CS303 Artifical Intelligence

Report for NCS Project

Hu yubin, 11712121[1]

[1]Department of Computer Science and Engineering, Southern University of Science and Technology

October 30, 2019

Abstract

NCS is a population-based search algorithm (like genetic algorithm) for continuous optimization problems. The difference is that NCS uses the idea of negatively correlated search to make individuals search different regions in the decision space. This project is to find the best parameter values (i.e. parameter tuning) as you can for NCS on two benchmark test functions (F6 and F12) and one application problem (OLMP).

Keywords
NCS, Population-based Search

1 Algorithm description

1.1 Main Idea of NCS

Negatively correlated search (NCS) maintains multiple individual search processes in parallel and models the search behaviors of individual search processes as probability distributions[1]. NCS explicitly promotes negatively correlated search behaviors by encouraging differences among the probability distributions (search behaviors). By this means, individual search processes share information and cooperate with each other to search diverse regions of a search space, which makes NCS a promising method for nonconvex optimization. The co-operation scheme of NCS could also be regarded as a novel diversity preservation scheme that, different from other existing schemes, directly promotes diversity at the level of search behaviors rather than merely trying to maintain diversity among candidate solutions. Empirical studies showed that NCS is competitive to well-established search methods in the sense that NCS achieved the best overall performance on 20 multimodal (nonconvex) continuous optimization problems. The advantages of NCS over state-of-the-art approaches are also demonstrated with a case study on the synthesis of unequally spaced linear antenna arrays.

In other words, NCS aims to control each search agent to search the part of the search space that other search agents will not search.

- NCS first calculates the negative correlation (i.e. search behavior diversity) between different search agents
- Then, selects the search agents that have higher negative correlation with other search agents.

We don't want two agents search the similar region like the left chart in Figure 1, so we wish distributions have the large distance like the right chart in Figure 1.

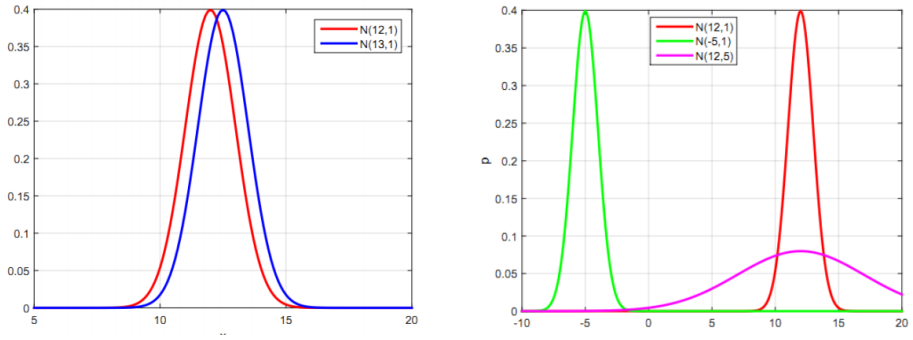

Figure 1: Distance between distributions

There are many metrics for measuring the distance between distributions, NCS adopts the Bhattacharyya distance.

- For continuous case: $D_B(p_i, P_j) = -\ln(\int \sqrt{p_i(x)p_j(x)}dx)$
- For discrete case: $D_B(p_i, P_j) = -\ln(\sum_{x \in X} \sqrt{p_i(x)p_j(x)})$

In case of $N(N > 2)$ agents, the distance of a distribution p_i to the other distributions is measured as: $Corr(p_i) = minD_B(p_i, p_j)|j \neq i$

Not only search behavior diversity (i.e. distribution distance), should also care about whether the search agent can find a better solution. We want to find the solution with both high fitness and large distance. The following heuristic is adopted to control the trade-off between two criteria:
where >0 is a parameter $\begin{cases} \mathrm{x}_i, & \text{if } \frac{f(x_i')}{Corr(p_i')} < \lambda \\ \mathrm{x}_i', & \text{if } otherwise \end{cases}$

Now, we discuss a simple instantiation of NCS, namely NCS-C is implemented to demonstrate the effectiveness of NCS on continuous multimodal optimization problems. We generate the x'_{id} by x_{id} with Gaussian mutation operator.

$$x'_{id} = x_{id} + \mathcal{N}(0, \sigma_i)$$

To keep the algorithm simple, all RLSs in NCS-C are initialized with the same value of i . Then, each i is adapted for every epoch iterations according to the 1/5 successful rule suggested[2]

$$\sigma_i = \begin{cases} \sigma_i \frac{}{r}, & \text{if } \frac{x}{epoch} > 0.2 \\ \sigma_i * r, & \text{if } \frac{x}{epoch} < 0.2 \\ \sigma_i, & \text{if } \frac{x}{epoch} = 0.2 \end{cases}$$

where r is a parameter that is suggested to be set beneath 1, and c is the times that a replacement happens (i.e., x'_i is preserved) during the past epoch iterations. Equation is designed based on the following intuition. A large c implies that the RLS frequently found better solutions in the past iterations, and the current best solution might be close to the global optimum. Thus, the search step-size should be reduced (by r times). On the other hand, if a RLS frequently failed to achieve a better solution in the past iterations, it might have been stuck in a local optimum. In this case, the search step-size will be increased (by r times) to help the RLS explore other promising regions in the search space.

The above is the main idea of the NCS algorithm, the following is the pseudo code of this algorithm.

Algorithm 1. NCS-C (T_{max}, $\boldsymbol{\sigma}$, r, $epoch$, N)

Randomly generate an initial population of N solutions.
Evaluate the N solutions with respect to the objective function f.
Identify the best solution $\mathbf{x}^*$ in the initial population and store it in *BestFound*.
Set $t \leftarrow 0$
While ($t < T_{max}$) **do**
  Set $\lambda_t \leftarrow \mathcal{N}(1, 0.1 - 0.1 * \frac{t}{T_{max}})$.
  For $i = 1$ **to** N
    Generate a new solution $\mathbf{x}'_i$ by applying Gaussian mutation operator with σ_i to $\mathbf{x}_i$.
    Compute $f(\mathbf{x}'_i)$, $Corr(p_i)$ and $Corr(p'_i)$.
  EndFor
  For $i = 1$ **to** N
    If $f(\mathbf{x}'_i) < f(\mathbf{x}^*)$
      Update *BestFound* with $\mathbf{x}'_i$.
    EndIf
    If $\frac{f(\mathbf{x}'_i)}{Corr(p'_i)} < \lambda_t$
      Update $\mathbf{x}_i$ with $\mathbf{x}'_i$.
    EndIf
  EndFor
  $t \leftarrow t + 1$
  If mod(t, $epoch$) = 0
    For $i = 1$ **to** N
      Update σ_i for each RLS according to the 1/5 successful rule.
    EndFor
  EndIf
EndWhile
Output *BestFound*

Figure 2: Pseudo code of NCS alorithm

1.2 Applications of NCS

The application of a communication system often requires solving a challenging optimization problem. For example, minimizing the Symbol-Error-Rate (SER) for Amplify-and-Forward Relaying Systems is nontrivial since the SER surface is non-convex and has multiple minima[3]. When tuning the protocol of sensornets[4], as mathematical modeling of sensornets involves numerous inherent difficulties, one might has to deal with the optimization problem without an explicit objective function and the quality of candidate protocol configurations could only be obtained from a simulation model. In the era of Big Data, as data analytics (e.g., machine learning) fast grows into a ubiquitous technology in many areas, including communications, the challenges brought by complex optimization problems become even more important than ever.

First, one of the major roles of big data analytics is to acquire knowledge from data in order to facilitate decision-making, e.g., managing network resources based on the analysis of user profiles to achieve higher endusers satisfaction. Thus, the value of big data usually needs to be created through tackling an optimization problem (i.e., to seek the optimal decision), which is formulated based on the knowledge obtained from big data analytics. Such optimization problems may not only be non-convex, but also be noisy due to the noise contained in the original data. Furthermore, the data analytics process may also involve complex optimization problems, such as training deep neural networks or tuning the hyper-parameter of Support Vector Machines. To cope with these complex optimization problems, Evolutionary Algorithms (EAs) have been widely adopted and been shown to be a family of powerful tools. In short, to create business values from big data analytics, optimization tools are indispensable.

1.3 Main Idea of OLMP

Layer-wise magnitude-based pruning (LMP) is a very popular method for deep neural network (DNN) compression. However, tuning the layerspecific thresholds is a difficult task, since the space of threshold candidates is exponentially large and the evaluation is very expensive. Previous methods are mainly by hand and require expertise. In this paper, we propose an automatic tuning approach based on optimization, named OLMP. The idea is to transform the threshold tuning problem into a constrained optimization problem (i.e., minimizing the size of the pruned model subject to a constraint on the accuracy loss), and then use powerful derivative-free optimization algorithms to solve it. To compress a trained DNN, OLMP is conducted within a new iterative pruning and adjusting pipeline. Empirical results show that OLMP can achieve the best pruning ratio on LeNet-style models (i.e., 114 times for LeNet-300-100 and 298 times for LeNet-5) compared with some state-ofthe-art DNN pruning methods, and can reduce the size of an AlexNet-style network up to 82 times without accuracy loss.

Layer-wise magnitude-based pruning (LMP) is an effective DNN compression method and has achieved significant results in many applications. The idea is to prune connections in each layer separately by removing the connections with absolute weight values lower than a layer-specific threshold. Given a threshold for each layer, LMP can prune connections in parallel, which is especially useful for DNNs with millions or billions connections. With well chosen pruning thresholds, LMP can achieve a significant reduction in the number of parameters, while maintaining a relatively low accuracy loss.

However, if the parameters of the LMP are manually adjusted, it is difficult to adjust the optimal parameters and it takes a lot of time and cost. So that the proposed OLMP adopts a heuristic optimization algorithm NCS[1] to optimize the pruning thresholds. Heuristic optimization algorithms have been used to evolve the network structure, and to prune the connections. However, most of them are effective only for shallow networks, and are computational impossible for DNNs with millions of parameters.

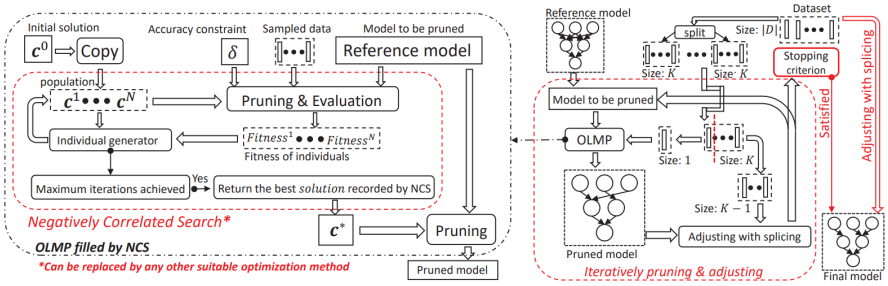

Figure 3: Illustration of DNN compression with OLMP, given an accuracy constraint , and a reference model as inputs. The *Right part* presents the compression process in which pruning and adjusting are conducted iteratively until the stopping criterion is fulfilled. The pruning step adopts OLMP to generate a pruned model, while the adjusting step recovers the accuracy loss of the pruned model. In each loop of pruning and adjusting, it will fetch a small set of data with K batches from the whole data set D, and use one batch for pruning while the rest for adjusting. The *Left part* illustrates the structure of OLMP using NCS. In OLMP, the threshold tuning problem is formulated as a single-objective optimization problem with an accuracy constraint , and then optimized using NCS to get the best found thresholds. After that, a pruned model can be derived by applying LMP on the reference model with the obtained thresholds.

1.4 Applications of OLMP

Without incurring any accuracy loss on test data, OLMP can prune 99.66% parameters of LeNet-5 and 99.12% parameters of LeNet-300-100, which are the best results in comparison to a number of state-of-the-art approaches. Prof.Tang apply OLMP to prune an AlexNet-style deep model and 98.78% parameters can be removed without sacrificing accuracy.[5]

Recently, Google Research successfully applied genetic algorithms to design the structures of DNNs by using clusters of GPU servers.

2 Parameter description

Parameters	Your final values and results		
	F6	**F12**	**OLMP**
lambda	1.1894114828452	0.992687616933805	6
r	0.524759241553746	0.569352821038384	0.1999
epoch	8	114	10
n	1	1	92
Final Result	390.00000000000347	-460	0.9889914106747684
Running Time	47.61029505729675	34.49642562866211	62.54827380180359

- *Summary*
  - *lambda*
    - *role*
      Not only search behavior diversity (i.e. distribution distance), should also care about whether the search agent can find a better solution. We want to find the solution with both high fitness and large distance. The following heuristic is adopted to control the trade-off between two criteria:
      where >0 is a parameter $\begin{cases} \mathrm{x}_i, & \text{if } \frac{f(x_i')}{Corr(p_i')} < \lambda \\ \mathrm{x}_i', & \text{if } otherwise \end{cases}$
      In order to balance better fitness and longer distances, there is a coefficient lambda
    - *effect*
      For F6 and F12, in order to balance better fitness and longer distances, lambda is

close to 1. But the specific value of the lambda needs to be decided together with r.
For OLMP, lambda is irrelevant to the answer. The answer is only related to n.

* *best range*
The experimental results show that the value of lambda is between 1.0 and 1.2.

- *r*
  * *role*
  r is a parameter that is suggested to be set beneath 1, and c is the times that a replacement happens (i.e., x_i' is preserved) during the past epoch iterations. Equation is designed based on the following intuition. A large c implies that the RLS frequently found better solutions in the past iterations, and the current best solution might be close to the global optimum. Thus, the search step-size should be reduced (by r times). On the other hand, if a RLS frequently failed to achieve a better solution in the past iterations, it might have been stuck in a local optimum. In this case, the search step-size will be increased (by r times) to help the RLS explore other promising regions in the search space.
  * *effect*
  For F6 and F12, when the replacement happens many times, the range of the Gaussian mutation may expand or shrink. r has more effect on answer d than lambda. But the specific value of the lambda needs to be decided together with lambda.
  For OLMP, r is irrelevant to the answer. The answer is only related to n.
  * *best range*
  The experimental results show that the value of r is between 0.4 and 0.6. I think this value is close to 0.5, similar to a binary search.

- *epoch*
  * *role*
  To keep the algorithm simple, all RLSs in NCS-C are initialized with the same value of i . Then, each i is adapted for every epoch iterations according to the 1/5 successful rule suggested[2]
  $$\sigma_i = \begin{cases} \sigma_i \frac{}{r}, & \text{if } \frac{x}{epoch} > 0.2 \\ \sigma_i * r, & \text{if } \frac{x}{epoch} < 0.2 \\ \sigma_i, & \text{if } \frac{x}{epoch} = 0.2 \end{cases}$$
  * *effect*
  The definition of epoch in deep neural networks is the number of cycles to train the entire training set.
  For F6 and F12, if the other three parameters are unchanged, only the epoch is changed, and the answer is different. But if we determine the values of epoch and n, only change lambda and r, we can always get a better solution (error is less than 10^{-10}).
  For OLMP, epoch is irrelevant to the answer. The answer is only related to n.
  * *best range*
  The experimental results show that the value of epoch is more than 100 is more suitable. Because when the value of epoch is greater than 100, it can appropriately adjust the range of Gaussian mutation to become larger or smaller, while achieving a better local optimal solution.

- *n*
  * *role*
  For the sake of simplicity, we consider a special case that each search process is a Randomized Local Search (RLS), which produces one new candidate solution in each iteration. In other words, we consider a population-based search method with population size of n($n > 1$), which runs n RLS procedures in parallel and iteratively updates each individual solution in the population with a randomized search operator

* *effect*
  For F6 and F12, because we run programs through multiple processes, we can run many different sets of parameters randomly, so n is not very important. In order to minimize the number of parameters affecting the result, we assume that n is equal to 1, so that the lambda has no effect on the answer, and we fixed the epoch, so the answer is related to r, so it is easy to climb the mountain algorithm or simulate annealing algorithm obtains the local optimal solution.
  For OLMP, n is the only parameter that determines the result.

* *best range*
  For F6 and F12, because we run programs through multiple processes, we think n equals 1 is well. For OLMP, experiments show that we traverse the range of n (1 to 100), we find that the results are independent of other parameters, and when n is equal to 92, the result is the best.

3 Tuning procedure

For F6 and F12, we first consider the extent to which the four parameters affect the answer. We tested some data and found that regardless of the epoch equal, the other three parameters have a chance to get better results. Because we run programs through multiple processes, we can run many different sets of parameters randomly, so n is not very important. In order to minimize the number of parameters affecting the result, we assume that n is equal to 1, so that the lambda has no effect on the answer, and we fixed the epoch, so the answer is related to r.

So we fixed n equal to 1, epoch is an arbitrary positive integer (preferably greater than 100), and the value of lambda is around 0.5. Then we rewrote NCS-client and made it a method that allows us to call multiple processes to get the result. Then we randomly generate the value of r according to the number of cores of our server, and bring the parameters into the rewritten NCS-client or the result. We know that the minimum value of F6 is 390, and the minimum value of F12 is -460. Therefore, we select the parameters of the previous random parameters whose results are close to the minimum value to run a further optimization algorithm.

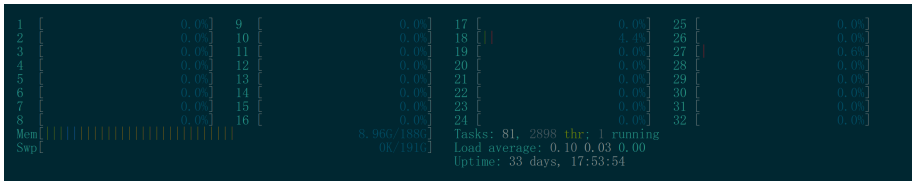

Figure 4: server: We have run 30 processes on this server.

C1 fx 390. 000000000059

	A	B	C	D	E	F	G
1	159771	476	390	0. 524757922	1. 189410763	8	1
2	159774	231	390	0. 524757922	1. 189410763	8	1
3	159771	371	390	0. 524757922	1. 189410763	8	1
4	159772	794	390	0. 524757922	1. 189410763	8	1
5	159771	832	390	0. 524757922	1. 189410763	8	1
6	159774	1267	390	0. 524757922	1. 189410763	8	1
7	159774	1307	390	0. 524757922	1. 189410763	8	1
8	159771	1768	390	0. 524757922	1. 189410763	8	1
9	159774	479	390	0. 524757922	1. 189410763	8	1
10	159773	569	390	0. 524757922	1. 189410763	8	1
11	159773	678	390	0. 524757922	1. 189410763	8	1
12	159771	743	390	0. 524757922	1. 189410763	8	1
13	159774	801	390	0. 524757922	1. 189410763	8	1
14	159773	870	390	0. 524757922	1. 189410763	8	1
15	159771	1021	390	0. 524757922	1. 189410764	8	1
16	159771	1051	390	0. 524757922	1. 189410763	8	1
17	159772	1102	390	0. 524757922	1. 189410763	8	1
18	159774	1334	390	0. 524757922	1. 189410763	8	1
19	159772	1516	390	0. 524757922	1. 189410763	8	1
20	159774	1732	390	0. 524757922	1. 189410763	8	1
21	159773	1767	390	0. 524757922	1. 189410763	8	1
22	159771	1846	390	0. 524757922	1. 189410763	8	1
23	159771	1894	390	0. 524757922	1. 189410763	8	1
24	159770	1952	390	0. 524757922	1. 189410764	8	1
25	159771	1954	390	0. 524757922	1. 189410763	8	1

Figure 5: hill climbing data

2.2.1. *F_6: Shifted Rosenbrock's Function*

$$F_6(\mathbf{x}) = \sum_{i=1}^{D-1}(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f\_bias_6,\ \mathbf{z} = \mathbf{x} - \mathbf{o} + 1, \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

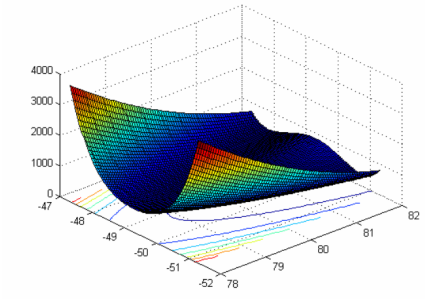

Figure 2-6 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Non-separable
- Scalable
- Having a very narrow valley from local optimum to global optimum
- $\mathbf{x} \in [-100, 100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_6(\mathbf{x}^*) = f\_bias_6 = 390$

Associated Data file:

Name: rosenbrock_func_data.mat
rosenbrock_func_data.txt

Variable: **o** 1*100 vector the shifted global optimum
When using, cut **o**=**o**(1:*D*)

Figure 6: F6

2.2.7. F_{12}: *Schwefel's Problem 2.13*

$$F_{12}(\mathbf{x}) = \sum_{i=1}^{D} (\mathbf{A}_i - \mathbf{B}_i(\boldsymbol{x}))^2 + f\_bias_{12}, \mathbf{x} = [x_1, x_2, ..., x_D]$$

$$\mathbf{A}_i = \sum_{j=1}^{D} (a_{ij} \sin\alpha_j + b_{ij} \cos\alpha_j), \mathbf{B}_i(x) = \sum_{j=1}^{D} (a_{ij} \sin x_j + b_{ij} \cos x_j), \text{ for } i = 1, ..., D$$

D: dimensions

A, **B** are two *D*D* matrix, a_{ij}, b_{ij} are integer random numbers in the range [-100,100],

$\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_D], \alpha_j$ are random numbers in the range $[-\pi, \pi]$.

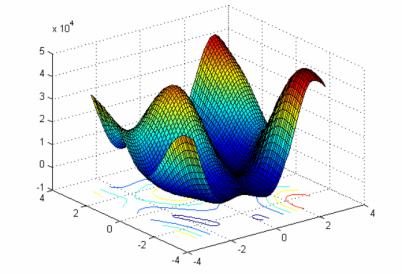

Figure 2-12 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Non-separable
- Scalable
- $\mathbf{x} \in [-\pi, \pi]^D$, Global optimum $\mathbf{x}^* = \boldsymbol{\alpha}, F_{12}(\mathbf{x}^*) = f\_bias_{12} = -460$

Figure 7: F12

After we have the appropriate parameters, we use the hill climbing algorithm as our optimization algorithm to continue to optimize the parameters. After running for 30 days, we got better results (error is less than 10^{-10})

Algorithm 1 Hill Climbing

```
function HILL-CLIMBING
    if vaild parameter then parameter = init()
    end if
    while stopping condition is not reached do: x' = Gaussian-mutation(x) ans = NCS(x) ans' = NCS(x')
        if ans'<ans then: x = x'
        end if
    end while
return x
end function
```

For OLMP, after several random attempts, we were pleasantly surprised to find that the results are not related to lambda, r, epoch, only related to n, so we traversed the range of n (1 to 100) and finally found that when n is equal to 92 At the time, the result is the biggest.

```
0.9889914106747684 {"r": 0.34519791617592477, "epoch": 59, "lambda": 9.224068206198924, "n": 92}
0.9889914106747684 {"r": 0.99, "epoch": 10, "lambda": 9, "n": 92}
0.9889914106747684 {"r": 0.92, "epoch": 92, "lambda": 9.2, "n": 92}
0.9637898053336333 {"r": 0.99, "epoch": 10, "lambda": 9, "n": 46}
0.986920970706275 {"r": 0.99, "epoch": 10, "lambda": 9, "n": 100}
0.971400172536664 {"r": 0.99, "epoch": 10, "lambda": 9, "n": 96}
```

Figure 8: Several parameters

References

[1] T. Ke, Y. Peng, and Y. Xin, "Negatively correlated search," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 542–550, 2016.

[2] H. G. Beyer and H. P. Schwefel, "Evolution strategies – a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[3] S. Ahmed, "Minimizing the symbol-error-rate for amplify and forward relaying systems using evolutionary algorithms," *IEEE Transactions on Communications*, vol. 63, no. 2, pp. 390–400, 2015.

[4] J. Tate, B. Woolford-Lim, I. Bate, and X. Yao, "Evolutionary and principled search strategies for sensornet protocol optimization,"

[5] G. Li, C. Qian, C. Jiang, X. Lu, and K. Tang, "Optimization based layer-wise magnitude-based pruning for dnn compression.," in *IJCAI*, pp. 2383–2389, 2018.