Training Dynamics Aware Neural Network Optimization with Stabilization

Zilin Fang[1,3], Mohamad Shahbazi[1], Thomas Probst[1], Danda Pani Paudel[1], Luc Van Gool[1,2]

zilin.fang@u.nus.edu, {mshahbazi, probstt, paudel, vangool}@vision.ee.ethz.ch

[1]Computer Vision Laboratory, ETH Zurich, Switzerland
[2]VISICS, ESAT/PSI, KU Leuven, Belgium
[3]School of Computing, NUS, Singapore

Abstract. We investigate the process of neural network training using gradient descent-based optimizers from a dynamic system point of view. To this end, we model the iterative parameter updates as a time-discrete switched linear system and analyze its stability behavior over the course of training. Accordingly, we develop a regularization scheme to encourage stable training dynamics by penalizing divergent parameter updates. Our experiments show promising stabilization and convergence effects on regression tasks, density-based crowd counting, and generative adversarial networks (GAN). Our results indicate that stable network training minimizes the variance of performance across different parameter initializations, and increases robustness to the choice of learning rate. Particularly in the GAN setup, the stability regularization enables faster convergence and lower FID with more consistency across runs. Our source code is available at: https://github.com/fangzl123/stableTrain.git

1 Introduction

Ever since Augustin-Louis Cauchy first described the method of gradient descent in 1847 [1], its efficiency and flexibility has inspired countless solvers and optimization algorithms until this day [2–9]. Gradient descent is, in fact, the backbone of the recent advancements in deep learning, in conjunction with the error back-propagation technique [10]. In particular, auto-differentiation offers gradient computation with negligible overhead on the function evaluation, making possible the optimization of large-scale non-linear functions with millions of parameters – such as deep neural networks – using gradient descent [11, 12].

Gradient descent (GD) and its derivatives have been extensively studied with regards to their convergence properties on various problems [13, 14]. For instance, the choice of the learning rate is crucial for (fast) convergence, and depends on the curvature around a local optimum. Choosing a high learning rate may cause oscillating and divergent behavior, whereas a low learning rate may cause the optimizer to never reach a good solution. Moreover, when using stochastic gradients in the case of typical neural network training, gradient noise can cause undesired updates. Other sources of noise can be multi-task training [15–17], data augmentation [18–20], or re-sampling operations [21–23].

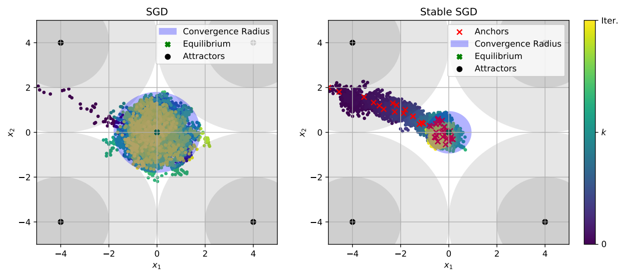


Fig. 1: **Trajectories with and without stabilization.** The loss function randomly switches between 4 quadratic attractors, and the convergence to the equilibrium at origin is desired. Switching causes SGD to oscillate, which is dampened by stabilization.

To this end, numerous GD-based optimization algorithms have been proposed to deal with the aforementioned issues. It has shown to be beneficial to modify the gradients before the update step to enforce a desired behavior [24, 25]. For instance, gradient clipping [24] offers an intuitive solution to the problem of diverging gradients. Introducing momentum can reduce noise and accelerate convergence in the presence of high curvature regions and is used by many modern optimizers [26, 8]. Furthermore, adaptive learning rates for each parameter (e.g. RMSprop, Adam) often offer robust learning behavior. With AdaDelta [27] there is even a hyperparameter-free optimizer with adaptive learning rates.

In general, the behavior of GD optimization can be understood from the viewpoint of a corresponding dynamic system [14]. The system intuitively models the evolution of the parameters (as the state variable) under the update given by the optimizer. For instance, in the case of a quadratic cost function, the dynamic system is given by a linear system (as discussed in Section 2.1). Based on the spectrum of the update matrix with respect to the learning rate, convergence criteria can be developed [13, 14]. In particular, we are interested in the stability of the weight updates, while stability analysis has been previously applied to the input-output dynamics of neural networks [28, 29].

In the general case of neural networks, however, mini-batch training, nonlinearities, and non-convex loss functions complicate such analysis. In this paper, we attempt a stability analysis of network training with the tools of switched linear systems (SLS) [30–34]. The key assumption is that evolution of parameters in a specific layer follows a time-variant linear dynamic over the course of iterations. This is motivated by two observations. First, each mini-batch contains different random samples. Second, the input activation to the layer change after each update iteration. In both cases, the update dynamics change accordingly, which we model by switching between corresponding linear systems.

The paper is structured as follows. We begin by presenting the theoretical concept of SLS in the context of gradient descent training in Section 2. Based on this model, we analyse the stability of the optimization procedure in Section 3 and develop a mechanism to control the parameter updates, such that the update dynamics remain stable. For our experiments, we approximate this mechanism by introducing a stability regularizer to be jointly optimized with the loss function. In Section 4, we empirically demonstrate the effect of stabilization in terms of variability and performance on various computer vision problems, including generative adversarial networks [35, 36] and crowd counting [37, 38].

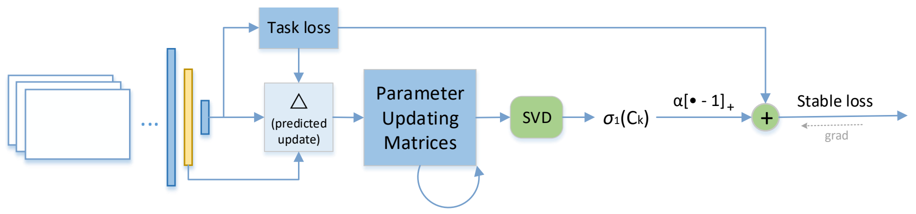


Fig. 2: **Stability regularization pipeline.** For training, the features of the last linear/convolutional layer or the weights and bias themselves are taken out to construct a time-variant state matrix which represents the parameters updating process at the current step. After left multiplying with the accumulated history matrix, singular value decomposition will be applied to obtain the matrix norm. By scaling with a hyper-parameter α and adding to the original loss, the training loss now is a combination of the task-specific loss and the regularization loss.

2 Neural Network Training Dynamics

We introduce the theoretical part by analyzing the update process of mini-batch gradient descent in 2.1, and by formulating a dynamic system to model the corresponding dynamics. Then we discuss how several aspects of GD can be modeled using linear systems in 2.2. Finally, we present a model for generic updates, and formulate the joint dynamic system evolving over training iterations. The joint dynamics will be analysed with regards to stability in Section 3.

2.1 Mini-Batch Training as a Dynamic System

Let us consider one update iteration for a parameter vector $\theta \subseteq \Theta$ of one layer during training of a network f_Θ using stochastic gradient descent with mini-batches. At iteration k, the update for batch $(\mathbf{x}_k, \mathbf{y}_k)$ is

$$\theta_{k+1} = \theta_k - \eta \frac{\partial}{\partial \theta} \underbrace{\mathcal{L}(f_\Theta, \mathbf{x}_k, \mathbf{y}_k)}_{:=\mathcal{L}^k}, \tag{1}$$

with learning rate η and loss functional $\mathcal{L}$.

We start with the example of a linear last layer with ℓ_2 regression loss,

$$\mathcal{L}(\mathsf{a}(x), y) = (\theta^T \mathsf{a}(x) + b - y)^2, \tag{2}$$

where a is the input activation, and y the label. Since the gradient $\frac{\partial}{\partial \theta}\mathcal{L}$ is a linear function of the parameters θ, the update (1) can be formulated as an affine system $(\mathbf{A}_k, \mathsf{b}_k)$,

$$\theta_{k+1} = \mathbf{A}_k \theta_k + \mathsf{b}_k. \tag{3}$$

For each iteration k, the parameter update is given by a corresponding system $(\mathbf{A}_k, \mathsf{b}_k)$. The training process can therefore be interpreted as an affine time-discrete switched linear system [34]. If A_k is stable, the system (3) tends to a different equilibrium θ_k^e for each mini-batch k,

$$\theta_k^e = (\mathbf{I} - \mathbf{A}_k)^{-1}\mathsf{b}_k. \tag{4}$$

The set of attainable equilibrium points during network training is denoted as,

$$\vartheta^e = \{\theta_k^e \in \mathbb{R}^m : \theta_k^e = (\mathbf{I} - \mathbf{A}_\mu)^{-1}\mathsf{b}_\mu\}, \tag{5}$$

formed by convex combinations of stable linear systems

$$\begin{gathered} \mathbf{A}_\mu = \sum_{i=0\ldots k} \mu_i \mathbf{A}_i, \quad \mathsf{b}_\mu = \sum_{i=0\ldots k} \mu_i \mathsf{b}_i, \\ \mu \in \{\mu \in \mathbb{R}^k : \sum_i \mu_i = 1, \mu_i \geq 0\, \forall i\}. \end{gathered} \tag{6}$$

Any solution θ^* of the neural network training is therefore found in $\theta^* \in \vartheta^e$.

However, not only can any number of systems in $\Sigma = \{(\mathbf{A}_i, \mathsf{b}_i)\}$ be unstable, but even switching between stable systems can create instability. This leads to oscillating behavior, especially with high learning rates η. On the other hand, clever switching of systems can stabilize the overall dynamic. In this paper, we are interested in studying the stability of deep learning optimization problems and finding ways for stabilising the network training.

2.2 Dynamics of Gradient Descent Optimizers

We can model the update dynamics of GD-based optimizers with learning rate η using a first order system as follows:

$$\begin{bmatrix} \theta_{k+1} \\ \Delta\theta_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{D}_k & -\eta \mathbf{H}_k \\ \mathbf{L}_k & \mathbf{M}_k \end{bmatrix} \begin{bmatrix} \theta_k \\ \Delta\theta_k \end{bmatrix} + \mathbf{B}_k \mathsf{u}_k, \tag{7}$$

with time-variant subsystems $\mathbf{D}_k$, $\mathbf{H}_k$, $\mathbf{L}_k$, and $\mathbf{M}_k$. $\mathbf{D}_k$ performs parameter updates proportional to the parameters θ, and can be used to represent ℓ_2 weight decay for $\mathbf{D}_k = \mathbf{I} - \eta\tau$, with regularization strength τ. The additive update is performed with subsystem $\mathbf{H}_k$, and is typically chosen as $\mathbf{H}_k = \mathbf{I}$, in case the learning rate is the same for all parameters. System $\mathbf{L}_k$ handles linear gradients, such as in the case of (3). The choice of $\mathbf{M}_k$ allows for introduction of gradient momentum, e.g. with $\mathbf{M}_k = \beta\mathbf{I}$. The affine bias terms u_k act on the update through the control matrix $\mathbf{B}_k$.

For many optimizers and loss functions, the linear assumption as in (3) does not apply. In this case, we omit the linear dynamics $\mathbf{L}_k = \mathbf{0}$ and introduce the additive update through the affine term as,

$$\begin{bmatrix} \theta_{k+1} \\ \Delta\theta_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{D} & -\eta\mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \theta_k \\ \Delta\theta_k \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{O}(\frac{\partial}{\partial\theta}\mathcal{L}^k), \tag{8}$$

where $\mathbf{O}(\frac{\partial}{\partial\theta}\mathcal{L}^k)$ is the update from the optimizer (e.g. Adam [8]). Since $\mathbf{O}$ already includes momentum terms, we set $\mathbf{M} = \mathbf{0}$. To simplify further analysis, we rewrite (8) into a homogeneous form with state $\tilde{\theta}_k$ and state transition matrix $\mathbf{A}_k$ as,

$$\tilde{\theta}_{k+1} = \begin{bmatrix} \theta_{k+1} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{D} & -\eta\mathbf{O}(\frac{\partial}{\partial\theta}\mathcal{L}^k) \\ 0 & I \end{bmatrix} \begin{bmatrix} \theta_k \\ 1 \end{bmatrix} = \mathbf{A}_k \tilde{\theta}_k. \tag{9}$$

The state of this affine switched linear system is therefore directly adapted by the optimizer algorithm. We can write the finite matrix left-multiplication chain up to iteration k as,

$$\begin{aligned} \tilde{\theta}_{k+1} &= \mathbf{A}_k \dots \mathbf{A}_0 x_0 = \prod_{i=0}^{k} \mathbf{A}_i x_0 \\ &= \underbrace{\begin{bmatrix} \mathbf{D}^k & -\eta \sum_{i=1}^{k} \mathbf{D}^{k-i}\mathbf{O}(\frac{\partial}{\partial\theta}\mathcal{L}^i) \\ 0 & I \end{bmatrix}}_{\mathbf{C}_k} \tilde{\theta}_0. \end{aligned} \tag{10}$$

$\mathbf{C}_k$ represents the joint dynamic evolving over iterations.

3 Stable Network Optimization

Motivated by stabilization through switching, in the following, we develop a stability criterion for the system at the next time step in order to stabilize the update dynamic. With the tools of Liapunov functions, we derive a constraint on the joint dynamic $\mathbf{C}_k$, to avoid unstable updates in 3.1. In practice, we relax this constraint in to ways: by introducing temporary anchors in 3.3, and by approximation through regularization in 3.4, to obtain an efficient algorithm.

3.1 Liapunov Stability

To analyse the stability of the system (10), we define a Liapunov energy function,

$$V(\xi) = \xi^T \mathbf{P} \xi - 1, \tag{11}$$

with $\xi = \left[(\theta - \theta_0)^T \ 1\right]^T$ around the intial parameter θ_0. For time-discrete systems, we consider the energy difference between two steps,

$$\Delta V = V(\xi_{k+1}) - V(\xi_k). \tag{12}$$

According to Liapunov's theorem, for $V(0) = 0$, and $\Delta V(x) \leq 0$, the system is stable around the origin [39]. Note that we do no desire asymptomatic stability, since we do not know the solution of the learning problem beforehand. Thus, we merely aim for stable behavior in the vicinity of the inital parameter θ_0. For our dynamic (10) we obtain,

$$\begin{aligned} \Delta V &= V(\mathbf{C}_k \xi) - V(\xi) \\ &= \xi^T (\mathbf{C}_k^T \mathbf{P} \mathbf{C}_k - \mathbf{P}) \xi. \end{aligned} \tag{13}$$

Therefore, we achieve $\Delta V \leq 0$ for,

$$\mathbf{C}_k^T \mathbf{P} \mathbf{C}_k - \mathbf{P} = -\mathbf{Q}. \tag{14}$$

with positive definite matrices $\mathbf{P}, \mathbf{Q} \succeq 0$. Choosing $\mathbf{P} = \mathbf{I}_{3x3}$, we obtain $\mathbf{Q} \succeq 0$, if

$$\begin{aligned} \mathbf{I} - \mathbf{C}_k^T \mathbf{C}_k &\succeq \mathbf{0} \\ \Rightarrow \max_i \lambda_i(\mathbf{C}_k^T \mathbf{C}_k) &\leq \min_i \lambda_i(\mathbf{I}) \\ \Rightarrow \sigma_1(\mathbf{C}_k) &\leq 1, \end{aligned} \tag{15}$$

where $\sigma_1(\mathbf{C}_k)$ is the largest singular value of $\mathbf{C}_k$.

3.2 Stable Network Training

To facilitate stable training, the goal is to find network parameters Θ, that minimize the loss $\mathcal{L}$ under the stability constraints given by (15). At iteration k, we therefore wish to control the network parameters Θ_k by solving,

$$\begin{aligned} \Theta_k = \arg\min_{\Theta} \quad & \mathcal{L}(f_\Theta, \mathbf{x}_k, \mathbf{y}_k) \\ s.t. \quad & \mathbf{C}_k(\Theta) = \mathbf{A}_k(\Theta) \mathbf{C}_{k-1}, \\ & \sigma_1(\mathbf{C}_k) \leq 1. \end{aligned} \tag{16}$$

This can be seen as a model predictive controller with a single step time horizon. Here we predict how θ_k would be updated according to $\mathcal{L}$, as represented by $\mathbf{A}_k(\Theta_k)$ (9). The actual update of $\theta_k \subseteq \Theta_k$ however is given by the solution Θ_k of (16), taking the constraints (15) into consideration.

In practice, we approximate the constrained problem of (16) by introducing a regularizer to the original loss as,

$$\mathcal{L}_{stable} = \mathcal{L}^k + \alpha \left[\sigma_1(\mathbf{C}_k) - 1 \right]_+ \tag{17}$$

where $\mathcal{L}^k$ is the task-specific mini-batch loss, $[\bullet]_+$ is a rectifier, and $\sigma_1(\mathbf{C}_k)$ is the stability-based regularizer with strength α.

Algorithm 1: Stability-regularized training process

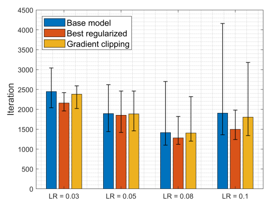
```
#theta: layer weights
#eta: learning rate
#alpha: regularizer strength
def train_epoch(batch_data, theta, eta, alpha):
  # reset history (optional)
  C = eye(len(theta)+1)

  for (x,y) in batch_data:

    #predict update $\mathbf{O}(\frac{\partial}{\partial\theta}\mathcal{L})$, e.g. with GD:
    update = -eta * compute_grad(theta, x, y)

    #construct joint system Eq.(9) and Eq.(10)
    A_tilde = affine_system(update)
    C_tilde = A.matmul(C_tilde)

    #compute total loss Eq.(17)
    loss = $\mathcal{L}$(x,y)+alpha*relu(svd(C_tilde)[0]-1)

    #backprop and parameter update
    loss.backwards()
    update_final = optimizer.step()

    #update history
    A = affine_system(update_final)
    C = A.matmul(C_tilde)
```

3.3 Anchoring

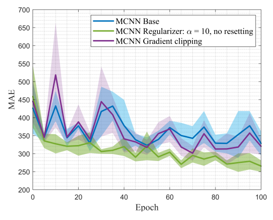
When encouraging Liapunov stability according to (17), we effectively search only in the vicinity of the initial solution. This is too restrictive in some cases. We therefore introduce a series of anchors, where the history gets re-initialized as $\mathbf{C}_k \leftarrow \mathbf{I}$. In our experiments, we investigate epoch-wise resetting, trading of stability and exploration. Figure 1 shows the behavior of stabilization with anchors, in a 2D toy example. In every iteration, one of 4 noisy attractors is randomly chosen to compute the gradient (mimicking mini-batch noise), and we desire to reach the equilibrium point at the origin. Stabilization (15) can mitigate the oscillations, while moving the anchor enables asymptotic convergence.

3.4 Training Algorithm

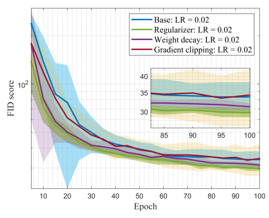
To optimize (17) using SGD, we first perform a forward pass to construct the current $\mathbf{A}_k(\Theta)$ and update the history matrix $\mathbf{C}_{k-1}$. Note that $\mathbf{C}_{k-1}$ contains updates from preceding mini-batches and is treated as a constant, while the updated $\mathbf{C}_k$ involves taking the derivative of $\mathcal{L}$ w.r.t. the parameters Θ. Since the gradients of (17) are a function of the update $\mathbf{O}$, we express the update analytically as a function of θ, if possible. For complex optimizers and loss functions, we assuming a locally constant gradient, and approximate the affine part as $\mathbf{O}(\theta)\approx\theta+\mathbf{O}(\frac{\partial}{\partial\theta}\mathcal{L})-\theta_{k-1}$. We summarize the training process in Algorithm 1. An illustration of the loss computation is given in Figure 2.



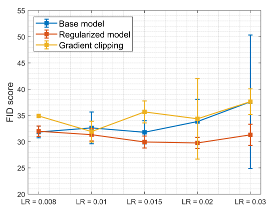
(a)



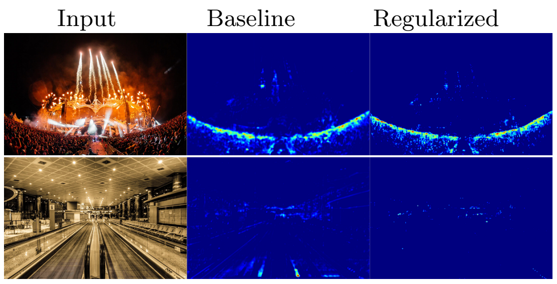
(b)



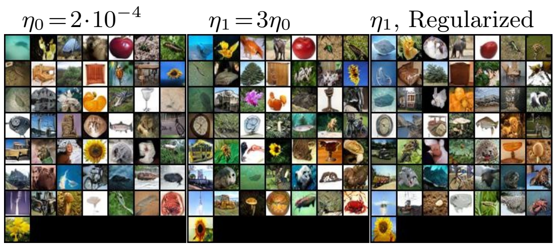
(c)



(d)



(e)



(f)

Fig. 3: **Effect of Stabilizing Regularizer.** (a) Average iterations needed to reach 90% accuracy on MNIST, with maximal and minimal iterations. Regularization reduces fluctuation across runs and can speed up convergence.(b) MAE (mean absolute error) for MCNN crowd counting network. Better and more stable results are obtained by regularizing. (c) SNGAN training with different regularizing techniques. (d) FID statistics of SNGAN achieved at different learning rates. Regularization reduces variance and can achieve better results. (e) Predicted crowd density maps from MCNN. Stabilization yields more distinct results for crowds (1st row) and sparser dots for negative samples (2nd row).(f) Generated images from fixed noise by BigGAN trained on CIFAR-100. The results of the base (2nd column) and regularized (3rd column) models are obtained with higher learning rates and half of the iterations. See Table 2 for more details. Besides, our method compares favourably also in terms of the effective training time.

4 Experiments

We now are going to empirically investigate our theoretical considerations. In particular, we evaluate Algorithm 1 on different tasks, network architectures, and datasets. The following presentation of our experiments is focused on regression-type problems, since we do not observe any significant beneficial nor detrimental effects of stability-aware training in the case of classification. Possible reasons are discussed in Section 5. Additionally, we experiment on the notoriously unstable training of GANs to show the potential of our approach in such highly dynamic environments. Our guiding hypothesis is that stability regularization can improve performance, and achieves higher consistency across multiple independent trials with random initialization. We start by providing an overview of the datasets and the tasks investigated for stability-aware training.

MNIST Digit Classification. We start with the popular MNIST [40] dataset which is designed for 10 digit classification. We convert the task into a scalar regression problem, and add a linear layer with one output unit to the end of the LeNet-5 [40] network. For evaluation, we bin the continuous output back into 10 classes. In particular, we are interested in the convergence with and without the regularizer, as measured by crossing the 90% accuracy threshold.

NWPU Crowd Counting. Next, we evaluate our regularizer on crowd counting. The task involves counting the number of people in an image of large crowds. NWPU-Crowd dataset [41] contains 5,109 images with various illumination and density range. With 351 negative samples and large appearance variations within the data, it is a challenging dataset. State-of-the-art methods typically approach crowd counting through regression of a pixel-wise density map [37, 38, 42]. The final count is obtained by summing over the density map.

CIFAR10/100 Image Generation. Furthermore, we evaluate stability regularization in the GAN setup, where gradients are naturally unstable, making it hard to train. This is due to the dynamics of the two-player game between the discriminator network (judging an image to be real or fake), and a generator, trying to fool the discriminator by generating realistic images. As training of a GAN aims to find the balance (i.e. the Nash equilibrium), unstable gradients will cause problems or even failure in training the model. We focus on CIFAR-10 [43] first to demonstrate the ability of our regularizer and compare to several other regularization methods. Test on CIFAR-100 [43] is to study the stability behavior further since there are fewer samples per class, which implicitly means the model suffers mode collapse easier.

Implementation Details. In all experiments, we stabilize the last layer only. As optimizers, we use SGD and Adam. As discussed in Section 3.4 and Algorithm 1, the generic training procedure includes two backpropagation steps. Because of the stateful nature of Adam, we copy weights and optimizer state for the last layer. Then the forward pass is performed, and we back-propagate to obtain the dynamics for the task-specific loss $\mathcal{L}^k$. Next, A_k is constructed by subtracting the old parameters from the updated ones. Finally, we update the original weights based on the gradients with respect to the complete loss $\mathcal{L}_{stable}$.

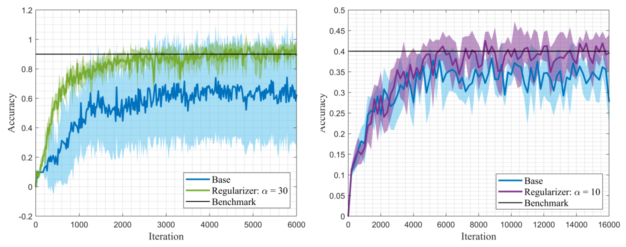


Fig. 4: **High Learning Rate behavior.** Accuracy curve across iterations for MNIST and CIFAR-10 when the learning rate is higher than the optimum setting. Top is the result for MNIST (LR = 0.12) and bottom is the result for CIFAR-10 (LR = 0.2). In both cases, the regularize improves convergence and reduces variance across runs.

4.1 Results

We summarize our main results for all tasks in Figure 3. To indicate the spread in some figures, we either plot the min/max bar, or one standard deviation as a shaded area. As a general trend, we observe that across tasks, the variance in the evaluation metric is significantly reduced when training with stabilization. In many cases, we can further see an accuracy improvement on average. The MNIST experiments in (a) show that the regularizer enables a significantly quicker convergence on average. In figure (b), we observe that a lower MAE with less variance is achieved in the case of crowd counting. The bottom row (c, d) summarizes our GAN experiments with the SNGAN [44] architecture, also exhibiting reduced variance across runs, as well as a signficiant improvement in FID across a range of learning rates. In the following, we present each of the experiments in more detail.

MNIST. We train the networks with different values for the learning rate η and the hyper-parameter α, that is, $\eta \in \{0.03, 0.05, 0.08, 0.1\}$ and $\alpha \in \{1, 3, 10, 30, 100\}$. Figure 3 (a) compares the iteration numbers needed for the base and the best regularized model to reach the benchmark for the first time under three different learning rate settings. The bar indicates the average speed over ten independent tests. The two models are comparable when the learning rate is below 0.1, and the best learning rate for the base model is around 0.08. As the learning rate increases, the base model takes more time to achieve the same accuracy, and the training process is more unstable, as indicated by the min-max distance in the figure. In contrast, our regularized model behaves better when using higher learning rates, resulting in both less average iterations and smaller variation range. From this experiment we can conclude that the regularization is orthogonal to a reduction in learning rate.

To further demonstrate the effectiveness of stabilization in the high learning rate regime, we experimented on both MNIST ($\eta = 0.12$) and CIFAR-10 ($\eta = 0.2$). The accuracy curves for the base and regularized models with error regions defined by one standard deviation in Figure 4. The optimal regularization strength α are found to be $\alpha = 30$ for MNIST and $\alpha = 10$ for CIFAR-10. As it can be seen, a high learning rate leads to performance degradation and model instability, and the base model can't reach the benchmark (0.9 and 0.4, respectively). However, if we use the stability constraint within the training, a stable behavior over multiple runs is maintained.

Crowd Counting. Next we evaluate our regularizer in the natural regression problem of crowd counting. After predicting the density map for an image, the the pixel-wise difference between the density map and group-truth map is measured with ℓ_2 loss. We follow MCNN [41] in setting up the original training hyper-parameters, including the optimizer (Adam) and the learning rate ($\eta = 1 \cdot 10^{-4}$). The results for MCNN architecture are posted in Figure 3 (b). While the Mean Absolute Error (MAE) of the base model fluctuates around 350 and finally goes down to 320, the regularized model leads to a better counting performance, as well as reduced variance. It reaches MAE 265 on average within 100 epochs with fewer fluctuations and has a minimum MAE of around 236 over ten experiments. We also compare it with gradient clipping, whose MAE curve is quite similar to the base model, with larger variance at some epochs. Our proposed regularizer is still the best among them, as can be seen in Table 1.

Method	mean MAE	best MAE
Base	328.386	284.518
Grad-clipping	301.421	281.763
Ours	**265.477**	**236.446**

Table 1: **Crowd Counting using MCNN.** Accuracy (in MAE) after 10 independent runs, for 100 epochs each.

The evaluation results for the more complex CSRNet [38], trained following the original protocol, can be seen in Figure 5. Both models behave similarly in terms of average MAE, but the regularized network shows a smaller error region over multiple runs. After epoch 60, the MAE of the base model largly fluctuates, while the regularized model generally decreases with few fluctuations. A similar behavior is observed on two other metrics: Mean Squared Error (MSE) and mean Normalized Absolute Error (NAE) in Figure 5. This may indicate that our approach is indeed helpful for the stability of the training process in the sense of stochastic dataset sampling and random initialization.

Generative Adversarial Network. We now investigate our regularizer applied to the naturally unstable GAN training. We choose SNGAN and BigGAN [45] for our experiments. We use hinge loss in both models which is the default loss function in BigGAN. We test SNGAN with SGD optimizer and BigGAN with Adam. At every epoch, we reset the history matrix, as explained in Section 3.3. For evaluation, we mainly compare the FID metric (less is better), which measures the diversity and quality of the generated images [46].

Motivated by the previous results, we start with a learning rate slightly higher than that of the best base model, and show the results for SNGAN in Figure 3 (c). Around 100 epochs, there is a four-unit FID gap between them, and the blue shaded area (base model) is much larger over the whole process. We

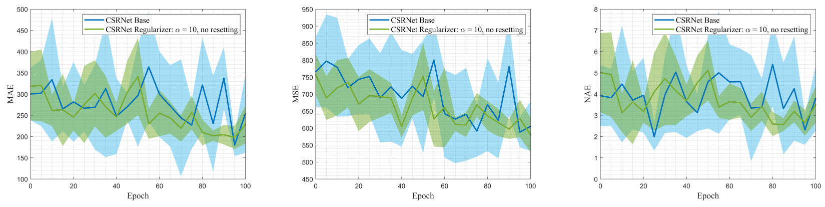


Fig. 5: **Crowd Counting using CSRNet.** From left to right: measured in MAE (mean absolute error), MSE (mean squared error), and NAE (MAE normalized by ground truth). Regularization significantly reduces the variance across runs.

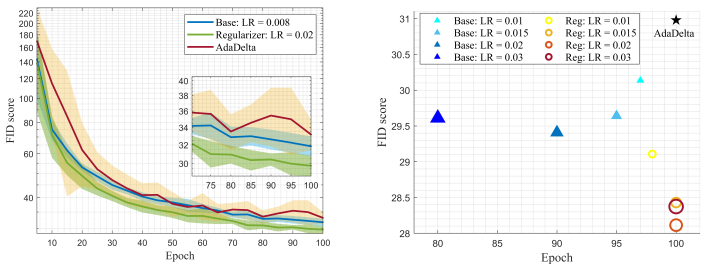


Fig. 6: **Regularization Impact on FID using SNGAN.** *Left:* Comparison between the best base model, our best regularized model and model trained by AdaDelta optimizer. *Right:* Lowest FID achieved within 100 epochs with and without regularizer.

also compare to the commonly used regularization methods weight decay and gradient clipping, to further demonstrate that our proposed regularizer is not equivalent to them in the GAN setup. To be similar to our stabilizing regularizer, we constrain the maximal matrix norm equal to one in gradient clipping. For the coefficient of weight decay, we alter among 0.1, 0.01, and 0.001. Since there is no significant difference in the results, we finally choose 0.01 for the comparison. As can be seen in the same figure, gradient clipping fails to improve the performance of SNGAN together with an even larger standard deviation. Weight decay is better than the other two models but still not as good as our regularized one. Note that even though we observe a large error region at epoch 10 or 20, it doesn't mean we can actually reach FID 30 here. This is merely due to the distortion of high-FID outliers on the symmetric standard deviation measure.

We also test our regularizer at different learning rates, and the quantitative results are reported in Figure 3 (d). Similar to the conclusion from the regression task, there is a negligible difference if we set the learning rate as the best for the base model. As the learning rate increases, the FID increases with a significantly larger variance, while the regularizer can maintain stable results, and even improve FID. Again, we see that the regularizer does not have the same effect on training as learning rate tuning, and offers new and better operating points.

Figure 6 left compares the best-regularized model with a learning-rate-setting-free optimizer, AdaDelta [27]. While the regularized model has an average FID below 30, the other two models converge around or over 32. We can conclude from here that a moderately higher learning rate with a regularizer will benefit the training for SNGAN, and it is necessary to find such an appropriate learning rate. To summarize the results for SNGAN, we provide the lowest FID every model can reach after 100 epochs in Figure 6 right. The best FID appears earlier if we increase the learning rate for the base model, indicating that the model is usually under-fitting. Adding the regularizer allows for using even higher learning rates, and the resulting FID exceeds the base model.

$\eta_0 = 2 \cdot 10^{-4}$							
Iters	55k	60k	65k	70k	75k	80k	85k
Base	9.641	9.363	9.078	**8.850**	8.900	8.972	8.865
Reg.	10.097	9.896	9.710	9.553	9.479	9.5470	9.508
$\eta_1 = 6 \cdot 10^{-4}$							
Iters	40k	45k	50k	55k	60k	65k	70k
Base	10.280	9.516	9.629	10.708	14.114	53.463	91.141
Reg.	9.396	9.062	**8.605**	9.048	10.802	27.608	82.379

Table 2: **BigGAN Training.** FID for CIFAR-100 when the learning rate is the default setting (top), and increased by a factor of 3 (bottom). With the higher learning rate, the regularized model can reach the lowest FID at 50k iterations.

We now test with the BigGAN architecture on the CIFAR-100 dataset. Table 2 shows the FID for two learning rates: default $\eta_0 = 2 \cdot 10^{-4}$, and three times higher at $\eta_1 = 6 \cdot 10^{-4}$. The behavior for default η_0 similar to previous tasks, which reveals that the regularizer malfunctions in this case, yielding a slightly higher FID than the base model. Increasing to η_1, the training process becomes fragile, and both models collapse after 65k iterations. Despite this, our regularized model can still reach an FID of 8.605, while the base model reaches only 9.516. Compared to the default setting, the lowest FID is achieved already at 50k iterations with the regularizer at η_1, as opposed to 70k iterations with the base model at η_0 . This observation indicates that the proposed regularizer can help with the stability of the network in some cases.

5 Discussion

While our formulation in Section 2 makes no assumption on the loss functions, we do not observe any significant impact of our regularizer on classification tasks such as CIFAR-100. We conjecture that regression, GAN training, and classification (using cross-entropy) losses introduce different types of dynamics, and only a subset can be stabilized with the proposed method.

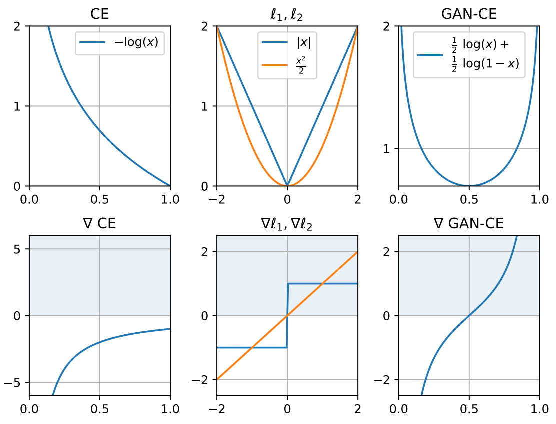


Fig. 7: **Loss profiles.** From left to right: Cross-entropy (CE), ℓ_1, ℓ_2, and GAN discriminator training (CE with balanced adversarial samples) and their corresponding gradients. Updates with big steps cause gradient inversion and/or an increase in gradient magnitude for ℓ_1, ℓ_2 and GAN losses. This is not the case for CE, where the gradient decreases without zero crossing.

One possible explanation for this phenomenon could lie in the zero crossing of the loss gradient around the optimum: for ℓ_1 and ℓ_2 loss functions, too big an update step close to the correct answer causes the gradient to invert and symmetrically regain its magnitude, as shown in Figure 7. This behavior may be a source of oscillation, that our method can compensate well. Such symmetry is not present in the one-sided cross-entropy loss, where the gradient (magnitude) monotonously decreases until the predicted score saturates. Even a big update step never causes a gradient to invert. In the case of GANs, even though technically a real/fake classification task, this symmetry might get introduced by the gradient reversal of the adversarial training.

6 Conclusion

In this work, we formulated the training of neural networks as a dynamic system and analyzed the stability behavior from the control theory point of view. Based on this theory, we develop a regularizer to stabilize the training process. The regularization strategy is based on the accumulated gradient information of the task-specific loss. Experimental results on different tasks, network architectures, and datasets show that our method stabilizes the training process across multiple independent runs and improves the task performance in conjunction with an appropriate slightly higher learning rate. That being said, we further found that the effect of the stabilization is orthogonal to a reduction of the learning rate. In many cases, we observe higher consistency and better results can be achieved with the appropriate parameters. For future work, it would be interesting to investigate why the stabilization has little effect on classification problems. Furthermore, we aim to analyse the effect of stabilization on other layers of the network, and see if it is beneficial to stabilize more than one layer.

References

1. Cauchy, A.: Methode generale pour la resolution des systemes d'equations simultanees. C.R. Acad. Sci. Paris **25** (1847) 536–538
2. Nesterov, Y.: A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. (1983)
3. Sutton, R.S.: Two problems with backpropagation and other steepest-descent learning procedures for networks. In: Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Hillsdale, NJ: Erlbaum (1986)
4. Duchi, J.C., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. In: J. Mach. Learn. Res. (2011)
5. McMahan, H.B., Streeter, M.J.: Delay-tolerant algorithms for asynchronous distributed online learning. In: NIPS. (2014)
6. Recht, B., Ré, C., Wright, S.J., Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: NIPS. (2011)
7. Zhang, S., Choromańska, A., LeCun, Y.: Deep learning with elastic averaging sgd. In: NIPS. (2015)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2015)
9. Dozat, T.: Incorporating nesterov momentum into adam. (2016)
10. Linnainmaa, S.: Taylor expansion of the accumulated rounding error. BIT Numerical Mathematics **16** (1976) 146–160
11. Griewank, A.: Who invented the reverse mode of differentiation. (2012)
12. LeCun, Y., Bottou, L., Orr, G., Müller, K.: Efficient backprop. In: Neural Networks: Tricks of the Trade. (2012)
13. Boyd, S.P., Vandenberghe, L.: Convex optimization. IEEE Transactions on Automatic Control **51** (2006) 1859–1859
14. Helmke, U., Moore, J.: Optimization and dynamical systems. Proceedings of the IEEE **84** (1996) 907–
15. Kendall, A., Gal, Y., Cipolla, R.: Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018) 7482–7491
16. Ranjan, R., Patel, V., Chellappa, R.: Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **41** (2019) 121–135
17. Popović, N., Paudel, D., Probst, T., Sun, G., Gool, L.: Compositetasking: Understanding images by spatial composition of tasks. ArXiv **abs/2012.09030** (2020)
18. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. ArXiv **abs/1708.04896** (2020)
19. Zhang, H., Cissé, M., Dauphin, Y., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. ArXiv **abs/1710.09412** (2018)
20. Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical data augmentation with no separate search. ArXiv **abs/1909.13719** (2019)
21. Vahdat, A., Kautz, J.: Nvae: A deep hierarchical variational autoencoder. ArXiv **abs/2007.03898** (2020)
22. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. CoRR **abs/1312.6114** (2014)
23. Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: ICML. (2014)

24. Pascanu, R., Mikolov, T., Bengio, Y.: Understanding the exploding gradient problem. ArXiv **abs/1211.5063** (2012)
25. Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., Finn, C.: Gradient surgery for multi-task learning. ArXiv **abs/2001.06782** (2020)
26. Vuckovic, J.: Kalman gradient descent: Adaptive variance reduction in stochastic optimization. ArXiv **abs/1810.12273** (2018)
27. Zeiler, M.D.: Adadelta: An adaptive learning rate method. ArXiv **abs/1212.5701** (2012)
28. Man, Z., Wu, H.R., Liu, S.X.F., Yu, X.: A new adaptive backpropagation algorithm based on lyapunov stability theory for neural networks. IEEE Transactions on Neural Networks **17** (2006) 1580–1591
29. Kang, Q., Song, Y., Ding, Q., Tay, W.P.: Stable neural ode with lyapunov-stable equilibrium points for defending against adversarial attacks. ArXiv **abs/2110.12976** (2021)
30. Ahmadi, A.A., Parrilo, P.A.: Joint spectral radius of rank one matrices and the maximum cycle mean problem. In: 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), IEEE (2012) 731–733
31. Ahmadi, A.A., Jungers, R.M., Parrilo, P.A., Roozbehani, M.: Joint spectral radius and path-complete graph lyapunov functions. SIAM Journal on Control and Optimization **52** (2014) 687–717
32. Altschuler, J.M., Parrilo, P.A.: Lyapunov exponent of rank-one matrices: Ergodic formula and inapproximability of the optimal distribution. SIAM Journal on Control and Optimization **58** (2020) 510–528
33. Daubechies, I., Lagarias, J.C.: Two-scale difference equations ii. local regularity, infinite products of matrices and fractals. SIAM Journal on Mathematical Analysis **23** (1992) 1031–1079
34. Deaecto, G.S., Egidio, L.N.: Practical stability of discrete-time switched affine systems. 2016 European Control Conference (ECC) (2016) 2048–2053
35. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems. (2014) 2672–2680
36. Shahbazi, M., Huang, Z., Paudel, D.P., Chhatkuli, A., Van Gool, L.: Efficient conditional gan transfer with knowledge propagation across classes. In: 2021 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021. (2021)
37. Zhang, Y., Zhou, D., Chen, S., Gao, S., Ma, Y.: Single-image crowd counting via multi-column convolutional neural network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2016) 589–597
38. Li, Y., Zhang, X., Chen, D.: Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2018) 1091–1100
39. Farina, L., Rinaldi, S.: Positive linear systems: Theory and applications. (2000)
40. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86** (1998) 2278–2324
41. Wang, Q., Gao, J., Lin, W., Li, X.: Nwpu-crowd: A large-scale benchmark for crowd counting and localization. IEEE transactions on pattern analysis and machine intelligence **43** (2020) 2141–2149
42. Sun, G., Liu, Y., Probst, T., Paudel, D., Popović, N., Gool, L.: Boosting crowd counting with transformers. ArXiv **abs/2105.10926** (2021)
43. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. (2009)

44. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957 (2018)
45. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096 (2018)
46. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: NIPS. (2017)