ICLR REPRODUCIBILITY CHALLENGE
SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSIVITY

Szymon Mikler, Elżbieta Plaszczyk
sjmikler@gmail.com

ABSTRACT

In this report we present results from reproduction of SNIP: one-shot neural network pruning method. We found this method very intriguing and we found that it prunes the network in a smart way, although we were unable to reproduce results as good as authors presented in original paper. We belive this is due to implementation details and not fully explained learning algorithm that authors used. In the report not only we test accuracy in similar conditions as authors, but also we present our own analysis of how SNIP method chooses unimportant connections.

1 DESCRIPTION OF THE METHOD

One can describe neural network's loss sensivity on zeroing weight w_j as $\widehat{s_j} = |L(w_j) - L(0)|$. Intuitivly, if such sensivity is small, then removing parameter, since it didn't affect the loss function, shouldn't prevent neural network from learning. Although, one usually wants to remove more than one parameter and, since they can be dependent from each other, to get the least loss affecting set of η parameters to prune, one needs to check every possible combination of η, thus make this many forward passes in a network. Since this is impossible, we are assuming that influence of w_j on ΔL ($|L(w_j) - L(0)|$) is independent from every other weight than w_j.

This is the first assumption we make. But since computing $\widehat{s_j}$ for every weight w_j separately is computionaly very expensive – it need m forward passes to compute (for m being number of parameters in the network) – we will assume that $|L(w_j) - L(0)|$ corresponds to $|L(w_j) - L(w_j - \varepsilon)|$ for infinitesimaly small ε.

This second assumption is the one that describes SNIP method. With it we can define salience s_j of weight w_j as $s_j = \frac{\partial L(w_j \cdot c_j)}{\partial c_j}$ for c_j being an indicator equal to 1. Such salience calculation can be done in modern frameworks very efficiently, for all the weights at once. Salience can be then standarized to see parameters' importances in percentages. One can try to determine salience of a weight using other criteria and a few of these will be described and compared to SNIP.

2 IMPLEMENTATION DETAILS

All pruning methods described in this report were compared on labels prediction task for MNIST dataset on a few different network architectures (the simplest - network with one hidden layer, LeNet-300-100, LeNet-5 and VGG network suited for MNIST). Network was trained on 60000 images and then tested on remaining 10000. All pruning techniques compared here were using following algorithm:

1. Randomly initialize weights, here usually with Xavier initialization (either uniform or normal),
2. One-shot prune the network to given sparsity level κ. In other words, set to 0 all the parameters w_j for which salience $s_j \leq s_\kappa$, where s_κ is κ-th salience in terms of magnitude out of all from the network (s_j is defined separately for each pruning technique),

3. Train the network with standard training techniques, here Adam optimizer with 0.0012 decaying learning rate was used most often (due to easier hyperparameters tuning than with SGD).

All the experiments were performed using Google Cloud Platform virtual machine with Tesla P100 GPU. The algorithms were implemented in Python and PyTorch and are published on github[1].

3 COMPARISION BETWEEN SNIP AND PRUNING WITH OTHER SALIENCE CRITERIA

The same tests for 5 different one-shot pruning criteria were run for 12 different sparsities ranging from 5% to 96% and for every such setting 10 lerning sessions (80 epochs each) were held and for each of them, the best validation score was choosen. Please note that for 96% sparsity LeNet-5 architecture has roughly 2000 parameters left, while training set consists of 60000 training examples and each of them is 784-dimensional. Even though, with proper pruning method, network is able to fit this dataset and predict labels of unseen data with acceptable accuracy.

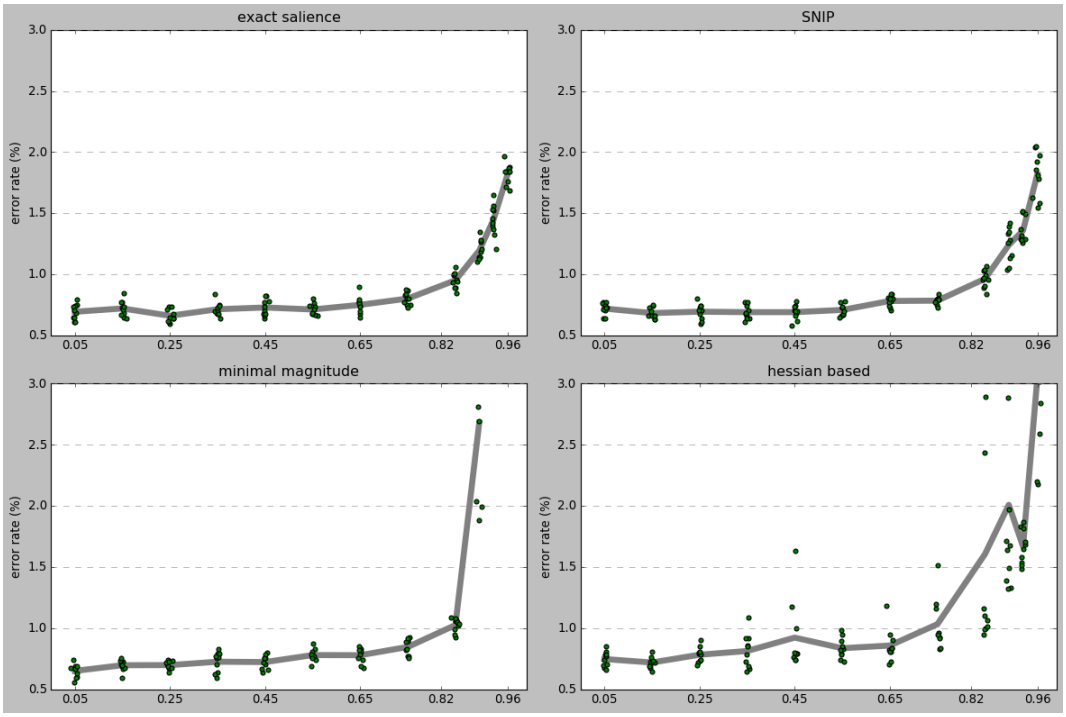

Figure 1: Error rates in label prediction task collected from 600 tests after 80 epochs long training of LeNet-5 network for various sparsity levels (x axis)

For visualization purposes, accuracies worse than 85% (so basically when the network gave random answers) were removed. It can be assumed, that these results were random due to structural damage in network architecture that prevented it from learning. In the following table one can see, what pruning methods and what sparsity levels produced such results.

[1] https://github.com/gahaalt/SNIP-pruning-method-analysis

method↓ sparsity→	75%	85%	90%	93%	96%
EXACT SALIENCE	0	0	0	0	0
SNIP	0	0	0	0	0
MINIMAL MAGNITUDE	0	0	4	10	10
HESSIAN BASED	1	0	0	0	0
RANDOM PRUNING	0	0	0	0	0

Figure 2: number of network destructions out of 10 learning sessions

From these charts, one can deduce that minimal magnitude pruning method is not well suited for one-shot pruning to extreme sparsities and even random pruning behaves better in the task. Although it may be explained by our choice of initialization method. Since we take into consideration the number of neurons from a layer, big layers usually have the smallest weights from the network and they might be pruned entirely. SNIP method seems to be very dependable, which is opposite to hessian and magnitude based methods.

Also one can find interesing the comparision between SNIP and the exact $|\Delta L|$ that SNIP, as stated at the beginning, tries to approximate. One can see a closer comparision between the two below.

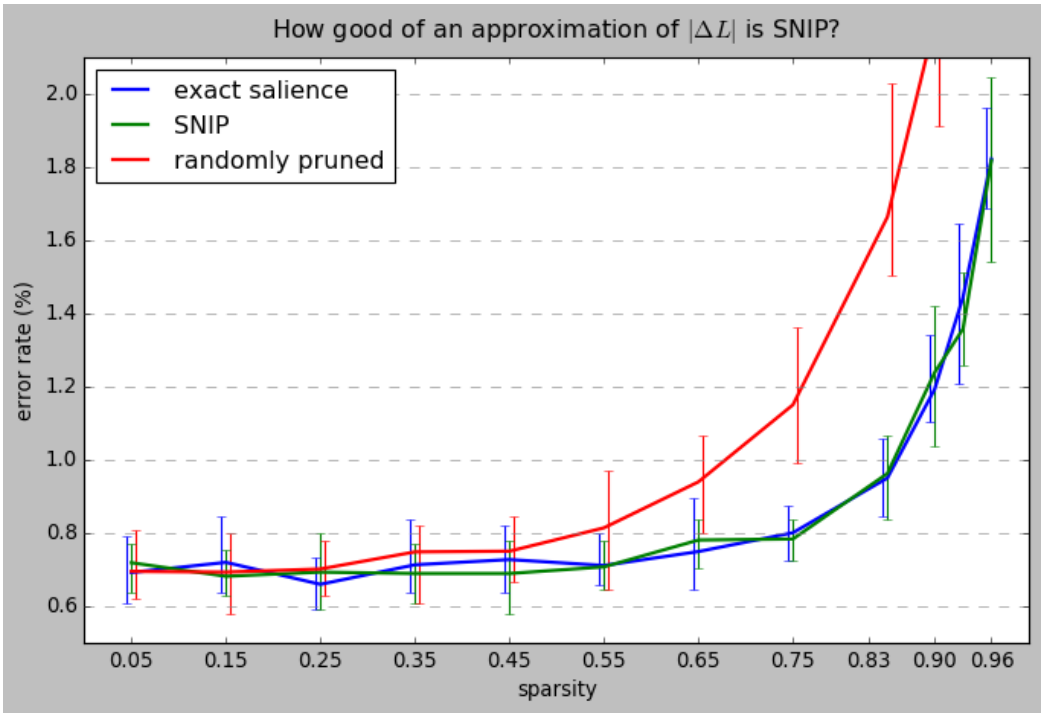

Figure 3: Comparision between exact salience criterium and SNIP

Both for SNIP and exact salience, error rate after 96% prune is twice as big from the original one. When training with different schedules, more precisely with SGD (initial learning rate: 0.1, momentum: 0.9, learning rate decay: 0.1 every 40 epochs, weight decay: 0.0005), the best achieved result was roguhly 1.4 error rate, however it took much more time to train the network, and it was hardly repeatable. It is worth to mention, even though exact salience is much more computionaly expensive to calculate than SNIP, it is possible to compute it for big networks like VGG in matter of minutes. Possibly if the network architecture was more complicated, SNIP wouldn't be as good of an approximation of $|\Delta L|$ as it is for LeNet-5.

DESCRIPTION OF CRITERIA

Random pruning has no salience criterium at all and criterium for SNIP was defined before. Remaining definitions of salience s_j for weight w_j are as follows:

- Exact salience: $s_j = |L(w_j) - L(0)|$
- Minimal magnitude: $s_j = |w_j|$
- Hessian based: $s_j = w_j^2 \cdot \frac{\partial^2 L}{\partial w_j^2}$

3.1 ADDITIONAL TESTS

In the following comparision, each learning session was run for 60 epochs on a simple neural network and after that its accuracy was checked and collected.

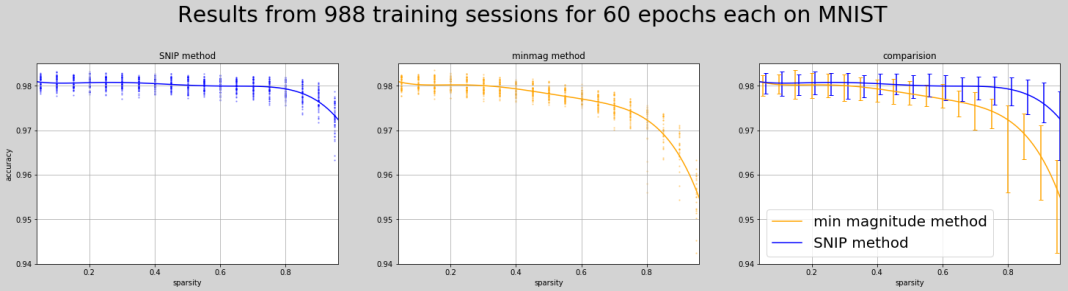

Figure 4: Accuracy comparision between SNIP and minimal magnitude salience criteria collected from large number of learning sessions of a simple neural network with one hidden layer pruned with sparsity varying from 5% to 95%.

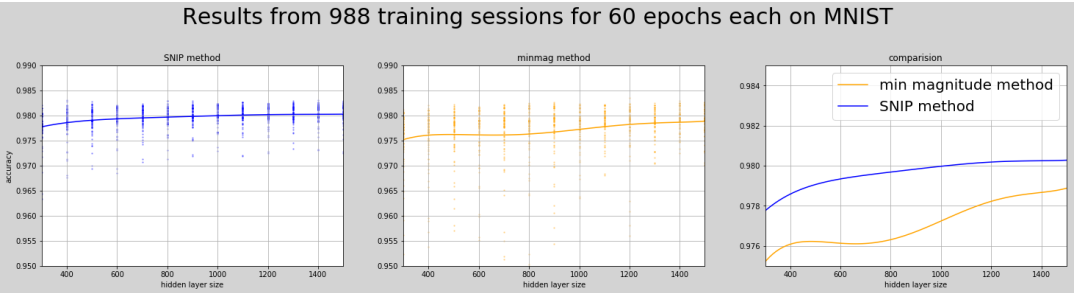

Figure 5: Accuracy comparision between SNIP and minimal magnitude salience criteria collected from large number of learning sessions of a simple neural network with one hidden layer pruned with hidden layer size varying from 300 to 1500.

Again, a fraction of minimal magnitude pruning results ended approximately at 10% accuracy level. Such results were removed from the data to prevent linear reggresion interference. Nevertheless, these cases occured for bigger architecures only, specificaly for hidden layer sizes between 1200 and 1500 and also for rather bigger sparsities. Such occurences didn't happen for SNIP even once, which suggests that SNIP method chooses which weights to prune sensibly - in a way that prevents from such architectural damage.

4 CLOSER LOOK AT HOW SNIP WORKS

To fully exploit and improve pruning methods it may be worth to analyse and establish why SNIP works (usually) better than other pruning methods and how it is able to choose non-important connections even before the training is held. Such analysis is provided in this section.

4.1 WHAT LAYERS ARE BEING PRUNED?

An interesting thing one can look at is quantity of weights in each layers of a network after SNIP pruning. In case of simple feedforward network, we have got results that makes us belive SNIP actually knows, how many parameters in which layer network needs for learning to proceed.

layer	**params**	50% **sparse**	95% **sparse**
full 1	235200	111628 (52.5%)	7693 (96.7%)
bias full 1	300	176 (41.3%)	2 (99.3%)
full 2	30000	20501 (31.7%)	5026 (83.2%)
bias full 2	100	85 (15.0%)	40 (60.0%)
full 3	1000	905 (09.5%)	562 (43.8%)
bias full 3	10	10 (00.0%)	8 (20.0%)
result	1.30%	1.46%	2.16%

Figure 6: LeNet-300-100 layer sparsities statistics after pruning with SNIP with weights' saliences evaluated on whole batch of MNIST dataset and validation accuracies for comparision.

layer	**params**	50% **sparse**	95% **sparse**
conv 1	150	143 (04.7%)	106 (29.3%)
bias conv 1	6	6 (00.0%)	3 (50.0%)
conv 2	2400	2118 (11.8%)	666 (72.3%)
bias conv 2	16	16 (00.0%)	6 (62.5%)
full 1	48000	22815 (52.5%)	1214 (97.5%)
bias full 1	120	94 (21.7%)	22 (81.7%)
full 2	10080	5022 (50.2%)	768 (92.4%)
bias full 2	84	66 (21.4%)	41 (51.2%)
full 3	840	563 (33.0%)	253 (69.9%)
bias full 3	10	10 (00.0%)	7 (30.0%)
result	0.65%	0.55%	1.33%

Figure 7: LeNet-5 layer sparsities statistics after pruning with SNIP with weights' saliences evaluated on whole batch of MNIST dataset and validation accuracies for comparision.

One could use such statistics also at the end of the training to conclude, which layers are overparametrizied and have no effect on the loss. Given loss function change is approximated well by SNIP method, then this should give very accurate results.

Also, it is worth mentioning, that these structures stay consinstent. In other words, parameters quantities on layers stay roughly the same when SNIP procedure is run second time with other set of randomly initialized parameters.

An interesting test to make is to prune VGG network (VGG16 architecture with smaller classificator and with dropout replaced by batch normalization) and then fit it to MNIST dataset. For 99% of sparsity (so having only half of the of parameters that LeNet-300-100 has!), following behaviour was observed:

- almost all (except last) bias layers were pruned completly (100% sparsity);
- biggest layers, so the ones that contain 2359296 parameters (there are 5 of them), were pruned to sparsity in consequent range 99.6% - 99.9%;
- as a result, obtained error rate was 0.41%, but network training, especially at the beginning, was rather unpredictable and unstable (shown in a graph below);
- accuracy for dense network with the same settings reached at most 0.33%;
- in summary, SNIP method chose 1% of important connections from very huge and overparameterized network.

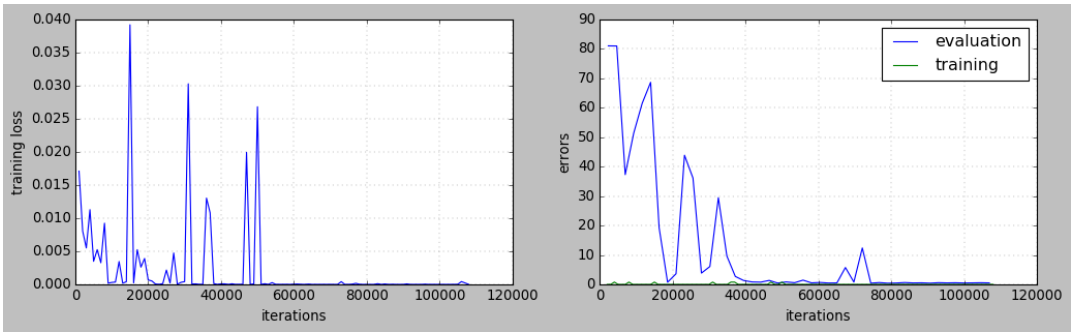

Figure 8: training process for 99% sparse VGG, resulting in 99.59% accuracy

4.2 WHAT WEIGHTS ARE BEING PRUNED?

When pruning network with magnitudes as salience criterion, we expect our weights' distribution to be zeroed in the center of the initial distribution and untouched on the sides. On the other hand, when using SNIP method, it is not obvious, what distribution weights are from.

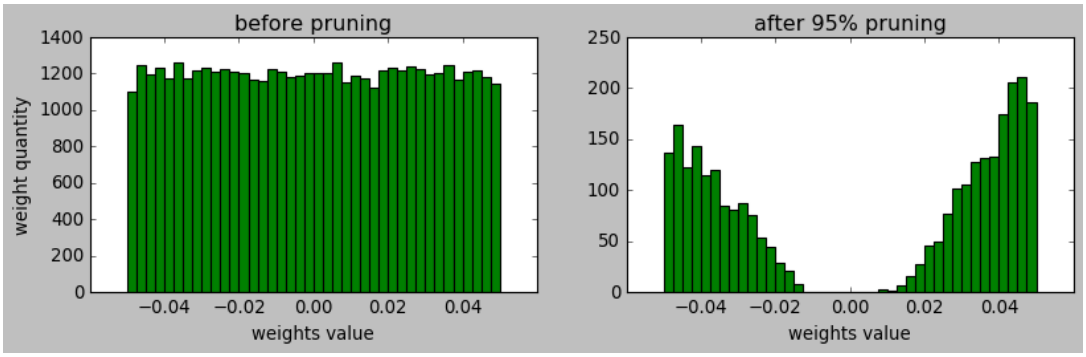

Figure 9: Distribution of the weights in the biggest layer of LeNet-5 network before (on the left) and after pruning (on the right) with uniform Xavier initialization. We observe that at the center weights are zeroed (since salience is 0), but many of only average magnitude weights were sustained, unlike when pruning by minimal magnitudes.

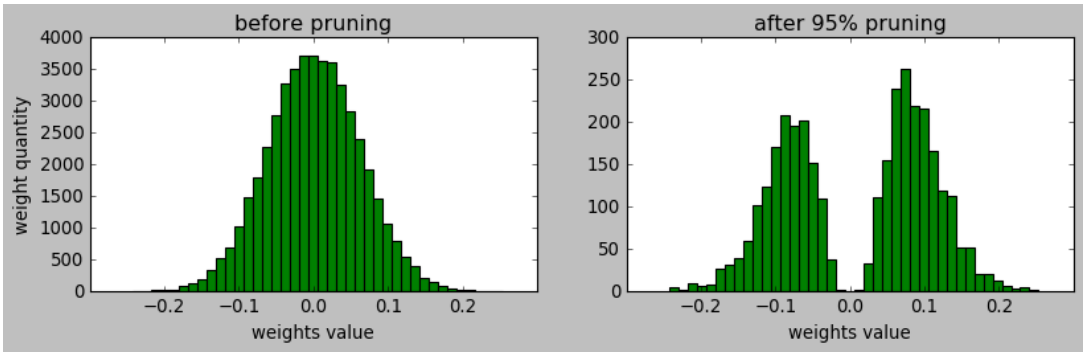

Figure 10: When normal Xavier initialization was used, the result were similar. The histogram here shows the distribution of the biggest layer of LeNet-5.

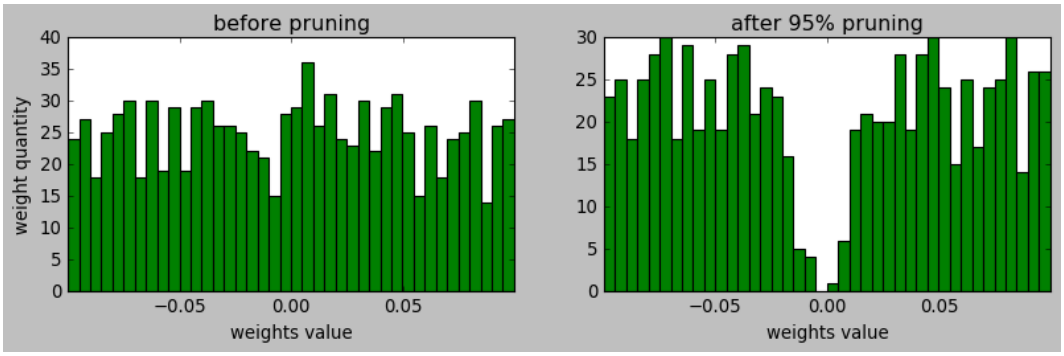

Figure 11: Similar results can be seen with every architecture. Here one can see distribution of the weights from last layer of LeNet-300-100. It is worth noticing, that this effect is not that visible in last layers, probably due to the fact, that less connections are pruned there.

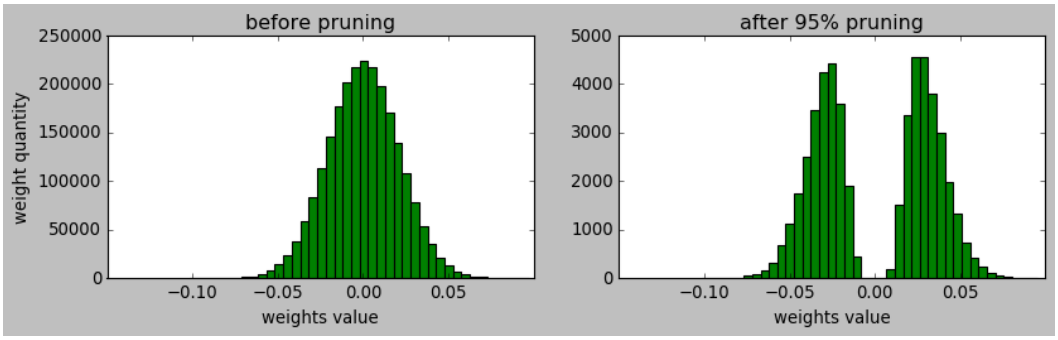

Figure 12: As it was said, the same pattern holds for different architectures. Here one can see the distribution of one of the biggest layers in VGG network.

Except for the center of the histogram, plots of a pruned network look very similar to the originals. One can bring a conclusion here, that SNIP method doesn't care about weight's magnitude (except for the smallest), and prunes almost constant percentage of weights from bins with sufficently big magnitude, thus general architectural results of pruning are not dependent on random initialization.

4.3 WHICH NEURON'S CONNECTIONS ARE BEING PRUNED?

One might be interested, which one is the case for neurons, after pruning is done:

- Connections of a neuron are pruned roughly equally, resulting in a real sparse network, meaning each neuron sustains a few important connections to few neurons in next layer,

- A few neurons remain fully connected and a few neurons are pruned entirely, in this case pruning does not make sense, since we could make a smaller network from the start.

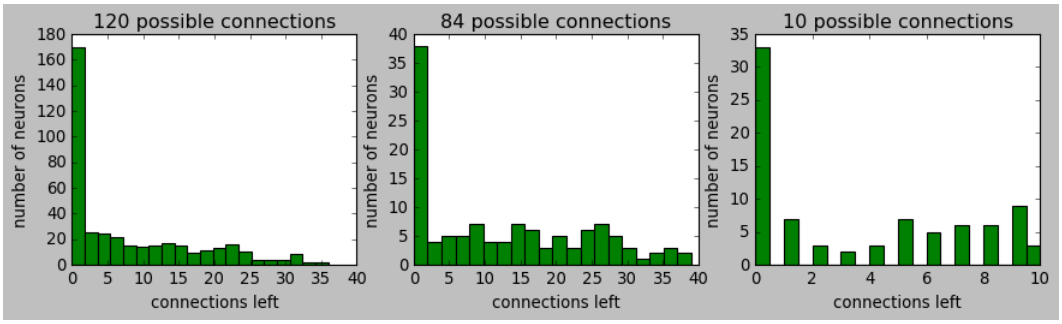

Figure 13: Statistics of weights remaining in LeNet-5 fully connected layers after 90% prune using SNIP method

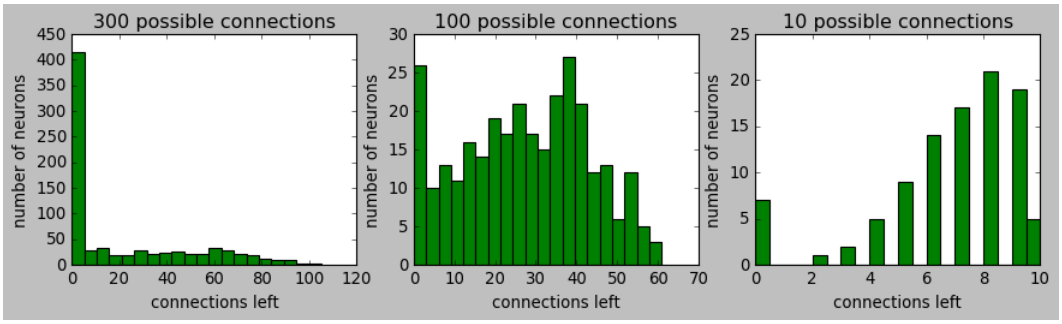

Figure 14: Statistics of weights remaining in LeNet-300-100 fully connected layers after 90% prune using SNIP method

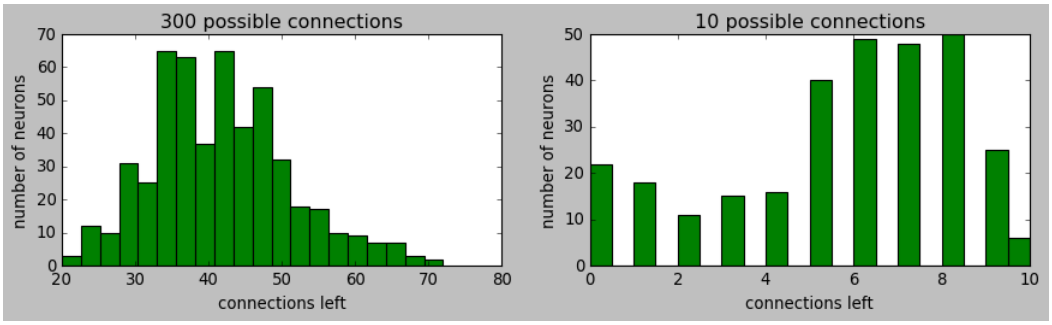

Figure 15: Statistics of weights remaining in VGG classifier layers after 90% prune using SNIP method

There is no definitive answer to this question. On MNIST dataset, since bigger part of the image is white background, most input neurons are being entirely pruned. But then it is sometimes the case, that fully connected neurons sustains roughly constant part of their connections and almost none of the neurons are entirely zeroed. Although that's not always the case. In LeNet-5 it seems like making last but one layer smaller from the beginning wouldn't make any difference. Furthermore, one can observe similar behaviour in convolutional layers, when looking at the kernels' statistics.

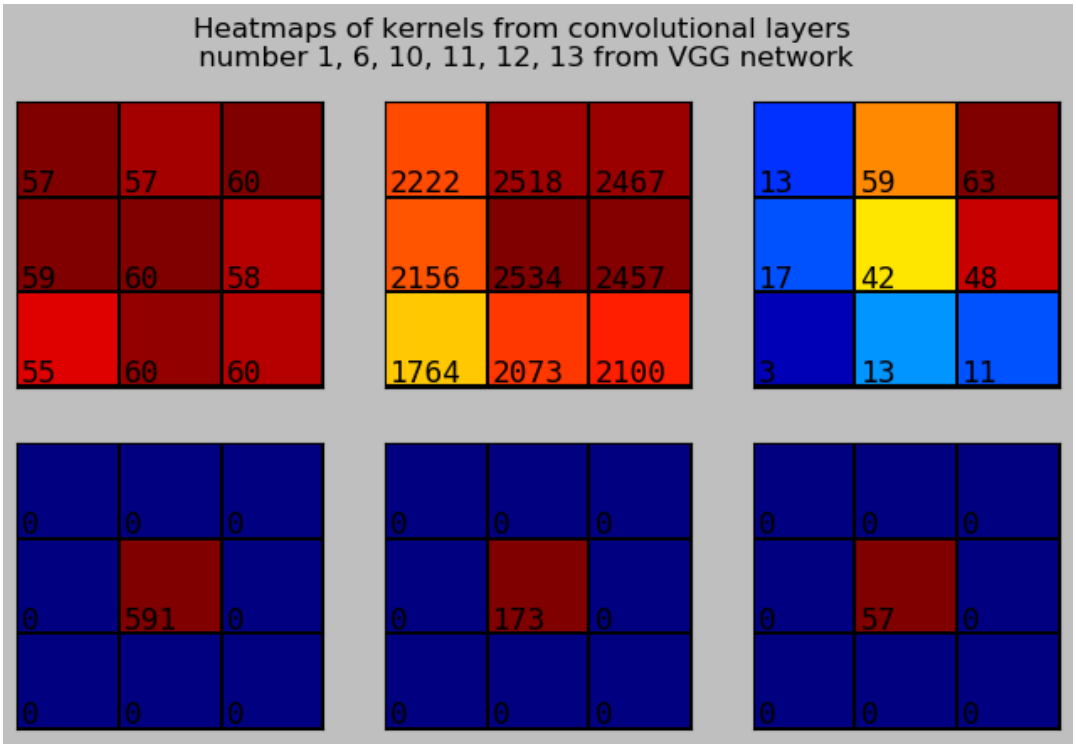

Figure 16: Heatmaps of kernels' (3×3) connections remaining after 99% SNIP prune in VGG network in a few choosen feature extractor layers. One can observe, that three last kernels are pruned in a way which suggests that these layers were the least important in the network architecture, thus probably could have been skipped without loss of accuracy.

5 WHY DO WE PRUNE?

Upgrading pruning methods is an important task in neural networks research. Sparse neural networks might, in fact, overcome their dense precedents, because they are less likely to overfit the data they train on. Moreover, with usage of appropriate frameworks, training such networks can be much faster than training dense ones. We belive SNIP method can bring a good approximation of the real importance of a parameter in neural network and, as a result, allow us to choose the set of weights that are necessary and sufficient for network to learn.