M-NAS: Meta Neural Architecture Search

Jiaxing Wang,[1,2*] **Jiaxiang Wu,**[3] **Haoli Bai,**[4] **Jian Cheng**[1,2,5†]
[1]NLPR, Institute of Automation, Chinese Academy of Sciences,
[2]University of Chinese Academy of Sciences,
[3]Tencent AI Lab, [4]The Chinese University of Hong Kong,
[5]Center for Excellence in Brain Science and Intelligence Technology, CAS
{jiaxing.wang, jcheng}@nlpr.ia.ac.cn, jonathanwu@tencent.com, hlbai@cse.cuhk.edu.hk

Abstract

Neural Architecture Search (NAS) has recently outperformed hand-crafted networks in various areas. However, most prevalent NAS methods only focus on a pre-defined task. For a previously unseen task, the architecture is either searched from scratch, which is inefficient, or transferred from the one obtained on some other task, which might be sub-optimal. In this paper, we investigate a previously unexplored problem: whether a universal NAS method exists, such that task-aware architectures can be effectively generated? Towards this problem, we propose Meta Neural Architecture Search (M-NAS). To obtain task-specific architectures, M-NAS adopts a task-aware architecture controller for child model generation. Since optimal weights for different tasks and architectures span diversely, we resort to meta-learning, and learn meta-weights that efficiently adapt to a new task on the corresponding architecture with only several gradient descent steps. Experimental results demonstrate the superiority of M-NAS against a number of competitive baselines on both toy regression and few shot classification problems.

Introduction

Neural architecture search (NAS) has made it possible to automatically find optimal deep neural network architectures, which conventionally requires hand-crafted heuristics, domain expertise, and repetitive trials. Existing studies show that NAS can better explore the large design space and the architectures found can outperform human-designed competitors in image classification, language modeling (Zoph and Le 2016; Pham et al. 2018; Liu, Simonyan, and Yang 2018), object detection (Zoph et al. 2018) and semantic segmentation (Chen et al. 2018; Liu et al. 2019).

Despite the success, most prevalent NAS methods can only enumerate the search space for a pre-defined task. However, in reality, heterogeneous tasks of various kinds need diverse network architectures, and it is rarely investigated whether a universal NAS method can be found to cope with the various properties of these tasks, and generate task-aware neural architectures. Nevertheless, it is non-trivial to establish such a universal neural architecture search method, which is challenged by the following two issues:

1) How can we find an architecture controller, such that task-aware candidate models can be effectively generated? Prevalent NAS methods utilize Recurrent Neural Networks (RNNs) (Zoph and Le 2016; Pham et al. 2018), random variables (Bender et al. 2018) or trainable architecture parameters (Liu, Simonyan, and Yang 2018) as network controllers to generate the child models. However, no external information about tasks is considered (Elsken, Metzen, and Hutter 2019), making it unable to generate task-aware child models. The design of discriminative controller to different tasks is still left unexplored.

2) How can we efficiently estimate optimal weights associated with a candidate architecture, so that the child model can be reliably evaluated? Training the model to exact convergence (Zoph and Le 2016) incurs unaffordable computational costs. Previously NAS methods only focus on a single fixed task, and adopt parameter sharing (Pham et al. 2018; Liu, Simonyan, and Yang 2018; Bender et al. 2018) to reduce the computational cost of estimating optimal weights associated with different architectures. However, when there are different tasks, parameter sharing deteriorates the model capacity, as the assumption that the same parameter value is shared across different tasks could be too strong. Therefore, it is desirable to develop a novel weight estimation scheme that can efficiently estimate the optimal weights of the candidate architecture on a new task.

To address the above issues, we propose meta neural architecture search (M-NAS). Specifically, a task-conditional controller is developed, which embeds a task and uses the task representation to generates task-aware candidate architectures. To efficiently estimate optimal weights of candidate architectures on different tasks, we resort to model-agnostic meta-learning (MAML) (Finn, Abbeel, and Levine 2017), which learns meta-weights as shared initialization for all the tasks from where the model can adapt to new tasks with a few gradient descent steps. By maintaining meta-parameters shared across different architectures, rather than the model weights itself, the proposed optimal weights estimation scheme avoids model capacity deterioration and

[*]This work is done when Jiaxing Wang works as an intern in Tencent AI Lab

[†]Corresponding Author

achieves efficient approximation because meta-parameters contain the knowledge of previous searches on other tasks.

Finally, with the task-aware architecture controller, candidate architecture suitable for a specific task can be effectively generated. The candidate architecture can then be efficiently evaluated with a few gradient descent steps starting from the meta-parameters, avoiding repetitive computational intense training. Extensive experiments are conducted on both toy examples and real-world datasets. The results show that our proposed M-NAS efficiently discover proper network architecture and good parameter solutions given a specific task, outperforming a set of competitive baselines.

In summary, the contributions of the paper are three-folds:

- We propose M-NAS which adopts a task-conditional controller for architecture generation and learns a set of meta weights for fast estimations of optimal weigths associated with the candidate architecture.
- We consider task-aware search on highly diversed tasks and address the problem with transferable knowledge modulation.
- Extensive experiments and ablation studies demonstrate the advantages of task-aware neural architecture search, as well as the superiority of our proposed method against a set of competitive meta-learning methods.

Related Work

Neural Architecture Search (NAS) automatically finds optimal network architectures, relieving the architecture design of repetitive trial and errors. Existing NAS methods can be divided into three categories according to the way they explore the architecture search space: reinforcement learning, evolutionary algorithm and gradient-based methods (Elsken, Metzen, and Hutter 2019). Reinforcement learning (RL) based methods (Zoph and Le 2016; Pham et al. 2018) train an RNN controller to generate network architectures. Evolutionary algorithm based methods (Real et al. 2019) evolve neural architectures and achieve comparable results with RL based methods. The more recently proposed gradient-based methods (Liu, Simonyan, and Yang 2018; Cai, Zhu, and Han 2019) continuously relax the discrete architectures, which makes it possible to jointly optimize the architecture structure and network weights with gradient descent. Candidate architectures are supposed to be properly evaluated to provide supervision for searching. Training each architecture to convergence, however, is time consuming. Parameter sharing (Pham et al. 2018) was proposed for efficient searching, where parameters are shared among child models so a one-shot training is enough for evaluation of different candidate architectures. NAS has also been used to automatic efficient model design (Tan et al. 2019; Guo et al. 2019), which conventionally relies on hand-crafted heuristics (Howard et al. 2017; Sandler et al. 2018) or model compression (Han, Mao, and Dally 2016; Wu et al. 2016; Hu, Wang, and Cheng 2018). However, in these methods, only one task is considered and efficient search on previously unseen tasks is left unexplored. In this paper, we incorporate the idea of meta-learning for efficient task-aware architecture search on different tasks.

Another branch of researches related is meta-learning, where a model is trained to quickly adapt to new tasks given only a few samples (Santoro et al. 2016). Various meta-learning methods have been proposed and the most relevant is the model agnostic meta-learning. MAML (Finn, Xu, and Levine 2018) contains a meta-train stage and a meta-test stage. During meta-train, the model extracts general knowledge shared across different tasks so that it can be utilized for fast adaptation in the meta-test stage. A set of globally shared meta-parameter, however, is insufficient for heterogeneous tasks. Following works (Yoon et al. 2018; Vuorio et al. 2018) learns task-specific meta-parameters to better master different tasks. (Yao et al. 2019) then applies a hierarchical task clustering component to cluster tasks and tailor the meta-parameters according to the cluster properties. These works aim at improving meta-learning methods. While in our work, we utilize meta-learning for efficient task-aware model architecture search. Most recently, fast NAS with meta-learning on different tasks is explored. (Anonymous 2019) maintains a set of meta-parameters on the architecture parameters so that a few gradient descent steps on the meta architecture parameters gives optimal architecture for a specific task while we utilize a task-aware architecture controller to generate task-specific architectures.

Preliminaries

Differentiable Neural Architecture Search

Given a pre-defined task, the goal of neural architecture search is to find an architecture α that maximize the model performance. For differentiable architecture search techniques like DARTS (Liu, Simonyan, and Yang 2018), this can be done by solving a bi-level optimization problem over the network architecture α and network parameters w:

$$\begin{array}{ll} \min_\alpha & \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ \text{s.t.} & w^*(\alpha) = \arg\min_w \mathcal{L}_{\text{train}}(w, \alpha) \end{array} \tag{1}$$

where the architecture α is updated on the validation set, and weights $w(\alpha)$ associated to α are minimized on the training set. For each evaluation step on a candidate model α in the outer loop, a full training procedure needs to be conducted to obtain the optimal $w^*(\alpha)$, which leads to unaffordable computation burden. Parameter sharing (Pham et al. 2018) is widely adopted to address the issue, where all candidate architectures share the same parameters of a large *one-shot* network during the searching phase, different architectures correspond to different paths in the one shot model.

Despite parameter sharing works empirically well on a single task, it is insufficient for task-specific architecture search because the optimal parameters depend on both the underlying architecture as well as the task property. In this paper, we seek to provide a novel technique for task-aware architecture search, which utilizes meta-learning to perform efficient parameter estimation.

Gradient-Based Meta Learning

Meta-learning is helpful to understand the task-aware architecture search and fast optimal weights estimation in our method. Suppose that amounts of tasks $\{\mathcal{T}\}$ are sampled from a task distribution $p(\mathcal{T})$. In each task $\mathcal{T}_i \sim p(\mathcal{T})$, we have a few examples $\{\mathbf{x}_{i,k}, \mathbf{y}_{i,k}\}_{k=1}^{K} \in \mathcal{D}_{\mathcal{T}_i}$ for each class to constitute the training set $D_{\mathcal{T}_i}^{tr}$ and the rest as validation set $D_{\mathcal{T}_i}^{val}$. For tasks that have altogether N classes, this typically forms a N way K shot problem. The training split samples $D_{\mathcal{T}_i}^{tr}$ are used for fast task adaptation and validation split samples $\mathcal{D}_{\mathcal{T}_i}^{val}$ are used for update of meta parameters. The main idea of MAML (Finn, Abbeel, and Levine 2017) is to learn meta weights $\tilde{w}$ for all tasks, from where a few gradient descent steps can lead to significant increase of performance on previously unseen tasks. The fast task adaptation is performed by:

$$w_i = \tilde{w} - \rho_{\text{inner}} \nabla_{\tilde{w}} \mathcal{L}(\tilde{w}, \mathcal{D}_{\mathcal{T}_i}^{tr}), \tag{2}$$

where ρ_{inner} is the adaptation learning rate of weights and $\mathcal{L}$ is the loss function. The meta weights $\tilde{w}$ is updated by

$$\min_{\tilde{w}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(\tilde{w} - \rho_{\text{inner}} \nabla_{\tilde{w}} \mathcal{L}(\tilde{w}, \mathcal{D}_{\mathcal{T}_i}^{tr}), \mathcal{D}_{\mathcal{T}_i}^{val}\right), \tag{3}$$

where one gradient step in adaptation phase is adopted as exemplary. The model learns the meta-weights $\tilde{w}$ as good initialization for adaptation when Equation (3) converges.

Methods

To search for task-aware neural networks, we proposed meta neural architecture search (M-NAS), whose framework is presented in Figure 1. M-NAS can effectively generate proper candidate architectures for different tasks. Given a candidate architecture, M-NAS then adapt the transferable knowledge learned from previous searches via meta-learning, so that optimal weights associated with the child model can be efficiently estimated within a few gradient descent steps. In real-world applications the tasks are always heterogeneous and a shared set of meta-parameters can be insufficient. In this case, a task-specific meta-weights modulator is introduced to modulated the shared transferable knowledge, which is the meta-parameters $\tilde{w}$ here.

In the following, we detail M-NAS with three consecutive stages, i.e., 1) task-aware architecture generation, 2) transferable knowledge modulation, and 3) efficient optimal weights estimation. We also introduce a variant called M-NAS-Shared, which searches a shared optimal architecture for all the different tasks. This variant can also be seen as a NAS enhanced model agnostic meta-learning method.

Task-Aware Architecture Generation

A first step to perform task conditional architecture generation is to represent the task $\{\mathbf{x}_{i,j}, \mathbf{y}_{i,j}\}_{j=1}^{n^{tr}} \in \mathcal{D}_{\mathcal{T}_i}^{tr}$ with a task embedding $\mathbf{z}_i$. Task embedding has been previously explored in (Vuorio et al. 2018; Yao et al. 2019). Good task embeddings should satisfy the following properties: enough distinctions between different tasks, sufficient similarities between similar tasks, and permutational invariance to its inputs.

We apply a recurrent autoencoder (LSTM or GRU) to learn task context embeddings z_i, which is shown to be effective in few shot problems (Yao et al. 2019). Observations are sequentially fed into the recurrent auto-encoder, i.e.:

$$\begin{aligned} \mathbf{z}_{i,j} &= \text{RNN}_{enc}(\mathcal{F}(\mathbf{x}_{i,j}^{tr}, \mathbf{y}_{i,j}^{tr}), \mathbf{z}_{i,j-1}) \\ \mathbf{d}_{i,j} &= \text{RNN}_{dec}(\mathbf{z}_{i,j}, \mathbf{d}_{i,j+1}) \end{aligned} \tag{4}$$

where we use $\mathcal{F}(\cdot,\cdot)$ to preliminarily embed both features and predictions of a data pair. $\mathbf{z}_{i,j}$ and $\mathbf{d}_{i,j}$ represents the learned representation and the reconstruction of the j-th sample, respectively. RNNenc and RNNdec stand for a recurrent encoder and a recurrent decoder. And the reconstruction loss for training the auto-encoder is as follows:

$$\mathcal{L}_r\left(\mathcal{D}_{T_i}^{tr}\right) = \sum_{j=1}^{n^{tr}} \left\| \mathbf{d}_{i,j} - \mathcal{F}\left(\mathbf{x}_{i,j}^{tr}, \mathbf{y}_{i,j}^{tr}\right) \right\|_2^2 \tag{5}$$

The task representation is aggregated over representations of all samples, i.e.,

$$\mathbf{z}_i = \frac{1}{n^{tr}} \sum_{j}^{n^{tr}} (\mathbf{z}_{i,j}) \tag{6}$$

However, the sequential feeding of samples makes the final task representation to be permutation sensitive, which violates the input invariance prerequisite. The problem can be addressed by pre-permuting all the samples and then fed them into the recurrent autoencoder (Hamilton, Ying, and Leskovec 2017). With the task representation extracted, task conditional candidate architectures can be readily generated by $\alpha_i = \mathcal{A}(\mathbf{z}_i)$, where $\mathcal{A}$ can be an LSTM controller (Zoph and Le 2016; Pham et al. 2018) or trainable architecture parameters (Liu, Simonyan, and Yang 2018). In our case, we use a multi-layer perceptron as the task-aware controller to generate architecture parameters and path dropout is applied on the architecture α to avoid local optima.

Transferable Knowledge Modulation

To perform fast optimal weights estimation on a new task, we try to leverage previous search knowledge on other tasks. Inspired by MAML (Finn, Xu, and Levine 2018), in M-NAS we maintain a set of meta-parameters $\tilde{w}$ on the big one-shot model, which is shared across all the child architectures. For a task-aware candidate architecture α_i, corresponding meta-parameters $\tilde{w}(\alpha_i)$ are triggered, from where a few gradient descent steps give approximated optimal weights associated with the architecture α_i. Nevertheless, the technique implicitly relies on the assumption that solutions for tasks are close to each other in the solution space. Properly optimized meta-parameters $\tilde{w}$ live in the "center" among the solutions so it can fast adapt to the solution for each task.

However, as discussed in (Vuorio et al. 2018), when the tasks are heterogeneous so that modes of task distribution are disjoint and far apart, a common initialization point $\tilde{w}$ for all tasks can be insufficient given the same adaptation routine. This problem can even be more serious in conjunction with NAS. When globally shared meta-parameters on the one-shot model can not master all tasks, the child models will not be properly evaluated with the fast adaptation scheme, which leads to unreliable supervising signals for architecture controller training, eventually resulting in deteriorated architectures.

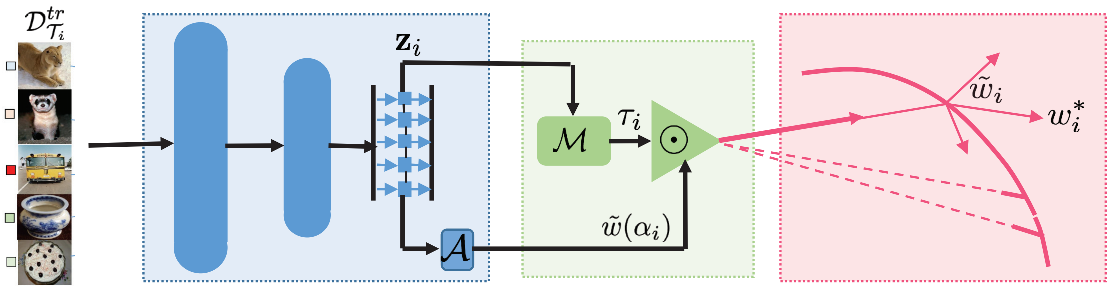


Figure 1: The framework of the proposed M-NAS involving three essential stages. (a) Task conditional architecture generating: we extract representation for the task $\mathcal{T}_i$ using an recurrent auto-encoder, which is then used to generate a task-aware model architecture α_i. (b) Transferable knowledge modulation: we tailor the globally shared meta-parameters associated with α_i, denoted as $\tilde{w}(\alpha_i)$, to a specific task $\tilde{w}_i$. (c) Fast optimal weights estimation: Given an architecture α_i and the modulated meta-parameters $\tilde{w}_i$, apply a few gradient descent steps to get an estimation of the optimal weights $w^*_i(\alpha_i)$.

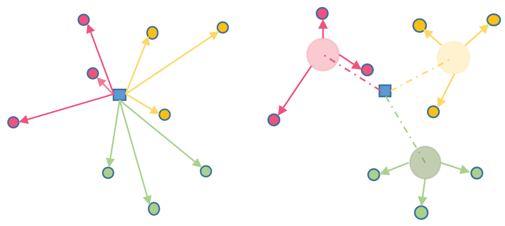


Figure 2: Task specific modulation on the meta-parameters. Circles represent optimal solutions for tasks, which forms a distribution with three modalities. (a) Globally Shared meta-parameters: model is having hard time to adapt to solutions when tasks are heterogeneous. (b) Meta-parameters with task-specific modulation. Meta-weights are first modulated to be closer to the task clusters.

To address the problem, we apply a task specific modulator $\mathcal{M}$ on the learned meta-weights $\tilde{w}$, similar to (Yao et al. 2019). The modulator generates modulation parameters:

$$\tau_i = \mathcal{M}(\mathbf{z}_i; \mathbf{W}_\mathcal{M}) \tag{7}$$

where $\mathcal{M}$ can be a fully-connected layer parameterized with $\mathbf{W}_\mathcal{M}$. The transferable knowledge, i.e., the meta-parameters $\tilde{w}$ is then modulated to $\tilde{w}_i = \tau_i \odot \tilde{w}$. The modulation acts on all the meta-parameters of the one-shot model, which can be seen as a kind of attention that similar tasks activate similar meta-parameters while different tasks activate disparate ones. Transferable knowledge modulation is visualized in Figure 2. Figure 2(a) shows that different tasks require substantially different parameters, and a single set of shared meta-parameters on the one-shot model (blue rectangle) can be insufficient mastering the full task distribution. In Figure 2(b), given the estimated task representation and the model architecture, our model then performs task specific modulation (dashed lines) on the meta-learned prior (blue rectangle) to move the meta-parameters to different initial positions, each of which covers different modes to better adapt to heterogeneous tasks.

Efficient Optimal Weights Estimation

Given an architecture α_i and the modulated meta-parameters $\tilde{w}_i = \tau_i \odot \tilde{w}$, which contain both the transferable knowledge across different tasks and the current task information. A few gradient descent steps will give an estimation of the optimal weights for a specific task:

$$w^*_i = \tilde{w}_i - \rho_{\text{inner}} \nabla_{\tilde{w}_i} \mathcal{L}(\tilde{w}_i, \mathcal{D}^{tr}_{\mathcal{T}_i}). \tag{8}$$

Architecture searching and shared meta-parameter training are carried out in the outer optimization:

$$\min_{\Theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(w^*_i(\alpha_i), \mathcal{D}^{val}_{\mathcal{T}_i}\right) + \beta \mathcal{L}_r(\mathcal{D}^{tr}_{\mathcal{T}_i}) \tag{9}$$

where $\Theta = \{\mathbf{W}_\mathcal{E}, \mathbf{W}_\mathcal{A}, \mathbf{W}_\mathcal{M}, \tilde{w}\}$ is the collection of encoder parameters $\mathbf{W}_\mathcal{E}$, architecture controller parameters $\mathbf{W}_\mathcal{A}$, modulator parameters $\mathbf{W}_\mathcal{M}$, as well as the shared meta-parameters $\tilde{w}$. The architecture α_i is generated with $\alpha_i = \mathcal{A}(z_i)$ where z_i is the task representation aggregated with $\mathcal{D}^{tr}_{\mathcal{T}_i}$ and β is used to balance the empirical risk and the reconstruction error of the recurrent auto-encoder.

Different from the objective for NAS in Equation (1), a fast adaptation scheme Equation (8) is adopted to substitute the inner optimization over network parameters for the candidate architecture of a specific task. Shared meta-parameters learning, task representation learning as well as transferable knowledge modulator learning are simultaneously conducted with architecture search. Therefore, M-NAS can be trained in an end to end manner. Finally, after M-NAS is well trained, given a few samples of a previously unseen task, architecture search can be finished with just a forward pass of the task encoder and controller. A complete algorithm of M-NAS is as shown in Algorithm 1.

Remark Although M-NAS aims to learn task-specific architectures, it can be easily reduced to searching a single

Algorithm 1 M-NAS: Meta Neural Architecture Search

```
Require: Meta-train dataset $\mathcal{D}_{\text{meta-train}}$, learning rates $\rho_{\text{inner}}$, $\rho_{\text{outer}}$, hyper-parameter $\beta$
Ensure:
    Task representation extractor: $\mathcal{E}$
    Task-aware architecture controller: $\mathcal{A}$
    Shared meta-parameters on the big one-shot model $\tilde{w}$
    Task-specific modulator $\mathcal{M}$
1:  Randomly initialize $\Theta = \{\mathbf{W}_{\mathcal{E}}, \mathbf{W}_{\mathcal{A}}, \mathbf{W}_m, \tilde{w}\}$
2:  while not done do
3:      Sample batch of tasks $\{\mathcal{T}\}$ in $\mathcal{D}_{\text{meta-train}}$.
4:      for $\mathcal{T}_i \in \mathcal{T}$ do
5:          Get the task representation $\mathbf{z}_i$ and generate an candidate architecture $\alpha_i$.
6:          Modulate the meta-parameters with $\tilde{w}_i = \tau_i \odot \tilde{w}$.
7:          Approximate the optimal weights with Equation (8).
8:      end for
9:      Update $\Theta$ by optimizing Equation (9).
10: end while
```

network architecture as base learner for different tasks. This can be done by optimizing the architecture parameters α as independent trainable variables and the resulting model is named M-NAS-Shared. M-NAS-Shared resembles (Kim et al. 2018) that attempts to find a better base learner architecture for meta-learning tasks. In experiments, we also investigate M-NAS-Shared to see if searching for a base learner benefits meta-learning tasks.

Experiments

We evaluate the effectiveness of M-NAS as well as its variant M-NAS-Shared on a toy regression example and two real-world image classification problems. To validate the necessity of the transferable knowledge modulator, the toy example and one of the image classification datasets are constructed to contain obviously heterogeneous tasks. We implement M-NAS in PyTorch (Paszke et al. 2017).

Baselines We compare our proposed methods against following baselines: (1) Gradient based meta-learning method with globally shared initialization, i.e., MAML (Finn, Abbeel, and Levine 2017); (2) Meta learning methods with task specific initialization including MT-Net (Lee and Choi 2018), BMAML (Yoon et al. 2018), MUMOMAMAL (Vuorio et al. 2018) and HSML (Yao et al. 2019); (3) Meta-learning with learned architecture shared for all tasks, including Auto-Meta (Kim et al. 2018) and our proposed M-NAS-Shared. We also conduct blation studies to investigate the contribution of transferable knowledge modulation. The method without the modulation is named M-NAS-NM.

Toy Regression

Dataset and Experimental Settings We consider similar settings as in (Vuorio et al. 2018) where tasks are sampled from different families of functions. We set up the task distribution with four task modes: (1) *sinusoidal*, $y(x) = a_s \sin(w_s x) + b_s$, $a_s \sim U[0.1, 5.0], w_s \sim U[0.8, 1.2]$ and $b_s \sim U[0, 2\pi]$; (2) *linear*, $y(x) = a_l x + b_l, a_l \sim U[-3.0, 3.0]$ and $b_l \sim U[-3.0, 3.0]$; (3) *quadratic* $y(x) = a_q x^2 + b_q x + c_q, a_q \sim U[-0.2, 0.2], b_q \sim U[-2.0, 2.0]$ and $c_q \sim U[-3.0, 3.0]$; (4) *harmonics* $f(x) = a_{h1} \sin(w_h \cdot x + b_{h1}) + a_{h2} \sin(2 \cdot w_h \cdot x + b_{h2})$ where $(a_{h1}, a_{h2}) \sim U[0.1, 5.0]^2, (b_{h1}, b_{h2}) \sim U(0, 2\pi)^2$ and $w_h \sim U[0.8, 1.2]$. $U[\cdot, \cdot]$ indicates the uniform distribution. Each task is randomly sampled from one of the four underlying function families, so that the constructed tasks are heterogeneous and multi-modal. Training samples x are sampled from normal distribution $x \sim \mathcal{N}(u, 2)$ where $u \sim U[-4, 4]$. Training samples are concentrated here compared to previous works where x are uniformly sampled from -5.0 to 5.0. With concentrated training samples, we can delve into the extrapolation ability of the proposed models in data sparse areas. We train all models for 10-shot regression and report the mean square error (MSE) as the evaluation metric.

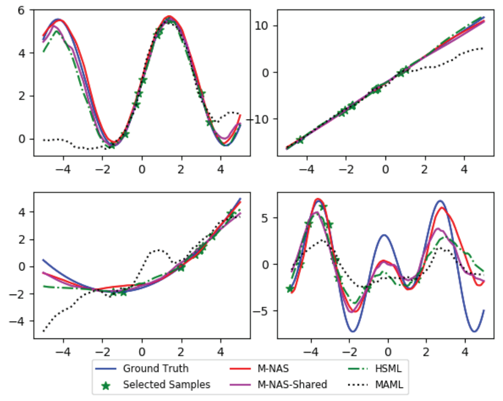


Figure 3: Few-shot adaptation for four toy regression problems. Models trained with searched architectures extrapolate better than models trained with hand-crafted architectures.

For toy regression tasks, previous works use a simple model with two fully-connected layers with Batch Normalization, each contains 40 neurons. The model capacity is too small to perform NAS. So for baseline methods, we consider a slightly larger model: one with 4 hidden fully-connected layers (altogether 5 layers), each with 50 neurons. For fully-connected layers We use a very simple search space similar to (Liu, Simonyan, and Yang 2018). Only three types of operations: linear, skip connection and zero, which means no connection, are considered. We use the Linear-Relu-BN for each linear operation. The searched models are represented with cells, a cell contains two input nodes, one output node, and three intermediate nodes. Each node accepts results of two previous nodes as inputs and outputs of all the intermediate nodes are mean aggregated to form the final cell output. For the toy regression task, we only use one cell for efficiency. We use the vanilla SGD for fast weights estimation in Equation (8) and Adam (Kingma and Ba 2014) for meta-updates Equation (9). The learning rates are ρ_{inner} = 0.001 and ρ_{outer} = 0.001 respectively. Specifically, we use smaller

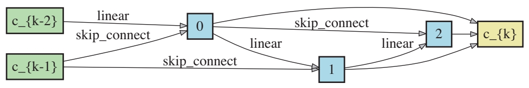


Figure 4: Architecture searched learning a shared single base learner in 10-shot setting of toy regression. The searched architecture contains altogether 5 fully-connected layers.

Table 1: Performance of MSE±95% confidence intervals on toy regression problem. Each experiment is averaged over 1000 tasks for each task family. 10-shots results are reported.

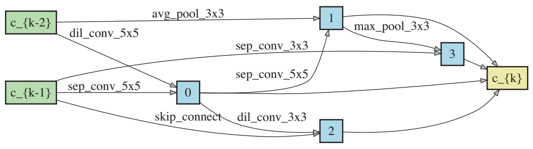
Model	Archs.	Harmon. MSE	Averaged MSE
MAML	5 FCs	2.210 ± 0.364	1.941 ± 0.376
MT-Net	5 FCs	2.213 ± 0.358	2.031 ± 0.359
BMAML	5 FCs	2.073 ± 0.361	1.925 ± 0.324
MUMOMAML	5 FCs	1.786 ± 0.184	0.524 ± 0.031
HSML	5 FCs	1.687 ± 0.171	0.502 ± 0.041
M-NAS-NM	Cell	1.973 ± 0.186	1.772 ± 0.356
M-NAS-Shared	Cell	$\mathbf{1.264 \pm 0.117}$	$\mathbf{0.360 \pm 0.037}$
M-NAS	Cell	$\mathbf{1.232 \pm 0.121}$	$\mathbf{0.352 \pm 0.032}$

learning rate $\rho_{\mathcal{A}}$ = 1e-4 for architecture parameters $\mathbf{W}_{\mathcal{A}}$. The balancing weight is set as $\beta = 0.01$ and fast adaptation for optimal weights estimation is carried out for 5 steps.

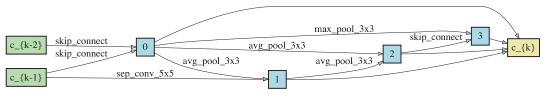
Architecture Evaluation We report the averaged MSE of four regression problems as well the MSE of harmonic regression, which is the most challenging problem. The other three families are relatively easier and less discriminative so we omit their MSE due to limited space. The results are shown in Table 1. It can be found that without task specific transferable knowledge modulation, MAML and M-NAS-NM can not handle the heterogeneous tasks thus perform poorly. Our proposed M-NAS outperforms all baselines with lower MSE. We also visualize each family of tasks in Figure 3, where models trained with searched architectures extrapolate better than the models trained with hand-crafted architectures, especially on the harmonic regression task with higher frequencies signals. With complex structures such as skip connections introduced, the searched architecture can better handle the harmonic regression, validating the effectiveness of our proposed model. As an example, one architecture searched by M-NAS-shared is shown in Figure 4.

Few Shot Classification

Datasets and Experimental Settings To verify whether M-NAS can search proper architectures for different tasks, we applied our method to MiniImagenet (Ravi and Larochelle 2017) datasets and a recently constructed benchmark Multi-datasets (Yao et al. 2019). MiniImagenet has been used extensively in meta-learning literature. However, recent analysis (Finn, Xu, and Levine 2018) shows that the sampled tasks in MiniImagenet benchmark do not have obvious heterogeneity and uncertainty. Multi-datasets

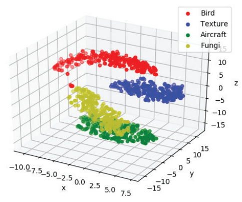


(a) Normal cell

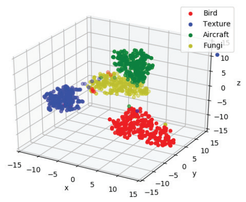


(b) Reduction cell

Figure 5: Architecture searched when learning a shared single base learner in 5-way 1-shot setting of Multi-datasets. The searched architecture contains 27.0K parameters



(a) M-NAS-Shared



(b) M-NAS

Figure 6: t-SNE visualization of tasks on Multi-datasets. Each group has 250 tasks. (a) M-NAS-Shared (b) M-NAS.

is constructed using four datasets: Caltech-UCSD Birds-200-2011 (Bird), Describable Textures Dataset (Texture), Fine-Grained Visual Classification of Aircraft (Aircraft), and FGVCx-Fungi (Fungi). Following the N-way K-shot protocol, each task is constructed as a fine-grained classification task where the observations are sampled from one of the four datasets so the tasks are more heterogeneous [1].

For architecture search, we employ the same operations as in (Liu, Simonyan, and Yang 2018): 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity and zero. For all datasets, we only use one {normal + reduction} cell for efficiency. Fast weights estimation in Equation (8) is carried out for 5 steps with vanilla SGD while meta-updates and searching Equation (9) are performed with Adam. The learning rates are $\rho_{\text{inner}} = 0.01$ and $\rho_{\text{outer}} = 0.001$ respectively. Specifically, we use smaller learning rate $\rho_{\mathcal{A}}$ = 1e-4 for architecture parameters $\mathbf{W}_{\mathcal{A}}$. The balancing weight is set as $\beta = 0.01$. Finally, since the training task is varying throughout the search process, we always use batch statistics for batch normalization instead of the global moving average.

Architectures Evaluation In architecture evaluation, for M-NAS-Shared we re-initialize the searched models and

[1]Visualization of different tasks is given in the appendix

Table 2: Comparison between M-NAS and other gradient-based meta-learning methods on the 5-way, 1-shot/5-shot image classification problem, averaged over 250 tasks for each dataset. Accuracy ± 95% confidence intervals are reported.

	Model	Params.	Bird	Texture	Aircraft	Fungi	Average
5 way 1 shot	MAML	32.9 K	53.94 ± 1.45%	31.66 ± 1.31%	51.37 ± 1.38%	42.12 ± 1.36%	44.77%
	MT-Net	32.9 K	58.72 ± 1.43%	32.80 ± 1.35%	47.72 ± 1.46%	43.11 ± 1.42%	45.59%
	BMAML	32.9 K	54.89 ± 1.48%	32.53 ± 1.33%	53.63 ± 1.37%	42.50 ± 1.33%	45.89%
	MUMOMAML	32.9 K	56.82 ± 1.49%	33.81 ± 1.36%	53.14 ± 1.39%	42.22 ± 1.40%	46.50%
	HSML	32.9 K	58.22 ± 1.48%	33.30 ± 1.36%	55.35 ± 1.38%	42.68 ± 1.40%	47.39%
	M-NAS-NM	31.6 K*	52.37 ± 1.50%	32.49 ± 1.48%	48.53 ± 1.42 %	40.68 ± 1.45%	43.52%
	M-NAS-Shared	27.0 K	57.13 ± 1.43%	**34.75 ± 1.33**%	56.32 ± 1.38%	43.33 ± 1.39%	**47.88**%
	M-NAS	29.5 K*	**58.76 ± 1.47**%	34.68 ± 1.36%	**57.13 ± 1.41**%	**43.71 ± 1.41**%	**48.57**%
5 way 5 shot	MAML	32.9 K	68.52 ± 0.79%	44.56 ± 0.68%	66.18 ± 0.71%	51.85 ± 0.85%	57.78%
	MT-Net	32.9 K	69.22 ± 0.75%	46.57 ± 0.70%	63.03 ± 0.69%	53.49 ± 0.83%	58.08%
	BMAML	32.9 K	69.01 ± 0.74%	46.06 ± 0.69%	65.74 ± 0.67%	52.43 ± 0.84%	58.31%
	MUMOMAML	32.9 K	70.49 ± 0.76%	45.89 ± 0.69%	67.31 ± 0.68%	53.96 ± 0.82%	59.41%
	HSML	32.9 K	71.31 ± 0.75%	47.11 ± 0.71%	**71.51 ± 0.69** %	54.30 ± 0.79%	61.06 %
	M-NAS-NM	31.6 K*	67.78 ± 0.81%	45.81 ± 0.72%	66.35 ± 0.73%	50.73 ± 0.86%	57.67 %
	M-NAS-Shared	27.0 K	**72.24 ± 0.75**%	47.15 ± 0.71%	70.87 ± 0.67%	55.23 ± 0.80%	**61.37** %
	M-NAS	29.5 K*	72.22 ± 0.78%	**48.17 ± 0.74**%	71.31 ± 0.70%	**55.85 ± 0.82**±	**61.89** %

Table 3: Comparison between our approach and previous few-shot learning methods on the 5-way, 1-shot MiniImagenet benchmark. * means average number of parameters of Archs. searched.

Methods	Archs.	Params.	1 shot
MAML	4 Convs	32.9K	48.70 ± 1.84%
MT-Net	4 Convs	32.9K	49.75 ± 1.83%
BMAML	4 Convs	32.9k	50.01 ± 1.86%
HSML	4 Convs	32.9K	50.38 ± 1.85%
Auto-Meta	Cell	28.0K	49.58 ± 0.20%
M-NAS-NM	Cell	27.1K*	**50.43 ± 1.73**%
M-NAS-Shared-M	Cell	26.3K	**50.78 ± 1.66**%
M-NAS	Cell	27.9K*	**51.37 ± 1.41**%

train them with 120000 tasks. The training process is also gradient based meta-learning sharing similar protocol as in searching: extract task representations, apply meta-parameters modulation and fast adapt from the meta-parameters, but this time with the fixed searched architecture and the task specific modulation acts on the searched model meta-parameters rather than on the large one-shot model meta-parameters. For M-NAS, searched task specific models are decoded from the shared large one-shot model by setting weights of operations that are not selected to 0 when the search process is near convergence. The task-aware controller is then tuned with straight through estimation (STE) (Bengio, Léonard, and Courville 2013). The model usually generates a few architecture for test tasks (4 to 10 for Multi-datasets), the decoded architectures can also be re-trained with moderate number of training tasks.

The classification result on MiniImagenet is shown in Table 3. All the baselines' performances are taken from (Yao et al. 2019). We see that the proposed methods outperform all the baselines with fewer model parameters, indicating that they successfully searched model architecture(s) better than the hand-crafted one. For MiniImagenet, There is no obvious heterogeneity so task-specific modulation is not a must.

For Multi-datasets, we report the averaged accuracy over 1000 tasks of 5-way1-shot/5-shot classification in Table 2. Baselines are taken from (Yao et al. 2019) except for HSML. HSML enjoys a hierarchical task clustering component which is absent in our case. We remove the hierarchical clustering component [2] of HSML and report results obtained by the published code [3]. For datasets with obvious heterogeneity, M-NAS-NM fails to find optimal architecture because, without modulation, the meta-parameters are unable to master all the task modalities. The fast adaptation then fails to properly evaluate the child architectures, resulting in deteriorated controller and improper model architecture.

From Table 2, we see that M-NAS consistently outperforms the other baselines on each dataset with smaller model parameters, which demonstrates the power of searching optimal architectures of M-NAS. Surprisingly, by searching an optimal architecture for all tasks, M-NAS-Shared also performs comparably well, which might be due to insufficient training of each child architecture in M-NAS. An example of shared architecture found by M-NAS-Shared is shown in Figure 5. Finally, to make sure the task-aware controller effectively utilizes the task information for architecture generation, we visualize task embeddings of 1000 test tasks by t-SNE (van der Maaten and Hinton 2008) in Figure 6. It can be observed that our proposed methods are able to identify the tasks in different clusters.

[2]Comparison with unmodified HSML is given in the appendix
[3]https://github.com/huaxiuyao/HSML

Conclusion and Discussion

In this paper, we introduce M-NAS to search optimal model architectures for different tasks. M-NAS incorporates meta-learning for fast evaluation of a candidate architecture. Compared with several baselines, experiments demonstrated the necessity of using different architectures for different tasks as well as the effectiveness of our algorithm in both toy regression and few-shot classification problems.

Although our method is widely applicable, there are limitations and interesting future directions. For example, Training of M-NAS is still time consuming as the searching requires second-order derivatives and naively apply first-order approximation as in (Finn, Abbeel, and Levine 2017) doesn't yield promising result. For future work, we will try to develop fast first-order algorithms to solve the problem.

Acknowledgment

We thank Yin Zheng and Dongze Lian for helpful discussions. This work was supported in part by National Natural Science Foundation of China (No.61972396, 61876182, 61906193), the Strategic Priority Research Program of Chinese Academy of Science(No.XDB32050200), the Advance Research Program (31511130301).

References

Anonymous, A. 2019. Towards fast adaptation of neural architectures with meta learning. In *https://openreview.net/forum?id=r1eowANFvr*.

Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and simplifying one-shot architecture search. In *ICML*, 550–559.

Bengio, Y.; Léonard, N.; and Courville, A. C. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*.

Chen, L.-C.; Collins, M.; Zhu, Y.; Papandreou, G.; Zoph, B.; Schroff, F.; Adam, H.; and Shlens, J. 2018. Searching for efficient multi-scale architectures for dense image prediction. In *NeuralPS*, 8699–8710.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural architecture search: A survey. *JMLR* 20(55):1–21.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 1126–1135.

Finn, C.; Xu, K.; and Levine, S. 2018. Probabilistic model-agnostic meta-learning. In *NeuralPS*. 9516–9527.

Guo, Y.; Zheng, Y.; Tan, M.; Chen, Q.; Chen, J.; Zhao, P.; and Huang, J. 2019. Nat: Neural architecture transformer for accurate and compact architectures. In *NeuralPS*. 735–747.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeuralPS*. 1024–1034.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, Q.; Wang, P.; and Cheng, J. 2018. From hashing to cnns: Training binary weight networks via hashing. In *AAAI*, 3247–3254.

Kim, J.; Choi, Y.; Cha, M.; Lee, J. K.; Lee, S.; Kim, S.; Choi, Y.; and Kim, J. 2018. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lee, Y., and Choi, S. 2018. Gradient-based meta-learning with learned layerwise metric and subspace. In *ICML*, 2933–2942.

Liu, C.; Chen, L.-C.; Schroff, F.; Adam, H.; Hua, W.; Yuille, A.; and Fei-Fei, L. 2019. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *CVPR*.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *Workshop on Autodiff at NeuralPS*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Ravi, S., and Larochelle, H. 2017. Optimization as a model for few-shot learning. In *ICLR*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. *AAAI*.

Sandler, M.; Howard, A. G.; Zhu, M.; Zhmoginov, A.; and Chen, L. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 4510–4520.

Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. Meta-learning with memory-augmented neural networks. In *ICML*, 1842–1850.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2820–2828.

van der Maaten, L., and Hinton, G. 2008. Visualizing data using t-SNE. *JMLR* 9:2579–2605.

Vuorio, R.; Sun, S.-H.; Hu, H.; and Lim, J. J. 2018. Toward multimodal model-agnostic meta-learning. In *Workshop on Meta-Learning at NeuralPS*.

Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; and Cheng, J. 2016. Quantized convolutional neural networks for mobile devices. In *CVPR*.

Yao, H.; Wei, Y.; Huang, J.; and Li, Z. 2019. Hierarchically structured meta-learning. In *ICML*, 7045–7054.

Yoon, J.; Kim, T.; Dia, O.; Kim, S.; Bengio, Y.; and Ahn, S. 2018. Bayesian model-agnostic meta-learning. In *NeuralPS*. 7332–7342.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *ICLR*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*, 8697–8710.