Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML

Aniruddh Raghu, Maithra Raghu , Samy Bengio, Oriol Vinyals

ICLR 2020

Conventional Machine Learning

Datasets $D = \{(x_1, y_1), \dots (x_N, y_N)\}$

Model $\hat{y} = f_\theta(x)$

Train Target $\theta^* = arg \min_\theta \mathcal{L}\,(D; \theta, w)$

Conventional Machine Learning

Datasets $D = \{(x_1, y_1), \ldots (x_N, y_N)\}$

Model $\hat{y} = f_\theta(x)$

Train Target $\theta^* = arg \min_\theta \mathcal{L}(D; \theta, \omega)$

ω is a condition => ‘how to learn’ θ that depended on this solution.
e.g. choice of optimizer for θ or function class for f, which we denote by ω.

Meta-Learning: Task-Distribution View

Meta train Dataset $\mathcal{D}_{source} = \left\{\left(\mathcal{D}_{source}^{train}, \mathcal{D}_{source}^{val}\right)^{(i)}\right\}_{i=1}^{M}$

Meta test Dataset $\mathcal{D}_{target} = \left\{\left(\mathcal{D}_{target}^{train}, \mathcal{D}_{target}^{val}\right)^{(i)}\right\}_{i=1}^{M}$

Meta train target $\omega^{*} = arg \max_{\omega} logp\,(\omega|\mathcal{D}_{source})$ meta-initialization

Meta test target $\theta^{*} = arg \max_{\theta} logp\,(\theta|\omega^{*}, \mathcal{D}_{target}^{train\ (i)})$ adaptation

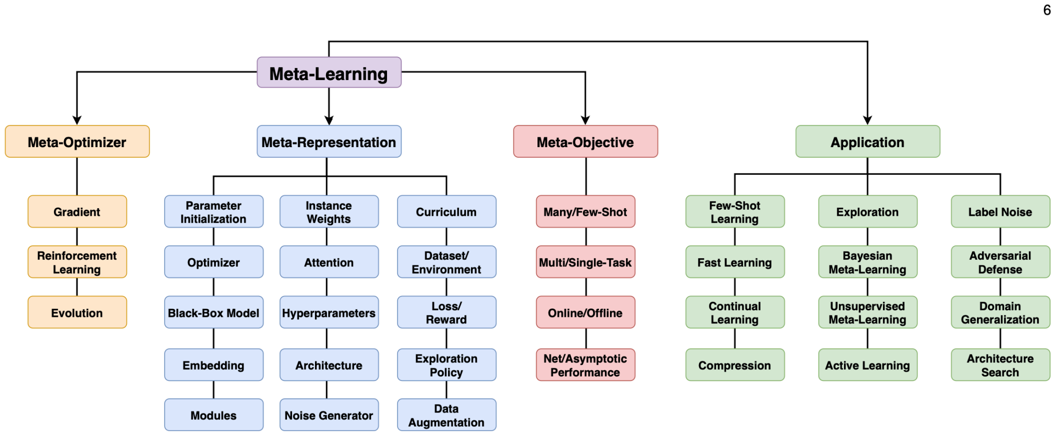
Meta-Learning: Task-Distribution View

Meta train Dataset $\mathcal{D}_{source} = \left\{\left(\mathcal{D}^{train}_{source}, \mathcal{D}^{val}_{source}\right)^{(i)}\right\}_{i=1}^{M}$

Meta test Dataset $\mathcal{D}_{target} = \left\{\left(\mathcal{D}^{train}_{target}, \mathcal{D}^{val}_{target}\right)^{(i)}\right\}_{i=1}^{M}$

Meta train target $\boxed{\omega^* = \arg\max_{\omega} \log p\left(\omega|\mathcal{D}_{source}\right)}$

Meta test target $\theta^* = \arg\max_{\theta} \log p\left(\theta|\omega^*, \mathcal{D}^{train}_{target}{}^{(i)}\right)$



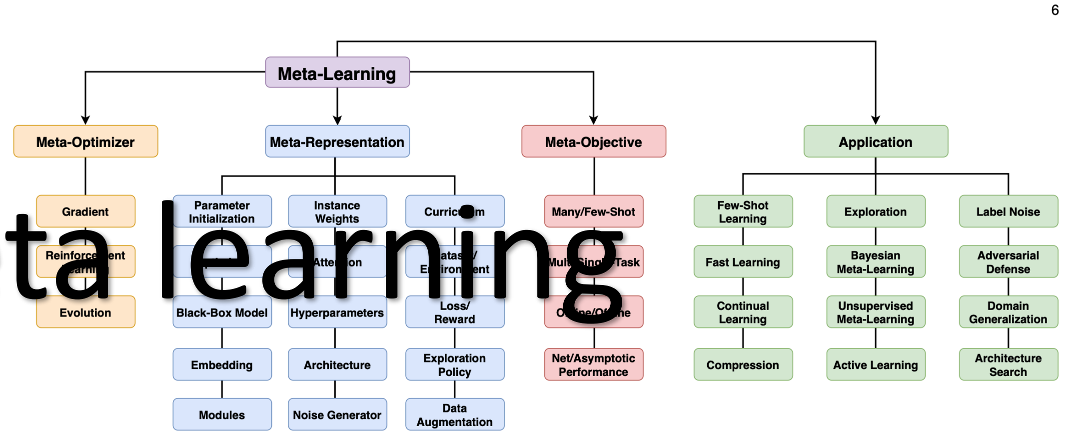
Meta-Learning: Task-Distribution View

Meta train Dataset $\mathcal{D}_{source} = \left\{ \left(\mathcal{D}_{source}^{train}, \mathcal{D}_{source}^{val} \right)^{(i)} \right\}_{i=1}^{M}$

Meta test Dataset $\mathcal{D}_{target} = \left\{ \left(\mathcal{D}_{target}^{train}, \mathcal{D}_{target}^{val} \right)^{(i)} \right\}_{i=1}^{M}$

Meta train target $\boxed{\omega^* = \arg\max_{\omega} \log p\left(\omega | \mathcal{D}_{source}\right)}$

Meta test target $\theta^* = \arg\max_{\theta} \log p\left(\theta | \omega^*, \mathcal{D}_{target}^{train\ (i)}\right)$



Model agnostic meta learning

Few Shot Learning

Many tasks, little data for each task

Few Shot Learning

(Optimization-based) Meta Learning Algorithms

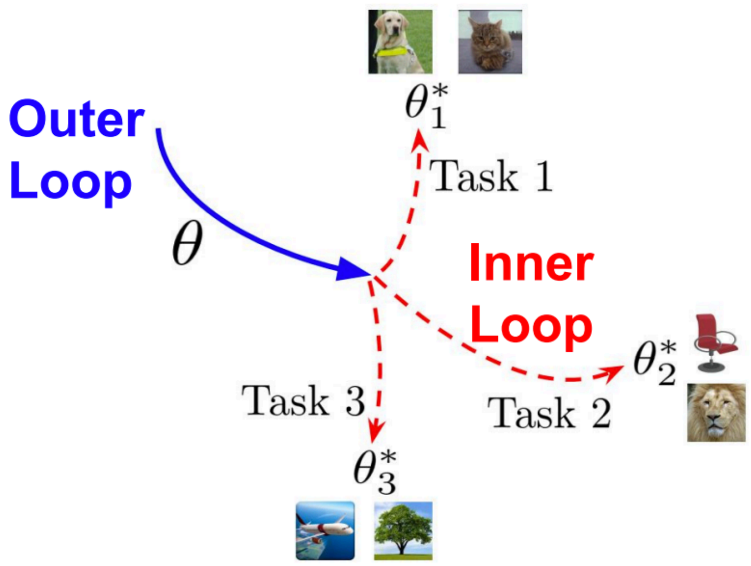
- Model Agnostic Meta Learning, (Finn et al), ICML 2017

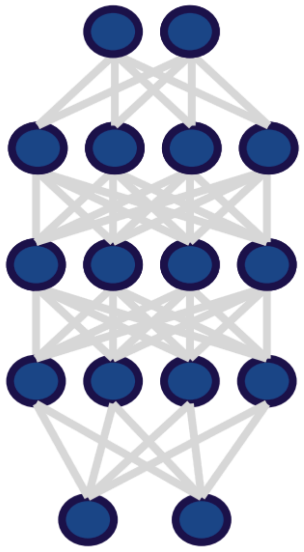
Few Shot Learning

(Optimization-based) Meta Learning Algorithms

● Model Agnostic Meta Learning, (Finn et al), ICML 2017

Outer Loop: meta-initialization; Inner Loop: adaptation

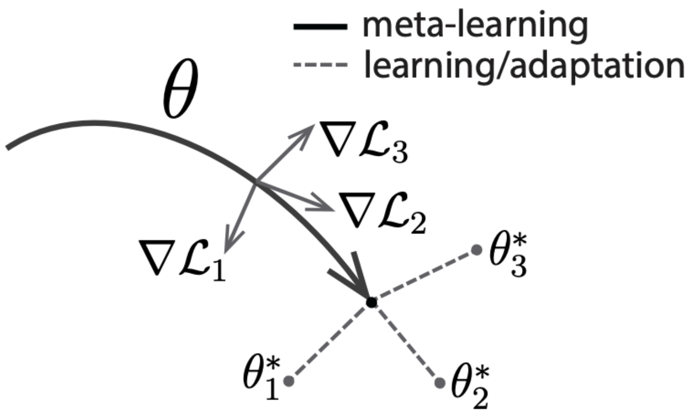




Few Shot Learning

(Optimization-based) Meta Learning Algorithms

- Model Agnostic Meta Learning, (Finn et al), ICML 2017



$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)},\mathbf{y}^{(j)}\sim\mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1-\mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)}))$$

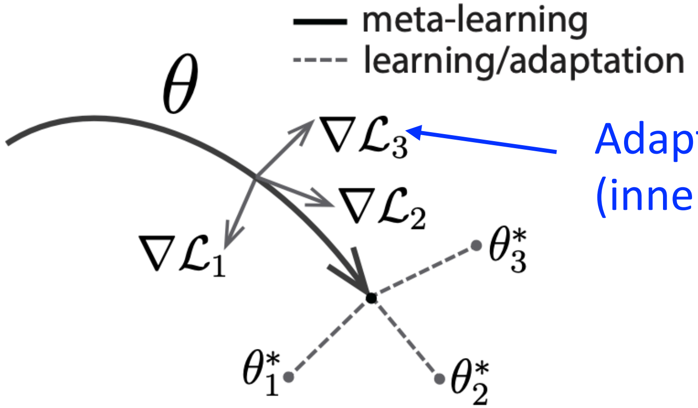
Algorithm 2 MAML for Few-Shot Supervised Learning

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: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
11: **end while**

Few Shot Learning

(Optimization-based) Meta Learning Algorithms

- Model Agnostic Meta Learning, (Finn et al), ICML 2017



Adaption
(inner loop)

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)},\mathbf{y}^{(j)}\sim\mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1-\mathbf{y}^{(j)})\log(1-f_\phi(\mathbf{x}^{(j)}))$$

Algorithm 2 MAML for Few-Shot Supervised Learning

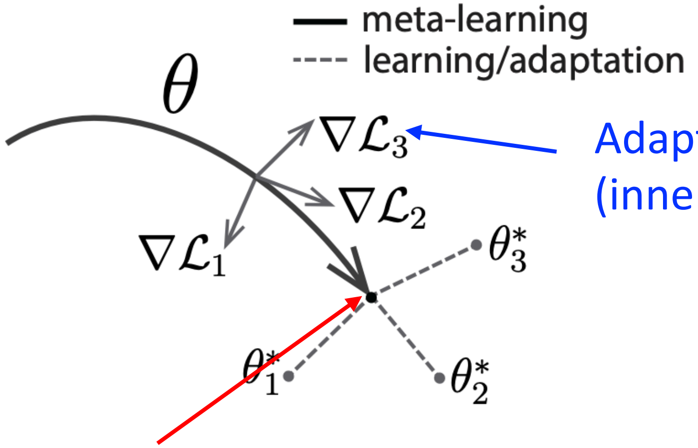
Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
    Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
  end for
  Update $\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each $\mathcal{D}'_i$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
end while

Few Shot Learning

(Optimization-based) Meta Learning Algorithms

- Model Agnostic Meta Learning, (Finn et al), ICML 2017



Adaption (inner loop)

Meta initialization (outer loop)

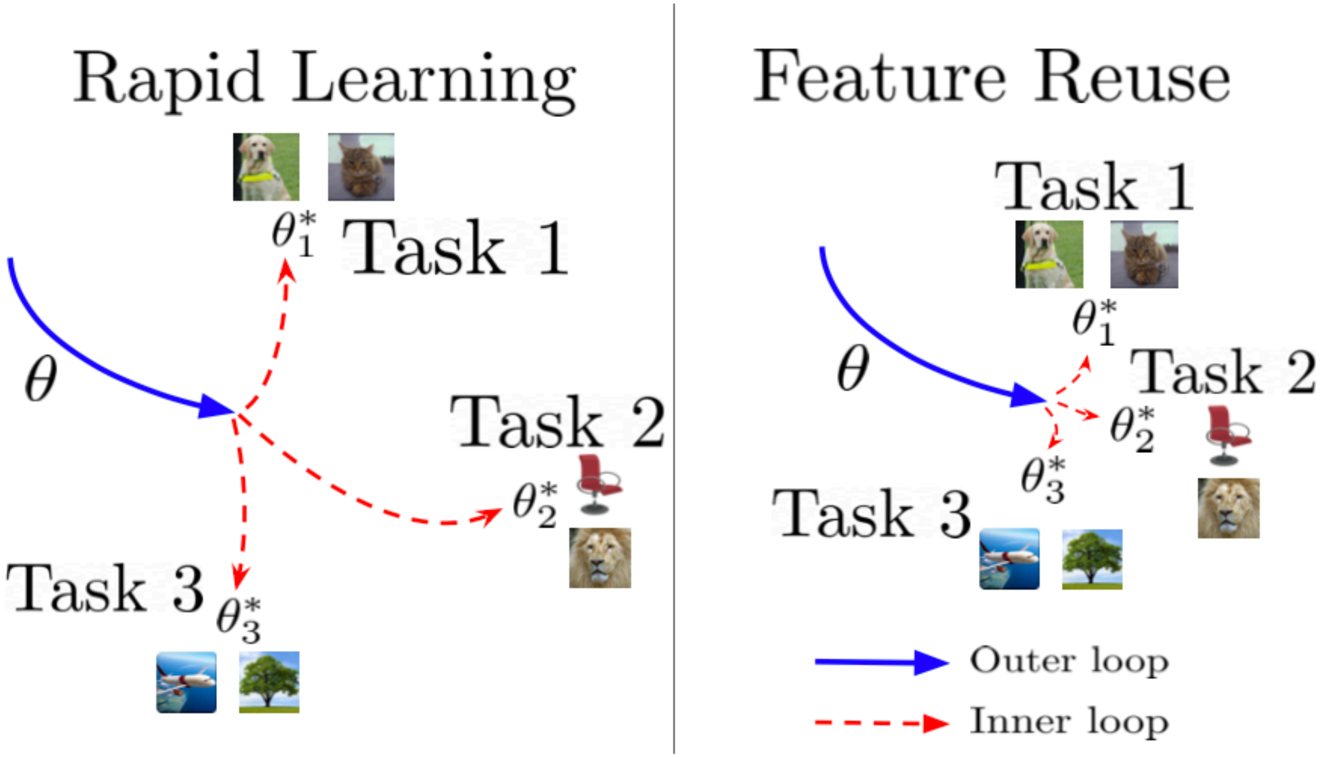
$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)},\mathbf{y}^{(j)}\sim\mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1-\mathbf{y}^{(j)})\log(1-f_\phi(\mathbf{x}^{(j)}))$$

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
  end for
  Update $\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
end while

Rapid Learning or Feature Reuse?

Outer Loop: meta-initialization; Inner Loop: adaptation



In feature reuse
=> little task-specific adaptation occurs

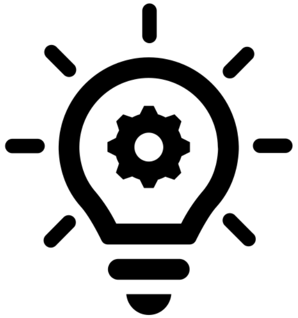
Rapid Learning or Feature Reuse?

Q : How do hidden representations behave (during inner loop)?

Rapid Learning or Feature Reuse?

Q : How do hidden representations behave (during inner loop)?

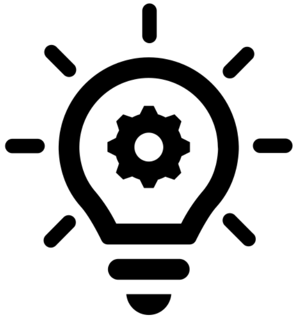
A: Measure Representation Similarity



Rapid Learning or Feature Reuse?

Q : How do hidden representations behave (during inner loop)?

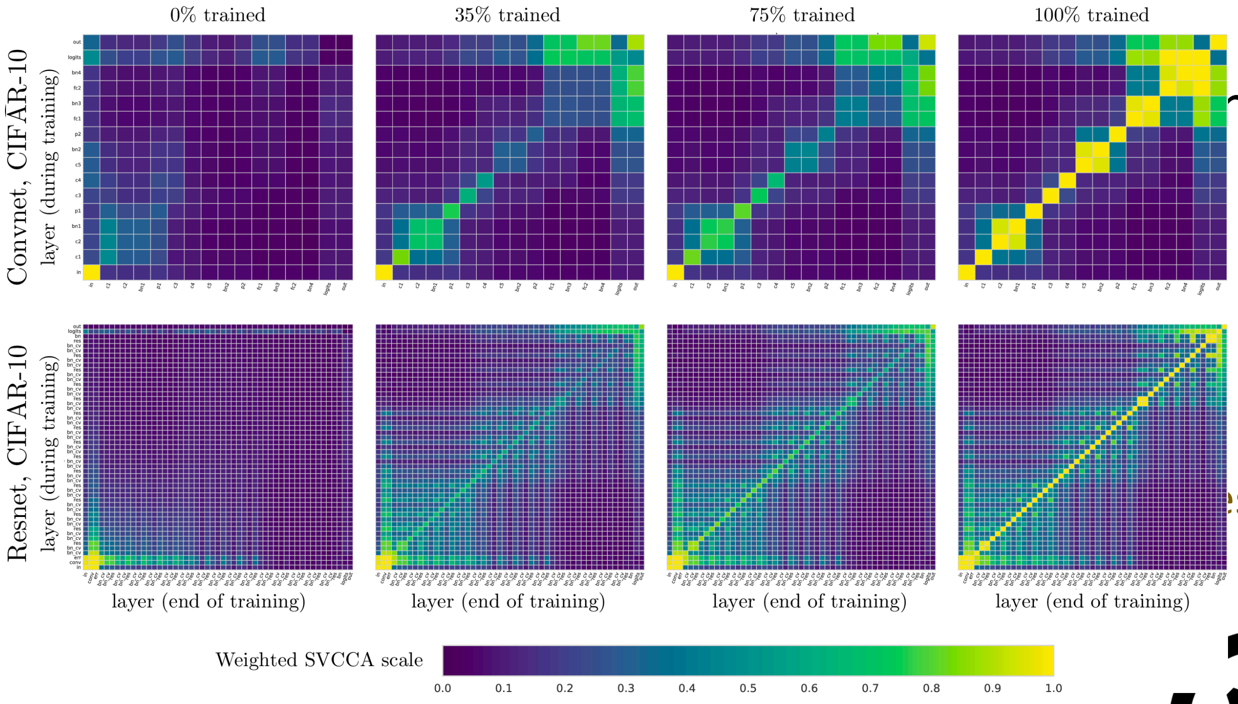
A: Measure Representation Similarity



SVCCA

Rapid Learning or Feature Reuse?

Q : H



nner loop)?

esentation Similarity

SVCCA

Rapid Learning or Feature Reuse?

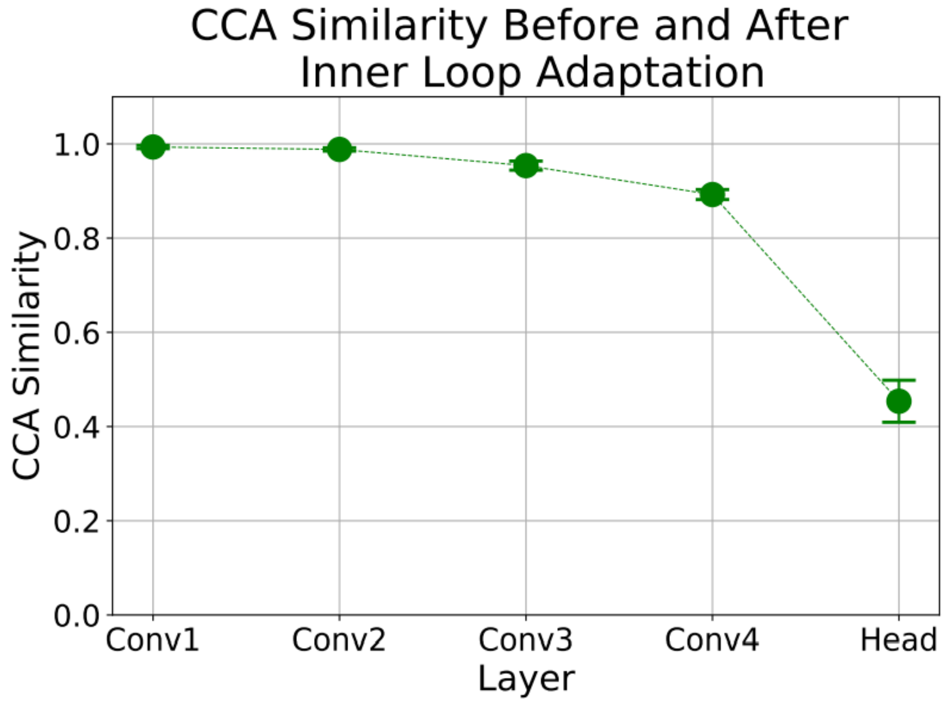
- Freezing Layer Representations
  - Body = > the earlier layers
  - Head => the final layer

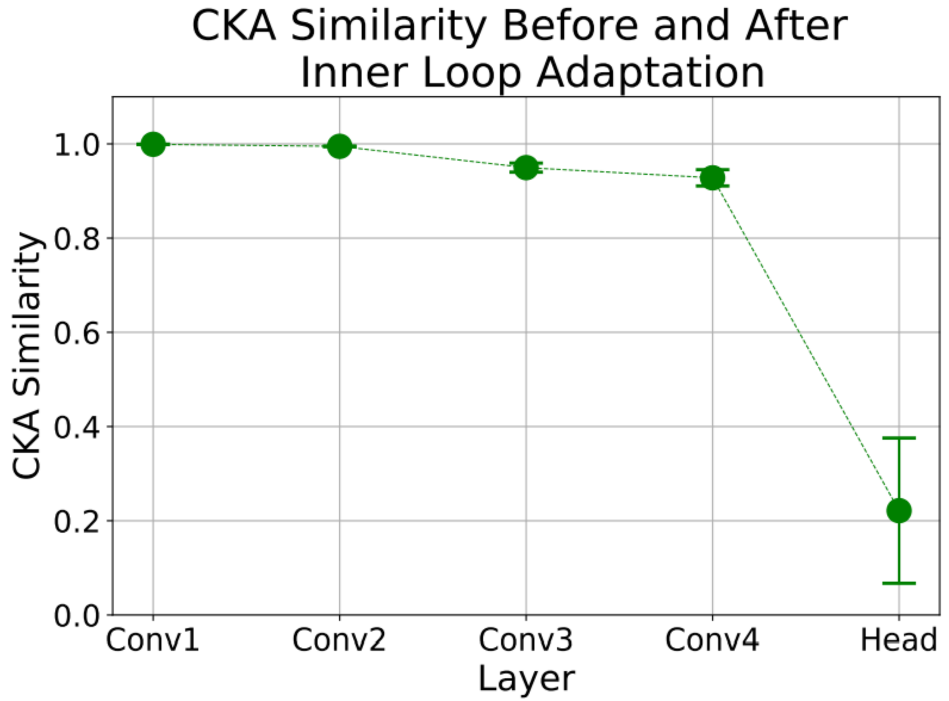
- At Test time => Freeze a contiguous subset of layers of the network

(No Adaptation)

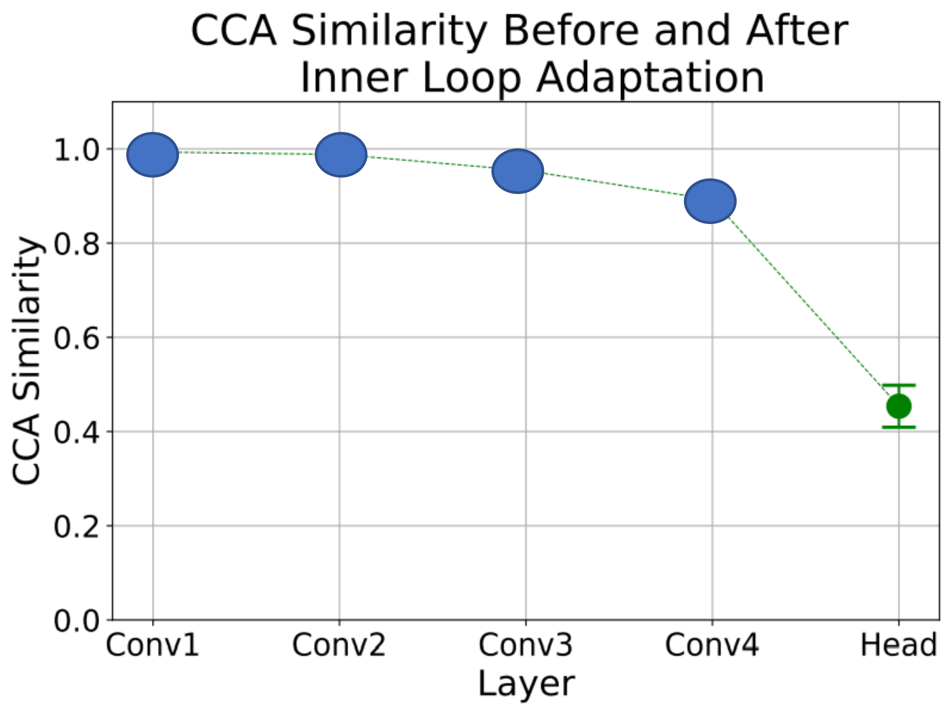
- The frozen layers are not updated at all to the test time task , and must reuse the features learned by the meta-initialization.

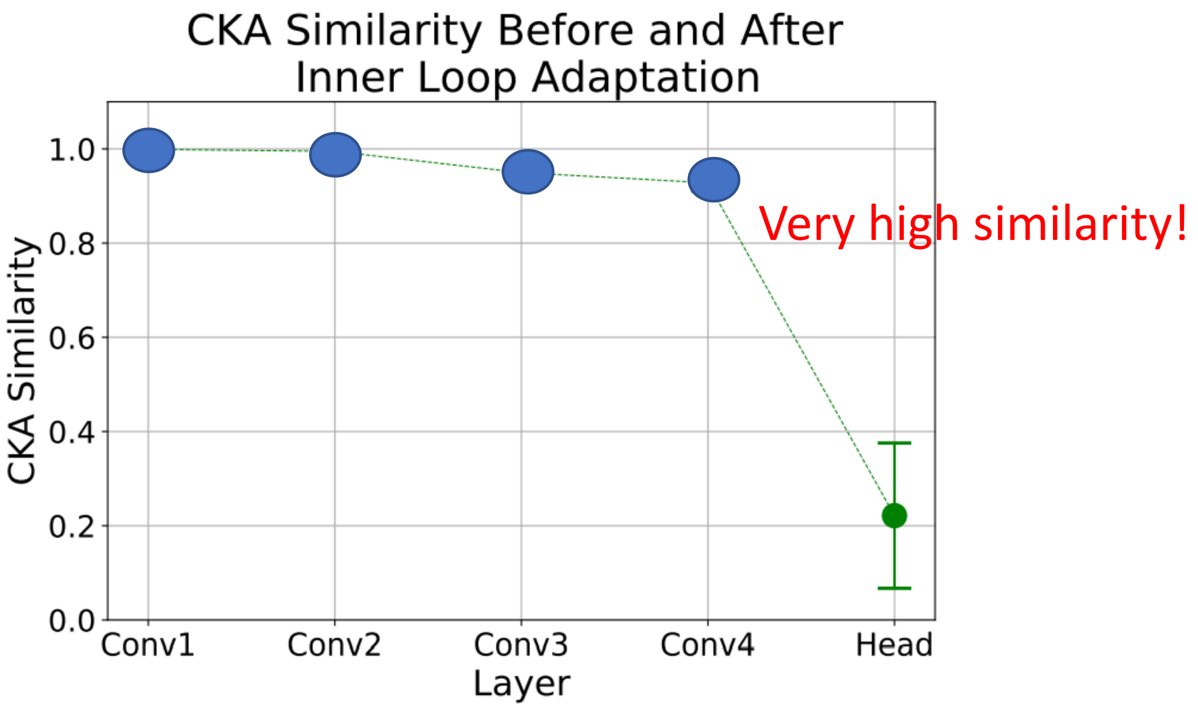
Rapid Learning or Feature Reuse?





Rapid Learning or Feature Reuse?





Rapid Learning or Feature Reuse?

Freeze layers	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
None	46.9 ± 0.2	63.1 ± 0.4
1	46.5 ± 0.3	63.0 ± 0.6
1,2	46.4 ± 0.4	62.6 ± 0.6
1,2,3	46.3 ± 0.4	61.2 ± 0.5
1,2,3,4	46.3 ± 0.4	61.0 ± 0.6

Table 1: **Freezing successive layers (preventing inner loop adaptation) does not affect accuracy, supporting feature reuse.** To test the amount of feature reuse happening in the inner loop adaptation, we test the accuracy of the model when we freeze (prevent inner loop adaptation) a contiguous block of layers at test time. We find that freezing even all four convolutional layers of the network (all layers except the network head) hardly affects accuracy. This strongly supports the feature reuse hypothesis: layers don't have to change rapidly at adaptation time; they already contain good features from the meta-initialization.

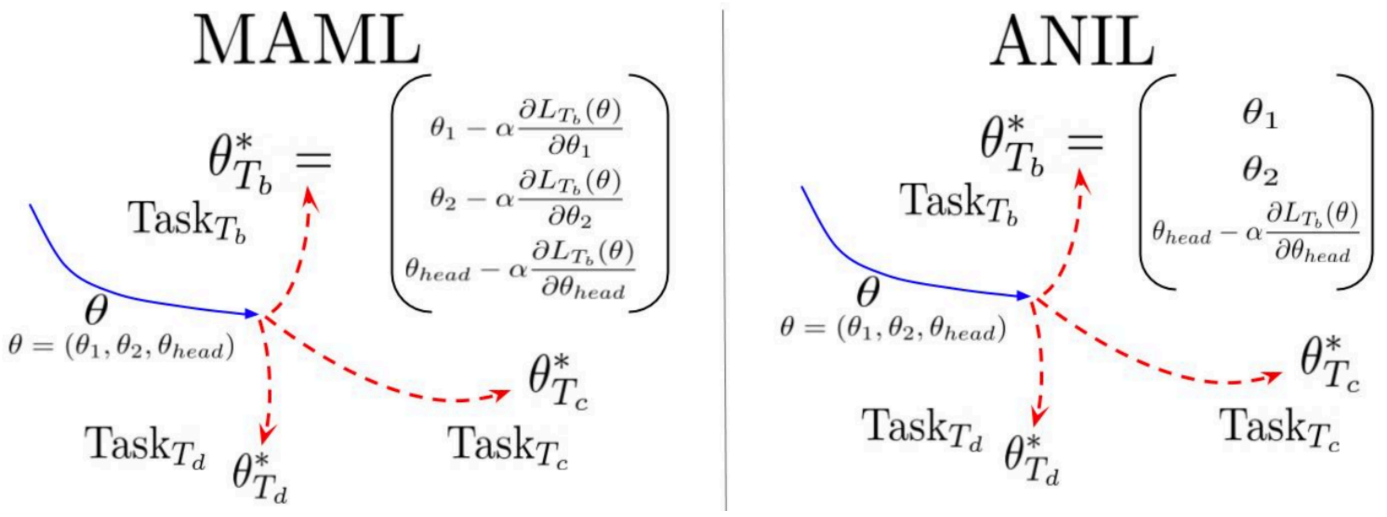
Rapid Learning or Feature Reuse?

Freeze layers	MiniImageNet-5way-1shot		MiniImageNet-5way-5shot
None	46.9 ± 0.2		63.1 ± 0.4
1	46.5 ± 0.3		63.0 ± 0.6
1,2	46.4 ± 0.4	trivial	62.6 ± 0.6
1,2,3	46.3 ± 0.4		61.2 ± 0.5
1,2,3,4	46.3 ± 0.4		61.0 ± 0.6

Table 1: **Freezing successive layers (preventing inner loop adaptation) does not affect accuracy, supporting feature reuse.** To test the amount of feature reuse happening in the inner loop adaptation, we test the accuracy of the model when we freeze (prevent inner loop adaptation) a contiguous block of layers at test time. We find that freezing even all four convolutional layers of the network (all layers except the network head) hardly affects accuracy. This strongly supports the feature reuse hypothesis: layers don't have to change rapidly at adaptation time; they already contain good features from the meta-initialization.

ANIL: Almost No Inner Loop Algorithm

ANIL: Almost No Inner Loop Algorithm



- **Removes inner loop** for all but head of network
- Much **more computationally efficient, same performance**
- **Insights** into meta learning and few shot learning

ANIL: Almost No Inner Loop Algorithm

Method	Omniglot-20way-1shot	Omniglot-20way-5shot	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML	93.7 ± 0.7	96.4 ± 0.1	46.9 ± 0.2	63.1 ± 0.4
ANIL	96.2 ± 0.5	98.0 ± 0.3	46.7 ± 0.4	61.5 ± 0.5

Method	HalfCheetah-Direction	HalfCheetah-Velocity	2D-Navigation
MAML	170.4 ± 21.0	-139.0 ± 18.9	-20.3 ± 3.2
ANIL	363.2 ± 14.8	-120.9 ± 6.3	-20.1 ± 2.3

Table 2: **ANIL matches the performance of MAML on few-shot image classification and RL.** On benchmark few-shot classification tasks MAML and ANIL have comparable accuracy, and also comparable average return (the higher the better) on standard RL tasks (Finn et al., 2017).

ANIL: Almost No Inner Loop Algorithm

	Training: 5way-1shot			Training: 5way-5shot		
	Mean (s)	Median (s)	Speedup	Mean (s)	Median (s)	Speedup
MAML	0.15	0.13	1	0.68	0.67	1
First Order MAML	0.089	0.083	1.69	0.40	0.39	1.7
ANIL	0.084	0.072	1.79	0.37	0.36	1.84

	Inference: 5way-1shot			Inference: 5way-5shot		
	Mean (s)	Median (s)	Speedup	Mean (s)	Median (s)	Speedup
MAML	0.083	0.078	1	0.37	0.36	1
ANIL	0.020	0.017	4.15	0.076	0.071	4.87

Table 6: **ANIL offers significant computational speedup over MAML, during both training and inference.** Table comparing execution times and speedups of MAML, First Order MAML, and ANIL during training (above) and inference (below) on MiniImageNet domains. Speedup is calculated relative to MAML's execution time. We see that ANIL offers noticeable speedup over MAML, as a result of removing the inner loop almost completely. This permits faster training and inference.

Computational benefit of ANIL

- average speedup of 1.7x per training iteration over MAML.
- average speedup of 4.1x per inference iteration over MAML.

ANIL: Almost No Inner Loop Algorithm

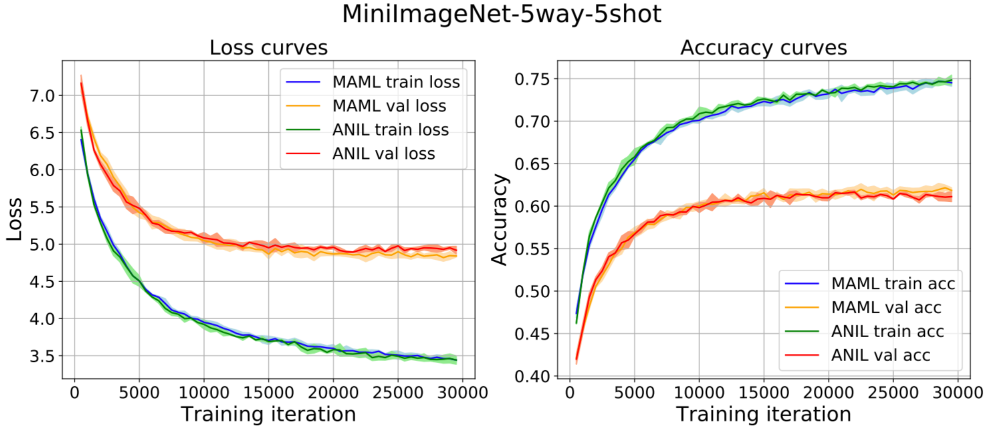


Figure 5: **MAML and ANIL learn very similarly**. Loss and accuracy curves for MAML and ANIL on MiniImageNet-5way-5shot, illustrating how MAML and ANIL behave similarly through the training process.

Model Pair	CCA Similarity	CKA Similarity
MAML-MAML	0.51	0.83
ANIL-ANIL	0.51	0.86
ANIL-MAML	0.50	0.83

Table 3: **MAML and ANIL models learn comparable representations.** Comparing CCA/CKA similarity scores of the of MAML-ANIL representations (averaged over network body), and MAML-MAML and ANIL-ANIL similarity scores (across different random seeds) shows algorithmic differences between MAML/ANIL does not result in vastly different types of features learned.

NIL: No Inner Loop Algorithm

NIL

- Train a model with ANIL/MAML algorithm
- At test time, remove the head of the trained model
- compute cosine similarities

Method	Omniglot-20way-1shot	Omniglot-20way-5shot	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML	93.7 ± 0.7	96.4 ± 0.1	46.9 ± 0.2	63.1 ± 0.4
ANIL	96.2 ± 0.5	98.0 ± 0.3	46.7 ± 0.4	61.5 ± 0.5
NIL	96.7 ± 0.3	98.0 ± 0.04	48.0 ± 0.7	62.2 ± 0.5

Conclusion

MAML => ~~Rapid Learning~~ or **Feature Reuse?**