Structured Adversarial Rewiring for Robust and Sparse Neural Networks

Benedikt Seidel

Institute of Theoretical Computer Science

Bachelor Thesis Biomedical Engineering

Winter Semester 2022/23

Abstract

Adversarial robust sparse neural networks have been shown to perform astonishingly close to their dense counterparts. Meeting hardware limitations in real world applications, their robustness, performance, and further optimisation for hardware are of main interest. We build up further on the paper *Training Adversarially Robust Sparse Networks via Bayesian Connectivity Sampling* by (Özdenizci and Legenstein, 2021) where the neural networks evolve during the adversarial training process using Bayesian connectivity sampling to converge to a optimal solution given a sparsity constraint. Based on these principles we here applied a structured rewiring approach using the minimal euclidean distance, which is also known as the l2-norm, to further increase efficiency. Using this new approach, the reinitialisation of weights, which have a value smaller or equal to zero, thus symbolising a dead-end of information flow, occurs close to active weights (values larger than 0). Ultimately, this process causes enforcement of a structure that optimises performance in terms of calculation speed on hardware while still letting the network evolve in a Bayes-optimal manner.

1 Introduction

Deep neural networks are currently being successfully implemented and widely used in various fields. Deep neural networks are more and more integrated into everyday life, therefore their intended usage from before, running primarily on highly performant hardware, shifts into the everyday consumer market. This poses two fundamental problems. Firstly with neural networks being used more widely, the impact of malicious attacks on aforementioned networks, is growing rapidly everyday. Secondly the computing power of mobile devices, laptops and also cars where deep neural networks are implemented is restricted not only due to cost but also due to the efficient use of power. As demonstrated by (Szegedy et al., 2013) a dense deep neural network can be fooled by adversarial attacks. These attacks maximise the error function by applying hardly perceptible perturbations to the input so that the input will be misclassified. An example where a Gaussian noise, almost imperceptible to the human eye, was applied to the picture of a panda until it was misclassified as a gibbon was given by (Chan, 2018).

With previously discussed hardware constraints, sparse neural networks are recently getting more appreciation. Although there is some improvement in robustness by decreasing the density, this however does not apply for very sparse models where the connectivity is often lower then 10% (which in this work we will just refer to as sparse models). Robustness-aware pruning, as demonstrated by (Sehwag et al., 2019) has been successful, but this implies pre-training a robust dense model beforehand. In the work by (Özdenizci and Legenstein, 2021) a method for end-to-end sparse training of neural networks with robust adversarial training objectives is

presented. We will use and further elaborate on this model. Based on the synaptic connectivity structure of the brain, this approach uses Bayesian connectivity sampling, during model training with robust learning objectives. This is achieved by estimating a posterior distribution that combines the robust training objective with a sparse connectivity prior on the network parameters in a Bayes optimal manner. The network parameters are sampled simultaneously with the sparse connectivity structure from the posterior allowing the network to self construct during training (Özdenizci and Legenstein, 2021).

This important combination of model sparsity and adversarial robustness is further developed in favour of computation speed by using a structured sparsity approach. Structured sparsity improves computation speed. We are starting off with a random distribution of weights that still evolves using Bayesian connectivity sampling, but the redistribution of lost weights happens by reinitialising these weights close to existing weights. These existing weights can be sampled either randomly or based on the values of the weights.

2 Background

2.1 Adversarial Robustness

As previously mentioned, adversarial attacks perturb the input to a neural network, in such a way that it is hardly perceptible by a human, but still maximises the error function, thus fooling the neural network. In recent years a lot of defence methods against adversarial attacks have been developed, often running into the same problem of defending against one or more attacks not solving problems, but not finding a general solution to adversarial attacks. The most effective way against unknown attacks is to drive the model more to the essence of what the input is supposed to be, rather than preventing attackers to abuse a models naivety. The most successful approach in that direction is to train the model on adversarial examples, thus increasing the robustness on them (Athalye et al., 2018).

2.1.1 Projected Gradient Descent

A quite common attack which is used to evaluate a worst case where the attacker, has access to the gradients of the model is the *projected gradient descent* (PGD), this was also used in our work to evaluate the robustness of our model (a *VGG11* that uses the *CIFAR-10* dataset to classify images). Projected gradient descent attack describes an attack where the perturbation is kept as small as specified/possible and the attack is optimised in such a way that the perturbation increases the loss function of the model so that it misclassifies the input.

2.1.2 TRADES

In order to prepare models for such attacks multiple approaches have been used, but the TRADES (TRadeoff-inspired Adversarial DEfense via Surrogate-loss minimization) is one of the state-of-the-art loss functions to do so (Zhang et al., 2019), that further refines the following loss function Equation 1 (where θ corresponds to the learnable parameters).

$$\mathcal{L}_{natural} = -\log p(y|x,\theta) \tag{1}$$

This procedure can be described as following by (Özdenizci and Legenstein, 2021): *"State-of-the-art robust learning objectives mainly rely on the TRADES loss proposed by (Zhang et al., 2019) with $\mathcal{L}_{robust}$ in Equation 2 defined via the regularized loss:*

$$\mathcal{L}_{TRADES} = \mathcal{L}_{natural} + \beta \cdot D_{KL}(p(y|x,\theta)||(p(y|\tilde{x},\theta)) \tag{2}$$

where β is the robustness-accuracy tradeoff parameter and $\tilde{x}$ is obtained by PGD using $D_{KL}(p(y|x,\theta)||(p(y|\tilde{x},\theta))$ as the loss to be maximized in the inner optimization step."

2.2 Sparse Neural Networks

Deep neural networks are being integrated more and more into everyday use and give rise to the problem of the integration with everyday hardware. To power a deep neural network, a lot of complicated and highly power consuming computations are required which pose a problem on battery-powered hardware with computing bottlenecks (Capra et al., 2020).

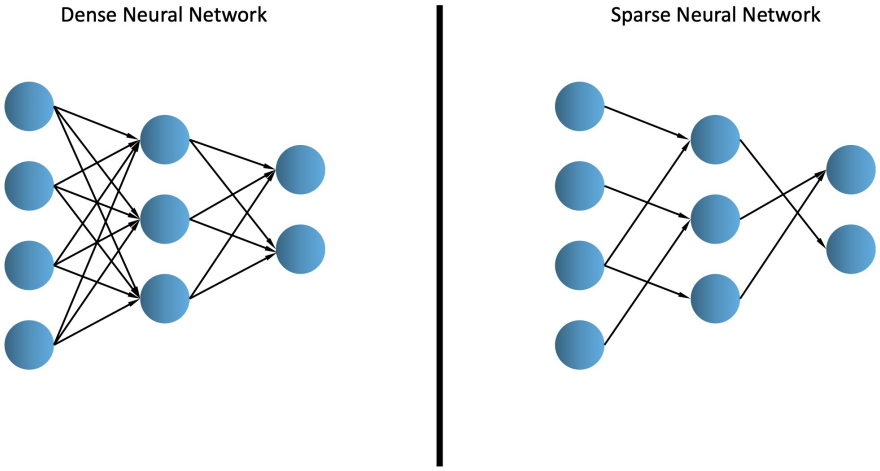

Figure 1: Depiction of a dense versus a sparse neural network

The goal of circumventing such energy and/or memory intensive operations gave rise to various efficient implementations of neural networks with sparse neural networks (see Figure 1) being at the pinnacle of interest. A sparse model can be thought of as a dense model where a number of weights has been set to zero (Géron, 2019, p. 359). If weights or even whole tensors are zeroed out, we save a lot of computation time. Interestingly in the work of (Özdenizci and Legenstein, 2021) we saw an astonishingly similar performance of sparse neural networks to their dense counterparts.

2.2.1 Unstructured Sparsity

As of right now most sparse neural networks are a derivative of a previously dense neural network, that has then been pruned for the sake of sparsity. This is the reason why the literature primarily focuses on structured and unstructured pruning of weights of dense neural networks. As shown in the work by (Narang et al., 2017) a very common and useful approach of model compression is unstructured-weight-pruning, which significantly cuts down the model size while at the same time keeping performance high. Unfortunately it proves difficult to support unstructured sparsity efficiently on common hardware (Han et al., 2016).

In the paper of (Özdenizci and Legenstein, 2021) we saw unstructured sparsity being implemented in a Bayes optimal manner using Bayesian connectivity sampling during adversarial training. Weights that decreased their value to zero or below during training, thus not carrying any valuable information further get reinitialised at new random locations. The spots for the "new" weights, in the sense of reinitialisation, get sampled randomly from the available places (a coordinate in a multidimensional layer where there is no connection / the weight in this place has a value of 0). Thus, the unstructured approach using Bayesian connectivity sampling ensures that the network converges to a Bayes optimal implementation. From unstructured

sparsity one can expect the best performance, but we still need to make a few extra computations for vectors that maybe only contain a single weight (the weights are not structured optimally in terms of computation speed).

2.2.2 Bayesian Connectivity Sampling

A sparse neural network can, as mentioned above, either be achieved by training a dense neural network which is pruned after the training is done, or as it is used in the work of (Özdenizci and Legenstein, 2021) and this work, one adds a sparsity constraint during training so that the network is already trained sparsely. In this work we focus on the Bayesian connectivity sampling approach. This approach is based on network rewiring which was introduced in the works (Bellec et al., 2017), (Kappel et al., 2017)and (Kappel et al., 2015) . It is based on the belief of the dynamic synaptic rewiring in the brain and uses a Bayesian prior (*sparse connectivity prior*) in order to enforce a sparsity belief. This sparsity belief enables the dynamic synaptic rewiring of weights during training in a Bayes optimal manner.

2.2.3 Structured Sparsity

As described above, unstructured sparsity performs great, but it is not the optimal way to implement a sparse neural network with regards to its computational speed. Studies by (Wang et al., 2020) showed that a structured pruning approach *"can provide significant speedups and compression rates on large models while losing minimal performance compared to other methods, including unstructured magnitude pruning."*(Wang et al., 2020) In layman's terms, we could say it is way easier to use hardware acceleration on densely packed tensors and additionally it is faster not having to compute values for tensors which are only containing zeros.

3 Training Adversarially Robust Networks with Structured Sparse Connectivity

3.1 Design Motivation

As previously shown by (Özdenizci and Legenstein, 2021) we are still motivated by the dynamic synaptic connectivity structure of the brain. Furthermore we want the neural network not only to evolve in a Bayes optimal manner using Bayesian connectivity sampling, but to also use the significant speedups that were demonstrated by (Wang et al., 2020) of a structured sparsity model. This lead us to the idea of a neural network that evolves with a sparsity belief as a prior, but also reinitialises the weights with some structure in mind to later make it easier to compress the model, therefore providing a blazingly fast deployment of the sparse model.

3.2 Rewiring of Weights optimising the Euclidean distance

The evolvement of a neural network provided by (Özdenizci and Legenstein, 2021) was taken and it starts with a random initialisation of the weights that are used (e.g. 5% corresponding to 95% sparsity). This is then sent through the same training steps as proposed by (Özdenizci and Legenstein, 2021). During training some weights will decrease their value to 0 or even contain negative values, thus presenting a dead end of information flow. Truly inspired by the human brain this weight, or neural connection is eliminated and reinitialised somewhere else at random. This reinitialisation with a pre-decided value for the weight acts sort of like a seed that gives the flow of information a possibility to flow through it and therefore growing the value of this weight in a new location.

That being a great way to let the network evolve in a Bayes-optimal manner keeping in mind the prior of the sparsity belief, it will by design not be a structured manner. That is exactly the reason why we decided to reinitialise the new weights at points close to weights

that still let information flow. We do that by sampling the number of weights that need to be reconnected from the weights that are still larger than zero, calculate their next neighbour (X in Equation 3) using the minimal euclidean distance and use their locations for reinitialisation:

$$X_{NearestNeighbour}(p) = \arg\min_{\chi \epsilon A} \sum_{n=1}^{D} |p_n^2 - \chi_n^2| \tag{3}$$

where D corresponds to the dimensions of the layer/space, p stands for the coordinates of one of the points sampled from the still connected weights and χ describes the coordinates of an unconnected weight / available coordinates (A) to rewire.

That way we let the model converge to a dense subset of weights with the rest of the values optimally being all zero. If there are still unstructured weights at the end of training we could still, further compress the network using a structured pruning approach.

3.2.1 Weight-Based Rewiring

In order to ensure faster convergence we made use of the notion, that the weights, with the highest value were forwarding the most information, and therefore would normally spread their load to their surrounding neighbours. To ensure a faster convergence instead of finding the nearest neighbours of random connected weights, we also implemented a further parameter that, if activated, would be used to calculate the n-nearest neighbours of the n-highest weights, thereby converging faster. This would mean that we arrive at the final structure where the Bayesian connectivity sampling approach and the structured rewiring would balance each other out will be reached in less training time.

Algorithm 1 Structured Rewiring

Input: neural network f, parameters Θ, number of parameters to reconnect n
connected coordinates = where $\Theta > 0$
unconnected coordinates = where $\Theta < 0$
τ = sort unconnected coordinates according to euclidean distance in Equation 3
if weight based == True **then**
  sample = take n points of connected coordinates according to the largest values in Θ
else
  sample = take n random points from connected coordinates
end if
η = lookup closest neighbours of sample in τ
Rewire f at η
return Rewired f

Expected outcomes of the euclidean distance minimisation

The minimal euclidean distance (l2-norm) in itself is not the optimal way to structure a complex multidimensional layer of a neural network, considering that we essentially create a conglomerate in form of hypersphere. The optimum would be to have a structure according to the structure of the layer, where one would fill the lowest dimensional vectors either completely with weights or with zeros and further enforce the same in the next higher dimension. Doing some preliminary tests this can easily inhibit the evolvement and the performance of the network due to multiple reasons. Creating one or multiple hyperspheres, however, does not cause the same problems as we in every training step do not force the network to have some vectors completely filled and others to be completely zeroed out, but rather we lead the weights that are to be rewired, to rewire according to Equation 3 where they can rewire throughout different dimensions within the layer. A visual example in 2D can be seen in Figure 2.

$$\begin{bmatrix} [0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 0] \\ [0 & \theta_1 & \theta_2 & 0] \\ [\theta_3 & \theta_4 & \theta_5 & \theta_6] \\ [\theta_7 & \theta_8 & \theta_9 & \theta_{10}] \\ [0 & \theta_{11} & \theta_{12} & 0] \\ [0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 0] \end{bmatrix} \neq \begin{bmatrix} [\theta_1 & \theta_2 & \theta_3 & \theta_4] \\ [0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 0] \\ [\theta_5 & \theta_6 & \theta_7 & \theta_8] \\ [0 & 0 & 0 & 0] \\ [\theta_9 & \theta_{10} & \theta_{11} & \theta_{12}] \\ [0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 0] \end{bmatrix}$$

Figure 2: A visual representation of the ideal structure (right) and our enforced hypersphere (left)

Using the enforcement of a minimal euclidean distance (l2-norm) between newly initialised weights and prevalent weights could counteract the Bayes optimal evolvement that was previously happening. There are 3 possible outcomes which one can expect by using our algorithm. These outcomes are highly dependent on the sparsity constraint one enforces during training and the belief that ideal structure compromises model performance.

1. **The structure does not converge:**
   We have a relatively high connectivity for a sparse model (i.e. 10% connectivity) this would cause an alternation between the Bayes optimal evolvement and the l2-norm, with probably a decent performance in the tasks the model has to perform but not an ideal structure. In layman's terms one could say that the weights would just jump between clusters of non-zero weights.

2. **Solely the structure converges:**
   We have a low connectivity (i.e. 0.05% connectivity) and as there are less weights it could be that every weight takes a turn in dying off during training as the model is too sparse. This is expected to result in a perfect hypersphere as convergence of the structure is reached comparably faster (Once a weight dies off, it is rewired according to the l2-norm and can therefore never go back to a random location without any neighbouring weights). The model would get stuck in performance as there are not enough weights to learn the task fast or good enough before the structure enforcement overpowers the Bayes optimal evolvement.

3. **Optimal evolvement:**
   Based on the very first initialisation of weights and an optimal, therefore unknown connectivity constraint, the structuredness and task performance converge at the same speed. We would arrive at a model which in itself contains one or multiple (way fewer than in case 1.) almost dense substructures, as the rest of the model is only containing zero weights. This would be optimal, due to the reason that our model in the lowest dimension (i.e. in a 2D layer the 1D tensors that in total make up the 2D layer), would be densely filled with parameters, therefore leaving computation speed for these vectors as with tensors in ordinary dense neural networks. In turn this would leave as many other tensors as feasible considering a hyperspherical evolvement remaining with only zeros. The dense part and the outer rim of each hypersphere would then be computed as previously practiced but the largest part could be completely left out as they do not contain any non-zero values.

It is to be stated, that under the premise that the models performance in executing the designed task is sufficiently, one could further fine-tune the model making it even sparser using structured pruning approaches, as the model is already inherently more structured in smaller substructures than other models.

4 Experimental Results

Our experiments were conducted as previously mentioned, with a *VGG11* model. The *VGG* architecture was introduced in the work of (Simonyan and Zisserman, 2014). We used the *VGG11* to train and classify the *CIFAR-10* dataset. We further used following configurations as previously used by (Özdenizci and Legenstein, 2021):

- **Optimization:** SGD with momentum and decoupled weight decay (Loshchilov and Hutter, 2017).
- **Epochs and Batch Size:** The models were trained for 100 epochs with a batch size of 128.
- **Default Learning Rate:** Piecewise constant decay learning rate and weight decay schedulers, with a initial learning rate of 0.1.
- **Weight Initialization:** Kaiming initialization (He et al., 2015).

4.1 Structure Results

In order to assess the structure we have visual representations of the weights in a 2D layer in form of a matrix plot (see Figure 3-Figure 11), where for every weight larger then 0 one can find a yellow square corresponding to this used weight. In Figure 3-Figure 11 we have the unstructured weight-matrix on the left, where we can see the weights (yellow squares) quite evenly and independently spread across the whole layer. In the middle of the mentioned figures one can find the structured approach using minimal euclidean distance, whereas on the right the weight based minimal euclidean distance approach as discussed previously was used.

Additionally a more quantitative approach to present the structure lies in the total percentage of the lowest dimensional vectors (the single vectors that make up a 2D layer which in turn could make up a 4D layer etc., see Figure 2 for an example) that contain only zeros (*zero vectors*). This percentage of lowest dimensional vectors containing only zeros is shown in Table 1. In order to furthermore refine the structure within the different layers we pruned all weights that singlehandedly occupied a vector. These percentage values can be found in Table 1 as well under *After pruning*.

4.1.1 Weight-matrix analysis

In Figure 3-Figure 5 we compared the different evolvement of the weight structure for a network that in total is 99% sparse As previously mentioned from the left to the right one can clearly see a difference. On the very left all the weights are unstructured, so they are evenly spread across the whole layer, occupying almost the whole layer and only leaving 2.066% (see Table 1) of vectors completely untouched. After pruning / zeroing all the vectors only containing a single weight, we arrived at 9.48%. As one can see in Figure 4 we have a drastic improvement of visual structuredness and the different weights form sort of flakes, because the algorithm did not converge in training time to a state where a few of these flakes start growing to a hypersphere/circle. The numbers in Table 1 do agree with our observation, but due to the fact that we do not fill the vectors up ideally and rather create conglomerates of weights, we still end up with only 7.45% of zero vectors. On the other hand if we now prune the weights in single weights vectors, we arrive at 19.145% of zero vectors. Similarly the weight-based structuring approach (see Figure 4) yielded 11.77% of zero vectors and after pruning we arrived at 24.05%. As this highly sparse network achieved the best structured result during our experiments, we further added Figure 12-Figure 14 depicting layer 8, where we found that the number of zero vectors changed from 6.25% up to 22.25%. After pruning the unstructured layer contained 22.51% zero vectors, which was then topped by the weight-based structured layer containing 47.41% zero vectors.

In Figure 6-Figure 11 one can compare the different weight structures of two networks that in total are 95% sparse. Specifically in Figure 6-Figure 8 we trained the network with a natural training loss, whereas in Figure 9-Figure 11 we trained it via the TRADES loss to make it adversarially robust. Already with 95% sparsity one can already see in the structured and weight-based structured figures, that an ideal structuredness gets harder to achieve as we have not larger singular clusters but many more clusters, which appear larger in the normal minimal euclidean-distance approach, but in the weight-based approach the main difference is just that there are many more. This makes it a lot harder to arrive at an ideally structured solution as a lot of dense medium sized clusters are spread throughout the respective layer maybe visually structured but can take a turn in occupying mostly vectors filled with zeros (see Table 1). Thus resulting in non-ideal distribution in terms of computation speed and even after pruning there is not a huge change as the images would suggest.

Parameters		Amount of lowest dimensional zero tensors		
Sparsity	Train-type	Unstructured	Structured	Structured weight-based
99%	natural	2.066%	7.45%	11.77%
95%	natural	0%	0.94%	1.086%
95%	TRADES	0.018%	0.57%	1.06%
After pruning				
∼99%	natural	9.48%	19.145%	24.05%
∼95%	TRADES	0.14%	1.125%	1.72%

Table 1: Percentage lowest dimensional vectors containing only zeros (*zero vectors*)

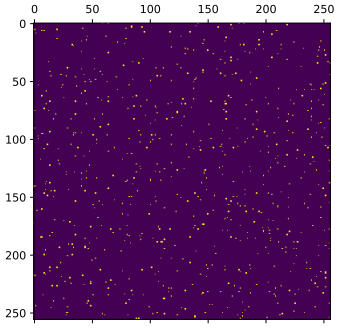

Figure 3: Unstructured weight matrix of a 2D layer where the complete network is 99% sparse

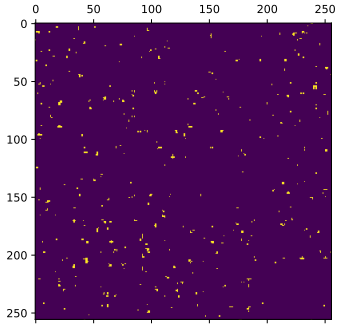

Figure 4: Structured weight matrix of a 2D layer where the complete network is 99% sparse

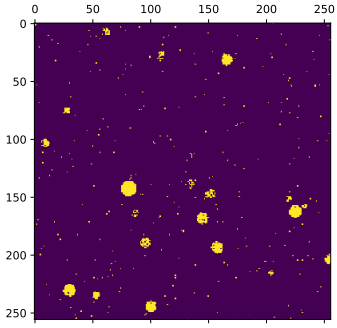

Figure 5: Weight-based structured weight matrix of a 2D layer where the complete network is 99% sparse

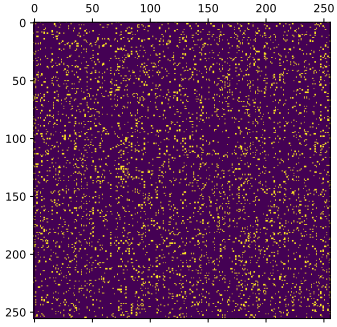

Figure 6: Unstructured weight matrix of a 2D layer where the complete network is 95% sparse

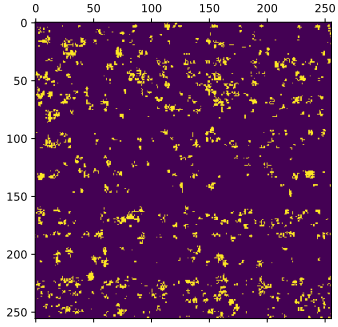

Figure 7: Structured weight matrix of a 2D layer where the complete network is 95% sparse

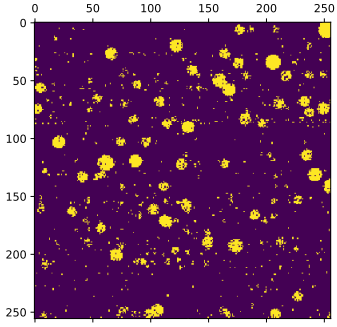

Figure 8: Weight-based structured weight matrix of a 2D layer where the complete network is 95% sparse

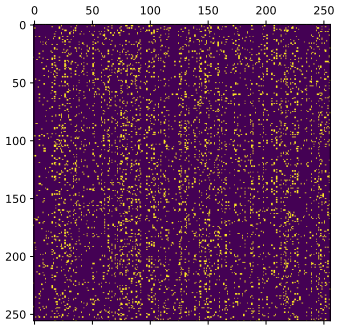

Figure 9: Unstructured weight matrix of a 2D layer where the complete network is 95% sparse and was trained adversarially robust using the TRADES loss

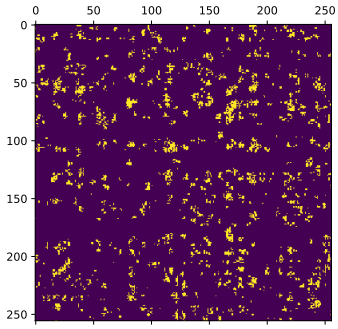

Figure 10: Structured weight matrix of a 2D layer where the complete network is 95% sparse and was trained adversarially robust using the TRADES loss

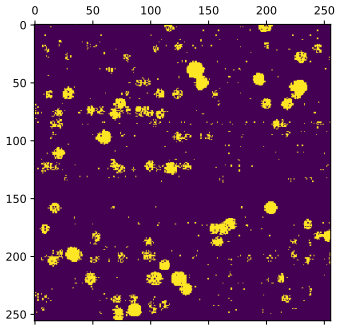

Figure 11: Weight-based structured weight matrix of a 2D layer where the complete network is 95% sparse and was trained adversarially robust using the TRADES loss

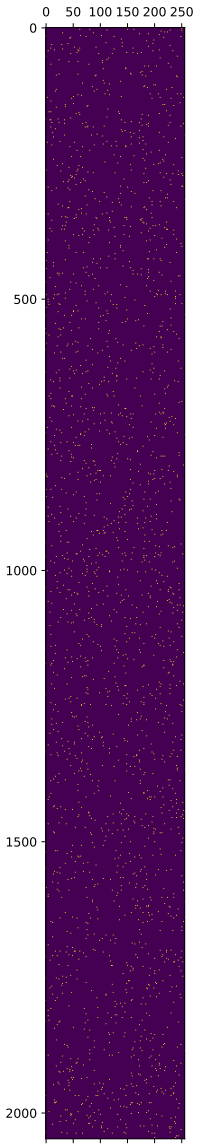

Figure 12: **99% sparse Network** - Unstructured layer 8 containing 6.25% zero vectors

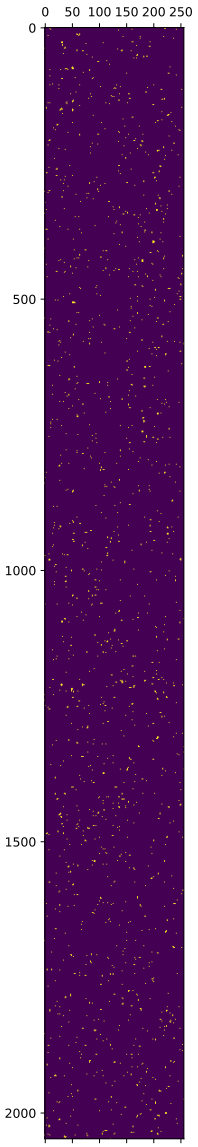

Figure 13: **99% sparse Network** - Structured layer 8 containing 15.87% zero vectors

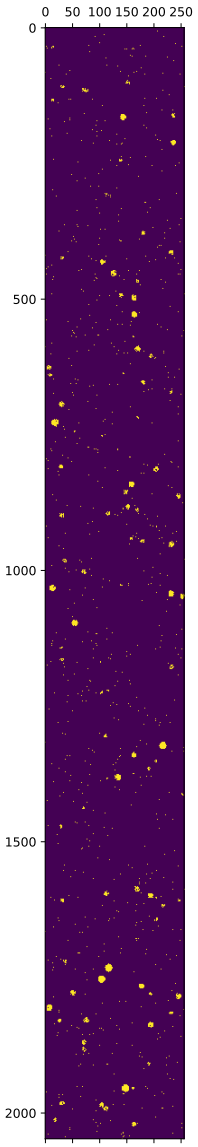

Figure 14: **99% sparse Network** - Weight-based structured layer 8 containing 22.56% zero vectors

4.2 Performance results

4.2.1 Conventional test accuracy

As interesting as sparsity and structured sparsity and how to evaluate those is, we still need to assess the accuracy on the task they were trained to perform. In our case we trained a *VGG11* that was used to classify the *CIFAR-10* dataset. In Table 2 we offer a comparison of the models test accuracy. Expectedly the adversarially robust model with 95% sparsity performed the worst as it can generalise better the specificity gets worse. The non robust model with 95% sparsity performed the best as it has a lot more weights as the 99% sparse network and has a higher specificity.

Interestingly we found that the structuring algorithm did not really dent the test accuracy of the network, and that after pruning the weight-based structured 99% sparse network to the 24.05% strict zero vectors, we still achieved a test accuracy of 73.11%. The non weight-based structuring approach only achieved a performance, of 58.29%. This discrepancy is most certainly due to the fact that in the weight-based approach we rewire new weights around weights with a higher value (a higher value weight contributes by definition more to the networks output as the abstracted input is multiplied to a corresponding higher value at this weight) there is a low probability of these weights solely occupying a vector that else contains zeros. However, in the normal structured approach a weight with a constant high-value will not slowly die off and therefore will not migrate towards a weight cluster and a weight cluster does not need to evolve around a high-value weight in this approach. So some high value weights will be left solely occupying vectors and therefore the network will perform worse after pruning using this approach.

To summarize one could say, that with the weight based approach we build clusters around high value weights and therefore when it comes to pruning we can more easily prune with a focus on structuredness as we already took care that the high value weights cannot be as easily deleted during pruning. The high performance of the pruned unstructured network is firstly because we pruned less than half of the weights of the weight-based structuring approach and secondly the high value weights are scattered evenly through the network so they are not any more prone to being pruned.

4.2.2 Robustness under PGD adversarial attacks

Robust accuracies are calculated under state-of-the-art *projected gradient descent* (PGD) attacks with 50 steps and 10 randomized restarts using the *Foolbox* implementation (Rauber et al., 2020). In the comparison of the unstructured approach of (Özdenizci and Legenstein, 2021) to different structured approaches it is great to see that the adversarial robustness does not change when a structured rewiring approach is used. This can be verified in Table 3 where the robust generalization gap (being the absolute difference in accurracy between the clean exam-

Parameters		Test accuracy		
Sparsity	Train-type	Unstructured	Structured	Structured weight-based
99%	natural	84.89%	84.91%	84.44%
95%	natural	87.7%	87.8%	87.6%
95%	TRADES	71.21%	71.01%	70.21%
After pruning				
∼99%	natural	76.12%	58.29%	73.11%
∼95%	TRADES	58.85%	46.76%	64.11%

Table 2: Network performance by connectivity, structuredness and type of training

ples and the adverserial examples) is almost the same for each approach. This should conclude that we can still maintain the robustness of the models while enforcing a structure. The model in Table 3 was trained using the TRADES loss as defined in Equation 2 with $\beta = 0.6$.

Parameters		Test accuracy on clean examples / Robust accuracy		
Sparsity	Train-type	Unstructured	Structured	Structured weight-based
95%	TRADES	71.19% / 40.74%	71.06% / 40.61%	69.98% / 39.46%

Table 3: Test accuracy compared to robustness under PGD adversarial attacks

5 Discussion

The results of this work allow the conclusion that a structured rewiring approach using the minimal euclidean distance may not be optimal to structure a sparse neural network, but nevertheless surprises with almost imperceptible changes in test accuracy as well as a constant adversarial robustness. This could probably be further improved by using square shaped kernels in the convolutional layers, due to the reason that a hypersphere would fit most optimal in a hypercube and less so in a hyperrectangle, that way filling less vectors more densely. Additionally our results suggest that, a faster convergence towards a structure using the weight-based minimal euclidean distance was desirable and resulted in a way more structured end product at the same amount of training.

Furthermore it is to be mentioned, that we tried to refine the structure by using a very simple structured pruning approach. We could see a significant gain in optimal structure in terms of the number of vectors that were completely empty, in the highly sparse (99% sparsity) network. The performance expectedly decreased, but this could be a starting point to further research structured pruning. If our weight-based structured rewiring approach is used in order to structure a network with (i.e. 98% sparsity) during training which is then pruned structure aware (to i.e. 99% sparsity) or a few weights are rewired into neighbouring dense tensors and then a few non-rewiring training steps are put on top, one could arrive at a solution that slowly converges to a useful structure in terms of computation speed.

References

Athalye, A., Carlini, N., & Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. https://doi.org/10.48550/ARXIV.1802.00420

Bellec, G., Kappel, D., Maass, W., & Legenstein, R. (2017). Deep rewiring: Training very sparse deep networks. *CoRR*, *abs/1711.05136*. http://arxiv.org/abs/1711.05136

Capra, M., Bussolino, B., Marchisio, A., Masera, G., Martina, M., & Shafique, M. (2020). Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, *8*, 225134–225180. https://doi.org/10.1109/ACCESS.2020.3039858

Chan, S. (2018). *Chapter 3 adversarial attack*. https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter3.pdf

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras & tensorflow: Concepts, tools, and techniques to build intelligent systems* (2nd). O'Reilly Media, Inc.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., & Dally, W. J. (2016). Eie: Efficient inference engine on compressed deep neural network. https://doi.org/10.48550/ARXIV.1602.01528

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, *abs/1502.01852*. http://arxiv.org/abs/1502.01852

Kappel, D., Habenschuss, S., Legenstein, R., & Maass, W. (2015). Synaptic sampling: A Bayesian approach to neural network plasticity and rewiring. *Advances in Neural Information Processing Systems*, *28*. https://proceedings.neurips.cc/paper/2015/file/b1a59b315fc9a3002ce38bbe070ec3f5-Paper.pdf

Kappel, D., Legenstein, R., Habenschuss, S., Hsieh, M., & Maass, W. (2017). A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. https://doi.org/10.48550/ARXIV.1704.04238

Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. https://doi.org/10.48550/ARXIV.1711.05101

Narang, S., Elsen, E., Diamos, G., & Sengupta, S. (2017). Exploring sparsity in recurrent neural networks. https://doi.org/10.48550/ARXIV.1704.05119

Özdenizci, O., & Legenstein, R. (2021). Training adversarially robust sparse networks via Bayesian connectivity sampling. *International Conference on Machine Learning*, 8314–8324.

Rauber, J., Zimmermann, R., Bethge, M., & Brendel, W. (2020). Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. *Journal of Open Source Software*, *5*, 2607. https://doi.org/10.21105/joss.02607

Sehwag, V., Wang, S., Mittal, P., & Jana, S. (2019). Towards compact and robust deep neural networks. https://doi.org/10.48550/ARXIV.1906.06110

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. https://doi.org/10.48550/ARXIV.1409.1556

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. https://doi.org/10.48550/ARXIV.1312.6199

Wang, Z., Wohlwend, J., & Lei, T. (2020). Structured pruning of large language models. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://doi.org/10.18653/v1/2020.emnlp-main.496

Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., & Jordan, M. I. (2019). Theoretically principled trade-off between robustness and accuracy. *CoRR*, *abs/1901.08573*. http://arxiv.org/abs/1901.08573