Meta-Learning to Improve Pre-Training

Aniruddh Raghu
Massachusetts Institute of Technology
`araghu@mit.edu`

Jonathan Lorraine
University of Toronto

Simon Kornblith
Google Research

Matthew McDermott
Massachusetts Institute of Technology

David Duvenaud
Google Research & University of Toronto

Abstract

Pre-training (PT) followed by fine-tuning (FT) is an effective method for training neural networks, and has led to significant performance improvements in many domains. PT can incorporate various design choices such as task and data reweighting strategies, augmentation policies, and noise models, all of which can significantly impact the quality of representations learned. The hyperparameters introduced by these strategies therefore must be tuned appropriately. However, setting the values of these hyperparameters is challenging. Most existing methods either struggle to scale to high dimensions, are too slow and memory-intensive, or cannot be directly applied to the two-stage PT and FT learning process. In this work, we propose an efficient, gradient-based algorithm to meta-learn PT hyperparameters. We formalize the PT hyperparameter optimization problem and propose a novel method to obtain PT hyperparameter gradients by combining implicit differentiation and backpropagation through unrolled optimization. We demonstrate that our method improves predictive performance on two real-world domains. First, we optimize high-dimensional task weighting hyperparameters for multitask pre-training on protein-protein interaction graphs and improve AUROC by up to 3.9%. Second, we optimize a data augmentation neural network for self-supervised PT with SimCLR on electrocardiography data and improve AUROC by up to 1.9%.

1 Introduction

A popular and important learning paradigm for neural networks is pre-training (PT) followed by fine-tuning (FT), an approach commonly used in transfer learning [13, 59, 19, 27, 52, 11, 37, 74, 35, 28], and semi-supervised learning [9, 8, 24]. This paradigm has led to performance improvements in many domains, including computer vision [13, 59, 19, 37, 74, 35], natural language processing [27, 52, 11, 40, 34], graph structured prediction [28], and clinical machine learning [45, 46, 2, 48], and is especially helpful in settings where downstream tasks have limited training data.

The PT & FT paradigm introduces high-dimensional, complex PT hyperparameters, such as parameterized data augmentation policies used in contrastive representation learning [8, 22] or the use of task, class, or instance weighting variables in multi-task PT to avoid negative transfer [70]. These hyperparameters can significantly affect the quality of pre-trained models [8], and thus finding techniques to set their values optimally is an important area of research.

Choosing optimal PT hyperparameter values is challenging, and existing methods do not work well. Simple approaches such as random or grid search are inefficient since evaluating a hyperparameter setting requires performing the full, two-stage PT & FT optimization, which may be prohibitively computationally expensive. Gradient-free approaches, such as Bayesian optimization or evolutionary algorithms [33, 61, 47], are also limited in how well they scale to this setting. Gradient-based approaches [44, 41, 43, 42] can be used online to jointly learn hyperparameters and model parameters and can scale to millions of hyperparameters [42], but typically deal with a standard *single-stage* learning problem (e.g., normal supervised learning) and are therefore not directly applicable to the *two-stage* PT & FT learning problem.

In this work, we address this gap and propose a method for high-dimensional PT hyperparameter optimization. We first formalize a variant of the PT & FT paradigm, which we call *meta-parameterized pre-training* (Figure 1), where *meta-parameters* refer to arbitrary PT hyperparameters or parameterizable architectural choices that can be optimized to improve the learned representations.[1] We outline a meta-learning problem characterizing the optimal meta-parameters propose a gradient-based method to learn meta-parameters. Our contributions are:

- We formalize *meta-parameterized pre-training*, a variant of the pre-training and fine-tuning (PT & FT) paradigm where PT is augmented to incorporate *meta-parameters*: arbitrary structures that can be optimized to improve learned representations.
- We propose a scalable gradient-based algorithm to learn meta-parameters using a novel method to obtain meta-parameter gradients through the two-stage PT & FT process. Our gradient estimator composes a constant-memory implicit differentiation approximation for the longer PT stage and exact backpropagation through training for the shorter FT stage.
- We show that our algorithm recovers optimal meta-parameters in toy experiments on synthetic data.
- In two real-world experimental domains, we demonstrate our algorithm improves performance. Firstly, on a multitask PT benchmark over biological graph-structured data [28], using our method to optimize meta-parameters representing task weights improves performance by up to 3.9% AUROC. Secondly, for semi-supervised learning using SimCLR [8] over electrocardiography data, using our algorithm to optimize meta-parameters representing the weights of a data augmentation neural network improves performance by up to 1.9% AUROC.

2 Problem Setup and Preliminaries

In this section, we define the meta-parameterized pre-training meta-learning problem, and compare it to traditional fine-tuning and pre-training. A full glossary of notation is in Appendix B, Table 3.

Notation. Let the subscript $\bullet$ be a placeholder for either PT (pre-training) or FT (fine-tuning), $\mathcal{X} \subseteq \mathbb{R}^d$ be our input domain, $\mathcal{Y}_\bullet$ and $\hat{\mathcal{Y}}_\bullet$ be the true and predicted output spaces for some model respectively, and $\Theta, \Psi_\bullet, \Phi$ be spaces of parameters for models. We will use $f_\bullet : \mathcal{X}; (\Theta, \Psi_\bullet) \to \hat{\mathcal{Y}}_\bullet$ to refer to a parametric model, with the semicolon separating the input space from the parameter spaces. We then define $f_\bullet = f_\bullet^{(\text{head})} \circ f^{(\text{feat})}$, such that $f^{(\text{feat})}(\cdot; \boldsymbol{\theta} \in \Theta)$ is a *feature extractor* that is transferable across learning stages (e.g., pre-training to fine-tuning), and $f_\bullet^{(\text{head})}(\cdot; \boldsymbol{\psi} \in \Psi_\bullet)$ is a stage-specific *head* that is not transferable. Given a data distribution $\mathbf{x}_\bullet, \mathrm{y}_\bullet \sim \mathcal{D}_\bullet$, parametric model $f_\bullet$, and loss function $\mathcal{L}_\bullet : \hat{\mathcal{Y}}_\bullet \times \mathcal{Y}_\bullet \to \mathbb{R}$, we will also define for convenience a corresponding expected loss $L_\bullet : \Theta, \Psi_\bullet \to \mathbb{R}$ via $L_\bullet(\boldsymbol{\theta}, \boldsymbol{\psi}_\bullet; \mathcal{D}_\bullet) = \mathbb{E}_{\mathcal{D}_\bullet}\left[\mathcal{L}_\bullet(f_\bullet(\boldsymbol{x}_\bullet; \boldsymbol{\theta}, \boldsymbol{\psi}_\bullet), y_\bullet)\right]$. We also adopt the convention that the output of the argmin operator is *any* arbitrary minimum, rather than the set of possible minima, to avoid complications in notation.

2.1 Problem Formulation

Supervised Learning (Fig. 1A). In a fully-supervised setting (our *fine-tuning* domain), we are given a data distribution $\mathcal{D}_{\text{FT}}$, model f, and loss $\mathcal{L}_{\text{FT}}$. Using a *learning algorithm* Alg_{FT} (e.g., SGD) that takes as input initial parameters $\boldsymbol{\theta}_{\text{FT}}^{(0)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}$, our goal is to approximate the $\mathcal{L}_{\text{FT}}$-optimal parameters: $\boldsymbol{\theta}_{\text{FT}}^*, \boldsymbol{\psi}_{\text{FT}}^* = \text{Alg}_{\text{FT}}(\boldsymbol{\theta}_{\text{FT}}^{(0)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}) \approx \operatorname{argmin}_{\boldsymbol{\theta} \in \Theta, \boldsymbol{\psi} \in \Psi_{\text{FT}}} L_{\text{FT}}(\boldsymbol{\theta}, \boldsymbol{\psi}; \mathcal{D}_{\text{FT}})$

Pre-training (Fig. 1B). For tasks where data is scarce, we can additionally incorporate a *pre-training* step and approximate the optimal initial parameters for FT (i.e., the final pre-trained weights are used as initialization weights of the FT stage), again via an optimization algorithm Alg_{PT}: $\boldsymbol{\theta}_{\text{PT}}^* = \text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}) \approx \operatorname{argmin}_{\boldsymbol{\theta} \in \Theta} L_{\text{FT}}(\text{Alg}_{\text{FT}}(\boldsymbol{\theta}, \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}); \mathcal{D}_{\text{FT}})$. [2]

[1] We use the term *meta-parameter* since these structures do not directly affect inference of the final model after FT, but instead inform the process of learning this model (by modulating the PT process).

[2] Note that we discard the PT head $\boldsymbol{\psi}_{\text{PT}}^*$ here as only the PT feature extractor $\boldsymbol{\theta}_{\text{PT}}^*$ is transferred.

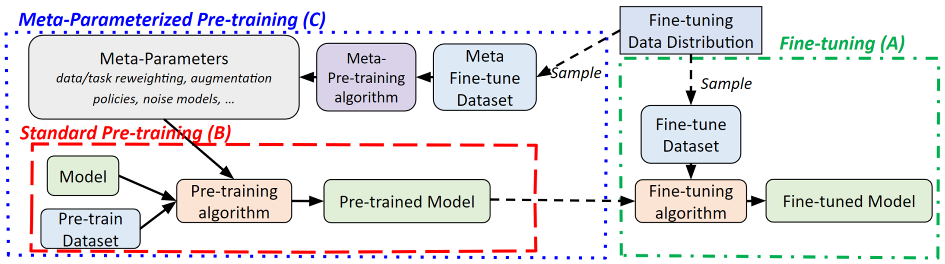

Meta-Parameterized Pre-training (C)
Meta-Parameters
data/task reweighting, augmentation policies, noise models, ...
Meta-Pre-training algorithm
Meta Fine-tune Dataset
Sample
Fine-tuning Data Distribution
Fine-tuning (A)
Sample
Fine-tune Dataset
Standard Pre-training (B)
Model
Pre-train Dataset
Pre-training algorithm
Pre-trained Model
Fine-tuning algorithm
Fine-tuned Model

Figure (1) **Meta-Parameterized Pre-Training.** A paradigm where meta-parameters — rich, potentially high dimensional structures that generalize PT hyperparameters — are incorporated in PT to improve the learned representations. Meta-parameters are optimized in a *meta-PT* phase, using data from FT task(s) in a meta-FT dataset. The FT and meta-FT datasets are (potentially overlapping) samples from the FT data distribution.

Meta-Parameterized PT (Fig. 1C). In *Meta-Parameterized PT*, we recognize that, in addition to taking as input the PT parameters $\boldsymbol{\theta}$, Alg_{PT} is itself parameterized by a set of *meta-parameters* $\boldsymbol{\phi} \in \Phi$: arbitrary, potentially high dimensional quantities that inform the structure of the algorithm directly. These could represent weighting strategies, data augmentation policies, or sampling processes. The optimal meta-parameters $\boldsymbol{\phi}^{(\text{opt})}$ are the solution to the following *meta-PT* optimization problem:

$$\boldsymbol{\phi}^{(\text{opt})} = \operatorname*{argmin}_{\boldsymbol{\phi}\in\Phi} L_{\text{FT}}\left(\text{Alg}_{\text{FT}}\left(\text{Alg}_{\text{PT}}\left(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}\right), \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}\right); \mathcal{D}_{\text{FT}}\right).$$

2.2 Example: Multitask Meta-Parameterized Pre-Training

To make our notation concrete, here we instantiate our setup for a multitask pre-training problem.

Problem: Suppose we have a multitask classification dataset, $(\mathcal{X} \times \mathcal{Y})^N$ such that $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_K$ consists of labels for K distinct tasks. Of this full set of tasks, we are interested only in a subset of M tasks, $S = \{t_1, \ldots, t_M\} \subseteq \{1, \ldots, K\}$.
Supervised FT: Under supervised FT alone, we can directly average a cross-entropy loss $\mathcal{L}_{\text{CE}}$ over *only the tasks in* S, $\mathcal{L}_{\text{FT}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{M}\sum_{j=1}^{M} \mathcal{L}_{\text{CE}}(\hat{y}^{(t_j)}, y^{(t_j)})$, and then solve this problem via SGD.
PT: If we assume that S is a *random* subset of the full set of tasks, we can introduce a PT stage over all tasks: $\mathcal{L}_{\text{PT}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{K}\sum_{i=1}^{K} \mathcal{L}_{\text{CE}}(\hat{y}^{(i)}, y^{(i)})$, followed by FT on S alone. As S is a random subset, leveraging all tasks for PT is well motivated and may improve performance.
Meta-Parameterized PT: In the case where T is not a random subset, the PT strategy described above is no longer well-motivated. However, using meta-parameterized PT, we can still effectively pre-train by introducing the meta-parameters that weight the tasks $\boldsymbol{\phi} = [\phi_1 \quad \ldots \quad \phi_K]$ and modulate the loss function $\mathcal{L}_{\text{PT}}$: $\mathcal{L}_{\text{PT}}(\hat{\boldsymbol{y}}, \boldsymbol{y}; \boldsymbol{\phi}) = \sum_{i=1}^{K} \phi_i \mathcal{L}_{CE}(\hat{y}^{(i)}, y^i)$. With optimal meta-parameters $\boldsymbol{\phi}^{(\text{opt})}$, the PT stage will leverage only that subset of tasks that best informs the final FT performance. This setting mirrors our real-world experiment in Section 5.

3 Methods: Optimizing Meta-Parameters for Two-Stage Training

We now introduce our gradient-based algorithm to optimize meta-parameters. We first describe how to efficiently approximate meta-parameter gradients through the two-stage PT and FT optimization. We then present our algorithm, and outline practical considerations when using it.

3.1 Efficient Computation of Meta-Parameter Gradients

We begin by defining:

$$g(\boldsymbol{\phi}; \boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}) = L_{\text{FT}}\Big(\underbrace{\text{Alg}_{\text{FT}}\big(\overbrace{\text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi})}^{\text{Parameter } \boldsymbol{\theta}_{\text{PT}}}, \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}\big)}_{\text{Parameters } \boldsymbol{\theta}_{\text{FT}}, \boldsymbol{\psi}_{\text{FT}}}; \mathcal{D}_{\text{FT}}\Big), \tag{1}$$

so that $\boldsymbol{\phi}^{(\text{opt})} = \text{argmin}_{\boldsymbol{\phi}\in\Phi}\, g(\boldsymbol{\phi})$.

We also define two best-response values:

$$\boldsymbol{\theta}^*_{\text{PT}}(\boldsymbol{\phi}) = \text{Alg}_{\text{PT}}(\boldsymbol{\theta}^{(0)}_{\text{PT}}, \boldsymbol{\psi}^{(0)}_{\text{PT}}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}),$$
$$\boldsymbol{\theta}^*_{\text{FT}}(\boldsymbol{\phi}), \psi^*_{\text{FT}}(\boldsymbol{\phi}) = \text{Alg}_{\text{FT}}(\boldsymbol{\theta}^*_{\text{PT}}(\boldsymbol{\phi}), \boldsymbol{\psi}^{(0)}_{\text{FT}}; \mathcal{D}_{\text{FT}}).$$

We do not explicitly include the dependence of the best responses on the initialization values for notational convenience.

With these defined, we now consider the desired gradient term, $\frac{\partial g}{\partial \boldsymbol{\phi}}$. Under our definitions, the direct partial derivatives $\frac{\partial L_{\text{FT}}}{\partial \boldsymbol{\phi}}$ and $\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\phi}}$ are zero, so $\frac{\partial g}{\partial \boldsymbol{\phi}}$ reduces to a simple expression of the chain rule:

$$\left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}'} = \underbrace{\left.\frac{\partial L_{\text{FT}}}{\partial \left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}\right|_{\boldsymbol{\theta}^*_{\text{FT}}(\boldsymbol{\phi}'), \boldsymbol{\psi}^*_{\text{FT}}(\boldsymbol{\phi}')}}_{\text{FT Loss Gradient}} \times \overbrace{\left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}^*_{\text{PT}}(\boldsymbol{\phi}')}}^{\text{FT Best Response Jacobian}} \times \underbrace{\left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}'}}_{\text{PT Best Response Jacobian}} . \quad (2)$$

The FT Loss Gradient term on the RHS of (2) is easily computed using backpropagation. Computing the other two terms is more involved, and we detail each below, beginning with the PT best response Jacobian. The full algorithm with both gradient estimation terms is provided in Algorithm 1.

PT Best Response Jacobian $\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}$. Using recent work in hyperparameter optimization with implicit differentiation [42], we re-express this term using the implicit function theorem (IFT). If we assume that $\boldsymbol{\theta}^*_{\text{PT}}(\boldsymbol{\phi}) = \text{Alg}_{\text{PT}}\left(\boldsymbol{\theta}^{(0)}_{\text{PT}}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}\right)$ is a good approximation of $\text{argmin}_{\boldsymbol{\theta} \in \Theta} L_{\text{PT}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}\right)$ (i.e., the PT model converges to $\mathcal{L}_{\text{PT}}$-optimal parameters), then under certain smoothness and regularity assumptions on the PT parameters and meta-parameters, the IFT allows us to re-express $\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}$ as:

$$\left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}'} = -\left[\frac{\partial^2 L_{\text{PT}}}{\partial \boldsymbol{\theta}_{\text{PT}}\, \partial \boldsymbol{\theta}_{\text{PT}}^\top}\right]^{-1} \times \left.\frac{\partial^2 L_{\text{PT}}}{\partial \boldsymbol{\theta}_{\text{PT}}\, \partial \boldsymbol{\phi}^\top}\right|_{\boldsymbol{\theta}^*_{\text{PT}}(\boldsymbol{\phi}'), \boldsymbol{\phi}'}, \quad (3)$$

which is the product of the inverse Hessian and a matrix of mixed partial derivatives. Following [42], the inverse can be efficiently approximated using a truncated Neumann series.

FT Best Response Jacobian $\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}$. First, note that without additional constraints on Alg_{FT}, the FT best response Jacobian may be zero. This is because L_{FT} has no functional dependence on the variable $\boldsymbol{\theta}_{\text{PT}}$ and, if we assume the convergence point $\boldsymbol{\theta}^*_{\text{FT}}$ is stable (as we did for the PT best response Jacobian), this implies that the gradient of $\boldsymbol{\theta}^*_{\text{FT}}$ with respect to $\boldsymbol{\theta}_{\text{PT}}$ would be zero. To enable effective learning, we must therefore either (1) impose restrictions on Alg_{FT} to ensure there is a dependence between the initialization point and the final loss value (e.g., proximal regularization [55]) or (2) leverage methods that do not differentiate through Alg_{FT} through convergence, as at non-converged points we will still observe nonzero L_{FT}-gradients [29, 51]. Given that the FT phase often involves shorter optimization horizons than PT, we take approach 2 here, and iteratively update $\boldsymbol{\theta}_{\text{FT}}$ for K steps. We first initialize the FT head $\boldsymbol{\psi}^{(0)}_{\text{FT}}$ and then compute:

$$\begin{aligned}
\boldsymbol{\theta}^{(0)}_{\text{FT}} &= \texttt{copy}(\boldsymbol{\theta}^*_{\text{PT}}) \qquad \text{(init with PT solution, implicitly performing stop gradient)} \\
\boldsymbol{\theta}^{(k)}_{\text{FT}}, \boldsymbol{\psi}^{(k)}_{\text{FT}} &= \left[\boldsymbol{\theta}^{(k-1)}_{\text{FT}}, \quad \boldsymbol{\psi}^{(k-1)}_{\text{FT}}\right] - \eta_{\text{FT}} \left.\frac{\partial L_{\text{FT}}}{\partial \left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}\right|_{\boldsymbol{\theta}^{(k-1)}_{\text{FT}}, \boldsymbol{\psi}^{(k-1)}_{\text{FT}}} \qquad k = 1, \ldots, K \qquad (4) \\
\boldsymbol{\theta}^*_{\text{FT}}, \boldsymbol{\psi}^*_{\text{FT}} &\approx \boldsymbol{\theta}^{(K)}_{\text{FT}}, \boldsymbol{\psi}^{(K)}_{\text{FT}},
\end{aligned}$$

and compute the gradient $\left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}^*_{\text{PT}}(\boldsymbol{\phi}')}$ by differentiating through this optimization.[3]

We can also choose to freeze the feature extractor parameters $\boldsymbol{\theta}_{\text{FT}}$ and update only the head parameters $\boldsymbol{\psi}_{\text{FT}}$ during truncated FT, and use this to obtain meta-parameter gradients. This resembles *linear evaluation*, where a linear classifier is trained on top of fixed, pre-trained feature extractors [50, 3, 63].

Together, these two approximations allow for efficient computation of meta-parameter gradients.

[3]While Equation 4 uses standard gradient descent, we could use other differentiable optimizers (e.g., Adam).

Algorithm 1 Gradient-based algorithm to learn meta-parameters. Notation defined in Appendix B, Table 3. Vector-Jacobian products (VJPs) can be efficiently computed by standard autodifferentiation.

```
1: Initialize PT parameters $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})}, \boldsymbol{\psi}_{\text{PT}}^{(\text{init})}, \boldsymbol{\psi}_{\text{FT}}^{(0)}$ and meta-parameters $\boldsymbol{\phi}^{(0)}$
2: for $n = 1, \ldots, N$ iterations do
3:     Initialize $\boldsymbol{\theta}_{\text{PT}}^{(0)} = \boldsymbol{\theta}_{\text{PT}}^{(\text{init})}$ and $\boldsymbol{\psi}_{\text{PT}}^{(0)} = \boldsymbol{\psi}_{\text{PT}}^{(\text{init})}$.
4:     for $p = 1, \ldots, P$ PT iterations do
5:         $\left[\boldsymbol{\theta}_{\text{PT}}^{(p)}, \boldsymbol{\psi}_{\text{PT}}^{(p)}\right] = \left[\boldsymbol{\theta}_{\text{PT}}^{(p-1)}, \boldsymbol{\psi}_{\text{PT}}^{(p-1)}\right] - \eta_{\text{PT}} \left.\frac{\partial L_{\text{PT}}}{\partial[\boldsymbol{\theta}_{\text{PT}}, \boldsymbol{\psi}_{\text{PT}}]}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(p-1)}, \boldsymbol{\psi}_{\text{PT}}^{(p-1)}}$
6:     end for
7:     Initialize FT encoder with PT solution: $\boldsymbol{\theta}_{\text{FT}}^{(0)} = \texttt{copy}(\boldsymbol{\theta}_{\text{PT}}^{(P)})$.
8:     Approximate $\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}$ using Eq. 4.
9:     Compute $g_1 = \left.\frac{\partial L_{\text{FT}}}{\partial[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}]}\right|_{\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}}$
10:    Compute VJP $g_2 = g_1 \left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(P)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}}$ using the unrolled learning step from line 8.
11:    Approximate VJP $\left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}} = g_2 \left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ using the IFT (Eq. 3).
12:    $\boldsymbol{\phi}^{(n)} = \boldsymbol{\phi}^{(n-1)} - \eta_{\text{V}} \left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$
13:    Update PT initialization by setting: $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})} = \boldsymbol{\theta}_{\text{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\text{PT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{PT}}^{(P)}$.
14: end for
```

3.2 Our Algorithm and Practical Considerations

By leveraging the above approximations, we obtain Algorithm 1 to optimize meta-parameters $\boldsymbol{\phi}$ online during PT & FT of the base model. Note that Alg_{PT} is explicitly written out as a sequence of gradient updates (lines 4-6 in Algorithm 1). We now discuss practical considerations when using this algorithm, with further details given in Appendix C.

(1) Access to $\mathcal{D}_{\text{FT}}$ and generalizing to new FT tasks: Solving the meta-PT problem requires availability of: the model $f_{\bullet}$, the PT data $\mathcal{D}_{\text{PT}}$, and the FT data $\mathcal{D}_{\text{FT}}$. In this work, we assume availability of the model and PT dataset, but since assuming access to the complete FT dataset at meta-PT time is more restrictive, we study two scenarios: *Full FT Access*, where all FT data that we expect to encounter is available at meta-PT time, and *Partial FT Access*, where the FT data available at meta-PT time is only a sample from a distribution of FT data that we may encounter later.

Full FT Access occurs in settings like semi-supervised learning, where we are given a large unlabelled PT dataset and a small labelled FT dataset and our goal is to achieve the best possible performance by leveraging these two fixed datasets [68, 73, 25, 24, 8, 9].

Partial FT Access occurs when our goal is to learn transferable representations: at meta-PT time, we might have limited knowledge of FT tasks or data. In evaluating this scenario, we examine generalizability to new FT tasks, given only small amounts of FT data/task availability at meta-PT time, demonstrating that even very limited FT access can be sufficient for effective meta-parameter optimization [11, 45, 56, 28].

(2) $\mathcal{D}_{\text{FT}}$ splits: In practice, we have access to finite datasets and use minibatches, rather than true data-generating processes. Following standard convention, we split $\mathcal{D}_{\text{FT}}$ into two subsets for meta-learning: $\mathcal{D}_{\text{FT}}^{(\text{tr})}$ and $\mathcal{D}_{\text{FT}}^{(\text{val})}$ (independent of any held-out $\mathcal{D}_{\text{FT}}$ testing split), and define the FT data available at meta-PT time as $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}^{(\text{tr})} \cup \mathcal{D}_{\text{FT}}^{(\text{val})}$. We use $\mathcal{D}_{\text{FT}}^{(\text{tr})}$ for the computation of $\left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(P)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}}$ and $\left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ and $\mathcal{D}_{\text{FT}}^{(\text{val})}$ for the computation of $\left.\frac{\partial L_{\text{FT}}}{\partial[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}]}\right|_{\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}}$ in Algorithm 1.

(3) Online updates: Given that PT phases often involve long optimization horizons, for computational efficiency, we update $\boldsymbol{\theta}_{\text{PT}}$ and $\boldsymbol{\psi}_{\text{PT}}$ online rather than re-initializing them at every meta-iteration (see Algorithm 1). FT phases are often shorter so we could in theory re-initialize $\boldsymbol{\psi}_{\text{FT}}$ at each meta-iteration, as is presented in Algorithm 1. However, it is more computationally efficient to also optimize this online, and we follow this approach in our experiments. A description of the algorithm with these details in Appendix C.

Note that prior work [67] has suggested that online optimization of certain hyperparameters (e.g., learning rates) using short horizons may yield suboptimal solutions. We comment on this in Appendix C, study this effect for our algorithm in synthetic experiments in Appendix E, and in real-world experiments on self-supervised learning in Appendix G, revealing it is not a significant concern.

(4) Computational tractability: Our method can scale to large encoder models and high-dimensional meta-parameters, despite the complexity of the two-stage PT & FT process. This is because: (i) meta-parameters are optimized jointly with the base model parameters; (ii) using the IFT to obtain gradients has similar time and memory complexity to one iteration of training [42]; (iii) the FT best response Jacobian can be approximated efficiently using a small number of unrolled optimization steps K, and by only unrolling the FT head of the network. In our real-world experiments (Sections 5 and 6), meta-parameterized PT has less than twice the time cost of standard PT. Further details on time and memory cost are provided in Appendices F and G.

(5) Setting optimizer parameters: Learning rates and momentum values can impact the efficacy of the algorithm. A discussion on how to set them in practice is provided in Appendix D.

4 Synthetic Experiments

We validate that our algorithm recovers optimal low and high dimensional meta-parameters in two synthetic MNIST experiments with *Full FT Access*. Further details and results are provided in Appendix E, including a study of how our method performs comparably to differentiating exactly through the entire learning process of PT & FT, without approximations.

First, we optimize low dimensional meta-parameters characterizing a data augmentation scheme. We tune a 1-D meta-parameter ϕ representing the mean of a Normal distribution $\mathcal{N}(\phi, 1^2)$ from which we sample rotation augmentations to apply to PT images. FT images undergo rotations from a Normal distribution $\mathcal{N}(\mu_{\text{FT}}, 1^2)$ with $\mu_{\text{FT}} = 90°$; we therefore expect that ϕ should converge to near μ_{FT}. Using Algorithm 1 to optimize ϕ we find that the mean error in the optimized meta-parameter over 10 different initializations is small: $7.2 \pm 1.5°$, indicating efficacy of the algorithm.

Next, we consider learning high dimensional meta-parameters that characterize a PT per-example weighting scheme. The PT dataset contains some examples that have noisy labels, and FT examples all have clean labels. The meta-parameters are the parameters of a neural network that assigns importance weights to each PT example, which is used to weight the loss on that example during PT. We use Algorithm 1 again to optimize ϕ, over 10 random initializations, finding the ratio of assigned importance weights between clean label PT examples and noisy label PT examples is greater than 10^2. This is expected since the noisy label classes may worsen the quality of the PT model and so should be down-weighted.

5 Meta-Parameterized Multitask Pre-Training for Graph Neural Networks

We consider optimizing PT task weights for a multitask PT & FT problem of predicting the presence of protein functions (multitask binary classification) given graph-structured biological data as input. We have two experimental goals: first, in the *Full FT Access* setting, where methods are given access to all FT data at PT time, we evaluate whether optimizing task weighting meta-parameters can improve predictive performance on the FT tasks. Second, motivated by how in typical transfer learning problems, new tasks or labels not available at PT time may become available at FT time, we study the *Partial FT Access* setting, investigating how our method performs when it only sees *limited* FT tasks at PT time. In both settings, our method outperforms baselines.

5.1 Problem Setup

Dataset and Task. We consider the transfer learning benchmark introduced in [28], where the prediction problem at both PT and FT is multitask binary classification: predicting the presence/absence of specific protein functions (y) given a Protein-Protein Interaction (PPI) network as input (represented as a graph $\boldsymbol{x}$). The PT dataset has pairs $\mathcal{D}_{\text{PT}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{PT}}|}$, where $y \in \{0,1\}^{5000}$ characterizes the presence/absence of 5000 particular protein functions. The FT dataset has pairs $\mathcal{D}_{\text{FT}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{FT}}|}$, where $y \in \{0,1\}^{40}$ now characterizes the presence/absence of 40 different protein functions. Further dataset details in Appendix F.

Meta-Parameterized Multitask PT. To define a meta-parameterized PT scheme, we let meta-parameters $\phi \in \mathbb{R}^{5000}$ be weights for the binary PT tasks. Then, we define a PT loss incorporating the weights: $\mathcal{L}_{\text{PT}} = \frac{1}{5000}\sum_{i=1}^{5000} 2\,\sigma(\phi_i)\,\mathcal{L}_{\text{CE}}(f_{\text{PT}}(\boldsymbol{x}; \boldsymbol{\theta}_{\text{PT}}, \boldsymbol{\psi}_{\text{PT}})_i, y_i)$,with i indexing the tasks, $\sigma(\cdot)$ representing the sigmoid function (to ensure non-negativity and clamp the range of the weights), and $\mathcal{L}_{\text{CE}}$ denoting the binary cross-entropy loss. With this loss defined, we use Algorithm 1 (with $P = 10$ PT steps and $K = 1$ truncated FT steps) to jointly learn ϕ and the feature extractor parameters $\boldsymbol{\theta}_{\text{PT}}$. For computational efficiency, we only update the FT head when computing the FT best response Jacobian and keep the feature extractor of the model fixed. We use the training and validation splits of the FT dataset $\mathcal{D}_{\text{FT}}$ proposed by the dataset creators [28] for computing the relevant gradient terms.

Baselines. Motivated by our goals, we compare with the following PT baselines:

- **No PT:** Do not perform PT (i.e., feature extractor parameters are randomly initialized).
- **Graph Supervised PT:** As explored in prior work on this domain [28], perform multitask supervised PT with $\mathcal{D}_{\text{PT}}$. This corresponds to setting all task weights to 1: $\phi_i = 1, i = 1, \ldots, 5000$.
- **CoTrain:** A common baseline that makes use of the FT data available during PT [70] (like meta-parameterized PT). We PT a model with $5000 + 40$ outputs (covering the space of PT and FT labels) jointly on both $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$. We do so by alternating gradient updates on batches sampled from each dataset in turn. Further details are in Appendix F.
- **CoTrain + PCGrad:** An extension of CoTrain, where we leverage the method PCGrad [72] to perform gradient projection and prevent destructive gradient interference between updates from $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$. Further details and variants we tried are in Appendix F.

Experimental Details. We use a standardized setup to facilitate comparisons. Following [28], all methods use the Graph Isomorphism Network architecture [69], undergo PT for 100 epochs, and FT for 50 epochs, over 5 random seeds, using early stopping based on validation set performance. During FT, we initialize a new FT network head and either FT the whole network or freeze the PT feature extractor and learn the FT head alone (Linear Evaluation [50]). We report results for the strategy that performed best (full results in the appendix). We consider two experimental scenarios: (1) *Full FT Access:* Provide methods full access to $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$ at PT time ($\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$) and evaluate on the full set of 40 FT tasks; (2) *Partial FT Access:* Limit the number of FT tasks seen at PT time, by letting $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ include only 30 of the 40 FT tasks. At FT time, models are fine-tuned on the held-out 10 tasks not in $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$. We use a 4-fold approach where we leave out 10 of the 40 FT tasks in turn, and examine performance across these 10 held-out tasks, over the folds.

5.2 Results

Key Findings. By optimizing PT task weights, meta-parameterized multitask PT improves performance on the FT problem of predicting presence/absence of protein functions given a protein-protein interaction graph as input. Performance improvements are also seen when generalizing to new FT tasks (protein functions), unseen at meta-PT time.

Table 1 presents quantitative results for the two experimental settings described. For the No PT and Graph Supervised PT baselines, we re-implement the methods from [28], obtaining improved results (full comparison in Appendix Table 5). In both full and partial FT access settings, meta-parameterized PT improves significantly on other methods, indicating that optimizing meta-parameters can improve predictive performance generally, and be effective even when new, related tasks are considered at evaluation time. Interestingly, we observe that CoTrain and CoTrain + PCGrad obtain relatively poor performance compared to other baselines; this could be because the methods overfit to the FT data during PT. Further analysis of this is presented in Appendix F.

Further experiments. In Appendix F, we study another partial FT access scenario with smaller $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$, setting $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right| = 0.5\,|\mathcal{D}_{\text{FT}}|$, and find that meta-parameterized PT again outperforms other methods. (Table 7). We also examine another meta-parameter learning baseline, namely a version of CoTrain where we optimize task weights using a traditional hyperparameter optimization algorithm [42] jointly with the main model. We find that our method outperforms this baseline also (Table 5).

Method	AUC ($\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$)	AUC ($\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ excludes tasks)
No PT	66.6 ± 0.7	65.8 ± 2.5
Graph Supervised PT	74.7 ± 0.1	74.8 ± 1.8
CoTrain	70.2 ± 0.3	69.3 ± 1.8
CoTrain + PCGrad	69.4 ± 0.2	68.1 ± 2.3
Meta-Parameterized PT	$\mathbf{78.6 \pm 0.1}$	$\mathbf{77.0 \pm 1.3}$

Table (1) **Meta-Parameterized PT improves predictive performance over baselines.** Table showing mean AUC and standard error for two evaluation settings. When provided all FT data at PT time (first results column), meta-parameterized PT significantly improves predictive performance. In a more challenging setting when $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ excludes FT tasks (10 of the 40 available tasks are held-out), evaluating mean AUC/standard error across four folds with each set of 10 FT tasks held out in turn, meta-parameterized PT again obtains the best performance: it is effective even with partial information about the downstream FT tasks.

Analysis of learned structures. In Appendix F, we conduct further analysis and study the effect of various PT strategies on the pre-trained representations (Figure 3), finding intuitive patterns of similarity between different methods. We also examine the learned task weights (Figure 4), and examine performance on a per-FT task basis with/without meta-parameterized PT (Figure 5), finding little evidence of negative transfer.

6 Meta-Parameterized SimCLR for Semi-Supervised Learning with ECGs

We now explore a second real-world application of our method: optimizing a data augmentation policy for self-supervised PT with SimCLR [8, 9] on electrocardiograms (ECGs). SimCLR is a popular self-supervised PT method that leverages data augmentations to define a contrastive PT objective (details in Appendix G.1). The choice/strength of the augmentations used significantly impacts the effectiveness of the algorithm [8]. In settings where relevant augmentations are known (e.g., natural images), SimCLR is readily applicable; however, for ECGs, effective augmentations are less clear, motivating the use of our algorithm to optimize the augmentation pipeline.

We have two experimental goals. Firstly, we examine the typical semi-supervised learning setting of *Full FT Access*: we explore whether optimizing the augmentations in SimCLR PT can improve performance on the supervised FT task of detecting pathologies from ECGs, given access to all FT data at meta-PT time. Secondly, to study the data efficiency of our method, we consider the *Partial FT Access* setting and explore performance given access to limited FT data at meta-PT time. We find that our method improves the performance of SimCLR, and that it is effective even with very limited amounts of FT data provided at meta-PT time.

6.1 Problem Setup

Dataset and Task. We construct a semi-supervised learning (SSL) problem using PTB-XL [64, 20], an open-source dataset of electrocardiogram (ECG) data. Let the model input at both PT and FT time be denoted by $\boldsymbol{x}$, which represents a 12-lead (or channel) ECG sampled at 100 Hz for 10 seconds resulting in a 1000×12 signal. Our goal is to pre-train a model f_{PT} on an unlabeled PT dataset of ECGs $\mathcal{D}_{\text{PT}} = \{\boldsymbol{x}_i\}_{i=1}^{|\mathcal{D}_{\text{PT}}|}$ using SimCLR PT [8], and then fine-tune it on the labeled FT dataset $\mathcal{D}_{\text{FT}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{FT}}|}$, where the FT labels $y \in \{0, 1\}^5$ encode whether the signal contains certain features indicative of particular diseases/pathologies. Further dataset details in Appendix G.

ECG Data Augmentations. To augment each ECG for SimCLR (example in Appendix G, Figure 6), we apply three transformations in turn (based on prior work in time series augmentation [30, 66]):

1. **Random cropping:** A randomly selected portion of the signal is zeroed out.
2. **Random jittering:** IID Gaussian noise is added to the signal.
3. **Random temporal warping:** The signal is warped with a random, diffeomorphic temporal transformation. This is formed by sampling from a zero mean, fixed variance Gaussian at each temporal location in the signal to obtain a velocity field, and then integrating and smoothing (following [4, 5]) to generate a temporal displacement field, which is applied to the signal.

	Test AUC at different FT dataset sizes $\|\mathcal{D}_{\text{FT}}\|$				
FT dataset size $\|\mathcal{D}_{\text{FT}}\|$	100	250	500	1000	2500
No PT	71.5 ± 0.7	76.1 ± 0.3	78.7 ± 0.3	82.0 ± 0.2	84.5 ± 0.2
SimCLR	74.6 ± 0.4	76.5 ± 0.3	79.8 ± 0.3	82.2 ± 0.3	85.8 ± 0.1
Meta-Parameterized SimCLR	$\mathbf{76.1 \pm 0.5}$	$\mathbf{77.8 \pm 0.4}$	$\mathbf{81.7 \pm 0.2}$	$\mathbf{84.0 \pm 0.3}$	$\mathbf{86.7 \pm 0.1}$

Table (2) **Meta-Parameterized SimCLR obtains improved semi-supervised learning performance.** Table showing mean AUC/standard error over seeds across 5 FT binary classification tasks for baselines and meta-parameterized SimCLR at different sizes of $\mathcal{D}_{\text{FT}}$, with $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$. We observe improvements in performance with meta-parameterized SimCLR, which optimizes the augmentation pipeline.

Meta-Parameterized SimCLR. To construct a meta-parameterized SimCLR PT scheme, we instantiate meta-parameters $\boldsymbol{\phi}$ as the weights of a neural network $w(\boldsymbol{x}; \boldsymbol{\phi})$ that takes in an input signal and outputs the warp strength: the variance of the Gaussian that is used to obtain the velocity field for temporal warping. This parameterization permits signals to be warped more/less aggressively depending on their individual structure. With this definition, the SimCLR PT loss is directly a function of the meta-parameters, and we can use Algorithm 1 (with $P = 10$ PT steps and $K = 1$ truncated FT steps) to jointly learn ϕ and the feature extractor parameters $\boldsymbol{\theta}_{\text{PT}}$. For computational efficiency, we only update the FT head when computing the FT best response Jacobian and keep the feature extractor of the model fixed. We use the training and validation splits of the FT dataset $\mathcal{D}_{\text{FT}}$ proposed by the dataset creators [64] for computing the relevant gradient terms.

Baselines. Our experimental goals suggest the following PT baselines:

- **No PT:** Do not perform PT (i.e., feature extractor parameters are randomly initialized).
- **SimCLR:** Pre-train a model using SimCLR with the above three augmentations *without* learning per-example temporal warping strengths.

Experimental Details. We standardize the experimental setup to facilitate comparisons. All methods use a 1D CNN based on a ResNet-18 [23] architecture. The temporal warping network $w(\boldsymbol{x}; \phi)$ is a four layer 1D CNN. SimCLR PT takes place for 50 epochs for all methods, over three PT seeds. At evaluation time, for all methods, we initialize a new FT network head over the PT network feature extractor and FT the whole network for 200 epochs, over five FT seeds. Validation set AUC is used for early stopping. We consider two experimental settings: (1) *Full FT Access*, standard SSL: consider different sizes of the labelled FT dataset $\mathcal{D}_{\text{FT}}$ and make all the FT data available at meta-PT time, $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$; and (2) *Partial FT Access*, examining data efficiency of our algorithm: SSL when only limited FT data is available at meta-PT time: $\mathcal{D}_{\text{FT}}^{(\text{Meta})} \subseteq \mathcal{D}_{\text{FT}}$. We evaluate performance across the 5 binary classification tasks in both settings. Further details are provided in Appendix G.

6.2 Results

Key Findings. By optimizing the data augmentation policy used in SimCLR PT, meta-parameterized SimCLR improves performance on the FT problem of detecting pathologies from ECG data. Even a small amount of FT data provided at meta-PT time can lead to improved FT performance.

Table 2 shows results for the *Full FT Access* setting, $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$: mean AUC/standard error over seeds across the 5 FT binary classification tasks at different sizes of $\mathcal{D}_{\text{FT}}$. We observe that meta-parameterized SimCLR improves on other baselines in all settings. Note that while these gains are modest, they are obtained with simple augmentation policies; our method may yield further improvements if applied to policies with more scope to specialize the augmentations.

Next, we consider the *Partial FT Access* scenario where $\mathcal{D}_{\text{FT}}^{(\text{Meta})} \subseteq \mathcal{D}_{\text{FT}}$, which is relevant when we only have a small amount of FT data at meta-PT time. Fixing $|\mathcal{D}_{\text{FT}}| = 500$, we find that with $|\mathcal{D}_{\text{FT}}^{(\text{Meta})}|$ as small as 50, we obtain test AUC of 81.3 ± 0.5, compared to 79.8 ± 0.3 with no optimization of augmentations: this shows that even small $|\mathcal{D}_{\text{FT}}^{(\text{Meta})}|$ appear to be sufficient for meta-parameter learning. Further results showing performance curves varying $|\mathcal{D}_{\text{FT}}^{(\text{Meta})}|$ are in Appendix G.

Further experiments. In Appendix G, we study other aspects of our method on this domain, including: (1) Exploring different values of K, the number of FT steps differentiated through when obtaining meta-parameter gradients; and (2) Examining a meta-parameter learning baseline where augmentations are optimized for supervised learning, using the method in [42], and then applied to semi-supervised learning (to compare how optimizing augmentations for supervised learning compares to optimizing them for semi-supervised learning). We find that our method is not very sensitive to the value of K (provided $K > 0$), and that it outperforms this additional baseline.

7 Related Work

Gradient-based hyperparameter optimization (HO): Gradient-based HO roughly falls into two camps. The simpler and less scalable approach differentiates through training [12, 44]. The other approach assumes that optimization reaches a fixed point, and approximates the best-response Jacobian [7, 41, 43, 42]. Neither of these approaches can be straightforwardly applied to scalably differentiate through two stages of optimization (PT & FT). Direct differentiation through both stages would be too memory-intensive. Approximating the best-response Jacobian using the IFT as in [42] twice is feasible, but requires changing the FT objective to include a proximal term [55], and tuning two sets of interacting approximations. Instead, we compose a constant-memory IFT approximation for the lengthy PT stage with an exact backprop-through-training for the shorter FT stage.

Applications of Nested Optimization: Many prior works frame learning as nested optimization, including few-shot learning [16, 1, 17, 55, 21, 58, 53, 75, 31, 38], neural network teaching [14, 15, 62, 54], learning data augmentation and reweighting strategies [32, 22, 57, 60, 29], and auxiliary task learning [49, 51, 39]. The majority of this work studies nested optimization in the standard one-stage supervised learning paradigm, unlike our setting: the two-stage PT & FT problem. The most closely related works to ours are [70], where PT task weights are learned for a multitask PT problem using electronic health record data, and [71], where a masking policy is learned for masked language modelling PT. In contrast to our work, which introduces the more general framing of meta-parameter optimization, [70] and [71] are focused only on specific instantiations of meta-parameters as task weights and masking policies. The learning algorithms in these works either: differentiate directly through truncated PT & FT [71] (which may not be scalable to longer PT/large encoder models), or leverage extensive first-order approximations [70], unlike our more generally applicable approach.

8 Scope and Limitations

Our gradient-based algorithm applies in situations where we want to optimize (potentially high-dimensional) PT hyperparameters, or *meta-parameters*, and have access to a model, PT data, and FT data. We demonstrated that even limited FT data availability can be sufficient to guide meta-parameter learning; however, our method would not apply when no FT data at all is available at meta-PT time, or if the model or PT data were not available. Our algorithm requires meta-parameters to be differentiable, and cannot directly be used to optimize meta-parameters that do not affect the PT optimization landscape (e.g., PT learning rates).

9 Conclusion

In this work, we studied the problem of optimizing high-dimensional pre-training (PT) hyperparameters, or *meta-parameters*. We formalized *Meta-Parameterized Pre-Training*, a variant of standard PT incorporating these meta-parameters, and proposed a gradient-based algorithm to efficiently learn meta-parameters by approximately differentiating through the two-stage PT & FT learning process. In experiments, we used our algorithm to improve predictive performance on two real-world PT tasks: multitask PT with graph structured data [28], and self-supervised contrastive PT on electrocardiogram signals using SimCLR [8]. Future work could apply our method to learn other potential instantiations of meta-parameters, such as learned auxiliary tasks and noise models.

Societal Impact. Our contribution in this work is methodological, namely a new algorithm to optimize high-dimensional pre-training hyperparameters. We do not expect there to be direct negative societal impacts of this contribution. However, to evaluate our method, we considered an experimental domain using healthcare data. Given the high risk nature of this domain, before use in real-world settings, the method should be validated in retrospective and prospective studies. This is to detect any failure modes and identify potential harm that may come from deploying it.

Acknowledgements

This work was supported in part by funds from Quanta Computer, Inc. The authors thank the members of the Clinical and Applied Machine Learning group at MIT and Paul Vicol for helpful feedback.

References

[1] A. Antoniou, H. Edwards, and A. Storkey. How to train your MAML. *arXiv preprint arXiv:1810.09502*, 2018.

[2] S. Azizi, B. Mustafa, F. Ryan, Z. Beaver, J. Freyberg, J. Deaton, A. Loh, A. Karthikesalingam, S. Kornblith, T. Chen, et al. Big self-supervised models advance medical image classification. *arXiv preprint arXiv:2101.05224*, 2021.

[3] P. Bachman, R. D. Hjelm, and W. Buchwalter. Learning representations by maximizing mutual information across views. *arXiv preprint arXiv:1906.00910*, 2019.

[4] G. Balakrishnan, A. Zhao, M. Sabuncu, J. Guttag, and A. V. Dalca. An unsupervised learning model for deformable medical image registration. *CVPR: Computer Vision and Pattern Recognition*, pages 9252–9260, 2018.

[5] G. Balakrishnan, A. Zhao, M. Sabuncu, J. Guttag, and A. V. Dalca. Voxelmorph: A learning framework for deformable medical image registration. *IEEE TMI: Transactions on Medical Imaging*, 38:1788–1800, 2019.

[6] A. Beatson and R. P. Adams. Efficient optimization of loops and limits with randomized telescoping sums. In *International Conference on Machine Learning*, pages 534–543. PMLR, 2019.

[7] Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8): 1889–1900, 2000.

[8] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[9] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*, 2020.

[10] C. Cortes, M. Mohri, and A. Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] J. Domke. Generic methods for optimization-based modeling. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012*, JMLR Proceedings. JMLR.org, 2012. URL `http://proceedings.mlr.press/v22/domke12.html`.

[13] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[14] Y. Fan, F. Tian, T. Qin, X.-Y. Li, and T.-Y. Liu. Learning to teach. *arXiv preprint arXiv:1805.03643*, 2018.

[15] Y. Fan, Y. Xia, L. Wu, S. Xie, W. Liu, J. Bian, T. Qin, X.-Y. Li, and T.-Y. Liu. Learning to teach with deep interactions. *arXiv preprint arXiv:2007.04649*, 2020.

[16] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

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

[18] G. Gidel, R. A. Hemmat, M. Pezeshki, R. Le Priol, G. Huang, S. Lacoste-Julien, and I. Mitliagkas. Negative momentum for improved game dynamics. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1802–1811. PMLR, 2019.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[20] A. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation*, 101 23:E215–20, 2000.

[21] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

[22] R. Hataya, J. Zdenek, K. Yoshizoe, and H. Nakayama. Meta approach to data augmentation optimization. *arXiv preprint arXiv:2006.07965*, 2020.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[24] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[25] O. Henaff. Data-efficient image recognition with contrastive predictive coding. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4182–4192. PMLR, 13–18 Jul 2020. URL `http://proceedings.mlr.press/v119/henaff20a.html`.

[26] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bklr3j0cKX`.

[27] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[28] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJlWWJSFDH`.

[29] Z. Hu, B. Tan, R. R. Salakhutdinov, T. M. Mitchell, and E. P. Xing. Learning data manipulation for augmentation and weighting. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[30] B. K. Iwana and S. Uchida. An empirical survey of data augmentation for time series classification with neural networks. *arXiv preprint arXiv:2007.15951*, 2020.

[31] K. Javed and M. White. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588*, 2019.

[32] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2304–2313, 2018.

[33] K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, B. Poczos, and E. P. Xing. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *arXiv preprint arXiv:1903.06694*, 2019.

[34] M. Kang, M. Han, and S. J. Hwang. Neural mask generator: Learning to generate adaptive word maskings for language model adaptation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6102–6120, 2020.

[35] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6(2):8, 2019.

[36] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[37] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.

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

[39] S. Liu, A. Davison, and E. Johns. Self-supervised generalisation with meta auxiliary learning. In *Advances in Neural Information Processing Systems*, pages 1679–1689, 2019.

[40] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[41] J. Lorraine and D. Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

[42] J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

[43] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations*, 2019.

[44] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[45] M. McDermott, B. Nestor, E. Kim, W. Zhang, A. Goldenberg, P. Szolovits, and M. Ghassemi. A comprehensive evaluation of multi-task learning and multi-task pre-training on ehr time-series data. *arXiv preprint arXiv:2007.10185*, 2020.

[46] M. McDermott, B. Nestor, E. Kim, W. Zhang, A. Goldenberg, P. Szolovits, and M. Ghassemi. A comprehensive ehr timeseries pre-training benchmark. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 257–278, 2021.

[47] J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404, 1975.

[48] B. Mustafa, A. Loh, J. Freyberg, P. MacWilliams, A. Karthikesalingam, N. Houlsby, and V. Natarajan. Supervised transfer learning at scale for medical imaging. *arXiv preprint arXiv:2101.05913*, 2021.

[49] A. Navon, I. Achituve, H. Maron, G. Chechik, and E. Fetaya. Auxiliary learning by implicit differentiation. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=n7wIfYPdVet`.

[50] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[51] H. Pham, Z. Dai, Q. Xie, M.-T. Luong, and Q. V. Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020.

[52] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[53] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2019.

[54] A. Raghu, M. Raghu, S. Kornblith, D. Duvenaud, and G. Hinton. Teaching with commentaries. *arXiv preprint arXiv:2011.03037*, 2020.

[55] A. Rajeswaran, C. Finn, S. Kakade, and S. Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 2019.

[56] R. Rao, N. Bhattacharya, N. Thomas, Y. Duan, X. Chen, J. Canny, P. Abbeel, and Y. S. Song. Evaluating protein transfer learning with tape. *Advances in Neural Information Processing Systems*, 32:9689, 2019.

[57] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, pages 4331–4340, 2018.

[58] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[59] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[60] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, pages 1919–1930, 2019.

[61] J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[62] F. P. Such, A. Rawal, J. Lehman, K. Stanley, and J. Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *International Conference on Machine Learning*, pages 9206–9216. PMLR, 2020.

[63] Y. Tian, D. Krishnan, and P. Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[64] P. Wagner, N. Strodthoff, R.-D. Bousseljot, D. Kreiseler, F. I. Lunze, W. Samek, and T. Schaeffter. PTB-XL, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):1–15, 2020.

[65] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2015.

[66] Q. Wen, L. Sun, X. Song, J. Gao, X. Wang, and H. Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[67] Y. Wu, M. Ren, R. Liao, and R. Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[68] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[69] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

[70] Y. Xue, N. Du, A. Mottram, M. Seneviratne, and A. M. Dai. Learning to select best forecast tasks for clinical outcome prediction. *Advances in Neural Information Processing Systems*, 33, 2020.

[71] Q. Ye, B. Z. Li, S. Wang, B. Bolte, H. Ma, W. tau Yih, X. Ren, and M. Khabsa. On the influence of masking policies in intermediate pre-training. *arXiv preprint arXiv:1801.06146*, 2021.

[72] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[73] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1476–1485, 2019.

[74] X. Zhai, J. Puigcerver, A. Kolesnikov, P. Ruyssen, C. Riquelme, M. Lucic, J. Djolonga, A. S. Pinto, M. Neumann, A. Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[75] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.

A Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 8 for a discussion of scope and limitations, and Section 3, Practical Considerations for other important comments. Further assumptions about the method in Appendix C.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 9.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Main code for our method is provided in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See experiments in Sections 4,5, and 6. Further details in Appendix, Sections E, F, G.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix, Sections E, F, G.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

B Notation and Acronyms

PT	Pre-training
FT	Fine-tuning
AUROC (AUC)	Area Under Receiver-Operator Characteristic
$\bullet$	Placeholder for either PT or FT
$\mathcal{X}$	Input domain to models
$\boldsymbol{x}$	Model input
$\mathcal{Y}_\bullet$	True output space (e.g., space of labels)
y	Label
$\hat{\mathcal{Y}}_\bullet$	Prediction output space of a model
$\hat{y}$	Predicted Label
Θ	Parameter space for feature extractors of models
$\boldsymbol{\theta}$	Feature extractor parameters
$\Psi_\bullet$	Parameter space for prediction head of model (output layer)
$\boldsymbol{\psi}_\bullet$	Head parameters
Φ	Space of meta-parameters
ϕ	Meta-parameters
$f_\bullet : \mathcal{X}; \Theta, \Psi_\bullet \to \hat{\mathcal{Y}}_\bullet$	General parameteric model satisfying compositional structure $f_\bullet = f_\bullet^{(\text{head})} \circ f^{(\text{feat})}$
$f^{(\text{feat})}(\cdot; \boldsymbol{\theta} \in \Theta)$	Feature extractor that is transferable across learning stages (e.g., PT to FT)
$f_\bullet^{(\text{head})}(\cdot; \boldsymbol{\psi} \in \Psi_\bullet)$	Output 'head' of a model that is stage-specific and not transferable.
$\mathcal{D}_\bullet$	General data distribution or dataset
$\mathcal{L}_\bullet : \hat{\mathcal{Y}}_\bullet \times \mathcal{Y}_\bullet \to \mathbb{R}$	Loss function
$\mathcal{L}_{\text{CE}}$	Example loss function: cross-entropy
$L_\bullet(\boldsymbol{\theta}, \boldsymbol{\psi}_\bullet; \mathcal{D}_\bullet)$	Expected loss over a data distribution $\mathbb{E}_{\mathcal{D}_\bullet}\left[\mathcal{L}_\bullet(f_\bullet(\boldsymbol{x}_\bullet; \boldsymbol{\theta}, \boldsymbol{\psi}_\bullet), y_\bullet)\right]$.
$\text{Alg}_\bullet$	Learning algorithm used for optimization (e.g., stochastic gradient descent)
$g(\phi)$	Meta-parameter optimization objective $L_{\text{FT}}\left(\text{Alg}_{\text{FT}}\left(\text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}), \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}\right); \mathcal{D}_{\text{FT}}\right)$
$\phi^{(\text{opt})}$	Optimal meta-parameters satisfying $\boldsymbol{\phi}^{(\text{opt})} = \text{argmin}_{\boldsymbol{\phi} \in \Phi}\, g(\boldsymbol{\phi})$
$\boldsymbol{\theta}_{\text{PT}}^*(\boldsymbol{\phi})$	PT best response values satisfying $\boldsymbol{\theta}_{\text{PT}}^*(\boldsymbol{\phi}) = \text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi})$
$\boldsymbol{\theta}_{\text{FT}}^*(\boldsymbol{\phi}), \psi_{\text{FT}}^*(\boldsymbol{\phi})$	FT best response values satisfying $\boldsymbol{\theta}_{\text{FT}}^*(\boldsymbol{\phi}), \psi_{\text{FT}}^*(\boldsymbol{\phi}) = \text{Alg}_{\text{FT}}(\boldsymbol{\theta}_{\text{PT}}^*(\boldsymbol{\phi}), \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}})$
$\frac{\partial g}{\partial \phi}$	Gradient w.r.t. meta-parameters, which we compute for gradient-based optimization of ϕ
$\left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]$	Shorthand to representation concatenation of parameter vectors.
$\frac{\partial L_{\text{FT}}}{\partial\left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}$	FT loss gradient: first term in meta-parameter gradient.
$\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}$	FT best response Jacobian: second term in meta-parameter gradient.
$\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}$	PT best response Jacobian: third term in meta-parameter gradient.
K	Number of steps we unroll in FT to compute FT best response Jacobian.
P	Number of PT steps before each meta-parameter update.
$\texttt{copy}(\theta)$	Make a copy of the parameters θ such that gradients do not flow through (like a stop-gradient).
$\mathcal{D}_{\text{FT}}^{(\text{tr})}$	Training split of the FT data set, used during meta-parameter learning for updating the FT parameters.
$\mathcal{D}_{\text{FT}}^{(\text{val})}$	Validation split of the FT data set, used during meta-parameter learning for optimizing meta-parameters.
$\mathcal{D}_{\text{FT}}^{(\text{Meta})}$	FT data available at PT time for meta-parameter learning. We have that $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}^{(\text{tr})} \cup \mathcal{D}_{\text{FT}}^{(\text{val})} \subseteq \mathcal{D}_{\text{FT}}^{(\text{all})}$.
IFT	Implicit Function Theorem
GIN	Graph Isomorphism Network
ECG	Electrocardiogram
η_{PT}	learning rate for PT
η_{FT}	learning rate for FT
η_{V}	learning rate for meta parameters

Table (3) Notation

C Our Algorithm: Further Details

Algorithm 2 Gradient-based algorithm to learn meta-parameters, incorporating other practical details not present in the main paper description. Notation defined in Table 3. Note that vector-Jacobian products (VJPs) can be efficiently computed by standard autodifferentiation.

```
1:  Initialize PT parameters $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})}, \boldsymbol{\psi}_{\text{PT}}^{(\text{init})}, \boldsymbol{\psi}_{\text{FT}}^{(\text{init})}$ and meta-parameters $\boldsymbol{\phi}^{(0)}$
2:  for $n = 1, \ldots, N$ iterations do
3:      Initialize $\boldsymbol{\theta}_{\text{PT}}^{(0)} = \boldsymbol{\theta}_{\text{PT}}^{(\text{init})}$ and $\boldsymbol{\psi}_{\text{PT}}^{(0)} = \boldsymbol{\psi}_{\text{PT}}^{(\text{init})}$.
4:      for $p = 1, \ldots, P$ PT iterations do
5:          $\left[\boldsymbol{\theta}_{\text{PT}}^{(p)}, \boldsymbol{\psi}_{\text{PT}}^{(p)}\right] = \left[\boldsymbol{\theta}_{\text{PT}}^{(p-1)}, \boldsymbol{\psi}_{\text{PT}}^{(p-1)}\right] - \eta_{\text{PT}} \left.\frac{\partial L_{\text{PT}}}{\partial[\boldsymbol{\theta}_{\text{PT}}, \boldsymbol{\psi}_{\text{PT}}]}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(p-1)}, \boldsymbol{\psi}_{\text{PT}}^{(p-1)}}$    # Unrolled step of $\text{Alg}_{\text{PT}}$
6:      end for
7:      if $n < N_{\text{warmup}}$ then
8:          Update PT initialization by setting: $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})} = \boldsymbol{\theta}_{\text{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\text{PT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{PT}}^{(P)}$
9:          Skip meta-parameter update and continue
10:     end if
11:     Initialize FT parameters $\boldsymbol{\psi}_{\text{FT}}^{(0)} = \boldsymbol{\psi}_{\text{FT}}^{(\text{init})}$ and $\boldsymbol{\theta}_{\text{FT}}^{(0)} = \texttt{copy}(\boldsymbol{\theta}_{\text{PT}}^{(P)})$.
12:     Approximate $\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}$ using (4), with $\mathcal{D}_{\text{FT}}^{(\text{tr})}$.
13:     Compute $g_1 = \left.\frac{\partial L_{\text{FT}}}{\partial[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}]}\right|_{\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}}$, using $\mathcal{D}_{\text{FT}}^{(\text{val})}$.    # FT Loss gradient
14:     Compute VJP $g_2 = g_1 \left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(P)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}}$ using the unrolled learning step from line 12, and $\mathcal{D}_{\text{FT}}^{(\text{tr})}$.
15:     Approximate VJP $\left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}} = g_2 \left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ using IFT (3).
16:     $\boldsymbol{\phi}^{(n)} = \boldsymbol{\phi}^{(n-1)} - \eta_{\text{V}} \left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$    # Update meta-parameters
17:     Update PT initialization by setting: $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})} = \boldsymbol{\theta}_{\text{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\text{PT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{PT}}^{(P)}$.
18:     Update FT initialization by setting: $\boldsymbol{\psi}_{\text{FT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{FT}}^{*}$.
19: end for
```

We include a more detailed algorithm in Algorithm 2 reflecting certain extra details that were excluded in the main text due to space restrictions. We discuss some of these details here.

$\mathcal{D}_{\text{FT}}$ splits. In practice, we have access to finite datasets and use minibatches, rather than data generative processes. Following standard convention, we split $\mathcal{D}_{\text{FT}}$ into two subsets for meta-learning: $\mathcal{D}_{\text{FT}}^{(\text{tr})}$ and $\mathcal{D}_{\text{FT}}^{(\text{val})}$ (independent of any held-out $\mathcal{D}_{\text{FT}}$ testing split), and define the FT data available at meta-PT time as $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}^{(\text{tr})} \cup \mathcal{D}_{\text{FT}}^{(\text{val})}$. We use $\mathcal{D}_{\text{FT}}^{(\text{tr})}$ for the computation of $\left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(P)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}}$ and $\left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ and $\mathcal{D}_{\text{FT}}^{(\text{val})}$ for the computation of $\left.\frac{\partial L_{\text{FT}}}{\partial[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}]}\right|_{\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}}$ in Algorithm 2. The description in Algorithm 2 includes details of the different datasets used for different computations.

Online updates. Given that PT phases often involve long optimization horizons, we update $\boldsymbol{\theta}_{\text{PT}}$ and $\boldsymbol{\psi}_{\text{PT}}$ online, jointly with ϕ, rather than re-initializing them at every meta-iteration (see Algorithm 2).

FT phases are typically shorter so we could in theory re-initialize $\boldsymbol{\psi}_{\text{FT}}$ at each meta-iteration, as is presented in the main text, Algorithm 1. However, for further computational and memory efficiency, in our experiments, we also optimize these parameters online. For $\boldsymbol{\psi}_{\text{FT}}$ this makes each meta-iteration resemble a "warm-start" to the FT problem. In the updated description in Algorithm 2, we update the notation for the FT head to describe this.

See below for a discussion on optimization horizons and considerations when jointly optimizing meta-parameters with PT and FT parameters.

Notational clarification: vector concatenation. We use the notation: $[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}]$ to represent concatenation of the two vectors $\boldsymbol{\theta}_{\text{FT}}$ and $\boldsymbol{\psi}_{\text{FT}}$. The output of Alg_{FT} contains two parameter vectors, and these are implicitly concatenated to make sure that dimensionalities agree in the algorithm.

Warmup iterations. We can optionally include *warmup iterations* where we optimize the PT parameters and do not perform updates to the meta-parameters. This is to ensure that the PT parameters are a reasonable approximation of $\mathcal{L}_{\text{PT}}$-optimal parameters. The description in the algorithm is updated to reflect this, with the N_{warmup} reflecting the number of warmup iterations.

On optimization horizons. Prior work [67] has suggested that online optimization of certain hyperparameters (such as learning rates) using short horizons may yield suboptimal solutions. This is known as the short-horizon bias (SHB) problem. We now discuss this concern further in the context of our algorithm.

- **What is the short-horizon bias (SHB) problem?** SHB is understood to be a special case of the bias induced by truncating telescoping sums for optimization parameters. The effects of the truncation can be pronounced with optimization parameters [67], but there exist methods like [6] to deal with these if they occur.
- **Do we expect this to be a concern in our setting?** There are two hypergradients in our system that could suffer from bias: the PT hypergradients and the FT hypergradients. In both cases, the impact from biased hypergradients appears to be minimal. We will argue for this claim through each hypergradient term separately.

  PT Hypergradient: The PT hypergradient does not suffer from the short-horizon bias because the PT model is expected to have approximately converged at each hyperparameter update. This is not only a requirement of the implicit function theorem and the algorithm from [42] to apply, but also is directly enforced in our system through the use of online-updates and a warmup period (see Algorithm 2).

  FT Hypergradient: For the gradient through FT, we acknowledge that differentiating through only one step could, in theory, produce biased hypergradients. However, several prior works on meta-learning various structures similar to what we consider [51, 54, 42, 49, 29] did not observe significant bias. Therefore, from an empirical standpoint, this bias is not necessarily expected to be a significant issue.

  As seen in our experimental results, we also observe improved experimental results by setting $K = 1$ in our algorithm, suggesting minimal SHB impact. To study this issue further, we include experiments comparing to full backpropagation through PT and FT in synthetic experiments (Appendix E), and compare different values of K in our semi-supervised learning experiments (Appendix G).
- **Why might SHB not be a concern with the hyperparameters we consider?** As stated, the SHB issue has mainly been observed in the context of optimization hyperparameters such as the learning rate. This could be because the learning rate directly affects the rate at which we approach the critical point, but it does not directly change the critical point. As seen in the analysis in [67], optimizing the LR with short rollouts results in (far too aggressively) decaying the step size to decrease variance and converge faster. In contrast, other hyperparameters, like weight decay or augmentations, directly change the fixed point that we are converging to (as opposed to just the rate). In setups where the hyperparameters directly affect the fixed point, SHB has not been observed — for example, see [57, 43]. These works do online, limited horizon optimization of hyperparameters directly affecting the critical point.

D Practical Heuristics for Tuning Optimizer Parameters

The optimization parameters used in nested optimization can be crucial for success. In our synthetic experiments in Section 4 we were able to use default optimizer selections; however these settings may not work in practice for all domains (as seen in our real-world experiments). Here, we list some basic guidelines a practitioner can iterate through to debug meta-parameter optimization.

Step 1: What to do if the meta-parameters are changing wildly? First, decrease the learning rate for the meta-parameters. Momentum parameters can be dangerous – see [18]; using an optimizer without momentum may work better in some situations. If the meta-parameters begin oscillating later into training, try decreasing momentum.

Step 2: What to do if the pre-training parameters are changing wildly? First, make sure the meta-parameters are not moving around rapidly. Once the meta-parameters are stable, you should be able to decrease the learning rate of the pre-training optimizer until convergence.

Step 3: What to do if the meta-parameters are not changing? First, make sure that your pre-training parameters are finding good solutions by examining the pre-training optimization and optimizer settings. Next, make sure that the IFT is giving a good approximation for the pre-training response. You should begin with 1 Neumann term (or an identity inverse-Hessian approximation), because this often works well; see [42]. If 1 Neumann term works, you can try adding more until they offer no benefit. Next, make sure that differentiation through optimization is giving a reasonable gradient. If the unrolled optimizer is diverging, this will not give us useful gradients, so we must make sure these FT parameters converge. After you verify these components, try increasing the meta-parameter learning rate.

E Synthetic Experiments: Further Details

We discuss further details on the synthetic experiments. All experiments in this section were run on Google Colab, using the default GPU backend.

E.1 Meta-Parameterized Data Augmentation

Here, we have additional details for the data augmentation synthetic experiments in Section 4.

Dataset. Both PT and FT tasks are supervised MNIST digit classification (i.e., a 10-class classification problem). Our pre-training dataset is 3000 randomly-sampled MNIST data points. The fine-tuning training and validation sets are a distinct set of 3000 randomly-sampled MNIST data points augmented with a rotational degree drawn from $\mathcal{N}(\mu, \sigma^2)$ for some mean μ and standard deviation σ.

In the main text (Section 4) we studied the situation where the FT rotation distribution was $\mathcal{N}(\mu, 1^2)$, $\mu = 90°$; note that the standard deviation is fixed at 1. We also examine here a situation where we try to learn both the mean and standard deviation (results in Figure 2a), where the FT rotation distribution is $\mathcal{N}(45, 15^2)$.

Defining meta-parameters. Our meta-parameters parameterize a rotational augmentation distribution that we apply to the PT images, $\mathcal{N}(\mu_{\text{PT}}, \sigma^2_{\text{PT}})$. We consider two scenarios. First, where we only optimize the mean: $\phi = \{\mu_{\text{PT}}\}$, and σ_{PT} is fixed to 1, which is the situation in Section 4. In this case, the initialization of ϕ is sampled uniformly from $[45, 135]$. Second, we optimize both the mean and the standard deviation of the rotation distribution: $\phi = \{\mu_{\text{PT}}, \sigma_{\text{PT}}\}$ (results in Figure 2a). In both settings, we expect the optimal PT rotation distribution for augmentations to be equal to what is used at FT time.

Model architectures. Our model is a fully-connected feedforward network, with 1 hidden layer with 64 hidden units and a ReLU activation.

Algorithm and Implementation details. We are able to use implicit differentiation with 1 Neumann term for the pre-training, and 1 step of differentiation through optimization for the fine-tuning training step. We use an Adam optimizer with a LR of 0.01 for pre-training and 0.3 for the meta-parameters. For fine-tuning, we use SGD (to match exactly the methods description in (4)) with the default learning rate of 0.01. We train with a batch size of 64 for each optimizer for 5 epochs. We alternate between taking 1 step of optimization for each set of parameters: $N_{\text{warmup}} = 0, K = 1, P = 1$. These hyperparameters were chosen based on a simple strategy discussed in Section D, without particular tuning.

Experimental setup. For the main experiments where we learn only the mean, we consider 10 different sampled mean initializations from $[45, 135]$. For the additional experiments where we learn the mean and the standard deviation, we fix the target distribution at $\mathcal{N}(45, 15^2)$ and examine two initializations: $\phi^{(0)} = \{0, 1\}$ and $\phi^{(0)} = \{90, 1\}$.

Results. When learning the mean, we are able to approximately recover the true rotation distribution after training with a final difference mean and standard error of $7.2 \pm 1.5°$, over 10 sampled mean rotations, indicating efficacy of the algorithm.

Next, we examine the results for learning the mean and standard deviation from different initializations, in Figure 2a, and observe that we can approximately recover the true augmentation distribution from both initializations.

E.2 Meta-Parameterized Per-Example Weighting

Here, we have additional details for the example weighting synthetic experiments in Section 4.

Dataset. PT and FT tasks are again based on supervised MNIST image classification. The PT task is adjusted to be a 1000-class problem, where MNIST digits in classes 0-4 keep their original labels, and MNIST digits in classes 5-9 are now assigned a noisy label: a random label between 5 and 1000. Our PT set is 3000 randomly-sampled MNIST data points. We use the standard MNIST training set for pre-training, and if any data point is in class 5-9 we noise it, by assigning a random label between 5 and 1000. We use the standard MNIST testing set for FT, split into a FT training and validation set. The FT set contains only images with classes 0-4.

Defining meta-parameters. Our meta-parameters ϕ are the parameters of a *weighting CNN* that assigns an importance weight to each PT data point, which is then used to weight the loss on that data point during PT. We expect the optimal weighting strategy to assign maximal weight to PT images in classes 0-4, since these are not noisy and are seen at FT time, and minimal weight to the other images, since these have noisy labels.

Model architectures. Our weighting network has an architecture of two convolutional layers, then a fully-connected layer. The first layer has 32 filters with a kernel size of 5, followed by batch-norm, with a ReLU activation and max pooling. The second convolutional layer is the same as the first, except with 64 filters and a kernel size of 3. The fully-connected layer has a 1-dimensional output with an activation of 2σ applied, so the output is in $(0, 2)$.

Implementation details. As with the MNIST augmentation experiments, we are able to use implicit differentiation with 1 Neumann terms for the PT, and 1 step of differentiation through optimization for the FT training. Again, we use an Adam optimizer with default parameters for PT and the meta-parameters, and SGD with default learning rate of 0.01 for FT. We use a batch-size of 100 for each optimizer and train each seed for 100 epochs. We alternate between taking 1 step of optimization for each set of parameters: $N_{\text{warmup}} = 0, K = 1, P = 1$. These hyperparameters were chosen based on a simple strategy discussed in Section D, without particular tuning.

Results. Using Algorithm 1 once again, we find that PT images from class 0-4 are assigned high weight, and those from classes 5-1000 are assigned low weight. This is an expected result: since the PT classes 0-4 are also the FT classes, we expect images from these classes to be upweighted. PT images not from these classes do not appear at FT and have noisy labels, hence are downweighted. This result is visualized in Figure 2b.

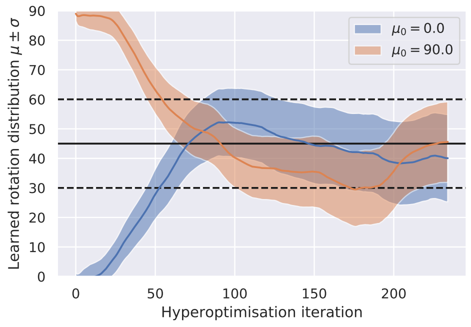
E.3 The Impact of Approximating Meta-Parameter Gradients

We now study how using the two gradient approximations in our algorithm compare to storing the entire PT and FT process in GPU memory and differentiating through the whole process to obtain meta-parameter gradients. We consider our first synthetic setting, where we aim to learn rotation augmentations for MNIST PT, given that the FT set is augmented in a specific way. In the following experiments, the FT set is augmented with rotations drawn from $\mathcal{N}(90, 1)$.

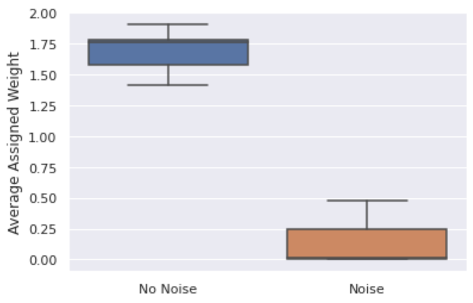
Experimental setup. We compare the following methods to study the impact of the gradient approximations.

- Backpropagation through training (BPTT): The PT augmentation distribution is initialized to $\mathcal{N}(45, 1)$. We do 500 steps of PT and 500 steps of FT steps, and use BPTT (through these 1000 optimization steps) to optimize the augmentations. This is near the limit of what we could fit into our GPU memory. This process is then repeated for 500 hyperparameter optimization steps.
- Meta-parameterized PT: We run our algorithm. The PT augmentation distribution is initialized to $\mathcal{N}(45, 1)$. We set $P = 1$, $K = 1$, running for 500 PT and FT steps overall (for a fair comparison with BPTT).
- Optimal augmentations: We set the PT augmentation distribution to be the optimal setting (i.e., identical to that used for FT): $\mathcal{N}(90, 1)$. This is also run for 500 PT and 500 FT steps.
- Initialization augmentations: We set the PT augmentation distribution to be: $\mathcal{N}(45, 1)$ as a baseline. This is also run for 500 PT and 500 FT steps.

Results.



(a) Visualizing the optimization of meta-parameters, which parameterize PT rotation augmentation distributions. We consider two different meta-parameter initializations of mean/standard deviation, $\phi = \{90, 1\}$ (orange) and $\phi = \{0, 1\}$ (blue), showing the mean (solid line) and standard deviation (shaded region) over learning. The rotation distribution for the FT validation set is $\mathcal{N}(45, 15^2)$, shown with a solid black line (mean) and dashed lines (plus/minus standard deviation). In both cases, we observe approximate recovery of the optimal meta-parameters, namely the FT mean and standard deviation. The final mean and standard deviation for the 90 initialization and 0 initialization are 42.4 ± 13.9 and 45.9 ± 15.5 respectively.



(b) The distribution of importance weights assigned to examples with/without noisy labels, over 10 random seeds of weights, produced by a weighting CNN. We show the average weight applied to non-noised and noised examples, normalized by dividing by the sum of the data weights. The weighted CNN has recovered the desired solution of down-weighting examples with noisy labels, indicating successful learning of high-dimensional meta-parameters.

Figure (2) **Results for learning pre-training augmentation meta-parameters.**

- BPTT without compute limitations: Running BPTT for 500 hyperparameter optimization steps takes about 20 hours. Doing so, it achieves a test accuracy of 88.0%.
- Meta-parameterized PT: Running our method takes about 30 minutes. This achieves a test accuracy of 87.6%.
- BPTT with compute limitations: Limiting the compute budget of BPTT to be similar to our method, it obtains a test accuracy 83.4%.
- Optimal augmentations: This achieves a test accuracy of 88.3%.
- Initialization augmentations: This achieves a test accuracy of 80.1%.

Analysis. As seen, our method, with about 2-3% of the compute time and significantly lower memory cost than BPTT, obtains very comparable performance in this toy domain, and almost matches the performance with the optimal hyperparameter setting. This indicates effective optimization of the hyperparameters.This performance is achieved even when differentiating through a short FT optimization of 1 step.

Conclusions. In this toy domain, our method obtains performance very comparable to BPTT and the optimal hyperparameter setting, and has a fraction of the compute and memory cost of BPTT. This suggests that optimizing the augmentations online is not incurring significant short horizon/truncation bias.

F Meta-Parameterized Multitask PT: Further Details

We provide further dataset details, experimental details, and results for the multitask PT experiments. All experiments in this section were run on a single NVIDIA V100 GPU.

F.1 Further dataset details

The transfer learning benchmark we consider is the biological data benchmark from Hu et al. [28] where the prediction problem at both PT and FT is multitask binary classification: predicting the presence/absence of specific protein functions (y) given a Protein-Protein Interaction (PPI) network as input (represented as a graph $\boldsymbol{x}$). The PT and FT datasets both contain 88K graphs.

Hu et al. [28] provide open-source code in their paper to download the raw dataset and then pre-process it. The important steps are extracting subgraphs of the PPI networks centered at particular proteins, and then using the Gene Ontology to identify the set of protein functions associated with each of the proteins.

Importantly, the set of protein functions that we predict at PT time and FT time are different. The PT targets represent coarse-grained biological functions, and the FT targets are fine-grained biological functions, which are harder to obtain experimentally and therefore there is interest in predicting them having pre-trained a model on predicting the targets that are more readily obtained. The PT dataset has labels $y \in \{0, 1\}^{5000}$, and the FT dataset has labels $y \in \{0, 1\}^{40}$.

Hu et al. [28] discuss the importance of appropriate train/validation/test set splitting for this domain. We follow their suggestion and use the *species split*, where the test set involves predicting biological functions for proteins from new species, not encountered at training/validation time.

We refer the reader to Hu et al. [28] for full details on the pre-processing and construction of subgraphs, the nature of the labels, and the splitting strategy for training, validation, and testing.

F.2 Further experimental details

F.2.1 Baselines

We include most important details for baselines in Section 5. Here, for the CoTrain + PCGrad baseline we provide further details, and we also include information about another baseline, CoTrain + Learned Task Weights.

CoTrain + PCGrad details: In our implementation, we computed gradient updates using a batch of data from $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$ separately, averaging the losses across the set of binary tasks in each dataset (5000 for $\mathcal{D}_{\text{PT}}$ and 40 for $\mathcal{D}_{\text{FT}}$). PCGrad [72] was then used to compute the final gradient update given these two averaged losses. We also experimented with: (1) computing the overall update using all 5040 tasks (rather than averaging), but this was too memory expensive; and (2) computing the overall update using an average over the 5000 PT tasks and each of the 40 FT tasks individually, but this was unstable and did not converge.

A further baseline: CoTrain + Learned Task Weights: We also tried a variant of CoTrain where we learn task weights for each of the 5040 tasks (from $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$), along with training the base model. We treat the task weights as high-dimensional supervised learning hyperparameters and optimize these task weights using traditional gradient-based hyperparameter optimization, following the work from [42]. These weights are optimized based on the model's loss on the validation set split of $\mathcal{D}_{\text{FT}}$.

F.2.2 Implementation details

General details for all methods. For all methods, we use the Graph Isomorphism Network (GIN) architecture [69], which was found to be effective on this domain [28].

All methods first undergo PT for 100 epochs with Adam, with a batch size of 32. We used LR=1e-3 for Graph Supervised PT, CoTrain and CoTrain + PCGrad, which is the default LR in the prior work [28]. For the two nested optimization methods that jointly pre-train and learn weights, CoTrain + Learned Task Weights and Meta-Parameterized PT, we used LR=1e-4; we originally tried LR=1e-3, but this led to unstable nested optimization.

After PT, all methods are then fine-tuned for 50 epochs using Adam, with a batch size of 32, over 5 random seeds, using early stopping based on validation set AUC (following [28]). We used 5 seeds rather than 10 (Hu et al. [28] used 10) for computational reasons. For all models, we initialize a new FT network head on top of the PT network body. At FT time, we either FT the whole network (Full transfer) or freeze the PT encoder and learn the FT head alone (Linear Evaluation [50]). We report results here for both FT policies for all methods.

When fine-tuning models using the Full Transfer paradigm, we found that methods were sensitive to LR choices and a FT LR of 1e-3 used in Hu et al. [28] was unstable. The Adam optimizer FT LRs of 1e-5, 3e-5, and 1e-4 were tried for different methods, with FT validation set AUC used to choose the best LR. For Meta-Parameterized PT, we used a full transfer FT LR of 1e-5, and for the other methods, we used 3e-5. For linear evaluation, we used Adam with an LR of 1e-4 for all methods, which was stable.

Further details for Meta-Parameterized PT. For meta-parameterized PT, during the meta-PT phase, we use the Adam optimizer with a learning rate of 1e-4 for both PT and FT parameters, and use Adam with a LR of 1 for meta-parameters. These values were set based on the methodology in Appendix D. In Algorithm 2, we use a Neumann series with 1 step in evaluating the inverse Hessian for PT, 1 warmup epoch, $P = 10$ PT steps, $K = 1$ FT steps; we did not search over values for these, and these choices were partly influenced by compute considerations (e.g., large K is more memory expensive).

With these settings, meta-parameterized PT on this task takes about 8-9 GB of GPU memory (about twice the memory cost of normal PT, which is 4-5 GB), and takes about 5 hours to run (as compared to about 2.5 hours for standard PT).

Further details for CoTrain + Learned Task Weights. Following a similar process to the above, we used Adam with LR of 1e-4 for the base parameters and LR of 1 for the task weights. We use a Neumann series with 1 step when using the method from [42] for the fairest comparison with meta-parameterized PT.

F.2.3 Experimental Setup

We re-state the two settings considered, and provide more details about an additional scenario in the Partial FT Access setting.

(1) *Full FT Access:* Provide methods full access to $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$ at PT time ($\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$) and evaluate on the full set of 40 FT tasks.

(2) *Partial FT Access:* Consider two situations. First, construct a scenario where we limit the FT data available at PT time directly: $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right| = 0.5\,|\mathcal{D}_{\text{FT}}|$. We assess performance on the full FT dataset, as before. Results for this were not presented in the main text due to space constraints.

Second, limit the number of FT tasks seen at PT time, by letting $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ include only 30 of the 40 FT tasks. At FT time, models are fine-tuned on the held-out 10 tasks not in $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$. We use a 4-fold approach where we leave out 10 of the 40 FT tasks in turn, and examine performance across these 10 held-out tasks, over the folds.

F.3 Further results

F.3.1 Quantitative Results

Summary of main quantitative results. Table 4 summarizes the main results across full and limited data/task regimes, reporting the better of Full Transfer and Linear Evaluation. We observe consistent improvements with the meta-parameterized PT strategy over the baselines on the three different experimental evaluation settings.

In the remainder of this section, we discuss these quantitative results further, showing both full transfer and linear evaluation results, and other analysis.

Method	AUC ($\left\|\mathcal{D}_{\mathrm{FT}}^{(\mathrm{Meta})}\right\| = \|\mathcal{D}_{\mathrm{FT}}\|$)	AUC ($\left\|\mathcal{D}_{\mathrm{FT}}^{(\mathrm{Meta})}\right\| = 0.5\,\|\mathcal{D}_{\mathrm{FT}}\|$)	AUC ($\mathcal{D}_{\mathrm{FT}}^{(\mathrm{Meta})}$ excludes tasks)
No PT	66.6 ± 0.7	66.6 ± 0.7	65.8 ± 2.5
Graph Sup PT	74.7 ± 0.1	74.7 ± 0.1	74.8 ± 1.8
CoTrain	70.2 ± 0.3	71.0 ± 0.2	69.3 ± 1.8
CoTrain + PCGrad	69.4 ± 0.2	71.1 ± 0.2	68.1 ± 2.3
Meta-Parameterized PT	$\mathbf{78.6 \pm 0.1}$	$\mathbf{78.2 \pm 0.1}$	$\mathbf{77.0 \pm 1.3}$

Table (4) **Meta-Parameterized PT improves predictive performance in three evaluation settings.** Table showing mean AUC and standard error on mean for three different evaluation settings. **First results column: Full FT Access, with evaluation on all tasks, with all FT data provided at PT time. Second results column: Partial FT Access, evaluation with limited FT data at PT time.** When only 50% of the FT dataset is provided at PT time, Meta-Parameterized PT can again improve on other methods in mean test AUC over 40 FT tasks, demonstrating sample efficiency. **Third results column: Partial FT Access, evaluation on new, unseen tasks at FT time.** When 10 of the 40 available FT tasks are held-out at PT, over four folds (each set of 10 FT tasks held out in turn), considering mean test AUC across tasks and folds (and standard error on the mean), meta-parameterized PT obtains the best performance: it is effective even with partial information about the downstream FT tasks.

Method	Full Transfer	Linear Evaluation
Rand Init (from [28])	64.8 ± 0.3	N/A
Rand Init (reimplement, lower FT LR)	66.6 ± 0.7	N/A
Graph Sup PT (from [28])	69.0 ± 0.8	N/A
Graph Sup PT (reimplement, lower FT LR)	73.9 ± 0.2	74.7 ± 0.1
CoTrain	70.2 ± 0.3	65.9 ± 0.1
CoTrain + PCGrad	69.4 ± 0.2	62.4 ± 0.3
CoTrain + Learned Task Weights	67.7 ± 0.2	64.4 ± 0.1
Meta-Parameterized PT	74.7 ± 0.3	78.6 ± 0.1

Table (5) **Meta-Parameterized PT results in improved predictive performance.** Table showing mean AUC and standard error on mean across 40 FT tasks on the held-out test set, over 5 random FT seeds. We observe that Meta-Parameterized PT outperforms other baselines in both Full Transfer and Linear Evaluation settings, with significant improvement with Linear Evaluation. Note that with a lower FT LR, baselines from [28] are improved relative to previously reported performance.

Further results for Full FT Access setting. Table 5 presents results for all methods across 40 FT tasks, considering both full transfer and linear evaluation. We observe that meta-parameterized PT improves on other baselines in both settings, but most noticeably so in linear evaluation. We also present the results for No PT and Graph Supervised PT from Hu et al. [28]. We observe improvements with our re-implementation, which uses lower FT LRs.

Studying potential overfitting in CoTrain strategies. For methods leveraging the FT dataset during PT, the process of performing FT might worsen performance if the model overfits the FT training set. We evaluate FT test performance 'online' during the PT phase, with results in Table 6, and observe that meta-parameterized PT outperforms other methods here also. We do observe some of this overfitting behaviour: note the improved performance on the test set with the learned weights strategy.

Further results for Partial FT Access setting. Table 7 shows improved performance even with smaller meta-FT datasets, and Table 8 shows improved performance even with limited tasks at meta-FT time.

F.3.2 Qualitative Results

We now analyze other aspects of meta-parameterized PT.

Analyzing learned representations. To understand the impact of meta-parameterized PT on what the model learns, we compare the learned representations on the FT data across the different PT

Method	Test AUC	Validation AUC
Meta-Parameterized PT	76.1	88.2
CoTrain	67.3	83.1
CoTrain + PCGrad	69.0	84.0
CoTrain + Learned Task Weights	70.7	84.6

Table (6) Mean AUC across FT tasks evaluated during PT, for methods that use the FT set at PT time. The separate FT stage may worsen performance of some of the methods, and evaluating in this manner helps account for that. In this setting also, meta-parameterized PT improves on other baselines, in both test and validation set performance.

Method	Full Transfer	Linear Evaluation
Rand Init (from [28])	64.8 ± 0.3	N/A
Rand Init (reimplement, lower FT LR)	66.6 ± 0.7	N/A
Graph Sup PT (from [28])	69.0 ± 0.8	N/A
Graph Sup PT (reimplement, lower FT LR)	73.9 ± 0.2	74.7 ± 0.1
CoTrain	71.0 ± 0.2	64.4 ± 0.1
CoTrain + PCGrad	71.1 ± 0.2	64.4 ± 0.1
CoTrain + Learned Task Weights	66.0 ± 0.3	64.6 ± 0.3
Meta-Parameterized PT	74.3 ± 0.2	78.2 ± 0.1

Table (7) **Meta-Parameterized PT also improves predictive performance with smaller MetaFT datasets.** In a setting where only 50% of the FT dataset is provided at PT time, Meta-Parameterized PT can again improve on other methods in mean test AUC over 40 FT tasks, indicating that it is effective even with limited amounts of FT data available at PT time.

strategies using Centered Kernel Alignment (CKA) [36, 10] in Figure 3. We observe that Meta-Parameterized PT most closely resembles a combination of CoTrain + Learned Weights and Supervised PT, which is sensible given that it blends aspects of both approaches.

Analyzing learned weights. Figure 4 compares learned weights for meta-parameterized PT and the CoTrain+Learned Weights strategies. We observe differences in the histogram of weights, and also the specific values on a per-task basis for these two strategies, indicating that they learn different structures.

Analyzing negative transfer. Figure 5 assesses potential negative transfer on a per-task basis, comparing performance with PT to performance after supervised PT and meta-parameterized PT. Both PT strategies have little negative transfer, and meta-parameterized PT obtains a small extra reduction in negative transfer over standard supervised PT.

Method	Full Transfer	Linear Evaluation
Rand Init	65.8 ± 2.5	N/A
Graph Sup PT	71.5 ± 1.6	74.8 ± 1.8
CoTrain	69.3 ± 1.8	67.0 ± 2.0
CoTrain + PCGrad	67.1 ± 1.5	68.1 ± 2.3
CoTrain + Learned Weights	65.4 ± 2.0	69.1 ± 2.6
Meta-Parameterized PT	71.3 ± 2.5	77.0 ± 1.3

Table (8) **When evaluating on new, unseen tasks at FT time, meta-parameterized PT again improves on other methods.** We consider a setting where 10 of the 40 available FT tasks are held-out at PT, and only provided at FT time. Over four folds (where different sets of 10 FT tasks are held out in turn), considering mean test AUC across tasks and folds (and standard error on the mean over folds), meta-parameterized PT obtains the best performance. This suggests that the method can perform well even with partial information about the downstream FT tasks.

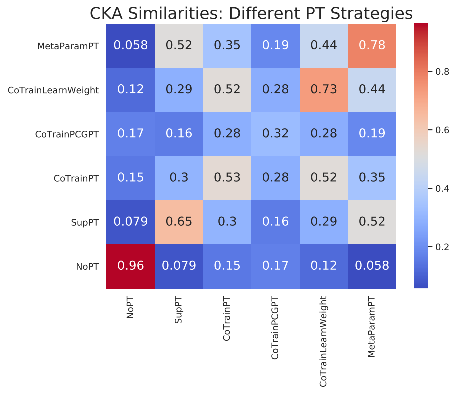


Figure (3) Comparing learned representations with different PT strategies using CKA [36]. We obtain model representations before the final linear layer across 6400 FT data points, and then compute CKA between pairs of models (averaging over different random initialisations). We observe that Meta-Parameterized PT most closely resembles a combination of CoTrain + Learned Weights and Supervised PT, which is sensible given that it blends aspects of both approaches: meta-parameterized PT learns task weights to modulate the learned representations (as in CoTrain + Learned Weights), and representations are adapted using the PT task alone (as in supervised PT). Interestingly, CoTrain + PCGrad has comparatively little similarity to most other methods in terms of its learned representations.

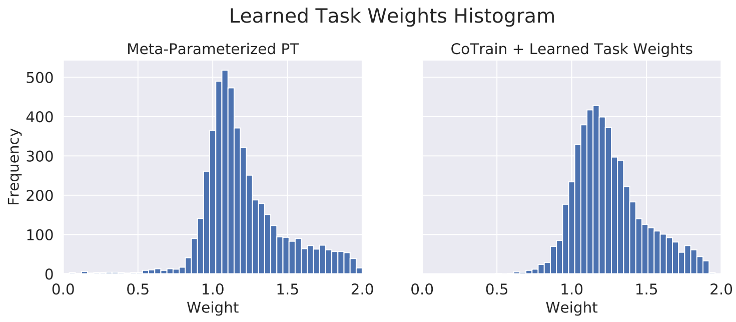


Figure (4) Comparing learned weights on 5000 PT tasks for meta parameterized PT and with CoTrain + Learned Weights. We observe that different structures appear to be learned by these approaches; the median/half IQR in absolute difference in learned weights is 0.13 ± 0.09. Meta-Parameterized PT appears to have more tasks downweighted (weights below 0.5) than the CoTrain approach.

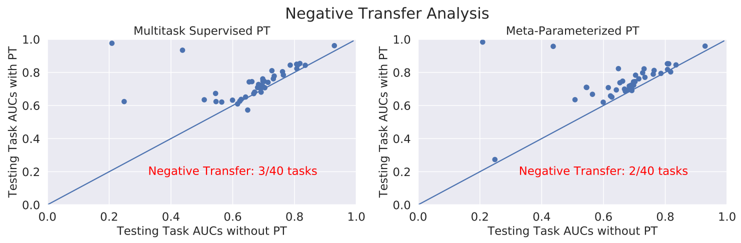


Figure (5) Comparing performance on FT tasks with and without two different PT strategies: standard supervised PT on the left, and meta-parameterized PT on the right. We show the mean performance over 5 seeds on each of the 40 FT tasks without PT (x axis) and with PT (y axis). A small improvement is observed in reduced negative transfer with Meta-Parameterized PT.

G Meta-Parameterized SimCLR PT: Further Details

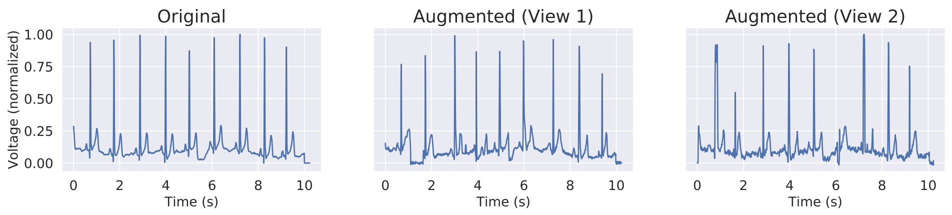


Figure (6) A single lead (or channel) of the 12 lead ECG signal and two augmented views (following cropping, jittering, and temporal warping) that are used in contrastive learning.

We provide further SimCLR details, dataset details, experimental details, and results for the SimCLR ECG experiments. All experiments in this section were run on a single NVIDIA V100 GPU.

G.1 SimCLR summary

SimCLR is a variant of contrastive self-supervised learning [65, 68, 50, 26]. During training, examples are augmented in two different ways to create two views $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, each of which are encoded independently to produce representations $f^{(\text{enc})}(\boldsymbol{x}_i) = \boldsymbol{h}_i$ and $f^{(\text{enc})}(\boldsymbol{x}_j) = \boldsymbol{h}_j$. These representations are further transformed using a multi-layer decoder ("projection head") to produce vectors $f^{(\text{dec})}(\boldsymbol{h}_i) = \boldsymbol{z}_i$ and $f^{(\text{dec})}(\boldsymbol{h}_j) = \boldsymbol{z}_j$. Models are trained to minimize the normalized temperature-scaled cross-entropy loss (NT-Xent), which contrasts the similarity between pairs of views derived from the same example against the other $2N-2$ views in a minibatch of size N:

$$\mathcal{L}_{\text{PT}}(\boldsymbol{z}_i, \boldsymbol{z}_j) = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} \tag{5}$$

where $\text{sim}(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}/(\|\boldsymbol{a}\|\|\boldsymbol{b}\|)$ is cosine similarity and τ is the temperature hyperparameter.

G.2 Further dataset details

We construct our semi-supervised learning (SSL) problem using PTB-XL [64, 20], an open-source dataset of electrocardiogram (ECG) data. Let the model input at both PT and FT time be denoted by $\boldsymbol{x}$, which represents a 12-lead (or channel) ECG sampled at 100 Hz for 10 seconds resulting in a 1000×12 signal. An example signal is in Figure 6. The PTB-XL dataset contains 21837 ECGs from 18885 unique patients. Each ECG has a 5-dimensional label $y \in \{0, 1\}^5$, where each dimension indicates whether the signal contains certain features indicative of particular diseases/pathologies, namely: Normal ECG, Myocardial Infarction, ST/T Change, Conduction Disturbance, and Hypertrophy. The dataset is split in 10 folds on a patient-level (ECGs from the same patient are all in the same fold), with a suggested train-validation-testing split.

To form an SSL problem from this dataset, we take the training and validation folds, remove the labels, and use only the unlabelled ECGs as the PT dataset. This PT dataset has 19634 unique ECGs. For the FT dataset, we take a random sample of $|\mathcal{D}_{\text{FT}}|$ ECG-label pairs from the training and validation folds. As is common in prior SSL work, we consider different sizes of $\mathcal{D}_{\text{FT}}$ to understand performance given different amounts of labelled data. The FT testing set is the testing fold of the original dataset, which has 2203 ECG-label pairs.

At both PT and FT time, ECGs are normalized before input to the model using zero mean-unit variance normalization, following Wagner et al. [64].

We refer the reader to the open-source data repository on PhysioNet [20], and the paper introducing the dataset [64] for further details.

G.3 Further experimental details

G.3.1 ECG Data Augmentations

To augment each ECG for SimCLR, we apply three transformations in turn (based on prior work in time series augmentation [30, 66]):

1. **Random cropping:** A randomly selected portion of the signal is zeroed out. We randomly mask up to 50% of the input signal.
2. **Random jittering:** IID Gaussian noise is added to the signal. The noise is zero mean and has standard deviation equal to 10% of the standard deviation of the original signal.
3. **Random temporal warping:** The signal is warped with a random, diffeomorphic temporal transformation. To form this, we sample from a Gaussian with zero mean, and a fixed variance at each temporal location, to generate a 1000 dimensional random velocity field. This velocity field is then integrated (following the scaling and squaring numerical integration routine used by Balakrishnan et al. [4, 5]). This resulting displacement field is then smoothed with a Gaussian filter to generate the smoothed temporal displacement field, which is 1000 dimensional. This field represents the number of samples each point in the original signal is translated in time. The field is then used to transform the signal, translating each channel in the same way (i.e., the field is the same across channels).

Two augmented views of an ECG are shown in Figure 6.

G.3.2 Implementation details

General details for all methods. For all methods, we use a 1D CNN based on a ResNet-18 [23] architecture as the base model that undergoes PT & FT. This model has convolutions with a kernel size of 15, and stride 2 (set based on the rough temporal window we wish to capture in the signal). The convolutional blocks have 32, 64, 128, and 256 channels respectively. The output of these layers is average pooled in the temporal dimension, resulting in a 256 dimensional feature vector. For SimCLR PT, the projection head takes this 256 dimensional vector as input and is a fully connected network with 1 hidden layer of size 256, and output size of 128, with ReLU activation. These hyperparameters were not tuned.

The SimCLR methods are first pre-trained on the PT dataset using SimCLR PT, with a temperature of 0.5 in the NT-Xent loss. We use Adam with an LR of 1e-4 for SimCLR PT, with a batch size of 256, and pre-train for 50 epochs. We consider 3 PT seeds. The methods are then fine-tuned on the FT dataset, replacing the projection head with a new linear FT network head. This whole network is fine-tuned for 200 epochs with Adam, learning rate of 1e-3, batch size of 256. We used an 80%-20% split of the labelled data to form training and validation sets, and validation set AUC was used for early stopping. We consider 5 FT seeds, resulting in a total of 15 runs for each method at each setting.

Further details for Meta-Parameterized PT. Meta-parameterized SimCLR incorporates a learned per-example temporal warping strength. We form this by instantiating a four-layer 1D CNN $w(\boldsymbol{x}; \phi)$ that takes in the input ECG $\boldsymbol{x}$ and outputs the variance (1-D output) of the velocity field used to generate the random velocity field. This network has four blocks of convolution, batch norm, and ReLU activation with a kernel size of 15, stride of 2, and 32 channels. We also a optimize a global warping strength scale that multiplies the network output to adjust the overall scale of the warping. The network weights and the global scale are optimized using Adam, with LR=1e-4 and LR=1 respectively. These values were set based on the methodology in Section D. In Algorithm 2, we use a Neumann series with 1 step when evaluating the inverse Hessian for PT, 1 warmup epoch, $P = 10$ PT steps, $K = 1$ FT steps; we did not search over these, and chose these values based on compute considerations. However, we do conduct a comparison with running for other values of K in Appendix G.4.

With these settings, meta-parameterized PT on this task takes about 8-9 GB of GPU memory (about twice the memory cost of normal PT, which is 4-5 GB), and takes about 3 hours to run (as compared to about 1.5 hours for standard PT).

When running meta-parameterized PT with very small meta-FT datasets, of size 10 or 25, the 80%-20% split is not as practical. When $\left|\mathcal{D}_{\text{FT}}^{\text{(Meta)}}\right| = 10$, we use a 50-50 split in training and validation, and when it is 25, we use a 60-40 split.

	Test AUC at different FT dataset sizes $|\mathcal{D}_{\text{FT}}|$			
	100	250	500	1000
No PT	71.5 ± 0.7	76.1 ± 0.3	78.7 ± 0.3	82.0 ± 0.2
SimCLR	74.6 ± 0.4	76.5 ± 0.3	79.8 ± 0.3	82.2 ± 0.3
SimCLR + OptSLA	74.6 ± 0.6	77.0 ± 0.3	79.6 ± 0.4	82.8 ± 0.2
Meta-Parameterized SimCLR	$\mathbf{76.1 \pm 0.5}$	$\mathbf{77.8 \pm 0.4}$	$\mathbf{81.7 \pm 0.2}$	$\mathbf{84.0 \pm 0.3}$

Table (9) **Meta-Parameterized SimCLR obtains improved semi-supervised learning performance.** Table showing mean AUC/standard error over seeds across 5 FT binary classification tasks for baselines and meta-parameterized SimCLR at different sizes of $\mathcal{D}_{\text{FT}}$, with $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$. We observe improvements in performance with meta-parameterized SimCLR over other baselines, including SimCLR + OptSLA, which optimizes the augmentations purely for one-stage supervised learning (rather than two-stage PT and FT).

An additional baseline: SimCLR + Optimized Supervised Learning Augmentations (SimCLR + OptSLA): We also investigated a baseline in this domain where the same parametric augmentation policy used for meta-parameterized PT above is: (1) optimized for supervised learning on the labelled FT set, $\mathcal{D}_{\text{FT}}$, following the method from [42]; (2) used as is in SimCLR PT to learn representations; (3) evaluated in a standard FT setting. When using the algorithm from [42], the augmentation meta-parameters are optimized based on the model's loss on the validation set split of $\mathcal{D}_{\text{FT}}$, as is typical in hyperparameter optimization. This baseline compares how optimizing augmentations over the two stage PT and FT compares to optimizing for supervised learning alone. We use Adam for optimization, and use 1 Neumann step in the algorithm from [42].

G.3.3 Experimental Setup

We re-state the two experimental settings considered. In both settings, we evaluate performance as average AUC across the 5 binary classification tasks, reporting mean and standard error over the 15 runs.

(1) *Full FT Access*, standard SSL: consider different sizes of the labelled FT dataset $\mathcal{D}_{\text{FT}}$ and make all the FT data available at meta-PT time, $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$.

(2) *Partial FT Access*, examining data efficiency of our algorithm: SSL when only limited FT data is available at meta-PT time: $\mathcal{D}_{\text{FT}}^{(\text{Meta})} \subseteq \mathcal{D}_{\text{FT}}$.

G.4 Further results

We now present additional results in the semi-supervised learning domain.

Further Full PT Access results with new baseline. We first present results in the Full PT Access setting, varying $|\mathcal{D}_{\text{FT}}|$ and setting $\mathcal{D}_{\text{FT}} = \mathcal{D}_{\text{FT}}^{(\text{Meta})}$, shown in Table 9. As seen, meta-parameterized SimCLR obtains improvements over the one-stage hyperparameter learning baseline, SimCLR + OptSLA, suggesting that learning augmentations for the two-stage PT & FT process is advantageous.

Impact of K. We now study the impact of different values of K, the number of unrolled differentation steps when computing the gradient through FT. In our main experiments, we set $K = 1$ for simplicity and computational efficiency. We now seek to understand the following alternative choices:

- $K = 0$: In this setting, we perform **no** FT when optimizing the meta-parameters; that is, we use a randomly initialized linear classifier on top of the PT representations when we compute the FT loss. The meta-learning problem here corresponds to learning PT meta-parameters that optimize the performance of a randomly initialized linear classifier. This experiment tests what happens when the gradient through FT is noisy, but the component through PT is informative.
- $K > 1$: This setting tests whether unrolling more steps during FT can improve the gradient signal received when optimizing meta-parameters.

	Test AUC at different FT dataset sizes $\lvert\mathcal{D}_{\text{FT}}\rvert$			
	100	250	500	1000
SimCLR	74.6 ± 0.4	76.5 ± 0.3	79.8 ± 0.3	82.2 ± 0.3
$K = 0$	75.3 ± 0.5	77.1 ± 0.5	80.5 ± 0.4	83.7 ± 0.3
$K = 1$	76.1 ± 0.5	77.8 ± 0.4	81.7 ± 0.2	84.0 ± 0.3
$K = 5$	76.6 ± 0.2	78.3 ± 0.3	81.9 ± 0.4	84.2 ± 0.2
$K = 10$	76.3 ± 0.5	78.1 ± 0.4	81.7 ± 0.4	84.3 ± 0.3

Table (10) **Examining how the number of unrolled FT steps affects semi-supervised learning performance.** Table showing mean AUC/standard error over seeds across 5 FT binary classification tasks for meta-parameterized SimCLR when we vary the number of unrolled FT steps used to compute the meta-parameter gradient. We observe that using a noisy FT gradient ($K = 0$) improves on not optimizing augmentations at all, but is worse than using a single step ($K = 1$). Using more unrolled steps can lead to small improvements.

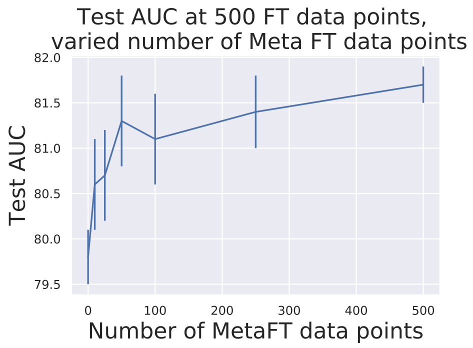


Figure (7) **Meta-Parameterized SimCLR is effective when only small amounts of FT data are available at PT time.** Test set AUC when varying $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$: the number of FT data points provided at PT time, considering $\lvert\mathcal{D}_{\text{FT}}\rvert = 500$. We see that meta-parameter learning is effective even at small $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$ (sharp improvement in performance at small $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$)

Results are shown in Table 10. As can be seen, unrolling through one step of FT ($K = 1$) improves upon using a noisy FT gradient ($K = 0$) in all cases. Using a noisy FT gradient but an informative PT component ($K = 0$) improves on not optimizing the augmentations at all (SimCLR). This implies that both the PT and FT components inform the optimization of the augmentations. When $K > 1$, we do observe some improvement with more steps, but diminishing returns as K increases further.

Further Partial PT Access results. We now consider the Partial FT Access setting. Firstly, Figure 7 shows the performance of meta-PT when we fix $\lvert\mathcal{D}_{\text{FT}}\rvert = 500$ and vary $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$. We find that meta-PT can be effective even with very small validation sets (consider the sharp improvement at small MetaFT data points, with 0 MetaFT points representing no optimization of the augmentations). This result was just considering the $\lvert\mathcal{D}_{\text{FT}}\rvert = 500$ setting; in Figure 8, we consider other FT dataset sizes and analyze performance. We see that in all regimes, there is a noticeable increase in performance at small meta-FT dataset sizes, which is a desirable result since it shows that our algorithm can be effective even with very limited labelled data available at meta-PT time.

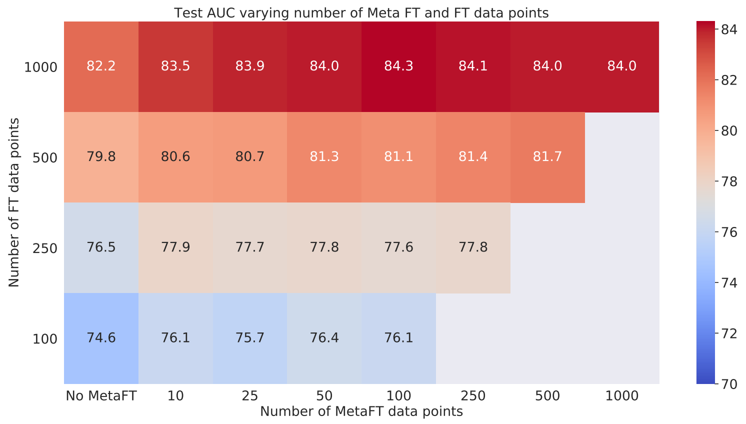


Figure (8) Sweeping over meta FT/FT data points and analyzing performance trends. We observe that across various settings of FT data availability, a small amount of MetaFT data can lead to significant performance improvements.