DEEP INPUT OPTIMIZATION: COMPARISON ANALYSIS OF VARIOUS OPTIMIZATION ALGORITHMS UNDER DIFFERENT INPUT PARAMETRIZATIONS

[1]HENRIK SERGOYAN, [2]GRIGOR BEZIRGANYAN

[1,2] Department of Mathematics, Technical University of Munich, 85748, Garching, Germany
Email: [1]henrik.sergoyan@tum.de, [2]grigor.bezirganyan@tum.de

Abstract - A number of important deep learning (DL) applications, such as adversarial machine learning, single-view reconstruction and AI-driven design are exploiting differentiability of deep neural networks to perform input optimization.
In this paper, we address one of the main weaknesses reported in the setup of input optimization algorithms getting stuck in local minima. For our experiments we consider a toy-example of image optimization w.r.t. pretrained MNIST image classification deep neural network, i.e. the task is to find a maximizer of the target MNIST class.As the main contribution of this work, we report comparative results for various optimization algorithms under different input parametrizations used to solve the aforementioned problem.

Keywords - Deep Learning, Input Optimization, Autoencoder, Image Classification

I. INTRODUCTION

With the advent of deep learning, a number of important DL applications are addressing the problem of optimization w.r.t input of deep neural networks. For instance, works [1,2,12] are using constraint optimization to produce targeted adversarial attacks.Furthermore, in geometric deep learning, input optimization plays a crucial role, as tasks such as single-view reconstruction [8, 11] or AI-driven design [7, 9,10] usually require 3D surface deformation to minimize the objective function.To perform surface mesh optimization robustly, a common practice is to rely on low-dimensional parameterizations that are either learned [4, 5,6,7] or hand-crafted [3,9,10]. However, as reported by the authors of [7], such kind of optimization in a complex 3D space is prone to getting stuck in local minima, leading to incomplete optimizations.

In this work, using a toy problem of MNIST [13] image classification, we are simulating a generalized input optimization task. In the next section we briefly describe the MNIST classification model we trained. Afterwards, we report comparative results for various optimization algorithms under 3 different parameterization techniques: at first we study unparametrized high-dimensional optimization, then we introduce a learnt prior by training an autoencoder with basic MSE loss, and finally we regularize the latent space of our autoencoder.

II. MNIST IMAGE CLASSIFIER

We trained ResNet-18 model [14] $f: \mathbb{R}^{28\times 28} \rightarrow \{0,1,..,9\}$ on the train set of MNIST for 20 epochs using Adam optimizer [15] with default parameters andlearning rate equal to 0.0003.It achieved 99.02% prediction accuracy on the test set. In the subsequent experiments the trained weights were frozen.

III. EXPERIMENTS

The general idea of our experiments was to study optimizers' ability of escaping local minima in the problem of input optimizations. For that purpose we optimized a randomly initialized input image $I \in \mathbb{R}^{28\times 28}$ to maximize the predicted probability produced by *f* for the given target digit *d*.
To achieve that, we computed the corresponding cross-entropy (CE) loss of *d*on the output of *f(I)* and performed backpropagation all the way back to *I*. So, for the digit *d* the loss was as follows:

$$CE(x, d) = -\log(Softmax(x)[d]),$$

where x is the vector of logits returned by *f(I)* and $Softmax(x)[d] = \frac{e^{x_d}}{\sum_i e^{x_i}}$.
For each of the approaches described below we compared different optimizers, namely SGD, SGD with momentum [19],Adam [15], Adadelta, [16] RMSProp [18] and Adagrad [17].
As a comparison metric we used the average optimization loss achieved by the algorithms over 10 runs. For each algorithm we carefully finetuned learning rate by running 10 independent optimizations with different random initialization for each target digit and chose the learning rate with the lowest average loss.For optimizers that had additional tunable parameters, like Adam, we used default values as they usually remain unchanged in most of the experiments performed by the research community. For SGD with momentum we used momentum coefficient of 0.9.Moreover, the default values used in PyTorch are known to usually be at least suboptimal. We used the following grid for learning rates: [0.0001, 0.0002, 0.0005, ..., 0.1, 0.2, 0.5, 1, 3, 5, 7, 10, 20, 50, 70, 100, 150, 200, 300, 500, 1000, 1500] (yes, in some cases the optimal learning rate were that big due to vanishing gradient!).

A: Approach 1: Pixel-wise optimization

As our first experiment, we tried to optimize the image pixels directly.In other words, each pixel of the input image I was directly optimized with respect to the loss $CE(f(I), d)$. An optimization example for digit 8 is illustrated in Figure 1.

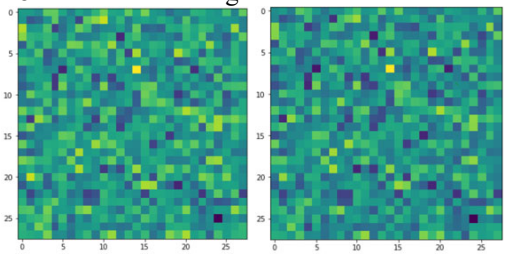


Figure 1: Optimizing noise to digit 8 using Adam

On the left is Input image which ResNet-18 model predicted 2 with 51.7% probability. On the right is Optimized image which ResNet-18 model predicted 8 with 99.9% probability.

Regardless of the chosen optimizer (e.g. Adam, SGD), this approach optimized randomly initialized image to meaningless noise, exploiting adversarial vulnerability of the network. However, finding such examples was not our goal.Inspired by [7, 5,6,4] we lowered the dimensionality of the optimization space by learning a parameterization of input images. We trained an autoencoder to learn the semantics of hand-written numbers and used this as a prior, thus constraining the optimization space to realistic images. We cover this in more detail in the next section.

B: Approach 2: Optimization with latent code space

We trained a simple autoencoder consisting of an encoder$enc(I) = c \in \mathbb{R}^{32}$ and a decoder$dec(c) = I'$.In this approach, we ran Adam optimizer with default hyper-parameters and batch size equal to 128 for 50 epochs.As a reconstruction loss we usedMean Squared Error (MSE) of an input and decoded images. The model achieved MSE=0.504 on the test set. Examples of reconstructed images from the test set are presented in Figure 2.

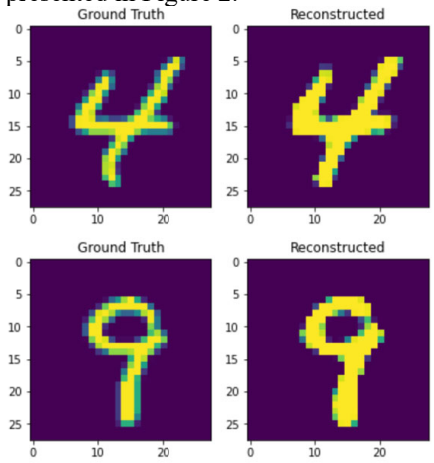


Figure 2: Reconstructing images using autoencoder

After training the autoencoder, we froze its weights and performed an optimization of the learned code. Starting from a random initialization of c, we maximized the predicted probability produced by *f(dec(c))* for the given target digit *d*.

In order to make the initializations favorable, we sampled codes c from a normal distribution with mean equal to the empirical mean of codes calculated on the training set and standard deviation (std) equal to $0.1\sigma$where σ is the empirical std of codes calculated on the training data. Both mean and std were computed coordinate-wise.

The results for different optimizers are presented in Table 1 (see Approach 2) while examples of optimized images are shown in Figure 3.

Optimizer/ Approach	Approach 2	Approach 3
SGD	(lr=70, loss=0.021)	(lr=0.002, loss=0.028)
SGD+ momentum	(lr=10, loss=0.030)	(lr=0.0005, loss=0.031)
Adam	(lr=3, loss=0.027)	(lr=0.01, loss=0.024)
Adadelta	(lr=1000, loss=0.012)	(lr=2, loss=0.036)
RMSProp	(lr=0.5, loss=0.043)	(lr=0.005, loss=0.016)
Adagrad	(lr=5, loss=0.083)	(lr=0.05, loss=0.016)

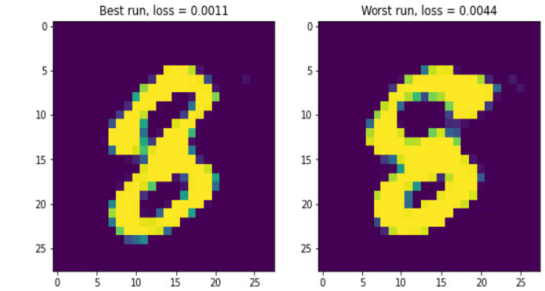
Table 1:Hyper parameter tuning results

C: Approach 3: Optimization with regularized codes

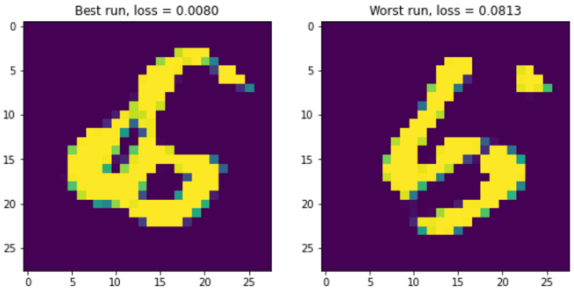
This approach is very similar to the previous one, except for one major difference: in addition we penalized the *L2* norm of the code c (output of the encoder), leading to the following loss function:

$$\boldsymbol{loss(I) = \frac{1}{n}\left\|I - dec\left(enc(I)\right)\right\|_2^2 + \frac{1}{m}\|enc(I)\|_2^2,}$$

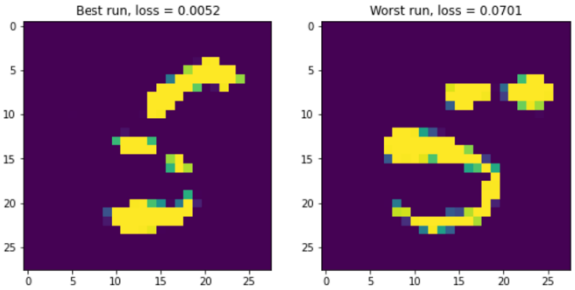
Where $n$is the dimensionality of I, *enc* and *dec*are encoder and decoder of the autoencoder respectively, and $m$is the dimension of $c = enc(I)$. This allowed us to ensure that codes of real images were grouped around the zero vector, so the topology of the code space for real digits was similar to a ball. Our hope was that this would allow optimizers to find realistic digits more often. The other details were exactly the same as for the previous approach. The results for the different optimizers and examples of obtained images are presented in Table 2 (see Approach 3) and Figure 4 respectively.



a) the result for Adagrad(lr=5) for digit 8



b) the result for Adam(lr=5) for digit 6



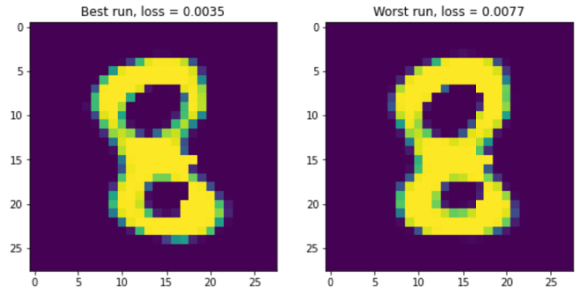
c) the result for SGD with momentum (lr=10) for digit 5

Figure 3: Examples of digit optimizations with autoencoder

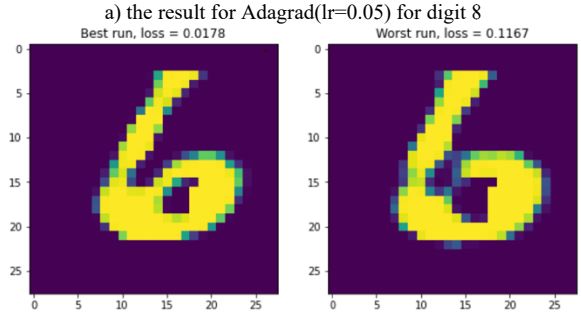
As you can see, the results are much better with code regularization, especially from the visual perspective. It means that our hypothesis about better code space topology was correct. According to our empirical results, in the given experimental setup, RMSprop and Adagrad are the best optimizers among others. It is also worth mentioning that unlike our expectation, SGD performed surprisingly well - better than Adam and Adadelta. In fact, we were anticipating SGD to have the worst performance because we optimized only one input rather than the whole dataset, hence it was impossible for SGD to avoid local minimas using stochasticity of batches (actually, SGD is GD here because dataset size is 1). However, the results refuted our expectation, subsequently creating a room for investigating the theoretical reasons behind this. Lastly, we obtained that momentum did not improve SGD performance.

IV. CONCLUSION

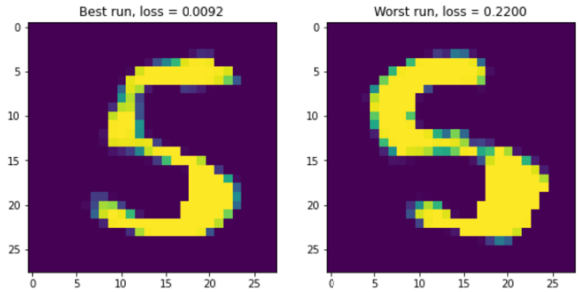
In summary, we looked at different input parametrizations, and for each case explored the behaviour of various optimization algorithms on the task of optimizing the given input image to be similar to the target image. We found that directly optimizing the pixels of the input image does not produce valuable results and leads to a meaningless noise, exploiting adversarial vulnerability of the neural network. We further showed that introducing a learnt prior by training an autoencoder, and furthermore regularizing the latent space, is a useful parametrization with good topology. Our empirical results showed that the best optimizers for this task are RMSProp and Adagrad.



a) the result for Adagrad(lr=0.05) for digit 8



b) the result for Adagrad(lr=0.01) for digit 6



c) the result for SGD with momentum (lr=10) for digit 5

Figure 4: Examples of digit optimizations with regularized autoencoder

REFERENCES

[1] Carlini N., & Wagner D. (2018). Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. 2018 IEEE Security and Privacy Workshops (SPW), 1–7.

[2] DongY., Liao F., PangT., Su, J.Zhu, X. Hu, X.& LiJ. (2018, June). Boosting Adversarial Attacks With Momentum. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[3] Remelli E., Tkach A., Tagliasacchi A., & Pauly M. (2017). Low-Dimensionality Calibration Through Local Anisotropic Scaling for Robust Hand Model Personalization. ICCV.

[4] Bagautdinov T., Wu C., Saragih J., Fua P., & SheikhY. (2018). Modeling Facial Geometry Using Compositional VAEs. CVPR.

[5] Blanz V., & Vetter T. (1999). A Morphable Model for the Synthesis of 3D Faces. SIGGRAPH.

[6] Park J. J., Florence P., Straub J., Newcombe R. A., & Lovegrove S. (2019). DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. CVPR. doi:10.1109/CVPR.2019.00025

[7] Remelli E., Lukoianov A., Richter S. R., Guillard B., Bagautdinov T., Baque P., &Fua P. (2020). MeshSDF: Differentiable Iso-Surface Extraction. arXiv preprint arXiv:2006. 03997.
[8] HendersonP., & FerrariV. (2020). Learning single-image 3D reconstruction by generative modelling of shape, pose and shading. IJCV, 128(4), 835–854.
[9] Umetani N., & Bickel B. (2018). Learning three-dimensional flow for interactive aerodynamic design. ACM Transactions on Graphics (TOG), 37(4), 1–10.
[10] Baqué P., RemelliE., Fleuret F., &Fua P. (2018). Geodesic Convolutional Shape Optimization. ICML.
[11] KanazawaA., Tulsiani S., EfrosA. A., & Malik J. (2018). Learning Category-Specific Mesh Reconstruction from Image Collections. ECCV.
[12] KosJ., Fischer I., & Song D. (2018). Adversarial Examples for Generative Models. 2018 IEEE Security and Privacy Workshops (SPW), 36–42.
[13] LeCun Y., & CortesC. (2010). MNIST handwritten digit database. Available: http://yann.lecun.com/exdb/mnist/
[14] He K., Zhang X., Ren S., & Sun J. (2015). Deep Residual Learning for Image Recognition. CoRR, abs/1512.03385. Available: http://arxiv.org/abs/1512.03385
[15] Kingma D., & BaJ. (12 2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.
[16] ZeilerM. D. (2012). ADADELTA: An Adaptive Learning Rate Method.CoRR, abs/1212.5701. Available: http://arxiv.org/abs/1212.5701
[17] DuchiJ., HazanE., & Singer Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12(61), 2121–2159. Available: http://jmlr.org/papers/v12/duchi11a.html
[18] Graves A. (2013). Generating Sequences With Recurrent Neural Networks. CoRR, abs/1308.0850. Available: http://arxiv.org/abs/1308.0850
[19] Sutskever I., Martens J., Dahl G., & Hinton G. (17-19 Jun 2013). On the importance of initialization and momentum in deep learning. in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research,S. Dasgupta & D. McAllester, Proceedings of the 30th International Conference on Machine Learning (pp. 1139-1147). Available: http://proceedings.mlr.press/v28/sutskever13.html

★★★