

Inspect MLP Responses and Feature Geometry

Gavin · AI-assisted course design

Contents

1	Goal	3
2	Part 1 — Trace a new MLP input	3
3	Part 2 — Test a dictionary under different feature combinations	4
4	Part 3 — Separate a distributed code from superposition	4
5	Part 4 — Optional standard-library implementation	5
6	Part 5 — Optional contrasting-prompt observation	5
6.1	Choose the documented specimen	5
6.2	Local runner and two observation phases	6
6.3	Fix prompts and a position before looking at activations	7
6.4	Name and verify the capture boundary	7
6.5	Explore, freeze a prediction, then check	8
6.6	Explain the first-block control	8
7	Analytically derived checkpoints	9
8	Submit evidence and an explanation	9
9	More Learning	9

1 Goal

Trace how hidden activations become a residual contribution, test a small feature dictionary, and practice limiting an interpretation to its evidence. An optional extension compares actual MLP responses across contrasting prompts in a trained Transformer.

Prerequisite: Lesson 2.3, including its two numerical constructions.

Time: 60–75 minutes for the hand-worked core. Allow additional setup and analysis time for the optional model observation.

Core requirements: paper or a text editor. Parts 1–3 need no account, API, network, model download, GPU, paid compute, or machine-learning package. Optional toy code uses only ordinary Python on a CPU.

Expected artifact: a short `submission.md` or equivalent note with predictions, calculations, a sketch, explanations, and limitations. If you run code, include the actual output and environment details. The provided analytical answers are not evidence of software execution.

2 Part 1 — Trace a new MLP input

Reuse the lesson’s matrices and biases, but use a fresh input:

$$x = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}, \quad W_{\text{in}} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 1 \end{bmatrix}, \quad b_{\text{in}} = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 1 \end{bmatrix},$$
$$W_{\text{out}} = \begin{bmatrix} 1 & 0 & -1 & 1 \\ 0 & 1 & 0 & -2 \\ 1 & -1 & 2 & 0 \end{bmatrix}, \quad b_{\text{out}} = \begin{bmatrix} 0 \\ 1/2 \\ 0 \end{bmatrix}.$$

As in the lesson’s toy, there is no normalization: $u = x$. Apply $z = W_{\text{in}}x + b_{\text{in}}$, $h = \text{ReLU}(z)$, $\Delta = W_{\text{out}}h + b_{\text{out}}$, and $y = x + \Delta$.

Before calculating, predict which units will be positive and whether the first residual coordinate will increase, decrease, or remain unchanged. Then record all four intermediate vectors and their shapes. Expand Δ into the separate column contributions $h_j c_j$ plus the output bias. Keep zero contributions visible.

For an immediate score, choose $s = y_1 + y_3$. Reset to baseline for each intervention:

1. Set only h_1 to zero after ReLU.
2. Set only h_3 to zero after ReLU.
3. Set only h_2 to zero after ReLU.

Predict the new y and s before computing them. Which equally activated units have unequal effects on the score? Explain using their output columns rather than their activation values alone. Does the zero effect of intervention 3 establish that unit 2 is irrelevant on other inputs?

These are activation replacements, not weight changes. Because this toy has only an affine output map after the replacement, its immediate effect can be calculated exactly by subtracting one contribution. In a deeper model, later computations must be rerun.

3 Part 2 — Test a dictionary under different feature combinations

Use the hand-designed dictionary

$$D = \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix}, \quad r = Da, \quad \hat{a} = \text{ReLU}(D^T r).$$

The three entries of a are nonnegative feature strengths. They are known generating variables in this construction. The two entries of r are representation coordinates, not two semantic feature labels. No weights are learned in this part.

Sketch the three columns as arrows and verify their lengths and pairwise dot products. Predict which of these inputs will reconstruct exactly:

- $a = [0, 2, 0]^T$;
- $a = [1, 0, 1]^T$;
- $a = [1, 2, 0]^T$;
- $a = [1, 1, 1]^T$.

For each input, calculate r , the pre-ReLU readout $D^T r$, the reconstruction $\hat{a}$, and squared error $\|\hat{a} - a\|_2^2$. Do not hide the negative entries before explaining what ReLU removes.

Then answer:

1. Why does singleton recovery not contradict the fact that three independent directions cannot fit in two dimensions?
2. Which failure could potentially be improved by choosing a different decoder, and which pair of inputs creates an unavoidable ambiguity for any decoder receiving only r ?
3. If inputs are usually singletons but occasionally dense, what additional information is needed to calculate average reconstruction error?
4. Is a sparse feature-strength vector necessarily a sparse representation vector? Use a displayed example.

For question 3, specify probabilities and strength distributions; merely saying “usually sparse” is insufficient to calculate an expectation. You do not need to invent or fit such a distribution.

4 Part 3 — Separate a distributed code from superposition

Use the orthogonal dictionary with columns

$$p = \frac{1}{\sqrt{2}}[1, 1]^T, \quad q = \frac{1}{\sqrt{2}}[1, -1]^T.$$

Encode feature strengths $(2, 1)$ as $r = 2p + q$. Recover them with the two dot products. Explain why observing that the first coordinate responds to both generating features is insufficient evidence of

superposition. In this example, the complete representation has enough independent directions for the two features.

Write two short claims about the triangle construction: one that your calculations establish and one that would require observing a trained network. Avoid turning “this geometry can work on these inputs” into “language models learned this exact geometry.”

5 Part 4 — Optional standard-library implementation

Implement checked matrix–vector multiplication, ReLU, vector addition, dot products, and squared error. Reject incompatible shapes instead of silently truncating arrays. Preserve immutable baseline values or copies so one intervention cannot change another case’s starting point.

Verify every Part 1 vector against the hand trace. For Part 2, report numerical error against the analytical answers with a declared tolerance, such as absolute error 10^{-12} for these small double-precision calculations. Near-zero roundoff from $\sqrt{3}$ should not be mislabeled a newly discovered feature.

The downloadable toy runner implements these constructions using only the standard library. Save it locally and run `python3 trace_mlp_geometry.py` after making your predictions. Save the program, invocation, Python version, and actual JSON output. A passing test checks implementation against this construction; it does not establish that a trained language model uses the same representation.

6 Part 5 — Optional contrasting-prompt observation

6.1 Choose the documented specimen

Reuse the local artifact from Lab 05, if available: EleutherAI/Pythia-14M at revision 94f7c35d5e9f2e9bac8ca839329f505b4d007d5d. The publisher’s immutable commit specifies six blocks, residual width 128, MLP width 512, GELU, and parallel residuals. The model card documents a February 2026 checkpoint replacement; do not substitute an unrecorded floating revision.

If downloading for the first time, allow only `config.json`, `tokenizer.json`, `tokenizer_config.json`, `special_tokens_map.json`, `generation_config.json`, and `model.safetensors` from that revision. The weights alone occupy 28,143,920 bytes, roughly 28.1 MB; inspect the total including tokenizer and configuration files before downloading. Do not fetch optimizer state, every training checkpoint, `.bin` weights, or custom Python files. Dependency installation can require much more disk space than the model weights.

The following requirements describe the supplied runner and any independent implementation. No expected trained-model activation values are supplied here. Use CPU float32 computation, evaluation mode, inference mode without gradients, and `use_cache=False`. Use built-in GPTNeoXForCausalLM, safetensors weights, a fast tokenizer, and `trust_remote_code=False`. After the bounded download, load locally with `local_files_only=True`. No paid inference service or API is needed.

Pin and record compatible, tested versions of Python, PyTorch, Transformers, tokenizers, safetensors, and the download client. The inspected source reference is Transformers v4.57.1. Recheck module boundaries if using a different version; do not enable remote custom code to repair a loading failure.

6.2 Local runner and two observation phases

Reuse the separate environment, pinned requirements and verified artifact directory from Lab 05. Save the MLP observation runner and its sibling specimen helper together. Both must be in the same directory. The tested versions and CPU eager backend match Lab 05; its download plan and six-file hashes apply here too. This runner requires the already downloaded artifacts and loads them offline.

First save your block 0 and block 2 structural predictions in `my-structure.txt`. Then run only the discovery pairs:

```
.venv-residual/bin/python inspect_mlp_responses.py discover --artifacts pythia-artifacts --structural-predictions my-structure.txt --output mlp-discovery
```

Read `discovery.json` and its complete captures. The runner selects three indices using the stated discovery contrast and tie-break rule. It also creates `predictions-template.json`. Copy that template to `my-predictions.json`. Keep its selected indices unchanged; fill the three directions entries with -1, 0, or 1, then write your reasoning and `structural_predictions`. These are your hypotheses, not automatic labels. Save this file before evaluating held-out prompts.

Only then run:

```
.venv-residual/bin/python inspect_mlp_responses.py reveal --artifacts pythia-artifacts --discovery mlp-discovery --predictions my-predictions.json --output mlp-heldout
```

The reveal phase validates the frozen discovery selection, writes your predictions to its output directory before inference, and evaluates only pairs 3 and 4. It retains every selected-unit value and contrast sign, including failures, without reselection. Manifest files record versions, specimen hashes and settings. Full prompt/token records and signed preactivation, post-activation and MLP output arrays remain in the local JSON files. Nonempty result directories are never overwritten. The phase split supports an honest sequence; it cannot prove that a person never inspected held-out data elsewhere.

For the additional surface-pattern challenge below, write exactly two new contexts as a JSON array in `challenge-contexts.json`. Copy your prediction file, preserve selected indices, and record your predicted directions and rationale for this new contrast. Both contexts receive the same suffix automatically; the runner limits each to 256 tokens.

```
.venv-residual/bin/python inspect_mlp_responses.py challenge --artifacts pythia-artifacts --discovery mlp-discovery --predictions challenge-predictions.json --contexts challenge-contexts.json --output mlp-challenge
```

The first context minus the second defines the contrast. This is a user-designed challenge, not another selection set or an automatic validation claim.

No expected trained-model activations are published. Maximum boundary errors, hook cleanup, repeatability and same-ID checks are recorded from the actual machine run. A failed structural check should prompt inspection before interpretation.

6.3 Fix prompts and a position before looking at activations

Use these four code/prose pairs as a small exploratory set. Each prompt ends in the identical suffix `\nSummary:.` Here `\n` denotes one actual newline, not two literal characters. These examples are original prompts, not known activation triggers.

1. Discovery pair 1

- Code context: `x = 2 + 3\nprint(x)`
- Prose context: Mira added two and three and wrote the total.

2. Discovery pair 2

- Code context: `items = [1, 2, 3]\nlen(items)`
- Prose context: Mira counted three items on the desk.

3. Held-out pair 3

- Code context: `names = ['Ada', 'Lin']\nnames[0]`
- Prose context: Mira chose Ada from a list of two names.

4. Held-out pair 4

- Code context: `word = 'blue'\nword.upper()`
- Prose context: Mira rewrote the word blue in capital letters.

Use the prose text exactly as written after its label, including its final period. Append the common suffix to every code and prose context.

Tokenize each complete prompt with `add_special_tokens=False`, no chat template, and no truncation. Run prompts individually to avoid padding. Save exact text, token IDs, decoded pieces, and zero-based positions. Choose the final token of the common suffix; verify that its token ID is identical across all eight prompts. If it is not, resolve the alignment and record the change before observing activations.

Matching that final ID controls one variable. It does not equalize prefix length, token positions, vocabulary, punctuation, or task demands. Record these differences as possible explanations. Do not call the pairings a complete semantic control.

Before measuring, predict:

1. Should the final-position MLP activation in zero-indexed block 0 differ between prompts with the same final token ID?
2. Could zero-indexed block 2 differ even though its MLP still operates separately at each position?
3. If a unit looks code-selective on the discovery pairs, what held-out result would weaken that interpretation?

6.4 Name and verify the capture boundary

Inspect GPTNeoXMLP in the linked implementation. At the chosen block, capture:

- z : output of `mlp.dense_h_to_4h`, before GELU;
- h : input to `mlp.dense_4h_to_h`, after GELU;
- Δ : output of the whole `mlp`, before residual addition.

An observation-only forward pre-hook on `dense_4h_to_h` exposes h . It must leave inputs unchanged and return `None`. Clone recorded tensors, clear captured values between runs, and remove hook handles

in a finally block, including when a check fails. Verify the actual return types rather than assuming every module returns a tuple.

Use a discovery prompt for the initial hook checks and repeatability tests; do not inspect held-out activations yet. Capture block 0 for the structural check and block 2 for the comparison; do not search all layers for the most flattering result. Expect shapes $(1, n, 512)$ for z, h and $(1, n, 128)$ for Δ . Check $h = \text{GELU}(z)$ using the model’s actual activation function, and reconstruct Δ with the stored output map and bias. Report maximum absolute discrepancies and declared tolerances, not just a pass label. Repeat one unchanged prompt and report the largest difference between runs.

GELU can produce negative values. Preserve signed activations; do not silently replace it with ReLU or treat all nonzero coordinates as equivalent active features.

6.5 Explore, freeze a prediction, then check

For block 2 and the aligned final position, calculate each unit’s mean paired discovery contrast:

$$\delta_j = \frac{1}{2} \sum_{k=1}^2 (h_{j,\text{code},k} - h_{j,\text{prose},k}).$$

Choose the three units with largest $|\delta_j|$, resolving ties by smaller zero-based unit index. Keep their signs. This is an exploratory selection from 512 candidates, not a significance test.

Before opening the held-out activations, save the selected indices, discovery values, and predicted contrast direction for each. Evaluate those same indices on pairs 3 and 4 without reselection. Report each pair’s values and signed difference, including failures. If held-out observations were viewed while selecting units, relabel the analysis exploratory and reserve genuinely fresh prompts for a later check.

Next write one additional contrast for a selected unit that challenges a surface explanation: for example, code-like punctuation in ordinary prose, or the same simple operation described without programming syntax. Save your prediction before running it. One such check is a useful challenge, not a validation suite.

Report one observation, two plausible explanations, and a test that would distinguish them. Do not claim that three selected coordinates are three discovered semantic features, that an activation contrast proves superposition, or that this tiny checkpoint represents larger models’ behavior. Real-model ablation is not required here; the toy already provided a fully accountable intervention.

6.6 Explain the first-block control

In this checkpoint, the initial residual state comes from token embeddings, and rotary position handling occurs inside attention. Parallel residuals make block 0’s MLP read a normalized version of that initial state, before the same block’s attention contribution is added. With the same target token ID and evaluation settings, its MLP therefore receives the same target vector across these prompts. This predicts matching target activations within numerical tolerance. A discrepancy should trigger a boundary, tokenization, or implementation check before a semantic story.

Block 2 reads states that preceding blocks have already contextualized. Its activations may differ. This comparison tests your understanding of the actual wiring as well as your hooks. Neither matching nor differing values have been supplied as observed results.

7 Analytically derived checkpoints

i Core solutions

For Part 1, $z = h = [2, 0, 2, 0]^T$. The nonzero column contributions are $2c_1 = [2, 0, 2]^T$ and $2c_3 = [-2, 0, 4]^T$. Thus $\Delta = [0, 1/2, 6]^T$ and $y = [2, 1/2, 7]^T$; baseline $s = 9$.

- Zero h_1 : $y = [0, 1/2, 5]^T$ and $s = 5$.
- Zero h_3 : $y = [4, 1/2, 3]^T$ and $s = 7$.
- Zero h_2 : baseline is unchanged, since $h_2 = 0$ on this input.

Units 1 and 3 both activate at two, but their score contributions are four and two. Unit 2's zero effect is conditional on this input and intervention.

For Part 2:

- $[0, 2, 0]$: $r = [-1, \sqrt{3}]^T$; readout before ReLU $[-1, 2, -1]^T$; reconstruction $[0, 2, 0]^T$; squared error 0.
- $[1, 0, 1]$: $r = [1/2, -\sqrt{3}/2]^T$; readout $[1/2, -1, 1/2]^T$; reconstruction $[1/2, 0, 1/2]^T$; squared error $1/2$.
- $[1, 2, 0]$: $r = [0, \sqrt{3}]^T$; readout $[0, 3/2, -3/2]^T$; reconstruction $[0, 3/2, 0]^T$; squared error $5/4$.
- $[1, 1, 1]$: $r = [0, 0]^T$; readout and reconstruction are zero; squared error 3.

Singleton recovery uses a restricted input family. All-one and all-zero inputs give the same representation, so no alternative decoder can distinguish them from r alone. Decoder improvements for selected mixtures do not remove this ambiguity. B alone supplies an example of sparse feature strengths with two nonzero representation coordinates.

For Part 3, $r = [3/\sqrt{2}, 1/\sqrt{2}]^T$. Its projections onto p and q recover two and one. The code is distributed but needs no overcomplete dictionary.

8 Submit evidence and an explanation

Include your original predictions, corrected MLP trace, three intervention outcomes, dictionary calculations, sketch, and distributed-code comparison. State explicitly that the numerical constructions were hand-designed.

For executed toy code, add its source and real output. For optional trained-model work, include checkpoint revision, dependency versions, settings, token records, hook boundaries, shape and numerical checks, discovery selections, predictions made before held-out inspection, complete held-out values, and limitations. Save the full final-position activation vectors so the selected-unit report can be checked against them.

A complete hand-worked core is a valid submission. After this Lab, return to Lab 04, Part 2 to connect the MLP's internal calculation to the original Transformer's architecture.

9 More Learning

- Lesson 2.3 supplies the distinction between units, features, and superposition.

- Toy Models of Superposition studies learned toy representations. Identify the training setup before comparing it with this Lab's chosen dictionary.
- Pythia-14M's pinned publisher commit and GPT-NeoX source establish the optional specimen and capture locations.