References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Greydanus et al., 2019] Greydanus, S., Dzamba, M., and Yosinski, J. (2019). Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363.

[Ha and Schmidhuber, 2018] Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

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

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

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

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

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

[Kidambi et al., 2020] Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

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

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

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

[Mirowski et al., 2016] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

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

[Nagabandi et al., 2020] Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. (2020). Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112.

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

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

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

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

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

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

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

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

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

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

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

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

[Tschiatschek et al., 2018] Tschiatschek, S., Sahin, A., and Krause, A. (2018). Differentiable submodular maximization. *arXiv preprint arXiv:1803.01785*.

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

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

[Zhai et al., 2020] Zhai, R., Dan, C., He, D., Zhang, H., Gong, B., Ravikumar, P., Hsieh, C.-J., and Wang, L. (2020). Macer: Attack-free and scalable robust training via maximizing certified radius. In *International Conference on Learning Representations*.

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

Due to the page limit this appendix only contains appendix A-D, for appendix E-J see the arxiv version of this work:`https://arxiv.org/abs/2009.10623`.

A Theoretical motivations of meta-tailoring

In this section, we study 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 in general may make them less related.

A.1 Meta-tailoring with a contrastive tailoring loss

Contrastive learning [Hadsell et al., 2006] has seen significant successes recently in problems of semi-supervised learning from images [Oord et al., 2018, He et al., 2019, Chen et al., 2020]. The main idea is to create multiple versions of each training image, and learn a representation in which variations of the same image are very close and variations of different images are far apart. Typical variations 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 θ_x obtained with a training dataset $S = ((x_i, y_i))_{i=1}^n$ and tailoring with the contrastive loss at the prediction point x. Theorem 2 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 $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{x,S})]$ (which is defined and analyzed in more detail in Appendix C), 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 the meta-tailoring algorithm, and the uniform stability ζ of the meta-tailoring algorithm. As a complement, instead of using the uniform stability ζ, Theorem 6 (detailed in appendix E) provides a similar bound with the Rademacher complexity [Bartlett and Mendelson, 2002] $\mathcal{R}_n(\mathcal{L}^{\text{sup}} \circ \mathcal{F})$ of the set $\mathcal{L}^{\text{sup}} \circ \mathcal{F} = \{(x, y) \mapsto \mathcal{L}^{\text{sup}}(f_{\theta_x}(x), y) : (x \mapsto f_{\theta_x}(x)) \in \mathcal{F}\}$. All proofs in this paper are deferred to Appendix E.

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 ζ-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 2. *Let $S \mapsto f_{\theta_{x,S}}(x)$ be a uniformly ζ-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 $\kappa \in [0, 1]$, $\mathbb{E}_{x,y}[\mathcal{L}^{\text{sup}}(f_{\theta_{x,S}}(x), y)] \leq \kappa \mathbb{E}_x\left[\mathcal{L}_{\text{cont}}(x, \theta_{\mathbf{x},S})\right] + (1 - \kappa)\mathcal{J}$, where $\mathcal{J} = \frac{1}{n}\sum_{i=1}^n \mathcal{L}^{\text{sup}}(f_{\theta_{x_i,S}}(x_i), y_i) + \frac{\zeta}{n} + (2\zeta + c)\sqrt{(\ln(1/\delta))/(2n)}$, 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 θ instead of a $\theta_{\mathbf{x}}$ tailored to each input x. These differences are marked in bold green in Definition 1 and Theorem 2. 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 (in addition to the empirical supervised loss), 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, the meta-tailoring can avoid this extra generalization error by directly minimizing $\mathbb{E}_x[\mathcal{L}_{\text{cont}}(x, \theta_{x,S})]$.

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 at prediction time), Theorem 2 still illustrates competitive generalization bounds of meta-tailoring with small κ, when compared to classical induction. For example, with $\kappa = 0$, it provides standard types of generalization bounds with the uniform stability for meta-tailoring algorithms. Even in this case, 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.

A.2 Meta-tailoring with general tailoring losses

The benefits of meta-tailoring go beyond contrastive learning: the following remark provides the generalization bounds for meta-tailoring with any tailoring loss $\mathcal{L}^{\text{tailor}}(x, \theta)$ and an arbitrary supervised loss $\mathcal{L}^{\text{sup}}(f_\theta(x), y)$.

Remark 1. *For any function φ 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 2 and 6 hold with the map $\mathcal{L}_{\text{cont}}$ being replaced by the function $\varphi \circ \mathcal{L}^{\text{tailor}}$.*

Remark 1 shows the benefits of meta-tailoring through its effects on three factors: the expected unlabeled loss $\mathbb{E}_x[\varphi(\mathcal{L}^{\text{tailor}}(x, \theta_{x,S}))]$, uniform stability ζ, and 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 $\mathbb{E}_x[\varphi(\mathcal{L}^{\text{tailor}}(x, \theta_{x,S}))]$, 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 φ is monotonically increasing and $\mathcal{L}^{\text{tailor}}(x, \theta)$ represents the physical constraints at each input x (as in the application in section 4.1), then classic induction requires the physical constraints of neural networks at the *training* points to generalize to the physical constraints at *unseen* (e.g., testing) points. Meta-tailoring avoids this requirement by directly minimizing violations of the physical constraints at each point at prediction time.

Another potential benefit of meta-tailoring can be understood through the improvement in the *parameter stability* ζ_θ 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. In the case of the meta-tailoring method of $\theta_{x,S} = \hat{\theta}_S - \lambda \nabla \mathcal{L}^{\text{tailor}}(x, \hat{\theta}_S)$, we can obtain an improvement on the parameter stability ζ_θ 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 $\cos(v_1, v_2)\frac{\|v_1\|}{\|v_2\|} > \frac{1}{2}$ where $\cos(v_1, v_2)$ 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 ζ and the parameter stability ζ_θ 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 $|\mathcal{L}^{\text{sup}}(f_{\theta_{x,S}}(x), y) - \mathcal{L}^{\text{sup}}(f_{\theta_{x,S'}}(x), y)| \leq C\|\theta_{x,S} - \theta_{x,S'}\| \leq \frac{C\zeta_\theta}{n}$.

B On the conditions in Theorem 1 and Corollary 1

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., $\sigma_\alpha(x) \to \text{relu}(x)$ as $\alpha \to \infty$, where relu represents the ReLU function.

In Theorem 1 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 $g^{(i)}(x) \neq 0$ satisfies it without the normalization.

C 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 ℓ_{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 θ 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 $\bar{\theta}^{(H+1)}$,*

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

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 2. 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, γ_i, and β_i are the transposes of the i-th row vectors of X, γ and β, 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 γ, β 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 γ_s, β_s which in turn affect $\mathcal{L}^{\text{sup}}$. Similar to MAML [Finn et al., 2017], 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 β, γ 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 4.1 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 [Flennerhag et al., 2019] 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 1. 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 4.2. Another advantage is that it makes training more stable, reducing 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

Algorithm 1: CNGRAD for meta-tailoring

Subroutine *Training(*f*,* $\mathcal{L}^{\text{sup}}$*,* λ_{sup}*,* $\mathcal{L}^{\text{tailor}}$*,* λ_{tailor}*,* $steps$*,*$((x_i, y_i))_{i=1}^n$*)*
  randomly initialize w `// All parameters except` γ, β`; 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**
        $\gamma_s = \gamma_{s-1} - \lambda_{tailor} \nabla_\gamma \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$
        $\beta_s = \beta_{s-1} - \lambda_{tailor} \nabla_\beta \mathcal{L}^{\text{tailor}}(w, \gamma_{s-1}, \beta_{s-1}, X)$
        $w = w - \lambda_{sup} \nabla_w \mathcal{L}^{\text{sup}}\left(f_{w, \gamma_s, \beta_s}(X), Y\right))$
        $\beta_s, \gamma_s = \beta_s.detach(), \gamma_s.detach()$ `// Optional operation: WarpGrad detach to avoid back-proping through multiple steps; reducing memory, and increasing stability, but adding bias.`
  return w
Subroutine *Prediction(*f*,* w*,* $\mathcal{L}^{\text{tailor}}$*,* λ*,* $steps$*,* 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 $f_{w, \gamma_{steps}, \beta_{steps}}(X)$

across tasks. We show the pseudo-code for few-shot supervised learning in algorithm 2. 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 γ, β 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 γ, β, 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 2: CNGRAD for meta-learning

Subroutine *Meta-training(f, $\mathcal{L}^{\text{sup}}$, λ_{inner}, λ_{outer}, steps,$\mathcal{T}$)*
  randomly initialize w `// All parameters except` γ, β `; 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, $\mathcal{L}^{\text{sup}}$, λ_{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` γ, β `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 $f_{w, \gamma^{test}_{steps}, \beta^{test}_{steps}}(X_{test})$