First Order Optimization Methods in Two-Layer Neural Networks Training

Anonymous Authors[1]

Abstract

We investigate the extents and limits of first order methods optimization algorithms when training two-layer neural networks for a finite number of time steps. We characterize rigorously the class of functions that can be learned efficiently from two-layer networks with a finite number of iterations from a general first order algorithms. We corroborate the claims with extensive numerical simulations.

1. Introduction

....

2. Main Results

Our main contribution in this paper are the following:

- We characterize the class of multi-index targets that can be learned efficiently from two-layer networks trained with extensive batch sizes ($n = \alpha d, \alpha = O(1)$) with a finite number of iterations of a general first order algorithms. This class coincides with the class of functions that can be learned with gradient descent, known in the literature as staircase functions.
- We write closed form equations governing the dynamics of the low-dimensional projections of the weights in the teacher subspace and their norms when training with gradient descent. We illustrate the technical differences when studying the online regime - resampling fresh batch of new data at every step - or in the correlated samples regime. The theoretical analysis is based on techniques coming from statistical physics (Dynamical Mean Field Theory, Saad Solla approach) and we believe is of independent interest.
- We corroborate the theoretical claims with extensive numerical simulations. We show that the theoretical predictions are in good agreement with the numerical simulations.

3. Related works

....

4. Setting and Notation

Let $\mathcal{D}$ be the set of labeled data $\{\vec{z}^\nu, y^\nu\}_{\nu\in[n]}$. We consider the labels generated by a multi-index functions:

$$y^\nu = f(W^\star z^\nu) + \sqrt{\Delta}\xi^\nu, \tag{1}$$

where $W^\star \in \mathbb{R}^{k\times d}$ is an orthogonal matrix and $\xi^\nu \sim \mathcal{N}(0,1)$ is the artificial noise. A key assumption is that the target depends only on a finite number of low-dimensional projections of the input, i.e., we assume $k = O(1)$. We are implying that y^ν depends on $\vec{z}^\nu \sim \mathcal{N}(0, I_d)$ just through a low-dimensional representation (linear latent variables).

We introduce the local fields as:

$$\vec{\lambda}^\nu := W\vec{z}^\nu \in \mathbb{R}^p, \quad \vec{\lambda}^{\star\nu} := W^\star\vec{z}^\nu \in \mathbb{R}^k \qquad \forall\nu \in [n] \tag{2}$$

We fit these data using a two-layer neural network. Let the first layer weights be $W \in \mathbb{R}^{p\times d}$, the second layer weights $\vec{a} \in \mathbb{R}^p$; the full expression of the network is given by

$$s(\vec{z}) = \frac{1}{p}\sum_{j=1}^{p} a_j\sigma(\vec{w}_j^\top\vec{z}),$$

where w_j are the rows of W and σ is the activation function; $a_j \sim \mathcal{N}(0,1)$ - or similar distribution, e.g,. $a_j = Ber(\pm1)$.

Since $\vec{z}^\nu$ is Gaussian and independent from $(W, W^\star)$, the pre-activations are jointly Gaussian vectors $(\vec{\lambda}^\nu, \vec{\lambda}^{\star\nu}) \sim \mathcal{N}(\vec{0}_{p+k}, \Omega)$ with covariance:

$$\Omega := \begin{pmatrix} Q & M \\ M^\top & P \end{pmatrix} = \begin{pmatrix} WW^\top & WW^{\star\top} \\ W^\star W^\top & W^\star W^{\star T} \end{pmatrix} \in \mathbb{R}^{(p+k)\times(p+k)} \tag{3}$$

Training algorithm

We are going to train the network with layer-wise SGD without replacement, using at each time step batches of size

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

n_b:

$$\ell = \frac{1}{2n_b}\sum_{\nu=1}^{n_b}(y^\nu - s(\vec{z}^\nu))^2$$

For the moment, let's suppose the second layer fixed. The gradient of the first layer weights

$$\nabla_{\vec{w}_j}\ell = -\frac{1}{pn_b}\sum_{\nu=1}^{n_b} a_j\sigma'(\lambda_j^\nu)\mathcal{E}^\nu\vec{z}^\nu \qquad \forall j\in[p]$$

where we defined for convenience the displacement vector

$$\mathcal{E}^\nu := y^\nu - s(\vec{z}^\nu). \tag{4}$$

Take now one gradient step with learning rate γ_d:

$$\vec{w}_j^{\tau+1} = \vec{w}_j^\tau - \gamma_d\nabla_{\vec{w}_j}\ell \tag{5}$$

By projecting the gradient update equation on the matrices (W, W^*) we obtain the following dynamics:

$$\begin{aligned} M_{jr}^\tau - M_{jr}^{\tau-1} =& \frac{\gamma_d}{pn_b}a_j^\tau\sum_{\nu=1}^{n_b}\sigma'(\lambda_j^\nu)\lambda_r^\star\mathcal{E}^\nu \\ Q_{jl}^\tau - Q_{jl}^{\tau-1} =& \frac{\gamma_d}{pn_b}\sum_{\nu=1}^{n_b}\left(a_j^\tau\sigma'(\lambda_j^\nu)\lambda_l^\nu + a_l^\tau\sigma'(\lambda_l^\nu)\lambda_j^\nu\right)\mathcal{E}^\nu \\ &+ \frac{\gamma_d^2}{p^2n_b^2}a_j^\tau a_l^\tau\sum_{\nu=1}^{n_b}\sum_{\nu'=1}^{n_b}\sigma'(\lambda_j^\nu)\sigma'(\lambda_l^{\nu'})\mathcal{E}^\nu\mathcal{E}^{\nu'}\vec{z}^{\nu\top}\vec{z}^{\nu'} \end{aligned} \tag{6}$$

The main technical contribtuion is to precisely characterize the dynamics of 2LNN in the high dimensional setting ($d\to\infty$) for two specific training algorithm: $(n_b, \gamma_d) = (1, \frac{\gamma_0}{d})$ (Online SGD) and $(n_b, \gamma_d) = (d, \gamma_0)$.

We will consider as well spherical gradient descent, i.e., the modification of eq. (**??**):

$$\vec{w}_j^{\tau+1} = \frac{\vec{w}_j^\tau - \gamma_d\nabla_{\vec{w}_j}\ell}{||\vec{w}_j^\tau - \gamma_d\nabla_{\vec{w}_j}\ell||} \tag{7}$$

One can show that by projecting on (W, W^*) also the spherical dynamics described in eq. (**??**) can be compactly written in terms of the overlaps.
We will show that in the high dimensional limit the key quantities which will determine the dynamical evolution of the overlaps are compactly written as:

$$\begin{aligned} \psi_{jr} =& \mathbb{E}\left[\sigma'(\lambda_j)\lambda_r^\star\mathcal{E}\right] \\ \phi_{jl}^{\text{GF}} =& \mathbb{E}\left[\sigma'(\lambda_j)\lambda_l\mathcal{E}\right] \\ \phi_{jl}^{\text{HD}} =& \mathbb{E}\left[\sigma'(\lambda_j)\sigma'(\lambda_l)\mathcal{E}^2\right] \end{aligned} \tag{8}$$

We can also introduce the population loss as

$$\mathcal{R} = \frac{1}{2}\mathbb{E}\left[\mathcal{E}^2\right], \tag{9}$$

since it is the quantity telling us the performace of our trained network.

5. Theoretical results

6. Numerical simulations