UCLA
UCLA Electronic Theses and Dissertations

Title

Small Data Big Insights : Investigating the performance of a Small Language Model in its ability to generate English Text

Permalink

https://escholarship.org/uc/item/7655v2fk

Author

Sharma, Tanya

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
Los Angeles

Small Data Big Insights : Investigating the performance of a Small Language Model in its ability to generate English text

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Statistics

by

Tanya Sharma

2024

© Copyright by

Tanya Sharma

2024

ABSTRACT OF THE THESIS

Small Data Big Insights : Investigating the performance of a Small Language Model in its ability to generate English text

by

Tanya Sharma
Master of Science in Statistics
University of California, Los Angeles, 2024
Professor Ying Nian Wu, Chair

It is widely held that larger language models, trained on vast quantities of text, excel at generating coherent and fluent text. But at the same time, Small Language Models still struggle to produce meaningful text beyond a few words. The specific scale at which these abilities emerge is still not well-defined. Consequently, the lingering question remains: must a model be large-scale to generate coherent text?

In this paper we have have trained a small language model on Tiny Stories, a synthetic dataset of short stories. The objective is to study the small language models in their ability to generate coherent and consistent English text. We have performed a comparative study where we have analyzed the convergence of loss and investigated how adjustments to the number of heads, layers, and embedding size affect the generation of English text in Small Language Models.

The thesis of Tanya Sharma is approved.

Dave Zes

Nicolas Christou

Xiaowu Dai

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2024

TABLE OF CONTENTS

1 Introduction **1**

2 Language Models Overview **3**

2.1 Statistical Language Models 4

2.2 Neural Language Models 5

2.3 Pre-Trained Language Models 7

2.4 Large Language Models 8

3 All about Transformer Architecture **10**

3.1 Model Architecture 11

3.1.1 Encoder 12

3.1.2 Decoder 12

3.2 Self Attention Mechanism 13

3.2.1 Step 1: Creation of key, query, value vectors 13

3.2.2 Step 2: Score Calculation 15

3.2.3 Step 2: Weight Calculation 16

3.2.4 Step 3: Output of the Self Attention Layer 18

3.3 Multi-Head Attention 19

4 Experiment Design **21**

4.1 Data Description 21

4.2 Experiment Set Up 22

5 Analysis **26**

5.1 Convergence of Validation Loss 26

5.2 Impact of increasing the number of heads or layers on English text generation 28

5.3 Impact of increasing the embedding size 30

6 Conclusion **33**

7 Future Scope **35**

References **37**

LIST OF FIGURES

3.1 Transformer model architecture. Adapted from [16] 11

3.2 Word Embeddings 13

3.3 Creation of Key, Query and Value 14

3.4 Attention Scores 15

3.5 Attention Weights for Encoder 16

3.6 Attention Weights for Decoder 17

3.7 Output of the Self Attention Layer in an Encoder block 19

3.8 Multi-Head Attention 20

5.1 Convergence of Validation Loss 27

LIST OF TABLES

4.1 Combinations of hyperparameters 23

5.1 Impact of increasing the number of heads and layers : Story Completions (20 tokens) 28

5.2 Impact of increasing embedding size : Story Completions (20 tokens) 31

CHAPTER 1

Introduction

Language is a mode of communication for humans. It is an ability that allow humans to express themselves. Within the realm of language, each linguistic system is governed by a unique set of grammatical rules, symbols and vocabulary. While the rules define the structure of the sentence, it is the amalgamation of vocabulary knowledge and contextual comprehension infused with an understanding of background, situation, and environment, that empowers humans to interpret meaning and reason cogently.

However, machines lack the innate cognitive capacities possessed by humans, thus presenting a significant challenge in enabling them to comprehend, interpret, and generate language proficiently and how to bridge this gap has been a longstanding research question for researchers. This is where Language Modeling comes into the picture. Technically Language Modeling aims to model the generative likelihood of word sequences, so as to predict the probabilities of future or missing tokens[2].

In the pursuit to better understand and generate language, Language Modeling has undergone a remarkable evolution, transitioning from traditional Statistical Language Models to contemporary Neural Language Models. Notably, recent advancements have introduced the concept of pre-trained Language Models, which involves pre-training Transformer models on vast corpora. These models are extremely large in size with millions or even billions of parameters. A model attains the classification of a 'Large Language Model' when it encompasses at least one billion parameters. With the strides made in this domain, Language models have achieved proficiency in executing a diverse array of functionalities, including

but not limited to generation, summarization, translation, and reasoning. The scope of this paper will be centered specifically on the study of coherent English text generation.

Large Language Models have shown a significant improvement in their capabilities of generating coherent English text and Small Language Models like GPT Neo and GPT-2 often falter in producing meaningful text even when they are trained on large corpus of text. This raises the question of whether the ability to generate meaningful and logical text occurs only at large scale? If not, then the next obvious question is why Small Language Models like GPT-2 or GPT Neo struggle to generate coherent English text beyond a few words?

The paper "TinyStories: How Small Can Language Models Be and Still Speak Coherent English?"[1] delved into this inquiry and unveiled compelling findings. It demonstrated that even small language models, with as few as 10 million parameters and simpler architectures, possess the remarkable ability to generate fluent and cohesive text. Beyond mere coherence, these models showcase proficiency in crafting diverse sentences with nearly perfect grammar and also exhibit reasoning capabilities. They propose that the limitation hindering Small Language Models from generating coherent English text may not stem from their inability to deconstruct and comprehend the intricacies of language. Instead, it could be attributed to they being overwhelmed by the sheer volume and diversity of information they must process and store which hinders their ability to learn the core mechanisms and principles of language.

Inspired by the aforementioned paper, my thesis aims to construct a Small Language Model from scratch. Trained on the Tiny Stories dataset, my objective is to expand upon the notion that even small language models have the ability to generate English text that is not only coherent and consistent but also meaningful. Moreover, in this study, we have taken an additional step by conducting a comparative analysis. We varied combinations of parameters, such as the number of heads, layers, and embedding size, and analyzed the convergence of loss. Additionally, we investigated how adjustments to these parameters affect the generation of English text.

CHAPTER 2

Language Models Overview

Language models have achieved proficiency in executing a diverse array of functionalities like generation, summarization, translation, and reasoning. However, the effectiveness of these functions relies heavily on the model's understanding of language syntax, grammar rules, vocabulary, and contextual nuances. For instance, consider the task of completing the sentence, "It was raining outside, so I took..." In order to logically complete this sentence, the language model must grasp the context surrounding it. An illogical completion, such as "took a pencil," would indicate a failure to generate relevant output in line with the given context. Conversely, a suitable completion, such as "took an umbrella," reflects the model's ability to generate output that aligns with the contextual cues provided. Likewise, when prompted, a language model should be capable of completing the following sentence: "The temperature dropped below freezing overnight, so I packed..." The model's ability to reason correctly would involve understanding that a drop in temperature below freezing implies chilly weather conditions, prompting the need for warm clothing such as a jacket and gloves. These instances illustrate the intricate nature of language tasks. Despite their complexity, we have observed a substantial advancement in the abilities and capabilities of language models over time. This evolution can be broadly categorized into four major stages of development[2].

2.1 Statistical Language Models

Statistical language models represent an approach to understand and predict language based on statistical principles. At the core of statistical language models lies the Markov assumption which is a fundamental concept in probability theory. This assumption proposes that the prediction of the next word depends only on a fixed, limited context, usually the preceding words in the sequence. In other words, the likelihood of a word occurring next is determined by the words that came before it.

The famous n-gram language model falls under this category. In n-gram, "n" represents the chunk of n consecutive words. So, in the context of an n-gram model, the prediction of the nth word depends on the preceding n-1 words. For instance, in a bi-gram language model, the prediction of the next word depends on the preceding word and similarly in the tri-gram language model, the prediction of the next word depends on the last two words . More generally speaking,

$$P(word_{(i)}|word_{(i-1)}, ..., word_{(i-n+1)}) = \frac{P(word_{(i)}, word_{(i-1)}, ..., word_{(i-n+1)})}{P(word_{(i-1)}, ..., word_{(i-n+1)})}$$

While the n-gram model is widely used and effective approach in natural language processing, it comes with its own limitations. One of the major drawback of the n-gram model is that its predictions are constrained by a fixed length context window. Since the model only considers a fixed number of preceding words to predict the next word, it may fail to capture long-range dependencies and contextual differences present in the text. For instance, in the sentence "Stella was not hungry so she decided to not have dinner", a bi or a tri-gram language model may fail to associate "she" with "Stella". This limitation becomes particularly more serious when dealing with complex language structures or ambiguous word sequences because the meaning of the word can change based on distant words. For instance, based on the context and usage the word 'bank' can refer to either a financial institution or the sloping land alongside a river. Additionally, the n-gram model suffers from the problem of sparsity, especially with higher-order n-grams. The occurrence of certain word combinations

may be rare or nonexistent in the training data which can result in inaccurate predictions. Moreover, the model struggles with out-of-vocabulary words. It cannot generate predictions for unseen or infrequently occurring words not present in the training data. Overall, while the n-gram model offers a simple and efficient approach to language modeling, its drawbacks highlight the need for more sophisticated techniques to address the challenges.

2.2 Neural Language Models

The Neural Language Models utilize the neural network architecture to get the probability distribution over the vocabulary to predict the next word in the sequence. Consider a scenario where the first n words in a text document are denoted as $x_1, x_2, ...x_n$ and the objective is to predict the $(n+1)^{th}$ word. The context window is of length 5. Then in this case, input to the model is the sequence of words $x_{n-4}, x_{n-3}, x_{n-2}, x_{n-1}, x_n$ which serves as the context for the model to make predictions. So, instead of looking at all the words from the beginning, we only look at the words that lie in the context window. The length of the input sequence can vary depending on the model's architecture and requirements that we define. The output of the model is the probability distribution $P(x_{n+1}|x_n, x_{n-1}, ..., x_1)$. The fundamental difference between the Statistical Language Models and the Neural Language Models emerges in the methodology of prediction of the next word. While the traditional statistical language models rely on the concept of conditional probability. The neural language models, rather than relying on statistical principles, these models pivot towards a neural network architecture. Here, the word embeddings serve as input to a neural network which finally outputs a probability distribution over the vocabulary.

It is important to note that, the neural language models have not only helped us understand and better implement the word sequence modeling task but they also introduced distributed word representation. Distributed representation are nothing but vector representation of words or in simple terms turning words into vectors. The idea is that words

used in similar contexts usually have similar meanings. These vectors, also known as word embeddings, help us represent words in a way that captures these relationships.

Language Models, relying on the architectural foundations of the Feed Forward Neural Network, Recurrent Neural Network (RNN), or Long Short-Term Memory Network (LSTM), collectively fall under the domain of Neural Language Models. Each architecture has its own strengths and limitations.

The Fixed window Neural Language Model based on the feed-forward neural network architecture marks as a good entry point into neural language modeling because of its less complex and simple architecture. However, the fixed window constraints the model's ability to capture sequences of varying lengths. Additionally, the model may also fail to capture long-range dependencies and contextual differences present in the text due to the fixed context window. Conversely, Language Models based on Recurrent Neural Network architecture go beyond the fixed window limitations and are capable of processing sequences of arbitrary length. Moreover, the inherent ability of RNNs to leverage information from multiple steps in a sequence facilitates the modeling of long-term dependencies, enhancing the model's predictive capabilities. However, RNNs have their own set of challenges as they suffer from the problem of vanishing gradients. The Long Short-Term Memory Network (LSTM) architecture emerges as a refinement of the RNN and it was proposed as a solution to vanishing gradients problem. However, LSTM presents a promising solution to this challenge, there is still no evidence that it can successfully mitigate it. Central to the LSTM architecture is the introduction of gating mechanisms. The gates selectively retain or discard information at each time step. This solves the context management problem in textual data by removing information no longer needed and adding information likely to be need later in the context using gates.

In summary, Neural Language Models have completely transformed Natural Language Processing. They've made word sequence modeling much better and also introduced distributed representations, which have been a game-changer.

2.3 Pre-Trained Language Models

Pre-trained language models serve as a subset within the broader domain of Neural Language Models. They undergo training on extensive text corpora, such as entire Wikipedia articles or vast collections of online text. Pre-training a model exposes it to vast amounts of textual material, which helps it to understand even the smallest linguistic distinctions. The model picks up a lot of knowledge about the linguistic components of the language during this phase, such as syntactic structures, grammar rules, and semantic associations. This gives the model a thorough grasp of the nuances of the language, enabling it to identify minute patterns and connections in the data.

After completing the pre-training phase, the model can be optimised for a particular purpose/task. This is called fine-tuning a model. During fine-tuning, the pre-trained model is exposed to task-specific data, which is often smaller in size, to adjust and specialise its learnt representations to the specifics of the given task. The difference between using an experienced team member's skills and on boarding a new person for a project can be compared to the process of training a model from scratch versus fine-tuning a pre-trained model. Starting from beginning when training a model is like taking a new employee and teaching them every task from the bottom up. On the other hand, fine-tuning a pre-trained model is like using the expertise of a seasoned worker who already has a thorough understanding of processes involved.

ElMo[14] was one of the earliest attempts in the realm of pre-trained models. It introduced the concept of context aware word representations and suggested that instead of using static word embeddings to represent a word, we can construct context aware word embeddings for which it pre-trained a bi-directional LSTM network. A lot of research is being conducted in this area and several pre-trained models have been released each with its own architecture and pre training strategies. Some of the pre-trained models include BERT (Bidirectional Encoder Representations from Transformers)[12], GPT-2 (Generative Pre-

trained Transformer 2)[11], BART (Bidirectional and Auto-Regressive Transformers)[10], among others.

2.4 Large Language Models

When we scale the pre-trained models, increasing both in terms of model size and amount of data that they're trained on to the point where they have over a billion parameters, we enter into the domain of Large Language Models. Some of the common Large Language Models include GPT-3 (Generative Pre-trained Transformer 3)[9] with 175 billion parameters, PaLM (Pathways Language Model)[6] with 540 billion parameters, LLaMA-2 (Large Language Model Meta AI))[3] with 70 billion parameters, among others.

Despite sharing similar architectural blueprints and pre-training methodologies with their smaller counterparts, Large Language Models (LLMs) exhibit marked differences in terms of their capabilities. With the strides made in this domain, Large Language Models have achieved proficiency in executing a diverse array of functionalities, including but not limited to generation, summarization, translation, and reasoning. Additionally, the launch of ChatGPT, for the first time, offered users firsthand experience of AI's potential. In-fact, since then, research in this field has expanded significantly, fueling further advancements and innovation. There is growing interest in integrating Large Language Models (LLMs) into various real-world applications. From enhancing traditional search engines to creating personalized chatbots, the spotlight is on harnessing the potential of LLMs to build practical and impactful solutions.

Despite the increasing interest in this field, a plethora of questions regarding Large Language Models still remain unanswered. Why superior capabilities emerge at large scale is still a mystery. Similarly, when and how they emerge is an open research question. Additionally, the large amount of training data from the internet used in the training of LLMs, poses challenges in mitigating biases and aligning model outputs with human values. Hallucinations

is another big issue with LLMs.

In this study, our focus is to dive deeper into the question of scale unlocking superior capabilities in LLMs. One might argue that enlarging model size and amount of training data allows LLMs to become adept at handling more complex language tasks because given that they are exposed to larger amount of data they are able to learn broader range of context and associations which then results in enhanced performance. However, whether scale is inducing these abilities in LLMs is still an open research question. Infact, in the later sections, we will try to break this belief and particularly in the context of English test generation demonstrate that even small language models possess the capability to generate English text that is both coherent and consistent, and at the same time meaningful.

Research is on-going in these and several other facets pertaining to LLMs. However, the research community itself is faced by certain limitations. For instance, given the computational power required to train them, it becomes difficult for the research community to train capable LLMs. Major tech giants have the resources to train large models but they end up not releasing the training details to the public. Research institutions in collaboration with the industry leaders are moving towards building AI infrastructure. There is still a long way ahead but we are moving forward.

Most of the LLMs currently are being developed on the Transformer architecture. In the next section we will go into the details of this architecture.

CHAPTER 3

All about Transformer Architecture

Up until now, we've delved into the journey of Language Modeling, exploring various advancements and weighing the strengths and weaknesses of different architectures at a broader level. Now, let's shift our attention to Transformer, another groundbreaking architecture within Language Modeling. This architecture has sparked a paradigm shift in the way we approach Language Modeling tasks, leading to a new era of innovation.

The Transformer architecture was introduced for the first time in the paper 'Attention is All You Need' in 2017 [16]. This paper was written by the researchers at Google Brain. The highlight of the paper was the 'Attention Mechanism'. The paper proposed a new and simple neural network architecture which was solely based on the Attention Mechanism and no longer required recurrent and convolutional layers which were widely used in sequence-to-sequence modeling up until then. Furthermore, what set the Transformer apart from other architectures was its ability to process input data in parallel, a departure from the sequential processing employed by recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks. It also provided a much more efficient way of capturing long-term dependencies within data.

In this section, we're going to take a deep dive into the architecture of the model. After we've got a good grasp of the overall architecture, we'll zoom into the details of the attention mechanism. Finally, we'll take a closer look at how this Transformer architecture can be used in language modeling.

3.1 Model Architecture

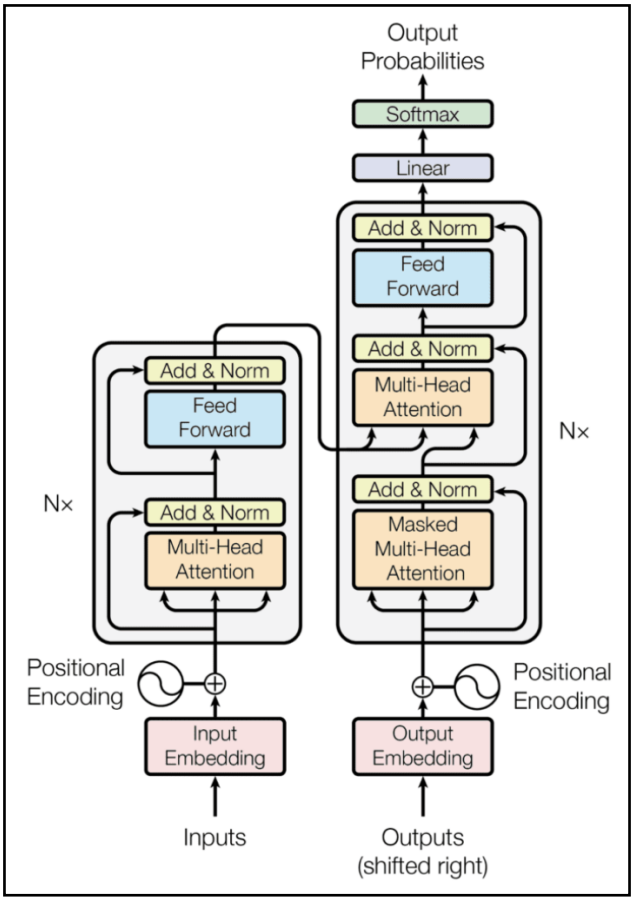

Figure 3.1: Transformer model architecture. Adapted from [16]

The transformer architecture, as outlined in the 'Attention is all you need' paper, introduces a revolutionary framework featuring an encoder-decoder structure. At its core, the encoder component processes an input sequence $(x_1, x_2, x_3...x_n)$ and transforms it into a sequence of continuous representations $(z_1, z_2, z_3...z_n)$. Subsequently, the decoder utilizes these representations $(z_1, z_2, z_3...z_n)$ to generate an output sequence $(y_1, y_2, y_3...y_m)$ sequentially.

To achieve this sophisticated task, the Transformer architecture implements a series of layers within both the encoder and decoder. Specifically, each component comprises a stack of self-attention layers alongside fully connected neural networks. This stacking mechanism

allows for multiple iterations of self-attention and neural network operations, facilitating intricate information processing and context modeling within the model.

3.1.1 Encoder

When examining figure 3.1, we observe that the encoder block is structured with a repetition of N_x identical layers, stacked one over the other. Each of these layers is composed of two distinct parts. Firstly, there's the multi-head self-attention layer, where the input flows through the self attention mechanism which allows every word to better comprehend the context. Following this, the resulting output transitions into a fully connected feed-forward neural network. However, the flow id not direct. Before reaching the neural network, the output takes a detour through what's called a residual connection. This means it merges with the original input. Additionally, there's a layer normalization step applied to the output, which ensures that the data remains standardized and stable throughout the process.

3.1.2 Decoder

Even in the decoder block, we find a structured composition of N_x identical layers, systematically arranged atop each other. However, what distinguishes the decoder from the encoder is a slight difference in the architecture. While the encoder employs two sublayers, the decoder expands this concept by incorporating three distinct sublayers.

Similar to the encoder's design, the decoder features a multi-head attention layer and a fully connected neural network layer. Additionally, it introduces a crucial step: applying another multi-head attention layer to the output of the encoder block. This layer facilitates the decoder in comprehending and aligning with the context established by the encoder.

Much like in the encoder, the decoder employs residual connections and layer normalization to maintain stability and facilitate information flow. However, an important difference lies in the handling of self-attention weights, a topic we'll dive in the next section.

3.2 Self Attention Mechanism

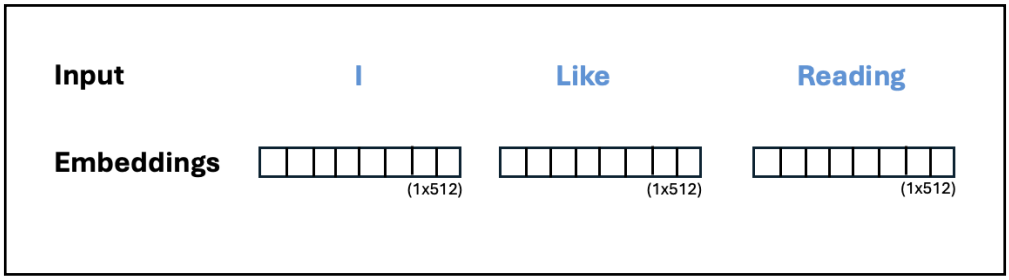

Figure 3.2: Word Embeddings

In our exploration of model architecture, we delved into the Transformer, recognizing it as a series of interconnected self-attention and fully connected layers. This begs the question: what sets it apart and makes it exceptionally efficient? The answer lies in the revolutionary self-attention layer. In the upcoming section, we'll embark on a comprehensive exploration of the inner workings of the self-attention layer to uncover its transformative impact. Lets take an example to understand the mechanics of the self attention layer : 'I like reading'. In this case the input to the model will be three words 'I','like', and 'reading'. Each word will have its own word embedding. Lets assume that the word embeddings are of dimension $(1 * 512)$ where $n_{embed} = 512$. So in this case each word will be represented by a word embedding , which is nothing but a vector of dimension $(1 * 512)$ as can be seen in the figure below and this will mark as the starting point.

3.2.1 Step 1: Creation of key, query, value vectors

The first step in the process involves the creation of three distinct vectors for each word, known as the key, query, and value vectors. This is achieved by multiplying the word embeddings by three weight matrices: the key weight matrix (W_{key}), the query weight matrix (W_{query}), and the value weight matrix (W_{value}), respectively. Each of these weight matrices has dimensions of (n_embed * head_size). For instance, when considering the word 'like', the

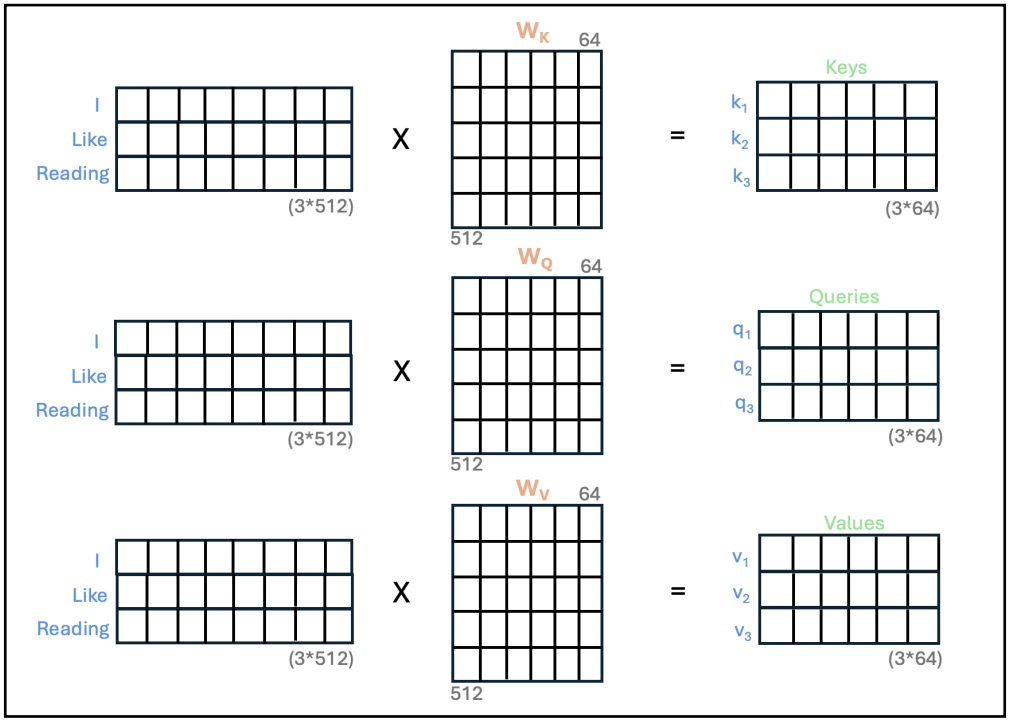

Figure 3.3: Creation of Key, Query and Value

resulting key vector is computed as follows:

$$K_{like} = E_{like} * W_{key}$$
$$(1 * 64) = (1 * 512) * (512 * 64) \rightarrow \text{Dimensions}$$

Here, head_size = 64

K_{like} = Key vector for the word 'Like'; dim: $(1 * 64)$

E_{like} = Word embedding vector for the word 'Like'; dim: $(1 * 512)$

W_{key} = key weight matrix; dim: $(512 * 64)$

It is worth noting that the choice of head_size is arbitrary and serves as a design decision in the architecture. Furthermore, the three weight matrices are trained during the training process.

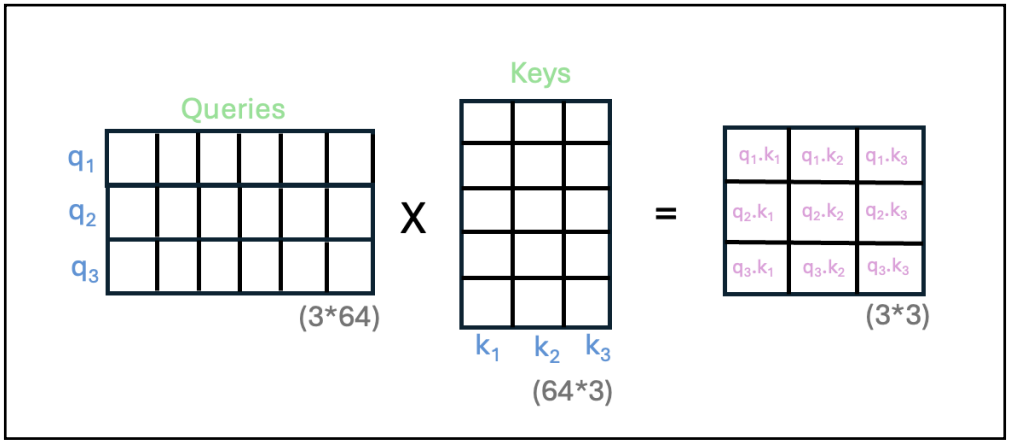

Figure 3.4: Attention Scores

3.2.2 Step 2: Score Calculation

In this step, we take the dot product of each word's key vector with query vector of all the other words in order to calculate the score. These scores are then used to calculate weights. Imagine each word has a unique "key" that represents its meaning. We compare the key of one word with the "query" of all the other words. It's like trying to find the best match for a puzzle piece – not every match will be perfect, but we give each match a score based on how well it fits. This score tells us how related each word is to all the others.It is this score in self attention mechanism that allows us to calculate weights and thereby weigh different input tokens in the sequence of words with respect to each word and hence help us quantify the association between words and capture long term dependencies. For instance, we take the dot product of (1*64) dimensional key vector of word 'like' with the (1*64) dimensional query vectors of all the words('I','Like','Reading'). We take the transpose of the Key matrix to comply with the rules of matrix multiplication.

$$Q.K^T = ScoreMatrix$$

$$(3 * 64).(64 * 3) = (3 * 3) \rightarrow \text{Dimensions}$$

3.2.3 Step 2: Weight Calculation

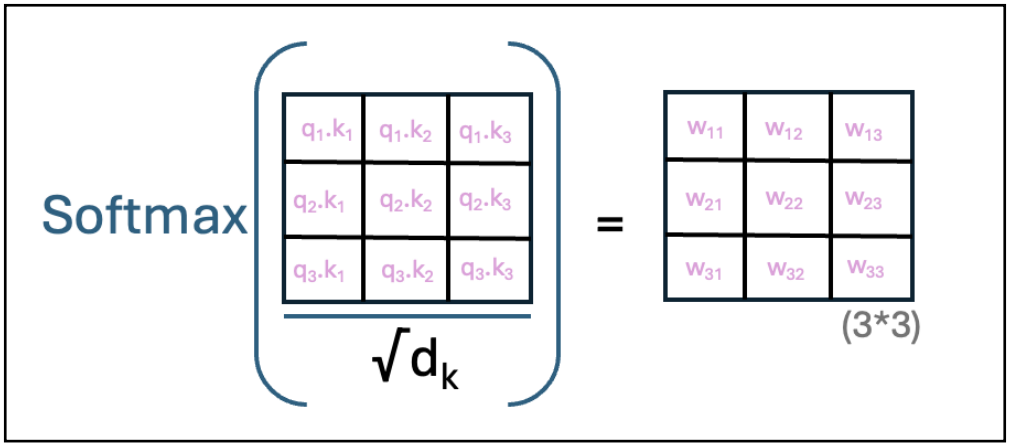

Figure 3.5: Attention Weights for Encoder

To calculate the weights, we start by dividing the computed scores by the $\sqrt{d_k}$ where d_k represents the dimension of the key vector. In our example $d_k = 64$. After obtaining the scores, we apply the softmax function to them. This function normalizes all scores, ensuring they are positive and sum up to 1. For instance, $w_{11} = softmax(\frac{q_1.k_1}{\sqrt{d_k}})$.

These weights play a crucial role in deciding how much attention we give to each word. At any given position, the word itself naturally receives the highest attention score. However, we also need to consider words in the surrounding context that are important for understanding the current word. These weights enable us to precisely determine how much attention we allocate to each word in the context. So, for every word, we obtain specific weights that guide our attention to the relevant words in its context. The weights w_{11}, w_{12}, w_{13} represent the softmax scores for the first word in our sequence, which is 'I'. These scores measure how much attention 'I' pays to the other words in the sequence. Specifically, w_{11} measures how much 'I' attends to itself, and it's naturally the highest. Similarly, w_{12} how much 'I' attends to the next word, 'Like', and so on for subsequent words in the sequence.

It's crucial to understand the distinct approach used for calculating weights in the encoder and decoder blocks. This difference is formulated mathematically while feeding the calculated

scores into the softmax function (we will look at it later). This disparity stems from the differing functionalities of these two blocks. Lets try to comprehend the difference.

Encoder Block: In this block, each word can attend to both past and future words. This means that when considering a particular word, it can gather information from all words that come before and after it in the sequence. For instance, in sentiment analysis, we can look at and attend to all the words in the sequence and get contextualised representations.

Decoder Block: Unlike the encoder, the decoder block has a unique constraint. It must ensure that it only attends to words from the past, not the future. This restriction is crucial for tasks like text generation, where we are predicting future words based on present information. Each word only attends to the words in the past or appearing before and never see the words in the future.

This difference is mathematically achieved by setting the scores for all the words appearing after the given word to negative infinity and the softmax$(-\infty) = 0$, so that makes the attention weights for all the words appearing in the future to zero ensuring that we don't attend to the future and only look at the past.(Fig - 3.6)

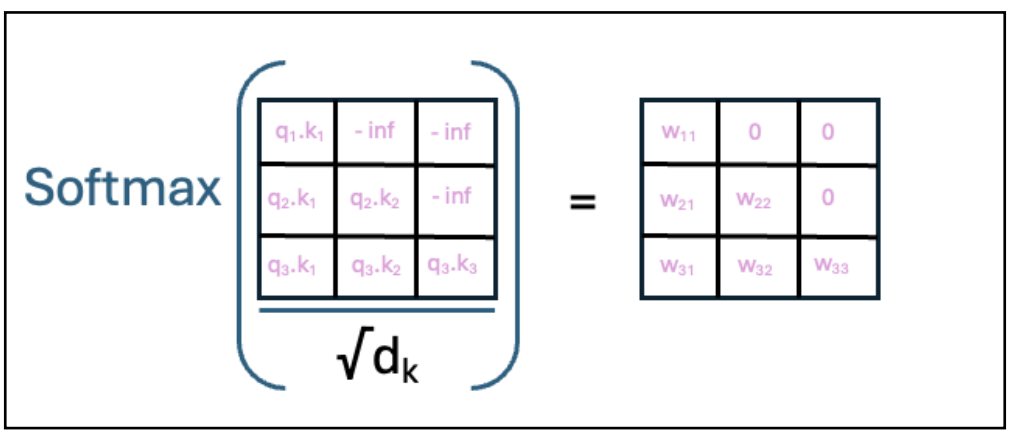

Figure 3.6: Attention Weights for Decoder

3.2.4 Step 3: Output of the Self Attention Layer

Once we have computed the attention weights, which essentially allow us to quantify the importance we give to words in the sequence. The next step in the process is to do a weighted sum of the value vectors where the weights are the attention weights.

Going back to our example, w_{11}, w_{12}, w_{13} are the attention weights for the first word i.e. w_{11} represents the attention first word gives to itself, w_{12} is the attention first word 'I' gives to the second word and so forth. In an encoder block the following operations will happen :

$$W * V = Z$$
$$(3 * 3).(3 * 64) = (3 * 64) \rightarrow \text{Dimensions}$$

here, W = (3*3) attention weights matrix,
V = (3*64) values matrix,
Z = (3*64) output of the self attention layer.

If we delve deeper into the details, let's consider a vector $z_1 = (z_{(1,1)}, z_{(1,2)}, ..., z_{(1,64)})$. This vector has a dimension of (1*64), meaning that it has 64 elements. Now lets focus on $z_{(1,1)}$ which is the first element of the vector and can be written as following weighted sum $z_{(1,1)} = w_{11} * v_{11} + w_{12} * v_{21} + w_{13} * v_{31}$. Similarly we can write the other elements of the vector z_1. So, if we really think about it, z_1, z_2 and z_3 are contextualised representations of the original words. The attention mechanism is like giving each word in a sentence the ability to focus on the words that matter most to it, while tuning out the rest. We achieve this by assigning attention weights, which determine how much importance each word places on its neighbors.

Think of it like human relationships. Each person decides how much attention they want to give to others. Some relationships are strong, so we pay more attention to those individuals, while others are more distant, so we give them less attention. Similarly, the attention mechanism empowers words to prioritize their connections within a sentence.

This concept might seem simple, but it's incredibly powerful. By allowing words to selectively focus on relevant information, the attention mechanism enhances our ability to understand the relationships between words and capture the context of a sentence and even language more effectively.

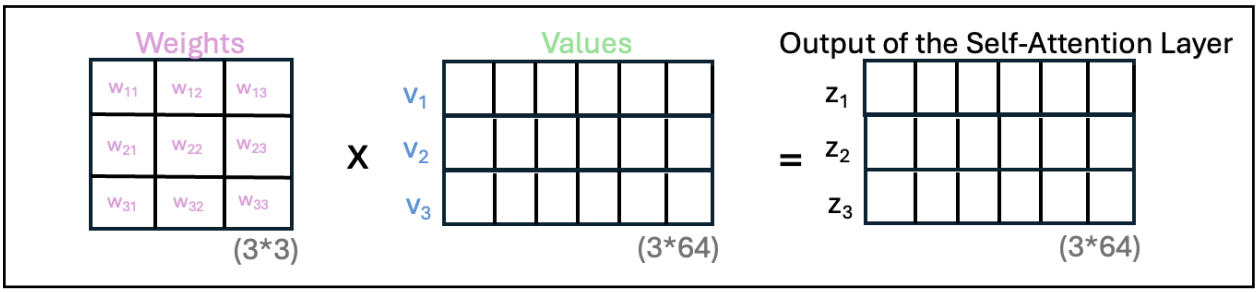

Figure 3.7: Output of the Self Attention Layer in an Encoder block

3.3 Multi-Head Attention

In our previous discussion, we delved into the intricacies of the self-attention layer's structure. Now, let's shift our focus to another critical component: multi-head attention.

To put it plainly, imagine if we could clone the self-attention architecture not just once, but N times. Each of these clones, or 'heads' that would independently attend to different parts of the input sequence. This ensemble of heads that work in parallel forms what we call multi-head attention mechanism. Essentially, it's like having N sets of eyes, each focusing on a different aspect of the input, allowing for a more comprehensive understanding of the context. However, each head will produce an output and we will have to figure out a way to consolidate the outputs of all the heads into one and feed it to the next layer.

Consider our example where we utilize 4 heads in the multi-head attention mechanism. Each of these heads independently processes the input and generates a matrix with dimensions of (3*64). Now, rather than treating these outputs separately, we consolidate them into a single matrix of dimensions (3*256) by stacking them vertically, one after the other.

Next, we take this consolidated matrix and perform a crucial step: we multiply it by another weight matrix, which has dimensions of (256*64). This multiplication operation essentially blends the information from all the heads together, transforming the (3*256) matrix into the final output of the multi-head attention layer with dimensions (3*64).

In simpler terms, it's like having different groups of eyes (the heads) looking at different aspects of the input, then combining their insights into a single coherent understanding through a weighted process.

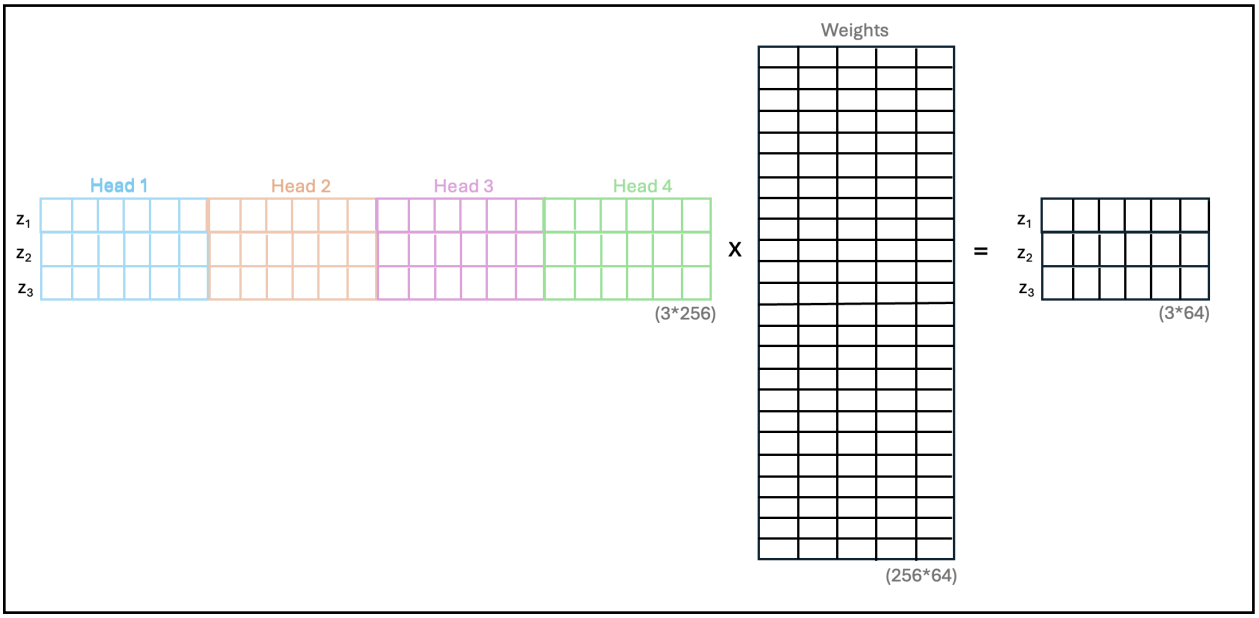

Figure 3.8: Multi-Head Attention

CHAPTER 4

Experiment Design

In this section, we will delve into the experiment design, providing comprehensive insights into its setup, methodology, and the data used for the analysis. To conduct our analysis, we trained a small language model similar to GPT, drawing inspiration from Andrej Kaparthy's GPT code. Once we finalized the model architecture, we embarked on a series of experiments, exploring various combinations of hyper-parameters such as the number of heads, layers, and embedding size. For each combination, we monitored both training and validation loss. Furthermore, we evaluated the model's ability to generate English text using stories from the test set.

4.1 Data Description

The data used for this comparative analysis is TinyStories dataset. The dataset was introduced in the paper 'TinyStories: How Small Can Language Models Be and Still Speak Coherent English?'. As implied by its name, this dataset comprises short stories. What makes it interesting is that these stories were generated by GPT-3.5. The authors of the paper while designing prompts ensured that it was for short story generation that featured basic vocabulary, akin to what a 3-4 year old could easily understand. While the stories are short with limited vocabulary, the corpus is rich in terms of having all the elements of language like vocabulary, grammar, reasoning etc.

Here is a sample story from the dataset -

One day, a little girl named Lily found a needle in her room. She knew it was difficult to play with it because it was sharp. Lily wanted to share the needle with her mom, so she could sew a button on her shirt. Lily went to her mom and said, "Mom, I found this needle. Can you share it with me and sew my shirt?" Her mom smiled and said, "Yes, Lily, we can share the needle and fix your shirt." Together, they shared the needle and sewed the button on Lily's shirt. It was not difficult for them because they were sharing and helping each other. After they finished, Lily thanked her mom for sharing the needle and fixing her shirt. They both felt happy because they had shared and worked together.

The dataset is loaded from the dataset repository maintained by Ronen Eldan. It contains 2,119,719 stories in the train set and 21,990 in the test set.

4.2 Experiment Set Up

For the analysis we loaded the data from the dataset repository maintained by Ronen Eldan. It contains 2,119,719 stories in the train set and 21,990 in the test set. Then, we take all the stories in the train set and make one single corpus of text. Following that, we use the GPT-2 tokenizer provided by the Hugging Face library to tokenize the text data. Specifically, we use the pre-trained tokenizer for the GPT-Neo model with a size of 1.3 billion parameters, which was developed by EleutherAI.

In our experiment, we've trained multiple small language models with different setups. We have experimented with various parameters like the number of heads, layers, and embedding size. Table 4.1 shows the specific setups for all 8 models we've worked with. Our aim is to understand how these different factors influence the generation of English text.

For instance, the number of heads in our models can be 1, 4, or 8. Similarly, we've experimented with different numbers of layers, choosing from 4, 6, or 8. Additionally, we've varied the embedding size, trying out three options: 64, 256, and 512.

When we look at the first 6 models in the table, we see that they all have similar number

Model	Number of Heads	Number of Layers	Embedding Size	Total Parameters
Model 1	1	4	64	6.69M
Model 2	1	8	64	6.89M
Model 3	4	4	64	6.69M
Model 4	4	8	64	6.89M
Model 5	8	4	64	6.69M
Model 6	8	8	64	6.89M
Model 7	1	4	256	29M
Model 8	8	6	512	70M

Table 4.1: Combinations of hyperparameters

of parameters, ranging from 6.6 million to 6.9 million. In these models, we've kept the embedding size constant at 64 while adjusting the number of heads and layers. We intentionally kept the embedding size at 64 for a specific reason. Increasing the embedding size would lead to a big increase in the number of parameters in the model. As we can see in the table below, increasing the embedding size from 64 to 256 (which is four times bigger) leads to an approximate five times increase in the number of parameters from around 7 million to 29 million. However, a similar four times increase in number of heads or layers does not increase the number of parameters by the same factor. So, by keeping embedding size the same, we could focus on understanding how changing the number of heads and layers affects models of similar size. Once we've understood that, we then take a step further and experimented with larger embedding sizes to see how they impact text generation.

For this analysis, we conducted our model training over a span of 20,000 epochs, monitoring both the training and validation loss at intervals of every 500 epochs. It's crucial to highlight that due to the relatively small scale of these models, achieving optimal results necessitates longer training duration. However, it's important to mention that, for the current study, our training is constrained to a maximum of 20,000 epochs.

In order to test how well our models perform, we use stories from the test set. for each story in the test set, we randomly pick a spot to split the story into two parts. The first part acts as the prompt and the second part is the original completion. Then we feed the prompt to our trained models and generate next 10 or 20 tokens. For instance, if we have the following story from the dataset -

Jane was a persistent girl. She loved to watch the Earth from her bedroom window. Every day, she thought about the Earth and all of its mysteries. She wanted to find out more about it.One day, Jane had a plan. She asked her mom if she could go outside and explore the Earth even more. Her mom said "yes" and helped her get dressed.Jane walked and walked until she came to a meadow. She found a pond and noticed some ducks swimming in it. She sat down and watched them for a while.Jane stared up at the sky and noticed some clouds in the sky. She was so amazed by the shapes that they made. She explored the Earth all day and when it was time to go home, she was sad. But she had a plan to come back soon. Jane was excited and full of energy as she walked back home. She knew she could explore the wonders of Earth with her persistent spirit.

We randomly select a point to split the story into two parts.

Prompt : *Jane was a persistent girl. She loved to watch the Earth from her bedroom window. Every day, she thought about the Earth and all of its mysteries. She wanted to find out more about it. One day, Jane had a plan. She asked her mom if she could go outside and explore the*

Original Completion : *Earth even more. Her mom said "yes" and helped her get dressed.Jane walked and walked until she came to a meadow. She found a pond and noticed some ducks swimming in it. She sat down and watched them for awhile.Jane stared up at the sky and*

noticed some clouds in the sky. She was so amazed by the shapes that they made. She explored the Earth all day and when it was time to go home, she was sad. But she had a plan to come back soon. Jane was excited and full of energy as she walked back home. She knew she could explore the wonders of Earth with her persistent spirit.

Model Completion (first 20 tokens) : *planet. Her mom said yes, so they set off on a journey. They walked and*

Lastly, we analyse the outputs of different models to see if they're forming the right words, using correct grammar, forming sentences correctly, and making sense based on the context. We're basically checking if they're generating text that fits well with what we expect. In the next chapter we will closely analyse the results.

CHAPTER 5

Analysis

This section is dedicated to the analysis of the output generated by the language models. As previously mentioned, we've already experimented with various configurations for our models. Now, in this section, we delve deeper into our findings. Using multiple examples we aim to compare the performance of these models. Our objective is to establish an understanding and rationale regarding how adjusting certain parameters, such as the number of heads, layers, or embedding size, affects the generation of English text.

5.1 Convergence of Validation Loss

During the training process of our models, we recorded both the training and validation loss at intervals of every 500 epochs. This approach allowed us to closely examine the convergence behavior of the validation loss over the course of training. In Figure 5.1, we present the validation loss exclusively for Models 6, 7, and 8. This selective focus was chosen because the validation loss trends for Models 1 through 5 closely mirrored that of Model 6. By narrowing our analysis to these specific models, we aim to provide a clearer and more concise illustration of the validation loss convergence patterns observed during our experiments.

For the same 20,000 epochs we observe a faster convergence of validation loss for Model 8 followed by Model 7 and then Model 6. Model 8 is the biggest model with 70 million parameters. It's Important to highlight that our analysis entails training all models exclusively

for 20,000 epochs. However, extending the training duration would likely result in further reductions in validation loss. Ideally, training iterations should continue until the validation loss either plateaus or shows signs of degradation. But in this case we stopped the training at 20,000 step.

Upon examining the plot, we notice something interesting. It seems like if we have a language model with fewer number of parameters, we might need to train it for a longer time. This means running more rounds of training, or epochs. On the other hand, if the model has more number of parameters, it can reach similar performance levels in fewer training rounds.

So, what does this mean? It suggests there's a sort of balancing act between how long we train a model and how complex it is. Models with fewer number of parameters might need extra time to learn, while those with more parameters can learn faster. It suggests a trade-off between training time and model complexity.

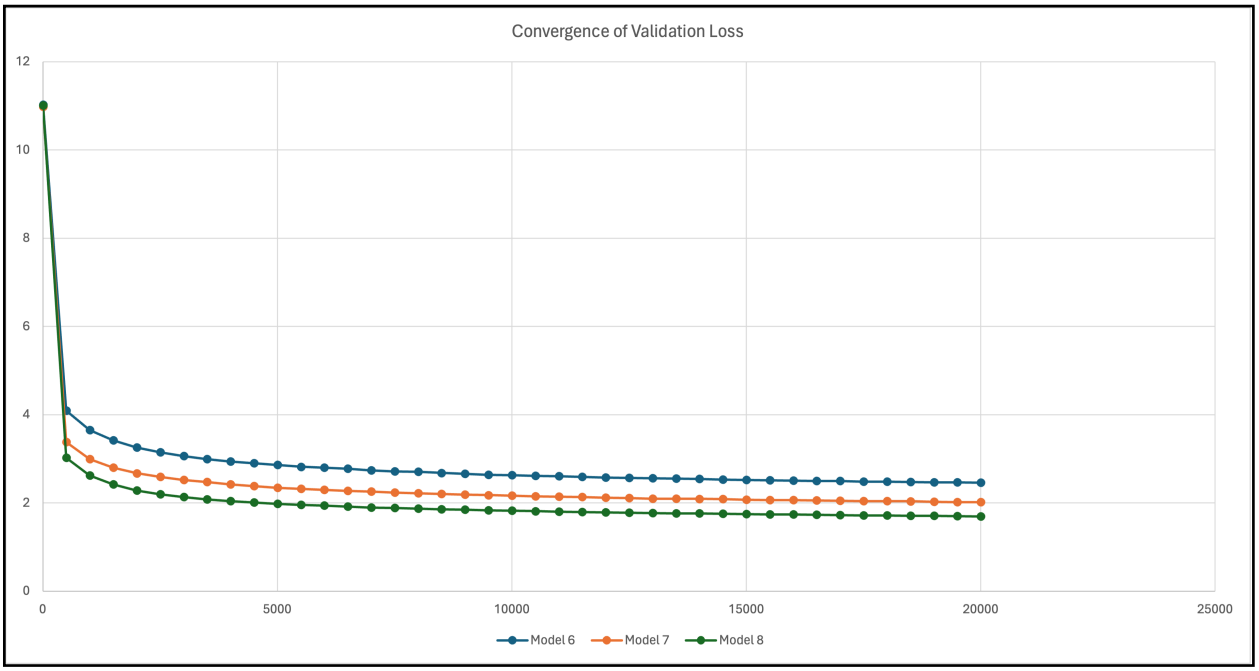

Figure 5.1: Convergence of Validation Loss

5.2 Impact of increasing the number of heads or layers on English text generation

Prompt	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Once there was a bunny and a bear. The bunny was envious of the bear's new red glove. The bunny complained that she wanted a	good haircut. He was happy and didn't want one idea, before acting it would stay up.	ant, so it didn't give up. The cater pillar and the bunny felt better	back and she loved it. Suddenly, the bunny stopped and went to his burrow and	robot to make it. The rabbit started to feel better. He knew everyone was very special	snack. But when she arrived, she was very scared and knew she did not be gone.	veterinarian to help but suddenly he stayed high up. The rabbit couldn't believe which anna was
Once upon a time there was a little girl. She was three years old and she loved to stretch. She stretched her arms and legs and fingers and toes. Every morning she would stretch her body in the sunshine.One day when she was stretching, she noticed something shiny under her feet. She bent down and picked it up. It was a little ashtray. She thought it was really pretty so she decided to keep it as a special	picture. When she got there, she did not find him. She started to find a magnet around	treat for her. She measured the magic purse, but she was persistent. From that day on her	day. What a jar rum across her best way home carrying a long fast in her pocket.	place. She measured a piece of pretty pieces and smelled the bright. There were different kinds of beautiful	texture. Everybody and read and her friends would have such a special squirrel who could go very creative without	The girl started to unstep, so she grew le no reach it. How
Once upon a time, there was a naughty toy. The toy wanted to have some fun all by itself. So it decided to go exploring.The toy went all around the house, looking for something interesting. Soon, it found a yummy cookie. But before it could eat it, a big hand caught the toy. It was Mama!Mama said, "Naughty toy, you need to stay in your room!" The toy was very sad and tried to explain it just wanted to have some fun. But Mama was not happy.Finally, Mama said, "Okay.	But if I can hear that the toy?" The toy just liked the toy and the toy	Let 's play," said, "Sure, that, I should go. Let's play with you	Then I will give it some milk just big respect the toy car. "Papa and Grand	Let's look back tomorrow, Mummy just play with the ball. You could not stand so much	Stay back." Soon finally arrived, Maggie played with the toys. She made lots of dolls	It can be fun to enjoy, animals will keep. "Mama smiled and said,

Table 5.1: Impact of increasing the number of heads and layers : Story Completions (20 tokens)

In order to compare the impact of increasing the number of heads on English text generation, we will compare models that share identical numbers of layers and embedding size but differ in the number of heads. By maintaining consistency in layers and embedding size while only altering the number of heads, we aim to isolate and scrutinize the precise

influence of this particular parameter adjustment. This is done to ensure that we compare apples with apples and not apples with oranges. Since Model 1, 3 and 5, each have 4 layers and an embedding size of 64, we will compare their outputs. Similarly, we will compare the outputs of Models 2, 4, 6 because all of them have 8 layers and an embedding size of 64.

Similar approach is adopted to study the impact of increase layers on text generation. We ensure that the models we use for comparison have similar number of heads and embedding size. Since Model 1 and 2, both have 1 head and an embedding size of 64, we will compare their outputs. Similarly, we will compare the outputs of Models 3 and 4 because both of them have 4 heads and an embedding size of 64 and Models 4 and 6 because both of them have 8 heads and an embedding size of 64.

In Table 5.1, we present three examples along with the prompts used for each of the six models and the corresponding output generated by each model. Each of our models demonstrates the ability to generate English words coherently and is not producing gibberish. Furthermore, the models demonstrate a rudimentary understanding of language rules, such as correctly punctuating sentences with full stops and capitalizing the first letter after a full stop. However, it's evident that there are some drawbacks in terms of grammar and sentence structure.

One potential explanation for these limitations is that our models were trained for a fixed number of epochs, specifically 20,000 epochs. Ideally, training should persist until the validation loss either reaches a plateau or starts to deteriorate. Additionally, we observe that the generated output often fails to consistently incorporate the context provided in the prompts, indicating a need for further refinement.

Nevertheless, it is important to note that while the generations might not be in line with the context, but the first few words generated make some sense given the last few words in the prompt. For example, given "*The bunny complained that she wanted a*", Model 1's output is "*good haircut*", which seems sensible. Interestingly, Model 1, which comprises 1 head and 4 layers, appears to exhibit slightly superior performance compared to other

models, despite the fact that other models possess a greater number of layers and heads.

One would expect that increasing the number of heads and layers should lead to improved performance, as these adjustments theoretically enable the model to capture more complex patterns and relationships within the data. However, in our experiments, we do not observe the expected improvement in performance.

One potential explanation for this unexpected outcome could be attributed to the fixed embedding size utilized in our experiments. Specifically, our models maintain a constant embedding size of 64 throughout. As we increase the number of heads from 1 to 4, the head size is proportionally reduced. Consequently, this reduction in head size may impose limitations on each head's capacity to effectively look at different aspects of the input data.

Moreover, word embeddings serve as vectorized representations of words, capturing both their semantic meaning and emotional connotations. When the embedding size is small, there's a risk wherein only the most crucial aspects of words are retained, while less significant nuances are overlooked. Consequently, adding layers and attention heads atop a small embedding size may yield diminished opportunities for meaningful learning. This constrained environment could restrict the model's capacity to capture the intricate subtleties inherent in language, thereby resulting in no significant improvement in the model's performance. This brings us to a question: does increasing the embedding size result in enhanced performance?

5.3 Impact of increasing the embedding size

In order to examine the influence of increasing the embedding size on English text generation, we will compare three models with embedding sizes of 64, 256, and 512, respectively. Notably, increasing the embedding size results in a considerable increase in the total number of model parameters. Specifically, the model with an embedding size of 64 encompasses approximately 7 million parameters, where those with embedding sizes of 256 and 512 harbor approximately

Prompt	Model 8	Model 7	Model 6
Once there was a bunny and a bear. The bunny was envious of the bear's new red glove.The bunny	wanted to help. He found a patch of the ground and happily ran back to help.	lived in a big castle. It was glad of the bear was special so high he could go!	and the bear were curious. The rabbit bear asked the rabbit the bear if he could grab something special
Once upon a time, there was a naughty	monkey. He lived in a big hole in the ground. He liked to play in the ground .	little boy named Timmy. Timmy loved to sleep, but he had never slept well on his	boy who lived too busy. He looked everywhere, though he was probably a big party. One day
Mum and Dad were getting ready for the day. They had been busy, packing and cleaning the house. Billy watched them and sighed. It was difficult for him to	finish breakfast just like before. His mummy and Daddy got into the car and drove to the	remain quiet. Even in the day, Billy 's pile with a hammer heard his mom my coming to	make a flame go. Johnny cried for an operation and there were sisters. So, he

Table 5.2: Impact of increasing embedding size : Story Completions (20 tokens)

29 million and 70 million parameters, respectively. The Table 5.2 provides a display of the outputs generated by each of these three models, facilitating a detailed comparison of their text generation capabilities.

In Table 5.2, we can distinctly observe the positive impact of increasing the embedding size on the quality of output generation. Notably, Model 1, having 70 million parameters, exhibits significant improvement in generating text characterized by proper word usage, grammatical correctness, and correct sentence structure. Although the outputs still fall short of fully capturing the contextual nuances provided in the prompts, the overall quality of English text generation has notably advanced.

Consider the example prompt: "*Mum and Dad were getting ready for the day. They had been busy, packing and cleaning the house. Billy watched them and sighed. It was difficult for him to*". In response, Model 8 generates the response: "*finish breakfast just like*

before. His mummy and Daddy got into the car and drove to the". While the output doesn't precisely encapsulate the entire context with perfect accuracy, it does reflect elements such as "getting ready for the day" and "packing" suggesting that the model is gradually learning to contextualize its output.

In contrast, Model 7 produces the response, "remain quiet. Even in the day, Billy 's pile with a hammer heard his mom my coming to". Although the sentence structure appears grammatically correct, the completion does not relate to the given context. This observation implies a promising trajectory towards enhancing the model's ability to grasp and reflect nuanced contextual cues in its generated text when we increase the embedding size and it further answers the question with which we started i.e. increasing the embedding size helps in enhancing model's performance.

CHAPTER 6

Conclusion

In this study, we aimed to investigate English text generation using a Small Language Model. To train our model, we utilized the TinyStories dataset. This dataset offers the advantage of having a limited vocabulary, while simultaneously being rich in linguistic elements such as grammar and reasoning. During the training process, we experimented with various model configurations.

This experiment has provided us with a deeper understanding of the fundamental workings of Small Language Models. We analyzed these models from two main perspectives: model training and English text generation. During the model training phase, we tracked the validation loss and compared its convergence across different model configurations. For the English text generation phase, we evaluated the performance of models with various configurations to determine how factors such as the number of attention heads, the number of layers, and the embedding size influence the quality of the generated text.

Our observations revealed that models with fewer parameters may require more time to learn effectively, while those with a larger number of parameters tend to learn more quickly. This indicates that to achieve comparable results, smaller models need to be trained for longer periods on more number of epochs. Consequently, we concluded that there is a trade-off between training time and model complexity.

Furthermore, our findings indicated that merely increasing the number of attention heads or layers does not necessarily lead to improved performance when the embedding size is kept constant. One plausible explanation is that increasing the number of heads while maintaining

a fixed embedding size reduces the size of each individual head. This reduction in head size may limit each head's ability to effectively learn and represent different aspects of the input data. Similarly, adding more layers without increasing the embedding size does not enhance model performance, likely because a limited embedding size restricts the model's ability to capture and represent the intricate nuances and complexities of the language data.

In conclusion, our study suggests that there must be a delicate balance between the number of attention heads, layers, and embedding size to optimize the performance of Small Language Models. Achieving this balance is crucial for enhancing the quality of text generation while maintaining efficient training times.

CHAPTER 7

Future Scope

There are several areas that require further investigation to build upon and further solidify this analysis.

First, exploring alternative methods of feeding input data. For this analysis, we combined all the stories in the TinyStories dataset into a single corpus of text, which we then used for training our model. Currently, our approach involves randomly selecting an index within the combined corpus and creating a block of a predefined size. The limitation of this method is that it might not allow the model to process entire stories. This could explain why our model performs better at sentence formation and adhering to grammar rules but struggles to maintain context. I propose that the model's performance could improve if we treat each story as a distinct block, meaning that each batch should consist of multiple complete stories. Given that the lengths of these stories vary, padding will be necessary. By feeding entire stories into the model, we could potentially resolve this issue and enhance the model's ability to capture and retain contextual information.

Second, evaluating the model's performance through a comprehensive analysis of its outputs using quantitative metrics, such as perplexity, is essential. Currently, we limited our training to a fixed number of epochs. Extending the training over a larger number of epochs and then comparing the perplexity scores could provide deeper insights into the model's performance in terms of English text generation.

Finally, in the previous section, we concluded that optimizing the performance of Small Language Models requires a delicate balance between the number of attention heads, layers,

and embedding size. It would be valuable to investigate when and how this balance can be achieved to maximize model performance and ensure the highest quality text generation.

In conclusion, these areas of investigation, alternative data input methods, comprehensive performance evaluation methods, and optimization of model architecture, are critical for advancing our understanding and capabilities in English text generation using Small Language Models.

REFERENCES

[1] Eldan, Ronen and Li, Yuanzhi, "TinyStories: How Small Can Language Models Be and Still Speak Coherent English?", 2023.

[2] Zhao, Wayne Xin and Zhou, Kun and Li, Junyi and Tang, Tianyi and Wang, Xiaolei and Hou, Yupeng and Min, Yingqian and Zhang, Beichen and Zhang, Junjie and Dong, Zican and Du, Yifan and Yang, Chen and Chen, Yushuo and Chen, Zhipeng and Jiang, Jinhao and Ren, Ruiyang and Li, Yifan and Tang, Xinyu and Liu, Zikang and Liu, Peiyu and Nie, Jian-Yun and Wen, Ji-Rong, Łukasz and Polosukhin, Illia,"A Survey of Large Language Models", 2023.

[3] Touvron, Hugo and Martin, Louis and Stone, Kevin and Albert, Peter and Almahairi, Amjad and Babaei, Yasmine and Bashlykov, Nikolay and Batra, Soumya and Bhargava, Prajjwal and Bhosale, Shruti and Bikel, Dan and Blecher, Lukas and Canton Ferrer, Cristian and Chen, Moya and Cucurull, Guillem and Esiobu, David and Fernandes, Jude and Fu, Jeremy and Fu, Wenyin and Fuller, Brian and Gao, Cynthia and Goswami, Vedanuj and Goyal, Naman and Hartshorn, Anthony and Hosseini, Saghar and Hou, Rui and Inan, Hakan and Kardas, Marcin and Kerkez, Viktor and Khabsa, Madian and Kloumann, Isabel and Korenev, Artem and Koura, Punit Singh and Lachaux, Marie-Anne and Lavril, Thibaut and Lee, Jenya and Liskovich, Diana and Lu, Yinghai and Mao, Yuning and Martinet, Xavier and Mihaylov, Todor and Mishra, Pushkar and Molybog, Igor and Nie, Yixin and Poulton, Andrew and Reizenstein, Jeremy and Rungta, Rashi and Saladi, Kalyan and Schelten, Alan and Silva, Ruan and Smith, Eric Michael and Subramanian, Ranjan and Tan, Xiaoqing Ellen and Tang, Binh and Taylor, Ross and Williams, Adina and Kuan, Jian Xiang and Xu, Puxin and Yan, Zheng and Zarov, Iliyan and Zhang, Yuchen and Fan, Angela and Kambadur, Melanie and Narang, Sharan and Rodriguez, Aurelien and Stojnic, Robert and Edunov, Sergey and Scialom, Thomas, "Llama 2: Open Foundation and Fine-Tuned Chat Models", 2023.

[4] Wei, Jason and Tay, Yi and Bommasani, Rishi and Raffel, Colin and Zoph, Barret and Borgeaud, Sebastian and Yogatama, Dani and Bosma, Maarten and Zhou, Denny and Metzler, Donald and Chi, Ed H. and Hashimoto, Tatsunori and Vinyals, Oriol and Liang, Percy and Dean, Jeff and Fedus, William,"Emergent Abilities of Large Language Models", 2022.

[5] Black, Sid and Biderman, Stella and Hallahan, Eric and Anthony, Quentin and Gao, Leo and Golding, Laurence and He, Horace and Leahy, Connor and McDonell, Kyle and Phang, Jason and Pieler, Michael and Prashanth, USVSN Sai and Purohit, Shivanshu and Reynolds, Laria and Tow, Jonathan and Wang, Ben and Weinbach, Samuel,"GPT-NeoX-20B: An Open-Source Autoregressive Language Model", 2022.

[6] Chowdhery, Aakanksha and Narang, Sharan and Devlin, Jacob and Bosma, Maarten and Mishra, Gaurav and Roberts, Adam and Barham, Paul and Chung, Hyung Won

and Sutton, Charles and Gehrmann, Sebastian and Schuh, Parker and Shi, Kensen and Tsvyashchenko, Sasha and Maynez, Joshua and Rao, Abhishek and Barnes, Parker and Tay, Yi and Shazeer, Noam and Prabhakaran, Vinodkumar and Reif, Emily and Du, Nan and Hutchinson, Ben and Pope, Reiner and Bradbury, James and Austin, Jacob and Isard, Michael and Gur-Ari, Guy and Yin, Pengcheng and Duke, Toju and Levskaya, Anselm and Ghemawat, Sanjay and Dev, Sunipa and Michalewski, Henryk and Garcia, Xavier and Misra, Vedant and Robinson, Kevin and Fedus, Liam and Zhou, Denny and Ippolito, Daphne and Luan, David and Lim, Hyeontaek and Zoph, Barret and Spiridonov, Alexander and Sepassi, Ryan and Dohan, David and Agrawal, Shivani and Omernick, Mark and Dai, Andrew M. and Sankaranarayana Pillai, Thanumalayan and Pellat, Marie and Lewkowycz, Aitor and Moreira, Erica and Child, Rewon and Polozov, Oleksandr and Lee, Katherine and Zhou, Zongwei and Wang, Xuezhi and Saeta, Brennan and Diaz, Mark and Firat, Orhan and Catasta, Michele and Wei, Jason and Meier-Hellstern, Kathy and Eck, Douglas and Dean, Jeff and Petrov, Slav and Fiedel, Noah,"PaLM: Scaling Language Modeling with Pathways", 2022.

[7] Kaplan, Jared and McCandlish, Sam and Henighan, Tom and Brown, Tom B. and Chess, Benjamin and Child, Rewon and Gray, Scott and Radford, Alec and Wu, Jeffrey and Amodei, Dario,"Scaling Laws for Neural Language Models", 2020.

[8] Bhattamishra, Satwik and Ahuja, Kabir and Goyal, Navin,"On the ability and limitations of transformers to recognize formal languages", 2020.

[9] Brown, Tom B. and Mann, Benjamin and Ryder, Nick and Subbiah, Melanie and Kaplan, Jared and Dhariwal, Prafulla and Neelakantan, Arvind and Shyam, Pranav and Sastry, Girish and Askell, Amanda and Agarwal, Sandhini and Herbert-Voss, Ariel and Krueger, Gretchen and Henighan, Tom and Child, Rewon and Ramesh, Aditya and Ziegler, Daniel M. and Wu, Jeffrey and Winter, Clemens and Hesse, Christopher and Chen, Mark and Sigler, Eric and Litwin, Mateusz and Gray, Scott and Chess, Benjamin and Clark, Jack and Berner, Christopher and McCandlish, Sam and Radford, Alec and Sutskever, Ilya and Amodei, Dario,"Language Models are Few-Shot Learners", 2020.

[10] Lewis, Mike and Liu, Yinhan and Goyal, Naman and Ghazvininejad, Marjan and Mohamed, Abdelrahman and Levy, Omer and Stoyanov, Ves and Zettlemoyer, Luke,"BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension", 2020.

[11] Radford, Alec and Wu, Jeffrey and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya,"Language Models are Unsupervised Multitask Learners", 2019.

[12] Liu, Yinhan and Ott, Myle and Goyal, Naman and Du, Jingfei and Joshi, Mandar and Chen, Danqi and Levy, Omer and Lewis, Mike and Zettlemoyer, Luke and Stoyanov, Veselin,"RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019.

[13] Thomas Wolf and Lysandre Debut and Victor Sanh and Julien Chaumond and Clement Delangue and Anthony Moi and Pierric Cistac and Tim Rault and Rémi Louf and Morgan Funtowicz and Joe Davison and Sam Shleifer and Patrick von Platen and Clara Ma and Yacine Jernite and Julien Plu and Canwen Xu and Teven Le Scao and Sylvain Gugger and Mariama Drame and Quentin Lhoest and Alexander M. Rush,"HuggingFace's Transformers: State-of-the-art Natural Language Processing", 2019.

[14] Peters, Matthew E. and Neumann, Mark and Iyyer, Mohit and Gardner, Matt and Clark, Christopher and Lee, Kenton and Zettlemoyer, Luke,"Deep contextualized word representations", 2018.

[15] Shen, Yikang and Lin, Zhouhan and Huang, Chin-Wei and Courville, Aaron,"Neural language modeling by jointly learning syntax and lexicon", 2017.

[16] Vaswani, Ashish and Shazeer, Noam and Parmar, Niki and Uszkoreit, Jakob and Jones, Llion and Gomez, Aidan N and Kaiser, Łukasz and Polosukhin, Illia,"Attention is all you need", 2017.

[17] Gao, Jianfeng and Lin, Chin-Yew,"Introduction to the special issue on statistical language modeling", 2004.