

Mechanistic Interpretability: an overview.

Presenter: Simone Petruzzi



SAPIENZA
UNIVERSITÀ DI ROMA

About me



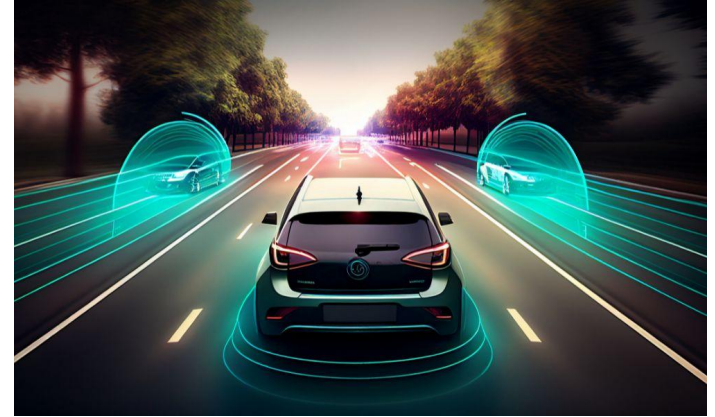
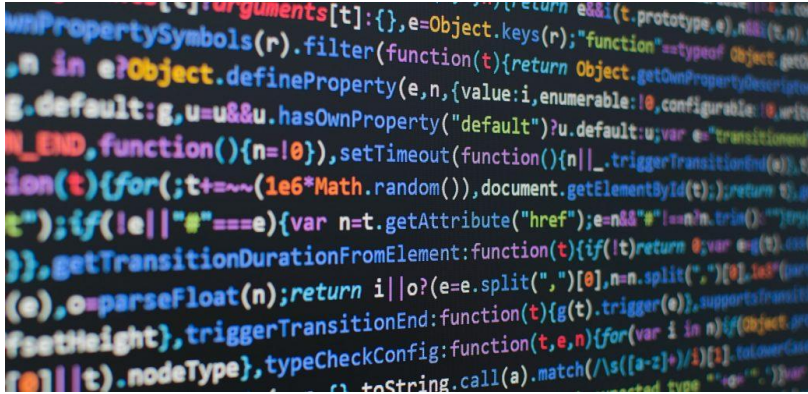
- Bachelor's Degree in Engineering in Computer Science (2021),
Sapienza University of Rome
- Master's Degree in Engineering in Computer Science (2024),
Sapienza University of Rome
- PhD in Data Science (2024-2027)
Sapienza University of Rome

An apology first

This collection of slides is mostly composed by screenshots taken on various blog posts and papers which I found very useful to prepare this lecture (you can find all the reference links during the slides).

I'm sorry but they were just too good !

Why interpretability ?



Why “mechanistic” interpretability

In 2020 a group of researchers from OpenAI led by [Christopher Olah](#), introduced the idea of “reverse-engineering” the computational **mechanisms** (which they call **circuits**) and representations of a neural network. This became known as mechanistic interpretability and largely grew outside the mainstream literature ([Anthropic](#), [lesswrong](#), [distill](#),...) as a reaction to xAI, which was mainly based on attribution maps.

First glance at circuits

[Zoom In: An Introduction to Circuits](#)

Windows (4b:237)
excite the car detector
at the top and inhibit
at the bottom.



Car Body (4b:491)
excites the car
detector, especially at
the bottom.



Wheels (4b:373) excite
the car detector at the
bottom and inhibit at
the top.



● positive (excitation)
● negative (inhibition)



A **car detector (4c:447)**
is assembled from
earlier units.

First glance at circuits

[Zoom In: An Introduction to Circuits](#)



THREE SPECULATIVE CLAIMS ABOUT NEURAL NETWORKS

Claim 1: Features

Features are the fundamental unit of neural networks.

They correspond to directions.¹ These features can be rigorously studied and understood.

Claim 2: Circuits

Features are connected by weights, forming circuits.²

These circuits can also be rigorously studied and understood.

Claim 3: Universality

Analogous features and circuits form across models and tasks.

Left: An activation atlas ^[13] visualizing part of the space neural network features can represent.

Features

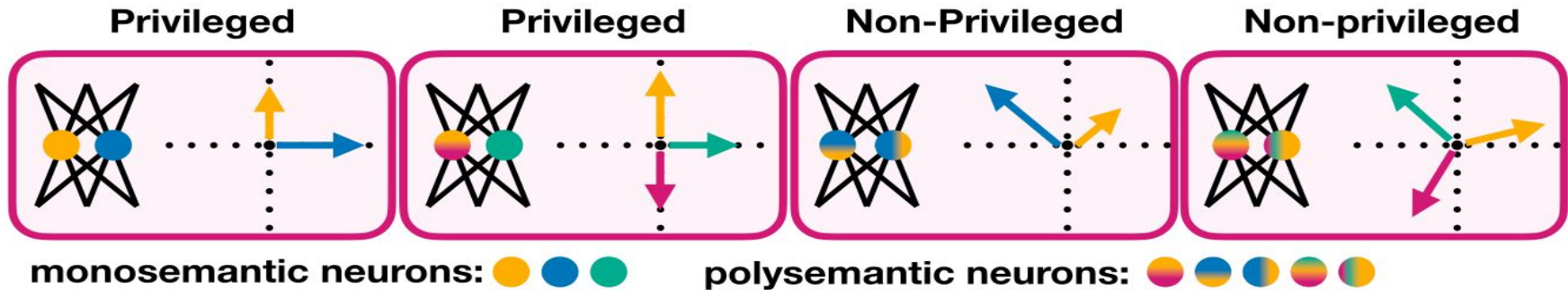
Definition 1: Feature

Features are the fundamental units of neural network representations that cannot be further decomposed into simpler independent factors.

- Features are used by NN as building blocks, aiming to capture concepts underlying the data
- MI aims to uncover the actual representation learned by complex models even if these diverge from human concepts

Nature of features

- Within a neural network representation $h \in \mathbb{R}^n$, the n basis directions are called **neurons**
- For a neuron to be meaningful, the basis directions must functionally differ from other directions, in the representation, forming a **privileged** basis



Monosemantic and polysemantic neurons

- A neuron corresponding to a single semantic concept is called **monosemantic**
- In models like transformers, neurons are often observed to be **polysemantic** (associated with multiple unrelated concepts)

Superposition

Hypothesis 1: Superposition

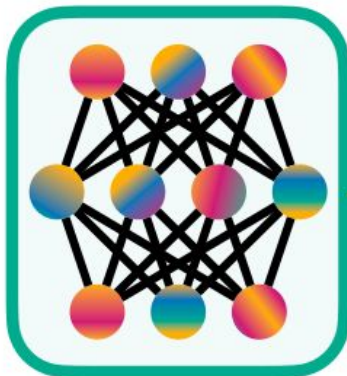
Neural networks represent more features than they have neurons by encoding features in overlapping combinations of neurons.

- NN may simulate computation with more neurons than they possess by allocating each feature to a linear combination of neurons, creating an overcomplete linear basis in the representation space.
- Features are encoded almost in orthogonal directions (they don't interfere each other)

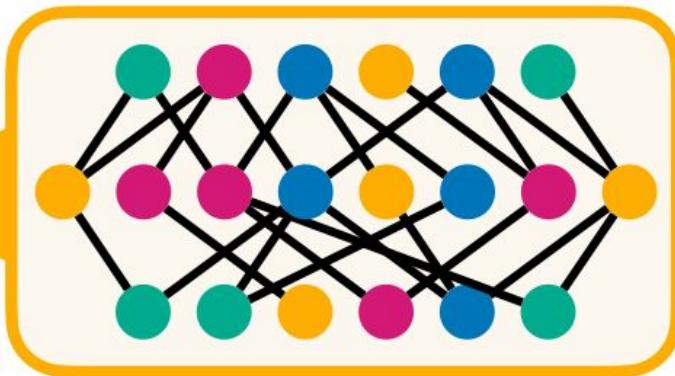
Superposition

Hypothesis 1: Superposition

Neural networks represent more features than they have neurons by encoding features in overlapping combinations of neurons.



Observed model



Hypothetical disentangled model

If not neurons, what are features then?

Hypothesis 2: Linear Representation

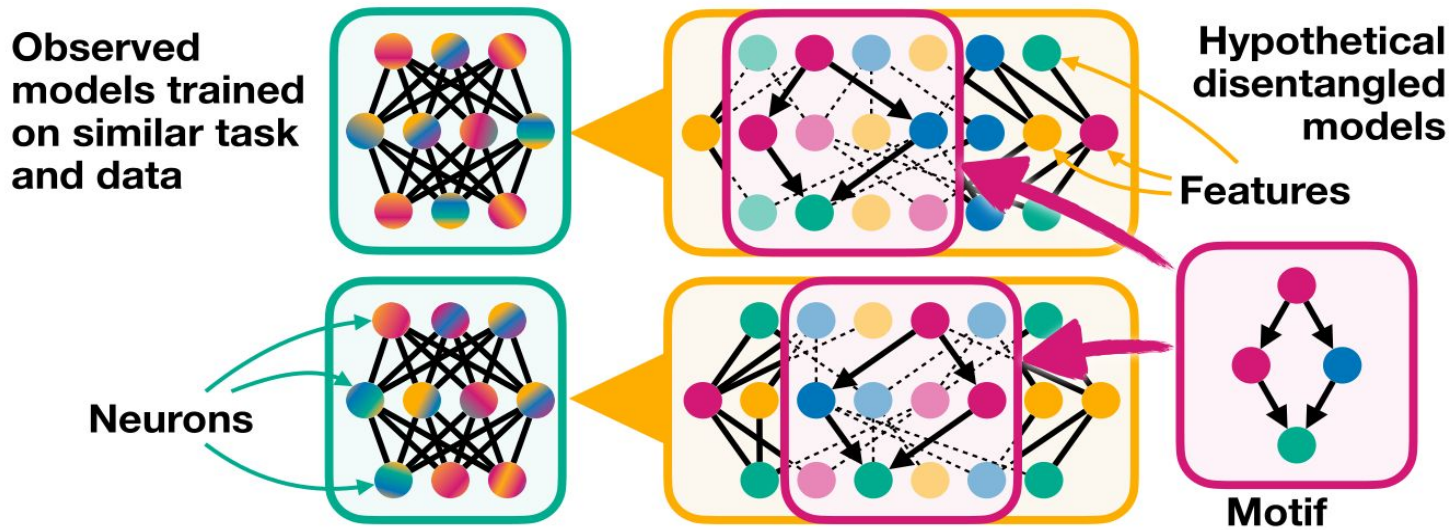
Features are directions in activation space, *i.e.*, linear combinations of neurons.

- Prevalence of linear layers in modern NN architectures.
- Empirical evidence largely supports the linear representation hypothesis in many contexts ([dictionary learning](#), [activation steering](#), [refusal](#), [linear probing](#), [representation engineering](#))
- Building on this linear representation hypothesis, recent works investigated the structural organization of these linear features within the representation space ([Park et al \(2023\)](#), [Park et al. \(2024\)](#))

Circuits as computational primitives

Definition 3: Circuit

Circuits are sub-graphs of the network, consisting of features and the weights connecting them.

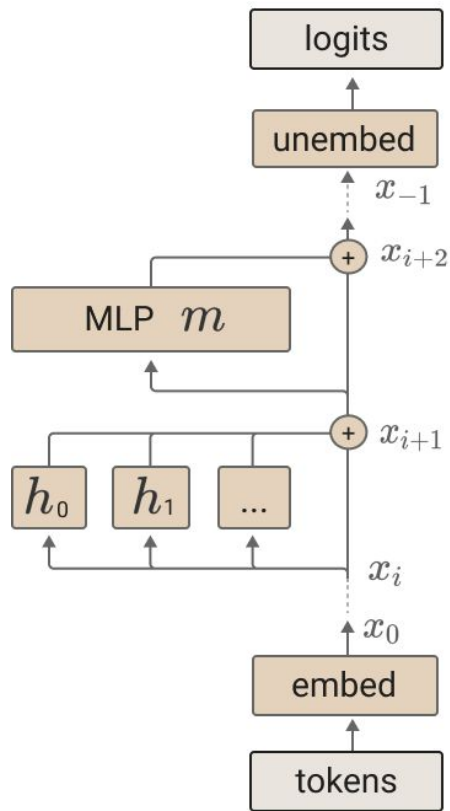


Mathematical framework

A core idea was a simple reformulation of the transformer model, showing that all operations can be understood as summing values over the original tokens.

1. **Residual stream**: the embedding of the token. Each head in the transformer “*reads*” and “*writes*” over this stream to perform computations.
2. Because of linearity, **virtual weights** can be created to represent connections between far-away layers.
3. **Manually** reading these weights allows the discovery of interesting circuits, such as **induction heads**.

Mathematical framework



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m , is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

One
residual
block

Each attention head, h , is run and added to the residual stream.

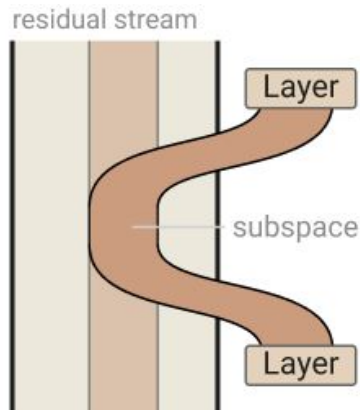
$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

Token embedding.

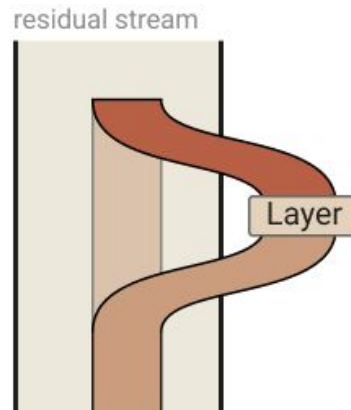
$$x_0 = W_E t$$

Mathematical framework

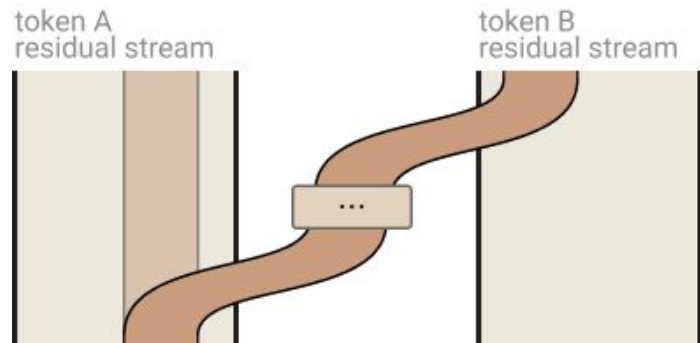
The residual stream is high dimensional, and can be divided into different subspaces.



Layers can interact by writing to and reading from the same or overlapping subspaces. If they write to and read from disjoint subspaces, they won't interact. Typically the spaces only partially overlap.



Layers can delete information from the residual stream by reading in a subspace and then writing the negative version.



Attention heads copy information from the residual stream of one token to the residual stream of another. They typically write to a different subspace than they read from.

Induction heads

For simple models, circuits end up being trivial, replicating either bigram statistics or copying mechanism.

The most interesting finding was the formalization of **induction heads** in larger models, simple circuits that perform a mapping:




$$[x] [y] \dots [x] \rightarrow [y]$$

i.e., looking for previous iterations of the current token and copying the corresponding next-token in output.

Induction heads

Induction Head - Example 1

Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the
Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the
Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the
Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the
Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the
Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the
Mr and Mrs Dursley, of ... such nonsense. Mr Dursley was the

-  Present Token
-  Attention
-  Logit Effect

Induction Head - Example 2

the Potters. Mrs ... the Potters arrived ... the Potters had ... keeping the Potters away; they
the Potters. Mrs ... the Potters arrived ... the Potters had ... keeping the Potters away; they
the Potters. Mrs ... the Potters arrived ... the Potters had ... keeping the Potters away; they
the Potters. Mrs ... the Potters arrived ... the Potters had ... keeping the Potters away; they
the Potters. Mrs ... the Potters arrived ... the Potters had ... keeping the Potters away; they

Induction heads

Successive work by the same team showed that induction heads might be fundamental for **in-context learning**, i.e., the capability of LLMs to solve tasks by generalizing from prompt examples.

Importantly, these ideas can be tested (among others) by simple **ablations** where the circuits are disabled at inference time to record the change in behaviour.

Core methods

Basically two types of tools:

1. Observation:

- a. Probes
- b. Logit lens
- c. Sparse autoencoders

2. Intervention:

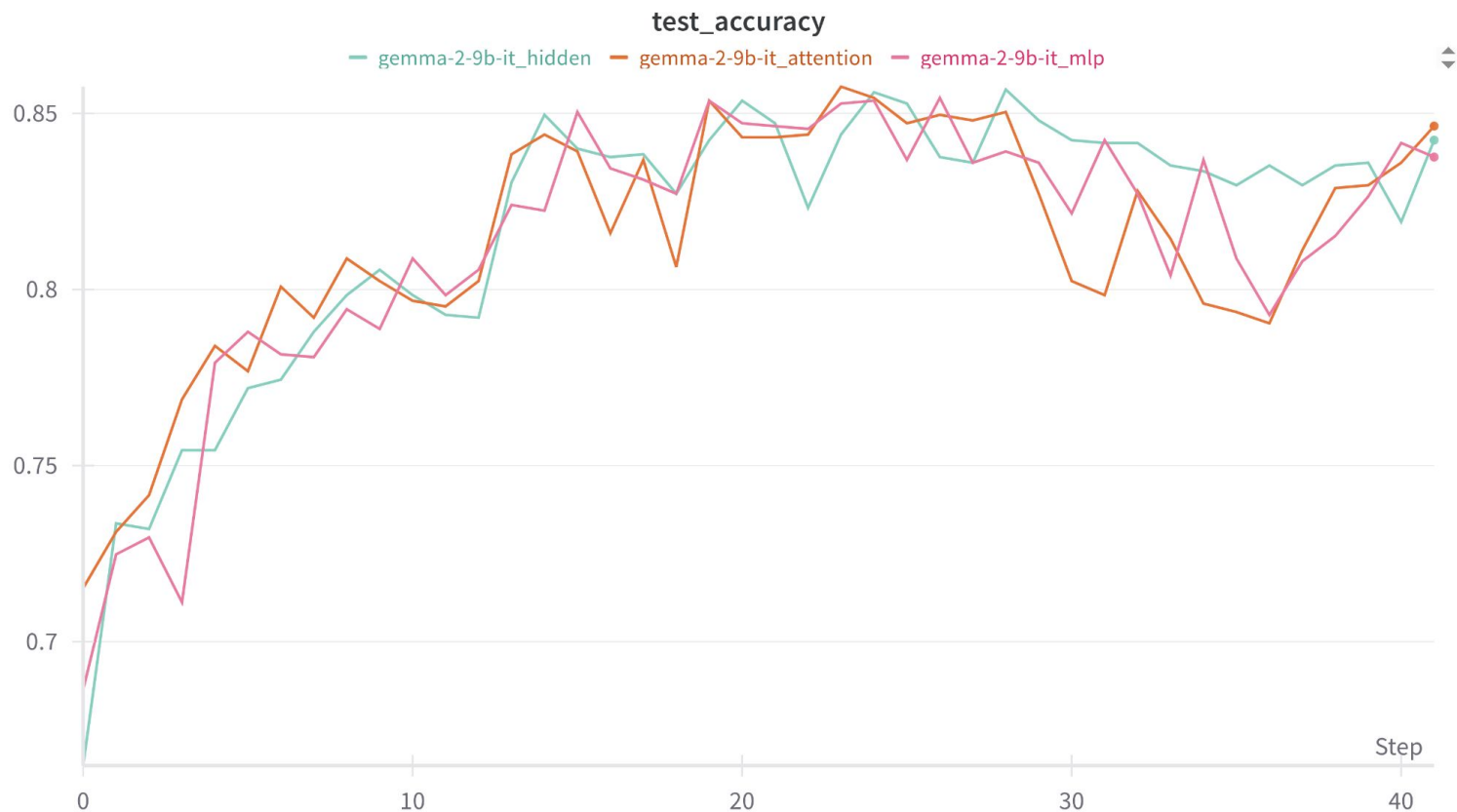
- a. Activation patching
- b. Attribution patching

Probes

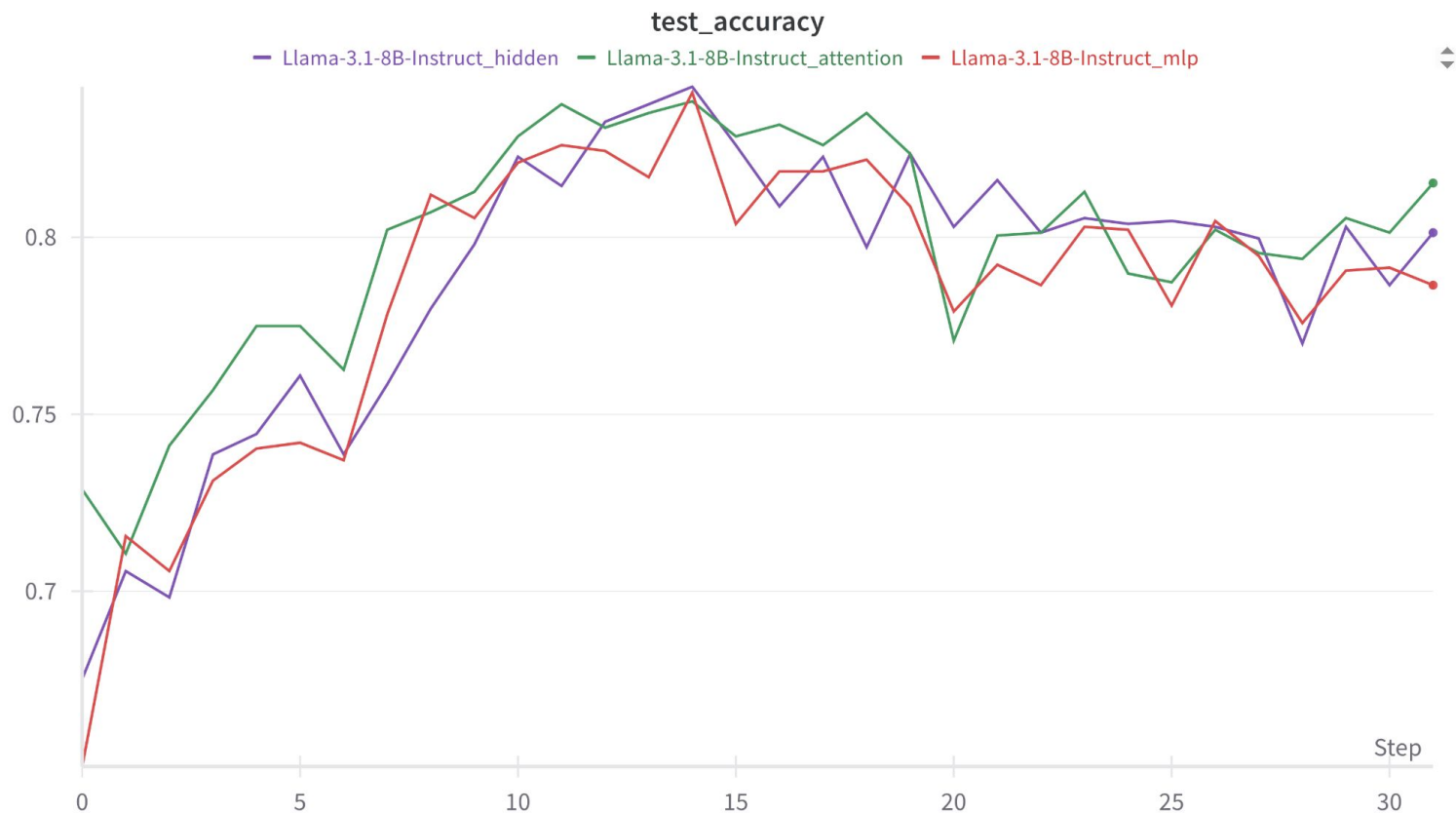
Probing involves training a classifier using the activations of a model and observe the performance of this classifier to deduce insights about model's behavior and internal representations.

Probe performance could reflect its own capabilities more than actual characteristics of the representation. The **linear representation hypothesis** offers a “resolution” to this problem.

gemma-2-9b-it (probe accuracy)



Llama-3.1-8B-instruct (probe accuracy)



Logit lens

<https://alessiodevoto.github.io/LogitLens/>

The core idea behind this method is to apply the model's output layer (unembedding matrix) to the hidden states at each layer of the Transformer

This Allow us to catch how model's internal representations change as the input progresses through the network

Logit lens

```
example = "The quick brown fox jumps over the lazy"  
inputs = tokenizer(example, return_tensors="pt").to(device)
```

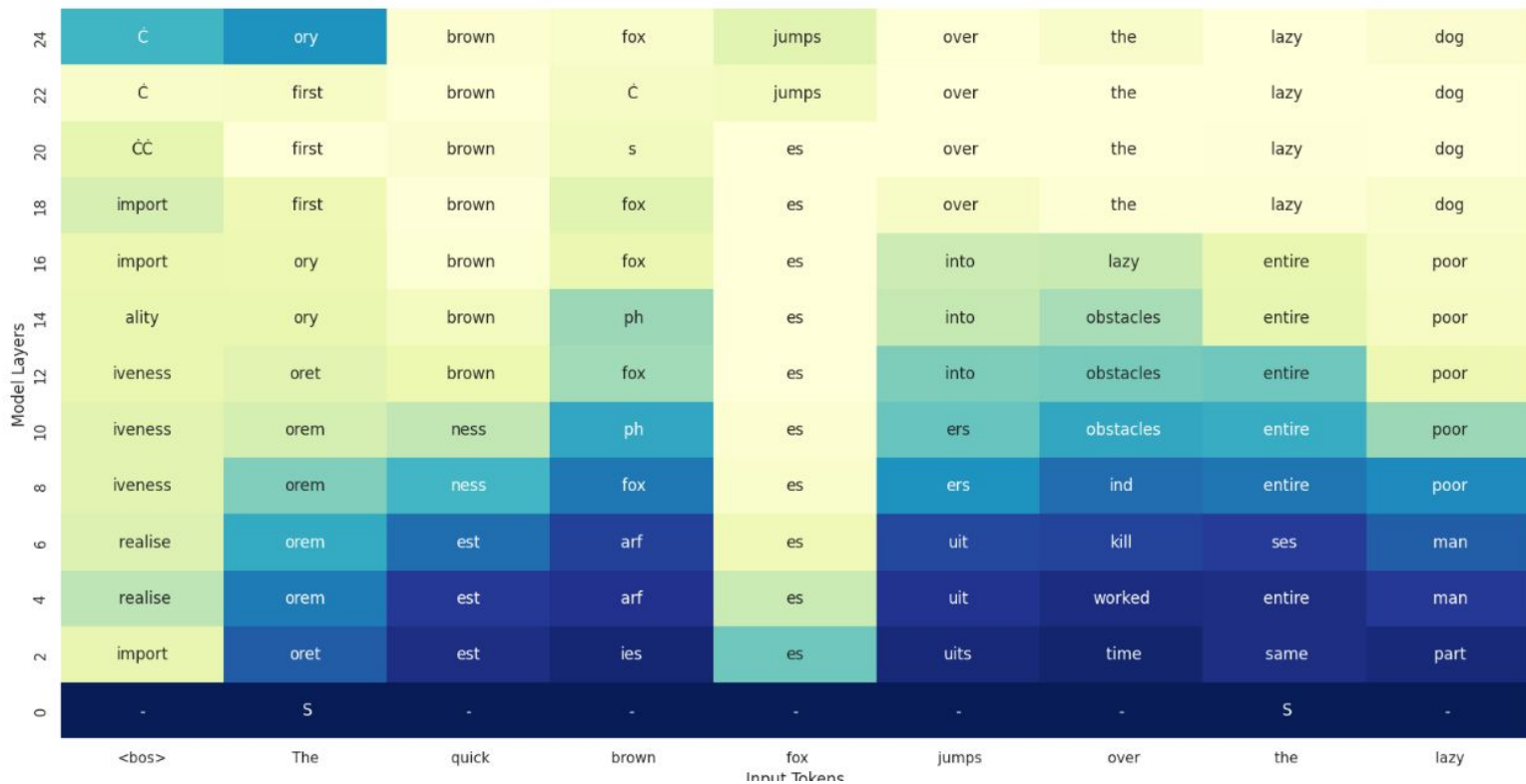
```
['<bos>', 'The', ' quick', ' brown', ' fox', ' jumps', ' over', ' the', ' lazy']
```

```
with torch.no_grad():  
    outputs = model(**inputs, output_hidden_states=True)
```

Logit lens

```
for i, hidden_state in enumerate(hidden_states):  
    # apply the language model head to the hidden states  
    logits = model.lm_head(hidden_state)  
  
    # decode the logits to get the predicted token ids  
    predicted_token_ids = logits.argmax(-1)  
  
    # convert the token ids to tokens  
    predicted_tokens = tokenizer.convert_ids_to_tokens(predicted_token_ids[0])  
    predicted_tokens = cleanup_tokens(predicted_tokens)  
  
    # append the predicted tokens to the list for later  
    logitlens.append(predicted_tokens)  
  
print(f"Layer {i}: {predicted_tokens}")
```

Logit lens



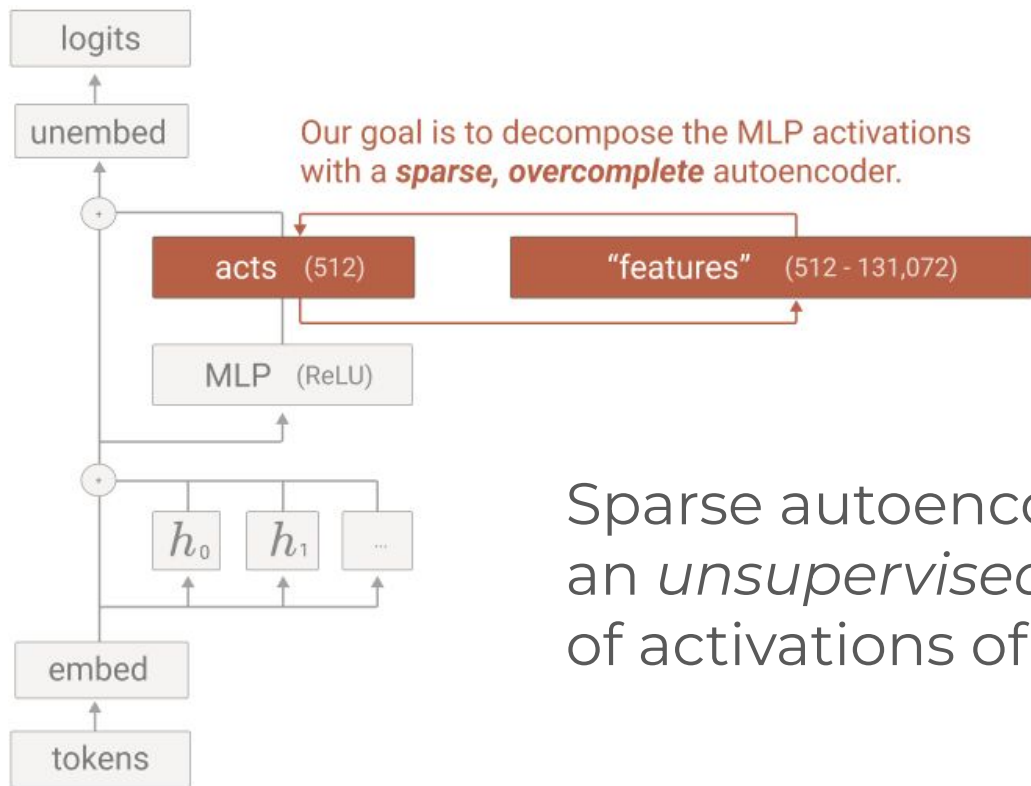
Sparse autoencoder

We stated that neurons are hopelessly polysemantic (“superposition hypothesis”), so there’s the need for bigger and sparser building blocks in order to interpret features.

A **sparse autoencoder** model, learns a sparse decomposition of the activation

$$\mathcal{L}(\boldsymbol{x}) = \underbrace{\|\boldsymbol{x} - \text{SAE}(\boldsymbol{x})\|_2^2}_{\mathcal{L}_{\text{reconstruction}}} + \underbrace{\lambda \|\boldsymbol{a}(\boldsymbol{x})\|_0}_{\mathcal{L}_{\text{sparsity}}}$$

Sparse autoencoder



Sparse autoencoders can be trained in an *unsupervised* way from a collection of activations of the model.

Sparse autoencoder

