PROJECT REPORT

Dario Filatrella, Rocco Giampetruzzi, Rolf Minardi, Mattia Scardecchia

Introduction

For our project, we decided to investigate the influence of several properties of the minimizers found by gradient descent during pre-training on the effectiveness of finetuning the resulting networks on a downstream task. In fact, nowadays it is common practice in many real-world applications of deep learning to finetune a very large model trained on an enormous corpus of data to the specific task at hand. It is therefore interesting to ask what properties of the final pretraining configuration are most relevant for finetuning performance and speed, which is also critical in resource-constrained scenarios.

More concretely, we decided to study the problem of image classification. The properties of the minimizers that we considered are the generalization error on the pretraining task, the ability to classify randomly corrupted inputs, the resilience to adversarial modifications of the images and the flatness around the minimizer in the loss landscape. The research question that we set out to investigate is the following: do more robust pretraining solutions, as measured by the metrics we mentioned, allow for faster and/or more effective finetuning? The rationale behind this is that more robust networks might have learnt to extract features that are more generally applicable, and may provide a better starting point to learn a new but related task.

In order to try and answer this question, we trained a convolutional neural network on a subset of Imagenet in different ways, to obtain networks that differ from one another in terms of the aforementioned properties. Then, we finetuned on the simpler CIFAR-10 dataset.

In the next section, we will go in detail about the methods that we used and the experiments that we carried out.

Methods

@ Data

For pretraining, we used 65000 images belonging to 50 different classes from the Imagenet dataset. The classes were randomly chosen from the Imagenet100 subset *[1]*.

Since absolute performance is incidental to our purposes, and since the Imagenet dataset is quite clean being it a standard benchmark in ML, we performed limited preprocessing on the images: we linearly squeezed all channels between 0 and 1, and we brought all images to a single resolution 112x112.

For finetuning, we used the whole CIFAR-10 dataset, processed in such a way to be suited for training with the networks resulting from pretraining.

@ Networks

For all our experiments, we used the ResNet18 architecture *[2]*. This relatively simple network was powerful enough to capture the complexity of our pretraining task, while remaining manageable computationally.

@ Pretraining

As we mentioned, our overall goal is to assess the importance of features such as generalization capabilities and robustness to perturbations of weights and inputs for finetuning performance. To this end, we performed the pretraining in three different ways, expecting that they should produce different results when it comes to those features. In all cases we carried out the optimization for a sufficient number of epochs so that the networks could learn the whole training set to a very high accuracy (95%).

All three optimization methods are based on the Adam optimizer and optimize the cross-entropy loss. The reason for this choice, as opposed for instance to SGD with momentum, is that due to its adaptive nature Adam can reach reasonably good performance without excessive finetuning of the learning rate schedule, which is instead crucial for SGD and would have been very expensive given our limited computational budget. Specifically, the three methods are:

- **Vanilla *Adam*.** We simply used *Adam* to optimize the cross-entropy loss. As for the hyperparameters, we tuned the learning rate and the batch size to maximize the generalization error on a held-out validation set. For batchnorm to perform well, it was important to have a batch size large enough that batch statistics are representative of the data (we ended up using 256). Given that our computational budget is limited, for the rest of the parameters, whose choice we estimated would have a lesser impact, we kept the default values.
- ***Adam* with corrupted images**. We used *Adam*, with the same hyperparameters as in the vanilla case, but training on randomly corrupted images. This is achieved by perturbing each channel of each image with additive Gaussian noise of a given intensity, every time that the image is presented to the network for training. We tried different standard deviations, and we selected one that is large enough to induce robustness without preventing the network from being able to learn the training set.
- ***Stochastic Weight Averaging*.** Using as base optimizer *Adam* with the same hyperparameters chosen for the vanilla case, we used *Stochastic Weight Averaging [3]* to bias the optimization towards the middle of flat minima in the loss landscape. To achieve this, we trained with *Adam* until a predefined accuracy objective (0.95) was reached; then, we started saving the configuration of the model at regular intervals to be used for averaging. Usually, a high learning rate is used in this second phase *[4]*, to allow the optimization to keep exploring the set of high-performing networks instead of simply converging to a single solution. For this second phase we used a constant learning rate, which we tuned to optimize test error, with a brief linear annealing phase to smooth out the transition.

In all three methods we used weight decay to regularize the training. These methods produced different networks after pretraining on Imagenet. To understand a bit better how different they are, we computed the euclidean distance between the weight configurations, both including and excluding the batchnorm layers, since these have a different typical magnitude than the others and the distances between them are harder to interpret. In addition, we performed the following experiment. For each pair of networks after pretraining, starting from one of them, we progressively perturbed its weights in the direction of the weights of the other, effectively moving along the line joining the two networks in weight space. In doing so, we kept track of how the cross-entropy and accuracy changed, obtaining a picture of a ‘slice’ of the landscape between the two solutions.

@ Properties of minimizers

In addition to the generalization error on the pretraining task, we wanted to investigate the influence on finetuning performance of several other metrics. Specifically, we considered:

- **Classification of noisy inputs.** We computed the cross-entropy and accuracy of the network when presented with noisy images to classify. First, we considered the case of Gaussian additive noise, just like we described for the pretraining. However, for one of the three methods this type of noise was seen during training, giving the networks trained that way an unfair advantage over the others in this respect. For this reason, we also considered a type of noise that was unseen by all the networks involved: we masked a randomly selected square in the image, substituting its content with black. In both cases, we used noise of varying intensities, and we repeated the experiments independently many times to reduce the effects of the inherent stochasticity.

- **Resilience to adversarial attacks.** We evaluated the robustness of the networks to adversarial attacks in the form of targeted modification of the input images. To achieve this without having to train a separate model, we employed a method called *Fast Gradient Sign Attack [5]* : given an image and the corresponding label, we compute the gradient of the cross-entropy with respect to the input image; then, having fixed a step size epsilon, we perturb each component of the input by epsilon times the sign of the partial derivative with respect to that component. If the loss function was a linear function, this would represent the most disruptive perturbation of the image with a fixed l1 norm epsilon. In reality, for a step size not too large, this method can be thought to provide a good approximation to the optimal perturbation. Also in this case, we repeated the attack for a range of values of epsilon.
- **Flatness in the loss landscape.** Given a network, we computed its local energy profile, a measure of the flatness of the loss landscape around the solution *[6]*. This is done by perturbing each weight by a multiplicative random term 1 + **σ**, where **σ** is sampled independently from a gaussian for each weight. Varying the standard deviation of the noise and repeating the experiment many times for each noise intensity provides a good aggregate measure of how fast the loss increases moving away from the reference configuration.

We compared all of the above metrics for the networks obtained through the three different pretraining methods.

@ Finetuning

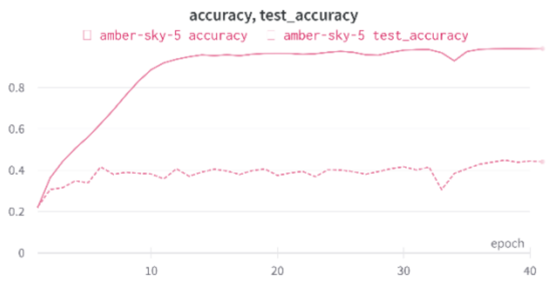
After the pretraining, we finetuned each of the resulting networks on the CIFAR-10 dataset. We used the *Adam* optimizer for all the networks with the same choice of hyperparameters, tuning learning rate and batch size and keeping the default values for the rest. Since the finetuning task is less computationally demanding than the pretraining, we restarted the training from each pretrained network multiple times to obtain statistically significant results.

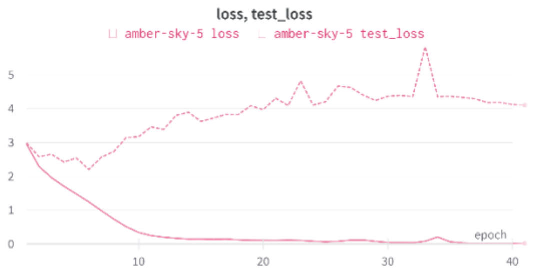
For each run, we tracked cross-entropy and accuracy both on the training and on the test set. Furthermore, we recorded the number of epochs necessary to reach a predefined level of accuracy (95%) on the training set, as a way to measure the speed of training starting from a given configuration. Finally, we kept track of the evolution of the Euclidean distance of the network configuration from the initial one, to understand how far from the initial configuration the network has to wonder in order to learn the finetuning task.

Results

@ Pretraining

In the pretraining task, all three models were able to learn the training set to a very high accuracy (above 95% with 50 classes). However, the generalization performance was not as good (between 40-47% accuracy), signalling that the models are overfitting. Two factors contributing to this might be the size of the dataset, which is not very large given the complexity of ResNet18, and the lack of dropout, a regularization technique often employed in computer vision, in the ResNet architecture. Below you can see the behaviour of the cross-entropy and accuracy during pretraining with vanilla adam. The curves obtained with the other methods have the same shape. Interestingly, while the test loss hits a minimum early and then starts increasing, the test accuracy remains roughly constant after the peak. The noise in the curves is likely due to the adam optimizer, which can be somewhat unstable.

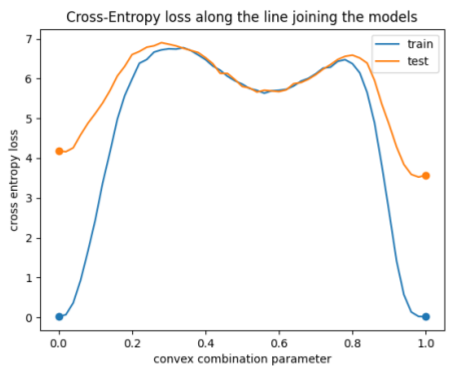


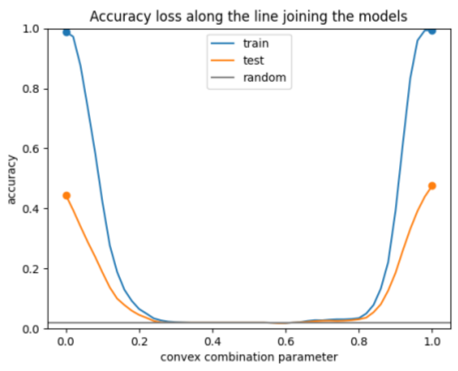


The method that did best in terms of generalization error was SWA (47.5% acc, 3.64 loss), followed by vanilla adam (44.5% acc, 4.1 loss) and adam trained with noisy inputs (40% acc, 4.8 loss).

Looking at the spatial relationships between the three solutions, we found that the distances between pairs of solutions are very close in magnitude to the typical norm of a single solution. This is also true considering separately batchnorm and non-batchnorm weights. These facts seem to indicate that the three solutions are relatively far apart in weight space. This however should be taken with a grain of salt. First of all, distances tend to lose their intuitive meaning and interpretation in high-dimensional spaces due to the so-called curse of dimensionality. Second, because of the numerous combinatorial symmetries intrinsic to a multi-layer perceptron architecture, being far apart in weight space does not necessarily mean implementing very different functions.

Traversing the loss landscape along the line joining each pair of solutions seems to confirm that the three methods found solutions lying in different minima (although this method is blind to anything that happens outside of that straight line – one could imagine there being a mountain between two solutions, but with a low-loss continuous path going around it). Below we show one of the three plots, moving from vanillla adam to swa; the other two are qualitatively very similar.





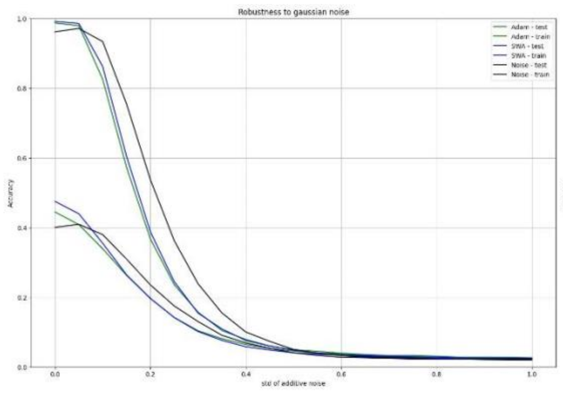
@ Robustness of minimizers

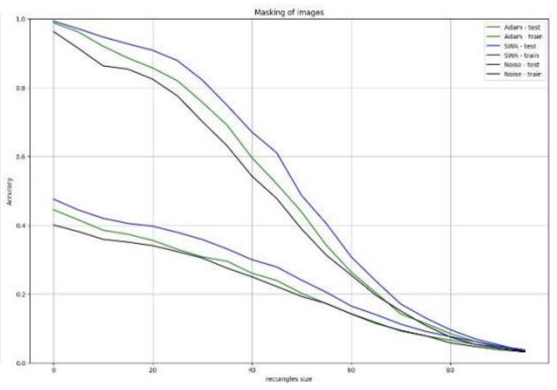
For all three pretrained networks, we evaluated their robustness to perturbations of inputs and of weights, as explained in the methods section. We did so both using the test set and using a stratified subset of the training set. Generally, the results obtained by evaluating on the training and test set agreed.

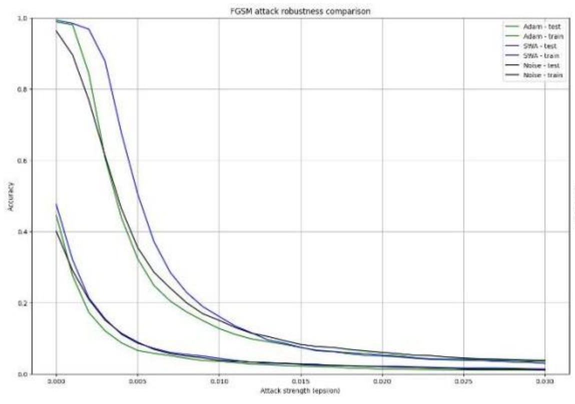
With additive Gaussian noise, unsurprisingly, the network trained with noisy adam performed significantly better than the other two. To some extent, this was expected because this is the type of noise the model saw during training. At the same time, however, the standard deviation of the noise used for training was 0.05, while here we see large improvements up to around 0.4, which does require some ability to generalize.

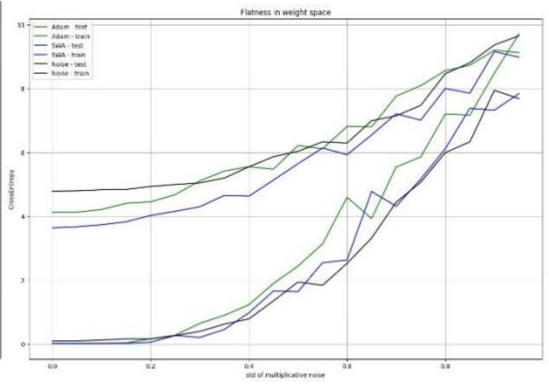
With randomly masked images, instead, the model trained with SWA was the best, followed by vanilla adam and noisy adam. SWA was also more resilient to adversarial attacks than the other two networks, by a significant margin.

Finally, the local energy is a bit harder to interpret. In fact, despite a high number of independent perturbations per noise intensity (25), the profile seems a bit noisy. Given how large the network is, it was impractical to do even more iterations. Anyways, close inspection of the graph suggests that the landscape around the solutions found by SWA is flatter, although slightly, than for vanilla and noisy adam. This is not surprising given that the rationale behind SWA is precisely that of targeting flat regions of the loss landscape.





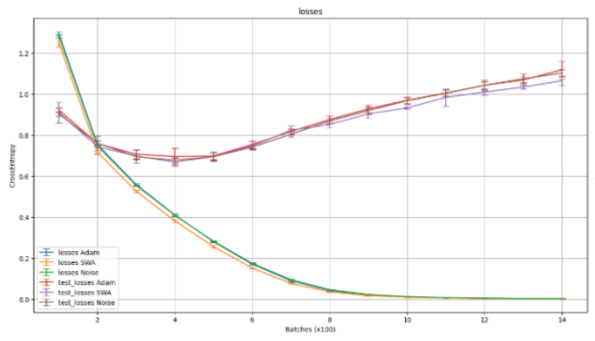


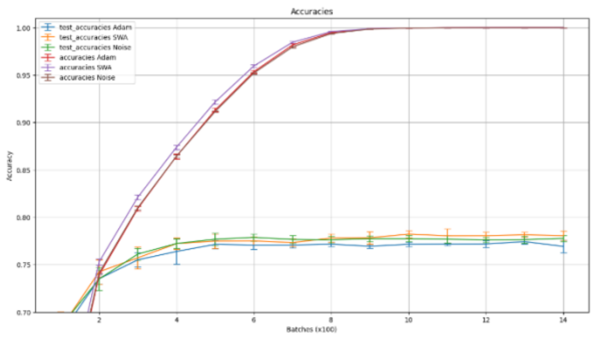


@ Finetuning

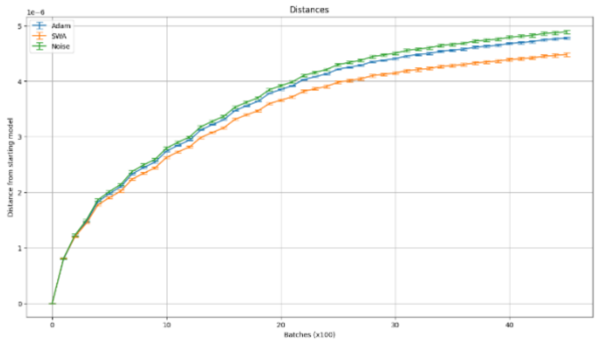
For the finetuning task, as expected, tuning the learning rate of adam showed that using a small learning rate is beneficial (plots are with lr 0.0001). All three networks managed to perfectly learn the training set in all runs, and the time required to do so did not vary significantly between different pretrained networks.

We did observe a small difference in generalization performance, averaging over 5 independent runs. The model pretrained with SWA did slightly better than noisy adam, but the difference is not significant once error bars are accounted for. On the other hand, models pretrained with SWA and noisy adam consistently obtained around 1% higher test accuracy than those pretrained with vanilla adam. Below you can see the evolution of loss and accuracy during training for the three networks, averaged over multiple runs.





Another difference was found in the time evolution of the Euclidean distance between the network configuration and the initial one: the network pretrained with SWA moved slightly less from the initial weights compared to the others, consistently across different runs.



Conclusions

We pretrained a resnet18 model in three different ways, and we evaluated the robustness of the resulting networks to perturbations of weights and of inputs (random and adversarial). We found that two of the three models (SWA and noisy adam) were better than the third model in at least one respect, and this was reflected in a slightly superior generalization performance in the finetuning task. Interestingly, this was despite the fact that noisy adam obtained a significantly worse generalization error than vanilla adam in the pretraining task. We also found that the network lying in the flatter region of the pretraining landscape (SWA) had to move less in weight space in order to learn the finetuning task.

Our study has some limitations. An important one is that, compared to what is usually done in applications, our pretraining task uses a somewhat limited amount of data. In fact, given our resources, tuning hyperparameters on a more complex task would have been infeasible. However, it would be interesting to carry out experiments in a more realistic scenario, pretraining on a large corpus of data and, accordingly, finetuning on a more challenging task. Using more data could also help mitigate the strong overfitting that we observed in the pretraining.
For similar reasons, we limited our exploration to three different ways of performing the pretraining. It would be interesting to try other flat minima optimizers (e.g. SAM, rSGD) or to employ different types of noise than just additive gaussian during training, to have even more different networks in terms of robustness to start the finetuning from.

An important remark is that, because of the way our experiments were designed, they can only detect correlation between various robustness metrics and finetuning performance, not causation. In fact, there could potentially be some confounding variables hidden in the different ways of carrying out the pretraining. To detect causality, one would likely need to restrict to a single pretraining algorithm, and vary as few parameters as possible (e.g., restrict to noisy adam and train with a large range of noise intensities).

There are a few interesting directions for further research. First, we could repeat our investigation using a different architecture (e.g., EfficientNet) or in a different context (e.g., an NLP task), to understand how widely applicable our conclusions are. Also, we could explore different ways of doing finetuning, freezing some of the early layers or even freezing the whole convolutional structure and training a logistic regression on top of the features it extracts.

To continue along the line of exploring the relationship of the various pretrained model with each other within the loss landscape, we could investigate the behaviour of the loss along arbitrary curves joining two solutions. To do so, we could optimize a weighted sum of the loss and the distance from one of the two configurations, starting from the other. This could allow to find, if they exist, continuous paths of low loss joining the two solutions, providing further insight on the geometry of the landscape.

Finally, it could be interesting to study a related scenario, that of continual learning, and investigate the influence of the robustness to perturbations of inputs and weights after learning part of the dataset on the ability to learn more data while avoiding the so-called 'catastrophic forgetting' phenomenon.

Bibliography

[1] ImageNet100 | Kaggle

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015). Deep Residual Learning for Image Recognition. https://doi.org/10.48550/arXiv.1512.03385

[3] Diederik P.Kingma, Jimmy Ba (2014). ***A Method for Stochastic Optimization.***
https://doi.org/10.48550/arXiv.1412.6980

[4] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, Andrew Gordon Wilson (2019). Averaging Weights Leads to Wider Optima and Better Generalization.
1803.05407.pdf (arxiv.org)

[5] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy (2015). Explaining and Harnessing Adversarial Examples.
1412.6572.pdf (arxiv.org)

[6] Fabrizio Pittorino, Carlo Lucibello, Christoph Feinauer, Gabriele Perugini, Carlo Baldassi, Elizaveta Demyanenko, Riccardo Zecchina (2020). Entropic gradient descent algorithms and wide flat minima.
https://doi.org/10.48550/arXiv.2006.07897