CSC485/2501 A2

TA: Ruiyu Wang

From Last Week…

- Now available!
- Due 17.00 Monday, Nov 3.

- Word Sense Disambiguation (q0, q1, q2) <- *last tutorial*
- **LLM causal tracing (q3) <- this tutorial!**
- After A2, you will be familiar with…
  - NLTK
  - WordNet
  - The Lesk algorithm
  - Word embeddings methods such as word2vec and BERT
  - **How LLMs work internally when prompting**

Casual Tracing

In question 3, we explore how modern transformer LLMs store facts

- Specifically, how they answer simple, relational questions
- e.g. The capital of France is ___? The CN Tower is located in the city of ___?

In order to find how LLMs interact with them, we measure a metric called the **indirect effect**

- i.e. the impact of disabling some hidden states in the model
- We will go through all the terminologies in this tutorial

Preliminaries

What is a neural network?

Two key components:

1. Preset network architectures (neurons)
2. Learn patterns of large datasets

Why is #1 important: you don’t compute and specify each parameter; the learning does

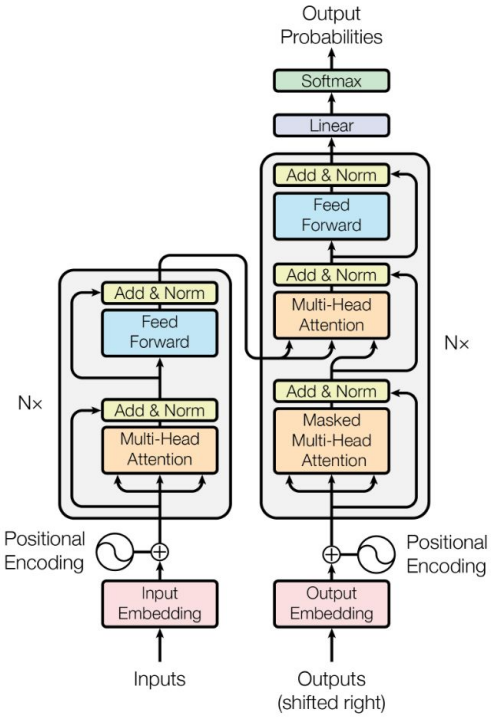
Why is #2 important: task matters!

Preliminaries: the Transformer

Attention Is All You Need. NeurIPS 2017.

Architecture:

- Multiple attention layers. In each layer:
  1. An attention block
  2. A feed-forward layer (basically, an MLP)
  3. Residual connections between the modules and the input
     i.e. `out = input + f(input)`
- Encoder: self-attention; decoder: cross-attention (Enc-dec)



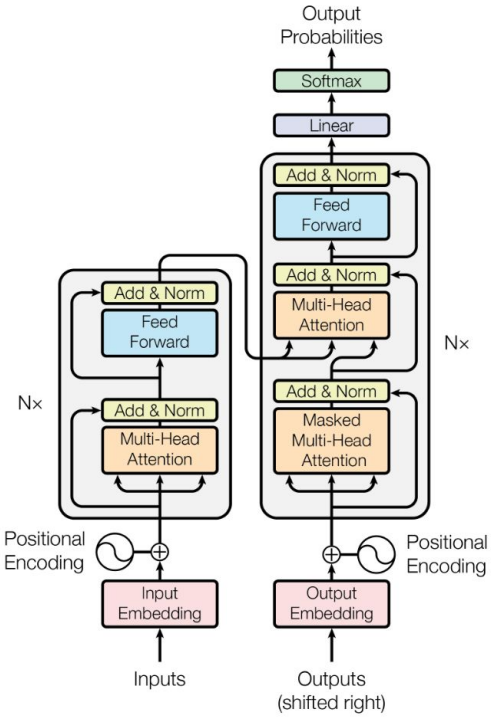
*we do not go in detail of this architecture. Just some high-level ideas

Preliminaries: the Transformer

Attention Is All You Need. NeurIPS 2017.

Task: next-token prediction

- Given the sentence prefix {t_0, …, t_(n-1)}, predict token t_n
- Applications: machine translations, language modelling, and a lot of extrinsic tasks (like how you use the LLMs these days)



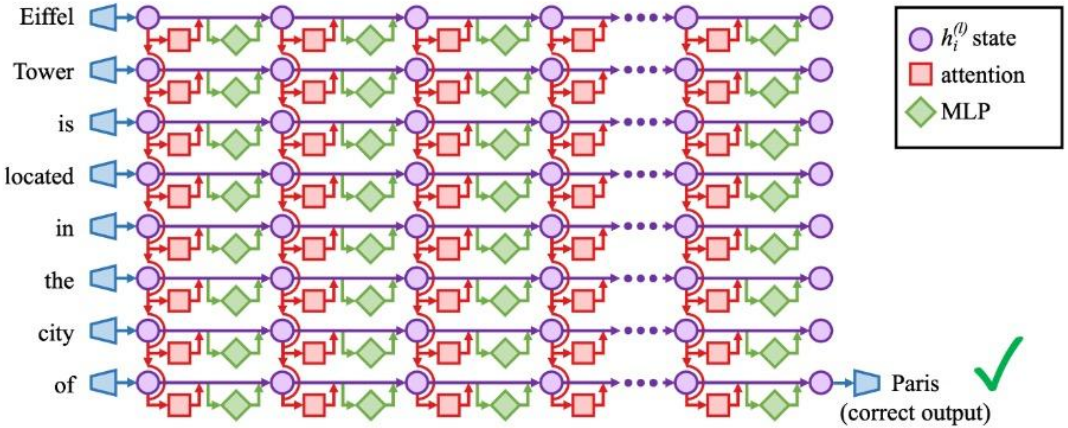
*we do not go in detail of this architecture. Just some high-level ideas

Preliminaries: GPT

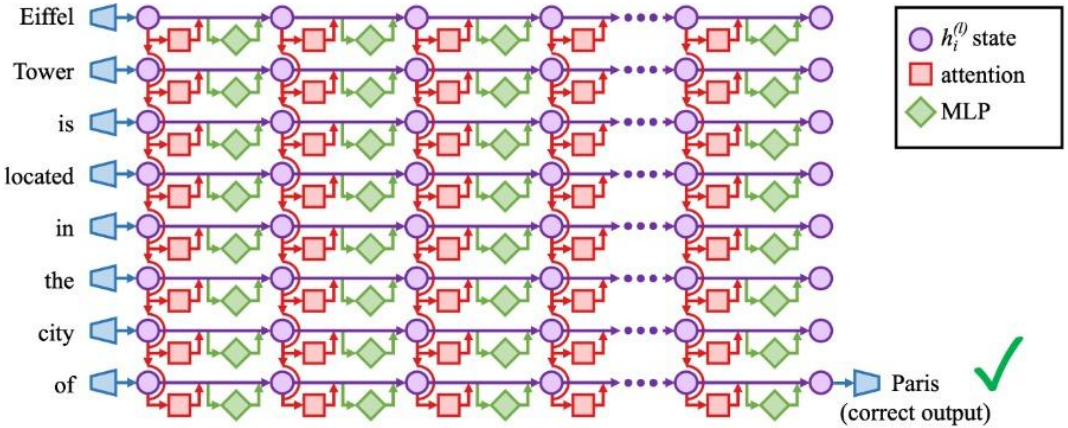
(GPT-2) *Language Models are Unsupervised Multitask Learners.* OpenAI tech report 2019

- The backbone model we are using!
- A decoder-only model: self-attention on the (auto-regressive) input

  … which means we can simplify the graph of information flows as below:



Casual Tracking



Given the sentence “Eiffel Tower is located in the city of …”, when does the language model realizes that the answer is “Paris”?

Assumption: the factual knowledges must be stored somewhere in the model weights

Casual Tracking

We explore three different location types:

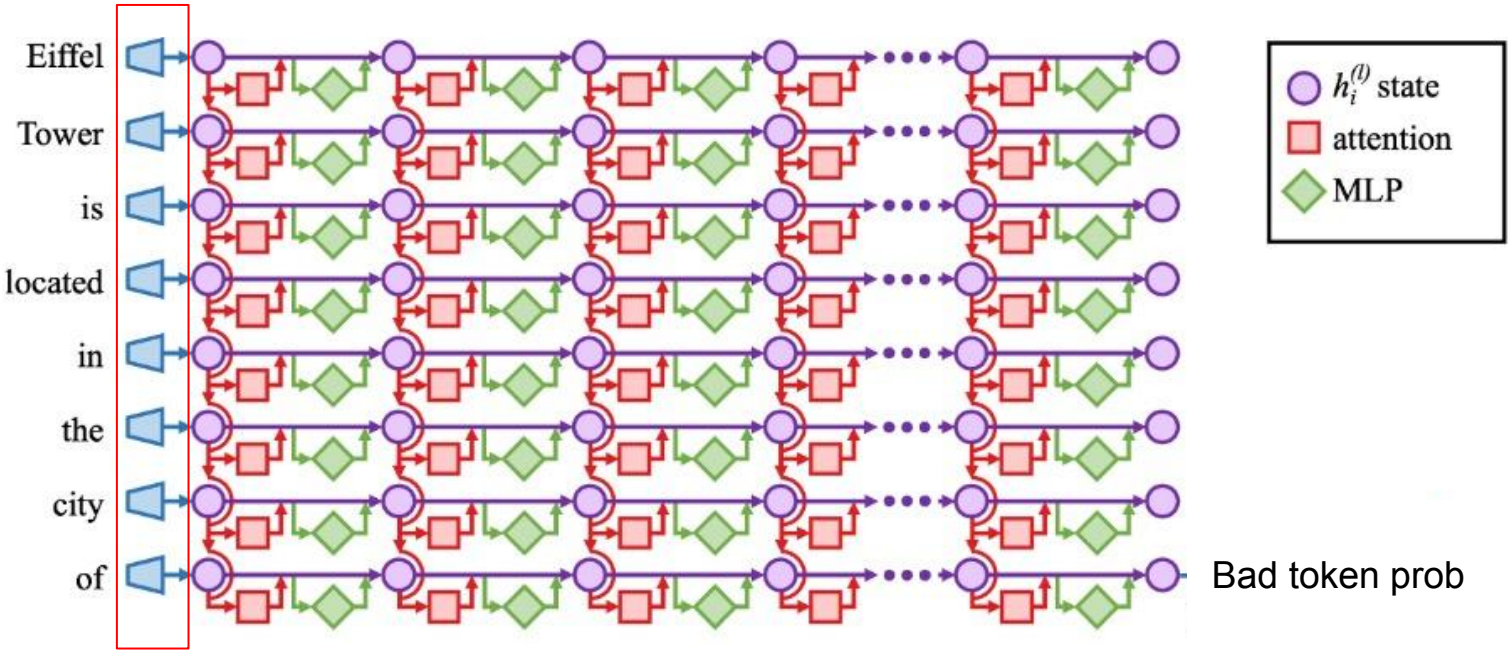
1. Each entire layer;
2. Each attention interval;
3. Each MLP interval (interval = a window of 10)

What we do:

1. Conduct a **corrupted run**: obfuscate some hidden states, compute the output distribution (the logit)
2. Conduct a **corrupted with restoration run**: corrupt the inputs, but restore the hidden state at some token positions and layers. Compute the output dist.
3. **Indirect Effect**: the difference between the corrupted and restored probability

How to corrupt: given the located hidden state, add a random noise (noise * randn)

1. Corrupt

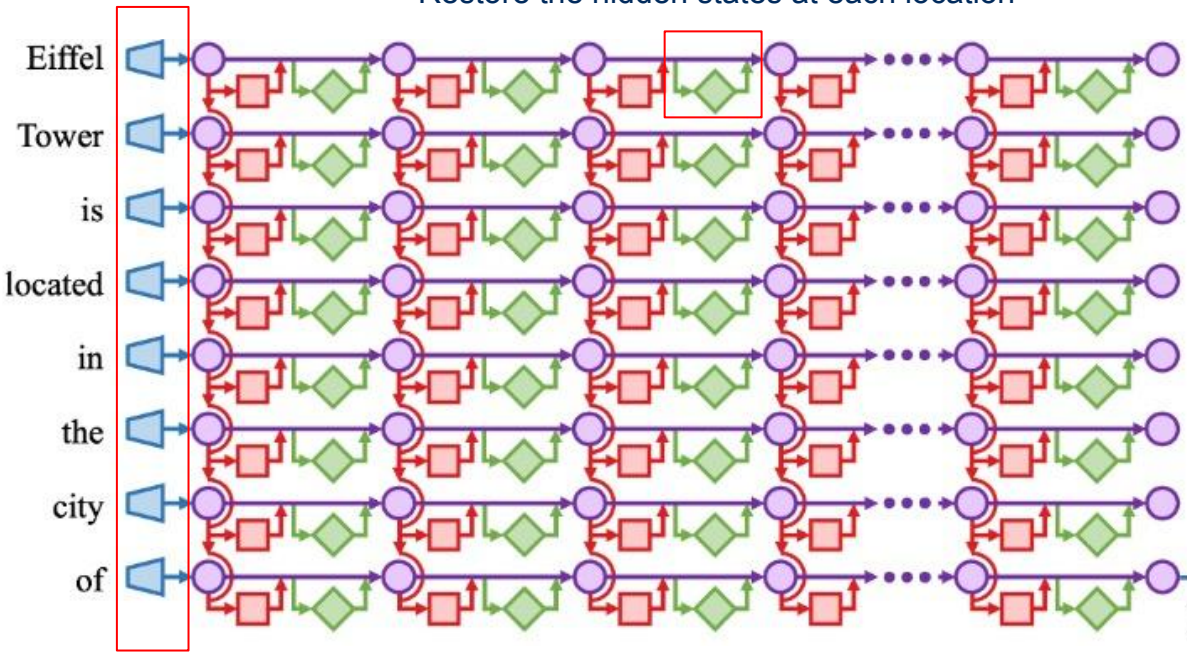


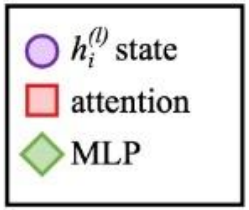
Bad token prob

Add random noise to these representations

2. Restore

Restore the hidden states at each location





Add random noise to these representations

Restored token prob

IE = P_clean - P_bad

Bad token prob

TransformerLens

A Library we use for mechanistic interpretability of generative language models.

How we run the process:

1. Specify a location (hook)
2. Define a function of what to do at that location
3. Run model.run_with_hooks

run_with_hooks():

- Input: the tokenized prompts, and a *forward_hook*
- Return: the logits

Hooks: List[Tuple[str, Callable]]: a list of hooked LM locations and the relevant functions

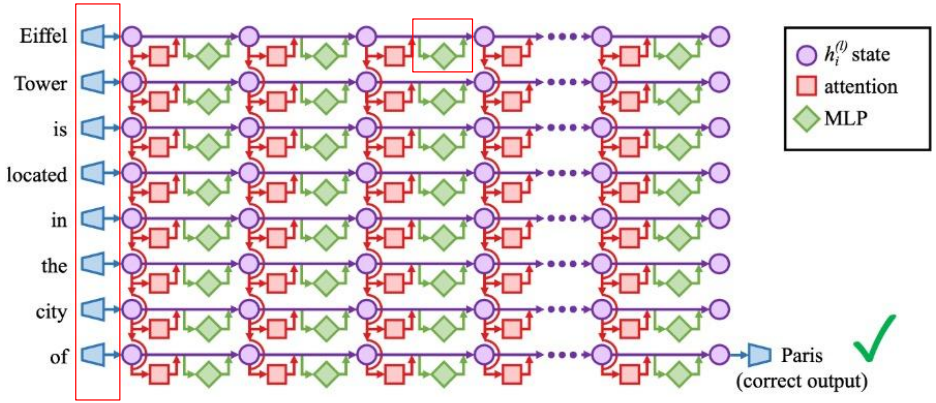
- How to retrieve the locations: utils.get_act_name(NAME)
- NAME could be ‘embed’, ‘resid_pre’, ‘mlp_post’, ‘attn_out’ # don’t forget the embedding!

How the Hook Works…

```
hooks = [
        (utils.get_act_name('embed'), fn),
        (utils.get_act_name('mlp_post', 2), restore_fn),
        …
]

logits = run_with_hooks(prompt_tokens, fwd_hooks=hooks)

# compute your prob based on the logits
```



Functions To-do

We have implemented these for you:

1. get_target_id(): get the id of the token (you will need it to get your logit probability)
2. record_clean_activations(): provide a record for you to restore the hidden states in the future

You should implement the following methods:

1. get_restore_fn(): provide a function that uses the activation record (computed above) to retrieve the activations for the specified token
2. get_patch_emb_fn(): provide a function that corrupts the specified span
3. get_corrupted_probs(): conduct the **corrupted run**
4. find_sequence_span(): find the token indices of the given sequence part
   e.g. find_sequence_span('The Space Needle', 'The Space Needle is located in …') = [0, 1, 2, 3, 4]
   # note that tokenization splits 'Needle' into 'Need', '#le'

Functions To-do

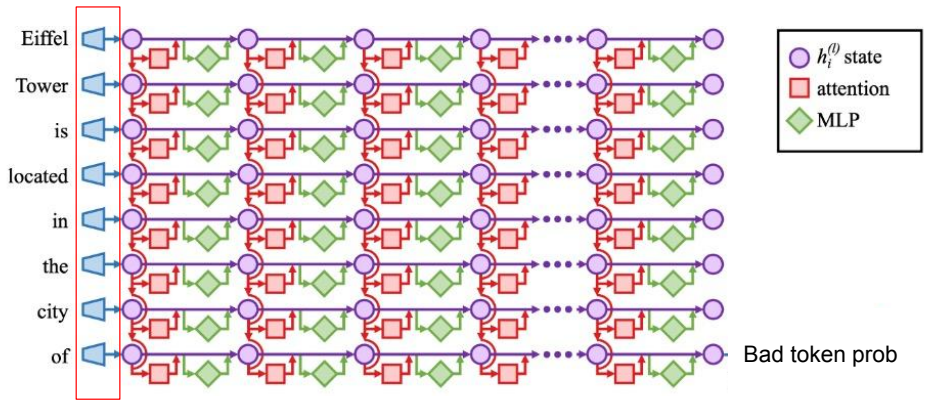
You should implement the following methods:

5. get_forward_hooks(): get your hook.
   - Recall: Hooks is List[Tuple[str, Callable]]
   - What does this list mean: for each LLM location (described by the str), use the relevant function to do certain computation (either corrupt the embeddings or restore the hidden states)
6. casual_trace_analysis():
   1. Corrupt the model and get the token probability
   2. For each layer and each token in the sequence, get the new probability computed by restoring the specific layer and token
   3. Compute the difference between the restored prob and the corrupted prob
   4. Result: a tensor of shape (seq_length - 1, layer_num)
      - # need to remove the start-of-sequence token!

The Complete Process

1. Implement your function to corrupt the embeddings (actually, the function to get that function)
2. Design a hook for the corrupt run
3. Run the corrupt run

Nothing happens inside of the LLM.



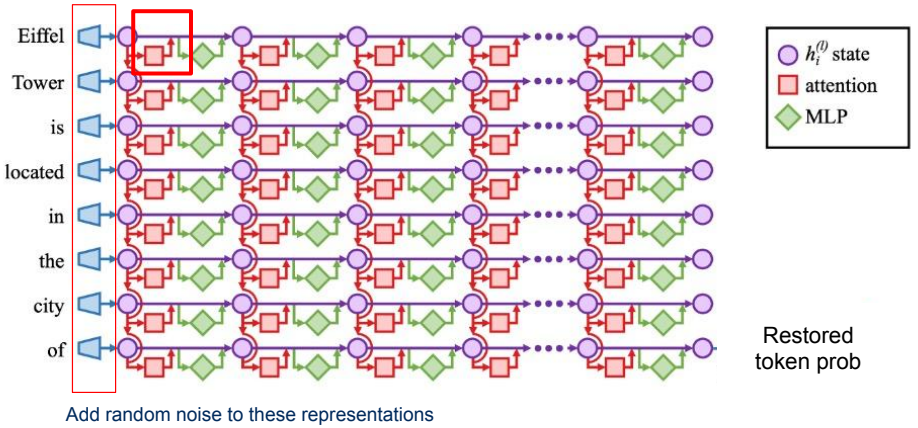
Add random noise to these representations

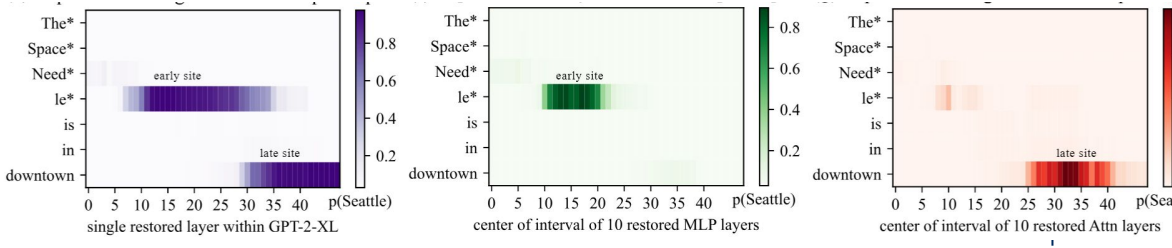
The Complete Process

After the corrupted run:

1. Implement your restoration function (actually, the function to get that function)
2. Design a hook for each token index and model layer, and run
3. Compute the probability difference, observe if there is an increase in the final likelihood
4. Form a graph!

run three times: layer, attn, mlp





UNIVERSITY OF TORONTO

Some Reminders

1. At get_forward_hooks(), you should have the MLP and Attn layers implemented with windowed layers and leave the entire-layer experiment individual
   i.e. [(*some_names*, *some_funcs*) for layer in window_layers] # you determine the windows

2. You should only take the last logit's distribution as your probability measurement

3. Easy assignment: x[span] = x_updated, as long as the size of the former matches the latter

4. The reason to create and retrieve a callable: we want dynamic function setups!
   a. get_restore_fn(record, 1) should yield a different callable than get_restore_fn(record, 2)

GLHF!

- If you have questions…
  - Ask now!
  - Post your questions on piazza (I do monitor the forum frequently)
- Other useful resources:
  - The Official Documentation of TransformerLens
  - https://rome.baulab.info/
  - https://stanfordnlp.github.io/pyvene/tutorials/advanced_tutorials/Causal_Tracing.html

Office Hour!
(No Attendance)

You are free to go if you don’t have any questions!