Training Recurrent Neural Networks by Diffusion

Hossein Mobahi
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology
Cambridge, MA, USA
hmobahi@csail.mit.edu

Abstract

This work presents a new algorithm for training recurrent neural networks (although ideas are applicable to feedforward networks as well). The algorithm is derived from a theory in nonconvex optimization related to the diffusion equation. The contributions made in this work are two fold. First, we show how some seemingly disconnected mechanisms used in deep learning such as smart initialization, annealed learning rate, layerwise pretraining, and noise injection (as done in dropout and SGD) arise naturally and automatically from this framework, without manually crafting them into the algorithms. Second, we present some preliminary results on comparing the proposed method against SGD. It turns out that the new algorithm can achieve similar level of generalization accuracy of SGD in much fewer number of epochs.

1 Introduction

Deep learning has recently beaten records in image recognition [Krizhevsky et al., 2012], speech recognition [Hinton et al., 2012a] and has made significant improvements in natural language processing [Bahdanau et al., 2014, Sutskever et al., 2014]. However, currently "training" deep networks, and specially recurrent neural networks (RNNs), is a challenging task [Martens and Sutskever, 2011]. To improve learning (in terms of convergence speed, attained training cost and generalization error) gradient based optimization methods are often used in combination with other techniques such as smart initialization [Sutskever et al., 2013], layerwise pretraining [Bengio et al., 2007], dropout [Hinton et al., 2012b], annealed learning rate, and curriculum learning [Bengio et al., 2009].

The difficulty in training deep networks is mainly attributed to their optimization landscape, where saddle points [Dauphin et al., 2014, Pascanu et al., 2014], plateaus, and sharp curvatures are prevalent. A general strategy for tackling difficult optimization problems is the ***continuation method***. This method gradually transforms a highly simplified version of the problem back to its original form while following the solution along the way. The simplified problem is

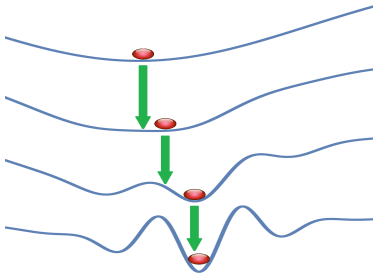


Figure 1: Optimization by the continuation method. Top is the simplified function and bottom is the original complex objective function. The solution of each subproblem initializes the subproblem below it.

supposedly easy to solve. Then, each intermediate subproblem is initialized by the solution from the previous subproblem until reaching the final problem (see Figure 1).

There are two loose ends for using optimization by continuation: 1. how to choose the simplified problem, 2. how to transform the simplified problem to the main task. For both of these questions, there are infinite answers. More precisely, given an objective function, there are infinite ways infinite smooth convex functions that could be used as initial "easy" task, and also infinite ways to gradually transform that to the main objective function. The quality of the solution attained by the continuation method ***critically depends*** on these choices. Recently we have proved that these choices can be made optimally via the ***diffusion equation*** [Mobahi and Fisher III, 2015]. Specifically, the objective function is considered as the initial heat distribution on a domain, and the heat is diffused over time according to the ***heat equation***.

The solution to the heat equation on $\mathbb{R}^n$ is known analytically: it is the ***convolution*** of the initial heat distribution (i.e., the objective function) with the ***Gaussian kernel***. Obviously, convolution with the Gaussian kernel smoothes the objective function[1]. The bandwidth parameter σ of the Gaussian kernel determines the amount of smoothing. The algorithm for optimization by diffusion starts from a large σ (highly simplified objective function), and then follows the minimizer as σ shrinks toward zero (which leads to the original cost function).

The optimality result we derived in [Mobahi and Fisher III, 2015] is a stepping stone for developing practical algorithms. Specifically, it suggests using Gaussian convolution for creating intermediate optimization tasks, but it does not answer whether the resulted convolution could be computed efficiently or not. In fact, the answer to this question is problem specific. We have shown that for some family of functions such as multivariate polynomials, the resulted convolution can be computed in ***closed form*** [Mobahi, 2016]. In this work,

[1] This happens when the objective function has well-defined Fourier transform. Then the convolution transform to product in the frequency domain. As the Fourier transform of the Gaussian is also a Gaussian, the resulted product attenuates higher frequencies.

we push that result further and show that, up to very reasonable approximation, common objective functions arising in deep learning also have a closed form Gaussian convolution. This is surprising because such objective function is highly nonlinear; involving a nested form of ill-behaved activation functions as such sign and ReLU.

By studying deep learning through the diffusion and continuation method, we discover two interesting observations. First, from theoretical viewpoint, some common and successful techniques to improve learning, such as noise injection [Hinton et al., 2012b], layerwise pretraining [Bengio et al., 2007], and annealed learning rate, ***automatically emerge*** from the diffused cost function. Therefore, this theory ***unifies*** some seemingly isolated techniques. Second, from a practical viewpoint, training deep networks by this method seems to result in a significant speed up compared to stochastic gradient descent (SGD) method. The preliminary results presented in this draft indicate up to 25% ***reduction in training time*** for learning RNNs.

This article is organized as follows. We first show that the diffused form of common activation functions has a closed form expression. After that, when we compute the diffused cost function for training a deep network, where the result depends on the diffused activation function introduced earlier. We discuss some properties of the diffused cost function and make connections to noise injection [Hinton et al., 2012b], layerwise pretraining [Bengio et al., 2007], and annealed learning rate. We conclude this article by presenting a preliminary evaluation of the proposed algorithm against SGD.

2 Optimization by Diffusion and Continuation

The optimality of using the diffusion equation for creating intermediate optimization problems is studied in our earlier work [Mobahi and Fisher III, 2015]. Briefly, diffusion is a relaxation of a time evolution process that converts an objective function to its convex envelope[2] [Vese, 1999]. The latter is a ***nonlinear*** partial differential equation that lacks a closed form, but once ***linearized***, the heat equation (a special type of diffusion equation) arises,

$$\frac{d}{dt}g(\boldsymbol{x},t) = \Delta_{\boldsymbol{x}} g(\boldsymbol{x},t) \quad , \quad \text{s.t. } g(\boldsymbol{x},0) = f(\boldsymbol{x})\,. \tag{1}$$

Here f is the original objective function, and g is its time evolution according to the heat equation. Here $\Delta_{\boldsymbol{x}}$ is the Laplace operator w.r.t. the variable $\boldsymbol{x}$. Diffusion is a powerful tool for simplifying the objective function. For example, the number of local minima in the Ackley's function [Ackley, 1987] is exponential in the number of variables. By diffusing this function via the heat equation, however, all local minima eventually disappear (see Figure 2).

[2]The convex envelope of a function is an interesting choice (versus any other convex function) for the initial simplified version of it for various reasons. 1. Any global minimizer of the cost function is also a global minimizer of its convex envelope. 2. it provides the tightest convex underestimator of the cost function. 3. Geometrically, tt is the function whose epigraph coincides with the convex hull of the epigraph of the cost function.

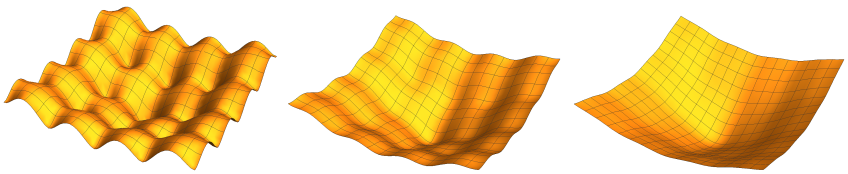


Figure 2: Diffusion of Ackley's function with time progressing from the left to the right plot.

Algorithm 1 Algorithm for Optimization by Diffusion and Continuation

1: Input: $f : \mathcal{X} \to \mathbb{R}$, Sequence $\infty > \sigma_0 > \sigma_1 > \cdots > \sigma_m = 0$.
2: $\boldsymbol{x}_0 =$ global minimizer of $g(\boldsymbol{x}; \sigma_0)$.
3: **for** $k = 1$ **to** m **do**
4: $\quad \boldsymbol{x}_k =$ Local minimizer of $g(\boldsymbol{x}; \sigma_k)$, initialized at $\boldsymbol{x}_{k-1}$.
5: **end for**
6: Output: $\boldsymbol{x}_m$

Going from the nonlinear PDE of [Vese, 1999] to the (linear) heat equation is computationally of great value. That is, the solution to the heat equation is known analytically [Widder, 1975]: it is the Gaussian convolution of the original (objective) function and the bandwidth parameter of the Gaussian determines the time point at which the diffused function is evaluated. Diffusion combined with the path following lead to a simple optimization algorithm listed in Algorithm 1.

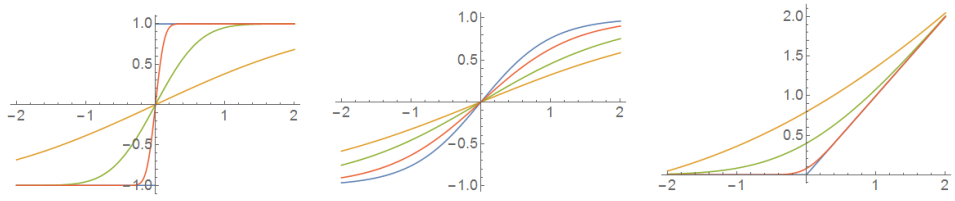
3 Diffused Activation Functions

Let $k_\sigma(\boldsymbol{w})$ be the Gaussian kernel with zero mean and covariance $\sigma^2 \boldsymbol{I}$. The diffused activation functions listed in Table 3 are simply obtained[3] by convolving them with the Gaussian k_σ. Similar forms of smoothed ReLU and sign are used by [Zhang et al., 2015] with a fixed $\sigma = \frac{1}{\sqrt{2\pi}}$, for a proving learnability of deep networks.

[3] All listed diffused functions are exact except tanh. Unfortunately, $\tanh \star k_\sigma$ does not have a closed form. We leverage the approximation $\tanh(y) \approx \operatorname{erf}(\frac{\sqrt{\pi}}{2} y)$. Notice that we know the exact diffused form for erf as listed in the table. Thus, by convolving both sides with k_σ we obtain $[\tanh \star k_\sigma](y) \approx \operatorname{erf}(\frac{\sqrt{\pi}}{2} \frac{y}{\sqrt{1+\frac{\pi}{2}\sigma^2}})$. The R.H.S. of the latter form can be again approximated via $\tanh(y) \approx \operatorname{erf}(\frac{\sqrt{\pi}}{2} y)$. This leads to the approximate identity $[\tanh \star k_\sigma](y) \approx \tanh(\frac{y}{\sqrt{1+\frac{\pi}{2}\sigma^2}})$.

Name	Original	Diffused
Sign	$\operatorname{sign}(x)$	$\operatorname{erf}(\frac{x}{\sqrt{2}\sigma})$
Error	$\operatorname{erf}(ax)$	$\operatorname{erf}(\frac{ax}{\sqrt{1+2(a\sigma)^2}})$
Tanh	$\tanh(x)$	$\tanh(\frac{x}{\sqrt{1+\frac{\pi}{2}\sigma^2}})$
ReLU	$\max(0,x)$	$\frac{\sigma}{\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}} + \frac{1}{2}x\big(1+\operatorname{erf}(\frac{x}{\sqrt{2}\sigma})\big)$

Table 1: List of some functions and their diffused form by the heat kernel.



Plot of smoothed responses of activation functions within $x \in [-2, 2]$. Blue is the original function. Red, green, and orange show the suggested functions with $\sigma_{\text{red}} < \sigma_{\text{grn}} < \sigma_{\text{orn}}$.

4 Training RNNs

4.1 RNN Cost Function

Given a set of S training sequences, each of length T. Denote the s'th sequence by $\langle(\boldsymbol{x}_{s,1}, \boldsymbol{y}_{s,1}), \ldots, (\boldsymbol{x}_{s,T}, \boldsymbol{y}_{s,T})\rangle$. Given some discrepancy function d. The problem of sequence learning by an RNN can be stated as below,

$$\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{m}_0,\boldsymbol{U},\boldsymbol{V},\boldsymbol{W}} \sum_{s=1}^{S}\sum_{t=1}^{T} d(h(\boldsymbol{n}_{s,t}) - \boldsymbol{y}_{s,t}) \tag{2}$$

$$\text{s.t.} \quad \boldsymbol{n}_{s,t} \triangleq \boldsymbol{W}\, h(\boldsymbol{m}_{s,t}) + \boldsymbol{b} \tag{3}$$

$$\boldsymbol{m}_{s,t} \triangleq \boldsymbol{U}\boldsymbol{x}_{s,t} + \boldsymbol{V} h(\boldsymbol{m}_{s,t-1}) + \boldsymbol{a}\,, \tag{4}$$

where $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{m}_0$, $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are the weights of the network. Denote the dimension of $\boldsymbol{x}_{s,t}$ and $\boldsymbol{y}_{s,t}$ be X and Y respectively. Also denote the number of neurons by H. Then, $\boldsymbol{a}$ is $H \times 1$, $\boldsymbol{b}$ is $Y \times 1$, $\boldsymbol{m}_0$ is $H \times 1$, $\boldsymbol{W}$ is $Y \times H$, $\boldsymbol{U}$ is $H \times X$, and $\boldsymbol{V}$ is $H \times H$. Obviously $\boldsymbol{n}_{s,t}$ is $Y \times 1$ and $\boldsymbol{m}_{s,t}$ is $H \times 1$.

Suppose $\boldsymbol{m}_{s,0} = \boldsymbol{m}_0$, i.e. the initial state is independent of the training sequence. Here h is some activation function. When the argument of h is a vector, the result will be a vector of the same size, whose entries consists of the element-wise application of h.

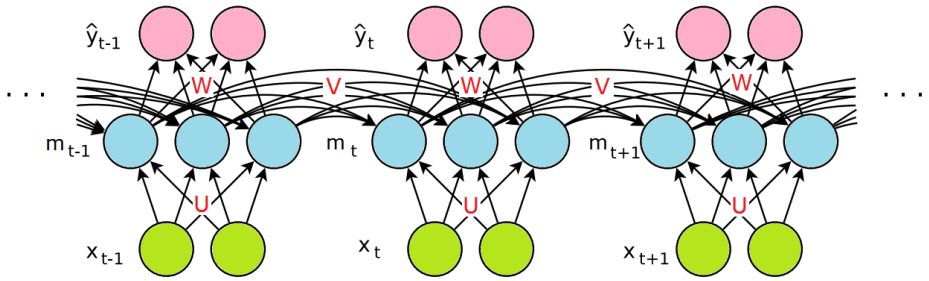


Figure 3: A Recurrent Neural Network. Figure is adapted with permission from [Martens and Sutskever, 2011] and slightly modified.

Treating each $\boldsymbol{n}_{s,t}$ and $\boldsymbol{m}_{s,t}$ as ***independent variables*** and forcing their definition (equality) by some penalty function, we arrive at the following ***unconstrained*** problem,

$$\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{m}_0,\boldsymbol{U},\boldsymbol{V},\boldsymbol{W},\boldsymbol{M},\boldsymbol{N}} \sum_{s=1}^{S}\sum_{t=1}^{T} \quad d(h(\boldsymbol{n}_{s,t}) - \boldsymbol{y}_{s,t}) \\ + \quad \lambda\Big(p\big(\boldsymbol{W}\,h(\boldsymbol{m}_{s,t}) + \boldsymbol{b} - \boldsymbol{n}_{s,t}\big) \;+\; p\big(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}h(\boldsymbol{m}_{s,t-1}) \;+ \boldsymbol{a} - \boldsymbol{m}_{s,t}\big)\Big)\,,$$

where the notation $\boldsymbol{N}$ and $\boldsymbol{M}$ are matrices whose columns are comprised of $\boldsymbol{n}_{s,t}$ and $\boldsymbol{m}_{s,t}$ for all choices of (s,t).

Letting, $d(\boldsymbol{e}) \triangleq \|\boldsymbol{e}\|^2$ (mean squared error) and $p(\boldsymbol{e}) \triangleq \|\boldsymbol{e}\|^2$ (quadratic penalty), the problem can be expressed as below,

$$\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{m}_0,\boldsymbol{U},\boldsymbol{V},\boldsymbol{W},\boldsymbol{M},\boldsymbol{N}} \sum_{s=1}^{S}\sum_{t=1}^{T} \quad \|h(\boldsymbol{n}_{s,t}) - \boldsymbol{y}_{s,t}\|^2 \\ + \quad \lambda\big(\|\boldsymbol{W}\,h(\boldsymbol{m}_{s,t}) + \boldsymbol{b} - \boldsymbol{n}_{s,t}\|^2 \;+\; \|\boldsymbol{U}\boldsymbol{x}_{s,t} + \boldsymbol{V}h(\boldsymbol{m}_{s,t-1}) \;+ \boldsymbol{a} - \boldsymbol{m}_{s,t}\|^2\big)\,.$$

Here λ determines the weight of the penalty for constraint violation.

4.2 Diffused Cost

When the objective function is evolved according to the diffusion equation (1), the diffused objective has a closed form expression. Specifically, it is obtained by the convolution of the original objective with the Gaussian kernel. This can be more formally expressed as the following. Arrange all optimization variables into a long vector $\boldsymbol{w}$, i.e. $\boldsymbol{w} \triangleq \mathrm{vec}(\boldsymbol{a},\boldsymbol{b},\boldsymbol{m}_0,\boldsymbol{U},\boldsymbol{V},\boldsymbol{W},\boldsymbol{M},\boldsymbol{N})$. Hence, the cost function can be denoted by $f(\boldsymbol{w})$. The diffused cost function g is obtained by:

$$g(\boldsymbol{w};\sigma) \triangleq [f \star k_\sigma](\boldsymbol{w})\,. \tag{5}$$

After computing this convolution, the variables in $\boldsymbol{w}$ can be replaced by their original names according to the arrangements made in $\boldsymbol{w} \triangleq \text{vec}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{m}_0, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{M}, \boldsymbol{N})$.

Denote the diffused form of the activation function h by $\tilde{h}_\sigma$, that is $\tilde{h}_\sigma(x) \triangleq [h \star k_\sigma](x)$. Similarly, define $\widetilde{h^2_\sigma}(x) \triangleq [h^2 \star k_\sigma](x)$. The diffused cost w.r.t. optimization variables has the following closed form (see Appendix A):

$$\begin{aligned}\sum_{s=1}^{S}\Bigg(\sum_{t=1}^{T} \quad & \|\widetilde{h_\sigma}(\boldsymbol{n}_{s,t}) - \boldsymbol{y}_{s,t}\|^2 + \|\sqrt{\widetilde{h^2_\sigma}}(\boldsymbol{n}_{s,t})\|^2 - \|\widetilde{h_\sigma}(\boldsymbol{n}_{s,t})\|^2 \\ + \quad & \lambda\big(\|\boldsymbol{W}\,\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}) + \boldsymbol{b} - \boldsymbol{n}_{s,t}\|^2 + \|\boldsymbol{U}\boldsymbol{x}_{s,t} + \boldsymbol{V}\widetilde{h_\sigma}(\boldsymbol{m}_{s,t-1}) + \boldsymbol{a} - \boldsymbol{m}_{s,t}\|^2 \\ & + \|\boldsymbol{W}\ \text{diag}(\sqrt{\widetilde{h^2_\sigma}}(\boldsymbol{m}_{s,t}))\|_F^2 - \|\boldsymbol{W}\ \text{diag}(\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}))\|_F^2 + \sigma^2 Y\,\|\widetilde{h_\sigma}(\boldsymbol{m}_{s,t})\|^2\big) \\ + \lambda\sum_{t=0}^{T-1} \quad & \|\boldsymbol{V}\ \text{diag}(\sqrt{\widetilde{h^2_\sigma}}(\boldsymbol{m}_{s,t}))\|_F^2 - \|\boldsymbol{V}\ \text{diag}(\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}))\|_F^2 + \sigma^2 H\,\|\widetilde{h_\sigma}(\boldsymbol{m}_{s,t})\|^2\Bigg)\,.\end{aligned}$$

Here $\|\,.\,\|_F$ denotes the Frobenius norm of a matrix.

4.3 Approximate Diffused Cost

Ideal solution requires $S \times T$ auxiliary variables for $\boldsymbol{n}_{s,t}$ and $\boldsymbol{m}_{s,t}$. This is not practical as often S is large. Thus, we resort to an ***approximate*** formulation here. Instead of solving for the optimal $\boldsymbol{n}_{s,t}$ and $\boldsymbol{m}_{s,t}$, we fix them as below,

$$\boldsymbol{n}_{s,t} \triangleq \boldsymbol{W}\,\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}) + \boldsymbol{b} \quad , \quad \boldsymbol{m}_{s,t} \triangleq \boldsymbol{U}\boldsymbol{x}_{s,t} + \boldsymbol{V}\widetilde{h_\sigma}(\boldsymbol{m}_{s,t-1}) + \boldsymbol{a}\,. \tag{6}$$

This allows us to ***drop*** $\boldsymbol{n}_{s,t}$ and $\boldsymbol{m}_{s,t}$ from the optimization variables. We computing the gradient, however, derivatives involving $\boldsymbol{n}_{s,t}$ and $\boldsymbol{m}_{s,t}$ must be handled carefully to recognize the dependency stated in (6). The simplified optimization problem is as below,

$$\begin{aligned}\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{m}_0,\boldsymbol{U},\boldsymbol{V},\boldsymbol{W}} \quad & \\ \sum_{s=1}^{S}\Bigg(\sum_{t=1}^{T} \quad & \|\widetilde{h_\sigma}(\boldsymbol{n}_{s,t}) - \boldsymbol{y}_{s,t}\|^2 + \|\sqrt{\widetilde{h^2_\sigma}}(\boldsymbol{n}_{s,t})\|^2 - \|\widetilde{h_\sigma}(\boldsymbol{n}_{s,t})\|^2 \\ + \quad & \lambda\big(\|\boldsymbol{W}\ \text{diag}(\sqrt{\widetilde{h^2_\sigma}}(\boldsymbol{m}_{s,t}))\|_F^2 - \|\boldsymbol{W}\ \text{diag}(\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}))\|_F^2 + \sigma^2 Y\,\|\widetilde{h_\sigma}(\boldsymbol{m}_{s,t})\|^2\big) \\ + \lambda\sum_{t=0}^{T-1} \quad & \|\boldsymbol{V}\ \text{diag}(\sqrt{\widetilde{h^2_\sigma}}(\boldsymbol{m}_{s,t}))\|_F^2 - \|\boldsymbol{V}\ \text{diag}(\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}))\|_F^2 + \sigma^2 H\,\|\widetilde{h_\sigma}(\boldsymbol{m}_{s,t})\|^2\Bigg) \\ \text{s.t.} \quad & \boldsymbol{n}_{s,t} \triangleq \boldsymbol{W}\,\widetilde{h_\sigma}(\boldsymbol{m}_{s,t}) + \boldsymbol{b} \quad , \quad \boldsymbol{m}_{s,t} \triangleq \boldsymbol{U}\boldsymbol{x}_{s,t} + \boldsymbol{V}\widetilde{h_\sigma}(\boldsymbol{m}_{s,t-1}) + \boldsymbol{a}\,.\end{aligned}$$

The gradient of this cost w.r.t. learning parameters are provided in Appendix B.

5 Properties of Diffused Cost

The optimization problem that arises from training a deep network is often challenging. Therefore, local optimization methods (e.g., SGD) are used with a combination of some helping techniques. Although these techniques seem disconnected from each other, some of them emerge automatically from the diffused cost function. Therefore, these techniques might be ***unified*** under one simple theory. These methods and their connection to the diffused cost are discussed in the following.

5.1 Careful Initialization

Local optimization methods are generally sensitive to initialization when it comes to nonconvex cost functions. Deep learning is not an exception [Sutskever et al., 2013]; a recent study shows that the performance of deep networks and recurrent networks critically depends on initialization [Safran and Shamir, 2015]. In contrast, the diffusion algorithm is deterministic and almost independent of initialization[4] for two reasons. First, after enough smoothing the cost function becomes unimodal, and in case of convexity, will have one global minimum. In fact, the minimizer of the heavily smoothed function coincides with its center mass [Mobahi, 2012]. Thus, diffusion provides an interesting deterministic initialization. Second, the update rules are completely deterministic (unless one chooses to use SGD instead of GD for local optimization in Algorithm 1) and no notion of randomness is involved in the updates.

5.2 Annealed Learning Rate

Each iteration of the gradient descent essentially sees the first order Taylor expansion of the cost function $g(\boldsymbol{x})$ at the current estimate of the solution point $\boldsymbol{x}_0$. The linear approximation has good accuracy only within a small neighborhood of $\boldsymbol{x}_0$, say of radius ρ. Enforcing accuracy by the constraint $\|\boldsymbol{x}-\boldsymbol{x}_0\| \leq \rho$, we arrive at the following problem,

$$\min_{\boldsymbol{x}} g(\boldsymbol{x}_0) + (\boldsymbol{x}-\boldsymbol{x}_0)^T \nabla g(\boldsymbol{x}_0) \qquad s.t. \quad \|\boldsymbol{x}-\boldsymbol{x}_0\| \leq \rho\,. \tag{7}$$

Using Lagrange multipliers method, the solution of this optimization turns out to be $\boldsymbol{x}^* = \boldsymbol{x}_0 - \rho \frac{\nabla g(\boldsymbol{x}_0)}{\|\nabla g(\boldsymbol{x}_0)\|}$.

The radius ρ could be chosen intelligently, e.g., by restricting the tolerated amount of linearization error. Specifically, in order to ensure $\forall \boldsymbol{x}\,;\, \|\boldsymbol{x}-\boldsymbol{x}_0\| \leq$

[4] Path following process could be sensitive to initialization when it reaches a saddle point. Due to instability of saddle points, the direction the algorithm takes could be affected even by small perturbations. Thus, different initializations may end up reaching different solutions. However, these saddle points often occur due to the symmetry in the problem (either the original or the diffused) and the chosen direction does not affect the quality of the solution. This contrasts to gradient descent on a nonconvex objective, where depending on initialization, very solutions of different quality might be reached.

$\rho \Rightarrow |g(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla g(\boldsymbol{x}_0) - g(\boldsymbol{x})| \leq \epsilon$, we can choose $\rho = \sqrt{\frac{\epsilon}{c_f}}\sigma$ (see Appendix C for proof). Here c_f is some number satisfying $c_f \geq \frac{1}{2\pi} \sum_{j,k} \| \frac{d^2 f}{dx_j \, dx_k} \|_{\frac{n}{2}}$, which obviously exists when the norm is bounded.

Putting the pieces together, the solution of the linearized problem can be expressed as $\boldsymbol{x}^* = \boldsymbol{x}_0 - \eta \, \sigma \, \frac{\nabla g(\boldsymbol{x}_0)}{\|\nabla g(\boldsymbol{x}_0)\|}$, where $\eta \triangleq \sqrt{\frac{\epsilon}{c_f}}$ is a constant. This is essentially a gradient descent update with a specific choice of the step size. Since σ decays toward zero within the continuation loop, the step size (also called learning rate) anneals form an initially large value to eventually a small value.

5.3 Noise Injection

Injection of random noise into the training process can lead to more stable solutions. This is often crucial in order to obtain satisfactory generalization in deep learning. The well known ***dropout*** is a specific way of noise injection: in each iteration, it eliminates a random subset of nodes throughout the learning [Hinton et al., 2012b]. The stochasticity in SGD is another relevant example. It is known that SGD achieves better generalization compared to a full batch gradient descent. More recently, it has been shown that adding Gaussian noise to the computed gradient can significantly improve learning for very deep networks [Neelakantan et al., 2015]. Although these schemes differ in details, e.g., the distribution of the noise or how it is applied to the learning process, they share the same idea of noise injection in learning.

It turns out that the diffused cost function also has this property. In order to see that, recall the definition of the diffused cost function from (5):

$$g(\boldsymbol{w}; \sigma) \triangleq [f \star k_\sigma](\boldsymbol{w}) = \int_{\mathcal{W}} f(\boldsymbol{w} - \boldsymbol{t}) k_\sigma(\boldsymbol{t}) \, d\boldsymbol{t} \tag{8}$$

Thus, the gradient at a point $\boldsymbol{w}_0$ has the following form.

$$\nabla g(\boldsymbol{w}_0; \sigma) = \int_{\mathcal{W}} \nabla f(\boldsymbol{w}_0 - \boldsymbol{t}) k_\sigma(\boldsymbol{t}) \, d\boldsymbol{t} \tag{9}$$

$$\approx \frac{1}{J} \sum_{j=1}^{J} \nabla f(\boldsymbol{w}_0 - \boldsymbol{t}_j) \quad , \quad \boldsymbol{t}_j \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}) \, . \tag{10}$$

This means if we were to approximate the gradient of the diffused cost by MCMC method, it would average over a number of ***noisified*** gradients. Specifically, the noise would be ***additive*** w.r.t. the weights of the network and it would have a ***normal distribution*** with zero mean and variance of σ^2. The noise injection of (10) has also been used by [Bachman et al., 2014] via numerical sampling exactly as in (10). From a higher level perspective, this noise injection has some similarity to SGD; the latter also averages (over multiple epochs) the effect of noisified gradients.

A key advantage of using the diffusion framework for noise injection, however, is that the expected noisified gradient (the integral in (9)) has a ***closed***

form expression, while the other schemes are mainly ***sampling*** based. This leads to a huge computational gain for the diffusion method: while other methods would need a lot of sampling iterations in order to reach a reasonable approximation to the expected noisified gradient (and the number of these samples could grow exponentially in the number of weights), the diffusion method achieves this with almost no computational effort and without any sampling.

5.4 Layerwise Pretraining

We argue that when σ is large, the network only focuses on short range dependencies, and as σ shrinks toward zero, longer range dependencies are gradually learned. In order to see why this happens, let's for example inspect the partial gradient $\nabla_{\boldsymbol{a}}\, g$, which has the form $\sum_{t=1}^{T} \boldsymbol{r}_t\, \boldsymbol{M}_t$ (see Appendix B for derivations and the definition of $\boldsymbol{r}_t$), where $\boldsymbol{M}_t \triangleq \boldsymbol{I} + \boldsymbol{V}\, \mathrm{diag}\left(\widetilde{h}'(\boldsymbol{m}_{t-1})\right)\boldsymbol{M}_{t-1}$ and $\boldsymbol{M}_1 \triangleq \boldsymbol{I}$. Resolving the recursion in $\boldsymbol{M}_t$ leads to,

$$\boldsymbol{M}_t = \boldsymbol{I} + \boldsymbol{V}\, \mathrm{diag}\left(\widetilde{h}'_\sigma(\boldsymbol{m}_{t-1})\right) + \boldsymbol{V}\widetilde{h}'_\sigma(\boldsymbol{m}_{t-1})\, \boldsymbol{V}\widetilde{h}'_\sigma(\boldsymbol{m}_{t-2}) + \dots\,.$$

When $\sigma \to \infty$, all the sigmoid-like activation functions listed in (3) become flat and their gradient vanishes $\widetilde{h}'_\sigma \to 0$. This implies that by choosing σ large enough, one can find a small enough ϵ that satisfies $\|\,\mathrm{diag}(\widetilde{h}'_\sigma)\| \leq \epsilon$. Since the contribution of each term in the above sum will be at most equal to its matrix norm, we can derive,

$$\|\boldsymbol{M}_t\| \leq \|\boldsymbol{I}\| + \epsilon\|\boldsymbol{V}\| + (\epsilon\|\boldsymbol{V}\|)^2 + (\epsilon\|\boldsymbol{V}\|)^3 + \dots\,.$$

when σ is very large, and thus ϵ is very small, we can ignore all the terms involving ϵ, which leaves us with $\boldsymbol{M}_t \approx \boldsymbol{I}$. As we gradually reduce σ, and thus increase ϵ, we can reconsider terms involving smaller exponents, while the higher order terms still remain negligible. By gradually decreasing σ, $\boldsymbol{M}_t$ can be approximated by $\boldsymbol{I}$, then, $\boldsymbol{I}+\boldsymbol{V}\, \mathrm{diag}\left(\widetilde{h}'_\sigma(\boldsymbol{m}_{t-1})\right)$, then $\boldsymbol{I}+\boldsymbol{V}\, \mathrm{diag}\left(\widetilde{h}'_\sigma(\boldsymbol{m}_{t-1})\right)+ \boldsymbol{V}\widetilde{h}'_\sigma(\boldsymbol{m}_{t-1})\, \boldsymbol{V}\widetilde{h}'_\sigma(\boldsymbol{m}_{t-2})$ and so on.

This is conceptually very similar to layerwise pretraining [Bengio et al., 2007], as the learning in each layer starts from considering only its immediate previous layer and then gradually switches to the full consideration by considering larger and larger number of previous layers.

6 Choice of the Activation Function

In order to implement the method, we need to obtain the explicit expressions of $\widetilde{h}_\sigma$ and $\widetilde{h^2}_\sigma$ for a given activation function h. For example, suppose we set $h(x) = \mathrm{erf}(ax)$, where a is a parameter that determines the sharpness of the activation function. Note that $\lim_{a\to\infty} \mathrm{erf}(ax) = \mathrm{sign}(x)$ and $\mathrm{erf}(\frac{\sqrt{\pi}}{2}x) \approx \tanh(x)$. The form of $\widetilde{h}_\sigma$ can be already looked up from Table 3, which is repeated below,

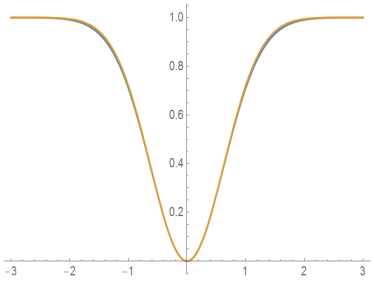


Figure 4: Blue and brown curves respectively plot $\mathrm{erf}^2(x)$ and $1 - e^{-\frac{4}{\pi}x^2}$. Due to the strong overlap, the blue curve is barely visible.

$$\tilde{h}(x) = \mathrm{erf}\left(\frac{ax}{\sqrt{1 + 2(a\sigma)^2}}\right). \tag{11}$$

In the following, we only focus on $\widetilde{h^2}_\sigma$. Unfortunately, $\widetilde{h^2}_\sigma(x)$ lacks a closed form expression. However, observe that $\mathrm{erf}^2(x) \approx 1 - e^{-\frac{4}{\pi}x^2}$. This approximation has a reasonably good accuracy as shown in Figure 4. Using this approximation, it follows that $[\mathrm{erf}^2(a\,\square) \star k_\sigma](x) \approx [1 - e^{-\frac{4}{\pi}(a\,\square)^2} k_\sigma](x)$.

$$\begin{aligned}
\widetilde{h^2}(x) &\triangleq [\mathrm{erf}^2(a\,\square) \star k_\sigma](x) \\
&\approx [1 - e^{-\frac{4}{\pi}(a\,\square)^2} k_\sigma](x) \\
&= 1 - \frac{\sqrt{\pi}\; e^{-\frac{4a^2x^2}{\pi + 8a^2\sigma^2}}}{\sqrt{\pi + 8a^2\sigma^2}}.
\end{aligned}$$

7 Preliminary Results

Here we present a comparison between SGD and the proposed diffusion framework. The hyperparameters in both methods are carefully searched to ensure a fair comparison. We use erf as the activation function. The task is to learn adding two numbers, and is adapted from [Martens and Sutskever, 2011]. The network consists of has 10 hidden units, and it has two inputs and one output. One of the input units reads a sequence of 10 real numbers, and the other a sequence of 10 binary numbers. The binary numbers are zero everywhere except two random locations. The task is to add the values from the first sequence, at the two locations marked by the second sequence.

We trained the network by 1000 sequences, and generalization is computed from a test set of 100 sequences. The result is shown in the plots. The horizontal axis shows the ***generalization error***, and the vertical axis shows how many ***epochs*** it takes to reach that generalization error. For example, with 50 batches of size 50 samples, in order to reach around error of 0.02, SGD (blue) needs about 90 epochs, while diffusion methods (red) needs about 20 epochs.

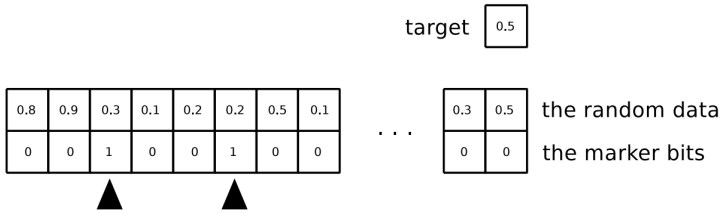


Figure 5: Learning to add by RNNs. Figure adapted with permission from [Martens and Sutskever, 2011].

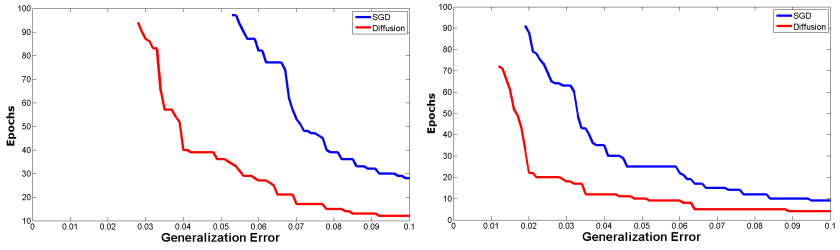


Figure 6: Experiments with mini batches of size 10 (left) and 50 (right).

8 Related Works & Future Directions

This work specifically studies the use of the diffusion equation for optimizing the objective function in deep learning. However, there is a growing number of techniques by others that propose new algorithms for deep learning. Using tensor decomposition techniques, [Janzamin et al., 2015] offers new algorithms for deep learning with performance guarantee. [Hazan et al., 2015] provides a conceptual similar algorithm to ours. However, instead of computing the convolution analytically, the latter work relies on numerical sampling. It can guarantee reaching the global minimum for certain scenarios.

This work relies on smoothing the objective function by convolving it with the Gaussian kernel. We have previously shown that this particular form of smoothing is optimal in a certain sense, by relating Gaussian convolution to a relaxation of the convex envelope. Although connection to the convex envelope is meaningful in the context of ***nonconvex*** objective functions, there are side benefits in smoothing even when the objective function is convex. For example, smoothing a nonsmooth ***convex*** objective function by convolution can improve the convergence rate of stochastic optimization algorithms [Duchi et al., 2012].

As discussed in Section 5.3, smoothing can be considered as means to inject noise into the training process. The idea of noise injection is already used in methods such as SGD or dropout [Hinton et al., 2012b] in order to improve

learning. The key advantage of our framework for noise injection, however, is that the noise injection can be achieved in closed form and without need of sampling. In order words, we can compute the effect of infinitely many noisified objective functions in closed form. This is similar to the idea of Marginalized Denoising Autoencoders (mDA) [Chen et al., 2014], where the effect of infinitely many nosified inputs is marginalized to obtain a closed form expression. However, mDA limits the form of the injected noise. Specifically, the marginalized effect is only computable in a ***linear*** reconstruction setup (nonlinearity is applied only after computation of the marginalized reconstruction). In addition, mDA performs noise injection layer by layer in a greedy fashion. In contrast, our framework is able to compute closed form expression for the entire deep network and allowing full nonconvexity of the associated optimization, up to reasonable approximation.

Diffusion equation provides an approximate evolution toward the convex envelope. Consequently, it is not perfect: if global minimum is very narrow, diffusion can miss that minima in favor of a wider minimum whose value is slightly larger than the narrow global minimum (see Figure 7). This may seem a disadvantage at the first glance. However, the wider minima are in fact more stable[5], which could be more desired in practice, e.g. generalizing better. In fact, a recent analysis has shown that SGD attains better generalization when the objective function is ***smoother*** [Hardt et al., 2015]. Note that in our framework, initializing the algorithm with larger σ automatically provides a smoother surrogate cost function where unstable minima disappear. Thus, it is more likely to remain in the basin of attraction of the stable minima. A thorough investigation of how smoothing the cost function in the diffusion setting may improve the generalization performance is a direction for future research.

A closely related work to ours is Annealed Gradient Descent [Pan and Jiang, 2015], where the objective landscape is also initially approximated by a smoother function and is gradually transformed to the original one. However, the unlike this work where Gaussian smoothing is theoretically motivated for nonconvex optimization [Mobahi and Fisher III, 2015], in [Pan and Jiang, 2015] coarse-to-fine approximation of the objective function is based on heuristically motivated procedure. More precisely, the latter uses vector quantization methods in order to generate a code book by which the coarse cost function is approximated. Another difference between these two works is that the representation of the smoothed function in our framework is simpler, as we directly obtain a closed form expression of the objective function. that is a simpler setup than approximation by codebook generation.

Very recently, the diffusion process has been proposed for learning difficult probability distributions [Sohl-Dickstein et al., 2015]. In forward time diffusion, the method converts any complex distribution into a simple distribution, e.g., Gaussian. It then learns the reverse-time of this diffusion process to define a generative model distribution. By sampling from such trained model, the

[5]By a stable minimum we mean that a small perturbation of the equilibrium resides in the basin of attraction of the same equilibrium. This is not true if the minimum is too narrow; slight perturbation may put the gradient decent into a different basin of attraction.

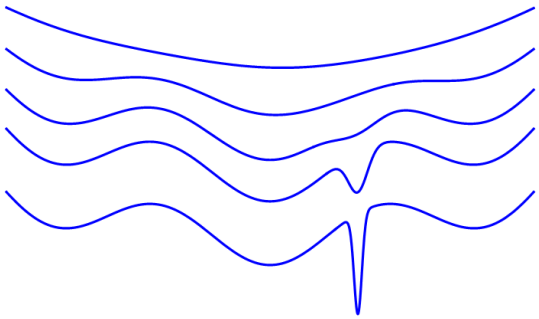


Figure 7: Starting from the original function at the bottom, moving upward the plots correspond to more aggressive smoothing (i.e. larger σ). The original function has three wide minima, and a narrow global minimum. Following the path of the minimizer from top to the bottom, it is obvious that the process misses the narrow global minimum and reaches one of the wider minima. However, among the three wide minima, it finds the lowest one.

authors have achieved inpainting of missing regions in natural images.

9 Acknowledgment

This research is partially funded by Shell Research. Hossein Mobahi is thankful to John W. Fisher, William T. Freeman, Yann LeCun, and Yoshua Bengio for comments and discussions and to Peter Bartlett and Fei Sha for suggesting connections to [Duchi et al., 2012, Chen et al., 2014]. Hossein Mobahi is grateful to Geoffrey Hinton, Marc'Aurelio Ranzato, and Philip Bachman for comments and Kate Saenko for discussions in earlier phase of this work.

References

[Ackley, 1987] Ackley, D. (1987). *A Connectionist Machine for Genetic Hill-climbing*, volume SECS28 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston. 3

[Bachman et al., 2014] Bachman, P., Alsharif, O., and Precup, D. (2014). Learning with pseudo-ensembles. In *Advances in Neural Information Processing Systems 27*. 9

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473. 1

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*. 1, 3, 10

[Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *ICML*. 1

[Chen et al., 2014] Chen, M., Weinberger, K. Q., Sha, F., and Bengio, Y. (2014). Marginalized denoising auto-encoders for nonlinear representations. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1476–1484. 13, 14

[Dauphin et al., 2014] Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. pages 2933–2941. 1

[Duchi et al., 2012] Duchi, J. C., Bartlett, P. L., and Wainwright, M. J. (2012). Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701. 12, 14

[Hardt et al., 2015] Hardt, M., Recht, B., and Singer, Y. (2015). Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240. 13

[Hazan et al., 2015] Hazan, E., Levy, K. Y., and Shalev-Shwartz, S. (2015). On graduated optimization for stochastic non-convex problems. *CoRR*, abs/1503.03712. 12

[Hinton et al., 2012a] Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012a). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97. 1

[Hinton et al., 2012b] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580. 1, 3, 9, 12

[Janzamin et al., 2015] Janzamin, M., Sedghi, H., and Anandkumar, A. (2015). Generalization bounds for neural networks through tensor factorization. *CoRR*, abs/1506.08473. 12

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 1

[Martens and Sutskever, 2011] Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *ICML*, pages 1033–1040. Omnipress. 1, 6, 11, 12

[Mobahi, 2012] Mobahi, H. (2012). *Optimization by Gaussian Smoothing with Application to Geometric Alignment*. PhD thesis, University of Illinois at Urbana Champaign. 8

[Mobahi, 2016] Mobahi, H. (2016). Closed form for some gaussian convolutions. *CoRR*. 2

[Mobahi and Fisher III, 2015] Mobahi, H. and Fisher III, J. W. (2015). On the Link Between Gaussian Homotopy Continuation and Convex Envelope. 2, 3, 13

[Neelakantan et al., 2015] Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *CoRR*, abs/1511.06807. 9

[Pan and Jiang, 2015] Pan, H. and Jiang, H. (2015). Annealed gradient descent for deep learning. In *Proc. of 31th Conference on Uncertainty in Artificial Intelligence (UAI 2015)*. 13

[Pascanu et al., 2014] Pascanu, R., Dauphin, Y. N., Ganguli, S., and Bengio, Y. (2014). On the saddle point problem for non-convex optimization. *CoRR*, abs/1405.4604. 1

[Safran and Shamir, 2015] Safran, I. and Shamir, O. (2015). On the quality of the initial basin in overspecified neural networks. *CoRR*, abs/1511.04210. 8

[Sohl-Dickstein et al., 2015] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585. 13

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 1, 8

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. 1

[Vese, 1999] Vese, L. (1999). A method to convexify functions via curve evolution. *Commun. Partial Differ. Equations*, 24(9-10):1573–1591. 3, 4

[Widder, 1975] Widder, D. V. (1975). *The Heat Equation*. Academic Press. 4

[Zhang et al., 2015] Zhang, Y., Lee, J. D., and Jordan, M. I. (2015). ℓ_1-regularized neural networks are improperly learnable in polynomial time. *CoRR*, abs/1510.03528. 4

Appendices

A Diffused RNN Training Cost

Diffusing the cost function w.r.t. $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ yields[6],

$$\sum_{t=1}^{T} \quad \|h(\boldsymbol{n}_t) - \boldsymbol{y}_t\|^2 \tag{12}$$

$$+ \quad \lambda\big(\|\boldsymbol{W}\, h(\boldsymbol{m}_t) + \boldsymbol{b} - \boldsymbol{n}_t\|^2 \,+\, \|\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V} h(\boldsymbol{m}_{t-1}) \,+ \boldsymbol{a} - \boldsymbol{m}_t\|^2 \tag{13}$$

$$+\sigma^2 Y(1 + \|h(\boldsymbol{m}_t)\|^2) \,+\, \sigma^2 H(1 + \|\boldsymbol{x}_t\|^2 + \|h(\boldsymbol{m}_{t-1})\|^2)\big)\,. \tag{14}$$

Smoothing w.r.t. $\boldsymbol{m}_t$ and $\boldsymbol{n}_t$ leads[7] to,

$$\sum_{t=1}^{T} \quad \|\widetilde{h}(\boldsymbol{n}_t) - \boldsymbol{y}_t\|^2 + \|\sqrt{\widetilde{h^2}}(\boldsymbol{n}_t)\|^2 - \|\widetilde{h}(\boldsymbol{n}_t)\|^2 \tag{15}$$

$$+ \quad \lambda\big(\|\boldsymbol{W}\, \widetilde{h}(\boldsymbol{m}_t) + \boldsymbol{b} - \boldsymbol{n}_t\|^2 \,+\, \|\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\widetilde{h}(\boldsymbol{m}_{t-1}) \,+ \boldsymbol{a} - \boldsymbol{m}_t\|^2 \tag{16}$$

$$+\sigma^2 Y(2 + \|\widetilde{h}(\boldsymbol{m}_t)\|^2) \,+\, \sigma^2 H(2 + \|\boldsymbol{x}_t\|^2 + \|\widetilde{h}(\boldsymbol{m}_{t-1})\|^2) \tag{17}$$

$$+\|\boldsymbol{W}\, \mathrm{diag}(\sqrt{\widetilde{h^2}}(\boldsymbol{m}_t))\|_F^2 - \|\boldsymbol{W}\, \mathrm{diag}(\widetilde{h}(\boldsymbol{m}_t))\|_F^2 \tag{18}$$

$$+\|\boldsymbol{V}\, \mathrm{diag}(\sqrt{\widetilde{h^2}}(\boldsymbol{m}_{t-1}))\|_F^2 - \|\boldsymbol{V}\, \mathrm{diag}(\widetilde{h}(\boldsymbol{m}_{t-1}))\|_F^2\big)\,. \tag{19}$$

Discarding constants terms, i.e. those that do not depend on neither of optimization variables $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{M}, \boldsymbol{N}$, simplifies the diffused cost to the following,

$$\sum_{t=1}^{T} \quad \|\widetilde{h}(\boldsymbol{n}_t) - \boldsymbol{y}_t\|^2 + \|\sqrt{\widetilde{h^2}}(\boldsymbol{n}_t)\|^2 - \|\widetilde{h}(\boldsymbol{n}_t)\|^2 \tag{20}$$

$$+ \quad \lambda\big(\|\boldsymbol{W}\, \widetilde{h}(\boldsymbol{m}_t) + \boldsymbol{b} - \boldsymbol{n}_t\|^2 \,+\, \|\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\widetilde{h}(\boldsymbol{m}_{t-1}) \,+ \boldsymbol{a} - \boldsymbol{m}_t\|^2 \tag{21}$$

$$+\|\boldsymbol{W}\, \mathrm{diag}(\sqrt{\widetilde{h^2}}(\boldsymbol{m}_t))\|_F^2 - \|\boldsymbol{W}\, \mathrm{diag}(\widetilde{h}(\boldsymbol{m}_t))\|_F^2 + \sigma^2 Y\, \|\widetilde{h}(\boldsymbol{m}_t)\|^2 \tag{22}$$

$$+\lambda \sum_{t=0}^{T-1} \quad \|\boldsymbol{V}\, \mathrm{diag}(\sqrt{\widetilde{h^2}}(\boldsymbol{m}_t))\|_F^2 - \|\boldsymbol{V}\, \mathrm{diag}(\widetilde{h}(\boldsymbol{m}_t))\|_F^2 + \sigma^2 H\, \|\widetilde{h}(\boldsymbol{m}_t)\|^2\,. \tag{23}$$

[6] We use the fact that convolution of $(\boldsymbol{x}^T\boldsymbol{y})^2$ with $k_\sigma(\boldsymbol{x})$ is $(\boldsymbol{x}^T\boldsymbol{y})^2 + \sigma^2\|\boldsymbol{y}\|^2$.

[7] We use the identity that convolution of $\|\boldsymbol{A}h(\boldsymbol{x}) + \boldsymbol{b}\|^2$ with $k_\sigma(\boldsymbol{x})$ is equal to $\|\boldsymbol{A}\widetilde{h}(\boldsymbol{x}) + \boldsymbol{b}\|^2 + \|\boldsymbol{A}\, \mathrm{diag}(\sqrt{\widetilde{h^2}}(\boldsymbol{x}))\|_F^2 - \|\boldsymbol{A}\, \mathrm{diag}(\widetilde{h}(\boldsymbol{x}))\|_F^2$.

B Gradient of Diffused Cost

Below $\odot$ denotes the element-wise product of two matrices.

$$\begin{aligned}
\frac{dg}{d\boldsymbol{b}} &= \sum_t \frac{\partial \boldsymbol{n}_t}{\partial \boldsymbol{b}} \frac{\partial g}{\partial \boldsymbol{n}_t} && (24)\\
&= \sum_t \boldsymbol{I}\Big(2\widetilde{h}'(\boldsymbol{n}_t) \odot (\widetilde{h}(\boldsymbol{n}_t) - \boldsymbol{y}_t) + \widetilde{h^2}'(\boldsymbol{n}_t) - 2\widetilde{h}'(\boldsymbol{n}_t) \odot \widetilde{h}(\boldsymbol{n}_t)\Big) && (25)\\
&= \sum_t \Big(\widetilde{h^2}'(\boldsymbol{n}_t) - 2\widetilde{h}'(\boldsymbol{n}_t) \odot \boldsymbol{y}_t\Big). && (26)
\end{aligned}$$

$$\begin{aligned}
\frac{dg}{d\boldsymbol{W}} &= \sum_t \frac{\partial g}{\partial \boldsymbol{W}} + \sum_d \frac{\partial g}{\partial n_t^{(d)}} \frac{\partial n_t^{(d)}}{\partial \boldsymbol{W}} && (27)\\
&= 2\lambda \boldsymbol{W} \operatorname{diag}\Big(\sum_{t=1}^{T} \big(\widetilde{h^2}(\boldsymbol{m}_t) - \widetilde{h}^2(\boldsymbol{m}_t)\big)\Big) && (28)\\
&\quad + \sum_{t=1}^{T} \Big(\widetilde{h^2}'(\boldsymbol{n}_t) - 2\widetilde{h}'(\boldsymbol{n}_t) \odot \boldsymbol{y}_t\Big) \widetilde{h}(\boldsymbol{m}_t)^T && (29)
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{r}_t \triangleq\ & \Big(\widetilde{h^2}'(\boldsymbol{n}_t) - 2\widetilde{h}'(\boldsymbol{n}_t) \odot \boldsymbol{y}_t\Big)^T \Big(\boldsymbol{W} \operatorname{diag}(\widetilde{h}'(\boldsymbol{m}_t))\Big)\\
&+\lambda\Big(\big(\widetilde{h^2}'(\boldsymbol{m}_t) - 2\widetilde{h}'(\boldsymbol{m}_t) \odot \widetilde{h}(\boldsymbol{m}_t)\big)^T \odot \big(\mathbf{1}^T(\boldsymbol{W} \odot \boldsymbol{W}) + \mathbb{I}_{t\neq T}\mathbf{1}^T(\boldsymbol{V} \odot \boldsymbol{V})\big)\\
&\quad +2\sigma^2(\mathbb{I}_{t\neq T}H + Y)(\widetilde{h}'(\boldsymbol{m}_t) \odot \widetilde{h}(\boldsymbol{m}_t))^T\Big). \qquad (30)
\end{aligned}$$

$$\begin{aligned}
(\frac{dg}{d\boldsymbol{a}})^T &= \sum_{t=1}^{T} (\frac{dg}{d\boldsymbol{m}_t})^T \frac{d\boldsymbol{m}_t}{d\boldsymbol{a}} && (31)\\
&= \sum_{t=1}^{T} ((\frac{\partial g}{\partial \boldsymbol{n}_t})^T \frac{\partial \boldsymbol{n}_t}{\partial \boldsymbol{m}_t} + (\frac{\partial g}{\partial \boldsymbol{m}_t})^T) \frac{d\boldsymbol{m}_t}{d\boldsymbol{a}} && (32)\\
&= \sum_{t=1}^{T} \boldsymbol{r}_t \boldsymbol{M}_t && (33)\\
\boldsymbol{M}_t &\triangleq \frac{d\boldsymbol{m}_t}{d\boldsymbol{a}} = \frac{\partial \boldsymbol{m}_t}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{m}_t}{\partial \boldsymbol{m}_{t-1}} \boldsymbol{M}_{t-1} = \boldsymbol{I} + \boldsymbol{V} \operatorname{diag}\big(\widetilde{h}'(\boldsymbol{m}_{t-1})\big) \boldsymbol{M}_{t-1} && (34)\\
\boldsymbol{M}_1 &\triangleq \boldsymbol{I}. && (35)
\end{aligned}$$

$$\frac{dg}{d\boldsymbol{V}} = \frac{\partial g}{\partial \boldsymbol{V}} + \sum_{t=1}^{T}\sum_{d}\frac{dg}{dm_t^{(d)}}\frac{dm_t^{(d)}}{d\boldsymbol{V}} \tag{36}$$

$$= \frac{\partial g}{\partial \boldsymbol{V}} + \sum_{t=1}^{T}\sum_{d}((\frac{\partial g}{\partial \boldsymbol{n}_t})^T\frac{\partial \boldsymbol{n}_t}{\partial \boldsymbol{m}_t} + (\frac{\partial g}{\partial \boldsymbol{m}_t})^T)^{(d)}\frac{dm_t^{(d)}}{d\boldsymbol{V}} \tag{37}$$

$$= 2\lambda \boldsymbol{V}\,\mathrm{diag}\left(\sum_{t=0}^{T-1}\left(\widetilde{h^2}(\boldsymbol{m}_t) - \widetilde{h}^2(\boldsymbol{m}_t)\right)\right) + \sum_{t=1}^{T}\sum_{d}\boldsymbol{r}_t^{(d)}\,\boldsymbol{M}_t^{(d)} \tag{38}$$

$$\boldsymbol{M}_t^{(d)} \triangleq \frac{d\boldsymbol{m}_t^{(d)}}{d\boldsymbol{V}} \tag{39}$$

$$= \frac{\partial \boldsymbol{m}_t^{(d)}}{\partial \boldsymbol{V}} + \sum_{d'}\frac{\partial \boldsymbol{m}_t^{(d)}}{\partial \boldsymbol{m}_{t-1}^{(d')}}\boldsymbol{M}_{t-1}^{(d')} \tag{40}$$

$$= \text{"Zero matrix except d'th row set to } \widetilde{h}^T(\boldsymbol{m}_{t-1})\text{"} + \sum_{d'}v_{d,d'}\widetilde{h}'(m_{t-1}^{(d')})\boldsymbol{M}_{t-1}^{(d')} \tag{41}$$

$$\boldsymbol{M}_1^{(d)} \triangleq \text{"Zero matrix except d'th row set to } \widetilde{h}^T(\boldsymbol{m}_0)\text{"}\,. \tag{42}$$

$$\frac{dg}{d\boldsymbol{U}} = \frac{\partial g}{\partial \boldsymbol{U}} + \sum_{t=1}^{T}\sum_{d}\frac{dg}{dm_t^{(d)}}\frac{dm_t^{(d)}}{d\boldsymbol{U}} \tag{43}$$

$$= \frac{\partial g}{\partial \boldsymbol{U}} + \sum_{t=1}^{T}\sum_{d}((\frac{\partial g}{\partial \boldsymbol{n}_t})^T\frac{\partial \boldsymbol{n}_t}{\partial \boldsymbol{m}_t} + (\frac{\partial g}{\partial \boldsymbol{m}_t})^T)^{(d)}\frac{dm_t^{(d)}}{d\boldsymbol{U}} \tag{44}$$

$$= 0 + \sum_{t=1}^{T}\sum_{d}\boldsymbol{r}_t^{(d)}\left(\boldsymbol{P}_t^{(d)}\right) \tag{45}$$

$$\boldsymbol{P}_t^{(d)} \triangleq \frac{d\boldsymbol{m}_t^{(d)}}{d\boldsymbol{U}} \tag{46}$$

$$= \frac{\partial \boldsymbol{m}_t^{(d)}}{\partial \boldsymbol{U}} + \sum_{d'}\frac{\partial \boldsymbol{m}_t^{(d)}}{\partial \boldsymbol{m}_{t-1}^{(d')}}\boldsymbol{P}_{t-1}^{(d')} \tag{47}$$

$$= \text{"Zero matrix except d'th row set to } \boldsymbol{x}_t^T\text{"} + \sum_{d'}v_{d,d'}\widetilde{h}'(m_{t-1}^{(d')})\boldsymbol{P}_{t-1}^{(d')} \tag{48}$$

$$\boldsymbol{P}_1^{(d)} \triangleq \text{"Zero matrix except d'th row set to } \boldsymbol{x}_1^T\text{"}\,. \tag{49}$$

$$
\begin{aligned}
(\frac{dg}{d\boldsymbol{m}_0})^T &= (\frac{\partial g}{\partial \boldsymbol{m}_0})^T + \sum_{t=1}^{T} (\frac{dg}{d\boldsymbol{m}_t})^T \frac{d\boldsymbol{m}_t}{d\boldsymbol{m}_0} && (50)\\
&= (\frac{\partial g}{\partial \boldsymbol{m}_0})^T + \sum_{t=1}^{T} ((\frac{\partial g}{\partial \boldsymbol{n}_t})^T \frac{\partial \boldsymbol{n}_t}{\partial \boldsymbol{m}_t} + (\frac{\partial g}{\partial \boldsymbol{m}_t})^T) \frac{d\boldsymbol{m}_t}{d\boldsymbol{m}_0} && (51)\\
&= \lambda\Big(\big(\widetilde{h^2}'(\boldsymbol{m}_0) - 2\widetilde{h}'(\boldsymbol{m}_0) \odot \widetilde{h}(\boldsymbol{m}_0)\big)^T \odot \big(\mathbf{1}^T(\boldsymbol{V} \odot \boldsymbol{V})\big) && (52)\\
&\qquad +2H\sigma^2 (\widetilde{h}'(\boldsymbol{m}_0) \odot \widetilde{h}(\boldsymbol{m}_0))^T \Big) && (53)\\
&\qquad + \sum_{t=1}^{T} \boldsymbol{r}_t\left(\boldsymbol{Q}_t\right) && (54)\\
\boldsymbol{Q}_t &\triangleq \frac{d\boldsymbol{m}_t}{d\boldsymbol{m}_0} = \frac{\partial \boldsymbol{m}_t}{\partial \boldsymbol{m}_{t-1}} \boldsymbol{Q}_{t-1} = \boldsymbol{V}\, \mathrm{diag}\left(\widetilde{h}'(\boldsymbol{m}_{t-1})\right) \boldsymbol{Q}_{t-1} && (55)\\
\boldsymbol{Q}_0 &\triangleq \boldsymbol{I}\,. && (56)
\end{aligned}
$$

C Bounding Linearization Error

Proposition 1 *Assume* $n \geq 5$, $c_f \geq \frac{1}{2\pi} \sum_{j,k} \| \frac{d^2 f}{dx_j\, dx_k} \|_{\frac{n}{2}}$ *and* $\rho^2 c_f \frac{1}{\sigma^2} \leq \epsilon$. *Then if follows that* $\forall \boldsymbol{x}\,;\, \|\boldsymbol{x} - \boldsymbol{x}_0\| \leq \rho \Rightarrow |g(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla g(\boldsymbol{x}_0) - g(\boldsymbol{x})| \leq \epsilon$.

Proof First we claim that $\frac{1}{2}\Lambda_g \leq \frac{1}{2\pi\sigma^2} \sum_{j,k} \| \frac{d^2 f}{dx_j\, dx_k} \|_{\frac{n}{2}}$. We prove this claim as below,

$$
\begin{aligned}
\frac{1}{2}\Lambda_g &\leq \max_{\boldsymbol{x}} \|\nabla^2 g(\boldsymbol{x})\|_F && (57)\\
&\leq \max_{\boldsymbol{x}} \sum_{j,k} |\frac{d^2 g}{dx_j\, dx_k}(\boldsymbol{x})| && (58)\\
&\leq \sum_{j,k} \max_{\boldsymbol{x}} |\frac{d^2 g}{dx_j\, dx_k}(\boldsymbol{x})| && (59)\\
&= \sum_{j,k} \|\frac{d^2 g}{dx_j\, dx_k}\|_\infty && (60)\\
&= \sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k} \star k_\sigma\|_\infty && (61)\\
&\leq \sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{p}{p-1}}\, \|k_\sigma\|_p && (62)\\
&\leq \Big(\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{p}{p-1}}\Big)\Big(\int_{\mathcal{X}} k_\sigma^p(\boldsymbol{x})\, d\boldsymbol{x}\Big)^{\frac{1}{p}} && (63)\\
&\leq \Big(\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{p}{p-1}}\Big)\Big(\frac{(2\pi)^{(1-p)}\sigma^{2(1-p)}}{p}\Big)^{\frac{n}{4p}} && (64)\\
&= \Big(\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{p}{p-1}}\Big)\Big(\frac{(2\pi)^{(1-p)}}{p}\Big)^{\frac{n}{4p}} \sigma^{\frac{n(1-p)}{2p}}\,, && (65)
\end{aligned}
$$

where (62) is due to Young's convolution inequality and holds for any $p \geq 1$. In particular, when $n \geq 5$, by setting $p = \frac{n}{n-4}$, we obtain

$$
\begin{aligned}
\frac{1}{2}\Lambda_g &\leq \Big(\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{p}{p-1}}\Big)\Big(\frac{(2\pi)^{(1-p)}}{p}\Big)^{\frac{n}{4p}} \sigma^{\frac{n(1-p)}{2p}} && (66)\\
\frac{1}{2}\Lambda_g &= \Big(\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{n}{2}}\Big)\frac{1}{2\pi\sigma^2}\Big(1-\frac{4}{n}\Big)^{\frac{n}{4}-1} && (67)\\
\frac{1}{2}\Lambda_g &\leq \Big(\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{n}{2}}\Big)\frac{1}{2\pi\sigma^2}\,. && (68)
\end{aligned}
$$

This proves our earlier claim that $\frac{1}{2}\Lambda_g \leq \frac{1}{2\pi\sigma^2}\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{n}{2}}$. Combining this with the assumption $\frac{1}{2\pi}\sum_{j,k} \|\frac{d^2 f}{dx_j\, dx_k}\|_{\frac{n}{2}} \leq c_f$, it follows that $\frac{1}{2}\Lambda_g \leq c_f \frac{1}{\sigma^2}$, which implies $\frac{1}{2}\rho^2\Lambda_g \leq \rho^2 c_f \frac{1}{\sigma^2}$. The latter combined with the assumption $\rho^2 c_f \frac{1}{\sigma^2} \leq \epsilon$ yields $\frac{1}{2}\rho^2\Lambda_g \leq \epsilon$. Combining this with the Taylor's remainder theorem $|g(\boldsymbol{x}_0)+(\boldsymbol{x}-\boldsymbol{x}_0)^T\nabla g(\boldsymbol{x}_0)-g(\boldsymbol{x})| \leq \frac{1}{2}\rho^2\Lambda_g$ gives $|g(\boldsymbol{x}_0)+(\boldsymbol{x}-\boldsymbol{x}_0)^T\nabla g(\boldsymbol{x}_0)-g(\boldsymbol{x})| \leq \epsilon$.

□