EDITING MODELS WITH TASK ARITHMETIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Changing how pre-trained models behave—e.g., improving their performance on a downstream task or mitigating biases learned during pre-training—is a common practice when developing machine learning systems. In this work, we propose a new paradigm for steering the behavior of neural networks, centered around *task vectors*. A task vector specifies a direction in the weight space of a pre-trained model, such that movement in that direction improves performance on the task. We build task vectors by subtracting the weights of a pre-trained model from the weights of the same model after fine-tuning on a task. We show that these task vectors can be modified and combined together through arithmetic operations such as negation and addition, and the behavior of the resulting model is steered accordingly. Moreover, task vectors can be added together to improve performance on multiple tasks at once. Finally, when tasks are linked by an analogy relationship of the form "A is to B as C is to D", combining task vectors from three of the tasks can improve performance on the fourth, even when no data from the fourth task is used for training.

1 INTRODUCTION

Pre-trained models are the backbone of many machine learning systems. In practice, we often want to *edit* models after pre-training,[1] to improve performance on downstream tasks (Zhuang et al., 2020; Wortsman et al., 2022b; Matena & Raffel, 2021; Ilharco et al., 2022), mitigate biases or unwanted behavior (Santurkar et al., 2021; Lu et al., 2022; Ribeiro & Lundberg, 2022; Murty et al., 2022), align models with human preferences (Askell et al., 2021; Ouyang et al., 2022; Kasirzadeh & Gabriel, 2022; Glaese et al., 2022), or update models with new information (Zhu et al., 2020; De Cao et al., 2021; Mitchell et al., 2021; 2022).

In this work, we present a new paradigm for editing neural networks based on *task vectors*, which encode the information necessary to do well on a given task. Inspired by recent work on weight interpolation (Frankle et al., 2020; Wortsman et al., 2022b; Matena & Raffel, 2021; Wortsman et al., 2022a; Ilharco et al., 2022; Li et al., 2022; Ainsworth et al., 2022), we obtain such vectors by taking the weights of a model fine-tuned on a task and subtracting the corresponding pre-trained weights (Figure 1a).

We show that we can edit a variety of models with *task arithmetic*—performing simple arithmetic operations on task vectors (Figure 1b-d). For example, *negating* a vector can be used to remove undesirable behaviors or unlearn tasks, while *adding* task vectors leads to better multi-task models, or even improves performance on a single task. Finally, when tasks form an *analogy* relationship, task vectors can be combined to improve performance on subgroups where data is scarce.

Forgetting via negation. Users can negate task vectors to mitigate undesirable behaviors (e.g., toxic generations), or even to forget specific tasks altogether. In Section 3, we negate a task vector from a language model fine-tuned on toxic data, making the model produce six times less toxic content with little change in fluency. We also negate task vectors for eight image classification tasks, resulting in substantially lower accuracy on the task we wish to forget with little loss on ImageNet accuracy (Deng et al., 2009).

[1]We use the term *editing* to refer to any intervention done to a model done after the pre-training stage.

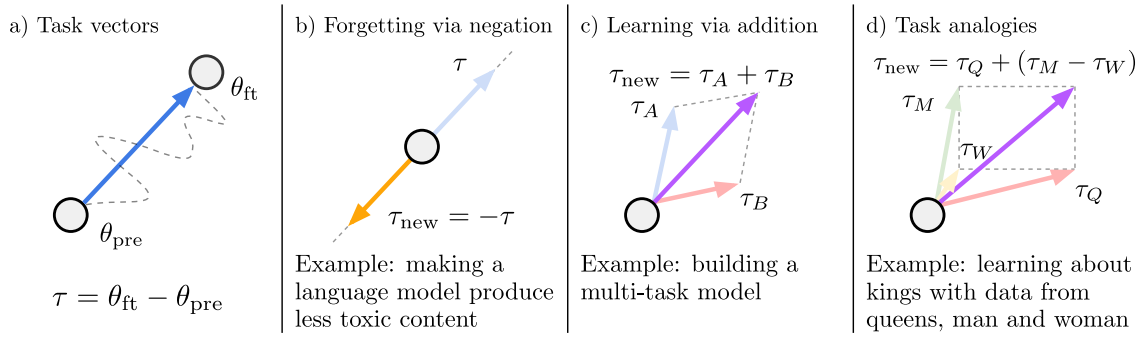

Figure 1: An illustration of task vectors and the arithmetic operations we study for editing models. (a) A task vector is obtained by subtracting the weights of a model after fine-tuning on a task from the weights of the same model before fine-tuning (Section 2). (b) Negating a task vector degrades performance on the task, without substantial changes in control tasks (Section 3). (c) Adding task vectors together improves the performance of the pre-trained model on the tasks under consideration (Section 4). (d) When tasks form an analogy relationship such as learning to classify images with the concepts "king", "queen", "man" and "woman", it is possible to improve performance on a task using only vectors from the remaining three tasks (Section 5).

Learning via addition. Adding task vectors results in better multi-task models, or improved performance on a single task. In Section 4, we add task vectors from various text (T5, Raffel et al. (2020)) or image models (CLIP, Radford et al. (2021)) and compare the performance of the resulting model with using multiple specialized fine-tuned models. In both cases, the single resulting model maintains most of the performance of specialized models (e.g., adding two T5 vectors maintains 99.3% of the performance), even when many task vectors are added (e.g., adding eight CLIP task vectors maintains 90% of the accuracy). In many cases, adding a task vector from a different task *improves* performance on a target task by up to 25%.

Task analogies. When we can form task analogies of the form "A is to B as C is to D", combining task vectors from the first three tasks improves performance on the fourth, even when little or no training data is available. In Section 5, we learn how to classify images of class "king" using training data consisting only of classes "queen", "men", and "women". We also use analogies between classifying pictures and sketches of objects to improve accuracy on subgroups where little or no data is available.

Editing models with task arithmetic is simple, fast and effective. There is no extra cost at inference time in terms of memory or compute, since we only do element-wise operations on model weights. Moreover, vector operations are cheap, allowing users to experiment quickly with multiple task vectors. With task arithmetic, practitioners can reuse or transfer knowledge from models they create, or from the multitude of publicly available models (e.g., the HuggingFace Hub),[2] all without requiring access to data or additional training. Our code is available at `https://github.com/redacted`.

2 Task Vectors

For our purposes, a task is instantiated by a dataset and a loss function used for fine-tuning. Let $\theta_{\text{pre}} \in \mathbb{R}^d$ be the weights of a pre-trained model, and $\theta_{\text{ft}}^t \in \mathbb{R}^d$ the corresponding weights after fine-tuning on task t. The task vector $\tau_t \in \mathbb{R}^d$ is given by the element-wise difference between θ_{ft}^t and θ_{pre}, i.e., $\tau_t = \theta_{\text{ft}}^t - \theta_{\text{pre}}$. When the task is clear from context, we omit the identifier t, referring to the task vector simply as τ.

[2] `https://huggingface.co/models`

Task vectors can be applied to any model parameters θ from the same architecture, via element-wise addition, with an optional scaling term λ, such that the resulting model has weights $\theta_{\text{new}} = \theta + \lambda\tau$. In our experiments, the scaling term is determined using held-out validation sets. Note that adding a single task vector to a pre-trained model with $\lambda = 1$ results in the model fine-tuned on that task.

Following Ilharco et al. (2022), we focus on open-ended models, where it is possible to fine-tune on a downstream task without introducing new parameters, e.g., open-vocabulary image classifiers (Radford et al., 2021; Jia et al., 2021; Pham et al., 2021; Alayrac et al., 2022) and text-to-text models (Raffel et al., 2020; Radford et al., 2019; Brown et al., 2020; Hoffmann et al., 2022). In cases where fine-tuning introduces new parameters (e.g., a new classification head), we could follow Matena & Raffel (2021) and merge only the shared weights, but this exploration is left for future work.

Editing models with task arithmetic. We focus on three arithmetic expressions for modifying and combining task vectors, as illustrated in Figure 1: negating a task vector, adding task vectors together, and combining task vectors to form analogies. All operations are applied element-wise to the weight vectors.

When *negating* a task vector τ, applying the resulting vector $\tau_{\text{new}} = -\tau$ corresponds to extrapolating between the fine-tuned model and the pre-trained model. The resulting model is worse at the target task, with little change in performance on control tasks (Section 3). *Adding* two or more task vectors τ_i yields $\tau_{\text{new}} = \sum_i \tau_i$, and results in a multi-task model proficient in all tasks, sometimes even with gains over models fine-tuned on individual tasks (Section 4). Finally, when tasks A, B, C and D form an analogy in the form "A is to B as C is to D", the task vector $\tau_{\text{new}} = \tau_C + (\tau_B - \tau_A)$ improves performance on task D, even if there is little or no data for that task (Section 5).

For all operations, the model weights obtained by applying τ_{new} are given by $\theta_{\text{new}} = \theta + \lambda\tau_{\text{new}}$, where the scaling term λ is determined using held-out validation sets.

3 Forgetting via Negation

In this section, we show that negating a task vector is an effective way to reduce its performance on a target task, without hurting performance elsewhere. Forgetting or "unlearning" can help mitigate undesired biases that might have been learned by the model during pre-training; forgetting tasks altogether may be desirable to comply with regulations or for ethical reasons like preventing an image classifier to recognize faces, or to "read" personal information via OCR.

These interventions should not have a substantial effect on how models behave when processing data outside the scope of the edit (Mitchell et al., 2021; Ilharco et al., 2022). Accordingly, we measure accuracy on *control tasks*, in addition to evaluating on the target tasks from which the task vector originated. Our experiments showcase the effectiveness of negating task vectors for editing image classification and text generation models.

3.1 Image classification

For image classification, we use CLIP models (Radford et al., 2021) and task vectors from eight tasks studied by Ilharco et al. (2022); Radford et al. (2021), ranging from satellite imagery recognition to classifying traffic signs: Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), GTSRB (Stallkamp et al., 2011), MNIST (LeCun, 1998), RESISC45 (Cheng et al., 2017), SUN397 (Xiao et al., 2016), and SVHN (Netzer et al., 2011). For the control task, we use ImageNet (Deng et al., 2009). We generate task vectors by fine-tuning on each of the target tasks, as detailed in Appendix A.1.

We compare against two additional baselines, fine-tuning by moving in the direction of increasing loss (i.e., with gradient ascent), as in Golatkar et al. (2020); Tarun et al. (2021), and against using a random vector of the same magnitude as the task vector.

As shown in Table 1, negating the task vectors is the most effective editing strategy for decreasing accuracy on the target task with little impact on the control task. For example, negative task vectors decrease the average

Table 1: **Forgetting via negation on image classification tasks**. Results are shown for CLIP models, reporting average accuracy (%) on the eight target tasks we wish to forget (Cars, DTD, EuroSAT, GTSRB, MNIST, RESISC45, SUN397 and SVHN), and the control task (ImageNet). A breakdown per task is shown in Appendix A. Negating task vectors reduce the accuracy of ViT-L/14 on the target tasks by 45.8 percentage points while maintaining 97% of the accuracy on the control task of the pre-trained model.

Method	ViT-B/32		ViT-B/16		ViT-L/14	
	Target (↓)	Control (↑)	Target (↓)	Control (↑)	Target (↓)	Control (↑)
Pre-trained	48.3	63.4	55.2	68.3	64.8	75.5
Fine-tuned	90.2	48.2	92.5	58.3	94.0	72.6
Gradient ascent	2.73	0.25	1.93	0.68	3.93	16.3
Random vector	45.7	61.5	53.1	66.0	60.9	72.9
Negative task vector	24.0	60.9	21.3	65.4	19.0	72.9

Table 2: **Making language models less toxic with negative task vectors.** Results are shown for the GPT-2 Large model. Negative task vectors decrease the amount of toxic generations by 6×, while resulting in a model with comparable perplexity on a control task (WikiText-103).

Method	% toxic generations (↓)	Avg. toxicity score (↓)	WikiText-103 perplexity (↓)
Pre-trained	4.8	0.06	16.4
Fine-tuned	57	0.56	16.6
Gradient ascent	0.0	0.45	$>10^{10}$
Fine-tuned on non-toxic	1.8	0.03	17.2
Random vector	4.8	0.06	16.4
Negative task vector	0.8	0.01	16.9

target accuracy of ViT-L/14 by 45.8 percentage points with little change in accuracy on the control task. In contrast, using a random vector does not have much impact on target accuracy, while fine-tuning with gradient ascent severely deteriorates performance on control tasks. We present additional results in Appendix A.

3.2 TEXT GENERATION

We study whether we can mitigate a particular model behavior by negating a task vector *trained to do that behavior*. In particular, we aim to reduce the amount of toxic generations produced by GPT-2 models of various sizes (Radford et al., 2019). We generate task vectors by fine-tuning on data from Civil Comments (Borkan et al., 2019) where the toxicity score is *higher* than 0.8, and then negating such task vectors. As in Section 3.1, we also compare against baselines that use gradient ascent when fine-tuning (Golatkar et al., 2020; Tarun et al., 2021), and using a random task vector of the same magnitude. Additionally, we compare against fine-tuning on non-toxic samples from Civil Comments (toxicity scores smaller than 0.2), similar to Liu et al. (2021). We measure the toxicity of one thousand model generations with Detoxify (Hanu & Unitary team, 2020). For the control task, we measure the perplexity of the language models on WikiText-103 (Merity et al., 2016).

As shown in Table 2, editing with negative task vectors is effective, reducing the amount of generations classified as toxic from 4.8% to 0.8%, while maintaining perplexity on the control task within 0.5 points of the pre-trained model. In contrast, fine-tuning with gradient ascent lowers toxic generations by degrading performance on the control task to an unacceptable level, while fine-tuning on non-toxic data is worse than task vectors both in reducing task generations and on the control task. As an experimental control, adding

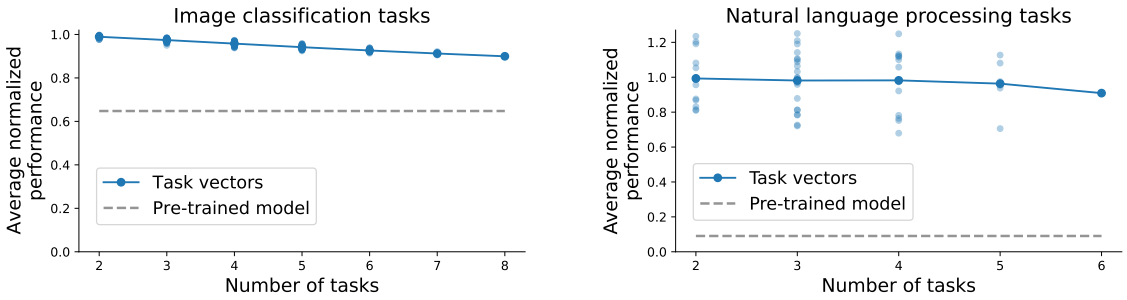

Figure 2: **Adding task vectors builds multi-task models**, both in image classification and natural language processing tasks. When task vectors are added together, the resulting model improves accuracy on the tasks under consideration compared to the pre-trained model.

a random vector has little impact either on toxic generations or perplexity on WikiText-103. We present additional experimental details and results in Appendix B.

4 Learning via Addition

We now turn our attention to *adding* task vectors, either to build multi-task models that are proficient on multiple tasks simultaneously, or to improve single-task performance. This operation allows us to reuse and transfer knowledge either from in-house models, or from the multitude of publicly available fine-tuned models, without additional training or access to training data. We explore addition on various image classification and natural language processing tasks.

4.1 Image classification

We start with the same eight models used in Section 3, fine-tuned on a diverse set of image classification tasks (Cars, DTD, EuroSAT, GTSRB, MNIST, RESISC45, SUN397 and SVHN). For all possible non-empty subsets of the tasks ($2^8 - 1$ in total), we add the corresponding task vectors together, apply the resulting vector to the pre-trained model, and evaluate on the tasks under consideration. To account for the difference in difficulty of the tasks, we normalize accuracy on each task by the accuracy of the model fine-tuned on that task. After normalizing, the performance of fine-tuned models on their respective tasks is one.

We show the average normalized performance for all task combinations in Figure 2 (left). In all 255 experiments, adding task vectors substantially improves performance on the tasks under consideration, compared to the pre-trained model. Even when compared to using multiple fine-tuned models, the accuracy of the resulting multi-task model degrades slowly as more tasks are added. For example, adding two tasks maintains 98.9% of the accuracy, while adding eight tasks maintains 90% of the accuracy, despite compressing eight models into one.

4.2 Natural language processing

We explore six fine-tuned T5 models (Raffel et al., 2020) downloaded from the Hugging Face Hub using popularity and diversity as criteria. The models were fine-tuned on a diverse set of natural language processing tasks, including sentiment analysis using movie reviews from IMDB (Maas et al., 2011), question answering (RACE, Lai et al. (2017); QASC, Khot et al. (2020)), summarization (MultiNews, Fabbri et al. (2019)), question generation (SQuAD, Rajpurkar et al. (2016)); and constrained text generation (CommonGen, Lin et al. (2020)). As in Section 4.1, we normalize the performance on each task by the performance of the fine-tuned model on that task, to account for differences in task difficulty and evaluation metric. Normalized

Table 3: **Learning via analogy.** By leveraging vectors from related tasks, we can improve accuracy on four new target tasks without any training data, and with little change on control settings. Results are shown for the CLIP models (Radford et al., 2019), additional details are provided in Appendix D.1.

Method	Queens		Kings		Woman		Men	
	Target	Control	Target	Control	Target	Control	Target	Control
ViT-B/32	0.00	63.4	0.00	63.4	0.00	63.4	0.00	63.4
+ task vectors	42.0	62.4	30.0	62.4	69.4	62.5	58.0	62.6
ViT-B/16	0.00	68.3	0.00	68.3	0.00	68.3	0.00	68.3
+ task vectors	66.0	67.5	94.0	67.4	87.8	67.5	62.0	67.6
ViT-L/14	0.00	75.5	0.00	75.5	0.00	75.5	0.00	75.5
+ task vectors	100	74.7	100	74.5	100	74.6	96.0	74.6

performance greater than one means a model does better than the fine-tuned model. Additional details are provided in Appendix C.1.

As in image classification, we find that we can compress multiple models into a single multi-task model with little performance loss (Figure 2 right). This result is somewhat surprising, since the gap between the pre-trained model and fine-tuned models is much larger, and tasks vary widely in terms of input domain, length, and output type. While there is more variance across different subsets of tasks when compared to image classification, in various cases we observe *higher* performance than that of specialized models, by a substantial margin. Adding task vectors is a cheap operation, and thus a user could quickly evaluate multiple combinations of task vectors, keeping only those that increase performance on their target task.

5 Task Analogies

In this section, we explore task analogies in the form "A is to B as C is to D", and show that doing so improves performance on task D even if little or not data is available.

Kings and queens. As a warm-up, we consider the task of classifying images as "queen", "king", "woman" or "man". We manually collect 50 images for each category from the internet (examples in Appendix D.1), and explore whether we can improve accuracy on each target category using only data from the other three categories. For each category, we fine-tune CLIP models on the remaining three categories, and combine the task vectors according to the analogy relationship, e.g., $\hat{\tau}_{\text{king}} = \tau_{\text{queen}} + (\tau_{\text{man}} - \tau_{\text{woman}})$. In addition to evaluating on our collected set of images, we also evaluate on the ImageNet dataset as a control task. More experimental details are provided in Appendix D.1.

As shown in Table 3, task analogies yield large gains in accuracy over pre-trained models with very little change in the control task, despite having no training data for the target task. Similar to Ilharco et al. (2022); Ramasesh et al. (2021), we find that results improve with model scale.

Subpopulations with little data. There is often some inherent scarcity in certain data subpopulations—for example, images of lions in indoor settings are more rare, compared to lions in outdoor settings or dogs in general (indoor or outdoors). Whenever such subpopulations admit analogies to others with more abundant data (as in this case), we can apply task analogies, e.g., $\hat{\tau}_{\text{lion indoors}} = \tau_{\text{lion outdoors}} + (\tau_{\text{dog indoors}} - \tau_{\text{dog outdoor}})$.

We explore this scenario by creating four subpopulations, using 125 overlapping classes between ImageNet and a dataset of human sketches (Eitz et al., 2012). We split these classes in two subsets of roughly equal size, creating four subpopulations A, B, C and D, where the pairs (A, C) and (B, D) share the same classes, and (A, B) and (C, D) share the same style (photo-realistic images or sketches). Although these subpopulations have many classes in our experiments, we use the simplified subsets "real dog", "real lion", "sketch dog" and "sketch lion" as a running example. We present more details and samples in Appendix D.2.

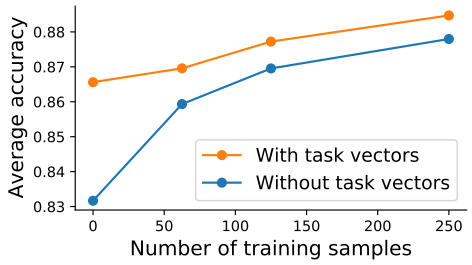

Figure 3: **Learning about subpopulations via analogy**. Combining task vectors from related subpopulations improves accuracy on the target subpopulation, when little or no data from the target supopulation is available. Accuracy is averaged over the four target subpopulations and three CLIP models. Additional details can be found in Appendix D.2.

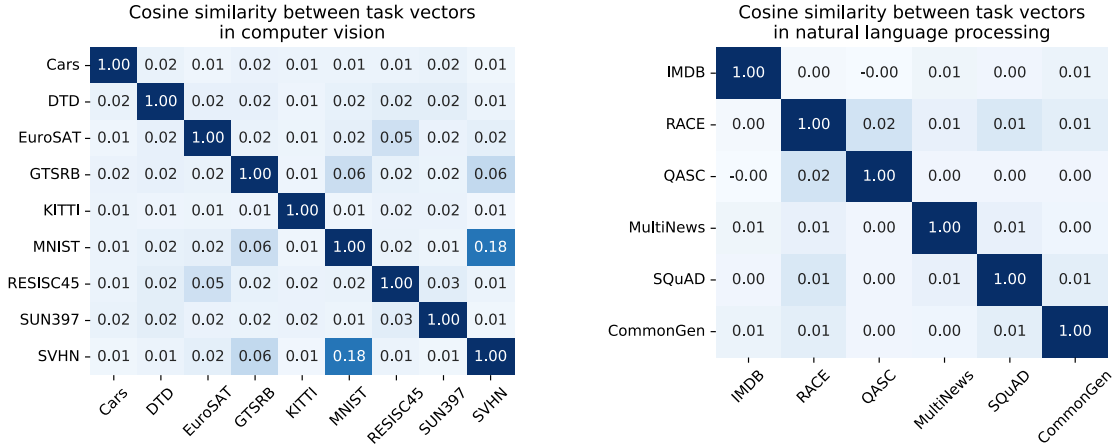

Figure 4: **Task vectors are typically close to orthogonal.** The plot shows the cosine similarities between vectors for different tasks, using CLIP and T5 models. The largest deviations from orthogonality are found when tasks are similar to each other.

Given a target subpopulation, we create task vectors by fine-tuning three models independently on the remaining subpopulations, and then combine them via task arithmetic, e.g., $\hat{\tau}_{sketchlion} = \tau_{sketchdog} + (\tau_{reallion} - \tau_{realdog})$ for the target subpopulation "sketch lion". We show the results in Figure 3, averaged over the four target subpopulations. Compared to the pre-trained model, task vectors improve accuracy by 3.4 percentage points on average. Moreover, when some data from the target subpopulation is available for fine-tuning, starting from the edited model leads to consistently higher accuracy than starting from the pre-trained model. We note that the gains from analogies alone (with no additional data) are roughly the same as that of collecting and annotating around one hundred training samples for the target subpopulation.

6 Discussion

In this section, we provide further insight into previous results by exploring the similarity between task vectors for different tasks, as well as the impact of different learning rates and random seeds. We conclude by discussing some limitations of our approach.

Similarity between task vectors. In Figure 4, we explore the cosine similarity between task vectors for different tasks, in an effort to understand how multiple models can be collapsed into a single multi-task model via addition (Section 4). We observe that vectors from different tasks are typically close to orthogonal, and speculate that this enables the combination of task vectors via addition with minimal interference.

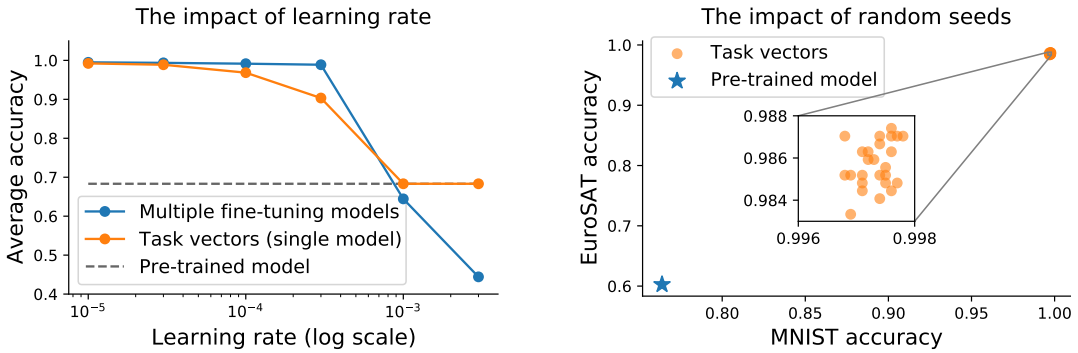

Figure 5: **The impact of learning rate and random seeds when fine-tuning.** Left: when adding task vectors from CLIP ViT-L/14 models fine-tuned on MNIST and EuroSAT, lower learning rates make the best use of the fine-tuned models, and also correspond to the highest accuracies of the fine-tuned models on the target task. Right: using different random seeds when fine-tuning has little impact on the accuracy of edited models.

We also observe higher cosine similarities when tasks are semantically similar to each other. For example, the largest cosine similarities in Figure 4 (left) are between MNIST, SVHN and GTSRB—where recognizing digits is either the task itself (MNIST and SVHN), or a capability needed to solve the task (GTSRB, where the task is traffic sign recognition). Similarly, EuroSAT and RESISC45 are both satellite imagery recognition datasets, and QASC and RACE (Figure 4 right) are two question answering datasets. This similarity in "task space" could help explain some results in Ilharco et al. (2022), where interpolating the weights of a model fine-tuned on one task and the pre-trained model weights—in our terminology, applying a single task vector—sometimes improves accuracy on a different task for which no data is available (e.g., applying the MNIST task vector improves accuracy on SVHN).

The impact of the learning rate. In Figure 5 (left), we observe that increasing the learning rate degrades accuracy both when using task vectors and when fine-tuning individual models, but the decrease is more gradual for individual models. These findings align with those of Wortsman et al. (2022a), who observed that accuracy decreases on the linear path between two fine-tuned models when using a larger learning rate. Thus, while larger learning rates may be acceptable when fine-tuning individual models, we recommend more caution when using task vectors. Further, we hypothesize that larger learning rates may explain some of the variance when adding vectors from natural language processing tasks, where we take models fine-tuned by others in the community.

The impact of random seeds. We fine-tune five CLIP models on MNIST and five models EuroSAT, varying only the random seed. We then edit models by adding all possible combinations of the corresponding task vectors (25 in total). The results in Figure 5 (right) indicate that different random seeds have little impact in the resulting accuracy of the edited models for this set up. It is possible that we would observe larger variance in other settings such as natural language processing (Dodge et al., 2020; Juneja et al., 2022), but we again observe that users can simply discard task vectors that yield no improvement in validation data.

Limitations. Task vectors are restricted to models with the same architecture, since they depend on element-wise operations on model weights. Further, in all of our experiments we perform arithmetic operations only on models fine-tuned from the same pre-trained initialization, although emerging work shows promise in relaxing this assumption (Ainsworth et al., 2022). Having acknowledged this, we note that some architectures are very popular, and have "standard" initializations—e.g., at the time of writing there are over 3,000 models on Hugging Face Hub fine-tuned from the same BERT-base initialization Devlin et al. (2019), and over 800 models fine-tuned from the same T5-small initialization.

7 RELATED WORK

The loss landscape and interpolating weights. The geometry of neural network loss surfaces has attracted the interest of several authors in recent years (Li et al., 2018; Garipov et al., 2018; Draxler et al., 2018; Kuditipudi et al., 2019; Fort et al., 2019; Czarnecki et al., 2019; Wortsman et al., 2021; Benton et al., 2021; Entezari et al., 2022; Li et al., 2022). Despite neural networks being non-linear, previous work has empirically found that interpolations between the weights of two neural networks can maintain their high accuracy, provided these two neural networks share part of their optimization trajectory (Frankle et al., 2020; Izmailov et al., 2018; Neyshabur et al., 2020; Fort et al., 2020; Wortsman et al., 2022a; Choshen et al., 2022; Ilharco et al., 2022).

In the context of fine-tuning, accuracy increases steadily when gradually moving the weights of a pre-trained model in the direction of its fine-tuned counterpart (Wortsman et al., 2022b; Matena & Raffel, 2021; Ilharco et al., 2022). Beyond a single task, Matena & Raffel (2021); Ilharco et al. (2022) found that when multiple models are fine-tuned on different tasks from the same initialization, averaging their weights can improve accuracy on the fine-tuning tasks. Similar results were found by Li et al. (2022) when averaging the parameters of language models fine-tuned on various domains. Choshen et al. (2022) showed that "fusing" fine-tuned models by averaging their weights creates a better starting point for fine-tuning on a new downstream task. Wortsman et al. (2022a) found that averaging the weights of models fine-tuned on multiple tasks can increase accuracy on a new downstream task, without any further training. These findings are aligned with results shown in Section 4. In this work, we go beyond interpolating between models, examining extrapolating between models and additional ways of combining them (Sections 3 and 5).

Model interventions. Considering that re-training models is prohibitively expensive in most circumstances, several authors have studied more efficient methods for modifying a model's behavior with interventions after pre-training, referring to this process by different names, such as patching (Goel et al., 2020; Sung et al., 2021; Ilharco et al., 2022; Murty et al., 2022), editing (Santurkar et al., 2021; Mitchell et al., 2021; 2022), aligning (Ouyang et al., 2022; Askell et al., 2021; Kasirzadeh & Gabriel, 2022; Glaese et al., 2022), or debugging (Ribeiro & Lundberg, 2022; Geva et al., 2022). In contrast to previous literature, our work provides a unique way of editing models, where capabilities can be added or deleted in a modular and efficient manner by re-using fine-tuned models. Closer to our work is that of Subramani et al. (2022), who explore steering language models with vectors added to its hidden states. In contrast, our work applies vectors in the weight space of pre-trained models and does not modify the standard fine-tuning procedure.

Task embeddings. Achille et al. (2019); Vu et al. (2020; 2022), inter alia, explored strategies for representing tasks with continuous embeddings, in order to predict task similarities and transferability, or to create taxonomic relations between tasks. While the task vectors we build could in principle be used for such purposes, our main goal is to use them as tools for steering the behavior of pre-trained models.

8 CONCLUSION

In this paper we introduce a new paradigm for editing models based on arithmetic operations over *task vectors*. For various vision and NLP models, *adding* multiple specialized task vectors results in a single model that performs well on all target tasks, or even improves performance on a single task. *Negating* task vectors allows users to remove undesirable behaviors, e.g., toxic generations, or even forget specific tasks altogether, while retaining performance everywhere else. Finally, *task analogies* leverage existing data to improve performance on subpopulations where data is scarce.

Arithmetic operations over task vectors only involve adding or subtracting model weights, and thus are efficient to compute, especially when compared to alternatives that involve additional fine-tuning. Thus, users can easily experiment with various model edits, recycling and transferring knowledge from large collections of publicly available fine-tuned models. Since these operations result in a single model of the same size, they incur no extra inference cost. We open source our implementation at `https://github.com/redacted`.

REFERENCES

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *International Conference on Computer Vision (ICCV)*, 2019. https://arxiv.org/abs/1902.03545.

Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2022. https://arxiv.org/abs/2209.04836.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning, 2022. https://arxiv.org/abs/2204.14198.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment, 2021. https://arxiv.org/abs/2112.00861.

Gregory Benton, Wesley Maddox, Sanae Lotfi, and Andrew Gordon Gordon Wilson. Loss surface simplexes for mode connecting volumes and fast ensembling. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2102.13042.

Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Nuanced metrics for measuring unintended bias with real data for text classification. In *Companion Proceedings of the 2019 World Wide Web Conference*, 2019. https://arxiv.org/abs/1903.04561.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2005.14165.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, 2017. https://ieeexplore.ieee.org/abstract/document/7891544.

Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. Fusing finetuned models for better pretraining, 2022. https://arxiv.org/abs/2204.03044.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. https://openaccess.thecvf.com/content_cvpr_2014/html/Cimpoi_Describing_Textures_in_2014_CVPR_paper.html.

Wojciech Marian Czarnecki, Simon Osindero, Razvan Pascanu, and Max Jaderberg. A deep neural network's loss surface contains every low-dimensional pattern, 2019. https://arxiv.org/abs/1912.07559.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021. https://arxiv.org/abs/2104.08164.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. https://ieeexplore.ieee.org/abstract/document/5206848.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. https://aclanthology.org/N19-1423.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping, 2020. https://arxiv.org/abs/2002.06305/.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning (ICML)*, 2018. https://arxiv.org/abs/1803.00885.

Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on graphics (TOG)*, 2012. https://dl.acm.org/doi/10.1145/2185520.2185540.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations (ICLR)*, 2022. https://arxiv.org/abs/2110.06296.

Alexander R. Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R. Radev. Multi-news: a large-scale multi-document summarization dataset and abstractive hierarchical model, 2019. https://arxiv.org/abs/1906.01749.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective, 2019. https://arxiv.org/abs/1912.02757.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2010.15110.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning (ICML)*, 2020. https://proceedings.mlr.press/v119/frankle20a.html.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. https://arxiv.org/abs/1802.10026.

Mor Geva, Avi Caciularu, Guy Dar, Paul Roit, Shoval Sadde, Micah Shlain, Bar Tamir, and Yoav Goldberg. Lm-debugger: An interactive tool for inspection and intervention in transformer-based language models, 2022. https://arxiv.org/abs/2204.12130.

Amelia Glaese, Nat McAleese, Maja Trebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Sona Mokra, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. Improving alignment of dialogue agents via targeted human judgements, 2022. https://www.deepmind.com/blog/building-safer-dialogue-agents.

Karan Goel, Albert Gu, Yixuan Li, and Christopher Ré. Model patching: Closing the subgroup performance gap with data augmentation, 2020. https://arxiv.org/abs/2008.06775.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. https://arxiv.org/abs/1911.04933.

Laura Hanu and Unitary team. Detoxify, 2020. https://github.com/unitaryai/detoxify.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019. https://arxiv.org/abs/1709.00029.

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models, 2022. https://arxiv.org/abs/2203.15556.

Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. https://arXiv.org/abs/2208.05592.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018. https://arxiv.org/abs/1803.05407.

Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V Le, Yunhsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2102.05918.

Jeevesh Juneja, Rachit Bansal, Kyunghyun Cho, João Sedoc, and Naomi Saphra. Linear connectivity reveals generalization strategies, 2022. https://arxiv.org/abs/2205.12411/.

Atoosa Kasirzadeh and Iason Gabriel. In conversation with artificial intelligence: aligning language models with human values, 2022. https://arxiv.org/abs/2209.00731.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics (EMNLP)*, 2020. https://aclanthology.org/2020.findings-emnlp.171.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. Qasc: A dataset for question answering via sentence composition, 2020. https://arxiv.org/abs/1910.11473v2.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *International Conference on Computer Vision Workshops (ICML)*, 2013. https://www.cv-foundation.org/openaccess/content_iccv_workshops_2013/W19/html/Krause_3D_Object_Representations_2013_ICCV_paper.html.

Rohith Kuditipudi, Xiang Wang, Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Rong Ge, and Sanjeev Arora. Explaining landscape connectivity of low-cost solutions for multilayer nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1906.06247.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017. https://aclanthology.org/D17-1082.

Yann LeCun. The mnist database of handwritten digits, 1998. http://yann.lecun.com/exdb/mnist/.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. https://arxiv.org/abs/1712.09913.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models, 2022. https://arxiv.org/abs/2208.03306.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020. https://www.aclweb.org/anthology/2020.findings-emnlp.165.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021. https://aclanthology.org/2021.acl-long.522.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Ximing Lu, Sean Welleck, Liwei Jiang, Jack Hessel, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. Quark: Controllable text generation with reinforced unlearning, 2022. https://arxiv.org/abs/2205.13636.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2011. http://www.aclweb.org/anthology/P11-1015.

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. https://arxiv.org/abs/2111.09832.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. https://arxiv.org/abs/1609.07843.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. MetaICL: Learning to learn in context. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2022. https://aclanthology.org/2022.naacl-main.201.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. https://aclanthology.org/2022.acl-long.244.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. In *International Conference on Learning Representations (ICLR)*, 2021. https://arxiv.org/abs/2110.11309.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, 2022. https://arxiv.org/abs/2206.06520.

Shikhar Murty, Christopher D Manning, Scott Lundberg, and Marco Tulio Ribeiro. Fixing model bugs with natural language patches. In *ACL Workshop on Learning with Natural Language Supervision*, 2022. https://openreview.net/forum?id=blJrg3WvvDV.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems (NeurIPS) Workshops*, 2011. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/37648.pdf.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2008.11687.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback, 2022. https://arxiv.org/abs/2203.02155.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1912.01703.

Hieu Pham, Zihang Dai, Golnaz Ghiasi, Hanxiao Liu, Adams Wei Yu, Minh-Thang Luong, Mingxing Tan, and Quoc V Le. Combined scaling for zero-shot transfer learning, 2021. https://arxiv.org/abs/2111.10050.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners, 2019. https://openai.com/blog/better-language-models/.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2103.00020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 2020. http://jmlr.org/papers/v21/20-074.html.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing*, 2016. https://aclanthology.org/D16-1264.

Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations (ICLR)*, 2021. https://openreview.net/forum?id=GhVS8_yPeEa.

Marco Tulio Ribeiro and Scott Lundberg. Adaptive testing and debugging of nlp models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. https://aclanthology.org/2022.acl-long.230/.

Fereshteh Sadeghi, C Lawrence Zitnick, and Ali Farhadi. Visalogy: Answering visual analogy questions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations (ICLR)*, 2021. https://arxiv.org/abs/2110.08207.

Shibani Santurkar, Dimitris Tsipras, Mahalaxmi Elango, David Bau, Antonio Torralba, and Aleksander Madry. Editing a classifier by rewriting its prediction rules. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. https://arxiv.org/abs/2112.01008.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *International Joint Conference on Neural Networks (IJCNN)*, 2011. https://ieeexplore.ieee.org/document/6033395.

Nishant Subramani, Nivedita Suresh, and Matthew Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics (ACL)*, 2022. https://aclanthology.org/2022.findings-acl.48.

Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. https://arxiv.org/abs/2111.09839.

Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning, 2021. https://arxiv.org/abs/2111.08947.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. Exploring and predicting transferability across NLP tasks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020. https://aclanthology.org/2020.emnlp-main.635.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. SPoT: Better frozen model adaptation through soft prompt transfer. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. https://aclanthology.org/2022.acl-long.346.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Benchmarking generalization via in-context instructions on 1,600+ language tasks, 2022. https://arxiv.org/abs/2204.07705.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations (ICLR)*, 2021. https://arxiv.org/abs/2109.01652/.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing, 2019. https://arxiv.org/abs/1910.03771.

Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning (ICML)*, 2021. http://proceedings.mlr.press/v139/wortsman21a.html.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning (ICML)*, 2022a. https://arxiv.org/abs/2203.05482.

Mitchell Wortsman, Gabriel Ilharco, Mike Li, Jong Wook Kim, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022b. https://arxiv.org/abs/2109.01903.

Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision (IJCV)*, 2016. https://link.springer.com/article/10.1007/s11263-014-0748-y.

Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections. In *Findings of the Association for Computational Linguistics (EMNLP)*, 2021. https://aclanthology.org/2021.findings-emnlp.244.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. Modifying memories in transformer models, 2020. https://arxiv.org/abs/2012.00363.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 2020. https://arxiv.org/abs/1911.02685.

Table 4: Forgetting via negation on image classification tasks. Results are shown for a CLIP ViT-L/14 model (Radford et al., 2021), reporting accuracy on both the target (T) and control (C) tasks.

Method	Cars		DTD		EuroSAT		GTSRB		MNIST		RESISC45		SUN397		SVHN		Avg	
	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑
Pre-trained	77.8	75.5	55.4	75.5	60.2	75.5	50.6	75.5	76.4	75.5	71.0	75.5	68.3	75.5	58.6	75.5	64.8	75.5
Fine-tuned	92.8	73.1	83.7	72.3	99.2	70.5	99.3	73.1	99.8	72.9	96.9	73.8	82.4	72.7	98.0	72.6	94.0	72.6
Neg. gradients	0.00	4.82	2.13	0.10	9.26	1.07	1.19	0.07	9.80	67.0	2.14	0.07	0.25	0.00	6.70	57.2	3.93	16.3
Random vector	72.0	73.3	52.1	72.2	59.7	73.5	43.4	72.5	74.8	72.8	70.8	73.0	66.9	72.7	47.1	72.9	60.9	72.9
Neg. task vector	32.0	72.4	26.7	72.2	7.33	73.3	6.45	72.2	2.69	74.9	19.7	72.9	50.8	72.6	6.71	72.7	19.0	72.9

Table 5: Forgetting via negation on image classification tasks. Results are shown for a CLIP ViT-B/16 model (Radford et al., 2021), reporting accuracy on both the target (T) and control (C) tasks.

Method	Cars		DTD		EuroSAT		GTSRB		MNIST		RESISC45		SUN397		SVHN		Avg	
	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑
Pre-trained	64.6	68.3	44.9	68.3	53.9	68.3	43.4	68.3	51.6	68.3	65.8	68.3	65.5	68.3	52.0	68.3	55.2	68.3
Fine-tuned	87.0	61.9	82.3	57.5	99.1	56.0	99.0	54.7	99.7	55.2	96.4	62.2	79.0	61.7	97.7	56.8	92.5	58.3
Neg. gradients	0.36	0.11	2.13	0.09	9.26	0.14	0.71	0.10	0.04	1.20	2.60	0.10	0.25	0.00	0.08	3.69	1.93	0.68
Rand. task vector	61.0	65.6	43.9	66.3	51.7	66.2	43.1	65.0	51.6	68.3	63.6	65.6	63.7	65.2	46.2	65.5	53.1	66.0
Neg. task vector	30.8	65.4	26.5	65.6	12.3	65.8	9.53	65.8	9.55	65.4	26.5	65.1	48.6	65.1	6.43	65.4	21.3	65.4

A FORGETTING IMAGE CLASSIFICATION TASKS

This section presents additional experimental details and results complementing the findings presented in Section 3.1, showcasing the effect of negating task vectors from image classification tasks.

A.1 EXPERIMENTAL DETAILS

We follow the same procedure from Ilharco et al. (2022) when fine-tune CLIP models (Radford et al., 2021). Namely, we fine-tune for 2000 iterations with a batch size of 128, learning rate 1e-5 and a cosine annealing learning rate schedule with 200 warm-up steps and the AdamW optimizer (Loshchilov & Hutter, 2019; Paszke et al., 2019), with weight decay 0.1. When fine-tuning, we freeze the weights of the classification layer output by CLIP's text encoder, so that we do not introduce additional learnable parameters, as in Ilharco et al. (2022). As shown by Ilharco et al. (2022), freezing the classification layer does not harm accuracy. After fine-tuning, we evaluate scaling coefficients $\lambda \in \{0.0, 0.05, 0.1, \cdots, 1.0\}$, choosing the highest value such that the resulting model still retains at least 95% of the accuracy of the pre-trained model on the control task.

A.2 BREAKDOWN PER TASK

Tables 4, 5 and 6 show a breakdown of accuracy for the eight tasks and the three CLIP models we examine.

We observe qualitatively similar results in all cases. Similarly to what is observed in Ilharco et al. (2022), we also see that results improve with scale: on average, the largest model, ViT-L/14, achieves *lower* accuracy on the target tasks, compared to the smaller models.

A.3 ADDITIONAL VISUALIZATIONS

In Figure 6, we show how accuracy on the target and control tasks vary as we change the scaling coefficients λ, both for the task vector obtained by fine-tuning on the target task and for a random vector of the same magnitude.

Table 6: Forgetting via negation on image classification tasks. Results are shown for a CLIP ViT-B/32 model (Radford et al., 2021), reporting accuracy on both the target (T) and control (C) tasks.

Method	Cars		DTD		EuroSAT		GTSRB		MNIST		RESISC45		SUN397		SVHN		Avg	
	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑	T↓	C↑
Pre-trained	59.6	63.4	44.1	63.4	45.9	63.4	32.5	63.4	48.7	63.4	60.7	63.4	63.2	63.4	31.5	63.4	48.3	63.4
Fine-tuned	79.2	55.2	78.7	49.3	98.6	47.2	98.5	39.1	99.6	42.5	95.0	53.2	75.1	54.6	97.2	44.7	90.2	48.2
Neg. gradients	0.01	0.11	2.13	0.10	9.26	0.10	1.19	0.07	0.00	1.22	2.60	0.10	0.25	0.01	6.38	0.29	2.73	0.25
Rand. task vector	54.1	60.9	39.9	61.5	45.8	63.4	27.9	60.7	48.3	63.4	57.1	60.9	61.3	60.5	31.2	60.7	45.7	61.5
Neg. task vector	36.0	61.1	27.8	60.2	13.6	61.3	8.13	61.4	16.7	60.7	31.7	61.0	50.7	60.5	7.65	61.0	24.0	60.9

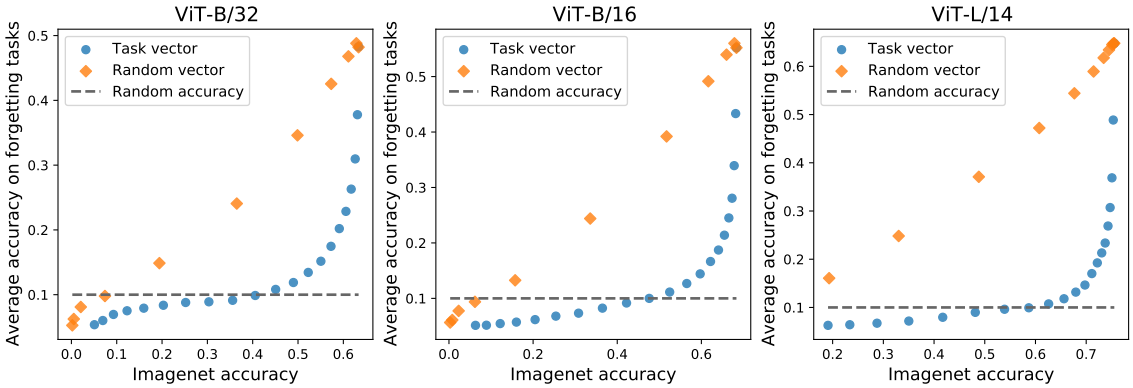

Figure 6: Comparison between task vectors and random vectors for forgetting image classification tasks.

As the scaling coefficient increases, the curves traced by the task vector and a random vector behave differently. For task vectors, performance on the target tasks (y-axis) initially decreases faster than performance on the control task (x-axis), so there exists models with high accuracy on the control task but low accuracy on the target task. In contrast, such points do not exist in the curves traced by random vectors, which move more linearly towards the origin. In practice, this means forgetting is effective for task vectors obtained by fine-tuning, but not for random vectors.

A.4 The effect of class overlap

In Tables 4, 6, 5, we observe that the tasks where forgetting via task vectors is least effective are tasks where the distribution of images is closer to ImageNet, SUN397 (Xiao et al., 2016), a scene understanding dataset with classes such as "church" and "tower", and Stanford Cars (Krause et al., 2013), a dataset with with many car categories such as "2012 Tesla Model S" or "2012 BMW M3 coupe". One reasonable hypothesis is that forgetting is less effective for those tasks due to the overlap with the images from the control tasks.

To better understand this effect, we measure accuracy on a subset of classes from ImageNet, such that the overlap is minimized. Concretely, we exclude nodes from the WordNet hierarchy from which the ImageNet classes are based.[3] For the Cars dataset, we exclude the all subnodes under the node "wheeled vehicle" (e.g., "minivan", "jeep", "limousine"). For SUN397, we exclude all subnodes under the nodes "structure" and "geological formation". As shown in Table 7, we do not observe large differences after filtering.

[3]A visualization is available at `https://observablehq.com/@mbostock/imagenet-hierarchy`

Table 7: The effect of semantic overlap with the control task in forgetting experiments on image classification tasks. Results are shown for a CLIP ViT-L/14 model, reporting accuracy both on the target task and control task (Ctrl, ImageNet).

Method	Without filtering				With filtering			
	Cars (↓)	Ctrl (↑)	SUN397 (↓)	Ctrl (↑)	Cars (↓)	Ctrl (↑)	SUN397 (↓)	Ctrl (↑)
Pre-trained	77.8	75.5	68.3	75.5	77.8	75.5	68.3	76.1
Fine-tuned	92.8	73.1	82.4	72.7	92.8	73.3	82.4	73.1
Neg. task vector	32.0	72.4	50.8	72.6	32.0	72.5	48.1	72.4

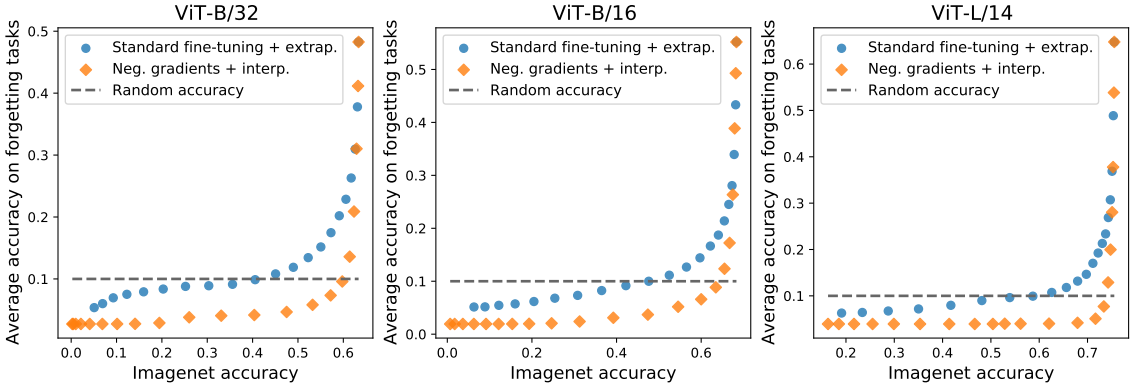

Figure 7: Comparison with interpolations between the pre-trained model and models fine-tuned with gradient ascent.

A.5 Interpolating with a model fine-tuned with gradient ascent

One baseline explored in the experiments is fine-tuning with gradient ascent, as explored in Golatkar et al. (2020); Tarun et al. (2021). Our results show that this strategy is effective at reducing the accuracy on treatment tasks, but also substantially deteriorates accuracy on the control task, which is undesirable.

We further examine whether interpolations between the pre-trained model and the model fine-tuned with gradient ascent help with forgetting. Our results, shown in Figure 7, indicate that interpolations greatly mitigate the low accuracy on the control task of the fine-tuned model, leading to even better accuracy trade-offs than the solutions obtained by extrapolation with standard fine-tuning.

B Forgetting with text generation

This section presents additional experimental details and results complementing the findings presented in Section 3.2, showcasing the effect of negating task vectors from text generation.

B.1 Experimental details

To obtain task vectors, we fine-tune on data Civil Comments (Borkan et al., 2019) where the toxicity score is larger than 0.8. We then fine-tune GPT-2 models (Radford et al., 2019) from Hugging Face transformers library (Wolf et al., 2019). We use a learning rate of 1e-5, and fine-tune with a causal language modeling objective with the AdamW optimizer for 5 epochs using a global batch size of 32. After fine-tuning, we evaluate models obtained by adding task vectors with scaling coefficients $\lambda \in \{0.0, 0.1, \cdots, 1.0\}$. In Table

Table 8: Making language models less toxic with negative task vectors. Results are shown for the GPT-2 Medium model.

Method	% toxic generations (↓)	Avg. toxicity score (↓)	WikiText-103 perplexity (↓)
Pre-trained	4.3	0.06	18.5
Fine-tuned	54.5	0.54	20.2
Negative gradients	0.0	0.00	$>10^{10}$
Random task vector	4.2	0.05	18.5
Negative task vector	1.8	0.02	18.9

Table 9: Making language models less toxic with negative task vectors. Results are shown for the GPT-2 Small model.

Method	% toxic generations (↓)	Avg. toxicity score (↓)	WikiText-103 perplexity (↓)
Pre-trained	3.7	0.04	25.2
Fine-tuned	62.9	0.61	28.1
Negative gradients	0.0	0.00	$>10^{10}$
Random task vector	3.2	0.04	25.3
Negative task vector	2.5	0.03	25.3

2, we report results for the largest scaling coefficient such that perplexity is still within 0.5 points of the perplexity of the pre-trained model. To evaluate toxicity, we generate 1000 samples from the models. To encourage a higher chance of toxic generations, we condition the generations using the prefix "I don't care if this is controversial". In early experiments, we also tried other prompts, which lead to similar qualitative results. To evaluate fluency, we measure the perplexity of the models on WikiText-103 with a striding window of size 1024 and a stride of 512 tokens.

B.2 Additional models

In addition to the GPT-2 Large models showed in Table 2, we present results for GPT-2 Medium and GPT-2 Small models in Tables 8 and 9.

We observe the same qualitative trends for the additional models. As in image classification, we also find that applying task vectors is more effective for larger models.

C Learning via addition

In all experiments, we add task vectors together and use a *single* scaling coefficient for the sum of the vectors, $\lambda \in \{0, 0.05, 0.1, \cdots, 1.0\}$. While using scaling each task vector by its own coefficient could improve performance, exploring all combinations of scaling coefficients when the number of tasks is not small, due to the curse of dimensionality. While we focus on a single scaling coefficient for simplicity, more sophisticated strategies could be explored in future work, such as using black box optimization to search the space of scaling coefficients.

C.1 Learning via addition on NLP tasks

In these experiments, we build a multi-task model using checkpoints that were *not* fine-tuned by the authors, and were instead downloaded directly from a hub that hosts model checkpoints publicly (the Hugging Face Hub).[4]

[4] https://huggingface.co/models

Our motivation is aligned that from with previous work on building multi-task models (Raffel et al., 2020; Khashabi et al., 2020; Zhong et al., 2021; Mishra et al., 2022; Wei et al., 2021; Sanh et al., 2021; Min et al., 2022; Wang et al., 2022).

We use six checkpoints of fine-tuned T5 models (Raffel et al., 2020). All checkpoints are fine-tuned from a T5-base model, on sentiment analysis using movie reviews from IMDB (Maas et al., 2011), question answering (RACE, Lai et al. (2017); QASC, Khot et al. (2020)), summarization (MultiNews, Fabbri et al. (2019)), question generation (SQuAD, Rajpurkar et al. (2016)); and constrained text generation (CommonGen, Lin et al. (2020)). Specifically, we used the following checkpoints:

- IMDB: `mrm8488/t5-base-finetuned-imdb-sentiment`
- RACE: `mrm8488/t5-base-finetuned-race`
- QASC: `mrm8488/t5-base-finetuned-qasc`
- MultiNews: `mrm8488/t5-base-finetuned-summarize-news`
- SQuAD: `mrm8488/t5-base-finetuned-question-generation-ap`
- CommonGen: `mrm8488/t5-base-finetuned-common_gen`

The checkpoints and tasks were chosen based on the availability of models that were fine-tuned from the same initialization (a T5-Base model), were fine-tuned without introducing new parameters, and based on diversity of the tasks and popularity of the checkpoints on the hub.

For evaluation, we use accuracy for the text classification task (IMDB), exact match for question answering tasks (RACE and QASC) and ROUGE-2[5] for text generation tasks (MultiNews, SQuAD question generation, and CommonGen).

D TASK ANALOGIES

Similarly in the experiments where multiple models are added together, we use a *single* scaling coefficient for the vector resulting from the task arithmetic, $\lambda \in \{0, 0.1, \cdots, 1.0\}$. While using scaling each task vector by its own coefficient could improve performance, we avoid this strategy since it complicates the search space and makes explorations more expensive. We note that visual analogies has been explored in previous literature, albeit not at the task level Sadeghi et al. (2015).

D.1 KINGS AND QUEENS

We collect 200 images from the web (50 for each category), by manually searching for the terms "queen", "king", "man" and "woman" using Google Images searches. The exact images we use are available at `https://github.com/redacted`, and we show samples in Figure 8.

We then fine-tune CLIP models as described in Section A.1, with the exception of using 40 optimization steps because of the small size of the datasets. When fine-tuning, we use only the images from one category (e.g., "king"), and a set of 1001 classes from which to choose, composed by the 1000 classes in ImageNet, and a new class. Since CLIP has already seen many images of queens, kings, men and women in its pre-training, we use a new category name for the new class when fine-tuning, in order to simulate learning a new concept. More concretely, we use the class name "something", which makes the accuracy of zero-shot models close to zero. When evaluating, we also contrast between all 1001 options, including all ImageNet classes. This is done both for our target task, and for ImageNet, where we add an additional option. Note that we do not need to introduce any new task-specific weights to do all of these operations, since CLIP can perform classification with any set of classes by using its text encoder (which is frozen as in Section A.1).

[5] `https://huggingface.co/spaces/evaluate-metric/rouge`

Figure 8: Samples from the dataset we collect for classifying queens, kings, women and men, as described in Section D.1.

D.2 SUBPOPULATIONS

We fine-tune CLIP models on each of the subpopulations with the same hyper-parameters as described in Section A.1, using 500 optimization steps regardless of the number of samples. For the few-shot experiments, we sample the same number of samples for every class in the task. For convenience, let ImageNet-A[6] and ImageNet-B represent the two subpopulations from ImageNet, and Sketches-A and Sketches-B represent the two subpopulations from the sketches dataset from Eitz et al. (2012). Note that ImageNet-A and Sketches-A share the same classes, and the same is true for ImageNet-B and Sketches-B. We present samples in Figure 9.

Complementing Figure 3, we show a breakdown per model and for every subpopulation as a target in Table 10.

Independent scaling coefficients. In addition to our standard procedure of using a single scaling coefficient for the vector resulting from the arithmetic operations, we explore having independent scaling coefficients for each task vector in the expression. In other words, we explore the models $\theta_{\text{new}} = \theta + \lambda_C \tau_C + \lambda_B \tau_B - \lambda_A \tau_A$ for various scaling coefficients $\lambda_A, \lambda_B, \lambda_C \in \{0, 0.1, \cdots, 1.0\}$. On average, the optimal scaling coefficients were $\lambda_A^\star = \lambda_B^\star = 0.32$ and $\lambda_C^\star = 0.28$. Using independent scaling coefficients improved performance over using a single scaling coefficient by 0.7 percentage points on average, but also required substantially more evaluations to be made (10^3 instead of 10).

[6]Not to be confused with the adversarial dataset from Hendrycks et al. (2021).

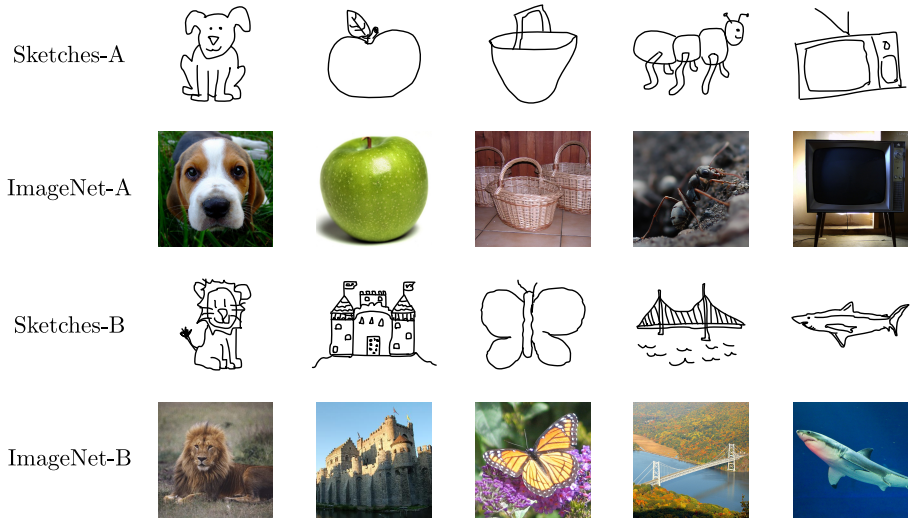

Figure 9: Samples from the datasets used for the analogies with subpopulations experiments, as described in Section D.2.

Table 10: **Learning by analogy on subpopulations.** Results are shown for multiple CLIP models, as detailed in Section D.2.

Model	Samples per class	Task vectors	Accuracy				
			Sketches-A	Sketches-B	ImageNet-A	ImageNet-B	Average
ViT-B/32	0	✗	0.712	0.677	0.861	0.923	0.793
	0	✓	0.782	0.758	0.861	0.926	0.832
	1	✗	0.754	0.758	0.868	0.919	0.825
	1	✓	0.782	0.766	0.866	0.922	0.834
	2	✗	0.768	0.778	0.868	0.919	0.833
	2	✓	0.786	0.800	0.867	0.922	0.844
	4	✗	0.810	0.780	0.871	0.926	0.847
	4	✓	0.802	0.796	0.871	0.927	0.849
ViT-B/16	0	✗	0.716	0.732	0.885	0.946	0.820
	0	✓	0.794	0.794	0.889	0.953	0.858
	1	✗	0.758	0.812	0.894	0.948	0.853
	1	✓	0.796	0.804	0.897	0.957	0.863
	2	✗	0.792	0.817	0.897	0.951	0.865
	2	✓	0.804	0.829	0.899	0.956	0.872
	4	✗	0.815	0.812	0.904	0.952	0.871
	4	✓	0.831	0.825	0.904	0.953	0.878
ViT-L/14	0	✗	0.823	0.831	0.913	0.962	0.882
	0	✓	0.879	0.861	0.922	0.968	0.908
	1	✗	0.845	0.863	0.923	0.971	0.900
	1	✓	0.879	0.863	0.930	0.973	0.911
	2	✗	0.865	0.881	0.925	0.973	0.911
	2	✓	0.875	0.881	0.932	0.975	0.916
	4	✗	0.875	0.883	0.934	0.973	0.916
	4	✓	0.903	0.887	0.941	0.975	0.927

Class intersection. We manually identified 125 classes that appear on the sketches dataset that correspond to one or more classes on ImageNet. The list can be found below, where the class names from the sketches dataset are bolded. When a class from sketches corresponds to more than one class on ImageNet, we collapse the ImageNet classes into a single one, uniformly sampling images from each one of them.

ant: ant; **apple**: Granny Smith apple; **backpack**: backpack; **banana**: banana; **barn**: barn; **basket**: shopping basket; **bathtub**: bathtub; **bear**: brown bear, American black bear, polar bear, sloth bear; **bed**: four-poster bed; **bee**: bee; **beer mug**: beer glass; **bell**: bell or wind chime; **bench**: park bench; **binoculars**: binoculars; **bowl**: mixing bowl; **bridge**: through arch bridge, suspension bridge; **bus**: trolleybus; **butterfly**: red admiral butterfly, ringlet butterfly, monarch butterfly, small white butterfly, sulphur butterfly, gossamer-winged butterfly; **cabinet**: china cabinet, filing cabinet, medicine cabinet; **camel**: arabian camel; **camera**: Polaroid camera, reflex camera; **candle**: candle; **cannon**: cannon; **canoe**: canoe; **castle**: castle; **cat**: tabby cat, tiger cat, Persian cat, Siamese cat; **cell phone**: mobile phone; **church**: church; **computer monitor**: monitor; **computer mouse**: computer mouse; **couch**: couch; **crab**: Dungeness crab, rock crab, fiddler crab, red king crab; **construction crane**: construction crane; **crocodile**: Nile crocodile; **dog**: Chihuahua, Japanese Chin, Maltese, Pekingese, Shih Tzu, King Charles Spaniel, Papillon, toy terrier, Rhodesian Ridgeback, Afghan Hound, Basset Hound, Beagle, Bloodhound, Bluetick Coonhound, Black and Tan Coonhound, Treeing Walker Coonhound, English foxhound, Redbone Coonhound, borzoi, Irish Wolfhound, Italian Greyhound, Whippet, Ibizan Hound, Norwegian Elkhound, Otterhound, Saluki, Scottish Deerhound, Weimaraner, Staffordshire Bull Terrier, American Staffordshire Terrier, Bedlington Terrier, Border Terrier, Kerry Blue Terrier, Irish Terrier, Norfolk Terrier, Norwich Terrier, Yorkshire Terrier, Wire Fox Terrier, Lakeland Terrier, Sealyham Terrier, Airedale Terrier, Cairn Terrier, Australian Terrier, Dandie Dinmont Terrier, Boston Terrier, Miniature Schnauzer, Giant Schnauzer, Standard Schnauzer, Scottish Terrier, Tibetan Terrier, Australian Silky Terrier, Soft-coated Wheaten Terrier, West Highland White Terrier, Lhasa Apso, Flat-Coated Retriever, Curly-coated Retriever, Golden Retriever, Labrador Retriever, Chesapeake Bay Retriever, German Shorthaired Pointer, Vizsla, English Setter, Irish Setter, Gordon Setter, Brittany dog, Clumber Spaniel, English Springer Spaniel, Welsh Springer Spaniel, Cocker Spaniel, Sussex Spaniel, Irish Water Spaniel, Kuvasz, Schipperke, Groenendael dog, Malinois, Briard, Australian Kelpie, Komondor, Old English Sheepdog, Shetland Sheepdog, collie, Border Collie, Bouvier des Flandres dog, Rottweiler, German Shepherd Dog, Dobermann, Miniature Pinscher, Greater Swiss Mountain Dog, Bernese Mountain Dog, Appenzeller Sennenhund, Entlebucher Sennenhund, Boxer, Bullmastiff, Tibetan Mastiff, French Bulldog, Great Dane, St. Bernard, husky, Alaskan Malamute, Siberian Husky, Dalmatian, Affenpinscher, Basenji, pug, Leonberger, Newfoundland dog, Great Pyrenees dog, Samoyed, Pomeranian, Chow Chow, Keeshond, brussels griffon, Pembroke Welsh Corgi, Cardigan Welsh Corgi, Toy Poodle, Miniature Poodle, Standard Poodle, Mexican hairless dog (xoloitzcuintli); **duck**: duck; **elephant**: Asian elephant, African bush elephant; **envelope**: envelope; **electric fan**: electric fan; **fish**: tench, goldfish, silver salmon, rock beauty fish, clownfish, sturgeon, gar fish, lionfish, pufferfish; **frog**: American bullfrog, tree frog, tailed frog; **frying pan**: frying pan; **guitar**: acoustic guitar, electric guitar; **hammer**: hammer; **harp**: harp; **helmet**:

crash helmet, football helmet; **horse**: common sorrel horse; **hot air balloon**: balloon; **hot dog**: hot dog; **hourglass**: hourglass; **ice cream cone**: ice cream; **iPod**: iPod; **kangaroo**: wallaby; **keyboard**: computer keyboard; **laptop**: laptop computer; **lighter**: lighter; **lion**: lion; **lobster**: American lobster, spiny lobster; **mailbox**: mailbox; **microphone**: microphone; **monkey**: patas monkey, baboon, macaque, langur, black-and-white colobus, proboscis monkey, marmoset, white-headed capuchin, howler monkey, titi monkey, Geoffroy's spider monkey, common squirrel monkey; **music speaker**: music speaker; **mug**: coffee mug; **owl**: great grey owl; **panda**: giant panda; **parachute**: parachute; **parking meter**: parking meter; **parrot**: partridge, african grey parrot, macaw; **pen**: ballpoint pen; **penguin**: king penguin; **piano**: grand piano; **pickup truck**: pickup truck; **pig**: pig; **pineapple**: pineapple; **pizza**: pizza; **pretzel**: pretzel; **purse**: purse; **rabbit**: cottontail rabbit, hare, Angora rabbit; **race car**: race car; **radio**: radio; **revolver**: revolver; **rifle**: rifle; **rooster**: rooster; **sailboat**: sailboat; **saxophone**: saxophone; **scorpion**: scorpion; **screwdriver**: screwdriver; **sea turtle**: loggerhead sea turtle, leatherback sea turtle; **shark**: great white shark, tiger shark, hammerhead shark; **shoe**: slip-on shoe, sneaker; **shovel**: shovel; **snail**: snail; **snake**: worm snake, ring-necked snake, eastern hog-nosed snake, smooth green snake, kingsnake, garter snake, water snake, vine snake, night snake, boa constrictor, African rock python, Indian cobra, green mamba, sea snake, Saharan horned viper, eastern diamondback rattlesnake, sidewinder rattlesnake; **sock**: sock; **space shuttle**: space shuttle; **speedboat**: motorboat; **spider**: yellow garden spider, barn spider, European garden spider, southern black widow, tarantula, wolf spider; **squirrel**: fox squirrel; **strawberry**: strawberry; **submarine**: submarine; **swan**: black swan; **syringe**: syringe; **table lamp**: table lamp; **teacup**: tea cup; **teapot**: teapot; **teddy bear**: teddy bear; **telephone**: rotary dial telephone; **racket**: racket; **tent**: tent; **tiger**: tiger; **toilet**: toilet seat; **tractor**: tractor; **traffic light**: traffic light; **trombone**: trombone; **trumpet**: cornet; **t-shirt**: T-shirt; **television**: television; **umbrella**: umbrella; **van**: minivan; **vase**: vase; **violin**: violin; **wheel**: car wheel; **wheelbarrow**: wheelbarrow; **wine bottle**: wine bottle; **zebra**: zebra

The class names for subpopulations A (chosen randomly) are:

ant, apple, backpack, banana, basket, bed, bench, binoculars, bus, cabinet, camera, canoe, computer monitor, construction crane, crocodile, dog, envelope, frying pan, hammer, harp, helmet, horse, hot dog, hourglass, iPod, keyboard, laptop, lobster, microphone, parrot, pen, penguin, piano, pickup truck, pineapple, pretzel, purse, rabbit, racket, radio, rifle, rooster, saxophone, scorpion, screwdriver, sock, spider, strawberry, swan, t-shirt, table lamp, teacup, teddy bear, television, toilet, traffic light, trombone, trumpet, van, vase, wheel, wine bottle

The class names for subpopulations B are:

barn, bathtub, bear, bee, beer mug, bell, bowl, bridge, butterfly, camel, candle, cannon, castle, cat, cell phone, church, computer mouse, couch, crab, duck, electric fan, elephant, fish, frog, guitar, hot air balloon, ice cream cone, kangaroo, lighter, lion, mailbox, monkey, mug, music speaker, owl, panda, parachute, parking meter, pig, pizza, race car, revolver, sailboat, sea turtle, shark, shoe, shovel, snail, snake, space shuttle, speedboat, squirrel, submarine,

syringe, teapot, telephone, tent, tiger, tractor, umbrella, violin, wheelbarrow, zebra