# Training on Parameters Subspaces

January 24, 2023

**Robert Adrian Minut**

## Abstract

Training a neural network can be seen as a walk through an objective landscape, which is fixed once we define a network architecture and a dataset. We can take slices of the full space using a Projection Matrix $P$ and a set of Parameters $W$, and train a network with parameters $K = PW$ in this subspace. We'll see through various experiments what such networks are capable of and how we can use them for studying model architectures and their efficacy on a set of tasks in more detail.

## 1. Introduction

Considering a network with parameters $K$, which lives in a $D$-dimensional space, we can define $W$ in a $d$-dimensional subspace such that: $K = PW$, where $P$ is a $D \times d$ projection matrix. We can now train a network using slices ($d < D$) of the objective landscape.
Given the computational and memory limits of Google Colab, the architectures considered are the Multi Layer Perceptron (MLP), for which we only introduce a Fully Connected layer between input and output, and the Standard LeNet(Li et al., 2018).

## 2. Method

One way of projecting the parameters $K$ onto a subspace is through Random Projections (Li et al., 2018). Another key idea is that we can actually consider the parameters of each layer separately and use a projection matrix for each of them. So we have a projection matrix $P_i$ for each of the $N$ layers:

$$K_i = P_i W_i \quad \text{where } K_i \in \mathbb{R}^{D_i}, \quad P_i \in \mathbb{R}^{d_i \times D_i} \quad \forall i$$

The reference paper doesn't consider this idea, but it was the only way for fitting everything inside the 12GB GPU memory offered by Google Colab when using a larger number of parameters. The results are in line with what they obtained, so it doesn't seem to have any impact onto the training of the network. It also introduces more hyperparameters which can be finetuned for better results, as shown in (Add.Material).

Email: R.A. Minut <minut.1942806@studenti.uniroma1.it>.

*Deep Learning and Applied AI 2022, Sapienza University of Rome*, 2nd semester a.y. 2021/2022.

### 2.1. Gaussian Random Projection

The first projection method consists of generating a $D \times d$ matrix $P$ where each element is sampled independently using a Gaussian distribution $p_{ij} \sim \mathcal{N}(0, 1)$. The large number of values and the use of the Gaussian distribution means that we will have an approximate orthogonal matrix, once we scale each column to unit length. We can also approximate this operation using a scaling factor of $1/\sqrt{d}$.

### 2.2. Orthogonal Random Projection

We can generate a Gaussian Random Projection (Section 2.1), and then orthogonalize $P$. In my implementation I used QR Decomposition since it's the most efficient algorithm offered by PyTorch (PyTorch, a) in terms of memory. By orthogonalizing the matrix we have some guarantee that the projection will introduce lower distortion, which also means that the optimization process will be smoother (Li et al., 2018).

### 2.3. Sparse Random Projection

With the method introduced by (Achlioptas, 2003), we can build a Random Projection $P$ which is both Sparse and an Approximate Orthogonal matrix.
We can optimize further by using Very Sparse Random Projections (Li et al., 2006). Instead of fixing $d = 3$, we can use the actual number of dimensions of the subspace. By doing so, the matrix is even more sparse, resulting in more efficient computations. The $i.i.d.$ values of $P$ in this case are sampled through:

$$p_{ij} = \begin{cases} \frac{\sqrt{\frac{1}{s}}}{\sqrt{D}} & \text{with prob.} \quad \frac{1}{2s} \\ 0 & \text{with prob.} \quad 1 - \frac{1}{s} \\ -\frac{\sqrt{\frac{1}{s}}}{\sqrt{D}} & \text{with prob.} \quad \frac{1}{2s} \end{cases} \tag{1}$$

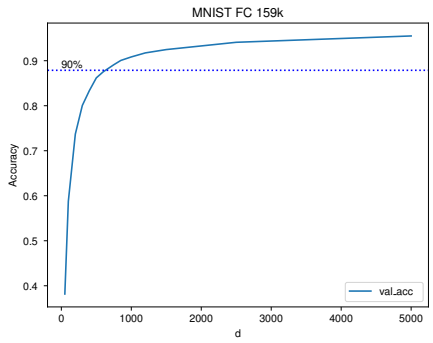


*Figure 1.* MNIST MLP Validation accuracy depending on $d$.

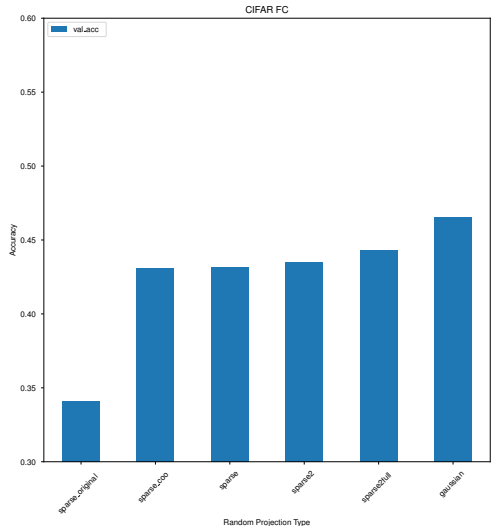


*Figure 2.* Accuracy of random projections with same architecture.

But the experiments (sparse_original in Fig. 2) show that such a projection matrix $P$ leads to bad performance. The problem is that $P$ is designed to project from the $D$-dimensional space to the $d$-dimensional subspace, while during training we actually project from the subspace to the full space. Keeping this in mind, we can adjust for it by swapping $D$ and $d$ in these formulas.
Other experiments were performed to test how (sparse, (1)) compares to (sparse2full, Achlioptas' method) when using $s = 1/\sqrt{D}$, and what happens if we only consider the positive values in our projection matrix (sparse2).

## 3. Experiments

**MNIST MLP** By incrementing the number of dimensions of the subspaces and measuring the accuracy (Fig. 1) we get a smooth line which shows us that by using around 750 parameters we can obtain $90\%$ accuracy of our baseline model, but not the original performance ($0.9763$).
Other plots can be found in (Add.Material), confirming the results for CIFAR as well, both for the MLPs and LeNets.

### 3.1. Orthogonal Random Projection

By orthogonalizing the projection matrix we don't gain much in terms of performance (Add.Material), and it requires a lot of VRAM for the QR decomposition process. For these reasons, mainly because of memory issues, this method wasn't considered in the other experiments.

### 3.2. Sparse Random Projection

PyTorch offers very limited support (PyTorch, b) for sparse matrices, especially for the gradient computations required by the backward pass. Still, it was possible to build an implementation. Multiple representations were considered: dense (baseline), COO, CSR, CSC.

In (Add.Material) there are benchmarks for multiple random projection types, all were tested using the model with most parameters (CIFAR MLP 472k), and $d = 4.5$k.
By not swapping the $D$ and $d$ dimensions (sparse_original), or by only having positive values in our projection matrix (sparse2), the model performs worse.
When we use as matrix elements $\pm\sqrt{D}$ (sparse2full) we unexpectedly get better performance, but by looking at the loss curves we can see that it's noisier with respect to the others, which is a sign of additional distortion introduced by the projection.

### 3.3. Smaller models

MLP models start from 7.9k parameters ($0.916$ acc.) for MNIST and 23.5k ($0.395$ acc.) for CIFAR (Add.Material). It wasn't possible to test models with the same amount of parameters using projection because of the large amount of VRAM required, but the same performance metrics are reached at $d = 1.2$k for MNIST and $d = 6.5$k for CIFAR. The Fully Connected layers aren't efficient in terms of parameters, we have a very large gap between the models with projection and the simple models. The gap in the case of CNNs is lower(more efficient), but it's still there. As we consider less parameters, we can see that the models using projection performs significantly better.

## 4. Conclusion

Although the large amount of memory required and the additional computational cost make this method impractical for production models, it brings some benefits. First of all, more control over the effective number of parameters,

since we disentangle them from the architecture. Another idea is that by using a random projection $P$, we can compress the size of the trained network by storing the seed for generating $P$ and $W$. We can also compare different model architectures on the same problem by using the same number of parameters $d$, the better performing model makes a more efficient use of its parameters.

## References


Achlioptas, D. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66 (4):671–687, 2003. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(03)00025-4. URL https://www.sciencedirect.com/science/article/pii/S0022000003000254. Special Issue on PODS 2001.

Add.Material. Experiments.pdf. https://github.com/adrianrob1/Deep-Learning-AI-Project.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, 2018.

Li, P., Hastie, T. J., and Church, K. W. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pp. 287–296, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150436. URL https://doi.org/10.1145/1150402.1150436.

PyTorch. Qr decomposition. https://pytorch.org/docs/stable/generated/torch.linalg.qr.html#torch.linalg.qr/, a.

PyTorch. Sparse api. https://pytorch.org/docs/stable/sparse.html, b.