

Python for LLM Research

The NNSight Python Library for Mechanistic Interpretability

- bolu ben-adeola



Intro To Mechanistic Interpretability

- 1 Neural networks solve an increasing number of important tasks really well.

Intro To Mechanistic Interpretability

2 It would be at least *interesting*, and probably *important* to understand how.

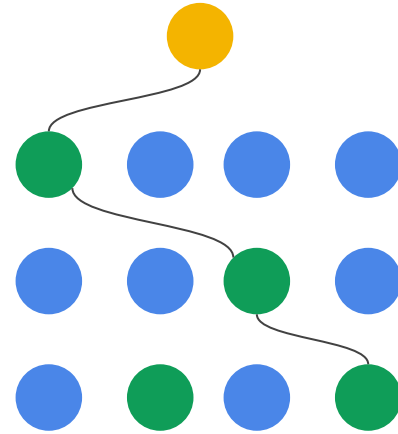
Intro To Mechanistic Interpretability

3 Mechanistic Interpretability (Mech Interp) tackles this problem.

Mech Interp

Mechanistic Model:

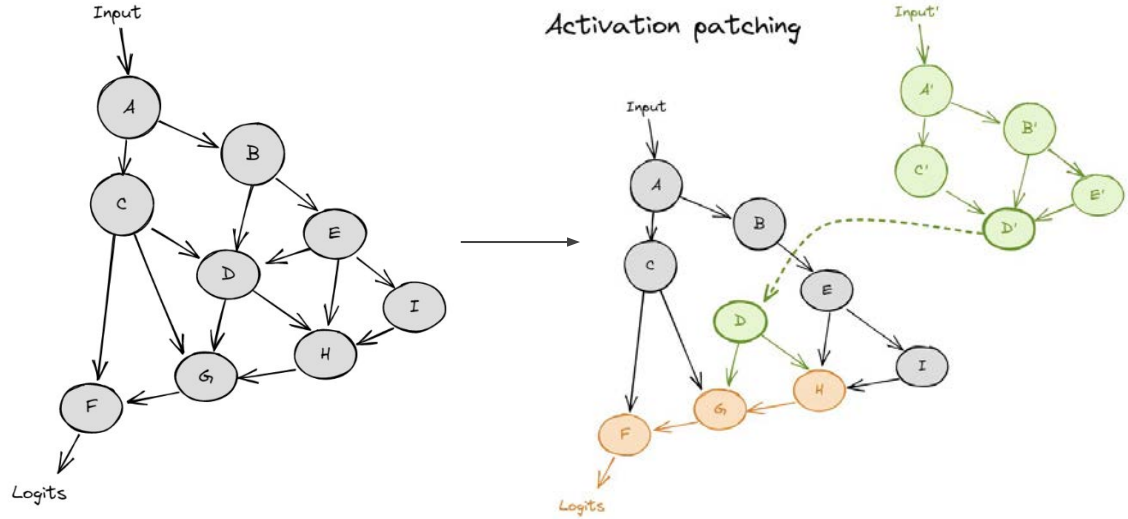
A granular causal model of how components in a network contribute to observed phenomena.



Mech Interp Toolkit: Causal Interventions

Measure the impact of perturbing some node in the computational graph on the final output.

This could be a full replacement, ablative, additive etc

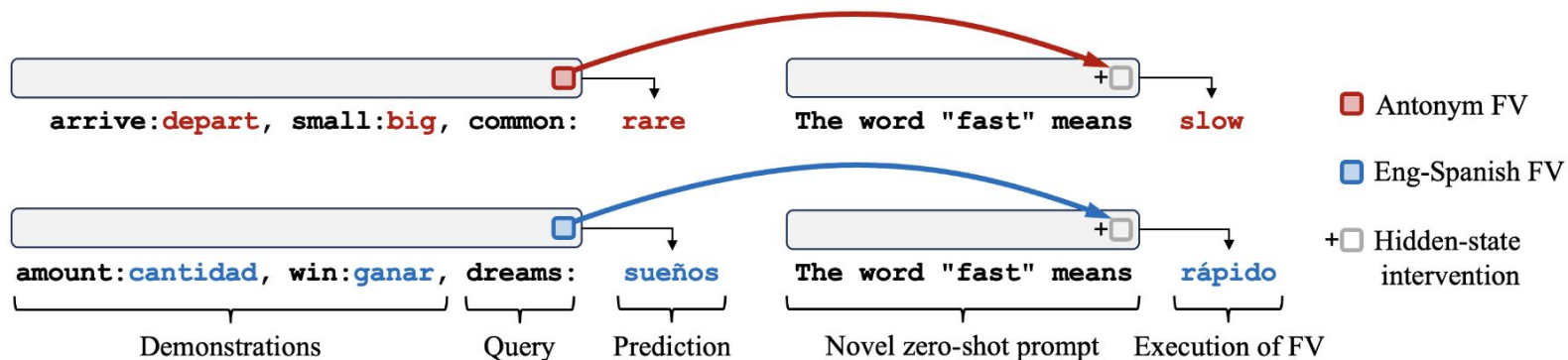


Example Mech Interp Research

FUNCTION VECTORS IN LARGE LANGUAGE MODELS

Eric Todd*, **Millicent L. Li**, **Arnab Sen Sharma**, **Aaron Mueller**,
Byron C. Wallace, and **David Bau**
Khoury College of Computer Sciences, Northeastern University

Activation Vectors



(a) Average Layer Activation

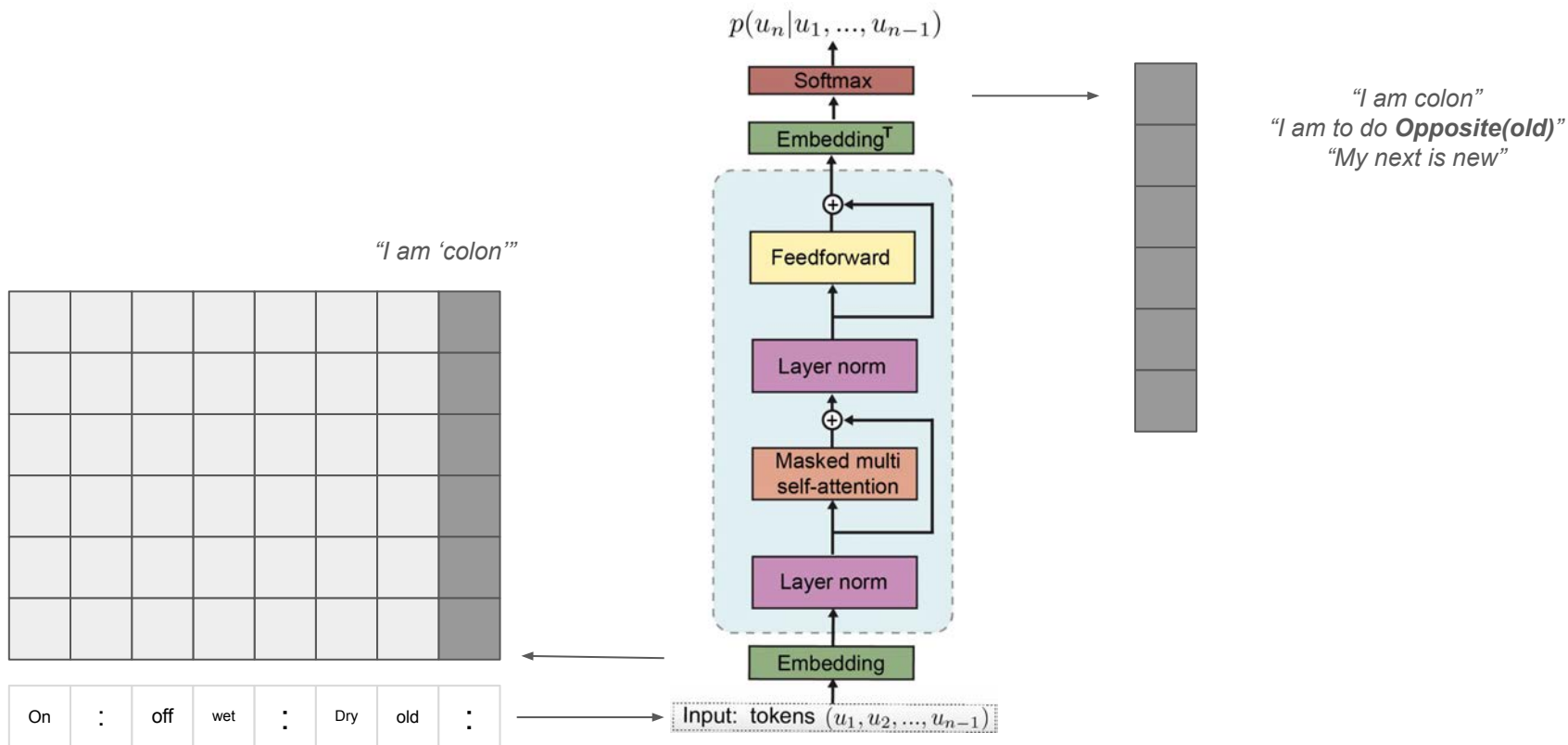


(b) Zero-Shot Intervention

$$\text{simple} + \mathbf{h}_\ell^t = \text{complex}$$

$$\text{encode} + \mathbf{h}_\ell^t = \text{decode}$$

Information Flow



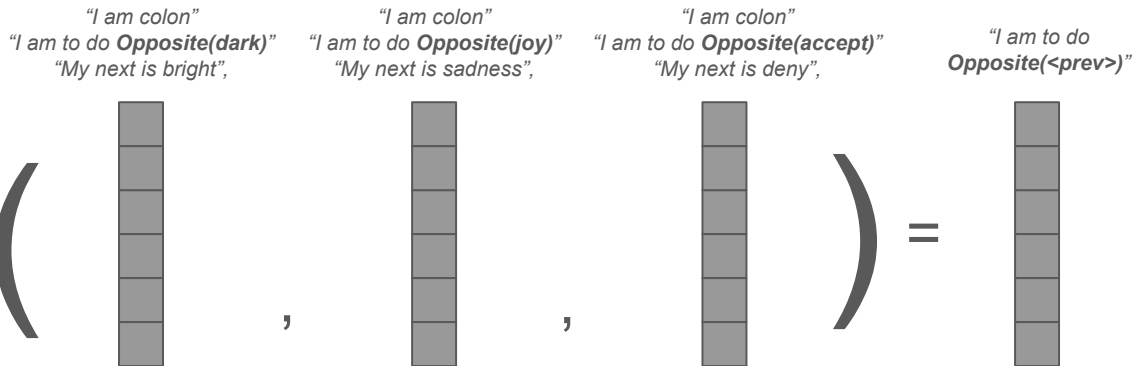
Averaged-out Information Vectors

(a) Average Layer Activation

old:young, vanish:appear, dark:
 awake:asleep, future:past, joy:
 top:bottom, tall:short, accept:

$\bar{\mathbf{h}}_t^t$

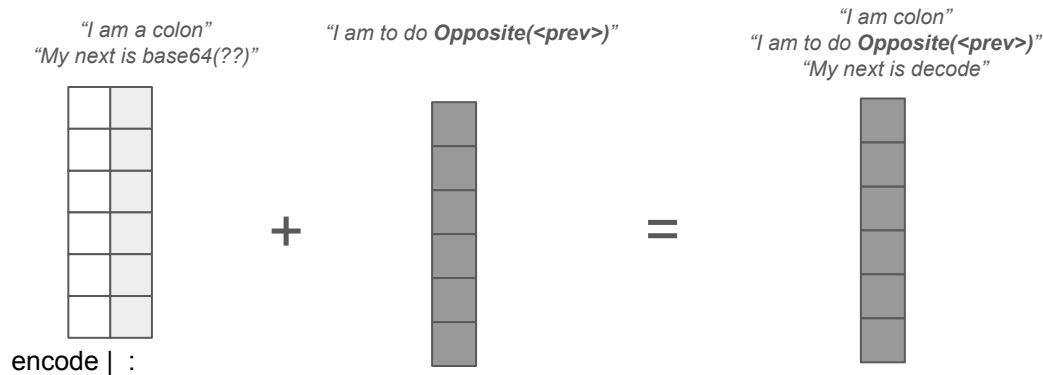
avg



(b) Zero-Shot Intervention

simple + $\bar{\mathbf{h}}_t^t$ = complex

encode + $\bar{\mathbf{h}}_t^t$ = decode



Interpretability Libraries and Packages

NNsight

Interpretable Neural Networks

NNsight (/en.sait/) is a package for interpreting and manipulating the internals of large models

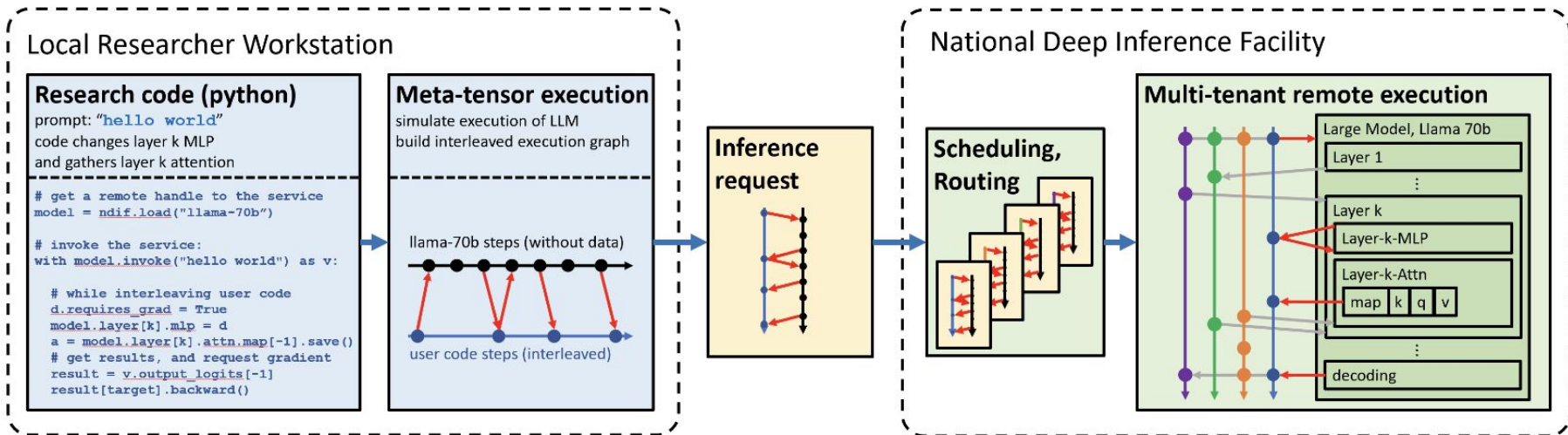
[Get Started](#)[Tutorials](#)[Docs](#)

TransformerLens

(Formerly known as EasyTransformer) `pypi v1.14.0`

A Library for Mechanistic Interpretability of Generative Language Models

NNsight Architecture



Example Intervention

```
1 from nnsight import LanguageModel
2 model = LanguageModel('meta-llama/Llama-2-70b-hf')
3 with model.forward(remote=True) as runner:
4     with runner.invoke('The Eiffel Tower is in the city of ') as invoker:
5         hidden_state = model.layers[10].input[0].save() # save one hidden state
6         model.layers[11].mlp.output = 0 # change one MLP module output
7 print('The model predicts', runner.output)
8 print('The internal state was', hidden_state.value)
```

Anatomy of an Intervention

`nnsight.contexts.Invoker`

`nnsight.contexts.Tracer / nnsight.tracing.Graph`

```
1 from nnsight import LanguageModel
2 model = LanguageModel('meta-llama/Llama-2-70b-hf')
3 with model.forward(remote=True) as runner:
4     with runner.invoke('The Eiffel Tower is in the city of ') as invoker:
5         hidden_state = model.layers[10].input[0].save() # save one hidden state
6         model.layers[11].mlp.output = 0 # change one MLP module output
7 print('The model predicts', runner.output)
8 print('The internal state was', hidden_state.value)
```

`nnsight.tracing.Node`

Model Internal I/O are nodes on the Intervention Graph

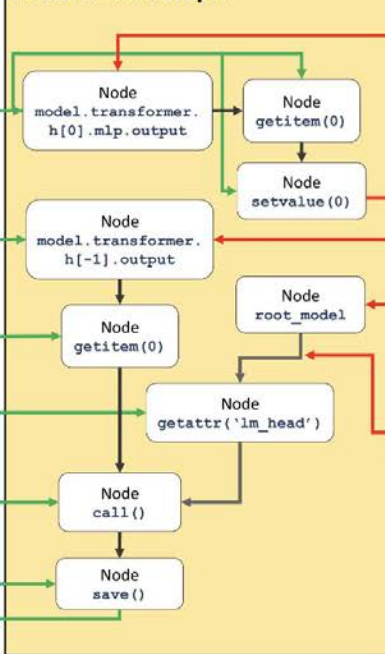
Research code (python)

```
prompt: "hello world"  
Code changes layer 0 MLP output at index 0  
Gathers layer k output at index 0, applies lm_head module  
Saves the result  
-----  
# get a remote handle to the service  
model = nnsight.LanguageModel("huge-model-1T")  
  
with model.forward() as runner:  
    with runner.invoke("hello world"):  
        model.transformer.h[0].mlp.output[0] = 0  
        hidden_states = model.transformer.h[-1].output[0]  
        decoded = model.lm_head(hidden_states).save()  
  
decoded = decoded.value
```

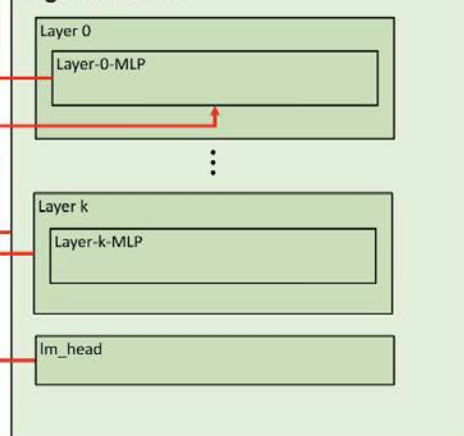
Legend

- Interaction between user code and nodes in the intervention graphs. (Node creation)
- Interaction between nodes in the intervention graph. (Node dependencies)
- Interaction between the local model and nodes in the intervention graph. (Node value injection)

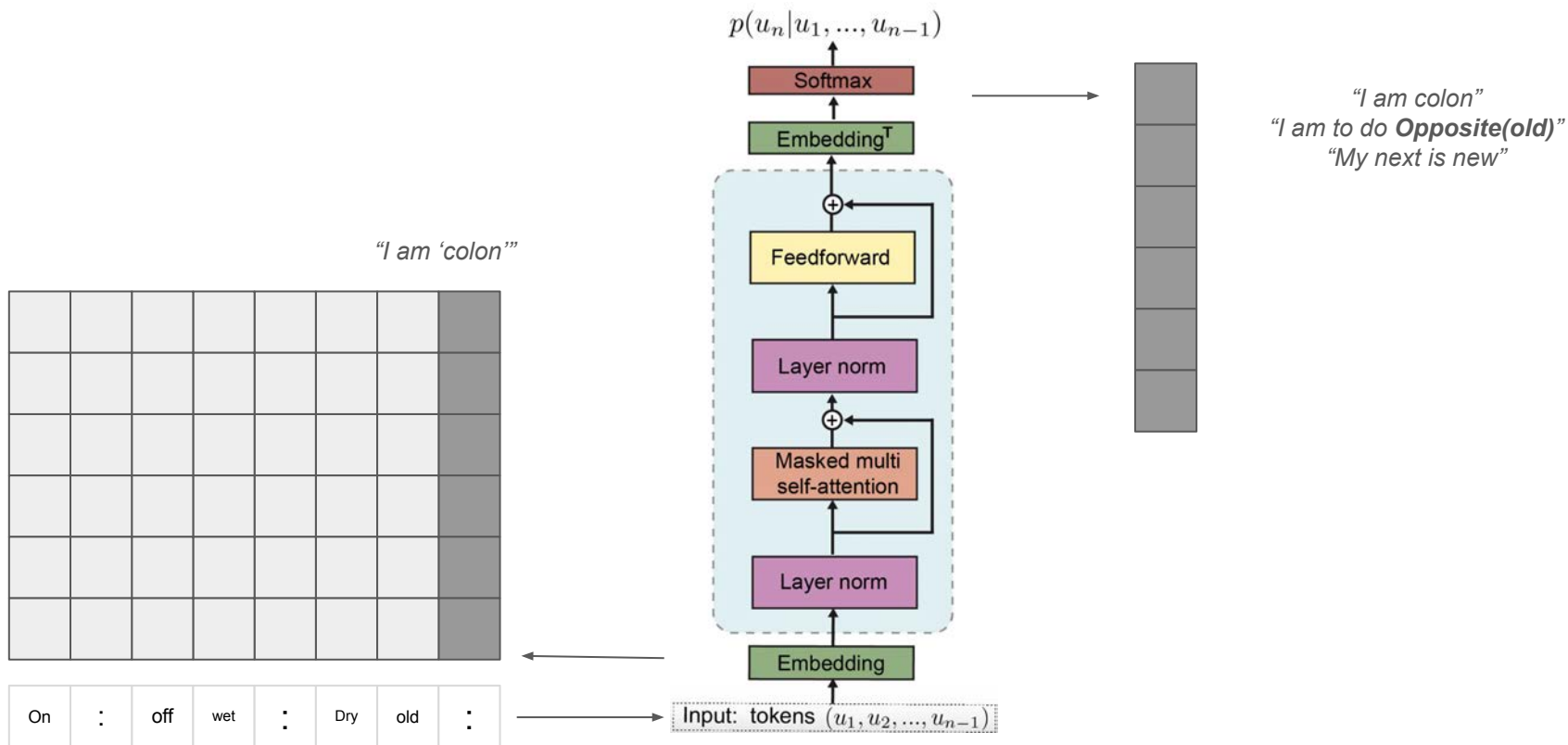
Intervention Graph



huge-model-1T



Information Flow



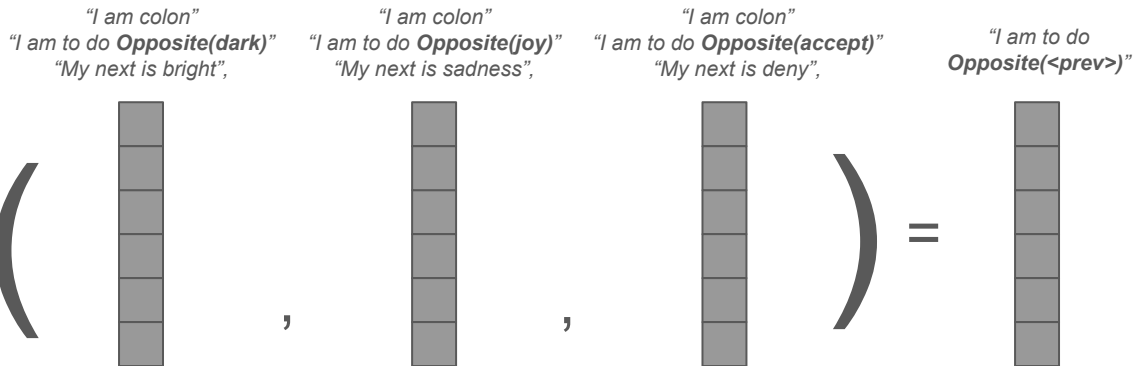
Averaged-out Information Vectors

(a) Average Layer Activation

old:young, vanish:appear, dark:
 awake:asleep, future:past, joy:
 top:bottom, tall:short, accept:

\vec{h}_t

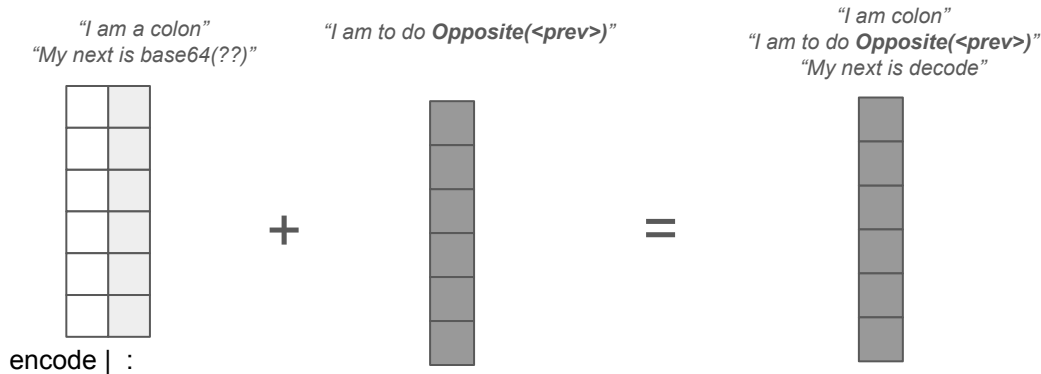
avg



(b) Zero-Shot Intervention

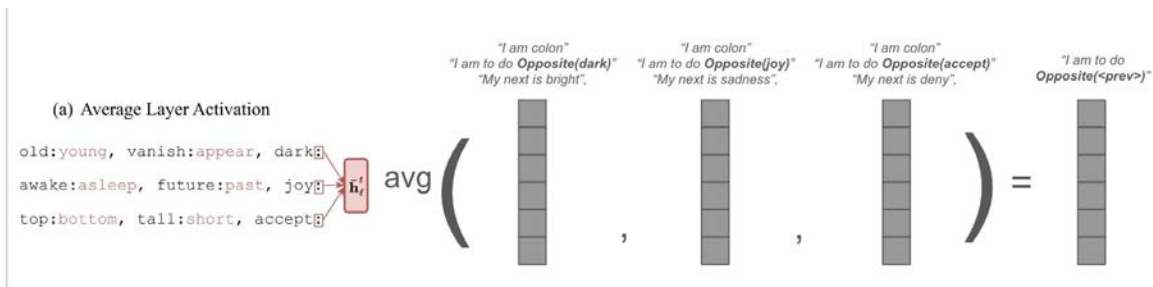
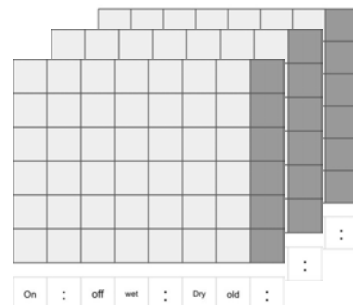
simple + \vec{h}_t = complex

encode + \vec{h}_t = decode



Getting the average activation Vector

```
1 sequence_position = -1
2 layer = 8
3 with model.forward(remote=True) as runner:
4     with runner.invoke(opposites_dataset) as invoker:
5         # shape: batch, sequence_length, d_model
6         hidden_states = model.transformer.h[layer].output[0]
7         # shape: batch, d_model
8         hidden_states_at_last_pos =[:, sequence_position]
9         # shape: d_model
10        average_hidden_state_at_last = hidden_states_at_last_pos.mean(dim=0).save()
11
12 h = average_hidden_state_at_last.value
```



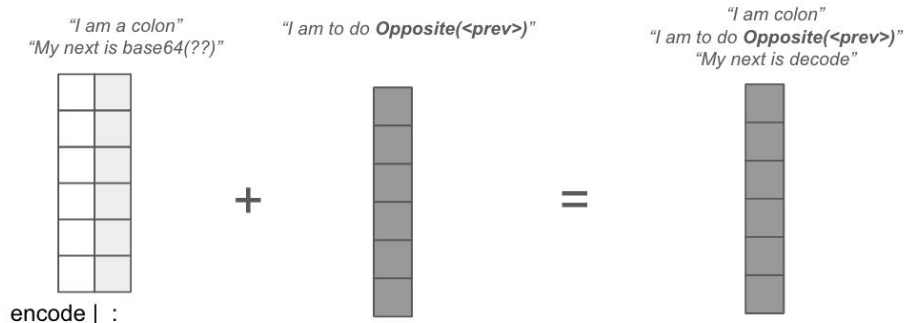
Adding the average to One-shot Activation Vectors

```
1 with model.forward(remote=REMOTE) as runner:
2
3     # First, run a forward pass where we don't intervene, just save token id completions
4     with runner.invoke(zero_shot_dataset.prompts) as invoker:
5         token_completions_zero_shot = model.lm_head.output[:, -1].save()
6
7     # Next, run a forward pass on the zero-shot prompts where we do intervene
8     with runner.invoke(zero_shot_dataset.prompts) as invoker:
9         # Add the h-vector to the residual stream, at the last sequence position
10        hidden_states = model.transformer.h[layer].output[0]
11        hidden_states[:, sequence_position] += h
12        # Also save completions
13        token_completions_intervention = model.lm_head.output[:, -1].save()
14
15 compare(token_completions_zero_shot, token_completions_intervention)
16
```

(b) Zero-Shot Intervention

simple[] + $\begin{bmatrix} -1 \\ \mathbf{h}_i \end{bmatrix}$ = complex

encode[] + $\begin{bmatrix} -1 \\ \mathbf{h}_i \end{bmatrix}$ = decode



Results With Average Activation Vector

| Prompt | Model's completion (no intervention) | Model's completion (intervention) | Correct completion |
|--------------|--------------------------------------|-----------------------------------|--------------------|
| minimum -> | ' minimum' | ' maximum' | ' maximum' |
| arrogant -> | ' arrogant' | ' arrogant' | ' humble' |
| inside -> | ' inside' | ' outside' | ' outside' |
| reject -> | ' reject' | ' reject' | ' accept' |
| invisible -> | ' invisible' | ' invisible' | ' visible' |
| victory -> | ' victory' | ' victory' | ' defeat' |
| up -> | ' up' | ' down' | ' down' |
| open -> | ' open' | ' closed' | ' closed' |
| under -> | ' under' | ' under' | ' over' |
| inside -> | ' inside' | ' outside' | ' outside' |
| solid -> | ' solid' | ' solid' | ' liquid' |
| optimist -> | '\n' | ' optim' | ' pessimist' |
| noisy -> | ' noisy' | ' noisy' | ' quiet' |
| guilty -> | ' guilty' | ' guilty' | ' innocent' |
| answer -> | ' yes' | ' answer' | ' question' |
| on -> | ' I' | ' on' | ' off' |
| junior -> | ' senior' | ' senior' | ' senior' |
| loose -> | ' loose' | ' loose' | ' tight' |
| introduce -> | ' introduce' | ' introduce' | ' remove' |
| innocent -> | ' innocent' | ' guilty' | ' guilty' |

Resources

Getting started with the NNSight Library: <https://nnsight.net/>

Learning Mech Interp: <https://www.arena.education/>