Speeding Up Convergence For Sparse Training Using Feature Extraction Behaviour

Andrei Radu
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.radu@student.utwente.nl

ABSTRACT

Deep learning is a powerful subset of machine learning algorithms, using multiple layers to learn complex patterns from large amounts of data. As deep neural networks require huge amounts of computations, sparsity addresses this problem by having removed a proportion of the connections in the network. Sparse Evolutionary Training (SET) is an approach to sparsity that involves removing connections based on the magnitude and adding new random ones, allowing for sparse-to-sparse training. A method to add connections to the network in a less random manner is introduced. With the help of skip layers connections and the linear features learned by them, during training, LiSET will add connections based on important linear features. While this method offers trade-offs, we argue that this approach leads to a faster convergence on certain datasets, especially when training on high levels of sparsity. An evaluation metric is proposed to estimate the speed up in convergence. On training the algorithm using stochastic gradient descent on two datasets, MNIST and ISOLET, it outperformed SET on the proposed evaluation metric and reached greater accuracy. This work presents a contribution to the research in sparse neural networks and speeding up the convergence of such models.

Keywords

artificial neural networks, sparse neural networks, sparse evolutionary training, fast convergence, linear features

1. INTRODUCTION

Deep learning is part of the family of Machine Learning methods that is leading the innovation in what we call today AI. The increasing quantity of data and the use of deep learning becoming more advanced and accessible make deep learning become a ubiquitous part of applications, representing a fundamental shift in how we develop software [10], and finding uses in tasks such as visual or speech recognition, natural language processing and others.

Artificial neural networks are inspired by the human brain, but the way these learn is not closely as efficient. As the field of deep learning progresses, better algorithms are designed, but the number of parameters used greatly increases [23]. Larger models will require greater storage capacity and huge amounts of computations. Thus it becomes hard for people using devices with limited capacity, including researchers, to keep up with the innovation in the field.

This is where sparsity comes in. In a trained artificial neural network, a high number of parameters are redundant to the performance [2]. Sparsity refers to having a percentage of the numbers in the matrices set to zero, in order to reduce the number of calculations involved in running a model. This makes the training of a network faster timewise (thus more energy efficient), minimizes the storage and helps improve generalization [13, 7].

Another method usually used to reduce computational complexity is feature selection. This method consists of picking a subset of the most relevant features which can efficiently represent the input data.

Different methods have been addressed to achieve results on both sparsity and feature selection [8, 1]. The research in this paper builds mainly upon Sparse Evolutionary Training (SET) of Mocanu et al. [17]. This method consists of continuously removing connections of small magnitude and adding new random ones. What is proposed in this work is a novel approach to adding connections in the network when using SET, inspired by feature selection behavior. A link to a repository implementing the method is provided in footnote[1].

The rest of the paper is organized as follows. Section 2 introduces concepts from the field and gives an overview of the related work and research the method is based on. Section 3 introduces the proposed method, with details on theory and implementation. In section 4, results on experiments on two datasets are shown, compared to vanilla SET, completed by a discussion in section 5 on the benefits and trade-offs of the proposed method. Finally, some conclusions of the research will be discussed in section 6 and possible future research not covered by this work.

35th Twente Student Conference on IT July 2nd, 2021, Enschede, The Netherlands.

[1] https://github.com/AndreiMVP/liset

2. BACKGROUND AND RELATED WORK

2.1 Deep learning

Deep learning is an area of machine learning methods based on artificial neural networks. An artificial Neural Network (ANN) is a brain-inspired mathematical model in which the algorithm learns its parameters based on given input and, in the case of supervised learning, output data. Deep learning architectures can be of many types such as deep neural networks, deep belief networks, convolutional neural networks, recurrent neural networks and others [15]. This work makes use only of deep neural networks.

A Deep Neural Network (DNN) is an artificial neural network containing more than two layers. The network is called dense when each neuron of a layer is connected with all the neurons of the next and previous layer. When data is given to run the network on, each neuron gains a different level of activation, depending on the connections and the values of the neurons associated with it. The activation function defines how the neuron should be activated: some time ago the most popular was the sigmoid function, which sets the activation to be a number between 0 and 1, but ReLU [19], which only sets negative activations of the neuron to zero, was found to be more efficient. The process described earlier proceeds from the input to the output layer and is called forward propagation. When training, the error between the computed and the desired output is calculated. The goal of training is to minimize this error, without overfitting the training data.

Backpropagation is used to alter the parameters in the network, depending on the error, to improve the performance of the given data. Gradient descent is the most popular algorithm used to optimize such networks, by iteratively attempting to find a minimum for the loss (error) function. The step toward that minimum is scaled by the learning rate. One of the simpler and more popular variations of gradient descent is Stochastic Gradient Descent (SGD), which takes a step in the gradient-based on batches of data and applying some smoothness properties and momentum [22].

2.2 Sparsity

As explained in the introduction, sparsity is having a chunk of the connection in the network set to 0, in order to reduce the computations involving them. While this topic has been explored for a long time since the 1990s [13, 7], it gained popularity only recently. Dey et al. show that 'larger and sparser' networks lead to better accuracy than 'smaller and denser' [3], as was tested on a dataset from each of 3 different types of datasets: image, speech and text.

In 2019, Frankle and Carbin articulated the lottery ticket hypothesis, which states that dense, randomly-initialized networks contain subnetworks that when trained in isolation reach test accuracy comparable to the original network [4]. Gale et al. [5] compared three state-of-the-art sparsification techniques: IO regularization, variational dropout and magnitude pruning. They implemented simple heuristics to achieve similar or better on a reduced computational budget, but not declaring a winning method. They also concluded that complex techniques perform inconsistently especially on small datasets.

Sparsity is not only limited to connection but larger units of the network as well. In 2021, Hoefler et al. offer an overview of different approaches taken to sparsify neural networks [8]. Sparsity has been explored to remove model parameters from weights (unstructured sparsity) to neurons and filters, channels, or heads (structured sparsity). There have also been diverse approaches taken towards how to select candidate elements for removal during sparsification. These have been put into three categories: data-free (local-features-based), data-driven (inference-based) and training-aware.

In 2018, Mocanu et al. have introduced sparse evolutionary training (SET), an approach to sparsity inspired by the simplicity of evolutionary approaches [17]. Instead of starting training from a dense neural network (dense-to-sparse training), SET starts with the fully connected layers replaced with Sparsely-Connected layers (SC). After each training epoch, for each SC, a fraction ζ of the smallest positive and the largest negative weights is removed. Figure 1 offers better visualization of this algorithm.

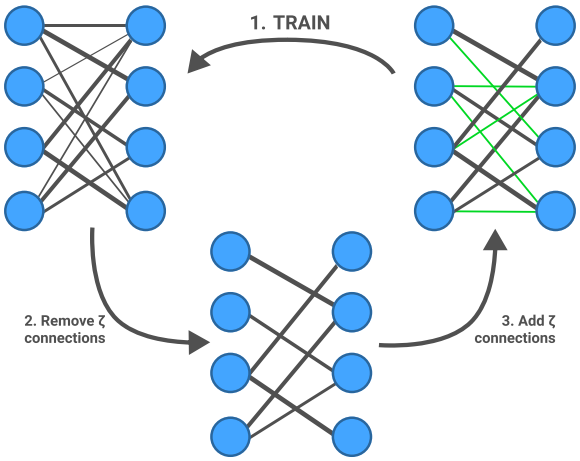


Figure 1: Visualization of SET algorithm.

Several variations of SET have been proposed [21, 11]. Pieterse and Mocanu have tried 5 different algorithms that perform cosine similarity-based connection removal and/or addition [21]. The algorithms were found to perform collectively better on 7 out of the 8 datasets trained on, with those involving cosine similarity-based connection addition performing better.

2.3 Feature selection

Feature selection is known as the process of selecting a subset of relevant features that can efficiently describe the input data, reducing the impact of redundant and irrelevant features [6]. It helps in reducing the complexity of the model and the computations needed.

In 2014, Chandrashekar and Sahin provided an overview of feature selection methods [1]. They classify feature selection methods in three categories: filter methods, which use variable ranking techniques as the principle criteria for selection, wrapper methods, which use the performance of the predictor as the feature selector criterion and embedded methods, which incorporate the feature selection as part of the training process.

Besides SET, the other method that this research was inspired by is LassoNet [14]. While lasso regression only applies to linear models, LassoNet is a neural network framework that is capable of performing global feature selection. It makes use of skip layer connections (connections that go from the neurons of the input layer to the output layer), using the weights as upper bounds for the first hidden layer weights. It manages thus to select some input features for

which it eliminates all of the weights.

3. LISET

In this research, a new method to add connections in SET is introduced, based on the linear features of the layers previous to each sparsely connected layer (LiSET). When performing an evolution, the SET algorithm, after removing connections, adds new ones randomly. The idea of LiSET is to make the addition part less random by making it more likely for each sparsely connected layer to add connections to more important features. What will often be referred to as features will be the neurons of the layer previous to a sparse one.

First, in order to set a base for the notation, we will start with the way the sparse layers were initialized, in the same manner as Mocanu et al. [17]. Each sparse connected layer SC^k has n^k neurons $[h_1^k, h_2^k, ..., h_{n^k}^k]$. Connection between h^k and h^{k-1} are collected in a sparse matrix $W^k \in R^{n^{k-1} \times n^k}$. W^k is initialized as a Erdös–Rényi random graph with probability of connection between neurons h_j^k and h_j^{k-1} given by the following equation:

$$p(W_{ij}^k) = \frac{\epsilon(n^k + n^{k-1})}{n^k n^{k-1}} \tag{1}$$

Parameter ϵ controls the sparsity level such that the smaller its value, the sparser the network.

While LassoNET uses skip layer connections from the input to the output layer, LiSET will use skip layer connections from each of the layers previous to the sparsely connected layer, as visualized in Figure 2. In a network with K layers $[L^1, L^2, ..., L^K]$ (including input L^1, output L^K and hidden layers) for any sparsely connected layer L^k (which we can write as SC^k), there is a fully connected layer from the previous layer L^{k-1} to the last, L^K, which we will call skip layer with weight matrix $\hat{W}^k$.

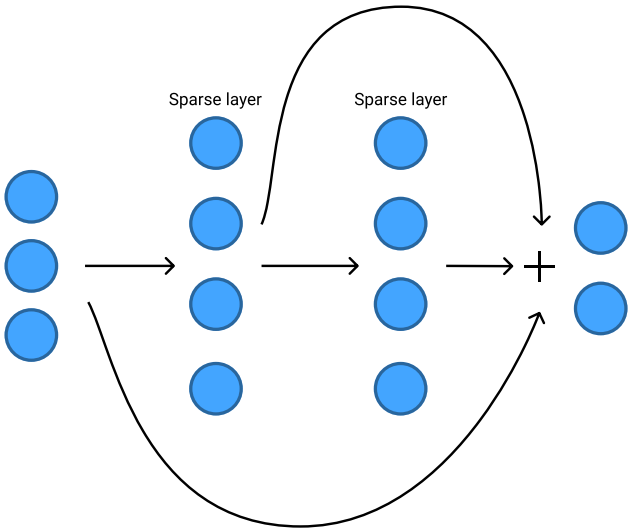


Figure 2: Architecture of network with skip layers

Skip layers will only be used during training. We will refer to the network not including the skip layers as the main network and that including them as the full network. When training, on a forward propagation the outputs from all the layers connected to the output are summed before applying any function. Each skip layer will find linear features that have not been fully utilized by the main network. The model will be evaluated only on the main network, so the goal is to maximize its performance. To make the model performance dependant only on the main network and the convergence faster, skip layers connections will be reinitialized after each evolution. It is also possible to use dropout on them, each skip layer having the probability of not being used equally to a given skip dropout rate.

Algorithm 1 LiSET pseudocode

```
/* Notation: K_s denotes the number of sparse
   layers, J_k denotes the number of neurons for
   layer k, Ŵ_j^k denotes the connections for neu-
   ron j of the feature layer corresponding to
   layer k                                        */
set ε, ζ and γ
initialize model
for each Fully-Connected (FC) layer of the ANN do
    replace FC with a Sparse Connected (SC) layer
end
for each epoch do
    perform standard training
    for k ∈ {1, ..., K_s} do
        /* For each sparse layer                 */
        remove ζ weights
        for j ∈ {1, ..., J_k} do
            w_j ← ||Ŵ_j^k||_1
        end
        w̃ ← σ(w / max(w) + γ)
        add weights according to w̃, in the same number as
         previously removed
        reinitialize Ŵ
    end
end
```

The LiSET algorithm is detailed in Algorithm 1. When performing an evolution based on the algorithm, we take the sparse layer SC^k and for each neuron of the feature layer $\hat{W}^k$ corresponding to SC^k we calculate the absolute norm of weight values going from that neuron to the output layer. We put the values in the vector $w^k = [\|\hat{W}_1^k\|, \|\hat{W}_2^k\|, ..., \|\hat{W}_n^k\|]$. The goal is to reach from w^k the vector of the same dimension $\tilde{w}^k$ using the parameter γ, which helps control the variation w^k and which will be explained in more detail shortly.

During the evolution, when adding a new connection, the connection will be more likely to come from a neuron which corresponds to a higher $\tilde{w}^k$ value. This likelihood is better generalized by Equation 2, stating the ratio between probability of connection between neurons $h_{i_1}^{k-1}$ and $h_{j_1}^{k-1}$, and probability of connection between neurons $h_{i_2}^{k-1}$ and $h_{j_2}^{k-1}$. The addition of weights in LiSET is also represented in Figure 4.

$$\frac{p(W_{i_1 j_1}^k)}{p(W_{i_2 j_2}^k)} = \frac{\tilde{w}_{i_1}^k}{\tilde{w}_{i_2}^k} \tag{2}$$

As shown in algorithm 1, the vector $\tilde{w^k}$ is computed from vector w^k and using the parameter γ with the aim of controlling the variation between values of w^k. By applying the sigmoid function, if $\gamma > 0$, as it increases, the variation between values of w^k will shrink, with the variation between greater values shrinking faster, as is exemplified in Figure 5. Similarly happens when $\gamma < 0$ and it decreases, except that the smaller values will shrink faster.

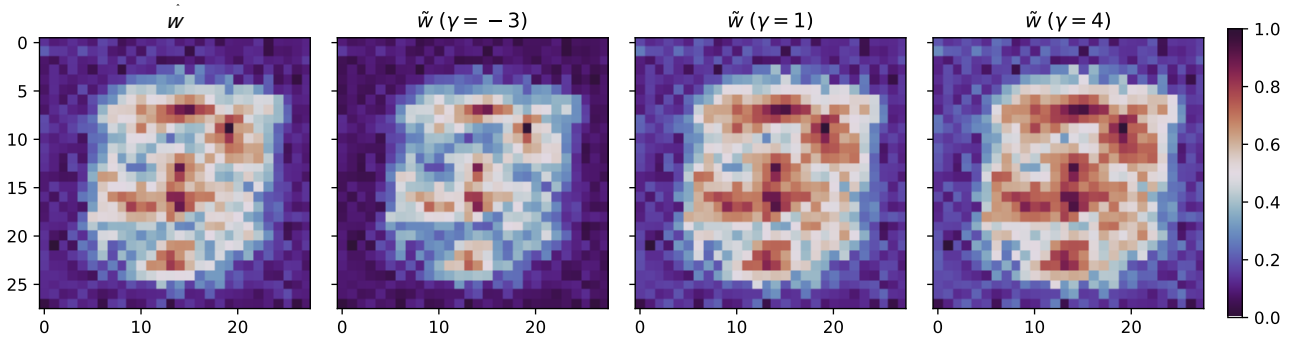


Figure 3: First figure to the left is an example of features w found by the skip layer from the input layer to the output layer of a network trained on MNIST database. The rest of the figures are values of $\tilde{w}$ resulted from applying different γ values. All the vectors have been normalized from 0 to 1.

In Figure 3, the distribution the standard deviation of $\tilde{w^k}$ is exemplified based on different γ values. One key difference between very big values and very small values of γ is that as big values converge at 1, their ratio as represented in Equation 2 becomes closer to 1, meaning that features are taken as more equal in importance and the connections are picked more randomly. Thus a γ value big enough adds connections in the same manner as SET. When γ is very small and converges to 0, that ratio can increase (for example, the ratio between 0.2 and 0.1 is very different from the ratio between 0.9 and 0.8). A smaller γ value puts more accent on fewer features.

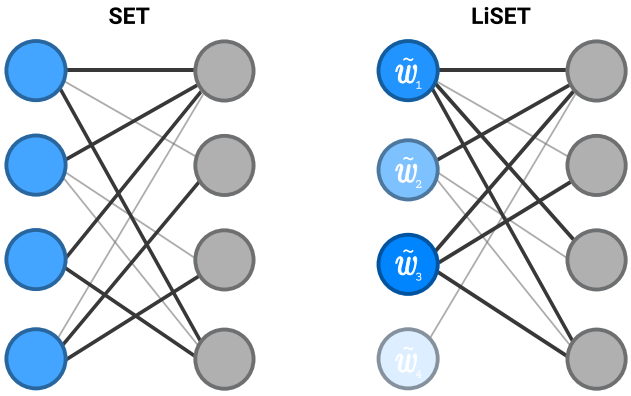


Figure 4: On the left, it is shown how weights are added randomly in SET. On the right, we see that in LiSET weights are added less randomly based on $\tilde{w}$. Neurons corresponding to a higher $\tilde{w}$ value will have more of the connections added to them.

By adding connections less randomly in the manner of LiSET, the network might catch on to important features faster, achieving a boost in accuracy early in the training. The learning of the network might be handicapped by the use of the skip layers, but on high levels of sparsity, the trade-off might be worth it. This will be further explored in experiments.

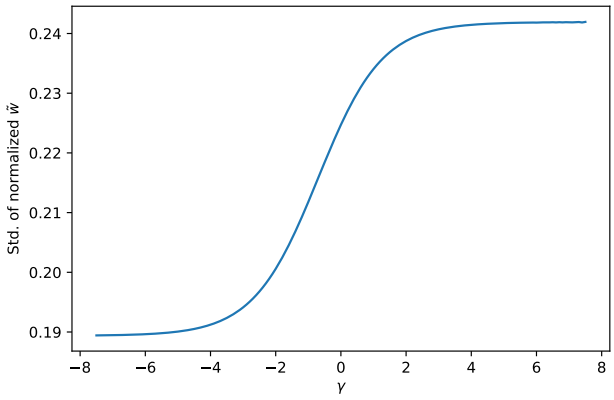


Figure 5: The distribution of standard deviations of $\tilde{w}$ normalized from 0 to 1 depending on the value of γ in a situation similar to that of Figure 3

3.1 Calculating the speed of training

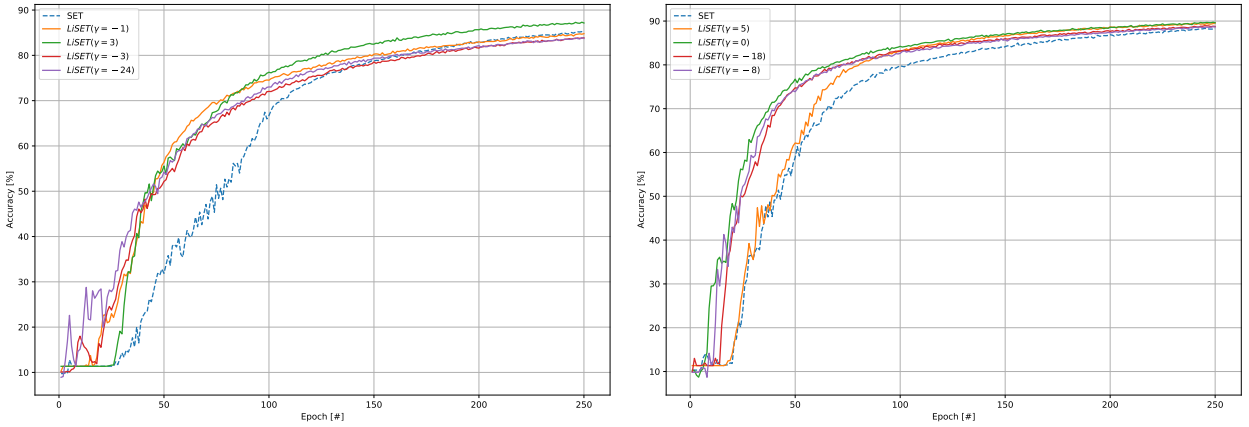
To calculate the speed at which the network improves the accuracy, a new evaluation metric is introduced which will be called Average Ratiod Accuracy (ARA). We take the accuracy a_i corresponding to i^{th} epoch of the n epochs trained and sum them up in the manner shown in Equation 3, with $c_1 = \frac{n-nb}{n-1}$ and $c_2 = n - nc_1$, for shorter notation. b represents the impact of the first training epoch on the final score compared to the last one, with this impact uniformly increasing from early to late epochs. If $b = 0.5$, the first epoch has half the impact of the last. When $b = 1$, the ARA is equivalent to the mean of the accuracy of each epoch.

$$ARA = \frac{\sum_{i=1}^{n} a_i(ic_1 + c_2)}{\frac{n^2(b+1)}{2}} \tag{3}$$

Greater ARA suggests greater overall accuracy along all epochs. A higher value of ARA and equivalent or higher final accuracy for a training interval compared to others suggest faster convergence.

Dataset	Batch size	Architecture	Learning rate used	ϵ used	Corresponding sparsity
MNIST	100	784-**300-100**-10	0.001	4.5	97.5%
				9	95%
ISOLET	100	617-**512-256-128**-26	0.01	21	90%
				42	80%

Table 1: Summarization of parameters used on the two databases



(a) Initialized with $\epsilon = 4.5$

(b) Initialized with $\epsilon = 9$

Figure 6: MNIST accuracy over epochs using different γ

4. EXPERIMENTS

4.1 Datasets

The performance was evaluated on two datasets:

- MNIST [12], consisting of 28x28 grayscale images of handwritten digits. The dataset consists of 50 000 train samples and 10 000 test samples.

- ISOLET [18], consisting of preprocessed speech data, of 617 quantities, of subjects speaking the names of the 26 English alphabet letters. The dataset consists of 6 238 train samples and 1 560 test samples.

The chosen datasets are often used in feature selection experiments, so they were picked with the fact that they will have some input features to be exploited by LiSET in mind.

4.2 Implementation details

LiSET was implemented and tested using the Pytorch library [20]. The optimizer used was SGD. Details of the parameters picked are shown in Table 1. Lower values for the learning rate were picked in order to better examine the effect of the method on convergence compared to SET on the number of epochs trained. Values of ϵ were picked to approximate certain levels of sparsity, shown in Table 1. For the experiments, there were more negative values of γ picked than positive ones, because the too large value of the parameter leads to randomly adding weights, as explained previously. The values for γ were picked from -24 to 5 in a non-uniform manner, as will be shown in experiments.

LeakyRELU [16] was used as the activation function. The loss function used was the negative log-likelihood loss. A dropout rate of 0.3 was used for the main network and 0.5 for the skip layers. For evolution, ζ of 0.3 was used. The models were trained for 250 epochs on constant sparsity detailed in Table 1.

Before delving into the results, it is important to bring to attention the fact that the conditions of SET training are different from those used in the literature, thus the results are not comparable. Besides different parameters used, different architecture was used for MNIST, fewer epochs trained and SReLU [9] was used as activation function by Mocanu et al. [17].

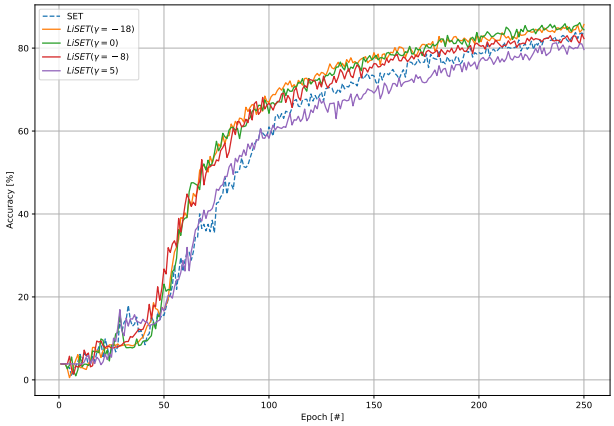
4.3 Results

For evaluation of the results, two metrics were picked, namely the accuracy achieved after the number of epochs and ARA with $b = 0.5$, Results of the experiments are shown in Table 2 for MNIST and Table 3 for ISOLET, with the best values written in bold. For each dataset and sparsity, two of the best and two of the worst results were picked and plotted along with SET in Figure 6 and 7 for MNIST and ISOLET, respectively.

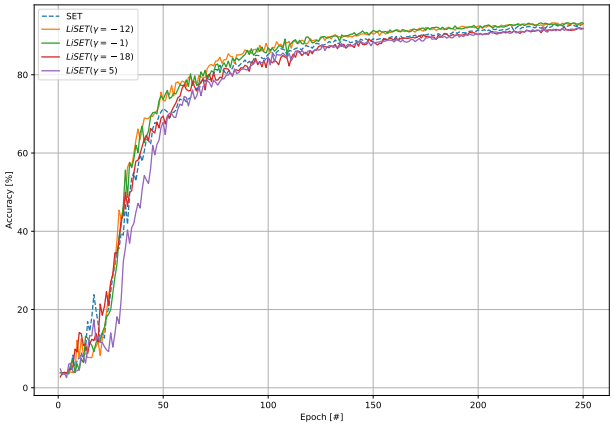
5. DISCUSSION

The experiments showed that LiSET converges faster, especially when higher sparsity was used. MNIST is a dataset that allowed for more extreme sparsity. LiSET managed to outperform SET on accuracy for both levels of sparsity, with it reaching 87.14 accuracy on 97.5% sparsity, while SET attained 85.14. LiSET's outperformance held also on 95% sparsity, where it consistently got better accuracy than the 88.28 accuracy of SET for all of the γ values tried, peaking at 89.87 for γ of 3.

As can be seen in Figure 6, at the beginning of training, LiSET manages to distribute connections to important features faster than SET. It also manages to develop a more consistent accuracy curve. This is best shown by the ARA metric shown in Table 2, with LiSET greatly outperforming SET on it on all tested parameter values except for γ of 5 and lower ϵ. It can be observed that on higher sparsity, LiSET got higher ARA when using the smaller γ values, while getting lower accuracy. This might be due to the algorithm distributing the few connections to very few neurons, which could slow the training on later



(a) Initialized with $\epsilon = 21$



(b) Initialized with $\epsilon = 42$

Figure 7: ISOLET accuracy over epochs using different γ

	$\epsilon = 4.5$				$\epsilon = 9$			
	LiSET		SET		LiSET		SET	
γ	ARA	Accuracy	ARA	Accuracy	ARA	Accuracy	ARA	Accuracy
-24	71.03	83.73			80.0	89.01		
-18	71.05	84.06			79.57	88.76		
-12	70.77	84.13			79.96	88.94		
-8	70.16	84.29			79.71	88.64		
-5	71.06	84.10			79.73	89.12		
-3	69.86	83.87	65.77	85.14	80.12	88.87	75.17	88.28
-1	71.35	84.79			80.20	89.22		
0	74.03	85.41			**81.26**	89.60		
1	**74.13**	86.05			80.89	89.86		
3	72.52	**87.14**			79.57	**89.87**		
5	63.92	84.91			77.55	89.58		

Table 2: Summarization of experiments on MNIST

epochs. The results on MNIST suggest that a γ value of around 0 to 3 might be best on that dataset.

In the ISOLET experiments, LiSET greatly outperformed when lower ϵ was used, reaching 85.70 accuracy where SET got 83.26. On lower sparsity, it reached 93.39 where SET got 92.88. On this dataset, it can be observed that there is a less clear correlation between the values of γ and the evaluation metrics, compared to MNIST. Figure 7 provides a visualization on the accuracy over epochs that suggests that LiSET converges faster on the higher sparsity, but the gap shrinks on the lower. The ARA metric in Table 3 suggests the same detail. For this dataset, it is unclear whether there are γ values that would consistently get the best results due to the lack of correlation mentioned earlier.

The results suggest that there were no certain values of γ that consistently got the best scores on both datasets. On MNIST, values of the parameter close to zero were shown to obtain better results, while on ISOLET small negative values were getting good results as well. If we were to pick a consistent value for γ, the experiments would suggest a value around 0.

As mentioned earlier, LiSET offers some trade-offs. Because of using the skip layers, during training, the number of connections is greater than the vanilla SET equivalent. For each sparse layer using LiSET, the number of connections added is equal to the number of neurons of the previous layer multiplied by the number of classes of the output layer. Thus higher number of classes will lead to more skip layers connections, which adds to complexity during training.

Some of the experiments point to the fact that some values of γ lead to LiSET underperforming on both evaluation metrics. This is best indicated by the values of ARA of LiSET being lower than the value of SET in the same conditions and is better visualized in Figure 7. One reason is that, since the values of γ are biggest in the cases where SET outperforms, LiSET distributes connections almost, if not, completely random. What happens is that, especially at the beginning of training, the skip layers will learn some important features quicker than the main network. Even if these layers get reinitialized on every epoch and dropped out, the main network gets delayed from learning those features. Performance of LiSET worse than SET also indicates that the linear features do not indicate the best neurons to reinitialize connections to.

LiSET provides an appealing alternative to vanilla SET for certain use cases. Faster convergence is a preference when training bigger models on huge amounts of data for example. Due to the duration of the training, these cases might demand fewer epochs of training, so it is important to make the most out of them. The way LiSET adds connections suggests the method also as a useful option when training large and sparse artificial neural networks. The results on MNIST suggest that the proposed method gains accuracy faster on extreme sparsity. When adding connections randomly as does SET, some of the few connections might get reinitialized on insignificant features, while

	$\epsilon = 21$				$\epsilon = 42$			
	LiSET		SET		LiSET		SET	
γ	ARA	Accuracy	ARA	Accuracy	ARA	Accuracy	ARA	Accuracy
-24	62.82	81.91			80.94	93.2		
-18	**64.59**	**85.70**			79.45	91.85		
-12	62.92	83.52			**81.56**	93.01		
-8	63.31	82.30			80.90	92.69		
-5	63.36	84.35			79.89	92.88		
-3	63.54	84.16	60.21	83.26	80.21	92.69	79.99	92.88
-1	63.83	83.77			81.33	93.20		
0	64.41	84.41			81.08	**93.39**		
1	63.96	82.75			80.43	92.56		
3	60.44	83.58			79.16	92.56		
5	58.54	79.67			78.12	91.79		

Table 3: Summarization of experiments on ISOLET

LiSET is able to consider some features as better to add connections to. The presented trade-offs need, of course, to be taken into account when considering the method.

6. CONCLUSIONS AND FUTURE RESEARCH

This research proposed a variant of SET that adds connections during evolution based on linear features. The experiments showed that it helped converge faster on MNIST and ISOLET dataset when using greater sparsity. Different values of γ lead to great out-performance of SET in the specified experiment conditions.

The proposed method involved a few components that were not fully experimented with so there are no general conclusions that can be taken based on the low range of experiments. Experiments shown in this paper have made use of SGD. However, different optimizers might lead to different behavior, depending on how the skip layers develop features. While the experiments shown in this work focused on datasets that are known to have some linear features, it would be interesting to test it on data that provide more non-linear features in order to see how it impacts performance, such as images with background noise.

The experiments were conducted using only multi-layer perceptron architectures. However, other architectures are more efficient nowadays for different uses, for example, convolutional neural networks or recurrent neural networks, and LiSET can be tested on some of them. Different types of architectures will add different constraints to how to apply the method. For example, convolutional layers play a key role in the performance of convolutional neural networks [12], but LiSET can not be applied to those. However, this type of architecture contains fully-connected layers it could be applied on to reduce the number of connections.

LiSET adds some computation complexity, especially because of the skip layers, but there are occasions when the trade-offs are worth it. The user of the method also has the option to reduce the computation complexity through different alterations. When performing sparse-to-sparse training on some datasets, especially with high sparsity, adding connections to linear features could prove to greatly improve convergence speed compared to SET and reach high accuracy faster.

LiSET provides a wide range of elements to alter and experiment with. For example, it has not been tested what is the impact of using the method on the earlier versus the deeper layers of the network. If the method proves to be impactful on all sparse layers, then different levels of epsilon might impact deeper layers differently from early ones.

While LassoNet ranks the importance of global features [14], LiSET made use only of linear features. It could be the case that methods that help find global features could provide better input of where to add new connections during evolution.

7. ACKNOWLEDGEMENTS

I would like to thank my supervisor Elena Mocanu for her feedback, patience and for guiding me in my research.

8. REFERENCES

[1] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[2] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543*, 2013.

[3] S. Dey, K.-W. Huang, P. A. Beerel, and K. M. Chugg. Pre-defined sparse neural networks with hardware acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):332–345, 2019.

[4] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[5] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[6] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[7] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[8] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[9] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. Deep learning with s-shaped rectified linear activation units. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[10] A. Karpathy. Software 2.0.

[11] V. Lapshyna. Sparse artificial neural networks: Adaptive performance-based connectivity inspired by human-brain processes. B.S. thesis, University of Twente, 2020.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[14] I. Lemhadri, F. Ruan, and R. Tibshirani. Lassonet: Neural networks with feature sparsity. In *International Conference on Artificial Intelligence and Statistics*, pages 10–18. PMLR, 2021.

[15] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[16] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[17] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[18] P. M. Murphy. Uci repository of machine learning databases. *ftp:/pub/machine-learning-databaseonics. uci. edu*, 1994.

[19] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[21] J. Pieterse and D. C. Mocanu. Evolving and understanding sparse deep neural networks using cosine similarity. *arXiv preprint arXiv:1903.07138*, 2019.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[23] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.