Notes for Sampad's Talk:

On the Optimization of Deep Networks: Implicit Acceleration by Overparametrization

Transcribed by Qilin Ye

March 27, 2023

Introduction, Motivation, & Results

It is of general consensus that when one increases the depth of a neural network, one improves a network's expressiveness at the cost of more complicated optimization. This paper, however, provides a counterintuitive message: deeper network *accelerates* optimization! In particular, it provides implicit momentum and adaptive regularization. In this paper we will discuss the following results:

(1) GD on overparamteization by depth is equivalent to GD on a shallow, 1-layer NN, while employing some particular pre-conditioning scheme.

(2) On simple ℓ^p loss, $p > 2$, overparametrization by depth significantly improves training speed. They sometimes even outperform well-known acceleration methods designed specifically for convex objectives (e.g. AdaGrad, AdaDelta).

(3) Implicit acceleration due to overparametrization by depth may somewhat be orthogonal[?] to explicit acceleration schemes.

Warmup: ℓ^p Regression

Instead of considering the ℓ^p loss

$$L(w) := \mathbb{E}_{(x,y)\sim S}[p^{-1}(x^T w - y)^p]$$

where S is the training set and $w \in \mathbb{R}^d$ the learned parameter vector, we apply a simple overparametrization by an additional scalar $w_2 \in \mathbb{R}$:

$$L(w_1, w_2) = \mathbb{E}_{(x,y)\sim S}[p^{-1}(x^T w_1 w_2 - y)^p].$$

The gradient descent on w is

$$w_1^{(t+1)} \leftarrow w_1^{(t)} - \eta \nabla_{w_1^{(t)}} \qquad w_2^{(t+1)} \leftarrow w_2^{(t)} - \eta_{w_2^{(t)}}.$$

If we view them as one parameter, $w = w_1 w_2$ (vector times scalar), then by some algebra

$$w^{(t+1)} \leftarrow w^{(t)} - \eta (w_2^{(t)})^2 \nabla_{w^{(t)}} - \eta (w_2^{(t)})^{-1} \nabla_{w_2^{(t)}} + \mathcal{O}(\eta^2).$$

If η is assumed to be small then writing $\rho^{(t)} = \eta(w_2^{(t)})^2 \in \mathbb{R}$ and $\gamma^{(t)} = \eta(w_2^{(t)})^{-1}\nabla_{w_2^{(t)}} \in \mathbb{R}$, we obtain

$$w^{(t+1)} \leftarrow w^{(t)} - \rho^{(t)}\nabla_{w^{(t)}} - \gamma(t)w^{(t)}. \tag{1}$$

Observations of (1):

(1) Both the learning rate (r) and momentum coefficient (γ) are time-varying and adaptive.

(2) ρ, γ are independent of w_1 and ∇w_1 but only on the extra parameter w_2. That is, optimization results are exclusively from overparametrization.

(3) Learning rate increases as w_2 drifts away from zero.

(4) About momentum: it has a tendency to proceed in the direction of w (the current direction of the parameters).

(5) Since w_1, w_2 are initialized near 0, so is w. At every iteration t, $w^{(t)}$ is a weighted combination of past gradients. That is, there exists $\mu^{(t,\tau)} \in \mathbb{R}$ such that

$$w^{(t+1)} \leftarrow w^{(t)} - \rho^{(t)}\nabla_{w^{(t)}} - \sum_{\tau=1}^{t-1} \mu^{(t,\tau)}\nabla_{w^{(\tau)}}. \tag{2}$$

It is noticed in discussion that this form resembles some second-order methods.

Linear Neural Nets

For a simple one-layer network, we can view the mapping as a matrix multiplication: $\varphi^1(x) := x \mapsto Wx$. For a n-layer network, similarly, $\varphi^n = W_n...W_1x$. We denote the loss by $L^1(W)$ and $L^n(W_nW_2...W_1)$, respectively.

Implicit Dynamics of Gradient Descent

When applied to L^N, gradient descent takes the following term:

$$W_j^{(t+1)} \leftarrow (1-\eta\lambda)W_j^{(t)} - \eta\frac{\partial L^N}{\partial W_j}(W_1^{(t)}, ..., W_N^{(t)})$$

for $1 \leqslant j \leqslant N$, where $\lambda \geqslant 0$ is an optional weight coefficient. We define $W_e := W_N...W_1$, the *end-to-end weight matrix*. Assuming $\eta^2 \approx 0$, we can translate the above equation roughly to the following set of differential equations (for $\leqslant j \leqslant N$):

$$\dot{W}_j(t) = -\eta\lambda W_j(t) - \eta\frac{\partial L^N}{\partial W_j}(W_1(t), ..., W_N(t)). \tag{6}$$

Theorem 1

Assume the weight matrices $W_1, ..., W_N$ follow (6). Assume their initial values at t_0 satisfy

$$W_{j+1}^T(t_0)W_{j+1}(t_0) = W_j(t_0)W_j^T(t_0).$$

Then, the end-to-end weight matrix W_e is governed by the following DE:

$$\dot{W}_e(t) = -\eta\lambda N W_e(t) - \eta\sum_{j=1}^{N}[W_e(t)W_e^T(t)]^{(j-1)/N} \cdot \frac{\mathrm{d}L^1}{\mathrm{d}W}(W_e(t)) \cdot [W_e^T(t)W_e(t)]^{(N-j)/N}$$

where the exponentiations are fractional powers defined on PSD matrices.

Gradient Flow: Update Rule

Translating the above theorem to discrete time, we obtain the end-to-end weight matrix:

$$W_e^{(t+1)} \leftarrow (1-\eta\lambda N)W_e(t) - \eta \sum_{j=1}^{N} [W_e^{(t)}(W_e^{(t)})^T]^{(j-1)/N} \cdot \frac{\partial L^1}{\partial W}(W_e^{(t)}) \cdot [(W_e^{(t)})^T W_e^{(t)}]^{(N-j)/N}. \tag{10}$$

Denoting vec(A) the column-first arrangement of a matrix A, the end-to-end update rule above becomes

$$\text{vec}(W_e^{(t+1)}) \leftarrow (1-\eta\lambda N) \cdot \text{vec}(W_e^{(t)}) - \eta \cdot P_{W_e^{(t)}}(\frac{\mathrm{d}L^1}{\mathrm{d}W}(W_e^{(t)})) \tag{11}$$

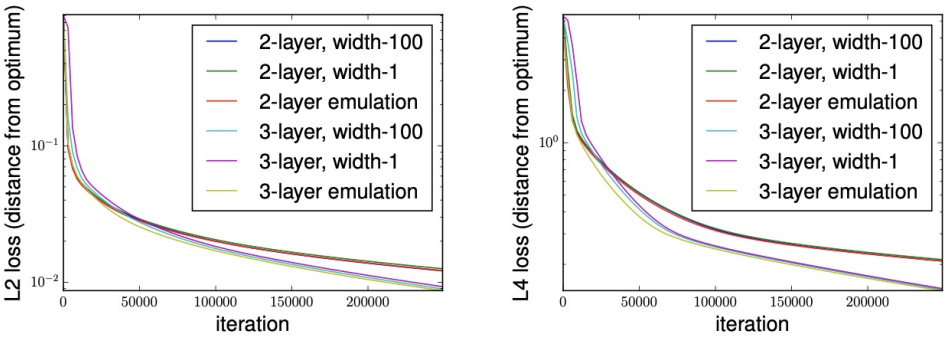
where $P_{W_e^{(t)}}$ is a PSD preconditioning matrix depending on $W_e^{(t)}$.
Further, $P_{W_e^{(t)}}$ is characterized by the following: if we denote the singular values of $W_e^{(t)}$ by $\sigma_1, ..., \sigma_{\max(k,d)}$ and left and right singular values by $u_1, ..., u_k$ and $v_1, ..., v_d$, then the eigenvectors of $P_{W_e^{(t)}}$ are

$$\text{vec}(u_r v_{r'}^T) : (r, r') \in \{1, ..., k\} \times \{1, ..., d\}$$

and corresponding eigenvalues

$$\sum_{j=1}^{N} \sigma_r^{2(N-j)/N} \sigma_{r'}^{2(j-1)/N}.$$

Empirical Validation: ℓ^2 vs. ℓ^4



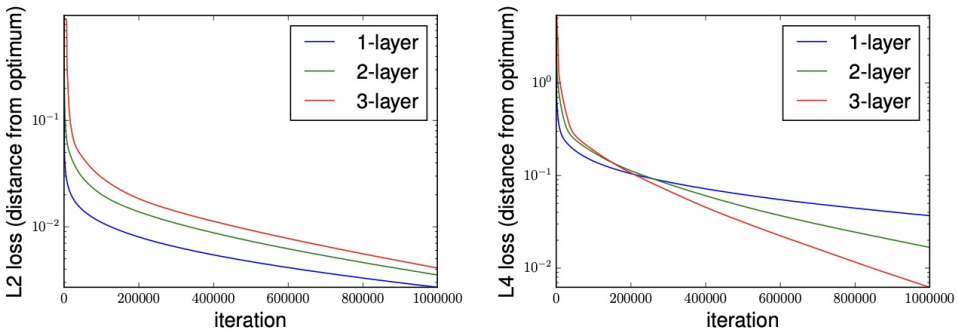
(1) Emulation follows closely.

(2) Network width does not matter (hidden layer of size 11 suffices; acceleration can be achieved without much computation overhead).

(3) Faster convergence with depth, using the same learning rate 10^{-3}.

(4) The authors do point that same learning rate for all models is unfair: that the learning rate must be tuned independently in all models. To address this, they test —

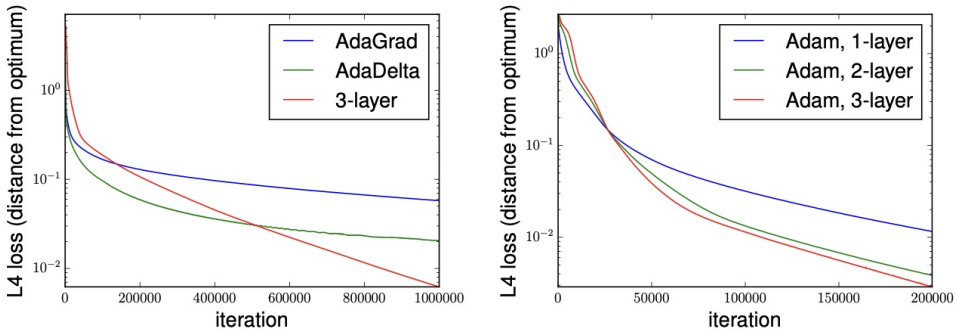
Per Model Learning Rates



(1) For ℓ^2, depth slightly hinders optimization (only for regression problems).

(2) For ℓ^4, depth accelerates optimization.

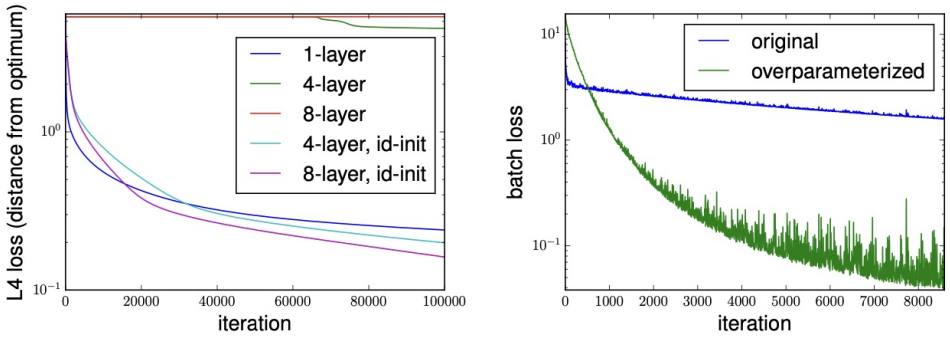
Implicit Acceleration vs. Explicit Schemes?



(1) Depth 3 converges faster than AdaGrad and AdaDelta but not Adam (left).

(2) Increasing depth helps convergence when used with Adam, suggesting implicit acceleration due to depth may be somewhat orthogonal to explicit acceleration.

Going Even Deeper & MNIST Results



On the left: one may encounter vanishing gradients due to conditions of zero-init and preconditioning scheme in (10). Therefore, it is not a very good idea to go overly deep.
On the right: the green represents an overparametrization by 15%, adding one linear hidden layer to all dense layers.