Exploring Task Affinities through NTK Alignment and Early Training Dynamics in Multi-Task Learning

Yoann Morello
Data Analytics Laboratory
Vrije Universiteit Brussel
yoann.morello@vub.be

Emilie Grégoire
Data Analytics Laboratory
Vrije Universiteit Brussel
emilie.gregoire@vub.be

Sam Verboven
Data Analytics Laboratory
Vrije Universiteit Brussel
sam.verboven@vub.be

Abstract

Multi-task learning (MTL) aims to leverage shared representations among tasks to improve generalization and training efficiency. However, the challenge of negative transfer between tasks remains a significant obstacle. In this work, we explore modifications to gradient-based measures for task similarity to identify effective task groupings early in training. We highlight key connections between existing measures through the Neural Tangent Kernel (NTK). Our analysis reveals that computing these measures during the initial training stages, averaged over multiple runs, provides a robust estimation of task affinities. We demonstrate the method's effectiveness on synthetic data, capturing both linear and non-linear relationships, and suggest its potential applicability to more complex datasets.

1 Introduction

Multi-task learning (MTL) [Caruana, 1997] aims to enhance generalization and efficiency by training multiple tasks simultaneously using a shared representation. Although this strategy can reduce data requirements and improve training efficiency, it often suffers from *negative transfer* [Standley et al., 2020], wherein gradients from different tasks interfere, leading to worse performance compared to training tasks independently. In hard-parameter-sharing, which is the focus of this work, a shared representation is learned among the tasks. Given such an architecture and a set of tasks to be learned, an important question arises: how should we partition these tasks across models to maximize positive transfer? This problem has received significant attention in recent research [Fifty et al., 2021, Standley et al., 2020, Sherif et al., 2024, Wang et al., 2024].

A key aspect of task partitioning is the definition of an affinity measure between tasks, which helps determine the task groupings that yield the greatest transfer benefits. Existing heuristic-based approaches for defining such measures include expert opinion [Zhang and Yang, 2021], meta-learning techniques [Song et al., 2022], and the use of data maps that track the confidence and variability of predictions for each data point during training [Sherif et al., 2024]. More mathematically grounded approaches, such as gradient-based methods [Fifty et al., 2021, Wang et al., 2024], have also been proposed. However, these methods come with a significant computational cost, as they require training a full multi-task learning (MTL) model while periodically interrupting the training to compute a gradient step for various combinations of losses across subsets of tasks.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

This work illuminates the theoretical connections between existing task-affinity measures (Section 2) and aims to leverage these insights to simplify their computation, thereby reducing computational demands. Two key simplification strategies are proposed. The first involves adopting specific linear approximations, categorized into two types: approximations based on linearity in the weights (Section 2) and approximations based on linearity in the input (Section 3). The second strategy explores the possibility of approximating the average task-affinity measure across the full learning trajectory by averaging the measure over the early stages of training, using multiple stochastic initializations of the model. Experimental analyses of the proposed methods are presented in Section 4.

The key contributions of our work are summarized below:

- We highlight the connections between existing gradient-based methods for measuring task-affinities works through the use of the Neural Tangent Kernel (NTK).
- Based on strong assumptions, we derive a simplified version of existing measures that can be computed without training the model. We apply this architecture-dependent measure to simple models and show it provides meaningful insights when applied to CIFAR-10.
- We provide some empirical analysis of the alignment measures and highlight their stochastic behavior during early training.
- Based on theoretical and empirical insights, we propose to average the variables used to compute the task-alignment over different initializations and show this yields an effective way to capture non-linear relationships in data.

2 Relating existing measures of task affinity through the NTK

In this section, we analyze the measures defined in Fifty et al. [2021], Wang et al. [2024] and Paul et al. [2021]. We demonstrate that these measures share significant similarities and show that the unnormalized alignment between the off-diagonal blocks of the NTK (Appendix A1) and the corresponding off-diagonal matrices of a deviation tensor, G — which we introduce along the way — are key quantities in the formulas derived for task similarities.

2.1 Background

We start by introducing two important gradient-based task-affinity measures from the literature.

Firstly, Fifty et al. [2021] propose to quantify the affinity between a task$a$ and a task b at time step t using the asymmetric measure $Z^t_{a\mapsto b}$ defined as,

$$Z^t_{a\mapsto b} = 1 - \frac{L_b(X;\phi_a^{t+1};\theta_b^t)}{L_b(X;\phi^t;\theta_b^t)} \approx \eta \frac{(\nabla_\phi L_b)^\top \nabla_\phi L_a}{L_b(\phi^t,\theta_b^t)}, \tag{1}$$

where L_a and L_b represent the losses for tasks a and b respectively, ϕ_i^{t+1} represents the updated shared parameters after a gradient step with respect to task i, θ_i corresponds to the task-i-specific parameters, and $\nabla_\phi L_i$ represent the gradients of the losses for tasks i with respect to the shared parameters ϕ (with $i \in \{a, b\}$). The approximation results from a first-order Taylor expansion on a batch X (as shown in Appendix A1).

Next,Wang et al. [2024] introduced another measure, $S^t_{a\mapsto b}$,

$$S^t_{a\mapsto b} = 1 - \frac{L_b(\phi^{t+1}_{\{a,b\}},\theta_b^{t+1})}{L_b(\phi^{t+1}_{\{b\}},\theta_b^{t+1})}, \tag{2}$$

as an enhancement of $Z^t_{a\mapsto b}$, resolving the need for the strong convexity assumption used by Fifty et al. [2021].

2.2 Relating existing measures

Based on a simple assumption below, we show that the task-affinity measures defined by equations (1) and (2), are intrinsically very similar. Specifically, we make the following assumption:

Assumption 1. *During a full epoch of training, the network weights, ϕ and θ, remain constant and the learning rate η is small.*

This assumption simplifies the analysis while retaining essential characteristics of most training dynamics [Elkabetz and Cohen, 2021] With assumption 1, one finds that:

$$S^t_{a\to b} \approx \eta \frac{(\nabla_{\phi,\theta_b} L_b)^\top \nabla_{\phi,\theta_a} L_a}{L_b(\phi^t, \theta^t_b)} = \eta \frac{(\nabla_\phi L_b)^\top \nabla_\phi L_a + (\nabla_{\theta_b} L_b)^\top \nabla_{\theta_a} L_a}{L_b(\phi^t, \theta^t_b)} \approx Z^t_{a\mapsto b}, \tag{3}$$

which demonstrates that, under 1, the two popular task-affinity measures are approximately the same, raising questions about the empirical differences claimed by Wang et al. [2024]. We can rewrite the resulting expression in terms of the Neural Tangent Kernel (NTK). In particular, taking the cross-entropy as the loss function, one obtains:

$$S^t_{a\to b} \approx Z^t_{a\mapsto b} \approx \eta \frac{(\mathbf{f}^b(X) - \mathbf{y}^b)^\top \mathbf{K}^{ba} (\mathbf{f}^a(X) - \mathbf{y}^b)}{N^2 \mathbf{L}_b(t)}, \tag{4}$$

where K_{ba} is the corresponding NTK matrix, where each entry represents the scalar product of the derivatives of the outputs y_a and y_b. We provide the detailed derivations in Appendix A2.

2.3 Connecting task affinity with data pruning

We note an interesting connection between the task-affinity scores and results from data-pruning literature. Essentially, Paul et al. [2021] analyze scores to estimate the impact of individual datapoints on the training performance, focusing on approximating or bounding the effect of a datapoint on the gradient of the loss of any other point. Extending this definition to the multi-task setting is straightforward: by taking means, we can similarly define the impact that training on the task-specific loss L_a will have on the gradient of the loss on task L_b. Calling the resulting measure P, we derive

$$P^t_{a\to b} \approx \eta \frac{(\mathbf{f}^b(X) - \mathbf{y}^b)^\top \mathbf{K}^{ba} (\mathbf{f}^a(X) - \mathbf{y}^a)}{N^2}, \tag{5}$$

The primary distinction between the existing measures defined in equation (2) lies in the denominator. This difference effectively makes $S^t_{a\to b}$ more sensitive to loss reductions when the loss is smaller. As a result, this highlights that the numerator of $S^t_{a\to b}$ is the critical component for the computation of the task-affinity score, while the denominator serves primarily as a normalization factor.
Based on these insights, we rewrite the numerators appearing both in equations (2) and (5) in a more convenient way:

$$\begin{aligned}(\mathbf{f}^b(X) - \mathbf{y}^b)^\top \mathbf{K}^{ba} (\mathbf{f}^a(X) - \mathbf{y}^a) &= \mathbf{Tr}(K_{ba} (\mathbf{f}^b(X) - \mathbf{y}^b)(\mathbf{f}^a(X) - \mathbf{y}^a)^\top) &(6)\\ &= \mathbf{Tr}(K_{ba} G_{ba}) &(7)\end{aligned}$$

where we defined the rank-four deviation tensor G as:

$$G_{kl}[i,j] = (\mathbf{f}^k(X_i) - \mathbf{y}^k_i)(\mathbf{f}^l(X_j) - \mathbf{y}^l_j)^\top.$$

The expression in equation (7) represents the unnormalized alignment of the NTK matrix K_{ba} and G_{ba}. In this context, we refer to the matrices G_{kk} (where $k = l$) as the diagonal blocks, and the matrices where $k \neq l$ as the off-diagonal blocks. Looking at the definitions of the task affinity measures in equations (3) and (5) suggest that the alignment of the off-diagonal blocks of K with those of G plays a crucial role in measuring task affinity, while the denominator L_b acts merely as a weighting factor. We experimentally analyze the evolution of the alignment between the off-diagonal matrices of K and G in section 4.

3 Architecture-dependent task affinities

A major drawback of the existing task-affinity scores, as defined in equations (1) and (2) is their computational inefficiency. In this section, we propose a strategy to avoid the need of the explicit training of the multi-task models to obtain task-affinity scores. While relying on strong assumptions, we show that this method is able to account for the dependency on the specific network architecture, which is essential for the task-grouping problem, as highlighted by Standley et al. [2020]. Applying this method to two basic architectures, we observe that one of them leads to some matrix alignment measure commonly used in statistics.

Paul et al. [2021] demonstrate that scores, which typically need to be computed throughout the entire training process to be informative, can also be estimated in the first few epochs by averaging them over several stochastic initializations. In this section, we further generalize this view, assuming that the expectation of our affinity measure at epoch 0, computed over multiple initializations, is indicative of its average over a full training trajectory from epoch 0 to T.

To apply this idea to the concept of cumulative task affinity, which is defined by equation (2),

$$S_{a\to b} = \sum_{t=1}^{T} S^t_{a\to b}, \tag{8}$$

we take the expected value of the task-affinity score at epoch 0 over different weight initialization $W \sim D$,

$$S_{a\to b} = \mathbb{E}_{W\sim D}\left[S^0_{a\to b|W}\right]. \tag{9}$$

3.1 Explicit expressions for linear networks

We apply the formula in equation (9), along with some Taylor approximations, to two basic architectures based on two-layer linear networks. Again relying on assumption 1, we derive explicit expressions for $\mathbb{E}_{W\sim D}[S_{a\mapsto b}]$.

Proposition 1. *For a 2-layer linear network with a sigmoid applied to the logits, small initialization weights and c classification tasks with the binary cross-entropy as the multi-task loss function defined as:*

$$\textit{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{c}\left[y_i^j \log(\hat{y}_i^j) + (1-y_i^j)\log(1-\hat{y}_i^j)\right],$$

and the task-specific weights $\theta_{a,b}$, being set equal at initialization, we have:

$$\mathbb{E}[S^0_{a\to b}] \propto \mathbf{d}_b^\top \mathbf{X}\mathbf{X}^\top \mathbf{d}_a, \tag{10}$$

where $\mathbf{d}_j \equiv 2\mathbf{y}^j - \mathbf{1}_N$ for class c and $\mathbf{1}_N$ is a N-dimensional vector containing ones.

The detailed derivations are available in Appendix B1. Next, we derive an expression that takes into account the balance of positive and negative samples in the classification tasks (derivation in Appendix B2).

Proposition 2. *With the same setting as in proposition 1 but with the cross-entropy loss adapted to balance the positive and negative samples:*

$$\textit{Loss}_j = 1/N\sum_{i=1}^{N}\left[(c-1)y_i^j \log(\sigma(z_i^j)) + (1-y_i^j)\log(1-\sigma(z_i^j))\right]$$

and

$$\textit{Loss} = \sum_{j=1}^{c} \textit{Loss}_j,$$

we have:

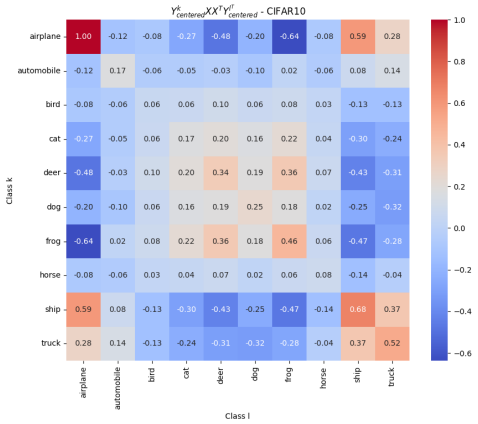
$$\mathbb{E}[S^0_{a\to b}] \propto \delta_b^\top \mathbf{X}\mathbf{X}^\top \delta_a, \tag{11}$$

where, $\delta_j \equiv \left(\mathbf{y}^j - \frac{1}{c}\mathbf{1}\right)$.

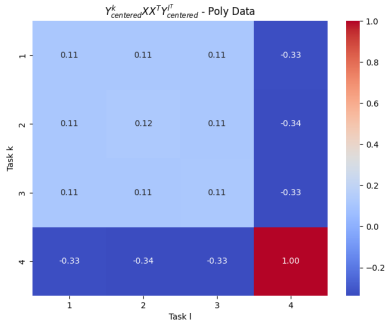
Several key observations can be made about equations (10) and (11). First, note that the use of the term "proportional to" is justified in our context, since the coefficients in front of our expressions are identical across all pairs of tasks. This allows us to focus on comparing tasks relative to one another, rather than being concerned with the specific coefficients themselves. Second, in equation 11, one recognizes the unnormalized Central Kernel Alignment (CKA) $\text{Tr}(\mathbf{X}\mathbf{X}^\top \delta_b \delta_a^\top)$ (see Appendix B3), assuming the input matrix $\mathbf{X}$ is centered. An intuitive way to understand this expression is as a correlation of the correlation matrices for X and for y (the correlations being taken between the datapoints rather than the features). Building on this interpretation of the Gram matrices as a form of correlation matrices, it becomes evident that such an affinity measure will only be capable of capturing linear relationships, as correlation matrices lack higher-order information. This is not

surprising, since we are dealing with a linear network, and the same limitation would apply when extending equations (10) and (11) to more complex models using a Taylor approximation of their output. This limitation highlights the main drawback of the approach in this section: while capturing the dependency on different architectures, these task-affinity metrics inherently lack the expressive power of non-linear functions, resulting in (non-linear) feature blindness.

Despite these limitations, we show that the expression in equation (11) is nevertheless sufficient to provide excellent semantic clustering on CIFAR10 as seen on Figure 1a. Specifically, the task-affinity measure is able to group tasks in a way consistent with the semantic clustering of living and non-living clustering, as suggested by the brighter squares in the middle and corners. It is worth noting that, in the absence of exhaustive benchmarks, it is common practice in papers on task grouping to compare the results of a heuristic to a semantic grouping [Sherif et al., 2024]. This capacity, combined with its linear limitations, makes $\delta_b^\top \mathbf{X}\mathbf{X}^\top \delta_a$ a suitable baseline for comparison with the measure we introduce in the next chapter.



(a) A simple measure based on a 2-layer network shows perfect semantic clustering.



(b) The same measure on synthetic data rightly shows a weaker interraction between task 4 and task 1,2,3 but misses that task 1 is more similar to 2 than to 3

Figure 1: Heatmaps for task affinities computed based on Proposition 2

4 Empirical analysis of NTK-based task-affinity formulas

Linearization in the inputs (Section 2) significantly lessens the computational cost of S and Z, but it fails to capture the full expressive power of neural networks. In contrast, work on telescoping Jeffares et al. [2024] has shown that linearization in the weights accurately approximates the dynamics of the next step in neural network training. To better capture the non-linear dynamics present in multi-task learning, we propose using the linearization in the weights defined in Equation (7).

We study the alignment properties of its two components, the NTK and G, which was shown to be key to understanding task-affinity measures in Section 2.3. As linearization in the weights presents a significant computational challenge, we suggest directions for leveraging alignment properties to reduce the computational cost.

Based on the resulting observations, we also explore whether averaging over initializations can shorten the training time required to observe expected non-linear similarities.

4.1 Empirical analysis of NTK–G alignment

In this section, we empirically analyze the evolution of alignment between the Neural Tangent Kernel (K) and the Gram matrices (G), as defined in Equation (7). Atanasov et al. [2021] and Shan and Bordelon [2021] identified a phenomenon termed *silent alignment*, which can be understood as the anisotropic learning of the Neural Tangent Kernel (NTK) while it adapts to features, followed by isotropic growth in norm after achieving alignment between the NTK diagonal block K_{aa} and the Gram matrix of the target $Y_a Y_a^T$. This behavior bears notable similarities to the expression in

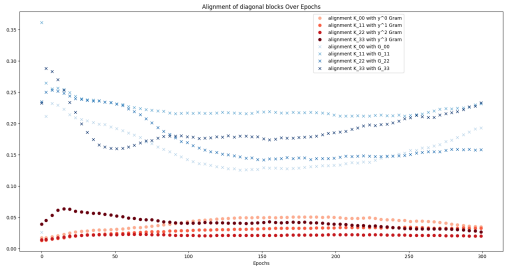
Equation 7, with the distinction that the latter pertains to off-diagonal blocks and involves replacing the target Gram matrices with the Gram matrices of deviations (G_{ab}).

This phenomenon is intriguing in its own right, but it could also lead to significant reductions in computational cost. If, as in the case of silent alignment, such alignment occurs early in training, one could compute S (or Z) only up to the point of alignment and then switch to an approximation. This approximation would involve calculating only the norms of the NTK and G, potentially resulting in substantially reduced computation.

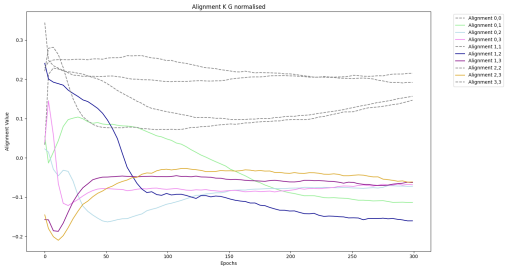
Experimental setup. We train a four-layer dense neural network on a synthetic dataset containing four classification tasks, designed as follows:

1. Tasks 1 and 2 share the quadratic feature $(x_1^2 + x_2^2)$,
2. Task 3 is partially related to Tasks 1 and 2 through (x_2^2), and
3. Task 4 is unrelated to the other tasks.

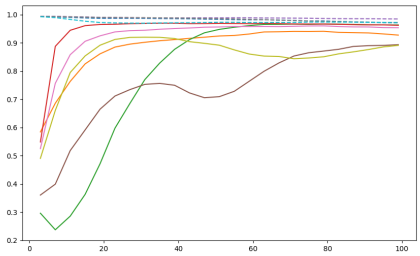
The dataset is structured so that Tasks 1, 2, and 3 exhibit equal affinities toward each other based on the measure defined in Equation (11). Detailed experimental settings can be found in Appendix C.



(a) Alignment of the diagonal blocks on synthetic data.



(b) Diagonal vs. off-diagonal matrices.



(c) The alignment between a single run and the mean NTK over 5 runs.

Figure 2: In **Figure 2a**, the red lines represent the alignment of K with the target Gram matrix, while the blue lines represent the alignment of K with G. Alignment between the diagonal blocks of K and the target Gram matrix increases slightly at the beginning of training before stabilizing. However, the alignment between K and G is substantially higher from the start and remains relatively stable, with minor fluctuations. In **Figure 2b**, the gray dotted lines indicate alignment with the diagonal blocks, and the colored lines correspond to the off-diagonal blocks. Alignment for off-diagonal blocks is chaotic and stabilizes over a longer period compared to diagonal blocks, highlighting the distinct dynamical behavior of off-diagonal components. In **Figure 2c**, we observe a relative convergence of the NTK computed from a single run toward the mean NTK over 5 runs. Diagonal blocks of the NTK quickly align with the mean NTK computed over 5 runs, while off-diagonal blocks achieve alignment levels exceeding 0.8 after 60 epochs, even for the most erratic blocks.

Key observations. Our analysis identifies three significant alignment properties:

1. **Diagonal block alignment:** The alignment between the diagonal blocks of K and G is comparable to—or even greater than—the alignment between K and the Gram matrix of $\mathbf{y}$.
2. **Off-diagonal block alignment:** The alignment of off-diagonal blocks exhibits greater instability and evolves over a longer timescale compared to the diagonal blocks, indicating distinct dynamics.
3. **Cross-model consistency:** The G_{ab} matrices across different initializations demonstrate strong alignment throughout training. In contrast, the K_{ab} matrices begin with significant divergence but align strongly after sufficient training.

Observation 1 aligns with findings from Shan and Bordelon [2021] and Atanasov et al. [2021], though the effect observed here is even more pronounced. This result is encouraging, as it suggests that after an initial phase of feature learning, the numerator in our measures may depend primarily on the norms of K and G. These norms could potentially be approximated using computationally cheaper methods than calculating the full NTK. Observation 3 suggests that averaging across model initializations can mitigate stochasticity, particularly for the off-diagonal blocks of K and G. However, Observation 2 highlights that relying solely on alignment predictability may not yield significant computational savings if alignment occurs late in training or its timing is uncertain. To address these limitations, we propose a method for computing S that averages the NTK and G matrices across multiple runs. Preliminary results on synthetic data suggest that this method becomes sensitive to non-linear features after approximately 50 epochs.

4.2 Reducing stochasticity through the mean over initialization.

Based on the observed fluctuations in the alignment of the off-diagonal blocks, we propose to reduce this stochasticity of the alignment by averaging K and G across several models. Two key observations support this decision:

1. Figures 2b and 2c indicate that the blocks of the NTK nearly converge across different runs when given sufficient training time. However, a high degree of stochasticity at the beginning of training obscures the information that might be present in the final NTK. The same examination of G reveals that the alignment between any two runs is nearly perfect (well above 0.9) from the very start of training. Based on these observations, we propose that averaging over multiple runs for both G and the NTK could help mitigate the stochasticity introduced by initialization.
2. The diagonal matrices of G, when squared element-wise and summed together, form a square matrix whose diagonal elements are $\sum_{k=1}^{c}(f^k(x_i) - y_i^k)^2 = ||f(x_i) - y_i||^2$. This expression is identical to the EL2N scores defined in Paul et al. [2021], where it was observed that these scores do not provide meaningful information for pruning if measured only at the beginning of training. However, they become informative when averaged over 10 different initializations. This observation suggests that averaging can mitigate some of the randomness from initialization, prompting the question of whether the same effect might occur with the full tensor G.

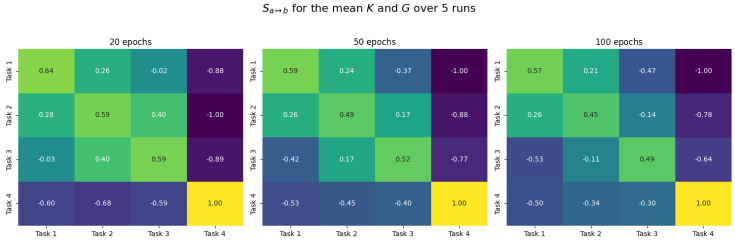


Figure 3: The tasks affinity computed at various epochs through the mean over 5 runs of K and G

We experimented with this approach by training five models, each with different initializations, on our synthetic data. As shown in Figure 3, the strong affinity between tasks 1 and 2 becomes apparent from epoch 50, while the other affinities align with our expectations. It is important to note that negative values of S or P do not indicate negative transfer between tasks; instead, these measures

are intended solely for comparing task pairs relative to each other. This leads us to propose that computing the measure from Equation 3 over a few epochs at the beginning of training, across several runs, could be an effective way to estimate $S_{a \mapsto b}$. However, further investigation is required to fully understand the potential savings and to assess its applicability to more complex datasets.

5 Conclusion

Our goal was to analyze and potentially accelerate the computation of gradient-based task similarity measures (such as S or Z), which are traditionally calculated over the entire training trajectory. To this end, we explored two forms of linearization to simplify the learning dynamics of neural networks.

The first, linearization in the input space, allows us to derive simple formulas for S or Z that are architecture dependent. One of these formulas connects the CKA to a 2 layers network for classification. However, this approach is fundamentally limited by its linear nature, rendering it unable to capture the non-linear relationships inherent in neural networks.

The second, more informative approach, involves linearization in the weight space. From a theoretical perspective, this revealed the equivalence of two widely used task-affinity measures and highlighted the importance of a key quantity: the alignment between the off-diagonal blocks of the NTK matrix (K) and the tensor G.

We empirically investigated the alignment properties between the block matrices of K and G and described three significant alignment properties. Additionally, we proposed a method to reduce the initial stochasticity of alignment in the off-diagonal blocks by averaging the NTK and G matrices over several stochastic initializations. On our synthetic data, this method successfully identified non-linear task similarities after 50 epochs of training.

More work is necessary to determine whether NTK-based methods can lead to more efficient computations of task-affinity measures. In particular, future research should investigate the impact of computing the NTK on subsets of data points of varying sizes, the speed of alignment across different architectures and datasets, and the regularity of the NTK norm evolution after alignment. These directions could further enhance the practicality of NTK-related methods for task grouping in multi-task learning.

References

Alexander Atanasov, Blake Bordelon, and Cengiz Pehlevan. Neural networks as kernel learners: The silent alignment effect. *arXiv preprint arXiv:2111.00034*, 2021.

Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13:795–828, 2012.

Omer Elkabetz and Nadav Cohen. Continuous vs. discrete optimization of deep neural networks. *Advances in Neural Information Processing Systems*, 34:4947–4960, 2021.

Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27503–27516. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/e77910ebb93b511588557806310f78f1-Paper.pdf`.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Alan Jeffares, Alicia Curth, and Mihaela van der Schaar. Deep learning through a telescoping lens: A simple model provides empirical insights on grokking, gradient boosting & beyond. *arXiv preprint arXiv:2411.00247*, 2024.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34: 20596–20607, 2021.

Haozhe Shan and Blake Bordelon. A theory of neural tangent kernel alignment and its influence on training. *arXiv preprint arXiv:2105.14301*, 2021.

Ammar Sherif, Abubakar Abid, Mustafa Elattar, and Mohamed ElHelw. Stg-mtl: scalable task grouping for multi-task learning using data maps. *Machine Learning: Science and Technology*, 5 (2):025068, 2024.

Xiaozhuang Song, Shun Zheng, Wei Cao, James Yu, and Jiang Bian. Efficient and effective multi-task grouping via meta learning on task combinations. *Advances in Neural Information Processing Systems*, 35:37647–37659, 2022.

Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9120–9132. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/standley20a.html`.

Chenguang Wang, Xuanhao Pan, and Tianshu Yu. Towards principled task grouping for multi-task learning. *arXiv preprint arXiv:2402.15328*, 2024.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE transactions on knowledge and data engineering*, 34(12):5586–5609, 2021.

A Theoretical derivations

A1 The Neural Tangent Kernel NTK

The NTK, introduced by Jacot et al. [2018], is defined as:

$$K_{kl}[x_i, x_j] = \nabla_\theta f^k(x_i)^T \nabla_\theta f^l(x_j),$$

where $f^k(x_i)$ represents the k-th output of the model for input x_i, and θ denotes the vector of network parameters. The NTK captures the relationship between changes in model parameters and output function changes, providing a way to study the model's behavior during training. The NTK naturally appears when linearizing the neural network's trajectory with respect to its weights during training via gradient descent with step η.

$$\frac{df(x_i)}{dt} = \nabla_\theta f(x_i)^T \frac{d\theta}{dt} = -\eta \sum_j K[x_i, x_j] \frac{\partial \mathcal{L}}{\partial f(x_j)}.$$

A key result from Jacot et al. [2018] is that, in the infinite width limit of neural networks, the NTK becomes deterministic and remains constant throughout training. In practice, the NTK's behavior in finite-width networks is more complex, evolving over time.

A2 Taylor approximation for Z

As under the assumption 1 the weights stay constant during a full epoch, we compute the update on a full epoch. The Taylor approximation gives:

$$L_b(\mathbf{X}; \phi_a^{t+1}; \theta_b^t) - L_b(\mathbf{X}; \phi^t; \theta_b^t) = (\nabla_{\phi^t} L^b)^T (\phi_a^{t+1} - \phi_a^t),$$

where ϕ_a^{t+1} are the share weights after training a full epoch on the loss a only. Using gradient descent,

$$\phi_a^{t+1} = \phi_a^t - \eta \nabla_\phi L^a(\phi^t).$$

we get

$$Z_{a \mapsto b}^t \approx \eta \frac{(\nabla_\phi L_b)^\top \nabla_\phi L_a}{L_b(\phi^t, \theta_b^t)}$$

where it is important to note that the gradients are computed only in relation to the shared weights.

A3 Taylor approximation for S

Derivation of S

$$L_b^b(t+1) - L_{a,b}^b(t+1) = (\nabla_{\phi,\theta_b} L^b)^T (\Delta W_{|b} - \Delta W_{|a,b}),$$

where $\Delta W_{|b}$ and $\Delta W_{|a,b}$ represent the parameter updates when training only on class b and jointly on classes a and b, respectively.

Using gradient descent, update on any set of classes S is

$$W_S^{t+1} = W^t - \eta \sum_{j \in S} \nabla_{\phi,\theta_j} L^b(W^t).$$

We compute the difference:

$$\Delta W_{|b} - \Delta W_{|a,b} = (W_{|b}^{t+1} - W^t) - (W_{|a,b}^{t+1} - W^t).$$

which gives:

$$S_{a \mapsto b}^t = \frac{L_b^b(t+1) - L_{a,b}^b(t+1)}{L_b^b(t)} = \eta \frac{(\nabla_{\phi,\theta_b} L^b)^T \nabla_{\phi,\theta_a} L^a}{L_b^b(t)}.$$

where

$$\nabla_{\phi,\theta_b} L_b = \frac{1}{N} \sum_{x \in X} \nabla_{\phi,\theta_b} L_b(x; \phi, \theta_b).$$

Substituting this into our earlier expression for $S_{a \mapsto b}^t$, we get:

$$S_{a \mapsto b}^t = \eta \frac{\left(\frac{1}{N} \sum_{x \in X} \nabla_{\phi,\theta_b} L_b(x; \phi, \theta_b)\right)^T \left(\frac{1}{N} \sum_{x \in X} \nabla_{\phi,\theta_a} L_a(x; \phi, \theta_a)\right)}{\frac{1}{N} \sum_{x \in X} L_b(x; \phi, \theta_b)}.$$

Given that we use cross-entropy as the loss function, we apply the chain rule to further refine the expression.

$$\nabla_{\phi,\theta_b} L_b = \frac{1}{N} \sum_{x \in X} \frac{\partial L_b(x)}{\partial f_b(x)} \nabla_{\phi,\theta_b} f_b(x) = \frac{1}{N} \sum_{x \in X} (f_b(x) - y_b(x)) \nabla_{\phi,\theta_b} f_b(x).$$

Similarly, for $\nabla_{\phi,\theta_a} L_a$:

$$\nabla_{\phi,\theta_a} L_a = \frac{1}{N} \sum_{x \in X} (f_a(x) - y_a(x)) \nabla_{\phi,\theta_a} f_a(x).$$

Let $\mathbf{d}_b \in \mathbb{R}^N$ and $\mathbf{d}_a \in \mathbb{R}^N$ be the vectors of deviations:

$$\mathbf{d}_b = \begin{bmatrix} f_b(x_1) - y_b(x_1) \\ \vdots \\ f_b(x_N) - y_b(x_N) \end{bmatrix}, \quad \mathbf{d}_a = \begin{bmatrix} f_a(x_1) - y_a(x_1) \\ \vdots \\ f_a(x_N) - y_a(x_N) \end{bmatrix}.$$

Let $K_{ba} \in \mathbb{R}^{N \times N}$ be the NTK matrix with entries $K_{ba}(x_i, x_j)$.

The numerator can then be written as:

$$\text{Numerator} = \frac{1}{N^2} \mathbf{d}_b^\top K_{ba} \mathbf{d}_a.$$

Substituting this back into $S_{a \mapsto b}^t$, we have:

$$S_{a \mapsto b}^t = \eta \frac{\frac{1}{N^2} \mathbf{d}_b^\top K_{ba} \mathbf{d}_a}{\frac{1}{N} \sum_{x \in X} L_b(x; \phi, \theta_b)} = \eta \frac{\mathbf{d}_b^\top K_{ba} \mathbf{d}_a}{N \sum_{x \in X} L_b(x; \phi, \theta_b)}.$$

B

B1 Derivation for proposition 1

We now consider a 2-layer neural network with sigmoid activation at each output and a binary cross-entropy (CE) loss summed over all classes. The input data is $\mathbf{X} \in \mathbb{R}^{N\times d}$, with first layer weights $\mathbf{W}^1 \in \mathbb{R}^{h\times d}$ and second layer weights $\mathbf{W}^2 \in \mathbb{R}^{c\times h}$. The hidden layer activations are $\mathbf{H} = \mathbf{X}{\mathbf{W}^1}^\top \in \mathbb{R}^{N\times h}$, and the logits are $\mathbf{O} = \mathbf{H}{\mathbf{W}^2}^\top \in \mathbb{R}^{N\times c}$. Labels for each class are denoted by $\mathbf{y}_a, \mathbf{y}_b \in \mathbb{R}^{N\times 1}$. The loss function is given by:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{c}\left[y_{ij}\log(\hat{y}_{ij}) + (1-y_{ij})\log(1-\hat{y}_{ij})\right].$$

We want to compute:

$$\mathbb{E}[S^0_{a\to b}] = \mathbb{E}[\frac{\Delta L_b}{L_b^{\{b\}}}].$$

where ΔL_b is the difference in losses between training on both classes $\{a, b\}$ and training only on class b. In the following derivation, as well as in B2, we approximate the denominator $L_b^{\{b\}}$ by the loss at initialization, $L(W)$, leveraging the assumption of small step sizes. Furthermore, $L(W)$ is assumed to be approximately constant across different random initializations of the network weights. This assumption is justified based on the concentration of measure phenomenon for Lipschitz functions of Gaussian random variables and our small weights initialization assumption. We develop this argument in the following points:

1. Lipschitz continuity of the loss function:

The loss function $L(W)$ depends on the weights W through the network outputs and the loss computation. We aim to show that $L(W)$ is Lipschitz continuous with respect to W.

1a. Sigmoid activation function:

The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ has the derivative:

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \le \frac{1}{4},$$

since $\sigma(z)(1-\sigma(z))$ reaches its maximum value of $\frac{1}{4}$ at $z = 0$. Therefore, the sigmoid function is Lipschitz continuous with Lipschitz constant $L_\sigma = \frac{1}{4}$:

$$|\sigma(z_1) - \sigma(z_2)| \le L_\sigma |z_1 - z_2|.$$

1b. Network outputs:

The pre-activation outputs (logits) Z_{ij} are linear functions of W. Since W has small entries, Z_{ij} are also small, and thus $\sigma(Z_{ij}) \approx 0.5$.

1c. Cross-entropy loss function:

For the binary cross-entropy loss, the loss for one sample and one class is:

$$L_{ij}(W) = -\left[y_{ij}\log(\hat{y}_{ij}) + (1-y_{ij})\log(1-\hat{y}_{ij})\right],$$

where $\hat{y}_{ij} = \sigma(Z_{ij})$.

Since $\hat{y}_{ij} \approx 0.5$, $\hat{y}_{ij}$ is bounded away from 0 and 1. The logarithmic functions $\log(\hat{y}_{ij})$ and $\log(1-\hat{y}_{ij})$ are Lipschitz continuous on the interval $[\delta, 1-\delta]$ for some $\delta > 0$, with Lipschitz constants $L_{\log}$ depending on δ.

1d. Combined Lipschitz constant:

Each $L_{ij}(W)$ is Lipschitz continuous with respect to W. Since

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{c} L_{ij}(W)$$

is a finite sum of Lipschitz functions, it is Lipschitz continuous with Lipschitz constant L_L, which is proportional to the constants from the activation and loss functions.

2. Concentration inequality:

Let $f : \mathbb{R}^d \to \mathbb{R}$ be an L_f-Lipschitz function, and let $Z \in \mathbb{R}^d$ be a random vector with independent Gaussian entries $Z_i \sim \mathcal{N}(0, \sigma^2)$. Then, from standard concentration inequalities for Lipschitz functions of Gaussian variables, we have:

$$P(|f(Z) - \mathbb{E}[f(Z)]| \geq t) \leq 2 \exp\left(-\frac{t^2}{2L_f^2 \sigma^2}\right).$$

Applying this to $L(W)$ with Lipschitz constant L_L, we get:

$$P(|L(W) - \mathbb{E}[L(W)]| \geq t) \leq 2 \exp\left(-\frac{t^2}{2L_L^2 \sigma^2}\right).$$

By choosing σ sufficiently small (as part of our small weights initialization assumption), we can make $L_L^2 \sigma^2$ arbitrarily small. Consequently, for any fixed $t > 0$, the probability

$$P(|L(W) - \mathbb{E}[L(W)]| \geq t)$$

becomes negligibly small.

Therefore, $L(W)$ concentrates sharply around its mean $\mathbb{E}[L(W)]$, and we can approximate $L(W) \approx \mathbb{E}[L(W)]$ across different initializations.

This justifies our assumption that the loss at initialization is effectively independent of the weights, allowing us to treat L as constant in our derivations.

From here we focus on computing the numerator. The gradient of the loss with respect to the first layer weights, $\mathbf{W}^1$, is:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}^1} = \frac{1}{N} \left(\left(\mathbf{W}^2\right)^T \left(\sigma(\mathbf{W}^2 \mathbf{W}^1 \mathbf{X}^T) - \mathbf{Y}^T\right) \right) \mathbf{X}.$$

Similarly, the gradient with respect to the second layer weights, $\mathbf{W}^2$, is:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}^2} = \frac{1}{N} \left(\sigma(\mathbf{W}^2 \mathbf{W}^1 \mathbf{X}^T) - \mathbf{Y}^T\right) \mathbf{X} (\mathbf{W}^1)^T.$$

Using the first-order Taylor approximation for the sigmoid function, $\sigma(x) \approx \frac{1}{2} + \frac{1}{4}x$, we obtain:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}^1} = \frac{1}{N} \left(\left(\mathbf{W}^2\right)^T \left(\frac{1}{2} \mathbf{1}_{Nc}^T + \frac{1}{4} (\mathbf{W}^2 \mathbf{W}^1 \mathbf{X}^T) - \mathbf{Y}^T \right) \right) \mathbf{X},$$

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}^2} = \frac{1}{N} \left(\frac{1}{2} \mathbf{1}_{Nc}^T + \frac{1}{4} (\mathbf{W}^2 \mathbf{W}^1 \mathbf{X}^T) - \mathbf{Y}^T \right) \mathbf{X} (\mathbf{W}^1)^T.$$

To compute the effect of training on both classes a and b, the loss denominator corresponds to training on the full dataset while considering only the loss from class b. This requires modifying $\mathbf{Y}$ to retain only the entries in column b. For the numerator, we retain both columns a and b as non-zero.

The difference in gradients between training on both classes $\{a, b\}$ and training only on class b is given by:

$$\Delta\left(\frac{\partial \text{Loss}}{\partial \mathbf{W}^1}\right) = \frac{1}{N}{\mathbf{w}_a^2}^\top \left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a + \frac{1}{4}\mathbf{O}_a\right)\mathbf{X}.$$

Note that the change in w_b^2 is the same whether training on b only or on a and b, therefore the difference in logits for class b after training on classes $\{a, b\}$ versus training only on b is:

$$\mathbf{O}_b^{\{a,b\}} - \mathbf{O}_b^{\{b\}} = -\eta\mathbf{X}\left(\Delta\left(\frac{\partial \text{Loss}}{\partial \mathbf{W}^1}\right)^\top\right){\mathbf{w}_b^2}^\top.$$

Substituting the expression for $\Delta\left(\frac{\partial \text{Loss}}{\partial \mathbf{W}^1}\right)$:

$$\mathbf{O}_b^{\{a,b\}} - \mathbf{O}_b^{\{b\}} = -\eta\mathbf{X}\left(\mathbf{X}^\top\left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a + \frac{1}{4}\mathbf{O}_a\right)\mathbf{w}_a^2\right){\mathbf{w}_b^2}^\top.$$

Assuming $\mathbf{w}_a^2 = \mathbf{w}_b^2$ (this is necessary as otherwise all terms average to 0 when taking the mean), the scalar product $s = \mathbf{w}_a^2{\mathbf{w}_b^2}^\top$ becomes $s = \|\mathbf{w}_a^2\|^2$, simplifying the difference in logits:

$$\mathbf{O}_b^{\{a,b\}} - \mathbf{O}_b^{\{b\}} = -\eta s\mathbf{X}\mathbf{X}^\top\left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a + \frac{1}{4}\mathbf{O}_a\right).$$

Finally, the change in the loss is:

$$\Delta L_b = \frac{1}{2N}\mathbf{d}_b^\top\left(\mathbf{O}_b^{\{a,b\}} - \mathbf{O}_b^{\{b\}}\right),$$

where $\mathbf{d}_b = 2\mathbf{y}_b - \mathbf{1}_N$. Substituting the expression for $\mathbf{O}_b^{\{a,b\}} - \mathbf{O}_b^{\{b\}}$:

$$\Delta L_b = -\frac{\eta s}{2N}\mathbf{d}_b^\top\mathbf{X}\mathbf{X}^\top\left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a + \frac{1}{4}\mathbf{O}_a\right).$$

Assuming $\mathbf{w}_a^2 = \mathbf{w}_b^2$ at initialization, the dot product $s = \|\mathbf{w}_a^2\|^2$ becomes deterministic, and the expected transfer gain $S_{a\to b}^0$ can be expressed as:

$$\mathbb{E}[S_{a\to b}^0] = \frac{\mathbb{E}[\Delta L_b]}{L_b^{\{b\}}}.$$

Supposing that the initialization is Gaussian i.i.d., i.e., $\mathbf{w}_i \sim \mathcal{N}(0, \sigma^2)$, we need to compute :

$$\mathbb{E}[\Delta L_b] = -\frac{\eta}{2N}\mathbb{E}\left[\left(\mathbf{w}_a^\top\mathbf{w}_a\right)\mathbf{d}_b^\top\mathbf{X}\mathbf{X}^\top\left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a + \frac{1}{4}\mathbf{O}_a\right)\right].$$

Term 1:

$$-\frac{\eta}{2N}\mathbb{E}\left[\left(\mathbf{w}_a^\top\mathbf{w}_a\right)\mathbf{d}_b^\top\mathbf{X}\mathbf{X}^\top\left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a\right)\right]$$

Since $\mathbf{w}_a$ has entries $w_{a,k} \sim \mathcal{N}(0, \sigma^2)$, we have:

$$\mathbb{E}\left[\mathbf{w}^\top\mathbf{w}\right] = h\sigma^2,$$

where h is the number of hidden units.

Term 2:

$$-\frac{\eta}{2N}\mathbb{E}\left[\left(\mathbf{w}_a^\top\mathbf{w}_a\right)\mathbf{d}_b^\top\mathbf{X}\mathbf{X}^\top\frac{1}{4}\mathbf{O}_a\right].$$

When expanding encounter cubic terms like $w_{a,k}^2 w_{a,k}$ and $w_{a,k}^2 w_i$ for $i \neq k$.

For Gaussian variables with zero mean, odd moments are zero:

$$\mathbb{E}[w_{a,k}^3] = 0.$$

For $i \neq k$:

$$\mathbb{E}[w_{a,k}^2 w_{a,i}] = \mathbb{E}[w_{a,k}^2]\mathbb{E}[w_{a,i}] = 0.$$

Note here that without our assumption that $w_a = w_b$ the expectation would have been 0. Instead, we get:

$$\mathbb{E}[\Delta L_b] = -\frac{\eta h \sigma^2}{2N} \mathbb{E}\left[\mathbf{d}_b^\top \mathbf{X}\mathbf{X}^\top \left(\frac{1}{2}\mathbf{1}_N - \mathbf{y}_a\right)\right].$$

As the purpose of the measure is to compare pairs of tasks between themselves, and making the approximation that the loss at initialization is always the same, we can drop the coefficients and get:

$$\mathbb{E}[S_{a\to b}^0] \propto \mathbf{d}_b^\top \mathbf{X}\mathbf{X}^\top \mathbf{d}_a$$

B2 Derivation for proposition 2

We consider c balanced tasks and a modified loss function adapted so that negative samples have the same impact as positive ones. For class j, the loss is defined as:

$$\text{Loss}_j = \sum_{i=1}^{N} \left[(c-1) y_i^j \log(\sigma(o_i^j)) + (1 - y_i^j) \log(1 - \sigma(o_i^j))\right],$$

where $\sigma(o_i^j) = \hat{y}_i^j$ is the sigmoid of the logit o_i^j.

The derivative of the loss with respect to o_i^j is:

$$\frac{\partial L_i}{\partial o_i^j} = (c-1) y_i^j (1 - \sigma(o_i^j)) - (1 - y_i^j)\sigma(o_i^j).$$

Building on the observation in Appendix B2 that the terms in O will average out, we drop them now to simplify and we approximate $\sigma(o_i^j) \approx 0.5$. Substituting this approximation gives:

$$\frac{\partial L_i}{\partial o_i^j} \approx \frac{1}{2}\left((c-1) y_i^j - (1 - y_i^j)\right).$$

This simplifies to:

$$\delta_i^j = \frac{1}{2}\left(c y_i^j - 1\right).$$

In vector form, the gradient $\delta_j \in \mathbb{R}^N$ can be expressed as:

$$\delta_j = \frac{1}{2}\left(c\mathbf{y}_j - \mathbf{1}\right),$$

where $\mathbf{y}_j$ is the vector of labels for class j and $\mathbf{1}$ is a vector of ones.

The gradient with respect to W_1 is:

$$\frac{\partial \text{Loss}_j}{\partial W_1} = \sum_{i=1}^{N} \delta_i^j \frac{\partial h_i^j}{\partial W_1}.$$

Since $o_i^j = W_{2j} h_i$ and $h_i = W_1 x_i$, where W_{2j} is a row vector, we have:

$$\frac{\partial o_i^j}{\partial W_1} = W_{2j}^\top x_i^\top.$$

With a learning rate η, the update to $\mathbf{W}^1$ due to training on both classes is:

$$\Delta \mathbf{W}^1 = -\eta \left(\frac{\partial \text{Loss}}{\partial \mathbf{W}^1} \right)_{\{a,b\}}.$$

The difference in $\mathbf{W}^1$ after training on both classes versus only class b is:

$$-\eta \Delta \left(\frac{\partial \text{Loss}}{\partial \mathbf{W}^1} \right) = -\frac{\eta}{N} \mathbf{w}_a^{2^\top} \boldsymbol{\delta}_a^\top \mathbf{X}.$$

From here we repeat the reasoning of Appendix B2 to get

$$\mathbb{E}[S^0_{a \to b}] \propto \boldsymbol{\delta}_b^\top \mathbf{X} \mathbf{X}^\top \boldsymbol{\delta}_a$$

B3 Centered Kernel Alignment (CKA)

The concept of CKA was introduced in Cortes et al. [2012] and is defined for two kernel matrices $\mathbf{K}$ and $\mathbf{K}'$ as follows:

$$\text{CKA}(\mathbf{K}, \mathbf{K}') = \frac{\langle \mathbf{K}_c, \mathbf{K}'_c \rangle_F}{\|\mathbf{K}_c\|_F \|\mathbf{K}'_c\|_F},$$

where $\mathbf{K}_c$ and $\mathbf{K}'_c$ are the centered versions of $\mathbf{K}$ and $\mathbf{K}'$, corresponding to centering the data in the feature space. Here, $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product, and $\| \cdot \|_F$ is the Frobenius norm.

Specializing this definition to our case of interest implies centering both $\mathbf{Y}$ and $\mathbf{X}$ before computing their Gram matrices, which is precisely what the $-\frac{1}{c}\mathbf{1}$ does if the classes are balanced and non-overlapping.

The CKA has been used in various studies to compare the representations learned across different neural networks, providing insights into the similarity and alignment of these representations.

C Data and models

C1 Data

In our experiments, we use a synthetic dataset designed to study task affinities under controlled conditions. The dataset consists of multiple tasks, each associated with a specific geometric structure in feature space. Specifically, tasks 1, 2, and 3 are constructed to have identical linear correlations as measured by the matrix product $\mathbf{X}\mathbf{X}^\top \mathbf{y}\mathbf{y}^\top$ between any pair of these tasks. This ensures that linear measures of similarity cannot distinguish between these tasks, allowing us to investigate the ability of non-linear measures to capture deeper relationships.

The dataset is generated as follows:

Feature space The feature matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$ consists of N samples with d features. For visualization and construction purposes, we focus on the first five features $(x_1, x_2, x_3, x_4, x_5)$.

Task 1: Cylindrical structure Task 1 is defined such that the positive samples form a space between two concentric cylinders aligned along the x_3 axis. Mathematically, the samples satisfy:

$$.65 \leq \sqrt{x_1^2 + x_2^2} \leq .7,$$

where r_1 and r_2 are the inner and outer radii of the cylinders, respectively. This creates a hollow cylindrical shell in the x_1-x_2 plane extended along x_3.

Task 2: Conical structure Task 2 is defined such that the positive samples satisfy:

$$0.4 < x_1^2 + x_2^2 - x_3 < 0.5.$$

This equation represents a cylinder in three-dimensional space, where the cross-section in the x_1-x_2 plane is a circle shifted along the x_3 axis. The inequality specifies a thin shell between two parallel circular surfaces, creating a hollow conic region.

Task 3: Spherical structure Task 3 is defined such that the positive samples form a space between two spheres centered at the origin (which is the shared center with Tasks 1 and 2). The samples satisfy:

$$.4 \leq \sqrt{x_1^2 + x_2^2 + x_3^2} \leq .5,$$

where s_1 and s_2 are the inner and outer radii of the spheres. This creates a spherical shell in the 3D space of x_1, x_2, and x_3.

Task 4 Task 4 designed to be unrelated to the first three tasks, serving as baseline.

C2 Ensuring identical linear correlations

By constructing the tasks as described, we ensure that the linear correlations between any pair of Tasks 1, 2, and 3, as measured by $\mathbf{X}\mathbf{X}^\top \mathbf{y}\mathbf{y}^\top$, are the same. This is because the tasks share the same center and their positive samples occupy regions that are symmetric and equidistant in the feature space.

Purpose of the design The design of the synthetic dataset serves to test the ability of task similarity measures to capture non-linear relationships. Since linear correlations cannot distinguish among Tasks 1, 2, and 3, a measure sensitive to non-linear features is required to correctly identify the underlying similarities and differences among these tasks.

C3 Model and training details

Our model is a neural network designed for multi-class classification with the following structure:

Network architecture

- **Input Layer:** The input shape is $(\text{batch\_size}, d)$, where d is the input dimensionality (set to 10 in our experiments).
- **First Hidden Layer:** A dense layer with hidden_width units (set to 256), weights initialized with a standard deviation of $W_{\text{std}}^{\text{first}} = 1$ and biases with $b_{\text{std}}^{\text{first}} = 0.05$. This layer is followed by Layer Normalization and a ReLU activation function.
- **Subsequent Hidden Layers:** We use num_layers $- 1$ additional dense layers, each with hidden_width units. The weights and biases are initialized with standard deviations of $W_{\text{std}}^{\text{other}} = 0.6$ and $b_{\text{std}}^{\text{other}} = 0.05$, respectively. Each layer is followed by Layer Normalization and a ReLU activation function. In our experiments, num_layers is set to 4.
- **Output Layer:** A dense layer with n_{classes} units, where $n_{\text{classes}} = 4$. The weights and biases of this layer are initialized with $W_{\text{std}}^{\text{other}} = 0.6$ and $b_{\text{std}}^{\text{other}} = 0.05$. A sigmoid activation function is applied to each output unit for independent multi-class predictions.

Training and loss function. The network is trained using stochastic gradient descent (SGD) with a learning rate of 1 (in jax which scales down the output of each layer by $\frac{1}{sqrtwidth}$). The training process uses a batch size of 128. The binary cross-entropy (BCE) loss function is employed with class weighting to address class imbalance:

$$w_i = \begin{cases} n_{\text{classes}} - 1 & \text{if } y_i = 1, \\ 1 & \text{otherwise.} \end{cases} \tag{12}$$

The BCE loss for a sample is computed as:

$$\mathcal{L} = -\sum_i w_i \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right], \tag{13}$$

where p_i is the predicted probability for the i-th sample. The use of class weights ensures that positive labels are emphasized during training. The training spans 300 epochs with alignment metrics computed every 4 epochs.

Hyperparameters. The key hyperparameters used in our model and training process are summarized as follows:

- **Number of Classes (n_{classes}):** 4
- **Input Dimension (d):** 10
- **Number of Layers (num_layers):** 4
- **Hidden Width (hidden_width):** 256
- **First Layer Weight Std ($W_{\text{std}}^{\text{first}}$):** 1
- **First Layer Bias Std ($b_{\text{std}}^{\text{first}}$):** 0.05
- **Other Layers Weight Std ($W_{\text{std}}^{\text{other}}$):** 0.6
- **Other Layers Bias Std ($b_{\text{std}}^{\text{other}}$):** 0.05
- **Batch Size:** 128
- **Learning Rate:** 1
- **Number of Epochs:** 300
- **Subsampling per Class:** 50 datapoints

This setup forms the basis of our model, training, and evaluation strategy for the multi-class classification task.

D Code availability

The code for reproducing the experiments and results presented in this paper will be made available at the following GitHub repository:

`https://github.com/yoannmorello/NTK-MTL.git`