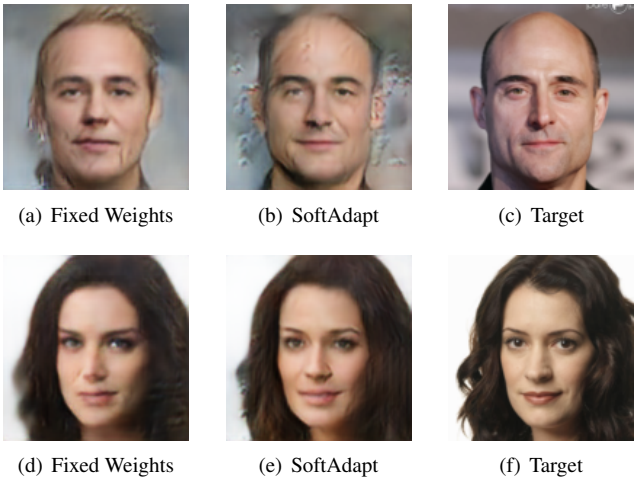
SoftAdapt: Techniques for Adaptive Loss Weighting of Neural Networks with Multi-Part Loss Functions

A. Ali Heydari[1] aheydari@ucmerced.edu
Craig A. Thompson[2] craigthompson@math.arizona.edu
Asif Mehmood[3] asif.mehmood.1@us.af.mil

Abstract

Adaptive loss function formulation is an active area of research and has gained a great deal of popularity in recent years, following the success of deep learning. However, existing frameworks of adaptive loss functions often suffer from slow convergence and poor choice of weights for the loss components. Traditionally, the elements of a multi-part loss function are weighted equally or their weights are determined through heuristic approaches that yield near-optimal (or sub-optimal) results. To address this problem, we propose a family of methods, called SoftAdapt, that dynamically change function weights for multi-part loss functions based on live performance statistics of the component losses. SoftAdapt is mathematically intuitive, computationally efficient and straightforward to implement. In this paper, we present the mathematical formulation and pseudocode for SoftAdapt, along with results from applying our methods to image reconstruction (Sparse Autoencoders) and synthetic data generation (Introspective Variational Autoencoders).



(a) Fixed Weights (b) SoftAdapt (c) Target

(d) Fixed Weights (e) SoftAdapt (f) Target

Figure 1. Reconstruction of the target image after 250 epochs using IntroVAE by Huang et al. [16]. **(a), (d)**: fixed "optimal" loss function weights (α, β) that Huang et al. found. **(b), (e)**: SoftAdapt adaptive weight balancing. SoftAdapt outperforms the "optimal" weights in different metrics, described in Section 4.2

1. Introduction

Almost all learning through neural networks require (i) a model describing the underlying structure of the training data, (ii) a loss function that gives a metric of how well the network is performing, and (iii) the optimization of the parameters to minimize the objective function. In the past, much of the research had been focused on network architectures [6, 21, 29], but recently, more work is being done on how loss functions affect learning [15, 4, 8]. Networks that perform challenging tasks or multiple tasks often require a combination of losses. Multiple losses are typically combined by taking an equally-weighted linear combination of each objective function; but the importance of each part could be different and thus components should be assigned weights as per their contribution to the learning. On the other hand, the scaling of each component of the loss function can inhibit the ability of the optimizer by only looking at loss components with the largest magnitude. The scaling for gradient descent-based optimizers has been a known issue [18], which our algorithm tries to address throughout the training.

In recent years, the need for weighting the components of multi-part functions has become more evident, and researchers have tried to develop different methods to adjust the weights on the linear combination of loss components. These methods often require defining new loss functions [4] or changing the optimization procedure [8], but there is limited research on the formulation of a general method that can be added to existing architectures. In most cases, the integration with the current models requires sophisticated adjustment or much longer computation time. The advantage of our method is compatibility with any gradient descent-based optimizers in machine learning. Our algo-

[1] Applied Mathematics Department, University of California, Merced
[2] Mathematics Department, University of Arizona
[3] Sensors Directorate, U.S. Air Force Research Laboratory
Preprint. Under review.

rithm can also be used in other optimization applications; for example, in convex optimization, the inverse of the Hessian is a popular preconditioner for gradient descent [25]. However, the Hessian may not be readily available for different applications (*e.g.* in machine learning). Our method may be viewed as using the previous/initial iterations to create a preconditioner matrix P that is a diagonal matrix, such that $P\nabla g$ is approximately isotropic in the parameter space, where ∇g is a partial gradients of the objective function.

In autoencoders (AE), where the goal is to reconstruct the input data using an encoder and a decoder, a regularization term can be added to the default reconstruction loss. This would encourage the model to have different properties (such as sparsity of the representation or robustness to noise). On the other hand, in variational autoencoders (VAE) [20], where the goal is to generate new data that is similar to the input, the two loss functions are Mean Squared Error (MSE) and Kullback-Leibler ($\mathcal{KL}$) divergence (assuming that the prior distribution is a Gaussian). In the case of VAEs, the two losses are crucial for the reconstruction of the input data and estimating the prior distribution to generate new samples; but in AEs the regularization may play a different role in training depending on the problem. An equally weighted linear combination of the losses would mean that each part of the loss function is equally as important in training, which is often not the case [8]. For example, sparse autoencoders [28] employ a very small fixed weight ($\ll 1$) on the regularization term as the sparsity parameter, often denoted by λ, which is usually found by trial and error. Our methods provide learnable parameters that are not fixed, *i.e.* they adapt depending on the performance of the model.

In this paper, we propose a set of Softmax-inspired methods that will adaptively update the weights of the linear combination of individual objective functions, depending on the performance of each part and the collective loss function as a whole. Our family of techniques, called SoftAdapt, can be thought of as "add-ons": one can use their choice of optimizer and only add the weights to the linear combination of the losses, as long as both the losses and the optimizers are suitable for the problem at hand. SoftAdapt evaluates the performance by approximating the rate of change of each loss function over a short history, which indicates if it has been increasing or decreasing. SoftAdapt then compares the individual rates of change and determines how visible each objective function should be to the optimizer.

In summary, our contribution is a family of methods that dynamically learn the best weighting on each part of a multi-component loss function, based on live performance metrics. SoftAdapt is fast, easy to implement and can be added to existing architectures that use any gradient descent-based optimizer.

2. Related Work

Multi-task learning, where the model tries to minimize multiple objective functions to produce an output, is necessary for more challenging tasks but are hard and expensive to train. This learning regime has a wide range of applications, from traffic prediction ([17]) to natural language processing ([9, 30]), and it was introduced even before the exploration of deep learning ([3, 7]). After the deep learning surge, most researchers studied various architectures for multi-objective networks, but more recently, some work has been done towards improving the optimizing of multi-tasking network, based on the optimization functions. Chen et al. [8] contemplate on normalizing the gradients for classification and regression tasks in computer vision.

Miranda and Von Zuben [29] explored the problems and limitations of equally-weighted linear combinations for multi-objective loss functions. Similar to our approach, they interpreted machine learning from a multi-objective optimization perspective. However, they introduced an alternative way of optimization using the gradient of the hypervolume, which is defined as the weighted mean of individual loss gradient. Our family of methods calculate the weights on the linear combination adaptively using the exact gradients computed by any traditional optimizer.

In 2019, Xu et al. [38] studied the importance of component-wise weighting of the loss function; they designed AutoLoss, a framework that learns and determines the scheduling of the optimization. Very similar to our approach, they realized that in multi-task learning it is important to dynamically set a schedule of training, depending on the network architecture. Our techniques use a different metric to find the importance of optimizing each element of the loss function and can be applied to any multi-part loss function. SoftAdapt can also be interpreted as a scheduling algorithm, but it does not assign discrete weights to the component losses. Our algorithm is fast and generalizable to any multi-part loss function since it uses an approximation to the rate of change of each part using a short history, and it is agnostic to the method of training or type of architecture (*e.g.* Autoencoders, GANs, *etc.*)

3. Methods and Approach

In this section, we first discuss the mathematical intuition behind multi-part loss weighting and the basic ingredients required for its formulation. Then we will discuss our algorithm SoftAdapt with its two normalized forms, and we provide pseudocode for the implementation.

3.1. Mathematical Formulation

Consider a loss function of the form

$$F(x) = \sum_{k=1}^{n} f_k(x) \text{for} \tag{1}$$

where we wish to minimize F w.r.t. $x \in \mathbb{R}^m$. Let us suppose that we wish to utilize with some gradient based optimizer $x^{i+1} = Q(x^i, g^i)$, where g^i is the stepping direction for x^i. Typically, we take $g^i = \nabla F(x^i)$. In general, without computing additional information, one could have g^i be dependent on past values of x^i as well as the component losses ($f_k(x^j)$) and the gradients of the component losses ($\nabla f_k(x^j)$), where iteration are denoted by $j = 0, \ldots, i$ and component by $k = 1, \ldots, n$. There are several methods which take advantage of gradient and step information from previous time-steps (*e.g.* Momentum [36], AdaGrad [10], Adam [19], etc), but few, if any, consider recombining the component loss functions; SoftAdapt is designed to address this issue. Let our modified step direction h^i to be given by

$$h^i = \sum_{k=1}^{n} \alpha_k^i \nabla f_k(x^i) \tag{2}$$

and substitute this into Q in place of g^i. We compute the weights α_k^i according to previous loss information. There are three main variations for computing α_k^i.

3.1.1 Original Variant (SoftAdapt)

Here we use the heuristic that it is better to favor the gradient of a function according to its recent performance. Let s_k^i be an approximation of the recent rate of change of the component loss $f_k^i := f_k(x^i)$ (*e.g.* $s_k^i = f_k^i - f_k^{i-1}$, or a more accurate finite difference approximation). Then take

$$\alpha_k^i = \frac{e^{\beta s_k^i}}{\sum_{\ell=1}^{n} e^{\beta s_\ell^i}}, \tag{3}$$

where β is a tunable hyper-parameter. If one chooses $\beta > 0$, SoftAdapt will assign more weight to the worst performing component of the loss function (*i.e.* the component with most positive rate of change). Setting $\beta < 0$ favors the best performing losses (most negative rate of change). Taking $\beta = 0$ gives equal weights. This is simply the classic Softmax evaluation of the vector $(s_1^i, \ldots, s_n^i)$, and is where the method, SoftAdapt, gets its name.

3.1.2 Loss Weighted

Here we modify the Softmax function to account for the current values of the losses, as well as their rates of change. Let

$$\alpha_k^i = \frac{f_k^i e^{\beta s_k^i}}{\sum_{\ell=1}^{n} f_\ell^i e^{\beta s_\ell^i}}. \tag{4}$$

For loss weighting, the component losses must share a minimum (in general, have intersecting minimal sets). The advantage of using this variant is in assigning smaller weights to functions that are close to their minima, even if rates of change stay constant or positive.

3.1.3 Normalized

If one wishes, they may normalize the vector $(s_1^i, \ldots, s_n^i)$ before applying it in Eq. (3) or Eq. (4). This has the effect of sharpening the distinction between small rates of change and softening it between large ones. Normalized and Loss Weighted may be used together if much smaller weights are desirable for loss functions near their minima.

3.2. SoftAdapt

Algorithm 1 Pseudocode for a SoftAdapt and variations: This algorithm is based on loss function L to be comprised of multiple losses. In general, let $L = l_1 + l_2 + \cdots + l_m$

Require: n: number of loss values to be stored
Require: Optimizer: An optimizer for the gradient descent-based method
Require: l_i: the values of the individual m loss functions
Require: variant: A list of variants to be applied to SoftAdapt. A potentially empty subset of {"Normalized", "Loss Weighted"}
Require: $\epsilon = 10^{-8}$ for numerical stability
Ensure: n many epochs/iterations have passed before calling *SoftAdapt*
Ensure: n many l_i have been stored for each l_i

while not converged **do**
  $\beta \leftarrow 0.1$ (default value that can be changed)
  $s_i \leftarrow$ the rate of change (up to $(n-1)$th order accurate) of the past l_i
  $f_i \leftarrow$ the average of up to n previous l_i
  if variant contains "Normalized" **then**
    $ns_i \leftarrow \dfrac{s_i}{(\sum_{i=1}^{m} |s_i|) + \epsilon}$
    $\alpha_i = \dfrac{e^{\beta(ns_i - \max(ns_i))}}{\left(\sum_{j=1}^{m} e^{\beta(ns_j - \max(ns_j))}\right) + \epsilon}$
  else
    $\alpha_i = \dfrac{e^{\beta(s_i - \max(s_i))}}{\left(\sum_{j=1}^{m} e^{\beta(s_j - \max(s_j))}\right) + \epsilon}$
  end if
  if variant contains "Loss Weighted" **then**
    $\alpha_i = \dfrac{f_i \alpha_i}{\left(\sum_{j=1}^{m} f_j \alpha_j\right) + \epsilon}$
  end if
  $TLoss \leftarrow l_1 + l_2 + \cdots l_m$ (true loss for performance measurer)
  $WLoss \leftarrow \alpha_1 l_1 + \alpha_2 l_2 + \cdots \alpha_m l_m$ (weighted loss for the optimizer)
  optimizer ($WLoss$)
end while

4. Experiments and Results

In this section, we conduct various experiments to evaluate the performance of SoftAdapt in different test cases. First, we test Original and Loss Weighted SoftAdapt on the *Rosenbrock* function [34] and Beale's function [18] (in *Supplementary Material* section) using gradient descent. Then, we will test our proposed method on an Introspective Variational Autoencoder (IntroVAE) [16] that uses fixed weights. Lastly, we examine SoftAdapt on a Sparse Autoencoder (SAE) [28] to find the sparsity parameter dynamically during training. For both IntroVAE and SAE, we only change the weighting on the loss components using Loss Weighted SoftAdapt while optimizing with Adam [19].

4.1. Gradient Descent Optimization

As an initial experiment, the SoftAdapt algorithm was tested on a simple gradient descent of standard functions of real vectors. Formally, the minimization problem is: given a smooth function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find an input x which minimizes it, or

$$x = \arg\min_{y \in \mathbb{R}^n} (f(y)) \tag{5}$$

In this section, we present our results on applying the SoftAdapt modified gradient to the classical gradient descent algorithm, both with fixed step size and adaptive step size. The first function to consider is the 2D Rosenbrock function [34]:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{6}$$

which exhibits a narrow valley that leads to a single global minimum at $(x, y) = (1, 1)$. Typically, gradient descent on the Rosenbrock function will either diverge at step sizes on the order of 10^{-2} and larger, or will take a long time to converge. For the experiment, we split $f(x, y) = f_1(x, y) + f_2(x, y)$ where

$$f_1(x, y) = (1 - x)^2 \quad \text{and} \quad f_2(x, y) = 100(y - x^2)^2 \tag{7}$$

For our update procedure we consider two cases. First, we have normal gradient descent:

$$x^{i+1} = x^i - \eta h^i \tag{8}$$

where η is the fixed learning rate, and h^i is one of the SoftAdapt variant gradients. Second, we will use an adaptive learning rate:

$$x^{i+1} = x^i - \eta^i h^i \tag{9}$$

where η^i is updated according to the Barzilai-Borwein scheme [5], subject to a minimum and maximum learning rate. Fig. 2 shows the trajectories for both, traditional gradient descent and gradient descent with SoftAdapt, and the number of steps taken to reach the minimum. Note that it is appropriate to use loss weighting here, as the minimal sets of f_1 and f_2 intersect at the true minimum. Our method performs well in three of the regimes and significantly outperforms gradient in fixed step, loss weighting (Fig. 2 **(b)**). We underperform in the case where both adaptive learning rates and loss weighting are used, but using values of $\beta < 0$ can improve performance. We also witnessed similar improvements in the gradient descent optimization for Beale's function, which is illustrated in *Supplementary Material*.

4.2. Introspective Variational Autoencoders

Introspective Variational Autoencoder (IntroVAE) was first introduced by Huang et al. [16] in 2018. IntroVAE is a single-stream generative model that self-evaluates the quality of the generated images, as opposed to Generative Adversarial Networks[12] (GAN), which have a separate network for generating samples and a separate network for discriminating between real and synthetic images. Their interesting approach is that "[IntroVAE] inference and generator models are jointly trained in an introspective way. On one hand, the generator is required to reconstruct the input images from the noisy outputs of the inference model as normal VAEs. On the other hand, the inference model is encouraged to classify between the generated and real samples while the generator tries to fool it as GANs." [16]. In the model, the authors use the following loss functions for the encoder (denoted by L_E) and for the generator (denoted by L_G):

$$L_E = L_{REG}(z) + \alpha \sum_{s=r,p} [m - L_{REG}(z_s)]^+ + \beta L_{AE}(x, x_r) \tag{10}$$

$$L_G = \alpha \sum_{s=r,p} L_{REG}(Enc(x_s)) + \beta L_{AE}(x, x_r) \tag{11}$$

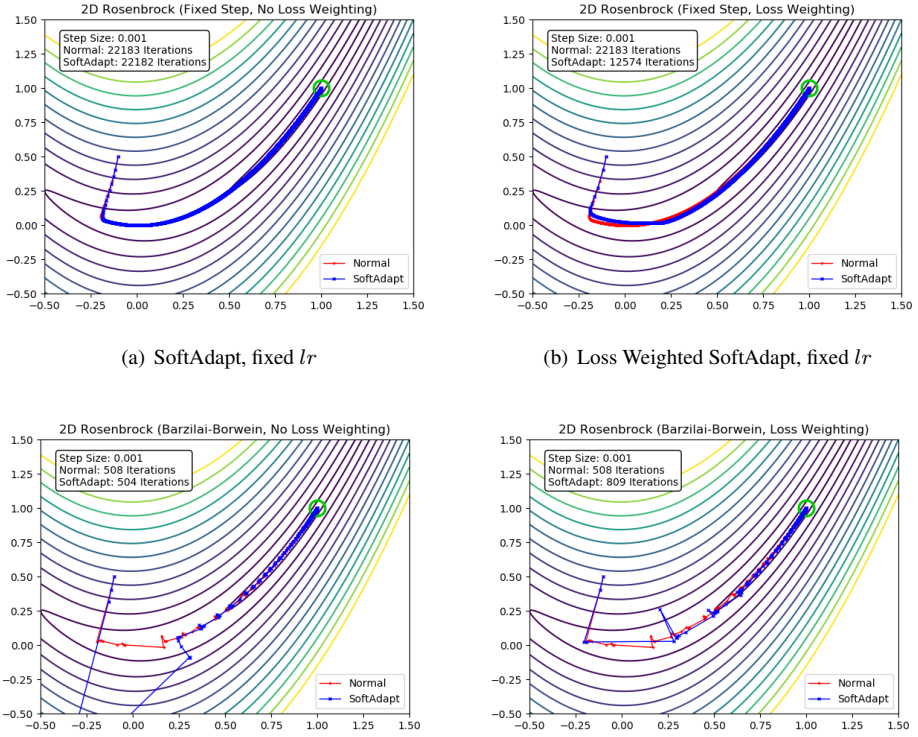
where L_{REG} is the $\mathcal{KL}$-divergence, which can be computed for N data samples (with dimension of z as M_z) as :

$$L_{REG}(z; \mu, \sigma) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M_z} (1 + \log(\sigma_{i,j}^2) - \mu_{i,j} - \sigma_{i,j}^2) \tag{12}$$

L_{AE} is the mean squared error: given x_r (the reconstructed image of x) and the dimension of x as M_x, we have :

$$L_{AE}(x, x_r) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M_x} \| x_{r,ij} - x_{ij} \|_F^2 \tag{13}$$

In Eq. (10), m is a number which is selected to keep L_{REG} below a threshold and $Enc(\cdot)$ represents function that the



(a) SoftAdapt, fixed lr

(b) Loss Weighted SoftAdapt, fixed lr

(c) SoftAdapt, adaptive lr

(d) Loss Weighted SoftAdapt, adaptive lr

Figure 2. Performance of SoftAdapt vs. gradient descent for the Rosenbrock function. The learning rate (lr) is changing according to the Barzilai-Borwein scheme [5] in **(b), (c)**. We see the most improvement (43.31% faster) for loss weighted SoftAdapt with fixed learning rate. Upon changing the value of β, significant improvements can be made, but the default values of parameters in our implementation are $\beta = 0.1$, $\eta = 10^{-3}$. The max and min lr are $\eta_{min} = 10^{-4}$, $\eta_{max} = 10^{-1}$

encoder is mapping. For this paper, our focus is on the α and β, which the authors note as the "weighting parameters used to balance the importance of each item." [16]

Our results show that the optimal set of α and β does not need to be known in advance since the importance of each part of the loss function can be determined adaptively throughout training using our method. Huang et al. find the "optimal" value of α and β empirically and by pre-training the networks for each different dataset; this results in a different set of α and β for different data. The authors make note of this issue and provide the readers with a set of values for each subset of the CELEBA dataset [27]. One can avoid finding these weights explicitly for various training data by using SoftAdapt instead since the weight would be learned adaptively during training. Using SoftAdapt, the weighted loss functions in Eq. (10), (13) will be

$$L_E^{(n+1)} = L_{REG}(z) + \alpha_1^{(n)} \sum_{s=r,p} [m - L_{REG}(z_s)]^+ + \alpha_2^{(n)} L_{AE}(x, x_r) \tag{14}$$

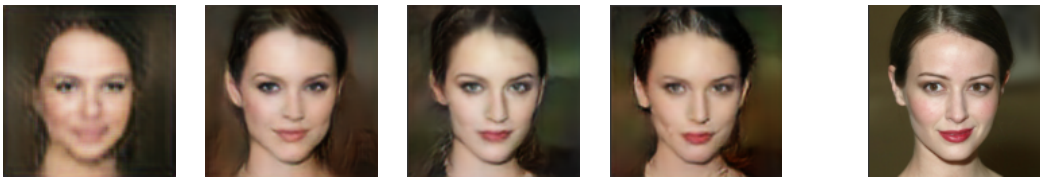
$$L_G^{(n+1)} = \alpha_1^{(n)} \sum_{s=r,p} L_{REG}(Enc(x_s)) + \alpha_2^{(n)} L_{AE}(x, x_r) \tag{15}$$

using

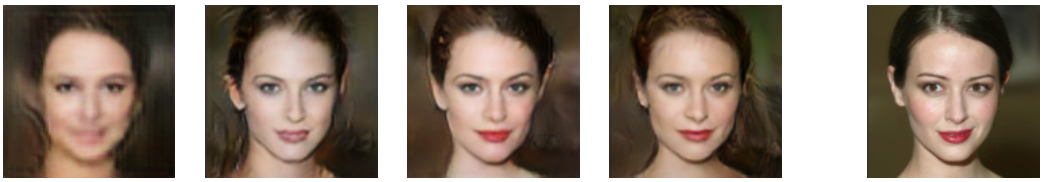
$$\alpha_i^{(n)} = SoftAdapt\left(L_{REG}^{(n)}, L_{AE}^{(n)}\right) \tag{16}$$

where $i = \{1, 2\}$ and $n \in \mathbb{N}$ denoting the time step for α_i. We initialize $\alpha_i^{(0)} = 0.5$ since we want to treat it without bias in the very beginning.

Tables 1 and 2 demonstrate the quantitative comparisons: Peak signal-to-noise ration (PSNR), Structural Similarity Index (SSIM) and Naturalness Image Quality Evaluator (NIQE) for Fig. 3, 4. These figures illustrate the IntroVAE reconstruction of random subset of 128×128 CELEBA dataset using authors' fixed weights versus using our method to find those weights dynamically (Fig. 3). The training time between the two methods were also very comparable, 1411.489043 minutes for fixed weights vs. 1413.112740 minutes with SoftAdapt.



(a) Epoch 12 (b) Epoch 100 (c) Epoch 200 (d) Epoch 250 (e) Target

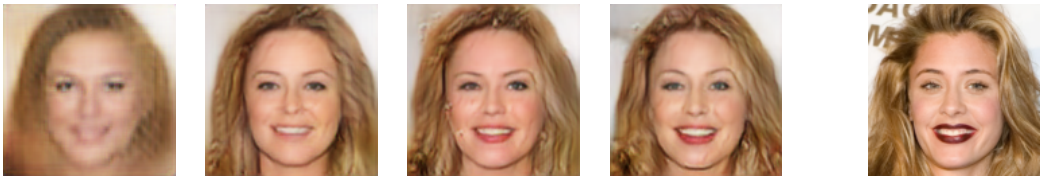


(f) Epoch 12 (g) Epoch 100 (h) Epoch 200 (i) Epoch 250 (j) Target

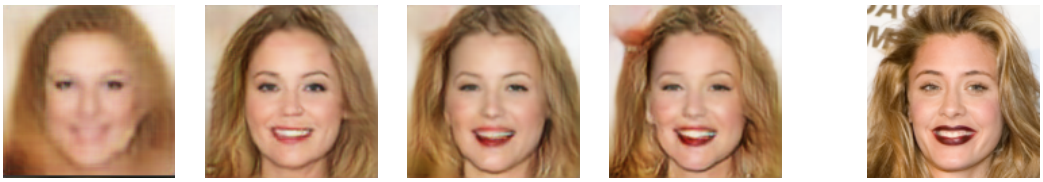
Figure 3. Reconstruction of the target image [**(e)**, **(j)**] using IntroVAE with fixed loss weighting from Huang et al. [16] [images **(a-d)**] vs our adaptive loss weighting with SoftAdapt [images **(f-i)**]

Table 1. Comparison between SoftAdapt and fixed loss weights for Fig. 3 (boldface indicates better performance).

	Ours (SoftAdapt)				**Huang et. al.**			
	Epoch 12	*Epoch 100*	*Epoch 200*	*Epoch 250*	*Epoch 12*	*Epoch 100*	*Epoch 200*	*Epoch 250*
SSIM	**0.7752**	**0.8331**	**0.8100**	**0.8473**	0.7551	0.8018	0.7847	0.7838
PSNR	**21.5620**	**23.3376**	**23.8525**	**23.9272**	21.4471	23.0899	23.2070	22.2415
NIQE	18.8726	**18.8715**	18.8720	**18.8705**	**18.8725**	18.8731	**18.8711**	18.8714



(a) Epoch 12 (b) Epoch 100 (c) Epoch 200 (d) Epoch 250 (e) Target



(f) Epoch 12 (g) Epoch 100 (h) Epoch 200 (i) Epoch 250 (j) Target

Figure 4. Reconstruction of the target image [**(e)**, **(j)**] using IntroVAE with fixed loss weighting from Huang et al. [16] [images **(a-d)**] vs our adaptive loss weighting with SoftAdapt **(f-i)**]

Table 2. Comparison between SoftAdapt and fixed loss weights for Fig. 4 (boldface indicates better performance).

	Ours (SoftAdapt)				**Huang et. al.**			
	Epoch 12	*Epoch 100*	*Epoch 200*	*Epoch 250*	*Epoch 12*	*Epoch 100*	*Epoch 200*	*Epoch 250*
SSIM	0.7940	**0.8260**	**0.8306**	**0.8303**	**0.8042**	0.8167	0.8214	0.8083
PSNR	18.0110	**19.4724**	**20.0634**	**19.8027**	**18.7680**	19.0574	19.1012	19.3225
NIQE	**18.8700**	18.8740	**18.8744**	**18.8750**	**18.8730**	**18.8731**	18.8763	18.8756

4.3. Sparse Autoencoders

Autoencoders (AEs) are models that aim to reconstruct the input as the output. These networks are comprised of two parts: 1) *Encoder*, a neural network where the data is mapped to a latent space, typically of a smaller dimension than the input. 2) *Decoder*, a neural network where the latent space is mapped back to the original dimension of the network, and, in an optimal case, an exact reconstruction of the input to the encoder. A general autoencoder has the loss of form

$$L(x, \hat{x}) = L(x, g\left(f(x)\right))$$

where x represents the data, $\hat{x}$ denotes the data reconstruction and $f(x), g(x)$ are the mappings of the encoder and the decoder respectively. To test the performance of SoftAdapt, we trained an AE to reconstruct the MNIST digits [22, 23] with a loss function :

$$L(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \lambda \sum_{i=1}^{m}|a_i^{(h)}|,$$

where the added L_1 regularization tries to penalize the absolute value of activation layer for a sample i in layer h; this is known as a sparse autoencoder [28] since the L_1 regularization on the activation of the hidden layers enforces activation of only a few neurons when a sample is inputted. Normally, the hyper-parameter λ is tuned to control the effect of the regularization by trial and error. We used SoftAdapt to dynamically adjust the effects of the penalty depending on the performance of each component (MSE and L_1 regularization) and the network as a whole. The new loss function using SoftAdapt becomes:

$$L(x, \hat{x})^{(k+1)} = \alpha_1^{(k)} \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \alpha_2^{(k)} \sum_{i=1}^{m}|a_i^{(h)}| \quad (17)$$

where k denotes the current iteration and

$$\alpha_i^{(k)} = SoftAdapt\left(\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2\right]^{(k)}, \left[\sum_{i=1}^{m}|a_i^{(h)}|\right]^{(k)}\right) \quad (18)$$

Fig. 5 shows that our method keeps the loss for both training and validation data lower than the traditional "optimal" λ, and Table 3 demonstrates that our reconstructions have a higher classification throughout training than the fixed optimal λ. We also show that our method is qualitatively comparable to training the network using the optimal $\lambda = 10^{-4}$ from the beginning (Fig. 6). It is worthy to note that the optimal λ is found through trial and error, in our case using a grid search which is expensive, but with SoftAdapt no prior knowledge of the values of λ is required. Details about network architecture and other hyper-parameters are presented in the *Supplemental Material* section.

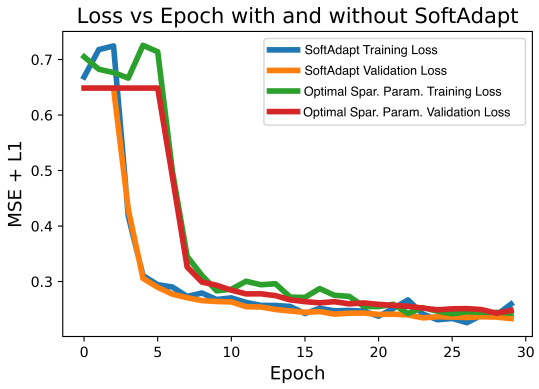
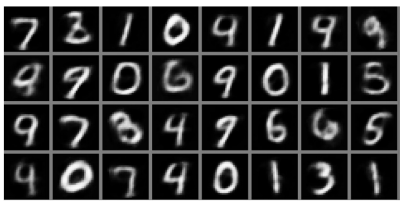


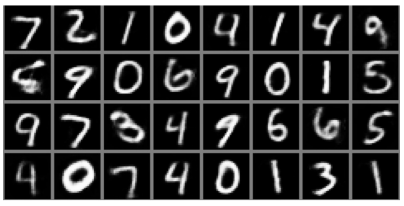
Figure 5. Loss vs. epoch for a sparse autoencoder trained with $\lambda = 10^{-4}$ ("optimal") against using SoftAdapt for weight balancing. Our method performs better throughout training, although the traditional method is comparable to ours for a larger number of epochs.

5. Conclusion

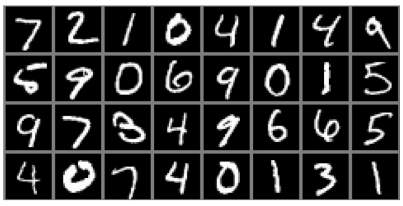
We have presented a set of optimization add-ons for weighting the importance of different components adaptively in multi-part objective functions. By adjusting the weights dynamically, the training can become much easier and faster since no prior knowledge of the network is required, *i.e.* no pre-training or grid search is needed. We outlined multiple variants of our Softmax-inspired algorithm and described the suitable application for each one. The first variant of SoftAdapt is a Softmax function where the rate of change is the input, which serves as a performance measure of each part. This is useful when the components of the loss function have the same order of magnitude (*e.g.* various euclidean norms). The second variant uses the magnitude of each part of the loss function as well as the rate of change, which gives the most improvement when the values of the objective functions are on different scales. This variant also has the advantage of assigning smaller weights to the loss functions that are close to their minima, even with large rates of change, and putting more importance on the rest of the objective functions. Our last variant uses normalized rates of change to ensure a better distribution of weights when the slopes possess vastly different scales. It is important to note that the second and third variants may be used together if needed. Our SoftAdapt algorithm is implemented in one easy-to-use package available online on the authors' websites (not included due to the blind review). Our results show that our algorithm works well in practice for a wide spectrum of problems in machine learning, such as image reconstruction and synthetic data generation, as well as general gradient decent optimizations where scaling is an issue.



(a) Trained with Fixed $\lambda = 10^{-4}$



(b) Trained with SoftAdapt



(c) Target

Figure 6. Reconstruction of a random set of MNIST [23] digits from the testing data with a sparse autoencoder using our algorithm *SoftAdapt*. With SoftAdapt, there is no need to find the sparsity parameter λ explicitly and by hand. The performance of the autoencoder using our algorithm is comparable to training the network with a fixed optimal value of λ found by trial and error.

Table 3. Classification and time comparison between adaptive weights (ours) and "optimal" λ (found manually) in Sparse Autoencoder with loss $L = MSE(\cdot) + L_1 Regularization$

	Ours (SoftAdapt)				**Fixed** $\lambda = 10^{-4}$			
	Epoch 2	*Epoch 5*	*Epoch 15*	*Epoch 30*	*Epoch 2*	*Epoch 5*	*Epoch 15*	*Epoch 30*
PCC	11%	**75** %	**87** %	**88** %	11%	52%	69%	82%
Time	8.986135 Minutes				**7.939554 Minutes**			

Acknowledgments

We would like to acknowledge Omar DeGuchy, Radoslav Vuchkov and Alina Gataullina for their constructive comments regarding our methods and writings. We would like to thank Fred Garber, Olga Mendoza-Shrock, Jamison Moody, Oliver Nina, Alexis Ronnebaum and Suzanne Sindi for their feedback and support. We also appreciate the computation resources provided by the University of California, Pacific Research Platform and the Wright State University to the authors in conducting this research.

References

[1] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

[2] Devansh Arpit, Yingbo Zhou, Hung Q. Ngo, and Venu Govindaraju. Why regularized auto-encoders learn sparse representation? *ArXiv*, abs/1505.05561, 2015.

[3] Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *J. Mach. Learn. Res.*, 4:83–99, Dec. 2003. 2

[4] Jonathan T. Barron. A general and adaptive robust loss function. *CVPR*, 2019. 1

[5] Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988. 4, 5

[6] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009. 1

[7] Rich Caruana. *Multitask Learning*, pages 95–133. Springer US, Boston, MA, 1998. 2

[8] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *CoRR*, abs/1711.02257, 2017. 1, 2

[9] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM. 2

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic

optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. 3

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press. 4

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[14] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 44–51, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[15] Chen Huang, Shuangfei Zhai, Walter Talbott, Miguel Ángel Bautista, Shih-Yu Sun, Carlos Guestrin, and Josh Susskind. Addressing the loss-metric mismatch with adaptive loss alignment. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2891–2900, 2019. 1

[16] Huaibo Huang, Zhihang Li, Ran He, Zhenan Sun, and Tieniu Tan. Introvae: Introspective variational autoencoders for photographic image synthesis. *CoRR*, abs/1807.06358, 2018. 1, 4, 5, 6

[17] Wenhao Huang, Haikun Hong, Man Li, Weisong Hu, Guojie Song, and Kunqing Xie. Deep architecture for traffic flow prediction. In Hiroshi Motoda, Zhaohui Wu, Longbing Cao, Osmar Zaiane, Min Yao, and Wei Wang, editors, *Advanced Data Mining and Applications*, pages 165–176, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 2

[18] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *IJMNO*, 4:150–194, 2013. 1, 4

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 3, 4

[20] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. 2

[21] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. 1

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. 7

[23] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. 7, 8

[24] Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. *ArXiv*, abs/1805.08114, 2018.

[25] Xi-Lin Li. Preconditioned stochastic gradient descent. *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):14541466, May 2018. 2

[26] Jinxiu Liang, Yong Xu, Chenglong Bao, Yuhui Quan, and Hui Ji. Barzilaiborwein-based adaptive learning rate for deep learning. *Pattern Recognition Letters*, 128:197 – 203, 2019.

[27] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 5

[28] Alireza Makhzani and Brendan J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013. 2, 4, 7

[29] Conrado Miranda and Fernando Von Zuben. Multiobjective optimization for self-adjusting weighted gradient in machine learning tasks. *arXiv preprint arXiv:1506.01113*, 06 2015. 1, 2

[30] Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *CoRR*, abs/1604.05529, 2016. 2

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2015.

[32] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[33] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2016.

[34] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960. 4

[35] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[36] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 3

[37] Anush Sankaran, Mayank Vatsa, Richa Singh, and Angshul Majumdar. Group sparse autoencoder. *Image Vision Comput.*, 60(C):64–74, Apr. 2017.

[38] Haowen Xu, Hao Zhang, Zhiting Hu, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Autoloss: Learning discrete schedules for alternate optimization. In *International Conference on Learning Representations*, 2019. 2