Censoring the auxiliary loss gradient to mitigate risks of negative transfer

Vivien Tran-Thien

Abstract

Auxiliary tasks have long been successfully used to train neural networks. They can be especially beneficial in the cases of supervised learning with too few labeled examples or reinforcement learning with sparse rewards. However, they may also hurt the performance on the main task. In this paper, I propose and test a simple method to mitigate such risk of negative transfer. It consists in systematically replacing the auxiliary loss gradient with its projection on the half-space of vectors that form an acute angle with the gradient for the main loss. I explain the underlying intuition, present two variants of this method and illustrate its advantages through several experiments.

1. Introduction

Multi-task learning refers to jointly learning several tasks in the hope of achieving better generalization performance (Caruana, 1997). It encompasses two similar but significantly different settings. In the first one, we are genuinely interested in accomplishing well all tasks. In the second one, we try to perform *auxiliary tasks* for the sole purpose of mastering a *main task*.

Multi-task learning has notably yielded satisfying empirical results with neural networks, in particular in speech processing (Sejnowski & Rosenberg, 1987), natural language processing (Collobert & Weston, 2008), computer vision (Pomerleau, 1989) and reinforcement learning (Jaderberg et al., 2017). Several mechanisms, such as statistical data amplification, representation bias (Caruana, 1997) and regularization (Ruder, 2017) have been put forward to explain this performance gain.

However, adding tasks may be unhelpful or even harmful. The latter case, which is called *negative transfer* (Rosenstein et al., 2005), is a central challenge of multi-task learning. Another issue relates to the most common approach used to train neural networks to achieve multiple tasks in parallel, i.e. minimizing a weighted sum of task-specific losses. This seems inappropriate in the case of auxiliary tasks because it suggests we would be willing to tolerate a higher value of the loss for the main task if it helped sufficiently decrease the loss of an auxiliary task. On the contrary, by definition of the auxiliary tasks, reducing their losses is only looked for to the extent this contributes to decreasing the loss of the main task.

In this paper[1], I focus on the case of multi-task learning with neural networks and auxiliary tasks and I aim at mitigating these two problems. The contributions of this paper are the following. In section 2, I propose to deviate from the standard training method for multi-task learning to better reflect the fundamental asymmetry between the main task and the auxiliary tasks. I do so by adjusting the gradients of the auxiliary losses at every optimization step so that they are less likely to increase the main loss. In section 4, I test the proposed method and two of its variants in a series of diverse experiments[2]. The results suggest that the proposed method can both improve performance and reduce the sensitivity to excessive weights of the auxiliary losses.

2. Proposed method

2.1. Standard approach for multi-task learning with neural networks

In the standard multi-task learning approach with neural networks, a single network combines several sub-networks, each of them corresponding to an individual task. These sub-networks either share some neurons (*hard parameter sharing*), as illustrated on Figure 1, or have some of their neurons constrained to be similar, e.g. by penalizing the square distance between their weights (*soft parameter sharing*) (Ruder, 2017). Such network is most commonly trained by defining a loss $\mathcal{L}_i$ for each task i and minimizing a weighted sum of these losses $\sum \lambda_i \mathcal{L}_i$ with a gradient descent-based algorithm.

2.2. Censoring the auxiliary loss gradient

This paper covers the situation of a main task along with one or several auxiliary tasks. For the sake of clarity, I further assume a single auxiliary task even though all the following can easily be extended to several auxiliary tasks. The overall loss mentioned above can then be rewritten, without loss of

[1] This paper is an extended version of a blog post entitled *Learning through Auxiliary Tasks* and published in February 2019.

[2] The code and experimental results to reproduce all experiments and figures in this paper are available on GitHub.

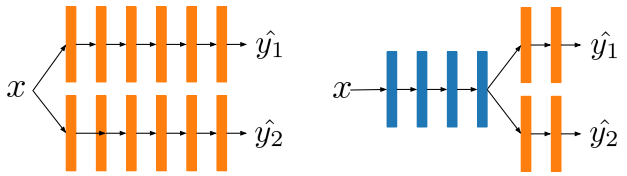

Figure 1. **Left**: *Single task learning* with one neural network responsible for each task. **Right**: *Multi-task learning* with a single neural network performing all tasks. The first hidden layers are typically shared across all tasks and followed by task-specific branches.

generality, as $\mathcal{L} = \mathcal{L}_{\mathbf{main}} + \lambda\mathcal{L}_{\mathbf{aux}}$. The weight λ balances the main task and the auxiliary task. $\lambda = 0$ corresponds to single task learning whereas increasing λ gives more and more influence to the auxiliary task.

When using a gradient descent-based optimization method, the parameters θ of the neural network are iteratively adjusted according to $\nabla\mathcal{L} = \nabla\mathcal{L}_{\mathbf{main}} + \lambda\nabla\mathcal{L}_{\mathbf{aux}}$. However, $\nabla\mathcal{L}_{\mathbf{aux}}$ and $\nabla\mathcal{L}_{\mathbf{main}}$ may point to completely different directions. In a pathological case, $\nabla\mathcal{L}_{\mathbf{aux}} = -\nabla\mathcal{L}_{\mathbf{main}}$ and if $\lambda = 1$, the auxiliary task and the main task negate each other and no progress on the main task is ever made. In the less extreme example depicted on the left part of Figure 2, $\nabla\mathcal{L}_{\mathbf{aux}}$ still cancels a significant part of $\nabla\mathcal{L}_{\mathbf{main}}$. To mitigate negative transfer and better prioritize the main task, I propose substituting $\nabla\mathcal{L}$ with $\nabla\mathcal{L}_{\mathbf{main}} + \lambda G_{\mathbf{aux}}$ where $G_{\mathbf{aux}}$ is:

- as close as possible to $\nabla\mathcal{L}_{\mathbf{aux}}$ to preserve the insights brought by the auxiliary task;
- closer to $\nabla\mathcal{L}_{\mathbf{main}}$ than $-\nabla\mathcal{L}_{\mathbf{main}}$ to avoid following a direction detrimental to the main task.

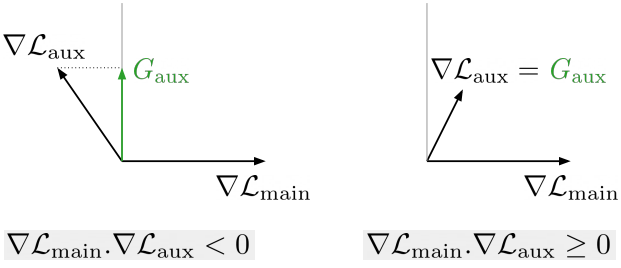

Figure 2. **Left**: If $\nabla\mathcal{L}_{\mathbf{main}}$ and $\nabla\mathcal{L}_{\mathbf{aux}}$ form an obtuse angle, they can significantly cancel each other. In this case, I propose to cancel the component of $\nabla\mathcal{L}_{\mathbf{aux}}$ parallel to $\nabla\mathcal{L}_{\mathbf{main}}$. **Right**: If not, I propose to keep $\nabla\mathcal{L}_{\mathbf{aux}}$ as it is.

Otherwise said, $G_{\mathbf{aux}}$ is the projection of $\nabla\mathcal{L}_{\mathbf{aux}}$ on the half-space of vectors whose dot product with $\nabla\mathcal{L}_{\mathbf{main}}$ is positive. If a critical point of $\mathcal{L}_{\mathbf{main}}$ has not been reached yet (i.e. if $\nabla\mathcal{L}_{\mathbf{main}} \neq 0$), this ensures that $\mathcal{L}_{\mathbf{main}}$ will decrease if the learning rate is small enough. This leads to Algorithm 1, which will be referred below as *Projection*.

Algorithm 1 *Projection* (learn a main task with the help of an auxiliary task)

1: **input**
- $\mathcal{L}_{\mathbf{main}}(y_1, \hat{y_1})$, a loss corresponding to the main task
- $\mathcal{L}_{\mathbf{aux}}(y_2, \hat{y_2})$, a loss corresponding to the auxiliary task
- λ, a weight for the auxiliary task
- $f(x, \theta) = (\hat{y_1}(x, \theta), \hat{y_2}(x, \theta))$, a neural network
- θ, an initial set of parameters for the neural network
- *Optimization Algorithm*, a gradient-descent based algorithm
- N, a total number of optimization steps

2: **for** $i = 1...N$ **do**
3: Compute $\nabla\mathcal{L}_{\mathbf{main}}$ through backpropagation
4: Compute $\nabla\mathcal{L}_{\mathbf{aux}}$ through backpropagation
5: **if** $\nabla\mathcal{L}_{\mathbf{main}}.\nabla\mathcal{L}_{\mathbf{aux}} \geq 0$ **then**
6: $G_{\mathbf{aux}} = \nabla\mathcal{L}_{\mathbf{aux}}$
7: **else**
8: $G_{\mathbf{aux}} = \nabla\mathcal{L}_{\mathbf{aux}} - (\nabla\mathcal{L}_{\mathbf{aux}}.\frac{\nabla\mathcal{L}_{\mathbf{main}}}{||\nabla\mathcal{L}_{\mathbf{main}}||})\frac{\nabla\mathcal{L}_{\mathbf{main}}}{||\nabla\mathcal{L}_{\mathbf{main}}||}$
9: **end if**
10: $G = \nabla\mathcal{L}_{\mathbf{main}} + \lambda G_{\mathbf{aux}}$
11: Update θ with G and *Optimization Algorithm*
12: **end for**
13: **return** θ

2.3. Dealing with noisy gradients

The assumption underlying this algorithm is that the direction of $\nabla\mathcal{L}_{\mathbf{main}}$ provides useful insights to mitigate negative transfer. This can hardly be the case if $\nabla\mathcal{L}_{\mathbf{main}}$ is very noisy, e.g. as a result of using small mini-batches. I therefore consider a variant where $\nabla\mathcal{L}_{\mathbf{main}}$ is replaced by its exponential moving average over time on lines 5 and 8 of Algorithm 1. The resulting algorithm is presented in Section A of the Supplementary Material and tested in the experiments of Section 4.

2.4. Applying *Projection* to the overall parameter vector or to each variable

The operations on gradients in Algorithm 1 are performed for the whole parameter vector θ. With machine learning frameworks like Tensorflow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019), it is however more straightforward to manipulate gradients for individual *variables*, one neural network being typically described by one weight variable and one bias variable for each layer. I then also consider a variant of Algorithm 1 where lines 3-10 are applied for each variable. Beside the ease of implementation,

this can also be more relevant if the gradients of the various variables are of significantly different magnitudes or if the transfer from the auxiliary task to the main task is positive for some layers but negative for others.

2.5. Beyond the minimization of an overall loss

For clarity of exposition, I presented the case of an overall loss linearly combining a main loss and an auxiliary loss. *Projection* is actually applicable more generally to the situations where θ is altered through the addition of two or more vectors, one of which corresponding to the main task. Examples of these situations are the second experiment (cf. 4.4), for which the main task and the auxiliary task are not performed on the same dataset, and the fourth experiment on weight decay (cf. 4.6), weight decay not being equivalent to $\mathcal{L}_2$ regularization when adaptive gradient algorithms such as Adam are used.

2.6. Complexity considerations

Compared to the standard multi-task approach, *Projection* requires to separately compute the gradient for each individual loss. As a result, the time and space requirements of the backpropagation phase scale linearly with the number of auxiliary tasks. This can become a hindrance for large neural networks with numerous auxiliary tasks.

3. Related work

3.1. Auxiliary tasks

Auxiliary tasks are used in the context of *sequential transfer learning* or *multi-task learning*. In the first case, training on the auxiliary tasks provides a pre-trained model, which is later fine-tuned on the main task. This has yielded spectacular results and become a standard practice in domains such as computer vision (Sharif Razavian et al., 2014) and natural language processing (Mikolov et al., 2013; Devlin et al., 2019). In the *multi-task learning* setting, which is the focus of this work, the auxiliary tasks and the main task are jointly learned. Examples provided in Section E of the Supplementary Material illustrate the diversity of the auxiliary tasks and the variety of situations in which they are used.

The standard multi-task approach consists in minimizing a linear combination of task-specific losses and is used pervasively, regardless of whether the intention is to perform well all tasks or only the main task. In a broadly equivalent variant, tasks-specific losses are minimized alternatively (Collobert & Weston, 2008; Jaderberg et al., 2017; Trinh et al., 2018).

3.2. Alternative methods to combine tasks

Several alternatives to the standard multi-task approach have been put forward. In the case where good performance on all tasks is looked for, Kendall et al. (2018), Chen et al. (2018) and Sener and Koltun (2018) propose to dynamically adjust the weights of the various tasks while Yu et al. (2020) suggest to also alter the directions of the corresponding gradients, following an approach very close to the one proposed here[3].

In the case of auxiliary tasks, there has been surprisingly little work aimed at replacing the standard multitask formulation in the general case of supervised learning. To the best of my knowledge, only Du et al. (2018) have explicitly addressed this issue. They propose to discard the gradient of the auxiliary loss if its angle with the gradient of the main loss is obtuse. In a work focused on algorithmic fairness, Zhang et al. (2018) aim at minimizing a prediction loss and maximizing an adversarial loss designed to enforce fairness while giving the priority to the latter. For this, they cancel the component of the gradient of the prediction loss parallel to the gradient of the adversary loss. These two approaches share commonalities with *Projection*: they also cancel all or part of the gradient of an auxiliary loss according to its angle with the gradient of a main loss. *Projection* can be seen as intermediate between them because it yields the same result as the former or the latter depending on whether this angle is acute or obtuse.

Lin et al. (2019) define a method to update the weights of the auxiliary tasks on the basis of the scalar products between the gradient of the main loss and the gradients of the auxiliary losses. This method is presented and tested in the context of reinforcement learning but could theoretically be applied to other forms of machine learning.

4. Experiments

4.1. Toy example

Before testing *Projection* on machine learning problems, I illustrate how censoring the gradient of a loss helps prioritize the decrease of another loss for a very simple multi-objective optimization problem. In the Euclidean plane $\mathbb{R}^2$, I aim at finding a point (x, y) that, above all, minimizes the square distance to the unit circle and, only if it does not increase this distance, also minimizes the square distance to point $(2, 0)$. I therefore define $\mathcal{L}_{\mathbf{main}} = (\sqrt{x^2 + y^2} - 1)^2$ and $\mathcal{L}_{\mathbf{aux}} = (x - 2)^2 + y^2$.

This is a slight departure from the principle of auxiliary tasks described above. Here, I want to accomplish both a

[3] A difference is that the method proposed by Yu et al. (2020) is adapted to the situation where we aim at performing well all tasks. Moreover they do not consider exponential smoothing.

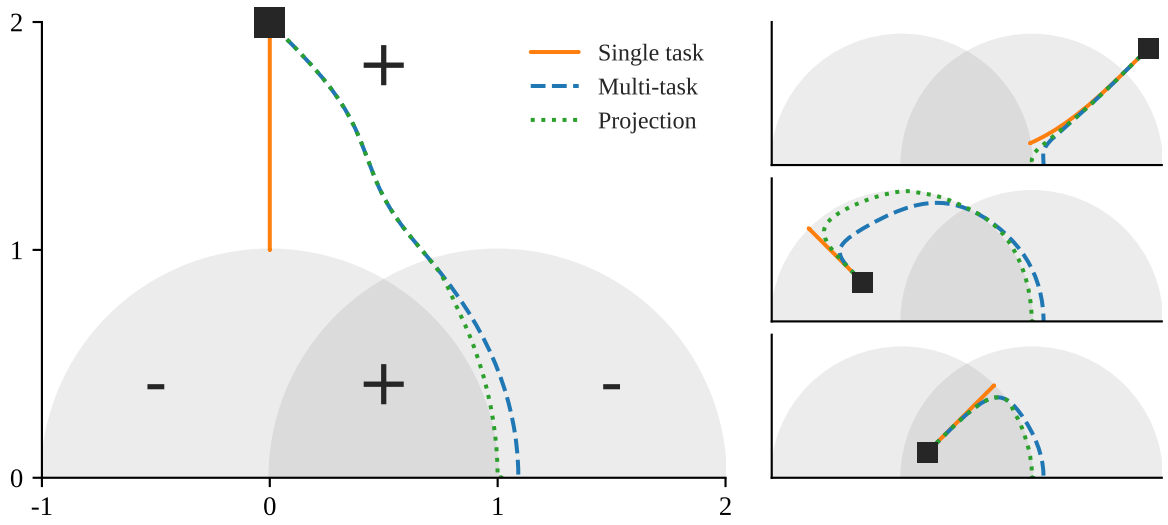

Figure 3. **Left**: In the toy example, I try to minimize both the square distance to the unit circle and the square distance to point $(2, 0)$, the first of these two objectives being my priority. The figure shows the trajectories corresponding to three methods when starting from point $(0, 2)$. The symbols $+$ and $-$ indicate the signs of the scalar products between $\nabla\mathcal{L}_{\mathbf{main}}$ and $\nabla\mathcal{L}_{\mathbf{aux}}$. **Right**: Trajectories with other starting points (■).

main task and an auxiliary task even if the main task is the absolute priority (performing the main task alone would be too easy). The solution of this problem is of course the point $(1, 0)$ and I try to reach it by starting from various starting points, using the Adam optimization algorithm (Kingma & Ba, 2015) and implementing 3 methods:

- *Single task*, where only $\mathcal{L}_{\mathbf{main}}$ is minimized;
- *Multi-task*, where $\mathcal{L}_{\mathbf{main}} + \lambda\mathcal{L}_{\mathbf{aux}}$ is minimized (here with $\lambda = 0.1$);
- *Projection*, applied to $\mathcal{L}_{\mathbf{main}}$, $\mathcal{L}_{\mathbf{aux}}$ and $\lambda = 0.1$.

Results are shown on Figure 3. The *Single task* trajectories quickly reach points of the unit disk close to the starting points. The *Multi-task* trajectories converge to a location close to but different from point (1, 0) because of the distraction created by the auxiliary task. The *Projection* and *Multi-task* trajectories are identical as long as the gradients of $\mathcal{L}_{\mathbf{main}}$ and $\mathcal{L}_{\mathbf{aux}}$ form an acute angle. Afterwards, the *Projection* trajectories slightly deviate from the *Multi-task* trajectories to reach point $(1, 0)$. This toy example shows that *Projection* can both properly take into account an auxiliary loss and give full priority to a main loss.

4.2. Experimental approach

I now compare *Projection* and its variants to *Multi-task* and *Single task* for a series of supervised learning problems. I choose *Multi-task* and *Single task* as baselines since alternative methods have not been tested for supervised learning problems or have not yielded consistently better results compared to the standard multi-task learning approach. More precisely:

- The methods proposed by Kendall et al. (2018), Chen et al. (2018) and Sener and Koltun (2018) aim at finding Pareto-optimal solutions for all tasks rather than the best solution for a given task. The experiments performed by Sener and Koltun (2018) show that these three methods do not perform noticeably better than *Multi-task* when the performance of only one task is focused on and the weights of the tasks are appropriately tuned;
- The methods proposed by Zhang et al. (2018) and Lin et al. (2019) were developed and evaluated in very specific contexts, respectively algorithmic fairness and reinforcement learning, which do not correspond to the settings of my experiments;
- The experiments conducted by Du et al. (2018) do not conclusively show whether their methods are superior to *Multi-task*.

I also provide additional experimental results with the alternative methods aiming at better exploiting auxiliary tasks in Section D of the Supplementary Material.

In the experiments below, I test *Projection*, *Multi-task*, and *Single task* for a wide range of values for λ. λ strongly influences whether the auxiliary task affects the outcome for the main task. Moreover, since *Projection* tends to reduce the norm of the gradient of the auxiliary loss, testing various values of λ is important to make sure that the differences observed do not simply result from this decrease.

Besides, since I focus on singling out the effects of censoring $\nabla\mathcal{L}_{\mathbf{aux}}$, I do not make specific efforts to achieve high predictive performance and I make "naive" design choices: I either use typical default hyper-parameters or reproduce the design of the original experiments that inspired mine.

For each of the experiments, I aim at answering the following questions:

- Do we achieve better predictive performance on the main task thanks to the auxiliary task?
- Does *Projection* yield better results than *Multi-task*?
- Does *Projection* influence the sensitivity of the predictive performance with regard to λ?

The full description of the experimental settings, the scripts necessary to reproduce the experiments and figures, as well as the raw experimental results are provided in Section B of the Supplementary Material and the associated repository.

4.3. Experiment 1: CelebA – classification and facial landmarks detection

The first experiment is based on the CelebA (or large-scale CelebFaces attributes) dataset (Liu et al., 2015) and inspired by Zhang et al (2014). The CelebA dataset contains face images annotated with 40 binary attributes such as "Attractive", "Young", "Eyeglasses" or "Black Hair", and the locations of five facial landmarks: left eye, right eye, left mouth corner, right mouth corner and nose.

Here, the main task is to determine whether a face picture was annotated as attractive and the auxiliary task is to locate the facial landmarks. The corresponding losses are the cross entropy error function and the average quadratic error. I downsample the pictures to 40×40 and convert them to gray-scale to make the tasks more challenging and reduce computation time. I include 10,000 of them in the training set, the validation set and the test set and used the same convolutional neural network architecture as Zhang et al. (2014). I train the neural network with the Adam algorithm and early stopping and I measure the Area Under the Receiver Operating Characteristic curve (AUC) for the primary task for 10 runs for each method and several values of λ. Figure 4 shows the corresponding results.

Unsurprisingly enough, the *Multi-task* and *Projection* curves get very close to the *Single task* curve as the influence of the auxiliary task becomes marginal for low λ values. The classification performance is better for *Multi-task* and *Projection* than for *Single task* for intermediate λ values. Moreover, the best classification performance achieved is slightly and statistically significantly higher for *Projection* than for *Multi-task* ($p = 0.03$[4]). For higher values of λ, classification performance decreases for both *Projection* and *Multi-task* but noticeably quicker for *Multi-task*.

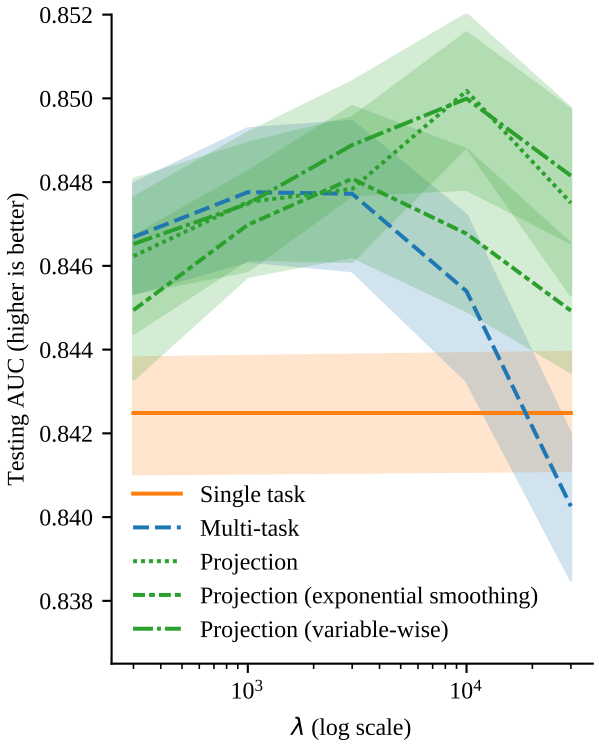

Figure 4. Testing AUC for the CelebA dataset in Experiment 1, *Single task*, *Multi-task* and *Projection*, and 5 values of λ (10 runs for each configuration). Two alternative variants of *Projection* are included: *Variable-wise* where *Projection* is applied to each "variable" (in the sense of Tensorflow or PyTorch, cf. Section 2.4) and *Exponential smoothing* ($\alpha = 0.01$) where $\nabla\mathcal{L}_{\mathbf{aux}}$ is compared to an exponential moving average of $\nabla\mathcal{L}_{\mathbf{main}}$ (cf. Section 2.3). The bands around the lines on this chart and the next ones represent the 95% confidence interval.

Figure 4 also shows the results for the two variants of *Projection* presented in Sections 2.3 and 2.4. I see that applying *Projection* to each variable rather than the whole parameter vector has limited effect. In contrast, replacing $\nabla\mathcal{L}_{\mathbf{main}}$ with an exponential moving average of $\nabla\mathcal{L}_{\mathbf{main}}$ ($\alpha = 0.01$) to alter $\nabla\mathcal{L}_{\mathbf{aux}}$ seems to lead to slightly lower performance.

[4] All p-values in this paper are computed with a permutation test.

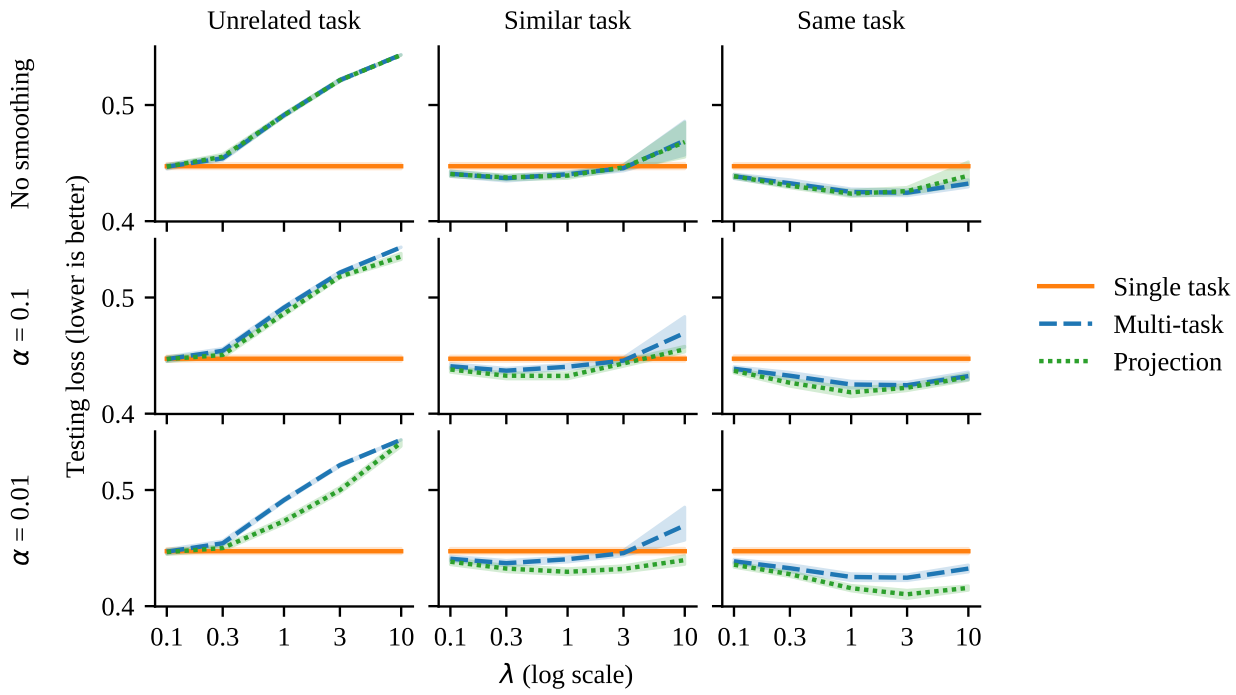

Figure 5. Testing loss for the synthetic dataset in Experiment 2. I test *Single task*, *Multi-task* and *Projection* for 5 values of λ, various levels of smoothing for $\nabla\mathcal{L}_{\mathbf{main}}$ and 3 auxiliary tasks which are unrelated, similar or identical to the main task (20 runs for each configuration).

4.4. Experiment 2: synthetic dataset – regression

For the second experiment, I follow Chen et al. (2018) and specify an artificial problem for which the adequacy of the auxiliary task can be controlled. I define:

- $f(B, x) = \tanh(Bx)$ where $\tanh(\cdot)$ acts element-wise, x is a real-valued vector of length 250 and B is a 100×250 real-valued matrix;
- $B_{\mathbf{main}}$, B and ϵ as constant 100×250 real-valued matrices whose elements are drawn i.i.d. respectively from $\mathcal{N}(0, 10)$, $\mathcal{N}(0, 10)$ and $\mathcal{N}(0, 3.5)$.

The main task and the auxiliary task consist in approximating respectively $f(B_{\mathbf{main}}, \cdot)$ and $f(B_{\mathbf{aux}}, \cdot)$ where $B_{\mathbf{aux}}$ can be equal to:

- $B_{\mathbf{main}}$ (both tasks are the same);
- $B_{\mathbf{main}} + \epsilon$ (both tasks are similar);
- B (both tasks are unrelated).

My loss function for both the main task and the auxiliary task is the mean squared error. For the input data (i.e. x), the training set, the validation set and the test set include respectively 1,000, 1,000 and 10,000 vectors whose elements are generated i.i.d. from $\mathcal{N}(0, 1)$. Moreover and as opposed to the other experiments, the training sets are distinct for the main task and the auxiliary task. Otherwise said, the training sets can be written as $\{(x_i, f(B_{\mathbf{main}}, x_i)), 1 \leq i \leq 1000\}$ and $\{(x'_i, f(B_{\mathbf{aux}}, x'_i)), 1 \leq i \leq 1000\}$ with $x_i \neq x'_i$. If these sets had the same x_i, the auxiliary task would not add any value when $B_{\mathbf{main}} = B_{\mathbf{aux}}$.

Like Chen et al. (2018), I use a multilayer perceptron with 4 shared RELU-activated hidden layers and 2 task-specific output layers, all of these layers being of size 100. I train this network with the Adam algorithm and early stopping and report the loss for the test set on Figure 5.

Without any smoothing of $\nabla\mathcal{L}_{\mathbf{main}}$ (top part of Figure 5), the results are very similar for *Multi-task* and *Projection*:

- when the auxiliary task is identical to the main task, it significantly contributes to the performance of the main task for all 5 values of λ;
- when the auxiliary task is similar to but different from the main task, it helps for $\lambda \leq 1$ but becomes neutral or harmful for $\lambda \geq 3$;
- B when the auxiliary task is unrelated to the main task,

it is detrimental to the performance of the main task for $\lambda \geq 0.3$.

When adjusting $\nabla\mathcal{L}_{\mathbf{aux}}$ on the basis of an exponential moving average of $\nabla\mathcal{L}_{\mathbf{main}}$, instead of simply $\nabla\mathcal{L}_{\mathbf{main}}$ (cf. Section 2.3), *Projection* yields markedly better results than *Multi-task*, in particular for $\alpha = 0.01$ (for example, $p < 10^{-5}$ when the auxiliary task and the main task are the same and $p = 0.02$ when they are similar). We can also see on Figure 8 in the Supplementary Material that applying *Projection* to the whole parameter vector or each "variable" (in the sense of Tensorflow or PyTorch, cf. section 2.4) does not significantly affect the results. For the following experiments, I apply *Projection* to each variable rather than the whole parameter vector.

4.5. Experiment 3: CoNLL 2003 – text chunking and part-of-speech tagging

The third experiment concerns natural language processing with text chunking and part-of-speech tagging as main and auxiliary tasks. Part-of-speech tagging refers to classifying words into syntactic categories, such as "adjective", "adverb" or "personal pronoun" while text chunking consists in "dividing a text into phrases in such a way that syntactically related words become member of the same phrase" (Sang & Buchholz, 2000).

The text corpus used is the English named entity recognition dataset of the 2003 CoNLL Shared Task (Tjong Kim Sang & De Meulder, 2003). It includes news wire articles from the Reuters Corpus. Each word of these articles has been annotated with a part-of-speech tag, a chunk tag and a named entity recognition tag (ignored here).

I use a neural network composed of an embedding layer of size 300 followed by two LSTM layers (Hochreiter & Schmidhuber, 1997) of size 128 shared across the two tasks and two task-specific output layers. Søgaard and Goldberg (2016) showed that low level auxiliary tasks (such as part-of-speech tagging) are more helpful to better achieve higher level main tasks (such as chunking) when parameters are shared only for the innermost layers. Here, I purposely share all parameters except for the output layers to increase the risk of negative transfer. I train the neural network during 300 epochs with the Adam algorithm, a batch size of 128, gradient clipping (with a value of 1) and dropout ($P = 0.2$).

Figure 6 shows the testing accuracy on the main task for *Single task*, *Multi-task* and *Projection*, and 9 values of λ. Like in the previous experiments, the impact of the auxiliary task is respectively negligible, positive and strongly negative for low, intermediate and high values of λ. Without exponential smoothing, *Multi-task* and *Projection* are almost indistinguishable. With exponential smoothing ($\alpha = 0.01$), I obtain a significantly higher peak value for the testing accuracy for

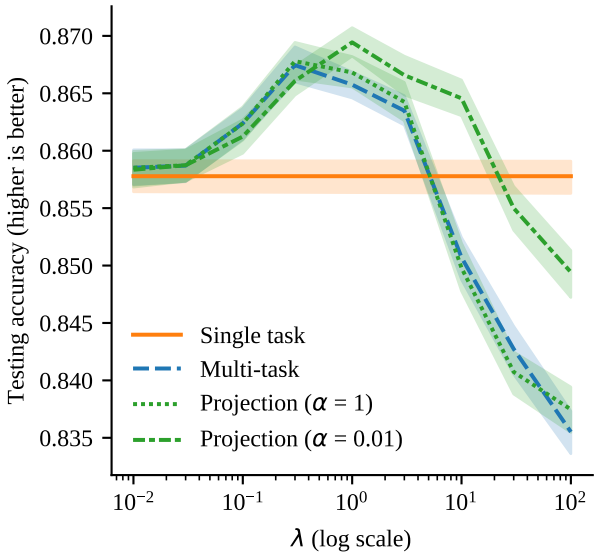

Figure 6. Testing accuracy for text chunking on the English CoNLL 2003 dataset in Experiment 3, for *Single task*, *Multi-task* and *Projection* (with or without exponential smoothing), and 9 values of λ (50 runs for each configuration).

Projection compared to *Multi-task* ($p = 0.03$) and this score is less degraded by excessive λ values.

4.6. Experiment 4: IMDB movie reviews – sentiment classification

In the last experiment, directly inspired by Chollet (2017), I test whether *Projection* could help weight decay enforce better regularization.

I perform binary sentiment classification on a corpus of IMDB movie reviews. The training set, validation set and test set respectively include 20,000, 5,000 and 25,000 reviews pre-processed with multi-hot encoding for the 10,000 most frequent words. The neural network is a multilayer perceptron with two RELU-activated hidden layers of size 512 and followed by a single logistic neuron. I deliberately choose excessively wide hidden layers to get overfitting and I try to achieve regularization through weight decay.

The loss for the main task is the binary cross-entropy. I train the neural network with the AdamW optimization algorithm (Loshchilov & Hutter, 2019), a batch size of 512, early stopping and weight decay. I do not truly have an auxiliary loss[5] but I replace $\nabla_\theta\mathcal{L}_{\mathbf{aux}}$ with θ for the application of *Projection*.

[5]Since the neural network is not trained with standard stochastic gradient descent, weight decay regularization and $\mathcal{L}_2$ regularization are not equivalent but here, θ plays a similar role as $\nabla_\theta\mathcal{L}_{\mathbf{aux}}$.

Figure 7 shows that weight decay has limited regularization effect for low values of λ and both *Multi-task* and *Projection* without exponential smoothing. However, high values of λ severely degrade the performance metric. In contrast, *Projection* with exponential smoothing ($\alpha = 0.01$) yields a statistically significant improvement ($p < 10^{-5}$) and strongly mitigates the increase of the testing loss for very high λ values.

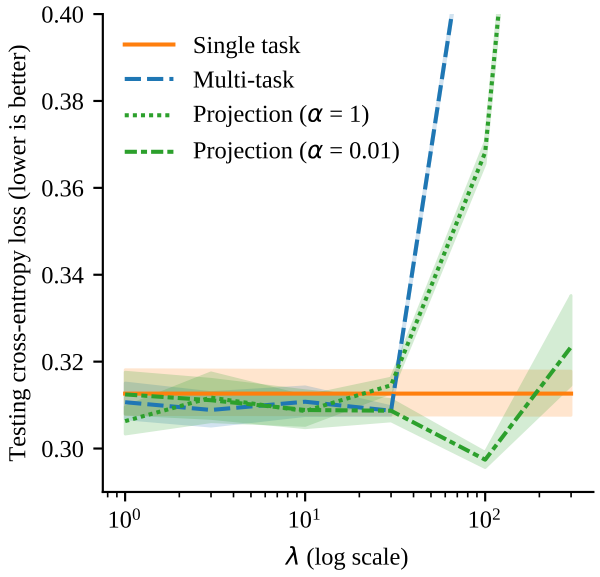

Figure 7. Testing cross-entropy loss in Experiment 4 for *Single task*, *Multi-task* and *Projection* (with or without exponential smoothing), and 6 values of λ (50 runs for each configuration).

5. Conclusions and future work

Summary of findings. This work focuses on the specific case of multi-task learning where we jointly learn a main task and one or several auxiliary tasks in the hope of achieving better performance for the main task. In this case, the standard multi-task learning approach does not adequately reflect the asymmetry between the main task and the auxiliary tasks. From this observation, I proposed an alternative method, *Projection*, designed to prioritize the main task and thus reduce the risks of negative transfer. In a series of experiments I highlighted various forms of negative transfer and showed that the proposed method could noticeably mitigate them:

- Negative transfer caused by an irrelevant auxiliary task (cf. the situation in the second experiment where the the auxiliary task is irrelevant by design);
- Negative transfer due to an excessive weighting of an auxiliary task (cf. all experiments for high λ values);
- Intermittent negative transfer of an otherwise useful auxiliary task: the fact that partially censoring $\nabla_\theta \mathcal{L}_{\mathbf{aux}}$ led to better results than the standard multi-task approach suggests that an auxiliary task is sometimes harmful even when its overall contribution is positive.

Otherwise said, *Projection* helped achieve better generalization performance when λ is appropriately tuned but also improved the resilience to an inadequate or improperly weighted auxiliary task.

Practical recommendations. Given these encouraging results and the ease of implementation of *Projection*, I recommend systematically testing it as an alternative to the standard multi-task approach when the number of auxiliary tasks is limited. The benefits are more uncertain with numerous auxiliary tasks: on the negative side, the time and space complexity of *Projection* scales linearly with their number but on the positive side, the reduced sensitivity to tasks weights becomes more important when it is more challenging to finetune them.

We also saw that using an exponential moving average of $\nabla_\theta \mathcal{L}_{\mathbf{main}}$ may be decisive to reap the benefits of *Projection*. This should then be considered. This leads to an additional hyperparameter to tune but simply using $\alpha = 0.01$ provided good results in the experiments. Finally the experiments further suggest that applying *Projection* at the level of the *variables* (in the sense of Tensorflow or PyTorch) rather than the whole parameter vector is sufficient, which makes the implementation more straightforward.

Directions for future work. Given the range of possible auxiliary tasks and the variety of contexts in which they may be helpful, the proposed method should be further tested to confirm its benefits and better understand the circumstances in which it is most adequate. Additional experiments could notably focus on reinforcement learning problems and cover cases where the auxiliary tasks are translated to auxiliary losses or to additional rewards.

In this work I did not analyze the mechanisms through which the auxiliary tasks lead to better generalization performance. Auxiliary tasks could contribute at various stages of the learning process by providing a promising starting point, by facilitating the navigation through the loss landscape or by favoring regularization during the final optimization steps. Auxiliary tasks could also help at some point in this process but harm at another point. We would ideally identify examples of these various situations and test whether *Projection* performs well in all of them or should be adapted.

We could also depart further from the standard multi-task approach and consider alternative ways to combine the main and auxiliary losses. A natural choice is to choose a function

of several real variables increasing along all its axes but there is no strong reason to restrict ourselves to linear functions.

Acknowledgements

Many thanks to Antonis Polykratis and François Chollet for having made available code samples I adapted for Experiments 3 and 4.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. In *ICLR*, 2019.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Caruana, R., Baluja, S., and Mitchell, T. Using the future to" sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *Advances in neural information processing systems*, pp. 959–965, 1996.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pp. 793–802, 2018.

Chollet, F. *Deep Learning with Python*. Manning, 2017.

Chollet, F. et al. Keras. https://keras.io, 2015.

Chronopoulou, A., Baziotis, C., and Potamianos, A. An embarrassingly simple approach for transfer learning from pretrained language models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2089–2095, 2019.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

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

Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G., and Larochelle, H. Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*, 2019.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=SJ6yPD5xg.

Kendall, A., Gal, Y., and Cipolla, R. 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*, pp. 7482–7491, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

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

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pp. 3730–3738, 2015.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. Curran Associates, Inc., 2013.

Mordan, T., Thome, N., Henaff, G., and Cord, M. Revisiting multi-task learning with rock: a deep residual auxiliary block for visual detection. In *Advances in Neural Information Processing Systems*, pp. 1310–1322, 2018.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Plank, B., Søgaard, A., and Goldberg, Y. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 412–418, 2016.

Pomerleau, D. A. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pp. 305–313, 1989.

Rosenstein, M. T., Marx, Z., Kaelbling, L. P., and Dietterich, T. G. To transfer or not to transfer. In *NIPS'05 Workshop, Inductive Transfer: 10 Years Later*. Citeseer, 2005.

Ruder, S. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. URL http://arxiv.org/abs/1706.05098.

Sang, E. T. K. and Buchholz, S. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop (CONLL/LLL 2000). Lissabon, Portugal, 13-14 september 2000*, pp. 127–132. ACL, 2000.

Sejnowski, T. J. and Rosenberg, C. R. Parallel networks that learn to pronounce english text. *Complex systems*, 1(1): 145–168, 1987.

Sener, O. and Koltun, V. Multi-task learning as multi-objective optimization. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 527–538. Curran Associates, Inc., 2018.

Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Søgaard, A. and Goldberg, Y. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 231–235, 2016.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Tjong Kim Sang, E. F. and De Meulder, F. Introduction to the conll-2003 shared task: language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pp. 142–147. Association for Computational Linguistics, 2003.

Trinh, T. H., Dai, A. M., Luong, T., and Le, Q. V. Learning longer-term dependencies in rnns with auxiliary losses. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4972–4981. PMLR, 2018. URL http://proceedings.mlr.press/v80/trinh18a.html.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

Zhang, B. H., Lemoine, B., and Mitchell, M. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 335–340, 2018.

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

Supplementary Material

A. Variant of Algorithm 1 with an exponential moving average

Algorithm 2 *Projection (with exponential smoothing)* (learn a main task with the help of an auxiliary task)

input
- $\mathcal{L}_{\mathbf{main}}(y_1, \hat{y_1})$, a loss corresponding to the main task
- α, a smoothing parameter for $\nabla\mathcal{L}_{\mathbf{main}}$
- $\mathcal{L}_{\mathbf{aux}}(y_2, \hat{y_2})$, a loss corresponding to the auxiliary task
- λ, a weight for the auxiliary task
- $f(x, \theta) = (\hat{y_1}(x, \theta), \hat{y_2}(x, \theta))$, a neural network
- θ, an initial set of parameters for the neural network
- *Optimization Algorithm*, a gradient-descent based algorithm
- N, a total number of optimization steps

$G_{\mathbf{main}}(0) = 0$
for $i = 1...N$ **do**
  Compute $\nabla\mathcal{L}_{\mathbf{main}}$ through backpropagation
  Compute $\nabla\mathcal{L}_{\mathbf{aux}}$ through backpropagation
  $G_{\mathbf{main}}(i) = \alpha\nabla\mathcal{L}_{\mathbf{main}} + (1-\alpha)G_{\mathbf{main}}(i-1)$
  if $G_{\mathbf{main}}(i).\nabla\mathcal{L}_{\mathbf{aux}} \geq 0$ **then**
    $G_{\mathbf{aux}} = \nabla\mathcal{L}_{\mathbf{aux}}$
  else
    $G_{\mathbf{aux}} = \nabla\mathcal{L}_{\mathbf{aux}} - (\nabla\mathcal{L}_{\mathbf{aux}} \cdot \frac{G_{\mathbf{main}}(i)}{||G_{\mathbf{main}}(i)||}) \frac{G_{\mathbf{main}}(i)}{||G_{\mathbf{main}}(i)||}$
  end if
  $G = \nabla\mathcal{L}_{\mathbf{main}} + \lambda G_{\mathbf{aux}}$
  Update θ with G and *Optimization Algorithm*
end for
return θ

B. Experimental settings

B.1. Toy example

Loss functions

- $\mathcal{L}_{\mathbf{main}} = (\sqrt{x^2 + y^2} - 1)^2$
- $\mathcal{L}_{\mathbf{aux}} = (x-2)^2 + y^2$
- Weight for the auxiliary task: $\lambda = 0.1$

Optimization algorithm: Adam with the default PyTorch parameters ($\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-8}$

B.2. Experiment 1: CelebA – classification and facial landmarks detection

Data

- Dataset: CelebA (Liu et al., 2015)
- images downsampled to 40×40, converted to grayscale and rescaled to $[0, 1]$
- 10,000 images randomly selected for the training set, the validation set and the test set

Loss functions

- $\mathcal{L}_{\mathbf{main}}$: cross entropy for the "Attractive" attribute
- $\mathcal{L}_{\mathbf{aux}}$: mean squared error for the 5 facial landmarks
- λ values tested: 300, 1000, 3000, 10000, 30000

Network architecture: same as Zhang et al. (2014), cf. Section 4 and Figure 3, with RELU activation functions for all hidden layers

Optimization procedure

- Algorithm: Adam with the default Tensorflow parameters ($\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-7}$)
- Batch size: 100
- Stopping criterion: early stopping, patience = 10

Number of runs for each configuration: 10

B.3. Experiment 2: synthetic dataset – regression

Data

- Dataset: same as Chen et al. (2018), cf. Section 4
- 1,000, 1,000, 1,000 and 10,000 observations generated i.i.d. for respectively the training set for the main task, the training set for the auxiliary task, the validation set and the test set

Loss functions

- $\mathcal{L}_{\mathbf{main}}$: mean squared error
- $\mathcal{L}_{\mathbf{aux}}$: mean squared error
- λ values tested: 0.1, 0.3, 1, 3, 10

Network architecture: same as Chen et al. (2018), cf. Section 4

Optimization procedure

- Algorithm: Adam with the default Tensorflow parameters ($\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-7}$)
- Batch size: 100
- Stopping criterion: early stopping, patience = 10

Number of runs for each configuration: 20

B.4. Experiment 3: CoNLL 2003 – text chunking and part-of-speech tagging

Data

- Dataset: English named entity recognition dataset of the 2003 CoNLL Shared Task (Tjong Kim Sang & De Meulder, 2003)
- Same partition into the training set, the validation set and the test set as the initial dataset

Loss functions

- $\mathcal{L}_{\mathbf{main}}$: cross entropy on the chunk tag
- $\mathcal{L}_{\mathbf{aux}}$: cross entropy on the part-of-speech tag
- λ values tested: 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100

Network architecture: embedding layer of size 300 followed by two LSTM layers of size 128 shared across the two tasks and two task-specific output layers, with dropout for the hidden layers ($P = 0.2$)

Optimization procedure

- Algorithm: Adam with a learning rate of 0.0001 and the default PyTorch parameters otherwise ($\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-8}$)
- Batch size: 128
- Gradient clipping: 1
- Stopping criterion: early stopping with a patience of 25 and a maximum number of epochs of 40

Number of runs for each configuration: 50

B.5. Experiment 4: IMDB movie reviews – sentiment classification

Data

- Dataset: IMDB Movie reviews sentiment classification dataset obtained through Keras (Chollet et al., 2015)
- Same test set as the initial dataset (25,000 reviews), training set of the initial dataset divided into a training set (20,000 reviews) and a validation set (5,000 reviews)
- Reviews multi-hot-encoded with a vocabulary size of 10,000

Loss functions: cross entropy on the binary sentiment classification task

- $\mathcal{L}_{\mathbf{main}}$: cross entropy
- $\mathcal{L}_{\mathbf{aux}}$: mean squared error
- Weight for the auxiliary loss: $\lambda \in [0.1, 0.3, 1, 3, 10]$

Network architecture: two RELU-activated fully-connected hidden layers of size 512 followed by a single logistic neuron

Optimization procedure

- Algorithm: Adam with the default Tensorflow parameters ($\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-7}$)
- Batch size: 512
- Stopping criterion: early stopping, patience = 5

Number of runs for each configuration: 50

All experiments were performed with a Tesla P100 GPU and either Tensorflow or PyTorch.

C. Additional experimental results

Figure 8 shows that applying *Projection* to the whole parameter vector or each "variable" in Experiment 2 does not really affect performance when the main and auxiliary tasks are the same and $\alpha = 0.01$. Similar results are obtained for other α values or the two other auxiliary tasks.

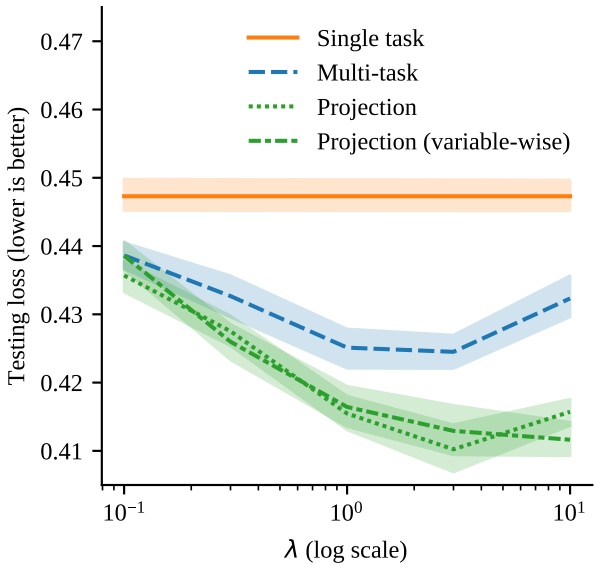

Figure 8. Illustration of the impact of applying *Projection* for the whole parameter vector or each "variable" (in the sense of Tensorflow or PyTorch) with the synthetic dataset, an auxiliary task identical to the main task and $\alpha = 0.01$.

D. Comparison with alternative methods

D.1. *Unweighted cosine* and *Weighted cosine* (Du et al., 2018)

Du et al. (2018) propose two methods, *Unweighted cosine* and *Weighted cosine*, that also modify $\nabla\mathcal{L}_{\mathbf{aux}}$ according to its angle with $\nabla\mathcal{L}_{\mathbf{main}}$.

As illustrated on Figure 9, if $\nabla\mathcal{L}_{\mathbf{main}}$ and $\nabla\mathcal{L}_{\mathbf{aux}}$ form an acute angle, both *Projection* and *Unweighted cosine* leave $\nabla\mathcal{L}_{\mathbf{aux}}$ unchanged whereas *Weighted cosine* reduces its norm with the cosine similarity of these vectors. In particular, *Weighted cosine* strongly shrinks $\nabla\mathcal{L}_{\mathbf{aux}}$ when it becomes almost orthogonal to $\nabla\mathcal{L}_{\mathbf{main}}$.

If $\nabla\mathcal{L}_{\mathbf{main}}$ and $\nabla\mathcal{L}_{\mathbf{aux}}$ form an obtuse angle, both *Unweighted cosine* and *Weighted cosine* just ignore $\nabla\mathcal{L}_{\mathbf{aux}}$. In contrast, *Projection* only negates the component of $\nabla\mathcal{L}_{\mathbf{aux}}$ which is collinear to $\nabla\mathcal{L}_{\mathbf{main}}$.

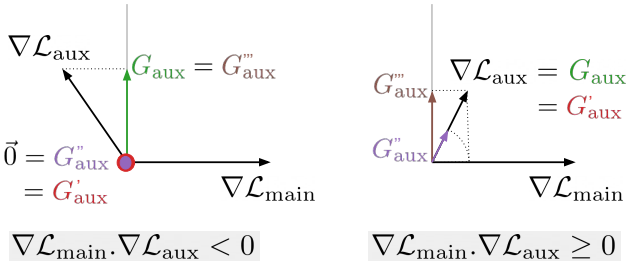

Figure 9. Transformation of $\nabla\mathcal{L}_{\mathbf{aux}}$ with *Projection* ($G_{\mathbf{aux}}$ in green), *Unweighted cosine* ($G'_{\mathbf{aux}}$ in red), *Weighted cosine* ($G''_{\mathbf{aux}}$ in purple) and *Orthogonal* ($G'''_{\mathbf{aux}}$ in brown).

Even if *Projection* on one hand and *Unweighted cosine* and *Weighted cosine* on the other hand appear similar, their motivations are different. Du et al. (2018) explicitly try to favor auxiliary tasks similar to the primary task. For example, *Weighted cosine* completely ignores the auxiliary task when $\nabla\mathcal{L}_{\mathbf{main}}$ and $\nabla\mathcal{L}_{\mathbf{aux}}$ are orthogonal. In contrast, *Projection* does not discard $\nabla\mathcal{L}_{\mathbf{aux}}$ when it is very dissimilar to $\nabla\mathcal{L}_{\mathbf{main}}$. *Projection* only cancels the component of $\nabla\mathcal{L}_{\mathbf{aux}}$ which seems harmful to the main objective. This allows the auxiliary task to bring a new perspective and nudge the learning process towards interesting regions of the parameter space that would not be explored with the main task alone.

Another distinction is that Du et al. (2018) effectively set λ to 1 when using *Unweighted cosine* and *Weighted cosine*. On the contrary, I consider that λ still remains to be tuned given that the relative scales of $\mathcal{L}_{\mathbf{main}}$ and $\mathcal{L}_{\mathbf{aux}}$ are arbitrary. The subsequent experimental results show that λ indeed strongly affects the performance of *Unweighted cosine* and *Weighted cosine*.

D.2. *Orthogonal*, inspired from (Zhang et al., 2018)

Zhang et al. (2018) try to prioritize a main loss over a secondary loss by canceling the component of the gradient of the secondary loss parallel to the gradient of the main loss. This situation is slightly different from the one considered here because Zhang et al. (2018) aim at minimizing the secondary loss, which is then not an auxiliary loss, provided that this does not substantially increase the main loss. Moreover they propose this only in the specific context of algorithmic fairness. Still, I test below this approach given its similarities with *Projection* and I refer to it as *Orthogonal*.

D.3. Experimental results for *Unweighted cosine*, *Weighted cosine* and *Orthogonal*

Figures 10, 11, 12 and 13 show the results of the four experiments of Section 4 for *Unweighted cosine*, *Weighted cosine* and *Orthogonal*.

Without exponential smoothing, *Orthogonal* performs comparably to or notably worse than *Multitask*. In particular, *Orthogonal* yields very poor results for high λ values for the first and second experiments.

With exponential smoothing ($\alpha = 0.01$), *Orthogonal* yields slightly better results than *Projection* for the first experiment ($p = 0.11$), the second experiment when the main and auxiliary tasks are similar ($p = 0.23$) and the third experiment ($p = 0.07$) but in these cases, the outcome for high λ values is much worse. In contrast, *Projection* performs better than *Orthogonal* for the second experiment when the auxiliary and main tasks are the same ($p = 0.02$) and the fourth experiment ($p = 0.02$).

Unweighted cosine and *Weighted cosine* perform generally worse than *Projection*. An exception is the second experiment when the auxiliary and main tasks are unrelated. In this case, *Weighted cosine* and sometimes *Unweighted cosine* prove much more resilient to the inadequate auxiliary task. This is unsurprising because these two methods are more circumspect in their use of the auxiliary task. The other exception is the third experiment for which *Weighted cosine* yields better results than the other methods (only slightly compared to *Orthogonal*).

Overall, when taking into account both the peak performance and the performance at high λ values over all experiments, *Projection* compares favorably with *Unweighted cosine*, *Weighted cosine* and *Orthogonal*. These results reinforce two conclusions stated of Section 5. Firstly, using an exponential moving average of $\mathcal{L}_{\mathbf{main}}$ can have a decisive impact on performance and should then be considered. Secondly, the respective behaviors of *Projection*, *Unweighted cosine*, *Weighted cosine* and *Orthogonal* are sometimes dissimilar from one experiment to another. For example for high λ values, *Unweighted cosine* and *Weighted cosine* give

very poor results in Experiment 1 but the best results for Experiment 4. This suggests that the transfer from the auxiliary task to the main task follows a different mechanism for these two experiments. A more in-depth analysis would be required to distinguish these various forms of transfer.

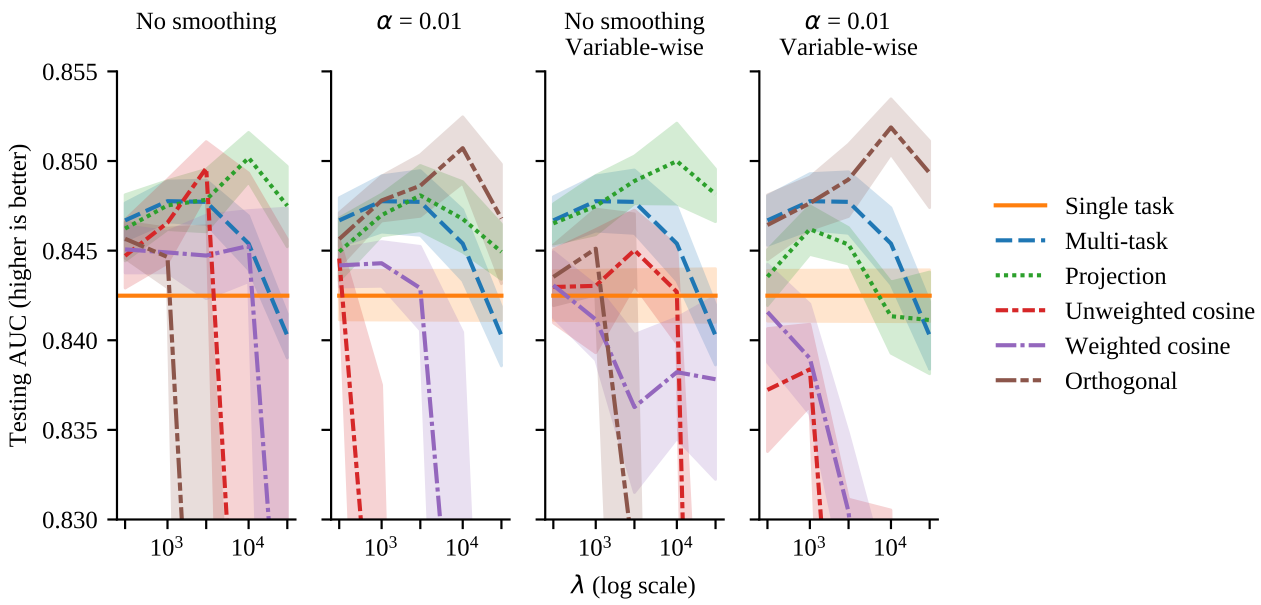

Figure 10. Testing AUC for the CelebA dataset in Experiment 1, *Single task*, *Multi-task*, *Projection*, *Unweighted cosine*, *Weighted cosine* and *Orthogonal*, and 5 values of λ (10 runs for each configuration). Both variants presented in Sections 2.3 and 2.4 are tested.

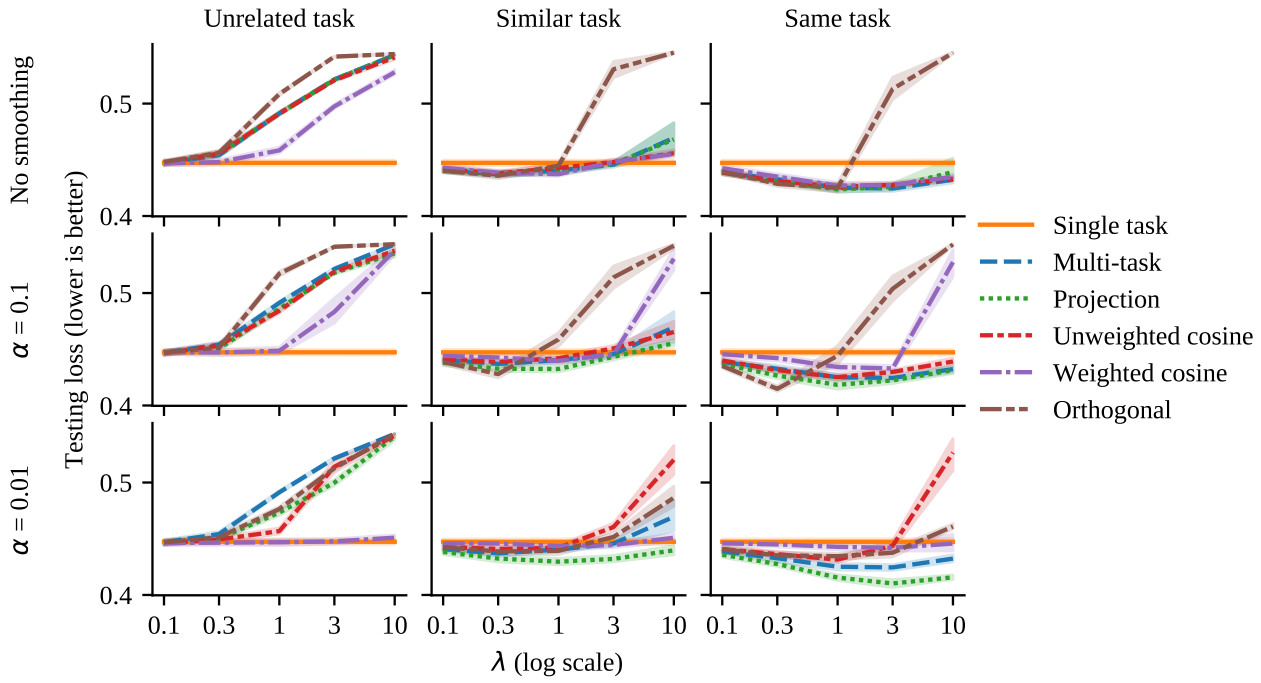

Figure 11. Testing loss for the synthetic dataset in Experiment 2. I test *Single task*, *Multi-task*, *Projection*, *Unweighted cosine*, *Weighted cosine* and *Orthogonal* for 5 values of λ, various levels of exponential smoothing for $\nabla\mathcal{L}_{\mathbf{main}}$ and 3 auxiliary tasks which are unrelated, similar or identical to the main task (20 runs for each configuration).

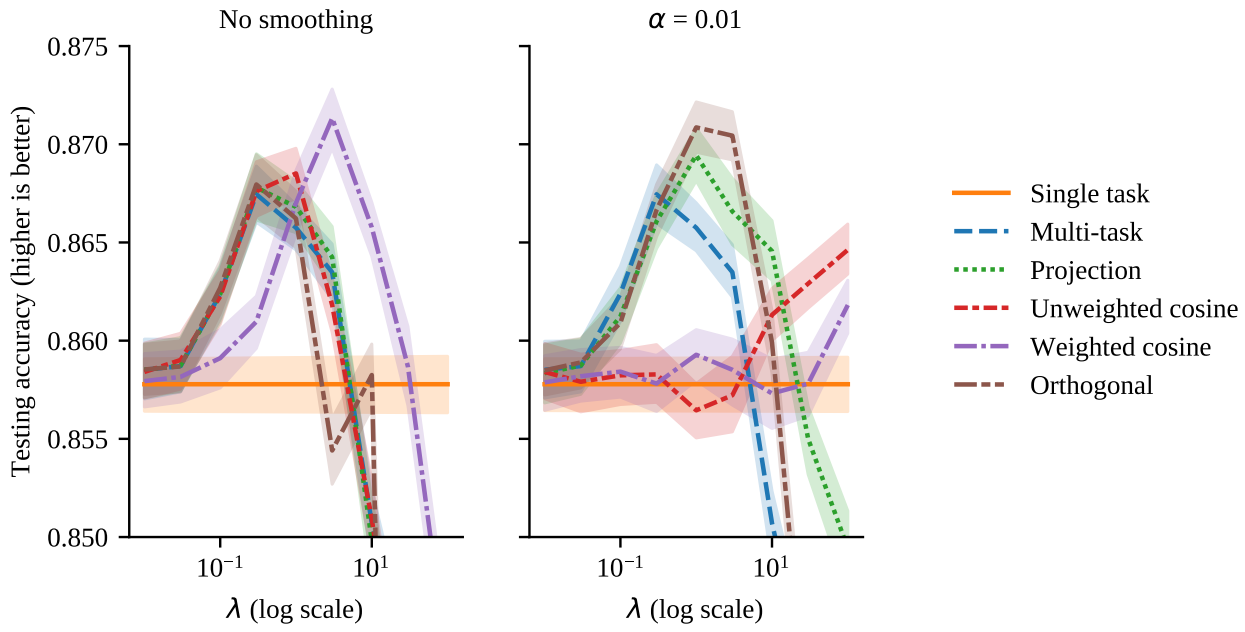

Figure 12. Testing accuracy for text chunking on the English CoNLL 2003 dataset in Experiment 3, for *Single task*, *Multi-task*, *Projection*, *Unweighted cosine*, *Weighted cosine* and *Orthogonal* (with or without exponential smoothing), and 9 values of λ (50 runs for each configuration).

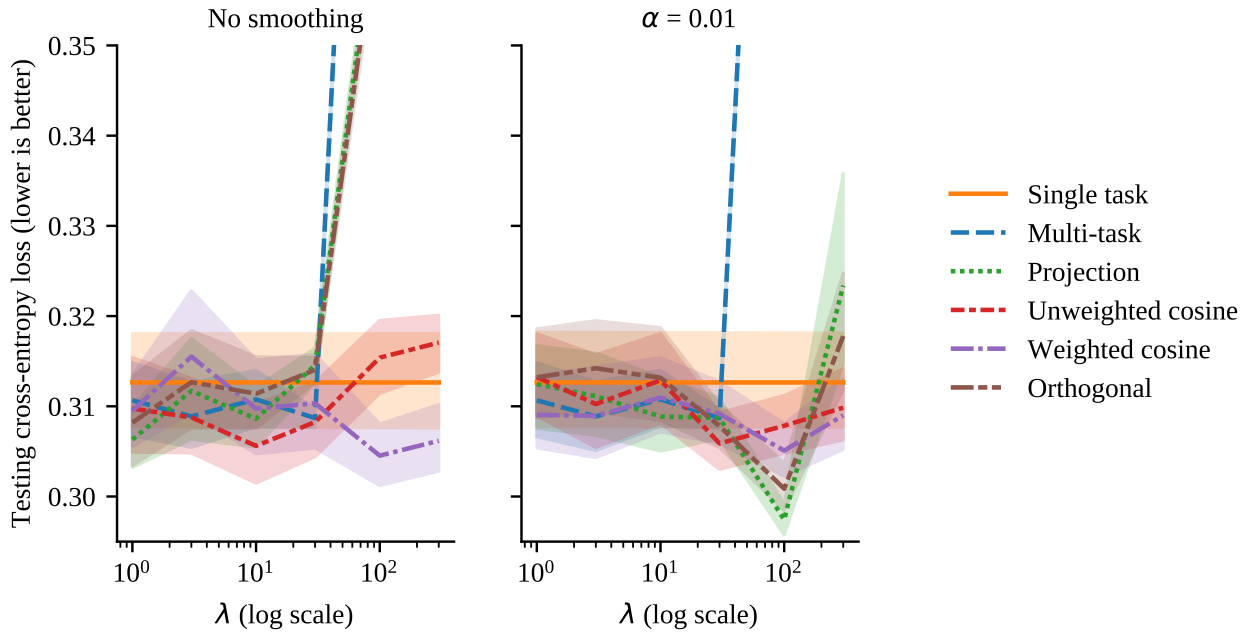

Figure 13. Testing cross-entropy loss in Experiment 4 for *Single task*, *Multi-task*, *Unweighted cosine*, *Weighted cosine* and *Orthogonal Projection* (with or without exponential smoothing), and 6 values of λ (50 runs for each configuration).

D.4. Adaptive auxiliary task weighting (Lin et al., 2019)

Lin et al. (2019) approximate the decrease of the main loss over N optimization steps and derive an update rule for the weights of the auxiliary tasks to maximize this approximated decrease. This method, *Adaptive auxiliary task weighting*, dynamically adjusts the weights of the auxiliary losses. It does not transform the gradients of these losses, unlike *Projection*, *Unweighted cosine*, *Weighted cosine* and *Orthogonal*, and could in principle be used in conjunction with one of these methods.

Adaptive auxiliary task weighting consists in two loops, one updating the neural network's parameters and one adjusting the weights of the auxiliary losses. It requires three hyperparameters: a learning rate for the neural network's parameters, a learning rate β for the weights of the auxiliary losses and a horizon N, which is the number of the first loop's iterations between the second loop's iterations.

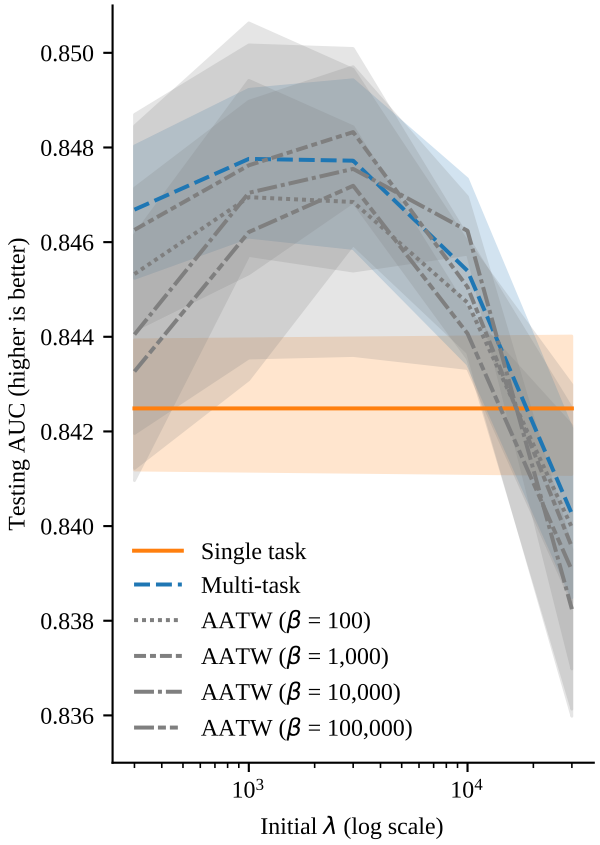

Figure 14. Testing AUC for the CelebA dataset in Experiment 1 and *Single task*, *Multi-task* and *Adaptive auxiliary task weighting*. Here, the logarithm function is applied to $\mathcal{L}_{\mathbf{main}}$ and $\mathcal{L}_{\mathbf{aux}}$ as suggested by Lin et al. (2019).

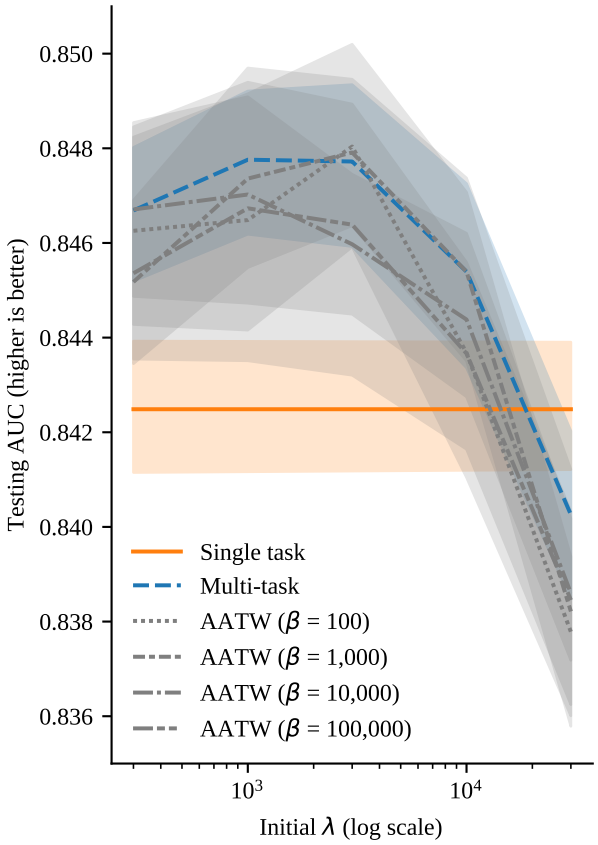

Figure 15. Testing AUC for the CelebA dataset in Experiment 1 and *Single task*, *Multi-task* and *Adaptive auxiliary task weighting*. For this figure, the logarithm function is not applied to $\mathcal{L}_{\mathbf{main}}$ or $\mathcal{L}_{\mathbf{aux}}$.

I test *Adaptive auxiliary task weighting* in the context of Experiment 1. I use the the Adam algorithm with the same learning rate (0.001) as before, $N = 5$ like Lin et al. (2019) and four values of β from 100 to 100,000. Lin et al. (2019) apply a logarithm function to $\mathcal{L}_{\mathbf{main}}$ or $\mathcal{L}_{\mathbf{aux}}$ in their algorithm. I try with and without this logarithm function and show the corresponding results in Figures 14 and 15. I did not find a statistically significant improvement over *Multi-task*.

E. Examples of auxiliary tasks

Table 1. Examples of auxiliary tasks.

Domain	Primary task	Auxiliary tasks	Source
Medical diagnosis	Identifying high risk pneumonia patients	Predicting the results of lab tests (recorded in the training dataset but not available at prediction time)	(Caruana et al., 1996)
Computer vision	Classifying images	Classifying the same images as for the primary task but at an intermediate layer of the neural network	(Szegedy et al., 2015)
Computer vision	Identifying facial landmarks on face pictures	Estimating head pose and predicting facial attributes ("wearing glasses", "smiling" and "gender")	(Zhang et al., 2014)
Computer vision	Detecting objects in indoor scenes	Predicting scene labels and evaluating depth and surface orientation at pixel level	(Mordan et al., 2018)
NLP	Determining the semantic role of syntactic constituents of a sentence	Performing POS tagging, chunking, named entity recognition and language modelling	(Collobert & Weston, 2008)
NLP	Annotating multilingual texts with part-of-speech tags	Predicting the log frequency of words	(Plank et al., 2016)
NLP	Classifying text	Predicting a word given its predecessors	(Chronopoulou et al., 2019)
Sequence modelling	Training a recurrent neural network on very long sequences	Reconstructing or predicting short segments of the sequences	(Trinh et al., 2018)
Reinforcement learning	Playing a video game	Modifying the image perceived by the agent and predicting short-term rewards	(Jaderberg et al., 2017)
Reinforcement learning	Playing a video game	Predicting the future state based on the current state and the current action	(Burda et al., 2019)
Reinforcement learning	Playing a video game	Computing Q-values over various time horizons	(Fedus et al., 2019)
Reinforcement learning	Performing robotic control tasks	Imitating demonstrations by humans	(Nair et al., 2018)