Sparse Mixture of Experts for Modularity

Yufang Liu

East China Normal University

ModuleFormer: Learning Modular Large Language Models From Uncurated Data

Yikang Shen[*]
MIT-IBM Watson AI Lab

Zheyu Zhang
Tsinghua University

Tianyou Cao
Tsinghua University

Shawn Tan
Mila/University of Montreal

Zhenfang Chen
MIT-IBM Watson AI Lab

Chuang Gan
MIT-IBM Watson AI Lab

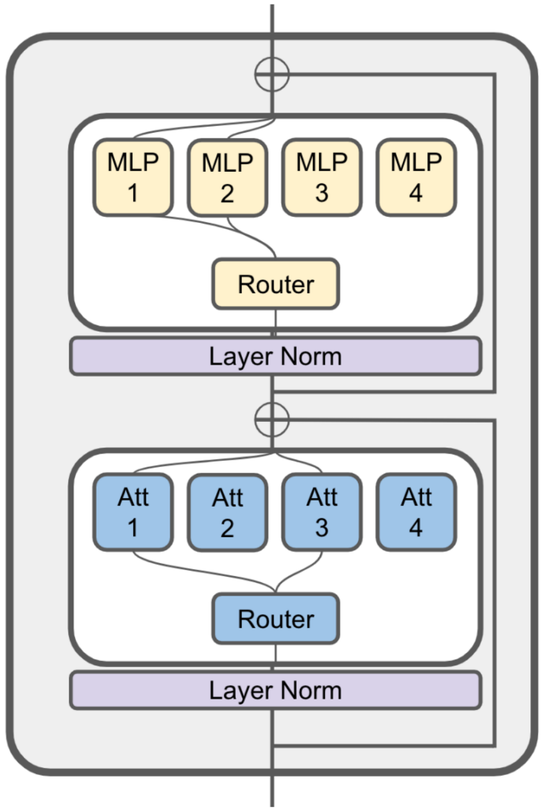
Arxiv 2023

Mixture-of-Experts layer

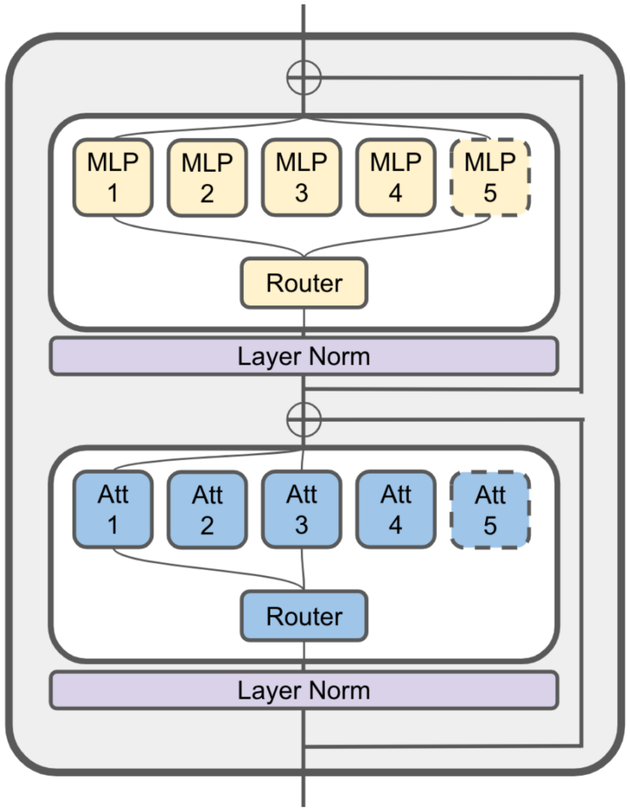
An MoE layer includes

- experts (FFN)
- trainable gating function mapping a subset of `best' experts for each input

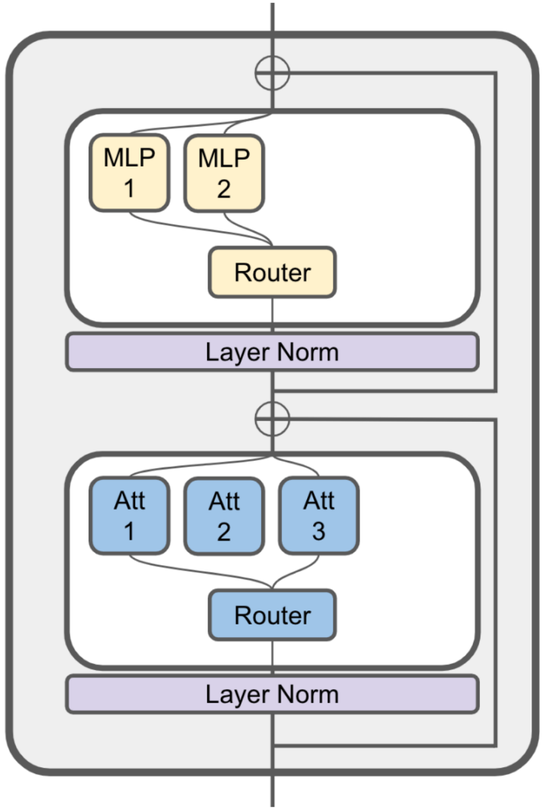
Mixture-of-Experts layer



(a) Sparse activation of modules



(b) Adding new modules



(c) Pruning modules

Methods

Mixture of Experts

$$g(m \mid \mathbf{x}) = \mathrm{Top}k\left(\frac{e^{h(\mathbf{x})_i}}{\sum e^{h(\mathbf{x})_j}}\right),$$

$$h(\mathbf{x}) = \mathbf{A}\,\mathrm{ReLU}(\mathbf{B}\mathbf{x}),$$

$$y = \sum_{m=1}^{N} g(m \mid \mathbf{x}) \cdot f_m(\mathbf{x})$$

FFN -> SMoE, MHA-> MoA

Methods

Mixture of Attention Heads

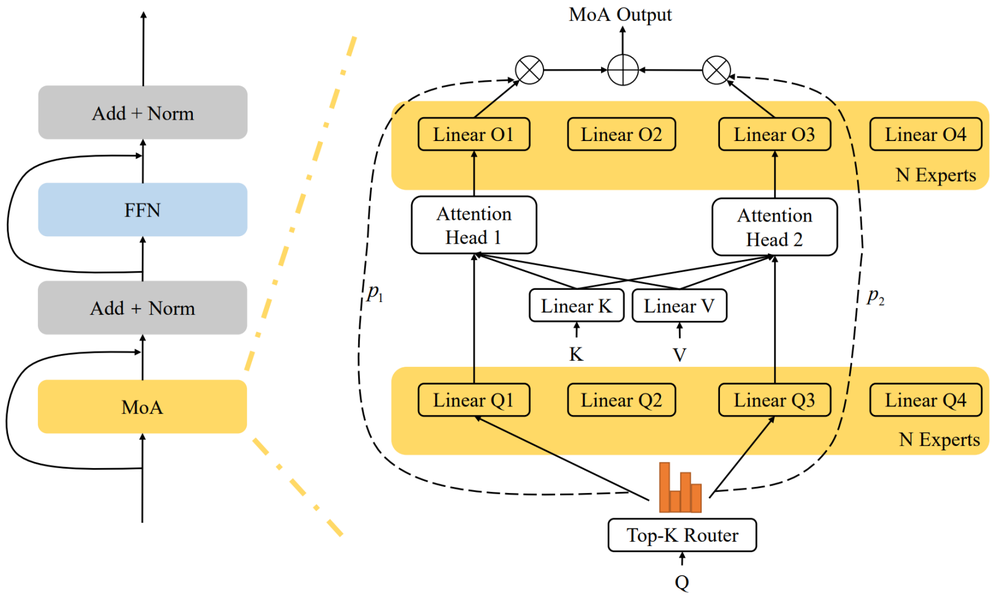


Figure 2: Mixture of Attention Heads (MoA) architecture. MoA contains two mixtures of experts. One is for query projection, the other is for output projection. These two mixture of experts select the same indices of experts. One routing network calculates the probabilities for each selected experts. The output of the MoA is the weighted sum of the outputs of each selected experts.

attention experts

$$y_t = \sum_{i \in G(q_t)} w_{i,t} \cdot E_i(q_t, K, V) \qquad G(q_t) \subseteq \{E_i\}$$

routing network

$$p_{i,t} = \text{Softmax}_i(q_t \cdot W_g) \qquad G(Q) = \text{TopK}(p_{i,t}, k)$$

$$w_{i,t} = \frac{p_i}{\text{Detach}\left(\sum_{j \in G(q_t)} p_j\right)}$$

$$W_{i,t}^{\text{att}} = \text{Softmax}\left(\frac{q_t W_i^q (K W^k)^T}{\sqrt{d_h}}\right)$$

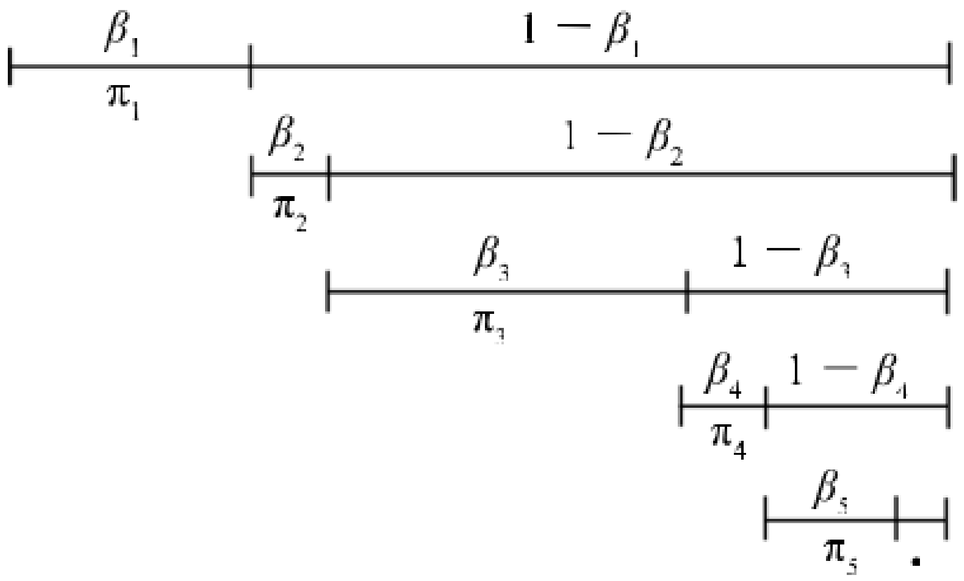
$$o_{i,t} = W_{i,t}^{\text{att}} V W^v \qquad E_i\left(q_t, K, V\right) = o_{i,t} W_i^o$$

shares W^k and W^v

Mixture of Attention Heads: Selecting Attention Heads Per Token, EMNLP 2022.

Methods

Stick-breaking Self-Attention head



remain length

$$L_1 = 1 - \beta_1$$

$$L_2 = L_1(1 - \beta_2)$$

$$L_3 = L_2(1 - \beta_3)$$

$$\beta_0 = 0$$

$$\pi_i = \beta_i \prod_{i=0}^{i-1}(1 - \beta_i)$$

$$\sum_{i=1}^{\infty} \pi_i = 1$$

Methods

Stick-breaking Self-Attention head

Given an input vector sequence of t time steps $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t$,

$$\beta_{i,t} = \text{sigmoid}(\mathbf{k}_i^\top \mathbf{q}_t).$$

$$p_{i,t} = \beta_{i,t} \prod_{i<j\leq t} (1 - \beta_{j,t})$$

selects the latest match token, implicitly encodes position information, no additional positional embedding

$$\mathbf{o}_t = \mathbf{W}_o^\top \sum_{i\leq t} p_{i,t} \mathbf{v}_i,$$

shares W^k and W^v

Methods

Load Balancing during Pretraining

maximize $\mathcal{L}_{\text{MI}} = \sum_{\mathbf{x}} \sum_{m} p(m, \mathbf{x}) \log \left(\frac{p(m, \mathbf{x})}{p(m)p(\mathbf{x})} \right)$ $p(\mathbf{x}) = \frac{1}{\mathcal{X}}$

$$p(m) = \sum_{\mathbf{x}} g(m|\mathbf{x})p(\mathbf{x})$$

$$\mathcal{L}_{\text{MI}} = \underbrace{\sum_{m=1}^{N} p(m) \log p(m)}_{-H(m)} - \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \underbrace{\sum_{m=1}^{N} g(m \mid \mathbf{x}) \log g(m \mid \mathbf{x})}_{H(m|\mathbf{x})},$$

$I(X;Y) = H(X) - H(X|Y)$

balances the load of each expert across the entire batch

encourages each input x to concentrate their gating probability to fewer modules

Methods

Load Concentration during Finetuning (prune)

minimize $\mathcal{L}_{\text{entropy}} = H(m) = -\sum_{m=1}^{N} p(m) \log p(m),$

encouraging the model to use fewer experts

Inserting new Modules for Continual Learning

$h'(\mathbf{x}) = \begin{bmatrix} \mathbf{A} \\ \mathbf{A}' \end{bmatrix} \text{ReLU}(\mathbf{B}\mathbf{x})$ router

- Fully Trainable (joint training, router is trainable)
- Partially Trainable (lifelong training A') $\mathcal{L}_{\text{rout\_reg}} = ||\mathbf{A}'||^2.$

Experiments

Language Modeling

- Pretrain -> Pile corpus, 300 billion tokens

Table 1: Language Model Hyperparameters

Model	D_{emb}	N_{layer}	N_{att}	D_{att}	N_{ffd}	D_{ffd}	Top$k$	total params	active params
MoLM-4B-K2	1024	24	16	1024	32	2048	2	4B	350M
MoLM-4B-K4	1024	24	16	1024	32	2048	4	4B	700M
MoLM-8B-K2	1024	48	16	1024	32	2048	2	8B	700M

Experiments

Language Modeling

GPT-2 pretrained on the pile

Table 2: Inference Speed and Zero-shot performance. We measure the end-to-end latency and peak memory consumption for an input of batch size 32 and length 1024 on an A100 80GB GPU. We also measure the throughput of each model when the GPU memory is fit with the maximum number of 1024-token sequences.

	Latency ms	Memory GB	Throughput tokens/sec	Hellaswag	PIQA	ARC-e acc	ARC-c	OBQA
Pythia 410M	554	25	59594	33.72	66.70	51.89	21.42	18.2
GPT-Neo 1.3B	991	23	32857	38.66	71.11	56.19	23.12	21.4
Pythia 1.4B	918	42	35559	40.41	70.84	60.52	26.11	22.2
MoLM-4B-K2	497	27	71017	39.21	70.13	56.44	23.55	20.8
GPT-Neo 2.7B	1737	35	18788	42.71	72.2	61.07	27.47	23.2
Pythia 2.8B	2111	70	15522	45.34	73.99	64.35	29.35	23.8
MoLM-4B-K4	863	27	39931	42.20	73.01	60.82	25.94	22.6
MoLM-8B-K2	939	38	37419	43.33	72.91	62.46	27.90	23.8

MoLM only uses around 25% active parameters per token compared to its dense counterpart

Experiments

Language Modeling

Table 3: Few Shot, Code Generation, and Language Modeling performance.

	TriviaQA			HumanEval pass@k [%]			Wikitext
	0-shot	1-shot	5-shot	k=1	k=10	k=100	PPL
Pythia 410M	2.32	5.02	6.42	1.20	3.85	9.98	20.09
GPT-Neo 1.3B	5.24	8.01	9.74	3.62	6.87	14.50	16.16
Pythia 1.4B	5.30	9.87	12.84	2.19	7.31	14.33	14.71
MoLM-4B-K2	5.40	11.12	13.70	3.04	6.99	13.79	15.15
GPT-Neo 2.7B	4.82	11.23	13.67	4.89	9.54	17.90	13.93
Pythia 2.8B	7.38	15.58	18.98	4.91	11.76	21.54	12.68
MoLM-4B-K4	9.07	14.24	16.49	5.50	10.65	20.27	13.20
MoLM-8B-K2	11.47	16.73	20.75	5.51	12.58	20.40	12.97

Experiments

Learning New Knowledge with New Modules

pre-train ModuleFormer and GPT-Neo on CC-100 Corpus

- continual joint pre-training (with English text 4:1)
- continual lifelong pre-training (no English text)

Experiments

continual joint pre-training

Table 4: Continual Joint Pre-Training Result (accuracy↑).

	Model	Trainable Params	Continual Training (de) en	Continual Training (de) de	Continual Training (vi) en	Continual Training (vi) vi
Freeze All	GPT-Neo-2.7B	0	78.6	64.5	78.6	**58.9**
	MoLM-4B-K4	0	**80.9**	**67.8**	**80.9**	58.4
Fully Tuned	GPT-Neo-2.7B	2.7B	75.1(-3.5)	71.0(+6.5)	75.1(-3.5)	66.0(+7.1)
	MoLM-4B-K4	4B	**80.1**(-0.8)	**71.5**(+3.7)	**80.1**(-0.8)	**67.9**(+9.5)
LoRA	GPT-Neo-2.7B	152M	75.1(-3.5)	66.7(+2.2)	74.4(-4.2)	62.0(+3.1)
	GPT-Neo-2.7B	261M	75.2(-3.4)	67.0(+2.5)	74.9(-3.7)	62.6(+3.7)
Insert New Experts	**MoLM-4B-K4**	266M	**79.2**(-1.7)	**70.8**(+3.0)	**79.4**(-1.5)	**66.2**(+7.8)

sparse models experience less interference

Experiments

continual lifelong pre-training

Table 5: Continual Lifelong Pre-Training Result (accuracy↑).

	Model	Trainable Params	Regularization	Continual Training (vi) en	Continual Training (vi) vi
LoRA	GPT-Neo-2.7B	24M	N/A	69.3(-9.3)	57.5(-1.4)
	GPT-Neo-2.7B	133M	0.25 Weight Decay	69.9(-8.7)	57.8(-0.6)
	GPT-Neo-2.7B	133M	N/A	70.3(-8.3)	59.8(+1.4)
Insert New Experts	**MoLM-4B-K4**	151M	N/A	74.5(-6.4)	**68.0**(+9.6)
	MoLM-4B-K4	151M	0.25 Rout Reg	76.0(-4.9)	64.8(+6.4)
	MoLM-4B-K4	151M	0.50 Rout Reg	76.3(-4.6)	64.5(+6.1)
	MoLM-4B-K4	151M	1.00 Rout Reg	**78.7**(-2.2)	63.0(+4.6)

balance and trade-off stability-plasticity with the help of the routing regularization loss

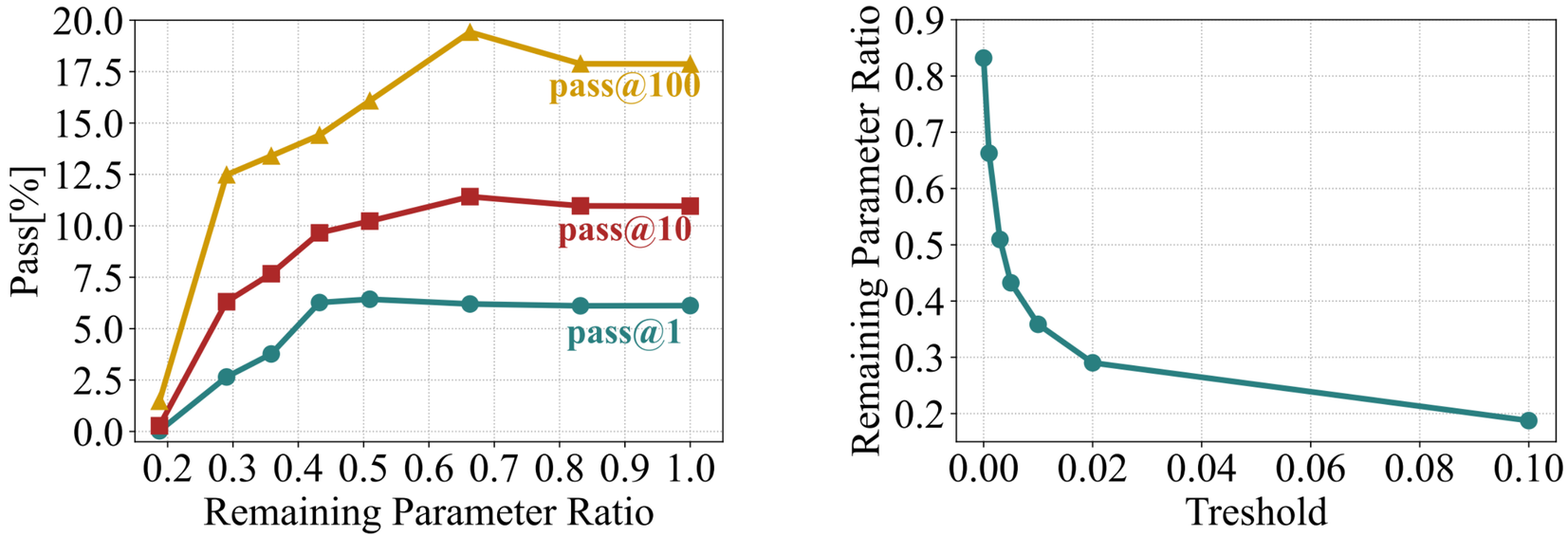
Experiments

Finetuning and Pruning Modules

- Finetuning (GitHub-code-clean 15B, finetune 2 epochs with load concentration loss)
- Pruning (count the activation frequency on dev)
- Evaluation (test on HumanEval)

Experiments

Finetuning and Pruning Modules



(a) ratios-pass@k

(b) threshold-ratios

Figure 2: Performance after Pruning on the HumanEval Dataset.

around half of the modules' activation frequency being less than 0.3% of the most frequently used experts

Conclusion

- same performance as dense LLMs with lower latency (50%)
- less susceptible to catastrophic forgetting
- specialize a subset of modules and the unused modules can be pruned

SPARSE MOE AS THE NEW DROPOUT: SCALING DENSE AND SELF-SLIMMABLE TRANSFORMERS

Tianlong Chen[1]*, Zhenyu Zhang[1]*, Ajay Jaiswal[1], Shiwei Liu[1], Zhangyang Wang[1]
[1]VITA Group, University of Texas at Austin
{tianlong.chen,zhenyu.zhang,ajayjaiswal,shiwei.liu,atlaswang}@utexas.edu

ICLR 2023

Motivation

Sparse Mixture-of-Experts (SMoEs)

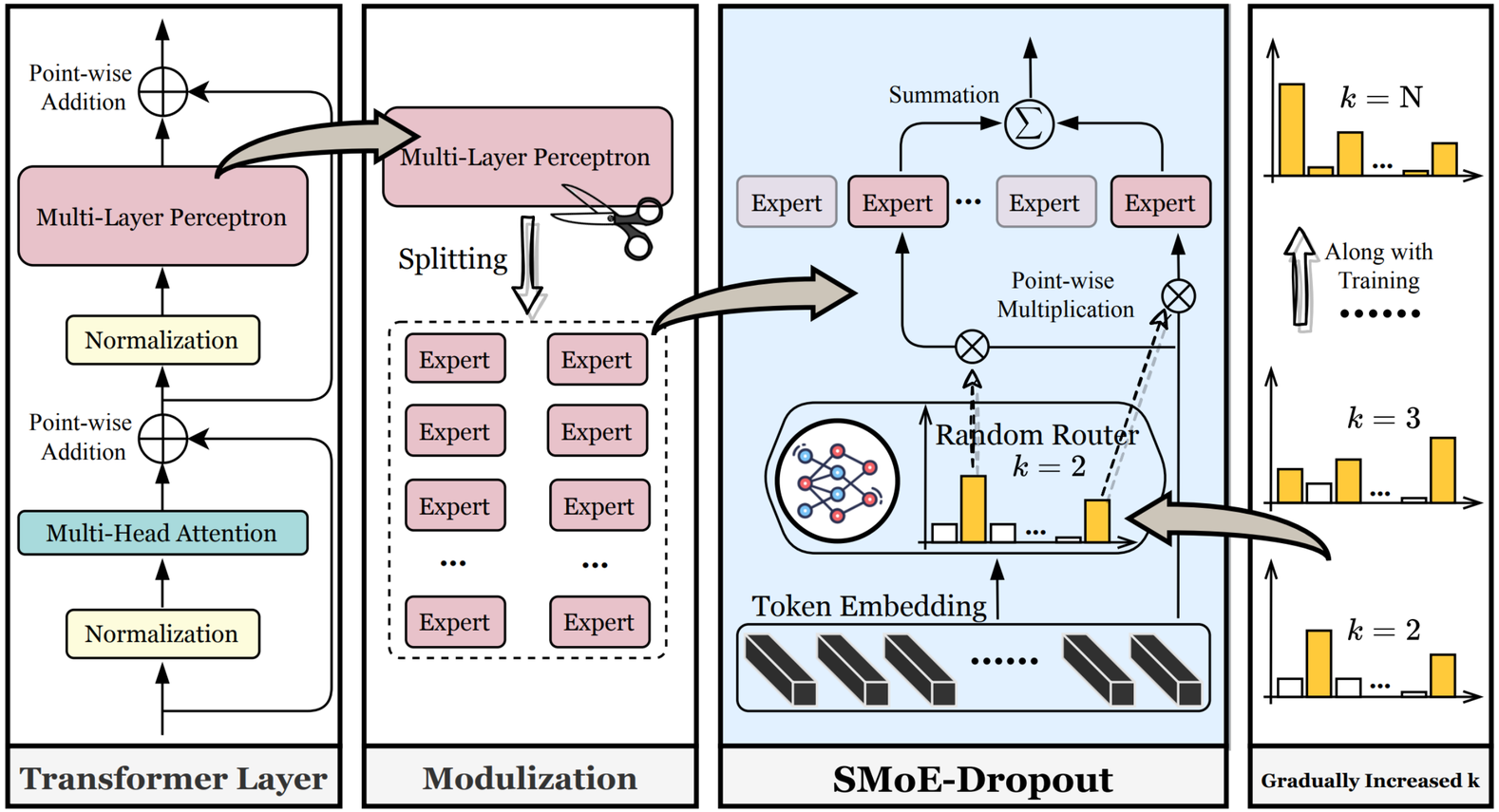
- scaling model capacity with a fixed computational cost
- learning-based routing mechanisms imply a trend toward representation collapse
  - tend to push hidden representations clustering around expert centroids
  - redundant experts, inferior expert specialization, sub-standard centroids
- poor scalability during inference and downstream fine-tuning (overfitting)

Motivation

Question ?

- *Does there exist a principled and pluggable approach to modify SMoE training that can enhance scalability at inference and downstream fine-tuning of large-scale transformers, by dynamically adapting the number of activated experts subject to resource availability?*

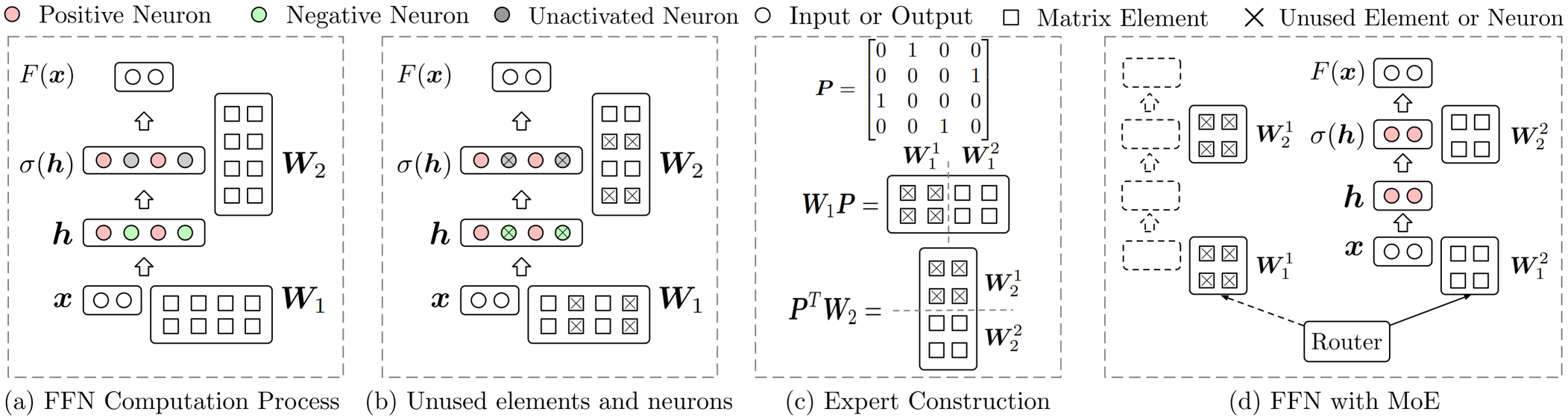
Methods


Point-wise Addition
Multi-Layer Perceptron
Normalization
Point-wise Addition
Multi-Head Attention
Normalization
Transformer Layer
Multi-Layer Perceptron
Splitting
Expert
Expert
Expert
Expert
Expert
Expert
Expert
Expert
Modulization
Summation
Expert
Expert
Expert
Expert
Point-wise Multiplication
Random Router
$k = 2$
Token Embedding
SMoE-Dropout
$k = N$
Along with Training
$k = 3$
$k = 2$
Gradually Increased k

Methods

Modulization

$$\boldsymbol{h} = \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1,$$
$$F(\boldsymbol{x}) = \sigma(\boldsymbol{h})\boldsymbol{W}_2 + \boldsymbol{b}_2,$$



(a) FFN Computation Process (b) Unused elements and neurons (c) Expert Construction (d) FFN with MoE

- Parameter Clustering Split (K-Means of W1)
- Co-Activation Graph Split (graph partitioning algorithms)

MoEfication: Conditional Computation of Transformer Models for Efficient Inference, ACL Findings, 2022.

Methods

Random Routing Policy

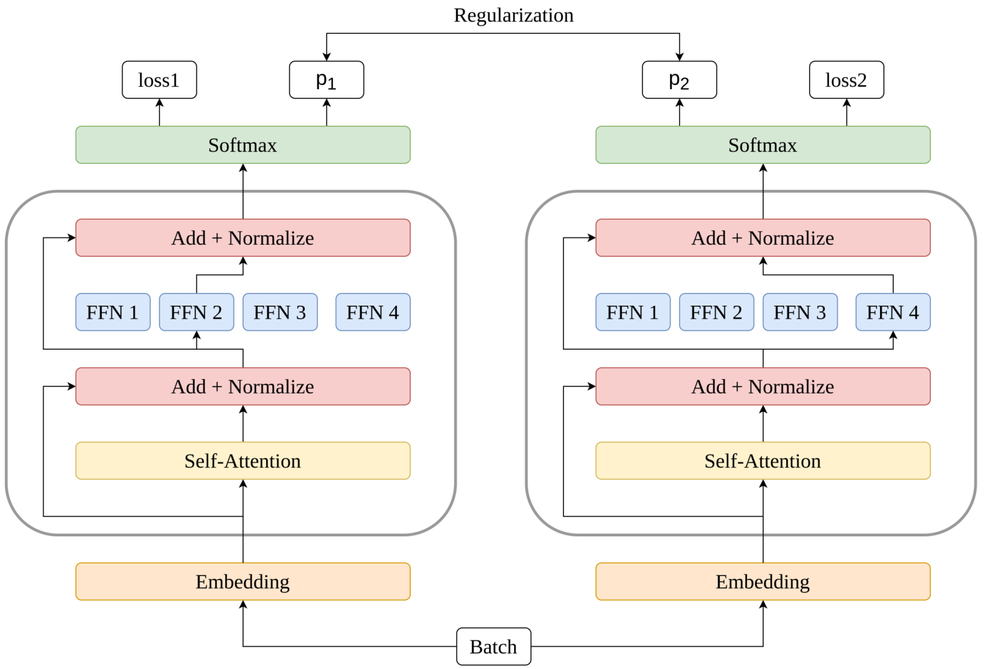
- a randomly initialized and fixed router network (single layer)
- linearly increasing k during training

Experiments

Baseline

- Densely Training w. Dropout
- Densely Training w. Concrete Dropout

  automatic tune the dropout probability
- Densely Training w. DropBlock
- Training w. THOR



Experiments

Pre-Training

Table 1: Testing performance of {Transformer-XL, BERT, RoBERTa} network backbones on {`enwik8`, `BookCorpus`, `BookCorpus`} datasets, respectively. **All models are compared under the same number of parameter counts**. Training time (s) and inference FLOPs ($\times 10^{10}$) are reported. For THOR (Zuo et al., 2022), SMoE, and SMoE-Dropout, evaluations are performed with half ($k = \frac{\mathrm{N}}{2}$) or all ($k = \mathrm{N}$) experts activated.

Methods	Transformer-XL			BERT			RoBERTa		
	BPC (↓)	Time	Infer. FLOPs	BPC (↓)	Time	Infer. FLOPs	BPC (↓)	Time	Infer. FLOPs
Dense w. Dropout	1.1623	5.1298	7.7579	7.6546	0.2088	135.72	8.0903	0.1898	101.75
Dense w. Concrete Dropout	1.3335	6.3519	7.7579	7.6419	0.3031	135.72	8.0820	0.2410	101.75
Dense w. DropBlock	1.2468	5.3902	7.7579	7.6349	0.2119	135.72	8.0773	0.1934	101.75
THOR ($k = \mathrm{N}$)	1.3110	4.8830	7.7620	7.6434	0.1439	135.73	8.0778	0.1607	101.76
SMoE ($k = \mathrm{N}$)	1.1896	4.7982	7.7620	7.7537	0.1387	135.73	8.4291	0.1538	101.76
SMoE-Dropout ($k = \frac{\mathrm{N}}{2}$)	1.1776	5.0220	5.6145	7.6372	0.1905	89.330	6.7693	0.1799	78.558
SMoE-Dropout ($k = \mathrm{N}$)	1.1486	5.0220	7.7620	7.6293	0.1905	135.73	6.5491	0.1799	101.76

Bits-Per-Character

Experiments

varying the expert numbers (parameter counts) during evaluation

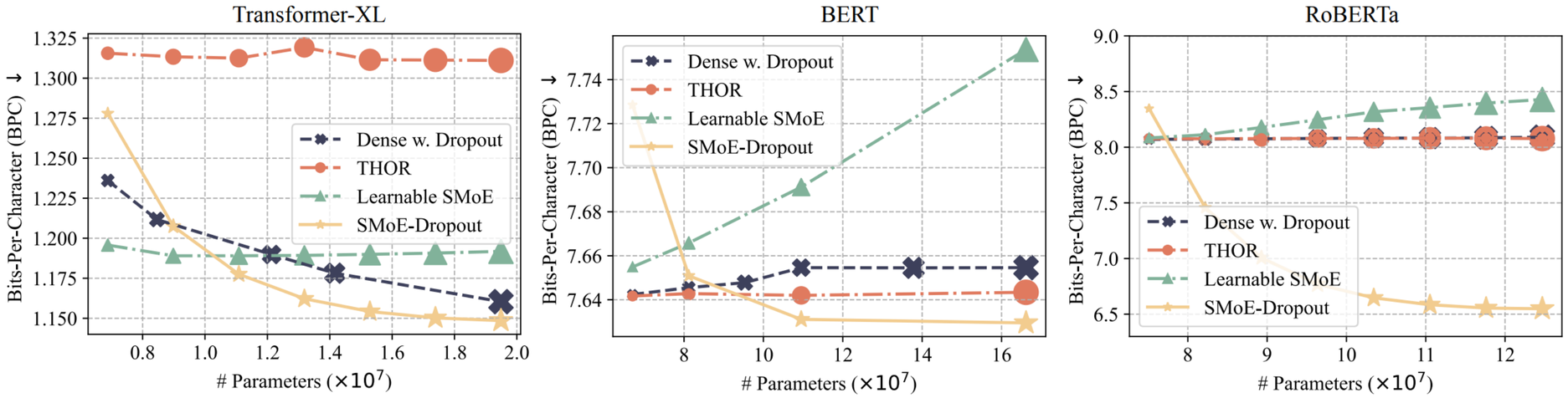


Figure 3: Testing performance over # parameter counts of {Transformer-XL, BERT, RoBERTa} networks on {`enwik8`, `BookCorpus`, `BookCorpus`} datasets, respectively. A smaller BPC suggests a better model.

Experiments

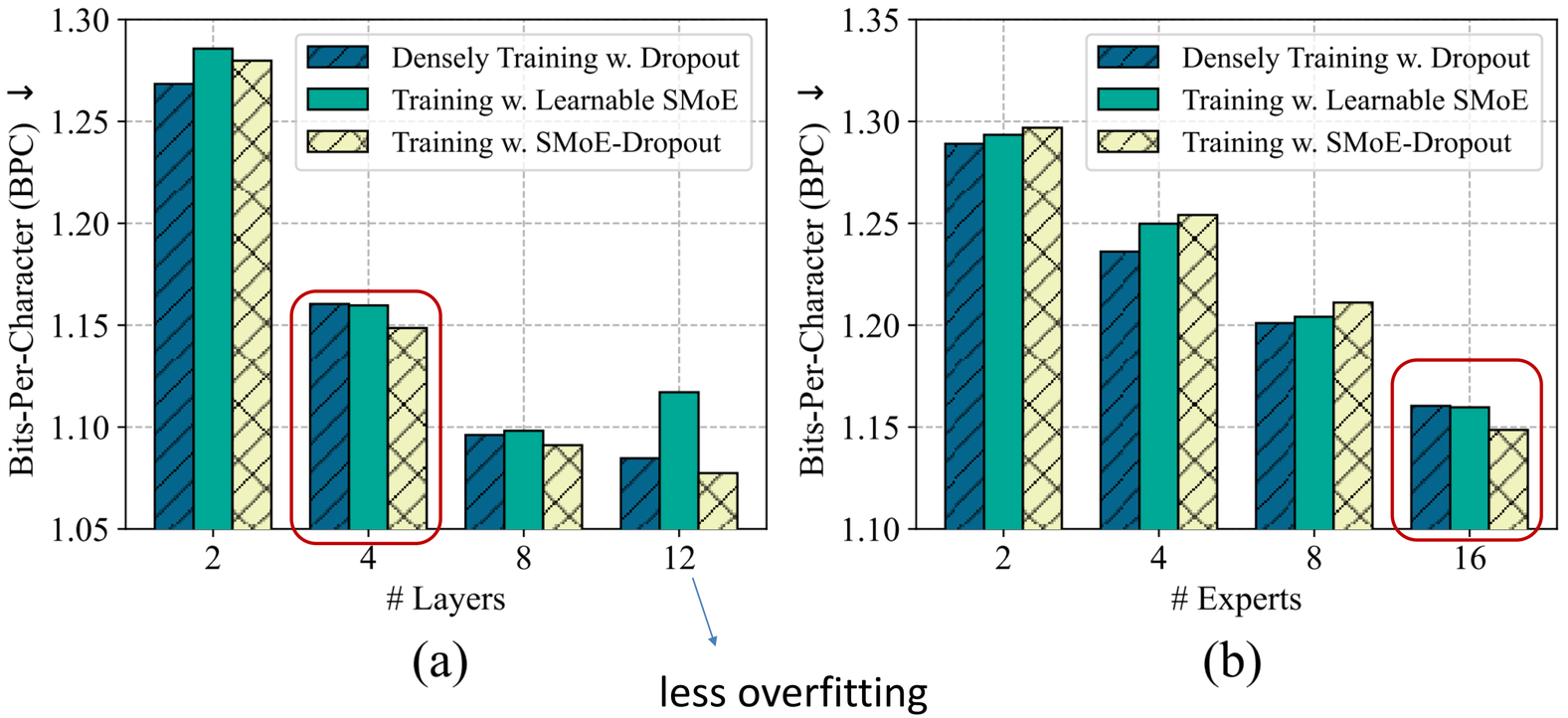
Finetuning results

Table 2: Transfer performance {Accuracy (% ↑), Problem Solving Rate (% ↑)} of {Transformer-XL, BERT, RoBERTa} networks on {SST-2, CSQA, ASDiv-A, MAWPS, SVAMP} datasets. **All models are compared under the same number of parameter counts**. The same densely fine-tuning is adopted for all approaches, while THOR, SMoE, and SMoE-Dropout are tuned with half ($k = \frac{\mathrm{N}}{2}$) or all ($k = \mathrm{N}$) experts activated.

Methods	Transformer-XL	BERT				RoBERTa		
	SST-2	CSQA	ASDiv-A	MAWPS	SVAMP	ASDiv-A	MAWPS	SVAMP
Dense w. Dropout	81.94	30.44	55.27	80.47	34.24	49.58	78.06	28.90
THOR ($k = \mathrm{N}$)	81.13	20.79	52.52	79.95	30.43	53.36	77.86	28.44
SMoE ($k = \mathrm{N}$)	79.98	29.27	55.88	80.73	33.15	52.10	77.86	27.98
SMoE-Dropout ($k = \frac{\mathrm{N}}{2}$)	81.60	30.32	54.97	80.99	33.65	52.94	76.30	31.19
SMoE-Dropout ($k = \mathrm{N}$)	82.41	30.51	56.30	81.25	35.33	55.46	78.13	33.94

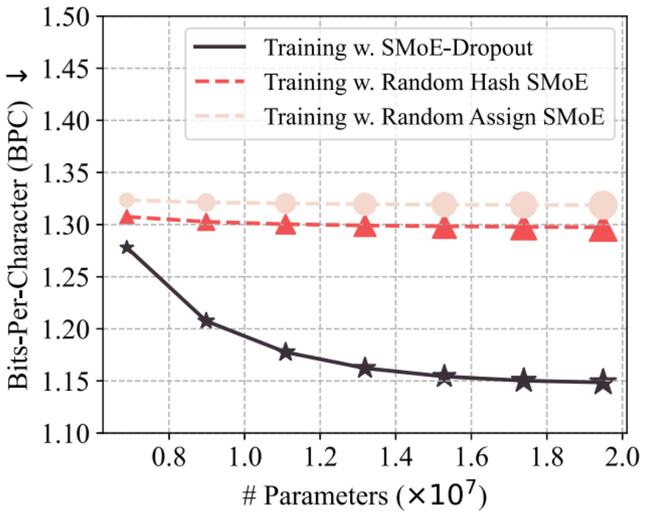
Experiments

Q1: When does SMoE-Dropout outperform other baselines? A1: Sufficient Model Capacity

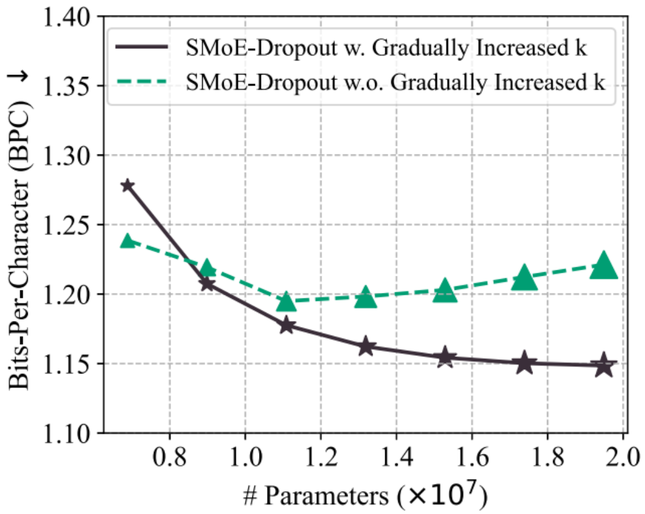


Experiments

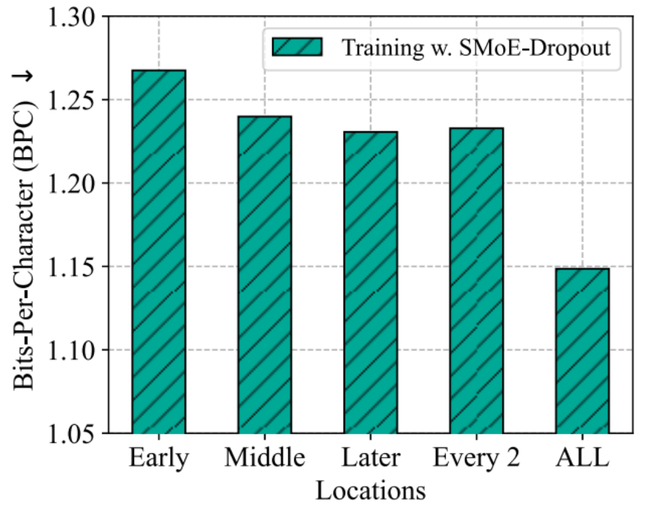
Q2: What is a better SMoE-Dropout design? A2: Random Weight Router; Later-layer SMoE.



(c)



(d)



(e)

Experiments

Q3: Extra benefits from SMoE-Dropout? A3: Improved Distillation and Less Overfitting.

distill knowledge into the same smaller variant with a single expert

Table 3: Distillation results of Transformer-XL on `SST-2`.

Method	Accuracy (↑)
Dense w. Dropout	81.25
THOR	80.76
SMoE	80.12
SMoE-Dropout	82.01