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 with multiple auxiliary tasks is a common practice used in deep learning for improving the performance on the 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. 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. We evaluate AuxiLearn in a series of tasks and domains, including image segmentation and learning with attributes. We find that AuxiLearn consistently improves accuracy compared with competing methods.

1 Introduction

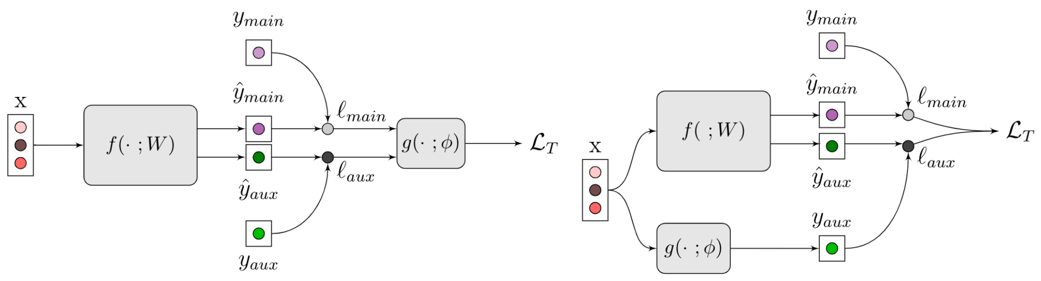
Training deep models on a given task often facilitated by adding auxiliary tasks [39]. For example, when learning to segment an image into objects, accuracy can be improved when the model is simultaneously trained to predict other properties of the image like pixel depth or 3D structure [44]. In this context, joint training with auxiliary tasks adds an inductive bias, encouraging the model to learn meaningful representations and avoid overfitting spurious correlations. Unfortunately, it is still not clear in which cases adding an auxiliary task would benefit training and which tasks should be added. For example, depth estimation was shown to be a useful auxiliary for semantic segmentation, but semantic segmentation is a harmful auxiliary task for depth estimation [44].

In some domains, it is 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 [18] and patch shuffling [12, 35]. In these cases, combining all auxiliary tasks into a single loss can be challenging [12]. The common practice is to compute a convex combination of pretext losses by tuning the weights of individual losses using a hyperparameter grid search. This approach limits the potential of auxiliary learning because the run time of grid search grows exponentially with the number of auxiliary tasks.

In many other domains, it is not even clear which auxiliary tasks could be beneficial. For example, for point cloud classification, few self-supervised tasks have been proposed, and their benefits are limited so far [1, 19, 40, 45]. This is also the case for learning in domains outside perception, where more

[*]Equal contributor
[†]Equal contributor

Preprint. Under review.



(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 $\mathcal{L}_A$.

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 name *AuxiLearn*. It leverages recent progress made in implicit differentiation for optimizing hyperparameters [27, 31]. We show the effectiveness of AuxiLearn in two types of problems. First, in **combining auxiliaries**, for problems 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 show how to 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. Learning an auxiliary 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. In both cases, the auxiliary network is then trained jointly with the primary network using implicit differentiation [27, 31].

Important distinction from previous works, such as [22, 29], 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, as we see it, is improving generalization rather than helping the optimization on the training data.

To validate our proposed solution, we perform an extensive experimental study on various tasks. 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 have infinite capacity. The second aims to understand which auxiliaries benefit 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: (i) A new way to combine multiple loss terms in a deep non-linear manner to produce a unified training objective; (ii) A new way for learning novel auxiliary tasks from data; (iii) New results on a variety of auxiliary learning tasks. We conclude that implicit differentiation can play a significant role in automating the design of auxiliary learning. We make our source code publicly available at `https://github.com/AvivNavon/AuxiLearn`.

2 Related work

Learning with multiple tasks. In Multitask Learning (MTL), we simultaneously solve multiple learning problems while sharing information and knowledge across tasks. MTL has shown to benefit the optimization process and improve task-specific generalization performance, compared to single-task learning in some cases [44]. In contrast with MTL, in learning with auxiliary tasks, we wish to optimize a model for a single, main task, and the purpose of all other tasks is to facilitate the learning on the primary task. At test time, only the main task is considered. This approach has been successfully applied in multiple domains, including computer vision [50], natural language processing [46], and reinforcement learning [28].

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 [22]. 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. Recent works propose scaling task weights using gradient magnitude [7], task uncertainty [22], or the rate of change in losses [30]. These methods assume all tasks are equally important, hence may not be suited for auxiliary learning. [13, 28] proposed weighting auxiliaries using gradient similarity. However, this method does not scale well to multiple auxiliaries and does 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 approaches have been proposed (see [21] for a recent survey), but the joint representation learned through SSL may suffer from negative transfer and hurt the main task [43]. [29] 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, while the auxiliary loss by its nature disrupts the training loss on the main task.

Implicit differentiation based optimization. Our formulation gives rise to a bi-level optimization problem. Such problems naturally arise in the context of meta-learning [15, 38] and hyperparameter optimization [5, 16, 25, 27, 31, 36]. 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. [32] proposed approximating the Hessian with the identity matrix, whereas [16, 36, 38] use Conjugate Gradient (CG) to approximate the product. Following [27, 31], we use a truncated Neumann series and efficient vector-Jacobian products, as it is 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.

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 an expert provides us with K auxiliary tasks. 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 $g : \mathbb{R}^{K+1} \to \mathbb{R}$ 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 (1) is given by $h(\cdot\,; \phi) = g(\boldsymbol{\ell}; \phi)$.

Generally, $g(\boldsymbol{\ell}; \phi)$ 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, since 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 neural net. This general form allows us to capture complex interactions between tasks and learn non-linear combinations of losses.

One way to look at 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 $\ell_{main}(\mathbf{x}, y^{main}) + \sum_j \alpha_{j,\mathbf{x}} \ell_j(\mathbf{x}, y^j)$, where $\alpha_{j,\mathbf{x}} = \frac{\partial g}{\partial \ell_j}$. This observation connects our approach to other works that adaptively weigh loss terms ([13, 30]).

Convolutional loss network. In certain problems, there exist a spatial relation among losses. For example, consider the tasks of semantic segmentation and depth estimation for images. The common approach is to average the losses over all locations. We can, however, leverage this spatial relation for creating a *loss-image*, in which each task forms a channel of pixel-losses induced by the task. We can now stack those channels and parametrize g as a CNN that acts on this loss-image. As a result, we can learn a spatial-aware loss function that captures interactions between task losses. See example in Figure 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.1 for a detailed discussion and empirical comparison to non-monotonic networks.

3.3 Learning new auxiliary tasks

The previous subsection focused on a case 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. How can we utilize the benefits of auxiliary learning in such cases? We propose using our framework to generate an auxiliary task with the auxiliary network and learning the main task and the auxiliary task with the primary network. The learned auxiliary task is tailor-made to help the learning of the main task.

Learning auxiliary classification tasks. Suppose we are presented with a learning problem, and we wish to train a network to generate an auxiliary classification task. We can use g to learn a soft labeling function and use f to learn the main and auxiliary tasks. As depicted in Figure 1, during training, we pass each example to both the primary model f to output the predictions $\hat{y}_{main}, \hat{y}_{aux}$, and to g to produce soft auxiliary labels y_{aux}. 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 $\mathcal{L}_A = \ell_{main}$. Here, h is given by $h = \ell_{aux}(f(\mathbf{x}_i^t; W), g(\mathbf{x}_i^t; \phi)))$. One immediate application of this method is in a semi-supervised setting, as we can generate auxiliary task labels for unlabeled data. Furthermore, we can easily extend this method to segmentation tasks by, e.g., learning soft labels per pixel in images, or point in point-clouds.

Although this setup is similar to that seen in [29], the two approaches are different. The solution proposed in [29] is optimized on the training data, and as a result, they experience mode collapse. To address this, they introduce an entropy term. On the other hand, our solution is more natural as we remove the need to add an artificial loss term.

3.4 Optimizing auxiliary parameters

We now return to the bi-level optimization problem in (2) and present the optimizing method for ϕ. Solving (2) for ϕ poses a problem due to the indirect dependence of $\mathcal{L}_A$ on the auxiliary parameters. To compute the gradients of the loss $\mathcal{L}_A$ w.r.t 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 [27, 31], we use the implicit function theorem (IFT) to evaluate $\nabla_\phi W^*$:

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

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

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

See Appendix A for detailed derivation. To compute the vector and Hessian inverse product, we use the algorithm proposed in [31], which uses a Neumann approximation and an 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 in Appendix A.

4 Analysis

4.1 Complexity of auxiliary hypothesis space

In our learning setup, an extra 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 non-intuitive. We now 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 example is important because it shows that even apparently simple looking auxiliary losses can overfit due to the interaction with the model hypothesis space. Furthermore, this example motivates our use of a seperate auxiliary set and validation 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 $\mathcal{L}_T(W,\phi) = \sum_i \ell_{main}(\mathbf{x}_i^t, \boldsymbol{y}_i^t, W) + \phi \cdot \ell_{aux}(\mathbf{x}_i^t, \boldsymbol{y}_i^t, W)$. 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*

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

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.

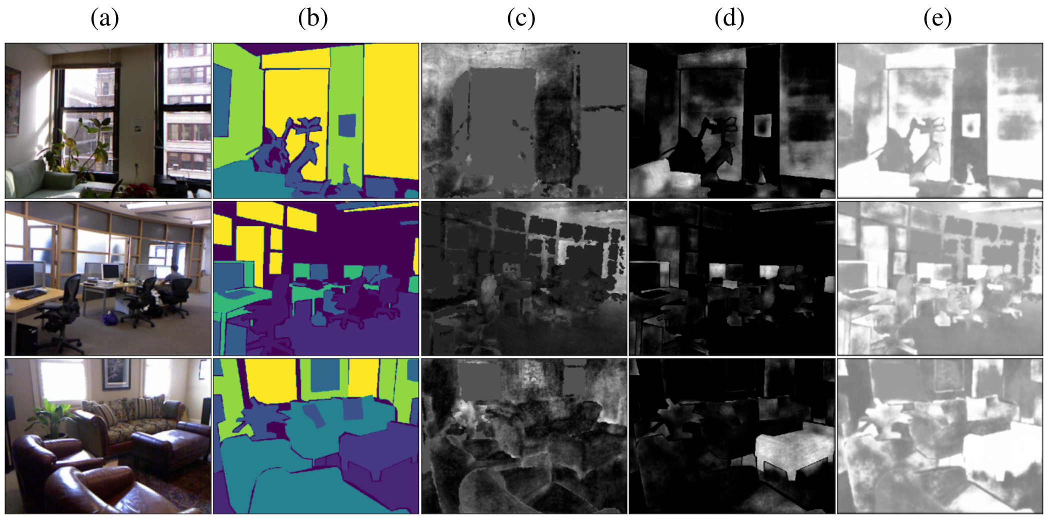


Figure 2: *Loss images* on test examples from NYUv2: **(a)** original image; **(b)** semantic segmentation ground truth; **(c)** auxiliaries loss; **(d)** segmentation (main task) loss; **(e)** adaptive pixel-wise weight $\sum_j \partial\mathcal{L}_T/\partial\ell_j$.

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

This simple result shows that the key quantity to observe is the Newton update, rather than the gradient as is often used [28, 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 two families of tasks: (i) Combining given auxiliary tasks into a unified loss (Sections 5.1 - 5.3), and (ii) Generating a new auxiliary task (Section 5.4). 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). See further experimental details, design choices, and analysis in the Appendix.

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 single-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, 41]. All variants are constrained to represent only monotone non-decreasing functions by clipping negative parameters.

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, $y_1 = \mathbf{w}^{\star T}\mathbf{x} + \epsilon_1$,

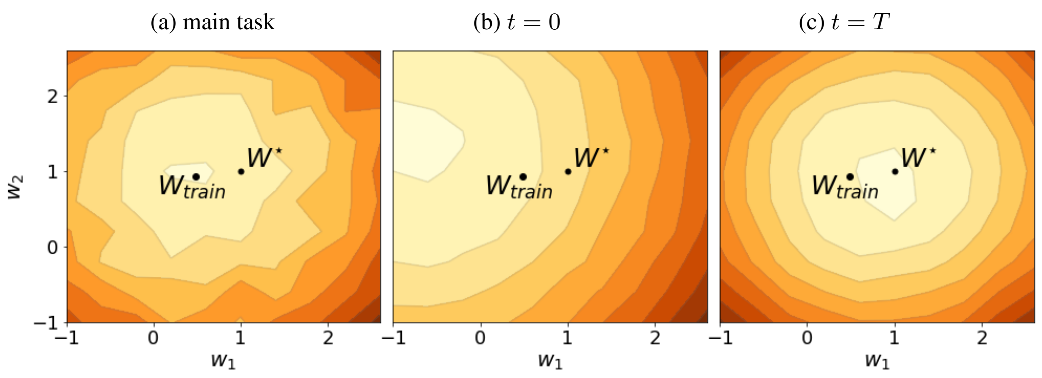


Figure 3: Loss landscape generated by the auxiliary network. Darker is higher. See text for details.

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 $\epsilon_0 \sim \mathcal{N}(0, \sigma^2_{main})$ and $\epsilon_1, \epsilon_2 \sim \mathcal{N}(0, \sigma^2_{aux})$. We optimize a linear model with weights $\mathbf{w} = (w_1, w_2)^T \in \mathbb{R}^2$ that are shared across tasks, i.e., no task-specific parameters. We set $\mathbf{w}^\star = (1, 1)^T$, $\tilde{\mathbf{w}} = (-4, 2)^T$ and $\sigma_{aux} = \sigma_{main}/4 = 1$. We train a linear auxiliary network to output task weights over the losses, $g(\boldsymbol{\ell}; \phi) = \sum_j \phi_j \ell_j$, and observe the changes to the loss landscape in Figure 3.

We can see that the auxiliary network learns to ignore the harmful auxiliary and use the helpful one to find a better solution by changing the loss landscape. The left plot shows the training set loss landscape for the main task (e.g. STL), with a training set optimal solution $\mathbf{w}_{train}$. Note that $\mathbf{w}_{train} \neq \mathbf{w}^*$ due to noise in the finite training data. At the beginning of training ($t = 0$) the relatively large harmful task weight pushes the minimizer far from $\mathbf{w}^\star$. Throughout the optimization process, the auxiliary network increases the weight of the helpful auxiliary task while decreasing the weight of the harmful one. At convergence ($t = T$) the minimizer is $\approx \mathbf{w}^\star$.

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 images with predictive visual attributes. As an example, consider the case of recognizing bird species, which would require an ornithologist, yet anyone 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 evaluated AuxiLearn in this setup on the CUB-200 dataset [48]. CUB is a dataset for fine-grained classification of bird species. Each image is associated with a specie (one of 200) and a set of 312 binary visual attributes, which we use as auxiliaries. We split the predefined test set to 2897 samples for validation and 2897 for testing. 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 that auxiliary labels are available for all images but only 5 and 10 labels per class of the main task (noted as 5-shot and 10-shot, respectively).

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 [22]	35.22 ± 0.3	54.99 ± 0.7	53.75 ± 0.6	73.25 ± 0.3
DWA [30]	41.82 ± 0.1	62.91 ± 0.4	54.90 ± 0.3	75.74 ± 0.3
GradNorm [7]	41.49 ± 0.4	63.12 ± 0.4	55.23 ± 0.1	75.62 ± 0.3
GCS [13]	42.57 ± 0.7	62.60 ± 0.1	55.65 ± 0.2	75.71 ± 0.1
AuxiLearn (ours)				
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 with competing approaches proposed for MTL and auxiliary learning: **(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 the losses based on gradient magnitude. **(4) Uncertainty:** [22], 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 to determine if an auxiliary should be used. **(6) Dynamic weight averaging (DWA):** [30], an MTL approach that sets task weights based on the rate of change of the loss over time.

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 indicates that at least part of the improvement achieved by our method is attributed to over-parametrization of the auxiliary network. In our experiments, the learning dynamics observed while training deep linear auxiliary networks were similar to those seen in deep non-linear models. Finally, we note that using the auxiliary terms in a fully-supervised scenario does not yield any performance gain. Nonetheless, while most other methods suffer from a large negative transfer, our method causes small performance degradation. The results on the full dataset and additional analysis are presented in Appendix C.5 and C.4 respectively.

5.3 Pixel-wise losses

We consider the indoor-scene segmentation task provided in [9], which uses the NYUv2 dataset [34]. We use the 13-class semantic segmentation as the main task, with depth prediction and surface-normal estimation [14] as auxiliaries. Simultaneous learning of these tasks has shown significant improvement compared to the STL models (e.g., [30, 33]). 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: Test semantic segmentation results on NYUv2 dataset, averaged over four runs ($\pm$ SEM).

	mIoU	Pixel acc.
STL	18.90 ± 0.21	54.74 ± 0.94
Equal	19.20 ± 0.19	55.37 ± 1.00
Uncertainty [22]	19.34 ± 0.18	55.70 ± 0.79
DWA [30]	19.38 ± 0.14	55.37 ± 0.35
GradNorm [7]	19.52 ± 0.21	56.70 ± 0.33
GCS [13]	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

Figure 2 shows examples of the resulting loss images, together with the adaptive pixel-wise loss weight $\partial \mathcal{L}_T / \partial \ell_j$, for three test images. The auxiliary network assigns considerable weight to regions with high segmentation loss but also focuses the learning on regions with high auxiliary loss and intermediate segmentation loss (see upper left part of the middle row).

Table 2 reports the mean Intersection over Union (mIoU) and pixel accuracy for the main segmentation task. All weighting methods achieve a performance gain over the STL model. The comparison shows that the ConvNet variant of our auxiliary network outperforms all competitors in terms of test mIoU. We present additional experiments on the Cityscapes dataset in the Appendix C.8.

5.4 Learning a classification auxiliary task

Table 3: Test accuracy on CIFAR10, CIFAR100, and MNIST datasets, averaged over three runs ($\pm$SEM).

	CIFAR10			CIFAR100			MNIST		
	5%	10%	15%	5%	10%	15%	5%	10%	15%
STL	50.8 ± 0.8	72.6 ± 2.1	80.3 ± 0.1	19.8 ± 0.7	31.3 ± 0.1	38.6 ± 0.7	$\mathbf{95.5 \pm 0.1}$	96.7 ± 0.1	97.1 ± 0.2
Random	56.1 ± 0.1	75.9 ± 0.6	81.2 ± 0.2	20.4 ± 0.6	32.5 ± 0.7	34.0 ± 0.4	95.0 ± 0.8	96.7 ± 0.5	97.0 ± 0.2
MAXL	58.2 ± 0.3	75.9 ± 0.3	$\mathbf{81.4 \pm 0.3}$	21.0 ± 0.4	32.8 ± 0.1	$\mathbf{42.0 \pm 0.4}$	95.2 ± 0.3	96.8 ± 0.5	97.5 ± 0.2
AuxiLearn (ours)	$\mathbf{60.7 \pm 1.3}$	$\mathbf{76.8 \pm 0.1}$	$\mathbf{81.4 \pm 0.3}$	$\mathbf{21.5 \pm 0.3}$	$\mathbf{33.2 \pm 0.3}$	40.6 ± 0.6	95.4 ± 0.3	$\mathbf{97.0 \pm 0.1}$	$\mathbf{97.8 \pm 0.1}$

In many cases, devising helpful auxiliaries is a challenging task. In the following experiment, we focus on learning a multiclass classification auxiliary task. We consider CIFAR10 [24], CIFAR100 [24], and MNIST [26] datasets, with the standard multiclass classification as the main task. For each dataset, we allocate 10% of the training data to construct a validation set for hyperparameter tuning and early stopping. Following [29], we learn a different auxiliary task for each class in the main task. 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 using only 5%, 10%, and 15% of the training examples. We use the same architecture for the primary and label generator network (both for AuxiLearn and for MAXL). We use VGG-16 [42] as the backbone for both CIFAR datasets and a 2-layers ConvNet for the MNIST experiment. Here, 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 [29] for learning auxiliary tasks, without the need for additional data. **(3) Random:** A randomly initialized label generator network baseline with fixed parameters.

The results are presented in Table 3. AuxiLearn outperforms all baselines in most datasets and dataset sizes, even though it sacrifices some of the training set to construct an auxiliary set. It is worth noting that the *Random label* baseline achieves better accuracy than the STL model on two datasets. We attribute this performance gain to the regularization effect of adding a random auxiliary task. We present a 2D t-SNE projection of the soft labels learned by the auxiliary network for two classes from the main task in CIFAR10, in Appendix C.7. We also extend the method to point-clouds for the task of part-segmentation and present the results in Appendix C.9.

6 Discussion

We presented a novel approach for learning how to combine auxiliaries and how to learn new auxiliary tasks. We show empirically that our method can offer significant improvement over existing methods.

This work opens interesting directions for future research. First, we observe that deep models benefits auxiliary networks, even in the case of deep linear networks that have the same expressive power as linear networks. this effect is similar to what have been observed 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. The broader question remains about the most efficient way to allocate labels to various components of the joint optimization problem. These topics await further research.

References

[1] Idan Achituve, Haggai Maron, and Gal Chechik. Self-supervised learning for domain adaptation on point-clouds. *arXiv preprint arXiv:2003.12641*, 2020.

[2] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations, ICLR*, 2017.

[3] Sanjeev Arora, N Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *35th International Conference on Machine Learning*, 2018.

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

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

[6] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[7] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[9] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

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

[13] Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

[14] David Eigen and Rob Fergus. 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, 2015.

[15] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[16] Chuan-sheng Foo, Chuong B Do, and Andrew Y Ng. Efficient multiple hyperparameter learning for log-linear models. In *Advances in neural information processing systems*, pages 377–384, 2008.

[17] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning (ICML)*, 2015.

[18] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

[19] Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8160–8171, 2019.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

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

[23] Diederik P. Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. In *Proc. of the 3rd International Conference on Learning Representations*, 2014.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[25] Jan Larsen, Lars Kai Hansen, Claus Svarer, and M Ohlsson. Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, pages 62–71. IEEE, 1996.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning (ICML)*, 2018.

[28] Xingyu Lin, Harjatin Baweja, George Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4773–4784, 2019.

[29] Shikun Liu, Andrew Davison, and Edward Johns. Self-supervised generalisation with meta auxiliary learning. In *Advances in Neural Information Processing Systems*, pages 1677–1687, 2019.

[30] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

[31] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. *arXiv preprint arXiv:1911.02590*, 2019.

[32] Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, pages 2952–2960, 2016.

[33] Arsalan Mousavian, Hamed Pirsiavash, and Jana Košecká. Joint semantic segmentation and depth estimation with deep convolutional networks. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 611–619. IEEE, 2016.

[34] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[35] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[36] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*, pages 737–746, 2016.

[37] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 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, 2017.

[38] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, pages 113–124, 2019.

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

[40] Jonathan Sauder and Bjarne Sievers. Self-supervised deep learning on point clouds by reconstructing space. In *Advances in Neural Information Processing Systems*, pages 12942–12952, 2019.

[41] Andrew M Saxe, James L Mcclelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *In International Conference on Learning Representations*. Citeseer, 2014.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] Trevor Standley, Amir R Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? *arXiv preprint arXiv:1905.07553*, 2019.

[44] Trevor Standley, Amir Roshan Zamir, Dawn Chen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? *CoRR*, abs/1905.07553, 2019.

[45] Lulu Tang, Ke Chen, Chaozheng Wu, Yu Hong, Kui Jia, and Zhixin Yang. Improving semantic analysis on point clouds via auxiliary supervision of local geometric priors. *arXiv preprint arXiv:2001.04803*, 2020.

[46] Trieu Trinh, Andrew Dai, Thang Luong, and Quoc Le. Learning longer-term dependencies in RNNs with auxiliary losses. In *International Conference on Machine Learning*, pages 4965–4974, 2018.

[47] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[48] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[49] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):1–12, 2016.

[50] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014.

Appendix: Auxiliary Learning by Implicit Differentiation

A Gradient derivation

We provide here the derivation of (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 have proposed in [31].

Algorithm 1: AuxiLearn

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

Algorithm 2: Hypergradient

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

B Experimental details

B.1 CUB 200-2011

Data. 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.

Training details for baselines. We fine-tuned a ResNet-18 [20] 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 [30], we searched over the temperature in $\{0.5, 2, 5\}$ and for

GradNorm [7], over α in $\{0.3, 0.8, 1.5\}$. The GSC [13] method has a computational complexity that grows with the number of tasks. As a result, we were able to run this method 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 [23] 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 used 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 [30], we resize images to 288×384 pixels for training and evaluation and use SegNet [4] based architecture as the backbone.

Similar to [30], we train the model for 200 epochs using Adam optimizer [23] 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 [30] we set the temperature hyperparameter to 2, as in the NYUv2 experiment in [30]. 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.

B.3 Learning auxiliaries

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 MNIST experiments, we train for 50 epochs using SGD with momentum 0.9, weight decay $5e-4$, and initial learning rates $1e-1$. We modify the learning rates with a cosine annealing scheduler. For MAXL [29], we tune the following hyperparameters for the label generating network: learning rate $\{1e-3, 5e-4\}$, weight decay $\{5e-4, 1e-4, 5e-5\}$, and entropy term weight $\{.2, .4, .6\}$ (see [29] 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.

C Additional experiments

C.1 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, 17]. For example, in domain adaptation, one might want to remove information that allows a discriminator to recognize the domain of a given sample [17]. 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	$\mathbf{47.07 \pm 0.10}$	$\mathbf{68.25 \pm 0.32}$
10-shot	Non-Monotonic	58.84 ± 0.04	77.67 ± 0.08
	Monotonic	$\mathbf{59.04 \pm 0.22}$	$\mathbf{78.08 \pm 0.24}$
Full Dataset	Non-Monotonic	74.74 ± 0.30	88.3 ± 0.23
	Monotonic	$\mathbf{74.92 \pm 0.21}$	$\mathbf{88.55 \pm 0.17}$

C.2 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 4 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.

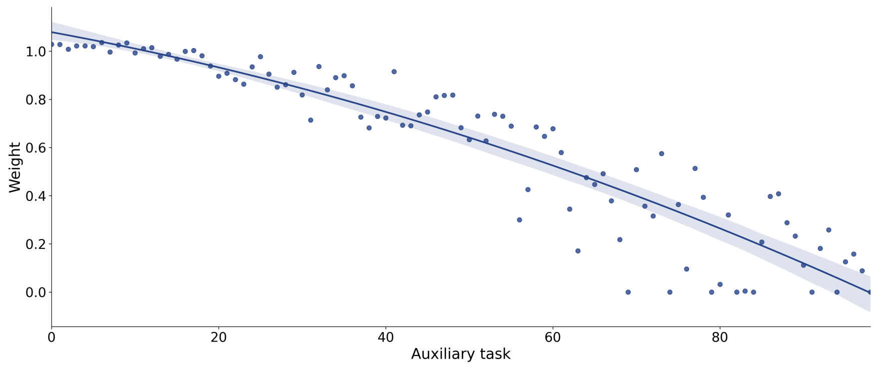


Figure 4: Learning with noisy labels: task ID is proportional to the label noise.

C.3 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 5 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.

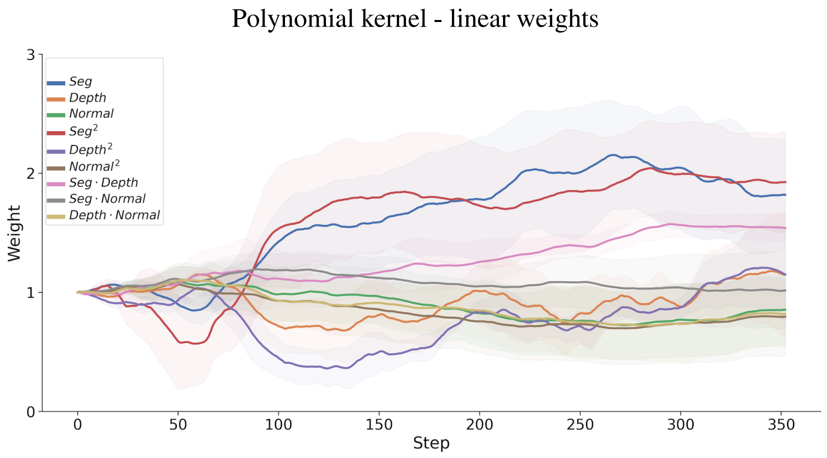
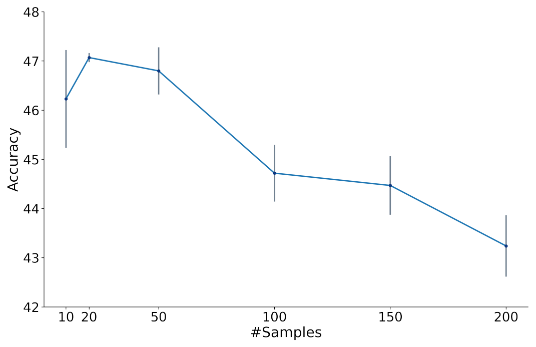


Figure 5: Learned linear weights for a polynomial kernel on the loss terms using NYUv2.

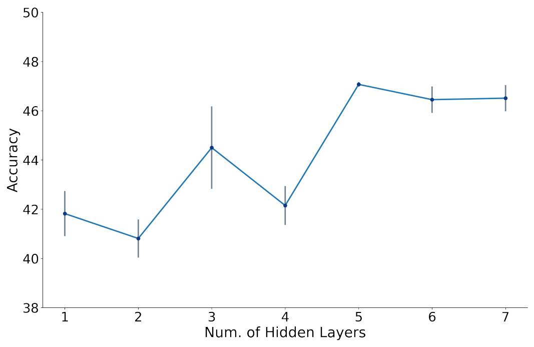
C.4 CUB sensitivity analysis

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 [31].



(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.5 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 (± SEM) averaged over three runs.

	Top 1	Top 3
STL	**75.2 ± 0.52**	88.4 ± 0.36
Equal	70.16 ± 0.10	86.87 ± 0.22
Uncertainty [22]	74.70 ± 0.56	88.21 ± 0.14
DWA [30]	69.88 ± 0.10	86.62 ± 0.20
GradNorm [7]	70.04 ± 0.21	86.63 ± 0.13
GSC [13]	71.30 ± 0.01	86.91 ± 0.28
AuxiLearn (ours)		
Linear	70.97± 0.31	86.92 ± 0.08
Deep Linear	73.6 ± 0.72	88.37 ± 0.21
Nonlinear	74.92 ± 0.21	**88.55 ± 0.17**

C.6 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.3, 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.7 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 7 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 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.

C.8 Cityscapes

Cityscapes [8] is a high-quality urban-scene dataset. We use the data provided in [30] 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.

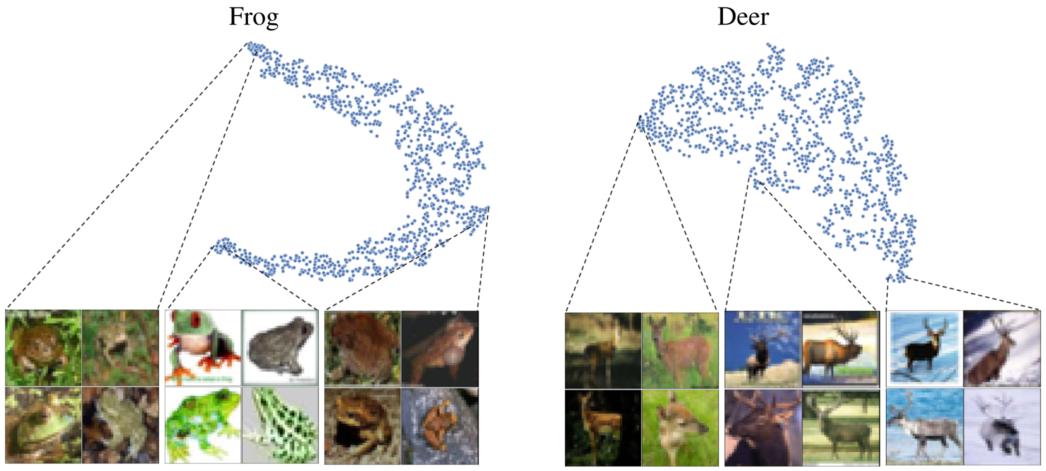


Figure 7: t-SNE applied to the auxiliaries learned for the *Frog* and *Deer* classes, in CIFAR10.

We train a SegNet [4] based model for 150 epochs using Adam optimizer [23] 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 [30], 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.

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

	mIoU	Pixel acc.
STL	30.18 ± 0.04	87.08 ± 0.18
Equal	30.45 ± 0.14	87.14 ± 0.08
Uncertainty [22]	30.49 ± 0.21	86.89 ± 0.07
DWA [30]	30.79 ± 0.32	86.97 ± 0.26
GradNorm [7]	30.62 ± 0.03	87.15 ± 0.04
GCS [13]	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	$\mathbf{30.99 \pm 0.05}$	$\mathbf{87.21 \pm 0.11}$

C.9 Learning segmentation auxiliary for 3D point clouds

Recently, several methods were offered for learning auxiliary tasks in point clouds [1, 19, 40]; 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 [49]. 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

Table 7: Segmentation results on ShapeNet part dataset with 30 samples labeled per class. Average over 3 runs.

	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
RegRec [1]	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 [40]	**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

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., [37, 47]). 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 [47] as the backbone feature extractor and for part segmentation. We evaluated performance using point-Intersection over Union (IoU) following [37].

We compared our approach to the following baselines: **(1) Single Task Learning (STL):** Training with the main task only. **(2) RegRec:** An auxiliary task of reconstructing a shape with a deformed region as proposed by [1]. **(3) Reconstructing Spaces (RS):** An auxiliary task of reconstructing a shape from a shuffled version of it [40]. and **(4) Denoising Auto-encoder (DAE):** An auxiliary task of reconstructing a point-cloud with an iid noise added per point 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 [23] optimizer with cosine scheduler. We applied early stopping based on the IoU of the validation set. We ran each configuration with 3 different seeds and report the average mIOU along with the SEM. We used the segmentation network proposed in [47] with an exception that the network wasn't supplied with the object label as input.

For AuxiLearn, we used a smaller version of PointNet [37] as the auxiliary network (that generates the point labels) 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 $\{5e-3, 5e-4\}$. We allocated 2 training samples from each class in the training set for the auxiliary set.

Table 7 shows the 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 [40] 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.