MetaFun: Meta-Learning with Iterative Functional Updates

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

Department of Statistics, University of Oxford[1]
Applied AI Lab, Oxford Robotics Institute, University of Oxford[2]
DeepMind[3]

Abstract

Few-shot supervised learning leverages experience from previous learning tasks to solve new tasks where only a few labelled examples are available. One successful line of approach to this problem is to use an encoder-decoder meta-learning pipeline, whereby labelled data in a task is encoded to produce task representation, and this representation is used to condition the decoder to make predictions on unlabelled data. We propose an approach that uses this pipeline with two important features. 1) We use infinite-dimensional functional representations of the task rather than fixed-dimensional representations. 2) We iteratively apply functional updates to the representation. We show that our approach can be interpreted as extending functional gradient descent, and delivers performance that is comparable to or outperforms previous state-of-the-art on few-shot classification benchmarks such as miniImageNet and tieredImageNet.

1 Introduction

Humans have a remarkable ability to generalise to new tasks and use past experiences to solve new problems quickly. Traditional machine learning algorithms struggle to do so. In recent years, significant effort has been devoted into addressing these issues under the field of meta-learning, whose goal is to be able to generalise to new tasks from the same task distribution as the training tasks. In supervised 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 meta-learning from different perspectives. Andrychowicz et al. (2016); Ravi and Larochelle (2016) propose to learn the optimisation algorithm from previous tasks which can be used for new tasks. Santoro et al. (2016) demonstrates that Memory-Augmented Neural Networks (MANN) can rapidly integrate new data into memory, and utilise this stored information to make predictions while only seeing a few examples of the new task. Model Agnostic Meta Learning (MAML) (Finn et al., 2017) learns an initialisation of the model parameters, and adapts to a new task by further running a few gradient steps. Koch (2015); Snell et al. (2017); Vinyals et al. (2016) explore the idea of learning a metric space from previous tasks in which new data points are compared to each other to make predictions at test time.

In this work, we are particularly interested in another family of meta-learning models that use an encoder-decoder pipeline (Garnelo et al., 2018a,b; Rusu et al., 2019). The encoder is a permutation-invariant function on the context set that summarises the task into a task representation, while the decoder is a predictive model that makes predictions on the target, conditioned on the task representation. The objective of meta-learning is then to learn the encoder and the decoder such that the predictive models generalise well to new tasks.

Previous works such as Latent Embedding Optimisation (LEO) (Rusu et al., 2019), Conditional Neural Process (CNP) and Neural Process (NP) (Garnelo et al., 2018a,b), all belong to this category. Despite their success on various tasks, NPs tend to underfit the context. Attentive Neural Process (ANP) (Kim et al., 2019) addresses this issue by modifying the encoder to produce summaries of the task using a target-specific representation, which allows each target to attend to context points more relevant to it. We show in Section 3.2 that this can be interpreted as representing the task

Preprint. Under review. Correspondence to: Jin Xu <jin.xu@stats.ox.ac.uk>

using a function of the target inputs. Moreover, different from previous perspectives, MAML (Finn et al., 2017), which meta-learns an initialisation and runs a few gradient steps on the context set of a new task starting from the initialisation during test time, can be reinterpreted under the encoder-decoder formulation in Section 2.1, with the very high-dimensinal model parameters seen as task representation. Suggested by the above, meta-learning models may benefit from having a very high-dimensional (like MAML) or even infinite dimensional (like ANP) task representation.

Generally speaking, designing an iterative update rule is often easier than finding the final solution: for example, it is not possible to derive closed-form solutions to most non-convex optimisation problems, but many iterative algorithms can be designed to effectively reach the optima; it can be challenging to sample directly from a high-dimensional target posterior distribution, but we can design a transition kernel for Markov chain Monte Carlo (MCMC) whose equilibrium distribution is the target distribution we are trying to sample from. In meta-learning, both learning to optimise (Andrychowicz et al., 2016; Ravi and Larochelle, 2016) and MAML can be seen as applying iterative updating procedures where the updating rule in MAML is given by the gradient.

In this work, we investigate more deeply the idea of summarising tasks using functional representation. Specifically, we focus on developing a model that learns to iteratively update task representations in the function space. Recently, Gordon et al. (2019) also considers using functional representations in CNP. However, they mainly focus on incorporating translation equivariance in the data as inductive bias. The primary contribution of this work is a meta-learning model that summarises the task into a functional representation, and iteratively applies functional updates based on the context set and current state of the functional representation. we apply our models to solve meta-learning problems on both regression and classification tasks, and achieve performance that is comparable to or outperforms previous state-of-the-art on heavily benchmarked datasets such as miniImageNet (Vinyals et al., 2016) and tieredImageNet (Ren et al., 2018). Moreover, we draw close connection to gradient-based meta-learning methods such as MAML under a unified perspective that can include many previous works. Furthermore, we show that our model is an extension of a classical notion called *functional gradient descent*. From this perspective, our model can also be seen as a learned optimiser operating in function space. Finally, we conduct ablation study to understand the effects of different components in our model.

2 Meta-Learning under the Encoder-Decoder Formulation

Meta-learning, or learning to learn, leverages past experiences in order to quickly adapt to new tasks $\mathcal{T} \sim p(\mathcal{T})$ from the same task distribution. In supervised meta-learning, a task $\mathcal{T}$ takes the form of $\mathcal{T} = \{\ell, \{\mathbf{x}_i, \mathbf{y}_i\}_{i\in C}, \{\mathbf{x}'_j, \mathbf{y}'_j\}_{j\in T}\}$, where ℓ is the loss function to be minimised, $\{\mathbf{x}_i, \mathbf{y}_i\}_{i\in C}$ is the context, and $\{\mathbf{x}'_j, \mathbf{y}'_j\}_{j\in T}$ is the target. A meta-learner adapts to a new task by inferring the parameters of a predictive model f from the context of the task, and the objective of meta-learning is to build a learning model $f = \Phi(\{\mathbf{x}_i, \mathbf{y}_i\}_{i\in C}; \phi)$ with parameters ϕ such that the total loss on the target under f is minimised:

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

2.1 Permutation-Invariant Representation

Many previous meta-learning models, e.g., CNP, NP, ANP as well as MAML and its modifications encode the context into a task representation using a permutation-invariant function. The task representation is then used to obtain a predictive model via a decoding step. Under this framework, the meta-learner consists of an encoder and a decoder, meta-learning corresponds to training the encoder-decoder pipeline, while learning is just a single forward pass through the encoder and the decoder. Formally, we construct the learning model as

$$\begin{aligned} f &= \Phi(\{\mathbf{x}_i, \mathbf{y}_i\}_{i\in C}; \phi) \\ &= \underbrace{\Phi_d(\mathbf{r}; \phi_d)}_{\textbf{Decoder}}, \; \mathbf{r} = \underbrace{\Phi_e(\{\mathbf{x}_i, \mathbf{y}_i\}_{i\in C}; \phi_e)}_{\textbf{Encoder}}, \end{aligned} \quad (2)$$

where $\mathbf{r}$ is the task representation.

The encoder in CNP corresponds to a summation of instance-level representation produced by a shared instance encoder h:

$$\mathbf{r} = \Phi_e(\{\mathbf{x}_i, \mathbf{y}_i\}_{i\in C}; \phi_e) = \frac{1}{|C|} \sum_{i\in C} h(\mathbf{x}_i, \mathbf{y}_i; \phi_e)\,. \quad (3)$$

NPs, on the other hand, use a probabilistic encoder with the same parametric form as in Equation (3), but producing a distribution of stochastic representation $\mathbf{r}$. Note that to represent permutation-invariant functions, a summation after shared instance-wise encoders is a generic form (Zaheer et al., 2017; Bloem-Reddy and Teh, 2019).

Interestingly, many gradient-based meta-learning methods can also be cast into this formulation, because a gradient descent step is actually a valid permutation-invariant function. To be specific, for a model $f(\cdot, \mathbf{r})$ parameterised by $\mathbf{r}$, one step of gradient descent on the context set with loss function ℓ and learning rate α has the following form, where $\mathbf{r}_0$ is the initialisation,

$$\mathbf{r} = \mathbf{r}_0 - \frac{\alpha}{|C|} \sum_{i \in C} \nabla_{\mathbf{r}} \ell(f(\mathbf{x}_i; \mathbf{r}_0), \mathbf{y}_i) \,. \tag{4}$$

This corresponds to the special case where we take the instance-wise encoder to be $h(\mathbf{x}_i, \mathbf{y}_i) = \mathbf{r}_0 - \alpha \nabla_{\mathbf{r}} \ell(f(\mathbf{x}_i; \mathbf{r}_0), \mathbf{y}_i)$. Moreover, multiple gradient-descent steps also result in a permutation-invariant function[1]. We refer to this as a gradient-based encoder.

What follows is that popular meta-learning methods such as MAML and LEO (Rusu et al., 2019) can be seen as part of this framework. More specifically, in MAML, $\mathbf{r}_0$ is the initialisation of the model parameters, and $\mathbf{r}$ becomes the task representation (albeit very high-dimensional). LEO composes a generic NP encoder, relation networks (Sung et al., 2018; Raposo et al., 2017)) and a gradient-based encoder, and therefore also falls into our formulation.

We see that many meta-learning methods use a permutation-invariant function to infer model parameters from the context, and that the differences between these methods come down to the choice of the permutation-invariant function (the encoder), and the dimensionality of the representation produced by the encoder. The two main function classes used in these models are the generic, more flexible neural-network-based encoder, and a gradient-based one, which is a special case of the former. The success of MAML can be partially explained by the observation that using gradient-based encoder constitutes a strong inductive bias for learning, which is absent in vanilla neural networks. LEO does rely on a more flexible generic encoder, but it combines it with gradient-based updates, therefore enjoying the best of both worlds.

With regard to the dimensionality of the representation, it is shown in Wagstaff et al. (2019) that if the dimensionality of the representation is smaller than the number of context points, there exists a permutation-invariant function that cannot be expressed in the form of Equation (3). Kim et al. (2019) further show that a finite-dimensional context representation can be quite limiting in its expressiveness, often resulting in underfitting for regression tasks. MAML circumvents this issue by using model parameters as a very high-dimensional task representation. While the approach we will propose uses a functional task representation which can be seen as having infinite dimensions.

3 Meta-Learning in Function Space

In this section we will consider an approach to meta-learning using functions to represent and summarise task-specific context sets. Taking an infinite-dimensional functional approach allows us to bypass issues of expressiveness associated with finite dimensional task representations. We motivate our approach by starting with a a high level description of classical functional gradient descent (Mason et al., 1999; Y. Guo and Williamson, 2001). We then show how we can replace each component with more flexible and learnable neural modules, and finally specialise our approach for both few-shot regression and classification tasks.

3.1 Functional Gradient Descent

For a supervised learning task $\mathcal{T}$, with context set $\{\mathbf{x}_i, \mathbf{y}_i\}_{i \in C}$, the central object of interest is to learn the prediction function $f : \mathcal{X} \mapsto \mathcal{Y}$. The idea of functional gradient descent is to learn f by directly computing its gradient and updating it in function space. To ensure that our functions are regularised to be smooth, we work with functions in a Reproducing kernel Hilbert space (RKHS) (Aronszajn, 1950; Berlinet and Thomas-Agnan, 2011) defined by a kernel $k(\mathbf{x}, \mathbf{x}')$. For our purposes it is sufficient to think of $k(\mathbf{x}, \mathbf{x}')$ as defining a measure of similarity between two points $\mathbf{x}$ and $\mathbf{x}'$ in the input space $\mathcal{X}$.

Given a function f in the RKHS, we are interested in minimising the supervised loss $L(f) = \frac{1}{|C|} \sum_{i \in C} \ell(f(\mathbf{x}_i), \mathbf{y}_i)$ with respect to f. We can do so by computing the functional derivative. This is itself a function of $\mathbf{x}$, and its evaluation at input point $\mathbf{x}$ can be shown to be (Mason et al., 1999; Y. Guo and Williamson, 2001) (see Appendix A.3 for more details):

$$\begin{aligned} \nabla_f L(f(\mathbf{x})) &= \nabla_f \left(\frac{1}{|C|} \sum_{i \in C} \ell(f(\mathbf{x}_i), \mathbf{y}_i) \right) (\mathbf{x}) \\ &= \frac{1}{|C|} \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \ell'(f(\mathbf{x}_i), \mathbf{y}_i) \end{aligned} \tag{5}$$

where ℓ' is the partial derivative of the loss ℓ with respect to its first argument. We can interpret Equation (5) as follows: the derivative of f at $\mathbf{x}$ is a linear combination of the derivatives at training points (context), $\ell'(f(\mathbf{x}_i), \mathbf{y}_i)$, weighted by the similarities $k(\mathbf{x}, \mathbf{x}_i)$.

We can use this functional derivative to directly update f iteratively:

$$f^{(t+1)}(\mathbf{x}) = f^{(t)}(\mathbf{x}) - \frac{\alpha}{|C|} \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i) \tag{6}$$

[1] A composition of permutation-invariant functions is permutation-invariant.

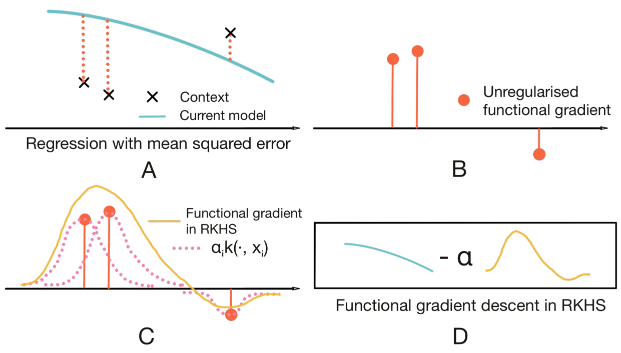
Context
Current model
Regression with mean squared error
A
Unregularised functional gradient
B
Functional gradient in RKHS
$\alpha_i k(\cdot, x_i)$
C
- α
Functional gradient descent in RKHS
D

Figure 1: (A) $1D$ regression task with Mean Squared Error (MSE) loss. (B) Unregularised functional gradient (strictly, subgradient). For MSE loss, $\ell'(f(\mathbf{x}_i), \mathbf{y}_i) = f(\mathbf{x}_i) - \mathbf{y}_i$ (see Appendix A.3). Therefore unregularised functional gradient is proportional to the difference between predictions and labels at the context, and undefined otherwise. However, updating using this unregularised functional gradient would lead to extreme overfitting because it does not generalise outside the context. (C) We consider functional gradient in a smoothed RKHS. (D) Functional gradient descent in RKHS.

with step size α. To gain more intuition, we illustrate running functional gradient descent on a simple 1D regression task in Figure 1. Obviously one cannot compute Equation (6) at all inputs $\mathbf{x} \in \mathcal{X}$. However it turns out that it is sufficient to compute it only on the context, since the function values outside the context does not affect the next functional update, hence does not affect the final model $f^T(\mathbf{x})$ after T iteration (see Equation (28) in Appendix A.3).

3.2 MetaFun

The updates above have no tunable parameters, except for the step size and kernel. In this section, we will develop MetaFun, a meta-learning framework with architecture inspired by the functional gradient descent updates above. Specifically, we can make the above procedure more flexible, by replacing each component with a neural network module:

1. We can use a latent functional representation $r(\mathbf{x})$ which is decoded into a prediction function $f(\mathbf{x})$.
2. We can replace the derivatives at context points, $\ell'(f(\mathbf{x}_i), \mathbf{y}_i)$, with a learned neural network $\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i))$.
3. We can use a deep kernel (Wilson et al., 2016) parameterised by a neural network to learn more complex similarity relationships among input points.
4. We can replace the kernel altogether with attention (Vaswani et al., 2017; Kim et al., 2019).

Using meta-learning, we can train the various modules to generalise well from context sets to target sets. Alternatively, we can think of our method as learning an optimiser which operates in function space to generalise well. In the rest of this Section we will elaborate on each of the modifications above.

As in Section 3.1, it is unnecessary to compute the functional representations $r(\mathbf{x})$ (or their functional updates) on all input points. Instead we will compute them only on the context points $\{\mathbf{x}_i, \mathbf{y}_i\}_{i \in C}$ and target points $\{\mathbf{x}'_j\}_{j \in T}$. We use $\mathbf{r} = [r(\mathbf{x}_1), \ldots, r(\mathbf{x}_{|C|}), r(\mathbf{x}'_1), \ldots, r(\mathbf{x}'_{|T|})]^\top$ to denote a matrix where each row is $r(\mathbf{x})$ evaluated on either context or target inputs. Using a deep kernel parameterised by a neural network input transformation $a(\mathbf{x})$, and replacing $\ell'(f(\mathbf{x}_i), \mathbf{y}_i)$ with another neural network $\mathbf{u}_i = u(\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i))$ which we call local update function, the kernel-based functional gradient of Equation (5) can be expressed as:

$$\textsc{kg}\,(Q, K, U) := k(Q, K)U \tag{7}$$

where $Q = [a(\mathbf{x}_1), \ldots, a(\mathbf{x}_{|C|}), a(\mathbf{x}'_1), \ldots, a(\mathbf{x}'_{|T|})]^\top$ is a matrix with rows being queries (consisting of both contexts and targets), $K = [a(\mathbf{x}_1), \ldots, a(\mathbf{x}_{|C|})]^\top$ is a matrix of keys, and $U = [\mathbf{u}_1, \ldots, \mathbf{u}_{|C|}]^\top$ a matrix of values (using terminology from the attention literature). The kernel $k(Q, K)$ computes the matrix of kernel/similarity values among the query and key points. A dot-product attention (Vaswani et al., 2017) can alternatively be used in place of a kernel, and now dot-product serves as a similarity metric:

$$\textsc{dp}\,(Q, K, U) := \mathrm{softmax}(QK^\top/\sqrt{d_k})U, \tag{8}$$

where d_k is the dimension of the query/key vectors. However, the similarity values in attention are normalised due to softmax. Note that Equation (8) is the same as the query-specific cross-attention module in the deterministic path of attentive neural processes (ANP) (Kim et al., 2019), with u serving as instance encoders. It is also possible to use other forms of attention, such as Laplace attention or multihead attention (Kim et al., 2019). We note that attention mechanisms need not correspond to kernels as they need not be symmetric or positive semi-definite.

Having defined functional representations of the updates, we can now write down a procedure to iteratively compute the functional representation of the task. We initialise the representation at $\mathbf{r}^{(0)}$, and update the

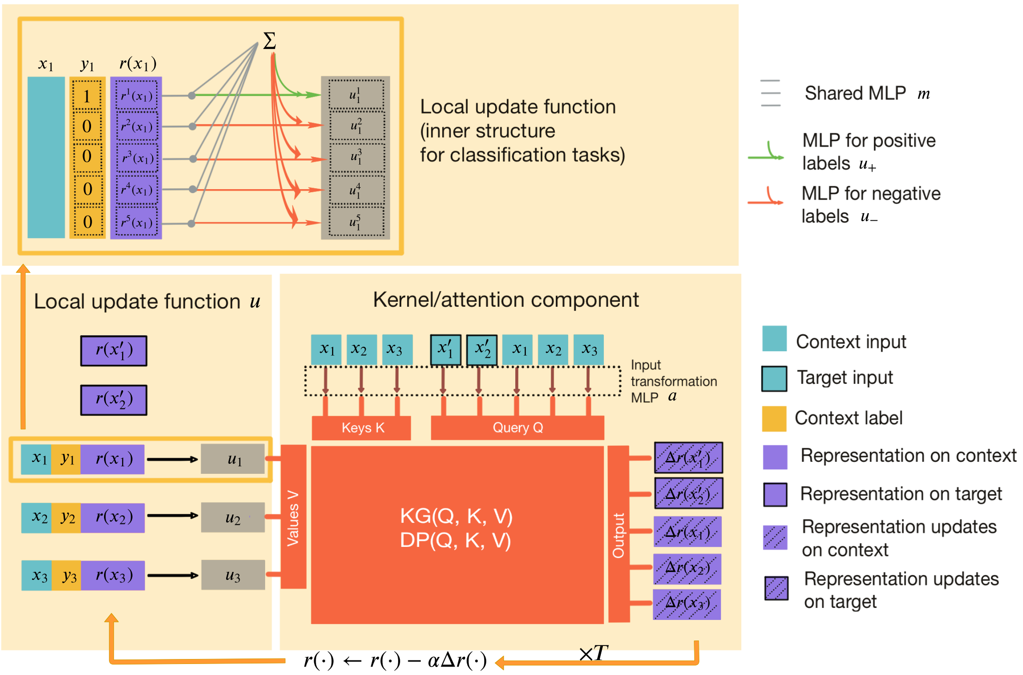
x_1
y_1
$r(x_1)$
1
0
0
0
0
$r^1(x_1)$
$r^2(x_1)$
$r^3(x_1)$
$r^4(x_1)$
$r^5(x_1)$
Σ
u_1^1
u_1^2
u_1^3
u_1^4
u_1^5
Local update function
(inner structure
for classification tasks)
Shared MLP m
MLP for positive labels u_+
MLP for negative labels u_-
Local update function u
$r(x'_1)$
$r(x'_2)$
x_1 y_1 $r(x_1)$ u_1
x_2 y_2 $r(x_2)$ u_2
x_3 y_3 $r(x_3)$ u_3
Kernel/attention component
x_1 x_2 x_3 x'_1 x'_2 x_1 x_2 x_3
Input transformation MLP a
Keys K
Query Q
Values V
KG(Q, K, V)
DP(Q, K, V)
Output
$\Delta r(x'_1)$
$\Delta r(x'_2)$
$\Delta r(x_1)$
$\Delta r(x_2)$
$\Delta r(x_3)$
$r(\cdot) \leftarrow r(\cdot) - \alpha\Delta r(\cdot)$
$\times T$
Context input
Target input
Context label
Representation on context
Representation on target
Representation updates on context
Representation updates on target

Figure 2: Updating functional representation in MetaFun. This figure illustrates one iteration of MetaFun. For classification problems, the local update function has special inner structures, which is further illustrated on top left.

representation a fixed number of times using:

$$\mathbf{u}_i^{(t)} = u(\mathbf{x}_i, \mathbf{y}_i, \mathbf{r}^{(t)}(\mathbf{x}_i)) \tag{9}$$

$$U^{(t)} = [\mathbf{u}_1^{(t)}, \dots, \mathbf{u}_{|C|}^{(t)}]^\top \tag{10}$$

$$\Delta\mathbf{r}^{(t)} = \textsc{kg} \ \text{ or } \ \textsc{dp}\left(Q, K, U^{(t)}\right) \tag{11}$$

$$\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} - \alpha\Delta\mathbf{r}^{(t)} \tag{12}$$

where α is the learning rate, and the final representation after T steps is $r(\mathbf{x}) = r^{(T)}(\mathbf{x})$. Note that the local update function u and the kernel/attention component is shared across iterations. This iterative procedure is illustrated in Figure 2. In practice, the performance of our method is not sensitive to the learning rate. This is expected as the function $u(\mathbf{x})$ is learned, and the output scale of u can account for different values of the learning rate. Furthermore, we found that zero-initialised $r^{(0)}(\mathbf{x}) = 0$ works reasonably well empirically, even though we also consider a constant-initialised $r^{(0)}(\mathbf{x}) = c$ and a parametric variant $r^{(0)}(\mathbf{x}) = r_{\theta_0}^{\text{init}}(\mathbf{x})$ during hyperparameter tuning.

Our approach mainly consists of three learnable components in total: the local update function, the kernel/attention component, and the decoder. For a new task, we simply run T iterations of our functional updates (eqs. (9) to (12)) to get a task representation $\mathbf{r}^{(T)}$, where each iteration shares the same local update function followed by the same kernel/attention component. The final representation is then used to condition the decoder $\Phi_d(\mathbf{r}(\mathbf{x}); \phi_d)$, which is parameterised as an multi-layer perceptron (MLP) or a linear transformation, to make predictions for the new task. During meta-training, we minimise the following objective:

$$L(\phi) = \left[\sum_{j \in T} \ell(f(\mathbf{x}'_j), \mathbf{y}'_j)\right] \tag{13}$$

$$\text{where } \ f = \Phi_d(\mathbf{r}^{(T)}(\mathbf{x}); \phi_d),$$

and $\mathbf{r}^{(T)}$ is given by Equations (9) to (12).

3.3 MetaFun for Regression and Classification

While the proposed framework can be applied to any supervised learning task, the specific parameterisation of learnable components does 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 an MLP, which takes as input the following concatenation $[\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]$ i.e $u([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = \textsc{mlp}\left([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]\right), i \in C$ and outputs functional updates. The input transformation function $a(\cdot)$ in the kernel/attention component is parameterised by another MLP in experiments, even though it is possible to use other architectures in general. The decoder in this case can be given by an MLP such that $\mathbf{w} = \text{MLP}\,(r(\mathbf{x}))$, and $\mathbf{w}$ is used to parameterise the predictive model $f = \text{MLP}\,(\mathbf{x}; \mathbf{w})$ which is also an MLP. It is also possible to use other types of decoder such as simply using an MLP taking the concatenation of $r(\mathbf{x})$ and $\mathbf{x}$ as inputs: $f(\mathbf{x}) = \text{MLP}\,([\mathbf{x}, r(\mathbf{x})]))$, or feeding $r(\mathbf{x})$ to each layer of the MLP. Note, our model can easily be modified to incorporate gaussian uncertainty by adding an extra output vector for the predictive standard deviation such as for example $P(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mu_{\mathbf{w}}(\mathbf{x}), \sigma_{\mathbf{w}}(\mathbf{x})), \mathbf{w} = \text{MLP}\,(r(\mathbf{x}))$. For architecture details about these MLPs, see Table 6.

Classification For a K-way classification task, the latent functional representation $r(\mathbf{x})$ is divided into K parts $[r^1(\mathbf{x}), ..., r^K(\mathbf{x})]$, where $r^k(\mathbf{x})$ corresponds to class k. Consequently, the local update function $u(\cdot)$ will also have K parts, i.e. $u([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = [u^1([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]), ..., u^K([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)])]$. In this case, $\mathbf{y}_i = [y_i^1, ..., y_i^K]$ corresponds to a one-hot vector describing the class label and u^k is defined as follows,

$$\begin{aligned} u^k([\mathbf{x}_i, \mathbf{y}_i, r(\mathbf{x}_i)]) = y_i^k u_+(m(r^k(\mathbf{x}_i)), \boldsymbol{m}_i) \\ + (1 - y_i^k) u_-(m(r^k(\mathbf{x}_i)), \boldsymbol{m}_i) \end{aligned} \tag{14}$$

where $\boldsymbol{m}_i = \sum_{k=1}^K m(r^k(\mathbf{x}_i))$ summarises representations of all classes, and where m, u_+, u_- are parameterised by separate MLPs. This formulation allows updating class representations using either u_+ (when the label matches k) or u_- (where the label is different than k), and in practice parameterising local update function in this way is critical for classification tasks. This design of local update function is also illustrated in Figure 2. In fact, it is partly motivated by the updating procedure of functional gradient descent for classification tasks, which we derive in Appendix A.3. Same as regression tasks, the input transformation function a in the kernel/attention component is still an MLP. The parametric form of the decoder is the same as LEO (Rusu et al., 2019). The class representation $r^k(\mathbf{x})$ generates softmax weights $\mathbf{w^k} \sim \mathcal{N}(\mu(r^k(\mathbf{x})), \sigma(r^k(\mathbf{x})))$ through an MLP or just a linear function $[\mu(r^k(\mathbf{x})), \sigma(r^k(\mathbf{x}))] = g_w(r^k(\mathbf{x}))$, and the final prediction is given by

$$P(\mathbf{y} = k|\mathbf{x}) = \text{softmax}(\mathbf{x}^T\mathbf{w})_k \tag{15}$$

where $\mathbf{w} = [\mathbf{w^1}, ..., \mathbf{w^K}], k = 1, ..., K$. Hyperparameters of all components can be found in Appendix B.

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 Appendix B.

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	$\mathbf{0.040 \pm 0.008}$	$\mathbf{0.017 \pm 0.005}$

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.

Visualisation of functional updates We train a T-step MetaFun with dot-product attention, 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 × 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 × 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 × 3 hidden layers), but the performance cannot be further improved (Figure 3 D). In Table 1, we compare the mean squared error averaged across

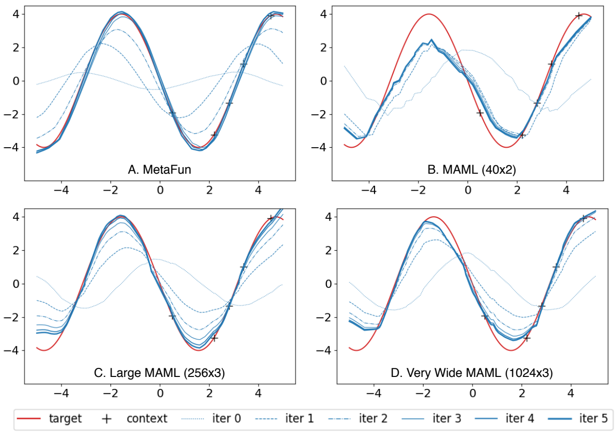

Figure 3: MetaFun is able to learn smooth updates, and recover the ground truth function almost perfectly. While the updates given by MAMLs are relatively not smooth, especially for MAML with less parameters.

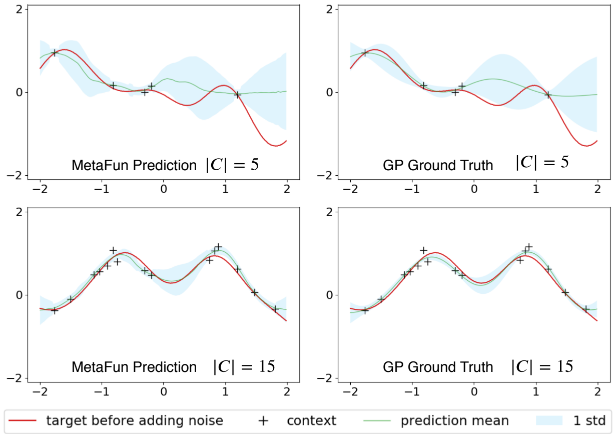

Figure 4: Predictive uncertainties for MetaFun matches those for the oracle Gaussian Process (GP) very closely in both 5-shot and 15-shot cases. The model is trained on random context size ranging from 1 to 20.

tasks. MetaFun performs much better than all MAMLs, even though less parameters (116611 parameters) are used compared to large MAML (132353 parameters).

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-Attention can produce somewhat piece-wise constant mean predictions which is less appealing in this situation. On the other hand, MetaFun-Kernel (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 and 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 and Bengio, 2012), and the search space is given in Table 4. 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., 2019), 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., 2019) 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.,

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 ± 0.84%	55.31 ± 0.73%
Meta-learner LSTM (Ravi and Larochelle, 2016)	43.44 ± 0.77%	60.60 ± 0.71%
MAML (Finn et al., 2017)	48.70 ± 1.84%	63.11 ± 0.92%
LLAMA (Grant et al., 2018)	49.40 ± 1.83%	-
REPTILE (Nichol et al., 2018)	49.97 ± 0.32%	65.99 ± 0.58%
PLATIPUS (Finn et al., 2018)	50.13 ± 1.86%	-
(Without data augmentation):		
Meta-SGD (Li et al., 2017)	54.24 ± 0.03%	70.86 ± 0.04%
SNAIL (Mishra et al., 2018)	55.71 ± 0.99%	68.88 ± 0.92%
Bauer et al. (2017)	56.30 ± 0.40%	73.90 ± 0.30%
Munkhdalai et al. (2018)	57.10 ± 0.70%	70.04 ± 0.63%
TADAM (Oreshkin et al., 2018)	58.50 ± 0.30%	76.70 ± 0.30%
Qiao et al. (2018)	59.60 ± 0.41%	73.74 ± 0.19%
LEO	**61.76 ± 0.08**%	77.59 ± 0.12%
MetaFun-Attention	**62.12 ± 0.30**%	77.78 ± 0.12%
MetaFun-Kernel	61.16 ± 0.15%	**78.20 ± 0.16**%
(With data augmentation):		
Qiao et al. (2018)	**63.62 ± 0.58**%	78.83 ± 0.36%
LEO	63.97 ± 0.20%	79.49 ± 0.70%
MetaOptNet-SVM (Lee et al., 2019)[1]	**64.09 ± 0.62**%	80.00 ± 0.45%
MetaFun-Attention	**64.13 ± 0.13**%	**80.82 ± 0.17**%
MetaFun-Kernel	63.39 ± 0.15%	**80.81 ± 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 ± 1.81%	70.30 ± 0.08%
Prototypical Nets (Snell et al., 2017)	53.31 ± 0.89%	72.69 ± 0.74%
Relation Net [in Liu et al. (2019)]	54.48 ± 0.93%	71.32 ± 0.78%
Transductive Prop. Nets (Liu et al., 2019)	57.41 ± 0.94%	71.55 ± 0.74%
(With deep residual networks feature extraction):		
Meta-SGD	62.95 ± 0.03%	79.34 ± 0.06%
LEO	66.33 ± 0.05%	81.44 ± 0.09%
MetaOptNet-SVM	65.81 ± 0.74%	81.75 ± 0.58%
MetaFun-Attention	**67.72 ± 0.14**%	82.81 ± 0.15%
MetaFun-Kernel	67.27 ± 0.20%	**83.28 ± 0.12**%

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-Attention (using dot-product attention) and MetaFun-Kernel (using deep kernels) are reported. Although both of them demonstrate state-of-the-art performance, MetaFun-Kernel generally outperforms MetaFun-Attention for 5-shot problems, but performs slightly worse for 1-shot problems.

4.3 Ablation Study

As stated in Section 3.3, our model has three learnable components: the local update function, the kernel/attention, 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 3.2, consistently outperforms gradient-based local update function, despite the latter having build-in inductive biases. Interestingly, the choice between attention and deep kernel is problem dependent. We found that MetaFun with deep kernels usually perform better than MetaFun with 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

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.

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

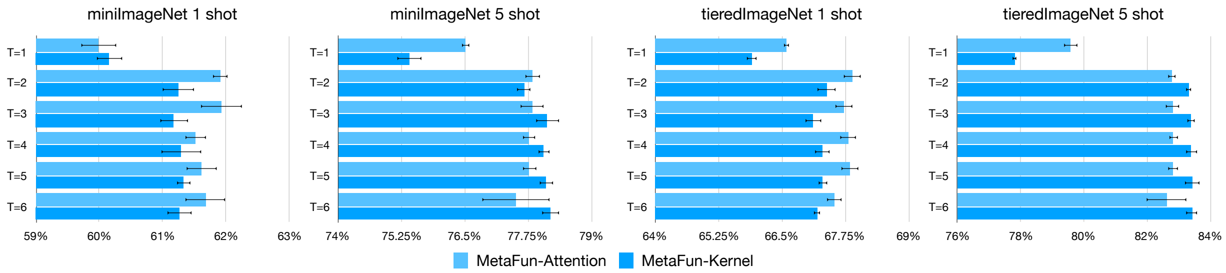

Figure 5: This figure illustrates the accuracy of our approach for varying number of iterations $T = 1, ..., 6$, over different few-shot learning problems. For each problem, we use the same configuration of hyperparameters except for the number of iterations and the choice between attention and deep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

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.

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 3.1). 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 approach for meta-learning called MetaFun. The proposed approach learns to iteratively update task representations in function space. We evaluate it on both few-shot regression and classification tasks, and demonstrate that it matches or exceeds previous state-of-the-art results on the challenging miniImageNet and tieredImageNet.

Interesting extensions to our work include: Exploring a stochastic encoder and hence working with stochastic functional representations, akin to NP, and not sharing the parameters in the local update functions and the kernel/attention components across iterations. The additional flexibility could lead to potential performance gains.

Acknowledgements

We would like to thank Jonathan Schwarz for valuable discussion. 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

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

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

Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartłomiej Świątkowski, Bernhard Schölkopf, and Richard E Turner. Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*, 2017.

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

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

Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetry and invariant neural networks. *arXiv preprint arXiv:1901.06082*, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey 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.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1690–1699, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

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

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *International Conference on Learning Representations*, 2019.

Llew Mason, Jonathan Baxter, Peter L Bartlett, Marcus Frean, et al. Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers. MIT Press*, 1999.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Tsendsuren Munkhdalai, Xingdi Yuan, Soroush Mehri, and Adam Trischler. Rapid adaptation with conditionally shifted neurons. In *International Conference on Machine Learning*, pages 3661–3670, 2018.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7229–7238, 2018.

David Raposo, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia. Discovering objects and their relations from entangled scene representations. *In Workshops at the International Conference on Learning Representations (ICLR)*, 2017.

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

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*, 2018.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. *arXiv preprint arXiv:1901.09006*, 2019.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

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

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

A Functional Gradient Descent

Functional gradient descent, just like gradient descent in parameter space, is an iterative optimisation algorithm for finding the minimum of a function. However, the function to be minimised now is actually a function on functions (*functional*). Formally, a functional $L : \mathcal{H} \to \mathbf{R}$ is a mapping from a function space $\mathcal{H}$ to a $1D$ Euclidean space $\mathbf{R}$, and the minimiser of $L(f)$ in function space is denoted as $\arg\min_{f \in \mathcal{H}} L(f)$

Just like gradient descent in parameter space taking steps proportional to the negative of the gradient, functional gradient descent updates f following the gradient in function space (*functional gradients*). In this work, we only consider a special function space called RKHS (Appendix A.1), and calculate functional gradients in RKHS (Appendix A.2). The algorithm is described in Appendix A.3 with further details using specific loss functions.

A.1 Reproducing Kernel Hilbert Space

A Hilbert space $\mathcal{H}$ extends the notion of Euclidean space by introducing inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which describes the concept of distance or similarity in this space. A RKHS $\mathcal{H}_k$ is a Hilbert space of functions with the reproducing property that for all $\mathbf{x}$ there exists a unique $k_{\mathbf{x}} \in \mathcal{H}_k$ such that the evaluation functional $E_{\mathbf{x}}(f) = f(\mathbf{x})$ can be represented by taking the inner product of an element $k_{\mathbf{x}}$ and f, formally written as:

$$E_{\mathbf{x}}(f) = \langle k_{\mathbf{x}}, f \rangle_{\mathcal{H}_k}. \tag{16}$$

Since $k_{\mathbf{x}'}$ is also a function in $\mathcal{H}_k$, we can define a kernel function $k(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$ by letting

$$k(\mathbf{x}, \mathbf{x}') = k_{\mathbf{x}'}(\mathbf{x}) = \langle k_{\mathbf{x}}, k_{\mathbf{x}'} \rangle_{\mathcal{H}_k}. \tag{17}$$

Using properties of inner product, it is easy to show that the kernel function $k(\mathbf{x}, \mathbf{x}')$ is symmetric and positive definite. We call this function *reproducing kernel* of the Hilbert space $\mathcal{H}_k$.

A.2 Functional Gradients

Functional derivative can be thought of as describing the rate of change of the output with respect to the input in a functional. Formally, functional derivative is defined as:

$$\frac{\partial L}{\partial f}(g) = \lim_{\epsilon \to 0} \frac{L(f + \epsilon g) - L(f)}{\epsilon}, \tag{18}$$

which is a function of g. This is known as *Fréchet derivative* in a Banach space, of which the Hilbert space is a special case.

Functional gradient, denoted as $\nabla_f L$, is related to functional derivative by the following equation:

$$\frac{\partial L}{\partial f}(g) = \langle \nabla_f L, g \rangle_{\mathcal{H}_k}. \tag{19}$$

It is straightforward to compute functional gradient of an evaluation functional in RKHS thanks to the reproducing property (Equation (16)):

$$\begin{aligned} E_{\mathbf{x}}(f + \epsilon g) &= \langle f + \epsilon g, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} \\ &= \langle f, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} + \epsilon \langle g, k_{\mathbf{x}} \rangle_{\mathcal{H}_k}. \end{aligned} \tag{20}$$

Therefore, the functional gradient of an evaluation functional $E_{\mathbf{x}}(f)$ is actually $k_{\mathbf{x}}$:

$$\frac{\partial E_{\mathbf{x}}}{\partial f}(\cdot) = \langle k_{\mathbf{x}}, \cdot \rangle_{\mathcal{H}_k} \tag{21}$$

$$\nabla_f E_{\mathbf{x}} = k_{\mathbf{x}}. \tag{22}$$

For a learning task with loss function ℓ and a context set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in C}$, the overall supervised loss on the context set can be written as:

$$L(f) = \frac{1}{|C|} \sum_{i \in C} \ell(f(\mathbf{x}_i), \mathbf{y}_i). \tag{23}$$

In this case, the functional gradient of L can be easily calculated by applying the chain rule:

$$\nabla_f L(f) = \frac{1}{|C|} \sum_{i \in C} \ell'(f(\mathbf{x}_i), \mathbf{y}_i) k_{\mathbf{x}_i} \tag{24}$$

$$= \frac{1}{|C|} \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \ell'(f(\mathbf{x}_i), \mathbf{y}_i). \tag{25}$$

This result matches Equation (5).

A.3 Functional Gradient Descent

To optimise the overall loss on the entire context set in Equation (23), we choose a suitable learning rate α, and iteratively update f with:

$$f^{(t+1)} = f^{(t)} - \alpha \nabla_f L(f^{(t)}) \tag{26}$$

$$= f^{(t)} - \frac{\alpha}{|C|} \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i) \tag{27}$$

In order to evaluate the final model $f^T(\mathbf{x})$ at iteration T, we only need to compute

$$f^{(T)}(\mathbf{x}) = f^{(0)}(\mathbf{x}) - \sum_{t=0}^{T-1} \frac{\alpha}{|C|} \sum_{i \in C} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i), \tag{28}$$

which does not depend on function values $f^{(t)}(\mathbf{x})$ outside the context from previous iterations $t < T$. In the case of a multidimensional function $f^{(t)}$, A multidimensional kernel should be used. However, in this work, we only consider a simple case where $k(\mathbf{x}, \mathbf{x}') = s(\mathbf{x}, \mathbf{x}')\mathbf{I}$, $s(\cdot, \cdot)$ produces a scalar, and $\mathbf{I}$ is an identity matrix. It is straightforward to derive the updating rule for a specific loss function ℓ. Below we consider two common cases: the mean square error loss for regression tasks, and the cross entropy loss for classification tasks which motivates our parametric form of the local update function used in Section 3.3.

Mean Square Error (MSE) loss for regression When MSE loss is adopted in a regression task, the loss function ℓ is defined as:

$$\ell(f(\mathbf{x}), \mathbf{y}) = \frac{1}{2}(f(\mathbf{x}) - \mathbf{y})^\top (f(\mathbf{x}) - \mathbf{y}) \tag{29}$$

Hence for context point $(\mathbf{x}_i, \mathbf{y}_i)$

$$\ell'(f(\mathbf{x}_i), \mathbf{y}_i) = f(\mathbf{x}_i) - \mathbf{y}_i \tag{30}$$

Note that this is simply the difference between prediction and label, which naturally describes how to change the prediction at location $\mathbf{x}_i$ in order to match the label $\mathbf{y}_i$. In Figure 1 we use a simple 1D regression task with MSE loss to illustrate functional gradient descent.

Cross Entropy (CE) loss for classification When we use cross entropy loss for a K-way classification problem, the model predicts K-dimensional logits $f(\mathbf{x}) = [f^1(\mathbf{x}), ..., f^K(\mathbf{x})]$. In this case, the cross entropy loss is

$$\ell(f(\mathbf{x}), \mathbf{y}) = -\sum_{k=1}^{K} y^k \log \frac{e^{f^k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{f^{k'}(\mathbf{x})}}, \tag{31}$$

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

Applying chain rule, functional gradient of the loss can be calculated as:

$$\nabla_f \ell(f(\mathbf{x}), \mathbf{y}) = [\nabla_{f^1}\ell(f(\mathbf{x}), \mathbf{y}), ..., \nabla_{f^K}\ell(f(\mathbf{x}), \mathbf{y})]$$

$$\nabla_{f^k} \ell(f(\mathbf{x}), \mathbf{y}) = \frac{e^{f^k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{f^{k'}(\mathbf{x})}} - y^k \tag{32}$$

This form has structure similar to the local update function we use for classification in Equation (14). The connection becomes clear if we let:

$$\begin{aligned} m(f^{k'}(\mathbf{x})) &= e^{f^{k'}(\mathbf{x})} \\ u_+(m(f^k(\mathbf{x})), \boldsymbol{m}) &= \frac{m(f^k(\mathbf{x}))}{\boldsymbol{m}} - 1 \\ u_-(m(f^k(\mathbf{x})), \boldsymbol{m}) &= \frac{m(f^k(\mathbf{x}))}{\boldsymbol{m}} \\ \boldsymbol{m} &= \sum_{k=1}^{K} m(f^{k'}(\mathbf{x})) \end{aligned} \tag{33}$$

and rewrite Equation (32) as:

$$\begin{aligned} \nabla_{f^k}\ell_{\mathcal{T}}(f(\mathbf{x}), \mathbf{y}) = y^k u_+(m(f^k(\mathbf{x})), \boldsymbol{m}) \\ + (1 - y^k) u_-(m(f^k(\mathbf{x})), \boldsymbol{m}) \end{aligned} \tag{34}$$

As our approach can be seen as extending functional gradient descent, Equation (34) motivates our design of local update function for classification problems.

B Experimental Details

All experiments in this work are implemented in Tensorflow(Abadi et al., 2015), and the code will be released upon publication. For miniImageNet and tieredImageNet, we conduct randomised hyperparameters search (Bergstra and Bengio, 2012) for hyperparameters tuning. Typically, 64 configurations of hyperparameters are sampled for each problem, and the best is chosen according to cross validation performance on the validation set. The considered range of hyperparameters is given in Table 4. The configurations of hyperparameters chosen to report the final classification accuracies are recorded in Table 5 for reference. For regression tasks, we simply use hyperparameters presented in Table 6 for both attention version and deep kernel version of our approach.

Table 4: Considered Range of Hyperparameters. The random generators such as randint or uniform use numpy.random syntax, so the first argument is inclusive while the second argument is exclusive. Whenever a list is given, it means uniformly sampling from the list. u_+ and u_- will be followed by a linear transformation with an output dimension of *dim-reprs*.

Components	Architecture
Shared MLP m	*nn-sizes* × *nn-layers*
MLP for positive labels u_+	*nn-sizes* × *nn-layers*
MLP for negative labels u_-	*nn-sizes* × *nn-layers*
Key/query transformation MLP a	$dim(\mathbf{x})$ × *embedding-layers*
Decoder	linear with output dimension $dim(\mathbf{x})$

Hyperparameters	Considered Range
num-iters	randint(2, 7)
nn-layers	randint(2, 4)
embedding-layers	randint(1, 3)
nn-sizes	$[64, 128]$
dim-reprs	=*nn-sizes*
Initial representation $\mathbf{r}^0$	[zero, constant, parametric]
Outer learning rate	10^{-5} × uniform(-5, -4)
Initial inner learning rate	$[0.1, 1.0, 10.0]$
Dropout rate	uniform(0.0, 0.5)
Orthogonality penalty weight	$10^{\text{uniform(-4, -2)}}$
L2 penalty weight	$10^{\text{uniform(-10, -8)}}$
Label smoothing	$[0.0, 0.1, 0.2]$

Table 5: Results of randomised hyperparameters search. Hyperparameters shown in this table are not guaranteed to be optimal within the considered range, because we conduct randomised hyperparameters search. Models configured by these hyperparameters perform reasonably well, and we used them to report final results comparing to other methods. Dropout is only applied to the inputs. Orthogonality penalty weight and L2 penalty weight are used in exactly the same way as in Rusu et al. (2019) and their released code at https://github.com/deepmind/leo. Inner learning rate is trainable so only an initial inner learning rate is given in the table.

	miniImageNet		**tieredImageNet**	
Hyperparameters (MetaFun-Attention)	1-shot	5-shot	1-shot	5-shot
num-iters	2	5	3	5
nn-layers	3	2	2	3
embedding-layers	2	2	1	1
nn-sizes	64	128	128	128
Initial state	zero	constant	constant	constant
Outer learning rate	8.56×10^{-5}	3.71×10^{-5}	5.55×10^{-5}	5.78×10^{-5}
Initial inner learning rate	0.1	10.0	1.0	1.0
Dropout rate	0.397	0.075	0.123	0.223
Orthogonality penalty weight	3.28×10^{-3}	1.56×10^{-3}	1.37×10^{-3}	2.58×10^{-3}
L2 penalty weight	1.32×10^{-10}	2.60×10^{-10}	1.92×10^{-9}	1.63×10^{-9}
Label smoothing	0.2	0.2	0.1	0.0

	miniImageNet		**tieredImageNet**	
Hyperparameters (MetaFun-Kernel)	1-shot	5-shot	1-shot	5-shot
num-iters	3	6	4	4
nn-layers	3	2	2	3
embedding-layers	2	2	1	1
nn-sizes	64	64	64	128
Initial state	zero	parametric	parametric	zero
Outer learning rate	4.21×10^{-5}	8.60×10^{-5}	8.01×10^{-5}	4.50×10^{-5}
Initial inner learning rate	0.1	0.1	0.1	0.1
Dropout rate	0.424	0.359	0.115	0.148
Orthogonality penalty weight	2.69×10^{-3}	2.73×10^{-4}	1.06×10^{-4}	7.33×10^{-3}
L2 penalty weight	1.19×10^{-9}	1.68×10^{-9}	4.90×10^{-9}	6.22×10^{-9}
Label smoothing	0.2	0.2	0.1	0.1

Table 6: Hyperparameters for regression tasks. Local update function and the predictive model will be followed by linear transformations with output dimension of *dim-reprs* and *dim*($\mathbf{y}$) accordingly.

Components	Architecture
Local update function	*nn-sizes* × *nn-layers*
Key/query transformation MLP a	*nn-sizes* × *embedding-layers*
Decoder	*nn-sizes* × *nn-layers*
Predictive model	*nn-sizes* × (*nn-layers*-1)

Hyperparameters	Considered Range
num-iters	5
nn-layers	3
embedding-layers	3
nn-sizes	128
dim-reprs	=*nn-sizes*
Initial representation $\mathbf{r}^0$	zero
Outer learning rate	10^{-4}
Initial inner learning rate	0.1
Dropout rate	0.0
Orthogonality penalty weight	0.0
L2 penalty weight	0.0