Using Hidden Tokens to Improve the Performance of Transformers

Leonard Kinsman, Samir Beall

April 2024

Abstract

Large language models (LLMs) such as Transformers often produce responses that do not align with input data, a problem known as hallucinations. Recent analyses attribute these inaccuracies to the models' limited computational capabilities, particularly in tasks requiring deep compositional logic. This paper proposes a novel approach to mitigate these limitations by incorporating hidden tokens into the outputs of Transformers. These tokens, not visible in the final output, serve as intermediate computational steps that theoretically enhance the model's output length and computational depth. By leveraging these hidden tokens, which emulate a Turing machine-like mechanism, the Transformer is expected to handle more complex calculations and extend its applicative scope in reasoning tasks. This study explores the potential of hidden tokens to improve the computational robustness of Transformers, addressing the key issues highlighted by prior research.

1 Introduction

1.1 Traditional Transformer Limitations

Transformers have revolutionized language processing and have since found a wide range of applications, many of which relate to problem-solving. Code generation is a good example of a problem-solving task that they are being trained to attempt. However, these models are frequently criticized for their tendency to produce hallucinations—responses that are inaccurate or inconsistent with given inputs. The issues seem to reveal them self as problems begin to require more complex reasoning, compositional understanding, or when working on longer inputs.

Transformers are classified within the complexity class logspace-uniform TC0 [MS23] [PNP24], which poses significant limitations on their computational abilities. This classification primarily restricts them to operations that can be performed in logarithmic space, essentially capping the amount of information they can process simultaneously and the depth of computation they can achieve. Given that many tasks in natural language processing, such as parsing complex sentences or handling nested dependencies, require more extensive memory and processing capabilities, this restriction fundamentally limits the performance of Transformers in more complex scenarios.

This restriction shows that they are provably unable to complete simple tasks, such as parity or matching parenthesis when given fixed input and output size.

1.2 Hidden Tokens

To address these challenges, this paper proposes an innovative modification to the traditional Transformer model: the integration of hidden tokens within the output sequences. These tokens are designed to act as placeholders or computational aids that do not appear in the final user-visible output but are instrumental during the processing phase. By functioning as discrete steps in a Turing machine-like computation process, hidden tokens theoretically extend the model's memory and enable it to simulate a greater depth of reasoning.

The introduction of hidden tokens aims to create a scaffold that allows Transformers to perform more complex operations that are typically beyond their reach. This approach enhances transformers' overall ability to tackle advanced computational tasks that require higher levels of abstraction

and logic. Later in this paper, we will discuss some ideas of how the network might use these tokens in an efficient and practical manner to allow it to perform simple operations.

When examining these hidden tokens, it is clear to see how they address the logspace memory issue cited by other papers. Models that are required to respond with a fixed or predetermined output size find themselves computationally constrained by the number of layers and heads that the model has. However, the addition of hidden tokens allows the model to perform more calculations on what it considers more challenging problems. These tokens can be used to artificially extend the model's "working memory" by serving as intermediate placeholders for complex computations. By integrating hidden tokens, we can expand the effective computational space available to the Transformer without altering the inherent architecture of its attention mechanisms. This allows the model to perform operations that resemble those in a Turing machine, where the hidden tokens act like the symbols on a Turing machine's tape—temporary and manipulable, facilitating more sophisticated operations and deeper reasoning than the model's standard configuration permits.

1.3 Practical Application

In essence, hidden tokens augment the model's internal computational resources, enabling it to store and manipulate additional information while processing an input. This expanded capability can significantly mitigate the effects of the logspace TC0 limitations by allowing Transformers to handle tasks that require longer chains of reasoning and more complex decision-making processes.

This has another practical implication, however. Modern LLMs are often parameterized by billions, or even sometimes, trillions of weights. This is due to the fact that this high parameter count allows the model to tackle more complex problems by using a deep network and many layers that are able to mimic complex thought. If we were to instead use the power of hidden tokens to give them this functionality, something we believe the model would be able to learn, we could downsize models and attain similar performance. Reducing the size of the model would have numerous benefits and would allow the model to become faster while not sacrificing the traits that we desire from more complex models.

2 Related Work

Many papers discuss the hallucinations and the inability of some models to solve even simple problem-solving tasks. All papers agree that they are inherently limited in the context of fixed input and output problems. Research has highlighted the computational limitations inherent to the Transformer architecture and pointed to this as a root cause of these issues. For instance, studies like those conducted by Sanford and others [PNP24] [SHT23] [DLS+23] demonstrate that Transformers struggle with tasks that demand sequential composition of operations or multi-step logical reasoning. These limitations are largely attributed to the finite memory and processing capabilities of standard Transformer layers, which restrict their ability to handle large domain functions or perform deep compositional tasks efficiently.

This paper constructs a novel mechanism to show that transformers can become turning complete due to the dynamic nature of our output space.

3 Methodology

3.1 Hidden Token Syntax

Our approach was to allow the hidden token to acts as a toggle switch, between our hidden state and the regular output state. As shown in figure 1. The first hidden token would activate "hidden mode" and the second would leave the hidden mode. We considered as well another syntax where we would use a hidden begin and hidden end token. This approach could theoretically lead to more complex functionality as it would lead to a more rich syntax however, we used the approach mentioned above as we believed it would be more easily discovered by the RL algorithm.

Figure 1: This token reads as 250 to the loss function / the output.

3.2 Dataset

We decided that arithmetic posed as a natural language problem would be a sufficient task to show the effectiveness of our new model. This is because arithmetic is fundamentally algorithmic. That is if the network can simply learn the basic algorithm to do arithmetic it can be an extremely simple model, when utilizing the hidden tokens to achieve this algorithmic behavior. This problem can be easily solved by a larger network, given that the dataset consists of manageable digit sizes. However, solving it with a smaller network that utilizes hidden tokens would effectively demonstrate the potential for greater parameter efficiency under the new hidden token regime. Specifically we choose to give the network multiplication problems which were less than 5 digits in the input, thus the maximum sized output was 10 digits.

3.3 Training Approach

When it came time to train the model, we decided to go with a pre-train and fine-tune approach. That is we trained a base network to do multiplication, then we would continue training with reinforcement learning. The reason we had to utilize reinforcement learning is there is no way to propagate gradient information to the hidden tokens.

3.3.1 Proximal Policy Optimization

Since the transformer network outputs a policy over tokens at each iteration, we decided that a policy gradient algorithm was the best approach for this problem. Specifically Proximal Policy Optimization has been a common approach for fine-tuning LLMs with RL [ZDG+23]. Here we will give a quick overview of the algorithm. PPO is a subset of actor critic models. These models have two basic components an actor, which deals with giving a policy for each state, and the critic which estimates the expected value of the return (discounted sum of all future rewards) given the current state. The actor critic algorithms basically boil down to this intuition, if the return we observed was better than expected increase the likelihood of the model taking this action, else decrease the

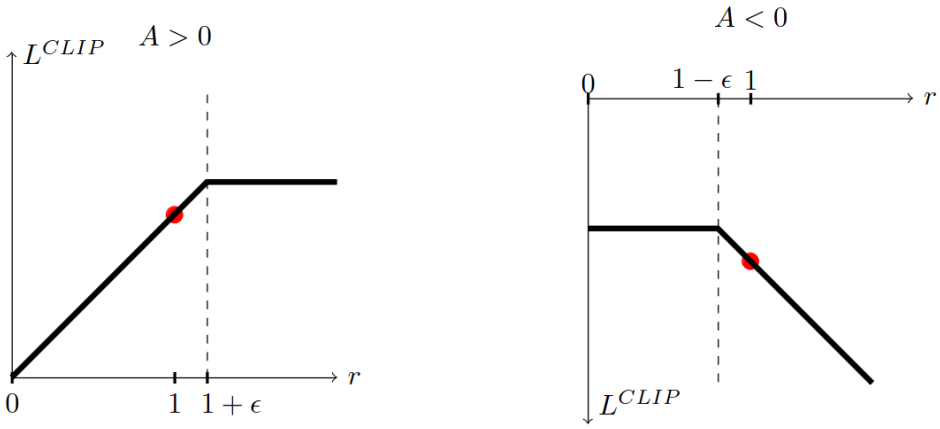

Figure 2: We aim to increase the L^{CLIP} function. In the case when $A > 0$, we performed better than expected so we increase L^{CLIP} by increasing the ratio (r) of our new policy to the old policy, thus making the action more likely. In the case when $A < 0$, we increase our objective by decreasing the likelihood, which would decrease the likelihood of the action that performed worse than expected.

likelihood of the model taking this action. Calculating the gradient with respect to the policy is a hard task since the policy is also the data generating process thus by changing the policy we are also chaining the likelihood of the data that we just used to change the policy. In other words we basically have a circular dependency. However luckily due to the policy gradient theorem, we can compute a value that is proportional to the gradient and use this for updates.

$$L_{clip}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}(a_t|s)_t}} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}(a_t|s)_t}}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \tag{1}$$

The loss in equation 1 is the PPO loss function as seen in [SWD+17]. The intuition described above can be described visually through the graph of the loss function. Note that since we are currently in the context of a reinforcement learning problem, we are attempting to maximize this function.

4 Experimental Setup

4.1 Model Architecture

Due to the fact that the model for both the actor and the critic must process the same information, we use a shared parameter approach. That is we have a intermediate transformer network that outputs vectors at each time step, and then use these vectors at each time-step as the input to the value head and actor head, which are both MLPs. An image of the architecture can be seen in the figure 3. The shared architecture is an encoder/decoder transformer. We must take advantage of the seq2seq functionality of the transformer to leverage the variable length decoder output. We also decided to bake the hidden token into our model. This way, we don't have to alter any weights or any of the architecture when we add the hidden tokens later on. This will save some time later on when we explain that we decided to use a supervised pre-training scheme.

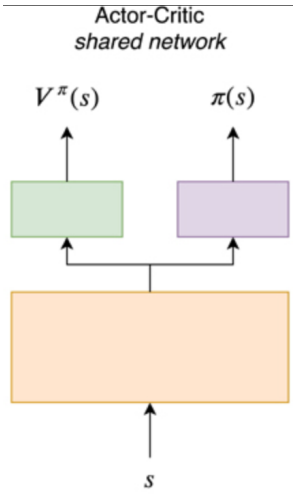

Figure 3: This figure depicts the shared base transformer model alongside the two value and policy MLPs, which transform the transformer output into the value and policy estimations.

4.2 Model Parameters

While we did have some hyper-parameters set in stone, such as the model sizes, we wanted to test out (2, 4) layer networks. We did preform a hyper-parameter search over all other parameters. Including the number of heads, the embedding dimension, the dimension of the feed-forward layer, etc. We did this via a random grid search. Our hyper-parameter search algorithm would perform 3 epochs, and if the model did not have sufficient convergence within those epochs, the algorithm would halt training and move on to the next set of parameters. For each set of hyper-parameters, we also tuned the learning rate. That is, for each of the models that the grid search produced, we would train multiple with different learning rates. This is because the learning rate is a very sensitive parameter that performs differently on different architectures and sized networks.

5 Experimental Results

Unfortunately, we struggled to get the Reinforcement Algorithm to produce improvements in our models. The RL algorithm fails to discover the utility of the hidden tokens. The reasons why we believe this to be true are described in the next section. We also will propose a few possible solutions that will possibly assist the algorithm in finding a way to utilize the hidden tokens. The reward and loss vs time graph displays this struggle. The reward graph is very noisy as to be expected with the reward of the algorithm on each epoch, however it shows little to no improvement. The average reward stays within the range of (0.2, 0.4). Since the model never gains the reward for using the hidden tokens properly this reward number essentially represents the accuracy of the model. In other words, the model after supervised learning is about 30%. This performance is acceptable given the complex task given to a small transformer. The transformer of 4 layers was expected to solve multiplication problems with results that could be 10 digits long. This model size was selected specifically to allow us to show the improvement of our models parameter efficiency and overall results.

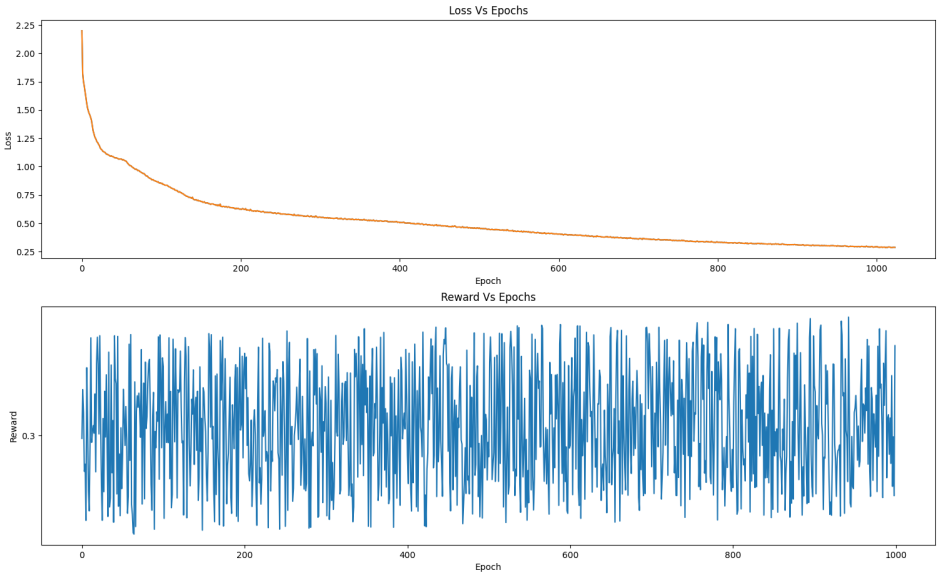

Figure 4: The graphs of the loss and average reward over time.

6 Discussion

Our project ended inconclusively since we were unable to train a model that effectively used hidden tokens to improve its performance. While we were able to train a lossy transformer model that could predict multiplication problems, we were unable to successfully improve the performance with RL.

While the architecture inherently includes the hidden tokens, during the supervised training, the model learns to never use them. The hope was that the RL would then allow the model to branch out, and a relaxed reward function would allow the model to learn how to use the hidden tokens on it's own.

However, the model never began to use hidden tokens, which is the result of the reward function's sparsity. It may not provide enough gradient for the model to learn to use hidden tokens effectively. This is a fundamental issue in this problem since we inherently want the reward function to be sparse to allow the model freedom to learn to use hidden tokens however it wants.

7 Conclusion

While this study was not successful in implementing the hidden tokens, it highlights some important considerations. It lays the groundwork for a powerful architecture. We are optimistic that this could be successfully implemented and would simply require a modified reinforcement learning process while maintaining a similar network architecture.

7.1 Future Work

1. Investigation into different reward structures could more effectively encourage the utilization of hidden tokens.
2. Exploration of curriculum learning approaches where tasks are gradually increased in complexity to better adapt the model to the use of hidden tokens.
3. Increasing requirements during supervised learning, such as requiring that the model outputs hidden tokens while initially training.

References

[DLS+23] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023.

[MS23] William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers, 2023.

[PNP24] Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On limitations of the transformer architecture, 2024.

[SHT23] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[ZDG+23] Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Secrets of rlhf in large language models part i: Ppo, 2023.

8 Appendices

8.1 Contributions

https://github.com/LeadFreeCandy/HiddenDigitTransformer

8.1.1 Lenny Kinsman

Lenny was responsible for taking charge of the creation of the dataset as well as the Supervised Learning aspect of training. He dealt with writing the hyper-parameter tuning system. Lenny also dealt with optimizing Samirs PPO algorithm to leverage batched computation, however the original implemention of the PPO algorithm was completed fully by Samir.

8.1.2 Samir Beall

Samir was responsible for researching and designing the PPO system and implementing the training framework for reinforcement learning. Samir was also fully responsible in finding the research and intuition relating to the computational complexity of the transformer model.