Pruning with Meta-Gradient Descent for Mobile Devices

Zheyu Shi and Qingshan Yao
Shanghai Jiao Tong University

1. INTRODUCTION

In the society where mobile devices are widely used, it is a hot issue how to transfer the theory and technology in the field of computer to mobile devices efficiently.Among these problems, deep network is undoubtedly one of the most popular ones.However,it is very difficult to train or run a neural network on a mobile device due to computational limitations.This makes tasks such as intelligent speech recognition possible only when connected to a network.In this case, it becomes an important task to get a small but efficient model that can be adjusted according to the characteristics of different users.

At present, there are few and single solutions to this problem, most of which are few shot learning works or pruning for large networks. Obviously, it is impossible to solve the above problems by separating these two parts. Therefore, we hope to combine the two directions to solve the problems, so as to get a network that meets our requirements.

In this paper, we propose a method that pruning the deep network with meta-gradient descent to get a small but well adapted network for few shot learning tasks. The idea of the meta-learning part comes from [Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, 2017] by (Chelsea Finn, Pieter Abbeel, Sergey Levine). For the pruning part, we tried different methods including filter pruning and weight pruning. Finally, we propose our own pruning strategy for this task–weight and gradient based weight pruning.

2. BACKGROUND & RELATED WORK

In practical application, we cannot get enough training samples in many cases, so few shot learning is a very important task.At present, the most widely used method is to obtain a certain amount of annotated data and then fine tune it based on a basic network. This method is relatively simple .The second method is based on metric.This method is to model the distance distribution between samples so that the homogeneous samples are close while the heterogeneous samples are far away.Some jobs under this method include the [Siamese neural networks for one-shot image recognition, 2015] proposed by (G Koch, R Zemel, and R Salakhutdinov), [Matching networks for one shot learning, 2016] by (Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al.) and [Prototypical networks for few-shot learning, 2017] by (Jake Snell, Kevin Swersky, and Richard S Zemel). By learning an end-to-end nearest neighbor classifier, it benefits from the advantages of both with and without parameters, so that it can not only quickly learn new samples, but also has good generalization for known samples.

Another direction is meta-learning. We expect the system to acquire multiple skills and adapt to a variety of environments, but training for each skill from zero is not sustainable.Therefore, we want it to be able to learn new skills quickly from previous experience, rather than thinking about new tasks in isolation.This approach, known as learning to learn, or meta-learning, enables our system to continuously learn a variety of tasks throughout its life cycle.

When meta-learning is applied for few shot tasks, there are three branches. The first one is a memory-augmented neural network proposed in 2016. The second one is the method in [Optimization as a model for few-shot learning, 2017] by (Ravi, Sachin and Larochelle, Hugo). The method we use as a part of our work is a model-agnostic method mentioned above.

The method makes it possible to obtain better generalization performance on a small number of samples with a few iterative steps, and the model is easily fine-tine.Moreover, this method does not need to be concerned about the form of the model or add new parameters to meta learning, but directly useS gradient descent to train learner.The core idea of this method is that the initialization parameters of the learning model maximize the accuracy of the new task after one or several iterations.It is not about updating functions or rules for model parameters, and it is not limited to the size of the parameters and the architecture of the model (as in RNN or Siamese).It is also essentially a good feature of learning that makes it suitable for many tasks (including categorization, regression, and reinforcement learning) and for good results by fine-tune.

For the pruning method, there are all kinds of method to choose. We tried the method using Taylor expansion proposed in [Pruning Convolutional Neural Networks For Resource Efficient Inference, 2017] and many other normal methods. But they did not work well. And we propose a new weight pruning method which makes a second choice based on both weight and gradient. This method not only adapts to our unique network but also gets good results obtained under high pruning rate.

3. MOTIVATION

With the development of mobile devices' performance, more and more AI technology have come to our daily life. Techniques like face recognition, voice recognition provide us with much convenience and security. And thanks to the promotion in the number and the functionality of sensors on mobile devices, rather than train and run a model by itself, mobile devices can simply call the API and complete the invoking of a AI function which is large and complex and can only run a computer with high performance.

But this means many AI functions highly depend on mobile internet's quality, and even some ordinary functions such as voice recognition will malfunction when there is no internet access. This really matters in environments of which the internet access quality is bad, as well as those devices designed to work without internet. So under this situation, the model should run locally on mobile devices.

What's worse, mobile device user can only provide very limited training data. Data that users can provide for training are at most some sentences they speak, or some labeled they attached when they have the mood to some labeling. We can't count on users to produce much data, so the model must be adjustable by only a few training data.

Since mobile devices usually have poor computing power, how can we run a network model locally on mobile devices when the Internet is not accessible? That brings the challenge of small-scale yet few-shot-adjustable model suitable for mobile devices.

On one hand, models on mobile devices for daily use can be scaled down. It is because mobile devices usually highly relate to one or a couple of users. Different from general purpose model which must be able to serve any individual, a model on mobile devices only need to serve certain users, so the information in the model's structure can be far less than the complete network. On the hand, few-shot adjustability is acquirable as well. This is related to an area called few-shot learning.

Therefore, we want to obtain a NN model that can locally run on mobile devices. It should have these properties:

1. Small enough, so it can run on mobile devices.
2. No Internet dependency.
3. Only need to serve certain users.
4. With the ability to quickly self-adjust for those certain users.

4. PROBLEM FORMULATION

4.1 few-shot learning task

We want to introduce few-shot learning here. As the name implies, few-shot learning refers to the process of feeding a learning model with a very small amount of training data, contrary to the normal practice of using a large amount of data. AI systems can learn some sophisticated skill from a huge amount of data from scratch. But sometimes we hope it can learn quickly from previous experience rather than consider each new task separately. So it is a process of "learning to learn". When the training data of the new tasks are very deficient–sometimes only five or less images– it is called few-shot learning.

Here is a notion of a few-shot learning problem. Each task T_i is defined by inputs x_t , outputs a_t, loss function $L_i(x_t, a_t)$, an initial distribution $q(x_1)$, a transition distribution $P_i(x_t|x_{t-1}, a_{t-1})$ and an episode length H_i. Given a distribution over tasks $T = P(T_i)$, the objective is to minimize the loss below:

$$min_\theta E_{T_i \sim T}\Big[\sum_{t=0}^{H_i} L_i(x_t, a_t)\Big] \tag{1}$$

More specifically, the most concerned problem in few-shot learning is N-way K-shot problem. The problem requires the model to be flexible and adjustable by only a few labeled data. In each epoch, the training data consists of a support set and a query set. The support set contains N classes, each having K examples, like images or audios etc. And the query set contains one example randomly selected from those N classes. As the name implies, the model should learn from the support set, and then use knowledge including both its long-term memory and lately learned information, to infer the label of the example in the query set.

Training a model for certain users can be considered as a few-shot learning task in fact. Taking voice recognition for example, since the voice of each individual vary, the recognition process for each individual can be seen as a independent task. These tasks are similar to each other (everyone says "hi" almost in the same way) yet with differences (the voice and the pronunciation habit may be very different among all people). For each user (i.e. each task) , training data that the user provides are the support set. After learning from them, the model is supposed to master the work in this task, that is, the work in regards to this user.

If we can achieve this, the few-shot adjustability is satisfied.

4.2 Pruning

The model needs to run on mobile devices, so its scale must be small. However, after looking through papers about few-shot learning, most of them don't care about the net structure's scale and the computation efficiency.

After considering several approaches to make the model small, we decided to using pruning technique the network to scale it down. We plan to firstly train a model without considering the scale or the number of neurons, then prune it to a small scale and try to keep its accuracy in the same time.

So we must find a way to correctly and efficiently prune a few-shot NN model. It is quite different from the pruning of usual NN, for the inner connections of neurons in a few-shot learning model can be quite different from those in a usual network model and we must consider both the common knowledge aspect and the adjustability aspect. More details will be discussed in next section.

5. PROPOSED METHODS

As we stated above, we are going to solve the problem in two part. Since the final method is the combination of these two methods, first we are going to introduce the meta-learning proposed in [Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, 2017] by (Chelsea Finn, Pieter Abbeel, Sergey Levine).

5.1 Few-shot learning method

For few-shot learning problem, we used meta-leaning method MAML. MAML is a meta-learning model for few-shot tasks.

Similar to other meta-learning model, MAML is trained by randomly selecting a broad range of tasks, and it can adjust it self to a new task through only a few steps or even one step of gradient update. In this process, the meta-learner will seek for parameters that can not only be adjusted to different tasks, but also learn and adjust fast(only few steps of gradient descent) and efficiently(using only few examples).

In the training process, given a set of $T_i \sim P(T)$, the model will first utilize the support set to update for one step,

$$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta) \tag{2}$$

then compute the loss, which is defined to be the loss of the query set, that is

$$L_{T_i} = L_{T_i}(f_{\theta_i'}) \tag{3}$$

This implies that we don't care about the loss "before learning from support set", instead we only care about the loss "after leaning". If the losses become low enough after epochs of training, it means the model has acquired the ability of fast learning.

So we are going to optimize parameters according to the meta-loss, and the update method is simply SGD or Adam, just like a general optimization process.

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim P(T)} L_{T_i}(f_{\theta_i'}) \tag{4}$$

In the graph above, the broad line represents the meta-learning process, and MAML use these randomly selected tasks to optimize the gradient descent process(represented by the dashed), and the parameters after optimization will be closer to those optimal parameter sets.

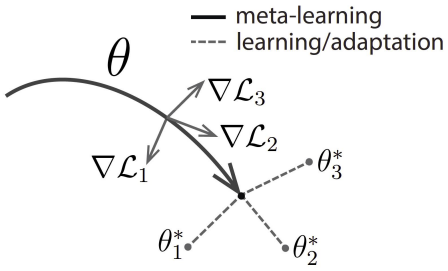

Fig. 1. MAML learning process

One reason that we choose this method is that the method is model-agnostic, which means we can do whatever on the structure of the network, such as pruning by neurons, by filters, even by layers is allowed. Thus when designing the pruning method, we don't need to consider the consequence of changing the network structure.

Another reason is that the method only needs one or a few epochs of updating based on the support set, and in real application scenarios there isn't much time for the model to update or fine-tune. This method can use that small number of training data fast and efficiently.

5.2 Pruning method

The procedures of our pruning method is presented in the graph below.

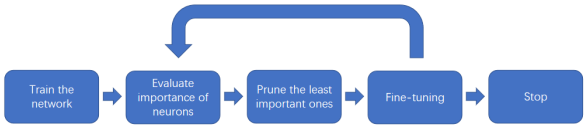

Fig. 2. Pruning procedure

Its similar to most network pruning methods, but the evaluation process is some what different and difficult.

It is quite hard to design a good evaluating method for a few-shot learning network. As we stated before, the model will first do some quick learning from the support set then perform the prediction. That means the parameters are not static even within each training epoch. Thus, there are some extra features of neurons, such as the sum of gradients' absolute differences from before learning to after learning. A good evaluation criterion should take these high-level features into consideration.

We have tried some trivial evaluating methods, however the performances of them are not satisfying. Those methods include evaluating by raw weights of neurons, which is data-free but the accuracy after finetuning drops at a little rate; as well as evaluating by the production sum of meta-gradients and weights, which intends to prune those neurons of which the pruning affect the loss most slightly.

In regard to the particularity of the model, we found that two features of neurons play an important role. One is the weight of neurons, the other is the difference of weight between and after learning in every task, i.e. the first-order gradient of the learner.

$$-\alpha\nabla_\theta L_{T_i}(f_\theta) \tag{5}$$

irst, neurons that support adjustability should not be pruned. If a neuron always change greatly after updating, it means that this neuron is somewhat adjustable between different tasks. It is these kind of neurons that render the model with the ability of fast learning. So the gradient must be small.

Second, neurons with large weight value should not be pruned. If we set those very small number to zero, it wont effect the model too much. However the pruning of some large weight will result in the sharp decreasing of the accuracy, and can not recover even with two times the epochs of finetuning. So the weight must be small.

Taking two aspects into consideration, we wish the pruned neurons are low in both features. Therefore we set the importance of one neuron to the production of its weight value and the sum of absolute value of its first-order gradient on the whole training set. Besides that, to avoid pruning the structure to a extremely uneven shape and to control the pruning ratio of each layer separately, we add a L_1 regularization item to the criterion. The importance of a neuron is

$$Imp(w_l^k) = w_l^k \cdot \sum_{T_i \sim P(T_i)} \left(\frac{\delta L_{T_i}(f_\theta)}{\delta w_l^k}\right)^2 \tag{6}$$

$$\hat{Imp}(w_l^k) = \frac{Imp(w_l^k)}{\sum_j |Imp(w_l^j)|} \tag{7}$$

, where w_l^k is the k-th neuron in layer i.

6. EXPERIMENTS

6.1 Dataset & Train the model

To implement and evaluate our method, we use the Omniglot as the dataset which contains handing writing of 1700+ characters from 50 languages.

We sample N classes from the dataset and K examples of each class. We then feed the corresponding NK example-label pairs to the network in a random order, followed by a new, unlabeled example from one of the N classes.

Because there is no suitable server, training a large network and need a certain amount of computing power, we chose to use the kernel in Kaggle to train the network. After 20000 epochs of training, the accuracy on test set reaches 94% and then we are going to prune this network with different methods.

6.2 Experiment one: Talor expansion based filter pruning

In this part, we use the product of output of a filter and its gradient to evaluate the importance of a filter(mentioned above in [Pruning Convolutional Neural Networks For Resource Efficient Inference, 2017])

Let h_i be the output produced from parameter i. In the case of feature maps, $h = \left\{z_0^{(1)}, z_0^{(2)}, \ldots, z_L^{(C_\ell)}\right\}$.For notational convenience, we consider the cost function equally dependent on parameters and outputs computed from parameters: $\mathcal{C}(\mathcal{D}|h_i) = \mathcal{C}(\mathcal{D}|(\mathbf{w}, b)_i)$. Assuming independence of parameters, we have:

$$|\Delta\mathcal{C}(h_i)| = |\mathcal{C}(\mathcal{D}, h_i = 0) - \mathcal{C}(\mathcal{D}, h_i)| \tag{8}$$

where $\mathcal{C}\left(\mathcal{D}, h_i = 0\right)$ is a cost value if output h_i is pruned, while $\mathcal{C}\left(h_i\right)$ is the cost if it is not pruned.

Use first-degree Taylor polynomial to approximate:

$$\mathcal{C}\left(\mathcal{D}, h_i = 0\right) = \mathcal{C}\left(\mathcal{D}, h_i\right) - \frac{\delta \mathcal{C}}{\delta h_i} h_i + R_1\left(h_i = 0\right) \tag{9}$$

Then by combining equation (8) and (9) and ignoring the reminder, we can get:

$$\Theta_{TE}\left(h_i\right) = \left|\Delta \mathcal{C}\left(h_i\right)\right| = \left|\frac{\delta \mathcal{C}}{\delta h_i} h_i\right| \tag{10}$$

Intuitively, this criterion prunes parameters that have an almost flat gradient of the cost function feature map h_i. This approach requires accumulation of the product of the activation and the gradient of the cost function to the activation, which is easily computed from the same computations for back-propagation. Θ_{TE} is computed for a multi-variate output, such as a feature map, by

$$\Theta_{TE}\left(z_l^{(k)}\right) = \left|\frac{1}{M} \sum_m \frac{\delta C}{\delta z_{l,m}^{(k)}} z_{l,m}^{(k)}\right| \tag{11}$$

where M is length of vectorized feature map. For a minibatch with $T > 1$ examples, the criterion is computed for each example separately and averaged over T.

In order to calculate this value after backward propogation, we use the specific mechanism in PyTorch called hook to catch the gradient we need and finish the calculation.By choosing some filters with the smallest weights, we reconstruct the network to achieve the purpose of pruning. In this way, we can control the number of filters each time to prune, which means we can kind of control the pruning speed.

The total number of filters in this network is 256. We set the number to prune in each epoch as 8. After each epoch of pruning, we fine-tune the network for 200 epochs. After 16 epochs of pruning, we get the result as follows:

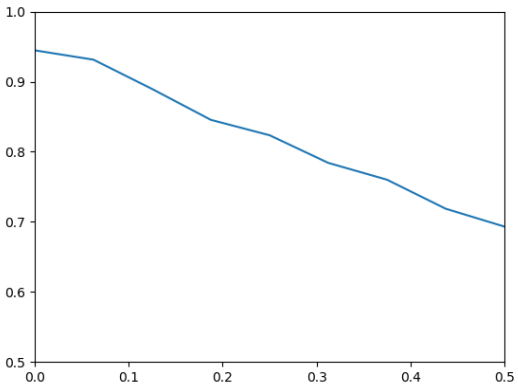

Fig. 3. Pruning result(bad one)

Obviously, the performance of the system is seriously reduced in this way, and the accuracy rate is reduced by 20% in the case of 50% pruning, which is undoubtedly unacceptable.We then analyzed this and tried to reduce the number of cuttings per round and increase the number of adjustments, but the result was still not very different from the figure above.

In our view, this kind of pruning is not suitable for this task because its pruning granularity is too coarse. In a sense, only the gradient is considered when calculating the weight, and the gradient of the feature map is used, which makes a large number of nodes important to the few shot learning task be subtracted together with the filter.

6.3 Experiment two: Pruning with our method

This method is a weight pruning method.At the very start, we want to regard gradient more important than weight.Therefore, we firstly choose some node with small gradient and then choose nodes with small weight from them.

Gradient represented the activity level of a node, while weight represented its influence on the current result. The first thing we should cut out is the inactive node, and the second is the less influential one.

In MAML's model, multiple derivatives are required to guide the learning, but in order to calculate the weight, we only need the first derivative, so we can't directly use the training function in that model.

After solving this problem relatively easily, we found that the performance of the system increased rather than decreased after a large number of pruning, which made us suspicious. After checking the network, we found that some weights would become non-zero in the subsequent fine-tune process after weight was set to zero.In fact, this question is similar to the last one.Because when we take the multiple derivatives in MAML based fine-tune process, even if we start with a weight of zero, after multiple back propagation, it will probably have a value.So we have to set it to zero before every back propagation to make sure that this node is prunned.

After solving these small problems, the pruning was carried out smoothly, and the performance of the system was not decreased obviously.So we started trying new ideas.The previous idea was that gradient is more important. But we think both of them contributes a lot to the choice of pruning.

Therefore, we examined the weight and gradient distributions of all nodes and obtained the following results(the gradient has been multiplied by 100 for convenience):

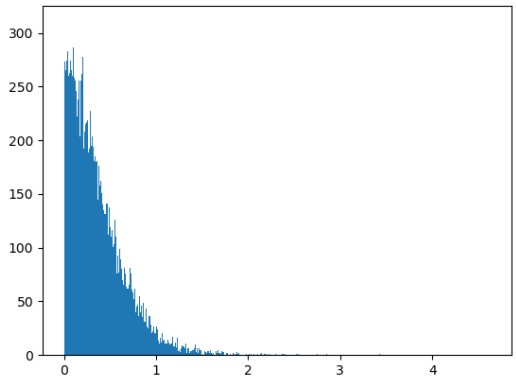

Fig. 4. Weight Distribution

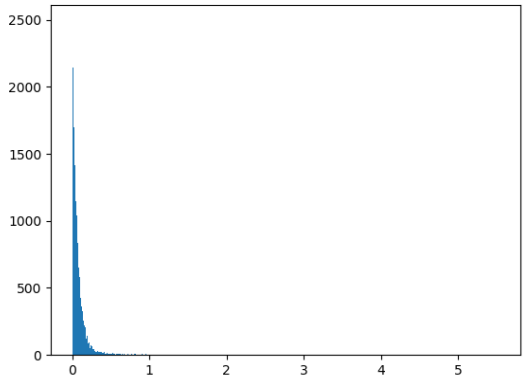

Fig. 5. Gradient Distribution

Obviously, the gradient distribution is more concentrated and the weight distribution is more discrete, But there are a lot of nodes in smaller parts for both of them,which means that we should try to use the multiplication to calculate the weight value for each node, so that the pruning effect will be better.

To verify our approach, we compared the effects of three weight calculation methods: gradient first, weight first and multiplication of them(equally important).There are more than 90,000 nodes in the entire network, and we cut off a thousand at a time. This is the result after 50 epochs of pruning.

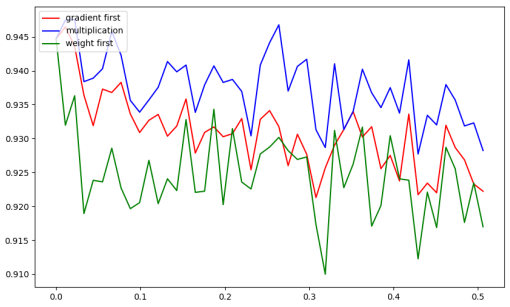

Fig. 6. Comparison of the three methods

The experimental results show that the proposed pruning scheme is feasible and the decrease of accuracy is small. And using product of weight and gradient is the best of the three choices.

7. CONCLUSION

We through the combination of meta-learning and network pruning designed a kind of method that can get a network can run on mobile devices, we not only improved with prior work , also design a new solution based on the problems in the experimental process and the unique attributes of the task, finally achieved A good result: the model compression and basic guarantee performance do not drop as much as possible.

This combination is groundbreaking.It effectively takes a network small enough to run on a mobile device. At the same time, the accuracy it shows in the few shot learning task offers it the ability to serve different users. For example, we can train a huge network which is universal for any speech recognition. Then we prune the network with a few voice messages from a user. Finally, we can get a small but targeted network that can run on this user's mobile phone.

In addition to the innovation of methods, we also benefited a lot from the knowledge we learned and the cutting-edge research achievements we learned in the whole process of the work.Discovering and solving problems has cultivated our abilities in all aspects.Thank you very much for your guidance and help. We think this is a very successful course design!

8. REFERENCES

[1]Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400, 2017.

[2]Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz.(NIVIDIA) Pruning Convolutional Neural Networks For Resource Efficient Inference. arXiv:1611.06440v2, 2017.

[3]G Koch, R Zemel, and R Salakhutdinov. Siamese neural networks for one-shot image recognition. In ICML Deep Learning workshop, 2015.

[4]Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In Advances in Neural Information Processing Systems, pages 36303638, 2016.

[5]Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. arXiv preprint arXiv:1703.05175, 2017.

[6]Santoro, Adam, Bartunov, Sergey, Botvinick, Matthew, Wierstra, Daan, and Lillicrap, Timothy. Meta-learning with memory-augmented neural networks. In International Conference on Machine Learning (ICML), 2016.

[7]Ravi, Sachin and Larochelle, Hugo. Optimization as a model for few-shot learning. In International Conference on Learning Representations (ICLR), 2017.