10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

# Reinforcement Learning:

## Markov Decision Processes

## +

## Value Iteration

Matt Gormley
Lecture 15
Mar. 24, 2021

# Reminders

- **Homework 5: Neural Networks**
  - Out: Thu, Mar. 18
  - Due: Mon, Mar. 29 at 11:59pm
- **Homework 6: Deep RL**
  - Out: Mon, Mar. 29
  - Due: Wed, Apr. 07 at 11:59pm


SOLID
STOP

# LEARNING PARADIGMS

# Learning Paradigms

| Paradigm | Data | |
|---|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ | $\mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| $\hookrightarrow$ Regression | $y^{(i)} \in \mathbb{R}$ | |
| $\hookrightarrow$ Classification | $y^{(i)} \in \{1, \ldots, K\}$ | |
| $\hookrightarrow$ Binary classification | $y^{(i)} \in \{+1, -1\}$ | |
| $\hookrightarrow$ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector | |

# Learning Paradigms

| Paradigm | Data | |
|---|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ | $\mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| ↪ Regression | $y^{(i)} \in \mathbb{R}$ | |
| ↪ Classification | $y^{(i)} \in \{1, \ldots, K\}$ | |
| ↪ Binary classification | $y^{(i)} \in \{+1, -1\}$ | |
| ↪ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector | |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ | $\mathbf{x} \sim p^*(\cdot)$ |

# Learning Paradigms

| Paradigm | Data |
|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ $\quad$ $\mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| $\hookrightarrow$ Regression | $y^{(i)} \in \mathbb{R}$ |
| $\hookrightarrow$ Classification | $y^{(i)} \in \{1, \ldots, K\}$ |
| $\hookrightarrow$ Binary classification | $y^{(i)} \in \{+1, -1\}$ |
| $\hookrightarrow$ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ $\quad$ $\mathbf{x} \sim p^*(\cdot)$ |
| Semi-supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$ |

# Learning Paradigms

| Paradigm | Data |
|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ $\quad$ $\mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| $\hookrightarrow$ Regression | $y^{(i)} \in \mathbb{R}$ |
| $\hookrightarrow$ Classification | $y^{(i)} \in \{1, \ldots, K\}$ |
| $\hookrightarrow$ Binary classification | $y^{(i)} \in \{+1, -1\}$ |
| $\hookrightarrow$ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ $\quad$ $\mathbf{x} \sim p^*(\cdot)$ |
| Semi-supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$ |
| Online | $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$ |

# Learning Paradigms

| Paradigm | Data |
|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \qquad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| ↪ Regression | $y^{(i)} \in \mathbb{R}$ |
| ↪ Classification | $y^{(i)} \in \{1, \ldots, K\}$ |
| ↪ Binary classification | $y^{(i)} \in \{+1, -1\}$ |
| ↪ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \qquad \mathbf{x} \sim p^*(\cdot)$ |
| Semi-supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$ |
| Online | $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$ |
| Active Learning | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost |

# Learning Paradigms

| Paradigm | Data |
|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \quad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| ↪ Regression | $y^{(i)} \in \mathbb{R}$ |
| ↪ Classification | $y^{(i)} \in \{1, \ldots, K\}$ |
| ↪ Binary classification | $y^{(i)} \in \{+1, -1\}$ |
| ↪ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \quad \mathbf{x} \sim p^*(\cdot)$ |
| Semi-supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$ |
| Online | $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$ |
| Active Learning | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost |
| Imitation Learning | $\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots\}$ |

# Learning Paradigms

| Paradigm | Data |
| --- | --- |
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \quad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| ↪ Regression | $y^{(i)} \in \mathbb{R}$ |
| ↪ Classification | $y^{(i)} \in \{1, \ldots, K\}$ |
| ↪ Binary classification | $y^{(i)} \in \{+1, -1\}$ |
| ↪ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \quad \mathbf{x} \sim p^*(\cdot)$ |
| Semi-supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$ |
| Online | $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$ |
| Active Learning | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost |
| Imitation Learning | $\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots\}$ |
| Reinforcement Learning | $\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots\}$ |

# REINFORCEMENT LEARNING

# Examples of Reinforcement Learning

- How should a robot behave so as to optimize its "performance"? (Robotics)

- How to automate the motion of a helicopter? (Control Theory)

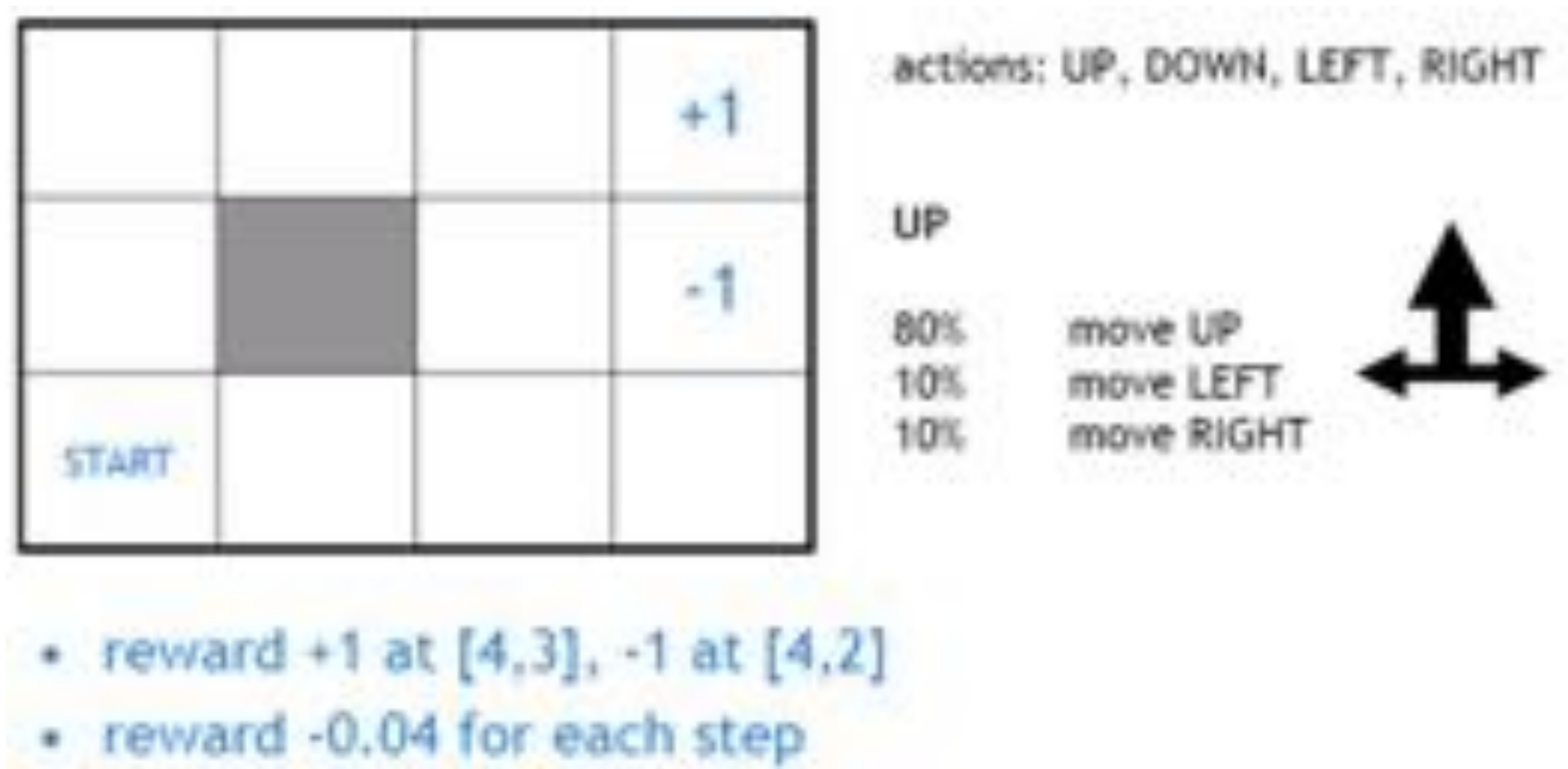
- How to make a good chess-playing program? (Artificial Intelligence)

# Autonomous Helicopter

Video:

https://www.youtube.com/watch?v=VCdxqn0fcnE

# Robot in a room




- what’s the strategy to achieve max reward?
- what if the actions were NOT deterministic?

# History of Reinforcement Learning

- Roots in the psychology of animal learning (Thorndike,1911).
- Another independent thread was the problem of optimal control, and its solution using dynamic programming (Bellman, 1957).
- Idea of temporal difference learning (on-line method), e.g., playing board games (Samuel, 1959).
- A major breakthrough was the discovery of Q-learning (Watkins, 1989).

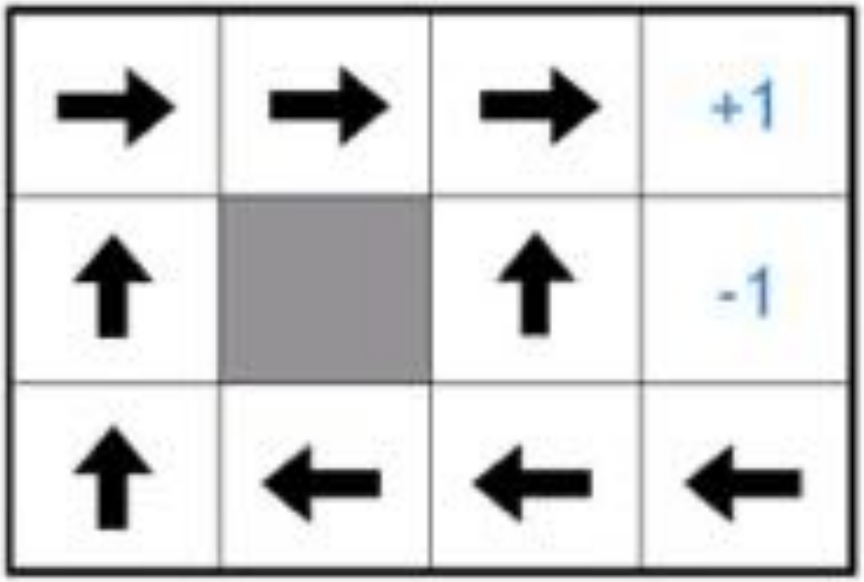
# What is special about RL?

- RL is learning how to map states to actions, so as to maximize a numerical reward over time.
- Unlike other forms of learning, it is a multistage decision-making process (often Markovian).
- An RL agent must learn by trial-and-error. (Not entirely supervised, but interactive)
- Actions may affect not only the immediate reward but also subsequent rewards (Delayed effect).

# Elements of RL

- A policy
  - A map from state space to action space.
  - May be stochastic.
- A reward function
  - It maps each state (or, state-action pair) to a real number, called reward.
- A value function
  - Value of a state (or, state-action pair) is the total expected reward, starting from that state (or, state-action pair).

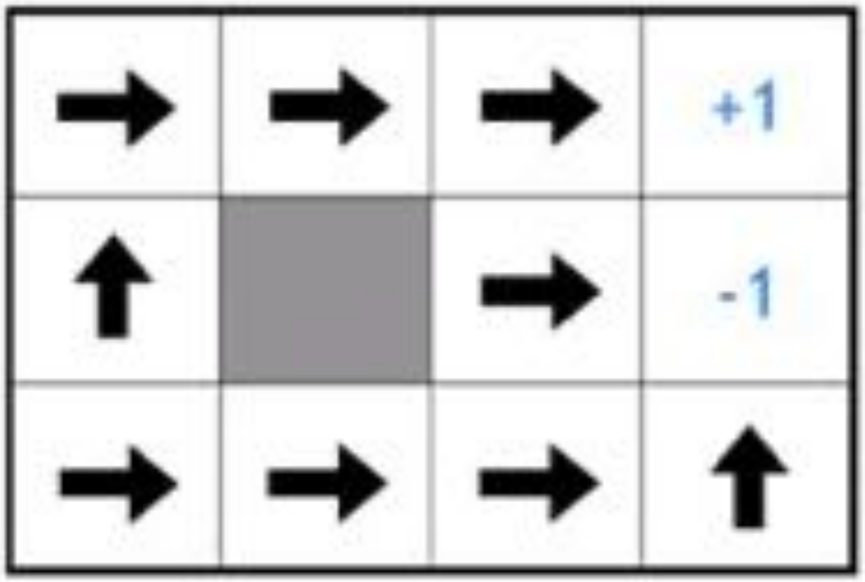
# Policy



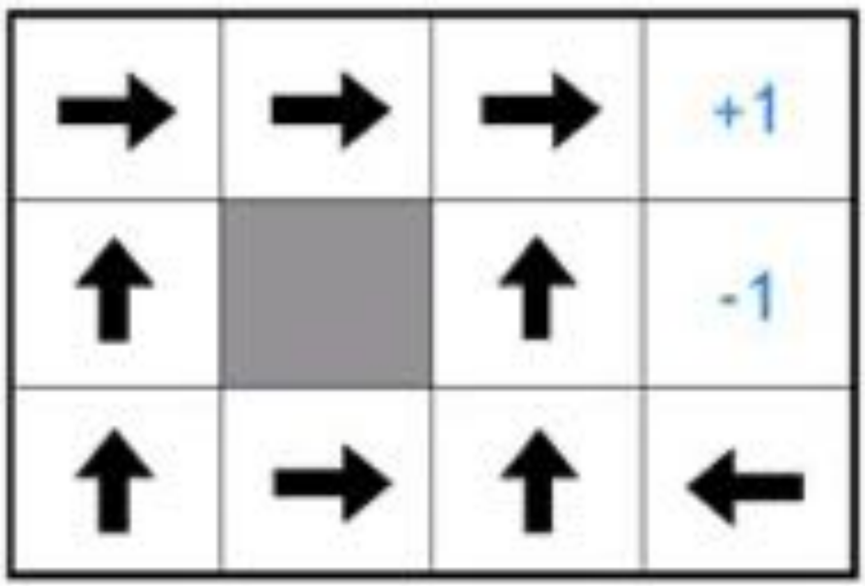

**Question:**
Is this policy optimal: yes or no? Briefly justify your answer.

**Answer:** (*Hint*: both yes and no are acceptable answers, I'm interested in your justification.)

# Reward for each step -2



# Reward for each step: -0.1



# The Precise Goal

- To find a policy that maximizes the Value function.
  - transitions and rewards usually not available

- There are different approaches to achieve this goal in various situations.

- Value iteration and Policy iteration are two more classic approaches to this problem. But essentially both are dynamic programming.

- Q-learning is a more recent approaches to this problem. Essentially it is a temporal-difference method.

# MARKOV DECISION PROCESSES

# Markov Decision Process

- For **supervised learning** the **PAC learning framework** provided assumptions about where our data came from:

$$\mathbf{x} \sim p^*(\cdot) \text{ and } y = c^*(\cdot)$$

- For **reinforcement learning** we assume our data comes from a **Markov decision process** (MDP)

# Markov Decision Process

*Whiteboard*

- Components: states, actions, state transition probabilities, reward function
- Markovian assumption
- MDP Model
- MDP Goal: Infinite-horizon Discounted Reward
- deterministic vs. nondeterministic MDP
- deterministic vs. stochastic policy

# MARKOV DECISION PROCESSES

# Markov Decision Process

- For **supervised learning** the **PAC learning framework** provided assumptions about where our data came from:

$$\mathbf{x} \sim p^*(\cdot) \text{ and } y = c^*(\cdot)$$

- For **reinforcement learning** we assume our data comes from a **Markov decision process** (MDP)

# Markov Decision Process

*Whiteboard*

- Components: states, actions, state transition probabilities, reward function
- Markovian assumption
- MDP Model
- MDP Goal: Infinite-horizon Discounted Reward
- deterministic vs. nondeterministic MDP
- deterministic vs. stochastic policy

# Exploration vs. Exploitation

*Whiteboard*

- Explore vs. Exploit Tradeoff
- Ex: k-Armed Bandits
- Ex: Traversing a Maze

# FIXED POINT ITERATION

# Fixed Point Iteration for Optimization

- Fixed point iteration is a general tool for solving systems of equations
- It can also be applied to optimization.

$$J(\boldsymbol{\theta})$$

$$\frac{dJ(\boldsymbol{\theta})}{d\theta_i} = 0 = f(\boldsymbol{\theta})$$

$$0 = f(\boldsymbol{\theta}) \Rightarrow \theta_i = g(\boldsymbol{\theta})$$

$$\theta_i^{(t+1)} = g(\boldsymbol{\theta}^{(t)})$$

1. Given objective function:
2. Compute derivative, set to zero (call this function $f$).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For $i$ in $\{1,\ldots,K\}$, update each parameter and increment $t$:
6. Repeat #5 until convergence

# Fixed Point Iteration for Optimization

- Fixed point iteration is a general tool for solving systems of equations
- It can also be applied to optimization.

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

1. Given objective function:
2. Compute derivative, set to zero (call this function $f$ ).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For $i$ in $\{1,\ldots,K\}$, update each parameter and increment $t$:
6. Repeat #5 until convergence

# Fixed Point Iteration for Optimization

We can implement our example in a few lines of python.

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

```python
def f1(x):
    '''f(x) = x^2 - 3x + 2'''
    return x**2 - 3.*x + 2.

def g1(x):
    '''g(x) = \frac{x^2 + 2}{3}'''
    return (x**2 + 2.) / 3.

def fpi(g, x0, n, f):
    '''Optimizes the 1D function g by fixed point iteration
    starting at x0 and stopping after n iterations. Also
    includes an auxiliary function f to test at each value.'''
    x = x0
    for i in range(n):
        print("i=%2d x=%.4f f(x)=%.4f" % (i, x, f(x)))
        x = g(x)
    i += 1
    print("i=%2d x=%.4f f(x)=%.4f" % (i, x, f(x)))
    return x

if __name__ == "__main__":
    x = fpi(g1, 0, 20, f1)
```

# Fixed Point Iteration for Optimization

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

```
$ python fixed-point-iteration.py
i= 0 x=0.0000 f(x)=2.0000
i= 1 x=0.6667 f(x)=0.4444
i= 2 x=0.8148 f(x)=0.2195
i= 3 x=0.8880 f(x)=0.1246
i= 4 x=0.9295 f(x)=0.0755
i= 5 x=0.9547 f(x)=0.0474
i= 6 x=0.9705 f(x)=0.0304
i= 7 x=0.9806 f(x)=0.0198
i= 8 x=0.9872 f(x)=0.0130
i= 9 x=0.9915 f(x)=0.0086
i=10 x=0.9944 f(x)=0.0057
i=11 x=0.9963 f(x)=0.0038
i=12 x=0.9975 f(x)=0.0025
i=13 x=0.9983 f(x)=0.0017
i=14 x=0.9989 f(x)=0.0011
i=15 x=0.9993 f(x)=0.0007
i=16 x=0.9995 f(x)=0.0005
i=17 x=0.9997 f(x)=0.0003
i=18 x=0.9998 f(x)=0.0002
i=19 x=0.9999 f(x)=0.0001
i=20 x=0.9999 f(x)=0.0001
```

# VALUE ITERATION

# Definitions for Value Iteration

*Whiteboard*

- State trajectory
- Value function
- Bellman equations
- Optimal policy
- Optimal value function
- Computing the optimal policy
- Ex: Path Planning