A Re-Implementation and Exploration of the Transformer Architecture for Text Generation

Albin Kempe
albinwk@kth.se

Marcus Oscarsson
moscars@kth.se

Elin Inoue
einoue@kth.se

Abstract

In recent years, Transformer-based models have garnered a lot of attention for their outstanding performance in various sub-fields of deep learning, being especially prevalent in the field of natural language generation. This study re-implements a decoder based Transformer and investigates several aspects of the Transformer architecture and evaluates the effect of distinct tokenization schemes and data augmentation on NLP tasks. Our findings suggest that models utilizing byte-pair encoding often generate more coherent results compared with a character level model and we find that larger models with single layer close to the size of state-of-the art models tend to outperform smaller ones, even in shorter training runs.

1 Introduction

Natural language processing has experienced a transformative shift with the advent of deep learning models. Initially, Recurrent Neural Networks and Long Short-Term Memory networks led the way, capturing temporal dependencies in text and considerably outperforming traditional statistical methods. However, their sequential nature limits their ability to capture long-range dependencies, leading to suboptimal performance in tasks such as text generation.

Transformer models, initially developed in 2017, diverged from the recurrent paradigm and gave rise to a major breakthrough. By harnessing the power of the self-attention mechanism, Transformers are capable of simultaneously processing entire sequences of tokens. This allows them to better capture long-range dependencies in text and significantly boosts the coherence and quality of generated text. The recent excitement in natural language processing (NLP) has largely revolved around these Transformer-based models, and their unprecedented performance in text generation tasks has paved the way for practical applications like machine translation, automated report generation, and conversational agents.

The aim of the project detailed in this report was to explore the state-of-the art in modern text generation through the construction and examination of a Transformer network. The network was implemented from scratch and trained with different hyperparameters and data representations to gain a better understanding of the architecture and the influence of different training settings. Overall, the results indicate that large models perform better and that sub-word-level models outperform character-level models in term of generated text quality.

2 Related Work

Transformers consist of two components: encoders and decoders. While models that use the transformer architecture can include both components, they can also exclusively use one of them. In those cases, the models are called decoder or encoder models [5]. Decoder models are specialized for natural language generation (NLG) tasks.

One example of a decoder model is the Generative pre-trained transformer (GPT) model [8]. GPT-2 [9], GPT-3 [1] and GPT-4 [7] are significant GPT models that, upon respective launch, have achieved state of the art NLP benchmarks results. GPT models shows great results in generating coherent and textually relevant human-like text.

Other decoder models include Transformer-XL [2] and CTRL [3]. Transformer-XL removes the restriction on context length, the number of previous data points the model considers when generating output, which greatly increases its performance in comparison to regular transformers. CTRL is trained with control codes which allows the user to control the output generation by for example providing guidelines for style or content.

3 Data

We have used two datasets to train and test our models: the WikiText-2 dataset and a collection of mostly fictional books. WikiText-2 is a publicly available dataset consisting of several quality-assessed articles in English from wikipedia. It contains approximately 33,000 different words and is divided into a training set of over 2,000,000 tokens (600 articles), and a testing and validation set of roughly 200,000 tokens each (60 articles) [6]. The book-dataset was constructed through handpicking a total of 28 books (Table 7) to match the amount of data in the WikiText-2 dataset. The books were obtained from the Project Gutenberg website, which mainly provides access to older literary work.[1] Only books released prior to 1850 were included in the dataset, to limit the amount of archaic English. The decision to mix the two datasets was an attempt to make the model more general and less specific to a certain genre of text.

3.1 Preprocessing

The articles in the WikiText-2 dataset have been cleaned up and tokenized whereas the books have not been subject to any prior preprocessing. Regardless of this, we performed some additional preprocessing to both datasets. Upon inspection, it was observed that both datasets contained some special characters and tokens. Some examples of these are highlighted in the excerpt from the WikiText-2 dataset below:

```
Senjō no Valkyria 3 : <unk> Chronicles ( Japanese :
戦場のヴァルキュリア3, lit . Valkyria of the Battlefield 3 )
```

These types of characters were considered inhibitory for the model's ability to learn proper English, and were thus removed. In doing so, there is a risk that the model learnt some other unwanted behaviour instead. For example, removing the ō character had the effect that words like "Kōsaku" became "Ksaku", which is clearly not a valid character sequence in the English language. Nevertheless, removing the character prompted the model to incorrectly learn that it was. An alternative solution would have been to change all occurrences of special characters to some other reasonable representation (such as o instead of ō), however, in the essence of time we opted to simply remove them entirely. Since the dataset is fairly large and since the removed characters are rarely used in the English language our approach likely had limited negative effect on the model's performance.

Furthermore, all meta-information such as table of contents, author and publisher, was removed from the books.

4 Methods

4.1 Transformer Architecture

4.1.1 Decoder block

Our model has a decoder only architecture. As can be seen in Figure 1, a decoder consists of a multi-head attention layer and a feed forward neural network. These parts allow the decoder to look at previous words in a sequence and predict what the next word should be.

[1] Project Gutenberg. https://www.gutenberg.org/

4.1.2 Self-Attention

Self-attention is the crux of how Transformers learn. Given a sequence of input tokens, each token is mapped to a set of three vectors: a query vector, a key vector, and a value vector, through multiplication with learned weight matrices. The self-attention mechanism then computes the relevance of every token to every other token in the sequence. This is achieved by calculating the dot product between the query vector of a particular token and the key vector of every other token, followed by a softmax operation to obtain attention scores. These scores reflect how much each token should contribute to the new representation of the considered token. Equation 1 shows this in mathematical notation[10]. Where Q is the matrix of queries (one row per token), K is the matrix of keys and V is the matrix of values. d_k is the dimension of the key, query and value vectors.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

4.1.3 Multi-Headed Attention

The part of the decoder where self-attention takes place is commonly referred to as a head. Transformers employ multi-headed attention which means that the self-attention mechanism is performed multiple times for the same input but with different weight matrices for Q, K and V. In other words, each decoder block consists of several heads. The output from all heads in the decoder are concatenated together and passed through a linear layer to generate a single attention score. The use of several heads allows the decoder to consider different aspects of the input when calculating the attention scores [10].

4.1.4 Architecture

Our implementation is a re-implementation of the decoder presented in the original Transformer paper[10], with slight modifications. As discussed in the paper "On Layer Normalization in the Transformer Architecture"[11], we adopt a pre-norm formulation approach. In this configuration, we apply layer normalization prior to the execution of the respective computational units, a procedure empirically demonstrated to yield better network stability. Figure 1 shows the architecture implemented in this paper.

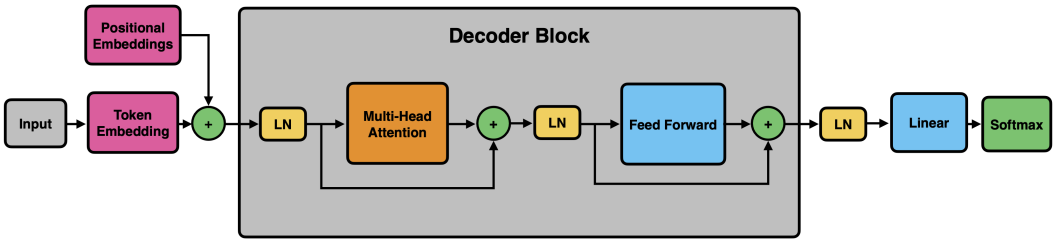

Figure 1: Decoder only Transformer architecture that is implemented with this paper. The number of sequentially stacked Decoder Blocks is refered to the number of layers in our network.

4.2 Evaluation

To evaluate our model we will record the token level cross entropy loss. That is for each token prediction our model makes we compute equation 2 where c is the probability distribution over the vocabulary V that our model produces and $\hat{c}$ is the one-hot encoded target token.

$$H(c, \hat{c}) = -\sum_{i=1}^{|V|} c_i \log(\hat{c}_i) \tag{2}$$

4.3 Tokenization

The Transformer architecture is general and may handle many different basic tokens. Thus we will initially use a character level tokenization scheme for model evaluation.

In an effort to further increase model performance we will experiment with switching to a byte pair encoding (BPE) tokenizer. The BPE algorithm starts with a set of basic tokens (commonly the letters that occur in the corpus) and then works by iteratively finding the most common bigram of tokens and creating a new token from this bigram. Then repeating this step until the desired vocabulary size is reached. Since the vocabulary size differs between the the BPE model and the character level model, doing a direct quantitative comparisons is difficult. Qualitative assessments can however be made.

4.4 Data Augmentation

In NLP data augmentation can be performed using a number of methods. In this project data augmentation is performed using the back translation method. This method involves translating text to another language and then translating it back to the original language to create new representations of the data. In our case, both the WikiText-2 and the book dataset were translated to French and then translated back to English. The generated paraphrases were then used to extend the datasets. Other data augmentation methods in NLP include replacing some words with their synonyms and masking words and exchanging them for predictions made by a Language model [4].

4.5 Other Apporoaches to NLG

Other prominent approaches to NLG include variants of the RNN architecture, with Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) networks being two of the most prevalent types.

Both GRU and LSTM networks attempt to resolve the vanishing gradient problem common in traditional RNNs. These variants address this problem by incorporating gating mechanisms, thereby regulating the temporal flow of information. However, training these networks poses challenges due to their sequential data processing requirement. Unlike GRUs and LSTMs, Transformer models handle time implicitly and thus enables the option to parallelize computation, making them notably more efficient.

5 Experiments

5.1 Investigating Hyperparameters

The transformer network contains several tuneable hyperparameters. In this project we have chosen to investigate the influence of the head size, number of heads, batch size and learning rate. To do so, two different sets of test were performed. The first pertained to investigating the architecture-related hyperparameters (head size and number of heads), whereas the second focused on the training parameters (batch size and learning rate).

5.1.1 Architecture Parameters

A grid search was conducted for different values of the number of heads and head size, while the remaining hyperparameters were kept constant. The constant hyperparameter settings are displayed in Table 4 in the appendix. One model was trained for each of the parameter settings. The training was conducted for three epochs and the models were then evaluated against the validation set. The result of the grid search is displayed in Table 1.

Judging by the results, it seems that increasing both the number of heads and the head size has a positive effect on model performance. One of the larger networks, with 16 heads and a head size of 32, outperformed the rest. However, there does not seem to be an entirely positive correlation between the size of the model and its performance, as the largest network performed significantly worse than some of the smaller networks. A common culprit for large networks is overfitting, but in this case, the final training loss was approximately the same as the validation loss (1.68) and

Number of parameters	Number of heads	Head size	Validation Loss	Validation Accuracy
6,960	1	16	2.25	35.62%
64,368	4	16	1.80	47.13%
226,928	8	16	1.63	51.69%
846,960	16	16	1.53	54.32%
19,952	1	32	2.00	42.14%
226,928	4	32	1.65	51.14%
846,960	8	32	1.55	53.68%
3,266,672	16	32	1.47	55.76%
64,368	1	64	1.83	46.57%
846,960	4	64	1.56	53.57%
3,266,672	8	64	1.49	55.22%
12,824,688	16	64	1.59	51.77%

Table 1: The results from the hyperparameter search for an optimal head size and number of heads. The best parameter settings are highlighted in green.

therefore indicated otherwise. A more likely explanation is therefore that three epochs was not sufficient time for the large number of parameters to adjust properly to the data.

5.1.2 Training Parameters

A parameter search for the batch size and learning rate was also conducted. The constant hyperparameter settings are presented in Table 5 in the Appendix. For each of the parameter settings, a model was trained for three epochs. After training, the model was evaluated against the validation set. The best performing model had batch size 8 and learning rate 0.001. The settings and validation accuracy for all models are presented in Table 6.

5.2 Investigating Data Representation

Another aspect investigated in this project was data representation. This was explored in two regards. Firstly, we investigated the effect of data augmentation. This was done by comparing two models trained on character level tokens, where one model was trained with data augmentation and one without. Secondly, we studied the effect of different tokenization schemes by comparing the performance of a model trained on character-level text representations to a model trained on BPE representations. In this case, both models were trained with data augmentation.

5.2.1 Performance

Table 2 shows a direct performance and size comparison with other character level models. The *RNN* model refers to the Vanilla RNN implementation from assignment 4 of this course. As can be seen in the table, our model performs better compared to these benchmarks. The results also suggest that data augmentation was not beneficial to model performance.

(These results are preliminary as the model with data augmentation has not finished training yet)

$Model$	Random	Bigram	RNN	Our Model$_{char}$	Our Model$_{charAug}$
n_{params}	0	12.5K	32.6K	8.2M	8.2M
$Loss$	4.71	2.51	1.85	1.18	1.21
$Accuracy$	0.9%	27.5%	44.8%	64.2%	64.1%

Table 2: Comparison of Different Models

Table 3 shows a direct size comparison between our model and Open-AIs smallest model available through their API, *Ada* which closely matches in performance to GPT-3 Medium from the original GPT-3 paper[1].

	n_{params}	n_{layers}	n_{heads}	d_{heads}	d_{embedd}	d_{vocab}
Our Model$_{char}$	8.2M	5	8	46	368	112
Our Model$_{BPE}$	9.6M	5	8	46	368	2 000
GPT-3 Medium	350M	24	16	64	1024	100 000

Table 3: Comparison between our best models and GPT-3 Medium

Due to the fact the GPT-3 models and our model have been trained with different sizes of corpus, it is difficult to perform a quantitative comparison between them. Because of this we have chosen to assess and compare them qualitatively. Below is a comparison between our models and GPT-3 Ada, all with the temperature 0.7 and the prompt: "This is a". The first output of each model is presented.

GPT-3 Ada: "This is a summary of the Security Advisory issued for this update. Click the names to read the full title for each title's details."

Our Model$_{BPE}$: "This is a species of fertilizations. Antimony is a size and are generally distinctive in appearance."

Our Model$_{char}$: "This is a certain figure with his hand which he is a man of being his face, and which is to be able to hear the hands, and he seems reported to the forth way."

As expected, GPT-3 has clearly generated the most coherent output of the three. It also appears that the BPE model is able to generate more reasonable text than the character-level model. There could be several explanations for this. One is that the same context length was used to train the models, meaning that the BPE-model was able to see a larger part of the text during training and thereby might have learnt to recognize word-level patterns better. Another reason for the discrepancy could be that the character-level model has to focus more on representing words correctly and therefore might not have had the capacity to learn other patterns that the BPE model was able to.

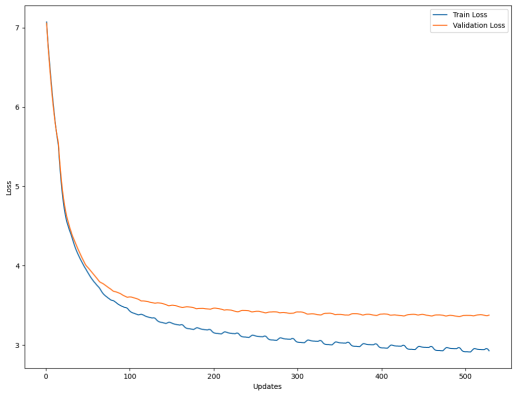

Figure 2: Train and validation loss for our BPE model during training

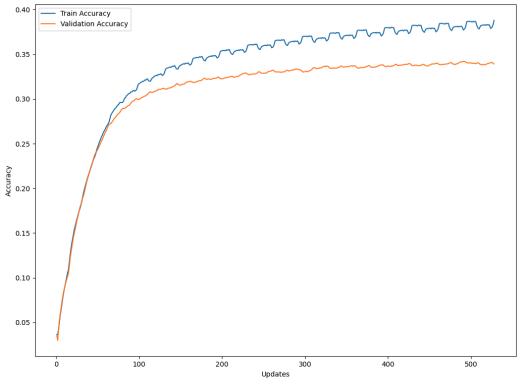

Figure 3: Train and validation accuracy for our BPE model during training

6 Conclusions

Through working on this project, we have deepened our understanding of decoder-based Transformer networks. Our experiments with different tokenization schemes, BPE and character level

tokenization, revealed a qualitative difference in model generation. With larger tokens the network has more room to focus on higher level language features as opposed to that combination of letters constitute valid words. Some additions to our project could be to scale up our model even further to see it we reach any significant bottlenecks and to experiment with different vocabulary sizes when using Bype-Pair encoding.

References

[1] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[2] Zihang Dai et al. “Transformer-xl: Attentive language models beyond a fixed-length context”. In: *arXiv preprint arXiv:1901.02860* (2019).

[3] Nitish Shirish Keskar et al. “Ctrl: A conditional transformer language model for controllable generation”. In: *arXiv preprint arXiv:1909.05858* (2019).

[4] Bohan Li, Yutai Hou, and Wanxiang Che. “Data augmentation approaches in natural language processing: A survey”. In: *AI Open* 3 (2022), pp. 71–90.

[5] Tianyang Lin et al. “A survey of transformers”. In: *AI Open* (2022).

[6] Stephen Merity et al. “Pointer sentinel mixture models”. In: *arXiv preprint arXiv:1609.07843* (2016).

[7] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[8] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[9] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9.

[10] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[11] Ruibin Xiong et al. “On Layer Normalization in the Transformer Architecture”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 10524–10533. URL: https://proceedings.mlr.press/v119/xiong20b.html.

Appendix

Parameter Settings	
Batch size	32
Learning Rate	0.001
Context Length	128
Decoder Layers	1

Table 4: The parameter settings used when performing the parameter search for head size and number of heads.

Parameter Settings	
Number of heads	8
Head size	16
Context Length	128
Decoder Layers	1

Table 5: The parameter settings used when performing the parameter search for batch size and learning rate.

Batch size	Learning rate	Validation loss	Validation Accuracy
8	0.1	3.25	18.47%
32	0.1	2.89	22.44%
64	0.1	2.86	24.06%
256	0.1	2.88	18.38%
8	0.01	2.50	28.58%
32	0.01	1.78	48.59%
64	0.01	1.72	50.03%
256	0.01	1.73	49.78%
8	0.001	1.63	52.49%
32	0.001	1.68	51.03%
64	0.001	1.72	50.18%
256	0.001	1.96	43.79%
8	0.0001	1.85	46.60%
32	0.0001	2.17	37.32%
64	0.0001	2.31	33.75%
256	0.0001	2.49	29.10%

Table 6: The results from the hyperparameter search for an optimal batch size and learning rate.

Title	Author	Published
Darwin	Gamaliel Bradford	1926
Slipstream	Eugene E. Wilson	1965
Crime and Punishment	Fyodor Dostoyevsky	1866
Nineteen Eighty-Four	George Orwell	1949
The Trial	Franz Kafka	1925
The Great Gatsby	F. Scott Fitzgerald	1925
Tarzan of The Apes	Edgar Rice Burroughs	1912
Winnie-the-Pooh	A. A. Milne	1926
The Story of Young Abraham Lincoln	Wayne Whipple	1915
Destiny times three	Fritz Leiber	1945
Wizard of Oz	L. Frank Baum	1900
A room with a view	Edward Morgan Forster	1908
Metamophosis	Franz Kafka	1915
Dubliners	James Joyce	1914
Siddharta	Herman Hesse	1922
White Fang	Jack London	1906
Poirot investigates	Agatha Christie	1924
The age of Innocence	Edith Wharton	1920
Peter Pan	J. M. Barrie	1904
The Enchanted April	Elizabeth von Arnim	1922
Alice's Adventures in Wonderland	Lewis Caroll	1865
A Tale of Two Cities	Charles Dickens	1859
War and Peace	Leo Tolstoy	1869
The King in Yellow	Robert W. Chambers	1895
The Adventures of Sherlock Holmes	Arthur Conan Doyle	1892
Little Women	Louisa May Alcott	1869
Animals of the Past	Frederic A. Lucas	1901
The Magic City	Edith Nesbit	1910
Dead Men Tell No Tales	E. W. Hornung	1899
The Beautiful Necessity	Claude Fayette Bragdon	1910

Table 7: List of the books in the book dataset.