197 Final Presentation

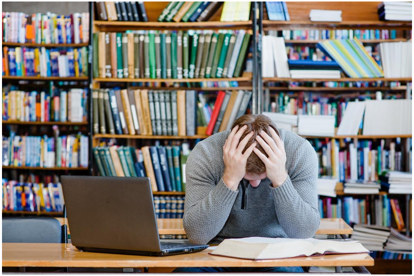
Signals, Structures, and Systems: Mathematical Perspectives on Deep Learning Models

Amshu Bellur, Darius Mahamedi, Angelina Jordan, and Natalie Wu

June 5th 2025

What does ML mean?
Machine learning, at a very high level, is a collection of algorithms which learn patterns from data and make decisions or predictions without being explicitly programmed

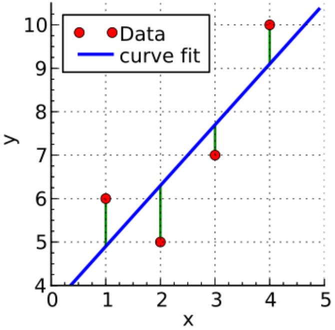
(a) **M**achine **L**earning

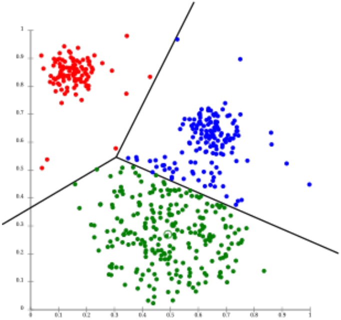


(b) ~~**Me Learning**~~

Introduction (Darius)

Basic Idea: The computer learns to approximate a hypothetical function given some input.
The program 'learns' to approximate the function better by updating what are called 'parameters'





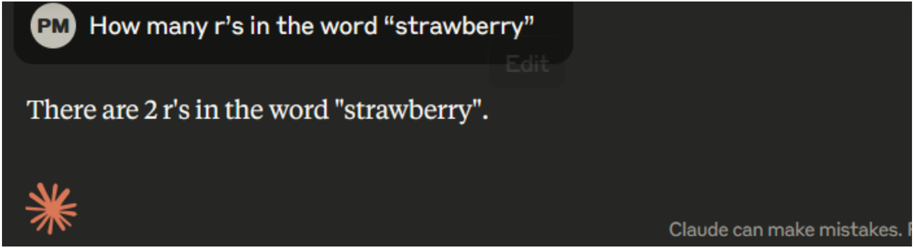
On the right, we are approximating a probability distribution

Introduction (Darius)

There are three types of machine learning

- **Supervised Learning:** The model learns from labeled data, using input-output pairs to make future predictions.
- **Unsupervised Learning:** The model finds hidden patterns or groupings in unlabeled data without explicit guidance
- **Reinforcement Learning:** The model learns by interacting with an environment, receiving rewards or penalties to guide its behavior.

Together, they can accomplish amazing things:



What Is a Neural Network?

In Simple Terms:

- A neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$, built from layers of tunable parameters (weights and biases) *(Sundararajan et al., 2017; Goodfellow et al., 2016)*.
- It maps input data (e.g., numbers, vectors, or images) to outputs (e.g., labels or continuous values) by applying a sequence of transformations *(LeCun et al., 2015)*.
- Each layer applies an affine transformation followed by a nonlinearity, gradually extracting and combining features from the data *(Nielsen, 2015)*.
- Neural networks are a core method in **machine learning**, where algorithms learn patterns from data to make predictions without being explicitly programmed *(Mitchell, 1997)*.

Simple Feedforward Neural Network

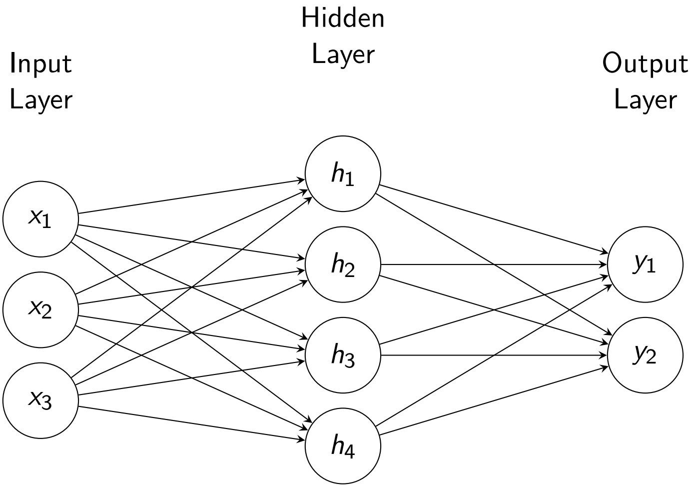


Diagram inspired by Nielsen (2015).

More about Neural Networks

- A basic neural network architecture is the **Multilayer Perceptron (MLP)**, which consists of a composition of layers of connected units (neurons). This composition can be written as:

$$f(x) = (f_L \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1)(x),$$

  where each f_l is a **layer function** that maps inputs to outputs *(Goodfellow et al., 2016)*.
- In an MLP, a layer function typically computes:

$$z_j = \sum_{i=1}^{M} w_{ji} x_i + b_j, \quad y_j = \sigma(z_j),$$

  where x_i are the inputs, w_{ji} are the weights, b_j is a bias term, and σ is a nonlinear activation function (e.g., ReLU). Thus, each layer in the composition applies an affine transformation followed by a nonlinearity *(Nielsen, 2015)*.
- The network is trained end-to-end to approximate a function $y = f(x; \theta)$, where θ represents the learnable parameters (weights and

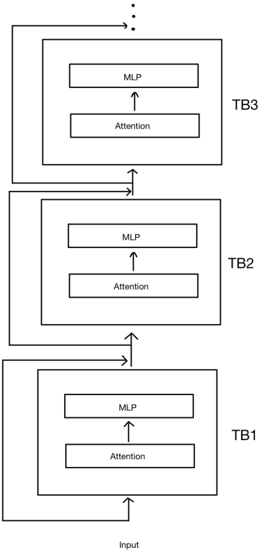
Why Are Neural Networks Important?

- Neural networks are the core building blocks of deep learning *(LeCun et al., 2015)*.
- Deep learning uses large neural networks with many layers to learn complex patterns in data *(Goodfellow et al., 2016)*.
- These networks can automatically extract useful features from raw input — such as images, audio, or text *(Krizhevsky et al., 2012)*.
- This ability has led to breakthroughs in areas like computer vision, natural language processing, and robotics *(Vaswani et al., 2017)*.

Attention

- Attention is building contextual representation based on the words around it
- Each layer of the transformer weighs and combines representations from other relevant tokens in the context from the previous layer to build the representation for tokens in the current layer
- Words get associated to vectors, the vectors are compared against each other with the matrices Q, K, and V. From that a new sequence is produced which captures some of the relationships between the initial vectors
- Attention $= \text{softmax}(\frac{QK^T}{\sqrt{d_k}})\text{V}$

Transformers



- It is the standard architecture for building larger models
- Each transformer block is made up of an attention layer and an MLP layer
- Transformers fundamentally changing the field of speech and language processing

Attribution Methods in Neural Networks

What Are Attribution Methods?

- Attribution methods explain which input features contribute most to a model's prediction.
- They help build trust, transparency, and diagnose models.
- We'll test Saliency, Gradient $\times$ Input, Integrated Gradients, and Shapley Values.

Reference: Jethani et al. (2021). *Fast Shapley Explanations for Neural Networks with Deep Approximate Shapley Propagation*.

Saliency Analysis

Formula:

$$R_i(x) = \left| \frac{\partial S(x)}{\partial x_i} \right|$$

$S(x)$: model output
x_i: input feature i

Description: Measures how much the output changes when feature x_i changes slightly.

Reference: Simonyan et al. (2014)

Saliency Analysis: Implementation Steps

- Set model to evaluation mode.
- Enable gradient tracking on the input.
- Perform a forward pass to get output score $S(x)$.
- Compute the gradient of that score with respect to each input x_i.
- Take the absolute value of the gradient as the attribution score.

Gradient × Input

Formula:

$$R_i(x) = \frac{\partial S(x)}{\partial x_i} \cdot x_i$$

$S(x)$: model output
x_i: value of input feature i
$\frac{\partial S(x)}{\partial x_i}$: gradient of the output with respect to x_i

Description: Multiplies each feature by how sensitive the output is to it.

Reference: Shrikumar et al. (2017); Ancona et al. (2018)

Gradient $\times$ Input: Implementation Steps

- Compute the gradient of the model's output with respect to input.
- Multiply each gradient value by its corresponding input value.
- The result reflects each input's contribution to the output.

Integrated Gradients

Formula:

$$R_i(x) = (x_i - \bar{x}_i) \cdot \int_{\alpha=0}^{1} \frac{\partial S(\tilde{x})}{\partial \tilde{x}_i}\bigg|_{\tilde{x}=\bar{x}+\alpha(x-\bar{x})} d\alpha$$

x_i: input feature i
$\bar{x}_i$: baseline value for feature i
α: interpolation factor between 0 and 1
$\tilde{x}$: interpolated input between baseline and input
$S(\tilde{x})$: model output

Description: Averages gradients along the path from a baseline to the actual input.

Baseline: A reference input used for comparison, typically representing the absence of features (e.g., a zero vector).

Reference: Sundararajan et al. (2017)

Integrated Gradients: Implementation Steps

- Choose a baseline input $\bar{x}$ (e.g., all zeros).
- Interpolate inputs between baseline and actual input.
- At each step, compute gradients of output w.r.t. input.
- Average the gradients and multiply by $(x - \bar{x})$.

Shapley Values

Formula:

$$R_i = \sum_{S \subseteq P \setminus \{i\}} \frac{|S|!(|P| - |S| - 1)!}{|P|!} \left[\hat{f}(S \cup \{i\}) - \hat{f}(S) \right]$$

P: set of all input features
S: subset of features excluding i
$\hat{f}(S)$: model output using only the features in S
$\sum_{S \subseteq P \setminus \{i\}} \frac{|S|!(|P|-|S|-1)!}{|P|!}$: all possible subsets S

Description: Averages the added value of feature i across all possible subsets, weighted fairly.

Reference: Lundberg and Lee (2017)

Shapley Values: Implementation Steps

- Define a baseline input (e.g., zeros).
- Sample many subsets S of features without i.
- For each subset, compute model output with and without feature i.
- Compute the difference and weight it based on subset size.
- Average the results to estimate the contribution of feature i.

Deep Approximate Shapley Propagation (DASP)

Formula:

$$\mathbb{E}[R_i] = \frac{1}{N} \sum_{k=0}^{N-1} \mathbb{E}_k[R_{i,k}]$$

$$\mathbb{E}_k[R_{i,k}] = \mathbb{E}_{S \subseteq P \setminus \{i\}, |S|=k}[f(x_{S \cup \{i\}})] - \mathbb{E}_{S \subseteq P \setminus \{i\}, |S|=k}[f(x_S)]$$

N: number of subset sizes used
k: number of features in subset S
P: set of all features
$f(x_S)$: model output using features in S

Description: Approximates Shapley values using random subsets and uncertainty propagation.

Reference: Jethani et al. (2021)

DASP: Implementation Steps

- Represent features as probabilistic distributions.
- Propagate these through layers using uncertainty propagation.
- Estimate marginal contributions without enumerating all subsets.
- Aggregate contributions to compute approximate Shapley values.

Example of Implementing Methods

Model Input

$$\texttt{input} = [0.5, -0.5]$$

Attribution Results

Method	Feature 1	Feature 2
Saliency	0.15	0.02
Gradient $\times$ Input	0.12	-0.03
Integrated Gradients	0.10	-0.01
Shapley Values	0.08	0.00

Darius

Definition We can take a *residual neural network* to be a neural network where the neuron activation functions are given as follows:

$$\begin{cases} x(k+1) = x(x) + \omega(k) \cdot \sigma(a(k)x(k) + b(k)) \\ x(0) = x \end{cases}$$

Here, k indicates the layer of the neuron, and $\sigma(\cdot)$ is a Lipschitz function.

Note that

$$a(k)x(k) + b(k)$$

is an affine transformation.

We can use this to approximate a derivative:

$$\begin{cases} \dot{x} = \omega(t) \cdot \sigma(a(t)x(t) + b(t)) \\ x(0) = x \end{cases}$$

Aside: After each layer in a transformer, we are left with an output vector. We assume that after each layer, the output is *normalized* so the output vector has norm one

Consequence: For simplicity, we can take the data/inputs to be on $\mathbb{S}^{d-1}$ throughout, where d is the original size of our input.

This means we can think of a transformer as a "flow map" on $(\mathbb{S}^{d-1})^n$.

Darius

We get the dynamics

$$\dot{x}_i(t) = P^{\perp}_{x_i(t)}\left(\frac{1}{Z_{\beta,i}(t)} \sum_{j=1}^{n} e^{\beta < Q(t)x_i(t),\ K(t)x_j(t) >} V(t)x_j(t)\right.$$

where

$$P^{\perp}_{x} y = y - < x,\ y > x$$

is the projection of $y \in \mathbb{R}^d$ onto $T_x \mathbb{S}^{d-1}$ and $Z_{\beta,i}(t) > 0$ is

$$Z_{\beta,i}(t) = \sum_{k=1}^{n} e^{\beta < Q(t)x_i(t),\ K(t)x_k(t) >}$$

Darius

Some Examples

Lets say $Q = K = V = Id$, let $\beta = 1$ and

- $x_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- $x_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

Then $< x_1, x_2 >=< x_2, x_1 >= 0$, and $< x_1, x_1 >=< x_2, x_2 >= 1$.
We have $Z_{1,1}(t) = Z_{1,2}(t) = e + 1$.

- $\dot{x}_1(t) = P^{\perp}_{x_1(t)}(\frac{1}{1+e} \cdot (e \cdot x_1(t) + x_2(t))) = P^{\perp}_{x_1(t)}(\begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix})$
- $\dot{x}_2(t) = P^{\perp}_{x_2(t)}(\frac{1}{1+e} \cdot (x_1(t) + e \cdot x_2(t))) = P^{\perp}_{x_2(t)}(\begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix})$

Darius

- $\dot{x}_1(t) = P^{\perp}_{x_1(t)}(\begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix}) = \begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix} - < \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix} > \begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix}$
$= \begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix} - \frac{e}{1+e} \begin{pmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{pmatrix} = \begin{pmatrix} \frac{e}{(1+e)^2} \\ \frac{1}{(1+e)^2} \end{pmatrix}$
- $\dot{x}_2(t) = P^{\perp}_{x_2(t)}(\begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix}) = \begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix} - < \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix} > \begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix}$
$= \begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix} - \frac{e}{1+e} \begin{pmatrix} \frac{1}{1+e} \\ \frac{e}{1+e} \end{pmatrix} = \begin{pmatrix} \frac{1}{(1+e)^2} \\ \frac{e}{(1+e)^2} \end{pmatrix}$

Multi-Headed Attention

$$\dot{x}_i(t) = P^{\perp}_{x_i(t)}\left(\frac{1}{Z_{\beta,i}(t)}\left(\sum_{h=1}^{H}\sum_{j=1}^{n} e^{\beta<Q_h(t)x_i(t),K_h(t)x_j(t)>} V_h(t)x_j(t)\right)\right)$$

Full Transformer

$$\dot{x}_i(t) = P^{\perp}_{x_i(t)}\left(\frac{1}{Z_{\beta,i}(t)}\left(\sum_{h=1}^{H}\sum_{j=1}^{n} e^{\beta<Q_h(t)x_i(t),K_h(t)x_j(t)>} V_h(t)x_j(t)\right) + \omega(t)\sigma(a(t)x_i(t) + b(t))\right)$$

Darius

Tragically, we can generalize the previous tools to get a Partial Differential Equation.

Let $\dot{x}_i(t) = \chi[\mu(t)](x_i(t))$

Where $\mu(t, \cdot) = \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i(t)}(\cdot)$

$\chi[\mu] : \mathbb{S}^{d-1} \to T\mathbb{S}^{d-1}$ is given by $\chi[\mu](x) = P_x^{\perp}(\frac{1}{Z_{\beta,\mu}(x)} \int e^{\beta<x,y>} y \, \mathrm{d}\mu(y))$

with $Z_{\beta,\mu}(x) = \int e^{\beta<x,y>} \mathrm{d}\mu(y)$ The evolution of $\mu(t)$ is governed by

$$\begin{cases} \partial_t \mu + \mathrm{div}(\chi[\mu]\mu) = 0, \text{ on } \mathbb{R}_{\geq 0} \times \mathbb{S}^{d-1} \\ \mu|_{t=0} = \mu(0), \text{ on } \mathbb{S}^{d-1} \end{cases}$$

The above is called the continuity equation, and it has been solved for simple cases (i.e., $Q = K = V = Id$), and we pursued solving through a spherical harmonic expansion:

$$\mu(\theta, \phi) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} a_{\ell m} Y_\ell^m(\theta, \phi)$$

Transcoder Over Transformer

Why Transcoder Over Transformer

Transformers

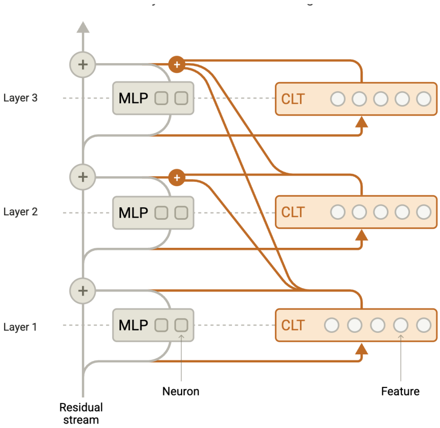
- It is dense which leads to containing a lot of information
- Having too much depth makes it hard to find those connections
- We don't know why certain words are making certain connections because there's too much going on

Transcoder

- **Solution:** Transcoder
- Helpful for interpreting the mechanism of transformers
- It is a simple neural network that has sparse entries
- Easier to analyze connections because it's less dense
- We are not only defining the object, but the relationship grammatically
- It is making connections to the other words in the sentence

Reference: Dunefsky et al. (2024)

Transcoder Diagram



Reference: Ameisen, Lindsey, Pearce et al. (2025)

Pseudocode

1. Load GPT-2 Model and Tokenizer

[1] *Initialize* `tokenizer` ← LoadGPT2Tokenizer *Initialize* `gpt2_model` ← LoadGPT2Model

2. Define a Simple 2-Layer Neural Network

[1] *Function* `SimpleNN`(*input_dim*, *hidden_dim*, *output_dim*) *Initialize* `W1` ← RandomWeights*input_dim*, *hidden_dim* *Initialize* `b1` ← Zeros*hidden_dim* *Initialize* `W2` ← RandomWeights*hidden_dim*, *output_dim* *Initialize* `b2` ← Zeros*output_dim* *Return* Neural Network with parameters (`W1`, `b1`, `W2`, `b2`) *Call* `neural_net` ← SimpleNN*gpt2_embedding_dim*, *hidden_layer_size*, *gpt2_embedding_dim*

3. Define a Loss Function

[1] *Function* `Loss`(*predicted_target*, *true_target*, *hidden_values*, *lambda_penalty*) `Difference_from_GPT2` ← MeanSquaredError*predicted_target*, *true_target* ← *lambda_penalty* × SumOfSquares*hidden_values* *Return* `Difference_from_GPT2` + `Penalty_for_large_hidden_values`

4. Train the Neural Network

[1] *step* from 1 to 10 *Choose a word* `word` *Tokenize* `input_tokens` ← tokenizer.encode`word` *Get GPT-2 activations* `gpt2_outputs` ← gpt2_model.forward`input_tokens` *Extract input vector* `input_vector` ← GetLayerActivation`gpt2_outputs`, *input_layer_index* *Extract target vector* `target_vector` ← GetLayerActivation`gpt2_outputs`, *target_layer_index*

Forward pass through neural network `hidden_output` ← ReLU(`input_vector` · `W1` + `b1`) `predicted_target` ← (`hidden_output` · `W2` + `b2`)

Calculate Loss `current_loss` ← Loss`predicted_target`, `target_vector`, `hidden_output`, *lambda_penalty*

Backpropagate and Adjust Network Weights MinimizeLoss`neural_net`, `current_loss`

5. Do Coreference Resolution

[1] *Define pronoun* `pronoun` ← "it" *Define possible meanings* `possible_meanings` ← ["cat", "mat", "dog", ...]

Tokenize pronoun `pronoun_tokens` ← tokenizer.encode`pronoun` *Get GPT-2 output for pronoun* `pronoun_gpt2_output` ← gpt2_model.forward`pronoun_tokens` *Get GPT-2 vector of the pronoun* `pronoun_vector` ← GetLastLayerEmbedding`pronoun_gpt2_output`

each `meaning` in `possible_meanings` *Tokenize meaning* `meaning_tokens` ← tokenizer.encode`meaning` *Get GPT-2 output for meaning* `meaning_gpt2_output` ← gpt2_model.forward`meaning_tokens` *Get GPT-2 vector of the meaning* `meaning_vector` ← GetLastLayerEmbedding`meaning_gpt2_output`

Loss Function

- Tells us how well our model is performing compared to a known model
- Act as a feedback mechanism
- Helps the model learn and improve over time
- **Loss Function**
- $||NN_i(\sum_{j=1}^{i} \vec{x_j}) - TB_i(x_i)|| + \lambda||L(NN_i(x_i))||$

Reference: Ameisen, Lindsey, Pearce et al. (2025)

Results

- We were unable to fully complete the code to work with all possible sentence inputs
- We hope to work more on it in the future
- We want to hopefully make it better one day.

Future

- Transformers outperform encoder-decoder models in translation
- Attention improves focus on relevant words
- Used in image generation, music, drug design
- Future: smarter assistants, tutoring systems, semantic robotics

Reference: Vaswani, Shazeer, Parmar et al. (2017)

Thank you!

Thank you, NordVPN

References

- Sundararajan, M., Taly, A., Yan, Q. (2017). Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*.
- Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press.
- LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
- Nielsen, M. (2015). *Neural Networks and Deep Learning*. neuralnetworksanddeeplearning.com
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
- Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *NeurIPS*.
- Vaswani, A., et al. (2017). Attention is all you need. *NeurIPS*.

References

- Jethani, N., Sundararajan, M., Wang, D., Varshney, K. R. (2021). *Fast Shapley Explanations for Neural Networks with Deep Approximate Shapley Propagation*. NeurIPS.
- Simonyan, K., Vedaldi, A., Zisserman, A. (2014). *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*.
- Sundararajan, M., Taly, A., Yan, Q. (2017). *Axiomatic Attribution for Deep Networks*. ICML.
- Lundberg, S. M., Lee, S.-I. (2017). *A Unified Approach to Interpreting Model Predictions*. NeurIPS.
- Ancona, M., Ceolini, E., Öztireli, C., Gross, M. (2018). *Towards better understanding of gradient-based attribution methods for Deep Neural Networks*. ICLR.
- Shrikumar, A., Greenside, P., Kundaje, A. (2017). *Learning Important Features Through Propagating Activation Differences*.

References

- Dunefsky, J. Chlenski, P (2024). *Transcoders Find Interpretable LLM Feature Circuits*. NeurIPS.
- Amesisen, E., Lindsey, J., Pearce, A, more (2025). *Circuit Tracing: Revealing Computational Graphs in Language Models*. Anthropic.
- Vaswani, A., Shazeer, N., Parmar, N, more (2017). *Attention Is All You Need*. NIPS