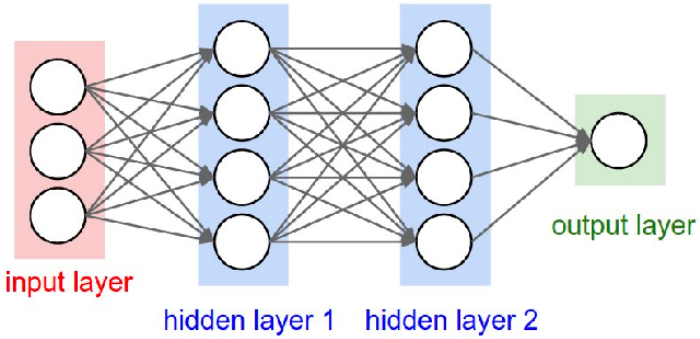


Lecture 6: Q-Network

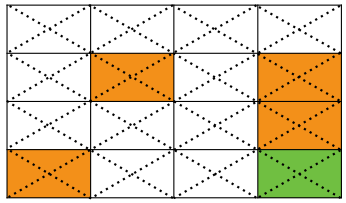
Reinforcement Learning with TensorFlow&OpenAI Gym

Sung Kim <hunkim+ml@gmail.com>

Q-Table (16x4)

(1) state, s

(2) action, a



(3) quality (reward)
for the given action
(eg, LEFT: 0.5, RIGHT 0.1
UP: 0.0, DOWN: 0.8)

Q (s, a)

Q-learning Test

created by Jae Hyun Lee(jaehyunlee25@gmail.com)

STATUS

EPISODE: 0 | MOVE: 0 times | TIME ELAPSED: 0.0 secs | Hole: 0 times | GOAL: 0 times

OPTIONS

SIZE: 4 | VELOCITY: normal | REPEAT: 1000 * 1 (1000) times

Hole: reward -1 penalty go on | Gamma(γ): 0.9 | Explo it

START

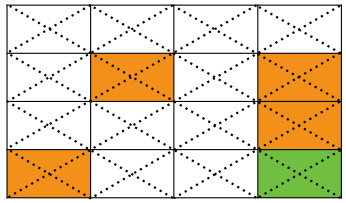
0	0 R=0 0	0	0	0 R=0 0	0	0	0 R=0 0	0	0	0 R=0 0	0
0	0 R=0 0	0	0	0 R=-1 0	0	0	0 R=0 0	0	0	0 R=-1 0	0
0	0 R=0 0	0	0	0 R=0 0	0	0	0 R=0 0	0	0	0 R=-1 0	0
0	0 R=-1 0	0	0	0 R=0 0	0	0	0 R=0 0	0	0	0 R=1 0	0

http://computingkoreanlab.com/app/jAI/jQLearning/

Q-Table (16x4)

(1) state, s

(2) action, a



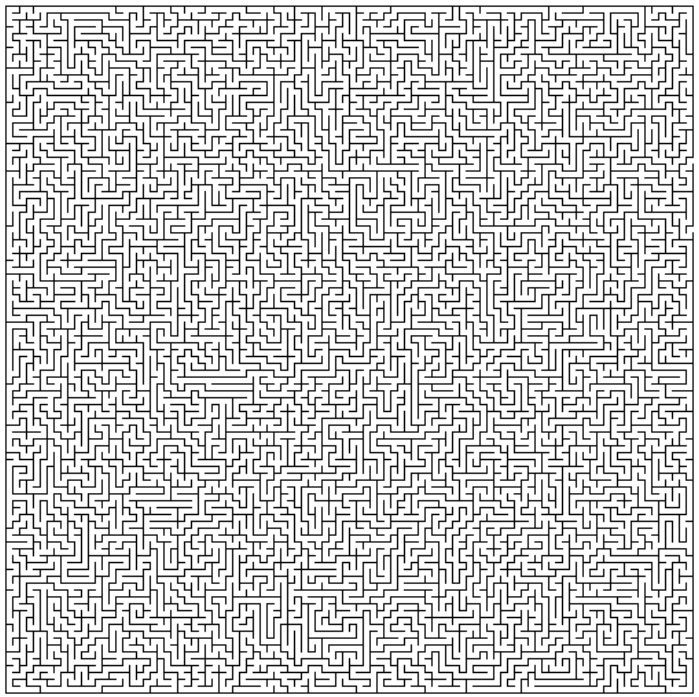
(3) quality (reward)
for the given action
(eg, LEFT: 0.5, RIGHT 0.1
UP: 0.0, DOWN: 0.8)

Q (s, a)

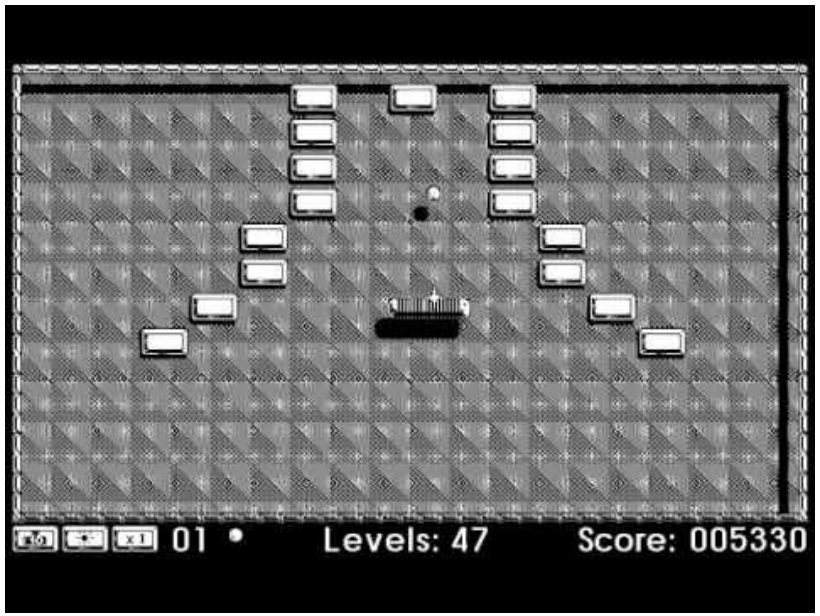
Q-Table (?)

160X160X4

$2^{80 \times 80}$



100x100 maze



80x80 pixel + 2 color (black/white)

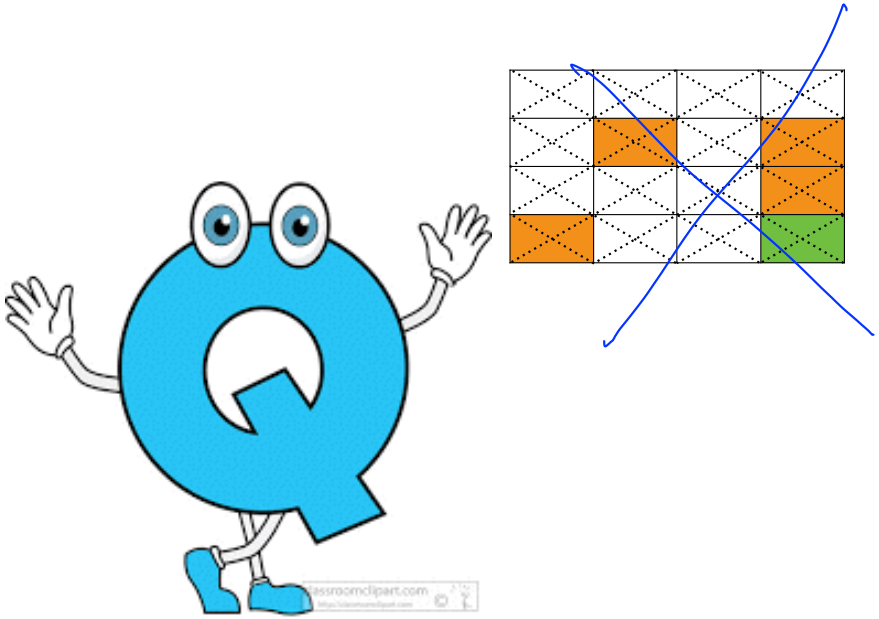
pow(2, 80*80)?

```
Sungs-MacBook-Pro:qlearning hunkim$ python
Python 2.7.10 (default, Jul 30 2016, 19:40:32)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> pow(2,80*80)
39081592266432387331746142836148367311267681070463412272506674720768535568430138381720495886911743307078182434501184483028127299512473215765131666243694265195661755211460607650816799767580417206793005441425788999968221814559073171121585237586191625968426486695394453385378020623210968986095655234800673206169578993893064621394479007844522654750932530260932906944511708573961116842051110072780702996021975530468334392852283568123658054474934577299768778927200578050584569281027959847481492880157513920578725804851336909371096131251468207589952539391748619172204401099253655425443334675779740145331744115766832347511737830030067538031141178372089229186020884093542742187687849517214364478837908382646195523281445267002410663402978244431485772594698434066258955213768118705885537084067810415837310291314293532224816692313560648857707617678657107208778727572115167144969844554782437658117922884282324307657985008269944091087969017270165493533858541923739452891021946640039871139014694517482748438203362034911702773985087028549861930279639065901103098389957112947553195190069994199572871577997339201518554694093470424633134190567138126513775817867770518591303000260334804048037845284233493883534489638498647260587064326521948900609819006765410832117094339235540061356424944564563951032375578180928992076476796598306704845779425650491079898232492863540154374240499247068529525671271000571306646256947045781135744092881405282687148040508264376853441383821755805287956740468541933707091945416554391392625479753506217540329184028243756578451089052788428260969378835272884507542718466673627090471859996737723112934060270454109059566034439294502155993552543831988709035361713548867020994349284913996584689674031362649588710526967617555709716501891685314826079439170843819922088878128902968582950531577390213885399071760414285719601697094720405328129745119056694810317474513400330333517233611193133379954007384384395034058799871873253376L
>>>
```

Q-Table (? x ?)

(1) state, s

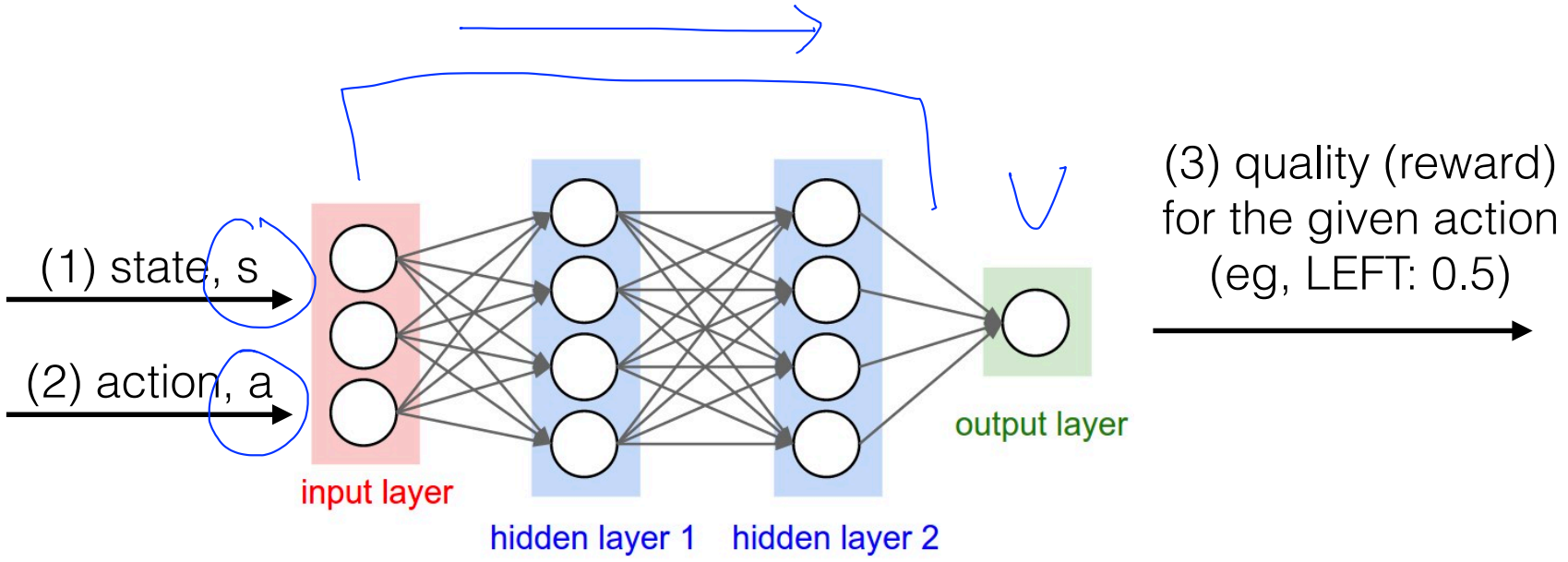
(2) action, a



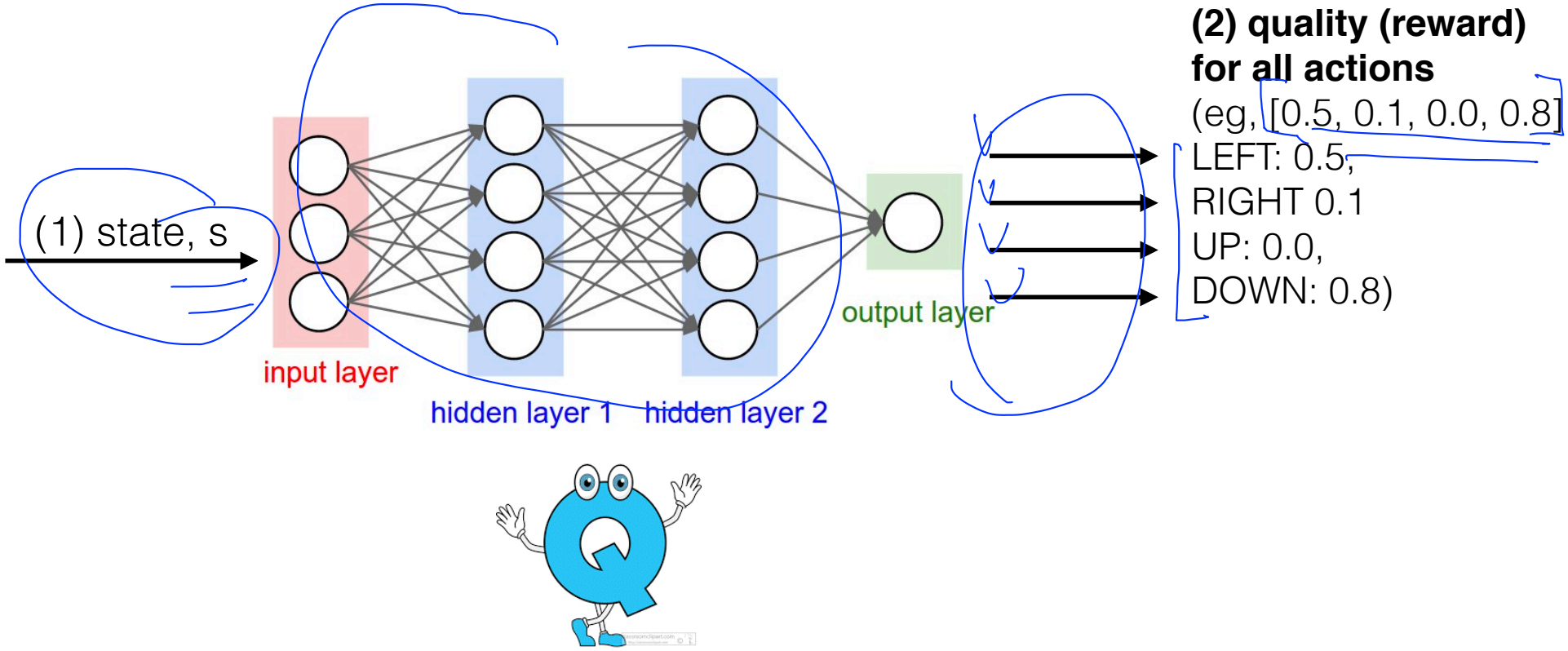
(3) quality (reward) for the given action (eg, LEFT: 0.5)

Q (s, a)

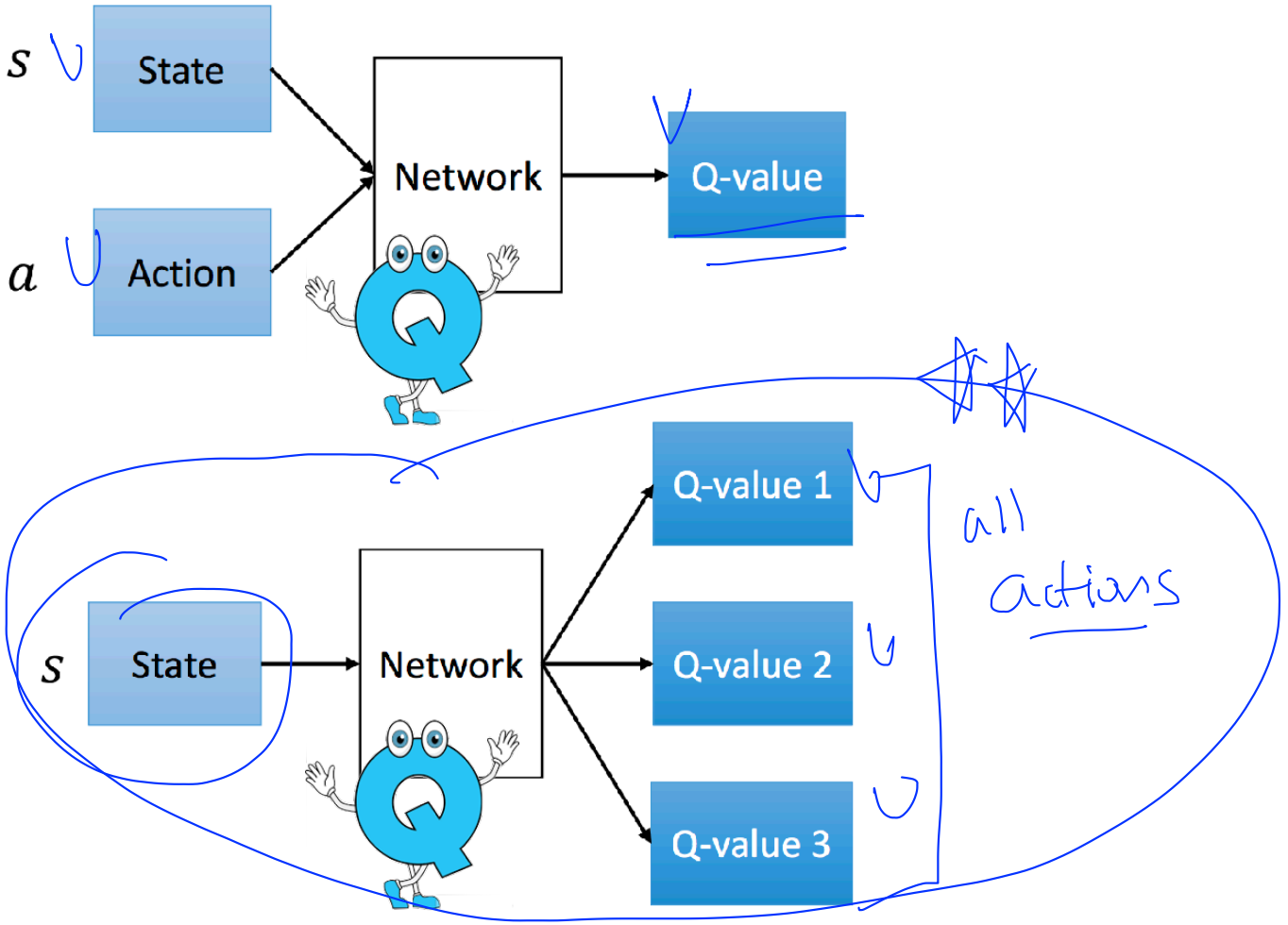
Q-function Approximation



Q-function Approximation



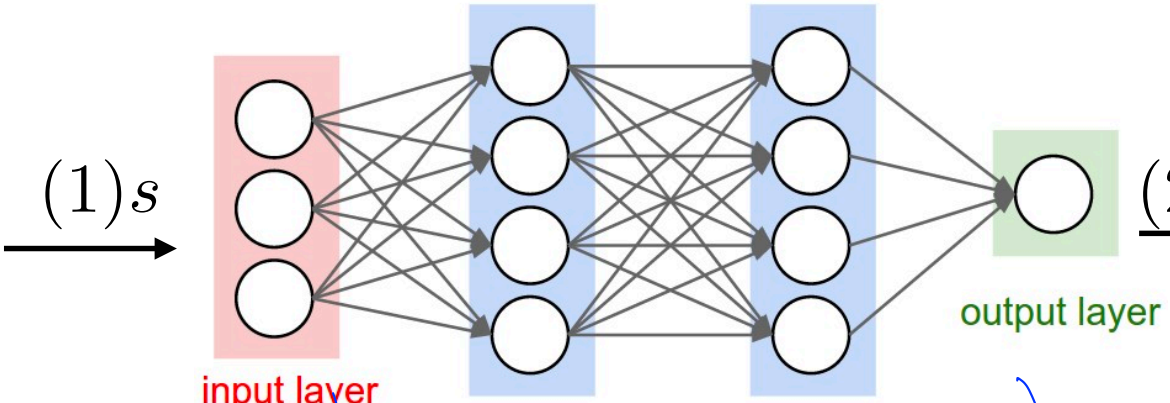
Q-function Approximation



Q-Network training (linear regression)

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

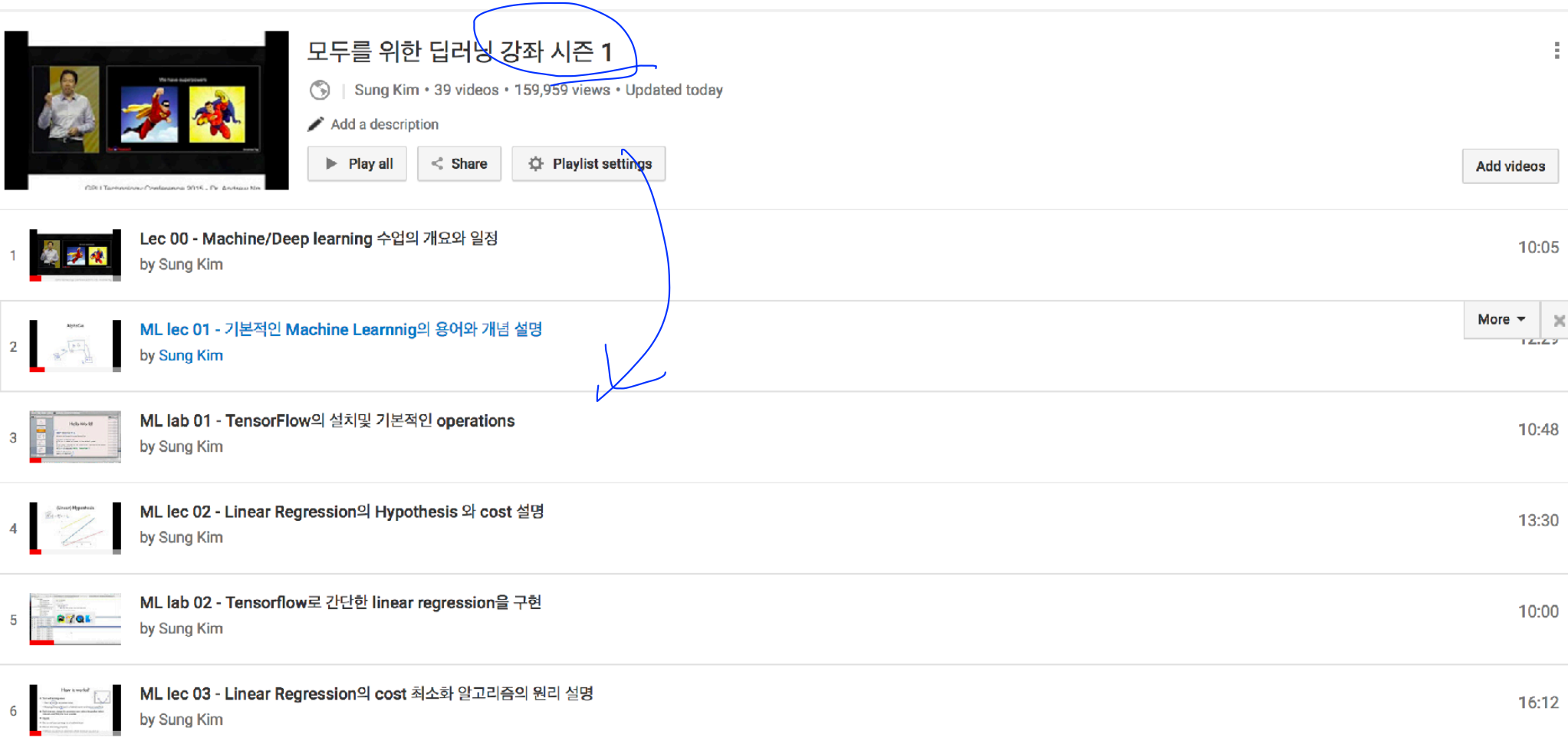


$\sim Q^*$

label

$Q(s)$

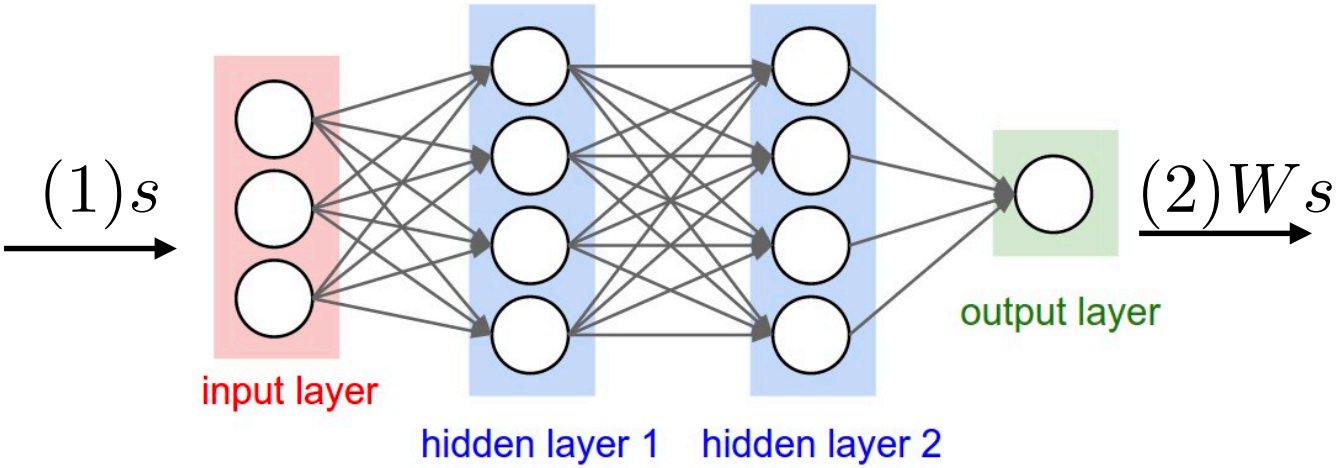
Prerequisite: http://hunkim.github.io/ml/ or https://www.inflearn.com/course/기본적인-머신러닝-딥러닝-강좌/



Q-Network training (linear regression)

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

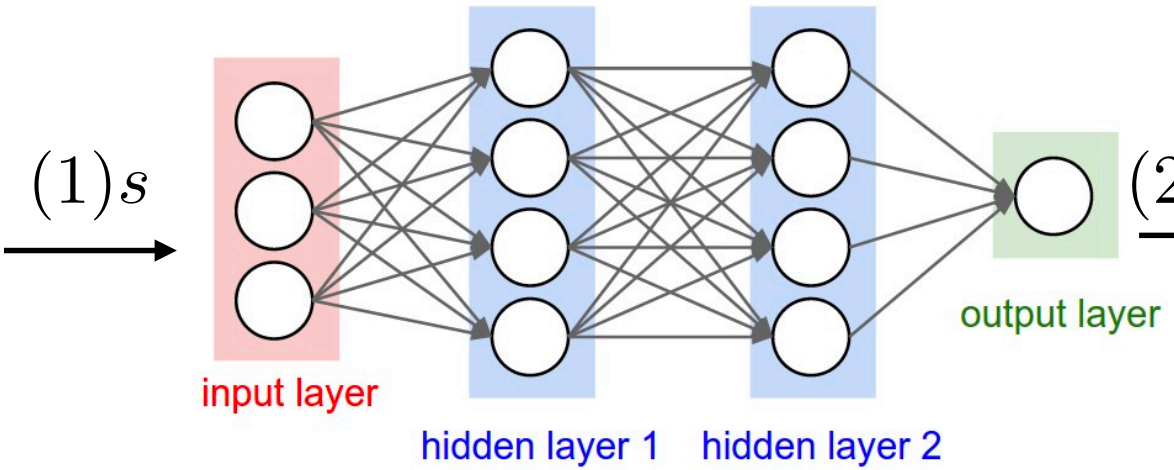


$Q(s)$

Q-Network training (linear regression)

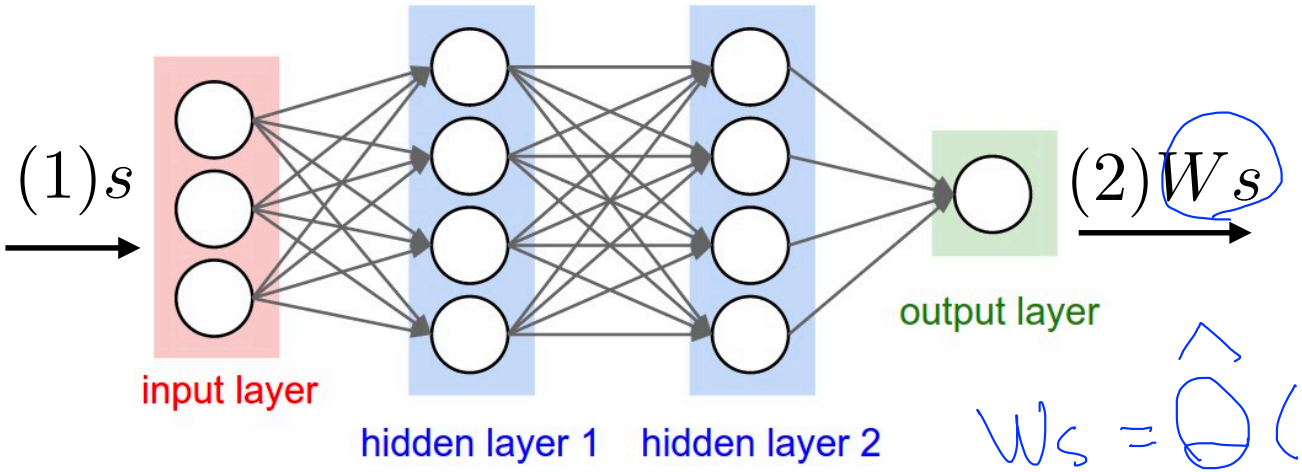
$$cost(W) = (Ws - y)^2$$

$$y = r + \gamma \max Q(s')$$



$Q(s)$

Q-Network training (math notations)

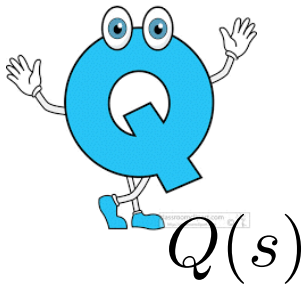


- Approximate Q* function using θ

$$\hat{Q}(s, a|\theta) \sim Q^*(s, a)$$

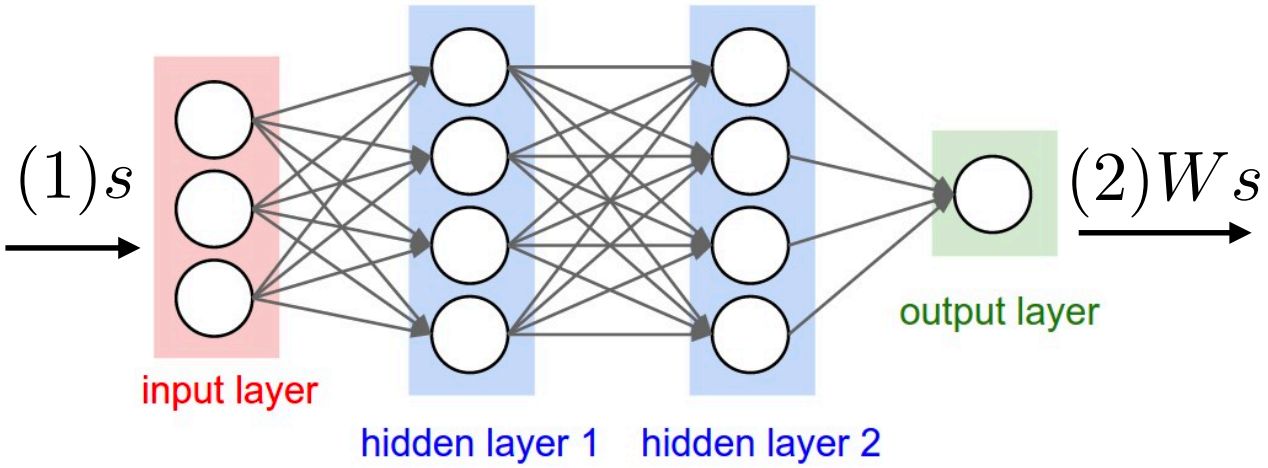
$Ws = \hat{Q}(s, a|\theta)$

weight (Network)



http://introtodeeplearning.com/6.S091DeepReinforcementLearning.pdf

Q-Network training (math notations)

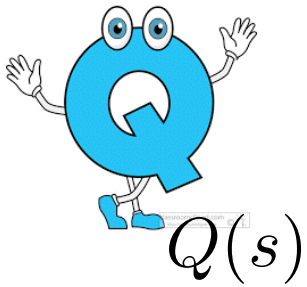


- Approximate Q* function using θ

$$\hat{Q}(s, a|\theta) \sim Q^*(s, a)$$

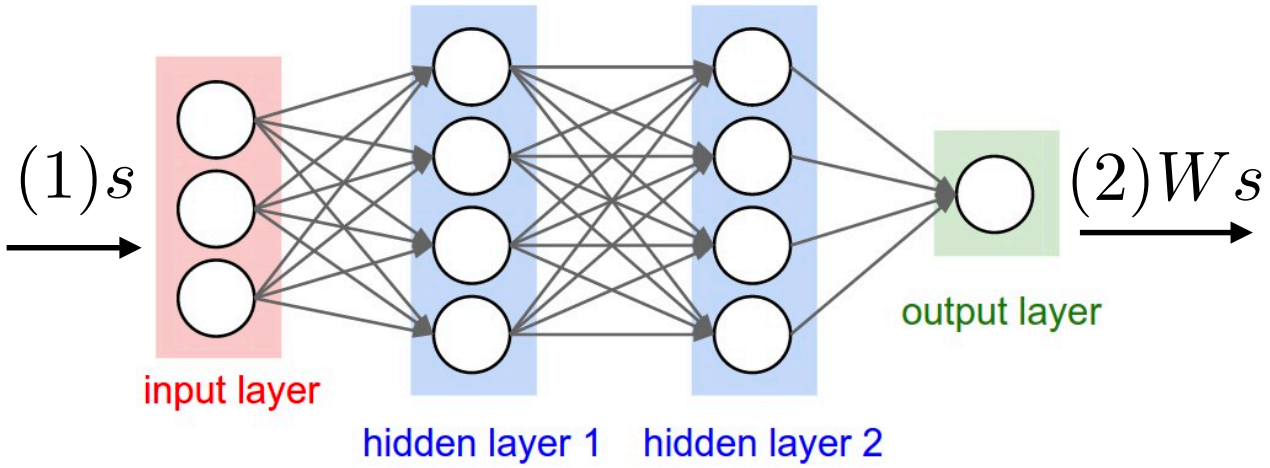
- Choose θ to minimize

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$



http://introtodeeplearning.com/6.S091DeepReinforcementLearning.pdf

Q-Network


$(1)s$
$(2)Ws$
input layer
hidden layer 1
hidden layer 2
output layer


Q

$Q(s)$

Algorithm

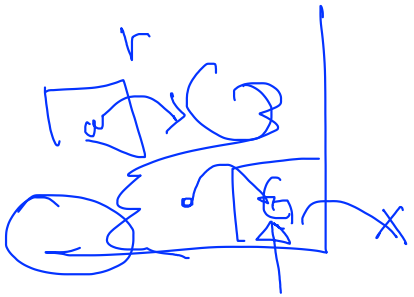
Algorithm 1 Deep Q-learning

Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\phi_{t+1} = \phi(s_{t+1})$

    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Playing Atari with Deep Reinforcement Learning - University of Toronto by V Mnih et al.

Y label and loss function



$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

target/label

Ws

Playing Atari with Deep Reinforcement Learning - University of Toronto by V Mnih et al.

Deterministic or Stochastic?

Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$$

Convergence

$\hat{Q}$ denote learner's current approximation to Q.

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Tutorial: Deep Reinforcement Learning, David Silver, Google DeepMind

Reinforcement + Neural Net

25

There are some research papers on the topic:

- Efficient Reinforcement Learning Through Evolving Neural Network Topologies (2002)
- Reinforcement Learning Using Neural Networks, with Applications to Motor Control
- Reinforcement Learning Neural Network To The Problem Of Autonomous Mobile Robot Obstacle Avoidance

And some code:

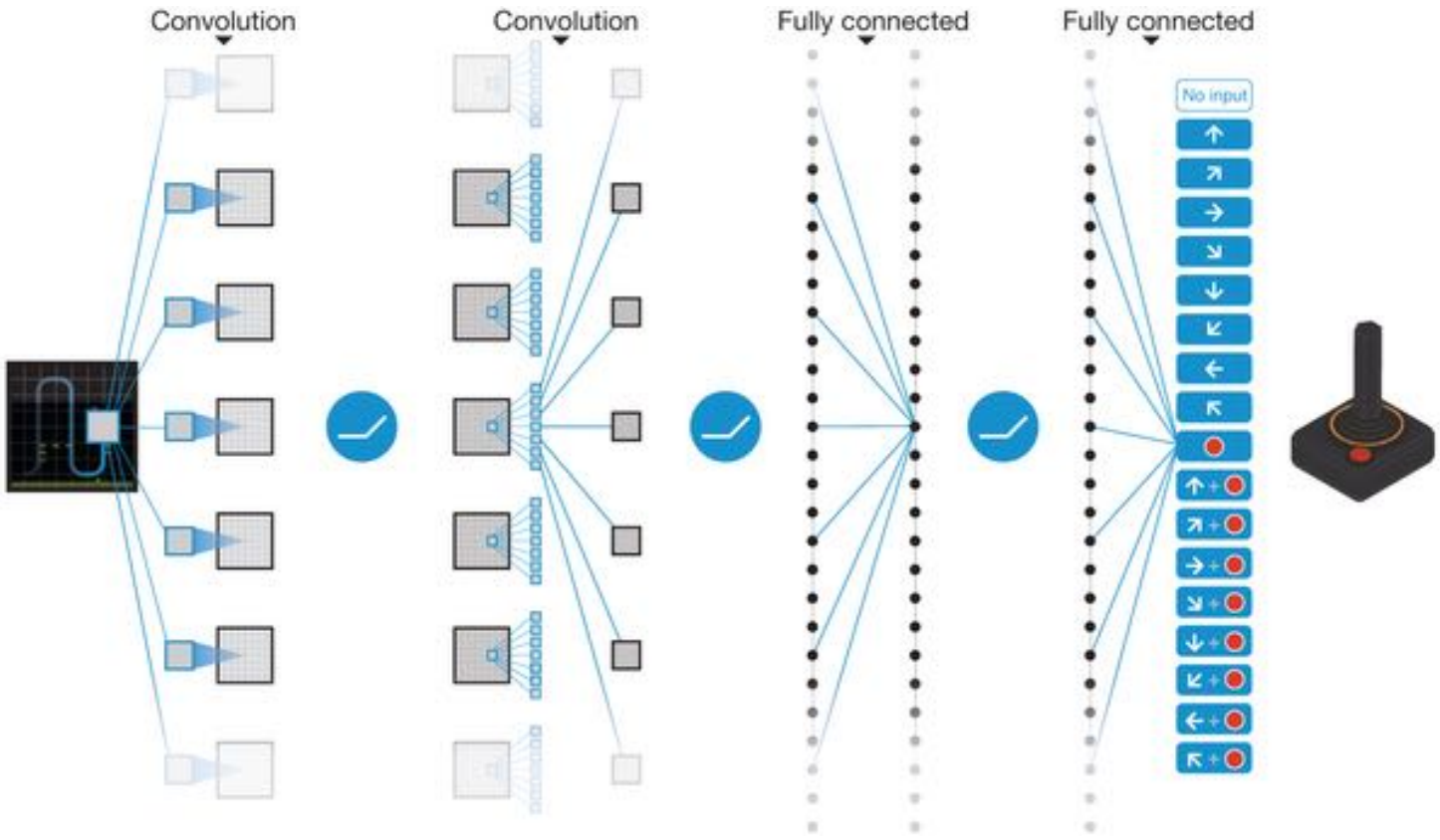
- Code examples for neural network reinforcement learning.

Those are just some of the top google search results on the topic. The first couple of papers look like they're pretty good, although I haven't read them personally. I think you'll find even more information on neural networks with reinforcement learning if you do a quick search on Google Scholar.

- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

http://stackoverflow.com/questions/10722064/training-a-neural-network-with-reinforcement-learning

DQN: Deep, Replay, Separated networks



Next

Lab: Q-network