Finding counterexamples via reinforcement learning

Adam Zsolt Wagner

Hausdorff School: “Machine Learning and Theorem Proving”

Day 1

September 18, 2023

Lecture series overview

There are many exciting ways machine learning has affected, or will affect mathematics. One of these is the emergence of better and better Large Language Models, such as ChatGPT.

Lecture series overview

There are many exciting ways machine learning has affected, or will affect mathematics. One of these is the emergence of better and better Large Language Models, such as ChatGPT.

The reasoning capabilities of LLMs is a fascinating topic, but we will not talk about these at all in this lecture series.

Lecture series overview

There are many exciting ways machine learning has affected, or will affect mathematics. One of these is the emergence of better and better Large Language Models, such as ChatGPT.

The reasoning capabilities of LLMs is a fascinating topic, but we will not talk about these at all in this lecture series.

Instead we will focus on some specific ways you can apply ML tools to some particular research problems in practice.

Lecture series overview

There are many exciting ways machine learning has affected, or will affect mathematics. One of these is the emergence of better and better Large Language Models, such as ChatGPT.

The reasoning capabilities of LLMs is a fascinating topic, but we will not talk about these at all in this lecture series.

Instead we will focus on some specific ways you can apply ML tools to some particular research problems in practice.

- Day 1: Reinforcement learning
- Day 2: Saliency analysis
- Day 3: Transformers, Makemore

Day 1 overview

Main idea: RL algorithms have managed to reach superhuman level play in Atari games, Go, Chess, starting from only the rules and learning everything else by themselves.

Day 1 overview

Main idea: RL algorithms have managed to reach superhuman level play in Atari games, Go, Chess, starting from only the rules and learning everything else by themselves.

Can we teach neural networks to reach superhuman level play in the "game" of constructing graphs without 4-cycles, with as many edges as possible?

Day 1 overview

Main idea: RL algorithms have managed to reach superhuman level play in Atari games, Go, Chess, starting from only the rules and learning everything else by themselves.

Can we teach neural networks to reach superhuman level play in the "game" of constructing graphs without 4-cycles, with as many edges as possible?

Can this same algorithm be used to try to learn to disprove any conjecture, by only inputting the statement and letting the algorithm figure out the rest?

Reinforcement learning

- An agent (player) plays a game many times.

Reinforcement learning

- An agent (player) plays a game many times.
- It knows what buttons it is allowed to press, but doesn't know what each button does.

Reinforcement learning

- An agent (player) plays a game many times.
- It knows what buttons it is allowed to press, but doesn't know what each button does.
- Throughout the game, we give it rewards based on how well it did.

Reinforcement learning

- An agent (player) plays a game many times.
- It knows what buttons it is allowed to press, but doesn't know what each button does.
- Throughout the game, we give it rewards based on how well it did.
- The agent gathers experience:

Reinforcement learning

- An agent (player) plays a game many times.
- It knows what buttons it is allowed to press, but doesn't know what each button does.
- Throughout the game, we give it rewards based on how well it did.
- The agent gathers experience:
  - given a state and action (button press) I did, what was the resulting next state?

Reinforcement learning

- An agent (player) plays a game many times.
- It knows what buttons it is allowed to press, but doesn't know what each button does.
- Throughout the game, we give it rewards based on how well it did.
- The agent gathers experience:
  - given a state and action (button press) I did, what was the resulting next state?
  - Given a state and action (button press), how much reward did I receive?

Reinforcement learning

- An agent (player) plays a game many times.
- It knows what buttons it is allowed to press, but doesn't know what each button does.
- Throughout the game, we give it rewards based on how well it did.
- The agent gathers experience:
  - given a state and action (button press) I did, what was the resulting next state?
  - Given a state and action (button press), how much reward did I receive?
- Agent tries to improve his total score (sum of all rewards) through some optimization algorithm.

Plan

Goal: find counterexamples to open conjectures via RL

Plan

Goal: find counterexamples to open conjectures via RL

- Try to avoid using human insights as much as possible

Plan

Goal: find counterexamples to open conjectures via RL

- Try to avoid using human insights as much as possible
- Would like a general setup: use the same program for every problem, only change reward function

Plan

Goal: find counterexamples to open conjectures via RL

- Try to avoid using human insights as much as possible
- Would like a general setup: use the same program for every problem, only change reward function
- Throw this setup at 100 open conjectures and hope for the best.

Example 1

Conjecture

For any graph G, we have $\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$.

Example 1

Conjecture

For any graph G, we have $\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$.

Refuted in 2010, but smallest counterexample found has 600 vertices.

Example 1

Conjecture

For any graph G, we have $\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$.

Refuted in 2010, but smallest counterexample found has 600 vertices.

Game: offer edges one by one, agent can accept/reject each. A game lasts $\frac{n(n-1)}{2}$ turns.

Example 1

Conjecture

For any graph G, we have $\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$.

Refuted in 2010, but smallest counterexample found has 600 vertices.

Game: offer edges one by one, agent can accept/reject each. A game lasts $\frac{n(n-1)}{2}$ turns.

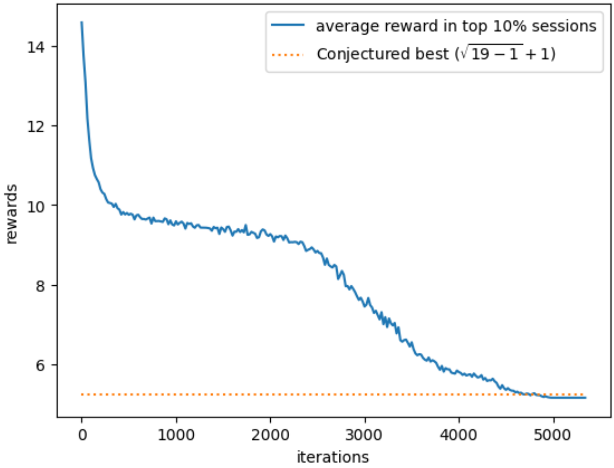
Reward: $\lambda_1 + \mu$ (minimize).

Run a reinforcement learning algorithm for $n = 19$:

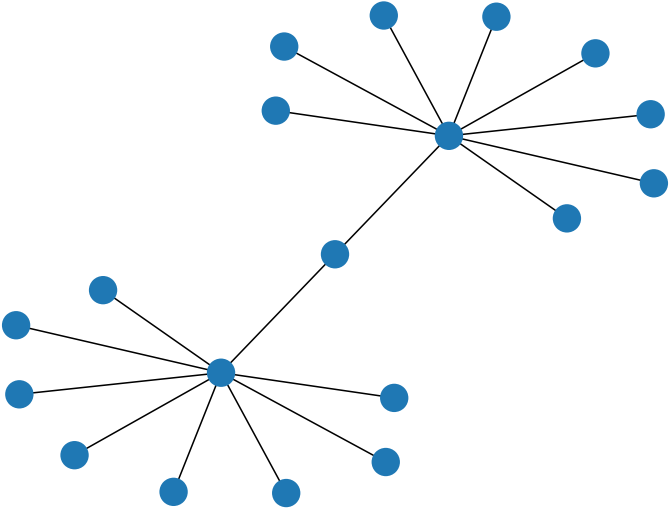
Example 1

Conjecture

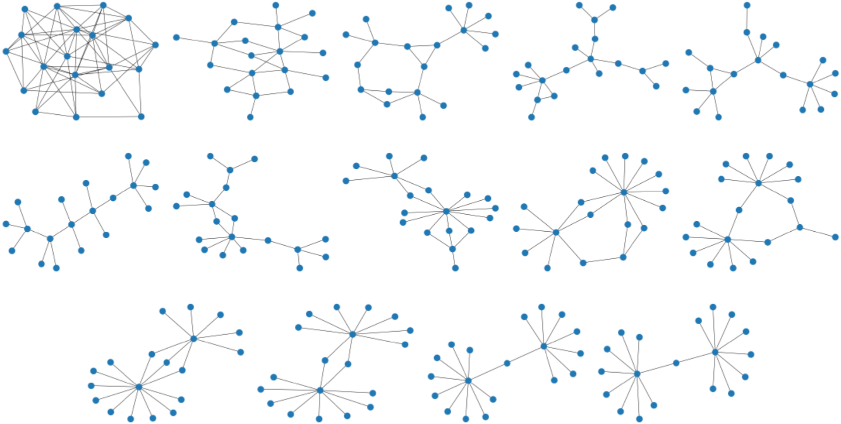
For any graph, $\lambda_1 + \mu \geq \sqrt{n-1} + 1$.



Example 1



Example 1



Example 1

- This was the dream scenario: there is an obvious way to phrase the conjecture as a game, there is an obvious choice for the score function, we plug these into the RL program and it spits out a counterexample.

Example 1

- This was the dream scenario: there is an obvious way to phrase the conjecture as a game, there is an obvious choice for the score function, we plug these into the RL program and it spits out a counterexample.

- When this happens, there is not much to talk about. But often it is not that simple.

Example 1

- This was the dream scenario: there is an obvious way to phrase the conjecture as a game, there is an obvious choice for the score function, we plug these into the RL program and it spits out a counterexample.
- When this happens, there is not much to talk about. But often it is not that simple.
- We will see 5 more examples. In each of them we will succeed in refuting an open conjecture, but each example will illustrate a unique thing that could "go wrong" and how to overcome it.

Example 2 – What if we don't succeed?

Conjecture (Auchiche–Hansen, 2016)

Let G be a connected graph with diameter D, proximity π and distance spectrum $\partial_1 \geq \ldots \geq \partial_n$. Then

$$\pi + \partial_{\lfloor \frac{2D}{3} \rfloor} > 0.$$

Example 2 – What if we don't succeed?

Conjecture (Auchiche–Hansen, 2016)

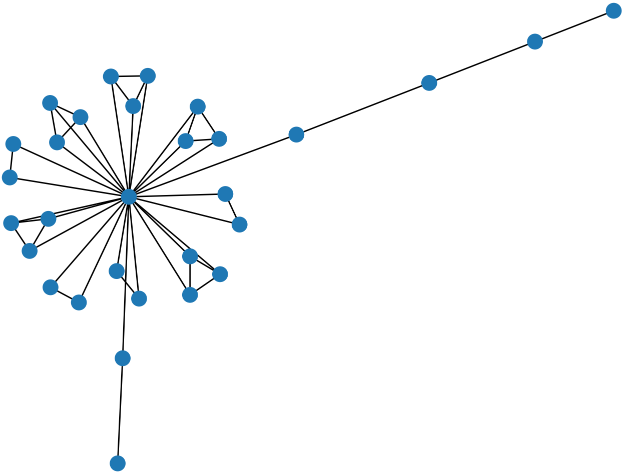
Let G be a connected graph with diameter D, proximity π and distance spectrum $\partial_1 \geq \ldots \geq \partial_n$. Then

$$\pi + \partial_{\lfloor \frac{2D}{3} \rfloor} > 0.$$

Reward: $\pi + \partial_{\lfloor \frac{2D}{3} \rfloor}$ (minimize).

Run it for $n = 30$:

Example 2



This is not quite a counterexample ($\pi + \partial_{\lfloor \frac{2D}{3} \rfloor} \approx 0.4$), but it tells us very clearly what counterexamples could look like.

Example 2

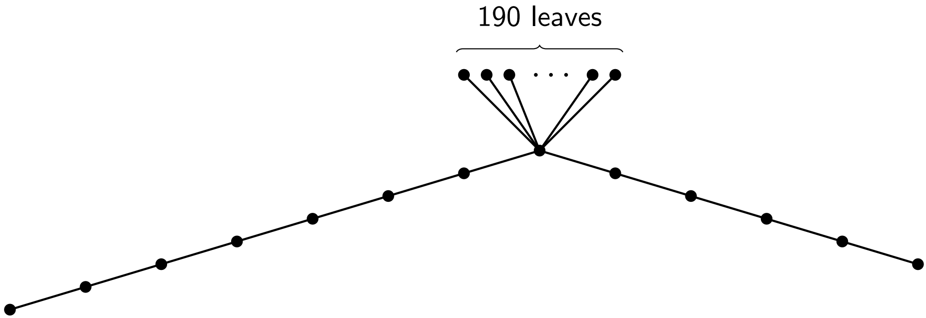


Figure: A counterexample to the conjecture

Question (Brualdi–Cao)

How large can the permanent of a 312-pattern avoiding 0-1 matrix be?

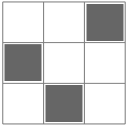


Figure: The pattern 312

$$\mathrm{per}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

Example 3 - Not just graphs

Question (Brualdi–Cao)

How large can the permanent of a 312-pattern avoiding 0-1 matrix be?

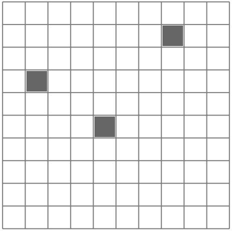


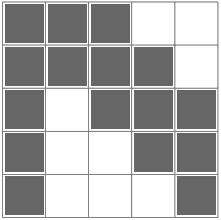
Figure: This is also not allowed

More precisely: we are not allowed to have three ones (dark squares) $(x_i, y_i) : i \in \{1, 2, 3\}$ such that $y_1 < y_2 < y_3$ and $x_2 < x_1 < x_3$.

Example 3

Conjecture (Brualdi–Cao, 2020)

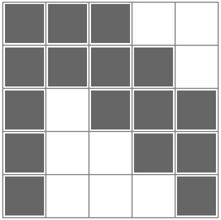
The best you can do is $Fib_{n+2} - 1$.



Example 3

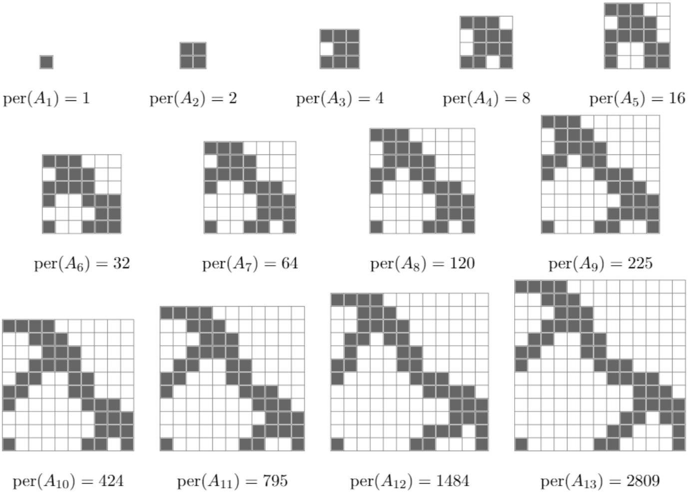
Conjecture (Brualdi–Cao, 2020)

The best you can do is $Fib_{n+2} - 1$.



Reward: $\text{per}(A) - \text{penalty}(\# \text{ of } 312\text{-s})$

Example 3



These are best possible for $n \leq 8$ (computer proof). So the sequence starts with $1, 2, 4, 8, 16, 32, 64, \mathbf{120}$.

Example 4 - Problems on trees

Conjecture (Collins, 1989)

Given a tree T, let $p(T)$ and $q(T)$ be the characteristic polynomials of the adjacency and the distance matrices of T, respectively. The coefficients of p and q are both unimodal, and their peaks are asymptotically at the same place.

Example 4 - Problems on trees

Conjecture (Collins, 1989)

Given a tree T, let $p(T)$ and $q(T)$ be the characteristic polynomials of the adjacency and the distance matrices of T, respectively. The coefficients of p and q are both unimodal, and their peaks are asymptotically at the same place.

Bad idea: generate graphs, reward function $= f_1(G) + f_2(G)$ where f_1 measures how close to a tree G is, and f_2 measures how far the peak coefficients are.

Example 4 - Problems on trees

Conjecture (Collins, 1989)

Given a tree T, let $p(T)$ and $q(T)$ be the characteristic polynomials of the adjacency and the distance matrices of T, respectively. The coefficients of p and q are both unimodal, and their peaks are asymptotically at the same place.

Bad idea: generate graphs, reward function $= f_1(G) + f_2(G)$ where f_1 measures how close to a tree G is, and f_2 measures how far the peak coefficients are.

Better: There is a bijection between trees and $[n]^{n-2}$ (Prüfer code). The game will last $n-2$ turns, in each turn we can make n possible decisions.

Example 4 - Problems on trees

Conjecture (Collins, 1989)

Given a tree T, let $p(T)$ and $q(T)$ be the characteristic polynomials of the adjacency and the distance matrices of T, respectively. The coefficients of p and q are both unimodal, and their peaks are asymptotically at the same place.

Bad idea: generate graphs, reward function $= f_1(G) + f_2(G)$ where f_1 measures how close to a tree G is, and f_2 measures how far the peak coefficients are.

Better: There is a bijection between trees and $[n]^{n-2}$ (Prüfer code). The game will last $n-2$ turns, in each turn we can make n possible decisions.

Reward: distance of the peaks.

Example 4 - Problems on trees

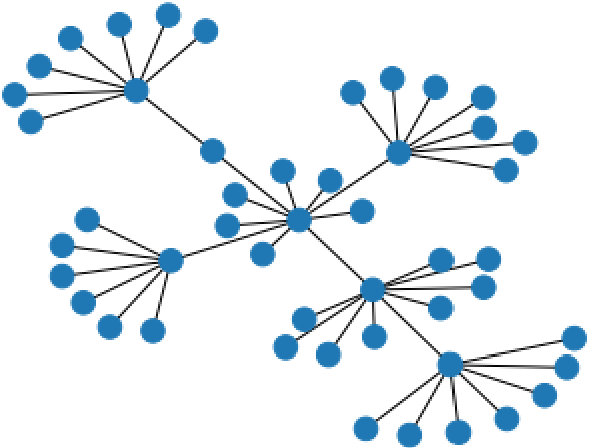


Figure: Best construction found for $n = 48$

Example 5 - non-obvious reward function

To a graph on n vertices, we can associate several $n \times n$ matrices. One example is the adjacency matrix.

Example 5 - non-obvious reward function

To a graph on n vertices, we can associate several $n \times n$ matrices. One example is the adjacency matrix.

Another example is the distance Laplacian $\mathcal{D}^L(G)$. Its (m,m) entry is $\sum_{i=1}^{n} d(v_m, v_i)$ and its (i,j) entry for $i \neq j$ is $-d(v_i, v_j)$.

Example 5 - non-obvious reward function

To a graph on n vertices, we can associate several $n \times n$ matrices. One example is the adjacency matrix.

Another example is the distance Laplacian $\mathcal{D}^L(G)$. Its (m,m) entry is $\sum_{i=1}^{n} d(v_m, v_i)$ and its (i,j) entry for $i \neq j$ is $-d(v_i, v_j)$.

The multiset of eigenvalues of $\mathcal{D}^L(G)$ is denoted by $\text{spec}_{\mathcal{D}^L}(G)$.

Example 5 - non-obvious reward function

To a graph on n vertices, we can associate several $n \times n$ matrices. One example is the adjacency matrix.

Another example is the distance Laplacian $\mathcal{D}^L(G)$. Its (m, m) entry is $\sum_{i=1}^{n} d(v_m, v_i)$ and its (i, j) entry for $i \neq j$ is $-d(v_i, v_j)$.

The multiset of eigenvalues of $\mathcal{D}^L(G)$ is denoted by $\text{spec}_{\mathcal{D}^L}(G)$.

Say that a graph property $\mathcal{P}$ is *preserved under* $\mathcal{D}^L(G)$*-cospectrality* if $\text{spec}_{\mathcal{D}^L}(G) = \text{spec}_{\mathcal{D}^L}(H)$ implies $\mathcal{P}(G) = \mathcal{P}(H)$.

Example 5 - non-obvious reward function

Property	$\mathcal{D}^L$
# Edges	No
Diameter	No
Girth	No
Planarity	No
Wiener index	Yes
Degree sequence	No
Transmission sequence	No
Transmission regularity	?
# connected components in $\overline{G}$	Yes

Question (Hogben–Reinhart)

Is transmission regularity preserved under $\mathcal{D}^L$-cospectrality?

Example 5 - non-obvious reward function

Property	$\mathcal{D}^L$
# Edges	No
Diameter	No
Girth	No
Planarity	No
Wiener index	Yes
Degree sequence	No
Transmission sequence	No
Transmission regularity	?
# connected components in $\overline{G}$	Yes

Question (Hogben–Reinhart)

Is transmission regularity preserved under $\mathcal{D}^L$-cospectrality?

Task: find two graphs G and H such that they have the same $\mathcal{D}^L$-eigenvalues, but G is transmission regular and H is not.

Example 5 - non-obvious reward function

A graph is transmission regular, if for each vertex, the sum of distances to all other vertices is the same. So if $\sum_w d(v, w) = \sum_w d(u, w)$ for all vertices u, v.

Task: find two graphs G and H such that they have the same $\mathcal{D}^L$-eigenvalues, but G is transmission regular and H is not.

RL has constructed two graphs. What should the reward function be?

Example 5 - non-obvious reward function

A graph is transmission regular, if for each vertex, the sum of distances to all other vertices is the same. So if $\sum_w d(v,w) = \sum_w d(u,w)$ for all vertices u, v.

Task: find two graphs G and H such that they have the same $\mathcal{D}^L$-eigenvalues, but G is transmission regular and H is not.

RL has constructed two graphs. What should the reward function be?

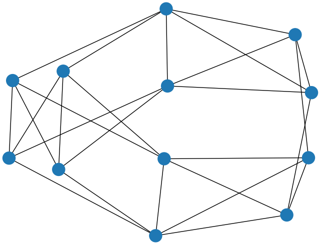
Idea:

$$\text{score}(G,H) = f_1(G,H) + f_2(G) + f_3(H),$$

where

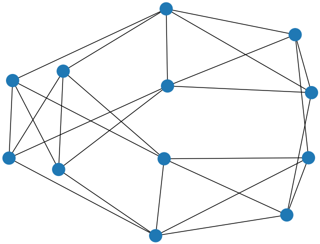
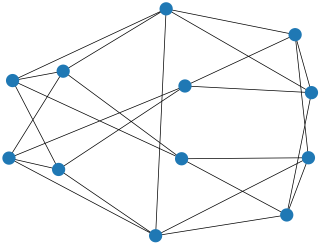
- f_1 measures how close the $\mathcal{D}^L$-spectrum of G and H is,
- f_2 measures how close G is to being transmission regular, and
- f_3 gives a penalty if H is transmission regular.

Example 5



The graph on the left is transmission regular, whereas the graph on the right is not. The characteristic polynomials of their distance Laplacians are the same $\left(x^{12} - 216x^{11} + 21188x^{10} - 1245904x^9 + 48797440x^8 - 1336652544x^7 + 26129121472x^6 - 364516883456x^5 + 3556516628224x^4 - 23113129559040x^3 + 90045806284800x^2 - 159318669312000x\right)$, so they are $\mathcal{D}^L$-cospectral.

Example 5

The graph on the left is transmission regular, whereas the graph on the right is not. The characteristic polynomials of their distance Laplacians are the same $\left(x^{12} - 216x^{11} + 21188x^{10} - 1245904x^{9} + 48797440x^{8} - 1336652544x^{7} + 26129121472x^{6} - 364516883456x^{5} + 3556516628224x^{4} - 23113129559040x^{3} + 90045806284800x^{2} - 159318669312000x\right)$, so they are $\mathcal{D}^L$-cospectral.

So transmission regularity is **not** preserved under $\mathcal{D}^L$-cospectrality.

Example 6 - Infinite problems?

Many interesting problems can not have finite counterexamples.

Conjecture (Erdős, 1962)

The function

$$K_4(G) + K_4(\bar{G})$$

is asymptotically minimized by random graphs.

Thomason (1989): This is false!

Example 6 - Infinite problems?

Many interesting problems can not have finite counterexamples.

Conjecture (Erdős, 1962)

The function

$$K_4(G) + K_4(\bar{G})$$

is asymptotically minimized by random graphs.

Thomason (1989): This is false!

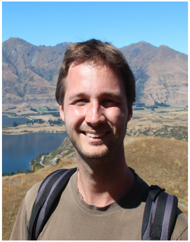


Figure: Gwenaël Joret

Example 6 - Infinite problems?

How can we refute such conjectures using RL?

- Find the best construction for $n = 50$, then generalize “by hand”.
- Somehow reduce to a finite conjecture.

Example 6 - Infinite problems?

How can we refute such conjectures using RL?

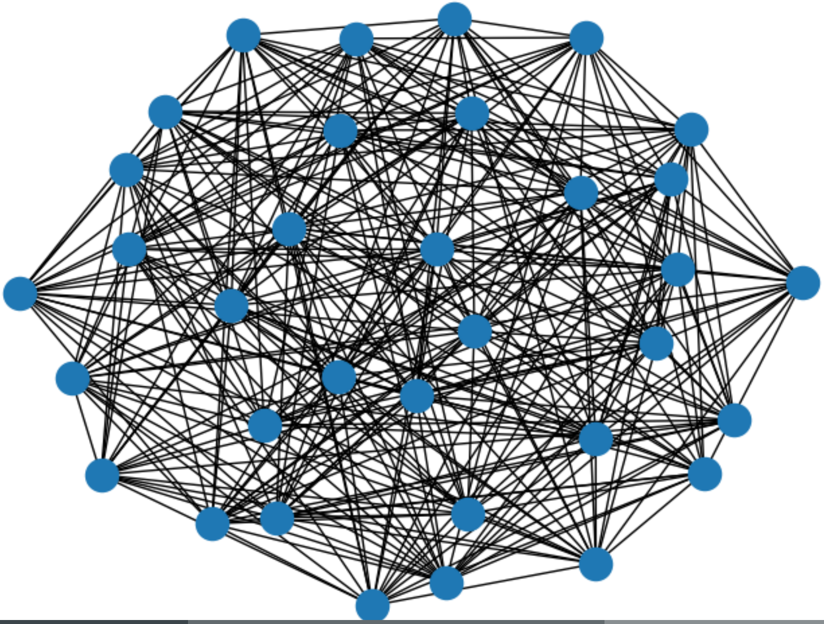
- Find the best construction for $n = 50$, then generalize "by hand".
- Somehow reduce to a finite conjecture.

Solution: "blowing up"! Construct a finite graph G, so that $G \times K_m$ is a counterexample as $m \to \infty$.

$\lim_{m\to\infty} \frac{K_4(G\times K_m)+K_4(\overline{G\times K_m})}{m^4}$ depends only on G, and there is an easy formula for it. This will be our reward function.

Run RL for $n = 34 \longrightarrow$ find a counterexample.

Example 6



Which RL algorithm to use?

- Value-based methods
  - Learn a value function: "I know how good this chess position is for black".
- Policy-based methods
  - Do not learn a value function: "I have no idea how good this chess position is for black, but I know the best move is c4".
- Mixed methods ("Actor-critic methods")

Which RL algorithm to use?

- Value-based methods
  - Learn a value function: "I know how good this chess position is for black".
- Policy-based methods
  - Do not learn a value function: "I have no idea how good this chess position is for black, but I know the best move is c4".
- Mixed methods ("Actor-critic methods")

Task: Find the best algorithm for our problems!

What RL setup to use?

Would like a general setup: use the same program for every problem, only change reward function.

What RL setup to use?

Would like a general setup: use the same program for every problem, only change reward function.

Implementation matters just as much as the algorithm choice. Many non-trivial decisions during implementation.

What RL setup to use?

Would like a general setup: use the same program for every problem, only change reward function.

Implementation matters just as much as the algorithm choice. Many non-trivial decisions during implementation.

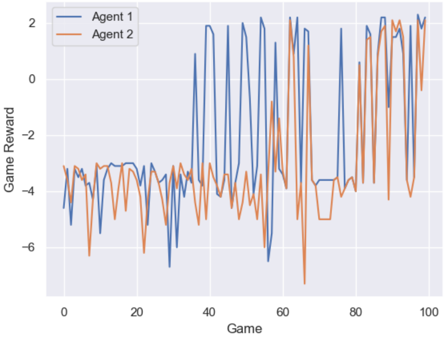
- Do we offer the edges one after the other, starting with $(1,2),(1,3)$, $\ldots(n-1,n)$?
- Or do we generate a graph vertex by vertex?
- We could repeatedly ask the neural network to pick one edge to add, out of all remaining edges?
- Do we offer edges in random order? Do we offer multiple edges at a time, is it beneficial to offer an edge again after it was rejected once?
- What neural network architecture to use? Dense layers may not be the best (doesn't understand symmetry).

Reasons an RL algorithm might not work

- Sparse rewards problem: we give rewards only at the end of a game.
- Credit assignment problem: which of my $\frac{n(n-1)}{2}$ moves was responsible for getting a bad score?
- Bad reward design
- Explore-exploit dilemma

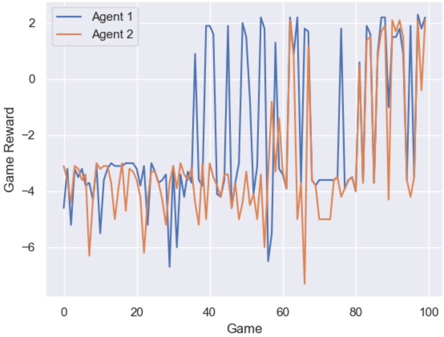
Practical problems

What do we conclude if the algorithm produces this output?



Practical problems

What do we conclude if the algorithm produces this output?



Learning rate too high? Some other hyperparameter is wrong? Not enough training? Coincidence? Maybe this algorithm is unsuited for this specific problem, but good for others?

Which RL algorithm to use?

More realistic goal: find an RL algorithm that works **well enough**.

Which RL algorithm to use?

More realistic goal: find an RL algorithm that works **well enough**.

Cross-entropy method:

- Simplest possible policy-based method.
- Not sensitive to hyperparameters $\implies$ can use same exact program for every problem.
- Fast convergence, very stable.
- This is the algorithm used in all the examples.

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly

- Input to neural network: a partial graph/matrix/word. Output: probability to take/reject next edge.

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly
- Input to neural network: a partial graph/matrix/word. Output: probability to take/reject next edge.
- Input empty construction, sample first edge according to output. Then input this new graph, sample second edge randomly. Repeat until decisions were made for all edges.

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly
- Input to neural network: a partial graph/matrix/word. Output: probability to take/reject next edge.
- Input empty construction, sample first edge according to output. Then input this new graph, sample second edge randomly. Repeat until decisions were made for all edges.
- Play 1000 games according to neural network

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly
- Input to neural network: a partial graph/matrix/word. Output: probability to take/reject next edge.
- Input empty construction, sample first edge according to output. Then input this new graph, sample second edge randomly. Repeat until decisions were made for all edges.
- Play 1000 games according to neural network
- Keep the top 100 with the highest score, throw away the rest

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly
- Input to neural network: a partial graph/matrix/word. Output: probability to take/reject next edge.
- Input empty construction, sample first edge according to output. Then input this new graph, sample second edge randomly. Repeat until decisions were made for all edges.
- Play 1000 games according to neural network
- Keep the top 100 with the highest score, throw away the rest
- For all (state, action) pairs in all top 100 games, adjust neural network weights to make it more likely that for this state we will choose this action.

Cross-entropy method

- There is no value function: use the neural network to predict the policy directly
- Input to neural network: a partial graph/matrix/word. Output: probability to take/reject next edge.
- Input empty construction, sample first edge according to output. Then input this new graph, sample second edge randomly. Repeat until decisions were made for all edges.
- Play 1000 games according to neural network
- Keep the top 100 with the highest score, throw away the rest
- For all (state, action) pairs in all top 100 games, adjust neural network weights to make it more likely that for this state we will choose this action.

Implementation details

- Offer edges one by one: $(1,2)$, then $(1,3)$, $\ldots$.
- Input: adjacency matrix we have created so far, and a matrix indicating which edge we consider now.
- Each game lasts $\frac{n(n-1)}{2}$ steps (if we generate a graph).
- The input is two vectors of length $n(n-1)/2$ (or equivalent). The first contains 1-s for each edge we have taken, and 0-s for each edge rejected, or not considered yet. The second is all zeros, except for one place, corresponding to the next edge offered.
- Architecture: dense net, three layers of sizes 128, 64, 4.
- Learn from top 10%, but keep the top 5% for the next iteration.

Improvements to the method

Don't do a generalist approach, focus on one conjecture only, and find the best setup for this one problem!

- Pick an architecture that takes the symmetries of the problem into account (transformers, GNNs, canonicalizing the data)
- We might know that the counterexample must have a specific structure $\rightarrow$ use it to massively restrict the search space