Towards Network Simplification for Low-Cost Devices by Removing Synapses

Martin Bulín(✉), Luboš Šmídl, and Jan Švec

Department of Cybernetics, University of West Bohemia, Pilsen, Czech Republic
{bulinm,smidl}@kky.zcu.cz, jan.svec@speechtech.cz

Abstract. The deployment of robust neural network based models on low-cost devices touches the problem with hardware constraints like limited memory footprint and computing power. This work presents a general method for a rapid reduction of parameters (80–90%) in a trained (DNN or LSTM) network by removing its redundant synapses, while the classification accuracy is not significantly hurt. The massive reduction of parameters leads to a notable decrease of the model's size and the actual prediction time of on-board classifiers. We show the pruning results on a simple speech recognition task, however, the method is applicable to any classification data.

Keywords: Pruning synapses · Network simplification
Minimal network structure · Low-cost devices · Speech recognition

1 Introduction

The recent trend of integrating smart electronic devices into a human every-day life calls for new methods for making the software both capable of performing high accuracies and meeting the hardware limitations. This so called "smartness" is often supported by sophisticated machine learning models, being developed on powerful computing machines and usually using a huge amount of data, which makes them robust and recently even surpassing human skills in a variety of cognitive tasks [1,2].

The next step for a practical use, however, is to take the trained models and run them on low-cost devices, where the resources are constrained in terms of computing power and memory size. Out of the wide range of applications we can give an example of a keyword spotting microcontroller - an always-on chip inside today's smartphones [3], where a robust neural network based model works on a hardware, which is limited in order to fit in a phone.

In [4], the authors made effort to meet the resource limitations by investigating and choosing from various network architectures (DNN, CNN, LSTM, ...) and used the Google speech commands dataset [5] for comparison. They compared the performance of different models in terms of memory footprint, number of operations needed for prediction and test accuracy.

A. Karpov et al. (Eds.): SPECOM 2018, LNAI 11096, pp. 58–67, 2018.
https://doi.org/10.1007/978-3-319-99579-3_7

In this work, we take 6 of their network schemes (3 DNN and 3 LSTM) as a baseline and make them learn the same data. Then we put our hypothesis that the number of operations and the memory footprint can be rapidly reduced by removing unimportant parameters (synapses) from fully-connected models, while the classification accuracy of original predictors is not significantly hurt. Hence, our contribution rests in presenting a general algorithm for finding and pruning redundant synapses in both feed-forward and recurrent neural networks.

1.1 Related Work

The problem of network pruning was touched by several researchers in the early 90's of the last century - a good survey of developed pruning methods is given by Reed [6] and a comparison of pruning methods can be found in [7]. Clearly, when trying to remove redundant parts of a neural network, the crucial question is how to distinguish them from the important ones. To briefly enumerate only the most relevant studies touching this problem:

1. *Skeletonization* [8] - a measure called "relevance" was introduced. It is computed as the error when the synapse is removed minus the error when the synapse is left in place.
2. *Optimal Brain Damage* [9] - Yann Lecun and his team presented a measure called "saliency" estimated by the second derivative of the error with respect to the weight.
3. *Karnin's measure* [10] - the author used the change in weight during the training process to compute a measure called "sensitivity" given as:

$$S_k = \sum_{n=0}^{N-1} \left[\Delta w_k(n)\right]^2 \frac{w_k(t_f)}{\eta(w_k(t_f) - w_k(0)} \tag{1}$$

where n runs over training epochs, $w_k(t_f)$ is the value of weight w_k after training and $w_k(0)$ is its initial value, η is a constant. Eq. 1 is shown here on purpose as it is relevant to the investigated measure introduced in Sect. 2.1 of this work.

1.2 Contribution of This Work

The aim of this work is to contribute to the state-of-the-art network minimalization research by introducing a method for a rapid reduction (80–90%) of the number of parameters by removing unimportant synapses from the network. As well as in case of the quantization flow in [4], the classification accuracy does not drop significantly after the intervention.

The reduction of redundant parameters leads to the reduction of the model size as well as the number of operations needed for prediction, which makes the method a perfect tool for designing on-board prediction models.

Unlike the other studies mentioned in the previous section, we come with a simple (in terms of computational demands and processing time) measure for distinguishing important synapses from the redundant ones (Sect. 2.1) and we

also introduce a general network pruning proceeder (Sect. 2.2). Although the performance is shown on the Google speech commands dataset [5] only, the approach is general and applicable on any classification problem.

2 Network Pruning

The rule of thumb in using artificial neural networks for classification nowadays is taking a fully connected structure - each neuron is synaptically connected to all units in the following layer in case of feedforward neural networks and similarly all possible synapses are present in case of recurrent networks. This leads to enormous numbers of parameters for networks with many neurons.

We agree that a fair amount of neurons is needed for a sufficient network performance, however, we believe that the number of parameters can be rapidly reduced by removing single synapses. Here we put the hypothesis that some of the synapses in fully connected (feedforward as well as recurrent) networks do not contribute to the classification at all and so their removal would not cause a significant classification accuracy drop. This idea is graphically illustrated in Fig. 1.

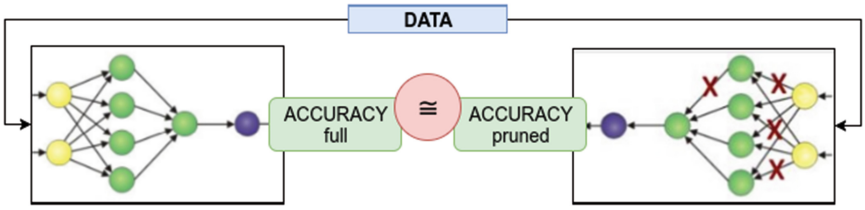

Fig. 1. Hypothesis: Removal of redundant synapses does not influence the performance.

2.1 Determining Synapse Significance

The crucial problem is to identify the redundant synapses in fully connected networks and to distinguish them from the important ones. To face this challenge we introduce a measure called `WSF` - *Weight Significance Factor* (Eq. 2).

$$\texttt{WSF}(w_k) = |w_k(t_f) - w_k(0)| \tag{2}$$

where $w_k(0)$ is the initial value of weight w_k and $w_k(t_f)$ is its value after network training. The idea is that the weight change over network training is related with the classification importance of the corresponding synapse, so that weights of redundant synapses do not significantly evolve during the training. Therefore, synapses with low `WSF` are considered less important than those with high `WSF` after training.

2.2 General Pruning Proceeder

The developed network pruning algorithm is an iterative process that is general in terms of using any of the discussed measures of weight significance [8–10]. The procedure is illustrated in Fig. 2.

First of all, a relevant (big enough in terms of number of layers/neurons) network is chosen and trained to a maximal test accuracy for given data. Next, the initial so-called percentile level (by default `P=75`) must be defined. Once the original network is trained, we call it a *processed network* and iteratively repeat the following steps:

1. Copy the *processed network* and so get the *working copy*
2. Take the *working copy* and remove `P%` of the synapses (the least important ones based on the chosen measure) and so get the *pruned working copy*
3. Retrain the *pruned working copy* with training data up to the best possible validation accuracy
4. Evaluate the *pruned working copy* on testing data and check if the required classification accuracy is kept
   - yes (accuracy kept) → take the *pruned working copy* as *processed network* and go to step 1
   - no (accuracy broken) → go to step 5
2. Check if the current percentile level `P` can be decreased (`P>0`)
   - yes → decrease the percentile level and go to step 1
   - no → pruning finished, take the *processed network* as a result

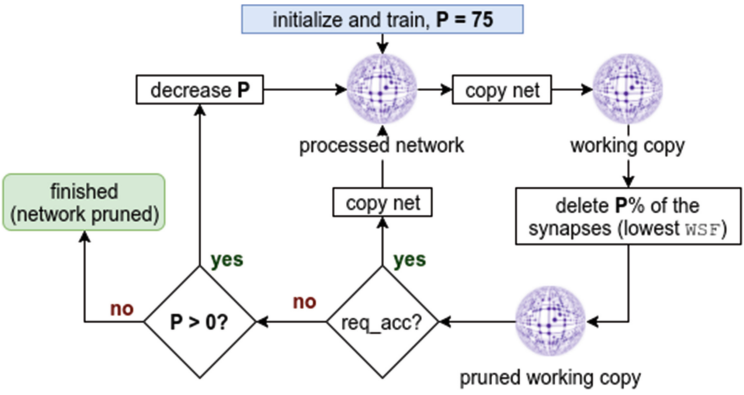

Fig. 2. Network pruning algorithm.

The retraining (step 3) can be skipped to speed up the process, however, in general the network reduction is much more significant when the retraining step is applied. The percentile level is usually being decreased in a predefined manner, by default `75` → `50` → `30` → `20` → `10` → `5` → `1` → `0`. Once the "percentile 0"

is reached, only one synapse, the one with the lowest `WSF`, was removed in the *working copy*. If even a single synapse removal breaks the accuracy, the percentile level is not decreased anymore and the network is considered pruned. In [7] we provide several experiments showing that the derived network has a minimal possible structure for given data in terms of number of synapses.

2.3 Dimensionality Reduction in Feed-Forward Networks

Getting back to the main motivation of this work, the goal is to make a network smaller in terms of a number of parameters, however, the number of operations and the memory footprint during prediction are the overall qualities that make the trained model useful for a target device.

The pruning algorithm described in previous sections is able to reduce the number of parameters by driving unimportant weights to zero. However, even though these parameters equal zero, they are still present and therefore the original dimensions of weight matrices are kept. The next step then is to take advantage of the pruning result by reducing these dimensions in order to decrease the number of operations as well as the memory footprint.

The following approach is applied to weight matrices layer by layer:

1. Remove all zero rows[1] corresponding to neurons with no inputs.
2. Remove the columns (see Footnote 1) in the weight matrix of the following layer corresponding to outputs of the removed neurons in the currently processed layer.
3. Remove all zero columns (see Footnote 1) corresponding to neurons with no outputs.
4. Remove the rows (see Footnote 1) in the weight matrix of the previous layer corresponding to inputs of the removed neurons in the currently processed layer.

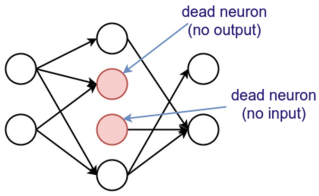

Fig. 3. Illustration of dead neurons in a pruned feedforward network.

Assuming the case in Fig. 3, corresponding dimensionality reduction of the weight matrix (hidden layer) after the removal of dead neurons is shown below.

$$W^0_{pruned} = \begin{bmatrix} w_{11} & 0 \\ w_{21} & w_{22} \\ 0 & 0 \\ w_{41} & w_{42} \end{bmatrix} \rightarrow W^0_{reducted} = \begin{bmatrix} w_{11} & 0 \\ w_{41} & w_{42} \end{bmatrix} \tag{3}$$

[1] Depending on the implementation rows/columns might correspond to layer inputs/outputs or outputs/inputs.

3 Experiments and Results

In this work, we use the Google speech commands dataset [5] and 6 baseline neural network architectures inspired by [4] to demonstrate the ability of the pruning algorithm:

1. to find a rapidly simplified (in terms of number of parameters, number of operations and size on drive) and comparably good classification models;
2. to deal with different network architectures (feedforward, recurrent).

3.1 Data for Demonstration

The dataset [5] consists of 65K samples - one second long audio clips recorded by thousands of different people. There are 30 different words among the samples (see Fig. 4) plus clips representing "silence" - combination of different kinds of noise like doing the dishes, miaowing or an artificially made white noise. We chose 10 keywords - "yes", "no", "up", "down", "left", "right", "on", "off", "stop" and "go" out of the dataset. These keywords alongside with "silence" and "unknown" represent 12 classes for training our models. The "unknown" group consists of the remaining 20 words from the dataset (the transparent ones in Fig. 4) like in [4,11].

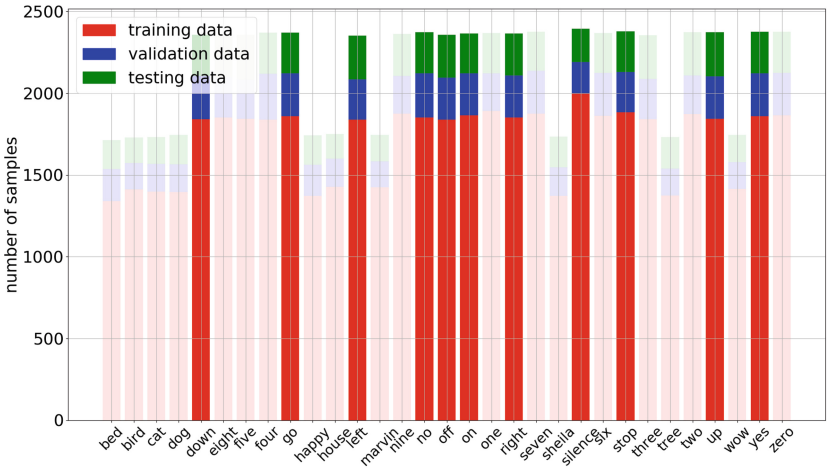

Fig. 4. Distribution of samples in the demonstration dataset.

The provided lists of validation and test samples ensure a controlled dataset split in the ratio of 80:10:10, while words of the same person stays in one set. We do not use any data augmentation. The feature vectors are formed differently for DNN and LSTM models, however, in both cases we use 10 MFCC features out of a window of length 40 ms with a 20 ms shift (settings adopted from [4]).

3.2 Experimental Setup

The experimental setup follows the baseline in [4] and the purpose is to show how the methods introduced in this work can contribute to the overall goal of model minimalization. Out of the wide scale of models presented in [4], we chose two architectures (DNN, Basic LSTM) and designed 3 versions of each differing in number of neurons. The last column in Table 1 (*ops*) stands for the number of operations needed for prediction of one sample (see [4]).

Table 1. Selected model architectures for experiments.

Model	Hidden neurons	# of params	Size on drive	Prediction time	*ops*
dnn_s	FF(144)-FF(144)-FF(144)	113K	468 kB	332 ms	158.8K
dnn_m	FF(256)-FF(256)-FF(256)	258K	1.0 MB	334 ms	397.1K
dnn_l	FF(436)-FF(436)-FF(436)	596K	2.4 MB	336 ms	990.2K
lstm_s	LSTM(118)	62K	261 kB	554 ms	5.9M
lstm_m	LSTM(214)	195K	793 kB	558 ms	18.9M
lstm_l	LSTM(344)	493K	2.0 MB	558 ms	47.9M

We used the Keras API [12] running on top of the TensorFlow [13] backend for training all models. Layers are followed by *tanh* activation and we used the *RMSprop* optimizer with a manually tuned learning rate individually for every model. Then we used the standard *categorical crossentropy* as the loss function and *categorical accuracy* is the observed metric. We fed the networks with samples in batches of size 512 and give them 1000 epochs at maximum for training (early stopping is performed when the validation loss is evidently impaired).

3.3 Training Results

The pruning algorithm takes a trained network as the input. Therefore the first step is to train all the models (from Table 1) up to their best possible performance using the configuration described in the previous section (Table 2).

Table 2. Training results.

Model	Train acc.	Val. acc.	Test acc.	# of epochs	Epoch time
dnn_s	90.5%	82.6%	80.1%	543	1 s
dnn_m	93.3%	82.9%	81.5%	432	1 s
dnn_l	94.2%	83.1%	81.8%	586	1 s
lstm_s	94.8%	89.9%	89.2%	150	13 s
lstm_m	96.5%	90.5%	89.7%	108	14 s
lstm_l	97.9%	91.7%	90.8%	105	15 s

Some of the training results are slightly worse compared to those published in [4] as the training configuration is also a bit different, however, achieving the best training results is not the goal of this work.

3.4 Pruning Results

The approach introduced in Sect. 2 was applied on the six models described in Table 1. We set up 25 retraining epochs (step 3 of the algorithm, Sect. 2.2), maximally 50 pruning iterations and the default sequence of percentile levels.

The pruning result is highly depended on the required classification accuracy we intend to keep. It is a parameter we choose, but naturally it must be less or equal the maximal possible accuracy of the original network.

Table 3. Pruning results. Number of parameters needed to reach required accuracy.

Model	Original		# parameters in pruned nets				
	Acc	# param.	Acc kept	Acc −1%	Acc −2%	Acc −5%	Acc −10%
dnn_s	80.1%	113K	91K	58K	41K	14K	4K
dnn_m	81.5%	258K	237K	154K	89K	23K	4K
dnn_l	81.8%	596K	322K	134K	89K	31K	9K
lstm_s	89.2%	62K	62K	20K	19K	15K	15K
lstm_m	89.7%	195K	181K	40K	37K	26K	32K
lstm_l	90.8%	493K	405K	118K	75K	72K	48K

In Table 3 and in Fig. 5, we can see results (the number of synapses) for various settings of the required-accuracy parameter.

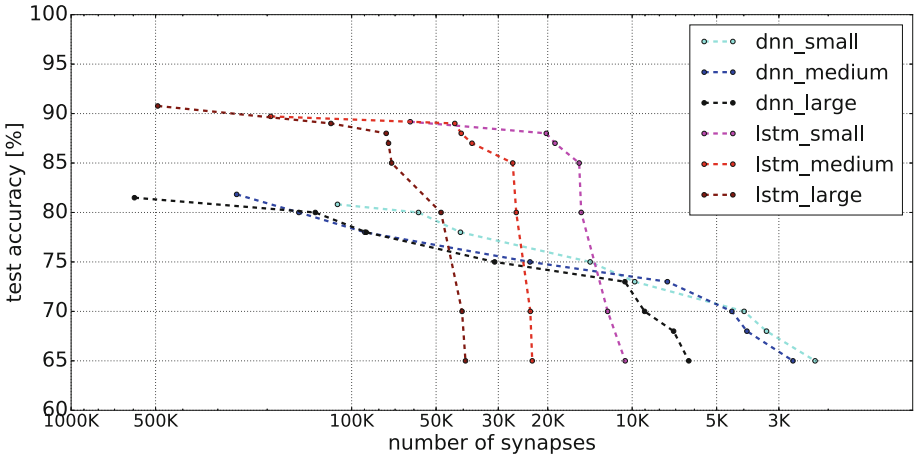

Fig. 5. Actual number of synapses needed to reach desired classification accuracy.

For instance, the `lstm_l` model (originally using 493K parameters with the accuracy of 90.8%) was reduced to 405K parameters, while the accuracy was not broken at all and, as another experiment, was reduced to 118K (24% of the original number) parameters, while the accuracy decreased by 1% to 89.8% only.

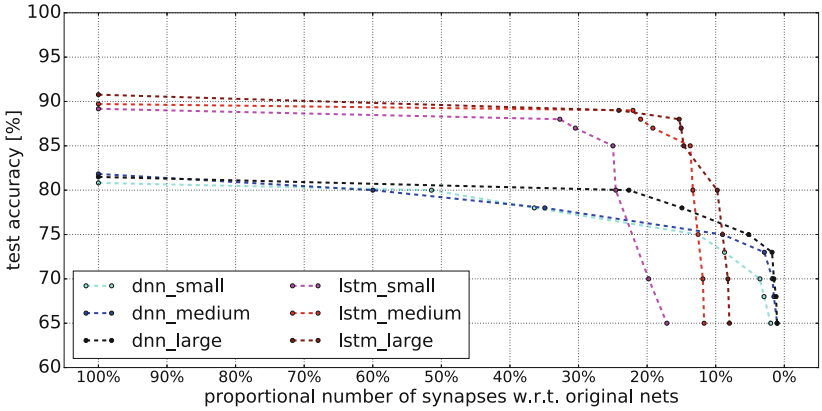

Fig. 6. Proportional number of synapses (with respect to the original network) needed to reach desired classification accuracy.

Figure 6 presents the same results as Fig. 5 did, however, here we have the proportional scale in order to illustrate the immense model reduction more clearly. One can see that the classification accuracy starts to decrease significantly, when the number of synapses is reduced to 30–20% of the original number for LSTM networks and to 10–5% for DNN models.

Figure 7 shows demonstrative DNN and LSTM models in terms of their size when saved on drive and the number of parameters. The goal is to keep them close to the origin in Fig. 7 and performing a high accuracy at the same time.

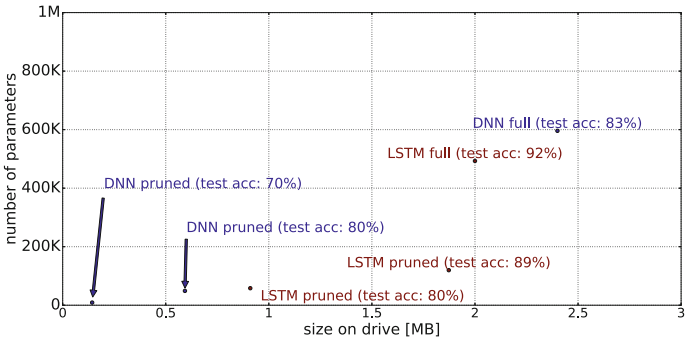

Fig. 7. Model size on drive vs. number of parameters for `dnn_large` and `lstm_large`.

4 Conclusion

The call for neural network based models runnable on low-cost devices for today's practical applications forces us to deal with constrained hardware parameters like limited memory footprint and computing power.

In this work, we introduced a general network pruning algorithm, capable of removing a notable amount of synapses from a trained network (generally 80–90%) that are believed to be unimportant for classification and so the final test accuracy is not significantly hurt. This immense reduction of model parameters leads to a decrease of the model's size and the prediction time.

The results of the pruning procedure are presented on the Google speech commands dataset [5] and the baseline network architectures designed for pruning are adopted from [4]. We showed the capability of the algorithm to deal with feedforward (DNN) and recurrent (LSTM) structures.

The developed methods are implemented in Python and are compatible with Keras [12], which makes it all together a powerful and fast tool for getting a minimized network structure for any classification data.

Acknowledgments. This research was supported by the Ministry of Education, Youth and Sports of the Czech Republic project No. LO1506.

References

1. Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., Feng, J.: Dual path networks. arXiv preprint arXiv:1707.01629 (2017)
2. Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., Stolcke, A.: The Microsoft 2017 conversational speech recognition system. CoRR,abs/1708.06073 (2017)
3. Chen, G., Parada, C., Sainath, T.N.: Query-by-example keyword spotting using long short-term memory networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2015). ISBN 978-1-4673-6997-8
4. Zhang, Y., Suda, N., Lai, L., Chandra, V.: Hello edge: keyword spotting on microcontrollers. arXiv arXiv:1711.07128v3 (2018)
5. Warden, P.: speech commands: a public dataset for single-word speech recognition (2017). http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz
6. Reed, R.: Pruning algorithms - a survey. IEEE Trans. Neural Netw. **4**, 740–747 (1993)
7. Bulín, M.: Optimization of neural network. Master thesis. University of West Bohemia. Univerzitní 8, 30100 Pilsen, Czech Republic (2017)
8. Mozer, M., Smolensky, P.: Skeletonization: a technique for trimming the fat from a network via relevance assessment. University of Colorado, Boulder, Department of Computer Science (1989)
9. LeCun, Y., Denker J.S., Solla, S.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)
10. Karnin, E.D.: A simple procedure for pruning back-propagation trained neural networks. IEEE Trans. Neural Netw. **1**, 239–242 (1990)
11. Kaggle Inc.: TensorFlow speech recognition challenge (2017). https://www.kaggle.com/c/tensorflow-speech-recognition-challenge
12. Chollet, F., et al.: Keras (2015). https://keras.io
13. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). tensorflow.org