Models with Conditional Computation Learn Suboptimal Solutions

Mohammed Muqeeth, Haokun Liu & Colin Raffel
Department of Computer Science
University of North Carolina at Chapel Hill
`{muqeeth,haokunl,craffel}@cs.unc.edu`

Abstract

Sparsely-activated neural networks with conditional computation learn to route their inputs through different subnetworks, providing a strong structural prior and reducing computational costs. Despite their possible benefits, models with learned routing often underperform their parameter-matched densely-activated counterparts as well as models that use non-learned heuristic routing strategies. In this paper, we hypothesize that these shortcomings stem from the gradient estimation techniques used to train sparsely-activated models with non-differentiable discrete routing decisions. To test this hypothesis, we evaluate the performance of sparsely-activated models trained with various gradient estimation techniques in three settings where a high-quality heuristic routing strategy can be designed. Our experiments reveal that learned routing reaches substantially worse solutions than heuristic routing in various settings. As a first step towards remedying this gap, we demonstrate that supervising the routing decision on a small fraction of the examples is sufficient to help the model to learn better routing strategies. Our results shed light on the difficulties of learning effective routing and set the stage for future work on conditional computation mechanisms and training techniques.

1 Introduction

Neural networks typically use all of their parameters to process an example. This means that the computational cost of a neural network is often directly related to the number of parameters it has. However, there are cases where it might be appropriate to use a model architecture where different parts of the model are active for different inputs. Such an architecture can decouple the computational cost of a model from the number of parameters that it has. This possibility is increasingly useful given the current trend of scaling up models (Kaplan et al., 2020) because there may be cases where it is beneficial to train a model with more parameters but it is prohibitively expensive to train a typical densely-activated neural network (Fedus et al., 2021). Separately, specializing different parts of the model to different types of data may reduce interference and allocate capacity more effectively across downstream tasks (Sanh et al., 2021; Wei et al., 2021; Zamir et al., 2018; Bao et al., 2021) or languages (Pires et al., 2019; Liu et al., 2020b; Xue et al., 2020).

Conditional computation techniques provide a possible way to attack these issues because they allow the network to selectively apply a subset of its parameters to an input. Specifically, models with condition computation typically consist of multiple *subnetworks* (often called "experts") controlled by *routers* that decide which subnetwork should be active. As a result, a model with many subnetworks can have a large number of parameters while incurring a lower computational cost by selecting a small number of subnetworks to activate. When the model is trained with diverse data, conditional computation can allow subnetworks to specialize to different types of inputs while allowing flexible knowledge sharing across subnetworks (Ma et al., 2019). However, because routing involves making

I Can't Believe It's Not Better Workshop at NeurIPS 2022.

a discrete decision as to which subnetwork to use, the loss on final prediction cannot back-propagate though the routing decision to update the router. Consequently, conditional computation often uses gradient estimation techniques to train the routers (Jang et al., 2016; Clark et al., 2022; Fedus et al., 2021; Bengio et al., 2013).

In practice, past work has shown models with conditional computation do not always learn effective routing strategies. For example, Mittal et al. (2022) investigate models with a continuous router in a controlled setting and find the models do not route examples from the same group to the same subnetworks, and perform poorly comparing to models with oracle routing. However, models with task-specific modules (Gururangan et al., 2021; Kudugunta et al., 2021) provide evidence that it is possible to train effective models with specialized subnetworks. As an extreme example, Roller et al. (2021) achieves results comparable to learned routing with a fixed random routing. Relatedly, Fedus et al. (2021) find the gain from scaling up parameters by $30\times$ with a sparsely-activated model is smaller than scaling up both parameters and FLOPs by $3\times$ in a dense model. As a possible explanation, Clark et al. (2022) characterize how models with conditional computation improve with scale and find a detrimental term that scales with the product of the log number of subnetworks and active parameters. Consequently, increasing the number of subnetworks yields limited returns and existing methods for training conditional computation models may only be helpful when the number of active parameters is moderate.

In this work, we hypothesize that issues with conditional computation stem from difficulties with gradient estimation. Specifically, we design experimental settings where we can compare learned routing to a performant hand-designed heuristic routing scheme. A schematic of these two possibilities is shown in fig. 1. We find that all gradient estimation techniques that we consider produce models that underperform the heuristic routing, even in cases where a better routing strategy than the hand-designed one is possible. We also experiment with training the router to follow the heuristic routing for different subsets of the training data and find that it consistently improves performance and sometimes results in better performance than using the heuristic routing alone. Our results shed light on existing shortcomings of models with conditional computation and provide a new testbed for designing more performant methods.

After providing the background on conditional computation models and gradient estimators in the following section, we report our experiment findings in section 3. Finally, we discuss related works in section 4 and conclude in section 5.

2 Background

To provide the necessary background for our experimental study, we first explain how sparsely-activated neural networks use conditional computation, then discuss gradient estimators that enable learning routing strategies. In addition, we introduce the idea of "heuristic" routing strategies in settings where a performant routing can be hand-designed.

To use conditional computation, models organize subnetwork modules by blocks and incorporate such blocks into deep neural network architectures. A conditional computation block B comprises a set of N subnetworks $\{f_1, f_2, \dots f_N\}$ and a router R. Subnetworks in the same block accept inputs of the same dimensionality. Given a hidden-state representation u of example x, the output of the i-th subnetwork is $f_i(u)$. In our work, the router chooses a single f_i to process the input of the block (though sparsely-activated models in other work may use more than one subnetwork (Shazeer et al., 2017; Du et al., 2022)). Thus we can use B like any regular building block in a neural networks.

2.1 Gradient Estimators

Because the router decision is discrete and therefore not differentiable, we can't train the router's parameters through standard gradient-based learning. Fortunately, gradient estimators can provide approximate gradients to the router parameters. There are a few common designs shared by models that use gradient estimators to train routers.

Their router R often applies a lightweight network to some intermediate hidden states v in the model rather than the original input x. The lightweight routing network yields a probability distribution $P(v)$ over all the N subnetworks. Different gradient estimators vary in how they make the routing decision from P and how they construct the output from the chosen subnetwork. Additionally, some estimators may introduce additional loss terms.

REINFORCE Gradients can be estimated through nondifferentiable operations through reinforcement learning techniques (Schulman et al., 2015; Bengio et al., 2013). In reinforcement learning, a policy loss is used to train an agent to learn optimal actions in an environment. In this paper, we use the REINFORCE algorithm which computes the policy loss as $\log(\pi)r$ where r denotes the received reward for taking an action whose assigned probability is π. In our conditional computation setup, we aim to train the model to choose which subnetwork to use to process a given input. Here, the router R acts an agent that samples a subnetwork to use according to the routing probabiltiies. In order to train such a router, we use the router's assigned probability to the sampled subnetwork as π. For the reward r, we use the negative of the model's loss. The router is therefore trained to pick subnetworks that maximize the reward which, in turn, minimizes the loss. The REINFORCE estimator often suffers from high variance because of the sampling operation. This motivates the use of baselines, which reduce variance without changing the optimal solution. Here, we follow the approach of Clark et al. (2022): The baseline b is generated by a small neural network with a single hidden layer that takes as input v and is trained with Huber loss. The overall loss function is

$$L = \mathbb{E}_{i \sim P(v)} \alpha \log P(v)_i (r - b) - \beta P(v) \log P(v) + \gamma L_{\text{Huber}}(r, b) \tag{1}$$

where $P(v)$ is the routing probability distribution and α, β, and γ are hyperparameters that correspond to policy gradient weight, policy entropy weight, and value loss weight. Finally, the output of the block B is just $f_i(u)$.

Straight Through Gumbel-Softmax (ST-Gumbel) The Gumbel-Softmax trick (Jang et al., 2016) provides a continuous differentiable approximation to sampling from a categorical distribution. Specifically, Gumbel noise is added to the logits of the distribution and a temperature scale is applied in the softmax operation. Denoting $g_i \sim$ Gumbel(0, 1) and τ as the temperature, the Gumbel-Softmax trick produces the following modified distribution:

$$\hat{P}(v)_i = \frac{\exp((\log(P(v)_i) + g_i)/\tau)}{\sum_{j=1}^{N} \exp((\log(P(v)_i) + g_i)/\tau)} \tag{2}$$

The subnetwork f_i with the highest assigned probability is chosen by applying an argmax operation over this distribution. In order to approximate gradients through the argmax operation, we use the Straight-Through estimator which replaces $f_i(u)$ with $(1 - \text{sg}[\hat{P}(v)_i] + \hat{P}(v)_i) f_i(u)$ where sg stands for the stop-gradient operator. During forward pass, the multiplier for $f_i(u)$ becomes 1 and the multiplier receives gradients for the term $\hat{P}(v)_i$ in the backward pass. This estimator gradually anneals temperature τ from a high to low value so that the approximated samples are more and more similar to discrete samples.

Top-k Shazeer et al. (2017) propose a gradient estimation scheme where the router sends the input through the k subnetworks that are assigned the highest probability. Fedus et al. (2021) later found that this router could be used effectively when $k = 1$. Specifically, the estimator selects the subnetwork with the highest probability and scale its output using its corresponding routing probability. The output of the block is therefore $p_i(v) f_i(u)$, where $i = \text{argmax}_i(P(v))$.

2.2 Heuristic Routing

In this work, we are interested in determining whether existing techniques for training sparsely-activated models can learn an effective routing strategy. As a point of comparison, we therefore hand-design three baseline routing strategies that do not require training a router.

Tag Routing If we have prior knowledge about the data that a model will be applied to, it can be possible to hand-design a heuristic routing strategy for choosing which subnetworks to use for a given example based on data properties.

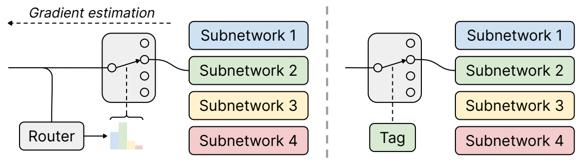

Figure 1: In this work, we compare learned routing (left) to heuristic routing (right) in models with conditional computation. Gradient estimation techniques must be used to train models with learned routing due to the discrete routing decision.

Routing	DAE-Syn	T5-GLUE	Res-Dom
Tag	0.2	81.3	62.5
Hash	1587.2	74.0	54.5
Monolithic	1018.6	77.9	60.5
Tag+	–	82.0	–
Top-k	0.7	79.2	61.5
ST-Gumbel	20.9	79.1	60.3
REINFORCE	26.0	80.5	61.5

Table 1: Performance of heuristic and learned routing strategies.

Tag routing takes advantage of tags associated with the examples (such as domain or dataset in multitask learning) and associates each subnetwork in a given conditional computation block with a particular tag. In this work, we assume each example has a single tag. As such, examples are routed to the subnetwork corresponding to their tag.

Hash Routing Roller et al. (2021) propose hash routing which uses a fixed hashing function to determine which subnetwork to use for a given example. Specifically, each example is assigned an approximately-random subnetwork to use in each conditional computation block consistently over the course of training. This approach disregards any shared characteristics across examples. We adopt this routing strategy by mapping each example to a fixed subnetwork within a given conditional computaion block and always route that example to that subnetwork throughout training.

Monolithic Routing As a baseline, we consider models where each conditional computation block only has a single subnetwork. This provides an important point of comparison as it is a degenerate solution that can be found with learned routing.

3 Experiments

To comprehensively study the capacity and issues of models with conditional computation, we perform experiments in three scenarios that differ in architecture and datasets. Specifically, we experiment with training a discrete autoencoder on synthetic images (DAE-Syn), fine-tuning T5-small (Raffel et al., 2020) on GLUE (Wang et al., 2018) with task adapters (Houlsby et al., 2019)(T5-GLUE), and using domain adapters for fine-tuning a ResNet18 (He et al., 2016) on DomainNet (Peng et al., 2019)(Res-Dom). In each scenario, we compare learned routing using the gradient estimators from section 2.1 to heuristic routing strategies from section 2.2. Full details of each scenario along with hyperparameters, model architectures, and dataset examples are detailed in appendix B. For scenarios with multiple datasets, we only provide the average performance across datasets in the main paper due to space limitations. All results are reported as the average of three runs with full results provided in appendix D.

3.1 Can learned routing find a performant solution?

Method To assess the overall effectiveness of learned routing strategies, we compare the performance of sparsely-activated models trained with the gradient estimators described in section 2.1 to models with different heuristic routing strategies. In addition, we note that learned routing strategies could in principle outperform tag routing by discovering and exploiting shared characteristics of the data. Based on prior results in transfer learning on GLUE that show that intermediate- or multi-task training on MNLI and RTE can improve performance on RTE (Phang et al., 2018; Devlin et al., 2018; Pruksachatkun et al., 2020; Vu et al., 2020), we include an additional tag routing scheme ("Tag+") for GLUE where in conditional computation blocks from the encoder we route examples from RTE and MNLI to the same subnetwork. We can directly assess the effectiveness of techniques for learning routing by observing whether they match or outperform heuristic routing schemes.

Results We find that in all settings, our hand-designed heuristic routing is better than learned routing schemes, though learned routing does usually outperform hash routing and monolithic routing. In all settings and across all gradient estimators, we find that learned routing underperforms heuristic routing. In particular, the best learned routing scheme underperforms heuristic routing by 0.8% in

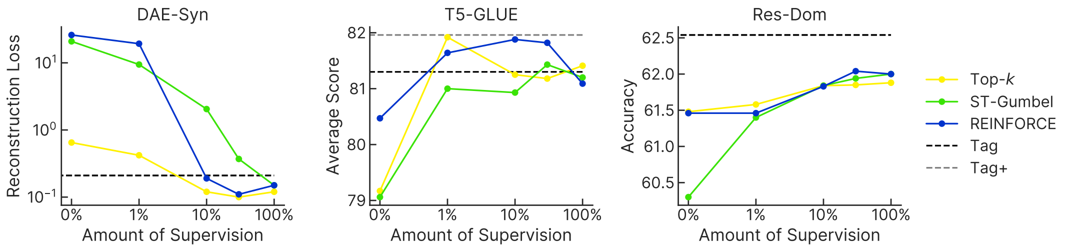

Figure 2: Effect of training the router to produce tag annotations for varying subsets of the training data. "Tag+" routing (for GLUE only) refers to routing MNLI and RTE to the same subnetwork.

T5-GLUE and by 1.0% in Res-Dom. In addition, the hand-designed RTE-MNLI routing strategy (which could in principle be found by learned routing) is 1.5% better than the best learned routing scheme, suggesting that gradient estimators are even further behind the best-possible performance. The only setting where a learned routing scheme achieved comparable performance to tag routing was the Top-k estimator on DAE-Syn, which achieved a slightly-worse loss of 0.7 compared to tag routing's 0.2. We note that top-k routing produces an autoencoder that is not strictly discrete because it involves rescaling the chosen subnetwork's output by the probability assigned to that subnetwork. This property could be the primary factor that contributes to its improved performance when compared to other estimators, which all attained substantially worse loss than tag routing. In all settings, we find that Hash routing significantly underperforms both tag and learned routing, suggesting that learned routing is at least better than a random routing baseline. For DAE-Syn and T5-GLUE, learned routing also outperformed monolothic routing. However, the worst-performing gradient estimator (ST-Gumbel) on Res-Dom had comparable performance to monolothic routing, suggesting that learned routing may have found a solution no better than a pathological one.

3.2 Can models learn effective routing with the help of extra supervision?

Method Our results from section 3.1 show that learned routing underperforms tag routing. One possible explanation is that gradients estimators are ineffective, but it is also possible that the tags are hard to discover from the input, so the routers can't simulate tag routing regardless of the gradient provided by the estimators. To investigate these possibilities, we add supervision on the routing decisions and observe the model performance. Specifically, we choose a subset (from 1% to all) of the training examples to include a tag annotation for and add supervision by training the router to output the tag for annotated examples via a standard cross-entropy classification loss. After training with the cross-entropy loss, we also test whether additional training of the converged model with the tag supervision removed causes the performance to improve or regress.

Result We find that even a small amount of supervision can help close the gap from tag routing. In all settings, we find that adding routing supervision improves performance. On DAE-Syn and T5-GLUE, supervision allows learned routing to match the performance of tag routing. Furthermore, in T5-GLUE, we found that learned routing with tag supervision actually outperformed tag routing in some cases, with the strongest performance of 81.9% being attained by top-k with 1% of training examples supervised, which nearly matched the "Tag+" (routing MNLI and RTE to the same subnetwork) performance of 82.0%. Notably, performance on T5-GLUE *decreased* back to tag routing-level performance (81.3%) as the proportion of tag-annotated examples increased. This suggests that only a hint of tag supervision is enough to enable gradient estimators to discover highly performant routing schemes. On the other hand, on Res-Dom we found that tag supervision only went about halfway towards closing the gap between learned and tag routing. In addition, we tested whether further training without supervision caused performance to increase or decrease. In all cases shown in table 2 (appendix), performance stayed consistent, which suggests that once a performant routing is found, the gradient estimators we considered do not cause the model to degrade to a worse solution. To investigate why learned routing on Res-Dom does not reach tag routing's performance, we trained a ResNet model to predict the domain of a given example from DomainNet and found that it only attained 85% accuracy on held-out data. This suggests that the tags might not be reconstructable from the input, so tag routing might not be a discoverable solution by learned routing in the Res-Dom case.

4 Related Works

Models with Conditional Computation Researchers have found a few ways to circumvent the difficulties in routing for multi-task learning. e.g. Deecke et al. (2020); Hazimeh et al. (2021); Dua et al. (2021) start training with most of the subnetworks activated and gradually introduce sparsity. Kudugunta et al. (2021); Ponti et al. (2022); Ma et al. (2019); Gupta et al. (2022) group examples from the same task together and introduce task-specific parameters in the router. Some works avoid learned routing by hand-crafting heuristic routing strategies. Gururangan et al. (2021) built sparsely-activated language models where different domains use separate subnetworks. On an unknown domain, the model assesses the subnetworks' fitness to combine the subnetworks. Li et al. (2022) repeat this process to initialize subnetworks for new domains and create a forest of subnetworks iteratively. Tang et al. (2022); Pfeiffer et al. (2022, 2020) specify task-subnetworks assignment based on human knowledge. Because sparsely-activated models disentangle computation and and parameter count, significant effort has gone into leveraging conditional computation to create massive pretrained models with a feasible computation cost (Fedus et al., 2022; Shazeer et al., 2017; Fedus et al., 2021; Du et al., 2022; Zoph et al., 2022; Yu et al., 2022). Many works explore different routing methods in this setting, with heavy focus on balancing the load of subnetworks (Lewis et al., 2021; Zhou et al., 2022; Kool et al., 2021; Roller et al., 2021). Another line of work aims to introduce ways to convert trained dense models into similar-sized sparse models with a lower computational footprint (Lee-Thorp & Ainslie, 2022; Zhang et al., 2022). More generally, there are other forms of conditional computation beyond the formulation we describe in this work (Han et al., 2021). Modular networks (Kirsch et al., 2018; Jiang & Bansal, 2019; Hu et al., 2017) use a router to assemble heterogeneous subnetworks into a network layout specific to given input example. Early exiting (Graves, 2016; Xin et al., 2020; Liu et al., 2020a; Bengio et al., 2015) saves computation by terminating inference in earlier layers or recurrent steps.

Gradient Estimation Techniques Many gradient estimators have been proposed to produce approximate gradients for learning discrete representations that involve sampling. Our work uses a learned baseline from Clark et al. (2022) to reduce the variance of the REINFORCE estimator. The REBAR estimator (Tucker et al., 2017) adds a reparameterizable term to REINFORCE as a baseline that results in a more effective unbiased estimator. This additional term uses a relaxed sample similar to Gumbel-Softmax (Jang et al., 2016). RELAX (Grathwohl et al., 2017) is similar to REBAR but uses a learnable neural network for the reparameterizable term. Kool et al. (2019) uses additional samples from the policy as a built-in baseline for REINFORCE. Yin & Zhou (2018) and Dong et al. (2020) use the idea of coupling between multiple samples to reduce the variance of the gradient estimator and are designed for binary latent variables. Dong et al. (2021) improve upon Yin & Zhou (2018) and Dong et al. (2020) by extending the estimator to categorical variables.

Issues with Conditional Computation Clark et al. (2022) study the scaling law of sparse language models, and discovered a computational cutoff above which no additional benefits are observed. Relatedly, Du et al. (2022) observe worse results when further scaling up the number of subnetworks. Chi et al. (2022) and Dai et al. (2022) discover the representation collapse and routing fluctuation issue, respectively. Mittal et al. (2022) create a set of simple and modular data modular distributions, and show that systems with modular architecture can not find the most beneficial solution when trained end-to-end. Ye et al. (2022) experiment with various designs for multi-task learning with task-level routing and find that the performance still cannot surpass simple multi-task baselines.

5 Conclusion

In this work, we tested whether conditional computation models with learned routing can produce effective routing strategies. Specifically, we designed three experimental settings where a performant heuristic routing strategy could be hand-designed. And our experiments showed learned routing with gradient estimators learn inferior solutions. As a small step towards a remedy, we experimented with using tag annotations to supervise learned routing, finding that this improved learned routing performance in all cases. Furthermore, we found one setting where learned routing could outperform tag routing when only 1% of examples were given tag annotations. Our results shed light on shortcomings of models with conditional computation and the gradient estimation techniques used to train them and provide a testbed for future work in the area.

References

Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*, 2009.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, and Furu Wei. On the representation collapse of sparse mixture of experts. *arXiv preprint arXiv:2204.09179*, 2022.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International Conference on Machine Learning*, pp. 4057–4086. PMLR, 2022.

Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. *arXiv preprint arXiv:2204.08396*, 2022.

Lucas Deecke, Timothy Hospedales, and Hakan Bilen. Latent domain learning with dynamic residual adapters. *arXiv preprint arXiv:2006.00996*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Zhe Dong, Andriy Mnih, and George Tucker. Disarm: An antithetic gradient estimator for binary latent variables. *Advances in neural information processing systems*, 33:18637–18647, 2020.

Zhe Dong, Andriy Mnih, and George Tucker. Coupled gradient estimators for discrete latent variables. *Advances in Neural Information Processing Systems*, 34:24498–24508, 2021.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Dheeru Dua, Shruti Bhosale, Vedanuj Goswami, James Cross, Mike Lewis, and Angela Fan. Tricks for training sparse translation models. *arXiv preprint arXiv:2110.08246*, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Shashank Gupta, Subhabrata Mukherjee, Krishan Subudhi, Eduardo Gonzalez, Damien Jose, Ahmed Hassan Awadallah, and Jianfeng Gao. Sparsely activated mixture-of-experts are robust multi-task learners. *ArXiv*, abs/2204.07689, 2022.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A Smith, and Luke Zettlemoyer. Demix layers: Disentangling domains for modular language modeling. *arXiv preprint arXiv:2108.05036*, 2021.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 804–813, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Yichen Jiang and Mohit Bansal. Self-assembling modular networks for interpretable multi-hop reasoning. *arXiv preprint arXiv:1909.05803*, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. *Advances in neural information processing systems*, 31, 2018.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019.

Wouter Kool, Chris J Maddison, and Andriy Mnih. Unbiased gradient estimation with balanced assignments for mixtures of experts. *arXiv preprint arXiv:2109.11817*, 2021.

Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. *arXiv preprint arXiv:2110.03742*, 2021.

James Lee-Thorp and Joshua Ainslie. Sparse mixers: Combining moe and mixing to build a more efficient bert. *arXiv preprint arXiv:2205.12399*, 2022.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *ArXiv*, abs/2208.03306, 2022.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*, 2020a.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020b.

Jiaqi Ma, Zhe Zhao, Jilin Chen, Ang Li, Lichan Hong, and Ed H Chi. Snr: Sub-network routing for flexible parameter sharing in multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 216–223, 2019.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a modular architecture enough? *arXiv preprint arXiv:2206.02713*, 2022.

Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1406–1415, 2019.

Jonas Pfeiffer, Ivan Vulic, Iryna Gurevych, and Sebastian Ruder. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *EMNLP*, 2020.

Jonas Pfeiffer, Naman Goyal, Xi Victoria Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. Lifting the curse of multilinguality by pre-training modular transformers. In *NAACL*, 2022.

Jason Phang, Thibault Févry, and Samuel R Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.

Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert? *arXiv preprint arXiv:1906.01502*, 2019.

Edoardo M Ponti, Alessandro Sordoni, and Siva Reddy. Combining modular skills in multitask learning. *arXiv preprint arXiv:2202.13914*, 2022.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R Bowman. Intermediate-task transfer learning with pretrained models for natural language understanding: When and why does it work? *arXiv preprint arXiv:2005.00628*, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Duyu Tang, Fan Zhang, Yong Dai, Cong Zhou, Shuangzhi Wu, and Shuming Shi. One model, multiple tasks: Pathways for natural language understanding. *ArXiv*, abs/2203.03312, 2022.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. Exploring and predicting transferability across nlp tasks. *arXiv preprint arXiv:2005.00770*, 2020.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*, 2020.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.

Qinyuan Ye, Juan Zha, and Xiang Ren. Eliciting transferability in multi-task learning with task-level mixture-of-experts. *arXiv preprint arXiv:2205.12701*, 2022.

Mingzhang Yin and Mingyuan Zhou. Arm: Augment-reinforce-merge gradient for stochastic binary networks. *arXiv preprint arXiv:1807.11143*, 2018.

Ping Yu, Mikel Artetxe, Myle Ott, Sam Shleifer, Hongyu Gong, Ves Stoyanov, and Xian Li. Efficient language modeling with sparse all-mlp. *arXiv preprint arXiv:2203.06850*, 2022.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3712–3722, 2018.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890, 2022.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing. *arXiv preprint arXiv:2202.09368*, 2022.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. Designing effective sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In appendix F
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In appendix E
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All of the code used in all of our experiments will be made publicly available. We use public datasets and all processing steps are documented in our code.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In appendix B
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We take average from three random seeds but did not compute or analyze the error bars.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In appendix A
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] .
   (b) Did you mention the license of the assets? [Yes] In appendix G
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] .
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Compute resources used

All models were trained either on 48GB A6000s or 24GB A5000s. Each experiment on synthetic image reconstruction took about 5 hrs to train. Each experiment on T5 with Adapters for GLUE (T5-GLUE) took about 16 hours to train. Each experiment on ResNet with Adapters for DomainNet (Res-Dom) took about 11 hours to train.

B Experiment Details

Discrete Autoencoders for Synthetic Images (DAE-Syn) In the first scenario, we consider synthetic image reconstruction using a discrete autoencoder model with categorical random variables. Such a model can be seen as a conditional computation model because the layer immediately after the discrete bottleneck can be seen as the chosen subnetwork. Figure 3 (appendix) shows a few images in this dataset. Each image is uniquely determined by 4 independent random variables, each of which take on 8 possible values, representing alphabet identity, alphabet location, foreground color, and background color. We treat these values as tags in tag routing, with $32 = 8 \times 4$ tags in total. We generate the $8^4 = 4096$ unique images as the training set and train the model with an L_2 reconstruction loss. We measure the model performance as the loss of the entire dataset after training for a fixed number of steps. Our model contains an encoder and a decoder. The encoder is a convolution network followed by four parallel routers, each of which chooses from 8 subnetworks. Each router consists of a batch normalization, a linear layer, and a softmax function. The decoder first uses 32 subnetworks that consist of a single embedding vector each. The subnetworks, when chosen, output the corresponding vector regardless of their input. Together with the routers in the encoder, these subnetworks form 4 conditional computation blocks placed in parallel. The chosen embedding vectors are used to compute the outputs of the conditional blocks and their sum is fed into a transpose-convolution network to reconstruct the image.

All of the experiments use learning rate of $1e^{-4}$ with batch size of 128 and were trained for $200k$ steps with the warmup ratio of 0.1. For ST-Gumbel estimator, we use τ value of 10 and anneal rate of $1e^{-6}$ in the equation 2. For REINFORCE, we use values of 1, $1e^{-6}$, and $1e^{-2}$ for the corresponding α, β, and γ hyperparameters in the equation 1. The values for α and γ are tuned over $\{1, 0.1\}$ and $\{1e^{-1}, 1e^{-2}, 1e^{-3}\}$ by fixing β to 0 and then the value of β is tuned over $\{0, 1e^{-6}, 1e^{-5}, 1e^{-4}\}$ by taking the best values of α and γ found in the previous setup. The loss weight for the supervising the router is taken as 10 for all of the estimators after tuning over $\{10, 1, 0.1, 0.01\}$.

T5 with Adapters for GLUE (T5-GLUE) In the second scenario, we focus on Adapters (Houlsby et al., 2019), which can be seen as conditional computation blocks that typically use tag routing. Specifically, we focus on training a T5 model (Raffel et al., 2020) on the GLUE benchmark (Wang et al., 2018) for natural language processing. GLUE consists of nine datasets ranging across sentimental analysis (SST-2 (Socher et al., 2013)), acceptability judgement (CoLA (Warstadt et al., 2019)), natural language inference (MNLI (Williams et al., 2017), RTE (Bentivogli et al., 2009)), semantic similarity (QQP[1], MRPC (Dolan & Brockett, 2005), STS-B (Cer et al., 2017)), question answering (QNLI (Rajpurkar et al., 2016)), and commonsense reasoning (WNLI (Levesque et al., 2012)). Following convention, we exclude WNLI and use the remaining 8 datasets. We adopt the method from Raffel et al. (2020) to process examples into input and target text and base our implementation on Mahabadi et al. (2021). When using tag routing, we consider the dataset of each example as the tags. We use the pretrained T5-small model as the backbone and adapt the model in a way similar to adding adapters Houlsby et al. (2019) for a single task, i.e. we keep all pretrained parameters frozen except for layer normalization parameters and insert conditional computation blocks after self-attention and feed-forward modules. Since T5-small has 6 Transformer layers each in the encoder and decoder and each layer includes one self-attention and one feed-forward module, there are 24 conditional computation blocks in total. In each block, we include 8 adapters (the number of datasets in GLUE) as the subnetworks. For the routers, we use a stack of a batch normalization, a linear layer, and a softmax nonlinearity to produce the routing probability distribution. In the encoder, a router takes as input the preceding hidden states, averaged over the input sequence. In the decoder, routers receive the average of the encoder's final hidden states instead of the decoder hidden states so that there is no information leakage from later target tokens to earlier target tokens.

[1] `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

Routing	DAE-Syn	T5-GLUE	Res-Dom
Top-k	0.1	81.9	61.8
from supervision checkpoint	0.1 (+0.0)	81.0 (-0.9)	62.2 (+0.4)
ST-Gumbel	0.4	81.0	61.8
from supervision checkpoint	0.2 (-0.2)	81.0 (+0.0)	61.5 (-0.3)
REINFORCE	0.1	81.6	61.8
from supervision checkpoint	0.1 (+0.0)	81.7 (+0.1)	62.1 (+0.3)

Table 2: Performance of estimators from further training of tag supervised models by removing the supervision. Tag supervision considered for DAE-Syn, T5-GLUE, and Res-Dom are 30%, 1%, and 10% of total tags.

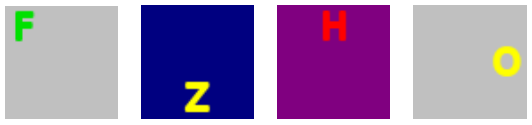

Figure 3: Synthetic images used for DAE-Syn. The (alphabet identity, alphabet location, foreground color, background color) tags for each image are (F, top-left, green, grey), (Z, bottom-mid, yellow, violet), (H, top-mid, red, purple) and (O, mid-right, yellow, grey) in the order of images.

All T5 models are trained for 2^{18} steps with learning rate of $1e^{-3}$, with $2k$ warmup steps, and batch size of 128. We use τ value of 10 and anneal rate of $1e^{-6}$ for the ST-Gumbel estimator. The values of α, β, and γ for the REINFORCE estimators are $1e^{-2}$, $5e^{-4}$, and $1e^{-2}$ following (Clark et al., 2022). The weight of the loss for supervising the router is taken as 1 for all of the estimators after tuning over $\{10, 1, 0.1, 0.01\}$.

The adapters used in this settings are simple bottleneck architectures with $swish$ non-linearity in between. The inputs values are added back to the output of the bottleneck block and then layer normalization is applied for calculating the final output of the adapter.

ResNet with Adapters for DomainNet (Res-Dom) Our last scenario is similar to the second but with images. We use DomainNet dataset (Peng et al., 2019) and ResNet18 backbone model (He et al., 2016). DomainNet is an object recognition task covering six distinct domains. All domains include the same 345 object categories. We treat domains as tags. As in the T5-GLUE scenario, we freeze the pretrained model, and insert 8 conditional computation blocks after each of the 8 residual layer groups in the model. Each condition computation block includes 6 Adapters (corresponding to the number of domains). We use the same architecture for routers as in T5-Small and feed average-pooled hidden states into the router to compute the routing probability.

All ResNet models are trained for $100k$ steps with batch size of 128 and learning rate of $1e^{-3}$ and no warm up. We use τ value of 10 and anneal rate of $1e^{-4}$ for the ST-Gumbel estimator. The values of α, β, and γ for the REINFORCE estimators are $1e^{-2}$, $5e^{-4}$, and $1e^{-2}$ similar to T5-GLUE experiments. The supervised loss weight is taken as 0.1 after tuning over $\{1, 0.1, 0.01\}$.

The adapters used are same bottleneck architectures with the same non-linearity as in T5-GLUE. The inputs are first batch normalized and then passed through the bottleneck architectures. The final output of the adapter is the sum of input and the output of the bottleneck block.

C Continue training from supervision checkpoint

In table 2 we show the performance of continuing training the model after removing routing supervision loss.

D Full results on T5-GLUE and Res-Dom

We show full results of T5-GLUE in table 3 and Res-Dom in table 4.

Routing	RTE acc	SST-2 acc	MRPC f1	MRPC acc	STS-B pearson	STS-B spearman	QQP f1	QQP acc	MNLI acc	QNLI acc	CoLA mcc	Average
Tag	58.9	90.9	90.7	87.2	88.1	87.7	85.9	89.5	81.7	90.2	43.5	81.3
Hash	51.7	88.5	84.4	78.3	84.6	84.4	82.8	87.2	79.1	88.6	6.5	74.0
Monolithic	60.1	91.3	88.3	84.1	88.0	87.8	84.7	88.6	81.1	90.0	13.0	77.9
Tag+	65.5	90.7	90.7	87.2	88.2	87.8	85.9	89.5	81.8	90.2	44.2	82.0
Top-k	58.7	90.5	91.1	88.0	88.0	87.7	85.3	89.0	81.3	89.9	21.2	79.2
w/ 1% supervision	65.7	90.4	91.2	87.8	88.2	87.9	85.9	89.4	81.6	90.1	42.8	81.9
w/ 10% supervision	60.9	90.2	90.1	86.4	88.6	88.2	85.9	89.4	81.8	90.2	42.3	81.2
w/ 30% supervision	59.7	90.6	90.7	87.4	88.2	87.9	86.0	89.5	81.7	90.1	41.2	81.2
w/ 100% supervision	64.0	90.9	91.0	87.7	88.0	87.7	86.0	89.5	81.6	90.2	39.2	81.4
from supervision checkpoint	60.1	90.0	90.1	86.7	88.4	88.1	86.2	89.6	82.0	89.8	39.7	81.0
ST-Gumbel	61.6	91.2	90.0	86.4	87.9	87.6	85.2	88.8	81.1	90.1	19.9	79.1
w/ 1% supervision	60.6	90.7	90.4	86.9	88.5	88.0	86.0	89.5	81.6	90.3	38.6	81.0
w/ 10% supervision	58.7	90.9	90.0	86.4	88.1	87.8	85.9	89.5	81.7	90.2	41.2	80.9
w/ 30% supervision	59.9	90.6	90.8	87.4	88.0	87.6	86.0	89.5	81.6	90.3	44.2	81.4
w/ 100% supervision	61.4	90.7	90.4	86.9	88.2	88.0	85.9	89.5	81.8	90.1	40.6	81.2
from supervision checkpoint	62.3	90.4	91.5	88.3	87.9	87.7	86.1	89.6	81.8	90.1	35.7	81.0
REINFORCE	64.7	90.5	90.6	87.5	88.3	88.1	85.6	89.2	81.6	90.3	28.8	80.5
w/ 1% supervision	64.0	90.9	90.7	87.2	88.4	88.0	85.8	89.4	81.5	90.1	42.2	81.6
w/ 10% supervision	62.6	91.3	91.9	89.0	88.3	88.0	85.9	89.4	81.8	90.3	42.2	81.9
w/ 30% supervision	62.8	91.1	91.6	88.7	88.0	87.6	85.9	89.4	81.8	90.3	42.9	81.8
w/ 100% supervision	59.7	91.3	91.6	88.7	88.3	87.9	85.9	89.4	81.8	90.3	37.3	81.1
from supervision checkpoint	65.9	90.6	91.2	88.0	87.6	87.1	86.1	89.6	82.1	89.9	40.0	81.7

Table 3: Full T5-GLUE results.

Routing	Clipart	Infograph	Painting	Quickdraw	Real	Sketch	Average
Tag	56.3	24.6	51.9	51.0	70.1	48.4	62.5
Hash	64.2	31.5	59.4	62.7	74.7	57.0	54.5
Monolithic	62.7	29.3	56.5	60.6	73.3	54.4	60.5
Top-k	63.7	30.8	57.5	61.6	74.0	55.6	61.5
w/ 1% supervision	63.6	30.6	57.2	62.9	74.2	55.7	61.6
w/ 10% supervision	63.3	30.8	56.9	62.8	74.3	55.6	61.8
w/ 30% supervision	63.5	30.5	57.4	62.9	74.2	55.3	61.9
w/ 100% supervision	63.6	30.6	57.2	62.9	74.2	55.7	61.9
from supervision checkpoint	63.5	30.6	57.0	63.8	74.3	56.1	62.2
ST-Gumbel	62.4	29.2	56.4	60.2	73.2	54.3	60.3
w/ 1% supervision	63.9	31.3	57.5	62.6	74.4	55.8	61.4
w/ 10% supervision	63.3	31.3	57.5	62.4	74.3	55.6	61.8
w/ 30% supervision	63.5	31.4	57.6	62.7	74.2	55.7	61.9
w/ 100% supervision	63.9	31.3	57.5	62.6	74.4	55.8	62.0
from supervision checkpoint	63.2	30.6	57.3	61.8	74.3	55.4	61.5
REINFORCE	62.8	30.5	57.4	62.3	73.8	55.1	61.5
w/ 1% supervision	63.6	31.4	57.7	62.6	74.4	55.9	61.5
w/ 10% supervision	63.3	31.1	57.6	62.5	74.2	55.7	61.8
w/ 30% supervision	63.6	31.2	57.7	62.7	74.5	55.9	62.0
w/ 100% supervision	63.6	31.4	57.7	62.6	74.4	55.9	62.0
from supervision checkpoint	63.8	31.1	57.1	63.5	74.2	56.0	62.1

Table 4: Full Res-Dom results.

E Societal Impacts Statement

We are not aware of any negative ethical implications of our work. Our work does not involve human subjects and is primarily focused on diagnosing issues with an efficient class of neural networks. While conditional computation has been used to design extremely large neural networks (Shazeer et al., 2017; Fedus et al., 2021; Du et al., 2022) that have high computational costs (and, correspondingly, energy usage), our work primarily focuses on smaller-scale models.

F Limitations

Our experiments focus on only three widely-adopted gradient estimators and relatively small models. All our models use example-level routing, where the routing decision is made on an example basis. Routing at the token or task level could have different behaviors.

G License

T5 is licensed under Apache 2.0. QNLI uses CC BY-SA 4.0 license. MultiNLI uses data sources of multiple different licenses(Williams et al., 2017). CoLA, SST-2, RTE, MRPC, STS-B, QQP, and DomainNet allow non-commercial research use cases. Our code for T5-GLUE is based on Hyperformer, which is shared under Apache 2.0.