ENTROPY-SGD: BIASING GRADIENT DESCENT INTO WIDE VALLEYS

Pratik Chaudhari[1], Anna Choromanska[2], Stefano Soatto[1], Yann LeCun[2,3]

[1] Computer Science, University of California, Los Angeles

[2] Courant Institute of Mathematical Sciences, New York University

[3] Facebook AI Research

Email: pratikac@ucla.edu, achoroma@cims.nyu.edu, soatto@cs.ucla.edu, yann@cs.nyu.edu

ABSTRACT

This paper proposes a new optimization algorithm called Entropy-SGD for training deep neural networks that is motivated by the local geometry of the energy landscape at solutions found by gradient descent. Local extrema with low generalization error have a large proportion of almost-zero eigenvalues in the Hessian with very few positive or negative eigenvalues. We leverage upon this observation to construct a local entropy based objective that favors well-generalizable solutions lying in the flat regions of the energy landscape, while avoiding poorly-generalizable solutions located in the sharp valleys. Our algorithm resembles two nested loops of SGD, where we use Langevin dynamics to compute the gradient of local entropy at each update of the weights. We prove that incorporating local entropy into the objective function results in a smoother energy landscape and use uniform stability to show improved generalization bounds over SGD. Our experiments on competitive baselines demonstrate that Entropy-SGD leads to improved generalization and has the potential to accelerate training.

1 INTRODUCTION

This paper focuses on developing new optimization tools for deep learning that are tailored to exploit the local geometric properties of the objective function. Consider the histogram in Fig. 1, showing the eigenspectrum of the Hessian at a local minimum discovered by Adam (Kingma & Ba, 2014) for a convolutional neural network on MNIST (LeCun et al., 1998) with about $47,000$ weights (cf. Sec. 5.1). It is evident that:

(i) a large number of directions ($\approx 94\%$) have near-zero eigenvalues (magnitude less than 10^{-4}),

(ii) positive eigenvalues (right inset) have a long tail with the largest one being almost 40,

(iii) negative eigenvalues (left inset), which are directions of descent that the optimizer missed, have a much faster decay (the largest negative eigenvalue is only -0.4).

Interestingly, this trend is not unique to this particular network. Rather, its qualitative properties are shared across a variety of network architectures, sizes, datasets or optimization algorithms (refer to Sec. 5 for more experiments). Local minima that generalize well and are discovered by gradient descent lie in "wide valleys" of the energy landscape, that look more like the bottom of a bucket than sharp isolated minima like the holes on a golf-course. Almost-flat regions of the energy landscape are robust to data perturbations, noise in the activations as well as perturbations of the parameters — all of which are widely-used techniques to achieve good generalization. This suggests that wide valleys should result in better generalization and, indeed, standard optimization algorithms in deep learning — without being explicitly tailored to do so — seem to discover exactly that.

Based on this understanding of how the local geometry looks at the end of optimization, can we modify SGD to actively seek such regions? We show that flat regions in the energy landscape correspond to regions where the log-partition function is small, if computed locally on the Gibbs

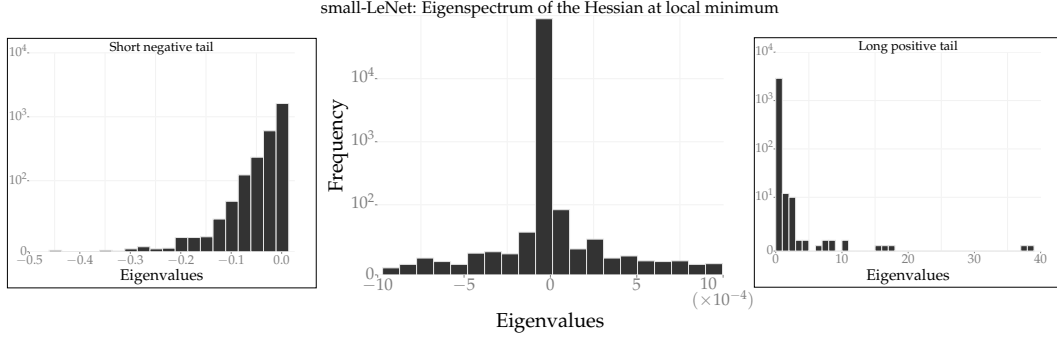small-LeNet: Eigenspectrum of the Hessian at local minimum

Figure 1: Eigenspectrum of the Hessian at a local minimum of a CNN on MNIST (two independent runs).
Remark: The central plot shows the eigenvalues in a small neighborhood of zero whereas the left and right insets show the entire tails of the eigenspectrum.

distribution corresponding to the optimization problem. Roughly, for parameters denoted by $x \in \mathbb{R}^n$, we modify a loss function $f(x)$ to be

$$f(x) - \log \int_{x' \in \mathbb{R}^n} \exp\left(-f(x') - \frac{\gamma}{2} \|x - x'\|_2^2\right) \, dx';$$

where the second term is a log-partition function which is seen as a "local free energy" and measures both the depth of a valley at a location x and its flatness through the entropy of $f(x')$. The Entropy-SGD algorithm presented in this paper employs a few iterations of stochastic gradient Langevin dynamics (SGLD) to approximate the second term before each update of the parameters using the back-propagated gradient. At a conceptual level, this algorithm resembles running two nested loops of SGD. We show that the above modified loss function results in a smoother energy landscape defined by the hyper-parameter γ which we can think of as a "scope" that seeks out valleys of specific widths. Actively biasing towards wide valleys aids generalization, in fact, we can optimize solely the free energy term to obtain similar generalization error as SGD on the original loss function. Our experiments in Sec. 5 on MNIST and CIFAR-10 (Krizhevsky, 2009) show that Entropy-SGD scales to deep networks used in practice and in terms of the number of updates to the parameters, trains much quicker than SGD.

2 RELATED WORK

Our above observation about the eigenspectrum of Hessian (further discussed in Sec. 5) is similar to results on a perceptron model in Dauphin et al. (2014) where the authors connect the loss function of a deep network to a high-dimensional Gaussian random field. They also relate to earlier studies such as Baldi & Hornik (1989); Fyodorov & Williams (2007); Bray & Dean (2007) which show that critical points with high training error are exponentially likely to be saddle points with many negative directions and all local minima are likely to have error that is very close to that of the global minimum. The authors also argue that convergence of gradient descent is affected by the proliferation of saddle points surrounded by high error plateaus — as opposed to multiple local minima. One can also see this via an application of Kramer's law: the time spent by diffusion is inversely proportional to the smallest negative eigenvalue of the Hessian at a saddle point (Bovier & den Hollander, 2006).

The existence of multiple, almost equivalent, local minima in deep networks has been predicted using a wide variety of theoretical analyses and empirical observations, e.g., papers such as Choromanska et al. (2015a;b); Chaudhari & Soatto (2015) that build upon results from statistical physics as also others such as Haeffele & Vidal (2015) and Janzamin et al. (2015) that obtain similar results for matrix and tensor factorization problems. Although assumptions in these works are somewhat unrealistic in the context of deep networks used in practice, similar results are also true for linear networks which afford a more thorough analytical treatment (Saxe et al., 2014). For instance, Soudry & Carmon (2016) show that with mild over-parameterization and dropout-like noise, training error for a neural network with one hidden layer and piece-wise linear activation is zero at every local minimum. All these results suggest that the energy landscape of deep neural networks should be easy to optimize and they more or less hold in practice — it is easy to optimize a prototypical deep network to near-zero loss *on the training set* (Hardt et al., 2015; Goodfellow & Vinyals, 2015).

Obtaining good *generalization* error, however, is challenging: complex architectures are sensitive to initial conditions and learning rates (Sutskever et al., 2013) and even linear networks (Kawaguchi, 2016) may have degenerate and hard to escape saddle points (Ge et al., 2015; Anandkumar & Ge, 2016). Techniques such as adaptive (Duchi et al., 2011) and annealed learning rates, momentum (Tieleman & Hinton, 2012), as well as architectural modifications like dropout (Srivastava et al., 2014), batch-normalization (Ioffe & Szegedy, 2015; Cooijmans et al., 2016), weight scaling (Salimans & Kingma, 2016) etc. are different ways of tackling this issue by making the underlying landscape more amenable to first-order algorithms. However, the training process often requires a combination of such techniques and it is unclear beforehand to what extent each one of them helps.

Such a paradox between a seemingly benign energy landscape and delicate training algorithms is also seen in problems such as random k-SAT (Achlioptas & Ricci-Tersenghi, 2006) and compressed sensing (Krzakala et al., 2012). Introducing non-local terms in classical inference techniques for these problems has led to state-of-the-art algorithms (Braunstein et al., 2005; Maneva et al., 2007). Closer to deep learning, although motivated from similar ideas, the authors in Baldassi et al. (2015; 2016a) show that, both theoretically and practically, ensembles of perceptrons can be trained in parallel by penalizing each individual model to be close to the ensemble average. Using a novel out-of-equilibrium analysis, the authors show that such an ensemble recovers dense clusters in the solution space. Such dense clusters generalize better than isolated local minima that may lie close to the global minimum. A related work in this direction is EASGD (Zhang et al., 2015) which trains multiple deep networks in parallel and encourages individual models to explore by modulating their distance to the ensemble average; this is a special case of robust ensembles.

An ensemble training procedure thus provides improved generalization but it incurs a large complexity in terms of hardware and communication across workers. In this paper, we construct an optimization algorithm for a general deep network that can exploit the same phenomenon of out-of-equilibrium dense clusters but does not require distributed training. The enabling technique in our case is that we approximate the local entropy using Langevin dynamics. Baldassi et al. (2016a) instead employ belief propagation to do so; this can only be done for simpler perceptron models considered there.

The authors in Hochreiter & Schmidhuber (1997b) introduce hard constraints on the width of local minima and training loss and show using the Gibbs formalism (Haussler et al., 1997) that this leads to improved generalization. However, as they mention, the effect of hyper-parameters for these constraints is intricately tied together and they are difficult to choose even for small problems. Their formulation depends on Hessian-vector products. It is unclear how this method could be scaled to modern deep networks. Motivated by the same final goal, viz., flat local minima, our approach eliminates these extraneous and difficult to control hyper-parameters and instead results in a scalable algorithm. Also recent work by Keskar et al. (2016) is relevant in the context of the main theme of our paper. The authors estimate the loss in a neighborhood of the weights to argue that small batch size in SGD (i.e., larger gradient noise) generalizes better than large mini-batches and also results in significantly flatter minima.

In the sense that local entropy results in a smoother energy landscape (cf. Sec. 4.4), it resembles continuation methods (Allgower & Georg, 2012) that convolve the loss function to solve sequentially refined optimization problems. These techniques however require a special class of loss functions where successively smaller convolutions result in minimizers that lie close to each other or closed-form expressions of smoothened loss functions (Hazan et al., 2016; Mobahi & Fisher III, 2015); these may not be possible for deep networks with isolated minima and highly-nonlinear loss functions. Our modified objective instead weighs each location by the local entropy in its neighborhood and does not require stringent assumptions on the objective, it is also applicable to full-scale deep networks.

3   LOCAL ENTROPY

We first provide a simple intuition for the concept of local entropy of an energy landscape. Consider a cartoon energy landscape in Fig. 2 where the x-axis denotes the configuration space of the parameters. We have constructed two local minima: a shallower although wider one at x_{robust} and a very sharp global minimum at $x_{\text{non-robust}}$. Under a Bayesian prior on the parameters, say a Gaussian of a fixed variance at locations x_{robust} and $x_{\text{non-robust}}$ respectively, the wider local minimum has a higher log-likelihood than the sharp valley on the right. In discrete optimization problems, such wide local

minima are dense clusters of solutions in the parameter space where nearby parameters have similar energy, say, the number of clauses violated in k-SAT (Sipser, 2006; Achlioptas & Ricci-Tersenghi, 2006). On the other hand, sharp local minima are isolated solutions that are far away from other solutions.

The above discussion suggests that parameters that lie in wider local minima like x_{robust}, which may possibly have a higher loss than the global minimum, should generalize better than the ones that are simply at the global minimum. This motivates us to use a very well-known quantity from thermodynamics known as "free energy" (also sometimes called "free entropy"):

free energy = internal energy − temperature × entropy.

Roughly, internal energy is the average energy of all configurations while entropy counts the number of distinct configurations in a physical system at a given temperature. In a neighborhood of x_{robost}, the entropy term term is small while internal energy is large. Conversely, near $x_{\text{non-robust}}$, internal energy is low while entropy is high. The "negative free entropy" computed locally near a particular configuration thus provides a way of picking large flat regions of landscape over sharp, narrow valleys in spite of the latter possibly having a lower loss. Quite conveniently, free energy is also the log-partition function of a system.

Figure 2: Local entropy concentrates on dense clusters in the energy landscape.

Formally, for a parameter vector $x \in \mathbb{R}^n$, consider a Gibbs distribution corresponding to a given energy landscape $f(x)$:

$$\mathrm{P}(x; \beta) = Z_\beta^{-1} \exp\left(-\beta \, f(x)\right); \tag{1}$$

where β is known as the inverse temperature and Z_β is a normalizing constant. As $\beta \to \infty$, the probability distribution above concentrates on the global minimum of $f(x)$ given as

$$x^* = \underset{x}{\mathrm{argmin}} \; f(x), \tag{2}$$

which establishes the link between the Gibbs distribution and a generic optimization problem (2). We would instead like the probability distribution — and therefore the underlying optimization problem — to focus on flat regions such as x_{robust}. With this in mind, let us construct a modified Gibbs distribution:

$$\mathrm{P}(x'; x, \beta, \gamma) = Z_{x,\beta,\gamma}^{-1} \exp\left(-\beta \, f(x') - \beta \, \frac{\gamma}{2} \, \|x - x'\|_2^2\right). \tag{3}$$

The distribution in (3) is a function of a dummy variable x' and is parameterized by the original location x. The parameter γ biases the modified distribution (3) towards x; a large γ results in a $P(x'; x, \beta, \gamma)$ with all its mass near x irrespective of the energy landscape of $f(x')$. For small values of γ, the term $f(x')$ in the exponent dominates and the modified distribution is similar to the original Gibbs distribution in (1). We will set the inverse temperature β to 1 because γ affords us similar control on the Gibbs distribution.

Definition 1 (Local entropy). The local free energy of the Gibbs distribution in (1), colloquially called "local entropy" in the sequel and denoted by $F(x, \gamma)$, is defined as the log-partition function of modified Gibbs distribution (3), i.e.,

$$F(x, \gamma) = \log Z_{x,1,\gamma}$$
$$= \log \int_{x'} \exp\left(-f(x') - \frac{\gamma}{2} \, \|x - x'\|_2^2\right) dx'. \tag{4}$$

The parameter γ is used to focus the modified Gibbs distribution upon a local neighborhood of x and we call it a "scope" henceforth. Fig. 2 shows the negative local entropy $-F(x, \gamma)$ for the original energy landscape. Note that $-F(x, \gamma)$ has a global minimum near x_{robust} which is exactly what we want; indeed, x_{robust} has a higher local entropy than $x_{\text{non-robust}}$.

Let us also note that the quantity we have defined as local entropy in Def. 1 is different from classical entropy which counts the number of likely configurations under a given distribution. This is given
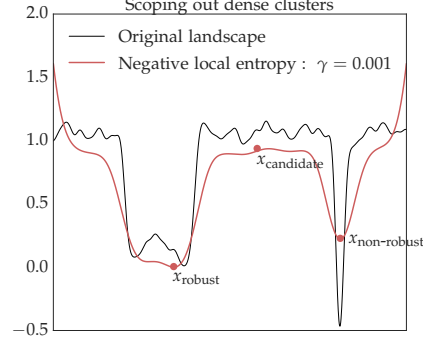
by $- \int_{x'} \log \text{P}(x'; x, \gamma) \, d\text{P}(x'; x, \gamma)$ for the Gibbs distribution in (3). Minimizing classical entropy however does not differentiate between flat regions that have very high loss versus dense regions that lie deeper in the energy landscape. For instance, in Fig. 2, classical entropy is smallest in the neighborhood of $x_{\text{candidate}}$ which is a large region with very high loss on the training dataset and is unlikely to generalize well.

4 Entropy-guided SGD

We now present the Entropy-SGD algorithm, a variant of SGD that is motivated from local entropy. Simply speaking, we modify the loss function of a deep network to include the negative local entropy from Sec. 3. The gradient of the modified loss can then be computed using Langevin dynamics. The Entropy-SGD algorithm (cf. Alg. 1) thus has a strong flavor of "SGD-inside-SGD": the outer SGD updates the parameters, while an inner SGD estimates the gradient of local entropy at each iteration.

Consider a typical classification setting, let $x \in \mathbb{R}^n$ be the weights of a deep neural network and $\xi_k \in \Xi$ be samples from a dataset Ξ of size N. Let $f(x; \xi_k)$ be the loss function, e.g., cross-entropy of the classifier on a sample ξ_k. The original optimization problem is:

$$x^* = \underset{x}{\text{argmin}} \; \frac{1}{N} \sum_{k=1}^{N} f(x; \, \xi_k); \qquad (5)$$

where the objective $f(x, \xi_k)$ is typically, a non-convex function in both the weights x and the sample ξ_k. The Entropy-SGD algorithm instead solves the problem

$$x^*_{\text{Entropy-SGD}} = \underset{x}{\text{argmin}} \; \rho \left(\frac{1}{N} \sum_{k=1}^{N} f(x; \, \xi_k) \right) - F(x, \gamma; \, \Xi); \qquad (6)$$

where we have made the dependence of local entropy $F(x, \gamma)$ on the dataset Ξ explicit. The objective in (6) thus scales the original loss by a hyper-parameter ρ and adds the negative local entropy term $F(x, \gamma, \Xi)$ introduced in Sec. 3. The parameter ρ allows us to control the contribution of the original back-propagated gradient in Entropy-SGD. Sec. 4.2 and Sec. 4.4 below discuss the effect of ρ and the scoping parameter γ.

4.1 Computing the gradient of local entropy

The gradient of local entropy over a randomly sampled mini-batch of m samples denoted by $\xi_{\ell_i} \in \Xi^\ell$ for $i \le m$ is easy to derive and is given by

$$\nabla_x F\left(x, \gamma; \, \Xi^\ell\right) = -\gamma \left(x - \left\langle x'; \, \Xi^\ell \right\rangle\right); \qquad (7)$$

where the notation $\langle \cdot \rangle$ denotes an expectation of its arguments (we have again made the dependence on the data explicit) over a Gibbs distribution of the original optimization problem modified to focus on the neighborhood of x; this is given by

$$\text{P}(x'; x, \gamma) \propto \exp\left[-\left(\frac{1}{m} \sum_{i=1}^{m} f\left(x; \, \xi_{\ell_i}\right) \right) - \frac{\gamma}{2} \, \|x - x'\|_2^2 \right]. \qquad (8)$$

Computationally, the gradient in (7) involves estimating $\langle x'; \Xi^\ell \rangle$ with the current weights fixed to x. This is an expectation over a Gibbs distribution and is hard to compute. We can however approximate it using Markov chain Monte-Carlo (MCMC) techniques. In this paper, we use stochastic gradient Langevin dynamics (SGLD) (Welling & Teh, 2011) that is an MCMC algorithm for drawing samples from a Bayesian posterior and scales to large datasets using mini-batch updates. Please see Sec. A in the Supplement for a brief overview of SGLD. For our application, as lines 3-6 of Alg. 1 show SGLD resembles a few iterations of SGD with additive gradient noise.

The gradient of the first term in (6) is computed using back-propagation and the final gradient is

$$\rho \left(\frac{1}{m} \sum_{i=1}^{m} \nabla f(x; \, \xi_{k_i}) \right) + \gamma \left(x - \left\langle x'; \, \Xi^\ell \right\rangle\right). \qquad (9)$$

Note that we use different mini-batches (denoted by k and ℓ respectively) to compute the back-propagated gradient and gradient of the local entropy. From our experiments, we observe that using the same mini-batches for the gradients performs marginally worse in terms of convergence speed. This is explained by the fact that local entropy is really a log-partition function and depends upon the entire dataset.

4.2 INTUITION BEHIND LOCAL ENTROPY

Let us give some intuition on how local entropy works using the expression for the gradient: the term $\langle x'; \cdot \rangle$ is the average over a locally focused Gibbs distribution and for two local minima in the neighborhood of x roughly equivalent in loss, this term points towards the wider one because $\langle x'; \cdot \rangle$ is closer to it. This results in a net gradient that takes SGD towards wider valleys. Moreover, if we unroll the SGLD steps used to compute $(x - \langle x'; \cdot \rangle)$ (cf. line 5 in Alg. 1), it resembles one large step in the direction of the (noisy) average gradient around the current weights x and Entropy-SGD thus looks similar to averaged SGD in the literature (Polyak & Juditsky, 1992; Bottou, 2012). These two phenomena intuitively explain the improved generalization performance of Entropy-SGD.

The local entropy term in the objective results in a smoother energy landscape than the original loss function. From our experience (cf. experiments in Sec. 5.2), we can also set ρ to zero, i.e., consider the modified loss function for Entropy-SGD to be simply $-F(x, \gamma)$ and completely eliminate $\nabla f(x)$ from the outer loop of Entropy-SGD. This gives similar generalization error as that of SGD for simpler architectures such as a convolutional neural network on MNIST. However, for more complex datasets and networks, the smooth energy landscape generated by $-F(x, \gamma)$ might have a different local minimum as compared to the original loss $f(x)$. We found that in order to fine-tune larger networks using the objective in (6) with a small ρ performs slightly better than setting ρ to zero.

4.3 ALGORITHM AND IMPLEMENTATION DETAILS

Alg. 1 provides the pseudo-code for one iteration of the Entropy-SGD algorithm. At each iteration, lines 3-6 perform L iterations of Langevin dynamics to estimate $\mu = \langle x'; \Xi^\ell \rangle$. The weights x are updated with the modified gradient on line 9.

We set the number of SGLD iterations L to 5-20 depending upon the complexity of the dataset, e.g., for experiments on convolutional networks on MNIST and CIFAR-10, we use $L = 20$ while for experiments on a fully-connected network on MNIST $L = 5$ works well (cf. Sec. 5). We perform exponential averaging of the Langevin iterates x' to estimate μ with a parameter $\alpha = 0.9$ so as to put more weight on later samples. In order to quickly transition to the Langevin dynamics phase of SGLD and ensure the we are sampling from the posterior even for small values of L, we set the learning rate η' on line 5 to be small factor of the outer SGD's learning rate: $\eta' = 0.1\,\eta$ (Welling & Teh, 2011).

Algorithm 1: Entropy-SGD algorithm

Input	:	current weights x, Langevin iterations L
Hyper-parameters	:	Scope γ, Data weight ρ, Learning rate η

// SGLD iterations;

1   $x', \mu \leftarrow x$;

2   **for** $\ell \leq L$ **do**

3      $\Xi^\ell \leftarrow$ sample mini-batch;

4      $dx' \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_{x'} f\left(x'; \xi_{\ell_i}\right) - \gamma \left(x - x'\right)$;

5      $x' \leftarrow x' - \eta'\, dx' + 2\sqrt{\eta'}\, \mathrm{N}(0, \mathrm{I})$;

6      $\mu \leftarrow (1 - \alpha)\mu + \alpha\, x'$;

// Update weights;

7   $\Xi^k \leftarrow$ sample mini-batch;

8   $dx \leftarrow \frac{1}{m} \sum_{i=1}^m \nabla_x f\left(x; \xi_{k_i}\right)$;

9   $x \leftarrow x - \eta \left(\rho\, dx + \gamma \left(x - \mu\right)\right)$

Let us note that although we have written Alg. 1 in the classical SGD setup, we can easily modify it to include techniques such as momentum and gradient pre-conditioning (Duchi et al., 2011) by changing lines 5 and 9. In our experiments, we have used both SGD with Nesterov's momentum (Sutskever et al., 2013) and Adam for outer and inner loops with similar qualitative results.

4.4 THEORETICAL PROPERTIES

Our main result in this section is to show that Entropy-SGD results in a smoother loss function and obtains better generalization error than the original objective (5) if trained for the same number of iterations. Proofs are deferred to the Appendix (see Sec. B).

With some overload of notation, we assume that the original loss $f(x)$ is β-smooth, i.e., for all $x, y \in \mathbb{R}^n$, we have $\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|$. We additionally assume for the purpose of analysis that no eigenvalue of the Hessian $\nabla^2 f(x)$ lies in the set $[-2\gamma - c, c]$ for some small $c > 0$.

Lemma 2. *The objective $F(x, \gamma; \Xi)$ in (6) is $\frac{\alpha}{1+\gamma^{-1} c}$-Lipschitz and $\frac{\beta}{1+\gamma^{-1} c}$-smooth.*

The local entropy objective is thus smoother than the original objective while the modified objective in (6) results in a factor of $\rho + \frac{1}{1+\gamma^{-1} c}$ in the above lemma.

Let us now obtain a bound on the improvement in generalization error. We denote an optimization algorithm, viz., SGD or Entropy-SGD by $A(\Xi)$, it is a function of the dataset Ξ and outputs the parameters x^* upon termination. Stability of the algorithm (Bousquet & Elisseeff, 2002) is then a notion of how much its output differs in loss upon being presented with two datasets Ξ and Ξ' that differ in at most one sample,

$$\sup_{\xi \in \Xi \cup \Xi'} \left[f(A(\Xi), \xi) - f\left(A(\Xi'), \xi\right) \right] \leq \varepsilon.$$

Hardt et al. (2015) connect uniform stability to generalization error and show that an ε-stable algorithm $A(\Xi)$ has generalization error bounded by ε, i.e., if $A(\Xi)$ terminates with parameters x^*,

$$\left| \mathbb{E}_\Xi \left(R_\Xi(x^*) - R(x^*) \right) \right| \leq \varepsilon;$$

where the left hand side is the generalization error: it is the difference between the empirical loss $R_\Xi(x) := \frac{1}{N} \sum_{k=1}^{N} f(x, \xi_k)$ and the population loss $R(x) := \mathbb{E}_\xi f(x, \xi)$.

We now employ the following theorem that bounds the stability of an optimization algorithm through the smoothness of its loss function and the number of iterations on the training set.

Theorem 3 (Hardt et al. (2015))**.** *For an α-Lipschitz and β-smooth loss function, if SGD converges in T iterations on N samples with decreasing learning rate $\eta_t \leq 1/t$ the stability is bounded by*

$$\varepsilon \lessapprox \frac{1}{N} \alpha^{1/(1+\beta)} T^{1-1/(1+\beta)}.$$

Using Lem. 2 and Thm. 3 we have

$$\varepsilon_{\text{Entropy-SGD}} \lessapprox \left(\alpha T^{-1} \right)^{\left(1 - \frac{1}{1+\gamma^{-1}c} \right) \beta} \varepsilon_{\text{SGD}}, \tag{10}$$

which shows that Entropy-SGD generalizes better than SGD for all $T > \alpha$. For the objective (6), we have the constant $1 - \rho - \frac{1}{1+\gamma^{-1} c}$ in the exponent instead.

As an aside, it is easy to see from the proof of Lem. 2 that for a convex loss function $f(x)$, adding the local entropy term does not change the minimizer of the original problem.

5 EXPERIMENTS

In Sec. 5.1, we discuss experiments that suggest that the characteristics of the energy landscape around local minimal accessible by SGD are universal to deep architectures. This observation motivates the Entropy-SGD algorithm considered in this paper and in Sec. 5.2 and 5.3, we present experimental results on two standard image classification datasets, viz., MNIST and CIFAR-10.

5.1 Universality of the Hessian at local minima

We use automatic differentiation[1] to compute the Hessian at a local minimum obtained at the end of training for the following networks:

 (i) **small-LeNet on MNIST**: This network has $47,658$ parameters and is similar to LeNet but with 10 and 20 channels respectively in the first two convolutional layers and 128 hidden units in the fully-connected layer. We train this with Adam to obtain a test error of 2.4%.

 (ii) **small-mnistfc on MNIST**: A fully-connected network ($50,890$ parameters) with one layer of 32 hidden units, ReLU non-linearities and cross-entropy loss; it converges to a test error of 2.5% with momentum-based SGD.

(iii) **char-lstm for text generation**: This is a recurrent network with 48 hidden units and Long Short-Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997a). It has $32,640$ parameters and we train it with Adam to re-generate a small piece of text consisting of 256 lines of length 32 each and 96-bit one-hot encoded characters.

(iv) **All-CNN-BN on CIFAR-10**: This is similar to the All-CNN-C network (Springenberg et al., 2014) with ≈ 1.6 million weights (cf. Sec. 5.3) which we train using Adam to obtain an error of 11.2%. Exact Hessian computation is in this case expensive and thus we instead compute the diagonal of the Fisher information matrix (Wasserman, 2013) using the element-wise first and second moments of the gradients that Adam maintains, i.e., $\mathrm{diag}(I) = \mathbb{E}(g^2) - (\mathbb{E}\,g)^2$ where g is the back-propagated gradient. Fisher information measures the sensitivity of the log-likelihood of data given parameters in a neighborhood of a local minimum and thus is exactly equal to the Hessian of the negative log-likelihood. We will consider the diagonal of the empirical Fisher information matrix as a proxy for the eigenvalues of the Hessian, as is common in the literature.

We choose to compute the exact Hessian and to keep the computational and memory requirements manageable, the first three networks considered above are smaller than standard deep networks used in practice. For the last network, we sacrifice the exact computation and instead approximate the Hessian of a large deep network. We note that recovering an approximate Hessian from Hessian-vector products (Pearlmutter, 1994) could be a viable strategy for medium-scale networks.
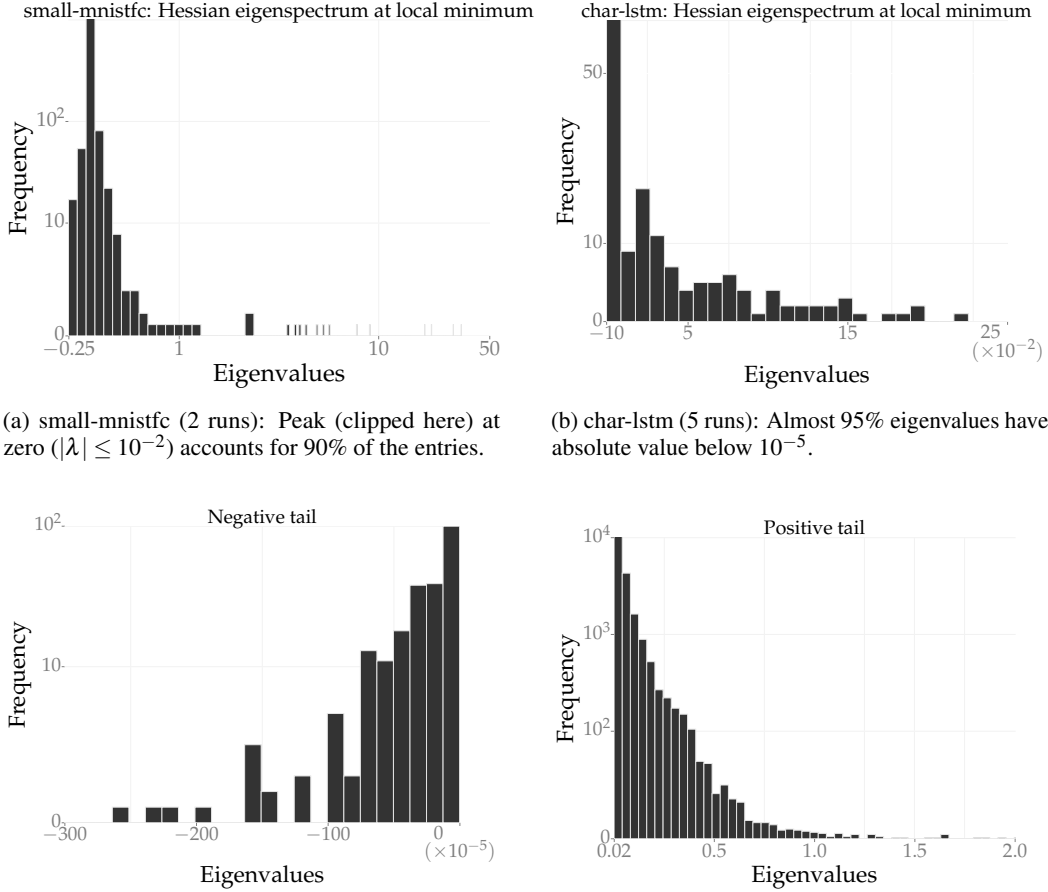
Fig. 1 in the introductory Sec. 1 shows the eigenspectrum of the Hessian for small-LeNet while Fig. 3 shows the eigenspectra for the other three networks. A large proportion of eigenvalues of the Hessian are very close to zero or positive with a very small (relative) magnitude. This is suggests that the local geometry of the energy landscape is almost flat at local minima discovered by gradient descent. This agrees with theoretical results such as Baldassi et al. (2016b) where the authors predict that flat regions of the landscape generalize better. Standard regularizers in deep learning such as convolutions, max-pooling and dropout seem to bias SGD towards flatter regions in the energy landscape. The right tails of the eigenspectra are much longer than the left tails. Indeed, as discussed in numerous places in literature (Bray & Dean, 2007; Dauphin et al., 2014; Choromanska et al., 2015a), SGD finds low-index critical points, i.e., optimizers with few negative eigenvalues of the Hessian. What is interesting and novel is that the directions of descent that SGD misses do not have a large curvature.

5.2 MNIST

In this section, we show that local entropy is a good measure of generalization, in fact, setting $\rho = 0$ in Entropy-SGD, i.e., completely disabling the original loss function is enough to train on MNIST. We consider two networks: the first is a fully-connected network with two hidden layers of 1024 units each which we denote as "mnistfc" and the second is a convolutional neural network similar to LeNet but which batch-normalization (Ioffe & Szegedy, 2015) and a dropout layer of probability 0.5 added after each convolutional layer. We train for 100 epochs with Adam and a learning rate of 10^{-3} that drops by a factor of 5 after every 30 epochs and compare the average error over 5 independent runs in Fig. 4a and Fig. 4b.

Entropy-SGD converges to an error of 1.39% in 22 epochs on mnistfc while Adam obtains the same (best) error after 67 epochs. For a small network such as this, with a presumably easy energy

---
[1] https://github.com/HIPS/autograd

small-mnistfc: Hessian eigenspectrum at local minimum

char-lstm: Hessian eigenspectrum at local minimum

(a) small-mnistfc (2 runs): Peak (clipped here) at zero ($|\lambda| \leq 10^{-2}$) accounts for 90% of the entries.

(b) char-lstm (5 runs): Almost 95% eigenvalues have absolute value below 10^{-5}.

Negative tail

Positive tail

(c) Negative and positive eigenvalues of the Fisher information matrix of All-CNN-BN at a local minimum (4 independent runs). The origin has a large peak with $\approx 95\%$ near-zero ($|\lambda| \leq 10^{-5}$) eigenvalues (clipped here).

Figure 3: Universality of the Hessian: for a wide variety of network architectures, sizes and datasets, optima obtained by SGD are mostly flat (large peak near zero), they always have a few directions with large positive curvature (long positive tails). A very small fraction of directions have negative curvature, and the magnitude of this curvature is extremely small (short negative tails).

landscape, we perform 5 SGLD iterations and set $\gamma = 1$. For LeNet, we perform 20 SGLD iterations and set $\gamma = 0.1$. In this case, Entropy-SGD converges around epoch 40 to an average error of 0.48%; Adam obtains a comparable error of 0.51% after 100 epochs. For this experiment, we use a small $\rho = 10^{-4}$ to fine-tune the network; the error without this is 0.52%.

Remark: A single "epoch" for all algorithms considered here corresponds to the number of parameter updates necessary to run through the entire dataset once. Note that both SGD and Entropy-SGD update parameters after each mini-batch, but Entropy-SGD also runs L iterations of SGLD between two updates of the parameters.

5.3   CIFAR-10

We train on CIFAR-10 without data augmentation after performing global contrast normalization and ZCA whitening (Goodfellow et al., 2013). As a baseline, we consider the All-CNN-C network of Springenberg et al. (2014) with a batch normalization layer after each convolutional layer; this is absent in the original paper. We recreate the training pipeline of the original authors with the same hyper-parameters and use SGD with Nesterov's momentum. We however decrease the initial learning rate of 0.1 by a factor of 5 after every 60 epochs for SGD and after every 5 epochs for Entropy-SGD. Probably due to batch-normalization, we obtain an average error of 8.30% over 5 runs of 200 epochs vs. 9.08% error in 350 epochs as the authors in Springenberg et al. (2014) report. This gives us a very competitive baseline to compare Entropy-SGD against. Let us note that the best result in literature on
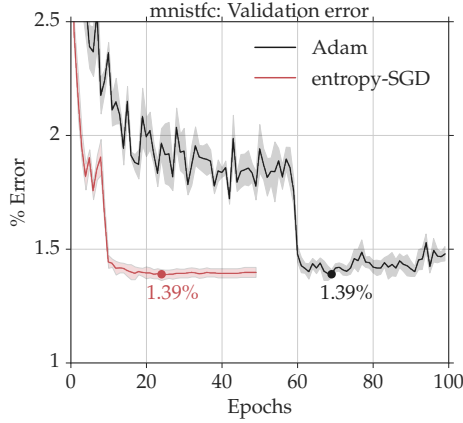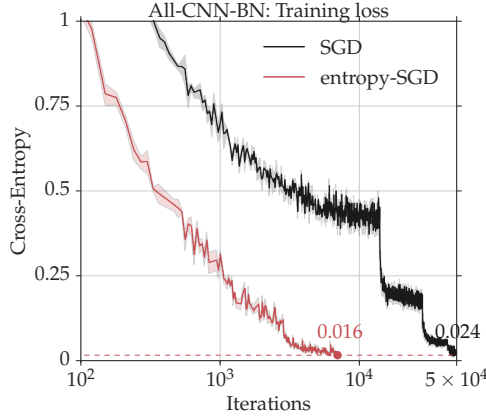
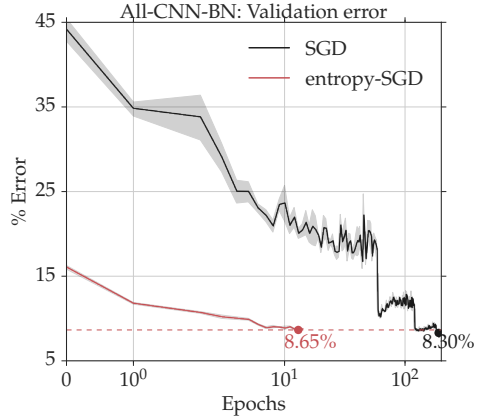(a) mnistfc: Entropy-SGD with $\rho = 0$, $L = 5$.
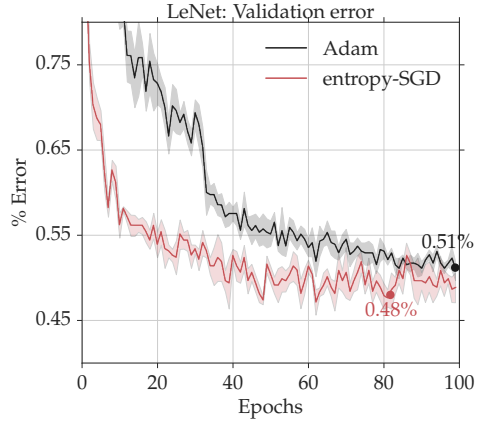
(b) LeNet: Entropy-SGD $\rho = 10^{-4}$, $L = 20$.

Figure 4: Comparison of Entropy-SGD vs. Adam on MNIST

(a) All-CNN-BN: Training loss

(b) All-CNN-BN: Validation error

Figure 5: Comparison of Entropy-SGD vs. Adam on CIFAR-10

non-augmented CIFAR-10 is the ELU-network by Clevert et al. (2015) with 6.55% test error while a wide ResNet of Zagoruyko & Komodakis (2016) obtains an error of 4.17% with data augmentation.

Fig. 5 compares Entropy-SGD against SGD with 20 SGLD iterations for 15 epochs. Since CIFAR-10 is a more challenging dataset than MNIST, we could not obtain better test errors than 9.5% while training with only the local entropy term ($\rho = 0$). We therefore set $\rho = 0.01$ to obtain an error of 8.65% over 5 independent runs. Refer to Fig. 5a, Entropy-SGD is designed to be an algorithm that seeks out wide local minima as opposed to sharp valleys; a side-effect of this is to smooth the energy landscape as we show in Sec. 4.4. In practice, we see this as a steady decrease in training loss for Entropy-SGD without any plateaus that appear in the training loss for SGD. Similarly, the decrease in validation error in Fig. 5b is also steady without any plateaus. Note that the original network All-CNN-C was trained using SGD for 350 epochs (Springenberg et al., 2014).

6 DISCUSSION

As discussed in Sec. 4.2, SGLD updates in the inner loop of our algorithm compute the average gradient of the original loss in the neighborhood of the current weights. An inexpensive and approximate way of computing these updates is to average the gradient over random perturbations of weights during the forward pass itself and, on the same mini-batch as the outer loop. In our experiments, using the same mini-batch performs slightly worse than Entropy-SGD although it is

still better than SGD. Such an implementation is close to averaged SGD of (Polyak & Juditsky, 1992; Bottou, 2012) and worth further study.

In our experiments, Entropy-SGD results in a comparable generalization error as SGD, but always has a lower cross-entropy loss. Roughly speaking, wide valleys favored by Entropy-SGD seem to have lower empirical loss than local minima discovered by SGD. Theoretical models of deep networks (cf. Sec. 2) suggest multiple local minima with the same loss but there is an absence of results in literature about the local geometry at these minima.

7 Conclusions

We introduced an algorithm named Entropy-SGD for optimization of deep networks. This was motivated from the observation that the energy landscape near a local minimum discovered by SGD is almost flat for a wide variety of deep networks irrespective of their architecture, input data or training methods. We connected this observation to the concept of local entropy which we used to bias the optimization towards flat regions that have low generalization error. Our experiments showed that this algorithm is applicable to large deep networks used in practice.

8 Acknowledgements

References

D. Achlioptas and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. In *ACM symposium on Theory of computing*, 2006.

Eugene L Allgower and Kurt Georg. *Numerical continuation methods: an introduction*, volume 13. Springer Science & Business Media, 2012.

A. Anandkumar and R. Ge. Efficient approaches for escaping higher order saddle points in non-convex optimization. *arXiv:1602.05908*, 2016.

C. Baldassi, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina. Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. *Physical review letters*, 115(12):128101, 2015.

C. Baldassi, C. Borgs, J. Chayes, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina. Unreasonable Effectiveness of Learning Neural Nets: Accessible States and Robust Ensembles. *arXiv:1605.06444*, 2016a.

C. Baldassi, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina. Local entropy as a measure for sampling solutions in constraint satisfaction problems. *Journal of Statistical Mechanics: Theory and Experiment*, 2016 (2):023301, 2016b.

P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pp. 421–436. Springer, 2012.

Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2 (Mar):499–526, 2002.

A. Bovier and F. den Hollander. Metastability: A potential theoretic approach. In *International Congress of Mathematicians*, volume 3, pp. 499–518, 2006.

A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2):201–226, 2005.

A. J. Bray and D. S. Dean. The statistics of critical points of gaussian fields on large-dimensional spaces. *Physics Review Letter*, 2007.

Pratik Chaudhari and Stefano Soatto. On the energy landscape of deep networks. *arXiv:1511.06485*, 2015.

C. Chen, D. Carlson, Z. Gan, C. Li, and L. Carin. Bridging the gap between stochastic gradient MCMC and stochastic optimization. *arXiv:1512.07962*, 2015.

T. Chen, E. B. Fox, and C. Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. In *ICML*, 2014.

A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015a.

A. Choromanska, Y. LeCun, and G. Ben Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *COLT*, 2015b.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv:1511.07289*, 2015.

Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron Courville. Recurrent batch normalization. *arXiv:1603.09025*, 2016.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Y. V Fyodorov and I. Williams. Replica symmetry breaking condition exposed by random matrix calculation of landscape complexity. *Journal of Statistical Physics*, 129(5-6),1081-1116, 2007.

R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *COLT*, 2015.

I. J. Goodfellow and O. Vinyals. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. *ICML*, 28: 1319–1327, 2013.

Benjamin D Haeffele and Rene Vidal. Global optimality in tensor factorization, deep learning, and beyond. *arXiv:1506.07540*, 2015.

Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv:1509.01240*, 2015.

David Haussler, Manfred Opper, et al. Mutual information, metric entropy and cumulative relative entropy risk. *The Annals of Statistics*, 25(6):2451–2492, 1997.

E. Hazan, K. Yehuda Levy, and S. Shalev-Shwartz. On graduated optimization for stochastic non-convex problems. In *ICML*, 2016.

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

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997b.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

M. Janzamin, H. Sedghi, and A. Anandkumar. Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods. *arXiv:1506.08473*, 2015.

K. Kawaguchi. Deep learning without poor local minima. In *NIPS*, 2016.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv:1609.04836*, 2016.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Computer Science, University of Toronto, 2009.

Florent Krzakala, Marc Mézard, François Sausset, YF Sun, and Lenka Zdeborová. Statistical-physics-based reconstruction in compressed sensing. *Physical Review X*, 2(2):021005, 2012.

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

Y.-A. Ma, T. Chen, and E. Fox. A complete recipe for stochastic gradient MCMC. In *NIPS*, 2015.

S. Mandt, M. D. Hoffman, and D. M. Blei. A variational analysis of stochastic gradient algorithms. *arXiv:1602.02666*, 2016.

Elitza Maneva, Elchanan Mossel, and Martin J Wainwright. A new look at survey propagation and its generalizations. *Journal of the ACM*, 54(4):17, 2007.

Hossein Mobahi and John W Fisher III. On the link between gaussian homotopy continuation and convex envelopes. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pp. 43–56. Springer, 2015.

R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

G. O. Roberts and O. Stramer. Langevin diffusions and metropolis-hastings algorithms. *Methodology and computing in applied probability*, 4(4):337–357, 2002.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv:1602.07868*, 2016.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*. 2014.

M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

D. Soudry and Y. Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv:1605.08361*, 2016.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806*, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

T Tieleman and G Hinton. Lecture 6.5: RmsProp, Coursera: Neural networks for machine learning. Technical report, Technical report, 2012.

L. Wasserman. *All of statistics: A concise course in statistical inference*. Springer Science & Business Media, 2013.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

S. Zagoruyko and N. Komodakis. Wide Residual Networks. *arXiv:1605.07146*, 2016.

S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In *NIPS*, 2015.

A STOCHASTIC GRADIENT LANGEVIN DYNAMICS (SGLD)

Local entropy in Def. (1) is an expectation over the entire configuration space $x \in \mathbb{R}^n$ and is hard to compute, we can however approximate its gradient using Markov chain Monte-Carlo (MCMC) techniques. In this section, we briefly review stochastic gradient Langevin dynamics (Welling & Teh, 2011) that is an MCMC algorithm designed to draw samples from a Bayesian posterior and scales to large datasets using mini-batch updates.

For a parameter vector $x \in \mathbb{R}^n$ with a prior distribution $p(x)$ and if the probability of generating a data item ξ_k given a model parameterized by x is $p(\xi_k|x)$, the posterior distribution of the parameters based on N data items can be written as

$$p(x|\xi_{k\leq N}) \; \propto \; p(x) \prod_{k=1}^{N} p(\xi_k|x). \tag{11}$$

Langevin dynamics (Neal, 2011) injects Gaussian noise into maximum-a-posteriori (MAP) updates to prevent over-fitting the solution x^* of the above equation. The updates can be written as

$$\Delta x_t = \frac{\eta}{2} \left(\nabla \log p(x_t) + \sum_{k=1}^{N} \nabla p(\xi_k|x_t) \right) + \sqrt{\eta}\; \varepsilon_t; \tag{12}$$

where $\varepsilon_t \sim N(0,1)$ is Gaussian noise and η is the learning rate. In this form, Langevin dynamics faces two major hurdles for applications to large datasets. First, computing the gradient $\sum_{k=1}^{N} \nabla p(\xi_k|x_t)$ over all samples for each update Δx_t becomes prohibitive. However, as Welling & Teh (2011) show, one can instead simply use the average gradient over m data samples (mini-batch) as follows:

$$\Delta x_t = \frac{\eta_t}{2} \left(\nabla \log p(x_t) + \frac{N}{m} \sum_{k=1}^{m} \nabla p(\xi_k|x_t) \right) + \sqrt{\eta_t}\; \varepsilon_t. \tag{13}$$

Secondly, Langevin dynamics in (12) is the discrete-time approximation of a continuous-time stochastic differential equation (Mandt et al., 2016) thereby necessitating a Metropolis-Hastings (MH) rejection step (Roberts & Stramer, 2002) which again requires computing $p(\xi_k|x)$ over the entire dataset. However, if the learning rate $\eta_t \to 0$, we can also forgo the MH step (Chen et al., 2014). Welling & Teh (2011) also argue that the sequence of samples x_t generated by updating (13) converges to the correct posterior (11) and one can hence compute the statistics of any function $g(x)$ of the parameters using these samples. Concretely, the posterior expectation $\mathbb{E}[g(x)]$ is given by $\mathbb{E}[g(x)] \approx \frac{\sum_{s=1}^{t} \eta_t\; g(x_t)}{\sum_{s=1}^{t} \eta_t}$; which is the average computed by weighing each sample by the corresponding learning rate in (13). In this paper, we will consider a uniform prior on the parameters x and hence the first term in (13), viz., $\nabla \log p(x_t)$ vanishes.

Let us note that there is a variety of increasingly sophisticated MCMC algorithms applicable to our problem, e.g., Hamiltonian Monte Carlo (SGHMC) by Chen et al. (2014) based on volume preserving flows in the "parameter-momentum" space, stochastic annealing thermostats (Santa) by Chen et al. (2015) etc. We can also employ these techniques, although we use SGLD for ease of implementation. The authors in Ma et al. (2015) provide an elaborate overview of these methods.

B PROOFS

Proof of Lemma 2. The gradient $-\nabla F(x)$ is computed in Sec. 4.1 to be $\gamma \left(x - \langle x';\, \Xi^{\ell} \rangle \right)$. Consider the term

$$x - \langle x';\, x \rangle = x - Z_{x,\gamma}^{-1} \int_{x'} x'\; e^{-f(x') - \frac{\gamma}{2}\, \|x-x'\|^2}\; dx'$$

$$\approx x - Z_{x,\gamma}^{-1} \int_{s} (x+s)\; e^{-f(x) - \nabla f(x)^{\top} s - \frac{1}{2}\, s^{\top} (\gamma + \nabla^2 f(x)) s}\; ds$$

$$= x \left(1 - Z_{x,\gamma}^{-1} \int_{s} e^{-f(x) - \nabla f(x)^{\top} s - \frac{1}{2}\, s^{\top} (\gamma + \nabla^2 f(x)) s}\; ds \right) - Z_{x,\gamma}^{-1} \int_{s} s\; e^{-f(x) - \nabla f(x)^{\top} s - \frac{1}{2}\, s^{\top} (\gamma + \nabla^2 f(x)) s}\; ds$$

$$= -Z_{x,\gamma}^{-1}\, e^{-f(x)} \int_{s} s\; e^{-\nabla f(x)^{\top} s - \frac{1}{2}\, s^{\top} (\gamma + \nabla^2 f(x)) s}\; ds.$$

The above expression is the mean of a distribution $\propto e^{-\nabla f(x)^\top s - \frac{1}{2} s^\top (\gamma + \nabla^2 f(x))s}$. We can approximate it using the saddle point method as the value of s that minimizes the exponent to get

$$x - \langle x'; \, x \rangle \approx \left(\nabla^2 f(x) + \gamma I \right)^{-1} \nabla f(x).$$

Let us denote $A(x) := \left(I + \gamma^{-1} \nabla^2 f(x) \right)^{-1}$. Plugging this into the condition for smoothness, we have

$$\| \nabla F(x, \gamma) - \nabla F(y, \gamma) \| = \| A(x) \, \nabla f(x) - A(y) \, \nabla f(y) \|$$

$$\leq \left(\sup_x \| A(x) \| \right) \beta \, \| x - y \|.$$

Unfortunately, we can only get a uniform bound if we assume that for a small constant $c > 0$, no eigenvalue of $\nabla^2 f(x)$ lies in the set $[-2\gamma - c, c]$. This gives

$$\left(\sup_x \| A(x) \| \right) \leq \frac{1}{1 + \gamma^{-1} c}.$$

This shows that a smaller value of γ results in a smoother energy landscape, except at places with very flat directions. The Lipschitz constant also decreases by the same factor. The objective $\rho f(x) - F(x, \gamma)$ results in a modified factor $\rho + \frac{1}{1 + \gamma^{-1} c}$. ∎