Meta-Learning with Warped Gradient Descent

Sebastian Flennerhag
DeepMind
The Alan Turing Institute
flennerhag@google.com

Andrei A. Rusu
DeepMind
andreirusu@google.com

Razvan Pascanu
DeepMind
razp@google.com

Francesco Visin
DeepMind
razp@google.com

Hujun Yin
University of Manchester
hujun.yin@manchester.ac.uk

Raia Hadsell
DeepMind
raia@google.com

Abstract

Current methods to gradient-based meta-learning rely on backpropagating through the learning process, limiting their scope to few-shot learning. In this work, we introduce *Warped Gradient Descent* (WarpGrad), a family of modular optimisers that can scale to arbitrary adaptation processes. WarpGrad methods meta-learn to warp task loss surfaces across the joint task-parameter distribution to facilitate gradient descent, achieved by sharing fixed, meta-learned layers across task learners that precondition task parameters during task adaptation. We find that WarpGrad optimisers converge faster and generalise better in a variety of settings, including few-shot, supervised, continual, and reinforcement learning.

1 Introduction

Empirical evidence from gradient-based few-shot learning suggests enforcing some form of strict parameter sharing improve performance by reducing overfitting [22, 31, 33], more robust convergence [48, 38], or superior final performance [35, 42, 21]. Strict parameter sharing can also encode a meta-learned update-rule [25, 22, 36]. These findings suggest that meta-learning fixed shared parameters offers a strong inductive bias in few-shot learning. However, it remains unclear whether this bias holds more generally. Unfortunately, because they backpropagate through the adaptation process, we cannot scale them to general forms of meta-learning [47, 7]. We resolve these limitations in a novel framework, Warped Gradient Descent (WarpGrad), that preserves the inductive bias of gradient-based few-shot learning while generalising to meta-learning over arbitrary adaptation processes.

WarpGrad is a family of modular optimisation methods that meta-learn shared *warp parameters* across task learners such that gradient-based adaptation of *task parameters* leads to fast learning and generalisation. WarpGrad methods belong to a class of meta-learners that precondition native gradients [25, 22, 36] that are endowed with convergence guarantees with respect to task parameters, in contrast to recurrent optimisers [39, 2, 24]. Uniquely among gradient-based meta-learners, WarpGrad methods meta-learn to precondition gradients over the joint task parameter distribution. This fundamental departure from the paradigm of backpropagation through the adaptation process lets them scale beyond few-shot learning without incurring a short-horizon bias [47]. WarpGrad methods can meta-learn over arbitrary adaptation processes, even if a final loss is not well defined.

2 Warped Gradient Descent

MAML In gradient-based few-shot learning, as defined by the Model-Agnostic Meta-Learner [MAML; 6], a task τ is defined by a training set $\mathcal{D}^{\tau}_{\text{train}}$ and a validation set $\mathcal{D}^{\tau}_{\text{test}}$, with

3rd Workshop on Meta-Learning at NeurIPS 2019, Vancouver, Canada.

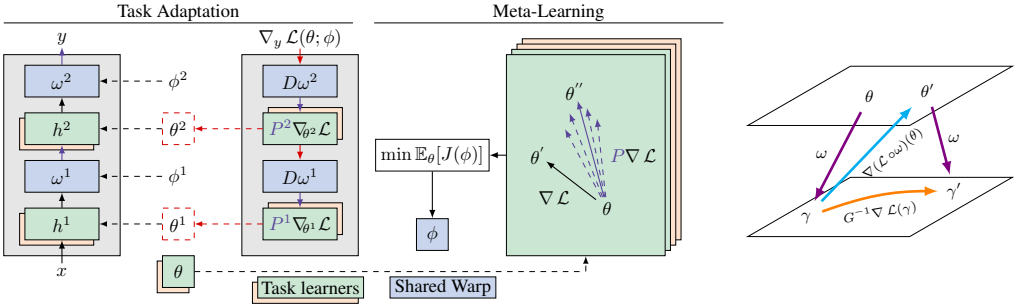


Figure 1: *Right:* WarpGrad define an optimiser (ω) by embedding it in task learners (h) through shared warp layers ($\omega^{1,2}$). These are fixed during adaptation of task parameters θ; backpropagating through warp layers yields a form of preconditioning (P). We meta-learn warps over the joint task parameter distribution to produce preconditioning that facilitates task learning. *Right:* gradient descent under WarpGrad amounts to Riemannian descent under a meta-learned metric G.

$\mathcal{D} \coloneqq \{(x_i, y_i)\}_{i=1}^n$ for some small n. We adapt a task learner h via a gradient-based update rule $\theta \leftarrow \theta - \alpha P(\theta; \phi) \nabla \mathcal{L}(\theta)$ with a meta-learned preconditioner P and initialisation θ_0,

$$C^{\text{MAML}}(\theta_0, \phi) \coloneqq \sum_{\tau \sim p(\tau)} \mathcal{L}_{\mathcal{D}^\tau_{\text{test}}} \left(\theta_0 - \alpha \sum_{k=0}^{K-1} P(\theta_k^\tau; \phi) \nabla \mathcal{L}_{\mathcal{D}^\tau_{\text{train}}}(\theta_k^\tau) \right). \tag{1}$$

This framework backpropagates through the adaptation process, limiting it to few-shot learning as it is computational expensive, susceptible to exploding/vanishing gradients and the credit assignment problem [3, 47]. Our goal is to develop a framework for meta-learning that overcomes all three limitations. In few-shot meta-learning, Meta-SGD [25] parameterise P as a fixed diagonal matrix while Meta-Curvature [36] allows for a block-diagonal P. T-Nets [23] precondition feed-forward networks, $h(x, \theta) = Wx$, by inserting non-learnable projections T. Backpropagating through $h(x, \theta, \phi) = TWx$ automatically preconditions task gradients and defines P via $P(\theta; \phi) \nabla \mathcal{L}_{\mathcal{D}^\tau_{\text{train}}}(\theta) = \nabla \mathcal{L}_{\mathcal{D}^\tau_{\text{train}}}(\theta; \phi)$, where P has a block-diagonal structure with block entries TT^T. We build on these works by (a) generalising embedded preconditioning beyond a block-diagonal structure (b) and deriving a novel meta-objective for general-purpose meta-learning.

Generalised preconditioning To generalise embedded preconditioning, we consider an arbitrary neural network $h = h^l \circ \cdots \circ h^1$, for instance, an MLP or an LSTM (see Appendix A for how to design WarpGrad optimisers), that we interleave with fixed shared layers ω, $h = \omega^l \circ h^l \circ \cdots \circ \omega^1 \circ h^1$. The gradient of task parameters θ^i of a layer h^i embeds preconditioning via backpropagation:

$$\frac{\partial \mathcal{L}}{\partial \theta^i} = \mathbb{E}\left[\nabla \ell^T \left(\prod_{j=0}^{l-(i+1)} D_x \omega^{l-j} D_x h^{l-j} \right) D_x \omega^i D_{\theta^i} h^i \right], \tag{2}$$

where D_x and D_θ denote the Jacobian with respect to input and parameters, respectively. Prior works enforce independent Jacobians $D_x \omega = T$. In WarpGrad, we relax this assumption by allowing general functional forms for ω. This gives rise to *gradient warping*, where the forward pass modulates Jacobians $D_x \omega$, which then precondition task parameter gradients $D_\theta h$ in the backward pass. This generalisation has profound implications. In our multi-shot supervised learning experiment, we find that allowing for non-linearities in ω improves accuracy from 83% to 89%; in our RL experiment, we introduce a novel form of meta-learned preconditioning by recurrent warps that modulate the backpropagation through time operator; in contrast, linear preconditioning fail to provide any benefits. This generality sacrifices an explicit Riemann metric for P, however WarpGrad methods are first-order equivalent to Riemannian Gradient Descent under a meta-learned metric (Appendix B, E).

Meta-learning over the joint task parameter distribution We consider a general meta-learning setting where we are given a task distribution $p(\tau)$. A task $\tau = (h^\tau, \mathcal{L}^\tau_{\text{meta}}, \mathcal{L}^\tau_{\text{task}})$ is defined by a task learner h^τ embedded with a shared WarpGrad optimiser, a meta-training objective $\mathcal{L}^\tau_{\text{meta}}$, and a task adaptation objective $\mathcal{L}^\tau_{\text{task}}$. We use $\mathcal{L}^\tau_{\text{task}}$ to adapt task parameters θ and $\mathcal{L}^\tau_{\text{meta}}$ to adapt warp parameters ϕ. Meta and task objectives can differ in arbitrary ways, but both are expectations over some data, i.e. $\mathcal{L}(\theta, \phi) = \mathbb{E}_{x,y}[\ell(h(x, \theta, \phi), y)]$. In the simplest case, they differ in terms of validation versus training data, but they may differ in terms of learning paradigm. For instance, we evaluate a WarpGrad optimiser in a continual learning experiment where $\mathcal{L}^\tau_{\text{task}}$ is the MSE on the current task, while $\mathcal{L}^\tau_{\text{meta}}$ is designed to prevent catastrophic forgetting across sequences of tasks (Section 3). WarpGrad is designed for integrating learning paradigms, a promising avenue for future research [28, 27].

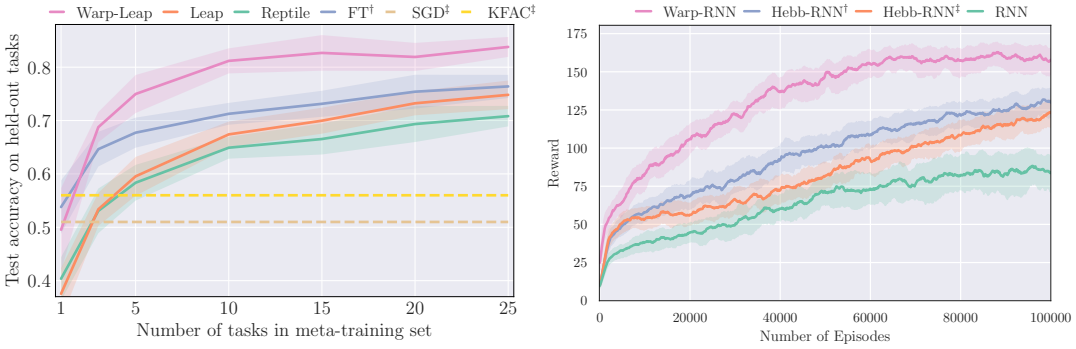


Figure 2: *Left:* Omniglot test accuracies on held-out tasks after meta-training on a varying number of tasks. Shading represents standard deviation across 10 independent runs. *Right:* On a RL maze navigation task, mean cumulative return is shown. Shading represents inter-quartile ranges across 10 independent runs.†Simple modulation and ‡retroactive modulation are used [30].

While MAML-based methods backpropagate through parameter trajectories, we treat them as Monte-Carlo samples from a conditional distribution $p(\theta \mid \tau)$ [10]. Together with the task distribution, these samples form an empirical distribution $p(\tau, \theta)$ that defines a joint search space across tasks. We meta-learn an optimiser that produces good parameter updates across this space. We exploit the gradient principle to define a meta-objective: a gradient update rule should point in the direction of steepest descent at each step of adaptation. Crucially, in WarpGrad this direction is *not* with respect to the objective that produced the gradient, $\mathcal{L}^\tau_{\text{task}}$, but with respect to the meta-objective $\mathcal{L}^\tau_{\text{meta}}$,

$$L(\phi) := \sum_{\tau \sim p(\tau)} \sum_{\theta^\tau \sim p(\theta|\tau)} \mathcal{L}^\tau_{\text{meta}} \left(\theta^\tau - \alpha \nabla \mathcal{L}^\tau_{\text{task}} \left(\theta^\tau; \phi \right); \phi \right). \tag{3}$$

Decoupling the task gradient operator $\nabla \mathcal{L}^\tau_{\text{task}}$ from the geometry encoded in ϕ lets us infuse global knowledge in the optimiser; for instance, in section 3 we meta-learn an optimiser against catastrophic forgetting. Uniquely among gradient-based meta-learners, the WarpGrad meta-objective is defined point-wise as an expectation over the joint task parameter distribution, allowing it scale beyond few-shot learning and generalise to arbitrary adaptation processes; it does not suffer from vanishing/exploding gradients nor the credit assignment problem. It does rely on second-order gradients, a requirement we can relax by detaching task parameter gradients ($\nabla \mathcal{L}^\tau_{\text{task}}$) in Eq. 3. This approximation only discards local second-order effects, which are typically dominated in long parameter trajectories [7]. The WarpGrad objective is defined in terms of warp parameters ϕ and takes θ as given. As such, we can integrate the WarpGrad objective with any meta-objective C defined over θ, in particular those that learn a "prior" over θ_0 (e.g. MAML or Leap [7]); see Appendix D.

To train a WarpGrad optimiser, we use stochastic gradient descent. We solve Eq. 8 by alternating between sampling task parameters given the current parameter values for ϕ and θ^τ_0 and taking meta-gradient steps over these samples, which can be done in a variety of ways (appendix C). In Algorithm 1, we illustrate a simple online version with constant memory and linear complexity in K. In Appendix C we detail a more complex offline training algorithm that uses a replay buffer for greater data efficiency; in our Omniglot experiment (Section 3), offline meta-training allows us to update warp parameters 2000 times with each meta-batch, in contrast to one update for other meta-learners, which improves final test accuracy from 76.3% to 84.3% (Appendix G).

Algorithm 1 WarpGrad: online meta-training

Require: $p(\tau)$: distribution over tasks
Require: α, β, λ: hyper-parameters
initialise ϕ and θ_0
while not done **do**
  Sample mini-batch of task $\mathcal{B}$ from $p(\tau)$
  $g_\phi, g_{\theta_0} \leftarrow 0$
  for all $\tau \in \mathcal{B}$ **do**
    $\theta^\tau_0 \leftarrow \theta_0$
    for all k in $0, \dots, K_\tau - 1$ **do**
      $\theta^\tau_{k+1} \leftarrow \theta^\tau_k - \alpha \nabla \mathcal{L}^\tau_{\text{task}} \left(\theta^\tau_k; \phi \right)$
      $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta^\tau_k)$
      $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta_0; \theta^\tau_{0:k})$
    end for
  end for
  $\phi \leftarrow \phi - \beta g_\phi$
  $\theta_0 \leftarrow \theta_0 - \lambda \beta g_{\theta_0}$
end while

3 Experiments and Conclusion

We evaluate WarpGrad methods in a set of experiments designed to answer three questions: (a) do WarpGrad methods retain the inductive bias of MAML-based few-shot learners? (b) Can WarpGrad methods scale to problems beyond the reach of such methods? (c) Can WarpGrad generalise to complex meta-learning problems? For WarpGrad methods, we insert warp-layers in a baseline architecture; see appendices for details.

(a) Few-Shot Learning We evaluate on the *mini*ImageNet [45, 39] and *tiered*ImageNet [41] datasets. The former is a random sub-sample of 100 classes from ILSVRC-12, each with 600 images; the latter stratifies 608 classes into 34 higher-level categories human-curated ImageNet hierarchy [4]. We use a Warp-MAML meta-learner ($J = L(\phi) + C^{\text{MAML}}(\theta_0)$) with linear warps for fair comparison. We report best results from our experiments or the literature in (Table 1).

(b) Multi-Shot Learning We WarpGrad on the Omniglot [19] protocol of [7], where each of the 50 alphabets is a 20-way classification task and adaptation involves 100 gradient steps, ruling out MAML-based approaches [7]. We train a Warp-Leap meta-learner ($J = L(\phi) + C^{\text{Leap}}(\theta_0)$) with an offline algorithm (Algorithm 2, Appendix C) that makes 2000 updates to warp parameters per meta mini-batch. Except for the case of one pretraining task, Warp-Leap outperforms all baselines (Figure 2); non-linear warps, which go beyond block-diagonal preconditioning, reach ~89% test accuracy (Appendix G). Finally, WarpGrad behave distinctly different from Natural Gradient Descent methods (Appendix H).

Table 1: Mean test accuracy after task adaptation on held out evaluation tasks. †Multi-headed. See Appendix F and I.

5-way 5-shot	***mini*ImageNet**
Reptile	66.0 ± 0.6
Meta-SGD	64.0 ± 0.9
CAVIA (512)	65.9 ± 0.6
MAML	63.2 ± 0.9
Warp-MAML	**68.4** ± 0.6
5-way 5-shot	***tiered*ImageNet**
MAML	70.3 ± 1.8
Warp-MAML	**74.1** ± 0.7
20-way 100-shot	**Omniglot**
Finetuning†	76.4 ± 2.2
Reptile	70.8 ± 1.9
Leap	75.5 ± 2.6
Warp-Leap	**83.6** ± 1.9

(c) General-Purpose Meta-Learning

(c.1) Reinforcement Learning To illustrate how WarpGrad may be used both with recurrent neural networks and in (meta-)meta-reinforcement learning, we evaluate it in a maze navigation task proposed by [29]. The environment is a fixed maze and a task is defined by randomly choosing a goal location. The agent's objective is to find the location as many times as possible, being teleported to a random location each time it finds it. The learner is advantage actor-critic with a basic recurrent neural network [46]. We design a Warp-RNN as a HyperNetwork [11] that uses a Warp-LSTM that is fixed during training. This amounts to meta-learning the optimiser for the meta-learner. The warp-LSTM modulates the weights of the learner (Appendix J), which in turn is trained on mini-batches of 30 episodes for 200 000 steps. Recurrence is crucial; linear warps (T-Nets) do worse than the baseline learner. We accumulate the gradient of fixed warp-parameters continually (Algorithm 3, Appendix C) at each task parameter update. Warp parameters are updated on every 30^{th} step on the learner's parameters (we control for meta-LSTM capacity in Appendix J). We compare against default Learning to Reinforcement Learn [46] and Hebbian meta-learning [29, 30]; see Appendix J for details. The Warp-RNN converges faster, more robustly, and to a higher level of performance (Figure 2).

(c.2) Continual Learning We test if a WarpGrad optimiser can prevent catastrophic forgetting [9]. To this end, we design a continual learning version of the sine regression meta-learning experiment [6] by splitting the input interval $[-5, 5] \subset \mathbb{R}$ into 5 consecutive sub-tasks [an alternative protocol was recently proposed independently of us by 15]. Each sub-task is a regression problem with the target being a mixture of two random sine waves; for each task, we train a 4-layer feed-forward task learner with interleaved warp layers incrementally on one sub-task at a time (see Appendix K for details). To isolate the behaviour of WarpGrad parameters, we use a fixed random initialisation for each task sequence. Warp parameters are meta-learned to prevent catastrophic forgetting by defining $\mathcal{L}^{\tau}_{\text{meta}}$ to be the average task loss over current and previous sub-tasks, for each sub-task in a task sequence. This forces warp-parameters to disentangle the adaptation process of current and previous sub-tasks. Evaluating our WarpGrad optimiser on held-out tasks (Appendix K), we find that it learns new sub-tasks while largely retaining performance on previous sub-tasks, providing an effective mechanism against catastrophic forgetting and promising avenue for further research.

Conclusion WarpGrad methods retain the inductive bias of MAML-based meta-learners while scaling to complex problems and architectures. WarpGrad provides a principled framework for general-purpose meta-learning that integrates arbitrary learning paradigms with novel forms of preconditioning, opening up for new perspective on gradient-based adaptation and optimisation.

References

[1] S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[2] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, 2016.

[3] A. Antoniou, H. Edwards, and A. J. Storkey. How to train your MAML. In *International Conference on Learning Representations*, 2019.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *International Conference on Computer Vision and Pattern Recognition*, 2009.

[5] G. Desjardins, K. Simonyan, R. Pascanu, and k. kavukcuoglu. Natural neural networks. In *Advances in Neural Information Processing Systems*, 2015.

[6] C. Finn, P. Abbeel, and S. Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*, 2017.

[7] S. Flennerhag, P. G. Moreno, N. D. Lawrence, and A. Damianou. Transferring knowledge across learning processes. In *International Conference on Learning Representations*, 2019.

[8] S. Flennerhag, H. Yin, J. Keane, and M. Elliot. Breaking the activation function bottleneck through adaptive parameterization. In *Advances in Neural Information Processing Systems*, 2018.

[9] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[10] E. Grant, C. Finn, S. Levine, T. Darrell, and T. L. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

[11] D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2016.

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

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

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

[16] T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. *arXiv preprint arXiv:1806.03836*, 2018.

[17] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[18] A. Lacoste, B. Oreshkin, W. Chung, T. Boquet, N. Rostamzadeh, and D. Krueger. Uncertainty in multitask transfer learning. In *Advances in Neural Information Processing Systems*, 2018.

[19] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2011.

[20] J. M. Lee. *Introduction to Smooth Manifolds*. Springer, 2003.

[21] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, 2019.

[22] S.-W. Lee, J.-H. Kim, J. Ha, and B.-T. Zhang. Overcoming Catastrophic Forgetting by Incremental Moment Matching. In *Advances in Neural Information Processing Systems*, 2017.

[23] Y. Lee and S. Choi. Meta-Learning with Adaptive Layerwise Metric and Subspace. In *International Conference on Machine Learning*, 2018.

[24] Z. Li and D. Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, 2016.

[25] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[26] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, 2015.

[27] R. Mendonca, A. Gupta, R. Kralev, P. Abbeel, S. Levine, and C. Finn. Guided meta-policy search. *arXiv preprint arXiv:1904.00956*, 2019.

[28] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. In *International Conference on Learning Representations*, 2019.

[29] T. Miconi, J. Clune, and K. O. Stanley. Differentiable plasticity: training plastic neural networks with backpropagation. In *International Conference on Machine Learning*, 2018.

[30] T. Miconi, J. Clune, and K. O. Stanley. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. In *International Conference on Learning Representations*, 2019.

[31] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations*, 2018.

[32] A. Mujika, F. Meier, and A. Steger. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2017.

[33] T. Munkhdalai, X. Yuan, S. Mehri, T. Wang, and A. Trischler. Learning rapid-temporal adaptations. In *International Conference on Machine Learning*, 2018.

[34] A. Nichol, J. Achiam, and J. Schulman. On First-Order Meta-Learning Algorithms. *arXiv preprint ArXiv:1803.02999*, 2018.

[35] B. N. Oreshkin, A. Lacoste, and P. Rodriguez. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, 2018.

[36] E. Park and J. B. Oliva. Meta-curvature. *arXiv preprint arXiv:1902.03356*, 2019.

[37] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Association for the Advancement of Artificial Intelligence*, 2018.

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

[39] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2016.

[40] S. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, 2017.

[41] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel. Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*, 2018.

[42] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

[43] J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[44] J. Suarez. Language modeling with recurrent highway hypernetworks. In *Advances in Neural Information Processing Systems*, 2017.

[45] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems*, 2016.

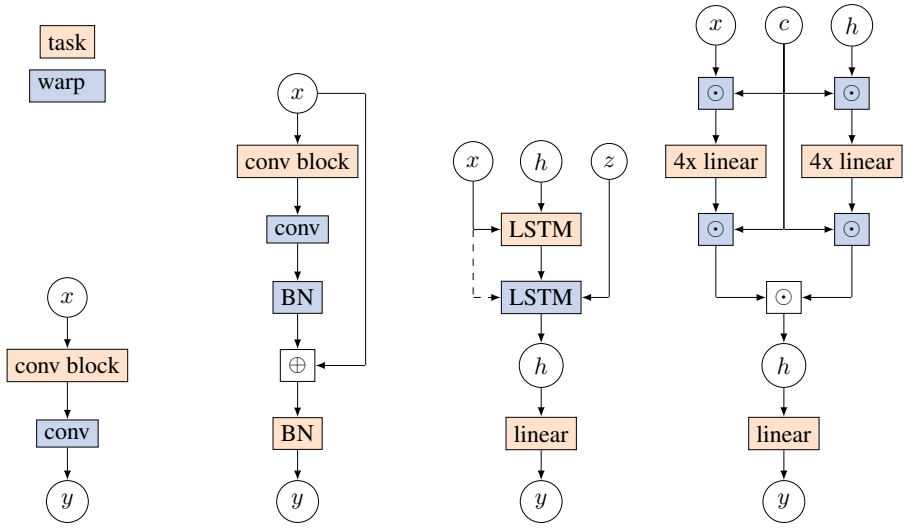
[46] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. In *Annual Meeting of the Cognitive Science Society*, 2016.

[47] Y. Wu, M. Ren, R. Liao, and R. B. Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

[48] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast Context Adaptation via Meta-Learning. *International Conference on Machine Learning*, 2019.

Appendix

A WarpGrad Design Principles for Neural Nets



(a) Warp-ConvNet (b) Warp-ResNet (c) Warp-LSTM (d) Warp-HyperNetwork

Figure 3: Illustration of possible WarpGrad architectures. Orange represents task layers and blue represents warp layers. $\oplus$ denotes residual connections and $\odot$ any form of gating mechanism. We can obtain warped architectures by interleaving task and warp layers (a, c) or by designating some layers in standard architectures as task-adaptable and some as warp layers (b, d).

WarpGrad is a general family of model-embedded meta-learned optimisers. This offers a great deal of flexibility in approaching a meta-learning problem. To provide some guidance in these design choices, we briefly discuss general principles we found useful. To embed warp layers given a task-learner architecture, we may either insert new warp layers in the given architecture or designate some layers as warp layers and some as task layers. We found that WarpGrad can both be used in a high-capacity mode where task learners are relatively weak to avoid overfitting, as well as in a low-capacity mode where task learners are powerful and warp layers are relatively weak. The best approach depends on the problem at hand. We highlight three approaches to designing WarpGrad optimisers:

(a) **Model partitioning**. Given a desired architecture, designate some operations as task-adaptable and the rest as warp layers. Task layers do not have to interleave exactly with warp layers as gradient warping arises both through the forward pass and through backpropagation. This was how we approached the *tiered*ImageNet and *mini*ImageNet experiments.

(b) **Model augmentation**. Given a model, designate all layers as task-adaptable and interleave warp layers. Warp layers can be relatively weak as backpropagation through non-linear activations ensures expressive gradient warping. This was our approach to the Omniglot experiment; our main architecture interleaves linear warp layers in a standard architecture.

(c) **Information compression**. Given a model, designate all layers as warp and interleave weak task layers. In this scenario, task learners are prone to overfitting. Pushing capacity into the warp allows it to encode general information the task learner can draw on during task adaptation. This approach is similar to approaches in transfer and meta-learning that restrict the number of free parameters during task training [40, 23, 48].

Note that in either case, once warp layers have been chosen, standard backpropagation automatically warps gradients for us. Thus, WarpGrad is fully compatible with any architecture, for instance, Residual Neural Networks [12] or LSTMs. For convolutional neural networks, we may use any form of convolution, learned normalization [e.g. 14], or adaptor module [e.g. 40, 37] to designed task and warp layers. For recurrent networks, we can use stacked LSTMs to interleave warped layers, as well as any type of HyperNetwork architecture [e.g. 11, 44, 8] or partitioning of fast and slow weights [e.g. 32]. Figure 3 illustrates this process.

B The Geometry of Warped Gradient Descent

The generality of WarpGrad methods means that we are not guaranteed to obtain a well-behaved preconditioning matrix. Further, while characterising WarpGrad methods as embedding preconditioning via gradient warping provides us with a framework for designing WarpGrad optimisers, it offers little guidance as to how we may meta-learn warp parameters. To address these points, we take a geometric perspective and reinstate a valid Riemann metric, from which we derive a framework for meta-learning warp parameters on first principles.

The inner update rule in Eq. 1 represents first-order Riemannian Gradient Descent on task parameters if $G := P^{-1}$ is a valid Riemann metric [1], which enjoys similar convergence guarantees to stochastic gradient descent. Gradient warping is thus well-behaved if it represents a valid (meta-learned) Riemann metric. Informally, a metric tensor G is a positive-definite, smoothly varying matrix that measures the curvature on a manifold $\mathcal{W}$. The metric tensor defines the steepest direction of descent by $-G^{-1}\nabla\mathcal{L}$ [20]. To show that a WarpGrad optimiser represents a metric tensor, we define an explicit warp Ω by reparameterising warp layers such that $\omega^i(h^i(x;\theta^i)) = h^i(x;\Omega(\theta)^i)\ \forall x, i$. Let Ω be a map from a Euclidean representation space $\mathcal{P}$ onto a Riemannian space $\mathcal{W}$ with $\gamma = \Omega(\theta;\phi)$. The metric tensor G can then be identified via push-forward:

$$\Delta\theta := \nabla\left(\mathcal{L}\circ\Omega\right)(\theta;\phi) = \left[D\Omega(\theta;\phi)\right]^T\nabla\mathcal{L}\left(\gamma\right) \tag{4}$$

$$\Delta\gamma := D\Omega(\theta;\phi)\,\Delta\theta = G(\gamma;\phi)^{-1}\nabla\mathcal{L}(\gamma), \tag{5}$$

where $G^{-1} := [D\Omega][D\Omega]^T$. Provided Ω is not degenerate (G is non-singular), G^{-1} is positive-definite, hence a valid Riemann metric. Because WarpGrad optimisers act in $\mathcal{P}$, they represent this metric implicitly. Task parameter gradients under non-linear warps are therefore not exactly equivalent to preconditioning under $P = G^{-1}$, but from first-order Taylor series expansion we have that

$$(\mathcal{L}\circ\Omega)(\theta - \alpha\Delta\theta) = \mathcal{L}(\gamma - \alpha\Delta\gamma) + \mathcal{O}(\alpha^2). \tag{6}$$

Consequently, gradient warping can be understood as warping the native loss surface into a surface conducive to task adaptation. Warp parameters ϕ control this geometry and therefore *what* task adaptation converges to. By meta-learning ϕ we can accumulate information that is conducive to task adaptation but that may not be available during that process. Because task adaptation relies on stochastic gradient descent, the meta-learned geometry should yield as effective updates as possible across expected task parameterisations, which implies a canonical meta-objective of the form

$$\min_{\phi}\ \mathbb{E}_{\mathcal{L},\gamma\sim p(\mathcal{L},\gamma)}\left[\mathcal{L}\left(\gamma - \alpha\, G(\gamma;\phi)^{-1}\nabla\mathcal{L}(\gamma)\right)\right]. \tag{7}$$

In contrast to MAML-based approaches (Eq. 1), this objective avoids backpropagation through the adaptation process. Instead, it defines task adaptation abstractly by introducing a joint distribution over objectives and parameterisations, opening up for general-purpose meta-learning at scale.

C WarpGrad Meta-Training Algorithms

In this section, we provide variants of WarpGrad training algorithms used in this paper. Algorithm 1 describes a simple online algorithm, which accumulates meta-gradients online during task adaptation. This algorithm has constant memory and scales linearly in the length of task trajectories. In Algorithm 2, we describe an offline meta-training algorithm. This algorithm is similar to Algorithm 1 in many respects, but differs in that we do not compute meta-gradients online during task adaptation. Instead, we accumulate them into a replay buffer of sampled task parameterisations. This buffer is a Monte-Carlo sample of the expectation in the meta objective (Eq. 8) that can be thought of as a dataset in its own right. Hence, we can apply standard mini-batching with respect to the buffer and perform mini-batch gradient descent on warp parameters. This allows us to update warp parameters several times for a given sample of task parameter trajectories, which can greatly improve data efficiency. In our Omniglot experiment, we found offline meta-training to converge faster: in fact, a mini-batch size of 1 (i.e. $\eta = 1$ in Algorithm 2 converges rapidly without any instability.

Finally, in Algorithm 3, we present a continual meta-training process where meta-training occurs throughout a stream of learning experiences. Here, C represents a multi-task objective, such as the average task loss, $C^{\text{multi}} = \sum_{\tau \sim p(\tau)} \mathcal{L}^{\tau}_{\text{task}}$. Meta-learning arise by collecting experiences continuously (across different tasks) and using these to accumulate the meta-gradient online. Warp parameters are updated intermittently with the accumulated meta-gradient. We use this algorithm in our maze navigation experiment, where task adaptation is internalised within the RNN task learner.

Algorithm 1 WarpGrad: online meta-training

Require: $p(\tau)$: distribution over tasks
Require: $\alpha, \beta,\ \lambda$: hyper-parameters
initialise ϕ and θ_0
while not done **do**
  Sample mini-batch of task $\mathcal{B}$ from $p(\tau)$
  $g_\phi, g_{\theta_0} \leftarrow 0$
  for all $\tau \in \mathcal{B}$ **do**
    $\theta^\tau_0 \leftarrow \theta_0$
    for all k in $0, \dots, K_\tau - 1$ **do**
      $\theta^\tau_{k+1} \leftarrow \theta^\tau_k - \alpha \nabla \mathcal{L}^\tau_{\text{task}}\left(\theta^\tau_k; \phi\right)$
      $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta^\tau_k)$
      $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta^\tau_0; \theta^\tau_{0:k})$
    end for
  end for
  $\phi \leftarrow \phi - \beta g_\phi$
  $\theta_0 \leftarrow \theta_0 - \lambda \beta g_{\theta_0}$
end while

Algorithm 2 WarpGrad: offline meta-training

Require: $p(\tau)$: distribution over tasks
Require: $\alpha, \beta,\ \lambda,\ \eta$: hyper-parameters
initialise ϕ and θ_0
while not done **do**
  Sample mini-batch of task $\mathcal{B}$ from $p(\tau)$
  $\mathcal{T} \leftarrow \{\tau : [\theta_0] \text{ for } \tau \text{ in } \mathcal{B}\}$
  for all $\tau \in \mathcal{B}$ **do**
    $\theta^\tau_0 \leftarrow \theta_0$
    for all k in $0, \dots, K_\tau - 1$ **do**
      $\theta^\tau_{k+1} \leftarrow \theta^\tau_k - \alpha \nabla \mathcal{L}^\tau_{\text{task}}\left(\theta^\tau_k; \phi\right)$
      $\mathcal{T}[\tau]$.append(θ^τ_{k+1})
    end for
  end for
  $i, g_\phi, g_{\theta_0} \leftarrow 0$
  while $\mathcal{T}$ not empty **do**
    sample τ, k without replacement
    $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta^\tau_k)$
    $g_{\theta_0} \leftarrow g_{\theta_0} + \nabla C(\theta^\tau_0; \theta^\tau_{0:k})$
    $i \leftarrow i + 1$
    if $i = \eta$ **then**
      $\phi \leftarrow \phi - \beta g_\phi$
      $\theta_0 \leftarrow \phi - \lambda \beta g_{\theta_0}$
      $i, g_\phi, g_{\theta_0} \leftarrow 0$
    end if
  end while
end while

Algorithm 3 WarpGrad: continual meta-training

Require: $p(\tau)$: distribution over tasks
Require: $\alpha, \beta,\ \lambda, \eta$: hyper-parameters
initialise ϕ and θ
$i, g_\phi, g_\theta \leftarrow 0$
while not done **do**
  Sample mini-batch of task $\mathcal{B}$ from $p(\tau)$
  for all $\tau \in \mathcal{B}$ **do**
    $g_\phi \leftarrow g_\phi + \nabla L(\phi; \theta)$
    $g_\theta \leftarrow g_\theta + \nabla C(\theta; \phi)$
  end for
  $\theta \leftarrow \theta - \lambda \beta g_\theta$
  $g_\theta, i \leftarrow 0, i + 1$
  if $i = \eta$ **then**
    $\phi \leftarrow \phi - \beta g_\phi$
    $i, g_\theta \leftarrow 0$
  end if
end while

D WarpGrad Optimisers

WarpGrad methods are compatible with any learner over θ; for instance, (a) *Multi-task solution*: in online learning, we can alternate between updating a multi-task solution and tuning warp parameters. We use this approach in our Reinforcement Learning experiment (Section 3); (b) *Meta-learned point-estimate*: when task adaptation occurs in batch mode, we can meta-learn a shared initialisation θ_0. Our few-shot and supervised learning experiments take this approach Section 3; (c) *Meta-learned prior*: we can use meta-learners that define a full prior [42, 35, 18, 16], enabling Bayesian approaches to WarpGrad. The family of WarpGrad methods is fully described by combining our meta-objective L (or $\hat{L}$) with a meta-objective C over θ_0,

$$J(\phi, \theta_0) \coloneqq L\left(\phi\right) + \lambda\, C\left(\theta_0\right), \tag{8}$$

where $\lambda \in [0, \infty)$ is a hyper-parameter. To make the objective in Eq. 8 tangible, we illustrate two WarpGrad optimisers used in our experiment (for detailed definitions of all WarpGrad optimisers we use, see Appendix D). We define a *Warp-MAML* that we use for few-shot learning by choosing C to be the MAML objective (Eq. 1). For experiments that MAML cannot scale to, we define *Warp-Leap* by choosing the Leap objective [7] for C. Leap is an attractive complement as it has linear complexity in task adaptation length as well as constant memory usage. It is defined by minimising the average distance between θ_0 and θ_K^τ, implemented by fixing θ_k^τ and pulling θ_{k-1}^τ forward towards it under the Euclidean norm in $\mathcal{P}$. By virtue of push-forward, this corresponds to the Riemannian norm on $\mathcal{W}$. Hence, Warp-Leap is a greedy search for geodesics in the meta-learned geometry defined by

$$C^{\text{Leap}}(\theta_0) := \sum_{\tau \sim p(\tau)} \sum_{k=1}^{K_\tau} \left\| \text{sg}\left[\vartheta_k^\tau\right] - \vartheta_{k-1}^\tau \right\|_2, \quad \vartheta_k^\tau = (\theta_{k,0}^\tau, \dots, \theta_{k,n}^\tau, \mathcal{L}_{\text{task}}^\tau\left(\theta_k^\tau; \phi\right)). \tag{9}$$

Next, we describe the full set of algorithms used in our experiments.

Warp-MAML We use this algorithm for few-shot learning (Section 3). We use the full warp-objective in Eq. 3 together with the MAML objective (Eq. 1),

$$J^{\text{Warp-MAML}} := L(\phi) + \lambda C^{\text{MAML}}(\theta_0), \tag{10}$$

where $C^{\text{MAML}} = L^{\text{MAML}}$ under the constraint $P = I$. In our experiments, we trained Warp-MAML using the online training algorithm (Algorithm 1).

Warp-Leap We use this algorithm for multi-shot meta-learning. It is defined by applying Leap to θ_0 (Eq. 9),

$$J^{\text{Warp-Leap}} := L(\phi) + \lambda C^{\text{Leap}}(\theta_0). \tag{11}$$

Note that the Leap meta-gradient makes a first-order approximation to avoid backpropagating through the adaptation process. It is given by

$$\nabla C^{\text{Leap}}(\theta_0) \approx -\sum_{\tau \sim p(\tau)} \sum_{k=1}^{K_\tau} \frac{\Delta \mathcal{L}_{\text{task}}^\tau\left(\theta_k^\tau; \phi\right) \nabla \mathcal{L}_{\text{task}}^\tau\left(\theta_{k-1}^\tau; \phi\right) + \Delta \theta_k^\tau}{\left\| \vartheta_k^\tau - \vartheta_{k-1}^\tau \right\|_2}, \tag{12}$$

where $\Delta \mathcal{L}_{\text{task}}^\tau\left(\theta_k^\tau; \phi\right) := \mathcal{L}_{\text{task}}^\tau\left(\theta_k^\tau; \phi\right) - \mathcal{L}_{\text{task}}^\tau\left(\theta_{k-1}^\tau; \phi\right)$ and $\Delta \theta_k^\tau := \theta_k^\tau - \theta_{k-1}^\tau$. In our experiments, we train Warp-Leap using Algorithm 1 in the multi-shot *tiered*ImageNet experiment and Algorithm 2 in the Omniglot experiment. We perform an ablation study for training algorithms, exact (Eq. 3) versus approximate (detaching inner gradient) meta-objective, and warp architecture on Omniglot in Appendix G.

Warp-RNN For our Reinforcement Learning experiment, we define a WarpGrad optimiser by meta-learning an LSTM that modulates the weights of the task learner (see Appendix J for details). For this algorithm, we face a continuous stream of experiences (episodes) that we meta-learn over using our continual meta-training algorithm (Algorithm 3). In our experiment, both $\mathcal{L}_{\text{task}}^\tau$ and $\mathcal{L}_{\text{meta}}^\tau$ are the advantage actor-critic objective [46]; C is computed on one batch of 30 episodes, whereas L is accumulated over $\eta = 30$ such batches, for a total of 900 episodes. As each episode involves 300 steps in the environment, we cannot apply the exact meta objective, but use the approximate meta objective that detaches the inner task gradient. Specifically, let $E^\tau = \{s_0, a_1, r_1, s_1, \dots, s_T, a_t, r_T, s_{T+1}\}$ denote an episode on task τ, where s denotes state, a action, and r instantaneous reward. Denote a mini-batch of randomly sampled task episodes by $\mathbf{E} = \{E^\tau\}_{\tau \sim p(\tau)}$ and an ordered set of K consecutive mini-batches by $\mathcal{E}^k = \{\mathbf{E}_{k-i}\}_{i=0}^{K-1}$. Then $\hat{L}(\phi; \mathcal{E}^k) = 1/n \sum_{\mathbf{E}_i \in \mathcal{E}^k} \sum_{E_{i,j}^\tau \in \mathbf{E}_i} \mathcal{L}_{\text{meta}}^\tau(\phi; \theta, E_{i,j}^\tau)$ and $C^{\text{multi}}(\theta; \mathbf{E}_k) = 1/n' \sum_{E_{k,j}^\tau \in \mathbf{E}_k} \mathcal{L}_{\text{task}}^\tau(\theta; \phi, E_{k,j}^\tau)$, where n and n' are normalising constants. The Warp-RNN objective is defined by

$$J^{\text{Warp-RNN}} := \begin{cases} L(\phi; \mathcal{E}^k) + \lambda C^{\text{multi}}(\theta; \mathbf{E}_k) & \text{if} \quad k = \eta \\ \lambda C^{\text{multi}}(\theta; \mathbf{E}_k) & \text{otherwise.} \end{cases} \tag{13}$$

WarpGrad for Continual Learning For this experiment, we focus on meta-learning warp-parameters. Hence, the initialisation for each task sequence is a fixed random initialisation, (i.e. $\lambda C(\theta^0) = 0$). For the warp meta-objective, we take expectations over N task sequences, where each task sequence is a sequence of $T = 5$ sub-tasks that the task-learner observes one at a time; thus while the task loss is defined over the current sub-task, the meta-loss averages of the current and all prior sub-tasks, for each sub-task in the sequence. See Appendix K for detailed definitions. Importantly, because WarpGrad defines task adaptation abstractly by a probability distribution, we can readily implement a continual learning objective by modifying the joint task parameter distribution $p(\tau, \theta)$ that we use in the meta-objective (Eq. 3). A task defines a sequence of sub-tasks over which we generate parameter trajectories $\boldsymbol{\theta}^\tau$. Thus, the only difference from multi-task meta-learning is that parameter trajectories are not generated under a fixed task, but arise as a function of the continual learning algorithm used for adaptation. We define the conditional distribution $p(\theta \mid \tau)$ as before by sampling sub-task parameters θ^{τ_t} from a mini-batch of such task trajectories, keeping track of which sub-task t it belongs to and which sub-tasks came before it in the given task sequence τ. The meta-objective is constructed, for any sub-task parameterisation θ^{τ_t}, as $\mathcal{L}^\tau_{\text{meta}}(\theta^{\tau_t}) = 1/t \sum_{i=1}^t \mathcal{L}^\tau_{\text{task}}(\theta^{\tau_i}, \mathcal{D}_i; \phi)$, where $\mathcal{D}_j$ is data from sub-task j (Appendix K). The meta-objective is an expectation over task parameterisations,

$$L^{\text{CL}}(\phi) \coloneqq \sum_{\tau \sim p(\tau)} \sum_{t=1}^{T} \sum_{\theta^{\tau_t} \sim p(\theta \mid \tau_t)} \mathcal{L}^\tau_{\text{meta}}\left(\theta^{\tau_t}; \phi\right). \tag{14}$$

E Synthetic Experiment

To build intuition for what it means to warp space, we construct a simple 2-D problem over loss surfaces. A learner is faced with the task of minimising an objective function of the form $f^\tau(x_1, x_2) = g^\tau_1(x_1)\exp(g^\tau_2(x_2)) - g^\tau_3(x_1)\exp(g^\tau_4(x_1, x_2)) - g^\tau_5\exp(g^\tau_6(x_1))$,, where each task f^τ is defined by scale and rotation functions g^τ that are randomly sampled from a predefined distribution. Specifically, each task is defined by the objective function

$$\begin{aligned} f^\tau(x_1, x_2) = b^\tau_1(a^\tau_1 - x_1)^2 \exp(-x_1^2 - (x_2 + a^\tau_2)^2) \\ - b^\tau_2(x_1/s^\tau - x_1^3 - x_2^5)\exp(-x_1^2 - x_2^2) \\ - b^\tau_3 \exp(-(x_1 + a^\tau_3)^2 - x_1^2)), \end{aligned}$$

where each a, b and s are randomly sampled parameters from

$$\begin{aligned} s^\tau &\sim \text{Cat}(1, \dots, 10) \\ a^\tau_i &\sim \text{Cat}(-1, 0, 1) \\ b^\tau_i &\sim \text{Cat}(-5, \dots, 5). \end{aligned}$$

The task is to minimise the given objective from a randomly sampled initialisation, $x_{\{i=1,2\}} \sim U(-3, 3)$. During meta-training, we train on a task for 100 steps using a learning rate of 0.1. Each task has a unique loss-surface that the learner traverses from the randomly sampled initialisation. While each loss-surface is unique, they share an underlying structure. Thus, by meta-learning a warp over trajectories on randomly sampled loss surfaces, we expect WarpGrad to learn a warp that is close to invariant to spurious descent directions. In particular, WarpGrad should produce a smooth warped space that is quasi-convex for any given task to ensure that the task learner finds a minimum as fast as possible regardless of initialisation.

To visualise the geometry, we use an explicit warp Ω defined by a 2-layer feed-forward network with a hidden-state size of 30 and $\tanh$ non-linearities. We train warp parameters for 100 meta-training steps; in each meta-step we sample a new task surface and a mini-batch of 10 random initialisations that we train separately. We train to convergence and accumulate the warp meta-gradient online (Algorithm 1). We evaluate against gradient descent in Figure 4 on randomly sampled loss surfaces. Both optimisers start from the same initialisation, chosen such that standard gradient descent struggles; we expect that the WarpGrad optimisers has learned a geometry that is robust to the initialisation (top row in Figure 4). This is indeed what we find; the geometry learned by WarpGrad smoothly warps the native loss surface into a well-behaved space where gradient descent converges to a local minimum.

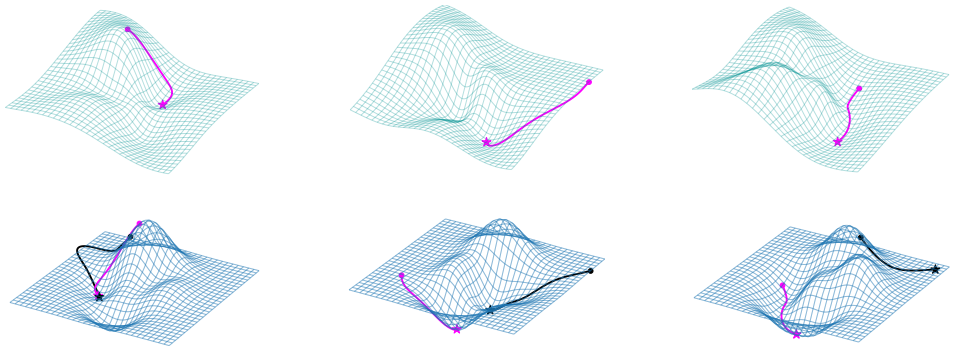


Figure 4: Example trajectories on three task loss surfaces. We start Gradient Descent (black) and WarpGrad (magenta) from the same initialisation; while SGD struggles with the curvature, the WarpGrad optimiser has learned a warp such that gradient descent in the representation space (top) leads to rapid convergence in model parameter space (bottom).

F Omniglot

We follow the protocol of Flennerhag et. al. [7], including choice of hyper-parameters. In this setup, each of the 50 alphabets that comprise the dataset constitutes a distinct task. Each task is treated as a 20-way classification problem. Four alphabets have fewer than 20 characters in the alphabet and are discarded, leaving us with 46 alphabets in total. 10 alphabets are held-out for final meta-testing; which alphabets are held out depend on the seed to account for variations across alphabets; we train an evaluate all baselines on 10 seeds. For each character in an alphabet, there are 20 raw samples. Of these, 5 are held out for final evaluation on the task while the remainder is used to construct a training set. Raw samples are pre-processed by random affine transformations in the form of (a) scaling between $[0.8, 1.2]$, (b) rotation $[0, 360)$, and (c) cropping height and width by a factor of $[-0.2, 0.2]$ in each dimension. This ensures tasks are too hard for few-shot learning. During task adaptation, mini-batches are sampled at random without ensuring class-balance (in contrast to few-shot classification protocols [45]). Note that benchmarks under this protocol are not compatible with few-shot learning benchmarks.

We use the same convolutional neural network architecture and hyper-parameters as in [7]. This learner stacks a convolutional block comprised of a 3×3 convolution with 64 filters, followed by 2×2 max-pooling, batch-normalisation, and ReLU activation, four times. All images are down-sampled to 28×28, resulting in a $1 \times 1 \times 64$ feature map that is passed on to a final linear layer. We create a Warp Leap meta-learner that inserts warp layers between each convolutional block, $W \circ \omega^4 \circ h^4 \circ \cdots \circ \omega^1 \circ h^1$, where each h is defined as above. In our main experiment, each ω^i is simply a 3×3 convolutional layer with zero padding; in Appendix G we consider both simpler and more sophisticated versions. We find that relatively simple warp layers do quite well. However, adding capacity does improve generalisation performance. We meta-learn the initialisation of task parameters using the Leap objective (Eq. 9), detailed in Appendix D.

Both $\mathcal{L}^\tau_{\text{meta}}$ and $\mathcal{L}^\tau_{\text{task}}$ are defined as the negative log-likelihood loss; importantly, we evaluate them on *different* batches of task data to ensure warp layers encourage generalisation. We found no additional benefit in this experiment from using hold-out data to evaluate $\mathcal{L}^\tau_{\text{meta}}$. We use the offline meta-training algorithm (Appendix C, Algorithm 2); in particular, during meta-training, we sample mini-batches of 20 tasks and train task learners for 100 steps to collect 2000 task parameterisations into a replay buffer. Task learners share a common initialisation and warp parameters that are held fixed during task adaptation. Once collected, we iterate over the buffer by randomly sampling mini-batches of task parameterisations without replacement. Unless otherwise noted, we used a batch size of $\eta = 1$. For each mini-batch, we update ϕ by applying gradient descent under the canonical meta-objective (Eq. 3), where we evaluate $\mathcal{L}^\tau_{\text{meta}}$ on a randomly sampled mini-batch of data from the corresponding task. Consequently, for each meta-batch, we take (up to) 2000 meta-gradient steps on warp parameters ϕ. We find that this form of mini-batching causes the meta-training loop to converge much faster and induces no discernible instability.

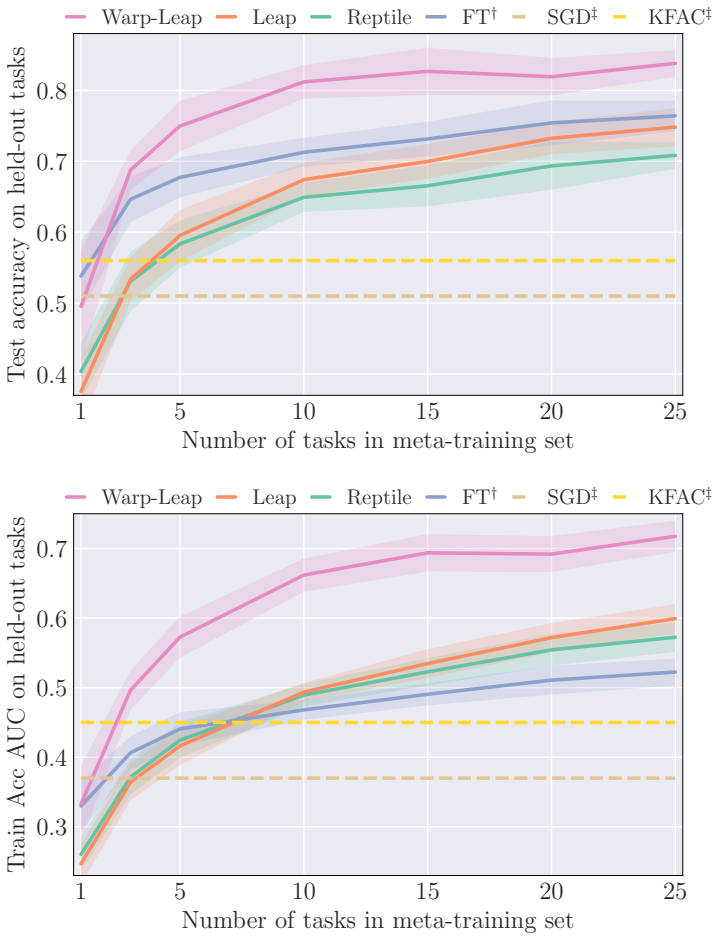


Figure 5: Omniglot results. *Top:* test accuracies on held-out tasks after meta-training on a varying number of tasks. *Bottom:* AUC under accuracy curve on held-out tasks after meta-training on a varying number of tasks. Shading represents standard deviation across 10 independent runs. We compare between Warp-Leap, Leap, and Reptile, multi-headed finetuning, as well as SGD and KFAC which used random initialisation but with 10x larger batch size and learning rate.

We compare Warp-Leap against no meta-learning with standard gradient descent (SGD) or KFAC [26]. We also benchmark against baselines provided in [7]; Leap, Reptile [34], MAML, and multi-headed fine-tuning. All learners benefit substantially from large batch sizes as this enables higher learning rates. To render no-pretraining a competitive option within a fair computational budget, we allow SGD and KFAC to use 10x larger batch sizes, enabling 10x larger learning rates. This renders them computationally costly, taking 2x and 4x longer to train on a given task during meta-test time than Warp-Leap, respectively.

G Ablation study: Warp Layers, Meta-Objective, and Meta-Training

WarpGrad provides a principled approach for model-informed meta-learning. It offers several degrees of freedom and to evaluate these design choices, we conduct an ablation study where we vary the design of warp layers as well as meta-training approach. For the ablation study, we fixed the number of pretraining tasks to 25 and report final test accuracy over 4 independent runs. All ablations use the same hyper-parameters, except for online meta-training which uses a learning rate of 0.001.

First, we vary the meta-training protocol by (a) using the approximate objective that detaches the inner task gradient, (b) using online meta-training (Algorithm 1), and (c) whether meta-learning the learning rate used for task adaptation is beneficial in this experiment. We meta-learn a single

Table 2: Mean test error after 100 training steps on held out evaluation tasks. †Multi-headed. ‡No meta-training, but 10x larger batch sizes (allows 10x larger learning rates).

Method No. Meta-training tasks	WarpGrad	Leap	Reptile	Finetuning†	MAML	KFAC‡	SGD‡
1	49.5 ± 7.8	37.6 ± 4.8	40.4 ± 4.0	53.8 ± 5.0	40.0 ± 2.6	**56.0**	51.0
3	**68.8** ± 2.8	53.4 ± 3.1	53.1 ± 4.2	64.6 ± 3.3	48.6 ± 2.5	56.0	51.0
5	**75.0** ± 3.6	59.5 ± 3.7	58.3 ± 3.3	67.7 ± 2.8	51.6 ± 3.8	56.0	51.0
10	**81.2** ± 2.4	67.4 ± 2.4	65.0 ± 2.1	71.3 ± 2.0	54.1 ± 2.8	56.0	51.0
15	**82.7** ± 3.3	70.0 ± 2.4	66.6 ± 2.9	73.5 ± 2.4	54.8 ± 3.4	56.0	51.0
20	**82.0** ± 2.6	73.3 ± 2.3	69.4 ± 3.4	75.4 ± 3.2	56.6 ± 2.0	56.0	51.0
25	**83.8** ± 1.9	74.8 ± 2.7	70.8 ± 1.9	76.4 ± 2.2	56.7 ± 2.1	56.0	51.0

scalar learning rate (as warp parameters can learn layer-wise scaling) along with a single scalar for momentum. Meta-gradients for the learning rate and momentum rate are clipped at 0.001 and we use a learning rate of 0.001. Note that when using offline meta-training, we store both task parameterisations and the momentum buffer at that point; when computing the canonical objective (Eq. 3) we use these in the update rule.

Further, we vary the architecture used for warp layers. We study simpler versions that use channel-wise scaling and more complex versions that use non-linearities and residual connections. We also evaluate a version where each warp-layer has two stacked convolutions, where the first warp convolution outputs 128 filters and the second warp convolution outputs 64 filters. Finally, in the two-layer warp-architecture, we evaluate a version that inserts a FiLM layer between the two warp convolutions. These are adapted during task training from a 0 initialisation; they amount to task embeddings that condition gradient warping on task statistics.

Table 3: Ablation study: mean test error after 100 training steps on held out evaluation tasks. Mean and standard deviation over 4 independent runs. *Offline* refers to offline meta-training (Appendix C), *online* to online meta-training Algorithm 1; *full* denotes Eq. 3 and *approx* denotes the first-order approximation;†Batch Normalization [14]; ‡equivalent to FiLM layers [37];§Residual connection [12], when combined with BN, similar to the Residual Adaptor architecture [40]; ¶FiLM task embeddings.

Architecture	Meta-training	Meta-objective	Accuracy
None (Leap)	Online	None	74.8 ± 2.7
3×3 conv (default)	Offline	full (L, Eq. 3)	84.4 ± 1.7
3×3 conv	Offline	approx	83.1 ± 2.7
3×3 conv	Online	full	76.3 ± 2.1
3×3 conv	Offline	full, learned α	83.1 ± 3.3
Scaling‡	Offline	full	77.5 ± 1.8
1×1 conv	Offline	full	79.4 ± 2.2
3×3 conv + ReLU	Offline	full	83.4 ± 1.6
3×3 conv + BN†	Offline	full	84.7 ± 1.7
3×3 conv + BN† + ReLU	Offline	full	85.0 ± 0.9
3×3 conv + BN† + Res§ + ReLU	Offline	full	86.3 ± 1.1
2-layer 3×3 conv + BN† + Res§	Offline	full	**88.0** ± 1.0
2-layer 3×3 conv + BN† + Res§ + TA¶	Offline	full	**88.1** ± 1.0

H Ablation study: WarpGrad and Natural Gradient Descent

Here, we perform ablation studies to compare the geometry that a WarpGrad optimiser learns to the geometry that Natural Gradient Descent (NGD) methods represent (approximately). For consistency, we run the ablation on Omniglot. As computing the true Fisher Information Matrix is intractable, we can compare WarpGrad against two common block-diagonal approximations, KFAC [26] and Natural Neural Nets [5].

First, we isolate the effect of warping task loss surfaces by fixing a random initialisation and only meta-learning warp parameters. That is, in this experiment, we set $\lambda C(\theta^0) = 0$. We compare

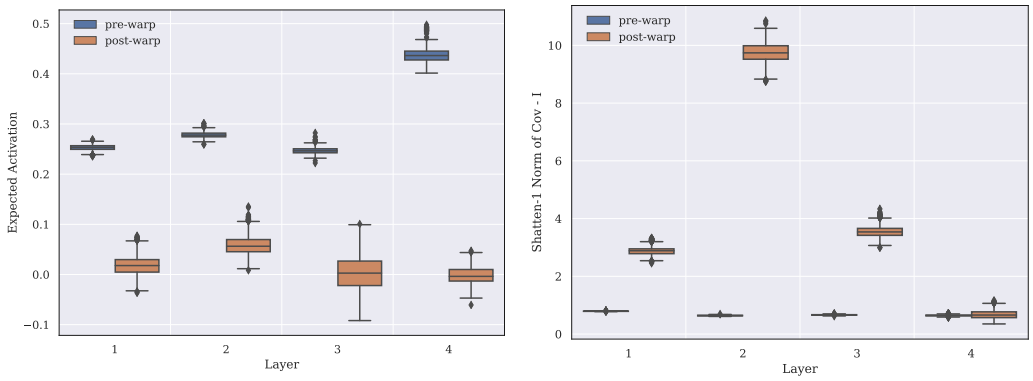


Figure 6: Ablation study. *Left:* Comparison of mean activation value $\mathbb{E}[h(x)]$ across layers, pre- and post-warping. *Right:* Shatten-1 norm of $\mathrm{Cov}(h(x), h(x)) - I$, pre- and post-norm. Statistics are gathered on held-out test set and averaged over tasks and adaptation steps.

against two baselines, stochastic gradient descent (SGD) and KFAC, both trained from a random initialisation. We use task mini-batch sizes of 200 and task learning rates of 1.0, otherwise we use the same hyper-parameters as in the main experiment. For WarpGrad, we meta-train with these hyper-parameters as well. We evaluate two WarpGrad architectures, in one, we use linear warp-layers, which gives a block-diagonal preconditioning, as in KFAC. In the other, we use our most expressive warp configuration from the ablation experiment in appendix G, where warp-layers are two-layer convolutional block with residual connections, batch normalisation, and ReLU activation. We find that warped geometries facilitate task adaptation on held-out tasks to a greater degree than either SGD or KFAC by a significant margin (table 4). We further find that going beyond block-diagonal preconditioning yields a significant improvement in performance.

Table 4: Ablation study: mean test error after 100 training steps on held out evaluation tasks from a random initialisation. Mean and standard deviation over 4 seeds.

Method	Preconditioning	Accuracy
SGD	None	40.1 ± 6.1
KFAC (NGD)	Linear (block-diagonal)	58.2 ± 3.2
WarpGrad	Linear (block-diagonal)	68.0 ± 4.4
WarpGrad	Non-linear (full)	81.3 ± 4.0

Second, we explore whether the geometry that we meta-learn under in the full Warp-Leap algorithm is approximately Fisher. In this experiment we use the main Warp-Leap architecture. We use a meta-learner trained on 25 tasks and that we evaluate on 10 held-out tasks. Because warp layers are linear in this configuration, if the learned geometry is approximately Fisher, post-warp activations should be zero-centred and the layer-wise covariance matrix should satisfy $\mathrm{Cov}(\omega^i(h^i(x)), \omega^i(h^i(x))) = I$, where I is the identify matrix [5]. If true, Warp-Leap would learn a block-diagonal approximation to the Inverse Fisher Matrix, as Natural Neural Nets.

To test this, during task adaptation on held-out tasks, we compute the mean activation in each convolutional layer pre- and post-warping. We also compute the Shatten-1 norm of the difference between layer activation covariance and the identity matrix pre- and post-warping, as described above. We average statistics over task and adaptation step (we found no significant variation in these dimensions).

Figure 6 summarise our results. We find that, in general, WarpGrad-Leap has zero-centered post-warp activations. That pre-warp activations are positive is an artefact of the ReLU activation function. However, we find that the correlation structure is significantly different from what we would expect if Warp-Leap were to represent the Fisher matrix; post-warp covariances are significantly different from the identity matrix and varies across layers.

These results indicate that WarpGrad methods behave distinctly different from Natural Gradient Descent methods. One possibility is that WarpGrad methods do approximate the Fisher Information Matrix, but with higher accuracy than other methods. A more likely explanation is that WarpGrad methods encode a different geometry since it can learn to leverage global information beyond the task at hand, which enables it to express geometries that standard Natural Gradient Descent cannot.

I *mini*ImageNet and *tiered*ImageNet

***mini*ImageNet** This dataset is a subset of 100 classes sampled randomly from the 1000 base classes in the ILSVRC-12 training set, with 600 images for each class. Following [39], classes are split into non-overlapping meta-training, meta-validation and meta-tests sets with 64, 16, and 20 classes in each respectively.

***tiered*ImageNet** As described in [41], this dataset is a subset of ILSVRC-12 that stratifies 608 classes into 34 higher-level categories in the ImageNet human-curated hierarchy [4]. In order to increase the separation between meta-train and meta-evaluation splits, 20 of these categories are used for meta-training, while 6 and 8 are used for meta-validation and meta-testing respectively. Slicing the class hierarchy closer to the root creates more similarity within each split, and correspondingly more diversity between splits, rendering the meta-learning problem more challenging. High-level categories are further divided into 351 classes used for meta-training, 97 for meta-validation and 160 for meta-testing, for a total of 608 base categories. All the training images in ILSVRC-12 for these base classes are used to generate problem instances for *tiered*ImageNet, of which there are a minimum of 732 and a maximum of 1300 images per class.

For all experiments, N-way K-shot classification problem instances were sampled following the standard image classification methodology for meta-learning proposed in [45]. A subset of N classes was sampled at random from the corresponding split. For each class, K arbitrary images were chosen without replacement to form the training dataset of that problem instance. As usual, a disjoint set of L images per class were selected for the validation set.

Few-shot classification In these experiments we used the established experimental protocol for evaluation in meta-validation and meta-testing: 600 task instances were selected, all using $N = 5$, $K = 1$ or $K = 5$, as specified, and $L = 15$. During meta-training we used $N = 5$, $K = 5$ or $K = 15$ respectively, and $L = 15$.

Task learners used 4 convolutional blocks defined by with 128 filters (or less, chosen by hyper-parameter tuning), 3×3 kernels and strides set to 1, followed by batch normalisation with learned scales and offsets, a ReLU non-linearity and 2×2 max-pooling. The output of the convolutional stack ($5 \times 5 \times 128$) was flattened and mapped, using a linear layer, to the 5 output units. The last 3 convolutional layers were followed by warp layers with 128 filters each. Only the final 3 task-layer parameters and their corresponding scale and offset batch-norm parameters were adapted during task-training, with the corresponding warp layers and the initial convolutional layer kept fixed and meta-learned using the WarpGrad objective. Note that, with the exception of CAVIA, other baselines do worse with 128 filters as they overfit; MAML and T-Nets achieve 46% and 49 % 5-way-1-shot test accuracy with 128 filters, compared to their best reported results (48.7% and 51.7%, respectively).

Hyper-parameters were tuned independently for each condition using random grid search for highest test accuracy on meta-validation left-out tasks. Grid sizes were 50 for all experiments. We choose the optimal hyper-parameters (using early stopping at the meta-level) in terms of meta-validation test set accuracy for each condition and we report test accuracy on the meta-test set of tasks. 60000 meta-training steps were performed using meta-gradients over a single randomly selected task instances and their entire trajectories of 5 adaptation steps. Task-specific adaptation was done using stochastic gradient descent without momentum. We use Adam [17] for meta-updates.

Multi-shot classification For these experiments we used $N = 10$, $K = 640$ and $L = 50$. Task learners are defined similarly, but stacking 6 convolutional blocks defined by 3×3 kernels and strides set to 1, followed by batch normalisation with learned scales and offsets, a ReLU non-linearity and 2×2 max-pooling (first 5 layers). The sizes of convolutional layers were chosen by hyper-parameter tuning to $\{64, 64, 160, 160, 256, 256\}$. The output of the convolutional stack ($2 \times 2 \times 256$) was flattened and mapped, using a linear layer, to the 10 output units.

Hyper-parameters were tuned independently for each algorithm, version and baseline using random grid search for highest test accuracy on meta-validation left-out tasks. Grid sizes were 200 for all multi-shot experiments. We choose the optimal hyper-parameters in terms of mean meta-validation test set accuracy AUC (using early stopping at the meta-level) for each condition and we report test accuracy on the meta-test set of tasks. 2000 meta-training steps were performed using averaged

meta-gradients over 5 random task instances and their entire trajectories of 100 adaptation steps with batch size 64, or inner-loops. Task-specific adaptation was done using stochastic gradient descent with momentum (0.9). Meta-gradients were passed to Adam in the outer loop.

We test WarpGrad against Leap, Reptile, and training from scratch with large batches and tuned momentum. We tune all meta-learners for optimal performance on the validation set. WarpGrad outperforms all baselines both in terms of rate of convergence and final test performance (Figure 7).

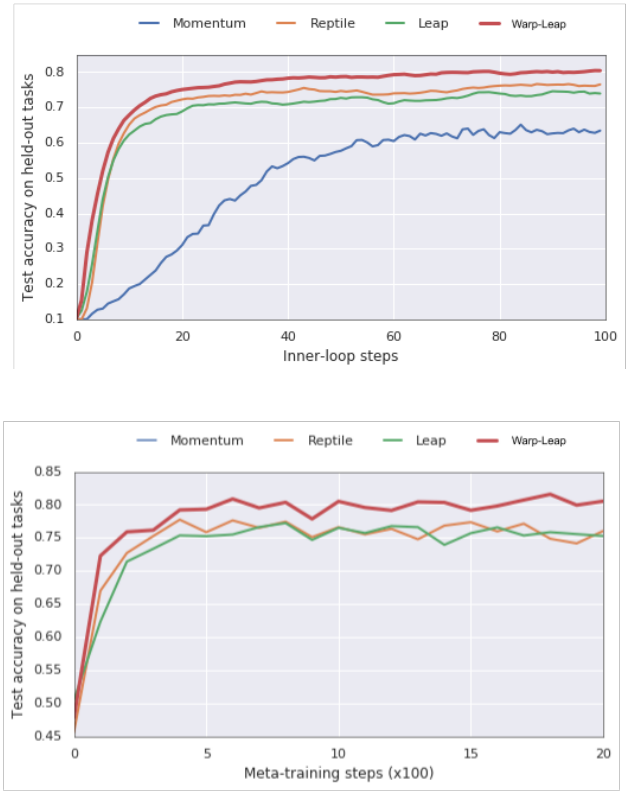


Figure 7: Multi-shot *tiered*ImageNet results. *Top:* mean learning curves (test classification accuracy) on held-out meta-test tasks. *Bottom:* mean test classification performance on held-out meta-test tasks during meta-training. Training from scratch omitted as it is not meta-trained.

J Maze Navigation

To illustrate both how WarpGrad may be used with Recurrent Neural Networks in an online meta-learning setting, as well as in a Reinforcement Learning environment, we evaluate it in a maze navigation task proposed by [29]. The environment is a fixed maze and a task is defined by randomly choosing a goal location in the maze. During a task episode of length 200, the goal location is fixed but the agent gets teleported once it finds it. Thus, during an episode the agent must first locate the goal, then return to it as many times as possible, each time being randomly teleported to a new starting location. We use an identical setup as [30], except our grid is of size 11×11 as opposed to 9×9. We compare our Warp-RNN to a Learning to Reinforcement Learn [46] and Hebbian meta-learners [29, 30].

The task learner in all cases is an advantage actor-critic [46], where the actor and critic share an underlying basic RNN, whose hidden state is projected into a policy and value function by two separate linear layers. The RNN has a hidden state size of 100 and $\tanh$ non-linearities. Following [30], for all benchmarks, we train the task learner using Adam with a learning rate of $1e-3$ for 200 000 steps using batches of 30 episodes, each of length 200. Meta-learning arises in this setting as each episode encodes a different task, as the goal location moves, and by learning across episodes the

RNN is encoding meta-information in its parameters that is can leverage during task adaptation (via its hidden state [13, 46]). See [30] for further details.

We design a Warp-RNN by introducing a warp-layer in the form of an LSTM that is frozen for most of the training process. Following [8], we use this meta-LSTM to modulate the task RNN. Given an episode with input vector x_t, the task RNN is defined by

$$h_t = \tanh\left(U^2_{h,t} V U^1_{h,t}\, h_{t-1} + U^2_{x,t} W U^1_{x,t}\, x_t + U^b_t\, b\right), \tag{15}$$

where $W,\ V,\ b$ are task-adaptable parameters; each $U^i_{j,t}$ is a diagonal warp matrix produced by projecting from the hidden state of the meta-LSTM, $U^i_{j,t} = \mathrm{diag}(\tanh(P^i_j z_t))$, where z is the hidden-state of the meta-LSTM. See [8] for details. Thus, our Warp-RNN is a form of HyperNetwork (see Figure 3, Appendix A). Because the meta-LSTM is frozen for most of the training process, task adaptable parameters correspond to those of the baseline RNN.

To control for the capacity of the meta-LSTM, we also train a HyperRNN where the LSTM is updated with every task adaptation; we find this model does worse than the WarpGrad-RNN. We also compare the non-linear preconditioning that we obtain in our Warp-RNN to linear forms of preconditioning defined in prior works. We implement a T-Nets-RNN meta-learner, defined by embedding linear projections T_h, T_x and T_b that are meta-learned in the task RNN, $h_t = \tanh(T_h V h_t + T_x W x_t + b)$. Note that we cannot backpropagate to these meta-parameters as per the T-Nets (MAML) framework. Instead, we train T_h, T_x, T_b with the meta-objective and meta-training algorithm we use for the Warp-RNN. The T-Nets-RNN does worse than the baseline RNN and generally fails to learn.

We meta-train the Warp-RNN using the continual meta-training algorithm (Algorithm 3, see Appendix C for details), which accumulates meta-gradients continuously during training. Because task training is a continuous stream of batches of episodes, we accumulating the meta-gradient using the approximate meta-objective that detaches the inner task gradient. Both $\mathcal{L}^\tau_{\text{task}}$ and $\mathcal{L}^\tau_{\text{meta}}$ are the advantage actor-critic objective. We and update warp-parameters on every 30th task parameter update. We detail the meta-objective in Appendix D (see Eq. 13). Our implementation of a Warp-RNN can be seen as meta-learning "slow" weights to facilitate learning of "fast" weights [43, 32]. Implementing Warp-RNN requires four lines of code on top of the standard training script. The task-learner is the same in all experiments with the same number of learnable parameters and hidden state size. Compared to all baselines, we find that the Warp-RNN converges faster and achieves a higher cumulative reward (Figure 2 and Figure 8).

K Meta-Learning for Continual Learning

Online SGD and related optimisation methods tend to adapt neural network models to the data distribution encountered last during training, usually leading to what has been termed "catastrophic forgetting" [9]. In this experiment, we investigate whether WarpGrad optimisers can meta-learn to avoid this problem altogether and directly minimise the joint objective over all tasks with every update in the fully online learning setting where no past data is retained.

Continual Sine Regression We propose a continual learning version of the sine regression meta-learning experiment in [6]. We split the input interval $[-5, 5] \subset \mathbb{R}$ evenly into 5 consecutive sub-intervals, corresponding to 5 regression tasks. These are presented one at a time to a task learner, which adapts to each sub-task using 20 gradient steps on data from the given sub-task only. Batch sizes were set to 5 samples. Sub-tasks thus differ in their input domain. A task sequence is defined by a target function composed of two randomly mixed sine functions of the form $f_{a_i,b_i}(x) = a_i \sin(x - b_i)$ each with randomly sampled amplitudes $a_i \in [0.1, 5]$ and phases $b_i \in [0, \pi]$. A task $\tau = (a_1, b_1, a_2, b_2, o)$ is therefore defined by sampling the parameters that specify this mixture; a task specifies a target function f_τ by

$$f_\tau(x) = \alpha_o(x) f_{a_1,b_1}(x) + (1 - \alpha_o(x)) f_{a_2,b_2}(x), \tag{16}$$

where $\alpha_o(x) = \sigma(x + o)$ for a randomly sampled offset $o \in [-5, 5]$, with σ being the sigmoid activation function.

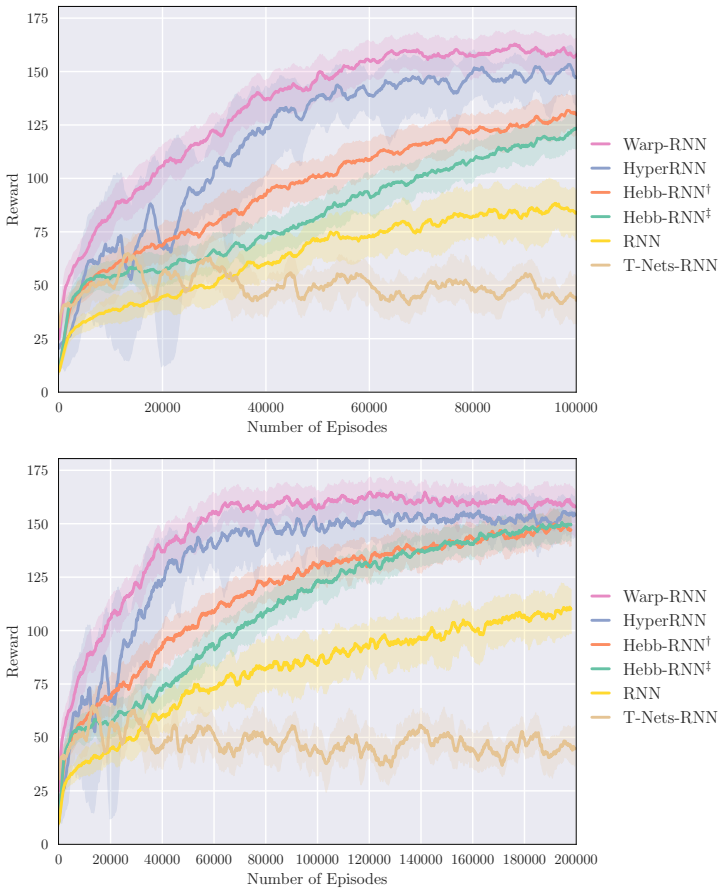


Figure 8: Mean cumulative return for maze navigation task, for 200000 training steps. Shading represents inter-quartile ranges across 10 independent runs.†Simple modulation and ‡retroactive modulation, respectively [30].

Model We define a task learner as 4-layer feed-forward networks with hidden layer size 200 and ReLU non-linearities to learn the mapping between inputs and regression targets, $h(\cdot, \theta, \phi)$. For each task sequence τ, a task learner is initialised from a fixed random initialisation θ_0 (that is not meta-learned). Each non-linearity is followed by a residual warping block consisting of 2-layer feed-forward networks with 100 hidden units and $\tanh$ non-linearities, with meta-learned parameters ϕ which are fixed during the task adaptation process.

Continual learning as task adaptation The task target function f_τ is partition into 5 sets of sub-task. The task learner sees one partition at a time and is given $n = 20$ gradient steps to adapt, for a total of $K = 100$ steps of online gradient descent updates for the full task sequence; recall that every such sequence starts from a fixed random initialisation θ_0. The adaptation is completely online since at step $k = 1, \ldots, K$ we sample a new mini-batch D^k_{task} of 5 samples from a single sub-task (sub-interval). The data distribution changes after each $n = 20$ steps with inputs x coming from the next sub-interval and targets form the same function $f_\tau(x)$. During meta-training we always present tasks in the same order, presenting intervals from left to right. The online (sub-)task loss is defined on the current mini-batch D^k_{task} at step k:

$$\mathcal{L}^\tau_{\text{task}}\left(\theta^\tau_k, D^k_{\text{task}}; \phi\right) = \frac{1}{2|\,\mathcal{D}^k_{\text{task}}\,|} \sum_{x \in D^k_{\text{task}}} \left(h(x, \theta^\tau_k; \phi) - f_\tau(x)\right)^2. \tag{17}$$

Adaptation to each sub-task uses sub-task data only to form task parameter updates $\theta^\tau_{k+1} \leftarrow \theta^\tau_k - \alpha \nabla \mathcal{L}^\tau_{\text{task}}\left(\theta^\tau_k, D^k_{\text{task}}; \phi\right)$. We used a constant learning rate $\alpha = 0.001$. Warp-parameters ϕ are fixed

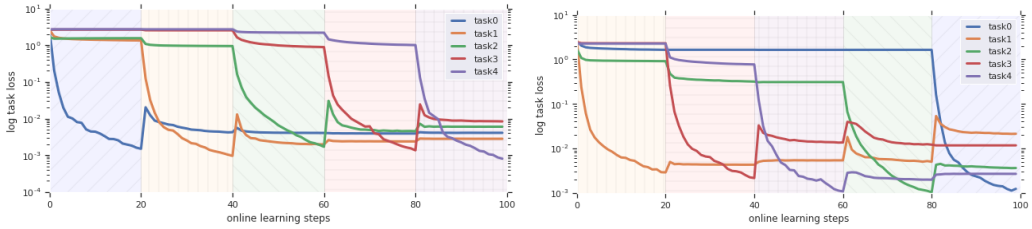
across the full task sequence during adaptation. They are meta-learned across random samples of task sequences, which we describe next.

Meta-learning an optimiser for continual learning To investigate the ability of WarpGrad to learn an optimiser for continual learning that mitigates catastrophic forgetting, we fix a random initialisation prior to meta-training that is not meta-learned; every task learner is initialised with these parameters. To meta-learn an optimiser for continual learning, we need a meta-objective that encourages such behaviour. Here, we take a first step towards a framework for meta-learned continual learning. We define the meta-objective $\mathcal{L}^{\tau}_{\text{meta}}$ as incremental multitask objective that, for each sub-task τ_t in a given task sequence τ, averages the validation sub-task losses (Eq. 17) for the current and every preceding loss in the task sequence. The task meta-objective is defined by summing over all sub-tasks in the task sequence. For some sub-task parameterisation θ^{τ_t}, we have

$$\mathcal{L}^{\tau}_{\text{meta}}\left(\theta^{\tau_t}; \phi\right) = \sum_{i=1}^{t} \frac{1}{n(T-t+1)} \mathcal{L}^{\tau}_{\text{task}}\left(\theta^{\tau_i}, D^{i}_{\text{val}}; \phi\right). \tag{18}$$

As before, the full meta-objective is an expectation over the joint task parameter distribution (Eq. 3); for further details on the meta-objective, see Appendix D, Eq. 14. This meta-objective gives equal weight to all the tasks in the sequence by averaging the regression step loss over all sub-tasks where a prior sub-task should be learned or remembered. For example, losses from the first sub-task, defined using the interval $[-5, -3]$, will appear nT times in the meta-objective. Conversely, the last sub-task in a sequence, defined on the interval $[3, 5]$, is learned only in the last $n = 20$ steps of task adaptation, and hence appers n times in the meta-objective. Normalising on number of appearances corrects for this bias. We trained warp-parameters using Adam and a meta-learning rate of 0.001, sampling 5 random tasks to form a meta-batch and repeating the process for 20 000 steps of meta-training.

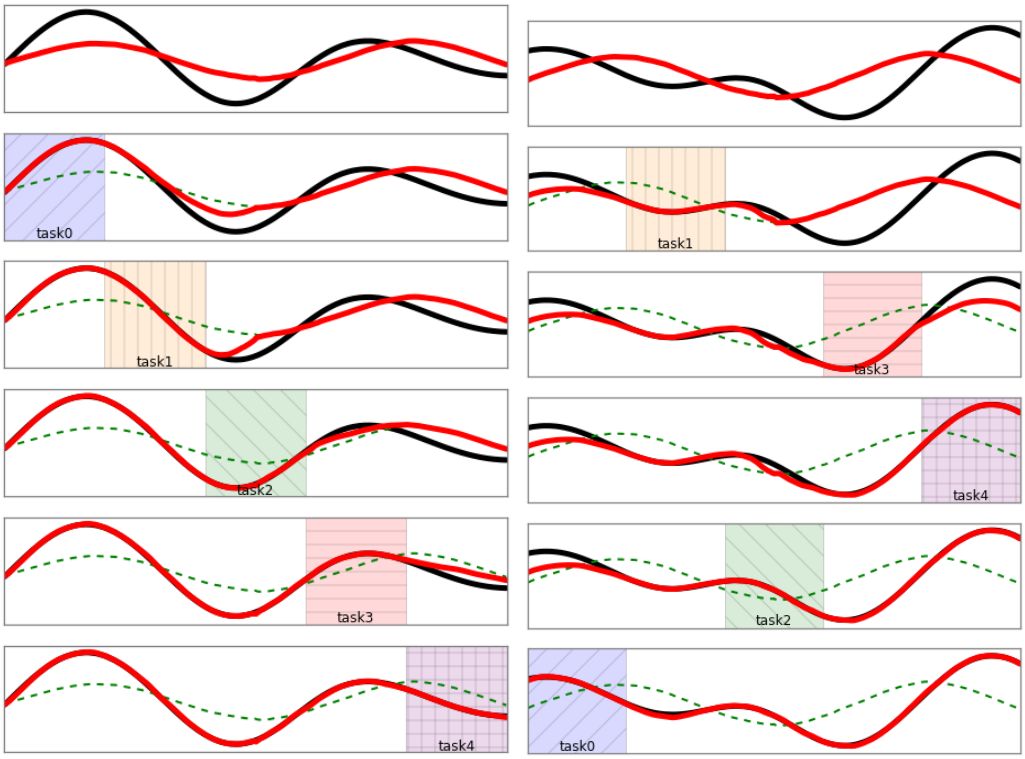
Results Figure 9 shows a breakdown of the validation loss across the 5 sequentially learned tasks over the 100 steps of online learning during task adaptation. Results are averaged over 100 random regression problem instances. The meta-learned WarpGrad optimiser reduces the loss of the task currently being learned in each interval while also largely retaining performance on previous tasks. There is an immediate relatively minor loss of performance, after which performance on previous tasks is retained. We hypothesise that this is because the meta-objectives averages over the full learning curve, as opposed to only the performance once a task has been adapted to. As such, the WarpGrad optimiser may allow for some degree of performance loss. Intriguingly, in all cases, after an initial spike in previous sub-task losses when switching to a new task, the spike starts to revert back some way towards optimal performance, suggesting that the WarpGrad optimiser facilitates positive backward transfer, without this being explicitly enforced in the meta-objective. Deriving a principled meta-objective for continual learning is an exciting area for future research.



(a) Task order as seen during meta-training.

(b) Random task order.

Figure 9: Continual learning regression experiment. Average log-loss over 100 randomly sampled tasks. Each task contains 5 sub-tasks learned (a) sequentially as seen during meta-training or (b) in random order [sub-task 1, 3, 4, 2, 0]. We train on each sub-task for 20 steps, for a total of $K = 100$ task adaptation steps.



(a) Task order seen during meta-training.

(b) Random task order.

Figure 10: Continual learning regression: evaluation after partial task adaptation. We plot the ground truth (black), task learner prediction before adaptation (dashed green) and task learner prediction after adaptation (red). Each row illustrates how task learner predictions evolve (red) after training on sub-tasks up to and including that sub-task (current task illustrate in plot). (a) sub-tasks are presented in the same order as seen during meta-training; (b) sub-tasks are presented in random order at meta-test time in sub-task order [1, 3, 4, 2 and 0].