Auxiliary Learning by Implicit Differentiation

Aviv Navon[*]
Bar-Ilan University, Israel
aviv.navon@biu.ac.il

Idan Achituve[*]
Bar-Ilan University, Israel
achitui@cs.biu.ac.il

Haggai Maron
NVIDIA, Israel
hmaron@nvidia.com

Gal Chechik[†]
Bar-Ilan University, Israel
NVIDIA, Israel

Ethan Fetaya[†]
Bar-Ilan University, Israel
ethan.fetaya@biu.ac.il

Abstract

Training neural networks with auxiliary tasks is a common practice for improving the performance on a main task of interest. Two main challenges arise in this multi-task learning setting: (i) designing useful auxiliary tasks; and (ii) combining auxiliary tasks into a single coherent loss. Here, we propose a novel framework, *AuxiLearn*, that targets both challenges based on implicit differentiation. First, when useful auxiliaries are known, we propose learning a network that combines all losses into a single coherent objective function. This network can learn *non-linear* interactions between auxiliary tasks. Second, when no useful auxiliary task is known, we describe how to learn a network that generates a meaningful, novel auxiliary task. Evaluation of AuxiLearn in a series of tasks and domains, including image segmentation and learning with attributes in the low data regime, shows consistent improvement in accuracy compared to competing methods.

1 Introduction

The performance of deep neural networks can significantly improve by training the main task of interest with additional auxiliary tasks [20, 23, 36]. For example, learning to segment an image into objects can be more accurate when the model is simultaneously trained to predict other properties of the image like pixel depth or 3D structure [48]. In the low data regime, models trained with the main task only are prone to overfit and generalize poorly to unseen data [51]. In this case, the benefits of learning with multiple tasks are amplified [55]. Training with auxiliary tasks adds an inductive bias that pushes learned models to capture meaningful representations and avoid overfitting to spurious correlations.

Unfortunately, training with auxiliaries in practice is very challenging because of two main reasons. First, *combining and selecting auxiliaries*. In some domains, it may be easy to design beneficial auxiliary tasks and collect supervised data. For example, numerous tasks were proposed for self-supervised learning in image classification, including masking [11], rotation [19] and patch shuffling [12, 38]. In these cases, combining all auxiliary tasks into a single loss can be challenging [12]. The common practice is to compute a weighted combination of pretext losses by tuning the weights of individual losses using hyperparameter grid search. This approach limits the potential of learning with auxiliary tasks because the run time of grid search grows exponentially with the number of tasks.

The second major challenge is obtaining good auxiliaries in the first place. In many domains, it is not clear which auxiliary tasks could be beneficial. For example, for point cloud classification, few

[*]Equal contributor
[†]Equal contributor

self-supervised tasks have been proposed, and their benefits are limited so far [1, 21, 43, 49]. For some domains specialized expert knowledge may be needed for designing useful auxiliary tasks. For these cases, it would be beneficial to automate the process of generating auxiliary tasks without domain expertise.

Our work takes a step forward in automating the use and design of auxiliary learning. We present an approach to guide the learning of the main task with auxiliary learning, which we term *AuxiLearn*. AuxiLearn leverages recent progress made in implicit differentiation for optimizing hyperparameters [30, 34]. We show the effectiveness of AuxiLearn in two types of problems. First, in **combining auxiliaries**, for cases where auxiliary tasks are predefined, we propose to train a deep neural network (NN) on top of auxiliary losses and combine them non-linearly into a unified loss. For instance, we combine per-pixel losses in image segmentation tasks using a convolutional NN (CNN). Second, **designing auxiliaries**, for cases where predefined auxiliary tasks are not available, we present an approach for learning such tasks without domain knowledge and from input data alone. It is achieved by training an auxiliary network to output auxiliary labels while training another, primary, network to predict both the original task and the generated task. One important distinction from previous works, such as [25, 32], is that we do not optimize the auxiliary parameters using the training loss but on a separate (small) *auxiliary set*, allocated from the training data. This is a crucial point since the goal of auxiliary learning is improving generalization and not helping the optimization on the training data.

To validate our proposed solution, we perform an extensive experimental study on various tasks in the low data regime. In this case the models suffer from severe overfitting and auxilary learning can be of most use. Our results indicate that using AuxiLearn leads to improved loss functions and auxiliary tasks, in terms of the performance of the resulting model on the main task. We complement our experimental section with two interesting theoretical insights regarding our model. The first shows that a relatively simple auxiliary hypothesis class may overfit. The second tries to understand which auxiliaries could help with the main task.

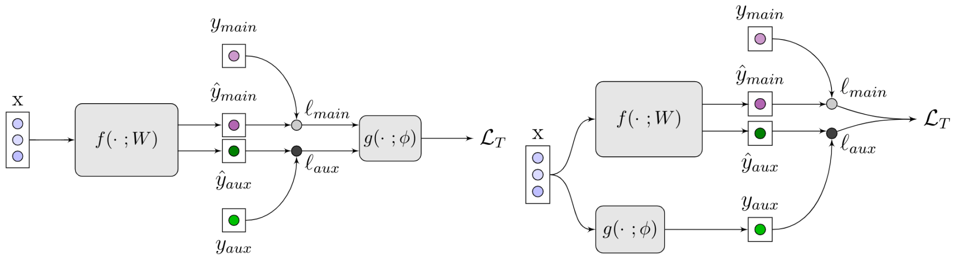
To summarize, we propose a novel general approach for learning with auxiliaries by utilizing implicit differentiation. We make the following novel contributions: (a) A unified approach for combining multiple loss terms in a flexible manner and for learning novel auxiliary tasks from the data alone; (b) A theoretical observation on auxiliary learning capacity; (c) We show that the key quantity for determining beneficial auxiliaries is the Newton update; (d) New results on a variety of auxiliary learning tasks with a focus on setups in which training data is limited. We conclude that implicit differentiation can play a significant role in automating the design of auxiliary learning setups.

2 Related work

Learning with multiple tasks. Multitask Learning (MTL) aims at simultaneously solving multiple learning problems while sharing information and knowledge across tasks. In some cases, MTL benefits the optimization process and improve task-specific generalization performance compared to single-task learning [48]. In contrast to MTL, auxiliary learning aims at solving a single, main task, and the purpose of all other tasks is to facilitate the learning of the primary task. At test time, only the main task is considered. This approach has been successfully applied in multiple domains, including computer vision [56], natural language processing [15, 50], and reinforcement learning [23, 31].

Dynamic task weighting. When learning a set of tasks, we assemble the overall loss using a combination of task-specific losses. The choice of proper loss blend is crucial, as MTL based models are susceptible to the relative task weights [25]. The most common approach for combining task losses is to use a linear combination. When the number of tasks is small, task weights are commonly tuned through simple grid search. This approach naturally cannot extend to a large number of tasks, or a more complex weighting scheme. Several recent studies proposed scaling task weights using gradient magnitude [7], task uncertainty [25], or the rate of change in losses [33]. [45] propose casting the multitask learning problem as Multi-Objective Optimization. These methods assume all tasks are equally important, hence they may not be suited for auxiliary learning. Both [13, 31] proposed to weight auxiliaries using gradient similarity. However, these methods do not scale well to multiple auxiliaries and do not take into account interactions between auxiliaries. In contrast, we propose to learn from data how to combine auxiliaries, possibly in a non-linear manner.

Devising auxiliaries. Designing an auxiliary task for a given main task is challenging because it may require domain expertise and additional labeling effort. For self-supervised learning (SSL), many



(a) Combining losses (b) Learning new auxiliary task

Figure 1: The AuxiLearn framework. **(a)** Learning to combine losses into a single coherent loss term. Here, the auxiliary network operates over a vector of losses. **(b)** Generating a novel auxiliary task. Here the auxiliary network operates over the input space. In both cases, $g(\cdot\,; \phi)$ is optimized using IFT based on $\mathcal{L}_A$.

approaches have been proposed (see [24] for a recent survey), but the joint representation learned through SSL may suffer from negative transfer and hurt the main task [48]. [32] proposed learning a helpful auxiliary in a meta-learning fashion, removing the need for handcrafted auxiliaries. However, their system is optimized on the training data, which leads to degenerate auxiliaries. To address this issue, an entropy term is introduced which forces the auxiliary network to spread the probability mass between the classes.

Implicit differentiation based optimization. Our formulation gives rise to a bi-level optimization problem. Such problems naturally arise in the context of meta-learning [16, 42] and hyperparameter optimization [5, 17, 29, 30, 34, 40]. The Implicit Function Theorem (IFT) is frequently used for computing the gradients of the upper-level function, which requires calculating a vector-inverse Hessian product. However, for modern neural networks, it is infeasible to calculate it explicitly, and an approximation must be devised. [35] proposed approximating the Hessian with the identity matrix, whereas [17, 40, 42] use Conjugate Gradient (CG) to approximate the product. Following [30, 34], we use a truncated Neumann series and efficient vector-Jacobian products, as it was empirically shown to be more stable than CG.

3 Our Method

We now describe the general AuxiLearn framework for jointly optimizing the primary network and the auxiliary parameters of the auxiliary network. First, we introduce our notations and formulate the general objective. Then, we detail two instances of this framework: combining auxiliaries and learning new auxiliaries. Finally, we present our optimization approach for optimizing these systems.

3.1 Problem definition

Let $\{(\mathbf{x}_i^t, \boldsymbol{y}_i^t)\}_i$ be the training set and $\{(\mathbf{x}_i^a, \boldsymbol{y}_i^a)\}_i$ be a distinct independent set which we term *auxiliary set*. Let $f(\cdot\,; W)$ denote the primary network, and let $g(\cdot\,; \phi)$ denote the auxiliary network. Here, W are the parameters of the model optimized on the training set, and ϕ are the auxiliary parameters trained on the auxiliary set. The training objective is defined as:

$$\mathcal{L}_T = \mathcal{L}_T(W, \phi) = \sum_i \ell_{main}(\mathbf{x}_i^t, \boldsymbol{y}_i^t; W) + h(\mathbf{x}_i^t, \boldsymbol{y}_i^t, W; \phi), \tag{1}$$

where ℓ_{main} denotes the loss of the main task and h is the overall auxiliary loss, controlled by ϕ. We note that h has access to both W and ϕ. The loss on the auxiliary set is defined as $\mathcal{L}_A = \sum_i \ell_{main}(\mathbf{x}_i^a, \boldsymbol{y}_i^a; W)$, since we are interested in the generalization performance of the main task.

We wish to find auxiliary parameters such that the parameters W, trained with the combined objective, generalize well. More formally, we seek

$$\phi^* = \arg\min_{\phi} \mathcal{L}_A(W^*(\phi)), \quad \text{s.t.} \quad W^*(\phi) = \arg\min_{W} \mathcal{L}_T(W, \phi). \tag{2}$$

3.2 Learning to combine auxiliary tasks

Suppose we are given K auxiliary tasks, usually designed using expert domain knowledge. We wish to learn how to optimally leverage these auxiliaries by learning to combine their corresponding losses. Let $\boldsymbol{\ell}(\mathbf{x}, \boldsymbol{y}; W) = (\ell_{main}(\mathbf{x}, y^{main}; W), \ell_1(\mathbf{x}, y^1; W), ..., \ell_K(\mathbf{x}, y^K; W))$ denote a loss vector. We wish to learn an auxiliary network $g : \mathbb{R}^{K+1} \rightarrow \mathbb{R}$ over the losses that will be added to ℓ_{main} in order to output the training loss $\mathcal{L}_T = \ell_{main} + g(\boldsymbol{\ell}; \phi)$. Here, h from Eq. (1) is given by $h(\cdot\,; \phi) = g(\boldsymbol{\ell}; \phi)$.

Commonly, $g(\boldsymbol{\ell}; \phi)$ is a linear combination of losses, namely $g(\boldsymbol{\ell}; \phi) = \sum_j \phi_j \ell_j$, with positive weights $\phi_j \geq 0$ that are tuned by grid search. However, this method can only scale to a few auxiliaries, as the run time of grid search is exponential in the number of tasks. Our method can handle a large number of auxiliaries and easily extends to a more flexible formulation in which g parametrized by a deep NN. This general form allows us to capture complex interactions between tasks, and learn non-linear combinations of losses. See Figure 1a for illustration.

One way to view a non-linear combination of losses is as an adaptive linear weighting, where losses have a different set of weights for each datum. If the loss at point $\mathbf{x}$ is $\ell_{main}(\mathbf{x}, y^{main}) + g(\boldsymbol{\ell}(\mathbf{x}, \boldsymbol{y}))$ then the gradients are $\nabla_W \ell_{main}(\mathbf{x}, y^{main}) + \sum_j \frac{\partial g}{\partial \ell_j} \nabla_W \ell_j(\mathbf{x}, y^j)$. This is equivalent to an adaptive loss where the loss of datum $\mathbf{x}$ is $\ell_{main} + \sum_j \alpha_{j,\mathbf{x}} \ell_j$ and, $\alpha_{j,\mathbf{x}} = \frac{\partial g}{\partial \ell_j}$. This observation connects our approach to other studies that assign adaptive loss weighs (e.g., Du et al. [13], Liu et al. [33]).

Convolutional loss network. In certain problems there exist a spatial relation among losses. For example, semantic segmentation and depth estimation for images. The common approach is to average the losses over all locations. We, on the other hand, can leverage this spatial relation for creating a *loss-image* in which each task forms a channel of pixel-losses induced by the task. We then parametrize g as a CNN that acts on this loss-image. As a result, we learn a spatial-aware loss function that captures interactions between task losses. See loss image example in Appendix C.2.

Monotonicity. It is common to parametrize the function $g(\boldsymbol{\ell}; \phi)$ as a linear combination with non-negative weights. Under this parametrization, g is a monotonic non-decreasing function of the losses. A natural question that arises is whether we should generalize this behavior and constrained $g(\boldsymbol{\ell}; \phi)$ to be non-decreasing w.r.t. the input losses as well? Empirically, we found that training with monotonic non-decreasing networks tends to be more stable and has a better or equivalent performance. We impose monotonicity during training by negative weights clipping. See Appendix C.3 for a detailed discussion and empirical comparison to non-monotonic networks.

3.3 Learning new auxiliary tasks

The previous subsection focused on situations where auxiliary tasks are given. In many cases, however, no useful auxiliary tasks are known in advance, and we are only presented with the main task. We now describe how to utilize AuxiLearn in such cases. The intuition is simple: We wish to learn an auxiliary task that pushes the representation of the primary network to better generalize on the main task, as measured by the auxiliary set. We do so in a student-teacher manner: an auxiliary "teacher" network produces labels and the primary network tries to predict these labels as an auxiliary task. The auxiliary “teacher" network is trained to define helpful auxiliary for the main task.

More specifically for the case of auxiliary classification task, we learn a soft labeling function $g(\mathbf{x}; \phi)$ over the input samples $\mathbf{x}$. These labels are then treated as ground truth auxiliary labels provided to the main network $f(\mathbf{x}; W)$. See Figure 1b for illustration. During training, the primary network $f(\mathbf{x}; W)$ outputs two predictions, $\hat{y}_{main}$ for the main task and $\hat{y}_{aux}$ for the auxiliary task. The auxiliary network $g(\mathbf{x}; \phi)$ produces soft auxiliary labels y_{aux}, which we treat as “adaptive ground truth". We then compute the full training loss $\mathcal{L}_T = \ell_{main}(\hat{y}_{main}, y_{main}) + \ell_{aux}(\hat{y}_{aux}, y_{aux})$ to update W. As before, we update ϕ using the auxiliary set with $\mathcal{L}_A = \ell_{main}$. Here, h from Eq. (1) is given by $h(\cdot\,; \phi) = \ell_{aux}(f(\mathbf{x}_i^t; W), g(\mathbf{x}_i^t; \phi))$. Intuitively, the teacher auxiliary network g is rewarded when it provides labels to the student that help it succeed in the main task, as measured by $\mathcal{L}_A$.

3.4 Optimizing auxiliary parameters

We now return to the bi-level optimization problem in Eq. (2) and present the optimizing method for ϕ. Solving Eq. (2) for ϕ poses a problem due to the indirect dependence of $\mathcal{L}_A$ on the auxiliary parameters. To compute the gradients of ϕ, we need to differentiate through the optimization process over W, since $\nabla_\phi \mathcal{L}_A = \nabla_W \mathcal{L}_A \cdot \nabla_\phi W^*$. As in [30, 34], we use the implicit function theorem (IFT) to evaluate $\nabla_\phi W^*$:

$$\nabla_\phi W^* = -\underbrace{(\nabla^2_W \mathcal{L}_T)^{-1}}_{|W|\times|W|} \cdot \underbrace{\nabla_\phi \nabla_W \mathcal{L}_T}_{|W|\times|\phi|}. \tag{3}$$

Thus, we can leverage the IFT to approximate the gradients of the auxiliary parameters ϕ:

$$\nabla_\phi \mathcal{L}_A(W^*(\phi)) = -\underbrace{\nabla_W \mathcal{L}_A}_{1\times|W|} \cdot \underbrace{(\nabla^2_W \mathcal{L}_T)^{-1}}_{|W|\times|W|} \cdot \underbrace{\nabla_\phi \nabla_W \mathcal{L}_T}_{|W|\times|\phi|}. \tag{4}$$

See Appendix A for detailed derivation. To compute the vector and Hessian inverse product, we utilize the algorithm proposed in [34], which uses Neumann approximation and efficient Vector-Jacobian Product. We note that accurately computing $\nabla_\phi \mathcal{L}_A$ by IFT requires finding a point such that $\nabla_W \mathcal{L}_T = 0$. In practice, we only approximate W^*, and simultaneously train both W and ϕ by altering between optimizing W on $\mathcal{L}_T$, and optimizing ϕ using $\mathcal{L}_A$. We summarize our method in Alg. 1 and 2.

Algorithm 1: AuxiLearn

```
Initialize auxiliary parameters $\phi$ and weights $W$;
while not converged do
    for $k = 1, ..., N$ do
        $\mathcal{L}_T = \ell_{main}(\mathbf{x}, y; W) + h(\mathbf{x}, y, W; \phi)$
        $W \leftarrow W - \alpha \nabla_W \mathcal{L}_T \big|_{\phi, W}$
    end
    $\phi \leftarrow \phi -$ Hypergradient$(\mathcal{L}_A, \mathcal{L}_T, \phi, W)$
end
return $W$
```

Algorithm 2: Hypergradient

```
Input: training loss $\mathcal{L}_T$, auxiliary loss $\mathcal{L}_A$, a
       fixed point $(\phi', W^*)$, number of
       iterations $J$, learning rate $\alpha$
$v = p = \nabla_W \mathcal{L}_A|_{(\phi', W^*)}$
for $j = 1, ..., J$ do
    $v -= \alpha v \cdot \nabla_W \nabla_W \mathcal{L}_T$
    $p += v$
end
return $-p \nabla_\phi \nabla_W \mathcal{L}_T|_{(\phi', W^*)}$
```

4 Analysis

4.1 Complexity of auxiliary hypothesis space

In our learning setup, an additional auxiliary set is used for tuning a large set of auxiliary parameters. A natural question arises: could the auxiliary parameters overfit this auxiliary set? and what is the complexity of the auxiliary hypothesis space $\mathcal{H}_\phi$? Analysing the complexity of this space is difficult, as it is coupled with the hypothesis space $\mathcal{H}_W$ of the main model. One can think of this hypothesis space as a subset of the original model hypothesis space $\mathcal{H}_\phi = \{h_W : \exists \phi \text{ s.t. } W = \arg\min_W \mathcal{L}_T(W, \phi)\} \subset \mathcal{H}_W$. Due to the coupling with $\mathcal{H}_W$ the behaviour can be un-intuitive and we will show that even simple auxiliaries can have infinite VC dimension.

Example: Consider the following 1D hypothesis space for binary classification $\mathcal{H}_W = \{\lceil \cos(Wx) \rceil, W \in \mathbb{R}\}$, which has infinite VC-dimension. Let the main loss be the zero-one loss and the auxiliary loss be $h(\phi, W) = (\phi - W)^2$, namely, an L_2 regularization with a learned center. As the model hypothesis space $\mathcal{H}_W$ has infinite VC-dimension there exists training and auxiliary sets of any size that are shattered by $\mathcal{H}_W$. Therefore, for any labeling on the auxiliary and training sets we can let $\phi = \hat{\phi}$, the parameter that perfectly classifies both sets. We then have that $\hat{\phi}$ is the optimum of the training with this auxiliary loss and we get that $\mathcal{H}_\phi$ also has infinite VC-dimension.

This important example shows that even apparently simple looking auxiliary losses can overfit due to the interaction with the model hypothesis space. Thus, it motivates our use of a separate auxiliary set.

4.2 Analysing an auxiliary task effect

When designing or learning auxiliary tasks one important question we can try to investigate is what makes an auxiliary task useful? Consider the following loss with a single auxiliary $\mathcal{L}_T(W, \phi) = \sum_i \ell_{main}(\mathbf{x}^t_i, \boldsymbol{y}^t_i, W) + \phi \cdot \ell_{aux}(\mathbf{x}^t_i, \boldsymbol{y}^t_i, W)$. Here $h = \phi \cdot \ell_{aux}$. Assume $\phi = 0$ so we optimize W only on the standard main task loss. We could check if $\frac{d\mathcal{L}_A}{d\phi}|_{\phi=0} > 0$, namely would it help to add this auxiliary task?

Proposition 1. *Let* $\mathcal{L}_T(W,\phi)=\sum_i \ell_{main}(\mathbf{x}_i^t,\boldsymbol{y}_i^t,W)+\phi\cdot\ell_{aux}(\mathbf{x}_i^t,\boldsymbol{y}_i^t,W)$. *Suppose that* $\phi=0$ *and that the main task was trained until convergence. We have*

$$\frac{d\mathcal{L}_A(W^*(\phi))}{d\phi}\bigg|_{\phi=0}=-\langle\nabla_W\mathcal{L}_A^T,\nabla_W^2\mathcal{L}_T^{-1}\nabla_W\mathcal{L}_T\rangle, \tag{5}$$

i.e. the gradient with respect to the auxiliary weight is the inner product between the Newton methods update and the gradient of the loss on the auxiliary set.

Proof. In the general case, the following holds $\frac{d\mathcal{L}_A}{d\phi}=-\nabla_W\mathcal{L}_A(\nabla_W^2\mathcal{L}_T)^{-1}\nabla_\phi\nabla_W\mathcal{L}_T$. For linear combination, we have $\nabla_\phi\nabla_W\mathcal{L}_T=\sum_i\nabla_W\ell_{aux}(\mathbf{x}_i^t,\boldsymbol{y}_i^t)$. Since W is optimized till convergence of the main task we obtain $\nabla_\phi\nabla_W\mathcal{L}_T=\nabla_W\mathcal{L}_T$. $\square$

This simple result shows that the key quantity to observe is the Newton update, rather than the gradient whihch is often used [31, 13]. Intuitively, the Newton update is the important quantity because if $\Delta\phi$ is small then we are almost at the optimum and due to quadratic convergence a single Newton step is sufficient for approximate converging to the new optimum.

5 Experiments

We evaluate the AuxiLearn framework in a series of tasks of two types: combining given auxiliary tasks into a unified loss (Sections 5.1 - 5.3), and generating a new auxiliary task (Section 5.4). Further experiments and analysis of both modules are given in Appendix C. Throughout all experiments, we use an extra data split for the auxiliary set. Hence, we use four data sets: training set, validation set, test set, and auxiliary set. The samples for the auxiliary set are pre-allocated from the training set. To ensure a fair comparison, these samples are used as part of the training set by all competing methods. Effectively, this means we have a slightly smaller training set for optimizing the parameters W of the primary network. In all experiments, we report the mean performance (e.g., accuracy) along with the Standard Error of the Mean (SEM). Full implementation details of all experiments are given in Appendix B. We make our source code publicly available at https://github.com/AvivNavon/AuxiLearn.

Model variants. For learning to combine losses, we evaluated the following variants of auxiliary networks: **(1) Linear**: a convex linear combination between the loss terms, **(2) Linear neural network (Deep linear)**: A deep fully-connected NN with linear activations. **(3) Nonlinear**: A standard feed forward NN over the loss terms. For the segmentation task only, **(4) ConvNet**: A CNN over the loss-images. The expressive power of the deep linear network is equivalent to that of a 1-layer linear network. However, from an optimization perspective, it was shown that over-parameterization introduced by the network depth could stabilize and accelerate convergence [3, 44]. All variants are constrained to represent only monotone non-decreasing functions.

5.1 An illustrative example

We first present an illustrative example of how AuxiLearn changes the loss landscape and helps generalization in the presence of label noise and harmful tasks. Consider a regression problem with $y_{main}=\mathbf{w}^{\star T}\mathbf{x}+\epsilon_0$ and two auxiliary tasks. The first auxiliary is helpful, $y_1=\mathbf{w}^{\star T}\mathbf{x}+\epsilon_1$, whereas the second auxiliary is harmful $y_2=\tilde{\mathbf{w}}^T\mathbf{x}+\epsilon_2$, $(\tilde{\mathbf{w}}\neq\mathbf{w}^\star)$. We let $\epsilon_0\sim\mathcal{N}(0,\sigma_{main}^2)$ and $\epsilon_1,\epsilon_2\sim\mathcal{N}(0,\sigma_{aux}^2)$. We optimize a linear model with weights $\mathbf{w}\in\mathbb{R}^2$ that are shared across tasks, i.e., no task-specific parameters. We set $\mathbf{w}^\star=(1,1)^T$ and $\tilde{\mathbf{w}}=(2,-4)^T$. We train an auxiliary network to output linear task weights and observe the changes to the loss landscape in Figure 2. The left plot shows the loss landscape for the main task, with a training set optimal solution $\mathbf{w}_{train}$. Note that $\mathbf{w}_{train}\neq\mathbf{w}^*$ due to the noise in the training data. The loss landscape of the weighted train loss at the beginning ($t=0$) and the end ($t=T$) of training is shown in the middle

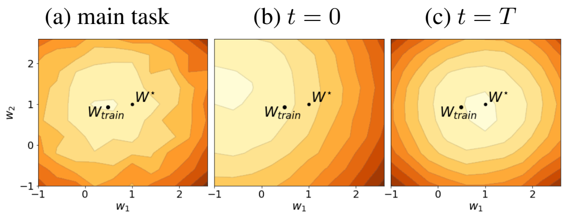


Figure 2: Loss landscape generated by the auxiliary network. Darker is higher. See text for details.

and right plots, respectively. Note how AuxiLearn learns to ignore the harmful auxiliary and use the helpful one to find a better solution by changing the loss landscape. In Appendix C.4 we extend this experiment and show that the auxiliary task weight is inversely proportional to the label noise.

5.2 Fine-grained classification with many auxiliary tasks

In tasks of fine-grain visual classification, annotating images requires that annotators are domain experts, and this makes data labeling challenging and expensive (e.g., in the medical domain). In some cases, however, non-experts can annotate predictive visual attributes. As an example, consider the case of recognizing bird species, which would require an ornithologist, yet a layman can describe the head color or bill shape of a bird. These features naturally form auxiliary tasks, which can be leveraged for training concurrently with the main task of bird classification.

We evaluate AuxiLearn in this setup on the fine-grained classification dataset, Caltech-UCSD Birds (CUB) 200-2011 [52]. The CUB dataset contains 200 bird species in 11,788 images, each is associated with a set of 312 binary visual attributes, which we use as auxiliaries. Since we are interested in setups where optimization based on the main task alone does not generalize well, we demonstrate our method in a semi-supervised setting: we assume auxiliary labels for all images but only 5 and 10 labels per class of the main task (noted as 5-shot and 10-shot).

Table 1: Test classification accuracy results on CUB 200-2011 dataset, averaged over three runs (± SEM)

	5-shot		10-shot	
	Top 1	Top 3	Top 1	Top 3
STL	35.50 ± 0.7	54.79 ± 0.7	54.79 ± 0.3	74.00 ± 0.1
Equal	41.47 ± 0.4	62.62 ± 0.4	55.36 ± 0.3	75.51 ± 0.4
Uncertainty	35.22 ± 0.3	54.99 ± 0.7	53.75 ± 0.6	73.25 ± 0.3
DWA	41.82 ± 0.1	62.91 ± 0.4	54.90 ± 0.3	75.74 ± 0.3
GradNorm	41.49 ± 0.4	63.12 ± 0.4	55.23 ± 0.1	75.62 ± 0.3
GCS	42.57 ± 0.7	62.60 ± 0.1	55.65 ± 0.2	75.71 ± 0.1
AuxiLearn				
Linear	41.71 ± 0.4	63.73 ± 0.6	54.77 ± 0.2	75.51 ± 0.7
Deep Linear	45.84 ± 0.3	66.21 ± 0.5	57.08 ± 0.2	75.3 ± 0.6
Nonlinear	**47.07 ± 0.1**	**68.25 ± 0.3**	**59.04 ± 0.2**	**78.08 ± 0.2**

We compare AuxiLearn the following MTL and auxiliary learning baselines: **(1) Single-task learning (STL):** Training only on the main task. **(2) Equal:** Standard multitask learning with equal weights to all auxiliary tasks. **(3) GradNorm:** [7], an MTL method that scales losses based on gradient magnitude. **(4) Uncertainty:** [25], an MTL approach that uses task uncertainty to adjust task weights. **(5) Gradient Cosine Similarity (GCS):** [13], an auxiliary-learning approach that uses gradient similarity between the main and auxiliary tasks. **(6) Dynamic weight averaging (DWA):** [33], an MTL approach that sets task weights based on the rate of loss change over time. The primary network in all experiments is ResNet-18 [22] pre-trained on ImageNet. We use a 5-layer fully connected NN for the auxiliary network. Sensitivity analysis of the network size and auxiliary set size is presented in Appendix C.5

Table 1 shows the test set classification accuracy. Most methods significantly improve upon the STL baseline, highlighting the benefits of using additional (weak) labels. Our Nonlinear and Deep linear auxiliary network variants outperform all previous approaches by a large margin. As expected, a non-linear auxiliary network is better than its linear counterparts. This suggests that there are some non-linear interactions between the loss terms that the non-linear network is able to capture. Also, notice the effect of using deep-linear compared to a (shallow) linear model. This result is an indication that at least part of the improvement achieved by our method is attributed to the over-parametrization of the auxiliary network. In the Appendix we further analyze the auxiliary networks. In Appendix C.6 we visualize the full optimization path of a linear auxiliary network over a polynomial kernel on the losses, and in Appendix C.7 we show that the last state of the auxiliary network is not informative enough. From both experiments, we conclude that the training dynamics and coupling between auxiliary and primary parameters is crucial.

5.3 Pixel-wise losses

We consider the indoor-scene segmentation task provided in [9], that utilize the NYUv2 dataset [46]. We use the 13-class semantic segmentation as the main task, with depth prediction and surface-normal estimation [14] as auxiliaries. Here, we use a SegNet [4] based model with 44M parameters for the primary network, and a 4-layer CNN for the auxiliary network.

In this task, since the losses are given at the pixel level, we can apply the ConvNet variant of the auxiliary network to the loss image, in which each task forms a channel. Table 2 reports the mean

Table 2: Test results for semantic segmentation on NYUv2, averaged over four runs (± SEM).

	mIoU	Pixel acc.
STL	18.90 ± 0.21	54.74 ± 0.94
Equal	19.20 ± 0.19	55.37 ± 1.00
Uncertainty	19.34 ± 0.18	55.70 ± 0.79
DWA	19.38 ± 0.14	55.37 ± 0.35
GradNorm	19.52 ± 0.21	56.70 ± 0.33
MGDA	19.53 ± 0.35	56.28 ± 0.46
GCS	19.94 ± 0.13	56.58 ± 0.81
AuxiLearn (ours)		
Linear	20.04 ± 0.38	$\mathbf{56.80 \pm 0.14}$
Deep Linear	19.94 ± 0.12	56.45 ± 0.79
Nonlinear	20.09 ± 0.34	$\mathbf{56.80 \pm 0.53}$
ConvNet	$\mathbf{20.54 \pm 0.30}$	56.69 ± 0.44

Intersection over Union (mIoU) and pixel accuracy for the main segmentation task. Here, we also compare with MGDA [45] which had extremely long training time in CUB experiments due to the large number of auxiliary tasks, and therefore wasn't compared against in Section 5.2. All weighting methods achieve a performance gain over the STL model. However, the comparison shows that the ConvNet variant of our auxiliary network outperforms all competitors in terms of test mIoU.

5.4 Learning an auxiliary classifier

Table 3: Learning auxiliary task. Test accuracy averaged over three runs (±SEM) without pre-training.

	CIFAR10 (5%)	CIFAR100 (5%)	SVHN (5%)	CUB (30-shot)	Pet (30-shot)	Cars (30-shot)
STL	50.8 ± 0.8	19.8 ± 0.7	72.9 ± 0.3	37.2 ± 0.8	26.1 ± 0.5	59.2 ± 0.4
MAXL-F	56.1 ± 0.1	20.4 ± 0.6	75.4 ± 0.3	39.6 ± 1.3	26.2 ± 0.3	59.6 ± 1.1
MAXL	58.2 ± 0.3	21.0 ± 0.4	75.5 ± 0.4	40.7 ± 0.6	26.3 ± 0.6	60.4 ± 0.8
AuxiLearn	$\mathbf{60.7 \pm 1.3}$	$\mathbf{21.5 \pm 0.3}$	$\mathbf{76.4 \pm 0.2}$	$\mathbf{44.5 \pm 0.3}$	$\mathbf{37.0 \pm 0.6}$	$\mathbf{64.4 \pm 0.3}$

In many cases, designing helpful auxiliaries is challenging. Here, we evaluate AuxiLearn in learning a multi-class classification auxiliary tasks. We do so on the following three multi-class classification datasets: CIFAR10, CIFAR100 [28], SVHN [37], and three fine-grain classification datasets: CUB-200-2011 [52], Oxford-IIIT Pet [39], and Cars [27]. Oxford-IIIT Pet contains 7349 images of 37 species of dogs and cats, and Cars contains 16,185 images of 196 cars. The official splits were used in all datasets.

Following [32], we learn a different auxiliary task for each class in the main task. Intuitively, this allows us to uncover hierarchies within each main task class. We set the number of classes to be learned to 5 for all experiments and all learned tasks. To examine the effect of learnable auxiliary in the low-data regime, we evaluate the performance while training with only 5% of the training set in CIFAR10, CIFAR100 and SVHN datasets, and ~ 30 samples per class in CUB, Oxford-IIIT Pet and Cars. We use VGG-16 [47] as the backbone for both CIFAR datasets, a 4-layers ConvNet for the SVHN experiment, and ResNet18 for the fine-grain datasets. In all experiments, the architectures of the auxiliary and primary networks was set the same, and the networks were trained from scratch without pre-training.

We compared our approach against the following baselines: **(1) Single-task learning (STL):** Training the main task only. **(2) MAXL:** Meta AuXiliary Learning (MAXL) recently proposed by [32] for learning auxiliary tasks. MAXL optimizes the label generator in a meta learning fashion. **(3) MAXL-F:** A frozen MAXL label generator, that is initialized randomly. It decouples the effect of having a teacher network from the additional effect brought by the training process.

Table 3 shows that AuxiLearn outperforms all baselines in all setups, even while sacrificing some of the training set for the auxiliary set. It is also worth noting that our optimization approach significantly improves upon MAXL, yielding $\times 3$ improvement in the run-time. In Appendix C.10 we present a 2D t-SNE projection of the soft labels learned by the auxiliary network for CIFAR10. Also, in Appendix C.11 and C.12 we show additional experiments of this setup, including an extension of the method to point-cloud part segmentation task.

6 Discussion

In this paper, we presented a novel and unified approach for learning how to combine and how to learn auxiliary tasks. We theoretically showed which auxiliaries can be beneficial and the importance of using a separate auxiliary set. We empirically showed that our method achieve significant improvement over existing methods on various datasets and tasks. This work opens interesting directions for future research. First, when training deep linear auxiliary networks, we observed similar learning dynamics to those of non-linear models. As a result, they generated better performance compared to their linear counterparts. This effect was observe in standard training setup, but the optimization path in auxiliary networks is very different. Second, we find that shifting labeled data from the training set to an auxiliary set is consistently helpful. A broader question remains about the most efficient way to do this allocation.

References

[1] Achituve, I., Maron, H., and Chechik, G. (2020). Self-supervised learning for domain adaptation on point-clouds. *arXiv preprint arXiv:2003.12641*.

[2] Alemi, A. A., Fischer, I., Dillon, J. V., and Murphy, K. (2017). Deep variational information bottleneck. In *International Conference on Learning Representations*.

[3] Arora, S., Cohen, N., and Hazan, E. (2018). On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*.

[4] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.

[5] Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900.

[6] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

[7] Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. (2018). Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR.

[8] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.

[9] Couprie, C., Farabet, C., Najman, L., and LeCun, Y. (2013). Indoor semantic segmentation using depth information. In *International Conference on Learning Representations*.

[10] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 248–255.

[11] Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430.

[12] Doersch, C. and Zisserman, A. (2017). Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060.

[13] Du, Y., Czarnecki, W. M., Jayakumar, S. M., Pascanu, R., and Lakshminarayanan, B. (2018). Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*.

[14] Eigen, D. and Fergus, R. (2015). Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658.

[15] Fan, X., Monti, E., Mathias, L., and Dreyer, M. (2017). Transfer learning for neural semantic parsing. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 48–56.

[16] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135.

[17] Foo, C.-s., Do, C. B., and Ng, A. Y. (2008). Efficient multiple hyperparameter learning for log-linear models. In *Advances in neural information processing systems*, pages 377–384.

[18] Ganin, Y. and Lempitsky, V. (2015). Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*.

[19] Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*.

[20] Goyal, P., Mahajan, D., Gupta, A., and Misra, I. (2019). Scaling and benchmarking self-supervised visual representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[21] Hassani, K. and Haley, M. (2019). Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8160–8171.

[22] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[23] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

[24] Jing, L. and Tian, Y. (2020). Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[25] Kendall, A., Gal, Y., and Cipolla, R. (2018). 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.

[26] Kingma, D. P. and Ba, J. (2014). ADAM: A method for stochastic optimization. In *International Conference on Learning Representations*.

[27] Krause, J., Stark, M., Deng, J., and Fei-Fei, L. (2013). 3D object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition*, Sydney, Australia.

[28] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

[29] Larsen, J., Hansen, L. K., Svarer, C., and Ohlsson, M. (1996). Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing VI. Proceedings of the IEEE Signal Processing Society Workshop*, pages 62–71. IEEE.

[30] Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., and Zemel, R. (2018). Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*.

[31] Lin, X., Baweja, H., Kantor, G., and Held, D. (2019). Adaptive auxiliary task weighting for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4773–4784.

[32] Liu, S., Davison, A., and Johns, E. (2019a). Self-supervised generalisation with meta auxiliary learning. In *Advances in Neural Information Processing Systems*, pages 1677–1687.

[33] Liu, S., Johns, E., and Davison, A. J. (2019b). End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880.

[34] Lorraine, J., Vicol, P., and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR.

[35] Luketina, J., Berglund, M., Greff, K., and Raiko, T. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, pages 2952–2960.

[36] Mirowski, P. (2019). Learning to navigate. In *1st International Workshop on Multimodal Understanding and Learning for Embodied Applications*, pages 25–25.

[37] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

[38] Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *Proceedings of the European Conference on Computer Vision*, pages 69–84. Springer.

[39] Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. (2012). Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[40] Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*, pages 737–746.

[41] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

[42] Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, pages 113–124.

[43] Sauder, J. and Sievers, B. (2019). Self-supervised deep learning on point clouds by reconstructing space. In *Advances in Neural Information Processing Systems*, pages 12942–12952.

[44] Saxe, A. M., Mcclelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *In International Conference on Learning Representations*. Citeseer.

[45] Sener, O. and Koltun, V. (2018). Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pages 527–538.

[46] Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In *Proceedings of the European conference on computer vision*, pages 746–760. Springer.

[47] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[48] Standley, T., Zamir, A. R., Chen, D., Guibas, L., Malik, J., and Savarese, S. (2019). Which tasks should be learned together in multi-task learning? *arXiv preprint arXiv:1905.07553*.

[49] Tang, L., Chen, K., Wu, C., Hong, Y., Jia, K., and Yang, Z. (2020). Improving semantic analysis on point clouds via auxiliary supervision of local geometric priors. *arXiv preprint arXiv:2001.04803*.

[50] Trinh, T., Dai, A., Luong, T., and Le, Q. (2018). Learning longer-term dependencies in RNNs with auxiliary losses. In *International Conference on Machine Learning*, pages 4965–4974.

[51] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

[52] Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology.

[53] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, 38(5):1–12.

[54] Yi, L., Kim, V. G., Ceylan, D., Shen, I.-C., Yan, M., Su, H., Lu, C., Huang, Q., Sheffer, A., and Guibas, L. (2016). A scalable active framework for region annotation in 3D shape collections. *ACM Transactions on Graphics*, 35(6):1–12.

[55] Zhang, Y. and Yang, Q. (2017). A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*.

[56] Zhang, Z., Luo, P., Loy, C. C., and Tang, X. (2014). Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer.

Appendix: Auxiliary Learning by Implicit Differentiation

A Gradient derivation

We provide here the derivation of Eq. (4) in Section 3. One can look at the function $\nabla_W \mathcal{L}_T(W, \phi)$ around a certain local-minima point $(\hat{W}, \hat{\phi})$ and assume the Hessian $\nabla^2_W \mathcal{L}_T(\hat{W}, \hat{\phi})$ is positive-definite. At that point, we have $\nabla_W \mathcal{L}_T(\hat{W}, \hat{\phi}) = 0$. From the IFT, we have that locally around $(\hat{W}, \hat{\phi})$, there exists a smooth function $W^*(\phi)$ such that $\nabla_W \mathcal{L}_T(W, \phi) = 0$ iff $W = W^*(\phi)$. Since the function $\nabla_W \mathcal{L}_T(W^*(\phi), \phi)$ is constant and equal to zero, we have that its derivative w.r.t. ϕ is also zero. Taking the total derivative we obtain

$$0 = \nabla^2_W \mathcal{L}_T(W, \phi) \nabla_\phi W^*(\phi) + \nabla_\phi \nabla_W \mathcal{L}_T(W, \phi) \,. \tag{6}$$

Multiplying by $\nabla^2_W \mathcal{L}_T(W, \phi)^{-1}$ and reordering we obtain

$$\nabla_\phi W^*(\phi) = -\nabla^2_W \mathcal{L}_T(W, \phi)^{-1} \nabla_\phi \nabla_W \mathcal{L}_T(W, \phi) \,. \tag{7}$$

We can use this result to compute the gradients of the auxiliary set loss w.r.t ϕ

$$\nabla_\phi \mathcal{L}_A(W^*(\phi)) = \nabla_W \mathcal{L}_A \cdot \nabla_\phi W^*(\phi) = -\nabla_W \mathcal{L}_A \cdot (\nabla^2_W \mathcal{L}_T)^{-1} \cdot \nabla_\phi \nabla_W \mathcal{L}_T. \tag{8}$$

As discussed in the main text, fully optimizing W to convergence is too computationally expensive. Instead, we update ϕ once for every several update steps for W, as seen in Alg. 1. To compute the vector inverse-Hessian product, we use Alg. 2 that was proposed in [34].

B Experimental details

B.1 CUB 200-2011

Data. To examine the effect of varying training set sizes we use all 5994 predefined images for training according to the official split and, we split the predefined test set to 2897 samples for validation and 2897 for testing. All images were resized to 256×256 and Z-score normalized. During training, images were randomly cropped to 224 and flipped horizontally. Test images were centered cropped to 224. The same processing was applied in all fine-grain experiments.

Training details for baselines. We fine-tuned a ResNet-18 [22] pretrained on ImageNet [10] with a classification layer on top for all tasks. Because the scale of auxiliary losses differed from that of the main task, we multiplied each auxiliary loss, on all compared method, by the scaling factor $\tau = 0.1$. It was chosen based on a grid search over $\{0.1, 0.3, 0.6, 1.0\}$ using the *Equal* baseline. We applied grid search over the learning rates in $\{1e-3, 1e-4, 1e-5\}$ and the weight decay in $\{5e-3, 5e-4, 5e-5\}$. For DWA [33], we searched over the temperature in $\{0.5, 2, 5\}$ and for GradNorm [7], over α in $\{0.3, 0.8, 1.5\}$. The computational complexity of GSC [13] grows with the number of tasks. As a result, we were able to run this baseline only in a setup where there are two loss terms: the main and the sum of all auxiliary tasks. We ran each configuration with 3 different seeds for 100 epochs with ADAM optimizer [26] and used early stopping based on the validation set.

The auxiliary set and auxiliary network. In our experiments, we found that allocating as little as 20 samples from the training set for the auxiliary set and using a NN with 5 layers and 10 units in each layer yielded good performance for both deep linear and non-linear models. We found that our method was not sensitive to these design choices. We use skip connection between the main loss ℓ_{main} and the overall loss term and Softplus activation.

Optimization of the auxiliary network. In all variants of our method, the auxiliary network was optimized using SGD with 0.9 momentum. We applied grid search over the auxiliary network learning rate in $\{1e-2, 1e-3\}$ and weight decay in $\{1e-5, 5e-5\}$. The total training time of all methods was 3 hours on a 16GB Nvidia V100 GPU.

B.2 NYUv2

The data consists of 1449 RGB-D images, split into 795 train images and 654 test images. We further split the train set to allocate 79 images, 10% of training examples, to construct a validation set. Following [33], we resize images to 288×384 pixels for training and evaluation and use SegNet [4] based architecture as the backbone.

Similar to [33], we train the model for 200 epochs using Adam optimizer [26] with learning rate $1e-4$, and halve the learning rate after 100 epochs. We choose the best model with early stopping on the preallocated validation set. For DWA [33] we set the temperature hyperparameter to 2, as in the NYUv2 experiment in [33]. For GradNorm [7] we set $\alpha = 1.5$. This value for α was used in [7] for the NYUv2 experiments. In all variants of our method, the auxiliary networks are optimized using SGD with 0.9 momentum. We allocate 2.5% of training examples to form an auxiliary set. We use grid search to tune the learning rate $\{1e-3, 5e-4, 1e-4\}$ and weight decay $\{1e-5, 1e-4\}$ of the auxiliary networks. Here as well, we use skip connection between the main loss ℓ_{main} and the overall loss term and Softplus activation.

B.3 Learning auxiliaries

Multi-class classification datasets. On the CIFAR datasets, we train the model for 200 epochs using SGD with momentum 0.9, weight decay $5e-4$, and initial learning rates $1e-1$ and $1e-2$ for CIFAR10 and CIFAR100, respectively. For the SVHN experiment, we train for 50 epochs using SGD with momentum 0.9, weight decay $5e-4$, and initial learning rates $1e-1$. The learning rate is modified using a cosine annealing scheduler. We use VGG-16 [47] based architecture for the CIFAR experimets, and a 4-layer ConvNet for the SVHN experiment. For MAXL [32] label generating network, we tune the following hyperparameters: learning rate $\{1e-3, 5e-4\}$, weight decay $\{5e-4, 1e-4, 5e-5\}$, and entropy term weight $\{.2, .4, .6\}$ (see [32] for details). We explore the same learning rate and weight decay for the auxiliary network in our method, and also tune the number of optimization steps between every auxiliary parameter update $\{5, 15, 25\}$, and the size of the auxiliary set $\{1.5\%, 2.5\%\}$ (of training examples). We choose the best model on the validation set and allow for early stopping.

Fine-grain classification datasets. In CUB experiments we use the same data and splits as described in Sections 5.2 and B.1. Oxford-IIIT Pet contains 7349 images of 37 species of dogs and cats. We use the official train-test split. We pre-allocate 30% from the training set to validation. As a results, the total number of train/validation/test images are 2576/1104/3669 respectively. Cars [27] contains 16,185 images of 196 car classes. We use the official train-test split and pre-allocate 30% from the training set to validation. As a results, the total number of train/validation/test images are 5700/2444/8041 respectively. In all experiments we use ResNet-18 as the backbone network for both the primary and auxiliary networks. Importantly, the networks are not pre-trained. The task specific (classification) heads in both the primary and auxiliary networks is implemented using a 2-layer NN with sizes 512 and C. Where C is number of labels (e.g., 200 for CUB and 37 for Oxford-IIIT Pet). In all experiments we use the same learning rate of $1e-4$ and weight decay of $5e-3$ which were shown to work best, based on a grid search applied on the STL baseline. For MAXL and AuxiLearn we applied a grid search over the auxiliary network learning rate and weight decay as described in the Multi-class classification datasets subsection. We tune the number of optimization steps between every auxiliary parameter update in $\{30, 60\}$ for Oxford-IIIT Pet and $\{40, 80\}$ for CUB and Cars. Also, the auxiliary set size was tuned over $\{0.084\%, 1.68\%, 3.33\%\}$ with stratified sampling. For our method, we leverage the module of AuxiLearn for combining auxiliaries. We use a Nonlinear network with either two or three hidden layers of sizes 10 (which was selected according to a grid search). The batch size was set to 64 in CUB and Cars experiments and to 16 in Oxford-IIIT Pet experiments. We ran each configuration with 3 different seeds for 150 epochs with ADAM optimizer and used early stopping based on the validation set.

C Additional experiments

C.1 Importance of Auxiliary Set

In this section we illustrate the importance of the auxiliary set to complement our theoretical observation in Section 4. We repeat the experiment in Section 5.1, but this time we optimize the auxiliary parameters ϕ using the training data. Figure 3 shows how the tasks' weights change during training. The optimization procedure is reduced to single-task learning, which badly hurts generalization (see Figure 2). These results are consistent with [32] that added an entropy loss term to avoid the diminishing auxiliary task.

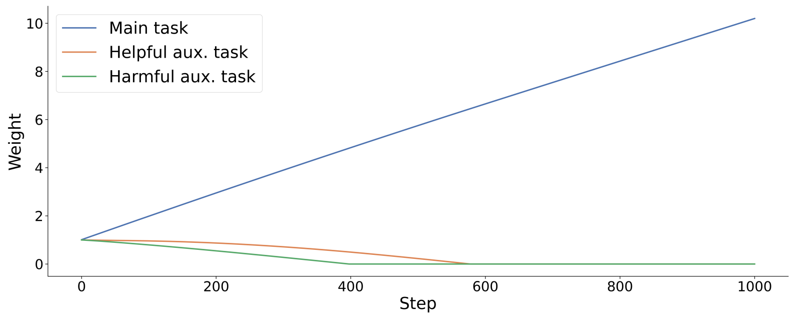


Figure 3: Optimizing task weights on the training set reduce to single-task learning.

C.2 Visualizing the Loss Image

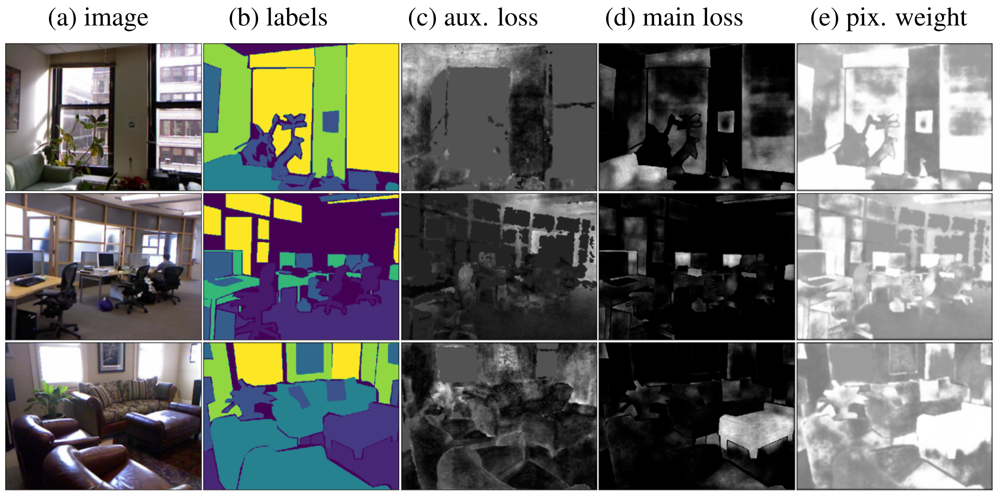


Figure 4: *Loss images* on test examples from NYUv2: **(a)** original image; **(b)** semantic segmentation ground truth; **(c)** auxiliaries loss; **(d)** segmentation (main task) loss; **(e)** adaptive pixel-wise weight $\sum_j \partial\mathcal{L}_T/\partial\ell_j$.

Figure 4 shows an example of the loss-images for the auxiliry (c) and main (d) tasks together with the pixel-wise weights (e). First, note how the loss-images resemble to the actual input images. Thus, a spatial relationship can be leveraged by using a CNN auxiliary network. Second, the pixel weights are a non-trivial combination of main and aux task losses. In the top (bottom) row, the plant (couch) is having low segmentation loss and intermediate auxiliary loss. As a result, a higher weight is allocated to these pixels which increase the error signal while back propagating.

C.3 Monotonocity

As discussed in the main text, it is a common practice to combine auxiliary losses as a convex combination. This is equivalent to parametrize the function $g(\boldsymbol{\ell};\phi)$ as a linear combination over losses $g(\boldsymbol{\ell};\phi)=\sum_{j=1}^{K}\phi_j\ell_j$, with non-negative weights, $\phi_j\geq 0$. Under this parametrization, g is a monotonic non-decreasing function of the losses, since $\partial\mathcal{L}_T/\partial\ell_j\geq 0$. The non-decreasing property means that the overall loss grows (or is left unchanged) with any increase to the auxiliary losses. As a result, an optimization procedure that operates to minimize the combined loss also operates in the direction of reducing individual losses (or not changing them).

A natural question that arises is whether the function g should generalize this behavior, and be constrained to be non-decreasing w.r.t. the losses as well? Non-decreasing networks can "ignore" an auxiliary task by zeroing its corresponding loss, but cannot reverse the gradient of a task by negating its weight. While monotonicity is a very natural requirement, in some cases, negative task weights (i.e., non-monotonicity) seem desirable if one wishes to "delete" input information not directly related to the task at hand [2, 18]. For example, in domain adaptation, one might want to remove information that allows a discriminator to recognize the domain of a given sample [18]. Empirically, we found that training with monotonic non-decreasing networks to be more stable and has better or equivalent performance, see Table 4 for comparison.

Table 4 compares monotonic and non-monotonic auxiliary networks in both the semi-supervised and the fully-supervised setting. Monotonic networks show a small but consistent improvement over non-monotonic ones. It is also worth mentioning that the non-monotonic networks were harder to stabilize.

Table 4: CUB 200-2011: Monotonic vs non-monotonic test classification accuracy ($\pm$ SEM) over three runs.

		Top 1	Top 3
5-shot	Non-Monotonic	46.3 ± 0.32	67.46 ± 0.55
	Monotonic	**47.07 ± 0.10**	**68.25 ± 0.32**
10-shot	Non-Monotonic	58.84 ± 0.04	77.67 ± 0.08
	Monotonic	**59.04 ± 0.22**	**78.08 ± 0.24**
Full Dataset	Non-Monotonic	74.74 ± 0.30	88.3 ± 0.23
	Monotonic	**74.92 ± 0.21**	**88.55 ± 0.17**

C.4 Noisy auxiliaries

We demonstrate the effectiveness of AuxiLearn in identifying helpful auxiliaries and ignoring harmful ones. Consider a regression problem with main task $y = \mathbf{w}^T\mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. We learn this task jointly with $K = 100$ auxiliaries of the form $y_j = \mathbf{w}^T\mathbf{x} + |\epsilon_j|$, where $\epsilon_j \sim \mathcal{N}(0, j \cdot \sigma^2_{aux})$ for $j = 1, ..., 100$. We use the absolute value on the noise so that noisy estimations are no longer unbiased, making the noisy labels even less helpful as the noise increases. We use a linear auxiliary network to weigh the loss terms. Figure 5 shows the learned weight for each task. We can see that the auxiliary network captures the noise patterns, and assign weights based on the noise level.

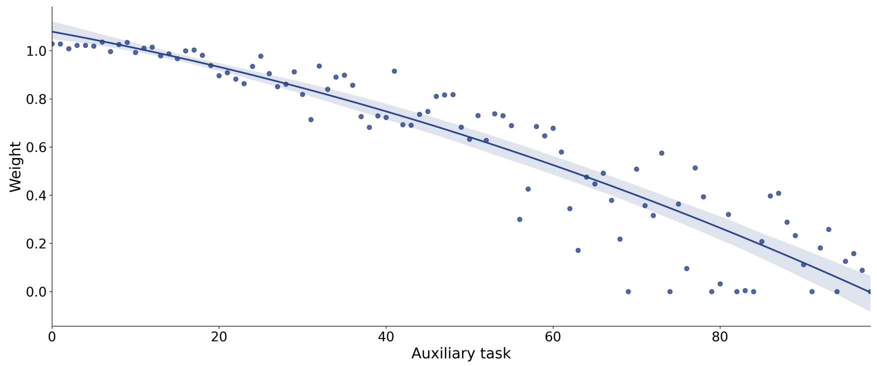
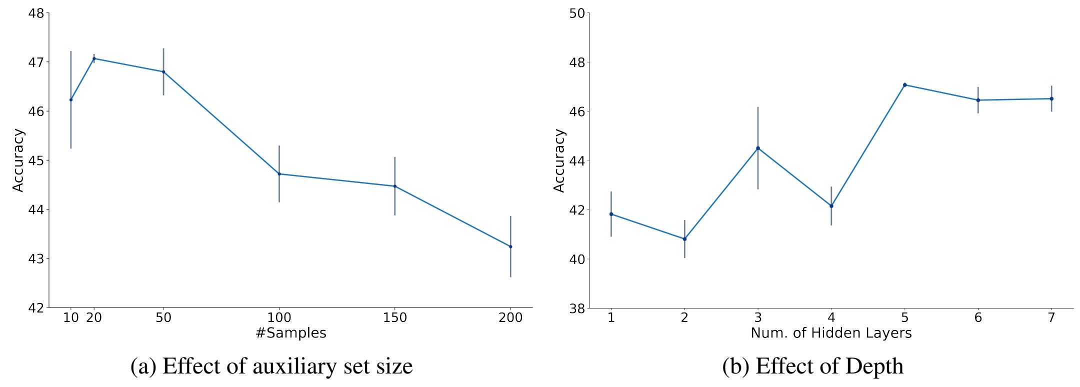


Figure 5: Learning with noisy labels: task ID is proportional to the label noise.

C.5 CUB sensitivity analysis

In this section, we provide further analysis for the experiments conducted on the CUB 200-2011 dataset in the 5-shot setup. We examine the sensitivity of a non-linear auxiliary network to the **size of the auxiliary set**, and the **depth of the auxiliary network**. In Figure 6a we test the effect of allocating (labeled) samples from the training set to the auxiliary set. As seen, allocating between $10 - 50$ samples results in similar performance picking at 20. The figure shows that removing too many samples from the training set can be damaging. Nevertheless, we notice that even when allocating 200 labeled samples (out of 1000), our nonlinear method is still better than the best competitor GSC [13] (which reached an accuracy of 42.57).

Figure 6b shows how accuracy changes with the number of hidden layers. As expected, there is a positive trend. As we increase the number of layers, the network expressivity increases, and the performance improves. Clearly, making the auxiliary network too large may cause the network to overfit the auxiliary set as was shown in Section 4, and empirically in [34].



(a) Effect of auxiliary set size

(b) Effect of Depth

Figure 6: Mean test accuracy ($\pm$ SEM) averaged over 3 runs as a function of the number of samples in the auxiliary set (left) and the number of hidden layers (right). Results are on 5-shot CUB 200-2011 dataset.

C.6 Linearly weighted non-linear terms

To further motivate the use of non-linear interactions between tasks, we train a linear auxiliary network over a polynomial kernel on the losses using the NYUv2 dataset. Figure 7 shows the learned loss weights. From the figure, we learn that two of the three largest weights at the end of training belong to non-linear terms, specifically, Seg^2 and $Seg \cdot Depth$. Also, we observe a *scheduling* effect, in which at the start of training, the auxiliary network focuses on the auxiliary tasks (first ~ 50 steps), and then draws most of the attention of primary network towards the main task.

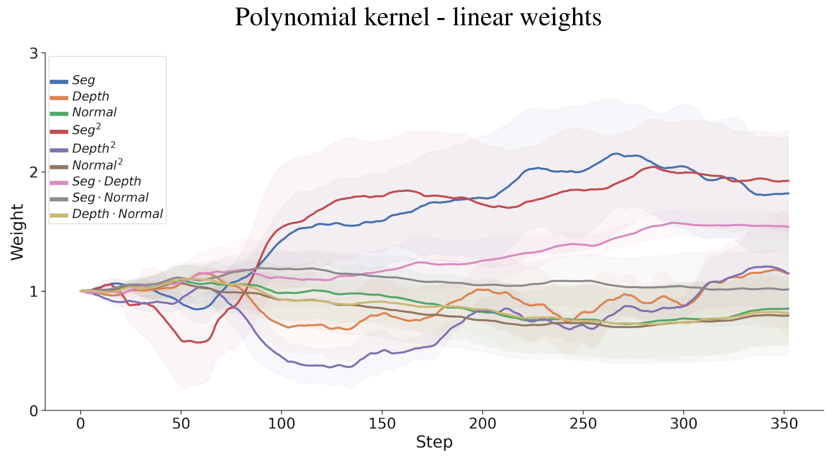


Figure 7: Learned linear weights for a polynomial kernel on the loss terms using NYUv2.

C.7 Fixed auxiliary

As a result of alternating between optimizing the primary network parameters and the auxiliary parameters, the weighting of the loss terms are updated during the training process. This means that the loss landscape is changed during training. This effect is observed in the illustrative examples described in Section 5.1 and Section C.6, where the auxiliary network focuses on different tasks during different learning stages. Since the optimization is non-convex, the end result may depend not only on the final parameters but also on the loss landscape during the entire process.

We examined this effect with the following setup on the 5-shot setting on CUB 200-2011 dataset: we trained a non-linear auxiliary network and saved the best model. Then we retrain with the same configuration, only this time, the auxiliary network is initialized using the best model, and is kept fixed. We repeat this using ten different random seeds, affecting the primary network initialization and data shuffling. As a result, we observed a drop of 6.7% on average in the model performance with an std of 1.2% (46.7% compared to 40%).

C.8 Full CUB dataset

In Section 5.2 we evaluated AuxiLearn and the baseline models performance under a semi-supervised scenario in which we have 5 or 10 labeled samples per class. For completeness sake, we show in Table 5 the test accuracy results in the standard fully-supervised scenario. As can be seen, in this case the STL baseline achieves the highest top-1 test accuracy while our nonlinear method is second on the top-1 and first on the top-3. Most baselines suffer from severe negative transfer due to the large number of auxiliary tasks (which are not needed in this case) while our method cause minimal performance degradation.

Table 5: CUB 200-2011: Fully supervised test classification accuracy ($\pm$ SEM) averaged over three runs.

	Top 1	Top 3
STL	**75.2 $\pm$ 0.52**	88.4 $\pm$ 0.36
Equal	70.16 $\pm$ 0.10	86.87 $\pm$ 0.22
Uncertainty	74.70 $\pm$ 0.56	88.21 $\pm$ 0.14
DWA	69.88 $\pm$ 0.10	86.62 $\pm$ 0.20
GradNorm	70.04 $\pm$ 0.21	86.63 $\pm$ 0.13
GSC	71.30 $\pm$ 0.01	86.91 $\pm$ 0.28
AuxiLearn (ours)		
Linear	70.97$\pm$ 0.31	86.92 $\pm$ 0.08
Deep Linear	73.6 $\pm$ 0.72	88.37 $\pm$ 0.21
Nonlinear	74.92 $\pm$ 0.21	**88.55 $\pm$ 0.17**

C.9 Cityscapes

Cityscapes [8] is a high-quality urban-scene dataset. We use the data provided in [33] with 2975 training and 500 test images. The data comprises of four learning tasks: 19-classes, 7-classes and 2-classes semantic segmentation, and depth estimation. We use the 19-classes semantic segmentation as the main task, and all other tasks as auxiliaries. We allocate 10% of the training data for validation set, to allow for hyperparameter tuning and early stopping. We further allocate 2.5% of the remaining training examples to construct the auxiliary set. All images are resized to 128×256 to speed up computation.

We train a SegNet [4] based model for 150 epochs using Adam optimizer [26] with learning rate $1e-4$, and halve the learning rate after 100 epochs. We search over weight decay in $\{1e-4, 1e-5\}$. We compare AuxiLearn to the same baselines used in Section 5.2 and search over the same hyperparameters as in the NYUv2 experiment. We set the DWA temperature to 2 similar to [33], and the GradNorm hyperparameter α to 1.5, as used in [7] for the NYUv2 experiments. We present the results in Table 6. The ConvNet variant of the auxiliary network achieves best performance in terms of mIoU and pixel accuracy.

C.10 T-SNE of learned auxiliaries

Section 5.4 of the main text shows how AuxiLearn can learn useful auxiliary tasks for the main task of interest using its training data alone. This is achieved by learning to assign labels to samples. Here we further examine the labels learned by AuxiLearn in that setting.

Figure 8 presents a 2D t-SNE projection of the learned soft labels for two classes of the CIFAR10 dataset, *Deer* and *Frog*. A clear structure in the label space is visible. The auxiliary network learns a

Table 6: 19-classes semantic segmentation test set results on Cityscapes, averaged over three runs (± SEM).

	mIoU	Pixel acc.
STL	30.18 ± 0.04	87.08 ± 0.18
Equal	30.45 ± 0.14	87.14 ± 0.08
Uncertainty	30.49 ± 0.21	86.89 ± 0.07
DWA	30.79 ± 0.32	86.97 ± 0.26
GradNorm	30.62 ± 0.03	87.15 ± 0.04
GCS	30.32 ± 0.23	87.02 ± 0.12
AuxiLearn (ours)		
Linear	30.63 ± 0.19	86.88 ± 0.03
Nonlinear	30.85 ± 0.19	87.19 ± 0.20
ConvNet	$\mathbf{30.99 \pm 0.05}$	$\mathbf{87.21 \pm 0.11}$

finer partition of the *Frog* class, separating real images and illustrations. The middle labels learned for *Deer* are more interesting, as it appears the auxiliary network captures more complex features, rather than relying on background colors alone. This region in the label space contains deer with antlers in various poses and varying backgrounds.

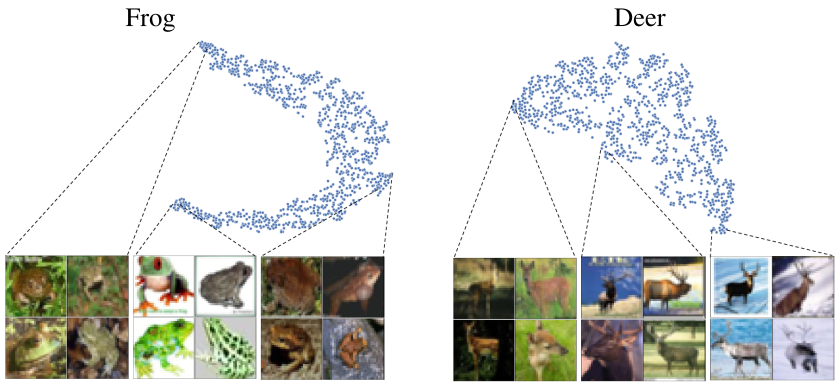


Figure 8: t-SNE applied to the auxiliaries learned for the *Frog* and *Deer* classes, in CIFAR10.

C.11 Learning segmentation auxiliary for 3D point clouds

Recently, several methods were offered for learning auxiliary tasks in point clouds [1, 21, 43]; however, this domain is still largely unexplored and it is not yet clear which auxiliary tasks could be beneficial beforehand. Therefore, it is desirable to automate this process, even at the cost of performance degradation to some extent compared to human designed methods.

We further evaluate our method in the task of generating helpful auxiliary tasks for 3D point-cloud data. We propose to extend the use of AuxiLearn for segmentation tasks. In Section 5.4 we trained an auxiliary network to output soft auxiliary labels for classification task. Here, we use a similar approach, assigning a soft label vector to each point. We then train the primary network on the main task and the auxiliary task of segmenting each point based on the learned labels.

We evaluated the above approach in a part-segmentation task using the ShapeNet part dataset [54]. This dataset contains 16,881 3D shapes from 16 object categories (including Airplane, Bag, Lamp), annotated with a total of 50 parts (at most 6 parts per object). The main task is to predict a part label for each point. We follow the official train/val/test split scheme in [6]. We also follow the standard experimental setup in the literature, which assumes known object category labels during segmentation of a shape (see e.g., [41, 53]). During training we uniformly sample 1024 points from each shape and we ignore point normals. During evaluation we use all points of a shape. For all methods (ours and baselines) we used the DGCNN architecture [53] as the backbone feature extractor and for part segmentation. We evaluated performance using point-Intersection over Union (IoU) following [41].

We compared AuxiLearn with the following baselines: **(1) Single Task Learning (STL):** Training with the main task only. **(2) DefRec:** An auxiliary task of reconstructing a shape with a deformed region [1]. **(3) Reconstructing Spaces (RS):** An auxiliary task of reconstructing a shape from a shuf-

Table 7: Learning auxiliary segmentation task. Test mean IOU on ShapeNet part dataset averaged over three runs (±SEM) - 30 shot

	Mean	Airplane	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motorbike	Mug	Pistol	Rocket	Skateboard	Table
Num. samples	2874	341	14	11	158	704	14	159	80	286	83	51	38	44	12	31	848
STL	75.6	68.7	**82.9**	85.2	**65.6**	82.3	70.2	86.1	75.1	68.4	94.3	55.1	91.0	72.6	60.2	72.3	74.2
DAE	74.0	66.6	77.6	79.1	60.5	81.2	**73.8**	87.1	77.0	65.4	93.6	51.8	88.4	**74.0**	55.4	68.4	72.7
DefRec	74.6	68.6	81.2	83.8	63.6	82.1	72.9	86.9	72.7	69.4	93.4	51.8	89.7	72.0	57.2	70.5	71.7
RS	**76.5**	**69.7**	79.1	**85.9**	64.9	**83.8**	68.4	82.8	79.4	**70.7**	94.5	**58.9**	91.8	72.0	53.4	70.3	**75.0**
AuxiLearn	76.2	68.9	78.3	83.6	64.9	83.4	69.7	**87.4**	**80.7**	68.3	**94.6**	53.2	**92.1**	73.7	**61.6**	**72.4**	74.6

fled version of it [43]. and **(4) Denoising Auto-encoder (DAE):** An auxiliary task of reconstructing a point-cloud perturbed with an iid noise from $\mathcal{N}(0, 0.01)$.

We performed hyper-parameter search over the primary network learning rate in $\{1e-3, 1e-4\}$, weight decay in $\{5e-5, 1e-5\}$ and weight ratio between the main and auxiliary task of $\{1:1, 1:0.5, 1:0.25\}$. We trained each method for 150 epochs, used the Adam optimizer with cosine scheduler. We applied early stopping based on the mean IoU of the validation set. We ran each configuration with 3 different seeds and report the average mean IOU along with the SEM. We used the segmentation network proposed in [53] with an exception that the network wasn't supplied with the object label as input.

For AuxiLearn, we used a smaller version of PointNet [41] as the auxiliary network without input and feature transform layers. We selected PointNet because its model complexity is light and therefore is a good fit in our case. We learned a different auxiliary task per each object category (with 6 classes per category) since it showed better results. We performed hyper-parameter search over the auxiliary network learning rate in $\{1e-2, 1e-3\}$, weight decay in $\{5e-3, 5e-4\}$. Two training samples from each class were allocated for the auxiliary set.

Table 7 shows the mean IOU per category when training with only 30 segmented point-clouds per object category (total of 480). As can be seen, AuxiLearn performance is close to RS [43] and improve upon other baselines. This shows that in this case, our method generates useful auxiliary tasks that has shown similar or better gain than those designed by humans.

C.12 Learning an auxiliary classifier - 15 samples per class

In Section 5.4 we showed how AuxiLearn improve upon baseline methods in fine grain classification with only 30 samples per class. Here we also compare AuxiLearn with the baseline methods when there are only 15 images per class. Table 8 shows that AuxiLearn is superior to baseline methods in this setup as well, even though it requires to allocate some samples from the training data to the auxiliary set.

Table 8: Learning auxiliary task. Test accuracy averaged over three runs (±SEM) - 15 shot

	CUB	Pet
STL	22.6 ± 0.2	13.6 ± 0.7
MAXL-F	24.2 ± 0.7	14.1 ± 0.1
MAXL	24.2 ± 0.8	14.2 ± 0.2
AuxiLearn	$\mathbf{26.1 \pm 0.7}$	$\mathbf{18.0 \pm 0.9}$