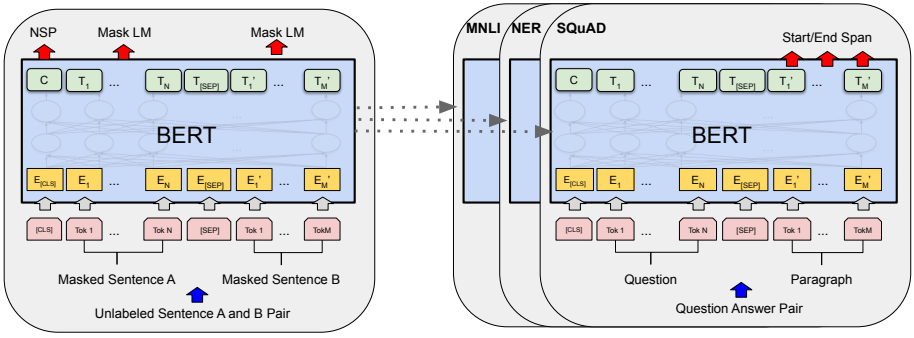
# Transfer Learning and Meta Learning

Tao LIN

May 18, 2023

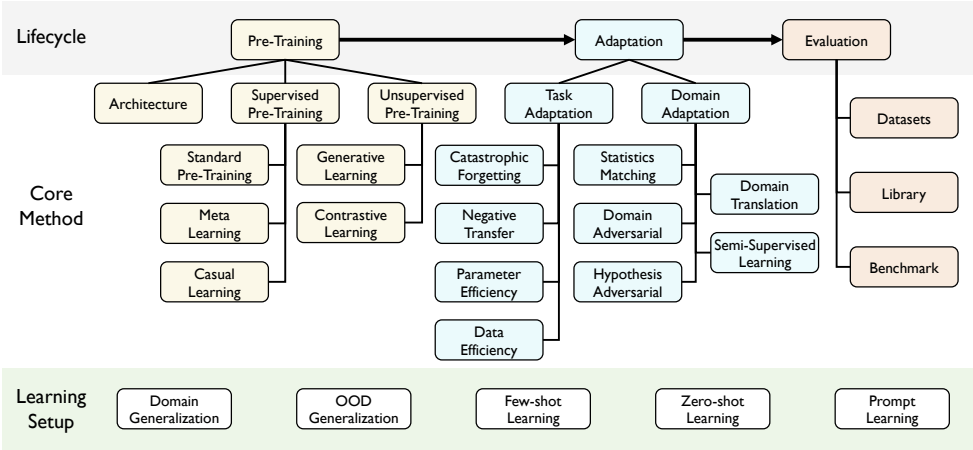
# Table of Contents




**Transfer Learning**: reuse pre-trained models by fine-tuning it for downstream tasks

[1] *Devlin et al.* Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

# Roadmap




[8] *Jiang et al.* Transferability in deep learning: A survey. 2022.

# Table of Contents

# Table of Contents

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data
- large models

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data
- large models
- trained till convergence using IID sampling

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data
- large models
- trained till convergence using IID sampling

But what if, for example,

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data
- large models
- trained till convergence using IID sampling

But what if, for example,

- we don't have a large dataset,

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data
- large models
- trained till convergence using IID sampling

But what if, for example,

- we don't have a large dataset,
- the data has a long tail,

The broad generalization of DNNs (supervised learning) stems from

- large, diverse data
- large models
- trained till convergence using IID sampling

But what if, for example,

- we don't have a large dataset,
- the data has a long tail,
- want to continuously adapt and learn on the job.

# The probabilistic view of meta learning

1. Extract prior information $\theta$ from a set of meta-training tasks,

# The probabilistic view of meta learning

1. Extract prior information $\theta$ from a set of meta-training tasks,
2. Learning a new task uses this prior and small training set

   $\rightarrow$ to infer most likely posterior parameters $\phi$.

# Table of Contents

# The complete meta learning optimization

**Data formulation:**

# The complete meta learning optimization

**Data formulation:**

- A collection of $M$ meta-training tasks $\mathcal{T}_i$ drawn from $P(\mathcal{T})$.

# The complete meta learning optimization

**Data formulation:**

- A collection of $M$ meta-training tasks $\mathcal{T}_i$ drawn from $P(\mathcal{T})$.
- Each task $\mathcal{T}_i$ is associated with a dataset $\mathcal{D}_i$.

# The complete meta learning optimization

**Data formulation:**

- A collection of $M$ meta-training tasks $\mathcal{T}_i$ drawn from $P(\mathcal{T})$.
- Each task $\mathcal{T}_i$ is associated with a dataset $\mathcal{D}_i$.
- Two disjoint sets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{ts}$ can be sampled from $\mathcal{D}_i$.

# The complete meta learning optimization

**Data formulation:**

- A collection of $M$ meta-training tasks $\mathcal{T}_i$ drawn from $P(\mathcal{T})$.
- Each task $\mathcal{T}_i$ is associated with a dataset $\mathcal{D}_i$.
- Two disjoint sets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{ts}$ can be sampled from $\mathcal{D}_i$.

**Optimization procedure (meta-training):**

$$\underbrace{\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})}_{\text{outer-level}}, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\underbrace{\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr})}_{\text{inner-level}}, \mathcal{D}_i^{ts}\right). \tag{1}$$

# The complete meta learning optimization

**Data formulation:**

- A collection of $M$ meta-training tasks $\mathcal{T}_i$ drawn from $P(\mathcal{T})$.
- Each task $\mathcal{T}_i$ is associated with a dataset $\mathcal{D}_i$.
- Two disjoint sets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{ts}$ can be sampled from $\mathcal{D}_i$.

**Optimization procedure (meta-training):**

$$\underbrace{\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})}_{\text{outer-level}}, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\underbrace{\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr})}_{\text{inner-level}}, \mathcal{D}_i^{ts}\right). \tag{1}$$

**Deployment (meta-test):**

For a new meta-test task $\mathcal{T}_j \sim P(\mathcal{T})$,

# The complete meta learning optimization

**Data formulation:**

- A collection of $M$ meta-training tasks $\mathcal{T}_i$ drawn from $P(\mathcal{T})$.
- Each task $\mathcal{T}_i$ is associated with a dataset $\mathcal{D}_i$.
- Two disjoint sets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{ts}$ can be sampled from $\mathcal{D}_i$.

**Optimization procedure (meta-training):**

$$\underbrace{\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})}_{\text{outer-level}}, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\underbrace{\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr})}_{\text{inner-level}}, \mathcal{D}_i^{ts}\right). \tag{1}$$

**Deployment (meta-test):**

For a new meta-test task $\mathcal{T}_j \sim P(\mathcal{T})$,
we can achieve good generalization performance by using $\boldsymbol{\phi}_j = \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_j^{tr})$.

# Connection to other problems

- **Multi-task learning:**
- **AutoML:**
- **Neural Architecture Search:**

# Connection to other problems

- **Multi-task learning:**

  learn model with $\boldsymbol{\theta}^\star$ that solves multiple tasks $\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{ts})$.

- **AutoML:**

- **Neural Architecture Search:**

# Connection to other problems

- **Multi-task learning:**

  learn model with $\boldsymbol{\theta}^\star$ that solves multiple tasks $\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{ts})$.

- **AutoML:**

  $\boldsymbol{\theta}$ = hyperparameters, $\phi$ = network weights.

- **Neural Architecture Search:**

# Connection to other problems

- **Multi-task learning:**

  learn model with $\boldsymbol{\theta}^\star$ that solves multiple tasks $\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{ts})$.

- **AutoML:**

  $\boldsymbol{\theta}$ = hyperparameters, $\phi$ = network weights.

- **Neural Architecture Search:**

  $\boldsymbol{\theta}$ = architecture, $\phi$ = network weights.

# Some directions for meta learning

Recap the optimization procedure for meta learning

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})\,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right)\,.$$

# Some directions for meta learning

Recap the optimization procedure for meta learning

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})\,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right)\,.$$

Some recent research directions

- Black-box Adaptation:
- Non-parametric methods:
- Gradient-based meta learning.

# Some directions for meta learning

Recap the optimization procedure for meta learning

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})\,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right)\,.$$

Some recent research directions

- Black-box Adaptation:

  Training a NN (e.g. LSTM) to represent $\boldsymbol{\phi}_i = \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr})$.
- Non-parametric methods:
- Gradient-based meta learning.

# Some directions for meta learning

Recap the optimization procedure for meta learning

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta})\,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right)\,.$$

Some recent research directions

- Black-box Adaptation:

  Training a NN (e.g. LSTM) to represent $\boldsymbol{\phi}_i = \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr})$.

- Non-parametric methods:

  Use non-parametric learner, e.g., weighted KNN in latent space.

- Gradient-based meta learning.

# Table of Contents

# Model-Agnostic Meta-Learning (MAML) [2]

Recap the meta-optimization procedure:

$$\boldsymbol{\theta}^{\star} := \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} F(\boldsymbol{\theta}) \,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right) \,.$$

# Model-Agnostic Meta-Learning (MAML) [2]

Recap the meta-optimization procedure:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta}) \,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right) \,.$$

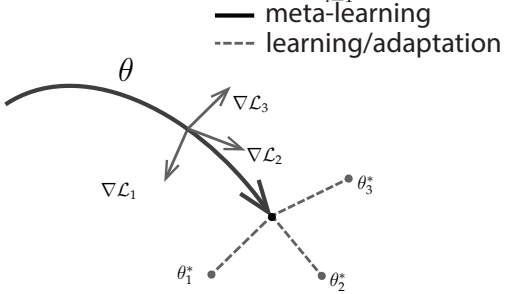



Figure: MAML finds a weight initialization $\boldsymbol{\theta}$ that allows for quick adaptation.

# Model-Agnostic Meta-Learning (MAML) [2]

Recap the meta-optimization procedure:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta}) \,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right) \,.$$

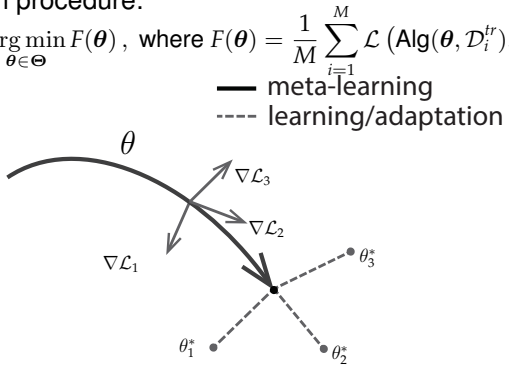



Figure: MAML finds a weight initialization $\boldsymbol{\theta}$ that allows for quick adaptation.

The inner level of MAML is the mini-batch SGD steps initialized at $\boldsymbol{\theta}$

$$\boldsymbol{\phi}_i := \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}) = \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}) \,. \tag{2}$$

# Model-Agnostic Meta-Learning (MAML) [2]

Recap the meta-optimization procedure:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} F(\boldsymbol{\theta}) \,, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}\right) \,.$$

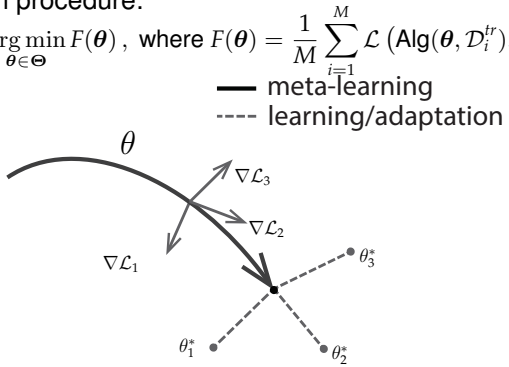



Figure: MAML finds a weight initialization $\boldsymbol{\theta}$ that allows for quick adaptation.

The inner level of MAML is the mini-batch SGD steps initialized at $\boldsymbol{\theta}$

$$\boldsymbol{\phi}_i := \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}) = \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}) \,. \tag{2}$$

Note that the update involves a gradient through the gradient.

# Challenge: second-order meta optimization can exhibit instabilities

Simplifying the notations by letting

$$\mathcal{L}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{ts}), \hat{\mathcal{L}}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{tr}), \mathsf{Alg}_i(\theta) := \mathsf{Alg}(\theta, \mathcal{D}_i^{tr}).$$

# Challenge: second-order meta optimization can exhibit instabilities

Simplifying the notations by letting

$$\mathcal{L}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{ts}) \,, \hat{\mathcal{L}}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{tr}) \,, \mathsf{Alg}_i(\boldsymbol{\theta}) := \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}) \,.$$

Considering the bi-level meta learning problem as

$$\boldsymbol{\theta}^\star := \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\min} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i \left( \mathsf{Alg}_i(\boldsymbol{\theta}) \right) \,, \text{ where } \boldsymbol{\phi}_i := \mathsf{Alg}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}_i(\boldsymbol{\theta}) \,.$$

# Challenge: second-order meta optimization can exhibit instabilities

Simplifying the notations by letting

$$\mathcal{L}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{ts}), \hat{\mathcal{L}}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{tr}), \mathsf{Alg}_i(\boldsymbol{\theta}) := \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}).$$

Considering the bi-level meta learning problem as

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right), \text{ where } \boldsymbol{\phi}_i := \mathsf{Alg}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}_i(\boldsymbol{\theta}).$$

By using the chain rule,

$$\frac{d\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)}{d\boldsymbol{\theta}} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}\nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\boldsymbol{\phi}\right)|_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}\nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right),$$

where

1. $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)$ is the gradient function evaluated at $\phi = \mathsf{Alg}_i(\boldsymbol{\theta})$,

# Challenge: second-order meta optimization can exhibit instabilities

Simplifying the notations by letting

$$\mathcal{L}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{ts}), \hat{\mathcal{L}}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{tr}), \mathsf{Alg}_i(\boldsymbol{\theta}) := \mathsf{Alg}(\boldsymbol{\theta}, \mathcal{D}_i^{tr}).$$

Considering the bi-level meta learning problem as

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right), \text{ where } \boldsymbol{\phi}_i := \mathsf{Alg}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}_i(\boldsymbol{\theta}).$$

By using the chain rule,

$$\frac{d\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)}{d\boldsymbol{\theta}} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}\nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\boldsymbol{\phi}\right)|_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}\nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right),$$

where

1. $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)$ is the gradient function evaluated at $\phi = \mathsf{Alg}_i(\boldsymbol{\theta})$,
2. $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$ is the Jacobian matrix (involve second order derivatives).

# Challenge: second-order meta optimization can exhibit instabilities

Considering the bi-level meta learning problem as

$$\boldsymbol{\theta}^{\star} := \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\min}\, \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right) \;,\; \text{where } \boldsymbol{\phi}_i := \mathsf{Alg}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}_i(\boldsymbol{\theta}) \,.$$

By using the chain rule,

$$\frac{d\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)}{d\boldsymbol{\theta}} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i\left(\boldsymbol{\phi}\right)|_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right) \;,$$

If $\mathsf{Alg}_i(\boldsymbol{\theta})$ is an iterative algorithm, then one way to compute $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$ is to propagate derivatives through the iterative process.

# Challenge: second-order meta optimization can exhibit instabilities

Considering the bi-level meta learning problem as

$$\boldsymbol{\theta}^{\star} := \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\min} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right) \ , \ \text{where } \boldsymbol{\phi}_i := \mathsf{Alg}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}_i(\boldsymbol{\theta}) \, .$$

By using the chain rule,

$$\frac{d\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)}{d\boldsymbol{\theta}} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i\left(\boldsymbol{\phi}\right)|_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right) \ ,$$

If $\mathsf{Alg}_i(\boldsymbol{\theta})$ is an iterative algorithm, then one way to compute $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$ is to propagate derivatives through the iterative process.

**Drawbacks:**

- Store the optimization path and thus memory insufficient.

# Challenge: second-order meta optimization can exhibit instabilities

Considering the bi-level meta learning problem as

$$\boldsymbol{\theta}^{\star} := \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\min}\, \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right) \;,\; \text{where } \boldsymbol{\phi}_i := \mathsf{Alg}_i(\boldsymbol{\theta}) = \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}_i(\boldsymbol{\theta}) \,.$$

By using the chain rule,

$$\frac{d\mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right)}{d\boldsymbol{\theta}} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i\left(\boldsymbol{\phi}\right)|_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} = \frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i\left(\mathsf{Alg}_i(\boldsymbol{\theta})\right) \;,$$

If $\mathsf{Alg}_i(\boldsymbol{\theta})$ is an iterative algorithm, then one way to compute $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$ is to propagate derivatives through the iterative process.

**Drawbacks:**

- Store the optimization path and thus memory insufficient.
- Computationally heavy.

# Attempts

- **First-order MAML (FOMAML)** [2]

# Attempts

- **First-order MAML (FOMAML)** [2]
  It ignores second derivative terms and sets $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \mathbf{I}$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i \left( \phi \right) |_{\boldsymbol{\phi} = \mathsf{Alg}_i(\boldsymbol{\theta})} \cdot$$

# Attempts

- **First-order MAML (FOMAML)** [2]
  It ignores second derivative terms and sets $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \mathbf{I}$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i \left(\phi\right) |_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} \cdot$$

- **Reptile** [10]

# Attempts

- **First-order MAML (FOMAML)** [2]
  It ignores second derivative terms and sets $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \mathbf{I}$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i \left(\phi\right) |_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} .$$

- **Reptile** [10]
  It uses the $\phi_i$ as targets and slowly moves $\boldsymbol{\theta}$ towards them:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \left(\boldsymbol{\theta} - \boldsymbol{\phi}_i\right) .$$

# Attempts

- **First-order MAML (FOMAML)** [2]
  It ignores second derivative terms and sets $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \mathbf{I}$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i \left(\phi\right) |_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} .$$

- **Reptile** [10]
  It uses the $\phi_i$ as targets and slowly moves $\boldsymbol{\theta}$ towards them:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \left(\boldsymbol{\theta} - \boldsymbol{\phi}_i\right) .$$

- **Implicit MAML (iMAML)** [12]

# Attempts

- **First-order MAML (FOMAML)** [2]
  It ignores second derivative terms and sets $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \mathbf{I}$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i \left(\boldsymbol{\phi}\right) |_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} \,.$$

- **Reptile** [10]
  It uses the $\boldsymbol{\phi}_i$ as targets and slowly moves $\boldsymbol{\theta}$ towards them:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \left(\boldsymbol{\theta} - \boldsymbol{\phi}_i\right) \,.$$

- **Implicit MAML (iMAML)** [12]
  - It uses a regularized algorithm, instead of $\mathsf{Alg}_i(\boldsymbol{\theta})$,

$$\mathsf{Alg}_i^\star(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\phi}'} \hat{\mathcal{L}}_i(\boldsymbol{\phi}') + \frac{\lambda}{2} \left\|\boldsymbol{\phi}' - \boldsymbol{\theta}\right\|^2 \,,$$

    where it works as an early stopping of MAML.

# Attempts

- **First-order MAML (FOMAML)** [2]
  It ignores second derivative terms and sets $\frac{d\mathsf{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \mathbf{I}$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i \left(\phi\right) |_{\boldsymbol{\phi}=\mathsf{Alg}_i(\boldsymbol{\theta})} \cdot$$

- **Reptile** [10]
  It uses the $\phi_i$ as targets and slowly moves $\boldsymbol{\theta}$ towards them:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\gamma}{M} \sum_{i=1}^{M} \left(\boldsymbol{\theta} - \boldsymbol{\phi}_i\right) .$$

- **Implicit MAML (iMAML)** [12]
  - It uses a regularized algorithm, instead of $\mathsf{Alg}_i(\boldsymbol{\theta})$,

$$\mathsf{Alg}_i^{\star}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\phi}'} \hat{\mathcal{L}}_i(\boldsymbol{\phi}') + \frac{\lambda}{2} \left\|\boldsymbol{\phi}' - \boldsymbol{\theta}\right\|^2 ,$$

    where it works as an early stopping of MAML.
  - It approximates the analytical solution of the Jacobian.

# More for iMAML (optional reading)

For the optimal solution of $\text{Alg}_i^\star(\boldsymbol{\theta})$, one can analytically compute the Jacobian as $\frac{d\text{Alg}_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \left(\mathbf{I} + \frac{1}{\lambda}\nabla^2_{\boldsymbol{\phi}}\hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1}$.

**Proof.**

The stationary point conditions imply that

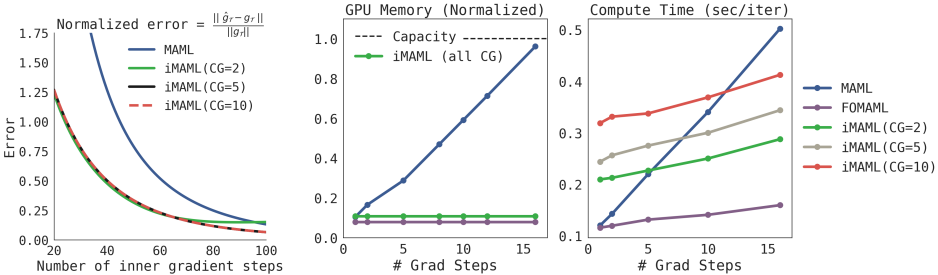
$$\nabla_\phi \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i) + \lambda(\boldsymbol{\phi}_i - \boldsymbol{\theta}) = 0 \rightarrow \boldsymbol{\phi}_i = \boldsymbol{\theta} - \frac{1}{\lambda}\nabla_\phi \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\,.$$

When the derivative exists, we have $\frac{d\boldsymbol{\phi}_i}{d\boldsymbol{\theta}} = \mathbf{I} - \frac{1}{\lambda}\nabla^2_{\boldsymbol{\phi}_i}\hat{\mathcal{L}}_i(\boldsymbol{\phi})\frac{d\boldsymbol{\phi}_i}{d\boldsymbol{\theta}}$. □

The meta-gradient $\mathbf{g}_i$ can be approximated:

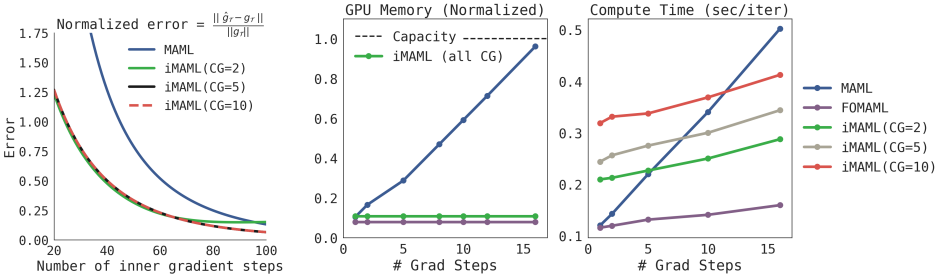
- approximated solution of $\text{Alg}_i^\star(\boldsymbol{\theta})$, i.e., $\boldsymbol{\phi}_i$.
- conjugate gradient algorithm to solve

$$\left\| \mathbf{g}_i - \left(\mathbf{I} + \frac{1}{\lambda}\nabla^2_{\boldsymbol{\phi}}\hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1} \nabla_{\boldsymbol{\phi}}\mathcal{L}_i\left(\boldsymbol{\phi}_i\right) \right\| \leq \delta'$$




(a) Synthetic example. (b) 20-way-5-shot Omniglot task.

Figure: Accuracy, computation, and memory trade-off.




(a) Synthetic example. (b) 20-way-5-shot Omniglot task.

Figure: Accuracy, computation, and memory trade-off.

| Algorithm | 5-way 1-shot | 5-way 5-shot | 20-way 1-shot | 20-way 5-shot |
|---|---|---|---|---|
| MAML [15] | $98.7 \pm 0.4\%$ | $\mathbf{99.9 \pm 0.1\%}$ | $95.8 \pm 0.3\%$ | $98.9 \pm 0.2\%$ |
| first-order MAML [15] | $98.3 \pm 0.5\%$ | $99.2 \pm 0.2\%$ | $89.4 \pm 0.5\%$ | $97.9 \pm 0.1\%$ |
| Reptile [43] | $97.68 \pm 0.04\%$ | $99.48 \pm 0.06\%$ | $89.43 \pm 0.14\%$ | $97.12 \pm 0.32\%$ |
| iMAML, GD (ours) | $99.16 \pm 0.35\%$ | $99.67 \pm 0.12\%$ | $94.46 \pm 0.42\%$ | $98.69 \pm 0.1\%$ |
| iMAML, Hessian-Free (ours) | $\mathbf{99.50 \pm 0.26\%}$ | $99.74 \pm 0.11\%$ | $\mathbf{96.18 \pm 0.36\%}$ | $\mathbf{99.14 \pm 0.1\%}$ |

Figure: Omniplot results.

# Case study: meta learning for continual learning [6, 7]

MAML-like algorithm solves few-shot learning.

# Case study: meta learning for continual learning [6, 7]

MAML-like algorithm solves few-shot learning.

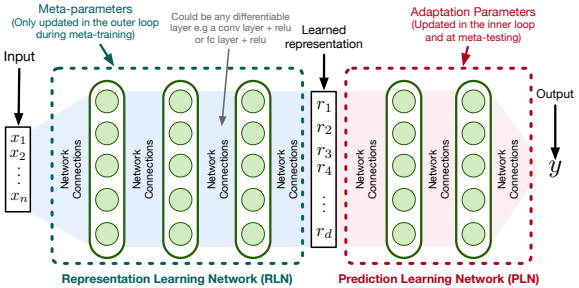
Now we look at how to use meta-learning to mitigate forgetting for better continual learning.

# Case study: meta learning for continual learning [6, 7]

MAML-like algorithm solves few-shot learning.

Now we look at how to use meta-learning to mitigate forgetting for better continual learning.

**MRCL** [6]:

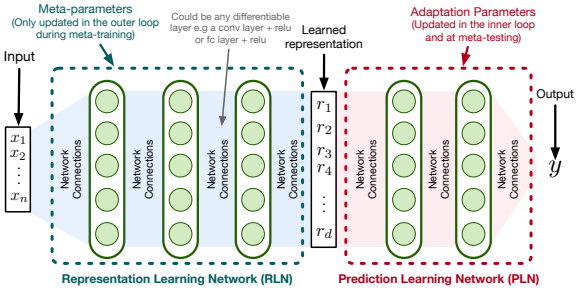


# Case study: meta learning for continual learning [6, 7]

MAML-like algorithm solves few-shot learning.

Now we look at how to use meta-learning to mitigate forgetting for better continual learning.

**MRCL** [6]:



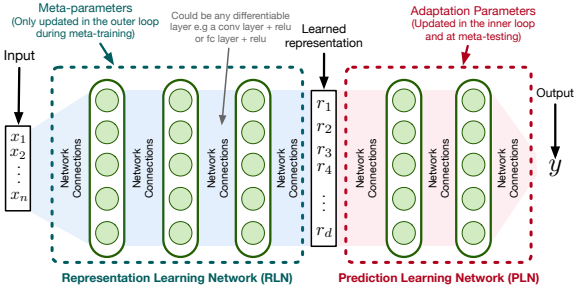

- trains the representation $f_{\theta}$ within the network with a meta-learning objective,

# Case study: meta learning for continual learning [6, 7]

MAML-like algorithm solves few-shot learning.

Now we look at how to use meta-learning to mitigate forgetting for better continual learning.

**MRCL** [6]:




- trains the representation $f_{\boldsymbol{\theta}}$ within the network with a meta-learning objective,
- optimizes through an online update for the later layers $g_{\mathbf{W}}$ used for task learning.

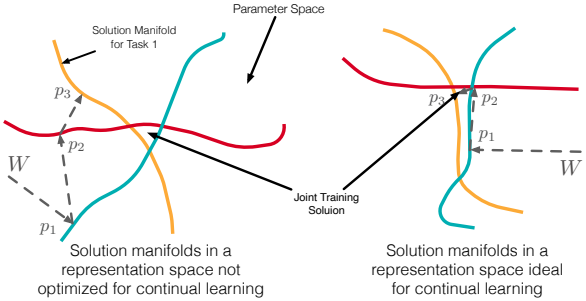



Figure: **Effect of the representation on continual learning** (an intuitive explanation), for a problem where targets are generated from three different distributions $p_1(Y|x), p_2(Y|x)$ and $p_3(Y|x)$.
The representation results in different solution manifolds for the three distributions; we depict two different possibilities here. We show the learning trajectory when training incrementally from data generates first by $p_1$, then $p_2$ and $p_3$. On the left, the online updates interfere, jumping between distant points on the manifolds. On the right, the online updates either generalize appropriately—for parallel manifolds—or avoid interference because manifolds are orthogonal.

# Table of Contents

# Rapid Learning or Feature Reuse? Towards understanding the effectiveness of MAML [11]

**Overview of MAML – learning meta-initialization parameters $\theta$**

- draw a batch $\{\mathcal{T}_1, \ldots, \mathcal{T}_B\}$ of $B$ tasks from task distribution $\mathcal{D}$

# Rapid Learning or Feature Reuse? Towards understanding the effectiveness of MAML [11]

**Overview of MAML – learning meta-initialization parameters $\theta$**

- draw a batch $\{\mathcal{T}_1, \ldots, \mathcal{T}_B\}$ of $B$ tasks from task distribution $\mathcal{D}$
- for each task $\mathcal{T}_b$,
  - *support set* of $\mathcal{S}_{\mathcal{T}_b}$ – inner loop update
  - *target set* of examples $\mathcal{Z}_{\mathcal{T}_b}$ – outer loop updates

# Rapid Learning or Feature Reuse? Towards understanding the effectiveness of MAML [11]

**Overview of MAML – learning meta-initialization parameters $\theta$**

- draw a batch $\{\mathcal{T}_1, \ldots, \mathcal{T}_B\}$ of $B$ tasks from task distribution $\mathcal{D}$
- for each task $\mathcal{T}_b$,
  - *support set* of $\mathcal{S}_{\mathcal{T}_b}$ – inner loop update
  - *target set* of examples $\mathcal{Z}_{\mathcal{T}_b}$ – outer loop updates
- optimization procedure:
  - let $\boldsymbol{\theta}_i^b$ be an indication of $\boldsymbol{\theta}$ after $i$ gradient updates for task $\mathcal{T}_b$
  - m-step inner loop optimization:

$$\boldsymbol{\theta}_m^b = \boldsymbol{\theta}_{m-1}^b - \eta \nabla_{\boldsymbol{\theta}_{m-1}^b} \mathcal{L}_{\mathcal{S}_{\mathcal{T}_b}}(\boldsymbol{\theta}_{m-1}^b)$$

  - meta loss:

$$\mathcal{L}_{meta}(\boldsymbol{\theta}) = \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Z}_{\mathcal{T}_b}}(\boldsymbol{\theta}_m^b)$$

  - the outer loop optimization:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta' \nabla_{\boldsymbol{\theta}} \mathcal{L}_{meta}(\boldsymbol{\theta})$$

# Rapid Learning or Feature Reuse? Towards understanding the effectiveness of MAML [11]

**Overview of MAML – learning meta-initialization parameters $\theta$**

- draw a batch $\{\mathcal{T}_1, \ldots, \mathcal{T}_B\}$ of $B$ tasks from task distribution $\mathcal{D}$
- for each task $\mathcal{T}_b$,
  - *support set* of $\mathcal{S}_{\mathcal{T}_b}$ – inner loop update
  - *target set* of examples $\mathcal{Z}_{\mathcal{T}_b}$ – outer loop updates
- optimization procedure:
  - let $\boldsymbol{\theta}_i^b$ be an indication of $\boldsymbol{\theta}$ after $i$ gradient updates for task $\mathcal{T}_b$
  - m-step inner loop optimization:

$$\boldsymbol{\theta}_m^b = \boldsymbol{\theta}_{m-1}^b - \eta \nabla_{\boldsymbol{\theta}_{m-1}^b} \mathcal{L}_{\mathcal{S}_{\mathcal{T}_b}}(\boldsymbol{\theta}_{m-1}^b)$$

  - meta loss:

$$\mathcal{L}_{meta}(\boldsymbol{\theta}) = \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Z}_{\mathcal{T}_b}}(\boldsymbol{\theta}_m^b)$$

  - the outer loop optimization:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta' \nabla_{\boldsymbol{\theta}} \mathcal{L}_{meta}(\boldsymbol{\theta})$$

- test time deployment:
  - draw unseen tasks $\{\mathcal{T}_1^{test}, \ldots, \mathcal{T}_n^{test}\}$ from $\mathcal{D}$

# Rapid Learning or Feature Reuse?

**Question**: Is MAML's efficacy predominantly due to (1) Rapid Learning or (2) Feature Reuse?

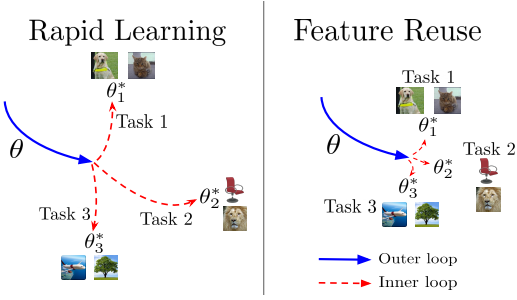



Figure: **Rapid learning and feature reuse paradigms**: the significance of the movement in the parameter space.

# Investigation on the “rapid learning” and “feature reuse”

**Assumption**: There is an important distinction between

1. the *head* (final layer) of the network
2. the earlier layers (network *body*)

# Investigation on the “rapid learning” and “feature reuse”

**Assumption**: There is an important distinction between

1. the *head* (final layer) of the network
   - in each few-shot learning task, there is a different alignment between the output neurons and classes.
2. the earlier layers (network *body*)

# Investigation on the “rapid learning” and “feature reuse”

**Assumption**: There is an important distinction between

1. the *head* (final layer) of the network
   - in each few-shot learning task, there is a different alignment between the output neurons and classes.
   - the head must necessarily change for each task to learn the new alignment.
2. the earlier layers (network *body*)

# Investigation on the “rapid learning” and “feature reuse”

**Assumption**: There is an important distinction between

1. the *head* (final layer) of the network
   - in each few-shot learning task, there is a different alignment between the output neurons and classes.
   - the head must necessarily change for each task to learn the new alignment.
2. the earlier layers (network *body*)

**Experimental design**: two sets of **numerical experiments**

# Investigation on the “rapid learning” and “feature reuse”

**Assumption**: There is an important distinction between

1. the *head* (final layer) of the network
   - in each few-shot learning task, there is a different alignment between the output neurons and classes.
   - the head must necessarily change for each task to learn the new alignment.
2. the earlier layers (network *body*)

**Experimental design**: two sets of **numerical experiments**

1. evaluate the performance when freezing parameters after MAML training, without test time inner loop adaptation.

# Investigation on the “rapid learning” and “feature reuse”

**Assumption**: There is an important distinction between

1. the *head* (final layer) of the network
   - in each few-shot learning task, there is a different alignment between the output neurons and classes.
   - the head must necessarily change for each task to learn the new alignment.
2. the earlier layers (network *body*)

**Experimental design**: two sets of **numerical experiments**

1. evaluate the performance when freezing parameters after MAML training, without test time inner loop adaptation.
2. use representational similarity tools to measure how much the network features and representations change through the inner loop.

# Freezing layer representations (test-time adaptation) experiments

Even when freezing all layers in the *network body*, performance hardly changes.

Table: Test accuracy. Freezing successive layers (preventing inner loop adaptation) does not affect accuracy, supporting feature reuse.

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

# Freezing layer representations (test-time adaptation) experiments

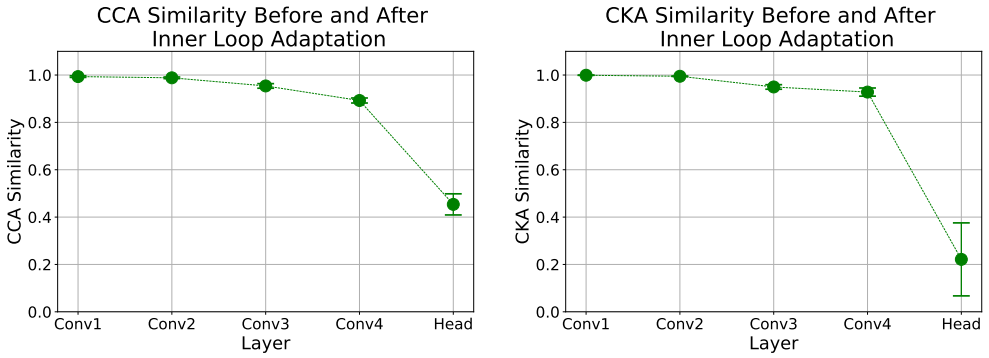
Even when freezing all layers in the *network body*, performance hardly changes.

The meta-initialization has already learned good enough features
that can be reused as is for each test time task.

Table: Test accuracy. Freezing successive layers (preventing inner loop adaptation) does not affect accuracy, supporting feature reuse.

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

## Representational similarity experiments




(a) (b)

Figure: **High CCA/CKA similarity between representations before and after adaptation for all layers except the head.** This suggests: these layers do not change much during adaptation, but mostly perform feature reuse.

## Feature Reuse happens early in learning

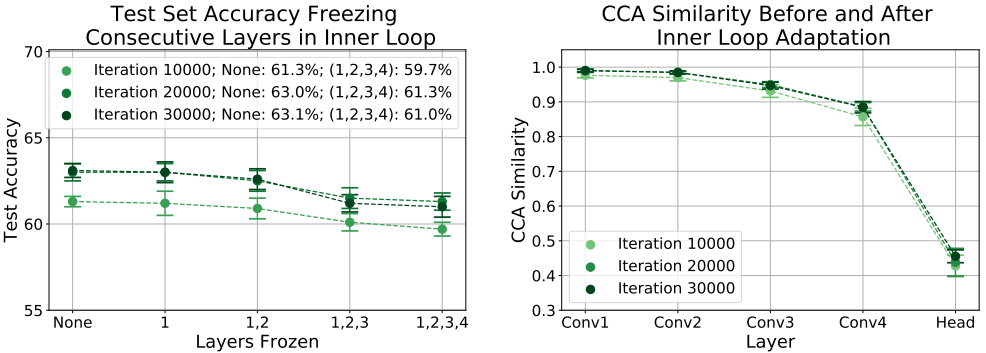



Figure: **Inner loop updates have little effect on learned representations from early on in learning.**

## From insights to efficient method design: ANIL (Almost No Inner Loop) algorithm

**Intuition**: The inner loop updates have little effect on the network body even early in training.

## From insights to efficient method design: ANIL (Almost No Inner Loop) algorithm

**Intuition**: The inner loop updates have little effect on the network body even early in training.

**ANIL (Almost No Inner Loop) algorithm:**
During training and test,

- the inner loop updates are removed for the network *body*.
- the inner loop adaptation is only applied to the *head*.

## From insights to efficient method design: ANIL (Almost No Inner Loop) algorithm

**Intuition**: The inner loop updates have little effect on the network body even early in training.

**ANIL (Almost No Inner Loop) algorithm:**
During training and test,

- the inner loop updates are removed for the network *body*.
- the inner loop adaptation is only applied to the *head*.

Table: **ANIL matches the performance of MAML on few-shot image classification and RL.**

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

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

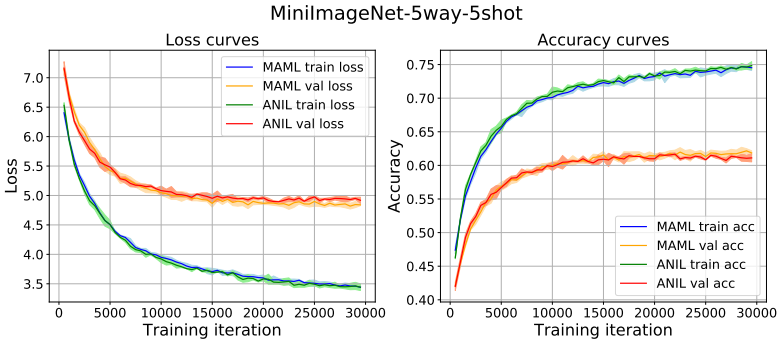



Figure: **MAML and ANIL learn very similarly**.

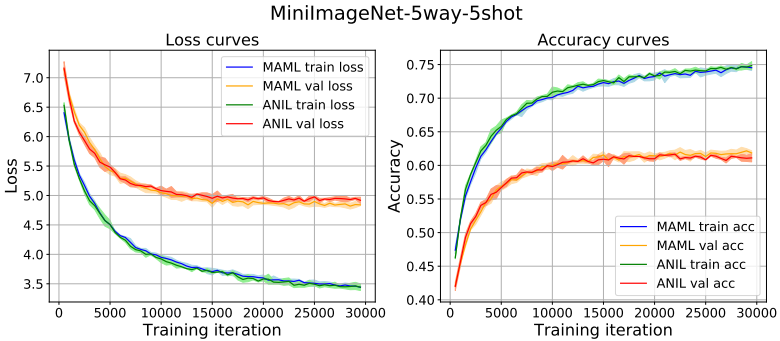



Figure: **MAML and ANIL learn very similarly**.

Table: **MAML and ANIL models learn comparable representations.** (across different random seeds): algorithmic differences between MAML/ANIL does not result in vastly different types of features learned.

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

## From insights to efficient method design: NIL (No Inner Loop) algorithm

**Question**: How important is the head at test time, when good features have already been learned?

[1] Each task first passes the $k$ support set samples through the network body to get their penultimate layer representations. For a test example, compute cosine similarities with support set samples and use them to weight the support set labels.

## From insights to efficient method design: NIL (No Inner Loop) algorithm

**Question**: How important is the head at test time, when good features have already been learned?

The representations can be used directly, with *no* adaptation:

[1] Each task first passes the $k$ support set samples through the network body to get their penultimate layer representations. For a test example, compute cosine similarities with support set samples and use them to weight the support set labels.

## From insights to efficient method design: NIL (No Inner Loop) algorithm

**Question**: How important is the head at test time, when good features have already been learned?

The representations can be used directly, with *no* adaptation:

- NIL trains a few-shot learning model with ANIL/MAML.

[1] Each task first passes the $k$ support set samples through the network body to get their penultimate layer representations. For a test example, compute cosine similarities with support set samples and use them to weight the support set labels.

## From insights to efficient method design: NIL (No Inner Loop) algorithm

**Question**: How important is the head at test time, when good features have already been learned?

The representations can be used directly, with *no* adaptation:

- NIL trains a few-shot learning model with ANIL/MAML.
- At test time, NIL removes the head entirely, and uses learned features and cosine similarity to perform effective classification[1].

[1] Each task first passes the $k$ support set samples through the network body to get their penultimate layer representations. For a test example, compute cosine similarities with support set samples and use them to weight the support set labels.

## From insights to efficient method design: NIL (No Inner Loop) algorithm

**Question**: How important is the head at test time, when good features have already been learned?

The representations can be used directly, with *no* adaptation:

- NIL trains a few-shot learning model with ANIL/MAML.
- At test time, NIL removes the head entirely, and uses learned features and cosine similarity to perform effective classification[1].

Table: **NIL algorithm performs as well as MAML and ANIL on few-shot image classification**: the strength of the learned features, and the relative lack of importance of the head at test time.

| Method | Omniglot-20way-1shot | Omniglot-20way-5shot | MiniImageNet-5way-1shot | MiniImageNet-5way-5shot |
|---|---|---|---|---|
| MAML | $93.7 \pm 0.7$ | $96.4 \pm 0.1$ | $46.9 \pm 0.2$ | $63.1 \pm 0.4$ |
| ANIL | $96.2 \pm 0.5$ | $98.0 \pm 0.3$ | $46.7 \pm 0.4$ | $61.5 \pm 0.5$ |
| NIL | $96.7 \pm 0.3$ | $98.0 \pm 0.0$ | $48.0 \pm 0.7$ | $62.2 \pm 0.5$ |

[1] Each task first passes the $k$ support set samples through the network body to get their penultimate layer representations. For a test example, compute cosine similarities with support set samples and use them to weight the support set labels.

**Question**: How important task alignment and the head are during training to ensure good features.

We answer this question by examining the quality of features arising from different training regimes for the body.

**Question**: How important task alignment and the head are during training to ensure good features.

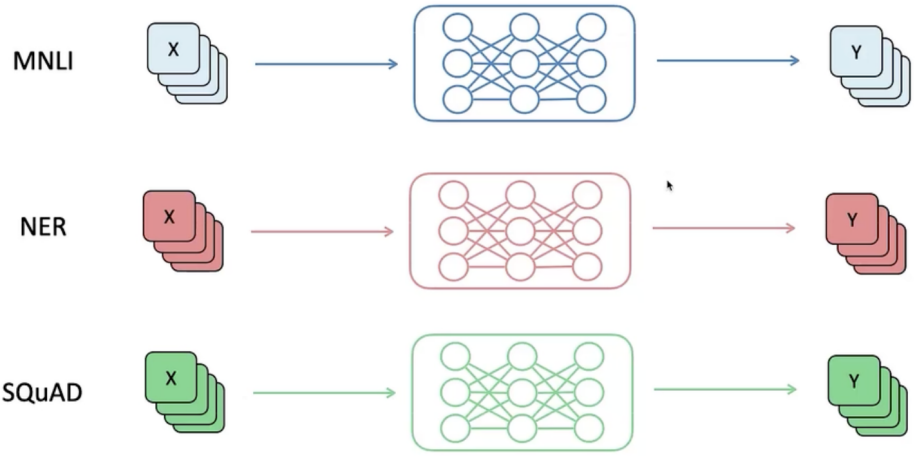
We answer this question by examining the quality of features arising from different training regimes for the body.

Table: **MAML/ANIL training leads to superior features learned** over other methods which do not have task-specific heads, supporting the importance of the head at training.

| Method | MiniImageNet-5way-1shot | MiniImageNet-5way-5shot |
|---|---|---|
| MAML training-NIL head | $48.4 \pm 0.3$ | $61.5 \pm 0.8$ |
| ANIL training-NIL head | $48.0 \pm 0.7$ | $62.2 \pm 0.5$ |
| Multiclass training-NIL head | $39.7 \pm 0.3$ | $54.4 \pm 0.5$ |
| Multitask training-NIL head | $26.5 \pm 1.1$ | $34.2 \pm 3.5$ |
| Random features-NIL head | $32.9 \pm 0.6$ | $43.2 \pm 0.5$ |
| NIL training-NIL head | $38.3 \pm 0.6$ | $43.0 \pm 0.2$ |

# Table of Contents

# Full Fine-Tuning




Each task needs a separate model copy

$\rightarrow$ expensive as the number of tasks and model size grow

# Parameter-Efficient Transfer Learning

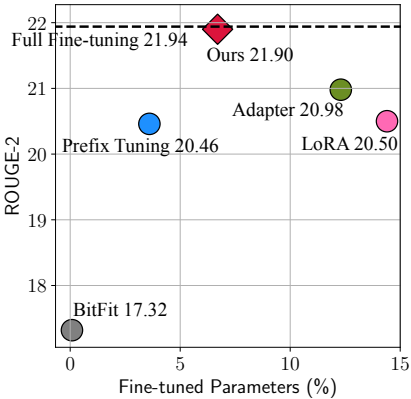
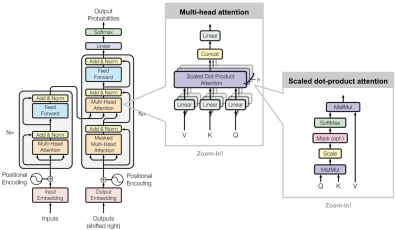


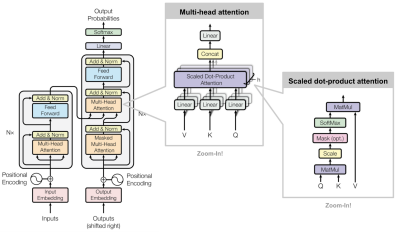

Figure: Performance of different methods on a summarization task. The number of fine-tuned parameters is relative to the tuned parameters in full fine-tuning.

Parameter-efficient tuning matters!

# Recap Transformer



# Recap Transformer

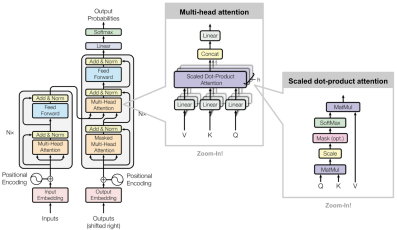



The conventional attention function maps queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\mathbf{K} \in \mathbb{R}^{m \times d_k}, \mathbf{V} \in \mathbb{R}^{m \times d_v}$:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\,, \tag{3}$$

and $n$ and $m$ are the # of queries and key-value pairs respectively.

# Recap Transformer



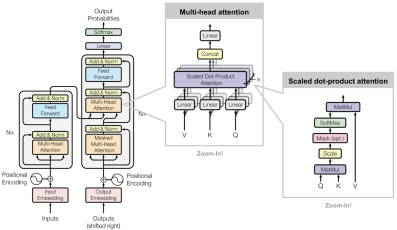

The conventional attention function maps queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, $\mathbf{V} \in \mathbb{R}^{m \times d_v}$:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\,, \tag{3}$$

and $n$ and $m$ are the # of queries and key-value pairs respectively.

**Multi-Head Attention** (MHA) performs the attention function in parallel over $N_h$ heads:

# Recap Transformer




The conventional attention function maps queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, $\mathbf{V} \in \mathbb{R}^{m \times d_v}$:

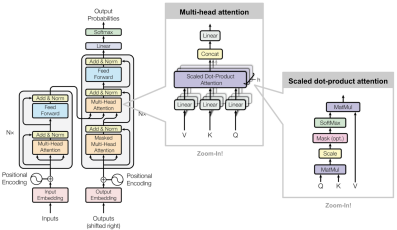
$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\,, \tag{3}$$

and $n$ and $m$ are the # of queries and key-value pairs respectively.

**Multi-Head Attention** (MHA) performs the attention function in parallel over $N_h$ heads:

- Each head is parameterized by $\mathbf{W}_q^{(i)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$. $d$ is the model dimension.

# Recap Transformer




The conventional attention function maps queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, $\mathbf{V} \in \mathbb{R}^{m \times d_v}$:

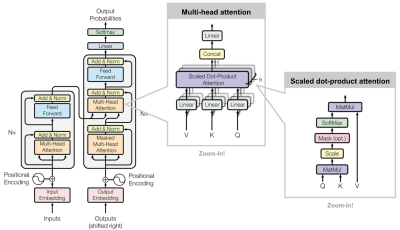
$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\,, \tag{3}$$

and $n$ and $m$ are the # of queries and key-value pairs respectively.

**Multi-Head Attention** (MHA) performs the attention function in parallel over $N_h$ heads:

- Each head is parameterized by $\mathbf{W}_q^{(i)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$. $d$ is the model dimension.
- Given a sequence of $m$ vectors $\mathbf{C} \in \mathbb{R}^{m \times d}$ and a query vector $\mathbf{x} \in \mathbb{R}^d$,

# Recap Transformer




The conventional attention function maps queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, $\mathbf{V} \in \mathbb{R}^{m \times d_v}$:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\,, \tag{3}$$

and $n$ and $m$ are the # of queries and key-value pairs respectively.

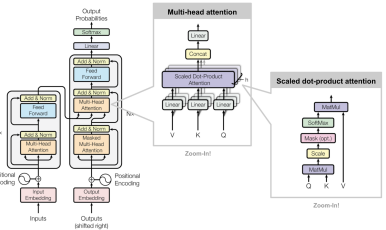
**Multi-Head Attention** (MHA) performs the attention function in parallel over $N_h$ heads:

- Each head is parameterized by $\mathbf{W}_q^{(i)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$. $d$ is the model dimension.
- Given a sequence of $m$ vectors $\mathbf{C} \in \mathbb{R}^{m \times d}$ and a query vector $\mathbf{x} \in \mathbb{R}^d$, multi-head attention (MHA) computes the output on each head and concatenates them:

$$\text{MHA}(\mathbf{C}, \mathbf{x}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}_o\,, \qquad \text{head}_i = \text{Attn}(\mathbf{x}\mathbf{Q}_q^{(i)}, \mathbf{C}\mathbf{K}_k^{(i)}, \mathbf{C}\mathbf{V}_v^{(i)})\,, \tag{4}$$

where $d$ is the model dimension, and in MHA, $d_h := \frac{d}{N_h}$.

# Recap Transformer




The conventional attention function maps queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\mathbf{K} \in \mathbb{R}^{m \times d_k}, \mathbf{V} \in \mathbb{R}^{m \times d_v}$:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\,, \tag{3}$$

and $n$ and $m$ are the # of queries and key-value pairs respectively.

**Multi-Head Attention** (MHA) performs the attention function in parallel over $N_h$ heads:

- Each head is parameterized by $\mathbf{W}_q^{(i)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$. $d$ is the model dimension.
- Given a sequence of $m$ vectors $\mathbf{C} \in \mathbb{R}^{m \times d}$ and a query vector $\mathbf{x} \in \mathbb{R}^d$, multi-head attention (MHA) computes the output on each head and concatenates them:

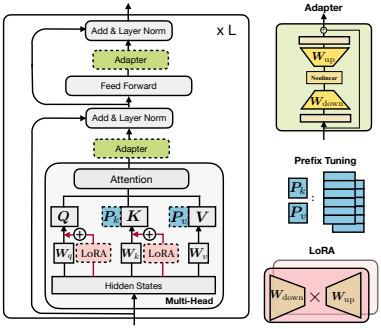
$$\text{MHA}(\mathbf{C}, \mathbf{x}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}_o\,, \qquad \text{head}_i = \text{Attn}(\mathbf{x}\mathbf{Q}_q^{(i)}, \mathbf{C}\mathbf{K}_k^{(i)}, \mathbf{C}\mathbf{V}_v^{(i)})\,, \tag{4}$$

where $d$ is the model dimension, and in MHA, $d_h := \frac{d}{N_h}$.

**Fully connected Feed-Forward Network** (FFN):

$$\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2\,, \qquad \mathbf{W}_1 \in \mathbb{R}^{d \times d_m}, \mathbf{W}_2 \in \mathbb{R}^{d_m \times d}, \text{and e.g., } d_m = 4d\,. \tag{5}$$

# Existing methods




- Adapters [4]

$$\mathbf{h} \leftarrow h + f\left(\mathbf{h}\mathbf{W}_{\text{down}}\right)\mathbf{W}_{\text{up}}\,, \qquad \mathbf{W}_{\text{down}} \in \mathbb{R}^{d\times r}, \mathbf{W}_{\text{down}} \in \mathbb{R}^{r\times d} \quad (6)$$

- Prefix tuning [9] (inspired by the success of textual prompting)

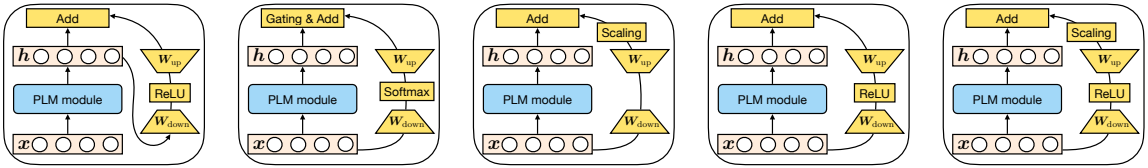
$$\text{head}_i = \text{Attn}\left(\mathbf{x}\mathbf{W}_q^{(i)}, \text{concat}\left(\mathbf{P}_k^{(i)}, \mathbf{C}\mathbf{W}_k^{(i)}\right), \text{concat}\left(\mathbf{P}_v^{(i)}, \mathbf{C}\mathbf{W}_v^{(i)}\right)\right),$$

where two sets of prefix vectors $\mathbf{P}_k, \mathbf{P}_v \in \mathbb{R}^{l\times d}$ are concatenated with the original key $\mathbf{K}$ and value $\mathbf{V}$.

- LoRA [5] (inject trainable low-rank matrices into layers)

$$\mathbf{h} \leftarrow \mathbf{h} + s \cdot \mathbf{x}\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}\,, \quad \mathbf{W}_{\text{down}} \in \mathbb{R}^{d\times r}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times k}, r \ll \{d, k\}$$

[4] *Houlsby et al.* Parameter-efficient transfer learning for NLP. 2019.
[9] *Li et al.* Prefix-tuning: Optimizing continuous prompts for generation. 2021.
[5] *Hu et al.* Lora: Low-rank adaptation of large language models. 2021.




(a) Adapter. (b) Prefix Tuning. (c) LoRA. (d) Parallel Adapter. (e) Scaled PA.

Figure: **Graphical illustration of existing methods and some other variants.** "PLM module" represents a certain sublayer of the PLM (e.g. attention or FFN) that is frozen.

[3] *He et al.* Towards a unified view of parameter-efficient transfer learning. 2022.

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[3] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[4] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[5] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[6] K. Javed and M. White. Meta-learning representations for continual learning. *NeurIPS*, 2019.

[7] K. Javed, H. Yao, and M. White. Is fast adaptation all you need? *NeurIPS workshop*, 2019.

[8] J. Jiang, Y. Shu, J. Wang, and M. Long. Transferability in deep learning: A survey. *arXiv preprint arXiv:2201.05867*, 2022.

[9] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[10] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[11] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019.

[12] A. Rajeswaran, C. Finn, S. Kakade, and S. Levine. Meta-learning with implicit gradients. *NeurIPS*, 2019.