AutoContrastive Decoding to tackle the Inverse Scaling problem

Shyam Sundar **Suriya Prakash** **Roshan Varghese** **Som Sagar**

Arizona State University

Abstract

This study delves into addressing the inverse scaling problem within large language models (LLMs), comparing two prominent solutions: auto-contrastive decoding and disagreement regularization. Our focus lies on auto-contrastive decoding, emphasizing its efficacy and practical advantages. Auto-contrastive decoding enables the model to discern and refine its own outputs, fostering self-improvement particularly beneficial for large datasets. Its computational efficiency, achieved through dynamic depth adjustment using intermediate layers, contributes to adaptive processing. Implementation is characterized by simplicity and intuitiveness compared to alternative methods. Furthermore, auto-contrastive decoding enhances model robustness by promoting differentiation capabilities. This practical approach stands out when contrasted with potential challenges in implementing complex academic research-derived methods, such as disagreement regularization. We argue that auto-contrastive decoding presents a more feasible and effective solution to the inverse scaling problem in LLMs, offering a streamlined implementation and robust self-improvement mechanisms.

1 Introduction

The inverse scaling problem in the domain of Large Language Models (LLMs) constitutes a significant challenge that emerges as these models undergo substantial increases in size and complexity. While conventional wisdom dictates that larger models should consistently yield improved performance on various natural language processing tasks, empirical observations reveal a counterintuitive phenomenon. Beyond a certain threshold of model size, the anticipated performance gains either diminish or, in some cases, exhibit a detrimental effect on task-specific outcomes. Moreover, as models grow in size, computational demands escalate substantially. Training and fine-tuning processes become computationally intensive, requiring vast computational resources and extended periods of time. This escalation in computational requirements poses practical challenges, limiting the accessibility of these advanced models to researchers and organizations with substantial computing infrastructure. Furthermore, the inverse scaling problem introduces intricacies in the training and fine-tuning processes. The optimization landscape becomes more complex, making it challenging to converge to an optimal solution efficiently. Fine-tuning larger models requires careful consideration of hyperparameters, regularization techniques, and training strategies to mitigate the adverse effects of overfitting and achieve meaningful improvements in performance.

In this study we use Auto-contrastive decoding which represents a novel technique aimed at ameliorating the adverse effects of the inverse scaling problem. This approach focuses on refining the decoding process, which is a critical stage in language model training where generated outputs are compared to target sequences. By incorporating auto-contrastive decoding, researchers aim to enhance the model's ability to generalize effectively, even as the model scales up in size.

CSE 576: Topics/Natural Language Processing (Fall 2023)

2 Auto-Contrastive Generation

This is a Contrastive Decoding method Li et al. (2022) Anonymous (2023) But in this setting, a single model instead of two distinct models, as seen in the conventional approach where a large and a small model of the same architecture are used. Our strategy follows the contrastive decoding (CD) approach to compute next-token predictions by contrasting the expert's predictions with those of the amateur. Here both the expert and the amateur are components within the same model, and they are defined by two different layers of that model. This method is termed Auto-contrastive Decoding(ACD) Gera et al. (2023) as shown in Fig 1. To perform contrastive decoding based on the logits (raw output scores before softmax) from two layers of a neural network, referred to as upper layer logits and lower layer logits. We by computing the softmax probabilities for both layers and establishing a threshold for plausible token probabilities. This threshold is determined relative to the top candidate probability, with a minimum threshold calculated based on user-defined parameters. The function then initializes tensors representing zero and negative infinity. It proceeds to calculate contrasted logits, which are the log ratios of the softmax probabilities between the upper and lower layers, selectively applied to tokens above the determined threshold. The redistributed probability mass for these tokens is computed using contrastive softmax scores. The final contrasted logits are obtained by combining the redistributed mass with the original upper layer logits for tokens above the threshold. The resulting tensor is then logarithmically transformed, and the final output is returned, ensuring it is moved to the specified computing device. This function essentially refines the probability distribution of tokens, emphasizing those deemed plausible based on the contrast between the upper and lower layer information.

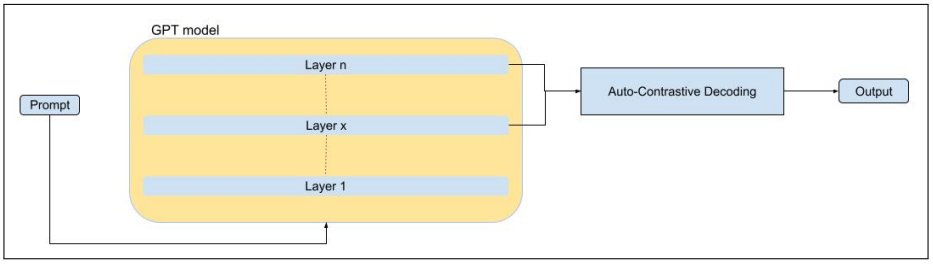

Figure 1: **auto-contrastive decoding with GPT2**

2.1 Training

The training procedure as shown in Fig 2 involves creating an amateur head in addition to the expert head in a pre-trained language model(PLM) to form a multi-exit model. The goal is to have an expert head, representing the final output layer of the original model, and an amateur head, which is a newly added linear head attached to an intermediate hidden layer of the model. The amateur head is responsible for generating a probability distribution over the vocabulary for predicting the next token in the sequence. The expert head is essentially the final output layer of the original PLM. The amateur head is added to one of the intermediate hidden layers of PLM. This introduces a new set of parameters that will be responsible for generating a probability distribution over the vocabulary for the next token. To train only the new amateur head, all the existing pre-trained weights of PLM are frozen. This means that the parameters in the original model are not updated during the training of the amateur head, ensuring that the expertise learned by the original model is retained.

To do this we extended the GPT2LMHeadModel class by augmenting the GPT-2 architecture with additional "exit heads" added to specific layers. These exit heads are linear layers appended to even-indexed layers of the GPT-2 transformer, each with an output size corresponding to the vocabulary size. This modification enables the model to generate intermediate predictions at these designated layers. The original GPT-2 layers' parameters are frozen to preserve their pre-trained weights, ensuring that the model's core language understanding capabilities remain intact. During the forward pass, the model processes the input data through the GPT-2 architecture, and for every even-indexed layer, it applies the respective exit head to the layer's output. The outputs from these exit heads are then collected and returned alongside the original GPT-2 logits.

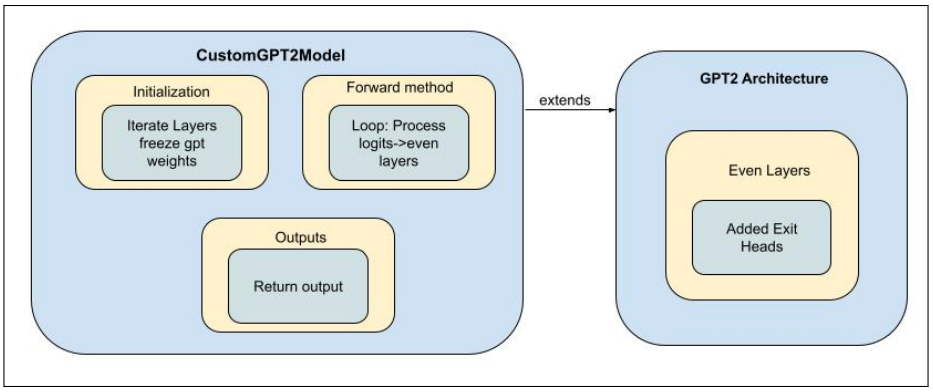

Figure 2: **Augmentation on GPT2**

3 Results and analysis

Before we infer the model on the Inverse scaling dataset, we had to benchmark the results of the paper on the LAMBADA dataset to validate the claims of the authors and to make sure our codebase works as intented. The reults of which are shown in 2

3.1 Inverse scaling problem

In our analysis of the modus tollens and memo trap classification datasets, we observed the "inverse scaling problem" as shown in Fig 3 and Fig 6. Contrary to expectations, increasing the model parameters, as evidenced by the comparison between GPT-2 large and GPT-2 medium, did not result in a proportional improvement in accuracy. Figure 1 and Figure 2 visually depict this trend, showcasing that GPT-2 large exhibited lower accuracy compared to GPT-2 medium, despite its significantly larger parameter count. This finding challenges the common assumption that larger models inherently lead to better performance, emphasizing the need for nuanced considerations in model selection and optimization.

Contrastive logit calculation The way the logits are constrasted with another layer has the following steps: **Softmax Calculation** Both sets of logits upper layer logits and lower layer logits are passed through the softmax function along the last dimension (dim=-1). This converts the logits into probability distributions. **Probability Thresholding** A probability threshold is set based on the maximum probability of the top candidate from lm logits upper. This threshold is calculated as the product of alpha and the maximum probability.
Contrastive Operation For tokens whose probability in logits upper is above the calculated threshold, a contrastive operation is performed. The contrasted logits are calculated as the log ratio of the probabilities in logits upper and logits lower.

3. Contrastive Operation:

$$\text{contrasted\_logits}_{ij} = \begin{cases} \log(\mathbf{p}_{ij}) - \log(\mathbf{q}_{ij}), & \text{if } \mathbf{p}_{ij} \geq \text{min\_threshold} \\ \mathbf{p}_{ij}, & \text{otherwise} \end{cases}$$

Softmax for Selected Tokens A softmax operation is applied to the contrasted logits, but only for tokens whose probability in lm logits upper is above the threshold. **Probability Redistribution** The probability mass of tokens above the threshold in logits upper is redistributed using the contrastive softmax scores. This is achieved by multiplying the softmax scores with the original probability mass. **Final Logarithmic Transformation** The final step involves taking the logarithm of the adjusted contrasted logits.

In summary, the function modifies the probabilities of tokens in upper layer logits based on a contrastive operation with lower layer logits for tokens above a certain probability threshold. The specific form of the contrastive operation aims to enhance the discriminative information between the two sets of logits.

Layer	Accuracy
6	0.4871
12	0.337
14	0.348
18	0.310

(a) Memo Trap results by contrasting with layer 24 of GPT-Medium.

Layer	Accuracy
4	0.5126
6	0.5096
8	0.5138

(b) Memo Trap results by contrasting with layer 12 of GPT-Small.

Table 1: Comparison of Memo Trap results with different layers of GPT models.

Model	Accuracy	Perplexity
GPT-Medium+ ACD	0.55	15.4

Table 2: Text generation results on LAMBADA.

Memo Trap The memo trap dataset puts forth prompts that are challenging for LLMs to respond as instructed. A typical example prompt would look like:

Write a quote that ends in the word 'thoughts':
Actions speak louder than
To find if the model adhere to the instruction, two choices are given ['thoughts', 'words']. if the model resort to the commonly used saying with 'words' then its using the parametric knowledge. We want the model to respond with 'thoughts' which tests if the model uses its intrinsic knowledge to respond.

The way we evaluate is by taking the probabilities of all tokens and get probabilities of the prompts with 2 choices (here: ['thoughts', 'words']) and compare those to yield a prediction. The results of the trained model on memo trap dataset that contrasts the final layer (24 - GPT2-Medium and 12 - GPT-Small) with layers 4, 6, 8, 12, 14, 18 are shown in Table 1. The results show poor performance . The paper shows an improved accuracy for open ended text generation tasks but for this task it shows poor performance. One possible reason could be , since the logits that are being contrasted specifically layer 6,12,18,14 carry different levels of abstraction and information, it is possible that the rebalancing of the scores does more damage than otherwise. Also, originally contrastive decoding techniques seems to do a better job of reducing repetition and increase coherence and diversity of the generated text. It is possible that in the process this might have increased the reliance on parametric knowledge which hampers the response.

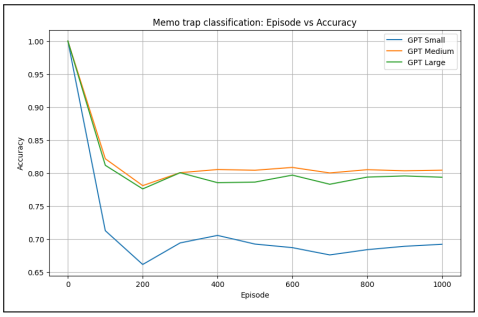

Figure 3: **Memo trap dataset: Inverse scaling on vanilla gpt**

Modus Tollens The modus tollens dataset puts forth prompts that are challenging to logically infer by the LLMs. A typical example prompt would look like:

Consider the following statements:
1. If Julia has a pet, then Julia has a dog.

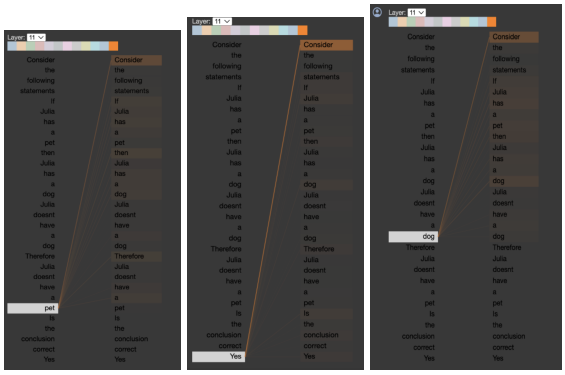

Figure 4: Attention weights of words on layer 12 of last attention head of GPT2-small

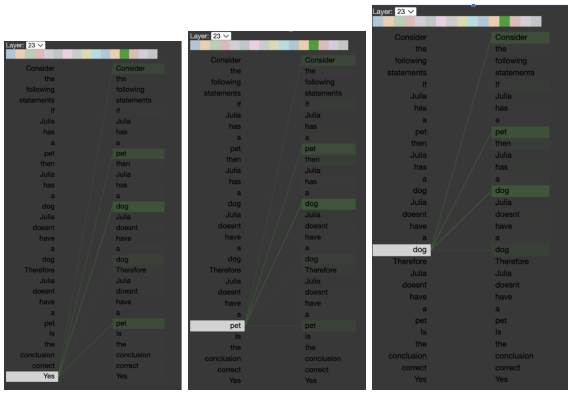

Figure 5: Attention weights of words on layer 24 of one of the attention head of GPT2-medium

2. Julia doesnt have a dog.
Conclusion: Therefore, Julia doesnt have a pet.
Question: Is the conclusion correct?
Answer:
The models need to answer Yes or No. The way we evaluate is by taking the probabilities of all tokens and get probabilities for "Yes" and "No" and compare those to yield a prediction The results of GPT-small+ACD and GPT-medium+ACD can be found in Table 3. Layer 4,6,8 information seems to help the model achieving an accuracy of around 50%.

ACD does improve the results of GPT2-small results without the augmentation. Looking at the attention weights give some interesting insights. The gpt-small arch 4 struggles to find associations between 'dog' and 'pet' which would help infer the correct answer. Almost all of the 12 heads struggle. GPT2-small+ACD seems to perform better on these prompts compared to GPT2-small. Whereas, the GPT2-medium 5 does strongly associate the words 'dog' and 'pet'. The conclusion 'Yes' also has higher weight w.r.t the words 'dog' and 'pet' while also taking into account words like 'doesn't' and 'then'. Moreover, most of the attention heads find some of these associations as well as opposed to gpt-small. ACD might have enhanced these associations while rebalancing the logit scores of these particular words. Logical reasoning of smaller models with ACD is worth exploring.

4 Conclusion and Future Work

We implemented the autocontrastive model and assessed its impact on the inverse scaling problem which yielded some interesting insights when tested with the Modus Tollens and Memo Trap datasets. ACD in essence re-balances the logit scores of the model to act as a watchdog for certain tasks. But despite this initial hypothesis that the autocontrastive approach would mitigate or resolve the inverse scaling issue, our findings indicate that the problem persists. This calls into question the notion that sometimes intermediate layers can track a vector of improvement and that we can extend this

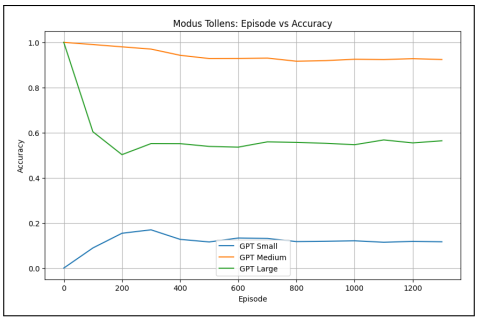

Figure 6: **Modus tollens dataset: Inverse scaling on vanilla GPT2**

Layer	Accuracy
4	0.5161
6	0.4919
8	0.5056

(a) Modus tollens results by contrasting layer 12 of GPT2-Small .

Layer	Accuracy
6	0.997
12	0.998
14	0.998

(b) Modus tollens results by contrasting layer 24 of GPT2-Medium.

Table 3: Comparison of Modus Tollens results with different layers of GPT models.

vector. It is plausible that the intricacies associated with the inverse scaling problem extend beyond the proposed solution. The inverse scaling problem despite implementing the autocontrastive model may be attributed to mismatched task complexities, sensitivity of the model architecture, and the need for fine-tuned hyperparameters. For future work we intend to address the persisting challenges in overcoming the inverse scaling problem by conducting extensive experiments with variations in model hyperparameters.

5 Individual Contributions

Name	Work Done	Contribution
Som Sagar	Implemented augmentations to the architecture	25%
Shyam Sundar	Conducted extensive research and did evaluations	25%
Roshan Varghese	Trained different models with different params for evaluations	25%
Suriya Prakash	Optimized codebase for efficiency	25%

Table 4: Work and Contribution Summary

References

Anonymous (2023). Contrastive decoding improves reasoning in large language models. In *Submitted to The Twelfth International Conference on Learning Representations*. under review.

Gera, A., Friedman, R., Arviv, O., Gunasekara, C., Sznajder, B., Slonim, N., and Shnarch, E. (2023). The benefits of bad advice: Autocontrastive decoding across model layers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10406–10420, Toronto, Canada. Association for Computational Linguistics.

Li, X. L., Holtzman, A., Fried, D., Liang, P., Eisner, J., Hashimoto, T., Zettlemoyer, L., and Lewis, M. (2022). Contrastive decoding: Open-ended text generation as optimization. *arXiv preprint arXiv:2210.15097*.