Report for Project NCS

Name: Zhang Yifan

SID: 11711335

1. Algorithm description

1.1 Main Idea of NCS

Evolutionary algorithms are powerful tools for complex optimization problems, which are ubiquitous in both communication and big data analytics. Evolutionary Algorithms search in the solution space of a challenge by iteratively generating a population of new candidate solutions until meeting the predefined halting condition. Information should be shared among individual solutions in the population so that they can cooperatively generate new, hopefully, more promising, candidate solutions and the algorithm could eventually search more effectively.

Negatively Correlated Search belongs to the category of the evolutionary algorithm. The core idea of Negatively Correlated Search is a new model for implementing the cooperation between individuals in a population, which was inspired by an interpretation of collaboration in human behaviours. It is an evolutionary algorithm which maintains multiple individual search process in parallel and models the actions of unique search processes as probability distributions.

We consider a population-based search process as multiple iterative search processes running in parallel. Each iterative search process consists of three procedures. First, one or more initial solutions create randomly. Then, a randomized search operator is applied to the current solutions to generate new candidate solutions iteratively. Finally, the search terminates when meeting the predefined criterion. Any population-based search method captured by the above procedures is similar to a multi-agent system, in which every agent strives to find a solution with better quality in terms of the value of the objective function of an optimization problem. Negatively Correlated Search aims to control each agent to search the part of the search space that other agents will not explore, so it adopts the strategy that different agents should cover different regions of the search space and have different search behaviours. Negatively Correlated Search first calculates the negative correlation between search agents and then selects the agents that have a higher negative relationship. For this reason, Negatively Correlated Search expects each search process to move towards a region that is both promising and is unlikely to be searched by other search processes.

To summarize, Negatively Correlated Search maintains a population of Randomized Local Search and iteratively generates new solutions. At each iteration, each Randomized Local Search is employed to create one unique solution. Then comparing the new solution is then with the solution generated by the same Randomized Local Search in the previous iteration, then discard one of the two solutions. The search terminates when a predefined halting condition is satisfied.

Most evolutionary algorithms maintain the diversity of a population to control the exploration power, to promote search in new promising regions and avoid premature convergence. However, the variety in the Negatively Correlated Search framework is very different from the diversity in other cases, e.g., niching. Negatively Correlated Search defines difference in terms of search behaviours, i.e., the probability distributions associated with Randomized Local Searches. It is the search behaviours, which captures the on-going interaction between search operators and solutions, that matters, not how a population of solutions “looks”. Negatively Correlated Search is capable of pushing a Randomized Local Search to visit regions that are seldom covered by other Randomized Local Searches.

The value of one of the parameters, lambda, can affect the search process and consequently, the performance of Negatively Correlated Search. Different values of λ might be suitable to varying stages of the search, so Negatively Correlated Search employs a time-variant λ. It indicates the correlation between Randomized Local Search and different values might be suitable to varying stages of the search.

In conclusion, Negatively Correlated Search is an evolutionary algorithm featured by its information sharing and cooperation schemes, which explicitly promote negatively correlated search behaviours to explore more effectively in the search space.

1.2 Applications of NCS

> *i.e. What problems NCS can solve*

Negatively Correlated Search can be applied to solve set-oriented optimization problems. Set-oriented optimization problems require seeking a set of solutions that are not only of high quality but also maintain some relationship between them. The simplest version might be to find multiple distinct optima for a multi-modal problem simultaneously. Due to its search mechanism, Negatively Correlated Search is naturally suitable for this type of problems.

Negatively Correlated Search can service in real-world complex optimization problems, such as minimizing the air resistance of automotive fluid design, minimizing the peak side-lobe level in the antenna array, and minimizing the loss of power equipment in the optimal economic scheduling. These complex optimization problems involve many local extreme solutions in the real parameter space. Usually, researchers design special simulation software to solve these problems. It is difficult to obtain explicit optimization functions and gradient information by fitting complex optimization scenarios. Such optimization problems are called multi-modal (Non-convex) real value optimization problems or black-box optimization problems.

Negatively Correlated Search is also a powerful tool for complex optimization problems. It could be used to tuning the protocol of sensor nets, data analytics, complicated optimization regarding communication, facilitate decision-making, training deep neutral issues, or tuning the hyper-parameter of Support Vector Machines. To be more specific, for example, managing network resources based on the analysis of user profiles to achieve higher end-user satisfaction.

Also, it is a promising method for non-convex optimization. We can extend the negatively correlated search to other optimization problems, for example, multi-objective real value optimization, combinatorial optimization, multi-task optimization, etc.

1.3 Main Idea of OLMP

Deep neural networks have outstanding performance. Still, they usually suffer from their enormous number of parameters, which makes them prohibitive to be deployed on platforms with limited memory and processing units, e.g., mobile phones. Layer-wise magnitude-based pruning is a prevalent method for deep neural network compression. It can be used to compress those trained DNNs with over-parametered structures. The idea is to prune connections in each layer separately by removing the relationships 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 of links. With well-chosen pruning thresholds, LMP can achieve a significant reduction in the number of parameters, while maintaining a relatively low accuracy loss. However, tuning the layer-specific thresholds is a difficult task, since the space of threshold candidates is exponentially ample, and the evaluation is costly. Previous methods are mainly by hand and require expertise. An automatic threshold adjusting method is added to the Layer-wise magnitude-based pruning to facilitate the use of the non-professional user.

Optimization-based Layer-wise Magnitude-based Pruning for DNN compression, abbreviated as OLMP, uses negatively correlated search as the derivative-free optimization method to introduce automatic parameter pruning, maximizing the pruning ratio under a constraint on the accuracy loss, and then uses a derivative-free optimization algorithm to solve the problem. Then it converts the problem into an optimizable issue and comes up with a new compression process--the OLMP process and parameter pruning go on iteratively, and the parameter pruning process includes error repairing, guaranteeing the sustainability and result of OLMP. It can choose the pruning thresholds on each layer automatically. The idea of OLMP 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.

The threshold tuning problem is formulated as a constrained optimization problem, which requires minimizing the size (i.e., the number of connections) of the pruned network, subject to a constraint on the accuracy loss. Then, a robust derivative-free optimization algorithm is employed to solve this problem. To deal with the costly evaluation of accuracy loss, OLMP evaluates the accuracy loss of the pruned model on a randomly sampled small data set instead of the whole data set. Note that to compress a trained DNN, OLMP is conducted within a new iterative pruning and adjusting pipeline. In each iteration of the pipeline, one split data set is selected for pruning and adjusting. OLMP is first used to prune the reference model based on one batch extracted from the selected split data set, and then the pruned model is adjusted on the rest batches of the split data set to recover its accuracy.

The iterative pipeline used in the framework is a combination of ITR and DS with the layer-wise hyper-parameters tuned by OLMP. Instead of using the whole dataset in adjusting of each iteration, the proposed pipeline only uses a fraction of data for speeding up the compression. By adopting different optimization algorithms for OLMP or selecting different values for hyper-parameters (i.e., δ, K and pruning loops), this framework of DNN compression can have different implementations.

OLMP is to make the parameter matrix sparse to the greatest extent, which means to reduce the number of zero elements in W as much as possible, and at the same time, the performance will not drop too much. And by sampling some samples from the training set, we can judge whether the current constraints are satisfied. Besides, this paper also uses the weight recovery mechanism to reply to the parameters deleted by mistake.

1.4 Applications of OLMP

> *i.e. What problems OLMP can solve*

DNNs usually suffer from their enormous number of parameters, which makes them prohibitive to be deployed on platforms with limited memory and processing units, e.g., mobile phones. OMLP is a way to tune the pruning thresholds for Layer-wise magnitude-based pruning automatically, and it dramatically facilitates non-professional users to use LMP,

OLMP introduces a new approach regarding deep neural network compression and acceleration. It has significant results in deep neutral networks compression and is an unusual approach to compress trained DNNs with over-parameterized structures.

Optimization-based Layer-wise Magnitude-based Pruning for DNN compression introduces a way to splice together several different technologies regarding deep neural network to solve the problem of model compression to a certain extent.

2. Parameter description

2.1 Final parameters and Results

PARAMETERS	F6	F12	OLMP
lambda	0.965100001	1	2.33456665
r	0.979409	0.7309130760954667	43.4536
epoch	6	290	456
n	1	1	92
Final Result	390.0007645544799	-460.0	0.988991410674
Running Time	48.094483852386475	33.32034230232239	55.97

2.2 Summary

> 1. *What's the role of the parameters*
> 2. *The effect about different values of the parameter on the final performance*
> 3. *The best range for the parameter*

2.2.1 Lambda

2.2.1.1 Role of parameter

λ ∈ (0,+∞) is a parameter.

The value of λ can affect the search process and consequently the performance of NCS. It is used to control the trade-off between two criteria and whether Xi is discarded is determined heuristically:

$$where\ \lambda > 0\ is\ a\ parameter \quad \begin{cases} discard\quad x_i,\ \ if\ \dfrac{f(xi')}{Corr(pi')} < \lambda \\ discard\quad x_i',\ \ otherwise \end{cases}$$

2.2.1.2 Effect of different values

Setting λ to 1 indicates that the solution quality and correlation between Local Searches are equally emphasized and hence can be used as the default setting. However, different values of λ might be suitable at different stages of the search.

A smaller lambda means the algorithm prefers the solutions with larger fitness rather than distribution, so the procedure is more cautious. Otherwise the algorithm will focus on solutions which are more distant.

NCS-C randomly samples a Gaussian distribution with expectation 1 to generate values of λ at each iteration. The standard deviation of the Gaussian distribution is first initialized to 0.1, and then shrink towards 0:

$$\lambda_t = N(1,\ 0.1 - 0.1 * \frac{t}{Tmax})$$

where Tmax is the user-defined total number of iterations for an execution of NCS-C.

2.2.1.3 Best range

I found the best value of λ for NCS is around 1. For F6 is slightly smaller than 1, for F12 is 1, so the range is about 0.95 to 1.

For OLMP, the impact of lambda on result is insignificant, and the final choice (with smallest running time) is slightly over 2.

2.2.2 R

2.2.2.1 Role of parameter

Given an existing solution Xi, the Gaussian mutation operator generates a new solution Xi' using

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

where Xid denotes the dth element of Xi and N(0,σi) denotes a Gaussian random variable with zero mean and standard deviation σi.

Each σi is adapted for every epoch iterations according to the 1/5 successful rule:

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

and c is the times that a replacement happens (i.e., Xi' is preserved) during the past epoch iterations.

2.2.2.2 Effect of different values

R is a parameter that is suggested to be set beneath 1.
A gigantic 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 an 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.

2.2.2.3 Best range

The best r for F6 and F12 are both beneath 1. R is initially set to 0.99 in the paper.

0.979409 for F6 and 0.7309130760954667 for F12. So the best range is about 0.7 to 1.

For OLMP, the impact of r on result is insignificant, and the final choice (with smallest running time) is 43.4536.

2.2.3 Epoch

2.2.3.1 Role of parameter

σ is adapted for every epoch iteration, where Xi' s are pre-served.

The 1/5 successful rule:

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

Epoch is used to update σ.

2.2.3.2 Effect of different values

In the procedure of NCS-C, the pseudo code shows:

```
while(t<Tmax) do:
    ......
    t <-- t + 1
    If mod(t, epoch) == 0
        For i = 1 to N
            Update σi for each RLS according to the 1/5
successful rule
        End For
    End If
End while
```

For every epoch iteration, update the search step-size for each RLS.

As shown by the code, the value of epoch determines the frequency of epoch iteration. If epoch is large, it will search deeply from the old solution, and may focus on the local optimum. If it's small, it's more likely to search a larger area and resulting in larger running time.

2.2.3.3 Best range

The epoch is initially set to 10.

I found the best value of epoch for F6 is 6, for F12 is 290, so for NCS the best range for epoch is below 500.

For OLMP, the impact of epoch on result is insignificant, and the final choice (with smallest running time) is 456.

2.2.4 N

2.2.4.1 Role of parameter

The NSC-C process contains the following steps:

1. N means the algorithm should first randomly generate an initial population of **N** solutions.
2. Then evaluate the **N** solutions with respect to the objective function f and
3. identify the best solution x* in the initial population and store it in BestFound.
4. During each iteration, there should also be **N** new solutions generated.

N stands for the number individuals are searched.

2.2.4.2 Effect of different values

If N is large, the search will carry more areas and it would be easier to find the optimal solution. If N is small, then the answer would probably not be optimal, but time might be shorter.

In the paper, N is set to be 100. The temperature was initialized to 1 and then decreased with a factor 0.85 for every 100 iterations.

2.2.4.3 Best range

The best N for F6 and F12 are both 1. The N for F12 can be slightly larger than this.

I found that for OLMP, the best value for N is 92. Regardless of the other three parameters, when N is 92 in OLMP, the compression rate is always 0.9889914106747684, but the running time might differ.

3. Tuning procedure(15)

> *How did you adjust the parameters above to get better performance? Please describe your tuning method along with the procedure.*

3.1 NCS

I mainly adjusted the parameters by the combination of two methods: circulation and stochastic.

First of all, I used the parameters settings introduced in the paper but found the result was way too big. Then I started searching by stochastically changing one of the parameters and see the results. Once the reduction of the result is prominent, I moved on to the next parameter.

Then, after all four parameters were set to the best range, I started circulation. By slightly changing the parameters in circulation, I could print out all four parameters, the final result and keep a record of those desired parameters and results.

I did this by editing the ncs_client.py, instead of using the parameter uploaded by the JSON file, I changed to using the parameters in the circulation, and then I print out the results.

The pseudo-code is shown as following:

```
For d = di to dj:
   For c = ci to cj:
      For b = bi to bj:
          For a = ai to aj:
              assign all _lambda, r, epoch, n
              send parameters to ncs_para and update ncs_c
              begin ncs_res loop
              PRINT a,b,c,d
              PRINT ncs_res
          End For
      End For
   End For
End For
```

3.2 OLMP

I stochastically adjusted the parameters for OLMP.

First I found that N is the only parameter that would change the result. The result would be the same, regardless of the other three parameters. So first the problem is to find the optimal N.

I found that for OLMP, the best N is 92. When N is 92 in OLMP, the compression rate is always 0.9889914106747684, but the running time might differ.

Then I tried to reduce the running time. At first it was over 70 seconds, but after I stochastically changed the other three parameters, I at last reduced it to 55 seconds.

4. Summary

By doing project two of CS303, I understood the main idea of Negatively Correlated Search and Optimization-based Layer-wise Magnitude-based Pruning for DNN compression. Also, I grasped the idea of parameter adjusting and worked out the best solutions. Thanks to all the teachers and teaching assistants for their patience and help.