Meta-learning / meta architecture search / PINNs

Contents

1 Motivation

- We are interested in the following problems:

Problem (1): Obtaining a well-performing set of hyperparameters for solving a specific task. Examples include:

- Fitting a regression curve to a given dataset; what's the best architecture?
- Solving a PDE with a fixed set of PDE parameters (PINNs); what's the best loss function to use?
- Using B-PINNs for obtaining a stochastic solution to a deterministic PDE; what's the best prior parameters?

Problem (2): Obtaining a well-performing set of hyperparameters such that if used for solving an arbitrary unseen task it will perform well. We assume that this new task is a sample from a task distribution that we have samples from. You may see it as few-shot/iterations learning in the literature. Examples include:

- New task will be given by $f(x) = sin(\lambda x)$ where $\lambda \in Uniform([0, 1])$. Find the architecture that will perform best if used for fitting data corresponding to a new λ value.
- New PDE will be given by $\lambda \partial_x^2 u - u = f(x)$ where f is known and $\lambda \in Uniform([0, 1])$. Find the loss function that will produce the most accurate $u(x)$ if used in conjunction with PINNs for a new λ value.
- New PDE will be given as above. Assuming a Gaussian prior for the parameters of the fitting NN, find the prior standard deviation such that if used in conjunction with B-PINNs for a new λ value it will produce the most accurate $u(x)$. This means that tasks *share the prior* and given a new task we still have to do posterior approximation! But hopefully it will be faster and/or more accurate.

Problem (3): Finding a way to optimize hyperparameters efficiently for a new task. You may see it as few-shot/iterations meta-learning in the literature. Examples include:

- For a new regression task (new λ) we want to make architecture search efficient. This means that tasks *do not share* some obtained architecture but *use the same technique* for finding it. Therefore, we get a task-specific optimal architecture!

  - For a new PDE we want to make loss function optimization efficient. Tasks do not share the same loss function but use the same technique for optimizing it. We get a *task-specific optimal loss function*!

2 General meta-learning formulation

- **Problems (1)** and **(2)** are considered first. See Section 5 for **Problem (3)**.
- Meta-training is commonly formalized as follows:

$$\begin{aligned} &\min_x \mathcal{L}_1(y^*(x), x) \\ \text{s.t. } &y^*(x) = argmin_y \mathcal{L}_2(y, x) \\ \text{i.e., for GD: } &y^*(x) = y_0(x) - \sum_{j=1}^{J} \epsilon_2 g_j(x, y_{j-1}) \end{aligned} \tag{1}$$

- This is called bi-level optimization: note that x and y are not optimized together.
- x denotes hyperparameters to be optimized and y the NN parameters.
- $\mathcal{L}_1$ is the meta-objective (or outer), $\mathcal{L}_2$ is the base-learning objective (or lower or inner).

  1. For **Problem (1)**:

$$\begin{aligned} \mathcal{L}_1(y^*(x), x) &= \mathbb{E}_{D_{val}} \mathcal{L}(y^*(x), x) \\ \mathcal{L}_2(y, x) &= \mathbb{E}_{D_{train}} \mathcal{L}(y, x) \end{aligned} \tag{2}$$

  2. For **Problem (2)**:

$$\begin{aligned} y^*(x) &= \{y^*_{task}\}_{tasks} \\ \mathcal{L}_1(y^*(x), x) &= \mathbb{E}_{tasks} \mathbb{E}_{D_{val,task}} \mathcal{L}(y^*_{task}(x), x) \\ \mathcal{L}_2(y, x) &= \mathbb{E}_{D_{train,task}} \mathcal{L}(y, x) \end{aligned} \tag{3}$$

- $\mathbb{E}_D \mathcal{L}(y, x)$ is the average loss over the data defined by D (e.g., squared error) evaluated on the NN parameters y. It may depend on x (e.g., parametrized loss function).
- Based on this formulation, **Problem (1)** is a special case of **Problem (2)**; just one task is considered and we "meta-overfit" on this task: most likely the obtained optimal hyperparameters cannot be transferred for solving other tasks.

3 Placing 4 meta-learning techniques under the general formulation

3.1 NAS-DARTS

x	architecture parameters
y	NN parameters
$y^*(x)$	optimal NN parameters corresponding to architecture defined by x

- As defined originally by Liu et al. (2019), it solves **Problem (1)**: finds best architecture for solving a specific task.
- One inner optimization step is typically used ($J = 1$).
- $y_0(x) = y^*$ of previous iteration: we start with an architecture, we update weights, then we update architecture, and then update weights again.
- For regression, $\mathcal{L}_1$ is the MSE on the validation data and $\mathcal{L}_2$ the MSE on the training data.
- $\mathcal{L}_1$ **does depend** explicitly on x. We take total derivative $\frac{d}{dx}\mathcal{L}_1(y^*(x), x)$ for updating x, so this is important.
- When done, NN parameters need retraining because the architecture is hard-pruned.

3.2 Prior optimization - Empirical Bayes

x	prior parameters
y	posterior parameters
$y^*(x)$	optimal posterior parameters corresponding to prior defined by x

- It solves **Problem (1)**: finds best prior for solving a specific task.
- One inner optimization step is typically used ($J = 1$).
- $y_0(x) = y^*$ of previous iteration: we start with a prior, we update posterior parameters, then we update prior, and then update posterior again.

- For VI, $\mathcal{L}_1$ and $\mathcal{L}_2$ are the ELBO on the training data (usually no validation data is used - check).
- $\mathcal{L}_1$ **does depend** explicitly on x.
- When done, posterior parameters can be used as they are. No need for retraining using the obtained prior.

3.3 MAML

x	initial NN parameters
y	NN parameters
$y^*(x)$	optimal NN parameters

- As defined originally by Finn et al. (2017), it solves **Problem (2)**: finds best initial parameters such that if used for a new learning task final parameters will be obtained fast and be well-performing.
- One inner optimization step is used ($J = 1$) but it is performed for every task.
- $y_0(x) = x$: note the difference!
- For regression, $\mathcal{L}_1$ and $\mathcal{L}_2$ are the mean MSE on randomly sampled data from randomly sampled tasks.
- $\mathcal{L}_1$ **does not depend** explicitly on x.

3.4 Loss function meta-learning

x	loss function parameters
y	NN parameters
$y^*(x)$	optimal NN parameters

- As defined originally by Bechtle et al. (2020), it solves **Problem (2)**: finds best loss function parameters such that if used for a new learning task final parameters will be obtained fast and be well-performing.
- One or more inner optimization steps are used ($J = 1$) and are performed for one task.

- $y_0(x)$ is randomly initialized.
- For regression, $\mathcal{L}_1$ is the MSE on randomly sampled data from the selected task and $\mathcal{L}_2$ is the *learned loss function* on randomly sampled data from the selected task.
- Selected task can change after a few iterations.
- $\mathcal{L}_1$ **does not depend** explicitly on x.

4 Making task-specific techniques (Problem 1) be task-agnostic (Problem 2)

- Based on Eqs. (2)-(3), for making a task-specific technique be task-agnostic we have to perform the inner-optimization for every task separately, obtain $y^*_{task}(x)$ for every task and then update x based on the average performance of the $y^*_{task}(x)$ values.
- Practically, this means that we can use NAS-DARTS to find an optimal architecture for a set of tasks. This is what Auto-Meta (Kim et al., 2018) does, but they use PNAS instead of DARTS for architecture search.
  - Has it been done for NAS-DARTS?
- Can we obtain also a good initialization for this shared architecture? It's not too hard to combine NAS and MAML (we can update architecture and initialization together in the outer loop).
  - Inner loop: Update parameters based on current architecture and current initial parameters
  - Outer update: Update architecture and initial parameters simultaneously
- Auto-MAML (Lian et al., 2019) is a variant of the above (algorithm p. 15 - algorithm in the paper has typos - check):
  - Inner loop 1: Update parameters based on current architecture and current initial parameters
  - Outer update 1: Update initial parameters as we do in MAML

  - Inner loop 2: Update parameters based on current architecture and current initial parameters
  - Outer update 2: Update architecture as we do in DARTS

- The only problem is that the NAS-obtained architecture needs to be sparsified before using it. So how good are the obtained initial parameters if we hard-prune the architecture after meta-training? Look in Elsken et al. (2020) (Section 4) for the solution they propose regarding this.

- Another instance of making task-specific techniques (**Problem 1**) be task-agnostic (**Problem 2**) would be to find a good prior that performs well for a distribution of tasks and not only for a single task.

5 Addressing Problem (3)

- It can be addressed as a combination of **Problems (1)** and **(2)**: Obtain a well-performing set of hyperparameters such that if used for some other hyperparameter optimization for an unseen task it will perform well.

- **Example:** Find some initial architecture that with only a few optimization/adaptation steps can become optimal for an unseen task. This is done by combining NAS-DARTS with MAML: Recall that in MAML we look for optimal initialization. Here we optimize architecture and NN parameters together and we look for a good initialization of both.

x	initial architecture and corresponding parameters
y	architecture and corresponding parameters
$y^*(x)$	optimal architecture and corresponding parameters

- This is what T-NAS (transferable NAS) does (Lian et al., 2019). Recall however that both NAS and MAML are two-step procedures. Thus, the above requires 3 steps which increases the computational cost. This is why Lian et al. (2019) propose to reduce NAS into a single step (weights are updated simultaneously with architecture). However, this is something that the original NAS authors (Liu et al., 2019 - Section 3.3) suggest to avoid because it leads to overfitting the architecture (we have exactly the same problem if we optimize prior and posterior simultaneously).

- In summary, this is T-NAS:
  - In meta-train time, it finds a good architecture and parameter initialization
  - In meta-test time, it quickly optimizes architecture and parameters with standard NAS (architecture is still continuous)
  - Then architecture is pruned
  - Finally, model is retrained with pruned architecture

  Inner loop for each task i ($\theta_i^0 = \tilde{\theta}$ and $w_i^0 = \tilde{w}$):

$$\begin{cases} w_i^{m+1} = w_i^m - \alpha_{\text{inner}} \nabla_{w_i^m} \mathcal{L}(g(\mathcal{T}_i^s; \theta_i^m, w_i^m)) \\ \theta_i^{m+1} = \theta_i^m - \beta_{\text{inner}} \nabla_{\theta_i^m} \mathcal{L}(g(\mathcal{T}_i^s; \theta_i^m, w_i^{m+1})) \end{cases}$$

  Outer update:

$$\begin{cases} \widetilde{w} = \widetilde{w} - \alpha_{\text{outer}} \nabla_{\widetilde{w}} \sum_{\mathcal{T}_i^q \sim p(\mathcal{T})} \mathcal{L}(g(\mathcal{T}_i^q; \theta_i^M, w_i^M)) \\ \widetilde{\theta} = \widetilde{\theta} - \beta_{\text{outer}} \nabla_{\widetilde{\theta}} \sum_{\mathcal{T}_i^q \sim p(\mathcal{T})} \mathcal{L}(g(\mathcal{T}_i^q; \theta_i^M, w_i^M)) \end{cases}$$

- Clearly, having to retrain after pruning poses a computational cost limitation.
- Elsken et al. (2020) addresses this issue by using soft-pruning: their technique is called Meta-NAS.

6 Bayesian way for addressing Problem (3)

This is "meta architecture search" as proposed by Shaw et al. (2019). Let's incorporate it into our general framework. First, we discuss Bayesian NAS which aims to solve **Problem (1)** (seems to be what BayesNAS in Zhou et al., 2019 does - check) and then we discuss the technique of Shaw et al. (2019) which aims to solve **Problem (3)**, but is essentially based on Bayesian NAS.

6.1 Bayesian NAS

- A first step towards a Bayesian variant of NAS is to replace the two-step optimization of NAS by one step: optimize x (architecture) and y (NN parameters) simultaneously. Once again, this leads to overfitting for the deterministic case (Liu et al., 2019).

- Next, since this is a simple optimization we can make it Bayesian: instead of looking for the best pair [architecture, parameters] we look for their posterior. The best pair [architecture, parameters] corresponds to the maximum posterior probability.
- After obtaining the posterior we can sample from it and use the obtained pair [architecture, parameters] as the predictive model.
- In summary, not only we have obtained the best architecture but we have also trained it. And by sampling the posterior we also have its uncertainty.
- In standard Bayesian framework a sample from the parameter posterior is used in the only architecture we have in order to obtain a sample prediction. In Bayesian NAS a sample from the posterior gives us both an architecture and its optimal parameters and we get a sample prediction by evaluating the architecture on these parameters.

6.2 Meta architecture search

- For an unseen task we want to perform Bayesian NAS efficiently.
- In other words, we want to obtain a prior for Bayesian NAS that works well for many tasks.
- Inner loop: perform Bayesian NAS for all tasks **separately** with fixed prior. Obtain one posterior of [architecture, parameters] for each task.
- Outer update: Update prior based on the performance of obtained posteriors towards maximizing average (across tasks) ELBO.

7 General algorithms

Algorithm for Problem 1: Good hyperparameters for a specific task

input: ϵ_1, ϵ_2, $\mathcal{L}_1$, $\mathcal{L}_2$ and task data

initialize x with x_0 ▷ `also` y_0^* `if needed (e.g., in NAS-DARTS)`

for $i \in \{1, \ldots, I\}$ **do**

  initialize y with y_0 ▷ $y_0 = y_0(x_{i-1})$ `or` $y_0(y_{i-1}^*)$

  for $j \in \{1, \ldots, J\}$ **do**

    $y_j = y_{j-1} - \epsilon_2 \nabla_y \mathcal{L}_2(y, x)\Big|_{y=y_{j-1},\ x=x_{i-1}}$ ▷ `Inner step`

  end

  set $y_i^* = y_J$ ▷ `for GD:` $y_i^* = y_0 - \sum_{j=1}^{J} \epsilon_2 g_j(y_{j-1}, x_{i-1})$

  $x_i = x_{i-1} - \epsilon_1 \boldsymbol{d}_x \mathcal{L}_1(y^*(x), x)\Big|_{y^*=y_i^*,\ x=x_{i-1}}$ ▷ `Outer step`

end

return x_I and y_I^*

Algorithm for Problem 2: Good hyperparameters for a task distribution

input: ϵ_1, ϵ_2, $\mathcal{L}_1$, $\mathcal{L}_2$ and task distribution $p(\mathcal{T})$

initialize x with x_0

for $i \in \{1, \ldots, I\}$ **do**

  sample T tasks from $p(\mathcal{T})$

  for $\tau \in \{1, \ldots, T\}$ **do**

    initialize $y^{(\tau)}$ with y_0 ▷ $y_0 = y_0(x_{i-1})$ `or` $y_0(y_{i-1}^{*(\tau)})$ `or random`

    for $j \in \{1, \ldots, J\}$ **do**

      $y_j^{(\tau)} = y_{j-1}^{(\tau)} - \epsilon_2 \nabla_y \mathcal{L}_2^{(\tau)}(y, x)\Big|_{y=y_{j-1}^{(\tau)},\ x=x_{i-1}}$ ▷ `Inner step for` τ

    end

    set $y_i^{*(\tau)} = y_J^{(\tau)}$

  end

  $x_i = x_{i-1} - \epsilon_1 \boldsymbol{d}_x \mathbb{E}_\tau \left[\mathcal{L}_1(y^*(x), x)\Big|_{y^*=y_i^{*(\tau)},\ x=x_{i-1}} \right]$ ▷ `Outer step`

end

return x_I

8 Gradient of meta-objective $\mathcal{L}_1$ with respect to x

- If x and y were one-dimensional:

$$\frac{d}{dx}\mathcal{L}_1(y^*(x), x) = \frac{\partial \mathcal{L}_1}{\partial x} + \frac{\partial \mathcal{L}_1}{\partial y^*}\frac{\partial y^*}{\partial x} \tag{4}$$

- First term relates to direct dependence of $\mathcal{L}_1$ on x and second term to dependence through the inner optimal $y^*(x)$
- You may see the second term as *"differentiating over the optimization path"*
- For multi-dimensional x and y we denote with bold $\boldsymbol{d}_x$ the total derivative with respect to x and with ∇_x the partial:

$$\boldsymbol{d}_x\mathcal{L}_1(y^*(x), x) = \nabla_x\mathcal{L}_1 + \mathcal{J}(y^*(x), x)\nabla_{y^*}\mathcal{L}_1 \tag{5}$$

where $\mathcal{J}$ is the Jacobian matrix of the transformation from x to $y^*(x)$

- If only one gradient descent inner optimization step is considered ($J = 1$, $y = y_0$):

$$\boldsymbol{d}_x\mathcal{L}_1(y^*(x), x) = \nabla_x\mathcal{L}_1 + \left[\mathcal{J}(y_0(x), x) - \epsilon_2\nabla^2_{x,y_0}\mathcal{L}_2\right]\nabla_{y^*}\mathcal{L}_1 \tag{6}$$

8.1 NAS-DARTS

- Initialization of inner optimization does not depend on x: it's just the y^* of the previous outer step
- Therefore, Eq. (6) becomes ($J = 1$, $y = y_0$):

$$\boldsymbol{d}_x\mathcal{L}_1(y^*(x), x) = \nabla_x\mathcal{L}_1 - \epsilon_2\nabla^2_{x,y_0}\mathcal{L}_2\nabla_{y^*}\mathcal{L}_1 \tag{7}$$

- Compare this result with Eq. (7) in the original paper Liu et al. (2019)
- Liu et al. (2019) proposes two approximation alternatives
    1. Finite difference approximation of $\nabla^2_{x,y_0}\mathcal{L}_2\nabla_{y^*}\mathcal{L}_1$
    2. Disregarding $\nabla^2_{x,y_0}\mathcal{L}_2\nabla_{y^*}\mathcal{L}_1$ (first-order approximation)

8.2 Prior optimization - Empirical Bayes

- Initialization of inner optimization does not depend on x: it's just the y^* of the previous outer step
- Therefore Eq. (6) becomes ($J = 1$, $y = y_0$):

$$\boldsymbol{d}_x \mathcal{L}_1(y^*(x), x) = \nabla_x \mathcal{L}_1 - \epsilon_2 \nabla^2_{x,y_0} \mathcal{L}_2 \nabla_{y^*} \mathcal{L}_1 \tag{8}$$

- Second term is usually omitted and only $\nabla_x \mathcal{L}_1$ is used (first-order like above)

8.3 MAML

- $\mathcal{L}_1$ does not depend explicitly on x
- Therefore, Eq. (5) (many inner steps) becomes:

$$\boldsymbol{d}_x \mathcal{L}_1(y^*(x), x) = \mathcal{J}(y^*(x), x) \nabla_{y^*} \mathcal{L}_1 \tag{9}$$

- Initialization of inner optimization depends on x: $y_0(x) = x$
- Therefore, Eq. (6) becomes ($J = 1$, $y = y_0$):

$$\boldsymbol{d}_x \mathcal{L}_1(y^*(x), x) = \left[I - \epsilon_2 \nabla^2_x \mathcal{L}_2\right] \nabla_{y^*} \mathcal{L}_1 \tag{10}$$

- Compare these results with Eq. (4) in Nichol et al. (2018)
- Indicatively, we have 4 alternatives
  1. Full backprop MAML: compute $\mathcal{J}(y^*(x), x)$
     - See code at: https://higher.readthedocs.io/en/latest/
  2. First-order MAML (FOMAML): Approximate $\mathcal{J}(y^*(x), x)$ by the identity matrix
  3. Reptile: Take $\boldsymbol{d}_x \mathcal{L}_1(y^*(x), x) = y_0 - y^*(x)$, i.e., $x - y^*(x)$
  4. Implicit MAML (iMAML): see Rajeswaran et al. (2019)
- Reptile has been later used also in the context of other meta-learning techniques (e.g., in Shaw et al., 2019)

8.4 Loss function meta-learning

- The technique of Bechtle et al. (2020) is considered
- Initialization of inner optimization does not depend on x
- $\mathcal{L}_1$ does not depend explicitly on x
- Therefore, Eq. (5) (many inner steps) becomes:

$$\boldsymbol{d}_x \mathcal{L}_1(y^*(x), x) = \mathcal{J}(y^*(x), x)\nabla_{y^*}\mathcal{L}_1 \tag{11}$$

- Indicative alternatives for obtaining $\mathcal{J}(y^*(x), x)\nabla_{y^*}\mathcal{L}_1$:
  1. Full backprop (See code at: https://higher.readthedocs.io/en/latest/)
  2. Kenji proposed to check implicit function theorem
  3. Approximations developed for other meta-learning techniques can be used in this context as well (first-order, reptile, etc)

9 Optimization comments

- We can refer to x and y as being optimized together/simultaneously if outer and inner steps are combined and a common loss function $\mathcal{L}$ is used:

$$\begin{bmatrix} y' \\ x' \end{bmatrix} = \begin{bmatrix} y \\ x \end{bmatrix} - \epsilon \begin{bmatrix} \frac{\partial \mathcal{L}(y,x)}{\partial y} \\ \frac{\partial \mathcal{L}(y,x)}{\partial x} \end{bmatrix} \tag{12}$$

- For example, Barron (2019) optimizes the loss function and the NN parameters simultaneously
- If they are optimized separately then x' depends on the updated y' and for one inner step ($J = 1$) we have:

$$y' = y - \epsilon_2 \frac{\partial \mathcal{L}_2(y, x)}{\partial y} \tag{13}$$

$$x' = x - \epsilon_1 \frac{d\mathcal{L}_1(y'(x), x)}{dx} \tag{14}$$

- For example, Bechtle et al. (2020) optimizes the loss function and the NN parameters separately

10 Benchmark regression tasks

10.1 *1D x / 1D y*

10.1.1 Sinusoidal functions with varying phase shifts

Tasks are given by

$$y = A\sin(\omega x + s) \tag{15}$$

where ω is fixed whereas A, s vary.

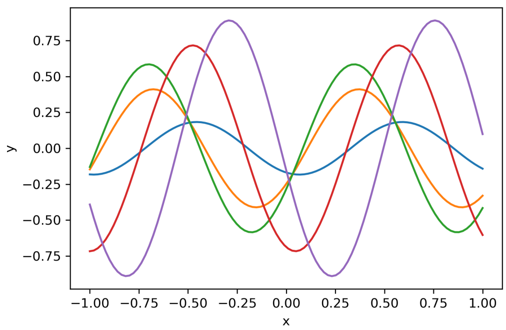

Fig. 1. 5 indicative tasks for $\omega = 6$, $x \in [-1, 1]$, A drawn from $Uniform([0.1, 1])$ and s drawn from $Uniform([-\pi, \pi])$.

10.1.2 Tanh + sinusoidal functions with varying sharpness

Tasks are given by

$$y = \alpha_1 \tanh(\omega_1 x) + \alpha_2 \sin(\omega_2 x) \tag{16}$$

where $\boldsymbol{\omega}$, and $\boldsymbol{\alpha}$ vary.

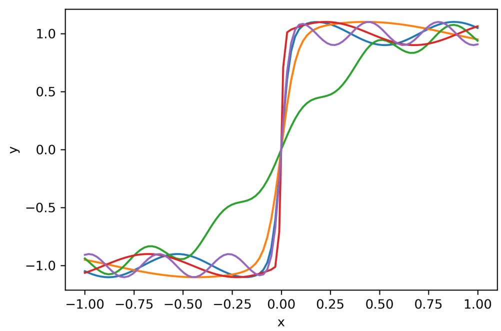

Fig. 2. 5 indicative tasks for $x \in [-1, 1]$, ω_1 given by $10^{\omega_1'}$ where ω_1' is drawn from $Uniform([0, 2])$ and ω_2 drawn from $Uniform([1, 20])$.

10.2 2D x / 1D y

10.2.1 Product of sines with varying frequency

Tasks are given by

$$y = \sin(\omega x_1)\sin(\omega x_2) \tag{17}$$

where ω varies.

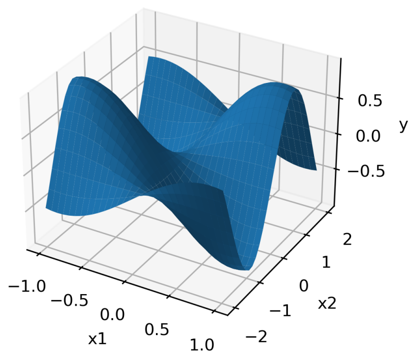

Fig. 3. An indicative task for $x_1 \in [-1, 1]$, $x_2 \in [-2, 2]$, and ω drawn from $Uniform([1, 3])$.

10.3 2D x / 2D y

10.3.1 Product of sines and cosines with varying frequencies

Tasks are given by

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} sin(\omega_1 x_1)\sin(\omega_1 x_2) \\ \cos(\omega_2 x_1)\cos(\omega_2 x_2) \end{bmatrix} \tag{18}$$

where ω_1, ω_2 vary.

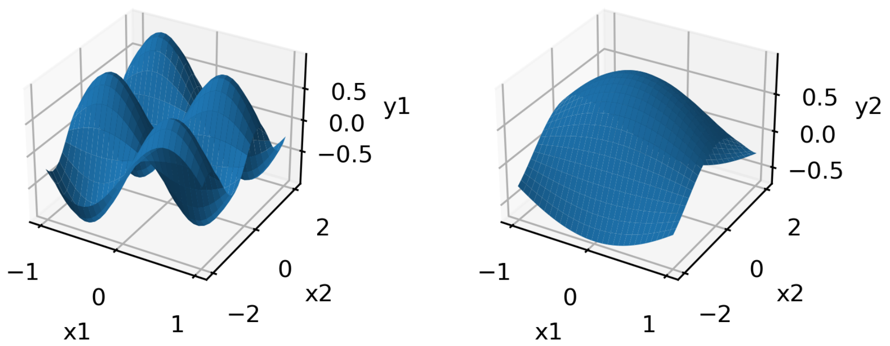

Fig. 4. An indicative task for $x_1 \in [-1, 1]$, $x_2 \in [-2, 2]$, and ω_1, ω_2 drawn from $Uniform([1, 3])$.

References

Barron, J. T. (2019). "A General and Adaptive Robust Loss Function". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4331–4339.

Bechtle, S. et al. (2020). "Meta-Learning via Learned Loss". *arXiv:1906.05374 [cs, stat]*. arXiv: 1906.05374 [cs, stat].

Elsken, T., Staffler, B., Metzen, J. H., and Hutter, F. (2020). "Meta-Learning of Neural Architectures for Few-Shot Learning". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12365–12375.

Finn, C., Abbeel, P., and Levine, S. (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". *arXiv preprint arXiv:1703.03400*. arXiv: 1703.03400.

Kim, J. et al. (2018). "Auto-Meta: Automated Gradient Based Meta Learner Search". *arXiv preprint arXiv:1806.06927*. arXiv: 1806.06927.

Lian, D. et al. (2019). "Towards Fast Adaptation of Neural Architectures with Meta Learning". *International Conference on Learning Representations*.

Liu, H., Simonyan, K., and Yang, Y. (2019). "DARTS: Differentiable Architecture Search". *arXiv:1806.09055 [cs, stat]*. arXiv: 1806.09055 [cs, stat].

Nichol, A., Achiam, J., and Schulman, J. (2018). "On First-Order Meta-Learning Algorithms". *arXiv:1803.02999 [cs]*. arXiv: 1803.02999 [cs].

Rajeswaran, A., Finn, C., Kakade, S., and Levine, S. (2019). "Meta-Learning with Implicit Gradients". *arXiv:1909.04630 [cs, math, stat]*. arXiv: 1909.04630 [cs, math, stat].

Shaw, A., Wei, W., Liu, W., Song, L., and Dai, B. (2019). "Meta Architecture Search". *Advances in Neural Information Processing Systems*, pp. 11227–11237.

Zhou, H., Yang, M., Wang, J., and Pan, W. (2019). "BayesNAS: A Bayesian Approach for Neural Architecture Search". *arXiv:1905.04919 [cs, stat]*. arXiv: 1905.04919 [cs, stat].