Find Optimal Model among Various Neural Networks Models Using Monte-Carlo Tree Search

Jaeyoung Moon, Misun Kim, Dongil Han*
Sejong University, Computer Engineering
moon920110@gmail.com, didi027@naver.com, dihan@sejong.ac.kr

Abstract

Nowadays, most researchers in machine learning field will agree that the deep neural networks(DNNs) provide the best performance in pattern recognition, computer vision, natural language processing and so on. So many DNNs architectures are built relying on mathematical heuristics and prior knowledge. We can make some new models by ourselves depending on the platforms, the type of data and applications. If we can experiment all possible cases, then we will find the best approximate solution. Unfortunately, it is hard to do like that because experimenting every possible case costs too much resources. In this paper, we suggest the method to find optimal neural networks model among various models by using Monte-Carlo Tree Search [1].

Keywords: Optimal model, Monte-Carlo Tree Search, Neural networks

1. Introduction

Recent years have seen an evolution of deep learning which was only made possible with the help of ever increasing computational power and easy to use deep learning frameworks. Such frameworks laid the path towards a new era of research where one can experiment with a variety of hyperparameters and ultimately finding the best performance models. However, researchers still must spend unnecessary time until one study result.

Suppose we need to compare the performance of n-different neural networks(NNs) models and train m-epochs for each NNs model to converge on reliable result. The total training time to figure out optimal NNs model will be proportional to n*m times. It causes too much cost if there are too many NNs models to compare or each NNs model has very deep architecture which spends too much time to train as like VGGNet 19 [14].

In this research, we used Monte-Carlo Tree Search method to find optimal NNs model among various NNs models fast and efficiently. Many techniques, such as pre-training, evaluation function, explore and exploit [8], were applied to adjust Monte-Carlo tree search method to our research, but they are a little different from general ways. As a result, we succeeded to find optimal model and remarkably reduced total training time. Additionally, we casually found that this method can improve general performance.

2. Techniques

2.1 Monte-Carlo Tree Search

Monte-Carlo simulation is a statistical method to get statistic convergence result by doing random simulation infinitely [7]. There are 4 steps in original Monte-Carlo Tree Search that is Selection, Expansion, Simulation and Backpropagation [1]. When a parent node selects a child node, there may be two cases. The first case is that all child nodes have been visited at least once. The other case is that some child nodes are not yet visited. In the first case, we should use the Selection step. The Selection is the way to choose next child using a function that follows a specific policy like UCB algorithm [5]. On the other hand, if there exist one or more unvisited child nodes, we should use the Expansion step which randomly chooses child node among unvisited child nodes.

We adopted this Monte-Carlo tree search algorithm for our research without the Simulation step. We just use the Selection and the Expansion steps to select next path. Each node of the tree is combination of the NNs elements such as convolution, pooling, batch normalization [4] and so on. We called a node that include NNs elements as a Block. And a special node called evaluation function node is placed between a parent Block and child Blocks. This evaluation function node helps to choose which direction to go to construct optimal model using explore and exploit method Moreover, an important aspect of our method is that child Blocks share the weights of the parents Block.

* Corresponding Author: Dongil Han

2.2 Pretraining

We applied the pre-training to all models contained from root node(Block) to each leaf node(Block) alternately n-times. Then initialize the weights of each Blocks by the pre-trained weights.

2.3 Evaluation Function

To find an optimal model, the child Block should be chosen such that it can construct the best performance model while tree is being branched. The evaluation function is used at this Selection step. The evaluation function node which is posed at branch point evaluates each child Block. The evaluation value is a sum of validation accuracy of recent n-steps and the evaluation function is the average of validation accuracy of recent n-steps. The separation of the evaluation value and the evaluation function is for the convenience of implementation.

The evaluation value is as follows:

$$\text{Evaluation Value} = \sum_{recent\ n\ steps} validation\ accuracy$$

The evaluation function e is as follows:

$$e = \begin{pmatrix} \dfrac{\sum_{recentR_c\ steps} validation\ accuracy}{R_c}, if\ R_c \leq n \\ \dfrac{\sum_{recent\ n\ steps} validation\ accuracy}{n}, otherwiase \end{pmatrix}$$

Where, R_c is the number of times child Block was referenced and n is constant.

There are three reasons to use validation accuracy of recent n-steps for the evaluation function. First, if the performance of the models is guaranteed to some degree, the training accuracy will converge to almost 100 percent. So, it is meaningless to compare the training accuracy for training dataset. Second, since the accuracy of the training models converges as the training progresses, in general, the reliability of the results are increased as the results are trained recently. In a similar sense, lastly, because there is some difference of the convergence speed for each model, the model with the highest evaluation value at the beginning may not be the optimal path at the end. Therefore, we need to concentrate on the recent training results.

2.4 Exploration and Exploitation

When the tree is branched, we must decide whether to take the path which is the best performing or taking the random path which may be getting better performance. These problems are called exploration and exploitation problems [8]. In this paper, we used UCB1 algorithm [5] to select child Block. UCB1 algorithm selects the next child path by appropriately combining v_i which is the average performance of each child path and $R_{c,i}$ which is the number of times the child paths are referenced. UCB1 algorithm is as follows:

$$\text{UCB1} = v_i + C\sqrt{\frac{2lnR_p}{R_{c,i}}}$$

Where R_p is the number of times parent Block was referenced and C is a constant which determine the ratio between the exploration and the exploitation. To apply UCB1 algorithm to our research, the left term can be replaced by the evaluation function *e*. Therefore, we can represent the UCB1 equation as follows:

$$\text{UCB1} = e_i + C\sqrt{\frac{2lnR_p}{R_{c,i}}}$$

Where, e_i is the evaluation function of each child.

3. Solution

In this section, we will see overall process how to find the optimal NNs model among various NNs models with the techniques explained above.

3.1 Building Tree Model and Pre-training

The models to be compared may have same Block structure which is possible to be shared by lower Blocks or may have different Block structure. We implemented it as having a same Block structure part and recommend implementing as having a same Block structure part, because it can help your models have high performance for unseen data. After making the models to be compared, organize them into a tree structure. If there exists any same part between the models and they are placed on similar order, place them as one. Figure 1 shows a very simple example. In this way, by combining same parts and separating different parts as child nodes, we can make several NNs models into a single tree structure.

After making the tree structure, we should perform the pretraining. Of course, you can ignore the pre-training section depending on your application.

3.2. Actual Training

Since there is only one root in the tree structure, the input data for the training is given only once for all models. But the only one model that is ultimately constructed through all evaluation functions along the tree structure is trained by that data.

If there exist more than or equal to one unvisited child Blocks in the evaluation function node, the evaluation function node will select random child Block among unvisited Blocks. Otherwise, the evaluation function node will choose the next child Block by using the UCB1 algorithm with the evaluation function. These are the Expansion and the Selection steps.

After a model is completely constructed by above methods, perform feedforward to the constructed model and the weights of the trained path are updated through the backpropagation [12] algorithm and the evaluation function nodes are updated by using the validation accuracy. For every evaluation function node in the selected path, accumulates the validation accuracy as the evaluation value and increase the number of times the child Block was referenced.

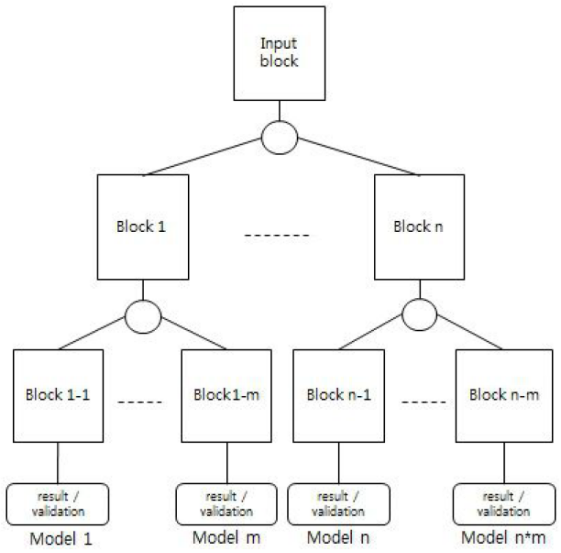

Figure1. **The overall configuration of the tree structure.** Suppose each Block has NNs elements and each complete NNs model is constructed by each path from the root to the leaf. The Block1is shared by the model 1 to the model m. Also, since all models in the tree pass the Input Block, the Input Block is shared by all models.

4. Experiment

Every experiment was operated 200 epochs each and pretrained 4 times. The constant C used in UCB1 algorithm for exploration and exploitation is fixed by 0.05. The experiments were conducted 10 times for each case and CIFAR-10 dataset [6] was used for every experiment. We also trained every independent model 10 times to compare the trained results with the results trained by our method.

4.1 Experiment 1: There Exists Only One Overwhelming Performance Model

First experiment is performed with 4 models. One of them has the overwhelming performance and rest of them have low performance. Details of the models are in the Table 2. This experiment is basically aimed to figure out whether our method can find optimal model among various models when there is a clear optimal model. For making only one overwhelming model, we constructed the last Block of the first path by wider architecture and constructed the last Blocks of others by narrower architecture.

Table 1: Operation of each level of tree

Level of Tree	Operation
Lv1	CONV CONV BATCH NORM
Lv2	POOLING CONV CONV BATCH NORM
Lv3	POOLING CONV CONV BATCH NORM FC FC OUPUT

Table 2: CNNs architectures used for the first experiment

Level of Tree	Model1	Model2	Model3	Model4
Lv1	3X3, 64 3X3, 64 -	3X3, 64 3X3, 64 -	3X3, 64 3X3, 64 -	3X3, 64 3X3, 64 -
Lv2	MAX 3X3, 128 3X3, 128 -	MAX 3X3, 128 3X3, 128 -	MAX 3X3, 128 3X3, 128 -	MAX 3X3, 128 3X3, 128 -
Lv3	MAX 3X3, 256 3X3, 256 - 1024 1024 10	AVG 3X3,32 3X3,32 - 16 16 10	AVG 3X3,32 3X3,32 - 16 16 10	AVG 3X3,32 3X3,32 - 16 16 10

Table 3: Result of the first experiment.

A: Average accuracy when the model is selected B: The number of times the model is selected as optimal C: Average accuracy of each independent model

Models	A	B	C
Model1	86.73	10	85.7
Model2	0	0	82.1
Model3	0	0	81.3
Model4	0	0	82.6

As a result, we were able to find and to train the optimal model for all 10 experiments. This result shows that our method can find and train the optimal model in short time if there is an overwhelming optimal model. The path of the optimal model was referenced on average 188 times in the first evaluation function node and 178 times in the second evaluation function node, respectively.

4.2 Experiment 2: The Performance Difference Between Models is Not Large

Second experiment is also performed with 4 different NNs models. However, the performance difference between each model is not as large as the first experiment. This experiment is aimed to figure out whether we can find the optimal model when the 4 models have similar performance but have clear rank. To prevent the training from moving in one direction in the early epochs, we assigned the models by zigzag according to their performance.

Table 4: CNNs architectures used for the second experiment

Level of Tree	Model1	Model2	Model3	Model4
Lv1	3X3, 64 3X3, 64 -	3X3, 64 3X3, 64 -	3X3, 64 3X3, 64 -	3X3, 64 3X3, 64 -
Lv2	MAX 3X3, 128 3X3, 128 -	MAX 3X3, 128 3X3, 128 -	MAX 3X3, 128 3X3, 128 -	MAX 3X3, 128 3X3, 128 -
Lv3	MAX 3X3, 256 3X3, 256 - 1024 1024 10	AVG 3X3,32 3X3,32 64 64 10	AVG 3X3,64 3X3,64 16 16 10	AVG 3X3,128 3X3,128 - 256 256 10

Table 5: Result of the second experiment.

A: Average accuracy when the model is selected B: The number of times the model is selected as optimal C: Average accuracy of each independent model

Models	A	B	C
Model1	86.5	9	85.7
Model2	0	0	83.9
Model3	0	0	84.1
Model4	86.56	1	84.9

As a result, the best performance model was selected nine times as an optimal model, and the second performance model was also chosen once as an optimal model. Since Monte-Carlo simulation is stochastically converged on numerical results, sometimes it makes mistake finding optimal model if some models have close performance to each other.

5. Conclusion

Through this experiment, we figured out that we can reduce training resources more than half compared to independent models by using our method. Despite of these successful results of experiments, there remain some challenges. We need to adopt this method to more diverse dataset. Also. the experimented models were very simple and small, and the number of models is small too. Therefore, to prove the general performance of our method, we need to expand the architecture of each NNs model and the structure of the tree. If we overcome these problems and find solutions, we can dramatically reduce the resources required to construct and to apply a new model to the application. Until the internal mechanism of NNs is figured out, this method that we proposed for finding optimal model using Monte-Carlo tree search will be very useful.

Acknowledgements

This work was supported by a National Research Foundation of Korea Grant funded by the Korean Government (NRF-2017R1D1A1B03028394), and Korea Institute of Planning and Evaluation for Technology in Food, Agriculture, Forestry and Fisheries(IPET) through (Advanced Production Technology Development Program), funded by Ministry of Agriculture, Food and Rural Affairs(MAFRA)(No 315091-3)

References

[1] Chaslot, Guillaume Maurice Jean-Bernard Chaslot. Monte-carlo tree search. Diss. Maastricht University, 2010.
[2] Deb, Kalyanmoy, et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II." IEEE transactions on evolutionary computation 6.2 (2002): 182-197.
[3] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[4] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
[5] Kocsis, Levente, and Csaba Szepesvári. "Bandit based monte-carlo planning." European conference on machine learning. Springer, Berlin, Heidelberg, 2006.
[6] Krizhevsky, Alex, and G. Hinton. "Convolutional deep belief networks on cifar-10." Unpublished manuscript 40 (2010): 7.
[7] Mahadevan, Sankaran. "Monte carlo simulation." MECHANICAL ENGINEERING-NEW YORK AND BASEL-MARCEL DEKKER- (1997): 123-146.
[8] March, James G. "Exploration and exploitation in organizational learning." Organization science 2.1 (1991): 71-87.
[9] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." Technical report, Neural Information Processing Systems (NIPS) Deep Learining Workshop, (2013).
[10] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.
[11] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." IEEE Transactions on knowledge and data engineering 22.10 (2010): 1345-1359..
[12] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." nature 323.6088 (1986): 533.
[13] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
[14] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).