# SMART: Self-organized Multitask Accelerate Resilient Learning

**Blind Review Version**


## Abstract

In this work, we propose a self-organized multitask accelerate resilient learning paradigm through evolution, named SMART, by constructing and organizing neuron assemblies into module-level building blocks. *Exploitation* and *Exploration* are the two executive phases of SMART, the former one, termed *self-organization*, is striving to satisfy required performance by leveraging existed modules. In contrast, the latter can change the number of modules by evolution to adjust the expressive ability of the network. The network is initialized with only one convolutional and one output module. By keeping learning different tasks continually, the network architecture evolves progressively. Finally, a specific-to-general combinatorial connectivity pattern within the neural network could be achieved. Furthermore, after learning a certain number of tasks, we can obtain a satisfying performance when encountering a new task only by *self-organization*. Experiments demonstrate that SMART dramatically increases the specific-to-general learning ability of networks. After training our network with five randomly selected MNIST binary classification tasks, *self-organization* boosts the transfer ability from 28.9% to 97% accuracy on unforeseen tasks. When transfer between different task sets (e.g. from CIFAR10, SVHN to CIFAR100), SMART increased 11.0%-17.2% transfer ability.


## Introduction

Fast, incremental, and continuous learning are the essential ingredients that enable intelligent systems to adapt to the changing world (Parisi et al. 2018)(Shin et al. 2017). However, the fundamental architectural abstraction of intelligence that allows the human brain to deal with various possible patterns and knowledge is still the biggest mystery. Nevertheless, this challenge does not prevent but encourages people to explore potential solutions to this ultimate goal in the artificial neural network research area (Xu, Tao, and Xu 2013)(Kirkpatrick et al. 2016). Recently, multitask learning (MTL) has been a promising learning paradigm in machine learning. An ideal MTL model does not need to memorize specialized parameters for each task. Instead, it can help improve the generalization performance of all tasks by leveraging useful information contained in multiple related tasks. (Zhang and Yang 2017).



On the other hand, the deep neural network has scaled up to deal with more complex tasks with hundreds of hyper-parameters and sophisticated architectures. How to design such systems has emerged as a major challenge in front of us (Jaderberg et al. 2017), (Szegedy et al. 2015a), (Szegedy et al. 2015b), (Baker et al. 2016). Manually designing network architecture through experimentation will become very difficult. Recently, some evolutionary optimization schemes of neural networks have been proposed trying to tackle this problem (Miikkulainen et al. 2017)(Liang, Meyerson, and Miikkulainen 2018)(Fernando et al. 2017),(Chen et al. 2018). In these papers, researchers are responsible for the high-level design and evolutionary strategies and leave the optimization systems to leverage the increasing available computational power to figure out the appropriate network architecture. However, leveraging the current components to promote learning capability for new tasks is sometimes overlooked.

This work touches on the field of MTL and evolving neural networks. We propose a self-organized multitask resilient learning paradigm through evolution named SMART. The underlying principle of SMART is that if new tasks are partially similar to previously encountered tasks, then leveraging past knowledge can increase the learning rate and capability for the new ones. We construct and organize neuron assemblies into module-level building blocks, and different executive phases are conducted on different granularities. Specifically, *Exploitation* and *Exploration* are the two executive phases of SMART. *Exploitation* consists of probing a limited region of the search space—current existed modules and updating learning parameters among modules. We termed the *Exploitation phase—"self-organization"*. *Exploration* phase, on the other hand, consists of probing a much larger portion of the search space to meet the required performance by incrementally adjusting the number of learnable modules and updating parameters both inside and among modules through evolution.

When encountering a new task, the *self-organization* (*Exploitation phase*) is launched firstly, trying to achieve the required performance by utilizing already existing modules. Suppose the network at its current evolutionary strategy can not reach the expectation. In that case, the *Exploration phase* will be executed by adding new learnable modules to enrich its expressive ability through fitness approximation and gra-

dient descent. The specific-to-general combinatorial connectivity pattern within the network could be shaped by learning more tasks with two phases conducted alternatively. Experiments on binary and multi-class classification tasks demonstrate that, after a certain number of learning tasks, a relatively satisfied performance can be achieved by simply *self-organizing* the currently existing modules.

In a nutshell, the novelty of the SMART is threefold:

- **Enhanced Modules.** Network performance highly relies on the sufficient expressive ability of the basic building block—modules. Two powerful modules named Res-Fire and Dimension Reduction modules (Zheng et al. 2018) are introduced into SMART to acquire noticeable improvement when transferred among different tasks.
- **Self-organization.** The exploitation of SMART is manipulated in the granularity of module-level through learnable scale parameters adjustment. The confined search space can achieve high-efficient specific-to-general learning ability when the network reaches a certain scale.
- **Evolutionary Algorithm.** The proposed evolutionary algorithm is the same as the traditional procedure of the evolutionary algorithm, containing *reproduction, mutation, recombination* and *selection*. To shorten the time-consuming process of the evolution phrase, we proposed rational evaluation criteria through experiments to balance the performance accuracy and convergence speed.

SMART is a step towards enabling deep MTL to realize 'specific-to-general' learning ability. We release our design in the TensorFlow version. The model and supplement materials are available at https://github.com/WindWing/SMART.

## Related Work

A selected number of previous papers touch MTL, and neural network design is reviewed before introducing the proposed SMART.

Traditional neural networks have brought great success in many fields(Simonyan and Zisserman 2014), (He et al. 2015)(Szegedy et al. 2016)(Xie et al. 2017). But the system performance highly relies on engineers to design a proper neural network architecture. This procedure is very time-consuming. There are some work demonstrate how evolution can be instrumental in advancing multitask network design(Fernando et al. 2017)(Meyerson and Miikkulainen 2017)(Liang, Meyerson, and Miikkulainen 2018)(Chen et al. 2018).

In order to transfer knowledge among tasks, Multitask learning(MTL)(Caruana 1998) has achieved satisfying performance in many area like computer vision(Bilen and Vedaldi 2017), speech recognization(Huang et al. 2015), natural language processing (NLP)(Dong et al. 2015)(Hashimoto et al. 2016), and reinforcement learning(Devin et al. 2017)(Jaderberg et al. 2016).

Large-Scale Evolution of Image Classification(Real et al. 2017) use evolutionary methods to finetune hyperparameters. The author set 1000 populations in the experiment and evolved it with 250 machines. By using the tournament selection method and random mutation, it achieves good results. However, the computational resources are unaffordable for most researchers. In PathNet(Fernando et al. 2017), Fernando uses an evolutionary algorithm to figure out which part of the network to be activated. To reduce the searching space of the evolutionary algorithm, PathNet uses modules to encapsulate a block of components and parameters, adopts a layered structure to arrange modules, and uses a sum operation to gather all the information from different modules before passing them to the next layer. In this way, the exploring space is sharply reduced when applying the evolutionary algorithm to investigate which subset of the modules is to be shared in the transfer phase. The structure of SMART is inspired by PathNet, which arranges resources in the module granularity.

Modularity is an efficient way to achieve transfer learning. In Meyerson's work (Meyerson and Miikkulainen 2017), their experiences revealed that permuted ordering of shared layers performances better than parallel ordering. Further, they demonstrated using scale parameters to approximate permuted order is a practical approach. To enhance the composability of the modules and improve the rate of modular reuse in the *self-organization* process, we also applied the same approach in SMART.

## SMART

The basic idea of SMART is that, considering there are total of $T$ tasks $\{t_1, t_2, ..., t_T\}$ need to be learned continually. Theoretically, there exists a set of different $F$ features $\{f_1, f_2, ..., f_F\}$. The combination of these features can handle all the $T$ tasks. In this way, if the network keeps learning new tasks, new modules can be employed by the evolutionary algorithm for new features. The already mastered feature set is marked as $\hat{F}$. After several iterative rounds of evolution and learning, the elements in set $\hat{F}$ keep increasing. After learning enough tasks, we can get a set $\hat{F}$, so the network is general enough to deal with unseen tasks $t_i$ even without further training. In the following pages, we first introduce the basic building blocks of SMART and then dig into the detailed implementation of our design.

### Building Blocks

Modularity is an effective and powerful approach to achieving component and parameter reuse and encapsulating subnetworks in the MTL paradigm (Fernando et al. 2017). The network's generalization ability highly relies on the sufficient expressive ability of the set of modules. For SMART, there are four basic building blocks, including convolutional modules, fully connected modules, and two newly proposed modules, named *Res-Fire (RF)* and *Dimension Reduction (DR)* modules, see Figure 1.

**Res-Fire Module:** *Fire module* is a two-layer structure that is introduced in the SqueezeNet (Iandola et al. 2016). The first layer is composed of one $1 \times 1$ convolution kernel and the second layer is composed of two different size of convolution kernels, one is $1 \times 1$ and the other is $3 \times 3$ ($padding = 1, stride = 1$). The output of two convolution

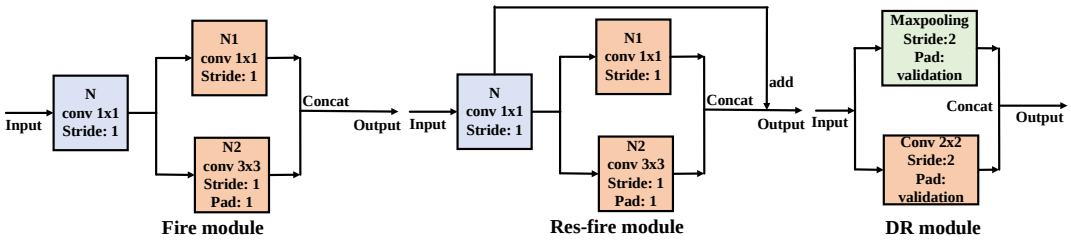


Figure 1: **Enhanced Modules.** *RF* module adds a bypass to the *Fire module*, and *DR* module combines the max-pooling with convolutional filters(Zheng et al. 2018).

kernels is contacted together. *RF* module is an enhanced version derived from *Fire module* with a residual link that adds input features to the output. Notice that the inputs and outputs should have the same channel number.

**Dimension Reduction Module:** Max-pooling is often used to reduce network size and prevent overfitting. Functionally, it can be regarded as a smoothing filter for the previous layer. Information irreversible and feature loss may arise if we only transmit the features through the max-pooling layer without data interaction before and after it. Therefore, *DR* module combines max-pooling with $2 \times 2$ convolutional filters($stride = 2$) to reduce network size while prevent irreversible information loss.

Experiments in Section  demonstrate that the transfer ability of the network is greatly improved after introducing these modules. Notice that SMART is not limited to the modules above. Other modules can be expanded and enrich the basic building blocks of SMART.

## Network Architecture

To characterize the continuous and multitask learning properties of SMART, two issues need to be addressed, 'what to share' and 'how to share.' 'What to share' asks which network components can be shared across tasks. Same as *PathNet* (Fernando et al. 2017), SMART is organized on the module granularity, neurons encapsulated in module-level are shared among different tasks. Intuitively, modules may be shared among different tasks with equal or different effectiveness, and this character should be reflected in the learnable parameters.

The structure of SMART draws inspiration from *PathNet*, see Figure2. *PathNet* applies a modular mechanism to achieve efficient components, and parameters reuse during transfer learning by freezing a sub-set of networks in the module granularity to avoid catastrophic forgetting. *PathNet* is a relatively redundant network comprising $L$ layers. Each layer contains $M$ modules, and up to $N$ distinct modules in one layer are permitted to be active simultaneously for a specific task. There are $F_M$ filters in each module. The outputs of activated modules are summed up before sending to the next layer to reduce the search space of the evolutionary algorithm. Notice that $L$, $M$, $N$, and $F_M$ are predetermined configuration parameters according to the complexity of different target tasks.

There are three main differences between SMART and *PathNet*: a) *PathNet* only applies convolutional and fully connected modules, we introduce enhanced modules (*RF* and *DR*) to improve the expressiveness of the network; b) we add learnable scale parameters $\alpha$ and $\beta$ associated with modules on the concentration and scatter phase between different layers to represent different effectiveness of features for different tasks, see Figure 2(b); c) The network structure of SMART is much more resilient, it is task-oriented rather than predefined, so the network is capable of expanding and shrinking according to different task requirements.

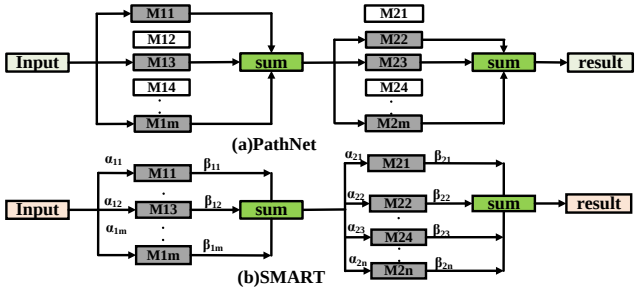


Figure 2: **Network Architecture.** *PathNet* is a redundant network. The number of layers ($L$) allowed to be activated and the total number of modules ($N$ and $M$) on each layer are predetermined. In (a), the dark-gray rectangles are the activated modules, while the white rectangles are inactivated modules. For SMART, it contains one convolutional module (Input rectangle) and one output module (result rectangle) initially. The proposed evolution algorithm automatically and progressively constructs the growth of the network structure to satisfy the task requirement.

## Evolutionary Algorithm

Design neural network architecture for multitask learning is a compelling domain for evolutionary optimization (Miikkulainen et al. 2017)(Liang, Meyerson, and Miikkulainen 2018). Manipulating building blocks in the module granularity reduces the exploring space when applying the evolutionary algorithm to construct the appropriate architecture.

When encountering a new task, the purpose of *self-organization* is to achieve the required performance by exploiting the available modules. The network initially contains one convolutional module and one output module. The network initially keeps adding new modules to satisfy new tasks. Afterward, during transfer learning, we freeze the parameters inside the modules and manipulate them with the scale parameters to examine whether previously stud-

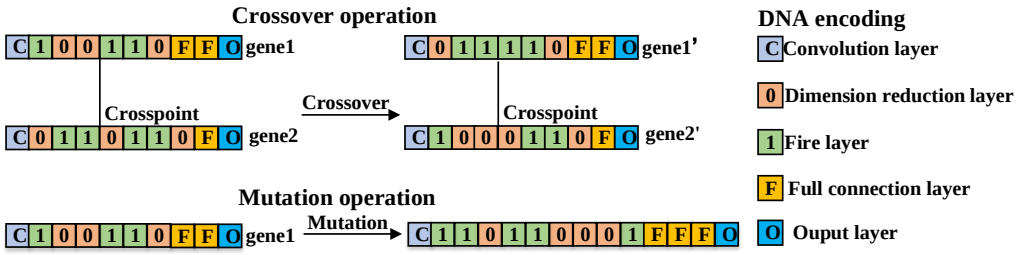


Figure 3: **Crossover and Mutation.** For all the experiments, the mutation rate is 0.2. The network begins with a convolution layer and ends with an output layer associated with several fully connected layers.

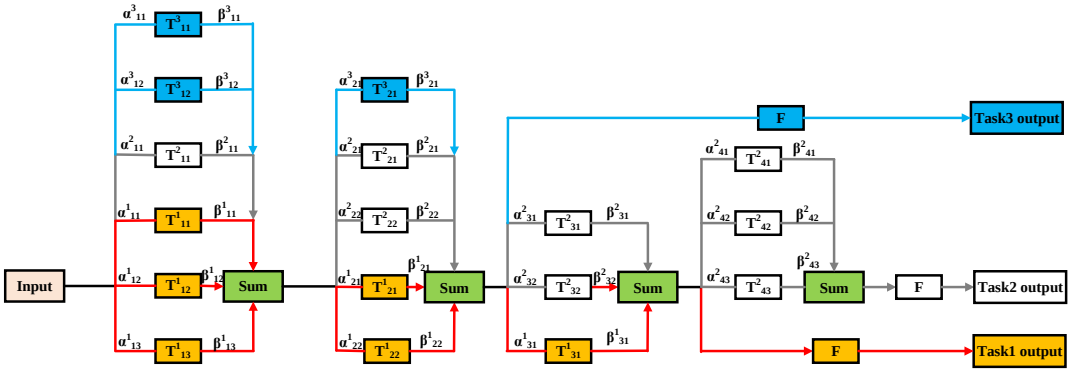


Figure 4: **Self-organization.** All the modules are shared with various tasks. $T_{lk}^{t}$ means it is generated as the $k$-th module through evolution when training task $i$ at layer $l$. As observed, the effective networks are different for different complex tasks. Self-organization harmonizes these modules for a specific task with learnable $\alpha$ and $\beta$ through gradient descent.

ied modules and their combination can handle the new task. If not, an evolutionary algorithm will be conducted to add more learnable modules to enrich the representation ability of the network. Then follow-up tasks are performed, and the network's generalization ability keeps increasing.

**Evolution Implementation:** Networks are trained for a given number of generations, and performance on the target task is considered a fitness criterion to select capable candidates in the population.

Candidates are trained for a certain number of steps. Afterward, the candidates are evaluated and sorted by their performance. The best two candidates are selected for further mutation and recombination to produce offspring. To keep the size of the population fixed, the two recombined offspring will replace the worst two parents. Notice that evolution is biased towards discovering fast learners instead of top performers due to limited computation power and time constraints. A trade-off should be required to balance performance and learning efficiency.

**Fitness Criterion:** The final accuracy of the network is the most suitable fitness criterion for evaluating the candidates. However, waiting for the network to be converged is very time-consuming. In practice, we must choose a proper fitness criterion to balance the execution time and accuracy.

Generally, there are four indicators: a) *Max accuracy*. We choose the best candidate with the highest accuracy at $S$ steps, b) *Mean accuracy*. We are selecting the candidate with the best average training accuracy over the last $S'$ steps, c) *Slope of the learning curve*. We choose the one with the most prominent slope of accuracy across the last $S''$ steps, d) *Fitting*. Using a curve-fitting method to predict the tendency of the learning curve and the best one is chosen. According to our experiments, we selected c) as our applied fitness criterion. The experiment results of different indicators are shown in Figure 6.

We design four experiments to find the best indicators. Networks evolved with four different indicators. In each experiment based on CIFAR-10. We stop training the network at 6 generations ($batchsize = 256$) for each competitor. Six competitors per generation, with a total evolution of 6 generations. The final accuracy of evolved networks are 76.5%(a), 76.4%(b), 78.5%(b), 73.0%(d). The Slope of the learning curve gets the best result, so we choose this fitness criterion in the following experiments.

After choosing the proper fitness criterion, we introduced the following *mutation* and *recombination* strategies.

**Mutation Strategy:**

$$len(seq') = len(seq) \times (1 - m_r) + m_r \times random(*) \quad (1)$$

$$\text{seq}[i] = \begin{cases} \text{seg}[i], & \text{prob } 1 - m_r \\ !\,\text{seq}[i], & \text{prob } m_r \end{cases} \quad (2)$$

**Crossover Strategy:**

$$n = random(0, length_{max}) \quad (3)$$

$$\begin{aligned} seq_1' &= concatenate(seq_1[:n], seq_2[n:]) \\ seq_2' &= concatenate(seq_2[:n], seq_1[n:]) \end{aligned} \quad (4)$$

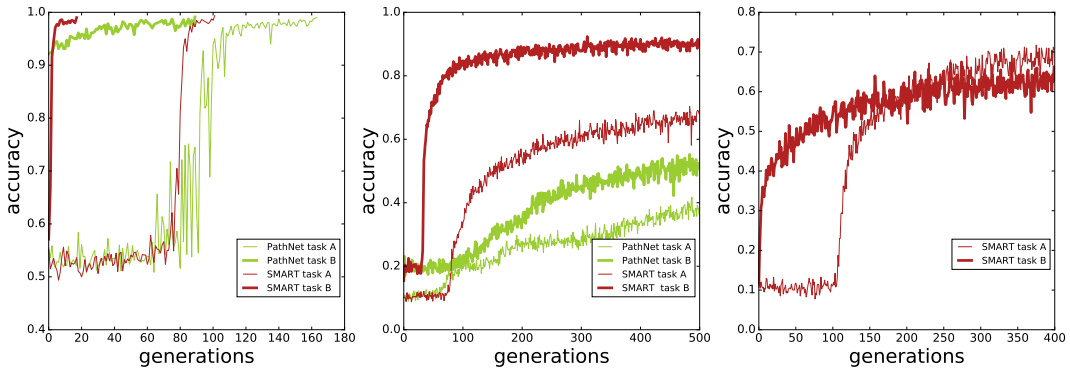


Figure 5: **Transfer Learning.** (a) MNIST binary classification tasks, *PathNet* with $L = 4$ layers, $M = 10$ modules and only up to $N = 3$ modules can be selected in each layer, $F_M = 20$ filters in each module. (b) CIFAR-10-SVHN tasks, $L = 4$, $M = 20$, $N = 5$ and $F_M = 20$ for *PathNet*. (c) Mini-ImageNet classification task, $L = 5$, $M = 30$, $N = 7$ and $F_M = 40$ for *PathNet*.

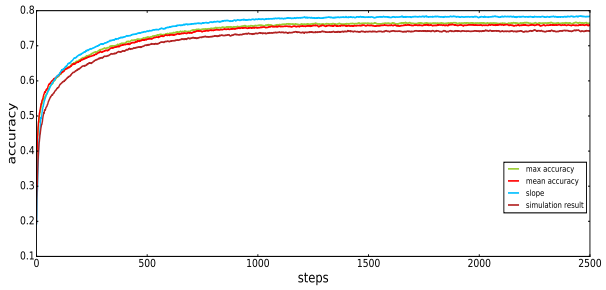


Figure 6: **Fitness Criterions Comparison on CIFAR-10** For the experiments, we terminate training the network at 3000 steps for CIFAR-10. The batch size is 25. We observed that the one with the biggest slope across previous steps gets the highest accuracy.

In the formula, $seq$ is a binary sequence like $(010010)$. $seq'$ is the updated sequence after mutation or crossover. The bit-length of it generally represents the number of layers($L$), module number in layer $L$ ($M_L$), as well as filter number in each module ($F_M$). $m_r$ is the mutation rate, which is a relatively small floating point number between 0 and 1. For all the following experiments, we set $m_r = 0.2$. $n$ is a number generated each time randomly with the uniform distribution between the defined minimal and maximal value of $seq$. $len(*)$ is a function that returns the bit-length of the operand. "!" is the bitwise operation, for example $!(0101b) = 1010b$. The mutation and crossover are conducted at layer, module, and filter levels. Notice that since $seq$ generally represents the $L$, $M_L$, and $F_M$, these strategy is well adapted to all the different levels. See Figure3.

In our evolution strategy, SMART strives to find the appropriate structure instead of a sizeable redundant structure. The slope of candidates after $S$ steps is considered fitness. Due to the limited training steps, the final evolution result will be a network with high accuracy and fast convergence speed with a reasonable scale. Eliminating the redundant modules by weight-based soft-pruning according to the $\alpha$ value can help prevent the network's endless expansion.

## Self-organization for Multitask Learning

Firstly, we need to answer whether the current evolved neural network is sufficient or not to handle the new tasks by enforcing the utilization of existing modules and their combination through manipulating the $\alpha$ and $\beta$ values. Classification accuracy is measured as the performance metric. As previously mentioned, evolution-based exploration is launched to learn the new features if the current $\hat{F}$ is not adequate to handle the new task $t_{new}$. The $\hat{F}$ is keeping accumulated and the ultimate goal an exhaustive set. That is to say that the neural network can deal with other not-encountered tasks through simply *self-organization*, see Figure4.

# Experiment

Five different datasets are selected for the multitask experiments, including MNIST, CIFAR-10, CIFAR-100, SVHN, and MiniImageNet. We use *PathNet* as the baseline for all the experiments.

## Classification Performance

This experiment shows how much benefit SMART can bring from enriched modules compared with *PathNet*. For MNIST binary classification, we randomly select two numbers in the dataset to form two binary classification tasks, A and B. Firstly, we train task A on the network to research 99% accuracy. Then test the classification accuracy on task B. For CIFAR-SVHN multi-class classification, we train CIFAR-10 on the network for 500 generations and test the classification accuracy on the SVHN dataset. For ImageNet, we randomly choose 10 classes from the dataset to form a 10-classes classification problem named MiniImageNet.

The comparison results of MNIST are shown in Figure 5(a), the training generation of SMART to reach 99% accuracy only takes 118 generations on task A and 17 generations on task B, whereas *PathNet* takes 163 generations for task A and 89 generations for task B. For CIFAR-SVHN tasks, see Figure 5(b), SMART achieves 69.1% and 91.5% accuracy on CIFAR (task A) and SVHN (task B) at 500 generations, respectively, whereas *PathNet* achieves 36.3% and 51.4% accuracy on CIFAR (task A) and SVHN (task B). SMART shows 1.9x and 1.8x more accuracy and achieves

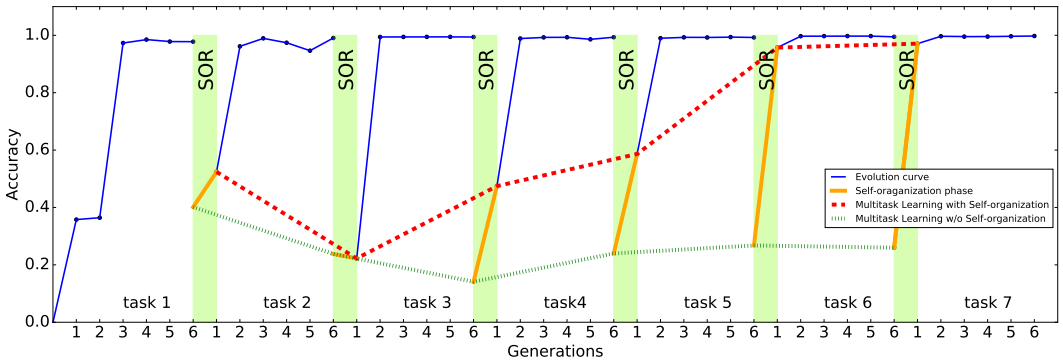


Figure 7: **Self-organization for MNIST dataset.** After learning five tasks, we can construct a well-performed network only by *self-organization* of the modules learned in previous tasks. By comparing the *self-organization* curve (red dotted line) with the green curve without *self-organization*, we can find that after several task evolutions, *self-organization* demonstrates an effective approach to achieving task transfer only by adjusting the scale parameters among modules without adding new modules or changing the parameters inside the modules. The *self-organization* operation helps boost their performance on not encountered tasks. It brings about 12.3%, -1.5%, 33.3%, 34.7%, 68.9% and 71.1% higher accuracy, when transfer from task1 to task7 sequentially.

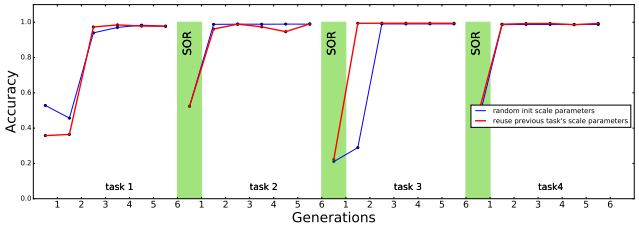


Figure 8: **Initalization Method.** Comparison two different initialization method for *self-organization* between two continuous tasks on MNIST Dataset. The network is evolved 6 generations and contains 6 candidates in each generation. Randomly initializing the scale parameters achieves slightly better performance. Sometimes it is difficult to distinguish versus freezing the scale parameter of the previous one.

faster convergence than *PathNet* on the respective tasks. For MiniImageNet tasks, see Figure 5(c), SMART achieves 70.8% and 66.3% accuracy on task A and task B. However, we fail to converge if we follow the *PathNet's* method at 400 generations. Obviously, SMART exhibits a strong learning ability than *Path*.

## Self-organization for Multitask Learning

**Binary Classification Tasks on MNIST Datasets:** Binary classification problems derived from the MNIST handwritten dataset to distinguish between two distinct randomly selected digits. In the experiments, we stop training the network at 1000 generations (batch size=256) for each competitor. There are six competitors in each generation, with a total evolution of 6 generations. The slope is used as the fitness criterion to evaluate the competitors. Figure 7 shows sequentially learned 7 binary classification tasks, the multitask learning capability is greatly improved by *self-orgnization*.

There is another question comes after that, whether or not should the next task inherit the scale parameters ($\alpha$ and $\beta$) of the previous one. According to our experiments, see Figure 8, we observed randomly reinitializing the scale parameters achieves a slightly better performance versus freezing these scale parameters between two continuous tasks. Therefore, there is no need to memorize the previous scale parameters.

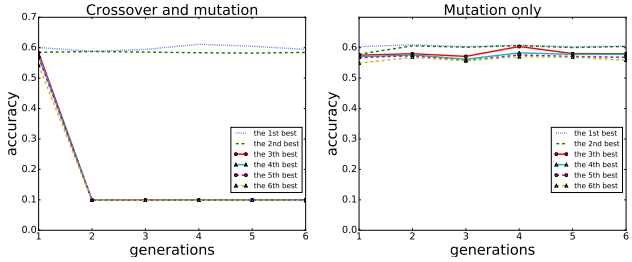


Figure 9: **Crossover and Mutation vs. Mutation Only** When we use crossover and mutation together, many competitors lose their ability to learn.

## Crossover and Mutation vs. Mutation Only

We investigate whether the crossover strategy will benefit the network performance. We conduct the following experiment. Randomly generating a seed network to evolve for CIFAR-10, and the training is terminated at 3000 steps (batch size is 256) for each competitor. Six competitors for a generation, and with a total evolution of 6 generations, We ranked each generation of competitors and eventually got Firgure.9. It can be observed that crossover might cause the network performance fluctuation, and mutation only sustains equivalent performance. Therefore, we have omitted the crossover procedure in the evolutionary algorithm.

**CIFAR10, SVHN and CIFAR100** SVHN is a real-world digit recognition dataset consisting of photos of house numbers in Google Street View images. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with

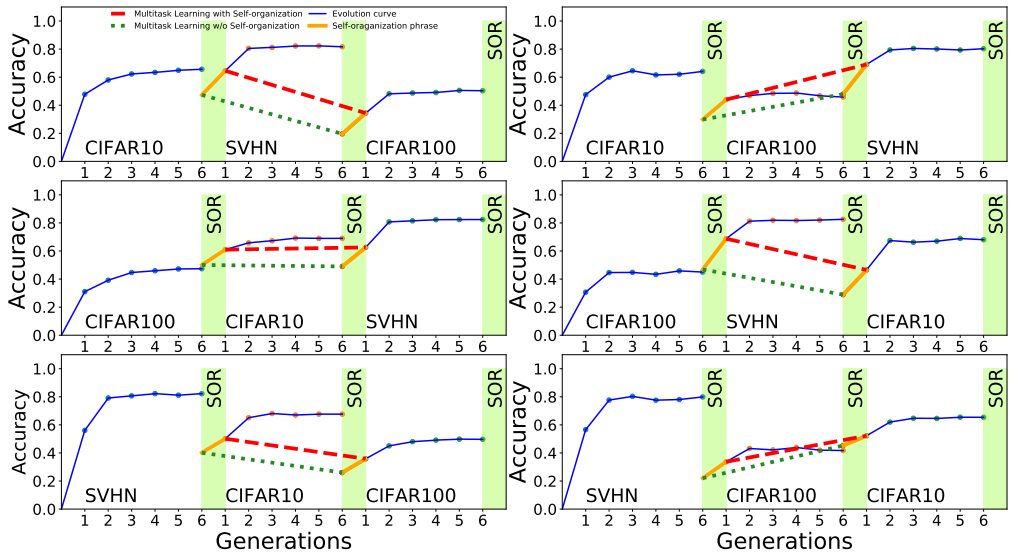


Figure 10: **Self-organization.** Self-organization for multitask learning on CIFAR-10, SVHN, and CIFAR-100 dataset.

6000 images per class. There are 50000 training images and 10000 test images. For CIFAR-100, the total number of images and image size is same as in CIFAR-10, but it includes 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

For the multi-classification experiments, we perform two 10-classification tasks on both the CIFAR-10 and SVHN, and a 20-classification task on CIFAR-100. Also, we stop training the network at 6 generations ($batchsize = 256$) for each competitor. Six competitors per generation, with a total evolution of 6 generations, the slope is still the fitness criterion for network performance evaluation. The *self-organization* helps boosting their transfer capability, brings about 17.2% and 15.8% higher accuracy, when transfers from CIFAR-10 to SVHN, and then from SVHN to CIFAR-100. The convergence accuracy is 79%, 88% and 60% at 6 generations, see the top-left diagram in Figure 10.

Then we tested all 6 combinations of the order of different tasks. *Self-organization* scheme increases the transfer performance. For example, brings about 11.0% and 13.6% higher accuracy, when transfer from CIFAR-100 to CIFAR-10, and then from CIFAR-10 to SVHN. The convergent accuracy is 59%, 75%, and 87% at six generations. Please refer to the mid-left diagram in Figure 10.

The total amount of computation can represent the scale of the network to a certain extent, Figure 11 demonstrates that the SMART is task-oriented rather than ever-expanding. as the computational curve rise and falls according to different task complexity.

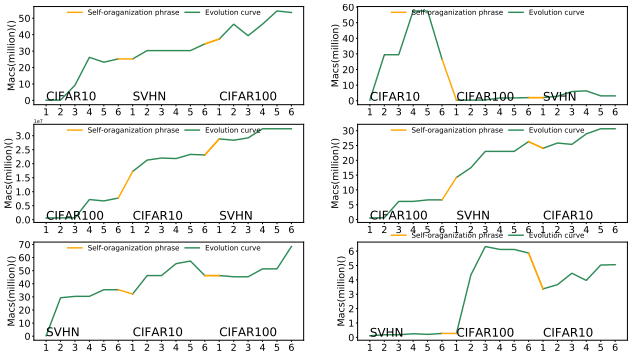


Figure 11: **Computational Cost.** Changes in the amount of computation for multitask learning on CIFAR-10, SVHN, and CIFAR-100 dataset.

## Conclusion and Future Work

It is well known that manually building dedicated deep neural network structures for multi-tasking learning is infeasible. Our studies on multitask learning through self-organization conducted in relatively small-scale applications with a limited variety are introduced. Different tasks learn to use the same set of modules with different weights. This phenomenon can help shed light on how tasks are related.

SMART contains a good expressive set of modules. The optimization in SMART is much easier than other deep learning approaches since it operates in module-level granularity, involving significantly reduced parameters. Over generations, modules concerning new features are added to the network incrementally. The model can potentially achieve specific-to-general multitask learning capability through evolution. Meanwhile, weight-based soft-pruning can be applied to eliminate the redundant modules and prevent the network architecture's endless expansion. SMART helps identify a set of generalizable modules assembled differently for different tasks. Still, several issues must be addressed, and follow-up work includes extending it to many applications that lend themselves to multitasking approaches like semantic segmentation, object tracking, and 3D reconstruction.

## References


Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2016. Designing neural network architectures using reinforcement learning.

Bilen, H., and Vedaldi, A. 2017. Universal representations:the missing link between faces, text, planktons, and cat breeds.

Caruana, R. 1998. *Multitask learning*. Springer US.

Chen, B.; Wu, H.; Mo, W.; Chattopadhyay, I.; and Lipson, H. 2018. Autostacker: A compositional evolutionary learning system.

Devin, C.; Gupta, A.; Darrell, T.; Abbeel, P.; and Levine, S. 2017. Learning modular neural network policies for multi-task and multi-robot transfer. In *IEEE International Conference on Robotics and Automation*, 2169–2176.

Dong, D.; Wu, H.; He, W.; Yu, D.; and Wang, H. 2015. Multi-task learning for multiple language translation. In *Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, 1723–1732.

Fernando, C.; Banarse, D.; Blundell, C.; Zwols, Y.; Ha, D.; Rusu, A. A.; Pritzel, A.; and Wierstra, D. 2017. Pathnet: Evolution channels gradient descent in super neural networks.

Hashimoto, K.; Xiong, C.; Tsuruoka, Y.; and Socher, R. 2016. A joint many-task model: Growing a neural network for multiple nlp tasks. 1923–1933.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. 770–778.

Huang, Z.; Li, J.; Siniscalchi, S. M.; Chen, I.; Wu, J.; and Lee, C. H. 2015. Rapid adaptation for deep neural networks through multi-task learning.

Iandola, F. N.; Moskewicz, M. W.; Ashraf, K.; Han, S.; Dally, W. J.; and Keutzer, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks.

Jaderberg, M.; Dalibard, V.; Osindero, S.; Czarnecki, W. M.; Donahue, J.; Razavi, A.; Vinyals, O.; Green, T.; Dunning, I.; and Simonyan, K. 2017. Population based training of neural networks.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; and Grabskabarwinska, A. 2016. Overcoming catastrophic forgetting in neural networks. *Proc Natl Acad Sci U S A* 114(13):3521–3526.

Liang, J.; Meyerson, E.; and Miikkulainen, R. 2018. Evolutionary architecture search for deep multitask networks.

Meyerson, E., and Miikkulainen, R. 2017. Beyond shared hierarchies: Deep multitask learning through soft layer ordering.

Miikkulainen, R.; Liang, J.; Meyerson, E.; Rawal, A.; Fink, D.; Francon, O.; Raju, B.; Shahrzad, H.; Navruzyan, A.; and Duffy, N. 2017. Evolving deep neural networks.

Parisi, G. I.; Kemker, R.; Part, J. L.; Kanan, C.; and Wermter, S. 2018. Continual lifelong learning with neural networks: A review.

Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y. L.; Tan, J.; Le, Q.; and Kurakin, A. 2017. Large-scale evolution of image classifiers.

Shin, H.; Lee, J. K.; Kim, J.; and Kim, J. 2017. Continual learning with deep generative replay.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *Computer Science*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015a. Going deeper with convolutions. In *Computer Vision and Pattern Recognition*, 1–9.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2015b. Rethinking the inception architecture for computer vision. *Computer Science* 2818–2826.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. 2016. Inception-v4, inception-resnet and the impact of residual connections on learning.

Xie, S.; Girshick, R.; Dollar, P.; Tu, Z.; and He, K. 2017. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 5987–5995.

Xu, C.; Tao, D.; and Xu, C. 2013. A survey on multi-view learning. *arXiv: Learning*.

Zhang, Y., and Yang, Q. 2017. A survey on multi-task learning.

Zheng, Z.; Wei, Y.; Zhao, Z.; Wu, X.; Li, Z.; Ren, P. 2018. Multitask Learning With Enhanced Modules. In IEEE International Conference on Digital Signal Processing.