Introduction to Artifitial Intelligence 10 (2019) Submitted 2019/11/15; Published 2019/11/15

Development of NAS

Feng Shi FENGS19@MAILS.TSINGHUA.EDU.CN
Class:91
Institute for Interdisciplinary Information Sciences
Tsinghua University, Beijing, China

Abstract

NAS is a set of algorithms to search for the neural architectures. From 2017, the publication date of the first essay delivered by Google Brain about NAS (Zoph and Le, 2016), NAS has become one of the topics of machine learning. In our survey, we will demonstrate the development of NAS from the conventional NAS to PNAS and ENAS. Simultaneously, in the fourth part of our survey, we will demonstrate an application of the NAS algorithms in the field of image recognition. In the fifth section, to demonstrate the convenience of NAS, we have done an experiment using AutoKeras on Colab. In the last part of our survey, we show a big picture of the researches done on NAS and then list several possible improvements of NAS in the future years.

Keywords: NAS, PNAS, ENAS, AutoML, AutoKeras

1. Introduction

NAS is an important part of AutoML (Automated Machine Learning). The full name of NAS is Neural Architecture Search. NAS is a set of algorithms with the purpose of searching for the best optimal neural architectures. It is widely acknowledged that neural architectures occupy the dominant position when training for an effective neural network. If the neural architecture has some inevitable flaws, the product cannot be satisfying no matter how to train the neural network. Sometimes, the neural architectures can be obtained by transferring learning but it's still a process of trial and error. However, in the face of this problem, NAS can design neural architecture automatically. In this survey, we will demonstrate the representative papers in this field mainly made by Google which is a pioneer of NAS. We can see how the NAS algorithms develop from using reinforcement learning by RNN to more efficient ways such as the optimizations in PNAS and ENAS. However, currently, NAS is not powerful enough to meet the actual demands. Hence, there is still much work to be done.

2. The Primitive Version of NAS

In ICRL 2017, Google brain delivered a paper of Neural Architecture Search With Reinforcement Learning (Zoph and Le, 2016). This paper for the first time introduced a method to search for the relatively best network architecture by algorithm automatically instead of searching manually. Also, the paper established the fundamental frame of NAS. To recapitulate, the author used a RNN (Recurrent Neural Networks) controller to run reinforcement

learning and generated the network without intervention. Figure 1 is an overview of this NAS algorithm. In order to get a further understanding, we need to learn a little about RNN at first.

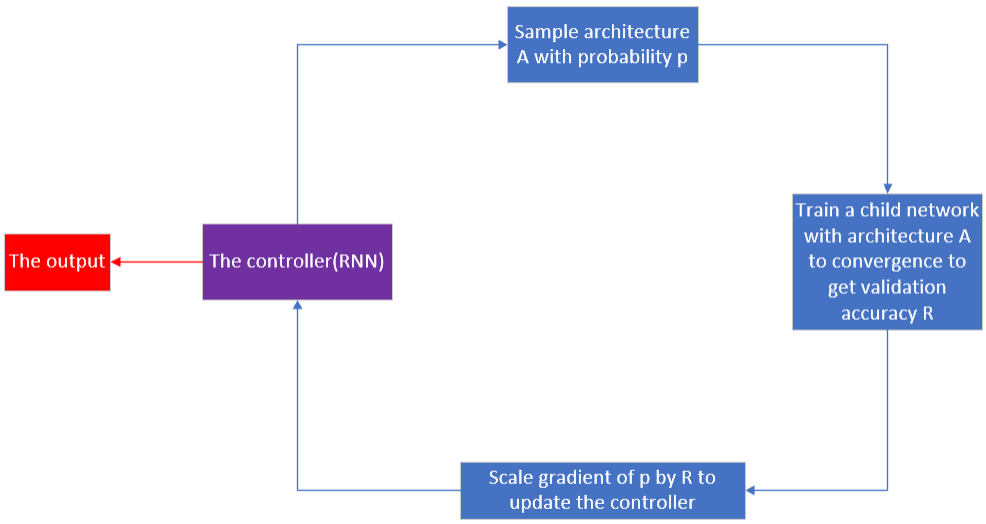

Figure 1: An Overview of NAS.

The connections of RNN network between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. The computing sequence according to a type of time series can solve the problems caused by the loops. Actually, it provides a unfolding view of the RNN structure. Further introduction of RNN can be learned in the paper of An Empirical Exploration of Recurrent Network Architectures (Jozefowicz et al., 2015).

According to the essay, "Our work is based on the observation that the structure and connectivity of a neural network can be typically specified by a variable-length string. It is therefore possible to use a recurrent network C the controller C to generate such string", we can see that the reason why using RNN as the controller is that RNN can generate the variable-length string. More precisely, as it shows in figure 2, the parameters such as numbers of filters, filter width and stride width can be store in a string such that every prediction is carried out by a softmax classifier and then fed into the next time step as input (Zoph and Le, 2016). At last, we can get a reward of the network by limiting the amount of the layers.

How to use the reward to update the parameters of the RNN controller is also critical. The RNN controller uses the prediction to design the child network consists of some actions $a_{1:T}$. Then we can maximize the expected reward $J(\theta_c) = E_{P(a_1:T;\theta_c)}[R]$ by the controller

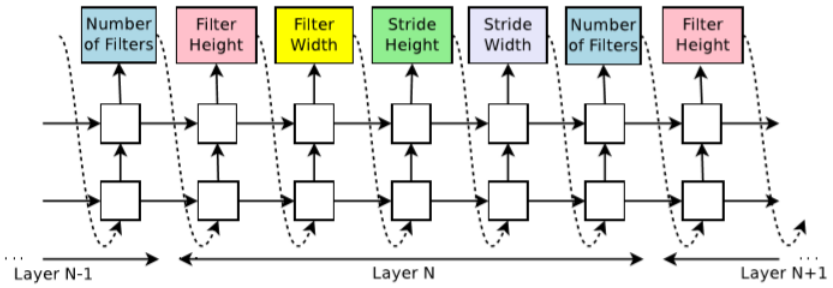

Figure 2: Use the Controller to Generate the Parameters as a Length-variable String (Zoph and Le, 2016).

in order to optimize the neural architecture. For the reward R is not differentiable so the author used policy-gradient algorithm to update θ_c. The formula is as below.

$$\nabla\theta_c J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}[\nabla\theta_c log P(a_t|a_{(t-1:1);\theta_c})R] \tag{1}$$

$$\approx \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla\theta_c log P(a_t|a_{(t-1):1};\theta_c)R_k \tag{2}$$

In order to reduce the variance of the estimation, the RNN finally uses an estimated quantity with baseline function to update θ_c as below:

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla\theta_c log P(a_t|a_{(t-1):1};\theta_c)(R_k - b) \tag{3}$$

Having the following method of the generating of the CNN (merely contains conv structure), the generating of some more complex networks is also possible. For example, the skip connections structure in the ResNet can be installed in and the obtained neural architectures which are able to deal with some more complicated features.

However, although the RNN controller with reinforcement learning already has the capacity to achieve some satisfying results such as the NASNet, which is an application for scalable image recognition (Zoph et al., 2017), the demand of computing power is still extremely huge. In order to popularize NAS to replace the conventional manual searching methods, the improvements of searching space, searching strategy are taken priority.

3. PNAS and ENAS

3.1 An Summary of PNAS

PNAS is Progressive Neural Architecture Search which optimizes the searching strategy of NASNet. The algorithm of NASNet is too slow to be applied to reality, especially the

companies and individuals can't afford the unbearable expenditure of GPUs (or TPUs). The researchers of Google found some optimizations to reduce the expenditure. At first, as the paper Progressive Neural Architecture Search (Liu et al., 2017) says, we can limit the searching space to a smaller set than previous one. Actually, the RL method of some operators are discovered never used before. The operator space can be narrowed to 8 operators. At last, the estimated total number of unique cells is 10^{12} when the previous one is about 10^{28}. However, the searching space is still large so the author used a layering algorithm from cell to CNN. In the nth iteration, there are K candidate cells with b pairwise in each cell. These K cells will be trained and return a reward then generate the cells with $b+1$ pairwise. Then using the surrogate function to select the best K ones. The whole process is showed in figure 3.

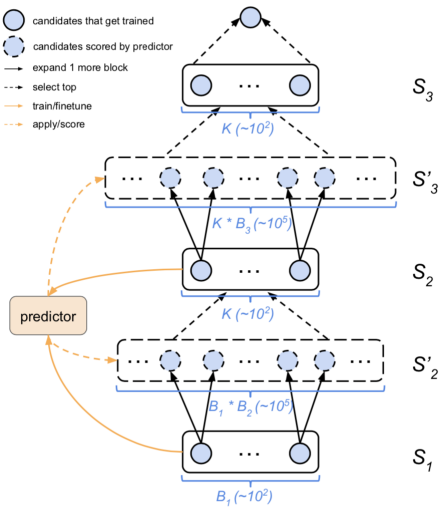

Fig. 2. Illustration of the PNAS search procedure when the maximum number of blocks is $B = 3$. Here $\mathcal{S}_b$ represents the set of candidate cells with b blocks. We start by considering all cells with 1 block, $\mathcal{S}_1 = \mathcal{B}_1$; we train and evaluate all of these cells, and update the predictor. At iteration 2, we expand each of the cells in $\mathcal{S}_1$ to get all cells with 2 blocks, $\mathcal{S}'_2 = \mathcal{B}_{1:2}$; we predict their scores, pick the top K to get $\mathcal{S}_2$, train and evaluate them, and update the predictor. At iteration 3, we expand each of the cells in $\mathcal{S}_2$, to get a subset of cells with 3 blocks, $\mathcal{S}'_3 \subseteq \mathcal{B}_{1:3}$; we predict their scores, pick the top K to get $\mathcal{S}_3$, train and evaluate them, and return the winner. $B_b = |\mathcal{B}_b|$ is the number of possible blocks at level b and K is the beam size (number of models we train and evaluate per level of the search tree).

Figure 3: The Whole Process of PNAS (Liu et al., 2017).

The paper also improved the predictor by using LSTM and MLP as the predictor which is similar to the model in the paper Peephole: Predicting Network Performance Before Training (Deng et al., 2017). Suppose I_1, I_2 are the characteristic patterns and O_1, O_2 are the operations on the patterns, correspondingly. Then a pairwise can be represented by $< I_1, I_2, O_1, O_2 >$. They are all encoded by one-hot vectors. By embedding, they can be converted to a $4D$ dimensions vector as the input of the LSTM and MLP. According to the experiments, the performance has improved but is still a little bit disappointing.

3.2 An Summary of ENAS

Now, let us turn to the ENAS. ENAS is a revolution of NAS in a way, which reduces the training time sharply again. The ENAS was first came up with in the paper Efficient Neural Architecture Search via Parameter Sharing (Pham et al., 2018) method. This is a fast and inexpensive approach to the automatical model designing. The main idea of ENAS

is forcing all child models to share weights to eschew training each child model from scratch to convergence. The paper represents the searching space of the NAS by a DAG (Directed acyclic graph) as shown in the figure 4 and then gives the training methods of 3 types of DAG. The first one is to train a structure similar to VGG (Simonyan and Zisserman, 2014). In order to design recurrent cells, the author employed a DAG with N nodes, where the nodes represent local computations, and the edges represent the flow of information between the N nodes. The RNN controller has 4 functions (tanh, ReLU, identity, sigmoid), which can generate $4N \times N$ architectures. This is because the nodes are independent and each node can be activated by one of the 4 functions. The whole algorithm behaves as shown in the figure 5.

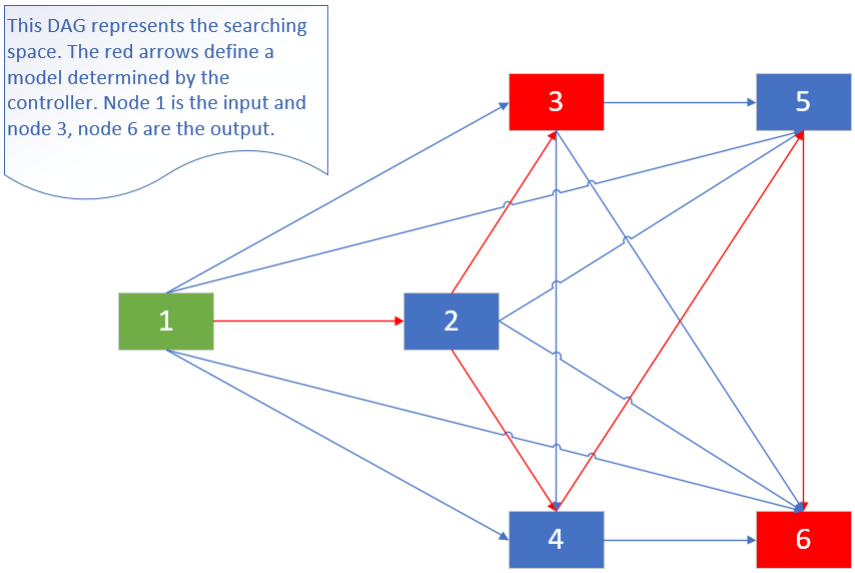

Figure 4: Use DAG to Represent the Searching Space.

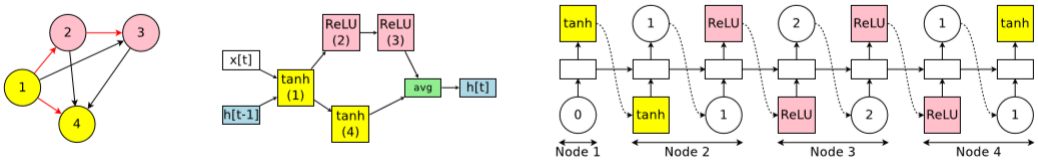

Figure 5: An Example of Using ENAS to Design Recurrent Cells (Pham et al., 2018).

The second DAG is an architecture like ResNet with shortcut connections. The architecture only has one input but includes 6 operations: convolutions with filter sizes 3×3 and 5×5, depthwise-separable convolutions with filter sizes 3×3 and 5×5 (Chollet, 2017), and max pooling and average pooling of kernel size 3×3. Suppose there are L layers, then we have $6L$ possibilities of operations. Then we have $6L \times 2 \times \frac{L(L-1)}{2}$ architectures for the graph is directed. The whole algorithm behaves as shown in the figure 6. And the last

one is to design convolutional cells only through training the block modules instead of the whole network. Based on the Block modules, we can get the the whole network architecture.Actually, for a B nodes NAG, it will generate $(5 \times (B-2)!)2$ network architectures. To sum up, the ENAS uses the technique of weights sharing based on the training of the Block modules to make the NAS algorithm efficient for the applications in reality.

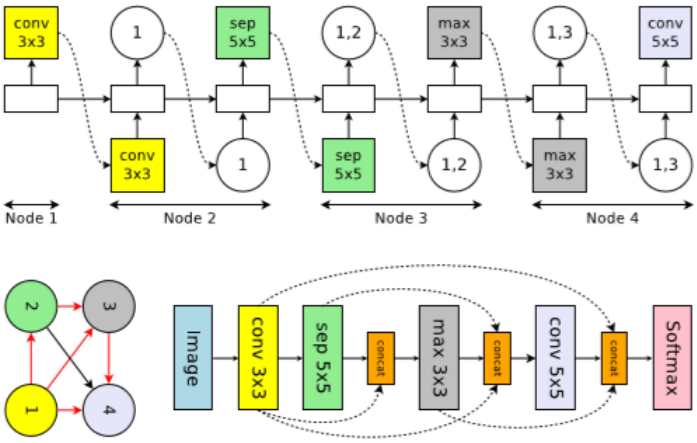

Figure 6: The Example of Using ENAS to Deriving Architectures (Pham et al., 2018).

As a consequence, we can see that PNAS and ENAS are both based on the conventional NAS while they are both revolutionaries in the area of NAS. PNAS and ENAS make the architecture search can be completed with little computation resource and render NAS become a trend in the area of AutoML. With the methods of differentiable architecture search (Liu et al., 2018), NAO based on encode-decode framework (Luo et al., 2018) and even random search (Li and Talwalkar, 2019), the efficiency of NAS has been already improved a lot. Actually, NAS has already replaced the manually architecture search in some of the applications of machine learning such as image recognition and object detection(such as the NAS-FPN recently by Google (Ghiasi et al., 2019)).

4. An Application of NAS

Let us have a brief summary of one of the applications of NAS. In the paper made by Google Brain (Zoph et al., 2017), the author applied NAS to scalable image recognition. The model is still based on the conventional NAS which is able to generate neural architectures automatically (Zoph and Le, 2016). However, it is inspired by ResNet and GoogleNet and used a structure to stack the cells (convolutional layers) together with more copies and each cell is with their own parameters to design a convolutional architecture. Actually, the convolutional cell is output by the RNN controller and consists of B layers. For every block we have 5 steps which can be seen in figure 7 and figure 8.

These 5 steps are the core of this paper, which is the searching space well designed by the author. The difficulty of searching has sharply decreased by constructing the convolution cells (Normal Cell and Reduction cell) and choose the well-performed ones in it. In order to

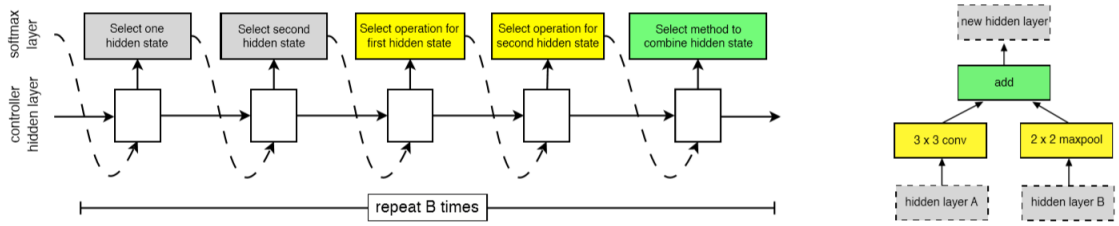

Figure 7: The Process of Constructing 1 Block (Zoph et al., 2017).

- Step 1: Select a hidden state from h_i, h_{i-1} or from the set of hidden states created in previous blocks.
- Step 2: Select a second hidden state from the same options as in Step 1.
- Step 3: Select an operation to apply the hidden state selected in Step 1.
- Step 4: Select an operation to apply the hidden state selected in Step 2.
- Step 5: Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.

Figure 8: The Prediction Steps.

predict Normal Cells and Reduction Cells simultaneously, there will be $2 \times 5 \times B$ outputs of the RNN controller, which uses $5 \times B$ outputs to predict B blocks of Normal Cells and another $5 \times B$ outputs for the Reduction Cells. The RNN controller is a LSTM with 100 nodes in every hidden layer.

Actually, NASNet came up with a method to search for a single unit of the whole network and then stack more unit on a big data set. Therefore, NAS can be applied to some relatively large data sets. The experiment shows that the architecture made by NAS indeed performs better than the best neural networks at that time such as the champion of ImageNet2017,SENet (Hu et al., 2018). We can see it from the data in figure 9. It indicates that NAS is able to take the place of algorithm engineers under some special situations(mainly about image processing). In fact, NAS has also achieved a lot in the area of image segmentation (Liu et al., 2019) and the next section also demonstrate the convenience of NAS on the subareas related to images. It's worth noting that NASNet was training on CIFAR-10 instead of ImageNet still because of the limitation of the computing power. We will mention this as a possibly improvement of NAS in the last section.

5. The Convenience of NAS

To show the convenience of NAS and make a closer approaching to the NAS algorithms, I have done a simple experiment using Auto Keras (Jin et al., 2019) on Colab. The source code is as shown in figure 10 (Vzquez, 2018). Actually, it is used for handwritten digit recognition. The main part of the code is not more than 10 rows, which is much more convenient than the conventional methods such as the similar projects with TensorFlow.

We can see how it works from the console of Colab. From figure 11 we can see that all the searching is automatical and every model will be trained completely. From the essay (Jin et al., 2019) made by the author of AutoKeras, we can learn that after an iteration,

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [60]	299×299	23.8 M	5.72 B	78.8	94.4
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [58]	299×299	55.8 M	13.2 B	80.1	95.1
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [68]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [69]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
SENet [25]	**320×320**	**145.8 M**	**42.3 B**	**82.7**	**96.2**
NASNet-A (6 @ 4032)	**331×331**	**88.9 M**	**23.8 B**	**82.7**	**96.2**

Figure 9: The Performance of NASNet (Zoph et al., 2017).

```
!pip install autokeras
from keras.datasets import mnist
import autokeras as ak
if __name__ == '__main__':
  (x_train, y_train), (x_test, y_test) = mnist.load_data()
  x_train = x_train.reshape(x_train.shape + (1,))
  x_test = x_test.reshape(x_test.shape + (1,))
  clf = ak.ImageClassifier(verbose=True, augment=False)
  clf.fit(x_train, y_train, time_limit=12 * 60 * 60)
  clf.final_fit(x_train, y_train, x_test, y_test, retrain=True)
  y = clf.evaluate(x_test, y_test)
  print(y * 100)
```

Figure 10: The Source Code of an Experiment.

the program will add some operations or adjust the network architecture according to losses and metric values in the log (actually by ENAS). AutoKeras totally hides the details of the constructing process of the neural architecture so the whole process is extremely concise and efficient. The development from the initial version of NAS to the PNAS and ENAS renders the users an unbelievable low cost of machine learning. The previous projects with thousands rows of code can be simplified to several hundreds rows of code. Nowadays, AutoKeras, Auto-Sklearn (Feurer et al., 2015) and Cloud AutoML (Li and Li, 2018) all provide the service of NAS. There is no doubt that NAS will become more and more convenient and widespread in the future.

```
Saving Directory: /tmp/autokeras_U1OHYP
Preprocessing the images.
Preprocessing finished.

Initializing search.
Initialization finished.


+----------------------------------------------+
|               Training model 0               |
+----------------------------------------------+

No loss decrease after 5 epochs.


Saving model.
+-----------+-----------------------+-----------------------+
|  Model ID |          Loss         |      Metric Value     |
+-----------+-----------------------+-----------------------+
|     0     |  0.1328190542757511   |  0.9880000000000001   |
+-----------+-----------------------+-----------------------+

+----------------------------------------------+
|               Training model 1               |
+----------------------------------------------+

No loss decrease after 5 epochs.


Saving model.
+-----------+-----------------------+-----------------------+
|  Model ID |          Loss         |      Metric Value     |
+-----------+-----------------------+-----------------------+
|     1     |  0.11870087087154388  |  0.9907999999999999   |
+-----------+-----------------------+-----------------------+

+----------------------------------------------+
|               Training model 2               |
+----------------------------------------------+

No loss decrease after 5 epochs.
```

Figure 11: The Running Log of the Program.

6. Some Further Improvement of NAS and Big Open Problems

6.1 More Researches and Possible Improvements of NAS

The basic idea of the recently designed NAS algorithms are still similar to the first essay of NAS delivered by Google (Zoph and Le, 2016). The papers after that mainly improve the performance of NAS in these 3 following aspects as shown in the figure 12.

The first one is to improve the searching space. With an oversized searching space, the NAS algorithm can certainly not achieve an acceptable performance. We can see that a large portion of the researches done on NAS contains the improvements of the searching space. Some NAS algorithms aim to search the whole neural structure. The typical example of this idea is the conventional NAS. Another idea is to search for some small structure

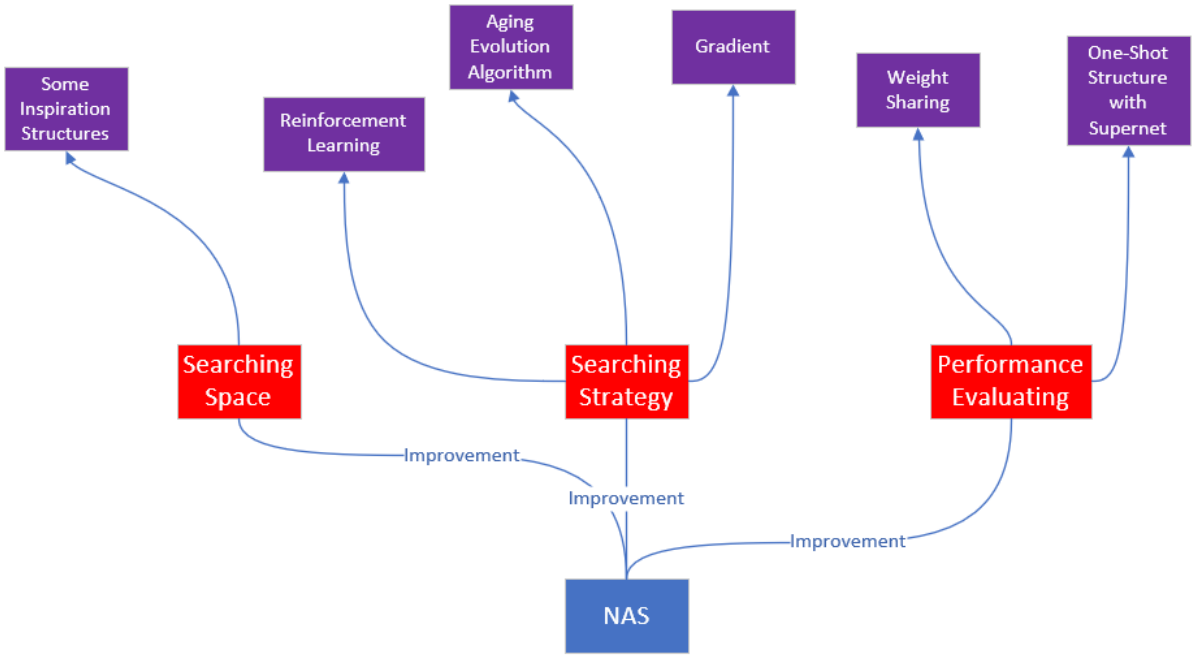

Figure 12: Three Aspects to Improve NAS.

like cells in the PNAS and then use these modules to generate the whole network. Simultaneously, adding some special structures into the searching space has also been thought about. For instance, the idea of residual connection structure and the idea of repeating the convolutional cells are adopted in NASNet (we can see it from figure 13). Thus it can be seen that the well designed searching space can improve the universality of the algorithms and meanwhile decrease the computation cost.

The second aspect is the searching strategy. Some early works used the Bayesian Optimization (Swersky et al., 2013) which is not the trending method after the NAS come up with in 2017. We focus on the works from 2017. The traditional method is the reinforcement learning with RNN controller. However, the critical defect of this strategy is that the efficiency is extremely low and the variance of the estimate is sometimes not appropriate for computation. The NAS algorithms in about 2017 are mostly of this type. In the paper by Google, Large-Scale Evolution of Image Classifiers, the method of aging evolution algorithm was adopted for training (Real et al., 2019). The results show that evolution can obtain results faster with the same hardware, especially at the earlier stages of the search. The recently study is concentrated on the method based on gradient. The DARTS raised the idea of expressing the network as a continuously distribution which leaded to the differentiable architecture search (Liu et al., 2018). It expressed the network by a matrix combined of the connection of the nodes and the activation functions. More specifically, it takes a convex combination from a group of operations $\{o_1, o_2, \cdots, o_m\}$. For instance, if giving a layer an input x, then the output will be as below.

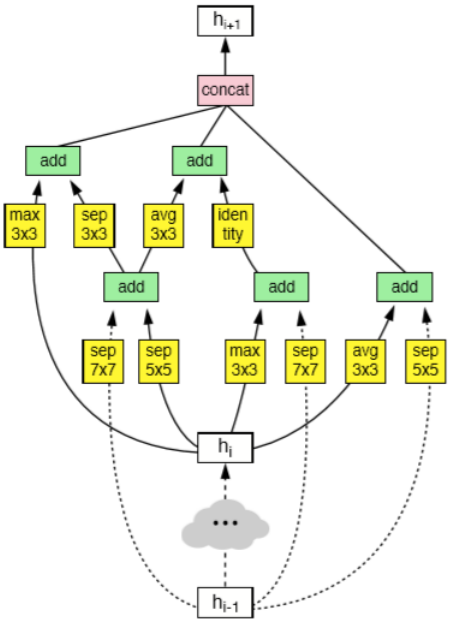

Figure 13: The Reduction Cell of NASNet-A (Zoph et al., 2017).

$$y = \sum_{i=1}^{m} \lambda_i o_i(x), \lambda_i \geq 0, \sum_{i=1}^{m} \lambda_i = 1 \tag{4}$$

The coefficients λ_i are efficient to parameterize the neural architecture. After the gradient descent optimization, we can update each layer with $i = argmax_i \lambda_i$ and finally get the discrete architecture. Simultaneously, another method based on encode-decode frame are raised by Microsoft and USTC in the paper of Neural Architecture Optimization (Luo et al., 2018). The 5 main types of NAS are shown in the figure 14.

The last aspect is the improvement of performance evaluation. The weight sharing idea used by ENAS, DARTS and NAO is one of the improvements towards this aspect. Nowadays, the prevailed method is one-shot searching structure. The method is featured by searching in a large searching space and alternatively training the parameters of networks and modules to find a best one. To solve the problem of the slow convergence of the reward, the SNAS designed by Sense Time made a progress by using gradient to optimize the objective function directly (Xie et al., 2018). ProxylessNAS (Cai et al., 2018) and Single Path One-Shot NAS (Guo et al., 2019) are delivered to solve another problem of the too much occupied storage of GPUs.

6.2 The Difference Made by NAS and Some Open Problems

The changes NAS can bring to us are the convenience of using the most advanced artificial intelligence technology. For some companies, they are just applying artificial intelligence on some relatively simple works. The convenience can spare the time of this companies for some

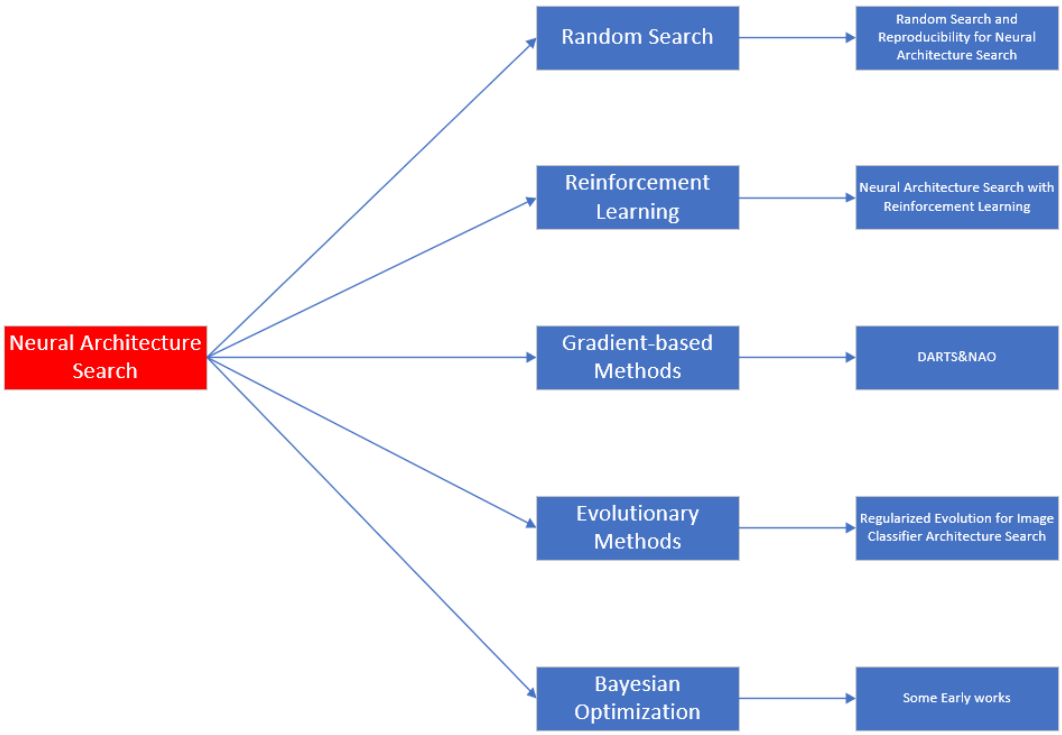

Figure 14: Five Types of NAS.

other things such as the designing of their products. Meanwhile, product iteration speed will noticeable increase so the technology used will no longer be the core competitiveness. The experience of the customers and the creativity will be the most valued. The popularization of NAS means that everyone can utilize the power of artificial intelligence with nearly no prerequisite knowledge. Obviously, the high cost of Cloud AutoML (provided by Google with 20 dollars per hour) and the low performance of the open source projects (mostly based on algorithm out of date such as Bayesian Optimization and ENAS) cannot reach the final purpose of NAS, with little access to or with low efficiency. Hence, in order to further improve this feature, there will be much improvement space for the artificial intelligence suppliers. They need to solve piles of big open problems in order to make the most cutting-edge AI technology available to everyone (Seif, 2018a).

In conclusion, NAS is developing towards the targets of multitasking and wide applicability. In fact, there are still some big problems in this area. The most tough one is the applicability width. Actually, NAS is mostly used in industries like image recognition and some recommendation system but in some other areas like NLP, it has not played a significant role. Recently, Google has done some work on it about the evolved transformer (So et al., 2019). Another possible improvement is the creativity of the NAS (Seif, 2018b). NAS needs a tremendous training set of marked data, which needs large amount of money and time. If the NAS can automatically generate some inspiration of the designing of the architecture (such as the residual connection), NAS can eliminate more work done by

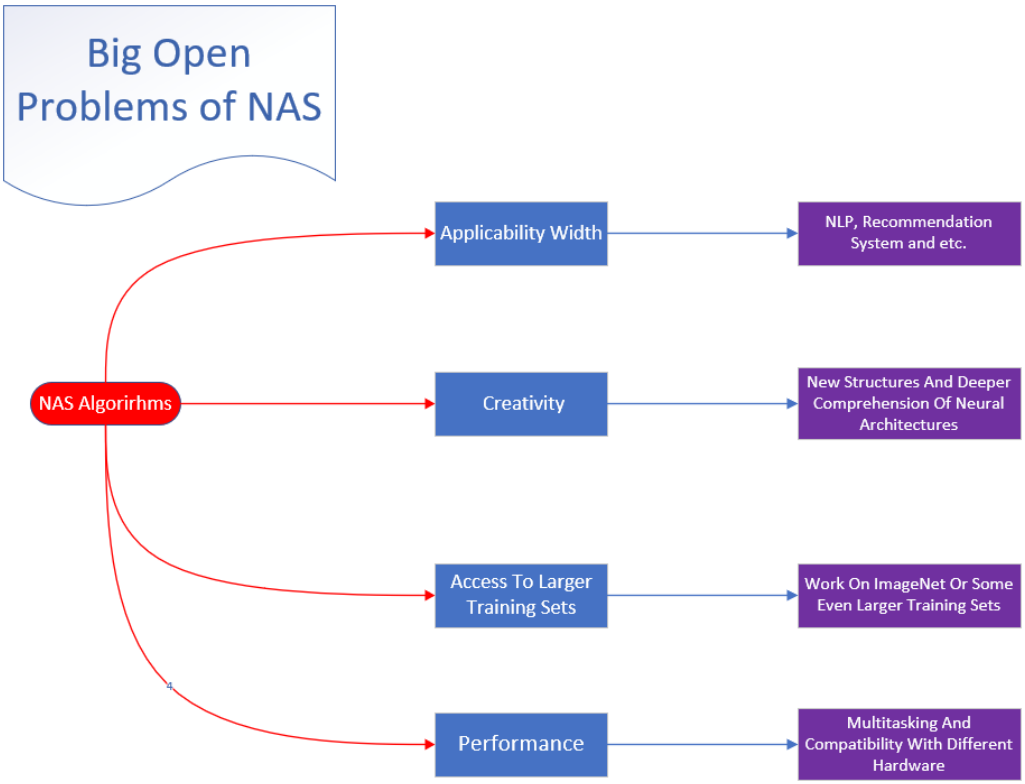

Figure 15: Big Open Problems of NAS.

human beings. Also, it still needs a lot of time to search for a neural architecture as the found network may not have the higher accuracy than the manual methods. We can feel this when we are using AutoKeras, it took me nearly half a day to obtain a model with nothing special. Finally, the most critical possibility is the wider range of the training set, which means that we can use NAS to train the neural architecture directly on the whole database instead of in a small training set. This requires high performance requirements so this object needs the development of the NAS and seemingly cannot be realized in a relatively short period of time. It means that in the future, NAS still needs to be improved in the three aspects of searching space, searching strategy and performance evaluation that mentioned above. In spite of these obstacles, NAS will absolutely be the final target of machine learning, with no manually operations.

References

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, July 2017. doi: 10.1109/CVPR.2017.195.

Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *ArXiv*, abs/1712.03351, 2017.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf`.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7036–7045, 2019.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, June 2018. doi: 10.1109/CVPR.2018.00745.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

FF Li and J Li. Cloud automl: Making ai accessible to every business. *Internet: https://www. blog. google/topics/google-cloud/cloud-automl-making-ai-accessible-everybusiness*, 2018.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Feifei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *ArXiv*, abs/1712.00559, 2017.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ArXiv*, abs/1802.03268, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

George Seif. Google's automl will change how businesses use machine learning. *Internet: https://towardsdatascience.com/googles-automl-will-change-how-businesses-use-machine-learning-c7d72257aba9*, 2018a.

George Seif. Everything you need to know about automl and neural architecture search. *Internet: https://towardsdatascience.com/everything-you-need-to-know-about-automl-and-neural-architecture-search-8db1863682bf*, 2018b.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

David R So, Chen Liang, and Quoc V Le. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019.

Kevin Swersky, Jasper Snoek, and Ryan P. Adams. Multi-task bayesian optimization. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2004–2012, USA, 2013. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=2999792.2999836`.

Favio Vzquez. Auto-keras, or how you can create a deep learning model in 4 lines of code. *Internet: https://towardsdatascience.com/auto-keras-or-how-you-can-create-a-deep-learning-model-in-4-lines-of-code-b2ba448ccf5e*, 2018.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2017.