Learning Network Size while Training with ShrinkNets

Anonymous Authors[1]

Abstract

Let's write the abstract at the end

1. Tentative outline

- Introduction
  - Finding an appropriately sized network is challenging
  - ML practitioners spend a large amount of time tuning the size of layers in a network in order to get the best possible accuracy.
  - Many techniques have been proposed for hyperparam opt but these are computationally expensive and take long to train
  - We propose ShrinkNets, an approach to tune the size of the network as it is being trained without incurring the significant costs of hyperparameter optimization.
  - Thus, our contribution are: ...
- Our Approach
  - Assign an on/off switch to each neuron. But this is np-hard, so we consider a relaxation
  - Formulation
  - Relationship to sparsity
  - Strategies to kill neurons (relation to theory above?)
  - Implementation details
- Related Work
  - Hyperparameter optimization: random, bayesian opt, bandit methods
  - distillation techniques
  - post-training compression techniques
  - group sparsity, non-parametric neural networks
  - training dynamics paper: first overfitting and then randomization?
- Experiments
  - Accuracy obtained by Shrinknets
  - Time taken to reach that accuracy compared with other hyperopt methods
  - Characterize method wrt params
  - Other experiments
- Discussion
  - Where does the method shine?
  - Side-effects of method: smaller networks, time to train
  - Potential extensions/limitations
- Conclusion

2. Introduction

One of the key defactors that affects neural net perforamnce is is the *shape* of the network, i.e., the number of layers, the number of neurons per layer, and the connections between layers. An *under-sized* network, with too few neurons or layers, is likely to have low accuracy because of insufficient capacity while an *over-sized* network is slow to train due to additional parameters and is computationally inefficient at both training and inference time. Consequently, many *hyperparameter optimization* techniques have been proposed to determine the optimal size of a neural network; these include random search (**?**), what-is-this-paper (**?**), meta-gradient descent (**?**), Gaussian processes (**?**), and many others. **Sam:** Are we really sure that no other techniques optimize the network size during training like we do? Don't we also require iterating over lambda? Is this really the key contribution of our work, that it reduces model search time? Isn't the key point that we find smaller, better models? These existing techniques all require a compute-intensive search of model space, often training of tens or hundreds of models. As a result, tuning these techniques require times longer (or many times more computational power) than the time take for a single training run.

In this paper we present a novel method to automatically find an appropriate network size, which drastically reduces

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

optimization time. **Sam:** In comparison to previous search models? The key idea is to learn the network size while optimizing the primary objective function. Our strategy, called **ShrinkNets**, is blah blah **MV:** Short blurb about the technique Our approach has two main benefits. First, we no longer need to choose a network size before training. We simply set an initial size for the network and then the algorithm will determine the best network that is smaller or equal in size to the inital size. Second, this optimization is done during a *single* training run, as opposed to the large number of training runs required by existing hyperparameter optimization techniques. As a result, we can find the best model faster as well as with less computational overhead.

In summary, our contributions are as follows:

1. We propose a novel technique based on dynamically swiching on and off neurons, which allows us to optimize the network size as the network is trained.
2. We show that our technique is a relaxation**MV:** ? of group sparsity and prove **MV:** fill in.
3. **Sam:** Some claim about model size vs performance
4. We demonstrate the efficacy of our technique on both convolutional and fully-connected neural nets, showing that ShrinkNets finds networks within +/-X% of best hand-crafted accuracy in XX% of training time compared to existing hyperparameter optimization methods.
5. We also demonstrate that ShrinkNets can achieve this accuracy with only YY% of neurons.
6. **Sam:** Some claim about inference time?
7. **Sam:** Some claim about compatibility with existing compression techniques?

3. Our Approach

from the universality theorem, we know that we can find a network that ca arbitrarly fit a function. Therefore, for a given error tolreance there exists an optimal size to solve a given problem.

If we have an oversized network then there exist a pruned version that can still achieve our goal. The main idea is to consider an on/off switch for each neuron, and we want to find an assignement for this switches that achieve a certain size/accuracy tradeof. We model these on/off switches by multiplying each input (or output) of each layer by a number 0 or 1. 0 will deactivate the neuron, 1 will let the signal go through. We want to minimize the number of on switches to reduce the model size as much as we can. This can be modeled solved by jointly minimizing the objective of the network and a factor of the L0 norm of the vector containing all the on/off switches.

Finding an optimal binary assignement is an NP-Hard problem (**Should we prove this ? I think it should be fairly doable reducing it to 3-SAT considering the structure of neural networks**). We decide to relax this problem because this is what people usually do with NP-Hard problem. Our relaxation is we allow $\boldsymbol{\theta}$ to be a real number instead of a boolean We also use L1 instead of L1. This way we obtain a non-convex, but at least differentiable (almost everywhere).

This approach assumes that we start with an upper bound on the model size. This obviously translates in a computational overhead. Our insight is that some usless neurons (we have multiple definitions below) can be removed early without impacting the final solution. It has two practical implications: It mitigate the issue we describe but it also allows other neurons to adapt as soon as one of their peer is killed. Existing technique usually remove them after convergence and require an extra fine-tuning process to compensate for the removal.

The key components of the system are: The filter vectors that simulate the continuous on/off switches, a regularization that tries to kill neurons, the neuron removal strategy that detects neurons that should probably be removed, and the garbage collection that effectively remove dead neurons from the model, and the simplifaction procedure that remove the filter vector for fast inference.

We will describe these components in the upcoming sections.

3.1. Notations

In order to avoid any potential ambiguity, in this section we will describe in details the mathematical notations used in this article. Non-bold letters represent scalar values, while bold lowercase and upper case repectively denote vectors and matrices. $\boldsymbol{A}^T$ stands for the transpose of the matrix $\boldsymbol{A}$. Subscripts are used to index particular elements of vectors and matrices. $\boldsymbol{x}_i$, $\boldsymbol{A}_i$, $\left(\boldsymbol{A}^T\right)_j$ and $\boldsymbol{A}_{i,j}$ respectively correspond to the i^{th} component of $\boldsymbol{x}$, the i^{th} row of $\boldsymbol{A}$, the j^{th} column of $\boldsymbol{A}$ and the j^{th} component of the i^{th} row of $\boldsymbol{A}$. All the following definitions assume $\boldsymbol{A}$ to be an $n \times p$ matrix. For any vetor $\boldsymbol{b}$ with n components, we define diag $(\boldsymbol{b})$ a $n \times n$ matrix such that: $\forall 1 \leq i \leq n$, diag $(\boldsymbol{b})_{i,i} = \boldsymbol{b}_i$ and 0 otherwise. For any $l \in [0, +\infty]$ we define the norm: $\|\boldsymbol{A}\|_l = \left(\sum_{i=1}^n \sum_{j=1}^p |\boldsymbol{A}_{i,j}|^l\right)^{\frac{1}{p}}$. For the rest of this paper and unless stated otherwise, $\boldsymbol{y}$ will represent the output of a network, $\boldsymbol{x}$ the input, $\boldsymbol{b}$ a bias, λ regularization factors, Ω regularization methods, $\boldsymbol{\theta}$ general model parameters and a will stand for any element-wise activation function. The only constraint that we want to enforce is that $a(0) = 0$. We use $[\![u, v]\!]$ to denote inteveral of integers, $\mathbf{0}$ is the null vector (size depending on the context). $\#S$ is meant to represent the cardinality of a set S. To simplify the notation of function composition we use the following operator: $g(f(\boldsymbol{x})) = (f \circ g)(\boldsymbol{x})$ and for a long sequence of functions from f_1 to f_n we use: $f_n(...f_1(\boldsymbol{x})) = \left(\bigcirc_{k=1}^{n} f_k\right)(\boldsymbol{x})$.

3.2. The Switch Layer

Do you like the name of the layer ?

Switch layers have weights in the range $[-\infty, +\infty]$ and are usually placed after linear and convolutional layers. The *Switch Layer* takes an input of size $(B \times C \times D_1 \times \cdots \times D_n)$, where B is the batch size, C the number of features (or channels, in the case of convolutional layers), and D any additional dimension. This structure makes it compatible with fully connected layers with $n = 0$ or convolutional layers with $n = 2$. Their crucial property is a parameter $\theta \in \mathbb{R}^C$. The output is defined as follows:

$$Switch(\boldsymbol{I}; \boldsymbol{\theta}) = \text{diag}\left(\boldsymbol{\theta}\right)\boldsymbol{I} \tag{1}$$

Where θ is expanded in all dimensions to match the input size (except the second one since they are equal by definition). It is easy to see that if for any k, if $\theta_k \leq 0$, the k^{th} input feature/channel is multiplied by zero and have no influence on the output. If this happens, we say the Switch layer deactivates the neuron. These disabled neurons/channels can be removed from the network without changing its output. Before explaining how that is achieved, we explain next how the weights of the Switch Layer are initialized and adjusted during training.

3.3. Training Procedure

Once Switch layers are placed in a network and initialized (sampled from the $\mathcal{N}(0, 1)$ distribution), we could train the network directly using our standard loss function, and we could achieve performance equivalent to a normal neural network. However, our goal is to find the smallest network with reasonable performance. We achieve that by introducing sparsity in the parameters of the *Switch Layers*, thus forcing the deactivation of neurons. To obtain this sparsity, we simply redefine the loss function:

$$L'(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{W}, \boldsymbol{\theta}) = L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{W}) + \lambda \left\|\theta\right\|_1 + \lambda_2 \left\|\boldsymbol{W}\right\|_p \tag{2}$$

The additional term $\lambda|\max(0, \theta)|$ introduces sparsity (see Lasso loss (**?**)). The second component of the loss increases the gradient with respect to θ, thus pushing its value towards zero. Neurons with little impact on the original loss (gradient lower than λ), will not be able to compete against this attraction towards zero. Because the entries in θ with a value of 0 or less correspond to dead neurons, λ effectively controls the number of neurons/channels in the entire network. Without the last term our problem sounds very similar to the Group Sparsity regularization which is well known in the area of linear and logistic regressions. In the next section we will try to undercover the relationship between these two problems, explain why we need this additional regularization term and what should be the value of p.

3.4. Relation to Group Sparsity

Our goal when designing ShrinkNets was to be able to remove inputs and outputs of layers. For classic fully connected layers, which are defined as :

$$f_{\boldsymbol{A},\boldsymbol{b}}(\boldsymbol{x}) = a(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \tag{3}$$

removing an input neuron j is equivalent to have $\left(\boldsymbol{A}^T\right)_j = \boldsymbol{0}$ and removing an output neuron i is the same as having $\boldsymbol{A}_i = \boldsymbol{0}$ and $\boldsymbol{b}_i = 0$. Solving optimization problems while trying to set entire groups of parameters to zero has been already studied and the most popular method is without doubt the group sparsity regularization [ref]. For any partitionning of the set of parameter defining a model in p groups: $\boldsymbol{\theta} = \bigcup_{i=1}^{P} \boldsymbol{\theta}_i$ we define it the following way:

$$\Omega_\lambda^{gp} = \lambda \sum_{i=1}^{p} \sqrt{\#\boldsymbol{\theta_i}} \left\|\boldsymbol{\theta_i}\right\|_2 \tag{4}$$

In the context of a fully-connected layer, the groups are either: columns of $\boldsymbol{A}$ if we want to remove inputs, or rows of $\boldsymbol{A}$ and the corresponding entry in $\boldsymbol{b}$ if we want to remove outputs. For simplicity, we will focus our analysis in the simple one-layer case. In this case filtering outputs does not make a lot of sense, this is why we will only consider the former case. The group sparsity regularization then becomes:

$$\Omega_\lambda^{gp} = \lambda \sum_{j=1}^{p} \left\|\left(\boldsymbol{A^T}\right)_{\boldsymbol{j}}\right\|_2 \tag{5}$$

Because $\forall i, \#\boldsymbol{\theta}_i = n$, To make the notation simpler, we embedded $\sqrt{n}$ inside λ.

Since group sparsity and ShrinkNets try to achieve the same goal we will try to understand their similarities and differences. First let's recall the two problems. The original ShrinkNet problem is:

$$\min_{\boldsymbol{A},\boldsymbol{\beta}} \left\|\boldsymbol{y} - \boldsymbol{A}\text{diag}\left(\boldsymbol{\beta}\right)\boldsymbol{x}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_1 \tag{6}$$

And the Group Sparsity problem is:

$$\min_{\boldsymbol{A}} \left\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\right\|_2^2 + \Omega_\lambda^{gp} \tag{7}$$

We can prove the under the condition: $\forall j \in [\![1, p]\!], \left\|\left(\boldsymbol{A}^T\right)_j\right\|_2 = 1$ the two problems are equivalent (proposition A.1). However if we relax this constraint then shrinknet becomes non-convex and has no global minimum (propositions A.2 and A.3). Fortunately, by adding an extra term to the ShrinkNet regularization term we can proove that:

$$\min_{\boldsymbol{A},\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{A}\mathrm{diag}\left(\boldsymbol{\beta}\right)\boldsymbol{x}\|_2^2 + \Omega_\lambda^s + \lambda_2 \|A\|_p^p \tag{8}$$

has many global minimum (proposition A.4) for all $p > 0$. This is the reason we defined the *regularized ShrinkNet penalty* earlier this way:

$$\Omega_{\lambda,\lambda_2,p}^{rs} = \lambda \|\boldsymbol{\beta}\|_1 + \lambda_2 \|\boldsymbol{\theta}\|_p^p \tag{9}$$

In practice we observed that $p = 2$ or $p = 1$ are good choice, while the latter will also introduce additional sparsity in the parameters.

3.5. Speeding up training

In order to have a practical implementation of ShrinkNets it is absolutely necessary to prune useless neurons as quick as possible. In order to try more different strategies we split this task in two subproblems: detecting potentially non-crucial neurons and setting their corresponding entry in the switch layer to exactly 0, and neural garbage collection that

3.5.1. Neuron killing

Threshold strategy: We kill neurons based on a threshold (in absolute value), this is the method used by deep compression. It is bad because it is scale dependant and is not robust to noise in the gradients (explain that)
Sign change strategy: We look at when the sign change. This is also sensitive to noise but has the advantage **Sign variance strategy**: We measure the variance of the sign $(-1, 1)$ of each component in the *Switch Layer*, and consider it dead when it goes over a predefined threshold. **Gl:** Maybe we should add some intuition on why it makes sense. **Gl:** We should probably explain the impact of γ and report numbers from the expertiment that show that it does not hurt too much to kill neurons, we have the data in this experiment in the evaluation that we probably want to remove later.

In our library we implemented a variant of the second strategy and the last one.

3.5.2. Neural garbage collection

It is possible to reduce the overhead of the training process by removing neurons as soon as they become deactivated by θ going to 0. To do this, we implemented a *neural garbage collection* mechanism which prunes deactivated neurons on-the-fly, reducing the processing time and memory overhead. To support this feature, it is crucial to understand the information flow between neurons and layers in the neural network. We achieve this by representing such information flow as a graph. Vertices represent layers, and edges are event-hubs responsible for propagating information about disabled neurons to the relevant layers. As soon as a layer receive an event, it updates his parameters to reflect the new size. Each operation is stored in a log that is used later to update other component of the system. For example some optimizers like Adam [Ref] store state about each parameter, their state needs to be updated the exact same way and at the exact same time as layers in order to obtain apropriate parameter updates.

3.6. Speeding up Inference

The sole puropose of *Switch Layer* are to detect which neurons should be killed and which should be left. When the training is over we will never remove more neurons. Therefore, they loose all their interest at inference time. To improve size and inference time we propose the following method to get rid of them without changing the output of the model (modulo floating point errors). The basic idea is to start from every *Switch Layer* and try to find the nearset linear operator (child or parent) and multiply its weights by the values in the switch layer. For example we can merge Filter layers with surrounding Convolutional layer, Linear layer or even BatchNormalization [ref] if they are before the *Switch Layer*. However we need to be extremely careful not to cross a non-linearity otherwise it would change the output of network.

4. Evaluation

4.1. Core evaluation (CHANGE_MY_NAME)

To show the versatility of our aproach we evaluate both on two very different scenario

4.1.1. `CIFAR10` dataset

Gl: Cite and describe the dataset

To solve this task we use the `VGG16` model [ref]. It is constituted of alternating convolutional layers and *MaxPool* layers interleaved by *BatchNorm* [ref] and *ReLU* [ref] layers. The two last layers are Fully connected layers separated by just a *ReLU* activation function.

To turn it into a ShrinkNet we introduce *Switch Layers* after each *BatchNorm* and each Fully connected (except the last one).

ShrinkNets assume that the starting size of the network is an upper bound on the optimal size. We though that picking two times the recommended size for each layer (that was designed for ImageNet[ref]), is a generous upper bound. For the classification layers we use 5000 neurons as an upper bound where the ImageNet version uses 4096 **Gl:** This is on the top of my head, need to be double checked.

We assume no prior knowledge on the optimal batch size, learning rate, λ or weight decay (λ_2). This is why, we randomly sample them from a range of reasonable values (**Gl:** should we make them explicit ?). For each of the models we trained, we pick the epoch with the best validation accuracy and report the corresponding testing accuracy. For each model, we also measure the total size, in number of floating point parameters, excluding the *Switch Layers* because as we saw in section **Gl:** REFERENCE_ME, we can get rid of them when training is done.

We want to compare against classical (*Static*) networks. The number of parameters that control the size is large: 13 for the size of convolutional layers and 2 for the fully connected ones. Without Shrinknets to help and fuse all these parameters in a single λ it is infeasible to sample reasonably well a search space of that size. This is why we have to rely on the very well known heuristic that the original VGG architecture (and many CNNs) **Gl:** try to find the paper that introduces this heuristic. For *Static Networks* we sample the size between 0.1 and 2 times the size optimized for ImageNet. We report the same numbers as we did for *ShrinkNets* and we compare the two distributions of models we obtain on the first plot of fig. 3.

4.1.2. COVERTYPE DATASET

We followed the same procedure **Gl:** Cite and describe the dataset

We try to fit

4.2. Benefits of smaller sizes

We showed in the previous section that *ShrinkNets* were able to find more or at least as efficient as *Static Networks*. In this experiment we want to determine the perf

4.3. Multi-Target Linear and Multi-Class Logistic regressions

As we showed, Group sparsity share similarities with our method, and we claim that ShrinkNets are a relaxation of group sparsity. In this experiment we want to compare the two aproaches. We decided to focus on multi-target linear regression because in the single target case, groups in the Group Sparsity problem would have a size of one ($\boldsymbol{A}$ would be a vector in this case).

The evaluation will be done on two datasets `scm1d` and `oes97` [ref] for linear regressions and we will use `gina_prior2` [ref] and the *Gas Sensor Array Drift Dataset* [ref] (that we shorten in `gsadd`) for logistic regressions.

For each dataset we fit with different regularization parameters and measure the error and sparsity obtained after convergence. In this context we define sparsity as the ratio of columns that have all their weight under 10^{-3} in absolute value. Regularization parameters were choosed in order to obtain the widest sparsity spectrum. Loss is normalized depending on the problem to be in the $[0, 1]$ range. We summarized the results in fig. 1. From our experiments it is clear that ShrinkNets can fit the data closer than Group Sparsity for the same amount of sparsity. The fact that we are able to reach very low loss demonstrate that even if our objective function is non convex, in practice it works as good or better as convex alternatives. Details about these datasets and the parameters used are available in appendix A.2.1.

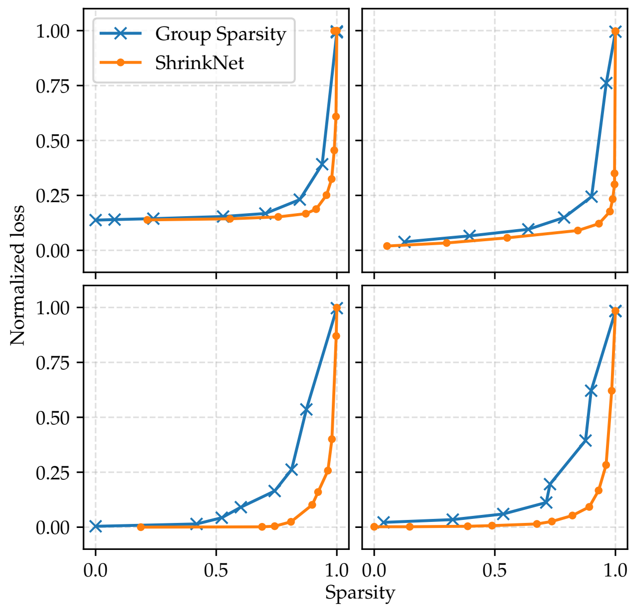

Figure 1. Loss/Sparsity trade off comparison between Group Sparsity and Shrinknet on linear and logistic regression. From top to bottom and left to right we show the results for `scm1d`, `oes97`, `gina_prior2` and `gsadd`.

4.4. Neuron Removal strategies

In our previous experiment, we showed that the ShrinkNet loss was reasonable in practice, we are now interested in the impact on early pruning. The method we suggest for early prunning uses a parameter γ that control the aggressiveness of neuron removal so we will try to evaluate its impact on the final loss achieved by the model and the cost required to train the model. Our cost model is simple and hardware independant, we sum the number of input neurons at each epoch. In theory the cost in time should be asymptotically linear to this metric. To have a baseline we also train the same model but without neuron removal. Keep in mind that this is just in order to have some reference. Indeed, if we were to remove the neurons with small weights it would deteriorate the loss (and picking the threshould would be completely arbitrary). Therefore the baseline is evaluated with all neurons. One could consider it as a ”theoretical lower bound” of the best achievable loss.

We picked multiple combinations of dataset and regularization parameters (λ) and for each we fit with different aggressiveness parameters (γ). We measure the loss after convergence and the total cost and report the result in fig. 2. In order to reduce the noise in the result, each experiment was performed 30 times and we display the range arround $\pm$ 1 standard deviation.

TODO: Interpret the results

5. Speeding up training with pruning

6. Related Work

- Hyperparameter optimization: random, bayesian opt, bandit methods
- distillation techniques
- post-training compression techniques
- group sparsity, non-parametric neural networks
- training dynamics paper: first overfitting and then randomization?

Given the importance of network structure, many techniques have been proposed to find the best network structure for a given learning task. These techniques broadly fall into four categories: hyperparameter optimization strategies, post-training model compression for inference as well as model simplification, techniques to resize models during training, and automated architecture search methods.

The most popular techniques for hyperparameter optimization include simple methods such as random search (**?**) which have been shown to work suprisingly well compared to more complex methods such as those based on Bayesian optimization (Snoek et al., 2012). Techniques such as (Snoek et al., 2012) model generalization performance via Gaussian Processes (Rasmussen & Williams, 2006) and select hyperparameter combinations that come from uncertain areas of the hyperparameter space. Recently, methods based on bandit algorithms (e.g. (Li et al., 2016; Jamieson & Talwalkar, 2016)) have also become a popular way to tune hyperparameters. As noted before, all of the above techniques require many tens to hundreds of models to be trained, making this process computationally inefficient and slow.

In contrast with hyperparameter tuning methods, some methods such as DeepCompression (Han et al., 2015) seek to compress the network to make inference more efficient. This is accomplished by pruning connections and quantizing weights. On similar lines, multiple techniques such

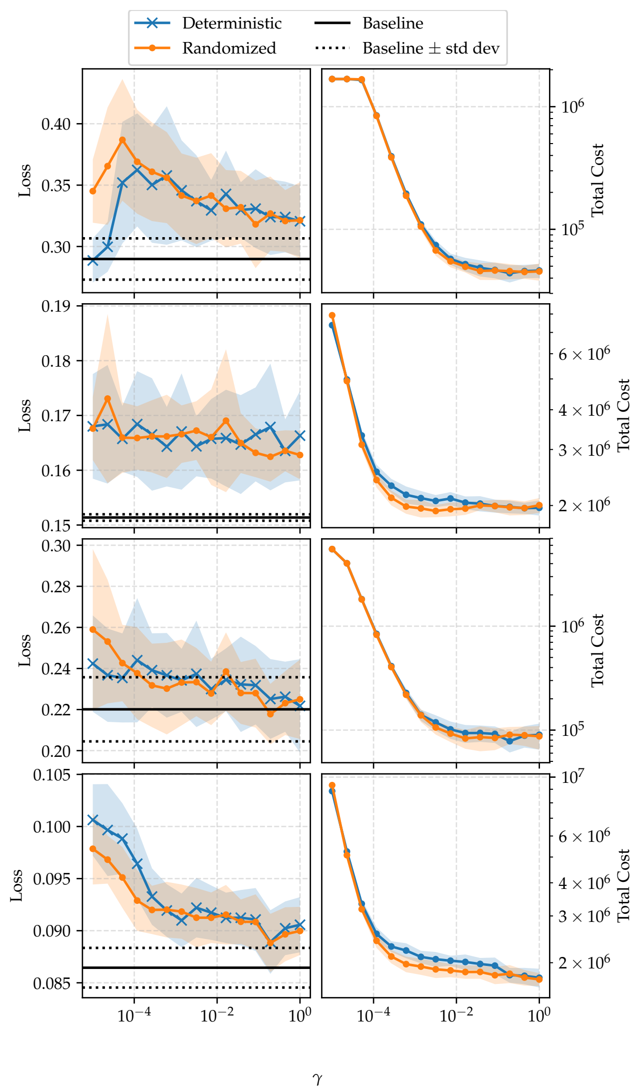

Figure 2. Effect of dynamic neuron removal for different γ. First column is the difference in the final loss in function of the removal factor. We plot theoretical baseline as a reference. Right column is a proxy of the total cost for training the model (i.e. the sum of input neurons at each epoch). Each row is a dataset/λ combination. From top to bottom we have: `scm1d`/0.1, `oes97`/0.1

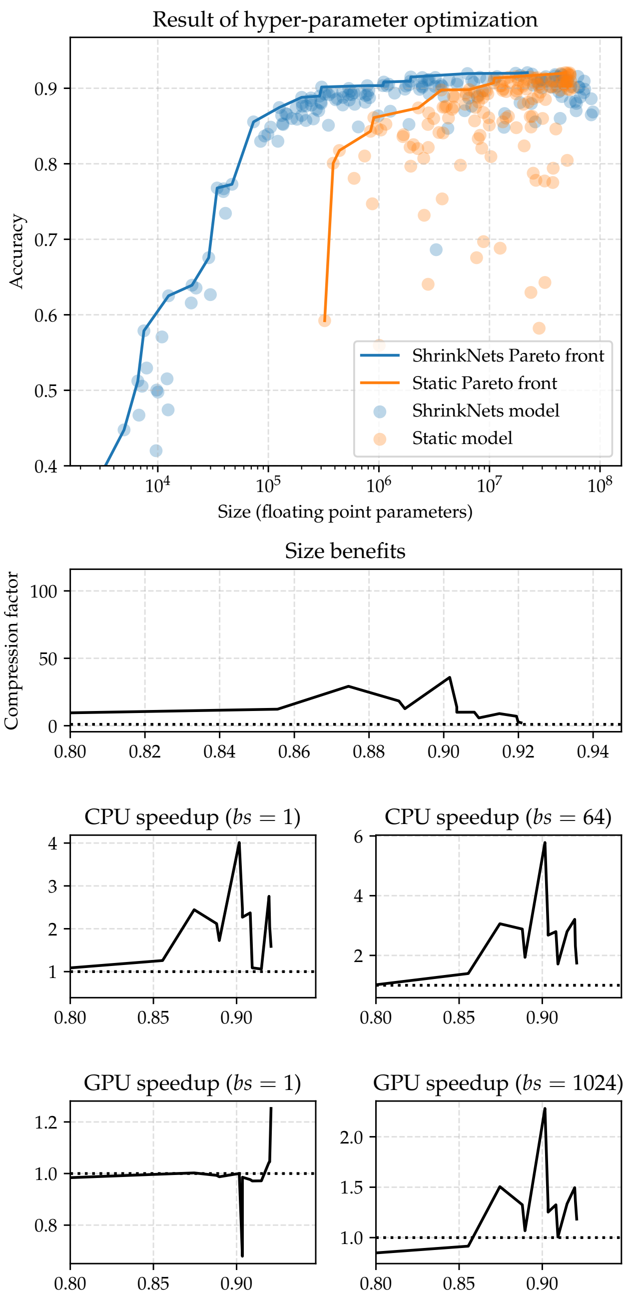

Figure 3. Summary of the result of random search over the hyper-parameters the `CIFAR10` dataset

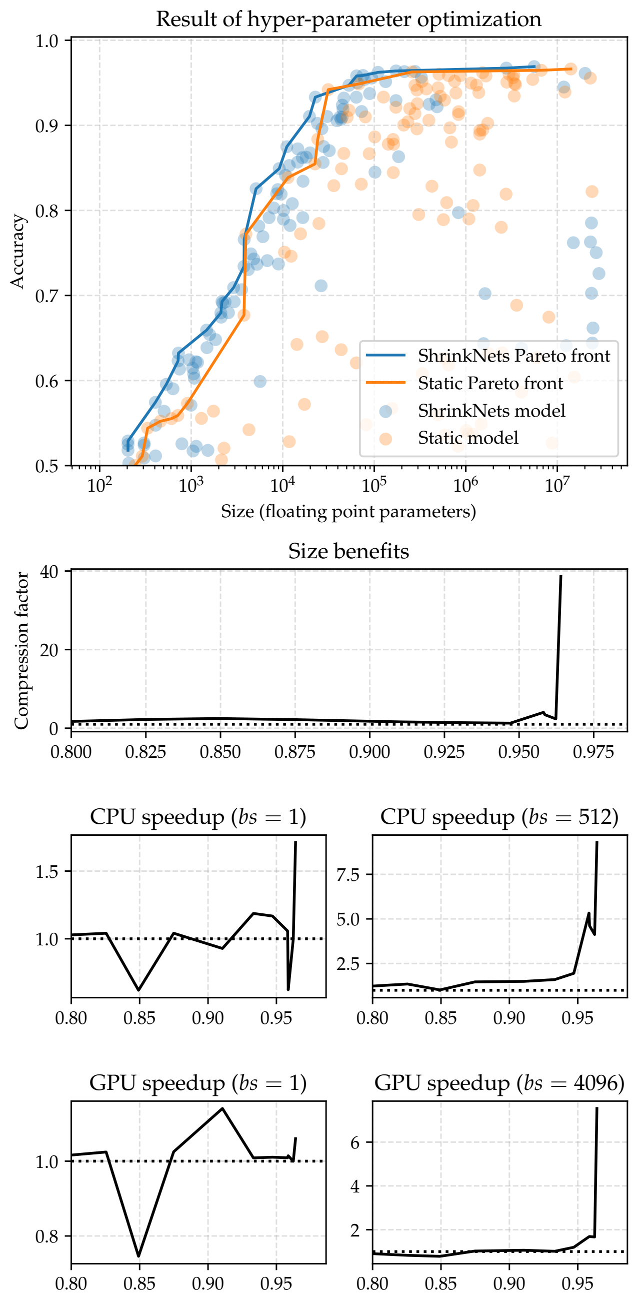

Figure 4. COVER

as (Romero et al., 2014; Hinton et al., 2015) have been proposed for distilling a network into a simpler network or a different model. Unlike our technique which works during training, these techiques are used after training and it would be interesting to apply them to ShrinkNets as well.

The techniques closed to our work are those based on group sparsity such as **MV:** Guillaume: fill in.

Finally, there has also been recent work in automatically learning model architecture through the use of genetic algorithms and reinforcement learning techniques (Zoph & Le, 2016; Zoph et al., 2017). These techniques are focused more on learning higher-level architectures (e.g. building blocks for neural network architectures) as opposed to learning network size.

7. Discussion

- Where does the method shine?
- Side-effects of method: smaller networks, time to train
- Potential extensions/limitations

8. Conclusion

A. Appendix

A.1. Proofs

Unless specified, all the proofs consider the Multi-Target linear regression problem

Proposition A.1. $\forall (n,p) \in \mathbb{N}_+^2, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{x} \in \mathbb{R}^p \lambda \in \mathbb{R}$

$$\begin{aligned}
&\min_{\mathbf{A}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} \left\| \left(A^T\right)_j \right\|_2 \\
&= \min_{\mathbf{A}', \boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A}' \mathit{diag}\left(\boldsymbol{\beta}\right) \mathbf{x}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \\
&\mathit{s.t.} \forall j \in [\![1,p]\!], \left\| \left(A'^T\right)_j \right\|_2^2 = 1
\end{aligned}$$

Proof. In order to prove this statement we will show that for any solution $\boldsymbol{A}$ in the first problem, there exists a solution in the second with the exact same value, and vice-versa. We now assume we have a potential solution $\boldsymbol{A}$ for the first problem and we define $\boldsymbol{\beta}$ such that $\boldsymbol{\beta}_j = \left\| \left(\boldsymbol{A}^T\right)_j \right\|_2^2$, and $\boldsymbol{A}' = \boldsymbol{A} \left(\operatorname{diag}\left(\boldsymbol{\beta}\right)\right)^{-1}$. It is easy to see that the constraint on $\boldsymbol{A}'$ is statisfied by construction. Now:

$$\begin{aligned}
&\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} \left\| \left(A^T\right)_j \right\|_2 \\
&= \|\mathbf{y} - \mathbf{A}' \operatorname{diag}\left(\boldsymbol{\beta}\right) \mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} \left\| \left(A'^T\right)_j \beta_j \right\|_2 \\
&= \|\mathbf{y} - \mathbf{A}' \operatorname{diag}\left(\boldsymbol{\beta}\right) \mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} |\beta_j| \cdot 1 \\
&= \|\mathbf{y} - \mathbf{A}' \operatorname{diag}\left(\boldsymbol{\beta}\right) \mathbf{x}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1
\end{aligned}$$

Assuming we take an $\boldsymbol{A}'$ that satisfy the constraint and a $\boldsymbol{\beta}$, we can define $\boldsymbol{A} = \boldsymbol{A'} \operatorname{diag}\left(\boldsymbol{\beta}\right)$. We can apply the same operations in reverse order and obtain an instance of the first problem with the same value. We can now see that the two problems must have the same minimum otherwise we would be able to construct a solution to the other with exact same value. □

Proposition A.2.

$$\|\mathbf{y} - \mathbf{A}\mathit{diag}\left(\boldsymbol{\beta}\right) \mathbf{x}\|_2^2$$

is not convex in $\boldsymbol{A}$ and $\boldsymbol{\beta}$.

Proof. To prove this we will take the simplest instance of the problem: with only scalars. We have $f(a, \beta) = (y - a\beta x)^2$. For simplicty let's take $y =$ and $x > 0$. If we take two candidates $s_1 = (0, 2)$ and $s_2 = (2, 0)$, we have $f(s_1) = f(s_2) = 0$. However $f(\frac{2}{2}, \frac{2}{2}) = x > \frac{1}{2} f(0, 2) + \frac{1}{2} f(2, 0)$, which break the convexity property. Since we showed that a particular case of the problem is non-convex then necessarly the general cannot be convex. □

Proposition A.3.

$$\min_{\mathbf{A}, \boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A}\mathit{diag}\left(\boldsymbol{\beta}\right) \mathbf{x}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1$$

has no solution if $\lambda > 0$.

Proof. Let's assume this problem has a minimum $\boldsymbol{A}^*, \boldsymbol{\beta}^*$. Let's consider $2\boldsymbol{A}^*, \frac{1}{2}\boldsymbol{\beta}^*$. Trivially the first component of the sum is identical for the two solutions, however $\lambda \left\| \frac{1}{2}\boldsymbol{\beta} \right\| < \lambda \|\boldsymbol{\beta}\|$. Therefore $\boldsymbol{A}^*, \boldsymbol{\beta}^*$ cannot be the minimum. We conclude that this problem has no solution. □

Proposition A.4. *For this proposition we will not restrict ourselves to single layer but the composition of an an arbitrary large (n) layers as defined individually as $f_{\boldsymbol{A}_i, \boldsymbol{\beta}_i, \boldsymbol{b}_i}(x) = a(\boldsymbol{A_i} \mathit{diag}\left(\boldsymbol{\beta_i}\right) \boldsymbol{x} + \boldsymbol{b_i})$. The entire network follows as: $N(\boldsymbol{x}) = \left(\bigcirc_{i=1}^{n} f_{\boldsymbol{A_i}, \boldsymbol{\beta_i}, \boldsymbol{b_i}}\right)(\boldsymbol{x})$. For $\lambda > 0$, $\lambda_2 > 0$ and $p > 0$ we have:*

$$\min \|\boldsymbol{y} - N(\boldsymbol{x})\|_2^2 + \Omega_{\lambda, \lambda_2, p}^{rs}$$

has at least 2^k global minimum where $k = \sum_{i=1}^{n} \# \boldsymbol{\beta_i}$

Proof. First let's prove that there is at least one minimum to this problem. The two components of the expression are always positive so we know that this problem is bounded by below by 0. Let's assume this function does not have a minimum. Then there is a sequence of parameters $(S_n)_{n>0}$ such that the function evaluated at that point convereges to the infimum of the problem. Since the function is defined everywhere does not have a minimum then this sequence must diverge. Since the entire sequence deverge the there is at least one individual parameter that diverges. First case, the parameter is a component k of some $\boldsymbol{\beta_i}$ for some i. Necessarly $\|\boldsymbol{\beta_i}\|_1$ diverge towards $+\infty$, which is incompatible with the fact that (S_n) converges to the infimum. We can have the exact same argument if the diverging parameter is in $\boldsymbol{A_i}$ or $\boldsymbol{b_i}$ because $p > 0$. Since there is always a contradiction then our assumption that the function has no global minimum must be false. Therefore, this problem has at least one global minimum.

Let's consider one optimal solution of the problem. For each component k of $\boldsymbol{\beta_i}$ for some i. Negating it and negating the k^{th} column of $\boldsymbol{A_i}$ does not change the the first part of the objetive because the two factors cancel each other. The two norms do not change either because by definition the norm is independant of the sign. As a result these two sets of parameter have the same value and are both global minimum. It is easy to see that going from this global minimum we can decide to negate or not each element in each $\boldsymbol{\beta_i}$. We have a binary choice for each parameter, there are $k = \sum_{i=1}^{n} \#\boldsymbol{\beta_i}$ parameters, so we have at least 2^k global minima.

□

A.2. Experiment details

A.2.1. Multi-Target Linear and Logistic regressions

A.2.2. Neuron Removal strategies

A.2.3. Convergence and Training Dynamics of Neural networks

A.2.4. Hyper-optimization of ShrinkNets

A.2.5. Performance

References

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jamieson, Kevin and Talwalkar, Ameet. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pp. 240–248, 2016.

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, and Talwalkar, Ameet. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Rasmussen, CE and Williams, CKI. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, 1 2006. ISBN 0-262-18253-X.

Romero, Adriana, Ballas, Nicolas, Kahou, Samira Ebrahimi, Chassang, Antoine, Gatta, Carlo, and Bengio, Yoshua. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 2951–2959. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-l pdf.

Vanschoren, Joaquin, van Rijn, Jan N., Bischl, Bernd, and Torgo, Luis. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. URL http://arxiv.org/abs/1611.01578.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.