NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2024-15

June, 2024

Stolen Attention in Transformers

Kyle Williams

Abstract

In the field of Natural Language Processing (NLP), today's best models rely on the construction of an embedding space. Upon input to a language model, tokens are translated into dense, high-dimensional vectors. Over the process of training, tokens' embeddings are organized for spatial-relatedness. This allows models to greatly generalize, as the meaning of a word can be expressed in terms of neighboring embeddings.

A language model has a fixed vocabulary, whose embeddings are expressed using weight matrices. The matrix at the input of the network translates a token to its internal representation. As it is shown a sequence, the language model builds a representation of its prediction vector. At the output, this vector is compared against embeddings in the output matrix to select the next word. This is traditionally done with dot-product softmax.

Unfortunately, certain tokens have reduced potential to receive probability mass under dot-product softmax. Embeddings which are interior to the convex hull of the output weight matrix cannot be assigned high probability. While this effect has been previously studied in other language models, the transformer architecture also uses dot-product softmax in another location: the attention mechanism. In this work, we demonstrate the Stolen Attention Effect, where certain embeddings cannot be assigned high attention weights. Further, we suggest an architectural change that may overcome the effect. By using a variant of Euclidean distance instead of dot-product in the attention mechanism, we yield a 14.27% improvement on OpenBookQA.

Keywords

Artificial Intelligence, Machine Learning, Transformers, Language Modeling, Question-Answering, Embedding Spaces, Generative AI

NORTHWESTERN UNIVERSITY

Stolen Attention in Transformers

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Computer Science

By

Kyle Williams

EVANSTON, ILLINOIS

June 2024

ABSTRACT

Stolen Attention in Transformers

Kyle Williams

In the field of Natural Language Processing (NLP), today's best models rely on the construction of an embedding space. Upon input to a language model, tokens are translated into dense, high-dimensional vectors. Over the process of training, tokens' embeddings are organized for spatial-relatedness. This allows models to greatly generalize, as the meaning of a word can be expressed in terms of neighboring embeddings.

A language model has a fixed vocabulary, whose embeddings are expressed using weight matrices. The matrix at the input of the network translates a token to its internal representation. As it is shown a sequence, the language model builds a representation of its prediction vector h_t. Then at the output, the prediction vector is compared against embeddings in the output matrix to select the next word. This is traditionally done with dot-product softmax.

Unfortunately, certain tokens have reduced potential to receive probability mass under dot-product softmax. Embeddings who are interior to the convex hull of the output weight matrix cannot be assigned high probability. While this effect has been previously studied

in other language models, the transformer architecture also uses dot-product softmax in another location: the attention mechanism. In this work, we demonstrate the Stolen Attention Effect, where certain embeddings cannot be assigned high attention weights. Further, we suggest an architectural change that may overcome the effect. By using a variant of Euclidean distance instead of dot-product in the attention mechanism, we yield a 14.27% improvement on OpenBookQA.

Acknowledgements

My Thesis committee consisted of Professors David Demeter and Doug Downey. As my first official research experience, their guidance was invaluable. They provided insightful feedback, challenged my assumptions, and encouraged me to think critically and creatively. Their mentorship significantly enhanced the quality of my work, and have left lasting impressions on me and my academic goals. They deserve special thanks for their devotion and support throughout this project.

Next, I'd like to thank Northwestern for an incredible five years. I've had a great experience as a part of the Computer Science department, and the faculty hold themselves to high standards. Their dedication to excellence has provided me with a robust education and a supportive environment for growth. I am especially grateful for the facilities and resources that have allowed me to explore my interests and push the boundaries of my knowledge. My time at Northwestern has been transformative; I will carry the lessons and experiences with me throughout my career.

Finally, I'd like to thank my family and friends. In the rocky parts of this project, I inevitably doubted myself. Luckily, I have an incredible support system whose unwavering support and encouragement motivated me to persevere. Without their love and understanding, my achievements would not have been possible. I am deeply grateful for their presence in my life.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1. Natural Language Processing and Representation Learning

Natural Language Processing (NLP) is a long-studied field of research. NLP is an umbrella term, which refers to any task involving human language. The motivating example is machine translation; if algorithms could successfully translate text from different languages, transcripts could become accessible to other languages besides the one in which it was written. Performing an NLP task starts with a language model. A language model aims to understand and generate human language by learning patterns from large datasets of text. Each dataset is known as a corpus, which contains sequences of tokens. A language model has a fixed vocabulary of tokens it can recognize and use. Using sentences from the corpus, it learns a conditional probability distribution to predict the next token given its preceding context.

In 2003, Bengio et al. [**1**] changed the landscape of NLP with the advent of representation learning. Until that point, neural language models represented tokens as a one-hot encoding. The use of large, sparse input vectors caused them to suffer from the curse of dimensionality. Furthermore, the parameter count increased too quickly to scale models to long sequence lengths, making them limited in practice. Finally, and most importantly, sequences not seen in the training corpus were impossible to model. To solve these problems, they first translated tokens from their sparse, one-hot representations to dense, low-dimensional vectors known as embeddings. Then a multi-layer perceptron could learn the embeddings and the conditional probability distribution simultaneously. These embeddings captured semantic relationships between tokens; the spatial locality of tokens can be used to model sequences that aren't explicitly in the training set.

While the formulation of Bengio et al. enabled modeling longer sequence lengths, its feed-forward architecture was limited. For starters, the context length was fixed and could never be enlarged. Furthermore, the embedding matrix was shared for all positions of the input. Thus, any temporal dependencies had to be captured by the weights of feed-forward layers. In 2010, Mikolov et al. [**15**] adapted the Recurrent Neural Network (RNN) to language modeling. RNNs handle sequences by maintaining a hidden state that evolves over time. Their construction ensures temporal dependencies between tokens are built-in as an inductive bias. More importantly, the context length can be arbitrarily large. They were eventually superseded by LSTMs [**8**], an improvement on RNNs with respect to modeling long contexts.

After its proposal by Vaswani et al. [**23**] in 2017, the transformer architecture has become dominant. A transformer is limited in that its maximum context length is bounded, and must be chosen prior to training [1]. While this is a step backward from the LSTM, it gains the ability to be parallelized with respect to the sequence. LSTMs must crunch a sequence one-token-at-a-time to build up its hidden state. Instead, the transformer utilizes attention blocks to capture long-range dependencies. Attention can be independently applied to all positions of the sequence, removing the sequential dependency. Thus,

[1]Some recent work [**17**], [**21**] has allowed transformers to arbitrarily extend their context lengths through approximations.

entire sequences can be computed in parallel, greatly increasing training speed. As hardware has continued to improve, transformers have demonstrated impressive scalability. When the number of layers and parameters increases, transformers seemingly show unbounded performance, a trend not observed in other architectures to the same extent. This scalability has led to the development of very large models, such as BERT [**6**], GPT [**19**], and T5 [**20**], which have set new benchmarks in various NLP tasks like translation, question-answering, and text summarization.

1.2. Dot-Product Softmax and Stolen Probability

While many different neural architectures have been proposed for NLP, one aspect of their design is unanimous. Recall that the goal of a neural language model is to learn the conditional probability distribution of the next token given its context. As such, it is crucial that the outputs of the network are non-negative and sum to 1. Due to the paradigm of representation learning, any model's internal vectors are less than the dimensionality of its vocabulary. A linear weight matrix W_O projects a model's final representation of the next token to a distribution over vocabulary words. This distribution is transformed via softmax to be interpreted as probabilities.

Recall that input tokens are translated to embeddings using a linear weight matrix W_I. Thus, $W_I \in \mathbb{R}^{|V| \times d}$, where d is the dimensionality of the model's embeddings and V is its vocabulary. Similarly, to translate embeddings into a distribution over the vocabulary, it must be that $W_O \in \mathbb{R}^{d \times |V|}$. The weights inside W_O can also be thought of as an embedding matrix. In fact, some transformers use tied embeddings, where the model explicitly uses $W_O = W_I^T$. Given a final representation h_t, the operation $h_t \cdot W_O$ essentially computes the dot-product similarity between h_t and the embeddings of our vocabulary. Applying softmax normalizes the dot-product scores into a probability distribution, and the entire operation is known as dot-product softmax.

In 2020, Demeter et al. [**5**] discovered a problem with the application of dot-product softmax at the output of LSTM language models. Again, to achieve probabilities as outputs, the final hidden state of the LSTM (h_t) is multiplied by a weight matrix W_O. Once the LSTM has completed training, W_O represents output embeddings that h_t will be ranked against using dot-product softmax. Unfortunately, tokens whose embeddings have small L_2 norms have their maximum probability thresholded. Specifically, W_O defines some set of vectors in a high-dimensional space. A subset of these vectors form a convex hull which contains all other vectors. Those who are interior to the hull cannot be assigned high probability, no matter the context.

To make matters worse, an association between a token's frequency in the training corpus and its embedding norm was discovered. Language modeling corpi are known as "long-tailed" because most tokens occur only a few times. Yet, leveraging an infrequent word is often paramount to the logical continuation of a sequence. For example, "America" is not a common word, but should almost always follow "United States of". Because high frequency words have larger embeddings on average, they are often vertices of the convex hull. Thus, these high-frequency, low-information token "steal" probability.

1.3. Stolen Attention in Transformers

In their work, Demeter et al. tested and documented Stolen Probability in LSTMs with an embedding dimensionality as large as $d = 200$ [**5**]. However, it was theorized that the Stolen Probability effect becomes marginal as d increases. Intuitively, with larger embeddings, there is more room to spread them around the embedding space. Then there should be less potential for a given embedding to be inside the convex hull. Even if an embedding has a small L_2 norm, with sufficient dimensionality, it will likely extend far enough in at least one dimension so as to not be contained.

Today's transformers use very large embedding spaces. For example, common choices in literature are $d = 512$, $d = 768$, or $d = 1024$. Moreover, high-performance foundation models like Google's PaLM-540B use excessive sizes like $d = 18432$ [**4**]. With embedding spaces this large, the potential for stolen probability is diminishing. However, transformers use dot-product softmax in another location besides the output: the attention mechanism. Here, multiple attention heads divide the model's embeddings into several subspaces. That is, with h heads, each head operates over vectors of size d/h. So, while Stolen Probability may not be present, Stolen Attention could be a very real threat. Despite its large overall dimensionality, even PaLM-540B uses a head dimensionality of 256, not much larger than was tested by Demeter et al. Smaller transformers may be even more at risk. In this work, we wish to document the extent to which Stolen Attention is present in transformer models.

1.4. Related Work

To our knowledge, we are the first to consider the impact of the distance metric (namely dot-product softmax) on the distribution of attention weights in transformer models. For the most part, work on the attention mechanism is focused on improving its computational efficiency. This is mostly done through modifications to the mechanism as a whole. For example, Wang et al. [**25**] linearize self-attention with the assumption that attention is low-rank. Or, Qin et al. [**18**] replace softmax entirely using other functions that maintain the non-negativeness of logits. Other bodies of work limit [**27**] or expand [**17**] the positions that can be paid attention to. These may have confounding effects on the development of the K matrix of attention heads, so we stick to vanilla self-attention.

CHAPTER 2

Background

2.1. The Transformer Architecture

A transformer has a fixed vocabulary V and can accept sequences up to length N. Thus the input to a transformer is a sequence of tokens $S = \{t_1, t_2, \ldots, t_N\}$, where $t_i \in V$. Each t_i is represented as a one-hot encoding, so $S \in \mathbb{R}^{N \times |V|}$. For representation learning, tokens are translated into embeddings of size d via an embedding matrix $E_{\text{in}} \in \mathbb{R}^{|V| \times d}$. After embedding the sequence, a transformer consists of L layers. Each layer $i \in \{1, 2, \ldots, L\}$ consists of a Multi-Head Attention (MHA) mechanism A_i followed by a feed-forward network F_i. Residual connections help propagate gradient signals, while layer normalization is added to avoid explosions. Let the input to layer i be $x_i \in \mathbb{R}^{N \times d}$. Then a transformer layer is a function $T_i : \mathbb{R}^{N \times d} \to \mathbb{R}^{N \times d}$ such that:

$$T_i(x_i) = F_i(A_i(x_i) + x_i) + A_i(x_i) + x_i \tag{2.1}$$

Above, the input to T_1 would be $x_1 = SE_{\text{in}}$. The input to T_2 would be $x_2 = T_1(x_1)$, the input to T_3 would be $x_3 = T_2(x_2)$, etc. While residual connections are shown in the formula above, layer normalization has been omitted. There are two main schools of thought in regards to layer normalization: Post-LN and Pre-LN. Post-LN was implemented in the original Transformer proposal [**23**], but Pre-LN was found to enable the training of deeper Transformers [**24**]. Nowadays, the options are relatively interchangeable. Large models can be trained with either strategy to comparable performance, given that the optimizer and learning rate schedule are chosen appropriately. We refer readers to [**26**] for a discussion of these strategies and their pros/cons.

The significant contribution of the transformer architecture, and the focus of this paper, is the multi-head attention (MHA) module A_i. In its most general form, MHA operates over three matrices $Q, K, V \in \mathbb{R}^{N\times d}$. Q is known as the query matrix, K as the key matrix, and V as the values matrix. Then $A_i(Q, K, V)$ is the output of the mechanism. Here, we're focused on a decoder-only architecture akin to GPT [**19**]. In this setting, $Q, K, V = x_i$. This is known as "self-attention", because the queries, keys, and values are derived from the same input.

In an MHA module with k heads, each head $j \in \{1, 2, \ldots, k\}$ has three weight matrices $W_Q^j, W_K^j, W_V^j \in \mathbb{R}^{d\times d_h}$, where $d_h = d/k$. Thus, the heads split the transformer's embeddings into several subspaces of smaller size. The use of multiple heads allows the model to jointly attend to information at different positions of the sequence. Furthermore, because each head has its own subspace, it will pay attention to different features of the embedding that may relate words. For example, some heads may focus on semantics, while others focus on positionality or punctuation. The output of head j in layer i is $h_j : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d_h}$:

$$h_j(x_i) = \text{softmax}[D(x_i W_Q^j, x_i W_K^j)] \cdot x_i W_V^j \tag{2.2}$$

Above, $x_i W_Q^j$, $x_i W_K^j$, and $x_i W_V^j$ can be viewed as head-specific query, key, and value matrices for head j. Then D is some distance metric measuring the similarity between the query and key embeddings of that head. Let $Q_j = x_i W_Q^j$ and $K_j = x_i W_K^j$. The standard function for D was introduced in the original Transformer paper [**23**] and has been largely unchanged. Known as "scaled dot-product attention", it is defined as:

$$D(Q_j, K_j) = \frac{Q_j K_j^T}{\sqrt{d_h}} \tag{2.3}$$

Putting it all together, the MHA module for layer i concatenates the outputs of each of its heads. Then it mixes their contributions using a linear weight matrix $W_O^i \in \mathbb{R}^{d \times d}$. As such, it is imperative that d is divisible by k; concatenation results in an output of size $\mathbb{R}^{N \times hd_h}$, and we need $hd_h = d$. Let h_{ij} be the output of head j in layer i, and $[\cdot]$ be the concatenation operation. Then MHA is defined as:

$$A_i(x_i) = [h_{i1}(x_i), h_{i2}(x_i), \ldots, h_{ik}(x_i)]W_O^i \tag{2.4}$$

Finally, the output of the transformer is a matrix $O \in \mathbb{R}^{N \times |V|}$. Each row $n \in \{1, 2, \ldots, N\}$ is a conditional probability distribution over the vocabulary space given the previous $n-1$ tokens of the sequence. In other words, $\forall j \in 1, 2, \ldots, |V|$, $O_{nj} = P(t_n = t_j \mid t_1, t_2, \ldots, t_{n-1})$. This is achieved by multiplying the output of the last transformer layer by a weight matrix $E_{\text{out}} \in \mathbb{R}^{d \times |V|}$. Similarly to the rows of the input embedding matrix, the columns of the output matrix represent embeddings for the tokens of models' vocabulary. For instance, Transformers with tied embeddings explicitly use $E_{\text{out}} = E_{\text{in}}^T$. Formally, the output is given as:

$$O = \text{softmax}[T_L(x_{L-1})E_{\text{out}}] \tag{2.5}$$

2.2. Stolen Probability

Any trained neural language model will have some output embedding matrix $E_{\text{out}} \in \mathbb{R}^{d \times |V|}$. While we focus on transformers in this work, our hypothesis is based on [**5**]. Here, Demeter et al. use an LSTM language model instead. We refer readers to the original LSTM paper [**8**] for details on its operation. With respect to our analysis, the important part of the LSTM is its output.

Again, let $S = \{t_1, t_2, \ldots, t_N\}$ be some sequence of tokens. For an LSTM, N is not bounded, and can be arbitrarily large. Then an LSTM receives tokens t_i sequentially as input. Along the way, it builds a hidden state vector representing its guess as to t_{i+1} given $\{t_1, t_2, \ldots, t_i\}$. Let this vector be $h_i \in \mathbb{R}^{1 \times d}$. At step i, the output of the LSTM is $O_i = \text{softmax}[h_i E_{\text{out}}]$.

Intuitively, $\forall j \in \{1, 2, \ldots, |V|\}$, each row e_j of E_{out}^T represents the embedding for token t_j. As such, the operation $h_i E_{\text{out}}$ can be thought of as calculating the dot-product similarity between the hidden state h_i and each of the tokens' embeddings e_j. These outputs (prior to softmax) are known as logits $z_j = \langle h_i, e_j \rangle$, where $\langle \cdot \rangle$ denotes inner-product. By performing softmax over this output, we assign probability such that tokens whose embeddings are most similar to our hidden state receive the highest probability. Using the formula for dot-product, these logits can be re-written as:

$$z_j = ||e_j||\ ||h_i|| \cos(\theta_j) \tag{2.6}$$

Above, $||\cdot||$ denotes L_2 norm, and θ_j is the angle between the vectors e_j, h_i. Here, h_i is fixed for all logits, and is known as the query point. For LSTMs, empirical evidence (not presented) shows that while embeddings e_j have a wide distribution of L_2 norms, they are organized in a narrow cone relative to some reference point. As such, there is little variation in θ_j, and $||e_j||$ dominates the calculation of logits. Dubbed the Stolen Probability Effect, this is a consequence of the way neural language models organize their embedding spaces.

To formalize, the convex hull is the unique minimal convex set containing a set of vectors X. The vectors in the set are the vertices of the convex hull; they encompass the remaining interior vectors. In this case, we're interested in C, the convex hull of E_{out}^T. Here, Demeter et al. [**5**] prove that if the embedding e_j for token t_j is interior to C, the maximum probability that can be assigned to t_j using dot-product softmax is bounded by the probability of at least one t_k $(k \neq i)$ whose embedding e_k is a vertex of C. Because the embedding matrix is fixed after training, some tokens may never receive high probability simply due to their placement within the embedding space.

In general, smaller embedding spaces suffer more drastically from Stolen Probability. The lower a model's embedding dimensionality d, the smaller the percentage of tokens whose embeddings are vertices of the convex hull of E_{out}^T. This means more tokens are interior, and have their maximum probability assignments bounded. Language models are commonly evaluated using a perplexity, a measure that essentially captures the confusion of a model (lower is better). In terms of this metric, the Stolen Probability Effect may not always be noticeable. Especially for larger d, fewer embeddings are interior to the hull, and the embedding space is fully expressive. Still, for certain sentences, there can be severely sub-optimal placements of probability mass.

CHAPTER 3

Stolen Attention

Transformers also use dot-product softmax at the output of the network, and presumably suffer from Stolen Probability. There, a hidden state vector h is compared against the embeddings in the weight matrix E_{out}^T. Then softmax is applied over the result to create a probability distribution over the vocabulary words. But Transformers also use dot-product softmax in the calculation of self-attention. In self-attention, each row i of $D(Q, K)$ (2.3) represents a dot-product similarity comparison between the query vector q_i at position i of the sequence and all key vectors in K. Then softmax is applied over the scores to assign attention weights.

The vectors in Q, K come from a smaller embedding space that is unique to each head. These spaces are formed by down-projecting the models' full-length, contextualized embeddings using the W_Q, W_K matrices in each head. When dot-product softmax is applied to them in the calculation of self-attention, heads may be subject to a new effect: Stolen Attention. Here, relative to a reference point q_i, tokens whose key vectors k_j lie inside the convex hull of the K matrix will have their maximum attention weight bounded. Each head may individually experience this effect depending on how it arranges its embedding space. Furthermore, each sequence incurs a different K matrix, making Stolen Attention highly nuanced. For example, certain tokens' key vectors may become vertices of the convex hull depending on their position in the sequence and the surrounding context.

The most important part of confirming the presence of Stolen Attention is the calculation of the convex hull. In self-attention, every head of every layer projects its input to an internal set of query, key, and value matrices. The queries and keys are compared using dot-product, then softmax is applied column-wise to assign separate attention weights to the keys for every query. This means we must calculate the convex hull of the embeddings of the K matrix for each head of each layer. With this information, we can see if the maximum attention weight assigned to a key vector is bounded when it is a vertex of the convex hull.

Because we focus on Decoder-only transformers, the size of the sequence is important in our consideration of the convex hull. Here, an auto-regressive mask ensures attention cannot be paid to tokens beyond the current position of the sequence. Thus, we can only consider the convex hull (and corresponding attention weights) formed over embeddings that are not eliminated by the mask. For example, let a sequence be $S = t_1, t_2, \ldots, t_N$. In the prediction of t_2, our only reference token is t_1. Thus, every attention head will place all of its attention on t_1. As we increase the sequence length to predict t_i given $t_1, t_2, \ldots, t_{i-1}$, there is more potential to vary attention weights. Simply put, longer sequences require us to rank our query vector q_i against more keys $k_1, k_2, \ldots, k_i$.

Think of the process of adding keys incrementally as the sequence length increases. To a certain extent, the keys we add represent entirely new directions in our embedding space. Thus, they must become vertices of the convex hull or they would not be contained by it. Beyond that point, new keys introduce no new variation in the embedding space, and will only become a vertex if they have a sufficiently large norm. We hypothesize that larger context lengths will worsen the Stolen Attention Effect as more embeddings become interior to the hull of the K matrix.

To that end, we want to analyze the Stolen Attention Effect upon fine-tuning a model in this work. Other studies have shown that pre-training perplexity is not necessarily indicative of downstream performance [**16**]. One potential explanation for this is that pre-training biases attention weights. Intuitively, the identity of the next token is most impacted by the few tokens that immediately precede it. As such, the length of the input sequence is not very important. With some minor exceptions, attention will always be paid to a select few tokens at the end of the context. Ideally, to demonstrate Stolen Attention, we need a task that has long context lengths and may require attention to be placed anywhere in the sequence.

CHAPTER 4

Experimental Setting

4.1. Architecture and Hyper-parameters

In this work, we focus on the potential for Stolen Attention in Transformers utilizing self-attention. Specifically, all models we study will be Decoder-only transformers. While encoder-only transformers like BERT [**6**] also use self-attention, their inability to generate text make them less universal. After pre-training, a BERT model must be augmented with a fine-tuned classification layer to solve downstream tasks. On the other hand, large generative models can solve a variety of downstream tasks without modifications. This is done through scaling, careful pre-training, and by following instructions [**19**], [**2**].

Unfortunately, we are in a somewhat resource-constrained environment. Thus, we make a number of architectural decisions to ensure training stability and good convergence without the need for hyperparameter tuning. For both pre-training and fine-tuning, we use RAdam [**11**] with $(\beta_1, \beta_2) = (0.9, 0.99)$ and $\epsilon = 1 \times 10^{-6}$. This variant of Adam [**9**] was invented to reduce the variance of adaptive learning rates without the need for warmup. Moreover, we use Pre-LayerNorm for two reasons. First, it has been shown to achieve good performance without warmup, giving us an additional reason to remove it [**26**]. Second, Post-LayerNorm Transformers are more sensitive to hyper-parameter choices [**12**]. For increased training stability, weights are initialized using the Scaled Embed setting suggested by Takase et al. [**22**]. The input and output embedding layers have no bias terms. Other layers' biases are initialized to 0. We use sinusoidal positional encodings [**23**].

We hypothesize that the Stolen Attention Effect will vary with a models' head dimensionality d_h. To change d_h for experimentation, we can either change the overall embedding dimensionality d, or the number of attention heads h. Between the two, we choose to vary h; this allows us to keep a constant parameter count between models. The table below summarizes the hyper-parameters of the four models we investigate in this work. To simplify the communication of results, each model is named with respect to the number of attention heads it has.

Name	d	h	d_h	f	p	L	N	$\|V\|$	**# Params**
8-heads	512	8	64	2048	0.1	12	512	16000	54.2M
32-heads	512	32	16	2048	0.1	12	512	16000	54.2M
64-heads	512	64	8	2048	0.1	12	512	16000	54.2M
128-heads	512	128	4	2048	0.1	12	512	16000	54.2M

Table 4.1. Hyper-parameters of trained models. d is model dimensionality, h is the number of heads, d_h is the dimensionality of a head, f is the feed-forward dimensionality, p is the dropout proportion, L is the number of layers, $|V|$ is the vocab size, and N is the maximum sequence length.

4.2. Datasets and Training Details

For reproducibility, all parts of the pre-training and fine-tuning processes are seeded identically. This ensures runs are repeatable, and that models receive the same batches of data at each step. Seeding is performed via `L.seed_everything(seed, workers=True)`, where `seed=7` for pre-training and `seed=10` for fine-tuning. For more details on this function, check out the PyTorch Lightning documentation. Our hardware was of a cluster of 3 NVIDIA Quadro RTX 8000 GPUs running PyTorch 1.12.1, PyTorch Lightning 2.1.0, and

CUDA 11.4. Computations were performed using automatic mixed precision. Gradients are clipped to a norm of 1.0.

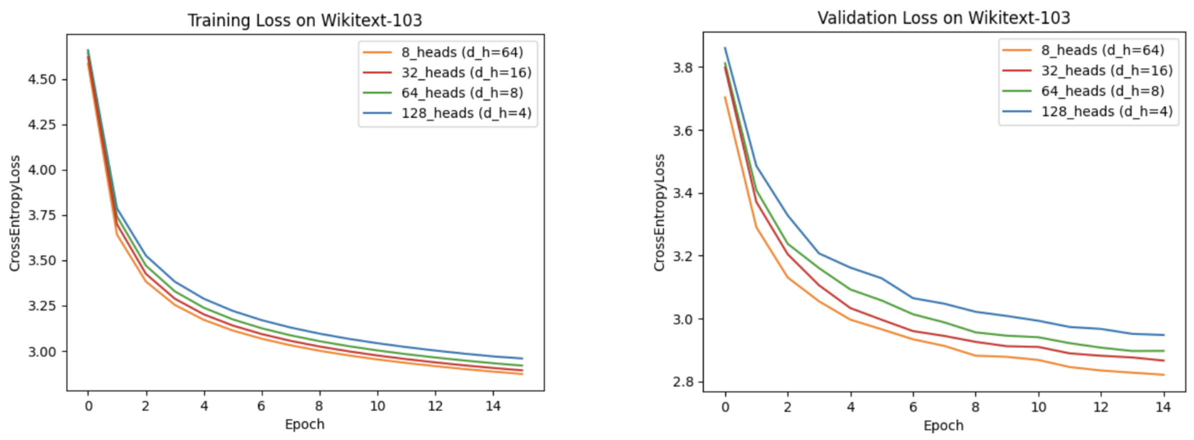
To pre-train models, we utilize the wikitext-103 corpus [**13**]. This dataset consists of full articles from the set of Good and Featured articles on Wikipedia. It has designated training, validation, and test sets; we pre-train on the training set and disregard the test set. Our models use a vocabulary of size $|V| = 16000$, which come from a Unigram tokenizer [**10**] that was trained on the raw text from the training set. We use the sentencepiece implementation of the unigram algorithm. To present data to the model, we simply tokenize the raw text, then pack tokens into sequences of length 512. We do not use `[SEP]` tokens, and our packing does not attempt to preserve full sentences. All models are trained for 100K steps using a batch size of 40, consuming a total of 2.05B tokens. We use an initial learning rate of 1×10^{-4}, which is decayed to 0 over time by a Reflected Exponential Schedule [**3**]. This schedule is empirically performant and has no hyper-parameters.

After pre-training, models are fine-tuned for the OpenBookQA [**14**] task. This dataset consists of 4957 train, 500 validation, and 500 test questions. We do not use the test set. Each multiple-choice question is based on one of 1326 elementary level science facts. The validation and test questions are considerably higher-quality (and hence more difficult). To achieve good performance on this task, a model must leverage background knowledge and multi-hop reasoning learned through pre-training. For each example, we present the model with the science fact on which the question is based, followed by the question itself, and then the answer choices. An example encoding can be found in Appendix A. The model must generate the token representing the correct answer choice (A, B, C, or D).

The token assigned the highest probability is considered the model's answer. Here, we omit the learning rate schedule and opt for a constant learning rate of 1×10^{-5}. Models are fine-tuned for 20 epochs with a batch size of 32.

4.3. Loss and Accuracy Curves

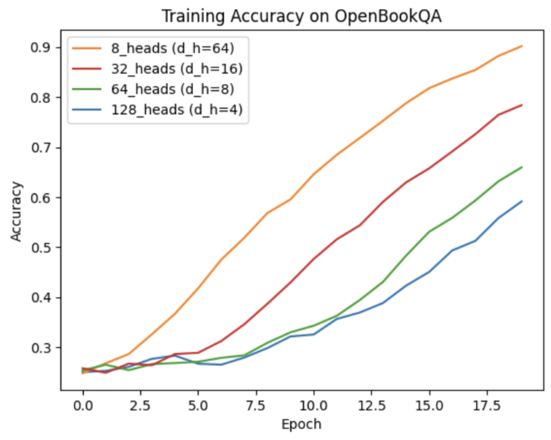
Below is the evolution of training and validation loss of the four models as they are trained on the wikitext-103 corpus. Training loss is averaged over all positions of the sequence, then averaged over the batches in the epoch. Validation loss is reported in a sliding window setting; it is averaged over the last position of all sequences in the validation set. The models are trained for 100K steps, and evaluated after every epoch they complete. The 100K-step threshold doesn't complete the 15th epoch, so the terminal validation loss of each model is not depicted in the plot. The models are ranked in order of number of heads, suggesting that having fewer heads is better for pre-training on wikitext-103.



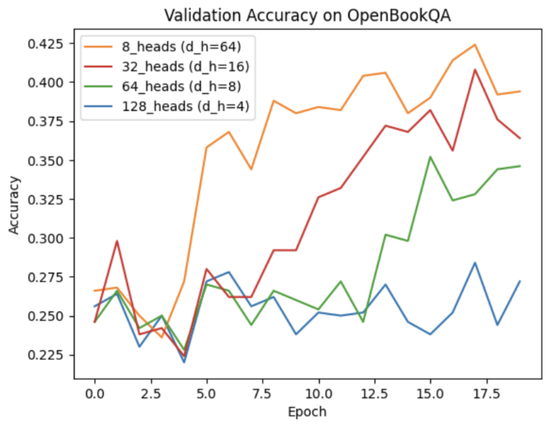
(a) Training Loss (b) Validation Loss

Figure 4.1. Training and Validation Loss during Pre-Training

Next is the evolution of training and validation accuracy of the four models as they are fine-tuned on the OpenBookQA dataset. Each model is fine-tuned from its final checkpoint at the completion of pre-training. The models are trained for 20 epochs, and evaluated after every epoch they complete. Performance is correlated with pre-training loss. That is, the better a model does on wikitext-103, the better it does on OpenBookQA. As such, having fewer heads seems to be advantageous for fine-tuning, too.



(a) Training Accuracy



(b) Validation Accuracy

Figure 4.2. Training and Validation Accuracy during Fine-Tuning

Future analysis involving OpenBookQA will be performed with the checkpoint that achieved the best validation accuracy, even if training accuracy continued to improve. Our attention statistics are collected within heads when decoding the token for the correct answer choice (A, B, C, or D). Due to the causal mask in a decoder-only transformer, this is the only place in the sequence where attention is spread along the entire context. This allows us to compare our query embedding (which is always for the token `[ANS]`) against the full K matrix. Here are the final training and validation performances of the model checkpoints on the respective datasets:

	Wikitext-103 Perplexity		**OpenBookQA Accuracy (%)**	
Name	**Train**	**Valid**	**Train**	**Valid**
8-heads	15.067	15.386	99.072	42.000
32-heads	15.430	15.805	89.974	40.800
64-heads	15.819	16.199	63.506	35.200
128-heads	16.448	17.185	63.204	28.400

Table 4.2. Training and Validation Performances of all Models

CHAPTER 5

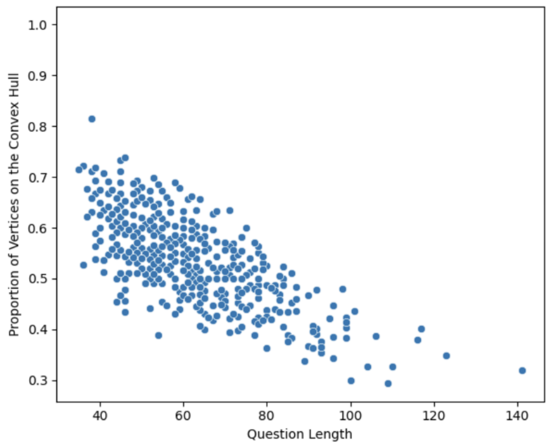
Observations and Analysis

To analyze the extent to which Stolen Attention exists in Transformers, we take the four model checkpoints (one for each of $h = 8, 32, 64, 128$) that had the best validation accuracy on OpenbookQA. Then we run questions from the validation set through each model. For each head of every layer, we capture a number of statistics. Specifically, for each token in the question, we note its position in the sequence, the norm of its key embedding inside each attention head, and the attention weight assigned. For $h = 64, 128$, we also calculate the convex hull and take note of whether a token was a vertex of it. Unfortunately, this cannot be done for all models; QuickHull is computationally intractable beyond $d_h = 8$.

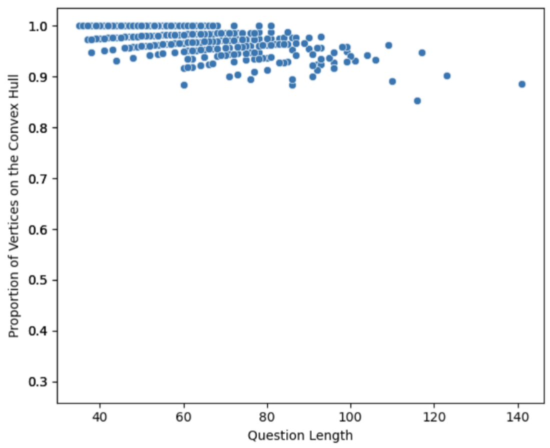
5.1. Convex Hull Observations

We start by analyzing the 64-heads and 128-heads models since we can calculate their convex hulls. For the sake of brevity, we will show visualizations from the same randomly-chosen attention head in the last layer of each transformer. After manually inspecting every head of every layer, the plots and statistics here are very representative. First, we wish to know the proportion of vertices that makes up the convex hull of each model. This will bring insight as to how many tokens may have their attention weights bounded.

Figure 5.1 depicts the proportion of keys which are vertices of the convex hull for each question of the validation set. Our tokenizer encodes the questions such that their lengths



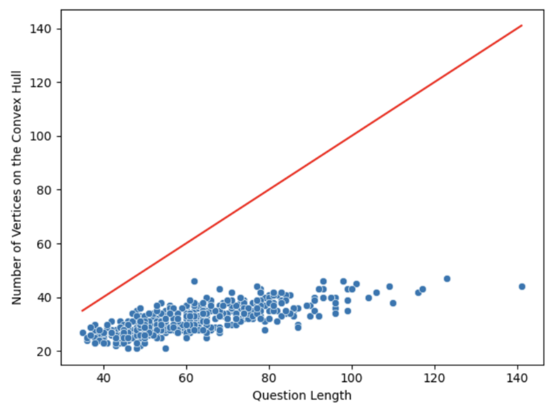
(a) 128-heads ($d_h = 4$)



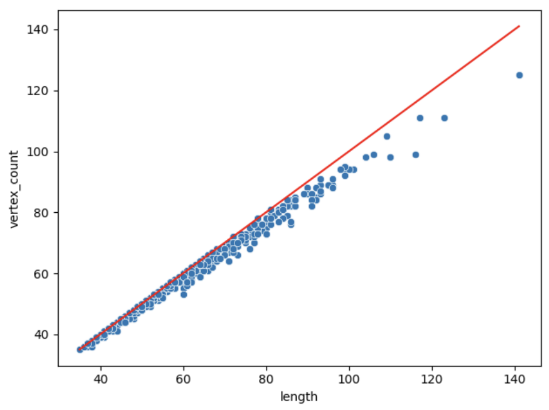
(b) 64-heads ($d_h = 8$)

Figure 5.1. The proportion of key embeddings on the convex hulls of randomly-sampled attention heads. Each data point represents a question from the validation set.

vary from 35 to 141 tokens. The average question length is 62 tokens. We see two facets of the Stolen Attention Effect being demonstrated here. First, with lower d_h, the Stolen Attention Effect is more prominent. Stolen Attention can only take place if embeddings are interior to the convex hull. Looking at the plot, there are numerous questions for which all keys can be placed on the hull of the 64-head model. On the other hand, the 128-head model lacks expressiveness; there are always some number of embeddings interior to its hull. Furthermore, the lowest vertex proportion in the 64-head model is larger than the highest proportion seen in the 128-head model. Second, Stolen Attention worsens as the context length increases. The 128-head model shows a clear downward trend between question length and proportion of vertices, meaning more keys are interior to the hull and have bounded attention. The 64-head model also displays this to an extent, although the vertex proportions follow a strange banded pattern.



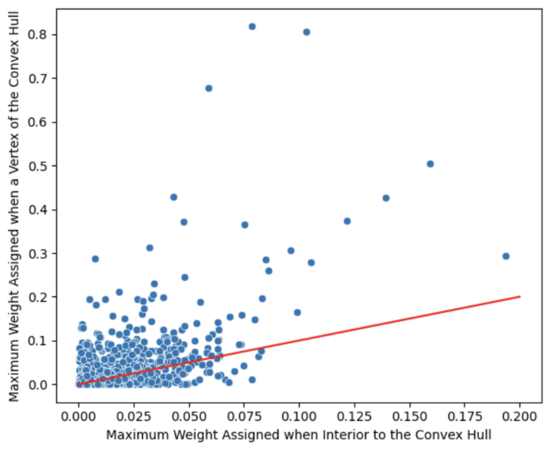
(a) 128-heads ($d_h = 4$)



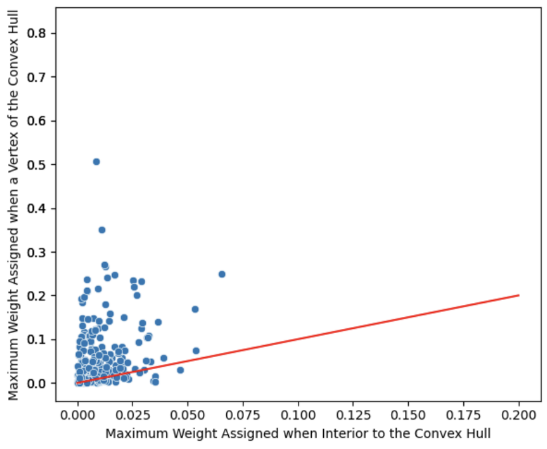
(b) 64-heads ($d_h = 8$)

Figure 5.2. The number of key embeddings on the convex hulls of randomly-sampled attention heads. Each data point represents a question from the validation set. The line $y = x$ (representing a vertex proportion of 1) is plotted for reference.

To further analyze the behavior seen in 5.1, we plot the number of key embeddings inside the same heads from both models. It was hypothesized that as the sequence length expands, a model would reach some maximum number of vertices on its convex hull. After that point, the sequence is too long, and some keys must become interior to the hull because the embedding space has run out of directions of variation. Figure 5.2 does not seem to confirm this hypothesis. Both models display a linearly increasing trend in the number of vertices on the convex hulls of their K matrices. However, the slope of the trend is much greater in the 64-head model; it sticks much closer to the $y = x$ reference line. This illustrates the trend in proportions seen in Figure 5.1. The 128-head model has more vertices as the context length increases, but the number of vertices grows more slowly than the question length. Meanwhile, the 64-head model is very close to accommodating all embeddings as vertices. Perhaps some threshold number of vertices would be realized with context lengths longer than what we tested with OpenBookQA.



(a) 128-heads ($d_h = 4$)



(b) 64-heads ($d_h = 8$)

Figure 5.3. The maximum attention weight assigned to tokens' key vectors when they are a vertex of the convex hull versus when they are interior. Each data point represents a token used at least once in the validation set of OBQA. The line $y = x$ (representing the same maximum weight) is plotted for reference.

The previous plots have confirmed that models with small d_h place some key vectors interior to the convex hull. Those key vectors will have their maximum attention weights bounded by some other key that is a vertex of the hull. To show the extent to which weights are bounded, Figure 5.3 displays the maximum attention weight assigned to tokens when they are a vertex versus when they are interior to the hull. Tokens that are never interior to the hull have been omitted from the plots. Both models display a tendency to award higher weights to the token when it appears as a vertex, shown by the prevalence of points above the $y = x$ reference line. Counterintuitively, the 64-heads model displays a larger bias toward vertices than the 128-heads model. We will give a potential explanation for this in Section 5.3.

5.2. Extrapolating to Larger Models

Without the ability to calculate the convex hulls of models with higher d_h, it is difficult to extend analysis to the 8-heads and 32-heads models. That being said, the main motivation for the occurrence of Stolen Attention is the arrangement of vectors. In Stolen Probability, empirical evidence showed that relative to a reference point, LSTMs arrange embeddings in a wide distribution of norms and a narrow range of angles [**5**]. Then by Equation 2.6, the norms dominate the dot-product comparison. Similarly, if Stolen Attention is occurring inside models' attention heads, then the norms of key vectors should be more widely distributed than the angles between the keys and the query vector.

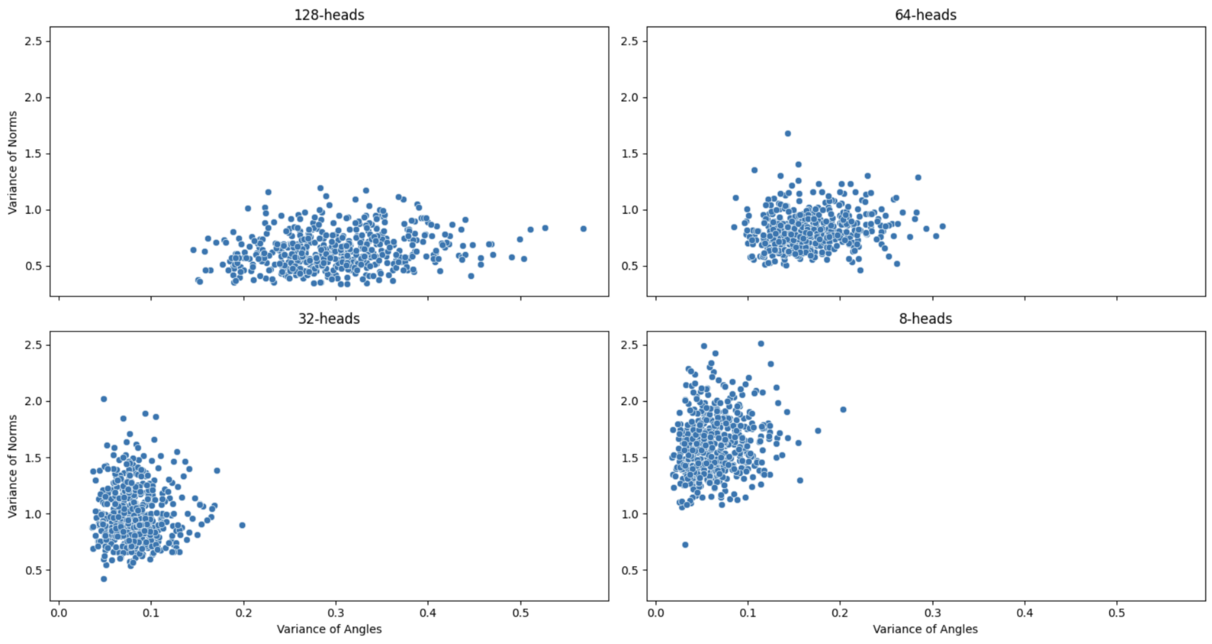


Figure 5.4. The variance of angles between the query embedding and key embeddings vs. the variance of key embedding norms. Each data point represents a question from the OBQA Validation set. Heads were randomly sampled from the last layer of each model.

Figure 5.4 displays the variances of those statistics in a representative attention head from each model. Specifically, for each question in the validation set, we collect the variance of angles between the query vector and key embeddings. We also plot the variance of key embedding norms. Unfortunately, this plot seems to show the reverse behavior of what we'd expect. In the 128-heads model, where Stolen Attention should be most prominent, we see the greatest variations in angles. As we decrease the number of heads, we observe less variance in the angles and more variance in the embedding norms. This shows the importance of calculating the convex hull in the detection of Stolen Attention. By these statistics alone, we might conclude that the 8-heads model is the most susceptible to Stolen Attention. Yet, we know that increasing d_h from 4 to 8 drastically reduced the proportion of keys on the convex hull. Thus, the 8-heads model should be the least susceptible to Stolen Attention.

5.3. Ablating Dot-Product Attention

Ultimately, the Stolen Attention Effect is a consequence of the use of dot-product (Equation 2.3) as the distance metric in a transformer's attention head. When dot-product is used, certain embeddings have their maximum attention weights bounded. Ideally, a transformer's potential to assign weights should be free of constraints. That way, we can pay the most attention to the true keys most relevant to the query. In this section, we wish to replace Equation 2.3 with a different distance metric. If another metric improves the assignment of weights to interior points, it may be a solution to Stolen Attention.

Here, we will use a variant of Euclidean distance to measure similarity. In a normal transformer, we compare embeddings of $Q, K \in \mathbb{R}^{N \times d_h}$ using QK^T. Dot-product similarity is theoretically unbounded; large, positive scores represent similar vectors, while negative scores indicate dissimilarity. Then softmax normalizes the scores into weights such that large positive scores are the highest. When comparing embeddings using Euclidean distance, all distances are positive, with ones closest to zero being the most similar. Thus, we use negative Euclidean distance to maintain the semantics of softmax (most positive implies most related). Furthermore, we use squared Euclidean distance to avoid taking the square root of the distances, which is an expensive operation for GPUs. The code for our implementation can be found in Appendix B. Let $Z \in \mathbb{R}^{N \times N}$ be the output of the distance metric $D(Q, K)$. Formally, we compare a query q_i with keys k_j using:

$$Z_{ij} = -(||q_i - k_j||_2)^2 \tag{5.1}$$

We train a variant of our 64-heads architecture that uses this new metric for its self-attention calculation. Besides the new metric, all details of the training process are identical to the original 64-heads model. Because of our seeding process, the weights of both models were initialized equally, and models receive the same batches of data at every step. Thus, the distance metric is the only source of variation. A comparison of the loss plots is shown in Figures 5.5 and 5.6.

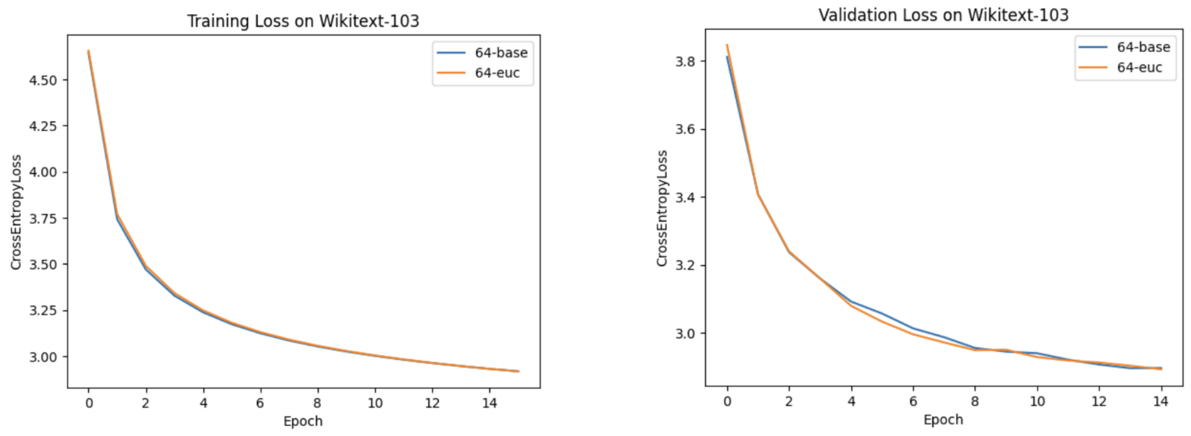


Figure 5.5. Evolution of 64-heads model variants' training on wikitext-103.

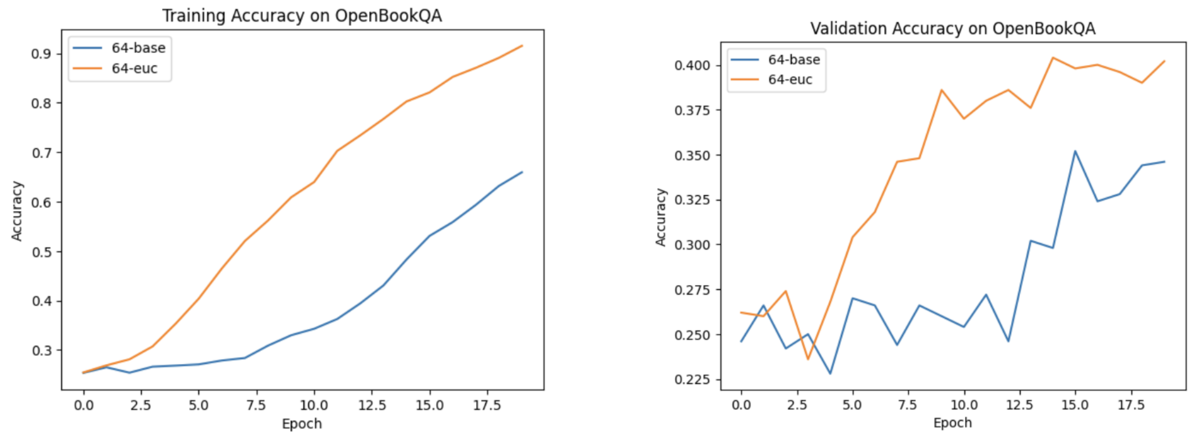
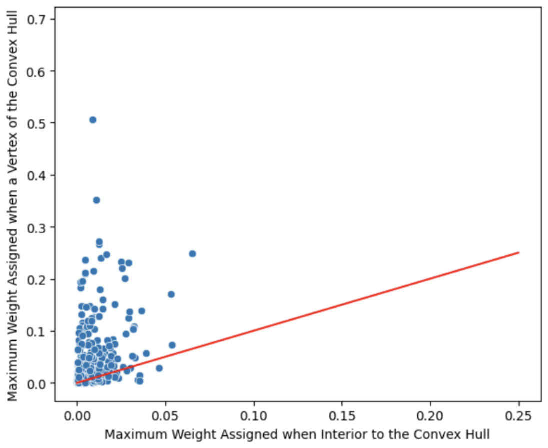
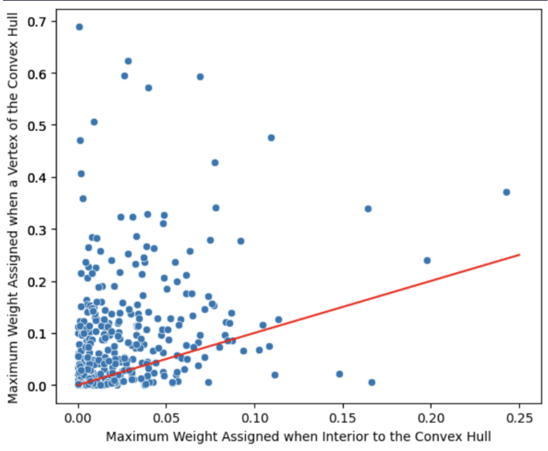


Figure 5.6. Evolution of 64-heads model variants' training on OBQA.

The use of Euclidean distance does not improve the model's pre-training loss (and therefore perplexity). However, during fine-tuning, the new metric has great improvement over standard dot-product. Specifically, it beats the dot-product model by 14.27% on OpenBookQA, achieving a validation accuracy of 40.4%. This serves as further evidence for the importance of comparing models on downstream tasks, as pre-training performance suggested the models were very similar.



(a) Base (Eq. 2.3)



(b) Euclidean (Eq. 5.1)

Figure 5.7. Comparison of the same head from the last layer of the 64-heads variants. Maximum attention weight assigned to tokens' key vectors when they are a vertex of the convex hull is shown against the maximum weight assigned when they are interior. Each data point represents a token. The line $y = x$ is plotted for reference.

Figure 5.7 demonstrates the effect of this intervention on weights assigned to tokens in OpenBookQA. Clearly, the heads of a model utilizing Euclidean distance are capable of assigning higher attention weights to tokens whose keys are interior to the convex hull. Thus, the improvement in performance may be partially attributed to the avoidance of the Stolen Attention Effect. Yet, for the majority of tokens, the discrepancy between maximum weights assigned is still large. Although the Euclidean distance model can place unbounded attention on interior points, it still chooses to pay more attention when they are vertices. This suggests that language models learn to place important embeddings on the convex hull irrespective of the distance metric. We leave this analysis to future work.

CHAPTER 6

Conclusion and Future Work

We present theoretical and qualitative analyses showing that the use of dot-product softmax inside of transformers' attention mechanisms limits their ability to spread attention. Specifically, key vectors who are interior to the convex hull have their attention weights relative to a query embedding bounded. We document the potential for Stolen Attention by showing how many vectors are on the convex hull of the K matrices of attention heads. We also show the discrepancy in maximum attention weight assigned when a token is interior to the hull. Finally, we ablate to replace scaled dot-product attention with negative squared Euclidean attention, which achieves higher performance on OpenBookQA.

This work was limited in part by computational resources. Unfortunately, the Quick-Hull algorithm is computationally intractable for embedding spaces beyond eight dimensions. To truly analyze the effects of Stolen Attention in larger models, we must resort to approximate detection algorithms. We leave this as an item of future work. Yet, the analysis presented here may be evidence that the Stolen Attention Effect does not impact models at scale. Even with spaces as small as $d_h = 8$, a high proportion of embeddings (85% or more, see 5.1b) can be placed on the convex hull. Although a model augmented with Euclidean Distance achieved higher accuracy, it also placed significantly higher weights on tokens when they were vertices. Thus, its performance boost may be simply because Euclidean distance better differentiates vectors at this scale. To test this modification and the source of its power is outside the scope of this research.

Future work should begin with a task that requires larger context lengths. Ultimately, our tokenization of OpenBookQA left much to be desired. With an average question length of 62 tokens, we were only able to saturate the embedding space of our smallest model (128-heads, $d_h = 4$). It's possible we did not pick a hard enough task to exemplify Stolen Attention. It would be interesting to see at which sequence lengths larger embedding sizes begin to struggle. Furthermore, a major component lacking from our analysis is the connection between Stolen Attention and evaluation metrics like perplexity or accuracy. While the Stolen Attention effect may be individually experienced per attention head, our qualitative analysis did not show much variation between attention heads. In fact, the plots presented are even representative of attention heads from different layers. Yet, work from Hao et al. [**7**] shows that certain attention heads are much more impactful than others in the predictions of a transformer. Perhaps there is a connection between the extent to which Stolen Attention is present and the importance of a given head.

References

[1] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. *J. Mach. Learn. Res. 3* (2003), 1137–1155.

[2] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

[3] Chen, J., Wolfe, C., and Kyrillidis, A. Rex: Revisiting budgeted training with an improved schedule, 2021.

[4] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. Palm: Scaling language modeling with pathways, 2022.

[5] Demeter, D., Kimmel, G. J., and Downey, D. Stolen probability: A structural weakness of neural language models. *ArXiv abs/2005.02433* (2020).

[6] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[7] Hao, Y., Dong, L., Wei, F., and Xu, K. Self-attention attribution: Interpreting information interactions inside transformer, 2021.

[8] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9* (1997), 1735–1780.

[9] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization, 2017.

[10] Kudo, T. Subword regularization: Improving neural network translation models with multiple subword candidates, 2018.

[11] Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the variance of the adaptive learning rate and beyond, 2021.

[12] Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. Understanding the difficulty of training transformers, 2023.

[13] Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

[14] Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018.

[15] Mikolov, T., Karafiát, M., Burget, L., ernocký, J. H., and Khudanpur, S. Recurrent neural network based language model. In *Interspeech* (2010).

[16] Pal, A., Karkhanis, D., Roberts, M., Dooley, S., Sundararajan, A., and Naidu, S. Giraffe: Adventures in expanding context lengths in llms, 2023.

[17] Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation, 2022.

[18] Qin, Z., Sun, W., Deng, H., Li, D., Wei, Y., Lv, B., Yan, J., Kong, L., and Zhong, Y. cosformer: Rethinking softmax in attention. *ArXiv abs/2202.08791* (2022).

[19] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners.

[20] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[21] Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023.

[22] Takase, S., Kiyono, S., Kobayashi, S., and Suzuki, J. Spike no more: Stabilizing the pre-training of large language models, 2024.

[23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.

[24] Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. Learning deep transformer models for machine translation, 2019.

[25] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity, 2020.

[26] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the transformer architecture, 2020.

[27] Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences, 2021.

APPENDIX A

OpenBookQA Encoding

Our encoding of the questions followed the template below. All items in brackets are filled in with question-specific details.

```
Fact:[FACT] Question:[STEM] A:[A] B:[B] C:[C] D:[D] Answer:
```

Above, the `[FACT]` is the gold science fact used to create the question `[STEM]`. Then the answer choices, e.g. `[A]`, are filled with the text representing that answer choice. Note that the actual answer choice is not provided in the prompt, it must be generated by the model. Of course, the token representing the correct answer choice is used in the calculation of loss to encourage the model to generate it. To visualize, below shows how a randomly-sampled question from the training set would be encoded. The correct answer choice is B, which corresponds to token 151 of the model's vocabulary.

```
Fact:  hawks eat lizards Question:  In the desert, a hawk may enjoy an
occasional A: coyote B: reptile C: bat D: scorpion Answer:
```

APPENDIX B

Negative Squared Euclidean Distance Calculation

Our codebase utilized an abstract `AttentionMechanism` class, where each module must implement the `get_logits` function that returns the comparison of Q, K matrices over which to apply softmax. Our dot-product models implement Equation 2.3 for this. Below is the implementation of this function for Negative Squared Euclidean Distance. Unfortunately, the extra matrix multiplication and aggregation operations cause it to be slower to compute than dot-product attention. Models trained with this variant took twice the amount of wall-clock time to complete. We believe the majority of this difference could be overcome with a fused kernel that computes the entire metric in a single call.

```
def get_logits(self, q, k):
   # q, k have size [bsz, n_heads, seq_len, d_h]
   Q_sq = torch.sum(torch.square(q), axis=3).unsqueeze(3)
   K_sq = torch.sum(torch.square(k), axis=3).unsqueeze(2)
   QK_dot = torch.matmul(q, k.mT)
   neg_sq_logits = -(Q_sq - 2*QK_dot + K_sq)
   return neg_sq_logits
```