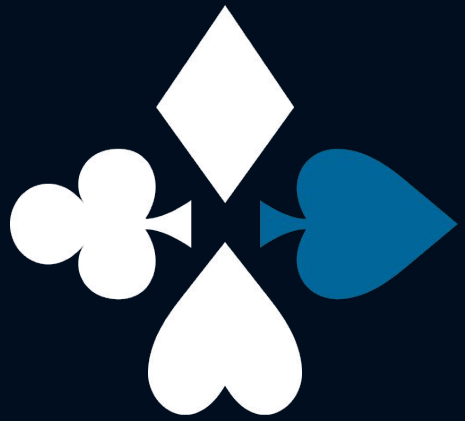


Pokerbots 2025

Lecture 6: Reinforcement Learning

Sponsors

CITADEL | CITADEL Securities

Announcements

reference-5-2025

- Coding demo available on GitHub
- ML pipeline using Pandas and scikit-learn
- Recording link in readme

Last Night's Lightning Tournament

1. Encore 1/2 Regs	$400
2. DKE juniors	$300
3. a	$225
4. all_luck_no_skill	$175
5. pocket fools	$150

Hackathon Tonight!

- Sponsored by **Codeium**
- 32-044, 7pm -- LATE
- Show up and grind
- Dinner provided
- Snacks, fun, and games
- Prizes for challenges and those that stick around
- Details will be announced on Piazza

No Class Tomorrow

Cancelled in light of Hackathon the night before and Codeium Tech Talk at 11:30am

Week 2 Bot Deadline

- First pokerbot submission due Friday 1/17, 11:59PM EST on scrimmage server
- Mini-tournament 2 will occur shortly after

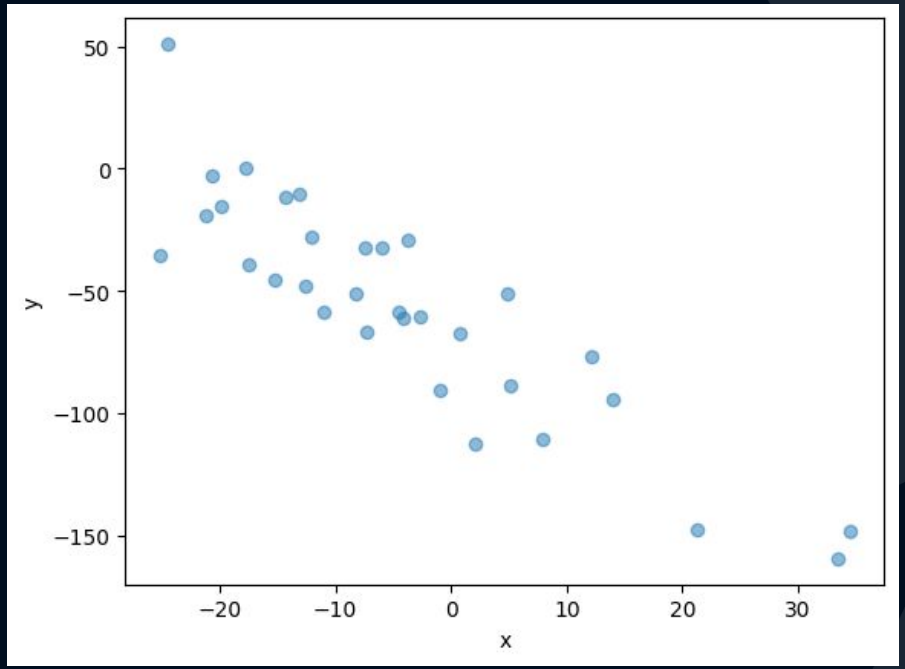
No Class Monday

Happy MLK Day!

Giveaways!

Regression: **pkr.bot/regression**

- Submit parameters (w,b) of linear model
  $y = w \cdot x + b$
- Winner decided by Mean Squared Error on displayed data
- Prize: Sony XM4



Reinforcement Learning

From yesterday

- Supervised Learning
  - Dataset has examples of correct behavior
  - “Questions with answers”
  - Prediction tasks
- Unsupervised Learning
  - Dataset has no notion of good behavior
  - Task is to find patterns to represent dataset
  - Clustering, compression, generative modeling
- Reinforcement Learning (RL)
  - Environment with feedback (rewards), but no correct target
  - Sequential decision making, control, pokerbots!

Overview and Landscape

Reinforcement Learning Overview

Definition: Training an agent to make decisions by interacting with an environment to maximize some reward signal aggregated over many steps.

The task usually involves sequentially taking actions and observing their effects.

Examples:

- Most board/card/video games
- Control: robotics, self-driving cars
- Recommendation: advertisements
- Agent settings: chatbots

Contrast with other types of Learning

Supervised and unsupervised learning analyze a fixed dataset and seek a model that closely aligns with that dataset.

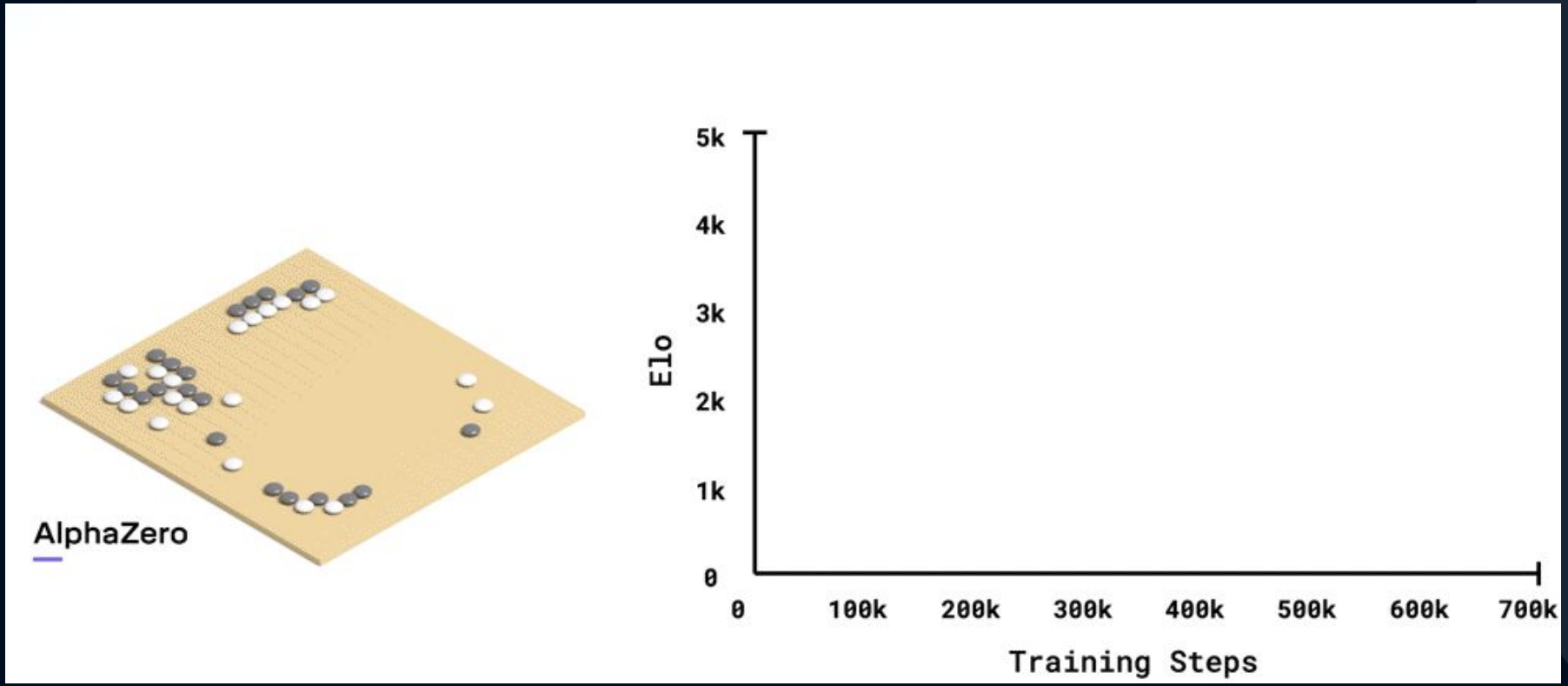
Reinforcement learning works *within* an environment and seek a behavior that performs well

- The data is the history of interactions with the environment
- Not necessarily fixed if more interactions can be made - sometimes deciding how to interact in order to collect more data is the hard part
- Sometimes the environment itself still needs to be studied, as this can help with navigating to get rewards

Successes

- AlphaZero: trained entirely from self-play
  - Beat best-in-world chess engine (stockfish) 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
  - Complex environments require lots of exploration

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

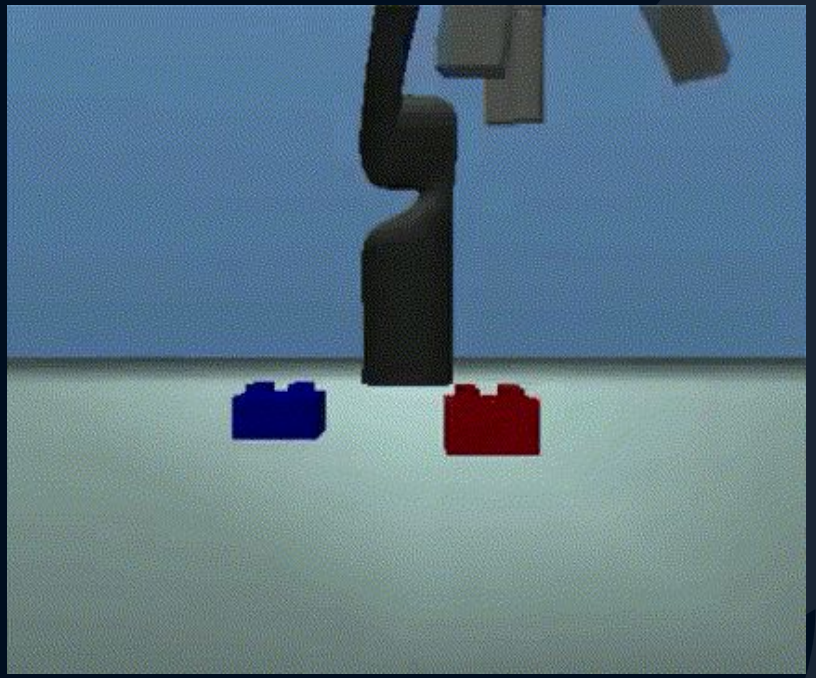
- In chess, if bot1 loses to bot2 and bot2 loses to bot3, then there is a good chance that bot3 is the strongest chess player
- Gives a clear route to *iterated improvement* with techniques such as *self-play*
- This is far from guaranteed in poker
- Game theoretic settings don’t have the luxuries of supervised learning

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 cycle without making meaningful improvements to performance
- A: rock ⟶ B: paper ⟶ A: scissors ⟶ B: rock ⟶
  A: paper ⟶ B: scissors ⟶ A: rock…
- Cycles are predictable, which is undesirable

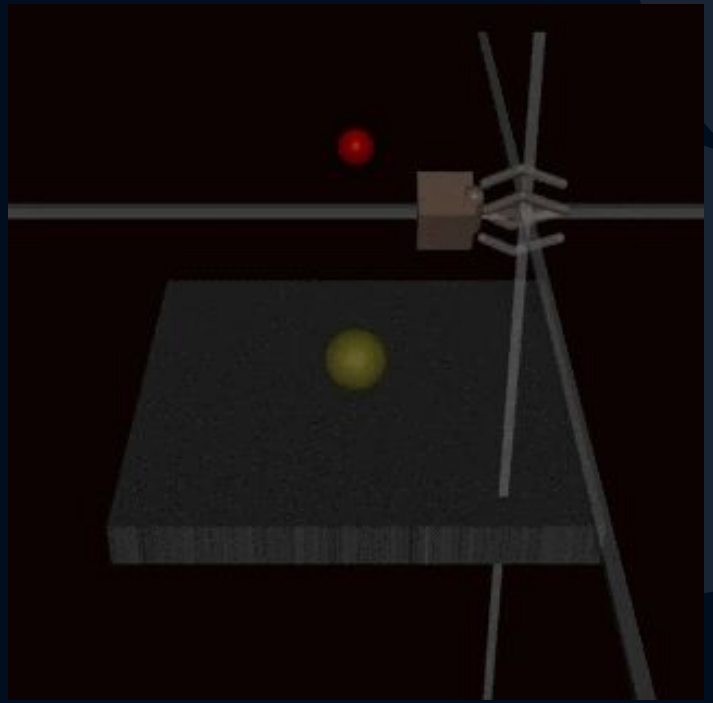
What to reward?

- Designing a good reward function is hard
- Luckily in poker, the job is done for us
- Specification gaming

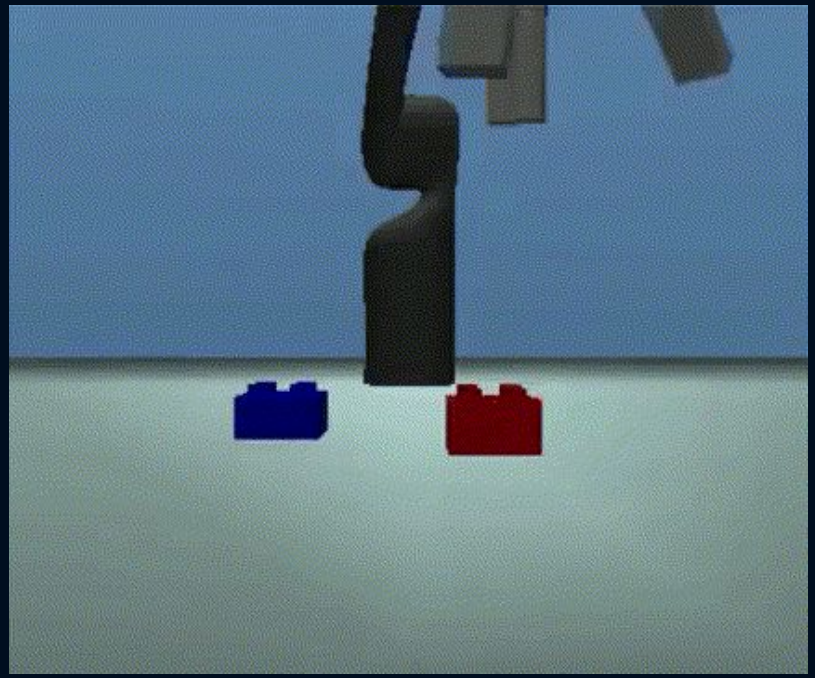


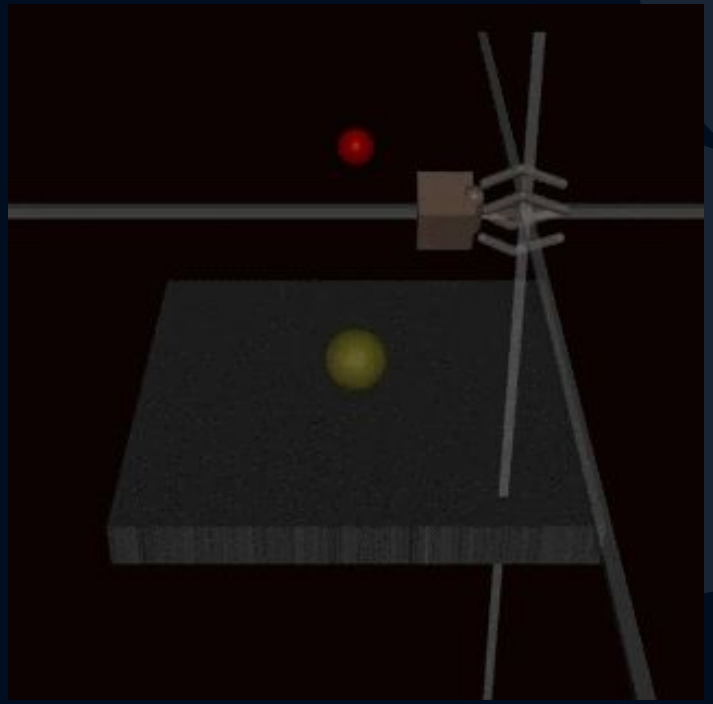
Specification Gaming

- Example: ChatGPT
- Reinforcement Learning from Human Feedback (RLHF)



Specification Gaming





Suppose we've considered all the warnings...

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

Formalization

Problem Setting

- Agent: Learner/decision-maker.
- Environment: World the agent interacts with.
- State: Current situation.
- Action: Decision made by the agent.
- Policy: How our agent decides which action to take.
- Reward: Feedback for the action.

Trajectory:

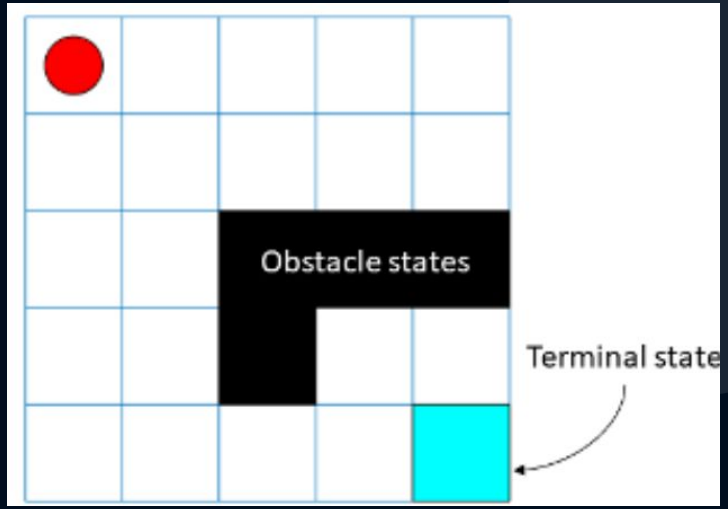
- (initial) state
- Action (chosen by agent's policy)
- Reward (environment
- Next state (environment)
- Action
- ...

Problem Setting

- State Space S
- Action Space A
- Reward Function $R(s,a)$
- Transition Function $P(s' \mid s,a)$
- Objective Expected sum of rewards

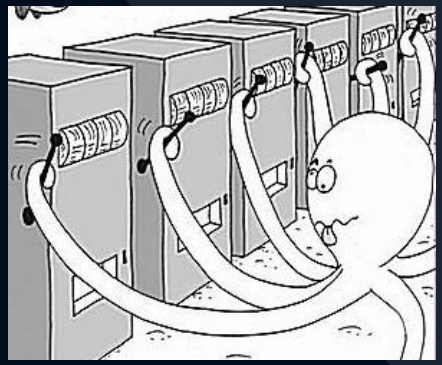
Example: Gridworld

- Example:
  - Environment: Gridworld.
  - State: Current grid position.
  - Actions: Up, Down, Left, Right. (unless obstacles
  - Reward: +10 for reaching terminal state, -1 for each step.
- Design rewards for goal of reaching terminal state as quick as possible



Simpler 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
- Examples:
  - Recommendation feeds (tiktok)
  - Choosing a restaurant to eat at



Approaching the Bandit Problem

- We would like to figure out which actions are the most fruitful (long term gain), while also using our turns to take actions that we already know are fruitful (short term gain)
- This tradeoff between *exploration* and *exploitation* is a general tradeoff that appears all throughout RL
- Biphasic Approach
  - randomly choose for some predetermined amount of steps
  - afterwards only pick the empirical best
- Epsilon-greedy
  - With some small probability, take a random action
  - Otherwise pick the action with the best average history of rewards

Bandits in Pokerbots: Learning across hands

- We have a collection of bots with different behaviors
- At each hand (or run of 25 hands), we pick a bot to use to decide the action
- Leads to some random reward (chip delta)
- Finding the bot that does well against our particular opponent across hands can help us counter them with an exploitative strategy
- This can be good for building on existing bots that don't have an obvious one that's better, and don't carry state variables across rounds/runs

RL Paradigms

Some General Approaches

We want to define our agent behavior using a function that evaluates its input:

- Q-learning: actions
- Value Iteration/Approximation: states
- Policy Methods: agent’s policy itself

Q-learning

- Let's return to thinking of poker as a multi-step process
    - Extensive form instead of matrix form in the game theory lecture
- Want to find a value function $Q(s, a)$ for each state-action pair representing how good it is to be in that state taking that action
- If it's accurate, then we simply choose the action that has the highest stored value!
- 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

More downsides

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

Value Iteration

- Instead of a function for every state-action pair, seek a function $V(s)$ that reports the value of being in a given state
- To choose actions, we try to pick one that leads to the highest next state (or average if nondeterministic transitions)
- This can save compared to Q-learning if action space is large
- However it still faces many of the other downsides in Q-learning

Policy Methods

Direct Policy Optimization:

- Our behavior is directly governed by a *policy* $\pi(a|s)$ that tells us how to (possibly randomly) choose an action from a given state.
    - Player.get_action
- If we believe there's some idea of correct actions, then we can apply supervised approaches

Actor-Critic Methods:

- Actor: Updates policy.
- Critic: Evaluates policy using value function.

Lunch 😋

Leave any type of feedback at **pkr.bot/feedback**!

Live Coding: reference-6-2025

Idea for Bandit Bot

At the beginning of every match:

- Initialize which bots we'll use

At new round:

- Pick which bot to use for the round

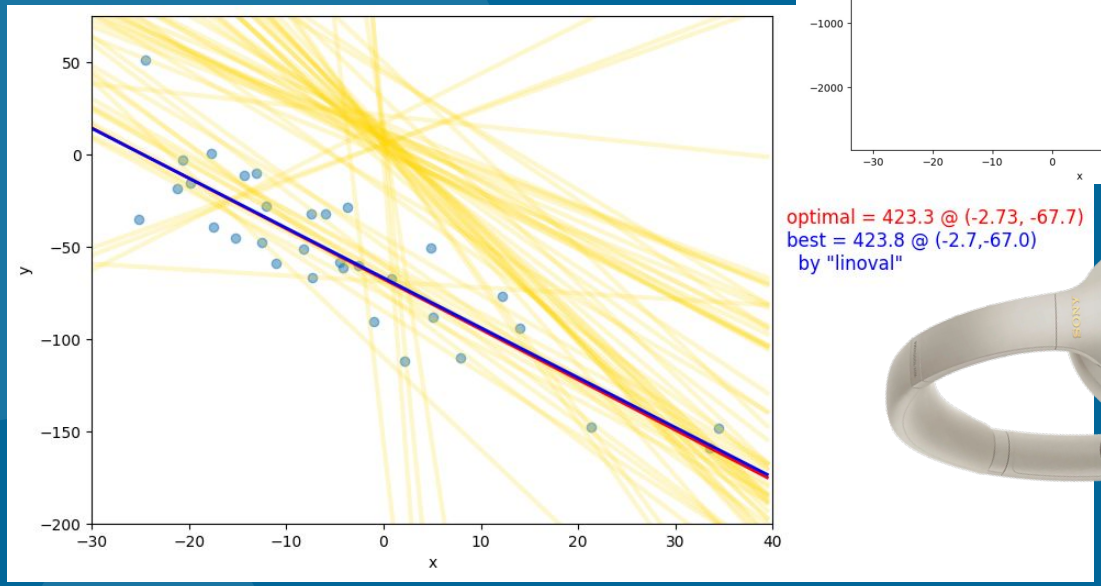
At each action:

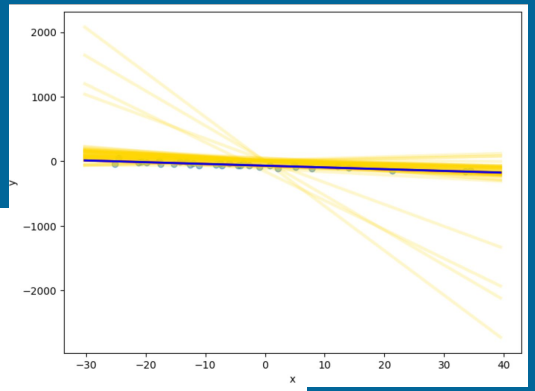
- Use the stored bot

At end of round:

- Update statistics based on delta

Regression Game: kerb “linoval”





optimal = 423.3 @ (-2.73, -67.7)
best = 423.8 @ (-2.7,-67.0)
by "linoval"

	MSE	(w,b)
OPTIMAL	423.3	(-2.73, -67.7)
linoval	423.8	(-2.7, -67.0)
brendonj	427.6	(-2.6, -68.0)
juszha	432.6	(-2.9, -70.0)
kevinmz	454.2	(-2.94, -64.0)
...	...	...
mandoj	17740.5	(-5.0, 50.0)
joyzhuo	261995.1	(-34.0, 13.0)
leanntai	404544.8	(-45.0, -156.0)
kjiang77	670438.5	(-54.0, 13.0)
chenl9	1091083.4	(-69.0, -3.141592)

64 rows × 2 columns

Thanks for watching!

Slides/notes will be posted on **pkr.bot/resources**

Make sure to check **pkr.bot/piazza** for updates

Lecture recordings at **pkr.bot/panopto**

Leave feedback at **pkr.bot/feedback**!