Training on parameters subspaces

June 21, 2022
Miriam Aquaro

Abstract

In this project, we train a simple model of neural network not in its native parameter space, instead on a smaller, randomly oriented subspace of it, in such a way that we can choose an arbitrary number of free parameters to optimize without changing the model's architecture.

1. Introduction

Standard neural network training consists in computing the weights optimization in the full parameter space (let us suppose to be $\mathbb{R}^D$), instead, in this project, we optimize in a randomly oriented d-dimensional subspace $\mathbb{R}^d \subseteq \mathbb{R}^D$. This is accomplished by writing the vector $w_D \in \mathbb{R}^D$ of the parameters as

$$w_D = w_0 + P w_d, \text{ where } P \in \mathbb{R}^{D\times d}, w_d \in \mathbb{R}^d, w_0 \in \mathbb{R}^D; \tag{1}$$

P is a projection matrix and, just like w_0, is randomly generated and frozen (not trained). The idea is to train a three layer perceptron (MLP) in order to test experimentally its performance in classifying Fashion-MNIST for different subspace dimensions and for different families of P. Moreover we report also on the main properties of such network compared to a standard smaller neural network.

2. Methodology

The Fashion-MNIST training data consists in the pairs $\{\boldsymbol{x}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\}_{n=1}^{\mathcal{N}}$, where $\mathcal{N}$ is the total number of observations, $\boldsymbol{x}^{(n)} \in \mathbb{R}^{784}$ is the vector containing the pixels of the n-image and $\hat{\boldsymbol{y}}^{(n)} \in \mathbb{R}^{10}$ is a binary vector that has zero in all entries except in the position corresponding to the category of $\boldsymbol{x}^{(n)}$, which is marked with a 1.
The network choosen has three layers: an input layer made up of $N = 784$ neurons ($\boldsymbol{x} \in \mathbb{R}^N$), an hidden layer of K neurons ($\boldsymbol{z} \in \mathbb{R}^K$) and an output layer of 10 neurons ($\boldsymbol{y} \in \mathbb{R}^{10}$). At the hidden layer we have

$$\boldsymbol{z} = g(\Gamma_1 \boldsymbol{x} + \boldsymbol{b}_1) \tag{2}$$

where $\Gamma_1 \in \mathbb{R}^{K\times N}, \boldsymbol{b}_1 \in \mathbb{R}^K$ and g is the Leaky ReLU activation function: $g(x) = \max(0.1x, x)$. Finally at the output we have:

$$\boldsymbol{y}_i = \text{SoftMax}(\Gamma_2 \boldsymbol{z} + \boldsymbol{b}_2)_i \tag{3}$$

where $\text{SoftMax}(\boldsymbol{x}_i) = \exp(\boldsymbol{x}_i)/\sum_j \exp(\boldsymbol{x}_j)$, $\Gamma_2 \in \mathbb{R}^{10\times K}, \boldsymbol{b}_2 \in \mathbb{R}^{10}$. By taking the maximum over the entries of $\boldsymbol{y}$, we can extract the network's classification answer. The loss function to be minimized during the training session is

$$\mathcal{L} = -\sum_{n=1}^{\mathcal{N}} \sum_{i=1}^{10} \hat{\boldsymbol{y}}_i^{(n)} \log(\boldsymbol{y}_i^{(n)}). \tag{4}$$

The total number of parameters of this network (including weights and biases) is

$$D = N \times K + K + K \times 10 + 10 \tag{5}$$

Instead of training the network directly on w_D (the vector whose entries are the D parameters of the network), we will train the network on w_d, which is given by equation (1). After the training session, we can evaluate the performance on the test set through the quantity

$$\text{Accuracy} = \frac{1}{\mathcal{M}} \sum_{n=1}^{\mathcal{M}} \delta_{c_n, \hat{c}_n} \tag{6}$$

where $\mathcal{M}$ is the total number of test observations, c_n is the network's prediction, i.e. $c_n = \underset{i}{\text{argmax}}(\boldsymbol{y}_i^{(n)})$ and $\hat{c}_n \in \{1, \ldots, 10\}$ is the true label.
There are several ways in which we can construct the projection matrix P: we can simply generate its columns with gaussian random entries[1] $P_{ij} \sim \mathcal{N}(0,1)(\forall i = 1, \ldots, D, \forall j = 1, \ldots, d)$ and then we normalize each column (*dense projection* matrix) or we can construct a sparse matrix (*sparse projection* matrix) as follows: each entry is chosen to be non-zero with probability $1/\sqrt{D}$, then, the entries with non zero values can be either positive or negative with probability $1/2$. Since we want also the column of the sparse projection matrix to be normalized, this is almost true (in the case $D \gg 1$) if the absolute values of the non zero entries is equal to $D^{1/4}$. In order to make the matrix more sparse, we repeat the last procedure with the probability for the entries to be non zero equal to $1/\sqrt{Dd}$ (*deeply sparse projection* matrix).

Email: Miriam Aquaro <miriam.aquaro@uniroma1.it>.

Deep Learning and Applied AI 2021, Sapienza University of Rome, 2nd semester a.y. 2020/2021.

[1]The columns will be approximately orthogonal if $D \gg 1$ because of the independence of the entries.

3. Experimental results

The test accuracy of the network (with P being the deeply sparse projection matrix) is reported in the following table:

$K \backslash d$	50	391	3065	24000
100	46.39%	75.52%	85.21%	**87.93%**
10	33.76%	67.88%	82.81%	84.51%

Now, let us compare two networks (sharing the same architecture described in the introduction): one with an hidden layer of K neurons and another with an hidden layer of $2K$ neurons. The smaller network has D_1 parameters (given by equation 5) whereas the second has $D > D_1$ parameters. Thus, to train them on a parameters space of the same dimension, we have to write w_D, i.e. the vector of the parameters of the bigger network, as prescripted by equation (1) with $P \in \mathbb{R}^{D \times D_1}$ and $\theta_0 \in \mathbb{R}^{D_1}$. In figure 1, there are the test accuracy and the training loss (after 1500 gradient descent iterations) of the smaller (standard) neural network and of the bigger (projected with dense sparse projection matrix) network as a function of D_1.

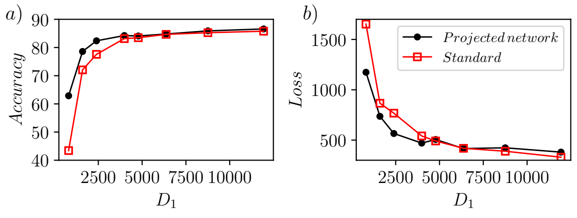

Figure 1. Test accuracy and training Loss as a function of the dimension of the training subspace D_1.

The projected network, even if trained on a (randomly oriented) parameters space of the same dimension of the full parameters space of the smaller network, performs better for $D_1 \lesssim 5000$. We can have an idea of why it happens with a simple but explanatory example. Let us suppose to have a set of data in $\mathbb{R}^2$ generated in a such a way that they can stay on the perimeters of two concentric circles centered at the origin; we want to establish a method to separate them according on whether they are on the inner or the outer circumference. First, we try to separate them through the following scalar function:

$$z_1(x, y) = x \sin(\phi) + y \cos(\phi), \tag{7}$$

as we can see from figure 2a, it is impossibile to distinguish the two families of points by varying ϕ and watching at the values of z_1 because of the overlap in the region $-1 \leq z_1 \leq 1$. Instead, if we use the following scalar function:

$$z_2(x, y) = z_1 \cos(\theta) + W(x, y) \sin(\theta) \tag{8}$$

where $W := \sqrt{x^2 + y^2}$, there are several values of θ for which the data set is separable into two partitions with empty intersection (see figure 2b). Let us note that z_2 has one more non-linearity with respect to z_1 and, in an analogous manner, a bigger network (with a wider hidden layer) performs better than a smaller network even if trained in a randomly oriented parameters space of the same dimension as that of the small network (e.g. in the toy example, both z_1 and z_2 are both scalar functions with values in $\mathbb{R}^1$).

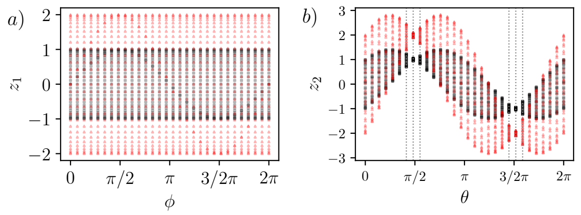

Figure 2. z_1 and z_2 scalar functions.

Finally let us observe that the use of sparse projection matrices allows us to save on the computational cost (see figure 3b): it is because we don't have to store a $D \times d$ matrix.

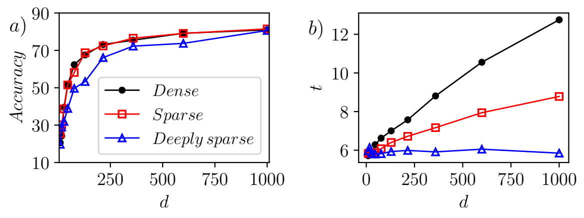

Figure 3. The width of the hidden layer in the simulation is $K = 10$, the test accuracy for the Dense and Sparse model is almost the same whereas it is lower in the Deeply sparse case because most of the entries of P are zero. However, the differences in the test accuracy between the different models disappear as d increases.

4. Discussion and conclusions

Using the strategy of training networks on parameters subspaces, we compared two neural networks with the same architecture (the first with an hidden layer twice as large as that of the other) trained on parameters spaces of the same dimension: we showed that the larger network has better performance than the other (despite the number of actual parameters being the same), but this effect disappears when the number of total parameters increases. So the projection strategy is not very useful to increase the performance, rather it is a method to understand how sensitive the architecture of a network is to a change in the number of parameters (Li et al., 2018). Moreover, the construction of the projection matrix becomes more and more computationally expensive as its size becomes larger (the complexity in terms of memory is of the order of $O(Dd)$); however, this obstacle can be overcome through the use of the deeply sparse matrix we introduced before which can reduces the complexity to an order of $O(\sqrt{D}d)$.

References

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the Intrinsic Dimension of Objective Landscapes. Technical Report arXiv:1804.08838, arXiv, April 2018. URL http://arxiv.org/abs/1804.08838. arXiv:1804.08838 [cs, stat] type: article.