Guiding Intuition with Saliency Analysis

Adam Zsolt Wagner

Hausdorff School: “Machine Learning and Theorem Proving”

Day 3

September 20, 2023

Day 2 overview

Main idea: use supervised learning to train a neural net on a large dataset, and find patterns in it.

The type of question we want to answer is: if we know a bunch of things about an object (such as a graph, a matrix, a knot), but not the object itself, can we predict other parameters of this object?

We will use this method to come up with conjectures, and to guide our intuition.

Supervised learning

“Learning with training labels”; this is the most commonly used learning technique.

We start with a dataset of lots of (input, correct output) pairs. We want to learn from these, and then be able to predict the outputs for some new input values.

Generally more data $\implies$ better predictions.

Typical supervised learning applications

Predicting housing market prices

- Given a lot of data on houses on the market currently, try to predict how much your house might be worth.

Rooms	Size	Has kitchen	Brick type	Haunted	...	Price
5	$130m^2$	Yes	Limestone	No	...	\$1000000
3	$50m^2$	Yes	Marble	No	...	\$98000
2	$25m^2$	No	Mud	No	...	\$37500
3	$80m^2$	No	Limestone	Yes	...	???

Typical ML applications: Classification problems

- Given 1000 labeled images of cats and dogs. (Training set)
- Given a new image, guess whether it's a cat or dog.

Given an email, classify if it is spam or not. Given a handwritten character, classify it as one of the known characters. Facial emotion classification.

Supervised learning

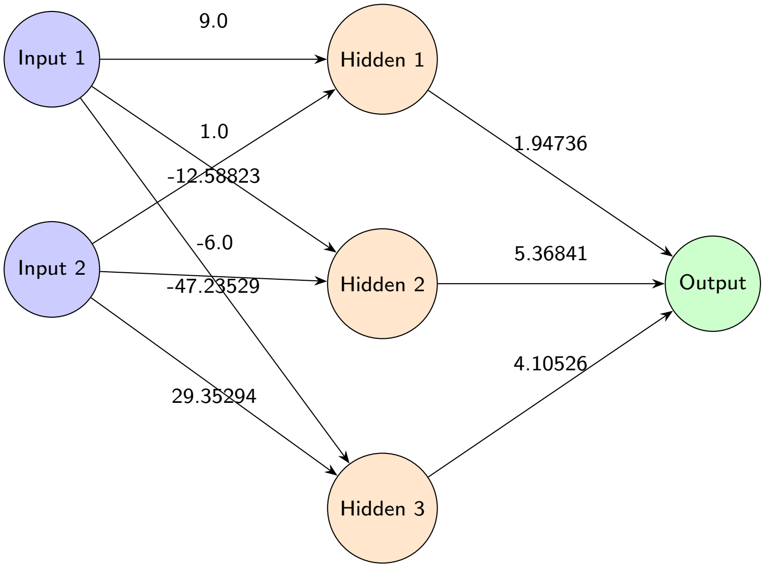
There is a correct function, that maps grayscale pictures of cats and dogs to $\{$cats, dogs$\}$. This is a function

$$F : [256]^{10000} \to \{\text{cat,dog}\}.$$

We want to learn this function, but we cannot list all the 256^{10000} correct (input, output) pairs. Instead, we will use neural networks as *function approximators*.

F is a very complicated function, we cannot completely learn it. Instead we will find a much simpler function f (given by the neural net), that is pretty close to F.

Neural networks



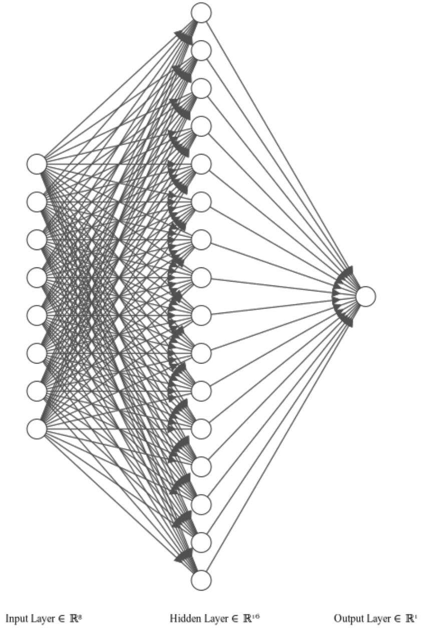
Supervised learning

Let's say we want to train a neural network that output a positive value on pictures of cats, and a negative value on pictures of non-cats.

We take a picture from the input dataset, let's say the label says "cat". Then we change the weights of the neural network a teeny-tiny bit in such a way that it outputs a bigger positive value next time we plug in this picture to the network.

We adjust the weights by using e.g. gradient descent, on the space of weights of the network. We repeat this process many times, until we get good at approximating F.

Supervised learning


Input Layer $\in \mathbb{R}^8$
Hidden Layer $\in \mathbb{R}^{16}$
Output Layer $\in \mathbb{R}^1$

Supervised learning

If a network has 1000 parameters (edges), then every time we adjust its weights, we are taking a tiny step in a 1000-dimensional space.

In high-dimensional spaces, most local extrema are saddle points. We are less likely to get stuck in a local minimum than with simpler search algorithms.

Any* function can be approximated with a large enough neural network.

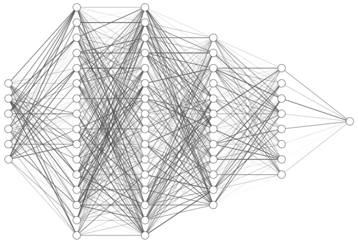
*: false in theory, and even more false in practice

TensorFlow Playground

Open TensorFlow Playground

Interpreting Neural Networks

- **Learning Patterns**: Neural networks excel at learning complex patterns from data.
- **Challenges for Humans**: However, understanding what these learned patterns represent can be difficult for humans.
- **Parameter Space**: The network's parameters capture the learned patterns, but they may not be interpretable by humans.



- **No Guarantee of Interpretability**: There is no guarantee that we can extract meaningful human-understandable insights directly from the network's parameters.

When does supervised learning work best?

- **Overcoming the curse of dimensionality**:
  - Deep learning excels at handling high-dimensional data.
  - Traditional methods struggle as the dimensionality increases.
- **Working in the unit cube**:
  - Normalize input data to the unit cube $[0, 1]^d$.
  - Helps gradients flow consistently during training.
- **Coordinates have low symbolic content**:
  - Don't want noise sensitivity. Small changes in input shouldn't affect outcome much.
  - Abstract, high-level features are often more valuable.

Let’s now see some simple examples in maths!

Simple examples in maths – the parity bit

- **Input:** 0-1 sequence of length 1000.
- **Output:** $\sum x_i \bmod 2$.

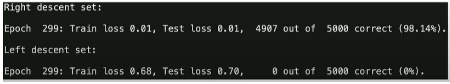
- **Noise Sensitivity**:
  - This is the most noise sensitive problem you can come up with!
  - In this case, every single bit flip changes the outcome.
  - Very difficult to learn with a vanilla neural net.

Learning Descent Sets in Permutations – By Joel Gibson

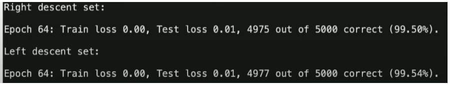
- Input: $x = (x_1, x_2, \ldots, x_n)$ a permutation of $[n]$.
- We would like to predict the right and left descent sets:
  - $R(x) = \{i : x_i > x_{i+1}\}$ (Right Descent Set).
  - $L(x) = \{i : i \text{ occurs to the right of } i+1 \text{ in } x\}$ (Left Descent Set).
- **Symmetry**:
  - $R(x) = L(x^{-1})$ (Symmetrical concepts).

Data Representation Matters

- **Observation**:
  - Input permutation as a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ - Neural network learns $R(x)$ quickly (for $n = 50$), but struggles on $L(x)$.
  - Input permutation as a permutation matrix - Neural network easily learns both $R(x)$ and $L(x)$.



```
Right descent set:
Epoch  299: Train loss 0.01, Test loss 0.01,  4907 out of  5000 correct (98.14%).
Left descent set:
Epoch  299: Train loss 0.68, Test loss 0.70,     0 out of  5000 correct (0%).
```



```
Right descent set:
Epoch 64: Train loss 0.00, Test loss 0.01, 4975 out of 5000 correct (99.50%).
Left descent set:
Epoch 64: Train loss 0.00, Test loss 0.01, 4977 out of 5000 correct (99.54%).
```

How we input data really matters!

I really recommend Geordie Williamson's fantastic talk "What can the working mathematician expect from deep learning?"

The Importance of the Training Dataset

- **Scenario**: Training a network to predict whether a matrix is invertible or not.
- **Dataset**: We start with a dataset of random matrices.

- **The Unexpected Issue**:
  - Every random matrix in the dataset is likely to be invertible!

Fixing the dataset

Imagine that our previous dataset consisted of 100×100 matrices, with random real entries between 0 and 1.

For half of the matrices in the dataset, we could pick a random row, and replace it with a random linear combination of all other rows. These matrices would now not be invertible.

Problem: these matrices will likely have entries less than 0 or bigger than 1, while the invertible ones in the dataset don't have such entries.

The network might just learn to predict "not invertible" if the matrix has an entry not in $[0, 1]$.

Issues with interpreting the result

Supervised learning is like non-linear regression, but we do not get a simple, human-friendly formula at the end.

What we get is an approximation of a very complicated function F by another complicated function f, given by the neural network. We also get the ability to compute f at any point we want.

Problem: in mathematics we would often like to understand what this function is!

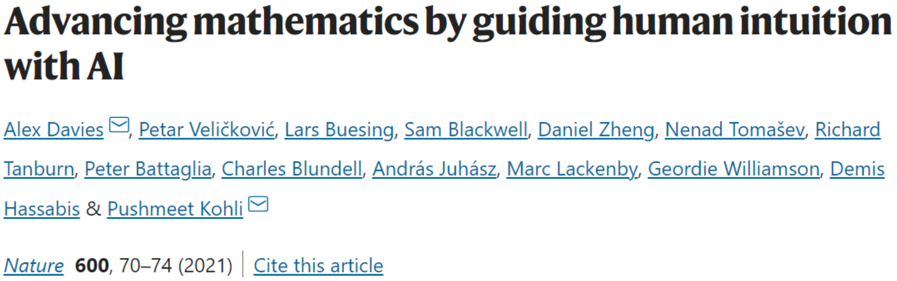
Saliency analysis to the rescue

Neural networks often appear as "black boxes". One way to get understanding out of a trained neural network is through *Saliency analysis*.

The main idea:

- We wiggle the input coordinates one by one.
- Observe how these perturbations affect the network's output.
- Coordinates causing significant output changes are deemed important features.
- If a coordinate's perturbation has little impact, the network likely doesn't rely on it for predictions.

A real application in maths

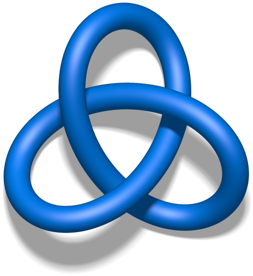


Advancing mathematics by guiding human intuition with AI

Alex Davies ✉, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis & Pushmeet Kohli ✉

Nature **600**, 70–74 (2021) | Cite this article

Knot theory



Type of questions knot theorists care about:

- Classifying all possible knots,
- deciding whether two knots are the same,
- how are various parameters of knots related, etc.

Knot Theory

Knot theory has three very distinct subfields:

- Hyperbolic knot theory
- Gauge/Floer theory
- Quantum topology

Knot Invariants by field:

Hyperbolic Invariants:

- Volume
- Cusp shape and volume
- Length spectrum
- Trace field

Gauge Theory Invariants:

- signature
- Heegaard Floer homology
- Instanton Floer homology
- s, τ, ϵ, Υ

See Marc Lackenby's talk “Using machine learning to formulate mathematical conjectures” for more details about these invariants.

Knot theory

We created a huge dataset. For many knots K, we calculate the vector

$v(K) =$ (volume, cusp shape, length spectrum, trace field, ...).

We also calculate the signature of K. The training set of the neural network will consist of lots and lots of

$$v(K) : \text{signature}(K)$$

pairs.

We train a neural network to try to predict the signature based on $v(K)$. If there is no connection between hyperbolic parameters in $v(K)$ and the signature, then the neural network will struggle. If the neural network does unreasonably well however, then there must be a connection between $v(K)$ and the signature that we don't know about!

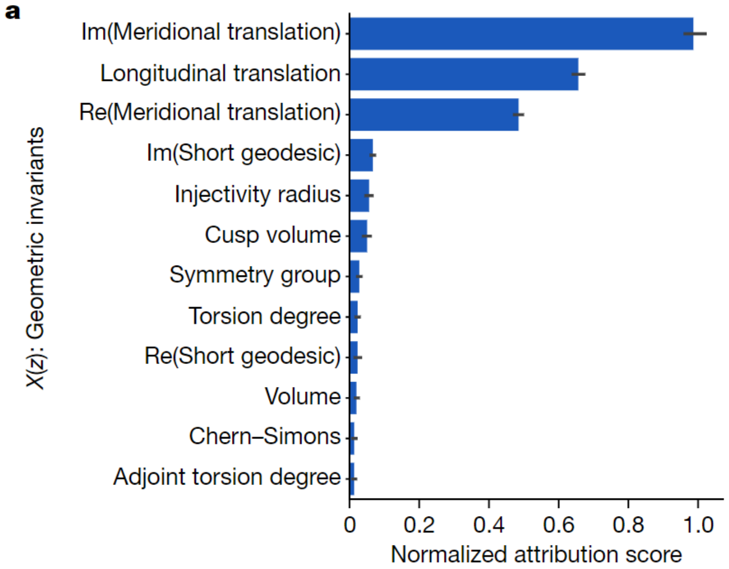
Knot theory

Turns out, the neural network does a really good job at predicting the signature.

Now all that is left is to figure out what the connection between all these parameters is.

Let's wiggle the entries of $v(K)$ and do saliency analysis:

Saliency analysis on knots



Knot theory

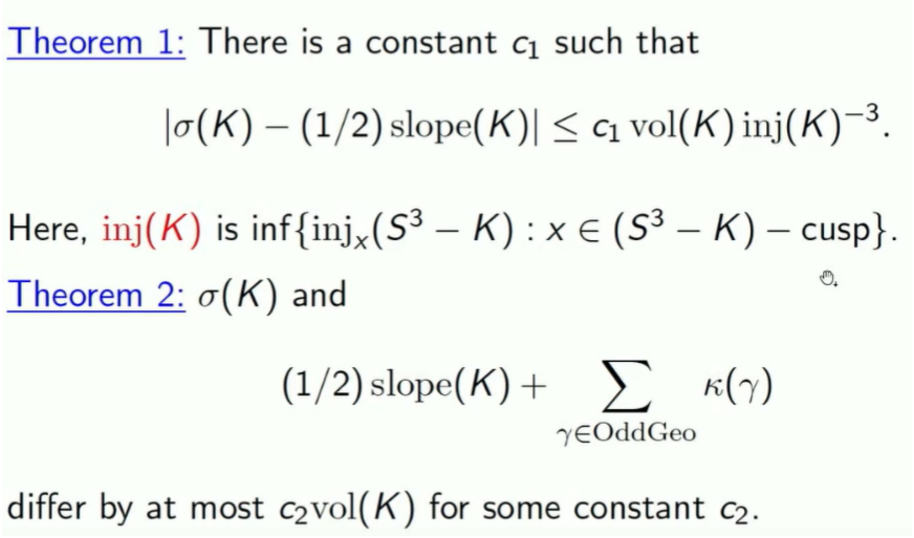
We combine the first three parameters in something called the *slope*, and by plotting the signature versus the slope we come up with the following conjecture:

Conjecture

$$\mathit{signature}(K) \approx \frac{1}{2}\mathit{slope}(K).$$

This turns out to be false. However, after a bit more work they proved the following two statements:

Knot theory



Theorem 1: There is a constant c_1 such that

$$|\sigma(K) - (1/2)\,\mathrm{slope}(K)| \leq c_1\,\mathrm{vol}(K)\,\mathrm{inj}(K)^{-3}.$$

Here, $\mathrm{inj}(K)$ is $\inf\{\mathrm{inj}_x(S^3 - K) : x \in (S^3 - K) - \text{cusp}\}$.

Theorem 2: $\sigma(K)$ and

$$(1/2)\,\mathrm{slope}(K) + \sum_{\gamma \in \mathrm{OddGeo}} \kappa(\gamma)$$

differ by at most $c_2\mathrm{vol}(K)$ for some constant c_2.

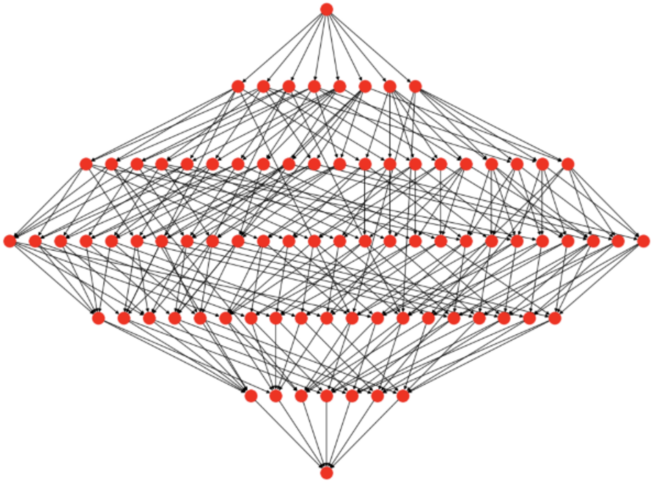
Representation theory

To any pair of permutations we can associate two objects: the Bruhat graph, and the Kazhdan-Lusztig polynomial.

Combinatorial invariance conjecture: the KL polynomial can be computed from the Bruhat graph.

A neural network was trained on 20,000 Bruhat graphs, and achieved 98% accuracy. Reason for optimism!

Representation theory



$\leftrightarrow 1 + 3q + q^2$

Representation theory

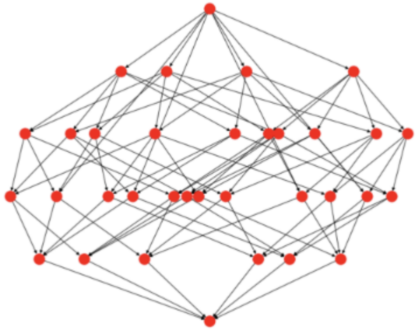
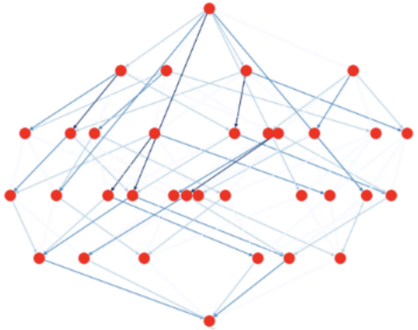
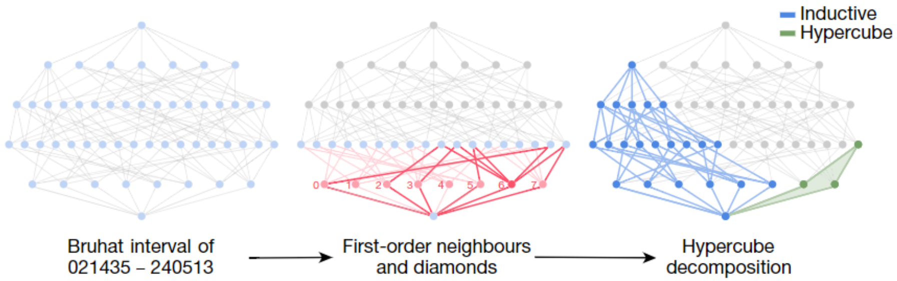



FIGURE 3. Bruhat interval pre and post saliency analysis.

Representation theory


Inductive
Hypercube
0
1
2
3
4
5
6
7
Bruhat interval of
021435 – 240513
First-order neighbours
and diamonds
Hypercube
decomposition