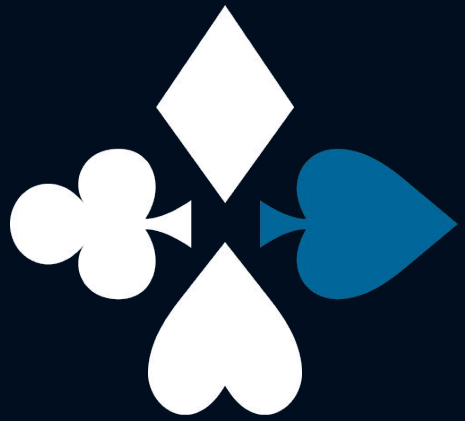


Pokerbots 2024

Lecture 5: Advanced Topics

Sponsors


HRT


CITADEL
CITADEL | Securities


DRW


Jane Street


jump trading


TREX
QUANT


SEVEN
EIGHT
CAPITAL, LP


SIG
SUSQUEHANNA


HAPCAPITAL


CTC
CHICAGO TRADING COMPANY


2σ
TWO SIGMA


akuna capital

Announcements

Tuesday's Lightning Tournament Results

Strategy Proof Turngate	$400
Pineapple	$300
shep-a-doodle	$225
Curtis James Jackson III	$175
DKE sophomores	$150

Weekly Tournament 2: Today @ 11:59pm EST

Poker Afternoon Study Break Next Tuesday!

Guest Speaker:
Noam Brown
Monday 1/29

Guest Speaker: GTO Wizard Wednesday 1/31

Final Tournament & Sponsor Networking: Friday, February 2nd

Today's Giveaway

24th submission wins!
pkr.bot/twentyfour

Hint: Last lecture giveaway entries were submitted roughly once every 2 seconds

Agenda

- Machine Learning
- Reinforcement learning
- CFR
- Neural networks

Machine Learning

The next big thing

a few morning headlines…

The Verge

Meta's new goal is to build artificial general intelligence

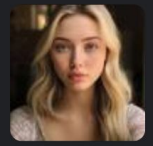
21 hours ago

Coveteur

I Got AI Headshots So You Don't Have To

I decided to get AI headshots on a whim. I ended up sending my BDD on a rollercoaster ride to hell–and back.

36 mins ago



Forbes

FORBES > INNOVATION > AI

Bluffing Beyond Human Capability: AI's Role In Revolutionizing Poker

Neil Sahota Contributor

Neil Sahota is a globally sought after speaker and business advisor.

Follow

0

Jan 19, 2024, 10:00am EST

Buzzwords

Deep learning neural networks for intelligent big data analytics with business-to business

automated artificial intelligent blockchains in the cloud!

...it’s really much more simple

Learning

Machine learning (ML) algorithms complete a process or task, but they get better at completing that task with experience. At a certain point, they can become good enough to handle the task with near perfect accuracy!

- Prediction
- Decision Making
- Automation

General workflow

ML techniques need experience to improve. That experience mainly comes from either looking at data or trying a new task over and over again. This is called *training*.

Typical training plan:

- Start with a simple strategy
- Take in experience
  - Look at a data point, try something new, etc.
- Learn something from that experience
- Update our strategy and repeat...

Areas of ML

Typical algorithms used in ML fall generally into three categories:

- Supervised Learning
  - We show our algorithm many input and output examples
- Unsupervised Learning
  - We ask our algorithm to recognize patterns without telling it the right answer
- Reinforcement Learning (RL)
  - Our algorithm learns from an environment and tries to get some “reward”

Supervised learning example

- We love cats and birds!
- We want our algorithm to take a photo and tell us if it’s a cat or a bird
- We have example photos and labels of both
- Let’s train a model on our example photos!

Supervised Learning: Training

Data

“Cat”

“Cat”

“Cat”

Labels

Supervised Learning: Training

“Bird”

“Bird”

“Bird”

https://mit.zoom.us/j/99610361734

Supervised Learning: Prediction

Cat or Bird?

Supervised learning algorithms

There are many algorithms that can do this kind of thing!

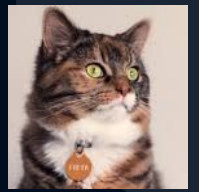
- Neural Networks
- Decision Trees
- Regression
- Support Vector Machine (SVM)

Unsupervised Learning

Group these by similarity...





Unsupervised Learning

Group 1

Group 2

Unsupervised learning algorithms

A lot of clustering and general data analysis

- *k*-means clustering
- *k*-nearest neighbors
- Expectation - Maximization Algorithm (EM)

What about a Pokerbot?

We can certainly use techniques from supervised and unsupervised learning for River of Blood Hold'em:

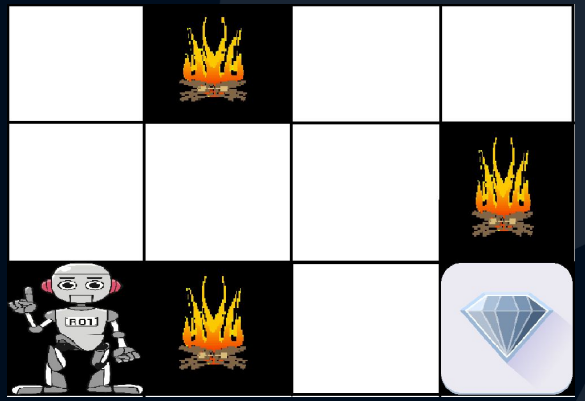
- Predict the strength of our cards
- How we should play on the Run

But we may benefit more from another machine learning area...

Reinforcement Learning (RL)

Reinforcement Learning: an overview

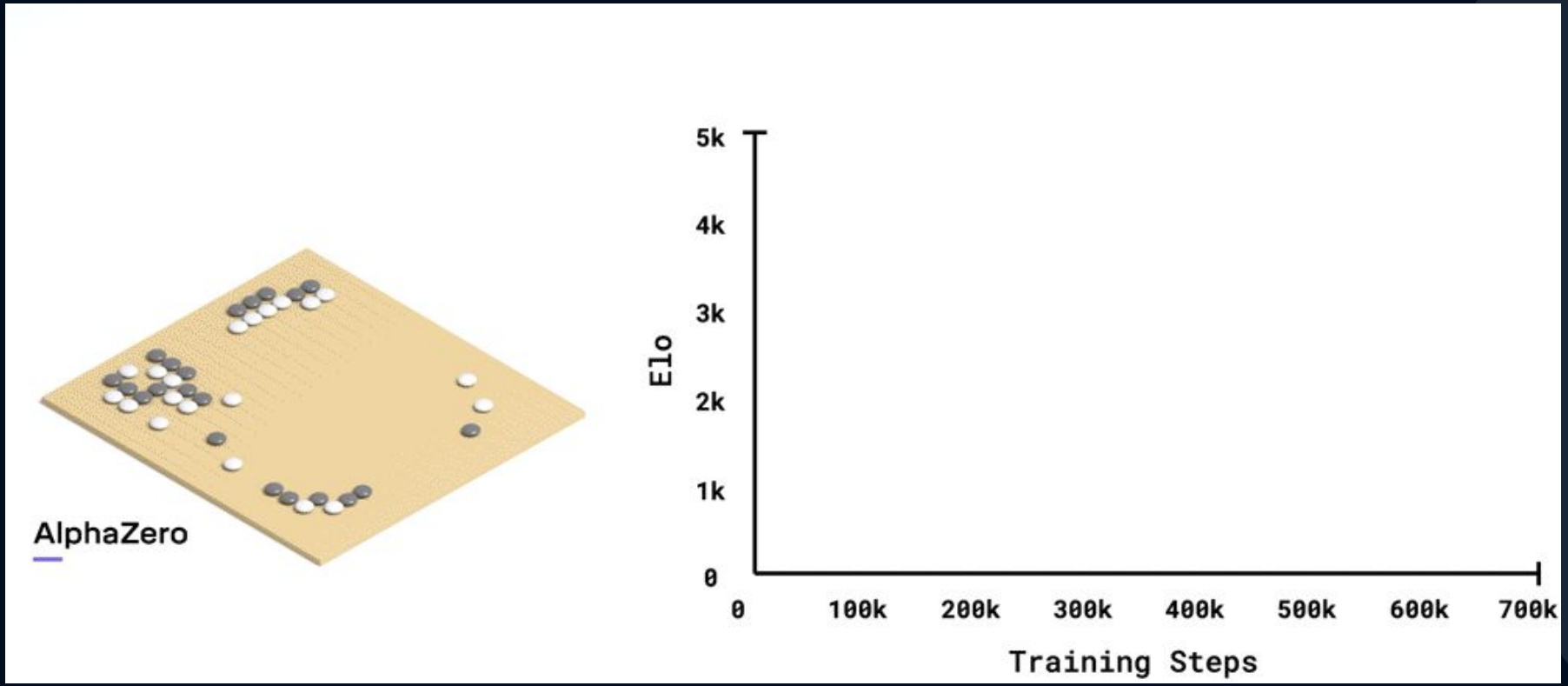
- An *agent* takes *actions* to move between *states*, with the goal of maximizing *reward*
- The agent has multiple attempts to learn an effective *policy* (strategy)
- Examples: self-driving cars, robotics, poker
- Poker framework
  - Reward: chips
  - States: the different betting rounds
  - Decisions: Check, Call, Raise, Fold, etc.



Successes

- AlphaZero: trained entirely from self-play
- Beat best-in-world chess engine starting from only the rules of the game
- DeepMind “parkour” paper:
  - Inputs: terrain map, joint angles, angular velocities
  - Reward: forward progress

AlphaZero: from zero to mastery in four hours



Parkour

Why don't we all use reinforcement learning?

- Hard to train
  - Sensitive to parameters
  - Escaping local optima
- Sample inefficient

5000

Number of processors needed to generate self-play games for AlphaZero’s training

6400

Number of CPU-hours needed to train DeepMind Parkour

Struggles with multi-agent scenarios

- In chess, if bot1 loses to bot2 and bot2 loses to bot3, then there is a good chance that bot3 is the strongest chess player
- Gives a clear route to *iterated improvement*
- This is far from guaranteed in poker

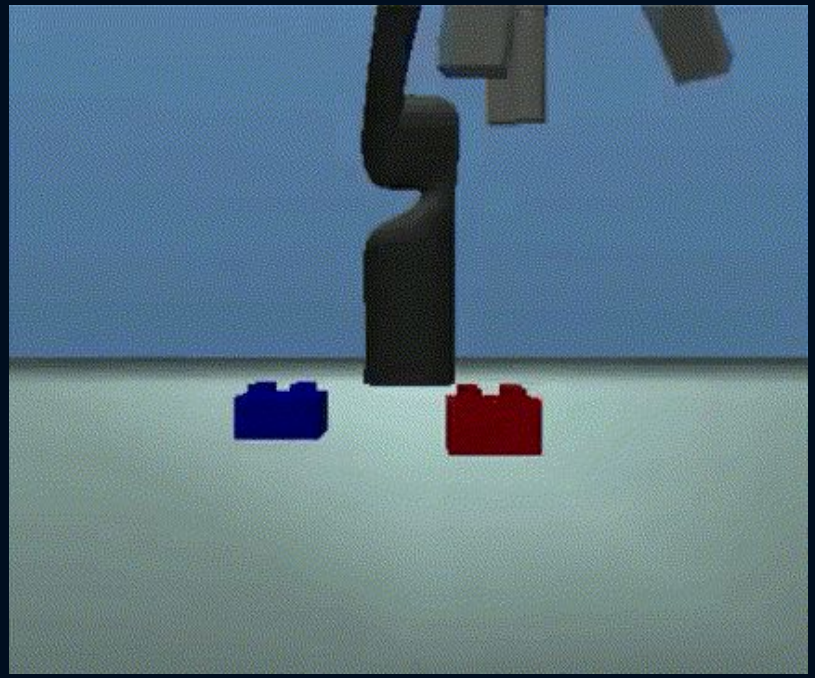
Return to rock-paper-scissors

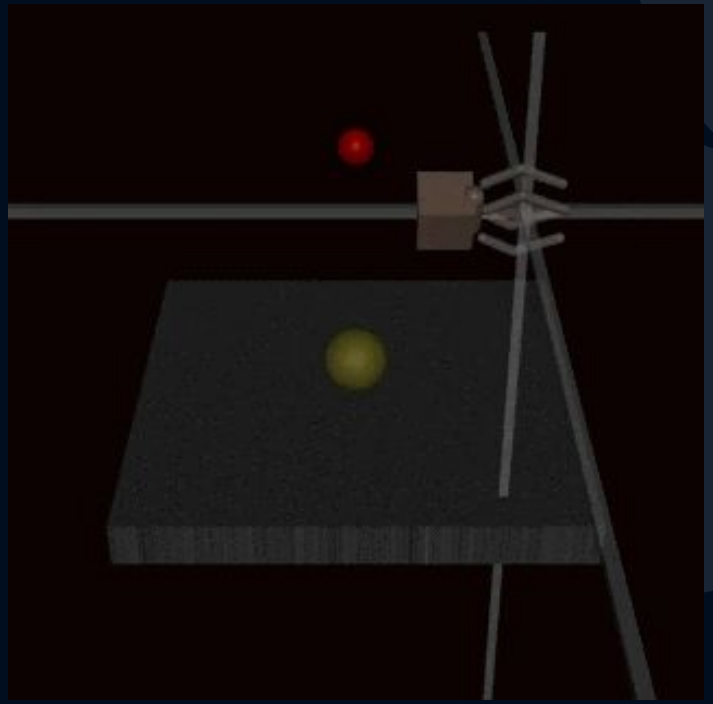
- When two reinforcement learning agents are trained against each other, they get very good at beating *each other*
- Risk of getting caught in a *policy cycle* without making meaningful improvements to performance
- Hero: rock ⟶ villain: paper ⟶ hero: scissors ⟶ villain: rock ⟶ hero: paper ⟶ villain: scissors ⟶ hero: rock…
- Cycles are predictable, which is undesirable

What to reward?

- Designing a good reward function is hard
- Luckily in poker, the job is done for us
- Example: reinforcement-learning Tetris
    - Misbuilt reward function led to the agent to pause the game when it was about to lose
- Reinforcement Learning from Human Feedback (RLHF)

Specification Gaming





Suppose we've considered all the warnings...

- Well-crafted reward function
    - Clear goal in mind for our policy
- Sufficient compute resources
- Handling multi-agent scenarios
- How do we train our policy?

Simple Example: Multi-armed Bandit

- The ‘game’ only has one state, with multiple fixed actions
- Each action has its own reward, potentially drawn from some distribution
- We have to choose one action to pick that gives us the most (expected) reward, but we can play many times
- Exploration vs Exploitation (general concept in RL)
    - Biphasic Approach
    - Epsilon-greedy

Multi-armed Bandit Examples

- Your Tiktok feed
- Choosing a restaurant to eat at
- Using multiple Pokerbots across rounds

Q-learning

Q-learning intuition

- Let’s return to thinking of poker as a multi-step process
    - Extensive form instead of matrix form in the game theory lecture
- Q-learning is a sample-based, one-agent way to tabulate *the game states* of this process and *the quality of each action* we could take in any given game state

Q-table

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Update rule

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

The upsides of Q-learning

- *General*: Q-learning can learn a policy to maximize final reward even if rewards happen incrementally

- *Simple and intuitive*: we repeatedly play games, and we take the deterministic action with the highest quality (do what worked well in the past)

- *Theoretically sound*: for any finite Markov decision process, Q-learning finds a policy that maximizes expected reward

Finite Markov decision process?

- Perfect information games: chess, Go
- Video games: Tetris
- Games with randomness: Monopoly

Partial observability

- Ordinary Q-learning is not guaranteed to work for imperfect information games:
  - Poker against an opponent with a fixed strategy
  - Trading
  - Liar’s dice

More downsides

- Slow to converge
- Prone to getting stuck in local optima
- Intractable if the state space is too large
- What do we do?
  - Use neural networks as an approximation
  - Use a specialized algorithm for large, imperfect information games

Counterfactual Regret Minimization (CFR)

lunch break 

Counterfactual Regret Minimization (CFR)

CFR Overview

- Have a *strategy profile*
  - set of strategies for each player which we hope to improve
- Assign each game state a *value*
  - assuming both players are using the strategy profile, what is the EV of the game's outcome from the given state?
- Calculate *(counterfactual) regrets* for each action
  - according to the above values, is the action better or worse than average, and by how much? High regret = 'good' action
- Use the regrets to inform our next strategy profile
  - We should make actions with high regret more likely, so we don't regret not doing them as much (minimizing counterfactual regret)
- Repeat

Regret-matching (Hart & Mas-Colell, 2000)

RPS: We play rock, opponent plays paper

Regrets: 0 for rock, +1 for paper, +2 for scissors ⟶ (0, 1, 2)

Mixed strategy: (r=0, p=1/3, s=2/3), opponent plays scissors

Value: -1/3 for our mixed strategy, 1 for rock, -1 for paper, 0 for scissors

New regrets: (4/3, -2/3, 1/3), new cumulative regrets: (4/3, 1/3, 7/3)

New mixed strategy: (r=1/3, p=1/12, s=7/12)

New *average* strategy: (r=1/6, p=5/24, s=5/8)

Averaging our regret-matching strategy

1. Compute a mixed strategy for each player by matching *cumulative* regrets. (If all cumulative regrets for a player are non-positive, use a random strategy.)

2. Select each player's action by sampling from their strategy.

3. Update cumulative regrets.

4. Repeat T times.

5. Return the *average* mixed strategy across the T iterations.

Counterfactual regret (Zinkevich et al., 2007)

- Instead of matching regret, we match *counterfactual regret*
- For a node n, counterfactual regret answers the question: what would n's value change to if I picked some pure action a?
- Value for a node n: values of n's children multiplied by the corresponding action probabilities (according to n's counterfactual regret-matching strategy)
- Goal: regret-matching algorithm on the game tree

Monte Carlo CFR algorithm (Lanctot et al., 2009)

1. Fix all random actions for a game.

2. Construct an entire game tree given the fixed random actions and each players' regret-matching strategies. Each node is weighted by its likelihood under the players' strategies. (Randomness is weighted implicitly by the first step.)

3. Compute counterfactual regrets at each node.

4. Run regret-matching to get new mixed strategies for each player.

5. At the end, use the *average* strategy to play poker.

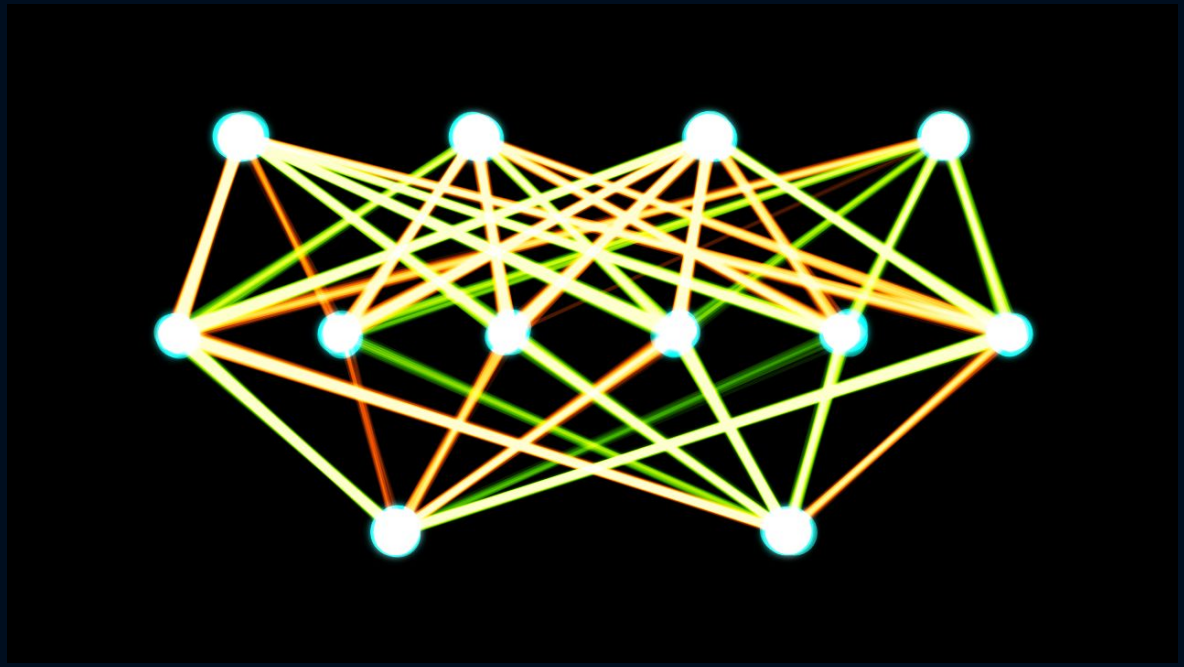
Problems and extensions

- The bucketing problem: how can we reduce the size of the game tree?
- Bot size limits
- Inference time constraints
- Extensions: CFR+, linear strategy weighting, external sampling
- http://modelai.gettysburg.edu/2013/cfr/index.html

Neural Networks

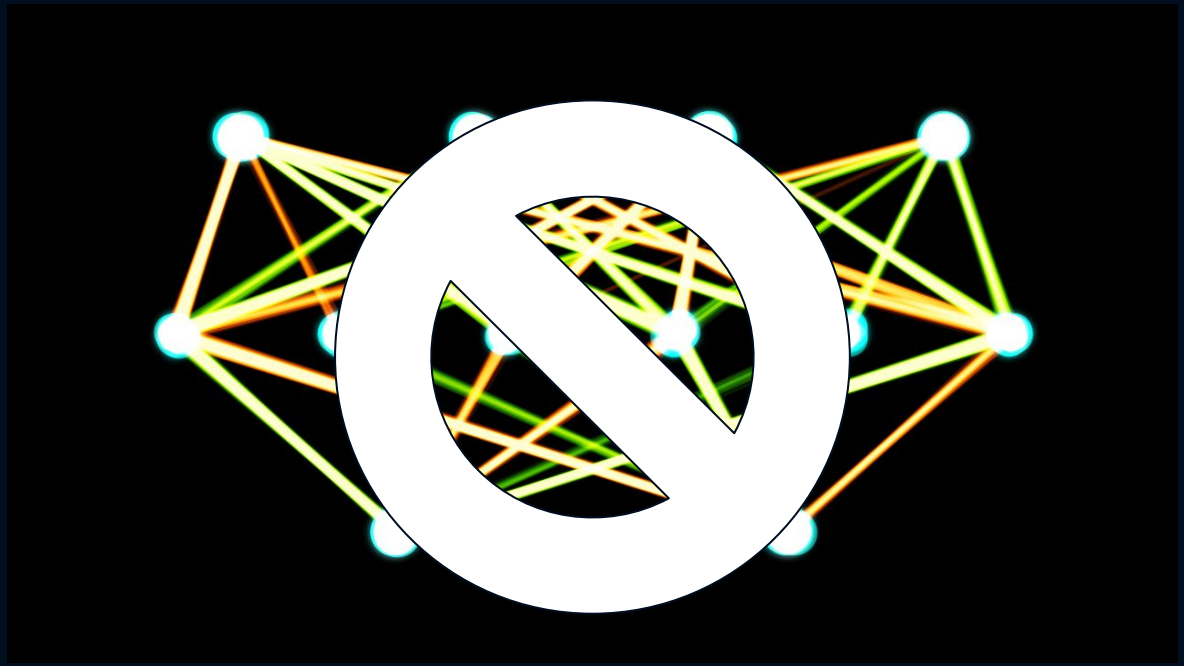
What is a neural network?

Multilayer perceptron?



What is a neural network?

Multilayer perceptron?



Yes, but...

- A more practical way to think of neural networks is as one of the most empirically successful ways to *approximate a function* based on a *limited number of input-output pairs*

- They come in many shapes and sizes, with bigger networks being more expressive, i.e. better able to approximate complicated functions

- What makes them successful?

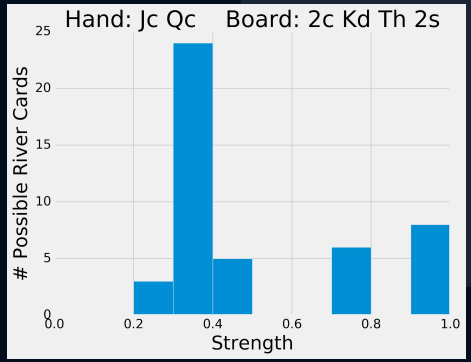
    - Ability to generalize to unseen data

What uses neural nets?

- Image and speech recognition
- Recommender systems
- AlphaZero
- DeepMind parkour
    - A Q-table is a function that we can approximate: deep Q-learning
- DeepStack

Neural networks and poker

- Lots of functions could be worth approximating:
  - Game state ⟶ strategy (distribution over actions)
  - Private cards and board cards ⟶ final hand strength curve
  - Strategy ⟶ rewards/regrets
  - Game state ⟶ CFR bucket



Proceed with caution

- Central problem: how do I train?
    - Need high-quality input-output pairs
- If I take samples from the scrimmage server, then I am approximating my opponents, which makes it less likely that I'll beat them
- If my samples are against a fixed strategy, then I will learn how to beat that one strategy but not how to play poker well
- If I play two neural nets against each other, then who knows if they will converge or get caught in cyclic behavior

Lecture Recap

- Machine Learning
  - Reinforcement Learning
    - CFR
- Neural Networks as universal approximators
- No coding session today
- Implementing these ideas can be very challenging - while we encourage exploring further and trying them yourself, it’s neither necessary nor expected

Giveaway Winners!

Twenty Four Game Winner

ricar60

1	Timestamp
2	1/19/2024 12:16:29
3	1/19/2024 12:16:29
4	1/19/2024 12:16:29
5	1/19/2024 12:16:32
6	1/19/2024 12:16:35
7	1/19/2024 12:16:35
8	1/19/2024 12:16:38
9	1/19/2024 12:16:40
10	1/19/2024 12:16:42
11	1/19/2024 12:16:45
12	1/19/2024 12:16:48
13	1/19/2024 12:16:51
14	1/19/2024 12:16:52
15	1/19/2024 12:16:53
16	1/19/2024 12:16:54
17	1/19/2024 12:16:57
18	1/19/2024 12:17:10
19	1/19/2024 12:17:12
20	1/19/2024 12:17:25
21	1/19/2024 12:17:25
22	1/19/2024 12:17:38
23	1/19/2024 12:17:40
24	1/19/2024 12:17:53
25	1/19/2024 12:18:15
26	1/19/2024 12:18:15
27	1/19/2024 12:18:15
28	1/19/2024 12:18:16
29	1/19/2024 12:18:19