Pokerbots 2021

Lecture 5: Advanced Topics

Announcements

Libratus Creator:
Noam Brown

This Friday @ 1:00pm EST!
pkr.bot/class

Poker Night with Sponsors!

(with 9 raffles + beginner/advanced tables!)

Wed 01/20

4:30 - 6pm EST

Final Tournament & Sponsor Networking: Friday, January 29th

Week 2 Tournament
Friday, 11:59pm EST!

New Prize: Strongest Alliance

Pokerbots is better with more teammates! Improve your bot and collaborate more effectively with an *Alliance*. $100 to the top three alliances in tournament 2!

Alliance: A new team formed from two or more existing teams (capped at 4 people)

Learn more and form your alliances at pkr.bot/alliance and @117 on Piazza!

Currently there is only 1 team eligible for 3 prizes! Reach out! Stick around after lecture for alliance forming!

Giveaway

50th submission wins!
pkr.bot/raffle

Agenda

- Machine Learning
- Reinforcement learning
  - Goals and fundamental challenges
  - Q-learning
  - CFR
- Neural networks

Machine Learning

The next big thing

NEWS · 30 NOVEMBER 2020

'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures

Google's deep-learning program for determining the 3D shapes of proteins stands to transform biology, say scientists.

Artificial intelligence / Machine learning

OpenAI's new language generator GPT-3 is shockingly good—and completely mindless

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 Blotto Hold'em:

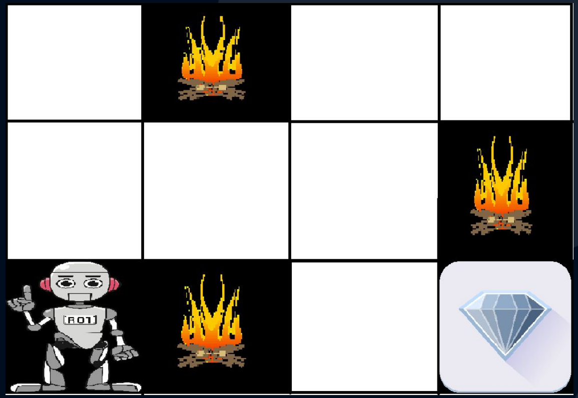
- Predict the strength of our cards
- Categorize the types of hands we could have
- Group our hole cards effectively

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

Reinforcement Learning

Reinforcement Learning (RL): an overview

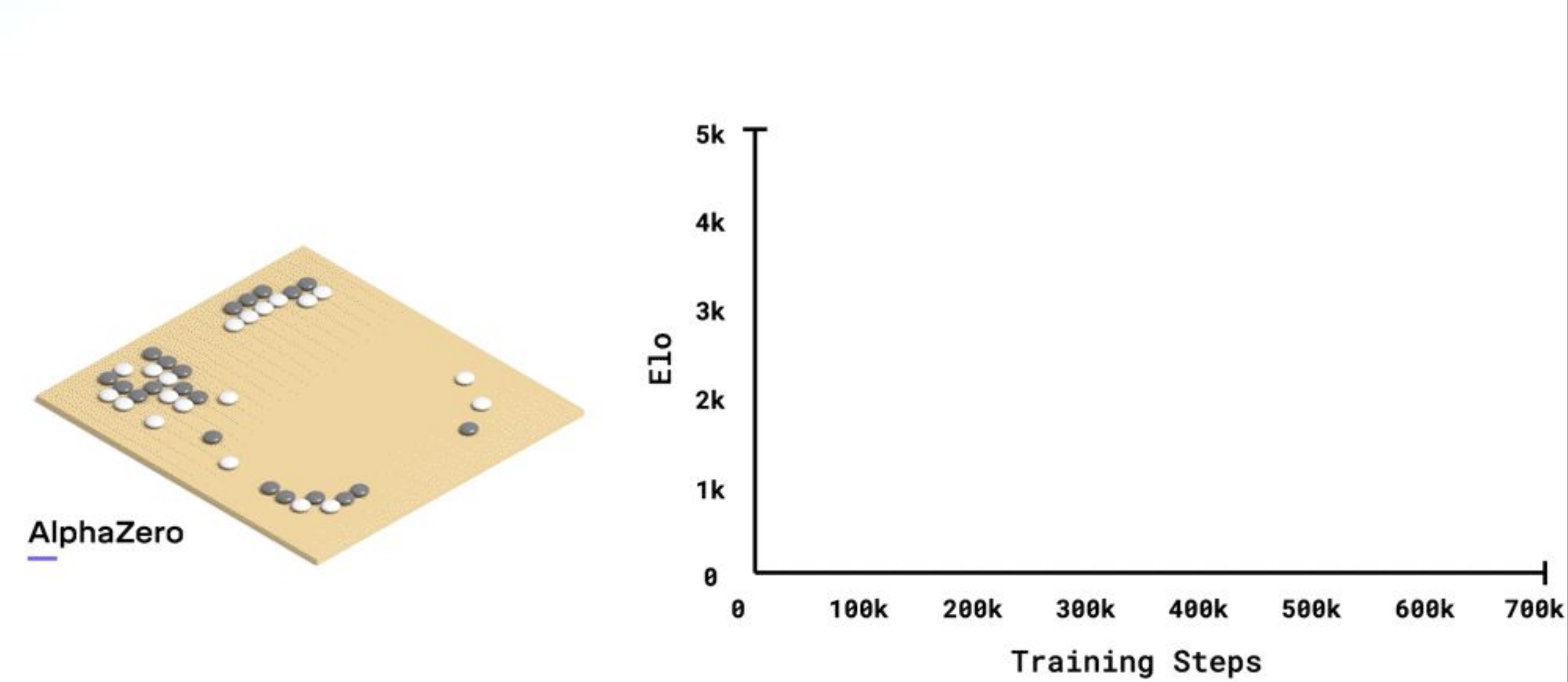
- An *agent* takes *actions* to move between *states*, with the goal of maximizing *reward*
- The agent has multiple attempts to learn an effective *policy* (strategy)
- Examples: self-driving cars, robotics, poker
- Poker framework
  - Reward: chips
  - States: the different betting rounds
  - Decisions: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



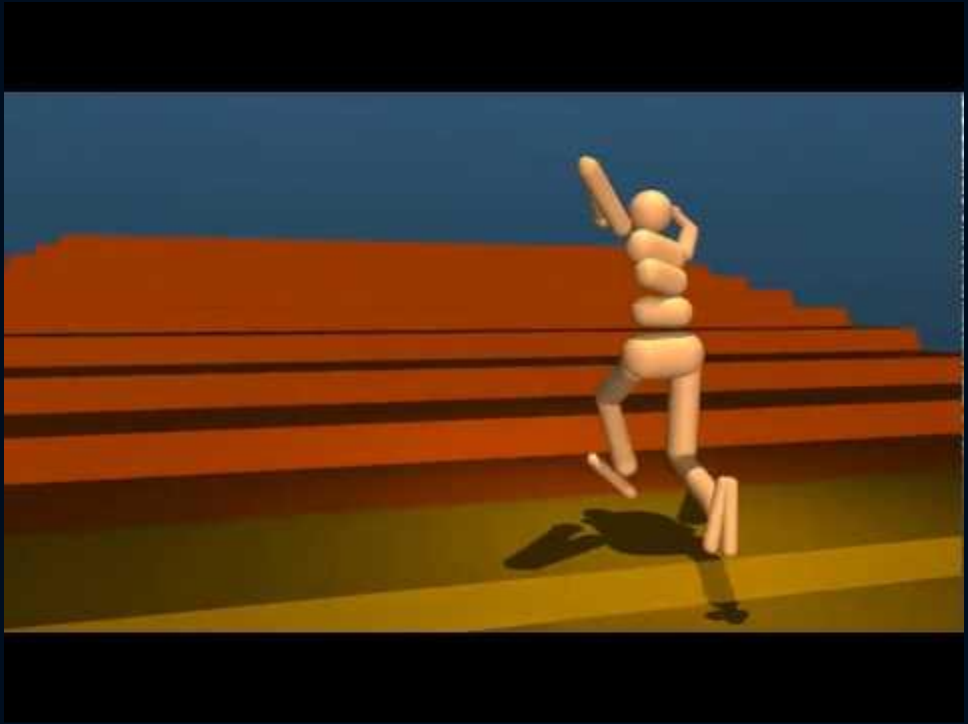
Successes

- 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

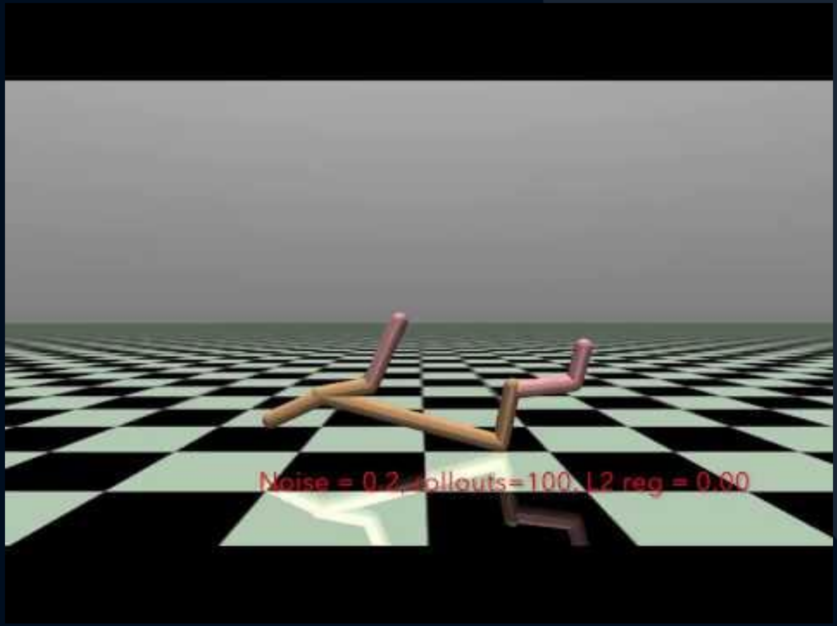


Parkour



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
- 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

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?

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

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

Q-table

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)

Regret-matching

RPS: We play rock, opponent plays paper

Regrets: 0 for rock, +1 for paper, +2 for scissors $\longrightarrow$ (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

- 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

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.

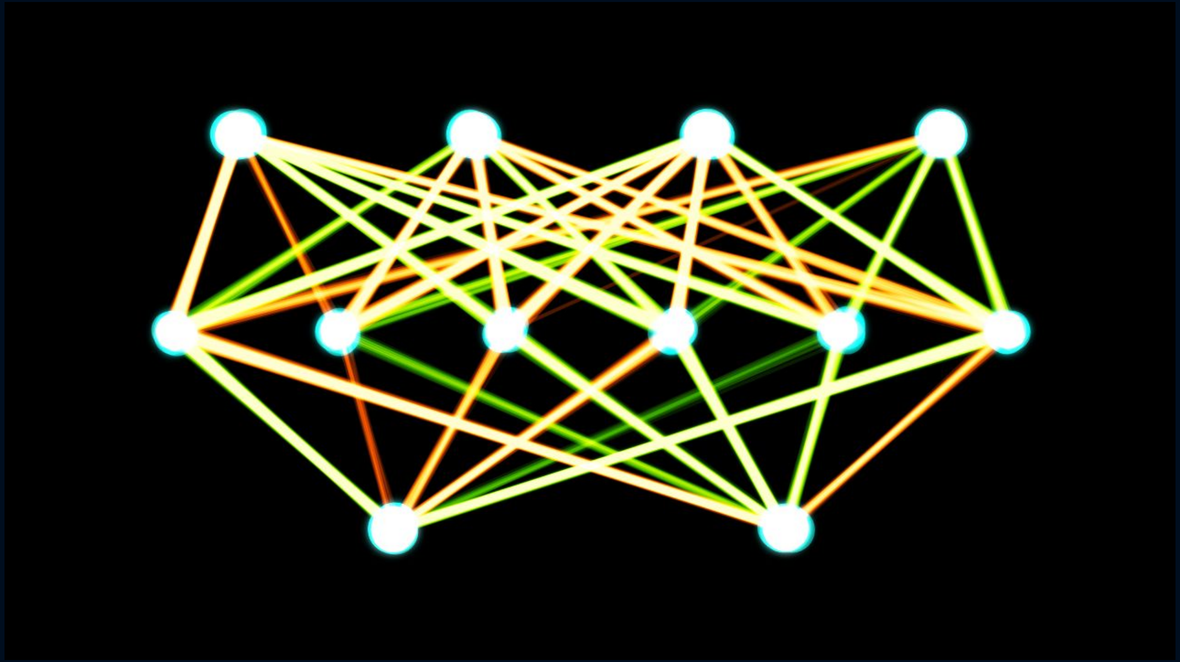
Problems and extensions

- The bucketing problem: how can we reduce the size of the game tree?
- Bot size limits
- 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?

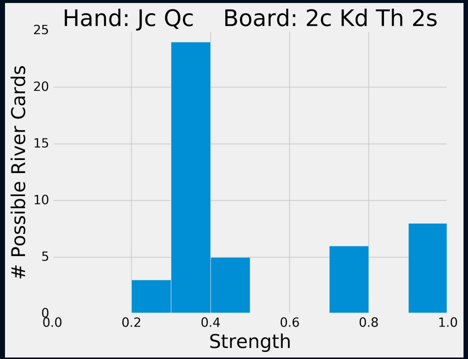
  - Ability to generalize to unseen data

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
  - Starting cards ⟶ best Blotto allocation
  - 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

Giveaway Winners!

Alliance Forming!