Q-learning

Tambet Matiisen

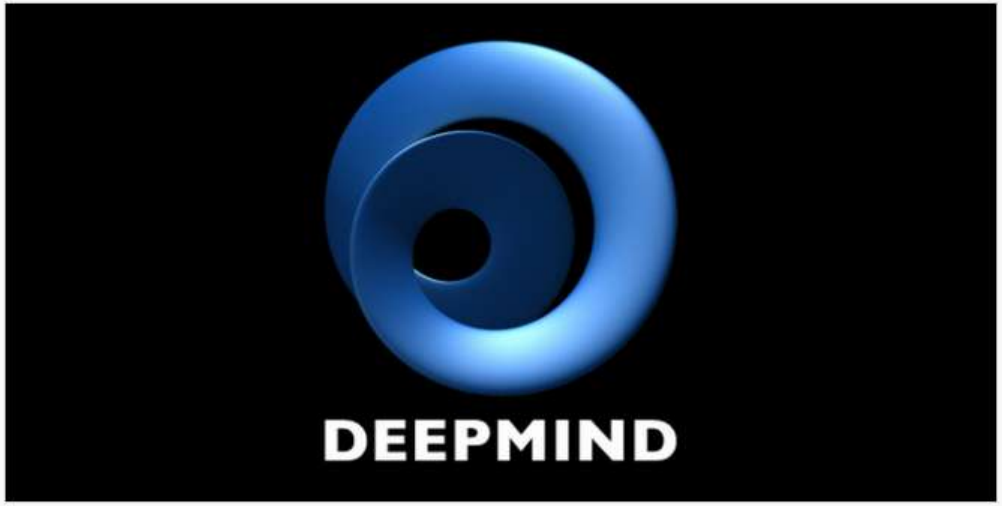
(based on chapter 11.3 of online book “Artificial Intelligence, foundations of computational agents” by David Poole and Alan Mackworth)

Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

Posted Jan 26, 2014 by Catherine Shu (@catherineshu)

74 Like 8.2k Tweet 2,183 Share 667

Next Story



Google will buy London-based artificial intelligence company DeepMind. The Information reports that the acquisition price was more than $500 million, and that Facebook was also in talks to buy the startup late last year. DeepMind confirmed the acquisition to us, but couldn't disclose deal terms.

The acquisition was originally confirmed by Google to Re/code.

CrunchBase

DeepMind

FOUNDED	TOTAL FUNDING
N/A	Not available

OVERVIEW

DeepMind is a cutting edge artificial intelligence

Stochastic gradient descent

Experience replay mechanism

Convolutional neural networks

Multilayer perceptrons

Deep learning

Restricted Boltzmann machines

Q-learning

Reinforcement learning

Temporal difference learning

Markov Decision Processes

Arcarde Learning Environment

Reinforcement learning

- ... is an area of machine learning inspired by **behaviorist** psychology, concerned with how to take actions in an environment so as to maximize cumulative **reward**.

- The agent learns “on go”, it has no prior knowledge of environment or which actions result in rewards.

- The agent must still be “hardwired” to recognize the reward.

Markov Decision Processes

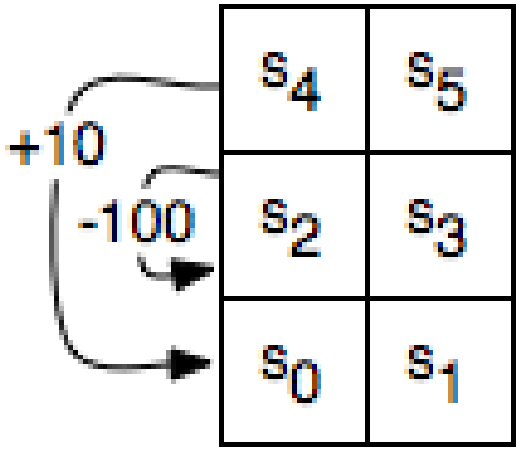
- *S* is a finite set of states.
- *A* is a finite set of actions.
- MDP: $<s_0,a_0,r_1,s_1,a_1,r_2,s_2,a_2,r_3,s_3,a_3,r_4,s_4...>$

- *P(s,a,s')* is the probability, that action *a* in state *s* will lead to state *s'*.
- *R(s,a,s')* is the immediate reward received after transition to state *s'* from state *s* with action *a*.

- In reinforcement learning *P(s,a,s')* and *R(s,a,s')* are unknown to the agent.

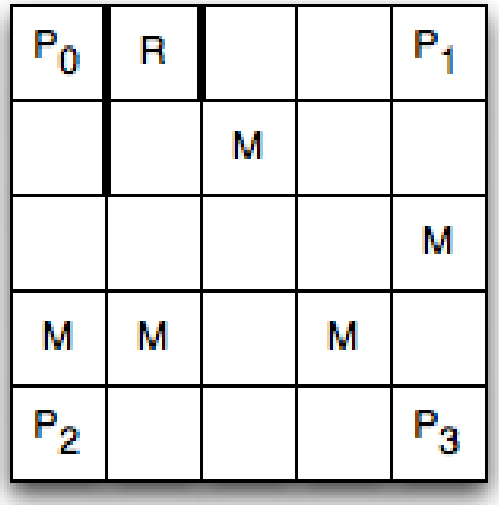
Example 1

- States: s_0, s_1, s_2, s_3, s_4, s_5
- Actions:
  - *right*
  - *left*
  - *up* (0.8 up, 0.1 left, 0.1 right)
  - *upC* ("up carefully", always up, but costs reward -1)
- Additional rewards:
  - Out of bounds -1



Example 2

- State: *<X, Y, P, D>*
  - *X*, *Y* – location of the agent
  - *P* – location of the price (0-4)
  - *D* – if agent is damaged
- Actions: *left*, *right*, *up*, *down*
  - not reliable
- Rewards:
  - Price +10
  - Out of bounds -1
  - Monster bites damaged agent - 10



Why is it hard?

- Blame attribution problem

- Explore-exploit dilemma

Temporal Differences

- $A_k=(v_1+...+v_k)/k$
- $kA_k = v_1+...+v_{k-1}+v_k = (k\text{-}1)\,A_{k-1} + v_k$
- $A_k = (1\text{-}1/k)A_{k-1} + v_k/k = A_{k-1} + 1/k(v_k - A_{k-1})$
- Let $\alpha_k = 1/k$, then $\boldsymbol{A_k = A_{k-1} + \alpha_k(v_k - A_{k-1})}$
- $v_k - A_{k-1}$ is called **temporal difference error**
- If α is a constant ($0 < \alpha \leq 1$) that does not depend on k, then latter values are weighted more.
- Convergence quaranteed if $\sum_{k=1..\infty}\alpha_k = \infty$ and $\sum_{k=1..\infty}(\alpha_k)^2 < \infty$

Discounted reward

- History: $<s_0,a_0,r_1,s_1,a_1,r_2,s_2,a_2,r_3,s_3,a_3,r_4,s_4...>$
- Total reward:
  $R = \sum_{t=1..\infty}\gamma^{t-1}r_t = r_1 + \gamma r_2 + \gamma^2 r_3 + ... =$
  $= r_1 + \gamma(r_2 + \gamma(r_3 + ...))$
- γ is the **discount factor** ($0 \leq \gamma < 1$), that makes the future rewards worth less than current.
- Total reward from time point t onward:
  $\boldsymbol{R_t = r_t + \gamma R_{t+1}}$

Q-learning definition

- ***Q*(s,a)*** is the expected value (**cumulative discounted reward**) of doing *a* in state *s* and then following the optimal policy.
- Q-learning uses **temporal differences** to estimate the value of *Q*(s,a)*.
- The agent maintains a table of *Q[S, A]*, where *S* is the set of states and *A* is the set of actions.
- *Q[s, a]* represents its current estimate of *Q*(s,a)*.

Q-learning update rule

- Experience <*s,a,r,s'*> provides a new data point to estimate $Q^*(s,a)$: $r + \gamma \max_{a'} Q^*(s',a')$. This is called **return**.

- The agent can use temporal difference equation to update its estimate for $Q^*(s,a)$:

$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$$

$$A_k = A_{k-1} + \alpha_k(v_k - A_{k-1}) \qquad R_t = r_t + \gamma R_{t+1}$$

Q-learning algorithm

controller *Q-learning*(*S*,*A*,*γ*,*α*)

2: **Inputs**
3: *S* is a set of states
4: *A* is a set of actions
5: *γ* the discount
6: *α* is the step size
7: **Local**
8: real array *Q[S,A]*
9: previous state *s*
10: previous action *a*
11: initialize *Q[S,A]* arbitrarily
12: observe current state *s*
13: **repeat**
14: select and carry out an action *a*
15: observe reward *r* and state *s'*
16: $Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
17: $s \leftarrow s'$ **until** termination

Q-learning example

s	a	r	s'	Update
s_0	*upC*	*-1*	s_2	$Q[s_0,upC]=-0.2$
s_2	*up*	0	s_4	$Q[s_2,up]=0$
s_4	*left*	10	s_0	$Q[s_4,left]=2.0$
s_0	*upC*	*-1*	s_2	$Q[s_0,upC]=-0.36$
s_2	*up*	0	s_4	$Q[s_2,up]=0.36$
s_4	*left*	10	s_0	$Q[s_4,left]=3.6$
s_0	*up*	*0*	s_2	$Q[s_0,upC]=0.06$
s_2	*up*	*-100*	s_2	$Q[s_2,up]=-19.65$
s_2	*up*	0	s_4	$Q[s_2,up]=-15.07$
s_4	*left*	10	s_0	$Q[s_4,left]=4.89$

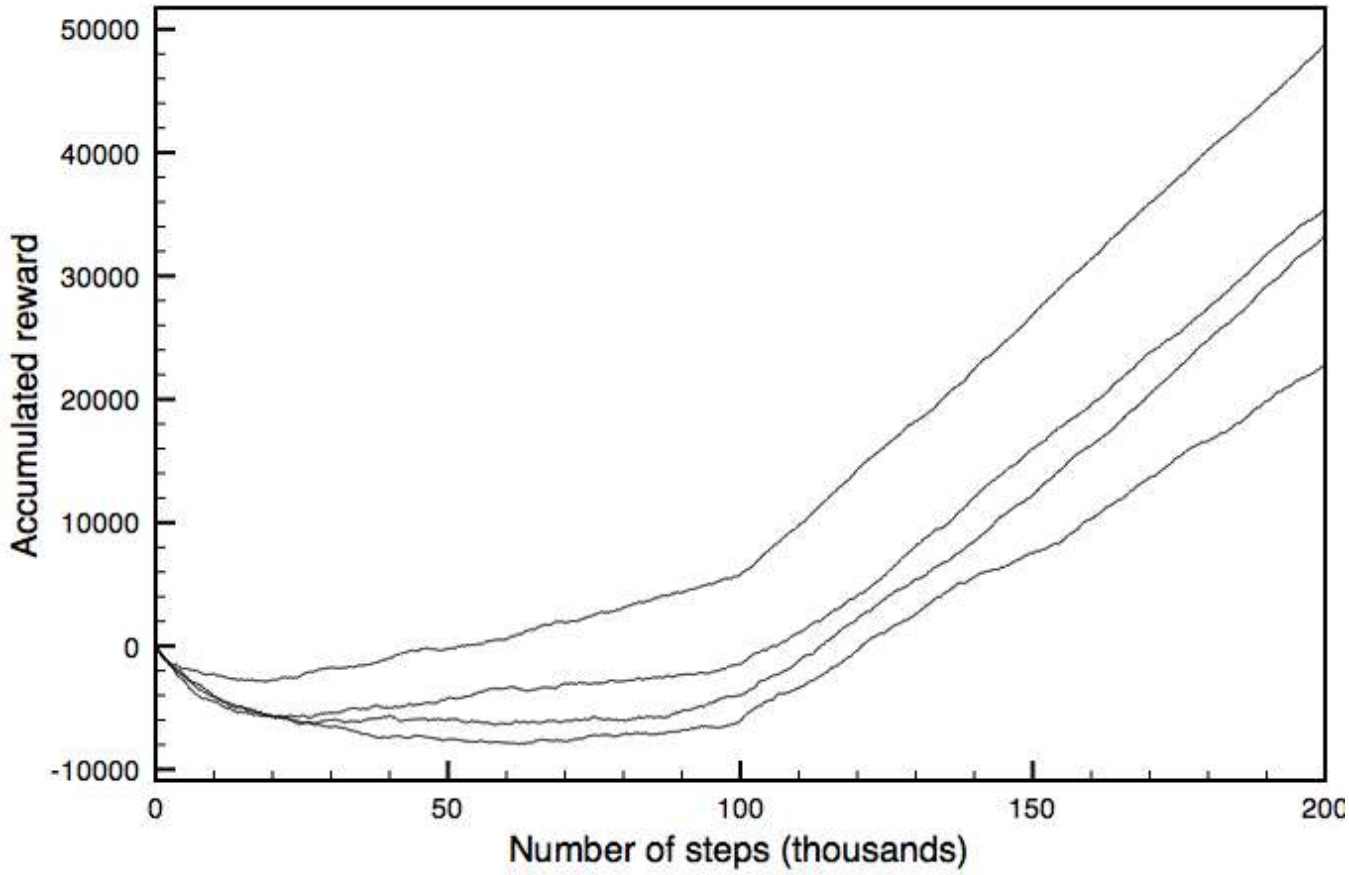
$\gamma=0.9$ and $\alpha=0.2$

Demo time!!!

Exploration vs Exploitation

- ***ε*-greedy strategy**:
  - choose random action with ε probability
  - otherwise choose action with maximum *Q[s,a]*
- **soft-max action selection**:
  - choose action *a* with a probability depending on the value of *Q[s,a]*, e.g. using Boltzmann distribution: $(e^{Q[s,a]/T})/(\sum_a e^{Q[s,a]/T})$, where *T* is the **temperature** specifying how randomly values should be chosen.
- **"optimism in the face of uncertainty"**:
  - initialize the *Q*-function to values that encourage exploration.

Evaluation of Reinforcement Learning Algorithms



Off-policy vs On-policy

- An **off-policy learner** learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy learner.
- An **on-policy learner** learns the value of the policy being carried out by the agent, including the exploration steps.
- It will find a policy that is optimal, taking into account the exploration inherent in the policy.

On-policy example: SARSA

- SARSA experience: *<s,a,r,s',a'>*
  - *a'* – next action chosen by policy based on *Q*
- SARSA update rule:
  Q[s,a]* ← *Q[s,a] + α(r + γQ[s',a'] - Q[s,a])
- SARSA is useful when you want to optimize the value of an agent that is exploring. If you want to do offline learning, and then use that policy in an agent that does not explore, Q-learning may be more appropriate.

Assigning Credit and Blame to Paths

- In Q-learning and SARSA, only the previous state-action pair has its value revised when a reward is received.
- How to update path:
  - **look-ahead or backup** – update *n* steps ahead or back.
  - **model-based methods** – the agent learns *P(s,a,s')* and *R(s,a,s'),* uses them to update other *Q*-values.

Reinforcement Learning with Features

- Usually, there are too many states to reason about explicitly. The alternative is to reason in terms of **features**.
- Q-function approximations:
  - **linear combination** of features,
  - **decision tree**,
  - **neural network**.
- **Feature engineering** - choosing what features to include - is non-trivial.

DeepMind

- **States:** last 4 images of 84x84, 128 graytones
  - State-space size: $128^{4x84x84} \sim 10^{59473}$
- **Features:** convolutional neural network
  - Input: 4x84x84 image
  - 3 hidden layers
  - Output: Q-value for each action
- **Actions:** 4-18, depending on a game
- **Rewards:** difference in score
  - Hardwired, also terminal states hardwired
- **Learning:** off-policy