UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

SEMINAR

A survey on Large Language Model interpretability methods

Jerko Šegvić
Mentor: *Laura Majer, Jan Šnajder*

Zagreb, May 2024.

CONTENTS

1. Introduction

Large language models (LLMs) are nowadays one of the most popular areas of interest in artificial intelligence. Their popularity skyrocketed in 2019 when first of them set new *state-of-the-art* results on almost all of NLP benchmarks, achieving results that were unimaginable before them. However, we still don't know how exactly they work. It is a very common problem with deep models, but with language models, it's particularly prominent. The problem with text is that it's not naturally a tensor (or vector), so it's not possible to process it with a neural model without some preprocessing. After the preprocessing (i.e. tokenization and embedding), it's not easy to interpret the embedding's meaning because the token embeddings are usually very high-dimensional vectors without some meaning that would make sense to humans (for example, images are also high-dimensional vectors/tensors, but when a human sees an image he "understands" what is on the picture).

1.1. Transformer block's interpretability

Today's LLMs are all based on transformer architecture introduced in Vaswani et al. (2017). The architecture of the generative decoder-based transformer is displayed on 1.1 (in the figure is a scheme of the GPT family, but others are very similar). Such models are trained for causal language modeling, i.e. predicting the next token of the sequence. Formaly, those models are modeling probability distribution

$$p(t_k|t_0, t_1, ...t_{k-1})$$

where t_i is the i-th token in the sequence.

The first part of the model (before transformer blocks) is responsible for embedding tokens. It transforms *one-hot* vectors to vectors representing tokens in the model's space and adds positional information. The second part is made of multiple transformer-decoder blocks. Each transformer block's main components are an attention module and a fully connected module. The attention mechanism is responsible for giving each token contextual information about other tokens and creating contextual embeddings. The last part of the model is a fully connected layer with softmax at the end. It's responsible for projecting the last hidden state of the model to the logit vector and transforming it with softmax to the probability distribution of the next token.

It's important to point out that in each module in transformer blocks, there is a residual connection, which means that we add up together the module's input and the module's output. The residual connections were first used in He et al. (2015). It turned out that models with residual connections don't suffer from vanishing gradient problems (or suffer less) than those without them so the models with residual connections can be deeper. The result of the layer with residual connection is equal to

$$F(x) = F'(x) + x$$

where F' is the layer's output. We can interpret that as "directing" the hidden state towards the desired state (e.g. desired probability distribution). With that in mind, it's reasonable to question if the model has learned the probability distribution of the next token before the last hidden state. Many papers research this topic.

In this seminar, the most popular ideas are going to be described and implemented. We will explore how well out model predicts the next token in every hidden state without additional training. Furthermore we will add new parameters on top of ecery hidden state and train them. Finaly, we will discuss the results of both experiments and try to interpret when which method works better.

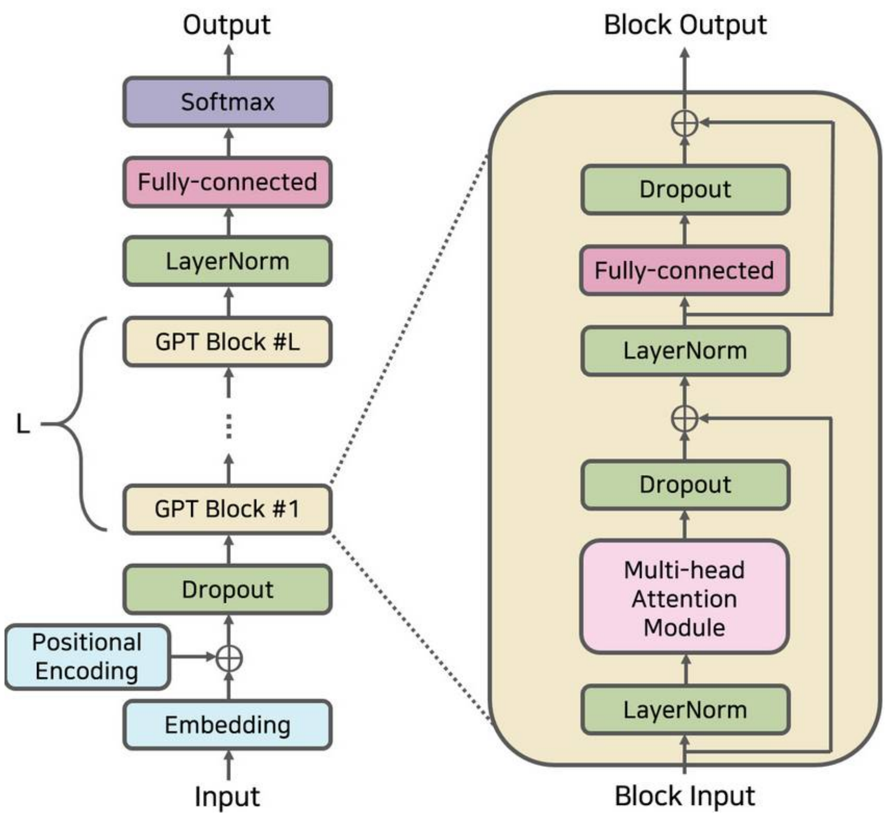

Figure 1.1: Arhitecture of decoder-based generative transformer (Lee (2023))

2. Logit lens

2.1. Method description

The most obvious way for interpreting hidden states in the model is directly projecting them to vocabulary (logit) space. Such an experiment was performed and described in nostalgebraist (2020). The projection is done by applying layer norm and linear layer which is used for projecting the last hidden state to vocabulary (see the top of 1.1). The performed operation is equal to

$$logits_{H_i} = Layer\_norm(H_i)W_u$$

where H_i is the projected hidden state, W_u is the unembedding parameter (matrix). The performance of every layer can be measured with cross-entropy (or $KL - divergence$) or the percentage of correct tokens being in the top k most probable tokens.

2.2. Experiments

For training data, we used the English Wikipedia dataset from Huggingface (https://huggingface.co/datasets/wikipedia). For getting sentences we used a similar algorithm to Din et al. (2023). We took 2000 different documents from Wikipedia and took one sentence from each of them. From each sentence, we take each token as a target and all tokens before them as a sequence from which we are predicting the next token. In that way, we got around 40000 examples which were used for training, and around 7000 examples for testing. In this experiment, we used only a test set. The results are displayed in 2.1. We use

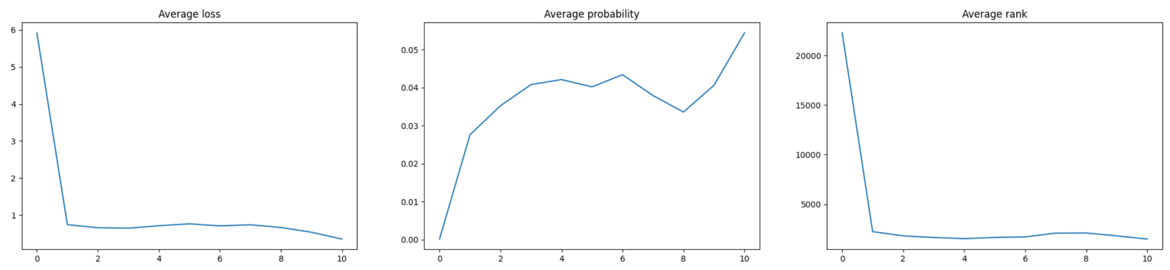

Figure 2.1: Logit lens results

average loss per layer, average probability per layer, and average rank per layer for performance metrics. From the results it is apparent that every metric has a huge jump from layer one to layer two. After the second layer, there are no major changes in performance. The expected behavior would be an increase in the performance in every layer. This might be due to the limited size of the validation set.

3. Tuned lens

3.1. Method description

The Logit lens approach can be improved if we add a module that learns a mapping between the hidden states and the final hidden state. If that module is much smaller in terms of a number of parameters than the corresponding transformer blocks between them, we can get a much smaller model that works comparably well. Such experiments were done in Din et al. (2023) and Belrose et al. (2023). Those two papers added a linear layer on top of every hidden state. The Din et al. (2023) trained linear layers to minimize MSE between the last hidden state and projection. The loss function for hidden state l and set of sequences τ is equal to

$$L(\tau, A) = \sum_{s \in \tau} \| Ah(s)^{l}_{i_s} - h(s)^{-1}_{i_s} \|^2$$

where i_s is the position of target token in sequence s. Belrose et al. (2023) trained linear layers to minimize $KL - divergence$ between the last hidden state and the projection. The loss function for hidden state l and set of sequences τ is equal to

$$P_{lens}(s, i_s) = softmax(Layer\_norm(Ah^{l}_{i_s})W_U)$$

$$P_{model}(s, i_s) = softmax(model\_logits)$$

$$L(\tau, A) = \sum_{s \in \tau} D_{KL}(P_{lens}(s, i_s), P_{model}(s, i_s))$$

where i_s is the position of target token in sequence s and W_u is the unembedding matrix of the model with shape $d\_model \times |vocabulary|$. In 3.1 is illustrated the difference between the logit lens and tuned lens. The blue line represents tuned lens, i.e. the projection of hidden state h_l is made with arbitrary matrix A. The red line represents logit lens, i.e. the projection is made with identity matrix.

It's possible to use other loss functions, for example, to train the linear layer to maximize the probability of the correct token (with cross-entropy loss) or to train linear layers combined with unembedding matrix to minimize $KL - divergence$ between the projected logits and the model's logits.

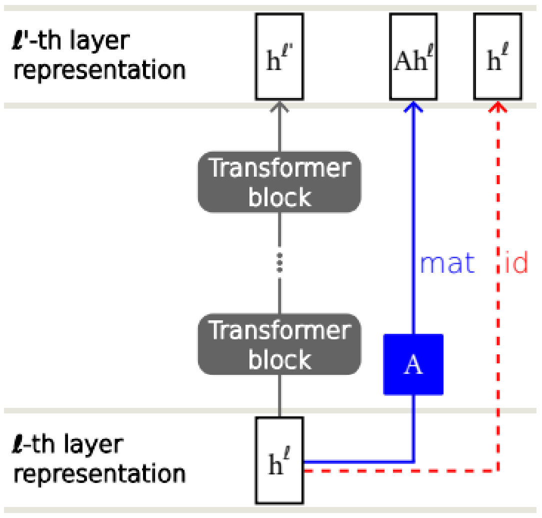

Figure 3.1: Logit lens (red) vs. tuned lens (blue) (Din et al. (2023))

3.2. Experiments

In this experiment, we used the same datasets from the previous experiments (Wikipedia sentences). We trained our linear layers to minimize $KL-divergence$ between the model's logits and logits from our hidden state projections. We measured the performance with the same metrics as in the previous experiment. The results are displayed in 3.2.

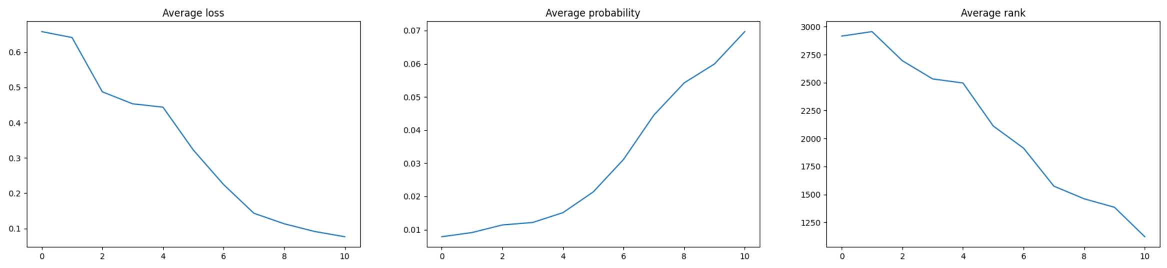

Figure 3.2: Tuned lens results

We can see that in this setup all metrics are more linear with respect to layers. In this setup, we trained the lens in a way that their logits are more similar to the model's logits. It's expected that with each layer we will get better performance. However, the lower $KL-divergence$ between the layer's logits and the model's logit in the train set doesn't need to work well on the validation set.

4. Comparison

We would like to know if the learned projections are any better than the logit lens. The comparison between the two is displayed in 4.1 and those results are shown in 4.1, 4.2 and 4.3.

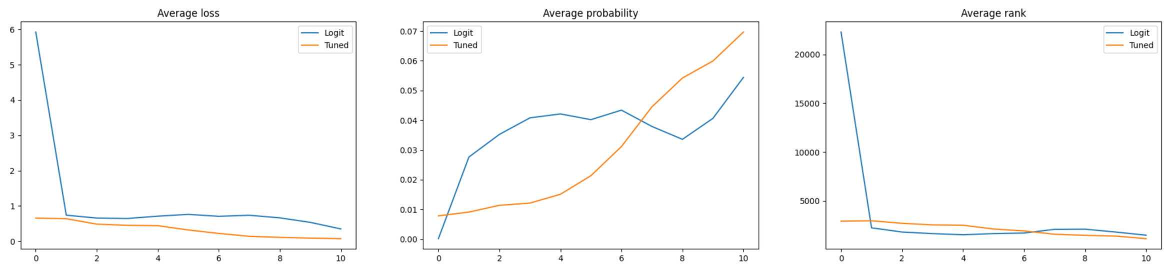

Figure 4.1: Combined results

Layer	1	2	3	4	5	6	7	8	9	10	11
Logit lens	5.92	0.74	0.656	0.65	0.71	0.76	0.71	0.74	0.67	0.54	0.35
Tuned lens	0.66	0.64	0.49	0.45	0.44	0.32	0.22	0.14	0.11	0.09	0.07

Table 4.1: Average $KL - divergence$ per layer

Layer	1	2	3	4	5	6	7	8	9	10	11
Logit lens	22285	2230	1799	1637	1521	1641	1699	2075	2092	1801	1477
Tuned lens	2915	2955	2695	2532	2496	2111	1913	1574	1461	1385	1122

Table 4.2: Average rank of correct token per layer

If we compare the performance of the logit lens with the tuned lens, we see that the $KL - divergence$ between the tuned lens's logits and logit lens's logits is always lower for the tuned lens. But average probability and average rank are better for the logit lens before the 7th layer while better for the tuned lens after the 7th layer. We argue that it is due to the overfitting of the tuned lens, which led to poor generalization on the validation set. We

Layer	1	2	3	4	5	6	7	8	9	10	11
Logit lens	0.01	2.7	3.5	4.1	4.2	4	4.3	3.7	3.3	4.1	5.4
Tuned lens	0.8	0.9	1.1	1.2	1.5	2.1	3.1	4.5	5.4	6	7

Table 4.3: Average probability of correct token per layer (percentege)

can further explore this thesis by executing experiments on a larger scale and with more validation (we didn't optimize any hyperparameters in those experiments).

5. Conclusion

In our experiments we showed that all LLMs hidden states contain information about the next token prediction. Furthermore, we showed that we can add another linear layer on top of every hidden state and train in to minimize $KL - divergence$ between next token distribution of hidden state and "real" model's probability distribution. Also we noticed that earlier layers tend to overfit in such experimental setup. We could probably reduce those effects by adding more data. In Voita et al. (2023), authors discoverd that many neurons of FFN modules in earlier transformer blocks are often "dead", i.e. they almost never activate. They interpreted dead neurons as detectors for shallow concepts such as lexical features of the embeddings. They argue that earlier blocks learns more simple features, while the higher blocks learn semantics and reasoning. Our experimental results fit well in this theory, because with every layer (which has more semantic information) we got better results. Also, the earlier layers that contain only more primitive features tend to overfit in experiment with tuned lens. It is expected behaviour for a model to overfit if the features are not informatie enough. For example, if we train a linear model to separate linearly non-separable points in Euclidean space, we won't get any good results, and model will probably overfit on train data. But, if we use RBF kernel functions or add polynomial features and train linear model on them, we will probably get much better generalization.

For further work we should execute experiments with different loss functions (explained in 3.1) and with more data. It will be interesting to see if we can get better results in such setups. We can also fine tune the unembedding matrix to see if that affects the performance. Another idea is to explore which are the examples that models predict well in earlier layers and which are those that fail. Out hypothesys is that examples that don't require any complex reasoning would work much better in earlier layers than those who do (for example the sentences with irony). We think that such experiments can largely increase our knowledge about how LLMs work and make them more interpretable.

6. Bibliography

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, i Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens, 2023.

Alexander Yom Din, Taelin Karidi, Leshem Choshen, i Mor Geva. Jump to conclusions: Short-cutting transformers with linear transformations, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition, 2015.

Minhyeok Lee. A mathematical investigation of hallucination and creativity in gpt models. *Mathematics*, 11:2320, 05 2023. doi: 10.3390/math11102320.

nostalgebraist. interpreting gpt: the logit lens. *LessWrong*, 2020. URL `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, i Illia Polosukhin. Attention is all you need, 2017.

Elena Voita, Javier Ferrando, i Christoforos Nalmpantis. Neurons in large language models: Dead, n-gram, positional, 2023.

7. Abstract

This seminar explores methods for interpreting LLM's hidden states. Since all transformer blocks in LLMs are the same in terms of architecture (they have different parameters), the question is what features are detected in each layer. One way to approach this problem is to see if it's possible to extract information about the next token prediction from every hidden state and see how the performance changes from one state to another. We evaluated the model with a method called logit lens, which does not require any additional training, and tuned lens, which adds a linear layer on top of every transformer block and learns projection to the final hidden state. We measure the performance of every layer with $KL-divergence$, the probability of the correct token, and the rank of the correct token. Furthermore, we discuss the meaning of the results and suggest further research.