LU TP 19-25
August 2019

Exploring the LASSO as a Pruning Method

Leo Ekbom

Department of Astronomy and Theoretical Physics, Lund University

Bachelor thesis supervised by Mattias Ohlsson

LUND
UNIVERSITY

Abstract

In this study, the efficiency of various pruning algorithms were investigated, with an emphasis on regularization methods. Pruning is a method which aims to remove excess objects from a neural network. In particular, this included the LASSO (Least Asbolute Shrinkage and Selection Operator) and the extensions derived from it, which were compared with other methods, including optimal brain damage and the elastic net. Initially, this was implemented for MLPs, but the same methods were extended to CNNs with some alterations for increased computational efficiency. Pruning was then implemented on the level of weights, neurons as well as filters. It was concluded that the LASSO tends to yield a superior sparsity on the level of weights, but the group LASSO's ability to select variables simultaneously is a worthwhile addition. Also, optimal results can be obtained by combining both while regularizing the cost function.

Populärvetenskaplig sammanfattning

I många däggdjurs hjärnor pågår en process som kallas ”synaptic pruning”, som går ut på att göra sig av med hjärnans mindre använda kopplingar. Likt detta kallar man processer för ta bort överflödiga objekt från neuronnätverk för ”pruning”. Ett av de ursprungliga sätten att åstadkomma detta var det så kallade ”optimal brain damage”, som definierade ett sätt att sortera parametrar så att prestationen påverkades så lite som möjligt. Man vill alltså ha svaret på hur mycket man kan skada nätverket, men fortfarande låta det bearbeta information på ett lika effektivt sätt. Det här genomförs eftersom att moderna nätverk, till exempel de som används för bildanalys eller röstigenkänning, ofta blir stora och oerhört komplexa. Genom pruning kan storleken minskas och hastigheten ökas, vilket är särskilt viktigt för mindre kraftfulla mobila apparater. Många andra metoder för pruning kretsar kring regularisering, vilket innebär att man på något sätt justerar nätverkets förlust för att få ett fördelaktigt beteende. En av de viktigaste kallas för LASSO (Least Absolute Shrinkage and Selection Operator). I den här studien testas LASSO som ett verktyg för pruning, och jämförs med många andra metoder, för olika typer av nätverk.

Contents

1 Introduction

Within machine learning, there has been an increasing interest in solving various processing tasks using ANNs (Artifical Neural Networks). While the original perceptrons only included up to a few hidden layers, modern deep learning networks may include several thousand, and there is hardly an upper bound for how complex such a network can become [1]. Accordingly, the task of finding optimal architectures and hyperparameters becomes all the more daunting. Furthermore, as applications increasingly need to run on portable devices with less powerful CPUs, more efficient network structures are needed, without compromising the accuracy of the task. With that, the importance of pruning methods has resurfaced. While a certain architecture can be suggested during training in a form of trial and error procedure, pruning offers the opportunity to remove excess objects from a network. Here, the objective becomes to sparsify the network of neurons, weights, or in some rare cases, biases. For instance, it is often the case that the majority of connections can be removed without greater consequence in terms of error [1][2][9].

While pruning for weights, one of the original benchmarks was studying the behavior of the second derivative of the loss function while searching for a pruning criterion. This idea was introduced with OBD (Optimal Brain Damage) [3]. These methods may be compared with the ones accomplished through regularization, as the task of increasing sparsity, speed and performance of an ANN is somewhat intertwined with the reduction of overtraining. A large amount of parameters in a network with complex connections will naturally result in overfitting of the data, and develop false dependencies. Another key objective then becomes to find a network with enough connections as to not significantly impact the test error, but which is still sparse enough to generalize well. In regularization, we adjust the loss function by adding a regularizer, which aims to give supplementary information as to improve the solution. The methods used here will be based on weights decay, which uses the $l2$ norm to regularize, and the LASSO (Least Absolute Shrinkage and Selection Operator), which uses the $l1$ norm [4].

All of these methods introduced may yield a weight-level type sparsity, with fewer connections. In some instances, there may also be interest to accom-

plish node-level sparsity, where entire neurons are removed. This gives rise to the Group LASSO, a modified LASSO operator which can take into account several objects simultaneously. Here, we may for instance define all outgoing weights from a node as a group, and with the right solution set, we may then remove the entire node [5].

Lastly, as an increasing majority of today's online information tends to be visual, there has also arisen a need to handle such information efficiently, which tends to be achieved through CNNs (Convolutional Neural Networks). Pruning CNNs is generally a more difficult and computationally expensive task. However, it can still often be possible to use analogous methods to the ones used for MLPs. To be investigated here is firstly a variation of OBD applied to kernel weights, as well as a group LASSO which aims to prune for entire filters. This concludes the main aim of the project, which is to investigate and compare brain damage methods with different types of regularization for pruning both MLPs and CNNs for different levels of sparsity.

2 Background

2.1 Optimal Brain Damage

For the rest of this section, we fix a training data set $\{x_i, y_i\}$, and let $L(y_i, f(x_i))$ be a generic loss function. Furthermore, we let the induced local field with weights w_{ij} be denoted $a_i = \sum_j w_{ij} x_j$ where, for this task, we are not interested in pruning bias weights. While pruning weights, we naturally aim to find those which affect the error the least during training. One of the most intuitive ways the accomplish this would be through magnitude based pruning, where one penalizes weights of larger size though each iteration of training, and introduces a cutoff where one decides to remove a weight completely. However, if we set this cutoff high enough to reach the sought after sparsity, it will most likely impact the test error more than desired.

In the introduction it was stated that the order in which weights could be

removed from the network can be decided through the second derivative behavior of the loss function. The rationale is that, once the network is trained to a local minimum, the weight influence on the loss function is dominated by the second order terms i a Taylor expansion. To avoid of the full Hessian matrix, it is possible to focus on the diagonal terms. This is a standard approximation while implementing OBD, originally from [3]. Through standard backpropagation, the diagonal second order derivatives can be calculated as

$$\frac{\partial^2 L}{\partial \omega_{ij}^2} = \frac{\partial^2 L}{\partial a_i^2} x_j^2. \tag{1}$$

The second order derivative aims to give information on the importance of a weight, and is then used as the pruning criterion for deciding which to discard from the network. In [3], the importance of the weight as determined by the second derivative is referred to as the saliency of the weight.

2.2 The Weight Decay and the LASSO

As stated, the relation between overtraining and pruning gives the impression that regularization techniques may successfully yield effective pruning criteria. Firstly, we have $l2$ regularization, which is also known as weight decay. This has a similar effect to the simplest instance of magnitude based pruning, where large weights are successively penalized in their impact. Weight decay is implemented by adjusting the loss function by restricting the weights in accordance with the $l2$ norm. We define the $l2$ norm regularization acting on the weights as:

$$R_{l2} = \frac{1}{2} \sum_i w_i^2 = ||w||_2 \tag{2}$$

With N being the size of the training data, we can then formulate our regularization, or pruning, problem as:

$$w^* = \text{argmin} \left(\frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda R_{l2} \right) \tag{3}$$

where λ is a regularization parameter. This is unique to each network, and in the case of layer-wise pruning, also unique to the layer.

Notably, the solution to the regularization problem provided by weight decay yields quite unsatisfactory results in terms of sparsity. While weights diminish by a value proportional to their magnitudes, this will never be identically zero. The value by which the weight decreases will become smaller as the weight becomes smaller, and the weight could thus only ever asymptotically approach zero. In other words, the step size becomes increasingly smaller, and always smaller than the magnitude of the weight itself. Again, we could introduce some type of cutoff for removing weights, but the same issues remain. Instead, we intuitively want a function whose sharpness allows for direct contact with the loss function and decision boundary, and thus provide null solutions. In the LASSO framework, we have a regularization operator of the form:

$$R_{l1} = \sum_i |w_i| = ||w||_1 \tag{4}$$

The absolute value allows for weights to be compressed to zero and thus removed from the network. While generally simple to implement, it is notable that the loss function under the LASSO will be non-differentiable at the origin. Instead, one normally sets the gradient to

$$R'_{l1} = \text{sign}(w_i) \tag{5}$$

Effectively, a constant value is subtracted from the weights, now independent of the weight, rather than one which is proportional to the weights itself, and we can theoretically obtain a zero value for some weights, as opposed to weight decay.

Further, in the case where we would like to have small weights, but still maintain sparsity, the norms can be directly added to create what is known as the elastic net regularizer [6]. This would then simply be of the form:

$$R_{el} = R_{l1} + R_{l2} \tag{6}$$

2.3 The Group LASSO

So far we have introduced OBD, weight decay, and the LASSO as tentative pruning algorithms on a weight-level. Although they all have associated advantages, none manage to accomplish pruning at a higher level, which may result in preferable density structure. The formulation of the group LASSO allows us to define what may constitute a suitable group. If we wish to prune for entire nodes, we may define all outgoing weights of a given node as a group. In the case where all the weights are found to be zero, the node is effectively removed from the network. If we let W_l denote the weight vector of a given group, let p_l be its size, and include $\sqrt{p_l}$ since groups may be of varying sizes, we may modify the LASSO norm such that

$$R_g = \lambda \sum_l \sqrt{p_l} ||W_l||_1 \tag{7}$$

where the 1-norm would now indicate taking the square root of the squared sum of all weights involved, rather than doing it separately. Performing pruning on node-level is somewhat similar to feature selection. Although not the main task, we may remove entire features by letting the group LASSO work on the input nodes, instead of the hidden layer nodes.

Now, since a group of weights is either completely zero or non-zero, there is an issue of some individual weights in non-zero groups possibly being insignificant enough such that they also could be removed. In statistics, this would be referred to as "within group-wise sparsity", which would here constitute the weights. This issue can simply be ameliorated by, similarly to what is accomplished by the elastic net, combining it with the $l1$ norm as a measure to achieve both effects to a certain extent. This is normally labeled the sparse group LASSO (SGL) [7], and would be of the form:

$$R_{SGL} = R_g + R_{l1} \tag{8}$$

2.4 The Convolutional Network

Due to the importance of visual processing tasks, many of the modern implementations of pruning are done on Convolutional Neural Networks (CNNs). Even though MLPs may be used efficiently for plenty of classication problems, when considering image processing, far too many nodes would be required in order to implement training. This is so even when pruning is taken into account, as one needs to consider every individual pixel. This complexity issue may be solved by introducing convolutional layers instead of having fully connected layers, which, together with the activation functions in between, can make up image structure. These can through convolutional operations analyze inputs of higher dimensionality with much greater detail. While performing these operations, we also define the filter matrix, a structure within each convolutional layer, which includes the weights and represents some feature. Each filter generally has the task of recognizing one feature within the input image, which becomes more complex deeper in the network. This is clearly a larger structure than the node, and a number of nodes is generally incorporated into a single filter. The generation of a convolutional layer is shown in Figure 1.

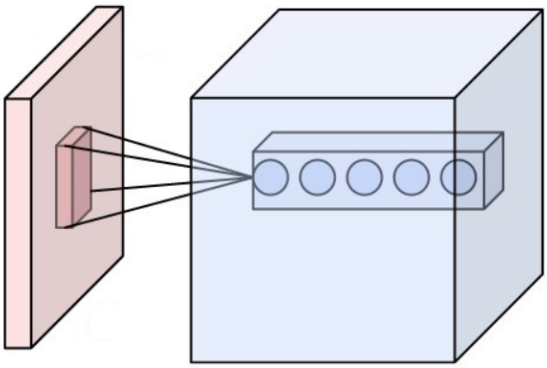


Figure 1: Illustrates the generation of a convolutional layer, including the neurons and the receptive field [8]

.

As previously mentioned, pruning of convolutional layers will be investigated using similar methods to the those of the MLP such that a definitive comparison can be made. Firstly, as a preliminary, the equivalent to magnitude

based pruning here would be to calculate the absolute sum of some set kernel weights $\sum_l |W_l|$, and sort the filters accordingly. For a pruning effect, this would then be similar to using the $l2$ norm to regularize within the kernels, and establish a cut-off, with the same possible disadvantages as earlier.

We can extend the tools previously used for pruning, namely the group LASSO and OBD, for usage on convolutional layers, with some modifications to be clarified in the method section. Analogously to before, we can also decide to prune on different levels of the network for different effects. Now, we can say that we want sparsity on a filter-level, or similarly on a kernel-level. Alternatively, we can obtain a form of sparsity on weights within the filter, which is similar to the weight-level sparsity of the MLPs. As will be seen, we can use a variation of the group LASSO and its combination with the $l1$ norm to prune filters, and OBD for ways to prune weights within.

2.5 Some Implementation Background

While optimizing the network, a form of stochastic gradient descent known as Adam was chosen. Apart from being computationally efficient, this has benefits when working with sparsity, by continuously adjusting the learning rate. It takes four parameters: α, which represents the learning rate, β_1, β_2 which represent decay rates, and ϵ, which is just used to avoid division by zero [10].

Another aspect to consider is the computational efficiency when generating convolutional layers, and it can be advantageous to attempt to alter the expression. There are methods that can reduce convolutions to ordinary matrix multiplication, making it more similar to what is implemented in the MLP. As is described in [13], the general convolution process can be reduced to a matrix multiplication, thus significantly reducing memory requirements.

Regarding the architecture of CNNs, the LeNet was used. This was thought to be a reasonable choice as the LeNet 5 is one of the simpler varieties of CNNs with only one fully connected layer, and three convolutional layers. It is also computationally efficient in that not all features used in the fully connected layers are used in the convolutional layers [11]. This can then make

the task of pruning these layers less daunting, and more focused. Naturally, architectures such as the AlexNet could also have been used, but were not deemed worthwhile for data sets used.

3 Method

3.1 Choice of architectures

The overall architectures of the MLPs used in experiments will generally be in agreement with the following pattern. Concerning activation functions, the ReLu(Rectified Linear Unit), was typically used for all hidden layers. This was due to its superior efficiency in gradient descent which is of great use in OBD, as well as the fact that it has a sparse activation [9]. This is particularly useful in cases where the network is deep enough to where the standard sigmoid functions start to become obsolete. The ReLu is simply of the form $g(x) = \max(0, x)$. In the case of classification, a softmax function of the form $g(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$ was used on the output layer, since it provides a form of probability distribution. In addition, for classification, a cross-entropy loss function of the general form $L(y, f(x)) = -\sum_i y_i \log(f(x_i))$ is also used, as it pairs well with this framework. In contrast, for problems of a regression type, a mean-squared error of the form $L = \frac{1}{N}\sum_{i-1}^{N}(y_i - f(x_i))^2$ was used. In the case of CNNs, a LeNet 5 architecture was used, with the same activation and loss functions as for classification with MLPs.

When pruning a network, weights are first sorted according to some "significance" criterion. For OBD, it is the saliency derived from the second order derivative information, for Lasso-based methods we used the conventional magnitude based measure, ω. Then, a cut-off is used to remove the least significant weights. Thus, the final pruning will depend on such a pruning cut-off, as well as any regularization parameter λ. To compare different pruning methods, we first determined a reasonable value of λ for each method, and then varied the pruning cutoff for that λ to see how different levels of pruning affected accuracy.

Like other hyperparameters, there exists no unassailable method to determine the regularization parameter λ, but there is typically a preferable order of magnitude. To find it, we first trained an initial network with $\lambda = 0$ to determine a base-line accuracy. With a fixed, bench-mark pruning cutoff value, which was 10^{-4} for saliency and for magnitude based pruning in different LASSO versions, λ was incrementally increased. When accuracy was seen to drop significantly, becoming about 3 percentage points lower than the base-line, the increase of λ was stopped, and the previously tested λ was selected as the "optimal" value.

For a network trained with the optimal λ, the bench-mark cutoff was then ignored, and the weights and/or nodes were removed in order of their significance quantity, thus systematically increasing "sparsity", meaning the fraction of removed relevant quantities. The accuracy as function of sparsity could then be determined, and different pruning methods was compared, as shown in several plots in the result section.

During this process, samples were split into training, validation and test sets according to the percentages 70-20-10, where random selections of date were chosen for validation and testing in each separate run. For each investigated λ, cross validation was used to determine hyperparameters such as learning rate, and with the resulting choices a network was trained on the combined training+validation data, and applied to the test set to get an unbiased measure of accuracy.

The experiments were all implemented in Python using the Tensorflow, Keras, Numpy and Sklearn libraries. All networks were optimized using the Adam algorithm. The Adam parameters were set as $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

3.2 Extension to CNNs

3.2.1 OBD

Since part of the comparison of MLPs was between OBD and the LASSO, it can be considered worthwhile to extend this comparison to convolutional layers, and investigate if conclusions can be similarly drawn. However, even with the diagonal approximation in effect, this is very computationally expensive for most data sets. Here, a more efficient method was to alter the pruning criterion in direct implementation of OBD. Instead of looking at second order terms in the Taylor expansion, it is possible to instead look at first order terms. While the loss function is assumed to be in a local minimum such that this will be zero, with a large enough sample size it can we worthwhile to instead study the variances of the gradient, which will be a non-zero quantity [12].

3.2.2 The LASSO

While previous methods can be efficient, there is still a variety ways to employ regularization for sparsifying CNNs. As is the case for feedforward networks, some of these still revolve around the LASSO and, here in particular, the group LASSO, with some variations to be taken into account. Looking at the W_l factor from the original group LASSO estimator, we still keep it as a general expression for all weights, and modify our group LASSO term to a more general general expression. This is done similarly to what is expressed in [13]. If $1 \leq l \leq L$, l being a specific layer, and letting N be the number of groups and $g = 1, ..., N$ a specific set of weights within one filter, we can express the regularization term as:

$$R_g = \lambda \sum_{1}^{L} \sum_{g=1}^{N} ||W_{lg}|| \tag{9}$$

3.3 Data sets

One of the major goals of this work will be to draw conclusions on the functionality of pruning for different varieties of data sets with a different number of parameters. When necessary, all data sets were preprocessed through normalization, and given a zero mean.

As a small and common data set for implementation of early experiments, the Iris data set was used. Generally, this results in a rather simple classification problem for a non-complex MLP, but can still be of interest to prune. It has three classes of iris plants, all with 50 samples.

Next, pruning was performed on a generating function with six features, which had the form: $y = 2x_0 + x_1x_2^2 + \exp(x_3) + 5x_0x_1 + 3\sin(2\pi x_1)$ This consisted of 10,000 samples.

Finally, for all CNN experiments, the MNIST data set was used. This consists of around 70,000 handwritten digits. Since this is a problem which may also be solved using MLPs, it provided a good comparison.

4 Results

Whether the implementation is on weights, neurons or filters, what constitutes a successful pruning method will be the relationship between sparsity and test error. Normally, the test error will not be smaller as a consequence of pruning, but rather strive to be as similar as possible to that achieved in the original architecture, such that it can act as a sparser replacement. As defined in section 3.1, a value of optimal sparsity is sought after. At some subsequent point, the network will carry so little information such that it will cease to be practical. This will be explored for each method on each of the data sets.

4.1 OBD

We begin by presenting results for optimal brain damage, or pruning with the saliency criterion. This can in a natural way be compared with another criterion, which was labeled the simplest method: the magnitude based pruning (MBD), which as stated merely sorts and removes weights in terms of their magnitudes.

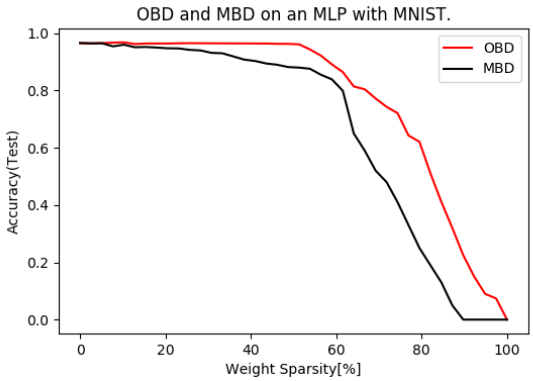


Figure 2: The two simplest pruning algorithms compared on the MNIST data set. In pruning, the region where the test error starts to rapidly increase was searched for, at which point once could no longer safely remove weights. For MBD, this decay clearly begins quite early, making it an inefficient pruning tool. For OBD, even for a large dataset like MNIST, more than half the weights could safely be removed.

As would have been expected, the saliency criterion appears much more effective in terms of sparsity than the magnitude variant, as can be seen from Figure 2. Still, it is not entirely satisfactory. The saliency criterion relies on the diagonal as well as the quadratic approximations, and appears to generally function well for slightly above half of the network weights.

The results for the rest of the datasets are shown in Table 1, which gives the sparsity of all weights as well as optimal test accuracy. When looking at the different datasets, the error at a certain sparsity level also appears to be proportional to the number of layers, as well as the number of weights per layer, in the network. For instance, in the iris case, four inputs and three

outputs were used, as well as only one hidden layer, for a total 15 neurons. Here, around four fifths of the weights can quite safely be removed. In the MNIST case, the MLP is more complicated, and the result will be later improved upon using CNNs, as is natural for such a large classification data set.

Table 1: Shows the result for optimal weight sparsity for each of the datasets, as well as the accuracy on the test set. The higher sparsity values show that OBD is the higher performer, proving quite effective for the small Iris dataset.

Dataset	Quantity	OBD	MBD
Iris	Weight Spars.	0.76	0.41
	Accuracy	0.98	0.98
Gen. Set	Weight Spars.	0.68	0.38
	Accuracy	0.97	0.97
MNIST	Weight Spars.	0.58	0.29
	Accuracy	0.97	0.98

4.2 The LASSO

Now, as a continuation of previous methods, we investigate what is theoretically a powerful tool for obtaining sparse networks: the LASSO. As explained, the loss function is adjusted with the $l1$ norm, and an overly small λ is chosen, typically of the order of 10^{-4}, and continuously increased until some degree of sparsity is observed, as is demonstrated in Figure 3 for the generated dataset. There is no exact method for obtaining a satisfactory value, but one needs to look at the corresponding test error, and choose a λ where sparsity no longer increases, but test error does. This value appeared to be somewhat related to the size of the dataset. For instance, for Iris, it was chosen to be 10^{-2}, for the generated set $5 \cdot 10^{-2}$, and for MNIST 10^{-4}.

Similarly to the magnitude based preliminary of OBD, it can be useful to observe how the weight decay contrasts the LASSO. In agreement with what was put forth in the theory section, this does not yield sparsity in the manner

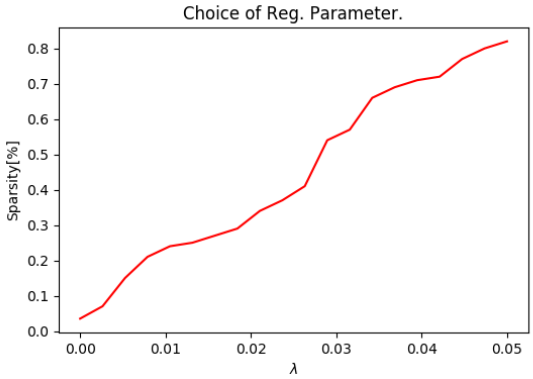


Figure 3: Shows the choice behind one regularization parameter for the generated set. As the parameter is increased, the sparsity clearly follows. This is the case up to a certain point, where the trade-off of greater test error no longer becomes worthwhile.

of the LASSO. While it can be a useful tool when it comes to things like overfitting, we can see from Figure 4 that it here fails, just like magnitude based pruning, when the main success criterion is sparsity. The rest of the LASSO results are summarized in Table 2. For each dataset and method, it shows the found optimal regularization parameter, sparsity percentage, and test accuracy. As we can see by comparing Table 2 and Table 1, on all of the datasets, the LASSO appears to consistently outperform OBD.

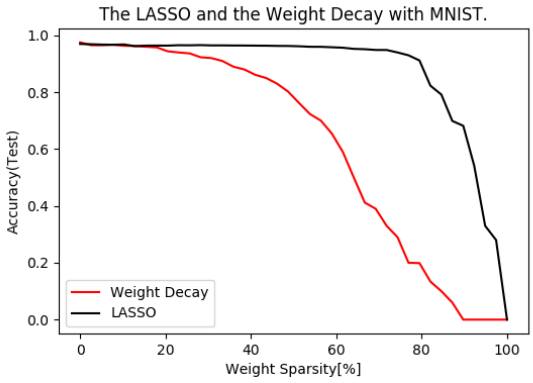


Figure 4: Demonstrates weight pruning using the LASSO and the weight decay on MNIST.

Table 2: Shows the result for optimal weight sparsity for each of the datasets.

Dataset	Quantity	LASSO	Weight Decay
Iris	λ	0.01	0.01
	Weight Spars.	0.84	0.39
	Accuracy	0.98	0.98
Gen. Set	λ	0.05	0.05
	Weight Spars.	0.81	0.32
	Accuracy	0.96	0.97
MNIST	λ	10^{-4}	10^{-4}
	Weight Spars.	0.83	0.24
	Accuracy	0.97	0.98

4.3 The Group LASSO

One of the main downsides of the conventional LASSO is that it has a form of early selection saturation point. This means that, correlated variables, which one would normally categorize in a group, are typically not selected together while performing the regularization [4][6]. The extension of the LASSO to the group LASSO described earlier aims to ameliorate this issue, and appears to accomplish it rather well. Mainly, the group LASSO will be implemented on hidden layers, and it is here the sparsity quantity will be defined; not on the inputs. In Figure 5, we see the comparison in the removal of these nodes accomplished by the group LASSO as opposed to the regular LASSO, and how it may impact the test error. Complete results are shown in Table 3.

From these results, mainly the weight sparsity we can observe the group LASSO still generates a more densely connected network than that of the LASSO. However, as will be seen in the next sections, the unique effect of node removal of the group LASSO still has a role to play in inducing sparsity.

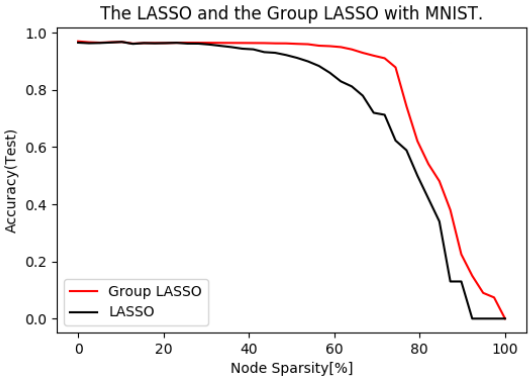


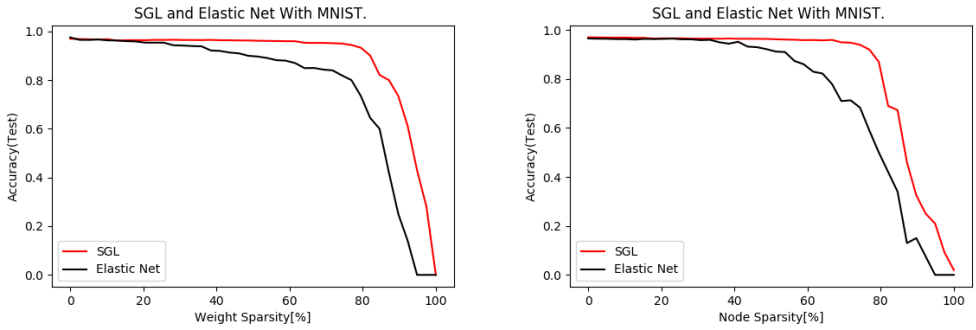
Figure 5: Demonstrates node pruning using the LASSO and the group LASSO on MNIST. Similarly to before, we search for an optimal sparsity level before test error significantly increases.

Table 3: Summarizes node pruning for all the datasets.

Dataset	Quantity	LASSO	Group LASSO
Iris	λ	0.01	0.01
	Weight Spars.	0.84	0.57
	Node Spars.	0.53	0.84
	Accuracy	0.98	0.98
Gen. Set	λ	0.05	0.05
	Weight Spars.	0.81	0.52
	Node Spars.	0.50	0.83
	Accuracy	0.96	0.96
MNIST	λ	10^{-4}	10^{-4}
	Weight Spars.	0.83	0.57
	Node Spars.	0.47	0.83
	Accuracy	0.97	0.97

4.4 Combinations of Norms

It is the purpose of the elastic net and the SGL to achieve a combined regularization effect by simply adding the respective norms involved. This was investigated, and found to have a certain degree of success.



(a) Weight pruning (b) Node pruning

Figure 6: Pruning using combinations of norms on MNIST.

As can be observed in Figure 6a) for weights and 6b) for nodes, the elastic net demonstrates a certain degree of node and weight sparsity. Since the $l2$ regularization has a theoretically greater grouping effect as compared to $l1$, it was considered worthwhile to be investigated. However, in comparison with other methods, it did not prove to be particularly useful as a pruning tool, as can be seen for all datasets in Table 4. Also, here one regularization parameter is required for each term, making the fine-tuning a bit more difficult.

The next step will be to examine the combination of the ordinary LASSO and the group LASSO, and see if this combination of sparsity and grouping is of greater success than the elastic net. While of similar motivations, this appears to yield the most effective sparsity of all the regularization methods, as again demonstrated in Figure 6a) and b) as well as Table 4. It can also be implemented using only one regularization parameter, despite the two terms. In terms of weights, it is on par with the regular LASSO. However, in terms of nodes, it even slightly outperforms the group LASSO.

4.5 Extension to CNNs

Finally, we will examine whether similar conclusions to those drawn above can be applied to the convolutional layers, with the addition of pruning for

Table 4: Summarizes norm combination for all the datasets.

Dataset	Quantity	El. Net	SGL
Iris	λ	0.01/0.01	0.01
	Spars. (Weights)	0.35	0.84
	Spars. (Nodes)	0.33	0.86
	Accuracy	0.98	0.98
Gen. Set	λ	0.04/0.07	0.04
	Spars. (Weights)	0.33	0.82
	Spars. (Nodes)	0.30	0.85
	Accuracy	0.96	0.96
MNIST	λ	$10^{-4}/10^{-4}$	10^{-4}
	Spars. (Weights)	0.33	0.82
	Spars. (Nodes)	0.29	0.85
	Accuracy	0.97	0.97

filters.

Concerning OBD, we see from Figure 7 a somewhat greater success from altering the pruning criterion, as compared to the second derivative information, in addition to now being more computationally efficient with the absence of Hessians. For weight sparsity, it here even appears to be slightly more efficient than the LASSO, which makes it possible to question the usefulness of second derivative methods altogether.

Next, we have the LASSO extensions applied to CNNs, with the goal of reducing the number of filters. The effectiveness of the group LASSO and the SGL are compared in Figure 8. Using both methods, and the two first convolutional layers' filters, significantly more than half of the filter could be pruned. For larger networks, it can then even be feasible to remove entire convolutional layers, when there are no more features. Similar methods can also be used to prune for network channels. Again, we can observe how the SGL provides the greatest pruning efficiency.

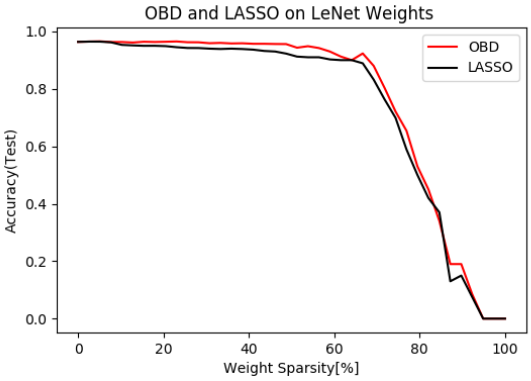


Figure 7: The new OBD variation and the LASSO applied on weights of the LeNet-5 with MNIST.

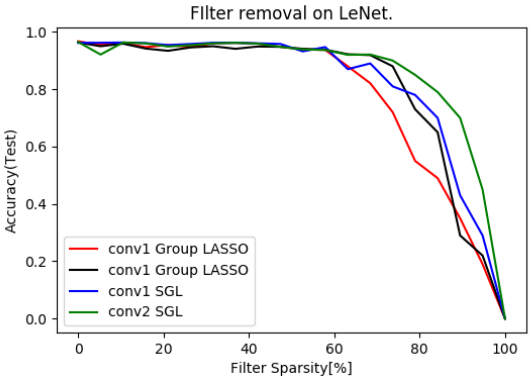


Figure 8: The Group LASSO and the SGL pruning filters of the two convolutional layers of LeNet-5 with MNIST.

5 Discussion

5.1 Comparison of Pruning Results

As was expected by the general theory, and is quite evident from many of the results, there is a certain order of effectiveness of the methods in terms of pruning. While OBD is a powerful preliminary, often allowing for removal

of more than half of a network's connections, it is quite weak in comparison to more modern methods. While it does yield much better weight sparsity than MBD as well as the weight decay, it is consistently of worse performance than the LASSO, as well as its extensions. A possibility could be that some of the approximations, in particular the diagonal approximation which is not theoretically motivated, are not particularly effective past a certain point. As mentioned, the group LASSO, while it can be used for other things such as feature selection, generally yields somewhat less weight sparsity than the conventional $l1$ LASSO. This can be the case due to the possibility of there being a multitude of individual superfluous weights not included in a any particular group. However, the two combined was observed to be highest performing type of pruning, with both nodes and individual weights being able to be removed, for the most compressed, and seemingly effective, network. The elastic net, while yielding some form of combination of smaller weights and sparsity, could not be considered a particularly effective pruning tool. The same thing can be concluded with node-level sparsity.

When it comes to the the convolutional networks, the new OBD criterion made for consistently more successful accuracy while pruning. This leads to the conclusion that this is a worthwhile substitution for a large enough network, even for MLPs, since it even slightly outperformed the LASSO in terms of weight sparsity in the LeNet CNN structure. Additionally, the LASSO variations here appear about as efficient as for MLPs with the SGL being able to prune for filters with rather good accuracy.

A last thing which could be of interest to investigate would be how, similarly to how the OBD and regularization methods were extended to CNNs, to see how they could be extended to RNNs, and if the SGL maintains similar efficiency.

5.2 Further Methods For Sparsity

As was briefly touched upon in earlier sections, there are alternatives to pruning for a more sparse, compact or simply faster and easier to understand network. One of these would be parameter prediction, where, instead of removing parameters and connections considered superfluous, one predicts

which ones may be important, and then construct the network accordingly. For instance, the model can learn a few weights corresponding to a certain feature and then use it as a form of dictionary for the rest of the structure [15].

Far from all variations of pruning were included in this project, with for instance pruning using dropout being something which could have been investigated. This is one of the more common forms of regularization, which makes it an obvious alternative to the LASSO variations. Here, nodes can be assigned a certain probability of not being included in the learning process, whose effect can subsequently lead to another form of pruning criterion [16].

Another compression technique is one which is generally referred to as quantization of neural networks. Here, one can for instance form clusters of parameters and use it to reconstruct the network in a way that saves computational resources [17]. Alternatively, there is also the method of distillation, aiming to directly transfer information from a more complex network, to one that is less so. Essentially, rather than using the original dataset, the smaller network is trained to replicate the performance of the larger as well as possible. A variation of this would be in new methods such as MorphNet, taking the larger network directly as input [18][19].

5.3 Conclusion

Based off our results, The LASSO's sparsity property clearly yields optimal weight sparsity results, rivaled only by the very efficient altered OBD condition used for CNNs. The group LASSO proved just as effective for pruning nodes, and the combination of both can be concluded as the ideal pruning tool out of those investigated.

References

[1] J.Sietsma and R.J.Dow, "Neural net pruning-why and how", in *Proceedings of the IEEE International Conference on Neural Networks*, ISan

Diego, CA, USA, Volume 1, pp. 325–333, 1998.

[2] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression", arXiv:1710.01878 [stat.ML].

[3] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage", *Advances in Neural Information Processing Systems*, vol. 2, pp. 598–605, 1989.

[4] R. Tibshirani, "Regression shrinkage and selection via the lasso", *Journal of the Royal Statistical Society*, pp. 267–288, 1996.

[5] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables", *Journal of the Royal Statistical Society*, vol. 68, no. 1, pp. 49–67, 2006.

[6] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society*, Series B (Statistical Methodology), vol. 67, no. 2, pp. 301–320, 2005.

[7] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso", arXiv:1001.0736 [math.ST].

[8] `https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Conv_layer.png`, "Input volume connected to a convolutional layer", 2015.[Online; accessed 14/05-2017].

[9] C. Aggarwal, "Neural Networks and Deep Learning", *Springer*, 2018.

[10] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", arXiv:1412.6980 [cs.LG].

[11] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition", *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[12] P.Molchanov, S.Tyree, T.Karras, T.Aila and J.Kautz, "Pruning Convolutional Neural Networks For Resource Efficient Inference", arXiv:1611.06440 [cs.LG].

[13] W.Wen, C.Wu, Y.Wang, Y.Chen, H.Li, "Learning Structured Sparsity in Deep Neural Networks", arXiv:1608.03665 [cs.NE].

[14] B.Liu, M.Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse Convolutional Neural Networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 806–814, 2015.

[15] M. Denil, B. Shakibi, L. Dinh, and N. de Freitas, "Predicting parameters in deep learning", in *Advances in Neural Information Processing Systems*, pp. 2148–2156, 2013.

[16] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization", arXiv:1412.6115 [cs.CV].

[18] G.Hinton, O.Vinyals, and J.Dean,"Distilling the knowledge in a neural network", arXiv:1503.02531 [stat.ML].

[19] A.Gordon, E.Eban, O.Nachum, B.Chen, H.Wu, T.Yang and E.Choi, "MorphNet: Fast and Simple Resource-Constrained Structure Learning of Deep Networks", arXiv:1711.06798 [cs.LG].