HyperRouter: Towards Efficient Training and Inference of Sparse Mixture of Experts
Giang Do, Khiem Le, Quang Pham, Trungtin Nguyen, Thanh-Nam Doan, Thanh-Binh Nguyen, Chenghao Liu, Savitha Ramasam, Xiaoli Li, Steven Hoi

To cite this version:
Giang Do, Khiem Le, Quang Pham, Trungtin Nguyen, Thanh-Nam Doan, et al.. HyperRouter: Towards Efficient Training and Inference of Sparse Mixture of Experts. The 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023 Main, Dec 2023, 2023 - Resorts World Convention Centre, Singapore. pp.1-12. hal-04257067v2

HAL Id: hal-04257067
https://hal.science/hal-04257067v2
Submitted on 17 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Abstract

By routing input tokens to only a few split experts, Sparse Mixture-of-Experts has enabled efficient training of large language models. Recent findings suggest that fixing the routers can achieve competitive performance by alleviating the collapsing problem, where all experts eventually learn similar representations. However, this strategy has two key limitations: (i) the policy derived from random routers might be sub-optimal, and (ii) it requires extensive resources during training and evaluation, leading to limited efficiency gains. This work introduces HyperRouter, which dynamically generates the router’s parameters through a fixed hyper-network and trainable embeddings to achieve a balance between training the routers and freezing them to learn an improved routing policy. Extensive experiments across a wide range of tasks demonstrate the superior performance and efficiency gains of HyperRouter compared to existing routing methods. Our implementation is publicly available at https://github.com/giandip2410/HyperRouter.

1 Introduction

Recent years have witnessed tremendous successes of the Transformer model (Vaswani et al., 2017) and its variants across a wide range of tasks, ranging from natural language and speech processing (Bao et al., 2022b; Gulati et al., 2020), computer vision (Dosovitskiy et al., 2021; Ruiz et al., 2021; Bao et al., 2022a), reinforcement learning (Chow et al., 2023), to life sciences (Rives et al., 2021). Since then, scaling up to larger models has become the prevailing approach for advancing the state-of-the-art in pre-training and finetuning tasks. However, training such large models comes with a high computational cost (Lin et al., 2022); therefore, there is a growing need to develop efficient strategies that facilitate the training of large language models (LLMs) (Fedus et al., 2022a). One of the most effective strategies thus far is the Sparse Mixture-of-Experts (SMoE) (Shazeer et al., 2017; Fedus et al., 2022b), which utilizes routers to direct each input token to a small subset of network parameters (experts). SMoE improves both efficiency and performance compared to dense training approaches (Lewis et al., 2021; Artetxe et al., 2022; Zhang et al., 2022; Du et al., 2022).

Despite the encouraging results, SMoE has been found to encounter the issue of representation collapse, where all experts converge to similar representations (Chi et al., 2022; Chen et al., 2022). This problem arises due to the learning process of the router encouraging experts to cluster around a centroid (Chi et al., 2022). Therefore, significant efforts have been dedicated to addressing the representation collapse issue while preserving the simplicity and efficiency of SMoE training. To this end, one of the most effective strategies is freezing the router, as demonstrated by SMoE-Dropout (Chen et al., 2023), where a randomly initialized router remains fixed throughout the training process. Additionally, Chen et al. (2023) found that progressively increasing the number of selected experts can be beneficial. However, we argue that such a naive strategy exhibits two key limitations. Firstly, the random routing policy may be sub-optimal, which
hinders the overall training process. Secondly, we will show in Section 3.3 that a fixed router will restrict the model’s representation capabilities, necessitating a progressive increase in the number of chosen experts to achieve satisfactory performance. Therefore, SMoE-Dropout inherently suffers from limited representation and does not offer efficiency gains during inference.

This work introduces a novel approach called HyperRouter to address the trade-off between fixed and trainable routers in SMoE training. Instead of employing a random but fixed router, HyperRouter utilizes a random but fixed hypernetwork (Ha et al., 2017) to generate the router’s parameters based on a trainable router embedding. By doing so, HyperRouter can improve the routing policy throughout training while mitigating the representation collapse issues. To demonstrate its effectiveness, we conduct extensive evaluations on various NLP tasks, comparing HyperRouter to several state-of-the-art SMoE routing strategies. Moreover, HyperRouter achieves the same performance threshold with fewer experts (compute) during inference, thereby significantly enhancing the efficiency of deploying LLMs in real-world applications. Fig. 1 shows that HyperRouter consistently outperforms other competitors when the same number of experts is used during inference.

2 Related Work

Existing Routing Mechanism for SMoE. One of the most important components for training SMoE is the expert routing strategy, which specifies experts to process input tokens. There are two common classes of token-expert assignment algorithms for SMoE training: (1) letting tokens select the top-\(k\) experts and (2) letting experts select the top-\(k\) tokens. For the first approach, (Fedus et al., 2022b, Switch Transformer), (Lepikhin et al., 2021, GShard), (Zuo et al., 2022, THOR), (Lewis et al., 2021, BASE), (Clark et al., 2022, S-BASE), and SMoE-Dropout (Chen et al., 2023) are representative methods. Meanwhile, Zhou et al. (2022) introduced expert choice that enables selecting different experts for each token and demonstrates the potential of the second approach.

On the Representation Collapse of SMoE. A major research focus is how to improve the token-expert assignment to avoid the representation collapse issue where: (i) all the inputs are routed to the same expert (Zuo et al., 2022) or (ii) all experts converge to similar representations (Chi et al., 2022; Chen et al., 2022). Such issues result in poor specialization among experts, parameter redundancy, and limited performance gains. Existing works address the issue (i) by employing various ad hoc-heuristics, e.g., adding Gaussian noise (Shazeer et al., 2017), limiting the maximum number of inputs that can be routed to an expert (Lepikhin et al., 2021), imposing a load balancing loss (Fedus et al., 2022b), using linear assignment (Lewis et al., 2021), or eliminating the necessity for router networks ((Zuo et al., 2022) employed a consistent regularized loss for stochastic expert assignment, (Roller et al., 2021) incorporated deterministic hashing assignments). To resolve the issue (ii), (Chi et al., 2022) proposed an X-MOE layer to improve the routing algorithm via dimension reduction, \(\ell_2\) normalization, and gating temperature. Furthermore, (Dai et al., 2022) proposed StableMoE with two training stages to reduce the router’s fluctuations. The work related to ours the most is SMoE-Dropout (Chen et al., 2023), which is considered a randomly initialized and fixed router to route tokens. Thus, the routing policy is stable during training and deterministic during inference. Our work improves upon SMoE-Dropout by extending the fixed router to a hypernetwork to further improve the routing policy during training while alleviating the collapsing issues.

3 Methodology

This section describes SMoE training and details of the proposed HyperRouter method.

3.1 SMoE Training

We first describe SMoE training of LLMs, which consists of a router \(R(\cdot)\) with parameter \(W_r\) and \(N\) expert networks \(\{E_i(\cdot)\}_{i=1}^N\) with parameters \(W_{e_i}\). We follow the most common implementation (Fedus et al., 2022b) to use a linear network as the router and split the feedforward networks in LLMs into \(N\) separated experts.

**Switch Transformer.** Given an input token \(x\) with its representation \(h \in \mathbb{R}^d\), the SMoE output \(y\) is calculated by routing \(h\) only to \(k\)-most suitable experts determined by the router, i.e.,

\[
y = \sum_{j=1}^{N} R(h)_j \cdot E_j(h) \quad \text{and} \quad (1)
\]

\[
R(h) = \text{TopK}((W_r \times h), k),
\]

where the \(\text{TopK}(\cdot, k)\) function keeps the largest \(k\)
values of a given vector while setting the remaining values to zero; $\sigma(\cdot)$ is the standard Softmax function. In practice, only a few experts are activated for the current token ($k \ll N$) for efficient training.

**SMoE-Dropout.** SMoE-Dropout (Chen et al., 2023) is a state-of-the-art strategy that addresses the representation collapse issues (Sec. 2) by simply fixing a randomly initialized router to improve the token routing consistency. Furthermore, SMoE-Dropout gradually increases the number of experts chosen throughout training, starting with $k = 2$ and ending at $k = N$. During the evaluation, SMoE-Dropout proposes to use half or all experts to balance its efficiency and performance.

### 3.2 HyperRouter

We describe our proposed HyperRouter strategy that balances between a random but fixed router and a router optimized during training. HyperRouter employs a fixed hypernetwork (Ha et al., 2017) $\mathcal{H}(\cdot)$ to dynamically generate the router’s parameters conditioned on a trainable router embedding $e$. Specifically, HyperRouter obtains the output $y$ as:

$$y = \sum_{j=1}^{N} \mathcal{R}(h)_j \cdot \mathcal{E}_j(h)$$

$\mathcal{R}(h) = \text{TopK}(\sigma(W_r \times h), k)$ and $W_r = \mathcal{H}(e)$, where $e$ is a low dimensional trainable vector associated with the current layer. All parameters are jointly optimized to minimize the task loss. Lastly, like SMoE-Dropout, HyperRouter also gradually increases the number of activated experts through-out training, starting at $k = 2$ and ending at $k = N$. Fig. 2 visually illustrates our proposed HyperRouter method.

### 3.3 A Deeper Analysis of HyperRouter

We now investigate the representation capabilities of HyperRouter compared to the naive SMoE and SMoE-Dropout. Due to space constraints, we only present the main results here and provide the detailed calculations in Appendix B. First, the naive SMoE jointly trains the router and expert parameters, leading to entanglement and eventually the representation collapse issue (Zuo et al., 2022). SMoE-Dropout proposed to fix the router parameter $W_r$, which we will show to result in a restricted representation. To do so, we need to calculate the Jacobian with respect to $h$, which characterizes how the feature $h$ is involved during training.

**Jacobian of SMoE and SMoE-Dropout.** Let $S_j^{\text{SMOE}}(h) = \sigma(W_r \times h)_j$, $\sigma^{\text{Hyper}}(h) = \sigma(\mathcal{H}(e) \times h)_j$, $I_{ji}$ be the indicator function, and $W_{r,i}$ be the $i$-th column of $W_r$, the Jacobians for SMoE and SMoE-Dropout are calculated as:

$$\nabla_h \mathcal{L} = J_1^\top \nabla_y \mathcal{L} + J_2^\top \nabla_y \mathcal{L}$$

$$= J_1^\top \nabla_y \mathcal{L} + \sum_{i=1}^{k} c_i W_{r,i},$$

where (2)

$$c_i = \sum_{j=1}^{k} S_j^{\text{SMOE}}(h) (I_{ji} - S_j^{\text{SMOE}}(h))$$

$$\times \left( \mathcal{E}_j(h)^\top \nabla_y \mathcal{L} \right).$$

Here, the Jacobian $\nabla_h \mathcal{L}$ is a combination of two terms related to $J_1$ and $J_2$. The $J_1$ component represents how $h$ contributes to the output $y$ and is the same for all methods. The second component related to $J_2$ characterizes learning better experts’ representation of $h$ and is the main difference. Since SMoE-Dropout fixes $W_r$, its $J_2 = J_2^\top \nabla_y \mathcal{L}$ term is expressed as a linear combination of columns in $W_r$, which lies in a lower dimensional subspace because $W_r \in \mathbb{R}^{k \times d}$ with $k \ll d$. The number of experts is usually smaller than the feature dimension. Thus, SMoE-Dropout alleviates the entanglement between the router and experts at the cost of restricting the experts’ representation capabilities.

**Jacobian of HyperRouter.** By a similar calculation, the Jacobian of HyperRouter is shown in equation 3. Given that $e$ is trainable, HyperRouter’s Jacobian is not expressed as a simple linear combination in a much lower-dimensional subspace. In contrast,
HyperRouter maps the feature to a subspace whose dimension can be controlled by $e$, which we can easily set to be greater than $k$. Furthermore, by fixing the hypernetwork, the Jacobian does not change freely as in SMoE, which helps alleviate the entanglement. Overall, HyperRouter can alleviate the representation collapse while not sacrificing the representation capabilities.

Jacobian of HyperRouter:

$$
\nabla_h \mathcal{L} = J_1^\top \nabla_y \mathcal{L} + \sum_{i=1}^{k} c_i \mathcal{H}_i(e), \text{ where } (3)
$$

$$
J_1 = \sum_{j=1}^{k} S_j^{\text{Hyper}}(h) \nabla_h (\mathcal{E}_j(h)),
$$

$$
c_i = \sum_{j=1}^{k} S_j^{\text{Hyper}}(h) \left( \mathbb{1}_{ji} - S_i^{\text{Hyper}}(h) \right) \times \left( \mathcal{E}_j(h)^\top \nabla_y \mathcal{L} \right).
$$

4 Experiments

We conduct experiments to investigate the following hypotheses. First, HyperRouter improves efficiency compared to existing routing strategies, requiring less computation to achieve similar performance thresholds during evaluation. Second, with the same amount of experts used during inference, HyperRouter can achieve better performances compared to other competitors.

4.1 Experiment Setting

Most of our experiments follow Chen et al. (2023) and consider a small TransformerXL (Dai et al., 2019) with four layers in our experiments due to the limited resources. We also investigate the scalability by considering larger variants of the TransformerXL with eight or twelve layers. We compare our HyperRouter with several state-of-the-art routing strategies: (i) SMoE (Fedus et al., 2022b) - training with trainable routers; (ii) THOR (Zhou et al., 2022) - replacing routers with a stochastic experts selection strategy and a consistency regularizer; (iii) SMoE-Dropout (Chen et al., 2023) - using a random router and gradually increasing the number of selected experts during training. We also include two traditional baselines: (i) Dense - the standard training of transformers where no routing mechanisms are implemented; and (ii) Dense+Dropout similar to Dense but with Dropout (Srivastava et al., 2014) inserted in the fully connected layers. All baselines use the same amount of trainable parameters, while our HyperRouter introduces a negligible 1024 trainable parameters for the router embeddings.

We consider two training tasks. First, pretraining on the enwik8 (Mahoney, 2011) and WikiText-103 (Merity et al., 2017) datasets, where we follow the same pre-training procedure as (Chen et al., 2023) to train the small and medium models for 400K iterations. We only train the large models for 100K iterations due to resource constraints. Second, finetuning on the SST-2 (Socher et al., 2013), SST-5 (Socher et al., 2013), IMDb (Maas et al., 2011), and BANKING77 (Casanueva et al., 2020) datasets using the model pre-trained on enwik8. Similar to (Chen et al., 2023), we also perform dense finetuning (always choosing all experts) for several epochs. In all experiments, we report the evaluation metrics on the test set when using different numbers of experts for routing methods. More implementation details and additional results are provided in the Appendix.

4.2 Pre-training Result

Table 1 reports the bit-per-character and perplexity on the enwik8 and WikiText-103 test sets, respectively. Lower is better. $k$ denotes the number of experts chosen during inference. The best results are in bold.

<table>
<thead>
<tr>
<th>$k$</th>
<th>Dense</th>
<th>Dense+Dropout</th>
<th>SMoE</th>
<th>SMoE-Dropout</th>
<th>HyperRouter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>7.20</td>
<td>3.02</td>
<td>1.48</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>3.30</td>
<td>1.61</td>
<td>1.25</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>1.68</td>
<td>1.29</td>
<td>1.19</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>1.32</td>
<td>1.19</td>
<td>1.17</td>
</tr>
<tr>
<td>16</td>
<td>1.25</td>
<td>1.16</td>
<td>1.27</td>
<td>1.16</td>
<td>1.16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$</th>
<th>Dense</th>
<th>Dense+Dropout</th>
<th>HyperRouter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>896.47</td>
<td>560.93</td>
<td>65.17</td>
</tr>
<tr>
<td>2</td>
<td>146.54</td>
<td>89.13</td>
<td>37.75</td>
</tr>
<tr>
<td>4</td>
<td>54.69</td>
<td>40.00</td>
<td>30.45</td>
</tr>
<tr>
<td>8</td>
<td>35.67</td>
<td>29.68</td>
<td>28.05</td>
</tr>
<tr>
<td>16</td>
<td>32.13</td>
<td>31.55</td>
<td>27.66</td>
</tr>
</tbody>
</table>

Table 1: Bit-per-character and Perplexity on the enwik8 and WikiText-103 datasets.
we observe the gap between our HyperRouter Tab. 3 reports the results of the finetuning experiment. The best results are in bold.

<table>
<thead>
<tr>
<th>k</th>
<th>TransformerXL Medium</th>
<th>TransformerXL Large</th>
<th>k</th>
<th>TransformerXL Medium</th>
<th>TransformerXL Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.170</td>
<td>3.741</td>
<td>1.457</td>
<td>2.170</td>
<td>3.741</td>
</tr>
<tr>
<td>2</td>
<td>1.935</td>
<td>1.642</td>
<td>1.201</td>
<td>2.170</td>
<td>3.741</td>
</tr>
<tr>
<td>4</td>
<td>1.260</td>
<td>1.257</td>
<td>1.156</td>
<td>2.170</td>
<td>3.741</td>
</tr>
<tr>
<td>8</td>
<td>1.230</td>
<td>1.167</td>
<td>1.137</td>
<td>2.170</td>
<td>3.741</td>
</tr>
<tr>
<td>16</td>
<td>1.226</td>
<td>1.141</td>
<td>1.132</td>
<td>2.170</td>
<td>3.741</td>
</tr>
</tbody>
</table>

Table 3: Transfer performance (Accuracy) on the SST-2, SST-5, IMDB, and BANKING77 datasets. Higher is better. k denotes the number of experts chosen during inference. The best results are in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>Router 1</th>
<th>Router 2</th>
<th>Router 3</th>
<th>Router 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMOE</td>
<td>2.5164</td>
<td>2.2126</td>
<td>2.2424</td>
<td>2.2583</td>
</tr>
<tr>
<td>SMOE-Dropout</td>
<td>2.5849</td>
<td>2.1624</td>
<td>2.2579</td>
<td>2.2690</td>
</tr>
<tr>
<td>HyperRouter</td>
<td>1.4393</td>
<td>0.8894</td>
<td>1.1269</td>
<td>1.3477</td>
</tr>
</tbody>
</table>

Table 4: Average entropy of the distribution of the routers on the enwik8 dataset. Lower is better.

We now investigate the distributional output from the routers (softmax of router’s output) trained with different strategies. Such outputs determine how tokens are assigned to experts. We hypothesize that high-entropy distributions are not preferred since they are closer to being uniform, indicating the router’s low confidence in choosing experts for the current token. In the extreme, the router outputs a uniform distribution and assigns tokens to all experts, eventually causing the collapse issue. Therefore, a router with a lower entropy is preferred since it confidently assigns tokens to a few experts, which can improve experts’ specialization. To this end, we report the entropy of each router in the small TransformerXL trained on the enwik8 dataset in Tab. 4. The entropy is calculated on all samples in the test set using the model obtained after training. The result clearly shows that HyperRouter achieved much lower entropy than SMOE and SMOE-Dropout. Due to space constraints, we will provide additional visualizations and discussions in Appendix D.

5 Conclusion

In this work, we investigated the potentials and limitations of SMOE for training LLMs and proposed HyperRouter to balance between two extremes of SMOE: training and freezing the router. Consequently, HyperRouter can learn better routing policies while alleviating the representation collapse of conventional SMOE training. Experiments on both pre-training and fine-tuning tasks demonstrated promising HyperRouter’s capabilities in facilitating efficient, effective training and inference compared to state-of-the-art strategies.
Limitations

Our work focuses on the efficiency and efficacy of training LLMs using SMoE. Despite the encouraging results, our experiments are conducted only on medium-scale datasets with a small TransformerXL due to computation limitations. Thus, further empirical evaluations are required to validate the scalability of HyperRouter and other SMoE strategies on recent LLMs and larger datasets.

Ethics Statement

Despite encouraging results, training large-scale LLMs is inevitably costly and requires extensive computational resources, which need to be properly managed. Moreover, our work used data collected on the web, which has been known to suffer from gender and racial biases and requires additional efforts to mitigate its negative impacts. Lastly, our study is a promising step towards facilitating the development of new LLMs, which still requires careful regularization to avoid potential misuses in harmful applications.

Acknowledgement

This research/project is supported by the National Research Foundation, Singapore, under its AI Singapore Programme (AISG Award No: AISG2-RP-2021-027).

References


Advances in Neural Information Processing Systems with a discussion on promising future research

The Jacobian of SMoE and SMoE-Dropout is similar to the Jacobian of HyperRouter. Therefore, only the later detailed calculation is shown. Recall that HyperRouter obtains the output \( y \) as follows:

\[
y = \sum_{j=1}^{N} \mathcal{R}(h)_j \cdot \mathcal{E}_j(h),
\]

\[
\mathcal{R}(h) = \operatorname{TopK}(\sigma(\mathcal{H}(e) \times h), k).
\]

Here, \( \sigma(\cdot) \) is the standard Softmax function given by:

\[
\sigma(\mathcal{H}(e) \times h)_j = \frac{\exp((\mathcal{H}(e) \times h)_j)}{\sum_{i=1}^{N} \exp((\mathcal{H}(e) \times h)_i)} \equiv S_{\text{Hyper}}^j(h), \quad \forall j = 1, \ldots, N.
\]

Without loss of generality and for simplicity, by rearranging the index of the top \( k \)-experts, HyperRouter can be rewritten as follows:

\[
y = \sum_{j=1}^{k} S_{\text{Hyper}}^j(h) \cdot \mathcal{E}_j(h), \quad \text{where} \quad S_{\text{Hyper}}^j(h) = \frac{\exp((\mathcal{H}(e) \times h)_j)}{\sum_{i=1}^{k} \exp((\mathcal{H}(e) \times h)_i)}, \quad \forall j = 1, \ldots, k.
\]

The Jacobian matrix \( J \) of the output \( y \) with respect to \( h \) can be decomposed into two terms as follows:

\[
\nabla_h y = \nabla_h \left( \sum_{j=1}^{k} S_{\text{Hyper}}^j(h) \cdot \mathcal{E}_j(h) \right)
= \sum_{j=1}^{k} S_{\text{Hyper}}^j(h) \nabla_h (\mathcal{E}_j(h))
+ \sum_{j=1}^{k} \nabla_h \left( S_{\text{Hyper}}^j(h) \right) \mathcal{E}_j(h)
= \sum_{j=1}^{k} S_{\text{Hyper}}^j(h) \nabla_h (\mathcal{E}_j(h))
+ \sum_{j=1}^{k} \sum_{i=1}^{k} S_{\text{Hyper}}^i(h) \left( \mathbb{1}_{ji} - S_{\text{Hyper}}^i(h) \right) \mathcal{E}_j(h) \mathcal{H}_i(e)^\top \equiv J_1 + J_2 \quad (4)
\]

(since \( e \) is independent of \( h \)).

Note that the last equation in (4) is obtained by using the chain rule, the logarithmic derivative, and the inner product as follows:

\[
\frac{\partial s_j}{\partial z_i} = s_j \cdot \frac{\partial}{\partial z_i} \log(s_j) = s_j \cdot (\mathbb{1}_{ji} - s_i), \quad \text{where} \quad s_j = \frac{\exp(z_j)}{\sum_{i=1}^{k} \exp(z_i)}, \quad \forall j = 1, \ldots, k.
\]

It is worth mentioning that the first term \( J_1 \) means to produce a better token representation given the current activation router \( S_{\text{Hyper}}^j(h) \), while the second term \( J_2 \) represents learning a better gating function for the appropriate activation score router \( S_{\text{Hyper}}^j(h) \). After the back-propagation, the gradi-
ent of the loss function $L$ is obtained from the two paths mentioned above and is written as follows
\[ \nabla_h L = \nabla_y L \times \nabla_h y = (J_1 + J_2)^\top \times \nabla_y L \] (using (4))
\[ = J_1^\top \nabla_y L + J_2^\top \nabla_y L. \] (5)

Finally, by expanding the second term as follows, we obtain the desired result:
\[ J_2^\top \nabla_y L = \sum_{i=1}^{k} \sum_{j=1}^{k} S_j^{\text{Hyper}}(h) \left( \mathbb{1}_{ji} - S_i^{\text{Hyper}}(h) \right) \]
\[ \times \left( E_j(h)^\top \nabla_y L \right) H_i(e) \]
\[ = \sum_{i=1}^{k} c_i H_i(e), \] where (6)
\[ c_i = \sum_{j=1}^{k} S_j^{\text{Hyper}}(h) \left( \mathbb{1}_{ji} - S_i^{\text{Hyper}}(h) \right) \]
\[ \times \left( E_j(h)^\top \nabla_y L \right). \]

C Additional Experiments

This section provides the implementation details of our experiments in Sec. 4.

C.1 General Setting

Our experiments are conducted based on the publicly available SMoE-Dropout (Chen et al., 2023) implementation\(^1\). Moreover, the pre-training experiments were conducted on a single A40 or A100 GPU, while the finetuning experiments were conducted on a single GeForce RTX 3090 GPU. We also emphasize that parallel training on multiple GPUs might yield different results.

Model architecture. The small TransformerXL variant (Chen et al., 2023) consists of 4 Transformer decoder layers with an input dimension of 256. Each layer consists of a self-attention layer with 8 attention heads, followed by a feed-forward network with an inner dimension of 512. The dropout ratio is kept at 0.1. We split the feed-forward network into 16 experts with the same dimension. For the HyperRouter\(^\text{2}\), we initialize the embeddings with size 256 and employ a 2-layer perceptron with an inner dimension of 256 as the hypernetwork. The ReLU function is used as the activation function for the hypernetwork. Totally our HyperRouter introduces an additional 1024 trainable parameters in the TransformerXL model. It is worth noting that the parameter overhead is fixed as we scale up to larger transformer models. For the medium and large variants, we scale the model to eight and twelves layers, respectively.

C.2 Pre-training Experiments

Tab. A1 provides the implementation details for pre-training our TransformerXL small and medium on enwik8 and WikiText-103. The TransformerXL large network was trained in the same manner, but only for 100K iterations.

Table A1: Implementation details for pre-training experiments on enwik8 and WikiText-103 datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input length</th>
<th>Batch size</th>
<th>Optimizer</th>
<th>Lr</th>
<th># Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>512</td>
<td>22</td>
<td>Adam</td>
<td>2.5e-4</td>
<td>400000</td>
</tr>
<tr>
<td>WikiText-103</td>
<td>512</td>
<td>22</td>
<td>Adam</td>
<td>2.5e-4</td>
<td>400000</td>
</tr>
</tbody>
</table>

\(^1\)https://github.com/VITA-Group/Random-MoE-as-Dropout

\(^2\)SMoE-Dropout
C.3 Finetuning Experiments

To conduct finetuning experiments, we use the same model architecture as in pre-training. Tab. A2 provides the implementation details used for finetuning experiments on four different datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input length</th>
<th>Batch size</th>
<th>Optimizer</th>
<th>Lr</th>
<th># Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST-2</td>
<td>512</td>
<td>16</td>
<td>Adam</td>
<td>1e-4</td>
<td>3</td>
</tr>
<tr>
<td>SST-5</td>
<td>512</td>
<td>16</td>
<td>Adam</td>
<td>1e-4</td>
<td>3</td>
</tr>
<tr>
<td>IMDB</td>
<td>512</td>
<td>4</td>
<td>Adam</td>
<td>1e-4</td>
<td>3</td>
</tr>
<tr>
<td>BANKING77</td>
<td>512</td>
<td>16</td>
<td>Adam</td>
<td>1e-4</td>
<td>15</td>
</tr>
</tbody>
</table>

C.4 Inference Efficiency

We discuss an implementation detail that makes our HyperRouter have the same efficiency during inference as SMoE and SMoE-Dropout. Particularly, before performing inference on any input samples, we use the hypernetwork to generate the router for each layer and then store them. During inference, HyperRouter does not need to re-generate routers. Therefore, HyperRouter enjoys the same inference complexity as SMoE and SMoE-Dropout because their routers have the same dimensionality.

Table A3 reports the number of FLOPs of different methods during inference on the enwik8 dataset according to a different number of experts chosen (k). Note that the FLOPs do not scale linearly with k since SMoE training is only applied to the fully connected layers, while other layers (multi-head self-attention, layer normalization, etc.) remain the same across baselines. In the extreme, using only one expert only incurs 45% of the FLOPs compared to using all 16 experts.

<table>
<thead>
<tr>
<th>Method</th>
<th>Router 1</th>
<th>Router 2</th>
<th>Router 3</th>
<th>Router 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMoE-Dropout</td>
<td>2.5164</td>
<td>± 0.31</td>
<td>± 0.30</td>
<td>± 0.25</td>
</tr>
<tr>
<td>HyperRouter</td>
<td>2.5849</td>
<td>± 0.11</td>
<td>± 0.28</td>
<td>± 0.27</td>
</tr>
<tr>
<td>SMoE</td>
<td>2.5164</td>
<td>± 0.31</td>
<td>± 0.30</td>
<td>± 0.25</td>
</tr>
<tr>
<td>HyperRouter</td>
<td>1.4393</td>
<td>± 0.42</td>
<td>± 0.46</td>
<td>± 0.50</td>
</tr>
</tbody>
</table>

C.5 Parameter Comparison

Tab. A4 provides the number of parameters in different components of SMoE-Dropout and HyperRouter. There are three categories: (i) trainable parameters which are the transformer backbone and the router embeddings; (ii) frozen parameters which are the routers SMoE-Dropout and hypernetworks in HyperRouter; (iii) dynamic parameters that are dynamically generated in each iteration, such as the routers in HyperRouter. Overall, the additional trainable parameters in HyperRouter is negligible. Moreover, the number of frozen parameters (hypernetworks) is quite small compared to the transformer backbone (5.7%). Investigating into sharing the hypernetworks across layers or generating the routers coordinately (Pham et al., 2022) can further reduce this cost while improving the performance, which we will leave for the future work.

D Routing Visualization

Table A5: Average entropy of the distribution of the routers’ output on the enwik8 dataset. Lower is better.

This Section provides the full details of the routers’ entropy and visualizes its distributional output. This is supplementary to Section 4.4. Tab. A5 reports the mean and standard deviations of the entropy at each router. This table is the full version of Tab. A5. We can see that all methods have rather low standard deviation, indicating that the differences are significant. We also provide an illustrative example of the routers’ outputs using a randomly picked sample on the test set in Fig. 4. Here we can clearly see that the policies from SMoE and SMoE-Dropout are much closer to uniform while HyperRouter’s policies are much sharper and have lower entropy. We emphasize that this example is not cherry-picked since we already calculated the averaged entropy.
on all samples in Tab. A5. Overall, this result shows insights into how HyperRouter can perform better than other state-of-the-art SMoE strategies.

E Future Work

Our HyperRouter opens several promising venues for future research. Particularly, we believe that investigating two HyperRouter components: (i) fixed hypernetwork, and (ii) trainable embedding can yield further performance and efficiency gains. Potential directions include incorporating regularization such as $\ell_2$-penalty or dropout (Peng et al., 2015) or using better hypernetwork initialization techniques (Chang et al., 2020). Furthermore, the current implementation uses a hypernetwork for each transformer layer, and it generates all parameters of the router. Sharing hypernetworks among layers or generating the router coordinate-wise can offer knowledge sharing (Yin et al., 2021; Pham et al., 2022), which can further improve the result.
Figure 4: Visualization of the distribution of the routers’ output.