Gating Dropout: Communication-efficient Regularization for Sparsely Activated Transformers

Rui Liu [*1] **Young Jin Kim** [2] **Alexandre Muzio** [2] **Hany Hassan Awadalla** [2]

Abstract

Sparsely activated transformers, such as Mixture of Experts (MoE), have received great interest due to their outrageous scaling capability which enables dramatical increases in model size without significant increases in computational cost. To achieve this, MoE models replace the feedforward sub-layer with Mixture-of-Experts sub-layer in transformers and use a gating network to route each token to its assigned experts. Since the common practice for efficient training of such models requires distributing experts and tokens across different machines, this routing strategy often incurs huge cross-machine communication cost because tokens and their assigned experts likely reside in different machines. In this paper, we propose *Gating Dropout*, which allows tokens to ignore the gating network and stay at their local machines, thus reducing the cross-machine communication. Similar to traditional dropout, we also show that Gating Dropout has a regularization effect during training, resulting in improved generalization performance. We validate the effectiveness of Gating Dropout on multilingual machine translation tasks. Our results demonstrate that Gating Dropout improves a state-of-the-art MoE model (Kim et al., 2021) with faster wall-clock time convergence rates and better BLEU scores for a variety of model sizes and datasets.

1. Introduction

Large transformer models have been shown to be effective and powerful in many natural language processing tasks such as machine translation (Lewis et al., 2019; Conneau & Lample, 2019) and also language understanding (Devlin et al., 2018; Liu et al., 2019; Brown et al., 2020; Radford et al., 2019). Sustaining the continued growth in size for these dense models requires tremendous compute resources, since all the parameters are used for all input examples. Seeking better computational efficiency, sparsely activated models have gained great interest; since they utilize a subset of the model weights for each input example (Shazeer et al., 2017; Lepikhin et al., 2020). This leads to a desirable characteristic: sparsely activated models can scale up to an order of magnitude larger than dense models without significant increase in computational cost. However, libraries and hardware accelerators widely used by machine learning practitioners are much efficient in supporting dense matrix operations. Mixture of Experts (MoE) models are a specific type of sparsely activated models that have had notable success in machine translation (Shazeer et al., 2018; Lepikhin et al., 2020) and recently been explored in other tasks (Riquelme et al., 2021) while running efficiently on top of existing libraries and hardware accelerators. In this paper, we focus on multilingual machine translation tasks.

A dense transformer model consists of a stack of layers composed of two sub-layers: a multi-head self-attention sub-layer followed by a feedforward network (FFN) sub-layer. An MoE model replaces the FFN sub-layer with a Mixture-of-Experts (MoE) sub-layer which takes as input a token representation and routes this token to the best top-k experts. The MoE sub-layer is essentially a group of experts with each expert still a FFN sub-layer. Typically, k is a very small value. As with (Fedus et al., 2021; Kim et al., 2021), we assume $k = 1$ by default. Note that our method can also be extended to the case when $k > 1$. During training and inference, an input token is routed to only one expert ($k = 1$), resulting in constant computational cost for one forward pass regardless of the total number of experts. The target expert for each token is determined by a gating network which is typically a small one-layer FFN.

To achieve highly efficient training, a combination of data, expert, and model parallelism is adopted: (1) the part of the model excluding MoE sub-layers is replicated across multiple machines in a data-parallel fashion, (2) experts from each MoE sub-layer are split across machines (i.e.,

[*]Work done while a research intern at Microsoft [1]University of Michigan, Ann Arbor [2]Microsoft. Correspondence to: Rui Liu <ruixliu@umich.edu>, Young Jin Kim <youki@microsoft.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

different machines hold the parameters of different experts), and (3) techniques such as tensor slicing[1] are used to partition tensors among multiple machines if they are too large for a single machine (e.g., partition along the d_{ff} dimension inside each expert when d_{ff} is too large) (Fedus et al., 2021). Since tokens and their assigned experts are likely to reside at different machines, we need to shuffle the tokens around via the communication link among machines. This is typically achieved by using the all-to-all collective operation, which is very costly even if the data is cast to bfloat16 precision (Fedus et al., 2021). If we assume that token representation is of d dimension, the sequence length is L, total batch size is B, then the total amount of data that needs to be handled by the all-to-all operation is $2BLd$ bytes[2]. As a commonly used setting in practice, we assume $d = 4096 = 2^{12}, L = 1024 = 2^{10}, B = 128 = 2^7$, then $2BLd = 2^{1+12+10+7} = 2^{30} = 1G$. In other words, the all-to-all operation needs to shuffle roughly $1GB$ of data at each MoE sub-layer for each forward pass. Therefore, the communication cost from all-to-all operations is greatly affecting the training speed for MoE models.

In this paper, we propose *Gating Dropout* as a communication-efficient regularization technique for training sparsely activated transformers, specifically for MoE models. Gating Dropout allows tokens to ignore the assignments from the gating network and stay at their local machines with certain probability, thus reducing the need for routing the tokens to experts at different machines. This significantly improves the throughput and convergence speed for MoE training. In addition, by forcing tokens to be routed to experts on the same machines (which might not be the optimal experts), we encourage the experts to learn a general ability. We think this leads to a regularization effect during training, resulting in better generalization performance when training is converged.

To validate the effectiveness of Gating Dropout, we have conducted extensive experiments on multilingual machine translation tasks using different model settings. With Gating Dropout, our results show that we improve a state-of-the-art MoE model (Kim et al., 2021) in terms of convergence speed (up to 58% less training time to target BLEU score), generalization performance (up to 0.6 BLEU score increase) and throughput (up to 16% more tokens per second). Our code will be available at `https://aka.ms/gating_dropout`.

[1]To understand the key idea of Gating Dropout, readers can focus on data and expert parallelism (i.e., ignore model parallelism such as tensor slicing) to simplify the complexity of parallelism.

[2]The factor 2 comes from the bfloat16 data representation, which uses 2 bytes per number.

2. Background and Motivation

2.1. From Densely Activated Transformers to Sparsely Activated Transformers

In Vaswani et al. (2017), the transformer architecture was introduced to the NLP community due to their superior performance in sequence-to-sequence tasks, such as neural machine translation. A sequence-to-sequence transformer contains an encoder module and a decoder module. The encoder module is a stack of encoder layers, each employing a self-attention sub-layer followed by an FFN sub-layer. The decoder module is a stack of decoder layers, each employing a self-attention sub-layer followed by a cross-attention sub-layer with an FFN sub-layer at the end. Ever since, transformer-based models have been the top performers in various NLP tasks, such as BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), GPT-3 (Brown et al., 2020), XLNet (Yang et al., 2019), ELECTRA (Clark et al., 2020) and TMR (Liu & Mozafari, 2022). However, these transformer models are densely activated, meaning that all model parameters are used to process all input examples. Scaling up the model size for them results in significant increase in computational cost during both training and inference. Therefore, the level of scaling depends on the availability of compute resources (e.g., GPU or TPU), which can be costly to obtain. For example, it is estimated that it takes 168 days to train a GPT-3 model with 178 billion parameters using 256 NVIDIA A100 GPUs (Narayanan et al., 2021). Megatron-Turing NLG 530B model with 530 billion parameters had been trained on NVIDIA's Selene supercomputer with 560 DGX A100 nodes, each node having 8 NVIDIA 80GB A100 GPUs connected to each other with high bandwidth Infiniband configuration (Smith et al., 2022).

On the other hand, sparsely activated transformers use a subset of model parameters for each different input example, leading to outrageous scaling capability. An Mixture-of-Experts (MoE) (Fedus et al., 2021) model has been scaled to 1.5 trillion parameters while GPT-3, one of the largest dense models, contains only 175 billion parameters. The core component of MoE models is the Mixture-of-Experts (MoE) sub-layer, which replaces the FFN sub-layer in the dense transformers, as illustrated in Figure 1. Let $\{E_i\}_{i=1}^N$ denote the set of experts at an MoE sub-layer where N is the total number of experts. Each expert E_i is still a feedforward network (FFN). The parameters of one expert is independent from those of another expert, meaning that they can be trained to converge to different values. Each token is routed to the top-k experts, which are determined by a gating network. Given a token $x \in \mathbb{R}^d$, where d is the hidden dimension, the output of a gating network is logits $h(x) = W_r \cdot x$, followed by a normalization via a softmax,

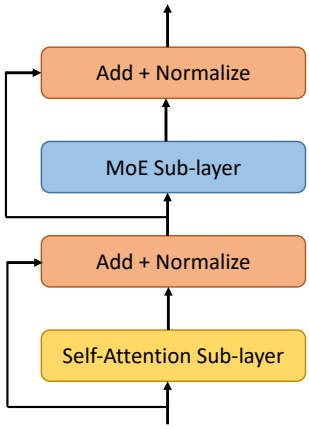

Figure 1. The MoE model architecture. The FFN sub-layer in the original transformer architecture is replaced by the MoE sub-layer, which consists of experts.

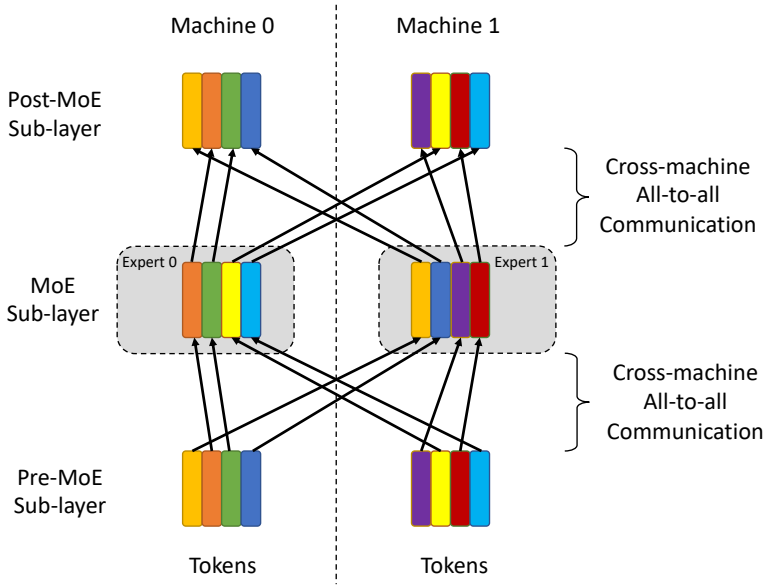

Figure 2. The all-to-all communication illustration of an MoE model with two experts in a two-machine setting. Expert 0 and 1 reside at machine 0 and 1, respectively. Each machine has a local batch of 4 tokens to process. All-to-all communication is needed right before (and right after) MoE sub-layer.

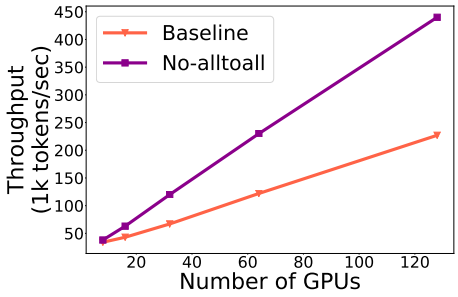

Figure 3. The all-to-all communication cost.

Table 1. Relative throughput improvement of no-alltoall compared to baseline.

Number of GPUs	Throughput Impr.
8	11.8%
16	46.5%
32	79.1%
64	88.5%
128	93.8%

i.e.,

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_{j=1}^{N} e^{h(x)_j}}, \tag{1}$$

where $W_r \in \mathbb{R}^{N \times d}$ is the trainable weight matrix of the gating network. Given the gating network output $\{p_i(x)\}_{i=1}^N$, we select the top-k experts to form the set of activated experts $\mathcal{T} \subset \{1, \cdots, N\}$, where $|\mathcal{T}| = k$. In other words, the token x is assigned to experts in $\mathcal{T}$. Typically k is much smaller than N. In Fedus et al. (2021), it is shown that using $k = 1$ is sufficient with better computational efficiency. In this paper, we assume $k = 1$ by default. Then the output of an MoE sub-layer for the input token x is

$$y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x). \tag{2}$$

2.2. All-to-all Communication Cost of MoE Models

Distributed training involving multiple machines is typically used to support highly scaled MoE models. Each machine contains a replication of the model parameters excluding experts (i.e., data parallelism), and a subset of experts (i.e., expert parallelism). Since experts are split across multiple machines, each token needs to be sent to the machine where its assigned expert resides. This is typically achieved by using the all-to-all collective communication (Fedus et al., 2021), which is illustrated in Figure 2 using a two-machine setting. We conducted some experiments to demonstrate the all-to-all communication cost (see the settings on WMT-10 dataset in Section 4.1 for details). Specifically, we use a cluster of NVIDIA V100 GPUs connected via a 100Gb/s InfiniBand fabric. We use a state-of-the-art MoE model (Kim et al., 2021) as the baseline, and a no-alltoall variant which is the same as baseline but with the all-to-all operations entirely skipped. We compare the throughput in terms of the number of tokens processed per second between the base-

line and the no-alltoall variant. To see how the throughput changes as we scale up the model, we increase the number of GPUs from 8 all the way to 128, and set the number of experts the same as the number of GPUs due to expert parallelism (Fedus et al., 2021). From Figure 3, we can see that as we increase the number of GPUs, the throughput for both the baseline model and no-alltoall variant increases. This is expected because using more GPUs provides greater computational power which increase the processing speed. We also observe that the no-alltoall variant has significantly higher throughput than the baseline when using the same number of GPUs. In Table 1, we list the relative throughput improvement of the no-alltoall compared to the baseline. As we increase the number of GPUs, the relative throughput improvement also increases. Especially, when the number of GPUs is 128, the relative improvement is more than 90%. Note that the difference between the baseline and no-alltoall comes from whether the all-to-all communication in the MoE sub-layers is invoked or not. From these results, we can see that all-to-all communication greatly affects the training speed. As the number of GPUs increases, the all-to-all communication becomes more costly because data exchange involves more machines and communication cost is proportional to the number of involved machines. In summary, to better support the scaling capability of MoE models, it will pay off to investigate methods that can reduce the all-to-all communication cost without sacrificing the performance.

3. Gating Dropout as Communication-efficient Regularization

In this section, we describe *Gating Dropout* as an effective way to reduce the all-to-all communication cost. We also show that Gating Dropout has a regularization effect during training, resulting in improved generalization performance. Gating Dropout essentially combines communication efficiency with regularization, which has been challenging to achieve yet desirable in distributed training, where communication cost is often the bottleneck. Gating Dropout works in the following way, which is also illustrated in Figure 4, at each training iteration:

- with probability p, all the tokens are sent to the local experts at the same machine without all-to-all communication involved;
- with probability $1-p$, all the tokens are sent to their assigned experts with all-to-all communication involved, as is done in normal MoE models.

The above probability p is the dropout rate. Note that, since all-to-all is a collective operation requiring all the machines to simultaneously participate, the decision of turning on/off Gating Dropout should be consensual among all the machines at each iteration. In other words, at each iteration, it is either (1) that all the machines keep their tokens locally (i.e., Gating Dropout is turned on), or (2) that all the machines use the all-to-all operations to send tokens to their assigned experts (i.e., Gating Dropout is turned off). To make sure all machines achieve consensual decisions, we appoint one machine as the coordinator, responsible for making the randomized decision, and broadcasting the decision to all the machines at each iteration. The overhead of broadcasting the decision is negligible, because the decision can be represented by a binary value.

During inference, we simply turn off the Gating Dropout by setting $p = 0$. We do not have the additional weight scaling step as in the traditional dropout (i.e., multiply the weight by p). In the traditional dropout, the weight scaling step is used to ensure the expected neuron output is the same between inference time and training time, because traditional dropout works by setting the neuron output as 0. However, Gating Dropout will not affect expected neuron output because it works by modifying the token routing strategy.

This technique can be easily implemented with several lines of code, by using the broadcast collective operation for broadcasting the decision and the conditional branch for skipping the all-to-all communication. We refer to this basic version of Gating Dropout as Gate-Drop in order to differentiate it from the extended version that will be introduced in Section 3.1.

The benefits of Gating Dropout are twofold:

- higher throughput due to reduced communication cost;
- better generalization performance due to its regularization effect.

Communication Efficiency. Gating Dropout reduces the all-to-all communication cost, thus improving the throughput during training. When the Gating Dropout is turned on at a certain iteration, all the tokens go to the local experts at the same machines. In this case, all-to-all communication is no longer needed at this iteration. However, when Gating Dropout is turned off, the all-to-all communication is still used. The dropout rate p essentially controls how often the gating dropout is turned-on on average, thus affecting the throughput during the training process. The no-alltoall variant in Figure 3 corresponds to the case when $p = 1$, which provides the upper-bound of the throughput improvement. It is worth noting that it is not possible to achieve this upper-bound since the model will not be able to learn any gating in such settings. Therefore, the model performance will deteriorate when p is too large. Next, we will show that Gating Dropout can lead to better performance when p is properly chosen.

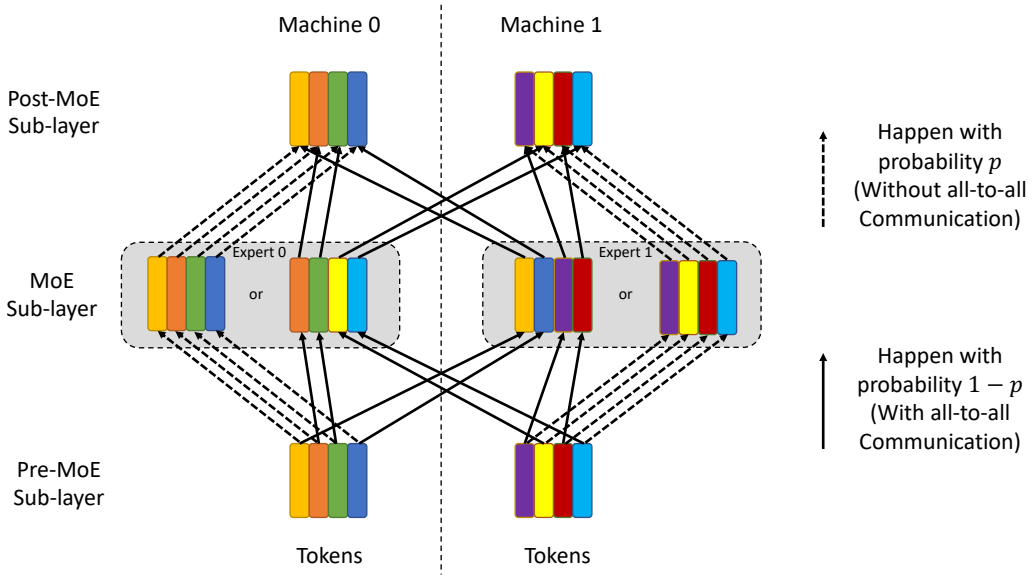

Figure 4. The Gating Dropout illustration of an MoE model with two experts in a two-machine setting. Expert 0 and 1 reside at machine 0 and 1, respectively. Each machine has a local batch of 4 tokens to process. Gating Dropout allows us to send tokens to the local experts at the same machines with probability p (denoted by the dashed arrows).

Regularization. Gating Dropout can be viewed as a regularization technique to improve the model's generalization performance. The gating network is used to determine which expert a token should be sent to. The gating network is supposed to select the best expert for each token, which is unknown in general. Therefore, the common practice in MoE models is that the gating network is initialized randomly, and being trained together with all the experts. Intuitively, Gating Dropout can break up the co-training between gating network and experts, just as traditional dropout is used to break up the co-adaptation among neurons (Srivastava et al., 2014). This forces experts to work in a robust way without relying too much on the guidance from the gating network (e.g., by obtaining a general ability). Another perspective that affects the dynamics of training with Gating Dropout comes from exploration-exploitation trade-off (Auer et al., 2002; Sutton & Barto, 2018). The gating network provides a discrete decision over all the experts. This decision may not be optimal simply because other experts could be under-explored. The epsilon-greedy method (Sutton & Barto, 2018) is a simple yet effective way to handle the exploration-exploitaion trade-off: with probability ϵ, make a random decision over all candidate options; with probability $1-\epsilon$, make the greedily best decision based on the available information so far. Gating Dropout can be viewed as a simplified variant of the epsilon-greedy method that is tailored for MoE model training while accounting for the communication cost. Specifically, instead of choosing a random expert, we choose the local expert at the same machine because the all-to-all communication is not needed to send tokens to their local experts. In our experiments, it is observed that this simplified variant of the epsilon-greedy method actually yields good performance improvement.

Gating Dropout vs Expert Dropout. It is worth noting the differences between Gating and Expert dropout. Expert Dropout is recommended in Fedus et al. (2021) as a technique to regularize the MoE models. However, both techniques have quite distinctive characteristics. Expert Dropout is used during fine-tuning on downstream tasks that have very few examples to overcome possible overfitting due to lack of enough examples. However, Gating Dropout is used in pre-training. Furthermore, Expert Dropout is essentially traditional dropout (Srivastava et al., 2014) where the dropout rate inside MoE sub-layers is explicitly increased. They observed better performance on several smaller downstream tasks by setting a smaller dropout rate at non-MoE sub-layers and a higher rate at MoE sub-layers. On the other hand, Gating Dropout acts as a regularization technique that is completely different from traditional dropout.

Gating Dropout vs Other Exploration-Exploitation Techniques. The exploration-exploitation trade-off is also considered in Fedus et al. (2021). They investigated several techniques such as sample softmax, input dropout and input jitter. First, none of these techniques is communication efficient, which is one of the main benefits of our Gating Dropout. Second, Gating Dropout is compatible with these techniques, so that they can be complimentary used together during training. For example, input jitter, which has the best empirical performance in their experiments, injects noise to the incoming token representation. Gating Dropout can

still be applied regardless of how the token representation is modified. In our experiments, we have adopted input jitter by default in the baseline.

3.1. Extension with Expert LayerDrop

It has been shown that skipping certain layers of neural networks during training can reduce the training time and also has a regularization effect (Huang et al., 2016; Fan et al., 2019). Inspired by LayerDrop proposed in Fan et al. (2019) which randomly drops layers in (densely activated) transformers, we consider an extension of Gating Dropout by dropping experts. Specifically, when Gating Dropout is turned on, all the tokens skip the local experts and directly go to the sub-layer following the MoE sub-layer. When the Gating Dropout is off, we still send the tokens to their assigned experts with all-to-all communication. We call this extended version Gate-Expert-Drop, while the basic version from the previous section is referred to as Gate-Drop. This extension improves the two strengths of Gating Dropout, since (1) skipping experts reduces the computational cost, thus improving the throughput, and (2) skipping layers randomly has been shown with a regularization effect.

4. Experiments

We evaluate our proposed Gating Dropout (both Gate-Drop and Gate-Expert-Drop) and compare it against a state-of-the-art MoE model (Kim et al., 2021) on multilingual translation tasks. Multilingual large-scale translation tasks have been a good use case for large dense and sparse transformer models due to their high level of complexity that necessitates such powerful models. We use the Z-code M3 model from Kim et al. (2021) as our **baseline**, which has been shown better than prior MoE models. All of our implementations are based on DeepSpeed library[3], due to its strong support for distributed training. We show that Gating Dropout can improve the throughput, BLEU score and convergence speed.

4.1. Setup

Datasets. Similar to prior work (Kim et al., 2021), we use the following datasets in our experiments.

- **WMT-10:** We use WMT-10 benchmark dataset which includes bitext data between English and other **10** languages (Wang et al., 2020). There are 32.5 million parallel sentences in total.
- **Web-50:** We use a dataset composed of an in-house **web**-crawled parallel dataset augmented with data from CCNet (Wenzek et al., 2019). It contains 700 million parallel sentences in **50** different languages. This dataset is much bigger than the WMT-10 dataset.

[3] https://github.com/microsoft/DeepSpeed

Model Settings. We use the settings recommended in (Fedus et al., 2021) to set up the MoE models. Specifically, MoE sub-layers are added to replace every other FFN sub-layers in both the encoder and decoder of the transformer. The capacity factor is set to 1.0 during training and 2.0 during testing (Fedus et al., 2021). Jittering noise is applied to the token representation right before the gating network. An additional balancing loss with multiplicative coefficient 0.01 is used to better balance the utilization among different experts. We use two different model sizes for experiments on the above two datasets.

- **WMT-10:** We adopt the architectural settings according to Transformer-base architecture with 12 encoder layers and 6 decoder layers (Vaswani et al., 2017). For each MoE sub-layer, we use 128 experts by default. The model contains around 5.6 billion parameters.
- **Web-50:** We adopt the architectural settings according to Transformer-big architecture with 24 encoder layers and 12 decoder layers (Vaswani et al., 2017). For each MoE sub-layer, we use 64 experts by default. The model contains around 10 billion parameters.

Hardware. We use a cluster of NVIDIA V100 GPUs connected via a 100Gb/s InifiBand fabric in most experiments. For some experiments on the Web-50 dataset, we run jobs on a cluster of NVIDIA A100 GPUs connected via a 1.6Tb/s InifiBand fabric. We use 16 GPUs for WMT-10 and 64 GPUs for Web-50 if not stated otherwise.

Training Details. For training, we use Adam as the optimizer with $\beta_1 = 0.9$ and $\beta = 0.99$. The learning rate is set 0.03 with 5000 warm-up steps and inverse square root scheduler as proposed in Raffel et al. (2019). We set the batch size to be equivalent to 435k tokens. For the gating dropout rate p, we set it to 0.3 for Gate-Drop and 0.2 for Gate-Expert-Drop by default, which are chosen because of their good performance (see Section 4.4). For training on the Web-50 dataset, we use Denoising Auto-Encoding (DAE) loss (Vincent et al., 2008) together with Machine Translation (MT) loss because multi-tasking learning has been shown to further improve the cross-language knowledge transfer (Wang et al., 2020; Kim et al., 2021).

Metrics. We compare between different methods using the following three metrics:

- **Throughput:** the number of tokens processed per second by the model during training.
- **Generalization Performance:** BLEU score on a hold-out validation set. Especially, we report the case-sensitive detokenized sacreBLEU (Post, 2018).

Table 2. The results on WMT-10 dataset using models with Transformer-base architecture. The target BLEU score is the value achieved by the baseline method at convergence, i.e., 24.60. The training time to the target BLEU is in seconds. Hash-Layer uses a random hash function to replace the gating network (Roller et al., 2021). We use - in the table to indicate that Hash-Layer cannot reach the target BLEU.

Method	Throughput	BLEU@convergence	Time to target BLEU	Steps to target BLEU
Baseline	129k	24.60	250k	74.24k
Hash-Layer	135k	23.06	-	-
Gate-Drop	143k	25.06	147k	48.64k
Gate-Expert-Drop	**150k**	**25.12**	**104k**	**35.84k**

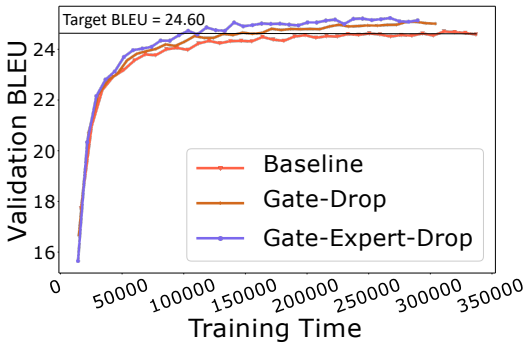

Figure 5. BLEU score vs. training time (in seconds) on the WMT-10 dataset. Both Gate-Drop and Gate-Expert-Drop reach the target BLEU (denoted as the black line) much faster than the baseline.

Table 3. The throughput results on Web-50 dataset measured in two different clusters.

Method	V100 Cluster	A100 Cluster
Baseline	126k	362k
Gate-Drop	140k	372k
Gate-Expert-Drop	146k	384k

- **Convergence Speed:** the amount of training time to reach a target BLEU score on a holdout validation set.

4.2. Results on WMT-10 Dataset

We train MoE models based on the Transformer-base architecture (Vaswani et al., 2017) on the WMT-10 dataset for more than 100k steps using different methods. The results are shown in Table 2. We see that both Gate-Drop and Gate-Expert-Drop have higher throughput than the baseline method with relative improvement 10.85% and 16.28% respectively. Gate-Expert-Drop has higher throughput than Gate-Drop because computational cost is further reduced by skipping experts.

As for the generalization performance, both Gate-Drop and Gate-Expert-Drop reach BLEU scores that are greater than that of the baseline when the training is converged. The BLEU score improvement is 0.46 for Gate-Drop and 0.52 for Gate-Expert-Drop. The slightly better performance of Gate-Expert-Drop is likely to come from the additional regularization effect of randomly skipping expert layers. From Table 2, we also observe that Gate-Drop and Gate-Expert-Drop have faster convergence speed than the baseline. We choose the target BLEU score as the score achieved by the baseline method at convergence (i.e., 24.60). The amount of training time to reach the target BLEU score is also significantly shorter for both Gate-Drop (by 40.89%) and Gate-Expert-Drop (by 58.48%) compared to the baseline. This can also be observed in Figure 5, where we show the curves of BLEU score vs. training time. Similarly, both Gate-Drop and Gate-Expert-Drop require smaller number of training steps to reach the target BLEU score compared to the baseline. These results confirm that our methods' fast convergence speed in terms of both the training time and the number of steps.

In addition, Roller et al. (2021) suggest replacing the parameterized gating network with some hash function, which can alleviate the co-training issue between the gating network and experts. As shown in Table 2, we also compare against their method Hash-Layer based on a random hash function which is simple to implement but has been shown to have very strong performance (Roller et al., 2021). Hash-Layer has slightly better throughput than the baseline, which is expected because invoking a hash function is more computational efficient than passing tokens through a gating network. However, Hash-Layer still needs the all-to-all communication to send tokens to their assigned experts (which are determined by the hash function), which explains why both Gate-Drop and Gate-Expert-Drop still achieve higher throughput than Hash-Layer. We also observe that Hash-Layer does not perform as well as our methods in terms of final BLEU at convergence.

4.3. Results on Web-50 Dataset

We train MoE models based on the Transformer-big architecture (Vaswani et al., 2017) with 10 billion parameters on the Web-50 dataset for up to 20k steps, it is very time consuming to train such large models for too many steps. In Table 3, we report the throughput of different methods measure in two different clusters: the cluster of V100 GPUs with

Table 4. The results on Web-50 dataset using models with Transformer-big architecture. **BLEU (avg)** is the average BLEU score for translating between English and other languages (both directions). **E→X** only considers translating from English to other languages. **X→E** only considers translating from other languages to English. And **(low)** is used to indicate that only low-resource languages are considered.

Method	BLEU (avg)	E→X	E→X (low)	X→E	X→E (low)
Baseline	28.63	23.01	22.15	34.26	33.89
Gate-Drop	**29.22**	**23.86**	**22.87**	**34.59**	**34.34**
Gate-Expert-Drop	28.85	23.22	22.61	34.49	34.22

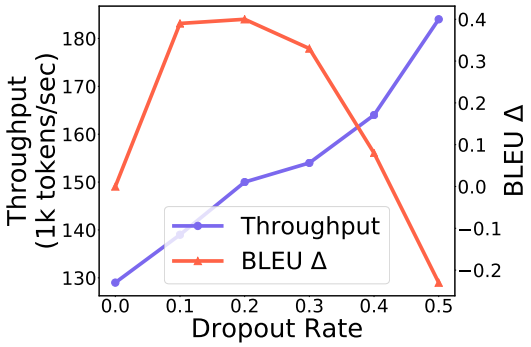

Figure 6. The effect of the dropout rate on throughput and BLEU Δ (i.e., difference w.r.t. the baseline's BLEU) for Gate-Expert-Drop on the WMT-10 dataset (measured after training for 35k steps).

a 100Gb/s InfiniBand fabric and the cluster of A100 GPUs with a 1.6Tb/s InfiniBand fabric. We observe that both Gate-Drop and Gate-Expert-Drop get higher throughput than the baseline in either cluster. It is noted that the relative throughput improvement of our methods in the V100 cluster is more significant because the hardware (GPU for computation and InfiniBand for communication) is much slower than the A100 cluster. Higher improvement is expected with slower hardware where communication is the bottleneck.

We report the best BLEU scores within 20k steps in Table 4. We can see that both Gate-Drop and Gate-Expert-Drop achieve larger BLEU scores than the baseline with the improvement 0.59 for Gate-Drop and 0.22 for Gate-Expert-Drop. Different from the results on the WMT-10 dataset (which is a much smaller dataset), Gate-Drop is better than Gate-Expert-Drop in terms of BLEU scores. We hypothesize that this is probably because Gate-Expert-Drop imposes too much regularization during training, limiting the model's fitting ability on such large scale dataset. We further investigate the results more by looking into the BLEU score for each translation direction, i.e., from English to other languages (E→X) and from other languages to English (X→E). Especially, we consider the performance on only low-resource languages, denoted with **(low)** in Table 4. Low-resource languages are those with few training data, such as Galician (Gl), Burmese (My), and Tagalog (Tl). It is difficult to achieve high quality translation performance on these low-resource languages. We see that both Gate-Drop and Gate-Expert-Drop improve upon the baseline on the low-resource language translation tasks by at least 0.3 increase in BLEU score (from Gate-Expert-Drop on **X→E (low)**). And Gate-Drop can yield more than 0.7 BLEU score increase on **E→X (low)**). These results suggest that our methods improve the model's translation quality, especially on low-resource languages which would really benefit from the regularization effect of Gating Dropout.

4.4. Effect of Dropout Rate

We investigate the effect of the dropout rate p of Gate-Expert-Drop on throughput and BLEU score using the WMT-10 dataset. Similar observations are expected for Gate-Drop, but we only report results for Gate-Expert-Drop because it outperforms Gate-Drop on the this dataset. We change the dropout rate from 0 to 0.5 for Gate-Expert-Drop. When dropout rate is 0, it is the same as the baseline method, because Gating Dropout will never be turned on in this case. From Figure 6, we see that, as we increase the dropout rate, the throughput also increases. This trend is expected because the dropout rate essentially controls how often the all-to-all communication (and the expert layer) is skipped during training. Larger dropout rate implies more skipping, thus resulting in higher throughput. However, when the dropout rate is too large, the BLEU score might become worse, as shown in Figure 6. Specifically, when the dropout rate is 0.5, BLEU score becomes even lower than the baseline (BLEU Δ is −0.23). Actually, the BLEU score is highest when the dropout rate is 0.2, and becomes smaller and smaller as we further increase the rate beyond 0.2. Similar experiments indicate that Gate-Drop achieves highest BLEU score when dropout rate is 0.3. This indicates that the dropout rate needs to be properly chosen in order for our methods to work as expected.

5. Related Work

Several techniques have been proposed to improve the efficacy of MoE models. A new routing network which takes as input both the shared token representation and the hier-

archical representation from different MoE layers is proposed in You et al. (2021) for speech recognition. Expert prototyping proposed in Yang et al. (2021) is shown to improve model quality by splitting experts into different prototypes, while Zuo et al. (2021) suggest to randomly select experts to encourage learning from other experts as teachers. Linear assignment is used to alleviate the load imbalance issue among experts (Lewis et al., 2021). As an effective way to regularize model training, the dropout idea has been widely studied for transformer models in general (Arora et al., 2021; Gao et al., 2019; Kingma et al., 2015; Boluki et al., 2020). Several dropout variants have been proposed in different settings, such as BPE-Dropout (Provilkov et al., 2019), R-Drop (Wu et al., 2021), MC-Dropout (Gal & Ghahramani, 2016), recurrent dropout (Semeniuta et al., 2016), curriculum dropout (Morerio et al., 2017) and so on. Unlike these dropout variants, our method considers the communication cost which is crucial in the distributed training setting.

6. Conclusion

We propose Gating Dropout as a communication-efficient regularization technique to train sparsely activated transformers. We observe that sparsely activated transformers, such as MoE models, typically have very high cross-machine communication cost, since they need to send tokens to their assigned experts via the all-to-all communication operations. Gating Dropout reduces the communication cost by randomly skipping the all-to-all operations. This random skipping also has a regularization effect during training, leading to improved generalization performance. Experiments on multilingual translation tasks demonstrate the effectiveness of the proposed method in terms of throughput, generalization performance and convergence speed.

As for future work, we are looking at how to improve the inference speed by possibly combining Gating Dropout with expert pruning. Gating Dropout currently has no effect on the inference speed, since it is simply turned off during inference. In addition, we are also interested in the effect of varying dropout rate throughout the training process because exploration might be much more important at the early stage of training from the exploration-exploitation perspective.

Acknowledgements

As part of Microsoft's effort led by Azure AI and Project Turing to build powerful large-scale language models, Gating Dropout has been utilized to train Z-Code MoE models which are adopted by Microsoft Translator[4] to provide the multilingual translation service to customers. Especially, the authors would like to thank the Z-Code team and DeepSpeed team at Microsoft for their support and feedback.

[4] https://translator.microsoft.com/

References

Arora, R., Bartlett, P., Mianjy, P., and Srebro, N. Dropout: Explicit forms and capacity control. In *International Conference on Machine Learning*, pp. 351–361. PMLR, 2021.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Boluki, S., Ardywibowo, R., Dadaneh, S. Z., Zhou, M., and Qian, X. Learnable bernoulli dropout for bayesian deep learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3905–3916. PMLR, 2020.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Conneau, A. and Lample, G. Cross-lingual language model pretraining. *Advances in Neural Information Processing Systems*, 32:7059–7069, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Gao, H., Pei, J., and Huang, H. Demystifying dropout. In *International Conference on Machine Learning*, pp. 2112–2121. PMLR, 2019.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Kim, Y. J., Awan, A. A., Muzio, A., Salinas, A. F. C., Lu, L., Hendy, A., Rajbhandari, S., He, Y., and Awadalla, H. H. Scalable and efficient moe training for multitask multilingual models. *arXiv preprint arXiv:2109.10465*, 2021.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28:2575–2583, 2015.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. *arXiv preprint arXiv:2103.16716*, 2021.

Liu, R. and Mozafari, B. Transformer with memory replay. *arXiv preprint arXiv:2205.09869*, 2022.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Morerio, P., Cavazza, J., Volpi, R., Vidal, R., and Murino, V. Curriculum dropout. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3544–3552, 2017.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters. *arXiv preprint arXiv:2104.04473*, 2021.

Post, M. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

Provilkov, I., Emelianenko, D., and Voita, E. Bpe-dropout: Simple and effective subword regularization. *arXiv preprint arXiv:1910.13267*, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Pinto, A. S., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2106.05974*, 2021.

Roller, S., Sukhbaatar, S., Szlam, A., and Weston, J. Hash layers for large sparse models. *arXiv preprint arXiv:2106.04426*, 2021.

Semeniuta, S., Severyn, A., and Barth, E. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-tensorflow: Deep learning for supercomputers. *arXiv preprint arXiv:1811.02084*, 2018.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., Zheng, E., Child, R., Aminabadi, R. Y., Bernauer, J., Song, X., Shoeybi, M., He, Y., Houston, M., Tiwary, S., and Catanzaro, B. Using deepspeed and megatron to train megatron-turing NLG 530b, A large-scale generative language model. *CoRR*, abs/2201.11990, 2022. URL https://arxiv.org/abs/2201.11990.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

Wang, Y., Zhai, C., and Awadalla, H. H. Multi-task learning for multilingual neural machine translation. *arXiv preprint arXiv:2010.02523*, 2020.

Wenzek, G., Lachaux, M.-A., Conneau, A., Chaudhary, V., Guzmán, F., Joulin, A., and Grave, E. Ccnet: Extracting high quality monolingual datasets from web crawl data. *arXiv preprint arXiv:1911.00359*, 2019.

Wu, L., Li, J., Wang, Y., Meng, Q., Qin, T., Chen, W., Zhang, M., Liu, T.-Y., et al. R-drop: regularized dropout for neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Yang, A., Lin, J., Men, R., Zhou, C., Jiang, L., Jia, X., Wang, A., Zhang, J., Wang, J., Li, Y., et al. Exploring sparse expert models and beyond. *arXiv preprint arXiv:2105.15082*, 2021.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

You, Z., Feng, S., Su, D., and Yu, D. Speechmoe: Scaling to large acoustic models with dynamic routing mixture of experts. *arXiv preprint arXiv:2105.03036*, 2021.

Zuo, S., Liu, X., Jiao, J., Kim, Y. J., Hassan, H., Zhang, R., Zhao, T., and Gao, J. Taming sparsely activated transformer with stochastic experts. *arXiv preprint arXiv:2110.04260*, 2021.