MoEC: Mixture of Expert Clusters

Yuan Xie *, Shaohan Huang, Tianyu Chen, Furu Wei
Microsoft Research Asia, China
{v-yuanxie, shaohanh, v-tianyuchen, fuwei}@microsoft.com

Abstract
Sparsely Mixture of Experts (MoE) has received great interest due to its promising scaling capability with affordable computational overhead. MoE models convert dense layers into sparse experts, and utilize a gated routing network to make experts conditionally activated. However, as the number of experts grows, MoE with outrageous parameters suffers from overfitting and sparse data allocation. Such problems are especially severe on tasks with limited data, thus hindering the progress towards improving performance by scaling up. We verify that there exists a performance upper bound of scaling up sparse MoE. In this work, we propose Mixture of Expert Clusters — a general approach to enable expert layers to learn more diverse and appropriate knowledge by imposing variance-based constraints on the routing stage. Given this, we could further propose a cluster-level expert dropout strategy specifically designed for the expert cluster structure. Our experiments reveal that MoEC could improve performance on machine translation and natural language understanding tasks. MoEC plays a positive role in mitigating overfitting and sparse data allocation problems, thus fully releasing the potential of large-scale sparse models.

Introduction
Scaling up the model capacity has shown to be promising to achieve better performance on a variety of tasks, including natural language understanding (Brown et al. 2020; Raffel et al. 2019) and visual representation learning (Dosovitskiy et al. 2020; Bao, Dong, and Wei 2021). The continued growth in model size and parameters brings higher computational cost, while large dense models have almost hit the boundary of hardware capacity. In pursuit of better computational efficiency, sparse Mixture-of-Experts (MoE) is proposed as an efficient alternative to dense models (Lepikhin et al. 2020; Fedus, Zoph, and Shazeer 2021; Riquelme et al. 2021; Lewis et al. 2021). The sparsely-gated MoE transformers, the feed-forward network (FFN) sub-layer will be replaced by a set of experts with independent parameters. The sparsity of MoE is brought by experts and the gated routing network. The gated routing network will calculate the routing score between input tokens and each expert and activate experts with top-k routing scores. Most experts will not be activated, thus forming a sparse structure. Since the computation cost is only proportional to the activated top-k sub-network, sparsely activated MoE models could scale up model parameters without significantly increasing computational cost. With affordable computational overhead, MoE models could achieve better performance than dense models on various tasks such as neural machine translation (Lewis et al. 2019; Conneau and Lample 2019; Lepikhin et al. 2020), image recognition (Riquelme et al. 2021) and speech recognition (Kumatani et al. 2021).

Recent studies have reached a consensus that more experts mean more parameters and large model capacity, which always bring improvements. However, some studies show that more trainable parameters and sparse conditional computation may introduce overfitting (Xue et al. 2021; Lou et al. 2021; Xue et al. 2022), especially for downstream tasks with limited data. As depicted in Figure 1, as the number of experts grows, overfitting gradually becomes apparent in the machine translation task. Moreover, we find that enlarging the size of the MoE will not always lead to improvement. There seems to exist a performance upper bound of scaling up experts with limited data.

Besides, we find an unreasonable phenomenon in Figure 1: 64-expert MoE with more parameters and larger model capacity has higher training loss than 32-expert MoE. It implies that large-scale MoE not only suffers from overfitting, but also has other hidden problems that affect training. According to our analysis, the probability of each expert getting a token reduces proportionately as the number of experts grows. With the same data, each expert will get less diverse samples. Insufficient data not only affects the training of expert layers, but also aggravate overfitting. Therefore, we want to explore ways in which experts could get diverse samples and learn abundant knowledge, thereby alleviating overfitting and sparse data allocation.

In this work, we propose Mixture of Expert Clusters (MoEC), a general optimizing strategy for MoE models. All the experts in MoE models are clustered to form several static expert clusters by minimizing the clustering loss associated with the routing probability among neighbor experts. The inductive bias expects that the similarity of intra-cluster experts is high while the similarity of inter-cluster experts is
As follows:

low. Experts within a cluster are prone to tokens with similar hidden states and could “share” similar tokens. Given the cluster structure, we further propose a cluster-level expert dropout strategy. Several experts in the cluster will be randomly dropped, and the dropped experts will not participate in the routing stage. The activated experts are selected from the remaining experts in the cluster. Implementing dropout within clusters could make tokens always dispatched to suitable experts, no matter how random the dropout is.

We evaluate our MoEC on machine translation and natural language understanding tasks. Experiment results show that MoEC outperforms dense models and baseline MoE models. It indicates that MoEC retains the advantages of the sparse structure of MoE, and alleviates overfitting and sparse data allocation problems. Our contributions are summarized as follows:

- We point out the overfitting and sparse data allocation problems for large-scale MoE models, and experts getting less diverse samples could be the common cause of both problems.
- We propose to build expert clusters by variance-based constraints, which allows experts to get a more diverse set of similar tokens. We also implement cluster-level expert dropout as a regularization method.
- We conduct experiments on machine translation and natural language understanding tasks. MoEC could improve performance and alleviate problems caused by scaling up experts without changing the model structure and routing strategy.
- We find that there exists a performance upper bound for scaling up MoE models with limited data. MoEC could raise the performance upper bound, thus exploiting the potential of large-scale sparse models.

Figure 1: A simple demonstration of loss curves of MoE models on WMT-14 English-to-German translation task. We show the loss curve of MoE baseline models with different experts. The value in the box represents the minimum loss.

Related Work

In the context of modern deep learning architectures, scaling up transformers using sparse Mixture of Experts (MoE) is proven to be effective to achieve state-of-the-art performance on various NLP and CV tasks (Shazeer et al. 2017; Lepikhin et al. 2020; Riquelme et al. 2021; Fedus, Zoph, and Shazeer 2021). Compared with dense transformers, an MoE model contains several experts (feed-forward networks), and a router to select top-k experts for input tokens. It increases the model capacity by such conditional computation while maintaining computational efficiency. To future explore the potential of MoE, some studies focus on router assignment algorithm (Lewis et al. 2021; Roller et al. 2021; Dai et al. 2022). Besides, some work focus on optimizing training methods for MoE models. Du et al. (2021) applied a temperature heating mechanism for sparse MoE models on the translation task. Chi et al. (2022) proposed a dimension reduction to estimate the routing scores between tokens and experts on a low-dimensional hyper-sphere. Our work is also proposed to optimize the MoE model. Instead of changing the model structure and routing strategy, MoEC establishes static expert clusters, which allows experts to be assigned similar and more diverse tokens.

Although MoE models have achieved promising results, they are proven to have overfitting problems (Fedus, Zoph, and Shazeer 2021; Wu et al. 2022; Xue et al. 2022) on downstream tasks with limited data. To mitigate overfitting, some works use knowledge distillation to distill MoE models into small-sized MoE models or dense models (Xue et al. 2022; Dai et al. 2022). Another approach is to apply the dropout strategy during training. Fedus, Zoph, and Shazeer (2021) set a small dropout rate at non-expert layers and a larger dropout rate at expert layers. Liu et al. (2022) propose gating dropout, which allows some tokens to ignore the gated routing network and stay at their local machines to reduce cross-machine communication. In our work, we propose the cluster-level expert dropout. Randomly selected experts in the cluster will be dropped so that they will not participate in the routing stage.

Preliminary

To build MoE transformers, it is a common practice to replace feed-forward network (FFN) sub-layers with a set of experts. The experts share the same structure with the FFN layer in the dense transformer model. We denote the hidden representation of input token $x$ as $h$, and the embedding for the $i$-th expert as $e_i$. The router computes the routing score $s_i = h^T e_i$ to compare the similarity between $h$ and the set of experts $E$. Then, the router utilizes a gating function $\alpha_i(\cdot)$ to compute the gated value of expert $i$.

$$\alpha_i = \begin{cases} \frac{\exp(s_i)}{\sum_{j=1}^{E} \exp(s_j)}, & \text{softmax gating} \\ \frac{1}{1 + \exp(-s_i)}, & \text{sigmoid gating} \end{cases}$$

The gating function $\alpha_i$ represents the probability of dispatching input token to expert $i$. The top-k gated-value is
used for dispatching the token \( x \) according to \( \alpha_i \). The corresponding \( k \) expert networks are conditionally activated. We denote the set of selected top-\( k \) indices as \( K \).

\[
y = \sum_{i \in K} \alpha_i \cdot E_i(x)
\]

where \( E_i(x) \) is the \( i \)-th expert network, which is a feed-forward network. The output of the gated routing network is the linearly weighted combination of each expert’s computation on the token by the gate value.

**Method**

In this work, our goal is to give experts access to more diverse training samples, thus mitigating overfitting and sparse data allocation. We encourage the static clustered structure by minimizing the clustering loss associated with the routing probability among neighbor experts. We apply the variance-based clustering loss to implement constraints. Besides, we further propose a cluster-level expert dropout strategy.

In our work, we use top-1 gating. Only the expert with the largest routing score is activated. And we choose softmax as our activation function. Experts in a cluster will be distributed on the same device to reduce communication costs.

**Mixture of Expert Clusters**

We illustrated our MoEC (Mixture of Expert Clusters) in Figure 2. For classic MoE, the routing probability of tokens will not be constrained. The router will always dispatch input tokens to their best-matched experts, while other similar tokens have little chance of being selected. When scaling up the number of experts, the sparse data distribution will cause each expert to get less diverse tokens. Expert layers could not get adequately trained. Also, the amount of data is insufficient to match the growing number of parameters, which is the main reason for overfitting. In order to solve the problems of classic MoE, our MoEC allows each expert to get more rich and diverse tokens. We impose variance-based constraints on the routing stage, aiming to make neighbor experts have similar routing probabilities for input tokens, thus forming expert clusters prone to tokens with similar hidden states. In MoEC, experts will get a more diverse set of similar input tokens by “sharing” input tokens with other experts in the cluster.

Compared with previous work related to MoE, our training objective added an extra term - cluster loss. The overall training objective is to minimize:

\[
L = L_{\text{task}} + L_{\text{balance}} + L_{\text{cluster}}
\]

\( L_{\text{task}} \) is determined by the specific task. In our work, we employ the label smoothed cross-entropy loss for neural machine translation, masked language modeling loss for pre-training language model, and negative log-likelihood loss (NLL loss) or mean-squared loss (MSE loss) for GLUE tasks. In the following, we will introduce \( L_{\text{balance}} \) and \( L_{\text{cluster}} \).

**Load Balancing Loss.** During training, there exists a load imbalance issue between experts (Shazeer et al. 2017; Lepikhin et al. 2020): Most tokens are dispatched to a small number of experts, while many other experts do not get sufficiently trained at all. Besides, imbalanced assignments will result in a high computational bottleneck in the MoE layer and thus limit the computational efficiency. We follow the work in (Fedus, Zoph, and Shazeer 2021) and add the balance loss to the training objective to encourage a balanced load across experts. Given \( N \) experts indexed by \( i=1 \) to \( N \), the balance loss is computed as follows:

\[
L_{\text{balance}} = \alpha N \cdot \sum_{i=1}^{N} f_i \cdot p_i
\]

where \( f_i \) is the fraction of tokens dispatching to expert \( i \). We denote the number of tokens dispatched to the \( i \)-th expert as \( \text{Count}_i \). Given a batch \( B \) with \( T \) tokens, \( f_i = \text{Count}_i / T \). \( p_i \) is the fraction of the routing probability allocated for expert \( i \) in the batch \( B \). It is calculated by averaging the probability of routing token \( x \) to expert \( i \) in the batch \( B \).

\[
p_i = \frac{1}{T} \sum_{x \in B} \alpha_i(x)
\]

where \( \alpha_i(x) \)is the gating function depicted in Equation 1, which represents the probability of dispatching token \( x \) to expert \( i \). The balance loss in Equation 4 encourages uniform routing since it would be minimized under a uniform distribution. To control the impact of balance loss in the training process, a hyper-parameter \( \alpha \) is applied as a multiplicative coefficient for the loss. Throughout this work, we use an \( \alpha = 10^{-3} \) which was sufficiently large to ensure load balancing while small enough not to overwhelm the primary cross-entropy objective.

**Clustering Loss.** In our work, we find the sparse allocation of data severely hinders the adequate training of MoE layers and exacerbates overfitting. In order to allow experts to get rich and diverse tokens to mitigate the impact of sparse allocation, we design the clustering loss. The loss is designed to constrain certain neighbor experts so that they will share similar routing probabilities to tokens, thus forming a static cluster-like distribution. For input tokens originally
dispatched to the best-matched experts, clustering loss will give them more opportunities to access other experts in the cluster. As a result, experts will be assigned a more diverse set of similar tokens, thus alleviating the problem of sparse allocation.

In MoE models with $N$ experts, the clustering loss will guide experts to form $m$ clusters ($N$ should be divisible by $m$), and each cluster contains $L = \frac{N}{m}$ experts. We use $E_i^j$ to represent the j-th expert in the i-th cluster, while $p_i^j$ represents the routing probability allocated for $E_i^j$ ($i = 0, 1, ..., m - 1; j = 0, 1, ..., L - 1$). According to the size and number of clusters, $p_i^0, p_i^1, ..., p_i^{L-1}$ will compose a one-dimensional matrix $\tilde{P}_i \in \mathbb{R}^L$ to represent the routing probabilities of the $L$ experts in the i-th cluster, and we denote the mean value of them as $\overline{p}_i$. We define the clustering loss as follows:

$$L_{clustering} = \beta N \cdot C_{intra} \cdot C_{inter}$$
$$= \beta N \cdot \sum_{i=0}^{m-1} \delta_i(\tilde{P}_i) \cdot e^{-\mu \max \{ \overline{p}_i \} - \mu \max \{ \gamma \} (\frac{m-1}{m})}$$

(6)

As can be seen from Equation 6, clustering loss is mainly composed of two parts: the variance-based intra-cluster constraint $C_{intra}$ and the difference-based inter-cluster constraint $C_{inter}$. $\delta_i(\tilde{P}_i)$ represents the variance of the routing probability in the i-th cluster. We compute the mean variance of $m$ clusters as the intra-cluster constraint $C_{intra}$, which will be minimized when the routing probabilities of experts within the same cluster are balanced.

Besides, we use $C_{inter}$ to measure the probability difference between the dispatched cluster and the sub-optimal cluster. $\max \{ \cdot \}$ means the max value of $\overline{p}_i$ ($i=0,1, ..., m-1$) and $\max_{2} \{ \cdot \}$ means the second max value. $C_{inter}$ will be minimized when the probability of a token being dispatched to a suboptimal cluster is low. $\mu$ is the coefficient used to control the value of $C_{inter}$. When we set $\mu = 0$, the probability difference between clusters will not be considered. We could also set $\mu$ to a non-zero value to activate $C_{inter}$. We will conduct in-depth experiments and analysis on it in Section 5.6.

To minimize clustering loss, the probability distribution within the cluster should be uniform, and the probability difference between the clusters should be more apparent (optional). In the initial training steps, the variance among experts will be very high, so the clustering loss will dominate the optimization and guide the rapid formation of expert clusters. When the intra-cluster variance is stable, the clustering loss will become relatively small to maintain the expert clusters. Similar to the practice in balance loss, a hyper-parameter $\beta$ is applied. The value of the $\beta$ should be relatively small, because a large $\beta$ means a strong clustering constraint, thus making experts in the cluster too similar. It will cause these experts to lose their characteristics, and the contributions of multiple similar experts are only approximately equal to one expert. In our work, we set the value of $\beta$ as $10^{-2}$ by default. Experiments on the selection of $\beta$

values could be found in Appendix A.

Cluster-Level Expert Dropout

When applying large-scale MoE models on tasks with limited data, overfitting issues naturally arise. Previous MoE-related work (Raffel et al. 2019; Fedus, Zoph, and Shazeer 2021) used dropout (Srivastava et al. 2014) at each layer to prevent overfitting. Here, cluster-level expert dropout acts as a regularization technique completely different from traditional dropout. It does not drop parameters, but drops some experts in the cluster, which makes the dispatching of tokens more random.

Implementation in Clusters. First, our cluster-level expert dropout works at the routing stage, so it will only be implemented at expert layers. For experts in a cluster, we randomly drop some of them by deleting the expert ids from the candidate expert list when calculating the routing probability. Thus, the corresponding experts will be ignored in the routing stage. Assume the dropout rate as $\gamma$, only the remaining $N(1 - \gamma)$ experts will participate in the calculation of routing probability during training. The dimension of the matrix $P$ will decrease from $\mathbb{R}^N$ to $\mathbb{R}^{N(1-\gamma)}$. All clusters implement the dropout simultaneously. It allows tokens to have more opportunities to be dispatched to other experts in the same cluster, instead of being repeatedly dispatched to the expert with the highest probability. From another perspective, each expert will receive more diverse tokens without adding training data.

Cluster-Level Expert Dropout vs Traditional Expert Dropout. Traditional expert dropout is recommended in Fedus, Zoph, and Shazeer (2021). It is a dropout technique (Srivastava et al. 2014) to regularize MoE models, which acts on the feed-forward layer to reduce overfitting caused by too many parameters. By setting a relatively small dropout rate at non-expert layers (0.1), expert dropout increases the dropout rate by an explicit amount at the interim feed-forward computation at each expert layer (0.4). Our expert dropout acts completely different from it. We perform random dropout on the candidate list of experts during the routing stage. It does not reduce the number of parameters during training but allocates tokens more diversely and flexibly. While traditional expert dropout is usually used for fine-tuning on downstream tasks, our cluster-level expert dropout
is a general regularization mechanism with strong generality. In addition, our dropout can be applied together with Fedus’ expert dropout, and they can work together to improve the performance of MoE.

Why Cluster-Level Is Better? It is natural to think that expert dropout could be implemented at the global level, which provides more opportunities for tokens to access other sub-optimal experts. But for global-level expert dropout, as shown in Figure 3, if a random dropout happens to drop suitable experts, tokens may be dispatched to less relevant experts. Inappropriate dispatching may negatively impact the learning of experts. In MoEC, we address this problem by exploiting the cluster-like structure and design a cluster-level expert dropout. Cluster-level dropout could give tokens the option to be randomly re-dispatched while confining the routing results to a more reasonable range. No matter how random the dropout is implemented, tokens will always be dispatched to experts with similar routing probability. We will conduct in-depth experiments and analysis in Section 5.5.

Experiments
We name our model MoEC (Mixture of Expert Clusters), and evaluate the performance on bilingual machine translation and natural language understanding tasks. We use the X-MoE model from Chi et al. (2022) as our backbone architecture, which has shown better performance than prior MoE models such as Switch Transformers (Fedus, Zoph, and Shazeer 2021) on widely-used cross-lingual understanding benchmarks.

Evaluation Dataset
WMT 2014 English-to-German. Ninth Workshop on Statistical Machine Translation (WMT 2014) releases a collection of datasets used in shared tasks including machine translation. We add additional news-commentary-v12 data from WMT-17 for training and validation. The total training data contains 3.96M English-to-German sentence pairs.

GLUE. General Language Understanding Evaluation (Wang et al. 2018) benchmark is a collection of tools for evaluating the performance of models across a diverse set of existing NLU tasks, including MNLI (Williams, Nangia, and Bowman 2017), CoLA (Warstadt, Singh, and Bowman 2019), SST-2 (Socher et al. 2013), QQP, QNLI (Rajpurkar et al. 2016), MRPC (Dolan and Brockett 2005) and STS-B (Cer et al. 2017). We do not perform experiments on RTE because previous work (Chen et al. 2022) demonstrated that MoE is not suitable for this task. It is worth mentioning that we will pre-train our model on the BooksCorpus (Zhu et al. 2015) and English Wikipedia corpus for 120k steps before fine-tuning on GLUE tasks.

Experiments Setup
Model Architecture. For our MoEC and all baseline models, we follow the recommended settings in Vaswani et al. (2017) and use Transformer-big as the unified backbone architecture on WMT 2014 English-German translation task. For GLUE tasks, we use Transformer-base as the backbone architecture. For MoE layers, we apply the 64-expert MoE model with 3 FFN sub-layers in the 3rd encoder block and 3rd decoder block (same as the setting in Lewis et al. (2021)). A more detailed model hyper-parameters could be found in Appendix B.

Baselines. We conduct two baselines in our experiments. The first is dense transformer (Vaswani et al. 2017). For another, we follow the work in Chi et al. (2022) and apply X-MoE as our MoE baseline. It could serve as a strong baseline that shows better performance than Switch Transformer (Fedus, Zoph, and Shazeer 2021) on widely-used cross-lingual understanding benchmarks. The MoE baseline estimates routing scores between tokens and experts on a low-dimensional hypersphere and adds a learnable temperature scalar in the gating function. For a fair comparison, the two baseline methods are built with the same setting as MoEC, which could be found in Appendix B.

MoEC Hyper-Parameters. For MoEC, several unique hyper-parameters are introduced. For clustering loss, we set \( \beta \) to \( 10^{-2} \) according to the experiment results (see Appendix A) and set \( \mu = 0 \) by default. For cluster size (the number of experts in a cluster) and expert dropout rate, we will have detailed related experiments in the following sections.

Training Hyper-Parameters. For a fair comparison, the dense model, MoE baseline model, and MoEC model share the same training hyper-parameters. All models are trained with the Adam optimizer (Kingma and Ba 2014) \( (\beta_1 = 0.9, \beta_2 = 0.98) \). The learning rate is set \( 5 \times 10^{-4} \) with 4000 warm-up steps and inverse square root scheduler (Raffel et al. 2019). Batch size, training steps, and dropout rate are set by different tasks, which are recorded in Appendix C.

---

<table>
<thead>
<tr>
<th>NMT</th>
<th>GLUE Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMT14 En-De</td>
<td>MNLI</td>
</tr>
<tr>
<td>Dense</td>
<td>27.10</td>
</tr>
<tr>
<td>MoE Baseline</td>
<td>30.59</td>
</tr>
<tr>
<td>MoEC (w/o expert dropout)</td>
<td>32.21</td>
</tr>
<tr>
<td>MoEC</td>
<td>32.50</td>
</tr>
</tbody>
</table>

Table 1: The performance on machine translation and GLUE tasks for baselines and MoEC models. WMT-14 is measured on the test set, while GLUE tasks are measured on the development sets. We report the average results by a set of seeds (see Appendix C). All experiments are conducted with 64 experts.
Experiments Results

We train dense models, baseline MoE and MoEC models on several widely-used evaluation tasks and the results are shown in Table 1. Compared with dense models, MoE models exhibit significant performance improvements, which benefit from the large model capacity. Our MoEC could bring notable improvement over the MoE baseline without applying the dropout strategy to experts. On WMT-14, it gives a 1.62 BLEU score boost. The advantage could be attributed to the clustered distribution of experts, which endows experts with more diverse training samples. Moreover, with the application of the cluster-level expert dropout strategy, the performance of MoEC will be further improved.

As shown in Figure 4, the MoE baseline severely suffers from overfitting on WMT-14, while our MoEC shows excellent ability to mitigate overfitting. The overfitting phenomenon on the validation set is almost eliminated, and the validation loss is relatively lower. It shows that when our MoEC solves the sparse allocation of data, each expert could get more abundant and diverse training samples. In this way, the training data of each expert is kept sufficient, thereby alleviating the phenomenon of overfitting. Furthermore, we found that MoEC converges slightly slower. It is due to the fact that each expert needs to learn from more diverse training samples, which takes more steps to allow the expert to get sufficiently trained.

Detailed Analysis of Expert Clusters

Next, we conduct a detailed analysis of expert clusters. Figure 5 shows the fraction of tokens dispatched to cluster 0 (including expert 0∼3) during training and inference. During training, experts in cluster 0 get similar input tokens. It reveals that clustering does not affects load balancing issues. During inference, the routing probabilities of experts in the cluster vary, which indicates that experts still retain their own characteristics. Experts could learn more fine-grained knowledge, which is the advantage of multiple similar experts compared to a single expert. For WMT14, the BLEU score of MoE with 16 experts is 30.49 (see Table 4), while the BLEU score of MoE with 16 clusters (cluster size=4) is 32.16. It proves that multiple similar experts have an obvious advantage over a single expert.

The cluster size also has a critical impact on the learning of MoEC, so we conduct experiments on different cluster sizes. As depicted in Table 2, the best performance is obtained when cluster size = 8. Compared to the MoE baseline with 64 experts, expert clusters could bring about a 1.62 BLEU scores improvement. When the cluster size is relatively small, the data shared among experts will be less, and the improvement brought by MoEC will not be fully exploited. As a special case, when cluster size=1, a single expert could not be called a cluster, and MoE is equivalent to MoE baseline. When the cluster size is large, the data shared among experts will increase, but the similarity and correlation of these data will become lower, which will lead to an adverse impact on the “professionalism” of each expert. When we expand the cluster size to 16, the performance of MoEC is even lower than that of the MoE baseline, which means that an excessively large cluster size will suppress the advantages of MoE structure and hurt the performance.

Expert Dropout: Cluster-Level vs Global-Level

In Table 3, we experiment on WMT-14 with the cluster-level expert dropout rate. We find that cluster-level dropout could enhance the generalization performance of MoEC. Such a regularization method could bring a 0.29 BLEU scores improvement for MoEC. Experimental results show that 0.5 is
Table 3: Cluster-level vs global-level expert dropout on WMT-14. All experiments are conducted on the 64-expert MoEC and cluster size = 8.

<table>
<thead>
<tr>
<th>Dropout rate</th>
<th>Cluster-level</th>
<th>Global-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32.21</td>
<td>32.21</td>
</tr>
<tr>
<td>0.25</td>
<td>32.32</td>
<td>31.88</td>
</tr>
<tr>
<td>0.5</td>
<td><strong>32.50</strong></td>
<td>31.53</td>
</tr>
<tr>
<td>0.75</td>
<td>32.02</td>
<td>29.73</td>
</tr>
</tbody>
</table>

Figure 6: Two sets of experiments on the inter-cluster constraint coefficient \(C_{inter}\). All experiments are performed on WMT14 En-De. The figure on the left is about experiments with different expert dropout rates (cluster size=8), and the figure on the right is about experiments with different cluster sizes (without expert dropout).

Table 4: Results of scaling up MoEC.

<table>
<thead>
<tr>
<th>Expert num</th>
<th>MoE baseline</th>
<th>MoEC</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>30.49</td>
<td>30.50</td>
<td>+0.01</td>
</tr>
<tr>
<td>32</td>
<td><strong>30.81</strong></td>
<td>30.84</td>
<td>+0.03</td>
</tr>
<tr>
<td>64</td>
<td>30.59</td>
<td><strong>32.50</strong></td>
<td>+1.91</td>
</tr>
<tr>
<td>128</td>
<td>30.21</td>
<td>32.40</td>
<td>+2.19</td>
</tr>
</tbody>
</table>

Conclusion

In our work, we point out the overfitting and the sparse data allocation problems of large-scale MoE models and propose a novel training strategy - MoEC to convert experts into clusters. Each expert could get more abundant and diverse training samples. In this way, the training data of each expert is kept sufficient, thereby alleviating overfitting. We also propose the cluster-level expert dropout to realize regularization. We conduct experiments on machine translation and natural language understanding tasks. Experiment results show MoEC could improve performance and alleviate problems caused by scaling up experts without changing the model structure and routing strategy. The superiority of the large-scale MoE model will be better exerted by MoEC, thereby raising the upper bound of MoE models. With the help of MoEC, we could try to build sparse models with more experts.
Selection of the Value of Clustering Coefficient

<table>
<thead>
<tr>
<th>Value of $\beta$</th>
<th>MoEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1e^{-3}$</td>
<td>32.21</td>
</tr>
<tr>
<td>$5e^{-3}$</td>
<td>32.17</td>
</tr>
<tr>
<td>$1e^{-2}$</td>
<td>32.32</td>
</tr>
<tr>
<td>$5e^{-2}$</td>
<td>31.21</td>
</tr>
</tbody>
</table>

Table 5: The performance of MoEC with different $\beta$ coefficients on WMT-14. All experiments are conducted with 64 experts. The cluster sizes=8, and expert dropout rate=0.25.

Table 5 presents the experiments on selecting the best value of $\beta$. MoEC works best when $\beta$ is set to $1e^{-2}$. And when the beta value is too large, the performance of MoEC drops significantly, which confirms our analysis in the main text. Based on the results, we uniformly set the value of $\beta$ as $10^{-2}$ as a default in all experiments above.

Architecture Parameters

Table 6 presents the architecture parameters for different tasks. For the WMT-14 task, we apply a larger Transformer model with more attention heads and larger embedding dimensions. As our backbone architecture, X-MoE (Chi et al. 2022) applies half of the number of experts as the routing dimension.

Training Hyper-Parameters

Table 7 presents the training hyper-parameters for WMT-14 and pre-training. To avoid the two regularization methods (MoE dropout and our cluster-level dropout) affecting each other, we do not use expert dropout for a fair comparison.

Table 8 presents the training hyper-parameters on downstream GLUE tasks. For GLUE tasks with a relatively large amount of training data (MNLI, SST-2, QQP, QNLI), we set smaller training epochs. For those with limited data (CoLA, STS-B, MRPC, RTE), we try to increase the number of training epochs to 10.

<table>
<thead>
<tr>
<th>-</th>
<th>WMT-14</th>
<th>Pretrain&amp;GLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer blocks</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Attention heads</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>Encoder ebd</td>
<td>1024</td>
<td>768</td>
</tr>
<tr>
<td>Decoder ebd</td>
<td>1024</td>
<td>768</td>
</tr>
<tr>
<td>FFN ebd</td>
<td>4096</td>
<td>3072</td>
</tr>
<tr>
<td>Experts</td>
<td>16,32,64,128</td>
<td>16,32,64,128</td>
</tr>
<tr>
<td>Routing dimension</td>
<td>8,16,32,64</td>
<td>8,16,32,64</td>
</tr>
<tr>
<td>MoE layers</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Sub-layers</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6: Architecture parameters for all tasks. “ebd” is short for “embedding”.

<table>
<thead>
<tr>
<th>-</th>
<th>WMT-14 En-De</th>
<th>Pre-train</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>Adam $\epsilon$</td>
<td>1e-6</td>
<td>1e-6</td>
</tr>
<tr>
<td>Adam $\beta$</td>
<td>(0.9,0.98)</td>
<td>(0.9,0.98)</td>
</tr>
<tr>
<td>Training steps</td>
<td>32k</td>
<td>125k</td>
</tr>
<tr>
<td>Batch size</td>
<td>8k</td>
<td>2k</td>
</tr>
<tr>
<td>Maximum LR</td>
<td>$5e^{-4}$</td>
<td>$5e^{-4}$</td>
</tr>
<tr>
<td>LR scheduler</td>
<td>inverse sqrt</td>
<td>inverse sqrt</td>
</tr>
<tr>
<td>Warmup steps</td>
<td>4k</td>
<td>4k</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>Attention dropout</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>Gradient clip norm</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>Label smoothing</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>Capacity factor</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MoE dropout</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Coefficient $\alpha$</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 7: Training hyper-parameters for all tasks. “LR” represents “learning rate”.

<table>
<thead>
<tr>
<th>-</th>
<th>CoLA,RTE</th>
<th>STS-B</th>
<th>MRPC</th>
<th>Else</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSZ</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Epochs</td>
<td>3.5,10</td>
<td>10,15,20</td>
<td>5,10,15,20</td>
<td>3.5</td>
</tr>
<tr>
<td>LR</td>
<td>$[1,2,4]e^{-5}$</td>
<td>$[1,2,4]e^{-5}$</td>
<td>$[1,2,4]e^{-5}$</td>
<td>$[1,2,4]e^{-5}$</td>
</tr>
<tr>
<td>Warm</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Seed</td>
<td>1,2,3</td>
<td>2,4,123</td>
<td>2,4,123</td>
<td>1,2,3</td>
</tr>
</tbody>
</table>

Table 8: Training hyper-parameters for GLUE. “BSZ” represents “Batch size”, “LR” represents “Learning rate” and “Warm” represents “Warmup steps”.

References


Conneau, A.; and Lample, G. 2019. Cross-lingual language...
Scaling vision with sparse mixture of experts. Advances in Neural Information Processing Systems, 34.


