Appendix
Evolutionary Algorithms in the Light of SGD: Limit Equivalence, Minima Flatness, and Transfer Learning

This is the appendix to the "Evolutionary Algorithms in the Light of SGD: Limit Equivalence, Minima Flatness, and Transfer Learning" paper currently in review.

Probability of Randomly Sampling a Vector Close to a Target Vector

Assuming the sampling is performed uniformly on a unit sphere, the probability to sample a vector within α degrees from a desired vector follows a regularized Beta-Incomplete function $\frac{1}{2}I_{sin^2\alpha}(\frac{n-1}{2}, \frac{1}{2})$. To visualize this function, we plotted the probability of a random sampling vector to land within α degrees from a reference vector as a function of angle α and dimension n on the Appendix Fig.6.

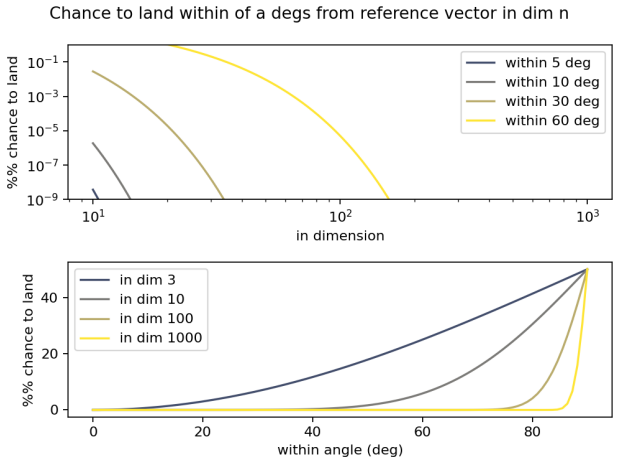

Figure 6: The probability (%) to land within α degrees from the reference vectors as a function of dimensions number for fixed angles and as a function of angles for fixed dimensions

A Trivial Algorithm in the GO-EA Class

A trivial example of an algorithm in GO-EA class is fixed population greedy random search, formally referred to as $(1, \lambda)$-Evolutionary Search in literature and also known as Evolutionary Strategies. The pseudo-code for the exact algorithm we used is presented in Appendix Alg.1. Namely, the version of GO-EA we use does not contain any drift or noise.

Neural Network Architecture

Our ConvNet has 9854 trainable parameters with specific architecture of 784-**[8C5-P2]**-**[16C5-P2]**-**[8C3]**-**[4C3]**-**[196L24]**-**[24L10]**-10. **8C5** denotes a convolution layer with an 8 feature map using a 5x5 filter with stride 1 and padding 2, **P2** is max polling with 2x2 filter and stride 2, and **24L10** is a linear layer mapping from 24 to 10 channels. Every bloc ([]) is followed by a ReLU activation function, and a drop-out normalization is applied to it (10% by default, abbreviated to DO:0.1), except for the last output layer. The input image is partially occluded by using a 10% drop-out on input (DI:0.1). The width of the layers is controlled by two key variables: latent maps in the first layer (LM:8) with subsequent ones following a predefined ratio, and linear width (LW:24), determining the number of neurons in the hidden linear layer.

In SGD mode, the model is trained using cross-entropy loss and SGD optimizer with a learning rate of 0.002 and no momentum, with a batch size of 32 (B:32) and 10 epochs (E:10) unless specified otherwise. The abbreviations above are used in figures, with **x** or **/** denoting the multiplication or division of hyperparameters by a factor (e.g. LWx2) compared to the reference and **:** defining parameter being set to a value (e.g. DO/DI: .0/.0). Finally, **-** means the parameter was set to 0 (e.g. -DO). All figures can be re-generated from code and run records provided, including in greyscale-uniform colormap.

Algorithm 1: Fixed population greedy random search

Input: loss function $\mathcal{L}_{\boldsymbol{\theta}}$, starting parameter θ_0
Parameter: update distance ϵ, population size N, generations M
Output: optimal parameters found θ_{fin}

Initialize $\theta = \theta_0$
for m in [1, .., M] **do**
  for n in [1, ..., N] **do**
    choose random update direction θ_{rand}
    if $\mathcal{L}_{\theta+\epsilon\theta_{rand}} < \mathcal{L}_{\theta}$ **then**
      $\theta = \theta + \epsilon\theta_{rand}$.
    end if
  end for
end for
return $\theta_{fin} = \theta$

An Simple Algorithm in the GO-EA Is Able to Train a ConvNet On MNIST

Given that optimized, tested, and scalable Evolutionary Algorithms in the GO-EA class have already been described and made available, notably by (Such et al., 2017), here we limited ourselves only to a proof-of-concept implementation. Specifically, due to the experimental infrastructure limitation, forcing a sequential sampling, we are sampling a population of 20 individuals per generation, evaluating the fitness (cross-entropy loss) on the whole training part of the MNIST dataset. Given the lack of optimization of code used to implement an instance of GO-EA class, each population sampling round takes slightly over a minute, leading to a slow - comparatively to SGD - performance.

However, as Fig.2 demonstrates, in about 800 updates, our GO-EA instance (edit distance ED:0.01, population POP:20) is able to take our 9854 parameters ConvNet from 10% accuracy on MNIST to 60% accuracy. In a single epoch with base hyperparameters, SGD performed over 1800 updates with an accuracy of only about 15% on the training set and will need 1800 more to get to 85% in the following epoch, albeit at a lower learning rate. Unlike SGD, our instance of GO-EA did not perform back-propagation, and the population is fully and efficiently scalable through random seed sharing, as explained by (Such et al., 2017).

While increasing the population size is a possible way to increase the speed of convergence, the behavior of other hyperparameters seems to be negligible, as a rapid exploration of parameter

Figure 7: Variation of the speed of convergence of our base ConvNet on MNIST dataset based on the hyperparameters of our implementation of GO-EA class algorithm

space over 30 generations near 60% accuracy suggests in Appendix Fig.7.

Model Validity

The base model we used is able to achieve an error rate below 2% on the validation set after 20 epochs of training and about 2.5% in 10 epochs. Our approach to measuring the ruggedness of loss landscape and minima flatness was consistent with prior measures and recapitulated the results by Li et al. (2018), which we focus on due to its applicability in the non-differentiable setting. Specifically, flatter minima resulted from more overparametrized models, dropout normalization, and smaller batches (cf Appendix Figs. 10, 9, 11). We also confirmed that the total batch number in training was not affecting the minima flatness (cf. Appendix Fig.12).

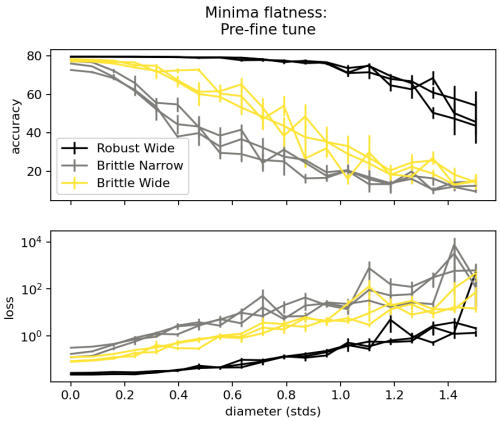

Figure 8: 8 class minima flatness for archetype models. Due to how accuracy is calculated, the best-expected model performance is 80%. Error bars derived from sampling replicas, lines are full independent replicas.

.

Minima Flatness Profiles

In order to obtain minima flatness profiles, several models with identical hyperparameters are trained independently. As described by Li et al. (2018), we measure the minima flatness or the local smoothness by selecting a random vector from an N-dimensional Gaussian distribution with covariance defined by the Identity matrix scaled for the standard deviation of activations in a given layer. Once a vector is selected at random, the model stability is recorded along the axis of its extension, performing a "sweep" of the fitness landscape in the neighborhood of the model parameter vector θ.

In the diagrams of the minima flatness we present, the x axis is in standard deviations of activations. For completeness, we present the degradation of both loss and accuracy, and for precision, for each parameter vector, we sample at least 5 independent filter-wise normalized random vectors, resulting in error bars when the error at a given distance for them diverges (sampling replicas). Each line corresponds to a completely independent parameter vector from a model trained from scratch with the same hyperparameters (full replicas).

Dropout, Overparametrization, Small Batches, but Not Length of Training Contribute to Flatter Minima

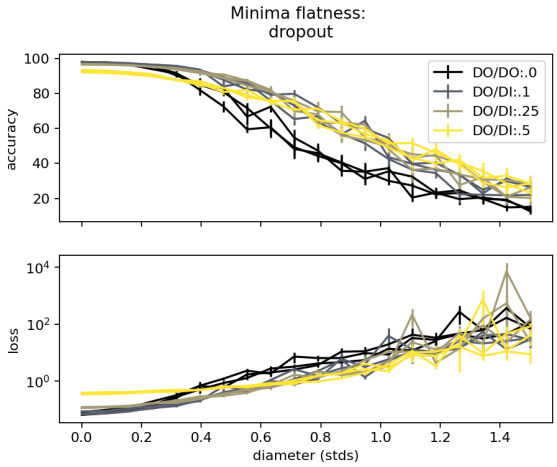

Figure 9: Effect of dropout and input dropout on minima flatness

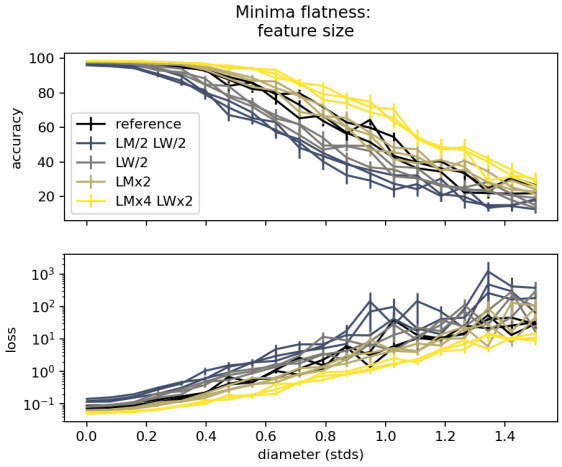

Figure 10: Effect of under- and over- parametrization of the ConvNet on the minima flatness

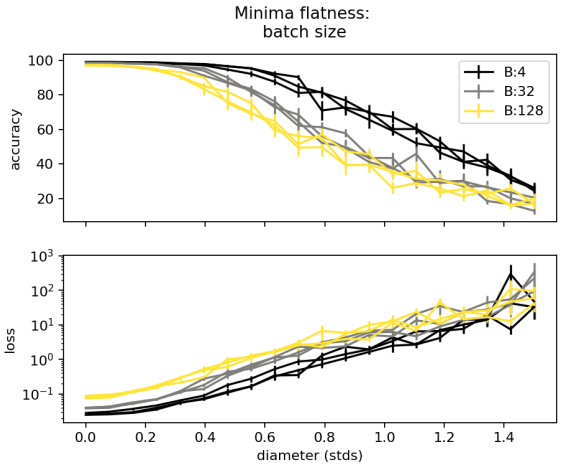

Figure 11: Effect of the small and large batch sizes on the minima flatness

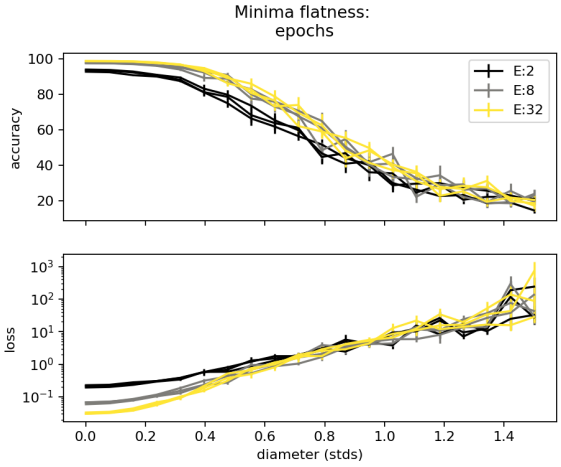

Figure 12: Larger number of updates with smaller batch size cannot explain flatter minima, given the length of training does not affect the minima flatness

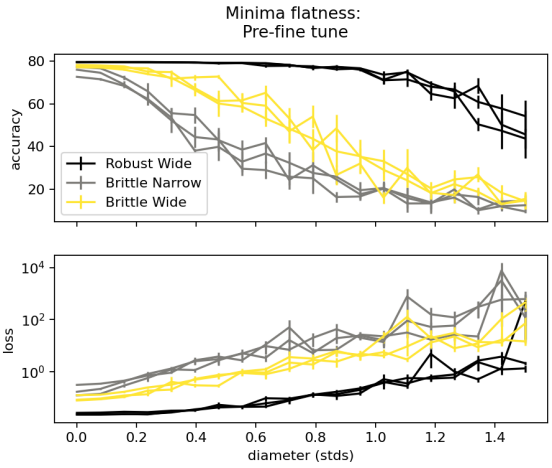

Figure 13: Minima flatness for 3 archetypes before transfer learning

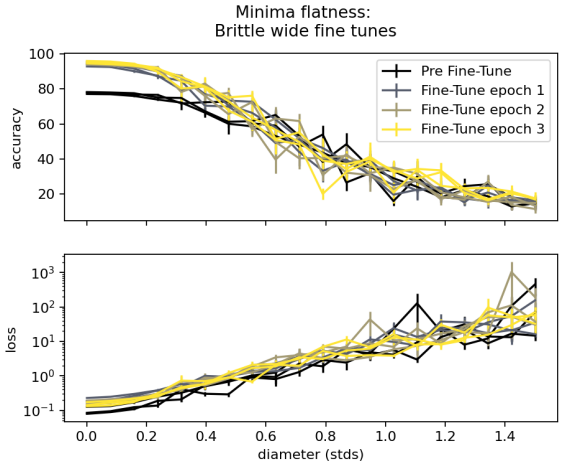

Figure 14: Change in minima flatness for the Brittle Wide archetype during transfer learning

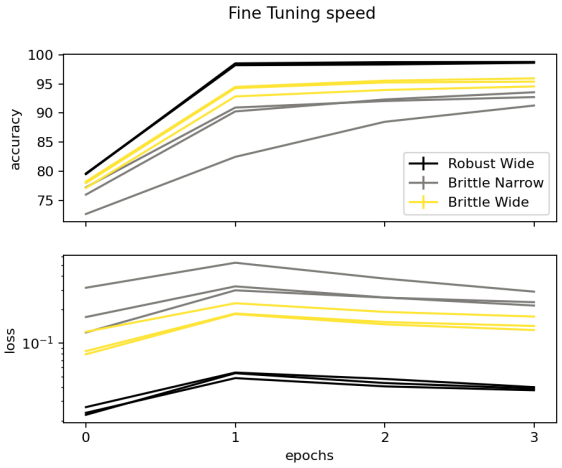

Figure 15: Change in accuracy and loss over epochs of transfer learning for the three archetype models. Minima flatness does not seem to affect transfer learning speed. Each line is a full independent replicate.

Experimental Setup and Computation Time

All the methods are evaluated on a workstation equipped with Intel Core i9-9900K (8 cores/16 threads CPU), 64 Gb of RAM clocked at 2666 MHz, 2 TB NVME M.2 SSD, and an RTX 3080 graphics cards, running an Ubuntu 20.04 LTS distribution. The evaluations were performed within a Docker container, Docker Community Edition, version 20.10.12. The code used Miniconda version 4.12.0 and Python 3.9 with CUDA version 11.3.1. More detailed information is available in the `requirements.txt` found in the code repository shared with this Appendix, https://github.com/chiffa/ALIFE2023_GOEA-SGD.

The total compute time for re-running experiments, except for the GO-EA training trace, is under 2 hours. GO-EA, due to non-optimized deterministic random update vector derivation from seed, shuttles data between CPU and GPU for every individual in every generation. Due to the overhead introduced, the GO-EA trace presented in Fig.2 took approximately 24 hours.

No hyperparameter optimization outside model versions presented here were presented. A common learning rate of 0.02 for training ConvNets on MNIST was divided by 10 to accommodate very small batches, resulting in 0.002. The same rate was also used for GO-EA training.