MetaFun: Meta-Learning with Iterative Functional Updates

Jin Xu [1] **Jean-Francois Ton** [1] **Hyunjik Kim** [2] **Adam R. Kosiorek** [1 3] **Yee Whye Teh** [1]

Abstract

We develop a functional encoder-decoder approach to supervised meta-learning, where labeled data is encoded into an infinite-dimensional functional representation rather than a finite-dimensional one. Furthermore, rather than directly producing the representation, we learn a neural update rule resembling functional gradient descent which iteratively improves the representation. The final representation is used to condition the decoder to make predictions on unlabeled data. Our approach is the first to demonstrates the success of encoder-decoder style meta-learning methods like conditional neural processes on large-scale few-shot classification benchmarks such as miniImageNet and tieredImageNet, where it achieves state-of-the-art performance.

1. Introduction

The goal of meta-learning is to be able to generalise to new tasks from the same task distribution as the training tasks. In supervised meta-learning, a task can be described as making predictions on a set of unlabelled data points (*target*) by effectively learning from a set of data points with labels (*context*). Various ideas have been proposed to tackle supervised meta-learning from different perspectives (Andrychowicz et al., 2016; Ravi & Larochelle, 2016; Finn et al., 2017; Koch, 2015; Snell et al., 2017; Vinyals et al., 2016; Santoro et al., 2016; Rusu et al., 2019). In this work, we are particularly interested in a family of meta-learning models that use an encoder-decoder pipeline, such as Neural Processes (Garnelo et al., 2018a;b). The encoder is a permutation-invariant function on the context that summarises the context into a task representation, while the decoder produces a predictive model for the targets, conditioned on the task representation. The objective of meta-learning is then to learn the encoder and the decoder such that the produced predictive model generalises well to the targets of new tasks.

Previous works in this category such as the Conditional Neural Process (CNP) and the Neural Process (NP) (Garnelo et al., 2018a;b) use sum-pooling operations to produce finite-dimensional, vectorial, task representations. In this work, we investigate the idea of summarising tasks with infinite-dimensional functional representations. Although there is a theoretical guarantee that sum-pooling of instance-wise representations can express any set function (*universality*) (Zaheer et al., 2017; Bloem-Reddy & Teh, 2020), in practice CNP and NP tend to underfit the context (Kim et al., 2019). This observation is in line with the theoretical finding that for universality, the dimension of the task representation should be at least as large as the cardinality of the context set, if the encoder is a smooth function Wagstaff et al. (2019). We develop a method that explicitly uses functional representations. Here the effective dimensionality of the task representation grows with the number of context points, which addresses the aforementioned issues of fixed-dimensional representations.

Moreover, in practice it is difficult to model interactions between data points with sum-pooling after instance-wise transformations, and inserting modules such as relation networks (Sung et al., 2018; Rusu et al., 2019) or set transformers (Lee et al., 2019a) before sum-pooling can only model within-context interactions but not context-target interactions. The construction of our functional representation involves measuring similarities between all data points, which naturally contains information regarding interactions between elements in either the context or the target.

Furthermore, rather than producing the functional representation in a single pass, we develop an approach that *learns* iterative updates to encode the context into the task representation. In general, learning via iterative updates is often easier than directly learning the final representation, because of the error-correcting opportunity at each iteration. For example, an iterative parameterisation of the encoder in Variational Autoencoders (VAEs) has been demonstrated to be effective in reducing the amortisation gap (Marino et al., 2018), while in meta-learning, both learning to learn methods (Andrychowicz et al., 2016; Ravi & Larochelle,

[1]Department of Statistics, University of Oxford, Oxford, United Kingdom [2]Google DeepMind, London, United Kingdom [3]Applied AI Lab, Oxford Robotics Institute, University of Oxford, Oxford United Kingdom. Correspondence to: Jin Xu <jin.xu@stats.ox.ac.uk>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

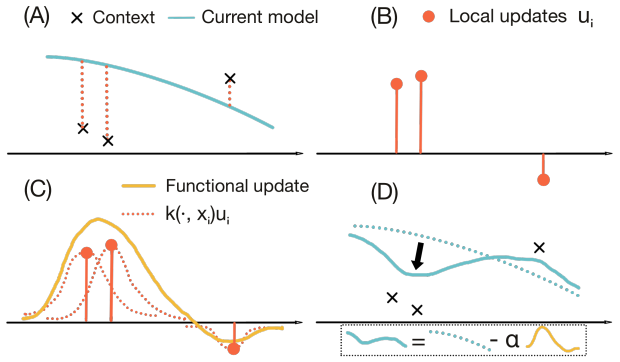

Figure 1. To illustrate the iterative procedure in MetaFun, we consider a simplified case where the functional representation of the task is the predictor itself. (A) This figure depicts a 1D regression task. (B) Local updates are computed by evaluating the functional representation (the current predictor) on the context inputs, and comparing it to corresponding context outputs. Here we simply measure differences between the evaluations (predictions) and the outputs. (C) We use functional pooling to aggregate local updates into a global functional update, which generalises the local updates to the whole input domain. (D) The functional update is applied to the current functional representation with a learning rate α.

2016) and Model Agnostic Meta Learning (MAML) (Finn et al., 2017) use iterative updating procedures to adapt to new tasks, although these update rules operate in parameter space rather than function space. Therefore, it is reasonable to conjecture that iterative structures are favourable inductive biases for the task encoding process.

In summary, the primary contribution of this work is a meta-learning approach that learns to summarise a task using a functional representation constructed via iterative updates. We apply our approach to solve meta-learning problems on both regression and classification tasks, and achieve state-of-the-art performance on heavily benchmarked datasets such as miniImageNet (Vinyals et al., 2016) and tieredImageNet (Ren et al., 2018), which has never been demonstrated with encoder-decoder meta-learning methods without MAML-style gradient updates. We also conducted an ablation study to understand the effects of the different model components.

2. MetaFun

Meta-learning, or learning to learn, leverages past experiences to quickly adapt to tasks $\mathcal{T} \sim p(\mathcal{T})$ drawn iid from some task distribution. In supervised meta-learning, a task $\mathcal{T}$ takes the form of $\mathcal{T} = \{\ell, \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}, \{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{j\in\mathbf{T}}\}$, where $\mathbf{x}_i, \mathbf{x}'_j \in \mathcal{X}$ are inputs, $\mathbf{y}_i, \mathbf{y}'_j \in \mathcal{Y}$ outputs, ℓ is the loss function to be minimised, $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$ is the context, and $\{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{j\in\mathbf{T}}$ is the target. We consider the process of learning as constructing a predictive model using the task context and refer to the mapping from context $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$ to a predictive model $f = \Phi(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}; \phi)$ as the *learning model* parameterised by ϕ. In our formulation, the objective of meta-learning is to optimise the learning model such that the expected loss on the target under f is minimised, formally written as:

$$f = \Phi(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}; \phi)$$
$$\phi^* = \arg\min_{\phi} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\frac{1}{|\mathbf{T}|} \sum_{j\in\mathbf{T}} \ell(f(\mathbf{x}'_j), \mathbf{y}'_j) \right], \quad (1)$$

where both $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$, $\{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{i\in\mathbf{T}}$ come from task $\mathcal{T}$.

2.1. Learning Functional Task Representation

Like previous works such as CNP and NP, we construct the learning model using an encoder-decoder pipeline, where the encoder $\Phi_e(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}; \phi_e)$ is a permutation-invariant function of the context producing a task representation. In past works, pooling operations are usually used to enforce permutation-invariance. CNP and NP use sum-pooling: $\mathbf{r} = \sum_{i\in\mathbf{C}} \mathbf{r}_i$, where $\mathbf{r}_i = h(\mathbf{x}_i, \mathbf{y}_i; \phi_e)$ is a representation for context pair $\mathbf{x}_i, \mathbf{y}_i$, and $\mathbf{r}$ is a fixed-dimensional task representation. Instead, we introduce functional-pooling operations, which also enforce permutation-invariance but output a function that can loosely be interpreted as an infinite-dimensional representation.

Definition 2.1 (Functional pooling)**.** Let $k(\cdot,\cdot)$ be a real-valued similarity measure, and $\{(\mathbf{x}_i, \mathbf{r}_i)\}_{i\in\mathbf{C}}$ be a set of key-value pairs with $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{r}_i \in \mathcal{R}$. *Functional pooling* is a mapping $\text{FUNPOOLING} : (\mathcal{X} \times \mathcal{R})^{|\mathbf{C}|} \to \mathcal{H}$ defined as

$$r(\cdot) = \text{FUNPOOLING}(\{(\mathbf{x}_i, \mathbf{r}_i)\}_{i\in\mathbf{C}}) = \sum_{i\in C} k(\cdot, \mathbf{x}_i)\mathbf{r}_i\,, \quad (2)$$

where the output is a function $r : \mathcal{X} \to \mathcal{R}$ and $\mathcal{H}$ is a space of such functions.

In practice, we only need to evaluate this function on a finite query set $\{(\mathbf{x}'_j, \mathbf{y}'_j)\}_{j\in\mathcal{Q}}$ (consisting of both contexts and targets; see below). That is, we only need to compute $R = [r(\mathbf{x}'_1), \ldots, r(\mathbf{x}'_{|\mathcal{Q}|})]^\top$, which can be easily implemented using matrix operations. We consider two types of FUNPOOLING here, though others are possible. The kernel-based FUNPOOLING reads as,

$$R = \text{KFP}\,(Q, K, V) := k_{\text{rbf}}(Q, K)V\,, \quad (3)$$

where k_{rbf} is the RBF kernel, $Q = [a(\mathbf{x}'_1), \ldots, a(\mathbf{x}'_{|\mathcal{Q}|})]^\top$ is a matrix whose rows are queries, $a(\cdot)$ is a transformation mapping inputs into features, $K = [a(\mathbf{x}_1), \ldots, a(\mathbf{x}_{|C|})]^\top$ a matrix whose rows are keys, and $V = [\mathbf{r}_1, \ldots, \mathbf{r}_{|C|}]^\top$ a matrix whose rows are values (using terminology from the attention literature). Parameterising input transformation a with a deep neural network can be seen as using deep kernels

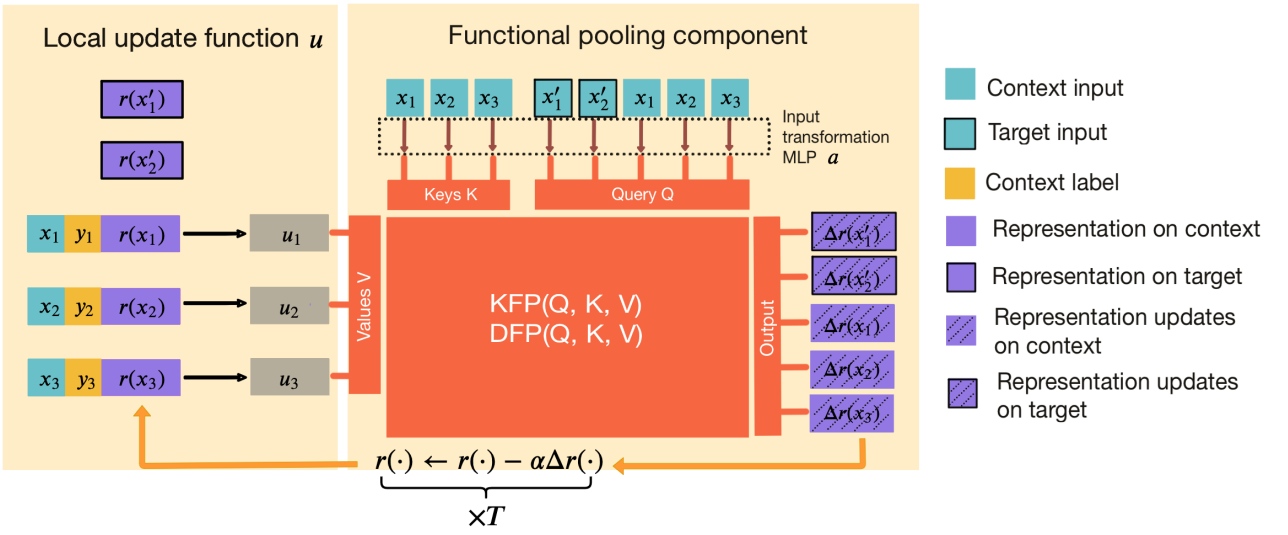

Figure 2. This figure illustrates the iterative computation of functional representation in MetaFun. At each iteration, we first evaluate the current functional representation at both context and target points. Then, the shared local update function u takes in each context point and the corresponding evaluation as inputs, and produces local updates u_i. Next, we apply (kernel-based or attention-based) functional pooling to aggregate local updates u_i into a functional update $\Delta r(\cdot)$, which at each query is a linear combination of all the u_i weighted by similarities between the query and all keys. Finally, the functional update is evaluated at both context and target inputs, and applied to the corresponding evaluations of functional representation with a learning rate α.

(Wilson et al., 2016) as the similarity measure. The second type of FUNPOOLING is given by dot-product attention,

$$R = \text{DFP}\,(Q, K, V) := \text{softmax}(QK^\top/\sqrt{d_k})V\,, \quad (4)$$

where d_k is the dimension of the query/key vectors.

Our second core idea is that rather than producing the task representation in a single pass (like previous encoder-decoder meta-learning approaches), we start from an initial representation $r^{(0)}(\cdot)$, and iteratively produce improved representations $r^{(1)}(\cdot), \ldots, r^{(T)}(\cdot)$. At each step, a parameterised local update rule u compares $r^{(t)}(\cdot)$ to the context input/output pairs, producing local update values $\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))$ for each $i \in \mathbf{C}$. These can then be aggregated into a global update to the task representation using functional pooling,

$$\begin{aligned} \mathbf{u}_i &= u(\mathbf{x}_i, \mathbf{y}_i, r^{(t)}(\mathbf{x}_i))\,, \\ \Delta r^{(t)}(\cdot) &= \text{FUNPOOLING}(\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i\in\mathbf{C}})\,, \\ r^{(t+1)}(\cdot) &= r^{(t)}(\cdot) - \alpha \Delta r^{(t)}(\cdot)\,, \end{aligned} \quad (5)$$

where α is the step size. Once the local update function u and the functional pooling operations are parameterised by neural networks, Equation (5) defines a neural update rule operating directly in function space. The functional update $\Delta r^{(t)}(\cdot)$ depends on the current representation $r^{(t)}(\cdot)$ and the context $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in\mathbf{C}}$. Figure 1 illustrates our iterative procedure in a simplified setting.

The final task representation can then be decoded into a predictor $f(\cdot) = \Phi_d(r^{(T)}(\cdot); \phi_d)$. The specific parametric forms of the decoder take different forms for regression and classification, and are described in Section 2.2. The decoder requires the evaluation of functional representation $r^{(T)}(\mathbf{x})$ at $\mathbf{x}$ only for predicting $f(\mathbf{x})$. Therefore, it is unnecessary to compute the functional representations $r(\cdot)$ (including their functional updates) on all input points. Instead, we compute them only on the context $\{\mathbf{x}_i\}_{i\in\mathbf{C}}$ and target inputs $\{\mathbf{x}'_j\}_{j\in\mathbf{T}}$. We use $\mathbf{r}^{(t)} = [r^{(t)}(\mathbf{x}_1) \ldots r^{(t)}(\mathbf{x}_{|\mathbf{C}|}), r^{(t)}(\mathbf{x}'_1) \ldots r^{(t)}(\mathbf{x}'_{|\mathbf{T}|})]^\top$ to denote a matrix where each row is $r^{(t)}(\mathbf{x})$ evaluated on either context or target inputs, and let $Q = [a(\mathbf{x}_1) \ldots a(\mathbf{x}_{|\mathbf{C}|}), a(\mathbf{x}'_1) \ldots a(\mathbf{x}'_{|\mathbf{T}|})]^\top$. Equation (5) can be implemented using matrix computations as follows,

$$\mathbf{u}_i^{(t)} = u(\mathbf{x}_i, \mathbf{y}_i, \mathbf{r}_i^{(t)})\,, \quad (6)$$

$$U^{(t)} = [\mathbf{u}_1^{(t)}, \ldots, \mathbf{u}_{|\mathbf{C}|}^{(t)}]^\top\,, \quad (7)$$

$$\Delta \mathbf{r}^{(t)} = \text{KFP or DFP}\left(Q, K, U^{(t)}\right)\,, \quad (8)$$

$$\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} - \alpha \Delta \mathbf{r}^{(t)} \quad (9)$$

where $\mathbf{r}_i^{(t)}$ denotes the i-th row of $\mathbf{r}^{(t)}$.

To obtain a prediction $\mathbf{f}_j$ for the the target $(\mathbf{x}'_j, \mathbf{y}'_j)$, we decode the final representation $\mathbf{f}_j = \Phi_d(\mathbf{r}^{(T)}_{|\mathbf{C}|+j}; \phi_d)$. The whole procedure can be learnt jointly by optimising the

following loss:

$$L(u, a, \phi_d) = \frac{1}{|\mathbf{T}|} \sum_{j \in \mathbf{T}} \ell(\mathbf{f}_j, \mathbf{y}'_j)\,, \tag{10}$$

where $\mathbf{f}_j$ depends on u, a, ϕ_d.

Assuming the width and depth of all our neural network components are bounded by W and D, respectively, and the output dimension of u is also less than W, the time complexity of our approach is $\mathcal{O}\big(W|\mathbf{C}|(|\mathbf{C}|+|\mathbf{T}|)+W^2D(|\mathbf{C}|T+|\mathbf{T}|)\big)$, and the space complexity is $\mathcal{O}\big((|\mathbf{C}|+WT)(|\mathbf{C}|+|\mathbf{T}|)+W^2D\big)$. For few-shot problems, $|\mathbf{C}|$ and $|\mathbf{T}|$ are typically small, and $T \leq 6$ in all our experiments.

2.2. MetaFun for Regression and Classification

While the proposed framework can be applied to any supervised learning task, the specific parameterisation of learnable components can affect the model performance. In this section, we specify the parametric forms of our model that work well on regression and classification tasks.

Regression For regression tasks, we parameterise the local update function $u(\cdot)$ using a multi-layer perceptron as $u([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = \text{MLP}\left([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]\right), i \in C$, where $[\cdot]$ is concatenation. We also use an MLP to parametrise the input transformation $a(\cdot)$ in the functional pooling. The decoder in this case is given by $\mathbf{w} = \text{MLP}\left(r(\mathbf{x})\right)$, another MLP[1] that outputs $\mathbf{w}$, which then parameterises the predictive model $f = \text{MLP}\left(\mathbf{x}; \mathbf{w}\right)$.

Note that our model can easily be modified to incorporate Gaussian uncertainty by adding an extra output vector for the predictive standard deviation: $P(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mu_{\mathbf{w}}(\mathbf{x}), \sigma_{\mathbf{w}}(\mathbf{x})), \mathbf{w} = \text{MLP}\left(r(\mathbf{x})\right)$. For further architecture details, see Appendix.

Classification For K-way classification, we divide the latent functional representation $r(\mathbf{x})$ into K parts $[r^1(\mathbf{x}), \ldots, r^K(\mathbf{x})]$, where $r^k(\mathbf{x})$ corresponds to the class k. Consequently, the local update function $u(\cdot)$ also has K parts, that is, $u([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = [u^1(\cdot), \ldots, u^K(\cdot)]$. In this case, $\mathbf{y}_i = [y_i^1, \ldots, y_i^K]$ is the class label expressed as a one-hot vector; the u^k is defined as follows,

$$\begin{aligned} u^k([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = {} & y_i^k u_+(m(r^k(\mathbf{x}_i)), \boldsymbol{m}_i) \\ & + (1 - y_i^k) u_-(m(r^k(\mathbf{x}_i)), \boldsymbol{m}_i)\,, \end{aligned} \tag{11}$$

where $\boldsymbol{m}_i = \sum_{k=1}^{K} m(r^k(\mathbf{x}_i))$ summarises representations of all classes, and m, u_+, u_- are parameterised by separate MLPs. With this formulation, we update the class representations using either u_+ (when the label matches k) or u_- (when the label is different to k), so that labels are not concatenated to the inputs, but directly used to activate different model components, which is crucial for performance. Furthermore, interactions between data points for classification problems include both within-class and between-class interactions. Our approach is able to integrate two types of interactions by having separate class functional representations and computing local updates for each class differently based on class membership of each data point. In fact, this formulation resembles the structure of the local update rule in functional gradient descent for classification tasks, which is a special case of our approach (see Section 3). Same as in regression tasks, the input transformation $a(\cdot)$ in the functional pooling is still an MLP. The parametric form of the decoder is the same as in Latent Embedding Optimisation (LEO) (Rusu et al., 2019). The class representation $r^k(\mathbf{x})$ generates weights $\mathbf{w^k} \sim \mathcal{N}(\mu(r^k(\mathbf{x})), \sigma(r^k(\mathbf{x})))$ where μ and σ are MLPs or just linear functions, and the final prediction is given by

$$P(\mathbf{y} = k|\mathbf{x}) = \text{softmax}(\mathbf{x}^T\mathbf{w})_k\,, \tag{12}$$

where $\mathbf{w} = [\mathbf{w^1}, \ldots, \mathbf{w^K}]$, $k = 1, \ldots, K$. Hyperparameters of all components are described in Appendix.

3. Related Work

Functional Gradient Descent Functional gradient descent (Mason et al., 1999; Y. Guo & Williamson, 2001) is an optimisation algorithm used to minimise the objective function by moving in the direction of the negative gradient in function space. To ensure smoothness, we may work with functions in a Reproducing kernel Hilbert space (RKHS) (Aronszajn, 1950; Berlinet & Thomas-Agnan, 2011) defined by a kernel $k(\mathbf{x}, \mathbf{x}')$. Given a function f in the RKHS, we are interested in minimising the supervised loss $L(f) = \sum_{i \in C} \ell(f(\mathbf{x}_i), \mathbf{y}_i)$ with respect to f. We can do so by computing the functional derivative and use it to iteratively update f (see Appendix for more details),

$$f^{(t+1)}(\mathbf{x}) = f^{(t)}(\mathbf{x}) - \alpha \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \nabla \ell(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i) \tag{13}$$

with step size α, and $\nabla \ell(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i)$ denotes gradient w.r.t. to predictions in the loss function ℓ.

The update rule in Equation (5) becomes that of functional gradient descent in Equation (13) when

(i) A trivial decoder $f(\mathbf{x}) = \Phi_d(r(\mathbf{x}); \phi_d)(\mathbf{x}) = r(\mathbf{x})$ is used, so the functional representation $r(\mathbf{x})$ is the same as the predictive model $f(\mathbf{x})$.

(ii) Kernel functional pooling KFP is used and the kernel function is fixed.

[1] It might be desirable to use other parameterisations of the input transformation $a(\cdot)$, and the decoder $f(\cdot)$, e.g., $f(\mathbf{x}) = \text{MLP}\left([\mathbf{x}, r(\mathbf{x})]\right)$, or feeding $r(\mathbf{x})$ to each layer of the MLP.

(iii) Using gradient-based local update function $u(\mathbf{x}, \mathbf{y}, f(\mathbf{x})) = \nabla \ell(f(\mathbf{x}), \mathbf{y})$.

Furthermore, for a K-way classification problem, we predict K-dimensional logits $f(\mathbf{x}) = [f^1(\mathbf{x}), \dots, f^K(\mathbf{x})]^\top$, and use cross entropy loss:

$$\ell(f(\mathbf{x}), \mathbf{y}) = -\sum_{k=1}^{K} y^k \log \frac{e^{f^k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{f^{k'}(\mathbf{x})}}, \tag{14}$$

where $\mathbf{y} = [y^1, \dots, y^K]^\top$ is the one-hot label for $\mathbf{x}$.

The gradient-based local update function is now $\nabla \ell(f(\mathbf{x}), \mathbf{y}) = [\partial_1 \ell(f(\mathbf{x}), \mathbf{y}), \dots, \partial_K \ell(f(\mathbf{x}), \mathbf{y})]^\top$ and $\partial_k \ell(f(\mathbf{x}), \mathbf{y})$ is the partial derivative w.r.t. each predictive logit:

$$\partial_k \ell(f(\mathbf{x}), \mathbf{y}) = \frac{e^{f^k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{f^{k'}(\mathbf{x})}} - y^k. \tag{15}$$

Here $\partial_k \ell(f(\mathbf{x}), \mathbf{y})$ is analogous to $u^k(\cdot)$ in Equation (11), which is the local update function for class k.

If m, u_+, u_- in Equation (11) are specified rather than being learned:

$$\begin{aligned} m(f^{k'}(\mathbf{x})) &= e^{f^{k'}(\mathbf{x})} \\ u_+(m(f^k(\mathbf{x})), \boldsymbol{m}) &= \frac{m(f^k(\mathbf{x}))}{\boldsymbol{m}} - 1 \\ u_-(m(f^k(\mathbf{x})), \boldsymbol{m}) &= \frac{m(f^k(\mathbf{x}))}{\boldsymbol{m}} \\ \boldsymbol{m} &= \sum_{k=1}^{K} m(f^{k'}(\mathbf{x})), \end{aligned} \tag{16}$$

Equation (15) can be rewritten as:

$$\begin{aligned} \partial_k \ell(f(\mathbf{x}), \mathbf{y}) = y^k u_+(m(f^k(\mathbf{x})), \boldsymbol{m}) \\ + (1 - y^k) u_-(m(f^k(\mathbf{x})), \boldsymbol{m}), \end{aligned} \tag{17}$$

which has a similar form as Equation (11).

Therefore, our approach can be seen as an extension of functional gradient descent, with an additional learning capacity due learnable neural modules which afford more flexibility. From this perspective, our approach tackles supervised meta-learning problems by learning an optimiser in function space.

Supervised Meta-Learning Various ideas have been proposed to solve the problem of supervised meta-learning. Andrychowicz et al. (2016); Ravi & Larochelle (2016) learn the neural optimisers from previous tasks which can be used to optimise models for new tasks. However, these learned optimisers operate in parameter space rather than function space as we do. MAML (Finn et al., 2017) learns the initialisation from which models are further adapted for a new task by a few gradient descent steps. Koch (2015); Snell et al. (2017); Vinyals et al. (2016) explore the idea of learning a metric space from previous tasks in which data points are compared to each other to make predictions at test time. Santoro et al. (2016) demonstrate that Memory-Augmented Neural Networks (MANN) can rapidly integrate the data for a new task into memory, and utilise this stored information to make predictions.

Our approach, in line with previous works such as CNP and NP, adopt an encoder-decoder pipeline to tackle supervised meta-learning. The encoder in CNP corresponds to a summation of instance-level representations produced by a shared instance encoder. NPs, on the other hand, use a probabilistic encoder with the same parametric form as CNP, but producing a distribution of stochastic representation. The Attentive Neural Process (ANP) (Kim et al., 2019) adds a deterministic path in addition to the stochastic path in NP. The deterministic path produces a target-specific representation, which can be interpreted as applying functional pooling (implemented with multihead attention (Vaswani et al., 2017)) to instance-wise representation. However, the representation is directly produced in a single pass rather than iteratively improved as we do, and only regression applications are explored as opposed to few-shot image classification. In fact, for classification tasks, it is crucial for CNP to only apply sum-pooling within each class (Garnelo et al., 2018a), and it is unclear how to follow similar practices in ANP with both within-class and between-class interactions still being modelled. Recently, Gordon et al. (2019) have also extended CNP to use functional representations, but for the purpose of incorporating translation equivariance in the inputs as an inductive bias rather than increasing representational capacity as we do. Their approach uses convnets to impose translation equivariance and does not learn a flexible iterative encoder.

Pooling operations are usually used in encoder-decoder meta-learning to enforce permutation invariance in the encoder. As an example, encoders in both CNP and NP use simple *sum-pooling* operations. More expressive pooling operations have been proposed to model interactions between data points. Murphy et al. (2019) introduces *Janossy pooling* which applies permutation-sensitive functions to all reorderings and averages the outputs, while Lee et al. (2019a) use *pooling by multihead attention* (PMA), which uses a finite query set to attend to the processed key-value pairs. Loosely speaking, attention-based functional pooling can be seen as having the whole input domain $\mathcal{X}$ as the query set in PMA.

Gradient-Based Meta-Learning Interestingly, many gradient-based meta-learning methods such as MAML can

also be cast into an encoder-decoder formulation, because a gradient descent step is a valid permutation-invariant function. For a model $f(\cdot, \theta)$ parameterised by θ, one gradient descent step on the context loss has the following form,

$$\theta_{t+1} = \theta_t - \alpha \sum_{i \in C} \nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)\,, \tag{18}$$

where ℓ is the loss function, α is the learning rate, and θ_t are model parameters after t gradient steps. This corresponds to a special case of a permutation-invariant function where we take the instance-wise encoder to be $h_t(\mathbf{x}_i, \mathbf{y}_i; \theta_t) = \theta_t/|\mathbf{C}| - \alpha \nabla_\theta \ell(f(\mathbf{x}_i; \theta_t), \mathbf{y}_i)$ and apply sum-pooling $\theta_{t+1} = \sum_i h_t(\mathbf{x}_i, \mathbf{y}_i; \theta_t)$. Moreover, multiple gradient-descent steps also result in a permutation-invariant function, which can be proved by induction. We refer to this as a gradient-based encoder. What follows is that popular meta-learning methods such as MAML can be seen as part of the encoder-decoder formulation. More specifically, in MAML, we learn an initialisation of the model parameters θ_0 from training tasks, and adapt to new tasks by running T gradient steps from the learned initialisation. Therefore, θ_T can be seen as the task representation (albeit very high-dimensional) produced by a gradient-based encoder. The success of MAML on a variety of tasks can be partially explained by the high-dimensional representation and the iterative adaptation by gradient descent, supporting our usage of a functional ('infinite-dimensional') representation and iterative updating procedure. Note, however, that the update rule in MAML operates in parameter space rather than function space as in our case.

Under the same formulation, a comparison regarding MAML and MetaFun can be made, which partially explains why MetaFun can be desirable: Firstly, the updates in MAML must lie in its parametric space, while there is no parametric constraint in MetaFun (better illustrated in Figure 3). Secondly, MAML uses gradient-based updates, while MetaFun uses learned local updates (potentially more flexible). Finally, MAML does not explicitly consider interactions between data points, while both within-context and context-target interactions are modelled in the formulation of MetaFun.

4. Experiments

We evaluate our proposed model on both few-shot regression and classification tasks. In all experiments that follow, we partition the data into training, validation and test meta-sets, each containing data from disjoint tasks. For quantitative results, we train each model with 5 different random seeds and report the mean and the standard deviation of the test accuracy. For further details on hyperparameter tuning, see the Appendix. All experiments are performed using TensorFlow (Abadi et al., 2015), and the code is available online [2].

4.1. 1-D Function Regression

We first explore a 1D sinusoid regression task where we visualise the updating procedure in function space, providing intuition for the learned functional updates. Then we incorporate Gaussian uncertainty into the model, and compare our predictive uncertainty against that of a GP which generates the data.

Table 1. Few-shot regression on sinusoid. MAML can beneift from more parameters, but MetaFun still outperforms all MAMLs despite less parameters being used compared to large MAML. We report mean and standard deviation of 5 independent runs.

Model	5-shot MSE	10-shot MSE
Original MAML	0.390 ± 0.156	0.114 ± 0.010
Large MAML	0.208 ± 0.009	0.061 ± 0.004
Very Wide MAML	0.205 ± 0.013	0.059 ± 0.010
MetaFun	$\mathbf{0.040 \pm 0.008}$	$\mathbf{0.017 \pm 0.005}$

Visualisation of functional updates We train a T-step MetaFun with dot-product functional pooling, on a simple sinusoid regression task from Finn et al. (2017), where each task uses data points of a sine wave. The amplitude A and phase b of the sinusoid varies across tasks and are randomly sampled during training and test time, with $A \in \mathcal{U}(0.1, 5.0)$ and $b \in \mathcal{U}(0, \pi)$. The x-coordinates are uniformly sampled from $\mathcal{U}(-5.0, 5.0)$. Figure 3 shows that our proposed algorithm learns a smooth transition from the initial state to the final prediction at $t = T = 5$. Note that although only 5 context points on a single phase of the sinusoid are given at test time, the final iteration makes predictions close to the ground truth across the whole period. As a comparison, we use MAML as an example of updating in parameter space. The original MAML (40 units $\times$ 2 hidden layers) can fit the sinusoid quite well after several iterations from the learned initialisation. However the prediction is not as good, particularly on the left side where there are no context points (see Figure 3 B). As we increase the model size to large MAML (256 units $\times$ 3 hidden layers), updates become much smoother (Figure 3 C) and the predictions are closer to the ground truth. We further conduct experiments with a very wide MAML (1024 units $\times$ 3 hidden layers), but the performance cannot be further improved (Figure 3 D). In Table 1, we compare the mean squared error averaged across tasks. MetaFun performs much better than all MAMLs, even though less parameters (116611 parameters) are used compared to large MAML (132353 parameters).

[2] Code is available at github.com/jinxu06/metafun-tensorflow

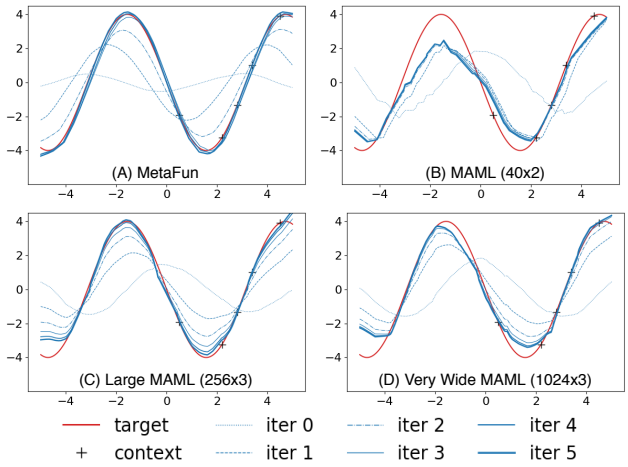

Figure 3. MetaFun is able to learn smooth updates, and recover the ground truth function almost perfectly. While the updates given by MAMLs are relatively not smooth, especially for MAML with less parameters.

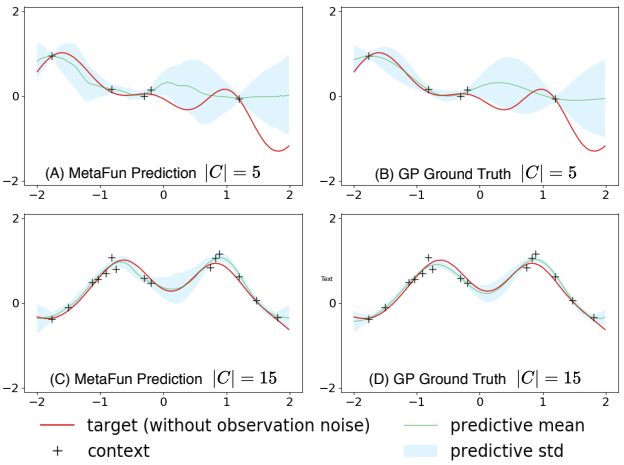

Figure 4. Predictive uncertainties for MetaFun matches those for the oracle GP very closely in both 5-shot and 15-shot cases. The model is trained on random context size ranging from 1 to 20.

Predictive uncertainties As another simple regression example, we demonstrate that MetaFun, like CNP, can produce good predictive uncertainties. We use synthetic data generated using a GP with an RBF kernel and Gaussian observation noise ($\mu = 0, \sigma = 0.1$), and our decoder produces both predictive means and variances. As in Kim et al. (2019), we found that MetaFun-DFP can produce somewhat piece-wise constant mean predictions which is less appealing in this situation. On the other hand, MetaFun-KFP (with deep kernels) performed much better, as can be seen in Figure 4. We consider the cases of 5 or 15 context points, and compare our predictions to those for the oracle GP. In both cases, our model gave very good predictions.

4.2. Classification: miniImageNet and tieredImageNet

The *miniImageNet* dataset (Vinyals et al., 2016) consists of 100 classes selected randomly from the ILSVRC-12 dataset (Russakovsky et al., 2015), and each class contains 600 randomly sampled images. We follow the split in Ravi & Larochelle (2016), where the dataset is divided into training (64 classes), validation (16 classes), and test (20 classes) meta-sets. The *tieredImageNet* dataset (Ren et al., 2018) contains a larger subset of the ILSVRC-12 dataset. These classes are further grouped into 34 higher-level nodes. These nodes are then divided into training (20 nodes), validation (6 nodes), and test (8 nodes) meta-sets. This dataset is considered more challenging because the split is near the root of the ImageNet hierarchy (Ren et al., 2018). For both datasets, we use the pre-trained features provided by Rusu et al. (2019).

Following the commonly used experimental setting, each few-shot classification task consists of 5 randomly sampled classes from a meta-set. Within each class, we have either 1 example (1-shot) or 5 examples (5-shot) as context, and 15 examples as target. For all experiments, hyperparameters are chosen by training on the training meta-set, and comparing target accuracy on the validation meta-set. We conduct randomised hyperparameters search (Bergstra & Bengio, 2012), and the search space is given in Appendix. Then with the model configured by the chosen hyperparameters, we train on the union of the training and validation meta-sets, and report final target accuracy on the test meta-set.

In Table 2 we compare our approach to other meta-learning methods. The numbers presented are the mean and standard deviation of 5 independent runs. The table demonstrates that our model outperforms previous state-of-the-art on 1-shot and 5-shot classification tasks for the more challenging tieredImageNet. As for miniImageNet, we note that previous work, such as MetaOptNet-SVM (Lee et al., 2019b), used significant data augmentation to regularise their model and hence achieved superior results. For a fair comparison, we also equipped each model with data augmentation and reported accuracy with/without data augmentation. However, MetaOptNet-SVM (Lee et al., 2019b) uses a different data augmentation scheme involving horizontal flip, random crop, and color (brightness, contrast, and saturation) jitter. On the other hand, MetaFun, Qiao et al. (2018) and LEO (Rusu et al., 2019), only use image features averaging representation of different crops and their horizontal mirrored versions. In 1-shot cases, MetaFun matches previous state-of-the-art performance, while in 5-shot cases, we get significantly better results. In Table 2, results for both MetaFun-DFP (using dot-product attention) and MetaFun-KFP (using deep kernels) are reported. Although both of

Table 2. Few-shot Classification Test Accuracy

Models	**miniImageNet 5-way 1-shot**	**miniImageNet 5-way 5-shot**
(Without deep residual networks feature extraction):		
Matching networks (Vinyals et al., 2016)	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
Meta-learner LSTM (Ravi & Larochelle, 2016)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML (Finn et al., 2017)	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
LLAMA (Grant et al., 2018)	$49.40 \pm 1.83\%$	-
REPTILE (Nichol et al., 2018)	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$
PLATIPUS (Finn et al., 2018)	$50.13 \pm 1.86\%$	-
(Without data augmentation):		
Meta-SGD (Li et al., 2017)	$54.24 \pm 0.03\%$	$70.86 \pm 0.04\%$
SNAIL (Mishra et al., 2018)	$55.71 \pm 0.99\%$	$68.88 \pm 0.92\%$
Bauer et al. (2017)	$56.30 \pm 0.40\%$	$73.90 \pm 0.30\%$
Munkhdalai et al. (2018)	$57.10 \pm 0.70\%$	$70.04 \pm 0.63\%$
TADAM (Oreshkin et al., 2018)	$58.50 \pm 0.30\%$	$76.70 \pm 0.30\%$
Qiao et al. (2018)	$59.60 \pm 0.41\%$	$73.74 \pm 0.19\%$
LEO	$\mathbf{61.76 \pm 0.08}\%$	$77.59 \pm 0.12\%$
MetaFun-DFP	$\mathbf{62.12 \pm 0.30}\%$	$77.78 \pm 0.12\%$
MetaFun-KFP	$61.16 \pm 0.15\%$	$\mathbf{78.20 \pm 0.16}\%$
(With data augmentation):		
Qiao et al. (2018)	$\mathbf{63.62 \pm 0.58}\%$	$78.83 \pm 0.36\%$
LEO	$63.97 \pm 0.20\%$	$79.49 \pm 0.70\%$
MetaOptNet-SVM (Lee et al., 2019b)[1]	$\mathbf{64.09 \pm 0.62}\%$	$80.00 \pm 0.45\%$
MetaFun-DFP	$\mathbf{64.13 \pm 0.13}\%$	$\mathbf{80.82 \pm 0.17}\%$
MetaFun-KFP	$63.39 \pm 0.15\%$	$\mathbf{80.81 \pm 0.10}\%$

Models	**tieredImageNet 5-way 1-shot**	**tieredImageNet 5-way 5-shot**
(Without deep residual networks feature extraction):		
MAML (Finn et al., 2017)	$51.67 \pm 1.81\%$	$70.30 \pm 0.08\%$
Prototypical Nets (Snell et al., 2017)	$53.31 \pm 0.89\%$	$72.69 \pm 0.74\%$
Relation Net [in Liu et al. (2019)]	$54.48 \pm 0.93\%$	$71.32 \pm 0.78\%$
Transductive Prop. Nets (Liu et al., 2019)	$57.41 \pm 0.94\%$	$71.55 \pm 0.74\%$
(With deep residual networks feature extraction):		
Meta-SGD	$62.95 \pm 0.03\%$	$79.34 \pm 0.06\%$
LEO	$66.33 \pm 0.05\%$	$81.44 \pm 0.09\%$
MetaOptNet-SVM	$65.81 \pm 0.74\%$	$81.75 \pm 0.58\%$
MetaFun-DFP	$\mathbf{67.72 \pm 0.14}\%$	$82.81 \pm 0.15\%$
MetaFun-KFP	$67.27 \pm 0.20\%$	$\mathbf{83.28 \pm 0.12}\%$

them demonstrate state-of-the-art performance, MetaFun-KFP generally outperforms MetaFun-DFP for 5-shot problems, but performs slightly worse for 1-shot problems.

4.3. Ablation Study

As stated in Section 2.2, our model has three learnable components: the local update function, the functional pooling, and the decoder. In this section we explore the effects of using different versions of these components. We also investigate how the model performance would change with different numbers of iterations.

Table 3 demonstrates that neural network parameterised local update functions, described in Section 2.1, consistently outperforms gradient-based local update function, despite the latter having build-in inductive biases. Interestingly, the choice between dot-product attention and deep kernel in functional pooling is problem dependent. We found that MetaFun with deep kernels usually perform better than MetaFun with dot product attention on 5-shot classification tasks, but worse on 1-shot tasks. We conjecture that the deep kernel is better able to fuse the information across the 5 images per class compared to attention. In the comparative experiments in Section 4.2 we reported results on both.

In addition, we investigate how a simple Squared Exponential (SE) kernel would perform on these few-shot classification tasks. This corresponds to using an identity input transformation function a in deep kernels. Table 3 shows that using SE kernel is consistently worse than using deep kernels, showing that the heavily parameterised deep kernel is necessary for these problems.

Table 3. Ablation Study. We conduct independent randomised hyperparameter search for each number presented, and reported means and standard deviations over 5 independent runs for each.

Functional pooling	Local update function	Decoder	MiniImageNet 1-shot	MiniImageNet 5-shot	tieredImageNet	
Attention	NN	✓	$\mathbf{62.12 \pm 0.30}\%$	$77.78 \pm 0.12\%$	$\mathbf{67.72 \pm 0.14}\%$	$82.81 \pm 0.15\%$
Deep Kernel	NN	✓	$61.16 \pm 0.15\%$	$\mathbf{78.20 \pm 0.16}\%$	$67.27 \pm 0.20\%$	$\mathbf{83.28 \pm 0.12}\%$
Attention	Gradient	✓	$59.63 \pm 0.19\%$	$75.84 \pm 0.04\%$	$62.55 \pm 0.10\%$	$78.18 \pm 0.09\%$
Deep Kernel	Gradient	✓	$59.73 \pm 0.21\%$	$76.41 \pm 0.14\%$	$65.24 \pm 0.11\%$	$80.31 \pm 0.16\%$
SE Kernel	NN	✓	$60.04 \pm 0.19\%$	$75.25 \pm 0.12\%$	$60.81 \pm 0.30\%$	$79.70 \pm 0.20\%$
Deep Kernel	Gradient	✗	$57.67 \pm 0.16\%$	$73.55 \pm 0.04\%$	$62.53 \pm 0.17\%$	$76.86 \pm 0.07\%$

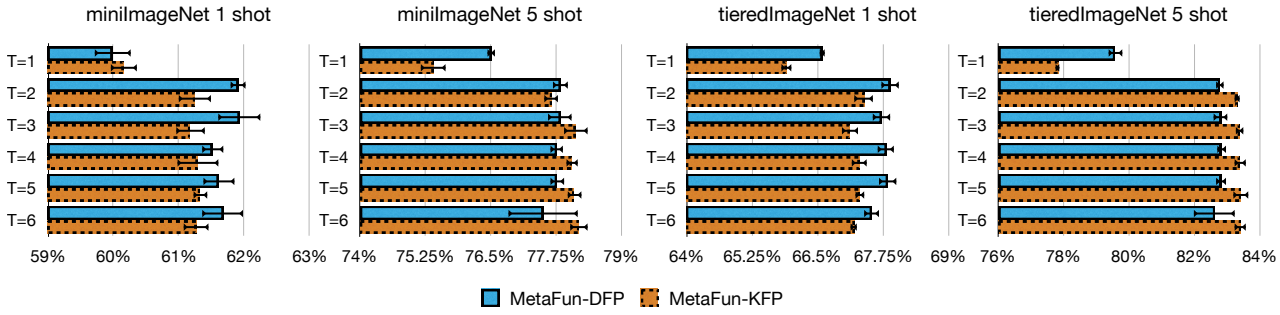

Figure 5. This figure illustrates the accuracy of our approach for varying number of iterations $T = 1, \ldots, 6$, over different few-shot learning problems. For each problem, we use the same configuration of hyperparameters except for the number of iterations and the choice between attention and deep kernels. Error bars (standard deviations) are given by training the same model 5 times with different random seeds.

Next, we looked into directly applying functional gradient descent with parameterised deep kernel to these tasks. This corresponds to removing the decoder and using deep kernels and gradient-based local update function (see Section 3). Unsurprisingly, this did not fare as well, given as it only has one trainable component (the deep kernel) and the updates are directly applied to the predictions rather than a latent functional representation.

Finally, Figure 5 illustrates the effects of using different numbers of iterations T. On all few-shot classification tasks, we can see that using multiple iterations (two is often good enough) always significantly outperform one iteration. We also note that this performance gain diminishes as we add more iterations. In Section 4.2 we treated the number of iterations as one of the hyperparameters.

5. Conclusions and Future Work

In this paper, we propose a novel functional approach for meta-learning called MetaFun. The proposed approach learns to generate a functional task representation and an associated functional update rule, which allows to iteratively update the task representation directly in the function space. We evaluate MetaFun on both few-shot regression and classification tasks, and demonstrate that it matches or exceeds previous state-of-the-art results on miniImageNet and tieredImageNet few-shot classification tasks.

Interesting future research directions include a) exploring a stochastic encoder and hence working with stochastic functional representations, akin to the Neural Process (NP), and b) using local update functions and the functional pooling components whose parameters change with iterations instead of sharing them across iterations, where the added flexibility could lead to further performance gains.

Acknowledgements

We would like to thank Jonathan Schwarz for valuable discussion, and the anonymous reviewers for their feedback. Jin Xu and Yee Whye Teh acknowledge funding from Tencent AI Lab through the Oxford-Tencent Collaboration on Large Scale Machine Learning project. Jean-Francois Ton is supported by the EPSRC and MRC through the OxWaSP CDT programme (EP/L016710/1).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pp. 3981–3989, 2016.

Aronszajn, N. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

Bauer, M., Rojas-Carulla, M., Świątkowski, J. B., Schölkopf, B., and Turner, R. E. Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*, 2017.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Berlinet, A. and Thomas-Agnan, C. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

Bloem-Reddy, B. and Teh, Y. W. Probabilistic symmetries and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61, 2020. URL http://jmlr.org/papers/v21/19-322.html.

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

Finn, C., Xu, K., and Levine, S. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pp. 9516–9527, 2018.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. Conditional neural processes. In *International Conference on Machine Learning*, pp. 1690–1699, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

Koch, G. Siamese neural networks for one-shot image recognition. *Master's thesis, University of Toronto*, 2015.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753, 2019a.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10657–10665, 2019b.

Li, Z., Zhou, F., Chen, F., and Li, H. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Liu, Y., Lee, J., Park, M., Kim, S., Yang, E., Hwang, S. J., and Yang, Y. Learning to propagate labels: Transductive propagation network for few-shot learning. In *International Conference on Learning Representations*, 2019.

Marino, J., Yue, Y., and Mandt, S. Iterative amortized inference. In *International Conference on Machine Learning*, pp. 3403–3412, 2018.

Mason, L., Baxter, J., Bartlett, P. L., Frean, M., et al. Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers. MIT Press*, 1999.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Munkhdalai, T., Yuan, X., Mehri, S., and Trischler, A. Rapid adaptation with conditionally shifted neurons. In *International Conference on Machine Learning*, pp. 3661–3670, 2018.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

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

Oreshkin, B., López, P. R., and Lacoste, A. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 721–731, 2018.

Qiao, S., Liu, C., Shen, W., and Yuille, A. L. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7229–7238, 2018.

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2016.

Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*, 2018.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850, 2016.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1199–1208, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Wagstaff, E., Fuchs, F. B., Engelcke, M., Posner, I., and Osborne, M. On the limitations of representing functions on sets. *arXiv preprint arXiv:1901.09006*, 2019.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial Intelligence and Statistics*, pp. 370–378, 2016.

Y. Guo, P. Bartlett, A. S. and Williamson, R. C. Norm-based regularization of boosting. *Submitted to Journal of Machine Learning Research*, 2001.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.