DYNASHARE: DYNAMIC NEURAL NETWORKS FOR MULTI-TASK LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter sharing approaches for deep multi-task learning share a common intuition: for a single network to perform multiple prediction tasks, the network needs to support multiple specialized execution paths. However, previous parameter sharing approaches have relied on a static network structure for each task. In this paper, we propose to increase the capacity for a single network to support multiple tasks by radically increasing the space of possible specialized execution paths. DynaShare is a new approach to deep multi-task learning that learns from the training data a hierarchical gating policy consisting of a task-specific policy for coarse layer selection and gating units for individual input instances, which work together to determine the execution path at inference time. Experimental results on standard multi-task learning benchmark datasets demonstrate the potential of the proposed approach.

1 INTRODUCTION

Multi-task learning (MTL) focuses on adapting knowledge across multiple related tasks and optimizes a single model to predict all the tasks simultaneously. In contrast to single-task learning, it can improve generalization and parameter efficiency, while reducing the training and inference time by sharing parameters across related tasks. Traditional deep multi-task learning approaches use hard or soft parameter sharing to train a single network that can perform multiple predictive tasks. Architectures that use hard parameter sharing are composed of a shared backbone of initial layers, followed by a separate branch for each task. The shared backbone learns generic representations, and the dedicated branches learn task-specific representations. However, hard parameter sharing imposes a restrictive tree structure on the architecture, and is also susceptible to negative transfer, in which some tasks are optimized at the expense of others. Architectures that use soft parameter sharing are composed of multiple task-specific backbones, and parameters are linked across backbones by regularization or fusion techniques. However, scalability is a challenge as the network grows proportionally with the number of tasks.

In either case of parameter sharing, the path through the network at inference time is the same for all tasks and inputs. The recently introduced AdaShare method (Sun et al., 2020) revisited traditional parameter sharing patterns by enabling each task to learn its own path through the network: a layer can be executed exclusively by one task, shared by multiple tasks, or skipped entirely. This is achieved by learning a task-specific gating policy over the network layers. In AdaShare, the execution path is task-adaptive.

Parameter sharing approaches share a common intuition: *for a single network to perform multiple prediction tasks, the network needs to support multiple specialized execution paths (paths through the network at inference time)*. However, in all previous parameter sharing approaches, the execution paths are fixed after training. At inference time, the path through the network is the same for any input instance. In this paper, we propose to increase the capacity for a single network to support multiple tasks by radically increasing the space of possible specialized execution paths. DynaShare is a new approach to deep multi-task learning that learns from the training data a hierarchical gating policy consisting of a task-specific policy for coarse layer selection and gating units for individual input instances, which work together to determine the execution path at inference time. In other words, the multi-task network is trained end-to-end to support a larger space of dynamic execution

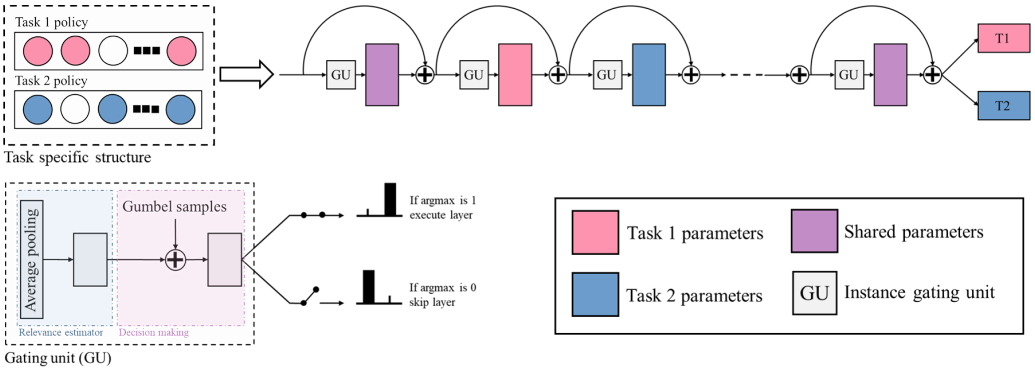

Figure 1: An overview of the proposed framework. DynaShare for multi-task learning enables dynamic selection of layers based on both the specific task and the individual input instance. At inference time, the selected layers for tasks is fixed but the network operates dynamically based on the input instance: the forward path is modified based on the computation needed for that particular instance. The gating unit for instance decision making consists of two parts: a) estimating relevance of an input to the layer b) making decision for executing the layer based on the relevance metric.

paths that are a function of both the task and the instance. The overall workflow of DynaShare is illustrated in Figure 1.

The contributions of this paper are:

- We propose a novel and fully differentiable dynamic method for multi-task learning where the network structure can be adapted based on both tasks and individual instances.
- We extensively verify the effectiveness of our method. Experimental results on two MTL benchmark datasets, UCI-Census-income (Census) and Medical Information Mart for Intensive Care (MIMIC-III), demonstrate the potential of DynaShare. For fair comparisons, a unified codebase will also be publicly available.

2 Related Work

Multi-Task Learning and Parameter Sharing: Learning multiple tasks with a single model is a challenge that is frequently encountered in natural language processing (Sanh et al., 2019; Liu et al., 2019c; Aksoy et al., 2020), computer vision (Sun et al., 2020; Liu et al., 2019b; Vandenhende et al., 2020), reinforcement learning (Pinto & Gupta, 2016; Hessel et al., 2019), and multi-modal learning (Pramanik et al., 2019; Lu et al., 2020), among other areas. Parameter or weight sharing is a well studied mechanism for facilitating shared knowledge transfer in neural networks. For example, in neural architecture search, parameter sharing enables a large number of candidate architectures to be simultaneously trained within a single supernet (Liu et al., 2019a; Guo et al., 2020; Wang et al., 2021). Each candidate architecture corresponds to a single execution path in the supernet. In multi-task learning, parameter sharing enables paths for multiple tasks to be simultaneously trained. Traditionally, this has been achieved via so-called hard or soft parameter sharing. Hard parameter sharing (Caruana, 1993; Kokkinos, 2017) involves training a shared backbone of layers followed by task-specific branches. Soft parameter sharing (Misra et al., 2016; Ma et al., 2018; Liu et al., 2019b) involves training multiple task-specific backbones and linking parameters across backbones using regularization or fusion techniques.

In contrast to hard or soft parameter sharing, AdaShare (Sun et al., 2020) learns which layers in a shared backbone network should be enabled for each task: a layer can be executed exclusively by one task, shared by multiple tasks, or skipped entirely. Each task learns a binary gating vector indicating which layers to execute. While AdaShare enables a network to support multiple execution paths, the gating decisions are static at inference time: for a given task, the same layers are always executed, independent of the input. In contrast, DynaShare adaptively activates an execution path based on both the task and input.

Dynamic Neural Networks: Dynamic neural networks have recently attracted attention from many research communities; a recent survey in the area is presented in (Han et al., 2021). Dynamic neural networks adapt the computation graph of the network depending on the input (instance). Among other applications such as inference acceleration, this adaptivity unlocks the potential of networks that can be trained once and deployed in diverse environments or under different operating conditions. For example, slimmable networks (Yu et al., 2019; Yu & Huang, 2019; Li et al., 2021) can be arbitrarily sliced channel-wise at deployment time according to the user's computational resource constraints.

Common strategies for adapting the computation graph in dynamic networks include modifying the input resolution (Wang et al., 2020a), selecting a subset of spatial locations (Verelst & Tuytelaars, 2020), selecting a subset of channels (Kim et al., 2018; Yu et al., 2019; Yu & Huang, 2019; Li et al., 2021), or selecting a subset of layers or blocks (Veit & Belongie, 2018; Wang et al., 2018; Wu et al., 2018). DynaShare follows this last strategy and selects a subset of blocks to build a computation graph that depends on both the input and the task.

Methods for dynamically selecting a subset of layers typically rely on policy networks (Wu et al., 2018; Chen et al., 2019) or gating functions (Wang et al., 2018; Veit & Belongie, 2018; Wang et al., 2020b). Policy learning approaches like BlockDrop (Wu et al., 2018) and GaterNet (Chen et al., 2019) use a policy network that, given an input, produces skipping decisions for all the layers in the backbone network. The main challenges of such methodologies are related to the handcrafted design and computational cost of the additional policy network. Gating approaches such as (Wang et al., 2018; Veit & Belongie, 2018; Wang et al., 2020b) introduce functions responsible for controlling the local inference graph of a layer or block. The main advantage of such techniques is related to the easy integration, in a plug-in fashion, of these gates to existing methodologies. Nevertheless, they often require costly training techniques to address the non-differentiability of the gates. DynaShare employs a fully differentiable, hierarchical design consisting of a task-specific policy for coarse layer selection that is further refined by instance-specific gating.

3 Proposed Method

DynaShare is an approach for training dynamic multi-task networks that adapt the execution path based on both the task and the individual instance. DynaShare learns from the training data a hierarchical gating policy consisting of a task-specific policy for coarse layer selection and gating units for individual input instances, which work together to determine the execution path at inference time. The policies and gates are learned jointly in a fully differentiable way. In this section, we first explain DynaShare's task-specific policy, which is inspired by AdaShare (Sun et al., 2020) and establishes a static layer execution policy for each task. We then build on the task-specific policy with instance-specific gating units, which learn a dynamic gating that further modifies the task-specific policy based on the instance.

3.1 Learning a Task-Specific Policy

The task-specific policy is a discrete vector that determines which layers to execute or skip for each task (Figure 1, top left). We follow the same designed loss functions as proposed in AdaShare (Sun et al. (2020)) for learning this policy. Learning a discrete policy in an standard end-to-end differentiable backpropagation fashion is challenging. Direct optimization of the task-specific policy is possible through the use of Gumbel-Softmax sampling (Jang et al. (2016)). We will revisit Gumbel-Softmax sampling in Section 3.3.

Without adding a significant number of parameters, we train the network weights and the feature sharing policy jointly. To designate a sharing pattern, we sample a task-specific policy from the learned distribution to specify which blocks are selected in different tasks. Once the task-specific policy is defined, the dynamic instance gating will refine the structure based on individual inputs.

3.2 Learning Instance-Specific Gating

DynaShare's instance-specific gating units learn to further adjust the selection decisions of the task-specific policy based on the characteristics of the input (instance). Selecting a subset of layers or

blocks based on the input has been demonstrated previously (Veit & Belongie, 2018; Wang et al., 2018; Wu et al., 2018). DynaShare is the first approach to learn a hierarchical task and instance adaptive layer gating mechanism and to illustrate its potential in dynamic multi-task learning.

Inspired by Veit & Belongie (2018), we construct an instance gating method with two parts: a) estimating relevance of each layer to the input of the same layer, b) making decision to keep or skip the layer based on the estimated relevance metric. Other approaches, such as policy optimization using reinforcement learning (Wang et al., 2018; Wu et al., 2018) are also possible. The relevance estimator is a lightweight design consisting of two convolution layers followed by an activation function. We use the computed output score of the relevance estimator to further make a decision on executing the layer for each individual input. Similar to task-specific policy learning, such a discrete controller is possible with the use of Gumbel-Softmax Sampling.

3.3 GUMBEL-SOFTMAX SAMPLING

Given a set of tasks $T = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_K\}$ over a dataset, we use Gumbel-Softmax sampling (Jang et al. (2016)) to learn both task-specific and instance-specific discrete value policies. With this method, instead of sampling from the distribution of a binary random variable to find the optimized policy for each block l and specific task $\mathcal{T}_k$, we can generate the decision from:

$$u_{l,k} = \underset{j \in \{0,1\}}{\arg\max}(\log \pi_{l,k}(j) + G_{l,k}(j)), \tag{1}$$

where $\pi_{l,k} = [1 - \alpha_{l,k}, \alpha_{l,k}]$ is its distribution vector with $\alpha_{l,k}$ representing the probability the l_{th} layer is executed for task $\mathcal{T}_k$. $G_{l,k} = -\log(-\log U_{l,k})$ are Gumbel random variables with $U_{l,k}$ sampled from a uniform distribution. With the reparameterization trick (Jang et al. (2016)), we can relax the non-differentiable argmax operator in Eq. 1:

$$u_{l,k}(j) = \frac{\exp\left((\log \pi_{l,k}(j) + G_{l,k}(j))/\tau\right)}{\sum\limits_{j \in \{0,1\}} \exp\left((\log \pi_{l,k}(j) + G_{l,k}(j))/\tau\right)}, \tag{2}$$

where τ is the softmax temperature. When $\tau \to 0$, the softmax function gets closer to the argmax function and we would have a discrete sampler, and when $\tau \to \infty$ it becomes a uniform distribution. Following Sun et al. (2020), for task-specific policy during training we use the relaxed version given by Eq. 2. We set $\tau = 5$ as initial value and gradually decrease it to 0. After training, we sample from the learned distribution to obtain a discrete task-specific policy. For learning the instance-specific policy, we get a discrete sample from Eq. 1 during the forward pass and we compute the gradient of relaxed version given by Eq. 2 in the backward pass.

3.4 LOSS FUNCTIONS

The goal of DynaShare is to achieve the best performance across all the tasks with the most efficient parameter sharing strategy. Therefore, while training the model we have to consider optimizing not only the accuracy but also minimising the number of parameters without performance degradation. We also need to encourage sharing the blocks among different tasks and prevent the case of splitting parameters with no knowledge sharing. To address these issues we used sparsity regularization and a sharing loss ($\mathcal{L}_{sparsity}$ and $\mathcal{L}_{sharing}$, respectively) to minimize the log-likelihood of the probability of a block being executed and maximise the knowledge sharing simultaneously:

$$\mathcal{L}_{sparsity} = \sum_{l \leq L, k \leq K} \log \alpha_{l,k}, \tag{3}$$

$$\mathcal{L}_{sharing} = \sum_{k_1, k_2 \leq K} \sum_{l \leq L} \frac{L - l}{L} |\alpha_{l,k_1} - \alpha_{l,k_2}|, \tag{4}$$

where L is the total number of layers. For learning the instance gating, we add a loss term that encourages each layer to be executed at a certain target rate t. We estimate the execution rates over each mini-batch and penalize deviation from given target rate. We calculate the instance loss as

$$\mathcal{L}_{instance} = \sum_{l \leq L} (\beta_l - t)^2, \tag{5}$$

where β_l is the fraction of instances within a mini-batch for which the l_{th} layer is executed. Including the task-specific losses, the final training loss of DynaShare is:

$$\mathcal{L}_{total} = \sum_k \lambda_k \mathcal{L}_k + \lambda_{sparsity}\mathcal{L}_{sparsity} + \lambda_{sharing}\mathcal{L}_{sharing} + \lambda_{instance}\mathcal{L}_{instance}, \tag{6}$$

where $\mathcal{L}_k$ is the task-specific loss, weights λ_k are used for task balancing, and $\lambda_{sparsity}$, $\lambda_{sharing}$, $\lambda_{instance}$ are the balance parameters for sparsity, sharing and instance losses, respectively.

4 EXPERIMENTS

In this section, we present experimental results and ablation studies to validate the proposed approach. We compare DynaShare to single-task learning, as well as to multi-task learning baselines including conventional hard parameter sharing and two multi-gate mixture-of-experts models: MMoE (Ma et al., 2018) and a recent extension with expert diversity, MMoEEx (Aoki et al., 2021). We then explore the effectiveness of both the task-specific policy and instance-adaptive gating by conducting ablation studies on each component.

4.1 DATASETS AND EVALUATION METRICS

We evaluate the performance of our proposed method on two multi-task learning datasets: Census (Ma et al., 2018) and MIMIC-III (Johnson et al., 2016; Harutyunyan et al., 2019).

UCI-Census-income dataset is extracted from US 1994 census database. It has 299,285 data points with 40 features. The features are extracted from the respondent's socioeconomic form. The three binary classification tasks associated with this dataset include respondent income exceeds \$50K (Income), respondent's marital status is "ever married" (Marital Stat), and respondent's education is at least college (Education).

MIMIC-III consists of patients information from over 40,000 intensive care units (ICU) stays. The four tasks of this dataset are Phenotype prediction (Pheno), In-hospital mortality prediction (IHM), Length-of-stay (LOS), and Decompensation prediction (Decomp). MIMIC-III is the main benchmark for heterogenous multi-task learning with time series. It is considered heterogenous due to the different task characteristics. The dataset includes two binary tasks, one temporal multi-label task, and one temporal classification task, respectively.

For a fair comparison we adopted the same split between train, validation, and test set as used by all the previous baselines. For both the Census and MIMIC-III datasets, the ratios are 70%, 15%, and 15%. The metric for comparing the results is AUC (Area Under The Curve) ROC for the binary tasks and Kappa Score for the multiclass tasks. We report the multi-task learning performance of each method with Δ as defined in Maninis et al. (2019), which compares results with their equivalent single task values:

$$\Delta_{\mathcal{T}_k} = \frac{1}{|M|} \sum_{j=0}^{|M|} (-1)^{l_j} (M_{\mathcal{T}_k,j} - M_{ST,j})/M_{ST,j} * 100\%, \tag{7}$$

where $l_j = 1$ if a lower value shows better performance for the metric M_j and 0 otherwise. $M_{ST,j}$ is the single task value of j metric. We calculate the overall performance by averaging $\Delta_{\mathcal{T}_k}$ over all tasks:

$$\Delta = \frac{1}{K} \sum_{k \leq K} \Delta_{\mathcal{T}_k}, \tag{8}$$

4.2 EXPERIMENTAL SETTING

Following AdaShare's task-specific policy training strategy (Sun et al., 2020), for the first few epochs we consider hard parameter sharing by sharing all blocks across tasks. This sets the network at a good starting point for policy learning. We use curriculum learning (Bengio et al., 2009) to encourage a better convergence. We then optimize the network and task-specific policy distribution parameters alternatively (Algorithm 1). After learning the task-specific policy distribution, we sample the distribution and fix the network structure based on task policy. Then, we re-train the network to learn the instance gating using the full training set (Algorithm 2).

Algorithm 1 DynaShare training algorithm

$t \leftarrow 1$ ▷ target rate equal to 1 means all the layers are executed for all the instances (Eq. 5)
for warm-up epochs **do** ▷ Warm-up stage: hard parameter sharing
    $\mathcal{L}_{Total} = \sum_k \lambda_k \mathcal{L}_k$
    optimise network weights
end for
while maximum epochs not reached **do** ▷ Optimise network and task policy distribution alternatively
    for e_1 epochs **do**
        $\mathcal{L}_{Total} = \sum_k \lambda_k \mathcal{L}_k$
        optimise network weights
    end for
    for e_2 epochs **do**
        $\mathcal{L}_{Total} = \sum_k \lambda_k \mathcal{L}_k + \lambda_{sparsity} \mathcal{L}_{sparsity} + \lambda_{sharing} \mathcal{L}_{sharing}$
        optimise task policy distribution
    end for
end while
return task policy distribution and network weights

Algorithm 2 DynaShare re-training algorithm

$t \leftarrow$ desired target rate ▷ (Eq. 5)
task network structure $\leftarrow$ sample of task policy
while maximum epochs not reached **do**
    $\mathcal{L}_{Total} = \sum_k \lambda_k \mathcal{L}_k + \lambda_{instance} \mathcal{L}_{instance}$
    optimise task network structure with target rate
end while
return network weights

We used PyTorch for implementation of our technique, ResNet-18 (9 blocks) as backbone, Adam as optimizer for policy distribution parameters and SGD to update network weights. We set the learning rate to 0.001 that is halved every 250 epochs, binary cross-entropy loss for binary tasks and cross-entropy loss for multi-label tasks. We used the task's AUC sum in the validation set to define the best model, where the largest sum indicates the best epoch, and consequently, the best model. For MIMIC-III, we trained the models using the training set with batch size of 256 for 1000 epochs. For Census, batch size and number of epoch are 450, and 500, respectively.

4.3 COMPARISON WITH EXISTING BASELINES

Census dataset is a homogeneous dataset where we aim to focus on benchmarking the flexibility of DynaShare to learn multiple tasks with similar optimization goals. The Census income, marital status and education dataset experiments are presented in Table 1. The best results are highlighted in bold, and the second-best results are underlined. DynaShare outperforms all the baselines for the education task, which can be considered the hardest due to its lower values. Our approach also performs similarly to the MMoEEx approach on the Marital Status task. The delta difference

Table 1: AUC results on all three tasks of the Census dataset. DynaShare outperforms the current approaches for the education task and obtains competitive performance overall.

Method	Income	Marital Stat	Education	Δ
Single Task	88.95	97.48	87.23	-
AdaShare	89.44	96.79	84.56	-1.07 %
MMoE	90.86	96.70	86.33	-0.28%
Shared Bottom	91.09	97.98	86.99	+0.85%
MMoEEx	**92.51**	**98.47**	87.19	**+1.65**%
DynaShare (Ours)	91.01	98.32	**87.31**	+1.09%

Table 2: AUC results for the MIMIC-III dataset. DynaShare outperforms all other baselines in two out of four individual tasks, and achieves an overall lift in relative performance when considering the full set of tasks.

Method	Pheno	LOS	Decomp	IHM	Δ
Single Task	77.00	45.00	91.00	86.00	-
Shared Bottom	73.36	30.60	94.12	82.71	-9.28%
MCW-LSTM	**77.40**	45.00	90.50	87.00	+0.28%
MMoE	75.09	54.48	96.20	90.44	+7.36%
MMoEEx - GRU	74.57	60.63	**97.03**	91.03	+11.00%
MMoEEx - RNN	72.44	63.45	96.82	90.73	+11.74%
DynaShare (Ours)	76.50	**71.86**	84.98	**94.25**	**+15.50**%

between DynaShare and the current state of the art of 0.56% is mainly due to the performance on the Income task. Overall, DynaShare shows competitive performance with respect to the state-of-the-art on this set of tasks, which have relatively low variability. The advantage offered by DynaShare's larger space of possible specialized execution paths is more clearly demonstrated when the tasks have higher variability, which we discuss next.

MIMIC-III dataset constitutes a well established benchmark for heterogeneous multi-task learning. In contrast to the Census dataset, the tasks have high variability: two binary tasks, one temporal multi-label task, and one temporal classification task. This diversity in task characteristics makes MIMIC-III a more challenging benchmark. For this dataset we compared our method with single-task training (separate networks trained for each task), hard-parameter sharing, channel wise LSTM (MCW-LSTM) (Johnson et al. (2016)), MMoE (Ma et al. (2018)), and two variants of MMoEEx with RNN and GRU (Aoki et al. (2021)). The full set of results on MIMIC-III dataset is presented in Table 2. DynaShare outperforms all approaches in two out of the four tasks. It is noticeable that for the LOS task, which can be considered the hardest, DynaShare outperforms all previous results by a large margin, with an improvement of 8%, including MMoEEx (Aoki et al., 2021) which was the previous state-of-the-art. In-Hospital Mortality is another task DynaShare outperforms the previous reported methods by a substantial margin, in this case 3%. Overall, DynaShare obtains the highest average delta improvement relative to the single-task model of 15.50%, which is a 3.7% improvement over the previous top performing approach.

These results suggest that DynaShare is particularly effective at learning multiple tasks with high variability, when compared to shared bottom approaches and mixture of experts baselines. When tasks have high variability, DynaShare's hierarchical task instance approach, which expands the space of possible execution paths, provides more flexibility to support diverse multi-task learning.

Table 3: Ablation studies to evaluate the impact of the task-specific policy and instance-specific gating on overall multi-task learning performance on the MIMIC-III dataset.

Method	Pheno	LOS	Decomp	IHM	Δ
Single Task	**77.00**	45.00	**91.00**	86.00	-
Task-specific policy only (AdaShare)	76.39	71.68	82.08	<u>94.15</u>	+14.54%
Instance-specific gating only	73.06	68.78	82.24	92.10	+11.30%
Task-specific policy with instance-specific gating (DynaShare)	<u>76.50</u>	**71.86**	<u>84.98</u>	**94.25**	**+15.50**%

4.4 Ablation studies

To evaluate the impact of both the task-specific policy and instance-specific gating components, we conducted ablation experiments on the MIMIC-III dataset. First, we trained the network with a task-specific policy only, which is equivalent to reducing the method to AdaShare (Sun et al. (2020)). With a task-specific policy only, we achieve a $+2.8\%$ lift in performance compared to the best previous baseline (MMoEEx-RNN). Next, we trained the network with instance-specific gating only. This achieves similar performance with state of the art technique. The full results of these ablation studies are reported in Table 3. In summary, both the task-specific policy and instance-specific gating improve the performance of DynaShare. For ResNet-18 the instance gating function only adds 0.4% to the task-specific network parameters at training, while it improves the average relative performance of all tasks by 0.96%.

5 Conclusion

In this paper, we presented a new method for deep multi-task learning, inspired by an interpretation of previous parameter sharing strategies as different ways for a single network to support multiple specialized execution paths. We increased the space of possible execution paths that the network can access by parameterizing the execution path according to both the task and the input instance. This is achieved by learning a hierarchical gating policy, consisting of a task-specific policy and instance-specific gating. We validated DynaShare on two publicly available multi-task learning benchmarks, showing promising performance with respect to state-of-the-art methods.

Reproducibility Statement

The paper uses data obtained from Medical Information Mart for Intensive Care (MIMIC-III) and UCI-Census-income (Census) (Johnson et al., 2016; Harutyunyan et al., 2019; Ma et al., 2018). To access MIMIC-III a request must be submitted to `https://mimic.mit.edu/iii/gettingstarted/`. The pre-processing steps of this dataset are available at `https://github.com/YerevaNN/mimic3-benchmarks`. Census dataset is publicly available and the full dataset and documentation can be downloaded from `http://archive.ics.uci.edu/ml`. Our code will be made publicly available on Github on paper acceptance.

References

Çagla Aksoy, Alper Ahmetoglu, and Tunga Güngör. Hierarchical multitask learning approach for BERT. *arXiv preprint arXiv:2011.04451*, 2020.

Raquel Aoki, Frederick Tung, and Gabriel L Oliveira. Heterogeneous multi-task learning with expert diversity. *arXiv preprint arXiv:2106.10595*, 2021.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning*, 2009.

Rich Caruana. Multitask learning: A knowledge-based source of inductive bias. In *International Conference on Machine Learning*, 1993.

Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gaternet for dynamic filter selection in cnns. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, 2020.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *arXiv preprint arXiv:2102.04906*, 2021.

Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. Multi-task learning and benchmarking with clinical time series data. *Scientific data*, 6(1):1–18, 2019.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. Technical report, DeepMind, 2019. URL `https://www.aaai.org/ojs/index.php/AAAI/article/view/4266`.

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

Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

Eunwoo Kim, Chanho Ahn, and Songhwai Oh. NestedNet: Learning nested sparse structures in deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Iasonas Kokkinos. UberNet: Training a 'universal' convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a.

Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019b.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Annual Meeting of the Association for Computational Linguistics*, 2019c. doi: 10.18653/v1/P19-1441.

Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 12-in-1: Multi-task vision and language representation learning. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2020.

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 *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

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

Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. *arXiv preprint arXiv:1609.09025*, 2016.

Subhojeet Pramanik, Priyanka Agrawal, and Aman Hussain. OmniNet: A unified architecture for multi-modal multi-task learning. *arXiv preprint arXiv:1907.07804*, 2019.

Victor Sanh, Thomas Wolf, and Sebastian Ruder. A hierarchical multi-task approach for learning embeddings from semantic tasks. In *AAAI Conference on Artificial Intelligence*, volume 33, 2019. doi: 10.1609/aaai.v33i01.33016949.

Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. AdaShare: Learning what to share for efficient deep multi-task learning. In *Advances in Neural Information Processing Systems*, 2020.

Simon Vandenhende, Stamatios Georgoulis, and Luc Van Gool. MTI-Net: Multi-scale task interaction networks for multi-task learning. In *European Conference on Computer Vision*, 2020.

Andreas Veit and Serge Belongie. Convolutional networks with adaptive computation graphs. In *European Conference on Computer Vision*, 2018.

Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable NAS. In *International Conference on Learning Representations*, 2021.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell1, and Joseph E. Gonzalez. SkipNet: Learning dynamic execution in residual networks. In *European Conference on Computer Vision*, 2018.

Yikai Wang, Fuchun Sun, Duo Li, and Anbang Yao. Resolution switchable networks for runtime efficient image recognition. In *European Conference on Computer Vision*, 2020a.

Yue Wang, Jianghao Shen, Ting-Kuei Hu, Pengfei Xu, Tan Nguyen, Richard Baraniuk, Zhangyang Wang, and Yingyan Lin. Dual dynamic inference: Enabling more efficient, adaptive and controllable deep inference. *IEEE Journal of Selected Topics in Signal Processing*, 2020b. doi: 10.1109/JSTSP.2020.2979669.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. BlockDrop: Dynamic inference paths in residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Jiahui Yu and Thomas Huang. Universally slimmable networks and improved training techniques. In *IEEE/CVF International Conference on Computer Vision*, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019.