Adapting Auxiliary Losses Using Gradient Similarity

Yunshu Du[*,†] **Wojciech M. Czarnecki**[†] **Siddhant M. Jayakumar,**
Razvan Pascanu, Balaji Lakshminarayanan
DeepMind
yunshu.du@wsu.edu, {lejlot, sidmj, razp, balajiln}@google.com

Abstract

One approach to deal with the statistical inefficiency of neural networks is to rely on auxiliary losses that help to build useful representations. However, it is not always trivial to know if an auxiliary task will be helpful for the main task and when it could start hurting. We propose to use cosine similarity between gradients of tasks as an adaptive weight for the auxiliary loss and show that our approach is guaranteed to converge to critical points of the main task. The practical usefulness of our algorithm is demonstrated on Atari games, a popular benchmark for deep reinforcement learning.[3]

1 Introduction

Neural networks are extremely powerful function approximators that have excelled on a wide range of tasks [6, 15, 26, 27, 30]. Despite the state of the art results across domains, they remain data-inefficient and expensive to train. In reinforcement learning (RL), agents typically consume millions of frames of experiences before learning to act in complex environments [3, 26], which not only puts pressure on compute power but also makes particular domains (e.g., robotics) impractical. In supervised learning (e.g., image classification), large deep learning (DL) benchmarks with millions of examples are needed for training [21]. This additional implication of requiring human intervention to label a large dataset can be prohibitively expensive.

Different techniques have been studied for improving data efficiency, from data augmentation [5, 13, 27], transfer learning [17, 28], to lifelong learning [2]. In this work, we focus on a particular setup for transfer learning. We assume that besides the main task, one has access to multiple auxiliary tasks that share some unknown structure with the main task. To improve data efficiency, these additional tasks can be used as auxiliary losses. Only the performance on the main task is of interest, even though the model is trained simultaneously on all these tasks. Any improvement on the auxiliary losses is useful only to the extent that it helps learning features or behaviors for the main task.

Auxiliary tasks have been shown to work well in practice. In image classification, Zhang et al. [33] used unsupervised reconstruction tasks. In RL, the UNREAL framework [12] incorporates unsupervised control tasks in addition to reward prediction learning as auxiliary tasks. Mirowski et al. [14] studied auxiliary tasks in the context of navigation. Papoudakis et al. [18] also explored auxiliary loses for VizDoom domain. However, their success depends on how well aligned the auxiliary losses are with the main task. Knowing this apriori is typically non-trivial and the usefulness of an auxiliary task can change through the course of training. In this work, we explore a simple yet effective approach for measuring the similarity between an auxiliary task and the main task of interest, given the value of the parameters. We show that this measure can be used to decide which auxiliary losses are helpful and for how long.

Notation and problem description Assume we have a main task $\mathcal{T}_{main}$ and an auxiliary task $\mathcal{T}_{aux}$ that induce two losses $\mathcal{L}_{main}$ and $\mathcal{L}_{aux}$. We care only about maximizing performance on $\mathcal{T}_{main}$; $\mathcal{T}_{aux}$ is an auxiliary task which is not of direct interest.[4] We propose to parameterize the solution

[*] Work done during an internship at DeepMind. Other affiliation: Washington State University.

[†] Equal Contribution

[3] We also conduct experiments on multi-task supervised learning using subsets of ImageNet and additional reinforcement learning experiments on gridworlds, see Appendix C.

[4] Note that this is different from multi-objective optimization in which both tasks are of interest.

Work under review at ICLR 2019.

for $\mathcal{T}_{main}$ and $\mathcal{T}_{aux}$ by two neural networks $f(\cdot, \boldsymbol{\theta}, \boldsymbol{\phi}_{main})$ and $g(\cdot, \boldsymbol{\theta}, \boldsymbol{\phi}_{aux})$ such that they share a subset of parameters denoted here by $\boldsymbol{\theta}$. Generally, the auxiliary loss literature proposes to minimize

$$\underset{\boldsymbol{\theta}, \boldsymbol{\phi}_{main}, \boldsymbol{\phi}_{aux}}{\arg\min} \quad \mathcal{L}_{main}(\boldsymbol{\theta}, \boldsymbol{\phi}_{main}) + \lambda \mathcal{L}_{aux}(\boldsymbol{\theta}, \boldsymbol{\phi}_{aux}) \tag{1}$$

under the intuition that modifying $\boldsymbol{\theta}$ to minimize $\mathcal{L}_{aux}$ will improve $\mathcal{L}_{main}$ if the two tasks are sufficiently related. We propose to modulate the weight λ at each learning iteration t by how useful $\mathcal{T}_{aux}$ is for $\mathcal{T}_{main}$ given $\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}_{main}^{(t)}, \boldsymbol{\phi}_{aux}^{(t)}$. That is, at each optimization iteration, we want to efficiently approximate the solution to

$$\underset{\lambda^{(t)}}{\arg\min}\, \mathcal{L}_{main}\left(\boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}}(\mathcal{L}_{main} + \lambda^{(t)} \mathcal{L}_{aux}), \boldsymbol{\phi}_{main}^{(t)} - \alpha \nabla_{\boldsymbol{\phi}_{main}} \mathcal{L}_{main}\right) \tag{2}$$

Note that the input space of $\mathcal{T}_{main}$ and $\mathcal{T}_{aux}$ do not have to match, and in particular, $\mathcal{T}_{aux}$ does not need to be defined for an input of $\mathcal{T}_{main}$ or the other way around.[5] Solving equation 2 is expensive. Instead, we look for a cheap heuristic to approximate $\lambda^{(t)}$ which is better than keeping $\lambda^{(t)}$ constant.

2 Cosine Similarity Between Gradients of Tasks

We propose to use cosine similarity of gradients between tasks as a measure of task similarity and hence for approximating $\lambda^{(t)}$. Consider the example in Figure 1, where the main function that we wish to minimize is $\mathcal{L}_{main} = (\theta - 10)^2$ and the auxiliary function is $\mathcal{L}_{aux} = \theta^2$. When θ is initialized at $\theta = -20$, the gradients of the main and auxiliary functions point in the same direction and the cosine similarity is 1; minimizing the auxiliary loss is beneficial for minimizing the main. However, at a different point, $\theta = 5$, the two gradients point in different directions and the cosine similarity is -1; minimizing the auxiliary loss would hinder minimizing the main loss.

This example suggests a natural strategy for approximating $\lambda^{(t)}$: *minimize the auxiliary loss as long as its gradient has non-negative cosine similarity with the main gradient; otherwise, the auxiliary loss should be ignored.* This follows the well-known intuition that if a vector is in the same half-space as the gradient of a function f, then it is a *decent direction* for f. This reduces our strategy to ask if the gradient of the auxiliary loss is a *descent direction* for the main loss of interest.

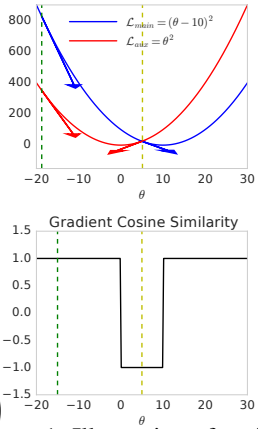


Figure 1: Illustration of cosine similarity between gradients on synthetic loss surfaces.

Proposition 1. *Given any gradient vector field $G(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ and any vector field $V(\boldsymbol{\theta})$ (such as the gradient of another loss function, but could be an arbitrary set of updates), an update rule of the form*

$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\left(G(\boldsymbol{\theta}^{(t)}) + V(\boldsymbol{\theta}^{(t)}) \max\left(0, \cos(G(\boldsymbol{\theta}^{(t)}), V(\boldsymbol{\theta}^{(t)}))\right)\right)$$

converges to the local minimum of $\mathcal{L}$ given small enough $\alpha^{(t)}$.

Proof is provided in Appendix A.1.

Note that the above statement guarantees only the lack of divergence, but not the improvement of convergence. That is, cosine similarity guarantees to drop the "worst-case scenarios" by ignoring the auxiliary loss when it is hurting the main loss, but does not guarantee positive transfer (e.g., the choice of auxiliary loss could be on its own harmful to the main loss). In Appendix B, we show example functions where the main loss's convergence is affected either positively (Figure 5) or negatively (Figure 6) by the auxiliary loss; the main task converges faster (or slower) when a suitable (or unsuitable) auxiliary task is chosen. Nevertheless, the convergence on the main task is guaranteed for our proposed strategy regardless of the choice of auxiliary task, as the proposition shows.

In addition, it is important to note that simply adding an arbitrary vector field does not have the convergence property. For example, use function $V(\boldsymbol{\theta}) = -\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \left[-\frac{\theta_2}{\theta_1^2+\theta_2^2}, \frac{\theta_1}{\theta_1^2+\theta_2^2}\right]^T$ as a two-dimensional case, which leads to an update rule of $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \left[-\frac{\theta_2}{\theta_1^2+\theta_2^2}, \frac{\theta_1}{\theta_1^2+\theta_2^2}\right]^T$. This

[5] In the supervised learning case when the input features are shared, it resembles the *multi-task learning without label correspondences* setting [20].

is a non-conservative vector field which causes the optimizer to follow concentric circles around the origin (see Figure 5d in Appendix B). This is crucial to note for some realistic scenarios where one does not always form a gradient field (e.g., the update rule of the Q-learning algorithm). We provide a few examples on quadratic functions of the proposed approach in Figure 5 to help intuitively understand the kind of scenarios for which our approach could help.

The above proposition refers to losses with the same set of parameters $\boldsymbol{\theta}$, while equation 2 refers to the scenario when each loss has task specific parameters (e.g. $\boldsymbol{\phi}_{main}$ and $\boldsymbol{\phi}_{aux}$). The following proposition extends to this scenario:

Proposition 2. *Given two losses parametrized with $\boldsymbol{\Theta}$ (some of which are shared $\boldsymbol{\theta}$ and some are unique to each loss $\boldsymbol{\phi}_{main}$ and $\boldsymbol{\phi}_{aux}$), learning rule:*

$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \Big(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{main}(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} \mathcal{L}_{aux}(\boldsymbol{\theta}^{(t)}) \max\big(0, \cos(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{main}(\boldsymbol{\theta}^{(t)}), \nabla_{\boldsymbol{\theta}} \mathcal{L}_{aux}(\boldsymbol{\theta}^{(t)}))\big) \Big)$$

$$\boldsymbol{\phi}_{main}^{(t+1)} := \boldsymbol{\phi}_{main}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{\phi}_{main}} \mathcal{L}_{main}(\boldsymbol{\Theta}^{(t)}) \quad \textit{and} \quad \boldsymbol{\phi}_{aux}^{(t+1)} := \boldsymbol{\phi}_{aux}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{\phi}_{aux}} \mathcal{L}_{aux}(\boldsymbol{\Theta}^{(t)})$$

leads to convergence to local minimum of $\mathcal{L}_{main}$ w.r.t. $(\boldsymbol{\theta}, \boldsymbol{\phi}_{main})$ given small enough $\alpha^{(t)}$.

Proof. Comes directly from the previous proposition that $G = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{main}$ and $V = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{aux}$. For any vector fields A, B, C, we have $\langle A, B \rangle \geq 0$ and $\langle C, B \rangle \geq 0$ implies $\langle A + C, B \rangle \geq 0$. □

Analogous guarantees hold for the *unweighted version* of this algorithm, where instead of weighting by $\cos(G, V)$ we use a binary weight $(\text{sign}(\cos(G, V)) + 1)/2$ which is equivalent to using V iff $\cos(G, V) > 0$. When training with mini-batches, accurately estimating $\cos(G, V)$ can be difficult due to mini-batch noises; the unweighted variant only requires $\text{sign}(\cos(G, V)$ which can be estimated more robustly. Hence, we use this variant in our experiments unless otherwise specified. Additionally, note that there is no guarantee that $\mathcal{L}_{aux}$ is optimized. For example, if $\mathcal{L}_{aux} = -\mathcal{L}_{main}$ then $\mathcal{L}_{aux}$ is ignored (see a visualization in Figure 5e).

Despite its simplicity, the proposed update rule can give rise to interesting phenomena. In particular, we can show that the emerging vector field could be non-conservative. We prove that this phenomenon has no negative effects on our proposed algorithm in Appendix A.2.

3 Experiments on Atari Games

In this section, we use the gradient cosine similarity to decide when the auxiliary task is beneficial to the main task. We test our approach in the Atari domain [1] (we also conduct experiments on image classification and RL grid world, see Appendix C). The training procedure is summarized in Algorithm 1 in Appendix C.1. Note that for practical reasons we pick a constant *threshold* for the proposed update rule instead of using 0 as described in Section 2. We use the same convolutional architecture as in previous works [3, 8, 15, 16] and train using the batched actor-critic with V-trace algorithm [3]. Experiment details are described in Appendix C.2.

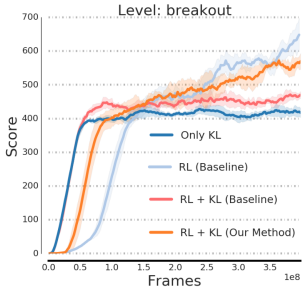


Figure 2: Results on Breakout. We look at the effects of distilling a suboptimal policy as an auxiliary task.

First, we train an agent to play Breakout given a *sub-optimal* solution to the agent. The sub-optimal solution is a distillation of behaviors from a pre-trained Breakout agent (the teacher) with a Kullback-Leibler (KL) loss [11, 22, 29]. Consider the RL loss as the main loss of interest and the KL distillation loss as the auxiliary loss. Figure 2 illustrates our results under four different settings. *RL (Baseline)* trains with only RL loss; *Only KL* shows that solely rely on the auxiliary leads to sub-optimal performance as the agent is purely mimicking a sub-optimal teacher; *RL + KL (baseline)* trains jointly with the auxiliary and RL loss leads to a slightly better, but also sub-optimal performance. While these approaches learn quickly at the beginning, they plateau much lower than the *RL (Baseline)* approach. *RL + KL (Our Method)* is able to take the benefit from both the *RL (Baseline)* and the *RL + KL (baseline)* approaches that it is able to learn quickly at the start and continue to fine-tune with RL loss once the auxiliary loss is zeroed out; the KL penalty is scaled at every time-step by the cosine similarity between the policy gradient loss and the distillation loss, once this falls below a fixed *threshold*, the loss is "turned off" to prevent negative effects to the main loss.

We then consider a multi-task setup to train an agent to play two Atari games, Breakout and Ms. PacMan, as the main task $\mathcal{T}_{main}$. Similarly, we have access to a teacher agent trained on Breakout, from which we distill a teacher policy. At every time step, the auxiliary KL loss $\mathcal{L}_{aux}$ (between the teacher and student policies of Breakout) is added to the RL multi-task loss $\mathcal{L}_{main}$. Intuitively, doing so would result in the agent learning only one of the tasks—the one the teacher knows about, as the gradients from distillation loss would interfere with the policy gradient.

Figure 3 shows that, compared to the baseline *Multitask* and the simple addition of *Multitask RL + Distillation* approaches where the agent learns one task at the expense of the other, our method of scaling the auxiliary loss by gradient cosine similarity is able to compensate for this by learning from the teacher and then turning off the auxiliary distillation; it learns Ms. PacMan without forgetting Breakout. The evolution of the gradient cosine similarity between Breakout and Ms. PacMan provides a meaningful cue for the usefulness of $\mathcal{L}_{aux}$.

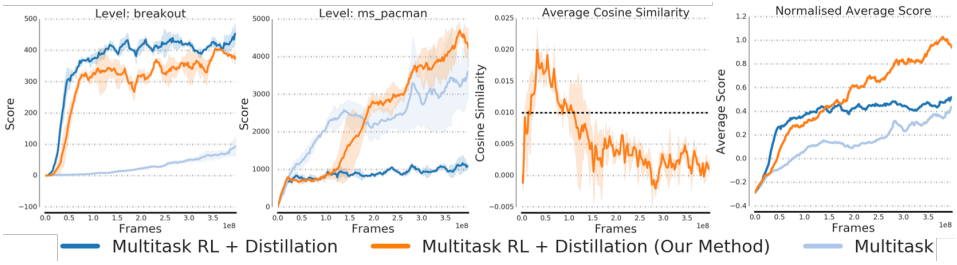


Figure 3: Results on Breakout and Ms. PacMan (averaged over 3 seeds). The two plots to the left show performance on Breakout and Ms. PacMan respectively. The third plot to the left shows how the gradient cosine similarity between the two tasks changes during training. The last plot shows an average score of the multi-task performance (normalized independently for each game based on the best score achieved across all experiments). Our method (orange line) is able to learn both games without forgetting and achieves the best average performance.

4 Discussion

In this work, we explored a simple yet efficient technique to ensure that an auxiliary loss does not hurt the learning on the main task. The proposed approach reduces to applying gradients of the auxiliary task only if they are a descent direction of the main task.

Though we have demonstrated the usefulness of the proposed method in practice, we discuss here a few shortcomings of this method. First, estimating the cosine similarity between the gradients of tasks could be expensive or noisy and that the threshold for turning off the auxiliary is a fixed constant. These could be addressed by calculating a running average of the cosine similarity to get a smoother result and potentially hyper-tune the threshold instead of setting it as a fixed constant. One might argue that our approach would fail in high-dimensional spaces since random vectors in such spaces tend to be orthogonal, so that cosine similarity will be naturally driven to 0. In fact, this is not the case; if two gradients are meant to be co-linear, the noise components cancel each other thus will not affect the cosine similarity estimation. We empirically explore this in Appendix D. Second, the new loss surface might be less smooth. This can be problematic when using optimizers that rely on statistics of the gradients or second order information (e.g. Adam or RMSprop). In these cases, the transition from just the gradient of the main task to the sum of the gradients can affect the statistics of the optimizer in unwanted ways. Lastly, although the proposed approach works well empirically on complex and noisy tasks like Atari games, as discussed in Section 2, it guarantees only that the main task will converge, but not how fast the convergence will be. While removing the worst case scenarios is important, one might care more for data efficiency when using auxiliary losses (i.e., faster convergence). Particularly in Appendix B Figure 6, we construct a counter-example where the proposed update rule slows down learning compared to optimizing the main task alone.

Nevertheless, we have empirically shown the potential of using the proposed hypothesis as a simple yet efficient way of picking a suitable auxiliary task. While we have mostly considered scenarios where the auxiliary task helps initially but hurts later, it would be interesting to explore settings where the auxiliary task hurts initially but helps in the end. Examples of such are annealing β in β-VAE [10] and annealing the confidence penalty in [19].

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[2] Z. Chen and B. Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[3] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[4] A. Goldstein. Cauchy's method of minimization. *Numerische Mathematik*, 4(1):146–150, 1962.

[5] S. Hauberg, O. Freifeld, A. B. L. Larsen, J. Fisher, and L. Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In *Artificial Intelligence and Statistics*, pages 342–350, 2016.

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

[7] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[8] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. Multi-task deep reinforcement learning with PopArt. *arXiv preprint arXiv:1809.04474*, 2018.

[9] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a Nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[10] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. β-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[12] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[14] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In *ICLR*, 2017.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[17] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[18] G. Papoudakis, K. C. Chatzidimitriou, and P. A. Mitkas. Deep reinforcement learning for doom using unsupervised auxiliary tasks. *CoRR*, abs/1807.01960, 2018.

[19] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[20] N. Quadrianto, J. Petterson, T. S. Caetano, A. J. Smola, and S. Vishwanathan. Multitask learning without label correspondences. In *NIPS*, 2010.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[22] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[23] S. Schmitt, J. J. Hudson, A. Zidek, S. Osindero, C. Doersch, W. M. Czarnecki, J. Z. Leibo, H. Kuttler, A. Zisserman, K. Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[28] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 2009.

[29] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. In *NIPS*, 2017.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[31] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[32] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[33] Y. Zhang, K. Lee, and H. Lee. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *ICML*, 2016.

A Propositions and Proofs

A.1 Proof for Proposition 1

Given any gradient vector field $G(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ and any vector field $V(\boldsymbol{\theta})$ (such as gradient of another loss function, but could be arbitrary set of updates), an update rule of the form

$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\Big(G(\boldsymbol{\theta}^{(t)}) + V(\boldsymbol{\theta}^{(t)})\max\big(0, \cos(G(\boldsymbol{\theta}^{(t)}), V(\boldsymbol{\theta}^{(t)}))\big)\Big)$$

converges to the local minimum of $\mathcal{L}$ given small enough $\alpha^{(t)}$.

Proof. Let us denote

$$G^{(t)} := G(\boldsymbol{\theta}^{(t)}) \qquad V^{(t)} := V(\boldsymbol{\theta}^{(t)}) \qquad \nabla\mathcal{L}^{(t)} := \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\Delta\boldsymbol{\theta}^{(t)} := G^{(t)} + V^{(t)}\max(0, \cos(G^{(t)}, V^{(t)})).$$

Our update rule is simply $\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\Delta\boldsymbol{\theta}^{(t)}$ and we have

$$\langle\Delta\boldsymbol{\theta}^{(t)}, \nabla\mathcal{L}^{(t)}\rangle = \langle G^{(t)} + V^{(t)}\max(0, \cos(G^{(t)}, V^{(t)})), \nabla\mathcal{L}^{(t)}\rangle \tag{3}$$
$$= \langle G^{(t)}, \nabla\mathcal{L}^{(t)}\rangle + \langle V^{(t)}\max(0, \cos(G^{(t)}, V^{(t)})), \nabla\mathcal{L}^{(t)}\rangle \tag{4}$$
$$= \|\nabla\mathcal{L}^{(t)}\|^2 + \tfrac{1}{\|V^{(t)}\|\|\nabla\mathcal{L}^{(t)}\|}\max(0, \langle\nabla\mathcal{L}^{(t)}, V^{(t)}\rangle)\langle V^{(t)}, \nabla\mathcal{L}^{(t)}\rangle \geq 0. \tag{5}$$

And it can be 0 if and only if $\|\nabla\mathcal{L}^{(t)}\| = 0$ (since sum of two non-negative terms is zero iff both are zero, and step from (4) to (5) is only possible if this is not true), thus it is 0 only when we are at the critical point of $\mathcal{L}$. Thus the method converges due to convergence of steepest descent methods, see "Cauchy's method of minimization" [4]. □

A.2 Proposition 3 and Proof

Our proposed update rule give rise to an interesting phenomenon that the emerging vector field could be non-conservative, meaning that there exists not a loss function for which it is a gradient. While this might seem problematic (for gradient-descent-based optimizers), it describes only the global structure—typically used optimizers are local in nature [25]. Consequently, in practice one should not expect any negative effects from this phenomenon, as it simply shows that our proposed technique is in fact qualitatively changing the nature of update rules for training.

Proposition 3. *In general, the proposed update rule does not have to create a conservative vector field.*

Proof. Proof comes from a counterexample (Visualized in Figure 4), let us define in 2D space:

$$\mathcal{L}_{main}(\theta_1, \theta_2) = a\theta_1$$
$$\mathcal{L}_{aux}(\theta_1, \theta_2) = \begin{cases} a\theta_1 & \text{if } \theta_1 \in [1,2] \wedge \theta_2 \in [0,1] \\ 0 & \text{therwise} \end{cases}$$

for some fixed $a \neq 0$. Let us now define two paths (parametrized by s) between points $(0,0)$ and $(2,2)$, path A which is a concatenation of a line from $(0,0)$ to $(0,2)$ (we call it U, since it goes up) and line from $(0,2)$ to $(2,2)$ (which we call R as it goes right), and path B which first goes right and then up. Let $V_{\cos}$ denote the update rule we follow, then:

$$\int_A V_{\cos}ds = \int_A \nabla\mathcal{L}_{\text{main}}ds = \int_U \nabla\mathcal{L}_{\text{main}}ds + \int_R \nabla\mathcal{L}_{\text{main}}ds = \int_R \nabla\mathcal{L}_{\text{main}}ds = 2a$$

At the same time, since gradient of $\mathcal{L}_{main}$ is conservative by definition:

$$\int_B V_{\cos}ds = \int_B \nabla\mathcal{L}_{\text{main}}ds + \int_C \nabla\mathcal{L}_{\text{aux}}ds = \int_A \nabla\mathcal{L}_{\text{main}}ds + \int_C \nabla\mathcal{L}_{\text{aux}}ds = 2a + \int_C \nabla\mathcal{L}_{\text{aux}}ds = 3a$$

where C is a part of B that goes through $[1,2]\times[0,1]$. We conclude that $\int_A V_{\cos}ds \neq \int_B V_{\cos}ds$, so our vector field is not path invariant, thus by Green's Theorem it is not conservative, which concludes the proof. □

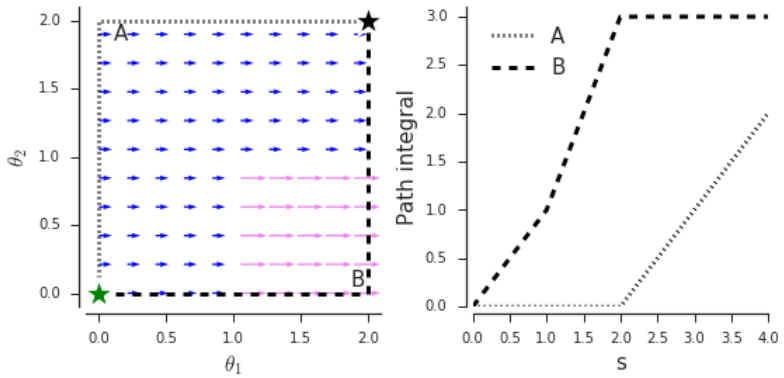


Figure 4: Visualization of the counterexample from Proposition 3. Stars denote starting (green) and end (black) points. Dotted and dashed lines correspond to path A and B respectively. Blue arrows represent gradient vector field of the main loss, violet arrows are the merged vector field.

B Examples of Positive and Negative Effect of Auxiliary task on Main task

In Figure 5 we show examples where our method of using cosine similarity of gradients as a mixing strategy that leads to faster convergence. Consider optimizing a main loss function $L_1(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$ via utilizing an auxiliary loss $L_2(\theta_1, \theta_2) = (\theta_1 - 1)^2 + (\theta_2 - 1)^2$. Their gradients are visualized in Figure 5a and in this case we expect ∇L_2 to be helpful for ∇L_1 since they are positively aligned. We use steepest descent method and define convergence time as the number of steps needed to get below 0.1 loss of L_1. Figure 5b shows that simple addition of ∇L_1 and ∇L_2 slows down the convergence, while our proposed scaling strategy (Figure 5c) leads to faster learning. Similarly when considering a non-conservative vector space $V(\theta_1, \theta_2) = [-\frac{\theta_2}{\theta_1^2+\theta_2^2} - 2\theta_1, \frac{\theta_1}{\theta_1^2+\theta_2^2} - 2\theta_2]$, simple mixing leads to divergence (Figure 5d) while our method still works well (Figure 5e).

We also discuss here a few potential issues of the proposed method. First, the method depends on being able to compute cosine between gradients. However, we rarely are able to compute exact gradients when a high variance estimator such as a deep neural networks (e.g., mini-batch noises in supervised learning or Monte Carlo estimations in RL). Consequently, estimating the cosine similarity might require additional tricks such as to keep moving averages of the estimation. Second, adding additional task gradient in selected subset of iterates can lead to very bumpy surface from the perspective of the optimizer, which leads to tracking a higher order derivatives of the gradient statistics thus less efficient and could hinders the optimization process. We construct one such function in Figure 6 as an illustration.

C More Experiments

We present in this section experiment details on Atari tasks, with additional experiments conducted in two more tasks: a supervised image classification task in the ImageNet dataset and an RL task in the maze domain.

C.1 Pseudocode

Algorithms 1 describes the *unweighted* variant of our method and Algorithm 2 describes the *weighted* variant of our method. See Section 2 for a detailed discussions.

Algorithm 1 Unweighted variant our method

Initialize shared parameters $\boldsymbol{\theta}$ and task specific parameters $\boldsymbol{\phi}_{main}, \boldsymbol{\phi}_{aux}$ randomly.
for iter $= 1 :$ max_iter **do**
  Compute $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}, \nabla_{\boldsymbol{\phi}_{main}}\mathcal{L}_{main}, \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}, \nabla_{\boldsymbol{\phi}_{aux}}\mathcal{L}_{aux}$.
  Update $\boldsymbol{\phi}_{main}$ and $\boldsymbol{\phi}_{aux}$ using corresponding gradients.
  if $\cos(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}, \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}) \geq$ threshold **then**
    Update $\boldsymbol{\theta}$ using $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main} + \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}$
  else
    Update $\boldsymbol{\theta}$ using $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}$

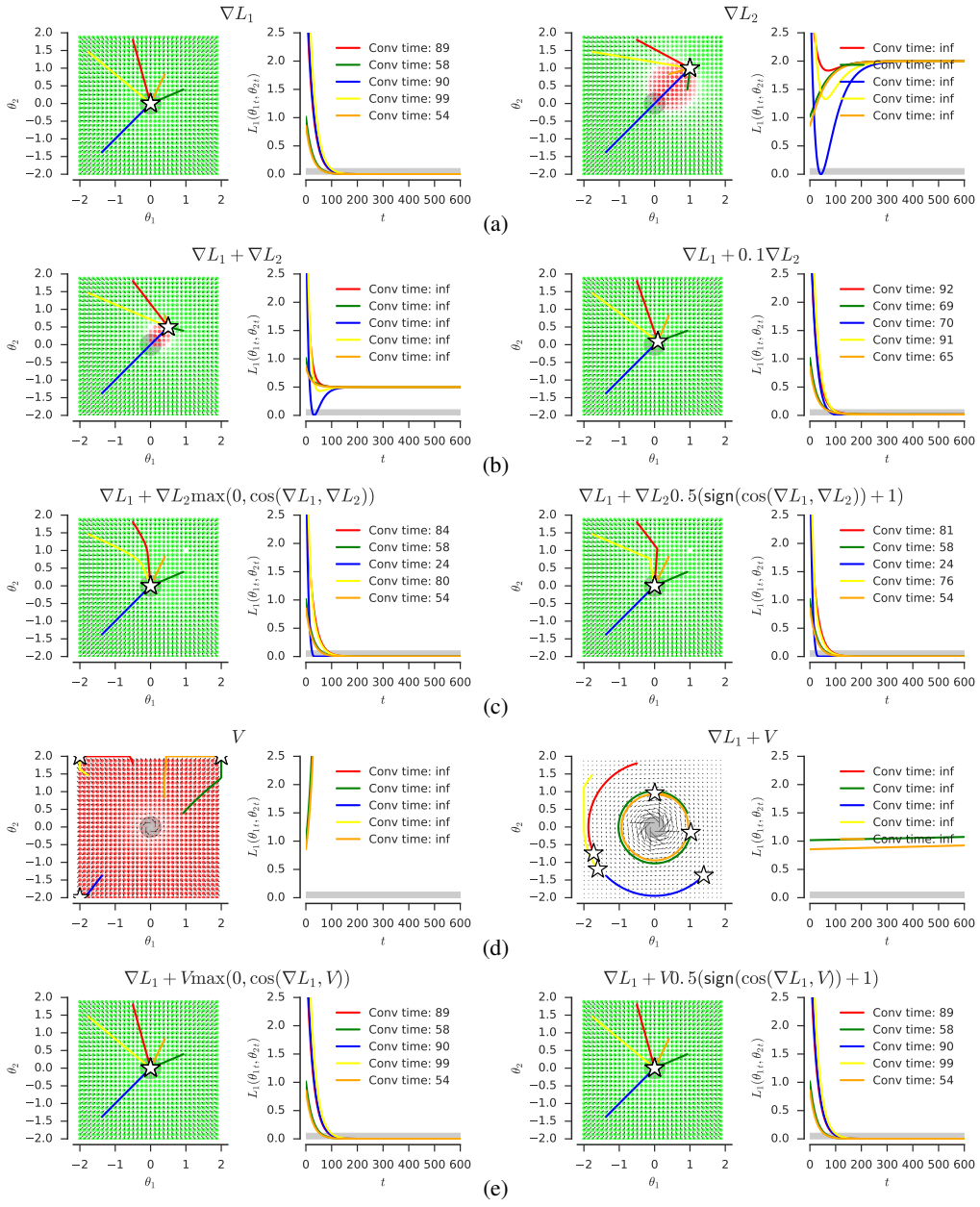


Figure 5: Positive example optimization for $L_1(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$, $L_2(\theta_1, \theta_2) = (\theta_1 - 1)^2 + (\theta_2 - 1)^2$ and $V(\theta_1, \theta_2) = [-\frac{\theta_2}{\theta_1^2+\theta_2^2} - 2\theta_1, \frac{\theta_1}{\theta_1^2+\theta_2^2} - 2\theta_2]$ where the proposed method speeds up the process (compared on all runs). Each colored trajectory represents one optimization run with random initial position. Star represents the convergence point. All experiments use steepest descent method and run 600 iterations with constant step size of 0.01. Convergence time is defined as number of steps needed to get below 0.1 loss of L_1 (gray region). Color of each point represents its alignment with ∇L_1 (green—positive alignment, red—negative alignment, white—directions are perpendicular). In this example L_2 is helpful for L_1 as it reinforces good descent directions in most of the space. However, simple mixing is actually slowing optimization down (or makes it fail completely, Figure 5b.), while the proposed methods (weighted and unweighted variants) converge faster (Figure 5c). When using non-conservative vector field V one obtains lack of convergence (cyclic behaviour, Figure 5d), while the proposed merging still works well (Figure 5e).

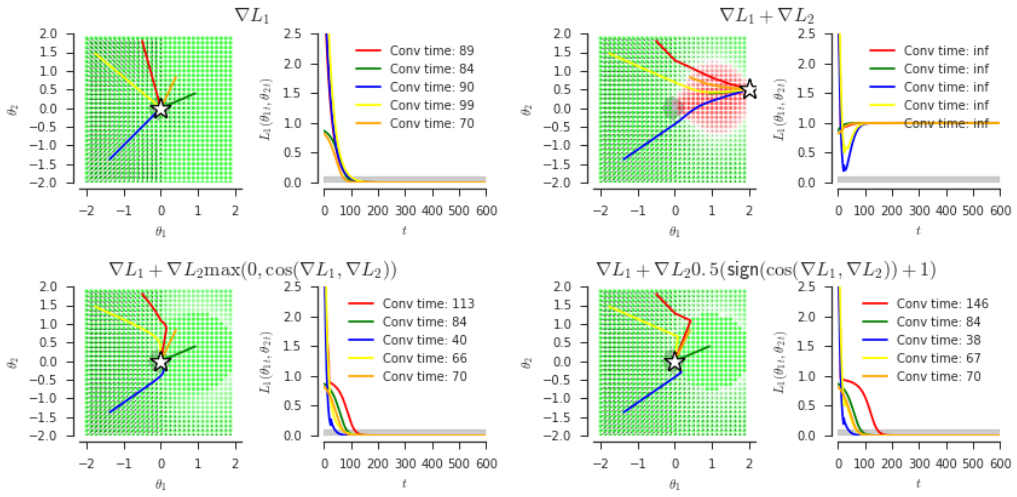


Figure 6: Negative example optimization for $L_1(\theta) = (\theta_1 < 0)(\theta_1^2+\theta_2^2)+(\theta_1 > 0)\Big(1-\exp\big(-2(\theta_1^2+\theta_2^2)\big)\Big)$ and $L_2(\theta) = (\theta_1-2)^2+(\theta_2-0.5)^2$ where the proposed method slows down the process (compared on red runs). For the ease of presentation, we choose L_1, which is non-differentiable/smooth when $\theta = 0$. But one can create any smooth functions with analogous properties. The core idea is, when there exists a flat region on the loss surface, the auxiliary lost tends to push the iterates to this region. Even though this move still decreases the loss (i.e., convergence is guaranteed), the optimization process will be slowed down.

Algorithm 2 Weighted variant our method

1: Initialize shared parameters $\boldsymbol{\theta}$ and task specific parameters ϕ_{main}, ϕ_{aux} randomly.
2: **for** iter $= 1 :$ max_iter **do**
3: Compute $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}, \nabla_{\phi_{main}}\mathcal{L}_{main}, \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}, \nabla_{\phi_{aux}}\mathcal{L}_{aux}$.
4: Update ϕ_{main} and ϕ_{aux} using corresponding gradients.
5: Update $\boldsymbol{\theta}$ using $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main} + \cos(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{main}, \nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux})\nabla_{\boldsymbol{\theta}}\mathcal{L}_{aux}$

C.2 Experiments on Atari Games

All experiments in the Atari tasks use the same convolutional architecture as was done in previous works [3, 8, 15, 16] and train with batched actor-critic with the V-trace algorithm [3]. We use a learning rate of 0.0006 and an entropy cost of 0.01 for all experiments, with a batch size of 32 and 200 parallel actors. We follow the unweighted variant of our method, described in Algorithm 1.

The cosine similarity is computed on a per-layer basis then take the averaged number. We additionally use a moving average of the cosine similarity computation over time ($0.999c^{(t-1)} + 0.001c^{(t)}$) to ensure there are no sudden spikes in the weighting (due to noisy gradients). For the single game experiment, Breakout, we use 0.02 as the threshold for the cosine similarity. For the multi-game experiment, Breakout and Ms PacMan, we use 0.01 as the threshold.

C.3 Experiments on Image Classification Tasks

We consider a classification problem on ImageNet [21] and design a simple multi-task binary classification task to test our hypothesis that *a high cosine similarity indicates that the auxiliary task will be more helpful to the main task (and vice versa)*. We take a pair of classes from ImageNet, refer to these as class A and class B, and refer to all the other 998 classes in ImageNet as the $background$. Our tasks $\mathcal{T}_{main}$ and $\mathcal{T}_{aux}$ are then formed as a multi-task binary classification of *if an image is class A (otherwise $background$)* and *if an image is class B (otherwise $background$)* respectively.

Identifying near and far classes in ImageNet Ideally, we want to pick groups of class pairs that reflect *near* or *far* distance, for the purpose of providing a baseline of helpfulness of the auxiliary to the main task. Therefore, we use two distance measures, *lowest common ancestor (LCA)* in the

ImageNet label hierarchy and *Frechet Inception Distance (FID)* [9] between image embedding, to serve as a ground truth of class similarity for selecting class pair A and B.

ImageNet follows a tree hierarchy where each class is a leaf node. We define the distance between a pair of classes as at which tree level their LCA is found. In particular, there are 19 levels in the class tree, each leaf node (i.e. class) is considered to be level 0 while the root node is considered to be level 19. We perform bottom-up search for one pair of random sampled classes and find their LCA node. The class distance is then defined as the level number of this node. For example, class 871 ("trimaran") and class 484 ("catamaran") has class distance 1 because their LCA is one level up. FID is used as a second measure of similarity. We obtain the image embedding of a pair of classes using the penultimate layer of a pre-trained ResNetV2-50 model [7], then compute the embedding distance using FID, which is defined in [9] as:

$$\text{FID} = d^2\big((m_1, C_1), (m_2, C_2)\big) = \|m_1 - m_2\|_2^2 + \text{Tr}\big(C_1 + C_2 - 2(C_1 C_2)^{1/2}\big). \tag{6}$$

where m_k, C_k denote the mean and covariance of the embeddings from class k.

We randomly sample 50 pairs of classes for each level of $LCA = \{1, 2, 3, 4, 16, 17, 18, 19\}$ (400 pair of classes in total) and compute their FID. Figure 7 shows a plot of LCA (x-axis) verses FID (y-axis) over our sampled class pairs, showing that LCA and FID are (loosely) correlated and that they reflect human intuition of task similarity for some pairs. For example, *trimaran* and *catamaran* (bottom-left) are similar both visually and conceptually, whereas *rock python* and *traffic light* (top-right) are dissimilar both visually and conceptually. However, there are contrary examples where LCA disagrees with FID; *monkey pinscher* and *doberman pinscher* (top-left) are visually dissimilar but conceptually similar, whereas *bubble* and *sundial* (bottom-right) are visually similar but conceptually dissimilar.

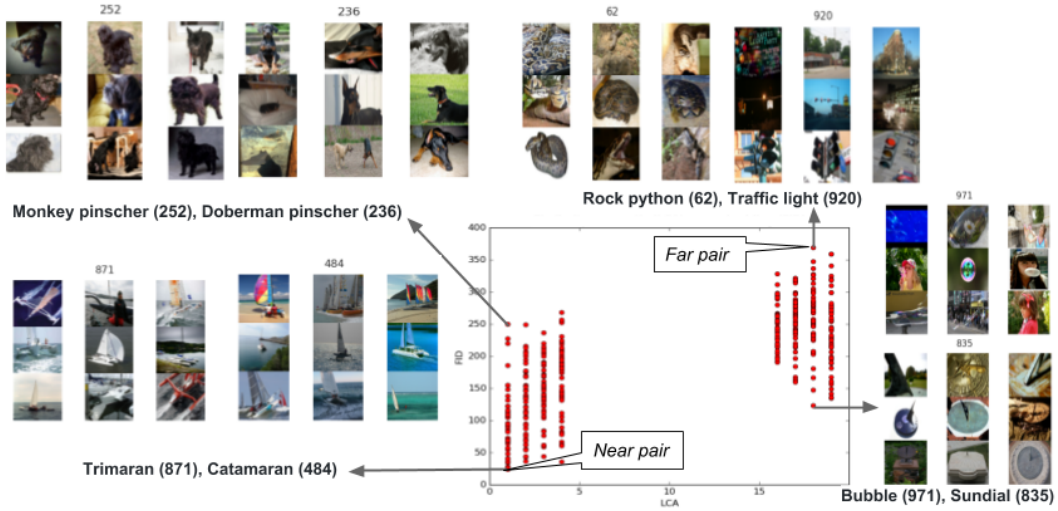
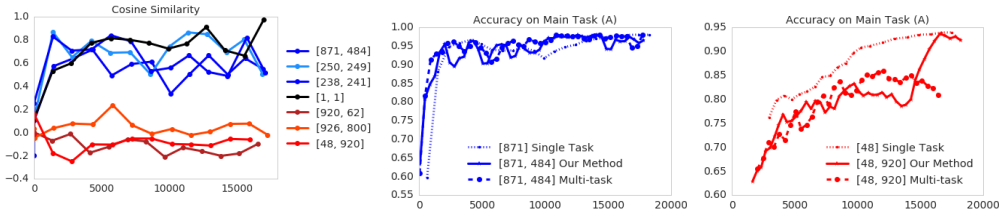


Figure 7: LCA (x-axis) versus FID (y-axis) as a ground truth for class similarity.

Based on these measures, we picked three pair of classes for *near*, class 871 vs. 484, 250 vs. 249 and 238 vs. 241; for *far*, class 920 vs. 62, 926 vs. 800, 48 vs. 920. From each pair we build $\mathcal{L}_{main}$ and $\mathcal{L}_{aux}$ by solving the binary classification problem.

Experiment details We use a modified version of ResNetV2-18 [7] as the training model for this experiment. All parameters in the convolutional layers are shared (denote as $\boldsymbol{\theta}$), followed by task specific parameters $\boldsymbol{\phi}_{main}$ and $\boldsymbol{\phi}_{aux}$. We follow Algorithm 1 and choose $\texttt{threshold} = 0$. First, we use a multi-task learning setup to minimize $\mathcal{L}_{main} + \mathcal{L}_{aux}$ and measure cosine similarity on $\boldsymbol{\theta}$ through the course of training. Figure 8a shows that cosine similarity is higher for close pairs (blue lines) and lower for far pairs (red lines); class 1 vs. 1 serves as a baseline that the same class should have the highest cosine similarity. Next, we compare training using *Single Task*, *Multi-Task*, and *Our Method*. Figure 8b shows that on a near pair of class, all three methods perform similarly. However, as Figure 8c shows, multi-task learning leads to poorer performance than single-task learning on $main$ for a *far* pair of class due to potential negative transfer, whereas our method uses gradient cosine similarity as the mixing strategy leads to improved performance.



(a) Cosine similarities on near pairs (blue) and far pairs (red).

(b) Near pair: 871 vs. 484

(c) Far pair: 48 vs. 920

Figure 8: Multi-task learning setup on ImageNet class pairs. (a): gradient cosine similarity is higher for near pairs and lower for far pairs. (b) and (c): testing accuracy on single task (dotted), naive multi-task (dashed) and our method (solid).

C.4 Experiments on Reinforcement Learning Grid Worlds

We consider a typical RL problem where one aims at finding a policy π that maximizes sum of future discounted rewards, $\mathbb{E}_\pi[\sum_{t'=1}^{N} \gamma^{t'-1} r_{t'}]$, in some partially observable Markov decision process (POMDP). There have been many techniques proposed to solve this optimization problem, from classical policy gradient [32] or Q-Learning [31] to more modern Proximal Policy Optimization [24] or V-Trace [3]. Inherently, these techniques are usually very data inefficient due to the complexity of the problem being solved. One way to address this issue is to use transfer learning, for example from pre-trained policies [22]. However, in the scenario of which a teacher policy is not available for the target task, one can train policies in other tasks that share enough similarities to hope for positive transfer. One way of exploiting this extra information is to use behavioural cloning, or distillation [11, 22], to guide the main RL algorithm in its initial learning phase [23]. In practice, it might be difficult to find a suitable strategy that combines the two losses and/or smoothly transition between them. Typically, the teacher policy can be treated as an auxiliary loss [23] or a prior [29] with a fixed mixing coefficient. However, these techniques become unsound if the teacher policy is helpful only in specific states, but leads to negative transfer in other states.

We propose a simple RL experiment to show that our method is capable of finding the strategy of combining auxiliary loss and the main loss. We define a distribution over 15×15 mazes, where an agent observes its surrounding (for up to 4 pixels away) and can move in 4 directions (with 10% transition noise). We randomly place two types of positive rewards, $+5$ and $+10$ points, both terminating an episode. We also place negative rewards (both terminating and non-terminating) to make problem harder. In order to guarantee a finite length of episodes, we add fixed probability of 0.01 of transitioning to a non-rewarding terminal state.

We first train a Q-learning agent on the maze and use it as the *teacher* policy π^{Q}. Then, we create a main task to which there is a possible positive transfer—an environment that keeps the same maze layout, but removes the $+10$ rewards (and corresponding states are no longer terminating). Consequently we have two tasks, $\mathcal{T}_{aux}$ where we have a strong teacher policy, and $\mathcal{T}_{main}$, the main task of interest. We sample $1,000$ such environment pairs and report expected returns obtained (100 evaluation episodes at each evaluation point) using various training regimes. See Figure 9 for a visualization of the task and an example solution.

One can use any RL method to solve $\mathcal{T}_{main}$ and learn π, here we use episode-level policy gradient [32] with value function as a baseline method. Policies are parameterized as logits $\boldsymbol{\theta}$ of $\pi(a|s) = \frac{\exp(\theta_{s,a})}{\sum_b \exp(\theta_{s,b})} \in [0, 1]$. Baselines are parameterized as $B_s \in \mathbb{R}$. We train with fixed learning rate of $\alpha = 0.01$, discount factor $\gamma = 0.95$ and $10,000$ training steps. Under this setup, the update rule for each sequence $\tau = \big((s_1, a_1, r_1), \dots (s_N, a_N, r_N)\big)$ is thus given by

$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} \log \pi(a_{t'}|s_{t'}) \left[\sum_{i=0}^{N-t'} r_{t'+i} - B_{s_{t'}} \right] = \alpha G^{(t)}, \quad \Delta B_{s_{t'}} = -\alpha \nabla_{B_{s_{t'}}} (B_{s_{t'}} - \sum_{i=0}^{N-t'} r_{t'+i})^2$$

This baseline method achieves a score of slightly above **1** point after $10,000$ steps.

To leverage teacher policies for $\mathcal{T}_{aux}$, we define the auxiliary loss to be a distillation loss, which is a per-state cross-entropy between the teacher's and student's distributions over actions. First, we test using solely distillation loss while sampling trajectories from the student. We recover a subset of

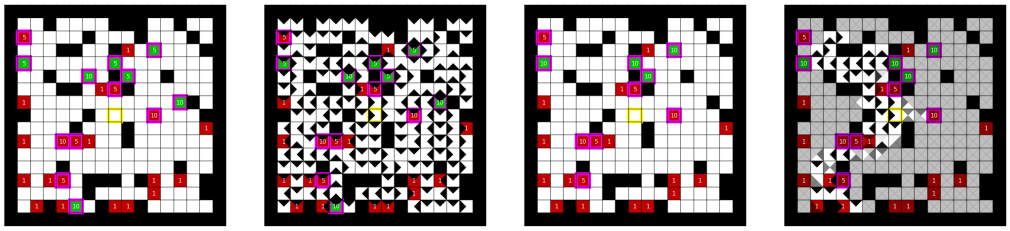


Figure 9: **Left most**: Initial task $\mathcal{T}_{main}$, yellow border denotes starting point and violet ones terminating states. Red states are penalizing with the value in the box while the green ones provide positive reward. **Middle Left**: Solution found by a single run of Q-learning with uniform exploration policy. **Middle Right**: Transformed task $\mathcal{T}_{aux}$. **Right most**: Solution found by gradient cosine similarity driven distillation with policy gradient.

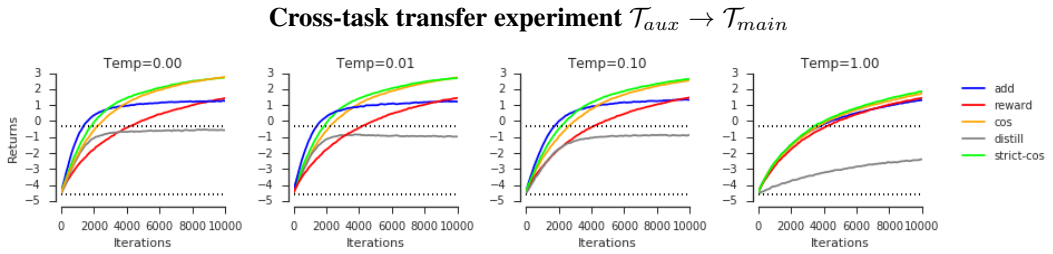


Figure 10: Expected learning curves for cross-environment distillation experiments, averaged across $1,000$ partially observable grid worlds. Teacher's policy is based on Q-Learning. Its performance on a new environment (with modified positive rewards) is represented by the top dotted line (bottom one represents random policy score). Each column represents different temperature applied to teacher policy. 0 temperature refer to the original deterministic greedy policy given by Q-Learning. We report five methods: **reward** using just policy gradient in the new task, **distill** using just distillation cost towards teacher, **add** adding the two above, **cos** using weighted variant in Algorithm 2, **strict cos** using unweighted variant in Algorithm 1.

teacher's behaviours and end up with **0** point—an expected negative transfer as the teacher is guiding us to states that are no longer rewarding.

Then, we test simply adding gradients estimated by policy gradient and distillation. The update rule is given by:

$$\Delta\boldsymbol{\theta} = \alpha \left[G^{(t)} - \nabla_{\boldsymbol{\theta}} \mathrm{H}^{\times}(\pi^{\mathrm{Q}}(\cdot|s_{t'}) \| \pi(\cdot|s_{t'})) \right] = \alpha [G^{(t)} + \sum_{a} \pi^{\mathrm{Q}}(a|s_{t'}) \nabla_{\boldsymbol{\theta}} \log \pi(a|s_{t'})],$$

where $V^{(t)} = \sum_a \pi^{\mathrm{Q}}(a|s_{t'}) \nabla_{\boldsymbol{\theta}} \log \pi(a|s_{t'})$ and $\mathrm{H}^{\times}(p, q) = -\sum_k p_k \log q_k$ is the cross entropy. The result policy learns quickly but saturates at a return of **1** point, showing limited positive transfer.

However, if we use the proposed gradient cosine similarity with the following update rule:

$$\Delta\boldsymbol{\theta} = \alpha \left[G^{(t)} + V^{(t)} \big(2 \cdot \mathrm{sign}(\cos(G^{(t)}, V^{(t)})) - 1 \big) \right].$$

we get a significant performance boost; learned policies reach baseline performance after just one third of steps and on average obtain **3** points after $10,000$ steps.[6] See Figure 10 for all of our results.

Our experiments show that gradient cosine similarity between tasks allows one to use knowledge from other related tasks in an automatic fashion. Agent is simply ignoring teacher signal when it disagrees with policy gradient estimator. When they agree in the action to reinforce, teacher's logits are used for better replication of useful policies. In particular, in Figure 11 we present experiments of transferring between the same task $\mathcal{T}_{main}$. We see that using cosine similarity under-performed

[6] We compute cosine similarity between a distillation gradient and a single sample of the policy gradient estimator, meaning that we are using a high variance estimate of the similarity. For larger experiments one would need to compute running means for reliable statistics, such as what was done in our Atari experiments in Appendix C.2.

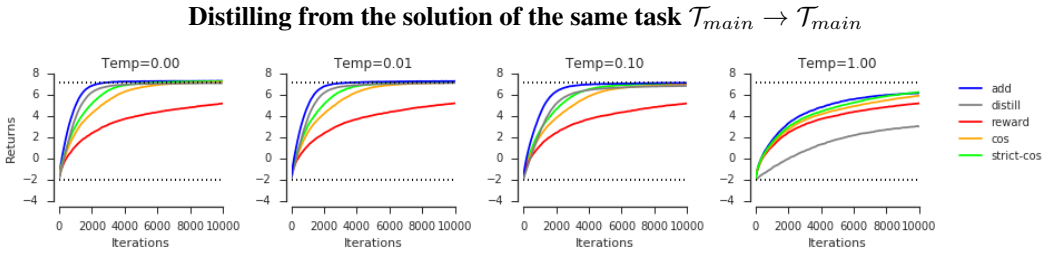


Figure 11: Experiments show scenarios when the teacher is perfect. The optimal behavior is thus to trust it everywhere. We report five methods: **reward** using just policy gradient in the new task, **distill** using just distillation cost towards teacher, **add** adding the two above, **cos** using weighted variant in Algorithm 2, **strict cos** using unweighted variant in Algorithm 1.

that of simply adding the two losses. This is expected as the noise in the gradients make it hard to measure if the two tasks are a good fit or not.

D Cosine similarity in high dimensions

Intuitively, our method could fail in high dimensional spaces since random vectors in such spaces tend to be orthogonal, so that cosine similarity will be naturally driven to 0. We explore here empirically that, in fact, the opposite happens. In Figure 12 we show that when two gradients are co-linear (e.g., when the main gradient and the auxiliary gradient are aligned), the noise components cancel each other thus the cosine similarity remains high in high dimensions.

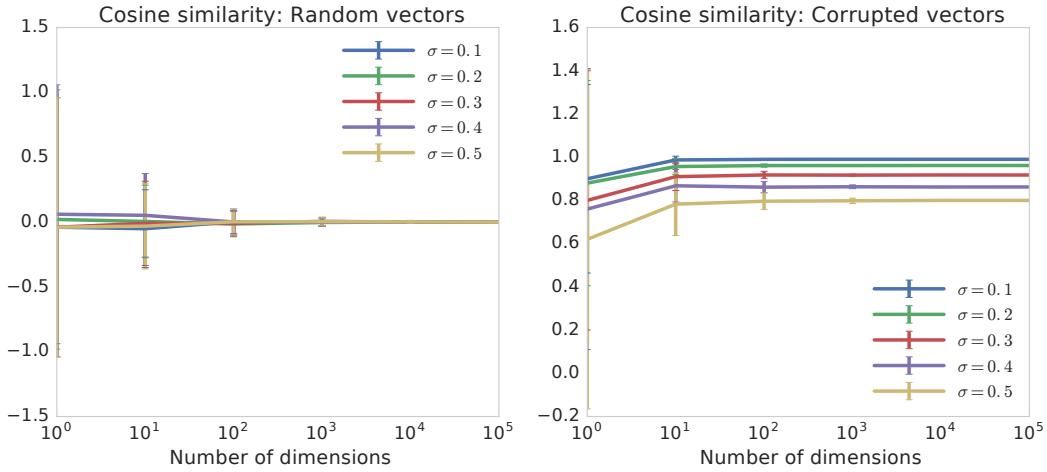


Figure 12: Cosine similarity as a function of dimensions. On the left, we generate two random vectors θ_1 and θ_2 from a Gaussian distribution with zero mean and variance σ^2 and as expected, the cosine similarity drops to zero very quickly as the number of dimensions increases. On the right, we mimic a scenario where the true gradients of the main and auxiliary are aligned, however we observe only *corrupted* noisy gradients which are noisy copies of the true underlying vector; we generate $\mu \sim \mathcal{N}(0, I_d)$ and generate $\theta_1 \sim \mathcal{N}(\mu, \sigma I_d)$ and $\theta_2 \sim \mathcal{N}(\mu, \sigma I_d)$. In this case, cosine similarity is larger in higher dimensions (as the inner product of the corruption noise goes to zero).