Asynchronous Convergence in Multi-Task Learning via Knowledge Distillation from Converged Tasks

Weiyi Lu[1], **Sunny Rajagopalan**[2,*], **Priyanka Nigam**[1], **Jaspreet Singh**[1], **Xiaodi Sun**[1]
Yi Xu[1], **Belinda Zeng**[1], **Trishul Chilimbi**[1]
[1]Amazon, [2]Google
[1]{weiyilu,nigamp,jazsingh,xiaodisu,yxaamzn,zengb,trishulc}@amazon.com
[2]sunny.rg@gmail.com

Abstract

Multi-task learning (MTL) aims to solve multiple tasks jointly by sharing a base representation among them. This can lead to more efficient learning and better generalization, as compared to learning each task individually. However, one issue that often arises in MTL is the convergence speed between tasks varies due to differences in task difficulty, so it can be a challenge to simultaneously achieve the best performance on all tasks with a single model checkpoint. Various techniques have been proposed to address discrepancies in task convergence rate, including weighting the per-task losses and modifying task gradients. In this work, we propose a novel approach that avoids the problem of requiring all tasks to converge at the same rate, but rather allows for "asynchronous" convergence among the tasks where each task can converge on its own schedule. As our main contribution, we monitor per-task validation metrics and switch to a knowledge distillation loss once a task has converged instead of continuing to train on the true labels. This prevents the model from overfitting on converged tasks while it learns the remaining tasks. We evaluate the proposed method in two 5-task MTL setups consisting of internal e-commerce datasets. The results show that our method consistently outperforms existing loss weighting and gradient balancing approaches, achieving average improvements of 0.9% and 1.5% over the best performing baseline model in the two setups, respectively.

1 Introduction

Over the past few years, large pretrained models have achieved great success on a variety of tasks in natural language processing (Devlin et al., 2019; Yang et al., 2019; Raffel et al., 2020; Brown et al., 2020; Lewis et al., 2020; Clark et al., 2020; He et al., 2021). Most work in this area typically follows the pretraining-finetuning paradigm, in which the model is first pretrained on large text corpora using a self-supervised language modeling objective and then finetuned using supervised data from a target task. In particular, when evaluating on a benchmark that contains multiple tasks such as GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019), a common method is to finetune a separate model for each task in order to achieve the best performance. Although such an approach produces excellent results, it has several drawbacks, including incurring the costs of repeated finetuning efforts and precluding the possibility of knowledge sharing among related tasks. In addition, when it comes to deploying these models for real-world applications, a separate model is deployed for each individual task which can pose challenges for model hosting and maintenance.

An alternative approach is multi-task learning (MTL), which solves multiple tasks together by sharing representations among them. This not only offers benefits in computational and storage efficiency, but also makes it possible to share knowledge among related tasks and encourages the model to learn more robust and generalizable representations (Ruder, 2017; Zhang and Yang, 2017; Crawshaw, 2020). However, one of the biggest challenges in MTL is to balance the convergence schedule across tasks. Differences in task difficulty can result in faster convergence on some tasks over others. As a result, the naive approach which simply adds together the losses of all tasks is typically sub-optimal (Sener and Koltun, 2018; Liu et al., 2019a), since the final model may overfit the tasks that have converged early on during training, while underfitting the others. To tackle this, a large body of work has explored various loss and gradient balancing strategies (Kendall et al., 2018; Sener and Koltun, 2018; Chen et al., 2018; Liu et al., 2019a; Yu et al., 2020; Wang et al., 2021), in order to enforce the same convergence speed across tasks. Despite these efforts, the problem still remains un-

*Work done while at Amazon.

solved. As we show in our experiments, existing approaches can still fail to balance learning across different tasks in practice.

In this work, we propose a different approach for coordinating the learning across tasks in MTL. Instead of artificially forcing all tasks to converge at the same rate, we allow each task to converge according to its own schedule. We call this asynchronous convergence, as opposed to previous methods which seek to achieve a synchronous convergence schedule across tasks. After each task converges, we avoid overfitting by switching to a knowledge distillation loss for that task for the remaining training steps. The intuition is that continuing to train using the true labels can lead to overfitting. Instead, the knowledge distillation loss encourages the model to maintain its output distribution and thus its performance on the converged tasks at the best level, while the model learns the remaining tasks. We evaluate the proposed method in two different 5-task MTL setups. Our results show that existing loss and gradient balancing approaches fail to produce meaningful improvements over the simple baseline which sums the losses from all tasks or lead to more costly and less efficient training. In contrast, our method achieves consistent improvements. In particular, when comparing the final checkpoint performance, our best performing approach achieves an average improvement of 0.9% over the best baseline model in the first setup and 1.5% in the second setup.

2 Related Work

As already mentioned, one key challenge in MTL is to balance the learning speed across tasks. Some existing methods address this by applying static weights to the losses of different tasks (Kendall et al., 2015; Liao et al., 2016; Kokkinos, 2017), where the weights are usually determined through extensive hyper-parameter search. However, such an approach tends to be sub-optimal (Sener and Koltun, 2018). One line of research improves this by designing algorithms to automatically determine the weights and dynamically adjust them during training (Guo et al., 2018; Kendall et al., 2018; Liu et al., 2019a). Going beyond the loss weighting approaches, there is also work that leverages gradient information and proposes to manipulate the magnitude and/or direction of the gradients from different tasks in order to better coordinate the learning among the tasks (Sener and Koltun, 2018; Chen et al., 2018; Yu et al., 2020; Wang et al., 2021). Recently, Liu et al. (2021) shows improved results by combining loss weighting and gradient manipulation approaches. In all above methods, the goal is to enforce roughly the same convergence speed across tasks, so that the final model fits all tasks well. However, we hypothesize that imposing such an artificial constraint leads to optimization challenges. Instead, we propose a simpler method which allows for asynchronous convergence among the tasks, where each task converges on its own schedule. We focus on avoiding overfitting after a task has converged, which is achieved by distilling from the task's best checkpoint for the remaining training steps.

On the subject of maintaining the performance of converged tasks, a related research area is continual learning (CL) (Parisi et al., 2018; Lange et al., 2021). CL studies the problem of learning tasks in a sequential manner with the goal of avoiding catastrophic forgetting (Goodfellow et al., 2014) of previous tasks while learning new tasks. Replay-based CL approaches are most relevant to our work. The idea is to periodically present the model with examples from past tasks to avoid forgetting (Rebuffi et al., 2017; de Masson d'Autume et al., 2019; Sun et al., 2020a). In particular, Hou et al. (2018) proposes to avoid forgetting by adding a distillation loss formulated using a small subset of examples from previous tasks. In our case, we experiment with a similar setup where we sequentially add one task at a time while using a distillation loss to preserve performance on converged tasks. However, because our focus is on MTL where we have access to the data of all tasks throughout training, we do not down-sample the data of converged tasks. Additionally, our use of the distillation loss is to not only avoid catastrophic forgetting but also overfitting.

Finally, our work is also related to the field of knowledge distillation (KD) (Hinton et al., 2015; Gou et al., 2021), where the goal is to transfer the knowledge of one network to another by training the latter network to mimic the predictions of the former network. It is widely used to distill the knowledge of a large teacher model to a small student model (Hinton et al., 2015; Kim and Rush, 2016; Urban et al., 2017) but has also been applied in MTL (Liu et al., 2019b; Clark et al., 2019) and CL (Hou et al., 2018; Chuang et al., 2020). In particular, Clark et al. (2019) train a multi-task model by distilling from multiple single-task models and

show this outperforms directly training a multi-task model. The main difference between this approach and our work is that we do not require separate teacher models per-task, but rather use intermediate checkpoints as teachers as the model converges on each task. Another related work by Wei et al. (2019) also similarly uses intermediate checkpoints for distillation. However, their focus is on training a single machine translation model, whereas we use the technique to train a multi-task model.

3 Proposal: Asynchronous Convergence via Knowledge Distillation

Consider a MTL scenario where we have T tasks and have $\mathcal{D}_t = \left\{\left(x_t^{(i)}, y_t^{(i)}\right)\right\}_{i=1}^{N_t}$ as the training set of task t, with $t \in \{1, \ldots, T\}$ and N_t being the total number of training examples for task t. Let f represent a neural network with parameters θ. In standard supervised training, we would train the network on task t by minimizing a loss $\mathcal{L}_t^{\text{ST}}$ (where ST stands for supervised training) formulated as

$$\mathcal{L}_t^{\text{ST}}\left(\mathcal{D}_t; \theta\right) = \sum_{i=1}^{N_t} \ell_t\left(f\left(x_t^{(i)}; \theta\right), y_t^{(i)}\right), \quad (1)$$

where $f\left(x_t^{(i)}; \theta\right)$ denotes the prediction of the model, e.g. probability distribution over all classes for a classification task or predicted score for a regression task, and ℓ_t denotes the corresponding loss function, e.g. cross entropy for classification or mean squared error for regression. As discussed, the challenge of MTL lies in balancing the optimization of the losses across different tasks. In this work, we propose to simply minimize the sum of all task losses, except that after the model has converged on task t, we would change the task's loss from $\mathcal{L}_t^{\text{ST}}$ to a KD loss $\mathcal{L}_t^{\text{KD}}$ formulated against the best checkpoint of task t.

Specifically, let $\widehat{\theta}_t$ denote the parameters of the checkpoint when the model converges on task t. We first use the checkpoint to run inference on the task's training set $\mathcal{D}_t$ to obtain $\widehat{\mathcal{D}}_t = \left\{\left(x_t^{(i)}, \widehat{y}_t^{(i)}\right)\right\}_{i=1}^{N_t}$, where $\widehat{y}_t^{(i)} = f\left(x_t^{(i)}; \widehat{\theta}_t\right)$. Then for the remaining training steps, the model would be trained on task t using a KD loss $\mathcal{L}_t^{\text{KD}}$ formulated as

$$\mathcal{L}_t^{\text{KD}}\left(\widehat{\mathcal{D}}_t; \theta\right) = \sum_{i=1}^{N_t} \ell_t\left(f\left(x_t^{(i)}; \theta\right), \widehat{y}_t^{(i)}\right). \quad (2)$$

Essentially, after the model has converged on task t, we no longer train on the true labels of the task. Rather, we use the KD loss to encourage the model to mimic the predictions from the checkpoint where the best performance is achieved. As we show in our experiments, this method effectively maintains the model's performance on converged tasks without overfitting, while the model continues to learn the remaining tasks. To summarize, we propose to minimize the following loss

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t, \text{ where } \mathcal{L}_t = \begin{cases} \mathcal{L}_t^{\text{ST}}, & \text{if task } t \text{ has not converged;} \\ \mathcal{L}_t^{\text{KD}}, & \text{otherwise.} \end{cases} \quad (3)$$

The question of determining when a task has converged still remains. For this, we monitor the validation performance of each task given some patience n_t. If the performance does not improve for n_t consecutive validation steps, we consider the model to have converged on task t. However, one issue with this approach is that, when the patience runs out, we would have already trained the model for some extra steps, and the latest checkpoint could already overfit the task. To resolve this, we rewind back to the checkpoint when the best validation performance is achieved, and use that checkpoint as the best checkpoint $\widehat{\theta}_t$ to formulate $\mathcal{L}_t^{\text{KD}}$. We also rewind and resume training from that checkpoint, effectively discarding the latest steps.

Using this method, we experiment with two different training settings. The first is called **the joint setting**, which is similar to the conventional MTL setup. The model is trained on all tasks together, and we swap in the KD loss as the model converges on different tasks. Training stops when all tasks converge. The second setting is called **the sequential setting** and is inspired by the typical CL setup. Here we start training on a single task and then add one new task at a time after the previous task converges. Following our proposal, we use the KD loss for all converged tasks, while training the model on the true labels of the new task. The process continues until all tasks converge.

One additional hyper-parameter in the sequential setting is the order in which to train the tasks. We experiment with a few different orders, including ordering from (1) the smallest to the largest task by dataset size (**Sequential (Small)**), (2) the largest to the smallest dataset size (**Sequential (Large)**), and (3) the easiest to the hardest (**Sequential (Easy)**). For lack of a better heuristic, Sequential (Easy) uses the order in which the tasks converge in the

Task	Task Type	Size (Train / Val / Test)
Duplicate Detection (Dedup)	Binary classification	3M / 435K / 849K
Perceived Duplicates I (PD-I)	Binary classification	107K / 8K / 29K
Perceived Duplicates II (PD-II)	Binary classification	609K / 30K / 52K
Variations (Var)	Binary classification	5M / 598K / 612K
Unit of Measurement Identification (UoMI)	3-class classification	494K / - / 10K

Table 1: Datasets used in the Related 5-task setup.

Task	Task Type	Size (Train / Val / Test)
Relevant Attribute Identification (RAI)	1359-class multi-label classification	4M / 474K / 474K
Purchase Similarities (SIMS)	Regression	3M / 391K / 390K
Tariff Classification (TC)	93-class classification	28K / - / 3K
Product Type Classification (PTC)	709-class classification	957K / - / 106K
Duplicate Detection (Dedup)	Binary classification	3M / 435K / 849K

Table 2: Datasets used in the Diverse 5-task setup.

joint setting as a proxy for task difficulty ranking. There are potentially better strategies to determine the task order or even strategies that can make the performance invariant of task order. We leave such investigations as future work.

4 Experiments

4.1 Data

We evaluate on proprietary datasets from an e-commerce company. The datasets are in English and include text attributes of product listings, such as title and product description. We experiment with two different 5-task MTL setups. The tasks in the first setup are more similar to each other and are all some form of classification task, while the ones in the second setup are more diverse in terms of application and task type. We evaluate on these two benchmarks to test the effectiveness and robustness of our method in different MTL scenarios. A summary of the tasks used in the two setups is provided in Table 1 and 2, respectively.

The tasks in the first setup, referred to as **the Related 5-task setup**, include (1) **Dedup**, which classifies whether two product listings are duplicates of each other, (2) **PD-I**, which classifies whether two listings have subtle differences but may be perceived as duplicates in search results, (3) **PD-II**, which is the same as PD-I but considers a different set of attributes for defining perceived duplicates, (4) **Var**, which classifies whether two listings are variations of each other along certain set of dimensions (e.g. color, size, flavor, etc.), and (5) **UoMI**, which classifies the unit of measurement of a listing. Among the tasks, the first 4 are closely related, in that they all focus on classifying some sort of similarity between two listings.

The tasks in the second setup, called **the Diverse 5-task setup**, include (1) **RAI**, which classifies whether a listing has any of the 1359 pre-defined attributes, (2) **SIMS**, which predicts how similar two products are in customers' purchase decisions, (3) **TC**, which classifies the tariff category of a listing, (4) **PTC**, which classifies the product type of a listing, and (5) **Dedup**, which classifies whether two product listings are duplicates of each other.

Note that Dedup is used in both setups. Also, UoMI, TC, and PTC do not have separate validation sets, and therefore, we monitor the performance directly on their test sets during training.

4.2 Baselines

We compare our proposal against several baselines, including the naive uniform loss weighting approach, static and dynamic loss weighting methods, and a method which leverages gradient information. Specifically, the baseline models include (1) **Uni. Weight**: This is the uniform weighting baseline where we simply optimize the sum of all task losses. (2) **Muppet**: This is a static loss weighting method proposed in Aghajanyan et al. (2021), which uses a simple heuristic to compute a loss weight for each task such that the losses will roughly be on the same scale after applying the weights. (3) **DWA**: This is the Dynamic Weight Averaging (DWA) method proposed in Liu et al. (2019a), which automatically and dynamically computes the loss weights of different tasks during training. The motivation is to ensure roughly the same decreasing rate across all task losses. (4) **GradNorm**: This is proposed in Chen et al. (2018), which is a method to dynamically adjust the loss weights such that the gradients of different tasks have roughly the same magnitude. See Appendix A for implementation details and hyper-parameters.

4.3 Results

Table 3 shows the results in the Related 5-task setup. Among the baseline approaches, Muppet and DWA

Name	Train Steps	Dedup	PD-I	PD-II	Var	UoMI	Avg
Baselines							
Uni. Weight (Final)	8200	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00
Uni. Weight (Best)		+0.30	+1.00	+1.50	**+1.00**	+0.10	+0.78
Muppet	8200	+0.10	-0.10	+0.10	-0.20	-0.20	-0.06
DWA	8200	+0.10	-0.30	+0.10	-0.10	+0.00	-0.04
GradNorm	8200	-15.50	-15.50	-12.70	-4.80	-2.30	-10.16
This Work							
Joint	8200	**+0.60**	+1.10	**+1.60**	+0.90	-0.30	+0.78
Sequential (Small)	9000	+0.00	-1.20	+1.30	+0.60	+0.10	+0.16
Sequential (Large)	12000	+0.50	**+1.30**	+1.50	+0.90	**+0.30**	**+0.90**
Sequential (Easy)	11000	+0.30	+0.90	+1.50	+0.90	+0.10	+0.74

Table 3: Results in the Related 5-task setup. We report accuracy for UoMI and PRAUC for all other tasks. For all models, we report the performance of the final checkpoint, while for Uni. Weight, in addition to the final performance (Uni. Weight (Final)), we also report the performance using the respective best checkpoint for each task (Uni. Weight (Best)). All results are reported as changes over Uni. Weight (Final). The best performance of each task is in bold, while the second best is underlined.

Name	Train Steps	RAI	SIMS	TC	PTC	Dedup	Avg
Baselines							
Uni. Weight (Final)	10600	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00
Uni. Weight (Best)		+1.10	**+0.10**	**+1.10**	+1.00	+0.10	+0.68
Muppet	10600	-1.00	-0.10	+0.50	+0.70	+0.10	+0.04
DWA	10600	+0.80	+0.00	+0.30	+0.10	+0.00	+0.24
GradNorm	10600	-10.40	-2.10	-1.20	-1.00	-14.30	-5.80
This Work							
Joint	10600	+5.70	+0.00	+0.60	+1.10	-0.40	+1.40
Sequential (Small)	16800	+5.70	**+0.10**	+0.90	**+1.40**	**+0.50**	**+1.72**
Sequential (Large)	37200*	**+7.20**	-0.20	+0.30	+0.80	-2.10	+1.20
Sequential (Easy)	45800*	+4.00	-0.20	+0.50	+1.10	-1.10	+0.86

Table 4: Results in the Diverse 5-task setup. We report accuracy for RAI, TC, and PTC, and PRAUC for Dedup. For SIMS, its labels have been normalized to be between 0 and 1, and we report (1-RMSE)×100. For all models, we report the performance of the final checkpoint, while for Uni. Weight, in addition to the final performance (Uni. Weight (Final)), we also report the performance using the respective best checkpoint for each task (Uni. Weight (Best)). All results are reported as changes over Uni. Weight (Final). The best performance of each task is in bold, while the second best is underlined. *These experiments have a smaller learning rate. See text for more details.

produce similar results as those of the final checkpoint of Uni. Weight, while GradNorm produces much worse results than all other methods. The reason that GradNorm underperforms is because the model still underfits most tasks when training finishes, which suggests that the method is less efficient than others. Overall, there is a substantial gap between the performance of the best checkpoint of Uni. Weight and all other baseline methods, suggesting that none of the methods are able to effectively balance the learning across tasks.

In contrast, both our joint setting and sequential setting are able to achieve the best or second best results across tasks, which shows the effectiveness of our methods. In particular, the joint setting is able to match or surpass Uni. Weight (Best) on all tasks, except for UoMI. Among the experiments with the sequential setting, we can see that task order does impact performance. The exact ordering of tasks in different experiments are shown in Table 7 in Appendix B. Both ordering from the largest to the smallest task and ordering from the easiest to the hardest produce similar results overall, and are comparable with the joint setting and Uni. Weight (Best) in terms of average performance. On the other hand, ordering from the smallest to the largest task produces worst results overall. One possible explanation is that it is easier for the model to overfit the smaller tasks, which not only harms the performance on the tasks themselves, but also provides a sub-optimal initialization point for the later tasks. Finally, comparing the total training steps, we can see that the sequential setting generally takes longer to run, suggesting that the joint setting is a more efficient training method.

In addition to the results in Table 3, we also show the validation plots of different methods in Figure 1 in Appendix C. We can see that all baseline methods show signs of overfitting on some tasks (except for GradNorm which underfits). In contrast, the plots of both our joint setting and sequential setting do not show downward trend in any task, suggesting that our method is indeed effective in maintaining the performance of converged tasks at the best level while the model learns the remaining tasks.

Table 4 shows the results in the Diverse 5-task

setup. This time, the Uni. Weight baseline shows much less overfitting, as can be seen from the validation plot in Figure 2a in Appendix C. Nonetheless, there is still a substantial gap between Uni. Weight (Final) and Uni. Weight (Best) on RAI, TC, and PTC. Muppet is able to almost close the gap on PTC, but it produces worse result on RAI, while DWA almost closes the gap on RAI, but does not improve the other tasks. GradNorm still underfits all the tasks given the same training budget, again showing its lack of efficiency.

Compared to the baseline methods, our joint setting achieves a substantial performance boost on RAI, greatly outperforming even Uni. Weight (Best). It also matches the performance of Uni. Weight (Best) on PTC, and improves over Uni. Weight (Final) on TC. However, the performance on Dedup turns out to be worse. Meanwhile, Sequential (Small), i.e. our sequential setting that goes from the smallest to the largest task, also achieves the same substantial performance gain on RAI, and additionally outperforms or matches Uni. Weight (Best) across tasks. This again validates the effectiveness of our proposed technique in producing better MTL models. For the other two sequential setting experiments with different task orderings, we encountered some issues with training stability when running the experiments, and had to lower the learning rate to 10^{-5} , compared to 10^{-4} used in all other experiments. While fixing the stability issue, this likely prevented the optimizer from fully exploring the loss landscape, which resulted in worse performance on SIMS and Dedup in these two experiments. Nonetheless, they still outperform Uni. Weight (Final) on the other tasks, with Sequential (Large) in particular achieving the highest score on RAI. It is also interesting to note that Sequential (Small) produces the worst results in the Related 5-task setup among the three orders, but actually produces the best results in the Diverse 5-task setup. This could suggest that the effect of task ordering depends on the tasks used, and the optimal ordering strategy differs among the two setups. Another reason why Sequential (Small) outperforms in the Diverse 5-task setup could again be due to the sub-optimal learning rate which we had to use with the other two orders. In the future, we will continue to investigate the effects of task ordering, as well as tackle the training stability issue with smarter learning rate schedules. Also, we can see that the sequential setting again requires substantially more training steps than does the joint setting. This can also potentially be alleviated through a better learning rate schedule. Besides, we will also explore other techniques that can further improve the efficiency of our method, such as reducing the batch proportion of converged tasks.

Name	Train Steps	Dedup	PD-I	PD-II	Var	UoMI	Avg
Exp. 1							
Joint	8200	+0.60	+1.10	+1.60	+0.90	-0.30	+0.78
BAM (M→M)	16400*	+0.40	+1.80	+1.70	+1.00	+0.00	+0.98
Exp. 2							
Sequential (Small)	9000	+0.00	-1.20	+1.30	+0.60	+0.10	+0.16
Continual MTL	10200	+0.00	-0.70	-0.20	+0.50	-0.60	-0.20

Table 5: Results of two additional experiments on the Related 5-task setup. The first experiment compares our joint setting with BAM (M→M) (Clark et al., 2019), while the second experiment compares our sequential setting with Continual MTL (Sun et al., 2020b). All results are reported as changes over Uni. Weight (Final). *Training steps of BAM (M→M) include the training of Uni. Weight. See text for more details.

5 Additional Discussions

In this section, we provide more discussions on the effects of the design choices in both our joint setting and sequential setting. We illustrate the effects through two additional experiments on the Related 5-task setup.

In the first experiment, we compare our joint setting against an alternative approach where we take the respective best checkpoints for each task from the Uni. Weight baseline, and distill them together into a single multi-task model. We call this approach BAM (M→M) as it is the Multi→Multi strategy proposed in Clark et al. (2019). Through this experiment, we seek to compare our way of continued training with a mixture of KD and supervision from true labels against pure KD training. We note that our setting is more challenging as the model needs to learn all tasks from scratch, whereas BAM (M→M) directly transfers previously learned knowledge for each task to the model. Also, we train BAM (M→M) for the same number of training steps as that of our joint setting. However, since we need to obtain the best checkpoint for each task from Uni. Weight, which is also

trained for 8200 steps, the total training budget for BAM (M→M) is actually twice as large. The results of this experiment are shown in Table 5 under Exp. 1. We can see that BAM (M→M) has better performance on four out of the five tasks. Nonetheless, our joint setting achieves comparable average performance, despite the fact that our model needs to learn all tasks from scratch and that it receives only half of the training budget.

In the second experiment, we compare our sequential setting with the Continual MTL method proposed in Sun et al. (2020b). It is similar to our sequential setting in that it adds one new task at a time. The main difference is that they always use real labels for training, whereas we use KD to avoid overfitting on converged tasks. Through this experiment, we seek to understand the effects of the KD loss in our sequential setting. Specifically, we choose Sequential (Small) and train Continual MTL using the same task ordering. The results are shown under Exp. 2 in Table 5. We can see that Continual MTL has better performance on PD-I, but is much worse on PD-II and UoMI. Figure 3 in Appendix C shows the validation plots. It is clear that continual MTL suffers from overfitting on PD-II and UoMI, while our sequential setting does not show signs of overfitting. This again validates our assumption that the KD loss is effective in avoiding overfitting on converged tasks.

6 Conclusion

In this work, we propose a new approach to tackle the challenge of task convergence in MTL. In contrast to conventional loss and gradient balancing methods which attempt to enforce a synchronous convergence schedule, we allow the tasks to converge on asynchronous schedules and use a KD loss to maintain the performance on converged tasks while the model learns the remaining tasks. We show that the proposed method consistently outperforms existing loss and gradient balancing approaches. For future work, we will explore strategies to make model performance invariant of task ordering in the sequential setting, or alternatively, explore strategies to optimally determine task ordering. Additionally, we will investigate techniques to improve the efficiency of our method, such as dynamically adjusting the learning rate during training and the batch proportion of converged tasks.

7 Acknowledgments

We would like to thank the M5 Foundational Technologies team within Amazon Search for building the training infrastructure, which enabled the experiments in this work.

References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning (ICML)*.

Yung-Sung Chuang, Shang-Yu Su, and Yun-Nung Chen. 2020. Lifelong language knowledge distillation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019. Bam! born-again multi-task networks for natural language understanding. In *Association for Computational Linguistics (ACL)*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations (ICLR)*.

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey. In *ArXiv*.

Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. 2019. Episodic memory in lifelong language learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep

bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Ian J. Goodfellow, Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. 2014. An empirical investigation of catastrophic forgeting in gradient-based neural networks. In *ArXiv*.

Jianping Gou, B. Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. In *International Journal of Computer Vision*.

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. 2018. Dynamic task prioritization for multitask learning. In *European Conference on Computer Vision (ECCV)*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations (ICLR)*.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *ArXiv*.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. 2018. Lifelong learning via progressive distillation and retrospection. In *European Conference on Computer Vision (ECCV)*.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Alex Kendall, Matthew Koichi Grimes, and Roberto Cipolla. 2015. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *International Conference on Computer Vision (ICCV)*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Iasonas Kokkinos. 2017. 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 (CVPR)*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory G. Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Association for Computational Linguistics (ACL)*.

Yiyi Liao, Sarath Kodagoda, Yue Wang, Lei Shi, and Y. Liu. 2016. Understand scene categories by objects: A semantic regularized scene classifier using convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Liyang Liu, Yi Li, Zhanghui Kuang, Jing-Hao Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. 2021. Towards impartial multi-task learning. In *International Conference on Learning Representations (ICLR)*.

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

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. In *Association for Computational Linguistics (ACL)*.

German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2018. Continual lifelong learning with neural networks: A review. In *Neural Networks*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. In *Journal of Machine Learning Research (JMLR)*.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, G. Sperl, and Christoph H. Lampert. 2017. icarl: Incremental classifier and representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. In *ArXiv*.

Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Fan-Keng Sun, Cheng-Hao Ho, and Hung yi Lee. 2020a. Lamol: Language modeling for lifelong language learning. In *International Conference on Learning Representations (ICLR)*.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020b. Ernie 2.0:

A continual pre-training framework for language understanding. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Özlem Aslan, Shengjie Wang, Abdel rahman Mohamed, Matthai Philipose, Matthew Richardson, and Rich Caruana. 2017. Do deep convolutional nets really need to be deep and convolutional? In *International Conference on Learning Representations (ICLR)*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. 2021. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. In *International Conference on Learning Representations (ICLR)*.

Hao-Ran Wei, Shujian Huang, Ran Wang, Xinyu Dai, and Jiajun Chen. 2019. Online distilling from checkpoints for neural machine translation. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Yu Zhang and Qiang Yang. 2017. A survey on multi-task learning. In *IEEE Transactions on Knowledge and Data Engineering*.

Shuai Zheng, Haibin Lin, Sheng Zha, and Mu Li. 2020. Accelerated large batch optimization of bert pretraining in 54 minutes. In *ArXiv*.

A Implementation Details and Hyper-parameters

For all experiments, we use the same pretrained model, which has a BERT-like architecture (Devlin et al., 2019) and is pretrained using the masked language modeling objective on an internal English corpus consisting of online product listings. The vocabulary is trained on the same corpus using SentencePiece (Kudo and Richardson, 2018) and has 32K tokens. The model has 38 transformer layers, with each layer having 16 attention heads, 1024 hidden dimension, and 4098 intermediate feedforward dimension. The total parameter count is roughly 500M. The model is trained using the LANS optimizer (Zheng et al., 2020) with a batch size of 8192 and a learning rate of 10^{-4}. We had to use a smaller learning rate in two experiments with our sequential setting, which is discussed in Section 4.3. For each batch, we sample heterogeneously from all tasks, and the sampling distribution is roughly based on the dataset sizes, with some manual adjustments to ensure the smaller tasks are not too under-represented.

We validate every 200 training steps. For both our joint setting and sequential setting, we use a patience of 3 validation steps to determine whether a task has converged, and training stops when all tasks converge. For the baseline models, since we lack an aggregated early stopping criterion, for fair comparison, we train for the same number of steps as it takes to train the model in our joint setting. For Muppet, the loss weights for different tasks are shown in Table 6. For DWA, we set the temperature T to 2, which is recommended in Liu et al. (2019a). For GradNorm, we experiment with $\alpha \in \{0.5, 1, 2\}$ and choose the best performing value based on validation performance, which turns out to be 0.5 in the Related 5-task setup and 1 in the Diverse 5-task setup.

Task	**Dedup**	**PD-I**	**PD-II**	**Var**	**UoMI**
Weight	3.3	3.3	3.3	3.3	2.1
Task	**RAI**	**SIMS**	**HS**	**PTC**	**Dedup**
Weight	0.32	1	0.51	0.35	3.3

Table 6: Loss weights used in Muppet in both 5-task setups.

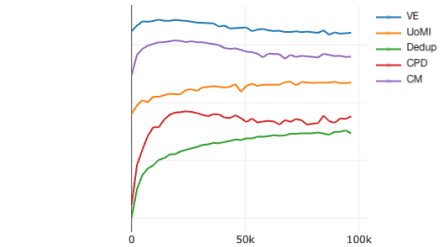
For all models, we report the performance of the final checkpoint on all tasks. For the Uni. Weight baseline, we additionally report the performance using the respective best checkpoint for each task, which can be used as a reference for the model's best performance on each task without overfitting.

	Task Order in the First 5-Task Setup
Sequential (Small)	PD-I → UoMI → PD-II → Dedup → Var
Sequential (Large)	Var → Dedup → PD-II → UoMI → PD-I
Sequential (Easy)	Var → PD-I → PD-II → UoMI → Dedup
	Task Order in the Second 5-Task Setup
Sequential (Small)	TC → PTC → Dedup → SIMS → RAI
Sequential (Large)	RAI → SIMS → Dedup → PTC → TC
Sequential (Easy)	PTC → SIMS → TC → Dedup → RAI

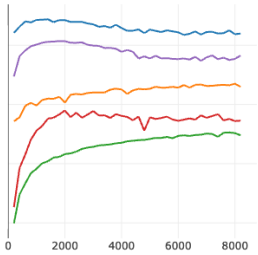
Table 7: Task order in the experiments with the sequential setting in both 5-task setups. Sequential (Small) orders the tasks from the smallest to the largest task by dataset size; Sequential (Large) orders from the largest to the smallest; Sequential (Easy) orders from the easiest to the hardest. For lack of a better heuristic, we use the order in which the tasks converge in the joint setting as a proxy for task difficulty ranking.

B Task Order in the Sequential Setting Experiments

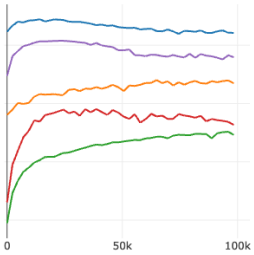
C Validation Plots



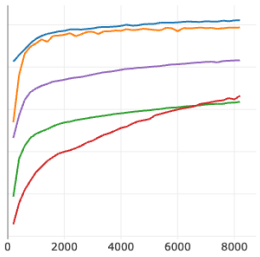
(a) Uni. Weight



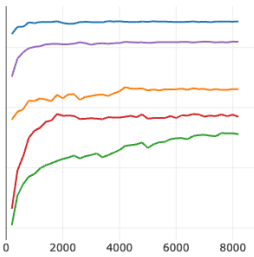
(b) Muppet



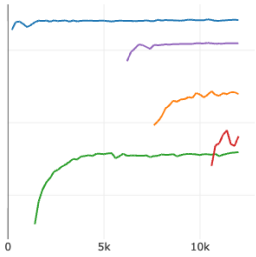
(c) DWA



(d) GradNorm

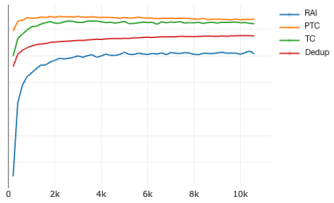


(e) Joint

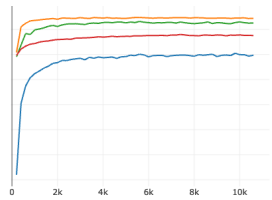


(f) Sequential (Large)

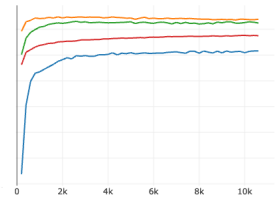
Figure 1: Validation plots of different methods in the Related 5-task setup. For the sequential setting, we only show the plot of Sequential (Large), as it has the best overall performance among different task orderings. Best viewed in color.



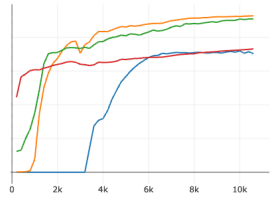
(a) Uni. Weight



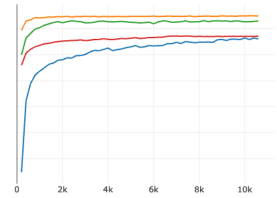
(b) Muppet



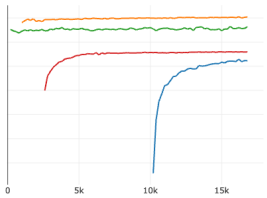
(c) DWA



(d) GradNorm

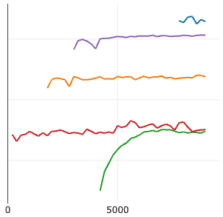


(e) Joint

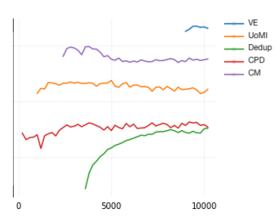


(f) Sequential (Small)

Figure 2: Validation plots of different methods in the Diverse 5-task setup. The plot of SIMS is omitted because its values are on a much smaller scale. For the sequential setting, we only show the plot of Sequential (Small), as it has the best overall performance among different task orderings. Best viewed in color.



(a) Sequential (Small)



(b) Continual MTL

Figure 3: Validation plots of Experiment 2 on the Related 5-task setup. Best viewed in color.