Derivative-Free Optimization of Neural Networks using Local Search

Ahmed Aly
Computer Engineering Program
University of Virginia
Charlottesville, VA
aaa2cn@virginia.edu

Gianluca Guadagni
Applied Math Program
University of Virginia
Charlottesville, VA
gg5d@virginia.edu

Joanne B. Dugan
Computer Engineering Program
University of Virginia
Charlottesville, VA
jbd@virginia.edu

***Abstract*—Deep Neural Networks have received a great deal of attention in the past few years. Applications of Deep Learning broached areas of different domains such as Reinforcement Learning and Computer Vision. Despite their popularity and success, training neural networks can be a challenging process. This paper presents a study on derivative-free, single-candidate optimization of neural networks using Local Search (LS).**

LS is an algorithm where constrained noise is iteratively applied to subsets of the search space. It is coupled with a Score Decay mechanism to enhance performance. LS is a subsidiary of the Random Search family.

Experiments were conducted using a setup that is both suitable for an introduction of the algorithm and representative of modern deep learning tasks, based on the FashionMNIST dataset. Training of a 5-Million parameter CNN was done in several scenarios, including Stochastic Gradient Descent (SGD) coupled with Backpropagation (BP) for comparison. Results reveal that although LS was not competitive in terms of convergence speed, it was actually able to converge to a lower loss than SGD. In addition, LS trained the CNN using Accuracy rather than Loss as a learning signal, though to a lower performance. In conclusion, LS presents a viable alternative in cases where SGD fails or is not suitable. The simplicity of LS can make it attractive to non-experts who would want to try neural nets for the first-time or on novel, non-differentiable tasks.

***Index Terms*—Deep Neural Networks, Optimization Algorithms, Machine Learning, Deep Learning, Fashion MNIST, Convolutional Neural Networks, Non-differentiable Tasks**

I. Introduction

Deep neural networks have found many applications over the past few years. From Computer Vision [1] to Robotics [2], neural networks were used by experts and successfully demonstrated their wonderful potential. Deep models perhaps earned their attractiveness from their immense representation capacity and the automatic learning of features from data [3]. Stemming from such undeniable interest, this study aims to introduce an optimization algorithm to train neural networks. The goal is to simplify the training process, and perhaps complement existing optimization algorithms by offering yet-another-tool for the researcher or practitioner.

This paper focuses on the use of the Local Search (LS) algorithm to train neural networks. LS is a single-candidate derivative-free optimization algorithm. It differs from evolutionary algorithms [4] which are also derivative-free but multi-candidate. The Local Search algorithm can be useful in cases where the neural network or task is non-differentiable, does not have a representative loss function or the derivatives are uninformative in guiding the optimizer. The study focuses only on the Multi-Layer Perceptron (MLP) neural implementation, i.e. feed-forward networks. The Local Search algorithm, however, may be useful in other neural network implementations. In the following paragraphs, the training loop of neural networks will be examined.

Typically training is an iterative optimization process before deployment. Its goal is to find a set of parameters (weights and biases) suitable for the goal task. The network's parameters are modified by an optimization algorithm in a loop until convergence. When put together, the network's parameters can be thought of as a single vector, $\boldsymbol{\theta} \in \mathbb{R}^N$, where N is the space dimensionality and is possibly on the order of 10^6. The optimizer moves the parameter vector $\boldsymbol{\theta}_i$, in parameter space (set of all possible values), where i is the i-th iteration in the optimization process.

Think of $\boldsymbol{\theta}_i$ as a point on a hypersurface of N orthogonal dimensions. In such a scenario, knowing the direction in which to move the solution (which elements in $\boldsymbol{\theta}$ to increase, and which to decrease) is extremely valuable. Computing the gradient of the loss function usually tells the optimizer about the direction (in parameter space) in which it should move the current solution $\boldsymbol{\theta}_i$ to produce the candidate solution $\boldsymbol{\theta}_{i+1}$. This information about search directions has made gradient-based optimization algorithms a staple in neural net training.

The typical training process exposes the neural network to samples from input space, and evaluates the outputs of the networks through what is called a loss function. By minimizing training loss, which is a function of network's parameters, the neural network learns important features in the input space. Those features usually generalize beyond the training dataset, and are validated using the validation dataset. This is the typical training dynamic in Deep Learning. The choice of loss function is thus quite important. In well-defined tasks such as Image Classification [5], there are widely-used loss functions such as Categorical Cross-Entropy. For other ill-defined tasks, a representative loss function may need to be "designed" by an expert.

The proposed algorithm (LS) does not need a loss function to train the network. It only needs a "training signal", that

978-1-7281-3885-5/19/$31.00 ©2019 IEEE

may be from a loss function or may indeed be from another source. In this, LS makes training simpler. However, to be sure, having a loss function may still be useful in many cases. In addition, LS uses randomness to search local spaces instead of the global parameter space. As such, on a high level, the philosophy is to turn the vast global space into a smaller local neighborhood. The search takes place in only a constrained number of directions concurrently. This will be discussed in the coming sections.

A. Training Accessibility

In a well-defined setting, it is common to find representative loss functions that can be used to tackle the task at hand. An example of this is Image Classification. It is a well-defined task, and usually loss functions such as Categorical Cross-Entropy can be used to train networks to solve it. However, there are other sorts of settings where loss functions may not be available or representative. An example of this is Hyper-parameter Tuning. There is no loss function that an optimizer can use to solve this problem. Another example is Reinforcement Learning. While the Bellman equation is commonly used, it is not always representative of the task. An expert is usually needed in this case to design a loss function for the task.

The Local Search algorithm can be used by non-experts as an alternative or as a complement to traditional optimization algorithms for neural nets. The LS algorithm does not require a loss function to train a neural net. The simplicity of the algorithm, the simplicity of its implementation and the fact that it does not require a loss function can make it ideal for non-experts or those who want to try neural networks for the first time or on novel tasks. In a nutshell, it can make training neural networks more accessible to more people.

II. Related work

Despite creative variations, e.g. GANs [6], fundamentals of neural network training have remained relatively constant with the reliance on the Backpropagation algorithm [7] and Stochastic Gradient Descent [8] variants such as ADAM [9]. This stagnation in innovating neural network training methods has also been remarked upon in [10] and [11].

The authors in [10] proposed a new training approach called Decoupled Neural Interfaces (DNI). This method is an alternative to backpropagation. It is still derivative-based, as it uses gradient descent. They validated their approach on a number of tasks including the MNIST and CIFAR-10 datasets. The method performed competitively in terms of final performance, compared to BP. The experiments were performed on relatively long optimization horizons (500K steps). Compared to DNI, BP was extremely quick to converge. However, DNI was primarily developed to address the "Locking" problem in the BP algorithm, and it is successful in that regard.

The Kick-back approach proposed in [11] is also presented as an alternative to backpropagation. Like DNI, it is still derivative-based. The approach is validated on simple robotic datasets, and not complex ones such as MNIST or CIFAR. The authors remark that their method was still in its infancy, and thus not suitable for modern deep learning tasks such as multiclass learning (e.g. image classification). Kick-back was mainly addressing the credit-assignment problem, however. The performance of Kickback was similar to BP in terms of convergence speed and accuracy.

In recent years there were attempts to propose alternative neural network training methods in different settings. For example, researchers in [12] and [13] used Genetic Algorithms and Evolution Strategies to train a network to solve typical reinforcement learning tasks such as Atari games. Both algorithms performed well on the given tasks. Both are evolutionary algorithms and use a population size on the order of hundreds to a thousand. At each optimization step, hundreds of function evaluations need to take place before the optimizer can produce the new set of candidate solutions. Thus, this approach can be costly in terms of number of function evaluations, especially when compared to single-candidate optimization approaches.

There is little evidence to suggest a focus in the machine learning community on variants of Random Search, such as our Local Search, as means to train neural networks. There were attempts in the 1990s such as [14] and [15] to train neural nets with random algorithms. The tasks in those papers, however, are far less complex than modern deep learning tasks. The networks themselves were much smaller and did not contain millions of parameters as is common nowadays. Interestingly, it was remarked upon in [12] that random search performs exceedingly well to train the neural network, and does in fact solve Atari games. Random Search has been found to produce even "highscores" in those games, beating modern deep reinforcement learning methods such as DQN [16].

III. Algorithm

A. Local Search

Local Search is built around the concept of managing the immense dimensionality of neural network parameter space. The main approach is to reduce the global search dimensions, N, into a local neighborhood composed of a relatively small number of dimensions, S. Instead of searching N dimensions, the LS optimizer partitions the space into $K = \frac{N}{S}$ batches. Every iteration(s) it will search a segment for a number of subsequent iterations. To move in all N dimensions, K iterations need to elapse. The dimensions in each batch are picked at random by shuffling an indices vector $I = [0, 1, 2, ..., N-1]$ and then splitting it into K batches of size S. The shuffling is repeated every K iterations.

The intuition in this approach is as follows. When searching a dimension, there is no information to guide the optimizer on which direction (positive or negative) to move along, and by how much. This leaves the matter to a pseudorandom guess sampled from a uniform distribution $U(-r, r)$, where r is an empirically-derived hyperparameter.s. In aggregate, the guesses are defined as a noise vector $\mathbf{d}$ whose components are independently sampled from $U(-r, r)$. The more concurrent "guesses" the optimizer has to take, i.e. as S increases, the

less likely it will yield a positive outcome. Thus, randomly searching all dimensions at once is extremely unlikely to yield improvements in the score. Consequently, the idea is to avoid the pitfalls of a large-dimensional random search by only searching a quite small number of dimensions, i.e. perform a local search. Naturally, doing this will come at a cost of convergence speed.

B. Score Decay

During training, the optimizer can get stuck in a local minimum. By being "greedy", the optimizer can discard solutions that perform slightly worse than the current solution but would lead to a better solution over the proceeding optimization steps. Score Decay worsens the score (i.e. increases if the objective was to minimize, or decreases if the objective was to maximize) decreases by a certain amount p_{score} (0.0002 in this case) each step. It is designed to help the optimizer move out of local minima and explore the search space more actively. The value of p_{score} is determined empirically. It should not be too large, else the optimizer may diverge.

It should be noted that using Score Decay may cause the score to oscillate between better and worse values. In other words, without Score Decay, the score should monotonically improve or remain constant but never worsen. In the ablation study that will follow, the importance of Score Decay will be examined. The entire algorithm is given in Algorithm 1.

Algorithm 1: Local Search w/ Score Decay (minimization mode)

Result: $\boldsymbol{\theta}$
Initialize model: $\boldsymbol{\theta}_0$;
Declare noise vector of size S: $\mathbf{d} \in [-r, r]^S$;
Initialize indices vector: $\boldsymbol{I} = [0, 1, 2, 3, ..., N-1]$;
Prepare I( )
    Shuffle $\mathbf{I}$;
    Split $\mathbf{I}$ into K batches of size S;
    Declare batch index: $k = 0$;
Declare penalty: $p = p_{score}$;
score $\leftarrow F(\boldsymbol{\theta}_0)$;
for $i \leftarrow 0$ *to* $200,000$ **do**
    if $k > K$ **then**
        `Prepare I()`
    $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i$;
    Sample noise from Uniform distribution: $\mathbf{d} \sim U([-r, r]^S)$;
    Add noise to selected indices in $\boldsymbol{\theta}_{i+1}$: $\boldsymbol{\theta}_{i+1}[\mathbf{I}_k] = \boldsymbol{\theta}_{i+1}[\mathbf{I}_k] + \mathbf{d}$;
    if $F(\boldsymbol{\theta}_{i+1}) < score$ **then**
        Update rules:
        $score \leftarrow F(\boldsymbol{\theta}_{i+1})$;
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{i+1}$;
    if *convergence OR stagnation* **then**
        $Terminate$;
    Score Decay: $score = score + p$;
    Increment batch index: $k = k + 1$

C. Code Release

The code needed to re-produce the results of this study is publicly available as part of our DNNOP repository. The repository can be accessed at:

https://github.com/AroMorin/DNNOP

IV. Experiments

This section presents the experiments that were performed using the Local Search algorithm. LS is tested on the Fashion-MNIST dataset. The choice of this dataset is not arbitrary. In many papers, it is standard procedure when introducing a new concept to first test it on the hand-written digit classification dataset MNIST. However, over the years, it has been remarked that this dataset is not quite representative of modern machine learning tasks, particularly in Computer Vision. For that reason, the machine learning community began introducing different datasets, and FashionMNIST is one of those. The classes are much more challenging than regular MNIST.

The trained model has 5+ Million parameters. The model is a CNN composed of: 2 convolutional layers with 32 and 64 kernels, a maxpool layer, 2 convolutional layers of 128 and 256 kernels, a maxpool layer, a fully-connected layer with 6400 neurons, a fully-connected layer with 768 neurons and an output fully-connected layer with 10 neurons representing the 10 classification classes. In this work, the activation function used after each layer is the hyperbolic tangent function. However, the Rectified Linear Unit function (ReLU) has also been tested and was found to yield similar results.

There are 4 experimental sets. The first is an overview of optimization algorithms' performance on FMNIST. The second is a minor ablation study on the LS algorithm. The third is a presentation of hyper-parameter tuning. Finally, the fourth is a training of the CNN with the Local Search algorithm using only the training accuracy as a learning signal, i.e. not using a loss function as is typical in SGD-based optimization.

The first experimental set is a direct comparison on training performance between Stochastic Gradient Descent (SGD + BP), Local Search (LS) and Random Search (RS). Each algorithm is allowed to converge fully. If it is clear that the algorithm has converged or will not improve further, it is terminated. The aim is to showcase Local Search between an upper-bound (SGD) and a lower-bound (RS).

The second experimental set is a minor ablation study on

TABLE I
Results of experiment 1

Name	Training Loss	Training Acc (%)	Test Loss	Test Acc (%)
LS	0.0165	100	0.6760	84
SGD	0.1431	97	0.4024	86
RS	2.7617	9	2.7552	9

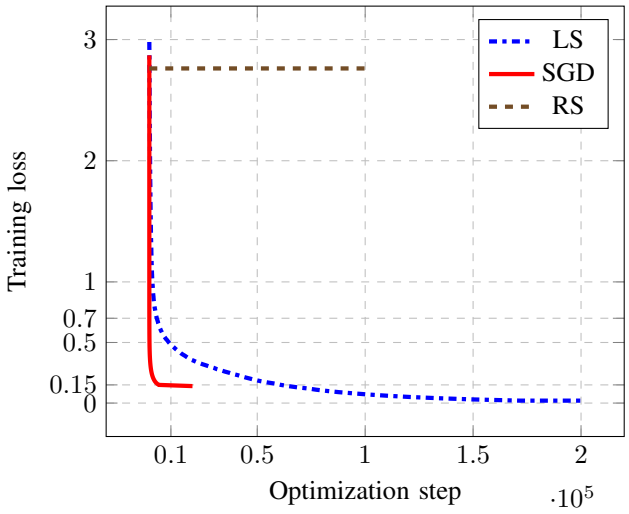

Fig. 1. Local Search takes $\sim$ 50X as many steps than SGD to reach the same loss. However, LS converges to a 0 loss, beating SGD. Random Search utterly fails.

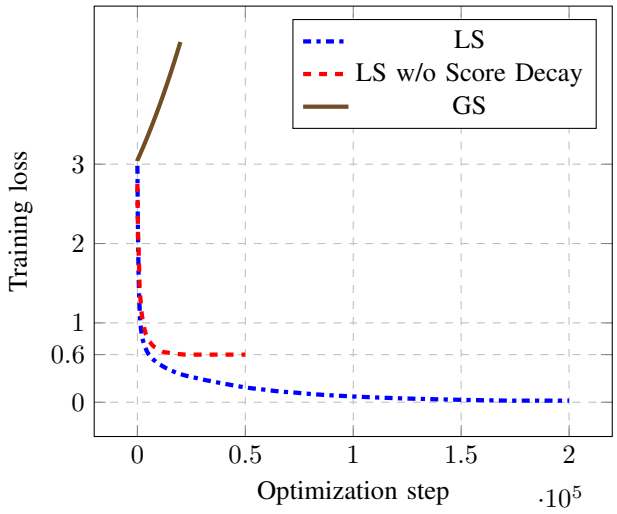

Fig. 2. Local Search is better when coupled with Score Decay. Global Search diverges.

two features of the LS algorithm. In each case, the same conditions are used except for the specific feature under examination. The most obvious feature is Locality. Thus, the first experiment is to conduct Global Search (i.e. use all directions). The second experiment is to remove the Score Decay feature. If the algorithm does not converge or it is clear it will not improve, training is terminated.

The third experimental set is an examination of the founding hypothesis of using Local Search to train neural nets: that the greater the number of dimensions the search is conducted over, the lower the likelihood of improvement is. In this set, the number of search dimensions is varied.

The fourth experimental set demonstrates the usefulness of LS when there is no loss function. Without a loss function, derivative-based methods would not be usable. In such a scenario, only derivative-free optimization methods are usable. The experiment examines the possibility of learning using only the training accuracy as a learning signal. The algorithm used to train the network is still Local Search. However, instead of minimizing the loss value, the algorithm attempts to maximize training accuracy. It should be noted that switching from loss to accuracy will likely require a re-tuning of LS hyper-parameters, such as S and r to achieve best performance in this task. We did not do re-tune the hyper-parameters, however, for the sake of conformity. The results of the four experimental sets are presented in Figures 1, 2, 3 and 4, as well as their corresponding tables.

A. Training Constraints

Local Search requires a static loss surface and thus is not amenable to mini-batch training as is used in SGD, which can estimate the gradient from a mini-batch. We can't use mini-batch training, but we nonetheless have three options. The first is to download the entire training set to GPU memory, and run inference on it. However, the FMNIST has 50,000 training images and could not be loaded entirely onto the GPU (Titan V 12GB).

The second option is to divide the training set into mini-batches in RAM, send each batch for inference on the GPU and then aggregate the metrics (loss and accuracy). This option means that the entire training set can be used. It comes at a cost of wall-clock time, however. There are more than 10 experiments performed in total in this study. Each experiment runs until the algorithm terminates, which can take up to hundreds of thousands of iterations. Running inference on the entire training set for millions of iterations is prohibitively time and resource-consuming.

The third option and the one used in this study is to download only a portion of the training set, as much as will saturate the GPU memory without blocking it. Inference is much faster, and the conclusions and insights drawn from the experiments should still hold. Thus, in this study, only 10% of the training set is used (5,000 images), picked at random.

In the case of validation, the entire set is used. Validation is performed only after the algorithm converges or is terminated, i.e. it is only performed once. For the validation phase, it is not prohibitive to divide the validation set into batches, send each to the GPU and aggregate the metrics (i.e. follow option 2 mentioned above). As such, in the validation runs, the entire validation set is used (10,000 images).

The relatively long horizons (high number of optimization

TABLE II
RESULTS OF EXPERIMENTAL SET 2: MINOR ABLATION STUDY.

Name	Training Loss	Training Acc (%)	Test Loss	Test Acc (%)
LS	0.0165	100	0.6760	84
LS w/o Score Decay	0.5972	78	0.6548	76
GS	3.0430	14	3.0656	13

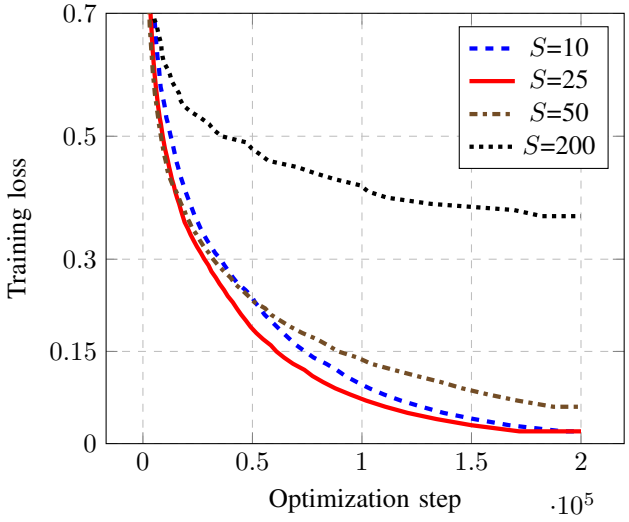

Fig. 3. Results suggest there is an optimal range for the number of search dimensions S, too high or too low degrades performance.

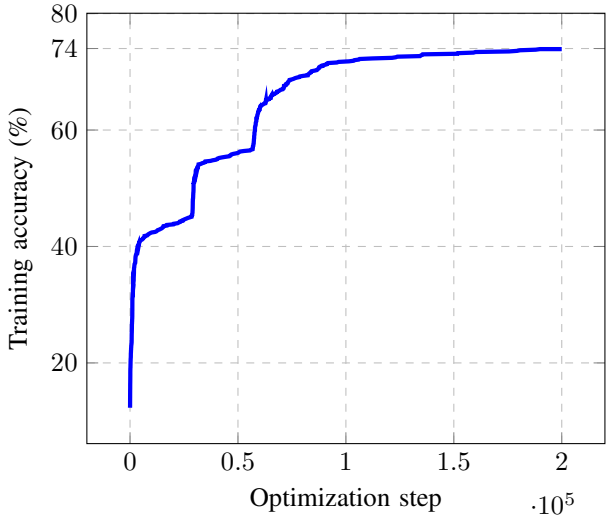

Fig. 4. What if you don't have a loss function? LS can train the CNN using only Accuracy as a learning signal.

steps) in this study obviate the need to generate confidence intervals. If there were a few "lucky" steps in the optimization trajectory, they would not affect the global behavior of the algorithm. Thus the experiment does not need to be repeated many times to assert performance.

V. Discussion

Each experiment will be discussed in turn. The first experimental set's results, as shown in Fig. 1, reveal that LS is not as fast as SGD in terms of convergence speed. It approximately takes SGD 1,000 steps to converge to a loss $\simeq 0.15$. LS takes 50,000 steps to reach the same loss, i.e. LS is up to 50X slower. This is not the whole story, as it is noticeable from the figure that the gap between SGD and LS widens as the optimization advances. It means that the 50X margin is a worst-case for this particular loss hypersurface. Interestingly, however, LS continues to improve until it reaches a loss of 0. SGD simply stagnates at 0.15. It may be suggestive of a resilience property of Local Search compared to Gradient-following approach.

In addition, it is clear that Random Search fails completely in this scenario. It does not improve the solution at all, even after running for 100,000 steps. This comes in contrast to earlier findings in [12]. In that paper, RS was found to be completely adequate to the point it surprised the authors. RS in those Atari game experiments achieved high-scores that compete with Deep Reinforcement Learning models. Here, however, it fails completely. It is perhaps suggestive on the different nature of the domains, and how and where randomness can be useful and to what extent. Thus, in this experiment, RS serves as a lower-bound and SGD as an upper-bound (in terms of convergence speed).

The second experimental set shows the important effect of dimensionality and Score Decay. It is clear from Fig. 2 that performing a Global Search (i.e. a search in all possible directions simultaneously) is not bountiful. Thus that algorithm terminated quickly after 10,000 steps. In addition, it is also clear that decaying the score has a large effect on convergence. In the early stages, using score decay has no significant bearing on convergence speed. However, as optimization advances, it is critical. The optimizer stagnates without it, and is terminated after 50,000 steps. Recall that score decay was introduced to help the optimizer overcome local minima and allow it to explore the search space more actively. Its effectiveness may suggest that the search space has deep local minima.

From set 3, the effect of varying the number of search dimensions is clear, as demonstrated by Fig. 3. First a note about the figure. That figure was cropped on the Y-axis so as to zoom-in on the bottom part. In the cropped-out part, the optimizer quickly descends as in Figures 1 and 2. There is not much difference between the competitors, i.e. little to no information. Thus, the figure is cropped so that the minute differences can be easily distinguishable.

Increasing the number of search dimensions (as seen when

TABLE III
Results of experimental set 3: Varying search dimensionality S

S	Training Loss	Training Acc (%)	Test Loss	Test Acc (%)
200	0.3696	88	1.0136	78
50	0.0584	99	0.7312	82
25	0.0323	100	0.6824	83
10	0.0223	100	0.6360	83

TABLE IV
Results of experimental set 4: Using Accuracy instead of Loss as a learning signal.

Training Loss	Training Acc (%)	Test Loss	Test Acc (%)
2.7539	74	2.9072	68

S=200) moves the LS algorithm to behave more like Global Search, and thus diminishes its value. This effect further confirms the hypothesis about search dimensionality and its bearing on the search process, and the effectiveness of LS in this case. Diminishing the number of search dimensions (as seen when S=10) also is not helpful. As the number of search dimensions gets smaller, the "impact" each guess has on the overall optimization is reduced. Consider the pathological case of a single search dimension. The optimizer would require a tremendous number of optimization steps to converge. This suggests that there is an upper and lower bound on the number of search dimensions to achieve optimal results with LS.

Finally, in the fourth experimental set the idea of training a neural network without a loss function is tested as seen in Fig. 4. Instead of using categorical cross-entropy as was with all other experiments, the training accuracy itself was used as a learning signal. It can be seen that the network does in fact learn, it achieves 74% accuracy on the training set. And as shown in Table IV the learnings are validated as it achieves 68% on the validation set.

The accuracy metrics are not as good as those achieved when a loss function is used for training. However, this experiment does indicate the plausibility of training without any loss function. Furthermore, it demonstrated that the hypersurface created by the loss function, i.e. loss hypersurface, is easier to navigate than the accuracy hypersurface. Remarkably, from Table IV, the both training and validation losses are quite high compared to the accuracy figures. This suggests that loss space is not the only representative space of the task. In other words, the task can be solved in other spaces which do not necessarily conform in topology to loss space. Also, there noticeable "jumps" in the graph. This suggests that learning through accuracy is sensitive to values of a small subset of $\boldsymbol{\theta}$.

VI. Conclusion

The Local Search algorithm was used to train a convolutional neural network with 5 Million parameters on the FashionMNIST dataset. Due to resource constraints and the number of experiments performed only 10% of the training data was used (5,000 images). The optimizer managed to achieve 100% training accuracy, and 84% validation accuracy on the entire validation set (10,000 images). LS outperformed SGD in terms of final accuracy and loss metrics. However, SGD was found to be up to 50X faster than LS. In addition, Random Search was found to be completely ineffective in training such a network, a departure from recent results in the deep reinforcement learning community suggesting otherwise.

This study, in addition to those findings, presented a minor ablation study and a survey on a significant hyper-parameter S (the number of search dimensions). The ablation study demonstrated the need for the features used in LS. Also, the intuition behind using LS has been confirmed when contrasted by the case of Global Search and generally high number of search dimensions S. It is certainly better to take random guesses in only a tiny subset of the entire search space. When S is too low, performance has worsened, which suggests there is an optimal range for the number of search dimensions S.

Finally, the case for training neural networks without a loss function has been presented. The CNN was trained to 74% training accuracy and 68% validation accuracy. Remarkably, those accuracy metrics coincided with relatively high loss metrics (2.75 and 2.9). This suggests that the optimization hypersurface created by accuracy is topologically different from that of the loss. Thus, training plausibility using accuracy was demonstrated, with an additional insight as a byproduct.

The experiments and approach presented in this study may hopefully act as a stepping stone towards further improvements in derivative-free single-candidate optimization algorithms for neural network training. The task of multi-class learning, the $5x10^6$ parameters and the use of FashionMNIST dataset all make the experiments here more applicable and representative of modern deep learning tasks. Though LS is unlikely to be a competitor to SGD at this stage, it may be used in tandem with it. In cases without loss functions or without differentiability, LS can be used as a go-to alternative to the researcher.

A. Future Work

This paper constitutes an introduction of Local Search to successfully train a convolutional neural network with more than 5-Million parameters. The LS algorithm still requires further studies in order to fully understand the manner in which it works under different training dynamics, and how it can be effectively used.

In particular, it is desirable to understand how the tuning of the hyperparameters of the task and the trained network will affect training performance and convergence. For example, in what way does the network architecture affect training performance when using LS? In addition, how will LS perform solved Deep Reinforcement Learning tasks.

Once these aspects are studied and understood, LS algorithm and its derivatives can start to act a replacement for the traditional SGD-based training approach. As such, the researcher can have multiple tools to train her network with and be able to select the best tool for any scenario.

References

[1] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[2] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[3] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT Press, 2016.

[4] Z. Michalewicz, "Evolution strategies and other methods," in *Genetic Algorithms+ Data Structures= Evolution Programs*, pp. 159–177, Springer, 1996.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 12 2015.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," 6 2014.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 10 1986.

[8] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.
[9] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 12 2014.
[10] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled Neural Interfaces using Synthetic Gradients," 8 2016.
[11] D. Balduzzi, H. Vanchinathan, and J. Buhmann, "Kickback cuts Backprop's red-tape: Biologically plausible credit assignment in neural networks," 11 2014.
[12] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.
[13] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.
[14] R. Battiti and G. Tecchiolli, "Training neural nets with the reactive tabu search," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1185–1200, 1995.
[15] K. Hirasawa, K. Togo, J. Murata, M. Ohbayashi, N. Shao, and J. Hu, "A new random search method for neural networks learning-random search with variable search length (RasVal)," in *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, vol. 2, pp. 1602–1607, IEEE.
[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.