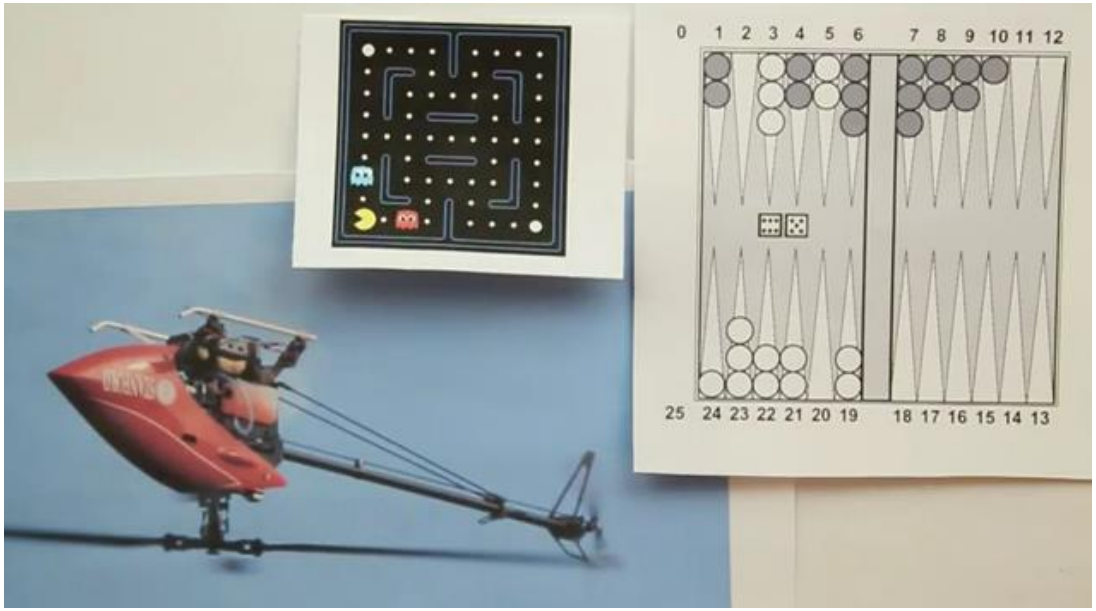
Computational Intelligence
Unit 11-3 - Qlearning



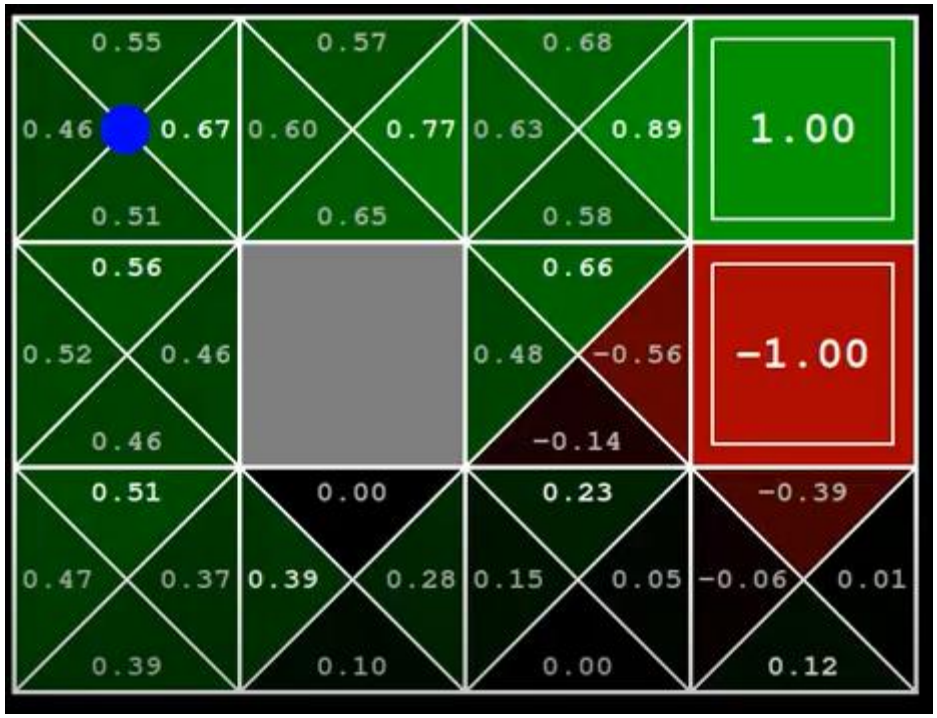
Acknowledgement

- Several examples of this lecture have been taken from Stanford AI class and Stanford Machine Learning class.

QLearning

- Q-learning is a reinforcement learning algorithm that does not need a model of its environment and can be used online. Q-learning algorithms work by estimating the values of state-action pairs.
- The agent can perform adaptively in a world without understanding it. All it tries to do is sort out good actions to perform from bad ones.

QLearning



Instead of having ‘value’ for each state. We now have a value for each state, action (s,a) pair.

QLearning

- The value Q(s, a) is defined to be the expected discounted sum of future payoffs obtained by taking action a from state s and following the current optimal policy thereafter. Once these values have been learned, the optimal action from any state is the one with the highest Q-value.

QLearning

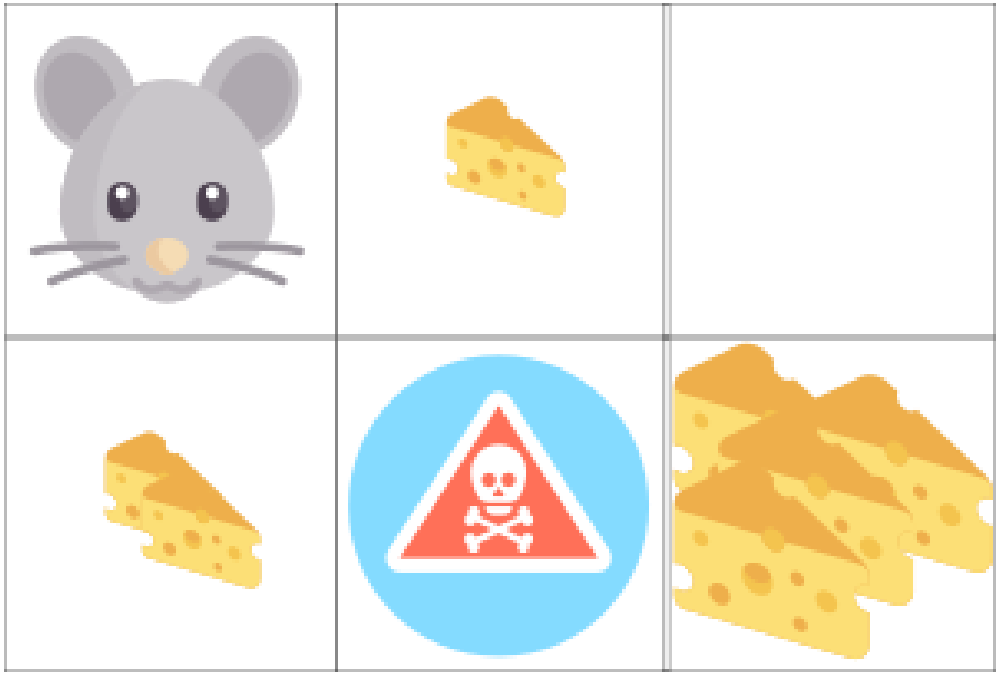
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Figure 6.12: Q-learning: An off-policy TD control algorithm.

Video

- https://www.youtube.com/watch?v=AMnW-OsOcl8

Making to cheese?



https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/

Q-table

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

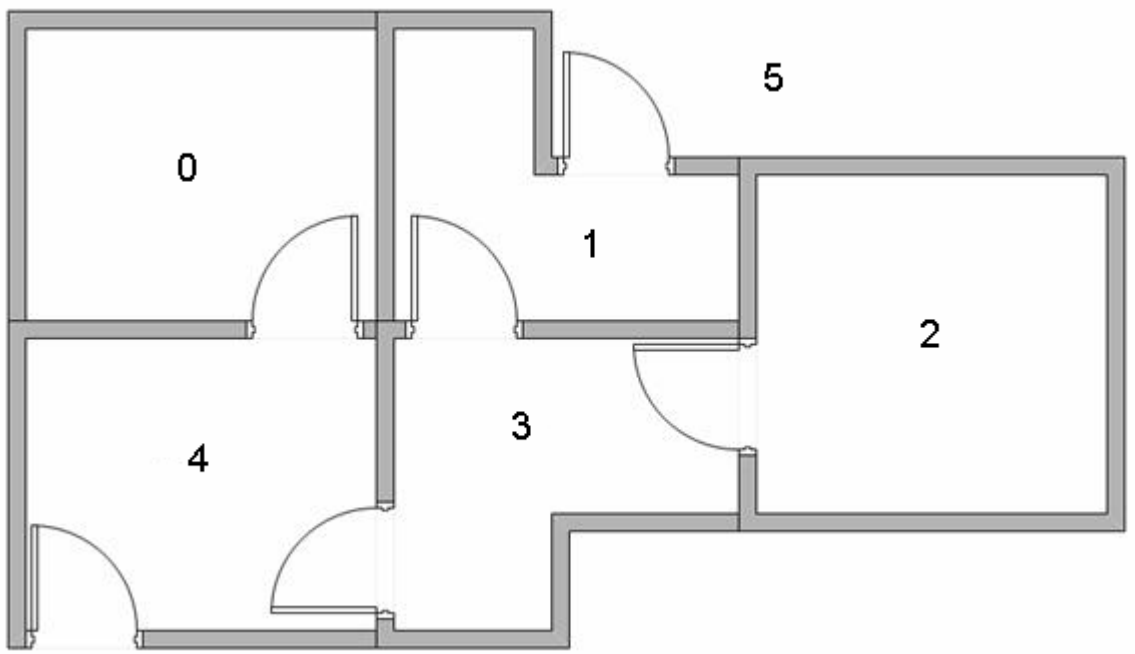
https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/

QLearning

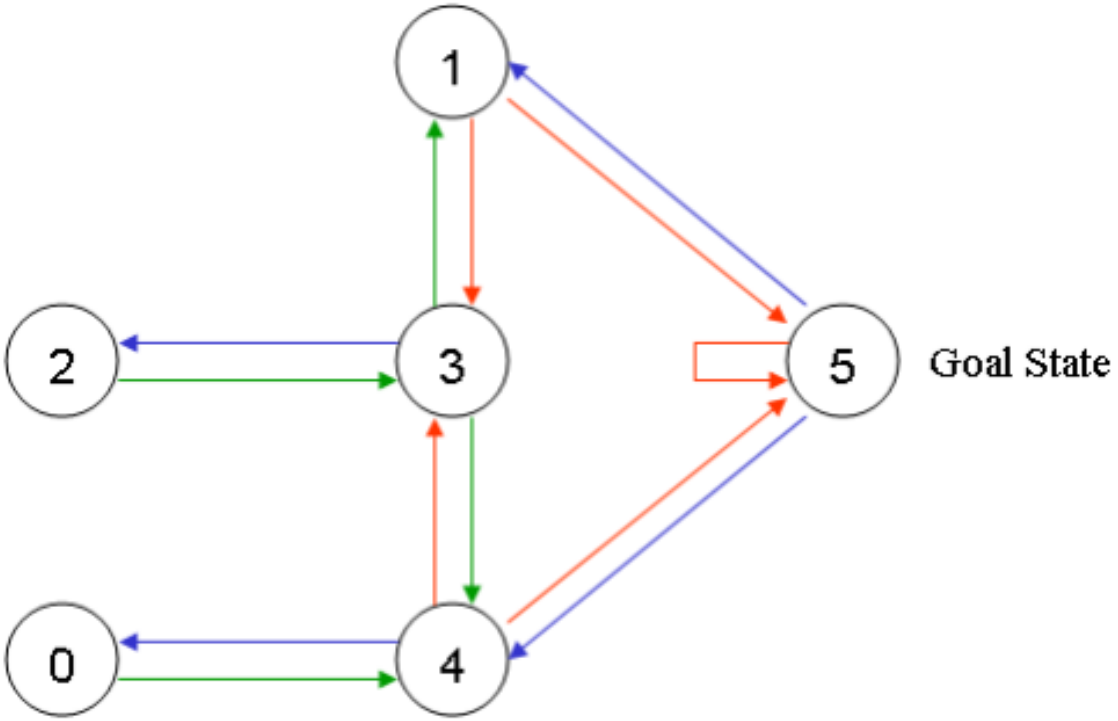
- Q-learning is *model-free*(calls this *primitive learning*). The agent can perform adaptively in a world without understanding it. All it tries to do is sort out good actions to perform from bad ones.

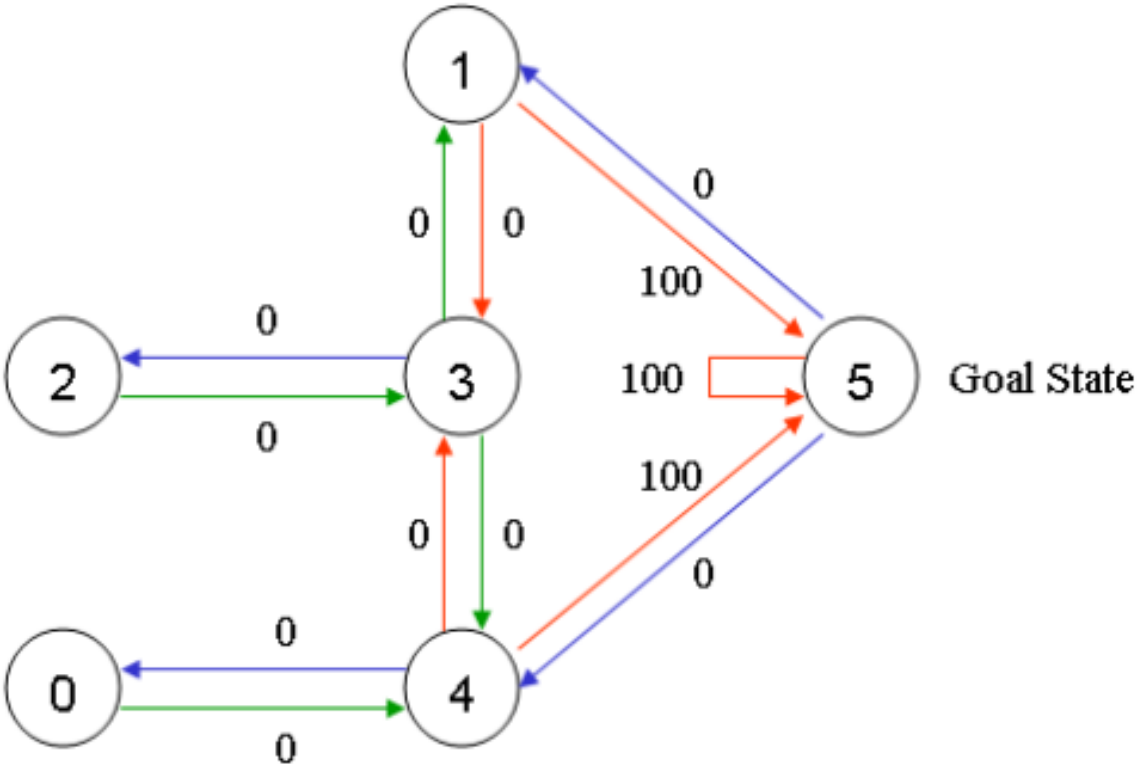
Example

- we want to model some kind of simple evacuation of an agent from any room in the building. Now suppose we have an agent in Room 2 and we want the agent to learn to reach outside the house (5)



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


1
2
3
5
Goal State
0
4


1
2
3
4
5
0
Goal State
0
0
0
100
0
100
100
0
0
0
0
0

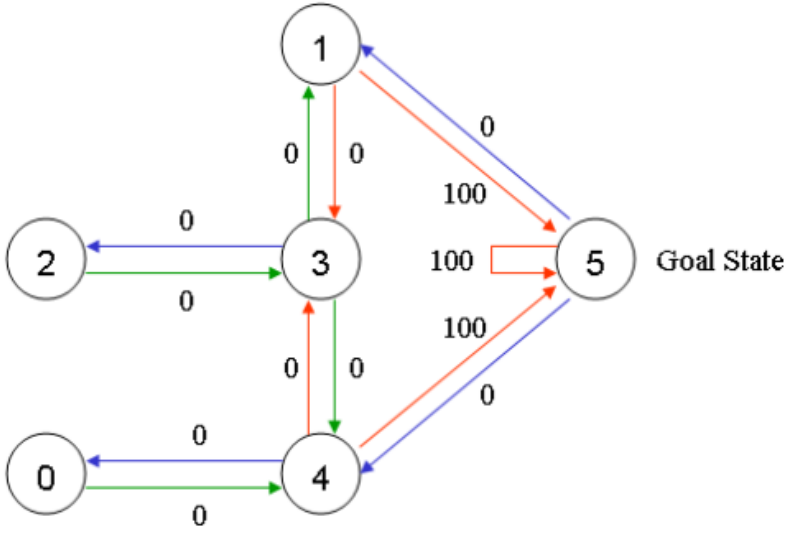
$$
R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
\end{array}
$$

Q Values

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

Updating QValue

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



- Being in state '3', choosing to perform action '1'

  Q(state, action) = R(state, action) +

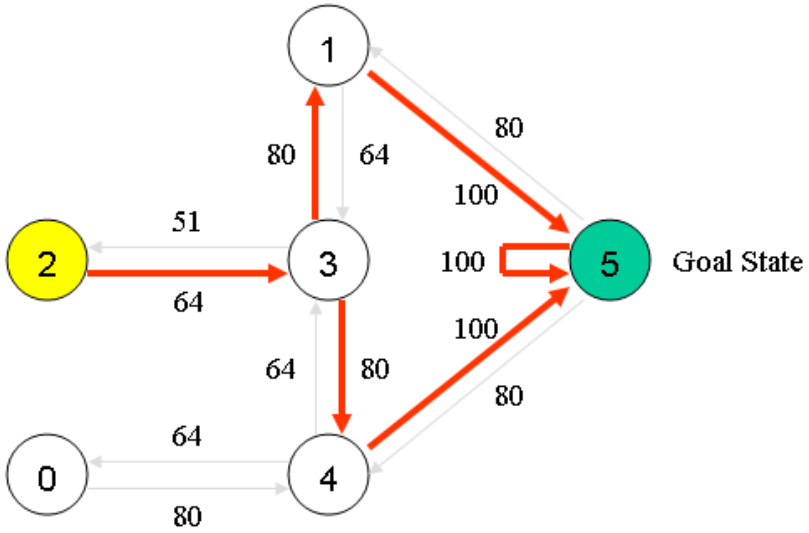
  Gamma * Max[Q(next state, all actions)]

  Q(3,1) = R(3,1) + 0.8 * Max[Q(1, 3), Q(1, 5)] = 0 + 0.8 * Max(0, 100) = 80

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

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

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



Choosing an action

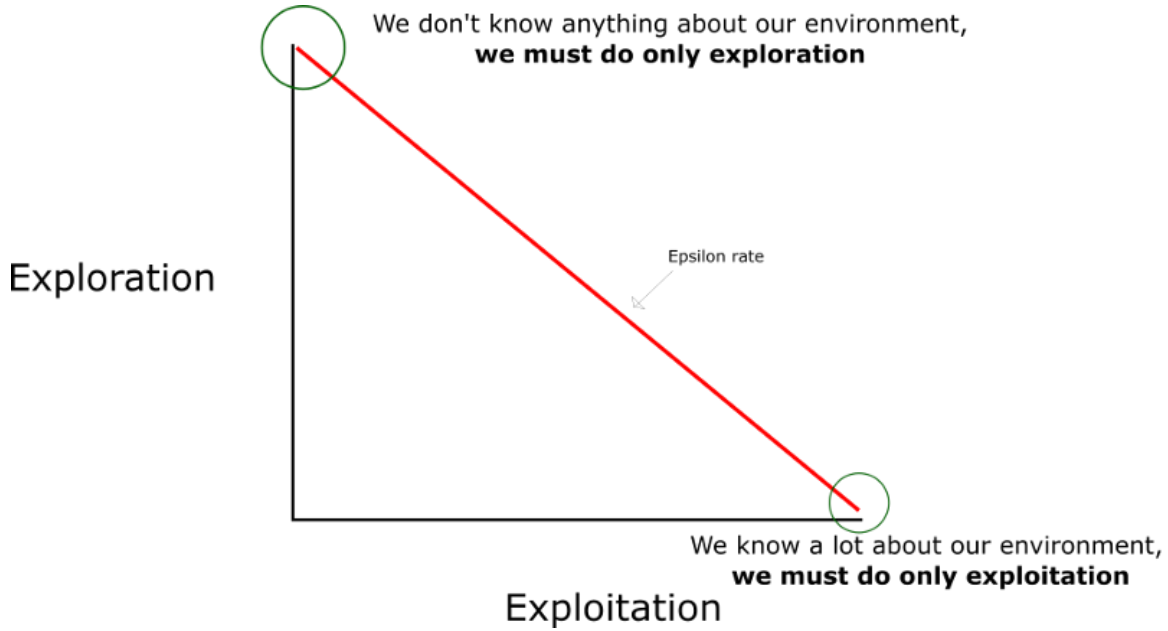
- how does the agent select actions during learning.
  - We will model this with a function that assigns a probability of being chosen for each possible action in a given state.
  - This function should tend to choose actions with higher Q values, but should sometimes select lower Q-value actions.
  - The probability of selecting the highest Q-value action should increase over time.

Action Selection Policies

- ε-greedy:
  - With probability 1- epsilon – we choose action with maximum value
  - With probability epsilon – we randomly choose an action from a set of all actions A

ε-greedy

- The idea is that we must have a big epsilon at the beginning of the training of the Q-function. Then, reduce it progressively as the agent becomes more confident at estimating Q-values.



Action Selection - Boltzman distribution

- One drawback of ε-greedy is that when it explores it chooses equally among all actions. This means that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action.
- In tasks where the worst actions are very bad, this may be unsatisfactory.
- The obvious solution is to vary the action probabilities as a graded function of estimated value.

$$P(a|s) = \frac{e^{Q(s,a)/k}}{\sum_j e^{Q(s,a_j)/k}}$$

- The k parameter (often referred to as temperature) controls the probability of selecting non-optimal actions. If k is large, all actions will be selected fairly uniformly. If k is close to zero, the best action will always be chosen. We begin with k large and gradually decrease it over time.

Dealing with continuous environment

Function Approximation

Q-Table

$Q(s, a) \rightarrow Q(3, 1) \rightarrow$

	s^0	s^1	s^2	s^3	s^4
a^0	+4.21	+4.88	+5.74	+6.25	+8.51
a^1	+3.72	+4.02	+4.48	+5.13	+5.22

$\rightarrow$ +5.13

Function Approximation

Q-Table

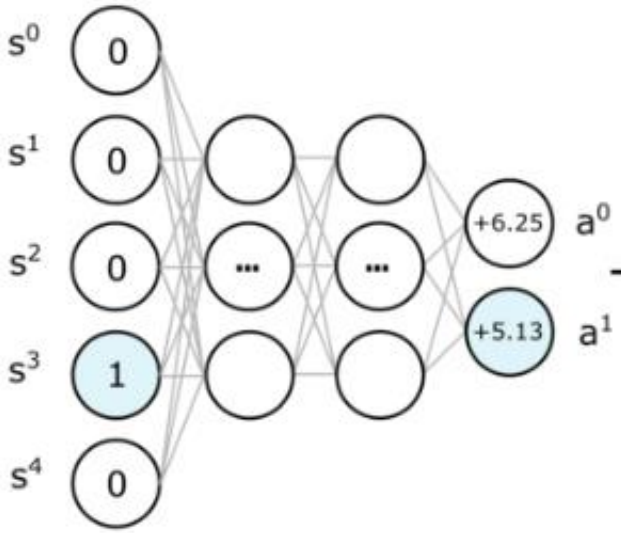
Q(s, a) → Q(3, 1) →

	s^0	s^1	s^2	s^3	s^4
a^0	+4.21	+4.88	+5.74	+6.25	+8.51
a^1	+3.72	+4.02	+4.48	+5.13	+5.22

→ +5.13

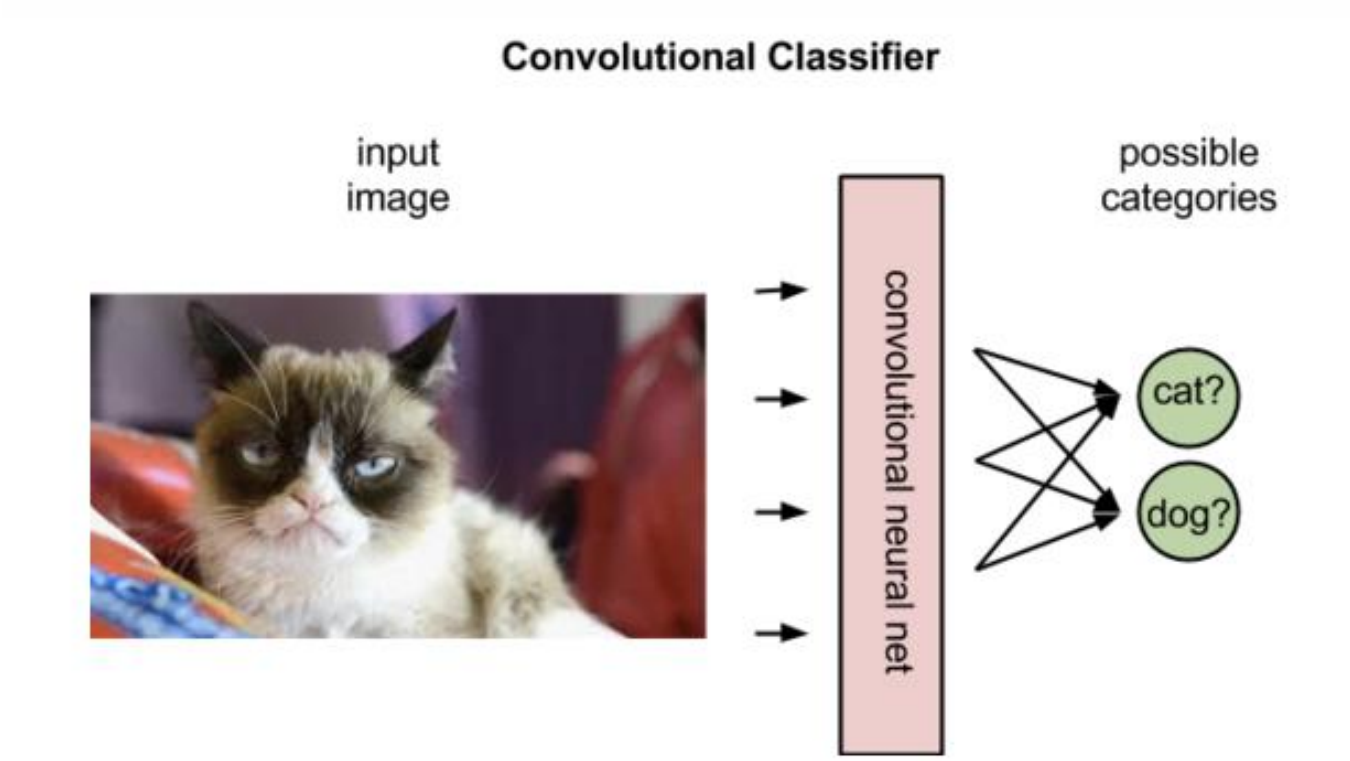
Neural net

Q(s, a) → Q(3, 1) →



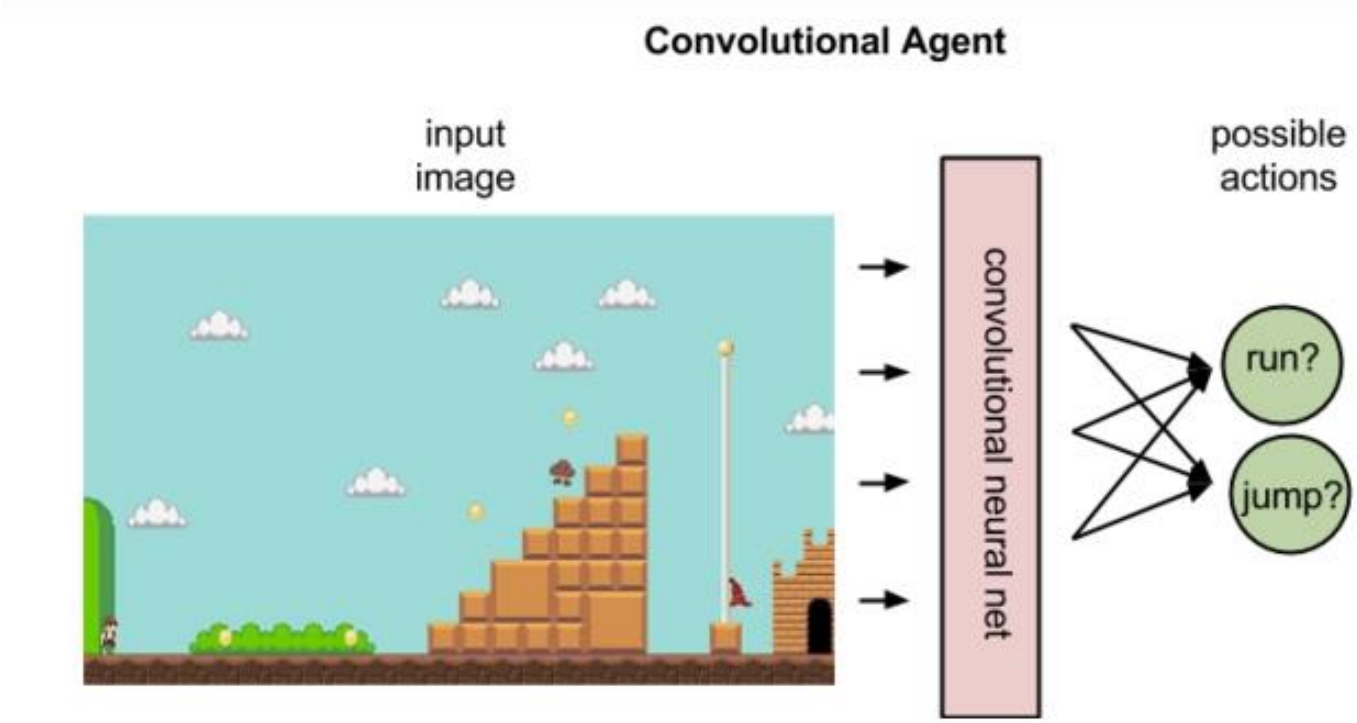
→ +5.13

Neural Networks in classification



https://pathmind.com/wiki/deep-reinforcement-learning

ANN in Qlearning



https://pathmind.com/wiki/deep-reinforcement-learning

Calculating Error

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{max\,Q'(s',a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - Q(s,a)]$$

Loss

$$loss = \left(\mathrm{r} + \gamma \max_{a`} \hat{Q}(s, a`) - Q(s, a) \right)^2$$

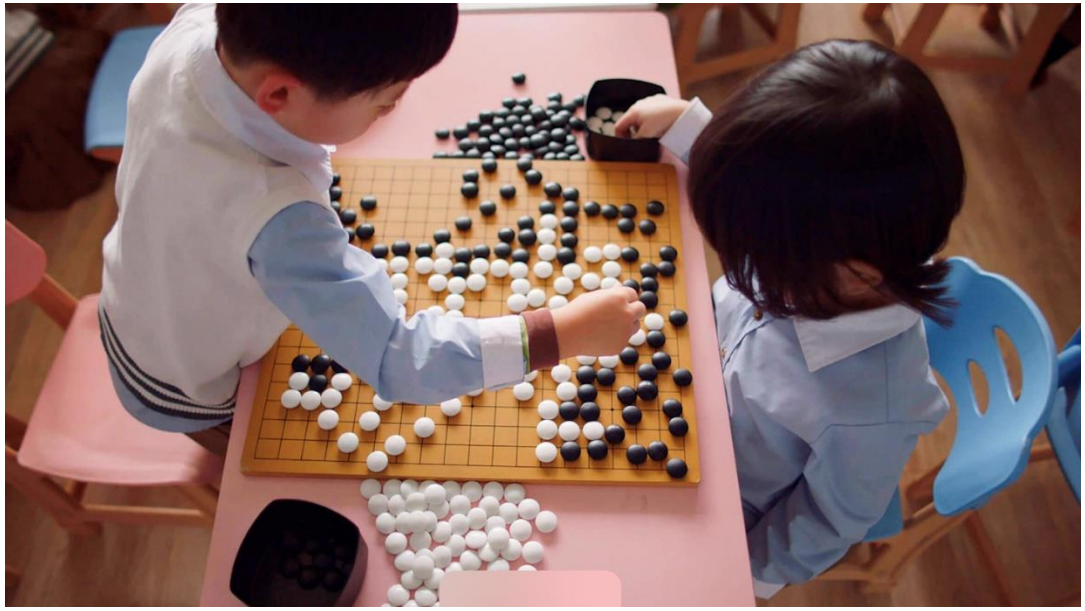
https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a

Deep Reinforcement Learning

- A Q-Learning example (huggingface.co)
- https://medium.freecodecamp.org/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8

AlphaGo

- Go is profoundly complex. There are an astonishing 10 to the power of 170 possible board configurations - more than the number of atoms in the known universe. This makes the game of Go a googol times more complex than chess.



AlphaGo

- AlphaGo competed against legendary Go player Mr Lee Sedol, the winner of 18 world titles, who is widely considered the greatest player of the past decade. AlphaGo's 4-1 victory in Seoul, South Korea, on March 2016 was watched by over 200 million people worldwide.

While AlphaGo learnt the game by playing thousands of matches with amateur and professional players, AlphaGo Zero learnt by playing against itself, starting from completely random play.

Some Videos

- https://www.youtube.com/watch?v=e3Jy2vShroE
- http://www.youtube.com/watch?v=Xf_IhCbTQGY&feature=BFa&list=FLTnVTxTVQSBDkVSWPhBpRTQ

References

- R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- http://reinforcementlearning.ai-depot.com/Main.html
- https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc
- http://amunategui.github.io/reinforcement-learning/
- http://mnemstudio.org/path-finding-q-learning-tutorial.htm
- https://www.cs.swarthmore.edu/~bryce/cs63/s16/slides/3-21_value_iteration.pdf