Machine Learning tools for String Theory

string_data 2018

Munich, 29.03.2018

Jim Halverson,
Northeastern University

Fabian Ruehle,
University of Oxford

Techniques from data science could lead to interesting physics results in many areas, and already have.

Even in string theory, one could use data science to study formal systems.

Instead, applications here will tend to focus one important big data problem: the string landscape.

Many “Landscapes”

Boldness / Controversy

Mostly Anthropic Multiverse

Partially Anthropic Multiverse

Many Pockets: Multiverse

Many Vacua

Many Possibilities

“Although string theory clearly pointed towards an “anthropic ensemble” of solutions since 1985, mentioning this in public was “not done”. Even nowadays, one can attend an entire conference on “string phenomenology” without any mention of anthropic arguments, except occasionally in a derogatory way.”

- *Schellekens,* The Emperor’s Last Clothes

KEY POINT:

regardless of which one each of us *currently* thinks is correct, many of us probably agree that this problem is very **important and difficult**, and involves **enormous datasets**.

(probably) so big we will never be able to process them on any conceivable classical computer.

what do we do!?

FORMAL THEORY

is clearly ***required*** to
better understand the landscape,
the swampland, and
what they mean for the real world.

but will it be enough on its own? (**no?**)

Can we make progress using the suite of techniques from data science? (**finding out . . .**)

Can this progress include rigorous results? (**yes**)

But can data science handle sets that will never fit on a computer!? (**sometimes**)

Can results obtained from data science provide motivation for formal theory? (**yes**)

Areas of Application

Predict / Classify (Fabian)

- Neural Networks (super- / unsupervised)

Make Rigorous (Jim)

- Intelligible AI
- Conjecture Generation

Data Generation (Fabian)

- Generative Adversarial Networks (GANs)

Search / Explore (Jim)

- Reinforcement Learning (RL)

Data Structure (Fabian)

- Persistent Homology

Connections (Jim)

- Graph Theory
- Network Science

Predict / Classify w/ Neural Networks

[Krefl,Seong`17], [He`17], [Ruehle `17], [Liu`17]

Some Textbooks

- Explains backpropagating NNs and train via genetic alg. Good textbook explaining standard techniques

  Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks
  Reed, Marks
  MIT Press, 1999

- Also gives intuition for NN from to actual neurons (bio)

  An Introduction To Neural Networks
  James A Anderson
  MIT Press, 1995.

- Standard textbook

  Neural Networks for Pattern Recognition
  Christopher M. Bishop
  Oxford press, 1995

- Online textbook (animated examples)

  Neural Networks and Deep Learning
  Michael Nielsen
  [http://neuralnetworksanddeeplearning.com], 2017

Some Libraries

- Tensorflow: Python library for neural networks (and much more in fact)
  - ⊕ Most-used in CS community ⇒ tons of examples (check github.com)
  - ⊕ Comes with CUDA / GPU support (in principle)
  - ⊕ Comes with tensor board (visualize flow, NN, …)
  - ⊖ Some things are cumbersome (e.g. storing trained NNs, Tensorboard takes some getting used to, …)

[tensorflow.com]

- Mathematica: 11.1 onward comes with NN functionality
  - ⊕ Most of us are familiar with mathematica (more than Python)
  - ⊕ Comes with CUDA / GPU support (in principle)
  - ⊕ Comes with tons of high-level layers and functionality (e.g. you can do `NetTrain[myNN, trainSet]`, …)
  - ⊖ Slower than Python
  - ⊖ Less examples than for e.g. Tensorflow

[mathematica.com]

Some Libraries

- Keras: Python library on top of Tensorflow / Theano

  - \+ Provides higher level functions for NN creation and training
  - \+ Uses Tensorflow or Keras in the back to do the computations (wrapper for these libraries)
  - \- If you want to do things that are not implemented in eras need to know Tensorflow anyways

[keras.io]

- Chainer: Yet another Python Library

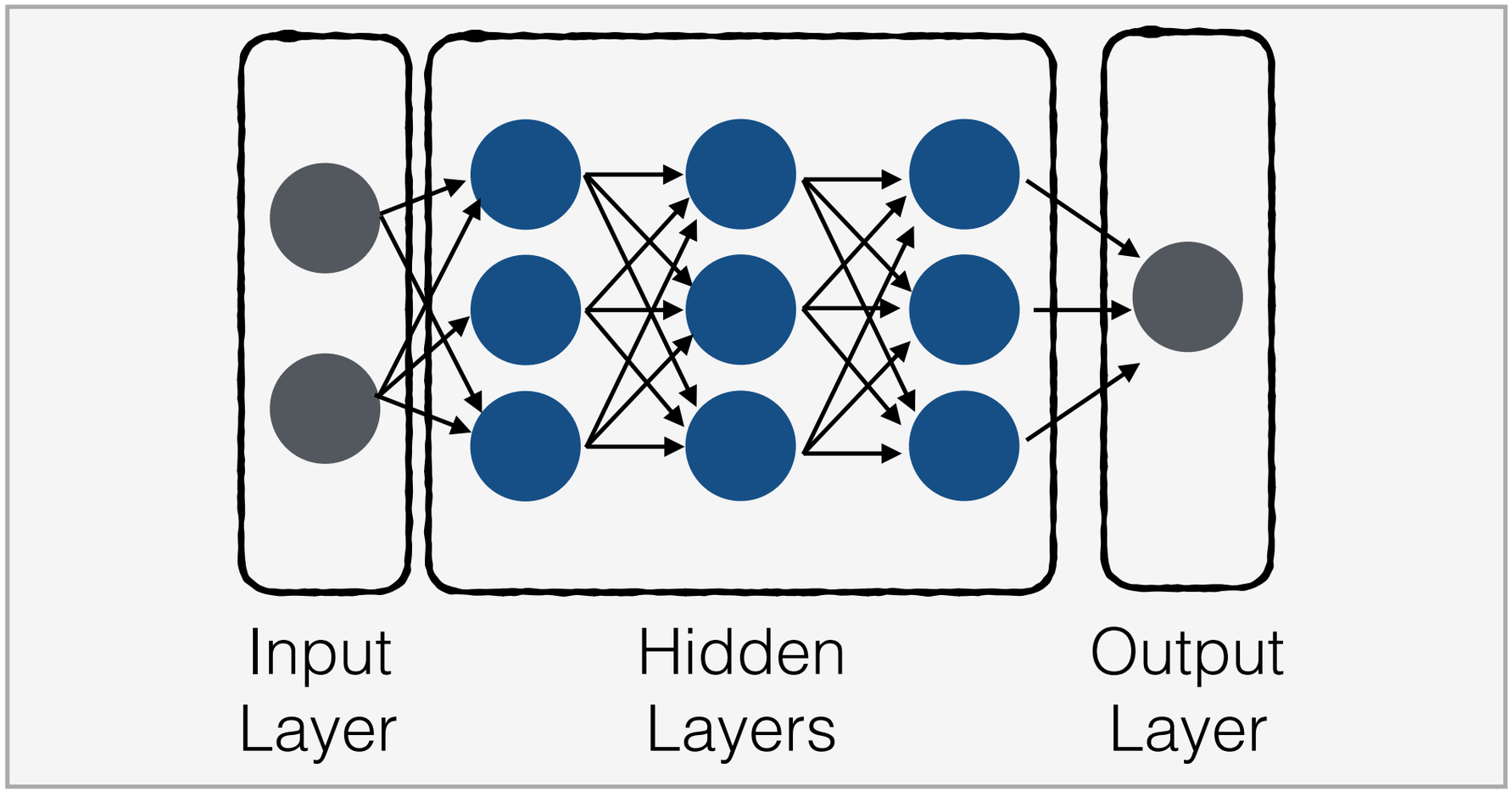
  - \+ ChainerRL library provides Reinforcement Learning
  - \+ Examples on GitHub how to connect this to OpenAI Gym
  - \- Less used than Tensorflow

[chainer.org]

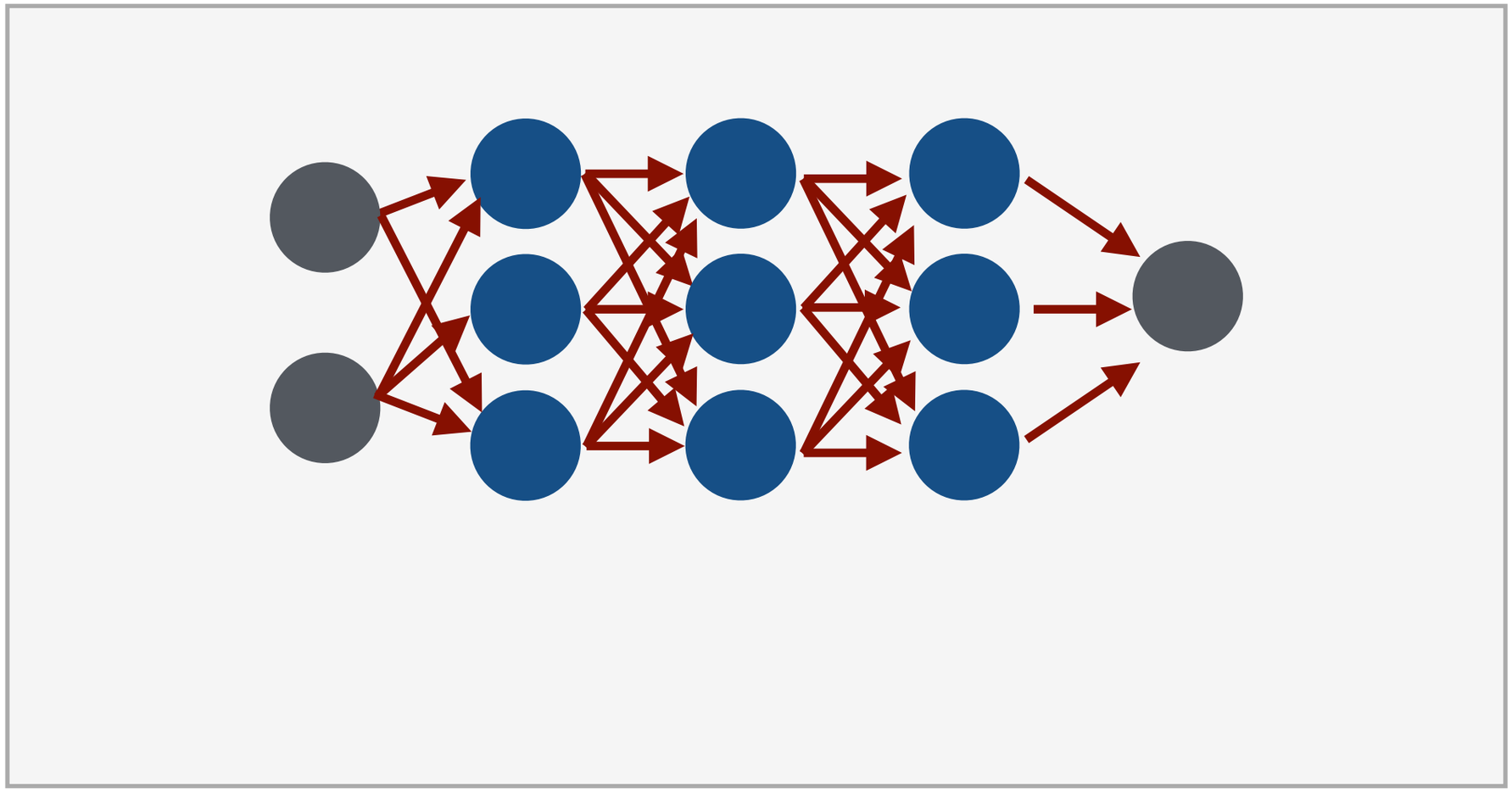
Neural Networks - Idea

- Based on information processing in human brain
- Perceptrons / Axons get activated by electric pulses
- Neurons connect the axons to create a neural network
- Artificial NNs:
  - Axons (connections) $\leftrightarrow$ Matrix multiplication
  - Perceptrons/ Neurons (nodes) $\leftrightarrow$ Activation / response function (Ramp, Sigmoid, Tanh, …)

Neural Networks - Idea

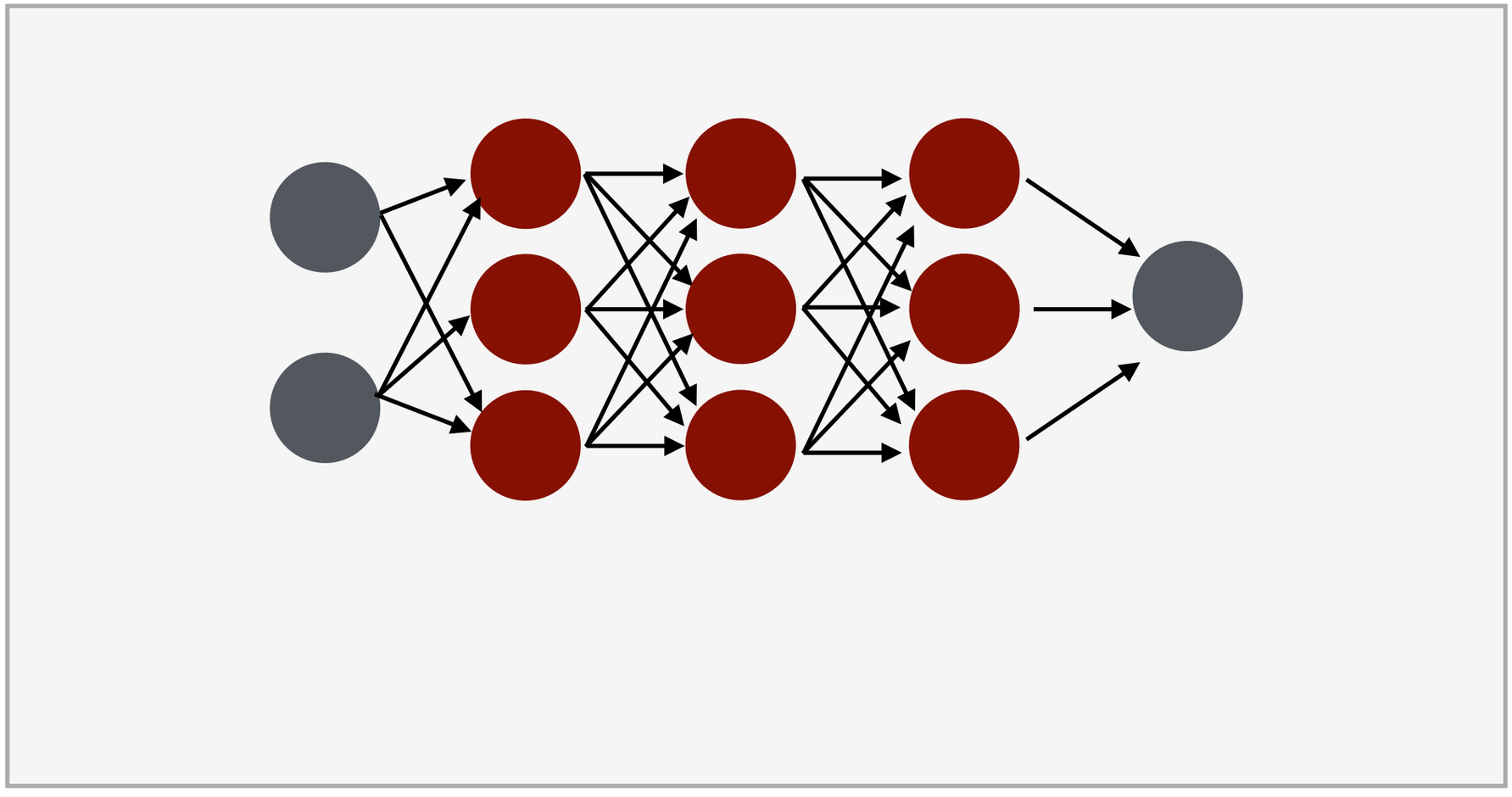


Neural Networks - Idea



- **Connections**: Matrix Multiplication
- Nodes: Apply some activation function f

Neural Networks - Idea



- Connections: Matrix Multiplication
- **Nodes**: Apply some activation function f

NNs - When to use

- NNs are in most ML applications (RL, GANs, …)
- Classical areas of applications for NNs

Universal approximation theorem:

A sufficiently complex ANN can approximate any function to an (in principle) arbitrarily high precision. [Cybenko ’89; Hornik ’91]

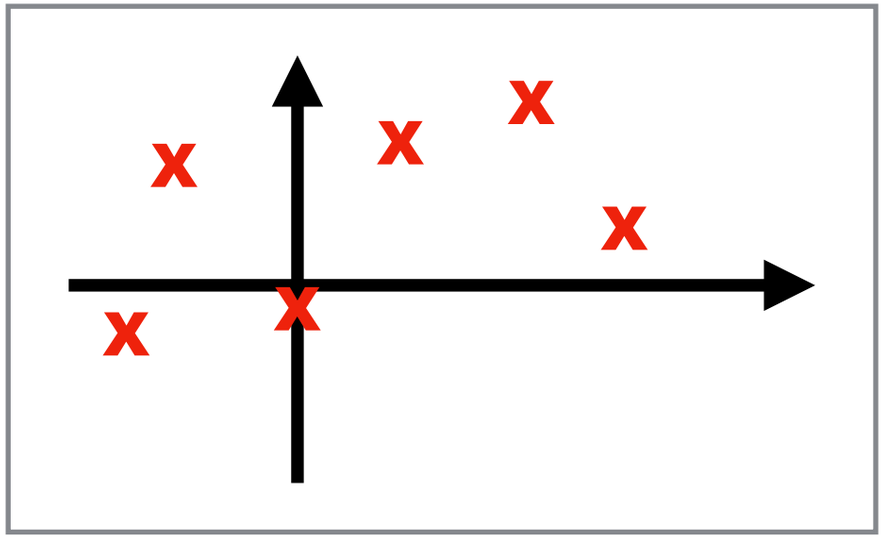
NNs - When to use

- NNs are in most ML applications (RL, GANs, …)
- Classical areas of applications for NNs

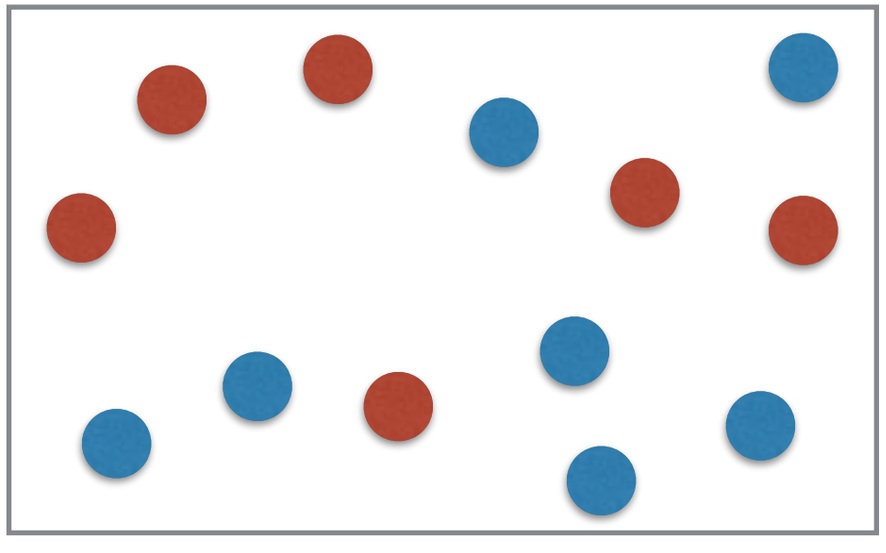
Universal approximation theorem:

A sufficiently complex ANN can approximate any function to an (in principle) arbitrarily high precision. [Cybenko ’89; Hornik ’91]

Prediction



Classification



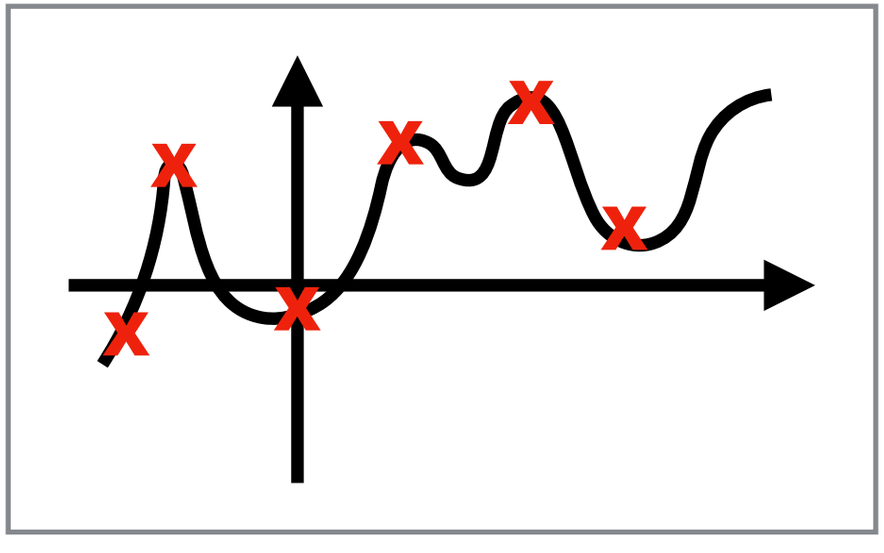
NNs - When to use

- NNs are in most ML applications (RL, GANs, …)
- Classical areas of applications for NNs

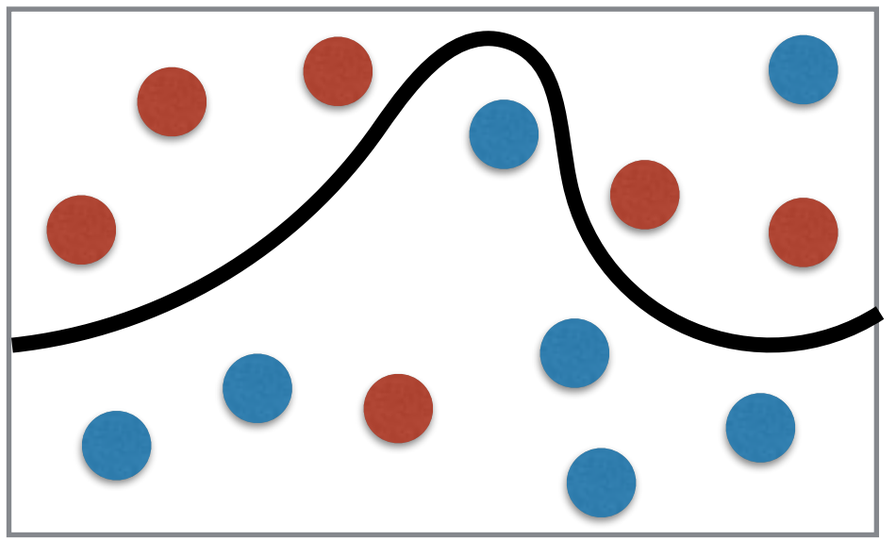
Universal approximation theorem:

A sufficiently complex ANN can approximate any function to an (in principle) arbitrarily high precision. [Cybenko ’89; Hornik ’91]

Prediction



Classification

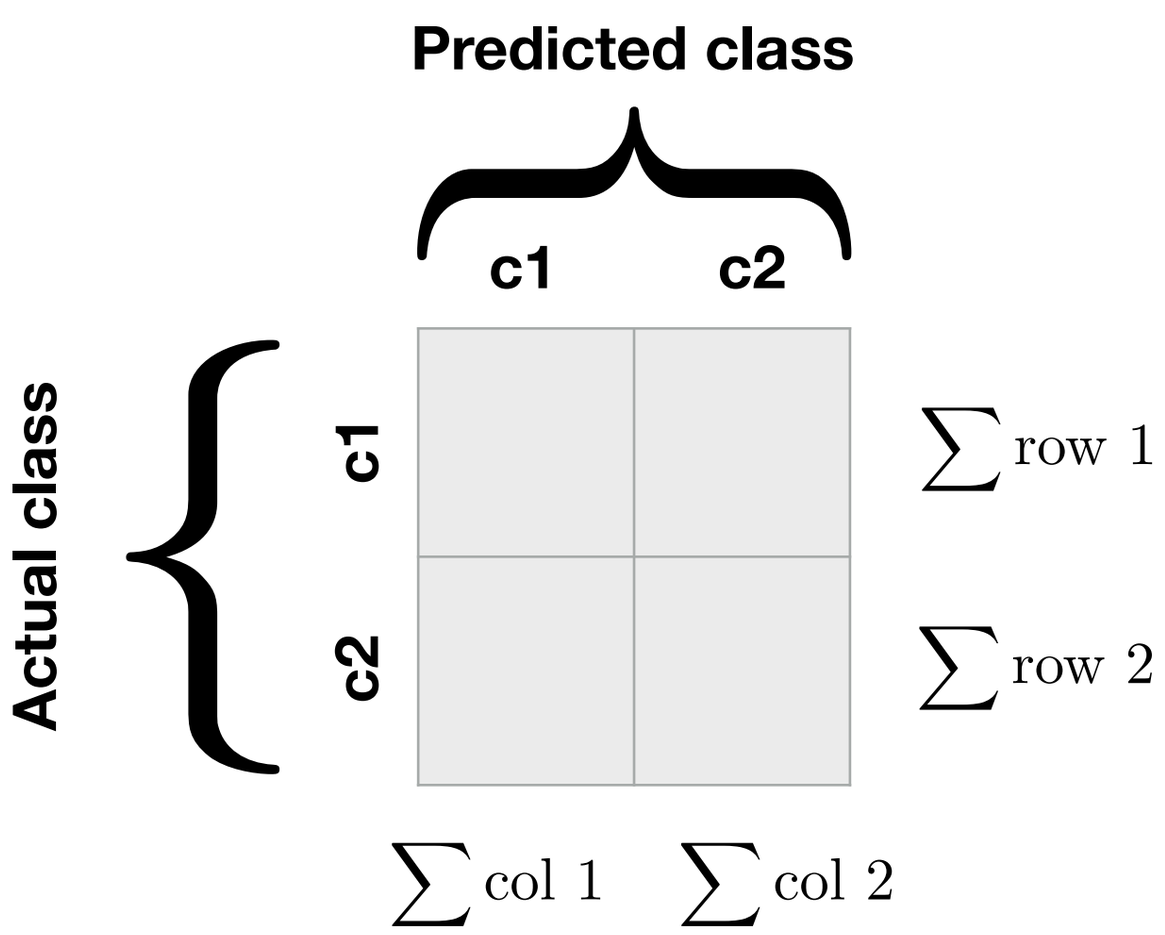


NNs - When to use

- Analyse prediction
  - Divide data into train set + validation set
  - Check performance on validation set
    - ✦ If machine performs much better on training set than on validation set $\Rightarrow$ overtraining
    - ✦ If machine performs equally bad on both $\Rightarrow$ NN not powerful enough
    - ✦ If machine performs equally well on both $\Rightarrow$ NN well designed

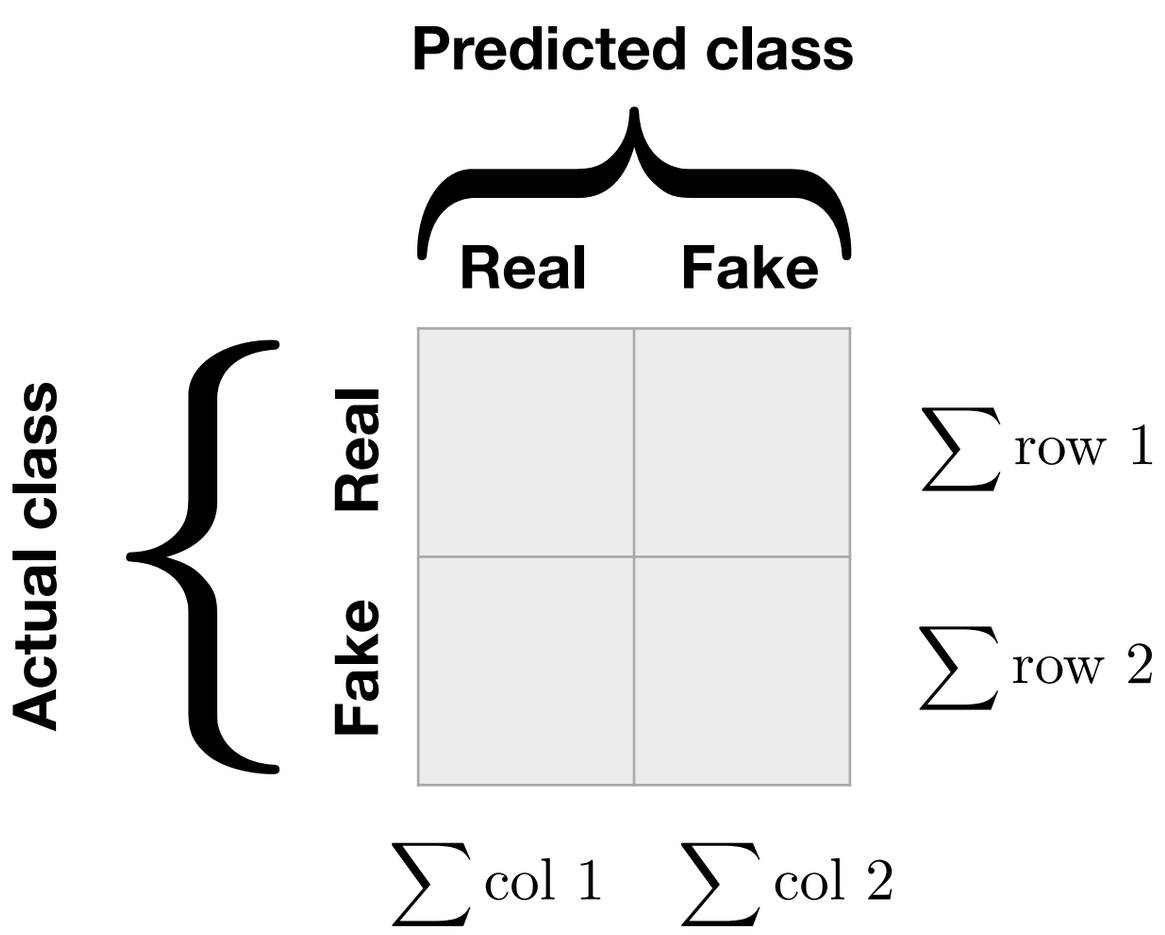
NNs - When to use

- Analyse classification: Confusion matrix, Matthews correlation coefficient



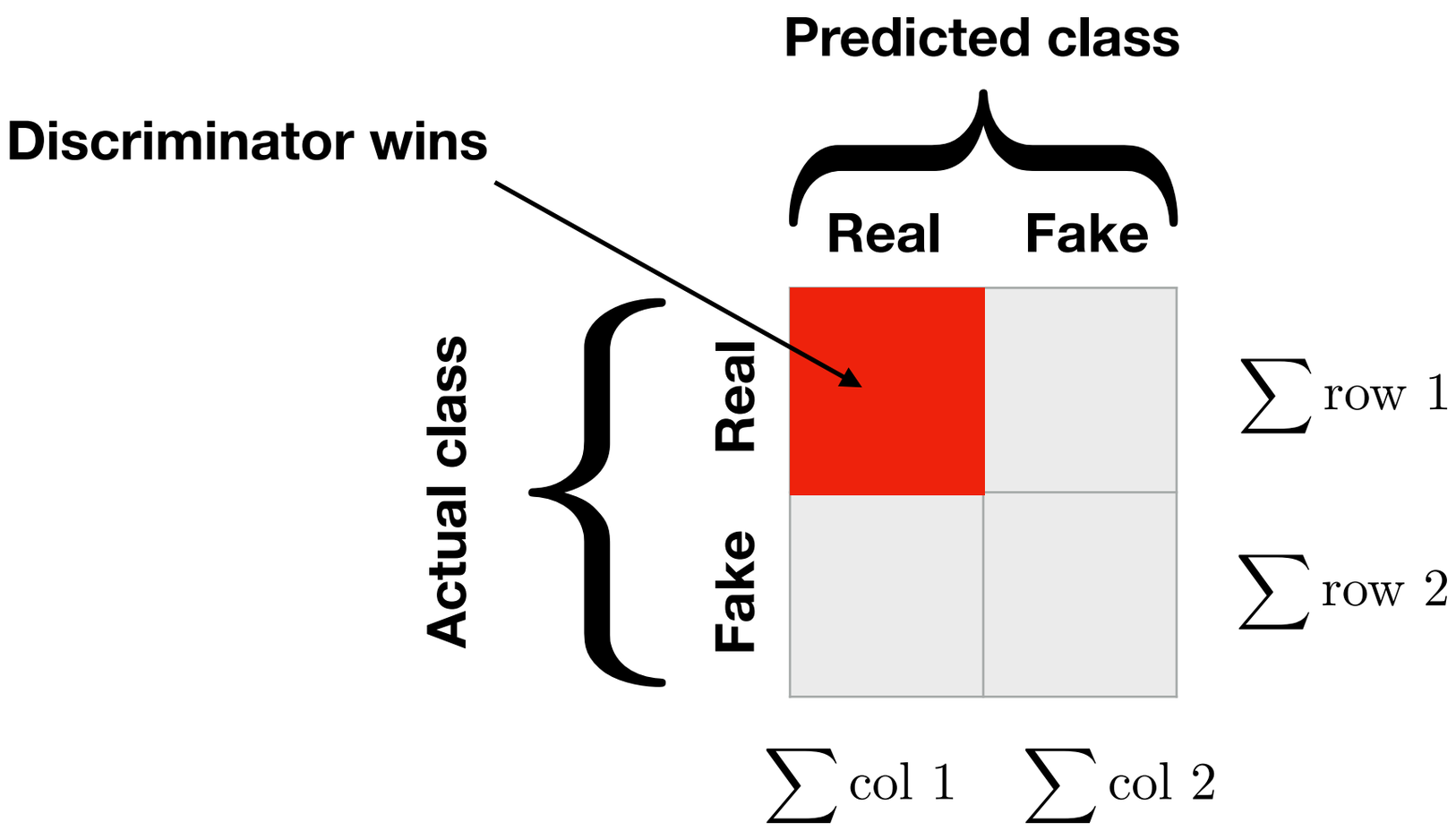
NNs - When to use

- Analyse classification: Confusion matrix, Matthews correlation coefficient
- Example: Discriminator



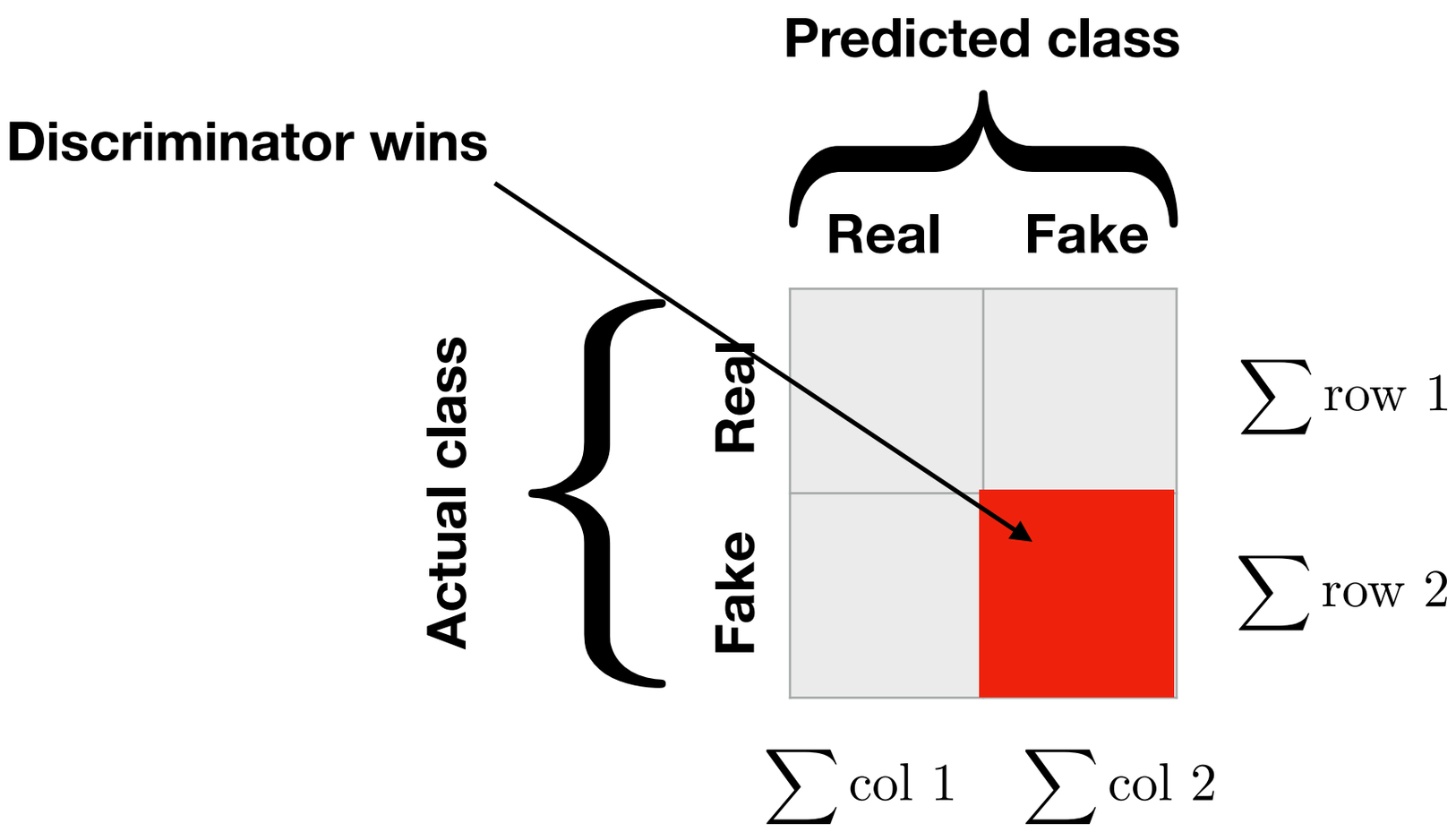
NNs - When to use

- Analyse classification: Confusion matrix, Matthews correlation coefficient
- Example: Discriminator



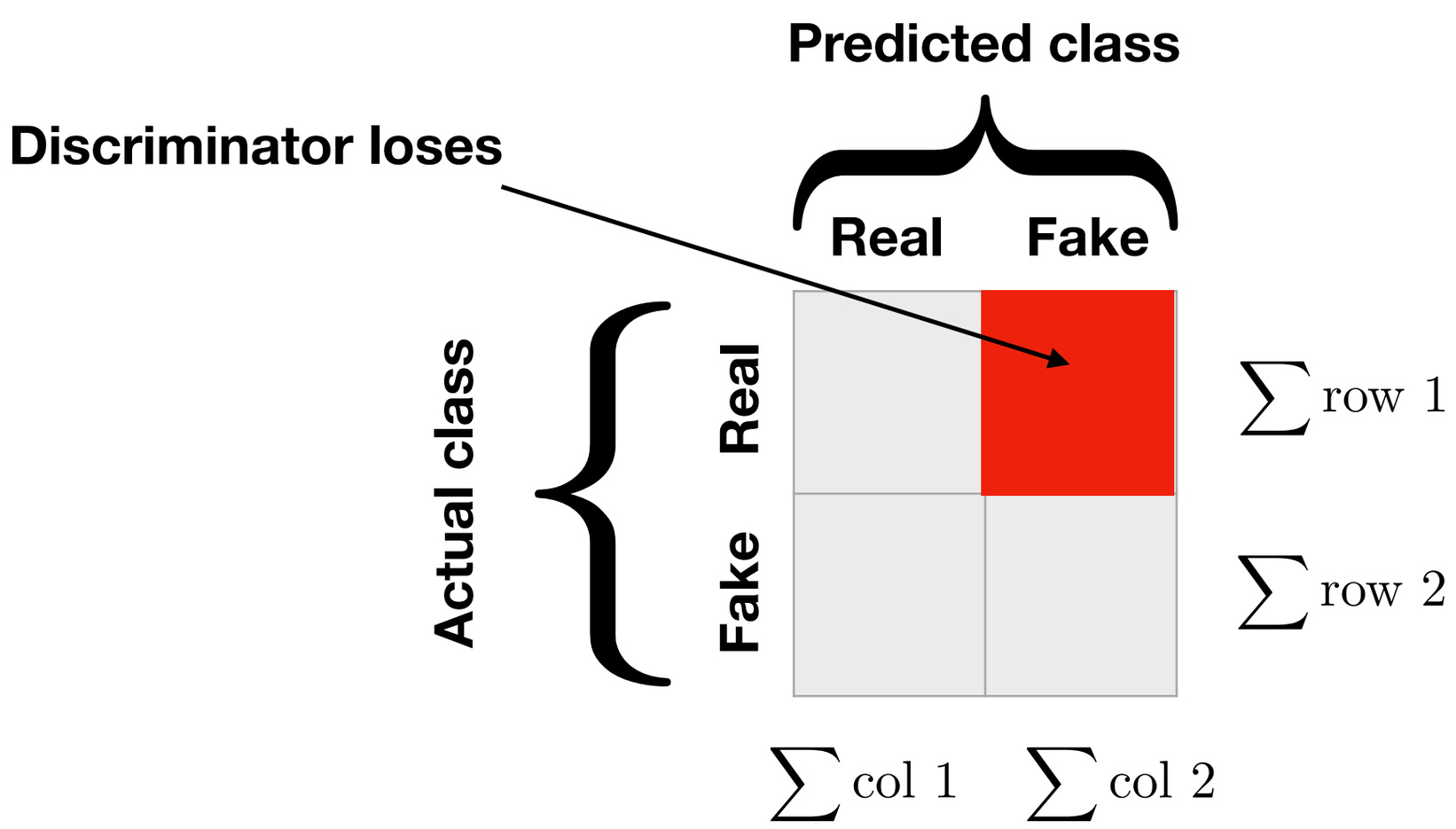
NNs - When to use

- Analyse classification: Confusion matrix, Matthews correlation coefficient
- Example: Discriminator



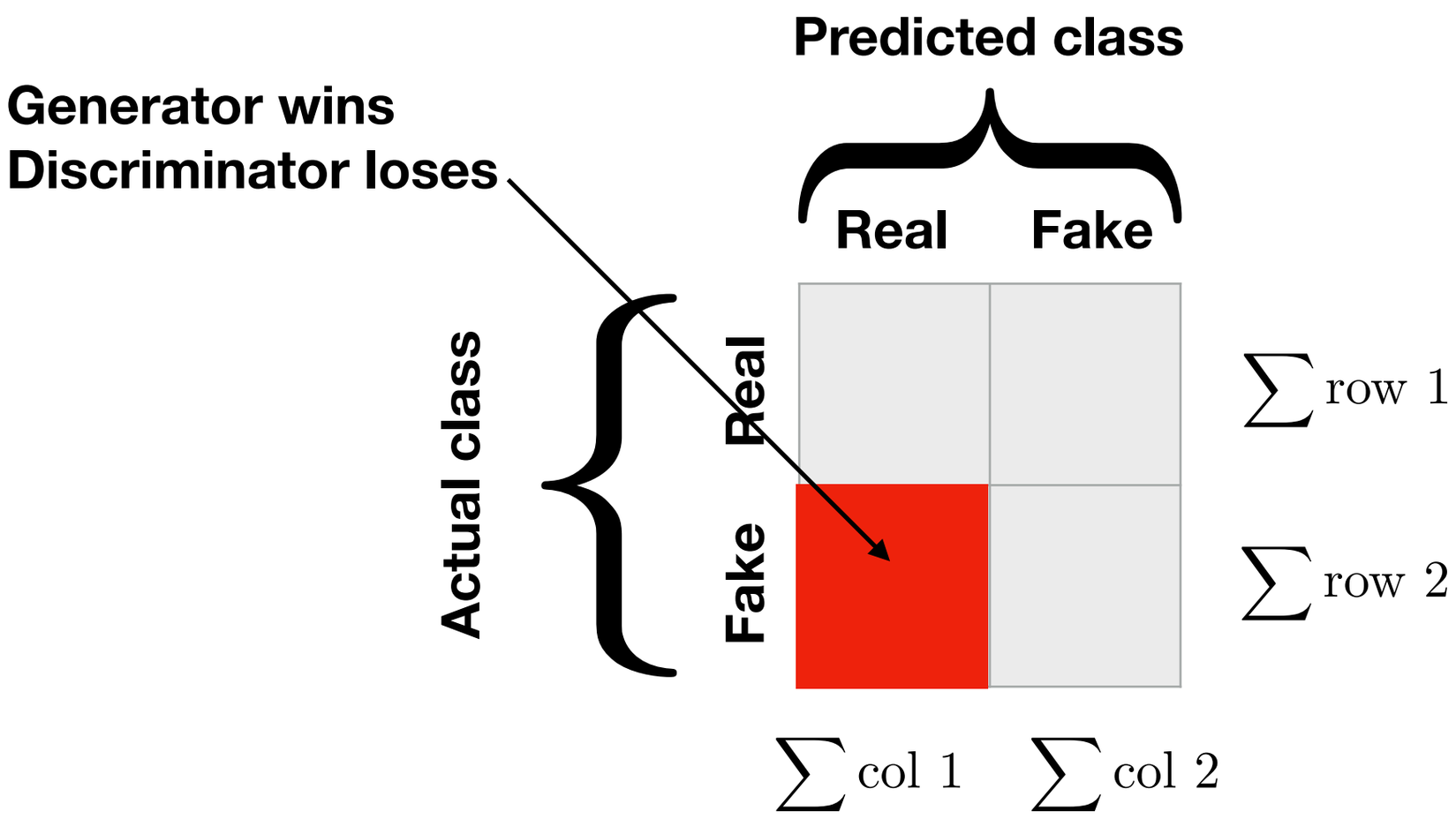
NNs - When to use

- Analyse classification: Confusion matrix, Matthews correlation coefficient
- Example: Discriminator



NNs - When to use

- Analyse classification: Confusion matrix, Matthews correlation coefficient
- Example: Discriminator



Intelligible AI & Conjecture Generation

Industry: e.g. [Weld, Bansal, 1803.04263]
Physics: [Carifio, Halverson, Krioukov, Nelson] [Halverson, Long, Ruehle, Tian to appear]

Intelligible AI: Idea

- **Algorithms learned from our data. Can we learn from them?**

- Via simplifications of an otherwise complex system?

- Pros and cons:
  neural nets are accurate, harder to understand (?).
  simpler algorithms can be less accurate, easier to understand.

- “Understanding” and theorems are different things.

  Strongest form of intelligble AI? [Carifio, Halverson, Krioukov, Nelson]
  Conjecture generation —> theorem.
  Rigorous results.

- **Recommended packages:** scikit-learn, TensorFlow

Example One: A ML-motivated Theorem

- **Full slide set in extra slides, punchline in interest of time.**

- **Use info from ML, think a bit, write down conjecture.**

Theorem: Suppose that with high probability the group G on v_{E_6} is $G \in \{E_6, E_7, E_8\}$ and that E_6 may only arise with $\tilde{m} = (-2, 0, 0)$. Given these assumptions, there are three cases that determine whether or not G is E_6.

a) If $a_{max} \geq 5$, $\tilde{m}$ cannot exist in Δ_g and the group on v_{E_6} is above E_6.

b) Consider $a_{max} = 4$. Let $v_i = a_i v_{E_6} + b_i v_2 + c_i v_3$ be a leaf built above v_{E_6}, and $B = \tilde{m} \cdot v_2$ and $C = \tilde{m} \cdot v_3$. Then G is E_6 if and only if $(B, b_i) > 0$ or $(C, c_i) > 0 \; \forall i$. Depending on the case, G may or may not be E_6.

c) If $a_{max} \leq 3$, $\tilde{m} \in \Delta_g$ and the group is E_6.

- **Key point:** ML-inspired focus on one particular variable, led quickly (< 24 hours) to a theorem once identified.

  "Back and forth" process, could be of broad applicability.

Example 1: Probability and Checks

- **Probability computation:**

$$P(E_6 \text{ on } v_{E_6} \text{ in } T) = \left(1 - \frac{36}{82}\right)^9 \left(1 - \frac{18}{82}\right)^9 \simeq .00059128$$

computed using # appropriate edge trees relative theorem.

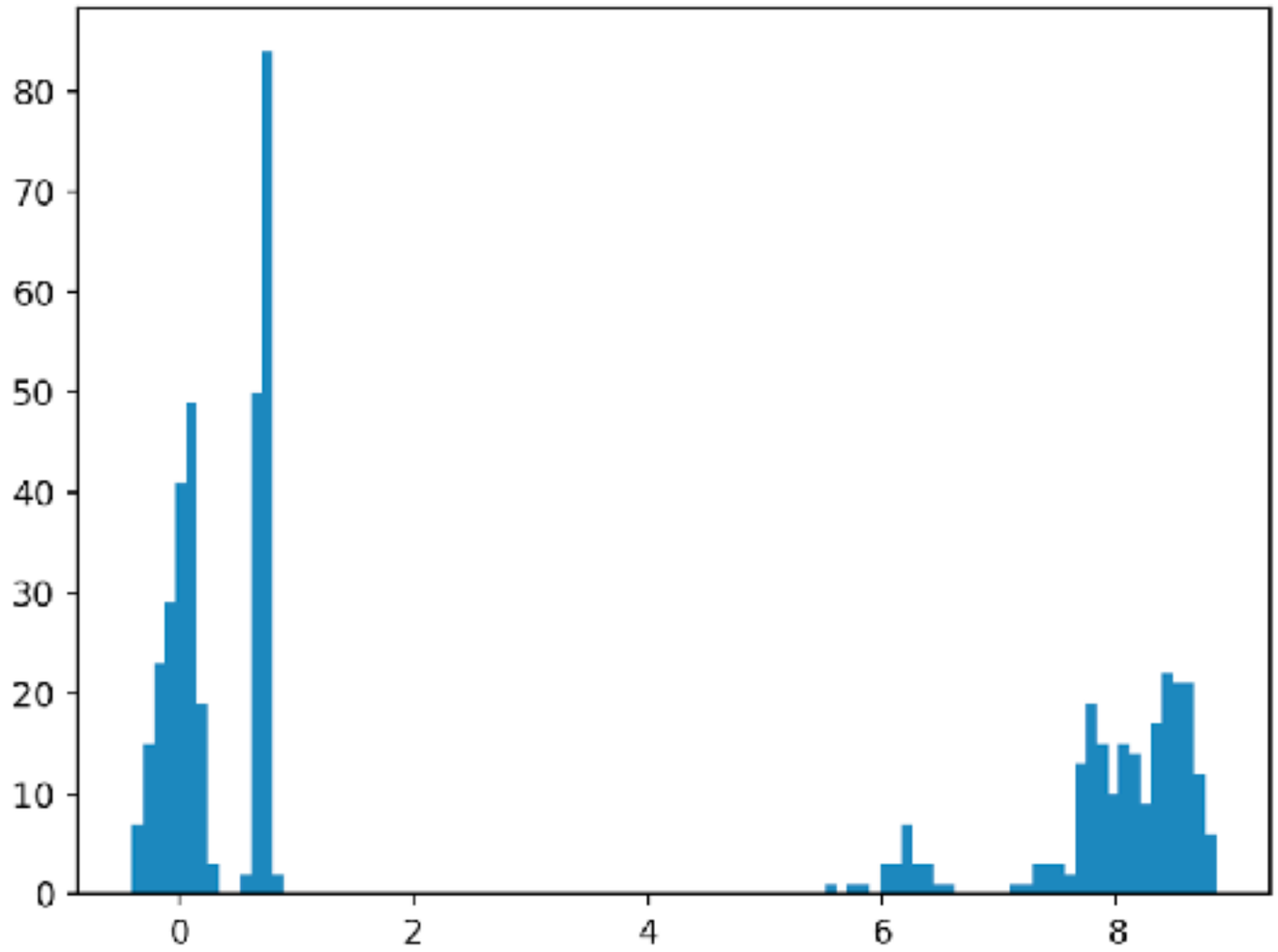
Result:

$$\text{Number of } E_6 \text{ Models on } T = .00059128 \times \frac{1}{3} \times 2.96 \times 10^{755} = 5.83 \times 10^{751}.$$

- **Check:** with 5 batches, 2 million random samples each.

$$\begin{aligned} \text{From Theorem} \quad &: \quad .00059128 \times 2 \times 10^6 = 1182.56 \\ \text{From Random Samples} \quad &: \quad 1183,\ 1181,\ 1194,\ 1125,\ 1195 \end{aligned}$$

Example 2: From Yesterday Histogram of Dangerous Trees



Data:
413264 First No Sen
407487 Last Sen

Accuracy: 98.5%

Similar, but **intuitive push-pull game**. Intercept -7, trees with coefs > 0 pull to right. Only need to add a few trees with coefficient > 5 to be in serious danger of no Sen Limit.

Example 2: From Yesterday Visualizing Dangerous Trees

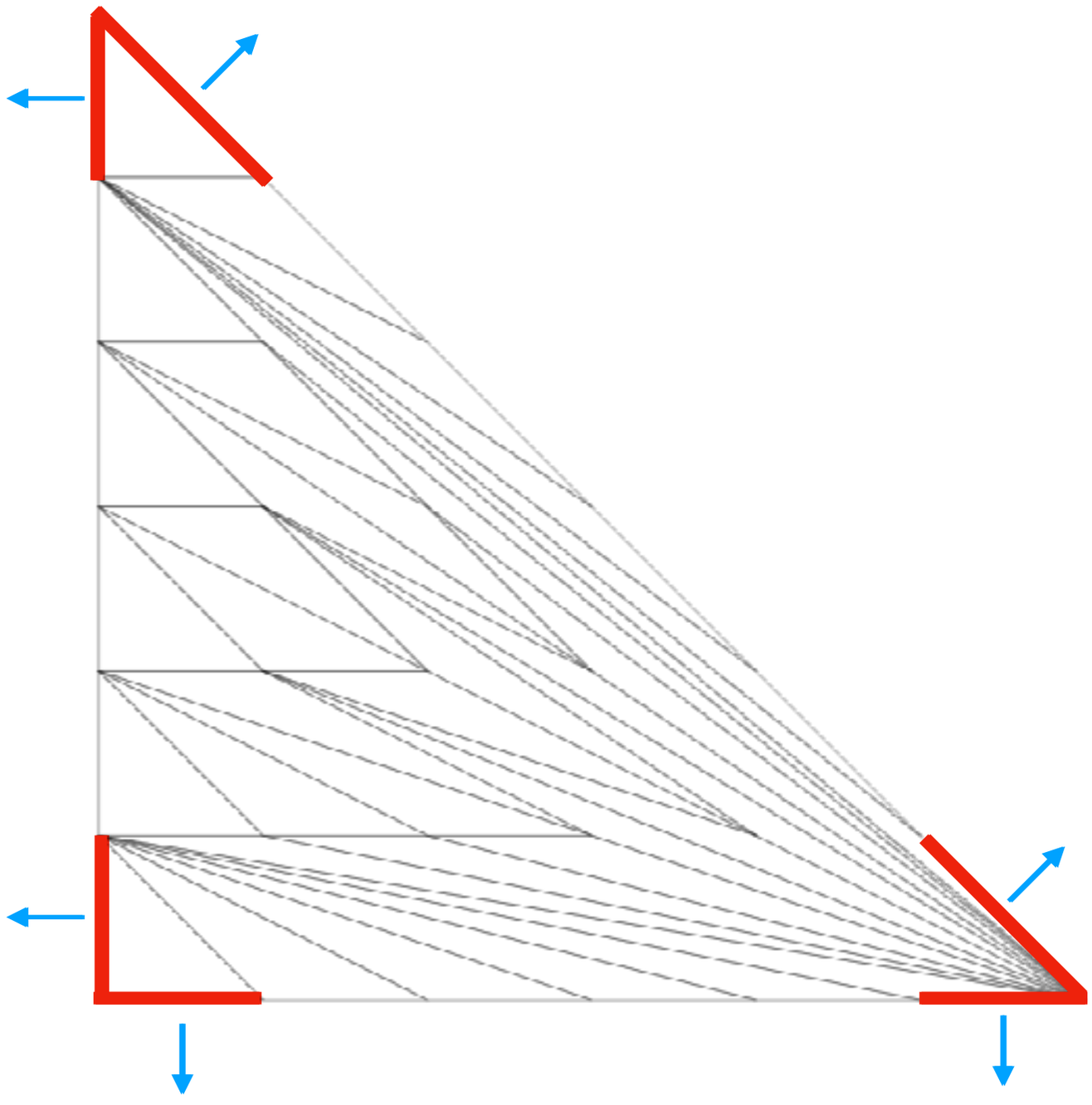
- **Red edges:** any trees there or on triangle wraparounds are in the set of dangerous trees.

  i.e. 12 bad ones associated with red edges.

- **25 other bad:** 18 of which are connected to vertices.

- remaining dangerous trees are of understandable type.

  intuitive, but **ML —> linchpins.**



Pause: This is about taking promising trends in data and trying to understand it better.

Moving from accurate pattern recognition to a **theory for the pattern**, to how the theory might make further predictions about the system.

This is in our bones as theoretical physicists. It is what we do.

Conjecture Generation - When to Use

- Where it might be good: theorems with lots of test cases.

- Alternatively: theorems where patterns in examples are very complicated, hard to figure out
  (machine figures out pattern, we prove it's the right pattern).

Other thoughts or questions?

Ideas about where to apply it specifically, or under what circumstances it might generally be applicable?

Data Generation GANs

So far no literature on the topic in string theory as far as we know, but [Halverson, Long, Ruehle] to appear

Some CS/Math Literature

- Original paper

  Generative Adversarial Networks
  Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu,
  David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
  https://arxiv.org/abs/1406.2661 (statistics)

- More accesible intro (also by Goodfellow)

  NIPS 2016 Tutorial: Generative Adversarial Networks
  Ian Goodfellow
  https://arxiv.org/abs/1701.00160 (CS)

- Example of GANs faking images of celebrities

  Progressive Growing of GANs for Improved Quality, Stability, and Variation
  Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen
  https://arxiv.org/abs/1710.10196 (CS)

- Overview of different GANs

  Are GANs Created Equal? A Large-Scale Study
  Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, Olivier Bousquet
  https://arxiv.org/abs/1711.10337 (statistics)

GANs - Idea

Discriminator vs Generator

- Learns to discriminate real/ fake models

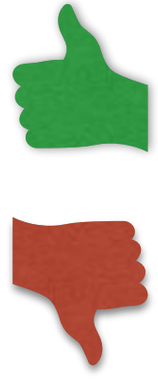
- Learns to fake models that look real to the discriminator

- Discriminator trained to identify “forged” models
- Generator trained to generate realistic “forgeries”
- The two compete against each other:
  - If discriminator catches fraud of generator: Generator loses
  - If discriminator does not catch the fraud: Discriminator loses

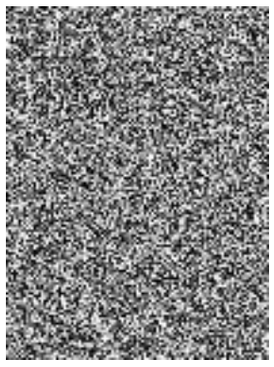
GANs - Idea

Real data

Discriminator



Output



Noise

Generator

Fake data

Influence GAN results

- Each machine tries to minimise its losses individually
- The game ends in a Nash equilibrium: No machine can improve without changing the parameters of the others
- Initial paper: Minmax game, but now much more GAN loss functions have been explored

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x-\mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$

[Lucic et al `17]

Influence GAN results

- Different loss functions lead to different GAN priorities:
  - Minmax: generated models have low probability of being fake
  - Non-saturating: generated models have large probability of being real
  - Wasserstein: generated models minimise the Wasserstein distance
    Distance = amount of data to be changed x distance from original distribution

- These are not the same thing!

GANs - When to use

- Great to generate more models that are “like others”
- But several drawbacks
  - Rather sophisticated (need a good background in NNs)
  - Train for a veeeery long time (need to use CUDA / GPUs)
  - Need parameters chosen consistently
    - No NN should over-power the other
    - Both should advance at the same speed (learning rate)

Search or Explore
(Reinforcement Learning)

Physics to appear: **[Halverson, Nelson, Ruehle] [Halverson, Long, Ruehle, Tian], [Halverson, Nilles, Ruehle, Vaudrevange], [Harries et al]**

Reinforcement Learning: Idea

supervised ML **predicts**, RL (AI) **explores / searches**
most famous examples: (?) AlphaGo & AlphaGo Zero

- an ***agent*** interacts in an ***environment***.
- it perceives a ***state*** from ***state space***.
- its ***policy*** picks and executes an action, given the state.
- agent arrives in new state, receives a ***reward***.
- succesive rewards accumulate into ***return***.
- return may penalize future rewards via ***discount factor***.
- policy optimized to maximize reward, i.e. **agent learns how to act!**

Example: AlphaGo Zero

"Mastering the game of Go without human knowledge."

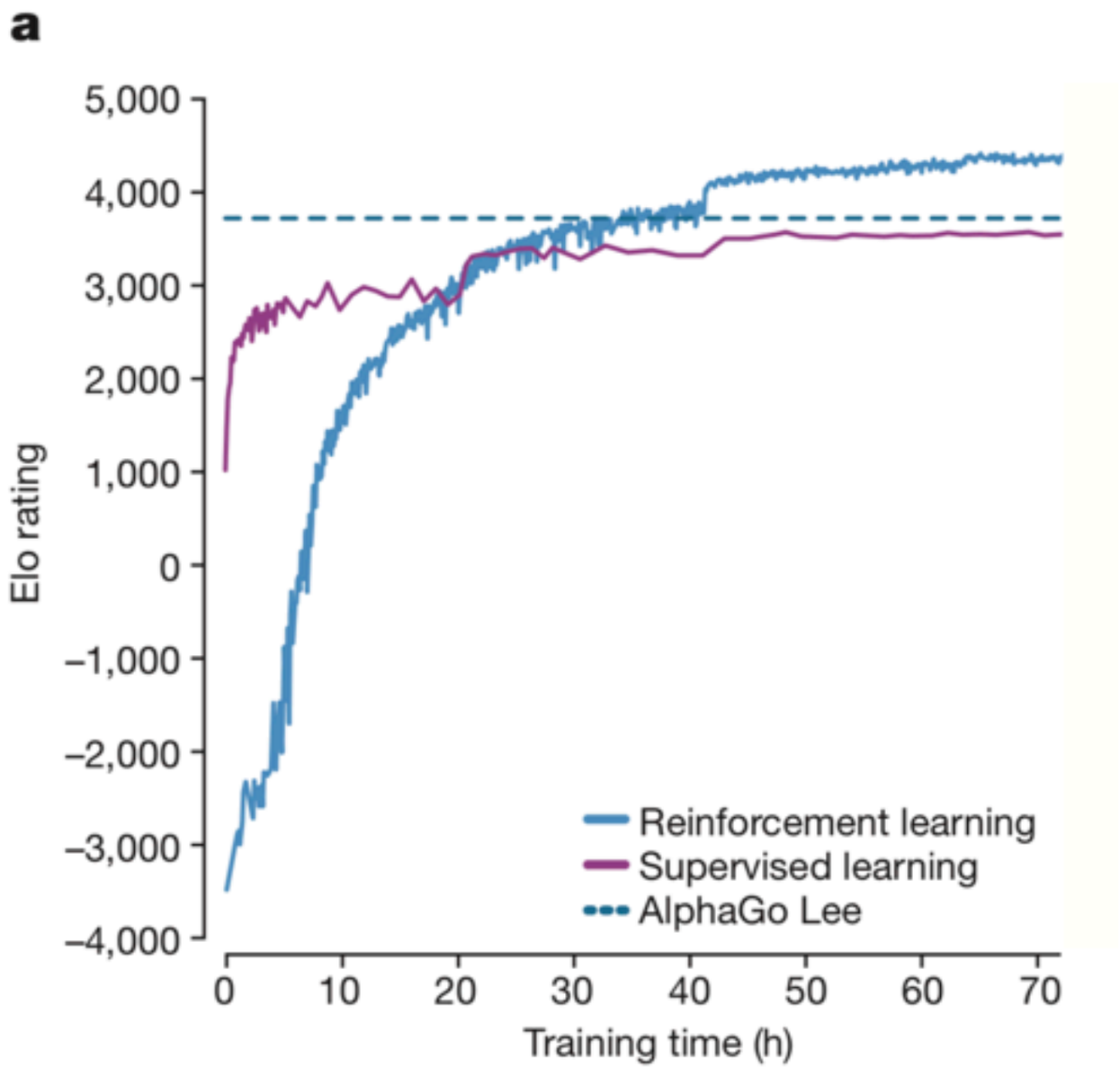
Silver et al. (Google DeepMind), Nature Oct. 2017.

A long-standing goal of artificial intelligence is an algorithm that learns, tabula rasa, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. **Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules.** AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. **Starting tabula rasa, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.**

Fact: Go has 10^{172} states, a "big" number, but for the task of playing excellently, superhuman progress achieved tabula rasa.

AlphaGo Zero: The Money Plot

Silver et al, Nature 2017.



here:

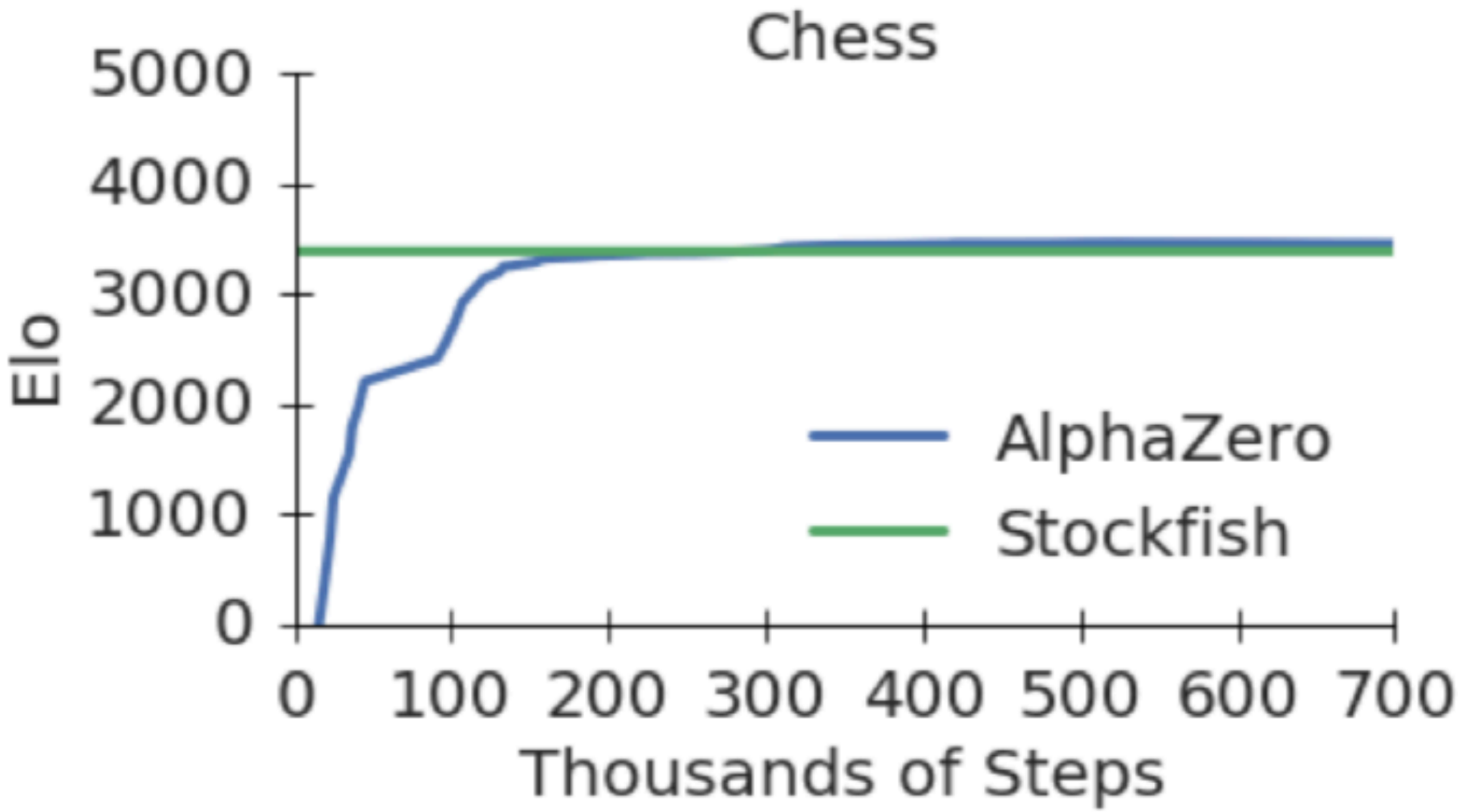
supervised learning = training on human expert games.

AlphaGo Lee = previous version from 2016 that beat world Champ Lee Sedol.

stronger than AlphaGo Lee in **under 48 hrs**, beat **100-0.**

AlphaZero for Chess

similar architecture, arXiv preprint. **Silver, Hubert, Schrittwieser et al**



stronger than Stockfish in **under 4 hrs**, beat thoroughly**.**

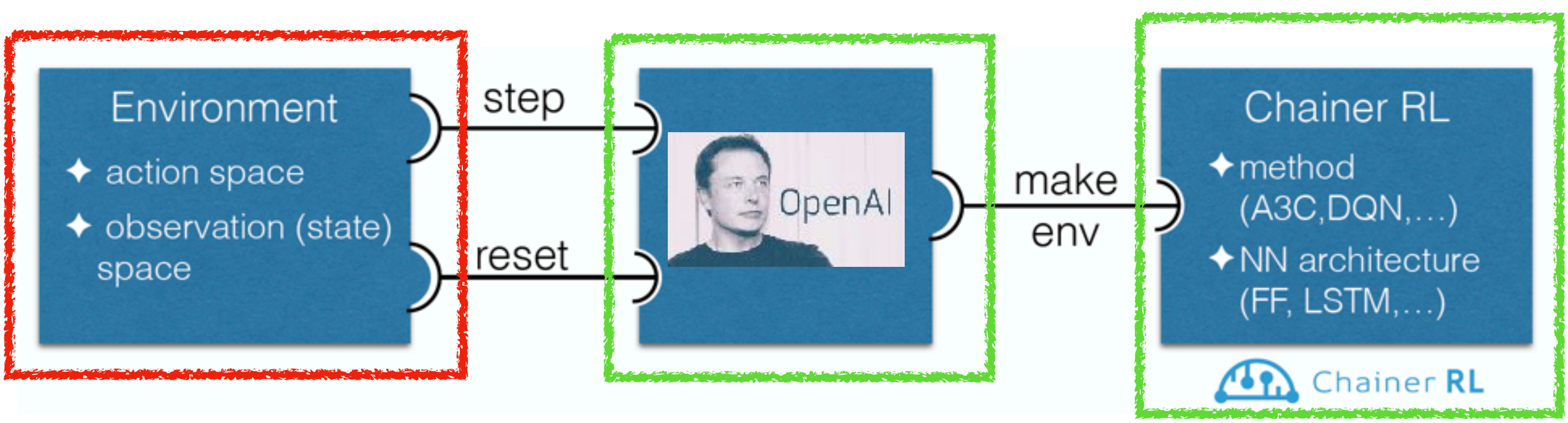
Implementation

model-free RL: want algorithms to work well regardless of environ.

means we can use CS-implemented algs!

three modules:

- Open AI (Musk) defines what an environ is and how to interface.
- ChainerRL provides RL algorithms and NN architecture.
- **Physicists provide:** the environment. two envs so far. **~50 new lines?**



algorithm: asynchronous advantage actor-critic (A3C) [Minh et al 2016]
(parallel CPU, not GPU)

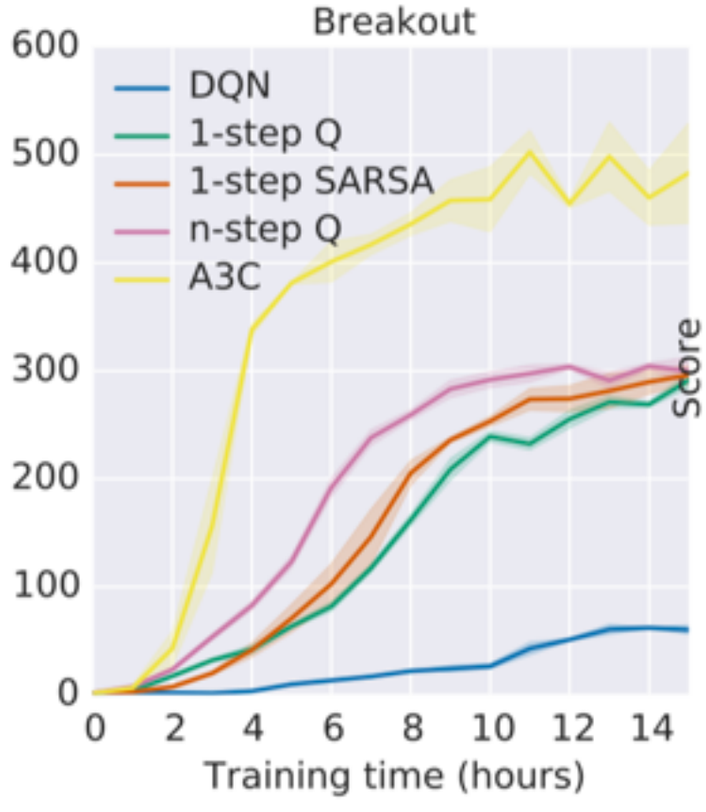
Asynchronous Advantage Actor-Critic (A3C)

[Mnih et al, DeepMind 2016]

"Our parallel reinforcement learning paradigm also offers practical benefits. Whereas previous approaches to deep reinforcement learning rely heavily on specialized hardware such as GPUs (Mnih et al., 2015; Van Hasselt et al., 2015; Schaul et al., 2015) or massively distributed architectures (Nair et al., 2015), ***our experiments run on a single machine with a standard multi-core CPU****. When applied to a variety of Atari 2600 domains, on many games asynchronous reinforcement learning achieves better results, in* ***far less time than previous GPU-based algorithms****, using far less resource than massively distributed approached" - Mnih et al,* Asynchronous Methods for Deep RL

- **Actor-Critic Methods:** NN for determining both policy (actor) and value (critic).
- **Asynchronous:** many worker bees explore, report back to king (critic) and queen (actor) bee.

  i.e. use **communal knowledge**.
- > some 2016 GPU algs. Simple to run. Learns strat.



Tell it how to step

```
def _step(self,action):
    done = False
    self.numsteps += 1

    self.previousstate = self.state[0:] # save curstate to previous before changing

    ### Add tree according to action size
    numnew = 0
    if action >= len(self.edgemoves):
        numnew = self.add_tree(self.facemoves[action-len(self.edgemoves)],action)
    else:
        numnew = self.add_tree(self.edgemoves[action],action)

    # chop and compute new sections
    self.fpts, self.gpts = self.chopfgpts(self.pts[-numnew:],self.fpts,self.gpts)
    self.fsecs = [[4+self.dot(v,fpt) for v in self.pts] for fpt in self.fpts]
    self.gsecs = [[6+self.dot(v,gpt) for v in self.pts] for gpt in self.gpts]

    my_reward = self.reward(numnew)
    if my_reward == self.out_of_bounds_reward:
        done = True
        prev, cur = self.list_to_binary(self.previousstate), self.list_to_binary(self.state[0:])
        alg = self.find_algebra(self.fsecs,self.gsecs)
        print alg
        print self.state
        print self.polypoints
        if prev not in self.lastsen:
            self.lastsen.add(prev)
            # modify output() to two functions, one for lastsenout, another for firstnosenout. then call one here
            self.output_ls(prev)

        if cur not in self.firstnosen:
            self.firstnosen.add(cur)
            # call other here
            self.output_fns(cur)

    return np.array(self.state), my_reward, done, {}
```

RL: Possible Use Cases

- IIa orbifolds for consistency and particle physics. (Ruehle, Nov talk)
- Studying the IIb lamppost. (Halverson's talk)
- Heterotic orbifolds. (Vaudrevange's talk)
- Heterotic free fermions. (Harries' talk)

We have robots that can learn to explore spaces intelligently.
What should we do?
What questions do you have?

Data Structure Persistent Homology

[Cirafici`15 - 1512.01170 (hep-th)], [Cole, Shiu`17 - 1712.08159]

Some Literature

- Physics paper w/ applications to Kreuzer-Skarke, CICYs, Landau-Ginzburg, flux vacua

  Persistent Homology and String Vacua
  Michele Cirafici
  https://arxiv.org/abs/1512.01170 (hep-th)

- Math paper that explains basics and compares libraries

  A roadmap for the computation of persistent homology
  Nina Otter, Mason A. Porter, Ulrike Tillmann, Peter Grindrod, Heather A. Harrington
  https://arxiv.org/abs/1506.08903 (Math/Algebraic Topology)

Some Libraries [1506.08903]

- Javaplex (used in physics paper) [http://appliedtopology.github.io/javaplex]
  - \+ Written in JAVA ⇒ platform independent
  - \+ Easy to use
  - \+ Implements a variety of complexes / algorithms
  - \- Slower than others
  - \- Had a bug (as of 2015) that caused some wrong results

- Perseus [http://www.sas.upenn.edu/~vnanda/perseus]
  - \+ C++ (but versions for all platforms available)
  - \+ Easy to use
  - \- Slower than others

Some Libraries [1506.08903]

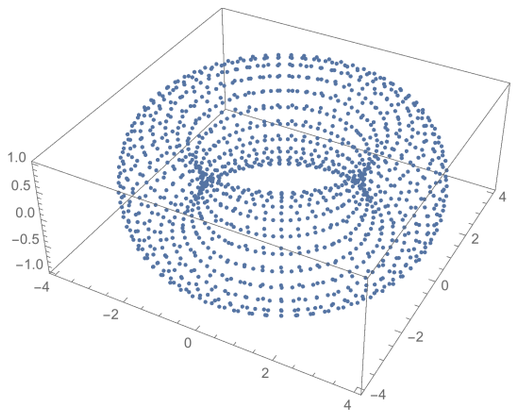
- GUHDI [https://project.inria.fr/gudhi/software/]
  - + Much faster than Javaplex / Perseus
  - + Needs less memory than DIPHA
  - - Slightly slower than DIPHA

- DIPHA [https://code.google.com/p/dipha]
  - + Much faster than Javaplex / Perseus
  - + Slightly faster than DIPHA
  - - Needs more memory than GUHDI

- Ripser [https://github.com/Ripser/ripser]
  - + Fastest
  - + Newest code
  - - Least tested (since newest)

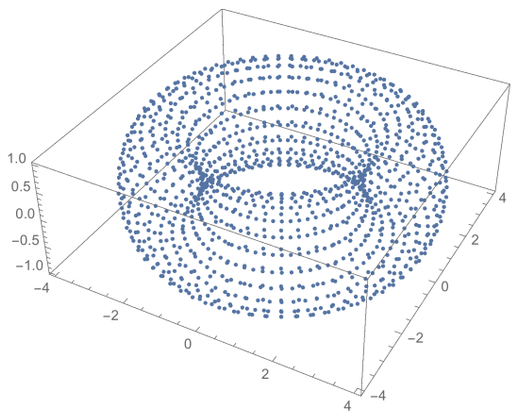
Persistent Homology - Idea

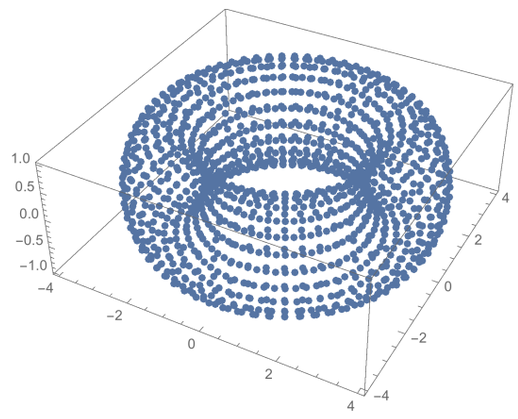
- Way to assign homology to discrete data / points



Persistent Homology - Idea

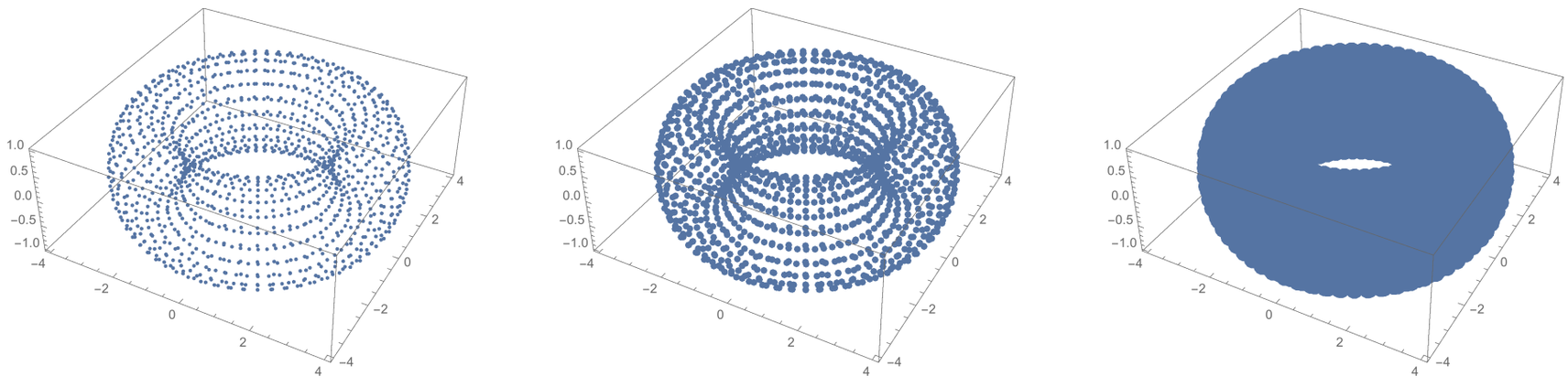
- Way to assign homology to discrete data / points
- Idea:
  - Replace data points by balls (several disconnected components)





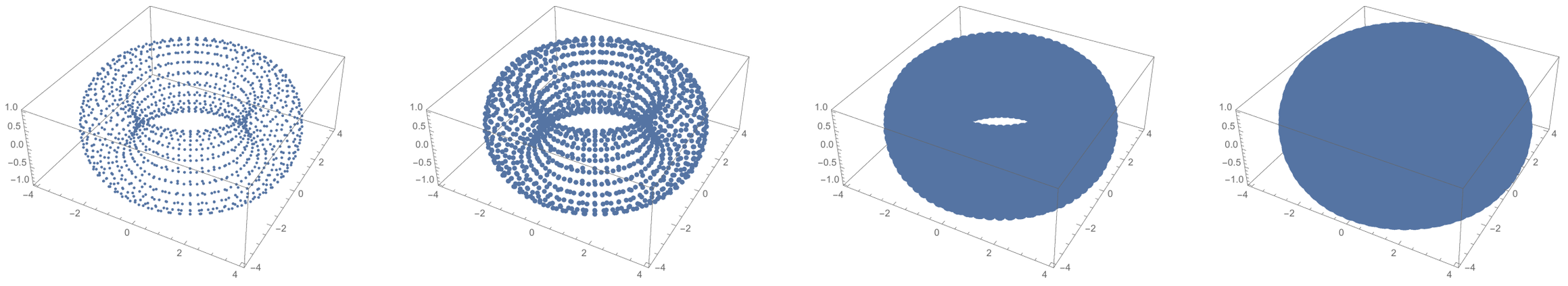
Persistent Homology - Idea

- Way to assign homology to discrete data / points
- Idea:
  - Replace data points by balls (several disconnected components)
  - As radius of points grow, components connect / form cycles / …



Persistent Homology - Idea

- Way to assign homology to discrete data / points
- Idea:
  - Replace data points by balls (several disconnected components)
  - As radius of points grow, components connect / form cycles / …
  - When radius grows further, cycles can disappear again



Persistent Homology - Idea

- Way to assign homology to discrete data / points
- Idea:
  - Replace data points by balls (several disconnected components)
  - As radius of points grow, components connect / form cycles / …
  - When radius grows further, cycles can disappear again
- For each k-cycle determine how long it exists as a function of the sphere radius $\Rightarrow$ barcode (Betti number vs radius)
- The longer a cycle persists the more likely it is to be a true feature

Persistent Homology - Idea

- Covering of metric space X with balls of radius ε
  $\Rightarrow$ Cech complex $\check{\mathrm{C}}\mathrm{ech}_\varepsilon(X)$
- Checking all overlaps/intersections too expensive $\Rightarrow$ approx.
  - Vietoris-Rips complex: Approximates Chech complex but requires only pairwise comparisons
  - Inclusion of spaces $X_{\varepsilon_i} \subset X_{\varepsilon_{i+1}}$ $\Rightarrow$ Filtration of simplices
    $\mathrm{VR}_{\varepsilon_i}(X) \subset \mathrm{VR}_{\varepsilon_{i+1}}(X)$ $\Rightarrow$ Inclusion of Homologies
    $H_i(\mathrm{VR}_{\varepsilon_i}(X)) \subset H_i(\mathrm{VR}_{\varepsilon_{i+1}}(X))$ $\Rightarrow$ Barcode
  - Delauny complex: Limits #(complexes) in high dim
    - Alpha-complex: Subcomplex of Delauny whose dim is at most that of X
    - Witness-complex: Subcomplex of Delauny constructed from a subset of all points in X

PH - When to use

- Analyze structure of discrete data independent of choice of metric / coordinates
- Problems:
  - Slow to compute higher Betti numbers $b^i \ \ (i > 2)$
  - Homologies (especially higher ones) hard to interpret if discrete data does not represent spatial information itself
  - Does PH encode information that cannot be gained from e.g. histograms, clustering, … ?

Connections
(Network Science)

e.g. **[Taylor, Wang] [Carifio, Cunningham, Halverson, Krioukov, Long, Nelson]**

Networks: Idea

- Graph theory applied to real world systems, where maybe the nodes and / or edges are decorated with extra data.

- **Pro:** abstract and “minimal”, can apply it to basically any objects that have relationships between them.

  Con: abstract and “minimal”, can apply it to basically any objects that have relationships between them.

- E.g. dynamics of epidemic spread and quarantine.
  [Vespignani et al]

- Tons of useful network / graph theory packages out there, e.g NetworkX, just Google and compare benchmarks.

- Many techniques and applications, but maybe not too many entries yet into the string literature of the more advanced techniques?
  (Our paper only needed very simple techniques, given the complicated graph).

Landscape as a Network

- **Semi-precise version:** (dream scenario if we knew everything)
  - Vacua are nodes, labelled with all assoc. phys. data.
  - Two directed edges between each node pair,
  - Label w/ tunneling rates each direction.
  - Determine right cosmology, compute vacuum selection.

- Coarse-grain 1: Topological Transitions (see Long's talk)
  - geometries are nodes, labelled by topological data.
  - edges if transition of chose type (e.g. blowup) exist between.
  - determine toy cosmology, compute vacuum selection.
  - consider robustness or not to deviations from toy model.

- Coarse-grain 2: Flux transitions
  - flux configurations are nodes
  - edges if two nodes differ by single flux unit
  - could be democratic, or label according to cohomology basis

Other natural networks in string theory?

BACKUP SLIDES

An E_6 Puzzle

- **Gauge group result:** dominated by $G_i \in \{E_8, F_4, G_2, A_1\}$

  (interesting: groups with only self-conjugate reps!)

- **Something SM-useful?** E6? SU(3)?

  - Simple conditions / probabilities for then not known. **JH, Long, Sung**
  - In random samples, prob ~ 1/1000.
  - When E6 arises in RS, on a distinguished vertex: (1,-1,-1).

- **Machine Learning:** **Carifio, Halverson, Krioukov, Nelson**

  Q: Can we train an ML model to accurately predict yes or no for E6 on (1,-1,-1)?

  Q: If so, can we learn how it makes its decision?

  in our paper: called **conjecture generation**.
  as a CS buzzword: **intelligible AI**.

Point: by using machine learning to generate conjectures, we may be able to take its numerical / empirical results and turn it into rigorous results.

Training the Model

- **Supervised machine learning:** given a large number of (input,output) pairs, **learn to predict** output given input, and then **test on unseen data**, see how well the model does.

- **Training data:**

  Input: (max height above v, # of such rays) for all v in polytope.
  Output: E6 on (1,-1,-1) or not.

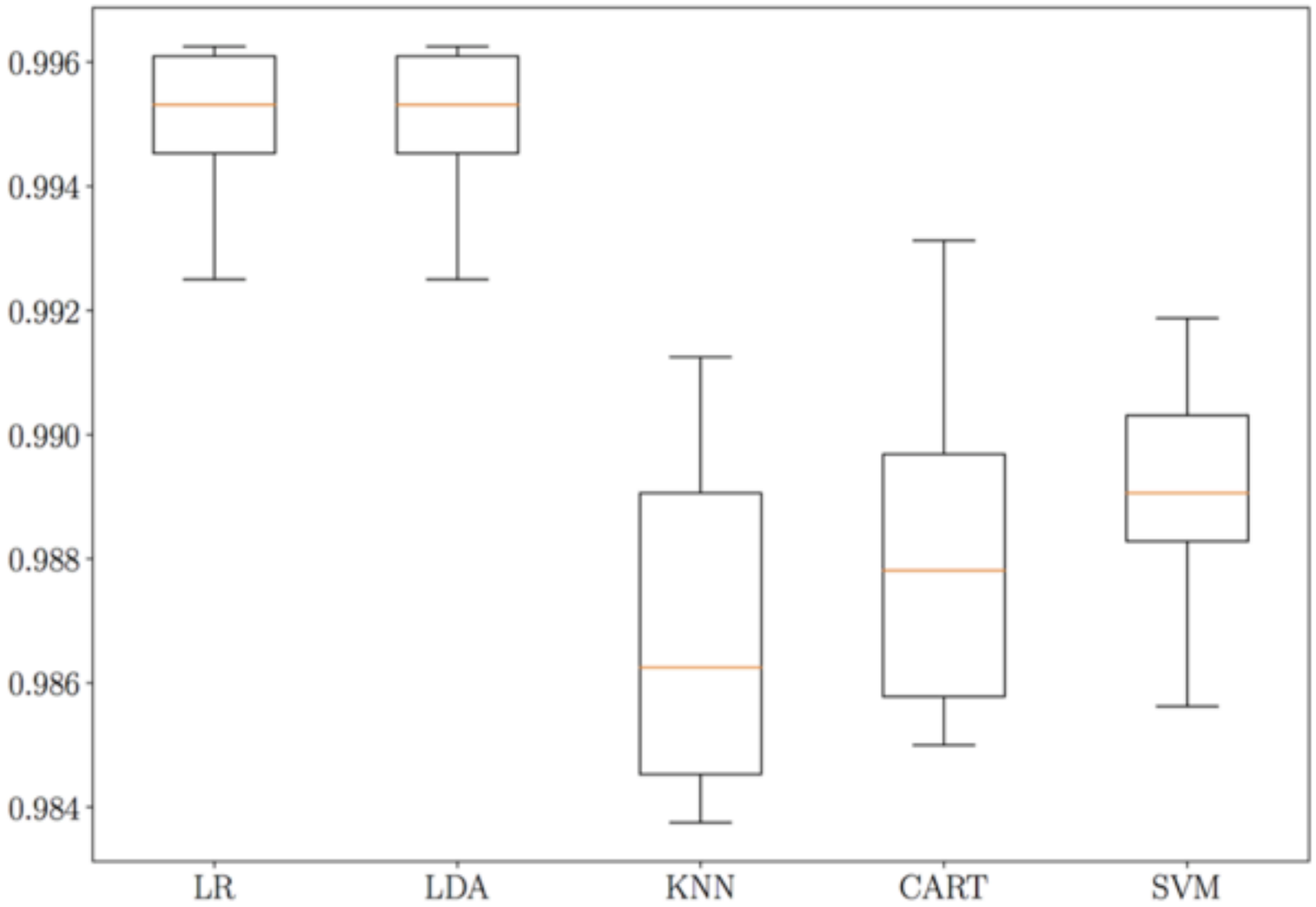
$$S_{a,v_1} := \{v \in V | v = av_1 + bv_2 + cv_3, \;\; a,b,c \geq 0\}$$

$$(a_{max}, |S_{a_{max},v}|) \quad \forall v \in \Delta_1^\circ \qquad \xrightarrow{A} \qquad E_6 \text{ on } v_{E_6} \text{ or not}$$

- **sklearn:** a very nice free Python package.

- **Training sample:** 10000 random with no E6, 10000 random with E6.

Evaluating the Model on Unseen Data



	LR	LDA	KNN	CART	SVM
50/50 Validation Set	.994	.994	.982	.987	.989
Unenriched Set	.988	.988	.981	.988	.983.

- **Displayed:** whisker plots of % accuracy with 10-fold cross validation.
- **Gold bar:** mean % accuracy.
- **Factor analysis:** only two of the variables really matter:

$$(a_{max}, |S_{a_{max}, v_{E6}}|)$$

Conjecture Generation

- **Organizing principle?** See what it gets right and wrong! (using the model trained with logistic regression.)

- **Observation:**
  amax = 5: always no
  amax = 4: usually no.

- **Initial Conjecture:**

Conjecture: If $a_{max} = 5$ for v_{E6}, then v_{E6} does not carry E_6. If $a_{max} = 4$ for v_{E6} it may or may not carry E_6, though it is more likely that it does.

a_{max}	$\lvert S_{a_{max},v_{E6}} \rvert$	Pred. for E_6 on v_{E_6}	Hyperplane Distance
4	5	No	0.88
4	6	No	0.29
4	7	Yes	−0.31
4	8	Yes	−0.90
4	9	Yes	−1.50
4	10	Yes	−2.09
4	11	Yes	−2.69
4	12	Yes	−3.28
4	13	Yes	−3.88
4	14	Yes	−4.47
4	15	Yes	−5.07
4	16	Yes	−5.67
4	17	Yes	−6.26
4	18	Yes	−6.85
4	19	Yes	−7.45
4	20	Yes	−8.04
4	21	Yes	−8.64
4	22	Yes	−9.23
4	23	Yes	−9.83
4	24	Yes	−10.42
5	1	No	7.34
5	2	No	6.75
5	3	No	6.15
5	4	No	5.56
5	5	No	4.96
5	6	No	4.37
5	7	No	3.78
5	8	No	3.18
5	9	No	2.59
5	10	No	1.99
5	11	No	1.40
5	12	No	0.80

Conjecture Refinement and Theorem

- **Use info from ML, think a bit, write down conjecture.**

Theorem: Suppose that with high probability the group G on v_{E_6} is $G \in \{E_6, E_7, E_8\}$ and that E_6 may only arise with $\tilde{m} = (-2, 0, 0)$. Given these assumptions, there are three cases that determine whether or not G is E_6.

a) If $a_{max} \geq 5$, $\tilde{m}$ cannot exist in Δ_g and the group on v_{E_6} is above E_6.

b) Consider $a_{max} = 4$. Let $v_i = a_i v_{E_6} + b_i v_2 + c_i v_3$ be a leaf built above v_{E_6}, and $B = \tilde{m} \cdot v_2$ and $C = \tilde{m} \cdot v_3$. Then G is E_6 if and only if $(B, b_i) > 0$ or $(C, c_i) > 0 \ \forall i$. Depending on the case, G may or may not be E_6.

c) If $a_{max} \leq 3$, $\tilde{m} \in \Delta_g$ and the group is E_6.

- **Key point:** ML-inspired focus on one particular variable, led quickly (< 24 hours) to a theorem once identified.

  “Back and forth” process, could be of broad applicability.

Probability and Checks

- **Probability computation:**

$$P(E_6 \text{ on } v_{E_6} \text{ in } T) = \left(1 - \frac{36}{82}\right)^9 \left(1 - \frac{18}{82}\right)^9 \simeq .00059128$$

computed using # appropriate edge trees relative theorem.

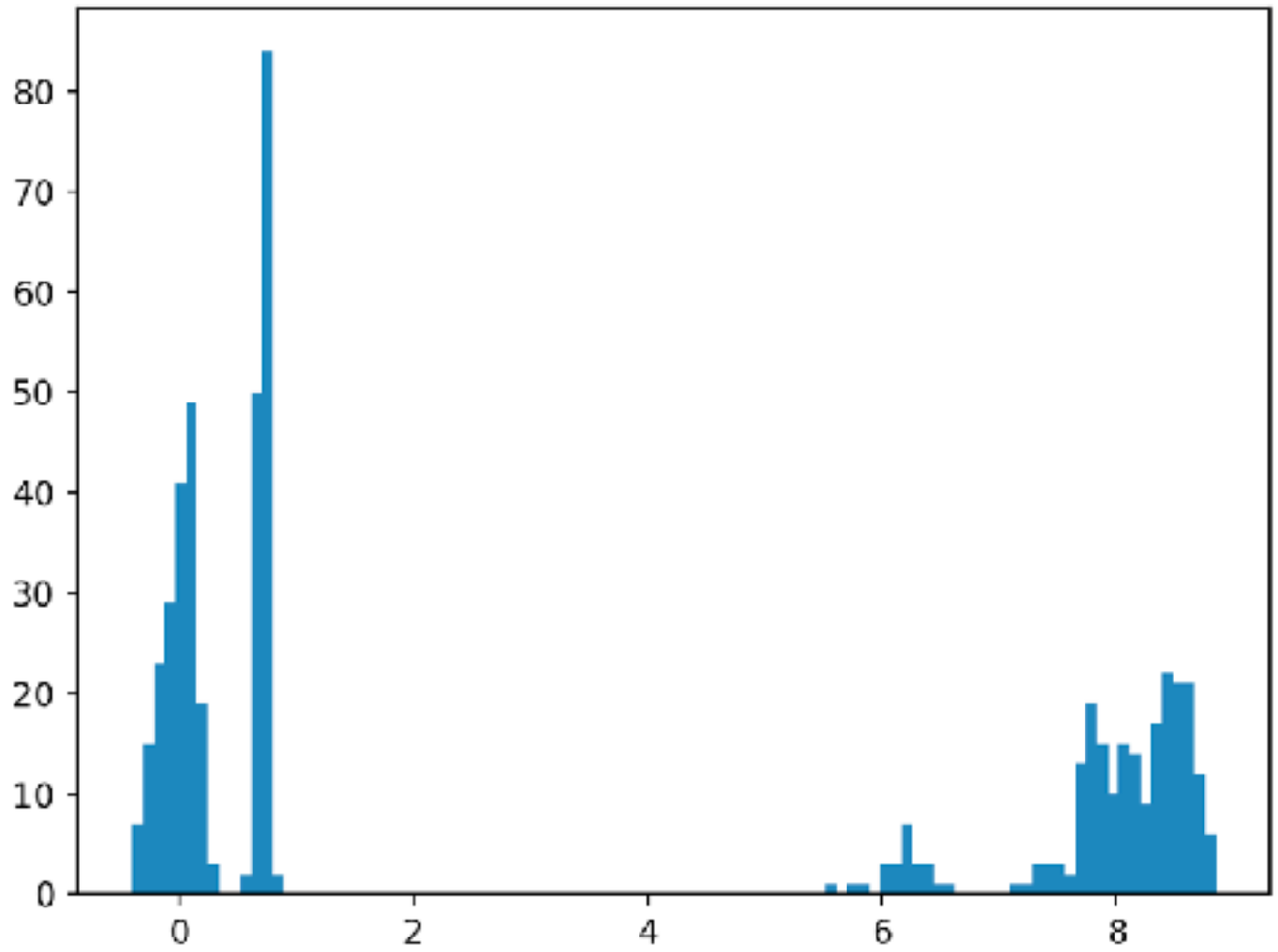
Result:

$$\text{Number of } E_6 \text{ Models on } T = .00059128 \times \frac{1}{3} \times 2.96 \times 10^{755} = 5.83 \times 10^{751}.$$

- **Check:** with 5 batches, 2 million random samples each.

$$\begin{aligned} \text{From Theorem} \quad &: \quad .00059128 \times 2 \times 10^6 = 1182.56 \\ \text{From Random Samples} \quad &: \quad 1183,\ 1181,\ 1194,\ 1125,\ 1195 \end{aligned}$$

Intelligible AI from Logistic Regression Coefficients Histogram



Data:
413264 First No Sen
407487 Last Sen

Accuracy: 98.5%

Similar, but **intuitive push-pull game**. Intercept -7, trees with coefs > 0 pull to right. Only need to add a few trees with coefficient > 5 to be in serious danger of no Sen Limit.

Visualizing Dangerous Trees, Working Towards Conjecture

- **Red edges:** any trees there or on triangle wraparounds are in the set of dangerous trees.

  i.e. 12 bad ones associated with red edges.

- **25 other bad:** 18 of which are connected to vertices.

- remaining dangerous trees are of understandable type.

  intuitive, but **ML —> linchpins.**