MINT: MATRIX-INTERLEAVING FOR MULTI-TASK LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning enables training of large and flexible function approximators from scratch at the cost of large amounts of data. Applications of neural networks often consider learning in the context of a single task. However, in many scenarios what we hope to learn is not just a single task, but a model that can be used to solve multiple different tasks. Such multi-task learning settings have the potential to improve data efficiency and generalization by sharing data and representations across tasks. However, in some challenging multi-task learning settings, particularly in reinforcement learning, it is very difficult to learn a single model that can solve all the tasks while realizing data efficiency and performance benefits. Learning each of the tasks independently from scratch can actually perform better in such settings, but it does not benefit from the representation sharing that multi-task learning can potentially provide. In this work, we develop an approach that endows a single model with the ability to represent both extremes: joint training and independent training. To this end, we introduce matrix-interleaving (Mint), a modification to standard neural network models that projects the activations for each task into a different learned subspace, represented by a per-task and per-layer matrix. By learning these matrices jointly with the other model parameters, the optimizer itself can decide how much to share representations between tasks. On three challenging multi-task supervised learning and reinforcement learning problems with varying degrees of shared task structure, we find that this model consistently matches or outperforms joint training and independent training, combining the best elements of both.

1 INTRODUCTION

While deep learning has enabled remarkable levels of generalization through the use of function approximators, this comes at the cost of large amounts of data, which remains a critical challenge in deploying deep learning to a number of domains. When combined with deep networks, multi-task learning offers the promise of building more powerful representations using less data per task, leading to greater performance and data efficiency. However, multi-task deep learning has also posed considerable challenges. Numerous works have observed that joint training on multiple tasks can actually *decrease* task performance due to the negative influence of other tasks (Parisotto et al., 2015; Rusu et al., 2016a). Indeed, training networks entirely independently on each task has remained a strong approach, to the point that multiple multi-task methods have first trained models independently before using them to train a multi-tasking model (Parisotto et al., 2015; Rusu et al., 2016a; Ghosh et al., 2017; Teh et al., 2017; Czarnecki et al., 2019). Moreover, our experiments in Section 6 indicate that three recently proposed methods for multi-task learning are all surpassed by training models independently per task. However, training independent models will only work well when provided enough data per task, and precludes potential positive data-efficiency gains from multi-task learning, only providing protection against negative transfer. Further, while a number of works have successfully shared parameters, finding an architecture with the appropriate level of parameter sharing for a given problem domain can require a considerable amount of manual engineering. In this work, we aim to develop a multi-task learning method that can perform well both when tasks share very little and when they share a large amount of structure.

To address this problem, we consider how we might develop a single neural network model that can both represent independent models, when optimization challenges prevail, and a single model with shared weights, when sharing may be beneficial, as well as intermediate levels of model sharing. One option for performing independent training within a single model is to put separate networks with independent weights into a single model, using the task ID to select which network prediction to output. However, this prevents any sharing. An alternative approach is to condition the model on the task ID, through various conditioning approachs, including additive and multiplicative approaches such as FiLM (Perez et al., 2018). In fact, point-wise multiplicative conditioning, as proposed in FiLM, can indeed represent separate networks by selecting which parts of the network to be used for different tasks, as can a number of of other approaches in multi-task learning (Rosenbaum et al., 2017; 2019; Fernando et al., 2017). Yet, these approaches still require an optimization over shared parameters in order to select how parameters used for each task, which can introduce significant optimization challenges.

We instead consider how to allow a model to perform optimization on only shared parameters, only disjoint parameters, or any combination thereof. We can achieve this by simply interleaving learned per-task matrices at each layer of a jointly-trained neural network. When optimization over shared parameters is ineffective, the model can still represent a full neural network per task, resulting in independent training; while using identical per-task matrices results in standard joint training. Intermediately, a mix of shared and per-task parameters may be used. In effect, by incorporating these matrices into the network, the optimizer itself can automatically and dynamically modulate the degree to which a representation is shared between tasks, depending on the problem domain and the optimization progress, and can do so *without* having to optimize shared parameters.

The primary contribution of this paper is a simple yet effective approach for multi-task learning that can represent and smoothly interpolate between independent training and joint training, via matrix interleaving (Mint). We describe how we can implement Mint in deep multi-task models and show its effectiveness in improving data efficiency and generalization in multi-task settings while providing intuition about the reasons why this architecture performs so well. Further, we show that the model can be extended to goal-conditioned reinforcement learning in a straightforward manner by allowing the model to generate the interleaved matrices conditioned on task information such as the goal. We evaluate Mint on sets of tasks with both high and low levels of shared structure and find that it performs well in both settings, performing comparably to or outperforming both joint training and independent training, effectively combining the best elements of both. Further, in comparison to previous methods that use multiplicative interactions for continual learning (Cheung et al., 2019) and for general conditioning (Perez et al., 2018), Mint is better able to separate tasks by avoiding the need to optimize over shared parameters and can *empirically* produce substantially better performance on a range of challenging multi-task problems. Finally, Mint also outperforms state-of-the-art approaches for multi-task learning while being significantly simpler to implement.

2 PRELIMINARIES

In multi-task learning, the goal is to find a θ-parameterized model f_θ that reaches high performance across all training tasks drawn from a task distribution $\mathcal{T}_i \sim p(\mathcal{T})$, i.e. $\min_\theta \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})}\left[\mathcal{L}_i(f_\theta)\right]$, where $\mathcal{L}_i$ denotes the loss function for task $\mathcal{T}_i$. In Section 4, we will study this multi-task problem in the context of supervised and reinforcement learning settings. In our multi-task learning set-up, we train a model that is conditioned on z_k, where the additional input z_k is used to specify which task should be performed and can be represented in a variety of ways, from simple categorical variables to learned task embeddings (Hausman et al., 2018). This formulation can be readily extended to a goal-conditioned reinforcement learning setting, where z_k indicates the desired goal.

Joint and fully independent training are the two extremes of multi-task learning. Assuming a set of n training tasks, we characterize multi-task training as independent if it optimizes a set of task-specific, disjoint parameters $\{\theta^k\}_{i=1}^n$ that parameterize the model f_{θ^k}. We define joint training as finding a single set of task-independent, shared parameters θ of f_θ. Note that joint training utilizes the fact that the parameters are shared *throughout* learning. While joint training has a number of data-efficiency and generalization benefits, it can be quite difficult to train effectively.

3 MULTI-TASKING WITH INTERLEAVED MATRICES

Considering fully independent and fully joint training as two ends of the spectrum in multi-task learning problems, we want to design an algorithm to get the best of both worlds – the stability of independent training and the parameter-sharing efficiency of fully jointly trained models. To this end, we propose to build models that allow the neural network to learn how much information should be shared and between which tasks throughout learning, in an adaptive fashion. We describe the details of our approach below.

3.1 MintNet: Matrix-Interleaved Networks

Given the success of independent training in comparison to a range of approaches, we hypothesize that significant optimization challenges for parameters shared across tasks have hindered performance. To this end, we aim to study how a model can represent independent training *without* having to optimize any shared parameters, effectively bypassing any optimization issues that may arise when training across disjoint tasks. In particular, we consider how we might allow the optimizer to dynamically modulate how much each learned representation is shared versus disjoint across different tasks.

In the case of neural network models, we view representations as network activations at different layers of the network. We aim to introduce a modification to the neural network that would allow the model to either form those activations in a completely task-specific way, in a completely shared way, or in a way that shares to an intermediate degree. To achieve this, we propose a model architecture that transforms the previous layer's representation both in a task-general way and in a task specific way, in sequence. When two tasks share very little, the network can optimize task-specific weights, while when the tasks share a considerable degree of structure, the network can leverage the shared weights. This modification, which we refer to as a matrix-interleaving (Mint), effectively allows us to "interpolate" between fully shared representations across tasks and learning separate representations for different tasks. Since these transformations are task-specific and can be introduced at various layers of the network, they allow for a different amounts of representation shared at different levels of the neural network model.

To understand the practical implementation, we consider the activations at layer l of a fully-connected neural network: $y^{(l)} = \sigma\left(W^{(l)}y^{(l-1)} + b^{(l)}\right)$, where $W^{(l)}$ is the weight matrix for layer l, $b^{(l)}$ is the bias vector for layer l, and σ is the non-linearity at each layer. The Mint layer augments the traditional fully-connected layer with task-specific weight matrix M_k and bias vector β_k, where k indexes the task. The forward pass of a Mint layer for some vector of activations y is presented in Definition 1.

Definition 1. *A Mint layer applies an affine transformation to activations $y \in \mathbb{R}^n$ as follows, yielding new activations a:*

$$a = \texttt{\textbf{Mint}}(y) = M_k y + \beta_k \tag{1}$$

where M_k and β_k are per-layer *task-specific* matrix and bias, respectively. A neural network augmented with Mint thus contains parameters that are both shared across all tasks as well parameters that are only used for a particular task for each layer l, i.e. $\theta = \{W^{(l)}, b^{(l)}, M_k^{(l)}, \beta_k^{(l)}\}$. See Figure 1 for a visual depiction of the application of Mint. We show how the regular fully-connected layers and Mint layers can be interleaved in Equation 2 and 3 below:

$$y^{(l)} = \sigma\left(W^{(l)}a^{(l-1)} + b^{(l)}\right) \tag{2}$$

$$a^{(l)} = M_k^{(l)}y^{(l)} + {\beta_k}^{(l)}. \tag{3}$$

Because this layer is fully-differentiable, we can learn the task-specific matrices $M_k^{(l)}$ and biases $\beta_k^{(1)}$ jointly with the shared parameters of the model.

When we apply Mint to tasks with very large numbers of tasks, or arbitrary task descriptors (e.g., goal-conditioning), we can train separate neural networks $T_\phi^l, T_\psi^l \rightarrow M_k^l, \beta_k^l$, to output the Mint matrices and biases at every layer, instead of storing independent matrices for each task. In this case, Mint resembles FiLM, only with a matrix transforming the activations at each layer instead of a point-wise multiplication by a vector. In the next section, we study a theoretical property of Mint. We validate the benefits of Mint in our experimental evaluation in Section 6.

3.2 Theoretical Analysis

We next aim to theoretically study the expressive power of Mint when the shared parameters are not optimized effectively (e.g. due to optimization challenges). We assume only that the shared parameter

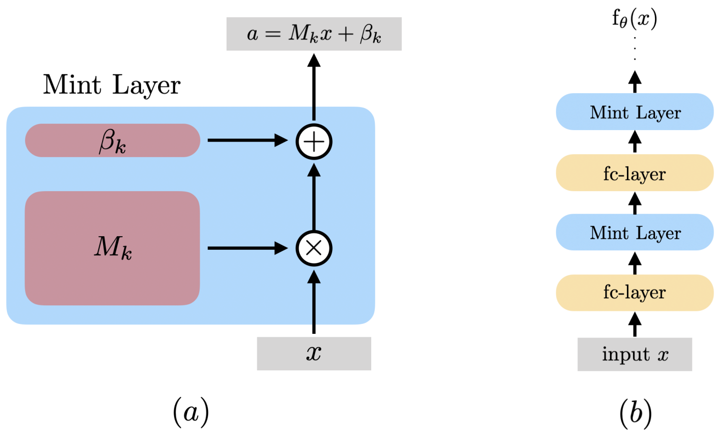

Figure 1: Mint architecture. (a) A Mint Layer applies a task-specific matrix multiplication and vector addition to an input vector x. The matrix and vector make up an affine transformation to the input vector, given by a task network. (b) The modified neural network model with a distinct Mint layer added after each fully-connected layer.

matrices $W^{(l)}$ are invertible, such that they do not lose information about the input. This assumption critically does not require the optimizer to productively update the shared parameters. Under this assumption, we show that Mint has the valuable property of universal function approximation *for each task*. That is, if we train a Mint-augmented network on a set of tasks $\{\mathcal{T}_i\}$, we show universal approximation guarantees for the function Mint learns for each individual task, even if the shared parameters are not optimized. This property is extremely significant in light of potential failures of optimization of shared weights during multi-task training. Further, we also relate this to the expressive power of FiLM.

Theorem 1. *Let $y^{(l-1)}$ be the activations of layer $l-1$, and let $W^{(l)}, b^{(l)}$ be the weight matrix and bias vector at the l-th layer. We assume that $W^{(l)}$ is a fixed invertible matrix, rather than a matrix of free parameters. Then, we have two results:*

1. *Applying Mint to $y^{(l-1)}$ allows us to express arbitrary affine transformations at layer l for each task.*
2. *Applying FiLM to $y^{(l-1)}$ does not allow us to express arbitrary affine transformations at layer l for each task.*

The proof can be found in Appendix A. Crucially, the implication of this theorem is that Mint satisfies the universal function approximation property (Hornik et al., 1989) of neural networks even when the shared weight matrices are not learned; this property does not hold when the per-task transformations correspond to point-wise multiplication of vectors. In the following sections, we empirically evaluate Mint, and observe that the small yet important distinction between task matrices and task vectors leads to pronounced chnages in multi-task learning performance and learning efficiency, and enables Mint to learn well when tasks are distinct.

4 Flavors of Mint

The general idea of Mint is implemented as follows in the supervised learning and reinforcement learning cases.

Multi-Task Supervised Learning In the case of supervised learning models, we simply apply Mint layers after every fully connected layer of the model. Specifically, for a task identifier $z_k \in \mathbb{R}^K$ where K is the number of tasks and for every layer l with activations $a^{(l)} \in \mathbb{R}^n$, nonlinearity σ and weight $W^{(l)}$ and bias $b^{(l)}$, we represent the transformation using two matrices $T_M^{(l)} \in \mathbb{R}^{n\times n\times K}$ and $T_\beta^{(l)} \in \mathbb{R}^{n\times K}$ that take in the task identifier and output the per-layer task-specific matrices $M^{(l)}$ and biases β^l respectively. The transformation can be summarized as follows:

$$a^{(l)} = (T_M^{(l)} z_k)\sigma(W^{(l)} a^{(l-1)} + b^{(l)}) + T_\beta^{(l)} z_k = {M_k}^{(l)} \sigma(W^{(l)} a^{(l-1)} + b^{(l)}) + {\beta_k}^{(l)}. \quad (4)$$

Multi-Task Reinforcement Learning For multi-task reinforcement learning, we implement the architecture similarity to the supervised learning case but we combine this with actor-critic RL algorithms by introducing this architecture into both the critic $Q(s, a, z_k)$ and the actor $\pi(a|s, z_k)$.

Goal-Conditioned Reinforcement Learning For the case of goal conditioned RL, we introduce a slightly modified Mint architecture into both the actor and the critic conditioned on the task goal g. Specifically, for every layer l with activations $a^{(l)}$, nonlinearity σ, weight $W^{(l)}$, and bias $b^{(l)}$, we represent two transformation function $T_\phi^{(l)}$ and $T_\psi^{(l)}$ by two simple neural networks that take in the goal and produces a per-layer goal-specific matrix $M^l(g)$ and the bias $\beta^{(l)}(g)$ respectively. The transformation can be summarized as:

$$a^{(l)} = T_\phi^{(l)}(g)\sigma(W^{(l)} a^{(l-1)} + b^{(l)}) + T_\psi^{(l)}(g) = M(g)^{(l)} \sigma(W^{(l)} a^{(l-1)} + b^{(l)}) + \beta(g)^{(l)}. \quad (5)$$

5 Related Work

Multi-task learning (Caruana, 1997; Bakker & Heskes, 2003; Ruder, 2017) focuses on the problem of finding a single model that can solve multiple different tasks. This formulation can be readily applied to a variety of learning domains, such as supervised learning (Zhang et al., 2014; Long & Wang, 2015; Yang & Hospedales, 2016; Sener & Koltun, 2018; Zamir et al., 2018), and multi-task (Espeholt et al., 2018; Wilson et al., 2007; Hessel et al., 2019) and goal-conditioned reinforcement learning (Kaelbling, 1993; Andrychowicz et al., 2017; Pong et al., 2018). While multi-task learning offers the promise of efficient training of shared representations, naïvely training a single model on multiple tasks often does not result in these desired benefits, due to the optimization challenges introduced by the multi-task setting (Teh et al., 2017; Ghosh et al., 2017; Rusu et al., 2016a).

In order to eliminate potential negative interference between different tasks during multi-task learning using a single model, many approaches propose to learn each task separately, to later combine their solutions into a single multi-task model (Levine et al., 2016; Teh et al., 2017; Ghosh et al., 2017; Rusu et al., 2016a; Czarnecki et al., 2019; Parisotto et al., 2015). In contrast to these works, we present a method that is able to train a single model on multiple tasks, and is able to interpolate between the extremes of joint and independent training,

More closely related to our approach, various architectural solutions have been proposed to increase the multi-task learning capability of the model. Example approaches include architectural changes that allow multiple modules or paths within the same network (Fernando et al., 2017; Devin et al., 2016; Misra et al., 2016; Rusu et al., 2016b; Vandenhende et al., 2019; Rosenbaum et al., 2017), transformation-based task conditioning (Perez et al., 2018), attention-based architectures (Liu et al., 2018; Maninis et al., 2019), multi-headed network solutions (Long & Wang, 2015; Riedmiller et al., 2018; Wulfmeier et al., 2019), and a variety of other approaches (Hashimoto et al., 2016; Ruder12 et al., 2017). We demonstrate an approach that allows for a middle ground between the conceptual extremes of fully independent training at one end and single-model joint training at the other. This added flexibility enables us to sidestep the negative effects of potential task interference while at the same time share parameters between the tasks when appropriate. In our experiments, we provide direct comparisons of our method to cross-stich (Misra et al., 2016), routing networks (Rosenbaum et al., 2019), the FiLM architecture (Perez et al., 2018), rotational superposition (Cheung et al., 2019), and multi-headed models.

6 Experiments

The goal of our experimental evaluation is to answer the following questions: (1) does our method enable effective multi-task learning both in settings where there is substantial overlap in tasks and where there is little overlap?, (2) how does our method compare to independent training and joint training in those settings?, (3) how does our method compare to prior state-of-the-art approaches?

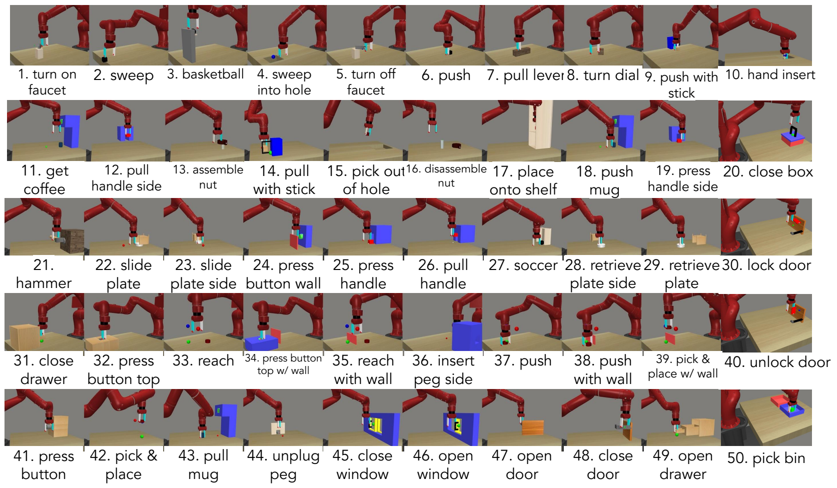

Figure 2: Visualization of the 50 tasks from Meta-World used in the MT50 evaluation. Mint is able to learn about 30 of these tasks.

To answer the above questions, we conduct experiments on both multi-task supervised and reinforcement learning domains. For multi-task supervised learning, we evaluate our method on the CIFAR-100 multi-task dataset, where we treat the 20 coarse labels as different tasks. For multi-task RL doamins, we perform experiments on two multi-task RL benchmark variants MT10 and MT50 (as showcased in Figure 2) proposed by Yu et al. (2019). Finally, to test if Mint can excel in continuous task distributions, we also evaluate the method on a goal-conditioned RL domain where a Sawyer robot arm is required to push a randomly initialized puck to different goal positions. For all RL experiments, we use the popular off-policy RL algorithm, soft actor-critic (SAC) (Haarnoja et al., 2018), which has shown to solve many RL benchmarks with great data-efficiency.

In the multi-task supervised learning setting, we consider the following comparisons:

- **task specific-1-fc (Rosenbaum et al., 2017)**: learn a shared network along with a final task-specific layer.
- **task specific-all-fc (Rosenbaum et al., 2017)**: learn a shared convolutional neural network and a task-specific fully-connected networks.
- **cross stitch-all-fc (Misra et al., 2016)**: learn one neural network per task while using cross-stitch units to share features across tasks.
- **routing-all-fc+WPL (Rosenbaum et al., 2019)**: learn a router that chooses a particular neural network for each task.
- **independent**: learn one separate network per task.

On the multi-task RL domains, we compare Mint to the following methods:

- **SAC**: train a vanilla SAC agent with task identifier as part of the input.
- **Multi-head SAC**: train a SAC agent where both the actor and critic are represented as multi-head feedforward neural networks where the number of heads is the number of tasks.
- **SAC (concat all fc)**: train a SAC agent where the task identifier z is concatenated with the activation at each layer and passed as inputs to the next layer.
- **FiLM (Perez et al., 2018)**: the actor and critic are learned with neural networks combined with FiLM.
- **superposition (Cheung et al., 2019)**: the actor and critic are learned with neural networks combined with superposition.
- **independent**: learn separate actor and critic per task.

We provide details of architecture design in each domain as well as environment set-up in the Appendix B.

6.1 Multi-Task Supervised Learning

On the CIFAR-100 multi-task domain, as shown in Table 1, Mint is able achieve the best classification accuracy and outperform other methods by a least 2%, which demonstrates that Mint attains the competitive results in established multi-task supervised learning benchmarks. Note that the number of parameters of Mint is much smaller than popular multi-task supervised learning methods such as cross stitch and routing networks and also independent networks, implying that Mint can efficiently learn to interpolate between representing separate networks and sharing common structure among tasks.

	% accuracy	# of parameters
task specific-1-fc (Rosenbaum et al., 2017)	42	104K
task specific-all-fc (Rosenbaum et al., 2017)	49	689K
cross stitch-all-fc (Misra et al., 2016)	53	1.23M
routing-all-fc + WPL (Rosenbaum et al., 2019)	64.1	1.31M
independent	64.3	1.56M
Mint (ours)	**66.3**	699K

Table 1: Results on CIFAR-100 multi-task dataset. Independent training of task models performs surprisingly well, outperforming three recently proposed approaches. Mint, however, achieves the best performance with less number of parameters comparing to more complicated methods such as routing-all-fc and independent training.

6.2 Multi-Task Reinforcement Learning

On the RL domain, we first investigate the ability of Mint to perform a set of distinct RL tasks. As discussed in Yu et al. (2019), MT10 and MT50 serve as representative benchmarks to evaluate multi-task learning algorithms on learning a diverse range of robotics manipulation tasks. We present the results in Figure 4. The success rates are averaged across tasks and we adopt the success metrics used in the Meta-World benchmark. We design appropriate architectures of the actor and the critic of the SAC algorithm for each method such that the number of parameters of each approach is around the same magnitude (see Appendix B for details).

For MT10, Mint learns all tasks with the best data efficiency, while independent networks also learn all of the tasks with slightly worse data-efficiency. The other methods are unable to acquire half of the skills. Mint, on the other hand, enjoys the expressive power to interpolate between independent learning and sharing while mitigating optimization challenges to attain the best results between the two extremes.

For MT50, where the evaluations are done on all 50 of the Meta-World environments, as shown on the right in Figure 4, Mint quickly learns to solve more than 60% of tasks in 20 million environment steps while SAC and SAC with multi-head architectures struggled in solve 40% of the tasks after 35 million steps. Independent networks learn to solve the tasks slower than Mint but eventually surpasses it.[1] This result also validates the expressive power of Mint to represent both separate learning and learning with shared networks.

6.3 Goal-Conditioned Reinforcement Learning

Next, We consider the question if Mint can be applied to a set of RL goals that have considerable shared structure. Hence, we evaluate all methods on the goal-conditioned sawyer pushing domain. In this environment, the goal space consists of the initial positions of the puck and the goal positions. For details of the goal-conditioned environment, see Appendix B. At each policy rollout step, we sample a batch of 9 goals and collect 3 paths for each goal, where all the paths are stored in the task-specific replay buffers. At each training step, we sample a batch of 9 goals and 128 samples per goal from their corresponding replay buffers. To prevent from creating infinite number of replay buffers, we discretize the goal space into 200 goals. Given that it's impractical to train 200 independent agents, we sample 10 goals from the goal space and train 10 independent SAC agents for estimating the performance of independent training in goal-conditioned pushing. As shown in Figure 3, Mint outperforms all

[1]We did not evaluate superposition and FiLM on MT50 given the poor performance of these methods on MT10.

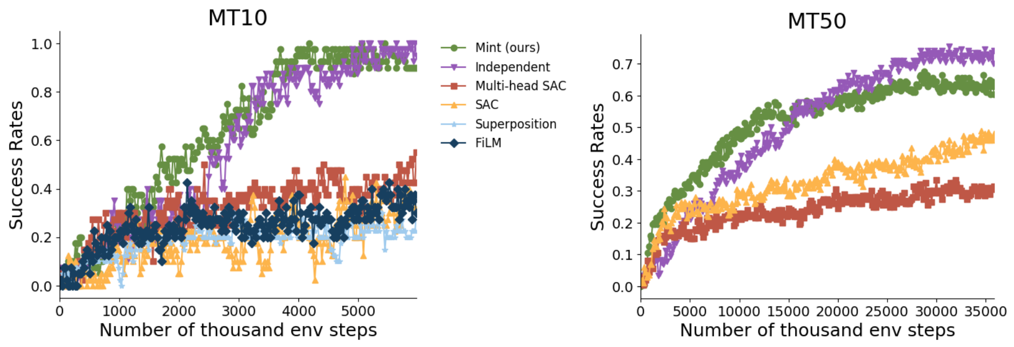

Figure 3: Learning curves on MT10 (left) and MT50 (right). We observe that independent training performs well on both benchmarks. Mint, unlike prior multi-task learning approaches, is able to perform at a similar level to independent training.

methods both in terms of data efficiency and distance to the goal. SAC (concat all fc) also achieves comparable performnce while independent networks fail to learn the task without sufficient amount of data, suggesting that the ability of Mint to represent both joint training and independent networks per task is crucial in multi-task learning and can lead to considerable improvement.

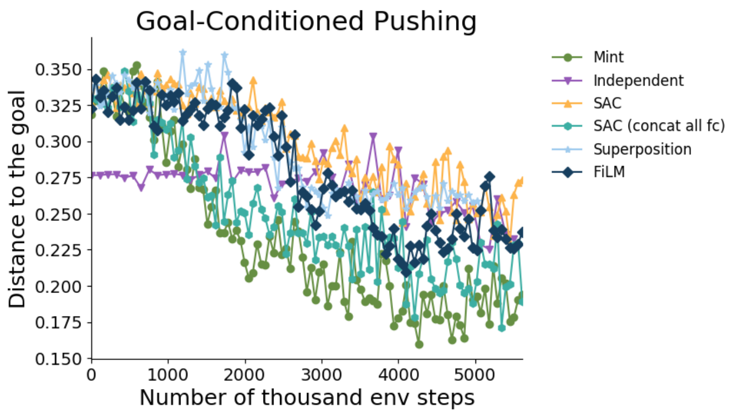

Figure 4: Learning curves on goal-conditioned pushing. Mint is able to outperform all other methods in terms of both distance to the goal and the required number of environment steps.

7 Conclusion

Simultaneous optimization of multiple, potentially unrelated tasks can prove challenging for deep neural networks. Recent multi-task learning architectures attempt to mitigate this issue by providing alternative pathways for information to flow through a neural network for each task. In this paper, we introduce a new multi-task learning module, Mint, which provides theoretical guarantees of universal approximation even for multi-task settings with no shared structure. We conjecture that this property, not shared by similar multi-task architectures, enables Mint to outperform other multi-task approaches on a variety of supervised and reinforcement learning benchmarks. We also observe that Mint is able to match or improve upon the performance of independent training.

While Mint exhibits strong performance gains over previous methods, one potential limitation is that the task matrices may introduces a significant number of parameters, particularly as the number of tasks increases. As discussed, this can be alleviated for problem domains with many tasks, by learning a single neural network that produces the matrices and biases conditioned on the task descriptor. Further, in our experiments, we find that Mint-based networks can outperform prior methods while using comparable or fewer parameters.

In summary, Mint is a simple, yet effective approach for deep multi-task learning. Its implementation requires minimal modifications over standard deep networks. As a result, we expect it to be straightforward for future work to build upon or use Mint for more effective multi-task learning in deep networks.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *J. Mach. Learn. Res.*, 2003.

Rich Caruana. Multitask learning. *Machine Learning*, 1997.

Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno A. Olshausen. Superposition of many models into one. *CoRR*, abs/1902.05522, 2019.

Wojciech M. Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant M. Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS*, 2019.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017. URL `http://arxiv.org/abs/1701.08734`.

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *CoRR*, abs/1711.09874, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 2018.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. 2018.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-end multi-task learning with attention. *CoRR*, abs/1803.10704, 2018.

Mingsheng Long and Jianmin Wang. Learning multiple tasks with deep relationship networks. *arXiv preprint arXiv:1506.02117*, 2, 2015.

Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. *CoRR*, abs/1904.08918, 2019.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *CoRR*, abs/1711.01239, 2017.

Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Sebastian Ruder12, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Sluice networks: Learning what to share between loosely related tasks. *stat*, 1050:23, 2017.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *4th International Conference on Learning Representations, ICLR*, 2016a.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016b.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pp. 527–538, 2018.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Simon Vandenhende, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: Deciding what layers to share. *CoRR*, abs/1904.02920, 2019.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022. ACM, 2007.

Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Regularized hierarchical policies for compositional transfer in robotics. *arXiv preprint arXiv:1906.11228*, 2019.

Yongxin Yang and Timothy M Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Sergey Levine, and Chelsea Finn. Meta-world: A benchmark and evaluation for multi-task and meta-reinforcement learning, 2019. URL `https://github.com/rlworkgroup/metaworld`.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pp. 94–108. Springer, 2014.

A Proof of Theorem 1

Theorem 1. *Let $y^{(l-1)}$ be the activations of layer $l-1$, and let $W^{(l)}, b^{(l)}$ be the weight matrix and bias vector at the l-th layer. We assume that $W^{(l)}$ is a fixed invertible matrix, rather than a matrix of free parameters. Then, we have two results:*

1. *Applying Mint to $y^{(l-1)}$ allows us to express arbitrary affine transformations at layer l for each task.*
2. *Applying FiLM to $y^{(l-1)}$ does not allow us to express arbitrary affine transformations at layer l for each task.*

Proof of 1.

Let $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ be an arbitrary weight matrix and bias vector. Suppose that for task k we wish to represent the affine transformation ${\mathbf{W_k}}^{(l)}y^{(l-1)} + {\mathbf{b_k}}^{(l)}$ at layer l of the network using the combination of Mint and the affine transformation described by applying $W^{(l)}$ multiplicatively and $b^{(l)}$ additively. Concretely, we wish to determine whether there exist a task-specific matrix $M(k)^{(l)}$ and task-specific bias vector $\beta(k)^{(l)}$ such that:

$$ {\mathbf{W_k}}^{(l)}y^{(l-1)} + {\mathbf{b_k}}^{(l)} = W^{(l)}(M(k)^{(l)}y^{(l-1)} + \beta(k)^{(l)}) + b^{(l)} \tag{6}$$

Let $M(k)^{(l)} = {W^{(l)}}^{-1}{\mathbf{W_k}}^{(l)}$ and $\beta(k)^{(l)} = {W^{(l)}}^{-1}({\mathbf{b_k}}^{(l)} - b^{(l)})$. Then, the above equality holds.

Proof of 2.

We wish show a similar existence proof as in Equation 6, except instead of a per-task weight matrix $M(k)^{(l)}$ at each layer, FiLM uses a per-task modulation vector $\beta(k)^{(l)}$. Concretely, we wish to determine whether there exist a task-specific feature modulation vector $v(k)^{(l)}$ and task-specific bias vector $\beta(k)^{(l)}$ such that:

$$ {\mathbf{W_k}}^{(l)}y^{(l-1)} + {\mathbf{b_k}}^{(l)} = W^{(l)}(v(k)^{(l)} \odot y^{(l-1)} + \beta(k)^{(l)}) + b^{(l)} \tag{7}$$

where $\odot$ denotes the Hadamard product. This is equivalent to representing $v(k)$ as a diagonal matrix $V(k)$ whose entries along the diagonal are the entries in $v(k)$. Equation 7 is equivalent to

$$\begin{aligned} {\mathbf{W_k}}^{(l)}y^{(l-1)} + {\mathbf{b_k}}^{(l)} &= W^{(l)}(V(k)^{(l)}y^{(l-1)} + \beta(k)^{(l)}) + b^{(l)} \\ &= W^{(l)}V(k)^{(l)}y^{(l-1)} + W^{(l)}\beta(k)^{(l)} + b^{(l)} \end{aligned}$$

Equating the corresponding terms on both sides of the equation (those which include $y^{(l-1)}$ and those which do not), equality holds only if:

$$\begin{aligned} &W^{(l)}V(k)^{(l)}y^{(l-1)} = {\mathbf{W_k}}^{(l)}y^{(l-1)} \\ &\rightarrow V(k)^{(l)} = {W^{(l)}}^{-1}{\mathbf{W_k}}^{(l)} \\ &\text{Since this equality must hold for all possible } y^{(l-1)} \end{aligned}$$

and $\beta(k)^{(l)} = {W^{(l)}}^{-1}({\mathbf{b_k}}^{(l)} - b^{(l)})$. However, since $V(k)^{(l)}$ and ${\mathbf{W_k}}^{(l)}$ is arbitrary, there is no guarantee that ${W^{(l)}}^{-1}{\mathbf{W_k}}^{(l)}$ will be a diagonal matrix.

B Experiment Details

On the CIFAR-100 multi-task dataset, as suggested in (Rosenbaum et al., 2019), we use a standard convolutional neural network with 3 convolutional layers with 160 3×3 filters followed by 2 fully connected layers with 320 hidden units interleaved with task-specific Mint matrices and biases. We apply max pooling and batch normalization after every convolutional layer.

We use SAC (Haarnoja et al., 2018) algorithm for all RL domains. For Mint, We use 3-layer fully-connected neural network interleaved with per-task Mint matrices and biases at each layer for both the actor and critic. All the other methods use 6-layer fully-connected neural networks for the actor and critic such that they maintain about the same number of total parameters as Mint. The number of hidden units per layer is 200, 160, and 150 for goal-conditioned pushing, MT10, and MT50 respectively for all methods.

On the goal-conditioned pushing experiment, we sample the (x, y) positions of the puck uniformly from the range $[-0.2, 0.6]$ to $[0.2, 0.7]$ and the positions of the goal uniformly from $[-0.2, 0.85]$ to $[0.2, 0.95]$.