Group reading

Towards Fast Adaptation of Neural Architectures with Meta Learning

Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, Shenghua Gao

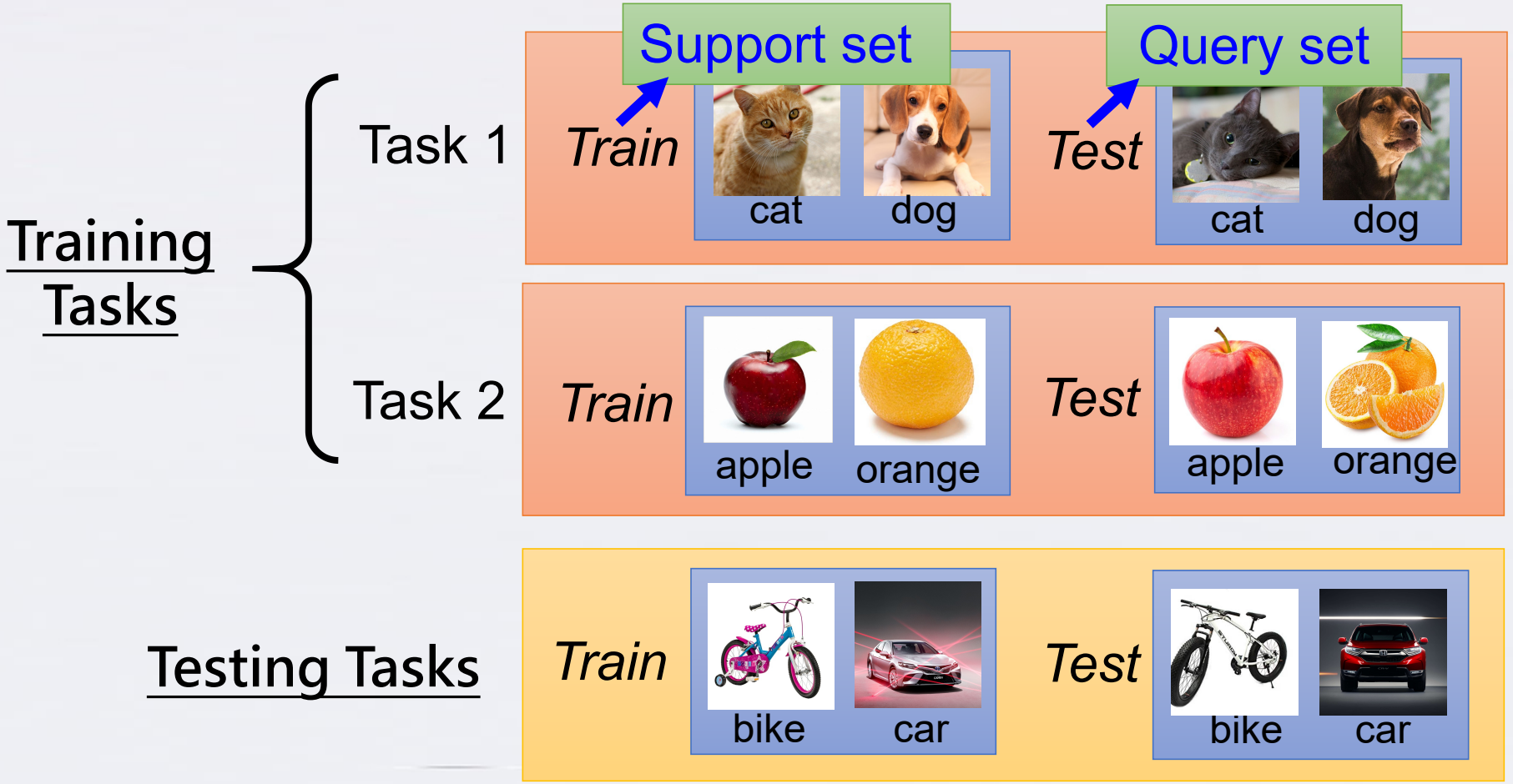
ICLR 2020

Shasha Zhou **11.22**

目录
CONTENTS

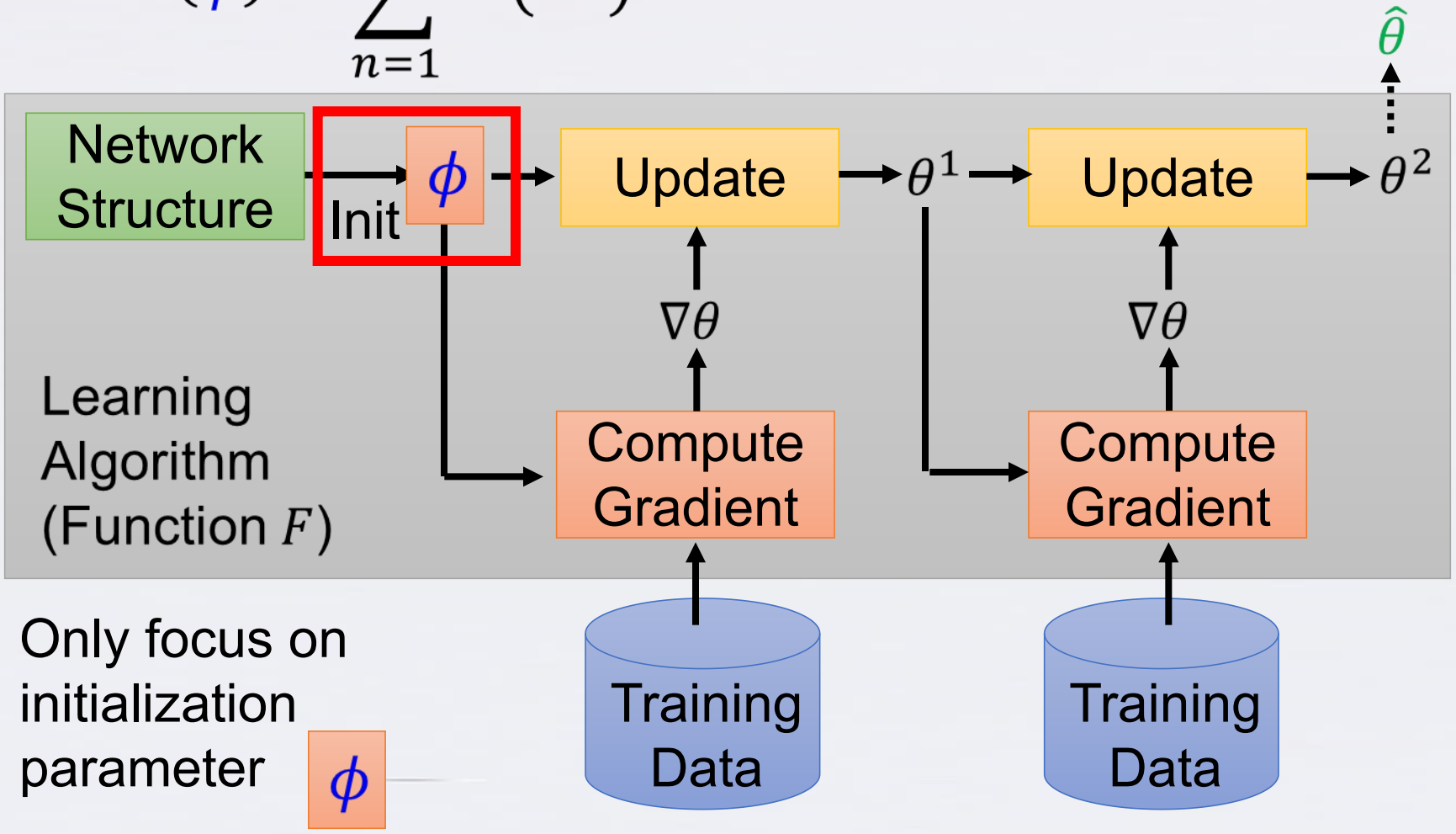
MAML

- **meta-learning**



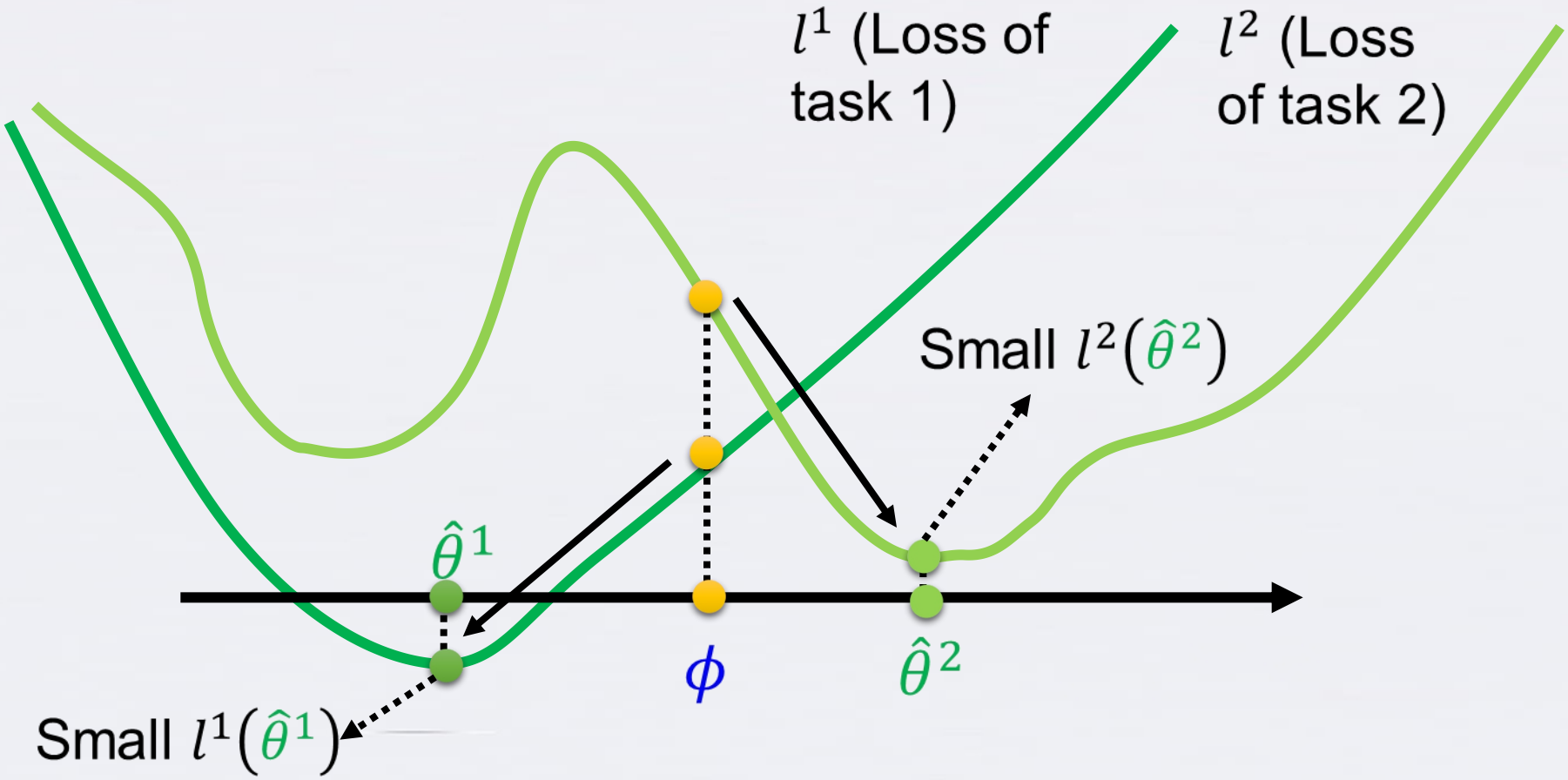
MAML

Loss Function: $L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)$



MAML

$$L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)$$



MAML

$$w_i^{m+1} = w_i^m - \alpha_{inner} \nabla_{w_i^m} \mathcal{L}(f(\mathcal{T}_i^s; w_i^m))$$

$$\tilde{w} = \tilde{w} - \alpha_{outer} \nabla_{\tilde{w}} \sum_{\mathcal{T}_i^q \sim p(\mathcal{T})} \mathcal{L}(f(\mathcal{T}_i^q; w_i^M))$$

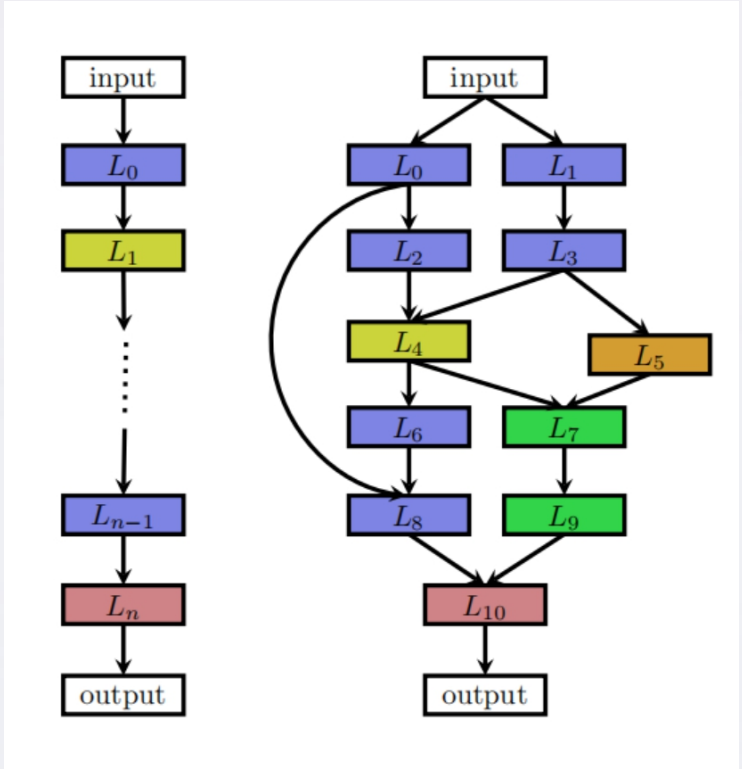
Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Neural Architecture Search(NAS)

- Search space
- Search strategy
- Performance estimation strategy

层/运算	超参数
卷积	卷积核数量 卷积核通道数 卷积核宽度 卷积核高度 水平方向步长 垂直方向步长
池化	池化核高度 池化核宽度 水平方向步长 垂直方向步长
全连接	神经元数量
激活	激活函数类型 各种激活函数的参数
相加（add）	无
拼接（contact）	无



Neural Architecture Search(NAS)

- **Search strategy**

Bayesian optimization

Evolutionary methods

Reinforcement learning

Gradient-based methods(DARTS)

……

Neural Architecture Search(NAS)

- **Performance estimation strategy**

Lower fidelity estimates

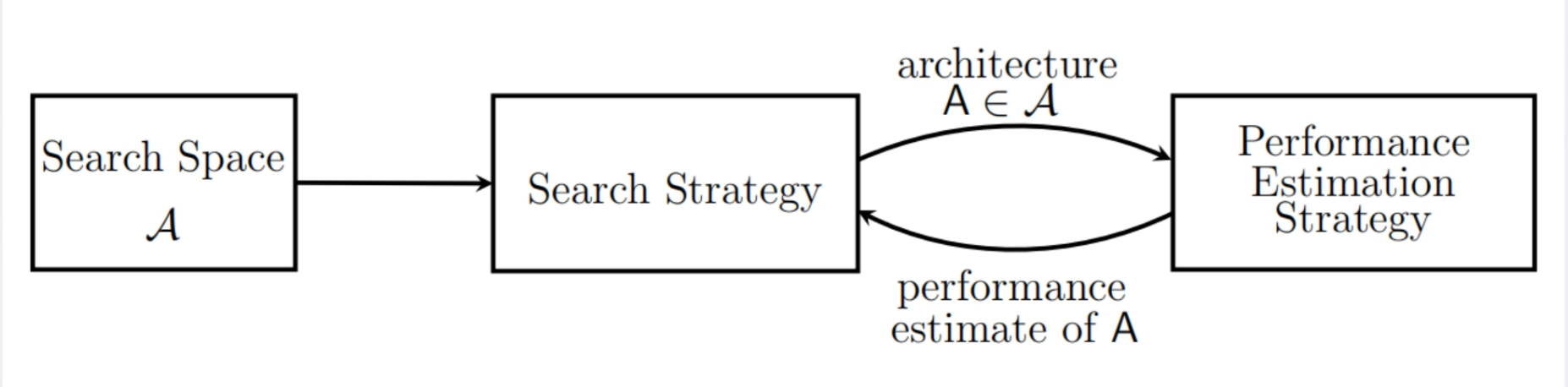
Learning curve extrapolation

Weight inheritance/ network morphisms

One-shot models/ weight sharing

……

Neural Architecture Search(NAS)



Neural Architecture Search(NAS)

- **Gradient-base methods(DARTS)**

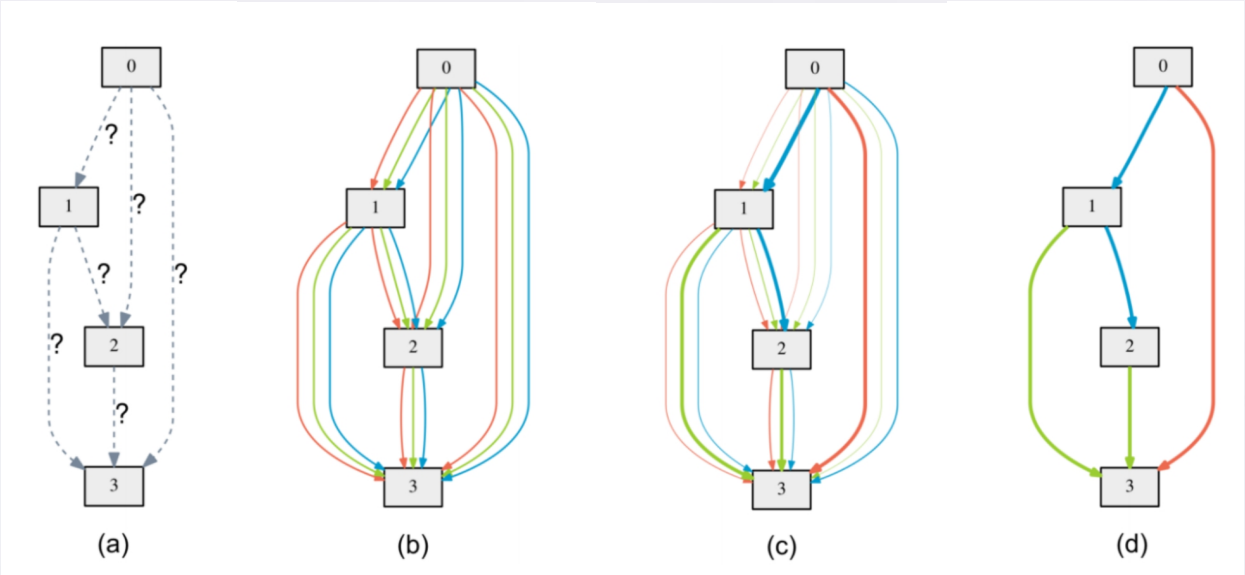
DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH

Hanxiao Liu, Karen Simonyan, Yiming Yang

ICLR 2019

Neural Architecture Search(NAS)

- **Gradient-base methods(DARTS)**



$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(j)})$$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp\left(\theta_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\theta_{o'}^{(i,j)}\right)} o(x)$$

Neural Architecture Search(NAS)

- **Gradient-base methods(DARTS)**

$$\min_{\theta} \mathcal{L}_{val}(w^*(\theta), \theta)$$

$$s.t. w^*(\theta) = argmin_w \mathcal{L}_{val}(w, \theta)$$

$$\begin{cases} \theta = \theta - \beta \nabla_\theta \mathcal{L}(w - \xi \nabla_w \mathcal{L}(w, \theta), \theta) \\ w = w - \xi \nabla_w \mathcal{L}(w, \theta) \end{cases}$$

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)

while *not converged* **do**

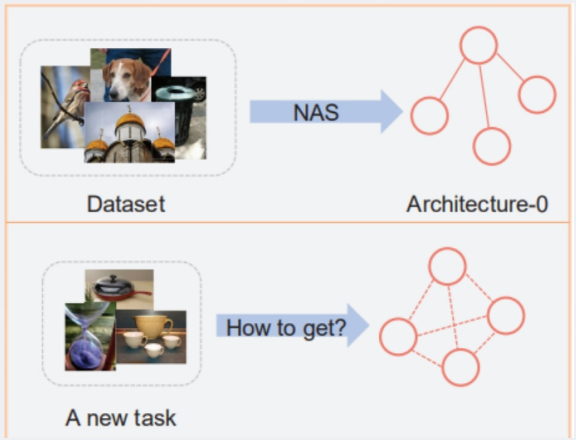
1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$ ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

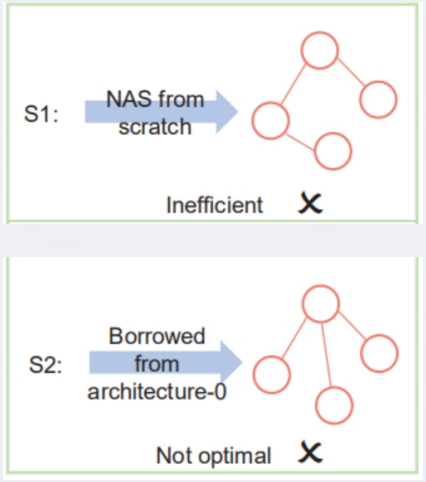
Derive the final architecture based on the learned α.

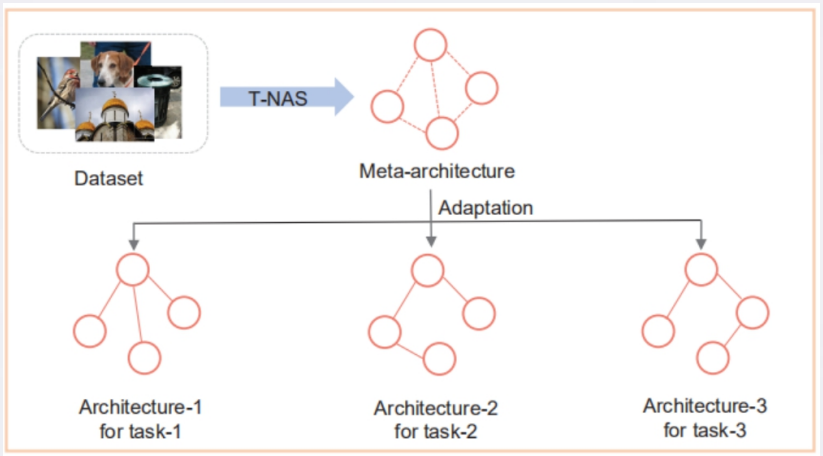
Transferable Neural Architecture(T-NAS)

- **Problem**

The existing NAS methods only target a specific task







Transferable Neural Architecture(T-NAS)

- **Solution**

$$w_i^{m+1} = w_i^m - \alpha_{inner} \nabla_{w_i^m} \mathcal{L}(f(\mathcal{T}_i^s; w_i^m))$$

$$\widetilde{w} = \widetilde{w} - \alpha_{outer} \nabla_{\widetilde{w}} \sum_{\mathcal{T}_i^q \sim p(\mathcal{T})} \mathcal{L}(f(\mathcal{T}_i^q; w_i^M))$$

$$\begin{cases} \theta = \theta - \beta \nabla_\theta \mathcal{L}(w - \xi \nabla_w \mathcal{L}(w, \theta), \theta) \\ w = w - \alpha \nabla_w \mathcal{L}(w, \theta) \end{cases}$$

$$\begin{cases} w_i^{m+1} = w_i^m - \alpha_{inner} \nabla_{w_i^m} \mathcal{L}(g(\mathcal{T}_i^s; \theta_i^m, w_i^m)) \\ \theta_i^{m+1} = \theta_i^m - \beta_{inner} \nabla_{\theta_i^m} \mathcal{L}(g(\mathcal{T}_i^s; \theta_i^m, w_i^{m+1})) \end{cases}$$

$$\begin{cases} \widetilde{w} = \widetilde{w} - \alpha_{outer} \nabla_{\widetilde{w}} \sum_{\mathcal{T}_i^q \sim p(\mathcal{T})} \mathcal{L}(g(\mathcal{T}_i^q; \theta_i^M, w_i^M)) \\ \tilde{\theta} = \tilde{\theta} - \beta_{outer} \nabla_{\tilde{\theta}} \sum_{\mathcal{T}_i^q \sim p(\mathcal{T})} \mathcal{L}(g(\mathcal{T}_i^q; \theta_i^M, w_i^M)) \end{cases}$$

Transferable Neural Architecture(T-NAS)

- **Algorithm**

Algorithm 1: T-NAS: Transferable Neural Architecture Search

Input: Meta-train dataset $\mathcal{D}_{\text{meta-train}}$, learning rate α_{inner}, α_{outer}, β_{inner} and β_{outer}.

Randomly initialize architecture parameter θ and network weights w.
while *not done* **do**
  Sample batch of tasks $\{\mathcal{T}\}$ in $\mathcal{D}_{\text{meta-train}}$;
  for $\mathcal{T}_i \in \{\mathcal{T}\}$ **do**
    Get datapoints $\mathcal{T}_i^s$;
    Compute $\mathcal{L}(g(\mathcal{T}_i^s; \theta_i^m, w_i^m))$ according to the standard cross-entropy loss;
    Alternatively update w_i^m and θ_i^m with Eq. (5) for M steps;
    Get datapoints $\mathcal{T}_i^q$ for meta-searcher;
  end
  Alternatively update $\widetilde{w}$ and $\widetilde{\theta}$ with Eq. (6);
end

Transferable Neural Architecture(T-NAS)

- **Optimization**

$$[w_i^{m+1}; \theta_i^{m+1}] = [w_i^m; \theta_i^m] - \boldsymbol{\eta}_{inner} \nabla_{[w_i^m; \theta_i^m]} \mathcal{L}(g(\mathcal{T}_i^s; \theta_i^m, w_i^m))$$

$$[\tilde{w}; \tilde{\theta}] = [\tilde{w}; \tilde{\theta}] - \boldsymbol{\eta}_{outer} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{[w_i^M; \theta_i^M]} \mathcal{L}(g(\mathcal{T}_i^q; \theta_i^M, w_i^M))$$

Algorithm 2: Adaptation and decoding

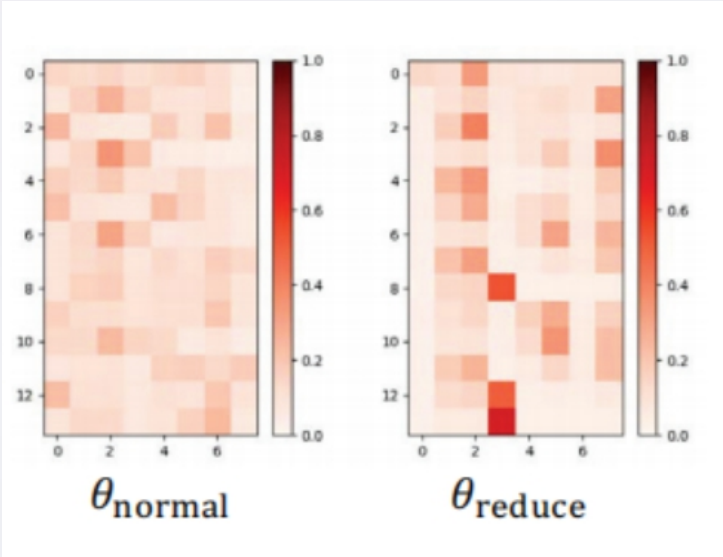
Input: Meta-test dataset $\mathcal{D}_{\text{meta-test}}$, learning rate α_{inner} and β_{inner}.
Output: The task-specific architecture θ_i^* for the i-th task $\mathcal{T}_i$.

1 Obtain the specific task $\mathcal{T}_i$ from $\mathcal{D}_{\text{meta-test}}$;
2 Update w_i^m and θ_i^m for M step with Eq. (7) and get θ_i^M;
3 Decoding θ_i^M to task-specific architecture θ_i^* by following the method in Liu et al. (2018b).

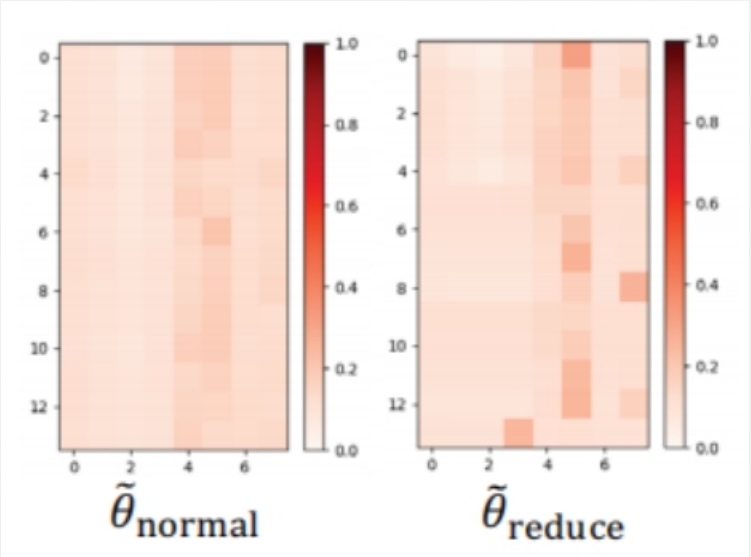
Transferable Neural Architecture(T-NAS)

- **Result**

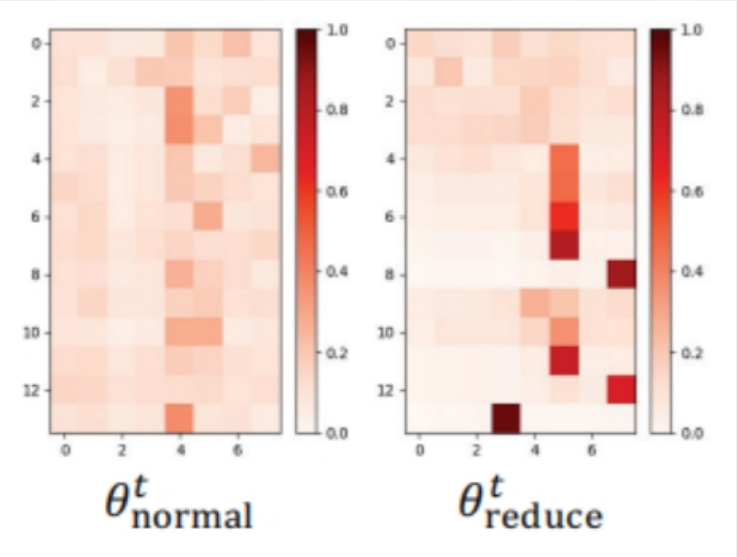
Auto-MAML



T-NAS



specific task



Transferable Neural Architecture(T-NAS)

- **Result**

Table 2: 5-way accuracy results on the Omniglot dataset.

Methods	1-shot	5-shot
Siamese Nets (Koch et al., 2015)	97.3%	98.4%
Matching nets (Vinyals et al., 2016)	98.1%	98.9%
Neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%
Memory Mod. (Kaiser et al., 2017)	98.4%	99.6%
Meta-SGD (Li et al., 2017)	**99.53 ± 0.26%**	99.93 ± 0.09%
MAML (Finn et al., 2017)	98.7 ± 0.4%	99.9 ± 0.1%
MAML++ (Antoniou et al., 2019)	99.47%	99.93%
Auto-MAML (ours)	98.95 ± 0.38%	99.91 ± 0.09%
T-NAS (ours)	99.16 ± 0.34%	99.93 ± 0.07%
T-NAS++ (ours)	99.35 ± 0.32%	**99.93 ± 0.07%**

Table 3: 5-way accuracy results on Mini-Imagenet.

Methods	Arch.	#Param.	1-shot	5-shot
Matching nets (Vinyals et al., 2016)	4CONV	32.9K	43.44 ± 0.77%	55.31 ± 0.73%
ProtoNets (Snell et al., 2017)	4CONV	32.9K	49.42 ± 0.78%	68.20 ± 0.66%
Meta-LSTM (Ravi & Larochelle, 2017)	4CONV	32.9K	43.56 ± 0.84%	60.60 ± 0.71%
Bilevel (Franceschi et al., 2018)	4CONV	32.9K	50.54 ± 0.85%	64.53 ± 0.68%
CompareNets (Sung et al., 2018)	4CONV	32.9K	50.44 ± 0.82%	65.32 ± 0.70%
LLAMA (Grant et al., 2018)	4CONV	32.9K	49.40 ± 1.83%	-
MAML (Finn et al., 2017)	4CONV	32.9K	48.70 ± 1.84%	63.11 ± 0.92%
MAML (first-order) (Finn et al., 2017)	4CONV	32.9K	48.07 ± 1.75%	63.15 ± 0.91%
MAML++ (Antoniou et al., 2019)	4CONV	32.9K	52.15 ± 0.26%	68.32 ± 0.44%
Auto-Meta (small) (Kim et al., 2018)	Cell	28/28 K	49.58 ± 0.20%	65.09 ± 0.24%
Auto-Meta (large) (Kim et al., 2018)	Cell	98.7/94.0 K	51.16 ± 0.17%	69.18 ± 0.14%
BASE (Softmax) (Shaw et al., 2018)	Cell	1200K	-	65.40 ± 0.74%
BASE (Gumbel-Softmax) (Shaw et al., 2018)	Cell	1200K	-	66.20 ± 0.70%
Auto-MAML (ours)	Cell	23.2/26.1 K	51.23 ± 1.76%	64.10 ± 1.12%
T-NAS (ours)	Cell	24.3/26.5 K*	52.84 ± 1.41%	67.88 ± 0.92%
T-NAS++ (ours)	Cell	24.3/26.5 K*	**54.11 ± 1.35%**	**69.59 ± 0.85%**

* means the average parameters of architectures for evaluation.

Table 4: 5-way accuracy results on FC100.

Methods	1-shot	5-shot	10-shot
MAML (Finn et al., 2017)	38.1 ± 1.7%	50.4 ± 1.0%	56.2 ± 0.8%
MAML++ (Antoniou et al., 2019)	38.7 ± 0.4%	52.9 ± 0.4%	58.8 ± 0.4%
Auto-MAML (ours)	38.8 ± 1.8%	52.2 ± 1.2%	57.5 ± 0.8%
T-NAS (ours)	39.7 ± 1.4%	53.1 ± 1.0%	58.9 ± 0.7%
T-NAS++ (ours)	**40.4 ± 1.2%**	**54.6 ± 0.9%**	**60.2 ± 0.7%**

Transferable Neural Architecture(T-NAS)

- **Result**

Table 5: 200-shot, 50-query, 10-way accuracy results of supervised learning on Mini-Imagenet.

Methods	200-shot	Time
Random	$61.20 \pm 0.09\%$	N/A
S1	$64.84 \pm 0.04\%$	266 min
S2	$62.99 \pm 0.05\%$	N/A
T-NAS (ours)	$\mathbf{64.23 \pm 0.05\%}$	**5 min**

Summary and reflection

- **Summary**

The existing NAS methods only target a specific task. This paper proposed a T-NAS based on meta-learning, which combines MAML and DARTS. T-NAS learns a meta-learning is able to adapt to a new task quickly through a few gradient steps.

Summary and reflection

- **Reflection**

1. T-NAS learns the parameter $\tilde{\theta}$ and $\tilde{w}$, while when training the new task, T-NAS doesn't use parameter $\tilde{w}$.
2. The transferability of NAS for tasks from different task distributions.
3. Meta-learning can be applied to gradient descent optimization, even if it is bilevel.