EASING NON-CONVEX OPTIMIZATION WITH NEURAL NETWORKS

David Lopez-Paz\textsuperscript{1*}, Levent Sagun\textsuperscript{2*}
\\textsuperscript{1}Facebook AI Research, \textsuperscript{2}Institut de Physique Théorique, CEA, CNRS
\texttt{dlp@fb.com, levent.sagun@ipht.fr}

\textbf{Abstract}

Despite being non-convex, deep neural networks are surprisingly amenable to optimization by gradient descent. In this note, we use a deep neural network with $D$ parameters to parametrize the input space of a generic $d$-dimensional non-convex optimization problem. Our experiments show that minimizing the over-parametrized $D \gg d$ variables provided by the deep neural network eases and accelerates the optimization of various non-convex test functions.

1 Introduction

We consider the ubiquitous problem of optimization
\[
    x^* = \arg\min_x f(x),
\]
where $f : \mathcal{X} \to \mathcal{Y}$ is a non-convex differentiable function. Without losing much generality, we consider the case where $\mathcal{X} = [0, 1]^d$, and $\mathcal{Y} = [0, 1]$.

\textbf{Contribution} We demonstrate that it is possible to use deep neural networks (Goodfellow et al., 2016) to over-parametrize the $d$-dimensional input space of non-convex optimization problems (1). Such over-parametrization eases and accelerates the optimization process. Our method is inspired by the fact that deep neural networks, although non-convex, are surprisingly well-behaved when optimized by gradient descent. One possible reason behind this phenomenon is that the parameters of deep neural networks are highly redundant (Denil et al., 2013). Such a redundancy through over-parametrization is one of the reasons for the ease of training in deep neural networks (Livni et al., 2014). Intuitively, over-parametrization is opportunity: damage coming from some poorly initialized or optimized parameters can be undone by the proper optimization of others. In this note, we question whether we can make use of this insight in generic non-convex optimization problems.

\textbf{Motivational Example} Consider the 1-dimensional parabola $f(x) = x^2$ over $[-1, 1]$. We can reparametrize the domain of $f$ by a 1-hidden layer network with two parameters, one fixed input, and linear activations: $x = w_1 w_2 \tilde{x}$. Then, $f(x) = f(w_1 w_2 \tilde{x})$ can be seen as a function of $(w_1, w_2) \in \mathbb{R}^2$. Figure shows how the over-parametrization of $f$ from 1-dimension to 2-dimensions introduces a continuum of equivalent solutions, steeper descent directions, and a flatter basin of global minima.

2 Method

We translate the optimization problem (1) into
\[
    z^*, \theta^* = \arg\min_{z, \theta} \frac{1}{n} \sum_{i=1}^{n} f(g_{\theta}(z_i)),
\]
where $g : \mathbb{R}^k \to [0, 1]^d$ is a deep neural network with parameters $\theta \in \mathbb{R}^p$, and accepting inputs $z = (z_1^*, \ldots, z_n^*) \in \mathbb{R}^{n \times k}$. After approximating the non-convex optimization problem (2), we return

*Alphabetical, joint first authors.
Figure 1: A very simple visual example of what over-parametrization does to the loss-surface: All of the points on the lines $w_1 = 0$ and $w_2 = 0$ correspond to the solution of the original problem.

the solution: $x^* = \min(g_{\theta^*}(z_1^t), \ldots, g_{\theta^*}(z_n^t))$ as our best estimate of the location of the global minima of the objective function $f$.

In a nutshell, we have translated the $d$-dimensional non-convex optimization problem (1) into the equivalent non-convex $D$-dimensional non-convex optimization problem (2), where $D = nk + p \gg d$ is the number of neural network parameters $\theta$ plus the number of variables contained in the $n$ neural network $k$-dimensional inputs $z$. Some observations are in order:

- We assume that the original function has non-degenerate critical points (as in the case of our examples in this note).
- The set of points in the weight space corresponding to different minima of the original function form an equivalence relation.
- Two points taken from different equivalence classes in the weight space has a barrier between them. In general, different solutions are not necessarily connected (in contrast to what’s claimed for deep neural networks (Freeman & Bruna, 2016; Sagun et al., 2017)).
- The minimum height of the barrier between such weights is the same as the minimum height of the barrier between the corresponding minima of the original function.

The problem (1) runs $n$ optimization problems in parallel, one per input in the “minibatch” $z$. These $n$ optimization problems run efficiently by virtue of vectorized (as well as GPU) implementations of deep learning software (Paszke et al., 2017). As it is the case with (1), the problem (2) can be optimized using any gradient-based optimizer. Finally, if the $d$ variables of $f$ are related to each other, the network $g$ may capture these dependencies, accelerating the optimization process.

3 EXPERIMENTS

We present some preliminary experiments comparing shallow versus deep non-convex optimization.

3.1 OPTIMIZER

Due to simplicity and space limitations, we restrict ourselves to vanilla gradient descent. For the deep optimizer, this leads to the updates

$$
\theta^{t+1} \leftarrow \theta^t - \alpha_t \cdot \frac{\partial}{\partial \theta^t} \left( \frac{1}{n} \sum_{i=1}^{n} f(g_{\theta^t}(z^t_i)) \right), \quad z^{t+1} \leftarrow z^t - \alpha_t \cdot \frac{\partial}{\partial z^t} \left( \frac{1}{n} \sum_{i=1}^{n} f(g_{\theta^t}(z^t_i)) \right),
$$

$$
x^{t+1} \leftarrow \min(g_{\theta^t}(z^t_1), \ldots, g_{\theta^t}(z^t_n)),
$$

...
where \( z^0, \theta^0 \) are drawn from a Gaussian distribution, as implemented in PyTorch 0.4.0 (Paszke et al., 2017). The best initial learning rate \( \alpha^0 \) is chosen from \( \{0.001, 0.01, 0.1, 1\} \), and multiplied by a factor 0.9 after each iteration without improvement. Our preliminary experiments showed that this simple strategy outperforms a variety of complex state-of-the-art optimizers (Wilson et al., 2017).

3.2 Test Functions

We perform non-convex optimization of the Ackley, Levy, Rastrigin, Schwefel, Sphere, Rosenbrock, Styblinski-Tang, and Michalewicz functions (Surjanovic & Bingham, 2013). See Appendix A for their expressions. Also, we perform non-convex optimization of the Hamiltonian of a 3-spin spherical glass model (Auffinger et al., 2013). We scale the suggested function input supports (Surjanovic & Bingham, 2013) to \([0, 1]^d\), and scale their output to map the recommended support into \([0, 1]\).

3.3 Neural Network Architecture

The deep optimizer \( g_\theta(z) \) is a feed-forward neural network composed of the following modules:

\[
\text{EmbeddingLayer}(n, n_h) \rightarrow \left[ \text{Linear}(k, n_h) \rightarrow \text{ReLU()} \right] \times n_l \rightarrow \text{Linear}(n_h, d).
\]

The input to the previous architecture are \( n \) \( n_h \)-dimensional embeddings, that is, \( n \) \( n_h \)-dimensional trainable inputs. These inputs are fed to \( n_l \) hidden layers of \( n_h \) ReLU neurons, followed by one final linear transformation into the target \( d \)-dimensional space. The experiments reported below are for \( n = 128, n_h = 128 \), and \( n_l = 1 \), and \( d = 50 \).

3.4 Results

![The Rastrigin function](image1.png) ![The Hamiltonian of the 3-spin model](image2.png)

Figure 2: Iteration number (log-scale) versus objective value for two of the examples tested. Figure 2(a) aggregates results for two learning rates; thus the x-axis is scaled by the learning rate.

After initializing all optimizers to produce the same initial \( x^0 \) value, we run the shallow optimizer 10,000 iterations for the test functions, and 1,000 iterations for the Hamiltonian. We then run the deep optimizer for the same clock time, and repeat this process for 10 different random initializations for the test functions (or three for the Hamiltonian). Figure 2 shows the evolution of the minimum error over all the \( n \) embeddings for the Rastrigin function and the Hamiltonian versus the logarithm of number of iterations. The complete results, measured in both iterations and clock time, can be found in Appendices B and C. Since there is no stochasticity in these experiments, the minimum value in each plot (from all curves at all locations) would be reported as the best estimate of the global minima of the non-convex optimization problem.

Conclusion The deep optimizer usually outperforms vanilla gradient descent in terms of best minima found. Perhaps surprisingly, the deep optimizer is also much faster as per clock time, as in most cases, the deep optimizer converges 10 times faster. Although more extensive experiments are necessary before accepting great claims, it seems that deep neural networks provide a powerful reparametrization that allows to ease and accelerate the optimization of generic non-convex problems.
REFERENCES


Sonja Surjanovic and Derek Bingham. Virtual library of simulation experiments: Test functions and datasets, 2013. URL [https://www.sfu.ca/~ssurjano/optimization.html](https://www.sfu.ca/~ssurjano/optimization.html).

A Test Functions

1. Ackley’s function

\[ f(x) = -20 \exp \left(-0.2 \sqrt{\frac{1}{d} \sum_{i=1}^{d} x_i^2} \right) - \exp \left( \frac{1}{d} \sum_{i=1}^{d} \cos(2\pi x_i) \right) + 20 + \exp(1). \]

2. Levy’s function, using \( w_i = 1 + \frac{x_i - 1}{4} \) for all \( i = 1, \ldots, d \):

\[ f(x) = \frac{1}{d} \left\{ \sin^2(\pi w_1) + \sum_{i=1}^{d-1} \left( w_i - 1 \right)^2 \left[ 1 + 10 \sin^2(\pi w_i + 1) \right] + \left( w_d - 1 \right)^2 \left[ 1 + \sin^2(2\pi w_d) \right] \right\}. \]

3. Rastrigin’s function

\[ f(x) = 10 + \frac{1}{d} \sum_{i=1}^{d} [x_i^2 - 10 \cos(2\pi x_i)]. \]

4. Schwefel’s function

\[ f(x) = 418.9829 - \frac{1}{d} \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|}). \]

5. Sphere’s function

\[ f(x) = \frac{1}{d} \sum_{i=1}^{d} x_i^2. \]

6. Rosenbrock’s function

\[ f(x) = \frac{1}{d} \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]. \]

7. Styblinski-Tang’s function

\[ f(x) = \frac{1}{2d} \sum_{i=1}^{d} \left( x_i^4 - 16x_i^2 + 5x_i \right). \]

8. Michalewicz’s function

\[ f(x) = -\frac{1}{d} \sum_{i=1}^{d} \sin(x_i) \sin^2 \left( \frac{ix_i^2}{\pi} \right). \]

9. The Hamiltonian of a three-dimensional spin glass model

\[ f(x) = \frac{1}{d^3} \sum_{1 \leq j_1, j_2, j_3 \leq d} A_{j_1, j_2, j_3} x_{j_1} x_{j_2} x_{j_3}, \]

where the entries of \( A \in \mathbb{R}^{d \times d \times d} \) are drawn from the Normal distribution. Before evaluating the Hamiltonian, \( x \) is renormalized to satisfy \( \|x\|_2 = 1. \)
B Optimization curves

Figure 3: Iteration number $\times$ learning rate in log-scale, versus minimum objective value. Red curves are the results of the shallow network, and the green curves are for the deep one.

Figure 4: Clock seconds in log-scale, versus minimum objective value. Red curves are the results of the shallow network, and the green curves are for the deep one.
## Optimization Values

<table>
<thead>
<tr>
<th>function, $lr = 0.01$</th>
<th>shallow</th>
<th>deep</th>
<th>lbfgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>rastrigin</td>
<td>0.0830</td>
<td>0.0172</td>
<td>0.0766</td>
</tr>
<tr>
<td>rosenbrock</td>
<td>0.0276</td>
<td>0.0017</td>
<td>0.0273</td>
</tr>
<tr>
<td>sphere</td>
<td>0.7805</td>
<td>0.0009</td>
<td>0.0000</td>
</tr>
<tr>
<td>tang</td>
<td>0.0797</td>
<td>0.0521</td>
<td>0.0796</td>
</tr>
<tr>
<td>michal</td>
<td>0.3884</td>
<td>0.1958</td>
<td>0.2921</td>
</tr>
<tr>
<td>levy</td>
<td>0.0164</td>
<td>0.0043</td>
<td>0.0162</td>
</tr>
<tr>
<td>ackley</td>
<td>0.0302</td>
<td>0.0276</td>
<td>0.0301</td>
</tr>
<tr>
<td>schwefel</td>
<td>0.2970</td>
<td>0.2255</td>
<td>0.2957</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>function, $lr = 0.10$</th>
<th>shallow</th>
<th>deep</th>
<th>lbfgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>rastrigin</td>
<td>0.0119</td>
<td>0.0126</td>
<td>0.0524</td>
</tr>
<tr>
<td>rosenbrock</td>
<td>0.0208</td>
<td>0.0002</td>
<td>-0.0000</td>
</tr>
<tr>
<td>sphere</td>
<td>0.0282</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>tang</td>
<td>0.0780</td>
<td>0.0121</td>
<td>0.0774</td>
</tr>
<tr>
<td>michal</td>
<td>0.3165</td>
<td>0.2394</td>
<td>0.3093</td>
</tr>
<tr>
<td>levy</td>
<td>0.0138</td>
<td>0.0025</td>
<td>0.0024</td>
</tr>
<tr>
<td>ackley</td>
<td>0.0275</td>
<td>0.0275</td>
<td>0.0275</td>
</tr>
<tr>
<td>schwefel</td>
<td>0.2616</td>
<td>0.2231</td>
<td>0.2225</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>function, $lr = 1.00$</th>
<th>shallow</th>
<th>deep</th>
<th>lbfgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>rastrigin</td>
<td>0.1037</td>
<td>0.0765</td>
<td>0.1919</td>
</tr>
<tr>
<td>rosenbrock</td>
<td>0.0766</td>
<td>0.0673</td>
<td>-0.0000</td>
</tr>
<tr>
<td>sphere</td>
<td>0.7805</td>
<td>0.3650</td>
<td>0.0000</td>
</tr>
<tr>
<td>tang</td>
<td>0.0886</td>
<td>0.0883</td>
<td>0.0175</td>
</tr>
<tr>
<td>michal</td>
<td>0.6988</td>
<td>0.7521</td>
<td>0.5469</td>
</tr>
<tr>
<td>levy</td>
<td>0.0126</td>
<td>0.0122</td>
<td>0.0059</td>
</tr>
<tr>
<td>ackley</td>
<td>0.0297</td>
<td>0.0295</td>
<td>0.0297</td>
</tr>
<tr>
<td>schwefel</td>
<td>0.3022</td>
<td>0.2977</td>
<td>NaN</td>
</tr>
</tbody>
</table>