Project Report

ANLP | Monsoon 2024

nvm. | Team 50

Nanda Rajiv | Vaishnavi Shivkumar | Monish Singhal

Introduction

How complex does a model need to be to generate coherent language? Are there specific characteristics to the kind of data that extremely small models when exposed to can serve as sufficient for the model to learn? Why is a transformer architecture even necessary, can certain language tasks or datasets be reduced to some simpler form that makes simpler models perform well at them? These are questions that we aim to address through this project.

Problem Statement

1. Explore capabilities of small transformers by training different language models on the TinyStories dataset, and evaluating their performance on the generated stories.
2. Analysing the dataset and its representativeness
3. Explore whether linear models can achieve comparable performance to transformers for this specific dataset and task. Further analyse what that tells us about the data, the task, and language on a more philosophical level.
4. Examine the evaluation criteria put forth by the original TinyStories paper, and attempt to use traditional, robust and objective additional metrics in addition

Dataset

TinyStories is a synthetic dataset created using GPT-3.5 and GPT-4. The data is structured in short (2-3 paragraphs long) stories, comprising of short sentences, a consistent theme and simple vocabulary that is likely encountered by a 3 or 4 year old (as claimed by the paper).

An example from the dataset:

> Once upon a time, in an ancient house, there lived a girl named Lily. She loved to decorate her room with pretty things. One day, she found a big box in the attic. She opened it and saw many shiny decorations. Lily was very happy and decided to use them in her room.
> As Lily was decorating her room, the sky outside became dark. There was a loud thunder sound, and Lily got scared. She ran to her mom and said, "Mommy, the thunder is so loud!" Her mom hugged her and said, "Don't worry, it will pass soon."
> But the thunder did not stop. It got louder and louder, and the ancient house started to shake. Suddenly, the roof fell down on the pretty decorations. Lily was sad because her room was not pretty anymore. The end.

The TinyStories dataset is found on Hugging Face.

The Good, the Bad, and the Ugly: Issues with the Dataset

The dataset has several problems with it that are not addressed by the original author. Most of this arises from improper pre-processing, that does not seem to be addressed anywhere. The authors seem to have been extremely sloppy in this aspect, and paper their efforts through the scale of the dataset (clocking at over 200,000 samples).

There are several spurious characters present, including CJK ideographs and emojis. There are also several occurances of improper quotations (ASCII quotations, hyphens and ellipses are big examples, however, there are other minor infractions). ASCII quotations, for example, are present in 10000 samples. Similarly, em-dashes are present in 3000 samples. A small, but not insignificant

number of stories have the problem of ending far too early, or not ending at all. (An actual example from the dataset: " ". Yes, blank characters. Another story involves counting from 1 to 194 in words, because GPT-4 got stuck in a response loop.) Such data cleaning issues add up for smaller models and tokenizers, and make it require more resources.

Another issue that was noted was a significant part of the vocabulary fell outside the reported 1500 basic words that the dataset was composed of. Any tokenizer that was trained on the dataset had an approximate vocabulary size of 50,000 (BPE encoding), with several words not being accomodated even then. No child is expected to know conglomerate, for example.

There are other problems with the paper in of itself: the configurations required to reproduce are not very clear, and nor is the metric of GPT-eval itself, making it hard to check if the claims made are accurate or no. The stories in of themselves are of a comparable quality, on manual inspection.

Model Architectures

Basic Transformer

The *BasicTransformer* model described here is designed as a **decoder-only model** for **autoregressive tasks**. This means that it generates output one step at a time, with each step depending on the previously generated tokens. The model is well-suited for tasks where sequential prediction is essential, such as text generation or language translation. Below is a breakdown of the model's key components:

Embedding Layer

The embedding layer converts each input token (e.g., a word or symbol) into a high-dimensional vector, which allows the model to work with numerical representations of tokens that capture more than just the raw token index. These embeddings are scaled by the square root of the model dimension d_{model}, ensuring appropriate value scaling before further processing.

This layer is essential because it provides the foundational representation for the tokens that the model can process effectively throughout the sequence generation task.

Positional Encoding

In sequence generation tasks, the order of tokens is crucial. For example, in a sentence, the meaning can change based on word order. The transformer architecture, by design, does not inherently capture token order, so *positional encoding* is added to the token embeddings to provide information about the position of each token in the sequence.

The positional encoding ensures that the model can differentiate between different positions in the input sequence and generate sequences in the correct order.

Self-Attention Mechanism

A core feature of the model is the **multi-head self-attention** mechanism. This allows each token in the input sequence to "attend" to other tokens, meaning the model can focus on different parts of the sequence when making predictions. In autoregressive models, the self-attention is restricted so that each token only attends to the tokens that precede it, ensuring that the model generates output step-by-step without looking ahead.

The self-attention mechanism is key for understanding context, as it enables the model to take into account relationships between tokens, even if they are far apart in the sequence.

Feed-Forward Network (FFN)

After processing through the self-attention mechanism, the output is passed through a **feed-forward network** (FFN). This consists of two linear transformations with a non-linear activation function (such as ReLU) in between.

The purpose of the FFN is to further process the token representations, enhancing the model's ability to generate coherent and meaningful sequences by refining the relationships between tokens.

Layer Normalization

Layer normalization is applied before the self-attention and feed-forward layers. This technique stabilizes training by normalizing the output of these layers, ensuring that the values remain within a suitable range. Layer normalization smooths the learning process and reduces the risk of instability during training.

Final Projection Layer

Once the sequence has been processed through the self-attention and feed-forward layers, the model generates its final output. The output is projected back into the vocabulary space using a **final feed-forward layer**, which converts the internal token representations into logits. These logits represent the likelihood of each possible token in the vocabulary.

These probabilities are then used to decide which token should be generated next, completing the autoregressive process.

Autoregressive Nature

The **autoregressive** nature of this model means that tokens are generated one at a time, with each token being dependent only on the tokens that came before it. This ensures that the model does not have access to future information while generating a sequence. For example, during text generation, the model only has access to previously generated words, ensuring predictions are made in a forward-only manner.

This property is fundamental for tasks like language modeling, where predicting the next word or token based on the previous ones is central to the task.

MODEL PARAMETERS CALCULATION

In an decoder-only transformer model, the total number of parameters can be divided into several key components:

Embedding Layer: The embedding layer maps input tokens into high-dimensional vectors. The number of parameters is:

$$\text{Embedding\_params} = \text{vocab\_size} \times \text{d\_model}$$

Multi-Head Self-Attention (MHA): The self-attention mechanism consists of several learned matrices for query, key, value, and output projections. The total number of parameters is:

$$\text{MHA\_params} = 4 \times d_{model}^2 + 4 \times d_{model}$$

Feed-Forward Neural Network (FFNN): Each layer contains a two-layer feed-forward network. The number of parameters is:

$$\text{FFNN\_params} = 2 \times d_{model} \times \text{dim\_feedforward} + d_{model} + \text{dim\_feedforward}$$

Layer Normalization: Layer normalization is applied twice in each layer, once before the self-attention mechanism and once before the feed-forward network. The number of parameters is:

$$\text{LayerNorm\_params} = 2 \times 2 \times d_{model}$$

Final Feed-Forward Layer: The final feed-forward layer projects the model's output from d_{model} dimensions to the vocabulary size to generate logits for each token. The parameters are:

$$\text{Final\_FFN\_params} = d_{model} \times \text{vocab\_size} + \text{vocab\_size}$$

Total Parameters:

$$\begin{aligned}\text{Total\_params} = {} & \text{embedding\_params} \\ & + \text{num\_layers} \times (\text{MHA\_params} + \text{FFNN\_params} + \text{LayerNorm\_params}) \\ & + 2 \times \text{LayerNorm\_params} \\ & + \text{Final\_FFN\_params}\end{aligned}$$

—

MLP Layer (Linear Model)

The *LinearModel* is a simple multilayer perceptron (MLP) model with an embedding layer followed by a series of linear transformations. This model is used for tasks where sequential information is important, and operates on a fixed-size sequence length.

Embedding Layer

The embedding layer maps input tokens into high-dimensional vectors. The model uses this to convert input tokens into a suitable form for further processing.

Linear Layers

The model includes several linear layers. The first linear layer reshapes the embeddings and applies a masked linear transformation, ensuring that each token only attends to previous tokens in the sequence. An activation function (ReLU) is optionally applied before the final output layer.

Masked Linear Transformation

A key feature of this model is the *masked linear transformation*, where a mask is applied to the weight matrices in the linear layers to enforce the autoregressive nature of the model, ensuring that future tokens are not attended to.

Final Linear Layer

The final linear transformation projects the output from the hidden layer back to the vocabulary size, producing logits for each token in the vocabulary.

MODEL PARAMETERS CALCULATION

The parameters of this MLP model are primarily determined by the embedding layer, the linear layers, and the final output layer. Each component contributes a certain number of parameters, and the total number of parameters is calculated accordingly.

Embedding Layer The number of parameters in the embedding layer is:

$$\text{Embedding\_params} = \text{num\_tokens} \times d$$

Linear Layers The first linear layer has parameters given by:

$$\text{Linear\_1\_params} = T \times d \times T \times d$$

The second linear layer, if used with activation, has:

$$\text{Linear\_2\_params} = d \times d$$

Final Linear Layer The final output layer parameters are:

$$\text{Final\_params} = d \times \text{num\_tokens}$$

Thus, the total number of parameters for the model can be calculated by adding the contributions from each component.

Auto-Regression is All You Need: Analysis of Model Architectures

In this work, we compare two different model architectures: small transformers and Multi-Layer Perceptrons (MLPs). The goal is to investigate whether more complex architectures are truly necessary for certain tasks or whether simpler models can perform just as well. To do this, we train both models on the TinyStories dataset, which consists of simple stories with clear, structured language. By evaluating the performance of both models, we aim to explore the role of architecture versus the power of autoregressive training.

What does this tell us about the dataset?

This tells us that TinyStories models simple language in such a way that even uncomplicated models (like MLPs) and small transformers can generate cohesive sentences. The sentence structure is not overly complex, yet there is still distinct structure in the dataset. Additionally, next-word prediction doesn't necessarily require advanced logical reasoning, but there is an emergence of some reasoning capability that is simple enough to be modeled by these basic models.

What does this tell us about auto-regression?

One key finding is the sheer power of autoregression. Tasks like next-word prediction are fundamentally different from classification tasks. With each iteration, we provide more information to our model. In this way, by constantly updating the input and labels with sampled information, we can model complex non-linear functions and patterns through the simple task of autoregression. This process can be thought of as a form of Chain-of-Thought reasoning. If there is enough data with distinct and significant structure (like TinyStories), performance can improve further. The architecture we used to train is less important than the data we use and the auto-regressive learning setup.

Why is this cool?

The foundation of next-word prediction lies in the auto-regressive nature of these models. This process is powerful enough to generate coherent sentences even with simple models. This is incredibly remarkable. Sometimes, the model's structure matters less than its ability to learn from the data.

Why the Linear Model Works

Despite its simplicity, the MLP with 775M+ parameters performs remarkably well on the TinyStories dataset, suggesting that the linear model's effectiveness comes from the structure of the dataset itself rather than the complexity of the architecture. The fact that such a simple model can generate coherent, structured text emphasizes the power of autoregressive training over intricate model design. The lack of activation functions and other complexities in the linear model allows it to focus solely on the essential task: predicting the next word based on prior context. In simple, constrained domains like TinyStories, simpler architectures can achieve impressive results without requiring the computationally expensive and more intricate layers typically found in larger models.

Why Small Transformers Still Work

Small transformers, though more complex than linear models, also perform very well on this task. The key factor is the multi-head self-attention mechanism, which allows even smaller models to capture rich contextual relationships between words. This architectural feature enhances the model's ability to understand dependencies between words in a sequence, making it more robust at generating coherent sentences. Additionally, despite their relative simplicity, these small transformers benefit from the autoregressive nature of training, where they predict one word at a time and refine their understanding with each new prediction. Therefore, even small transformers can produce high-quality results for structured tasks like TinyStories, without needing to scale up excessively in terms of size or computational cost.

Score me like one of your French girls: Using Traditional Machine Translation Metrics to Evaluate our Models

In this section, we aim to look at objective criteria for evaluating the models by comparing sentences generated by them. These include scores such as Rouge, BLEU, BertScore, ChrF, etc., to compare similarities between the gold and generated completions for the TinyStories dataset by the models we trained.

BERTScore

BERTScore measures the similarity between the predicted completions and actual texts by using contextual embeddings from BERT. It computes precision, recall, and F1-score based on the cosine

similarity between the token embeddings.

The goal is to capture semantic similarity by comparing the deep, context-aware representations of words in sentences, rather than relying on surface-level matches (e.g., exact word overlap). The score is computed in two variants: raw (without rescaling) and with rescaling based on a baseline model.

BLEU (Bilingual Evaluation Understudy)

BLEU is a precision-based metric that compares n-grams (e.g., unigrams, bigrams) in the generated text (completions) to n-grams in the reference text (actuals). It gives a score between 0 and 1, with higher values indicating better quality.

BLEU evaluates the overlap of n-grams between the predicted and reference texts, rewarding higher overlaps. It emphasizes precision and punishes overly short or overly repetitive output. It is widely used in machine translation.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE is a set of metrics that measure recall (how much of the reference text is captured by the generated text) using various n-grams and longest common subsequences (LCS). The common versions of ROUGE are ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L (LCS).

The focus is on recall rather than precision, meaning it rewards completions that contain more of the important content from the reference text. It is useful for tasks like summarization, where capturing the essence of the source is important.

ChrF (Character F-score)

ChrF is a character-level evaluation metric that calculates the F-score (harmonic mean of precision and recall) for character n-grams (e.g., bigrams or trigrams). It evaluates similarity based on characters instead of words.

ChrF is effective for languages with complex morphology or when word boundaries are less clear. It focuses on character-level precision and recall, which can be especially useful in tasks like machine translation or text generation, where minor errors in spelling can affect word-level metrics. The `beta` parameter adjusts the weighting of recall versus precision, with higher values giving more weight to recall.

Contrast to the Original Paper

The original paper relies on GPT-eval, which uses the GPT-4 model to grade sentences out of 10 on Grammar, Creativity and Consistency.
Some of the concerns with this is that these are not objective or frozen. A sentence once evaluated a certain score may obtain a different score a few days later, bringing into question how reproducible and robust the evaluations of these models are by using GPT-eval.
(Also we don't have money for GPT API Access :))

If some more objective method was available for comparing models, it would be a good addition to the GPT-eval method. In addition, these scoring methods are much more interpretable, so one can comment on what exactly certain models are doing right or wrong. For example, a high ChrF score but a low BERTscore could be indicative that the model is making typos (not likely, but imagine it as an example), or using similar-looking words that mean different thing. For example, the sentences "I'm indifferent." and "I'm different." would hypothetically get a high ChrF but low BERTScore.

Limitations of MT-focused Scores

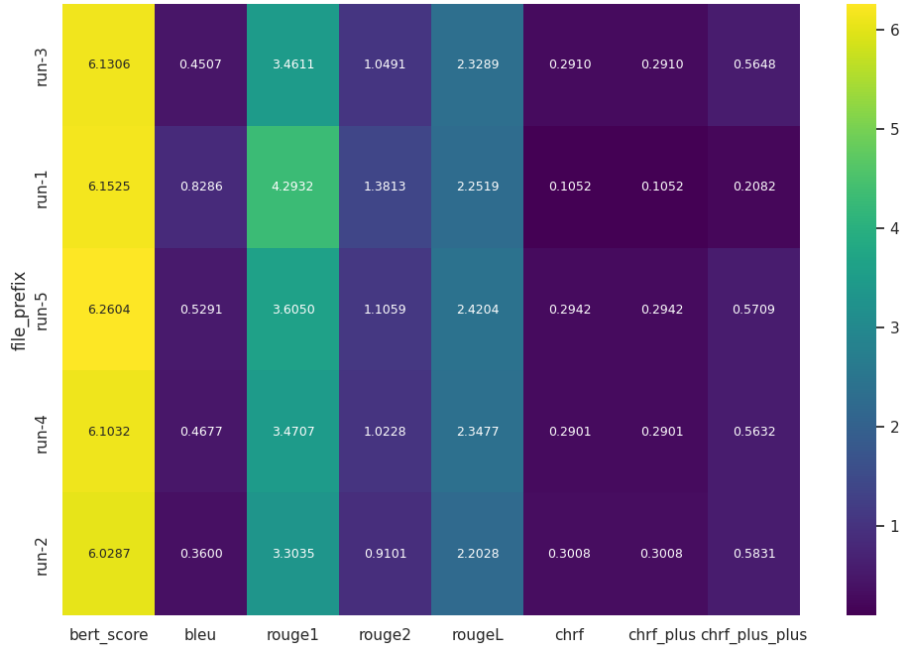
As is clear from the name, they are used largely for machine translation. Standards are different when it comes to completing stories (the goal of our models) and machine translation. The latter

demands a less flexible set of predictions from the models as compared to generation, which is the task at hand. Making judgments about creativity is not effective, and may actually be inversely proportional to these scoring methods.

Between Model Comparison

We computed the above metrics and scaled them to a 10-point range. Following are the hyperparameter settings:

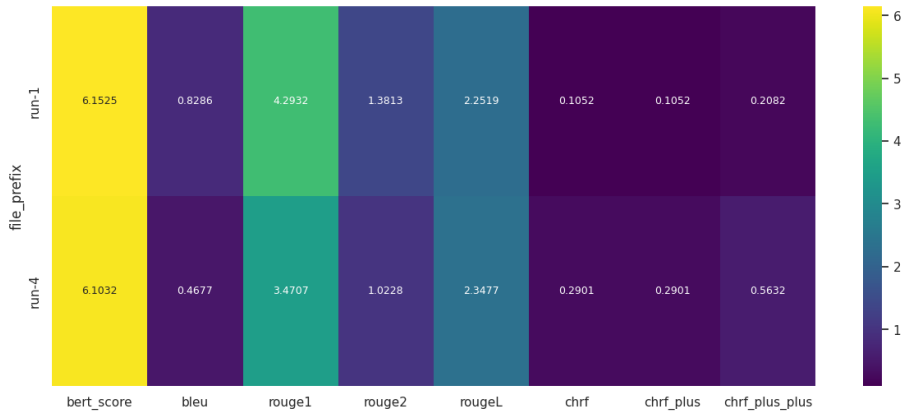
- Run 1: hidden dimension 512 | layers 6 | heads 8 | data 10% | epochs 10
- Run 2: hidden dimension 1024 | layers 6 | heads 8 | data 10% | epochs 10
- Run 3: hidden dimension 256 | layers 6 | heads 8 | data 10% | epochs 10
- Run 4: hidden dimension 512 | layers 6 | heads 8 | data 100% | epochs 1
- Run 5: hidden dimension 768 | layers 12 | heads 8 | data 100% | epochs 2



Comparison between All Models Trained

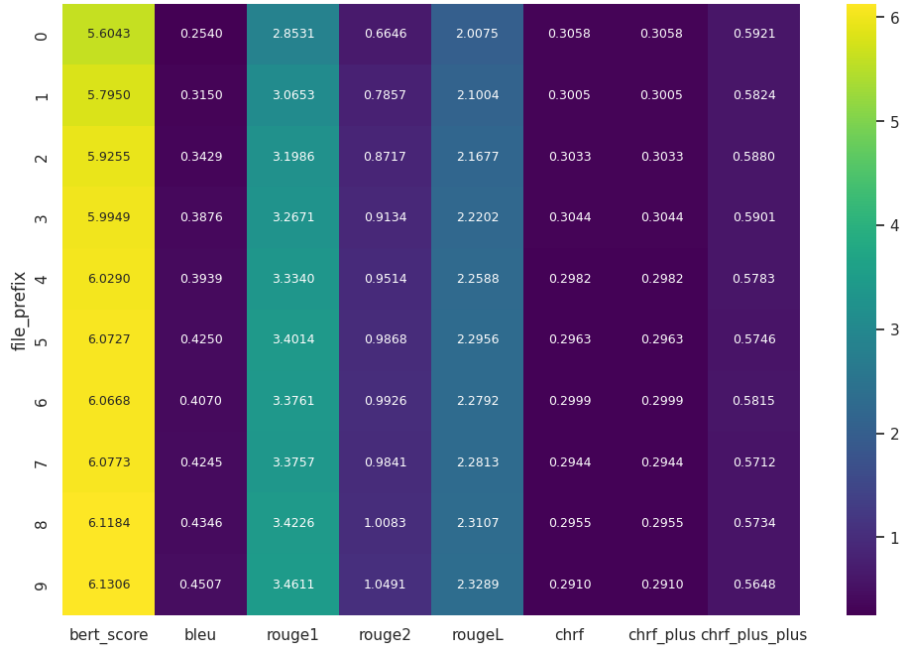
How Much of a Role does Number of Training Epochs Play?

We compare 2 models that have the same architecture and hyperparameter settings, except for the size of data and number of epochs. For one model, we used 10% of the data and ran for 10 epochs, and the other with 100% of the data for 1 epoch.



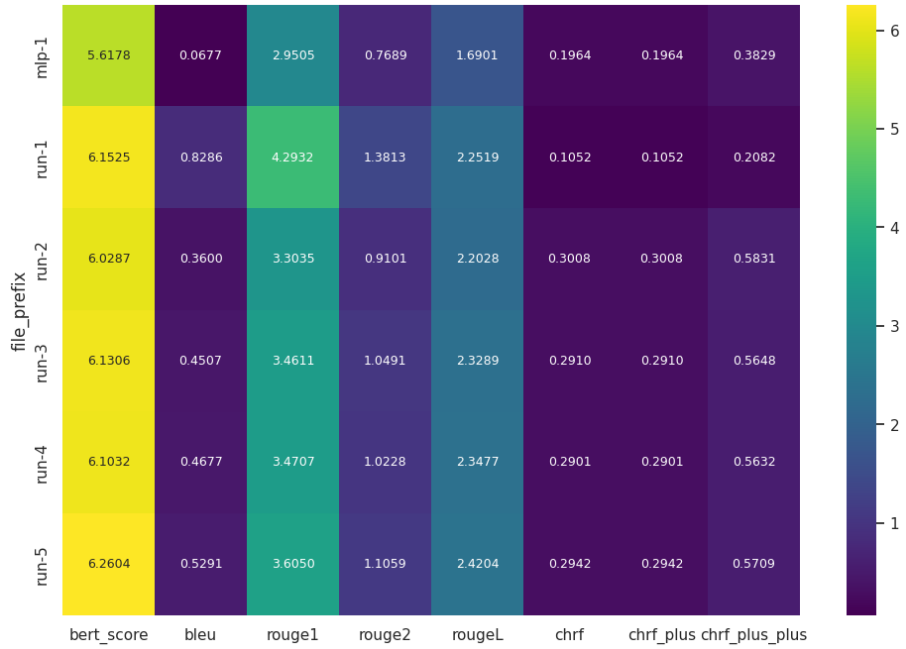
Comparison between Models Trained for Different Number of Epochs

We also compared the performance of one model over each of the 10 epochs it was trained for, by using model checkpoints.



Comparison between Epochs of the Same Model

Comparing the five models with the MLP, we can observe that the MLP performs worse compared to the other transformer models by these measures.



Comparison between MLP and the Transformers

Results

In this section, we present the results of our experiments with different configurations of transformer and linear models. Below are the configurations we used for training and evaluation.

Transformer Configurations

We evaluate the following transformer configurations:

Model Name	Vocab Size	d_{model}	Num Encoder Layers	Nheads	Epochs
TR-MODEL-1	8192	512	6	8	10 (10% Data)
TR-MODEL-2	8192	512	6	8	1 (100% Data)
TR-MODEL-3	8192	256	6	8	10 (10% Data)
TR-MODEL-4	8192	768	12	8	1 (100% Data)

Table 1: Transformer Configurations

Linear Model Configurations

The configurations used for training the linear model are as follows:

Model Name	Learning Rate	Weight Decay	Epochs	Context Length
LIN-MODEL-1	0.0005	0.1	1	64
LIN-MODEL-2	0.0005	0.1	2	64

Table 2: Linear Model Configurations

Parameter Count

Model Name	Number of Parameters
tr-model-1	27.313M
tr-model-3	12.093M
tr-model-4	78.76M
lin-model-1	775M

Table 3: Model Names and Their Corresponding Number of Parameters

Loss Plots

To analyze the performance of both the transformer and linear models, we present the following loss plots:

Training Loss

Below are the training loss plots for each configuration, presented one after the other:

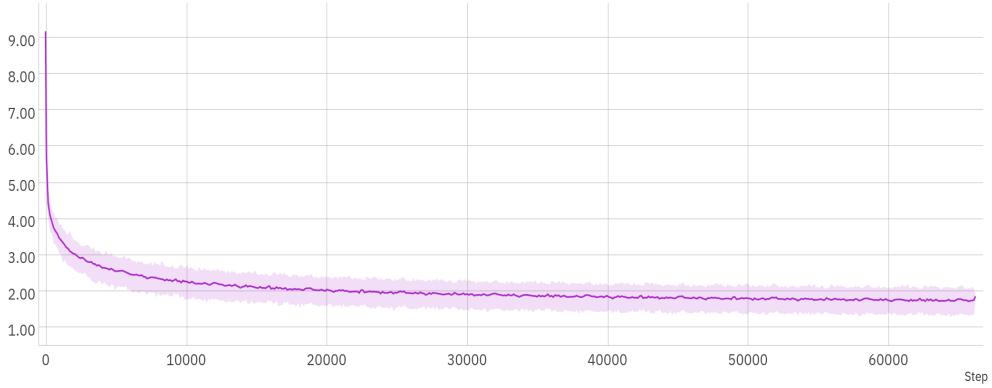


Figure 1: Training Loss for TR-MODEL-1

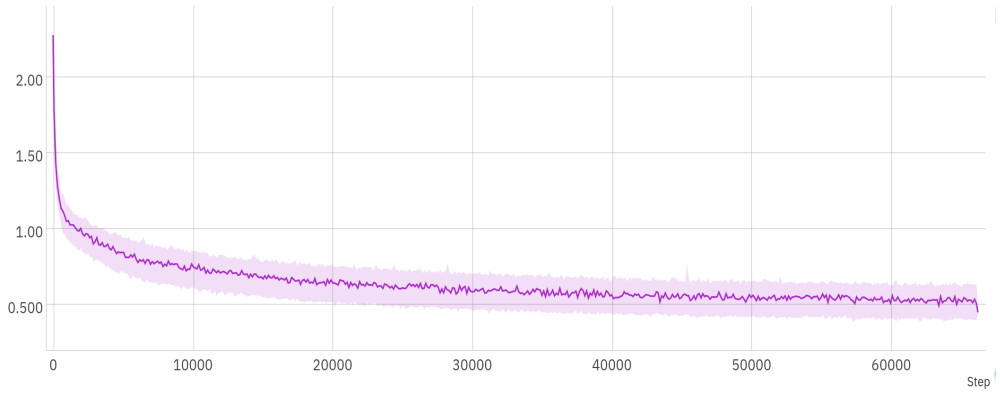


Figure 2: Training Loss for TR-MODEL-2

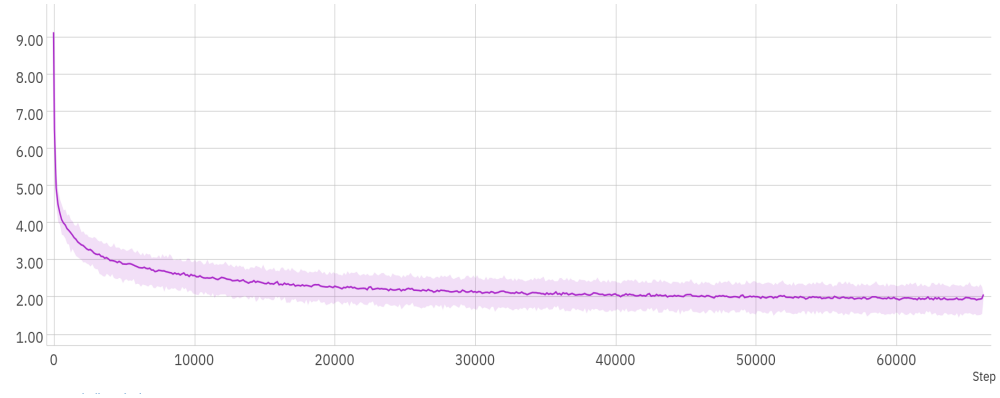


Figure 3: Training Loss for TR-MODEL-3

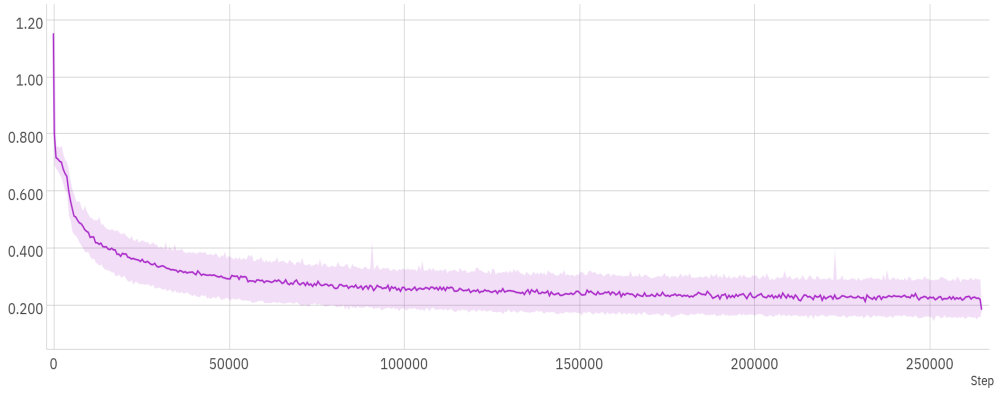


Figure 4: Training Loss for TR-MODEL-4

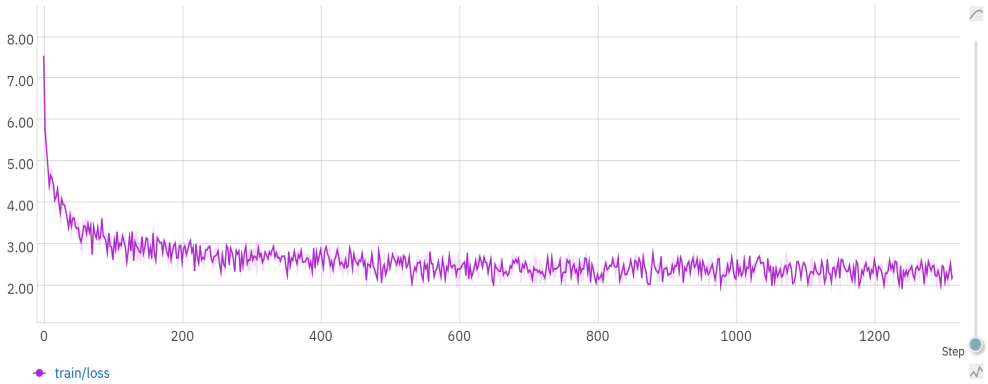


Figure 5: Training Loss for Linear Model (1 Epoch)

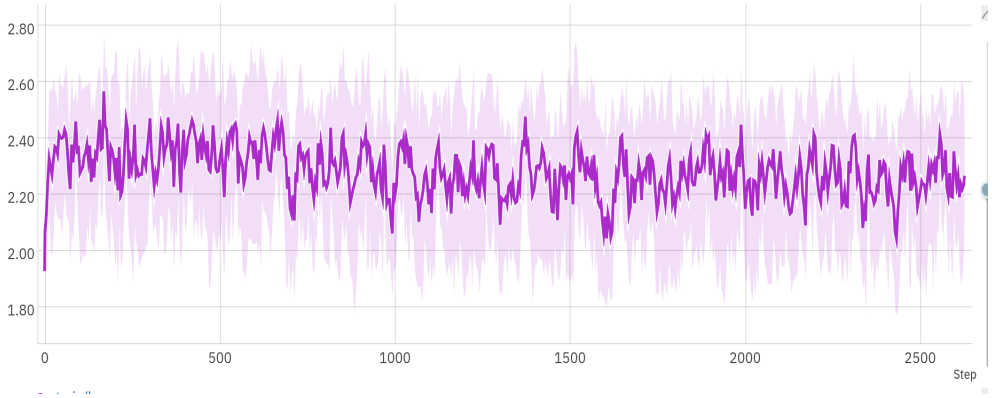


Figure 6: Training Loss for Linear Model (2 Epochs)

Evaluation Loss

Below are the evaluation loss plots for each configuration, presented one after the other:

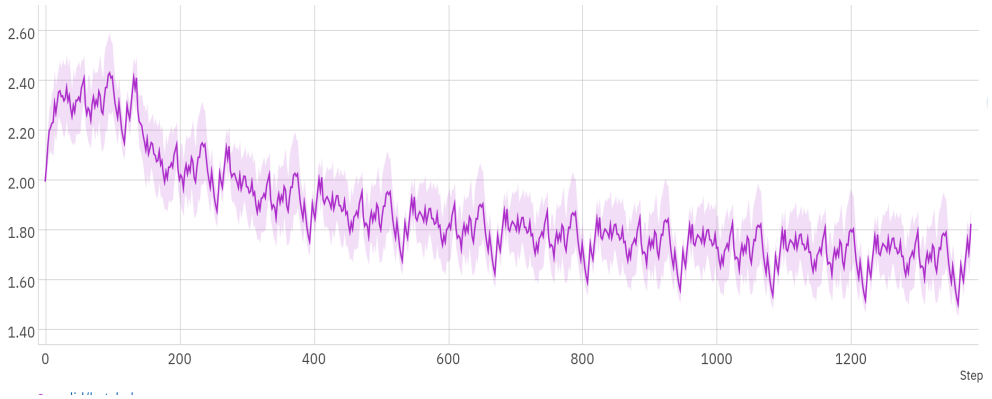


Figure 7: Evaluation Loss for TR-MODEL-1

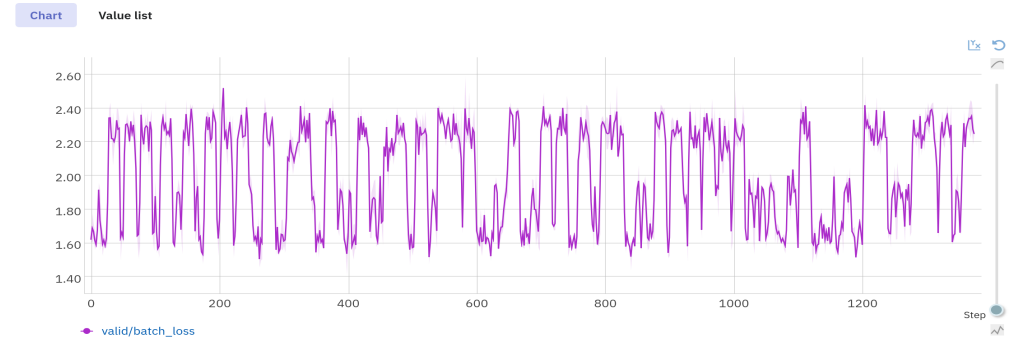


Figure 8: Evaluation Loss for TR-MODEL-2

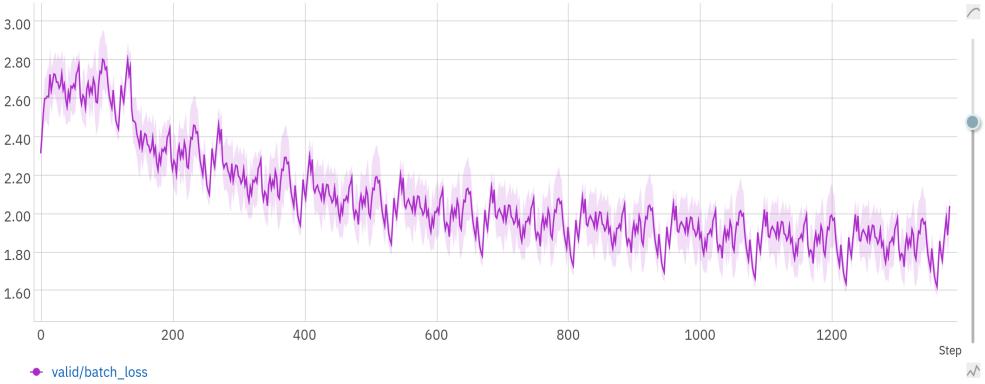


Figure 9: Evaluation Loss for TR-MODEL-3

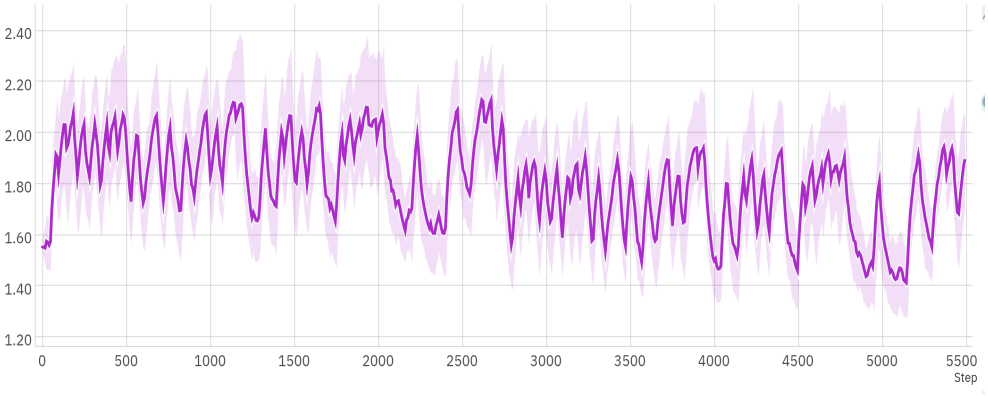


Figure 10: Evaluation Loss for TR-MODEL-4

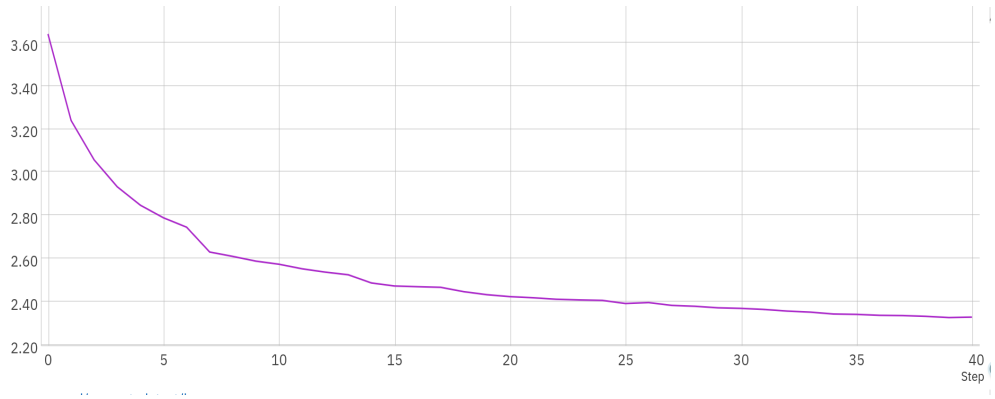


Figure 11: Evaluation Loss for Linear Model (1 Epoch)

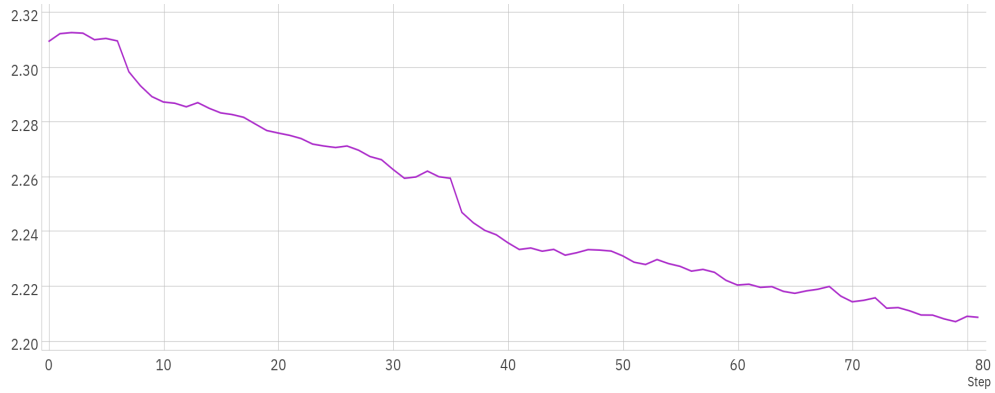


Figure 12: Evaluation Loss for Linear Model (2 Epochs)

Analysis:

1. *TR-MODEL-1*: The model with a large dimensionality ($d_{model} = 512$) and 6 encoder layers, trained on 10% of the data for 10 epochs, should be able to effectively learn the patterns in the data and show a steep decrease in both training and evaluation loss.
2. *TR-MODEL-2*: The model with the same dimensionality ($d_{model} = 512$) and encoder layers as TR-MODEL-1, but trained on 100% of the data for only 1 epoch, may struggle to fully learn the complex patterns in the data, resulting in a slower decrease in both training and evaluation loss compared to TR-MODEL-1. Here we see that evaluation loss oscillates. This potentially could be due to only training it for 1 epoch.
3. *TR-MODEL-3*: The model with a smaller dimensionality ($d_{model} = 256$) and the same number of encoder layers as TR-MODEL-1, trained on 10% of the data for 10 epochs, may have more difficulty learning the intricate patterns in the data, leading to a slower decrease in training and evaluation loss compared to TR-MODEL-1. Here we do see loss decreasing in both evaluation and training, which is impressive considering the model is of size 12M parameters.
4. *TR-MODEL-4*: The model with the largest dimensionality ($d_{model} = 768$) and 12 encoder layers, trained on 100% of the data for only 1 epoch, may be able to quickly learn the general patterns in the data, resulting in a decrease in training loss. Similar to the other model trained only for one epoch, we don't see much of a convergence in the evaluation loss. This tells us that it's better to see lesser data for longer, rather than more data for a short amount of time. We can also speculate that the distribution of the data is near-uniform.
5. *LIN-MODEL-1*: Although we do see fluctuations, we see that the training loss decreases. Figure 11 shows the evaluation loss for a linear model trained with 1 epoch, which decreases sharply initially and then gradually flattens out.
6. *LIN-MODEL-2*: We see almost no changes in the training loss. This could be indicative of the model not learning. This could mean that the model has already identified all necessary

patterns in 1 epoch. Figure 12 depicts the evaluation loss for a linear model trained with 2 epochs, which exhibits a similar decreasing trend as the 1 epoch model but with a lower final loss value.

Conclusion

In conclusion, this paper acts as an exploratory analysis of the three pillars of neural language modelling: dataset, models and evaluation. We see how the dataset is simple, yet efficient at representing structure. We also see how the dataset has redundant information, and might be hard to generalize to regular language. Similarly, we test the bounds of "small" models, by analyzing various models. We also push the bounds of what a "simple" model is by looking at a linear model. We finally look at quantitative evaluation metrics which we leverage to tell us more about the models and their performance on different tasks within language modelling.

References

1. Clark, K., Khandelwal, U., Levy, O., & Manning, C. D. (2019). What Does BERT Look at? An Analysis of BERT's Attention. In T. Linzen, G. Chrupała, Y. Belinkov, & D. Hupkes (Eds.), Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP (pp. 276–286). Association for Computational Linguistics. https://doi.org/10.18653/v1/W19-4828
2. Eldan, R., & Li, Y. (2023). TinyStories: How Small Can Language Models Be and Still Speak Coherent English? https://arxiv.org/abs/2305.07759
3. Li, J., Chen, X., Hovy, E., & Jurafsky, D. (2016). Visualizing and Understanding Neural Models in NLP. In K. Knight, A. Nenkova, & O. Rambow (Eds.), Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 681–691). Association for Computational Linguistics. https://doi.org/10.18653/v1/N16-1082
4. Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., Lai, L., & Chandra, V. (2024). MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. https://arxiv.org/abs/2402.14905
5. Vig, J. (2019). A Multiscale Visualization of Attention in the Transformer Model. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 37–42. https://doi.org/10.18653/v1/P19-3007
6. Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., & others. (2023). Efficient large language models: A survey. ArXiv Preprint ArXiv:2312.03863, 1.
7. Warstadt, A., Mueller, A., Choshen, L., Wilcox, E., Zhuang, C., Ciro, J., Mosquera, R., Paranjabe, B., Williams, A., Linzen, T., & Cotterell, R. (2023). Findings of the BabyLM Challenge: Sample-Efficient Pretraining on Developmentally Plausible Corpora. In A. Warstadt, A. Mueller, L. Choshen, E. Wilcox, C. Zhuang, J. Ciro, R. Mosquera, B. Paranjabe, A. Williams, T. Linzen, & R. Cotterell (Eds.), Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning (pp. 1–34). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.conll-babylm.1
8. Wu, R., & Papyan, V. (2024). Linguistic Collapse: Neural Collapse in (Large) Language Models. https://arxiv.org/abs/2405.17767
9. Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2024). Judging LLM-as-a-judge with MT-bench and Chatbot Arena. Proceedings of the 37th International Conference on Neural Information Processing Systems.
10. E. Malach, "Auto-Regressive Next-Token Predictors are Universal Learners," arXiv, 2023. [Online].
11. T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating Text Generation with BERT," arXiv, 2020. [Online].

12. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A Method for Automatic Evaluation of Machine Translation," Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002), 2002, pp. 311-318. [Online].