Royal Holloway University of London
Center for Reliable Machine Learning
Department of Computer Science

Methods of Training Task Decompositions in Gated Modular Neural Networks

Yamuna Krishnamurthy

A thesis submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science of the Royal Holloway University of London
March 2023

To my parents

How do I know what I think until I see what I say?

E.M. Forster

All the interests of my reason, speculative as well as practical, combine in the three following questions: 1. What can I know? 2. What ought I to do? 3. What may I hope?

Immanuel Kant, The Critique of Pure Reason

Abstract

Mixture of experts (MoE), introduced over 20 years ago, is the simplest gated modular neural network architecture. The gate in the MoE architecture learns task decompositions and individual experts (modules) learn simpler functions appropriate to the gate's task decomposition. This could inherently make MoE interpretable as errors can be attributed either to gating or to individual experts thereby providing either a gate or expert level diagnosis. Due to the specialization of experts they could modularly be transfered to other tasks.

However, our initial experiments showed that the original MoE architecture and its end-to-end expert and gate training method does not guarantee intuitive task decompositions and expert utilization, indeed it can fail spectacularly even for simple data such as MNIST.

This thesis therefore explores task decompositions among experts by the gate in existing MoE architectures and training methods and demonstrates how they can fail for even simple datasets without additional regularizations. We then propose five novel MoE training algorithms and MoE architectures: (1) Dual temperature gate and expert training that uses a softer gate distribution for training experts and a harder gate distribution to train the gate; (2) Two no-gate expert training algorithms where the experts are trained without a gate: (a) *loudest expert* method which selects the expert with the lowest estimate of its own loss for the sample both during training and inference; and (b) *peeking expert* algorithm that selects and trains the expert with the best prediction probability for the target class of a sample during training. A gate is then reverse distilled from the pre-trained experts for conditional computation during inference; (3) Attentive gating MoE architecture that computes the gate probabilities by attending to the expert outputs with additional attention weights during training. We then distill the trained attentive gate model to a simpler original MoE model for conditional computation during inference; and (4) Expert loss gating MoE architecture where the gate output is not the expert distribution but the expert log loss.

We also propose a novel flexible data driven soft constraint, L_s, that uses similarity between samples to regulate the gate's expert distribution. We empirically validate our methods on MNIST, FashionMNIST and CIFAR-10 datasets. The empirical results show that our novel training and regularization algorithms outperform benchmark MoE training methods.

Acknowledgements

The soul dries up without the company of the good.

Mahatma Gandhi

I would like to extend my profound gratitude and thanks to Professor C.S. Moghe and other distinguished faculty at VRCE, who set me off on this amazing path of exploring computer science. Moghe Sir, as we called him, made sure that we had a sound foundation in the fundamental concepts of computer science. Doing my Bachelor of Engineering at VRCE in Nagpur was also the first time I was away from home. The independence and perseverence, to make it on my own, that I learnt there has stood me in good stead over the years. My core CS group Ruchita, Manoj, Anubhav, Amit and Ajay are still an integral part of my life as are also Kiran, Panna, Paro, Bala, Divya, Gouselya, Sudha, Shalini, Poornima, Deepak and many other VRCEans, you know who you are. Rachna and Shweta thanks for all your collaboration during our final year project and all the fun we had working on it together.

Professor Douglas C. Schmidt gave me the opportunity to pursue my Master of Science in CS and be part of the DOC group at Washington University in St. Louis. There I developed a keen interest in modular and distributed computing which has since become my preferred approach to solving problems. This interest is what led me to my doctoral thesis topic. Doug also instilled in me the importance of following good coding standards and being proactive, something I adhere to religiously to date. I cannot thank Doug enough. I made lifelong friends and family at DOC group. Special mention to Kirthika (my housemate), Vishal, Marina, Pradeep, Nanbor, Naga, Alex, Irfan, Ossama, Bala, Carlos, Angelo, Kitty and my now brother-in-law Darrell. You all made the transition from India to US so smooth and fun. Genny, Adi, Netra, Vijay, Yasra, Nisha, Aarathi and Naveed, thanks for making the St. Louis days so memorable.

I would like to thank Professor Jeff Schneider, my master's advisor at Carnegie Mellon University (CMU), for making me part of Auton Lab and Professor Howie Choset who has been my mentor. I owe my exposure and grounding in machine learning and robotics to the outstanding faculty at CMU. Roman, Vlad, Randy, Mike, Dino, Lin, Fernando and Prateek made Pittsburgh a whole lot of fun too.

Thanks to Professor Katharina Morik for letting me be part of her Lehrstuhl 8 group at TU Dortmund and the opportunity to experience Germany. I have such fond memories of my time

in Bonn and Dortmund with Michael, Nessi, Marion, Hendrik, Simon, Lau, Silvia, Nico, Sibyll, Wouter, Christian, Marco and Tobias.

Needless to say I owe my doctoral journey to my supervisor Professor Chris Watkins. It has been such a tremendous learning experience and Chris' advise, support and the complete freedom he gave me to pursue my research have been invaluable and impactful for finishing my PhD. It is also always such fun to talk to Chris about any topic, be it about how birds learn or weapons of math desctruction or even Sasha Baron Cohen. I would like to thank my doctoral viva examiners, Professor Mahesan Niranjan (University of Southampton) and Senior Lecturer Nicola Paoletti (King's College London), for their valuable insights and discussions during the viva and to Professor Matt Hague (Royal Holloway University of London) for being the independent chair. I could not have finished all my thesis experiments without Dr. Nicolo Colombo who gave me access to the N8 CIR's Bede cluster and Francesco and Narinder who provided all the system support that I needed. Thanks to Kostas and Yang for making me part of their nurse decison support system project. It was a great learning experience. Many thanks to all the RHUL faculty that I worked with as a teaching assistant and teaching fellow. I am also grateful to Royal Holloway University of London for the full financial support I received during the course of my PhD.

I had such a fun time in Egham with the weekly pub quizzes at Beehive with my math and friends group Tom, Vicky, Maya, Darren, Pavel, Stergios, Mandy, Felix, Tabby and Sofiya. Thanks Emma and Hyosun for being sounding boards to talk about the fun and difficulties of doing a PhD. Nats and Abhi thanks for being there and letting me crash at your place when I am in London. Thanks Ayesha for all the fun trips, fancy dinners and high teas. Special thanks to Raja for the hikes, board games, restaurant and market hopping in London, LA and NYC, but most of all for being there at my hour of need. I am eternally grateful. Marco and Raya, thanks for continuing to be there for me, be it to put my furniture together, going to plays or giving me recommendations in Torino. I would go amiss if I did not mention the whole Nottingham German Stammtisch gang and extended family Roisin, Andrew, Orla, Anne, Sonja, Lauren, Rod, Kerstin and Rob.

Many thanks to Misha Bilenko for giving me the internship opportunity at Microsoft (MS) and continuing to mentor and help me. I had a blast at MS with Saghar, Elham, Daniel, Pragnya, Anirudh and many others. We have some fun dance videos to prove that.

A big shout out to my NYC friends Prem, Saba, Bhupesh, Nishta, Arvind, Tamanna, Julie, Zehra, Rahat, Sumaiyah, Batool, Asma, Dessi, Zoheb, Hari, Olga, Sundar, Ajey, Shirley, Pablo and Annie. We had such good times together and still continue to do so. Special mention to my dear friend Prem who introduced me to two of my favorites, Machine Learning and Thomas Gärtner. Also a very special thanks and gratitude to Saba's parents who have been my family away from family over the years. Another NYC shout out to all my NYU friends Fernando, Laura, Remi, Vicky, Raoni, Dani, Sonia, Roque, Kien, Aecio, Juliana, Masayo, Vitaly, Jorge, Ann, Vittorio, Paolo, Anshul, Neel, Jun, Eleni, Sungmin, Heejong, Joschi and Narges. Our trips to Ballston and Washington DC and all the caipirinhas will always be cherished. I am grateful to Professor Juliana Freire and Professor Claudio Silva for giving me the opportunity to be part of the fun VIDA Lab at NYU Tandon School of Engineering. There are many other friends and colleagues at Goldman Sachs and Carlin Equities among others who have all been a part of my life over the years. I cannot name all of you here but know that I am ever so glad that our paths crossed.

I could not have come this far without my family. This thesis is dedicated to my parents who provided me with every opportunity, advice, support and so much love. All the successes I have achieved are due to the values you have taught and instilled in me. My siblings Gowri, Devi and Vichu are my rocks. I know they will always have my back as I will always have theirs. I have the best brothers-in-law, Chandru and Darrell, who I know I can rely on anytime. Much love to my nieces Anushka and Emma and my nephews Bharat and Edward. You make me want to do my best so I can lead by example. My deepest respect and thanks to my grandparents who were an inspiration in so many ways. Much love and gratitude to my uncles, aunts and cousins for always being there for us. Many thanks to Thomas' family for looking out for me and being supportive.

My path to PhD has been a rather long one spread across different continents and countries. I could not have reached the end of this path without the patience, support, love and guidance from Thomas along with all the wines, beers, 5 course meals, tatorts and adventures.

Publications

The thesis is partially based on the following publication:

- **Yamuna Krishnamurthy** and C. Watkins. **Interpretability in Gated Modular Neural Networks**. In *Explainable AI approaches for debugging and diagnosis Workshop at NeurIPS*, 2021

During my DPhil, I also co-authored the publications below, which are not incorporated in this thesis:

- P. Lertvittayakumjorn, I. Petej, Y. Gao, **Yamuna Krishnamurthy**, A. Van Der Gaag, R. Jago, and K. Stathis. **Supporting Complaints Investigation for Nursing and Midwifery Regulatory Agencies**. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 81–91, Online, Aug. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.acl-demo.10

- R. Jago, A. van der Gaag, K. Stathis, I. Petej, P. Lertvittayakumjorn, **Yamuna Krishnamurthy**, Y. Gao, J. C. Silva, M. Webster, A. Gallagher, and Z. Austin. **Use of Artificial Intelligence in Regulatory Decision-Making**. *Journal of Nursing Regulation*, 12(3): 11–19, 2021. ISSN 2155-8256. doi: https://doi.org/10.1016/S2155-8256(21)00112-5. URL https://www.sciencedirect.com/science/article/pii/S2155825621001125

- I. Drori, **Yamuna Krishnamurthy**, R. de Paula Lourenco, R. Rampin, K. Cho, C. Silva, and J. Freire. **Automatic Machine Learning by Pipeline Synthesis using Model-Based Reinforcement Learning and a Grammar**. In *AutoML Workshop at ICML*, 2019

Contents

List of Tables

List of Figures

Notational Conventions

We will follow the notational conventions given below.

- Calligraphic letters ($\mathcal{A}, \mathcal{B}, \ldots$) denote sets or particular spaces:
  - $\mathcal{X}$ is an instance space,
  - $\mathcal{Y}$ is a label set,
  - $\mathcal{D}$ is a set of samples.
- Capital letters ($A, B, ...$) denote matrices or numbers:
  - Q is a query matrix,
  - K is a key matrix,
  - W is a weight matrix,
  - M is number of experts,
  - N is number of samples.
- Bold letters denote vectors
- Lowercase letters ($a, b, ...$) denote vectors, elements of some set, or functions:
  - x a single instance, and
  - y a single label
- Symbols:
  - $f : \mathcal{X} \to \mathcal{Y}$ denotes a function from $\mathcal{X}$ to $\mathcal{Y}$.
  - $f(\cdot)$ is for the function whereas $f(x)$ is only for the value of the function $f(\cdot)$ applied to x.

  - A^T denotes the transpose of the matrix A.

- Other notational conventions and exceptions:

  - $A_{m \times n}$ denotes a matrix A with dimension $m \times n$
  - $h_{1 \times n}$ denotes a vector h with dimension $1 \times n$
  - $\mathbb{R}$ is the set of all real numbers.
  - $\mathbb{N}$ is the set of all natural numbers $1, 2, 3, \ldots$

Chapter 1

Introduction

1.1 Motivation

Deep neural networks are currently the de facto machine learning models in diverse domains with large scale data especially image recognition, natural language processing and speech recognition. Processing image, text and speech data requires large deep learning models with billions of parameters. Training these models is computationally expensive as one has to evaluate the whole model for each sample during feedforward and backpropagation. Hence typically the models are pre-trained on large datasets and then fine-tuned for downstream tasks. But these new tasks may be subsets of the tasks on which the large model was trained on, so, using the entire pre-trained model is inefficient. In continual learning settings and when there are distribution shifts one has to re-train the whole model to prevent *catastrophic forgetting*. This is again computationally expensive. Large monolithic models are also complex and hence difficult to interpret.

Modular Neural Networks (MNN) are a promising solution to these challenges. They are composed of simple modules that can specialise in subtasks. Specialised modules can then be reused for other tasks. When new tasks arrive or there is a distribution shift, the network can be grown dynamically to add and train new modules to specialize in them. This prevents *catastrophic forgetting*.

Recently, there is a renewed interest in a class of MNNs called Gated Modular Neural Networks

(GMNN). GMNNs are MNNs with an addtional routing module called the *gate*. The gate typically learns the module distribution over data samples that decides which samples get routed to which module. This facilitates *conditonal computation* which is what makes GMNNs attractive for building large scale models. Conditional computation is when the gate introduces sparsity by using only some modules during training and inference per sample. This substantially reduces computational time and facilitates extremely large modular networks.

Since GMNNs are composed of simpler neural networks and the task is distributed among the individual modules, it seems intuitive that they should inherently be more interpretable. Individual modules can learn simpler functions appropriate to the decomposition and errors can be attributed to individual modules or the gate. Hence, it is imperative to look more closely at how the tasks are decomposed between modules in the existing GMNN architectures.

Mixture of Experts (MoE) introduced by Jacobs et al. (1991b) over 20 years ago is the simplest and most used GMNN. It consists of experts (modules) that are simple neural networks and another simple gate gate network. The gate is a soft switch that learns to route samples to the experts. Different MoE architectures can be realized based on how the expert outputs are aggregated and gating decisions computed. Typically the gate and the experts are trained end-to-end but they can also be trained separately.

Research in MoE has concentrated on improving computation time and successively building outrageously large networks (Shazeer et al., 2017; Rajbhandari et al., 2022; He et al., 2021) leveraging conditional computation and massively parallelizing expert computations. More recent works involve scaling transformers by replacing the single Feed Forward Network (FFN) with MoE (Lepikhin et al., 2021; Fedus et al., 2022). The goal here is distribute the input tokens across different hardware accelerators. Hence, by replacing a single FFN with multiple experts, each expert can run on a dedicated hardware accelerator. The gate then is required to equally distribute the samples across the experts, for optimal use of each expert, that also improves performance. Hence, the focus here is scaling the models and not what the experts are learning and how the task is distributed among the experts. That is, if each sub task of the task is learnt by just one module or is distributed among multiple models. A clear task decomposition is desirable for interpretability and transferability. In my research I have attempted to improve performance while also ensuring true modularity that would allow interpretability and modular transferability. By true modularity we mean a clear separation of tasks between the experts.

1.2 Contributions

The first step towards achieving true task modularity was to understand how the tasks are decomposed in existing MoE methods. Our initial systematic experiment showed that original MoE end-to-end training of experts and gate fails to find an optimal task decomposition, in terms of performance and expert usage, even for simple datasets such as MNIST. When instead we pre-trained the experts with specific subtasks and we saw that it improves the performance and results in equitable expert usage. The experiment thus showed that the end-to-end combined training of experts and gate leads to suboptimal results.

This encouraged me to look into alternate training algorithms and architectures for MoE to improve both performance and expert usage which results in true modularity. I started by decoupling the training of experts and the gate using a dual temperature training algorithm. In this method the expert was trained with a soft gate distribution and the gate was trained with a hard gate distribution. This allowed a fair distribution of the samples to the experts during the inital part of the training which gave each expert an opportunity to learn the tasks.

Since the gate fails to learn an optimal task decomposition but is required for conditional computation, could we instead train the experts without the gate and then train the gate to just route to the optimal expert? This lead me to trying two no-gate methods: (a) the *loudest expert* algorithm where the experts are trained using their own estimation of loss for the sample, choosing the one with the lowest loss during training and inference; and (b) the *peeking expert* algorithm where the expert with the highest prediction probability for the sample target is chosen during training. A gate is later trained with the pre-trained experts to enable conditional computation. The peeking method outperformed benchmark MoE architectures both in terms of performance and expert usage.

When training the gate and expert end-to-end, for each batch, the gate makes the routing decision on its own without looking at how the experts perform on the current batch. This leads to the gate and expert learning from the input distribution separately. So we designed a novel attentive gate architecture where the gate's expert distribution computation for the batch depends on the expert's predictions on the same batch. We achieved this by adding attention weights and computing the gate distribution using scoring mechanism similar to that proposed by Bahdanau et al. (2015). The attentive gate architecture again outperformed benchmark

MoE architectures.

Yet another completely novel approach was to train the gate to learn each expert's log loss instead of just the distribution. This seemed a more useful quantity for the gate to learn. Also, small improvements in expert performance requires a large shift in gate probability for that expert to subsequently select that expert. Instead when learning log losses it is much faster to learn their small increments or decrements. This approach performed better than some of the benchmark MoE architectures.

Besides novel MoE training algorithms and architectures we also developed a novel soft constraint to distribute the samples based on their similarity. It is reasonable to assume that samples of the same task has similar feature distributions and hence are similar. Similarity can be measured using any reasonable distance function. *Euclidean* distance is one such simple distance function. We saw that this soft-constraint performed as well as or better than the benchmark $L_{importance}$ soft-constraint by Shazeer et al. (2017) that aims at equal distribution of samples across all available experts while using less experts. Table 1.1 provides a summary of how each of the methods introduced in this thesis is trained. *Distilled gate* is explained in Section 2.3.

Table 1.1: Comparing and summarizing the training methods introduced in the thesis

Training Method	**End-to-End Training**	**Distilled Gate**	**Regularization**	
			$\mathbf{L_{importance}}$	$\mathbf{L_s}$
Decoupling training of experts and gate	$softmax$ without temperature for gate, $softmax$ with temperature for experts	No	Yes	Yes
Loudest expert	only experts trained without a gate	No	No	No
Peeking expert	only experts trained without a gate	Yes	No	No
Attentive gating	no $softmax$ for experts or gate, $softmax$ only for attentive score	Yes	No	No
Expert loss gating	$softmax$ for experts, no $softmax$ for gate as it predicts expert log loss	No	No	No

To summarize, the main contributions of the thesis are:

- Our finding and clear presentation of two crucial problems in training MoE models: (1) that original MoE training methods lead to inequitable and unintuitive task decompositions that have both poor error and loss; and (2) how the tasks are distributed among the experts is relevant to both their performance and scalability.

- Five novel MoE training algorithms and architectures that outperform benchmark MoE architectures
- A novel soft-constraint that provides a flexible data-driven approach to regulate gate sample distribution to the experts that uses less experts than the benchmark $L_{importance}$ regularization.

Let us now deep dive into the state of the art in modular neural networks, understand how the tasks are currently distributed among the modules and go into the details of our novel MoE training algorithms, architectures and soft-constraint.

Chapter 2

Background of Gated Modular Neural Networks

There are many different modular neural network architectures inspired by modularity in biological systems (Ballard, 1987) or creating natural abstractions of concepts or actions (Knoblock, 1990; Sutton et al., 1999). Gated modular neural networks (GMNN) are currently the most successful MNNs. GMNNs are inherently more interpretable since the gate can learn insightful problem decomposition, individual modules can learn simpler functions appropriate to the decomposition and errors can be attributed either to gating or to individual modules.

The modules of GMNN are commonly referred to as experts as each module learns a specific subtask well. Mixture of Experts (MoE) is the simplest of the wider class of gated architectures, introduced by Jacobs et al. (1991b,a), where data-flow is dynamically configured among the experts, according to the input, by the gate. Different MoE architectures can be realized by different combinations of expert outputs described in Section 2.1.

Hinton (1999) alternatively proposed a product of experts architecture which combines the outputs of experts multiplicatively.

Since I am interested in interpretable results I decided to look more closely at what the MoE architecture learns, as it linearly combines the outputs of the individual experts. To the best of our knowledege there is no existing work that delves into the potential for interpretability and transferability in GMNNs.

2.1 Mixture of Experts (MoE)

The MoE architecture, shown in Figure 2.1, was introduced by Jacobs et al. (1991b) over 20 years ago. It has since been successfully applied to learning problems such as reinforcement learning (Gimelfarb et al., 2018), transfer learning (Mihai and Lascarides, 2017), building large computationally efficient neural networks for language models and machine translation (Shazeer et al., 2017; Rajbhandari et al., 2022), continual learning (Veniat et al., 2021; Hihn and Braun, 2022) and learning multiple domains, such as image classification, machine translation, and image captioning, concurrently (Kaiser et al., 2017).

MoE is a gated modular neural network architecture. It consists of modules, called *experts*, and a *gate*. The experts and the gate are simple neural networks. The experts compute functions that are useful in different regions of the input space. The output of an expert, for each sample, is either the learnt class distribution for a classification problem or the learnt regression function output for a regression problem.

The output of the gate is a vector of weights, one for each expert. The weights determine how much an expert contributes towards an MoE's prediction for a sample. This is called *conditional computation* as only some experts are computed conditioned on the gate probabilities. Conditional computation is an important feature of an MoE as it makes training and inference faster. The gate learns a sample based expert selection policy. It allocates samples to experts, decomposing the task between experts. Ideally we want the gating network to learn a meaningful decomposition of the state space and the experts to learn simpler functions in different parts of the state space that results in better performance of the MoE model.

The MoE model output is some combination or selection of the outputs, weighted or otherwise, of the individual experts. The experts and gate are usually trained together end-to-end. Many different MoE architectures can be realized with different combinations of expert outputs mediated by the gate with different loss functions. The basic architectures are presented here.

2.1.1 Output Mixture Model

Jacobs et al. (1991b) introduced the architecture in Figure 2.1 where the expert networks compete to learn the training patterns, and the gating network mediates this competition.

After training, expert networks 1, 2 and 3 compute different functions that are useful in different regions of the input space.

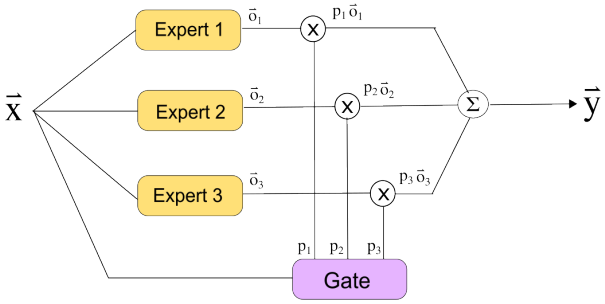


Figure 2.1: Architecture of mixture of networks

Let the vectors $\mathbf{o_1}, \mathbf{o_2}$ and $\mathbf{o_3}$ denote the outputs of the three expert networks. The gating network decides whether expert 1, 2 or 3 is currently applicable. Scalars p_1, p_2 and p_3 denote the 3 output units of the gating network. In general, the architecture may contain any number of expert networks. If there are M expert networks, then the gating network has M output units. The output of the entire architecture, $\hat{\mathbf{y}}$, is the expected sum of the outputs of the individual experts:

$$\hat{\mathbf{y}} = \sum_{i=1}^{M} p_i \cdot \mathbf{o_i} \tag{2.1}$$

and the loss L is:

$$L = l(y, \hat{\mathbf{y}}) \tag{2.2}$$

where y is the desired output and l is a loss function.

Since the output is a sum of proportions of the outputs of the experts, the experts are tightly coupled. Change in weights of one expert changes the residual error and hence affects the weights of all other experts. The *output mixture model* could seem to be not truly realizing conditional computation. In practice, however, the probabilities for some experts are small enough to be neglected. Those expert outputs need not be computed and so indeed does enable conditional computation.

2.1.2 Gate Choice or Stochastic Model

In their subsequent work, Jacobs et al. (1991a), introduced a gating network that makes a stochastic decision of which expert output should be selected during training. The output of the entire architecture, y is therefore one of $\mathbf{o_1}$ to $\mathbf{o_M}$ expert outputs sampled according to the distribution learnt by the gate network. The sampling during training allows for a more exploratory selection of the experts. The loss L is computed as the expected sum of the individual expert losses as shown in Equation 2.5.

During inference the MoE prediction, $\hat{\mathbf{y}}$, is one of $\mathbf{o_1}, \ldots, \mathbf{o_M}$ expert outputs corresponding to the expert that has the highest gate probability as shown in Equation 2.4.

$$I = \arg\max_{i \to M} \mathbf{p} \tag{2.3}$$

$$\hat{\mathbf{y}} = \mathbf{o_I} \tag{2.4}$$

$$L = \sum_{i=1}^{M} p_i \cdot l(y, \mathbf{o_i}) \tag{2.5}$$

where $\mathbf{p}$ is the gate probability output for a sample, y is the desired output and l is a loss function. Notice that in this loss function, each expert is required to produce the whole of the output vector rather than a residual. As a result, the goal of a local expert on a given training case is not directly affected by the weights within other local experts. Since only one expert output is used for MoE prediction this architecture provides true conditional computation.

2.1.3 Top-k Model

Shazeer et al. (2017) introduced the *top-k* gating where only the top-k values of the gate *pre-softmax* output, $\mathbf{z}$, are kept and all the rest are set to $-\infty$. When we compute the *softmax* of $\mathbf{z}$, to get the gate output, p, then all the values of $\mathbf{z}$ set to $-\infty$ become 0. This is because $p_i = softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}, i \in \{1, \ldots, M\}$ where M is the number of experts. When z_i is $-\infty$ then we have $exp(z_i) = exp(-\infty)$ which is 0. This results in $softmax(z_i) = 0, \forall z_i = -\infty$.

Hence, the derivatives corresponding to the gate output, $\mathbf{p}$, whose values are 0 are also 0 during back propagation. This implies that for a given sample only k experts will be selected and trained, which results in good sparsity due to conditional computation. Equation 2.6 defines the *top-k* gating selection. The most common k values used in current literature are *top-1* and *top-2*.

$$KeepTopK(z,k) = \begin{cases} z_i & \text{if } z_i \text{ is in the top k elements of } \mathbf{z} \\ -\infty & \text{otherwise} \end{cases} \tag{2.6}$$

The output of the entire architecture, $\hat{\mathbf{y}}$, is the expected sum of the outputs of the top-k experts as shown in Equation 2.7.

$$\hat{y} = \sum_{i=1}^{k} p_i \cdot \mathbf{o_i} \tag{2.7}$$

and the loss L is computed as in Equation 2.8, where y is the target class. This is similar to the *output mixture* model, except that only k experts are selected for each sample.

$$L = l(y, \hat{\mathbf{y}}) \tag{2.8}$$

2.1.4 Pre-Softmax Model

We designed another architecture which is also similar to *output mixture model* but where the gate smoothly combines the outputs of each expert network before applying the *softmax* to compute the MoE output $\hat{\mathbf{y}}$. The *softmax* is then applied to the resulting expected sum of the *pre-softmax* layers as shown in Equation 2.9, where M is the number of experts, $\mathbf{p}$ is the gate probabilities for the sample and $\mathbf{o_i}$ is the output of expert i.

$$\hat{y} = \text{softmax}\left(\sum_{i=1}^{M} p_i \cdot \mathbf{o_i}\right) \tag{2.9}$$

The loss L is computed as shown in Equation 2.10 where l is a loss function and y is the target.

$$L = l(y, \hat{\mathbf{y}}) \tag{2.10}$$

2.1.5 Hierarchical and Multilevel Models

Some problems require multi-level architectures. Jordan and Jacobs (1993) proposed a hierarchical mixture of experts, in which the experts and gate are generalized linear models. Learning is treated as a maximum likelihood problem and the parameters of the architecture are adjusted using Expectation-Maximization (EM) algorithm.

Another composition of multi-level architecture of experts and gates was proposed by Kirsch et al. (2018). They too propose an EM algorithm to learn the parameters of the architecture. The difference between (Jordan and Jacobs, 1993) and (Kirsch et al., 2018) is that while the former has experts only in the first level, the latter is a composition of experts at each level where the choice of the experts at each level is determined by the gate at that level.

2.2 Recent Mixture of Experts Research

Recently, there has been renewed interest in MoE architectures for scaling it to large models leveraging its modularity and conditional computaion. Combining experts has been applied to learning problems like reinforcement learning (Gimelfarb et al., 2018; Vasic et al., 2020), transfer learning (Mihai and Lascarides, 2017), and to build outrageously large computationally efficient neural networks (Shazeer et al., 2017; Wang et al., 2018; Rajbhandari et al., 2022). We will discuss some of the main problems being tackled in MoE.

Expert specialization in MoE: Much of the MoE research so far has concentrated on the performance of the MoE model and not on how the task is decomposed between the experts. Mittal et al. (2022) and Krishnamurthy and Watkins (2021) arrived at the same conclusion that the original MoE training methods indeed lead to poor expert specialization and that a good task decomposition results in better expert specialization and better performance. In Mittal et al. (2022) the authors use synthetic data distributions to analyse module collapse and expert specialization in modular neural networks. They use four different modular architectures

with different degrees of expert specializations and compare them. The least specialized is the modular architecture where the experts learn the specialization through end-to-end training and the most specialized is the one where the experts are assigned specific tasks and do not learn the expert task assignment. They also propose different metrics to measure collapse and expert specialization. Their experiments show that end-to-end training of modular networks is prone to module collapse and has poor expert utilization. They however did not investigate the problem in real datasets.

Chen et al. (2022) have presented the theoretical aspects of learning in an MoE architecture which is valuable to understanding how the MoE can learn to decompose the input space, but they did this using the *top-1* training method and a regularization. In the we show that we can achieve a good input space partition without regularization and load balancing techniques.

Optimal task decomposition among the experts by the gate has been a pesky problem as we will discover in Chapter 3. Makkuva et al. (2019) designed an algorithm to train the MoE with global consistency guarantees to avoid local minimas. However, their method requires preprocessing and/or transforming the input/outputs which may not work for all data types.

Shazeer et al. (2017) proposed the *top-k* expert selection algorithm which has been the dominating method in recent works (Riquelme et al., 2021; Fedus et al., 2022; Rajbhandari et al., 2022; Lepikhin et al., 2021). It provides high gate sparsity as for each sample only k experts are used. k is normally set to 1 or 2. Hence it is computationally very effcient and allows scaling to large number of experts.

Expert specialization with regularization: Since the end-to-end MoE training provides no incentive for an equitable sample distribution to the experts, auxiliary losses were added as regularizations by Lewis et al. (2021); Shazeer et al. (2017). However, both their methods simply use all available experts even when it may not be required for the task. In our experiments we compare our algorithms's performance to MoE model trained with Shazeer et al. (2017)'s $L_{importance}$ regularization as it is a more generic regularization than that proposed by Lewis et al. (2021) which uses tokenized inputs.

Many load balancing methods for distributing the batch across the experts have been introduced (Shazeer et al., 2017; Fedus et al., 2022). But they aim to equally distribute the samples across the experts. This may not always be desirable. For example, when the dataset is unbalanced.

Token based routing: There have been quite a few token based routing approaches recently (Fedus et al., 2022; Kudugunta et al., 2021; Lepikhin et al., 2021; Lewis et al., 2021; Riquelme et al., 2021; Zhou et al., 2022; Zuo et al., 2022). Kudugunta et al. (2021) proposed a task aware routing. This requires knowledge of how to distribute the tasks which may not always be available. (Fedus et al., 2022; Lepikhin et al., 2021; Riquelme et al., 2021; Zhou et al., 2022; Zuo et al., 2022) have added sparsity to transformer architectures by replacing the dense feed forward layer with MoE for vision and language problems. They propose different token routing strategies. But they each aim to distribute the tokens equally among the experts which may not be suitable for all problems. (Fedus et al., 2022; Lepikhin et al., 2021; Riquelme et al., 2021; Zuo et al., 2022) use the *top-k* algorithm to route to the experts, while Zhou et al. (2022) let each expert choose k tokens. Since these methods use transformers they may not be suitable for all problems. They use only a few experts in the transformer dense layer which are simple one or two feed forward layers. All their approaches are based on the inputs to the MoE being either text or image tokens. Most of the learning in these methods is done in the transformer attention layers which are not part of the MoE.

Attentive gating: To the best of our knowledge our attentive gate architecture is novel. Pfeiffer et al. (2021) have used attention like mechanism to combine adapters learnt from a pre-trained model for multi-task learning. Their architecture is different from an MoE as there is no gate. Another related work we found was by Liu et al. (2020), who have used the attention mechanism in the MoE gate to focus the gate on different aspects of the input and target images. The gate then learns good segmentation of the input images and assigns the different segments to different experts. Their approach is similar to the original MoE where the gate independently decides the tasks to be assigned to the experts by attending to the data. Our approach learns the gate's expert distribution by attending to the experts.

No-gate MoE: There has been some work recently (Lewis et al., 2021; Roller et al., 2021; Zuo et al., 2022) that have realized the problem with end-to-end expert and gate training and have proposed no-gate MoE training alternatives. Lewis et al. (2021) propose an MoE architecture without gates. They instead formulate and solve the optimal equal assignment of tokens of text data to the experts as a linear assignment problem. They too aim at equal assignment of tokens to experts. Their work is the closest to our work on no-gate training. But unlike their approach we aim for an optimal task dependent distribution of samples and

not an equal sample distribution. Roller et al. (2021) changed the transformer's feedforward layer to hash to different weights for different tokens in the sequence. They showed that their method outperforms BASE Layers (Lewis et al., 2021) and Switch transformers (Fedus et al., 2022). Zuo et al. (2022) have removed the gate and shown that randomly selecting the experts actually works, but with two forward passes per batch and a consistency regularization. They also have a larger variance in their inference due to the stochasticity without the gate. However all these methods use transformers with tokenized inputs and replace the dense feed forward layer with MoE which may not be suitable for all problems as discussed earlier. Our no-gate method does not make any assumption about the inputs or architectures of the experts and the gate. It however currently is limited to supervised learning tasks with single (not hierarchical) layer of experts and gate. We hope to extend this later to more complex MoE architectures.

Other: Hazimeh et al. (2021) propose an alternate differentiable method to compute gate expert selection weighting using a differentiable selection of experts using a novel binary encoding formulaiton. Their method is best suited for problems like multi-task learning where usually we want to share expert parameters. They have shown that their method outperforms *top-k*. But since *top-k* is the more dominant strategy currently used we chose it as our benchmark. Also, in our Step 2 we can use any existing method to train the gate. So we should be able to use their method in our Step 2.

2.3 Model Distillation

The concept of ***model distillation*** was introduced by Hinton et al. (2015). Hinton referred to the idea of transfering the knowledge in a large ensemble model to a single model as *distillation*. It has since been used in the literature more broadly as the process of going from a larger model with more parameters to a smaller model with lesser parameters (not necessarily to a single model). We use this interpretation of *model distillation*. We also use ***reverse distillation*** in this thesis, which is the process of going from a smaller model with less parameters to a bigger model with more parameters.

Chapter 3

Understanding Mixture of Experts Architectures

All the MoE research so far, however, only show performance results of the overall architecture and not what each expert learns. It is important to learn what each expert learns in order to assess the interpretability and transferability of the MoE architectures. The intuition is, the better the expert specialization the easier it is to attribute errors to the expert and the gate, and experts should be better suited for transferability. In order to learn how the tasks are distributed among the experts in the various architectures, we performed a suite of experiments to better understand what each expert is learning. The goal of the experiments was to answer the following questions:

1. How are tasks divided into sub tasks by each architecture?
2. Does original MoE training find intuitive task decompositions?
3. Do intuitive task decompositions have better performance?
4. What are the pathological cases?

The results and analysis of the experiments are presented in the following sections.

3.1 How are tasks divided into sub tasks by the gate in the MoE architecture?

A good task decomposition, is one in which either: (1) the gating network learns a meaningful decomposition of the input space into regions with natural 'rules'. For example, for a classification task it would use different experts to predict different classes; or (2) each expert learns non-intersecting functions or subsets of the task which implicitly satisfies case (1). Hence, it is important to see how the task is allocated to the experts by the gate and what the experts and the gate are learning in order to analyze interpretability.

So, we implemented the different MoE architectures outlined in Section 2.1[1]. Since it is always best to start simple, we started with a toy classification problem and a toy regression problem from Kirsch et. al. Kirsch et al. (2018).

3.1.1 Toy classification problem

The dataset for the toy classification problem is a $2D$ mixture of Gaussians with 6 components, shown in Figure 3.1.

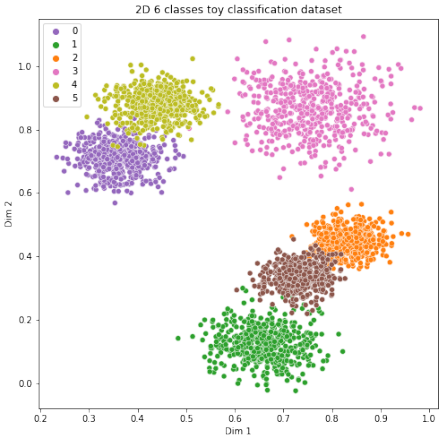
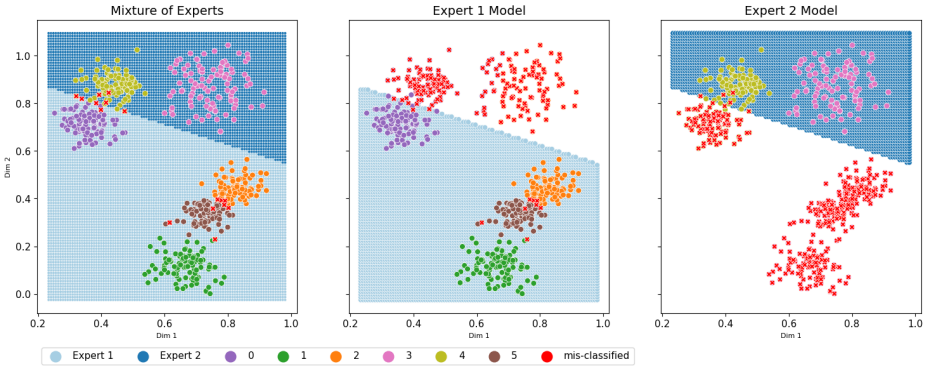


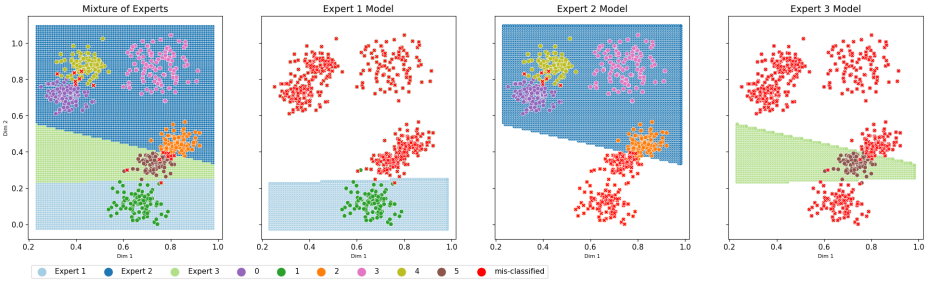
Figure 3.1: Toy classification dataset with simple 2D 6 classes Gaussian mixture

Each component corresponds to a class and so we have a 6 class classification problem. There are $2,400$ training samples and 600 test samples. The MoE model consists of simple linear

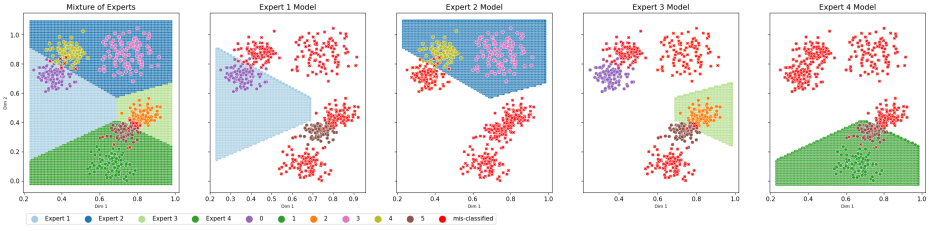
[1] we did not implement the multi-level experts and gates but used the EM algorithm from Kirsch et. al. Kirsch et al. (2018) to train a single level of experts and one gate to compare it to training with the other architectures that use back propagation.



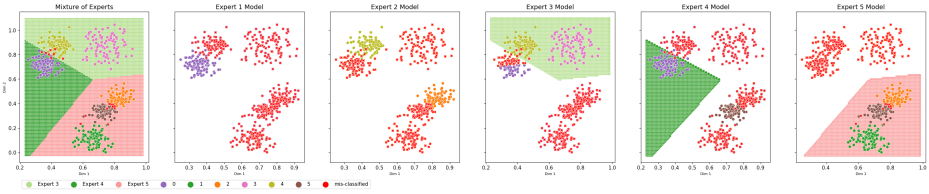
(a) 2 experts



(b) 3 experts



(c) 4 experts

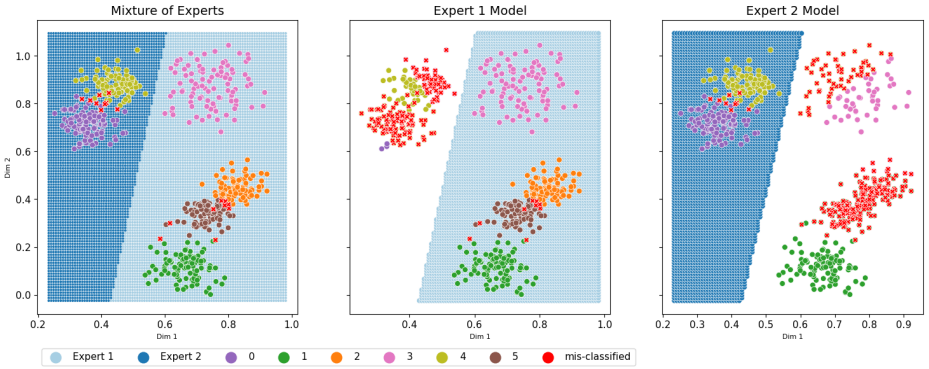


(d) 5 experts

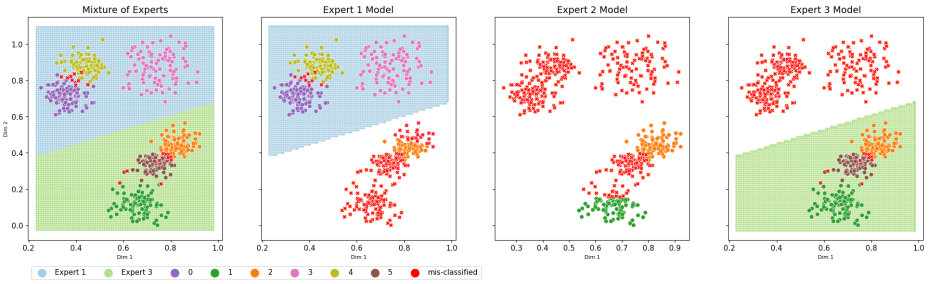
Figure 3.2: Toy classification test data class predictions by the trained *output mixture* model. The shaded regions are the decision boundaries of the individual experts for the samples sent to them by the gate. Red 'x' are the mis-classified samples.

experts and gate networks. We used RMSProp optimizer for updating the parameters. The inputs to all the expert and gate networks are the same. Each of the models was run for different number of experts ranging from 2 to 5.

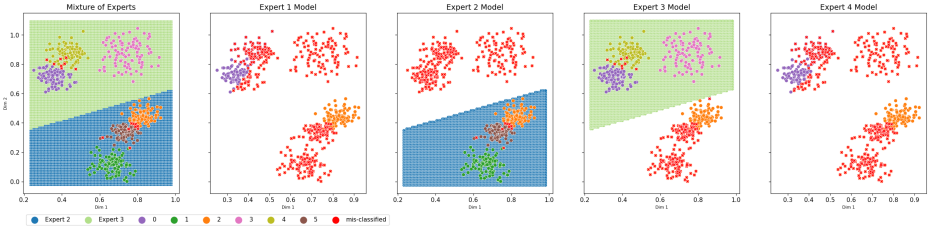
We did not try 6 experts for the simple toy classification problem as that is the same number as the classes. We want to use less experts than the classes and still get good performance. Having 6 experts would just increase the number of overall parameters of the MoE models and result in each expert learning a single class and hence underutilizing the expert. This however



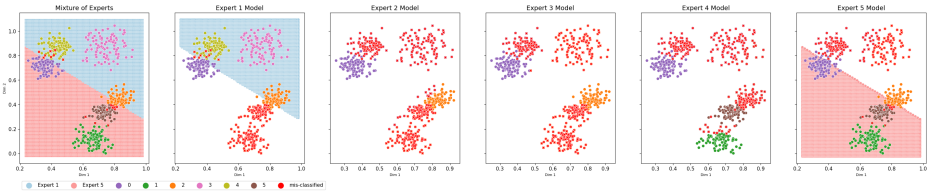
(a) 2 experts



(b) 3 experts



(c) 4 experts

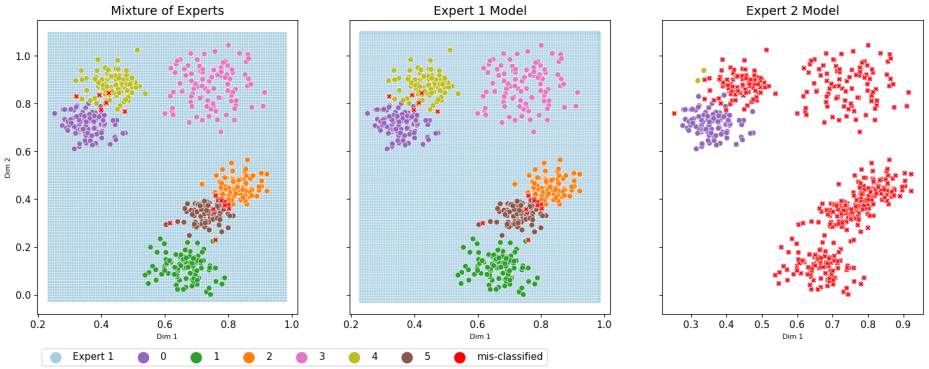


(d) 5 experts

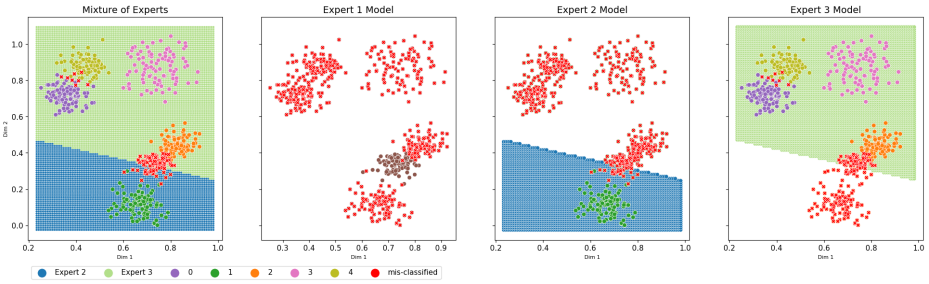
Figure 3.3: Toy classification test data class predictions by the trained *stochastic* model. The shaded regions are the decision boundaries of the individual experts for the samples sent to them by the gate. Red 'x' are the mis-classified samples.

may not be the case for more complex datasets where the data of each class could have distinct sub-clusters or sub-problems. In such cases having experts $\geq$ the number of classes may result in better task decomposition and performance. The experts and the gate are jointly trained on the loss for the corresponding architectures.

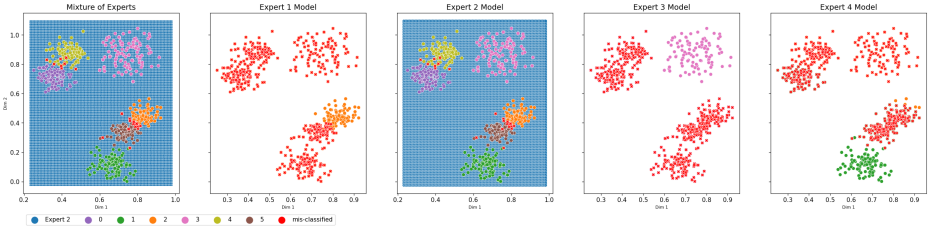
Figures 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7 show the test class predictions of the toy dataset with the learnt *output mixture*, *stochastic*, *top-1*, *top-2*, *pre-softmax* and *EM* models for different number of experts. The first column of each of these figures is the output of the MoE model. The



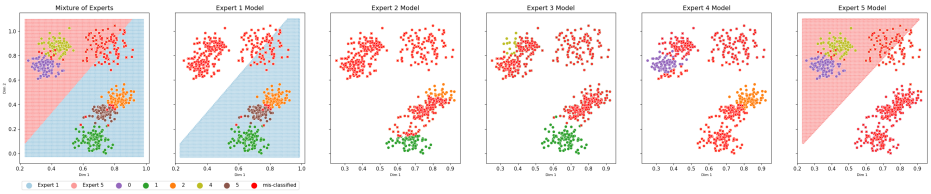
(a) 2 experts



(b) 3 experts



(c) 4 experts

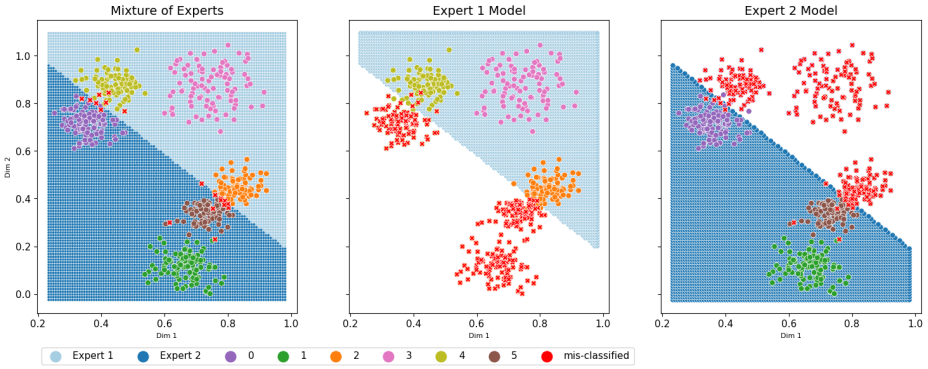


(d) 5 experts

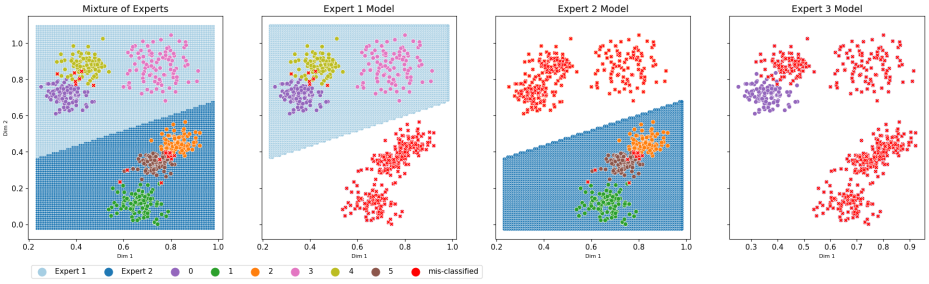
Figure 3.4: Toy classification test data class predictions by the trained *top-1* model. The shaded regions are the decision boundaries of the individual experts for the samples sent to them by the gate. Red 'x' are the mis-classified samples.

subsequent columns are the test class predictions of the toy dataset for each of the individual experts. Mis-classfications are marked as red 'x'. The shaded regions are the input space to which the samples sent to expert by the gate belong.

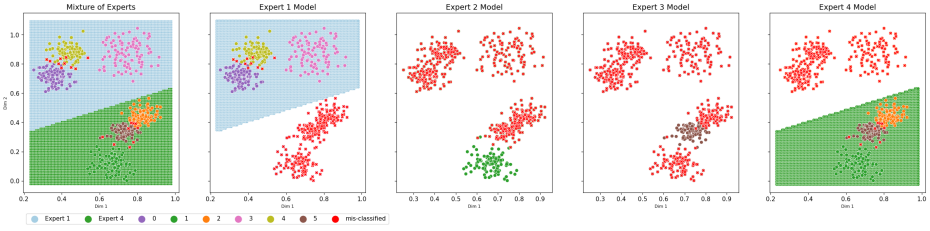
Figures 3.2, 3.3 and 3.5 show that the *output mixture*, *stochastic* and *top-2* models, consistently partition the data cleanly for the toy classification dataset. The test class predictions by the selected experts shows that the classification task of 6 classes is divided among the experts. That is, each expert, if selected, learns to classify a subset of the classes as expected.



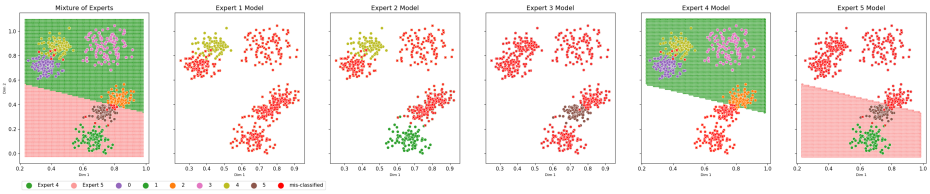
(a) 2 experts



(b) 3 experts



(c) 4 experts

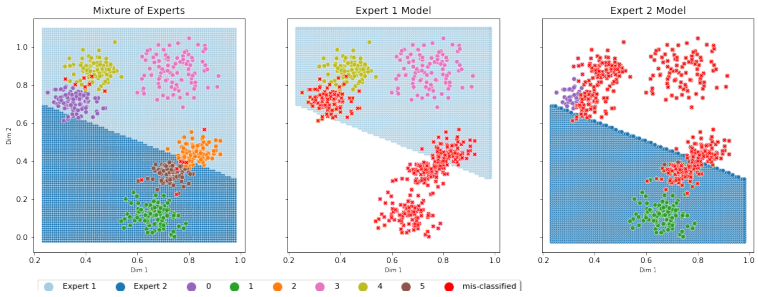


(d) 5 experts

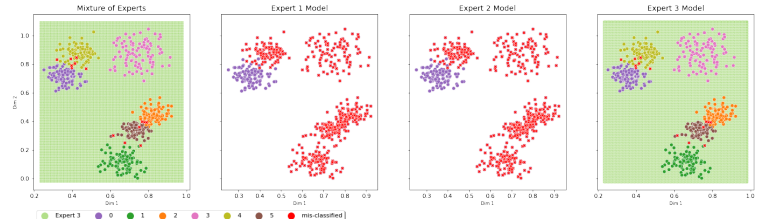
Figure 3.5: Toy classification test data class predictions by the trained *top-2* model. The shaded regions are the decision boundaries of the individual experts for the samples sent to them by the gate. Red 'x' are the mis-classified samples.

Figures 3.4, 3.6 and 3.7 show that *top-1*, *pre-softmax* and *EM* models do not necessarily result in a good task decomposition even for the easily classifiable toy dataset. We see cases of module collapse, such as for 2 and 4 experts with top-1 method, where all classes are learnt by just one expert and hence there is no task decomposition.

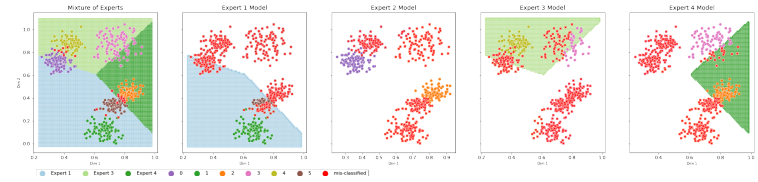
Let us now look at the performance of the models with 3 and 5 experts. Tables 3.1 and 3.2 compare the different models using the benchmarking metrics (discussed in Chapter 4) test accuracy, mutual information between experts and MoE output, $I(E;Y)$, gating sparsity



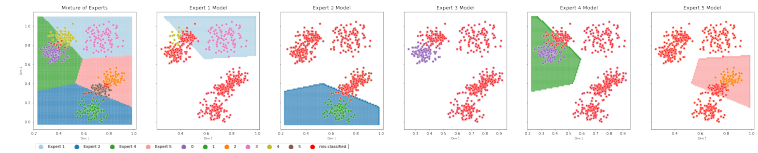
(a) 2 experts



(b) 3 experts



(c) 4 experts

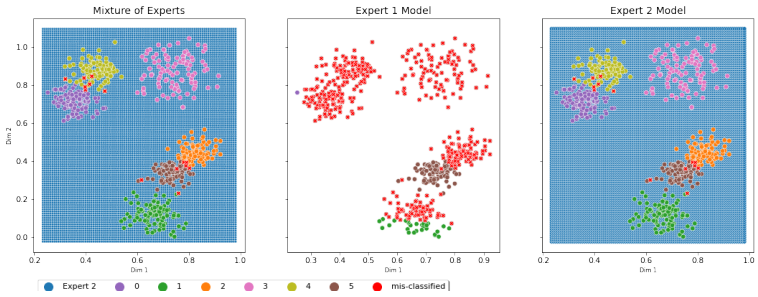


(d) 5 experts

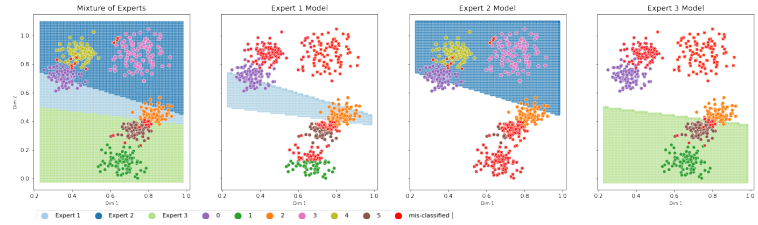
Figure 3.6: Toy classification test data class predictions by the trained *pre-softmax* model. The shaded regions are the decision boundaries of the individual experts for the samples sent to them by the gate. Red 'x' are the mis-classified samples.

H_s, and expert usage on the test set for the toy classification dataset. The tables show that *output mixture*, *stochastic*, *top-2* and *pre-softmax* models perform consistently well. *EM* model performance depends on the number of experts. *top-1* model performs the worst. Though the *pre-softmax* and *EM* model perform well, they do not result in a clean decomposition as seen in Figures 3.6 and 3.7. The *EM* model has the slowest training time compared to the other architectures.

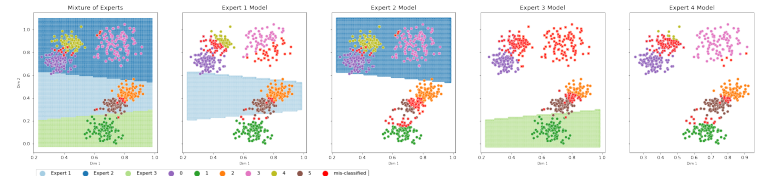
We choose *output mixture*, *stochastic* and *top-2* models as the benchmark models as they have both good performance and good task decompositions. We also included *top-1* for completion of the top-k model.



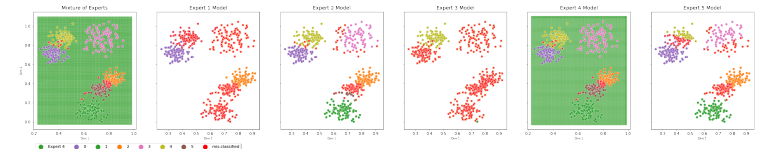
(a) 2 experts



(b) 3 experts



(c) 4 experts



(d) 5 experts

Figure 3.7: Toy classification test data class predictions by the trained *EM* model. The shaded regions are the decision boundaries of the individual experts for the samples sent to them by the gate. Red 'x' are the mis-classified samples.

Table 3.1: Performance of the models on the test set for the toy classification dataset with 3 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
output mixture	**96.83**	1.271	0.166	1.269
stochastic	96.66	1.000	0	1.000
top-1	82.00	0.751	0.160	0.792
top-2	96.33	1.000	0.004	1.000
pre-softmax	94.50	1.518	0.997	1.560
EM	72.16	0	0.019	0.023

3.1.2 Toy regression problem

The toy regression problem was used as an example by Kirsch et al. (2018). The data $\mathcal{D}$ contains samples (x_n, y_n). x_n are generated from a mixture of Gaussians with two compo-

Table 3.2: Performance of the models on the test set for the toy classification dataset with 5 experts. Best result is highlighted.

Experiment	**Test Accuracy**	**I(E; Y)**	**H_s**	**H_u**
output mixture	96.83	1.459	0.093	1.457
stochastic	96.83	0.830	0	0.999
top-1	80.50	0.984	0.133	0.983
top-2	97.00	0.933	0.130	0.932
pre-softmax	96.16	1.527	0.831	1.539
EM	**97.66**	0	0.008	0.009

nents with uniform latent mixture probabilities, $p(s_n = 1) = p(s_n = 2) = 1/2$ such that $x_n|s_n \approx \mathcal{N}(x_n|\mu_{s_n}, \sum_{s_n})$. Depending on the component s_n, the target y_n is generated by linearly transforming the input x_n according to:

$$y_n = \begin{cases} Rx_n & \text{if } s_n = 1 \\ Sx_n & otherwise \end{cases} \tag{3.1}$$

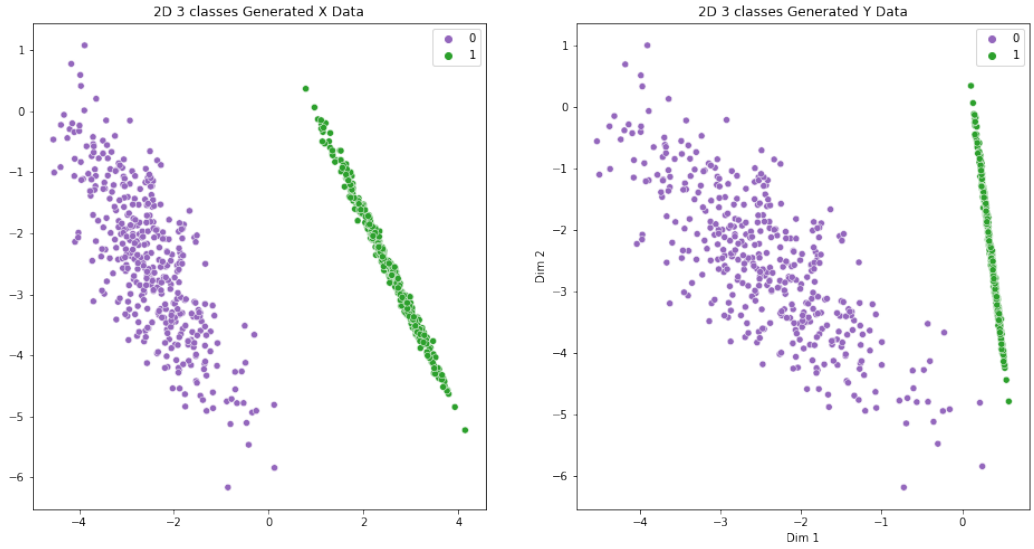


Figure 3.8: Toy regression data. Left plot shows the x data and the right plot shows the y data which is the rotated and translated x data

R is a rotation matrix and S is a scaling matrix. Figure 3.8 shows x_n and y_n data using the following R and S matrices:

$$R = \begin{bmatrix} 0.9081 & 0.4188 \\ -0.4188 & 0.9081 \end{bmatrix}$$

$$S = \begin{bmatrix} 0.0603 & 0.0000 \\ 0.0000 & 0.9340 \end{bmatrix}$$

Each modular network consists of 2 experts each with experts having one hidden layer with 2 inputs and 2 outputs. This corresponds to a 2×2 weight matrix corresponding to the rotation and scaling matrices. The goal of the experiment was to see how well each type of modular network model learns the rotation and scaling matrices. The biases are set to zero and their gradient is not computed. Only the weights are learnt. The gate has the same architecture as the expert but both weights and biases are learnt. After training we see that one expert's weights are similar to R and the other to S. So the gate decomposes the task using the experts. The learnt weights of the experts for the *EM*, *output mixture* (in the regression problem the *output mixture* and *pre-softmax* models are the same) and *stochastic* models are:

Expectation Maximization Model

Expert 1 weights

$$W_1 \approx R = \begin{bmatrix} 0.9081 & 0.4188 \\ -0.4188 & 0.9081 \end{bmatrix}$$

Expert 2 weights

$$W_2 \approx S = \begin{bmatrix} 0.0603 & -0.0000 \\ 0.0000 & 0.9340 \end{bmatrix}$$

Output Mixture Model

Expert 1 weights

$$W_1 \approx R = \begin{bmatrix} 0.9136 & 0.4152 \\ -0.4201 & 0.9089 \end{bmatrix}$$

Expert 2 weights

$$W_2 \approx S = \begin{bmatrix} 0.0588 & -0.0005 \\ 0.0002 & 0.9340 \end{bmatrix}$$

Stochastic Model

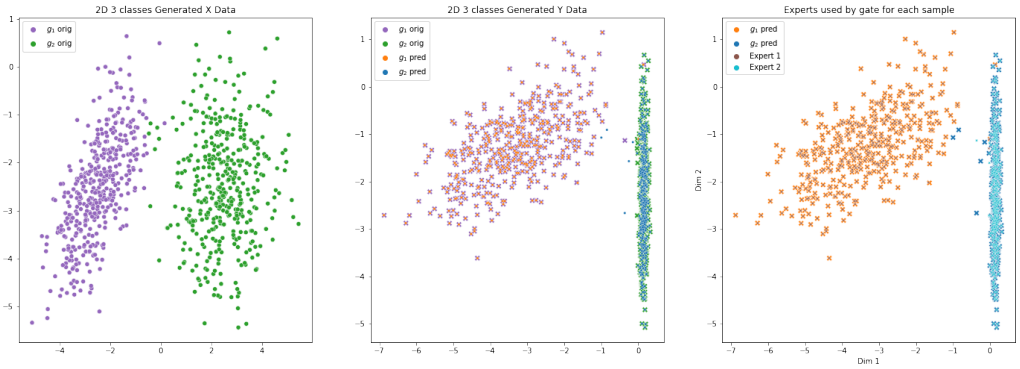
Expert 1 weights

$$W_1 \approx R = \begin{bmatrix} 0.9235 & 0.4045 \\ -0.4213 & 0.9104 \end{bmatrix}$$

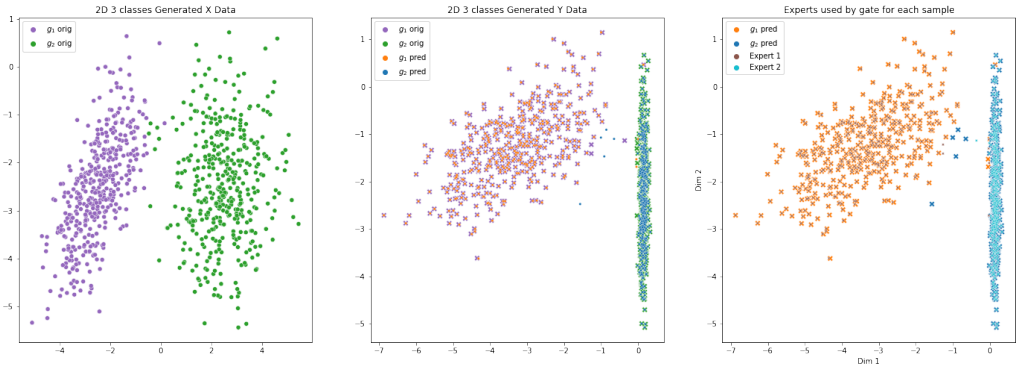
Expert 2 weights

$$W_2 \approx S = \begin{bmatrix} 0.0625 & 0.0019 \\ -0.0003 & 0.9337 \end{bmatrix}$$

Figure 3.9 shows that each expert learns one of the two task parameters.



(a) MoE *output mixture* model



(b) MoE *stochastic* model

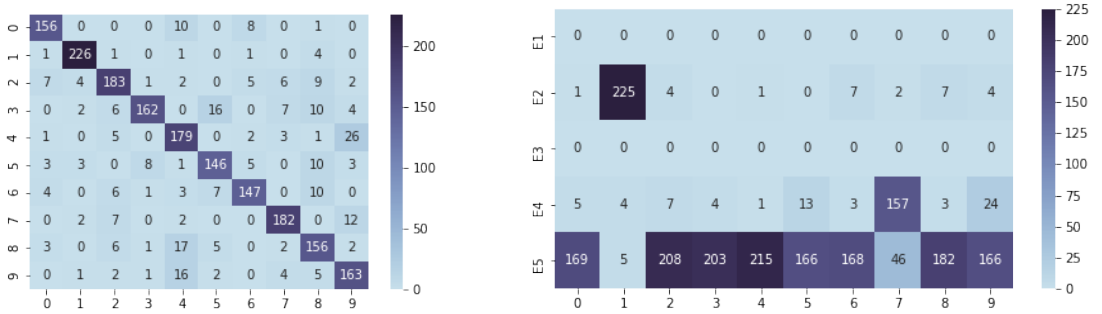
(c) MoE *EM* model

Figure 3.9: Comparison of original/predicted data and experts used for subtasks

3.1.3 MNIST and FashionMNIST Datasets

Let us proceed to the more complex problem of MNIST (LeCun and Cortes, 2010) digits classification and again check if the cases are satisfied. We used $\approx 10,000$ training samples and $2,000$ test samples containing all the digits. Each expert and the gate is a simple convolutional network with a single convolutional layer and 2 hidden layers with ReLU activation. From Figure 3.10a, the confusion matrix of the predictions by the MoE *output mixture model*, we see that the model accuracy is good. However, from the expert utilization table in Figure 3.10b, we see that only 3 of 5 experts are used with expert 5 used for most of the samples. For some digits, such as, 7 and 9, both experts 4 and 5 are selected. So, for the more complex problem, the gate task decomposition is not interpretable as it does not achieve either case (1) or (2).

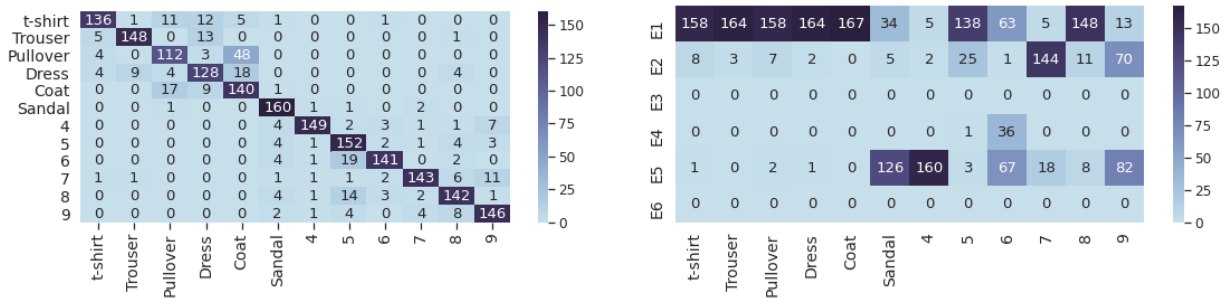
In Figures 3.10a and 3.10b the legend on the right shows the color gradient corresponding to the number of samples per class in the test dataset approximately. For the MNIST dataset we selected $2,000$ test samples of 10 digits and each digit has ≈ 200 samples.



(a) (b)

Figure 3.10: (a) Confusion matrix of predictions of the MNIST test data by MoE *output mixture model* (b) Experts used by the gate for classification of each digit.

Since the MNIST dataset is homogeneous, as it contains only digits images, we thought we should try with a dataset that contains clearly very different sets of images, and see if the gate can better decompose the task between the experts. We created such a dataset by combining the FashionMNIST (FMNIST) Xiao et al. (2017) and MNIST datasets. We chose 6 classes, $[t{-}shirt, trouser, pullover, dress, coat, sandal]$, from FMNIST and 6 classes,$[4, 5, 6, 7, 8, 9]$, from MNIST and combined the data to create one dataset of 12 classes. For combined MNIST and FMNIST data we selected $2,000$ test samples of 12 classes and we have ≈ 166 samples per class. This is reflected in the legend on the right of the Figures 3.11a and 3.11b.



(a) (b)

Figure 3.11: (a) Confusion matrix of predictions of the combined FMNIST and MNIST test data by MoE *output mixture model* (b) Experts used by the gate for classification of each image.

Figures 3.11b shows that the gate is still not able to do a good task decomposition as it uses expert 5 to learn class *sandal* from FMNIST and digit 4 from MNIST and expert 1 to learn a mix of classes from FMNIST and MNIST. Hence, we see that a good task decomposition in MoEs is not always guaranteed even in a seemingly trivial case where the images of FMNIST and MNIST are clearly quite different from each other. Let us now analyse the possible reasons for bad task decompositions.

3.2 Does an intuitive task decomposition actually exist and can the gate learn it?

The simplest method to train an MoE is to train the gate and experts at the same time, 'end-to-end', by gradient descent. During training the gating probabilities, for each sample, determine which experts get trained on that sample. That is, gating interacts with training and in effect experts are trained only when they are chosen by the gating network. Existing MoE architectures trained 'end-to-end' do not decompose the task intuitively among the experts as we saw in Section 3.1.3. The question we are trying to answer is: does the 'end-to-end' MoE training find a gating decomposition that performs well for the task, even though it seems surprisingly counter-intuitive? Or, is the search for gating decomposition simply bad?

We designed an experiment, summarized in Figure 3.12, to answer these questions. What we need for this is: (1) a gate trained with un-trained experts, using the original MoE model, resulting in unintuitive task decomposiiton as in Section 3.1.3; and (2) a gate trained with experts pre-trained with custom intuitively plausible partitions of the dataset. We then use

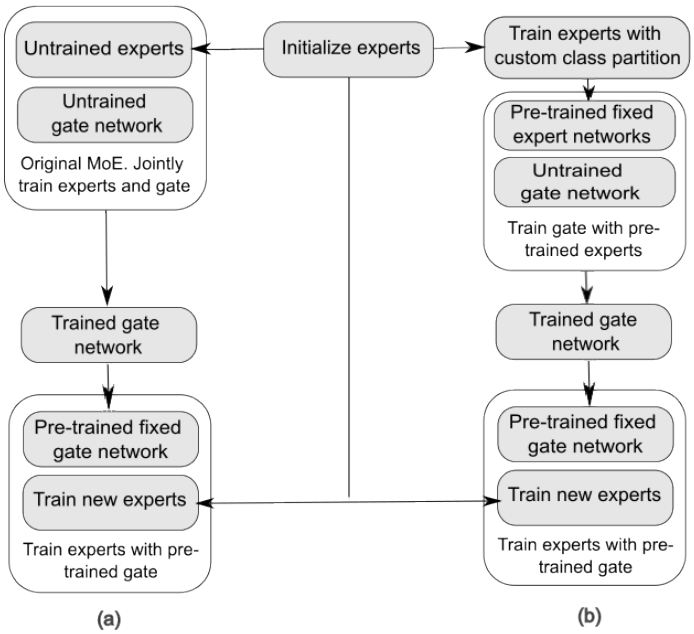
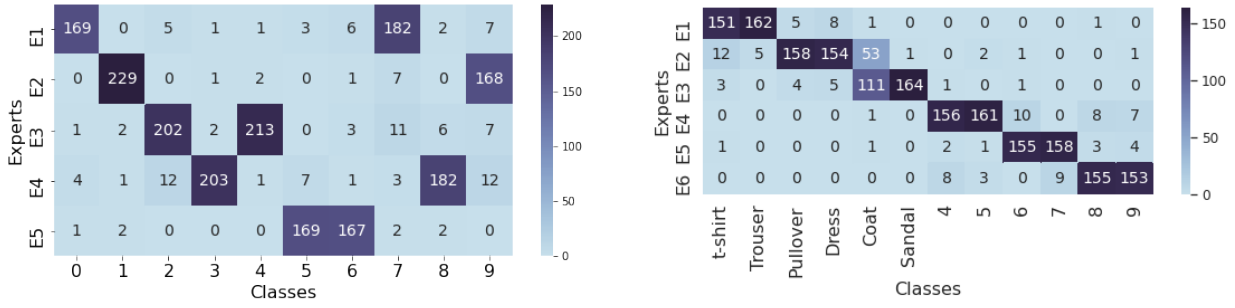


Figure 3.12: Experiment designed to analyse if intuitive task decompositions have better performance. Refer to the Table 3.3b for results of the experiment.

each of these two pre-trained gates to train a new set of experts with the same decomposition of the task as the experts the gates were trained with. This enables us to check the performance of the gate task decompositions for an unintuitive partition vs an intuitive partition.

Firstly, let us define a more intuitive task decomposition for the MNIST dataset and determine if the gate can learn this decomposition. We split the 10 digits into 5 sets of 5 pairs of digits, such as $\{[0,7],[1,9],[2,4],[3,8],[5,6]\}$. We used 5 experts, each of which was trained with only data samples of one of the 5 pairs of digits. So the pairs of digits are distributed equally among the experts.



(a) MNIST (b) FMNIST and MNIST

Figure 3.13: Expert selection table of models trained with experts pre-trained on custom splits of the classes: (a) MNIST: {[0,7], [1,9], [2,4], [3,8], [5,6]} and (b) combined FMNIST and MNIST: {[t-shirt,Trouser], [Pullover,Dress],[Coat,Sandal],[4,5],[6,7],[8,9]}

We then fixed the parameters of these pre-trained experts and trained the gate with them.

From the gate expert selection table in Figure 3.13a, we see that the gate can indeed learn to select the correct expert for each digit and hence learn an intuitive task decomposition. Figure 3.13b shows the gate expert selection table for one split of the combined FMNIST and MNIST dataset, trained in the same way as with the MNIST dataset. We again see that the gate can learn to select the correct expert for each class in the combined dataset.

We then fixed the parameters of the pre-trained gate and trained the MoE model with the pre-trained gate and new experts. Both the pre-trained gates decomposed the tasks exactly as in Figures 3.10b and 3.13a respectively for the MNIST dataset and similarly as Figures 3.11b and 3.13b for the combined FMNIST and MNIST dataset. Hence we see that a gate can learn an intuitive task decomposition.

Let us now check the training loss and test error of the models with intuitive and unintuitive task decompoitions. Tables 3.3a and 3.3b show the average training loss and average test error, both averaged over 5 runs of the experiment for MNIST and combined FMNIST and MNIST datasets. We see that the model trained with pre-trained experts has a lower training loss than the model trained with un-trained experts and has a lower error rate for both datasets.

Table 3.3: Comparison of average training loss and test error for MoE models: (a) with inequitable task decompositions; and (b) with equitable task decompositions, from the experiment detailed in Figure 3.12, for MNIST and combined MNIST and FMNIST datasets.

Models	**Test Error**	**Train Loss**
(a)	0.12	0.19
(b)	0.08	0.05

(a) MNIST

Models	**Test Error**	**Train Loss**
(a)	0.15	0.28
(b)	0.10	0.13

(b) MNIST and FMNIST

The experiment shows that intuitive task decompositions do exist with much better performance. The gate, however, does not learn them when both experts and the gate are jointly trained ‘end-to-end’. The gate initially finds a poorly performing and unintuitve task decomposition and reinforces that throughout the training. If we have prior knowledge of a good task decomposition then it would be best to pre-train the experts on these sub-tasks and then train the gate. Typically we do not know a plausible task decomposition and it is what we wish to find, but ‘end-to-end’ MoE training fails to do so, even in this simple case.

3.3 What are the pathological cases of task decomposition?

3.3.1 Only some experts are used

We have already seen this case in Figures 3.10b and 3.11b where only a few experts are used. In end-to-end expert and gate training we saw in Section 3.1.3 that there is no incentive for the gate to distribute samples equitably to all the experts. Hence, some experts could be starved. This could also occur when more experts than subtasks are used.

3.3.2 One expert learns all the classes

This is a known problem and Kirsch et al. (2018) refer to it as *module collapse*. This occurs when the gate selects the same expert for all the samples. In this case the MoE output does not depend on the gate. There is no true modularity as this is the same as the single model.

Module collapse is observed when: (1) a single expert is complex enough to solve the task, the gate sends all the samples to a single expert. Simpler experts encourage the gate to choose different experts for the different subtasks as shown by Chen et al. (2022); (2) one expert is better initialised than the other experts and learns faster and hence is assigned all the samples by the gate.

3.3.3 One expert learns only one class

This case has not been previously explicitly reported and occurs when an expert classifies all inputs as the same class. In this case the gate does all the classification and it is not really modular as the experts are trivial.

All these cases suggest poor allocation of data samples to the experts, by the gate, subsequently leading to either over use or starvation of experts. For interpretability and transferability we need to avoid all these cases.

3.4 Summarizing the Issues with MoE Training

The simplest way to train a MoE is to train the gate and experts at the same time, 'end to end', by gradient descent, but this can lead to poor results. This is because gating interacts with training. Hence, poor gating decisions lead to two issues that significantly affect interpretability and the ability to perform transfer learning: (1) Poor allocation of training examples leading to poor expert utilization and starvation; and (2) unpredictable task decomposition.

Ideally we want the gating network to learn a meaningful decomposition of the state space, and the experts to learn simpler functions adapted for particular circumstances. During training the gating probabilities on each example determine which experts get trained on that example: in effect experts are trained only when they are chosen by the gating network. Unfortunately this leads to the simplest expert learning early and grabbing the training data, starving more complex expert models of training data.

In recent work, Shazeer et al. (2017) proposed a soft constraint approach by adding a regularization term to the loss that measures the batch-wise coefficient of variation of the gate output probabilities to avoid module collapse. They define an importance factor, I, over all samples X in a batch, computed in Equation 3.2, that measures the relative importance of the expert to the batch.

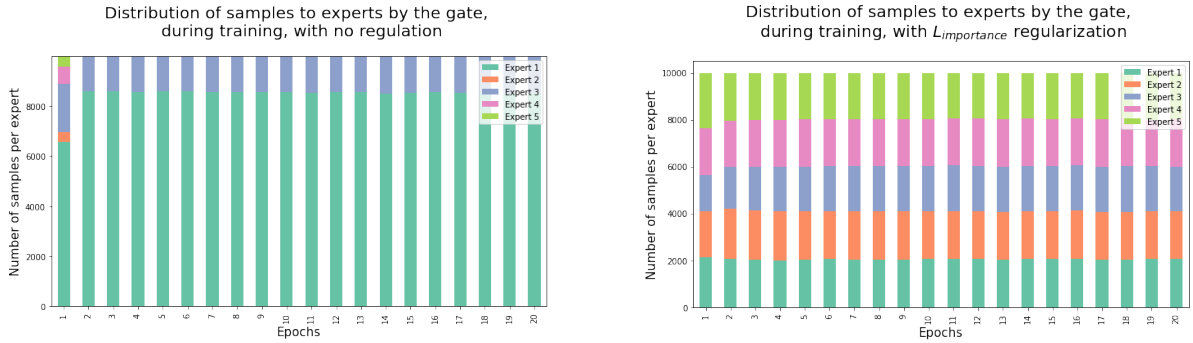
$$\mathbf{I} = \text{Importance}(X) = \sum_{x \in X} \mathbf{p_x} \tag{3.2}$$

where $\mathbf{p_x}$ is the gate probability distribution for sample $x \in X$. X is the number of samples in the batch.

The goal was to assign equal importance to all experts for a batch which is achieved by enforcing a low coefficient of variation (CV) of the importance, $\mathbf{I}$, values by minimizing the loss term in Equation 3.3. $w_{importance}$ is a hand tuned parameter. $CV = \sigma/\mu$, where σ is the standard deviation (std) and μ is the mean computed on $\mathbf{I}$.

$$
\begin{aligned}
L_{importance}(X) &= w_{importance} \cdot CV(\mathbf{I}) \\
&= w_{importance} \cdot \frac{\text{std}(\mathbf{I})}{\text{mean}(\mathbf{I})}
\end{aligned} \tag{3.3}
$$

Figure 3.14a shows that original MoE model suffers from module collapse where expert 1 very quickly gets all the samples while the other experts starve. Figure 3.14b shows that with $L_{importance}$ regularization, we have an equitable distribution of samples to the experts. Both models have the same architecture and were trained with 10,000 samples of MNIST data.



(a) (b)

Figure 3.14: Distribution of 10,000 MNIST samples by the gate to the experts during training: (a) without regularization and (b) with $L_{importance}$.

The $L_{importance}$ regularization however simply distributes the samples equally over all available experts. This may not always be desirable. For example, this method will try to use all experts while just a few experts may be sufficient for the task. We further discuss this in Chapter 9.

In this thesis we propose novel MoE training algorithms and architectures that improve expert usage. We use the $L_{importance}$ as a benchmark regularization.

Chapter 4

Performance Metrics for MoE

Accuracy or error is not sufficient to measure the performance of an MoE as we are also interested in measuring gating sparsity and expert usage. We will here define the information theoretic performance metrics we use to analyse the training of the MoE. These metrics measure how well the gate distributes the samples to the experts and how well it utilizes the experts.

4.1 Measuring Gating Sparsity

Conditional computation is an important feature of the MoE. Sparser gating probabilities are desirable because they result in better conditional computation. The sparsity per sample can be measured by the average per sample expert selection entropy, H_s, in Equation 4.1, over a batch. N is the number of samples in a batch and $\mathbf{p_i} = [p_{i1}, \ldots, p_{iM}]$ are the gate probabilities for M experts, for each sample i, $i \in \{1, ldots, N\}$. A low value of H_s indicates sparse gating probabilites and hence better conditional computation.

$$H_s = \frac{1}{N} \sum_{i=1}^{N} H(\mathbf{p_i}) \tag{4.1}$$

4.2 Measuring Expert Utilization

Ideally we want the sub-tasks of the task to be distributed equitably between the experts to avoid module collapse. This will require the average gate probabilities for each of the experts, over all the samples, to be equal. The distribution of the experts over the samples can be measured by the entropy of the average gate probabilities over all samples in a batch, H_u, as in Equation 4.2. A high H_u indicates a more equitable gate probability distribution and hence better utilization of experts. A low H_u indicates unequal utilization of experts. For example, in the case of module collapse, when all samples get sent to the same expert, that expert's average gate probability is 1.0. The probabilities of all the other experts will be zero. This will result in $H_u = 0$.

$$H_u = H\left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i\right) \tag{4.2}$$

4.3 Measuring model output dependency on expert selection

We also introduce a new metric to measure the dependency of the class distribution Y on the gate's expert selection distribution E. An equitable gate task decomposition among experts results in a high mutual dependence between Y and E.

In the case of module collapse, one expert does all the work and the gate does not contribute to solving the task. There is then no dependency between E and Y. In the case where each expert is assigned just one sub-task, the gate does all the work. There is then a higher dependency between E and Y. Hence, the more equitable the task distribution between the experts the higher the dependence between E and Y.

The mutual dependency between E and Y can be measured by computing their mutual information, I(E;Y), as shown in Equation 4.3, where $H(E)$ is the marginal entropy of E, $H(Y)$ is the marginal entropy of Y and $H(E,Y)$ is the joint entropy of E, Y. Higher $I(E;Y)$ values indicate better dependence between E and Y and subsequently more equitable task decomposition.

$$I(E;Y) \equiv H(E) + H(Y) - H(E,Y) \tag{4.3}$$

Since we do not have the true marginal and joint probabilities of E and Y, we compute them empirically. The sample sizes are large enough to not introduce significant estimation bias.

Mutual information, $I(E;Y)$, between the class distribution Y and the gate expert distribution E, can be computed by first computing the count c_{ij}, where $i \in \{1\ldots,M\}$ and $j \in \{1,\ldots,K\}$, M is the number of experts and K is the number of classes in the task. c_{ij} is the number of times expert E_i is selected for samples of class Y_j. c_{ij} is computed for each expert, for each class and hence we have an $M \times K$ count matrix C.

Table 4.1: Matrix C of count of number of times E_i is selected for class Y_j

Count(E,Y)	Y_1	$\ldots$	Y_K
E_1	c_{11}	$\ldots$	c_{1K}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
E_M	c_{M1}	$\ldots$	c_{MK}

From C, we can compute the batchwise joint and marginal probabilities of E, Y in Table 4.2 using Equations 4.4, 4.5 and 4.6, where N is the total number of samples in a batch:

Joint Probabilty $P(E,Y)$:

$$P(E{=}E_i, Y{=}Y_j) = p(E_i, Y_j) = c_{ij}/N \tag{4.4}$$

Marginal Probability $P(E)$:

$$P(E{=}E_i) = p(E_i) = \sum_{j=1}^{K} p(E_i, Y_j) \tag{4.5}$$

Marginal Probability $P(Y)$:

$$P(Y{=}Y_j) = p(Y_j) = \sum_{i=1}^{M} p(E_i, Y_j) \tag{4.6}$$

We can now compute the required entropies in Equation 4.3 from quantities computed in Table 4.2 using Equations 4.7, 4.8, 4.9. Subsequently we can compute the mutual information $I(E;Y)$

Table 4.2: Joint and marginal probabilities of E and Y

$P(E,Y)$	Y_1	$\dots$	Y_K	$P(E_i)$
E_1	$p(E_1,Y_1)$	$\dots$	$p(E_1,Y_K))$	$p(E_1)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
E_M	$p(E_M,Y_1)$	$\dots$	$p(E_M,Y_K)$	$p(E_M)$
$P(Y_j)$	$p(Y_1)$	$\dots$	$p(Y_K)$	1

as in Equation 4.3.

$$H(E) = \sum_{i=1}^{M} -p(E_i)\log_2\, p(E_i) \tag{4.7}$$

$$H(Y) = \sum_{j=1}^{K} -p(Y_j)\log_2\, p(Y_j) \tag{4.8}$$

$$H(E,Y) = \sum_{i=1}^{M}\sum_{j=1}^{K} -p(E_i,Y_j)\log_2\, p(E_i,Y_j) \tag{4.9}$$

Chapter 5

Decoupling Training of Experts and Gating

5.1 Motivation

Module collapse, where only one expert is used, is a result of bad batch distribution by the gate. The gate's output, $p(e_i|x_j)$, is the probability of the sample x_j being sent to the expert e_i. The gate quickly starts preferring one well performing expert and starving the other experts. This prevents a more fair learning opportunity for the experts as we saw in Section 3.3.

The reason for this is that when experts and gate are trained together end-to-end, the gate's *softmax* output of the expert probabilities converges to high probabilities for the preferred expert and very low probablities for the other experts. This leads to data starvation of the experts with low gate probabilities, which then results in them not learning anymore.

In order to prevent this hard distribution of samples we propose decoupling the gate and expert learning by smoothing the gate's expert probability distribution for learning the experts, but use the harder distribution to learn the gate. What are hard and soft distributions? We can illustrate this with an example. Say we have a classification problem with 3 classes. A hard probability distribution for them would be $[0.97, 0.01, 0.02]$ where the model is very confident that the class is 1. A soft distribution, on the other hand would be $[0.65, 0.15, 0.25]$. While a hard distribution may be desirable for classification, it is not always desirable for the gate's

expert distribution, especially during initial training when experts are beginning to learn. A softer gate distribution would provide experts a more equitable initial learning opportunity.

A softer distribution can be achieved by regulating the *softmax* by applying a temperature T to its logit input (Hinton et al., 2015). If z_i is the logit for class i then the *softmax* converts it into a probability p_i, using Equation 5.1:

$$p_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{5.1}$$

Normally T is set to 1. Higher value of T produces a softer probability distribution. The *softmax* distribution depends on the differences between the logit values. By dividing the logit with T, we reduce this difference and hence make the distribution softer. For example, let the logit values be $\mathbf{z} = [0.7, 0.2, 0.1]$, then for $T = 1$, $\text{softmax}(\mathbf{z}/T) = [0.4640, 0.2814, 0.2546]$ and for $T = 3$, $\mathbf{z}/T = [0.2333, 0.0667, 0.0333]$ and $\text{softmax}(\mathbf{z}/T) = [0.3752, 0.3176, 0.3072]$. We see that with $T = 3$ the *softmax* distribution is softer and more equitable than with $T = 1$.

5.2 Training experts and gate with dual temperature

We will describe the dual temperature training using the *output mixture* MoE. In Section 2.1.1, we saw that for the *output mixture* method, the loss L on which both the gate and experts are trained end-to-end is computed as in Equation 2.2 when the output of the model is computed as in Equation 2.1, repeated here again in Equations 5.2 and 5.3 where $\mathbf{o_i}$ is the output of expert i, M is the number of experts, p_i is the probability of the gate choosing expert i for the corresponding sample, computed as in Equation 5.1, and y is the target label.

$$\hat{\mathbf{y}} = \sum_{i=1}^{M} p_i \cdot \mathbf{o_i} \tag{5.2}$$

$$L = l(y, \hat{\mathbf{y}}) \tag{5.3}$$

In the dual temperature training we decouple the end-to-end expert and gate training by: (1) defining a different loss L_g for the gate and L_e for the expert using gate probabilities computed

Algorithm 5.1.1: Dual Temperature Training for Output Mixture Model

Input: $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{N}, \mathcal{X} \in \mathbb{R}^u$
epochs $\in \mathbb{N}$
$M \in \mathbb{N}$ /* number of experts */
$K \in \mathbb{N}$ /* number of classes */
$f_i : \mathcal{X} \to \mathbb{R}^K, i \in \{1, \ldots, M\}$ /* expert neural network */
$g : \mathcal{X} \to \mathbb{R}^M$ /* gate neural network */
$l : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ /* loss function */
expert optimizer O_e
gate optimizer O_g
$T > 1.0$
$T_{decay} < 1.0$
$T_{decay\_start} < epochs$
Output: $g(\cdot), f_i(\cdot), i \in \{1, \ldots, M\}, \hat{y} \in \mathbb{R}^K$
for $epoch = \{1, \ldots, epochs\}$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    $o_i \leftarrow f_i(x), i \in \{1, \ldots, M\}$ /* expert outputs */
    $z \leftarrow g(x)$ /* gate logit output */
    $p_i^{'} \leftarrow \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$ /* gate softmax output with temperature */
    $p_i \leftarrow \frac{exp(z_i)}{\sum_j exp(z_j)}$ /* gate softmax output without temperature */
    $\hat{y}^{'} \leftarrow \sum_{i=1}^{M} p_i^{'} \cdot o_i$ /* predicted output for expert loss computation */
    $\hat{y} \leftarrow \sum_{i=1}^{M} p_i \cdot o_i$ /* MoE predicted output */
    $L_e \leftarrow l\left(y, \hat{y}^{'}\right)$ /* expert loss */
    $L_g \leftarrow l\left(y, \hat{y}\right)$ /* gate loss */
    compute expert gradients with L_e
    compute gate gradients with L_g
    update O_e
    update O_g
    if $epoch \geq T_{decay\_start}$ **then**
      $D \leftarrow \left(\frac{1}{(1+T_{decay})}\right) \cdot (epoch - T_{decay\_start})$
      $T \leftarrow T * D$
    end
  end
end

with high and low temperatures as in Equations 5.4 and 5.5 respectively; and (2) have different optimizers for experts and gate. The expert optimizer only optimizes the expert parameters and the gate optimizer only optimizes the gate parameters.

$$L_g = l(y, \sum_{i=1}^{M} p_i \cdot \mathbf{o_i}) \text{ where } p_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{5.4}$$

$$L_e = l(y, \sum_{i=1}^{M} p_i^{'} \cdot \mathbf{o_i}) \text{ where } p_i^{'} = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \text{ for } T > 1 \tag{5.5}$$

z_i is the logit input to the gate's *softmax* layer corresponding to the selection of expert i. During inference the output of the MoE model, $\hat{\mathbf{y}}$, is computed using $T = 1$ as in Equation 5.2. The dual temperature training with *output mixture model* is summarized in Algorithm 5.1.1.

5.3 Experiments

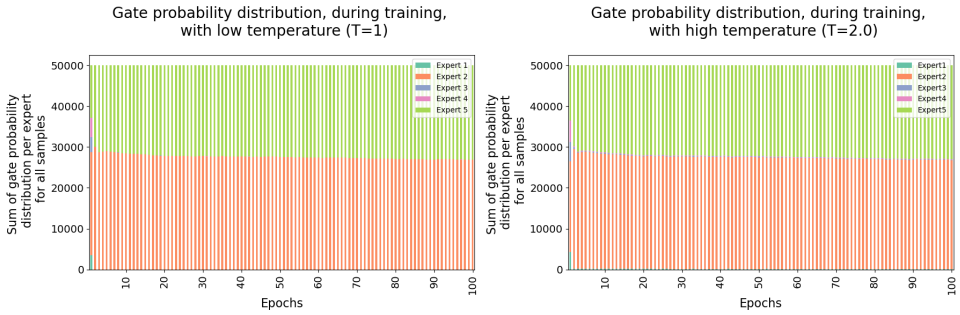
We evaluate the dual temperature training method on the MNIST LeCun and Cortes (2010) dataset. We ran the experiments with 5 experts. Details of expert and gate architectures for MNIST are in Appendix A.1.

All models were trained with Adam optimizer with 0.001 learning rate and 100 epochs. Temperatures used were $T = [1.1, 1.2, 1.3, 1.4, 1.5, 2, 4, 6]$. Each experimnent was run 10 times.

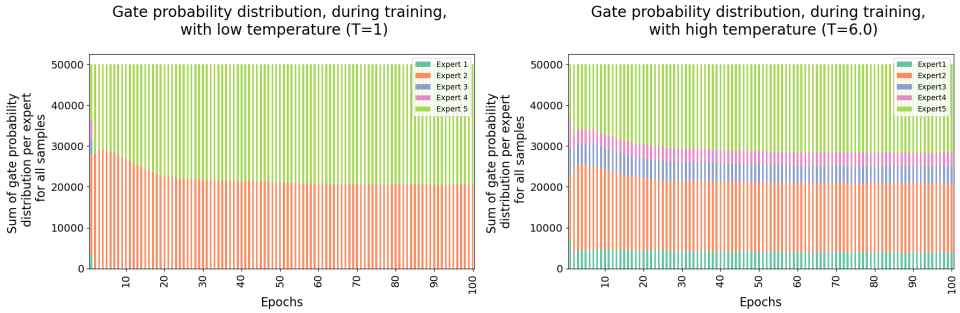
We first tried with the same temperature for all epochs. Then we tried with temperature decay after 25 epochs. The temperature decay factor, D, was computed as $D = (1/(1 + T_{decay})) \cdot (epoch - T_{decay\_start})$ where T_{decay} is the factor by which we want to decay the temperature. We used $T_{decay} = [0.001, 0.01, 0.1]$. T_{decay_start} is the epoch from which the temperature decay should be applied. The temperature of the next epoch T_{i+1} is updated as $T_{i+1} = T_i \cdot D$, where T_i is the temperature at the current epoch i,

Figures 5.1 and 5.2 show gate probability distribution per expert over all the samples for the *output mixture model (OMM)* and *stochastic (STO) model* respectively. They show how the samples are distributed among the experts by the gate while training the model with MNIST data with different temperatures. We see that as we increase the temperature there is a more equitable distribution of samples and hence a better expert utilisation. Since the expert and the gate are training simultaneously, and for each epoch the gate and expert weights are updated the gate probability distribution is different for $T = 1$ when training with different temperatures

for the expert. Hence we see a difference in the top and bottom left figures in Figures 5.1 and 5.2 for $T = 1$.



(a) T=2



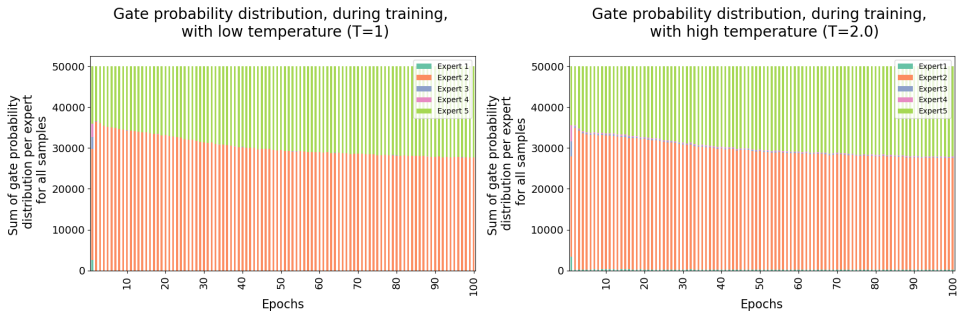
(b) T=6

Figure 5.1: Gate probability distribution during training for *output mixture model* for MNIST dataset. The figure on the left is the gate probability distribution using *softmax* computed with $T = 1$ with which the gate is trained. The figure on the right is the gate probability distribution using *softmax* computed with $T > 1$ on which the experts are trained.

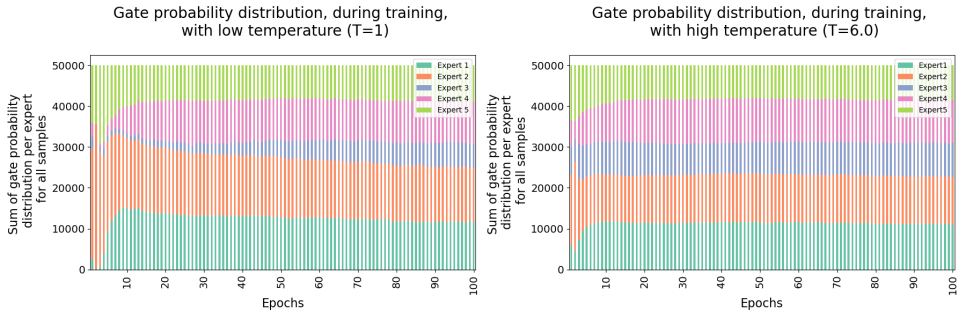
We compared the performance of the dual temperature training method on *output mixture model* and *stochastic model* with the *output mixture model* with $L_{importance}$ regularization. Figures 5.3 and 5.4 compare the training and validation errors for *single model*, *output mixture model*, *stochastic model*, *output mixture model* with $L_{importance}$ regularization and MoE with dual temp training with and without decay for *output mixture model* and *stochastic model*. For $L_{importance}$ we used $w_{importance} = [0.2, 0.4, 0.6, 0.8, 1.0]$.

We see that dual temperature training and validation errors with temperature decay are better than without decay. Dual temperature training with decay for *output mixture model* and *stochastic model* outperforms single model and *output mixture model* without regularization for all $T > 1$. *Output mixture model* and *stochastic model* with dual temperature training with decay have comparable performance to *output mixture model* with $L_{importance}$ regularization for higher values of T.

We now compare the expert usage with dual temperature training. Figure 5.5 shows the



(a) T=2



(b) T=6

Figure 5.2: Gate probability distribution during training for *stochastic model* for MNIST dataset. The figure on the left is the gate probability distribution using softmax computed with $T = 1$ with which the gate is trained. The figure on the right is the gate probability distribution using *softmax* computed with $T > 1$ on which the experts are trained.

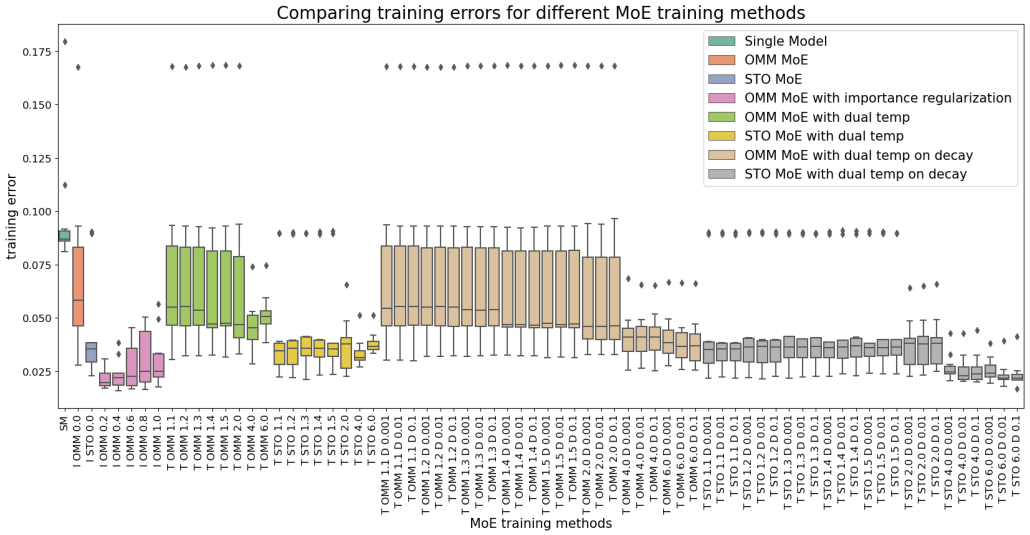


Figure 5.3: Comparing training error for MoE without regularization, MoE with $L_{importance}$ regularization for $w_{importance}$ $I = 0.2 - 1.0$ and dual temperature training with and without decay for temperature $T = 1.1 - 6$ and $T_{decay} = [0.001, 0.01, 0.1]$ for *output mixture model* and *stochastic model* with MNIST dataset.

expert usage entropy, computed using Equation 4.2, over 10 runs for *output mixture model*, *output mixture model* with $L_{importance}$ regularization and training *output mixture model* and *stochastic* model with dual temperature. We can see that *output mixture model* with $L_{importance}$ regularization has the best expert usage. We do see that with dual temperature training the

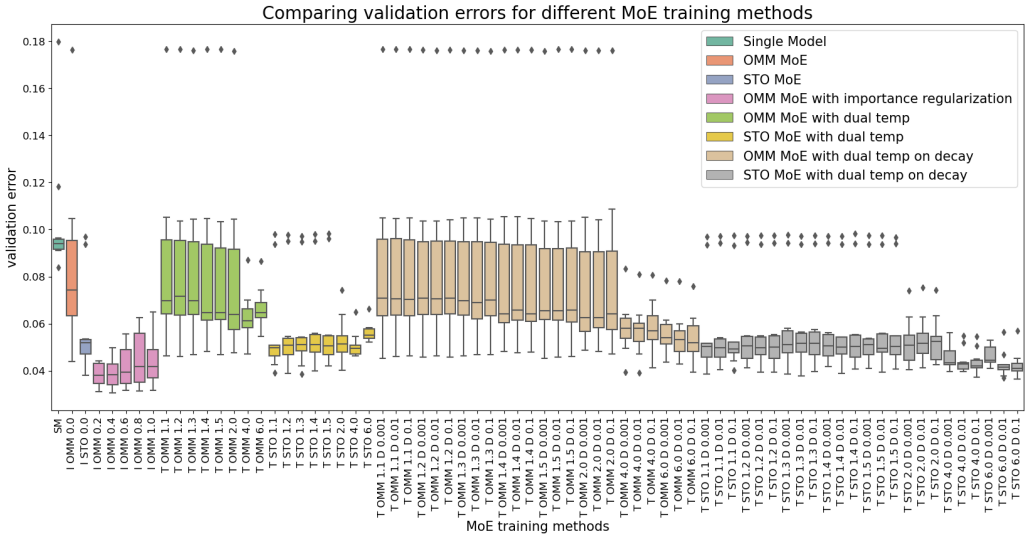


Figure 5.4: Comparing validation error for MoE without regularization, MoE with $L_{importance}$ regularization for $w_{importance}$ $I = 0.2 - 1.0$ and dual temperature training with and without decay for temperature $T = 1.1 - 6$ and $T_{decay} = [0.001, 0.01, 0.1]$ for *output mixture model*) and *stochastic model* with MNIST dataset.

expert usage does improve over *output mixture model* especially for higher temperatures and there is no module collapse.

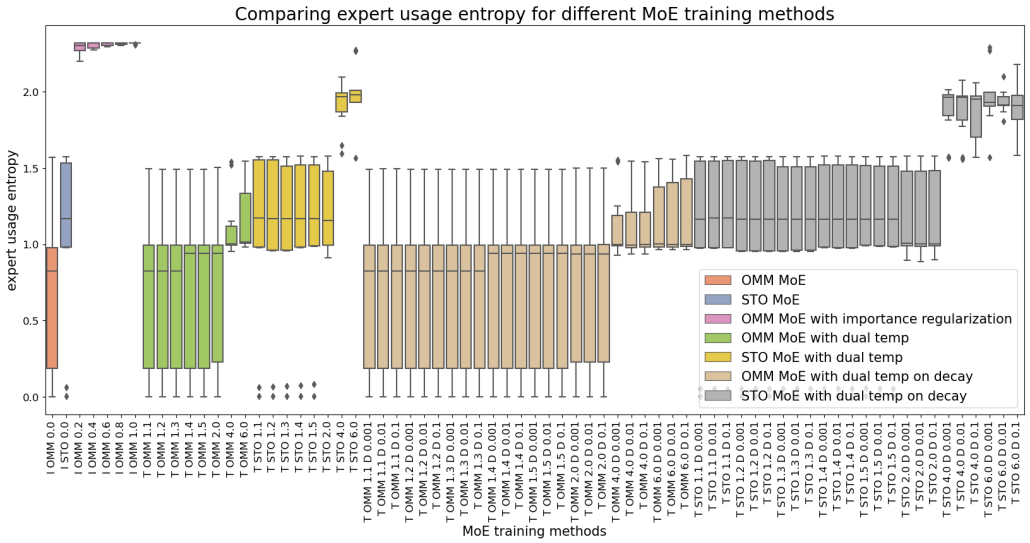


Figure 5.5: Comparing expert usage entropy for MoE without regularization, MoE with $L_{importance}$ regularization for $w_{importance}$ $I = 0.2 - 1.0$ and dual temperature training with and without decay for temperature $T = 1.1 - 6$ and $T_{decay} = [0.001, 0.01, 0.1]$ for the *output mixture model* and *stochastic model* with MNIST dataset.

Figure 5.6 shows the per sample expert usage entropy as computed in Equation 4.1. The lower the entropy the better as it shows higher conditional computation. We see that per sample entropy for training with dual temperature is lower than training with $L_{importance}$ regularization. But this could be because training with dual temperature has a lower expert usage.

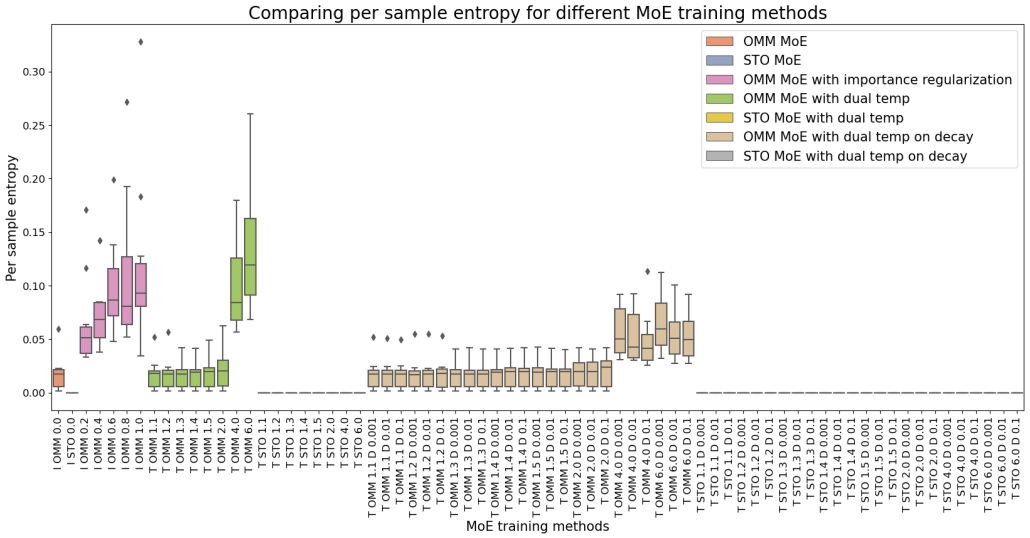


Figure 5.6: Comparing per sample expert usage entropy for MoE without regularization, MoE with $L_{importance}$ regularization for $w_{importance}$ $I = 0.2 - 1.0$ and dual temperature training with and without decay for temperature $T = 1.1 - 6$ and $T_{decay} = [0.001, 0.01, 0.1]$ for the *output mixture model* and *stochastic model* with MNIST dataset.

Figure 5.5 shows the expert selection and model output joint distribution mutual information computed as in Equation 4.3. Higher mutual information is better as indicates a dependency between expert and model output implying a better task distribution between the experts. We see that $L_{importance}$ regularization has higher expert and output mutual information than training with dual temperature. But both *output mixture model* and *stochastic model* have higher mutual information than *output mixture model* without regularization for higher temperatures.

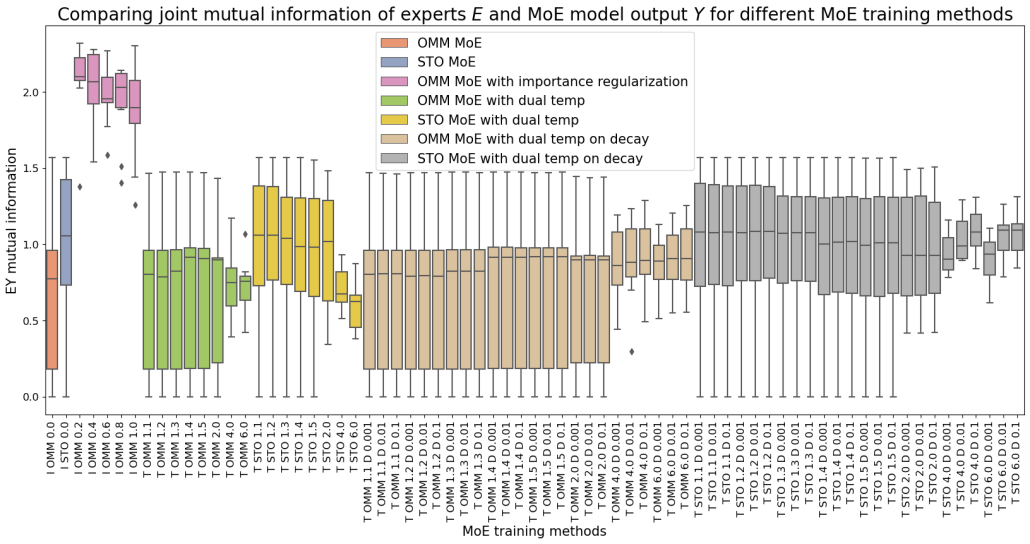


Figure 5.7: Comparing mutual information for MoE without regularization, MoE with $L_{importance}$ regularization for $w_{importance}$ $I = 0.2 - 1.0$ and dual temperature training with and without decay for temperature $T = 1.1 - 6$ and $T_{decay} = [0.001, 0.01, 0.1]$ for *output mixture model* and *stochastic model* with MNIST dataset.

We compared the performance of: (1) single model which has the same architecture as one

expert; (2) original *output mixture* MoE; (3) *stochastic* MoE; (4) *top-1* MoE; (5) *top-2* MoE; (6) *output mixture* MoE with $L_{importance}$ regularization for different values of $w_{importance}$; (7) *output mixture* MoE with dual temperature training with decay; and (8) *stochastic* MoE with dual temperature training with decay.

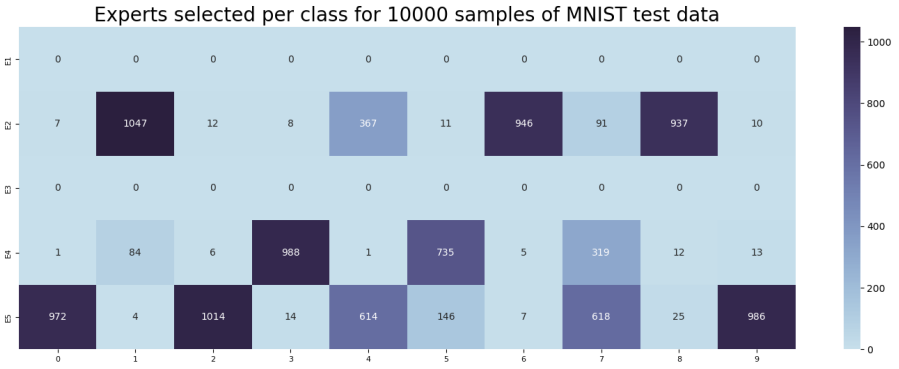
Table 5.1: Performance on the MNIST test set of the models with the minimum validation error with 5 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
output mixture MoE	95.83 ± 0.039	1.568	0.034	1.570
stochastic MoE	96.13 ± 0.021	1.569	0	1.569
top-1 MoE	94.29 ± 0.039	1.000	0	1.000
top-2 MoE	96.67 ± 0.017	1.552	0.029	1.570
output mixture MoE with $\mathbf{L_{importance}}$	**96.87 ± 0.008**	2.280	0.053	2.322
dual temp output mixture model with decay	96.31 ± 0.020	1.937	0.089	1.937
dual temp stochastic model with decay	96.39 ± 0.006	1.919	0.017	1.513

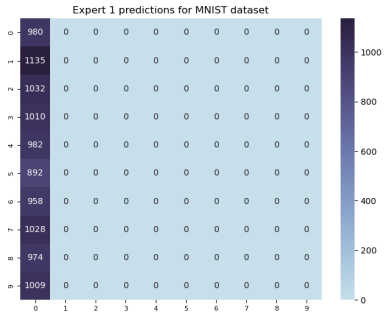
Table 5.1 shows that dual temperature training outperforms *single* and *output mixture* MoE without regularization and is comparable to *stochastic*, *top-1* and *top-2* MoEs. It results in better batch distribution between the experts. But the $L_{importance}$ regularization outperforms the dual temperature training method.

Let us now see what the gate and the experts are learning. Figure 5.8a shows the gate expert selection table per class for MNIST test dataset with 5 experts using the *output mixture model* with temperature decay. We use the model with the best validation error. We see that experts 2, 4 and 5 were selected by the gate for respective classes. The table shows the samples of the digits on which each expert was mostly trained on and has specialized for. For example, expert 2 was trained on samples of digits 1, 4, 6, 7, 8. This is further reiterated by the predictions for each test sample by expert 2 in its confusion matrix shown in Figure 5.8c. The figure shows that for the digits 1, 4, 6, 7 and 8, expert 2 predicts the samples correctly with few errors. But for the other digits, that it is not trained on, it predicts one of the digits it has specialized in. Hence, it fails on digits it is not trained on as expected.

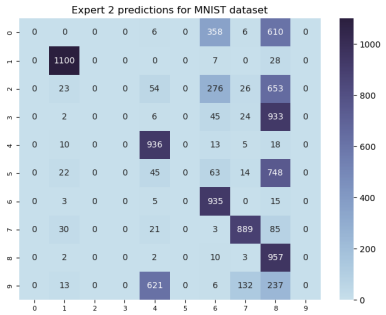
Figures 5.8c, 5.8e and 5.8f show that experts 2, 4 and 5 are all trained on samples of digit 7. The gate chooses each of them for different samples of digit 7. Figures 5.8b and 5.8d show that experts 1 and 3 have been trained on only samples of digit 0. This indicates that the gate stopped selecting them after a few epochs of training or never.



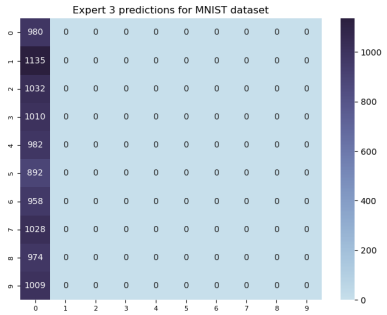
(a) Expert selection table per class with 5 experts



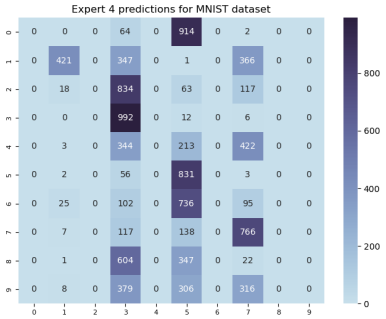
(b) Confusion matrix of expert 1 predictions on all data



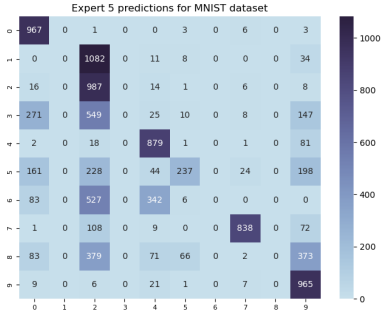
(c) Confusion matrix of expert 2 predictions on all data



(d) Confusion matrix of expert 3 predictions on all data



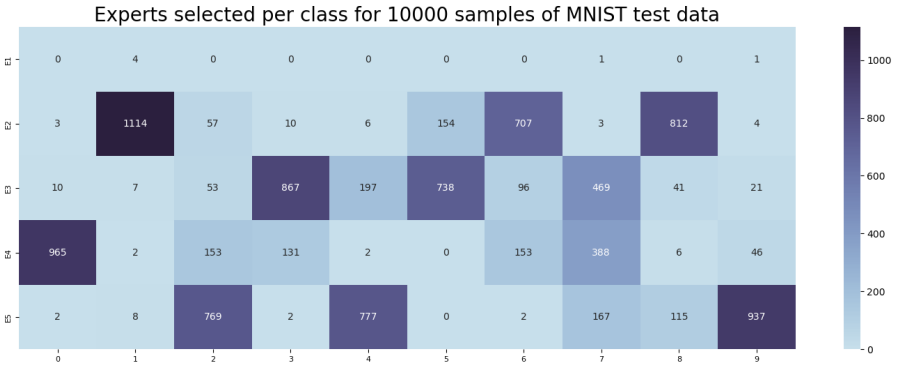
(e) Confusion matrix of expert 4 predictions on all data



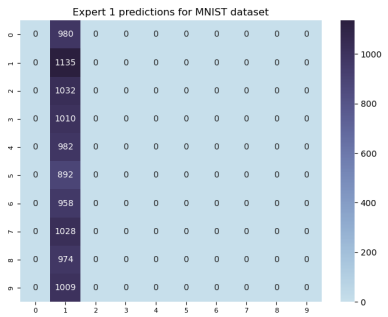
(f) Confusion matrix of expert 5 predictions on all data

Figure 5.8: (a) Expert selection table showing the experts selected per class during inference with MNIST test data with *output mixture* model with dual temperature training with decay. (b),(c),(d),(e),(f) Confusion matrix of predictions of each of the 5 experts for the MNIST test data.

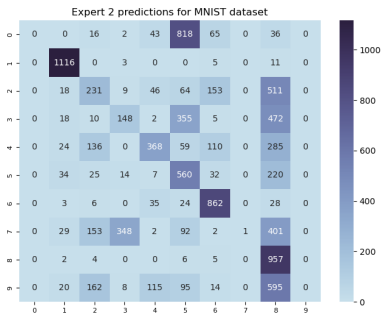
Figure 5.9a shows the gate expert selection table per class for MNIST test dataset with 5 experts using the *Stochastic Model* with temperature decay. We use the model with the best validation error. Figures 5.9b, 5.9c, 5.9d, 5.9e and 5.9f show the confusion matrix of predictions for the test samples for the corresponding experts. We see similar results as we saw for *output mixture model* in Figure 5.9.



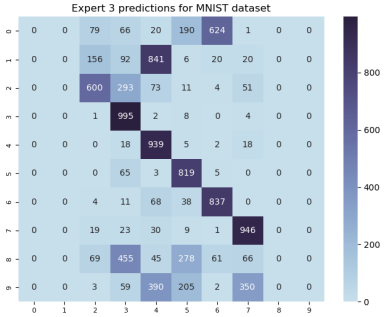
(a) Expert selection table per class with 5 experts



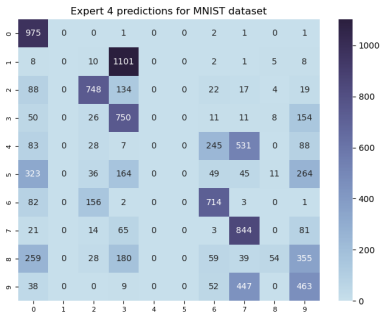
(b) Confusion matrix of expert 1 predictions on all data



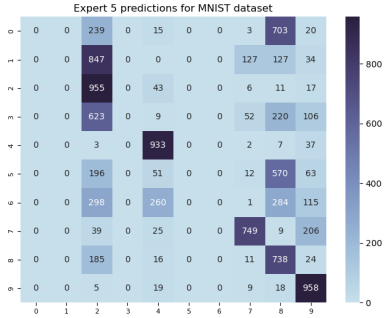
(c) Confusion matrix of expert 2 predictions on all data



(d) Confusion matrix of expert 3 predictions on all data



(e) Confusion matrix of expert 4 predictions on all data



(f) Confusion matrix of expert 5 predictions on all data

Figure 5.9: (a) Expert selection table showing the experts selected per class during inference with MNIST test data with *stochastic* model with dual temperature training with decay. (b),(c),(d),(e),(f) Confusion matrix of predictions of each of the 5 experts for the MNIST test data.

5.4 Discussion

The results show that dual temperature training improves expert utilization, performance and prevents module collapse compared to the original MoE models. Dual temperature training however, does not have a significant performance improvement over MoE with $L_{importance}$ regularization and top-2 models. A better temperature decay schedule could potentially improve performance. We also see that the experts are trained on samples of a subset of classes and specialize in them.

Chapter 6

No-Gate Expert Training

6.1 Motivation

We have thus far seen that end-to-end gate and expert training results in poor and unintuitive distribution of the input space among the experts without additional regularization. Decoupling the gate and expert training, in Chapter 5, clearly improved the expert utilization. It also resulted in experts getting a fair distribution of samples during initial rounds of training. However, it still left some experts unused and was outperformed by the efficient $L_{importancee}$ regularization. This is because we are using the gate to learn both a good task distribution to train the expert during training and to select the best expert for a sample during inference.

The gate is crucial for conditional computation during training and inference. However, during training, if the gate made a mistake and selected a poorly performing expert for the sample, that expert will be trained on that sample instead of the expert that actually performed better on the sample. What if we removed the gate altogether when training the experts? What if we just select the expert that performs best on a sample and train that expert during training? We can then use reverse distillation to add a gate and train it with the pre-trained experts. This ensures conditional computation during inference. Hence, we could completely separate the expert and gate training.

We introduce two algorithms for no-gate expert training called *loudest expert* in Section 6.2 and *peeking expert* in Section 6.3. In the *loudest expert* algorithm we select the expert with

the lowest prediction entropy for a sample. This is its estimate of its expected loss for that sample. If o_{iy} is the predicted probability for the correct class y by expert i, then $\log(o_{iy})$ is the loss of the expert. Hence, prediction entropy is the expected sum of the loss. We call it the *loudest expert* method because the expert with the lowest prediction entropy wins, that is, the one that is most confident or loudest, but not necessarily correct.

In the *peeking expert* algorithm we select the expert with the lowest *surprisal*. Surprisal of an expert is $-\log_2(o_{iy})$, where o_{iy} is the prediction probablity of the expert i for the correct target class y for a sample. Lower surprisal implies a high probability of the correct class. It is called the *peeking expert* method as we peek at the correct predictions for the sample during training. Since we do not have the correct class during inference we could train a gate with these pre-trained peeking experts. In effect we then use the gate for soft switching between the trained experts during inference for conditional computation and not for task decomposition among the experts during training.

Our experiments, in Section 6.4 show that the *peeking expert* method outperforms benchmark methods for classification problems on all metrics. For the experiments we ran it results in the cleanest task decomposition of all methods we have tried. Both the no-gate methods we instroduced are best suited for classification tasks and is what we have tested them for. But with some modifications we should be able to apply them for regression tasks.

6.2 Loudest expert algorithm

In this method we both train the experts and do inference without a gate. During training, we compute the prediction entropy, $H(\mathbf{o_i})$, from the expert prediction $\mathbf{o_i} = [o_{i1} \dots o_{iK}]$ for each expert i, per sample, as shown in Equation 6.1. K is the number of classes. We compute the distribution of the entropies over all the experts for the sample to get a weight vector w using *softmin*, as shown in Equation 6.2, where $softmin(\mathbf{x}) = softmax(-\mathbf{x})$. M is the number of experts. We use *softmin* because $H(\mathbf{o_i})$ is the expected loss and hence lower the $H(\mathbf{o_i})$ the better. We use w to weigh the gradient contribution of each expert when computing the loss, L, shown in Equation 6.3. Gradient is not computed with respect to the weight vector itself. y is the target class.

$$H(\mathbf{o_i}) = -\sum_{k=1}^{K} o_{ik} \cdot \log(o_{ik}) \tag{6.1}$$

$$\mathbf{w} = \text{softmin}(\,[H(\mathbf{o_1}), \ldots, H(\mathbf{o_M})]/T\,) \tag{6.2}$$

$$L = \sum_{i=1}^{M} w_i \cdot l(y, \mathbf{o_i}) \tag{6.3}$$

T in Equation 6.2 is the temperature factor that smooths the *softmin*, as we discussed in Section 5.1. A high T, especially at the beginning of the training, makes the weights more equal, allowing all experts to be trained on the samples. This prevents module collapse. A low T, during the later part of the training, ensures a high weight for only one expert. For each sample we select the expert that has the lowest prediction entropy for that sample. This results in only that expert getting trained, specializing it for the sample.

Inference is done by selecting the output of the expert with the lowest prediction entropy, $H(\mathbf{o_i})$, for the sample as in Equations 6.4 and 6.5, where $\hat{\mathbf{y}}$ is the MoE model output.

$$I = \underset{i \to M}{argmin}\ [H(\mathbf{o_1}), \ldots, H(\mathbf{o_M})] \tag{6.4}$$

$$\hat{\mathbf{y}} = \mathbf{o_I} \tag{6.5}$$

Experiments in Section 6.4.1 show that the test accuracy for *loudest expert* method is higher than the *output mixture* and *top-1* MoE methods but lower than that of *stochastic* and *top-K* models. Though there is a good expert usage with high H_u, the task distribution to the experts is not intuitive or clean as indicated by the low mutual information $I(E;Y)$. This motivated us to try the second no-gate expert training method described in the following Section 6.3. Algorithm 6.2.1 provides the pseudocode for training *loudest expert model*.

Algorithm 6.2.1: Training Loudest Expert Model

Input: $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{N}, \mathcal{X} \in \mathbb{R}^u$
epochs $\in \mathbb{N}$
$M \in \mathbb{N}$ /* number of experts */
$K \in \mathbb{N}$ /* number of classes */
$f_i : \mathcal{X} \to \mathbb{R}^K, i \in \{1, \ldots, M\}$ /* expert neural network */
$l : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ /* loss function */
expert optimizer O_e
$T \geq 1.0$
Output: $f_i(\cdot), i \in \{1, \ldots, M\}, \hat{y} \in \mathbb{R}^K$
for $epoch = \{1, \ldots, epochs\}$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    $o_i \leftarrow f_i(x), i \in \{1, \ldots, M\}$ /* expert outputs */
    $H(o_i) \leftarrow -\sum_{k=1}^{K} o_{ik} \cdot \log(o_{ik}), i \in \{1, \ldots, M\}$ /* expert prediction entropy */
    $w \leftarrow \text{softmin}(\,[H(o_1), \ldots, H(o_M)]/T\,)$ /* expert entropy distribution */
    $L \leftarrow \sum_{i=1}^{M} w_i \cdot l(y, o_i), y \in \mathcal{Y}$ /* MoE loss, also expected expert loss */
    $I \leftarrow \underset{i \to M}{argmin}\,[H(o_1), \ldots, H(o_M)]$ /* expert with min prediction entropy */
    $\hat{y} \leftarrow o_I$ /* MoE predicted output */
    compute expert gradients with L
    update O_e
  end
end

6.3 Peeking expert algorithm

The *loudest expert* is the most confident and specialized expert for the sample. But its prediction may be incorrect. So, instead of computing the prediction entropy, why not just compute the per sample surprisal using the correct prediction for the sample, since we have the correct predictions during training? Surprisal, S, is $-\log_2(o_{iy})$, where o_{iy} is the prediction probablity of an expert i, for a sample, for the correct target class y. We then select the expert with the lowest surprisal and train only that expert for that sample thereby specialising the expert for the sample.

During inference, however, we do not have the correct predictions. But we have the pre-trained experts that are already specialized for corresponding samples. We need a gate that can select the right expert for a sample during inference. So, we train a gate, with the same data and the pre-trained specialized experts. Hence this is a two-step training method. In the first step, we train the experts without the gate using per sample surprisal. In the second step, we train the

gate using the experts trained in the first step, which is then used as the inference model. The steps are as follows:

Step 1 **Training experts without a gate using surprisal:** We first train the experts without a gate. To train the experts we compute the per sample surprisal, S_i, corresponding to the target class y, for that sample, for expert i. M is the number of experts. The MoE output for a sample, $\hat{\mathbf{y}}$, is then the output of the expert with the lowest surprisal for the correct prediction of the sample as computed in Equations 6.6 to 6.8.

$$S_i = -\log_2(o_{iy}), i \in \{1, \ldots, M\} \tag{6.6}$$

$$I = \underset{i=1 \to M}{argmin} \, [S_1, \ldots, S_M] \tag{6.7}$$

$$\hat{\mathbf{y}} = \mathbf{o_I} \tag{6.8}$$

$$L = l(y, \hat{\mathbf{y}}) \tag{6.9}$$

The MoE is trained using loss L computed with the MoE predicted output $\hat{\mathbf{y}}$ as in Equation 6.9. Hence, gradient is computed only for the selected expert for the selected sample. This results in each expert specializing for some samples and hence some classes. Our experiments in Section 6.4.2 show that experts specialize to specific samples very quickly. For the CIFAR-10 dataset the experts specialize in less than 20 epochs. Hence, the first step of training is very quick.

Step 2 **Training a gate with pre-trained experts from Step 1:** Since during inference we do not have the target class we cannot use Equations 6.6, 6.7 and 6.8 for inference. Instead we train a gate with the experts trained in **Step 1**, that is, we train an MoE model with a gate and the pre-trained experts from **Step 1** using the same data on which the experts were trained in **Step 1**. The expert parameters are fixed for some epochs in the beginning so the gate learns to switch to the expert which specialises in the corresponding sample. After this initial training the expert parameters are no longer fixed so the gate and experts are trained end-to-end for the rest of the epochs. We used the *output mixture*, *stochastic* and *top-k* MoE architectures to train the gate to compare the performances. Our experiments in Section 6.4.2 show that the gate learns to select the correct expert

when trained with the output mixture, stochastic and top-k and performs better than all the benchmark models.

Algorithms 6.3.1 and 6.3.2 provide the pseudocode for the two-step training.

Algorithm 6.3.1: Step 1: Training peeking experts without a gate using surprisal

Input: $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{N}, \mathcal{X} \in \mathbb{R}^u, y_i \in \{y_{i1}, \ldots, y_{iK}\}$

epochs $\in \mathbb{N}$

$M \in \mathbb{N}$ /* number of experts */

$K \in \mathbb{N}$ /* number of classes */

$f_i : \mathcal{X} \to \mathbb{R}^K, i \in \{1, \ldots, M\}$ /* expert neural network */

$l : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ /* loss function */

expert optimizer O_e

Output: $f_i(\cdot), i \in \{1, \ldots, M\}, \hat{y} \subset \mathbb{R}^K$

for $epoch = \{1, \ldots, epochs\}$ **do**

  for $(x, y) \in \mathcal{D}$ **do**

    $o_i \leftarrow f_i(x), i \in \{1, \ldots, M\}$ /* expert outputs */

    $S_i \leftarrow -\log_2(o_{iy}), i \in \{1, \ldots, M\}$ /* correct prediction surprisal */

    $I \leftarrow \underset{i=1 \to M}{argmin}\ [S_1, \ldots, S_M]$ /* expert with min surprisal */

    $\hat{y} \leftarrow o_I$ /* MoE predicted output */

    $L \leftarrow l(y, \hat{y})$ /* MoE loss that is also the selected expert loss */

    compute expert gradients with L

    update O_e

  end

end

6.4 Experiments

We evaluate our methods on the MNIST (LeCun and Cortes, 2010) and CIFAR-10 (Krizhevsky, 2009) datasets. For both datasets we ran the experiments with 5 and 10 experts. For the *loudest expert* method we used only the experts. For the *peeking expert* method we first trained the experts and then trained one gate with the pre-trained experts. Details of expert and gate architectures for MNIST and CIFAR-10 are at Appendix A.1 and A.3.

All models were trained with Adam optimizer with 0.001 learning rate. We used 100 epochs for the MNIST dataset and 200 epochs for the CIFAR-10 dataset. Each experiment was run 10 times for both datasets.

Algorithm 6.3.2: Step 2: Training a gate with pre-trained peeking experts from Step 1 using output mixture model

Input: $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{N}, \mathcal{X} \in \mathbb{R}^u, y_i \in \{y_{i1}, \ldots, y_{iK}\}$
epochs $\in \mathbb{N}$
$epochs_{split} \in \mathbb{N}$ /* epochs at which to split the step 2 training */
$M \in \mathbb{N}$ /* number of experts */
$K \in \mathbb{N}$ /* number of classes */
$f_i : \mathcal{X} \to \mathbb{R}^K, i \in \{1, \ldots, M\}$ /* experts trained in Step 1 */
$g : \mathcal{X} \to \mathbb{R}^M$ /* gate neural network */
$l : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ /* loss function */
optimizer O
Output: $f_i(\cdot), g(\cdot), i \in \{1, \ldots, M\}, \hat{y} \in \mathbb{R}^K$
for $epoch = \{1, \ldots, epochs\}$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    $o_i \leftarrow f_i(x), i \in \{1, \ldots, M\}$ /* expert outputs */
    if $epoch < epochs_{split}$ **then**
      Freeze the weights of the experts /* Do not train experts */
    end
    else
      Unfreeze the weights of the experts /* Train experts */
    end
    $p \leftarrow g(x)$ /* gate output */
    $\hat{y} \leftarrow \sum_{i=1}^{M} p_i \cdot o_i$ /* MoE predicted output */
    $L \leftarrow l(y, \hat{y})$ /* MoE loss */
    if $epoch > epochs_{split}$ **then**
      compute expert gradients with L /* start training experts */
    end
    compute gate gradients with L
    update O
  end
end

6.4.1 Results for loudest expert method

We compared: (1) single model equivalent to a single expert; (2) *output mixture* MoE; (3) *stocastic* MoE; (4) *top-1* MoE; (5) *top-2* MoE; (6) *output mixture* MoE with $L_{importance}$ regularization; (7) *top-2* MoE with $L_{importance}$ regularization; and (8) *loudest expert* method.

The experiment results for MNIST dataset with 5 experts are in Table 6.1. The experiment results for CIFAR-10 dataset with 5 experts are in Table 6.2. The results with 10 experts for MNIST and CIFAR-10 datasets are in Tables B.1 and B.2 in Appendix .

The smoothing temperature for MNIST was varied over the 100 epochs with the schedule

$T = [10.0 * 25, 1.0 * 25, 0.01 * 25, 0.001 * 25]$, that is, the temperature was changed at 25 epoch intervals. For CIFAR-10 the smoothing temperature T was varied at intervals of 50 as $T = [10.0 * 50, 1.0 * 50, 0.01 * 50, 0.001 * 50]$ over 200 epochs. The results for each method of training, in the tables, are performance metrics computed on the model with the best validation error for the corresponding model type. The standard deviation of the test accuracy over the 10 runs of the selected model is also reported.

Table 6.1: Performance of the *loudest expert* model. Results are inference on MNIST test data with models with minimum validation error with 5 experts. Best result is highlighted.

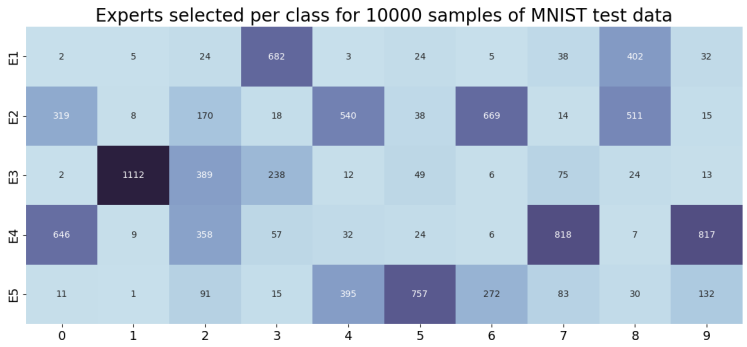
Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
output mixture MoE	95.83 ± 0.039	1.568	0.034	1.570
stochastic MoE	96.13 ± 0.021	1.569	0	1.569
top-1 MoE	94.29 ± 0.039	1.000	0	1.000
top-2 MoE	96.67 ± 0.017	1.552	0.029	1.570
output mixture MoE with $L_{importance}$	96.87 ± 0.008	2.280	0.053	2.322
top-2 MoE with $\mathbf{L_{importance}}$	**97.26 ± 0.004**	2.022	0.071	2.322
loudest expert method	94.98 ± 0.003	1.284	0	2.314

Table 6.2: Performance of the *loudest expert* model. Results are inference on CIFAR-10 test data with models with minimum validation error with 5 experts. Best result is highlighted.

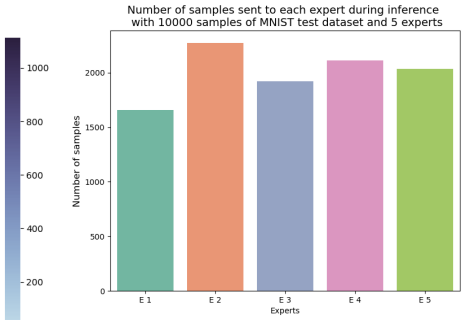
Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	42.74 ± 0.011	NA	NA	NA
output mixture MoE	70.44 ± 0.016	1.381	0.066	1.392
stochastic MoE	73.63 ± 0.021	1.889	0	1.960
top-1 MoE	67.46 ± 0.015	0.961	0	0.977
top-2 MoE	78.89 ± 0.025	1.845	0.259	1.957
output mixture MoE with $L_{importance}$	77.61 ± 0.024	2.315	0.195	2.314
top-2 MoE with $\mathbf{L_{importance}}$	**79.90 ± 0.029**	2.317	0.296	2.316
loudest expert method	72.82 ± 0.006	0.699	0	2.303

Tables 6.1 and 6.2 show that the test accuracy of *loudest expert* method is better than the single model and *top-1* method but not as well as the *output mixture*, *stochastic* and *top-2* methods. The *top-2* method with $L_{importance}$ regularization outperforms all the models. But we see that the *loudest expert* method has a high H_u which indicates good expert usage. This is also reiterated by the equitable test sample distribution in Figures 6.1b and 6.1d for MNIST dataset and Figures 6.2b and 6.2d for CIFAR-10 dataset.

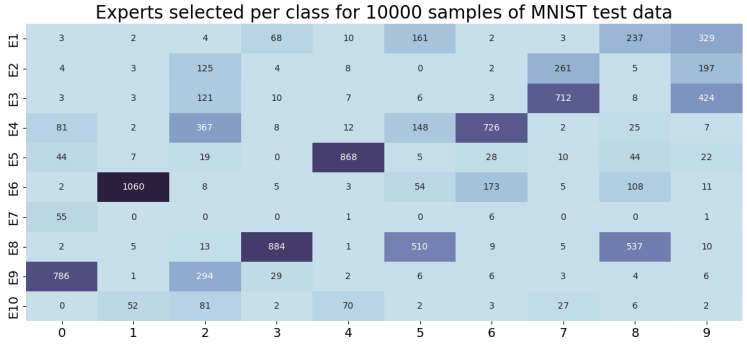
However, the expert selection tables for test samples in Figures 6.1a and 6.1c for MNIST dataset and Figures 6.2a and 6.2c for CIFAR-10 show that the distribution of the samples



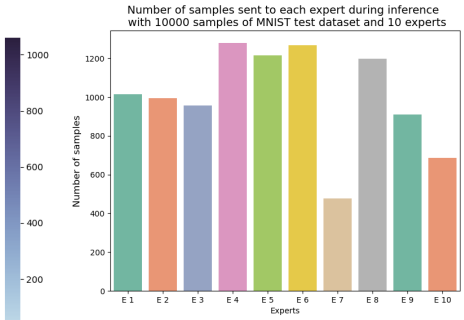
(a) Expert selection table per class with 5 experts



(b) Test sample distribution with 5 experts



(c) Expert selection table per class with 10 experts



(d) Test sample distribution with 10 experts

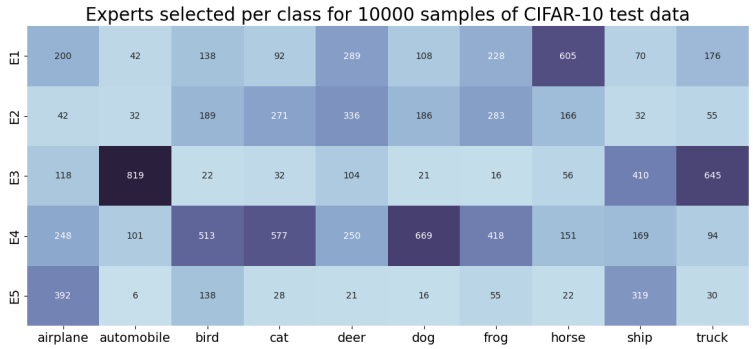
Figure 6.1: (a), (c) Expert selection tables showing the experts selected per class during inference with MNIST test data with *loudest expert* method. (b),(d) MNIST test sample distribution per expert during inference.

to the experts is not an intuitive and clean decomposition. This is also indicated by the low mutual information, $I(E;Y)$. Hence, though we improve the expert usage, the resulting sample distribution is not optimal.

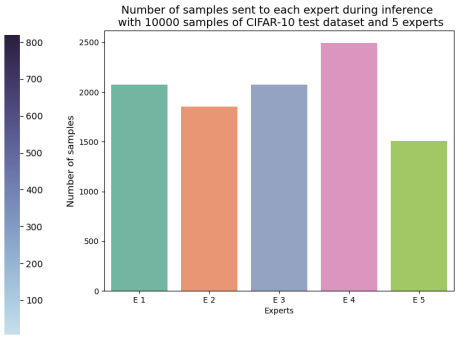
Figure 6.3 shows the sample distribution during training for MNIST and CIFAR-10 datasets using *loudest expert* method with 10 experts. We can clearly see the effect of the temperature schedule, T, on the sample distribution during training. During the initial epochs when T is high all experts get equal number of samples as they are all equally likely to be selected. As the temperature is decreased the experts start specialising for different samples. This explains the high expert usage of this method.

6.4.2 Results for peeking expert method

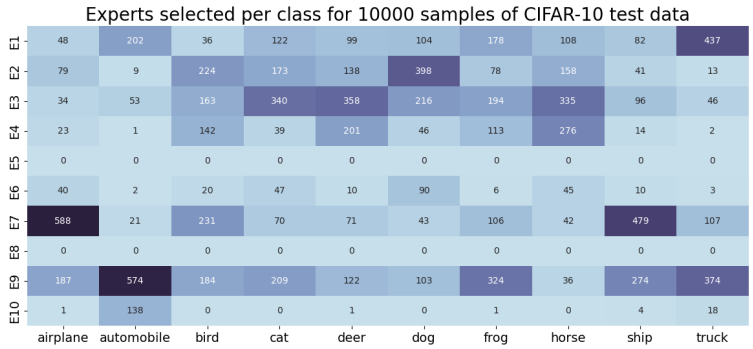
We trained the MoE model in two steps. In Step 1 we trained the experts using surprisal for 20 epochs. In Step 2 we reverse distilled a gated MoE with the pre-trained experts from Step 1.



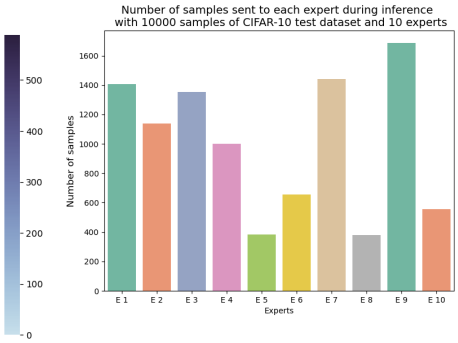
(a) Expert selection table per class with 5 experts



(b) Test sample distribution with 5 experts

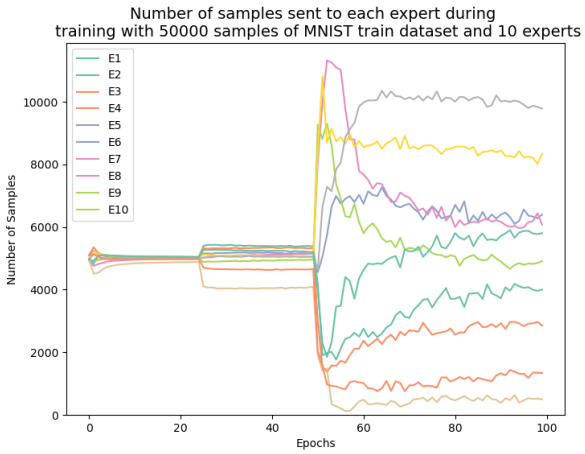


(c) Expert selection table per class with 10 experts

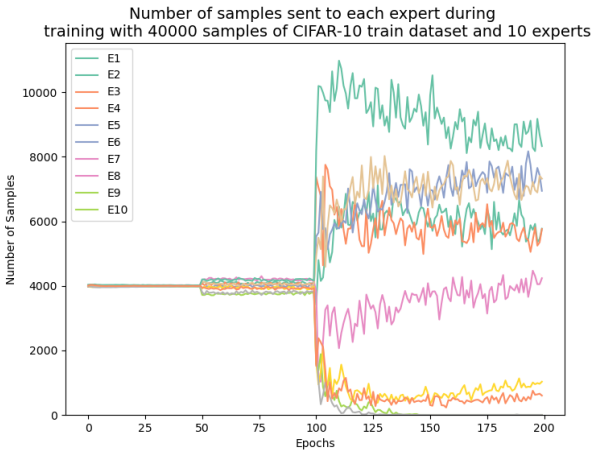


(d) Test sample distribution with 10 experts

Figure 6.2: (a), (c) Expert selection tables showing the experts selected per class during inference with CIFAR-10 test data with *loudest expert* method. (b),(d) CIFAR-10 test sample distribution per expert during inference



(a)



(b)

Figure 6.3: Sample distribution during training with *loudest expert* model with 10 experts: (a) for MNIST dataset with temperature schedule $T = [10.0 * 25, 1.0 * 25, 0.01 * 25, 0.001 * 25]$ over 100 epochs and (b) CIFAR-10 dataset with with temperature schedule $T = [10.0 * 50, 1.0 * 50, 0.01 * 50, 0.001 * 50]$ over 200 epochs.

The gate was trained for 100 epochs for MNIST dataset and 200 epochs for CIFAR-10 dataset.

We report the performance results of the resulting reverse distilled gated MoE models.

We compared: (1) single model equivalent to a single expert; (2) *output mixture* MoE; (3) *stochastic* MoE; (4) *top-1* MoE; (5) *top-2* MoE; (6) *output mixture* MoE with $L_{importance}$ regularization; (7) *top-2* MoE with $L_{importance}$ regularization; (8) *reverse distilled gated MoE, using pre-trained peeking experts, trained with output mixture*; (9) *reverse distilled gated MoE, using pre-trained peeking experts, trained with stochastic MoE*; (10) *reverse distilled gated MoE, using pre-trained peeking experts, trained with top-1 model*; (11) *reverse distilled gated MoE, using pre-trained peeking experts, trained with top-2 model.*

The experiment results for MNIST dataset with 5 experts are in Table 6.3. MNIST results with 10 experts are in Table B.3 in Appendix B.1.3. The results for each method of training, in the table, are performance metrics computed using the model with the best validation error for the corresponding model type. The standard deviation of the test accuracy over the 10 runs of the selected model is also reported.

Table 6.3: Performance of the *peeking expert* model. Results are inference on MNIST test data with models with minimum validation error with 5 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
output mixture MoE	95.38 ± 0.021	1.488	0.031	1.490
stochastic MoE	96.20 ± 0.028	1.405	0	1.516
top-1 MoE	94.83 ± 0.335	1.357	0	1.357
top-2 MoE	96.25 ± 0.007	1.663	0.074	1.912
output mixture MoE with $L_{importance}$	96.87 ± 0.008	2.280	0.053	2.322
top-2 MoE with $L_{importance}$	97.26 ± 0.004	2.022	0.071	2.322
reverse distilled gate, with peeking experts, trained with output mixture MoE	97.09 ± 0.002	2.237	0.013	2.237
reverse distilled gate, with peeking experts, trained with stochastic MoE	**97.34 ± 0.002**	2.175	0	2.176
reverse distilled gate, with peeking experts, trained with top-1 MoE	96.58 ± 0.011	2.231	0	2.231
reverse distilled gate, with peeking experts, trained with top-2 MoE	97.19 ± 0.015	2.235	0.015	2.239

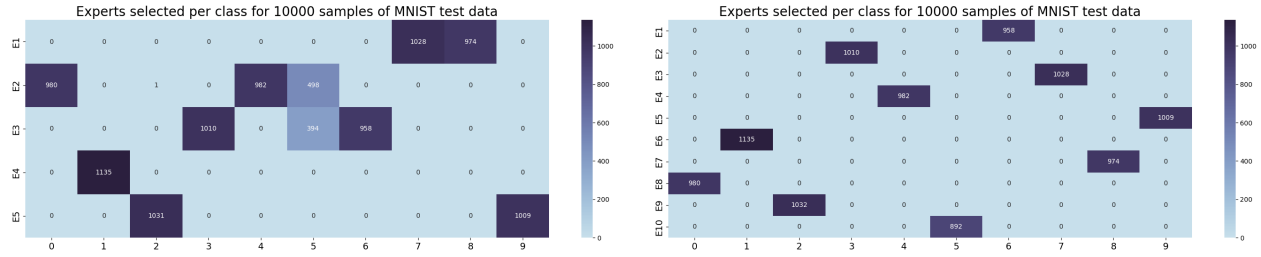
The experiment results for CIFAR-10 dataset with 5 experts are in Table 6.4. CIFAR-10 results with 10 experts are in Table B.4 in Appendix B.1.4.

Tables 6.3 and 6.4 show that the *peeking expert* method for both datasets outperforms all the benchmark methods, including $L_{importance}$ regularization, in terms of test accuracy, expert usage H_u and gate sparsity H_s. The performance improvement for the MNIST dataset is not significantly high. But the performance for the CIFAR-10 dataset was improved significantly by

Table 6.4: Performance of the *peeking expert* model. Results are inference on CIFAR-10 test data with models with minimum validation error with 5 experts. Best result is highlighted.

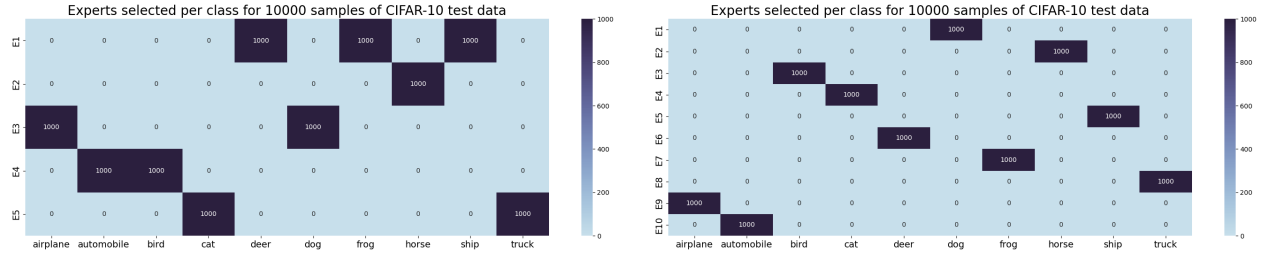
Experiment	**Test Accuracy**	**I(E; Y)**	**H_s**	**H_u**
single model	42.74 ± 0.011	NA	NA	NA
original MoE	70.44 ± 0.016	1.381	0.066	1.392
stochastic MoE	73.63 ± 0.021	1.889	0	1.960
top-1 MoE	67.46 ± 0.015	0.961	0.044	0.977
top-2 MoE	78.89 ± 0.025	1.845	0.259	1.957
output mixture MoE with $L_{importance}$	77.61 ± 0.024	2.315	0.195	2.314
top-2 MoE with $L_{importance}$	79.90 ± 0.029	2.317	0.296	2.316
reverse distilled gate, with peeking experts, trained with output mixture MoE	83.60 ± 0.012	2.249	0.028	2.249
reverse distilled gate, with peeking experts, trained with stochastic MoE	**84.80 ± 0.013**	2.243	0	2.243
reverse distilled gate, with peeking experts, trained with top-1 MoE	66.21 ± 0.039	0	0	0
reverse distilled gate, with peeking experts, trained with top-2 MoE	85.33 ± 0.010	2.245	0.118	2.245

≈ 5% from the best performing benchmark method of *top-2* MoE with $L_{importance}$ regularization.



(a) 5 experts (b) 10 experts

Figure 6.4: Expert selection table per class for MNIST test data using *peeking experts* after Step 1 training with 5 and 10 experts.

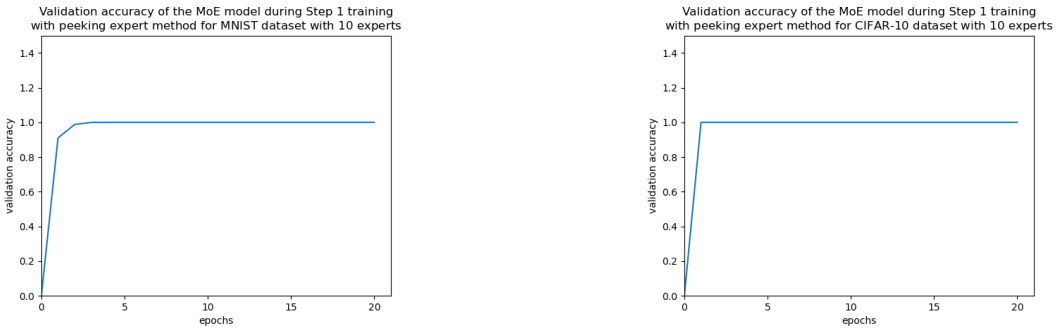


(a) 5 experts (b) 10 experts

Figure 6.5: Expert selection table per class for CIFAR-10 test data using *peeking experts* after Step 1 training with 5 and 10 experts.

Let us now see what each expert learns after each step of training. Figures 6.4 and 6.5 show the experts selected per class, using *peeking experts* after Step 1 training, for MNIST and CIFAR-10 test samples respectively. We see that there is a clean task specific distribution of samples across the experts.

Figure 6.6 shows the validation performance during Step 1 of *peeking expert* training. It shows that the *peeking experts* specialize in < 20 epochs for both MNIST and CIFAR-10 datasets. Hence, Step 1 training is very quick.

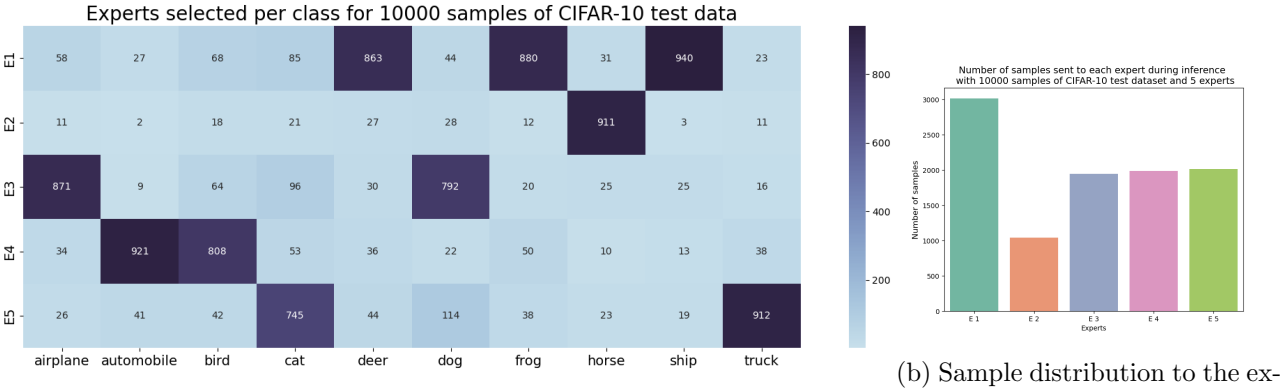


(a) MNIST (b) CIFAR-10

Figure 6.6: Validation accuracy over 20 epochs of Step 1 training for (a) MNIST and (2) CIFAR-10 datasets during Step 1 no-gate training of *peeking experts* for 10 experts.

We now look at what the gate learns after Step 2, the reverse distillation of the gate with the pre-trained *peeking experts* from Step 1, with different methods of MoE training. We use the CIFAR-10 dataset for our analysis. Appendix B.1 has similar results for MNIST dataset.

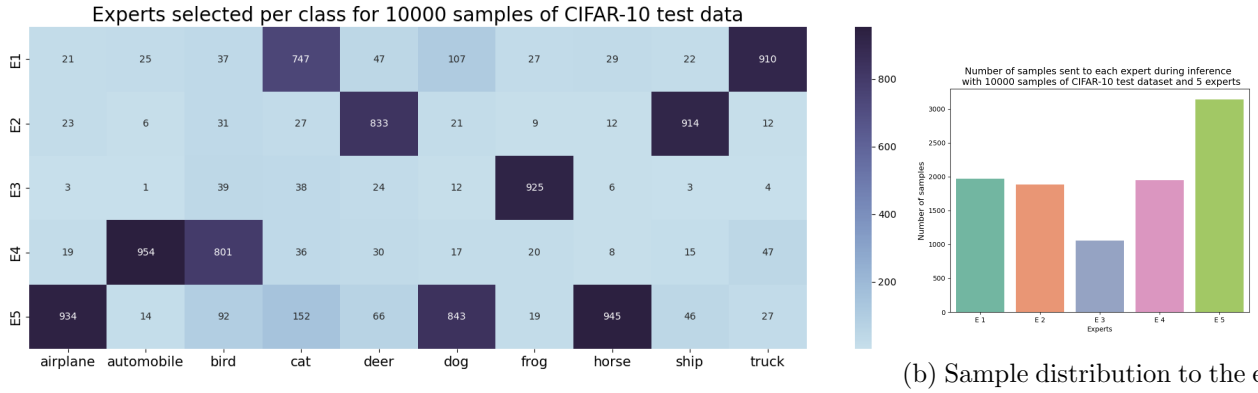
Figures 6.8, 6.12, 6.9 and 6.10 show the experts selected by the gate, in the reverse distilled gated MoE models trained with the pre-trained *peeking experts* using *output mixture* model, *stochastic*, *top-1* and *top-2* models respectively with 5 experts. We can see that the gate does learn to select the correct experts for the samples with *output mixture*, *stochastic* and the *top-2* methods. We see module collapse with *top-1* method. The model trained with *output mixture*, *stochastic* and *top-2* methods outperforms *top-1* method.



(a) Gate expert selection table (b) Sample distribution to the experts by the gate

Figure 6.7: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5a for CIFAR-10 dataset. Gate is trained using *output mixture* model.

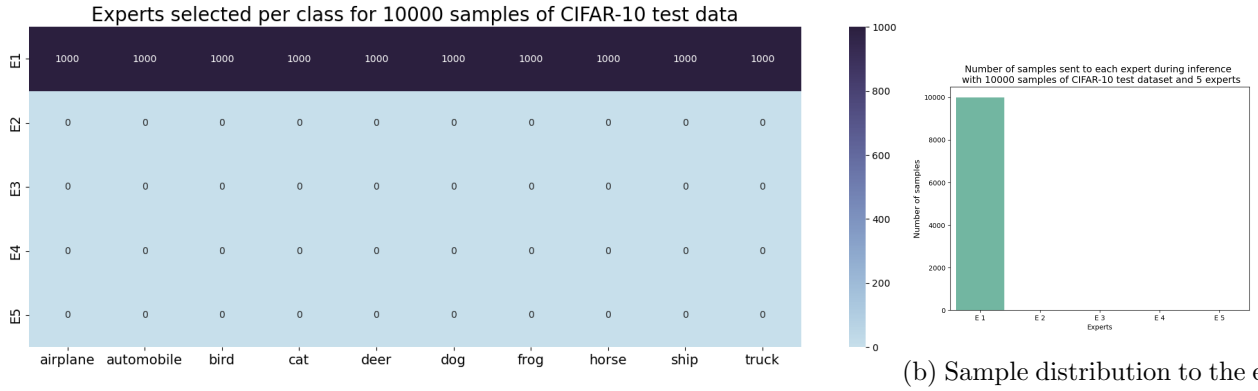
Figures 6.11, 6.12, 6.13 and 6.14 show that with 10 experts all 4 MoE training methods, used to



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

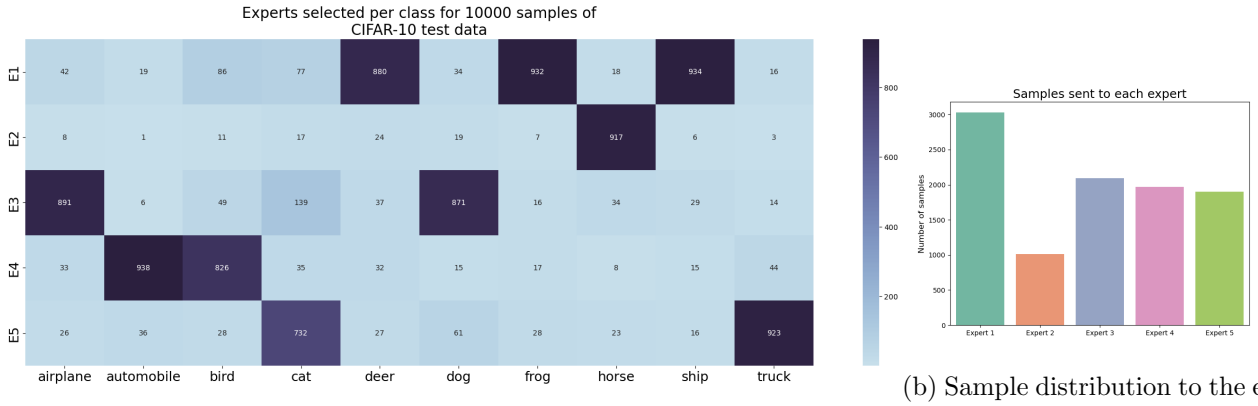
Figure 6.8: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5a for CIFAR-10 dataset. Gate is trained using *stochastic* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 6.9: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5a for CIFAR-10 dataset. Gate is trained using output *top-1* model.

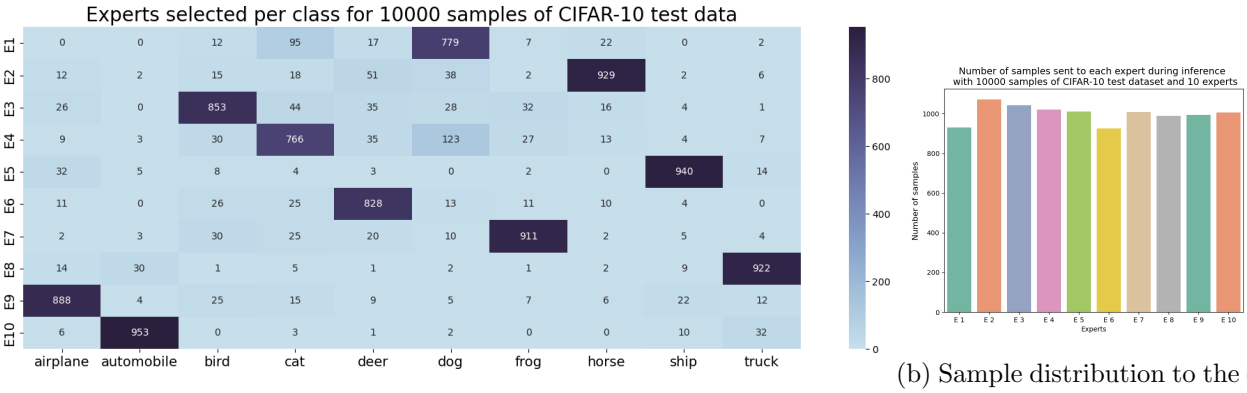


(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 6.10: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5a for CIFAR-10 dataset. Gate is trained using output *top-2* model.

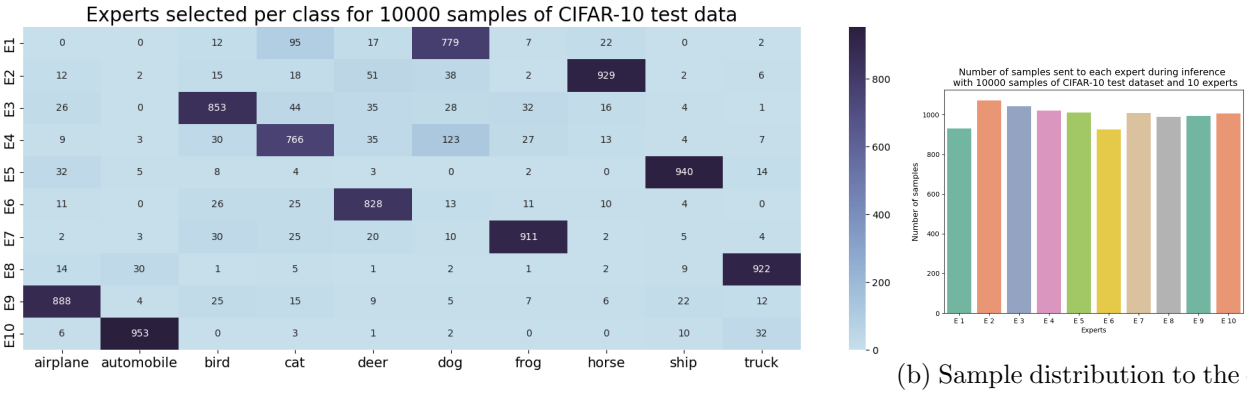
reverse distill the gated MoE, perform well. But *output mixture* and *stochastic* methods have the best test accuracy, use the experts equitably, and result in a cleaner decomposition.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

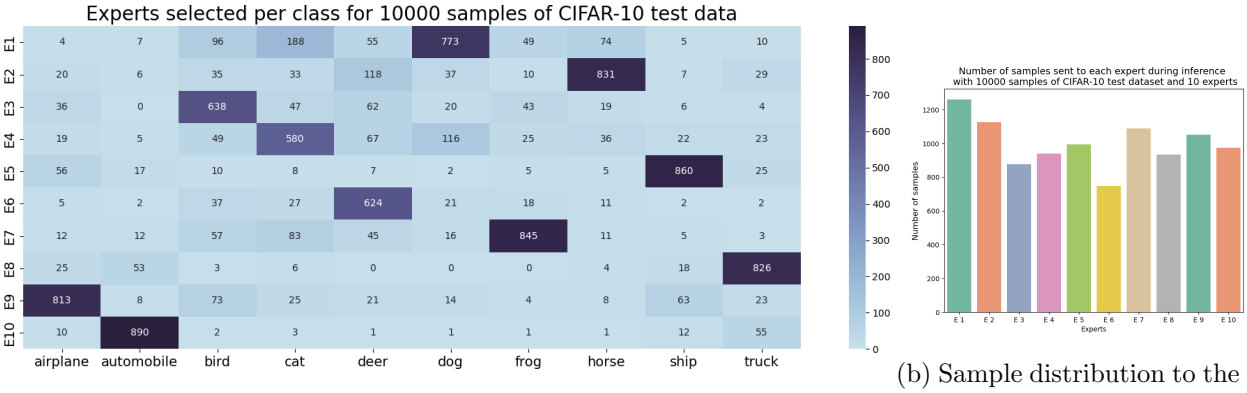
Figure 6.11: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5b for CIFAR-10 dataset. Gate is trained using *output mixture* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

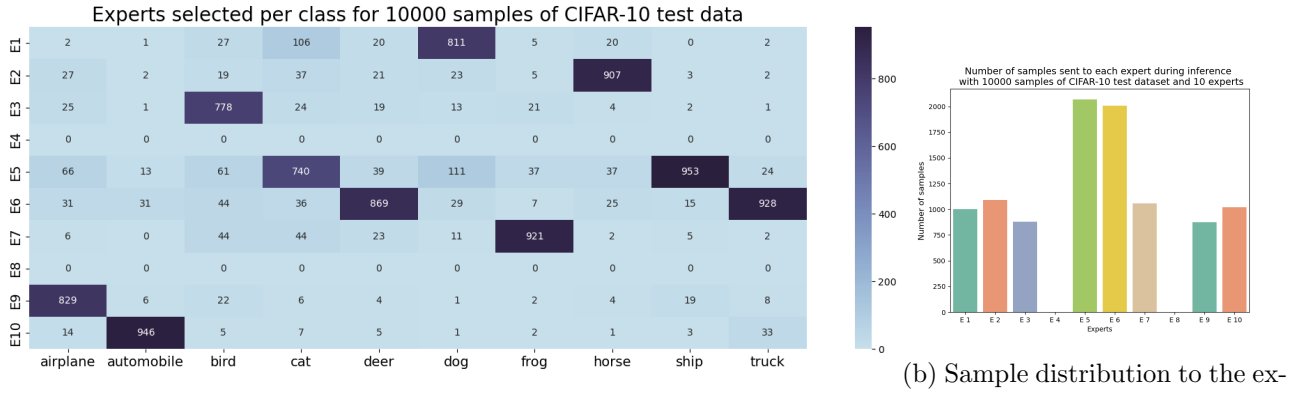
Figure 6.12: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5b for CIFAR-10 dataset. Gate is trained using *stochastic* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 6.13: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5b for CIFAR-10 dataset. Gate is trained using *top-1*.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 6.14: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.5b for CIFAR-10 dataset. Gate is trained using *top-2*.

6.5 Discussion

We introduced two no-gate expert training algorithms: (1) *loudest expert* algorithm; and (2) *peeking expert* algorithm. Both algorithms avoid module collapse and result in equitable sample distribution to the experts. However, the *loudest expert* method does not result in a clean task decomposition, while the *peeking expert* method does result in a clean task decomposition. Scaling the temperature for *loudest expert*, that is, finding the correct schedule is difficult. It is possible that with a good schedule we can improve its performance.

The *peeking expert* method outperforms the *loudest expert* method and all the benchmark MoE methods with regularization. It is a more intuitive method of training an MoE model that completely separates the training of the experts and the gate by using the domain knowledge of the target labels for training the experts. We see that the reverse distillation of the gate with the pre-trained *peeking experts* results in the gate learning to use the correct experts for samples that are specialised for those samples. The current limitation for both these methods are that they are best suited for classification problems. We believe that it can be adapted for regression and unsupervised datasets by designing the appropriate loss. For example, for regression problem we can simply select the expert that has the least mean squared error for the sample.

Chapter 7

Attentive Gating MoE Architecture

7.1 Motivation

In the current MoE architectures, gate computations are entirely separate from the expert and hence may be duplicating the effort. Both the gate and the expert are trained together end-to-end. The expert distribution predicted by the gate for a batch does not depend on the expert predictions on that batch. This is to allow conditional computation during training. Both the expert and gate learn sample classification and expert distribution, respectively, based on the same input distribution. It then seems reasonable not to duplicate this learning.

We designed a novel intuitively plausible MoE architecture, shown in Figure 7.1, that uses the expert's computations to compute the gating distribution. The proposed algorithm computes the gate's expert distribution using a scoring method similar to that proposed by Bahdanau et al. (2015), from the gate and expert hidden layer outputs. Hidden layers are the pre-output layers. This is effectively asking the question, **which experts should the gate attend to for a given sample?**

In the attentive gating architecture the experts have the same architecture as in other MoE architectures. But we also have access to the expert's logit layer before applying the *softmax* layer. The gate also has the same architecture except that the output is just the logit layer, that is the *pre-softmax* layer. During training we add the additional weights W_k and W_q shown in Figure 7.1. Hence, there are additional parameters in the attentive gating architecture. The

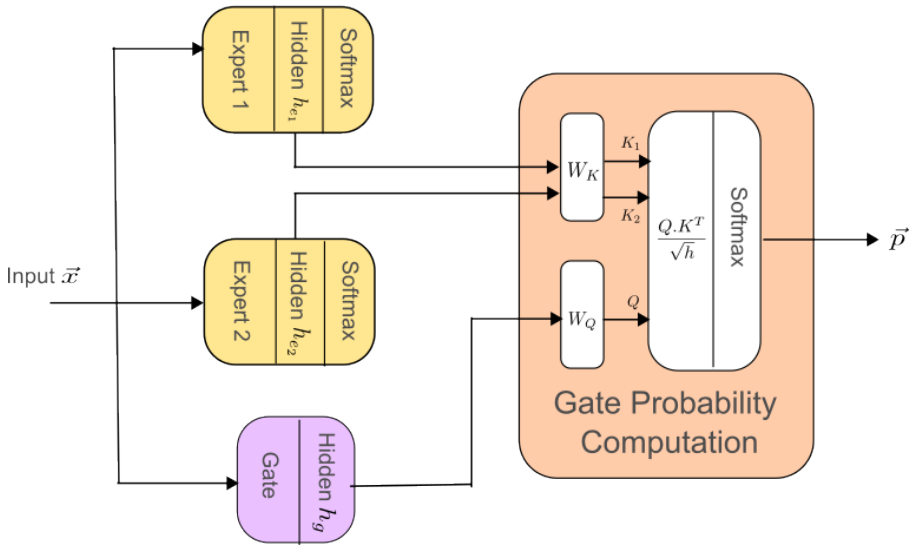


Figure 7.1: Attentive gating MoE architecture with 2 experts and a gate. The hidden layer outputs of the experts, $\mathbf{h_{e1}}$ and $\mathbf{h_{e2}}$, are used to compute the keys, K. The hidden layer output of the gate, $\mathbf{h_g}$, is used to compute the query Q. The gate probability, $\mathbf{p}$, is computed from Q and K as $softmax(Q_{1\times h} \cdot K^T_{M\times h}/\sqrt{h})$.

gate's expert distribution for a given sample is computed as the attention score from the gate and experts hidden layer outputs for the given batch. Once the gate expert distribution is computed we can use any existing expert aggregation method to compute the MoE output.

Since the computation of the gate distribution depends on the experts hidden layer outputs on the batch, the output of all experts needs to be computed during training. Gradient is still only computed for those experts that are selected by the gate. Since this is expensive during inference, when we desire conditional computation, we can learn a new gated MoE model, through distillation, from the trained attentive gate model. The resulting distilled gated MoE model is then used for inference and allows conditional computation. During distillation we use the same output aggregation method used while training the attentive gate model. Hence, though attentive gate model has addtional weights added during training, they can be removed for inference. It is also convenient that we can use all the existing MoE methods to train the attentive gate model.

Our experiments, in Section 7.3, show that the attentive gate model performs better than the benchmark models without regularizations. It also results in more equitable expert usage and cleaner task decomposition.

7.2 Training the attentive gate MoE architecture

Let us now dive into the details of the attentive gate architecture. During training, the gate's output is the current query or token of interest and the expert hidden layer outputs are the sequence of tokens that are attended to. The gate's hidden output, $\mathbf{h_{g_{1\times h}}}$ (subscripts are the dimensions of the vectors and matrices just for clarity), is used to compute the *Query*, $Q_{1\times h}$, as in Equation 7.1 and the expert hidden outputs, $\mathbf{h_{ei_{1\times h}}}$, are used to compute the *Keys*, $K_{i_{1\times h}}$, as in Equation 7.2, for expert i where $i = [1, \ldots, M]$. M is the number of experts. h is the size of the hidden layers of the experts and the gate. $W_{q_{h\times h}}$ and $W_{k_{h\times h}}$ are the query and key weight matrices.

The gate probability distribution, p, is then computed as in Equation 7.3:

$$Q_{1\times h} = \mathbf{h_{g_{1\times h}}} \cdot W_{q_{h\times h}} \tag{7.1}$$

$$K_{i_{1\times h}} = \mathbf{h_{ei_{1\times h}}} \cdot W_{k_{h\times h}}, i \in 1, \ldots, M \tag{7.2}$$

$$\mathbf{p} = softmax\left(\frac{Q_{1\times h} \cdot K^T_{M\times h}}{\sqrt{h}}\right) \tag{7.3}$$

$$\hat{\mathbf{y}} = \sum_{i=1}^{M} p_i \cdot \mathbf{o_i} \tag{7.4}$$

$$L = l(y, \hat{\mathbf{y}}) \tag{7.5}$$

The gate probabilities $\mathbf{p} = [p_1, \ldots, p_M]$ are used to select the corresponding expert and to compute the MoE output and loss while training. We can use any of the existing methods to compute the MoE output such as *output mixture* model, *stochastic* model or *top-k* model. If for example, we choose the *output mixture* model then the MoE output $\hat{\mathbf{y}}$ and loss L, during training, will be computed as in Equations 7.4 and 7.5, where $\mathbf{o_i}$ is the prediction output of expert i. Algorithm 7.2.1 provides pseudocode for training the attentive gate model.

Algorithm 7.2.1: Attentive Gate Training with Output Mixture Model

Input: $\mathcal{D} = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{N}, \mathcal{X} \in \mathbb{R}^u$
epochs $\in \mathbb{N}$
$M \in \mathbb{N}$ /* number of experts */
$K \in \mathbb{N}$ /* number of classes */
$h \in \mathbb{N}$ /* hidden layer size */
$f_i : \mathcal{X} \to \mathbb{R}^h, i \in \{1, \ldots, M\}$ /* expert neural network */
$d_i : f_i(x) \to \mathbb{R}^K, x \in \mathcal{X}, i \in \{1, \ldots, M\}$ /* dense layer of expert */
$g : \mathcal{X} \to \mathbb{R}^h$ /* gate neural network */
$W_k \subset \mathbb{R}^{h \times h}$ /* key weight matrix */
$W_q \subset \mathbb{R}^{h \times h}$ /* query weight matrix */
$l : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ /* loss function */
optimizer O /* gate, expert, W_k and W_q optimizer */
Output: $g(\cdot), f_i(\cdot), d_i(\cdot), i \in \{1, \ldots, M\}, W_k, W_q, \hat{y} \subset \mathbb{R}^K$
for $epoch = \{1, \ldots, epochs\}$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    $h_{ei_{1\times h}} \leftarrow f_i(x), i \in \{1, \ldots, M\}$ /* expert hidden outputs */
    $o_i \leftarrow softmax\left(d_i\left(h_{ei_{1\times h}}\right)\right), i \in \{1, \ldots, M\}$ /* expert outputs */
    $h_{g_{1\times h}} \leftarrow g(x)$ /* gate hidden layer output */
    $K_{i_{1\times h}} \leftarrow h_{ei_{1\times h}} \cdot W_{k_{h\times h}}, i \in \{1, \ldots, M\}$ /* key */
    $Q_{1\times h} \leftarrow h_{g_{1\times h}} \cdot W_{q_{h\times h}}$ /* query */
    $p \leftarrow softmax\left(\frac{Q_{1\times h} \cdot K^T_{M\times h}}{\sqrt{h}}\right)$ /* gate probability distribution */
    $\hat{y} \leftarrow \sum_{i=1}^{M} p_i \cdot o_i$ /* MoE predicted output */
    $L \leftarrow l(y, \hat{y})$ /* MoE loss */
    compute expert, gate, W_k and W_q gradients with L
    update O
  end
end

7.2.1 Distilling attentive gating MoE model for conditional computation

In the attentive gate architecture expert computations are used by the gate during feed forward. This does not allow for conditional computation during inference. To address this we distill the attentive gate model, trained as above, into one of the MoE models such as *output mixture model*, *stochastic* model or *top-k* models. We remove the weights W_k and W_q and add additional *output* and *softmax* layers to the gate to predict the expert distribution. We fix the parameters of the experts learnt using the attentive gate, initialise the gate of the new MoE model with the trained gate parameters of the attentive gate model and proceed to train the new MoE model

and gate. Our experiments, in Section 7.3, show that the distilled models perform as well as the attentive gate models they are distilled from, for all the training methods.

7.3 Experiments

We evaluate our methods on the MNIST (LeCun and Cortes, 2010) and CIFAR-10 (Krizhevsky, 2009) datasets. For both datasets we ran the experiments with 5 and 10 experts. Details of expert and gate architectures for MNIST and CIFAR-10 are at Appendix A.1 and A.3.

All models were trained with Adam optimizer with 0.001 learning rate. We used 100 epochs for MNIST dataset and 200 epochs for CIFAR-10 dataset. Each experiment was run 10 times for both datasets.

We compared: (1) single model which has the same architecture as one expert; (2) *output mixture* MoE; (3) *stochastic* MoE; (4) *top-1* MoE; (5) *top-2* MoE; (6) *output mixture* MoE with attentive gating; (7) *stochastic* MoE with attentive gating MoE; (8) *top-1* MoE with attentive gating MoE; (9) *top-2* MoE with attentive gating MoE; (10) distilled MoE with *output mixture* MoE; (11) distilled MoE with *stochastic* MoE; (12) distilled MoE with *top-1* MoE; and (13) distilled MoE with *top-2* MoE.

Table 7.1: Performance of the model with minimum validation error, for attentive gate and distilled attentive gate models, on the test set for MNIST dataset with 5 experts. Best result for benchmark models, attentive gate models and distilled attentive gate models are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
original MoE	95.83 ± 0.039	1.568	0.034	1.570
stochastic MoE	96.13 ± 0.021	1.569	0	1.569
top-1 MoE	94.29 ± 0.039	1.000	0	1.000
top-2 MoE	**96.67 ± 0.017**	1.552	0.029	1.570
attentive output mixture gate MoE	96.51 ± 0.010	2.119	0.006	2.173
attentive stochastic gate MoE	**96.80 ± 0.008**	1.970	0	2.119
top-1 with attentive gate MoE	95.92 ± 0.038	1.966	0	1.969
top-2 with attentive gate MoE	96.43 ± 0.005	2.000	0.026	2.173
distilled MoE with output mixture MoE	96.42 ± 0.012	2.165	0.008	2.182
distilled MoE with stochastic MoE	96.55 ± 0.009	2.164	0	2.190
distilled MoE with top-1 MoE	96.05 ± 0.153	2.130	0	2.130
distilled MoE with top-2 MoE	**96.74 ± 0.007**	1.770	0.017	1.870

Table 7.2: Performance of the model with minimum validation error, for attentive gate and distilled attentive gate models, on the test set for CIFAR-10 dataset with 5 experts. Best result for benchmark models, attentive gate models and distilled attentive gate models are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	42.74 ± 0.011	NA	NA	NA
output mixture MoE	70.44 ± 0.015	1.381	0.066	1.392
stochastic MoE	73.63 ± 0.021	1.889	0	1.960
top-1 MoE	67.46 ± 0.015	0.961	0.044	0.977
top-2 MoE	**78.89 ± 0.025**	1.845	0.26	1.957
attentive output mixture gate MoE	65.29 ± 0.061	0	0	0
attentive stochastic gate MoE	81.26 ± 0.032	2.141	0	2.270
top-1 with attentive gate MoE	66.46 ± 0.038	0	0	0
top-2 with attentive gate MoE	**83.35 ± 0.005**	2.222	0.108	2.321
distilled MoE with output mixture MoE	65.43 ± 0.235	0	0	0
distilled MoE with stochastic MoE	81.38 ± 0.035	2.153	0	2.277
distilled MoE with top-1 MoE	64.42 ± 0.277	0	0	0
distilled MoE with top-2 MoE	**82.87 ± 0.004**	2.269	0.081	2.321

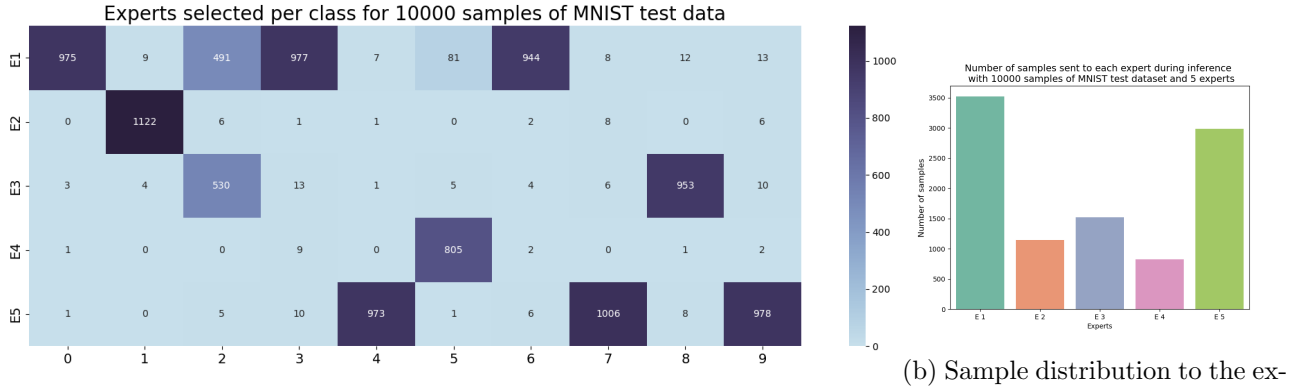
The experiment results for MNIST dataset are in Table 7.1. The experiment results for CIFAR-10 dataset are in Table 7.2. The results for each method of training, in the tables, are the performance metrics computed on the test set, with the the model that has the minimum validation error for the corresponding model type. The standard deviation of the test accuracy over the 10 runs of the selected model is also reported.

Tables 7.1 and 7.2 show that the attentive gating model performs better than the benchmark MoE models for both MNIST and CIFAR-10 datasets. They also show that the distilled models perform as well as the attentive gate models they are distilled from with better expert usage. From Tables 7.1 and B.5 (Appendix B.2.3) we see that for MNIST dataset the MoE model with 10 experts performs better than with 5 experts. From Tables 7.2 and B.6 (Appendix B.2.8) we see that for CIFAR-10 dataset the MoE model with 10 experts performs better than with 5 experts for some methods of training.

The performance improvement for the MNIST dataset is not significantly high. But the performance for the CIFAR-10 dataset was improved significantly by 4% from the best performing benchmark method of *top-2* MoE.

Let us now look at the task decomposition over the experts by the gate after training using the attentive gate model. Figure 7.2 shows the experts used during inference, on the test set, using attentive gate model trained with *stochastic* method for MNIST dataset with 5 experts. It is

the best performing model for MNIST dataset in Table 7.1. Results with 5 and 10 experts with different methods of training the attentive gating model are in Appendix B.2.1 and B.2.3. Note that this is the model with the attention weights. We see that attentive gate model results in clean task decompositions with good expert usage, without regularizations, for MNIST dataset for all methods of training with both 5 and 10 experts.

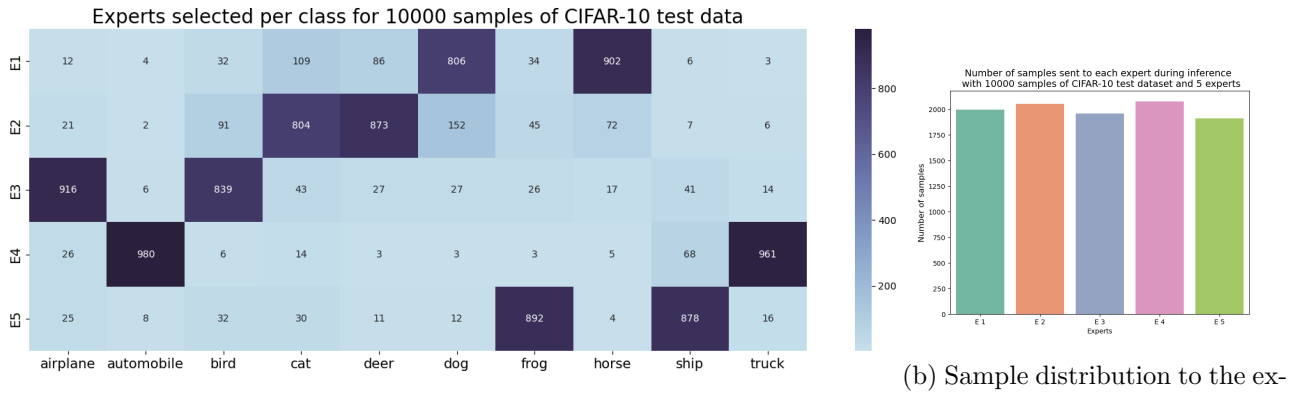


(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 7.2: Gate expert selection table of the attentive gate model trained with 5 experts using *stochastic* MoE on MNIST test dataset. It is the best performing attentive gate model for MNIST dataset.

Figure 7.3 shows the experts used during inference, on the test set, using attentive gate model trained with *top-2* method for CIFAR-10 dataset with 5 experts. It is the best performing attentive gate model for CIFAR-10 in Table 7.2. Results with 5 and 10 experts for all methods of training the attentive gate model are in Appendix B.2.5 and B.2.8. We see that there is equitable and clean task decomposition when trained with *stochastic* method but not with *output mixure* and *top-1* methods.



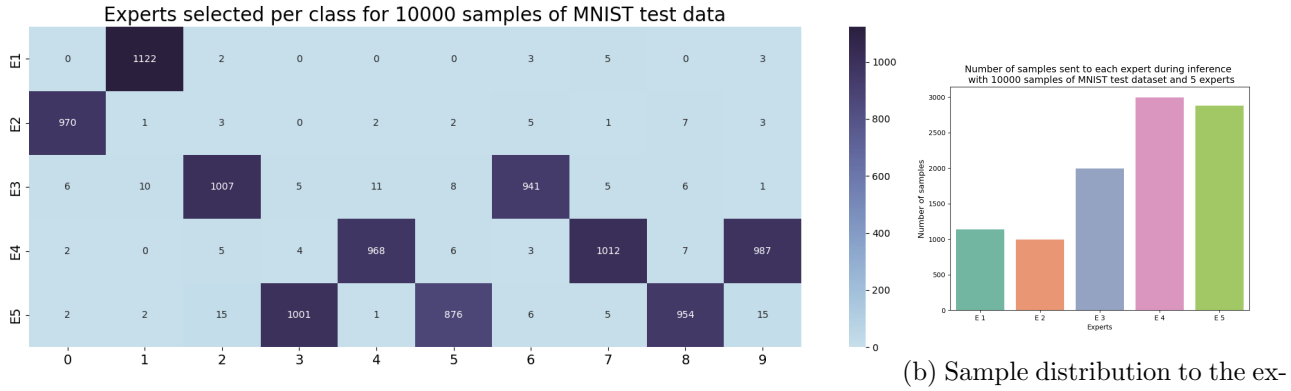
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 7.3: Gate expert selection table of the attentive gate model trained with 5 experts using *top-2* MoE on CIFAR-10 test dataset. It is the best performing attentive gate model for CIFAR-10 dataset.

Figure 7.4 shows the experts used during inference, on the test set, using the distilled gate model trained from the attentive gate model with *stochastic* method for MNIST dataset with

5 experts. It is the model distilled from the best performing attentive gate model for MNIST dataset in Table 7.1. Results with 5 and 10 experts with different methods of training the distilled model are in Appendix B.2.2 and B.2.4. We see that the distilled model also results in equitable and clean task decompositions without regularizations for MNIST dataset for all methods of training.

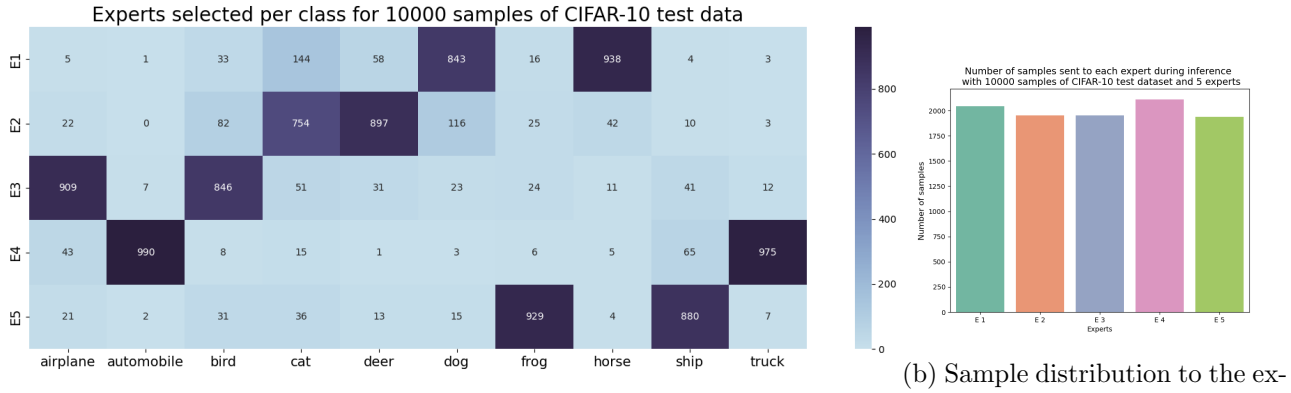


(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 7.4: Gate expert selection table of the distilled model trained with 5 experts using *stochastic* MoE on MNIST test dataset. It is the model distilled from the best performing attentive gate model for MNIST dataset.

Figure 7.5 shows the experts used during inference, on the test set, using the distilled gate model trained from the attentive gate model with *top-2* method for CIFAR-10 dataset with 5 experts. It is the model distilled from the best performing attentive gate for CIFAR-10 in Table 7.2. Results with 5 and 10 experts for all methods of training the distilled model are in Appendix B.2.5 and B.2.8. We see that the distilled model also results in equitable and clean task decompositions without regularizations for CIFAR-10 dataset for all methods except of training except for the *top-1* method.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 7.5: Gate expert selection table of the distilled model trained with 5 experts using *top-2* MoE on CIFAR-10 test dataset. It is the model distilled from the best performing attentive gate model for CIFAR-10 dataset.

7.4 Discussion

The above experiments show that training with attention results in both cleaner task decompositions to the experts and a more equitable expert usage without additional regularizations. We also see that the performance with the attentive gate model is better than the benchmark MoE methods. Attentive gating mechanism does not replace the benchmark methods and regularizations. It only changes how the gate probablities are computed. Hence the attentive gate model can be trained with any of the benchmark methods.

While there is no conditional computation during feedforward during attentive gate MoE training. There is still conditional weight updates during training depending on the method of computing loss. Hence, the method is still computationally efficient. Distilling the attentive gate model enables conditional computation during inference.

Chapter 8

Expert Loss Gating MoE Architecture

8.1 Motivation

Yet another criterion to train MoE is for the gate to predict the log loss of the experts for a given sample instead of just predicting probabilities of selecting an expert. We can then use these predicted losses as a measure by which to select the expert. This will allow us to train the expert and gate separately.

When the gate just predicts probabilities for expert selection, it can have very large values for some experts. If there is a small increase in the expert performance, a large change in gate probabilities is required to select that expert. But if the gate predicts the expert losses then it can easily learn the change in expert performance quickly. This is the reason why it is not difficult to learn the expert loss. The gate now predicts a more meaningful quantity, the expert loss, than just the expert distribution over the samples which we have seen does not lead to a globally optimal solution.

The experts have the same architecture as that of the previous models. The gate also has the same architecture except for the last layer where we do not apply a *softmax* layer as the gate predicts log loss and not expert distribution. Our experiments show that the *expert loss gate* MoE performs better than all benchmark MoE architectures for some datasets.

8.2 Training the gate to predict expert log loss

In *expert loss gate* MoE we separate training of the experts and the gate. The output of the gate, $\mathbf{g}$, is the log loss, o_{g_i}, of the corresponding expert i. During training and inference, the MoE output, $\hat{\mathbf{y}}$, is the output, $\mathbf{o_{eI}}$, of the expert I with the maximum predicted log loss, as in Equation 8.2. Note that the gate predicts log loss and not negative log log loss. Hence, the best expert is the one with maximum log loss (which is also the minimum negative log loss).

$$I = \arg\max_{i=1 \to M} o_{g_i} \tag{8.1}$$

$$\hat{\mathbf{y}} = \mathbf{o_{eI}} \tag{8.2}$$

The selected experts are trained with loss, L as shown in Equation 8.5, where y is the target class. For each expert output, $\mathbf{o_{ei}}$, we also compute its per sample log loss, L_{ei}, as shown in Equation 8.3 for the i^{th} expert and true class y. o_{iy} is the predicted probability for the true label y by expert i. The gate is trained with loss L_g, shown in Equation 8.4, to minimise the sum of the absolute difference between the predicted log loss, o_{gi}, for the i^{th} expert and the true computed *log* loss for that expert, L_{ei}.

$$L_{ei} = \log(o_{i_y}), i \in \{1, \ldots, M\} \tag{8.3}$$

$$L_g = \sum_{i=1}^{M} |o_{g_i} - L_{ei}| \tag{8.4}$$

$$L = l(y, \hat{\mathbf{y}}) \tag{8.5}$$

$$\tag{8.6}$$

During training we need to compute the loss of each expert to train the gate. Hence there is no conditional computation during feed forward during training. But since only the selected expert's gradient is computed there is conditional computation in back propagation. There is conditional computation during inference as we select only one expert, that has the least loss, as predicted by the gate. Algorithm 8.2.1 summarizes the training with expert loss gate.

Algorithm 8.2.1: Training with expert loss gate

Input: $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{N}, \mathcal{X} \in \mathbb{R}^u\}$
epochs $\in \mathbb{N}$
$M \in \mathbb{N}$ /* number of experts */
$K \in \mathbb{N}$ /* number of classes */
$f_i : \mathcal{X} \to \mathbb{R}^K, i \in \{1, \ldots, M\}$ /* expert neural network */
$g : \mathcal{X} \to \mathbb{R}^M$ /* gate neural network */
$l : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ /* loss function */
gate optimizer O_g
MoE optimizer O
Output: $g(\cdot), f_i(\cdot), i \in \{1, \ldots, M\}, \hat{y} \subset \mathbb{R}^K$
for $epoch = \{1, \ldots, epochs\}$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    $o_{ei} \leftarrow f_i(x), i \in \{1, \ldots, M\}$ /* expert outputs */
    $o_g \leftarrow g(x)$ /* gate output */
    $I \leftarrow \arg\max_{i=1 \to M} o_{g_i}$ /* expert with min loss */
    $\hat{y} \leftarrow o_{eI}$ /* MoE predicted output */
    $L_{ei} \leftarrow \log(o_{i_y}), y \in \mathcal{Y}, i \in \{1, \ldots, M\}$ /* expert log loss */
    $L_g \leftarrow \sum_{i=1}^{M} |o_{g_i} - L_{ei}|$ /* gate loss */
    $L \leftarrow l(y, \hat{y}), y \in \mathcal{Y}$ /* MoE loss and also expert loss */
    compute gate gradients with L_g
    compute expert gradients witj L
    update O
    update O_g
  end
end

8.3 Experiments

We evaluate our methods on the MNIST LeCun and Cortes (2010) and CIFAR-10 Krizhevsky (2009) datasets. For both datasets we ran the experiments with 5 and 10 experts. Details of expert and gate architectures for MNIST and CIFAR-10 are in Appendix A.1 and A.3.

All models were trained with Adam optimizer with 0.001 learning rate. We used 100 epochs for MNIST dataset and 200 epochs for CIFAR-10 dataset. Each experiment was run 10 times for both datasets.

We compared: (1) single model which has the same architecture as one expert; (2) original MoE *output mixture* MoE; (3) *stochastic* MoE; (4) *top-1* MoE; (5) *top-2* MoE; (6) *expert loss gating* MoE.

Tables 8.1 and 8.2 show results for MNIST and CIFAR-10 datasets with 5 experts. Results for both datasets with 10 experts are in Tables B.7 and B.8 in Appendix B.3.

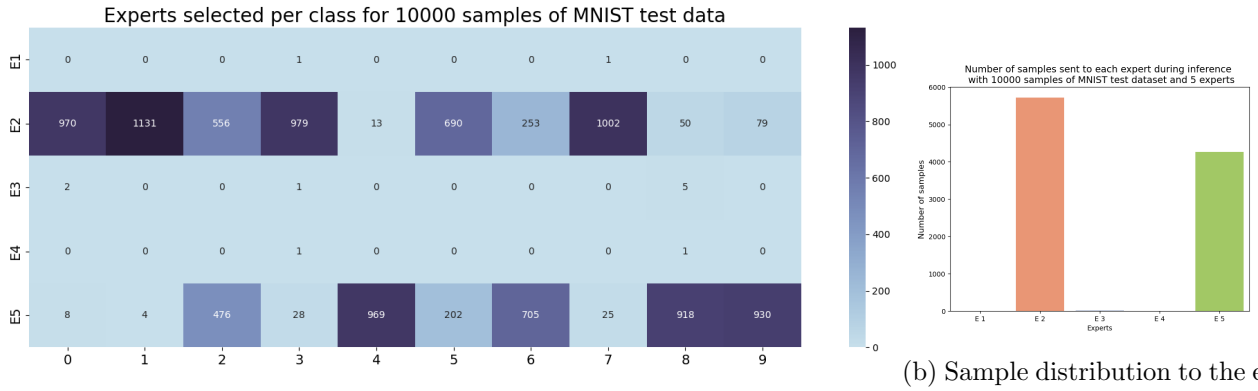
Table 8.1: Performance on the test set by the *expert loss gate* model with the minimum validation error for MNIST dataset with 5 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
output mixture MoE	95.83 ± 0.039	1.568	0.034	1.570
stochastic MoE	96.13 ± 0.021	1.569	0	1.569
top-1 MoE	94.29 ± 0.039	1.000	0	1.000
top-2 MoE	**96.67 ± 0.017**	1.552	0.029	1.570
expert loss gate MoE	93.87 ± 0.0120	.673	0	0.998

Table 8.2: Performance on the test set by the *expert loss gate* model with the minimum validation error for CIFAR-10 dataset with 5 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	42.74 ± 0.011	NA	NA	NA
output mixture MoE	70.44 ± 0.016	1.381	0.066	1.392
stochastic MoE	73.63 ± 0.021	1.889	0	1.960
top-1 MoE	67.46 ± 0.015	0.961	0	0.977
top-2 MoE	**78.89 ± 0.025**	1.845	0.259	1.957
expert loss gate MoE	76.68 ± 0.052	1.611	0	1.666

Tables 8.1 and 8.2 show that for both datasets with 5 experts the *expert loss gate* MoE performs better than some of the benchmark MoEs but not as well as the *top-2* MoE. However, Table B.8 shows that for CIFAR-10 dataset with 10 experts the *expert loss gate* MoE performs better than all the bechmark models. Hence the *expert loss gate* MoE could potentially work very well for some datasets.

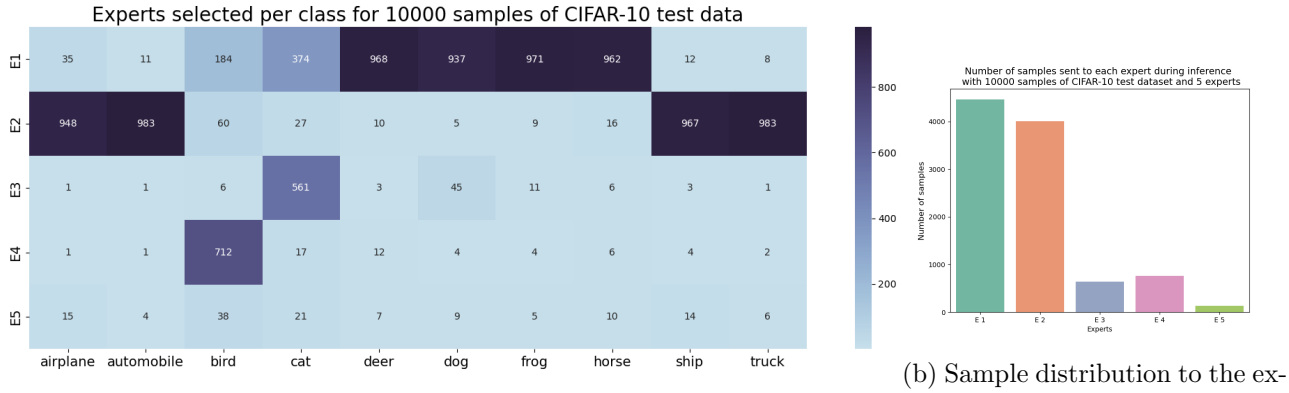


(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 8.1: Gate expert selection table of the *expert loss gate* model on MNIST test dataset. Model is trained with 5 experts.

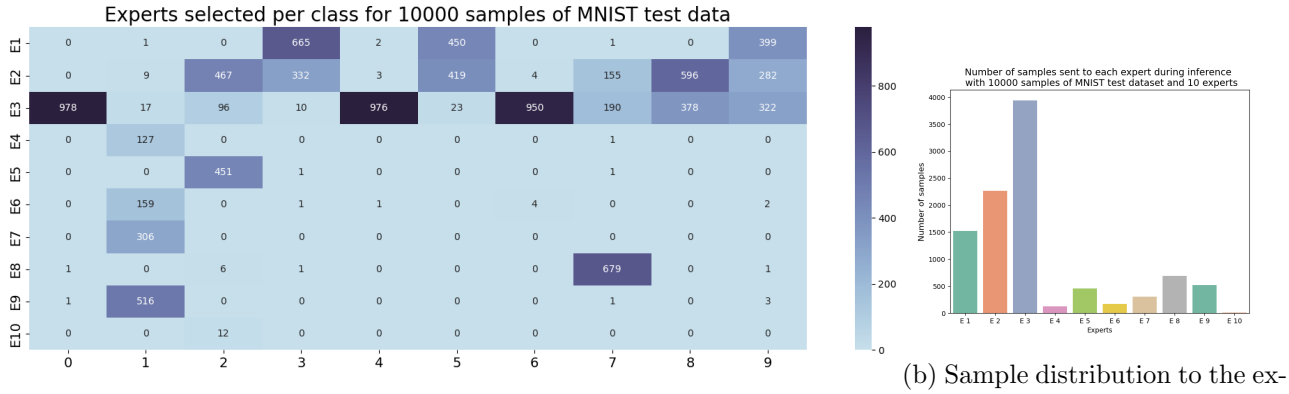
Let us now see how the task is distributed across the experts by the gate using *expert gate*



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

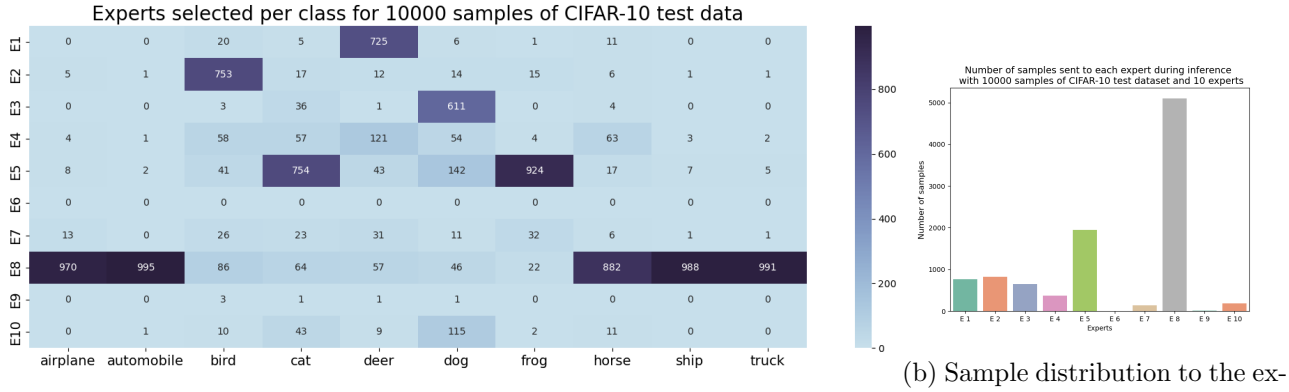
Figure 8.2: Gate expert selection table of the *expert loss gate* model on CIFAR-10 test dataset. Model is trained with 5 experts.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 8.3: Gate expert selection table of the *expert loss gate* model on MNIST test dataset. Model is trained with 10 experts.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 8.4: Gate expert selection table of the *expert loss gate* model on CIFAR-10 test dataset. Model is trained with 10 experts.

loss MoE. Figures 8.1 and 8.2 show the gate expert selection table when the *expert loss gate*, trained with 5 experts, is used to perform inference on the corresponding test dataset. We see that, for both datasets, very few experts are used. We could potentially improve the expert usage with a soft constraint which we will look into in the future.

Figures 8.3 and 8.4 show the gate expert selection table when the *expert loss gate* model, trained with 10 experts, is used to do inference on the corresponding test dataset. We see that for both datasets the expert usage has improved. But some experts are favored more than the others.

8.4 Discussion

The results from the experiments show that the performance of the *expert loss gate* MoE varies over datasets and the number of experts. It beats all the benchmark models without regularizations for some datasets. The *expert loss gate* MoE does not use the experts optimally. As observed earlier, we could increase expert usage through carefully crafted soft contrains. We hope to look into this in the future.

Chapter 9

Gating with Sample Similarity

9.1 Motivation

Previous results have shown that end-to-end training, of gate and experts, does not distribute the samples equitably among the experts. Hence, there is a need for soft constraints to guide the gate to optimally distribute samples to the experts. In Section 3.4 we saw that the soft constraint, $L_{importance}$ proposed by Shazeer et al. (2017), just aims at an equal sample distribution to all the available experts. This is not necessarily always desirable. For example, we may not want an equal distribution of samples to the experts for an imbalanced dataset.

We propose an intuitive data-driven soft regulatization, L_s, based on the properties of the samples in the dataset. It is reasonable to assume that similar samples have similar feature distributions. Our hypothesis here is that, routing similar samples to the same expert and dissimilar samples to different experts will ensure cleaner and more equitable task decomposition. It provides flexibility to incorporate domain knowledge into the training. Similarity measures are however task and sample dependent. The simplest measure to use is the *Euclidean distance* between the samples.

Our experiments, detailed in Section 9.3, show that L_s regularization performs as well as or better than $L_{importance}$ regularization, and additionally uses less experts for similar performance.

9.2 Sample similarity based soft regularization L_s

We describe our L_s approach using a distance measure $d(x, x')$, for pairs of samples $x, x' \in X$, where X is a batch of size N. Any reasonable similarity measure $d(\cdot)$ might work.

We compute a *same expert allocation term*, $S(x, x')$, for each pair of samples, which is a measure of similar samples x and x' being sent to the same expert, as shown in Equation 9.2. We also compute a *different experts allocation term*, $D(x, x')$, for each pair of samples, which is a measure of dissimilar samples x and x' being sent to different experts, as shown in Equation 9.3.

L_s is the combined optimization of the similar and dissimilar expert allocation terms $S(x, x')$ and $D(x, x')$, as shown in Equation 9.1, such that, $S(x, x')$ is minimised and $D(x, x')$ is maximised as the sample distance, $d(x, x')$, increases as shown in Equations 9.2 and 9.3 respectively. M is the number of experts in the model, $e, e' \in E_M$ are the experts assigned to samples x, x' respectively and β_s, β_d are tunable hyperparameters.

$$L_s(X) = \frac{1}{(N^2 - N)} \Big[\sum_{x,x' \in X} \beta_s \cdot S(x, x') - \beta_d \cdot D(x, x') \Big] \tag{9.1}$$

$$S(x, x') = \frac{1}{M} \sum_{e} p(e|x) \cdot p(e|x') \cdot d(x, x') \tag{9.2}$$

$$D(x, x') = \frac{1}{(M^2 - M)} \sum_{e \neq e'} p(e|x) \cdot p(e'|x') \cdot d(x, x') \tag{9.3}$$

Hence, L_s allows the gate to learn expert selection probabilities, $p(e|x)$, based on sample similarity. As sample similarity between samples x, x' decreases, we want the probablity that they be allocated to the same expert to also decrease, that is, $p(e|x) \cdot p(e|x')$ should be small, while the probablity that they be allocated to different experts should increase, that is, $p(e|x) \cdot p(e'|x')$ should be large.

9.3 Experiments

We evaluate our methods on the MNIST LeCun and Cortes (2010) and CIFAR-10 Krizhevsky (2009) datasets. For both datasets we ran the experiments with 5 and 10 experts. Details of

expert and gate architectures for MNIST and CIFAR-10 are at Appendix A.1 and A.3.

All models were trained with Adam optimizer with 0.001 learning rate. We used 100 epochs for MNIST dataset and 200 epochs for CIFAR-10 dataset. Each experiment was run 10 times for both datasets.

For the MNIST dataset theL_s was computed using Euclidean distance between the original MNIST images after flattening them into 1-D from original 2-D images. For CIFAR-10 dataset the Euclidean distance between the original images leads to poor performance. So we first trained a WideResNet (WRN)[1] (Zagoruyko and Komodakis, 2017) model with the full CIFAR-10 training data to learn the representations for the samples. WRN is a much faster training and more accurate residual network. We then compute the Euclidean distance between the representations of the samples produced by the trained WRN model. We found that the performance results improved significantly with the WRN representations of the CIFAR-10 training samples. Note that the representations were used only to compute L_s and not as an input to the experts or gate.

Table 9.1: Performance with $L_{importance}$ and L_s regularizations of models with minimum validation error on the test set for MNIST dataset with 5 experts. Best performance with $L_{importance}$ and L_s are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
output mixture MoE with $L_{importance}$	96.87 ± 0.008	2.280	0.053	2.322
top-2 MoE with $\mathbf{L_{importance}}$	**97.26 ± 0.004**	2.022	0.071	2.322
output mixture MoE with L_s	96.74 ± 0.009	1.877	0.048	1.986
top-2 MoE with L_s	97.06 ± 0.007	1.828	0.034	1.838
output mixture with attentive gate MoE with $L_{importance}$	96.73 ± 0.005	2.321	0.006	2.321
top-2 with attentive gate MoE with $L_{importance}$	97.00 ± 0.003	2.183	0.020	2.322
attentive gate MoE with L_s	96.76 ± 0.004	2.249	0.004	2.249
top-2 with attentive gate MoE with L_s	96.80 ± 0.005	2.156	0.022	2.313
distilled MoE with output mixture MoE with $L_{importance}$	96.77 ± 0.005	2.310	0.009	2.320
distilled MoE with top-2 MoE with $L_{importance}$	96.93 ± 0.025	2.244	0.019	2.321
distilled MoE with output mixture MoE with $\mathbf{L_s}$	**97.09 ± 0.005**	2.251	0.007	2.250
distilled MoE with top-2 MoE with L_s	96.80 ± 0.020	2.148	0.023	2.312

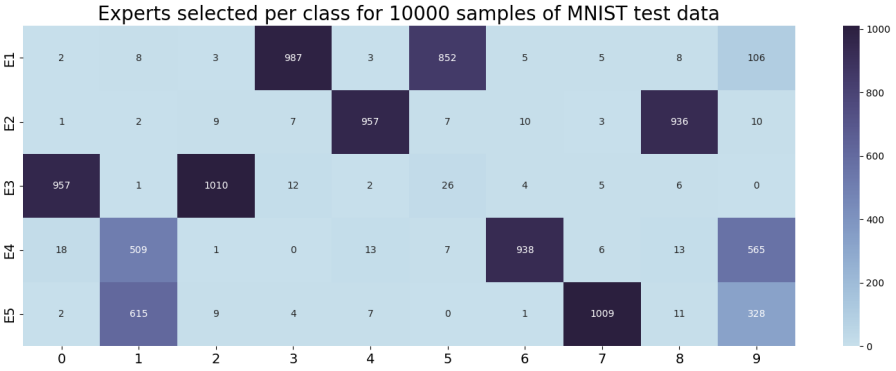
We compared: (1) *output mixture* MoE with $L_{importance}$ regularization for different values of $w_{importance}$; (2) *top-2* MoE with $L_{importance}$ regularization for different values of $w_{importance}$; (3) *output mixture* MoE with L_s regularization for different values of β_s and β_d; (4) *top-2* with L_s

[1] https://github.com/xternalz/WideResNet-pytorch

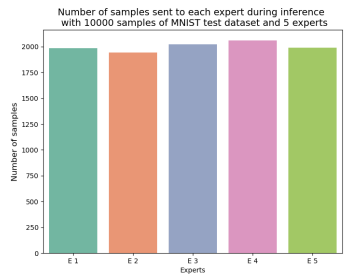
regularization for different values of β_s and β_d; (5) attentive gating with *output mixture* MoE and $L_{importance}$ regularization for different values of $w_{importance}$; (6) *top-2* with attentive gating MoE and $L_{importance}$ regularization for different values of $w_{importance}$; (7) attentive gating *output mixture* method and L_s regularization for different combinations of values of β_s and β_d; (8) *top-2* with attentive gating and L_s regularization for different combinations of values of β_s, β_d;

The experiment results for MNIST dataset with 5 experts are in Table 9.1. The results with 10 experts are in Table B.9. Table 9.1 shows that $L_{importance}$ and L_s regularization perform almost equally well with almost the same accuracies for each model type.

Let us now look at the task decompositions with $L_{importance}$ and L_s for MNIST dataset. We select the best performing model with $L_{importance}$ and the best performing model with L_s for MNIST dataset with 5 and 10 experts.

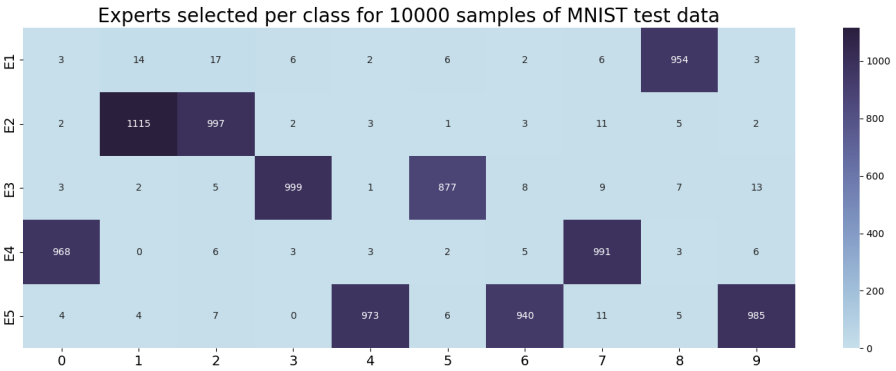


(a) Gate expert selection table

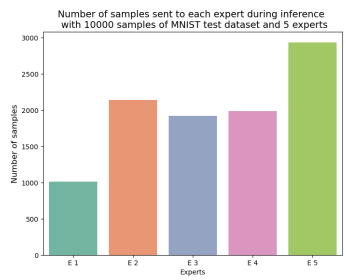


(b) Sample distribution to the experts by the gate

Figure 9.1: Gate expert selection table of the best performing model with $L_{importance}$ with 5 experts, which is the *top-2* model trained with $L_{importance}$ for MNIST test dataset.



(a) Gate expert selection table



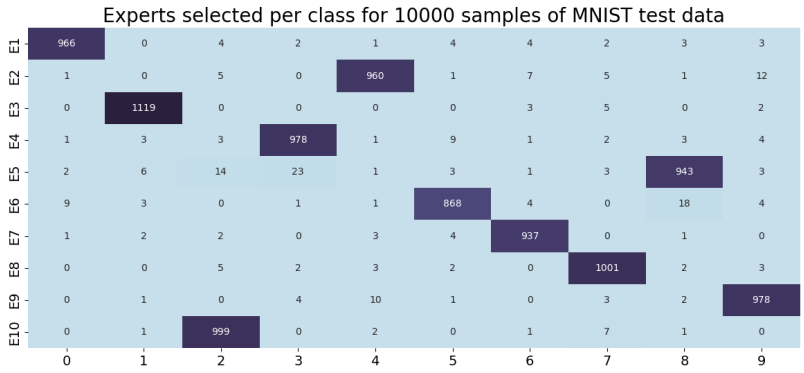
(b) Sample distribution to the experts by the gate

Figure 9.2: Gate expert selection table of the best performing model with L_s with 5 experts, which is the *distilled MoE with output mixture* model trained with L_s for MNIST test dataset.

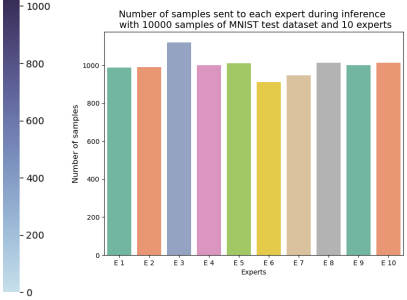
The best performing model with $L_{importance}$ and L_s for MNIST with 5 experts are the *top-2* MoE with $L_{importance}$, shown in Figure 9.1, and distilled MoE with *output mixture* MoE with

L_s, shown in Figure 9.2, respectively. We see that the best performing L_s model with a test accuracy of 97.09% performs as well as the best performing $L_{importance}$ model with test accuracy of 97.26% on the MNIST test dataset with 5 experts.

The gate per class expert selection table and sample distribution plots for L_s in Figure 9.2 show that L_S has a high expert usage as is also indicated by a high expert usage entropy, H_u, in Table 9.1.

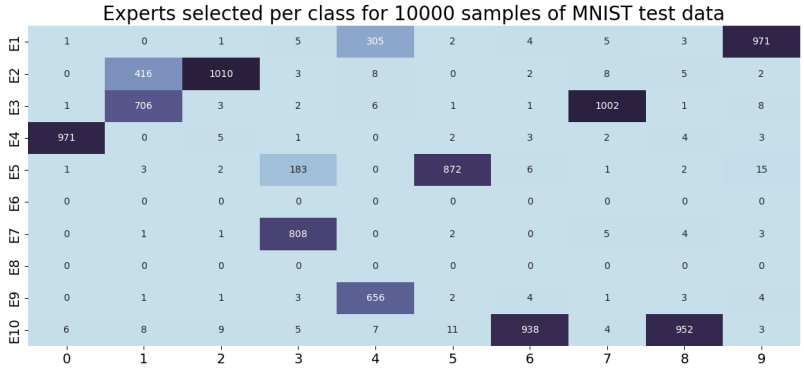


(a) Gate expert selection table

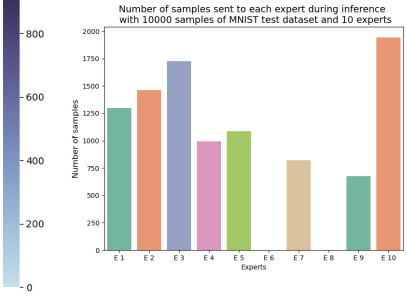


(b) Sample distribution to the experts by the gate

Figure 9.3: Gate expert selection table of the best performing model with $L_{importance}$ with 10 experts, which is the *top-2* model trained with $L_{importance}$ for MNIST test dataset.



(a) Gate expert selection table



(b) Sample distribution to the experts by the gate

Figure 9.4: Gate expert selection table of the best performing model with L_s with 10 experts, which is the *top-2 attentive gate MoE* model trained with L_s for MNIST test dataset.

The best performing model with $L_{importance}$ and L_s for MNIST with 10 experts are the *top-2* MoE with $L_{importance}$, shown in Figure 9.3, and *top-2 with attentive gate MoE* with L_s, shown in Figure 9.4, respectively. We see that the best performing L_s model with a test accuracy of 97.24% performs as well as the best performing $L_{importance}$ with test accuracy of 97.83% on the MNIST test dataset with 10 experts.

The sample distribution plot for L_s in Figure 9.4b also shows that we do not need an equal sample distribution to the experts to achieve good performance. While the $L_{importance}$ model

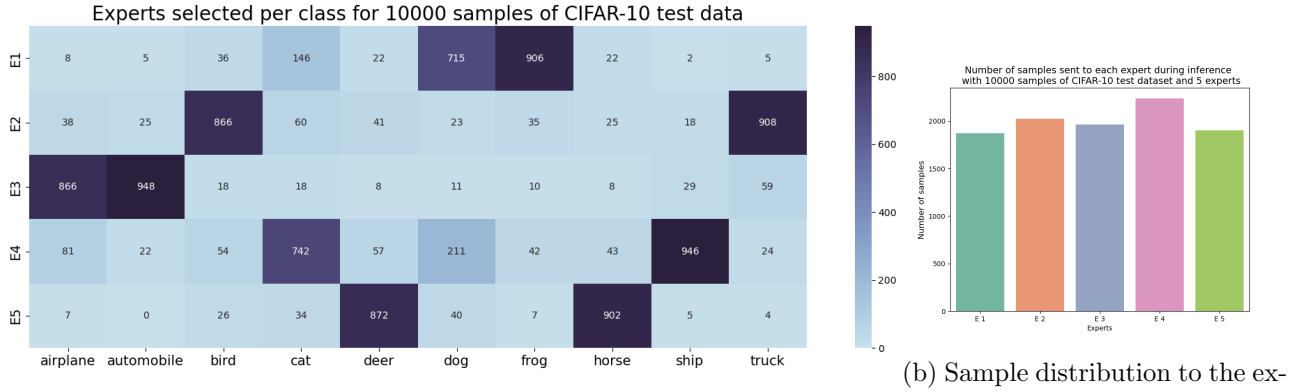
in Figure 9.3b uses all 10 experts to achieve 97.83% (refer to Table B.9 in Appendix B.4.1) test accuracy, the L_s model achieves 97.79% (refer to Table B.9 in Appendix B.4.1) test accuracy with just 8 experts.

The experiment results for CIFAR-10 dataset with 5 experts are in Table 9.2. The results with 10 experts are in Table B.10. Table 9.2 shows that $L_{importance}$ and L_s regularization perform almost equally well with almost the same accuracies for each model type.

Table 9.2: Performance with $L_{importance}$ and L_s regularizations of models with minimum validation error on the test set for CIFAR-10 dataset with 5 experts. Best performance with $L_{importance}$ and L_s are highlighted.

Experiment	**Test Accuracy**	**I(E; Y)**	**H_s**	**H_u**
output mixture MoE with $L_{importance}$	77.61 ± 0.024	2.315	0.195	2.314
top-2 MoE with $L_{importance}$	79.90 ± 0.029	2.317	0.296	2.316
output mixture MoE with L_s	79.00 ± 0.033	2.320	0.167	2.320
top-2 MoE with L_s	80.27 ± 0.026	2.319	0.240	2.319
output mixture with attentive gate MoE with $L_{importance}$	75.81 ± 0.041	2.225	0.020	2.317
top-2 with attentive gate MoE with $L_{importance}$	**84.38** ± 0.007	2.206	0.154	2.319
output mixture with attentive gate MoE with L_s	72.60 ± 0.125	2.314	0.021	2.314
top-2 with attentive gate MoE with L_s	84.24 ± 0.015	2.091	0.134	2.312
distilled MoE with output mixture MoE with $L_{importance}$	81.05 ± 0.018	2.320	0.028	2.322
distilled MoE with top-2 MoE with $L_{importance}$	83.97 ± 0.048	2.228	0.101	2.321
distilled MoE with output mixture MoE with L_s	77.28 ± 0.216	2.309	0.039	2.319
distilled MoE with top-2 MoE with L_s	**84.38** ± 0.040	2.063	0.128	2.307

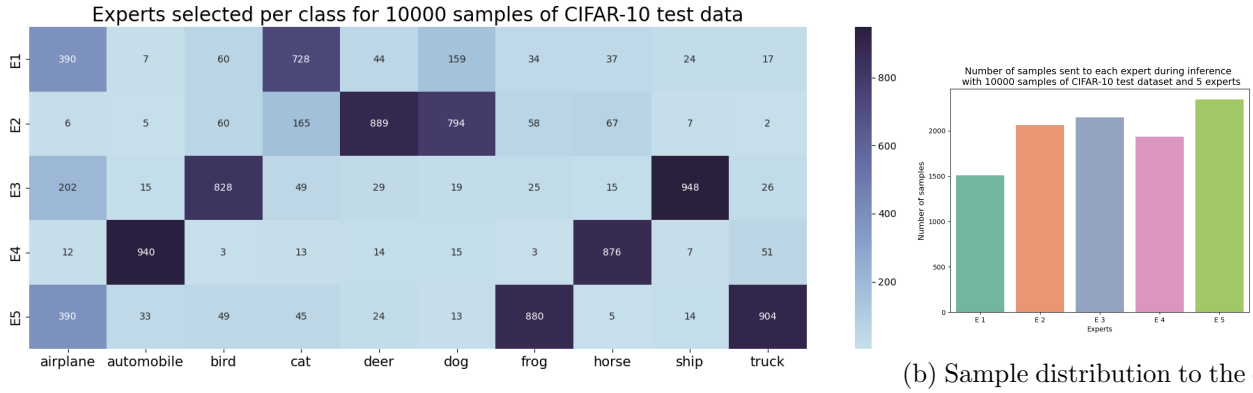
Let us now look at the task decompositions with $L_{importance}$ and L_s for CIFAR-10 dataset. We select the best performing model with $L_{importance}$ and the best performing model with L_s for CIFAR-10 dataset with 5 and 10 experts.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 9.5: Gate expert selection table of the best performing model with $L_{importance}$ with 5 experts, which is the *top-2 attentive gate* model trained with $L_{importance}$ for CIFAR-10 test dataset.



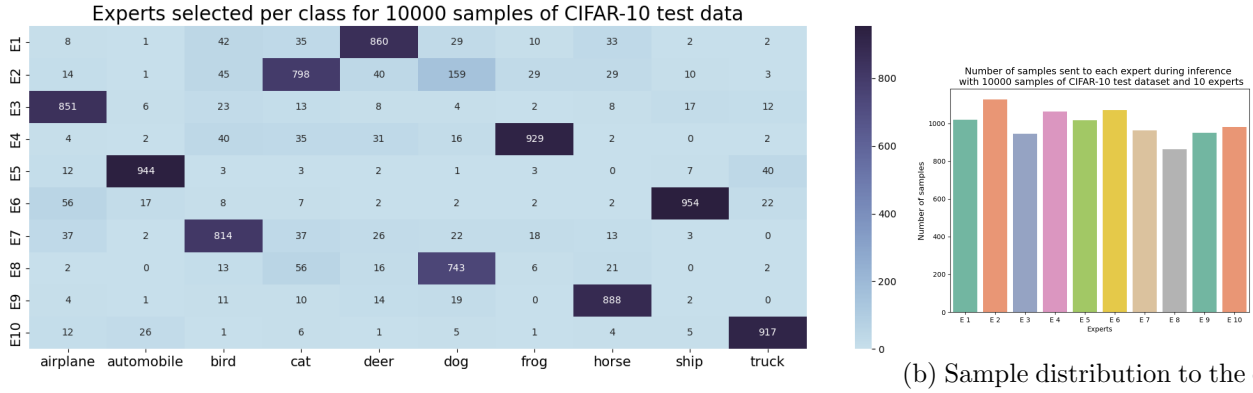
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 9.6: Gate expert selection table of the best performing model with L_s with 5 experts, which is the *distilled MoE with top-2* model trained with L_s for CIFAR-10 test dataset.

The best performing model with $L_{importance}$ and L_s for CIFAR-10 with 5 experts are the distilled MoE with *top-2* MoE with $L_{importance}$, shown in Figure 9.5, and *distilled MoE with top-2* with L_s, shown in Figure 9.6, respectively. We see that the best performing L_s model with a test accuracy of 84.38% performs as well as the best performing $L_{importance}$ model with test accuracy of 84.38% on the CIFAR-10 test dataset.

The sample distribution plots in Figures 9.5b and 9.6b show that we do not need an equal sample distribution to the experts to achieve good performance.

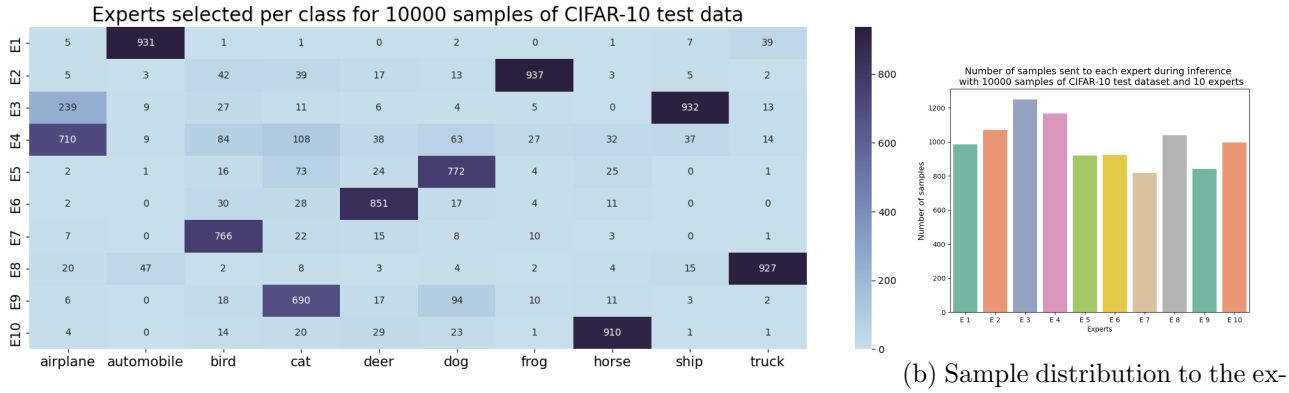


(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 9.7: Gate expert selection table of the best performing model with $L_{importance}$ with 10 experts, which is the distilled MoE with *top-2 attentive gate* model trained with $L_{importance}$ for CIFAR-10 test dataset.

The best performing model with $L_{importance}$ and L_s for CIFAR-10 with 10 experts are the distilled MoE with *top-2* MoE with $L_{importance}$, shown in Figure 9.7, and *attentive MoE with top-2 MoE* with L_s, shown in Figure 9.8, respectively. We see that the best performing L_s model with a test accuracy of 87.46% performs as well as the best performing $L_{importance}$ model with test accuracy of 87.20% on the CIFAR-10 test dataset. The sample distribution plot for



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure 9.8: Gate expert selection table of the best performing model with L_s with 10 experts, which is the *top-2 with attentive gate* model trained with L_s for CIFAR-10 test dataset.

L_s, in Figure 9.8b, shows that L_s achieves good expert usage as is also evident from its high expert usage entropy, H_u, in Table B.10 in Appendix B.4.2.

9.4 Discussion

The results from the experiments show that L_s regularization performs as well as or better than $L_{importance}$ with good expert usage as indicated by high expert usage entropy. It also achieves high expert sparsity as indicated by the low sample entropy, H_s. Additionally L_s uses less experts than $L_{importance}$ for the same performance. L_s is a flexible method that allows using different similarity or other measures based on sample properties for task decomposition. L_s is hence a more data driven approach to distributing the samples than just an equal distribution of samples to experts like $L_{importance}$.

Chapter 10

Conclusion

Gated modular deep neural networks are a promising solution towards achieving interpretability and transferability in deep neural networks. The gate's task decomposition among the simple experts allows error attribution to the experts or the gate. Specialized experts can be transfered to other similar tasks. In Chapter 3 of this thesis we saw how the existing MoE architectures and training algorithms distribute the tasks to the experts. While the *output mixture* (Jacobs et al., 1991b), *stochastic* (Jacobs et al., 1991a) and *top-k* (Shazeer et al., 2017) MoE showed good task decomposition on toy classification datasets, the *EM* (Kirsch et al., 2018) MoE failed even on toy classification dataset and was computationally few orders of magnitude slower. In Section 3.2 we showed through a careful experiment that the end-to-end expert and gate training fails spectacularly even for simple datasets like MNIST. The gate does not find an optimal decomposition that separates the tasks into the experts and results in pathological cases where only a few or just one expert is used. Hand chosen decompositions were vastly better. We have also seen through all the experiments in the thesis that when expert usage entropy, H_u, is high and the gate sparsity, H_s, is low then the MoE model generalizes well and results in specialized experts. To achieve high expert usage and gate sparsity we require soft constraints in addition to the loss while training with existing MoE training methods.

Current MoE research has concentrated more on load balancing for equal expert usage and increase conditional computation and performance and not on task specific expert specialization. In this line of research the experiments are on web scale and they do not report simple experimemts to show what the expert learns. We are more interested in a good separation of

tasks in the experts as we believe that it can facilitate interpretability and transferability. As far as we know ours is the first work that not only looks at performance of the MoE model but also carefully investigates what each expert learns and how the gate distributes the tasks to the experts.

We systematically developed a suite of novel multi-stage MoE training algorithms and architectures that separate the training of experts and gates, outperforming existing methods without regularizations and result in cleaner task decompositions as indicated by the high expert usage entropy, H_u, and high gate sparsity with low per sample entropy, H_s. Our no-gate no-regularization *peeking expert* method is the best performing method of our proposed approaches, outperforming existing methods even with regularizations. Our methods show that multi-stage expert and gate training works well along with distillation.

We proposed a novel data driven soft constraint, L_s, based on the similarity of samples, that performs as well as or better than the existing soft constraint, $L_{importance}$ by Shazeer et al. (2017). $L_{importance}$ is a simple and very effective soft constraint. But unlike $L_{importance}$ and other exisiting soft constraints and load balancing methods in current literature, that aim at equal sample distribution to experts, L_s distributes the samples based on the subtask they belong to. L_s is also flexible as we can use different similarity functions for task decomposition. We have empirically tested our methods on MNIST, FMNIST and CIFAR-10 datasets.

In the immediate future we would like to explore the *peeking expert* algorithm for regression and unsupervised data. This would require exploring measures we can use to evaluate the best expert for a given sample. We evaluated our methods on image classification datasets. We would like to evaluate them with other regression and unsupervised datasets and see how our methods scale with large datasets. We would also like to explore how we can attribute errors to experts and gate and also evaluate transferability of the MoE models.

Bibliography

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. Jan. 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

D. H. Ballard. Modular learning in neural networks. In *AAAI'87: Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, Seattle, Washington, 1987. AAAI Press. ISBN 0934613427.

Z. Chen, Y. Deng, Y. Wu, Q. Gu, and Y. Li. Towards understanding the mixture-of-experts layer in deep learning. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=MaYzugDmQV.

I. Drori, **Yamuna Krishnamurthy**, R. de Paula Lourenco, R. Rampin, K. Cho, C. Silva, and J. Freire. **Automatic Machine Learning by Pipeline Synthesis using Model-Based Reinforcement Learning and a Grammar**. In *AutoML Workshop at ICML*, 2019.

W. Fedus, B. Zoph, and N. Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23:1–40, 2022. ISSN 15337928.

M. Gimelfarb, S. Sanner, and C.-G. Lee. *Reinforcement Learning with Multiple Experts: A Bayesian Model Combination Approach*. Curran Associates, Inc., 2018.

H. Hazimeh, Z. Zhao, A. Chowdhery, M. Sathiamoorthy, Y. Chen, R. Mazumder, L. Hong, and E. Chi. DSelect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors,

Advances in Neural Information Processing Systems, 2021. URL https://openreview.net/forum?id=tKlYQJLYN8v.

J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang. Fastmoe: A fast mixture-of-expert training system, 2021.

H. Hihn and D. A. Braun. Mixture-of-variational-experts for continual learning. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022. URL https://openreview.net/forum?id=HhzzNQPZLb9.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015. URL http://arxiv.org/abs/1503.02531. NIPS 2014 Deep Learning Workshop.

G. E. Hinton. Products of experts. *IEE Conference Publication*, 1(470):1–6, 1999. ISSN 05379989. doi: 10.1049/cp:19991075.

R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixture of local expert. *Neural Computation*, 3:78–88, 02 1991a. doi: 10.1162/neco.1991.3.1.79.

R. A. Jacobs, M. I. Jordan, and A. G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science"*, 15 (2):219 – 250, 1991b. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(91)80006-Q.

R. Jago, A. van der Gaag, K. Stathis, I. Petej, P. Lertvittayakumjorn, **Yamuna Krishnamurthy**, Y. Gao, J. C. Silva, M. Webster, A. Gallagher, and Z. Austin. **Use of Artificial Intelligence in Regulatory Decision-Making**. *Journal of Nursing Regulation*, 12(3): 11–19, 2021. ISSN 2155-8256. doi: https://doi.org/10.1016/S2155-8256(21)00112-5. URL https://www.sciencedirect.com/science/article/pii/S2155825621001125.

M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Proceedings of the International Joint Conference on Neural Networks*, 2:1339–1344, 1993. ISSN 0899-7667.

L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. One model to learn them all. *CoRR*, abs/1706.05137, 2017.

L. Kirsch, J. Kunze, and D. Barber. Modular networks: Learning to decompose neural computation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett,

editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

C. A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 2*, AAAI'90, pages 923–928. AAAI Press, 1990. ISBN 0-262-51057-X.

Y. Krishnamurthy and C. Watkins. **Interpretability in Gated Modular Neural Networks**. In *Explainable AI approaches for debugging and diagnosis Workshop at NeurIPS*, 2021.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

S. Kudugunta, Y. Huang, A. Bapna, M. Krikun, D. Lepikhin, M. Luong, and O. Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 3577–3599. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-emnlp.304. URL https://doi.org/10.18653/v1/2021.findings-emnlp.304.

Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

P. Lertvittayakumjorn, I. Petej, Y. Gao, **Yamuna Krishnamurthy**, A. Van Der Gaag, R. Jago, and K. Stathis. **Supporting Complaints Investigation for Nursing and Midwifery Regulatory Agencies**. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 81–91, Online, Aug. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.acl-demo.10.

M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer. BASE Layers: Simplifying Training of Large, Sparse Models. 2021. URL http://arxiv.org/abs/2103.16716.

J. Liu, C. Desrosiers, and Y. Zhou. Att-MoE: Attention-based Mixture of Experts for nuclear and cytoplasmic segmentation. *Neurocomputing*, 411:139–148, 2020. ISSN 18728286. doi: 10.1016/j.neucom.2020.06.017. URL https://doi.org/10.1016/j.neucom.2020.06.017.

A. Makkuva, P. Viswanath, S. Kannan, and S. Oh. Breaking the gridlock in mixture-of-experts: Consistent and efficient algorithms. In *ICML*, 2019.

D. Mihai and A. Lascarides. *Combining a Mixture of Experts with Transfer Learning in Complex Games*. AAAI Press, Stanford, 2017.

S. Mittal, Y. Bengio, and G. Lajoie. Is a Modular Architecture Enough? pages 1–46, 2022. URL http://arxiv.org/abs/2206.02713.

J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503. Association for Computational Linguistics, Apr. 2021. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39.

S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Yazdani Aminabadi, A. A. Awan, J. Rasley, and Y. He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. ArXiv, January 2022.

C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. S. Pinto, D. Keysers, and N. Houlsby. Scaling Vision with Sparse Mixture of Experts. *Advances in Neural Information Processing Systems*, 11:8583–8595, 2021. ISSN 10495258.

S. Roller, S. Sukhbaatar, a. szlam, and J. Weston. Hash layers for large sparse models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17555–17566. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/92bf5e6240737e0326ea59846a83e076-Paper.pdf.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. https://arxiv.org/abs/1701.06538, 2017.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, Aug. 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1.

Yamuna Krishnamurthy and C. Watkins. **Interpretability in Gated Modular Neural Networks**. In *Explainable AI approaches for debugging and diagnosis Workshop at NeurIPS*, 2021.

M. Vasic, A. Petrovic, K. Wang, M. Nikolic, R. Singh, and S. Khurshid. Mo{et}: Interpretable and verifiable reinforcement learning via mixture of expert trees. https://openreview.net/forum?id=BJlxdCVKDB, 2020.

T. Veniat, L. Denoyer, and M. Ranzato. Efficient continual learning with modular networks and task-driven priors. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=EKV158tSfwv.

X. Wang, F. Yu, R. Wang, Y. Ma, A. Mirhoseini, T. Darrell, and J. E. Gonzalez. Deep mixture of experts via shallow embedding. *CoRR*, abs/1806.01531, 2018.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. https://github.com/zalandoresearch/fashion-mnist, 2017. URL http://arxiv.org/abs/1708.07747.

S. Zagoruyko and N. Komodakis. Wide residual networks, 2017.

Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. Dai, Z. Chen, Q. Le, and J. Laudon. Mixture-of-experts with expert choice routing, 2022. URL https://arxiv.org/abs/2202.09368.

S. Zuo, X. Liu, J. Jiao, Y. J. Kim, H. Hassan, R. Zhang, J. Gao, and T. Zhao. Taming sparsely activated transformer with stochastic experts. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=B72HXs80q4.

Appendix A

Neural Network Architecture and Parameter Details for MoE Models in Experiments

The number of experts in the MoE model chosen, for the MNIST, FMNINST, CIFAR-10 datasets, is 5 and 10 experts. 5 is half the number of classes in these datasets. 10 is the same number of experts are the classes. We chose experts to be half the number of classes for the dataset as this allows for an equitable distribution of 2 classes per expert (all datasets used in the paper have even number of classes). And with 10 experts we can see how the tasks are distributed when there is potentially an expert for each class. All the MoE models used in the paper have one 1 gate.

For each dataset we tried different expert and gate architectures and parameters for the original MoE model. We then chose the MoE model with minimum validation error,for the given dataset. We used the same expert and gate architectures and parameters of the selected MoE model for all the training methods on that dataset.

We used PyTorch for our implementation. All experiments were run on a single GPU.

A.1 MoE model for MNIST dataset

MoE model for MNIST dataset has 5 experts and 1 gate. The training set has $60,000$ samples which was split into $50,000$ training samples and $10,000$ validation samples. The test set has $10,000$ samples. There are 10 classes.

Expert: Each expert of all the MoE models has 1 convolutional layer, 2 hidden layers and 1 output layer. The details of the layers are as follows:

- 1 convolutional layer with 1 input channel, 1 output channel and a kernel size of 3, with $ReLU$ activation and max pooling with kernel size 2 and stride 2,
- 2 hidden layers with $ReLU$ activation. First hidden layer has input of $1 * 13 * 13$ and output of 5. Second hidden layer has input of 5 and output of 32,
- 1 output layer with input 32 and output 10, which is the number of classes, with $ReLU$activation and
- softmax layer

Original Gate: The gate for the original MoE model has 1 convolutional layer, 2 hidden layers and 1 output layer. The details of the layers are as follows:

- 1 convolutional layer with 1 input channel, 1 output channel and a kernel size of 3, with $ReLU$ activation and max pooling with kernel size 2 and stride 2,
- 2 hidden layers with $ReLU$ activation. First hidden layer has input of $1 * 13 * 13$ and output of 128. Second hidden layer has input of 128 and output of 32,
- 1 output layer with input 32 and output 5, which is the number of experts, with $ReLU$activation
- softmax layer

Attentive Gate: The attentive gate has 1 convolutional layer and 2 hidden layers. The details of the layers are as follows:

- 1 convolutional layer with 1 input channel, 1 output channel and a kernel size of 3, with $ReLU$ activation and max pooling with kernel size 2 and stride 2. This is the same as the original gate,

- 2 hidden layers. First hidden layer has input of $1 * 13 * 13$ and output of 128 with *ReLU* activation, this is the same as the orginal gate for the first hidden layer. Second hidden layer has input of 128 and output of 32 and no activation. This is the output of the attentive gate used to compute the query and attention score.
- There are no output and softmax layers.

Expert Loss Gate: The expert loss gate has:

- The same number of convolutional layers with the same number of parameters and structure as the original gate,
- 2 hidden layers with *ReLU* activation. First hidden layer has input of $1 * 13 * 13$ and output of 128. Second hidden layer has input of 128 and output of 32,
- 1 output layer with input 32 and output 5, which is the number of experts, with $ReLU$activation
- There is no softmax layer as we want to learn expert log loss and not a distribution.

A.2 MoE model for combined FashionMNIST (FMNIST) and MNIST dataset

MoE model for combined FNIST and MNIST dataset has 6 experts and 1 gate. The training set has $10,000$ samples and the test set has $2,000$ samples. We chose the first 6 classes, *[t-shirt, trouser, pullover, dress, coat, sandal]*, from FMNIST and last 6 classes,*[4, 5, 6, 7, 8, 9]*, from MNIST and combined the data to create one dataset of 12 classes.

Expert: Each expert, of all the MoE models for the combined FMNIST and MNIST dataset, has the same architecture and parameters as that for the MNIST dataset in Appendix A.1. Only the output layer output is 12 as the combined FMNIST and MNIST dataset has 12 classes.

Original Gate: The gate for the MoE model has the same architecture as that for the MNIST dataset in Appendix A.1, but with different parameters.

- 1 convolutional layer with 1 input channel, 1 output channel and a kernel size of 5, with $ReLU$ activation and max pooling with kernel size 2 and stride 2,
- 2 hidden layers with $ReLU$ activation. First hidden layer has input of $1 * 12 * 12$ and output of 128. Second hidden layer has input of 128 and output of 32,
- 1 output layer with input 32 and output 6, which is the number of experts, with $ReLU$activation
- softmax layer

A.3 MoE model for CIFAR-10 dataset

MoE model for CIFAR-10 dataset has 5 experts and 1 gate. The training set has $50,000$ samples which was split into $40,000$ training samples and $10,000$ validation samples. The test set has $10,000$ samples. There are 10 classes.

Expert: Each expert of all the MoE models for the CIFAR-10 dataset has:

- 4 convolutional layers. All the convolutional layers have a kernel size of 3. All the max pooling layers have kernel size 2 and stride 2. The following are parameter details of each convolutional layer.
  - First convolutional layer has 3 input channels, 3 output channels with $ReLU$ activation and max pooling,
  - Second convolutional layer has 3 input channels, 6 output channels with batch normalization, $ReLU$ activation and max pooling,
  - Third convolutional layer has 6 input channels, 12 output channels with $ReLU$ activation and max pooling,
  - Fourth convolutional layer has 12 input channels, 12 output channels with batch normalization, $ReLU$ activation and max pooling,
- 2 hidden layers with $ReLU$ activation. First hidden layer has input of $12 * 4 * 4$ and output of 64. Second hidden layer has input of 64 and output of 32,
- 1 output layer with input 32 and output 10, which is the number of classes, with $ReLU$ activation and

- softmax layer

Original Gate: The gate for the original MoE model has:

- 4 convolutional layers. All the convolutional layers have a kernel size of 3. All the max pooling layers have kernel size 2 and stride 2. The following are the parameter details of each convolutional layer.
  - First convolutional layer has 3 input channels, 64 output channels with $ReLU$ activation and max pooling,
  - Second convolutional layer has 64 input channels, 128 output channels with batch normalization, $ReLU$ activation and max pooling,
  - Third convolutional layer has 128 input channels, 256 output channels with $ReLU$ activation and max pooling,
  - Fourth convolutional layer has 256 input channels, 256 output channels with batch normalization, $ReLU$ activation and max pooling,
- 2 hidden layers with $ReLU$ activation. First hidden layer has input of $256 * 4 * 4$ and output of 512. Second hidden layer has input of 512 and output of 32,
- 1 output layer with input 32 and output 5, which is the number of experts, with $ReLU$ activation and
- softmax layer

Attentive Gate: The attentive gate has:

- The same number of convolutional layers with the same number of parameters and structure as the original gate,
- 2 hidden layers. First hidden layer has input of $256 * 4 * 4$ and output of 512 with $ReLU$ activation, this is the same as the orginal gate for the first hidden layer. Second hidden layer has input of 512 and output of 32 and no activation. This is the output of the attentive gate used to compute the query and attention score.
- There are no output and softmax layers.

Expert Loss Gate: The expert loss gate has:

- The same number of convolutional layers with the same number of parameters and structure as the original gate,

- 2 hidden layers with *ReLU* activation. First hidden layer has input of $256 * 4 * 4$ and output of 512. Second hidden layer has input of 512 and output of 32,
- 1 output layer with input 32 and output 5, which is the number of experts, with *ReLU* activation
- There is no softmax layer as we want to learn expert log loss and not a distribution.

Table A.1: Values of hyperparameters (H) β_s and β_d for datasets (D).

D/H	$\beta_{\mathbf{s}}$	$\beta_{\mathbf{d}}$
MNIST	{1e-6, 1e-5}	$\{10^{-i} \mid i \in \{1, \ldots, 6\}\}$
CIFAR-10	{1e-7, 1e-3}	$\{10^{-i} \mid i \in \{1, \ldots, 7\}\}$

A.4 Hyperparameter Values Used for Experiments

In our experiments we trained each model with different values of the corresponding hyperparameters. We then chose the model with the lowest training error for each category of the model and training methods. The hyperparameters we tuned are $w_{importance}$ for $L_{importance}$ regularization and β_s and β_d for L_s regularization.

The values used for the $w_{importance}$ hyperparameter of the $L_{importance}$ regularization, for all datasets, are $w_{importance}$=$\{0.2, 0.4, 0.6, 0.8, 1.0\}$.

The values used for β_s and β_d hyperparameters of the L_s regularization, for different datasets are summarized in Table A.1

Appendix B

Additional Experiment Results

B.1 Results for no-gate MoE experiments

B.1.1 MNIST inference with *loudest expert* with 10 experts

Table B.1: Performance of the *loudest expert* model. Results are inference on MNIST test data with models with minimum validation error with 10 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
output mixture MoE	95.38 ± 0.021	1.488	0.031	1.490
stochastic MoE	96.20 ± 0.028	1.405	0	1.516
top-1 MoE	94.83 ± 0.335	1.357	0	1.357
top-2 MoE	96.25 ± 0.007	1.663	0.074	1.912
output mixture MoE with $L_{importance}$	96.87 ± 0.008	2.280	0.053	2.322
top-2 MoE with $\mathbf{L_{importance}}$	**97.26 ± 0.004**	2.022	0.071	2.322
loudest expert method	95.98 ± 0.003	1.864	0	3.273

B.1.2 CIFAR-10 inference with *loudest expert* with 10 experts

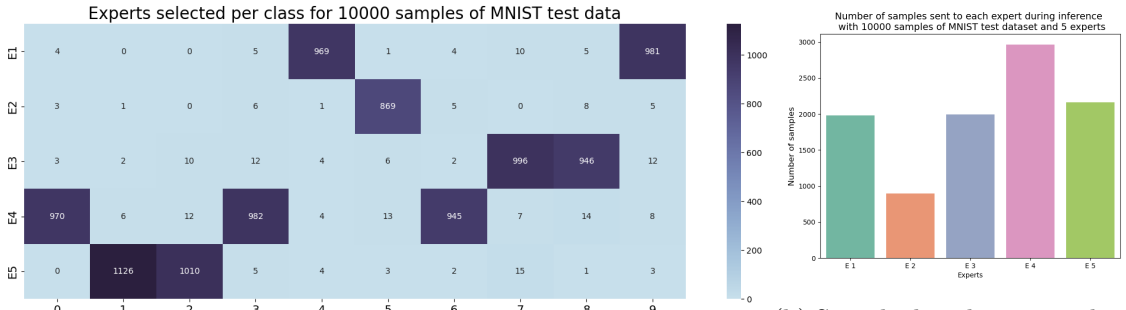
Table B.2: Performance of the *loudest expert* model. Results are inference on CIFAR-10 test data with models with minimum validation error with 10 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	42.74 ± 0.011	NA	NA	NA
original MoE	71.70 ± 0.018	1.839	0.161	1.867
stochastic MoE	72.23 ± 0.013	1.367	0	1.395
top-1 MoE	66.45 ± 0.034	0.969	0	0.991
top-2 MoE	77.60 ± 0.021	1.710	0.146	1.731
output mixture MoE with $L_{importance}$	81.49 ± 0.026	3.304	0.262	3.304
top-2 MoE with $L_{importance}$	81.73 ± 0.132	3.311	0.376	3.310
output mixture MoE with $L_{importance}$	97.72 ± 0.010	3.319	0.053	3.319
top-2 MoE with $\mathbf{L_{importance}}$	**97.83 ± 0.005**	3.283	0.054	3.320
loudest expert method	73.35 ± .006	0.659	0	3.163

B.1.3 MNIST inference with *peeking expert* with 10 experts

Table B.3: Performance of the *peeking expert* model. Results are inference on MNIST test data with models with minimum validation error with 10 experts. Best result is highlighted.

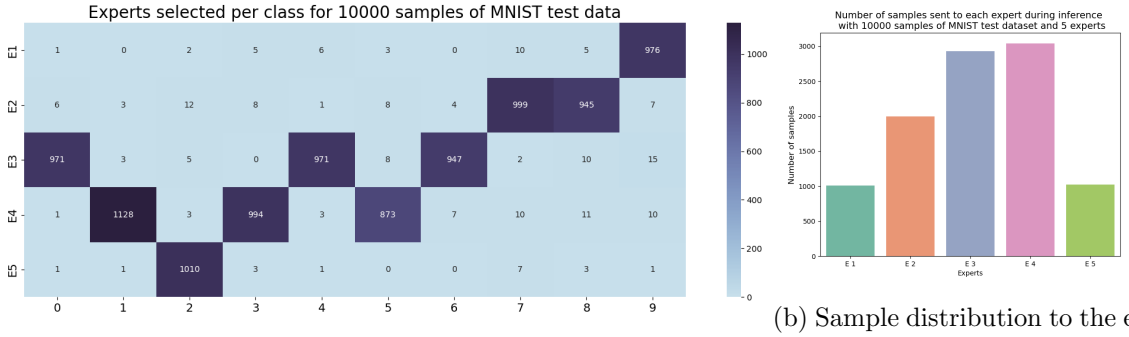
Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
original MoE	95.38 ± 0.021	1.488	0.031	1.490
stochastic MoE	96.20 ± 0.028	1.405	0	1.516
top-1	94.83 ± 0.335	1.357	0	1.357
top-2	96.25 ± 0.007	1.663	0.074	1.912
output mixture MoE with $L_{importance}$	97.72 ± 0.010	3.319	0.053	3.319
top-2 MoE with $\mathbf{L_{importance}}$	**97.83 ± 0.005**	3.283	0.054	3.320
reverse distilled gate, with peeking experts, trained with output mixture MoE	**97.27 ± 0.006**	3.252	0.010	2.257
reverse distilled gate, with peeking experts, trained with stochastic MoE	**97.72 ± 0.005**	3.316	0	3.319
reverse distilled gate, with peeking experts, trained with top-1	96.339 ± 0.008	3.185	0	2.212
reverse distilled gate, with peeking experts, trained with top-2	96.502 ± 0.008	2.403	0.029	1.694



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

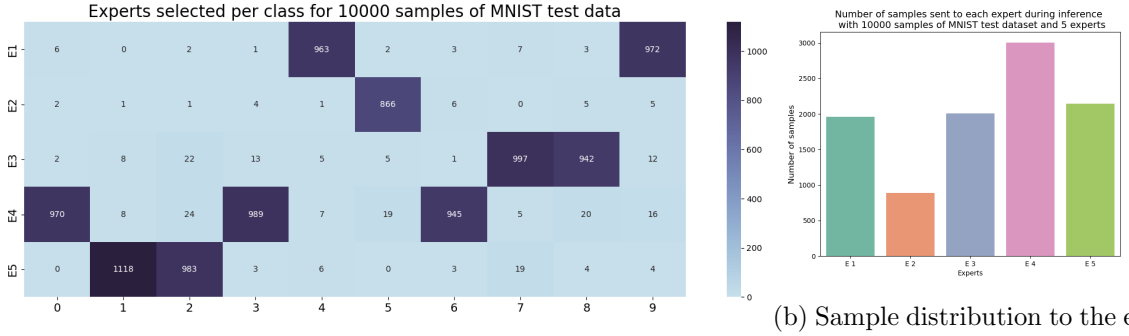
Figure B.1: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4a for MNIST dataset. Gate is trained using *output mixture* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

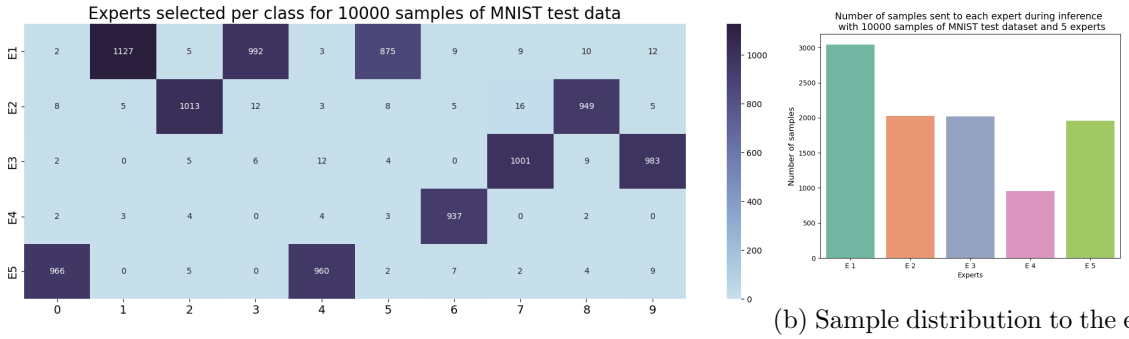
Figure B.2: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4a for MNIST dataset. Gate is trained using *stochastic* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

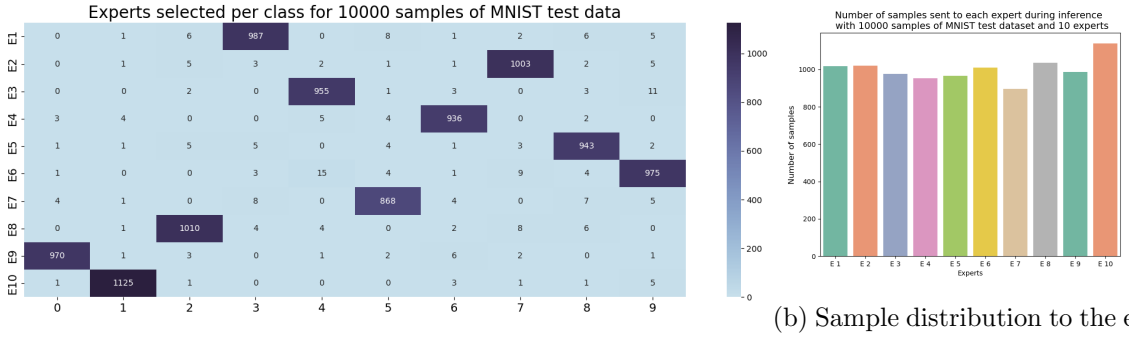
Figure B.3: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4a for MNIST dataset. Gate is trained using output *top-1* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

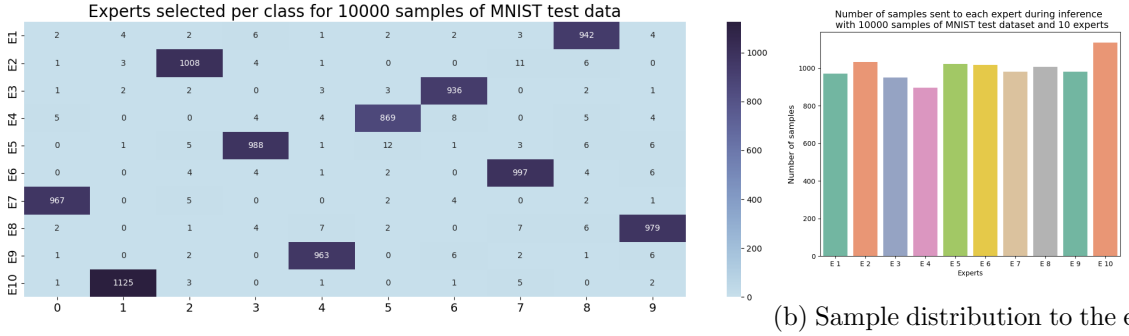
Figure B.4: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4a for MNIST dataset. Gate is trained using output *top-2* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

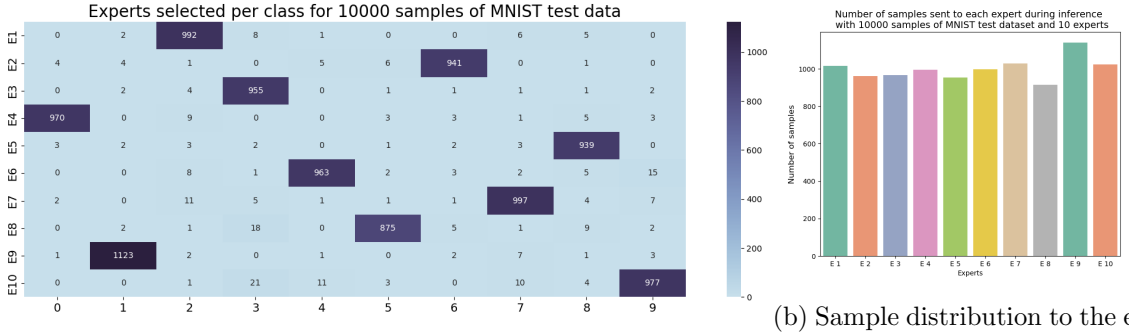
Figure B.5: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4b for MNIST dataset. Gate is trained using *output mixture* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

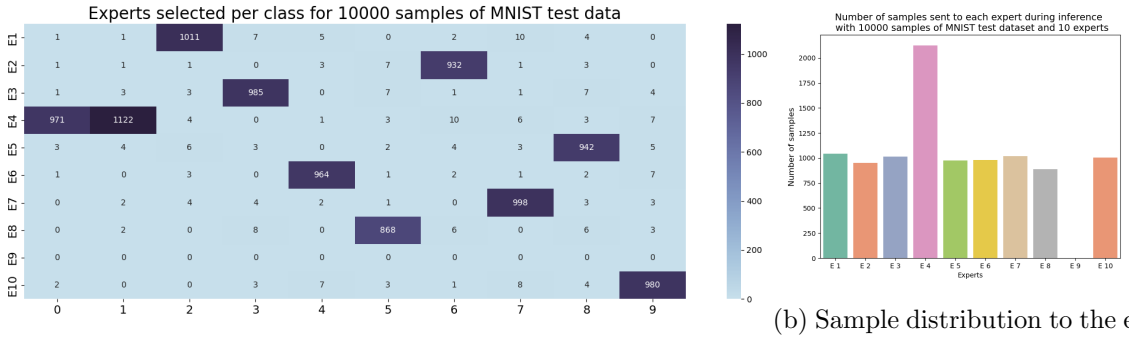
Figure B.6: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4b for MNIST dataset. Gate is trained using *stochastic* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.7: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4b for MNIST dataset. Gate is trained using output *top-1* model.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.8: Gate expert selection table of the reverse distilled gated MoE model trained in Step 2 of the *peeking expert* method from the pre-trained experts in Step 1, shown in Figure 6.4b for MNIST dataset. Gate is trained using output *top-2* model.

B.1.4 CIFAR-10 inference with *peeking expert* with 10 experts

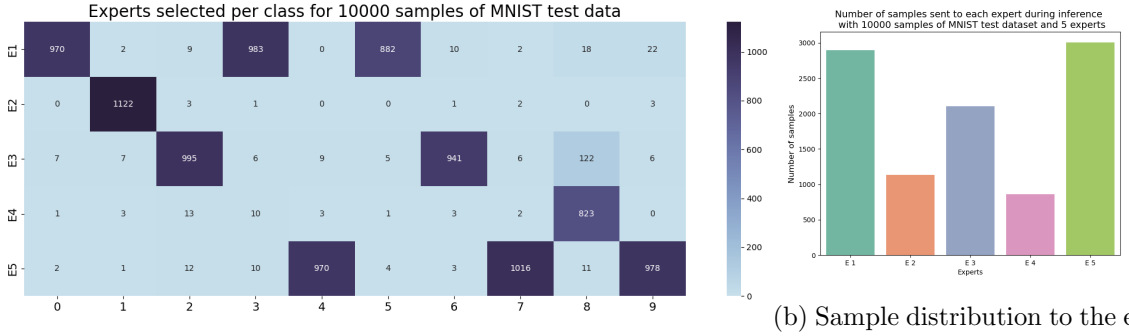
Table B.4: Performance of the *peeking expert* model. Results are inference on CIFAR-10 test data with models with minimum validation error with 10 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
original MoE	71.70 ± 0.018	1.839	0.161	1.867
stochastic MoE	72.23 ± 0.013	1.367	0	1.395
top-1 MoE	66.45 ± 0.034	0.969	0	0.991
top-2 MoE	77.60 ± 0.021	1.710	0.146	1.731
output mixture MoE with $L_{importance}$	81.49 ± 0.026	3.304	0.262	3.304
top-2 MoE with $L_{importance}$	81.73 ± 0.132	3.311	0.376	3.310
reverse distilled gate, with peeking experts, trained with original MoE	87.87 ± 0.003	3.320	0.1	3.320
reverse distilled gate, with peeking experts, trained with stochastic MoE	**88.00 ± 0.003**	3.320	0	3.320
reverse distilled gate, with peeking experts, trained with top-1	77.05 ± 0.018	3.309	0	3.309
reverse distilled gate, with peeking experts, trained with top-2	86.29 ± 0.013	3.032	0.145	3.113

B.2 Results for attentive gate architecture experiments

B.2.1 MNIST inference with attentive gate models with 5 experts

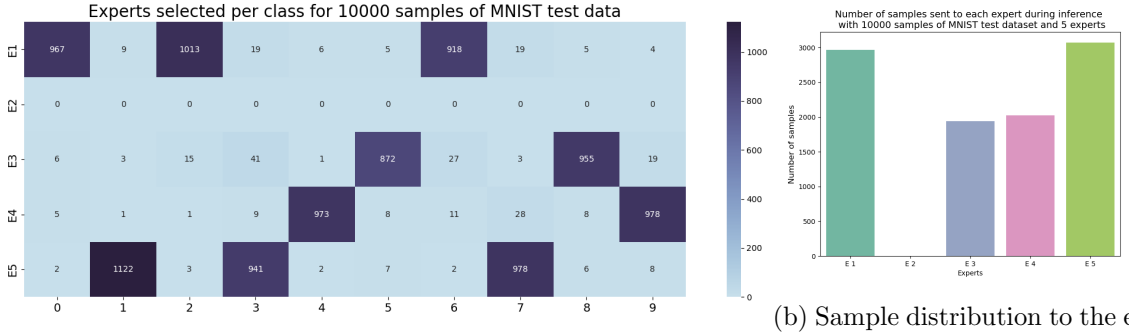
Figures B.9, B.10 and B.11 show the experts used during inference, on the test set, using attentive gate model trained with *output mixture*, *top-1* and *top-2* methods for MNIST dataset with 5 experts. We see that there is good expert usage and clean decompositions for all methods of training.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

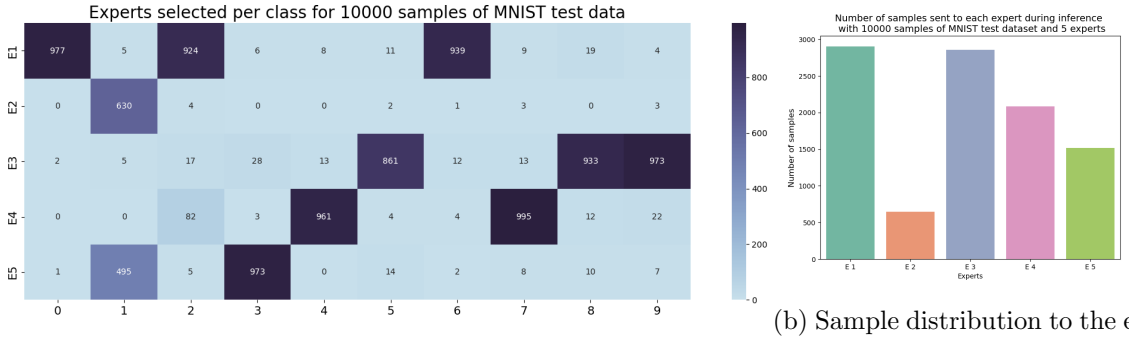
Figure B.9: Gate expert selection table of the attentive gate model trained with 5 experts using *output mixture* model on MNIST test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.10: Gate expert selection table of the attentive gate model trained with 5 experts using top-1 model on MNIST test dataset.



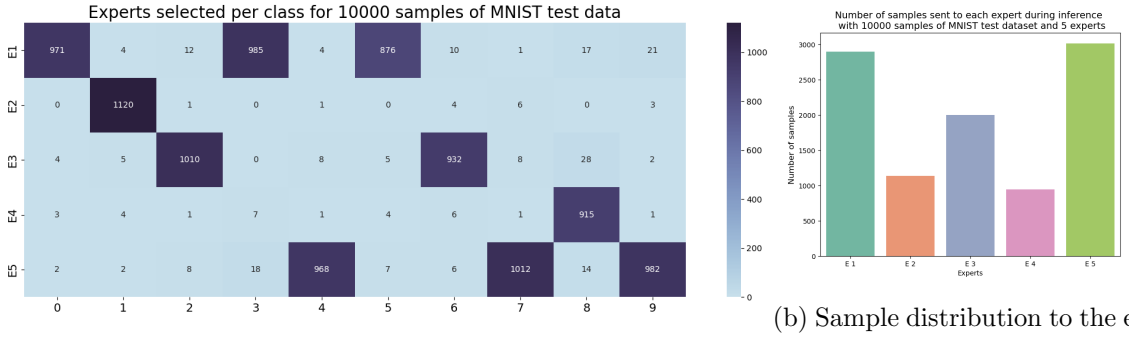
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.11: Gate expert selection table of the attentive gate model trained with 5 experts using top-2 model on MNIST test dataset.

B.2.2 MNIST inference with distilled model with 5 experts

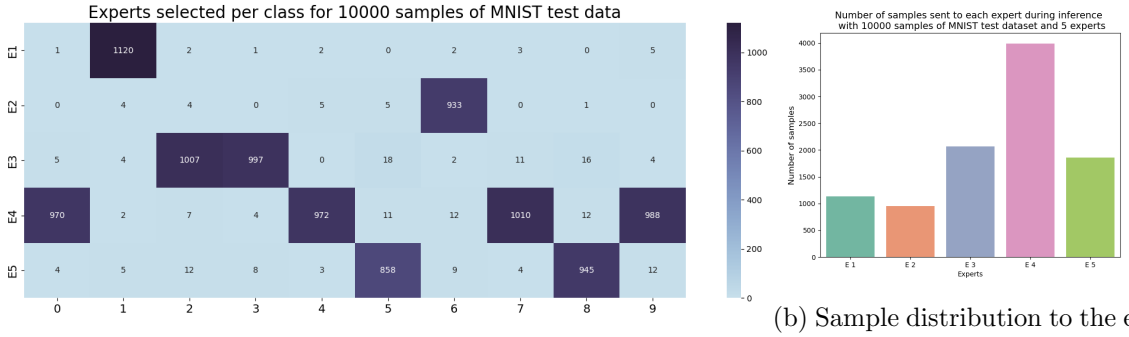
Figures B.12, B.13 and B.14 show the experts used during inference, on the test set, using distilled model trained with *output mixture*, *top-1* and *top-2* methods for MNIST dataset with 5 experts. We see that there is good expert usage and clean decompositions for all methods of training. The performance of the distilled models is comparable to the corresponding attentive models they were distilled from.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

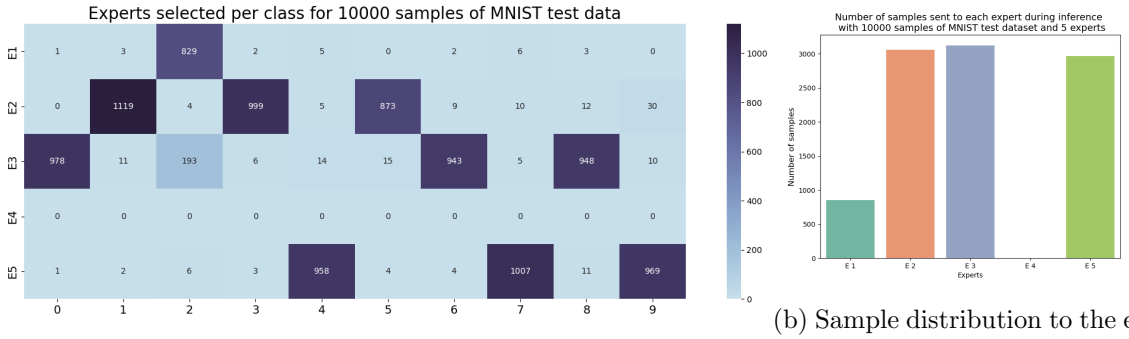
Figure B.12: Gate expert selection table of the distilled model trained with 5 experts using *output mixture* model on MNIST test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.13: Gate expert selection table of the distilled model trained with 5 experts using top-1 model on MNIST test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.14: Gate expert selection table of the distilled model trained with 5 experts using top-2 model on MNIST test dataset.

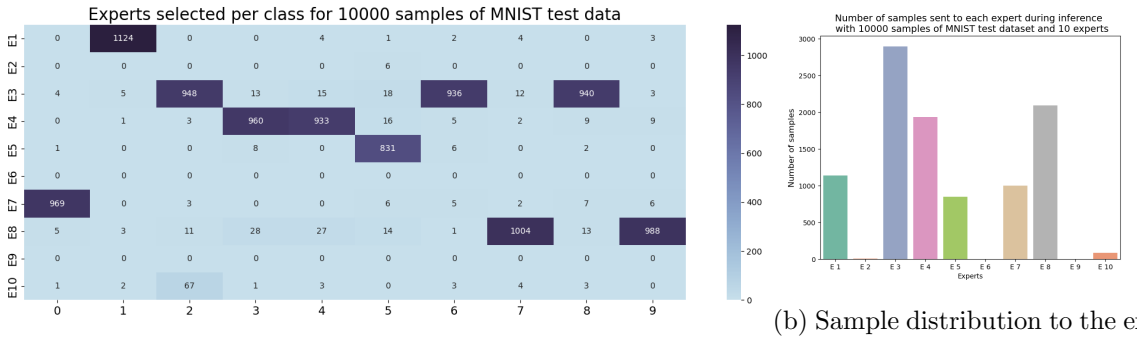
B.2.3 MNIST inference with attentive gate model with 10 experts

Table B.5 shows that the attentive gating model also performs better than the benchmark MoE models for MNIST dataset with 10 experts. It also shows that the distilled models perform as well as the attentive gate models they are distilled from, with better expert usage, with 10 experts.

Table B.5: Performance of the model with minimum validation error, for attentive gate and distilled attentive gate models, on the test set for MNIST dataset with 10 experts. Best result for benchmark models, attentive gate models and distilled attentive gate models are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
original MoE	95.38 ± 0.021	1.488	0.031	1.490
stochastic MoE	96.20 ± 0.028	1.405	0	1.516
top-1 MoE	94.83 ± 0.335	1.357	0	1.357
top-2 MoE	**96.25** ± 0.007	1.663	0.074	1.912
attentive output mixture gate MoE	96.29 ± 0.013	2.456	0.006	2.504
attentive stochastic gate MoE	96.68 ± 0.006	2.436	0	2.669
top-1 with attentive gate MoE	95.55 ± 0.048	2.058	0	2.076
top-2 with attentive gate MoE	**96.91 ± 0.005**	2.131	0.028	2.399
distilled MoE with output mixture MoE	96.57 ± 0.013	2.447	0.009	2.447
distilled MoE with stochastic MoE	96.70 ± 0.008	2.415	0	2.674
distilled MoE with top-1 MoE	95.83 ± 0.270	2.050	0	2.084
distilled MoE with top-2 MoE	**96.91 ± 0.078**	2.553	0.034	2.736

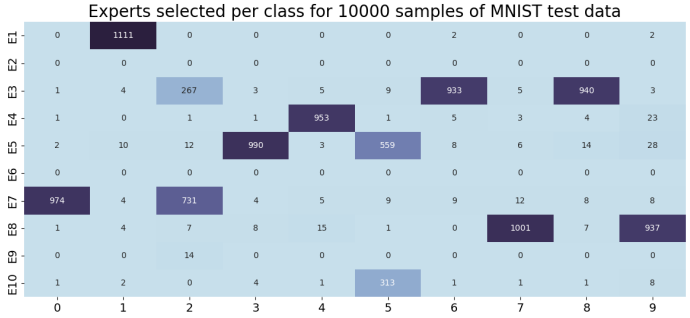
Figures B.15, B.17 and B.18 show the experts used during inference, on the test set, using attentive gate model trained with *output mixture*, *top-1* and *top-2* methods for MNIST dataset with 10 experts. We see that not all experts are used but the decompositions are clean for all methods of training. This shows that we do not have to use all the experts equitably for better performance.



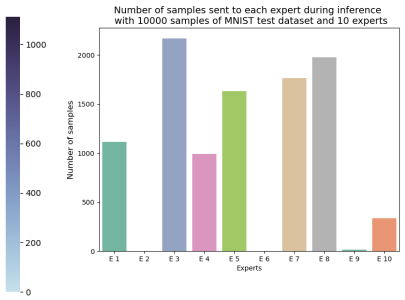
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.15: Gate expert selection table of the attentive gate model trained with 10 experts using *output mixture* model on MNIST test dataset.

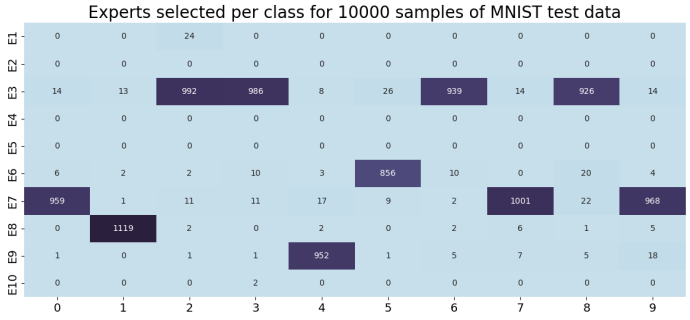


(a) Gate expert selection table

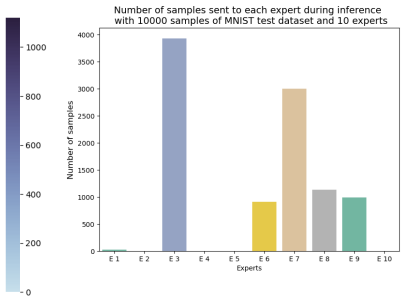


(b) Sample distribution to the experts by the gate

Figure B.16: Gate expert selection table of the attentive gate model trained with 10 experts using *stochastic* model on MNIST test dataset.

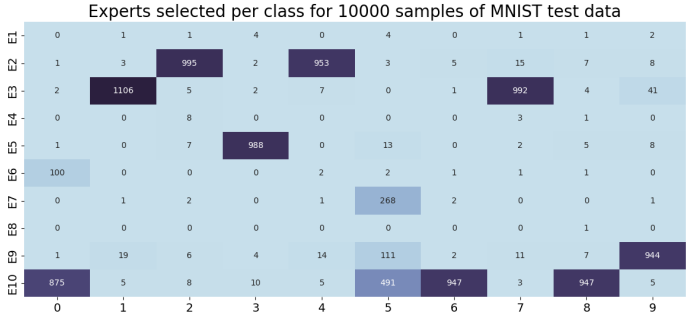


(a) Gate expert selection table

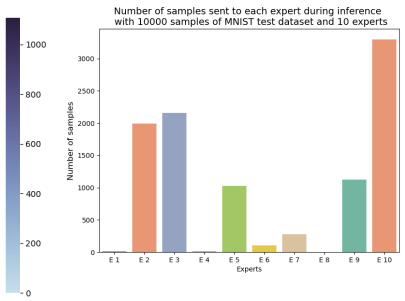


(b) Sample distribution to the experts by the gate

Figure B.17: Gate expert selection table of the attentive gate model trained with 10 experts using *top-1* model on MNIST test dataset.



(a) Gate expert selection table

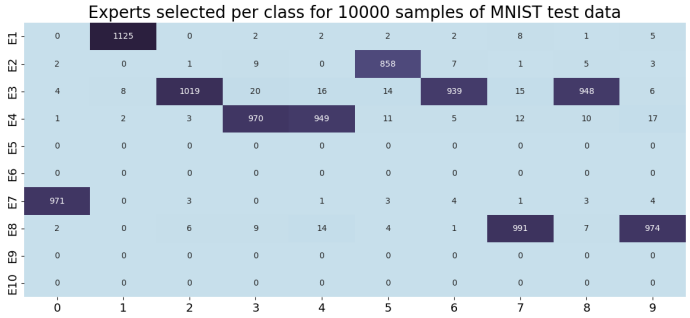


(b) Sample distribution to the experts by the gate

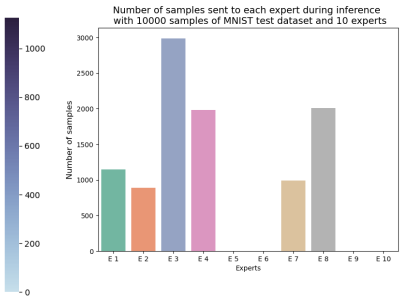
Figure B.18: Gate expert selection table of the attentive gate model trained with 10 experts using *top-2* model on MNIST test dataset.

B.2.4 MNIST inference with distilled model with 10 experts

Figures B.19, B.21 and B.22 show the experts used during inference, on the test set, using distilled model trained with *output mixture*, *top-1* and *top-2* methods for MNIST dataset with 10 experts. We see that there is good expert usage and clean decompositions for all methods of training. The performance of the distilled models is comparable to the corresponding attentive models they were distilled from.

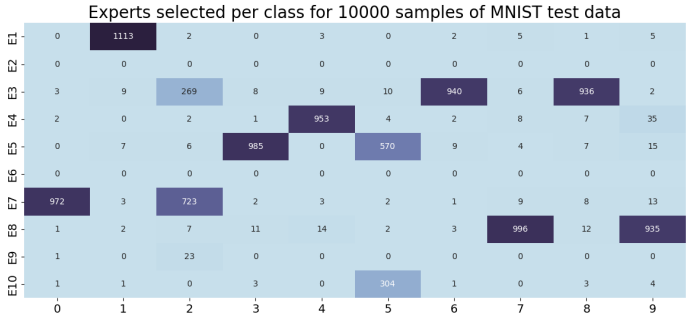


(a) Gate expert selection table

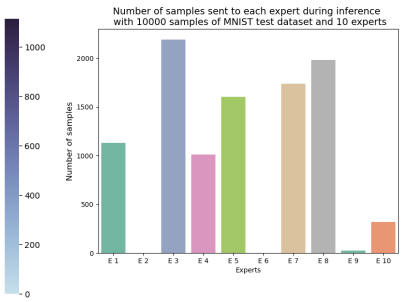


(b) Sample distribution to the experts by the gate

Figure B.19: Gate expert selection table of the distilled model trained with 10 experts using *output mixture* model on MNIST test dataset.

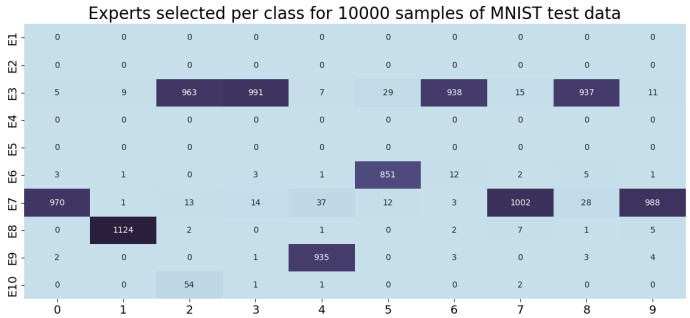


(a) Gate expert selection table

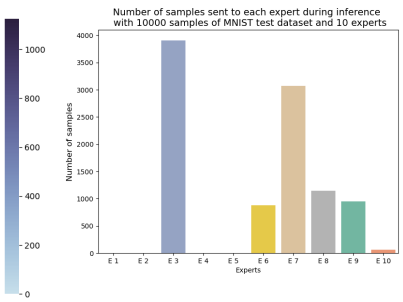


(b) Sample distribution to the experts by the gate

Figure B.20: Gate expert selection table of the distilled model trained with 10 experts using *stochastic* model on MNIST test dataset.

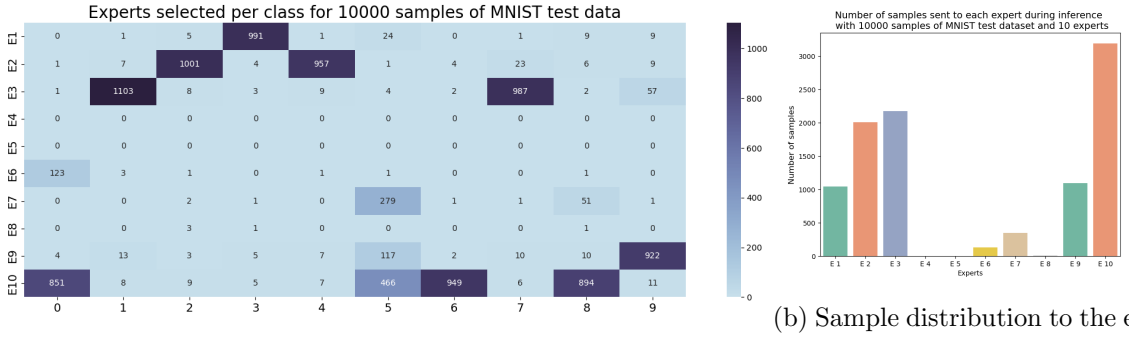


(a) Gate expert selection table



(b) Sample distribution to the experts by the gate

Figure B.21: Gate expert selection table of the distilled model trained with 10 experts using *top-1* model on MNIST test dataset.



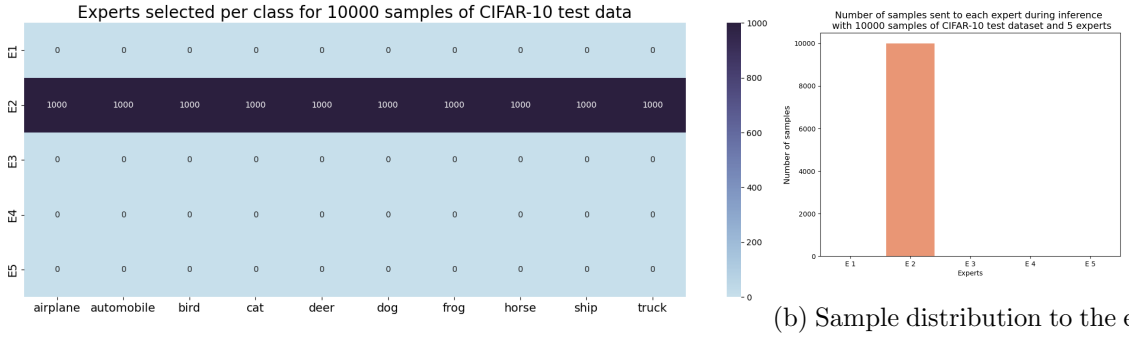
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.22: Gate expert selection table of the distilled model trained with 10 experts using *top-2* model on MNIST test dataset.

B.2.5 CIFAR-10 inference with attentive gate model with 5 experts

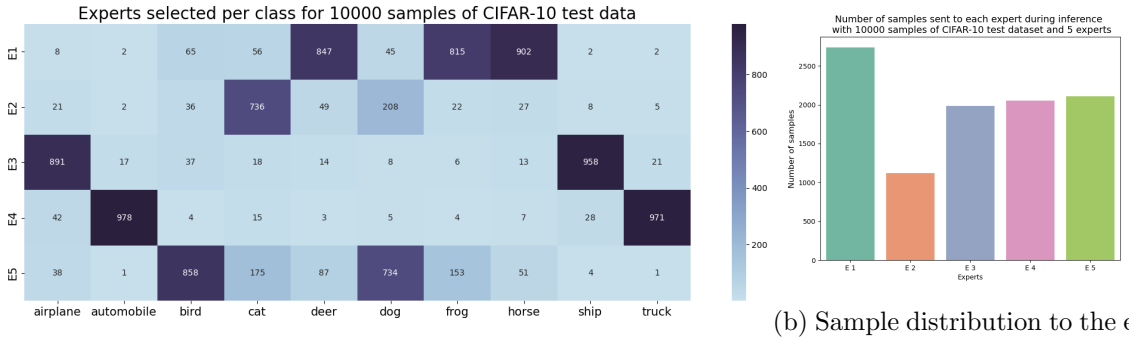
Figures B.23, B.24 and B.25 show the experts used during inference, on the test set, using attentive gate model trained with *output mixture*, *stochastic* and *top-1* methods for CIFAR-10 dataset with 10 experts. We see that there is equitable and clean task decomposition when trained with *stochastic* methods. With *output mixture* and *top-1* methods we see *module collapse*. Hence, for CIFAR-10 dataset not all methods of training are suitable.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

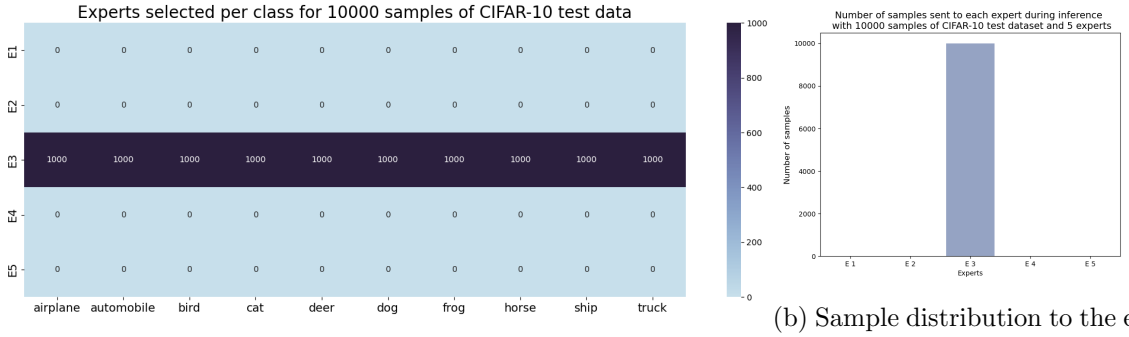
Figure B.23: Gate expert selection table of the attentive gate model trained with 5 experts using *output mixture* model on CIFAR-10 test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.24: Gate expert selection table of the attentive gate model trained with 5 experts using *stochastic* model on CIFAR-10 test dataset.



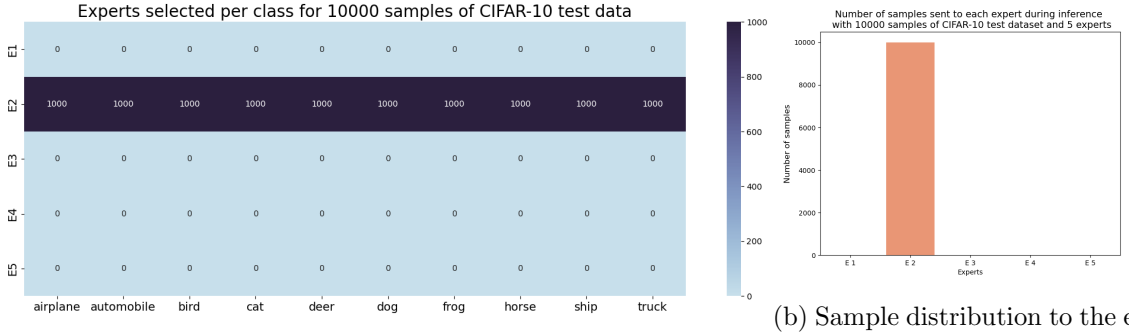
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.25: Gate expert selection table of the attentive gate model trained with 5 experts using *top-1* model on CIFAR-10 test dataset.

B.2.6 CIFAR-10 inference with distilled model with 5 experts

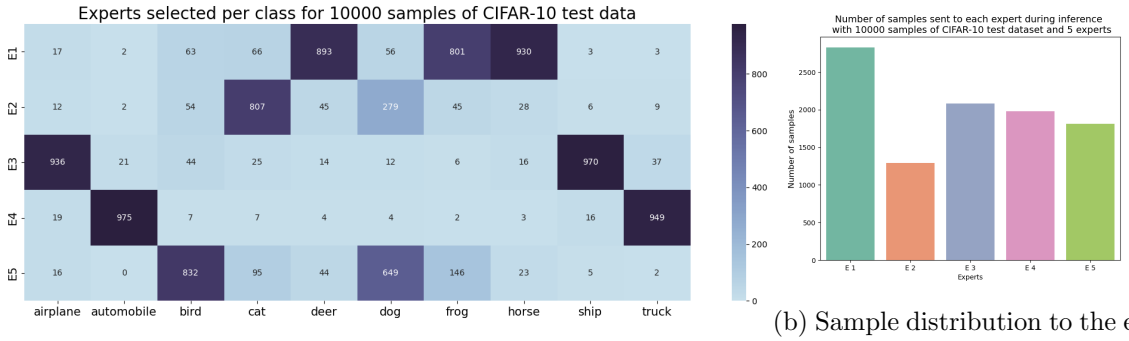
Figures B.26, B.27 and B.28 show the experts used during inference, on the test set, using the distilled gate model trained from the attentive gate model with *output mixture*, *stochastic* and *top-1* methods for CIFAR-10 dataset with 5 experts. We see that the distilled model also results in equitable and clean task decompositions without regularizations for CIFAR-10 dataset for *stochastic* method. With *output mixture* and *top-1* methods we see *module collapse*. Hence, for CIFAR-10 dataset not all methods of training are suitable.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

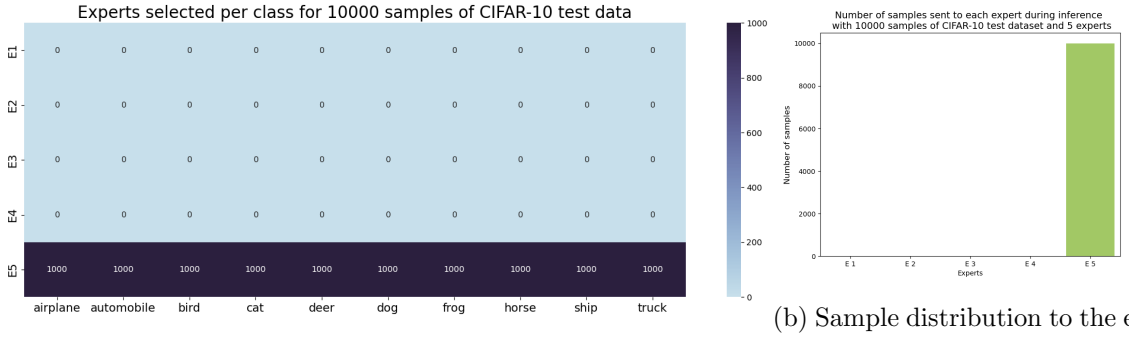
Figure B.26: Gate expert selection table of the distilled model trained with 5 experts using *output mixture* model on CIFAR-10 test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.27: Gate expert selection table of the distilled model trained with 5 experts using *stochastic* model on CIFAR-10 test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.28: Gate expert selection table of the distilled model trained with 5 experts using *top-1* model on CIFAR-10 test dataset.

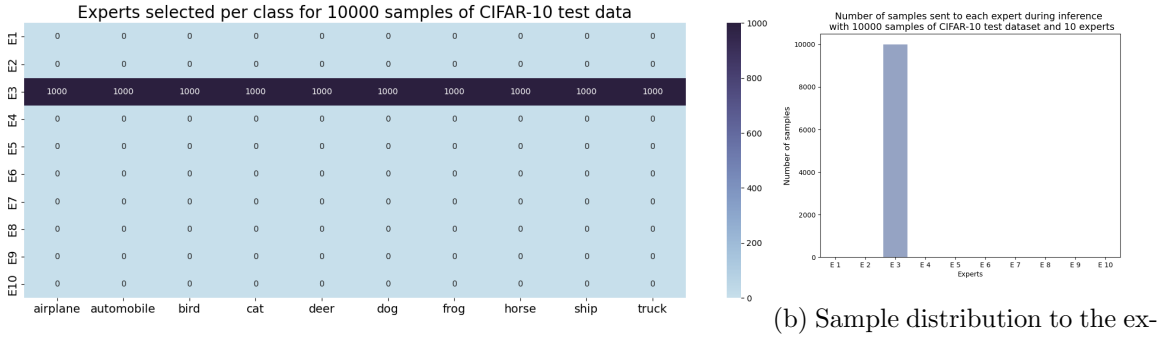
B.2.7 CIFAR-10 inference with attentive gate model with 10 experts

Table B.6 shows that the attentive gating model performs better than the benchmark MoE models for CIFAR-10 datasets with 10 experts. They also show that the distilled models perform as well as the attentive gate models they are distilled from with better expert usage.

Table B.6: Performance of the model with minimum validation error, for attentive gate and distilled attentive gate models, on the test set for CIFAR-10 dataset with 10 experts. Best result for benchmark models, attentive gate models and distilled attentive gate models are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	64.944 ± 0.008	NA	NA	NA
original MoE	71.70 ± 0.018	1.839	0.161	1.867
stochastic MoE	72.23 ± 0.013	1.367	0	1.395
top-1 MoE	66.45 ± 0.034	0.969	0	0.991
top-2 MoE	**77.60 ± 0.021**	1.710	0.146	1.731
attentive output mixture gate MoE	64.85 ± 0.105	0	0	0
attentive stochastic gate MoE	85.91 ± 0.061	3.180	0	3.310
top-1 with attentive gate MoE	64.32 ± 0.076	0	0	0
top-2 with attentive gate MoE	**86.57 ± 0.006**	2.944	0.171	3.247
distilled MoE with output mixture MoE	63.74 ± 0.212	0	0	0
distilled MoE with stochastic MoE	**87.44 ± 0.066**	3.262	0	3.321
distilled MoE with top-1 MoE	64.26 ± 0.259	0	0	0
distilled MoE with top-2 MoE	82.62 ± 0.078	2.704	0.133	3.136

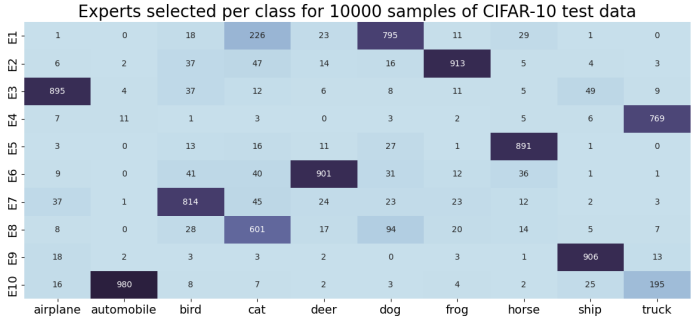
Figures B.29, B.30 and B.31 show the experts used during inference, on the test set, using attentive gate model trained with *output mixture*, *stochastic* and *top-1* methods for CIFAR-10 dataset with 10 experts. We see that there is equitable and clean task decomposition when trained with *stochastic* and *top-2* methods. With *output mixture* and *top-1* methods we see *module collapse*. Hence, for CIFAR-10 dataset not all methods of training are suitable.



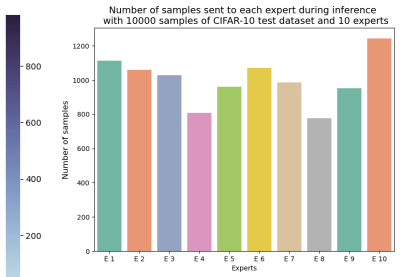
(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.29: Gate expert selection table of the attentive gate model trained with 10 experts using *output mixture* model on CIFAR-10 test dataset.

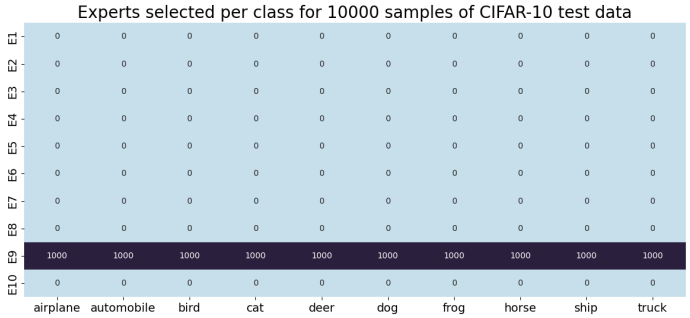


(a) Gate expert selection table

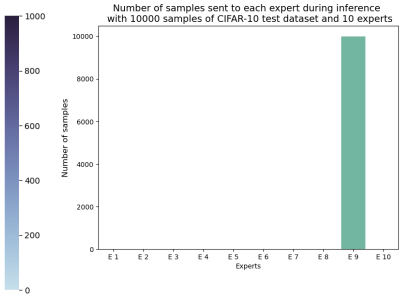


(b) Sample distribution to the experts by the gate

Figure B.30: Gate expert selection table of the attentive gate model trained with 10 experts using *stochastic* model on CIFAR-10 test dataset.



(a) Gate expert selection table

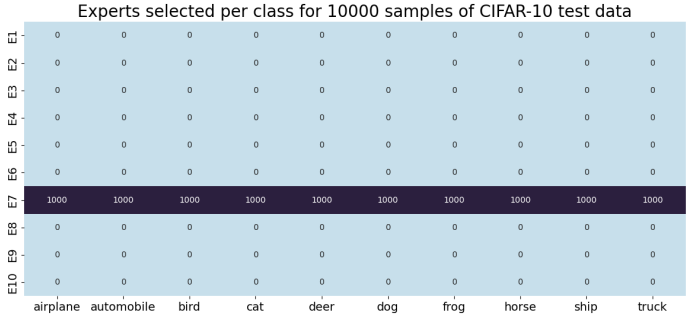


(b) Sample distribution to the experts by the gate

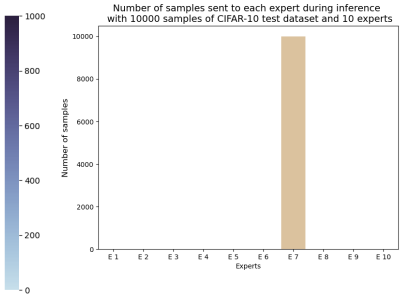
Figure B.31: Gate expert selection table of the attentive gate model trained with 10 experts using *top-1* model on CIFAR-10 test dataset.

B.2.8 CIFAR-10 inference with distilled model with 10 experts

Figures B.32, B.33 and B.34 show the experts used during inference, on the test set, using the distilled gate model trained from the attentive gate model with *output mixture*, *stochastic* and *top-1* methods for CIFAR-10 dataset with 10 experts. We see that the distilled model also results in equitable and clean task decompositions without regularizations for CIFAR-10 dataset for *stochastic* and *top-2* methods. With *output mixture* and *top-1* methods we see *module collapse*. Hence, for CIFAR-10 dataset not all methods of training are suitable.

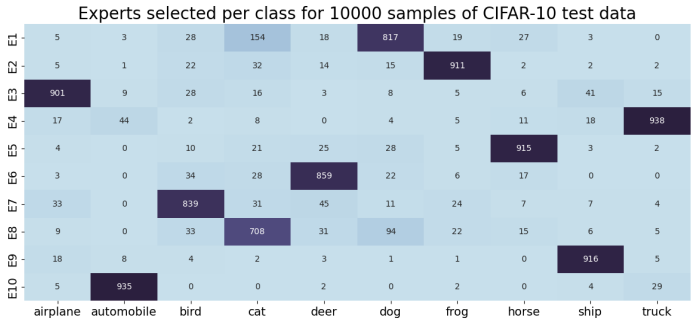


(a) Gate expert selection table

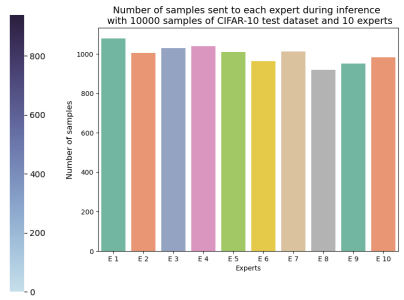


(b) Sample distribution to the experts by the gate

Figure B.32: Gate expert selection table of the distilled model trained with 10 experts using *output mixture* model on CIFAR-10 test dataset.

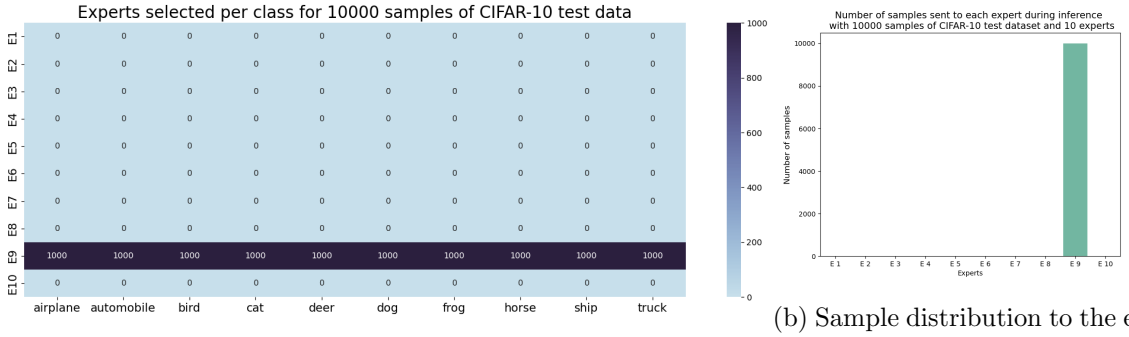


(a) Gate expert selection table



(b) Sample distribution to the experts by the gate

Figure B.33: Gate expert selection table of the distilled model trained with 10 experts using *stochastic* model on CIFAR-10 test dataset.



(a) Gate expert selection table

(b) Sample distribution to the experts by the gate

Figure B.34: Gate expert selection table of the distilled model trained with 10 experts using *top-1* model on CIFAR-10 test dataset.

B.3 Results for Expert Loss Gate Experiments

B.3.1 MNIST inference with *expert loss gate* MoE with 10 experts

Table B.7: Performance on the test set by the *expert loss gate* model with the minimum validation error for MNIST dataset with 10 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	92.43 ± 0.031	NA	NA	NA
original MoE	95.38 ± 0.021	1.488	0.031	1.490
stochastic MoE	96.20 ± 0.028	1.405	0	1.516
top-1 MoE	94.83 ± 0.335	1.357	0	1.357
top-2 MoE	**96.25** ± 0.007	1.663	0.074	1.912
expert loss gate MoE	94.41 ± 0.020	1.556	0	2.462

B.3.2 CIFAR-10 inference with *expert loss gate* MoE with 10 experts

Table B.8: Performance on the test set by the *expert loss gate* model with the minimum validation error for CIFAR-10 dataset with 10 experts. Best result is highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
single model	42.74 ± 0.011	NA	NA	NA
original MoE	71.85 ± 0.019	1.839	0.161	1.867
stochastic MoE	72.82 ± 0.061	1.46	0	1.505
top-1 MoE	67.18 ± 0.036	0.969	0	0.969
top-2 MoE	77.02 ± 0.017	1.710	0.148	1.731
expert loss gate MoE	**78.4 ± 0.050**	1.925	0	2.172

B.4 Results for Sample Similarity Experiments

B.4.1 MNIST inference with sample similarity regularization with 10 experts

Table B.9: Performance with $L_{importance}$ and L_s regularizations of models with minimum validation error on the test set for MNIST dataset with 10 experts. Best performance with $L_{importance}$ and L_s are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
output mixture MoE with $L_{importance}$	97.72 ± 0.010	3.319	0.053	3.319
top-2 MoE with $\mathbf{L_{importance}}$	**97.83 ± 0.005**	3.283	0.054	3.320
output mixture MoE with L_s	96.65 ± 0.023	2.384	0.060	2.476
top-2 MoE with L_s	97.20 ± 0.007	2.233	0.039	2.248
output mixture attentive gate MoE with $L_{importance}$	97.49 ± 0.005	3.320	0.007	3.320
top-2 attentive gate MoE with $L_{importance}$	97.59 ± 0.005	3.258	0.017	3.320
output mixture attentive gate MoE with L_s	97.00 ± 0.010	2.948	0.010	3.102
top-2 with attentive gate MoE with $\mathbf{L_s}$	**97.24 ± 0.005**	2.632	0.045	2.921
distilled MoE with output mixture MoE with $L_{importance}$	97.79 ± 0.044	3.318	0.008	3.319
distilled MoE with top-2 MoE with $L_{importance}$	97.04 ± 0.068	2.897	0.038	3.147
distilled MoE with output mixture MoE with L_s	97.08 ± 0.042	2.928	0.015	2.971
distilled MoE with top-2 MoE with L_s	96.91 ± 0.078	2.553	0.034	2.736

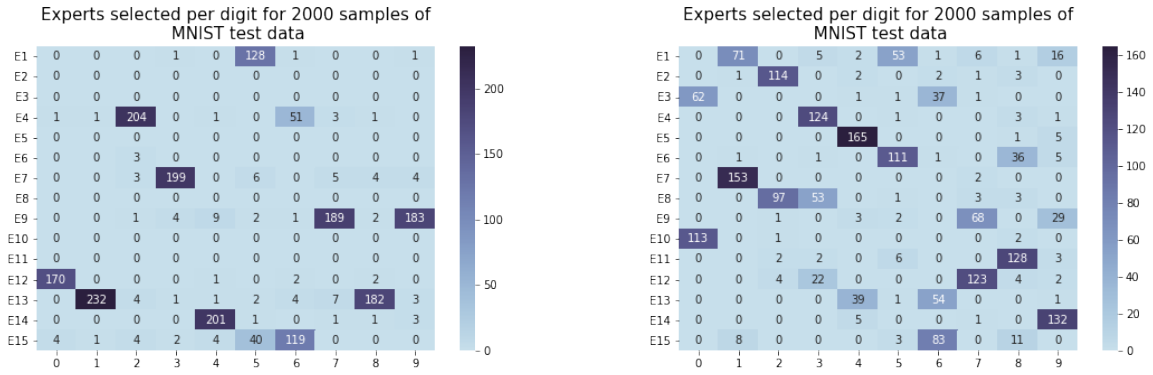
B.4.2 CIFAR-10 inference on test data with sample similarity regularization with 10 experts

Table B.10: Performance with $L_{importance}$ and L_s regularizations of models with minimum validation error on the test set for CIFAR-10 dataset with 10 experts. Best performance with $L_{importance}$ and L_s are highlighted.

Experiment	**Test Accuracy**	$\mathbf{I(E;Y)}$	$\mathbf{H_s}$	$\mathbf{H_u}$
output mixture MoE with $L_{importance}$	81.49 ± 0.026	3.304	0.262	3.304
top-2 MoE with $L_{importance}$	81.73 ± 0.132	3.311	0.376	3.310
output mixture MoE with L_s	82.85 ± 0.053	3.309	0.166	3.310
top-2 MoE with $\mathbf{L_s}$	83.50 ± 0.036	3.114	0.295	3.254
output mixture with attentive gate MoE with $L_{importance}$	80.51 ± 0.032	3.319	0.031	3.319
top-2 with attentive gate MoE with $L_{importance}$	86.50 ± 0.009	3.229	0.148	3.317
output mixture with attentive gate MoE with L_s	74.73 ± 0.136	3.099	0.017	3.319
top-2 with attentive gate MoE with $\mathbf{L_s}$	**87.46 ± 0.007**	3.077	0.155	3.310
distilled MoE with output mixture MoE with $L_{importance}$	85.77 ± 0.006	3.317	0.034	3.317
distilled MoE with top-2 MoE with $\mathbf{L_{importance}}$	**87.20 ± 0.074**	3.282	0.099	3.318
distilled MoE with output mixture MoE with L_s	84.51 ± 0.270	3.306	0.047	3.314
distilled MoE with top-2 MoE with $\mathbf{L_s}$	87.17 ± 0.082	3.266	0.116	3.319

B.4.3 Expert usage by $L_{importance}$ and L_s for MNIST dataset

In Section 9.3 we showed that when we use 10 experts for 10 classes in the MNIST dataset, the $L_{importance}$ regularization uses all the experts whereas the L_s regularization uses only 8 experts for the same performance. We also increased the number of experts to 15 which is the more number of experts than the number of classes in MNIST. Figure B.35 shows that L_s regularization results in more optimal use of experts while $L_{importance}$ uses all the experts.



(a) L_s (b) $L_{importance}$

Figure B.35: Expert selection table of MoE model trained with L_s and $L_{importance}$ regularizations with 15 experts.