Compressing Neural Networks by Applying Frequent Item-Set Mining

Zi-Yi Dou, Shu-Jian Huang(✉), and Yi-Fan Su

National Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210023, China
141242042@smail.nju.edu.cn, huangsj@nju.edu.cn, suyf@nlp.nju.edu.cn

Abstract. Deep neural networks have been widely used contemporarily. To achieve better performance, people tend to build larger and deeper neural networks with millions or even billions of parameters. A natural question to ask is whether we can simplify the architecture of neural networks so that the storage and computational cost are reduced. This paper presented a novel approach to prune neural networks by frequent item-set mining. We propose a way to measure the importance of each item-set and then prune the networks. Compared with existing state-of-the-art pruning algorithms, our proposed algorithm can obtain a higher compression rate in one iteration with almost no loss of accuracy. To prove the effectiveness of our algorithm, we conducted several experiments on various types of neural networks. The results show that we can reduce the complexity of the model dramatically as well as enhance the performance of the model.

Keywords: Neural networks · Frequent item-set mining · Deep learning

1 Introduction

In recent years, neural networks have been proven to be a powerful tool in many fields, including object classification [1] and speech recognition [2]. Typically, people have a tendency to build deeper and larger neural networks. In the field of computer vision, starting with LeNet-5 which requires 431 thousand parameters [3], Krizhevsky *et al.* designed AlexNet with 60 million parameters in 2012 [4] and Sermanet *et al.* won the ImageNet competition using 144 million parameters in 2013 [1]. For natural language processing tasks, a recent state-of-the-art neural machine translation system requires over 200 million parameters [5].

It is true that enormous amount of parameters dramatically improve the performance ,but we should be aware that there is significant redundancy in the parameterization of several deep learning models [6]. Over-parametrization can also lead to problems like over-fitting which can result in low generalization ability. In addition , training and using such large neural networks requires

This work is supported by NSFC No. 61672277, 61472183 and the Collaborative Innovation Center of Novel Software Technology and Industrialization, China.

long running time and costs much energy and memory. The trend of applications of machine learning shifting toward embedded devices, which have limited storage size and computational ability, makes the problem of utilizing so many parameters more severe.

All these issues have motivated the idea of neural network compression, which aims to reduce the storage and energy required to run inference on large neural networks without losing any accuracy. Several methods have been proposed to tackle this problem and here we introduce a rather novel way to compress neural networks based on frequent item-set mining which can be implemented easily and achieve relatively high compression rate in one iteration.

2 Related Work

Network Pruning. Pruning the parameters from a neural network have been investigated for several decades. Basically, many of the traditional algorithms, such as penalty term methods or magnitude based methods, defines a measure of each connection or node and remove the element with the least effect [7].

In recent years, the deep learning renaissance has prompted a re-investigation of network pruning for modern models and tasks [8]. Admittedly, a lot of traditional pruning algorithms can achieve great performance. However, when it comes to large deep neural networks, the high computational complexity cannot be tolerated. In 2015, Han *et al.* proposed an algorithm that can remain efficient in deep neural networks [9].

Inspired by Han *et al.*'s paper, several methods have been put forward to prune the neural networks. DropNeuron [10] adds two more regularization terms into the objective function and thus makes it possible to delete more neurons in neural networks. Network Trimming prunes the neurons with zero activation [11].

To our knowledge, no one has ever applied association rule or frequent item-set generation into pruning neural networks. In addition, only a few papers [8] discuss their applications in recurrent neural networks, which are widely used in areas like sentiment analysis or language models.

Frequent Item-Set Mining. Nowadays there are many sophiscated algorithms for frequent item-set mining. One of the most popular algorithm is Apriori [12]. However, Apriori is inefficient in some scenarios. To resolve the issue, Lin *et al.* use MapReduce to accelerate the algorithm [13]. Cut-Both-Ways (CBW) algorithm first finds all frequent item-sets of a pre-defined length and then employs search in both directions [14]. However, these methods all have exponential complexity and thus cannot be applied in large scale problem.

3 Our Proposed Methods

In this section, we first give a brief overview of frequent item-set mining. Then we illustrate how to apply it in neural networks. Finally we propose our algorithm based on frequent item-set mining.

3.1 Frequent Item-Set Mining Task

Frequent item-set mining is often regarded as the first part of association rule mining. the problem of frequent item-set mining can be defined as [15]:

Let $IT = \{it_1, it_2, ..., it_n\}$ be a set of n binary attributes called *items.* and $D = \{t_1, t_2, ..., t_m\}$ be a set of transactions called the *database.*

Each *transaction* in D has a unique transaction ID and contains a subset of the items in IT. Suppose T is a set of transactions of a given database, then the *support* of an item-set X with respect to T is defined as the proportion of transactions t in the database which contains item-set X, which can be expressed as

$$Supp(X) = \frac{|\{t \in T; X \subset t\}|}{|T|}. \tag{1}$$

The task of frequent item-set mining is to discover all item-sets that satisfy a user-specified minimum support *minsup.*

3.2 Frequent Item-Set Mining in Neural Networks

Symbols and Definitions. In this section, we first introduce some symbols and definitions which will be used in later parts of the paper. Since our strategy prunes the neural networks layer by layer, without loss of generality, the following parts all consider one single layer.

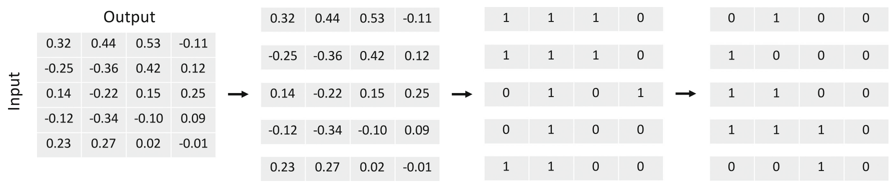
Suppose the layer consists of m input nodes $I = \{i_1, i_2, ..., i_m\}$ and n outputs nodes $O = \{o_1, o_2, ..., o_n\}$, its weight matrix can be represented as a $m \times n$ matrix W. In order to express whether there exists an connection between two nodes, we need an extra connection matrix C, where C is an $m \times n$ boolean matrix and $C_{ab} = 1$ if and only if the output node o_b is connected to the input node i_a.

Therefore, the layer L can be represented as a tuple $< I, O, W, C >$. Normally, for a fully connected layer, the elements of C all equal to one and our mission is to turn as many elements in C into zero as possible without the loss of accuracy.

Item-Sets in Neural Networks. In order to apply frequent item-set mining into pruning neural networks, we must first construct item-sets in neural networks. Based on the definition above, given a layer $L =< I, O, W, C >$, each input node is connected to several output nodes S_i, namely a subset of O. Each element o_i in O can be considered as an item. Thus, we can delete some nodes from S_i and take the rest of the nodes as an item-set. In the end, we can construct m item-sets in total, which is then viewed as our set of transactions T. In our approach, we specify a constant ϵ in advance and delete the nodes whose absolute value of weights of connections to the input node are smaller than ϵ, *i.e.*

$$o_i \in t_j \Leftrightarrow |W_{ij}| > \epsilon, for\ each\ t_j \in T$$

In order to make this point clear, let's consider a fully connected layer whose $m = 5$ and $n = 4$. The weight matrix is shown in Fig. 1.



Input \ Output				
	0.32	0.44	0.53	-0.11
	-0.25	-0.36	0.42	0.12
	0.14	-0.22	0.15	0.25
	-0.12	-0.34	-0.10	0.09
	0.23	0.27	0.02	-0.01

$\rightarrow$

0.32	0.44	0.53	-0.11
-0.25	-0.36	0.42	0.12
0.14	-0.22	0.15	0.25
-0.12	-0.34	-0.10	0.09
0.23	0.27	0.02	-0.01

$\rightarrow$

1	1	1	0
1	1	1	0
0	1	0	1
0	1	0	0
1	1	0	0

$\rightarrow$

0	1	0	0
1	0	0	0
1	1	0	0
1	1	1	0
0	0	1	0

Fig. 1. Three steps to apply frequent item-set mining in neural networks: separate each row of W, view each o_k as an item; construct set of transactions T; applying frequent item-set mining algorithm directly.

We view each output node as an item. First, we consider each row separately and those connections whose weights less than $\epsilon = 0.2$ are removed from the item-sets. In the end, we get five item-sets from this layer.

Frequent Item-Set Mining in Neural Networks. Once we have constructed the item-sets, we can directly use current frequent item-set mining algorithms like Apriori to find out all the item-sets whose support is greater than the pre-defined *minsup*. Then we can select the top m item-sets with the highest support so that the basic architecture of neural networks remain the same.

Again, let us consider the previous example. After applying frequent item-set mining, the result is shown in Fig. 1.

Importance Measure of Item-Sets. The support of one item-set cannot be the only measure. If one item-set is frequent, *i.e.* its support is greater than a constant, then all its subsets are frequent. Therefore, the sets with few elements always have the highest support. As we can see from the figure, three out of five frequent item-sets only consist of one element. To fix the issue, we should measure the importance of an item-set not only in terms of its support. Here we propose an importance measure of an item-set S:

$$Importance(S) = \frac{|\{t \in T; S \subset t\}|}{|T|} + \lambda * e^{|S|/n} \tag{2}$$

Here n denotes the total number of output nodes, T represents the item-sets constructed from neural networks and λ is a pre-defined hyper-parameter. The first term on the right side is the original definition of support and the second term tries to model the effect of the size of one item-set. In this way, we hope the length of item-sets will counteract the influence of support.

3.3 FIMP Algorithm

The naive idea is to first use frequent item-set mining algorithm such as Apriori to calculate the support of each item-set and then re-rank all the remaining item-set according to their importance as defined in Eq. 2. Finally, we select m item-set with the highest importance.

However, we should be aware that this simple idea has some potential drawbacks. First, the computational cost is growing exponentially. Second, the item-sets cannot be repetitive, which means that in extreme cases where $m >> n$, there will be not enough item-sets.

Here we improve our former idea and propose an algorithm that can solve all the drawbacks mentioned above. We call it FIMP algorithm, which is short for Frequent Item-set Mining based Pruning.

In the first step, we construct all the item-sets in every layer as described in Sect. 3.2, corresponding to the third matrix from the left in Fig. 1. Then, for every item-set, we apply greedy search or beam search to calculate the importance of its subsets. Then we choose the subset with largest importance and continue the above procedure until during the search no subset has higher importance than the current item-set. The full procedure is shown in Algorithm 1. S_i is a subset of S with $size(S_i) >= size(S) - k$, where k is a hyper-parameter.

Algorithm 1. FIMP Algorithm

```
Input: Neural Network N, $\epsilon$
Output: Pruned Neural Network PN
1: for each layer with $m_i \times n_i$ nodes do
2:     Construct item-sets following the procedure described in Sect. 3.2
3:     for each item-set $S$ do
4:         while $Importance(S_i) > Importance(S)$ do
5:             $max\_id = argmax_{1 \leq i \leq m_i} Importance(S_i)$
6:             $S = S_{max\_id}$
7:         end while
8:     end for
9: end for
```

4 Experiment

4.1 Baseline Models

We compare our model with two baseline methods: (1) The method proposed by Han *et al.*, where they simply remove the connection with least weight [9]. (2) *DropNeuron* proposed by Pan *et al.* where they add two more regularization terms so that more neurons can be removed [10].

We list the percentage of connections left after pruning for each layer L, represented as $W^L\%$, and the performance of the model before and after pruning.

4.2 Deep Antoencoder

First we conducted experiment on deep autoencoder. We considered the image dataset of MNIST [16]. The number of training examples and test examples are

60000 and 10000 respectively, the image sizes are 28×28 digit images and 10 classes. We used the same setting as [10], where they use $784 \rightarrow 128 \rightarrow 64 \rightarrow 128 \rightarrow 784$ autoencoder and all units were logistic with mean square error as loss.

Table 1. Results of pruning autoencoder

	$W^{FC1}\%$	$W^{FC2}\%$	$W^{FC3}\%$	$W^{FC4}\%$	$W^{total}\%$	NMSE (before)	NMSE (pruned)
Han *et al.*∗	15.18%	46.29%	52.53%	17.54%	18.86%	0.011	0.011
Pan *et al.*∗	16.00%	44.47%	54.11%	18.14%	19.50%	0.012	0.012
FIMP	14.27%	29.00%	39.94%	8.95%	13.34%	0.009	0.007

∗ Means the result is cited from Pan *et al.* [10]

The results can be seen at Table 1. Here we use the standard normalized mean square error (NMSE) metric, *i.e.* $NMSE = \frac{\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{N} y_t^2}$, to evaluate the prediction accuracy of the model. From the table, we can see that our method can achieve greater compression rate compared with [9] and [10].

4.3 Fully Connected Neural Networks

We also implemented our algorithm on fully connected neural networks. Here we mainly focus on the representative neural networks LeNet. LeNet-300-100 is a fully connected network with two hidden layers, with 300 and 100 neurons each, which achieves 1.6% error rate on MNIST. Unfortunately, we could not find any pre-trained model of LeNet-300-100 on TensorFlow and it is hard for us to get 1.6% error rate. So we just use the best model we can find with 98.24% accuracy and prune it. As we can see from Table 2, our compression rate is still higher than the other two pruning strategies. Even though our accuracy is a little bit lower, we should notice that the accuracy is actually higher after pruning, which suggests that the relatively lower accuracy may result from the unsatisfied initial model.

Table 2. Results of pruning LeNet-300-100

	$W^{FC1}\%$	$W^{FC2}\%$	$W^{FC3}\%$	$W^{total}\%$	Accuracy (before)	Accuracy (pruned)
Han *et al.* ∗∗	8%	9%	26%	8%	98.36%	98.41%
Pan *et al.*∗	9.56%	11.16%	54.5%	9.91%	98.13%	98.17%
FIMP	6.19%	19.00%	38.50%	7.76%	98.24%	98.27%

∗ Means the result is cited from Pan *et al.* [10]
∗∗ Means the result is cited from Han *et al.* [9]

Figure 2 shows the sparsity pattern of the first fully connected layer of LeNet-300-100 after pruning. The matrix size is $784 * 300$ and the white regions of the figure indicate non-zero parameters. The figure demonstrates how FIMP affects the network. Since digits are written in the center of image, it is no surprising that the graph is concentrated in the middle and sparse on the left and right.

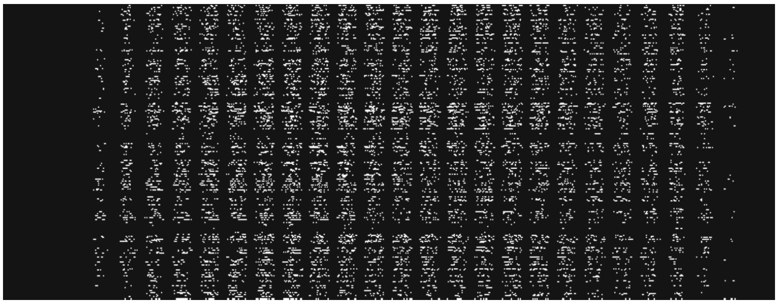


Fig. 2. Visualization of the first FC layer's sparsity pattern of Lenet-300-100.

4.4 Convolutional Neural Networks

LeNet-5 is a convolutional network that has two convolutional layers and two fully connected layers, which achieves 0.8% error rate on MNIST. Actually, our algorithm is the same as the other strategy when pruning convolutional layers, thus we can just compare the performance of three algorithms on the fully connected layers.

Table 3. Results of pruning LeNet-5

	W^{FC1}%	W^{FC2}%	W^{total}%	Accuracy(before)	Accuracy(pruned)
Han *et al.* **	8%	19%	8%	99.20%	99.23%
Pan *et al.**	1.44%	16.82%	1.49%	99.07%	99.14%
FIMP	2.95%	17.42%	3.00%	99.05%	99.12%

* Means the result is cited from Pan *et al.* [10]
** Means the result is cited from Han *et al.* [9]

The results are shown in Table 3. As we can see from the table, the second algorithm did quite well in this task, however, our algorithm can still obtain a similar result which is far better than the first algorithm.

4.5 Recurrent Neural Networks

In this experiment we turn our attention to a recurrent neural network and use it on a challenging task of language modeling. The goal is to fit a probabilistic model which assigns probabilities to sentences. It does so by predicting next words in a text given a history of previous words. We use the Penn Tree Bank (PTB) dataset and perplexity, a common way of evaluating language models, to evaluate the performance of models. We use LSTM model with 2 layers and the hidden size is set to 200 (Table 4).

Table 4. Results of pruning LSTM on language model

	$W^{total}\%$	Perplexity(before)	Perplexity(pruned)
Han *et al.*	10.21%	115.910	109.032
Pan *et al.*	11.37%	115.910	109.724
FIMP	9.33%	115.910	108.999

As we can see from the table, the perplexity after pruning is clearly lower than the original one, which indicates better performance of the model.

5 Discussion and Conclusion

In this paper we present a novel way to prune neural network motivated by the intuition that frequent pattern should be more important. We hope this method could provide the reader with a new perspective toward pruning neural networks. Although FIMP has shown promising results, there are still some unfinished work. For example, we could add constraints that favor the emergence of repeating connectivity patterns so that higher compression rate could be achieved. Also, we could try different measure of importance. Right now larger λ in Eq. 2 means more connections would be pruned and more time would be cost. In this work we set λ to a small value like $1e-5$ and it now takes about half an hour to prune a fully connected layer on a PC while the other two algorithms only cost a few seconds. Although compared with training a large neural network this amount of time is negligible, we may still find some way to speed up FIMP.

References

1. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv: 1312.6229 (2013)
2. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)

3. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989)
4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
5. Luong, M.T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025 (2015)
6. Denil, M., Shakibi, B., Dinh, L., de Freitas, N.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems, pp. 2148–2156 (2013)
7. Augasta, M.G., Kathirvalavakumar, T.: Pruning algorithms of neural networks—a comparative study. Cent. Eur. J. Comput. Sci. **3**(3), 105–115 (2013)
8. See, A., Luong, M.T., Manning, C.D.: Compression of Neural Machine Translation Models via Pruning. arXiv preprint arXiv:1606.09274 (2016)
9. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
10. Pan, W., Dong, H., Guo, Y.: DropNeuron : Simplifying the Structure of Deep Neural Networks. arXiv preprint arXiv:1606.07326 (2016)
11. Hu, H., Peng, R., Tai, Y.W., Tang, C.K., Trimming, N.: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. arXiv preprint arXiv:1607.03250 (2016)
12. Borgelt, C.: Frequent item set mining. Wiley Interdisc. Rev.: Data Min. Knowl. Discov. **2**(6), 437–456 (2012)
13. Lin, M.Y., Lee, P.Y., Hsueh, S.C.: Apriori-based frequent itemset mining algorithms on MapReduce. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, p. 76. ACM (2012)
14. Su, J.H., Lin, W.: CBW: an efficient algorithm for frequent itemset mining. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004, p. 9. IEEE (2004)
15. Agrawal, R., Imieliski, T., Swami, A.: Mining association rules between sets of items in large databases. In: ACM SIGMOD Record, vol. 22, no. 2, pp. 207–216. ACM (1993)
16. Lecun, Y., Cortes, C.: The mnist database of handwritten digits (2010)