Exercise Sheet 2

November 17, 2025

Practical Notes

- Let’s set November 26th as the deadline.
- Use whatever tools work best for you—Google Colab, your university cluster, local setup, etc.
- These exercises require looking things up. Documentation, implementation details, examples online—not everything will be handed to you.
- Some experiments might not work perfectly on your first try. That’s completely normal and part of the process.
- Do experiments! Yes, I had specific ideas when designing this sheet, but there’s so much to discover! Document whatever you find interesting—there’s no single “right answer” here.

Part 1: What We Want to Tackle

Section 1: Preliminaries

In the last exercise sheet, we experimented with LogitLens. You should have noticed that tokens often appear in their English form before being "translated" into the target language. For example, you might have found how the model "predicts" the token "city" in an earlier layer before outputting the Modern Greek variant "πόλη" after the final layer. In Part 3, you should have implemented that this is not a one-time phenomenon, but rather that MLLMs systematically predict English tokens before switching to the target language.

However, we have also frequently observed that the LogitLens can only decode what the unembedding matrix is "receptive" to. If the unembedding matrix is "incompatible" with certain information in the residual stream (which might be needed for communication between layers), we have no way to query this with the LogitLens. The LogitLens only tells us what would be predicted; it doesn't give us causal mechanisms or anything else.

In the last exercise sheet, we also thought about how to design an experiment that builds on our initial exploratory insights. We now want to implement a specific experiment in this sheet. This experiment is less exploratory but should enable more detailed insights. At the same time, we want to take this opportunity to finally try out activation patching.

Section 2: Some Theory Building

The results from LogitLens suggest the theory that there is a "multilingual space" in the middle layers of the Transformer. Early layers project their language-specific information into this multilingual space. The "actual processing" happens there, which is largely language-independent. At the end, in the final layers, the answer must be projected back into a language-specific space to predict the correct token.

We were able to recognize the language-specific token at the end, but we also had to admit that we wouldn't really "notice" if the language information was already contained in the residual stream earlier. We now want to build on this and explore: How and where is concept information processed? How and where is language information processed? Are these two things connected?

We want to find out whether this is the case, whether we can find these 3 regions (language-specific, multilingual, language-specific again) and to what extent language

and content are coupled there. At this point, it makes sense to introduce some notation:

We are interested in an abstract concept C that can be expressed in different languages. C^ℓ would then be a language-specific version. For example, a concept could be $C = \text{CAT}$, a language $\ell = \text{EN}$, and the language-specific version of the concept C^{EN} would then be "cat", C^{DE} would be "Katze".

Additionally, we will only look at individual layers for now and not examine more granular units (MLPs, Attention Heads), as we already have enough to deal with.

Can we now find C and ℓ separately in the model? Since English occupies a "special position" in our LogitLens experiments and is presumably closely connected to the "multilingual space," we now focus on languages that are not English, so that we don't provoke a confounding factor here.

We have enough languages though. In our case, we settle on German, Italian, French, and Chinese. But we still have a problem: We don't have direct access to "the concept CAT" in the model. We can't simply say "Here is the abstract concept CAT, please decode it in Italian." The model works with tokens and text, not with abstract concepts. The only way to get a concept into the model is through a concrete linguistic realization - we have to input "Katze" or "cat" or "chat". But as soon as we do that, we've already chosen a language.

A workaround for this is a translation task: We give the model a concept in an input language (e.g., "Katze" in German) and ask it to produce the same concept in an output language (e.g., "gatto" in Italian). When the model solves this task, it must internally:

- Extract the concept from the input language
- Transfer it to the output language

The question now is: Does this happen directly (German→Italian) or is there an intermediate step via a language-independent concept representation?

For this, a somewhat complicated setup is necessary:

We construct two completely independent translation tasks:

- Source task (S): Translate concept C_S from German to Italian
  - Example: "Buch" → "libro"
- Target task (T): Translate concept C_T from French to Chinese

- Example: “citron” → “柠檬”

This way, the tasks have no common language and different concepts.

And now activation patching comes into play: The idea is that we take the hidden states from the source prompt and “implant” them into the target prompt at a specific location. Concretely, we perform two forward passes - one for the source prompt and one for the target prompt. During the target forward pass, we then replace the activation at a specific position and in a specific layer with the activation from the source prompt. We patch at the last token of the prompt, exactly at the point where the model should predict the concept to be translated. At this position, the model has just processed all information from the context and must now decide which token comes next. After that, we let the model continue normally and observe what it predicts.

When we implant the activation from the source prompt (which should translate “Buch” from German to Italian) into the target prompt (which should translate “citron” from French to Chinese), we can observe which information from the source prompt “comes along.” Depending on which layer we patch at, we expect four possible outputs:

- C_S^{IT}: The source concept (BOOK) in the source output language (IT) = “libro”
- C_S^{ZH}: The source concept (BOOK) in the target output language (ZH) = “书”
- C_T^{IT}: The target concept (LEMON) in the source output language (IT) = “limone”
- C_T^{ZH}: The target concept (LEMON) in the target output language (ZH) = “柠檬”

If we patch and the model says C_T^{ZH} (柠檬), it means: The patched activation has overwritten neither concept nor language - the model simply continues with what was in the target prompt.

But if we patch and suddenly see C_T^{IT} (limone), then the source activation has overwritten the output language (from Chinese to Italian), but the concept is still from the target prompt (LEMON). And if we patch in late layers and get C_S^{IT} (libro), then both language and concept were taken over from the source prompt.

Part 2: Implementation Task

Let’s try to make it work. We could use this as the start of our script:

```
import csv
import random
```

```
import torch
import torch.nn as nn
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

CSV_PATH = "en_fr_de_it_zh_candidates.csv"
MODEL_ID = "CohereForAI/aya-expanse-8b"
USE_4BIT = True
DEVICE_MAP = "auto"
SEED = 42

random.seed(SEED)
torch.manual_seed(SEED)
torch.set_grad_enabled(False)


bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.float16,
)

tokenizer = AutoTokenizer.from_pretrained(MODEL_ID, use_fast=True)
model = AutoModelForCausalLM.from_pretrained(
    MODEL_ID,
    device_map=DEVICE_MAP,
    trust_remote_code=True,
    quantization_config=bnb_config
)
model.eval()
```

Task 1: Prompt Construction

Build the few-shot translation prompts for both source (DE→IT) and target (FR→ZH) tasks. The prompt format should follow this structure:

For German→Italian:

```
Deutsch: "Haus" - Italiano: "casa"
Deutsch: "Katze" - Italiano: "gatto"
Deutsch: "Buch" - Italiano: "
```

For French→Chinese:

```
Français: "maison" - 中文: "房子"
Français: "chat" - 中文: "猫"
Français: "citron" - 中文: "
```

The prompt must end exactly at the opening quotation mark before the translation. The third row is the example to be completed. Avoid including the query concept itself in the preceding few-shot examples.

concept-language会产生四个不同结果，但是产生的是token，需要检测输出的那四个单词的“ 第一个 token ” 是否冲突

Task 2: Tokenization and Collision Detection

Understand how tokenization works after a quotation mark and implement collision detection. The first tokens of “libro”, “书”, “limone”, and “柠檬” must not overlap - otherwise you can’t cleanly measure which concept-language combination the model predicts. Keep in mind that the tokenization of " libro" can differ from the tokenization of “libro”.

Once you can extract first tokens reliably, filter your concept pairs to keep only those without collisions across all four tracked sets (C_S^{IT}, C_S^{ZH}, C_T^{IT}, C_T^{ZH}).

Task 3: Activation Extraction and Patching

Implement the core patching mechanism: extract hidden states at the last token position for all layers, patch these activations from source into target at a specified layer, and continue the forward pass to get next-token predictions.

A small hint (you *can* use it, but you don’t *have to* use it!): PyTorch’s `register_forward_hook` allows you to intervene during computation. A hook function receives `(module, input, output)` and can return a modified output. The hook signature looks like:

```
def hook_fn(module, input, output):
    # Modify output here
    return modified_output

handle = layer.register_forward_hook(hook_fn)
# run forward pass
handle.remove()
```

Task 4: Probability Tracking and Experiment Execution

From the model’s next-token distribution, compute probability mass for each of the four tracked combinations. Then run the complete experiment: loop over valid concept pairs,

patch at every layer, and aggregate results.

Task 5: Analysis

Create visualizations that reveal the three-phase pattern. Your analysis should identify:

- When does the language switch occur?
- When does the concept switch occur?

Task 6: Reflection

Write down everything you found interesting!