Regrowing Strategies for Dynamic Sparse Training

HARMEN SCHOLTE (H.T.), University of Twente, The Netherlands

Deep learning yields currently the most, practical results in the field of machine learning. There is however one big problem. Solving real-world, practical problems requires very large neural networks. Networks consisting of millions or even billions of parameters, are being used. The size of these large networks leads to impractical training times and high computation costs. People try to solve these problems by inducing sparsity into the networks, either by pruning (dense-to-sparse training), or by static or dynamic sparse-to-sparse training. Sparse Evolutionary Training (SET) introduced dynamic sparse-to-sparse training, and Adaptive Performance-based Connectivity of SET (AccSET) tries to improve on SET by focusing on the amount of connections that should be regrown. In this research, the regrow step is analyzed further. This research develops strategies regarding (1) which connections should be regrown, (2) on what performance metric should we base the connectivity of SET, and (3) does alternating between static and dynamic sparse training improve the performance. These strategies are tested on a multi-layer perceptron model that classifies images from the CIFAR-10 dataset.

Additional Key Words and Phrases: regrowing strategies, dynamic sparse training, sparse deep learning, scalability, neural networks, sparse evolutionary training

1 INTRODUCTION

Deep learning shows state-of-the-art results in the field of artificial intelligence [17]. It achieves good results in a wide range of fields, including image recognition, speech recognition and natural language processing. But, the inherent structure of deep neural networks has its complications.

Deep neural networks consist of layers with connections (weights). The amount of connections grows quadratically with the number of neurons and layers, which limits the scalability of the network. Contrary to the structure of the human brain [11], neural networks are often fully connected, which means that each neuron is connected with all the neurons from the next layer, which leads to lots of connections, and thus computation costs.

To counteract these problems, Mocanu et al. introduced Sparse Evolutionary Training, a method that dynamically trains sparse-to-sparse networks. It evolves an initial sparse topology into a scale-free topology during training [18]. This means that the method makes use of sparsely connected layers, which drastically reduces the amount of connections, and thus cuts down on computation costs. Lapshyna shows that this amount of connections can be reduced further, by introducing Adaptive Performance-based Connectivity of SET [13]. This paper introduces a method that reduces the amount of regrown weights based on the current accuracy of the network.

TScIT 37, July 8, 2022, Enschede, The Netherlands

1.1 New addition methods

SET and AccSET make use of random weight regrowth, which seems like it should be able to improve upon, so in this research, two alternative methods, based on local connectivity, are considered. The first method has the constraint that the random weight needs to have at least one activated neighbour. SET uses a binary mask to simulate sparsity, and this method looks whether the corresponding weight has at least one activated weight neighbouring the initial weight. The second method regrows weights based on the amount of activated neighbours, so the weights that have the most activated neighbours are regrown.

The hypothesis behind these methods is that regrowing weights that are similar to weights that show to perform well, will also show good results. A parallel can be made to the theory of magnitude-based pruning [10], where weights with low magnitude gets pruned. Likewise, it feels intuitively correct to grow more weights similar to high-magnitude weights, to make the network balance itself out. It feels like, by regrowing weights besides weights with high magnitudes, the high magnitude weights will balance out with the, currently, zero magnitude weights around them, which could potentially improve the performance of the network.

1.2 Adaptive performance based connectivity

Besides these two new methods of picking which connections should be regrown, this paper also tests the theory of Adaptive Performance-based Connectivity of SET, by using different functions to determine the amount of weights to be regrown. This paper introduces **LSET**, which is Adaptive Performance-based Connectivity based on the loss function, instead of the accuracy function from AccSET.

Also, this paper introduces **DynSET**, which alternates between static and dynamic sparse training, based on short-term performance comparisons. This method has the same inner workings as AccSET, but it has the possibility to circumvent the Sparse Evolutionary Training component, and resume normal training, when it sees that the network decreases its performance over short periods of time. This means that when the network's performance decreases, the weight removal and addition functions get bypassed, which leads to the network only being improved by the regular training methods. If the method sees that the performance increases again, the Sparse Evolutionary Training component gets activated again. The idea behind this is that when the, by mutation inspired, techniques of SET do not yield good results, it might be more valuable to just resume normal training, while maintaining the current sparse structure, this makes it (temporarily) more of a training problem, than a search-space problem, which can be considered how SET tries to improve the performance.

2 BACKGROUND

2.1 Artificial neural networks

Neural networks are among the top-performing machine learning methods [23], and are a technique inspired by neuroscience. A neural network is also the biological term for parts of the human brain, so when we discuss neural networks with regard to artificial intelligence, we are actually talking about Artificial Neural Networks (ANNs).

Artificial neural networks consist of three types of layers. There is an input layer, one or several hidden layers, and an output layer. The input layer encodes the inputs that are fed into the network, the hidden layer tries to find the "equation" which corresponds to the, during training provided, input and output pairs [6], and the output layer decodes the information it receives from the hidden layers to try to come up with a logical output. Deep neural networks are neural networks consisting of more than one hidden layer, although definitions vary a bit (some define neural networks only as deep when there are at least three hidden layers [6]).

2.2 Sparse evolutionary training

Sparse training is training with a sparse architecture from the start (sparse-to-sparse training), as opposed to training with a fully connected network (dense architecture) from the start (dense-to-sparse training). Sparse training can be divided into two variants, namely static sparse training (SST), and dynamic sparse training (DST). Static sparse training induces sparsity at the initialization of the training, but, after that, does not change its structure. The structure of the network stays the same during the whole training process.

Dynamic sparse training (as introduced in SET [18]) **does** change its structure. Dynamic sparse training has three steps. One, a random sparse initialization before training (the same as SST). Two, remove a fraction of the weights after each training round (which does not happen with SST). And three, regrow randomly this fraction of weights after the removal (does not happen with SST). As you can see, the last two steps determine the difference between static sparse training and dynamic sparse training.

The methods proposed in this paper will focus on this last step, the addition/regrow step.

2.3 AccSET

AccSET [13] introduced the theory of removing weights during training, and removing more weights as the accuracy increases. This was done using a function which determined how many weights needed to be regrown. The method still removed the same number of weights as normal SET, but the amount of weights that were regrown is different here.

AccSET works as follows:

You basically determine how many weights were removed Δ_e^l, and multiply that by the regrowth ratio θ_e^l, which is inversely based on how accurate the current network is. So if the network has an accuracy of 1, no new weights get added, and if the network has an accuracy of 0, the full amount of removed weights Δ_e^l is regrown.

Algorithm 1 Accuracy SET pseudo-code

$k \leftarrow 0$
for each training epoch e **do**
  perform training
  for each layer l **do**
    remove ζ weights closest to zero
    λ_e^l := current number of connections
    γ^l := initial number of connections
    $\Delta_e^l \leftarrow \gamma^l - \lambda_e^l$
    if e is not the last training epoch **then**
      $\theta_e^l \leftarrow 1 - \frac{(acc^l - acc^l \times k)}{(k - |acc^l| \times 2 \times k + 1)}$
      add $\theta_e^l \times \Delta_e^l$ random connections
    end if
  end for
end for

3 RELATED WORK

3.1 Pruning

Because of the size of ANNs [2, 9] that can give meaningful answers to, for example, image classification, machine translation and text-to-speech problems [9], a while ago (1989), pruning algorithms were invented [3, 21] (Le Cun et al., Mozer & Smolensky). These algorithms had the purpose of cutting away superfluous parameters, weights and neurons from deep neural networks, in order to dramatically decrease the size of these networks. This way, the inference phase could use far smaller networks and decrease memory, computation and energy requirements (dense-to-sparse training) [10, 15] (Han et al., Liu et al.).

Han et al. talks about pruning neural networks as an iterative process at the end of the training phase by looking at the magnitude of weights [10]. This meant that he iterated between training and pruning. After each training round, the weights with lower magnitudes are considered to have less of an impact on the network's performance, and can thus safely be pruned. This theory still holds up after seven years and is used in papers building on top of this foundation.

Frankle & Carbin proposed the "Lottery Ticket Hypothesis", which states that densely connected networks contain subnetworks which perform just as well, when trained from the start [8]. After which Liu et al. [15] found that Frankle & Carbin did not improve over random initialization. Gale et al. [9] finds that simple magnitude-based pruning performs better than more complex methods (variational dropout [19], and L_0 Regularization [16]).

This pruning of ANNs took place after the training phase, which decreased the size of the network for the inference phase, but required the network to be completely connected during the training phase. These techniques are also called **dense-to-sparse** training.

New methods were invented to also decrease the size of the network in the training phase [14] (SNIP), which would, besides inference costs, also decrease training times and costs. These methods fall under the category of **sparse-to-sparse** training.

3.2 Sparse-to-sparse training

Research is currently performed regarding dynamic sparse training, first introduced in SET [18] (Mocanu et al.). Dynamic sparse training, as opposed to static sparse training, has a network that dynamically evolves, through magnitude-based removal and random regrowth. Evci et al. [7] uses the same method as SET, but regrows connections based on the gradient signal, and Dettmers & Zettlemoyer [5] have developed sparse momentum, which uses cosine decay to determine which weights to regrow. Cosine similarity was proposed by Zahra et al. ([1]), which proposed a new method of determining the importance of connections. Mostafa & Wang [20] have proposed dynamic sparse reparameterization as an effective way of training deep convolutional networks.

There is currently also work being done on deep neural networks with dynamic rerouting [22] (Qin et al.), which makes these networks better suitable for environments with computation constraints, as these networks can dynamically switch to less dense structures, decreasing computation costs for the inference phase. More research is also being done on sparse training for scalable and efficient agents [17] (Mocanu et al.), and it gives a very clear overview of the latest methods and their scalability.

4 PROPOSED REGROWTH STRATEGIES

4.1 Local connectivity

In SET, weights are regrown completely at random [18]. Weights are removed based on magnitude, the weights with the lowest magnitude after each training round are removed, but weights/connections are regrown randomly. It should be able to improve on this. In this research, two new algorithms for connection regrowth are created. Both are based on the activity of the connection in the binary mask that overlays the weights, but it should also be able to implement these methods in the "normal" neural network.

4.1.1 One activated neighbour. The first method, regrown weights should have at least one activated neighbour, still works with randomly chosen connections, but in addition, the weights also need to have one activated neighbouring connection in the binary mask. The binary mask is a two-dimensional array that determines which weights are activated in the neural network.

w_i, j exist iff $w\_\{i \mp 1, j \mp 1\} \neq 0$

Fig. 1. Connection (5,5) has 1 activated neighbour

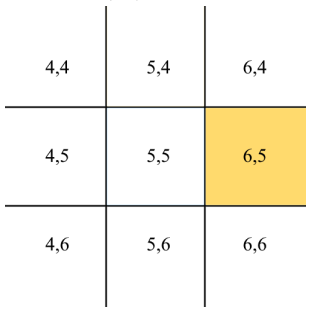


Figure 1 shows that connection (5,5) is not activated, and has 1 activated neighbour, so it gets activated.

4.1.2 Regrow based on most activated neighbours. The second method regrows weights solely based on the amount of activated neighbours they have. If the method calculates that 1000 connections need to be regrown, it sorts all connections based on the local activity, and regrows the first 1000.

4.2 Various performance methods

In AccSET, the accuracy function was used to determine how correct the network was, how good it was performing. But, the loss function can also be used for this purpose. One could argue that the network can still improve its performance while the accuracy decreases, as it gets overall more certain of predicted outcomes, this indicates that the loss function is possibly a better fit for calculating how many weights need to be regrown.

4.2.1 DynSET. Lastly we introduce the DynSET algorithm, which alternates between static and dynamic sparse training, by looking at short-term performance comparisons, these performance comparisons can likewise be done on basis of the loss function, or on basis of the accuracy function.

DynSET works as follows: (Algorithm 2)

Algorithm 2 Dyn SET pseudo-code

```
initLoss ← 0
lossHist = array()
for each training epoch e do
    perform training
    if e = 0 then
        initLoss := currentLoss
    end if
    lossHist.append(currentLoss)
    pauseSet ← 0
    lengthList ← length(lossHist)
    if lengthList > 4 then
        idxOne ← lengthList − 1
        idxTwo ← lengthList − 4
        if lossHist[idxOne] > lossHist[idxTwo] then
            pauseSet := 1
        end if
    end if
    weightsEvolution(pauseSet)
end for
```

This method registers the initial loss, checks if there are enough entries in the loss history to compare the current loss with the loss from five epochs ago, then checks if the current loss is worse than the loss from five epochs ago, and then decides whether or not to pause the Sparse Evolutionary Training for a round. This can of course also be done with the validation function, you only need to check if the accuracy is less, instead of checking if the loss is more.

Table 1. Hyper-parameters for training on CIFAR-10

Parameter	Value
Activation function	LeakyReLU
Optimizer	Stochastic Gradient Descent
Learning rate	0.01
Momentum	0.9
Sparsity level (as discussed in SET)	20
Deletions ratio after training (ζ)	0.3
Batch size	100
k	0

5 RESULTS

The DynSET and LSET algorithms were built on top of the already existing code from SET [18] and AccSET [13]. The code is built on top of TensorFlow and Keras, and adds a binary mask to simulate sparsity. Real sparse implementations are also being created, for example in Curci et al. [4].

Table 1 shows the hyper-parameters that have been used in the experiments done on the CIFAR-10 dataset [12]. This dataset contains images of 10 classes, which divide 6000 images per class. There are 50.000 training images, and 10.000 test images.

5.1 Effect of local connectivity

These are the results from training equal networks with equal parameters on the CIFAR-10 dataset [12], the only difference being the addition method: random addition or addition by the algorithms provided above.

As can be seen from figures 2 and 3, while it is clear that SET outperforms AccSET in terms of accuracy and loss, no significant performance increase can be seen if you compare these two new addition methods with the random addition method from AccSET. So this means that addition based on the activity in the surrounding neighbours, does not increase performance in SET (and AccSET).

Fig. 2. Accuracy of various addition methods on CIFAR-10 (100 epochs)

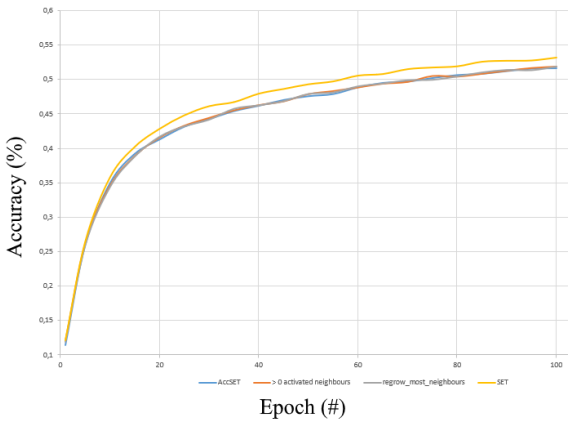
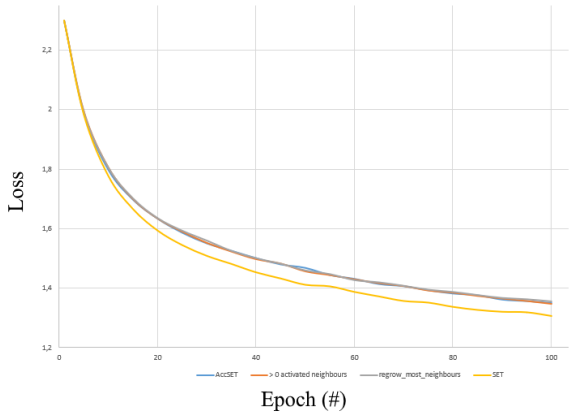


Fig. 3. Loss of various addition methods on CIFAR-10 (100 epochs)



5.2 Effect of various performance methods

Table 2 shows the results of the experiments. The first column shows the main method, the second column specifies how the number of additions are calculated. In the case of SET, this is exactly equal to the number of removals, in the other cases, the number of additions is dependent on a function which uses either the loss, the validation_loss, or the validation_accuracy of the training. The connection amount is reported after additions.

As can be seen in Table 2, the SET method performs the best, getting an accuracy of 0,6682 and a loss of 0,9411. But, it also has the most connections (342493). The algorithm that comes closest to this performance is DynSET, with as the *calculate_amount_of_additions* function, the function provided in Algorithm 1 based on the loss during training. This method has an accuracy of 0,6653 and a loss of 0,9475. But, this method uses only 253916 connections, which is a **25,9% decrease** compared to SET.

5.2.1 Largest connection decrease. The method that manages to reduce the most connections, is the DynSET method based on a function of the validation_accuracy from training. It manages, compared to SET, to reduce the amount of connections with **36,3%**.

This result is very similar to the result of the AccSET method, which yields a similar connection reduction and a similar performance. This makes it clear that the strength of this method is mostly based on "basing the number of connections to be regrown on the validation_accuracy function". It can be observed that DynSET improves a little on LSET, which has the same integral structure, but just does not use the "alternating between static and dynamic sparse training" method.

So, one can conclude from this that basing Adaptive Performance-based Connectivity [13] on the loss function gives the highest performance, and basing it on the accuracy function (or validation_accuracy) reduces the most connections.

5.2.2 Relative performance. We can also look at the relative performance, so, the performance connected to the number of connections. This yields Table 2.

Table 2. Various performance methods results on CIFAR-10 (1000 epochs)

Method	Weight addition by [#]	Accuracy	Loss	Connections [#]	α	β
SET	num. removals	0,6682	0,9411	342493	$1,95 \times 10^{-6}$	322320,16
AccSET	f(val_acc)	0,6494	0,9810	218730	$2,97 \times 10^{-6}$	214574,13
LSET	f(loss)	0,6611	0,9524	254087	$2,60 \times 10^{-6}$	241992,46
DynSET	f(loss)	0,6653	0,9475	253916	$2,62 \times 10^{-6}$	240585,41
$DynSET_2$	f(val_loss)	0,6582	0,9640	242430	$2,72 \times 10^{-6}$	233702,52
$DynSET_3$	f(val_acc)	0,6522	0,9802	218322	$2,99 \times 10^{-6}$	213999,22

α shows the accuracy divided by the amount of connections, so how much accuracy does every connection yield (higher is better), and β shows the loss multiplied by the amount of connections, which says something about the ratio between loss and connections (lower is better). Here you can see that DynSET and AccSET based on the validation_accuracy are the most efficient, they have the best performance/connections ratio. It can also be seen that every method outperforms SET, they are all more efficient (performance/connections wise). Something to notice is that, although the margins are small, DynSET systematically outperforms its "non alternating between static and dynamic" counterparts. DynSET based on f(val_acc) outperforms AccSET (based on f(val_acc)), and DynSET based on f(loss) outperforms LSET (based on f(loss)).

It can also be spotted that f(val_loss) outperforms f(loss), but this is less relevant, as this is dependent on the training and validation data, which will be different for every dataset. Not always will the validation data be structurally easier to predict than the training data.

5.2.3 Graphs. SET outperforms the other methods in terms of accuracy; DynSET based on the loss function and LSET are the better performing methods of the other methods, this can be seen in the zoomed-in section of the graph, which shows the last 100 epochs.

In terms of loss, exactly the same can be seen. SET outperforms the other methods, and the zoomed-in graph, which shows the last 100 epochs, shows that DynSET based on the loss function and LSET are the better performing methods of the other methods. DynSET based on the validation_accuracy function and AccSET are the worst performing methods.

Finally, the connection amounts. It is basically the inverse of the performance graphs. SET is straight on top, having a fixed amount of connections (connections are reported after regrowth). DynSET based on the validation accuracy and AccSET perform the best, having the lowest amounts of connections, which are nearly identical, not strange given the fact that they use the same function to determine how many connections should be regrown. And finally, LSET performs the worst from the other methods. DynSET based on the loss function could unfortunately not be added in this graph, as the amount of connections was not reported during the experiment. But, given that DynSET_validation_accuracy and AccSET perform identical with the same function to determine the amount of weights to be regrown, we can assume that DynSET_loss would perform similarly to LSET (so not very good in terms of connection removal).

Fig. 4. Accuracy of various performance methods on CIFAR-10 (1000 epochs)

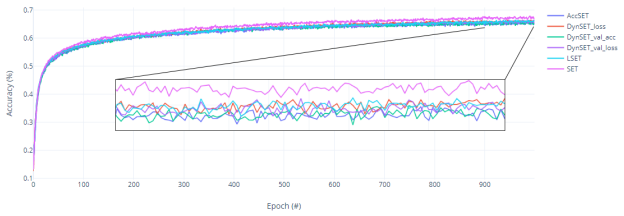


Fig. 5. Loss of various performance methods on CIFAR-10 (1000 epochs)

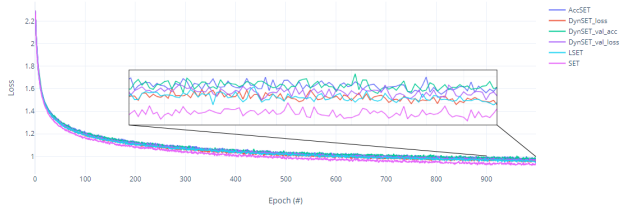
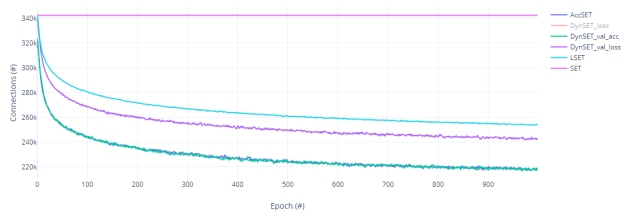


Fig. 6. Connections of various performance methods on CIFAR-10 (1000 epochs)



6 CONCLUSION AND FUTURE WORK

Firstly, Sparse Evolutionary Training [18] makes training neural networks on classification problems way more efficient. The 342.493 connections that it needs here, are a big reduction compared to the 20.328.000 connections that a normal multi-layer perceptron algorithm, using the same architecture, needs [13] (a reduction of 98,3%). Secondly, AccSET, with the validation_accuracy function as guideline to determine how many weights need to be regrown, makes the training even more efficient (Table 2). Thirdly, other functions can also be used as guideline to determine how many weights need to be regrown, and basing it on the loss function gives, after SET, the best performance (Table 2), while still reducing the connection amount with 25.9% compared to SET. Finally, DynSET, alternating between static and dynamic sparse training, performs structurally a tiny bit better than the non-alternating counterparts, making it the most efficient algorithm discovered during this research (Table 2).

The two new addition methods introduced in this research did not perform better than its random regrowth counterpart, but it is still the assumption of this research that there should be a way to improve on this method.

This research has not compared SET to DynSET, so DynSET without the Adaptive Performance-based Connectivity component. This is something that can be done, to check if the method also provides value in this scenario.

Also, the AccSET method used in this paper does only decrease the amount of connections. It might be possible that there is value in also having the possibility to regrow more connections than the amount of connections that were removed, to really determine what the optimal amount of connections is.

Finally, it should probably be possible to improve on random regrowth. It might be that Sparse Evolutionary Training performs so well, precisely because it uses random regrowth, but intuitively it should still be possible to improve on this.

ACKNOWLEDGMENTS

I would like to thank Dr. Elena Mocanu for her support, attentiveness and help.

REFERENCES

[1] Zahra Atashgahi, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond N. J. Veldhuis, and Mykola Pechenizkiy. 2019. A Brain-inspired Algorithm for Training Highly Sparse Neural Networks.

[2] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. 2013. Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 1337–1345. https://proceedings.mlr.press/v28/coates13.html

[3] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 598–605.

[4] Selima Curci, Decebal Constantin Mocanu, and Mykola Pechenizkiy. 2021. Truly Sparse Neural Networks at Scale. *CoRR* abs/2102.01732 (2021). arXiv:2102.01732 https://arxiv.org/abs/2102.01732

[5] Tim Dettmers and Luke Zettlemoyer. 2019. Sparse Networks from Scratch: Faster Training without Losing Performance. *CoRR* abs/1907.04840 (2019). arXiv:1907.04840 http://arxiv.org/abs/1907.04840

[6] IBM Cloud Education. 2020. *Neural Networks*. IBM Cloud Education. Retrieved May 5, 2022 from https://www.ibm.com/cloud/learn/neural-networks

[7] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the Lottery: Making All Tickets Winners. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 2943–2952. https://proceedings.mlr.press/v119/evci20a.html

[8] Jonathan Frankle and Michael Carbin. 2018. The Lottery Ticket Hypothesis: Training Pruned Neural Networks. *CoRR* abs/1803.03635 (2018). arXiv:1803.03635 http://arxiv.org/abs/1803.03635

[9] Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The State of Sparsity in Deep Neural Networks. *CoRR* abs/1902.09574 (2019). arXiv:1902.09574 http://arxiv.org/abs/1902.09574

[10] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. *CoRR* abs/1506.02626 (2015). arXiv:1506.02626 http://arxiv.org/abs/1506.02626

[11] Suzana Herculano-Houzel, Bruno Mota, Peiyan Wong, and Jon H. Kaas. 2010. Connectivity-driven white matter scaling and folding in primate cerebral cortex. *Proceedings of the National Academy of Sciences* 107, 44 (2010), 19008–19013. https://doi.org/10.1073/pnas.1012590107 arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1012590107

[12] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (05 2009), 32–35.

[13] Viktoriia Lapshyna. 2020. Sparse artificial neural networks : Adaptive performance-based connectivity inspired by human-brain processes. http://essay.utwente.nl/80559/

[14] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2018. SNIP: Single-shot Network Pruning based on Connection Sensitivity. *CoRR* abs/1810.02340 (2018). arXiv:1810.02340 http://arxiv.org/abs/1810.02340

[15] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the Value of Network Pruning. *CoRR* abs/1810.05270 (2018). arXiv:1810.05270 http://arxiv.org/abs/1810.05270

[16] Christos Louizos, Max Welling, and Diederik P. Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. https://doi.org/10.48550/ARXIV.1712.01312

[17] Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H. Nguyen, Madeleine Gibescu, Damien Ernst, and Zita A. Vale. 2021. Sparse Training Theory for Scalable and Efficient Agents. *CoRR* abs/2103.01636 (2021). arXiv:2103.01636 https://arxiv.org/abs/2103.01636

[18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. 2017. Evolutionary Training of Sparse Artificial Neural Networks: A Network Science Perspective. *CoRR* abs/1707.04780 (2017). arXiv:1707.04780 http://arxiv.org/abs/1707.04780

[19] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational Dropout Sparsifies Deep Neural Networks. https://doi.org/10.48550/ARXIV.1701.05369

[20] Hesham Mostafa and Xin Wang. 2019. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 4646–4655. https://proceedings.mlr.press/v97/mostafa19a.html

[21] Michael C Mozer and Paul Smolensky. 1989. Using relevance to reduce network size automatically. *Connection Science* 1, 1 (1989), 3–16.

[22] Minghai Qin, Tianyun Zhang, Fei Sun, Yen kuang Chen, Makan Fardad, Yanzhi Wang, and Yuan Xie. 2021. Compact Multi-level Sparse Neural Networks with Input Independent Dynamic Rerouting. *ArXiv* abs/2112.10930 (2021).

[23] Iqbal H Sarker. 2021. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science* 2, 3 (2021), 1–21.