A METHODS

A.1 MODELS

Here we detail the model architectures used in this study. We use standard versions of the popular VGG (Simonyan & Zisserman, 2014), ResNet (He et al., 2015), and DenseNet (Huang et al., 2016) architectures. These architectures have varying structural properties (e.g., presence or absence of skip connections), meaning observed behaviors common to all of them are likely to be robust and generalizable.

VGG: The VGG model we use consists of five stages. Each stage comprises a convolutional layer, a ReLU nonlinearity, another convolutional layer, another ReLU nonlinearity, and finally a MaxPool layer. Each convolution uses a 3-by-3 kernel with unit stride and padding. The MaxPool operation uses a 2-by-2 kernel with stride 2. The number of channels by stage is 16, 32, 64, 128, 128. We do not use batch normalization in the VGG.

We test two different version of the VGG: one with just the stages described above, and another with two fully connected layers, of width 1024, after the convolutional layers.

ResNet: Our ResNet consists of a initial 3-by-3 convolutional layer with 32 channels, followed by a 2d batch norm operation. This initial pair is followed by four stages, each consisting of two residual blocks per stage. Each block consists of two conv-BN-ReLU sequences, with a shortcut layer directly routing the input to add to the final ReLU preactivations. All convolutions are 3-by-3, with unit padding. In all but the first block, the first convolutional layer downsamples, with a stride of 2; the second convolutional layer maintains stride of 1. The number of channels doubles each block, starting from a base level of 32 channels in the first block.

DenseNet: Like ResNet, our DenseNet consists of an initial 3-by-3 convolution, followed by four dense blocks. In between each pair of dense blocks is a transition block. Following the final dense block is a batch-normalization, ReLU, and average pooling operation to generate the final features. Our DenseNet is characterized by a growth rate of 12 and compression rate of 0.5. The first stage of the DenseNet features 6 blocks, with the number of blocks doubling in each subsequent stage.

A.2 TRAINING

All networks are trained using cross-entropy loss with SGD with momentum (β = 0.9), using a batch size 128. We do not use learning-rate schedules here, leaving that investigation to future work. To better correspond with practical situations, however, we choose a learning rate and total number of epochs such that interpolation (of the training set) occurs. For the split CIFAR-10 task, this typically corresponds to training for 30 epochs per task with a learning rate of 0.01 (VGG), 0.03 (ResNet), and 0.03 (DenseNet). We use weight decay with strength 1e-4, and do not apply any data augmentation. For the split CIFAR-100 task, we usually train for 60 or 90 epochs per task, with other the other hyperparameters identical to the CIFAR-10 case.

For multi-head networks (used in split CIFAR-10), each head is initialized with weights drawn from a normal distribution with variance $1/n_f$, where n_f is the number of features; biases are initialized to zero. We do not copy the parameters from the old head to the new when switching tasks.

For the split CIFAR-10 setup, the plots shown in the main text refer to experiments in which the initial task was classifying between the five categories *airplane*, *automobile*, *bird*, *cat*, and *deer*; and the second task comprising *dog*, *frog*, *horse*, *ship*, and *truck*. To verify that our results were not unique to the particular split of CIFAR-10 we chose, we used three other splits: *airplane*, *bird*, *deer*, *frog*, *ship* (task 1) and *automobile*, *cat*, *dog*, *horse*, *truck* (task 2); *automobile*, *cat*, *dog*, *horse*, *truck* (task 1) and *airplane*, *bird*, *deer*, *frog*, *ship* (task 2); and *dog*, *frog*, *horse*, *ship*, *truck* (task 1) and *airplane*, *automobile*, *bird*, *cat*, *deer* (task 2).

A.3 CIFAR-100 DISTRIBUTION SHIFT TASK

Here we provide the concrete example of the CIFAR-100 distribution-shift task shown in Figure 1. This is meant to capture it the scenario where catastrophic forgetting arises through *input distribution shift* — the input data to the neural network undergoes a distribution shift, causing the neural network to perform poorly on earlier data distributions (Arivazhagan et al., 2019; Snoek et al., 2019; Rabanser et al., 2019). As mentioned in the main text, in this task, the model must identify, for CIFAR-100 images, which of the 20 superclasses the image belongs to. The difference in tasks comes from the difference in subclasses which make up the superclass. As an example, we take the five superclasses *aquatic mammals*, *fruits and vegetables*, *household electrical devices*, *trees*, and *vehicles-1*[1], with the corresponding task 1 subclasses (1) dolphin, (2) apple, (3) lamp, (4) maple tree, and (5) bicycle and task 2 subclasses (1) whale, (2) orange, (3) television, (4) willow, and (5) motorcycle. A key feature of this setting is that task-specific components (including multiple heads) are precluded since the model does not know the task identity either at inference or time. While we do not explore it here, this setup also allows for continuously varying the data distribution, another situation likely to occur in practice.

A.4 REPRESENTATIONAL SIMILARITY AND CENTERED KERNEL ALIGNMENT

To understand properties of neural network hidden representations, we turn to representational similarity algorithms. A key challenge in analyzing neural network hidden representations is the lack of *alignment* — no natural correspondence exists between hidden neurons across different neural networks. Representational similarity algorithms propose different ways to overcome this — one of the first such algorithms, SVCCA (Raghu et al., 2017; Morcos et al., 2018), uses a Canonical Correlation Analysis (CCA) step to align neurons (enable invariance) through linear transformations.

We use centered kernel alignment (CKA), proposed by Kornblith et al. (2019), to measure the similarity between two representations of the same dataset which is invariant to orthogonal transformation and isotropic scaling (but not arbitrary linear transformations). Given a dataset of m examples, we compare two representations X and Y of that dataset (say, from two different neural networks or from the same neural network with different parameters), with n_x and n_y features respectively; that is, $X \in \mathbb{R}^{m \times n_x}$ and $Y \in \mathbb{R}^{m \times n_y}$. Then the linear-kernel CKA similarity between the two representations is given by

$$\mathrm{CKA}(X, Y) = \frac{||X^T Y||_F^2}{||X^T X||_F^2 ||Y^T Y||_F^2} \tag{4}$$

A.5 SUBSPACE SIMILARITY

For the subspace similarity computations we pick a fixed threshold k for all layers so that the first k principal components capture 80% of variance in the final layer. This gives $k = 3$ for Resnet on CIFAR-10, $k = 4$ for Resnet on CIFAR-100, $k = 4$ for VGG on CIFAR-10 and $k = 7$ for VGG on CIFAR-100 and $k = 4$ for DenseNet on CIFAR-10 and $k = 5$ for DenseNet on CIFAR-100.

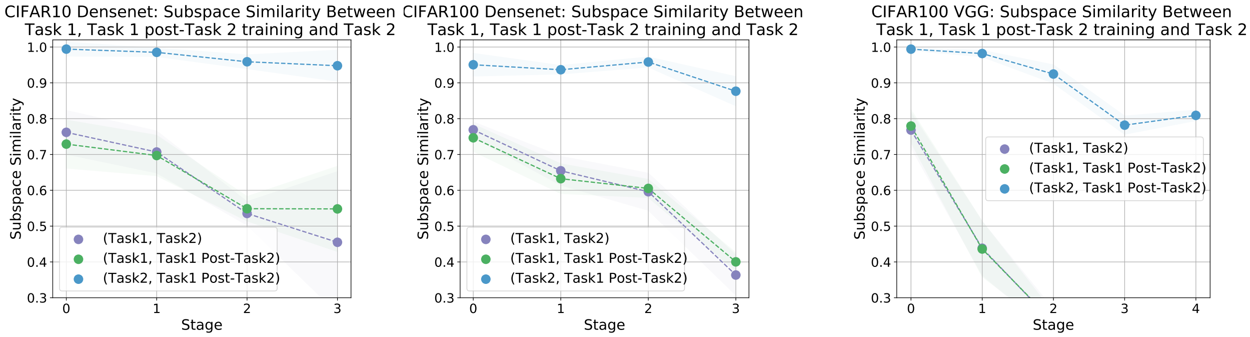


Figure 10: **Subspace similarity analysis shows feature reuse in lower layers and Task 1 subspace erasure in higher layers.**

---

[1] The CIFAR-100 dataset features two vehicle superclasses, denoted *vehicles-1* and *vehicles-2*

A.6 MITIGATION METHODS

A.6.1 ELASTIC WEIGHT CONSOLIDATION

Developed by Kirkpatrick et al. in 2017, elastic weight consolidation (EWC) adds a term to the loss function of the new task to get the regularized loss function $\mathcal{L}(\vec{\theta})$:

$$\mathcal{L}(\vec{\theta}) = \mathcal{L}_B(\vec{\theta}) + \frac{\lambda}{2}\sum_i F_i \cdot \left(\theta_i - \theta^*_{A,i}\right)^2 . \tag{5}$$

Here, $\vec{\theta}$ are the model parameters; $\mathcal{L}_B(\vec{\theta})$ is the unregularized loss function of the new task; λ is a parameter which controls the strength of the regularization; F_i is the diagonal of the Fisher information matrix with respect to the parameters; and $\vec{\theta}^*_A$ are the model parameters after having been trained on task A. We compute the Fisher information via using the squared gradients of the log-likelihood, averaged over a subset of the trainset. We use the empirical Fisher as an approximation.

Following standard practice, in our experiments we estimate the Fisher information using a subset of the full training data; for all of our models on CIFAR-10, 200 samples proved sufficient to be reasonably confident of a converged estimate. We do not include a delay between the start of second-task training and the application of the EWC penalty. Naturally, we do not apply the EWC penalty to the parameters of the head in the multi-head setting.

A.6.2 SYNAPTIC INTELLIGENCE

Developed by Zenke et al. (2017), synaptic intelligence, like EWC, is a structural regularization method which adds an elastic penalty to the second task's loss function:

$$\mathcal{L}(\vec{\theta}) = \mathcal{L}_B(\vec{\theta}) + \frac{\lambda}{2}\sum_i C_i \cdot \left(\theta_i - \theta^*_{A,i}\right)^2 . \tag{6}$$

The difference from EWC is in how the elastic penalties are calculated. In synaptic intelligence, for each parameter, a contribution to the loss is tracked via the product of the gradient of the loss and the stepwise change in the parameter value. The coefficient is then constructed such that changing the parameter by an amount equal to its change during task-1 training will incur a loss penalty equal to the change in loss from the initial value. Our implementation does not differ from the standard implementation.

B SUPPORTING EXPERIMENTS

Here we present experiments omitted from the main text for space, which help flesh out some of the conclusions. This section is roughly organized with layerwise experiments first, followed by semantic experiments.

B.1 LAYER RESET EXPERIMENTS

As described in the main text, we perform an additional experiment to check that deeper representations get corrupted more than shallower representations. This experiment, corresponding to Figure 11, shows that when layers are reset to their pre-forgetting value, performance improves more dramatically when deeper layers are reset first.

B.2 TASK-SPECIFIC STAGES

Our observations that forgetting is driven primarily by the higher layers suggests that in a setting with known task identities (i.e. in the split CIFAR-10 setting described in the main text but not the CIFAR-100 distribution-shift setting), making the deeper stages of the network task-specific would likely mitigate a large portion of forgetting on the old tasks while still allowing for full performance on the new tasks. In particular, since the changes in deeper represntations are much greater than their shallower counterparts, only a few task-specific stages should be necessary to get significant performance increases on the old task. For each of the three network architectures we considered

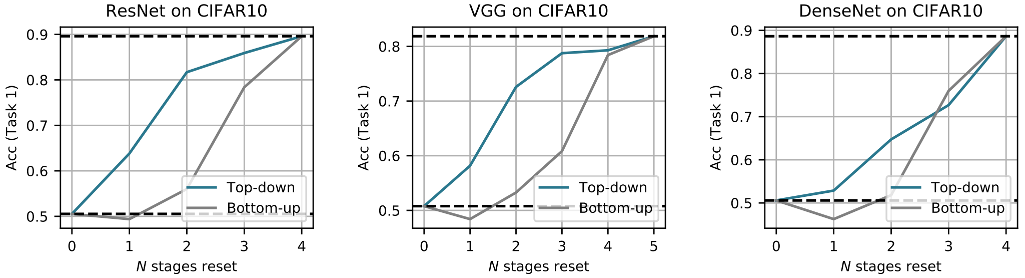


Figure 11: **Layer reset experiments.** After training on Task 2 we reset contiguous blocks of layers to their values before training and record the resulting accuracy on Task 1 (bottom row). We see a significant increase in accuracy when resetting the highest N layers (blue line) compared to resetting the N lowest layers (gray line). Together, these results demonstrate that higher layers are disproportionately responsible for forgetting.

(VGG, ResNet, and DenseNet), we investigated the performance of these task-specific networks on the split CIFAR task. We intend these results to be further evidence of the role of deeper layers in forgetting, not a proposed mitigation scheme.

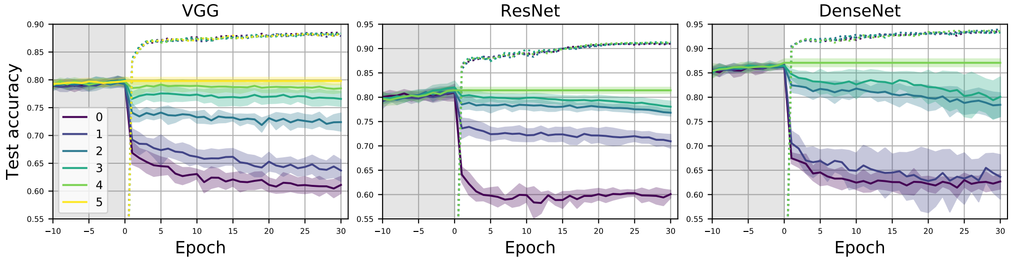


Figure 12: **Forgetting in networks with task-specific stages on CIFAR-10**. Using the same settings as Figure 1 of the main text, we run networks with task-specific layers on split CIFAR-10. We make deeper layers task-specific before shallower layers. Performance increases with number of task-specific layers, denoted by the line color (see legend). In all architectures, two task-specific stages is sufficient to recover significant performance.

Results, shown in figure 12, show that for all architectures, performance dramatically rises with the number of task-specific stages. For all architectures, having the deepest two stages be task-specific recovers a significant fraction of the forgotten task performance.

B.3 LAYER RESET AND RETRAIN EXPERIMENTS

As an additional probe of the degree to which catastrophic forgetting impacts the representations of various network layers, we perform a reset-and-retrain experiment on the split CIFAR-10 task, using the same training setup as in Figure 3 of the main text. The reset-and-retrain experiment is designed to probe the degree to which network representations get corrupted and can no longer be used by a network of the same architecture to productively classify images. To investigate this, after having trained the network on Task 2, we freeze parameters of stages 1 through N to their post-task-2 values, reset the parameters of stages $N + 1$ onwards to their post-task-1 values, and then retrain those parameters on the Task 1 loss function. This is a natural, operational measure of the degree to which layer 1 through N representations have been corrupted.

The results of this experiment, shown in Figure 13, tell a similar story to that of Figure **??** of the main text. In particular, we see even retraining the network with all but the last couple of layers frozen recovers nearly the pre-forgetting performance of the network.

Some of the performances in Figure 13 even outperform the pre-forgetting network performance, suggesting that network representations of intermediate layers may actually be improved by training on Task 2. Further suggestive results are shown in Figure 14, in which we retrain the full network again on Task 1 after the training on Task 2. For ResNet and, to some degree, VGG, training on the second task clearly improves performance on the first.

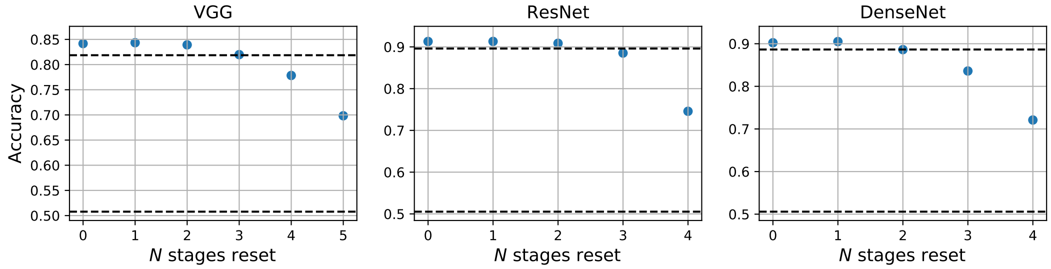


Figure 13: **Layer reset and retrain experiments.** Consistent with the observations in the freeze-training, layer reset, and CKA experiments, resetting and retraining only a few stages from the top is enough to recover the performance before forgetting.

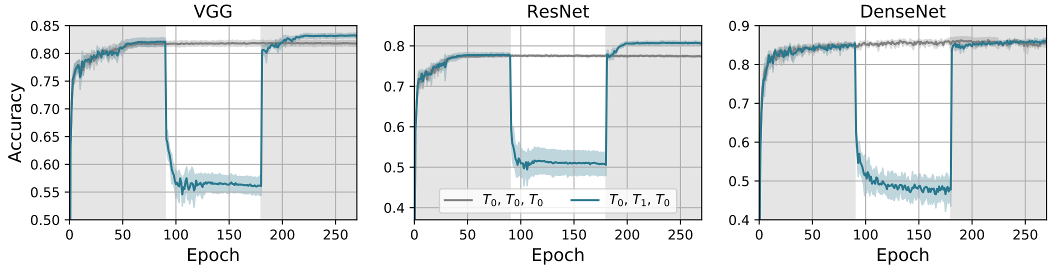


Figure 14: **Training on a different task can improve performance on the original task**. Networks are trained on two five-class splits of CIFAR-10. The gray curve shows the test accuracy on task 0 during training on solely that task for 270 epochs; the teal curve shows the task-0 test accuracy when training on task 0 for 90 epochs, then on task 1 for 90 epochs, and task 0 again for 90 epochs. While we see a sharp performance drop for the duration of training on task 1, the performance is quickly recovered once we begin training on task 0, and in fact surpasses the performance of the model only exposed to task-0 data. This effect is consistent across different splits of the original dataset. Of the architectures we studied, this effect is most pronounced for ResNet, but also exists to some degree for VGG and DenseNet models.

B.4 REPRESENTATION CHANGES IN THE CIFAR-100 DISTRIBUTION SHIFT EXPERIMENTS

Here we show the change in representations due to training on distribution-shifted CIFAR-100 for the VGG (Figure 15) and DenseNet (Figure 16) networks, to accompany the ResNet results in the main text.

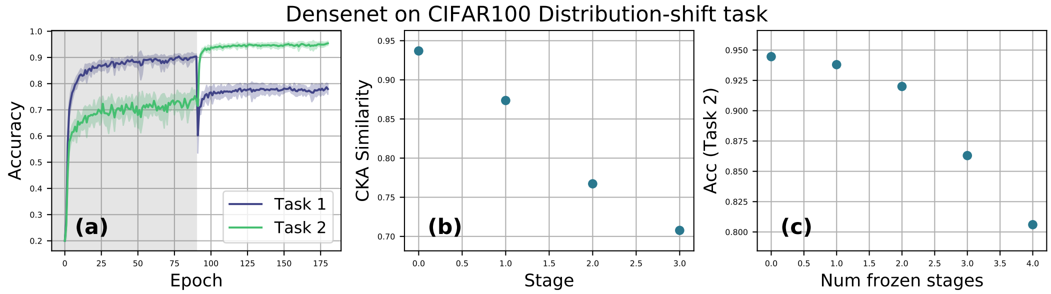


Figure 15: **Change in Densenet representations under CIFAR-100 distribution shift**.

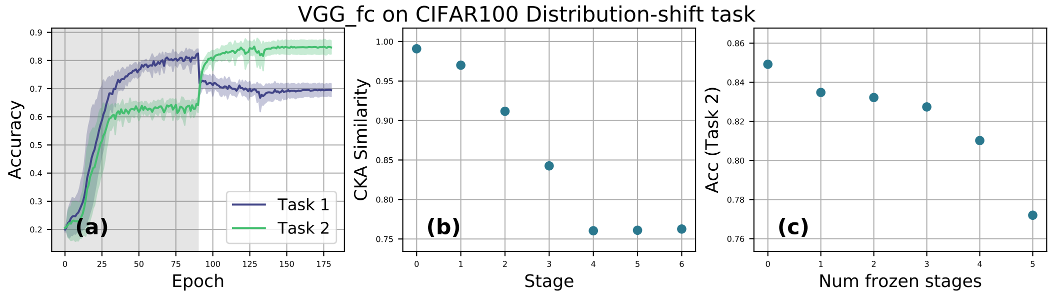


Figure 16: **Change in VGG representations under CIFAR-100 distribution shift**.

B.5 ADDITIONAL EXPERIMENTS FOR SUBSPACE SIMILARITY OF MITIGATION METHODS

We include additional experiments on computing subspace similarities for different mitigation methods across different architectures. The results support the orthogonal storage of Task 1, Task 2 representations by replay, while EWC and SI enforce greater feature reuse in the higher layers.

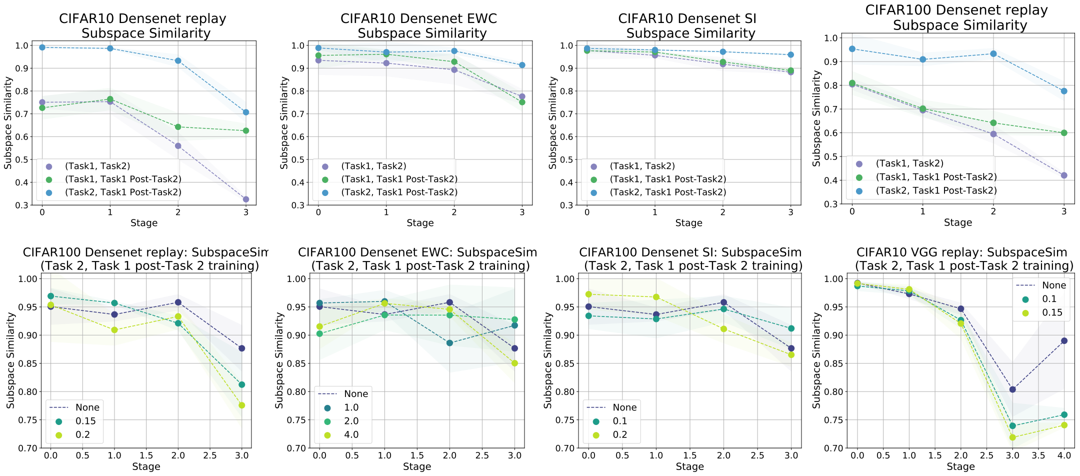


Figure 17: **Subspace similarity analysis reveals differences between different mitigation methods, with replay storing Task 1 and Task 2 representations in orthogonal subspaces, while EWC and SI promote feature reuse in the higher layers.** Additional results, compare to Figure 6 in the main text.

B.6 LINEAR REGRESSION ON FORGOTTEN NETWORK ACTIVATIONS

To further probe the manner in which catastrophic forgetting affects the internal representation of networks, we perform the following experiment (in the split CIFAR-10 setting). We train a (multi-head) model on the first task for some time, resulting in model M_1, and then train on the second task for some time, resulting in model M_2. Due to forgetting, the model M_2 of course performs significantly worse than does M_1 on the first task. But, to probe how much information has been lost internally, we train a linear model to classify between the first-task categories using only the internal activations of M_2 on the first-task images. We measure both the amount of performance which we can recover using this linear regression, and the weight placed by the linear model on various stages of the network.

Model performances are given in Table 1. For all three architectures, the performance of the model before forgetting is naturally the highest, with a performance drop of up to forty percent due to forgetting. Remarkably, however, training a linear model on the internal post-forgetting activations recovers a substantial fraction of the performance loss: for ResNet and VGG, the linear model on post-forgetting activations is only a percent less accurate than the model before forgetting. As a control, we also train a linear model on the activations of networks at initialization (denoted the

	DenseNet	ResNet	VGG
Pre-forgetting accuracy	0.869	0.7714	0.7970
Post-forgetting accuracy	0.4754	0.5924	0.5877
Linear regression on activations	0.802	0.7592	0.7896
Linear regression on random lift	0.447	0.4198	0.543

Table 1: **Performance of linear model trained on internal activations**.

random lift in the table). Because the number of activations is much higher than the number of pixels in a CIFAR-10 image, this control is necessary to ensure that the boost in performance we see is not simply due to performing a high-dimensional lift which renders the data linearly separable. As the table shows, however, this random lift performance is quite poor.

Figure 18 shows the weights placed by the linear model on the various internal activations of the network. We use the activations both before and after forgetting. Before forgetting, a linear model trained on the activations places most importance on the final layers, as expected; however, after forgetting, the model learns to use information stored in earlier-layer representations of the network. The reduced weighting of the final layers is consistent with the experiments we describe in the main text showing that deeper representations suffer the most from forgetting.

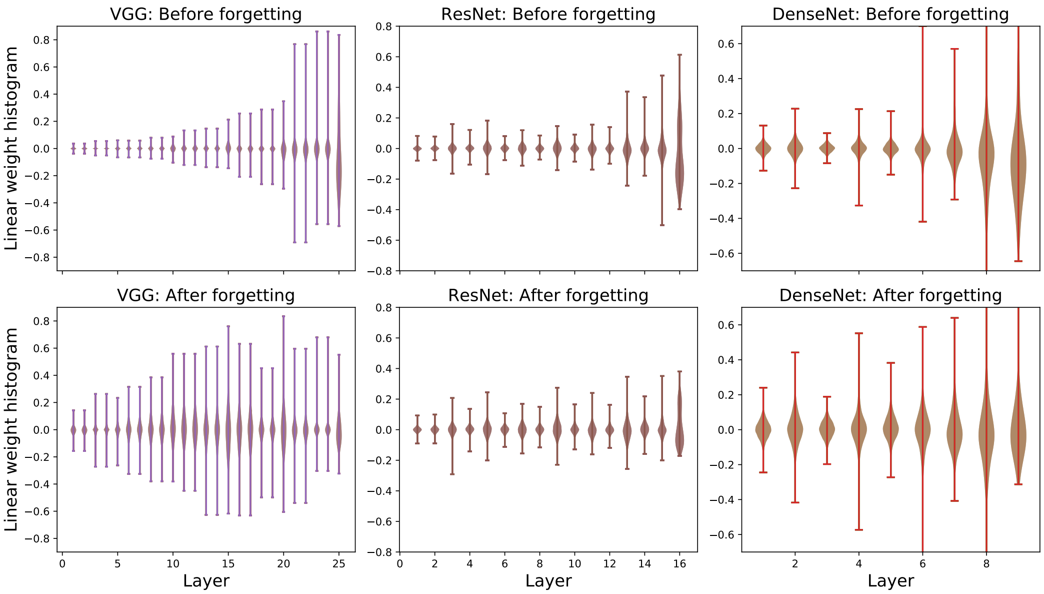


Figure 18: **Weights of linear models trained on internal activations.**

B.7 CKA MEASUREMENTS ON MITIGATION METHODS

This section gives CKA similarity measurements to accompany those in the main text presented in figure 5. The main text figure showed results for ResNet; here we show corresponding results for all architectures on both split CIFAR10 and split CIFAR100 tasks, in figures 19 and 20, repsectively. These plots show that the results discussed in the main text apply broadly across architectures and datasets.

B.8 OTHER CATEGORY IN RESNET AND DENSENET

In Figure 8 of the main text, we showed for the VGG architecture that adding an ‘other’ category (by sampling from all images in the dataset which are not in either of the main tasks) can mitigate forgetting by reducing the similarity between the model’s representations of the task 1 data versus

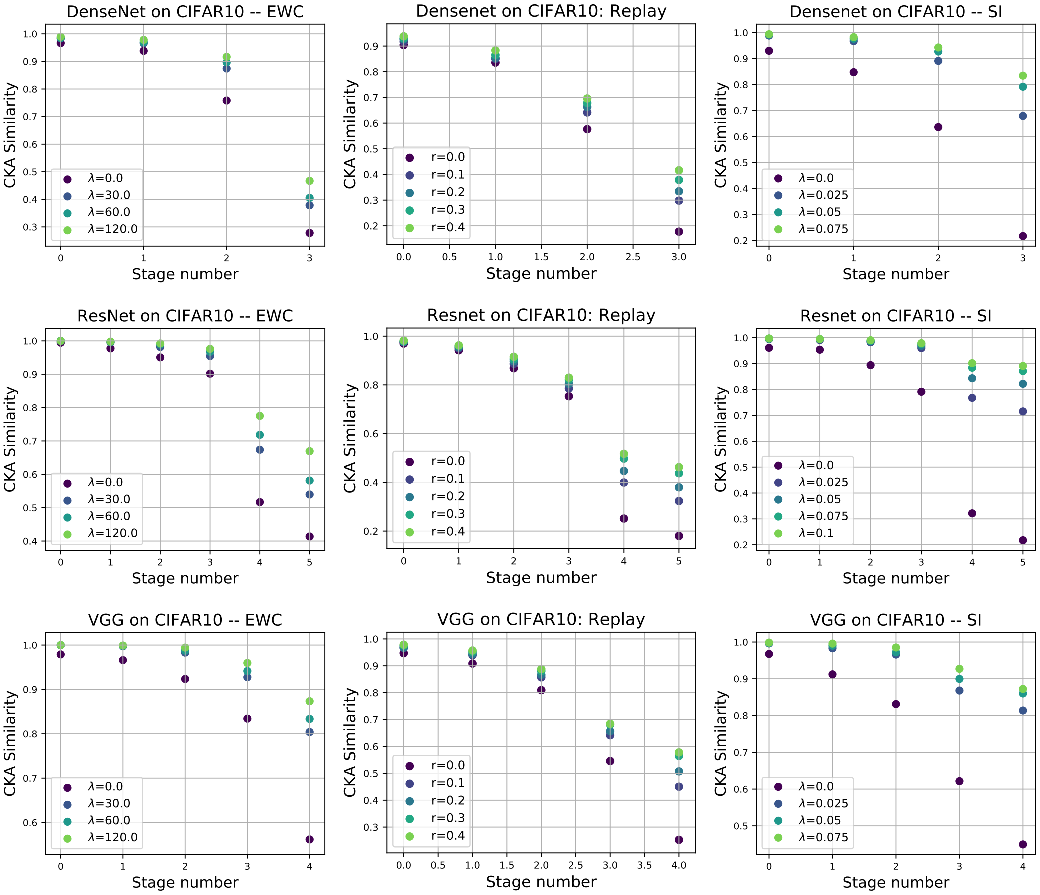


Figure 19: **CKA Analysis of mitigation methods on the split CIFAR10 task**

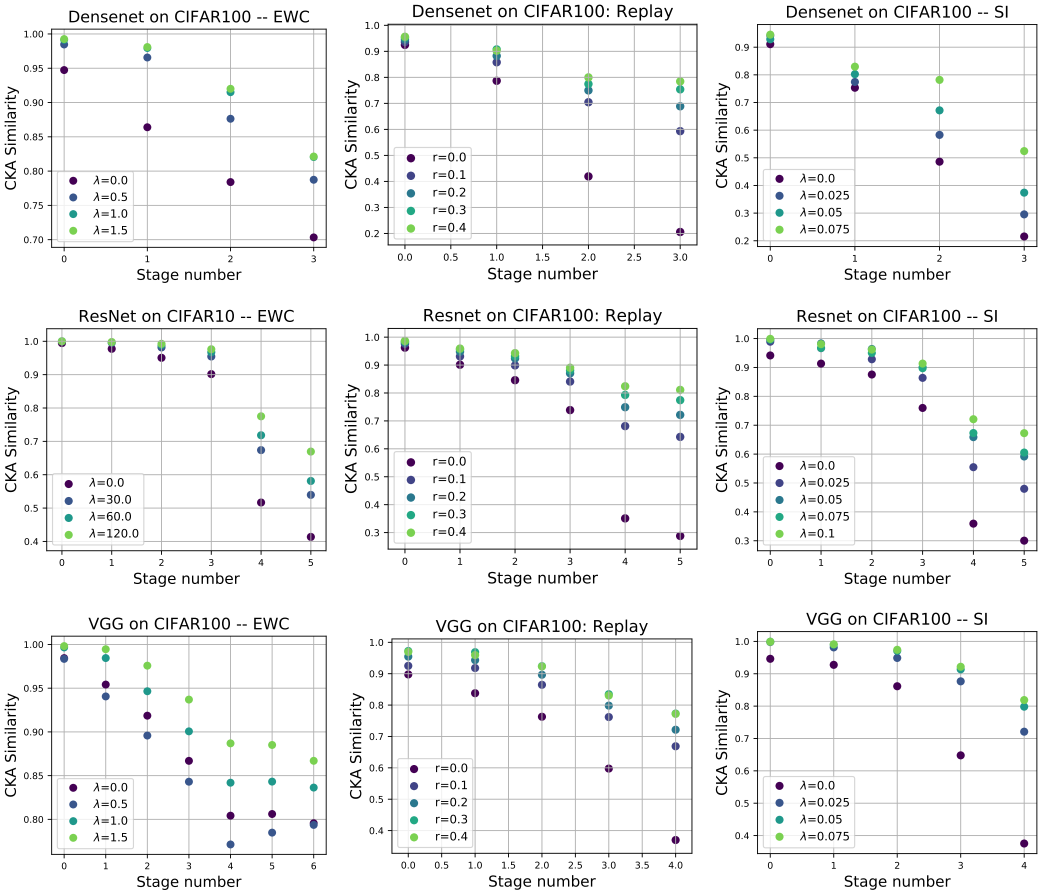


Figure 20: **CKA Analysis of mitigation methods on the split CIFAR100 task**

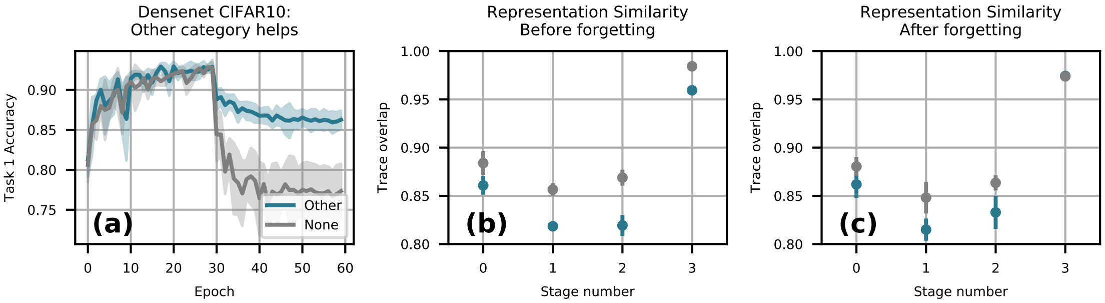


Figure 21: **Adding an ‘other’ category during initial training helps performance through orthogonalization: Densenet**. Model is trained initially on classifying two objects: airplane/automobile, and subsequently on two animals: deer/dog. When we add the ‘other’ category, it comprises a random sampling of images from all other classes. This reduces the similarity between the model’s representations of each tasks’ datapoints, leading to improved continual learning.

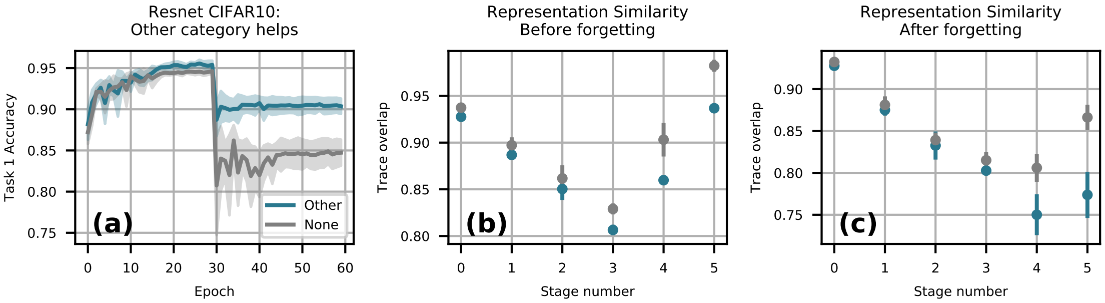


Figure 22: **Adding an ‘other’ category during initial training helps performance through orthogonalization: Resnet**. Model is trained initially on classifying two objects: airplane/automobile, and subsequently on two animals: deer/dog. When we add the ‘other’ category, it comprises a random sampling of images from all other classes. This reduces the similarity between the model’s representations of each tasks’ datapoints, leading to improved continual learning.

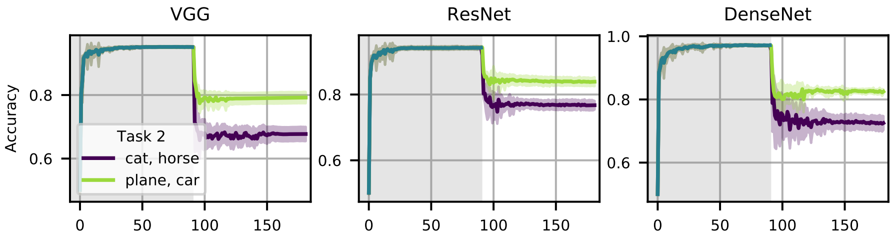
the task 2 data. As our analytic model suggested, increasing this dissimilarity should help forgetting. Figures 21 and 22 show corresponding plots for the Resnet and Densenet architectures.

B.9 ADDITIONAL SEMANTIC EXPERIMENTS

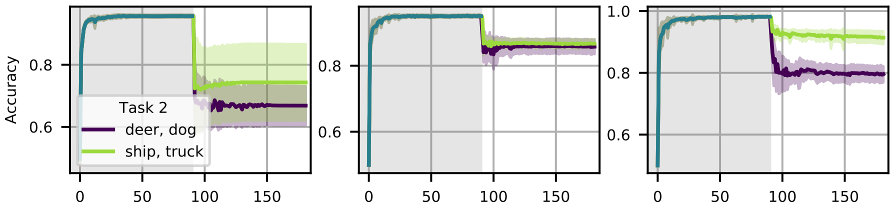
In this section, comprising figures 23, 24, and 25, we present additional realizations of the semantic experiments presented in Section 6, Figure 7. In particular we present results for sequential binary classification tasks, four-class classification followed by two-class classification, and the CIFAR-100 distribution shift task. We find consistent results across different choices of objects and animals and across VGG, ResNet, and DenseNet architectures. In particular, we find that when models are encouraged to distinguish objects and animals, dissimilar categories lead to less forgetting. In contrast when models are trained with only a single semantic category, there is no pressure to distinguish objects from animals and similar categories lead to less forgetting. We observe one exception to the intuitive semantic groupings of categories. For VGG in Setup 2 (Figure 24) the truck category behaves similarly to animal classes in that the performance suffers more when the second task is animal classification then when it is object classification.

C FROZEN-FEATURE MODEL: FURTHER DETAILS

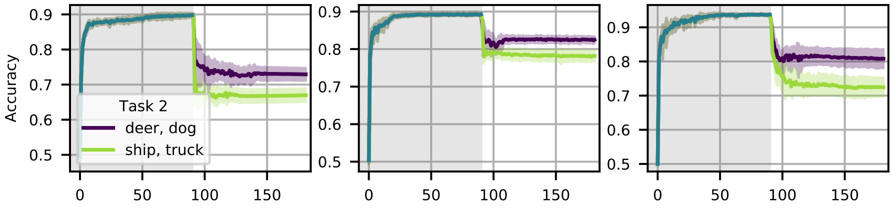
Here we consider extensions and applications of the analytic model introduced in Section 6.2. Both the single and multi-head models relate the change in predictions during training on a second task to



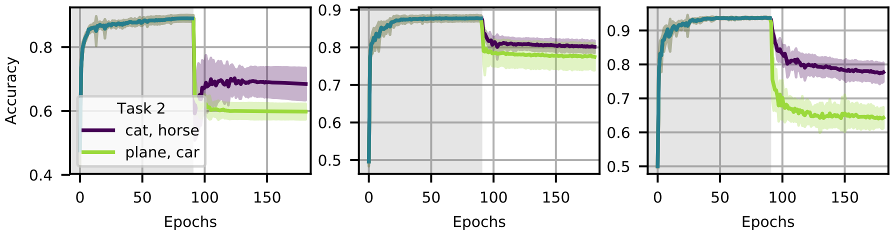
Task 1 - ship, truck, Task 2 - cat, horse or plane, car



Task 1 - plane, car, Task 2 - deer, dog or ship, truck



Task 1 - cat, horse, Task 2 - deer, dog or ship, truck



Task 1 - deer, dog, Task 2 - cat, horse or plane, car

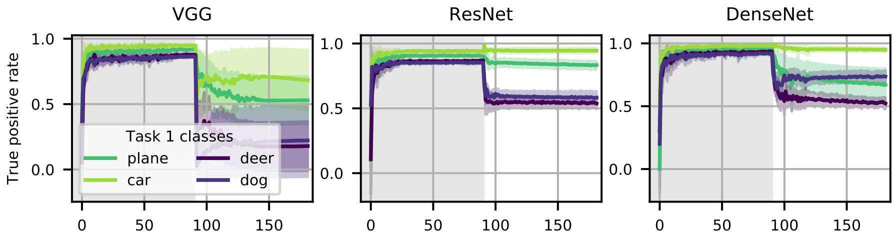
Figure 23: **Sequential binary classification tasks show consistent semantic structure.** We train on sequential binary subsets of CIFAR-10, distinguishing between two animals or two objects. We find that less forgetting occurs when the initial task is more similar to the second task.

the overlap between features, $\Theta(x, x') = \sum_\mu g_\mu(x) g_\mu(x')$, evaluated on the initial and second task data.[2]

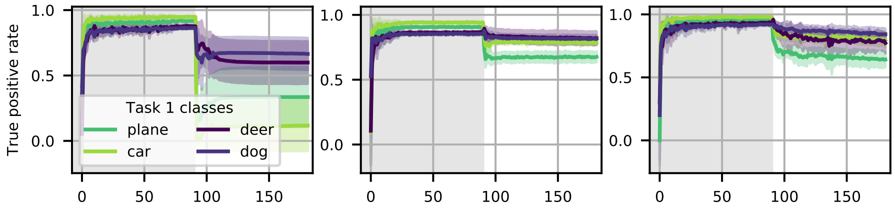
Multi-head model In the main text we focused for simplicity on an analytic model of single head forgetting. Here we construct a solvable model for the multi-head setup. We consider a model

---

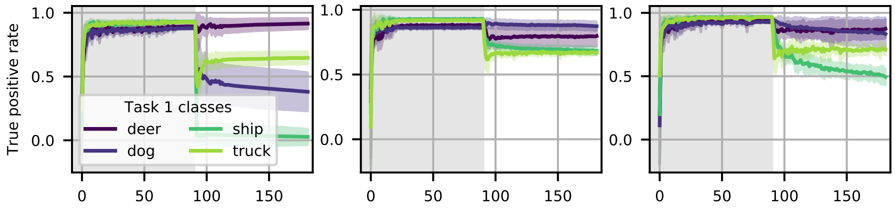
[2] This feature overlap matrix shows up frequently when studying linear models, where it is sometimes known as a reproducing kernel, or wide neural networks, where it is called the neural tangent kernel Jacot et al. (2018) and governs SGD evolution.



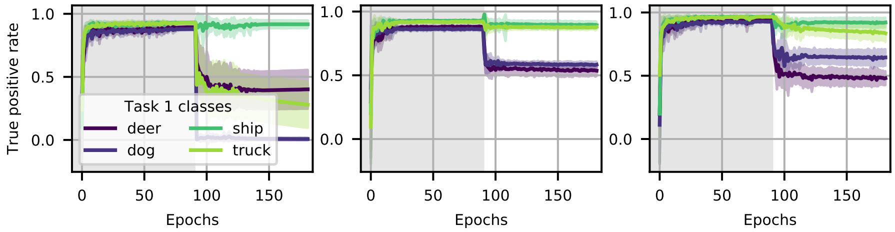
Task 1 - plane, car, deer, dog, Task 2 - cat, horse



Task 1 - plane, car, deer, dog, Task 2 - ship, truck



Task 1 - deer, dog, ship, truck, Task 2 - plane, car



Task 1 - deer, dog, ship, truck, Task 2 - cat, horse

Figure 24: **Sequential four class and binary tasks show decreased forgetting for dissimilar tasks.** We train on sequential subsets of CIFAR-10, initially distinguishing between four classes (two animals and two objects) and then distinguishing between either two animals or two objects. We find that less forgetting occurs for categories that are dissimilar to those used in the second task.

consisting of non-linear features $g(w;x)$, a linear layer with weights, θ, and a read out head $h^{(i)}$.

$$f^{(i)}(x) = \sum_{a,\mu} h_a^{(i)} \theta_{a\mu} g_\mu(w;x)\,. \tag{7}$$

Here μ runs over the feature dimensions, while a runs over the output dimensions of our additional linear layer. For the initial task we use the head $h^{(1)}$, while for the second task, we swap the head and use $h^{(2)}$.

Again we consider a model trained without restrictions on Task 1 using head $h^{(1)}$. For Task 2, we first swap the head and then perform head only training on head $h^{(2)}$. After training the head, we

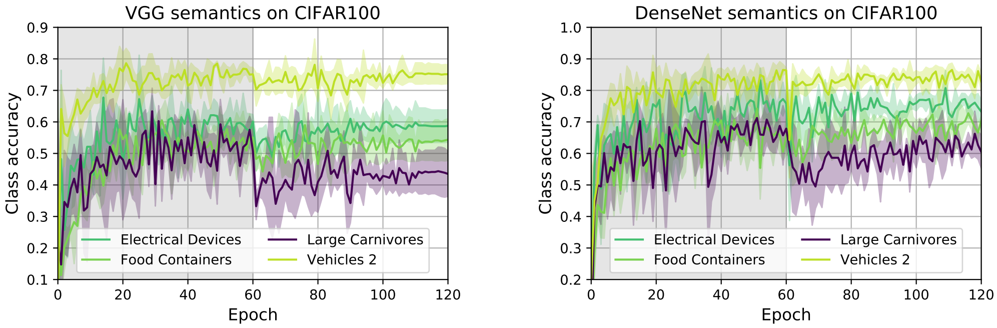


Figure 25: **Semantic structure of forgetting on the CIFAR-100 distribution-shfit task for VGG and DenseNet models**

	VGG	ResNet	DenseNet
No headfirst training	0.147 ± 0.051	0.183 ± 0.001	0.162 ± 0.008
5 epochs headfirst training	0.550 ± 0.029	0.673 ± 0.010	0.700 ± 0.012

Table 2: **Headfirst CKA similarity**. CKA similarity between the readout layer after headfirst training and after final training on the second task.

freeze the head, $h^{(2)}$, and the features $g(w;x) = g(\hat{w};x)$, where $\hat{w}$ is the value of the feature weights after training on the initial task. This model is again inspired by our observation that forgetting is driven by changes in the latter network layers. The multi head model is additionally inspired by our observing that after head first training, the CKA similarity between the second task head before and after second task training is relatively high (see Table 2).

After training the second task head, we continue training the weights θ. The model output evolves as

$$\Delta f_t^{(i)}(x) = -\eta \sum_{x',y' \in \mathcal{D}_{\text{train}}^{(2)}} \Theta(x,x') \frac{\partial L(f^{(j)}(x'),y')}{\partial f} h^{T(j)} h^{(i)} \,. \tag{8}$$

Again we find that if the representation overlap matrix $\Theta(x,x') = \sum_\mu g_\mu(\hat{w};x) g_\mu(\hat{w};x')$ is small between the features evaluated on the initial task and second task data then the predictions do not change significantly. If this overlap is zero, then the predictions are constant. In the multi-head setup we are considering here, we see that the change in predictions is further proportional to the similarity between the model heads, $h^{T(j)} h^{(i)}$.

Finally, we note that we have considered a final linear layer before the readout head for simplicity. One can instead include a ReLU non-linearity without changing the essential point, that the change in model output is governed by the overlap matrix, Θ.

Rotating representations in the analytic model. We can use our analytic model (Section 6.2) to investigate how forgetting depends on representation similarity. We again consider sequential binary tasks (car vs plane) followed by (cat vs horse). We then tune the overlap of our initial task and final task features by explicitly rotating the frozen features for the second task to produce new features $g'(\theta;\hat{w};x)$.

$$g'(\theta;\hat{w};x) = R(\theta) g(\hat{w};x) \,. \tag{9}$$

Here, if we have P features, $R(\theta) \in \mathbb{R}^{P\times P}$ is a one-parameter family of rotation matrices designed such that $R(0) = \mathbf{1}$ and $g^T(\hat{w};x) g'(\pi/2;\hat{w};x')$ for $x \in X_{\text{test}}^{(1)}$ and $x' \in X_{\text{train}}^{(2)}$ is small. Explicitly, we take

$$R(\theta) = V^T \prod_{i=1}^{\lfloor P/2 \rfloor} r_{i,P-i}(\theta) V \,, \tag{10}$$

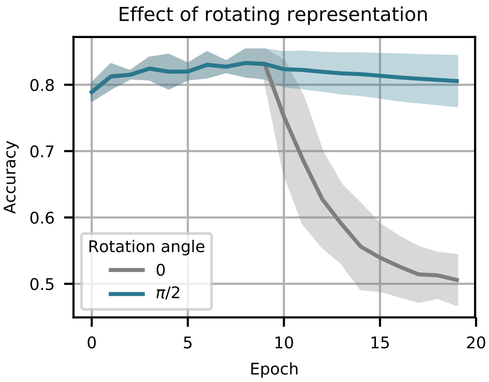


Figure 26: **Rotating representations shows diminished forgetting for dissimilar tasks**. Here we consider the performance of our frozen feature model on sequential binary CIFAR-10 tasks (Task 1: car vs plane, Task 2: cat vs horse). We use a model built from a two hidden-layer fully connected network with ReLU activations. We increase decrease representational similarity by explicitly rotating the second task features and find as predicted that this leads to diminished forgetting.

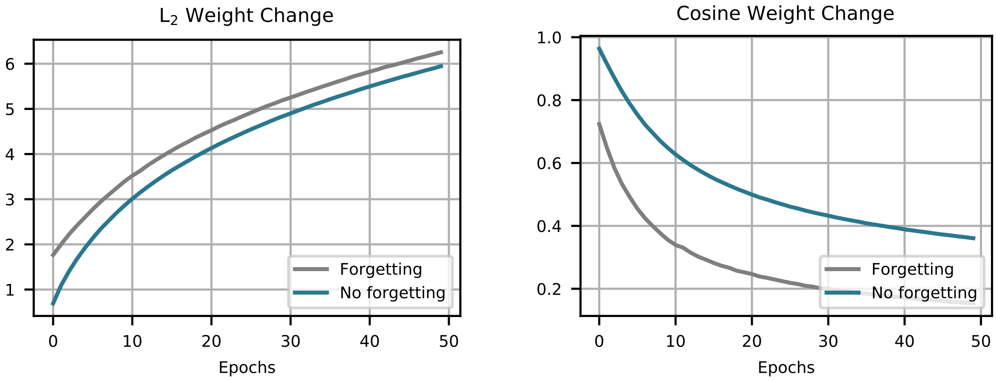


Figure 27: **Stabilizing final layer weights is not necessary to stop forgetting**. Here we look at the change in final layer weights during second task training for a model with severe forgetting (gray) and forgetting mitigated by rotating the second task representations (teal). We see that final layer weights become less and less similar in both cases. The setup is identical to Figure 26

where $r_{ij}(\theta)$ is the two dimensional rotation matrix between axes i and j, and V is the orthogonal matrix appearing in the singular value decomposition of the feature by data matrix on the initial task, $g(X^{(1)}_{\text{test}}) = UDV$. We find that explicitly enforcing task dissimilarity via a large rotation minimizes the effect of forgetting (Figure 26).

Stopping forgetting without stabilizing weights We saw in Section 4 that both EWC and replay buffers stabilize deeper network representations. As mentioned above, this need not be the case a priori. As an illustrative example, we construct a setup where sequential training leads to no forgetting despite a significant change to final layer weights. We consider the single head model of Section 6.2 where the Task 1 and Task 2 representations are completely orthogonal, $\Theta(x, x') = 0$ for $x \in X^{(1)}_{\text{test}}$ and $x' \in X^{(2)}_{\text{train}}$. In this case the Task 1 predictions remain constant throughout all of Task 2 training, despite significant changes to the final layer weights and predictions on the second task. In Figure 27 we present an example of this setup. The representations are again made orthogonal by explicitly rotating the second task features by the rotation matrix, $R(\pi/2)$ defined in Equation equation 10.

D ADDITIONAL EXPERIMENTS

This section presents two additional experiments which are slightly unrelated to the main thrust of the paper. We examine how the degree to which a network forgets is influenced by its width, and also how training the head independently of the rest of the network, in a multi-head setting, can mitigate forgetting.

D.1 FORGETTING VERSUS NETWORK WIDTH

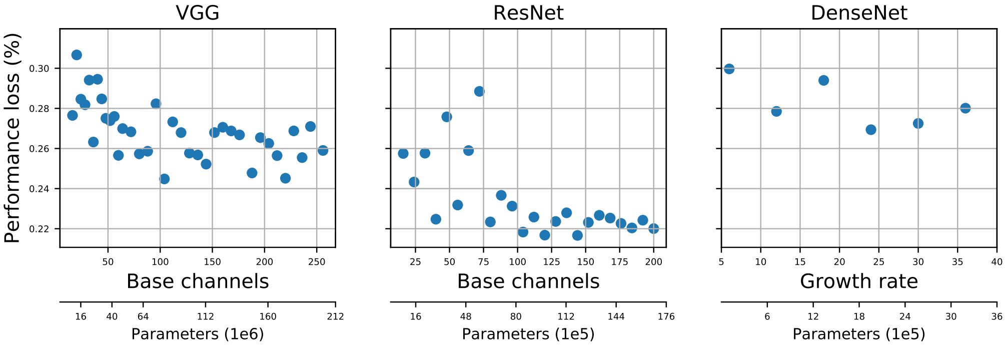


Figure 28: **Forgetting as a function of network width**. (a) VGG, (b) ResNet, and (c) DenseNet networks of varying widths were trained on two subsets of CIFAR-10 classes. We trained each of these (multi-head) networks for 30 epochs on each task with constant η, enough to achieve perfect training accuracy. Further training details can be found in the text. From the plots, which show the percent drop in accuracy (on task 1) due to training on task 2, the effect of width on the performance drop seems to be minimal, even sweeping across roughly a factor of ten in the number of model parameters.

Inspired by recent work of Gilboa & Gur-Ari (2019) showing that wider networks can learn better features than their narrower counterparts, we sought to determine whether wider networks also exhibit less catastrophic forgetting than narrower ones. We tested forgetting versus width for VGG, DenseNet, and ResNet networks on the split CIFAR-10 task in the multi-head setting. For VGG and ResNet, the width was varied by changing the number of channels in the initial convolutional layer and scaling the rest of the network layers concurrently while maintaining the network shape; for DenseNet, the width was varied by by changing the growth rate.

Surprisingly, though we found that the performances of the networks on each task improved as they grew wider, the amount of forgetting in each network, measured by the percent drop in accuracy (on task 1) due to training on task, exhibited a minimal dependence on network width. These results are shown in Figure 28.

D.2 HEADFIRST TRAINING

When training multi-head networks on sequences of tasks, we found that the networks suffered a smaller performance drop on the original task, i.e. forgot less, if *only* the new task head was trained for a few epochs prior to training the full network. In Figure 29, this effect is shown for all three network architectures on the split CIFAR-10 task. Training only the head for up to five epochs seemed to improve the original-task performance without sacrificing performance on the new task. This suggests that coadaptation between the freshly-initialized head and the rest of the model contributes to forgetting when there is no headfirst training.

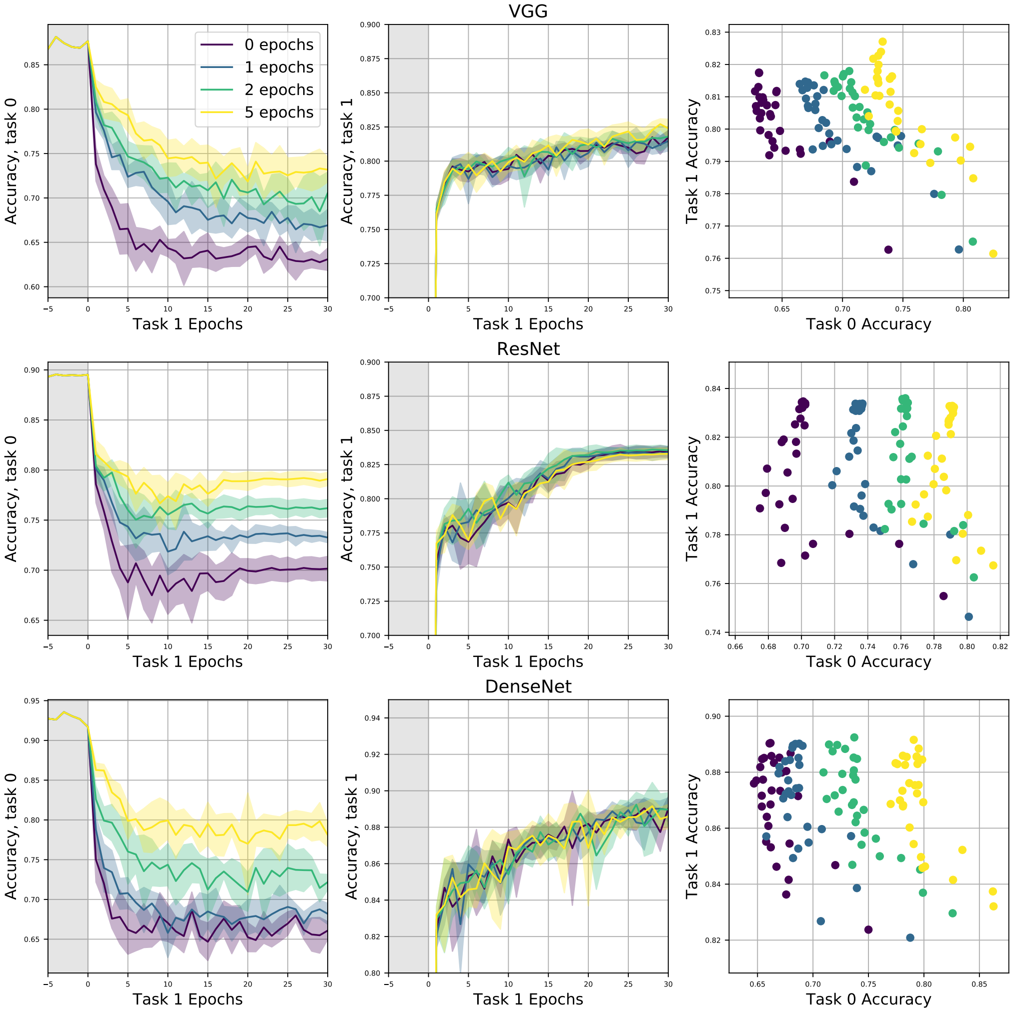


Figure 29: **Effect of training readout layer separately between task switches**. (a) VGG, (b) ResNet, and (c) DenseNet models were trained on two subsets of CIFAR-10 classes: task 0 (*dog*, *frog*, *horse*, *ship*, and *truck*), and task 1 (*airplane*, *automobile*, *bird*, *cat*, *deer*). When switching tasks, we first trained *only* the final classification layer (known as the *readout layer* or *head*) for a specified number of epochs (here 0, 1, 2, and 5) from a random Gaussian initialization. During this period, the rest of the model parameters are held fixed; after the head-only training, the entire model is trained as usual. Across all architectures, training only the head consistently improves the forgetting performance of the model (its accuracy on task 0) while having a negligible impact on the performance on the next task (task 1).