Appendix for MetaFun: Meta-Learning with Iterative Functional Updates

A. Functional Gradient Descent

Functional gradient descent (Mason et al., 1999; Y. Guo & Williamson, 2001) is an iterative optimisation algorithm for finding the minimum of a function. However, the function to be minimised is now 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 $\mathbf{R}$. Just like gradient descent in parameter space which takes steps proportional to the negative of the gradient, functional gradient descent updates f following the gradient in function space. In this work, we only consider a special function space called Reproducing kernel Hilbert space (RKHS) (Appendix A.1), and calculate functional gradients in RKHS (Appendix A.2). The algorithm is further detailed in Appendix A.3.

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 real-valued functions on $\mathcal{X}$ with the reproducing property that for all $\mathbf{x} \in \mathcal{X}$ 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 this element $k_{\mathbf{x}}$ and f, formally as:

$$E_{\mathbf{x}}(f) = \langle k_{\mathbf{x}}, f \rangle_{\mathcal{H}_k}. \tag{1}$$

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

$$k(\mathbf{x}, \mathbf{x}') = k_{\mathbf{x}'}(\mathbf{x}) = \langle k_{\mathbf{x}}, k_{\mathbf{x}'} \rangle_{\mathcal{H}_k}. \tag{2}$$

Using properties of inner product, it is easy to show that the kernel function $k(\mathbf{x}, \mathbf{x}')$ is symmetric and positive definite, and we call it the *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 at point f in the direction of g is defined as:

$$\frac{\partial L}{\partial f}(g) = \lim_{\epsilon \to 0} \frac{L(f + \epsilon g) - L(f)}{\epsilon}, \tag{3}$$

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:

$$\frac{\partial L}{\partial f}(g) = \langle \nabla_f L, g \rangle_{\mathcal{H}_k}. \tag{4}$$

Thanks to the reproducing property, it is straightforward to calculate functional derivative of an evaluation functional in RKHS:

$$\begin{aligned} E_{\mathbf{x}}(f + \epsilon g) &= \langle f + \epsilon g, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} \\ &= \langle f, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} + \epsilon \langle g, k_{\mathbf{x}} \rangle_{\mathcal{H}_k} \end{aligned} \tag{5}$$

$$\frac{\partial E_{\mathbf{x}}}{\partial f}(g) = \langle k_{\mathbf{x}}, g \rangle_{\mathcal{H}_k} \tag{6}$$

Therefore, the functional gradient of an evaluation functional is:

$$\nabla_f E_{\mathbf{x}} = k_{\mathbf{x}}. \tag{7}$$

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

$$L(f) = \sum_{i \in \mathbf{C}} \ell(f(\mathbf{x}_i), \mathbf{y}_i). \tag{8}$$

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

$$\nabla_f L = \sum_{i \in \mathbf{C}} \ell'(f(\mathbf{x}_i), \mathbf{y}_i) k_{\mathbf{x}_i} \tag{9}$$

$$= \sum_{i \in \mathbf{C}} k(\cdot, \mathbf{x}_i) \ell'(f(\mathbf{x}_i), \mathbf{y}_i). \tag{10}$$

A.3. Functional Gradient Descent

To optimise the overall loss on the entire context in Equation (8), we choose a suitable learning rate α, and iteratively update f with:

$$f^{(t+1)}(\mathbf{x}) = f^{(t)}(\mathbf{x}) - \alpha \nabla_f L(f^{(t)})(\mathbf{x}) \tag{11}$$

$$= f^{(t)}(\mathbf{x}) - \alpha \sum_{i \in \mathbf{C}} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i) \tag{12}$$

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

$$f^{(T)}(\mathbf{x}) = f^{(0)}(\mathbf{x}) - \sum_{t=0}^{T-1} \alpha \sum_{i \in \mathbf{C}} k(\mathbf{x}, \mathbf{x}_i) \ell'(f^{(t)}(\mathbf{x}_i), \mathbf{y}_i), \tag{13}$$

which does not depend on function values outside the context from previous iterations $t < T$.

B. Experimental Details

We run experiments on Nvidia's GeForce GTX 1080 Ti, and it typically takes about 20–40 minutes to train a few-shot model on a single GPU card until early-stopping is triggered (after seeing $10k$–$100k$ tasks). For miniImageNet and tieredImageNet, we conduct randomised hyperparameters search (Bergstra & Bengio, 2012) for hyperparameters tunning. Typically, 64 configurations of hyperparameters are sampled for each problem, and the best configuration is chosen by comparing accuracy on the validation set. The considered range of hyperparameters is given in Table 1, and the chosen hyperparameters are shown in Table 2. For regression tasks, we simply use hyperparameters listed in Table 3 for both MetaFun-DFP and MetaFun-KFP.

Table 1. 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* $\times$ *nn-layers*
MLP for positive labels u_+	*nn-sizes* $\times$ *nn-layers*
MLP for negative labels u_-	*nn-sizes* $\times$ *nn-layers*
Key/query transformation MLP a	*dim*($\mathbf{x}$) $\times$ *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	$10^{-5} \times$ `uniform(-5, -4)`
Initial inner learning rate	$[0.1, 1.0, 10.0]$
Dropout rate	`uniform(0.0, 0.5)`
Orthogonality penalty weight	$10^{\texttt{uniform(-4, -2)}}$
L2 penalty weight	$10^{\texttt{uniform(-10, -8)}}$
Label smoothing	$[0.0, 0.1, 0.2]$

Table 2. 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. However, models configured with these hyperparameters perform reasonably well, and we used them to report final results comparing to other methods. Furthermore, 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). Inner learning rate α is trainable so only an initial inner learning rate is given in the table.

	miniImageNet		**tieredImageNet**	
Hyperparameters (for MetaFun-DFP)	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 (for MetaFun-KFP)	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 3. 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*(**y**) accordingly.

Components	Architecture
Local update function	*nn-sizes* $\times$ *nn-layers*
Key/query transformation MLP a	*nn-sizes* $\times$ *embedding-layers*
Decoder	*nn-sizes* $\times$ *nn-layers*
Predictive model	*nn-sizes* $\times$ (*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

References

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

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.

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.

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