Self-Net: Lifelong Learning via Continual Self-Modeling

Blake Camp*  
Department of Computer Science  
Georgia State University  
Atlanta, GA 30319  
bcamp2@student.gsu.com

Jaya Krishna Mandivarapu*  
Department of Computer Science  
Georgia State University  
Atlanta, GA 30319  
jmandivarapu1@student.gsu.edu

Rolando Estrada  
Department of Computer Science  
Georgia State University  
Atlanta, GA 30319  
restrada1@gsu.edu

Abstract

Continual learning (CL) is one of the most challenging problems in artificial intelligence. While several recent approaches achieve some degree of CL in deep neural networks, they are generally marred by unscalable storage requirements, inefficient training regimes, or model saturation. In this paper, we present a scalable approach to continual learning that offers a practical solution to these problems. Motivated by the biological mechanisms responsible for consolidating knowledge and encoding experiences for long term storage, we present Self-Net, a novel framework which auto-encodes its own networks in a continual fashion. We show that a modified contractive autoencoder can efficiently integrate entire networks into a compact latent space, and we demonstrate that the latent representations can be used to generate high-fidelity recollections of their original counterparts. The result is a single, compact model capable of generating the entire set of task-specific networks, each individually trained on a different task during the lifetime of the system. Our technique outperforms other state-of-the-art approaches on numerous datasets, including continual versions of MNIST, CIFAR10, CIFAR100, and Atari. To the best of our knowledge, we are the first to demonstrate the efficacy of using autoencoders to sequentially encode entire sets of networks in order to facilitate continual learning.

1. Introduction

Continual or lifelong learning (CL) is one of the most challenging problems in machine learning, and it remains a significant hurdle in the quest for artificial general intelligence (AGI) [2, 5]. In this paradigm, a single system must learn to solve new tasks without forgetting previously learned information. Here, a task refers to a desired mapping between inputs and outputs. Different tasks might require different data (e.g., images vs. text) or they might process the same data in different ways (e.g., classifying an object in an image vs. segmenting it). Crucially, in CL there is no point at which a system stops learning; it must always be able to update its representation of its problem domain.

CL is particularly challenging for deep neural networks because they are trained end-to-end. That is, in standard deep learning we tune all of the network’s parameters based on training data, usually via backpropagation[20]. While this paradigm has proven highly successful for individual tasks, it is not suitable for continual learning because we overwrite existing weights (a phenomenon evocatively dubbed catastrophic forgetting [19]). For example, if we first train a network on task A and then on task B, the latter training will modify the original learned weights, thus likely reducing the network’s performance on the first task.

There are several approaches that can achieve some degree of continual learning in deep networks. Existing methods, though, have at least one of three limitations: they either (1) restrict the network’s ability to learn new tasks by penalizing changes to existing weights [7, 27, 22, 14]; (2) expand the model size linearly as the number of tasks grows [21, 10] (or dynamically define task-specific sub-networks [26, 11], which is asymptotically equivalent); or (3) retrain
on old tasks. In the latter, we either store some of the old training data [13, 17, 14], thus increasing storage requirements linearly (and at a faster rate than increasing the network, since data tends to be higher dimensional), or by compressing data [16, 6, 23, 25], which makes training more difficult.

In this paper, we propose a novel approach, Self-Net, that overcomes the aforementioned limitations by decoupling how it learns a new task from how it stores it. Our framework is loosely inspired by the role that the hippocampus is purported to play in memory consolidation [24]. As noted in [15], during learning the brain first forms an initial neural representation in cortical regions; the hippocampus then consolidates this representation into a form that is optimized for storage and retrieval. These complementary biological mechanisms enable continual learning by efficiently consolidating knowledge and compressing prior experiences.

In this spirit, we propose a system that consists of two types of artificial networks: (1) a set of reusable, task-networks (TNs) with potentially different architectures, and (2) a lifelong autoencoder (AE) that efficiently compresses the task-network parameters for all previously learned tasks. Our system self-models its own behavior, which allows it to approximate previously learned parameters instead of having to store them directly, as described in detail in Section 3. In short, when our system needs to learn a new task, it first trains an appropriate task network without any restrictions, akin to standard deep learning. Then, it uses the autoencoder to compress the learned weights into a compact latent vector. It then stores only the latent vector, not the original weights. If the Self-Net needs to solve a previously learned task, it generates an approximation of the original weights—by feeding the corresponding latent vector through the autoencoder—and loads the reconstructed weights onto the appropriate task-network.

Our approach leverages the flexibility of conventional neural networks while avoiding their inability to remember old tasks. The system achieves continual learning because it can learn multiple consecutive tasks in a sequential manner while retaining knowledge gained from previous tasks. The result is a single model that can generate approximations of all previously-learned task-specific networks. In contrast to several competing techniques, we place no restrictions on the task-network as it updates parameters during training. This is permissible and encouraged because the parameters themselves are never explicitly saved, but are rather encoded and approximated by the AE. As a result, it matters not if the original learned parameters for task $t_1$ are overwritten during training of task $t_2$. Further, this allows the framework to leverage the benefits of fine-tuning by initializing the task-network with the weights from a previous, closely related task. Additionally, our approach is not limited to using a single architecture; as we validate in our experiments (Section 4), our AE can simultaneously store architectures of multiple types and sizes which makes training more difficult.

2. Prior work

Several methods have recently emerged for continual learning in deep networks. As noted above, though, existing approaches either (1) restrict new learning, (2) grow the number of parameters linearly, or (3) require old training data. Notable examples of the first type include Elastic Weight Consolidation (EWC) [7], Synaptic Intelligence [27], Variational Continual Learning [14] (which also reuses old data), and Progress & Compress [22]. The last method also uses a task-network and a long-term storage network, but they use EWC to update the weights of the latter, so it has a very similar performance to this method. These approaches use different regularization methods to determine which weights to restrict and by how much. For example, EWC uses the diagonal of the Fisher information matrix between the weights learned for the new task vs. the old tasks. These regularization methods reuse a single network over and over, so they typically use constant storage.

The second category includes Progressive Networks [21], Dynamically Expandable Networks [26], and Context-Dependent Gating [11]. These methods achieve excellent performance, but the number of parameters grows linearly with the number of tasks, which is asymptotically the same as training independent networks. Thus, they cannot scale to large numbers of tasks. Their main advantage is that they facilitate transfer learning, i.e., leveraging previous learning to speed up new learning.

Finally, some methods store a fraction of the old training data and use it to retrain the network on previously learned tasks. Key approaches include Experience Replay [13] iCarl [17], Variational Continual Learning [14], Learning without Forgetting [10]. Unfortunately, this paradigm combines the drawbacks of the previous two. First, most of these methods use a single network, so they cannot continually learn a large number of tasks well. Second, their storage requirements grow linearly in the number of tasks because they have to store old training data. Moreover, data usually takes up orders of magnitude more space than the network itself because a trained network is effectively a compressed representation of the training set [1]. A few methods reduce this storage requirement by storing a compressed representation of the data. Methods of this type include Lifelong Generative Modeling [16], FearNet [6], and Deep Generative Replay [23]. Our proposed approach uses a similar idea but instead stores the networks themselves, rather than the

---

1 As noted in [4], standard EWC stores an $O(n)$ set of Fisher weights for each task, so it actually grows linearly. The modified version proposed in [4] uses constant space, though.
data. Our scheme has two advantages over compressing the data. First, networks are much smaller, so we can encode them more quickly and using less space. Second, by retraining the networks directly, we do not need to retrain on the old tasks.

3. Methodology

Figure 1 provides a high-level overview of our proposed approach. Our Self-Net system has two components: a set of reusable task-networks (TNs) and a lifelong autoencoder (AE). In addition, we store an $O(\log (n))$ latent vector for each task. Each task-network is just a standard neural network. For ease of discussion, we will assume there is a single task-network; the extension to multiple networks is trivial. In either case, it does not matter whether our system has to learn regression, classification, or reinforcement learning tasks (or some combination of the three). The AE, on the other hand, consists of two parts: an encoder that compresses an input vector into a lower-dimensional, latent vector $e$ and a decoder that maps $e$ back to the higher-dimensional space. The goal of the AE is to reconstruct the original input as faithfully as possible, despite this intermediate compression. As detailed below, we use a modified contractive autoencoder [18] due to its ability to quickly incorporate new values into its latent space.

In CL, we must learn $k$ different tasks sequentially. To learn these tasks independently of each other, one would need to train and save $k$ networks, with $O(n)$ parameters each. Thus, this scheme uses $O(kn)$ space in total. In contrast, we propose encoding each of these $k$ networks as an $O(\log (n))$ latent vector using our AE, for an overall storage requirement of only $O(k\log(n))$. Despite this compression, our experiments show that we can reproduce a high-quality approximation of any learned task-network that can still solve the original task. Below, we first describe how to encode a single task-network before discussing how to encode multiple tasks in a continual fashion.

3.1. Single-network encoding

Let $t$ be a task (e.g., recognizing faces) and let $\theta$ be the $O(n)$-dimensional vector of parameters of a network trained to solve $t$. That is, using a task-network with parameters $\theta$, we can achieve a performance $p$ on $t$ (e.g., a classification accuracy of 95%). Now, let $\hat{\theta}$ be the approximate reconstruction of $\theta$ by our autoencoder and let $\hat{p}$ be the performance that we obtain by using these reconstructed weights for task $t$. Our goal is to minimize any performance loss w.r.t. the original weights. If the performance is acceptable, then we can simply store the $O(\log (n))$ latent vector $e$, instead of the $O(n)$ original vector $\theta$.

If we had access to the test data for $t$, we could assess this difference in performance directly and train our autoencoder until we satisfied some acceptable margin $\epsilon$:

$$p - \hat{p} \leq \epsilon.$$  \hspace{1cm} (1)

For example, for a classification task we could stop training our AE if the drop in accuracy is less than 1%. In a continual learning setting, though, we would need to store the old test data for each task, which is infeasible for large numbers of tasks. Instead, we measure the distance between the original and reconstructed weights and stop training when we achieve a suitably close approximation. Empirically, we determined that the cosine similarity

$$\cos(\theta, \hat{\theta}) = \frac{\theta \cdot \hat{\theta}}{\|\theta\| \|\hat{\theta}\|} = \frac{\sum_{i=1}^{n} \theta_i \hat{\theta}_i}{\sqrt{\sum_{i=1}^{n} \theta_i^2} \sqrt{\sum_{i=1}^{n} \hat{\theta}_i^2}},$$ \hspace{1cm} (2)

is an excellent proxy for a network’s performance. Unlike the mean-squared error, this distance metric is scale-invariant, so it is equally suitable regardless of the scale of the original weights. As detailed in Section 4, a cosine similarity of 0.997 or higher yielded near-optimal performance across a wide variety of tasks and network architectures.

3.2. Continual encoding

We will now detail how to use our Self-Net to encode a sequence of trained networks in a continual fashion. As noted above, we first train each task-network using conventional backpropagation. Now, assume that our AE has already learned to encode the first $k$ task-networks. We will now show how to encode the network corresponding to task $t_{k+1}$ into a compressed representation $e_{k+1}$ while still remembering all previously trained networks.

As noted above, we first learn the parameters $\theta_{k+1}$ by training a task-network until we achieve a suitable performance threshold. In our experiments, this step consists of conventional ANN training, i.e., backpropagation w.r.t. to a large training set. We then extract and flatten the learned parameters into an $n$-dimensional vector, where $n$ is the total number of parameters in the network.

Let $E$ be the set of latent vectors for the first $k$ networks. In order to integrate the new network $\theta_{k+1}$ into the latent space, we first recollect all previously trained networks by feeding each $e \in E$ as input to the decoder of the AE. We thus generate a set $R$ of recollections, or approximations, of the original networks (see Figure 1). We then append $\theta_{k+1}$ to $R$ and retrain the AE on all $k+1$ networks until it can reconstruct all of them, i.e., until their respective cosine similarities are all above the predefined threshold. Algorithm 1 summarizes our continual learning strategy.

As we show in our experiments, our retrained AE encodes latent representations of all networks, including the new network $\theta_{k+1}$, that achieve nearly identical performance to the original parameters. Since each $\theta \in R$ is
Figure 1: **Framework overview**: our proposed system has two components: a set of reusable task-specific networks (TN) and a lifelong, AutoEncoder (AE). Given a new task $t_{k+1}$, we first train a TN to learn a set of parameters $\theta_{k+1}$ for this task. We then incorporate the new parameters $\theta_{k+1}$ into our long-term representation by retraining the AE on both its approximations of previously learned networks as well as the new network. When a network is needed (e.g. when a task is revisited), we reconstruct its weights and load them onto the corresponding TN (solid arrow). Even when the latent representation $e_k$ is asymptotically smaller than $\theta_k$, the reconstructed network closely approximates the performance of the original.

**Algorithm 1** Lifelong Learning via Continual Self-Modeling

1. Let $T$ be the set of all Tasks encountered during the lifetime of the system
2. $E = []$
3. initialize AE
4. for idx, curr-task in enumerate($T$) do
5. - Initialize TN with suitable architecture and previously-trained weights for some previous task
6. - Train the TN for curr-task until desired performance
7. $R = []$
8. for encoded-network in $E$ do
9. $r = \text{AE}.\text{Decoder}(\text{encoded-network})$
10. $\text{R.append}(r)$
11. extract and flatten parameters $\theta_{idx}$ from TN
12. $R.\text{append}(\theta_{idx})$
13. cosinesimilarities = []
14. while all cosinesimilarities < .997 do
15. for idx, r ∈ enumerate($R$) do
16. loss = mCAE(r)
17. back-propagate AE w.r.t loss
18. cosinesimilarities[idx] = cos(r, AE(r))
19. $E.\text{append}(\text{AE.\text{Encoder}(R[idx])})$

simply a vector of network parameters, it can easily be loaded back onto a task-network with the correct architecture. This allows us to discard the original networks and store $k$ networks using only $O(k \log (n))$ space. In addition, our framework can efficiently encoding many different types and sizes of networks in a continual fashion. In particular, we can encode a network of arbitrary size $m$ using a constant-size AE (that takes inputs of size $n$) by splitting the input network into $r$ subvectors, such that ($n = m/r^2$).

As we verify in Section 4, we can effectively reconstruct a large network from its subvectors and still achieve a suitable performance threshold.

### 3.3. Modified contractive autoencoder

In our experiments, we train a modified Contractive-AutoEncoder (mCAE) to iteratively approximate the complete set of previously learned task-networks, as well as the new, recently trained network. Our modified Contractive Loss function is identical to that of a standard CAE [18], with one small exception: in addition to the mean squared error and the contractive loss,

$$\|J_{f}(\theta)\|^2_F = \sum_{ij} \left( \frac{\partial h_j(\theta)}{\partial \theta_i} \right)^2$$  \hspace{1cm} (3)
as defined in [18], we also include an additional penalty for deviations in cosine similarity between the input $\theta$ and the reconstructed output $\hat{\theta}$. Thus, our autoencoder’s loss function is given by:

$$mCAE(\theta) = MSE(\theta, \hat{\theta}) + \lambda ||J_f(\theta)||_F^2 + \gamma \cos(\theta, \hat{\theta}).$$

(4)

where $\lambda$ and $\gamma$ are weighing constants. In our experiments, we used a value of 0.0001 for both constants.

Empirically, we found a strong correlation between the degree of cosine similarity of the reconstructed network, relative to its original counterpart, and its performance. Intuitively, this implies that vectors of network parameters which have a cosine similarity approaching 1 will exhibit nearly identical performance on the underlying task. Thus, the cosine similarity can be used as a terminating condition during retraining of the AE. That is, there exists a cosine similarity threshold above which the performance of the reconstructed network can be expected to be sufficiently similar to that of the original. In practice, we found a threshold of .997 to be sufficient.

CAE’s are less sensitive to small variations in the training set due to the regularization term measured by the Frobenius norm $F$ of the Jacobian $J_f(\theta)$ (Equation (3)). Due to this characteristic, and the nature of large sets of network parameters (our inputs), we found CAE’s to be more suitable than other types of Autoencoders, such as variational autoencoders [1], for this problem. In the following section, we offer empirical results which demonstrate the efficacy and flexibility of our approach.

4. Experimental Results

In order to evaluate the continual-learning performance of Self-Net, we carried out a range of experiments on a variety of datasets, in both supervised and reinforcement-learning (RL) settings. We first performed a robustness analysis to establish the degree to which an approximation of a network can deviate from the original and still retain comparable performance on the underlying task (Section 4.1). Then, we evaluated the performance of our approach on the following continual-learning tasks: Permuted MNIST [7], Incremental MNIST, Incremental CIFAR-10 [27], Incremental CIFAR-100 [27], and successive Atari games [13] (we describe each dataset below). As our experiments show, Self-Net can effectively encode each of these different types of networks in sequential fashion, effectively achieving continual learning and outperforming several competing techniques. We also analyzed our system’s performance under three additional scenarios: (1) a very large number of tasks, (2) different sizes of AEs, and (3) multiple task-network architectures.

4.1. Robustness analysis

Our approach relies upon approximations of previously learned networks, and we assume no access to validation data for previously learned tasks during the lifetime of the system. Intuitively, this requires a method for estimating the performance of a reconstructed network which does not rely upon explicit testing on a validation set.

Figure 2 shows the relationship between performance and deviations from the original parameters as measured by cosine similarity, for three datasets. There is a clear correlation between the amount of parameter dissimilarity and the probability of a decrease in performance. That is, given an approximate network, which deviates from the original by some amount, the potential still exists that such a network will retain comparable performance. However, as the degree of deviation increases, the probability that the performance remains high falls steadily. Thus, in order to assume, with reasonable confidence, that the performance of a reconstructed network will be sufficiently high, the AE must minimize the degree of deviation as much as possible.

Empirically, we established a cosine similarity threshold above which the probability of high task-performance stabilizes, as seen in Figure 2. This threshold can be used as a terminating condition during retraining of the AE, and it allows the performance of a reconstructed network to be approximated without access to any validation data. In our experiments, a common threshold yields good performance across a variety of different types and sizes of networks.
4.2. Permuted MNIST

As an initial evaluation of Self-Net’s CL performance, we trained convolutional feed-forward neural networks with 21,840 parameters on successive tasks, each defined by distinct permutations of the MNIST dataset [9], for 10-digit classification. We used networks with 2 convolution layers (kernels of size 5x5, and stride 1x1), 1 hidden layer (320x50), and 1 output layer (50x10). Our CAE had three, fully connected layers with 21,840, 2000, and 20 parameters, resp. Thus, our latent vectors were of size 20. Each task-network was encoded by our lifelong AE in sequential fashion, and the accuracies of all reconstructed networks were examined at the end of each learning stage (i.e., after learning a new task). Figure 3a shows the mean performance after each stage. Our technique almost perfectly matches the performances achieved by independently trained networks, and it dramatically outperforms other state-of-the-art approaches including EWC [7], Online EWC (the correction to EWC proposed in [4]), and Progress & Compress [22]. As a baseline, we also show the results for SGD (no regularization), L2-based regularization in which we compare our new weights to all the previous weights, and Online L2, in which we only measure deviations from the weights learned in the previous iteration. Not only does our technique allow for superior knowledge retention, but it does not inhibit knowledge acquisition necessary for new tasks. The result is minimal degradation in performance as the number of tasks grow.

4.3. Incremental MNIST

We performed a similar continual learning task in which we incrementally learned new digits (Incremental MNIST). Our task-networks and CAE were the same as for Permuted MNIST (except that the outputs of the task-networks were binary, instead of 10 classes). Tasks were specified as binary digit classification, defined by tuples comprised of the positive and negative digit class(es), e.g., (pos={1}, neg={6,7,8,9}, pos={6}, neg={1,2,3,4}, etc.). Here, the training and test sets consisted of approximately 40% positive examples and 60% negative examples. In this domain, too, our technique dramatically outperformed competing approaches, as seen in Figure 3b.

4.4. Incremental CIFAR-10

We then verified that our proposed approach could reconstruct larger, more sophisticated networks. Similar to the Incremental MNIST experiments in section 4.3, we divided the CIFAR-10 dataset [8] into multiple training and test sets and proceeded to train separate task-specific networks, one per class, for binary classification. Here, we used TNs having an architecture which consisted of 2 convolutional layers, followed by 3 fully connected hidden layers, and a final layer having 2 output units. In all, these task networks consisted of more than 60K parameters. Our CAE had three, fully connected layers with 20442, 1000, and 50 parameters, resp. As noted below, we split the 60K

Figure 3: CL performance comparisons with Average test set accuracy on all observed tasks at each stage for (a) Permuted MNIST, (b) Incremental MNIST, and (c) Incremental CIFAR-10.
networks into three subvectors to encode them with our autoencoder. The individual task-networks achieved accuracies ranging from 78% to 84%, and a mean accuracy of approximately 81%. Importantly, we encoded these larger networks using almost the same CAE architecture as the one used in the MNIST experiments. This was achieved by splitting the 60K parameter vectors into three subvectors. As noted in Section 3, by splitting a larger input vector into smaller subvectors, we can encode networks of arbitrary sizes. As seen in Figure 3c, the accuracies of the reconstructed CIFAR networks also nearly matched the performances of their original counterparts, while also outperforming all other techniques.

4.5. Incremental CIFAR-100

We applied the same incremental learning approach for the CIFAR-100 dataset [8]. Due to the limited amount of per-class images in the CIFAR-100 dataset, we split the dataset into 10 distinct batches comprised of 10 classes of images each. This resulted in 10 separate datasets, each designed for 10-class classification tasks. We used the same task-network architecture as that used in our CIFAR-10 experiments, modified slightly to accommodate a 10-class classification objective. The trained networks achieved accuracies ranging from 46% to 59%. We then encoded these networks using the same CAE architecture described in the previous experiments, again accounting for the input size discrepancy by splitting the task-networks into smaller subvectors. As seen in Figure 4, our technique almost perfectly matches the performances achieved by independently trained networks, and it dramatically outperforms other state of the art approaches.

4.6. Incremental Atari

To evaluate the CL performance of Self-Net in the challenging context of reinforcement learning, we used the code available at [3] to implement a modified Async Advantage Actor-Critic (A3C) framework, originally introduced in [12], to attempt to learn successive Atari games while retaining good performance across all games. In contrast to a standard Deep Q-Network, as in [13], A3c simultaneously learns a policy and a value function for estimating expected future rewards. Specifically, the model we used was comprised of 4 convolutional layers (kernals of size 3x3, and strides of size 2x2), a GRU layer (800x256), and two output layers: an Actor (256xNum_Actions), and Critic (256x1), resulting in a complete model with over 800K parameters. Critically, this entire model can be flattened and encoded by the single AE in our Self-Net framework having three, fully connected layers with 76863, 2000, and 200 parameters, resp.

Similar to previous experiments, we trained our system on successive tasks, hereby defined as the following Atari games: Boxing, Star Gunner, Kangaroo, Pong, and Space Invaders. Figure 5 shows the near-perfect retention of performance on each of the 5 games over the lifetime of the system. This is accomplished by training on each game only once, never revisiting the game for training purposes. The dashed, vertical lines demarcate the different stages of continual learning. That is, each stage signifies that a new network has been trained for a new game, over 40M frames. Afterwards, the mean (dashed, horizontal black lines) and standard-deviation (solid, horizontal black lines) of the network’s performance are computed by allowing the it to play the game, unrestricted, for 80 episodes. During each stage, the AE is trained to integrate this new network while retaining all previously-learned networks, as described in the preceding sections. After each stage, the performances of all reconstructed networks were examined by re-playing each game with the appropriate reconstructed network. As Figure 5 shows, the cumulative means and SD’s of the reconstructed networks closely mimic those achieved by their original counterparts.

Significantly, even though the AE is retrained on recollections of networks, it still retains superb performance, even in the challenging domain of Atari and, more broadly, reinforcement learning. To clarify, the network used to play the game of Boxing is recollected and reconstructed a total of 4 times during the course of these experiments. Similarly, the Star Gunner network was recollected 3 times; Kangaroo, 2 times; and Pong, 1 time. That is, during the 5th, or rightmost stage in Figure 5, the newly trained network for Space Invaders is the process of being integrated into the latent space, but all 4 previously-learned networks are being retained using only their recollections, not the originally learned parameters. Figure 5 shows how the AE re-
Figure 5: CL on Five Atari Games with Self-Net. To evaluate the reconstruction score at each stage, we ran the reconstructed networks for 80 full game episodes. The cumulative mean score is nearly identical to the original TN at each and every stage.

contains a large amount of information while integrating a new network into its latent space. The cosine-similarities of the previously-learned networks remain high, even while the AE learns to also reconstruct the new network.

4.7. CL on a large number of tasks

To illustrate the scalability of our approach, we examined the performance of the system on 200 Incremental MNIST tasks. For this experiment, we used the same AE architecture and types of tasks described in Section 4.3. As seen in Figure 6, the mean accuracy of the reconstructed networks remains extremely high (within 1% of the original networks) even after performing continual learning on 200 incremental MNIST tasks. Thus, even for large numbers of tasks, Self-Net can maintain excellent performance.

4.8. Splitting networks and multiple architectures

Splitting larger networks into smaller sub-vectors allows us to use a smaller autoencoder. As an additional analysis, we verified that the smaller AE can be trained in substantially less time than a larger one. Figure 7 shows the respective training rates of an AE with 20,000 input units (blue line)—trained to reconstruct 3 sub-vectors of length 20,000—compared to that of a larger one, with 61,000 input units (yellow line), trained on the original 60K network. Clearly, using more inputs for a smaller autoencoder enables us to more quickly encode larger networks. Finally, we also demonstrated that the same AE can be used to encode trained networks of different sizes and architectures. Figure 7 also shows that the same AE can simultaneously reconstruct 5 MNIST networks and 1 CIFAR network so that all approach their original baseline accuracies.

5. Conclusions and future work

In this paper, we introduced a scalable approach for multi-context continual learning in which we decouple how we learn a set of parameters from how we store them in memory. Our proposed framework makes use of state-of-the-art autoencoders to facilitate lifelong learning via continual self-modeling. Our empirical results confirm that our method can efficiently acquire and retain knowledge in continual fashion. In future work, we aim to further improve the efficiency with which the autoencoder can continually model vast numbers of task-networks. Furthermore, we will explore how to use the latent space to extrapolate to new tasks based on existing learned tasks with little or no training data. We will also try to compress the latent space even further (e.g., using only $\log (k)$ latent vectors for $k$ tasks). Promising approaches include clustering the latent representations into sets of closely related networks and using sparse latent representations. Finally, we will also investigate how to infer the current task automatically, obviating the need for a task label.

References

[1] C. Doersch. Tutorial on Variational Autoencoders. ArXiv e-prints, June 2016.
[2] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211, 2013.
Figure 7: The AE training efficiency is improved when large networks are split into smaller subvectors (upper figure). The lower plot shows that a single AE can encode networks of different architectures and sizes.

[3] S. Greydanus. baby-a3c. https://github.com/greydanus/baby-a3c, 2017. 7

[4] F. Huszár. Note on the quadratic penalties in elastic weight consolidation. Proceedings of the National Academy of Sciences, 115(11):E2496–E2497, 2018. 2, 6

[5] R. Kemker, A. Abitino, M. McClure, and C. Kanan. Measuring catastrophic forgetting in neural networks. CoRR, abs/1708.02072, 2018. 1

[6] R. Kemker and C. Kanan. Fearnet: Brain-inspired model for incremental learning. In International Conference on Learning Representations, 2018. 1, 2

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences, 114(13):3521–3526, 2017. 1, 2, 5, 6

[8] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 6, 7

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, Nov 1998. 6

[10] Z. Li and D. Hoiem. Learning without forgetting. In ECCV, 2016. 1, 2

[11] N. Y. Masse, G. D. Grant, and D. J. Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. CoRR, abs/1802.01569, 2018. 1, 2

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. CoRR, abs/1602.01783, 2016. 7

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. ArXiv e-prints, Dec. 2013. 2, 5, 7

[14] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. In International Conference on Learning Representations, 2018. 1, 2

[15] A. Preston and H. Eichenbaum. Interplay of hippocampus and prefrontal cortex in memory. Current Biology, 23(17):R764 – R773, 2013. 2

[16] J. Ramapuram, M. Gregorova, and A. Kalousis. Lifelong Generative Modeling. ArXiv e-prints, May 2017. 2

[17] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science, 7(2):123–146, 1995. 1

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. Nature, 323:533–536, Oct 1986. 1

[19] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive Neural Networks. ArXiv e-prints, June 2016. 1, 2

[20] J. Schwarz, J. Luketina, W. M. Czamecki, A. Grabska-Barwinska, Y. Whye Teh, R. Pascanu, and R. Hadsell. Progress & Compress: A scalable framework for continual learning. ArXiv e-prints, May 2018. 1, 2, 6

[21] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. CoRR, abs/1705.08690, 2017. 2

[22] T. J. Teyler and P. DiScenna. The hippocampal memory indexing theory. Behavioral Neuroscience, 100(2):147–154, 1986. 2

[23] A. R. Triki, R. Aljundi, M. B. Blaschko, and T. Tuytelaars. Encoder based lifelong learning. CoRR, abs/1704.01920, 2017. 2

[24] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. In International Conference on Learning Representations, 2018. 1, 2

[25] F. Zenke, B. Poole, and S. Ganguli. Improved multitask learning through synaptic intelligence. CoRR, abs/1703.04200, 2017. 1, 2, 5