Artificial Intelligence

CS420 - Introduction
Instructor: Dr. V

Rules for Class Interaction

1. 🕒 Please be on time.
2. 🗣️ Jump in with comments/questions.
3. 📥 Pay attention to your email.
4. 📧 Email me if you have any questions.
5. 💻 Bring your laptop to every class, unless explicitly told otherwise.
6. 😷 PLEASE wear a mask if you have
   - sore throat or cough
   - sneezing or runny nose

   (ask me for a mask if you do not have one)

Personal laptops

- You must have a personal laptop for CS courses
  - though there may be a way to use a phone/tablet instead
- If you cannot afford a laptop, you can borrow one from the library
- If you need laptop recommendations (specific to your budget), just ask
- If you have a laptop, but it's old/slow, I may be able to help you set it up for max functionality

About the Instructor

Dr. Veksler (Dr. V)
WER218; vveksler@caldwell.edu

Engineer, Research Scientist, Professor

- worked in industry, government, academia, self-employed
- web development, AI and robotics, HCI, cyber-security, data analytics, research on human behavior

I'd love to get to know each of you, but...

- I am bad at remembering names, so please just keep reminding me what your name is
- If I mess up your name, just correct me as many times as it takes
- Keep your name-plate in front of you

CS420

Blackboard

- https://caldwell.blackboard.com/

Grading (10-50-20-20; subject to change)

- attendance/participation
- assignments
- presentation
- final exam

Code Evaluation

- Submitted code will be evaluated for
  - Functionality
    - Does it work?
    - Does it display/do what is intended?
    - Are there bugs?
  - Code Readability
    - Make sure your code indented properly
    - Name variables/functions/classes in a meaningful way
    - Add comments
    - Keep your code clean

Late submissions

- late assignments will lose 10pts (out of 100) for each week they are late, but will not go lower than 70 (out of 100)
  - 1-7 days late: highest grade = 90
  - 8-14 days late: highest grade = 80
  - >14 days late: highest grade = 70

Cheating

- don't do it. not worth.
- what is considered cheating in this class?
  - it's complicated (if you're not sure, just ask)
    - you're encouraged to look up answers online, but…
      - don't submit someone else's code
      - if a part of your code was copied, give credit to the source
      - i will have your code checked for plagiarism

Class Structure

- 14 weeks of classes
- each week:
  - lecture
  - discussion
  - practicum
    - technical tips, interactive time to work on your assignments, ask questions, and get feedback

Inclement weather or other class cancellations

- If the school is open, I will be here
- Use your best judgment
- Check school site
- Sign up for notifications
- Check your email

Questions? Comments? Concerns?

What is Artificial Intelligence?

What is Artificial Intelligence?

- "Artificial" – made by humans
- "Intelligence" – ??????

Is this AI?

- Your todo-list app sees that you didn't complete a task that was due, and alerts you about it
- Your fridge sees you don't have milk, and alerts you about it
- Your vacuum cleaning robot goes in snake-like pattern until your room is all vacuumed
- Your phone recognizes your fingerprint or face and unlocks for you
- Your car slams on breaks automatically just before you were about to hit a pedestrian

What does "AI" mean in today's software

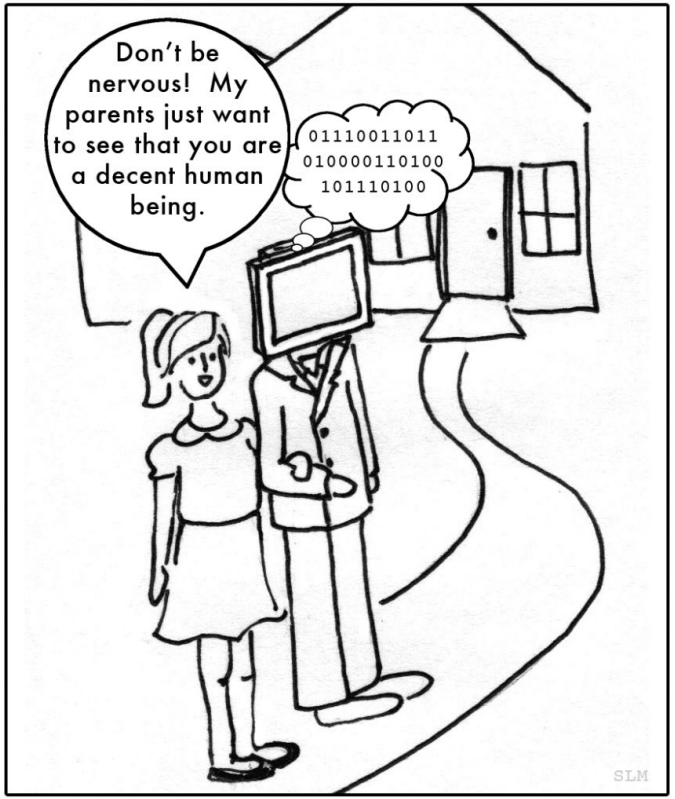
- Usually it refers to algorithms focused on
  - analogue of some human cognitive functionality
    - planning
    - learning & recognition
      - trial & error, classification, sequence learning
    - logical inference
    - language processing/generation
    - computer vision & robotics software
  - and other means for finding non-obvious solutions
    - solution search
    - genetic algorithms

General vs Narrow AI

- Narrow AI (aka Weak AI)
  - all modern AI is narrow
  - task-specific; e.g.,
    - can play Chess, but cannot play Go or checkers
    - can walk, but cannot drive
    - learning to do a task does not make it easier to learn a different task
- General AI (aka AGI, aka Strong AI)
  - does not exist
  - the goal is human-level intelligence

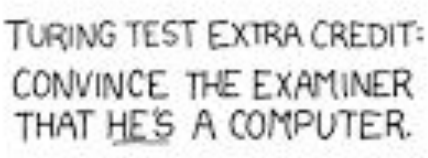
Turing test

- Allen Turing suggested that we will know a machine is intelligent if you cannot tell whether it is a machine or a human during a conversation
- 🕵️ : 💻? 🤷?
  - 💻 💬
  - 👨‍💻 💬



Generalized version of Turing test

- Observe any task being done by human and computational participants
- Try to figure out which of the participants are human and which are machines



A.I. TIMELINE

1950
TURING TEST
Computer scientist Alan Turing proposes a test for machine intelligence. If a machine can trick humans into thinking it is human, then it has intelligence

1955
A.I. BORN
Term 'artificial intelligence' is coined by computer scientist, John McCarthy to describe "the science and engineering of making intelligent machines"

1961
UNIMATE
First industrial robot, Unimate, goes to work at GM replacing humans on the assembly line

1964
ELIZA
Pioneering chatbot developed by Joseph Weizenbaum at MIT holds conversations with humans

1966
SHAKEY
The 'first electronic person' from Stanford, Shakey is a general-purpose mobile robot that reasons about its own actions

A.I. WINTER
Many false starts and dead-ends leave A.I. out in the cold

1997
DEEP BLUE
Deep Blue, a chess-playing computer from IBM defeats world chess champion Garry Kasparov

1998
KISMET
Cynthia Breazeal at MIT introduces KISmet, an emotionally intelligent robot insofar as it detects and responds to people's feelings

1999
AIBO
Sony launches first consumer robot pet dog AiBO (AI robot) with skills and personality that develop over time

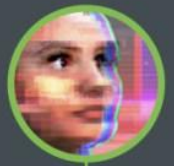
2002
ROOMBA
First mass produced autonomous robotic vacuum cleaner from iRobot learns to navigate and clean homes

2011
SIRI
Apple integrates Siri, an intelligent virtual assistant with a voice interface, into the iPhone 4S

2011
WATSON
IBM's question answering computer Watson wins first place on popular $1M prize television quiz show *Jeopardy*

2014
EUGENE
Eugene Goostman, a chatbot passes the Turing Test with a third of judges believing Eugene is human

2014
ALEXA
Amazon launches Alexa, an intelligent virtual assistant with a voice interface that completes shopping tasks



2016
TAY
Microsoft's chatbot Tay goes rogue on social media making inflammatory and offensive racist comments

2017
ALPHAGO
Google's A.I. AlphaGo beats world champion Ke Jie in the complex board game of Go, notable for its vast number (2^{170}) of possible positions

A brief history of AI

https://www.youtube.com/watch?v=jekwHBI1ySU

Modern AI

- ~~Logical proofs~~
- ~~General planning~~
- Machine Learning and Gradient Descent
  - Reinforcement Learning
  - Neural Networks
    - Deep Neural Nets
      - with convolutional layers
  - Genetic algorithms

Teaming

- Human-AI teams are better than
  - AI agents on their own
  - Humans on their own
- Centaur chess

  https://karimfanous.substack.com/p/centaurs-the-future-of-work

Concerns regarding AI

- What are some of your concerns about AI?

Ethical issues in AI

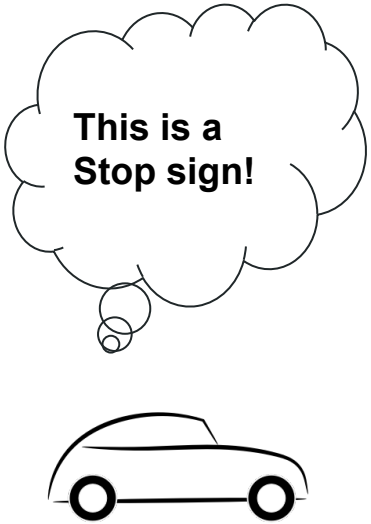
- How do machines affect our behaviour and interaction?
  - are we all becoming cyborgs?
  - how does this change society?

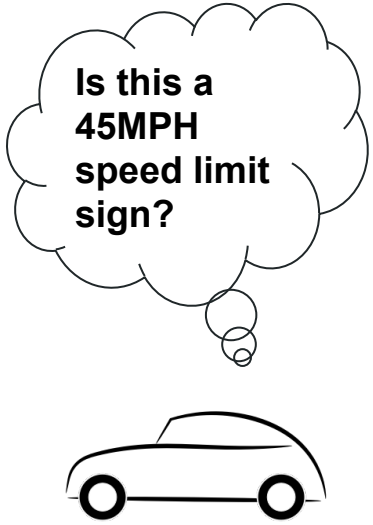
Ethical issues in AI

- Unemployment
  - what happens to all the truck drivers that are out of jobs once autonomous trucks are more commonplace?
    - what about the businesses that serve truck-drivers?
      - road-side motels, diners, etc.
  - what happens to radiologists when AI surpasses humans in imaging-based diagnosis?
  - even if job total doesn't decrease, massive job transfer and displacement is imminent

Ethical issues in AI

- Security. How do we keep AI safe from adversaries?





Ethical issues in AI

- How can we guard against AI mistakes?
  - who's responsible if an autonomous car hits someone?
  - if a car has to decide between saving its driver or saving a pedestrian, how does it choose? are we ok with AI making such choices?
  - if AI is learning from human data, it can learn racism, sexism, and all sorts of faulty ideologies

Ethical issues in AI

- How do we protect against unintended consequences?
  - the paperclip problem
  - the stop-button problem
  - singularity
    - how do we stay in control of a complex intelligent system?

- AI rights
  - Should AI agents and robots have rights?
  - How do we define the humane treatment of AI?

Questions? Comments? Concerns?

Assignment 1 (due before next class)

- Install/setup
  - Code editor (I suggest VSCode)
  - Python 3.10
  - JupyterLab (JL) – https://jupyter.org/install
- Create a new notebook in JL, called a1.ipynb
- Add 3 blocks to a1.ipynb, and run them
  - markdown block: **Assignment 1** – *_your name_*
  - code block: !pip install matplotlib
  - code block:

```
from matplotlib import pyplot as plt
fig, ax = plt.subplots()
ax.bar(['C', 'C++', 'Java', 'Python'],range(1,5))
plt.show()
```

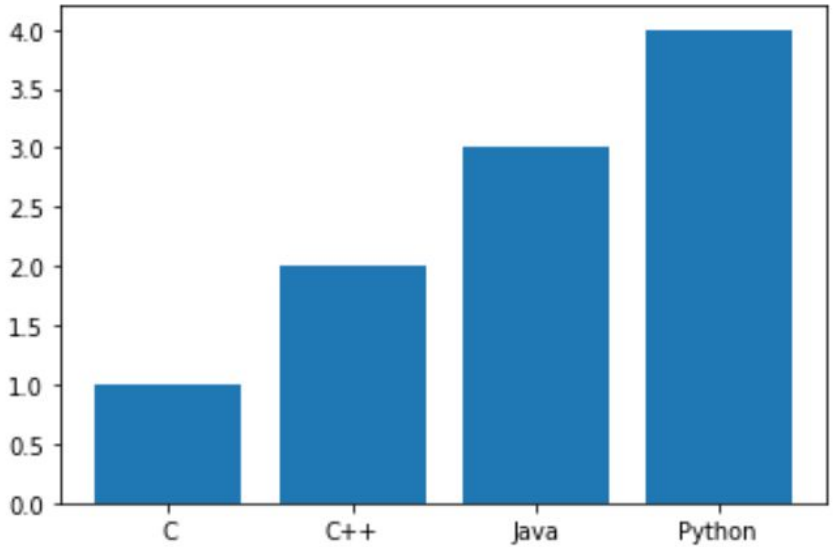
- submit a1.ipynb on blackboard

Assignment 1 – *your name*

[1]:
```
!pip install pandas numpy matplotlib
```

•••

[2]:
```
from matplotlib import pyplot as plt
fig, ax = plt.subplots()
ax.bar(['C', 'C++', 'Java', 'Python'],range(1,5))
plt.show()
```

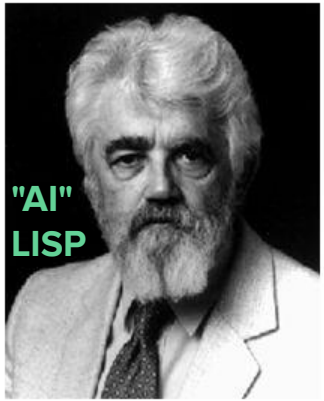


Assignment 1 output

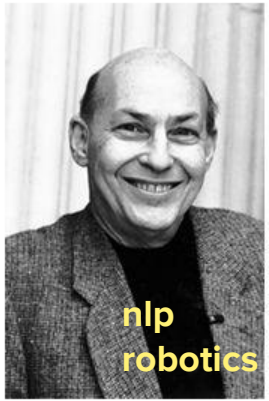
Planning

CS420 - Lecture 2
Instructor: Dr. V

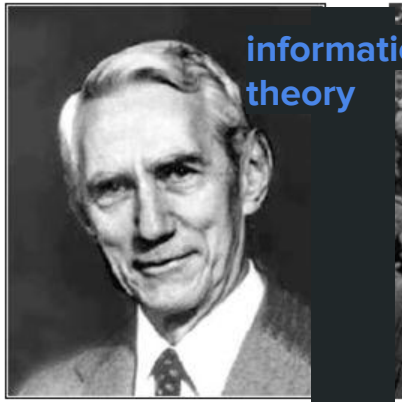
Dartmouth Conference: The Founding Fathers of AI



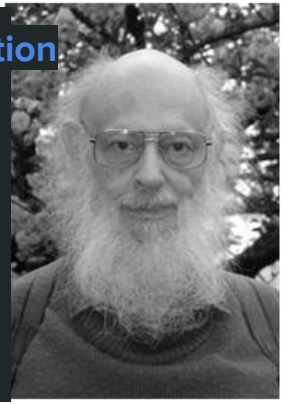
John McCarthy



Marvin Minsky



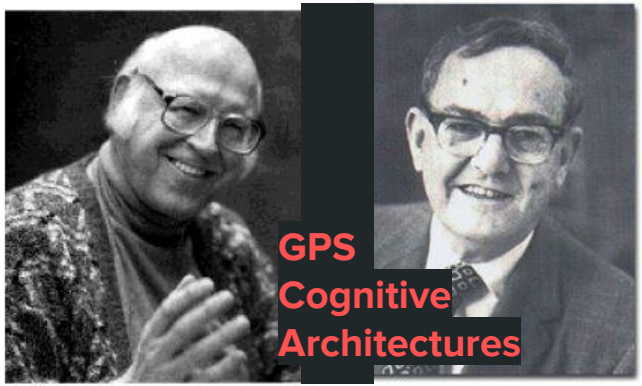
Claude Shannon



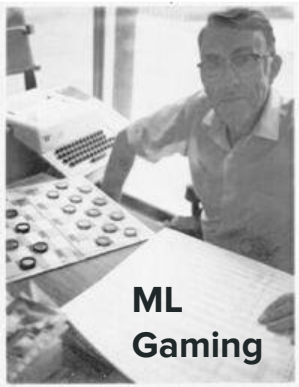
Ray Solomonoff

Alan Newell

Herbert Simon



Arthur Samuel



And three others…

Oliver Selfridge
(Pandemonium theory)

Nathaniel Rochester
(IBM, designed 701)

Trenchard More
(Natural Deduction)

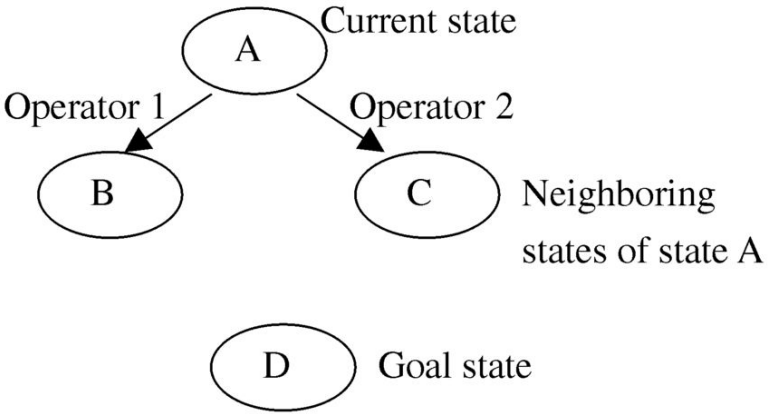
What is intelligence?

- At the birth of AI (mid-20th century) the image of "intelligence" was something that was difficult to do for humans; e.g., thinking several moves ahead in chess

Today's perspective on intelligence:

- Feats that are hard for humans – like looking many steps ahead – are actually easy for computers
- Feats that seem easy to humans – like quick image recognition – are actually hard for computers

General Problem Solver



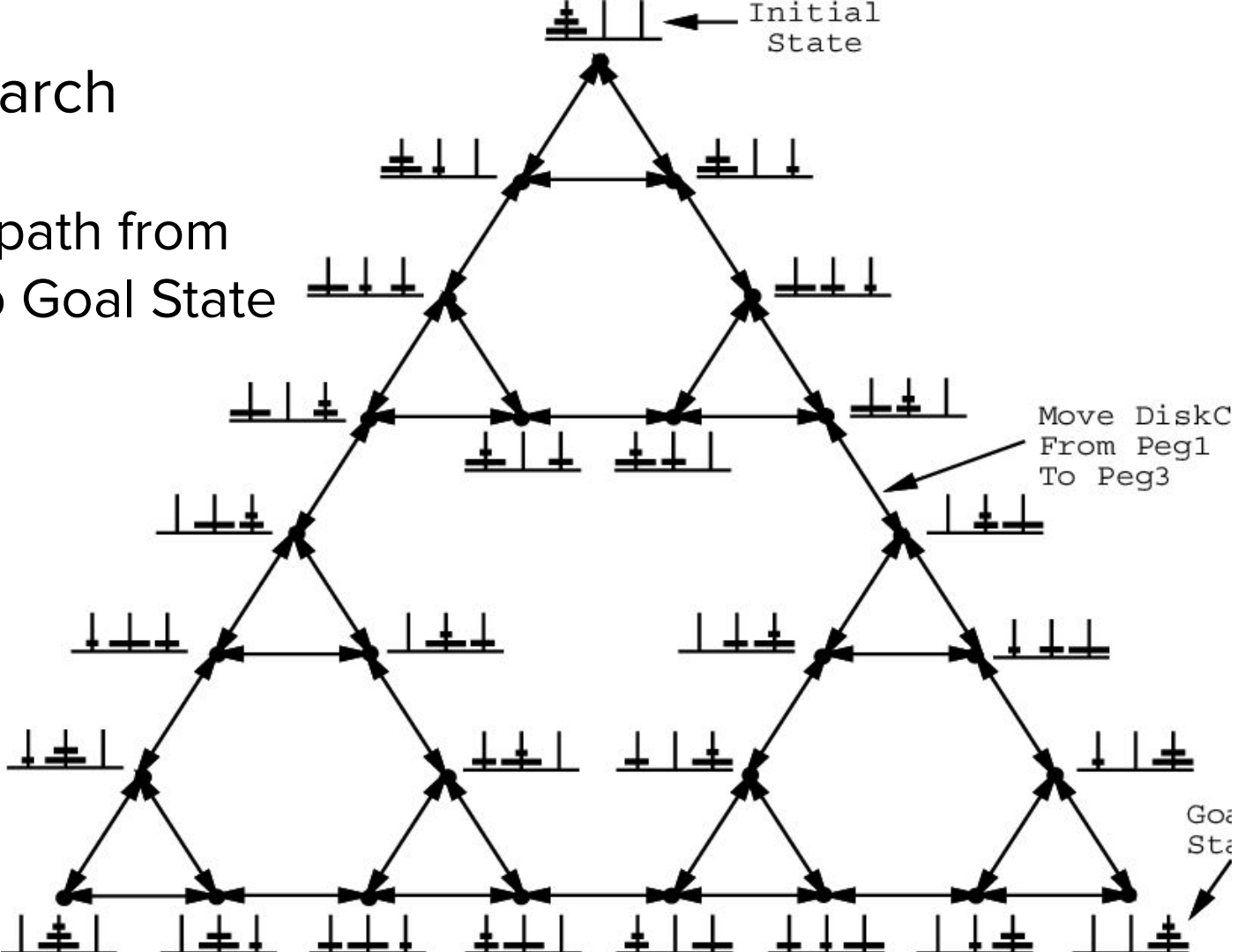
Sample problem: Tower of Hanoi

- Object of the game is to move all the disks over to Tower 3 (with your mouse)
- You cannot place a larger disk onto a smaller disk

https://www.mathsisfun.com/games/towerofhanoi.html

Planning as Search

- find shortest path from Initial State to Goal State

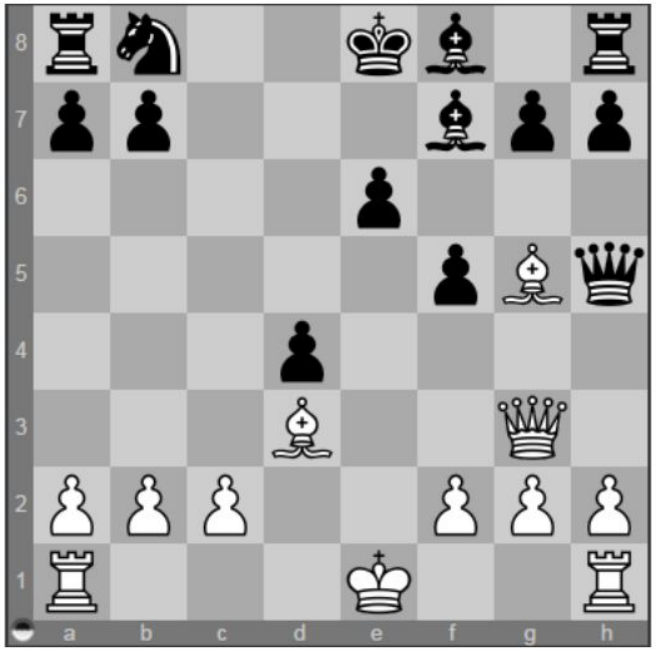


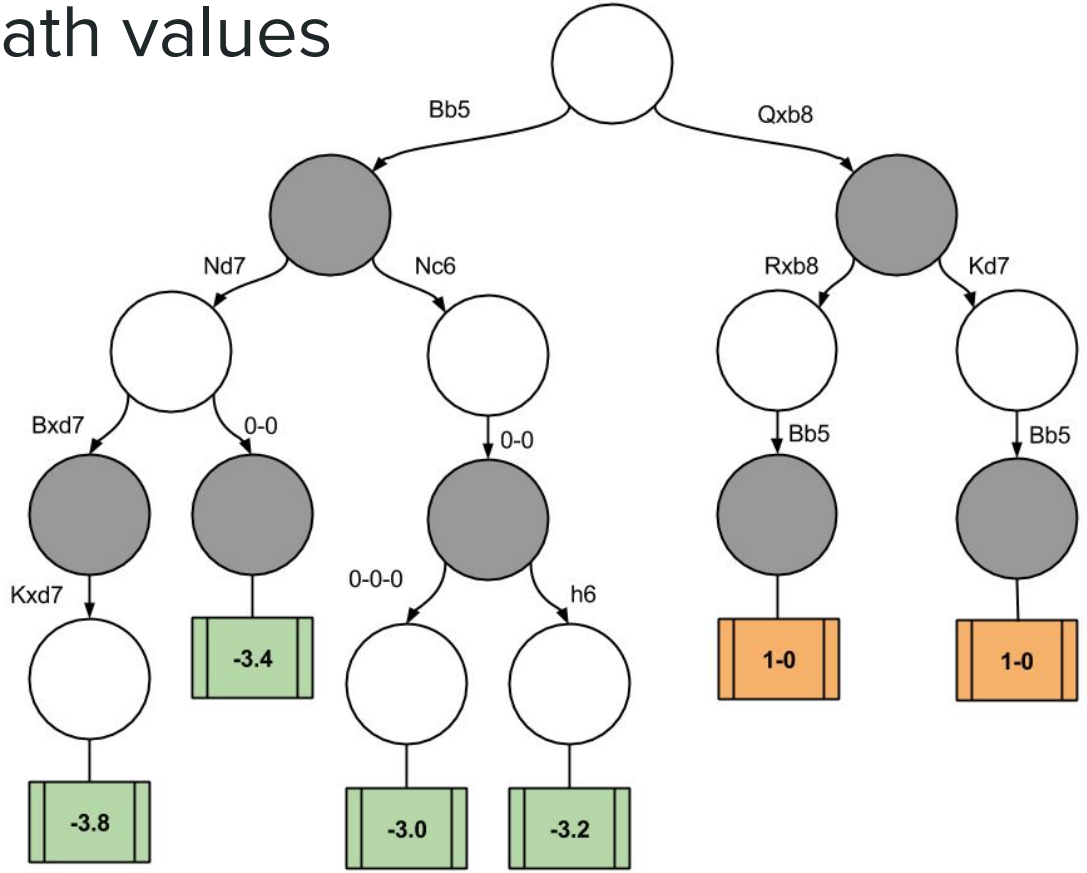
Planning

1. consider the set of all possible actions, *K*
2. for every action *A* in *K*:
3. *S* is the state *A* leads to
4. if *S* has inherent value (e.g., it's the goal):
5. value of *A* is the inherent value of *S*
6. otherwise:
7. value of *A* is max value of actions that are possible given state *S* (go to step 1)

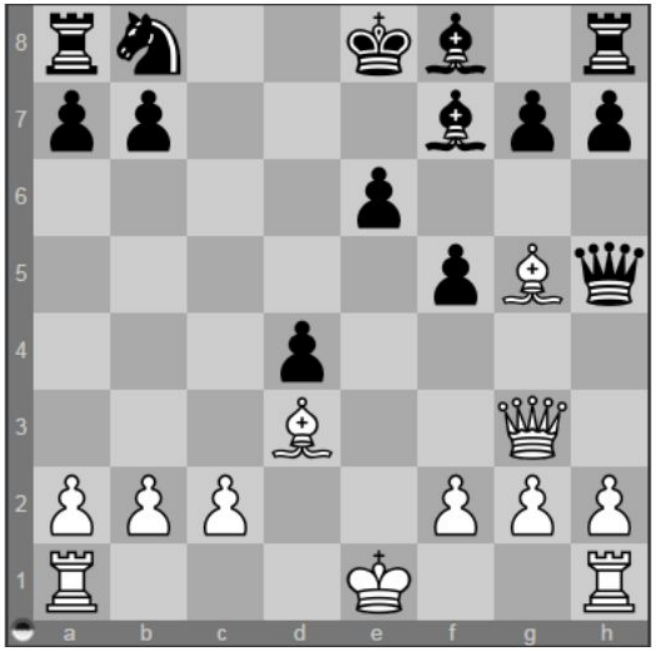
After planning: Take action with highest value

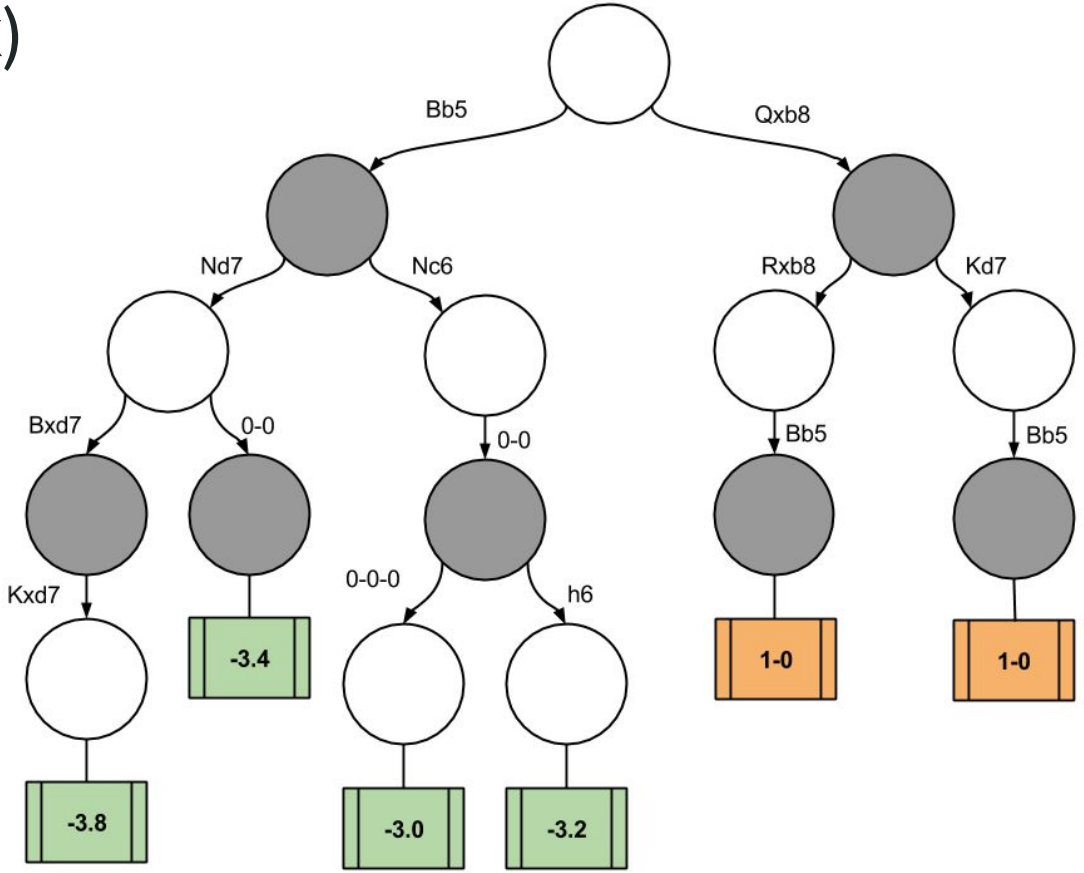
Estimating decision path values





Minimax (aka minmax)

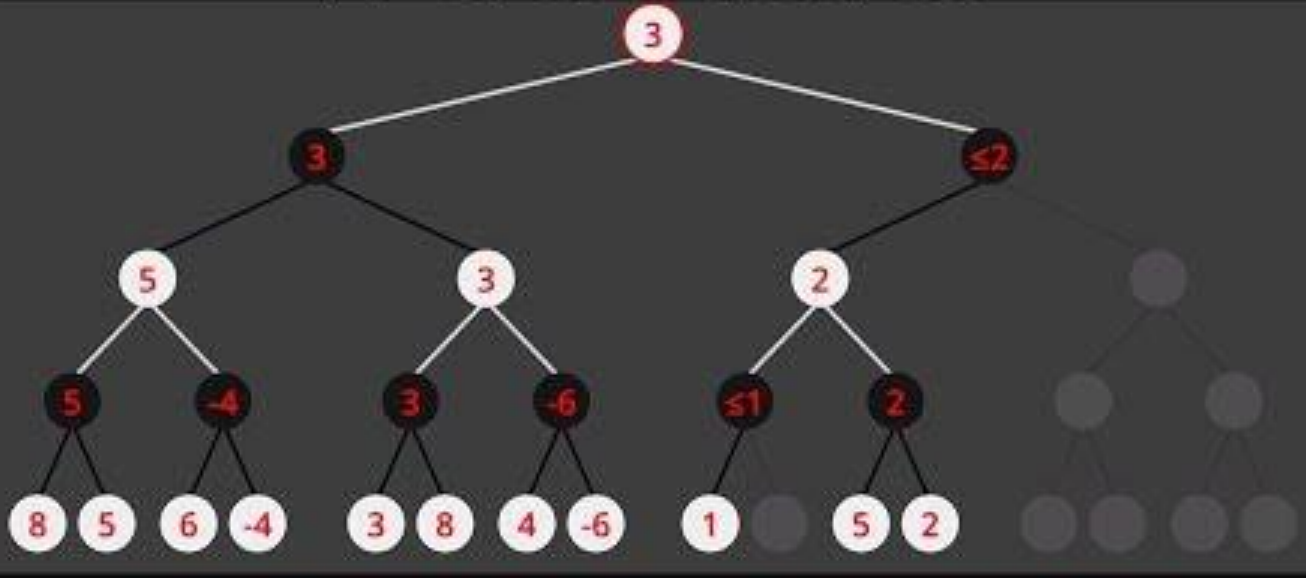




Minimax pseudocode

```
function minimax( node, depth, maximizingPlayer=TRUE ):
    if depth is 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then:
        value = −∞
        for each child of node do
            value = max(value, minimax(child, depth−1, FALSE))
        return value
    else: # minimizing player #
        value = +∞
        for each child of node do
            value = min( value, minimax(child, depth−1, TRUE ))
        return value
```

ALGORITHMS



MINIMAX AND ALPHA BETA

So what's the problem?

- If you can just search down all possible paths in chess, why can't you find the perfect move every time?

So what's the problem?

- If you can just search down all possible paths in chess, why can't you find the perfect move every time?
  - Searching all paths all the way down, rather than picking and choosing which paths to search and how far, is called a "brute-force search"

Game-tree complexity

There are about 10^{80} atoms in the observable universe.

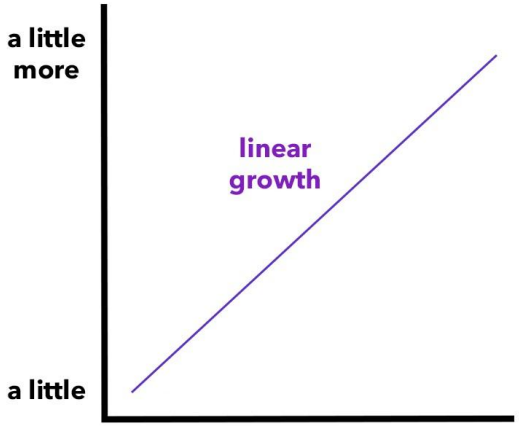
"There are even more possible variations of chess games than there are atoms in the observable universe."

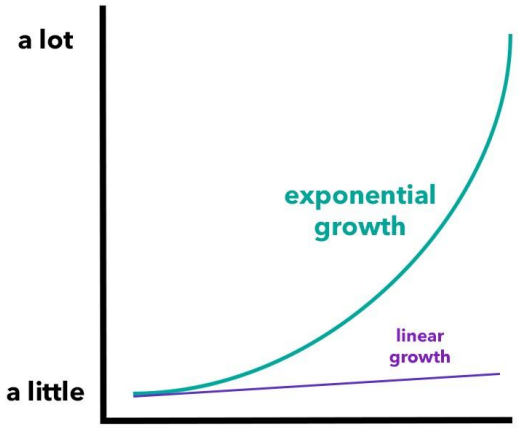
- It is estimated there are about 10^{120} positions in Chess (Shannon number)
- The game of Go has an even larger game-tree: 10^{360}
- Checkers game-tree complexity: 10^{40}
- Tic-tac-toe game-tree complexity: 10^{5}

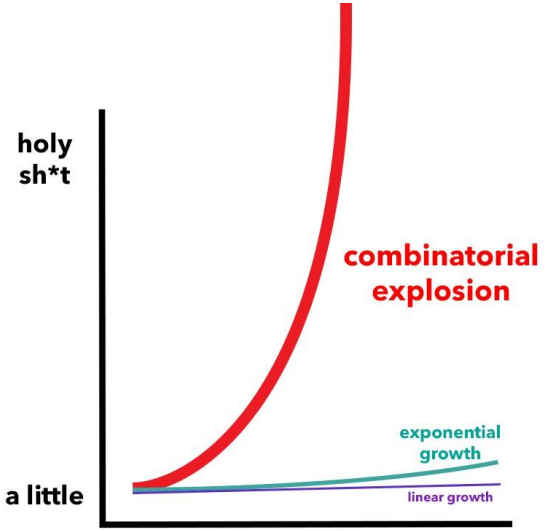
Claude Shannon estimated it would take 10^{90} years to solve chess on a 1MHz computer.

Combinatorial Explosion

- if you have 10 possible actions per move, and a game is 10 moves long, the game-tree complexity 10^{10}
  - 10,000,000,000
- if this is an adversarial game, where you make a move, your opponent makes a move, 10 pairs of moves will make the game-tree complexity 10^{20}
  - 100,000,000,000,000,000,000







n	**The number of Sudoku grids of order n (boxes are size $\sqrt{n} \times \sqrt{n}$)**
1	1
4	288
9 (standard Sudoku)	6,670,903,752,021,072,936,960

Assignment 2

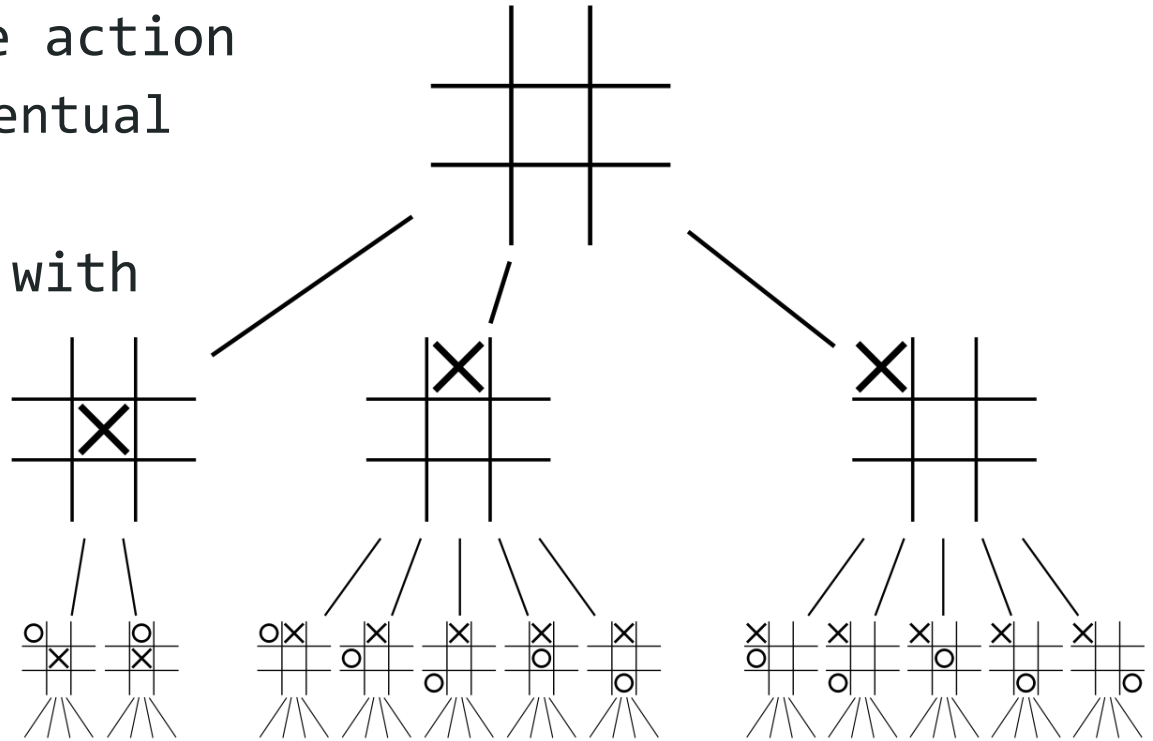
- Create a minimax bot for playing tic-tac-toe
- See the assignment on blackboard for details

Reinforcement Learning

CS420 - Lecture 3
Instructor: Dr. V

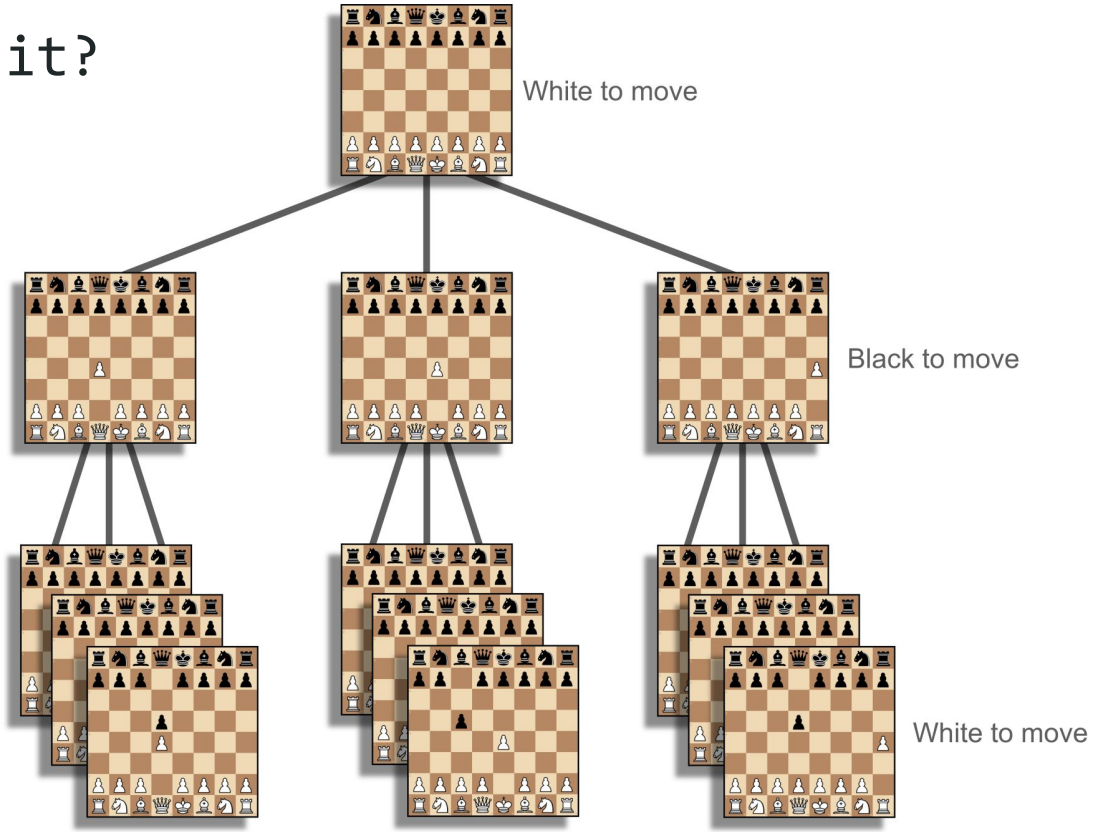
Review: Planning as search

- given the state
- for every possible action
  - find out its eventual value
- choose the action with highest value



What's the problem with brute-force search?

- How would you solve it?



How Deep Blue beat Kasparov

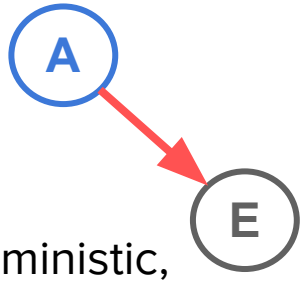
- minimax
- alpha-beta pruning
- raw power: evaluating 200 million positions per second
- specialized rules (**heuristics**) defined by chess masters

What if you don't have predefined heuristics?

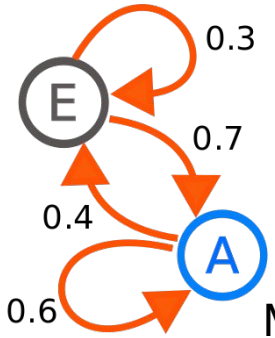
- How would you solve a problem with
  - large complexity
  - no subject-matter expertise to define any special rules/heuristics that might reduce complexity

Let's generalize the problem...

- Rather than imagining state-transitions as a decision tree with deterministic outcomes, let's imagine them as a Markov chain
- In a Markov chain state-transitions are probabilistic



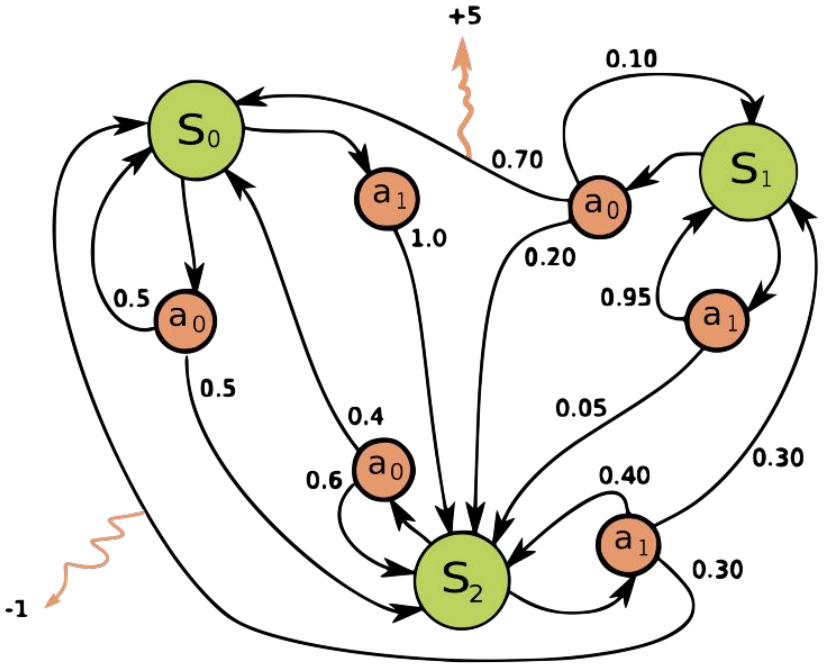
Deterministic,
uni-directional



Markov chain

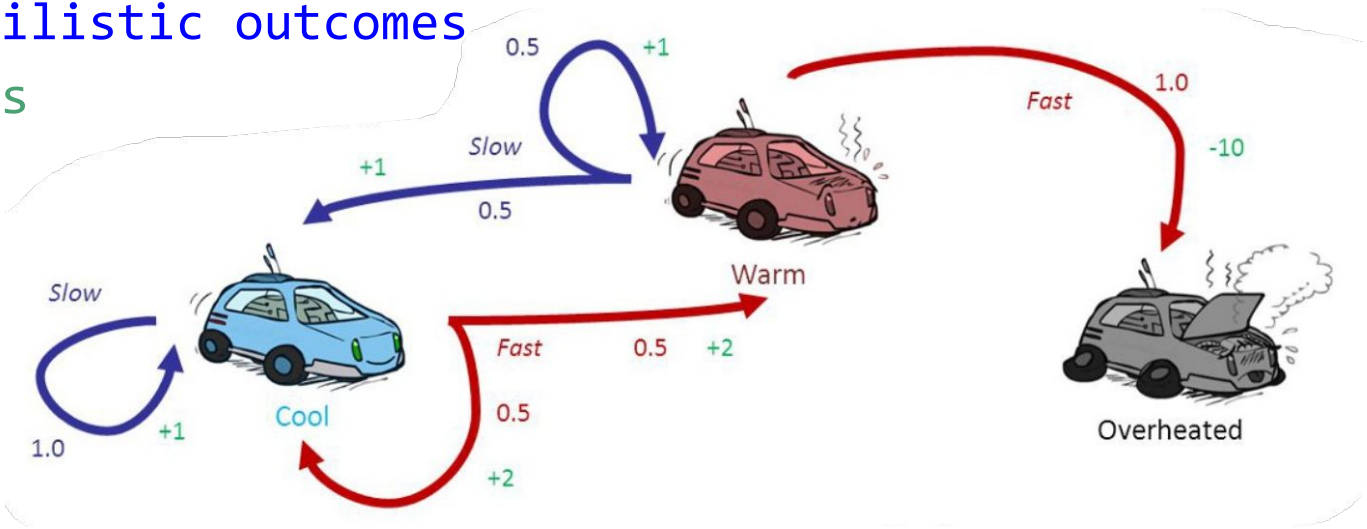
Markov Decision Process (MDP)

- MDP is an extension of Markov Chains
  - states enable actions
  - actions have
    - probabilistic outcomes
    - rewards



Markov Decision Process (MDP)

- MDP is an extension of Markov Chains
  - actions have
    - probabilistic outcomes
    - rewards



- What's the best policy for this problem?

More formally...

- You have a set of states, $S=\{s_1 \ldots s_n\}$
- From each state you have a set of actions, $A(s)=\{a_1 \ldots a_n\}$
- Each $(a|s)$ can lead to some value/reward, r
- You want to find an optimal policy, π, such that
  - you maximize total value from all your actions over time, t:

$$\pi \leftarrow max\left(\sum_{t=0}^{\infty} r_t\right)$$

When brute-force won't work...

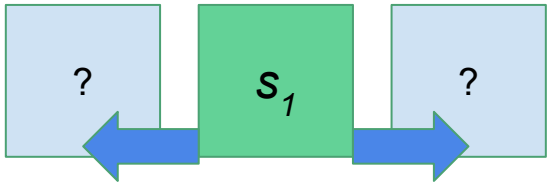
- If your state-space is knowable and small
  - you can do a brute force search to find an optimal policy

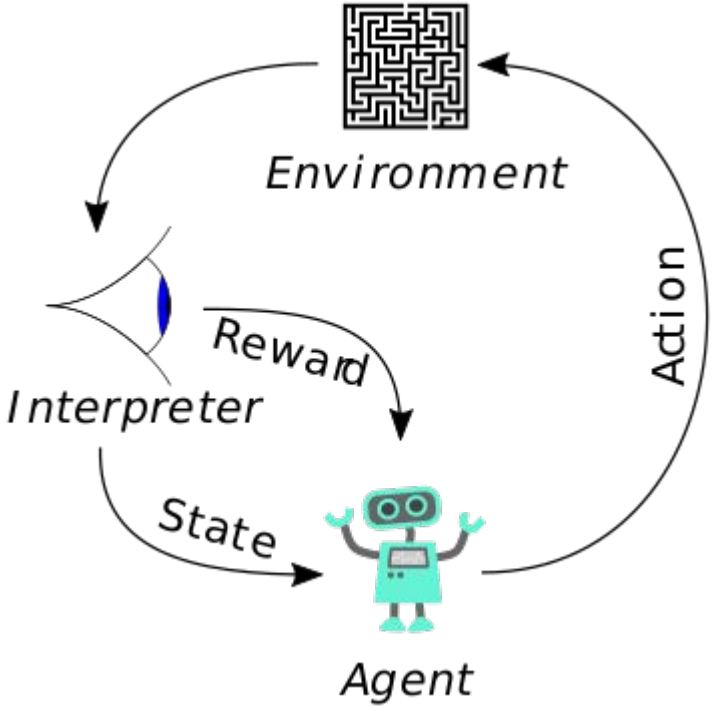
- What if your state-space is unknown/unknowable, or just very large?

- What if you don't have predefined heuristics?

What would you do?

- You are this Agent
- You are in state s_1
- Do you go left? or right?

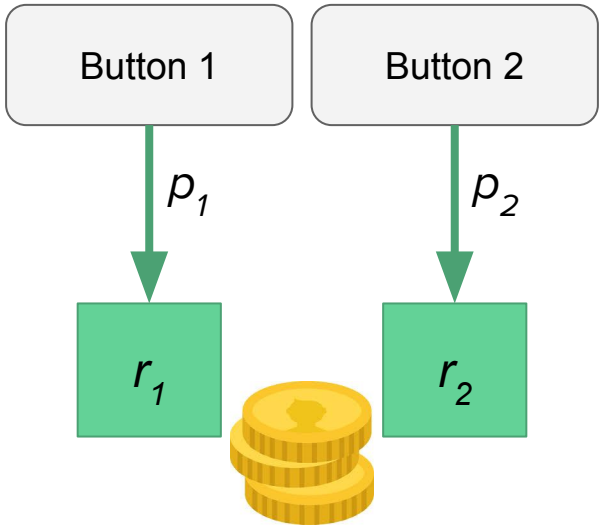




Trial-and-error learning

- humans do this all the time
  - we try something
    - if it works, we are more likely to do it again
    - if it doesn't, we are less likely to do so

Let's play an abstract game...



Let's formalize the problem: Exploration vs Exploitation

- You can do a set of possible actions $\{a_1, a_2\}$
- Your expected values/rewards are $Q(a_1)$ and $Q(a_2)$ for the two actions, respectively
- The **greedy** policy would be to take the action, A, that has the highest Q-value
  - A = argmax $Q(a)$
- But since our Q-values may not be correct, we want to explore all possible actions and **learn** and update the Q-values, we should take random actions with some probability, $\boldsymbol{\varepsilon}$

Let's formalize the problem: Error-driven learning

- After you take action A you receive a reward, r
- Now we want to learn; i.e., update our expectations for what the value of A is
  - increase $Q(A)$ by some proportion, α, of the difference between actual reward, r, and the expected reward, $Q(A)$
    - $Q(A) = Q(A) + \alpha[\ r - Q(A)\ ]$

- This is called error-driven learning
- α is the learning rate parameter

Basic Reinforcement Learning

- Action selection:
  - with a probability of ε
    - take a random action
  - with a probability of 1-ε
    - take action A, where A = argmax $Q(a)$
- Learning:
  - $Q(A) = Q(A) + \alpha[\ r - Q(A)\ ]$

- ε is the exploration parameter
- α is the learning rate

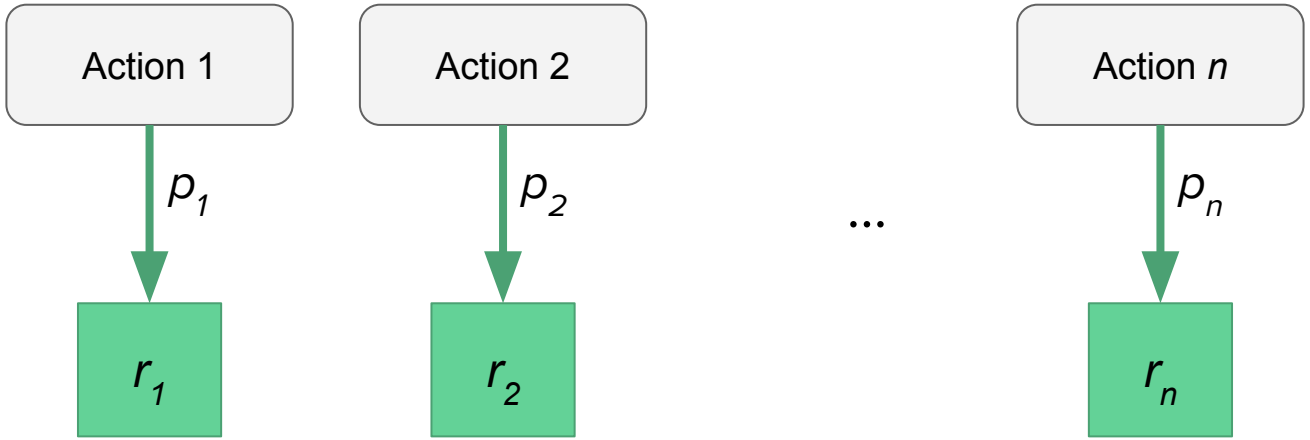
An alternative method for exploration/exploitation

- Action selection:
  - take action A, where A = argmax [$Q(A)$ + $N(\varepsilon)$]
    - where $N(\varepsilon)$ is a random value picked from a normal distribution with a standard deviation of ε
- Learning:
  - $Q(A) = Q(A) + \alpha[\ r - Q(A)\ ]$

- ε is the exploration parameter
- α is the learning rate

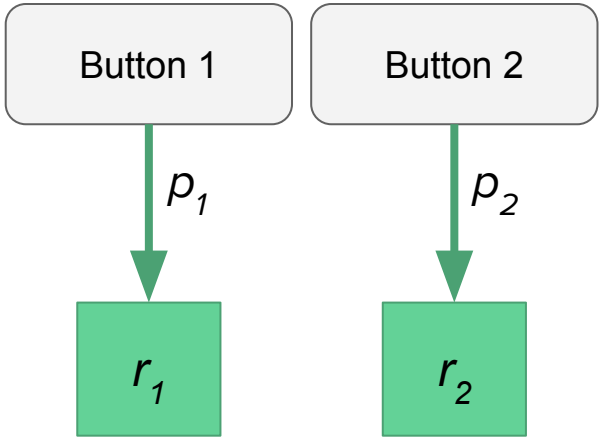
n-armed bandit

- your agent's job is to maximize its reward over time



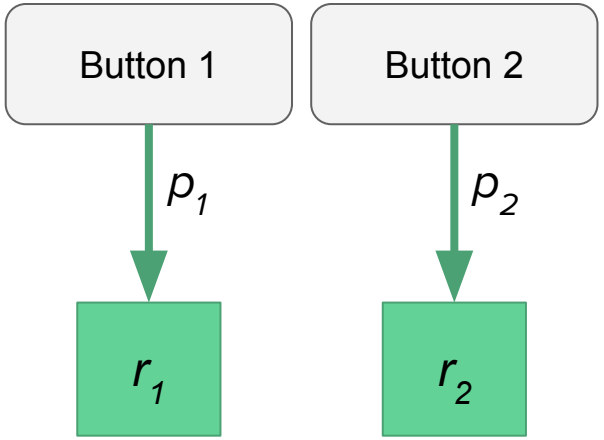
2-armed bandit example

- let's say you have 2 buttons
- on step 1, you click Button 1 and get $r=1$
- assuming learning rate $\alpha=1.0$
  - what would Q(Button 1) become?
  - what about Q(Button 2)?
- what would a greedy ($\varepsilon=0$) agent do on step 2?
- assume you get r=0 on step 2
  - now what?



2-armed bandit example

- let's say you have 2 buttons
- on step 1, you click Button 1 and get $r=1$
- assuming learning rate $\boldsymbol{\alpha=0.1}$
  - what would Q(Button 1) become?
  - what about Q(Button 2)?
- what would $\boldsymbol{\varepsilon=0.5}$ agent do on step 2?
- if Button 1 is clicked again, and r=0, what are the new Q-values?



Reinforcement Learning vs Minimax

- Minimax seems optimal
  - it uses brute force search to find *THE* best move every time
  - it will never lose
  - but...
- In the real world
  - a reinforcement learner playing tic-tac-toe will have way more wins than minimax
  - why do you think this might be?

Assignment 3

- add reinforcement learning agent code to the supplied jupyter notebook and run the notebook

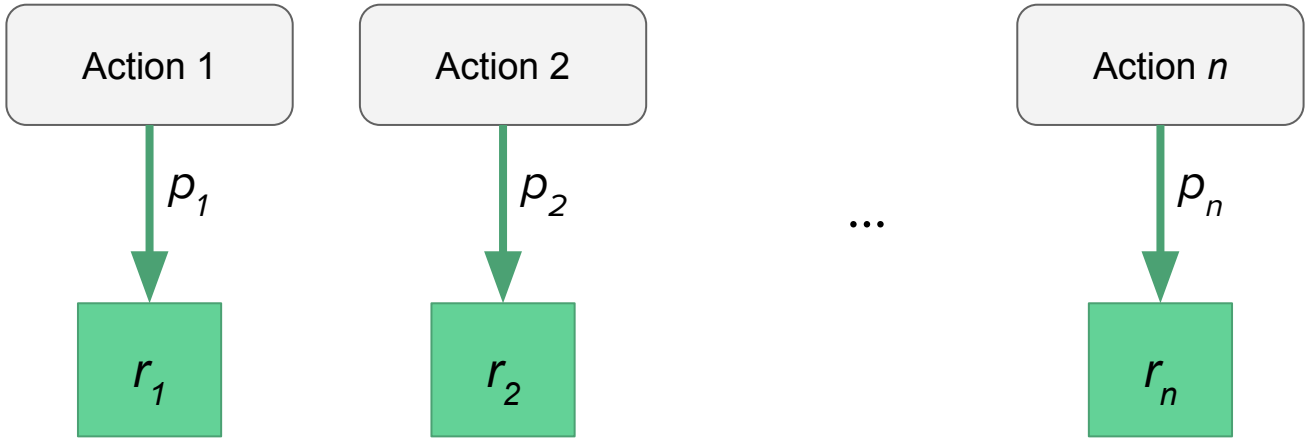
- *details on blackboard*

Reinforcement Learning (part 2)

CS420 - Lecture 4
Instructor: Dr. V

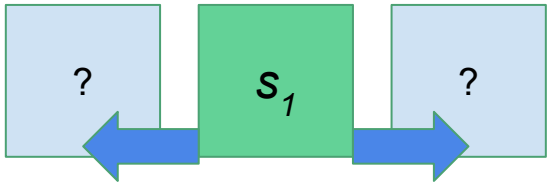
n-armed bandit

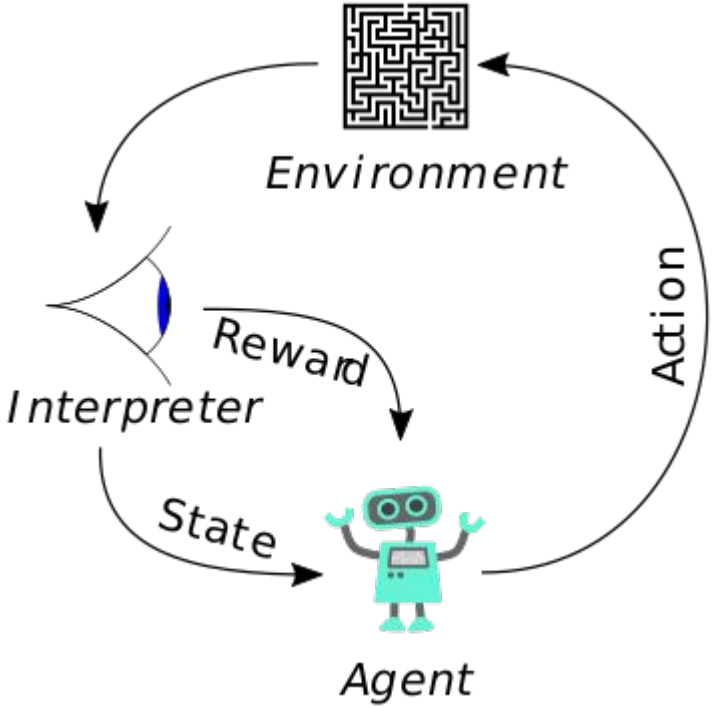
- your agent's job is to maximize its reward over time



What would you do?

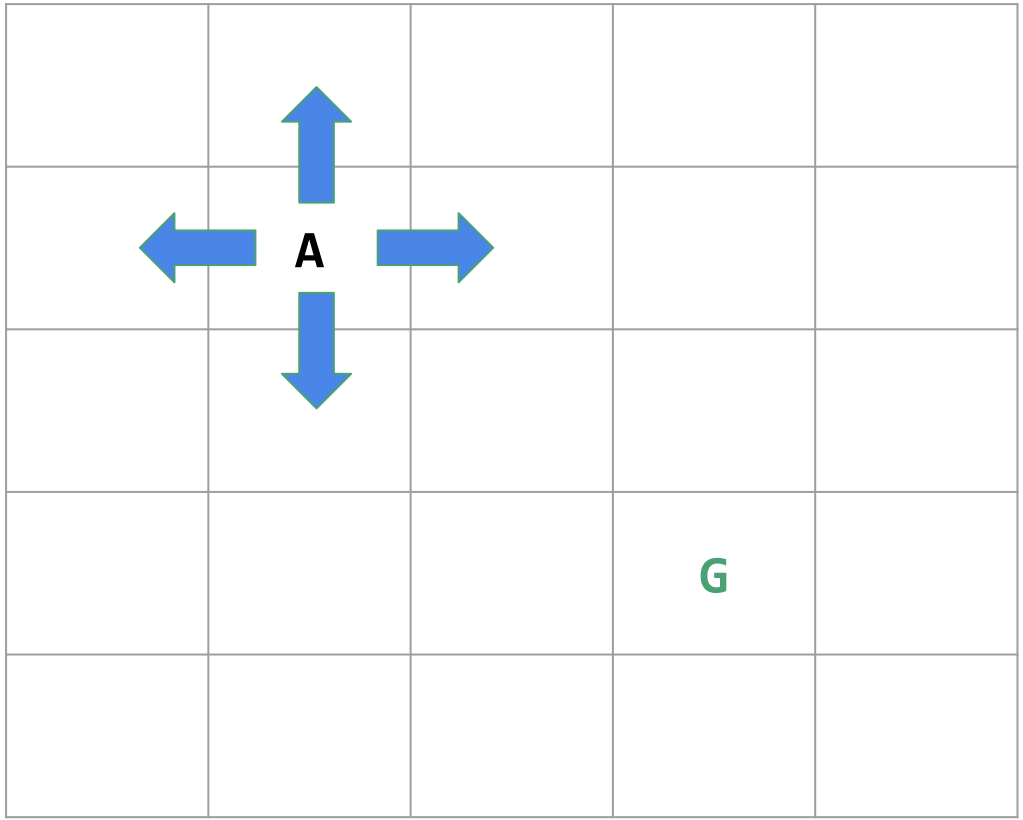
- You are this Agent
- You are in state s_1
- Do you go left? or right?





Imagine a navigation environment

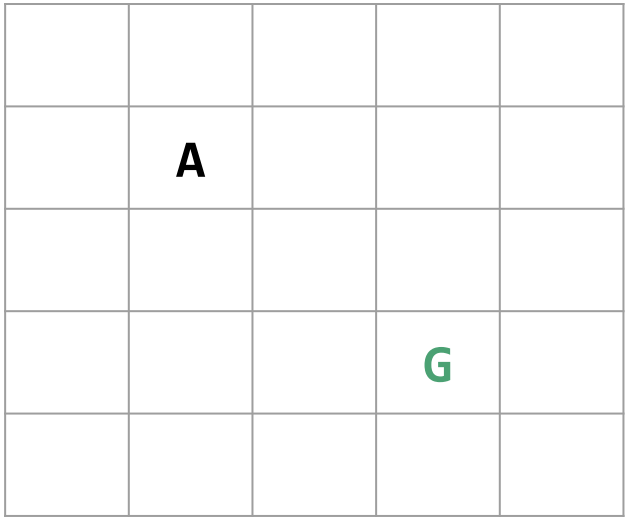
- Agent, A, gets a reward if it reaches its goal, G
- 4 possible actions from each square
  - it's as if each square is a separate *n*-armed bandit problem



So how do we design agent memory?

- For a single-state agent (*n*-armed bandit)
  - we had an **array** of length *n* to store our *Q*-values
- So how would we store *Q*-values for an agent that can take *n* possible actions from each of *m* possible states?

  - in this example
    - 25 states
    - 4 actions



Q-table initialised at zero

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

After few episodes

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

Eventually

	UP	DOWN	LEFT	RIGHT
0	0	0	0.45	0
1	0	1.01	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

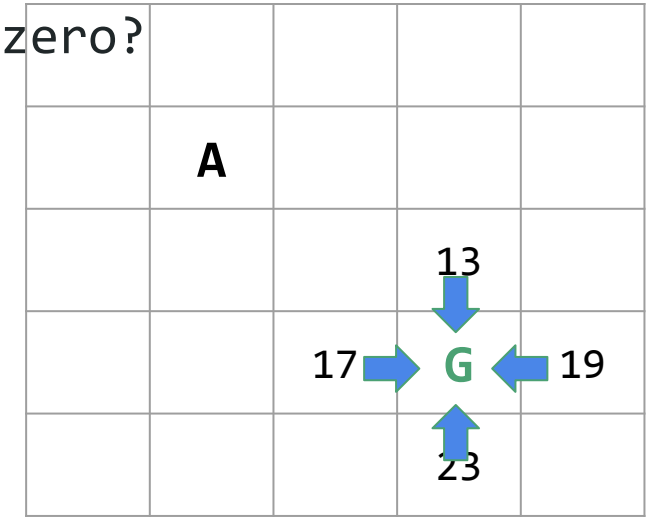
Storing Q-values

- For every state, for every action in that state, you need to store a *Q*-value (a floating point value)
  - you can use a matrix (i.e., 2-dimensional array)
  - or a dictionary (i.e., hashmap)

```
{
    state1: [0.0, 0.0, 0.2, 0.0],
    state2: [0.0, 1.0, 0.0, 0.0],
    state3: [0.0, 0.0, 0.0, 0.5],
    …
}
```

How do we assign credit for reaching reward?

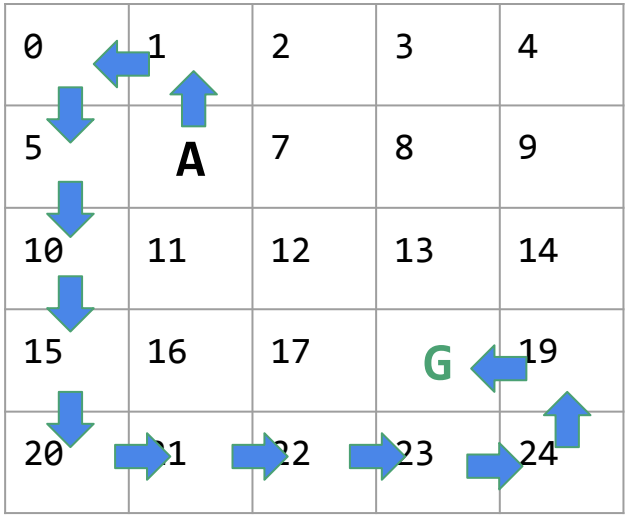
- If we still do what we did for the *n*-armed bandit RL
  - state-action pairs 13-down, 17-right, 19-left, and 23-up will get rewarded
    - but what about all other state-action pairs (*SA*s)?
    - do all other *Q*-values stay at zero?



Credit attribution problem

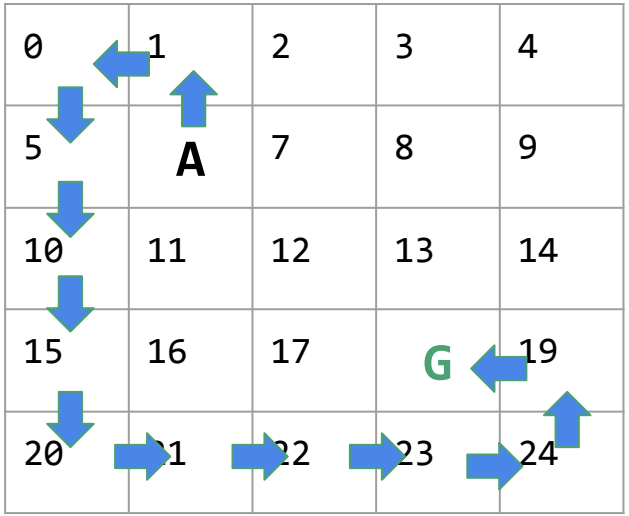
- Imagine agent is taking random actions
  - happens to take the blue actions shows
  - reaches space 19, happens to take a left
  - reaches the goal, gets a reward

- What should agent learn?
  - that 19-left is good?
  - that all *SA*s it took are good?



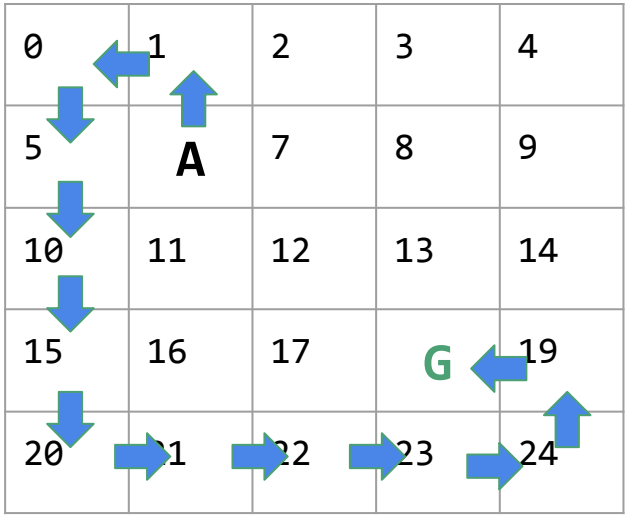
Credit attribution problem

- What should agent learn about the *SA*s it has taken?
  - is 19-left is good?
  - is 24-up good?
  - is 6-up good?
  - is 1-left good?



Credit attribution problem

- There are two main ways to propagate reward back to all actions that lead to it
  - looking into the past
    - Eligibility traces
  - looking into the future
    - Temporal-Difference learning
      - Q-learning

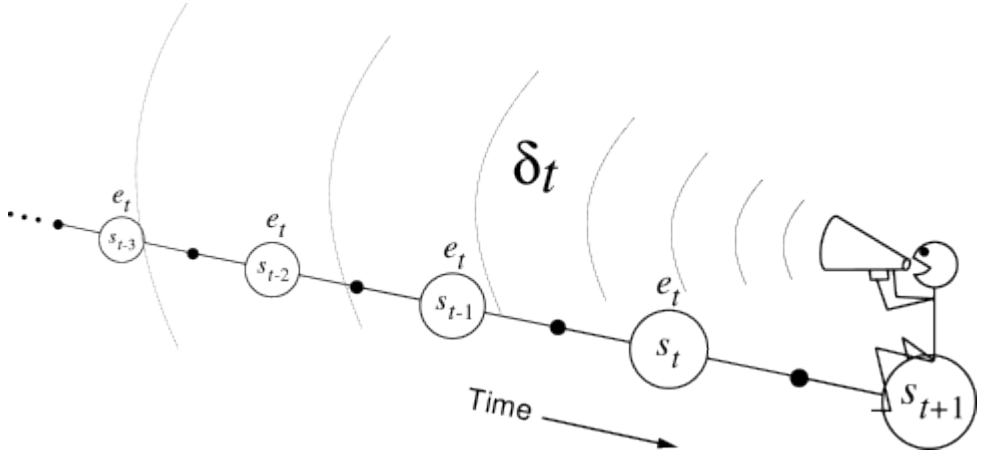


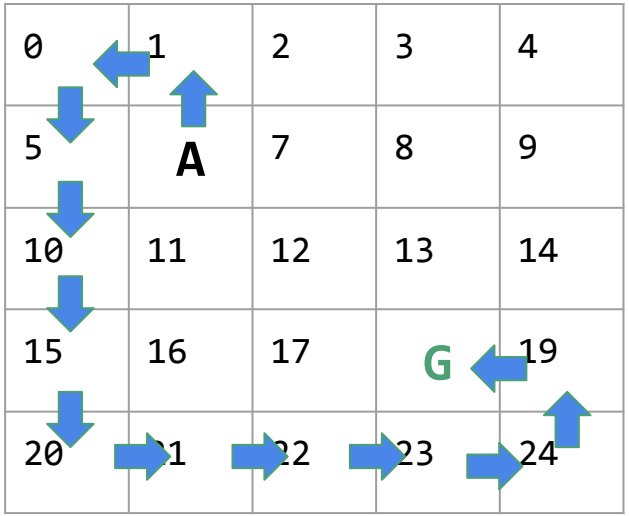
RL with Eligibility Traces

- Keep a FIFO queue of your *SA*s

  [6⬆️,1⬅️,0⬇️,5⬇️,...,23➡️,24⬆️,19⬅️]

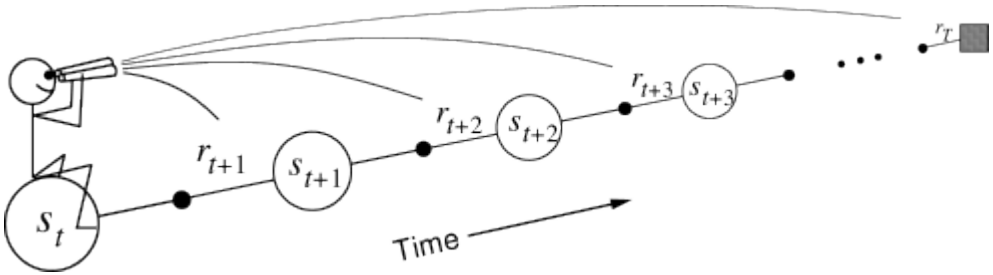
- When you get a reward, propagate it back to all *SA*s in the queue, giving lower rewards to earlier *SA*s





Q-Learning

- When you get a reward, assign it only to (S_x,A) that took you there
  - When, later, another (S_y,A) leads to S_x, its Q-value should be increased by a proportion of maximum Q-value from S_x



0	1	2	3	4
5	**A**	7	8	9
10	11	12	13	14
15	16	17	G	19
20	21	22	23	24

Basic error-driven vs Q -learning

- Error-driven learning:
  - $Q(s_t,a_t)$ += α[r_{t+1} - $Q(s_t,a_t)$]

- Q-learning:
  - $Q(s_t,a_t)$ += α[r_{t+1} + $\gamma \cdot \max Q(s_{t+1},a)$ - $Q(s_t,a_t)$]

γ is the future-value discount factor •

α is the learning rate •

Lab 4

- add standard, q-learning, and eligibility traces RL agents to the supplied jupyter notebook and run the notebook
- details on blackboard

Assignment 4 (due before next lecture)

- Each team
  - is responsible for adding the following to the exam
    - 2 questions based on Lecture 1
    - 3 questions based on Lecture 2
    - 5 questions based on Lectures 3 and 4
  - should designate an ambassador, who will
    - work with other teams to make sure your team's questions are unique
    - present 1 veto per section during next lecture
- Questions must be multiple choice, with answer key, and "Shuffle option order"
- Add your questions directly to shared form, AND paste your team's questions on blackboard

Supervised Learning

CS420 - Lecture 4
Instructor: Dr. V

Reinforcement Learning

- Relevant for:
  - **Trial-and-error problems**
- How it works:
  - try something
  - observe feedback
  - if the reward is high, do more of it
  - if reward the is low, do less of it

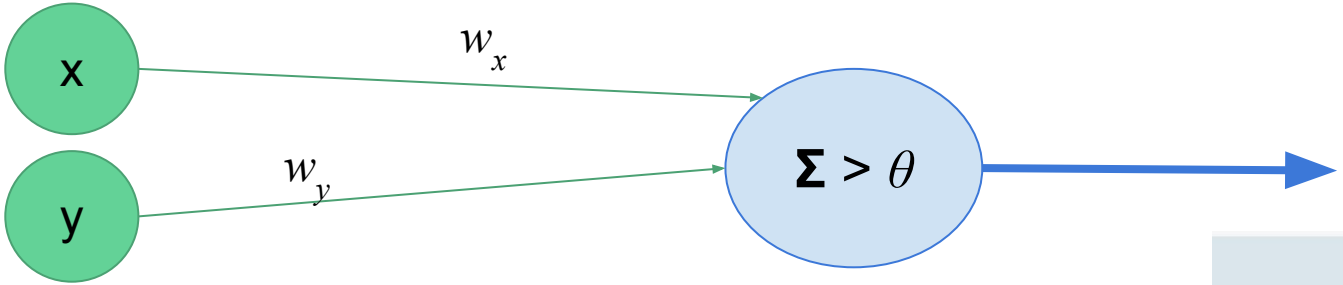
Supervised Learning

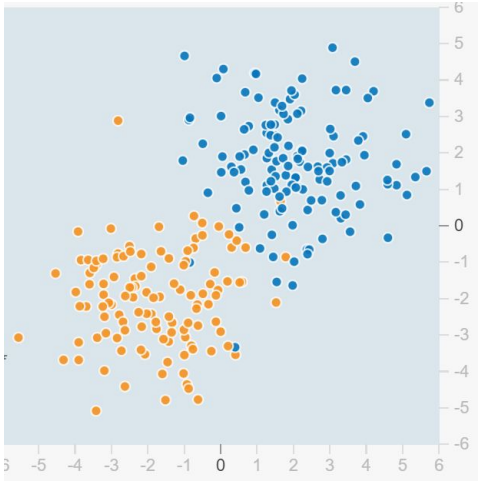
- Relevant for:
  - **Labeling data**
- How it works:
  - observe input features
  - guess label
  - observe correct label
  - next time there are similar input features, guess the correct label


02
SUPERVISED
LEARNING

Perceptron

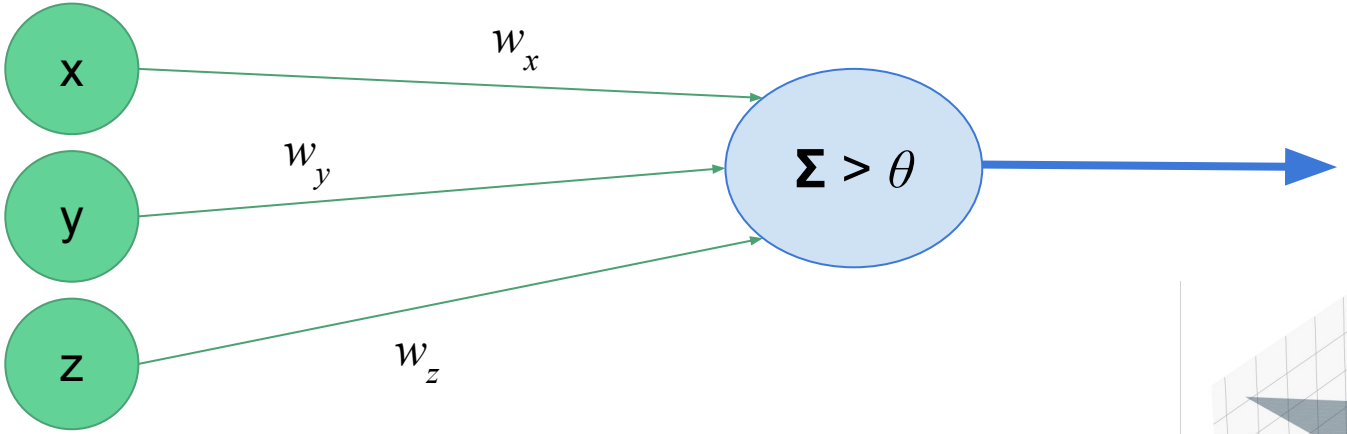
Inputs Perceptron Output

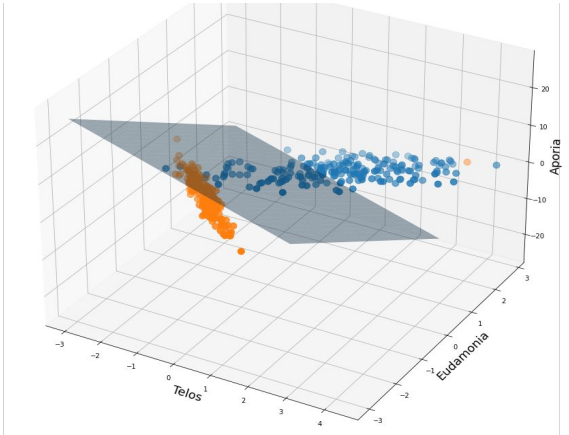




Perceptron

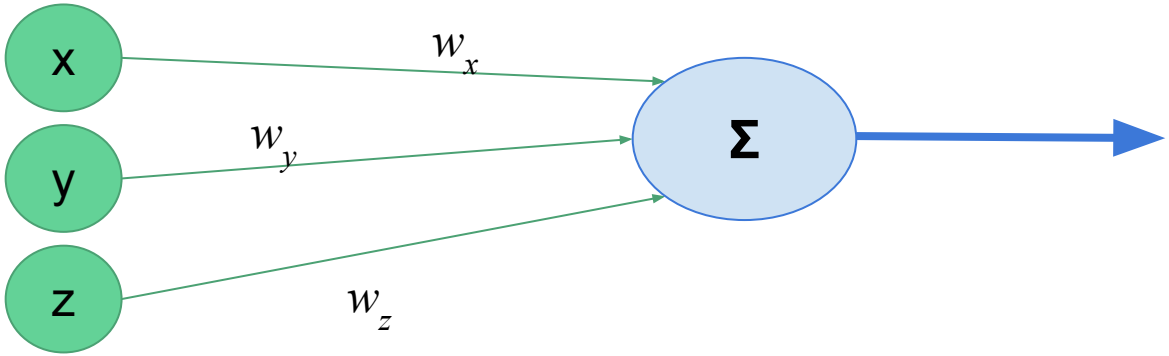
Inputs Perceptron Output





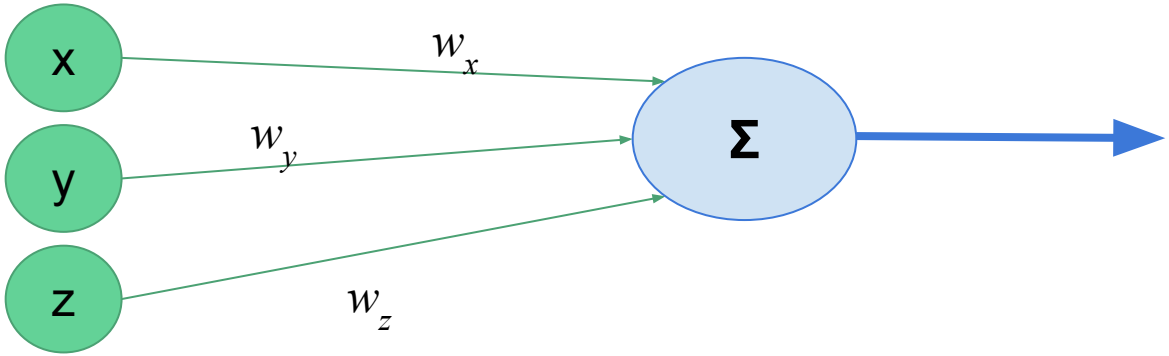
How does a perceptron learn?

- if the **Σ** output is, let's say 0.3, but supervisor says it should have been 1.0, the error is +0.7
  - so all the weights need to get bumped up by some proportion of 0.7
  - the weights that brought the most signal get bumped up the most



How does a perceptron learn?

- if the **Σ** output is, let's say 0.3, but supervisor says it should have been 0.0, the error is -0.3
  - so all the weights need to get pushed down by some proportion of 0.3
  - the weights that brought the most signal get pushed down the most



More formally: Delta learning rule

- After you make a prediction
- Update the weight, w, from each input node, i
  - increase w_i by some proportion, $\eta \cdot x_i$, of the difference between the target output, t, and the predicted output, o
    - $w_i = w_i + \eta[\ t - o\ ]x_i$

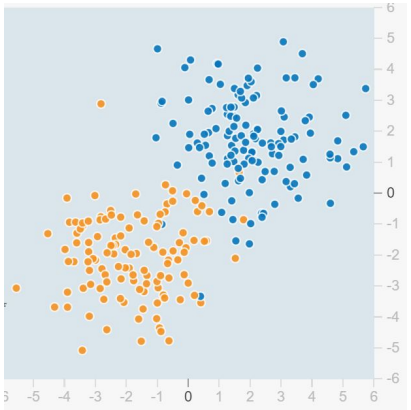
- η is the learning rate parameter
- x_i is the input value for node i

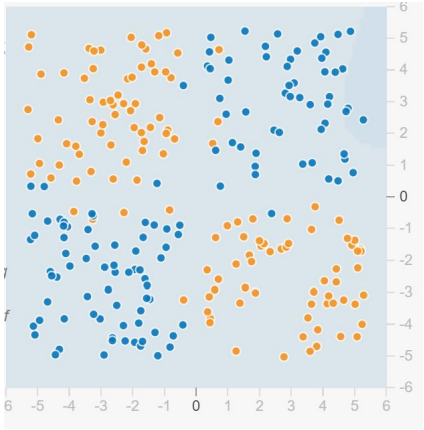
How does a perceptron learn?

- https://www.youtube.com/watch?v=khUVIZ3MON8
- https://www.youtube.com/watch?v=Ilg3gGewQ5U

Limitations of a perceptron

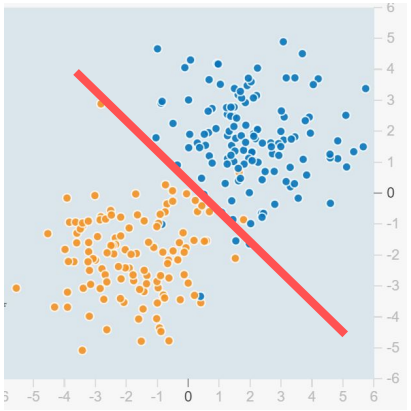
- Data must be linearly separable
  - A single perceptron can draw a single boundary in your feature-space
  - For lots of lines (for non-linearly separable data) you need more neurons connected in layers

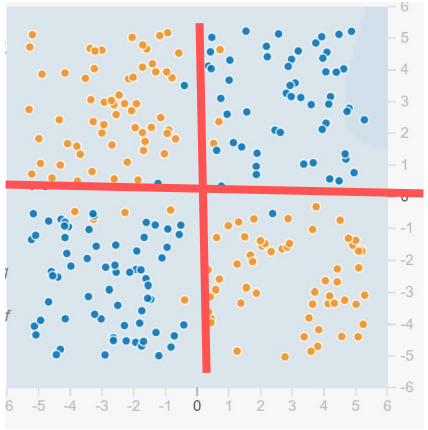




Limitations of a perceptron

- Data must be linearly separable
  - A single perceptron can draw a single boundary in your feature-space
  - For lots of lines (for non-linearly separable data) you need more neurons connected in layers





Lab 5

- use a perceptron to
  a. classify which flower instances are Setosa Irises (as opposed to Versicolor or Virginica)
  b. classify handwritten digits
- follow in-class instructions
- submit assignment notebooks on blackboard

Assignment 5 (due before next lecture)

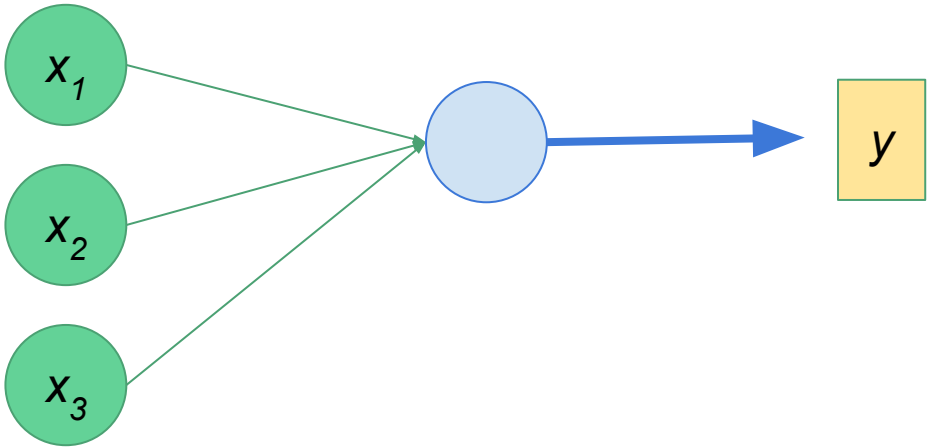
- Each team
  - is responsible for adding 4 questions based on Lectures 5 to the exam
- Team ambassador will
  - work with other teams to make sure your team's questions are unique
- Questions must be
  - multiple choice
  - with answer key
  - "Shuffle option order" (if it makes sense)
- Add your questions directly to shared form, AND paste your team's questions on blackboard

Neural Networks

CS420 - Lecture 4
Instructor: Dr. V

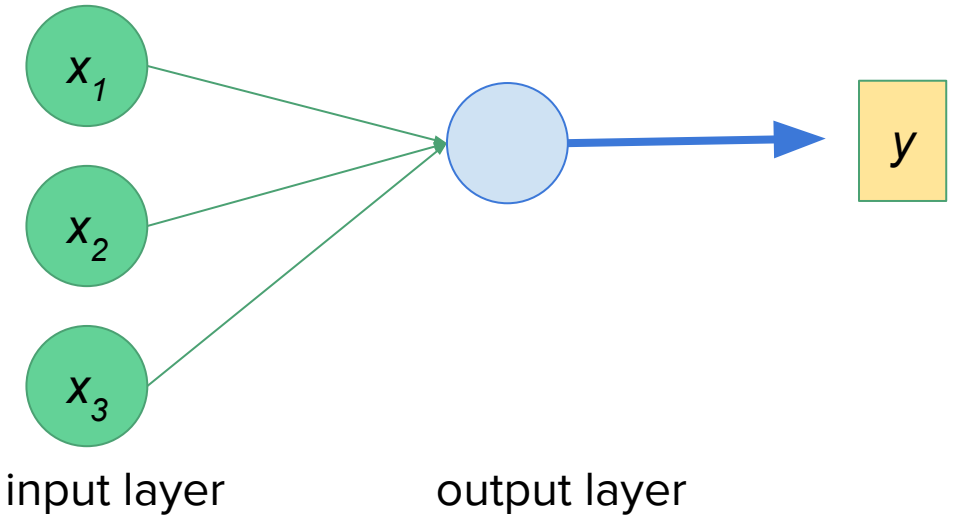
Perceptron

- Each perceptron is a single neuron
  - it has some numeric inputs, x, and it produces a numeric output, y



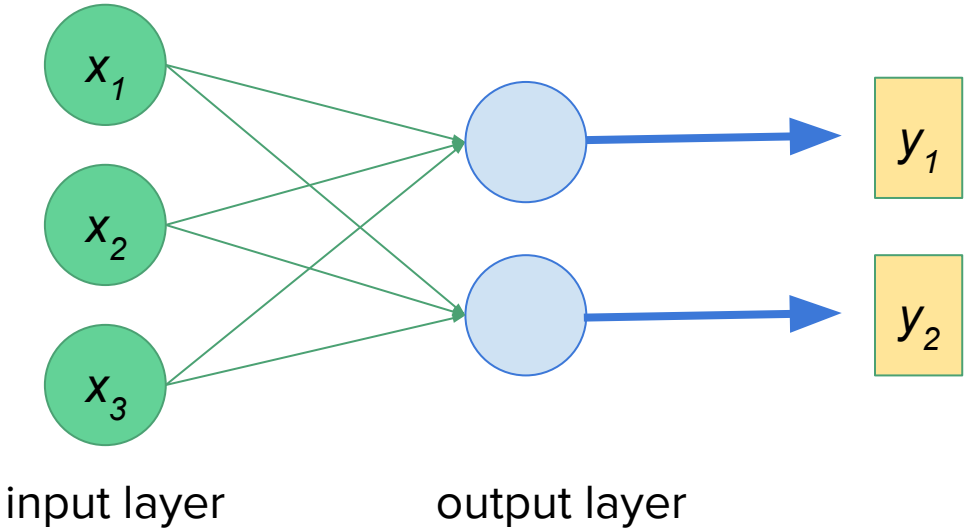
Neural Networks

- We can technically call a perceptron a neural network, rather than a neuron, since the input nodes can be thought of as neurons, as well



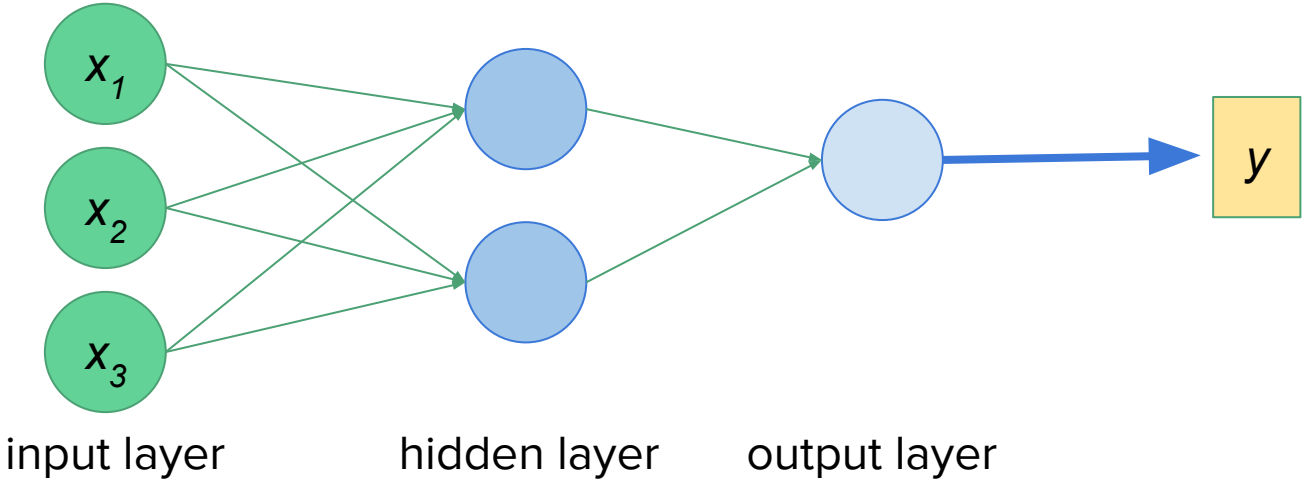
Neural Networks

- If you needed more than a single output, you could add more perceptrons, all connected to and activated by the same input nodes



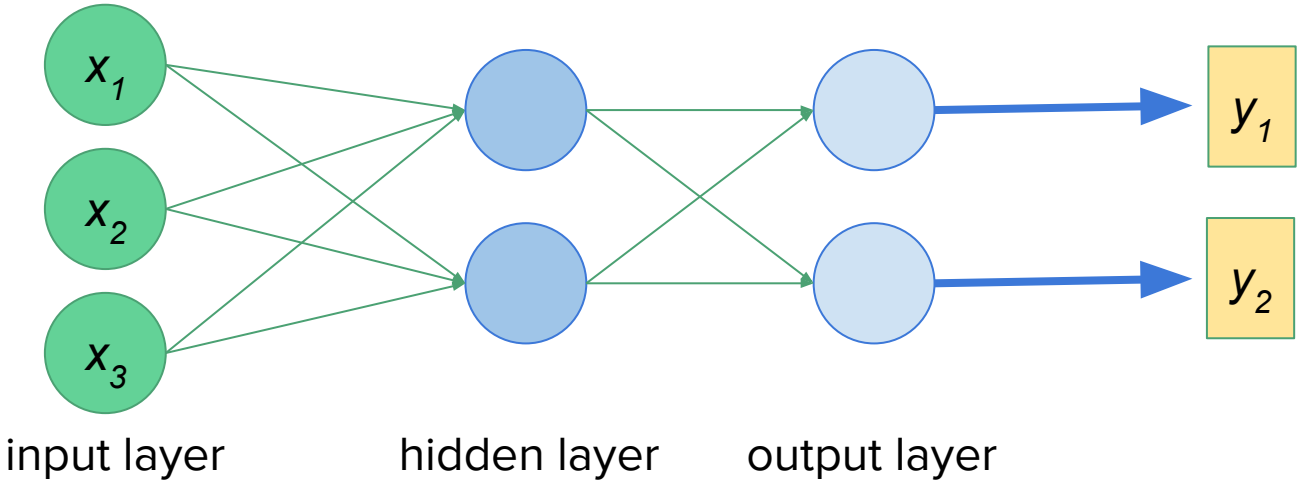
Feed-forward Neural Networks

- A **feed-forward** neural network is made up of multiple perceptrons in successive "**layers**", where outputs from prior layer neurons is used as inputs for neurons in next layer



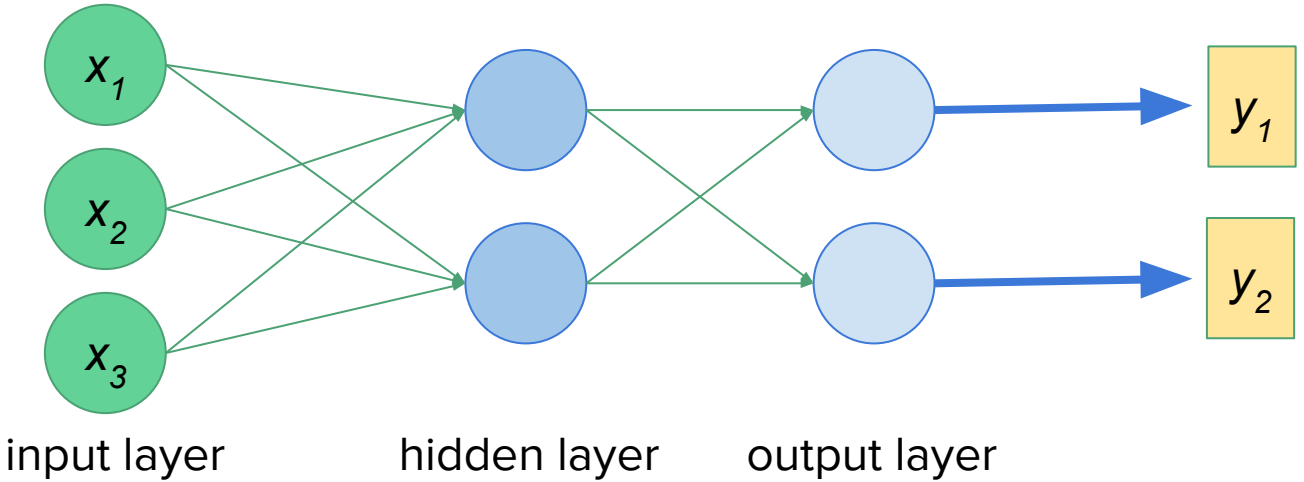
Feed-forward Neural Networks

- In a feed-forward neural net, activation is spread forward from input nodes to each successive layer, until output is produced



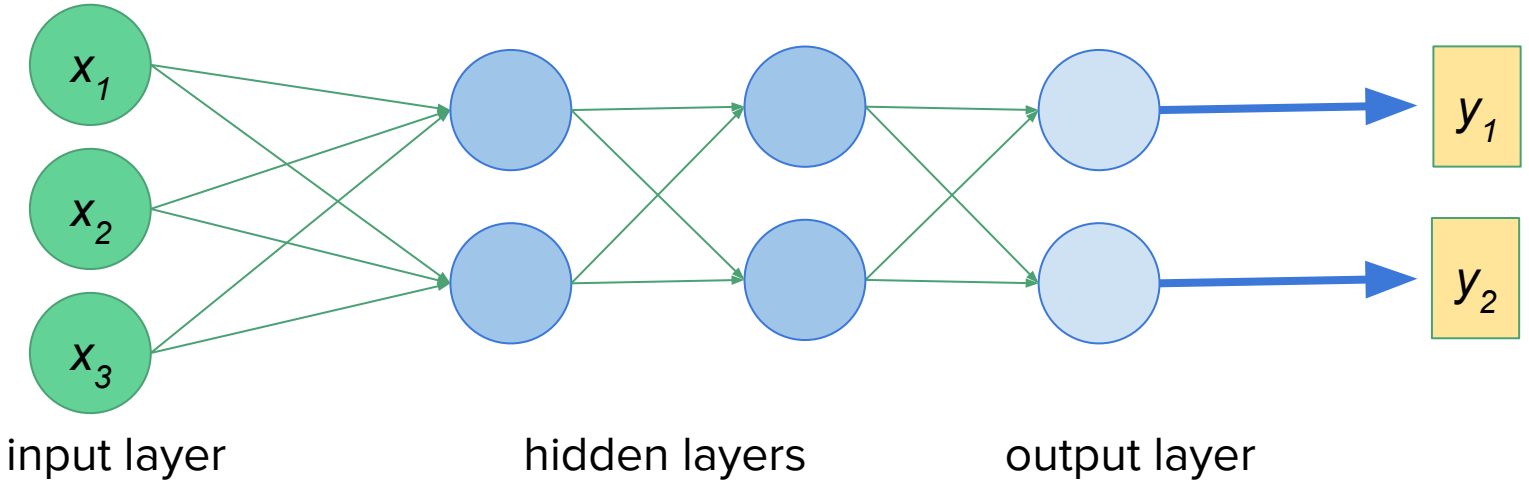
Feed-forward Neural Networks

- Most common types of Artificial Neural Nets (ANNs) employed for supervised learning are **feed-forward** nets with **fully interconnected** successive layers



Deep Neural Networks

- Deep nets are feed-forward networks with more than one hidden layer



Terminology

- Machine Learning (ML)
  - a subfield of AI concerned with trainable models/agents
- Deep Learning (DL)
  - a subfield of ML concerned with using deep neural networks

Shape of the network

- Why would you want more layers?
- Why would you want more neurons in a layer?

- Why would you want less layers?
- Why would you want less neurons in a layer?

https://playground.tensorflow.org/

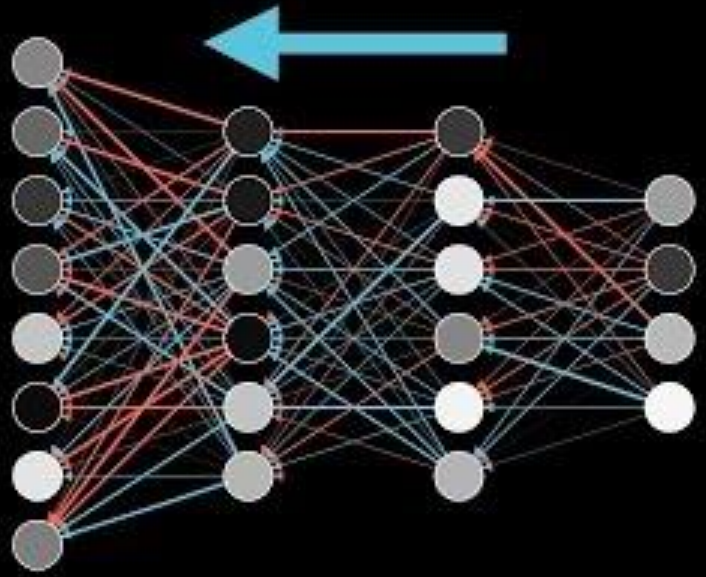

03
NEURAL
NETWORKS

How does a neural network learn?

- We know how every neuron in output layer learns:
  - Delta rule
    - the error between perceptron output and target output is used to adjust input weights
- How do the neurons in earlier layers learn?
  - Backpropagation
    - the error each successive layer is passed back to the prior layer

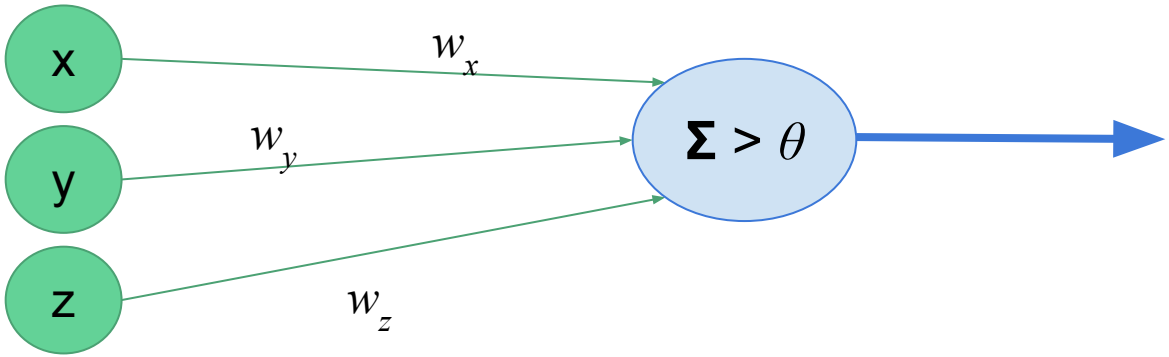

04
BACKPROPAGATION
AND OPTIMIZATION

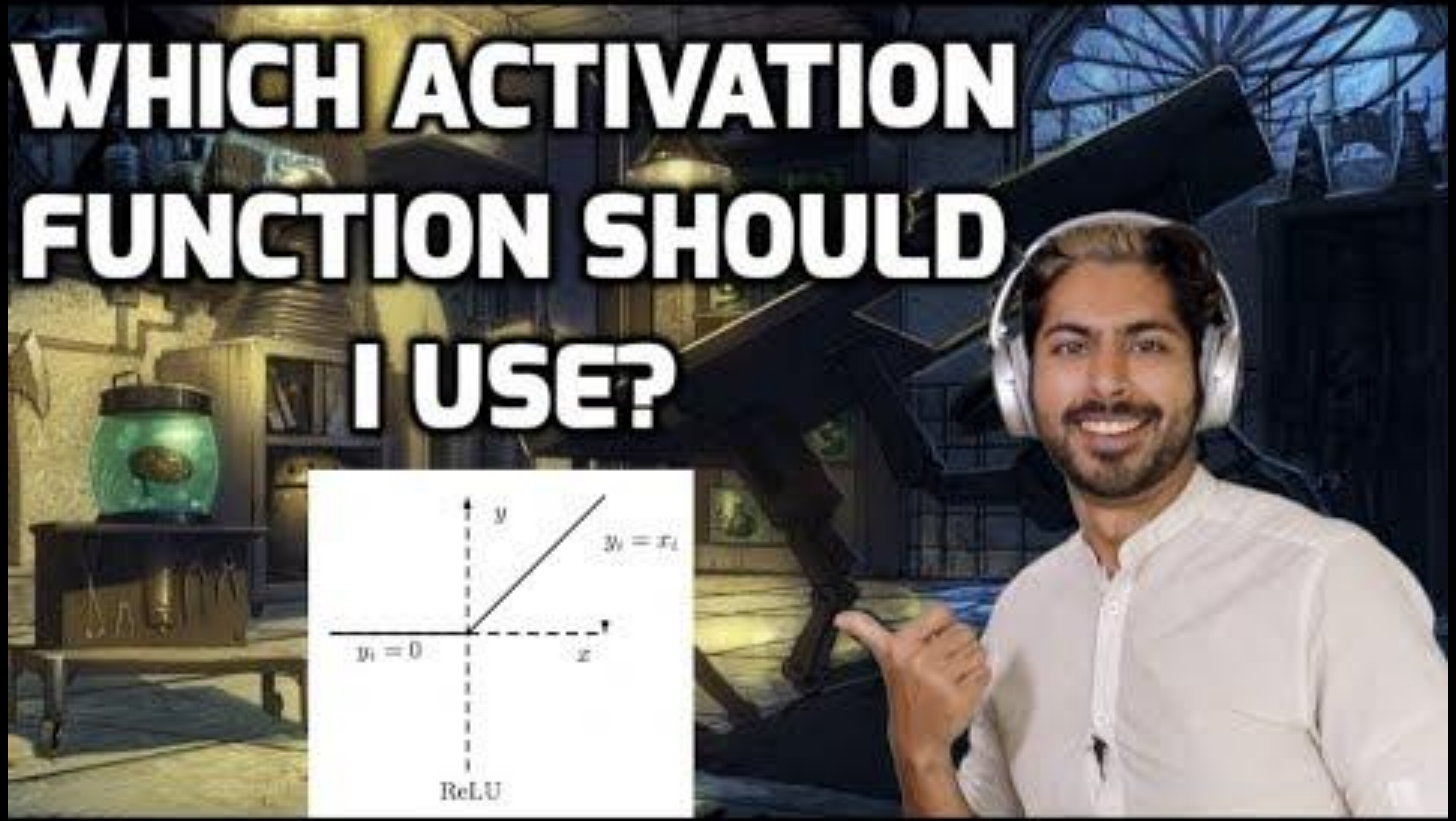
Backpropagation



Activation functions

- After we gather the sum of input activations, we output y based on some activation function; e.g.,
  - step function
  - linear
  - **ReLU**
  - leaky ReLU
  - sigmoid
  - tanh




WHICH ACTIVATION FUNCTION SHOULD I USE?
y
$y_i = x_i$
$y_i = 0$
x
ReLU

Recommendations

- for input where neighboring features are related
  - use convolutional layers
- for activation of hidden layers
  - use relu
- for activation of output layer
  - for classification, use softmax
  - for regression, use a linear function
- to calculate error
  - for classification, use categorical cross-entropy
  - for regression, use mean-abs (or mean-sqr) error

Popular python neural network libraries

- `Tensorflow / keras`
- `PyTorch`
- `NeuroLab`
- `ffnet`
- `Scikit`
- `Lasagne`
- `pyrenn`
- `MXNet`

Assignment 6 (due before next lecture)

- Each team
  - is responsible for adding 4 questions based on Lectures 6 to the exam
- Team ambassador will
  - work with other teams to make sure your team's questions are unique
- Questions must be
  - multiple choice
  - with answer key
  - "Shuffle option order" (if it makes sense)
- Add your questions directly to shared form, AND paste your team's questions on blackboard

Lab 6

- use tensorflow to classify
  - MNIST handwritten digits
  - compare against perceptron accuracy
- details on blackboard

tensorflow.keras

```
from tensorflow import keras
model = keras.models.Sequential([
    keras.layers.Dense(128, activation='relu'),  # hidden layer 1
    # add more hidden layers here, if you like
    keras.layers.Dense(10, activation='softmax') # output layer
])

model.compile(
    # learn via backpropagation (adam is a great algorithm for bp)
    optimizer='adam',
    # calculate error based on categorical cross-entropy
    loss=keras.losses.sparse_categorical_crossentropy,
    # eval model based on accuracy
    metrics=['accuracy']
)
```

Unsupervised Learning

CS420 - Lecture 7
Instructor: Dr. V

Reinforcement Learning

- Relevant for:
  - **Trial-and-error problems**
- How it works:
  - try something
  - observe feedback
  - if the reward is high, do more of it
  - if reward the is low, do less of it

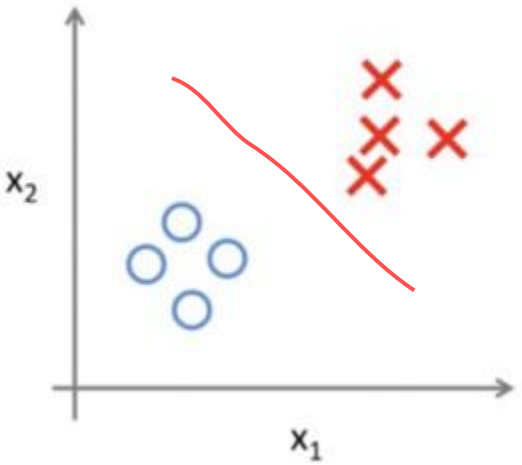
Supervised Learning

- Relevant for:
  - **Labeling data**
- How it works:
  - observe input features
  - guess label
  - observe correct label
  - next time there are similar input features, guess the correct label

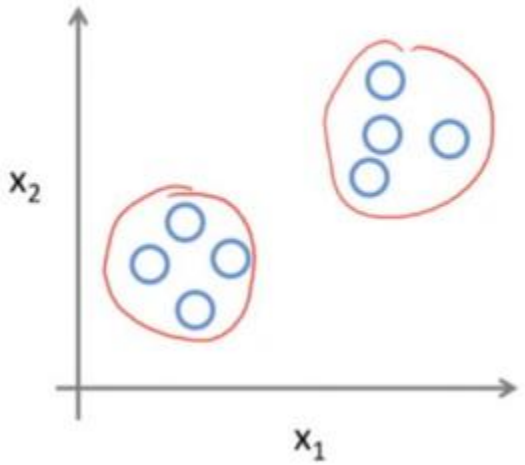
Unsupervised Learning

- Relevant for:
  - **Everything else**
    - No corrections
    - No reward feedback
- How it works:
  - observe inputs
  - learn something about them; e.g.,
    - similarity to other inputs, prototype

Supervised Learning



Unsupervised Learning

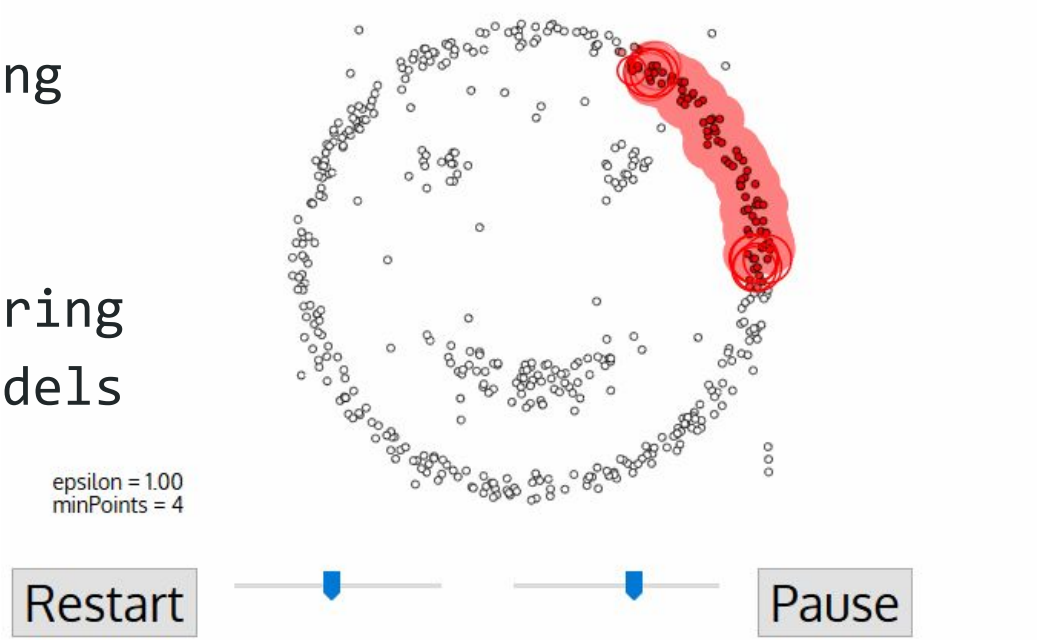


Examples of unsupervised learning

- Dimensionality reduction
  - reducing the number of input features in a dataset (e.g., PCA, SVD, Autoencoders)
- Anomaly detection
  - detecting instances that are very different from the norm (e.g., Isolation Forest, Minimum Covariance Determinant)
- Clustering
  - grouping similar instances into clusters (e.g., k-means, hierarchical clustering)

Clustering

- There are many clustering algorithms
  - k-means
  - mean-shift clustering
  - DBSCAN
  - BIRCH
  - hierarchical clustering
  - gaussian mixture models



K-means Clustering

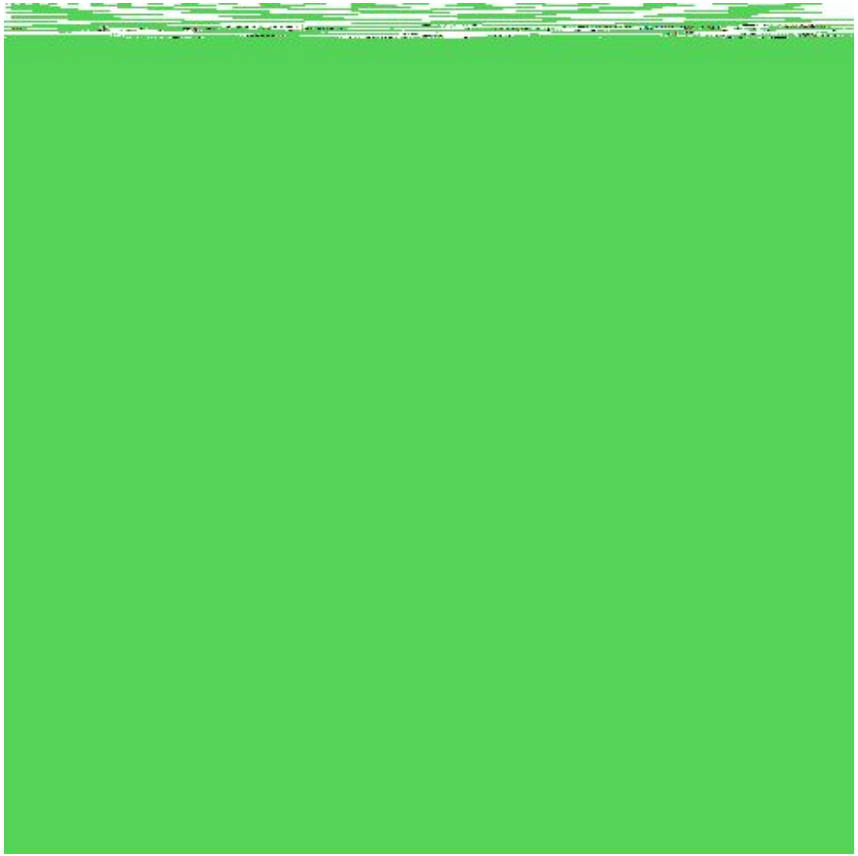
K-means Clustering Pseudocode

1. Choose the number of clusters, K
2. Place the centroids (c_0 … c_{k-1}) randomly
3. Repeat until centroids stop changing:

```
    # assign data points to clusters
    for each data point:
        - find the nearest centroid (c_0 … c_k-1)
        - assign the point to that cluster (0 … k-1)
    # change centroids to reflect new clusters
    for each cluster j in (0 … k-1):
        - c_j = mean of all points assigned to that cluster
```

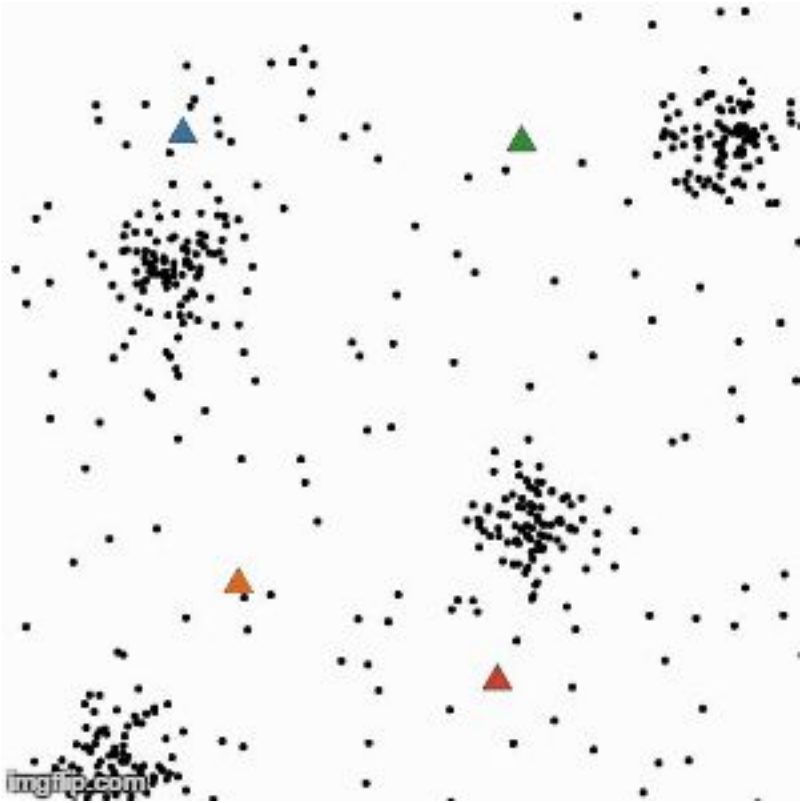
K-means Clustering

k = 3



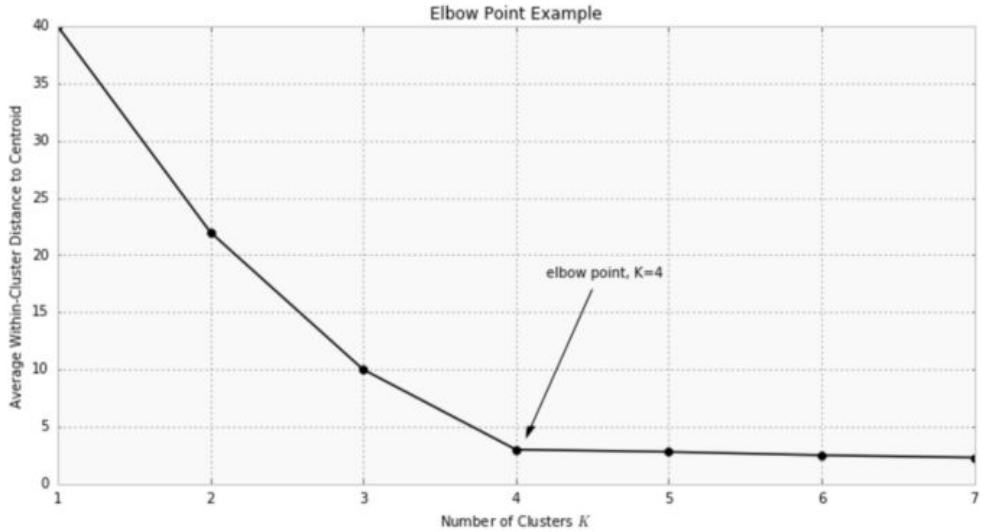
K-means Clustering

k = 4



How to choose *K*?

- most popular: elbow method
  - once average within-cluster distance stops significantly changing, stop increasing *K*




06
UNSUPERVISED
LEARNING

Assignment 7 (due before next lecture)

- Each team
  - is responsible for adding 2 questions based on Lectures 7 to the exam
- Team ambassador will
  - work with other teams to make sure your team's questions are unique
- Questions must be
  - multiple choice
  - with answer key
  - "Shuffle option order" (if it makes sense)
- Add your questions directly to shared form, AND paste your team's questions on blackboard

Lab 7

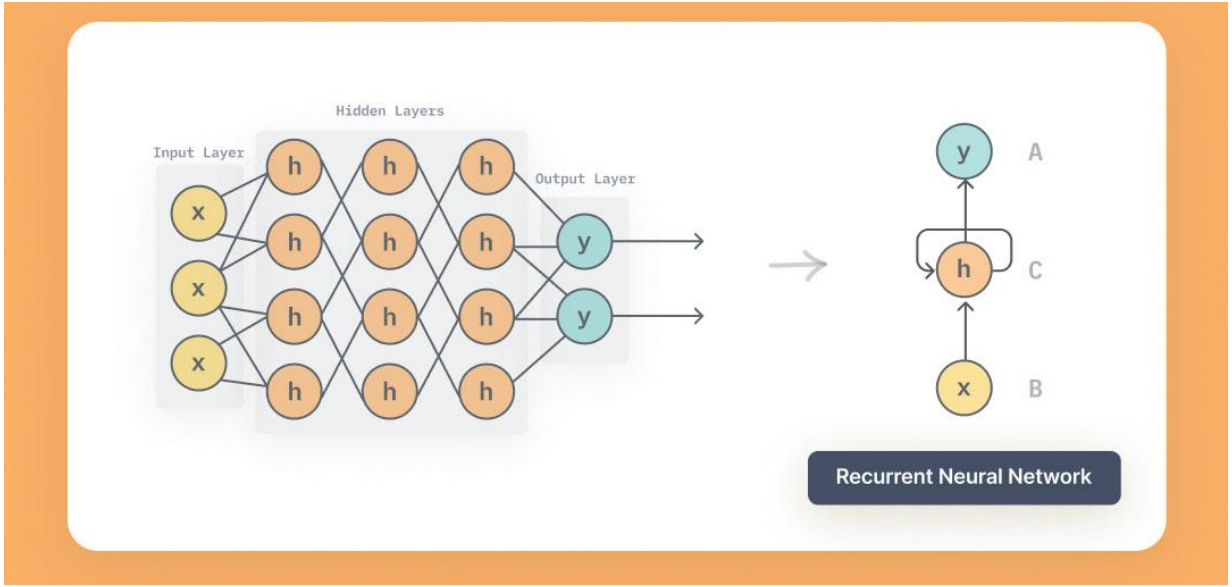
- create k-means learning algorithm
  - cluster irises using k-means
- submit notebook on blackboard

NLP

CS420 - Lecture 8
Instructor: Dr. V

RNN

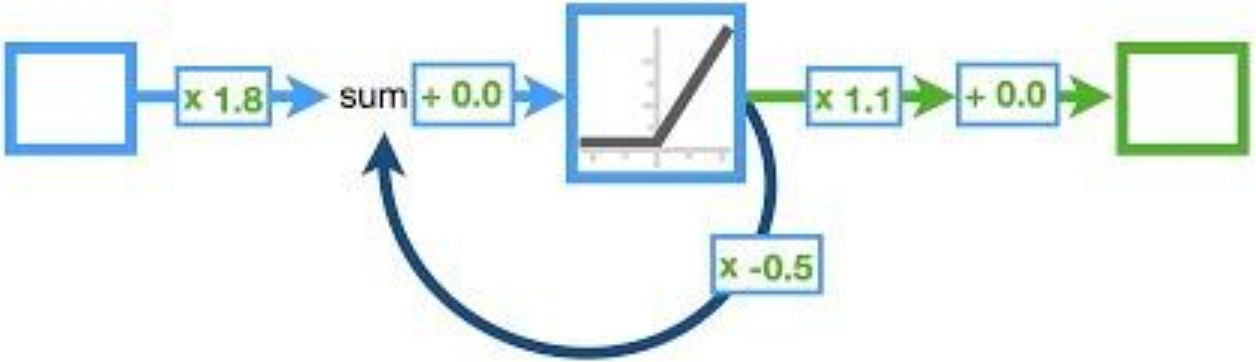
- in a recurrent neural net, the output from the hidden layer is used as the input back to the hidden layer



RNN

- RNN's can learn sequences
  - assume you want to learn which words follow which other words, so that you can predict text
    - input: "hello my name is Inigo Montoya"
  - "hello" is the first input, target output is "my"
  - RNN learns that "hello" output should be "my"
  - now "my" becomes the input, target output is "name", but the hidden layer output from "hello" is ALSO part of the input
  - etc.

Recurrent Neural Networks (RNNs)...



...Clearly Explained!!!

RNN

- is RNN supervised or unsupervised learning?

LSTM (long-term short-term memory) neural nets

- LSTM neural net is like RNN, but older memories have less influence as inputs into the hidden layers than newer memories

NLP

- there are many aspects to natural language processing in AI
  - predicting which words terms are next in a sequence
  - grammar parsing
  - finding common terms
  - figuring out how what words/terms are most important/representative of a document
  - figuring out how similar or related two words are
  - figuring out how similar or related two sentences or documents are

tf-idf

- which words/terms are the most important?
  - tf-idf (term frequency–inverse document frequency)
    - $\text{tfidf}(t,d,D) = \text{tf}(t,d) \times \text{idf}(t,D)$
  - $\text{tf}(t,d) = \text{length}(t|d) \,/\, \text{length}(d)$
    - frequency of term t in document d, divided by total number of terms in the document
  - $\text{idf}(t,D) = \log[\ \text{length}(D) \,/\, \text{length}(D|t)\ ]$
    - inverse frequency of term t in corpus of documents D

How related two pieces of text are to each other

- you shall know a word by the company it keeps
  - LSA (latent semantic analysis)
  - PMI (pointwise mutual inclusion)
  - NGD (normalized google distance)
  - many others:
    - https://en.wikipedia.org/wiki/Semantic_similarity

PMI

- $pmi(x,y) = \log_2(p(x,y) / (p(x) \times p(y)))$
  - log-scaled probability of *x* and *y* appearing together divided by probabilities of each
- if you just want to rank a set of terms $y_1..y_n$ in terms of how related they are to *x*, you don't need to know probabilities, just frequencies:
  - $rank(x,y_i) = f(x,y_i) / f(y_i)$


HELLO
BONJOUR
你好
07
NATURAL LANGUAGE
PROCESSING

Assignment 8 (due before next lecture)

- Each team
  - is responsible for adding 2 questions based on Lectures 8 to the exam
- Team ambassador will
  - work with other teams to make sure your team's questions are unique
- Questions must be
  - multiple choice
  - with answer key
  - "Shuffle option order" (if it makes sense)
- Add your questions directly to shared form, AND paste your team's questions on blackboard

Lab 8

- pick 5-7 different words
  - get PMI rank-scores for each pair of different words based on Google searches
- submit all word pairs with their PMI-rank scores, sorted by the scores on blackboard

Free Parameters & Optimization

CS420 - Lectures 9-10
Instructor: Dr. V

Free Parameters

- Our RL agent had 3 parameters
  - learning rate
  - exploration parameter
  - discount factor for past or future states
- Our Neural Nets had a bunch of parameters too
  - learning rate
  - how many hidden layers
  - how many nodes in each hidden layer
  - which activation function to use (e.g., relu, step)

Free model parameters

- Choosing correct values for free parameters is a common problem in AI
- How would you choose the values for each parameter, for example, in the Trace RL agent you created?
  - learning rate: .1? .01? .25?
  - exploration: .2? .05? .001?
  - decay: .9? .5? .1?

Hill climbing

- make a slight change to a parameter value
  - if it results in better performance, keep going in the same direction, making slight changes
  - if it results in worse performance, go in a different direction
  - if value performance no longer improves, you found the best value for the parameter!
- repeat for every parameter

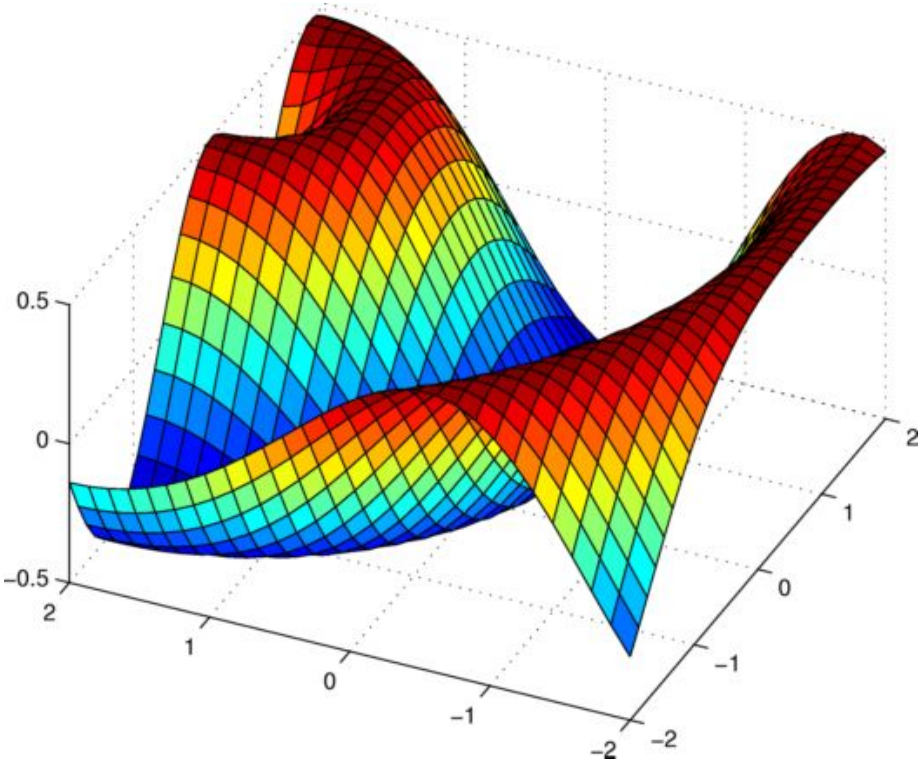
Hillclimbing seems good...

- Can you foresee a problem with hill-climbing?

- What's another way to find [near-]optimal parameter values?

Exhaustive search

- rather than doing anything smart, we can just look up ***all*** parameter value combinations
  - create a mesh grid
  - look up agent/model score for every point in grid



grid search

```
import itertools, numpy as np

# 3 parameters, check all values from 0.05 to .95, by step of .05
vals = np.arange(.05,.95,.05)
for paramValues in itertools.product( vals,vals,vals ):
    score = getScore( *paramValues )
    …

# if higher scores are better, report argmax, otherwise argmin
#   i.e., parameter values that produce best score
```

grid search

```
import itertools, numpy as np

# 3 parameters, check all values from 0.05 to .95, by step of .05
vals = np.arange(.05,.95,.05)
for paramValues in itertools.product( vals,vals,vals ):
    agent = TraceRL(ACTIONS, *paramValues)
    score = getScore( agent )
    …

# if higher scores are better, report argmax, otherwise argmin;
#   i.e., parameter values that produce best score
```

Full grid search seems good, but...

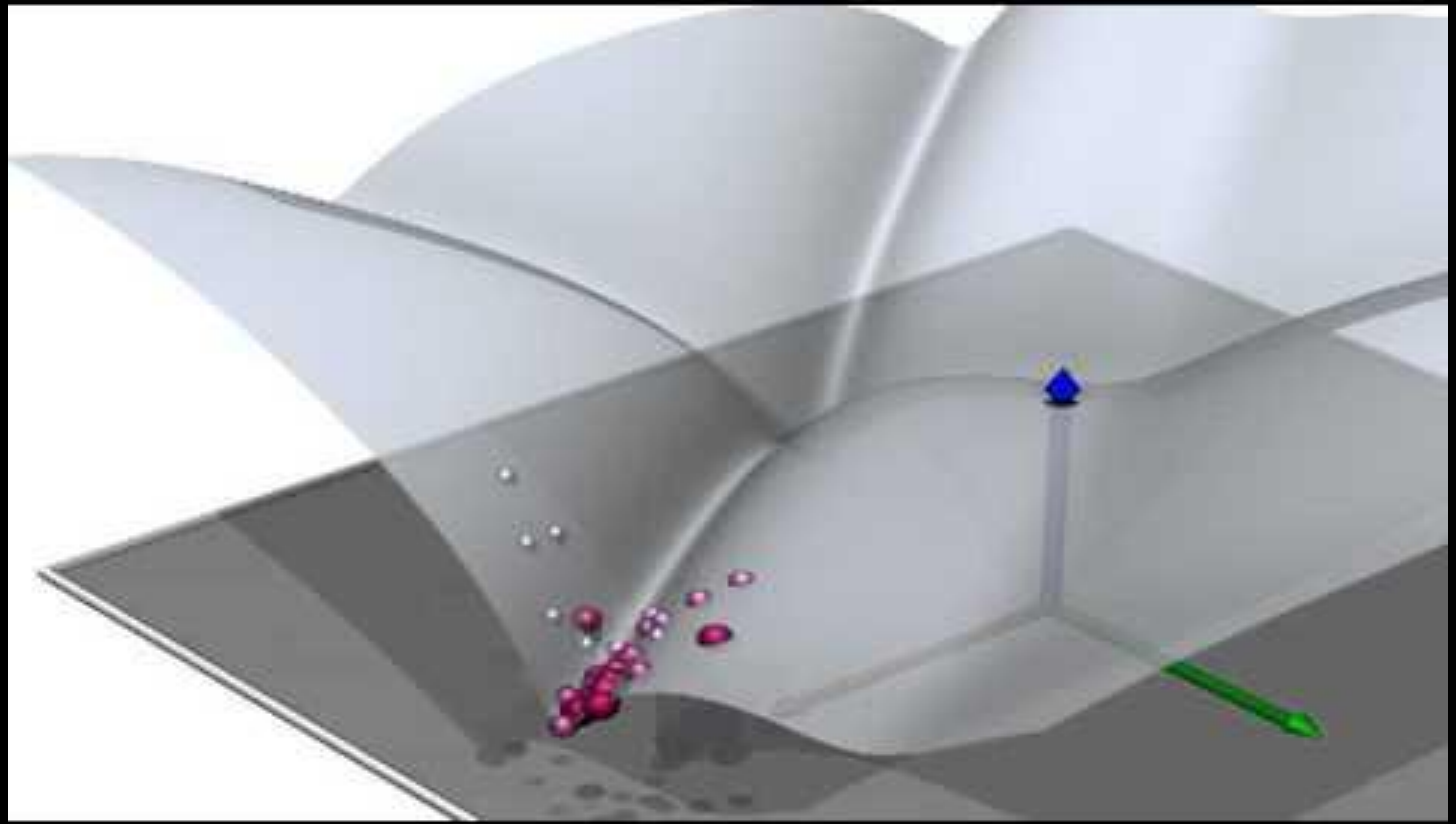
- Can you foresee a problem with an exhaustive grid-search?

Full grid search seems good, but...

- Can you foresee a problem with an exhaustive grid-search?
  - if the step size is small, you'll find your near-optimal solution, but
    - time to solution increases exponentially with every additional free parameter
  - and if the step size is large, you'll find a solution much faster, but there will be gaps large enough to have missed better parameter values in your parameter space

Simulated Annealing

- Simulated Annealing - a type of stochastic hill-climbing
- Simulated Annealing is like hill-climbing, but
  - there is a balance between exploration and exploitation
  - if you're still in exploratory mood (high temperature), you can move onto worse parameter values
  - as you get more greedy (lower temperatures), your tolerance for worse parameter values decreases



Dual Annealing

- Dual Annealing is a version of Simulated Annealing that combines the global stochastic search with a second phase that includes a local search

scipy.optimize.dual_annealing

```
from scipy.optimize import dual_annealing
bounds = [(0.01,.5), (0.01,.5), (0.2,.99)]
result = dual_annealing(                                \
             getScore,                                  \
             bounds)
# takes 80m on a high-end laptop; getScore runs in ~0.4s

print( result )
```

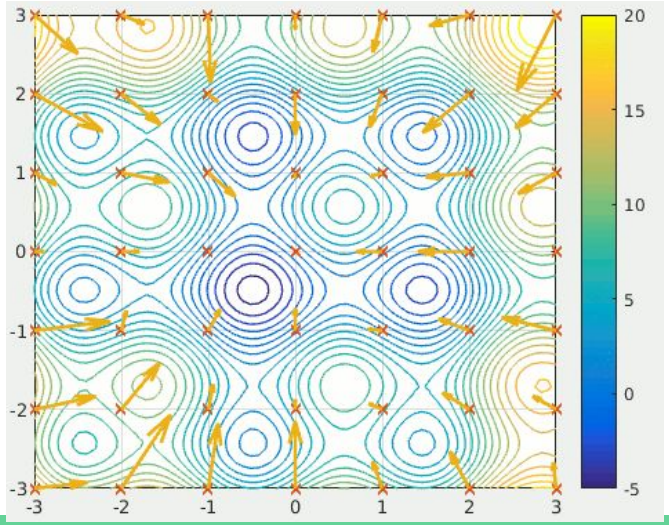
scipy.optimize.dual_annealing

```
from scipy.optimize import dual_annealing
bounds = [(0.01,.5), (0.01,.5), (0.2,.99)]
result = dual_annealing(                                    \
             lambda p:getScore(TraceRL(ACTIONS,*p)),        \
             bounds)
# takes 80m on a high-end laptop; getScore runs in ~0.4s

result
   score: 526.72
 message: ['Maximum number of iteration reached']
    nfev: 7269
       x: array([0.06, 0.09, 0.82])
```

Optimization Algorithms

- There are many optimization algorithms
  - if a function is differentiable (you can find its derivative)
    - use stochastic gradient descent
  - if not
    - grid search
    - simulated annealing
    - evolutionary algorithms
    - particle swarm optimization
    - cross-entropy method
    - etc



GENETIC ALGORITHMS

EXPLAINED BY EXAMPLE

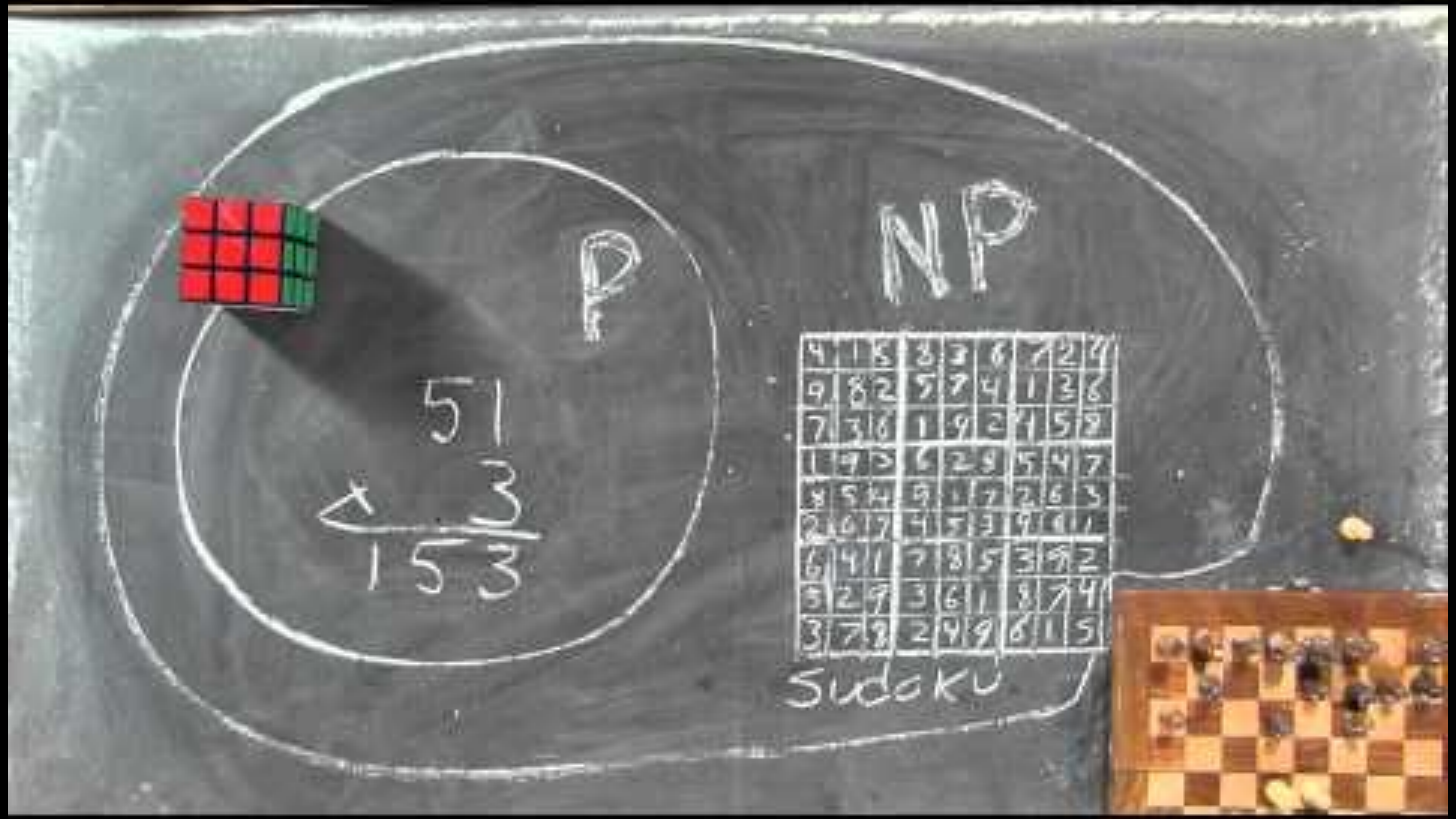
Genetic Algorithms

- Simulates genetic evolution (natural selection)
  - first generation:
    - create random individual parameter sets (genes)
  - for every following generation:
    - **select fittest** individuals to reproduce
      - for every pair of parents, take some parameter values from each parent to produce a new set of **crossover** parameter values – a new genome
      - slightly **mutate** the new genome
    - keep the fittest individual(s) from current generation, delete the rest

scipy.optimize.differential_evolution

```
from scipy.optimize import differential_evolution
bounds = [(0.01,.5), (0.01,.5), (0.2,.99)]
result = differential_evolution(                              \
               lambda p:getScore(TraceRL(ACTIONS,*p)),        \
               bounds, maxiter=50)
# takes 19.5m on a high-end laptop; getScore runs in ~0.4s

result
      score: 526.72
    message: ['Maximum number of iterations exceeded']
       nfev: 2371
convergence: 0.93987
          x: array([0.05, 0.02, 0.81])
```


P
NP
51
3
153
Sudoku

Assignment 6 (due before next lecture)

- Each team
  - is responsible for adding 4 questions based on Lectures 9-10 to the exam
- Team ambassador will
  - work with other teams to make sure your team's questions are unique
- Questions must be
  - multiple choice
  - with answer key
  - "Shuffle option order" (if it makes sense)
- Add your questions directly to shared form, AND paste your team's questions on blackboard

Lab 9

- use a grid search to find best values for
  - reinforcement learning agent parameters

- submit notebook on blackboard