
# 10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

# Deep RL
# +
# K-Means

Matt Gormley
Lecture 25
Apr. 17, 2019

# Reminders

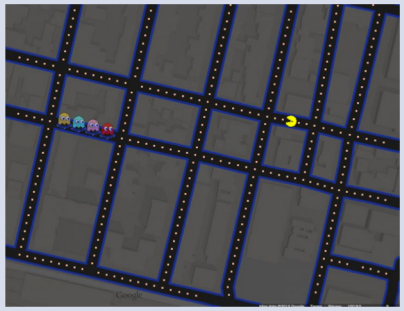
- **Homework 8: Reinforcement Learning**
  - **Out: Wed, Apr 10**
  - **Due: Wed, Apr 24 at 11:59pm**

- **Today's In-Class Poll**
  - **http://p25.mlcourse.org**

# Q&A

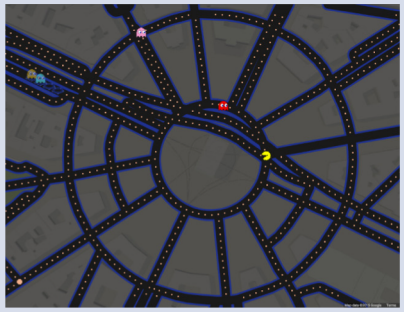
**Q:** Do we have to retrain our RL agent every time we change our state space?

**A:** Yes. But whether your state space changes from one setting to another is determined by your design of the state representation.



Two examples:

- State Space A: <x,y> position on map
  e.g. $s_t$ = <74, 152>
- State Space B: window of pixel colors centered at current Pac Man location
  e.g. $s_t$ =

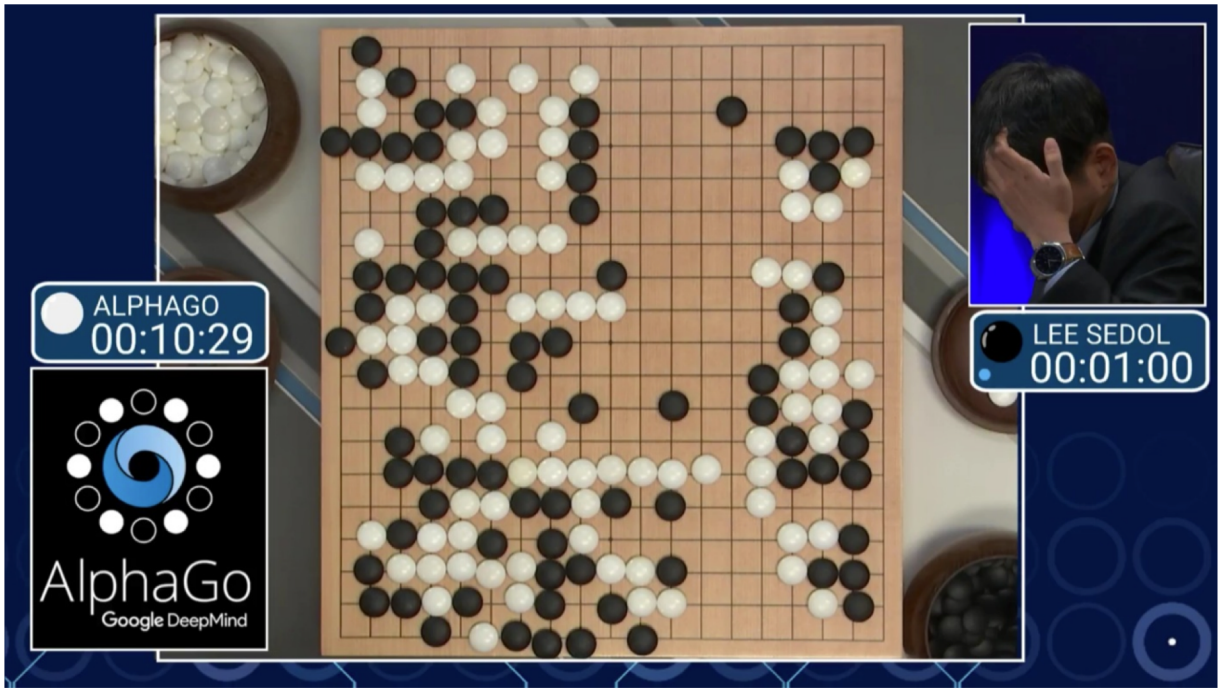


| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

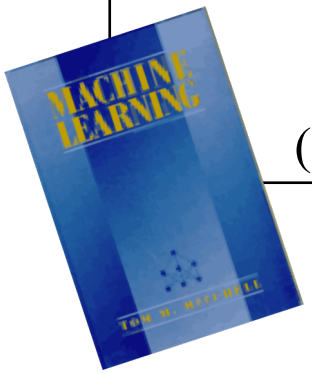
# DEEP RL EXAMPLES

# TD Gammon → Alpha Go

## Learning to beat the masters at board games

| THEN | NOW |
| --- | --- |
| "…the world's top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself…" |  |

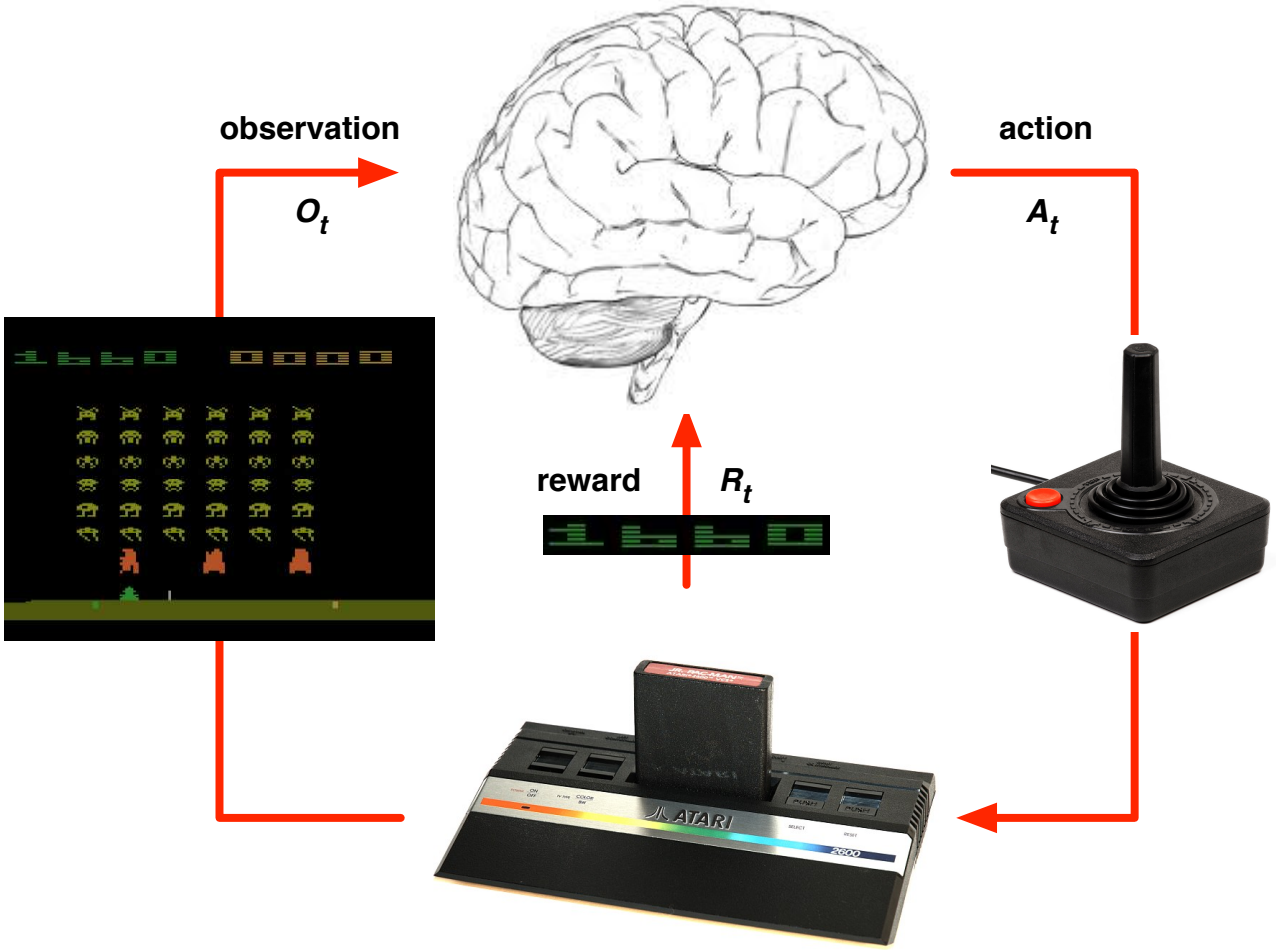





(Mitchell, 1997)

# Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen
- It receives rewards as the game score
- Actions decide how to move the joystick / buttons




Figures from David Silver (Intro RL lecture)

# Playing Atari with Deep RL

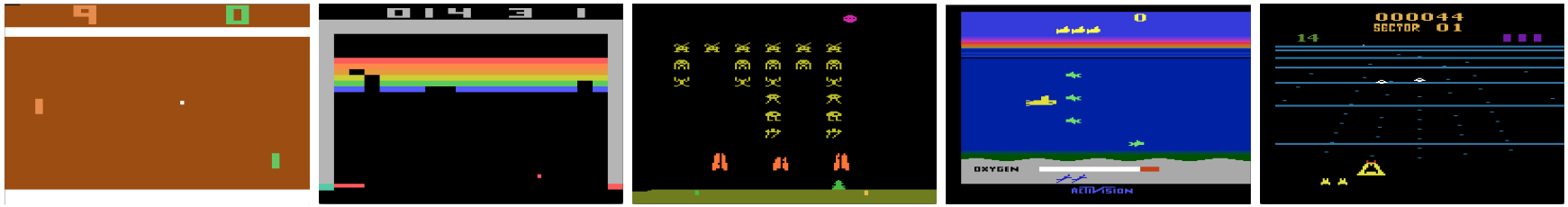



Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

## Videos:

- Atari Breakout: https://www.youtube.com/watch?v=V1eYniJ0Rnk
- Space Invaders: https://www.youtube.com/watch?v=ePv0Fs9cGgU

Figures from Mnih et al. (2013)

# Playing Atari with Deep RL

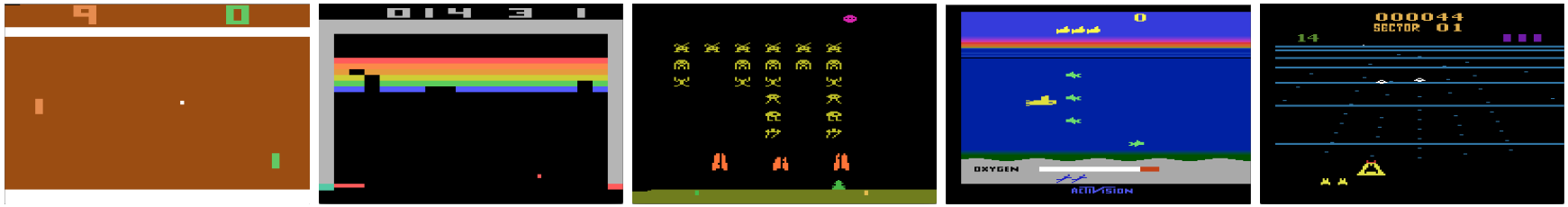


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

| | B. Rider | Breakout | Enduro | Pong | Q*bert | Seaquest | S. Invaders |
|---|---|---|---|---|---|---|---|
| **Random** | 354 | 1.2 | 0 | −20.4 | 157 | 110 | 179 |
| **Sarsa** [3] | 996 | 5.2 | 129 | −19 | 614 | 665 | 271 |
| **Contingency** [4] | 1743 | 6 | 159 | −17 | 960 | 723 | 268 |
| **DQN** | **4092** | **168** | **470** | **20** | **1952** | **1705** | **581** |
| **Human** | 7456 | 31 | 368 | −3 | 18900 | 28010 | 3690 |
| **HNeat Best** [8] | 3616 | 52 | 106 | 19 | 1800 | 920 | **1720** |
| **HNeat Pixel** [8] | 1332 | 4 | 91 | −16 | 1325 | 800 | 1145 |
| **DQN Best** | **5184** | **225** | **661** | **21** | **4500** | **1740** | 1075 |

Table 1: The upper table compares average total reward for various learning methods by running an $\epsilon$-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an $\epsilon$-greedy policy with $\epsilon = 0.05$.



Figures from Mnih et al. (2013)

# Deep Q-Learning

**Question:** *What if our state space S is too large to represent with a table?*

**Examples:**

- $s_t$ = pixels of a video game
- $s_t$ = continuous values of a sensors in a manufacturing robot
- $s_t$ = sensor output from a self-driving car

**Answer:** Use a parametric function to approximate the table entries

**Key Idea:**

1. Use a neural network $Q(s,a; \theta)$ to approximate $Q^*(s,a)$
2. Learn the parameters $\theta$ via SGD with training examples $< s_t, a_t, r_t, s_{t+1} >$

# Deep Q-Learning

*Whiteboard*

- Strawman loss function (i.e. what we cannot compute)
- Approximating the Q function with a neural network
- Approximating the Q function with a linear model
- Deep Q-Learning
- function approximators
  (<state, $action_i$> → q-value
  **vs.**
  state → all action q-values)

# Experience Replay

- **Problems** with online updates for Deep Q-learning:
  - not i.i.d. as SGD would assume
  - quickly forget rare experiences that might later be useful to learn from
- **Uniform Experience Replay** (Lin, 1992):
  - Keep a *replay memory* D = $\{e_1, e_2, \ldots, e_N\}$ of N most recent experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Alternate two steps:
    1. Repeat T times: randomly sample $e_i$ from D and apply a Q-Learning update to $e_i$
    2. Agent selects an action using epsilon greedy policy to receive new experience that is added to D
- **Prioritized Experience Replay** (Schaul et al, 2016)
  - similar to Uniform ER, but sample so as to prioritize experiences with high error

# Alpha Go

## Game of Go (圍棋)

- 19x19 **board**
- Players alternately play black/white **stones**
- **Goal** is to fully encircle the largest region on the board
- **Simple** rules, but **extremely complex** game play

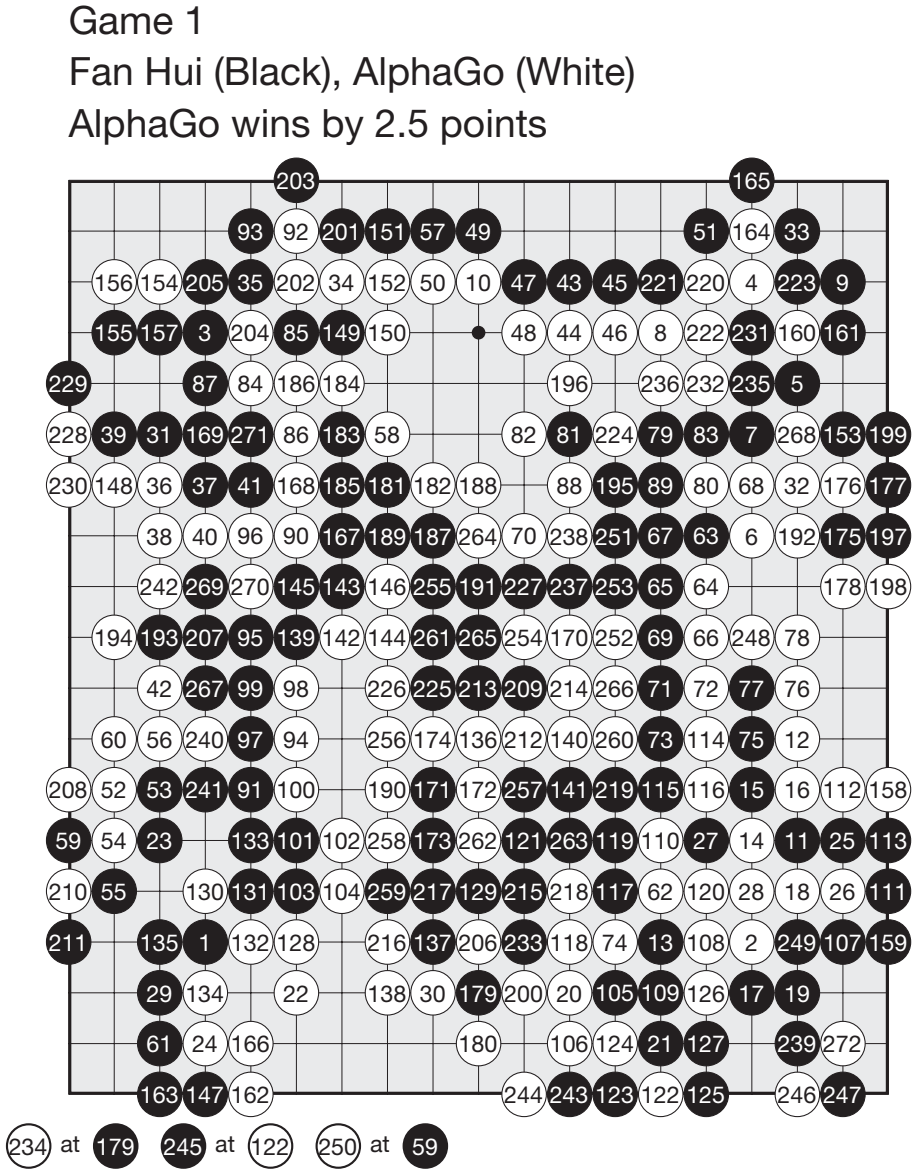



Figure from Silver et al. (2016)

# Alpha Go

- State space is too large to represent explicitly since # of sequences of moves is $O(b^d)$
  - Go: $b=250$ and $d=150$
  - Chess: $b=35$ and $d=80$
- Key idea:
  - Define a neural network to approximate the value function
  - Train by policy gradient

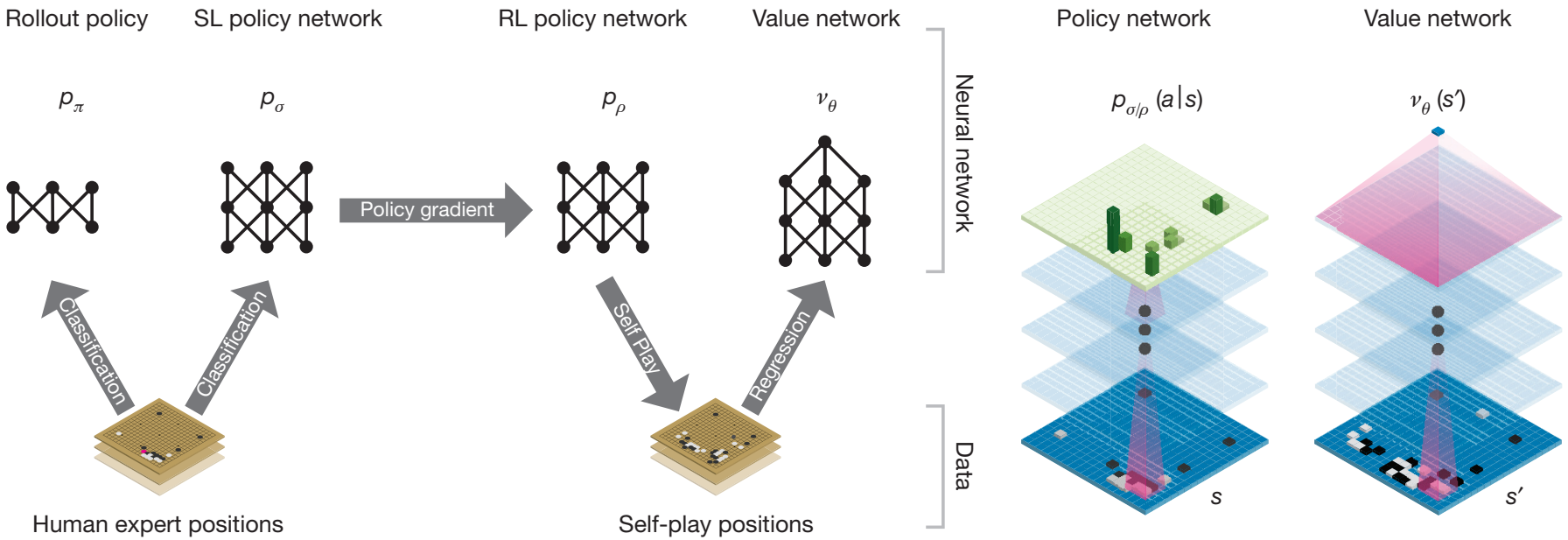



Figure from Silver et al. (2016)

# Alpha Go

- Results of a tournament
- From Silver et al. (2016): "a 230 point gap corresponds to a 79% probability of winning"

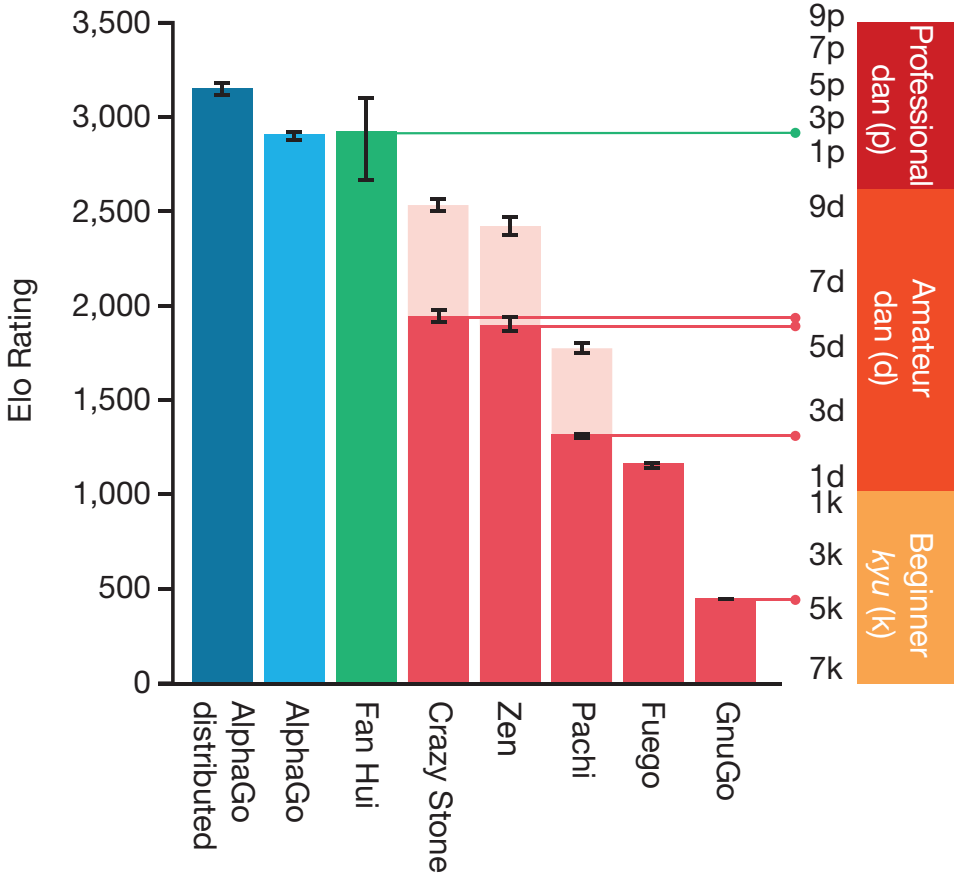



Figure from Silver et al. (2016)

# Learning Objectives

**Reinforcement Learning: Q-Learning**

*You should be able to…*

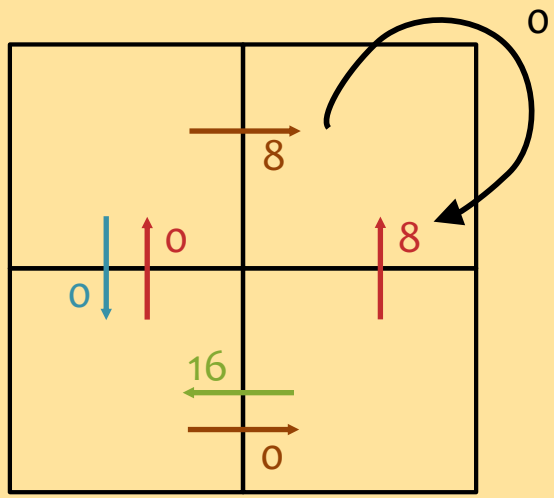
1. Apply Q-Learning to a real-world environment
2. Implement Q-learning
3. Identify the conditions under which the Q-learning algorithm will converge to the true value function
4. Adapt Q-learning to Deep Q-learning by employing a neural network approximation to the Q function
5. Describe the connection between Deep Q-Learning and regression

# Q-Learning

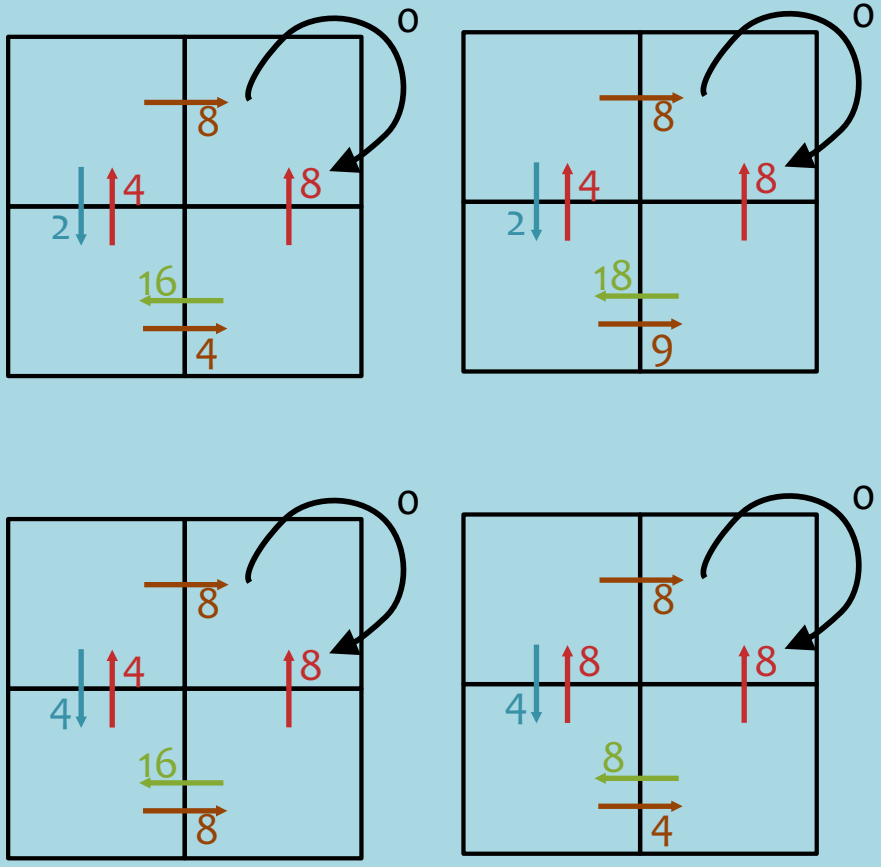
**Question:**

For the R(s,a) values shown on the arrows below, which are the corresponding Q*(s,a) values?

Assume discount factor = 0.5.




**Answer:**



# K-MEANS

# K-Means Outline

- **Clustering: Motivation / Applications**
- **Optimization Background**
  - Coordinate Descent
  - Block Coordinate Descent
- **Clustering**
  - Inputs and Outputs
  - Objective-based Clustering
- **K-Means**
  - K-Means Objective
  - Computational Complexity
  - K-Means Algorithm / Lloyd's Method
- **K-Means Initialization**
  - Random
  - Farthest Point
  - K-Means++

# Clustering, Informal Goals

**Goal**: Automatically partition unlabeled data into groups of similar datapoints.

**Question**: When and why would we want to do this?

**Useful for:**

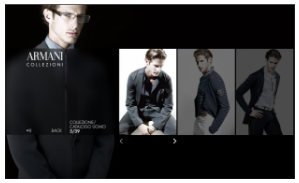
- Automatically organizing data.
- Understanding hidden structure in data.
- Preprocessing for further analysis.
    - Representing high-dimensional data in a low-dimensional space (e.g., for visualization purposes).

Slide courtesy of Nina Balcan

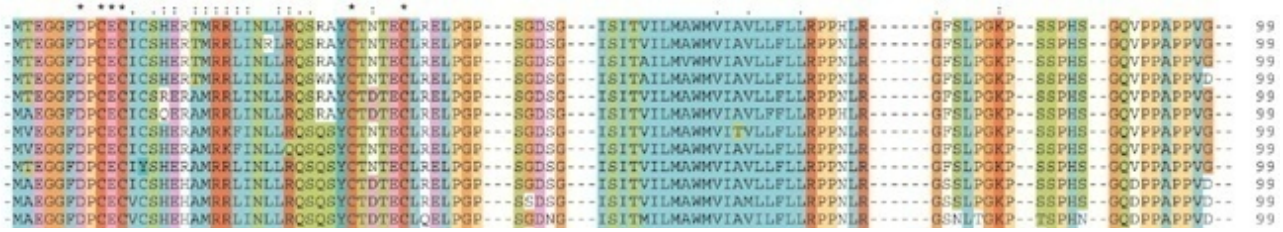
# Applications (Clustering comes up everywhere…)

- Cluster news articles or web pages or search results by topic.

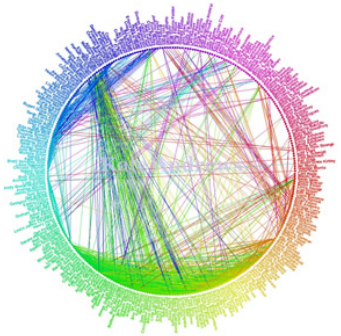




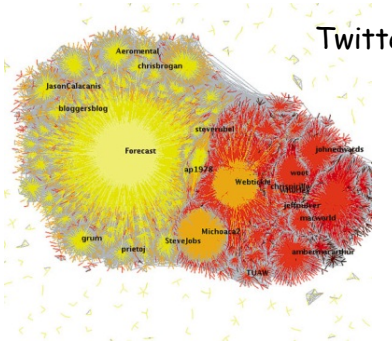

- Cluster protein sequences by function or genes according to expression profile.

- Cluster users of social networks by interest (community detection).



Facebook network



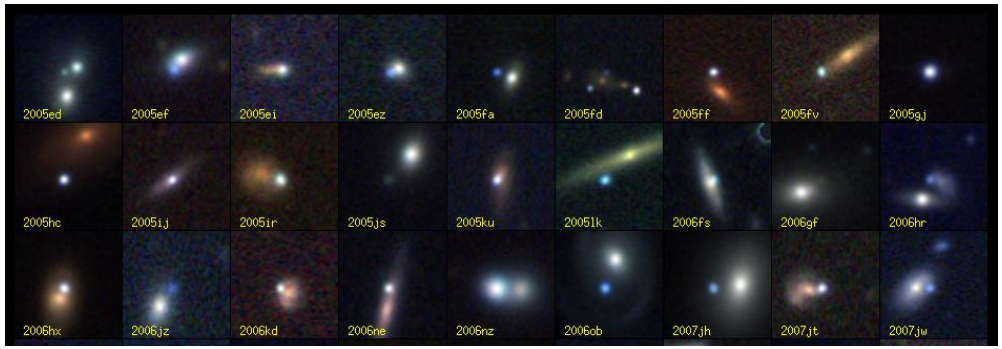

Twitter Network

Slide courtesy of Nina Balcan

# Applications (Clustering comes up everywhere…)

- Cluster customers according to purchase history.


- Cluster galaxies or nearby stars (e.g. Sloan Digital Sky Survey)




- And many many more applications….

Slide courtesy of Nina Balcan

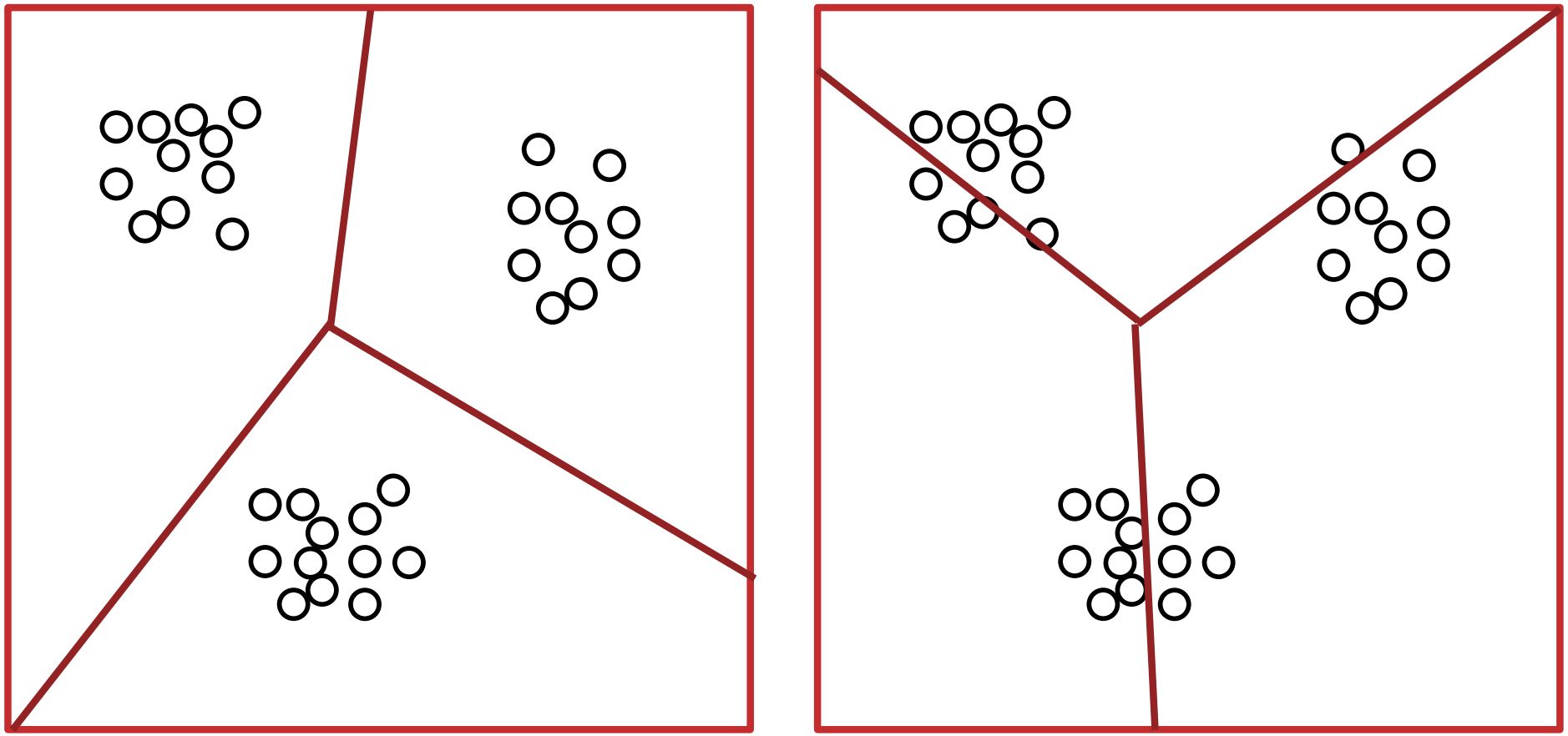
# Optimization Background

*Whiteboard:*

- Coordinate Descent
- Block Coordinate Descent

# Clustering

Question: Which of these partitions is "better"?



# K-Means

*Whiteboard:*

- Clustering: Inputs and Outputs
- Objective-based Clustering
- K-Means Objective
- Computational Complexity
- K-Means Algorithm / Lloyd's Method

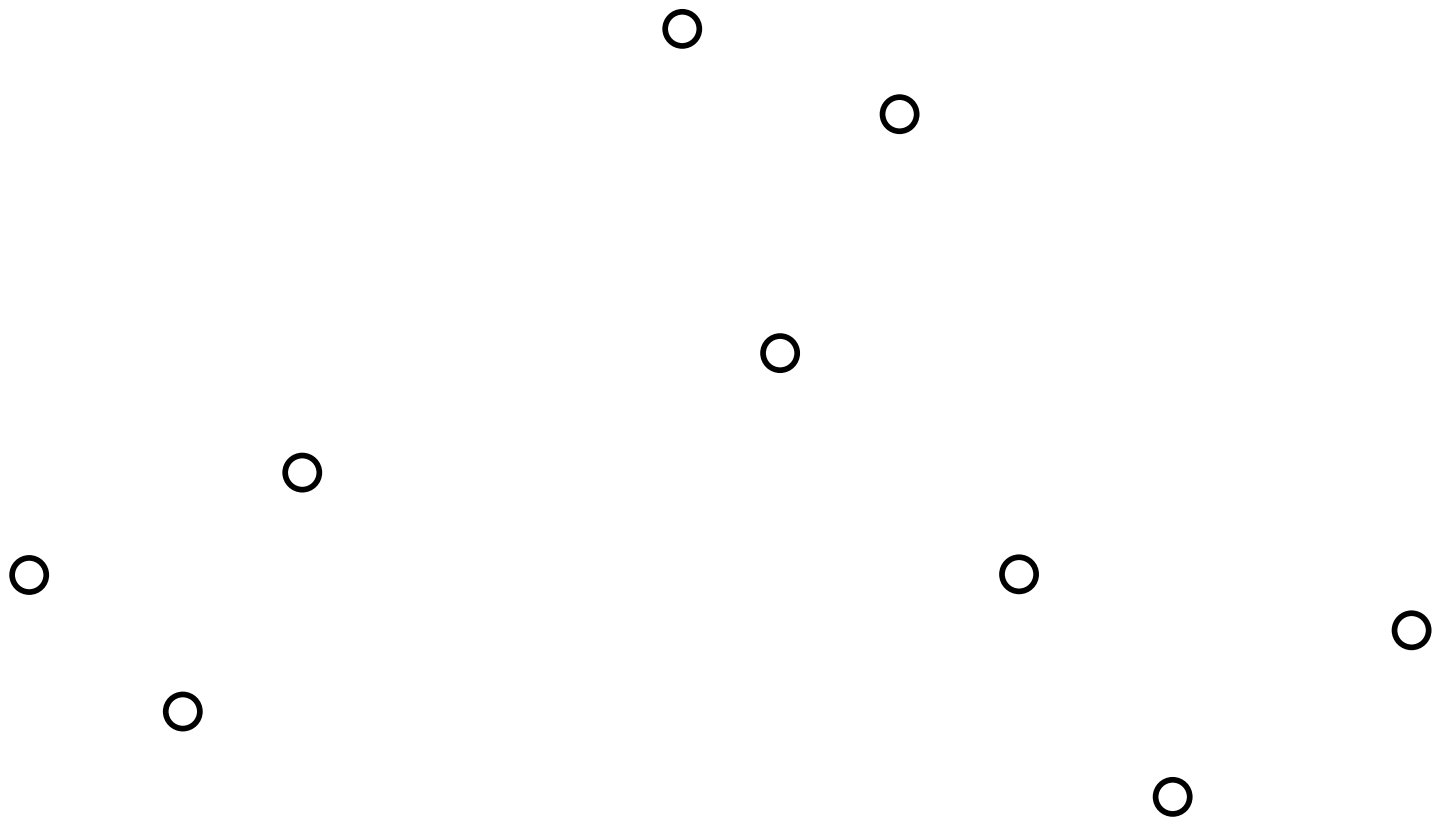
# K-Means Initialization

*Whiteboard:*

- Random
- Furthest Traversal
- K-Means++

# Lloyd's method: Random Initialization

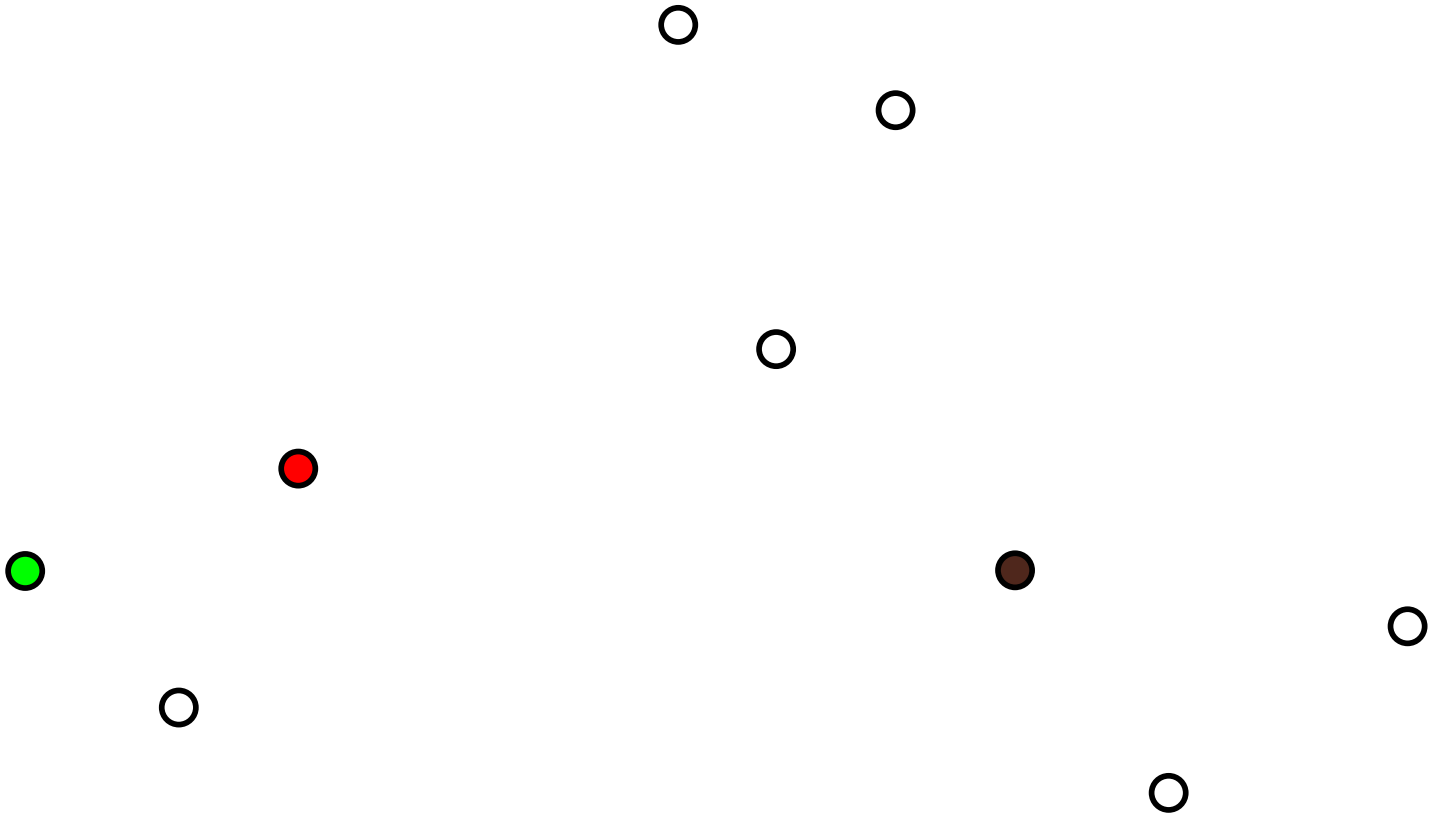
Example: Given a set of datapoints



Slide courtesy of Nina Balcan

# Lloyd's method: Random Initialization

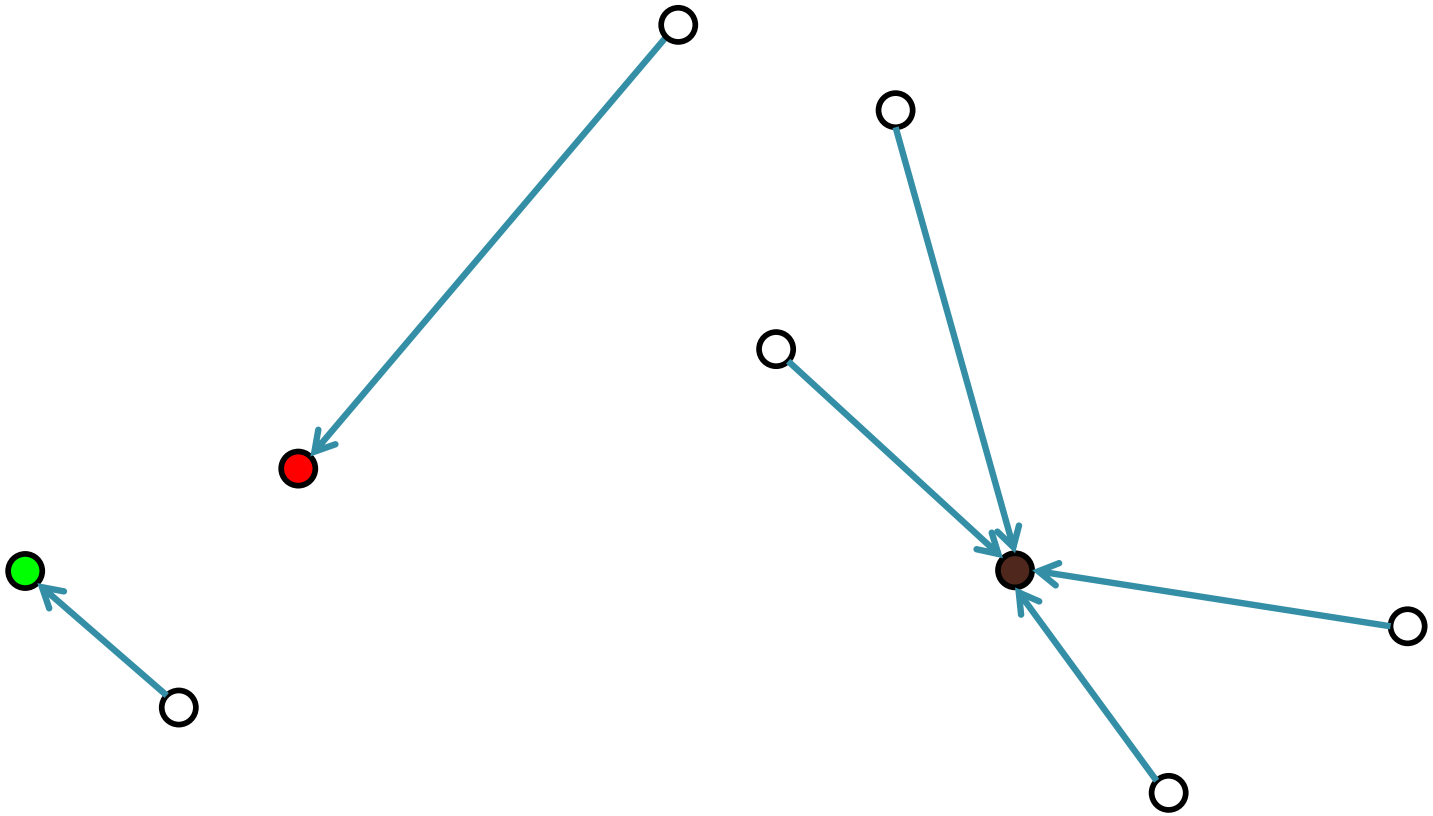
Select initial centers at random



Slide courtesy of Nina Balcan

# Lloyd's method: Random Initialization

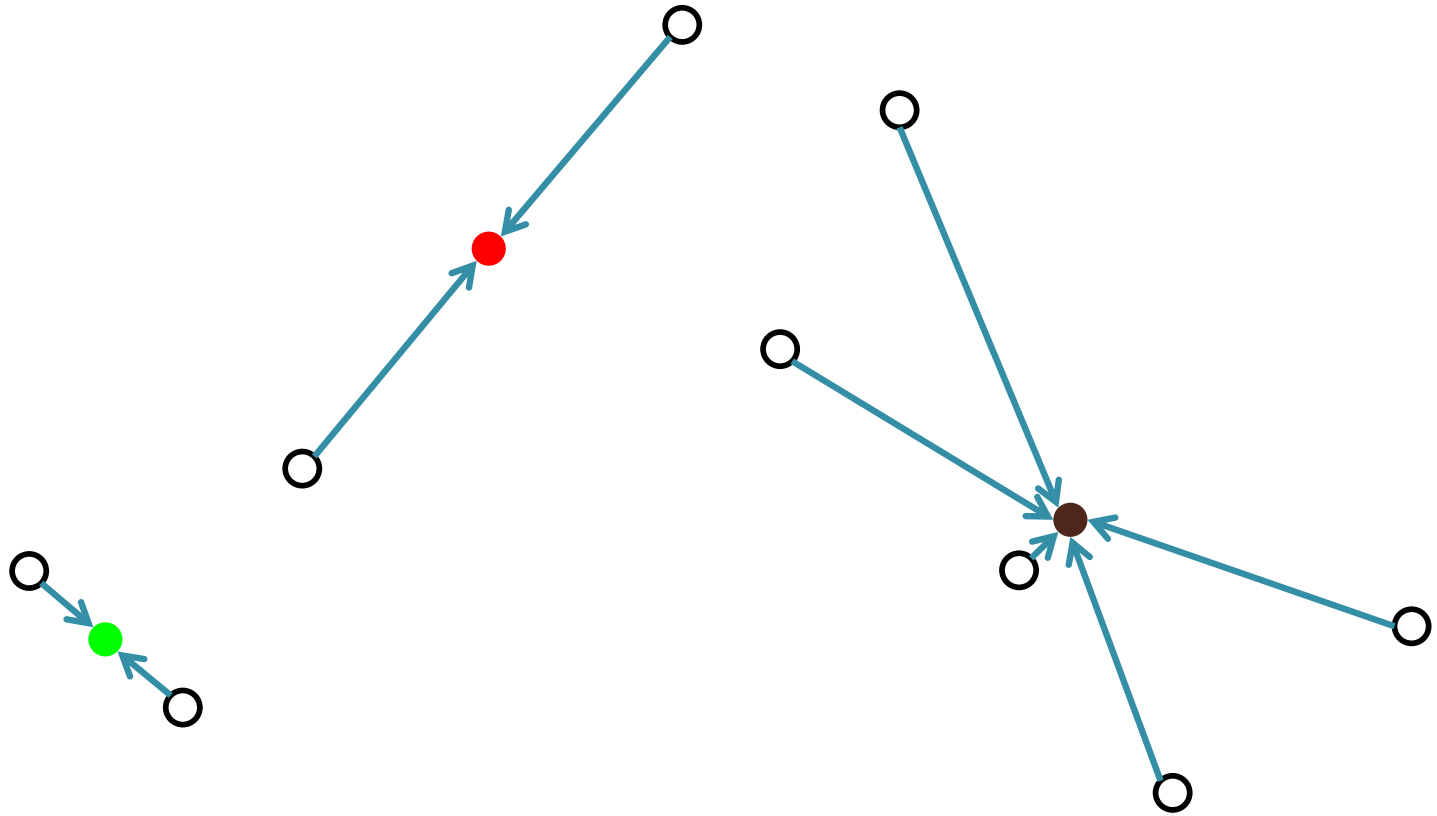
Assign each point to its nearest center



Slide courtesy of Nina Balcan

# Lloyd's method: Random Initialization

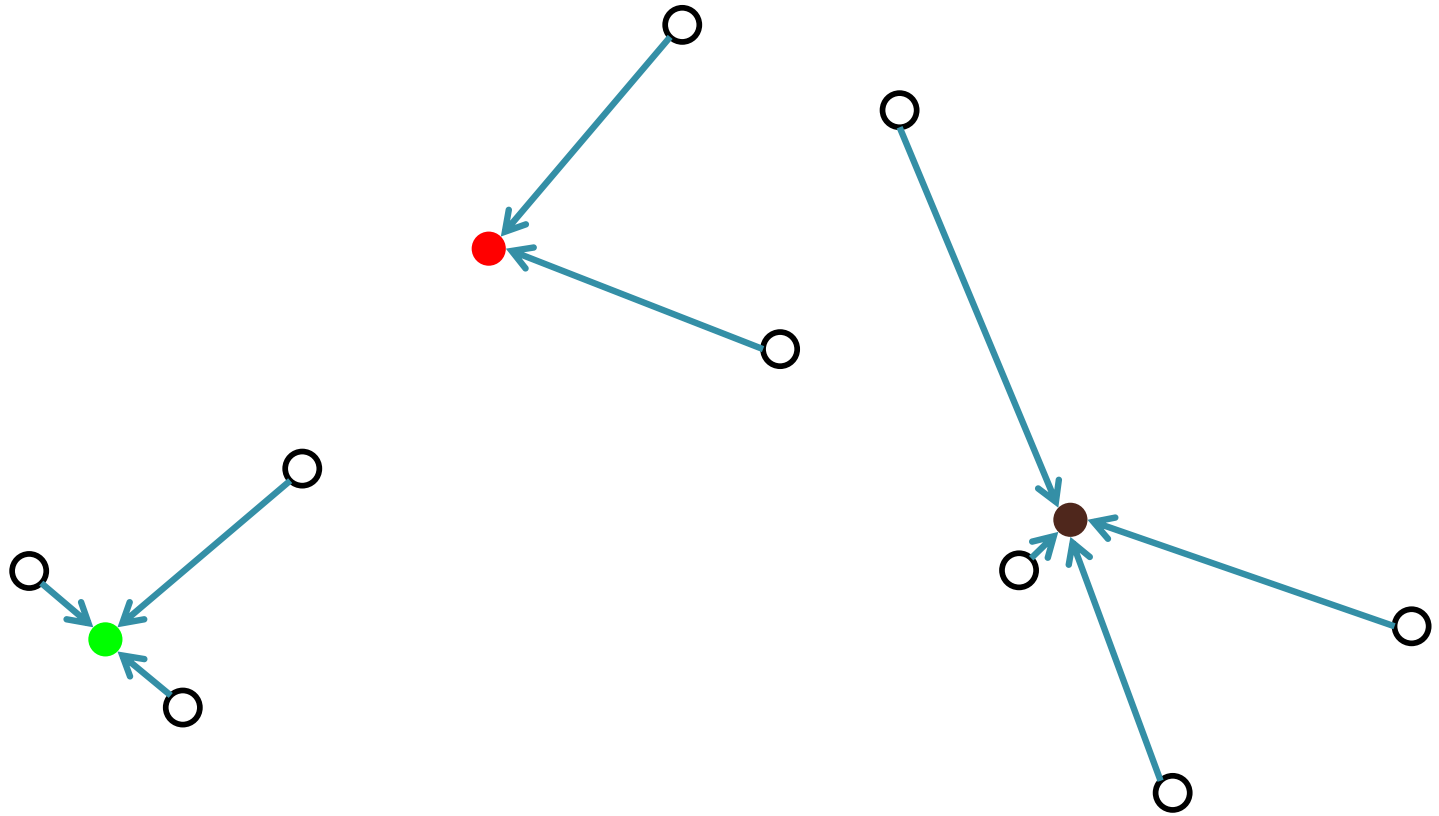
Recompute optimal centers given a fixed clustering



Slide courtesy of Nina Balcan

# Lloyd's method: Random Initialization

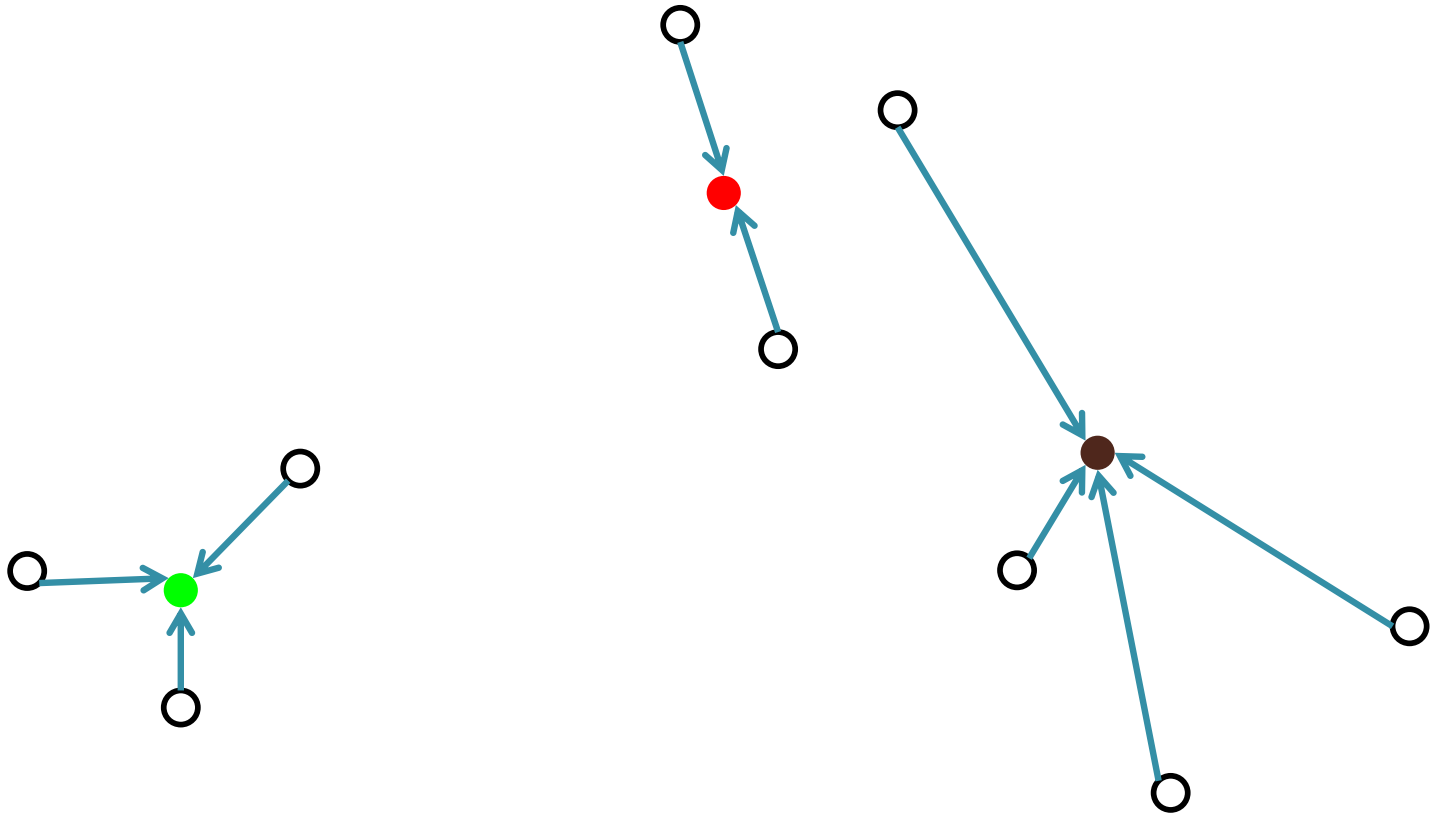
Assign each point to its nearest center



Slide courtesy of Nina Balcan

# Lloyd's method: Random Initialization

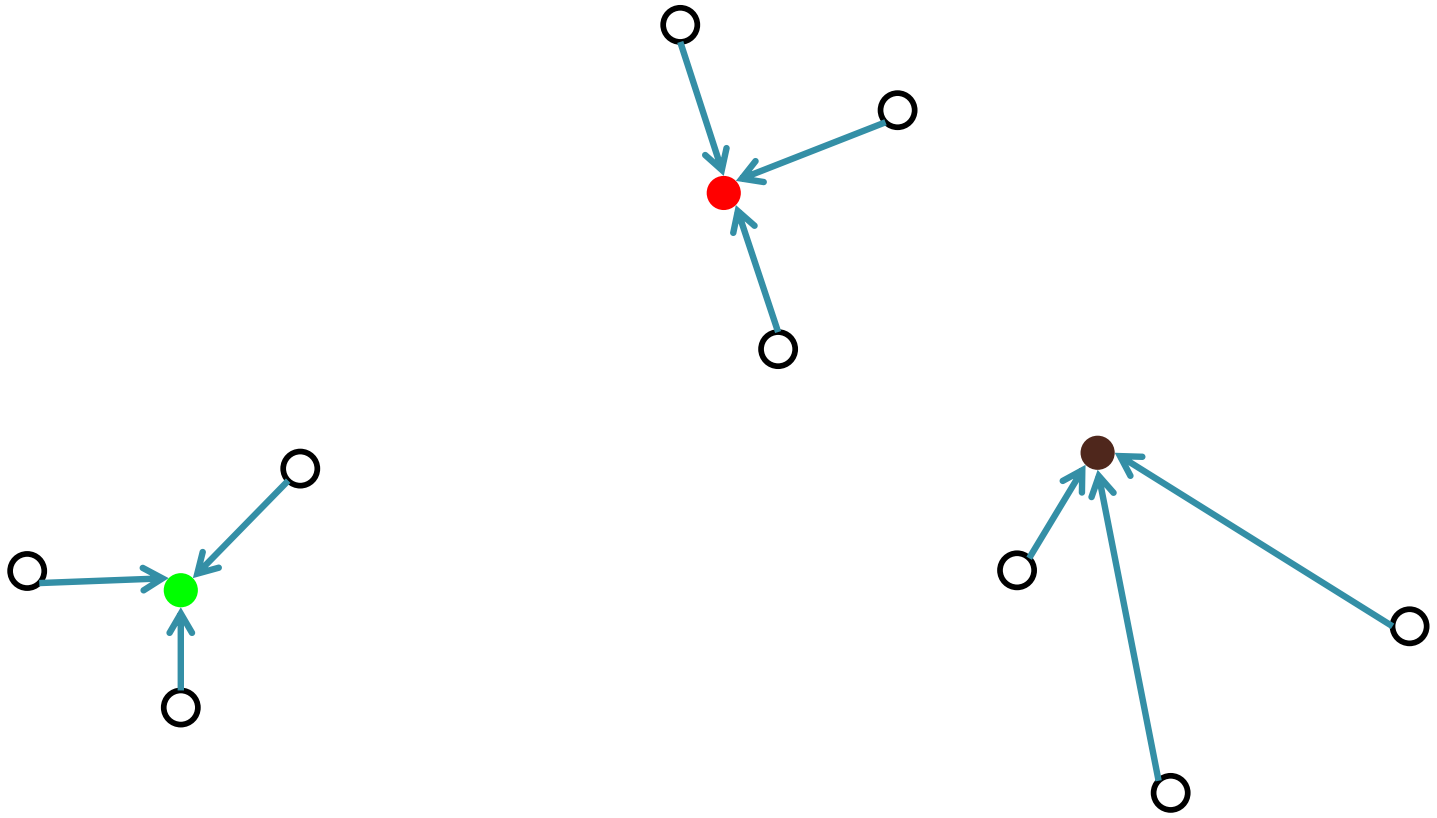
Recompute optimal centers given a fixed clustering



Slide courtesy of Nina Balcan

# Lloyd's method: Random Initialization

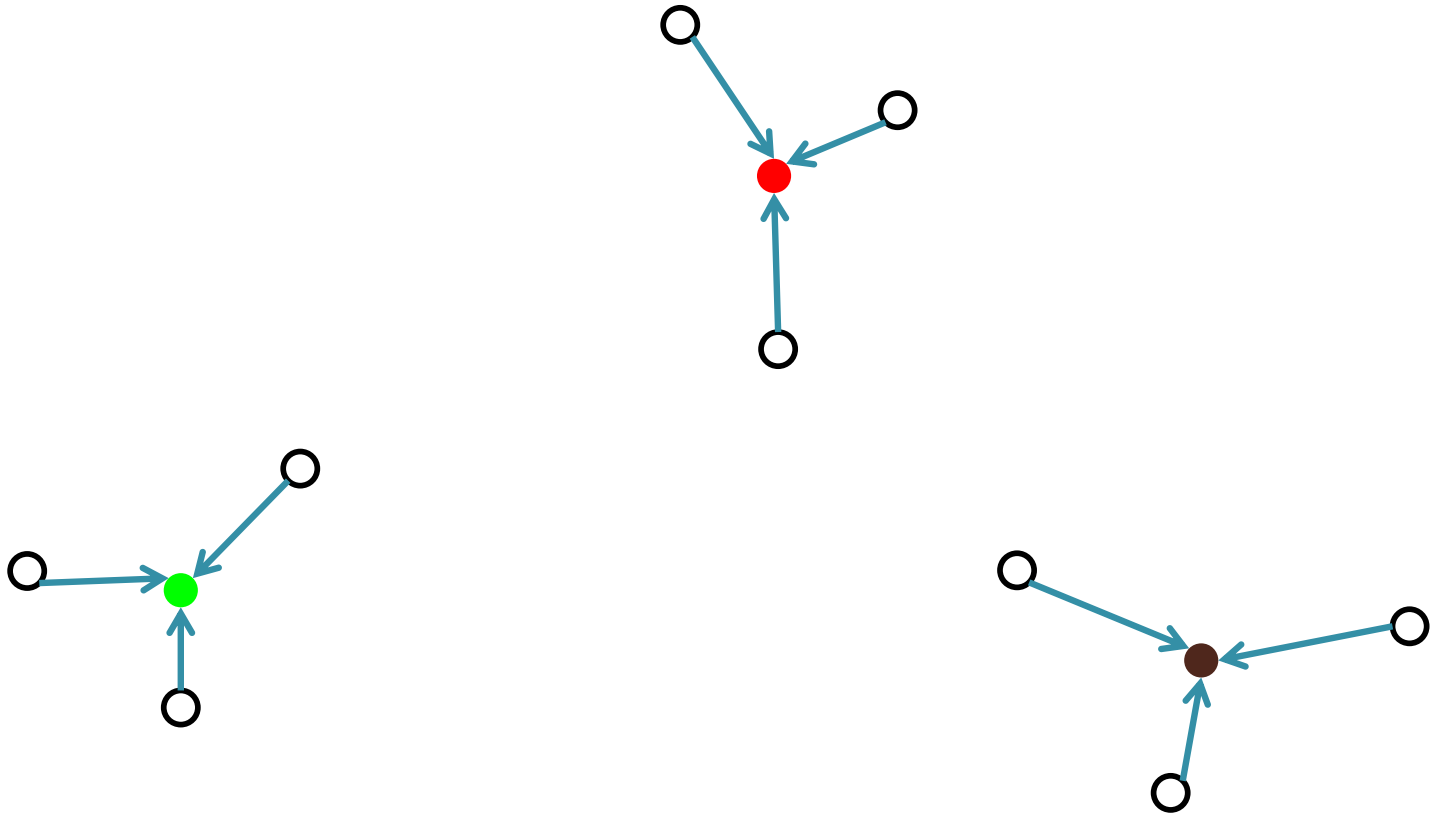
Assign each point to its nearest center



Slide courtesy of Nina Balcan

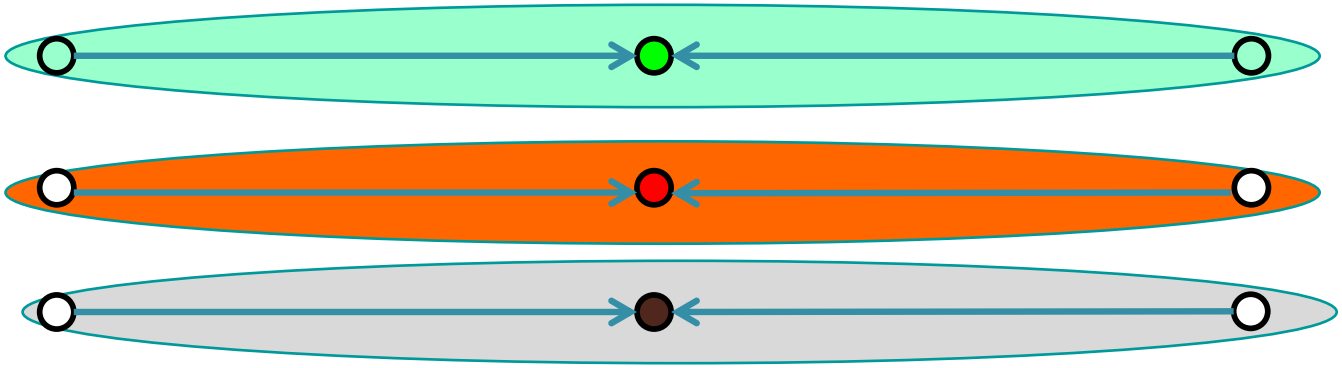
# Lloyd's method: Random Initialization

Recompute optimal centers given a fixed clustering



Get a good quality solution in this example.

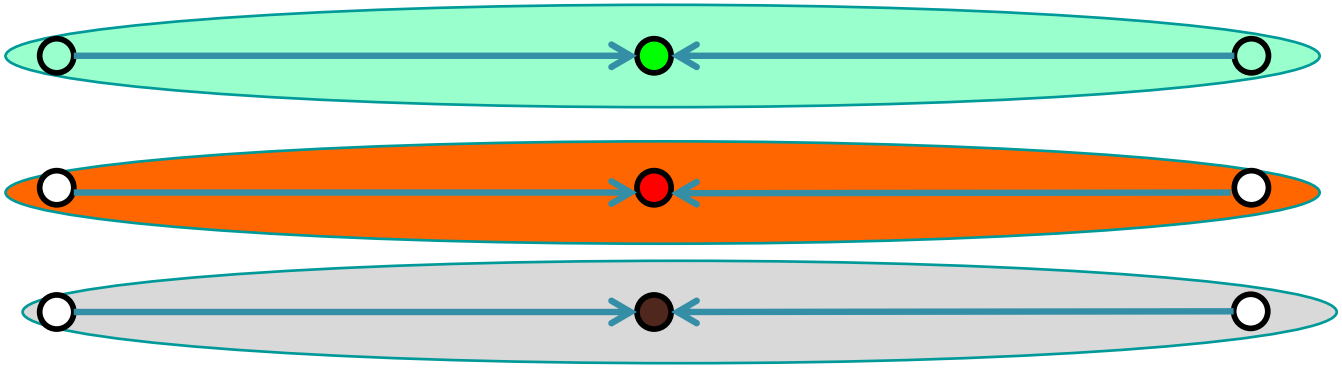
# Lloyd's method: Performance



It always converges, but it may converge at a local optimum that is different from the global optimum, and in fact could be arbitrarily worse in terms of its score.

Slide courtesy of Nina Balcan

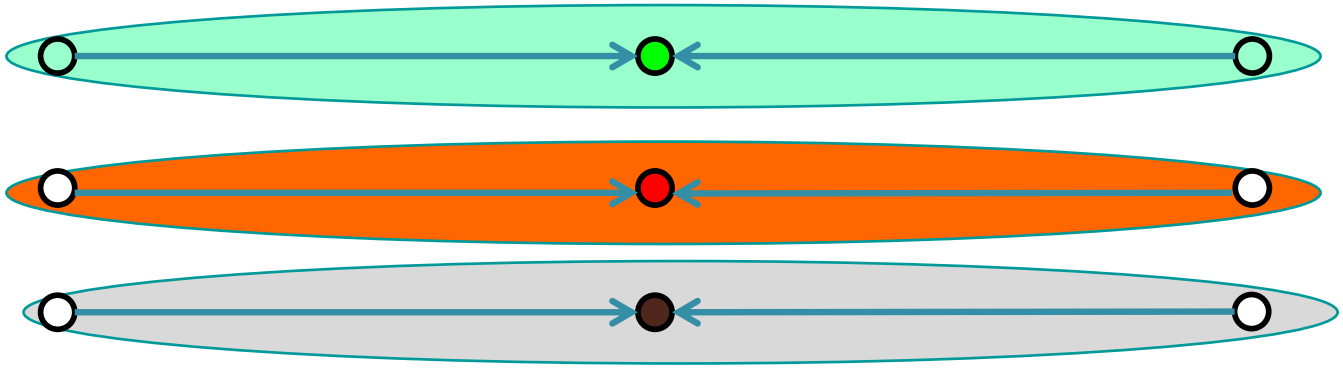
# Lloyd's method: Performance



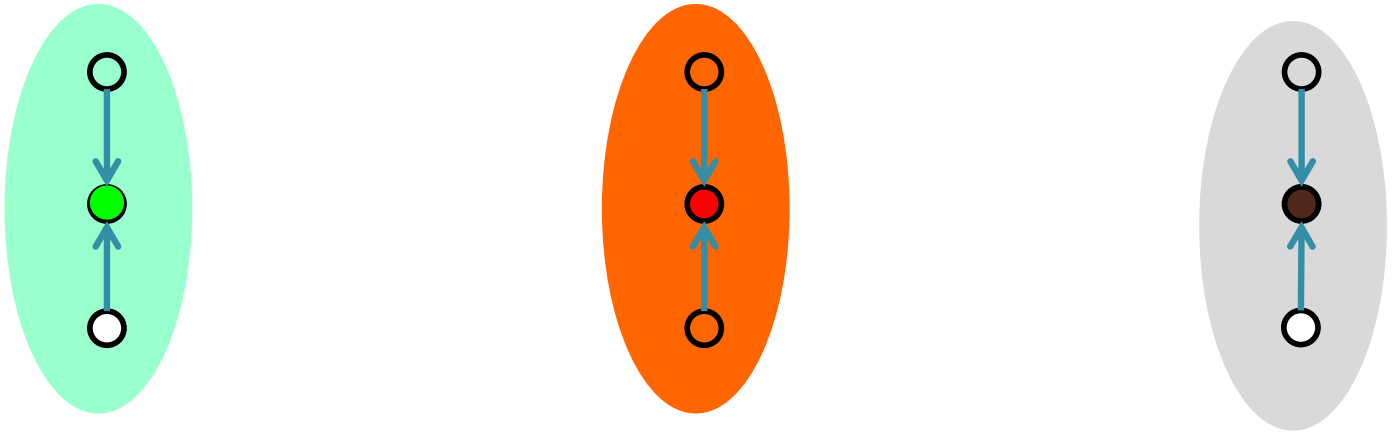
Local optimum: every point is assigned to its nearest center and every center is the mean value of its points.

Slide courtesy of Nina Balcan

# Lloyd's method: Performance

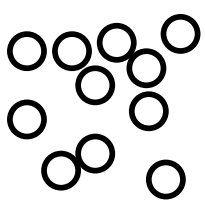


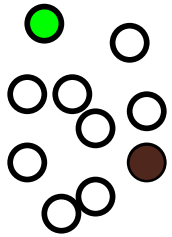
.It is arbitrarily worse than optimum solution….

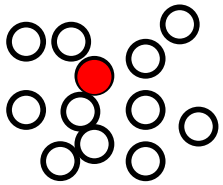


Slide courtesy of Nina Balcan

# Lloyd's method: Performance



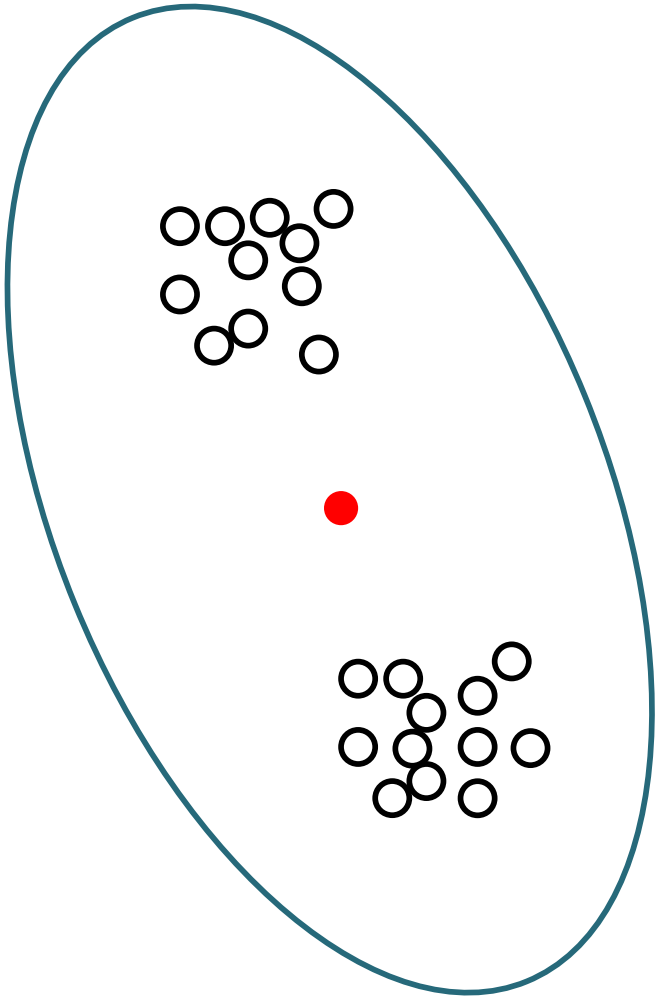


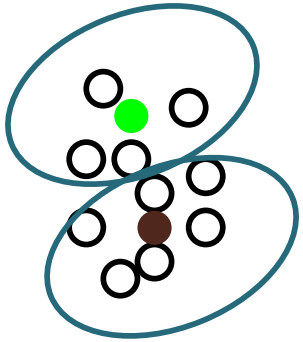


This bad performance, can happen even with well separated Gaussian clusters.

Slide courtesy of Nina Balcan

# Lloyd's method: Performance





This bad performance, can happen even with well separated Gaussian clusters.

Some Gaussian are combined……

# Learning Objectives

**K-Means**

*You should be able to…*

1. Distinguish between coordinate descent and block coordinate descent
2. Define an objective function that gives rise to a "good" clustering
3. Apply block coordinate descent to an objective function preferring each point to be close to its nearest objective function to obtain the K-Means algorithm
4. Implement the K-Means algorithm
5. Connect the nonconvexity of the K-Means objective function with the (possibly) poor performance of random initialization