How nonconvex is it? Exploring neural network geometry through Dynamic String Sampling

C. Daniel Freeman[1, *] and Joan Bruna[2]

[1]*Department of Physics, University of California, Berkeley, CA 94720, USA*
[2]*Department of Statistics, University of California, Berkeley, CA 94720, USA*

(Dated: July 15, 2016)

The natural tool for understanding neural network training is *nonconvex optimization*. In practice, this means that, for some given learning task, a loss function is optimized via some flavor of gradient descent. While this strategy has led to great success, it is surprising that not much is known about the geometry of the loss function itself. In this manuscript, we examine the geometry of these loss functions for several learning tasks (QUAD, MNIST). In so doing, we provide both a quantitative handle on problem nonconvexity as well as an algorithm for estimating that nonconvexity (Dynamic String Sampling). Operationally, given two models which have nearly the same loss L_0 on test data, our algorithm searches for continuous paths in the space of model parameters which do not exceed L_0. Whether such a path can be found defines a notion of connectedness, and thus nonconvexity.

I. INTRODUCTION

II. QUANTIFYING NONCONVEXITY

A. Definitions

For a model with network parameters θ_i, and a learning problem with sample space X, the fundamental object of study is the loss function, $L(X, \theta_i)$. In practice, one only has access to an estimate of the loss function over some restricted subset, χ_i, of the sample space: $E(L(X, \theta_i), \chi_i)$. Unless otherwise stated, the loss functions computed throughout are assumed to be on a restricted test set not used during training.

A key ingredient of the algorithm is the use of an *interpolated model*. For two given models, θ_1 and θ_2, we defined the interpolated model with parameter t as follows:

$$\Theta(\theta_1, \theta_2, t) := \theta_1(1-t) + \theta_2 t \tag{1}$$

Thus, the interpolated model parameters—i.e., weights and biases—are simply linearly interpolated between two given models.

Additionally, the algorithm requires an estimate of the interpolated loss curve:

$$\gamma(\theta_1, \theta_2) := L(X, \Theta(\theta_1, \theta_2, t)), t \in [0, 1] \tag{2}$$

or, an estimate of the loss on those models which are linear interpolations sitting between θ_1 and θ_2. More specifically, we seek efficient estimates of the location of the maxima, $t^* := \frac{d\gamma(\theta_1,\theta_2,t)}{dt}\Big|_{t^*} = 0, \frac{d^2\gamma(\theta_1,\theta_2,t)}{dt^2}\Big|_{t^*} < 0$. While in principle, the interpolated loss curve could have rich structure, in practice it is generally fairly smooth, thus straightforward hill climbing algorithms can efficiently locate these points.

Finally, for a pair of models (θ_1, θ_2), it will be convenient to define the maximum interpolated error:

$$\Gamma(\theta_1, \theta_2) := \min_{\Theta^*(\theta_1,\theta_2)} \max \mathrm{L}(\mathrm{X}, \theta_\mathrm{i})\Big|_{\theta_\mathrm{i} \in \Theta^*} \tag{3}$$

where $\Theta^*(\theta_1, \theta_2)$ is *any* continuous path in the space of weights connecting θ_1 and θ_2. Thus, $\Gamma(\theta_1, \theta_2)$ represents the minimum possible maximum loss achieved by those paths in the space of weights connecting θ_1 and θ_2. More intuitively, if $\Gamma(\theta_1, \theta_2) \leq \max(\mathrm{L}(\mathrm{X}, \theta_1), \mathrm{L}(\mathrm{X}, \theta_2))$, then the models are "connected"—there exists a continuous path in the space of models with total loss never exceeding the maximum loss achieved by θ_1 or θ_2.

B. The Algorithm

1. Train two models θ_i and θ_j to a threshold loss value, L_0.
2. Determine the location of the global maxima, t^*, on the interpolated loss curve $\gamma(\theta_i, \theta_j)$.
3. Perform gradient descent on the interpolated model $\Theta(\theta_i, \theta_j, t^*) := \theta_{i,j}$ until it is below αL_0 for some $\alpha \in [0, 1]$.
4. Calculate the maxima of the interpolated losses $\gamma(\theta_i, \theta_{i,j})$ and $\gamma(\theta_{i,j}, \theta_j)$. If these maxima are below L_0, then stop recursing on this branch and proceed to remaining branches(see 5). If not, proceed to step 5.
5. For those pairs, θ_a, θ_b from step 4 for which the maxima exceeds L_0, start a new branch by returning to step 2 and making the replacement $i- > a$ and $j- > b$. If depth exceeds d, stop (see below).

We provide a cartoon of the algorithm in Fig. 1. If the algorithm succeeds, then the output of the algorithm is a sequence of models, θ_i such that the pairwise interpolated loss curve between each in a sequence will be less than the threshold L_0. Thus, the algorithm outputs a continuous path in parameter space connecting the original two models such that everywhere along the path, the

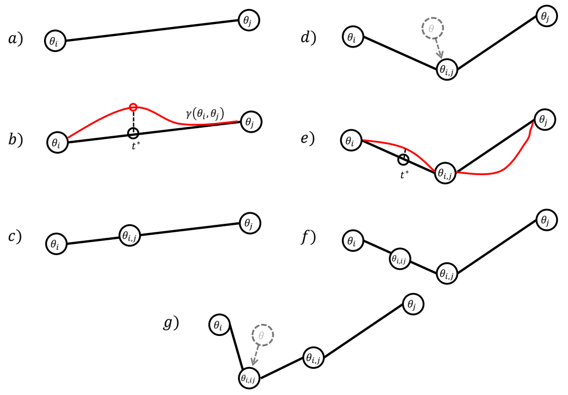

FIG. 1. A cartoon of the algorithm. *a*) : The initial two models with approximately the same loss, L_0. *b*) : The interpolated loss curve, in red, and its global maximum, occuring at $t = t^*$. *c*) : The interpolated model $\Theta(\theta_i, \theta_j, t^*)$ is added and labeled $\theta_{i,j}$. *d*) : Stochastic gradient descent is performed on the interpolated model until its loss is below αL_0. *e*) : New interpolated loss curves are calculated between the models, pairwise on a chain. *f*) : As in step *c*), a new model is inserted at the maxima of the interpolated loss curve between θ_i and $\theta_{i,j}$. *g*) : As in step *d*), gradient descent is performed until the model has low enough loss.

total loss is less than or equal to the loss of the original models.

As written, if a path does *not* exist, then the algorithm will clearly not converge. Thus, on top of the parameter α, discussed below, the algorithm has an additional free parameter in the *depth* chosen to explore. For convenience, we define the string of models produced by the algorithm at depth d with parameter α to be the *interpolated string*, $S(\theta_1, \theta_2, \alpha, d)$. These are precisely those models recursively generated by the algorithm in step 3. Further, these models are naturally ordered along a path, starting from θ_1 and terminating on θ_2, as indicated in Fig. 1.

Finally, to use this as a tool to diagnose convexity, we define the *maximum interpolated error at depth d and tolerance* α:

$$\tilde{\Gamma}(\theta_1, \theta_2, d, \alpha) := \max \gamma(\theta_\mathrm{i}, \theta_\mathrm{j}, \mathrm{t}), \mathrm{i},\ \mathrm{j\ neighbors\ in\ S}(\theta_1, \theta_2, \alpha, \mathrm{d}) \tag{4}$$

where by "neighbors in $S(\theta_1, \theta_2, \alpha, d)$", we only mean that the models are immediately adjacent on the interpolating string. This quantity upper bounds the true maximum interpolated error, i.e. (3).

In summary: the algorithm recursively produces and trains new models lying on a continuous path in the space of model parameters, i.e. a string. Training via gradient descent biases the path towards valleys on the loss surface, thus encouraging the loss along this path to be low. In practice, the parameter α is chosen to be less than 1 to aid convergence. We provide numerical and theoretical evidence for this choice in section SECTIONGOHERE.

III. NUMERICAL EXPERIMENTS

For our numerical experiments, we aimed to extract qualitative features of both small, toy networks, as well as of larger workhorse networks suitable for use on real world tasks (e.g. MNIST). At its core, the maximum interpolated error (i.e., (3)) is a measure of problem nonconvexity—or, more precisely, of the nonconvexity of the loss surface of a given architecture on a particular learning problem.

A. Polynomial Regression

Polynomial function regression is a task for which small neural networks can achieve extremely high accuracy. For our numerical experiments, we studied a 1-4-4-1 fully connected multilayer perceptron style architecture with RELU activation and RMSProp optimization. For ease-of-analysis, we restricted the family of polynomials to be strictly contained in the interval $x \in [0, 1]$ and $f(x) \in [0, 1]$.

Discussion of different Loss functions

etc.

B. MNIST

IV. DISCUSSION