REINFORCEMENT LEARNING
(part 3)

Nguyen Do Van, PhD

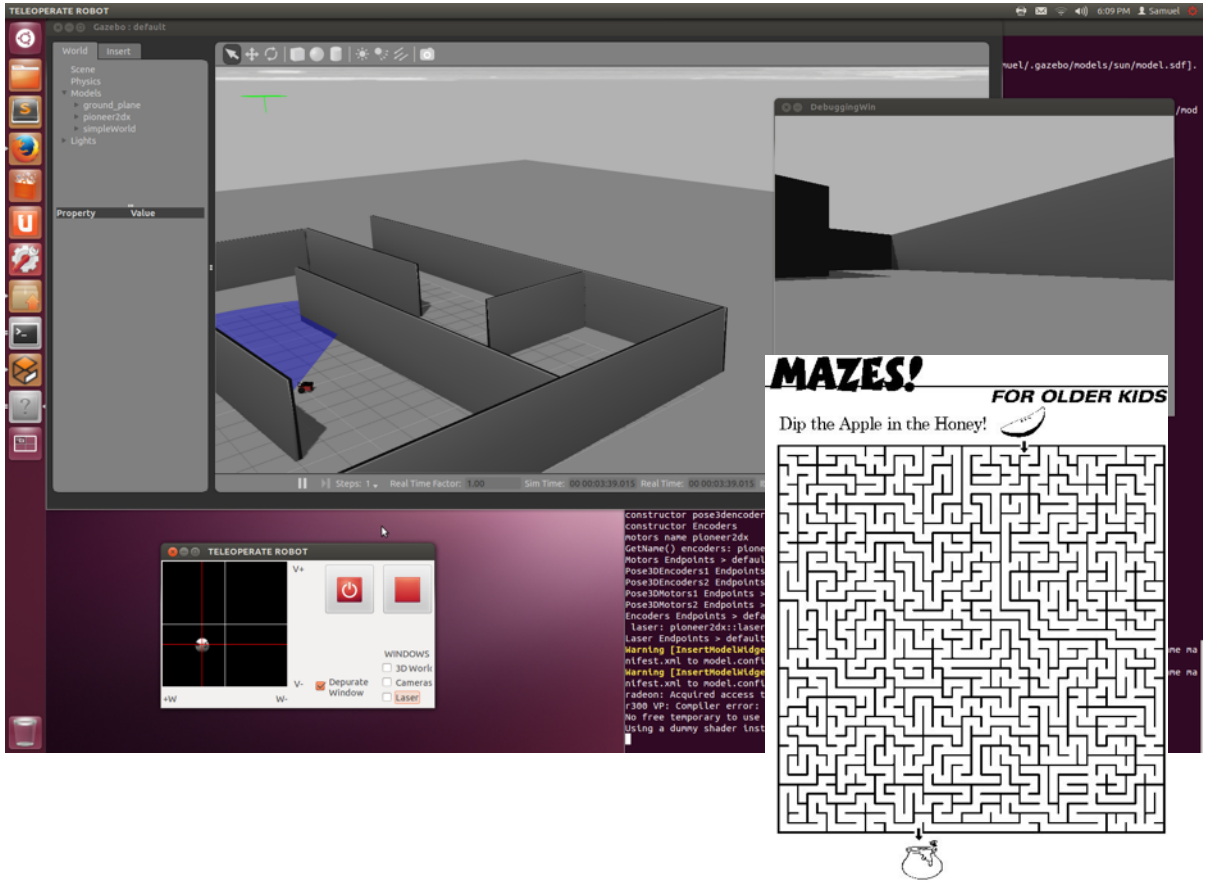
VIASM
Vietnam Institute for
Advanced Study in Mathematics

Part of the Lecture

- Challenge Discussion
- Exploration and Exploitation
- Deep Q-Learning
- Policy Gradient with Actor-critic
- Alpha-Go
- Q&A on Assignment 2 and Final Examination

Discussion on challenges

- Designing the problem domain
  - State representation
  - Action choice
  - Cost/reward signal
- Acquiring data for training
  - Exploration / exploitation
  - High cost actions
  - Time-delayed cost/reward signal
- Function approximation
- Validation / confidence measures

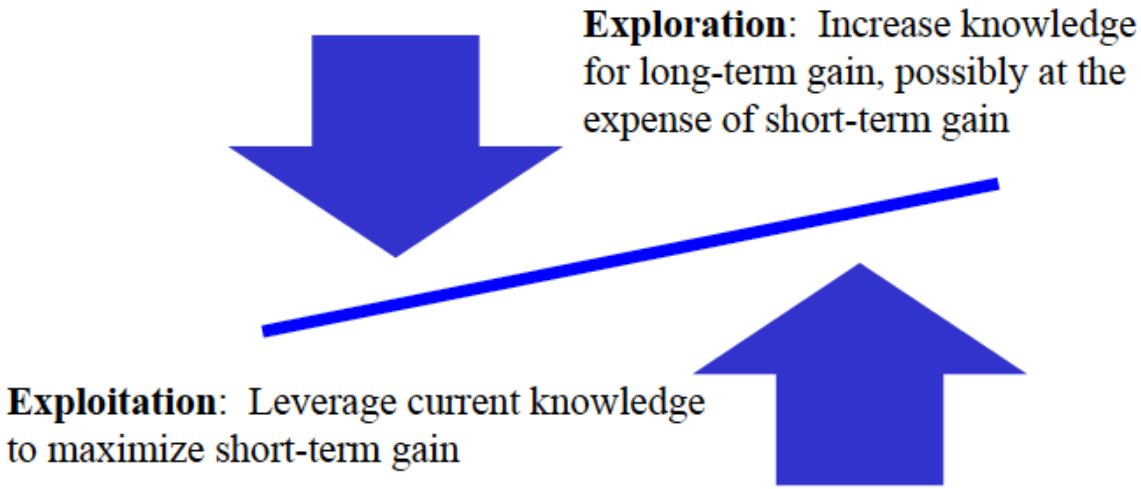


Vietnam Institute for Advanced Study in Mathematics

EXPLORATION AND EXPLOITATION

Exploration vs Exploitation

- Online decision-making
  - Exploitation Make the best decision given current information
  - Exploration Gather more information
- The best long-term strategy may involve short-term sacrifices
- Gather enough information to make the best overall decisions



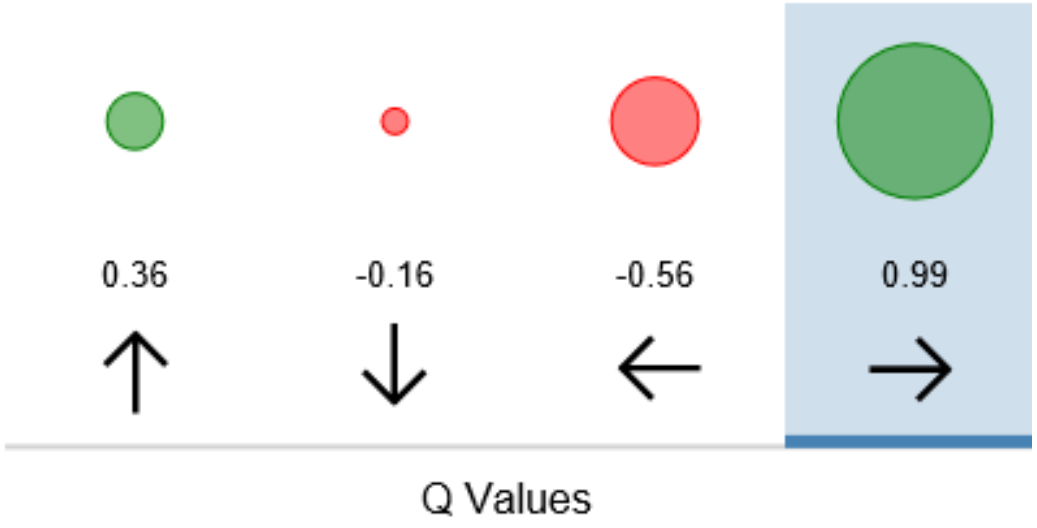
Trade-off between Exploration and Exploitation

Examples

- Restaurant Selection
  - Exploitation Go to your favourite restaurant
  - Exploration Try a new restaurant
- Online Banner Advertisements
  - Exploitation Show the most successful advert
  - Exploration Show a different advert
- Oil Drilling
  - Exploitation Drill at the best known location
  - Exploration Drill at a new location
- Game Playing
  - Exploitation Play the move you believe is best
  - Exploration Play an experimental move

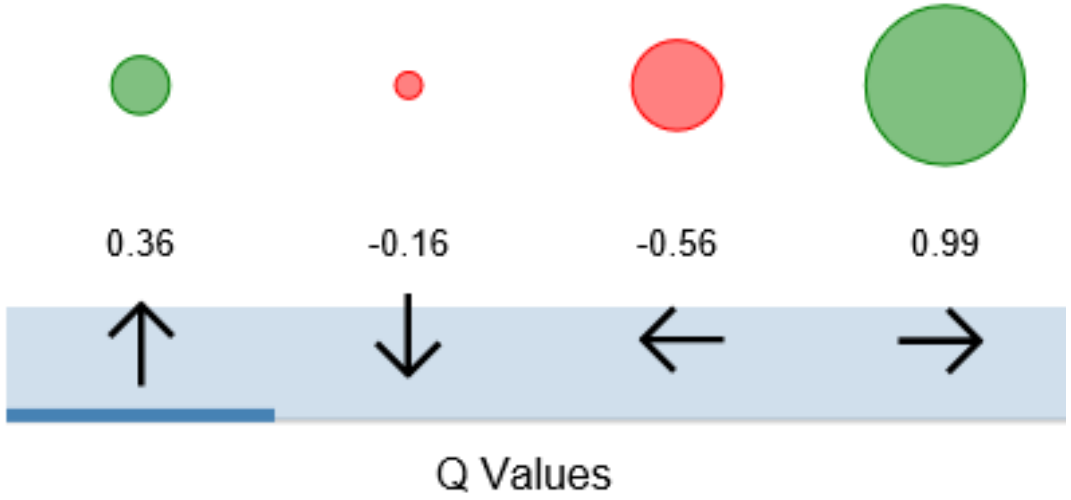
Greedy Approach

- Simply choose the action provide the greatest reward
- Best at the current moment, current knowledge
- No potential exploration



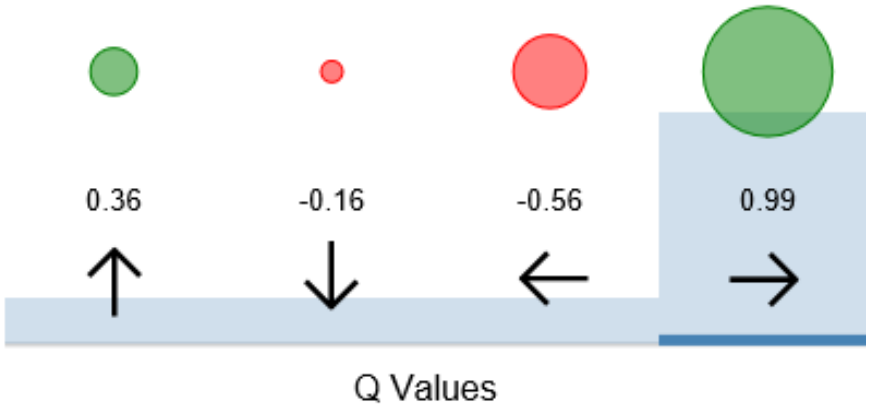
Random Approach

- Opposite to greedy selection: simply choose random action
- More to explore the environment, slow to converge
- Useful in DQN to initial means of sampling to fill an experience buffer



ϵ-greedy

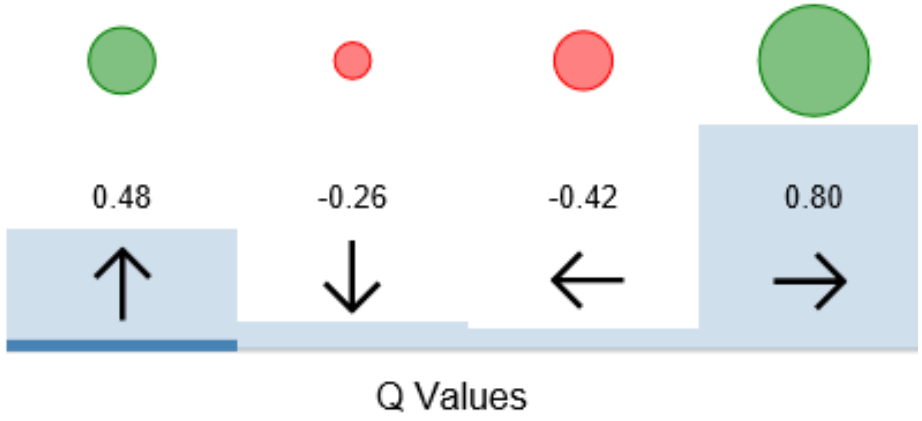
- Combination of greedy and random
- Sometime choose optimal, sometime choose random
- Adjustable parameter ϵ determines the probability of taking a random
- Technique for most recent reinforcement learning algorithms, including DQN and its variants
- Shortcomings: far from optimal



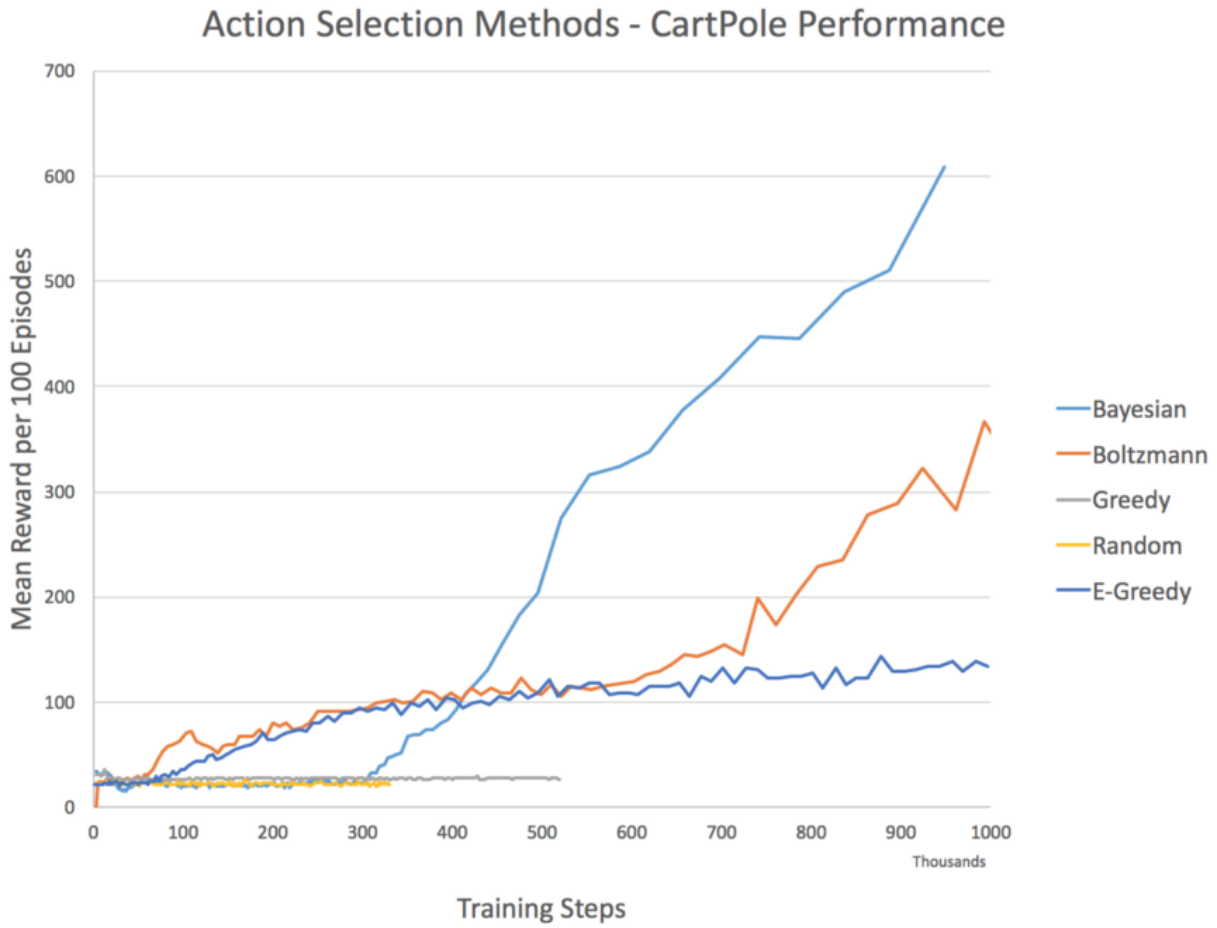
Boltzmann Approach

- Choosing actions with weighted probabilities.
- Use a softmax over networks estimate of each action
- Using parameter to control spread of softmax distribution
- Shortcoming: Based on agent estimating how optimal the action is

$$P_t(a) = \frac{\exp(q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(q_t(i)/\tau)},$$



Comparison of Action Selection Methods

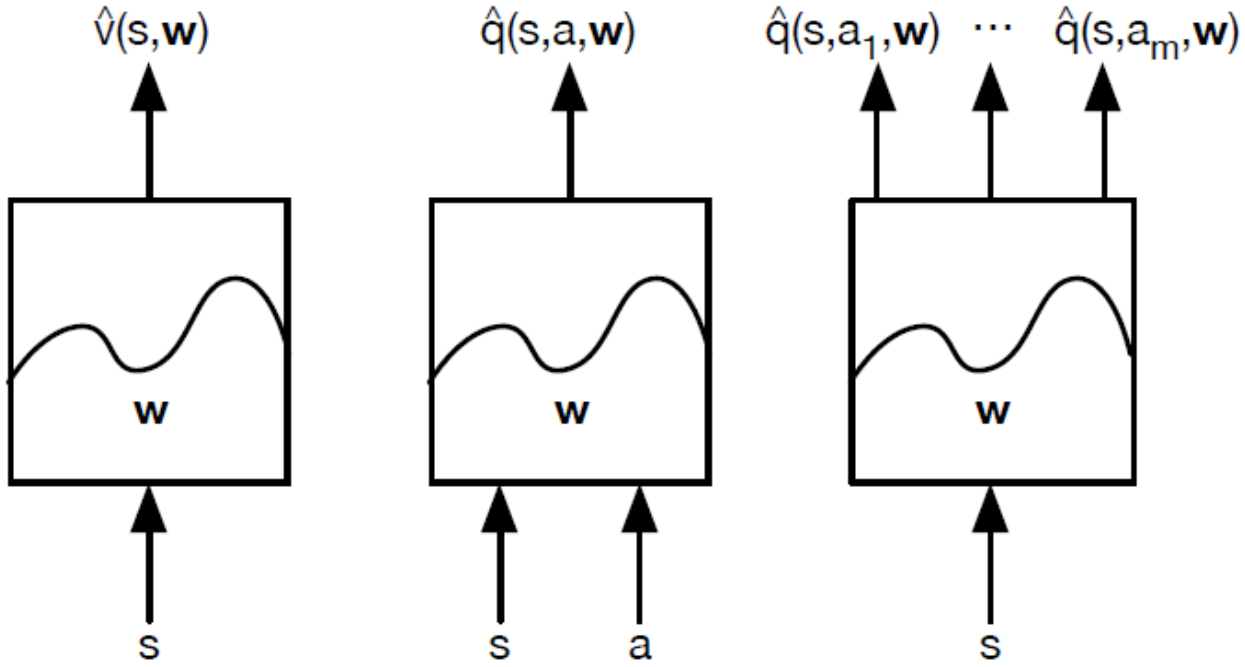


Vietnam Institute for Advanced Study in Mathematics

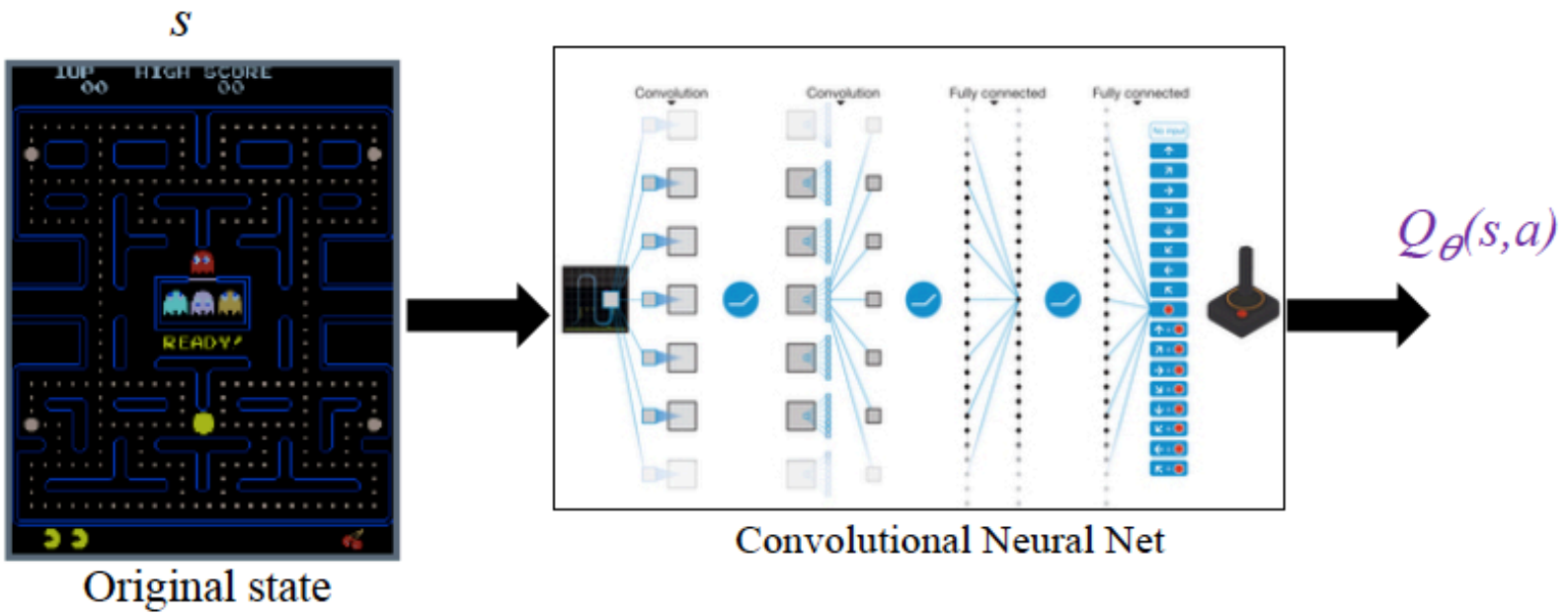
DEEP Q-LEARNING

Function Approximation

- Differentiable function approximation
  - Linear combination of feature
    - Robots: distance from checking point, target, dead mark, wall
    - Business Intelligence Systems: Trends in stock market
  - Neural Network
    - Deep Reinforcement Learning
- Training strategies



Deep Reinforcement Learning: DeepNN + RL



Deep Q-Network trained with stochastic gradient descent.

[DeepMind: Mnih et al., 2015].

Deep Q Learning

- Optimal Q value function

$$Q^*(s,a) = \mathbb{E}\left[R_{t+1} + \max_b Q^*(S_{t+1}, b) \mid S_t = s, A_t = a\right]$$

- Temporal-Difference learning

Target value

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t)\right)$$

- Learning Approximate Q-value function

$$\Delta \boldsymbol{w} = \alpha\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})\right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

Deep Q Learning

- Issue of instance update: small update to Q leads to significant change in policy and correlation between Q and target values
- Key ideas: experience replay and target network
  - Copy network from Q-value function
  - Store experience and sample for training
  - More like supervised learning

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

Deep Q Learning

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1,\mathrm{T}$ **do**

With probability ε select a random action a_t

otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

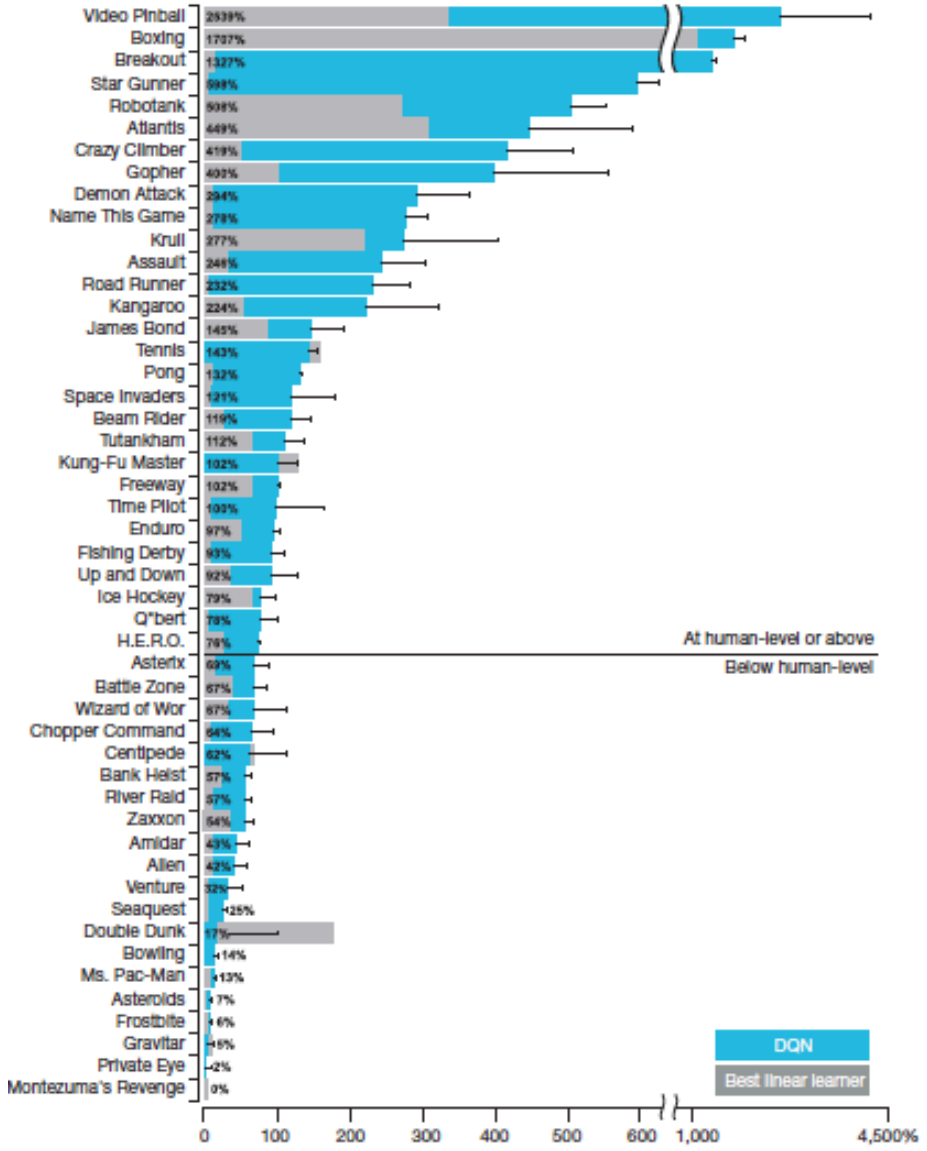
Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For



Vietnam Institute for Advanced Study in Mathematics

Double Q-Learing

- At time t: in state s do action a, observe reward r and state s'
- Q-learning:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha\left(r + \gamma \max_b Q_t(s',b) - Q_t(s,a)\right)$$

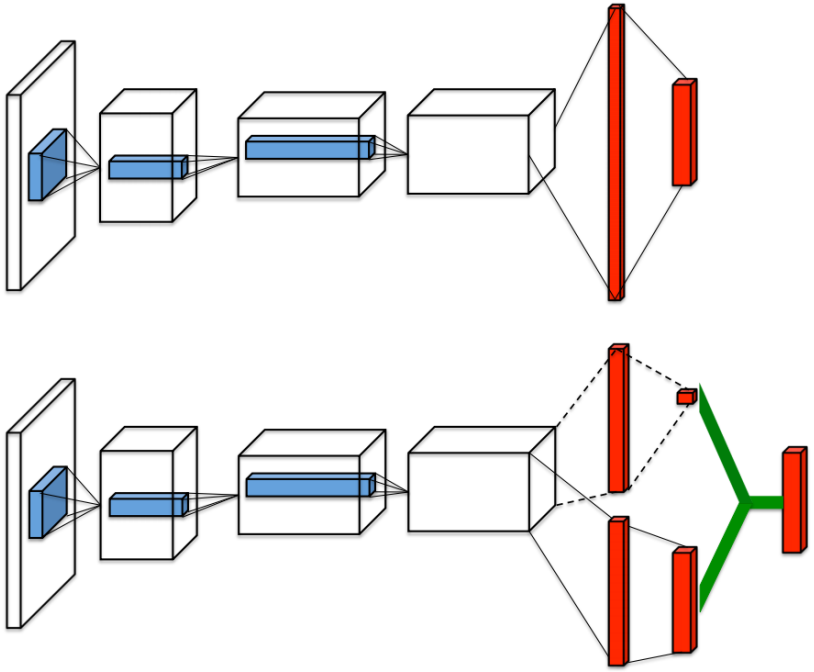
- Q_t is **noisy approximation** of optimal Q^*
- Idea: apply double estimator to Q-learning
- Use two Q-functions: Q^A and Q^B

$$Q^A_{t+1}(s,a) \leftarrow Q^A_t(s,a) + \alpha\left(r + \gamma Q^B_t(s', a^*) - Q^A_t(s,a)\right) \quad \text{or}$$
$$Q^B_{t+1}(s,a) \leftarrow Q^B_t(s,a) + \alpha\left(r + \gamma Q^A_t(s', b^*) - Q^B_t(s,a)\right)$$

where $a^* = \arg\max_a Q^A(s',a)$

$b^* = \arg\max_a Q^B(s',a)$

- Each time step: update only one (e.g., random pick Q^A or Q^B)
- Use average of Q^A and Q^B to choose actions
- Same time and space complexity as Q-learning



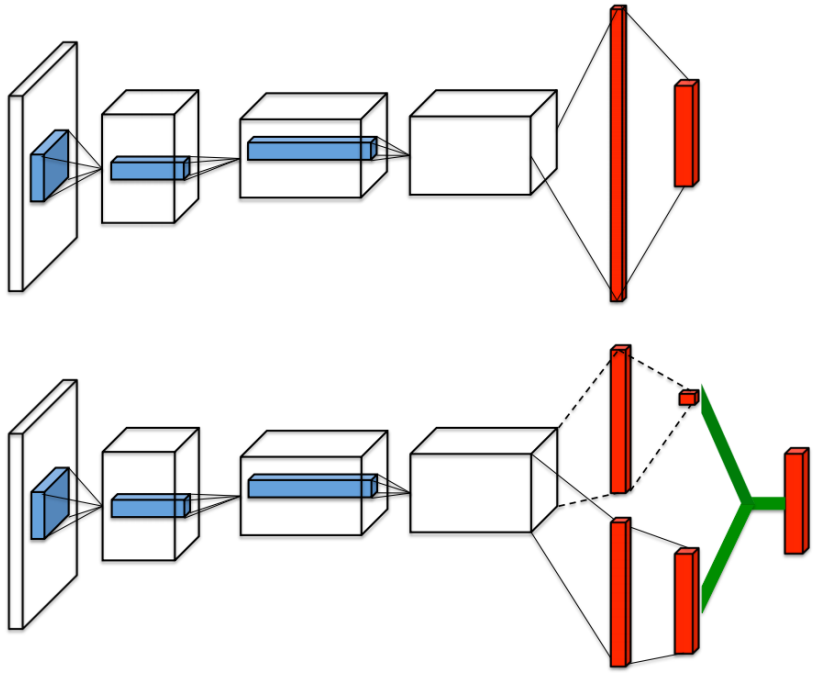
Above: DQN
Bellow: Dueling DQN

Dueling Q-Learning

- Separate value and advance functions

  $Q(s,a) = V(s) + A(a)$

- Combine them back into a single Q-function at the final layer
- Key idea:
  - Agents not need to care about both value and advantage at any given time.
  - Example: Robots see and evaluate state while not need to do actions



Above: DQN
Bellow: Dueling DQN

POLICY GRADIENTS AND ACTOR-CRITIC METHODS

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find the best θ
- Define objective function to measure quality of policy
  - In episodic environments, objective function is the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

  - In continuing environments, objective function is average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

  - Or average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Optimization

- Policy based RL is an optimization problem
- Find θ that maximize objective function $J(\theta)$
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending gradient of the policy $\Delta\theta = \alpha \nabla_\theta J(\theta)$

Theorem

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

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

Reducing Variance using a Critic

- Monte-Carlo policy gradient has high variance
- A Critic is used to estimate action-value function

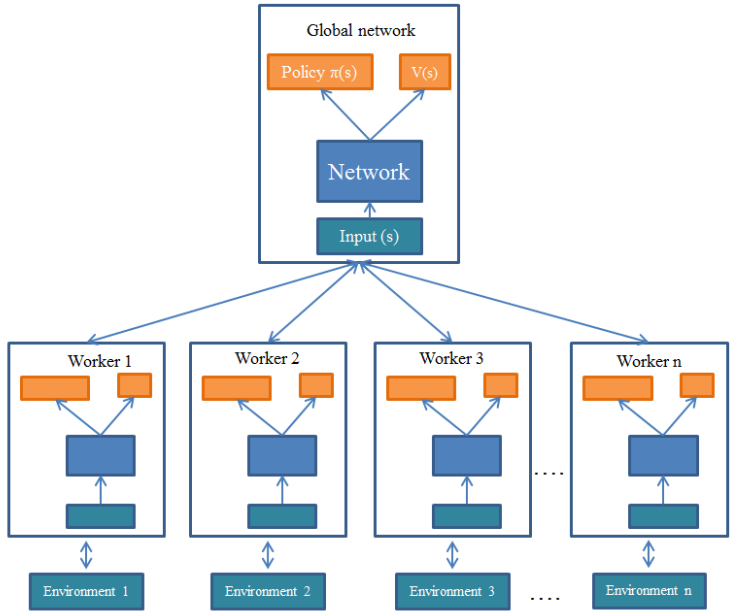
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

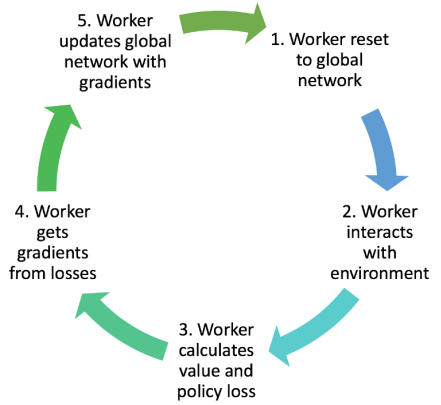
- Actor-critic algorithms maintain two sets of parameters
  - Critic: Updates action-value function parameter w
  - Actor: Updates policy parameter θ
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)$$

Asynchronous Advantage Actor-Cı (A3C)

- Asynchronous:
  - Multiple worker have their own set of network parameters
  - Agents interacts with it's own copy of the environment at the same time as the other agents
- Actor-Critic: Estimate both a value function $V(s)$ (how good a certain state is to be in) and a policy $\pi(s)$ (a set of action probability outputs)
- Advantage Estimate: $A=R-V(s)$



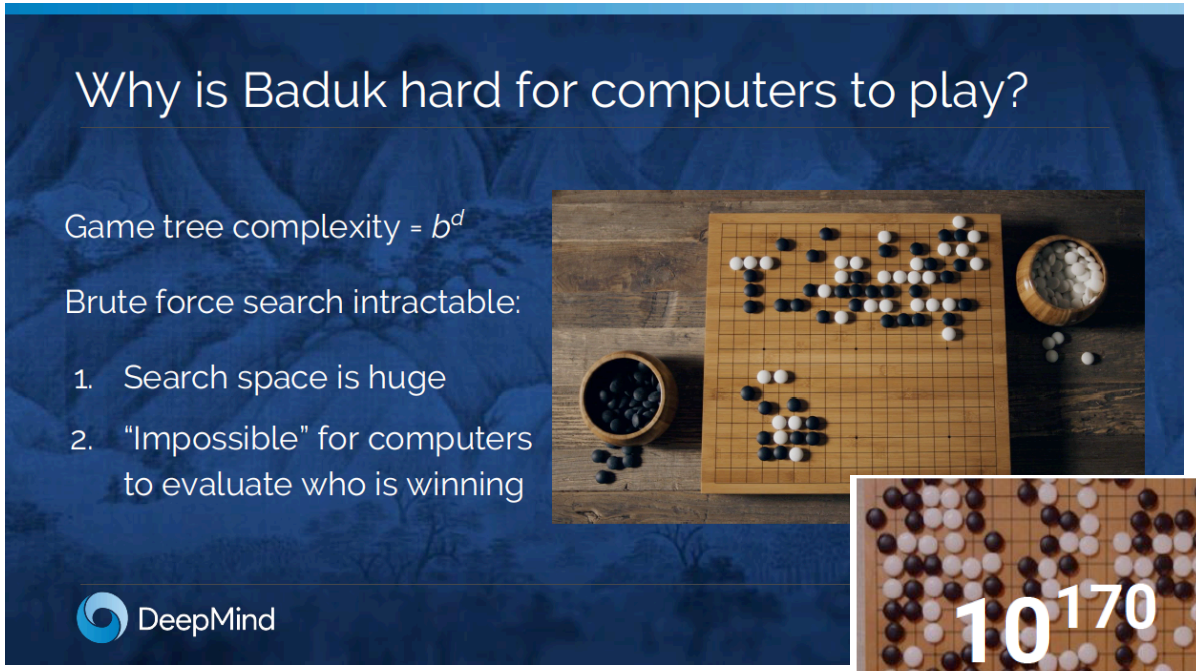


Vietnam Institute for Advanced Study in Mathematics

ALPHA-GO

Alpha Go

- The game of Go
  - Number of legal moves per position b: 250
  - Game length (depth) d: 150
  - Much more than chess (b=35, d =80)
- Deal with valuated network and MC Tree Search




ARTICLE

doi:10.1038/nature16961

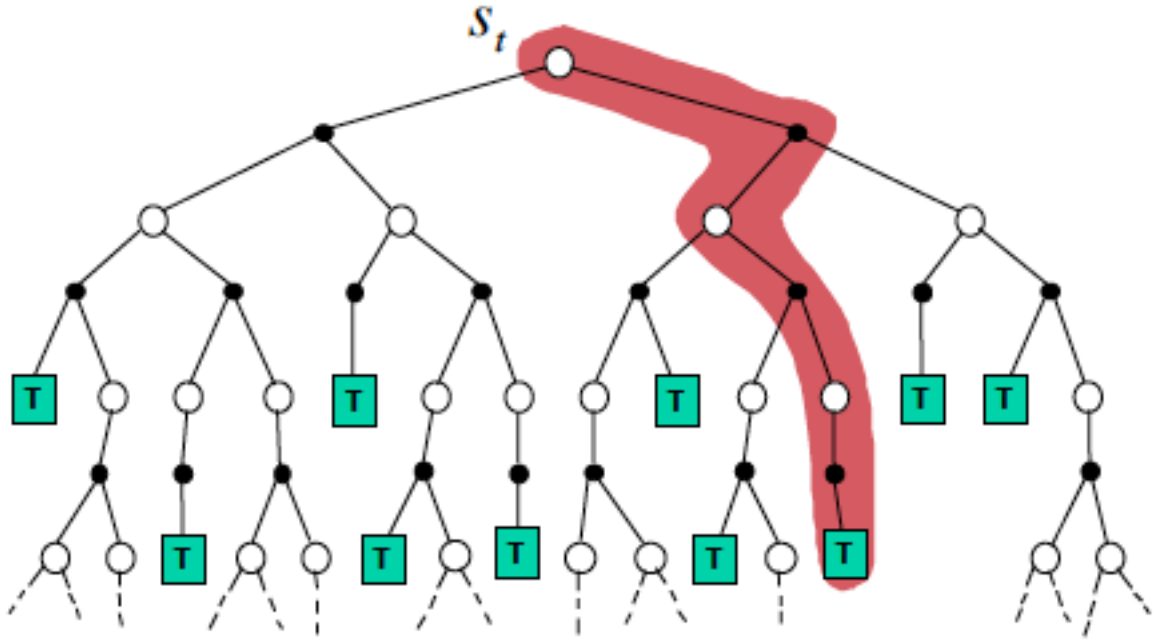
Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[3], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

VIASM
Vietnam Institute for Advanced Study in Mathematics

Policy search

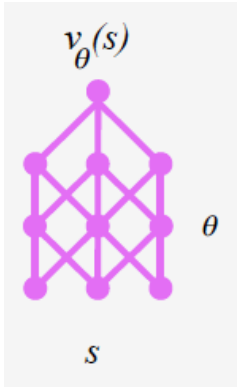
- Forward search algorithms select the best action by look ahead
  - Search tree with current sate s(t) as root
  - Using a model to look ahead
  - No need to solve the whole MDP, just sub of MDP from the current state
- Simulated-Based Search
  - Simulate episodes of experience from current state
  - Apply model-free RL to simulated episodes

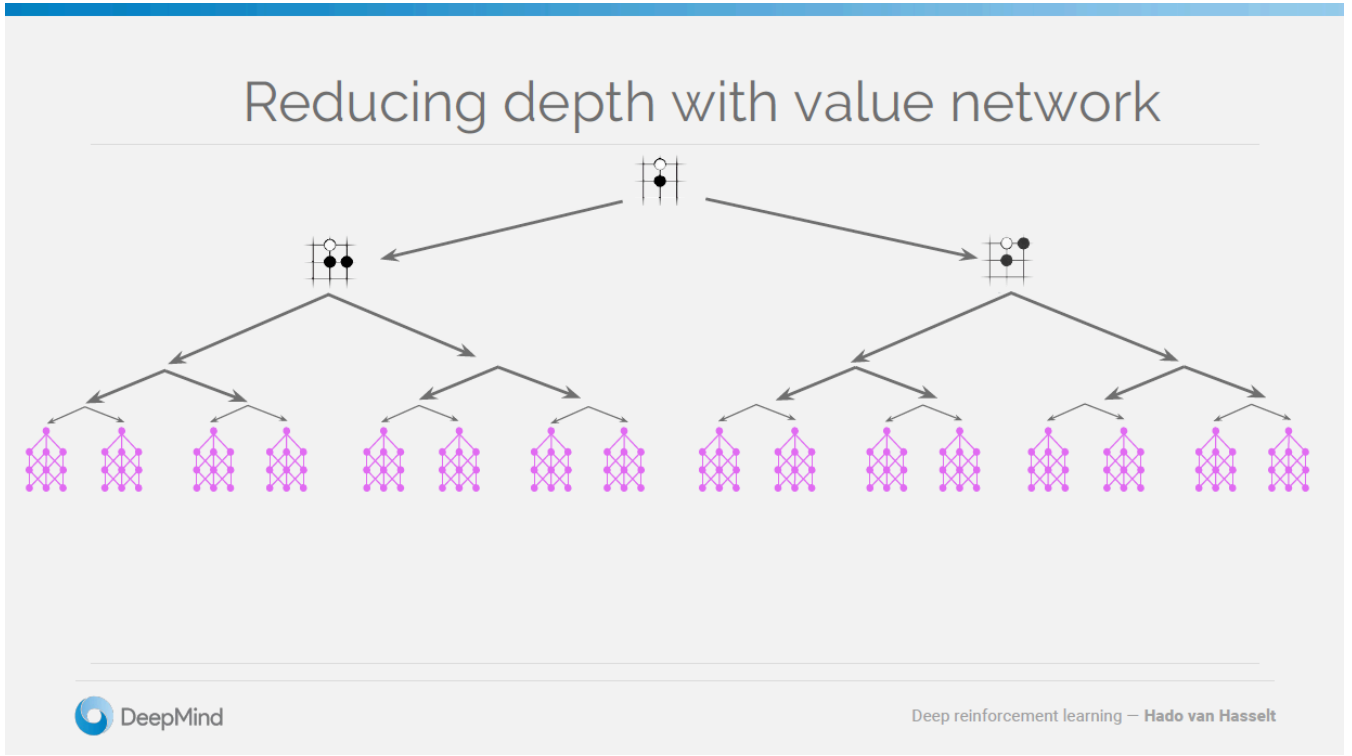


$$\{s_t^k, A_t^k, R_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu$$

Policy search motivation

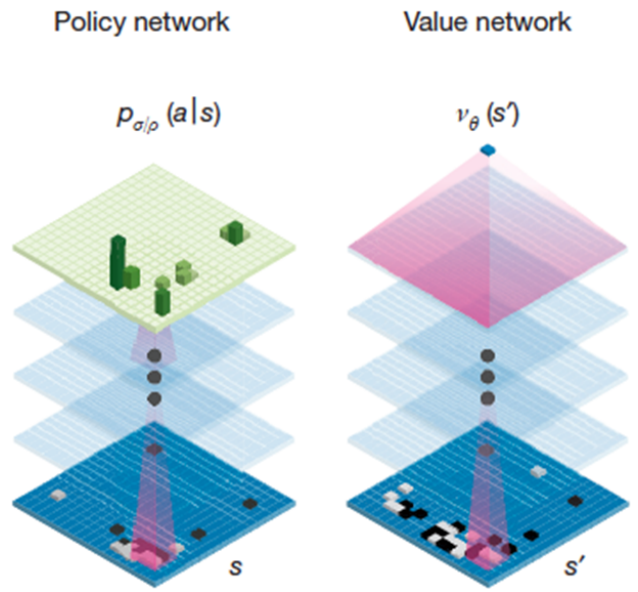
- RL for the game of Go
  - Searching for state value
  - How to reduce search space?
  - State value:

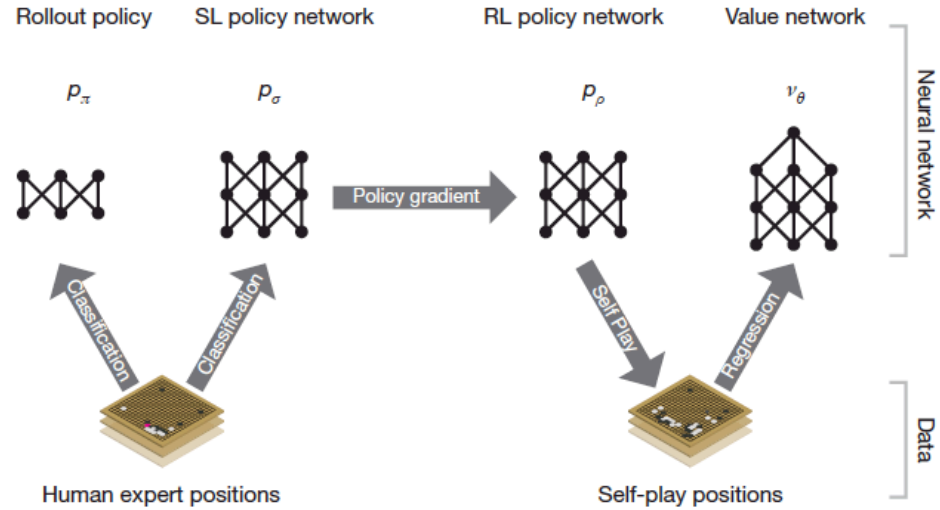




Neural network training pipeline and architecture

- Supervised train two policy network
  - Fast and less accurate p_π
  - More layers network p_σ
- Self train SL policy network to improve policy network p_ρ
- Self train value network v_θ

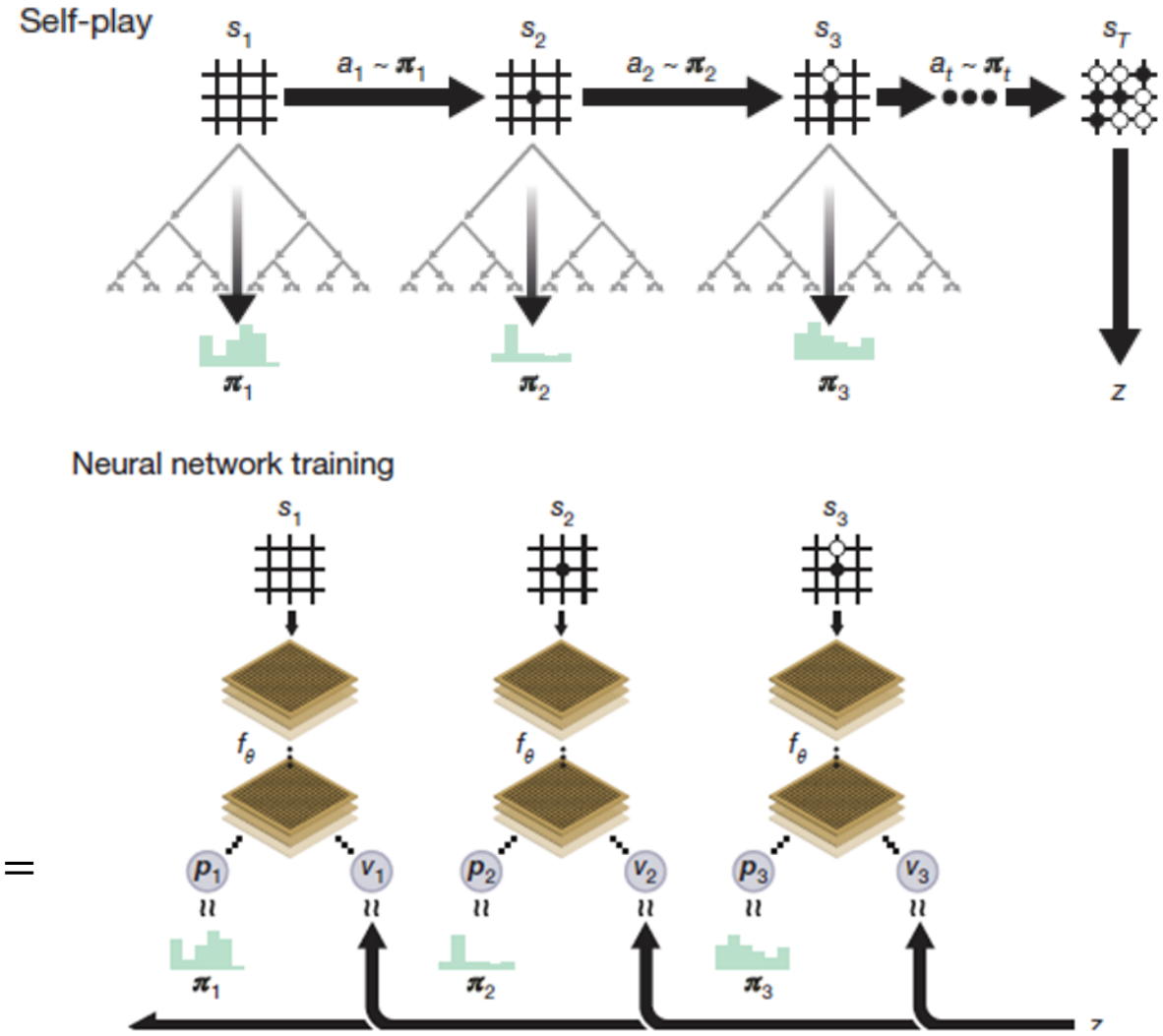




Vietnam Institute for Advanced Study in Mathematics

AlphaGo Zero (Chess)

- Self-play pipeline
  - Build search tree
  - Sample action $a_t \sim \pi_t$ using MCTS with the latest network f_θ
  - Playing at the current state
  - Update weigh vectors
  - At the end, store winner z
- Neural network training
  - Initial to random weight θ_0
  - At time t, execute network $f_{\theta_{t-1}}$ using the last iteration θ_{t-1}
  - When each episode ends, at T, score the reward $r_T = \{\pm 1\}$
  - Store winner score for any step $\{s, \pi, z\}$
  - In parallel, train θ_t using data sample from $\{s, \pi, z\}$ with loss function l



VIASM
Vietnam Institute for Advanced Study in Mathematics

Visualization and Codes

- https://cs.stanford.edu/people/karpathy/reinforcejs/index.html
- https://github.com/awjuliani/DeepRL-Agents/blob/master/Q-Network.ipynb
- http://awjuliani.github.io/exploration/index.html
- https://github.com/awjuliani/DeepRL-Agents/blob/master/Q-Exploration.ipynb

Recap

- Many Challenges on Data and Planning
- When to Exploration and Exploitation
- Deep Q-Learning
- Policy Gradient with Actor-critic
- Alpha-Go

Final project

- Final Project
  - Group score: score for assignment
  - Individual score: score for final examination
  - Time: 9 June 2020
- Day off: 2 June 2020

THANK YOU!

Vietnam Institute for
Advanced Study in Mathematics