INTRODUCTION TO Q-LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group

REFERENCES

The material for this talk is primarily drawn from the slides, notes and lectures of these courses with occasional reference to Sutton and Barto's book.

"Demystifying Deep Reinforcement Learning," 2015, Tambet Matiisen

- https://www.intel.ai/demystifying-deep-reinforcement-learning/

CS188 course at University of California, Berkeley:

- *CS188 – Introduction to Artificial Intelligence*, Profs. Dan Klein, Pieter Abbeel, et al. http://ai.berkeley.edu/home.html

CS181 course at Harvard University:

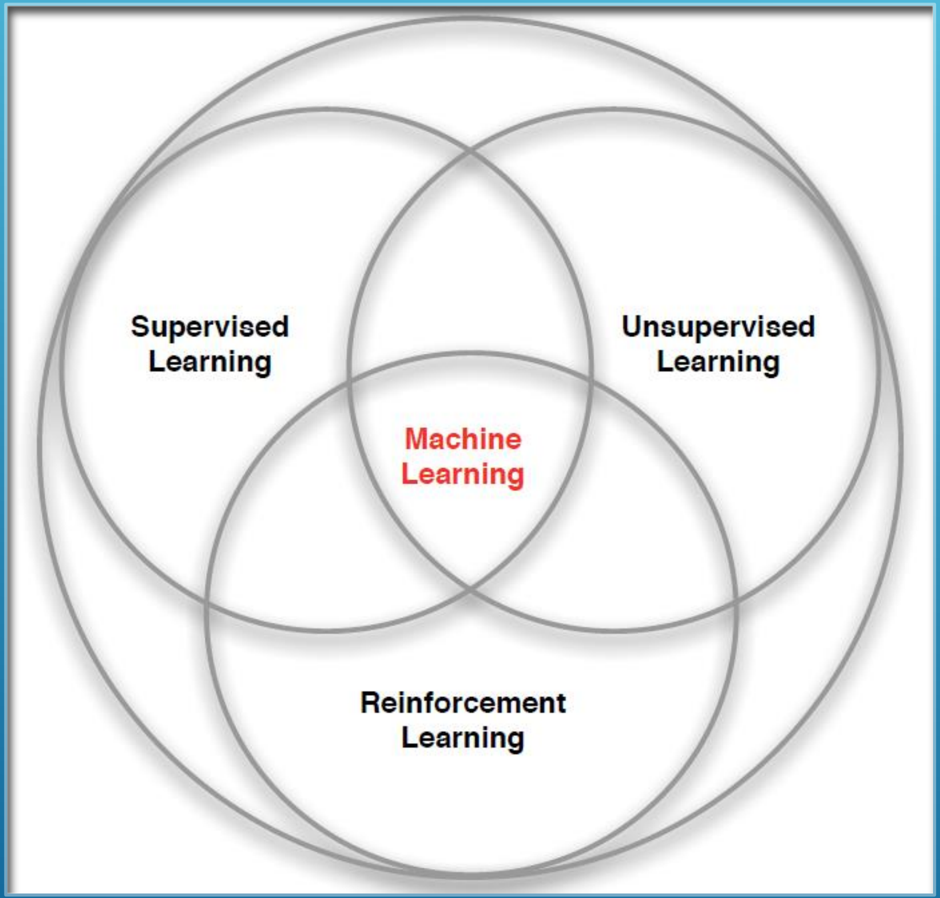
- *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Sarah Finney, Spring 2009
- *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Prof. David C Brooks, Spring 2011
- *CS181 – Machine Learning*, Prof. Ryan P. Adams, Spring 2014. https://github.com/wihl/cs181-spring2014
- *CS181 – Machine Learning*, Prof. David Parkes, Spring 2017. https://harvard-ml-courses.github.io/cs181-web-2017/

Reinforcement learning: an introduction R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

UC BERKELEY CS188 IS A GREAT RESOURCE!

- Websites:
  - http://ai.berkeley.edu/home.html
  - http://gamescrafters.berkeley.edu/~cs188/sp20/
  - http://gamescrafters.berkeley.edu/~cs188/{sp | fa}<yr>/
- Covers:
  - Search
  - Constraint Satisfaction
  - Games
  - Reinforcement Learning
  - Bayesian Networks
  - Surveys Advanced Topics
  - And more…
- Features:
  - Contains high quality YouTube videos, PowerPoint slides and homework.
  - Projects are based on the video game PacMan.
  - Material is used in many courses around the country.

3 TYPES OF MACHINE LEARNING



Supervised Learning – Learn a function from labeled data that maps input attributes to an output label e.g., linear regression, decision trees, SVMs.

Unsupervised Learning – Learn patterns in unlabeled data e.g., principle component analysis or clustering algorithms such as K-means, HAC, or Gaussian mixture models.

Reinforcement Learning – An agent learns to maximize rewards while acting in an uncertain environment.

Credit: image borrowed from lecture slides David Silver, DeepMind, "Introduction to Reinforcement Learning

THE MARKOV DECISION PROCESS

- The ***Markov Decision Process*** (MDP) provides a mathematical framework for reinforcement learning.

- Markov decision processes use probability to model uncertainty about the domain.

- Markov decision use utility to model an agent's objectives. The higher the utility, the "happier" your agent is.

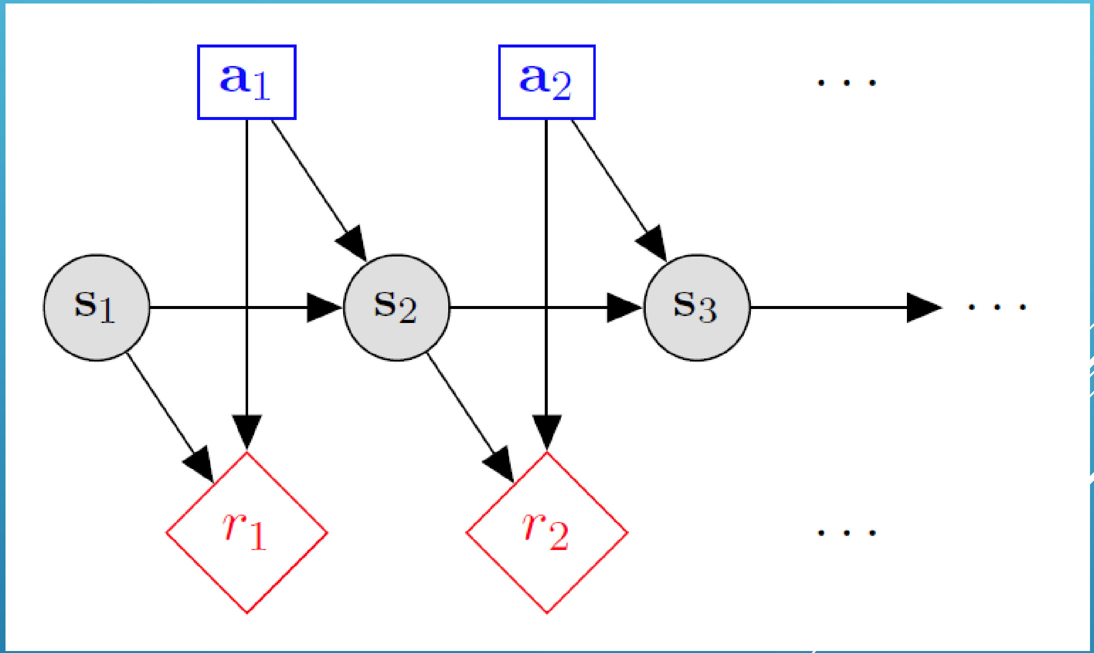
- MDP algorithms discover an optimal decision policy **π** specifying how the agent should act in all possible states in order to maximize its expected utility.

MARKOV DECISION PROCESSES

- **States:** $s_1, \dots, s_n$
- **Actions:** $a_1, \dots, a_m$
- **Reward Function:**

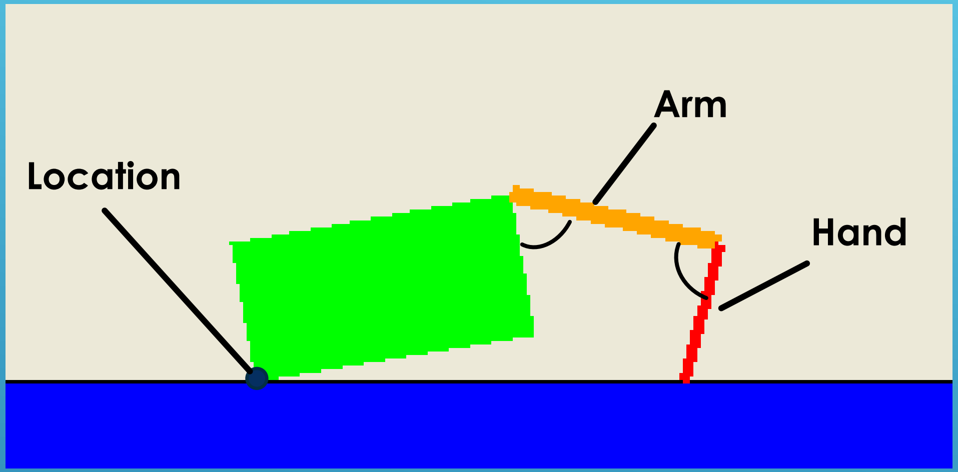
  $r(s, a, s') \in R$
- **Transition model:**

  $T(s,a,s') = P(s'|s, a)$
- **Discount factor:** $\gamma \in [0, 1]$



<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

APPLICATION: CRAWLER ROBOT



- **States:** <Location, Arm angle, Hand angle>
- **Actions:** increase Arm angle, decrease Arm angle, increase Hand angle, decrease Hand angle.
- **Reward Function:** +1 if robot moves right, -1 if robot moves left.
- **Transition model:** model of box movement caused by arm movements.

ALGORITHMS BASED ON THE MARKOV DECISION PROCESS

- **Classifying algorithms based on the Markov Decision Process**
  One way to classify these algorithms is on how much is known about the environment.

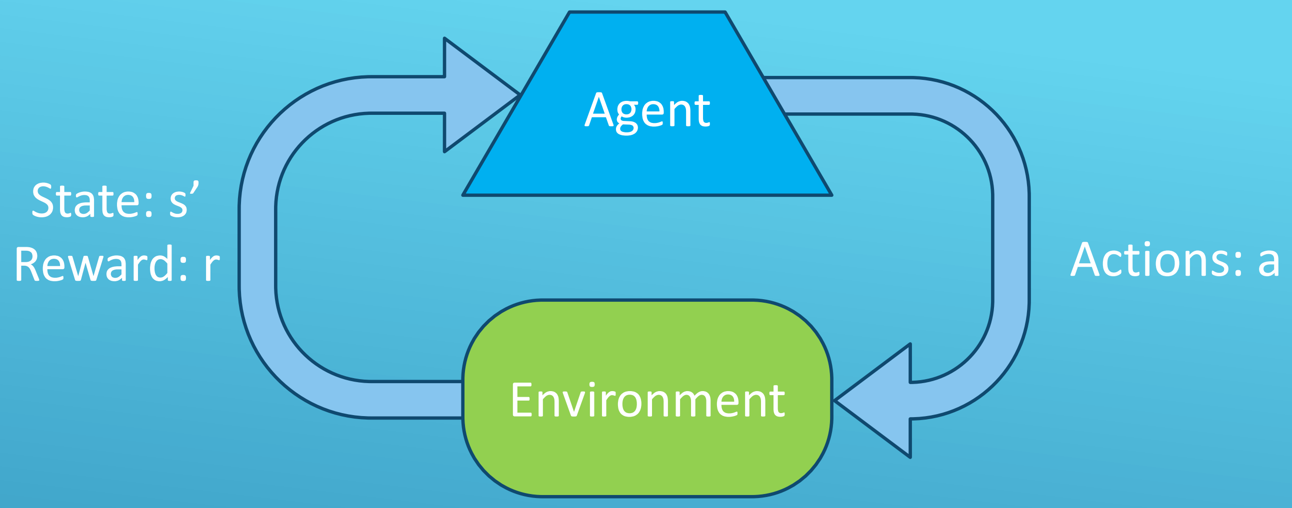
- **MDP algorithms**
  These algorithms assume perfect knowledge of the states, actions, rewards and transitions of the problem space.

- **Reinforcement learning algorithms:**
  These algorithms have knowledge of the states and actions but have no knowledge of the rewards and transitions of the problem space.

4 MDP ALGORITHMS

- **Expectimax** – a top-down recursive tree search algorithm similar to the minimax game search algorithm with a fixed search depth (finite horizon) that finds the optimal expected value of the current state.
- **Finite Horizon Value Iteration** – a bottom-up dynamic programming algorithm that finds the optimal expected value of every state within a finite horizon.
- **Infinite Horizon Value Iteration** – a bottom-up dynamic programming algorithm that finds the optimal expected value of every state.
- **Policy Iteration** - a bottom-up dynamic programming algorithm that finds the optimal policy.

THE REINFORCEMENT LEARNING PROBLEM



- Agent must learn to act to maximize expected rewards.
- Agent knows the current state s, takes action ***a***, receives a reward ***r*** and observes the next state ***s'***.
- Agent has **no access** to the reward model ***r(s,a,s')*** or the transition model ***T(s,a,s')***.
- All learning is based on observed samples of outcomes.

REINFORCEMENT LEARNING ALGORITHMS

- **Model-based RL algorithms** are based on the various MDP algorithms and try to learn the reward and transition models by exploring the environment.
- **Model-free RL algorithms** ignore the reward and transition models and try to learn the value functions directly.

MODEL-BASED RL PROS AND CONS

- **Pros:**
  - Makes maximal use of experience.
  - Solves model optimally, given enough experience.
- **Cons:**
  - Requires a computationally expensive solution procedure.
  - Requires the model to be small enough to solve

MODEL-FREE RL PROS AND CONS

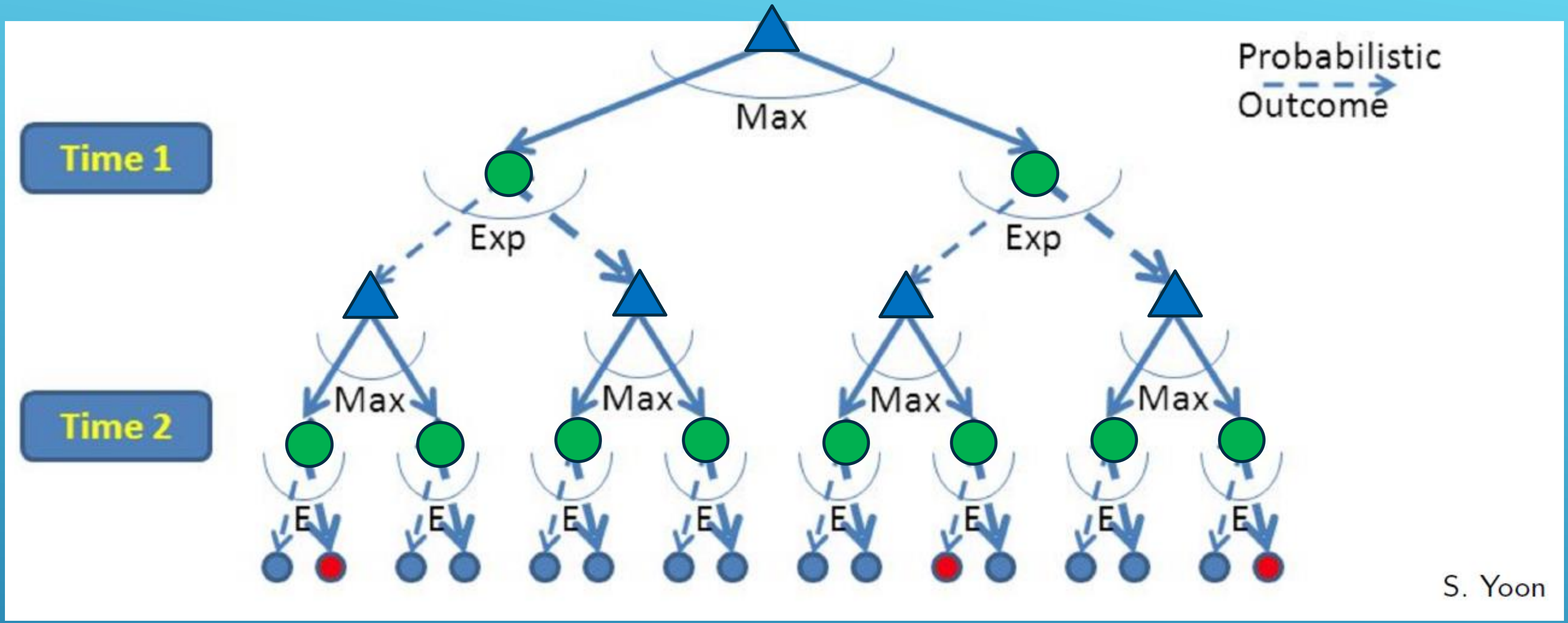
- **Pros:**
  - The solution procedure is much more efficient.
  - Can handle much larger models.

- **Cons:**
  - Learns more slowly.
  - Does not learn as much as a model-based RL in a single training episode since it doesn't learn the transition and reward models.

- **Conclusion:**
  - Model-free approaches are used for most real-world problems.
  - Examples: Q-Learning, Monte Carlo, SARSA, TD-Learning, Deep QL, etc.

Q-LEARNING

RL IS LIKE A GAME AGAINST NATURE



- Reinforcement learning is like a game-playing algorithm.
- Nodes where you move are called **states**: S ( )
- Nodes where nature “moves” are called **Q-states**: <S,A> ( )

QUANTITIES TO OPTIMIZE

- The value (utility) of a **state s:**

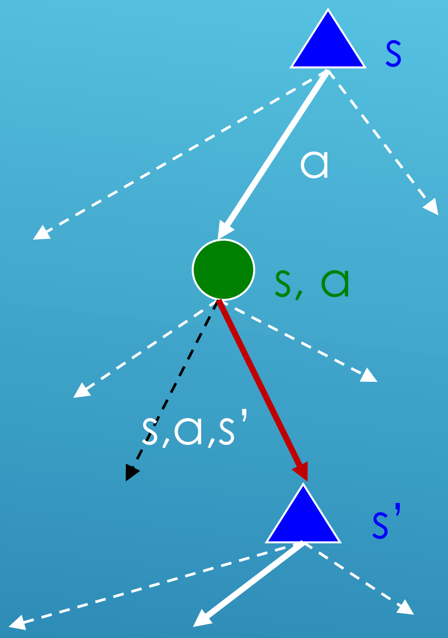
  V(s) = expected utility starting in s and acting optimally

- The value (utility) of a **q-state (s,a)**:

  Q(s,a) = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  π(s) = optimal action from state s



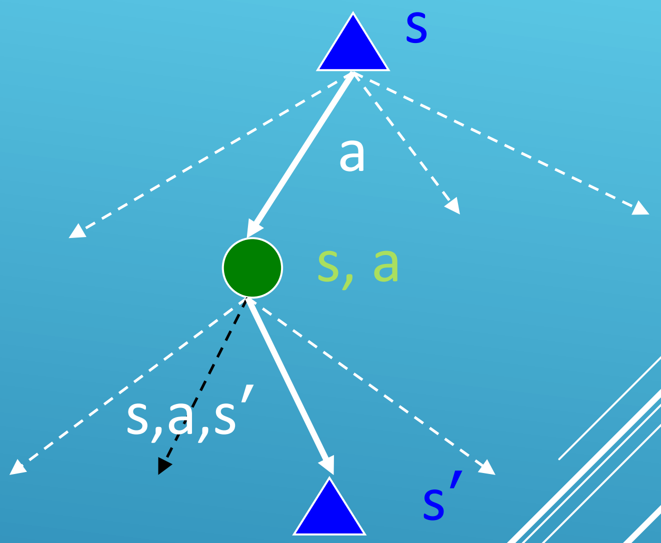
s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

THE BELLMAN EQUATIONS

- The Bellman Equations define a relationship, which when satisfied guarantees that $\boldsymbol{V(s)}$ and $\boldsymbol{Q(s,a)}$ are optimal for each state and action.
- This in turn guarantees that the policy π^* is optimal.
- There is one equation $\boldsymbol{V^*(s)}$ for each state *s.*
- There is one equation $\boldsymbol{Q^*(s,a)}$ for each state *s* and action *a.*

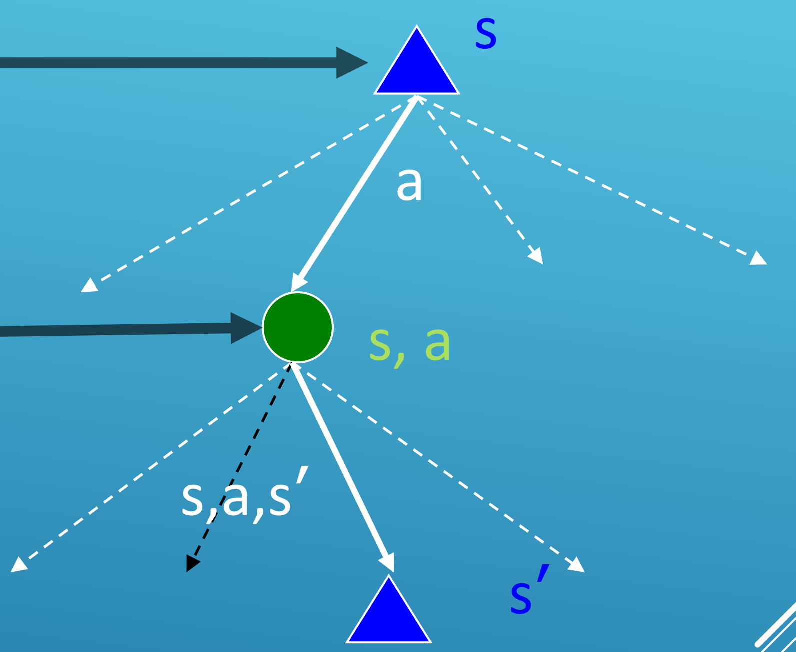


Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

THE BELLMAN EQUATIONS

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

THE OPTIMAL VALUE UTILITY EQUATION V*

- Focusing on different Bellman Equations Gives Different Algorithms
- The V* equation gives rise to these algorithms previously discussed:
  - Expectimax
  - Value Iteration
  - Policy Iteration

$$V^*(s) = \max_a Q^*(s,a) \quad [1]$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \quad [2]$$

$$\Downarrow$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

THE OPTIMAL VALUE UTILITY EQUATION Q*

The Q* equation gives rise to the Q-Learning algorithm.

$$V^*(s) = \max_a Q^*(s,a) \qquad \textbf{[1]}$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \qquad \textbf{[2]}$$

$$\Downarrow$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

Q-LEARNING UPDATE RULE (1)

- What to do about *T(s,a,s')* and *R(s,a,s')*, since we don't have these functions?

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

- Use sampling to learn Q(s,a) values as you go
  - Receive a sample transition: **(s,a,r,s')**
  - Consider your old estimate: $\mathbf{Q}(s,a)$
  - Consider your new sample estimate: $r + \gamma \max_{a'} Q_k(s',a')$
  - Incorporate the new estimate into a running average based on the learning rate α:

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

Q-LEARNING UPDATE RULE (2)

- On transitioning from state *s* to state *s'* after taking action *a*, and receiving reward *r*, update:

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

- α is the **learning rate**
  - A large α results in quicker learning but may not converge.
  - α is often decreased as learning goes on.
- γ is the **discount rate** i.e., discounts future rewards.

Q-LEARNING ALGORITHM

For each state *s* and action *a*:

$$Q(s,a) \leftarrow 0$$

Observe initial state *s*

Repeat:

Select and carry out an action *a*

Receive reward *r* and observe new state *s'*

With transition <s,a,r,s'>, update Q(s,a):

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

$s \leftarrow s'$

Until terminated

THE Q-LEARNING ALGORITHM CONVERGES TO THE TRUE Q-VALUE

- The $\max_{a'} Q(s',a')$ that we use to update $Q(s,a)$ is only an approximation and in early stages of learning it may be completely wrong.

- However the approximation get more and more accurate with every iteration and it has been shown, that if we perform this update enough times, then the Q-function will converge and represent the true Q-value.

<https://www.intel.ai/demystifying-deep-reinforcement-learning> [accessed 29 January 2020]

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- How should an agent choose an action? An obvious answer is simply to follow the current policy. However, this is often not the best way to improve your model.

- **Exploit:** use your current model to maximize the expected utility now.

- **Explore:** choose an action that will help you improve your model.

Ε-GREEDY METHOD

- At time *t*, estimate the expected value for each action:

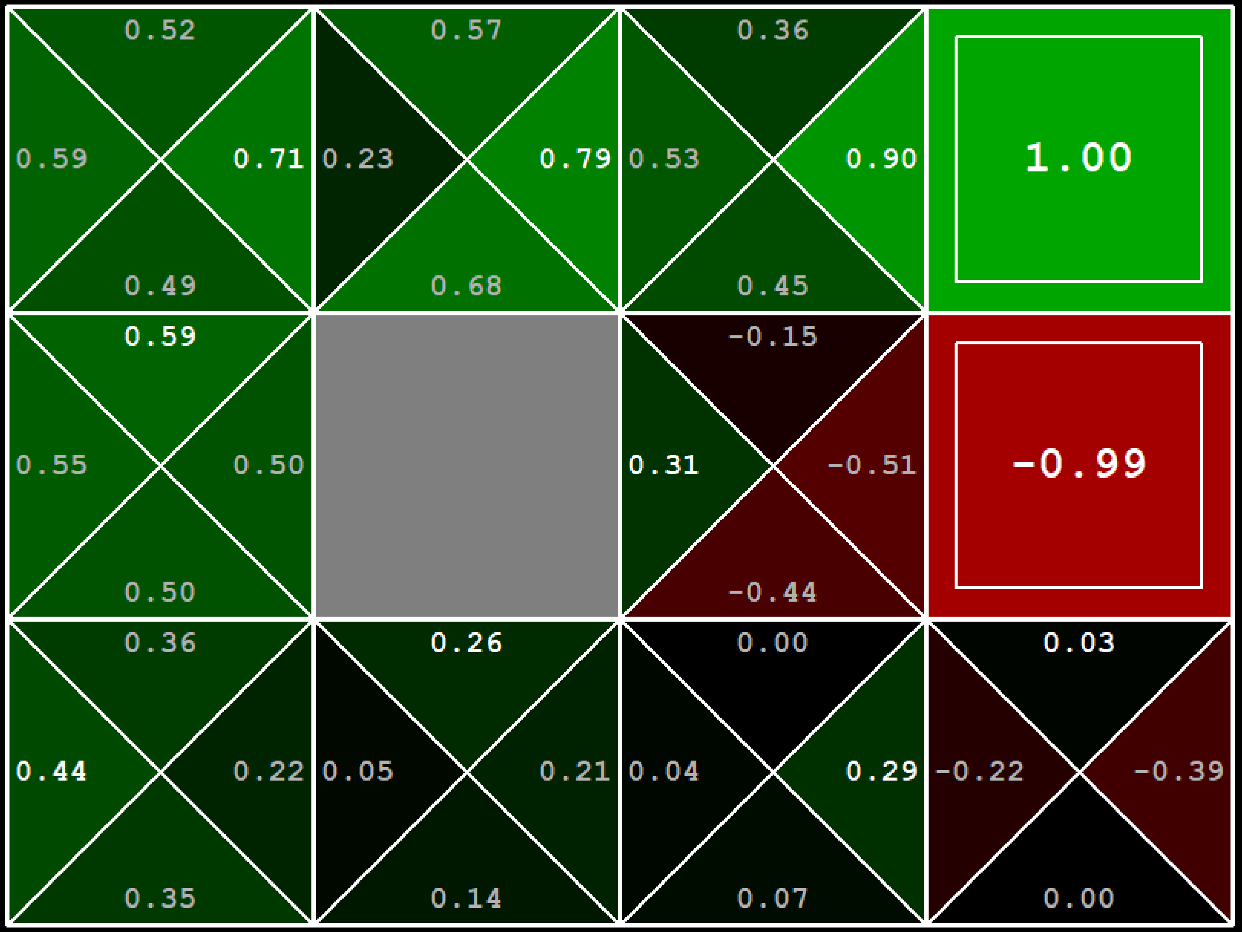
$$Q_t(a) = \frac{\text{Sum of rewards when action } a \text{ is taken prior to } t}{\text{Number of times action } a \text{ is taken prior to } t}$$

- With probability 1 - ϵ, select the action with the maximum value.

$$A_t = argmax\ Q_t(a)$$

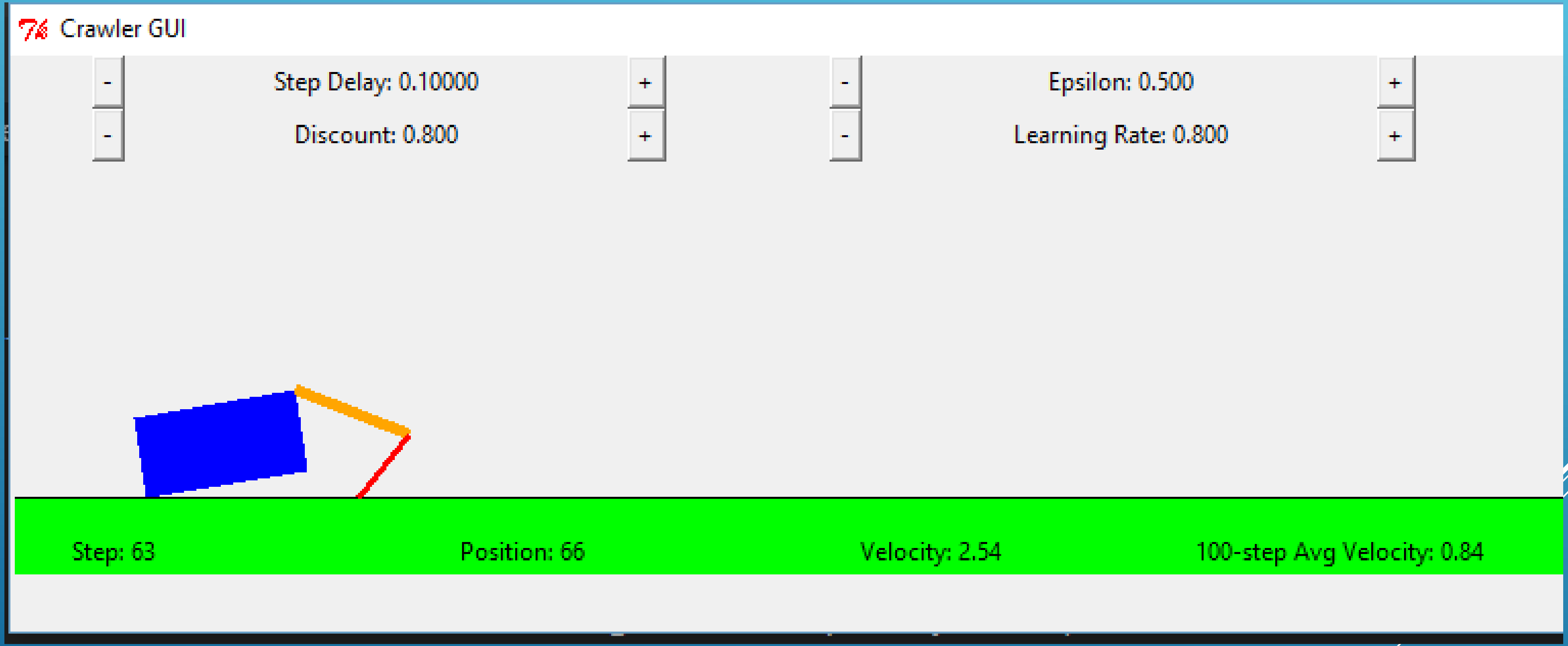
- With probability ϵ, randomly select an action from all the actions with equal probability.

Q-LEARNING EXAMPLE: GRIDWORLD



Q-VALUES AFTER 40 EPISODES

Q-LEARNING EXAMPLE: CRAWLER ROBOT



Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

Q-LEARNING EXAMPLE: DISCOUNT EFFECT

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

Description	Training Steps	Discount (γ)	Learning Rate (α)	Avg Velocity
Default	~100K	**0.8**	0.8	~1.73
Low γ	~100K	**0.5**	0.8	0.0
High γ	~100K	**0.919**	0.8	~3.33
Low α	~100K	0.8	0.2	~1.73
High α	~100K	0.8	0.9	~1.73
High γ, Low α	~100K	0.919	0.2	~3.33

INTRODUCTION TO Q-LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group