RE-EXAMINING ROUTING NETWORKS FOR MULTI-TASK LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We re-examine Routing Networks, an approach to multi-task learning that uses reinforcement learning to decide parameter sharing with the goal of maximizing knowledge transfer between related tasks while avoiding task interference. These benefits come with the cost of solving a more difficult optimization problem. We argue that the success of this model depends on a few key assumptions and, when they are not satisfied, the difficulty of learning a good route can outweigh the benefits of the approach. In these cases, a simple unlearned routing strategy, which we propose, achieves the best results.

1 INTRODUCTION

When multi-task learning (MTL) works well, the model is able to leverage commonalities between tasks to perform better than it could if it was trained on any single task. It isn't always so simple though. If the model is not set up correctly, inference between tasks or other optimization issues can cause the performance from MTL to be worse than the single task baseline (Wu et al., 2020). Ruder (2017) points out that "recent approaches [to MTL] have...looked towards *learning* what to share" in order to avoid the brittleness of early approaches to MTL like hard parameter sharing.

We focus on a particular approach to learned parameter sharing known as routing networks, first proposed by Rosenbaum et al. (2018). The routing network model consists of two components: a router and a set of one or more function blocks. A function block may be any neural network – for example a fully-connected or a convolutional layer. Given an input the router makes a routing decision, choosing a function block to apply and passing the output back to the router recursively. The role of the router is to maximize sharing between related tasks that can benefit from each other and to limit negative transfer between interfering tasks.

We hypothesize that for the benefits of routing networks to be realized, some key assumptions need to hold:

1. It must be practical to learn model weights and routing policies simultaneously.
2. The routing policy learner can handle combinatorial explosions in the number of possible route choices.
3. For real-world multi-task datasets, it matters which tasks share parameters with each other.

While the degree to which these assumptions are true likely depends on the specific data and model that one is working with, our analysis, using setups similar to previous published work, indicates that there are some significant caveats to these assumptions that can limit the practical effectiveness of routing networks. For example, when the number of tasks is large ($n \geq 10$), the network is deep (three or more routing layers), and the tasks are not too dissimilar (interference is low) the model with learned routing can struggle to beat the baselines. This insight led us to propose a new baseline for learned parameter sharing where the policy learner is replaced by an untrained model that fixes the route randomly at model initialization. Our experiments show that fixed random routing is competitive with or even exceeds the performance of routing networks on multiple datasets with a large number of tasks.

2 Modeling Approach

2.1 Standard MTL Baselines

We first introduce the shared bottom model and no sharing multi-task model as standard baselines used in MTL. These baselines represent the two extremes where either there is the maximum amount of parameter sharing (shared bottom) or nothing at all is shared between tasks (no sharing). All of the other MTL methods try to achieve better results by using an intermediate level of information sharing.

The shared bottom framework (Baxter, 1997) has been widely adopted in multi-task learning applications (Salakhutdinov et al., 2011; Li et al., 2016). Assume we have a multi-task learning dataset $D = \{\mathbf{X}^k, \mathbf{Y}^k\}_{k=1}^K$ in which $\mathbf{X}^k$ denotes the input features and $\mathbf{Y}^k$ denotes the ground truth label for k-th task. Given K tasks, the goal is to learn a low dimensional representation network g and K tower networks f^k $f^k \circ g : \mathcal{X}^k \mapsto \mathcal{Y}^k$ that minimizes the expected loss. As for the no sharing model, the tasks are completely independent. Each task has its individual network and the parameters are not shared.

2.2 Task-specific Policy Learning

We follow the framework of Rosenbaum et al. (2018) opting to use their published code with a few enhancements to be described later. The authors employ the WPL algorithm (Abdallah & Lesser, 2006) to jointly train the router and modules. In the simplest version of routing networks, the router's decision is based purely on the task id. The routing policy is learned using reinforcement learning whereas the function block or module weights are learned using standard gradient descent methods.

The routing network consists of a router and a set of modules. A module can be any neural network layer such as a fully connected or a convolutional layer. The input to the network is an instance x^k to be classified, where $x \in \mathbb{R}^d$ is a d dimension vector and k represents the k-th task. The route selects from a set of modules, choose a module to apply and get the output. The routing decision function $router$ maps the current output and depth to the next module index to visit. Such routing decision is made recursively until a certain depth is reached. The output from the max depth is the prediction $\hat{y}^t$. For classification problems, we set the reward R as 1 if the network prediction is correct, and -1 otherwise. The trace, including the sequence of visited states, the sequence of the actions taken and the final reward, is stored to trained the router. The WPL algorithm is applied to train the router.

The module weights along the selected route and router are trained alternatively until the max epoch is reached. The detailed training steps are summarized in Algorithm 1.

Algorithm 1: Routing Network for Multi-task Learning

```
Input: Training set, training epoch $E$, max depth $d$
Output: Well-trained modules and router
1  for $e = 1, \dots, E$ do
2      Set state $s_0^k = x^k$;
3      for $i = 1, \dots, d$ do
4          Select action $a_i = \arg\max_a router(s_{i-1}^k, i, a)$;
5          $s_i^k = f_{a_i}(s_{i-1}^k)$;
6      end
7      Set the network prediction as the last output $\hat{y}^k = s_d^k$;
8      Store a trace $T = (S, A, R, r_f)$, where $S$ is the sequence of visited states $\{s_i^k\}_{i=1}^d$, $A$ is the
         sequence of actions taken $\{a_i\}_{i=1}^d$, $R$ is the sequence of immediate action rewards $\{r_i\}_{i=1}^d$
         and $r_f$ is the final reward;
9      Minimize the loss between the network prediction $\hat{y}^k$ and the ground truth $y^k$ to update the
         module parameters along the selected route;
10     Use the trace $T$ to train the router via the WPL algorithm;
11 end
```

2.3 Fixed Random Routing

We want to see how much of the benefit of routing networks comes from learning which tasks ought to share parameters with each other versus simply having some partial amount of sharing. To test this, we propose a fixed random selection as an alternative to learned parameter sharing. Fixed random selection of the routing policy bypasses the optimization difficulties of simultaneously learning the weights and routes and the instability of the routing reward.

Suppose we have a modular network, with d layers having m^i modules in each layer, where $i = 1, \ldots d$. Each module can either be a convolutional layer or a fully connected layer. K routes $\{R^k\}_{k=1}^K$ are initialized randomly, each route R^k is an integer vector of d dimensions, which describes the module visited in each layer along the route. After a random route for each task is chosen, given an input, the router selects a module to apply and passes it down to the next layer along the route recursively, until the fixed recursion depth is reached. The module weights on the route are optimized during the training. When $m^i \gg K$, tasks, the possibility of tasks selecting the same module is small; otherwise, tasks are more likely to share modules.

3 Experiments

3.1 Datasets

We use two datasets for our experiments: CIFAR-100 (Krizhevsky et al., 2009), Fashion-MNIST (Xiao et al., 2017).

CIFAR-100 has 100 classes containing 600 images each. Each class has 500 training instances and 100 testing instances. The 100 classes are grouped into 20 superclasses (Krizhevsky et al., 2009). The classes include objects like apples and pears. The corresponding superclass for these objects is fruit & vegetables. The CIFAR-100 becomes a multitask dataset by grouping the 100 classes into 20 tasks where each task has to assign the true class to the label out of the five possibilities. Rosenbaum et al. (2018) grouped the classes based on their alphabetical order ignoring the relationship between classes and superclasses. We opt to define the tasks based on the 20 superclasses because it is more challenging for the model to discriminate between objects in the same superclass. The accuracy of the no sharing baseline trained with the alphabetical grouping is 73.5% compared to 82.3% for the superclass grouping.

Fashion-MNIST contains 10 classes, where each class has a training set of 6,000 examples and a test set of 1,000 examples. Each instance is a 28×28 grayscale image. To adapt it into a multi-task learning dataset, we create 10 tasks and for each we use 6,000 instances of one class as the positive class and 600 each of the remaining 9 classes as the negative class during training, then we test on 1,000 instances of the class and 100 each of the 9 classes.

3.2 Experimental Setup

We use a network consisting of 6 components: 3 convolutional (Conv) layers and 3 fully connected (FC) layers. Each convolutional layer has 3×3 convolution and 32 filters and is followed by Batch Normalization (Ioffe & Szegedy, 2015) and a ReLU layer with 64 hidden units. The final layer is unshared for each task. Following Rosenbaum et al. (2018), we use the WPL algorithm (Abdallah & Lesser, 2006) for routing, and experiment with applying the routing to each Conv or FC layer. We also experiment routing sets of adjacent layers in the network.

We use separate optimizers for the weight learner and router. We both test on the setting in the original paper (Rosenbaum et al., 2018) and conduct a parameter sweep to find the best learning rates for weight learner and router respectively. We try both SGD and Adam (Kingma & Ba, 2018) and choose Adam for better performance. Similar to Maziarz et al. (2020), we find that it is better to have a higher learning rate on the route optimizer compared to the weight learner. The learning rate of weight learner is 1.5^{-4} and the learning rate of router is 2.0^{-2}. We use dropout in all layers and test the fraction from 0.0 to 0.9. We choose to use no dropout. The training epoch is 200 for CIFAR-100 and 50 for Fashion-MNIST. We use data augmentation for all the methods including three transformations: random crop, rotation and random horizontal flip.

Table 1: Accuracy (%) on CIFAR-100 Dataset. Mean accuracy is taken over five independent experiments.

Method	Mean	Best
Routing Network (Public code)	60.5±0.75	61.1
Routing Network (Corrected)	38.8±1.87	40.9
Separate Optimizers	71.4±0.37	71.9
Routing in Conv 1	71.9±0.19	72.2
Routing in Conv 1, 2, 3	69.0±0.41	69.2
Shared bottom	72.7±0.26	72.9
No sharing	73.5±0.30	73.8
Fixed random selection in FC layers	72.8±0.44	73.3
Fixed random selection in Conv 3	**73.8**±0.45	**74.5**

3.3 Results

Tables 1 and 2 contain the results from experiments on the CIFAR-100 and Fashion-MNIST data respectively. The first entry in each table (marked as "public code") is based on the unaltered code from Rosenbaum et al. (2018) taken from Github.[1] As explained in Appendix A.1, the public code has an unfortunate issue with parameter initialization leading to exploding gradients that render the policy learner inoperable after the first mini-batch. The performance of these models is not reflective of learned parameter sharing. The next line in each table is the result of re-running the experiments after making the minimal change to avoid the just mentioned issue with the exploding gradients, and setting the loss reduction to none. Paradoxically, the exploding gradients disabling the router policy learner at the start of training is not crippling to the model and the overall performance can still be reasonable. After that, we made additional improvements to the model including using separate optimizers for the model weights and routing policy and applying a Softmax activation function in the routing policy layer.

Why does correcting the parameter initialization result in a drop in performance for one dataset and an increase for the other? We follow the rule of thumb from Rosenbaum et al. (2018) of setting the number of function blocks to match the number of tasks in the dataset. Since, CIFAR-100 has twice as many tasks as Fashion-MNIST, it is much more challenging to learn a reasonable routing policy in the former than it is in the case of the latter.

For both datasets we see that the no sharing and shared bottom baselines are relatively close to each other in terms of accuracy. In the case of CIFAR-100, both baselines perform better than any of the methods that use learned routing. The best overall method for CIFAR-100 is using all shared parameters except in the 3rd convolutional layer. For the Fashion-MNIST dataset, using learned routing in the convolutional layers is better than the shared bottom baseline and near in performance to the no sharing baseline. The best overall performance was from using fixed random routing in the third convolutional layer similar to what was done in CIFAR-100.

4 Analysis

4.1 Simultaneous learning of routes and weights

Learning the parameters of a deep neural network can be difficult by itself and it is only made harder when a routing component is added. One challenge is the cold start problem: before the model has made progress in learning the tasks, the reward signal for the router is unstable. The issue affects other models for learned parameter sharing besides routing networks. Sun et al. (2019) need to employ some special training techniques to avoid a difficulty in "find[ing] the optimal policy in the early training stage." and Maziarz et al. (2019) note that even though a Mixture-of-Experts architecture is more powerful than a shared bottom, in the case of their dataset "optimization difficulties outweigh that benefit."

[1]See the cd4e9af commit at `https://github.com/cle-ros/RoutingNetworks`.

Table 2: Accuracy (%) on Fashion-MNIST Dataset. Mean accuracy is taken over five independent experiments.

Method	Mean	Best
Routing Network (Public code)	64.2±0.57	64.8
Routing Network (Corrected)	83.7±0.57	84.3
Separate Optimizers	95.9±0.17	96.2
Routing in Conv 1	95.9±0.19	96.1
Routing in Conv 1, 2, 3	96.1±0.09	96.2
Shared bottom	95.6±0.27	96.0
No sharing	96.2±0.20	96.4
Fixed random selection in FC layers	96.3±0.13	96.4
Fixed random selection in Conv 3	**96.4**±0.18	**96.6**

We investigated if these optimization difficulties show up when modeling the CIFAR-100 dataset. Figure 1 plots the perplexity of each of the three routing layer's policies during training in a model with ten function blocks and no collaboration reward. The perplexity for the earliest layer converges relatively rapidly but the perplexity of the second and third fully connected layers initial rise before gradually falling. Even after 120 epochs, they see no sign of converging on a single route. Our conclusion is that optimization difficulties do play a role here.

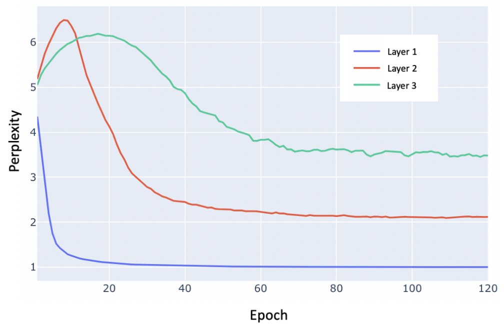


Figure 1: The perplexity of the routing policy in each layer during training.

To test if the optimization difficulties go away once the model weights have been "warmed up", we performed an additional experiment where during the first 20 epochs all parameters are shared across each task before enabling the routing policy learner after the 20th epoch. We visualize the similarity of the routing policy when using the warm up strategy during the training, which is shown in Figure 2. We can see that the policy layers of the tasks are quite different from each other. Different tasks will not voluntarily share the same module, as sharing will not help reduce the training loss. This can indicate that either no-sharing is the best policy on this dataset or else it is still hard to learn a better policy even after the weights are warmed up. Since the warm up strategy ended up learning a no sharing policy, the accuracy on CIFAR improved from 71.4% to be similar to the no sharing baseline at 73.1%.

4.2 Combinatorial explosion of routes

Some of the previous work on learned parameter sharing, e.g. Sun et al. (2019), use only a small number of tasks. When there are only two tasks there are considerably less routes to choose from and the optimization problem becomes much simpler. The model used by Rosenbaum et al. (2018) applies routing to three fully connected layers. The authors choose to set the number of function blocks equal to the number of tasks (n=20 for CIFAR-100 dataset). Three layers of routing over 20 blocks means there are $20^3 = 8000$ possible routes for each task and 160,000 possible ways of routing all 20 tasks together.

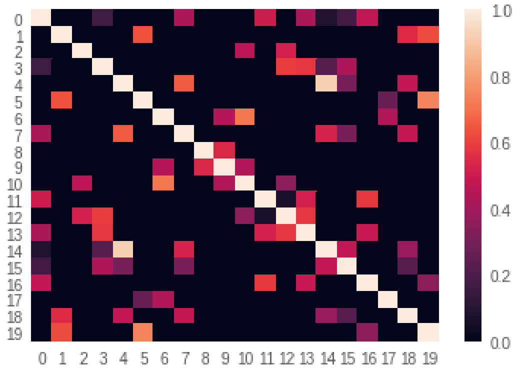


Figure 2: Pairwise cosine similarities of the policy layer for the 20 tasks of CIFAR-100 when using the "warm up" strategy. Light orange pixels denote similarity of 1 (policy layers are the same), while on the other end of the spectrum dark purple denote similarity of 0 (policy layers are orthogonal).

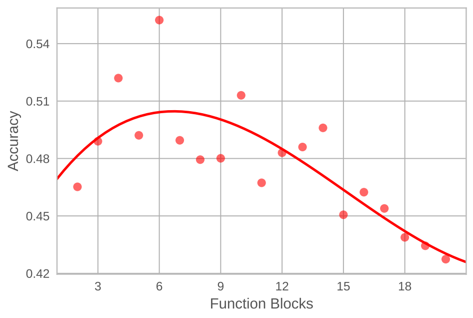


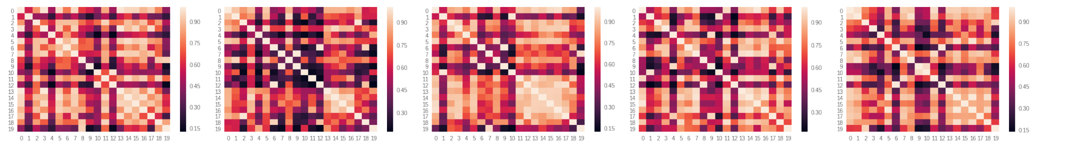
Figure 3: Relationship between the number of function blocks and model accuracy on CIFAR-100. Each point is an independent experiment run for a fixed number of epochs. The curve is a cubic polynomial best fit.

In Figure 3, we show how the accuracy of a model trained on the CIFAR-100 dataset varies according to the number of function blocks in the fully connected layers. The best performance was achieved when there were only six function blocks. Using six blocks means that there are only 3% of the number of routes to choose from compared to when there were twenty blocks.

4.3 SIMILAR TASKS GROUPED TOGETHER

We ran two sets of experiments using the CIFAR-100 data where five models were trained in each set. The first set used 6 function blocks per layer and the second set had 10. Each experiment used a unique random seed but otherwise the experiments within each set are identical. If the role of the policy learner is to discover which tasks ought to share parameters with each other and which ought to be separated than we would expect that when the model is retrained it will make similar choices about what tasks to group together. Figure 4 visualizes the similarity of the routing policy for the first fully connected layer for the models using six function blocks (a-e) and the ones using ten (f-j). We observe there is some amount of similarity but also that the routes are highly variable.

Perhaps, the tasks in CIFAR-100 are already too similar to each other that task interference is not an issue and the particular grouping of tasks is not that important. Other datasets might have more diverse collections of tasks and in those cases it might be imperative to allow the model freedom to choose how to group tasks. See, for example, Figure 6 in Maziarz et al. (2020), where the authors have combined Fashion-MNIST, CIFAR-100, and MNIST into a single dataset of 40 tasks.


(a) (b) (c) (d) (e)

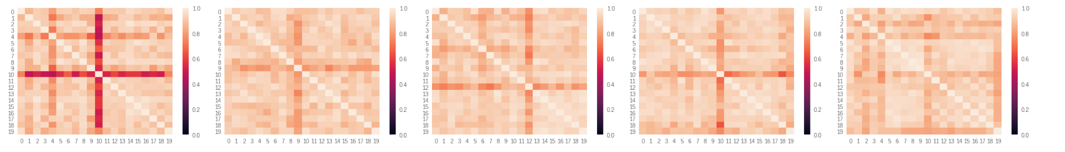

(f) (g) (h) (i) (j)

Figure 4: Cosine similarities of the policy layers when number of modules is 6 (above) and 10 (below)

5 Related Work

Focusing on a single task model may ignore commonalities and differences across related tasks. Sharing the parameters between different tasks can improve the model generalization and performance for each task (Baxter, 2000). One of the most widely used model is a shared bottom model (Baxter, 1997), where the bottom layers are shared in all tasks, and each task has its task-specific layer at top. In this case, the probability of overfitting for shared parameters is relatively low, compared to non-shared parameters.

When the correlation between tasks is weak, task interference often reduces the benefits of sharing. Hence, researchers consider modeling the relationship between multiple tasks (Bakker & Heskes, 2003; Duong et al., 2015; Søgaard & Goldberg, 2016; Yang & Hospedales, 2017; Long et al., 2017; Kendall et al., 2018). Duong et al. (2015) added L-2 constraints between two networks. Long et al. (2017) proposed Multilinear Relationship Networks (MRN) to discover task relationship based on tensor normal prior. It tackles multi-task deep learning by jointly learning transferable features and relationships of tasks. Yang & Hospedales (2017) applied tensor factorisation to divide each set of parameters into shared and task-specific modules. The approaches outperform shared bottom methods when task differences reduces the benefits of sharing. However, these methods specify which layer should be shared, resulting in poor scalability in not highly correlated tasks.

In order to decide which layers should be shared and control the amount of sharing, Cross-stitch Networks (Misra et al., 2016) and Leaky Multi-Task CNN (Xiao et al., 2018) use parameters to control the flow between layers of different networks and learn a unique combination of hidden layers for each task. Based on previous work, Sluice (Ruder et al., 2019) enables sharing of all combinations of layers, subspaces, and skip connections. However, as task-specific layers require large amount of training data, these methods may not be efficient in large scale models.

In recent work, researchers propose to use automatic learning and ensemble methods to learn task-specific models for multi-task learning (Shazeer et al., 2017; Fernando et al., 2017; Pham et al., 2018; Wang et al., 2018; Ahn et al., 2019). Routing networks, which learn a composition from a set of modules in a large network, have drawn much attentions (Veit et al., 2016; Sabour et al., 2017; Rosenbaum et al., 2018; Ramachandran & Le, 2018). Reinforcement learning based methods use a router to make routing decisions, choosing a module to apply based on reward functions. Rosenbaum et al. (2018) employ a collaborative multi-agent reinforcement learning (MARL) approach to jointly train the router and modules. Maziarz et al. (2019) use Gumbel-Matrix to select multiple modules at the same time. In Adashare, Sun et al. (2019) use Gumbel-Softmax Sampling to decide whether to skip or select a module in different tasks. Some work are built upon the field of the Mixture of Experts (MoE) (Jacobs et al., 1991; Rasmussen & Ghahramani, 2002). The MoE layer (Eigen et al., 2013; Shazeer et al., 2017) is proposed to choose k modules in a way that enables gradients to flow to the router. Ramachandran & Le (2018) and Ma et al. (2018) use MoE layers to select modules based on the input of the layer and output the next module to use. Collier et al. (2020) addresses poorly initialized experts when new tasks are presented in continual Learning. We examine these work in this paper and propose a random selection policy to achieve better accuracy while saving computational costs.

6 Conclusion and Future Work

We have examined the performance of routing networks and identified a few challenges. First, reinforcement learning based routing networks suffer from the cold start problem. Before weights have mostly converged, the routing reward signal can be very unstable. Second, combinatorial explosion of routing choices make it difficult for optimization. Third, in datasets like those used in prior work, the tasks are similar enough to each other that the model does not display a strong preference for grouping them in one way or another. So, there is no great harm in using an unoptimized route. Based on our analysis, we proposed fixed random selection as an alternative to learned parameter sharing to obtain the benefits of partial parameter sharing while bypassing some of the difficulties of learning the routes. Experiments on two datasets demonstrate the strong performance of the random selection. We hope that future work on learned parameter sharing can consider fixed random routes as a new baseline in addition to the common baselines of shared bottom and no sharing.

There are several interesting future directions for investigation. First, we are interested to see if improvements to the learner for the router can make a difference on these datasets. There are many unexplored techniques that can be attempted to alleviate the combinatorial explosion of routing choices such as weight tying between routing layers for the same task. Second, the datasets that we used in our experiments originated as multiclass classification datasets and were transformed into MTL datasets in previous work. Additional experiments can be done on datasets where the tasks are more diverse. Perhaps, in that setting, the routing decisions will be more consequential and the learned routing strategies will dominate the fixed random selection.

Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

References

Sherief Abdallah and Victor Lesser. Learning the task allocation game. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 850–857, 2006.

Chanho Ahn, Eunwoo Kim, and Songhwai Oh. Deep elastic networks with model selection for multi-task learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6529–6538, 2019.

Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4(May):83–99, 2003.

Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39, 1997.

Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12: 149–198, 2000.

Mark Collier, Efi Kokiopoulou, Andrea Gesmundo, and Jesse Berent. Routing networks with co-training for continual learning. *arXiv preprint arXiv:2009.04381*, 2020.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 845–850, 2015.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Xi Li, Liming Zhao, Lina Wei, Ming-Hsuan Yang, Fei Wu, Yueting Zhuang, Haibin Ling, and Jingdong Wang. Deepsaliency: Multi-task deep neural network model for salient object detection. *IEEE transactions on image processing*, 25(8):3919–3930, 2016.

Mingsheng Long, Zhangjie Cao, Jianmin Wang, and S Yu Philip. Learning multiple tasks with multilinear relationship networks. In *Advances in neural information processing systems*, pp. 1594–1603, 2017.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1930–1939, 2018.

Krzysztof Maziarz, Efi Kokiopoulou, Andrea Gesmundo, Luciano Sbaiz, Gabor Bartok, and Jesse Berent. Gumbel-matrix routing for flexible multi-task learning. *arXiv preprint arXiv:1910.04915*, 2019.

Krzysztof Maziarz, Efi Kokiopoulou, Andrea Gesmundo, Luciano Sbaiz, Gabor Bartok, and Jesse Berent. Flexible multi-task networks by learning parameter allocation. 2020.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3994–4003, 2016.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pp. 4095–4104, 2018.

Prajit Ramachandran and Quoc V Le. Diversity and depth in per-example routing models. In *International Conference on Learning Representations*, 2018.

Carl E Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. In *Advances in neural information processing systems*, pp. 881–888, 2002.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *International Conference on Learning Representations*, 2018.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4822–4829, 2019.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

Ruslan Salakhutdinov, Antonio Torralba, and Josh Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR 2011*, pp. 1481–1488. IEEE, 2011.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 231–235, 2016.

Ximeng Sun, Rameswar Panda, and Rogerio Feris. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*, 2019.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pp. 550–558, 2016.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424, 2018.

Sen Wu, Hongyang R Zhang, and Christopher Ré. Understanding and improving information transfer in multi-task learning. *arXiv preprint arXiv:2005.00944*, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Liqiang Xiao, Honglun Zhang, Wenqing Chen, Yongkun Wang, and Yaohui Jin. Learning what to share: Leaky multi-task network for text classification. In *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 2055–2065, 2018.

Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: A tensor factorisation approach. In *International Conference on Learning Representations*, 2017.

A Appendix

A.1 Policy Initialization and Exploding Gradients

Rosenbaum et al. (2018) have a simplex projection function as the last step of their paper's Algorithm 3, defined as $clip(\pi)/\sum(clip(\pi))$ where $clip(\pi_i) = \max(0, \min(1, \pi_i))$. However, the associated code[2] does not follow this equation and instead uses the function $clip'(\pi_i) = \pi_i - \min_j(\pi_j) + \epsilon$ with $\epsilon = 10^{-6}$. Because π is initialized uniformly[3], in the first batch $\pi_i = \min_j(\pi_j)$ and the gradient of the simplex projection becomes proportional to $1/\epsilon^2 \approx 10^{12}$. The net effect of this particular exploding gradient issue is that the parameters of the routing policy are prone to taking on extreme values after the first weight update and then remaining fixed in that extreme position for the rest of training. The policy learner is not able to explore additional routing choices beyond the first weight update.

[2] `https://github.com/cle-ros/RoutingNetworks/blob/master/PytorchRouting/DecisionLayers/ReinforcementLearning/WPL.py#L32`

[3] `https://github.com/cle-ros/RoutingNetworks/blob/master/PytorchRouting/DecisionLayers/PolicyStorage.py#L44`

A.2 STATISTICAL SIGNIFICANCE

A two-sample one-tail t-test is conducted to validate whether there is sufficient statistical correlation to support the hypothesis that the performance of random selection is better than no sharing.

Let μ_f be the mean accuracy of fixed random selection and μ_τ the mean of no sharing. The null hypothesis is H_0, and the alternative hypothesis is H_1. Here the hypothesis of interest can be expressed as:

$$\begin{aligned} H_0 &: \mu_f - \mu_\tau \leq 0 \\ H_1 &: \mu_f - \mu_\tau > 0 \end{aligned} \tag{1}$$

The result show that there is statistical evidence on CIFAR-100 dataset, with $t = -2.2421$, $df = 8$, $p - value = 0.02762$ to reject the H_0 hypothesis, which is the evidence that the performance of random selection is better than no sharing. And we also find statistical evidence on Fashion-MNIST dataset, with $t = -2.0305$, $df = 8$, $p - value = 0.0384$.