UNIVERSITAT D'ALACANT
UNIVERSIDAD DE ALICANTE
Escola Politècnica Superior
Escuela Politécnica Superior

MSc in Artificial Intelligence

Natural Language Processing Techniques's Assignment

Mechanistic Interpretability of Transformers Models

Limitation of GPT-2 in Mathematical Comparisons

Prepared by:

Gabriel Masella
Bahar Eddine Laihi

January 24, 2025

Contents

1 Introduction

The field of mechanistic interpretability aims to understand how neural networks can make decisions. Transformers, such as GPT-2, are widely used in natural language processing, so it is important to understand how they work internally.
The objective of this assignment is to explore Mechanistic Interpretability in Transformer models through the implementation of activation patching on a GPT-2 model. The task involves the execution of GPT-2 with two inputs, consisting of a clean text and a corrupted text that differs by a single token. The model output probabilities will then be analysed to identify the changes caused by the corruption. By intervening in specific model activations and comparing the probabilities, it is possible to gain insights into how individual activations influence the model's predictions.
In particular, in this project the objective is to investigate the activation patching process in two scenarios: firstly, by demonstrating the correct propagation of information in a basic example; and secondly, by testing the model with a more complex example involving a mathematical comparison.

2 Experimentation

2.1 GPT-2 Model Implementation

For this project a PyTorch re-implementation of GPT-2, both training and inference, is used called minGPT that tries to be small, clean, interpretable and educational, as most of the currently available GPT model implementations can a bit sprawling.
The minGPT library is composed of three files:

- mingpt/model.py contains the actual Transformer model definition,
- mingpt/bpe.py contains a mildly refactored Byte Pair Encoder that translates between text and sequences of integers exactly like OpenAI did in GPT,
- mingpt/trainer.py is (GPT-independent) PyTorch boilerplate code that trains the model.

The implementation of this project required several modifications to the GPT model architecture in order to enable interpretability analysis through activation patching.
The main changes focused on expanding the forward pass of the model to support two key functionalities: activation storage and targeted patching. A new "`layer_activations`" dictionary attribute was introduced to store intermediate activations at each transformer layer, with each activation tensor properly detached and cloned to prevent unwanted gradient flow and ensure data persistence. A "`store_activations`" parameter has been added to the forward method which, when enabled, captures and stores these layer-wise activations. In addition, a "`patch_params`" parameter was implemented to allow precise intervention at specific layers and positions, allowing the substitution of activations from one forward pass to another. This modification facilitates comparative analysis between different model runs. The implementation also includes the storage of the last token's logits through the "`last_token_logits`" attribute, which is crucial for analysing the model's final predictions.
These architectural changes preserve the model's original functionality while adding the necessary infrastructure for detailed mechanistic interpretability studies.

2.2 Testing Code

This experiment analyzes how a GPT model processes clean and corrupted inputs, emphasizing differences in token predictions and the impact of patching clean activations into corrupted inputs.

2.2.1 Model Initialization

Two distinct types of GPT models (namely, "`GPT-2-medium`" and "`GPT-2-xl`") are employed, followed by the loading and initialisation of the model in evaluation mode.

Listing 1: Model Initialization

```python
model_type = "GPT-2-medium"  # or "GPT-2-xl" for more parameters
# Initialize model
model = GPT.from_pretrained(model_type)
model.to(device)
model.eval()
```

2.2.2 Tokenization

The tokenizer is initialized and used to tokenize the clean and corrupted inputs.

Listing 2: Tokenization

```python
tokenizer = BPETokenizer()
clean = tokenizer(CLEAN_INPUT + END).to(device)
corrupted = tokenizer(CORRUPTED_INPUT + END).to(device)
```

2.2.3 Forward Pass

The tokenized input is passed through the model, generating logits (raw scores for the likelihood of each token in the vocabulary). The logits and intermediate layer activations are stored for later use.

Listing 3: Get predictions for clean input

```python
logits_clean, _ = model(clean, store_activations=True)
clean_activations = model.layer_activations.copy()
```

2.2.4 Prediction Analysis

The ”`get_top_predictions` function processes the logits by applying a softmax operation to convert raw scores into probabilities.

Listing 4: Prediction Analysis

```python
def get_top_predictions(logits, tokenizer, k=20):
    """Get top k predictions from logits"""
    probs = F.softmax(logits, dim=-1)
    top_probs, top_indices = th.topk(probs, k)

    predictions = []
    for prob, idx in zip(top_probs[0], top_indices[0]):
        # Convert single index to tensor before decoding
        idx_tensor = th.tensor([idx.item()])
        token = tokenizer.decode(idx_tensor)
        predictions.append((token, prob.item()))
    return predictions
```

2.2.5 Patching Mechanism

The patching mechanism involves systematically analyzing the impact of corruption by replacing activations from the corrupted input with those from the clean input during the forward pass of the transformer model.
This process is carried out layer by layer, iterating through each layer of the model and substituting the corrupted activations with the corresponding clean activations at every layer and token position.
The effect of these substitutions is measured by observing changes in the probability of the chosen tokens for each patched configuration, allowing for a detailed assessment of the corruption's impact.

Listing 5: Patching Mechanism

```python
smith_idx = tokenizer(" Jones")[0]
reference_logits = model.last_token_logits[0]
smith_prob = reference_logits[smith_idx].item()

# Setup patching loop
n_layers = len(model.transformer.h)
print(f"Number of layers: {n_layers}")
seq_length = corrupted.size(1)
print(f"Sequence length: {seq_length}")
diff_matrix = np.zeros((n_layers, seq_length))
```

```python

# Get tokens for x-axis labels
tokens = [tokenizer.decode(th.tensor([corrupted[0, i].item()])) for i in range(seq_length)]

# Iterate through layers and positions
for layer in range(n_layers):
    for pos in range(seq_length):
        # Forward pass with patching
        logits_patched, _ = model(corrupted.to(device),
                                  patch_params=(layer, pos,
                                                clean_activations[f'layer_{layer}'][:, pos, :]))

        # Convert logits to probabilities
        clean_probs = F.softmax(reference_logits, dim=-1)
        patched_probs = F.softmax(model.last_token_logits[0], dim=-1)

        # Compute probability difference
        diff_matrix[layer, pos] = patched_probs[smith_idx].item() - clean_probs[smith_idx].item()

# Get predictions for corrupted input
logits_corrupted, _ = model(corrupted.to(device), patch_params=(n_layers,seq_length,
                                          clean_activations[f'layer_{n_layers-1}'][:, seq_length-1,
    :]))
```

3 Result

In this chapter, the results of the study are presented, with a focus on the model's processing of both clean and corrupted inputs. The aim is to examine the activation patching process in two distinct scenarios:

- Firstly, the basic example is taken to demonstrate how the information is propagated correctly in the model.
- In the second case, a more complex example is considered, in which the model is tested using a small mathematical comparison. This comparison is based on a small curiosity about the achievements of two well-known football players, Cristiano Ronaldo and Lionel Messi, in the Champions League tournament.

3.1 Basic Example

Clean Input: Michelle Jones was a top-notch student. Michelle
Corrupted Input: Michelle Smith was a top-notch student. Michelle

In order to examine the model's output probability for the token following Michelle's second appearance, the texts were fed into GPT-2. The following ten most probable tokens were obtained, as illustrated in Table 1.

	Clear Input		**Currepted Input**	
Position	**Token**	**Probability**	**Token**	**Probability**
1	was	0.1730	Smith	0.1700
2	Jones	0.1592	was	0.1609
3	's	0.0924	's	0.0984
4	had	0.0447	had	0.0472
5	,	0.0394	,	0.0432
6	is	0.0264	is	0.0227
7	and	0.0182	and	0.0198
8	also	0.0156	also	0.0142
9	has	0.0141	worked	0.0137
10	worked	0.0135	has	0.0133

Table 1: Prediction Comparison of Clean and Corrupted Data

In the case of the clean input, the token "Jones" is associated with a notably high probability. However, when the corrupted input is utilized, the probability of "Jones" as the subsequent token declines significantly, while the probability of "Smith" ascends to the primary position with 17% of probability.
A patching heatmap, as shown in Fig. 1 is employed to analyze further the influence of clean activations on corrupted input predictions. This heatmap, which illustrates the information flow, demonstrates that the information moves from left to right and from the top to the bottom of the image due to the attention mask and layer arrangement.

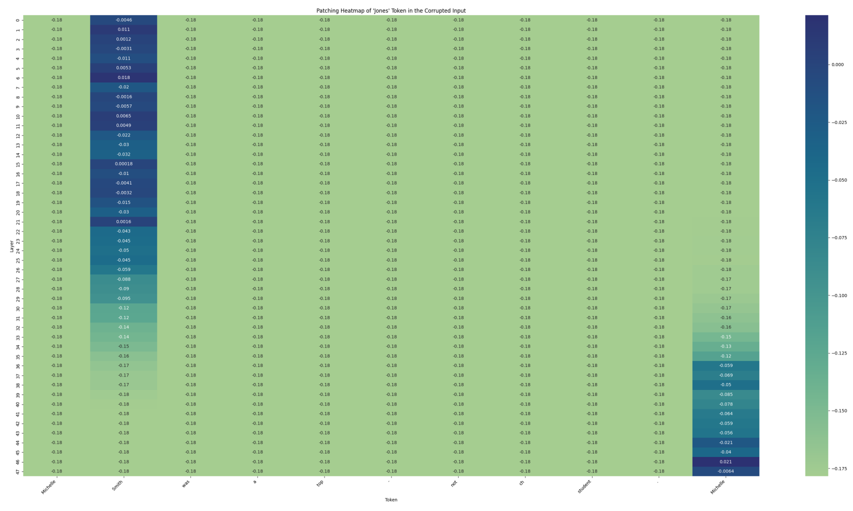


Figure 1: Patching Heatmap of the "`Jones`" token

The analysis focuses on specific layers and token positions where the clean context significantly influences restoring the probability of "Jones". Changes in the first column do not impact predictions, as embeddings are identical for both clean and corrupted models in the same context. Similarly, patching embeddings from the third to the penultimate column exhibits minimal effect. Intervening in the embeddings of multiple layers of the second token shifts the prediction towards "Jones" (evidenced by the darkening color as the logit difference between "Smith" and "Jones" becomes negative). Modifying the embeddings of the final layers of the second token has a much smaller effect. In the final position ("Michelle"), the embeddings of the last layers appear to anticipate the token to predict.

3.2 Mathematical Comparison Example

Further analyses are conducted to assess the capabilities of the GPT-2 model by introducing more complex prompts based on mathematical comparisons. Specifically, the model is expected to respond with the subject containing the highest number among two statements, following an ending sentence. In order to achieve this goal, the present study employed two curiosities based on the achievements in the Champions League tournament of two of the most famous soccer players in the world: Cristiano Ronaldo and Lionel Messi.

> **Clean Input:** Cristiano Ronaldo has won five Champions League titles. Lionel Messi has won four Champions League titles. The player who has won the most Champions League titles is
> **Corrupted Input:** Cristiano Ronaldo has won two Champions League titles. Lionel Messi has won four Champions League titles. The player who has won the most Champions League titles is

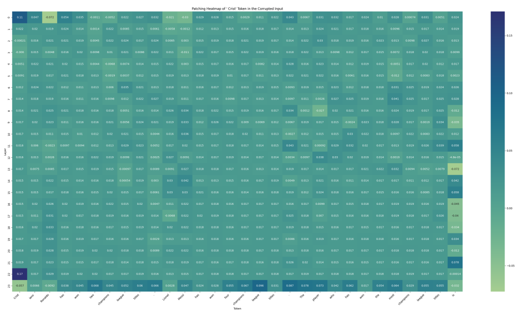
Given the tokenization of the sentences, the following tokens are taken in examination: "`Crist`", "`Ronaldo`", "`Lionel`", "`Messi`".
The expected response for the clean input is that the tokens associated with Cristiano Ronaldo have a higher probability than the tokens associated with Lionel Messi, given that Ronaldo is the player with the most Champions League appearances. For the corrupted input, the expected response is contrary

to this, due to the alteration of a single token.

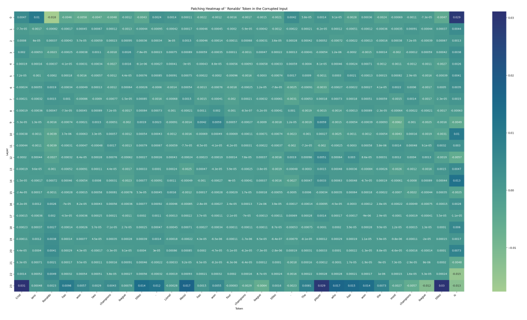
		GPT-2-medium		GPT-2-xl	
	Position	**Token**	**Probability**	**Token**	**Probability**
Clean	1	Crist	0.3803	Crist	0.2345
	2	Ronaldo	0.0421	Ronaldo	0.0272
	3	Lionel	0.0212	Lionel	0.0184
	4	Messi	0.0081	Messi	0.0082
Corrupted	1	Crist	0.3974	Crist	0.2478
	2	Ronaldo	0.0421	Ronaldo	0.0298
	3	Lionel	0.0216	Lionel	0.0187
	4	Messi	0.0083	Messi	0.0083

Table 2: Performance Comparison of Clean and Corrupted Data with GPT-2 Medium and XL Models

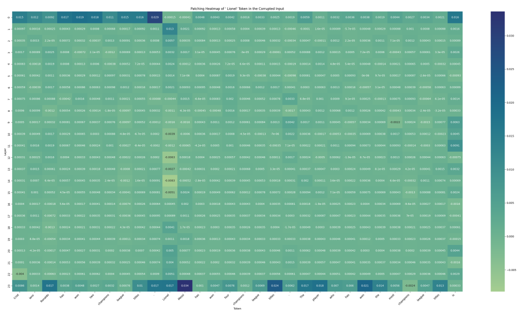
It was found that the probabilities, shown in table 2 from clean to corrupted did not change, maintaining a high probability for the same token. This occurred even though, in the corrupted input, the right answer was related to the other player.
In order to investigate transformer model functionality, a heatmap with activation patching was plotted for each desired token. Analysis of the figure 2 revealed that, in contrast to the basic example, there was no evidence of information propagation.



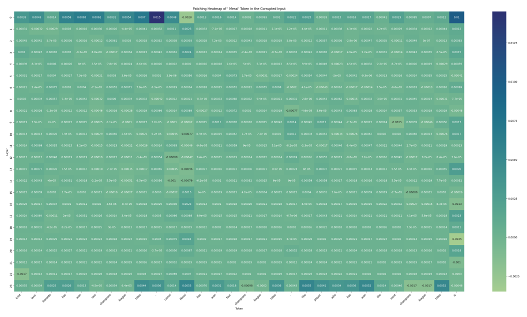
"`Crist`" token



"`Ronaldo`" token



"`Lionel`" token



"`Messi`" token

Figure 2: Heatmaps for Clean Activation Patching with GPT-2 Medium Model

In the course of the research, further attempts were made to modify the numerical values within the sentences, operating under the assumption that they are comprised of a single token. The outcomes of

these attempts exhibited a consistent pattern. However, a notable exception was observed when the order of the first two sentences was reversed, such that the statistic related to Messi appeared before the one associated with Ronaldo.
Examination of the probabilities presented in Table 3 reveals that the "`Messi`" token consistently exhibits the highest probability, independently of the correctness of the information.

		GPT-2-medium		**GPT-2-xl**	
	Position	**Token**	**Probability**	**Token**	**Probability**
Clean	1	Crist	0.0379	Crist	0.0162
	2	Ronaldo	0.0296	Ronaldo	0.0113
	3	Lionel	0.2607	Lionel	0.2351
	4	Messi	0.0404	Messi	0.0152
Corrupted	1	Crist	0.0409	Crist	0.0173
	2	Ronaldo	0.0310	Ronaldo	0.0121
	3	Lionel	0.2616	Lionel	0.2373
	4	Messi	0.0434	Messi	0.0157

Table 3: Performance Comparison of Clean and Corrupted with GPT-2 Medium and XL Models inverting the order of the two initial sentences.

4 Conclusion

This study explored the concept of mechanistic interpretability in GPT-2 models through activation patching. By comparing clean and corrupted inputs, it was shown how specific activations influence model predictions. The basic example demonstrated clear information propagation, while the complex mathematical comparison revealed limitations in the model's ability to handle numerical reasoning. Despite modifications to the GPT-2 architecture for interpretability, the model struggled with this intricate task. The findings emphasise the necessity of comprehending the internal workings of neural networks to enhance the reliability and efficacy of models in natural language processing applications.