Do Current Multi-Task Optimization Methods in Deep Learning Even Help?

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent research has proposed a series of specialized optimization algorithms for deep multi-task models. It is often claimed that these multi-task optimization (MTO) methods yield solutions that are superior to the ones found by simply optimizing a weighted average of the task losses. In this paper, we perform large-scale experiments on a variety of language and vision tasks to examine the empirical validity of these claims. We show that, despite the added design and computational complexity of these algorithms, MTO methods do not yield any performance improvements beyond what is achievable via traditional optimization approaches. We highlight alternative strategies that consistently yield improvements to the performance profile and point out common training pitfalls that might cause suboptimal results. Finally, we outline challenges in reliably evaluating the performance of MTO algorithms and discuss potential solutions.

1 Introduction

Multitask models are ubiquitous in deep learning [1, 2, 14]. This popularity stems from the fact that these models can potentially leverage transfer learning in between different tasks and modalities. Moreover, by reducing the number of the models that need to be maintained, multitask models greatly simplify serving users.

Multitask models come with their own challenges and downsides. Different tasks often compete with each other for model capacity, leading to *the task interference* problem. Finding the training setting that strikes the right balance between different tasks is an engineering intensive endeavor that requires extensive trial and error for most realistic setups.

Over the past few years, a vast number of multi-task optimization (MTO) algorithms have been proposed in the literature that claim to alleviate the task interference problem [5, 8, 18, 17, 21, 23, 24]. These algorithms typically leverage clever intuition about the training process to dynamically balance the different tasks throughout training. However, in exchange for this, these algorithms often drastically add to the computational and design complexity of the model.

The goal of this paper is not to provide another MTO algorithm. Instead, we provide a large-scale empirical study of the algorithms presented in the literature; we examine to what degree the improvements presented in the literature are reproducible and whether these algorithms reduce loss interference between different tasks. We provide the following observations:

- Despite the added complexity, MTO algorithms fail to improve the interference profile beyond what is achievable by simple static weighting of different tasks (Section 4).

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

- The performance of multi-task models is sensitive to basic optimization parameters such as learning rate and weight-decay. Insufficient tuning of these hyper-parameters in the baselines, along with the complexity of evaluation for multi-task models, can create a false perception of performance improvement (Section 4).
- In some instances, the gains reported in the MTO literature are due to flaws in the experimental design. In some cases, these reported gains disappear with better tuning of the baseline hyperparameters. In other cases, we were unable to reproduce them (Section 4.2).
- Finally, we discuss the implications for the community and the potential steps that need to be taken to standardize evaluation for multi-task models (Section 5).

2 Setting

We focus our discussion on supervised learning. We consider the setting where the model parameters, $\boldsymbol{\theta} \in \mathbb{R}^p$, are trained on K different tasks. We denote the loss associated with task i with $\mathcal{L}_i(\boldsymbol{\theta})$.

For some problem instances, the parameter space contains a globally optimal point that achieves the best possible performance on all tasks. Figure 1 (left) provides a cartoon example of one such scenario. However, for most realistic setups, a globally optimal $\boldsymbol{\theta}$ doesn't exist. In these cases, different tasks compete with each other for model capacity. In these scenarios, the concept of Pareto optimality is used to capture the optimal trade-off in between the tasks:

Definition (Pareto Optimality)**.** *$\boldsymbol{\theta} \in \mathbb{R}^p$ Pareto dominates another $\boldsymbol{\theta}'$ if $\forall 1 \leq i \leq K$, $\mathcal{L}_i(\boldsymbol{\theta}) \leq \mathcal{L}_i(\boldsymbol{\theta}')$ and there exists a task j where $\mathcal{L}_j(\boldsymbol{\theta}) < \mathcal{L}_j(\boldsymbol{\theta}')$. $\boldsymbol{\theta}$ is Pareto optimal if it is not dominated by any other solution. The collection of the Pareto optimal points is denoted as Pareto front.*

Figure 1 (center) provides a cartoon representation of the Pareto front for a two-task setup. The Pareto front represents the collection of the parameters that achieve the best possible trade-off profile between the tasks. A practitioner can aim to land on a particular point on this trade-off curve depending on their (implicit or explicit) utility function. The location and the curvature of the Pareto curve represent the severity of the interference problem. We would like to find training protocols that push the trade-off curve towards the origin as much as possible (Figure 1-right).

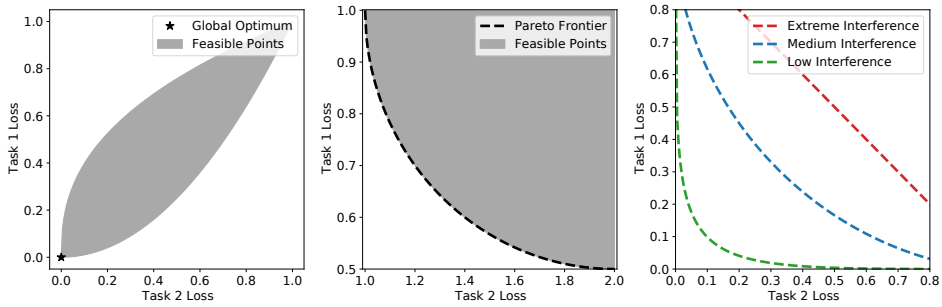

Figure 1: A cartoon representation of the achievable trade-offs in a two-task setup.

The traditional approach to optimize multi-task models is via *scalarization* [3]:

$$\hat{\boldsymbol{\theta}}(\boldsymbol{w}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{w}) \quad \text{where} \quad \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{w}) \equiv \sum_{i=1}^{K} \boldsymbol{w}_i \mathcal{L}_i(\boldsymbol{\theta}), \quad \boldsymbol{w} > 0, \quad \sum_i \boldsymbol{w}_i = 1. \tag{1}$$

Here, $\boldsymbol{w}$ is a fixed vector of task weights determined by the practitioner beforehand. The algorithmic and computational simplicity of this approach has made scalarization highly popular in practice.

Scalarization comes with certain theoretical guarantees. It can be easily shown that any solution to problem (1) is guaranteed to be Pareto optimal. In addition, when $\{\mathcal{L}_i\}_{i=1}^K$ are convex, there exists a partial converse: Let $\boldsymbol{\theta}^{\#}$ be a point on the Pareto front. Then there exists $\boldsymbol{w}^{\#} \geq 0$ such that

scalarization with $\boldsymbol{w}^{\#}$ yields $\boldsymbol{\theta}^{\#}$. [1] These results suggest that, at least for convex setups, sweeping the task weights should be sufficient for full exploration of the Pareto frontier. In particular, **in the convex setting it is provable that no algorithm can outperform properly chosen scalarization that has been trained to convergence.**

The results above raise a series of questions. Where are the reported improvements of MTO algorithms coming from? Is non-convexity adding additional complexity which makes scalarization insufficient for tracing out the Pareto front? Is it the case that neural networks trained via a combination of scalarization and standard first-order optimization methods are not able to reach the Pareto Frontier? Do MTO algorithms achieve a better performance trade-off curve? In following sections, we empirically examine these questions for several popular deep learning workloads.

3 Prior Work

There has been a flurry of research on MTO algorithms over the past few years. [10, 4] argue that finding the appropriate scalarization weights is often costly. To alleviate this, they provide algorithms that aim to automatically find a reasonable set of task weights. Sener & Koltun (2019) [21] approach multi-task learning from a multi-objective optimization view point and suggests Multiple Gradient Descent Algorithm (MGDA) for efficiently finding Pareto optimal solutions. [24, 23, 18, 15] hypothesize that negative interactions between the gradients of different tasks is a significant contributor to the interference problem. As such, these studies put forward various suggestions for projecting out conflicting gradients in order to improve the optimization dynamics. Finally, [5, 17] propose algorithms that inject randomization into the training pipeline and argue that this added randomness improves the training dynamics by allowing the optimization trajectory to escape poor local minima.

It is important to note that MTO algorithms often come with substantial computational overhead. [5] report 2-5 fold increase in the training time for a 40-task benchmark. Similarly, [13] observe that on some benchmarks MTO algorithms can train as much as 35 times slower compared to scalarization.

Recently, there has been a number of studies that critically question the benefits of MTO algorithms. The closest such study to ours is Kurin et al. (2022) [13] that appeared on Arxiv during the preparation of this manuscript. The paper argues that MTO algorithms implicitly regularize the model and shows that with careful regularization, scalarization with equal weights can match the performance of MTOs on various popular benchmarks.

In contrast, we argue that MTO algorithms yield different solutions on the same trade-off curve (See Figure 2 for an example). In most cases, these solutions tend to be different from equal weighting scalarization solution. When performance on popular benchmarks is concerned, we argue that scalarization baselines are often undertuned. With additional tuning of the hyper-parameters we find that most optimizers yield comparable results.

4 Experiments

4.1 Multilingual Machine Translation

In this section, we examine the effect of MTO algorithms on multilingual neural machine translation (NMT). In particular, we focus on translation out of English as prior work has reported significant task interference in this translation direction [1].

We start off by examining models trained jointly on English→{French, Chinese} translation tasks. The two-task setup allows us to effectively picture the performance trade-off curves. French and Chinese are specifically chosen due to the large difference in their semantic and syntactic structures. Here, we anticipate a large degree of interference among the tasks—a setting where MTOs claim to improve upon scalarization. We repeat our experiments for English→{French, German} and

[1] The precise statements and their proofs are provided in the appendix.

English→{French, Romanian} translation tasks to ensure that our observations generalize across different task setups with different levels of data imbalance. See Table 1 for an overview of data sources. All models use (pre-LN) Transformer architecture [22] and have been trained using early stopping. See Appendix A for training details.

Table 1: Overview of data sources used in our NMT experiments.

Language Pair	Dataset	# Train Examples	# Eval Examples
English-French	WMT15	$40,853,298$	$4,503$
English-Chinese	WMT19	$25,986,436$	$3,981$
English-German	WMT16	$4,548,885$	$2,169$
English-Romanian	WMT16	$610,320$	$1,999$

We compare the performance trade-offs achieved by various popular MTO algorithms with the Pareto frontier of scalarization. Following the NMT literature's convention, we implement scalarization via proportional sampling. Here, the average number of observations in the batch corresponding to task i is proportional to $\boldsymbol{w}_i$. In this setup, the expected training loss is equal to

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}}[\ell(\boldsymbol{x};\boldsymbol{\theta})] = \sum_{i=1}^{K} \mathbb{P}(\boldsymbol{x} \in \text{task } i)\mathbb{E}_{\boldsymbol{x}}[\ell(\boldsymbol{x};\boldsymbol{\theta})|\boldsymbol{x} \in \text{task } i] = \sum_{i=1}^{K} \boldsymbol{w}_i \mathcal{L}_i(\boldsymbol{\theta}).$$

We compare scalarization with a series of popular MTO algorithms: Multiple Gradient Descent (MGDA) [21, 6, 15], GradNorm [4], Gradient Surgery (PCGrad) [24], IMTL [18], and Random Loss Weighting (RLW) [17]. For GradNorm's α hyper-parameter, we perform a grid search and report all non-Pareto dominated models. Moreover, to give an apples-to-apples comparison, all models have been trained with the same batch-size for the same number of training steps. All models use Adam [11] as the base optimizer. For all these optimizer categories, we tune the learning rate on a grid from 5×10^{-2} to 5 and report all non-Pareto dominated models. Details of the training and hyper-parameters are presented in Appendix A.

The overview of our experimental findings are presented in Figures 2, 4, and 5. The blue dashed line corresponds to the Pareto front achieved via proportional sampling with English→French sampling rate ranging from 10% to 90%. Several observations are in order:

No Improvements from MTO Algorithms Despite the promise to alleviate interference among the tasks, all of the MTO algorithms in our study simply yield performance trade-off points on the scalarization Pareto front. As such, their performance can be fully replicated by simply optimizing a weighted average of the losses. To understand this phenomenon better, in Figure 3, we plot the evolution of the task weights for PCGrad, MGDA, GradNorm, and IMTL during training. We observe that for the majority of the training runs, the dynamically assigned task weights do not move significantly. As such, in effect, these MTO algorithms behave similar to static weighting.

Other Language Pairs En→Fr and En→Zh are both high-resource tasks with $O(10^7)$ training examples. In these experiments, we observe minimal overfitting and excellent agreement between train and test behaviors. One might argue that MTO algorithms possess transfer learning and regularization capabilities En→{Zh, Fr} experiments downplay [2]. To address this, we repeat our experiments in two new task setups where En→Zh task is substituted with En→De (mid-resource) and En→Ro (low-resource).

Figures 4 & 5 present the results of these experiments. En→{De, Fr} experiments closely resemble En→{Zh, Fr} ones: MTO algorithms simply achieve different trade-off points on the scalarization Pareto front. The results for En→{Ro, Fr} are more interesting. We still observe a clear Pareto front for the training performance; different MTO algorithms achieve different points on this curve. For the generalization behavior however, the Pareto frontier ceases to exist. Instead, we observe

[2] See [13] for an overview of proposals on how MTO algorithms can perform implicit regularization.

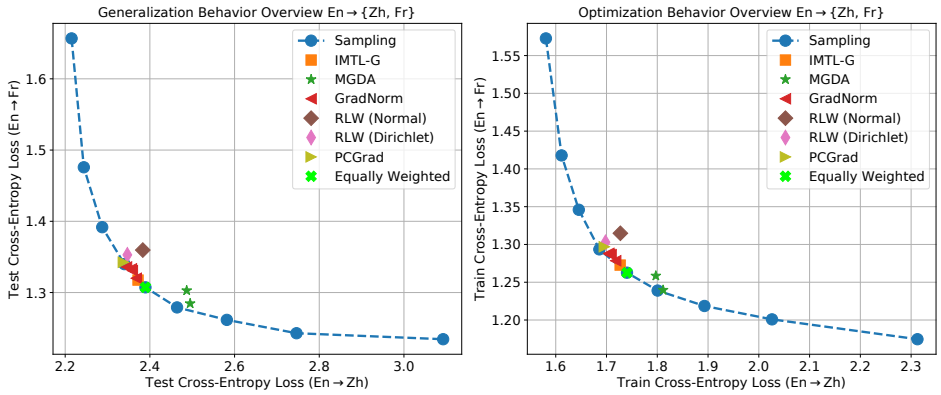

Figure 2: Performance trade-off behavior for En→{Fr, Zh} models. Each point corresponds to the final performance of a model. We observe no improvements in terms of final performance or training behavior from MTO algorithms. See Appendix B for more comparisons.

Figure 3: The evolution of task weights during training. All models are trained with the same (near optimal) base learning rate of 0.5. For the majority of the runs, the task weights barely move.

models that are *globally optimal*. Interestingly, these globally optimal solutions are found only by scalarization (with sampling rates close to $(0.3, 0.7)$). MTO algorithms find solutions with near equal task weights. As such, their generalization performance is significantly suboptimal. These results suggest that the extra compute power spent on MTO algorithms might be better spent on proper tuning of scalarization.

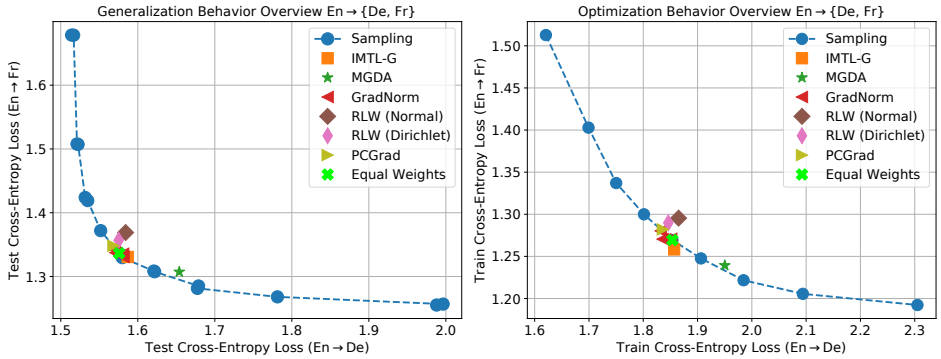

Figure 4: Performance trade-off behavior for En→{De, Fr} models. The results closely resemble our observations on En→{Zh, Fr} experiments.

Evaluation Challenges Our experiments suggest that the model performances are highly sensitive to the choice of hyper-parameters. Even subtle choices regarding the hyper-parameter grid can

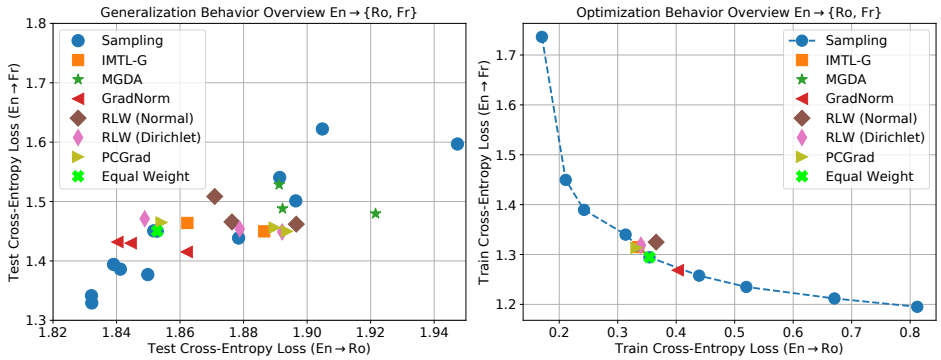

Figure 5: Performance trade-off behavior for En→{Ro, Fr} models. *Left:* We do not observe a Pareto frontier for the test performance. Instead, scalarization with weights $(0.3, 0.7)$ achieves a globally optimal trade-off and outperforms MTO algorithms. *Right:* For the training loss, a clear Pareto frontier appears and MTO algorithms simply selects a point on the performance profile traced out by the scalarization sweep.

drastically change the results. For example, it is common practice to tune the learning rate on a sparse grid, say sweeping $\eta \in [10^{-3}, 10^{-2}, 10^{-1}]$. How much do our reported metrics suffer from such a sparse sweep, and how much performance can be gained on average from further tuning of just this one hyperparameter? To answer this question we simulate running multiple instances of sparse grid searches of the form $\{k \times 10^{-3}, k \times 10^{-2}, k \times 10^{-1}\}$ for $1 \leq k \leq 9$; each choice of k produces a tuning study sweeping over 3 learning rates. We then measure the variance in the optimal performance of each 3 trial study, as k is varied from 1 to 9. The results are shown in Figure 6 (left). For comparison, we plot the variance in performance resulting from running the best η multiple times with different seeds. Notably, the effective standard deviation resulting from sparse learning rate tuning is 6 to 7 times the standard deviation observed from varying the random seed for a fixed hyperparameter point. The upshot is, **estimating trial variance by rerunning multiple seeds is insufficient for concluding that performance gains from a new algorithm are significant when the hyperparameters are sampled on a sparse grid**.

The established convention in the literature for ranking MTO performances is to compare some kind of average of the per-task performances. The specific average used is fixed somewhat arbitrarily for the purposes of benchmarking. However, in applied settings the utility function for ranking algorithms may vary dramatically depending on the goals of the practitioner. Thus, useful MTOs need to be robust to changes in the utility function. Either they need to improve upon the performance profile curve traced out by a sampling sweep, or reliably find better points on the profile curve with minimal tuning as the utility function is varied. Unfortunately, the current practice to consider just one (arbitrary) weighting scheme will bias evals towards algorithms that perform well on that specific scheme but are not robust to changes in the utility function. For example, Figure 6 (right) ranks 3 MTOs as the eval per-task weighting is varied. As the En→Zh weight is varied, the ranking shifts from MGDA being the best MTO to PCGrad being the best. This is a natural consequence of Figure 2 which shows that different MTOs find different points on the same Pareto front traced out by a sampling sweep. Notably, no algorithm outperforms sampling with a well-chosen sampling ratio.

Alternative Approaches As discussed in Section 3, MTO algorithms often drastically increase the algorithmic and computational complexity of the training process. In our experiments, we observed that the requirement to compute per-task gradients (which is necessary for many MTO algorithms) led to a significant reduction in the number of training steps per second (from ≈ 12 to ≈ 5). Given these observations, it is natural to wonder if there are more effective ways to spend this extra compute budget. Figure 7 examines how scaling the model size changes the performance trade-off behavior. We examine increasing the model depth by a factor of $\{1, 2, 3, 4\}$. Our largest model achieves on average 5.4 training steps per second, which is comparable with the models trained using per-task

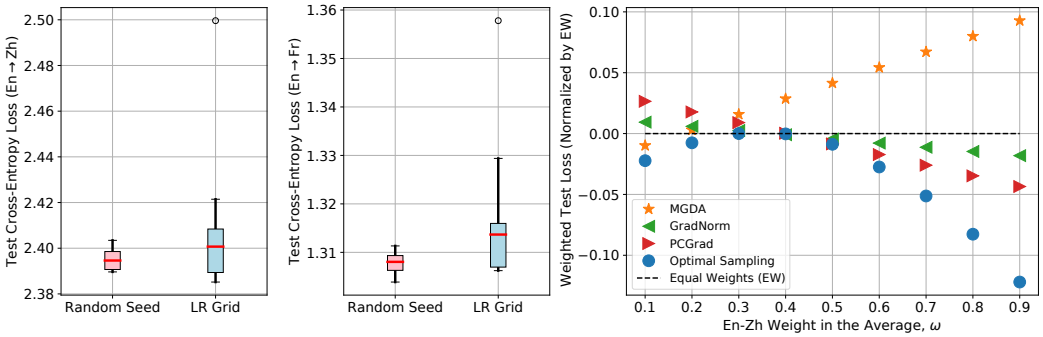

Figure 6: *Left:* Sparse sampling of learning rates has a significantly larger affect on reported performance than varying the random seed of a particular hyperparameter point. *Right:* Rankings between algorithms can depend on how tasks are weighted at evaluation time. For example, if the eval performance is ranked according to .9 En-Fr + .1 En-Zh, then MGDA is preferred to GradNorm and PCGrad. At equal weighting, GradNorm and PCGrad outperform MGDA. For any weighting, all MTOs underperform an optimally chosen sampling scheme.

gradients. Our results suggest that, unlike the observed behavior with MTO algorithms, allocating more compute to scale the model yields consistent improvements across the board. Larger models achieve Pareto fronts strictly to the lower-left of the base model, which corresponds to a performance improvement for all utility functions.

Table 2: Overview of models used in the scaling experiment.

Model	Optimizer	# Parameters	# Steps Per-Sec
3L-3L (base)	Scalarization	120M	11.52
3L-3L	MGDA	120M	4.81
6L-6L	Scalarization	142M	8.28
9L-9L	Scalarization	165M	6.68
12L-12L	Scalarization	187M	5.48

4.2 Benchmarks from the Literature

The observations of Section 4.1, run contrary to many recent influential studies proposing MTOs for multi-task models [4, 5, 8, 24, 21]. These papers often compare the performance of their proposed algorithm with traditional training strategies and report significant gains. In this section, we attempt to reproduce these results on a number of supervised-learning benchmarks. We present comparisons for Multi-MNIST [20] and CelebA [19] datasets in the main text.[3]

For these experiments, we closely follow the experimental setup and the publicly available code from [21]. We modified the code sparingly to address bugs, update deprecated libraries, and speed up the data loader. We perform an extensive grid search for learning rate, weight decay, and dropout. All models use early stopping. Our implementation details are presented in the appendix. [4]

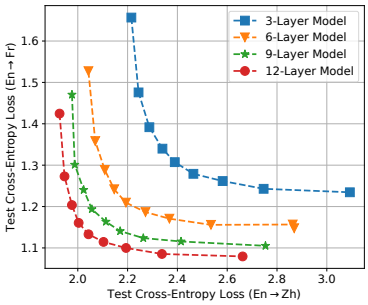

Figure 7: The effect of model size on the Pareto frontier for En→{Zh, Fr} models.

[3] See Appendix C for more comparisons and details.

[4] The code will be made available on GitHub after the review period.

4.2.1 Multi-MNIST

Multi-MNIST is MNIST cast in a multi-task setting by overlaying two random digits into the same image, one onto the right hand side and one onto left hand side. We treat the classification of each digit as its own task and follow the construction listed in Sener & Koltun (2019) [21]. We use a LeNet architecture and perform a hyper-parameter search through learning rate and dropout rate. All models are trained with early stopping. Figure 8 suggests the performance of MTO algorithms on this benchmark closely follows our observations from earlier sections.

4.2.2 CelebA

CelebA dataset [19] is a collection of 200K face images annotated with 40 attributes. This dataset provides a popular benchmark for MTO research where each attribute is treated as a separate binary classification task. In Section 4.1, we identified a number of evaluation challenges for MTO algorithms, namely the significance of exact hyper-parameter tuning and the difficulty of comparing models via average performances. These evaluation challenges become highly visible for CelebA.

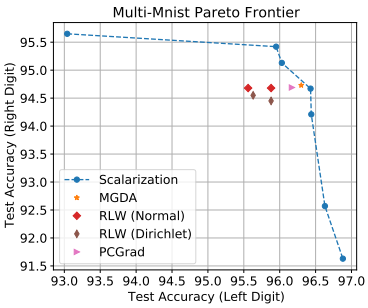

Figure 8: Test accuracy behavior on Multi-MNIST dataset aligns with our observation of Section 4.1.

Figure 9 presents the overview of our results. We report average performance across the tasks. Our results suggest that scalarization performance is comparable with the performance of popular MTO algorithms. This is inline with recent findings in the literature [13]. More importantly, Figure 9 shows the importance of careful hyper-parameter tuning: even in the presence of early-stopping, there is significant variation in the final performance of the models that is drastically larger than the effect of the MTO algorithm choice.

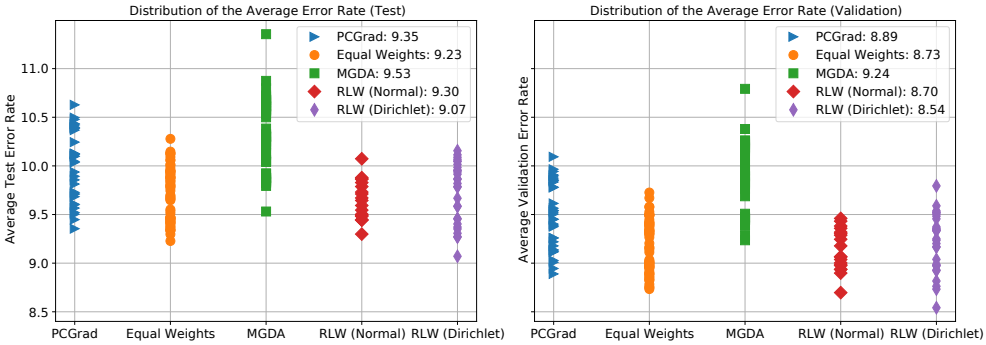

Figure 9: For CelebA, the effect of tuning hyper-parameters is much more significant than the effect of the MTO algorithm choice. Each point here corresponds to the performance of an early-stopped model. We vary the learning rate from 10^{-4} to 5×10^{-1} and weight decay from 0 to 5×10^{-3}.

This acute hyperparameter sensitivity can lead to a scenario where insufficient tuning of baseline hyper-parameters give illusions of significant performance gains. We suspect such evaluation challenges play a prominent role in the significant disagreements we observe in the literature on the effect & ranking of MTO algorithms. Table 3 presents an overview of the results presented in the literature. As the table suggests different papers report wildly different performance for the same algorithm. A large fraction of reported statistics resemble the quantities we observe on the *validation* dataset. This is concerning as validation performance on CelebA tends to be noisy. If only the validation performance is reported, a combination of early stopping and high evaluation frequency can artificially boost the scores. This artificial boost in scores is clearly visible between the left and the right sides of Figure 9.

Table 3: An overview of reported results in the literature for CelebA benchmark. There is significant disagreement between reported statistics from different studies. We suspect improper tuning of baseline hyper-parameters is a likely culprit (compare reported statistics with Figure 9).

Study	[5]	[21]	[24]	[18]	[13]	Ours (Test)	Ours (Validation)
Scalarization	8.71	9.62	–	9.99	9.1	9.23	8.73
MGDA	10.82	8.25	8.95	9.96	9.78	9.53	9.24
GradNorm	8.68	8.44	–	10.08	–	–	–
PCGrad	8.72	–	8.69	10.01	9.07	9.35	8.89
GradDrop	8.52	–	–	–	9.02	–	–

5 Conclusions

In this paper, we presented a large-scale empirical study examining the effects of multi-task optimization methods. It is often assumed that these algorithms enhance the optimization dynamics of multi-task models and yield desirable solutions that cannot be achieved via scalarization. Our results suggest the contrary. Across a variety of language and vision tasks, we showed that scalarization, with appropriate weights, can match both the optimization and the generalization behaviors of MTO algorithms. As such, in effect, scalarization solutions form a superset for MTO solutions. Our experimental results suggest effective exploration of the scalarization solution set might be a more reliable and effective strategy for boosting the model performance (See Fig 5). Examining strategies for efficiently searching this solution space (such as [9, 12]) is a fruitful future research direction.

Our observations suggest the final performance of multi-task models is highly sensitive to the choice of training hyper-parameters. Often times, the effect size associated with subtle design decisions in the choice of the hyper-parameter grid is orders of magnitude larger than the MTO effect size (See Fig 9). As such, researchers can unknowingly create the illusion of significant performance gains by simply under-tuning the competing baselines. The fact that different studies are reporting drastically different numbers for the same dataset-algorithm pair (Table 3) suggests this phenomenon is prevalent in this literature.

It is difficult to provide sufficient evidence that performance gains are due to inherent advantages offered by the algorithm and not due to inadequate tuning of the baseline. Some works attempt a fair comparison by using the same sparse tuning protocol for both their method and the baseline, however this will only bias evals towards methods which work well specifically with the sampled parameters. For example, suppose method A and method B have comparable performance, but the optimal learning rate for method A is $\eta^* = .05$ and for B is $\eta^* = .1$. In this case the "fair" comparison of running a learning rate sweep over values $\{10^{-3}, 10^{-2}, 10^{-1}\}$ will favor B over A, whereas a different sparse sweep will favor A. Common practice of reporting variance over random seed gives an illusion of significant gains when in reality it does not account for the much larger performance differences which arise from choices in the tuning protocol.

A potential solution to partially alleviate this problem is to adopt the Common Task Framework (CTF) [7, 16] to reliably identify and measure algorithmic improvements in multi-task optimization. The creation of a commonly used competitive benchmark with a proper validation / test split, baselines will naturally become stronger as subsequent papers find progressively —this makes substantial gains more convincing than current practice where baselines rerun by the authors themselves.

References

[1] Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*, 2019.

[2] Ankur Bapna, Colin Cherry, Yu Zhang, Ye Jia, Melvin Johnson, Yong Cheng, Simran Khanuja, Jason Riesa, and Alexis Conneau. mslam: Massively multilingual joint pre-training for speech and text. *arXiv preprint arXiv:2202.01374*, 2022.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR, 2018.

[5] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *Advances in Neural Information Processing Systems*, 33:2039–2050, 2020.

[6] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318, 2012.

[7] David Donoho. 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4):745–766, 2017.

[8] Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[9] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017.

[10] 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*, pages 7482–7491, 2018.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Julia Kreutzer, David Vilar, and Artem Sokolov. Bandits don't follow rules: Balancing multi-facet machine translation with multi-armed bandits. *arXiv preprint arXiv:2110.06997*, 2021.

[13] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M Pawan Kumar. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122*, 2022.

[14] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[15] Xian Li and Hongyu Gong. Robust optimization for multilingual translation with imbalanced data. *Advances in Neural Information Processing Systems*, 34, 2021.

[16] Mark Liberman. Obituary: Fred jelinek. *Computational Linguistics*, 36(4):595–599, 2010.

[17] Baijiong Lin, Feiyang Ye, and Yu Zhang. A closer look at loss weighting in multi-task learning. *arXiv preprint arXiv:2111.10603*, 2021.

[18] Liyang Liu, Yi Li, Zhanghui Kuang, J Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. ICLR, 2021.

[19] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[20] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

[21] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 525–536, 2018.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[23] Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *arXiv preprint arXiv:2010.05874*, 2020.

[24] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Sections 4.1
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [N/A]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See appendix for implementation details. The code will be made public after the review period

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figure 6

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Compute details are provided in the appendix

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.2.2

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]