Introduction to Deep Reinforcement Learning

Shenglin Zhao
Department of Computer Science & Engineering
The Chinese University of Hong Kong

Outline

- Background
- Deep Learning
- Reinforcement Learning
- Deep Reinforcement Learning
- Conclusion

Outline

- Background
- Deep Learning
- Reinforcement Learning
- Deep Reinforcement Learning
- Conclusion

Milestone Issues

- NIPS 2013, DeepMind, Playing Atari with Deep Reinforcement Learning, https://arxiv.org/abs/1312.5602
- Nature cover paper 2015, DeepMind, Human-level control through deep reinforcement learning, www.nature.com/articles/nature14236
- Nature cover paper 2016, DeepMind, Mastering the game of Go with deep neural networks and tree search, www.nature.com/articles/nature16961

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

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

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

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

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

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Examples of Deep RL@DeepMind

- Play games: Atari, poker, Go, ...
- Explore worlds: 3D worlds, Labyrinth, ...
- Control physical systems: manipulate, walk, swim, ...
- Interact with users: recommend, optimise, personalise, ...

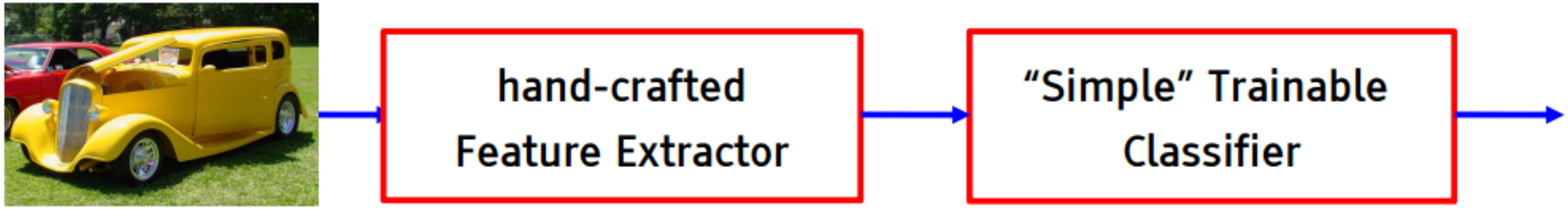
http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Outline

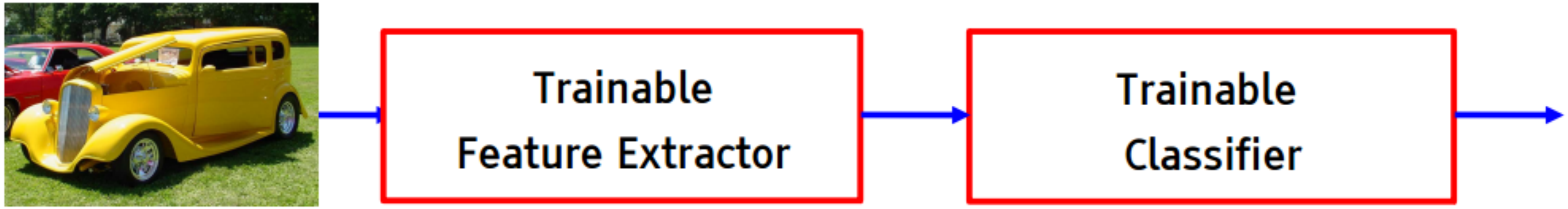
- Background
- Deep Learning
- Reinforcement Learning
- Deep Reinforcement Learning
- Conclusion

Deep Learning = Learning Representations/Features

- Traditional Model



- Deep Learning



http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

Deep Representations

- A deep representation is a composition of many functions

$$
\begin{tikzcd}
x \arrow[r] & h_1 \arrow[r] & \ldots \arrow[r] & h_n \arrow[r] & y \arrow[r] & l \\
& \mathbf{w_1} \arrow[u] & \ldots & \mathbf{w_n} \arrow[u] & &
\end{tikzcd}
$$

- Its gradient can be backpropagated by the chain rule

$$
\begin{tikzcd}
\frac{\partial l}{\partial x} & \frac{\partial l}{\partial h_1} \arrow[l, "\frac{\partial h_1}{\partial x}"'] \arrow[d, "\frac{\partial h_1}{\partial w_1}"'] & \ldots \arrow[l, "\frac{\partial h_2}{\partial h_1}"'] & \frac{\partial l}{\partial h_n} \arrow[l, "\frac{\partial h_n}{\partial h_{n-1}}"'] \arrow[d, "\frac{\partial h_n}{\partial w_n}"'] & \frac{\partial l}{\partial y} \arrow[l, "\frac{\partial y}{\partial h_n}"'] \\
& \frac{\partial l}{\partial \mathbf{w_1}} & \ldots & \frac{\partial l}{\partial \mathbf{w_n}} &
\end{tikzcd}
$$

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Deep Neural Network

A deep neural network is typically composed of:

- Linear transformations

$$h_{k+1} = Wh_k$$

- Non-linear activation functions

$$h_{k+2} = f(h_{k+1})$$

- A loss function on the output, e.g.
  - Mean-squared error $l = ||y^* - y||^2$
  - Log likelihood $l = \log \mathbb{P}[y^*]$

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Example-CNN

Convolutional Operator

$$(f * g)(t) \overset{\mathrm{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

Discrete Form

$$(f * g)[n] \overset{\mathrm{def}}{=} \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$

Matrix Element-wise Multiplication

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 2 & 7 \\ 5 & 0 & 7 \\ 1 & 2 & 1 \end{bmatrix} = 7$$

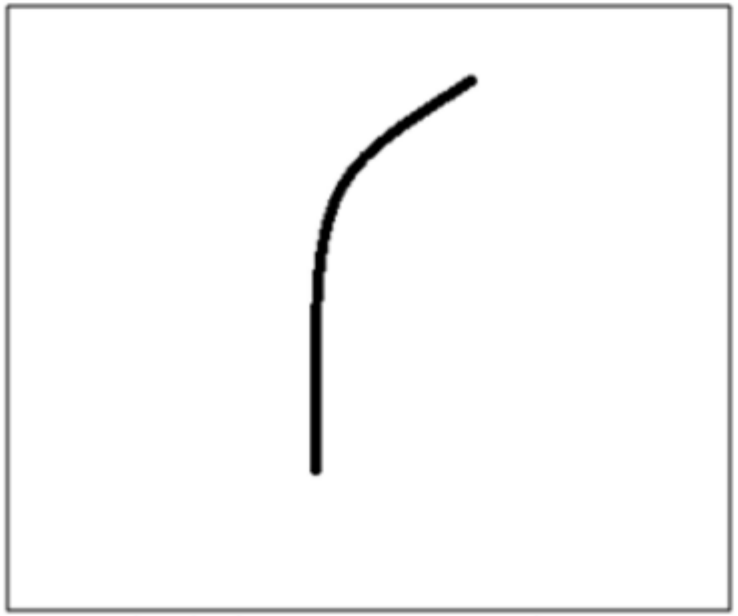
http://xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/

Example-CNN

Pixel representation of filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Visualization

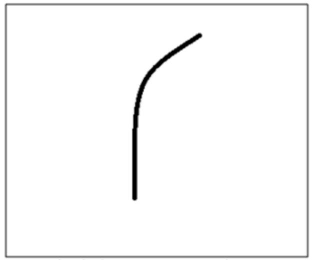


https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

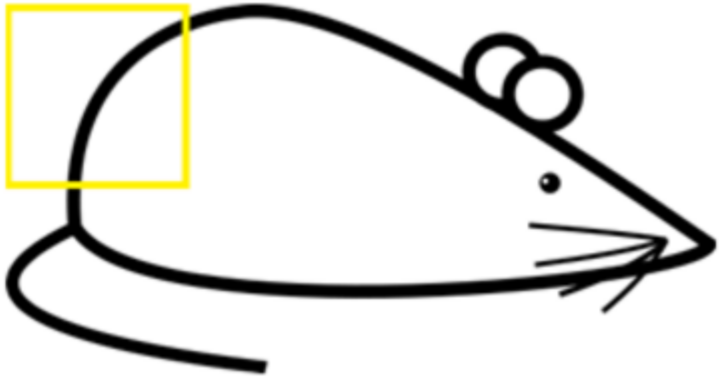
Example-CNN

Original image

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

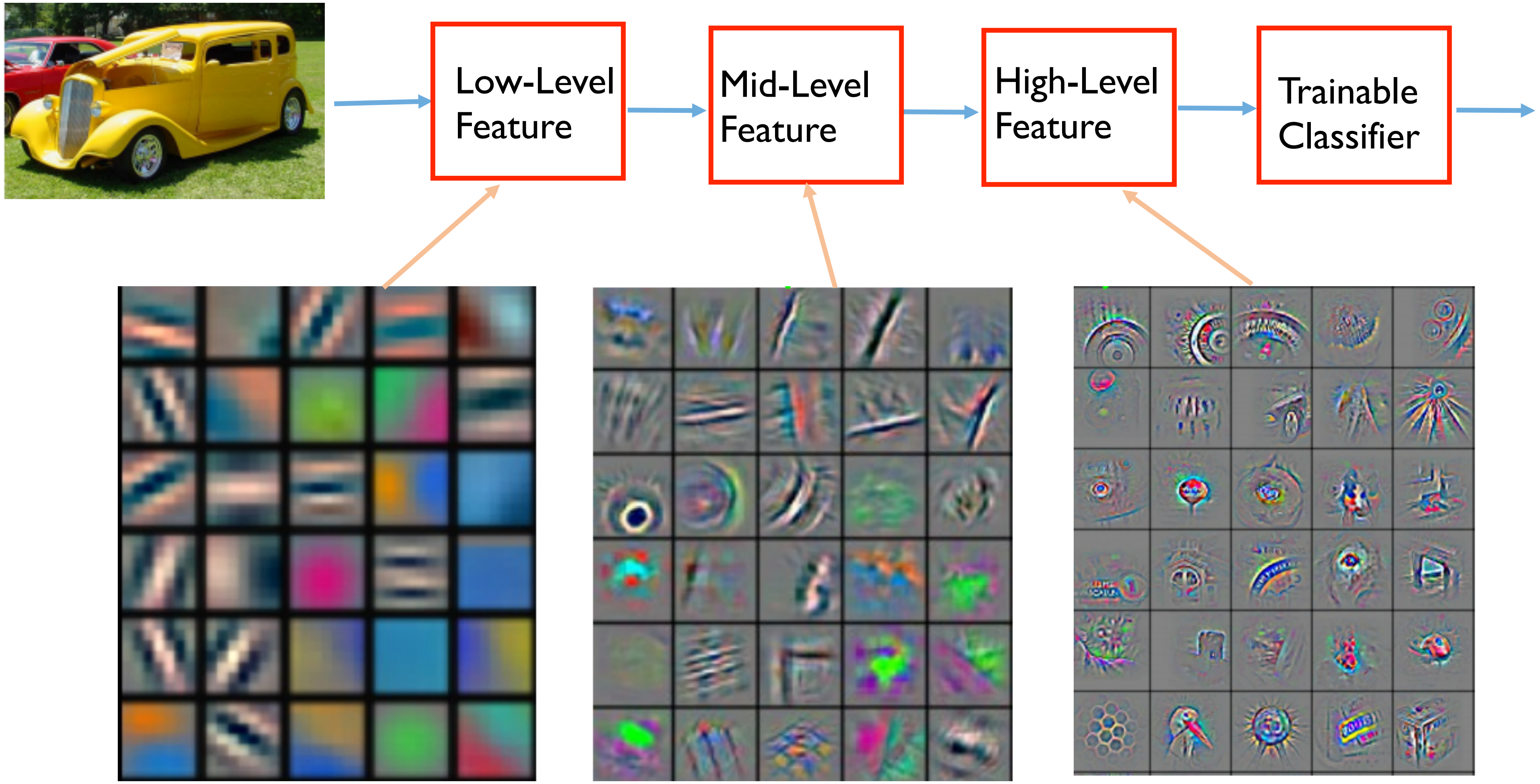


What we find



https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Example-CNN

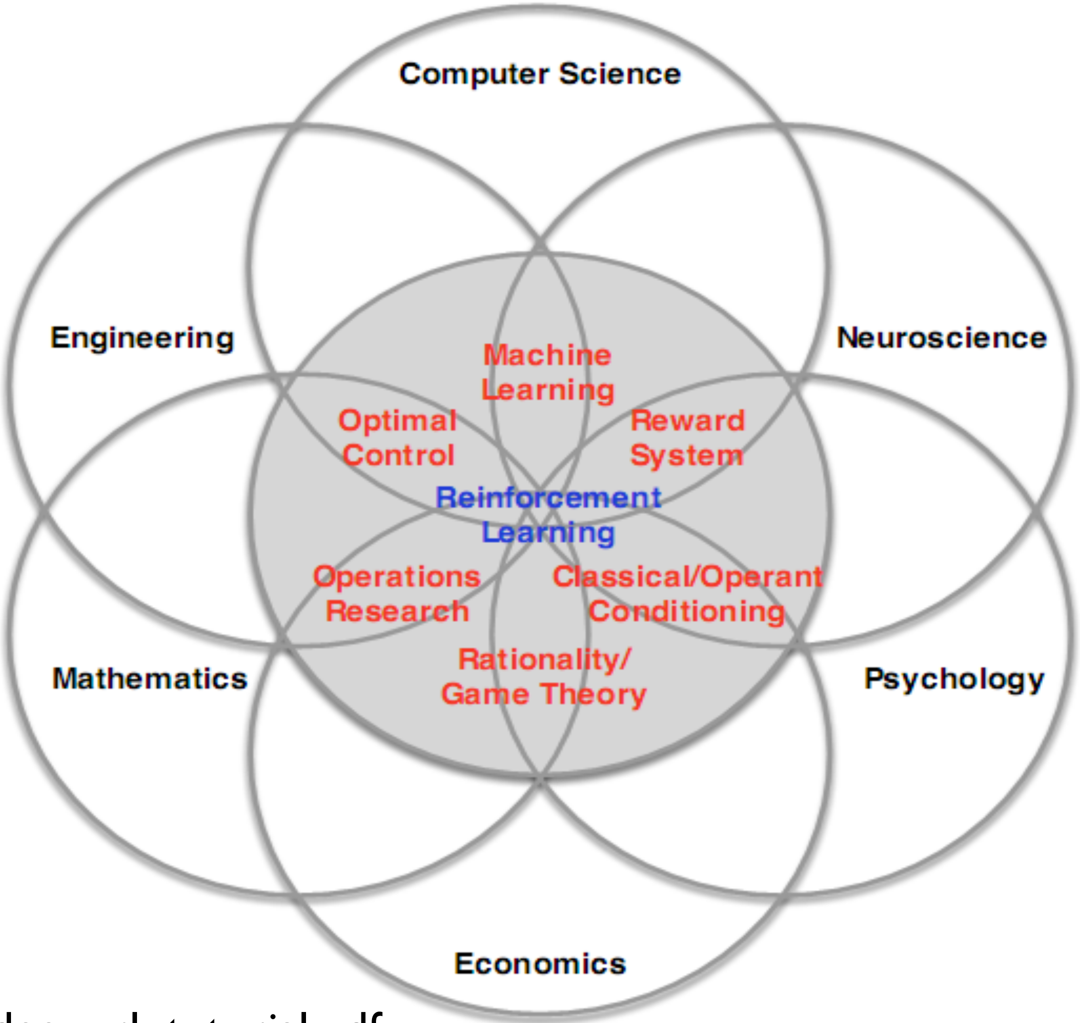


http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

Outline

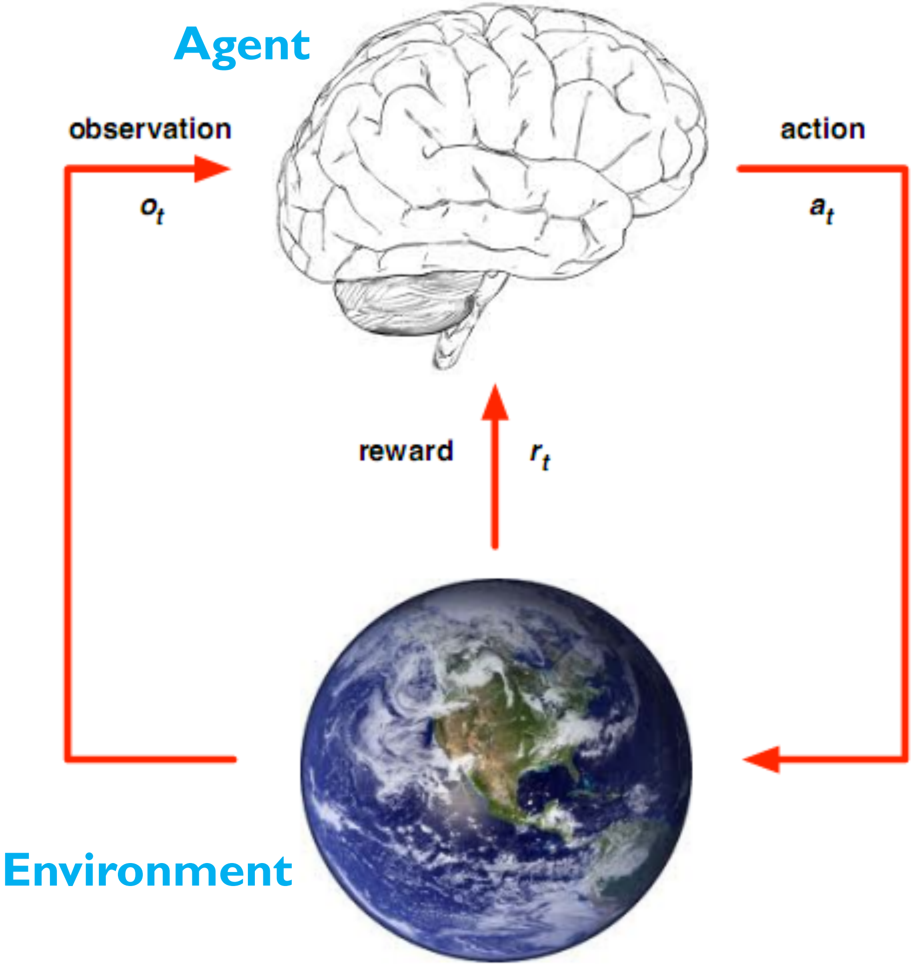
- Background
- Deep Learning
- Reinforcement Learning
- Deep Reinforcement Learning
- Conclusion

Reinforcement Learning



http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Agent and Environment



- 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}

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

State

- Experience is a sequence of observations, actions, rewards

$$o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t$$

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t)$$

- In a fully observed environment

$$s_t = f(o_t)$$

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Major Components

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Policy

- A policy is the agent's behaviour
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[a|s]$

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Value Function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

Bellman equation

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

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Optimal Case

- An optimal value function is the maximum achievable value

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

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s, a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

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

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

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

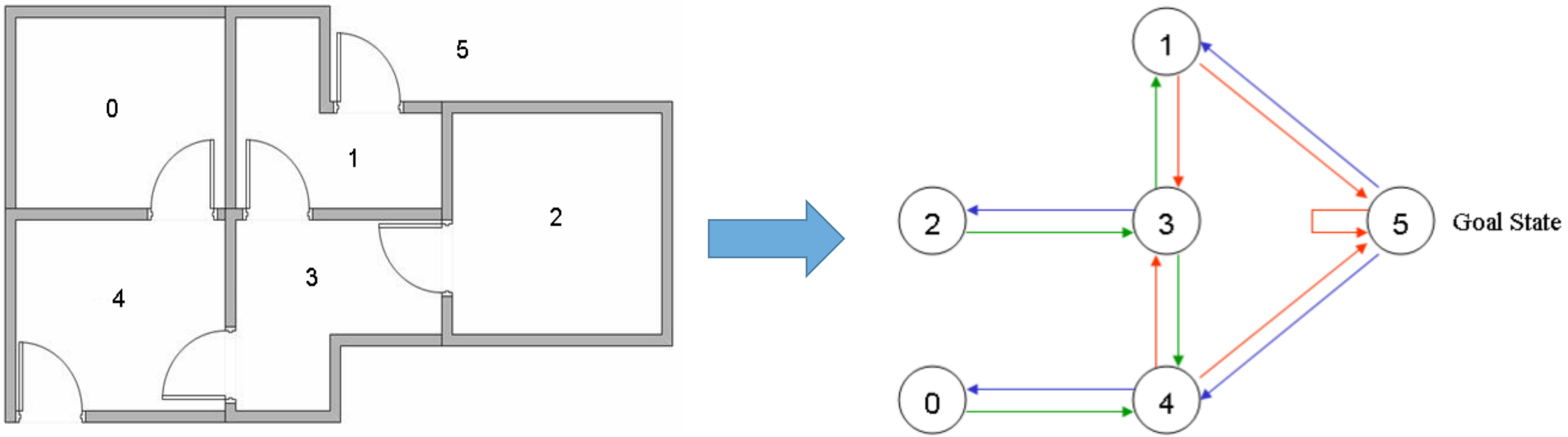
Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

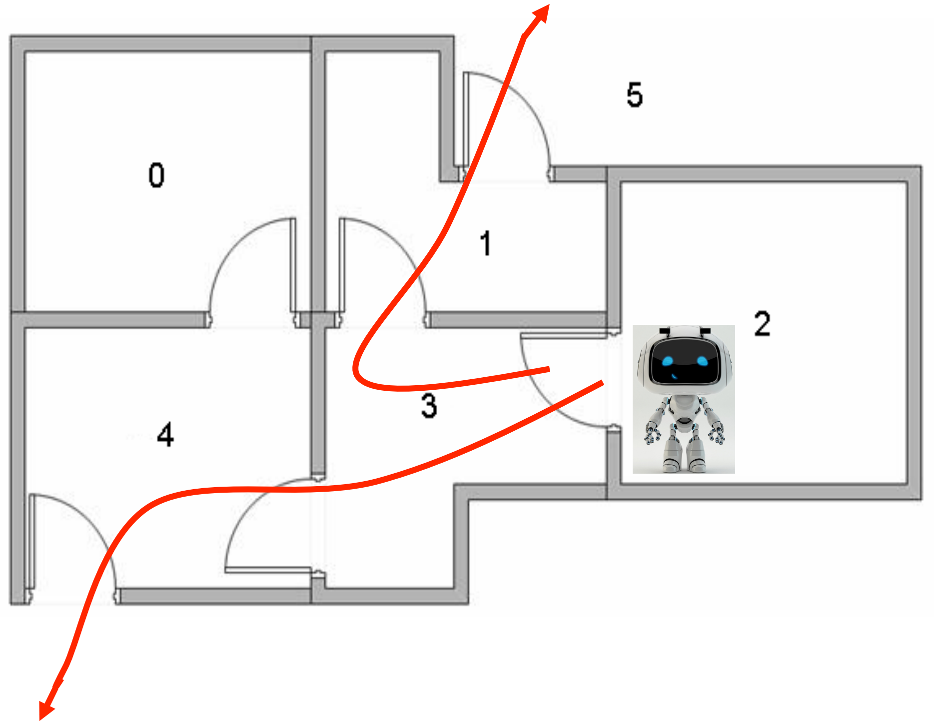
RL Example

- Assumption
  - Suppose we have 5 rooms in a building connected by doors
  - The outside of the building can be thought of as one big room (5)
- Target
  - Put an agent in any room, and from that room, go outside the building



http://mnemstudio.org/path-finding-q-learning-tutorial.htm

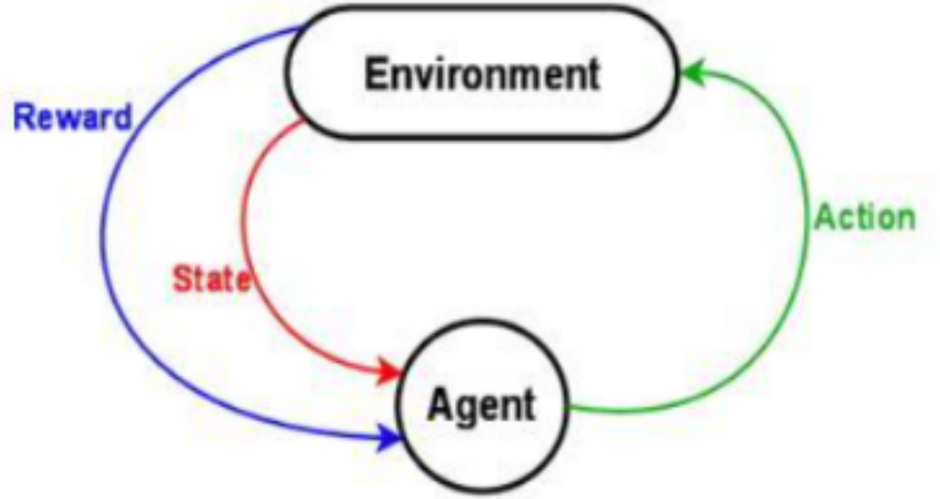
RL Example



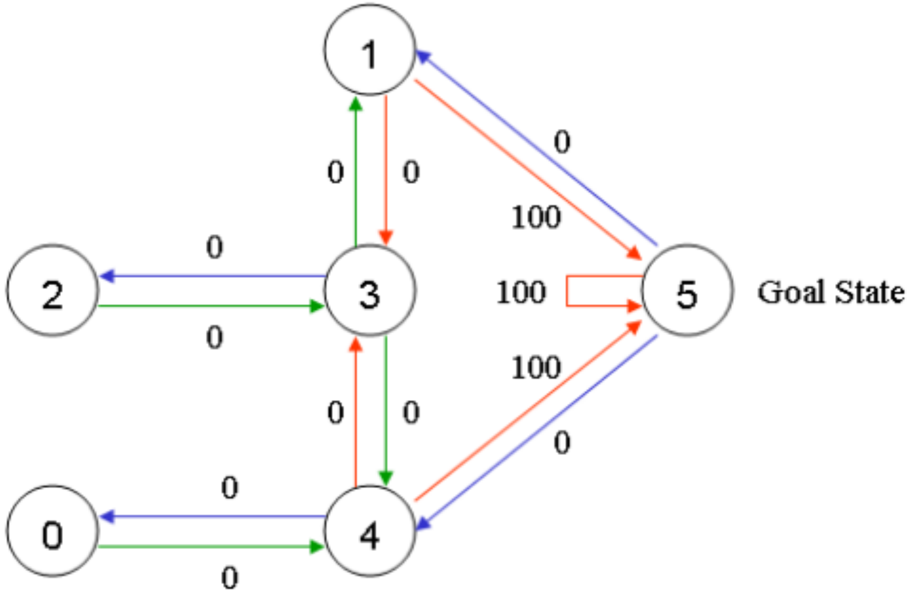
Q-learning for the RL Problem

- Assuming rewards for each step, the goal is to reach the state with the highest reward.
- Terms— state: room, action: move decision, reward: 0 or 100

RL Problem



Markov Decision Process



http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Q-learning for the RL Problem

Reward Table

$$
R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix} \end{array}
$$

Q-value Table

$$
Q = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 80 & 0 \\
0 & 0 & 0 & 64 & 0 & 100 \\
0 & 0 & 0 & 64 & 0 & 0 \\
0 & 80 & 51 & 0 & 80 & 0 \\
64 & 0 & 0 & 64 & 0 & 100 \\
0 & 80 & 0 & 0 & 80 & 100
\end{bmatrix} \end{array}
$$

Q-learning for the RL Problem

- Q-table is the brain of our agent, representing the memory of what the agent has learned through experience.
- The agent starts out knowing nothing, the matrix Q is initialized to zero.
- Simple transition rule of Q learning,

$$Q(\text{state}, \text{action}) = R(\text{state}, \text{action}) + \text{Gamma} * \text{Max}[Q(\text{next state}, \text{all actions})]$$

http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Q-learning for the RL Problem

The Q-Learning algorithm goes as follows:

1. Set the gamma parameter, and environment rewards in matrix R.

2. Initialize matrix Q to zero.

3. For each episode:

   Select a random initial state.

   Do While the goal state hasn't been reached.

   - Select one among all possible actions for the current state.
   - Using this possible action, consider going to the next state.
   - Get maximum Q value for this next state based on all possible actions.
   - Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
   - Set the next state as the current state.

   End Do

   End For

http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Q-learning for the RL Problem

- Example
  - Initial state: room 1, action: move to 5

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(1, 5) = R(1, 5) + 0.8 * Max[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0.8 * 0 = 100

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \Rightarrow Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Q-learning for the RL Problem

- Example
  - Initial state: room 3, action: move to 1

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(3, 1) = 0 + 0.8 * 100 = 80

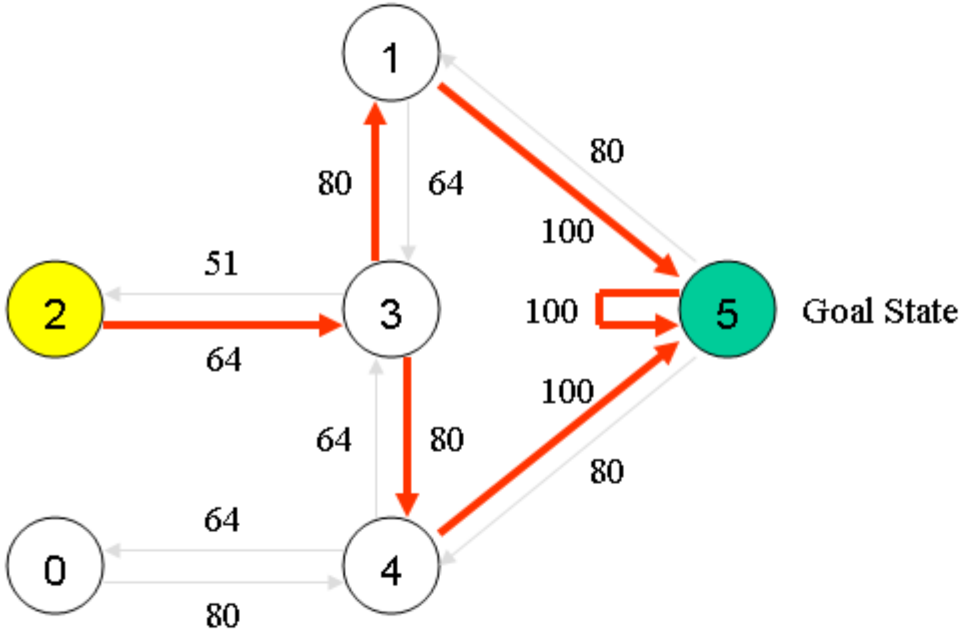
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Q-learning for the RL Problem

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

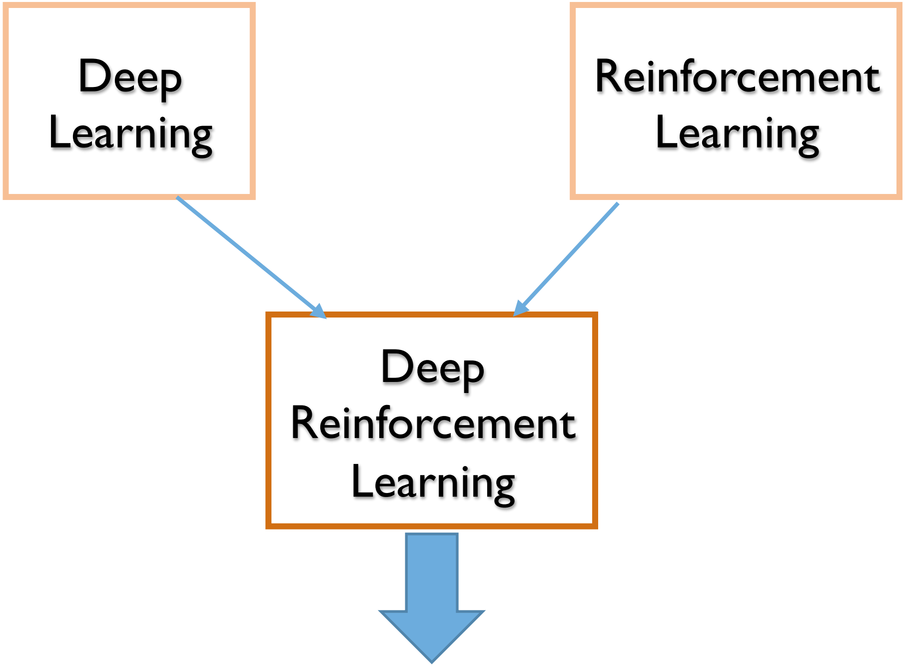


http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Outline

- Background
- Deep Learning
- Reinforcement Learning
- Deep Reinforcement Learning
- Conclusions

Deep Reinforcement Learning



- Use deep network to represent value function / policy / model
- Optimise value function / policy /model end-to-end
- Using stochastic gradient descent

http://videolectures.net/rldm2015_silver_reinforcement_learning/

Deep Q-learning

LETTER

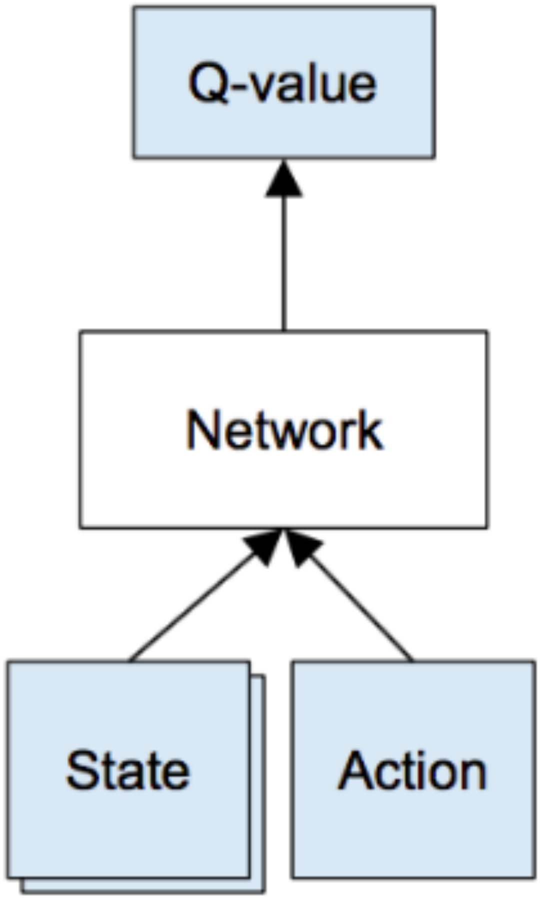
doi:10.1038/nature14236

Human-level control through deep reinforcement learning

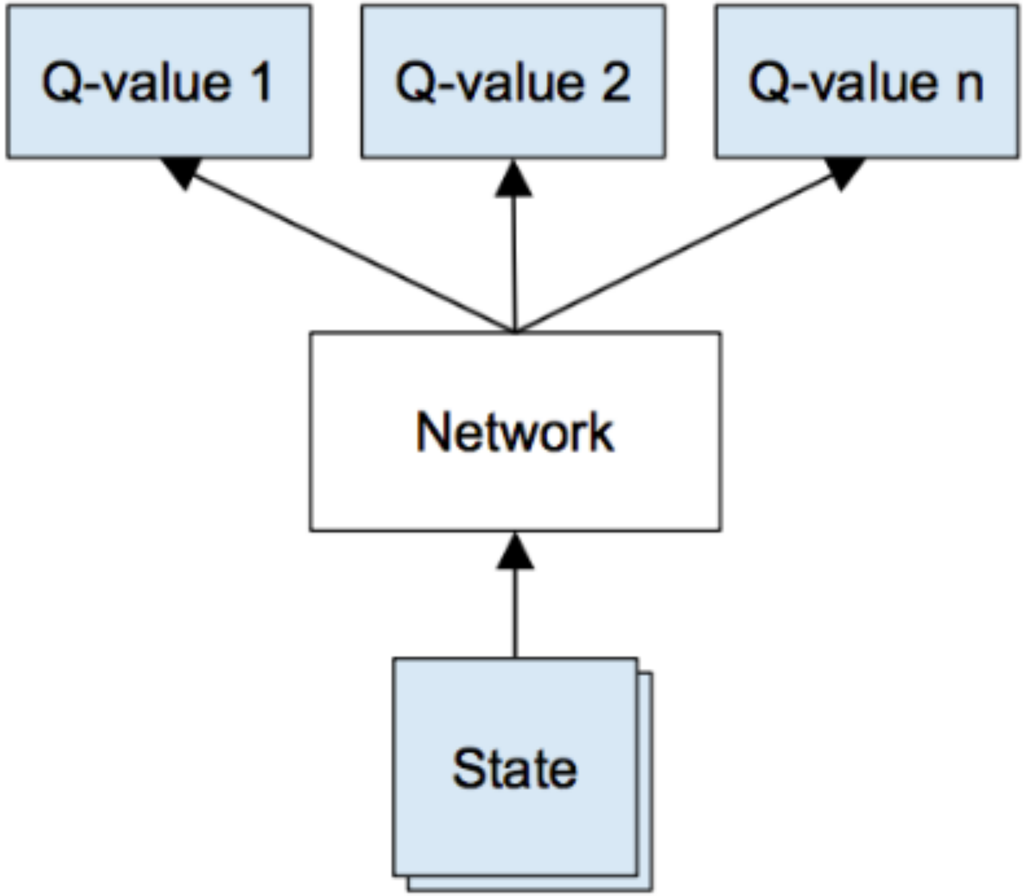
Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Deep Q-Network (DQN)

DQN Architecture



Naive formulation of deep Q-network



DQN in DeepMind paper

https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

DQN Architecture

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

DQN

Loss function

$$L = \frac{1}{2}[\underbrace{r + max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Q-table update algorithm

1. Do a feedforward pass for the current state *s* to get predicted Q-values for all actions.
2. Do a feedforward pass for the next state *s'* and calculate maximum overall network outputs $max_{a'} Q(s', a')$.
3. Set Q-value target for action to $r + \gamma max_{a'} Q(s', a')$ (use the max calculated in step 2). For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.
4. Update the weights using backpropagation.

https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

Exploration-Exploitation

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
      select an action a
            with probability ε select a random action
            otherwise select a = argmax_a′ Q(s,a′)
      carry out action a
      observe reward r and new state s′
      store experience <s, a, r, s′> in replay memory D

      sample random transitions <ss, aa, rr, ss′> from replay memory D
      calculate target for each minibatch transition
            if ss′ is terminal state then tt = rr
            otherwise tt = rr + γmax_a′ Q(ss′, aa′)
      train the Q network using (tt - Q(ss, aa))^2 as loss

      s = s'
until terminated
```

Value Iteration

- Represent value function by deep Q-network with weights w

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

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Policy Iteration

- Represent value function by Q-network with weights w

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

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Sarsa gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be large
     unstable when backpropagated

http://videolectures.net/rldm2015_silver_reinforcement_learning/

DQN

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
   - Using off-policy Q-learning
2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

http://videolectures.net/rldm2015_silver_reinforcement_learning/

Experience Replay

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

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Optimise MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]$$

http://videolectures.net/rldm2015_silver_reinforcement_learning/

Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters w^-

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

- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

- Periodically update fixed parameters $w^- \leftarrow w$

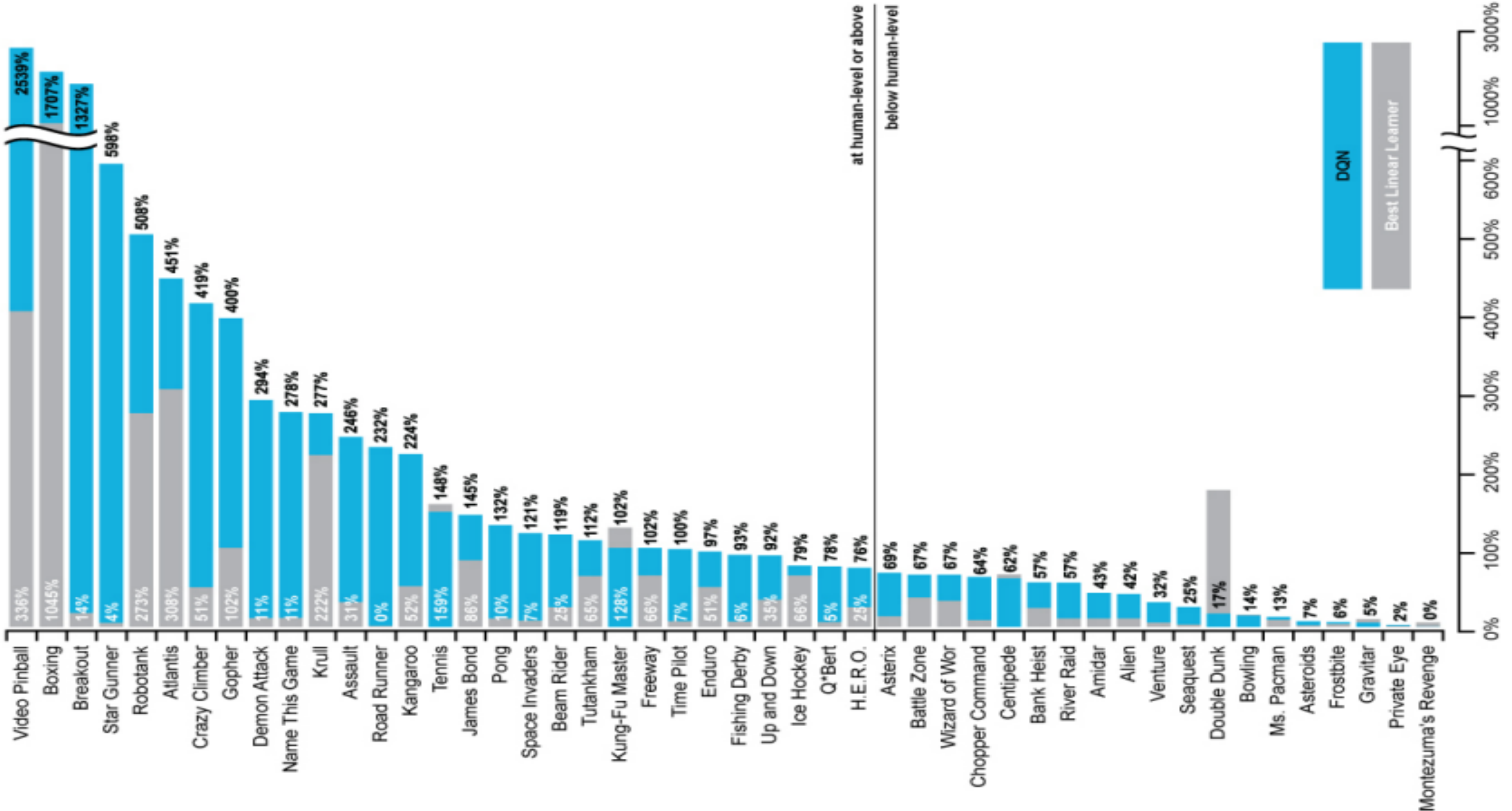
http://videolectures.net/rldm2015_silver_reinforcement_learning/

Reward/Value Range

- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned

http://videolectures.net/rldm2015_silver_reinforcement_learning/

DQN Results in Atari



DQN Atari Demo

DQN paper

www.nature.com/articles/nature14236

DQN source code:

sites.google.com/a/deepmind.com/dqn/

Conclusion

- RL provides a general-purpose framework for AI
- RL problems can be solved by end-to-end deep learning
- A single agent can now solve many challenging tasks
- Reinforcement learning $+$ deep learning $=$ AI

Demo

References

- http://karpathy.github.io/2016/05/31/rl/
- https://gym.openai.com/docs/rl
- https://www.nervanasys.com/demystifying-deep-reinforcement-learning/
- http://mnemstudio.org/path-finding-q-learning-tutorial.htm
- http://videolectures.net/rldm2015_silver_reinforcement_learning/
- http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf