MetaFun: Meta-Learning with Iterative Functional Updates

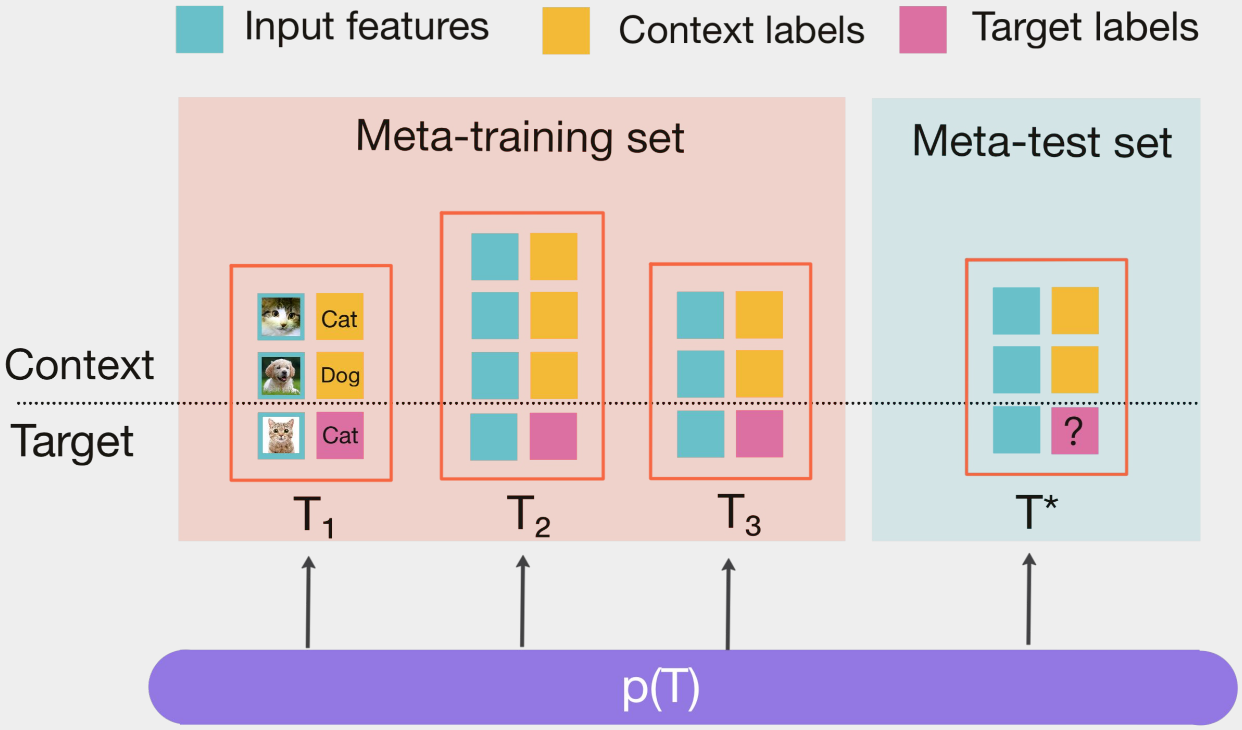
Jin Xu, Jean-Francois Ton, Hyunjik Kim, Adam R. Kosiorek, Yee Whye Teh

UNIVERSITY OF OXFORD

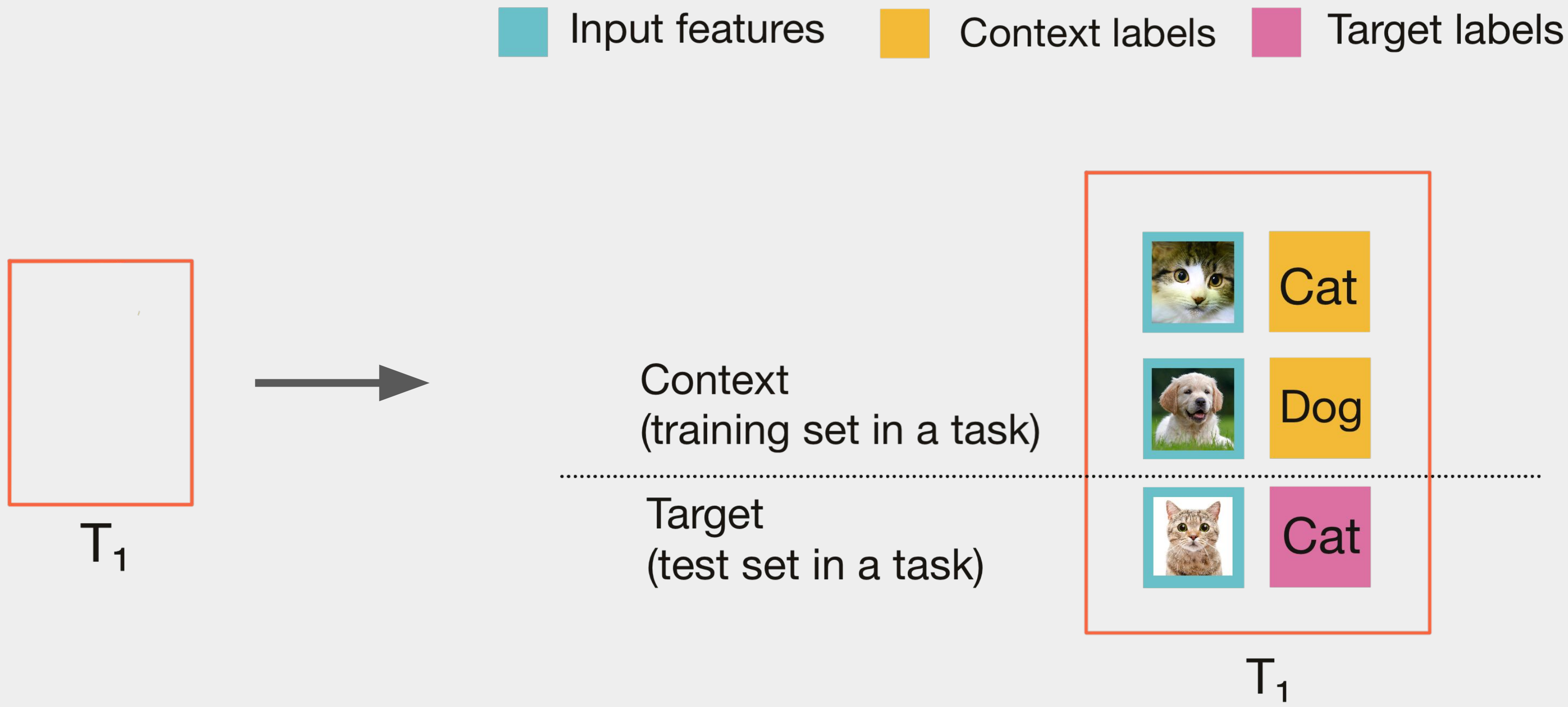
DeepMind

37th International Conference on Machine Learning

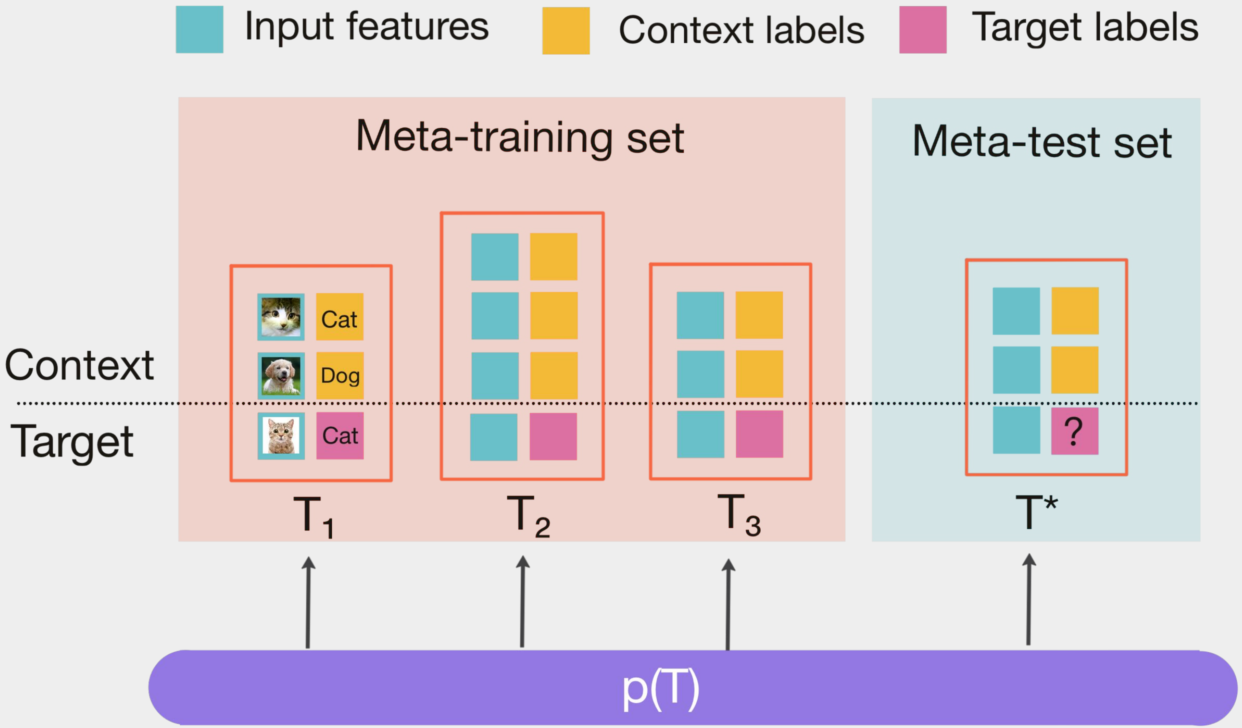
Supervised Meta-Learning



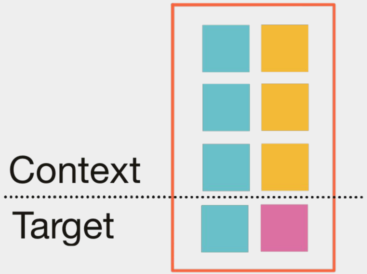
Supervised Meta-Learning



Supervised Meta-Learning

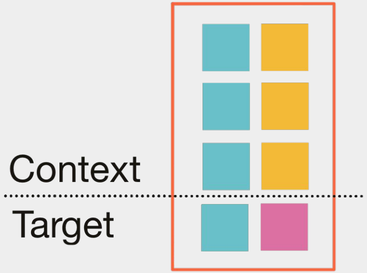


Encoder-Decoder Approaches to Supervised Meta-Learning



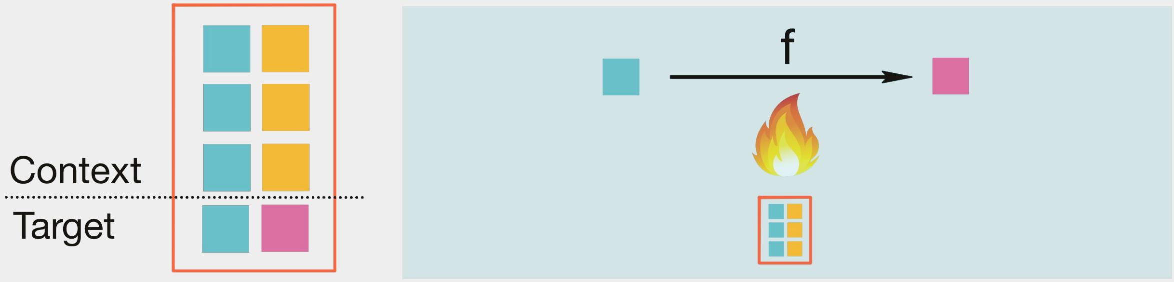
Encoder-Decoder Approaches to Supervised Meta-Learning

What is learning ?

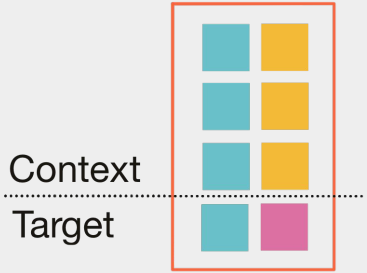


Encoder-Decoder Approaches to Supervised Meta-Learning

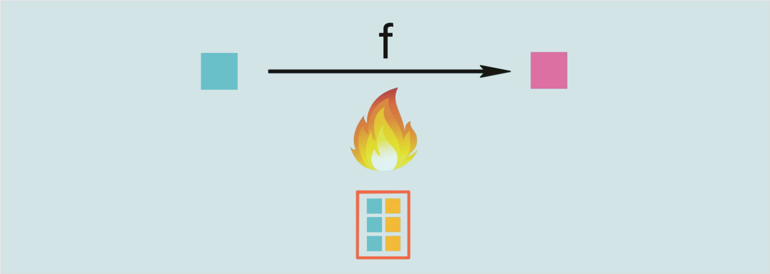
What is learning ?



Encoder-Decoder Approaches to Supervised Meta-Learning



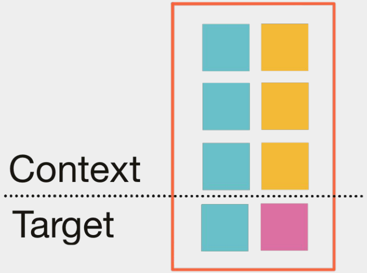
What is learning ?



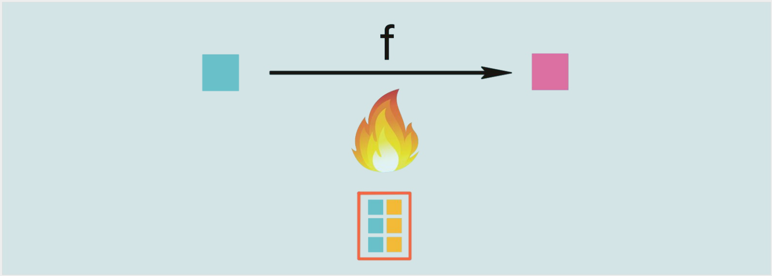
What is meta-learning?

(in encoder-decoder approaches like CNP[1])

Encoder-Decoder Approaches to Supervised Meta-Learning

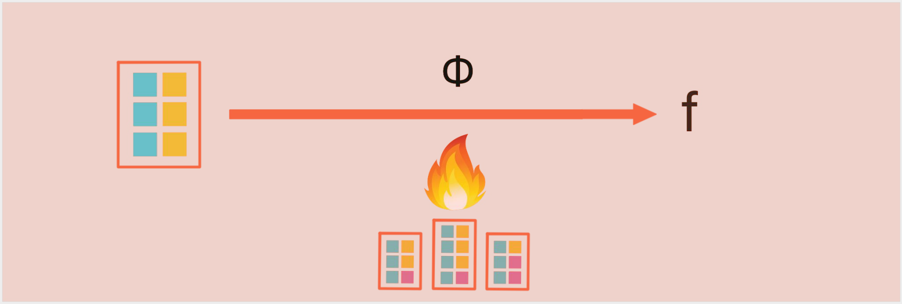


What is learning ?



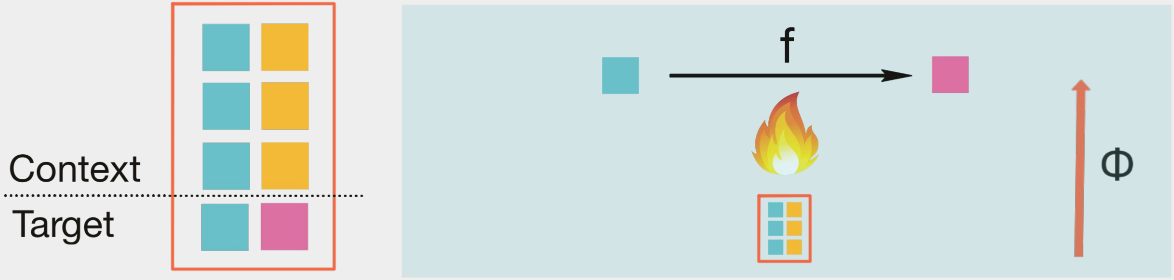
What is meta-learning?

(in encoder-decoder approaches like CNP[1])



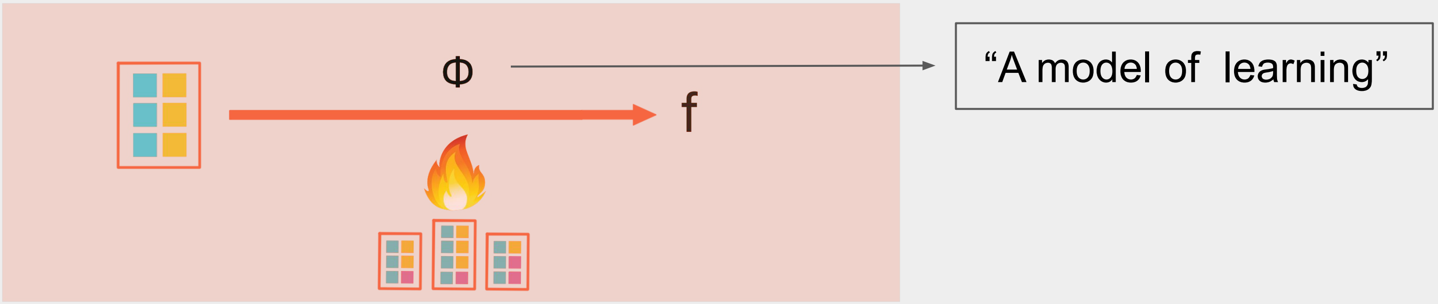
Encoder-Decoder Approaches to Supervised Meta-Learning

What is learning ?

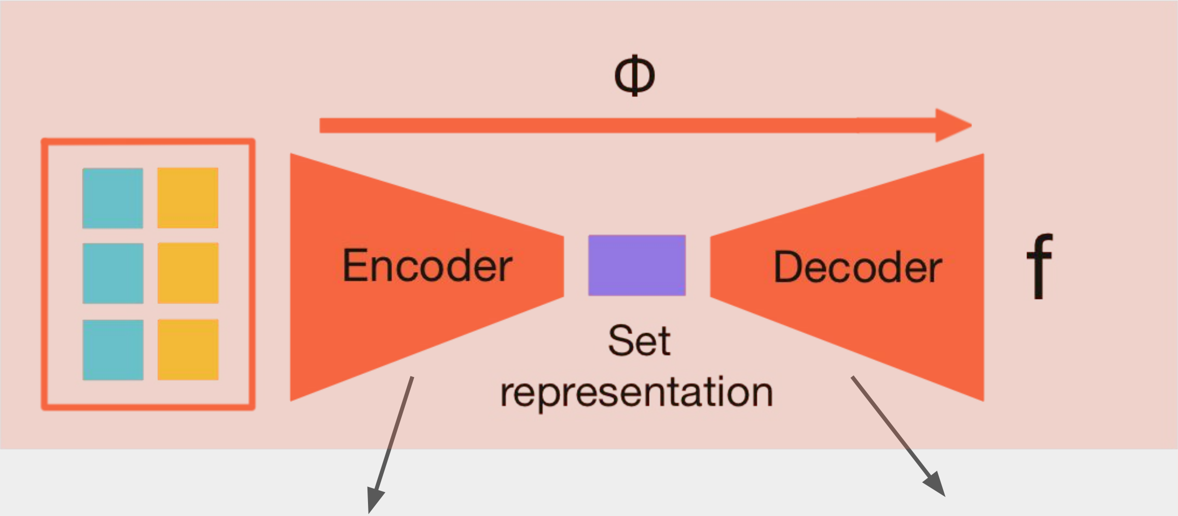


What is meta-learning?

(in encoder-decoder approaches like CNP[1])



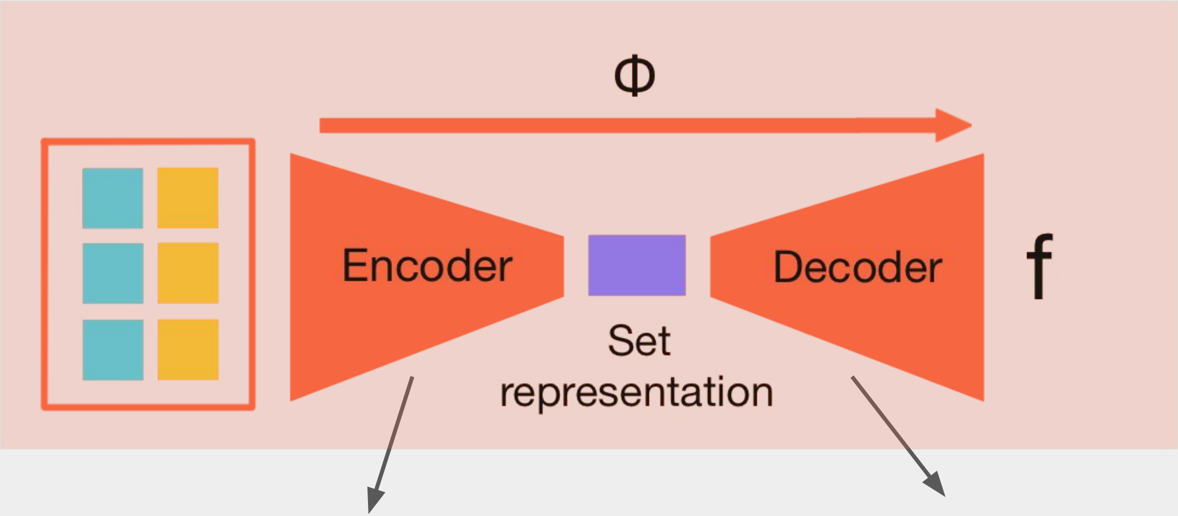
Encoder-Decoder Approaches to Supervised Meta-Learning



Summarises
the context

Predicts conditioned
on the representation.

Encoder-Decoder Approaches to Supervised Meta-Learning



Summarises
the context

Predicts conditioned
on the representation.

Both parameterised by NNs

Incorporating Inductive Biases into Deep Learning Models

Classifier

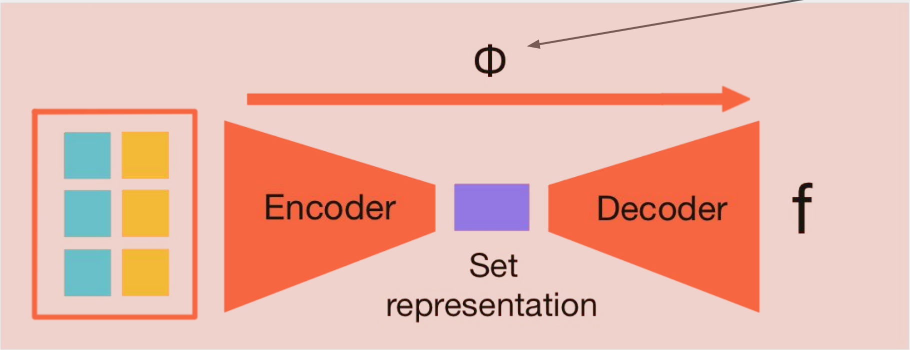
Dog

Convolutional structure as inductive bias.

Incorporating Inductive Biases into Deep Learning Models

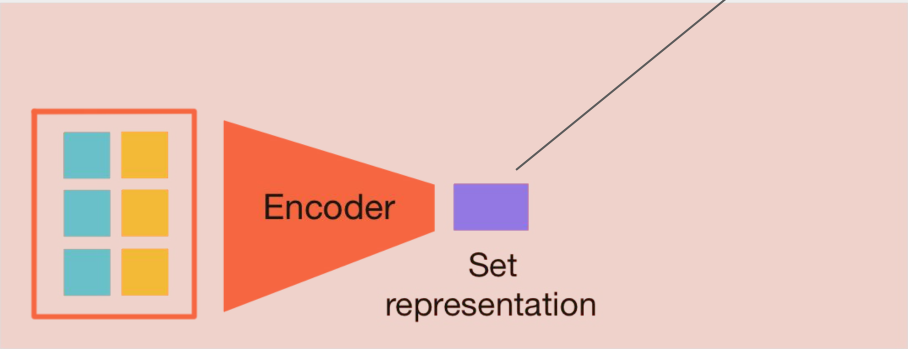
Convolutional structure as inductive bias.

What are good inductive biases for "a model of learning"?



MetaFun Overview

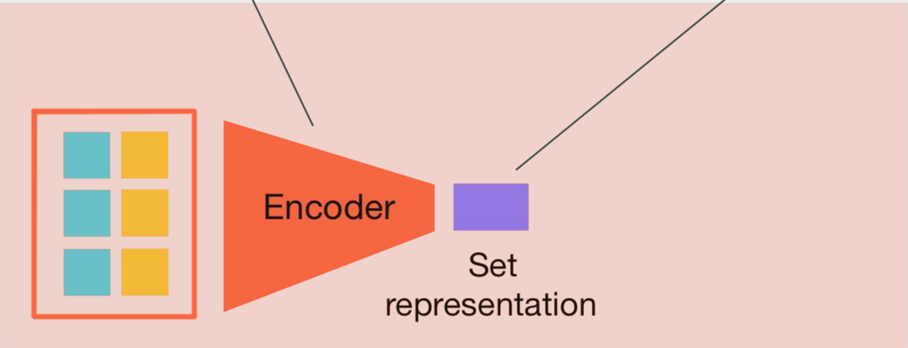
What is a better form of set representation?



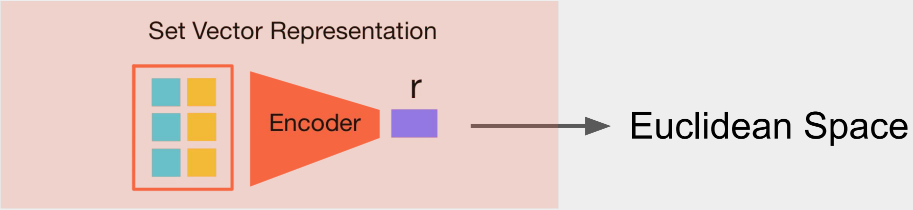
MetaFun Overview

What are good inductive biases/structures for the encoder?

What is a better form of set representation?

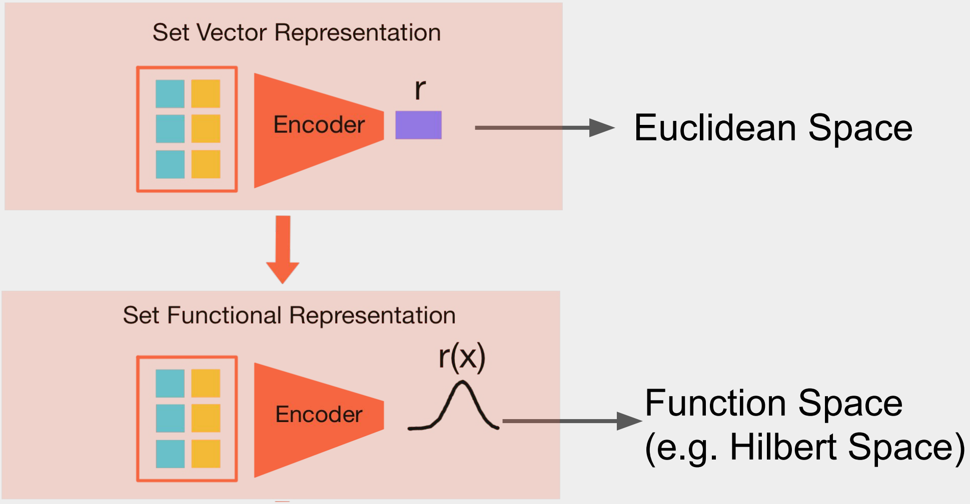


MetaFun Overview



MetaFun Overview

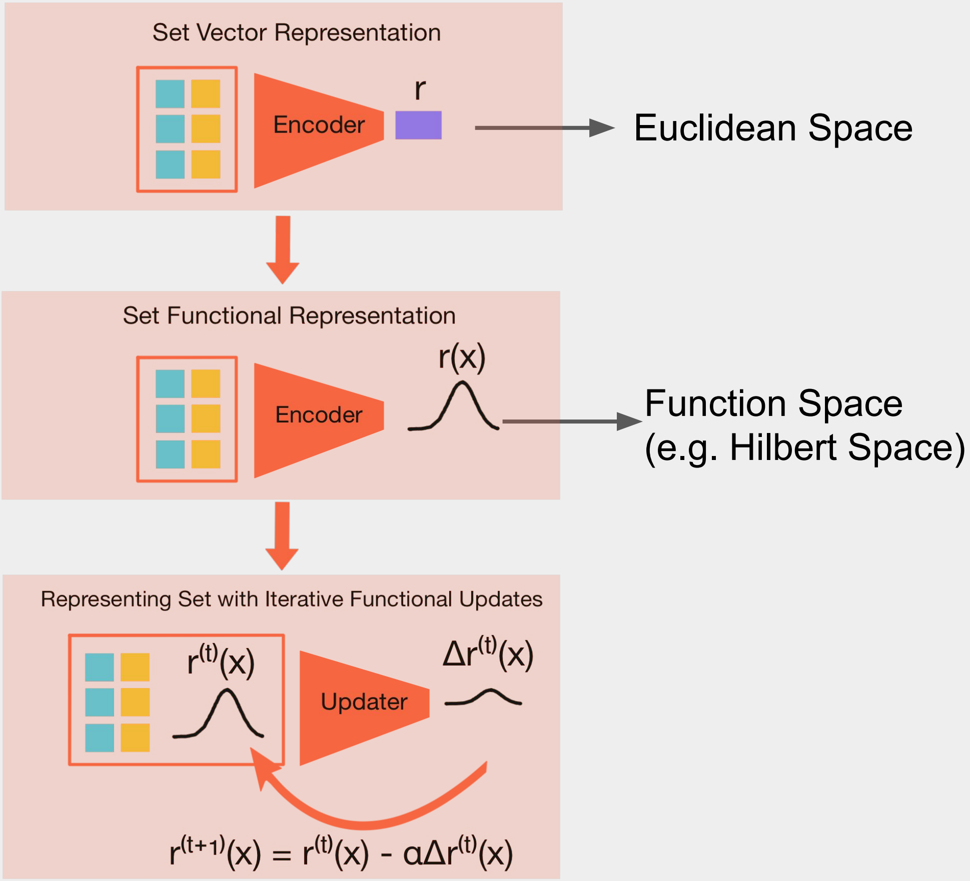
Functional Representation



MetaFun Overview

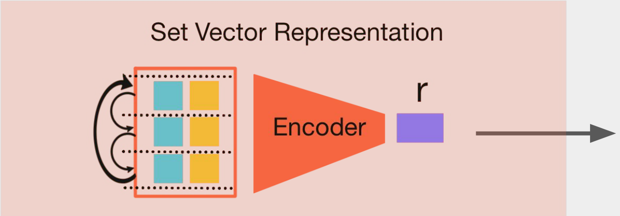
Functional Representation

Encoders with Iterative Structure



MetaFun Overview

Functional Representation



Euclidean Space

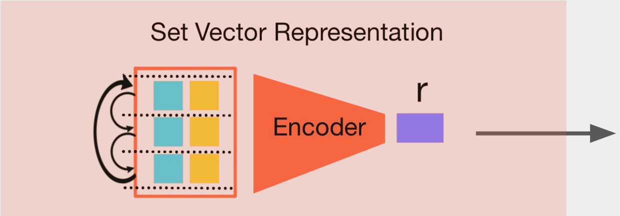
(permutation of data points should not change set representation)

Encoders with Iterative Structure

MetaFun Overview

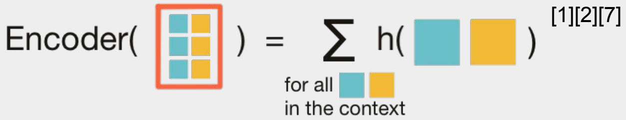
Functional Representation

Encoders with Iterative Structure



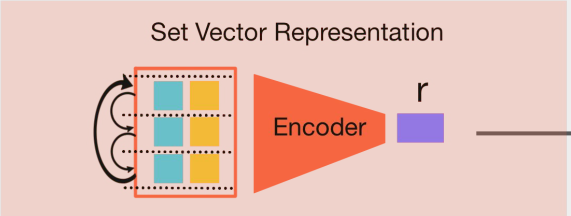
Euclidean Space

(permutation of data points should not change set representation)



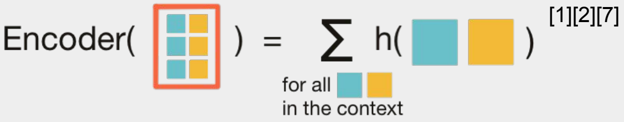
MetaFun Overview

Functional Representation



→ Euclidean Space

(permutation of data points should not change set representation)



Fixed dimensional representation can be limiting for large set size[4], and often lead to underfitting[3].

Encoders with Iterative Structure

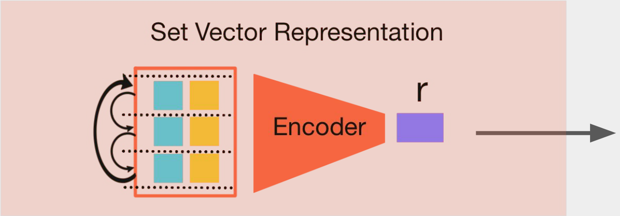
MetaFun Overview

Functional Representation

Permutation invariance

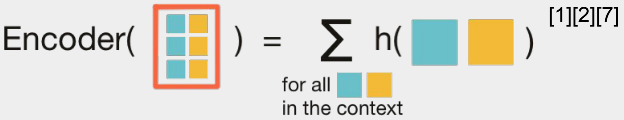
Flexible capacity

Encoders with Iterative Structure



Euclidean Space

(permutation of data points should not change set representation)



Fixed dimensional representation can be limiting for large set size[4], and often lead to underfitting[3].

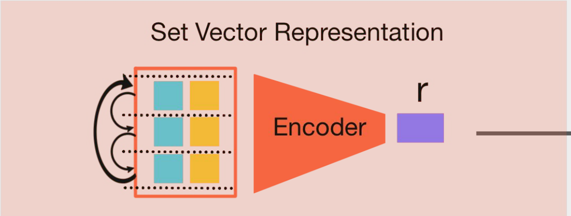
MetaFun Overview

Functional Representation

Permutation invariance

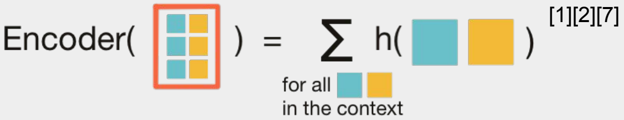
Flexible capacity

Encoders with Iterative Structure



→ Euclidean Space

(permutation of data points should not change set representation)



Fixed dimensional representation can be limiting for large set size[4], and often lead to underfitting[3].

Self-attention modules[6] or relation network[9] can model interaction within the context, but not context-target interaction

MetaFun Overview

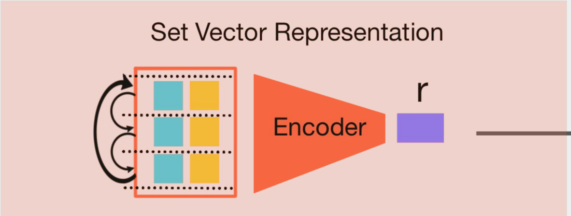
Functional Representation

Permutation invariance

Flexible capacity

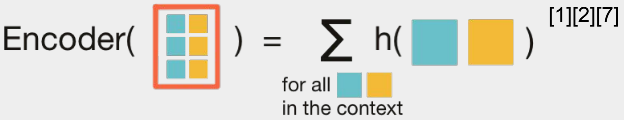
Within-context and context-target interaction

Encoders with Iterative Structure



→ Euclidean Space

(permutation of data points should not change set representation)



Fixed dimensional representation can be limiting for large set size[4], and often lead to underfitting[3].

Self-attention modules[6] or relation network[9] can model interaction within the context, but not context-target interaction

MetaFun Overview

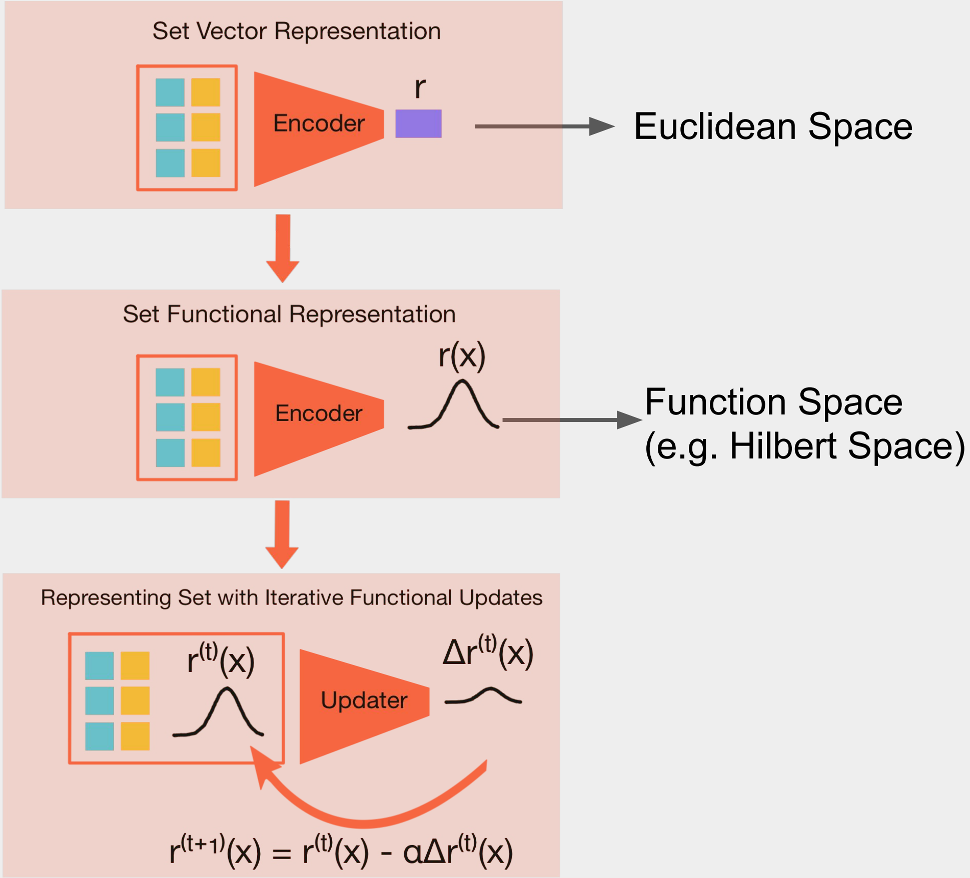
Functional Representation

Permutation invariance

Flexible capacity

Within-context and context-target interaction

Encoders with Iterative Structure



MetaFun Overview

Functional Representation

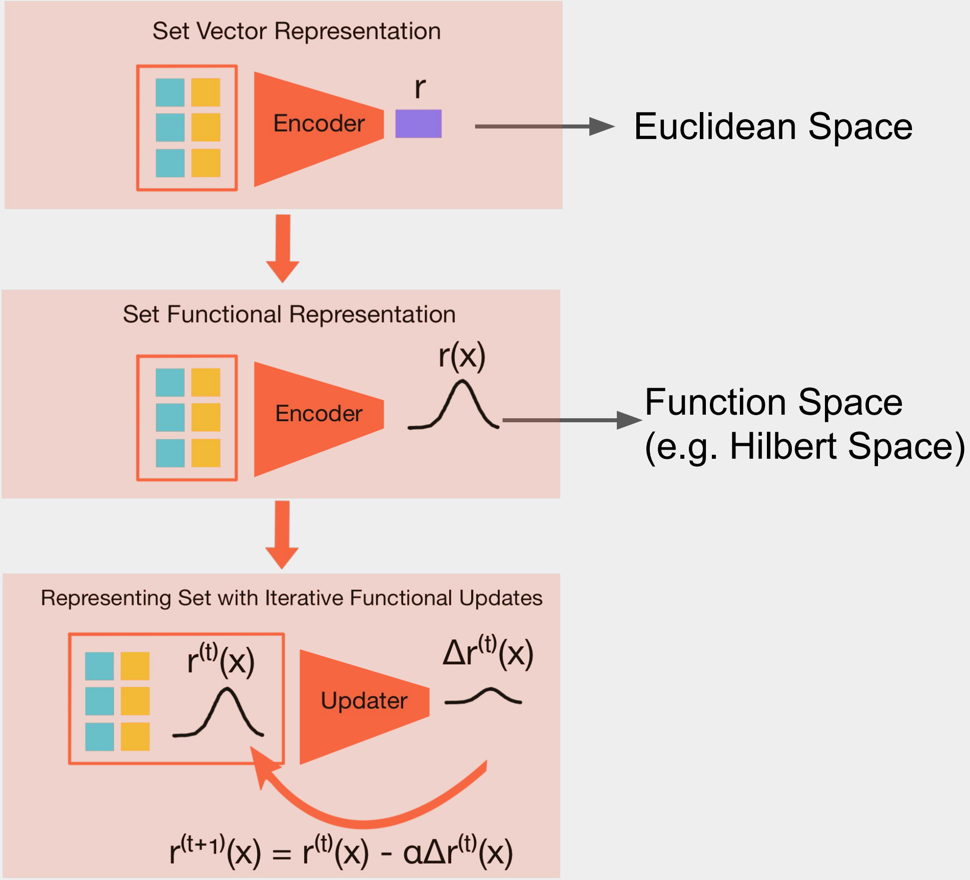
Permutation invariance

Flexible capacity

Within-context and context-target interaction

Encoders with Iterative Structure

Learning to update representation with feedback is easier than learning representation directly



MetaFun Overview

Functional Representation

Permutation invariance

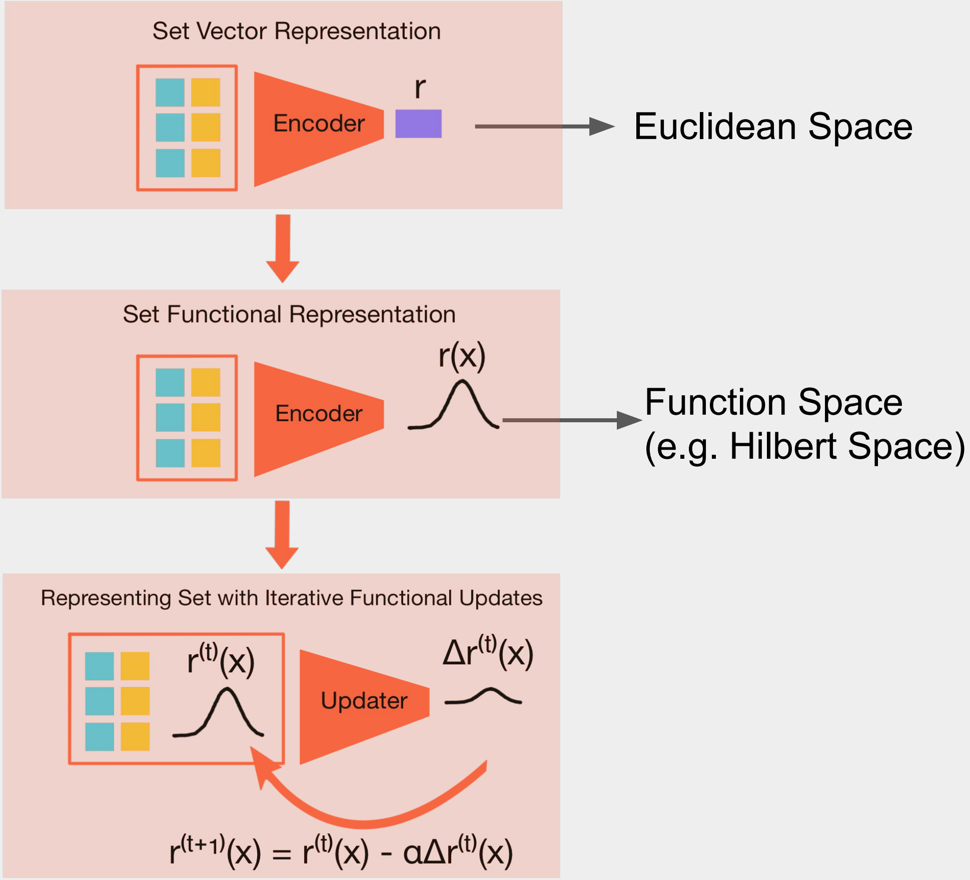
Flexible capacity

Within-context and context-target interaction

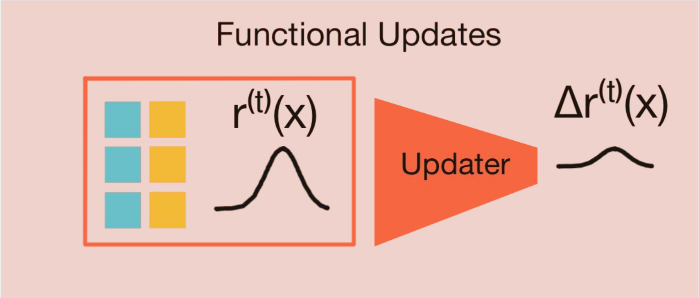
Encoders with Iterative Structure

Learning to update representation with feedback is easier than learning representation directly

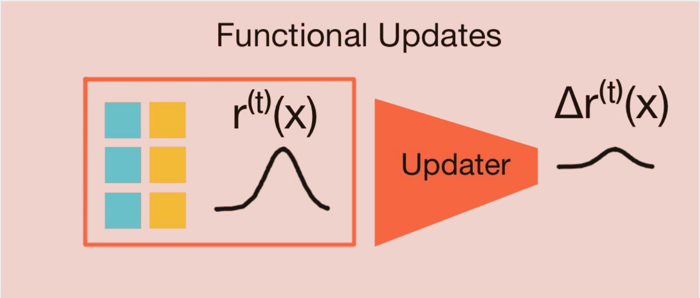
Iterative structure may be a good inductive bias for "the model of learning". (Learning algorithms are often iterative, such as gradient descent)



MetaFun



MetaFun and Functional Gradient Descent



Gradient Descent

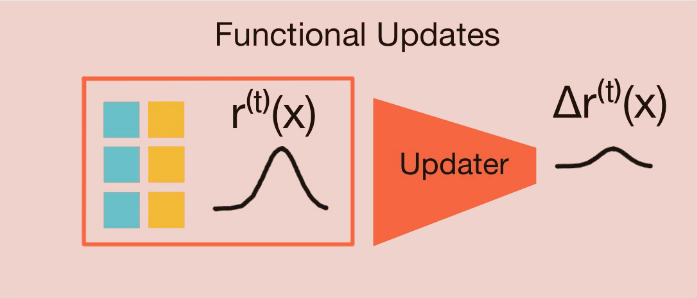
solve

$$\arg\min_{\theta} L(\theta)$$

by iterative optimisation

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$$

MetaFun and Functional Gradient Descent



Gradient Descent

solve

$$\arg\min_{\theta} L(\theta)$$

by iterative optimisation

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} L(\theta_t)$$

Functional Gradient Descent

solve

$$\arg\min_{f} L(f)$$

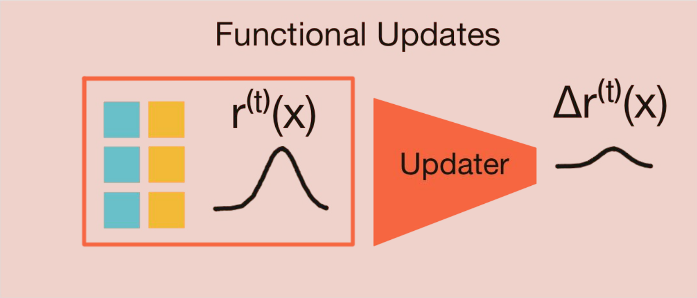
by iterative optimisation

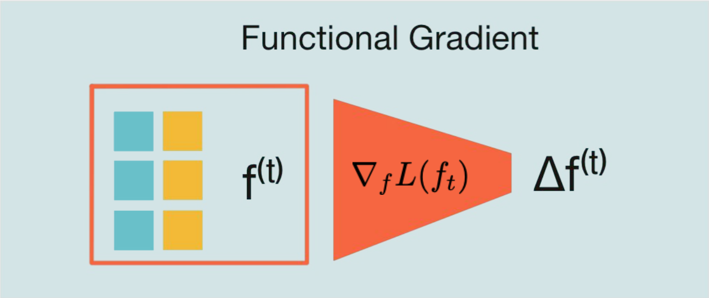
$$f_{t+1} = f_t - \alpha \nabla_f L(f_t)$$

For supervised learning problems, the objective function often has this form:

$$L(f_t) = \frac{1}{|C|} \sum_{i \in C} l(f_t(\mathbf{x}_i), \mathbf{y}_i)$$

MetaFun and Functional Gradient Descent





Gradient Descent

solve

$$\arg\min_{\theta} L(\theta)$$

by iterative optimisation

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$$

Functional Gradient Descent

solve

$$\arg\min_{f} L(f)$$

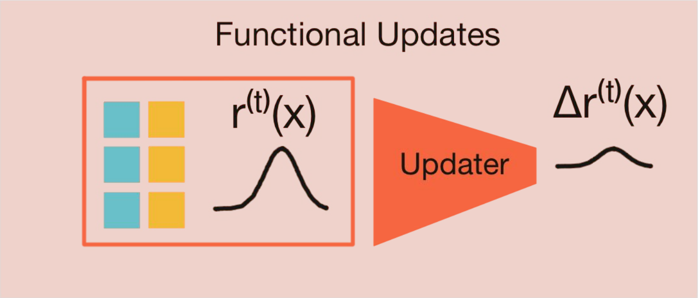
by iterative optimisation

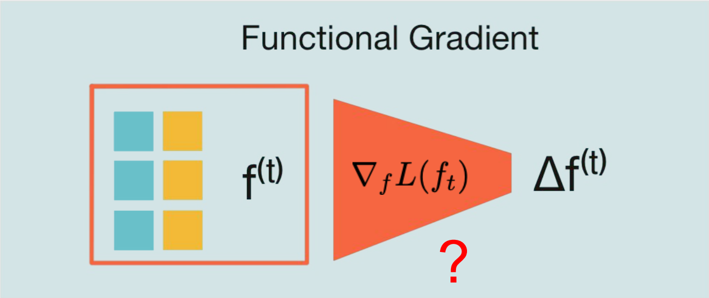
$$f_{t+1} = f_t - \alpha \nabla_f L(f_t)$$

For supervised learning problems, the objective function often has this form:

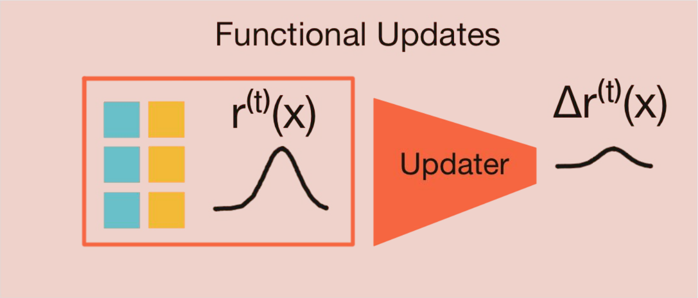
$$L(f_t) = \frac{1}{|C|} \sum_{i \in C} l(f_t(\mathbf{x}_i), \mathbf{y}_i)$$

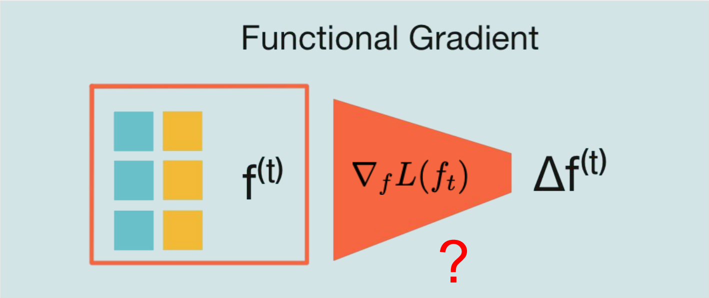
MetaFun and Functional Gradient Descent

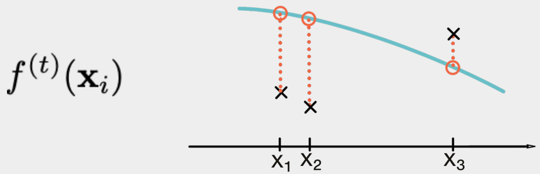




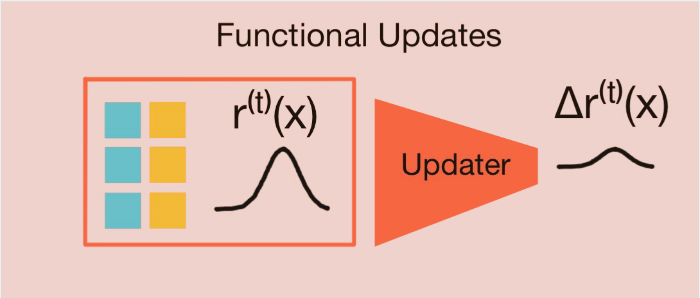
MetaFun and Functional Gradient Descent

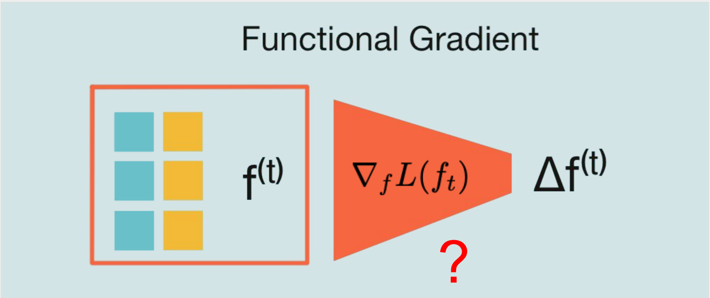


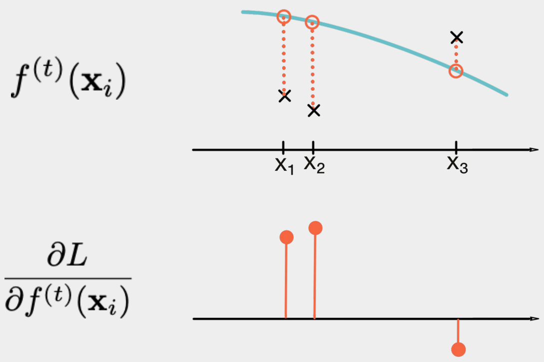




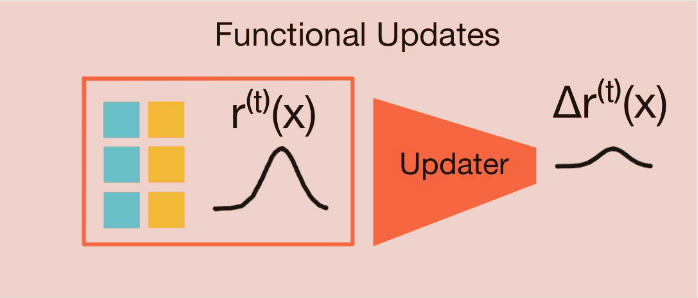
MetaFun and Functional Gradient Descent

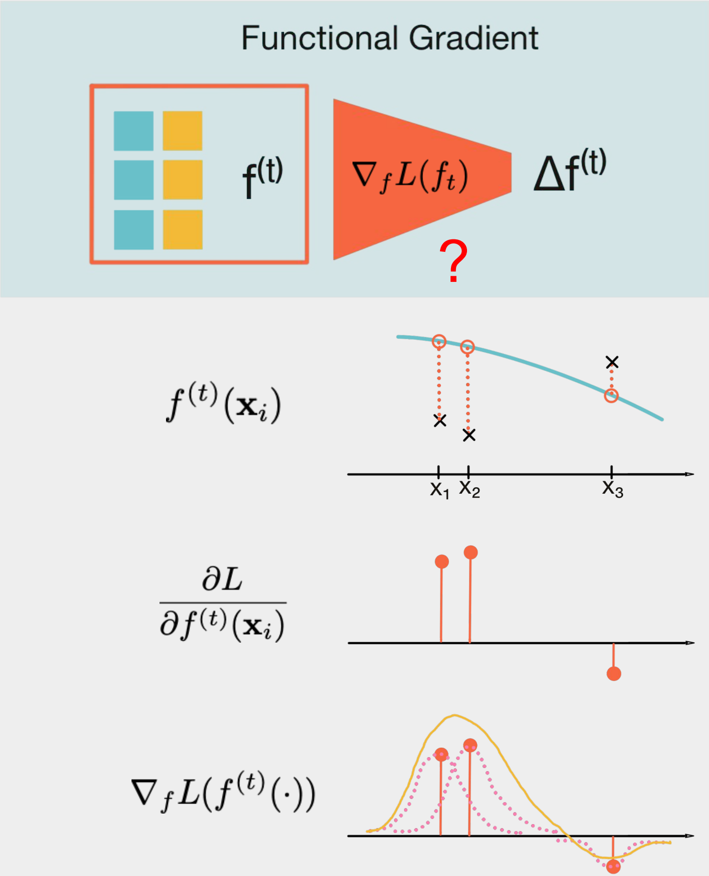




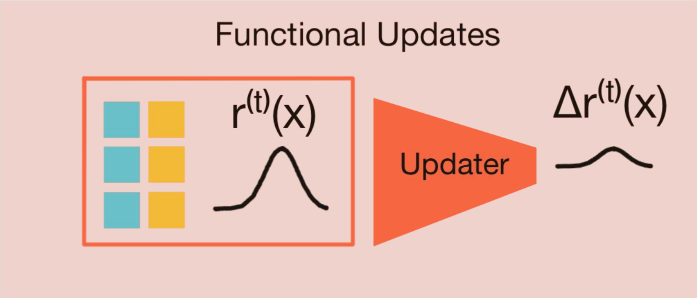


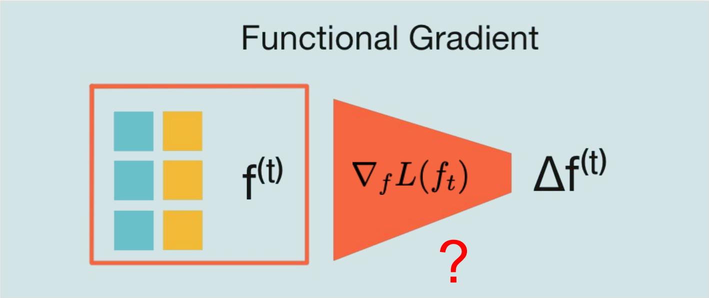
MetaFun and Functional Gradient Descent





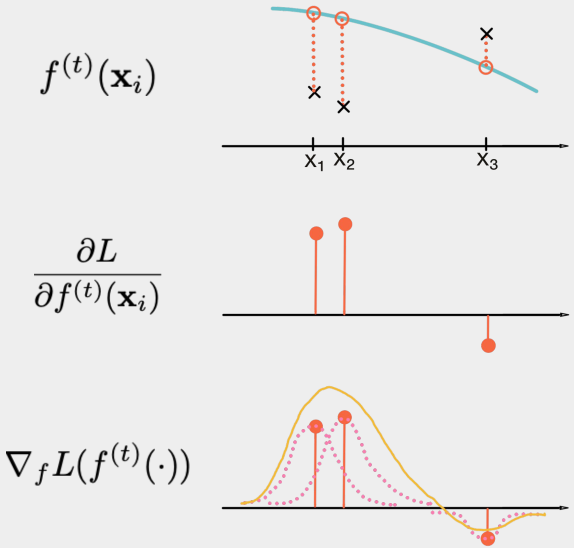
MetaFun and Functional Gradient Descent



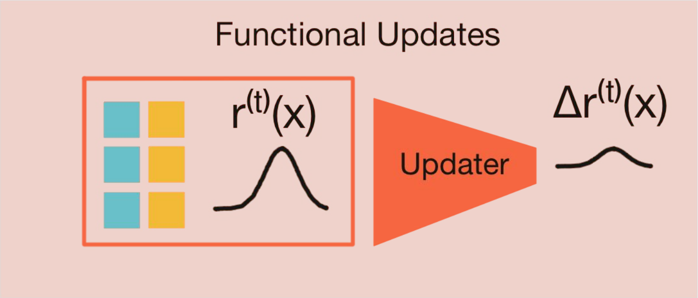


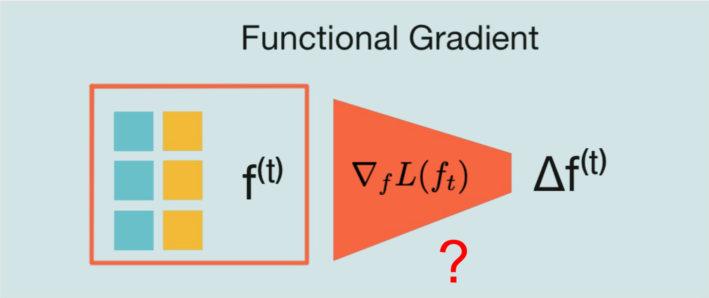
Evaluate functional representation at context:

$$r^{(t)}(\mathbf{x}_i), \forall i \in C$$



MetaFun and Functional Gradient Descent



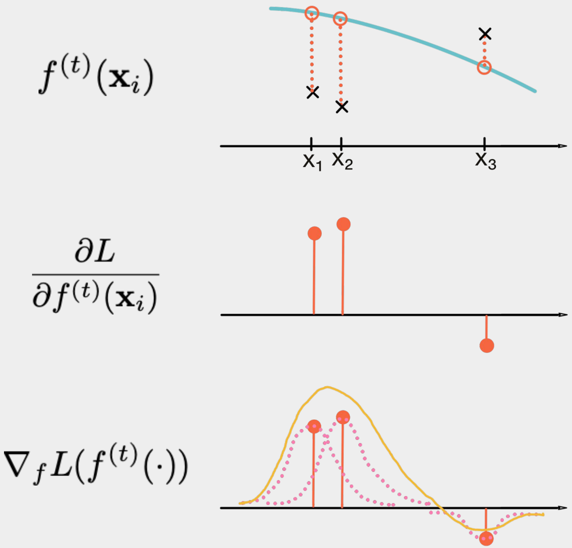


Evaluate functional representation at context:

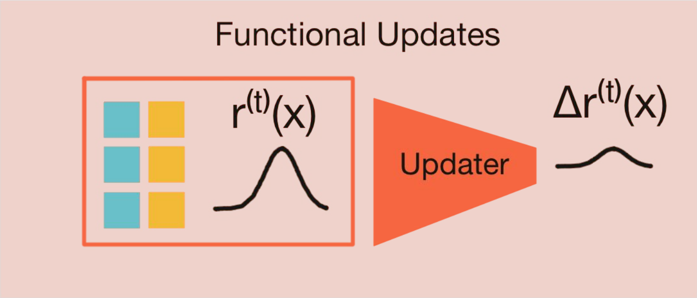
$$r^{(t)}(\mathbf{x}_i), \forall i \in C$$

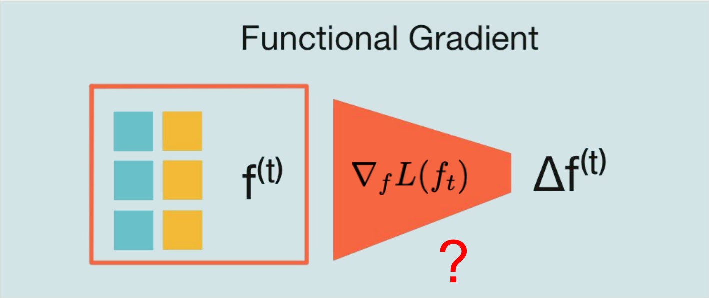
Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$



MetaFun and Functional Gradient Descent





Evaluate functional representation at context:

$$r^{(t)}(\mathbf{x}_i), \forall i \in C$$

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

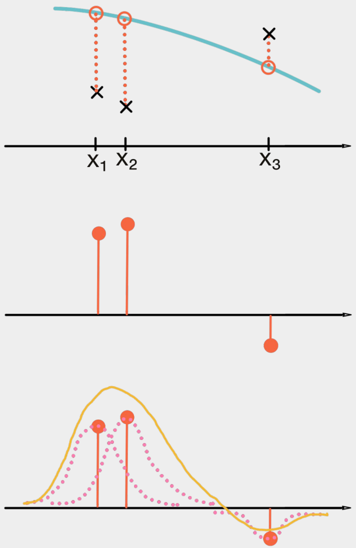
Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

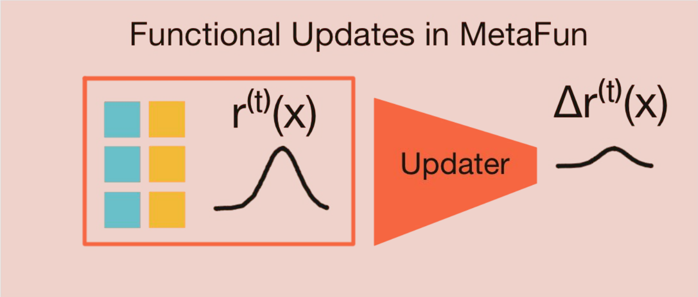
$$f^{(t)}(\mathbf{x}_i)$$

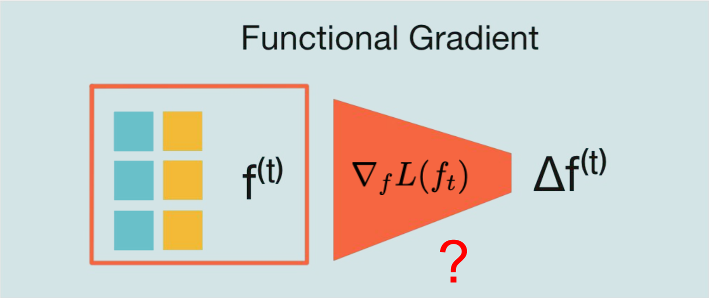
$$\frac{\partial L}{\partial f^{(t)}(\mathbf{x}_i)}$$

$$\nabla_f L(f^{(t)}(\cdot))$$



MetaFun and Functional Gradient Descent





Evaluate functional representation at context:

$$r^{(t)}(\mathbf{x}_i), \forall i \in C$$

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

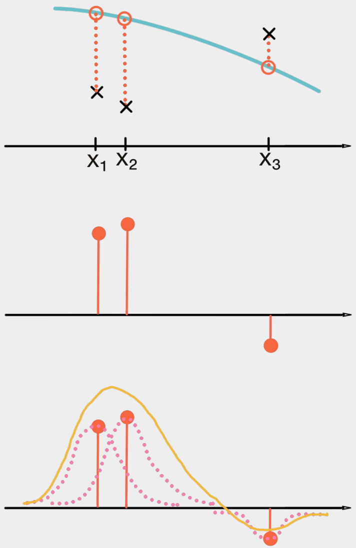
Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

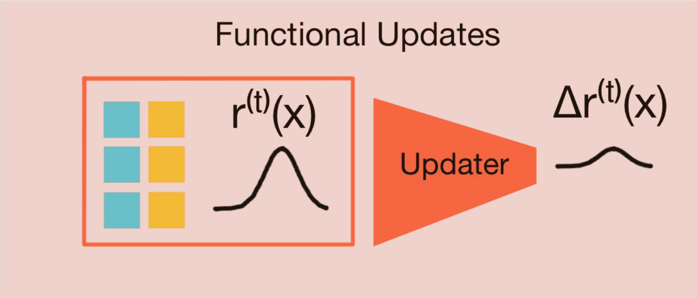
$$f^{(t)}(\mathbf{x}_i)$$

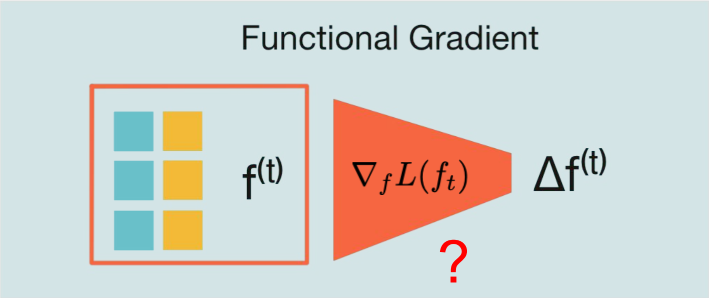
$$\frac{\partial L}{\partial f^{(t)}(\mathbf{x}_i)}$$

$$\nabla_f L(f^{(t)}(\cdot))$$



MetaFun and Functional Gradient Descent





Evaluate functional representation at context:

$$r^{(t)}(\mathbf{x}_i), \forall i \in C$$

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

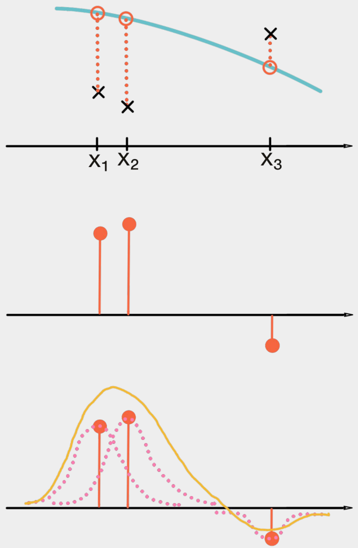
Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

$f^{(t)}(\mathbf{x}_i)$

$\dfrac{\partial L}{\partial f^{(t)}(\mathbf{x}_i)}$

$\nabla_f L(f^{(t)}(\cdot))$



MetaFun

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

$r^{(T)}(\cdot)$ will be the final representation after T iterations

MetaFun

Functional Representation

Permutation invariance ✓

Flexible capacity ✓

Within-context and context-target interaction

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

MetaFun

Functional Representation

Permutation invariance ✔

Flexible capacity ✔

Within-context and context-target interaction ✔

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

Both the within-context interaction and the interaction between context and target are considered when updating the representation at each iteration.

MetaFun

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

MetaFun for Classification

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

Deep kernels or attention modules

MetaFun for Classification

Regression:
MLP on concatenation of inputs

Classification:
?

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

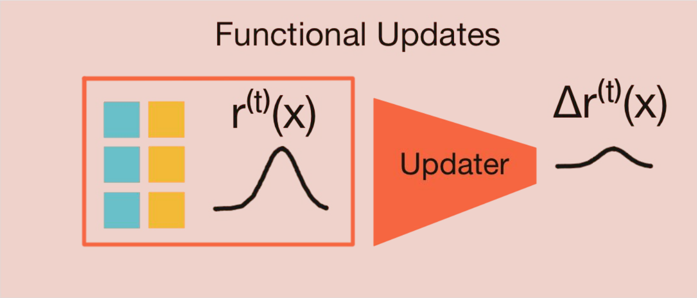
$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

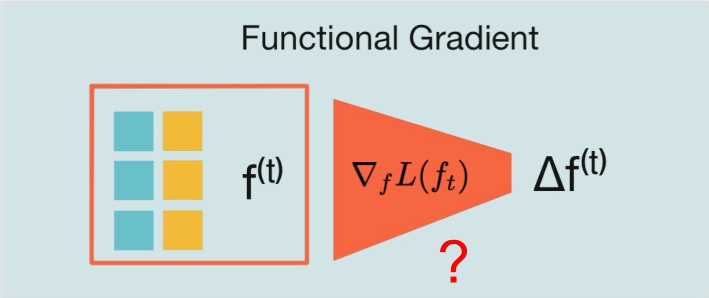
Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

Deep kernels or attention modules

MetaFun for Classification





Evaluate functional representation at context:

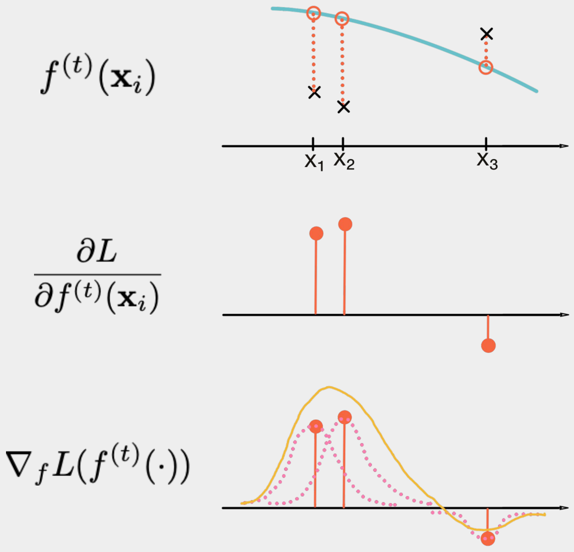
$$r^{(t)}(\mathbf{x}_i), \forall i \in C$$

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$



MetaFun for Classification

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

$$\frac{\partial L}{\partial f^{(t)}(\mathbf{x}_i)}$$

MetaFun for Classification

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

$$\frac{\partial L}{\partial f^{(t)}(\mathbf{x}_i)} \longrightarrow \left[\frac{\partial\ (\text{cross entropy loss})}{\partial\ (\text{predictive logit } k)}\right]^{\top}_{k=1:K}$$

MetaFun for Classification

Regression:
MLP on concatenation of inputs

Classification:
With structure similar to

$$\left[\frac{\partial\,(\text{cross entropy loss})}{\partial\,(\text{predictive logit } k)}\right]^{\top}_{k=1:K}$$

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

Deep kernels or attention modules

MetaFun for Classification

Regression:
MLP on concatenation of inputs

Classification:
With structure similar to

$$\left[\frac{\partial\,(\text{cross entropy loss})}{\partial\,(\text{predictive logit } k)}\right]_{k=1:K}^{\top}$$

Incorporate label information into the network structure rather than concatenating the label to the inputs

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i\in C}) = \sum_{i\in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

Deep kernels or attention modules

MetaFun for Classification

Regression:
MLP on concatenation of inputs

Classification:
With structure similar to

$$\left[\frac{\partial\ (\text{cross entropy loss})}{\partial\ (\text{predictive logit } k)}\right]^{\top}_{k=1:K}$$

Incorporate label information into the network structure rather than concatenating the label to the inputs

Naturally integrate within-class and between-class interaction

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

Deep kernels or attention modules

MetaFun and Gradient-Based Meta-Learning

Model Agnostic Meta-Learning (MAML)[8]

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

MetaFun and Gradient-Based Meta-Learning

Model Agnostic Meta-Learning (MAML)[8]

During meta-training phase, MAML finds a good initialisation from related tasks.

During test time, MAML runs a few gradient descent steps from the learned initialisation on the context of a new task.

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \text{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

MetaFun and Gradient-Based Meta-Learning

Model Agnostic Meta-Learning (MAML)[8]

During meta-training phase, MAML finds a good initialisation from related tasks.

During test time, MAML runs a few gradient descent steps from the learned initialisation on the context of a new task.

$$\theta_{t+1} = \theta_t - \alpha \sum_{i \in C} \nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)$$

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

MetaFun and Gradient-Based Meta-Learning

Model Agnostic Meta-Learning (MAML)[8]

During meta-training phase, MAML finds a good initialisation from related tasks.

During test time, MAML runs a few gradient descent steps from the learned initialisation on the context of a new task.

$$\theta_{t+1} = \theta_t - \alpha \underbrace{\sum_{i \in C}}_{\text{SumPooling}} \underbrace{\nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)}_{\text{Local updates}}$$

SumPooling (permutation-invariant)

Local updates (following gradient)

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

MetaFun and Gradient-Based Meta-Learning

Model Agnostic Meta-Learning (MAML)[8]

During meta-training phase, MAML finds a good initialisation from related tasks.

During test time, MAML runs a few gradient descent steps from the learned initialisation on the context of a new task.

$$\theta_{t+1} = \theta_t - \alpha \underbrace{\sum_{i \in C}}_{\text{SumPooling}} \underbrace{\nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)}_{\text{Local updates}}$$

SumPooling (permutation-invariant)

Local updates (following gradient)

MetaFun Iteration

Local update funcion:

$$\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$$

Functional pooling:

$$\Delta r^{(t)}(\cdot) = \textsc{FunPooling}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i \in C}) = \sum_{i \in C} k(\cdot, \mathbf{x}_i)\mathbf{u}_i$$

Apply functional updates:

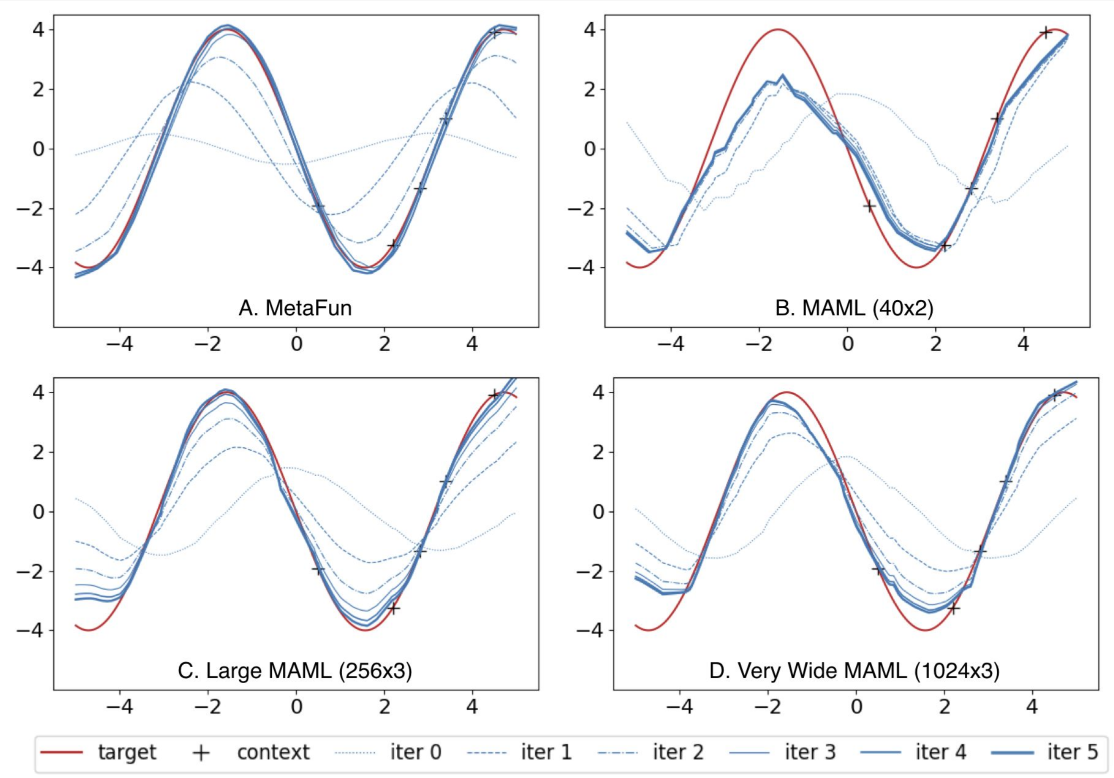
$$r^{(t+1)}(\cdot) = r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)$$

FunPooling

Local update function (parameterised by NNs)

MetaFun and Gradient-Based Meta-Learning

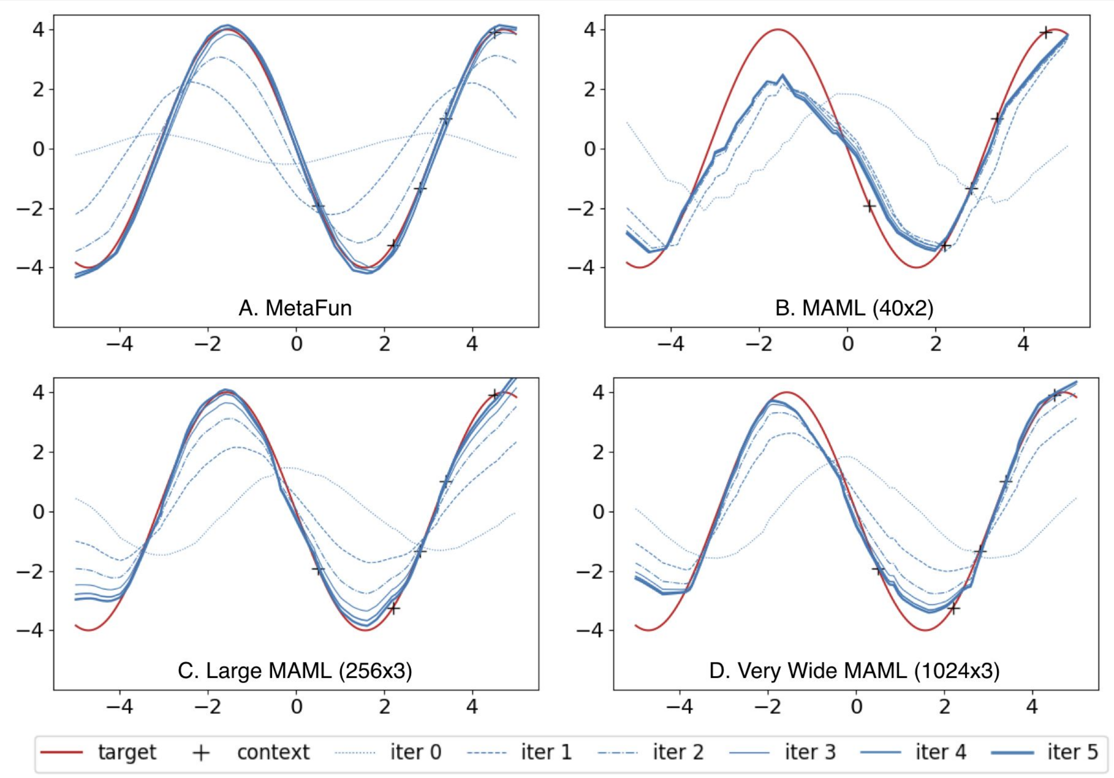
1D Sinusoid Regression Tasks



MetaFun and Gradient-Based Meta-Learning

1D Sinusoid Regression Tasks

MetaFun:
Smooth updates and match the ground truth very well across the whole period.

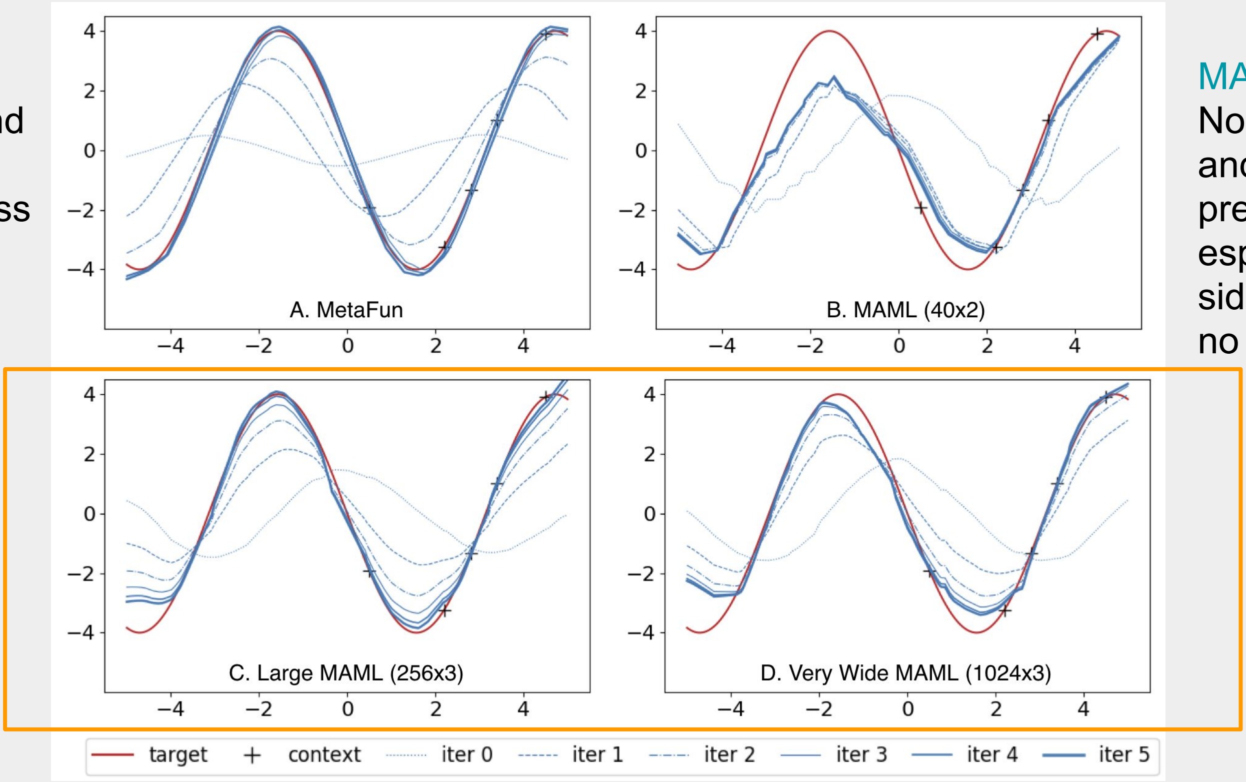


MAML:
Non-smooth updates and not as good predictions especially on the left side where there is no context points.

MetaFun and Gradient-Based Meta-Learning

1D Sinusoid Regression Tasks

MetaFun:
Smooth updates and match the ground truth very well across the whole period.



MAML:
Non-smooth updates and not as good predictions especially on the left side where there is no context points.

Large-Scale Few-shot Classification

miniImageNet

(without data augmentation)

Model	1-shot	5-shot
LEO[9]	**61.76 ± 0.08%**	77.59 ± 0.12%
MetaFun (deep kernel version)	61.16 ± 0.15%	**78.20 ± 0.16%**
MetaFun (attention version)	**62.12 ± 0.30%**	77.78 ± 0.12%

(with data augmentation)

Model	1-shot	5-shot
LEO	63.97 ± 0.20%	79.49 ± 0.70%
MetaOptNet-SVM[10]	**64.09 ± 0.62%**	80.00 ± 0.45%
MetaFun (deep kernel version)	63.39 ± 0.15%	**80.81 ± 0.10%**
MetaFun (attention version)	**64.13 ± 0.13%**	**80.82 ± 0.17%**

tieredImageNet

(without data augmentation)

Model	1-shot	5-shot
LEO	66.33 ± 0.05%	81.44 ± 0.09%
MetaOptNet-SVM	65.81 ± 0.74%	81.75 ± 0.58%
MetaFun (deep kernel version)	67.27 ± 0.20%	**83.28 ± 0.12%**
MetaFun (attention version)	**67.72 ± 0.14%**	82.81 ± 0.15%

Large-Scale Few-shot Classification

We demonstrates that encoder-decoder style meta-learning methods like conditional neural processes can also also achieves SOTA on large-scale few-shot classification benchmarks.

miniImageNet

(without data augmentation)

Model	1-shot	5-shot
LEO[9]	**61.76 ± 0.08%**	77.59 ± 0.12%
MetaFun (deep kernel version)	61.16 ± 0.15%	**78.20 ± 0.16%**
MetaFun (attention version)	**62.12 ± 0.30%**	77.78 ± 0.12%

(with data augmentation)

Model	1-shot	5-shot
LEO	63.97 ± 0.20%	79.49 ± 0.70%
MetaOptNet-SVM[10]	**64.09 ± 0.62%**	80.00 ± 0.45%
MetaFun (deep kernel version)	63.39 ± 0.15%	**80.81 ± 0.10%**
MetaFun (attention version)	**64.13 ± 0.13%**	**80.82 ± 0.17%**

tieredImageNet

(without data augmentation)

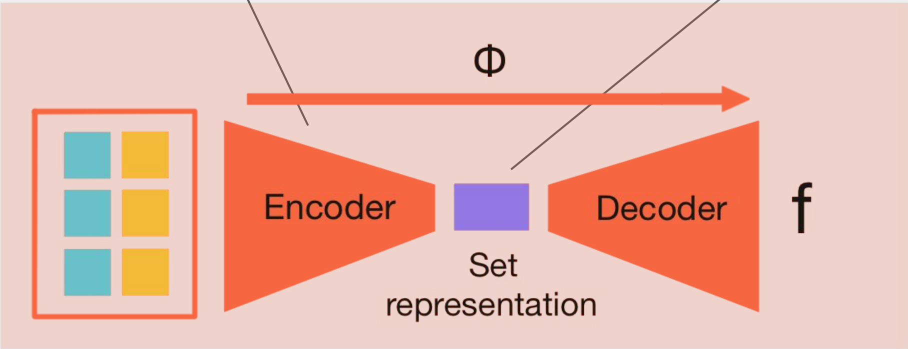
Model	1-shot	5-shot
LEO	66.33 ± 0.05%	81.44 ± 0.09%
MetaOptNet-SVM	65.81 ± 0.74%	81.75 ± 0.58%
MetaFun (deep kernel version)	67.27 ± 0.20%	**83.28 ± 0.12%**
MetaFun (attention version)	**67.72 ± 0.14%**	82.81 ± 0.15%

Large-Scale Few-shot Classification

We demonstrates that encoder-decoder style meta-learning methods like conditional neural processes can also also achieves SOTA on large-scale few-shot classification benchmarks.

Iterative structure for the encoder?

Functional set representation



Thank you!

jin.xu@stats.ox.ac.uk

@jinxu06 (code available here)

@jinxu06

References

[1] Garnelo, Marta, et al. "Conditional Neural Processes." International Conference on Machine Learning. 2018.
[2] Garnelo, Marta, et al. "Neural processes." arXiv preprint arXiv:1807.01622 (2018).
[3] Kim, Hyunjik, et al. "Attentive neural processes." International Conference on Learning Representations. 2019.
[4] Wagstaff, Edward, et al. "On the Limitations of Representing Functions on Sets." International Conference on Machine Learning. 2019.
[5] Bloem-Reddy, B. and Teh, Y. W. "Probabilistic symmetries and invariant neural networks." Journal of Machine Learning Research, 21(90):1–61, 2020.
[6] Lee, Juho, et al. "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks." International Conference on Machine Learning. 2019.
[7] Zaheer, Manzil, et al. "Deep sets." Advances in neural information processing systems. 2017.
[8] Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." International Conference on Machine Learning. 2017.
[9] Rusu, Andrei A., et al. "Meta-learning with latent embedding optimization." International Conference on Learning Representations. 2019.
[10] Lee, Kwonjoon, et al. "Meta-learning with differentiable convex optimization." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.