Deep Learning

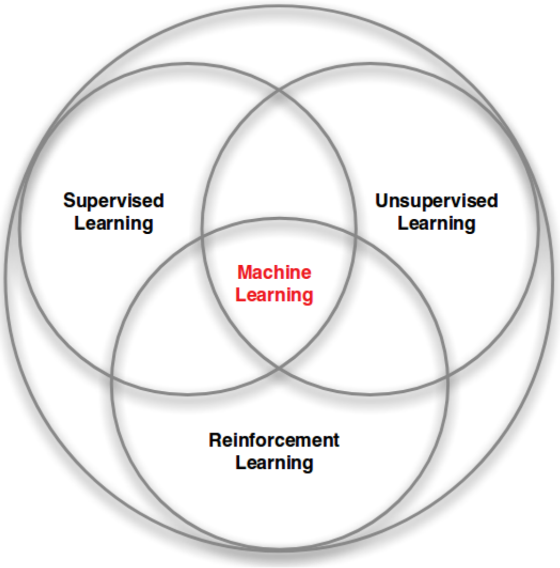
13. Reinforcement learning.

Viacheslav Dudar

Taras Shevchenko National University of Kyiv

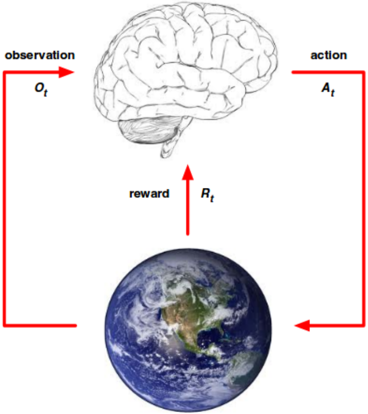
2018

Machine learning branches



What is reinforcement learning?

Agents receive **observations** and take **actions** in some **envinronment** to maximize **cumulative reward**.



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Reward

Reward hypothesis: all goals can be described by maximization of expected cumulative reward.

Examples:

- Play chess: + reward for win and - for lose
- Make a robot walking: + for forward motion, - for falling

Agent's aim

Goal: select actions maximizing total future reward
Difficulties:

- Actions may have long term consequences
- Agent doesn't know how envinronment works
- Envinronment could be stochastic
- Positive experience could be very rare

State

History: all observable variables up to time t:

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

State: information used to determine what happens next, function of history

$$S_t = f(H_t)$$

Envinronment state S_t^e: information envinronment uses to pick next observation / reward
Agent state: information agent uses to pick next observation / reward

Markov state

Information state (Markov state) contains all information from the history:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_n]$$

State is sufficient statistics of future

Examples: positions and velocities in the dynamical system, board and player turn in chess.

Observability

Full observability: agent observes full envinronment state (MDP: Markov decision process)

$$O_t = S_t^a = S_t^e$$

Example: chess
Partial observability: agent observes envinronment indirectly (POMDP: partially observable Markov decision process)

$$S_t^a \neq S_t^e$$

Example: Autonomous vehicle

Agent components

Policy π: agent's behaviour

- Deterministic: $a = \pi(s)$
- Stochastic: $\pi(a|s) = P(A_t = a|S_t = s)$

Value function: prediction of total future reward

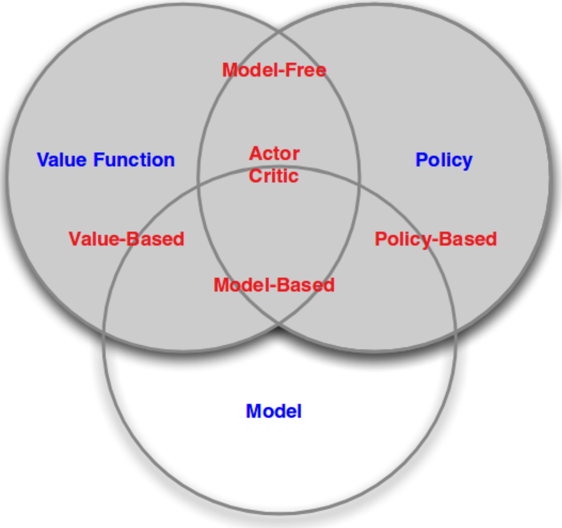
$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$$

Model: prediction of next state and reward given action.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

Agent types



Model-Free
Value Function
Actor
Critic
Policy
Value-Based
Policy-Based
Model-Based
Model

Markov process

State transition probability:

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix:

$$\mathcal{P} \quad = \begin{array}{cc} & \textit{to} \\ \textit{from} & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

Markov reward process

- S: finite set of states
- P: state transition probability matrix
- R: reward function, $R_s = E[R_{t+1} | S_t = s]$
- γ: discount factor

Return and value function

Return: total discounted reward from time-step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Value function:

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Bellman equation

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

$$
v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')
$$

Solution:

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(I - \gamma \mathcal{P})\, v &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

Markov decision process

MDP = Markov reward process with decisions

- S: finite set of states
- A: finite set of actions
- P: state transition probability matrix

$$P(S_{t+1} = s' | S_t = s, A_t = a)$$

- R: reward function, $R_s^a = E[R_{t+1} | S_t = s, A_t = a]$
- γ: discount factor

Policy

Policy determines behaviour of the agent:

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

Given MDP and policy π state and reward sequence is a Markov reward process:

$$\mathcal{P}^{\pi}_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{R}^{a}_{s}$$

Action-value function

State-value function:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Bellman Expectation equation

For state value:

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\right)$$

For action value:

$$q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')$$

Optimal value function

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Optimal policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy* π_* *that is better than or equal to all other policies,* $\pi_* \geq \pi, \forall \pi$
- *All optimal policies achieve the optimal value function,* $v_{\pi_*}(s) = v_*(s)$
- *All optimal policies achieve the optimal action-value function,* $q_{\pi_*}(s, a) = q_*(s, a)$

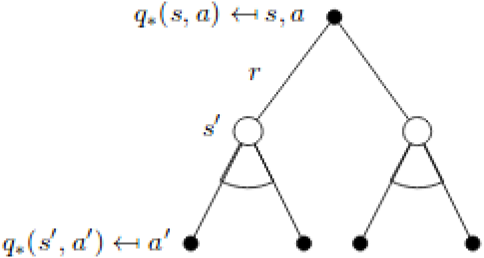
Optimal policy search

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & otherwise \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Bellman optimality equation



$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

Optimality equation is nonlinear - iterative methods are used to solve.

Approaches to RL

Approaches To Reinforcement Learning

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- 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

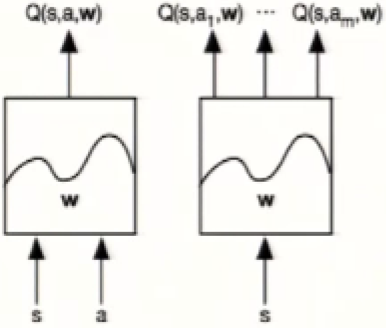
Deep reinforcement learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimise loss function by stochastic gradient descent

Q-network

Represent value function by Q-network with weights $\mathbf{w}$

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



Q-learning

Q-Learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

DQN

To remove correlations, build data-set from agent's own experience

	s_1, a_1, r_2, s_2	
	s_2, a_2, r_3, s_3	$\rightarrow$ s, a, r, s'
	s_3, a_3, r_4, s_4	
	$\ldots$	
$s_t, a_t, r_{t+1}, s_{t+1}$ $\rightarrow$	$s_t, a_t, r_{t+1}, s_{t+1}$	

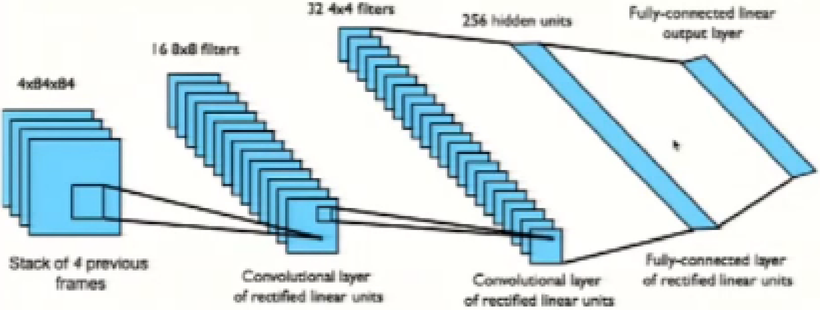
Sample experiences from data-set and apply update

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

Improvements over DQN

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

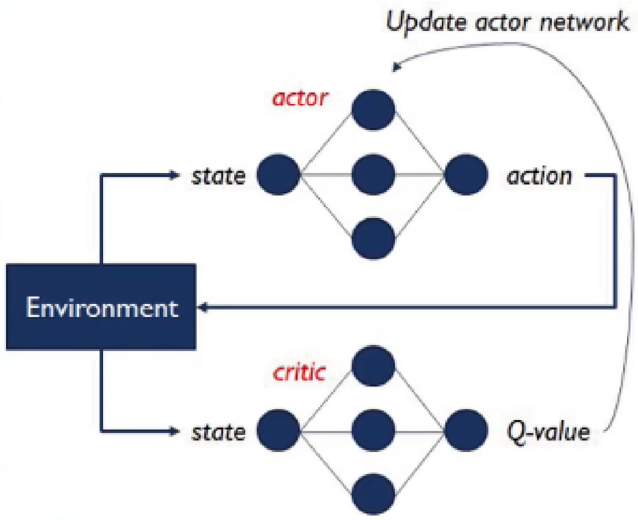
- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Duelling network: Split Q-network into two channels
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Actor-critic model



Continuous action spaces

DPG is the continuous analogue of DQN

- Experience replay: build data-set from agent's experience
- Critic estimates value of current policy by DQN

$$l_w = \left(r + \gamma Q(s', \pi(s', u^-), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

  To deal with non-stationarity, targets $\mathbf{u}^-, \mathbf{w}^-$ are held fixed

- Actor updates policy in direction that improves Q

$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

- In other words critic provides loss function for actor

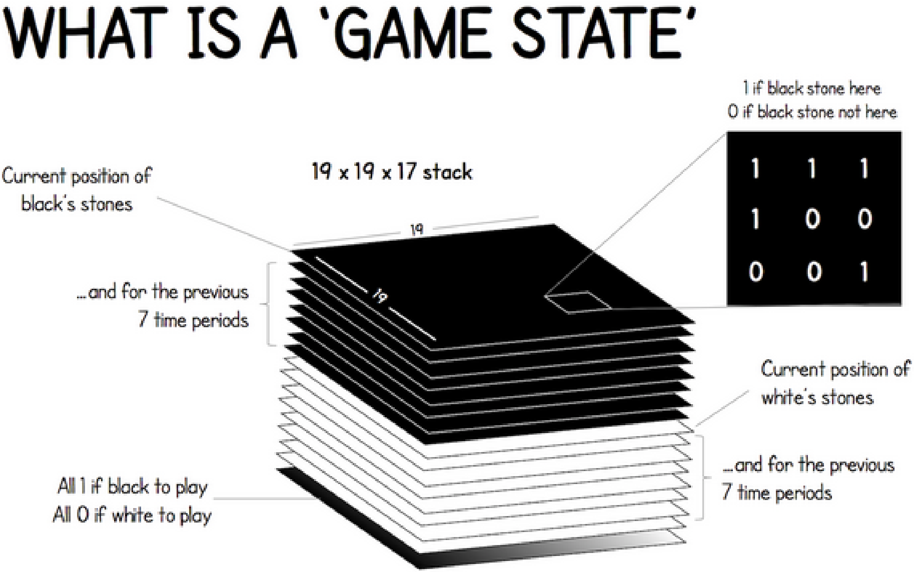
Model based

- Challenging to plan due to compounding errors
  - Errors in the transition model compound over the trajectory
  - Planning trajectories differ from executed trajectories
  - At end of long, unusual trajectory, rewards are totally wrong

Alpha Go Zero

- Trains entirely on self-play
- No handcrafted features
- Residual network
- Combined policy and value net

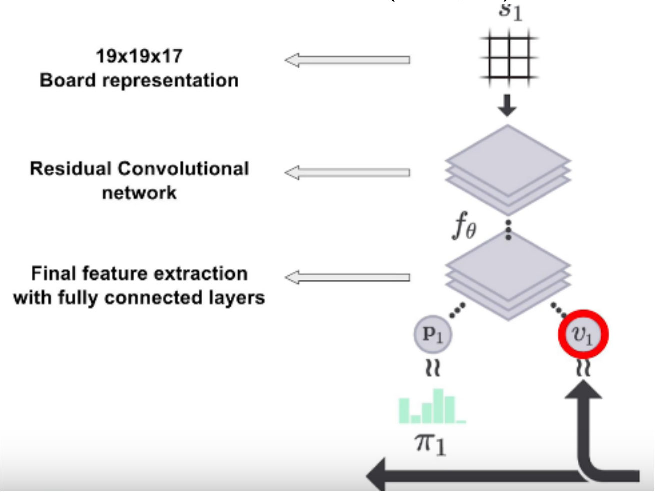
State



WHAT IS A 'GAME STATE'
1 if black stone here
0 if black stone not here
19 x 19 x 17 stack
Current position of black's stones
19
19
1 1 1
1 0 0
0 0 1
...and for the previous 7 time periods
Current position of white's stones
...and for the previous 7 time periods
All 1 if black to play
All 0 if white to play

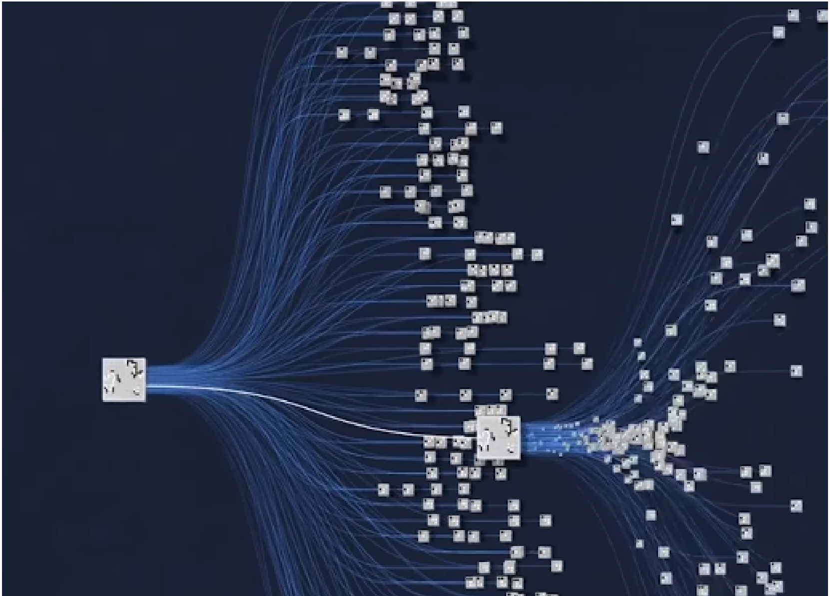
Network architecture

Residual convolutional network (34 layers)

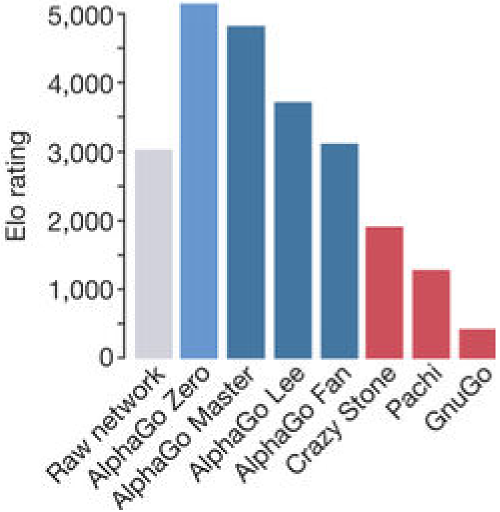


Make training stable

Monte-Carlo tree search (1600 positions):



Alpha Go comparison



Read more

10 lectures on reinforcement learning: https://www.youtube.com/playlist?list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT
Deep RL: https://www.youtube.com/watch?v=M5a6HasTHs4
Deep RL: https://www.youtube.com/watch?v=lvoHnicueoE
OpenAI gym (train to walk): https://gym.openai.com/
Textbook on AI (before neural nets): https://www.cin.ufpe.br/~tfl2/artificial-intelligence-modern-approach.9780131038059.25368.pdf