Sparse Transformers are (in)Efficient Learners

Comparing Sparse Feedforward Layers in Small Transformers

Yijun Wu

Supervisor(s): Maliheh Izadi, Arie van Deursen, Aral de Moor

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Yijun Wu
Final project course: CSE3000 Research Project
Thesis committee: dr. Thomas Abeel, dr. Maliheh Izadi, prof. dr. Arie van Deursen, Aral de Moor

Abstract

Although transformers are state-of-the-art models for natural language tasks, obtaining reasonable performance still often requires large transformers which are expensive to train and deploy. Fortunately, there are techniques to increase the size of transformers without extra computing costs. One such technique is sparsity. However, it remains unclear whether sparse architecture is intrinsically more efficient than its dense counterpart. In this paper, we investigate whether replacing the feedforward networks in small transformers with sparse alternatives results in better predictions and faster inference. We found that although inference speed does not increase due to software and hardware limitations, certain sparse alternatives do result in better language understanding. Our research contributes to smarter architectural decision making when designing small language models.

1 Introduction

Transformers are state-of-the-art models for natural language tasks such as next word prediction, text summarisation, and sentiment analysis. They are the backbones of tools including ChatGPT[1] and GitHub Copilot[2]. Although performance tends to scale with model size for transformers, bigger transformers are also slower to train and occupy more space. Fortunately, an array of techniques exists to improve a transformer's efficiency [24]. Sparsity is one such technique. Sparse transformers are transformer models that only activate a portion of their parameters when processing an input. This allows the model to contain more parameters, and thus greater learning capacities, without extra computational costs. Sparse models such as GLaM [5], Switch Transformer [7], and Scaling Transformer [9] have been shown to outperform archetypal dense transformers including GPT-3 [3], T5 [21], and Pegasus [34].

The architectural reason for the sparse transformer's success remains unclear. While the number of learnable parameters certainly plays a role, current research is inconclusive on whether sparsity is intrinsically related to performance, in the sense of a more efficient utilization of parameters. This is indeed plausible as earlier works on network pruning demonstrated that it is possible to achieve better [1] or at least comparable performance [13] to the original model with a sparsified network.

This question calls for controlled and comprehensive experiments. Controlled in terms of the model size, as the sparse model needs to have a comparable size as the dense model to negate the higher capacity gained from the additional parameters. Comprehensive with respect to the implementations of sparsity. There are many ways to sparsify both the attention and the feedforward layer of the transformer. This paper investigates only the feedforward layer to reduce the search space.

[1]https://openai.com/chatgpt/

[2]https://github.com/features/copilot

We restrict our attention to small transformers, that is, transformers with less than 10M parameters and 256 embedding dimensions as defined by Eldan and Li [6]. The choice to focus on small transformers is based on two rationales. Firstly, small transformers can be pretrained and finetuned within reasonable time using commercial hardware. In practice, it means they can be deployed locally to perform domain-specific tasks rather than being hosted on the cloud, thereby circumventing privacy and security issues. Secondly, current research in large language models is approaching a bottleneck as large language models are predicted to require more training data than what is available in the world by 2032 [27]. Thus, small, sample-efficient transformers are becoming a promising research direction. When model size is a limiting factor, architectural decisions are all the more important.

This paper explores a new avenue to improve small transformers via alternate architectures. More specifically, we empirically investigate whether sparsifying the feedforward layers in small transformers, without increasing the number of learnable parameters, improves language understanding and inference speed. Our contribution is as follows:

- We found, through empirical study, that inference speed does not increase with sparse feedforward layers because current machine learning frameworks do not have good support for sparse models and sparsity does not receive as many benefits on a single device compared to distributed training in the small language model setting.
- We found that sparse feedforward layers under the right configurations are better at language understanding than dense feedforward networks.
- We contribute to a unified understanding of sparse feedforward layers by deriving a mixture of experts interpretation of the controller feedforward layer.
- A replication package[3] for reproducing our findings, and our models[4] published on HuggingFace.

2 Background

This section elaborates on the transformer model and discusses related works on efficient transformers.

2.1 Transformer

Transformers [26] are composed of blocks of attention layers and feedforward layers. The attention layer represents a token, for instance, a word, a subword, or a character, in terms of its relation with other tokens in the input sequence. The feedforward layer applies a non-linear transformation to the features learned by the previous attention layer. The original transformer [26] consists of an encoder and a decoder. For missing token prediction tasks, which are the focus of this paper, encoder or decoder alone suffices. This paper investigates two baseline architectures, GPT-2 [19] and BERT [4].

The transformer output's interpretation depends on the pretraining objective. GPT-2, when pretrained under a causal

[3]https://github.com/AISE-TUDelft/tiny-transformers

[4]https://huggingface.co/collections/AISE-TUDelft/brp-tiny-transformers-666c352b3b570f44d7d2a519

language modelling objective, outputs a sequence of logit arrays, one for each input token. Each logit array represents the predicted probability distribution of the next token given the previous tokens. For BERT, two objectives were originally proposed: masked language modelling and next sentence prediction. This paper is only concerned with the former for the ease of assessing the model's language understanding. This means the model is given an input sequence containing masked tokens and is pretrained to uncover those masked tokens.

2.2 Related Works

There exist various paradigms for efficient transformers [24]. The majority focuses on reducing the attention layer's quadratic space and time complexities. However, since feedforward layers make up two-thirds of the total parameters in a transformer [18], this paper focuses on efficient feedforward techniques, of which there are two main approaches: lowrankness and sparsity.

Low-rank techniques factorize the weight matrix $W_{m\times n}$ in the feedforward layer as a product of lower-rank matrices $U_{m\times l} \times V_{l\times n}$. This leads to computational speedups if $l < \frac{mn}{m+n}$ [11]. In the context of transformers, [33] demonstrated that with the same number of model parameters, low-rank feedforward layers perform favourably to their full-rank counterparts.

Sparse techniques mask the weight matrices so only a fraction of their entries are used in computations. Unlike low-rank techniques, it is currently uncertain if sparse models can outperform dense models of the same size.

3 Approach

This section describes the feedforward layer along with its main sparse variants and derives their sizes. This is followed by new insights into a unified view of sparse feedforward layers.

3.1 Feedforward Network (FFN)

In baseline transformer architectures, the feedforward layer is simply a feedforward network (FFN), defined as:

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2 \tag{1}$$

where σ is the activation function, typically ReLU.

Suppose the dimensionality of the input token embedding is d_{in} and that the intermediate size is $d_m = Md_{\text{in}}$, then the total number of parameters is:

$$T_{\text{FFN}} = 2Md_{\text{in}}^2 + (M+1)d_{\text{in}} \tag{2}$$

Note that because all models share the same tokenizer, d_{in} is fixed.

Sparsely Gated Mixture of Experts (MoE)

Sparely gated mixture of experts (MoE), originally proposed in [23] for recurrent models was adapted into transformers such as GLaM [5], GShard [14], Switch Transformer [7], and Mixtral of Experts [10]. MoE consists of an ensemble of small feedforward networks called experts $\text{FFN}_1, \text{FFN}_2, ..., \text{FFN}_n$ and a routing function G, such that the output is a linear combination of expert outputs:

$$\text{MoE}(x) = \sum_{i=1}^{n} G(x)_i \text{FFN}_i(x) \tag{3}$$

Sparsity is created when $G(x)$ is zero for all but the top k experts (typically $k = 2$). In [23], $G(x)$ is defined as:

$$G(x) = \text{Softmax}(\text{Top}_k(xW_g + Z \odot \text{Softplus}(xW_\epsilon))) \tag{4}$$

where Z is a standard normal vector.

Assuming an expert ensemble of size N_{experts}, where each expert has intermediate size M, then the total number of parameters across the expert ensemble is $N_{\text{experts}}d_{\text{in}}(2Md_{\text{in}} + M + 1)$. In addition, the routing function requires $2N_{\text{experts}}d_{\text{in}}$ parameters. The total number of parameters in MoE is thus:

$$T_{\text{MoE}} = N_{\text{experts}}d_{\text{in}}(2Md_{\text{in}} + M + 3) \tag{5}$$

The number of active parameters for top k expert activation is:

$$T_{\text{MoE}}^{\text{active}} = k(2d_m d_{\text{in}} + d_m + d_{\text{in}}) + 2N_{\text{experts}}d_{\text{in}} \tag{6}$$

Controller Feedforward (CNT)

The controller feedforward layer (CNT) implements sparsity by applying a learned mask (called the controller) on the weight matrix [9, 8, 25]. Using the Scaling Transformer [9] as a representative, this feedforward layer can be defined as:

$$\text{CNT}(x) = \sigma(xW_1 + b_1) \odot \text{Controller}(x)W_2 + b_2 \tag{7}$$

In this case, the controller masks out all but k entries of the activation vector, effectively activating only k rows of W_2.

In the sparse feedforward model from [9], the controller is implemented as a linear map followed by argmax activations over N_{blocks} blocks. The linear map is factorized into the product of two rank $\frac{d_{\text{in}}}{N_{\text{blocks}}}$ matrices. The total number of CNT parameters is:

$$T_{\text{CNT}} = 2(M + \frac{M+1}{2N_{\text{blocks}}})d_{\text{in}}^2 \tag{8}$$

The number of active parameters is:

$$T_{\text{CNT}}^{\text{active}} = 2N_{\text{blocks}}d_{\text{in}} + N_{\text{blocks}} + d_{\text{in}} + \frac{d_{\text{in}}^2}{N_{\text{blocks}}} + \frac{d_m d_{\text{in}}}{N_{\text{blocks}}} \tag{9}$$

Product Key Memory (PKM)

The product key memory layer (PKM) [12, 30, 20] is an alternative to the vanilla feedforward network. The weight matrices are replaced by learned query matrix Q, key matrix K, and value matrix V. Under this architecture, the input is first converted to the query vector:

$$q(x) = xQ \tag{10}$$

Only the values corresponding to best matching keys under dot product similarity are activated. The output is given by:

$$\text{PKM}(x) = \text{Softmax}(\text{Top}_k(q(x)K^T))V \tag{11}$$

Each row in K is a key corresponding to a row in the value matrix V. The similarity between the query $q(x)$ and a key

is measured by their dot product. The output is a linear combination of the top k values weighed by the softmax of the query-key similarity.

In product key memory, the query matrix has shape $d_{\text{in}} \times d_q$. The key table K is a Cartesian product of two subkey tables, each of shape $N_{\text{subkeys}} \times d_{\text{subkey}}$, where $2d_{\text{subkey}} = d_q$. The value table, therefore, has shape $N_{\text{subkeys}}^2 \times d_{\text{in}}$. Instead of having a single query matrix and a pair of subkey tables, [12] proposes a multi-head approach, with N_{heads} query matrices and pairs of subkey tables, but a shared value table. Under this approach, the total number of parameters is:

$$T_{\text{PKM}} = 2d_{\text{in}}d_{\text{subkey}}(N_{\text{heads}} + \frac{N_{\text{subkeys}}^2}{2d_{\text{subkey}}} + \frac{N_{\text{heads}}N_{\text{subkeys}}}{d_{\text{in}}}) \quad (12)$$

With multiple heads, the number of active parameters depends on the input (as multiple heads can select the same value), but the lower bound given top k key activation is:

$$T_{\text{PKM}}^{\text{active}} \geq N_{\text{heads}}(d_q d_{\text{in}} + N_{\text{subkeys}} d_q) + k d_{\text{in}} \quad (13)$$

3.2 A Unified View of Sparse Feedforward Layers

We demonstrate that CNT is the same as MoE with best expert activation and weight sharing. This transitively means that FFN, MoE, CNT, and PKM can all be viewed as neural memories. A unified view of sparse feedforward layers simplifies theoretical analysis and allows a generalized explainability technique to be developed for all sparse feedforward layers.

Previous works proved that FFN, MoE, and PKM can be interpreted as neural memories [16]. A neural memory with n memory cells is a function of the form:

$$y = \sum_{i=0}^{n-1} \alpha_i(x) v_i \quad (14)$$

The output of neural memory is a linear combination of value vectors $v_0, ..., v_{n-1}$ weighed by the input-dependent memory coefficients $\alpha_0, ..., \alpha_{n-1}$.

PKM can be trivially expressed in this form. The value vectors are the rows of the value matrix V and the memory coefficients are the dot product between the query vector and the keys.

As for MoE, assume it has routing function G and N_e experts, each defined as a FFN with intermediate size d_m and no biases, $\text{FFN}_i(x) = \sigma(xU_i)V_i$. It can be expressed in the form of neural memory with $N_e d_m$ memory cells as:

$$\begin{aligned} \text{MoE}(x) &= \sum_{l=0}^{N_e d_m - 1} \alpha_l(x) V_{\lfloor \frac{l}{d_m} \rfloor}[l \bmod d_m, :] \\ \alpha_l(x) &= G(x)_{\lfloor \frac{l}{d_m} \rfloor} \sigma(xU_{\lfloor \frac{l}{d_m} \rfloor})[l \bmod d_m] \end{aligned} \quad (15)$$

The mixture of experts interpretation of CNT is based on Equation (7). Assuming the controller is in N_{blocks} blocks and has intermediate size d_m, an equivalent MoE can be constructed. This MoE has $\binom{d_m}{N_{\text{blocks}}}$ experts, each with intermediate size N_{blocks}. The controller can be seen as a routing function that only selects the top expert. The experts are the combinations of every N_{blocks} row in the weight matrices. To be precise, every size N_{blocks} subset, s, of $\{0, ..., d_m - 1\}$ defines an expert:

$$\text{FFN}_s(x) = \sigma(xW[:, s] + b_1[s])W_2[s, :] + b_2 \quad (16)$$

When viewed as a neural memory, the CNT has $\binom{d_m}{N_{\text{blocks}}} N_{\text{blocks}}$ memory cells. However, due to weight sharing, there are at most d_m unique value vectors.

4 Experimental Setup

This section reports the research questions answered by the experiments, the dataset used in pretraining, the evaluation metrics, and the hyperparameters[5].

4.1 Research Questions

The experiments address the following questions:

1. *Can sparse feedforward layers offer better language understanding than a dense feedforward layer of the same size?* We compare the performance of vanilla GPT-Neo and RoBERTa against their sparse variants augmented with MoE, CNT, and PKM as feedforward layers while keeping the number of parameters as close as possible.
2. *Is there a direct relationship between the degree of sparsity in the feedforward layer and the model performance?* For each sparse variant, we evaluate several configurations corresponding to different levels of sparsity.
3. *How does the type of sparse feedforward layer affect inference speed?* We measure the forward pass time for MoE, CNT, and PKM.

4.2 TinyStories

All models investigated in this paper are pretrained on the TinyStories dataset. This is a collection of short stories generated by GPT-3.5 and GPT-4 tailored to match the cognitive level of a 3-4 year-old [6]. It is a suitable dataset for pretraining small generative language models because it covers a wide range of grammatically correct stories without forcing the model to learn a vast amount of domain knowledge.

4.3 BabyLM Pipeline

A model's language understanding is measured in this paper by the performance metrics from BabyLM[6], namely BLiMP and (Super)GLUE. BablyLM is a challenge on sample-efficient pretraining of small language models [31]. We used a modified implementation of the BabyLM evaluation pipeline to accommodate the custom feedforward layers.

BLiMP originally consisted of 67000 pairs of sentences, where the sentences in each pair are minimally different, but one contains one of twelve types of grammar error [32]. For BabyLM, this dataset set was extended with supplement tasks, covering a wider range of grammar errors. Performance is measured as classification accuracy, where a model

[5]Our implementation can be found at: github.com/AISE-TUDelft/tiny-transformers

[6]The official evaluation pipeline can be found at: https://github.com/babylm/evaluation-pipeline-2023

classifies a pair correctly if it assigns the grammatically correct sentence a higher likelihood.

GLUE is a collection of causal reasoning, question answering, word sense disambiguation, and coreference resolution tasks [28]. It is designed to test a model's general linguistic knowledge and offers a difficult human baseline for language models to compete against. Like GLUE, SuperGLUE is also designed to assess a model's language understanding, albeit through harder tasks [29]. BabyLM uses a mixture of GLUE and SuperGLUE tasks.

4.4 Hyperparameters

The baseline for all experiments are GPT-Neo [2], which is an implementation of GPT-2, and RoBERTa [15], which is an implementation of BERT. The number of transformer blocks is fixed to 2. Token embedding dimensionality is set to $d_{\text{in}} = 256$ and the context length is kept at 512. For the baseline feedforward network, the intermediate size is $d_m = 4096$. Sparse transformers are constructed by replacing all the feedforward layers with their sparse variants.

For MoE, the number of experts is limited to $N_{\text{experts}} = 4$ and each expert has intermediate size $d_m = 1023$. We experiment with activating top $k = 1, 2, 3$ experts.

For CNT, we experiment with block counts $N_{\text{blocks}} = 64, 32, 16$, and associated intermediate size $d_m = 4032, 4032, 3968$.

As for PKM, all models have $N_{\text{head}} = 4$ heads, $N_{\text{subkeys}} = 56$ subkeys, and query dimensionality $d_q = 1024$. Models with top $k = 14, 28, 42$ (approximately top $25\%, 50\%, 75\%$) key activation are investigated. Table 1 summarizes the model size.

We pretrain all models for 2 epochs using the AdamW optimizer with an initial learning rate of 0.0005 and linear learning rate decay. The batch size and the gradient accumulation steps are set to 16. All pretraining is accelerated by an NVIDIA A100 GPU.

To measure the average inference time of each type of feedforward layer, we measure the inference time on a batch of 16 input sequences, each of length 256 and embedding dimensionality of 256. This is measured across 1000 batches. Prior to the experiment, the GPU is warmed up for 100 batches.

5 Results

5.1 Pretraining

The evaluation loss curves during pretraining are displayed in Figure 1. This figure shows that although no sparse models achieved lower evaluation loss than their respective baselines, at least one configuration from every sparse RoBERTa-based-models reached a similar loss as the baseline after one epoch, despite starting at a higher initial loss. Even among the GPT-Neo-based models, the MoE variant achieved a similar evaluation loss as the baseline.

5.2 Evaluation

Table 2 demonstrates that all types of sparse feedforward layers can offer better performance for GPT-Neo and RoBERTa than FFN on (Super)GLUE tasks. As for BLiMP tasks, the PKM variant is unable to outperform the baseline for RoBERTa. The same applies to the MoE and PKM variants for GPT-Neo. On the other hand, the MoE variant performs consistently best on (Super)GLUE, while the CNT variant is the best on BLiMP. Detailed task scores for BLiMP and (Super)GLUE are reported by Table 5 and Table 4 in Appendix A.

Table 1: Models involved in our experiments and their number of learnable parameters and the lower bound sparsity ratio (ratio between the number of active and total parameters) in their feedforward layers. While these models have various sparsity ratios, their total parameter count is approximately the same.

Model	Parameter Count	Sparsity Ratio
GPT-Neo	7421440	1.00
GPT-Neo MoE ($k = 1$)	7422968	0.25
GPT-Neo MoE ($k = 2$)	7422968	0.50
GPT-Neo MoE ($k = 3$)	7422968	0.75
GPT-Neo CNT ($N_{\text{blocks}} = 16$)	7425280	0.04
GPT-Neo CNT ($N_{\text{blocks}} = 32$)	7424384	0.02
GPT-Neo CNT ($N_{\text{blocks}} = 64$)	7390080	0.02
GPT-Neo PKM ($k = 14$)	7396352	0.61
GPT-Neo PKM ($k = 28$)	7396352	0.62
GPT-Neo PKM ($k = 42$)	7396352	0.62
RoBERTa	7500048	1.00
RoBERTa MoE ($k = 1$)	7501576	0.25
RoBERTa MoE ($k = 2$)	7501576	0.50
RoBERTa MoE ($k = 3$)	7501576	0.75
RoBERTa CNT ($N_{\text{blocks}} = 16$)	7503888	0.04
RoBERTa CNT ($N_{\text{blocks}} = 32$)	7502992	0.02
RoBERTa CNT ($N_{\text{blocks}} = 64$)	7468688	0.02
RoBERTa PKM ($k = 14$)	7474960	0.61
RoBERTa PKM ($k = 28$)	7474960	0.62
RoBERTa PKM ($k = 42$)	7474960	0.62

Figure 2 and Figure 3 illustrate the relationship between the proportion of activate parameters (sparsity ratio) in the feedforward layer and the model performance on (Super)GLUE and BLiMP tasks respectively. The trend lines suggest that performance tends to increase with the sparsity ratio. However, this is not always the case, as the trend is reversed for RoBERTa-based models on (Super)GLUE tasks. Furthermore, the positive correlation between the sparsity ratio and performance is only moderate as indicated by the Pearson coefficients ($0.3 < r < 0.5$).

5.3 Inference Speed

Table 3 reports the average inference time per batch for each feedforward layer investigated. FFN is by far the fastest. This is followed by MoE, which is around 20% slower. In comparison, CNT is more than twice as slow as MoE. PKM is much slower than all other kinds of feedforward layers.

6 Discussion

6.1 Performance of Sparse Feedforward Layers

Whether sparse feedforward layers can achieve comparable performance as the vanilla FFN depends on the baseline architecture and type of sparse feedforward layer as Figures 2 and 3 demonstrate. Regardless, sparse models seem to be fast learners since the loss difference between the sparse models and the baseline is much smaller by the end of pretraining than at the start. This is likely because sparse feedforward layers approximate a larger network than an FFN of the same size. Sparse feedforward layers are more flexible learners. In terms of neural memories, although MoE, CNT, and PKM

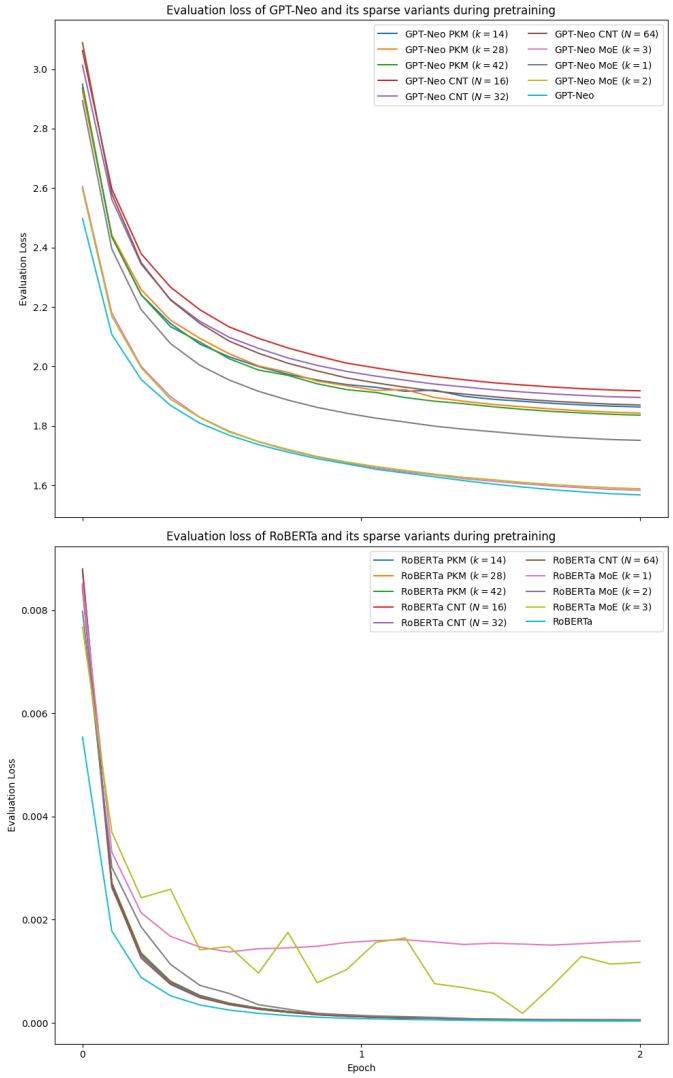

Figure 1: Evaluation loss for GPT-Neo (top), RoBERTa (bottom), and their sparse variants during pretraining. Many sparse models reached a similar loss as the baseline by the end.

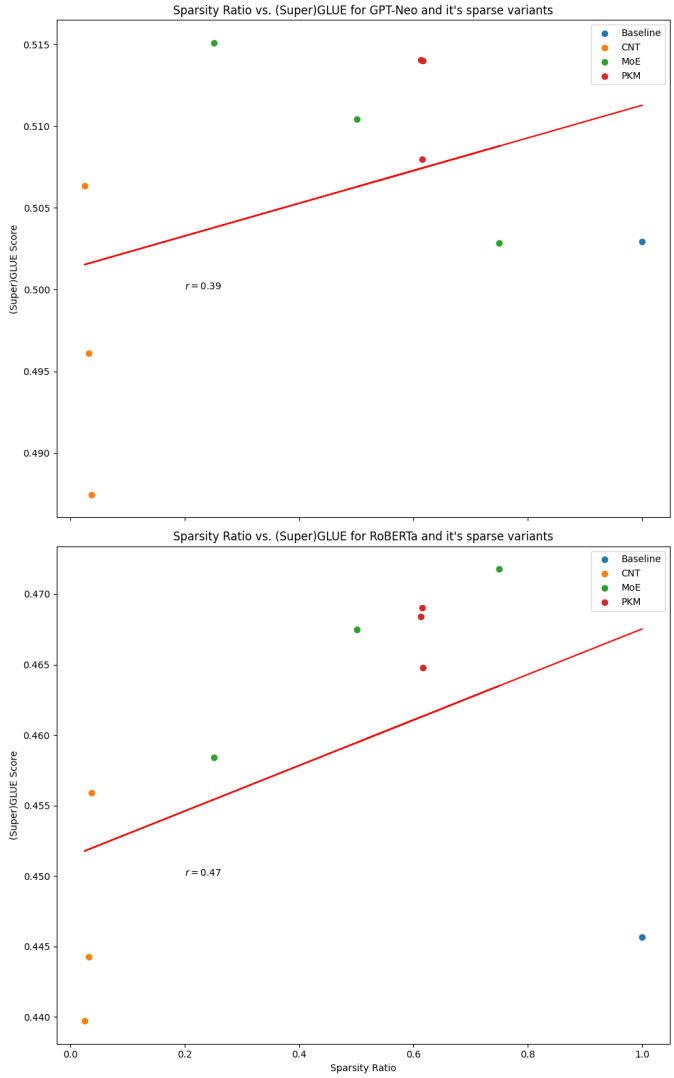

Figure 2: Sparsity ratio (percentage of active parameters in the feedforward layer) of the feedforward layers for GPT-Neo (top), RoBERTa (bottom) and their sparse variants vs. their respective overall (Super)GLUE scores. There is no strong correlation between the sparsity ratio and the (Super)GLUE score.

do not have as many unique value vectors as FFN, their memory coefficients are computed from more complex, non-linear functions.

6.2 Role of Sparsity Ratio

There is inconclusive evidence on the number of active parameters directly affecting performance. Even when considering a sparse variant such as MoE in isolation, it is not guaranteed for performance to decrease or increase with the sparsity ratio. A plausible explanation is that the feedforward architecture is far more important to language understanding than the proportion of active parameters. In other words, how parameters are activated is far more crucial than the amount of active parameters. This hypothesis is corroborated by the CNT results. CNT outperformed the baselines on both BLiMP and (Super)GLUE tasks despite having less than 5% parameter activation in the feedforward layer. In contrast, PKM cannot compete with the BLiMP baselines even when more than 60% of its parameters are active because it is not as architecturally efficient as CNT.

6.3 Inference Cost of Sparse Models

Despite the theoretical speedups sparsity and conditional computation bring, sparse feedforward layers are slowed down by software limitations and the small transformer setting. All models are implemented with PyTorch [17], which, at time the of writing, lacks support for various essential features on sparse tensors, such as reshapes and automatic differentiation, resulting in inefficient implementations. Furthermore, the speedups of conditional computation often manifest in a distributed setting, as is the case in [7], where an MoE model is scaled up by distributing the experts across multi-

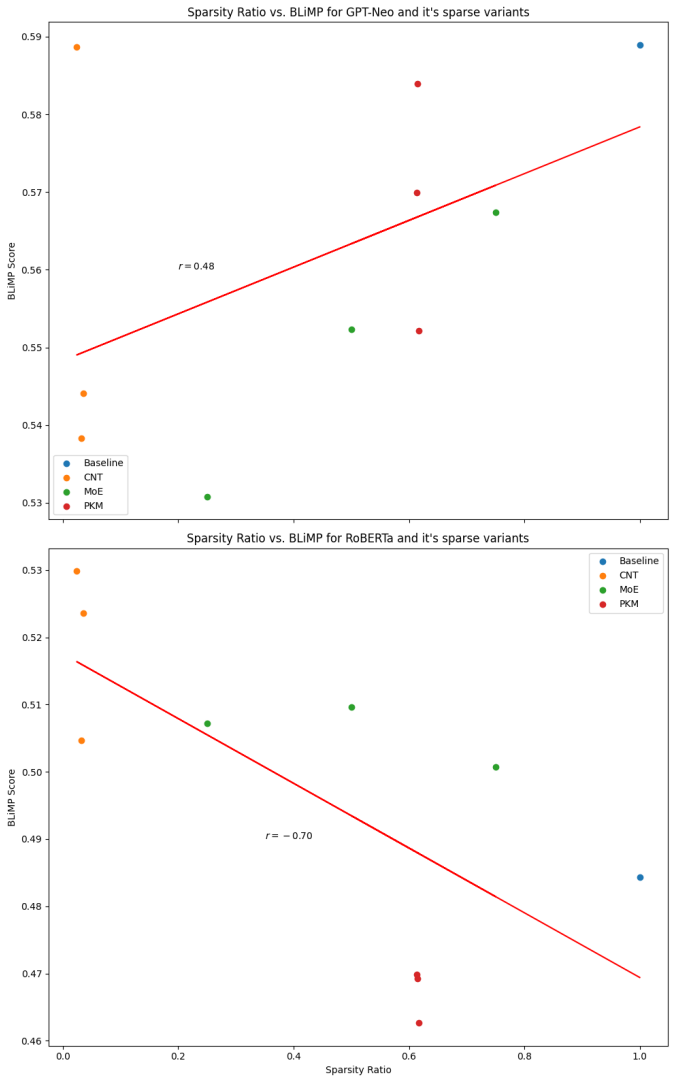

Figure 3: Sparsity ratio (percentage of active parameters in the feedforward layer) of the feedforward layers for GPT-Neo (top), RoBERTa (bottom) and their sparse variants vs. their respective overall BLiMP scores. There is no strong correlation between the sparsity ratio and the BLiMP score.

ple devices. Small transformers do not enjoy this advantage because all training happens on one GPU.

6.4 Threats to Validity

Threats to the validity of this work can be categorized into three categories: threats to internal validity, which are the confounders of the experiment, threats to external validity, which are the issues surrounding the generalizability of the experiment, and threats to construct validity, which concerns the soundness of the research questions.

Internal Validity

While the sizes of the sparse feedforward layers are set to be as close as possible to their dense counterparts in the experiments, it is not possible for them to be identical. Nonetheless, the relative difference is insignificant so it is unlikely to result in a huge change in learning capacity. Other hyperparameters may have a significant influence on the experiment result. For example, we hypothesize based on Figure 1 that all sparse models will converge to the same evaluation loss as that of the baselines given more epochs. However, this would require more computing budget than what is available for this research.

Table 2: BLiMP and (Super)GLUE scores for GPT-Neo, RoBERTa, and their sparse variants. The CNT variant is the best on BLiMP while the MoE variant is the best on (Super)GLUE.

Name	BLiMP	(Super)GLUE
GPT-Neo	0.589	0.503
GPT-Neo CNT ($N = 16$)	0.544	0.487
GPT-Neo CNT ($N = 64$)	0.538	0.496
GPT-Neo CNT ($N = 32$)	**0.589**	0.506
GPT-Neo MoE ($k = 1$)	0.531	**0.515**
GPT-Neo MoE ($k = 3$)	0.567	0.503
GPT-Neo MoE ($k = 2$)	0.552	0.510
GPT-Neo PKM ($k = 14$)	0.570	0.514
GPT-Neo PKM ($k = 42$)	0.552	0.514
GPT-Neo PKM ($k = 28$)	0.584	0.508
RoBERTa	0.484	0.446
RoBERTa CNT ($N = 16$)	0.524	0.456
RoBERTa CNT ($N = 64$)	0.505	0.444
RoBERTa CNT ($N = 32$)	**0.530**	0.440
RoBERTa MoE ($k = 1$)	0.507	0.458
RoBERTa MoE ($k = 3$)	0.501	**0.472**
RoBERTa MoE ($k = 2$)	0.510	0.467
RoBERTa PKM ($k = 14$)	0.470	0.468
RoBERTa PKM ($k = 42$)	0.463	0.465
RoBERTa PKM ($k = 28$)	0.469	0.469

Table 3: Batch inference speed of vanilla and sparse feedforward networks measured in milliseconds. FFN is faster than the sparse feedforward layers.

Model	Mean (ms)	Standard Deviation (ms)
FFN	**12.83**	0.10
MoE ($k = 1$)	16.14	0.07
MoE ($k = 2$)	16.24	0.08
MoE ($k = 3$)	16.45	0.17
CNT ($N = 16$)	34.42	0.09
CNT ($N = 32$)	33.92	0.09
CNT ($N = 64$)	34.06	0.25
PKM ($k = 14$)	43.79	1.30
PKM ($k = 28$)	77.35	3.00
PKM ($k = 42$)	117.48	0.04

External Validity

It must be emphasized that the model configurations investigated in the experiments are by no means exhaustive. As showcased by Table 1, despite considering different configurations for each type of sparse feedforward layer, the sparsity ratio is in some cases almost identical. This is because for some models, such as PKM, a noticeable change in sparsity ratio requires a huge increase in parameter count.

All experiments are conducted on a single GPU. While this is unlikely to affect pretraining and evaluation, changing the hardware setup will alter the inference speed of sparse transformers, especially if the models are distributed over multiple devices.

Construct Validity
There is no universal measure of sparsity. In this research, our measure of sparsity differs per model. For MoE, the degree of sparsity is defined as the number of experts selected. For, CNT, the degree of sparsity is measured in the number of blocks the activation vector is split into. For PKM, the degree of sparsity is defined as the number of keys selected. These definitions do not always correspond to the sparsity ratio well. Although the sparsity ratio appears to be a universal definition of sparsity, it too has shortcomings. As an example, PKM's sparsity ratio is challenging to measure as it depends on the input. Even under the neural memory framework, the number of active memory cells is not guaranteed to be static, so it too is a difficult sparsity measure to implement.

6.5 Future Work

Table 2 shows that CNT is an efficient architecture. It outperforms the baselines on both BLiMP and (Super)GLUE while using the lowest percentage of parameters. This paper investigated CNT under a very low sparsity ratio. For future research, we recommend studying the performance of CNT with more active parameters. For other models, we recommend reducing the intermediate size, and instead increasing the number of experts for MoE or keys for PKM. We conjecture that a very small intermediate size does not necessarily lead to worse performance, as shown by CNT under the mixture of experts interpretation. Lastly, we recommend adding more layers and controllers to CNT to mimic hierarchical MoE. Weight sharing the MoE and PKM is also a good direction for further research.

7 Conclusions

This paper investigates whether sparse feedforward layers in small transformers result in better language understanding and faster inference than the standard feedforward network. We found that inference speed does not increase due to software and hardware limitations associated with our small language model setting. Some sparse feedforward layers, such as the controller feedforward layer, in the right configurations, achieves better language understanding than the standard feedforward network while activating just a fraction of their parameters. We hypothesize that this is because the sparse feedforward layers more efficiently utilize their parameters by approximating larger networks. We recommend future research exploring more configurations of different sparse feedforward architectures.

8 Responsible Research

Deep learning, among other fields, faces a reproducibility crisis [22]. We combated this issue in our research by providing a replication package containing all the source code and instructions to reproduce the results presented in this paper. Furthermore, we reported our hyperparameters and hardware in Section 4. All our models were pretrained and evaluated with fixed seeds. These techniques together ensure that our experiments and findings can be replicated exactly.

In accordance with the Netherlands Code of Conduct for Research Integrity, we reported all our experiment results in this paper under the principles of honesty and transparency. In addition, following the standards for good research practices (chapter 3), we refrained from data fabrication and manipulation. For example, our pretraining and evaluation sets are separate and all models were given to the same data.

Our work raises ethical concerns as it is related to language models. Language models may be trained on copyrighted or private texts. In addition, when trained with domain knowledge, language models can be misused by malicious actors. To mitigate these issues, all our models were pretrained on a synthetic dataset. The dataset contains no knowledge of any specific domain.

References

[1] Luis Balderas, Miguel Lastra, and José M. Benítez. Optimizing dense feed-forward neural networks. *Neural Networks*, 171:229–241, March 2024. ISSN 0893-6080. doi: 10.1016/j.neunet.2023.12.015. URL https://www.sciencedirect.com/science/article/pii/S0893608023007219.

[2] Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL https://zenodo.org/record/5297715.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020. URL https://arxiv.org/abs/2005.14165. Version Number: 4.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv:1810.04805 [cs].

[5] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts, August 2022. URL http://arxiv.org/abs/2112.06905. arXiv:2112.06905 [cs].

[6] Ronen Eldan and Yuanzhi Li. TinyStories: How Small Can Language Models Be and Still Speak Coherent English?, May 2023. URL http://arxiv.org/abs/2305.07759. arXiv:2305.07759 [cs].

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity, June 2022. URL http://arxiv.org/abs/2101.03961. arXiv:2101.03961 [cs].

[8] Matt Gorbett, Hossein Shirazi, and Indrakshi Ray. Sparse Binary Transformers for Multivariate Time Series Modeling, August 2023. URL http://arxiv.org/abs/2308.04637. arXiv:2308.04637 [cs].

[9] Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Łukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is Enough in Scaling Transformers, November 2021. URL http://arxiv.org/abs/2111.12763. arXiv:2111.12763 [cs].

[10] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of Experts, January 2024. URL http://arxiv.org/abs/2401.04088. arXiv:2401.04088 [cs].

[11] Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N. Gomez. Exploring Low Rank Training of Deep Neural Networks, September 2022. URL http://arxiv.org/abs/2209.13569. arXiv:2209.13569 [cs, stat].

[12] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large Memory Layers with Product Keys, December 2019. URL http://arxiv.org/abs/1907.05242. arXiv:1907.05242 [cs].

[13] Yann LeCun, John Denker, and Sara Solla. Optimal Brain Damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[14] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding, June 2020. URL http://arxiv.org/abs/2006.16668. arXiv:2006.16668 [cs, stat].

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, July 2019. URL http://arxiv.org/abs/1907.11692. arXiv:1907.11692 [cs].

[16] Zeyu Leo Liu, Tim Dettmers, Xi Victoria Lin, Veselin Stoyanov, and Xian Li. Towards A Unified View of Sparse Feed-Forward Network in Pretraining Large Language Model, October 2023. URL http://arxiv.org/abs/2305.13999. arXiv:2305.13999 [cs].

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019. URL http://arxiv.org/abs/1912.01703. arXiv:1912.01703 [cs, stat].

[18] Telmo Pessoa Pires, António V. Lopes, Yannick Assogba, and Hendra Setiawan. One Wide Feedforward is All You Need, October 2023. URL http://arxiv.org/abs/2309.01826. arXiv:2309.01826 [cs].

[19] Alec Radford, Jeff Wu, R. Child, D. Luan, Dario Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners. 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

[20] Jack W. Rae, Jonathan J. Hunt, Tim Harley, Ivo Danihelka, Andrew Senior, Greg Wayne, Alex Graves, and Timothy P. Lillicrap. Scaling Memory-Augmented Neural Networks with Sparse Reads and Writes, October 2016. URL http://arxiv.org/abs/1610.09027. arXiv:1610.09027 [cs].

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, September 2023. URL http://arxiv.org/abs/1910.10683. arXiv:1910.10683 [cs, stat].

[22] Harald Semmelrock, Simone Kopeinik, Dieter Theiler, Tony Ross-Hellauer, and Dominik Kowald. Reproducibility in Machine Learning-Driven Research, July 2023. URL http://arxiv.org/abs/2307.10320. arXiv:2307.10320 [cs, stat].

[23] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer, January 2017. URL http://arxiv.org/abs/1701.06538. arXiv:1701.06538 [cs, stat].

[24] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey, March 2022. URL http://arxiv.org/abs/2009.06732. arXiv:2009.06732 [cs].

[25] Vithursan Thangarasa, Mahmoud Salem, Shreyas Saxena, Kevin Leong, Joel Hestness, and Sean Lie. MediSwift: Efficient Sparse Pre-trained Biomedical Language Models. 2024. doi: 10.48550/ARXIV.2403.00952. URL https://arxiv.org/abs/2403.00952. Publisher: arXiv Version Number: 1.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz

Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. URL http://arxiv.org/abs/1706.03762v5. arXiv:1706.03762 [cs].

[27] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? Limits of LLM scaling based on human-generated data, June 2024. URL http://arxiv.org/abs/2211.04325. arXiv:2211.04325 [cs].

[28] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding, February 2019. URL http://arxiv.org/abs/1804.07461.

[29] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems, February 2020. URL http://arxiv.org/abs/1905.00537.

[30] Hai Wang and David McAllester. On-The-Fly Information Retrieval Augmentation for Language Models. In *Proceedings of the First Joint Workshop on Narrative Understanding, Storylines, and Events*, pages 114–119, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.nuse-1.14. URL https://www.aclweb.org/anthology/2020.nuse-1.14.

[31] Alex Warstadt, Leshem Choshen, Aaron Mueller, Adina Williams, Ethan Wilcox, and Chengxu Zhuang. Call for Papers – The BabyLM Challenge: Sample-efficient pretraining on a developmentally plausible corpus, January 2023. URL http://arxiv.org/abs/2301.11796. arXiv:2301.11796 [cs].

[32] Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The Benchmark of Linguistic Minimal Pairs for English, February 2023. URL http://arxiv.org/abs/1912.00582.

[33] Genta Indra Winata, Samuel Cahyawijaya, Zhaojiang Lin, Zihan Liu, and Pascale Fung. Lightweight and Efficient End-to-End Speech Recognition Using Low-Rank Transformer, February 2020. URL http://arxiv.org/abs/1910.13923. arXiv:1910.13923 [cs, eess].

[34] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization, July 2020. URL http://arxiv.org/abs/1912.08777. arXiv:1912.08777 [cs].

A BLiMP and (Super)GLUE Task Scores

Table 4: (Super)GLUE scores for GPT-Neo, RoBERTa, and their sparse variants.

Name	Overall	BoolQ	CoLA	MNLI	MNLI-MM	MRPC	MultiRC	QNLI	QQP	RTE	SST-2	WSC
GPT-Neo	0.503	0.510	0.504	0.644	0.647	0.466	0.329	0.478	0.688	0.297	0.763	0.206
GPT-Neo CNT ($N = 16$)	0.487	0.471	0.525	0.621	0.634	0.476	0.322	0.465	0.664	0.281	0.729	0.175
GPT-Neo CNT ($N = 64$)	0.496	0.489	0.546	0.625	0.632	0.476	0.316	0.474	0.666	0.340	0.740	0.153
GPT-Neo CNT ($N = 32$)	0.506	0.500	0.507	0.620	0.642	0.527	0.314	0.482	0.665	0.370	0.750	0.192
GPT-Neo MoE ($k = 1$)	0.515	0.479	0.520	0.632	0.632	0.490	0.319	0.482	0.622	0.399	0.755	0.337
GPT-Neo MoE ($k = 3$)	0.503	0.497	0.510	0.645	0.654	0.395	0.296	0.600	0.693	0.290	0.769	0.183
GPT-Neo MoE ($k = 2$)	0.510	0.488	0.488	0.644	0.654	0.509	0.352	0.479	0.682	0.344	0.731	0.244
GPT-Neo PKM ($k = 14$)	0.514	0.490	0.527	0.643	0.646	0.483	0.353	0.483	0.696	0.424	0.771	0.139
GPT-Neo PKM ($k = 42$)	0.514	0.514	0.501	0.630	0.630	0.524	0.338	0.468	0.665	0.353	0.749	0.283
GPT-Neo PKM ($k = 28$)	0.508	0.504	0.527	0.630	0.637	0.486	0.330	0.477	0.660	0.341	0.732	0.264
RoBERTa	0.446	0.449	0.505	0.535	0.522	0.505	0.178	0.409	0.606	0.266	0.722	0.205
RoBERTa CNT ($N = 16$)	0.456	0.449	0.505	0.523	0.554	0.505	0.177	0.459	0.593	0.330	0.714	0.205
RoBERTa CNT ($N = 64$)	0.444	0.449	0.505	0.534	0.524	0.505	0.214	0.472	0.603	0.168	0.707	0.205
RoBERTa CNT ($N = 32$)	0.440	0.449	0.505	0.532	0.537	0.505	0.177	0.449	0.610	0.132	0.736	0.205
RoBERTa MoE ($k = 1$)	0.458	0.449	0.505	0.528	0.515	0.505	0.316	0.442	0.629	0.324	0.710	0.119
RoBERTa MoE ($k = 3$)	0.472	0.449	0.505	0.557	0.554	0.505	0.333	0.413	0.618	0.302	0.748	0.205
RoBERTa MoE ($k = 2$)	0.467	0.449	0.505	0.523	0.531	0.505	0.355	0.419	0.577	0.332	0.742	0.205
RoBERTa PKM ($k = 14$)	0.468	0.449	0.505	0.544	0.534	0.505	0.232	0.475	0.602	0.365	0.737	0.205
RoBERTa PKM ($k = 42$)	0.465	0.449	0.505	0.543	0.543	0.505	0.186	0.474	0.602	0.369	0.731	0.205
RoBERTa PKM ($k = 28$)	0.469	0.449	0.505	0.534	0.550	0.505	0.198	0.476	0.606	0.401	0.730	0.205

Table 5: BLiMP scores for GPT-Neo, RoBERTa, and their sparse variants.

Model	Overall	ANA AGE	AGR STR	BIN	CTRL RAIS	D-N AGR	ELLIPSIS	FILLER GAP	HYP	IRR FRM	ISLAND	NPI	QA EASY	QA TRICKY	QNT	S-A INV	S-V AGR	TURN
GPT-Neo	0.589	0.819	0.602	0.655	0.582	0.627	0.529	0.570	0.491	0.663	0.427	0.440	0.547	0.394	0.660	0.731	0.500	0.561
GPT-Neo CNT ($N = 16$)	0.544	0.728	0.575	0.591	0.606	0.580	0.431	0.608	0.515	0.551	0.484	0.397	0.484	0.436	0.382	0.619	0.502	0.611
GPT-Neo CNT ($N = 64$)	0.538	0.764	0.548	0.638	0.583	0.568	0.472	0.592	0.520	0.528	0.423	0.296	0.406	0.448	0.504	0.636	0.511	0.554
GPT-Neo CNT ($N = 32$)	0.589	0.785	0.584	0.652	0.575	0.601	0.431	0.626	0.512	0.598	0.456	0.489	0.469	0.455	0.691	0.701	0.496	0.529
GPT-Neo MoE ($k = 1$)	0.531	0.727	0.539	0.566	0.585	0.544	0.456	0.564	0.513	0.537	0.413	0.380	0.547	0.406	0.518	0.655	0.489	0.468
GPT-Neo MoE ($k = 3$)	0.567	0.635	0.575	0.617	0.586	0.583	0.479	0.617	0.501	0.599	0.423	0.434	0.500	0.394	0.683	0.643	0.504	0.596
GPT-Neo MoE ($k = 2$)	0.552	0.707	0.544	0.595	0.600	0.547	0.410	0.575	0.488	0.537	0.467	0.474	0.406	0.388	0.625	0.628	0.494	0.550
GPT-Neo PKM ($k = 14$)	0.570	0.780	0.560	0.643	0.603	0.575	0.467	0.594	0.491	0.591	0.483	0.444	0.516	0.388	0.636	0.619	0.515	0.514
GPT-Neo PKM ($k = 42$)	0.552	0.758	0.571	0.587	0.601	0.575	0.393	0.592	0.488	0.600	0.463	0.352	0.422	0.467	0.611	0.624	0.524	0.564
GPT-Neo PKM ($k = 28$)	0.584	0.796	0.587	0.606	0.598	0.600	0.393	0.627	0.458	0.652	0.473	0.434	0.500	0.503	0.690	0.715	0.512	0.589
RoBERTa	0.484	0.391	0.558	0.363	0.574	0.484	0.378	0.290	0.507	0.557	0.515	0.716	0.484	0.382	0.339	0.543	0.480	0.539
RoBERTa CNT ($N = 16$)	0.524	0.274	0.541	0.616	0.578	0.484	0.407	0.502	0.520	0.367	0.430	0.658	0.406	0.418	0.469	0.566	0.507	0.382
RoBERTa CNT ($N = 64$)	0.505	0.334	0.557	0.422	0.583	0.480	0.391	0.290	0.488	0.476	0.447	0.711	0.453	0.400	0.632	0.600	0.476	0.600
RoBERTa CNT ($N = 32$)	0.530	0.399	0.566	0.442	0.592	0.487	0.377	0.526	0.526	0.591	0.502	0.734	0.500	0.376	0.422	0.555	0.496	0.600
RoBERTa MoE ($k = 1$)	0.507	0.340	0.569	0.480	0.597	0.489	0.402	0.495	0.516	0.586	0.462	0.509	0.484	0.364	0.385	0.629	0.497	0.482
RoBERTa MoE ($k = 3$)	0.501	0.456	0.567	0.423	0.590	0.484	0.372	0.461	0.512	0.561	0.498	0.509	0.453	0.364	0.458	0.553	0.520	0.393
RoBERTa MoE ($k = 2$)	0.510	0.343	0.567	0.441	0.590	0.484	0.387	0.481	0.516	0.578	0.507	0.553	0.469	0.382	0.486	0.587	0.505	0.321
RoBERTa PKM ($k = 14$)	0.470	0.382	0.552	0.427	0.576	0.492	0.402	0.291	0.517	0.560	0.497	0.322	0.438	0.382	0.679	0.504	0.501	0.389
RoBERTa PKM ($k = 42$)	0.463	0.384	0.553	0.424	0.577	0.495	0.403	0.290	0.521	0.507	0.466	0.335	0.469	0.382	0.605	0.481	0.503	0.382
RoBERTa PKM ($k = 28$)	0.469	0.350	0.553	0.428	0.575	0.498	0.404	0.294	0.515	0.552	0.502	0.373	0.453	0.376	0.600	0.482	0.505	0.375