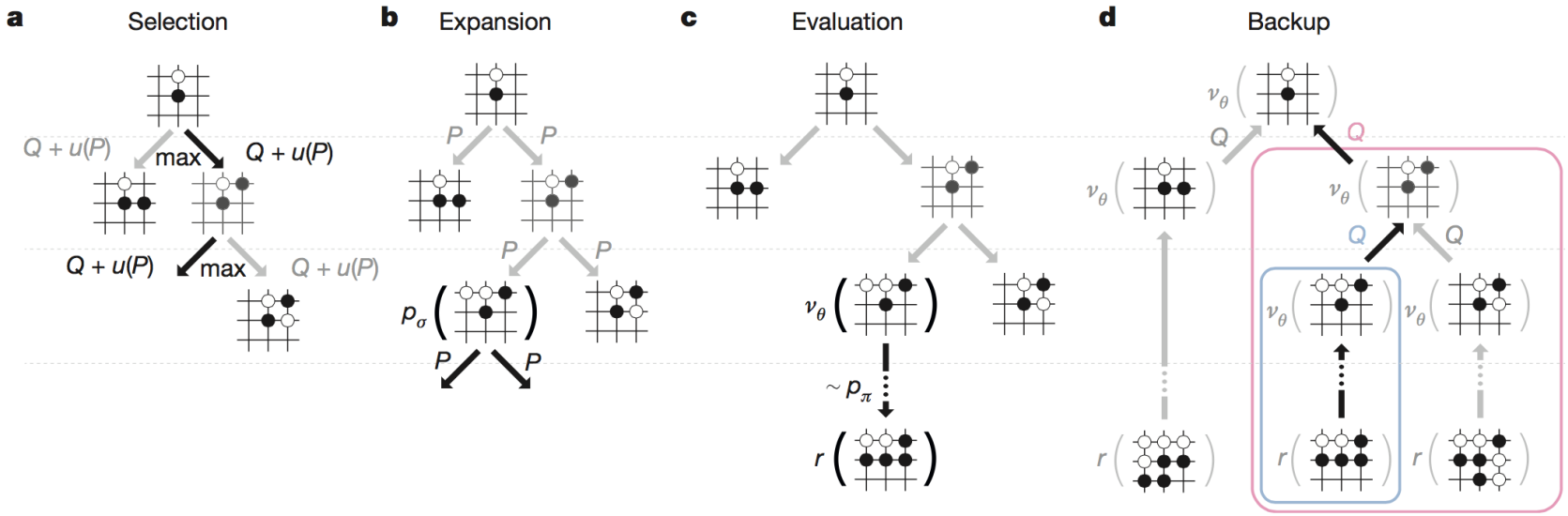
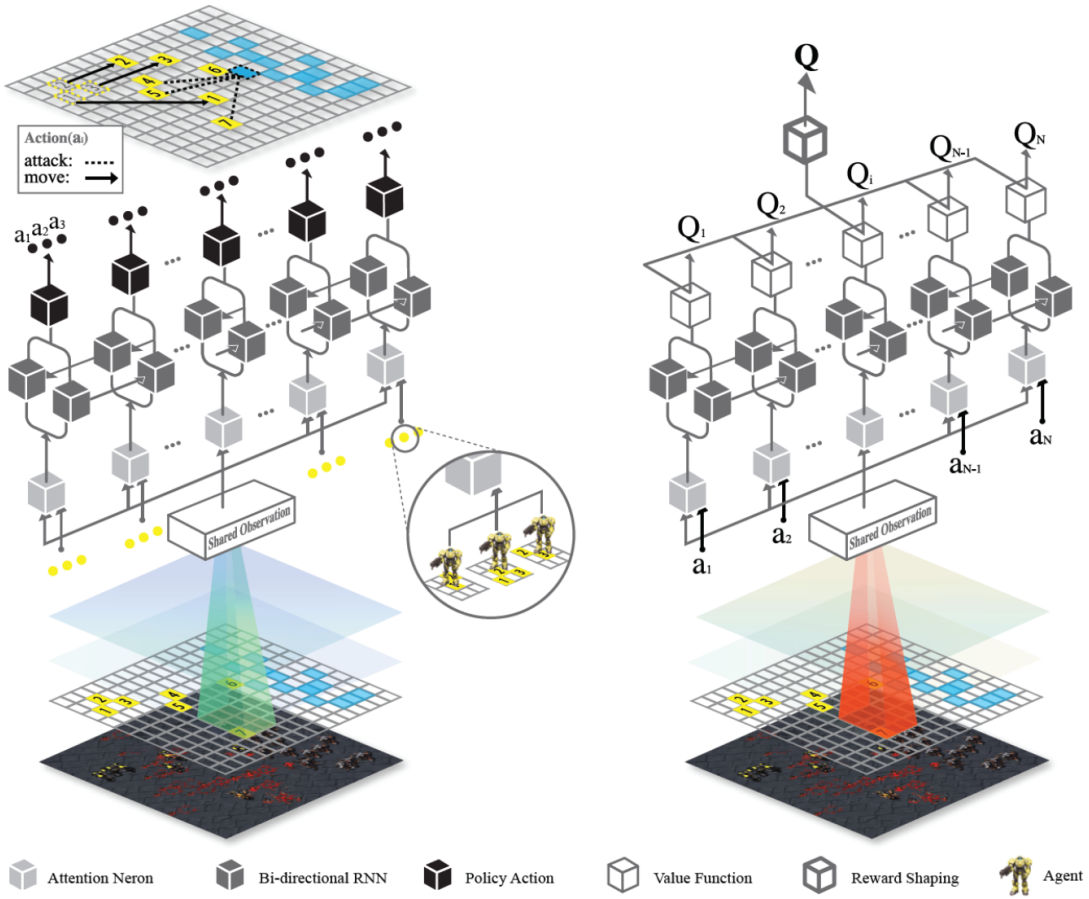


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

(Once the search is complete, the algorithm chooses the most visited move from the root position.)

(David Silver et al., Mastering the game of Go with deep neural networks and tree search, Nature, January 2016.)

StarCraft



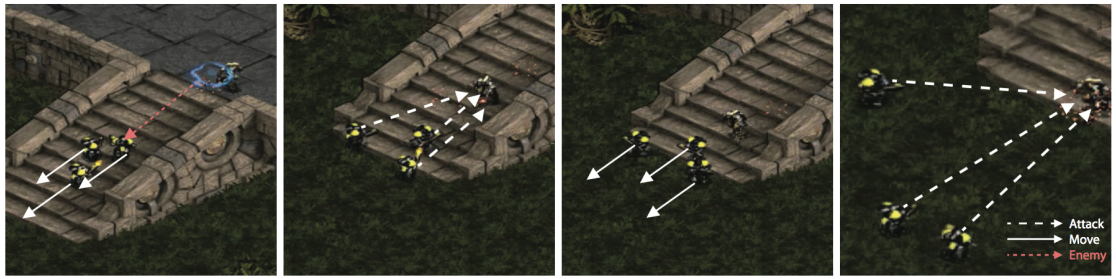
(a) Multiagent policy networks with grouping (b) Multiagent Q networks with reward shaping

Figure 1: Bidirectionally-Coordinated Net (BiCNet).

(Peng Peng et al., Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games, arXiv, June 2017.)

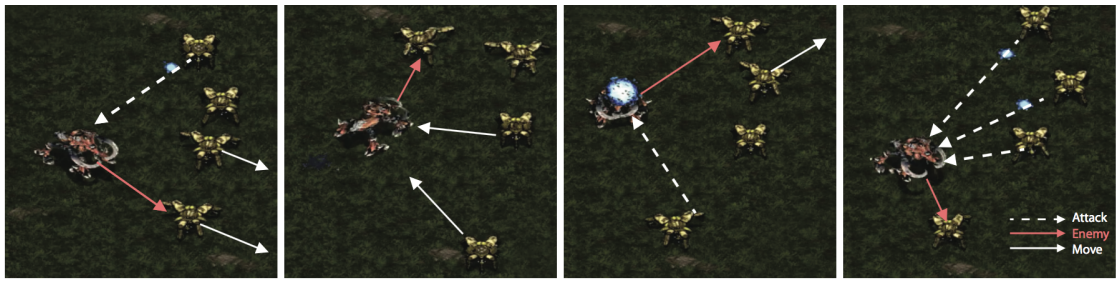
(a) Early stage of training (b) Early stage of training (c) Well-trained (d) Well-trained

Figure 6: Coordinated moves without collision in combat *3 Marines (ours) vs. 1 Super Zergling (enemy)*. The first two (a) and (b) illustrate that the collision happens when the agents are close by during the early stage of the training; the last two (c) and (d) illustrate coordinated moves over the well-trained agents.



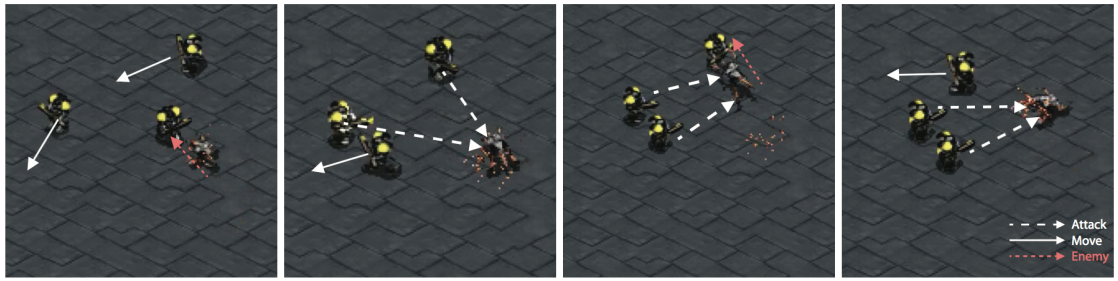
(a) time step 1: run when attacked (b) time step 2: fight back when safe (c) time step 3: run again (d) time step 4: fight back again

Figure 7: *Hit and Run* tactics in combat *3 Marines (ours) vs. 1 Zealot (enemy)*.



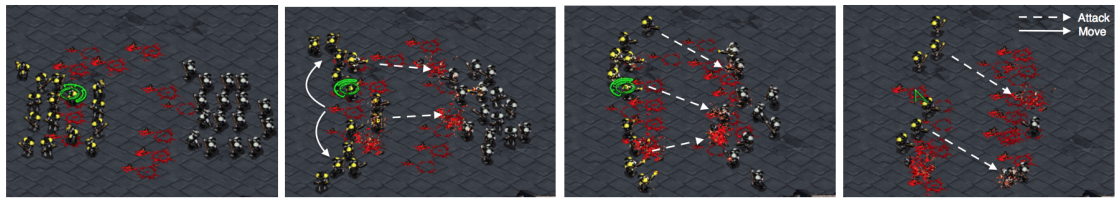
(a) time step 1 (b) time step 2 (c) time step 3 (d) time step 4

Figure 8: Coordinated cover attack in combat *4 Dragoons (ours) vs. 2 Ultralisks (enemy)*.



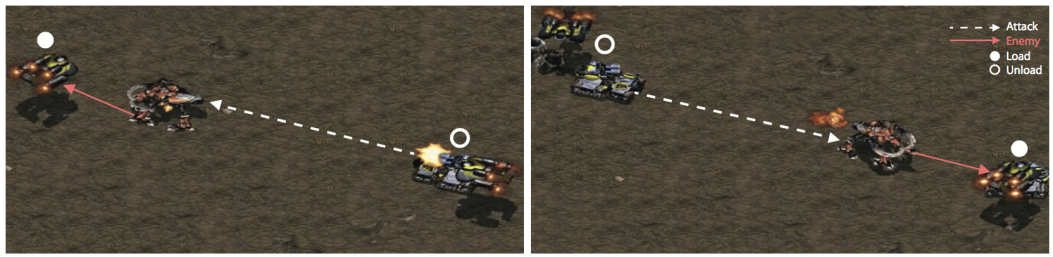
(a) time step 1 (b) time step 2 (c) time step 3 (d) time step 4

Figure 9: Coordinated cover attack in combat *3 Marines (ours) vs. 1 Zergling (enemy).*



(a) time step 1 (b) time step 2 (c) time step 3 (d) time step 4

Figure 10: "focus fire" in combat *15 Marines (ours) vs. 16 Marines (enemy).*



(a) time step 1 (b) time step 2

Figure 11: Coordinated heterogeneous agents in combat 2 *Dropships* and 2 *tanks* vs. 1 *Ultralisk.*

Dialogue Agent / Chatbot

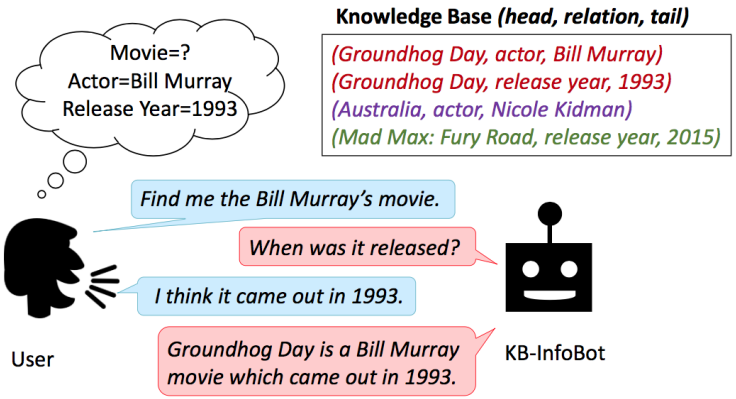


Figure 1: A dialogue example between a user looking for a movie and the KB-InfoBot. The knowledge base is shown above the KB-InfoBot.

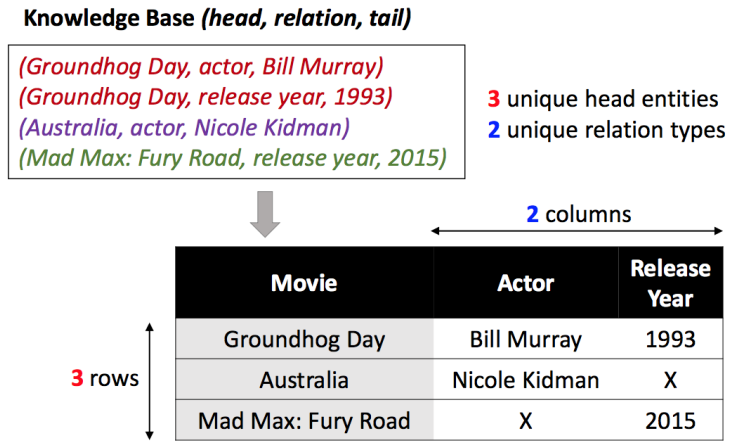


Figure 2: An entity-centric knowledge base, where head entities are movies. **Top:** Conventional (h, r, t) format. **Bottom:** Table format. Missing values are denoted by X.

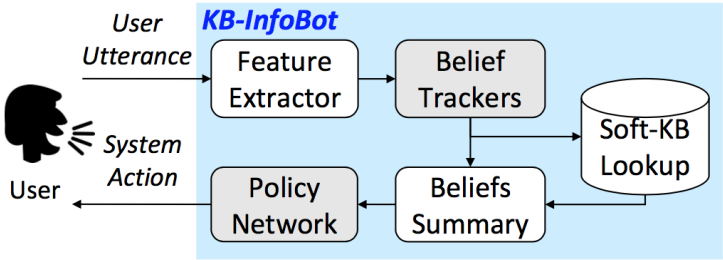


Figure 3: High-level overview of the end-to-end KB-InfoBot. Components with trainable parameters are highlighted in gray.

Bhuwan Dhingra, et al., End-to-End Reinforcement Learning of Dialogue Agents for Information Access, ACL, 2017.

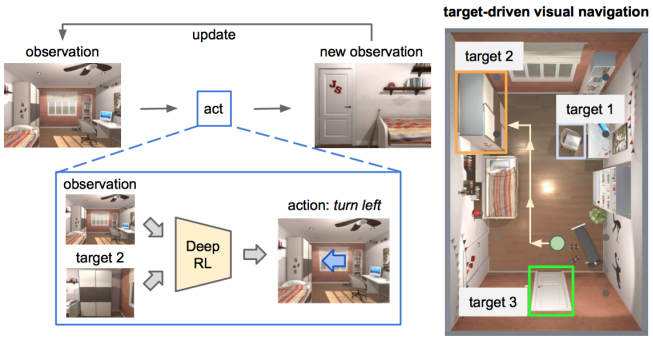


Fig. 1. The goal of our deep reinforcement learning model is to navigate towards a visual target with a minimum number of steps. Our model takes the current observation and the image of the target as input and generates an action in the 3D environment as the output. Our model learns to navigate to different targets in a scene without re-training.

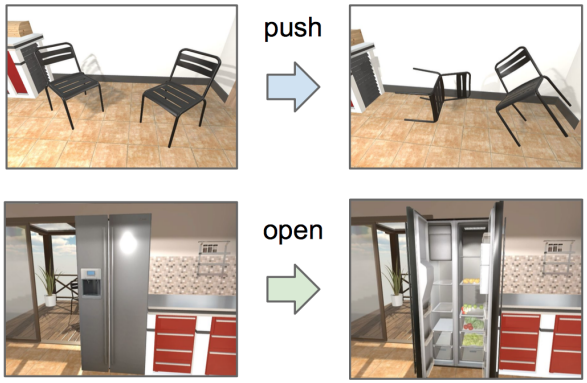


Fig. 3. Our framework provides a rich interaction platform for AI agents. It enables physical interactions, such as pushing or moving objects (the first row), as well as object interactions, such as changing the state of objects (the second row).

Fig. 2. Screenshots of our framework and other simulated learning frameworks: ALE [36], ViZDoom [37], UETorch [38], Project Malmo [39], SceneNet [40], TORCS [41], SYNTHIA [42], Virtual KITTI [43].

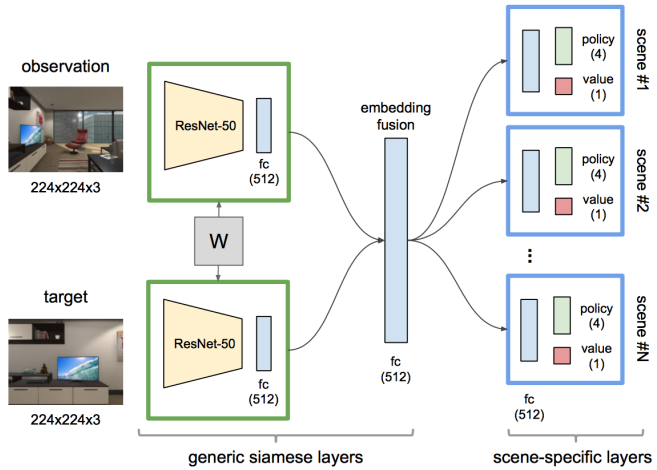
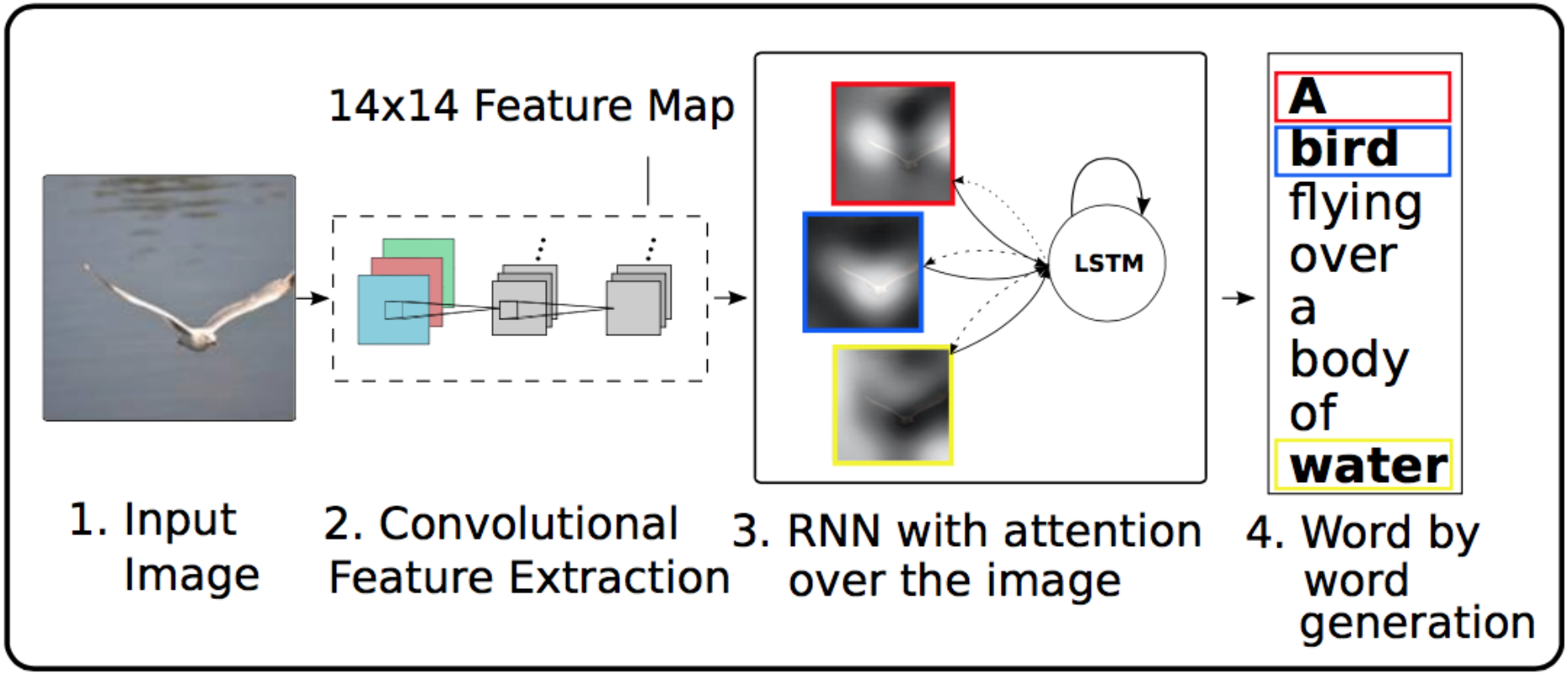


Fig. 4. Network architecture of our deep siamese actor-critic model. The numbers in parentheses show the output dimensions. Layer parameters in the green squares are shared. The ResNet-50 layers (yellow) are pre-trained on ImageNet and fixed during training.

(Yuke Zhu, et al., Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning, arXiv, 2016.)

Image Captioning



(Kevin Xu et al., Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015)

Neural Architecture Design

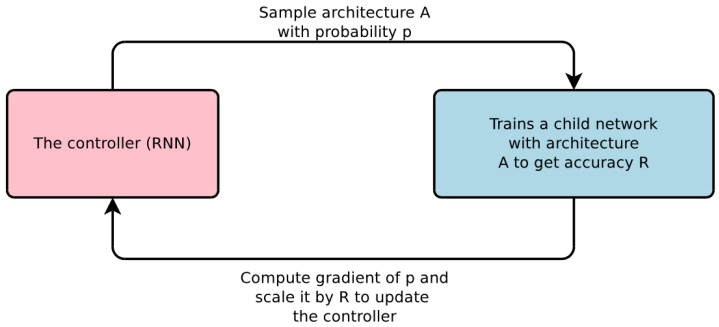
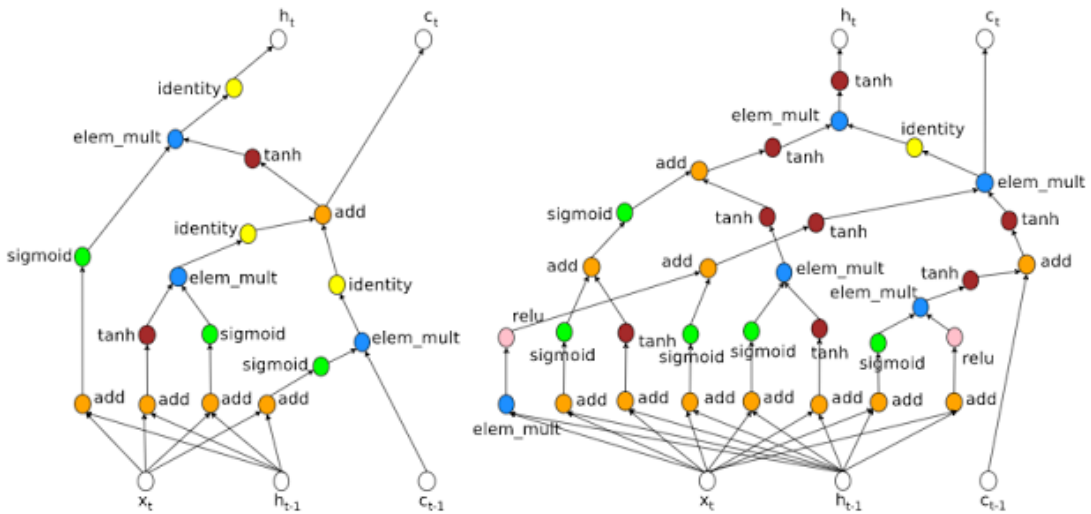


Figure 1: An overview of Neural Architecture Search.



(Barret Zoph and Quoc V. Le, Neural Architecture Search with Reinforcement Learning, ICLR, 2017.)

(https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html)

Ads / Business Management

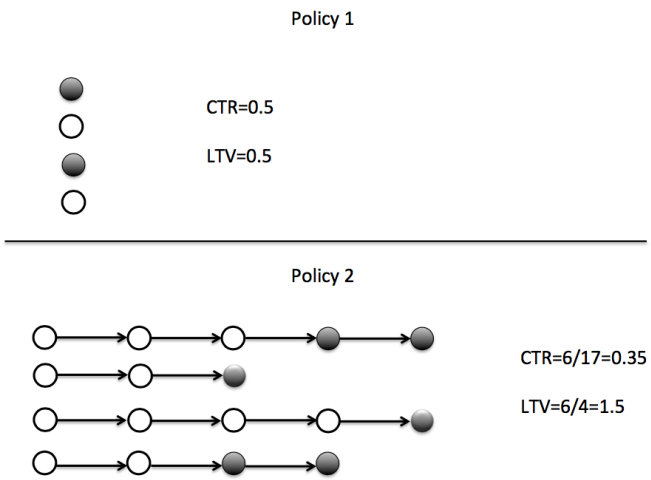


Figure 1: The circles indicate user visits. The black circles indicate clicks. Policy 1 is greedy and users do not return. Policy 2 optimizes for the long-run, users come back multiple times, and click towards the end. Even though Policy 2 has a lower CTR than Policy 1, it results in more revenue, as captured by the higher LTV. Hence, LTV is potentially a better metric than CTR for evaluating ad recommendation policies.

$$\text{CTR} = \frac{\text{Total \# of Clicks}}{\text{Total \# of } \mathbf{Visits}} \times 100,$$

$$\text{LTV} = \frac{\text{Total \# of Clicks}}{\text{Total \# of } \mathbf{Visitors}} \times 100.$$

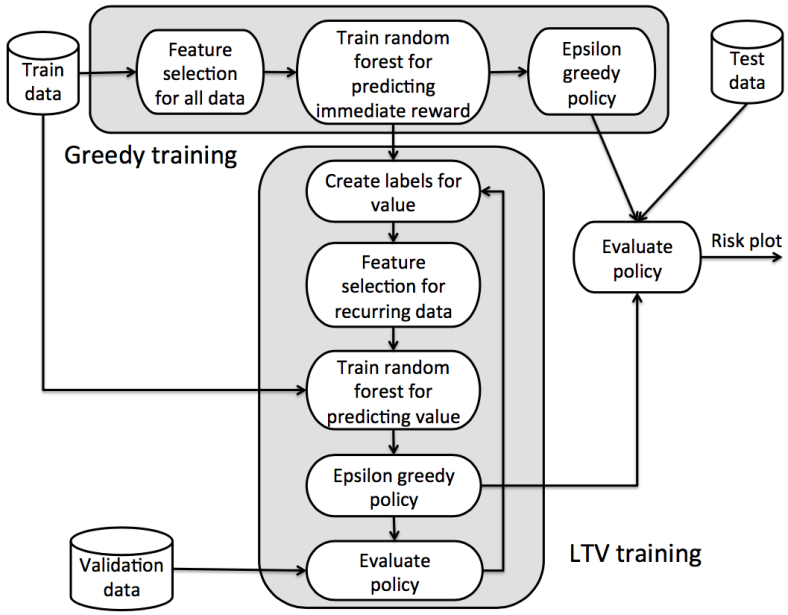


Figure 2: This figure shows the flow for training Greedy and LTV strategies.

(Georgios Theocharous et al., Personalized Ad Recommendation Systems for Life-Time Value Optimization with Guarantees, IJCAI, 2015)

Option Pricing / Finance

- a call option gives the holder the right, not the obligation, to buy the underlying asset, e.g., a share of a stock, by a certain date (the maturity date), T, for a certain price (the strike price), K
  - a put option gives the right to sell
- the payoff function: $\max(0, S_t - K)$. S_t : price at time t
- an American option can be exercised any time before the maturity date
  - an exercise policy determines when to exercise it
- there is a fee to hold an option contract, pricing

- reinforcement learning setup
  - state: underlying asset price, e.g., stock price

    other relevant economic/financial variables, like interest rate
  - action: exercise or continue to hold
  - reward: payoff function
  - goal: option pricing/find optimal exercise policy

(Francis Longstaff and Eduardo Schwartz, Valuing American options by simulation: a simple least-squares approach, Review of Financial Studies, 14(1):113-47, 2001.)
(John N. Tsitsiklis and Benjamin Van Roy, Regression Methods for Pricing Complex American-Style Options, IEEE Transactions on Neural Networks, 12(4) (special issue on computational finance), 694-703, July 2001.)

Dynamic Treatment Regime / Healthcare

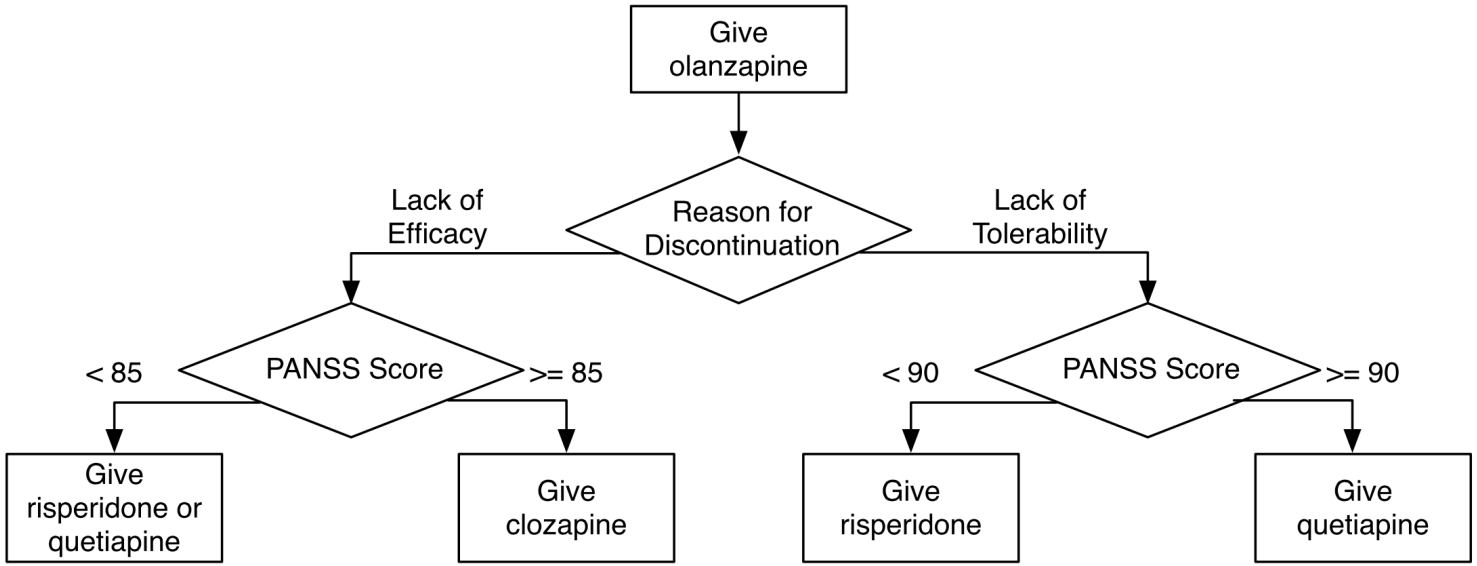


Fig. 3 Optimal treatment policy learned from 25 imputations of the CATIE data, with the total reward defined as the negative area under the PANSS curve for the 18 months of the CATIE study. The state representation is defined in Sect. 3.1 and the Q-function form used is described in Sect. 4.2

The Clinical Antipsychotic Trials of Intervention Effectiveness (CATIE) was an 18 month multistage clinical trial of 1460 patients with schizophrenia.

The Positive and Negative Syndrome Scale (PANSS) score is a medical scale designed to measure symptom severity in patients with schizophrenia. The higher PANSS score, the more psychotic symptoms.

Olanzapine, risperidone, quetiapine, clozapine are antipsychotic medication.

(Susan M. Shortreed et al., Informing sequential clinical decision-making through reinforcement learning: an empirical study , Machine Learning, 2011.)

Idustry 4.0

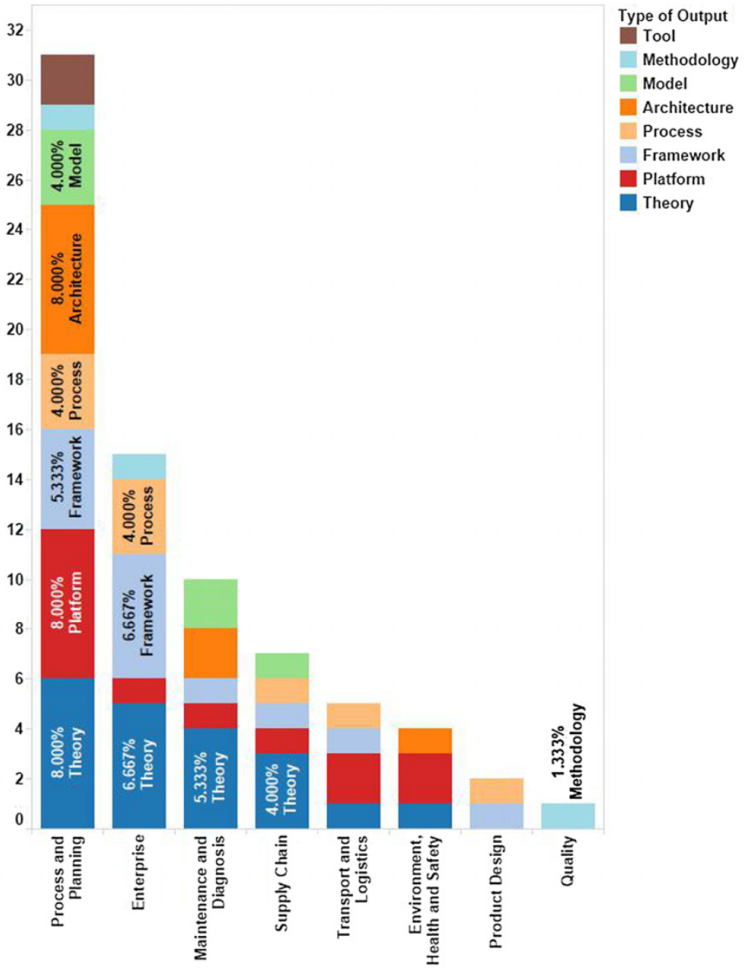


Fig. 13 Areas of manufacturing with significant research contributions

(Peter O'Donovan et al., Big data in manufacturing: a systematic mapping study, Journal of Big Data, 2015.)

Smart Grid

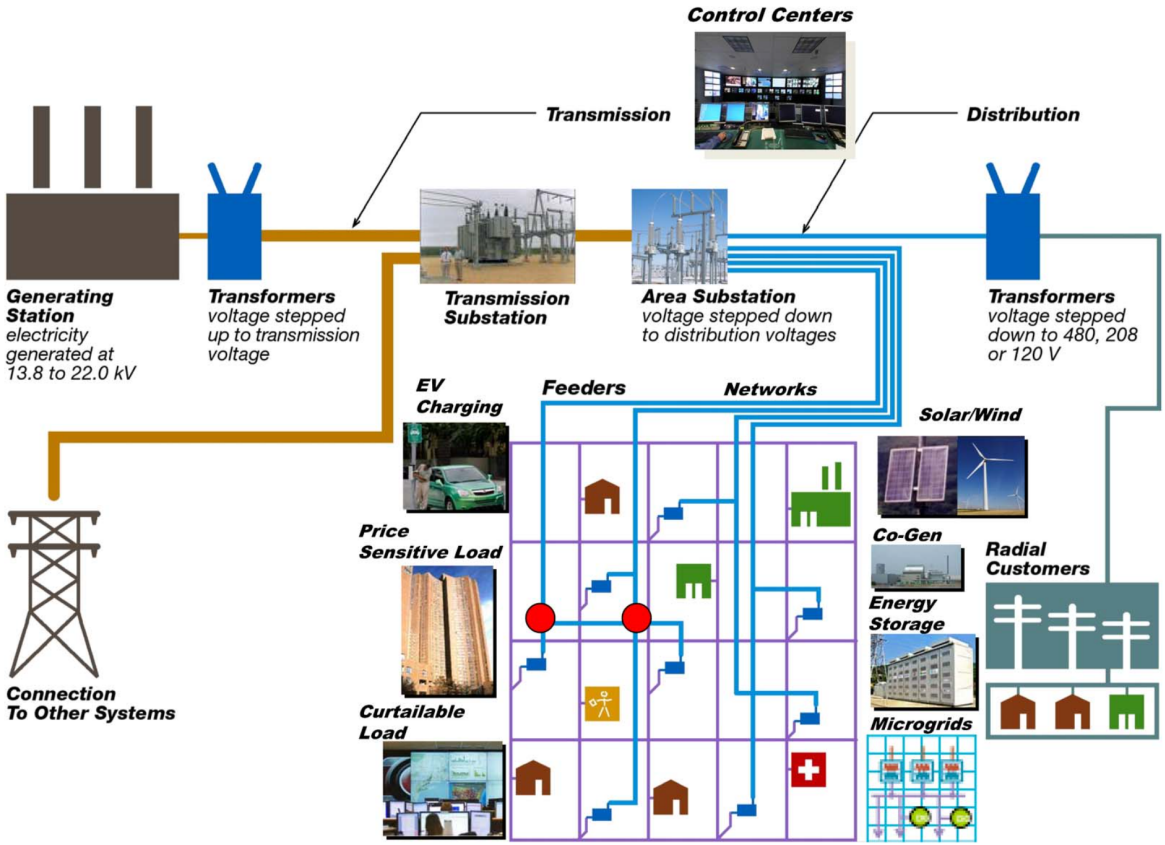


Fig. 1. ***The adaptive stochastic control (ASC) of the Smart Grid must simultaneously optimize supply and demand from many new, distributed loads and sources such as the following: price sensitive and curtailable loads, intermittent solar and wind generation, distributed energy storage, EV charging, and microgrids. (Source: Modified after Con Edison drawing).***

requirements: self-healing, flexible, predictive, interactive, optimal, secure

(Roger N. Anderson et al., Adaptive Stochastic Control for the Smart Grid, Proceedings of the IEEE, June 2011.)

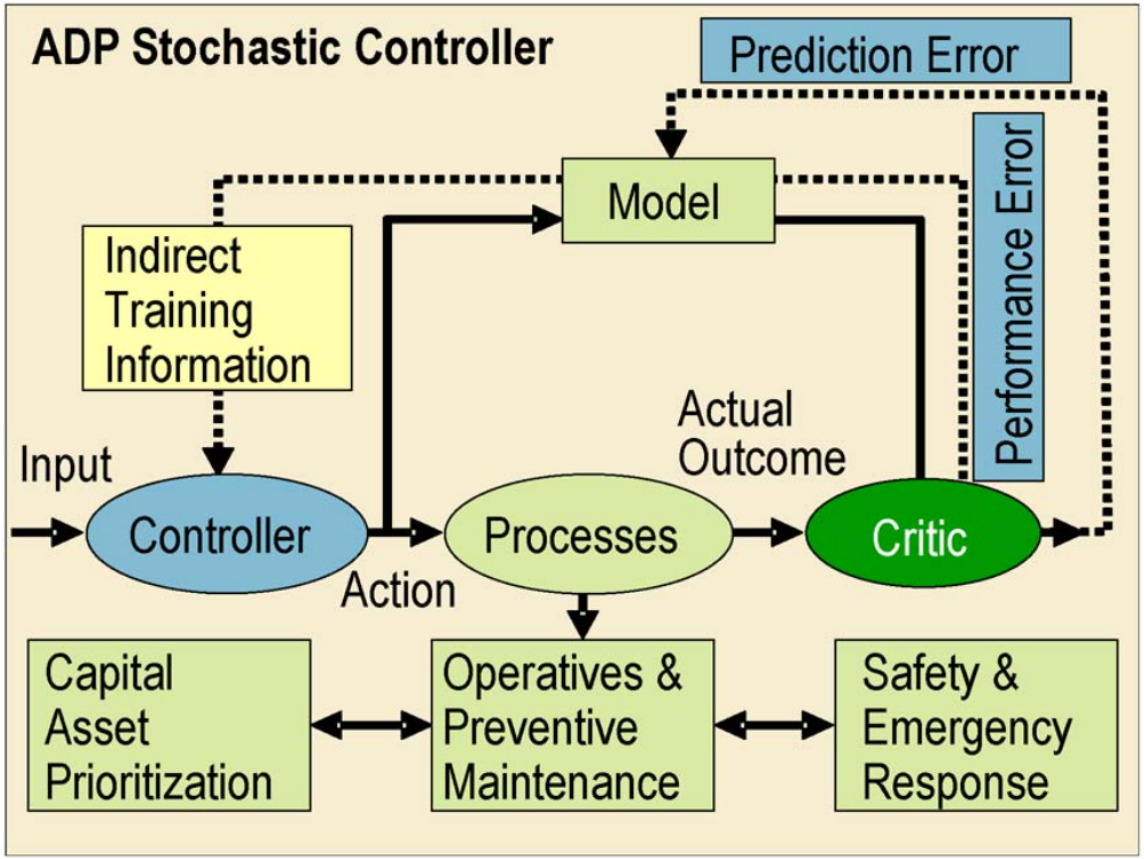


Fig. 3. ***The feedback loops for an Adaptive Stochastic Controller for the Smart Grid optimally interprets incoming data from many new distributed sources and simultaneously manages asset prioritization, operational actions, maintenance tasks, and emergency responses.***

(Roger N. Anderson et al., Adaptive Stochastic Control for the Smart Grid, Proceedings of the IEEE, June 2011.)

Adaptive Traffic Signal Control/Intelligent Traffic Systems

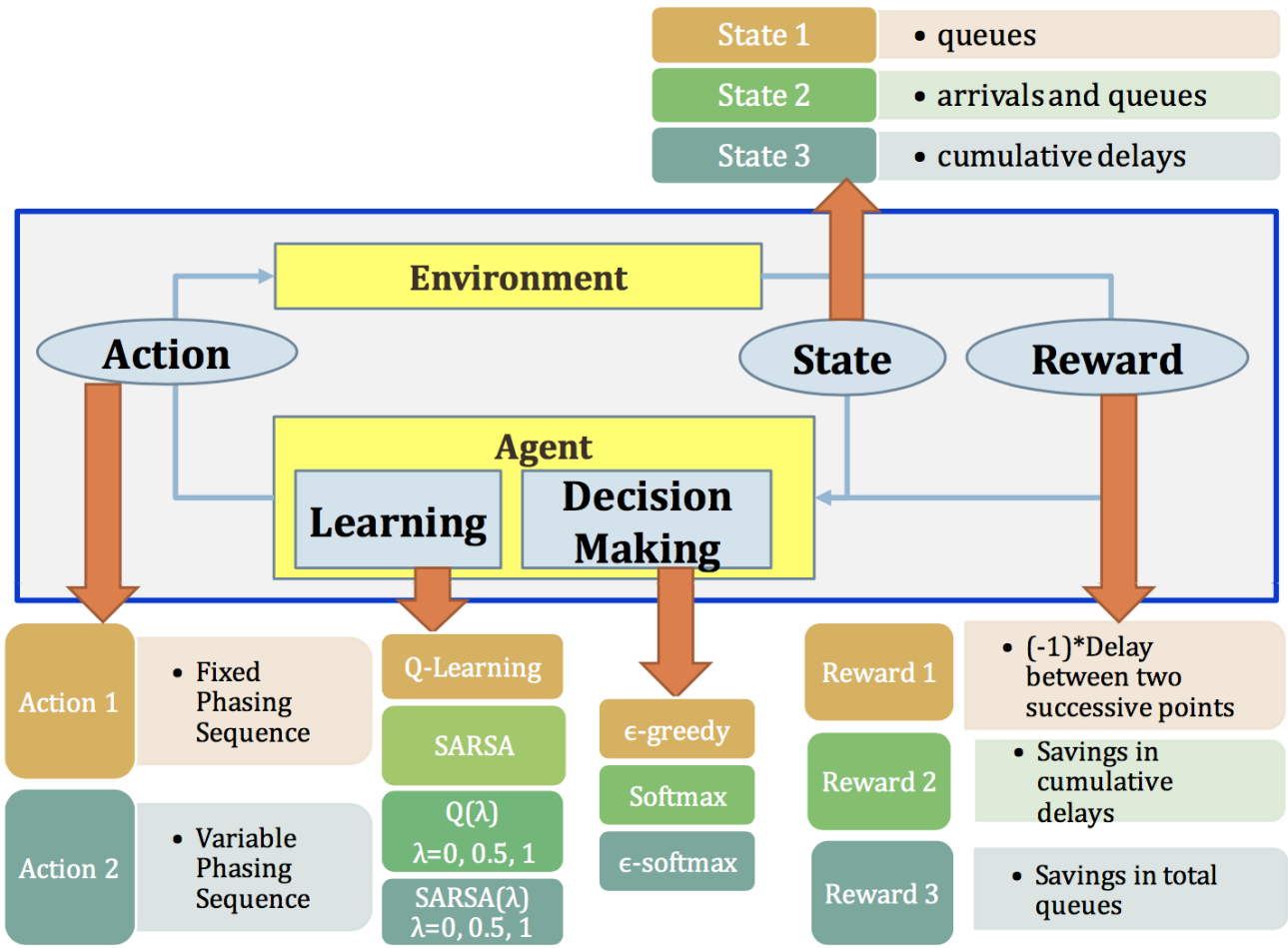
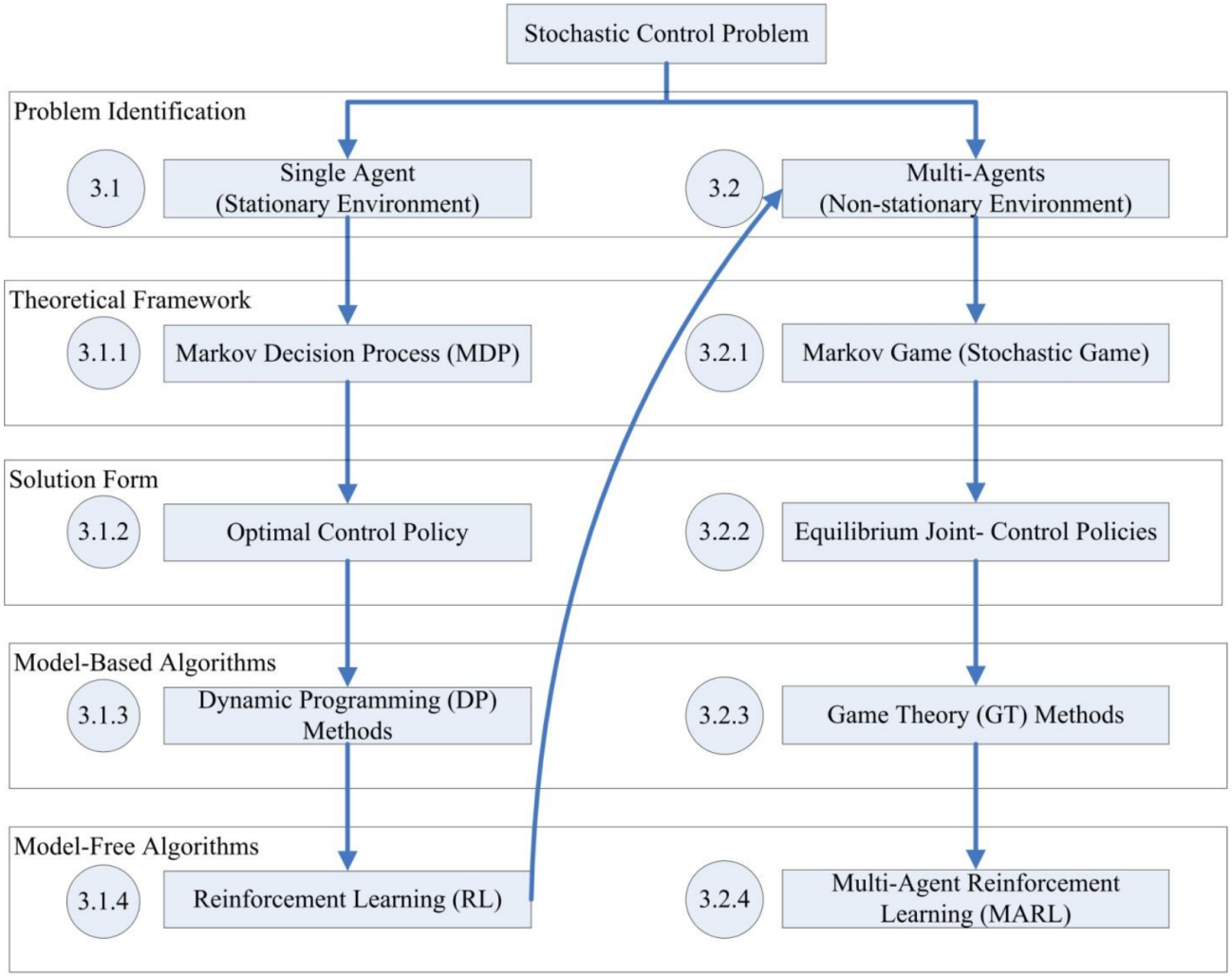


Figure 5-6 RL Architecture and Design Parameters

(Samah El-Tantawy, Multi-Agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers , PhD thesis, 2012.)

Adaptive Traffic Signal Control (background knowledge)



(Samah El-Tantawy, Multi-Agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers , PhD thesis, 2012.)

Device Placement Optimization / Computer Systems

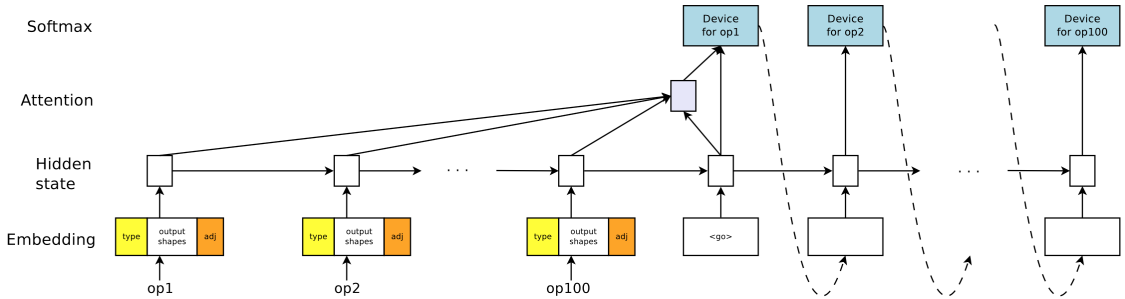


Figure 2. Device placement model architecture.

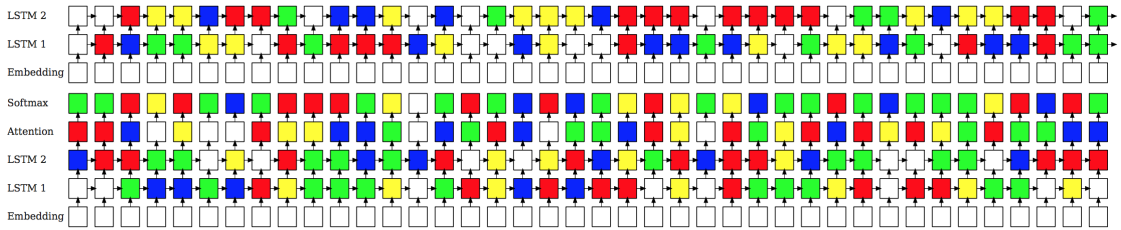


Figure 4. RL-based placement of Neural MT graph. Top: encoder, Bottom: decoder. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. This placement achieves an improvement of 19.3% in running time compared to the fine-tuned expert-designed placement.

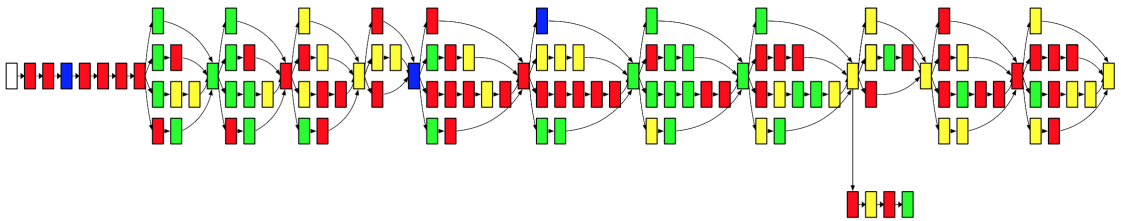


Figure 5. RL-based placement of Inception-V3. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. RL-based placement achieves the improvement of 19.7% in running time compared to expert-designed placement.

(Azalia Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, ICML, 2017.)

Differentiable Neural Computer

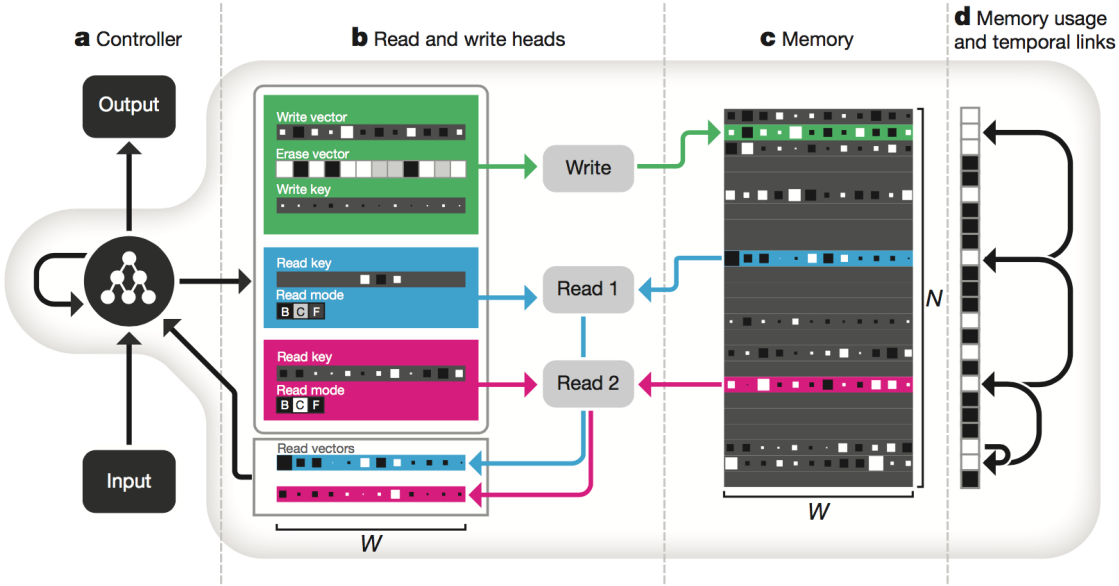


Figure 1 | DNC architecture. a, A recurrent controller network receives input from an external data source and produces output. **b**, **c**, The controller also outputs vectors that parameterize one write head (green) and multiple read heads (two in this case, blue and pink). (A reduced selection of parameters is shown.) The write head defines a write and an erase vector that are used to edit the $N \times W$ memory matrix, whose elements' magnitudes and signs are indicated by box area and shading, respectively. Additionally, a write key is used for content lookup to find previously written locations to edit. The write key can contribute to defining a weighting that selectively focuses the write operation over the rows, or locations, in the memory matrix. The read heads can use gates called read modes to switch between content lookup using a read key ('C') and reading out locations either forwards ('F') or backwards ('B') in the order they were written. **d**, The usage vector records which locations have been used so far, and a temporal link matrix records the order in which locations were written; here, we represent the order locations were written to using directed arrows.

a neural network can read from and write to an external memory

learn end-to-end with gradient descent

solve complex, structured problems, like question answering, shortest path finding in transportation networks, relationship inference in a family tree, and a moving blocks puzzle with changing goals specified by symbol sequences

(Alex Graves et. al., Hybrid computing using a neural network with dynamic external memory. Nature, 538:471-476, 2016.)

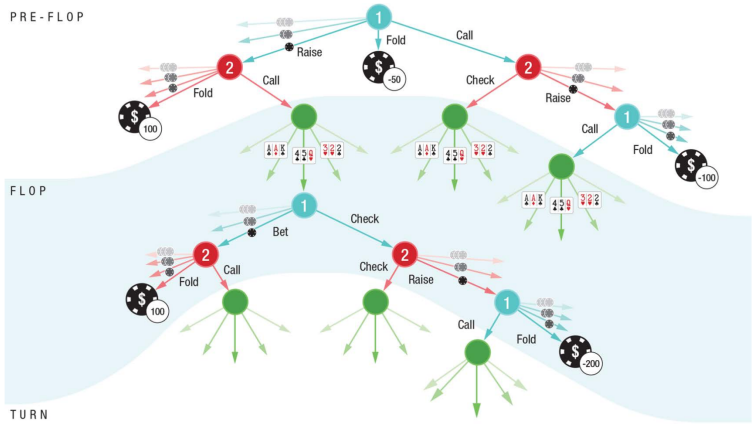


Fig. 1. A portion of the public tree in HUNL. Nodes represent public states, whereas edges represent actions: red and turquoise showing player betting actions, and green representing public cards revealed by chance. The game ends at terminal nodes, shown as a chip with an associated value. For terminal nodes where no player folded the value is returned by a function of the players' joint private information.

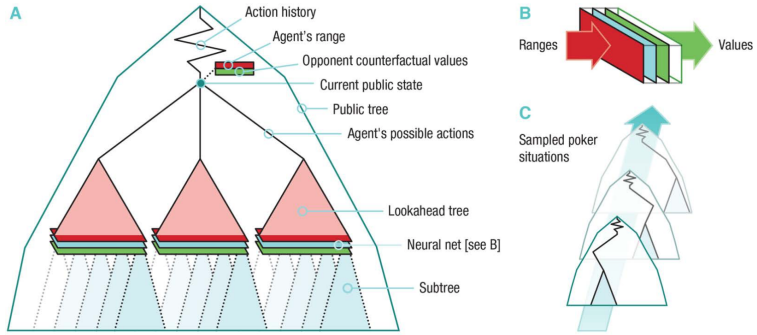


Fig. 2. DeepStack overview. (A) DeepStack reasons in the public tree always producing action probabilities for all cards it can hold in a public state. It maintains two vectors while it plays: its own range and its opponent's counterfactual values. As the game proceeds, its own range is updated via Bayes' rule using its computed action probabilities after it takes an action. Opponent counterfactual values are updated as discussed under "Continual re-solving". To compute action probabilities when it must act, it performs a re-solve using its range and the opponent counterfactual values. To make the re-solve tractable it restricts the available actions of the players and lookahead is limited to the end of the round. During the re-solve, counterfactual values for public states beyond its lookahead are approximated using DeepStack's learned evaluation function. (B) The evaluation function is represented with a neural network that takes the public state and ranges from the current iteration as input and outputs counterfactual values for both players (Fig. 3). (C) The neural network is trained prior to play by generating random poker situations (pot size, board cards, and ranges) and solving them to produce training examples. Complete pseudocode can be found in algorithm S1 (*10*).

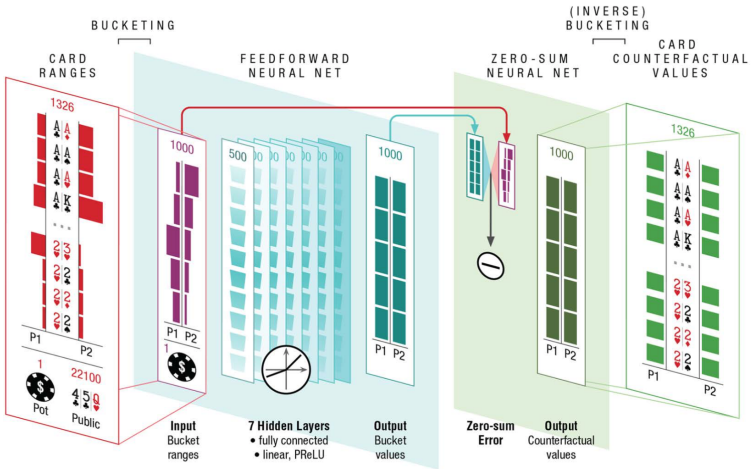


Fig. 3. Deep counterfactual value network. The inputs to the network are the pot size, public cards, and the player ranges, which are first processed into hand clusters. The output from the seven fully connected hidden layers is postprocessed to guarantee the values satisfy the zero-sum constraint, and then mapped back into a vector of counterfactual values.

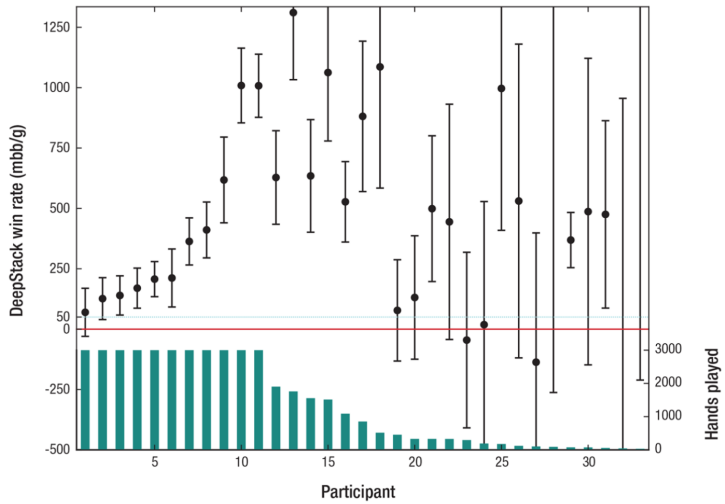


Fig. 4. Performance of professional poker players against DeepStack. Performance estimated with AIVAT along with a 95% confidence interval. The solid bars at the bottom show the number of games the participant completed.

a solution to a family of imperfect information games, cf. CMU Libratuss

(M. Moravčík et al., DeepStack: Expert-level artificial intelligence in heads-up no-limit poker, Science, 2017)

Learning with Small Data

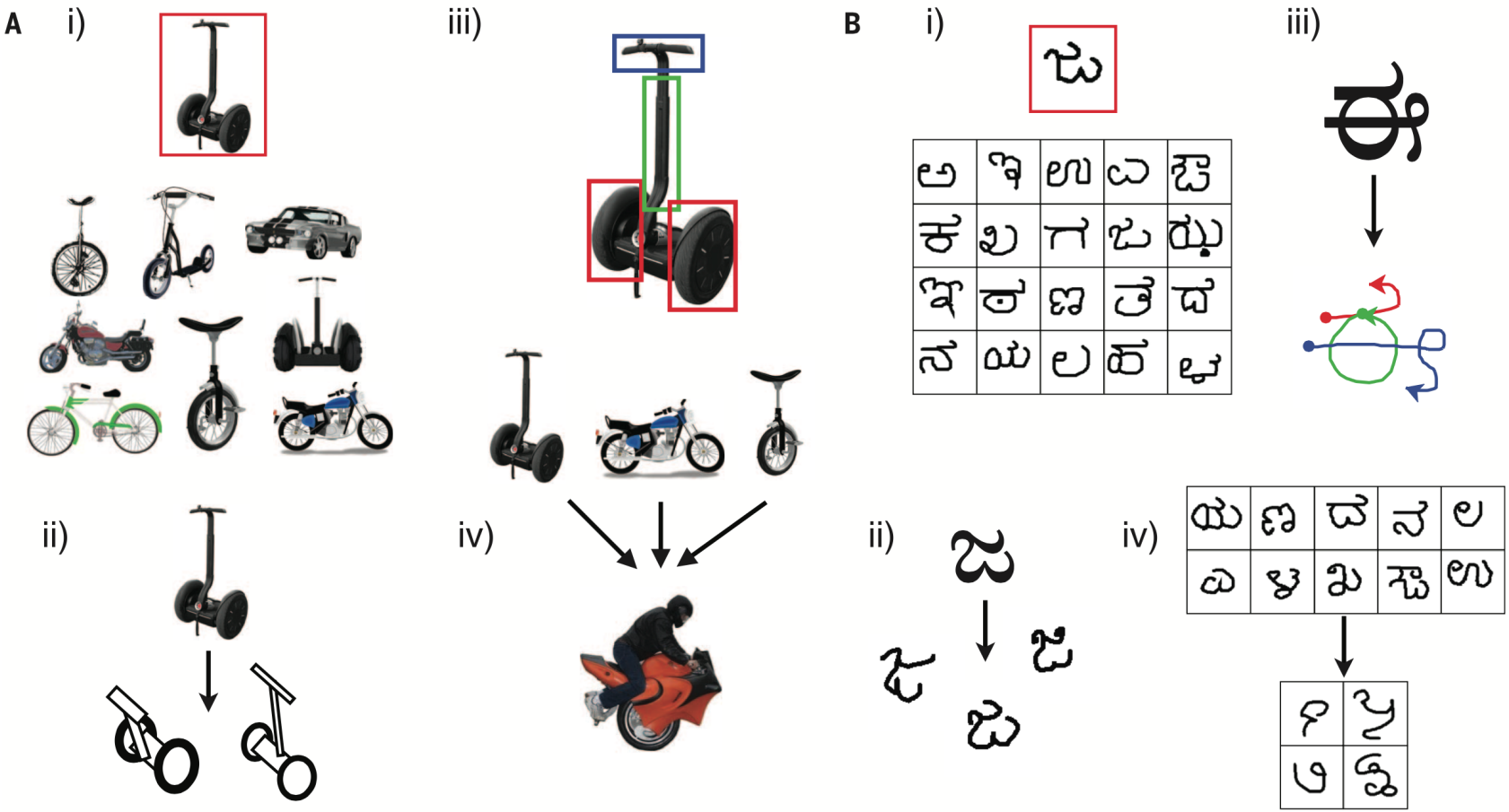


Fig. 1. People can learn rich concepts from limited data. (**A** and **B**) A single example of a new concept (red boxes) can be enough information to support the (i) classification of new examples, (ii) generation of new examples, (iii) parsing an object into parts and relations (parts segmented by color), and (iv) generation of new concepts from related concepts. [Image credit for (A), iv, bottom: With permission from Glenn Roberts and Motorcycle Mojo Magazine]

(Brenden M. Lake, Ruslan Salakhutdinov, Joshua B. Tenenbaum, Human-level concept learning through probabilistic program induction, Science, December 2015)
(Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, Samuel J. Gershman, Building Machines That Learn and Think Like People, 2016)

Generative Adversarial Nets

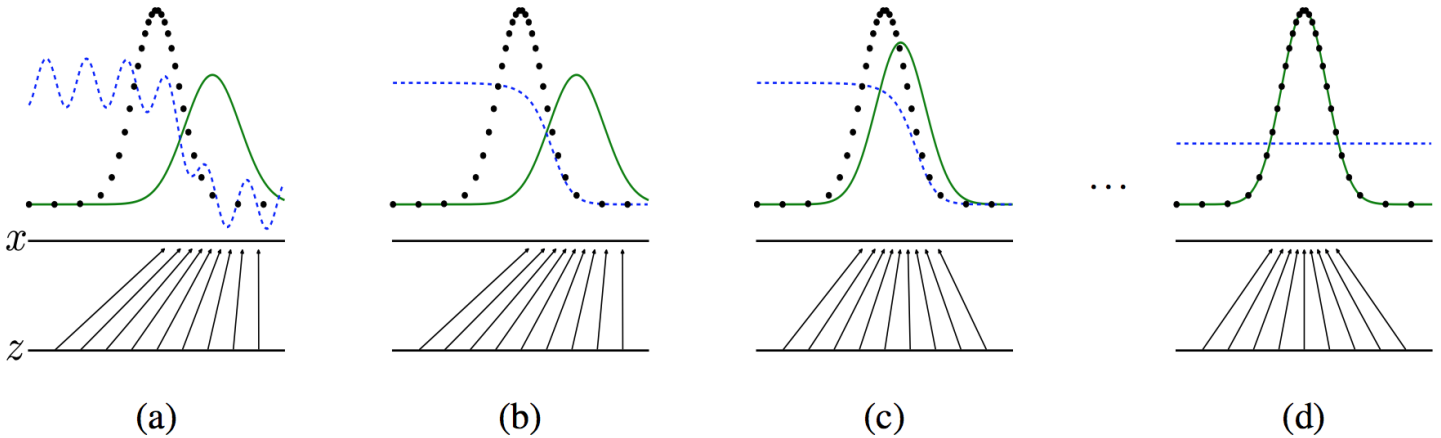


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

(Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, Generative Adversarial Nets, NIPS 2014)
(Ian J. Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv 2017.)
(Martin Arjovsky, Soumith Chintala, and Leon Bottou, Wasserstein GAN, ArXiv e-prints, 2017)
(David Pfau, Oriol Vinyals, Connecting Generative Adversarial Networks and Actor-Critic Methods, 2016)

Deep Reinforcement Learning: An Overview

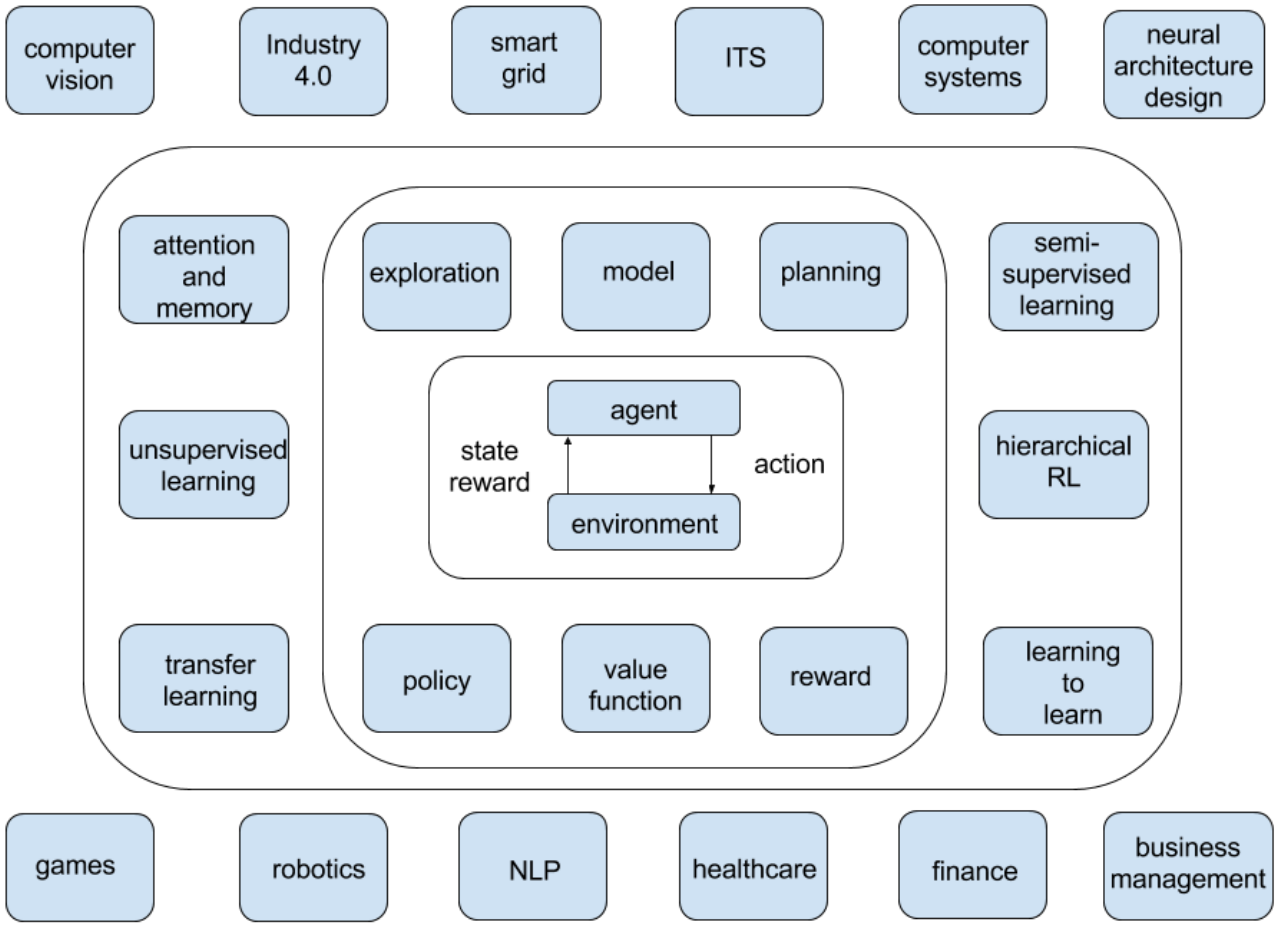
Yuxi Li (yuxili@gmail.com)

Abstract

We give an overview of recent exciting achievements of deep reinforcement learning (RL). We discuss six core elements, six important mechanisms, and twelve applications. We start with background of machine learning, deep learning and reinforcement learning. Next we discuss core RL elements, including value function, in particular, Deep Q-Network (DQN), policy, reward, model, planning, and exploration. After that, we discuss important mechanisms for RL, including attention and memory, in particular, differentiable neural computer (DNC), unsupervised learning, transfer learning, semi-supervised learning, hierarchical RL, and learning to learn. Then we discuss various applications of RL, including games, in particular, AlphaGo, robotics, natural language processing, including dialogue systems (a.k.a. chatbots), machine translation, and text generation, computer vision, neural architecture design, business management, finance, healthcare, Industry 4.0, smart grid, intelligent transportation systems, and computer systems. We mention topics not reviewed yet. After listing a collection of RL resources, we present a brief summary, and close with discussions.

MIT Technology Review, 10 Breakthrough Technologies

- Deep learning, 2013
  - With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
- Reinforcement learning, 2017
  - By experimenting, computers are figuring out how to do things that no programmer could teach them.



Yuxi Li, Deep reinforcement learning: an overview, arXiv, 2017

AI Ecosystem in Canada ... Where are we?