Feed Forward Neural Network Sparsification with Dynamic Pruning

Andreas Chouliaras
University of Thessaly
Department of Electrical & Computer Engineering
Volos, Greece
achouliaras@uth.gr

Evangelia Fragkou
University of Thessaly
Department of Electrical & Computer Engineering
Volos, Greece
efragkou@uth.gr

Dimitrios Katsaros
University of Thessaly
Department of Electrical & Computer Engineering
Volos, Greece
dkatsar@inf.uth.gr

ABSTRACT

A recent hot research topic in deep learning concerns the reduction of the model size of a neural network by pruning, in order to minimize its training and inference cost and thus, being capable of running on devices with memory constraints. In this paper, we employ a pruning technique to sparsify a Multi-Layer Perceptron (MLP) during training, in which the number of topology connections, being pruned and restored, is not stable, but it adopts either one of the following rules: Linear Decreasing Variation (LDV) rule or Oscillating Variation (OSV) rule or Exponential Decay (EXD) rule. We conducted experiments on three MLP Network topologies, implemented with Keras, using the Fashion-MNIST dataset and results showed that the EXD method is a clear winner since, in that case our proposed sparse network has a faster convergence than the dense version of the same one, while it achieves approximately the same high accuracy (around 90%). Furthermore, it is shown that the memory footprint of the aforementioned sparse techniques is at least 95% less instead of the dense version of the network, due to the weights removed. Finally, we present an improved version of the SET implementation in Keras, using *Callbacks API*, making the SET implementation more efficient.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Machine learning**; **Machine learning approaches**; **Neural networks**;

KEYWORDS

Neural network sparsification, Keras, Multi-Layer Perceptron, Deep learning

ACM Reference Format:
Andreas Chouliaras, Evangelia Fragkou, and Dimitrios Katsaros. 2021. Feed Forward Neural Network Sparsification with Dynamic Pruning. In *25th Pan-Hellenic Conference on Informatics (PCI 2021), November 26–28, 2021, Volos, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3503823.3503826

PCI 2021, November 26–28, 2021, Volos, Greece

ACM ISBN 978-1-4503-9555-7/21/11…$15.00
https://doi.org/10.1145/3503823.3503826

1 INTRODUCTION

The unprecedented success of Deep Learning has made Neural Networks (NN) a very useful tool in research and development. However, the workload in order for a NN to be trained is great: there is a huge number of parameters, in order for the model to converge, hence, the training time is huge and has high memory requirements. While most of the articles, presented in literature to address the problem of accelerating NN training focus on implementing pruning techniques after the model being converged, our approach aims at enhancing the training time of a model during its training procedure. The only prior work, found in literature is the one presented in [8], in which SET algorithm prunes a specific number of close-to-zero weights in a MLP and restore them randomly, in every epoch. We are, clearly, being motivated by the encouraging results the SET algorithm gave, however, the number of linkages, being modified (*zeta* parameter) in every layer is stable, and specifically, is about 30%, even if we are in the last steps. This may not be beneficial as, technically, in the last step, our final model will have 30% of its weights reinitialized to be trained for only a single epoch. This might be a waste of resources.

So, we examine three different approaches, regarding *zeta* parameter, which is the one, responsible for the number of weights, being pruned and reconnected in every epoch. In our first one, zeta changes by a small value per epoch. We define a high and low pruning value, and we go from the one to the other linearly (Linear Decreasing Variation - LDV). In our second proposed technique, the value of zeta fluctuates between a low and a high value, following a cosine function rule (Oscillating Variation - OSV), while in our third and final method, zeta parameter undergoes about 1% reduction in every epoch, reducing exponentially the remodeling rate (Exponential Decay - EXD). The more effective technique appears to be the EXD method, in which we achieve comparable accuracy, regarding the initial dense implementation and reduced memory footprint along with a small speedup in training time.

The rest of the paper is structured as follows: the following sub-section presents our motivations and contributions, section 2 presents the related work, section 3 describes the proposed methods, and in section 4 we evaluate them based in classification accuracy, training time, and memory footprint. Finally, section 5 concludes this article.

1.1 Motivations and Contributions

The SET procedure tries to simulate the synapse remodeling phenomenon that occurs in biological brains by relating the training epochs of a NN to the days experienced in biological brains. We

wanted to zoom out a bit on the time scale, from a day-to-day scale towards a year-to-year scale while also retaining the changes that occur on a day-to-day scenario. Our approach involves modifying the weight remodelling rate throughout the training, giving greater opportunities to the model to mature and gain wisdom. In biological brains their structure changes not only from one day to the next, but also during the course of a whole year and those changes accumulate leading to more and more changes over the course of an animal's life. Our contributions is a simple abstraction to the complex mechanisms that are involved in biological brains. Another piece of our contribution presented in this article is our endeavor to improve the Keras implementation provided by Mocanu et al. [8]. In their original implementation, that was released in 2018, not much to their luck, the Keras API placed many restrictions to their efforts, resulting in an implementation that, even though it works fine, is it greatly un-optimized and leads to severe performance drops in time when compared to their dense counterpart. The modern Keras API, however incorporates the Callbacks API, that enables developers to insert custom code and modify the training sequence seamlessly resulting to much greater performance.

We would also like to align our methods, to the categories Hoefler et al. [5]. Among different compression techniques, our methods belong into the model sparsification category because they are the result of applying an Erdos-Renyi random graph to a much bigger initial dense model. This also explains why our methods also fall into the sparse training category. They start from a sparse model which they update during training by modifying its underlying structural topology. During training our methods prune and reconnect parameters at a variable rate that changes during the training process. When choosing candidates for removal, we use a data-free selection based on weight magnitude. And finally, when we need to regrow the network we apply a random regrowth technique that chooses new weights to reconnect randomly.

2 RELATED WORK

One of the first families of acceleration methods includes members that meant to replace gradient (steepest) descent. Whereas gradient descent is based on a first order Taylor series approximation of the performance function, methods based on second order Taylor series were investigated, such as Newton's method(s). Other algorithms are those based on Conjugate Gradient, and quasi-Newton method(s), e.g., Broyden-Fletcher-Goldfarb-Shanno (BFGS). Recently, fast optimizers have been proposed such as Adam, Adadelta [9]. Another family for training acceleration is based on variable learning rates, e.g., the delta-bar-delta method [6]. The topic of network architecture search [10] is a research area of great importance. However, these methods require a strong hardware infrastructure, search space [7] or more hyperparameters. Dropout [11] accelerates training by randomly dropping units during training. In similar spirit, are the methods which compute only a subset of gradients during back propagation, e.g., meProp [12]. An intriguing piece of work [3] suggests that only a part (subnetwork) of a neural network is responsible for carrying out accurately a particular prediction, and thus if we can detect which this subnetwork is, we can then train only this, gaining significant speedups at the training stage in the process. The family of methods that are related mostly to the present work are those based on neural topology sparsification [5]; there have been proposed in the literature, methods which specifically prune the connection between the neurons [4]. However, these linkage sparsification techniques do not aim at mimicking the topological structure of real neural networks, but are mainly based on eliminating close-to-zero weighted connections. The most closely related work to ours is that reported in [8], in which they start from a completely unstructured topology basis, i.e., purely random network, having a specific, stable number of connections removed and added in each epoch, which is not efficient enough, especially when we are in the last epoch and the model is almost trained. Our contributions involve improving the SET implementation [8] using the Callbacks API and introducing three new methods that add variability to the pruning and regrowth rate of the connections in the network during training to achieve greater performance.

3 PROPOSED TECHNIQUES

The SET procedure as we have seen, uses a constant zeta parameter equal to 0.3 or 30% that keeps the evolution rate unchanged throughout training. The weight evolution does not differentiate between first and last epochs, even thought it obvious that a neural network's behavior and stability changes as it converges towards the minimum over the training. Drawing inspiration from biology, where the synapse remodeling rate in biological brains changes throughout an animal's life, we tried making this weight evolution rate change during training.

The methods we proposed cover two main categories. Firstly, by making the hyperparameter zeta follow a genuinely declining function, of a linear one and a much steeper exponential one, we simulate the aging factor in biological brains that may help the neural network in our case 'mature' in a much smoother rate. Secondly, causing the parameter zeta to oscillate, we display a seasonal pattern that tries to simulates the seasonal changes in an animal's brain that occur during a chronological year (hibernation, brumation etc.).

The truth is, that a lot of functions can fall into these categories but the methods we implemented for our experiments, were designed to introduce as few new hyperparameters as possible but with decent performance. The idea of making the parameter zeta a variable also draws inspiration from variable learning rate techniques, a concept that has already existed for many years. Our implementation was based on the original author's Keras implementation but that one was implemented in a way that caused the sparse methods to run a lot slower than their dense MLP counterpart. A year or two after the original paper's release, Keras introduced the Callbacks API and using it, we constructed an implementation that is more efficient and closer to the desired concept.

3.1 Exponential Decay

Drawing inspiration from the field of finance we used a simple exponential decreasing function that makes the parameter zeta decay each epoch by a constant fraction called interest, as depicted in the following equation:

$$\zeta_i = \zeta_0(1 - interest)^{curr\_iter} \quad (1)$$

where ζ_0 is the starting zeta value equal to 0.3 just as the SET procedure and the interest was chosen to be equal to 0.01 for our experiments after trying several values, so that the decay in the zeta parameter was neither too steep nor too gentle that it didn't approach the zero for the number of epochs we used. We call this method Exponential Decay (EXD) and the relevant pseudo-code is depicted in Algorithm 1.

3.2 Linear Decreasing Variation

In a similar fashion to the variable learning rate methods used by Abbas et al. [1] we applied a linear decreasing curvature to the parameter zeta that is adjusted to the maximum number of epochs. We call this method Linear decreasing Variation (LDV), with the relevant pseudo-code in Algorithm 1. The following equation depicts this procedure:

$$\zeta_i = \zeta_{min} + (\zeta_{max} - \zeta_{min}) \times \frac{max\_iter - curr\_iter}{max\_iter} \quad (2)$$

We set a maximum zeta value of 0.3, just as the SET procedure, and a minimum one of 0.01 so that it starts with the maximum zeta value at the start of the training and linearly decreases to the minimum zeta value over the entire training process. We wanted the minimum to be close to zero but not equal so the procedure won't shut down completely by the end.

3.3 Oscillating Variation

Here, once again following the steps of Abbas et al. [1] we applied an oscillating variation curve to the zeta parameter, adjusted to the maximum number of epochs. We call this method Oscillating Variation (OSV). Just like the LDV method we need a minimum and maximum zeta value that is defined as 0.01 and 0.3 respectively. The following equations shows how this is done:

$$\zeta_i = \frac{\zeta_{max} + \zeta_{min}}{2} + \frac{\zeta_{max} - \zeta_{min}}{2} \times cos\left(\frac{2\pi \cdot curr\_iter}{T}\right) \quad (3)$$

$$T = \frac{2 \cdot max\_iter}{3 + 2k} \quad (4)$$

In the definition of the period T we use the parameter k to control the frequency of the oscillations and was set equal to 1 for our experiments after performing a small random search. The relevant pseudo-code can be seen in Algorithm 1.

Algorithm 1 Pseudo-code of the proposed techniques

```
Procedure Sparse_Training (Sparsity level ε, ζ_max, ζ_min, Frequency κ)
Initialize Fully Connected Neural Network model;
Sparsify_network(ε);
Initialize training algorithm parameters;
for i in range(1,epochs) do
    Feed_Forward();
    Back_Propagation();
    if method == EXD then
        Apply_EXD_rule( ζ_0, interest, i, epochs);    // Equation 1
    else if method == LDV then
        Apply_rule( ζ_max, ζ_min, i, epochs);         // Equation 2
    else if method == OSV then
        Apply_rule( ζ_max, ζ_min, κ, i, epochs);      // Equation 3
    end if
    Weight_evolution(ζ);
end for
```

4 EXPERIMENTAL EVALUATION

We evaluate our methods using the following metrics: a) memory footprint, b) accuracy on the test dataset and c) training time. The training time metric depicts the total elapsed time it needs to train a model. We conducted three experiments for each method and the results are calculated by their mean values.

4.1 Evaluation Settings

For our experiments, we used the Fashion MNIST dataset, one of datasets also used by Mocanu et al. One of the main reasons for this decision is that we can have a common ground for comparing our methods to the SET procedure and the Dense MLP networks, making our results more understandable. The Fashion MNIST is a dataset of 60,000 training examples and 10,000 test examples of 28 by 28 gray scale images, depicting clothes that belong into 10 classes. We also use data augmentation techniques to make the models generalize better and show greater accuracy.

As far as the models are concerned we chose three neural network setups. We use the Multi Layer Perceptron for all topologies in our experiments. The first topology has three hidden layers with 1000 neurons in each one (called MLP-1K). The second one is closely related to the experiments of Mocanu et al. and uses three hidden layers with 4000 neurons on the first, 1000 neurons on second and 4000 neurons on third (called MLP-4K). The final topology uses four hidden layers with 4000 neurons on the first, 2000 neurons on second and third layer and 1000 neurons on fourth (called MLP-4K4L). Although, deeper and larger DNN architectures are the most common approaches, they may incur substantial redundancy [2], which may lead in heavy computational cost, without any special contribution, regarding accuracy. After a lot of experiments, we decided to use the aforementioned topologies to cover a wide range of topology characteristics that will help outline differences in the behavior of the tested methods. We train the models for 500 epochs using the default values for most of the hyper parameters. The experiments were executed on a desktop computer with 16 GB of RAM and a Quad-Core Intel Core i7 processor clocked at 4.0 GHz. For the sparse models (SET, LDV, EXD, OSV) we assign the sparsity level ε equal to 20. The value zeta stays at 30% for the SET procedure, as this was the proposed value from its creators, and we use this same value as well for the initialization of zeta on our methods. The parameters' initialization values that we used for each method, are reported along with the methods they belong to at section 3.

4.2 Evaluation Results

The fist benefit of our methods is their reduced memory footprint especially compared to their dense MLP counterpart. Both the SET and our methods use the same Erdos-Renyi random graph procedure to make their networks sparse at the beginning of the algorithm. As seen in Table 1 we achieve at least 95,7% reduction in parameters compared to the dense topology. This creates compression rates from ×23 to ×45 depending on the topology. Due to the nature of both SET and our methods, we know the exact model size before training, because our methods belong to the sparse training

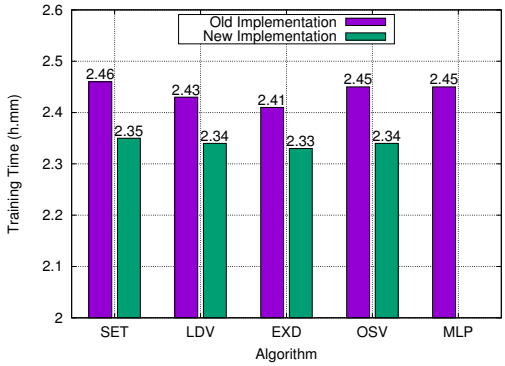

Figure 1: Competitors' training time between the old and the improved Keras implementation based on the following network architecture: MLP 1000-1000-1000 (MLP-1K).

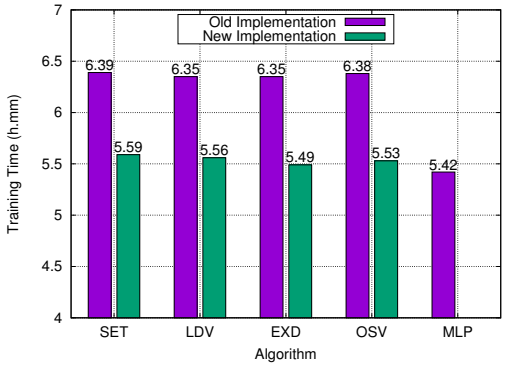

Figure 2: Competitors' training time between the old and the improved Keras implementation based on the following network architecture: MLP 4000-1000-4000 (MLP-4K).

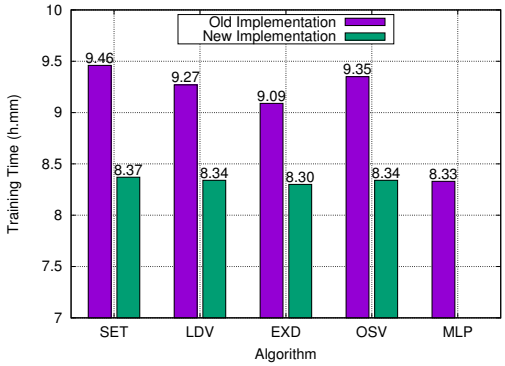

Figure 3: Competitors' training time between the old and the improved Keras implementation based on the following network architecture: MLP 4000-2000-2000-1000 (MLP-4K4L).

category of model pruning, which means that even though the model's parameters are pruned and reconnected throughout the whole training process, the total amount of parameters at the end of each epoch stays the same. This is very beneficial feature, as we can tune the parameter ε before training to achieve the desired compression rate, which helps on models that are designed to run in IoT devices with small memory capacities.

The second part of evaluating our methods is the training time. As illustrated in Figures 1-3 the speed improvement comes in two ways. Firstly, when comparing the dense MLP to the sparse models we see same training time costs in the MLP-1K topology Figure 1, but the sparse models become slower and slower in comparison as the model grows in size MLP-4K and MLP-4K4L Figures 2 and 3, even though their memory footprints are much better than the dense MLP. Using Keras Callbacks these differences are now greatly reduced as the weight evolution is now performed more smoothly.

We would expect that the sparse models should be faster than the dense, considering the compression rate achieved, but as Hoefler et al. [5] pointed out this in not always the case when using such libraries. The reason is that, most modern libraries are not designed to use sparse matrices and as a result they don't always exploit their reduced size efficiently. Another speed improvement that is evident from the same Figures is how our methods compare to SET and the dense MLP network. Our methods are consistently slightly faster on average than the SET procedure with the Exponential Decay (EXD) method being the winner among the sparse methods when also taking the improved implementation into consideration. We attribute this behavior due to the reduced weight pruning and reconnecting that happens as a direct result of the parameter ζ reduction.

The final part of the evaluation involves accuracy, a metric usually considered the most important for the model's performance. As we can observe from Figures 4–6 our methods remain competent to the SET and dense MLP models. We can see that the EXD method outperforms the rest, for the topologies MLP-4K and MLP-4K4L. In the smaller MLP-1K topology though we can see that the reduced number of total parameters in the network has a toll on the performance of all the sparse methods.

Hidden Layer Architecture	Dense Size	Sparse Size	Compression rate	Percentage Reduction
MLP 1000-1000-1000 (MLP-1K)	2.797.010	≈ 120.000	×23	95,7%
MLP 4000-1000-4000 (MLP-4K)	11.185.010	≈ 350.000	×32	96,8%
MLP 4000-2000-2000-1000 (MLP-4K4L)	17.155.010	≈ 380.000	×45	97,7%

Table 1: Comparing the parameter reduction between the dense MLP and the sparse methods.

We can therefore safely declare the EXD method as a clear winner in accuracy as well among the sparse methods. Our other two methods although not as good as the EXD, we can see that they still manage to outperform the SET method by a margin depending to the topology used. We can observe that all our methods and especially EXD, have the potential to outperform the dense MLP when using sufficiently large topologies.

We should keep in mind however that the dataset used belong into the image classification category and is only one kind of problems that neural networks are used for. Our methods may perform differently on other types of datasets, and this is mainly the reason that we didn't discarded the losers, so that they may provide some inspiration for other experiments or to maybe find application to other areas. But if anyone would consider trying one of our methods, the EXD method should be the default or at least the first choice.

5 CONCLUSIONS

In this paper we apply methods of pruning in neural networks using sparse training techniques in order to boost their performance. We improved the original SET Keras implementation and examined three new methods to further improve upon its qualities. Our improved implementation using Keras Callbacks API shows much more promising results in training time when comparing the sparse methods over the dense MLP.

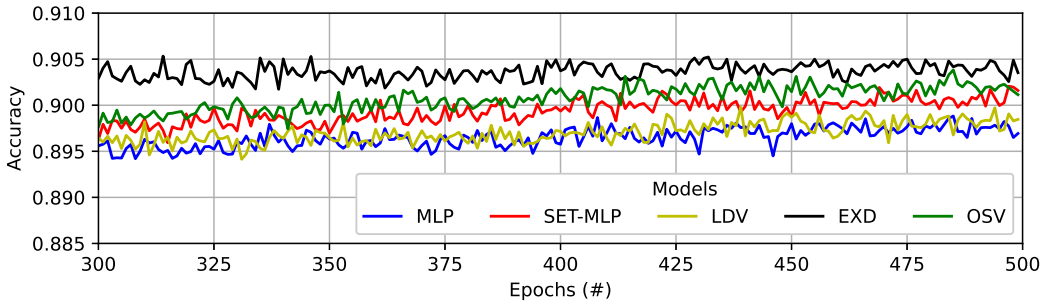

Figure 4: Competitors' accuracy based on the following network architecture: MLP 1000-1000-1000 (MLP-1K).

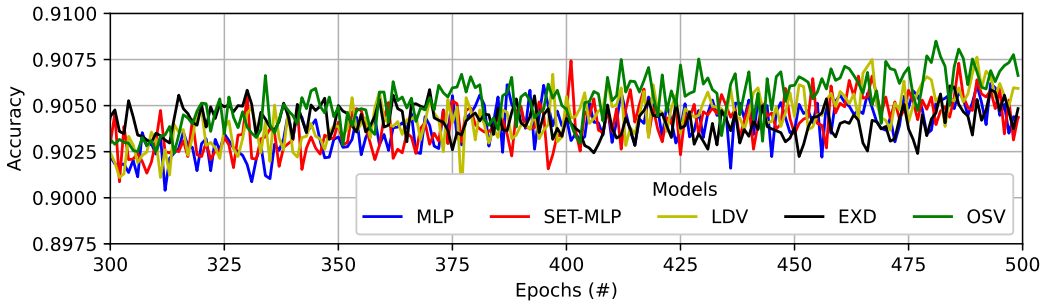

Figure 5: Competitors' accuracy based on the following network architecture: MLP 4000-1000-4000 (MLP-4K).

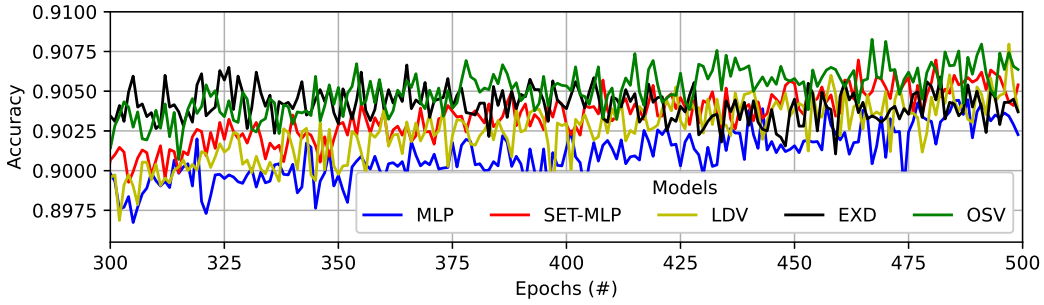

Figure 6: Competitors' accuracy based on the following network architecture: MLP 4000-2000-2000-1000 (MLP-4K4L).

In memory footprint we confirm the dominance of the sparse methods as opposed to the dense MLP by achieving at least 95.7% reduction in memory footprint, indicating them as a potential great fit for small IoT devices. Furthermore, the methods we propose also seem to display some benefits over the SET procedure in training time while staying competitive in accuracy. Among them, the Exponential Decay method (EXD) really stands out, surpassing the other methods on almost every experimental scenario. The EXD method is around 8-13% faster in training speed and constantly slightly more accurate than the SET procedure. Overall, we chose to keep a simple and elegant design approach for our methods.[1]

REFERENCES

[1] Q. Abbas, F. Ahmad, and M. Imran. 2016. Variable learning rate based modification in backpropagation algorithm (MBPA) of artificial neural network for data classification. *Science International* 28, 3 (2016), 2369–2380.

[2] X. Dai, H. Yin, and N. K. Jha. 2018. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. Available at: https://arxiv.org/abs/1711.02017.

[3] J. Frankle and M. Carbin. 2019. The Lottery Ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR).*

[4] S. Han, H. Mao, and W. J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR).*

[5] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. 2021. Sparsity in deep learning: Pruning and growh for efficient inference and training in neural networks. Available at https://arxiv.org/abs/2102.00554.

[6] Robert A. Jacobs. 1988. Increased rates of convergence through learning rate adaptation. *Neural Networks* 1, 4 (1988), 295–307.

[7] J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang. 2020. AtomNAS: Fine-grained end-to-end neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR).*

[8] D. Mocanu, E. Mocanu, P. Stone, P. Nguyen, M. Gibescu, and A. Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications* 9 (2018).

[9] S. J. Reddy, S. Kale, and S. Kumar. 2019. On the convergence of Adam and beyond. In *Proceedings of the International Conference on Learning Representations (ICLR).*

[10] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and W. Wang. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *Comput. Surveys* 54, 76 (2021), 1–34.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958.

[12] X. Sun, X. Ren, S. Ma, and H. Wang. 2020. Training simplification and model simplification for deep learning: A minimal effort back propagation method. *IEEE Transactions on Knowledge and Data Engineering* 32, 2 (2020), 374–387.

[1] GitHub Code: https://github.com/achouliaras/sparse_ann_training_dynamic_pruning