# Tailoring: encoding inductive biases by optimizing unsupervised objectives at prediction time

**Ferran Alet, Maria Bauza, Kenji Kawaguchi,**
**Nurullah Giray Kuru, Tomás Lozano-Pérez, Leslie Pack Kaelbling**
MIT
{alet,bauza,kawaguch,ngkuru,tlp,lpk}@mit.edu

## Abstract

From CNNs to attention mechanisms, encoding inductive biases into neural networks has been a fruitful source of improvement in machine learning. Adding auxiliary losses to the main objective function is a general way of encoding biases that can help networks learn better representations. However, since auxiliary losses are minimized only on training data, they suffer from the same generalization gap as regular task losses. Moreover, by adding a term to the loss function, the model optimizes a different objective than the one we care about. In this work we address both problems: first, we take inspiration from transductive learning and note that after receiving an input but before making a prediction, we can fine-tune our networks on any unsupervised loss. We call this process *tailoring*, because we customize the model to each input to ensure our prediction satisfies the inductive bias. Second, we formulate *meta-tailoring*, a nested optimization similar to that in meta-learning, and train our models to perform well on the task objective after adapting them using an unsupervised loss. The advantages of tailoring and meta-tailoring are discussed theoretically and demonstrated empirically on a diverse set of examples.

## 1 Introduction

The key to successful generalization in machine learning is the encoding of useful inductive biases. A variety of mechanisms, from parameter tying to data augmentation, have proven useful to improve the performance of models. Among these, auxiliary losses can encode a wide variety of biases, constraints, and objectives; helping networks learn better representations and generalize more broadly. Auxiliary losses add an extra term to the task loss that is minimized over the training data.

However, they have two major problems:

1. Auxiliary losses are only minimized at training time, but not for the query points. This leads to a generalization gap between training and testing, in addition to that of the task loss.
2. By minimizing the sum of the task loss plus the auxiliary loss, we are optimizing a different objective than the one we care about (only the task loss).

In this work we propose a solution to each problem:

1. We use ideas from *transductive learning* to minimize unsupervised auxiliary losses at each query, thus eliminating their generalization gap. Because these losses are unsupervised, we can optimize them at any time inside the prediction function. We call this process *tailoring*, since we customize the model to each query.
2. We use ideas from *meta-learning* to learn a model that performs well on the task loss after being tailored with the unsupervised auxiliary loss; i.e. *meta-tailoring*. This effectively trains the model to leverage the unsupervised tailoring loss in order to minimize the task loss.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

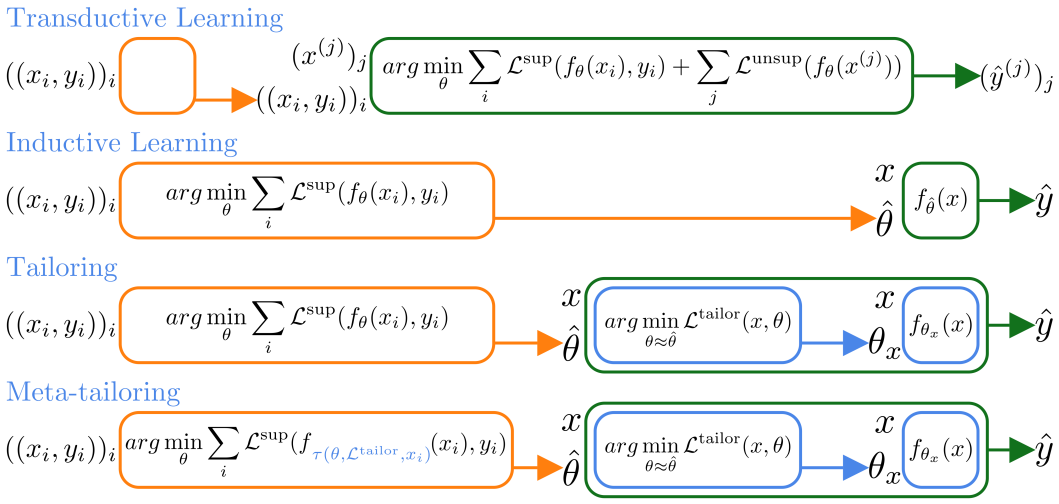


Figure 1: Comparison of several learning settings with *offline* computation in the orange boxes and *online* computation in the green boxes, with tailoring in blue. For meta-tailoring training, $\tau(\theta, \mathcal{L}^{\text{tailor}}, x) = \text{argmin}_{\theta' \approx \theta} \mathcal{L}^{\text{tailor}}(x, \theta')$ represents the tailoring process resulting in $\theta_x$.

**Illustrative example** Imagine you want to use a neural network to predict the motion of a planetary system: given the positions and velocities of each planet, the network predicts their future positions and velocities. Additionally, we could encode energy and momentum conservation by adding an auxiliary loss encouraging the neural network to conserve energy and momentum for the training examples. However, this does not guarantee that the network will conserve them for test queries. Alternatively, we can exploit that evaluating these conservations requires comparing only the input with the prediction without needing access to the true target. Therefore, we can enforce these conservations by optimizing an unsupervised objective within the prediction function. In doing so, we *tailor* the model to each individual query to ensure it satisfies energy and momentum conservation. Taking into account this prediction-time adaptation during training leads to a two-layer optimization, where we train to make accurate predictions after encouraging the physical conservations.

**Tailoring a predictor** Traditionally, supervised learning is approached within the inductive learning framework, shown in the second row of Figure 1. There, an algorithm consumes a training dataset of input-output pairs, $((x_i, y_i))_{i=1}^n$, and produces a set of parameters $\hat{\theta}$ by minimizing a supervised loss $\sum_{i=1}^n \mathcal{L}^{\text{sup}}(f_\theta(x_i), y_i)$ and, optionally, an unsupervised auxiliary loss $\sum_{i=1}^n \mathcal{L}^{\text{unsup}}(\theta, x_i)$. These parameters specify a hypothesis $f_{\hat{\theta}}(\cdot)$ that, given a new input $x$, generates an output $\hat{y} = f_{\hat{\theta}}(x)$. This problem setting misses a substantial opportunity: before the learning algorithm sees the query point $x$, it has distilled the data down to the parameters $\hat{\theta}$, which are frozen during inference, and so it cannot use new information about the *particular* $x$ that it will be asked to make a prediction for.

Vapnik recognized an opportunity to make more accurate predictions when the query point is known, in a framework that is now known as *transductive learning* [50, 11], illustrated in the top row of Figure 1. In transductive learning, a single algorithm consumes both labeled data, $((x_i, y_i))_{i=1}^n$, and a set of input queries for which predictions are desired, $(x^{(j)})_j$, and produces predictions $(\hat{y}^{(j)})_j$ for each query. In general, however, we do not know queries *a priori*, and instead, we want an inductive function that makes predictions online, as queries arrive. To obtain such an online prediction function from a transductive system, we would need to take the training data and the single unlabeled query and encapsulate the entire transductive learning procedure inside the prediction function itself. This strategy would achieve our objective of taking $x$ into account at prediction time but would be computationally much too slow [12].

This approach for combining induction and transduction would reuse the same training data and objective for each prediction, only changing the single unlabeled query. Consequently, it would perform extremely similar computations for each prediction. Therefore, we propose to effectively reuse the shared computations and find a "meta-hypothesis" that can then be efficiently adapted to each query. As shown in the third row of Figure 1, we propose to first run regular supervised learning to obtain parameters $\hat{\theta}$. Then, given a query input $x$, we fine-tune $\hat{\theta}$ on an unsupervised loss $\mathcal{L}^{\text{tailor}}$ to obtain cus-

**Algorithm 1 MAM**mo**T**h: **M**odel-**A**gnostic **M**eta-**T**ailoring

**Subroutine** *Training(*$f$, $\mathcal{L}^{\text{sup}}$, $\lambda_{sup}$, $\mathcal{L}^{\text{tailor}}$, $\lambda_{tailor}$, $\mathcal{D}_{train}$,$b$*)*
  randomly initialize $\theta$
  **while** *not done* **do**
    Sample batch of samples $(x_i, y_i) \sim \mathcal{D}_{train}$
    **forall** $(x_i, y_i)$ **do**
      $\theta_{x_i} = \theta - \lambda_{tailor} \nabla_\theta \mathcal{L}^{\text{tailor}}(\theta, x_i)$ `// Inner step with tailor loss`
    $\theta = \theta - \lambda_{sup} \nabla_\theta \sum_{(x_i,y_i)} \mathcal{L}^{\text{sup}}\left(f_{\theta_{x_i}}(x_i), y_i\right)$ `// Outer step with supervised loss`
  **return** $\theta$

tomized parameters $\theta_x$ and use them to make the final prediction: $f_{\theta_x}(x)$. We call this process *tailoring*, because we adapt the model to each particular input for a customized fit. Notice that tailoring optimizes the loss at the query input, eliminating the generalization gap on the unsupervised auxiliary loss.

**Meta-tailoring** Since we will be applying tailoring at prediction time, it is natural to incorporate this adaptation during training, resulting in a two-layer optimization similar to those used in meta-learning. Because of this similarity, we call this process *meta-tailoring*, illustrated in the bottom row of Figure 1. Now, rather than letting $\hat{\theta}$ be the direct minimizer of the supervised loss, we set it to

$$\hat{\theta} \in \arg\min_{\theta} \sum_{i=1}^{n} \mathcal{L}^{\text{sup}}(f_{\tau(\theta, \mathcal{L}^{\text{tailor}}, x_i)}(x_i), y_i).$$

Here, the inner loop optimizes the unsupervised tailoring loss $\mathcal{L}^{\text{tailor}}$ and the outer loop optimizes the supervised task loss $\mathcal{L}^{\text{sup}}$. Notice that now the outer process optimizes the only objective we care, $\mathcal{L}^{\text{sup}}$, instead of a proxy combination of $\mathcal{L}^{\text{sup}}$ and $\mathcal{L}^{\text{unsup}}$. At the same time, we learn to leverage $\mathcal{L}^{\text{tailor}}$ in the inner loop to affect the model before making the final prediction, both during training and evaluation. Adaptation is especially clear in the case of a single gradient step, as in MAML [19]. We show its translation, MAMmoTh (Model-Agnostic Meta-Tailoring), in algorithm 1.

In many settings, we want to make predictions for a large number of queries in a (mini-)batch. While MAMmoTh adapts to every input separately, it can only be run efficiently in parallel in some deep learning frameworks, such as JAX [10]. Inspired by conditional normalization (CN) [18] we propose CNGRAD, which adds element-wise affine transformations to our model and only adapts the added parameters in the inner loop. This allows us to independently *tailor* the model for multiple inputs in parallel. We prove theoretically, in Sec. 4, and provide experimental evidence, in Sec. 5.1, that optimizing these parameters alone has enough capacity to minimize a large class of tailoring losses.

**Relation between (meta-)tailoring, fine-tuning transfer, and meta-learning** Fine-tuning pre-trained networks is a fruitful method of transferring knowledge from large corpora to smaller related datasets [17]. This allows us to reuse features on related tasks or for different distributions of the same task. When the data we want to adapt to is unlabeled, we must use unsupervised losses. This can be useful to adapt to changes of task [16], from simulated to real data [52], or to new distributions [46].

Tailoring performs unsupervised fine-tuning and is, in this sense, similar to test-time training(TTT) [46] for a single sample, which adapts to distribution shifts. However, tailoring is applied to a single query; not to a data set that captures distribution shift, where batched TTT sees most of its benefits. Thus, whereas regular fine-tuning benefits from more adaptation data, tailoring would be hindered by adapting simultaneously to more data. This is because tailoring aims at building a custom model for each query to ensure the network satisfies a particular inductive bias. Customizing the model to multiple samples makes it harder, not easier. We show this in Figure 2, where TTT with 6400 samples performs worse than tailoring with a single sample. Furthermore, tailoring adapts to each query one by one, not globally from training data to test data. Therefore, it also makes sense to do tailoring on training queries (i.e., meta-tailoring).

Meta-tailoring has the same two-layer optimization structure as meta-learning. More concretely, it can be understood as the extreme case of meta-learning where each single-query prediction is its own task. However, whereas meta-learning tasks use one loss and different examples for the inner and outer loop, meta-tailoring tasks use one example and different losses for each loop ($\mathcal{L}^{\text{tailor}}$, $\mathcal{L}^{\text{sup}}$). We emphasize that meta-tailoring does not operate in the typical multi-task meta-learning setting. Instead, we are leveraging techniques from meta-learning for the classical single-task setting.

**Contributions** In summary, our contributions are:

1. Introducing *tailoring*, a new framework for encoding inductive biases by minimizing unsupervised losses at prediction time, with theoretical guarantees and broad potential applications.
2. Formulating *meta-tailoring*, which adjusts the outer objective to optimize only the task loss, and developing a new algorithm, CNGRAD, for efficient meta-tailoring.
3. Demonstrating *meta-tailoring* in 3 domains: encoding hard and soft conservation laws in physics prediction problems (Sec. 5.1 and Sec. 5.2), enhancing resistance to adversarial examples by increasing local smoothness at prediction time (Sec. 5.4), and improving prediction quality both theoretically (Sec. 3.1) and empirically (Sec. 5.3) by tailoring with a contrastive loss.

## 2 Related work

Tailoring is inspired by transductive learning. However, transductive methods, because they operate on a batch of unlabeled queries, are allowed to make use of the underlying distributional properties of those queries, as in semi-supervised learning [12]. In contrast, tailoring does the bulk of the computations before receiving any query; vastly increasing efficiency. Similar to tailoring, local learning [9] also has input-dependent parameters. However, it uses similarity in raw input space to select a few labeled data points and builds a local model instead of reusing the global prior learned across the whole data. Finally, some methods [21, 33] in meta-learning propagate predictions along the test samples in a semi-supervised transductive fashion.

Similar to tailoring, there are other learning frameworks that perform optimization at prediction time for very different purposes. Among those, energy-based models do generative modeling [2, 27, 32] by optimizing the hidden activations of neural networks, and other models [4, 49] learn to solve optimization problems by embedding optimization layers in neural networks. In contrast, tailoring optimizes the parameters of the model, not the hidden activations or the output.

As discussed in the introduction, unsupervised fine-tuning methods have been proposed to adapt to different types of variations between training and testing. Sun et al. [46] propose to adapt to a change of distribution with few samples by unsupervised fine-tuning at test-time, applying it with a loss of predicting whether the input has been rotated. Zhang et al. [54] build on it to adapt to group distribution shifts with a learned loss. Other methods in the few-shot meta-learning setting exploit test samples of a new task by minimizing either entropy [16] or a learned loss [5] in the inner optimization. Finally, Wang et al. [51] use entropy in the inner optimization to adapt to large-scale variations in image segmentation. In contrast, we propose (meta-)tailoring as a general effective way to impose inductive biases in the classic machine learning setting. Whereas in the aforementioned methods, adaptation happens from training to testing, we independently adapt to every single query.

Meta-learning [44, 7, 48, 28] has the same two-level optimization structure as meta-tailoring but focuses on multiple prediction tasks. As shown in Alg. 1 for MAML [19], most optimization-based meta-learning algorithms can be converted to meta-tailoring. Similar to CNGRAD, there are other meta-learning methods whose adaptations can be batched [40, 3]. Among these, [55, 41] train FiLM networks [39] to predict custom conditional normalization (CN) layers for each task. By optimizing the CN layers directly, CNGRAD is simpler, while remaining provably expressive (section 4). CNGrad can also start from a trained model by initializing the CN layers to the identity function.

## 3 Theoretical motivations of meta-tailoring

In this section, we study the potential advantages of meta-tailoring from the theoretical viewpoint, formalizing the intuitions conveyed in the introduction. By acting symmetrically during training and prediction time, meta-tailoring allows us to closely relate its training and expected losses, whereas tailoring alone does not have the same guarantees. First, we analyze the particular case of a contrastive tailoring loss. Then, we will generalize the guarantees to other types of tailoring losses.

### 3.1 Meta-tailoring with a contrastive tailoring loss

*Contrastive learning* [24] has seen significant successes in problems of semi-supervised learning [37, 26, 13]. The main idea is to create multiple versions of each training image and learn a representation in which variations of the same image are close while variations of different images are far apart. Typical augmentations involve cropping, color distortions, and rotation. We show theoretically that, under reasonable conditions, meta-tailoring using a particular contrastive loss $\mathcal{L}_{\text{cont}}$ as $\mathcal{L}^{\text{tailor}} = \mathcal{L}_{\text{cont}}$ helps us improve generalization errors in expectation compared with performing classical inductive learning.

When using meta-tailoring, we define $\theta_{x,S}$ to be the $\theta_x$ obtained with a training dataset $S = ((x_i, y_i))_{i=1}^n$ and tailored with the contrastive loss at the prediction point $x$. Theorem 1 provides an upper bound on the expected supervised loss $\mathbb{E}_{x,y}[\mathcal{L}^{\text{sup}}(f_{\theta_{x,S}}(x), y)]$ in terms of the expected contrastive loss $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{x,S})]$ (analyzed in App. B), the empirical supervised loss $\frac{1}{n}\sum_{i=1}^n \mathcal{L}^{\text{sup}}(f_{\theta_{x_i,S}}(x_i), y_i)$ of meta-tailoring, and its uniform stability $\zeta$. Theorem 6 (App. C) provides a similar bound with the Rademacher complexity [6] $\mathcal{R}_n(\mathcal{L}^{\text{sup}} \circ \mathcal{F})$ of the set $\mathcal{L}^{\text{sup}} \circ \mathcal{F}$, instead of using the uniform stability $\zeta$. Proofs of all results in this paper are deferred to App. C.

**Definition 1.** Let $S = ((x_i, y_i))_{i=1}^n$ and $S' = ((x'_i, y'_i))_{i=1}^n$ be any two training datasets that differ by a single point. Then, a meta-tailoring algorithm $S \mapsto f_{\theta_{\mathbf{x},S}}(x)$ is *uniformly $\zeta$-stable* if $\forall (x, y) \in \mathcal{X} \times \mathcal{Y},\ |\mathcal{L}^{\text{sup}}(f_{\theta_{\mathbf{x},S}}(x), y) - \mathcal{L}^{\text{sup}}(f_{\theta_{\mathbf{x},S'}}(x), y)| \leq \frac{\zeta}{n}$.

**Theorem 1.** *Let $S \mapsto f_{\theta_{\mathbf{x},S}}(x)$ be a uniformly $\zeta$-stable meta-tailoring algorithm. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over an i.i.d. draw of $n$ i.i.d. samples $S = ((x_i, y_i))_{i=1}^n$, the following holds: for any $\kappa \in [0, 1]$, $\mathbb{E}_{x,y}[\mathcal{L}^{\text{sup}}(f_{\theta_{\mathbf{x},S}}(x), y)] \leq \kappa \mathbb{E}_x\left[\mathcal{L}_{\text{cont}}(x, \theta_{\mathbf{x},S})\right] + (1 - \kappa)\mathcal{J}$, where $\mathcal{J} = \frac{1}{n}\sum_{i=1}^n \mathcal{L}^{\text{sup}}(f_{\theta_{\mathbf{x}_i,S}}(x_i), y_i) + \frac{\zeta}{n} + (2\zeta + c)\sqrt{(\ln(1/\delta))/(2n)}$, and $c$ is the upper bound on the per-sample loss as $\mathcal{L}^{\text{sup}}(f_\theta(x), y) \leq c$.*

In the case of regular inductive learning, we get a bound of the exact same form, except that we have a single $\theta$ instead of a $\theta_{\mathbf{x}}$ tailored to each input $x$. This theorem illustrates the effect of meta-tailoring on contrastive learning, with its potential reduction of the expected contrastive loss $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{x,S})]$. In classic induction, we may aim to minimize the empirical contrastive loss $\frac{1}{\bar{n}}\sum_{i=1}^{\bar{n}} \mathcal{L}_{\text{cont}}(x_i, \theta)$ with $\bar{n}$ potentially unlabeled training samples, which incurs the additional generalization error of $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{x,S})] - \frac{1}{\bar{n}}\sum_{i=1}^{\bar{n}} \mathcal{L}_{\text{cont}}(x_i, \theta)$. In contrast, meta-tailoring can avoid this extra generalization error by directly minimizing a custom $\theta_{\mathbf{x}}$ on each $x$: $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{\mathbf{x},S})]$.

In the case where $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{x,S})]$ is left large (e.g., due to large computational cost), Theorem 1 still illustrates competitive generalization bounds of meta-tailoring with small $\kappa$. For example, with $\kappa = 0$, it provides generalization bounds with the uniform stability for meta-tailoring algorithms. Even then, the bounds are not equivalent to those of classic induction, and there are potential benefits of meta-tailoring, which are discussed in the following section with a more general setting.

### 3.2 Meta-tailoring with general tailoring losses

The benefits of meta-tailoring go beyond contrastive learning: below we provide guarantees for meta-tailoring with arbitrary pairs of tailoring loss $\mathcal{L}^{\text{tailor}}(x, \theta)$ and supervised loss $\mathcal{L}^{\text{sup}}(f_\theta(x), y)$.

**Remark 1.** *For any function $\varphi$ such that $\mathbb{E}_{x,y}[\mathcal{L}^{\text{sup}}(f_\theta(x), y)] \leq \mathbb{E}_x[\varphi(\mathcal{L}^{\text{tailor}}(x, \theta))]$, Theorems 1 and 6 hold with the map $\mathcal{L}_{\text{cont}}$ being replaced by the function $\varphi \circ \mathcal{L}^{\text{tailor}}$.*

This remark shows the benefits of meta-tailoring through its effects on three factors: the expected unlabeled loss $\mathbb{E}_x[\varphi(\mathcal{L}^{\text{tailor}}(x, \theta_{\mathbf{x},S}))]$, uniform stability $\zeta$, and the Rademacher complexity $\mathcal{R}_n(\mathcal{L}^{\text{sup}} \circ \mathcal{F})$. It is important to note that meta-tailoring can directly minimize the expected unlabeled loss $\mathbb{E}_x[\varphi(\mathcal{L}^{\text{tailor}}(x, \theta_{\mathbf{x},S}))]$, whereas classic induction can only minimize its empirical version, which results in the additional generalization error on the difference between the expected unlabeled loss and its empirical version. For example, if $\varphi$ is monotonically increasing and $\mathcal{L}^{\text{tailor}}(x, \theta)$ represents the physical constraints at each input $x$ (as in the application in section 5.1), then classic induction requires a neural network trained to conserve energy at the *training* points to generalize to also conserve it at *unseen* (e.g., testing) points. Meta-tailoring avoids this requirement by directly minimizing violations of energy conservation at each point at prediction time.

Meta-tailoring can also improve the *parameter stability* $\zeta_\theta$ defined such that $\forall (x, y) \in \mathcal{X} \times \mathcal{Y}, \|\theta_{x,S} - \theta_{x,S'}\| \leq \frac{\zeta_\theta}{n}$, for all $S, S'$ differing by a single point. When $\theta_{x,S} = \hat{\theta}_S - \lambda \nabla \mathcal{L}^{\text{tailor}}(x, \hat{\theta}_S)$, we obtain an improvement on the parameter stability $\zeta_\theta$ if $\nabla \mathcal{L}^{\text{tailor}}(x, \hat{\theta}_S)$ can pull $\hat{\theta}_S$ and $\hat{\theta}_{S'}$ closer so that $\|\theta_{x,S} - \theta_{x,S'}\| < \|\hat{\theta}_S - \hat{\theta}_{S'}\|$, which is ensured, for example, if $\|\cdot\| = \|\cdot\|_2$ and $\text{cos\_dist}(v_1, v_2)\frac{\|v_1\|}{\|v_2\|} > \frac{1}{2}$ where $\text{cos\_dist}(v_1, v_2)$ is the cosine similarity of $v_1$ and $v_2$, with $v_1 = \hat{\theta}_S - \hat{\theta}_{S'}$, $v_2 = \lambda(\nabla \mathcal{L}^{\text{tailor}}(x, \hat{\theta}_S) - \nabla \mathcal{L}^{\text{tailor}}(x, \hat{\theta}_{S'}))$ and $v_2 \neq 0$. Here, the uniform stability $\zeta$ and the parameter stability $\zeta_\theta$ are closely related as $\zeta \leq C\zeta_\theta$, where $C$ is the upper bound on the Lipschitz constants of the maps $\theta \mapsto \mathcal{L}^{\text{sup}}(f_\theta(x), y)$ over all $(x, y) \in \mathcal{X} \times \mathcal{Y}$ under the norm $\|\cdot\|$, since $|\mathcal{L}^{\text{sup}}(f_{\theta_{x,S}}(x), y) - \mathcal{L}^{\text{sup}}(f_{\theta_{x,S'}}(x), y)| \leq C\|\theta_{x,S} - \theta_{x,S'}\| \leq \frac{C\zeta_\theta}{n}$.

**Algorithm 2** CNGRAD for meta-tailoring

**Subroutine** *Training(*$f$*,* $\mathcal{L}^{\text{sup}}$*,* $\lambda_{sup}$*,* $\mathcal{L}^{\text{tailor}}$*,* $\lambda_{tailor}$*, steps,*$\mathcal{D}_{train}$*,b)* `// Only in meta-tailoring`

  randomly initialize $w$ `// All parameters except` $\gamma, \beta$`; trained in outer loop`

  **while** *not done* **do**

    $X, Y \sim^b \mathcal{D}_{train}; grad_w = 0$ `// Sample batch; initialize outer gradient`

    $\gamma_0 = \mathbf{1}_{b,\sum_l m_l}; \beta_0 = \mathbf{0}_{b,\sum_l m_l}$ `// Initialize CN layers to the identity`

    **for** $1 \leq s \leq steps$ **do**

      $\gamma_s = \gamma_{s-1} - \lambda_{tailor} \nabla_\gamma \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$ `// Inner step w.r.t.` $\gamma$

      $\beta_s = \beta_{s-1} - \lambda_{tailor} \nabla_\beta \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$ `// Inner step w.r.t.` $\beta$

      $\gamma_s, \beta_s = \gamma_s.detach(), \beta_s.detach()$ `// Only in` $1^{st}$ `order CNGrad`

      $grad_w = grad_w + \nabla_w \mathcal{L}^{\text{sup}}\left(f_{w,\gamma_s,\beta_s}(X), Y\right)$ `// Outer gradient w.r.t.` $w$

    $w = w - \lambda_{sup} grad_w$ `// Apply outer step after all inner steps`

  **return** $w$

**Subroutine** *Prediction(*$f$*,* $w$*,* $\mathcal{L}^{\text{tailor}}$*,* $\lambda$*, steps,* $X$*)* `// Both in meta-tailoring & tailoring`

  $\gamma_0 = \mathbf{1}_{X.shape[0],\sum_l m_l}; \beta_0 = \mathbf{0}_{X.shape[0],\sum_l m_l}$

  **for** $1 \leq s \leq steps$ **do**

    $\gamma_s = \gamma_{s-1} - \lambda \nabla_\gamma \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$

    $\beta_s = \beta_{s-1} - \lambda \nabla_\beta \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$

  **return** $f_{w,\gamma_{steps},\beta_{steps}}(X)$

## 4 CNGRAD: a simple algorithm for expressive, efficient (meta-)tailoring

In this section, we address the issue of using (meta-)tailoring for efficient GPU computations. Although possible in JAX [10], efficiently parallelizing MAMmoTh across inputs is not possible in other frameworks. To overcome this issue, building on CAVIA [55] and WarpGrad [20], we propose CNGRAD which adapts only *conditional normalization* parameters and enables efficient GPU computations for (meta-)tailoring. CNGRAD can also be used in meta-learning, providing a parallelizable alternative to MAML (see App. D).

As done in batch-norm [30] after element-wise normalization, we can implement an element-wise affine transformation with parameters $(\gamma, \beta)$, scaling and shifting the output $h_k^{(l)}(x)$ of each $k$-th neuron at the $l$-th hidden layer independently: $\gamma_k^{(l)} h_k^{(l)}(x) + \beta_k^{(l)}$. In conditional normalization, Dumoulin et al. [18] train a collection of $(\gamma, \beta)$ in a multi-task fashion to learn different tasks with a single network. CNGRAD brings this concept to the meta-learning and (meta-)tailoring settings and adapts the affine parameters $(\gamma, \beta)$ to each query. For meta-tailoring, the inner loop minimizes the tailoring loss at an input $x$ by adjusting the affine parameters and the outer optimization adapts the rest of the network. Similar to MAML [19], we implement a first-order version, which does not backpropagate through the optimization, and a second-order version, which does. CNGRAD efficiently parallelizes computations of multiple tailored models because the adapted parameters only require element-wise multiplications and additions. See Alg. 2 for the pseudo-code.

CNGRAD is widely applicable since the adaptable affine parameters can be added to any hidden layer and only represent a tiny portion of the network (empirically, around $1\%$). Moreover, we can see that, under realistic assumptions, we can minimize the inner tailoring loss using only the affine parameters. To analyze properties of these adaptable affine parameters, let us decompose $\theta$ into $\theta = (w, \gamma, \beta)$, where $w$ contains all the weight parameters (including bias terms), and the $(\gamma, \beta)$ contains all the affine parameters. Given an arbitrary function $(f_\theta(x), x) \mapsto \ell_{\text{tailor}}(f_\theta(x), x)$, let $\mathcal{L}^{\text{tailor}}(x, \theta) = \sum_{i=1}^{n_g} \ell_{\text{tailor}}(f_\theta(g^{(i)}(x)), x)$, where $g^{(1):(n_g)}$ are arbitrary input augmentation functions at prediction time.

Corollary 1 states that for any given $\hat{w}$, if we add any non-degenerate Gaussian noise $\delta$ as $\hat{w} + \delta$ with zero mean and any variance on $\delta$, the global minimum value of $\mathcal{L}^{\text{tailor}}$ w.r.t. all parameters $(w, \gamma, \beta)$ can be achieved by optimizing only the affine parameters $(\gamma, \beta)$, with probability one. In other words, the CN parameters $(\gamma, \beta)$ have enough capacity to optimize optimize the inner tailoring loss.

**Corollary 1.** *Under the assumptions of Theorem 2, for any* $\hat{w} \in \mathbb{R}^d$*, with probability one over randomly sampled* $\delta \in \mathbb{R}^d$ *accordingly to any non-degenerate Gaussian distribution, the following holds:* $\inf_{w,\gamma,\beta} \mathcal{L}^{\text{tailor}}(x, w, \gamma, \beta) = \inf_{\gamma,\beta} \mathcal{L}^{\text{tailor}}(x, \hat{w} + \delta, \gamma, \beta)$ *for any* $x \in \mathcal{X}$*.*

The assumption and condition in theorem 2 are satisfied in practice (see App. A). Therefore, CNGRAD is a practical and computationally efficient method to implement (meta-)tailoring.

| **Method** | loss | relative |
|---|---|---|
| Inductive learning | .041 | - |
| Opt. output(50 st.) | .041 | $(0.7 \pm 0.1)\%$ |
| 6400-s. TTT(50 st.) | .040 | $(3.6 \pm 0.2)\%$ |
| Tailoring(1 step) | .040 | $(1.9 \pm 0.2)\%$ |
| Tailoring(5 steps) | .039 | $(6.3 \pm 0.3)\%$ |
| Tailoring(10 st.) | .038 | $(7.5 \pm 0.1)\%$ |
| Meta-tailoring(0 st.) | .030 | $(26.3 \pm 3.3)\%$ |
| Meta-tailoring(1 st.) | .029 | $(29.9 \pm 3.0)\%$ |
| Meta-tailoring(5 st.) | .027 | $(35.3 \pm 2.6)\%$ |
| Meta-tailoring(10 s.) | .026 | $(36.0 \pm 2.6)\%$ |

Table 1: Test MSE loss for different methods; the second column shows the relative improvement over basic inductive supervised learning. The test-time training (TTT) baseline uses a full batch of 6400 test samples to adapt, not allowed in regular SL. With a few gradient steps, tailoring significantly over-performs all baselines. Meta-tailoring improves even further, with $35\%$ improvement.

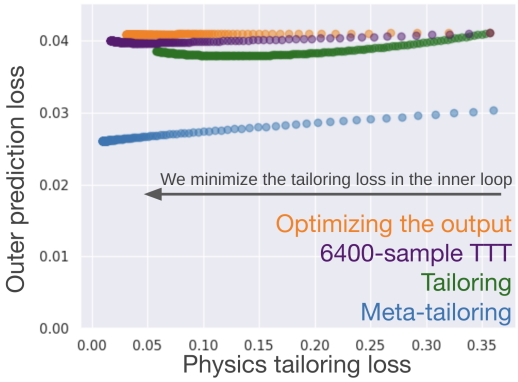


Figure 2: Optimization at prediction time on the planet data; each path going from right to left as we minimize the physics tailoring loss. We use a small step size to illustrate the path. Tailoring and the two baselines only differ in their test-time computations, thus sharing their starts. Meta-tailoring has a lower starting loss, faster optimization, and no overfitting during tailoring.

# 5 Experiments

## 5.1 Tailoring to impose symmetries and constraints at prediction time

Exploiting invariances and symmetries is an established strategy for increasing performance in ML. During training, we can regularize networks to satisfy specific criteria; but this does not guarantee they will be satisfied outside the training dataset [45]. (Meta-)tailoring provides a general solution to this problem by adapting the model to satisfy the criteria at prediction time. We demonstrate the use of tailoring to enforce physical conservation laws for predicting the evolution of a 5-body planetary system. This prediction problem is challenging, as $m$-body systems become chaotic for $m > 2$. We generate a dataset with positions, velocities, and masses of all 5 bodies as inputs and the changes in position and velocity as targets. App. E further describes the dataset.

Our model is a 3-layer feed-forward network. We tailor it by taking the original predictions and adapting the model using the tailoring loss given by the $L_1$ loss between the whole system's initial and final energy and momentum. Note that ensuring this conservation does not guarantee better performance: predicting the input as the output conserves energy and momentum perfectly, but it is not correct.

While tailoring adapts some parameters in the network to improve the tailoring loss, an alternative for enforcing conservation would be to adapt the output $y$ value directly. Table 1 compares the predictive accuracy of inductive learning, direct output optimization, and both tailoring and meta-tailoring, using varying numbers of gradient steps. Tailoring is more effective than adapting the output, as the parameters provide a prior on what changes are more natural. For meta-tailoring, we try both first-order and second-order versions of CNGRAD. The first-order gave slightly better results, possibly because it was trained with a higher tailor learning rate ($10^{-3}$) with which the second-order version was unstable (we thus used $10^{-4}$). More details can be found in App. E.

Finally, meta-tailoring without any query-time tailoring steps already performs much better than the original model, even though both have almost the same number of parameters and can overfit the dataset. We conjecture meta-tailoring training adds an inductive bias that guides optimization towards learning a more generalizable model. Fig. 2 shows prediction-time optimization paths.

## 5.2 Tailoring to softly encourage inductive biases

A popular way of encoding inductive biases is with clever network design to make predictions translation equivariant (CNNs), permutation equivariant (GNNs), or conserve energy [23]. However, if an inductive bias is only partially satisfied, such approaches overly constrain the function class. Instead, tailoring can softly impose this bias by only fine-tuning the tailoring loss for a few steps.

We showcase this in the real pendulum experiment used by Hamiltonian Neural Networks (HNNs) [23]. HNNs have energy conservation built-in and easily improve a vanilla MLP. We meta-tailor this vanilla MLP with energy conservation without changing its architecture. Meta-tailoring significantly improves over the baseline and HNNs, since it can encode the *imperfect* energy conservation of real systems. We compare results in Fig. 3 and provide extra details in App. F. Note that, with inexact losses, fully enforcing them provides sub-optimal results. Thus, we pick the tailoring learning rate that results in the lowest long-term prediction loss during training.

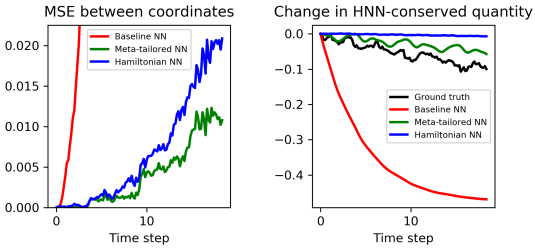


Figure 3: By softly encouraging energy conservation, meta-tailoring improves over models that don't and models that fully impose it.

### 5.3 Tailoring with a contrastive loss for image classification

Following the setting described in section 3.2, we provide experiments on the CIFAR-10 dataset [31] by building on SimCLR [13]. SimCLR trains a ResNet-50 [25] $f_\theta(\cdot)$ coupled to a small MLP $g(\cdot)$ such that the outputs of two augmentations of the same image $x_i, x_j \sim \mathcal{T}(x)$ agree; i.e. $g(f_\theta(x_i)) \approx g(f_\theta(x_j))$. This is done by training $g(f(\cdot))$ to recognize one augmentation from the other among a big batch of candidates with the cross-entropy loss. To show that the unsupervised training of $f_\theta$ provides a useful representation, SimCLR trains a single linear layer on top of it, $\phi(f_\theta(\cdot))$, achieving good classification results.

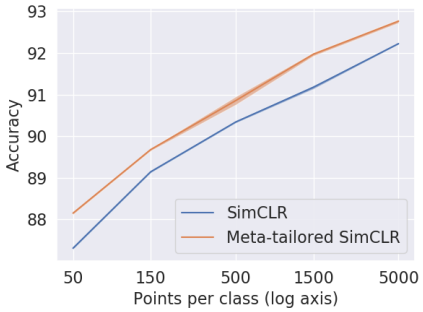


Figure 4: Meta-tailoring the linear layer with the contrastive loss results in consistent accuracy gains between $0.5\%$ and $0.8\%$. This is approximately the same gain as that of doubling the amount of labeled data (note the logarithmic x-axis).

We now observe that we can tailor $f_\theta$ at prediction-time by optimizing $g(f_{\theta_x}(x))$, which quantifies the agreement between different augmentations of the same input; thus 'learning' about its particularities. To make the image classification prediction, we feed the final tailored representation to the linear layer: $\phi(f_{\theta_x}(x))$. To match the evaluation from SimCLR, we do not redo SimCLR's unsupervised learning, which provides $\theta$. The meta-tailoring outer loop trains $\phi$ to take the tailored representations $f_{\theta_x}(x)$ instead of the original $f_\theta(x)$. Thus, $\theta$ is unsupervisedly fine-tuned in the prediction function leading to $\theta_x$, but never supervisedly trained as this would break the evaluation protocol (in meta-tailoring's favor). We also implement a TTT [46] baseline with their original rotation-prediction loss. Moreover, TTT modifies $\theta_x$ at test time, but does not take this adaptation into account when training $\phi$ (see App. G for more details). TTT worsened base SimCLR despite significant hyper-parameter tuning. We conjecture this is because TTT was designed for OOD generalization, not in-distribution. In contrast, as shown in Fig. 4, we observe that meta-tailoring provides improvements over base SimCLR equivalent to doubling the amount of labeled data.

### 5.4 Tailoring for robustness against adversarial examples

Neural networks are susceptible to adversarial examples [8, 47]: targeted small perturbations of an input can cause the network to misclassify it. One approach is to make the prediction function smooth via adversarial training [34]; however, this only ensures smoothness in the training points. Constraining the model to be smooth everywhere makes it lose capacity. Instead, (meta-)tailoring asks for smoothness *a posteriori*, only on a specific query.

We apply meta-tailoring to robustly classifying CIFAR-10 [31] and ImageNet [15] images, tailoring predictions so that they are locally smooth. This is similar to VAT [36] but instead optimizes the loss within the prediction function, not as an auxiliary loss. Inspired by the notion of adversarial examples being caused by predictive, but non-robust, features [29], we meta-tailor our model by enforcing smoothness on the vector of features of the penultimate layer (denoted $g_\theta(x)$):

$$\mathcal{L}^{\text{tailor}}(x, \theta) = \mathbb{E}[\text{cos\_dist}(g_\theta(x), g_\theta(x + \delta))], \delta \sim N(0, \nu^2),$$

| $\sigma$ | **Method** | 0.0 | 0.5 | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | ACR |
|---|---|---|---|---|---|---|---|---|---|
| 0.25 | (Inductive) Randomized Smoothing | 0.67 | 0.49 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.470 |
| | Meta-tailored Randomized Smoothing | **0.72** | **0.55** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | **0.494** |
| 0.50 | (Inductive) Randomized Smoothing | 0.57 | 0.46 | 0.37 | 0.29 | 0.00 | 0.00 | 0.00 | 0.720 |
| | Meta-tailored Randomized Smoothing | 0.66 | 0.54 | **0.42** | **0.31** | 0.00 | 0.00 | 0.00 | **0.819** |
| 1.00 | (Inductive) Randomized Smoothing | 0.44 | 0.38 | 0.33 | 0.26 | 0.19 | 0.15 | 0.12 | 0.863 |
| | Meta-tailored Randomized Smoothing | 0.52 | 0.45 | 0.36 | **0.31** | **0.24** | **0.20** | **0.15** | **1.032** |

Table 2: Fraction of points with certificate above different radii for ImageNet. Meta-tailoring improves average certification radius (ACR) of different models by $5.1\%, 13.8\%, 19.6\%$. Results for Randomized Smoothing come from [53].

We build on Cohen et al. [14], who developed a method for certifying the robustness of a model via randomized smoothing (RS). RS samples points from a Gaussian $N(x, \sigma^2)$ around the query and, if there is enough agreement in classification, it provides a certificate that a small perturbation cannot adversarially modify the query to have a different class. We show that meta-tailoring improves the original RS method, testing for $\sigma = 0.25, 0.5, 1.0$. We use $\nu = 0.1$ for all experiments. We initialized with the weights of Cohen et al. [14] by leveraging that CNGRAD can start from a pre-trained model by initializing the extra affine layers to the identity. Finally, we use $\sigma' = \sqrt{\sigma^2 - \nu^2} \approx 0.23, 0.49, 0.995$ so that the points used in our tailoring loss come from $N(x, \sigma^2)$.

Table 7 shows our results on CIFAR-10 where we improve the average certification radius (ARC) by $8.6\%, 10.4\%, 19.2\%$ respectively. In table 2, we show results on Imagenet where we improve the ARC by $5.1\%, 13.8\%, 19.6\%$ respectively. We chose to meta-tailor the RS method because it represents a strong standard in certified adversarial defenses, but we note that there have been advances on RS that sometimes achieve better results than those presented here [53, 43], see App. I. However, it is likely that meta-tailoring could also improve these methods.

These experiments only scratch the surface of what tailoring allows for adversarial defenses: usually, the adversary looks at the model and gets to pick a particularly bad perturbation $x + \delta$. With tailoring, the model responds, by changing to weights $\theta_{x+\delta}$. This leads to a game, where both weights and inputs are perturbed, similar to $\max_{|\delta|<\epsilon_x} \min_{|\Delta|<\epsilon_\theta} \mathcal{L}^{\text{sup}}\left(f_{\theta+\Delta}(x+\delta), y\right)$. However, since we don't get to observe $y$; we optimize the weight perturbation by minimizing $\mathcal{L}^{\text{tailor}}$ instead.

# 6 Discussion

## 6.1 Broader Impact

Improving adversarial robustness: having more robust and secure ML systems is mostly a positive change. However, improving adversarial defenses could also go against privacy preservation, like the use of adversarial patches to gain anonymity from facial recognition. Encoding desirable properties: By optimizing an unsupervised loss for the particular query we care about, it is easier to have guarantees on the prediction. In particular, there could be potential applications for fairness, where the unsupervised objective could enforce specific criteria at the query or related inputs. More research needs to be done to make this assertion formal and practical. Potential effect on privacy: tailoring specializes the model to each input. This could have an impact on privacy. Intuitively, the untailored model can be less specialized to each input, lowering the individual information from each training point contained in the model. However, tailored predictions extract more information about the queries, from which more personal information could be leaked.

## 6.2 Limitations

Tailoring provides a framework for encoding a wide array of inductive biases, but these need to be specified as a formula by the user. For instance, it would be hard to programatically describe tailoring losses in raw pixel data, such as mass conservation in pixel space. Tailoring also incurs an extra time cost at prediction time, since we make an inner optimization inside the prediction function. However, as shown in Table 1, meta-tailoring often achieves better results than inductive learning even without adaptation at test-time, enabling better predictions at regular speed during test-time. This is due to meta-tailoring leading to better training. Moreover, optimization can be sped up by only tailoring the last layers, as discussed in App. D. Finally, to the best of our knowledge using MAMmoTh for meta-tailoring would be hard to parallelize in PyTorch [38] and Tensorflow [1]; we

proposed CNGRAD to make it easy and efficient. JAX[10], which handles per-example weights, makes parallelizing tailoring effortless.

Theory in Sec. 3 applies only to meta-tailoring. Unlike tailoring (and test-time training), meta-tailoring performs the same computations at training and testing time, which allows us to prove the results. Theorem 2 proves that optimizing the CN layers in CNGRAD has the same expressive power as optimizing all the layers for the inner (not outer) loss. However, it does not guarantee that gradient descent will find the appropriate optima. The study of such guarantee is left for future work.

### 6.3 Conclusion

We have presented *tailoring*, a simple way of embedding a powerful class of inductive biases into models, by minimizing unsupervised objectives at prediction time. Tailoring leverages the generality of auxiliary losses and improves them in two ways: first, it eliminates the generalization gap on the auxiliary loss by optimizing it on the query point; second, tailoring only minimizes task loss in the outer optimization and the tailoring loss in the inner optimization. This results in the model optimizing the only objective we care about in the outer loop, instead of a proxy loss. Beyond inductive biases, tailoring shows that model adaptation is useful even when test queries comes from the same distribution as the training data. This suggests one can improve models by performing prediction-time optimization, trading off large offline data&compute efforts with small online computations.

Tailoring is broadly applicable, as one can vary the model, the unsupervised loss, and the task loss. We show its applicability in three diverse domains: physics prediction time-series, contrastive learning, and adversarial robustness. We also provide a simple algorithm, CNGRAD, to make meta-tailoring practical with little additional code. Currently, most unsupervised or self-supervised objectives are optimized in task-agnostic ways; without taking into account the supervised downstream task. Instead, meta-tailoring provides a generic way to make these objectives especially useful for each application. It does so by learning how to best leverage the unsupervised loss to perform well on the final task we care about.

## Acknowledgments and Disclosure of Funding

We would like to thank Kelsey Allen, Marc de la Barrera, Jeremy Cohen, Dylan Doblar, Chelsea Finn, Sebastian Flennerhag, Jiayuan Mao, Josh Tenenbaum, and Shengtong Zhang for insightful discussions. We would also like to thank Clement Gehring for his help with deploying the experiments and Lauren Milechin for her help with leveraging the MIT supercloud platform [42].

We gratefully acknowledge support from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from the Honda Research Institute, from MIT-IBM Watson Lab; and from SUTD Temasek Laboratories. We also acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the reported research results. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

## References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[2] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[3] Alet, F., Weng, E., Lozano-Perez, T., and Kaelbling, L. Neural relational inference with fast modular meta-learning. In *Advances in Neural Information Processing Systems (NeurIPS) 32*, 2019.

[4] Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 136–145. JMLR, 2017.

[5] Antoniou, A. and Storkey, A. J. Learning to learn by self-critique. In *Advances in Neural Information Processing Systems*, pp. 9940–9950, 2019.

[6] Bartlett, P. L. and Mendelson, S. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[7] Bengio, S., Bengio, Y., and Cloutier, J. On the search for new learning rules for anns. *Neural Processing Letters*, 2(4):26–30, 1995.

[8] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402. Springer, 2013.

[9] Bottou, L. and Vapnik, V. Local learning algorithms. *Neural computation*, 4(6):888–900, 1992.

[10] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs. *github*, 2018. URL `http://github.com/google/jax`.

[11] Chapelle, O., Vapnik, V., and Weston, J. Transductive inference for estimating values of functions. In *Advances in Neural Information Processing Systems*, pp. 421–427, 2000.

[12] Chapelle, O., Scholkopf, B., and Zien, A. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

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

[14] Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pp. 1310–1320, 2019.

[15] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[16] Dhillon, G. S., Chaudhari, P., Ravichandran, A., and Soatto, S. A baseline for few-shot image classification. In *International Conference on Learning Representations*, 2020.

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

[18] Dumoulin, V., Shlens, J., and Kudlur, M. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629*, 2016.

[19] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[20] Flennerhag, S., Rusu, A. A., Pascanu, R., Yin, H., and Hadsell, R. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

[21] Garcia, V. and Bruna, J. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017.

[22] Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

[23] Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pp. 15353–15363, 2019.

[24] Hadsell, R., Chopra, S., and LeCun, Y. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pp. 1735–1742. IEEE, 2006.

[25] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[26] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[27] Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14 (8):1771–1800, 2002.

[28] Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[29] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pp. 125–136, 2019.

[30] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[31] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[32] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[33] Liu, Y., Lee, J., Park, M., Kim, S., Yang, E., Hwang, S. J., and Yang, Y. Learning to propagate labels: Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002*, 2018.

[34] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[35] Mityagin, B. The zero set of a real analytic function. *arXiv preprint arXiv:1512.07276*, 2015.

[36] Miyato, T., Maeda, S.-i., Koyama, M., and Ishii, S. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

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

[38] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

[39] Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[40] Rakelly, K., Zhou, A., Quillen, D., Finn, C., and Levine, S. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

[41] Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. Fast and flexible multi-task classification using conditional neural adaptive processes. In *Advances in Neural Information Processing Systems*, pp. 7957–7968, 2019.

[42] Reuther, A., Kepner, J., Byun, C., Samsi, S., Arcand, W., Bestor, D., Bergeron, B., Gadepally, V., Houle, M., Hubbell, M., et al. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–6. IEEE, 2018.

[43] Salman, H., Li, J., Razenshteyn, I., Zhang, P., Zhang, H., Bubeck, S., and Yang, G. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, pp. 11289–11300, 2019.

[44] Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[45] Suh, H. and Tedrake, R. The surprising effectiveness of linear models for visual foresight in object pile manipulation. *arXiv preprint arXiv:2002.09093*, 2020.

[46] Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A. A., and Hardt, M. Test-time training for out-of-distribution generalization. *arXiv preprint arXiv:1909.13231*, 2019.

[47] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[48] Thrun, S. and Pratt, L. *Learning to learn*. Springer Science & Business Media, 1998.

[49] Tschiatschek, S., Sahin, A., and Krause, A. Differentiable submodular maximization. *arXiv preprint arXiv:1803.01785*, 2018.

[50] Vapnik, V. N. The nature of statistical learning theory, 1995.

[51] Wang, D., Shelhamer, E., Olshausen, B., and Darrell, T. Dynamic scale inference by entropy minimization. *arXiv preprint arXiv:1908.03182*, 2019.

[52] Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, W. T., and Tenenbaum, J. B. Marrnet: 3d shape reconstruction via 2.5 d sketches. *arXiv preprint arXiv:1711.03129*, 2017.

[53] Zhai, R., Dan, C., He, D., Zhang, H., Gong, B., Ravikumar, P., Hsieh, C.-J., and Wang, L. Macer: Attack-free and scalable robust training via maximizing certified radius. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rJx1Na4Fwr`.

[54] Zhang, M., Marklund, H., Dhawan, N., Gupta, A., Levine, S., and Finn, C. Adaptive risk minimization: A meta-learning approach for tackling group distribution shift. *arXiv preprint arXiv:2007.02931*, 2020.

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

## 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] In section 6.2.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In section 6.1.
   (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? [Yes] Important assumptions are in the main text; all assumptions are detailed in the appendix, particularly in appendix A.
   (b) Did you include complete proofs of all theoretical results? [Yes] In appendix C.
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)? [No] We plan, however, to open-source our codebase once cleaned.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Distributed accross multiple sections in the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We report them for the planet experiments and the contrastive experiments. The adversarial experiments are extremely computationally expensive and we only ran them once.
   (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] In each relevant appendix.
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? [Yes] In the appendix.
   (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]

# A Theorem 2, Corollary 1 and interpretation of their conditions

**Assumption 1.** *(Common activation)* The activation function $\sigma(x)$ is real analytic, monotonically increasing, and the limits exist as: $\lim_{x\to-\infty}\sigma(x)=\sigma_->-\infty$ and $\lim_{x\to+\infty}\sigma(x)=\sigma_+\leq+\infty$.

**Theorem 2.** *For any $x\in\mathcal{X}$ that satisfies $\|g^{(i)}(x)\|_2^2-g^{(i)}(x)^\top g^{(j)}(x)>0$ (for all $i\neq j$), and for any fully-connected neural network with a single output unit, at least $n_g$ neurons per hidden layer, and activation functions that satisfy Assumption 1, the following holds: $\inf_{w,\gamma,\beta}\mathcal{L}^{\text{tailor}}(x,w,\gamma,\beta)=\inf_{\gamma,\beta}\mathcal{L}^{\text{tailor}}(x,\bar{w},\gamma,\beta)$ for any $\bar{w}\notin\mathcal{W}$ where Lebesgue measure of $\mathcal{W}\subset\mathbb{R}^d$ is zero.*

Assumption 1 is satisfied by using common activation functions such as sigmoid and hyperbolic tangent, as well as *softplus*, which is defined as $\sigma_\alpha(x)=\ln(1+\exp(\alpha x))/\alpha$ and satisfies Assumption 1 with any hyperparameter $\alpha\in\mathbb{R}_{>0}$. The softplus activation function can approximate the ReLU function to any desired accuracy: i.e., $\sigma_\alpha(x)\to\mathrm{relu}(x)$ as $\alpha\to\infty$, where relu represents the ReLU function.

In Theorem 2 and Corollary 1, the condition $\|g^{(i)}(x)\|_2^2-g^{(i)}(x)^\top g^{(j)}(x)>0$ (for all $i\neq j$) can be easily satisfied, for example, by choosing $g^{(1)},\ldots,g^{(n_g)}$ to produce normalized and distinguishable argumented inputs for each prediction point $x$ at prediction time. To see this, with normalization $\|g^{(i)}(x)\|_2^2=\|g^{(j)}(x)\|_2^2$, the condition is satisfied if $\|g^{(i)}(x)-g^{(j)}(x)\|_2^2>0$ for $i\neq j$ since $\frac{1}{2}\|g^{(i)}(x)-g^{(j)}(x)\|_2^2=\|g^{(i)}(x)\|_2^2-g^{(i)}(x)^\top g^{(j)}(x)$.

In general, the normalization is not necessary for the condition to hold; e.g., orthogonality on $g^{(i)}(x)$ and $g^{(j)}(x)$ along with $g^{(i)}(x)\neq0$ satisfies it without the normalization.

# B Understanding the expected meta-tailoring contrastive loss

To analyze meta-tailoring for contrastive learning, we focus on the binary classification loss of the form $\mathcal{L}^{\text{sup}}(f_\theta(x),y)=\ell_{\text{cont}}(f_\theta(x)_y-f_\theta(x)_{y'=\neg y})$ where $\ell_{\text{cont}}$ is convex and $\ell_{\text{cont}}(0)=1$. With this, the objective function $\theta\mapsto\mathcal{L}^{\text{sup}}(f_\theta(x),y)$ is still non-convex in general. For example, the standard hinge loss $\ell_{\text{cont}}(z)=\max\{0,1-z\}$ and the logistic loss $\ell_{\text{cont}}(z)=s\log_2(1+\exp(z))$ satisfy this condition.

We first define the meta-tailoring contrastive loss $\mathcal{L}_{\text{cont}}(x,\theta)$ in detail. In meta-tailoring contrastive learning, we choose the probability measure of positive example $x^+\sim\mu_{x^+}(x)$ and the probability measure of negative example $x^-,y^-\sim\mu_{x^-,y^-}(x)$, both of which are tailored for each input point $x$ at prediction time. These choices induce the marginal distributions for the negative examples $y^-\sim\mu_{y^-}(x)$ and $x^-\sim\mu_{x^-}(x)$, as well as the unknown probability of $y^-=y$ defined by $\rho_y(\mu_{y^-}(x))=\mathbb{E}_{y^-\sim\mu_{y^-}(x)}(\mathbb{1}\{y^-=y\})$. Define the lower and upper bound on the probability of $y^-=y$ as $\underline{\rho}(x)\leq\rho_y(\mu_{y^-}(x))\leq\bar{\rho}(x)\in[0,1)$.

Then, the first pre-meta-tailoring contrastive loss can be defined by

$$\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta)=\mathbb{E}_{\substack{x^+\sim\mu_{x^+}(x),\\x^-\sim\mu_{x^-}(x)}}[\ell_{\text{cont}}(h_\theta(x)^\top(h_\theta(x^+)-h_\theta(x^-)))],$$

where $h_\theta(x)\in\mathbb{R}^{m_H+1}$ represents the output of the last hidden layer, including a constant neuron corresponding the bias term of the last output layer (if there is no bias term, $h_\theta(x)\in\mathbb{R}^{m_H}$). For every $z\in\mathbb{R}^{2\times(m_H+1)}$, define $\psi_{x,y,y^-}(z)=\ell_{\text{cont}}((z_y-z_{y^-})h_\theta(x))$, where $z_y\in\mathbb{R}^{1\times m_H}$ is the $y$-th row vector of $z$. We define the second pre-meta-tailoring contrastive loss by

$$\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)=\max_y\mathbb{E}_{y^-\sim\mu_{y^-}(x)}[\psi_{x,y,y^-}(\theta^{(H+1)})-\psi_{x,1,2}([u_h^+,u_h^-]^\top)],$$

where $u_h^+=\mathbb{E}_{x^+\sim\mu_{x^+}(x)}[h_\theta(x^+)]$ and $u_h^-=\mathbb{E}_{x^-\sim\mu_{x^-}(x)}[h_\theta(x^-)]$. Here, we decompose $\theta$ as $\theta=(\theta^{(1:H)},\theta^{(H+1)})$, where $\theta^{(H+1)}=[W^{(H+1)},b^{(H+1)}]\in\mathbb{R}^{m_y\times(m_H+1)}$ represents the parameters at the last output layer, and $\theta^{(1:H)}$ represents all others.

Then, the meta-tailoring contrastive loss is defined by

$$\mathcal{L}_{\text{cont}}(x,\theta)=\frac{1}{1-\bar{\rho}(x)}\left(\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta)+\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)-\underline{\rho}(x)\right).$$

Theorem 3 states that for any $\theta^{(1:H)}$, the convex optimization of $\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta) + \mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)$ over $\theta^{(H+1)}$ can achieve the value of $\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta)$ without the value of $\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)$, allowing us to focus on the first term $\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta)$, for some choice of $\mu_{x^-,y^-}(x)$ and $\mu_{x^+}(x)$.

**Theorem 3.** *For any $\theta^{(1:H)}, \mu_{x^-,y^-}(x)$ and $\mu_{x^+}(x)$, the function $\theta^{(H+1)} \mapsto \mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta) + \mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)$ is convex. Moreover, there exists $\mu_{x^-,y^-}(x)$ and $\mu_{x^+}(x)$ such that, for any $\theta^{(1:H)}$ and any $\bar{\theta}^{(H+1)}$,*

$$\inf_{\theta^{(H+1)} \in \mathbb{R}^{m_y \times (m_H+1)}} \mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta) + \mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta) \le \mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta^{(1:H)},\bar{\theta}^{(H+1)}).$$

# C Proofs

In order to have concise proofs, we introduce additional notations while keeping track of dependent variables more explicitly. Since $h_\theta$ only depends on $\theta^{(1:H)}$, let us write $h_{\theta^{(1:H)}} = h_\theta$. Similarly, $\mathcal{L}_{\text{cont}}(x,\theta^{(1:H)}) = \mathcal{L}_{\text{cont}}(x,\theta)$. Let $\theta(x) = \theta_x$ and $\theta(x,S) = \theta_{x,S}$. Define $\mathcal{L} = \mathcal{L}^{\text{sup}}$.

## C.1 Proof of Theorem 2

*Proof of Theorem 2.* The output of fully-connected neural networks for an input $x$ with a parameter vector $\theta = (w,\gamma,\beta)$ can be represented by $f_\theta(x) = W^{(H+1)}h^{(H)}(x) + b^{(H+1)}$ where $W^{(H+1)} \in \mathbb{R}^{1\times m_H}$ and $b^{(H+1)} \in \mathbb{R}$ are the weight matrix and the bias term respectively at the last layer, and $h^{(H)}(x) \in \mathbb{R}^{m_H}$ represents the output of the last hidden layer. Here, $m_l$ represents the number of neurons at the $l$-th layer, and $h^{(l)}(x) = \gamma^{(l)}(\sigma(W^{(l)}h^{(l-1)}(x)+b^{(l)})) - \beta^{(l)} \in \mathbb{R}^{m_l}$ for $l = 1,\dots,H$, with trainable parameters $\gamma^{(l)},\beta^{(l)} \in \mathbb{R}^{m_l}$, where $h^{(0)}(x) = x$. Let $z^{(l)}(x) = \sigma(W^{(l)}h^{(l-1)}(x) + b^{(l)})$.

Then, by rearranging the definition of the output of the neural networks,

$$\begin{aligned} f_\theta(x) &= W^{(H+1)}h^{(H)}(x) + b^{(H+1)} \\ &= \left(\sum_{k=1}^{m_H} W_k^{(H+1)}\gamma_k^{(H)}z^{(H)}(x)_k + W_k^{(H+1)}\beta_k^{(H)}\right) + b^{(H+1)} \\ &= [W^{(H+1)} \circ z^{(H)}(x)^\top, W^{(H+1)}] \begin{bmatrix}\gamma^{(H)}\\ \beta^{(H)}\end{bmatrix} + b^{(H+1)}. \end{aligned}$$

Thus, we can write

$$\begin{bmatrix} f_\theta(g^{(1)}(x)) \\ \vdots \\ f_\theta(g^{(n_g)}(x)) \end{bmatrix} = M_w \begin{bmatrix}\gamma^{(H)}\\ \beta^{(H)}\end{bmatrix} + b^{(H+1)}\mathbf{1}_{n_g} \in \mathbb{R}^{n_g}, \tag{1}$$

where

$$M_w = \begin{bmatrix} W^{(H+1)} \circ z^{(H)}(g^{(1)}(x))^\top, W^{(H+1)} \\ \vdots \\ W^{(H+1)} \circ z^{(H)}(g^{(n_g)}(x))^\top, W^{(H+1)} \end{bmatrix} \in \mathbb{R}^{n_g \times 2m_H},$$

and $\mathbf{1}_{n_g} = [1,1,\dots,1]^\top \in \mathbb{R}^{n_g}$.

Using the above equality, we show an exitance of a $(\gamma,\beta)$ such that $\mathcal{L}^{\text{tailor}}(x,\bar{w},\gamma,\beta) = \inf_{w,\gamma,\beta}\mathcal{L}^{\text{tailor}}(x,\theta)$ for any $x \in \mathcal{X} \subseteq \mathbb{R}^{m_x}$ and any $\bar{w} \notin \mathcal{W}$ where Lebesgue measure of $\mathcal{W} \subset \mathbb{R}^d$ is zero. To do so, we first fix $\gamma_k^{(l)} = 1$ and $\beta_k^{(l)} = 0$ for $l = 1,\dots,H-1$, with which $h^{(l)}(x) = z^{(l)}(x)$ for $l = 1,\dots,H-1$.

Define $\varphi(w) = \det(M_w M_w^\top)$, which is analytic since $\sigma$ is analytic. Furthermore, we have that $\{w \in \mathbb{R}^d : M_w \text{ has rank less than } n_g\} = \{w \in \mathbb{R}^d : \varphi(w) = 0\}$, since the rank of $M_w$ and the rank of the Gram matrix are equal. Since $\varphi$ is analytic, if $\varphi$ is not identically zero ($\varphi \neq 0$), the Lebesgue

measure of its zero set $\{w \in \mathbb{R}^d : \varphi(w) = 0\}$ is zero [35]. Therefore, if $\varphi(w) \neq 0$ for some $w \in \mathbb{R}^d$, the Lebesgue measure of the set $\{w \in \mathbb{R}^d : M_w \text{ has rank less than } n_g\}$ is zero.

Accordingly, we now constructs a $w \in \mathbb{R}^d$ such that $\varphi(w) \neq 0$. Set $W^{(H+1)} = \mathbf{1}_{m_H}^\top$. Then,

$$M_w = [\bar{M}_w, \mathbf{1}_{n_g, m_H}] \in \mathbb{R}^{n_g \times m_H}.$$

where

$$\bar{M}_w = \begin{bmatrix} z^{(H)}(g^{(1)}(x))^\top \\ \vdots \\ z^{(H)}(g^{(n_g)}(x)(x))^\top \end{bmatrix} \in \mathbb{R}^{n_g \times m_H}$$

and $\mathbf{1}_{n_g, m_H} \in \mathbb{R}^{n_g \times m_H}$ with $(\mathbf{1}_{n_g, m_H})_{ij} = 1$ for all $i, j$. For $l = 1, \ldots, H$, define

$$G^{(l)} = \begin{bmatrix} z^{(l)}(g^{(1)}(x))^\top \\ \vdots \\ z^{(l)}(g^{(n_g)}(x))^\top \end{bmatrix} \in \mathbb{R}^{n_g \times m_l}.$$

Then, for $l = 1, \ldots, H$,

$$G^{(l)} = \sigma(G^{(l-1)}(W^{(l)})^\top + \mathbf{1}_{n_g}(b^{(l)})^\top),$$

where $\sigma$ is applied element-wise (by overloading of the notation $\sigma$), and

$$(\bar{M}_w)_{ik} = (G^{(H)})_{ik}.$$

From the assumption $g(x)$, there exists $c > 0$ such that $\|g^{(i)}(x)\|_2^2 - \langle g^{(i)}(x), g^{(j)}(x)\rangle > c$ for all $i \neq j$. From Assumption 1, there exists $c'$ such that $\sigma_+ - \sigma_- > c'$. Using these constants, set $W_i^{(1)} = \alpha^{(1)} g^{(i)}(x)^\top$ and $b_i^{(1)} = c\alpha^{(1)}/2 - \alpha^{(1)}\|g^{(i)}(x)\|_2^2$ for $i = 1, \ldots, n_g$, where $W_i^{(1)}$ represents the $i$-th row of $W^{(1)}$. Moreover, set $W^{(l)}_{1:n_g, 1:n_g} = \alpha^{(l)} I_{n_g}$ and $b_k^{(l)} = c'\alpha^{(l)}/2 - \alpha^{(l)}\sigma_+$ for all $k$ and $l = 2, \ldots, H$, where $W^{(l)}_{1:n_g, 1:n_g}$ is the fist $n_g \times n_g$ block matrix of $W^{(1)}$ and $I_{n_g}$ is the $n_g \times n_g$ identity matrix. Set all other weights and bias to be zero. Then, for any $i \in \{1, \ldots, n_g\}$,

$$(G^{(1)})_{ii} = \sigma(c\alpha^{(1)}/2),$$

and for any $k \in \{1, \ldots, n_g\}$ with $k \neq i$,

$$(G^{(1)})_{ik} = \sigma(\alpha^{(1)}(\langle g^{(i)}(x), g^{(k)}(x)\rangle - \|g^{(k)}(x)\|_2^2 + c/2)) \leq \sigma(-c\alpha^{(1)}/2).$$

Since $\sigma(c\alpha^{(1)}/2) \to \sigma_+$ and $\sigma(-c\alpha^{(1)}/2) \to \sigma_-$ as $\alpha^{(1)} \to \infty$, with $\alpha^{(1)}$ sufficiently large, we have that $\sigma(c\alpha^{(1)}/2) - \sigma_+ + c'/2 \geq c_1^{(2)}$ and $\sigma(-c\alpha^{(1)}/2) - \sigma_+ + c'/2 \leq -c_2^{(2)}$ for some $c_1^{(2)}, c_2^{(2)} > 0$. Note that $c_1^{(2)}$ and $c_2^{(2)}$ depends only on $\alpha^{(1)}$ and does not depend on any of $\alpha^{(2)}, \ldots, \alpha^{(H)}$. Therefore, with $\alpha^{(1)}$ sufficiently large,

$$(G^{(2)})_{ii} = \sigma(\alpha^{(2)}(\sigma(c\alpha^{(1)}/2) - \sigma_+ + c'/2)) \geq \sigma(\alpha^{(2)} c_1^{(2)}),$$

and

$$(G^{(2)})_{ik} \leq \sigma(\alpha^{(2)}(\sigma(-c\alpha^{(1)}/2) - \sigma_+ + c'/2)) \leq \sigma(-\alpha^{(2)} c_2^{(2)}).$$

Repeating this process with Assumption 1, we have that with $\alpha^{(1)}, \ldots, \alpha^{(H-1)}$ sufficiently large,

$$(G^{(H)})_{ii} \geq \sigma(\alpha^{(H)} c_1^{(H)}),$$

and

$$(G^{(H)})_{ik} \leq \sigma(-\alpha^{(H)} c_2^{(H)}).$$

Here, $(G^{(H)})_{ii} \to \sigma_+$ and $(G^{(H)})_{ik} \to \sigma_-$ as $\alpha^{(H)} \to \infty$. Therefore, with $\alpha^{(1)}, \ldots, \alpha^{(H)}$ sufficiently large, for any $i \in \{1, \ldots, n_g\}$,

$$\left|(\bar{M}_w)_{ii} - \sigma_-\right| > \sum_{k \neq i} \left|(\bar{M}_w)_{ik} - \sigma_-\right|. \tag{2}$$

The inequality (2) means that the matrix $\bar{M}'_w = [(\bar{M}_w)_{ij} - \sigma_-]_{1 \leq i,j \leq n_g} \in \mathbb{R}^{n_g \times n_g}$ is strictly diagonally dominant and hence is nonsingular with rank $n_g$. This implies that the matrix $[\bar{M}'_w, \mathbf{1}_{n_g}] \in$

$\mathbb{R}^{n_g \times (n_g+1)}$ has rank $n_g$. This then implies that the matrix $\tilde{M}_w = [[(\bar{M}_w)_{ij}]_{1 \le i,j \le n_g}, \mathbf{1}_{n_g}] \in \mathbb{R}^{n_g \times (n_g+1)}$ has rank $n_g$, since the elementary matrix operations preserve the matrix rank. Since the set of all columns of $M_w$ contains all columns of $\tilde{M}_w$, this implies that $M_w$ has rank $n_g$ and $\varphi(w) \neq 0$ for this constructed particular $w$.

Therefore, the Lebesgue measure of the set $\mathcal{W} = \{w \in \mathbb{R}^d : \varphi(w) = 0\}$ is zero. If $w \notin \mathcal{W}$, $\{(f_{\bar{w},\bar{\gamma},\bar{\beta}}(g^{(1)}(x)), \dots, f_{\bar{w},\bar{\gamma},\bar{\beta}}(g^{(n_g)}(x)) \in \mathbb{R}^{n_g} : \bar{\gamma}^{(l)}, \bar{\beta}^{(l)} \in \mathbb{R}^{ml}\} = \mathbb{R}^{n_g}$, since $M_w$ has rank $n_g$ in (1) for some $\bar{\gamma}^{(l)}, \bar{\beta}^{(l)}$ for $l = 1, \dots, H-1$ as shown above. Thus, for any $\bar{w} \notin \mathcal{W}$ and for any $(w, \gamma, \beta)$, there exists $(\bar{\gamma}, \bar{\beta})$ such that

$$(f_{w,\gamma,\beta}(g^{(1)}(x)), \dots, f_{w,\gamma,\beta}(g^{(n_g)}(x)) = (f_{\bar{w},\bar{\gamma},\bar{\beta}}(g^{(1)}(x)), \dots, f_{\bar{w},\bar{\gamma},\bar{\beta}}(g^{(n_g)}(x))$$

which implies the desired statement.

□

### C.2 Proof of Corollary 1

*Proof of Corollary 1.* Since non-degenerate Gaussian measure with any mean and variance is absolutely continuous with respect to Lebesgue measure, Theorem 2 implies the statement of this corollary.

□

### C.3 Proof of Theorem 1

The following lemma provides an upper bound on the expected loss via expected meta-tailoring contrastive loss.

**Lemma 4.** *For every $\theta$,*

$$\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] \le \mathbb{E}_x \left[ \frac{1}{1 - \bar{\rho}(x)} \left( \mathcal{L}_{\text{cont}}^{x^+, x^-}(x, \theta^{(1:H)}) + \mathcal{L}_{\text{cont}}^{x^+, x^-, y^-}(x, \theta) - \underline{\rho}(x) \right) \right]$$

*Proof of Lemma 4.* Using the notation $\rho = \rho_y(\mu_{y^-}(x))$,

$$\begin{aligned}
&\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] \\
&= \mathbb{E}_{x,y} \left[ \frac{1}{1-\rho} \left( (1-\rho)\mathcal{L}(f_\theta(x), y) \pm \rho \right) \right] \\
&= \mathbb{E}_{x,y} \left[ \frac{1}{1-\rho} \left( (1-\rho)\ell_{\text{cont}}(f_\theta(x)_y - f_\theta(x)_{y^- \neq y}) + \rho \ell_{\text{cont}}(f_\theta(x)_y - f_\theta(x)_{y^- = y}) - \rho \right) \right] \\
&= \mathbb{E}_{x,y} \left[ \frac{1}{1-\rho} \left( \mathbb{E}_{y^- \sim \mu_{y^-}(x)}[\ell_{\text{cont}}(f_\theta(x)_y - f_\theta(x)_{y^-})] - \rho \right) \right] \\
&= \mathbb{E}_{x,y} \left[ \frac{1}{1-\rho} \left( \mathbb{E}_{y^- \sim \mu_{y^-}(x)}[\psi_{x,y,y^-}(\theta^{(H+1)})] - \rho \right) \right] \\
&\le \mathbb{E}_{x,y} \left[ \frac{1}{1-\rho} \left( \psi_{x,1,2}([u_h^+, u_h^-]^\top) + \mathcal{L}_{\text{cont}}^{x^+, x^-, y^-}(x, \theta) - \rho \right) \right] \\
&\le \mathbb{E}_{x,y} \left[ \frac{1}{1-\rho} \left( \mathcal{L}_{\text{cont}}^{x^+, x^-}(x, \theta^{(1:H)}) + \mathcal{L}_{\text{cont}}^{x^+, x^-, y^-}(x, \theta) - \rho \right) \right]
\end{aligned}$$

where the third line follows from the definition of $\mathcal{L}(f_\theta(x), y)$ and $\ell_{\text{cont}}(f_\theta(x)_y - f_\theta(x)_{y'=y}) = \ell_{\text{cont}}(0) = 1$, the forth line follows from the definition of $\rho$ and the expectation $\mathbb{E}_{y^- \sim \mu_{y^-}(x)}$, the fifth line follows from $f_\theta(x)_y = \theta_y^{(H+1)} h_{\theta^{(1:H)}}(x)$ and $f_\theta(x)_{y^-} = \theta_{y^-}^{(H+1)} h_{\theta^{(1:H)}}(x)$, the sixth line follows from the definition of $\mathcal{L}_{\text{cont}}^{x^+, x^-, y^-}$. The last line follows from the convexity of $\ell_{\text{cont}}$ and Jensen's inequality: i.e.,

$$\begin{aligned}
&\psi_{x,1,2}([u_h^+, u_h^-]^\top) \\
&= \ell_{\text{cont}}(\mathbb{E}_{x^+ \sim \mu_{x^+}(x)} \mathbb{E}_{x^- \sim \mu_{x^-}(x)}[(h_{\theta^{(1:H)}}(x^+) - h_{\theta^{(1:H)}}(x^-))^\top h_{\theta^{(1:H)}}(x)]) \\
&\le \mathbb{E}_{x^+ \sim \mu_{x^+}(x)} \mathbb{E}_{x^- \sim \mu_{x^-}(x)} \ell_{\text{cont}}((h_{\theta^{(1:H)}}(x^+) - h_{\theta^{(1:H)}}(x^-))^\top h_{\theta^{(1:H)}}(x)).
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x),y)] \\
&\le \mathbb{E}_{x,y}\left[\frac{1}{1-\rho_y(\mu_{y^-}(x))}\left(\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta^{(1:H)})+\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)-\rho_y(\mu_{y^-}(x))\right)\right] \\
&\le \mathbb{E}_{x}\left[\frac{1}{1-\bar{\rho}(x)}\left(\mathcal{L}_{\text{cont}}^{x^+,x^-}(x,\theta^{(1:H)})+\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x,\theta)-\underline{\rho}(x)\right)\right]
\end{aligned}$$

where we used $\underline{\rho}(x) \le \rho_y(\mu_{y^-}(x)) \le \bar{\rho}(x) \in [0,1)$. □

**Lemma 5.** *Let $S \mapsto f_{\theta(x,S)}(x)$ be an uniformly $\zeta$-stable tailoring algorithm. Then, for any $\delta > 0$, with probability at least $1-\delta$ over an i.i.d. draw of $n$ i.i.d. samples $S = ((x_i, y_i))_{i=1}^n$, the following holds:*

$$\mathbb{E}_{x,y}[\mathcal{L}(f_{\theta(x,S)}(x),y)] \le \frac{1}{n}\sum_{i=1}^n \mathcal{L}(f_{\theta(x_i,S)}(x_i),y_i) + \frac{\zeta}{n} + (2\zeta + c)\sqrt{\frac{\ln(1/\delta)}{2n}}.$$

*Proof of Lemma 5.* Define $\varphi_1(S) = \mathbb{E}_{x,y}[\mathcal{L}(f_{\theta(x,S)}(x),y)]$ and $\varphi_2(S) = \frac{1}{n}\sum_{i=1}^n \mathcal{L}(f_{\theta(x_i,S)}(x_i),y_i)$, and $\varphi(S) = \varphi_1(S) - \varphi_2(S)$. To apply McDiarmid's inequality to $\varphi(S)$, we compute an upper bound on $|\varphi(S) - \varphi(S')|$ where $S$ and $S'$ be two training datasets differing by exactly one point of an arbitrary index $i_0$; i.e., $S_i = S'_i$ for all $i \ne i_0$ and $S_{i_0} \ne S'_{i_0}$, where $S' = ((x'_i, y'_i))_{i=1}^n$. Let $\tilde{\zeta} = \frac{\zeta}{n}$ Then,

$$|\varphi(S) - \varphi(S')| \le |\varphi_1(S) - \varphi_1(S')| + |\varphi_2(S) - \varphi_2(S')|.$$

For the first term, using the $\zeta$-stability,

$$\begin{aligned}
|\varphi_1(S) - \varphi_1(S')| &\le \mathbb{E}_{x,y}[|\mathcal{L}(f_{\theta(x,S)}(x),y) - \mathcal{L}(f_{\theta(x,S')}(x),y)|] \\
&\le \tilde{\zeta}.
\end{aligned}$$

For the second term, using $\zeta$-stability and the upper bound $c$ on per-sample loss,

$$\begin{aligned}
|\varphi_2(S) - \varphi_2(S')| &\le \frac{1}{n}\sum_{i\ne i_0} |\mathcal{L}(f_{\theta(x_i,S)}(x_i),y_i) - \mathcal{L}(f_{\theta(x_i,S')}(x_i),y_i)| + \frac{c}{n} \\
&\le \frac{(n-1)\tilde{\zeta}}{n} + \frac{c}{n} \le \tilde{\zeta} + \frac{c}{n}.
\end{aligned}$$

Therefore, $|\varphi(S) - \varphi(S')| \le 2\tilde{\zeta} + \frac{c}{n}$. By McDiarmid's inequality, for any $\delta > 0$, with probability at least $1-\delta$,

$$\varphi(S) \le \mathbb{E}_S[\varphi(S)] + (2\zeta + c)\sqrt{\frac{\ln(1/\delta)}{2n}}.$$

The reset of the proof bounds the first term $\mathbb{E}_S[\varphi(S)]$. By the linearity of expectation,

$$\mathbb{E}_S[\varphi(S)] = \mathbb{E}_S[\varphi_1(S)] - \mathbb{E}_S[\varphi_1(S)].$$

For the first term,

$$\mathbb{E}_S[\varphi_1(S)] = \mathbb{E}_{S,x,y}[\mathcal{L}(f_{\theta(x,S)}(x),y)].$$

For the second term, using the linearity of expectation,

$$\begin{aligned}
\mathbb{E}_S[\varphi_2(S)] &= \mathbb{E}_S\left[\frac{1}{n}\sum_{i=1}^n \mathcal{L}(f_{\theta(x_i,S)}(x_i),y_i)\right] \\
&= \frac{1}{n}\sum_{i=1}^n \mathbb{E}_S[\mathcal{L}(f_{\theta(x_i,S)}(x_i),y_i)] \\
&= \frac{1}{n}\sum_{i=1}^n \mathbb{E}_{S,x,y}[\mathcal{L}(f_{\theta(x,S^i_{x,y})}(x),y)],
\end{aligned}$$

where $S^i$ is a sample of $n$ points such that $(S^i_{x,y})_j = S_j$ for $j \neq i$ and $(S^i_{x,y})_i = (x, y)$. By combining these, using the linearity of expectation and $\zeta$-stability,

$$\begin{aligned}\mathbb{E}_S[\varphi(S)] &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{S,x,y}[\mathcal{L}(f_{\theta(x,S)}(x), y) - \mathcal{L}(f_{\theta(x,S^i_{x,y})}(x), y)] \\ &\leq \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{S,x,y}[|\mathcal{L}(f_{\theta(x,S)}(x), y) - \mathcal{L}(f_{\theta(x,S^i_{x,y})}(x), y)|] \\ &\leq \frac{1}{n}\sum_{i=1}^{n} \tilde{\zeta} = \tilde{\zeta}.\end{aligned}$$

Therefore, $\mathbb{E}_S[\varphi(S)] \leq \tilde{\zeta}$.

□

*Proof of Theorem 1.* For any $\theta$ and $\kappa \in [0, 1]$,

$$\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] = \kappa \mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] + (1 - \kappa)\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)].$$

Applying Lemma 4 for the first term and Lemma 5 yields the desired statement.

□

### C.4 Statement and proof of Theorem 6

**Theorem 6.** *Let $\mathcal{F}$ be an arbitrary set of maps $x \mapsto f_{\theta_x}(x)$. Then, for any $\delta > 0$, with probability at least $1-\delta$ over an i.i.d. draw of $n$ i.i.d. samples $((x_i, y_i))_{i=1}^n$, the following holds: for all maps $(x \mapsto f_{\theta_x}(x)) \in \mathcal{F}$ and any $\kappa \in [0, 1]$, we have that $\mathbb{E}_{x,y}[\mathcal{L}^{\text{sup}}(f_{\theta_x}(x), y)] \leq \kappa \mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_x)] + (1-\kappa)\mathcal{J}'$, where $\mathcal{J}' = \frac{1}{n}\sum_{i=1}^n \mathcal{L}^{\text{sup}}(f_{\theta_{x_i}}(x_i), y_i) + 2\mathcal{R}_n(\mathcal{L}^{\text{sup}} \circ \mathcal{F}) + c\sqrt{(\ln(1/\delta))/(2n)}$.*

The following lemma is used along with Lemma 4 to prove the statement of this theorem.

**Lemma 7.** *Let $\mathcal{F}$ be an arbitrary set of maps $x \mapsto f_{\theta_x}(x)$. For any $\delta > 0$, with probability at least $1 - \delta$ over an i.i.d. draw of $n$ i.i.d. samples $((x_i, y_i))_{i=1}^n$, the following holds: for all maps $(x \mapsto f_{\theta_x}(x)) \in \mathcal{F}$,*

$$\mathbb{E}_{x,y}[\mathcal{L}(f_{\theta_x}(x), y)] \leq \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i) + 2\mathcal{R}_n(\mathcal{L} \circ \mathcal{F}) + c\sqrt{\frac{\ln(1/\delta)}{2n}}.$$

*Proof of Lemma 7.* Let $S = ((x_i, y_i))_{i=1}^n$ and $S' = ((x'_i, y'_i))_{i=1}^n$. Define

$$\varphi(S) = \sup_{(x \mapsto f_{\theta_x}(x)) \in \mathcal{F}} \mathbb{E}_{x,y}[\mathcal{L}(f_{\theta_x}(x), y)] - \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i).$$

To apply McDiarmid's inequality to $\varphi(S)$, we compute an upper bound on $|\varphi(S) - \varphi(S')|$ where $S$ and $S'$ be two training datasets differing by exactly one point of an arbitrary index $i_0$; i.e., $S_i = S'_i$ for all $i \neq i_0$ and $S_{i_0} \neq S'_{i_0}$. Then,

$$\varphi(S') - \varphi(S) \leq \sup_{(x \mapsto f_{\theta_x}(x)) \in \mathcal{F}} \frac{\mathcal{L}(f_{\theta(x_{i_0})}(x_{i_0}), y_{i_0}) - \mathcal{L}(f_{\theta(x'_{i_0})}(x'_{i_0}), y'_{i_0})}{n} \leq \frac{c}{n}.$$

Similarly, $\varphi(S) - \varphi(S') \leq \frac{c}{n}$. Thus, by McDiarmid's inequality, for any $\delta > 0$, with probability at least $1 - \delta$,

$$\varphi(S) \leq \mathbb{E}_S[\varphi(S)] + c\sqrt{\frac{\ln(1/\delta)}{2n}}.$$

Moreover, with $f(x) = f_{\theta_x}(x)$,

$$\begin{aligned}
\mathbb{E}_S[\varphi(S)] &= \mathbb{E}_S\left[\sup_{f\in\mathcal{F}} \mathbb{E}_{S'}\left[\frac{1}{n}\sum_{i=1}^n \mathcal{L}(f_{\theta(x_i')}(x_i'), y_i')\right] - \frac{1}{n}\sum_{i=1}^n \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i)\right] \\
&\le \mathbb{E}_{S,S'}\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n (\mathcal{L}(f_{\theta(x_i')}(x_i'), y_i') - \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i)\right] \\
&\le \mathbb{E}_{\xi,S,S'}\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n \xi_i(\mathcal{L}(f_{\theta(x_i')}(x_i'), y_i') - \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i))\right] \\
&\le 2\mathbb{E}_{\xi,S}\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n \xi_i\mathcal{L}(f_{\theta_{x_i}}(x_i), y_i))\right]
\end{aligned}$$

where the fist line follows the definitions of each term, the second line uses the Jensen's inequality and the convexity of the supremum, and the third line follows that for each $\xi_i \in \{-1, +1\}$, the distribution of each term $\xi_i(\mathcal{L}(f_{\theta(x_i')}(x_i'), y_i') - \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i))$ is the distribution of $(\mathcal{L}(f_{\theta(x_i')}(x_i'), y_i') - \mathcal{L}(f_{\theta_{x_i}}(x_i), y_i))$ since $\bar{S}$ and $\bar{S}'$ are drawn iid with the same distribution. The forth line uses the subadditivity of supremum. □

*Proof of Theorem 6.* For any $\theta$ and $\kappa \in [0, 1]$,

$$\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] = \kappa\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] + (1-\kappa)\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)].$$

Applying Lemma 4 for the first term and Lemma 7 yields the desired statement.

□

### C.5 Proof of Theorem 3

*Proof of Theorem 3.* Let $\theta^{(1:H)}$ be fixed. We first prove the first statement for the convexity. The function $\theta^{(H+1)} \mapsto \psi_{x,y,y^-}(\theta^{(H+1)})$ is convex, since it is a composition of a convex function $\ell_{\text{cont}}$ and a affine function $z \mapsto (z_y - z_{y^-})h_{\theta^{(1:H)}}(x))$. The function $\theta^{(H+1)} \mapsto \mathbb{E}_{y^-\sim\mu_{y^-}(x)}[\psi_{x,y,y^-}(\theta^{(H+1)}) - \psi_{x,1,2}([u_h^+, u_h^-]^\top)]$ is convex since the expectation and affine translation preserves the convexity. Finally, $\theta^{(H+1)} \mapsto \mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x, \theta^{(1:H)}, \theta^{(H+1)})$ is convex since it is the piecewise maximum of the convex functions

$$\theta^{(H+1)} \mapsto \mathbb{E}_{y^-\sim\mu_{y^-}(x)}[\psi_{x,y,y^-}(\theta^{(H+1)}) - \psi_{x,1,2}([u_h^+, u_h^-]^\top)]$$

for each $y$.

We now prove the second statement of the theorem for the inequality. Let us write $\mu_{x^+} = \mu_{x|y}$ and $\mu_{x^-} = \mu_{x|y^-}$. Let $U = [u_1, u_2]^\top \in \mathbb{R}^{m_y \times (m_H+1)}$ where $u_y = \mathbb{E}_{x\sim\mu_{x|y}}[h_{\theta^{(1:H)}}(x)]$ for $y \in \{1, 2\}$. Then,

$$u_h^+ = \mathbb{E}_{x^+\sim\mu_{x^+}(x)}[h_{\theta^{(1:H)}}(x^+)] = \mathbb{E}_{x^+\sim\mu_{x|y}}[h_{\theta^{(1:H)}}(x^+)] = u_y,$$

and

$$u_h^- = \mathbb{E}_{x^-\sim\mu_{x^-}(x)}[h_{\theta^{(1:H)}}(x^-)] = \mathbb{E}_{x^-\sim\mu_{x|y^-}}[h_{\theta^{(1:H)}}(x^-)] = u_{y^-}.$$

Therefore,

$$\psi_{x,1,2}([u_h^+, u_h^-]^\top) = \psi_{x,y,y^-}(U),$$

with which

$$\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x, \theta) = \max_y \mathbb{E}_{y^-\sim\mu_{y^-}(x)}[\psi_{x,y,y^-}(\theta^{(H+1)}) - \psi_{x,y,y^-}(U)].$$

Since $U$ and $\theta^{(1:H)}$ do not contain $\theta^{(H+1)}$, for any $U, \bar{\theta}^{(1:H)}$, there exists $\theta^{(H+1)} = U$ for which $\psi_{x,y,y^-}(\theta^{(H+1)}) - \psi_{x,y,y^-}(U) = 0$ and hence $\mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x, \theta) = 0$. Therefore,

$$\begin{aligned}
&\inf_{\theta^{(H+1)}\in\mathbb{R}^{m_y\times(m_H+1)}} \mathcal{L}_{\text{cont}}^{x^+,x^-}(x, \theta^{(1:H)}) + \mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x, \theta^{(1:H)}, \theta^{(H+1)}) \\
&= \mathcal{L}_{\text{cont}}^{x^+,x^-}(x, \theta^{(1:H)}) + \inf_{\theta^{(H+1)}\in\mathbb{R}^{m_y\times(m_H+1)}} \mathcal{L}_{\text{cont}}^{x^+,x^-,y^-}(x, \theta^{(1:H)}, \theta^{(H+1)}) \\
&\le \mathcal{L}_{\text{cont}}^{x^+,x^-}(x, \theta^{(1:H)}).
\end{aligned}$$

□

### C.6 Proof of Remark 1

*Proof of Remark 1.* For any $\theta$,

$$\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] = \inf_{\kappa \in [0,1]} \kappa \mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] + (1-\kappa)\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)].$$

Applying Lemma 5 for Theorem 1 (and Lemma 7 for Theorem 6) to the second term and the assumption $\mathbb{E}_{x,y}[\mathcal{L}(f_\theta(x), y)] \leq \mathbb{E}_x[\mathcal{L}_{\text{un}}(f_\theta(x))]$ to the first term yields the desired statement.

□

## D Details and description of CNGRAD

In this section we describe CNGRAD in greater detail: its implementation, different variants and run-time costs. Note that, although this section is written from the perspective of meta-tailoring, CNGRAD is also applicable to meta-learning, we provide pseudo-code in algorithm 4. The main idea behind CNGRAD is to optimize only conditional normalization (CN) parameters $\gamma^{(l)}, \beta^{(l)}$ in the inner loop and optimize all the other weights $w$ in the outer loop. To simplify notation for implementation, in this subsection only, we overload notations to make them work over a mini-batch as follows. Let $b$ be a (mini-)batch size. Given $X \in \mathbb{R}^{b \times m_0}$, $\gamma \in \mathbb{R}^{b \times \sum_l m_l}$ and $\beta \in \mathbb{R}^{b \times \sum_l m_l}$, let $(f_{w,\gamma,\beta}(X))_i = f_{w,\gamma_i,\beta_i}(X_i)$ where $X_i$, $\gamma_i$, and $\beta_i$ are the transposes of the $i$-th row vectors of $X$, $\gamma$ and $\beta$, respectively. Similarly, $\mathcal{L}^{\text{sup}}$ and $\mathcal{L}^{\text{tailor}}$ are used over a mini-batch. We also refer to $\theta = (w, \gamma, \beta)$.

**Initialization of $\gamma, \beta$** In the inner loop we always initialize $\gamma = \mathbf{1}_{b,\sum_l m_l}, \beta = \mathbf{0}_{b,\sum_l m_l}$. More complex methods where the initialization of these parameters is meta-trained are also possible. However, we note two things:

1. By initializing to the identity function, we can pick an architecture trained with regular inductive learning, add CN layers without changing predictions and perform tailoring. In this manner, the prediction algorithm is the same regardless of whether we trained with meta-tailoring or without the CN parameters.
2. We can add a previous normalization layer with weights $\gamma'^{(l)}, \beta'^{(l)}$ that are trained in the outer loop, having a similar effect than meta-learning an initialization. However, we do not do it in our experiments.

**First and second order versions of CNGRAD:** $w$ affect $\mathcal{L}^{\text{sup}}$ in two ways: first, they directly affect the evaluation $f_{w,\gamma_s,\beta_s}(X)$ by being weights of the neural network; second, they affect $\nabla_\beta \mathcal{L}^{\text{tailor}}, \nabla_\gamma \mathcal{L}^{\text{tailor}}$ which affects $\gamma_s, \beta_s$ which in turn affect $\mathcal{L}^{\text{sup}}$. Similar to MAML [19], we can implement two versions: in the first order version we only take into account the first effect, while in the second order version we take into account both effects. The first order version has three advantages:

1. It is very easy to code: the optimization of the inner parameters and the outer parameters are detached and we simply need to back-propagate $\mathcal{L}^{\text{tailor}}$ with respect to $\beta, \gamma$ and $\mathcal{L}^{\text{sup}}$ with respect to $w$. This version is easier to implement than most meta-learning algorithms, since the parameters in the inner and outer loop are different.
2. It is faster: because we do not back-propagate through the optimization, the overall computation graph is smaller.
3. It is more stable to train: second-order gradients can be a bit unstable to train; this required us to lower the inner tailoring learning rate in experiments of section 5.1 for the second-order version.

The second-order version has one big advantage: it optimizes the true objective, taking into account how $\mathcal{L}^{\text{tailor}}$ will affect the update of the network. This is critical to linking the unsupervised loss to best serve the supervised loss by performing informative updates to the CN parameters.

**WarpGrad-inspired stopping of gradients and subsequent reduction in memory cost:** WarpGrad [20] was an inspiration to CNGRAD suggesting to interleave layers that are adapted in the inner loop with layers only adapted in the outer loop. In contrast to WarpGrad, we can evaluate inputs (in meta-tailoring) or tasks (in meta-learning) in parallel, which speeds up training and inference. This also simplifies the code because we do not have to manually perform batches of tasks by iterating through them.

WarpGrad also proposes to stop the gradients between inner steps; we include this idea as an optional operation in CNGRAD, as shown in line 12 of 3. The advantage of adding it is that it decreases the memory cost when performing multiple inner steps, as we now only have to keep in memory the computation graph of the last step instead of all the steps, key when the networks are very deep like in the experiments of section 5.4. Another advantage is that it makes training more stable, reducing

**Algorithm 3** CNGRAD for meta-tailoring

**Subroutine** *Training(*$f$*,* $\mathcal{L}^{\text{sup}}$*,* $\lambda_{sup}$*,* $\mathcal{L}^{\text{tailor}}$*,* $\lambda_{tailor}$*,* $steps$*,*$((x_i, y_i))_{i=1}^n$*)*

randomly initialize $w$ `// All parameters except` $\gamma, \beta$ `; trained in outer loop`

**while** *not done* **do**

**for** $0 \leq i \leq n/b$ **do** `//` $b$ `batch size`

$X, Y = x_{ib:i(b+1)}, y_{ib:i(b+1)}$ $\gamma_0 = \mathbf{1}_{b,\sum_l m_l}$ $\beta_0 = \mathbf{0}_{b,\sum_l m_l}$

**for** $1 \leq s \leq steps$ **do**

$\gamma_s = \gamma_{s-1} - \lambda_{tailor} \nabla_\gamma \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$ `// Inner step w.r.t.` $\gamma$

$\beta_s = \beta_{s-1} - \lambda_{tailor} \nabla_\beta \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$ `// Inner step w.r.t.` $\beta$

$\gamma_s, \beta_s = \gamma_s.detach(), \beta_s.detach()$ `// Optional operation, only in` $1^{st}$ `order CNGrad: WarpGrad detach to avoid back-proping through multiple steps; reducing memory, and increasing stability, but adding bias.`

$w = w - \lambda_{sup} \nabla_w \mathcal{L}^{\text{sup}}\left(f_{w,\gamma_s,\beta_s}(X), Y\right)$ `// Outer step`

**return** $w$

**Subroutine** *Prediction(*$f$*,* $w$*,* $\mathcal{L}^{\text{tailor}}$*,* $\lambda$*,* $steps$*,* $X$*)* `// For meta-tailoring & tailoring`

`// X contains multiple inputs, with independent tailoring processes`

$b = X.shape[0]$ `// number of inputs`

$\gamma_0 = \mathbf{1}_{b,\sum_l m_l}$ $\beta_0 = \mathbf{0}_{b,\sum_l m_l}$

**for** $1 \leq s \leq steps$ **do**

$\gamma_s = \gamma_{s-1} - \lambda \nabla_\gamma \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$

$\beta_s = \beta_{s-1} - \lambda \nabla_\beta \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$

**return** $f_{w,\gamma_{steps},\beta_{steps}}(X)$

variance, as back-propagating through the optimization is often very noisy for many steps. At the same time it adds bias, because it makes the greedy assumption that locally minimizing the decrease in outer loss at every step will lead to low overall loss after multiple steps.

**Computational cost:** in CNGRAD we perform multiple forward and backward passes, compared to a single forward pass in the usual setting. In particular, if we perform $s$ tailoring steps, we execute $(s+1)$ forward steps and $s$ backward steps, which usually take the same amount of time as the forward steps. Therefore, in its naive implementation, this method takes about $2s+1$ times more than executing the regular network without tailoring.

However, it is well-known that we can often only adapt the higher layers of a network, while keeping the lower layers constant. Moreover, our proof about the capacity of CNGRAD to optimize a broad range of inner losses only required us to adapt the very last CN layer $\gamma^{(H)}, \beta^{(H)}$. This implies we can put the CN layers only on the top layer(s). In the case of only having one CN layer at the last network layer, we only require one initial full forward pass (as we do without tailoring). Then, we have $s$ backward-forward steps that affect only the last layer, thus costing $\frac{1}{H}$ in case of layers of equivalent cost. This leads to a factor of $1 + \frac{2s}{H}$ in cost, which for $s$ small and $H$ large (typical for deep networks), is a very small overcost. Moreover, for tailoring and meta-tailoring, we are likely to get the same performance with smaller networks, which may compensate the increase in cost.

**Meta-learning version:** CNGRAD can also be used in meta-learning, with the advantage of being provably expressive, very efficient in terms of parameters and compute, and being able to parallelize across tasks. We show the pseudo-code for few-shot supervised learning in algorithm 4. There are two changes to handle the meta-learning setting: first, in the inner loop, instead of the unsupervised tailoring loss we optimize a supervised loss on the training (support) set. Second, we want to share the same inner parameters $\gamma, \beta$ for different samples of the same task. To do so we add the operation "repeat_interlave" (PyTorch notation), which makes $k$ contiguous copies of each parameter $\gamma, \beta$, before feeding them to the network evaluation. In doing so, gradients coming from different samples of the same task get pooled together. At test time we do the same for the $k'$ queries ($k'$ can be different than $k$). Note that, in practice, this pooling is also used in meta-tailoring when we have more than one data augmentation within $\mathcal{L}^{\text{tailor}}$.

**Algorithm 4** CNGRAD for meta-learning

**Subroutine** *Meta-training(*$f$*,* $\mathcal{L}^{\text{sup}}$*,* $\lambda_{inner}$*,* $\lambda_{outer}$*, steps,*$\mathcal{T}$*)*

randomly initialize $w$ `// All parameters except` $\gamma, \beta$ `; trained in outer loop`

**while** *not done* **do**

**for** $0 \leq i \leq n/b$ **do** `//` $b$ `batch size`

$X_{train}, Y_{train} = [\,], [\,]$ $X_{test}, Y_{test} = [\,], [\,]$ **for** $ib \leq j \leq i(b+1)$ **do**

$(inp, out) \sim_k \mathcal{T}_j$ `// Take` $k$ `samples from each task for training`

$X.append\,(inp)\,; Y.append\,(out)$ $(query, target) \sim'_k \mathcal{T}_j$ `// Take` $k'$ `samples from each task for testing`

$X.append\,(query)\,; Y.append\,(target)$

`// We can now batch evaluations of multiple tasks`

$X_{train}, Y_{train} = concat\,(X_{train}, dim = 0)\,, concat\,(Y_{train}, dim = 0)$

$X_{test}, Y_{test} = concat\,(X_{test}, dim = 0)\,, concat\,(Y_{test}, dim = 0)$ $\gamma_0 = \mathbf{1}_{b, \sum_l m_l}$

$\beta_0 = \mathbf{0}_{b, \sum_l m_l}$ **for** $1 \leq s \leq steps$ **do**

`// We now repeat the CN parameters` $k$ `times so that samples from the same task share the same CN parameters`

$\gamma^{tr}_{s-1}, \beta^{tr}_{s-1} = \gamma_{s-1}.repeat\_interleave(k, 1), \beta_{s-1}.repeat\_interleave(k, 1)$

$\gamma_s = \gamma_{s-1} - \lambda_{innner} \nabla_\gamma \mathcal{L}^{\text{sup}}(f_{w, \gamma^{tr}_{s-1}, \beta^{tr}_{s-1}}(X_{train}), Y_{train})$

$\beta_s = \beta_{s-1} - \lambda_{innner} \nabla_\beta \mathcal{L}^{\text{sup}}(f_{w, \gamma^{tr}_{s-1}, \beta^{tr}_{s-1}}(X_{train}), Y_{train})$

$\gamma^{test}_s, \beta^{test}_s = \gamma_s.repeat\_interleave(k', 1), \beta_s.repeat\_interleave(k', 1)$

$w = w - \lambda_{outer} \nabla_w \mathcal{L}^{\text{sup}} \left(f_{w, \gamma^{test}_s, \beta^{test}_s}(X_{test}), Y_{test})\right)$

$\beta_s, \gamma_s = \beta_s.detach(), \gamma_s.detach()$ `// WarpGrad detach to not backprop through multiple steps`

**return** $w$

**Subroutine** *Meta-test(*$f$*,* $w$*,* $\mathcal{L}^{\text{sup}}$*,* $\lambda_{inner}$*,steps,*$X_{train}$*,* $Y_{train}$*,* $X_{test}$*)*

`// Assuming a single task, although we could evaluate multiple tasks in parallel as in meta-training.`

$\gamma_0 = \mathbf{1}_{1, \sum_l m_l}$ `// single` $\gamma, \beta$ `because we only have one task`

$\beta_0 = \mathbf{0}_{1, \sum_l m_l}$ **for** $1 \leq s \leq steps$ **do**

$\gamma^{tr}_{s-1}, \beta^{tr}_{s-1} = \gamma_{s-1}.repeat\_interleave(k, 1), \beta_{s-1}.repeat\_interleave(k, 1)$

$\gamma_s = \gamma_{s-1} - \lambda_{innner} \nabla_\gamma \mathcal{L}^{\text{sup}}(f_{w, \gamma^{tr}_{s-1}, \beta^{tr}_{s-1}}(X_{train}), Y_{train})$

$\beta_s = \beta_{s-1} - \lambda_{innner} \nabla_\beta \mathcal{L}^{\text{sup}}(f_{w, \gamma^{tr}_{s-1}, \beta^{tr}_{s-1}}(X_{train}), Y_{train})$

$\gamma^{test}_{steps}, \beta^{test}_{steps} = \gamma_{steps}.repeat\_interleave(k', 1), \beta_{steps}.repeat\_interleave(k', 1)$ **return** $f_{w, \gamma^{test}_{steps}, \beta^{test}_{steps}}(X_{test})$

## E Experimental details of physics experiments

**Dataset generation** As mentioned in the main text, 5-body systems are chaotic and most random configurations are unstable. To generate our dataset we used Finite Differences to optimize 5-body dynamical systems that were stable for 200 steps (no planet collisions and no planet outside a predetermined grid) and then picked the first 100 steps of their trajectories, to ensure dynamical stability. To generate each trajectory, we randomly initialized 5 planets within a 2D grid of size $w = 600, h = 300$, with a uniform probability of being anywhere in the central grid of size $w/2, h/2$, each with a mass sampled from a uniform between $[0.15, 0.25]$ (arbitrary units) and with random starting velocity initialized with a Gaussian distribution. We then use a 4th order Runge-Kutta integrator to accurately simulate the ODE of the dynamical system until we either reach 200 steps, two planets get within a certain critical distance from each other or a planet gets outside the pre-configured grid. If the trajectory reached 200 steps, we added it to the dataset; otherwise we made a small random perturbation to the initial configuration of the planets and tried again. If the new perturbation did not reach 200 steps, but lasted longer we kept the perturbation as the new origin for future initialization perturbations, otherwise we kept our current initialization. Once all the datasets were generated we picked those below a threshold mean mass and partitioned them randomly into train and test. Finally, we normalize each of the 25 dimensions (5 planets and for each planet $x$,$y$,$v_x$,$v_y$,$m$) to have mean zero and standard deviation one. For inputs, we use each state and as target we use the next state; therefore, each trajectory gives us 100 pairs.

For more details, we attach the code that generated the dataset.

**Implementation of tailoring, meta-tailoring and** CNGRAD All of our code is implemented in PyTorch [38], using the higher library [22](`https://github.com/facebookresearch/higher`) to implement the second-order version of CNGRAD. We implemented a 3-layer feedforward neural network, with a conditional normalization layer after each layer except the final regression layer. The result of the network was added to the input, thus effectively predicting the delta between the current state and the next state. For both the first-order and second-order versions of CNGRAD, we used the detachment of WarpGrad (line 12 in algorithm 3). For more details, we also attach the implementation of the method.

**Compute and hyper-parameter search** To keep the evaluation as strict as possible, we searched all the hyper-parameters affecting the inductive baseline and our tailoring versions with the baseline and simply copied these values for tailoring and meta-tailoring. For the latter two, we also had to search for $\lambda_{tailor}$.

The number of epochs was 1000, selected with the inductive baseline, although more epochs did not substantially affect performance in either direction. We note that meta-tailoring performance plateaued earlier in terms of epochs, but we left it the same for consistency. Interestingly, we found that regularizing the physics loss (energy and momentum conservation) helped the inductive baseline, even though the training data already has 0 physics loss. We searched over $[10^{-4}, 3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}, 10^{-2}]$ for the weight assigned to the physics loss and chose $2 \cdot 10^{-3}$ for best performance in the inductive baseline. To balance between energy and momentum losses we multiplied the momentum loss by 10 to roughly balance their magnitudes before adding them into a single physics loss, this weighting was not searched. We copied these settings for meta-tailoring.

In terms of the neural network architecture, we chose a simple model with 3 hidden layers of the same size and tried $[128, 256, 512]$ on the inductive baseline, choosing $512$ and deciding not to go higher for compute reasons and because we were already able to get much lower training loss than test loss. We copied these settings for the meta-tailoring setup. We note that since there are approximately $O(m_h^2)$ weight parameters, yet only $O(m_h)$ affine parameters used for tailoring, adding tailoring and meta-tailoring increase parameters roughly by a fraction $O(1/m_h)$, or about $0.2\%$. Also in the inductive baseline, we tried adding Batch Normalization [30], since it didn't affect performance we decided not to add it.

We chose the tailoring step size parameter by trying $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$, finding $10^{-3}$ worked well while requiring less steps than using a smaller step size. We therefore used this step for meta-tailoring as well, which worked well for first-order CNGRAD, but not for second-order CNGRAD, whose training diverged. We thus lowered the tailoring step size to $10^{-4}$ for the second-order version, which worked well. We also tried clipping the inner gradients to increase the stability of the second-

order method; since gains on preliminary experiments were small, we decided to keep it out for simplicity.

For meta-tailoring we only tried 2 and 5 tailoring steps (we wanted more than one step to show the algorithm capability, but few tailoring steps to keep inference time competitive). Since they worked similarly well, we chose 2 steps to have a faster model. For the second-order version we also used 2 steps, which performed much better than the inductive baseline and tailoring, but worse than the first-order version reported in the main text(about $20\%$ improvement of the second-order version vs. $7\%$ of tailoring and $35\%$ improvement of the first-order version).

For the baseline of optimizing the output we tried a step size of $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$. Except for a step size of $10^{-1}$, results optimized the physics loss and always achieved a very small improvement, without overfitting to the physics loss. We thus chose a big learning rate and high number of steps to report the biggest improvement, of $0.7\%$.

**Runs, compute and statistical confidence:** we ran each method 2 times and averaged the results. Note that the baseline of optimizing the output and *tailoring* start from the inductive learning baseline, as they are meant to be methods executed after regular inductive training. This is why both curves start at the same point in Figure 2. For those methods, we report the standard deviation of the mean estimate of the *improvement*, since they are executed on the same run. Note that the standard deviation of the runs would be higher, but less appropriate. For meta-tailoring, we do use the standard deviation of the mean estimate of both runs, since they are independent from the inductive baseline.

All experiments were performed on a GTX 2080 Ti with 4 CPU cores.

# F Experimental details on real pendulum

We modify the real pendulum of Hamiltonian Neural Networks [23]. In particular we pick the energy of the system and use it as a tailoring loss. Greydanus et al. [23] train a vanilla MLP and show that its non-conservation of energy results in poor generalization from train to test for long-term predictions. With HNNs that automatically discover an energy function and encode hamiltonian dynamics into the network architecture, the system conserves this proxy energy even in long predictions, resulting in better generalization. We meta-tailor the vanilla MLP, with no change in its architecture, beyond adding CN layers to efficiently perform tailoring. We try different inner learning rates ($1e-3, 1e-2, 1e-1, 1e0$) as well as number of steps $(1, 2, 3)$ and evaluate on long-term *training* loss. Since training is 4 times as long as test, we divide training into 4 equally-big trajectories and choose the configuration with the best loss: 3 steps and $1e-1$ inner learning rate. It is worth noting that these long term predictions use scipy's ODE integrator, which is also used for the vanilla MLP as well as for HNNs. We see that, by not fully enforcing energy conservation, meta-tailoring improves over both an inductive baseline of the same architecture and HNNs.

Experiments were performed with a Volta V-100 and 10 CPU cores, taking a couple of hours to run in total.

# G Experimental details on contrastive learning

We take the implementation of SimCLR [13] from `https://github.com/leftthomas/SimCLR` evaluating on CIFAR-10 [31].

As detailed in the main text we train the vanilla SimCLR to get an unsupervised representation. We than train *only* the linear layer with different amounts of training data, from 50 to 5000 points per class. Vanilla SimCLR follows regular inductive learning with supervised labels for the linear layer. Meta-tailoring uses the same augmentations provided by SimCLR and minimizes the SimCLR loss on each particular input before feeding the tailored representations to the linear layer. The linear layer is trained to take these adapted representations. We tried different hyper-parameters: $[4, 8, 16]$ augmentations, $1, 2,$ inner optimization steps, inner learning rate of $[1e-1, 3e-1, 1e0, 3e0, 1e1, 3e1]$ and whether to tailor the CN layers of the CNN representation or tailor the representations $h$ directly. We found very consistent results where all stable inner optimizations improved over vanilla SimCLR, and longer optimizations with larger learning rates and more augmentations gave bigger improvements. Tailoring the CN layers or the representations directly didn't make a big effect, the latter being slightly more

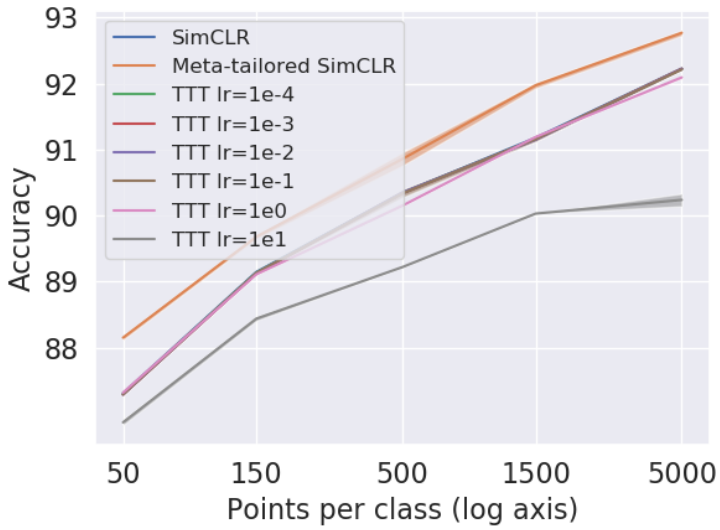


Figure 5: Test-time training (TTT) with its original rotation-prediction auxiliary task performs worse than vanilla SimCLR. Performance degrades as we increase the inner learning rate, thus increasing its power.

stable, and providing somewhat larger results. It is also much faster as we do not need to back-prop back through the CNN. We thus chose 3 steps $1e1$ learning rate, 16 augmentations and tailoring the representations directly. For TTT we kept the 3 steps and tried $0, 1e{-}4, 1e{-}3, 1e{-}2, 1e{-}1, 1e0, 1e1$ learning rates. We noticed that all these inner optimizations were stable, yet performance degraded with learning rate. Thus the best learning rate was $0$, equivalent to not doing TTT.

For all methods we follow the code-base and keep the best validation. Keeping running averages or choosing concrete epochs gave very similar results because of the stability of training a linear layer. We averaged over 5 different trainings of the linear layer, all using the same SimCLR base. The TTT baseline uses the best SimCLR epoch for each of these 5 runs, so that using TTT with lr=0 gives exactly the same results as the baseline.

For TTT we trained on the rotation prediction task proposed in the original paper. To minimize differences, we use the same architecture as the MLP from SimCLR, except with 4 output logits, one per rotation. It achieves $80.5\%$ test accuracy on rotation prediction. TTT proposes to back-propagate this rotation prediction loss back at test-time, but does not take this procedure into account at training time. We consistently find that TTT worsens the performance, with the gap becoming worse as we increase the learning rate. We think the reason why this loss is helpful in a very similar dataset and architecture in Sun et al. [46], yet hurts performance in this case is due to two factors. First, it can be observed in the original paper that rotation-prediction provides consistent, but small gains, in the 1-sample case, with much larger gains in its online multi-sample version. Second, Sun et al. [46] focus on out-of-distribution generalization, where weights are trained on a different data distribution and are thus sub-optimal. The linear layer receiving OOD inputs in this same-distribution case hurts performance, but in their OOD application, even the unmodified inputs were already OOD. Meta-tailoring takes the adaptation into account, thus the inputs of the linear layer are always in-distribution, thus being able to help performance.

Experiments were performed with a set of Volta V-100 and 10 CPU cores. SimCLR takes around a day to train. All other experiments training the linear layer from different initializations and for all data quantities take a few hours for a single set of hyper-parameters.

## H Toy adversarial examples experiment

We illustrate the tailoring process with a simple illuminating example using the data from Ilyas et al. [29]. They use discriminant analysis on the data in Figure 6 and obtain the purple linear separator. It

has the property that, under assumptions about Gaussian distribution of the data, points above the line are more likely to have come from the blue class, and those below, from the red class. This separator is not very adversarially robust, in the sense that, for many points, a perturbation with a small $\delta$ would change the assigned class. We improve the robustness of this classifier by tailoring it using the loss $\mathcal{L}^{\text{tailor}}(x, \theta) = \text{KL}(\phi(f_\theta(x)) \,||\, \phi(f_\theta(x + \text{argmax}_{|\delta|<\varepsilon} \sum_j e^{f_\theta(x+\delta)_j})))$, where KL represents the KL divergence, $\phi$ is the logistic function, and $\phi(f_\theta(x)_i)$ is the probability of $x$ being in class $i$, so that $\phi(f_\theta(x))$ represents the entire class distribution.

With this loss, we can adjust our parameters $\theta$ so that the KL divergence between our prediction at $x$ is closer to the prediction at perturbed point $x + \delta$, over all perturbations in radius $\varepsilon$. Note that we initialized the models with the weights of Cohen et al. [14] to speed up training in all ImageNet experiments and to avoid training divergence for CIFAR-10 with $\sigma = 1$ (this divergence was already noted by Zhai et al. [53]). Each of the curves in Figure 3 corresponds to a decision boundary induced by tailoring the original separator with a different value for the maximum perturbation $\varepsilon$. Note that the resulting separators are non-linear, even though we are tailoring a linear separator, because the tailoring is local to the prediction point. We also have the advantage of being able to choose different values of $\varepsilon$ at prediction time.

**Hyper-parameters** the model does not have any hyper-parameters, as we use the model from Ilyas et al. [29], which is based on the mean $\mu$ and standard deviation $\sigma$ of the Gaussians. For tailoring, we used a $5 \times 5$ grid to initialize the inner optimization to find the point of highest probability within the $\epsilon$-ball. Using a single starting point did not work as gradient descent found a local optima. Using more ($10 \times 10$) did not improve results further, while increasing compute. We also experimented between doing a weighted average of the predictions by their energy to compute the tailoring loss or picking the element with the biggest energy. Results did not seem to differ much (likely because likelihood distributions are very peaked), so we picked the simplest option of imitating the element of highest probability. Doing a single tailoring step already worked well (we tried step sizes of $10^{-1}, 1, 10, 30$ with 10 working best), so we kept that for simplicity and faster predictions.

Regarding compute, this experiment can be generated in a few minutes using a single GTX 2080 Ti.

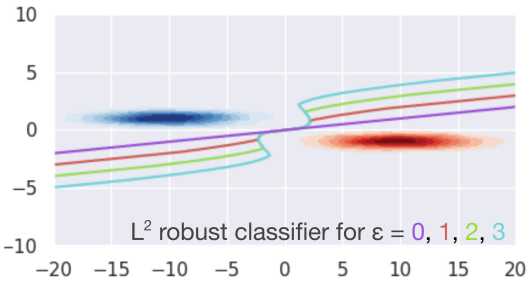


Figure 6: Decision boundary of our model at multiple levels of robustness on an example from Ilyas et al. [29].

# I Experimental details of adversarial experiments

Results for CIFAR-10 and ImageNet experiments comparing to state-of-the-art methods can be found in tables 7, 8, and 9. Table 7 only includes results for Randomized Smoothing (RS).

**Hyper-parameters and other details of the experiments** there are just three hyper-parameters to tweak for these experiments, as we try to remain as close as possible to the experiments from Cohen et al. [14]. In particular, we tried different added noises $\nu \in [0.05, 0.1, 0.2]$ and tailoring inner steps $\lambda \in [10^{-3}, 10^{-2}, 10^{-1}, 10^{0}]$ for $\sigma = 0.5$. To minimize compute, we tried these settings by tailoring (not meta-tailoring) the original model and seeing its effects on the smoothness and stability of optimization, choosing $\nu = 0.1, \lambda = 0.1$ (the fact that they're the same is a coincidence). We chose to only do a single tailoring step to reduce the computational burden, since robustness certification is very expensive, as each example requires 100k evaluations (see below). For simplicity and to avoid

| $\sigma$ | **Method** | 0.0 | 0.25 | 0.5 | 0.75 | 1.0 | 1.25 | 1.5 | 1.75 | 2.00 | 2.25 | ACR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.25 | (Inductive) RS | 0.75 | 0.60 | 0.43 | 0.26 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.416 |
| | Meta-tailor RS | **0.80** | **0.66** | **0.48** | 0.29 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | **0.452** |
| 0.50 | (Inductive) RS | 0.65 | 0.54 | 0.41 | 0.32 | 0.23 | 0.15 | 0.09 | 0.04 | 0.00 | 0.00 | 0.491 |
| | Meta-tailor RS | 0.68 | 0.57 | 0.45 | **0.33** | 0.23 | 0.15 | 0.08 | 0.04 | 0.00 | 0.00 | **0.542** |
| 1.00 | (Inductive) RS | 0.47 | 0.39 | 0.34 | 0.28 | 0.21 | 0.17 | **0.14** | 0.08 | 0.05 | 0.03 | 0.458 |
| | Meta-tailor RS | 0.50 | 0.43 | 0.36 | 0.30 | **0.24** | **0.19** | **0.14** | **0.10** | **0.07** | **0.05** | **0.546** |

Figure 7: Percentage of points with certificate above different radii, and average certified radius (ACR) for on the CIFAR-10 dataset. Meta-tailoring improves the Average Certification Radius by $8.6\%, 10.4\%, 19.2\%$ respectively. Results for Cohen et al. [14] are taken from Zhai et al. [53] because they add more measures than the original work, with similar results.

| $\sigma$ | **Method** | 0.0 | 0.25 | 0.5 | 0.75 | 1.0 | 1.25 | 1.5 | 1.75 | 2.00 | 2.25 | ACR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.25 | RandSmooth | 0.75 | 0.60 | 0.43 | 0.26 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.416 |
| | Salman | 0.74 | 0.67 | 0.57 | 0.47 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.538 |
| | MACER | **0.81** | **0.71** | **0.59** | 0.43 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | **0.556** |
| | Meta-tailored | 0.80 | 0.66 | 0.48 | 0.29 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.452 |
| 0.50 | RandSmooth | 0.65 | 0.54 | 0.41 | 0.32 | 0.23 | 0.15 | 0.09 | 0.04 | 0.00 | 0.00 | 0.491 |
| | Salman | 0.50 | 0.46 | 0.44 | 0.40 | 0.38 | 0.33 | 0.29 | 0.23 | 0.00 | 0.00 | 0.709 |
| | MACER | 0.66 | 0.60 | 0.53 | **0.46** | 0.38 | 0.29 | 0.19 | 0.12 | 0.00 | 0.00 | **0.726** |
| | Meta-tailored | 0.68 | 0.57 | 0.45 | 0.33 | 0.23 | 0.15 | 0.08 | 0.04 | 0.00 | 0.00 | 0.542 |
| 1.00 | RandSmooth | 0.47 | 0.39 | 0.34 | 0.28 | 0.21 | 0.17 | 0.14 | 0.08 | 0.05 | 0.03 | 0.458 |
| | Salman | 0.45 | 0.41 | 0.38 | 0.35 | **0.32** | 0.28 | **0.25** | **0.22** | **0.19** | **0.17** | 0.787 |
| | MACER | 0.45 | 0.41 | 0.38 | 0.35 | **0.32** | **0.29** | **0.25** | **0.22** | 0.18 | 0.16 | **0.792** |
| | Meta-tailored | 0.50 | 0.43 | 0.36 | 0.30 | 0.24 | 0.19 | 0.14 | 0.10 | 0.07 | 0.05 | 0.546 |

Figure 8: Percentage of points with certificate above different radii, and average certified radius (ACR) for on the CIFAR-10 dataset, comparing with SOA methods. In contrast to pretty competitive results in ImageNet, meta-tailoring improves randomized smoothing, but not enough to reach SOA. It is worth noting that the SOA algorithms could also likely be improved via meta-tailoring.

| $\sigma$ | **Method** | 0.0 | 0.5 | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | ACR |
|---|---|---|---|---|---|---|---|---|---|
| 0.25 | RandSmooth | 0.67 | 0.49 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.470 |
| | Salman | 0.65 | 0.56 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.528 |
| | MACER | 0.68 | **0.57** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | **0.544** |
| | Meta-tailored RS | **0.72** | 0.55 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.494 |
| 0.50 | RandSmooth | 0.57 | 0.46 | 0.37 | 0.29 | 0.00 | 0.00 | 0.00 | 0.720 |
| | Salman | 0.54 | 0.49 | **0.43** | **0.37** | 0.00 | 0.00 | 0.00 | 0.815 |
| | MACER | 0.64 | 0.53 | **0.43** | 0.31 | 0.00 | 0.00 | 0.00 | **0.831** |
| | Meta-tailored RS | 0.66 | 0.54 | 0.42 | 0.31 | 0.00 | 0.00 | 0.00 | 0.819 |
| 1.00 | RandSmooth | 0.44 | 0.38 | 0.33 | 0.26 | 0.19 | 0.15 | 0.12 | 0.863 |
| | Salman | 0.40 | 0.38 | 0.33 | 0.30 | **0.27** | **0.25** | **0.20** | 1.003 |
| | MACER | 0.48 | 0.37 | 0.34 | 0.30 | 0.25 | 0.18 | 0.14 | 1.008 |
| | Meta-tailored RS | 0.52 | 0.45 | 0.36 | 0.31 | 0.24 | 0.20 | 0.15 | **1.032** |

Figure 9: Percentage of points with certificate above different radii, and average certified radius (ACR) for on the ImageNet dataset, including other SOA methods. Randomized smoothing with meta-tailoring are very competitive with other SOA methods, including having the biggest ACR for $\sigma = 1$.

excessive tuning, we chose the hyper-parameters for $\sigma = 0.5$ and copied them for $\sigma = 0.25$ and $\sigma = 1$. As mentioned in the main text, $\sigma = 1$ required initializing our model with that of Cohen et al. [14] (training wasn't stable otherwise), which is easy to do using CNGRAD.

In terms of implementation, we use the codebase of Cohen et al. [14](`https://github.com/locuslab/smoothing`) extensively, modifying it only in a few places, most notably in the architecture to include tailoring in its forward method. It is also worth noting that we had to deactivate their disabling of gradients during certification, because tailoring requires gradients. We chose to use the first-order version of CNGRAD which made it much easier to keep our implementation very close to the original. It is likely that doing more tailoring steps would result in better performance.

We note that other works focused on adversarial examples, such as Zhai et al. [53], Salman et al. [43], improve on Cohen et al. [14] by bigger margins. However, tailoring and meta-tailoring can also improve a broad range of algorithms in applications outside of adversarial examples. Moreover, they could also improve these new algorithms further, as these algorithms can also be tailored and meta-tailored.

**Compute requirements** For the CIFAR-10 experiments building on Cohen et al. [14], each training of the meta-tailored method was done in a single GTX 2080 Ti for 6 hours. Certification was much more expensive (10k examples with 100k predictions each for a total of $10^9$ predictions). Since certifications of different images can be done in parallel, we used a cluster consisting of 8 GTX 2080 Ti, 16 Tesla V-100, and 40 K80s (which are about 5 times slower), during 36 hours.

For the ImageNet experiments, we fine-tuned the original models for 5 epochs; each took 18 hours on 1 Tesla V-100. We then used 30 Tesla V-100 for 20 hours for certification.