Toward Training Recurrent Neural Networks for Lifelong Learning

Shagun Sodhani
sshagunsodhani@gmail.com
Sarath Chandar
sarath.chandar@mila.quebec
Mila, University of Montréal, Montreal, Quebec H3T 1J4, Canada

Yoshua Bengio
yoshua.bengio@mila.quebec
Mila, University of Montréal, Montreal, Quebec H3T 1J4, Canada, and CIFAR

Catastrophic forgetting and capacity saturation are the central challenges of any parametric lifelong learning system. In this work, we study these challenges in the context of sequential supervised learning with an emphasis on recurrent neural networks. To evaluate the models in the lifelong learning setting, we propose a curriculum-based, simple, and intuitive benchmark where the models are trained on tasks with increasing levels of difficulty. To measure the impact of catastrophic forgetting, the model is tested on all the previous tasks as it completes any task. As a step toward developing true lifelong learning systems, we unify gradient episodic memory (a catastrophic forgetting alleviation approach) and Net2Net (a capacity expansion approach). Both models are proposed in the context of feedforward networks, and we evaluate the feasibility of using them for recurrent networks. Evaluation on the proposed benchmark shows that the unified model is more suitable than the constituent models for lifelong learning setting.

1 Introduction

Lifelong machine learning considers systems that can learn many tasks (from one or more domains) over a lifetime (Thrun, 1998; Silver, Yang, & Li, 2013). This has several names and manifestations in the literature—incremental learning (Solomonoff, 1989), continual learning (Ring, 1997), explanation-based learning (Thrun, 1996, 2012), never-ending learning (Carlson et al., 2010), and others. The underlying idea motivating these efforts is that: lifelong learning systems would be more effective at learning...
and retaining knowledge across different tasks. By using prior knowledge and exploiting similarity across tasks, they would be able to obtain better priors for the task at hand. Lifelong learning techniques are very important for training intelligent autonomous agents that would need to operate and make decisions over extended periods of time. These characteristics are especially important in industrial setups where deployed machine learning models are being updated frequently with new incoming data whose distribution need not match the data on which the model was originally trained.

The lifelong learning paradigm is not just restricted to the multitask setting with clear task boundaries. In real life, the system has no control over what task it receives at any given time step. In such situations, there is no clear task boundary. Lifelong learning is also relevant when the system is learning just a single task but the data distribution changes over time.

Lifelong learning is an extremely challenging task for machine learning models because of two primary reasons:

1. **Catastrophic forgetting**: As the model is trained on a new task (or a new data distribution), it is likely to forget the knowledge it acquired from the previous tasks (or data distributions). This phenomenon is also known as catastrophic interference (McCloskey & Cohen, 1989).

2. **Capacity saturation**: Any parametric model, however large, can have only a fixed amount of representational capacity to begin with. Given that we want the model to retain knowledge as it progresses through multiple tasks, the model would eventually run out of capacity to store the knowledge acquired in the successive tasks. The only way for it to continue learning while retaining previous knowledge is to increase its capacity on the fly.

Catastrophic forgetting and capacity saturation are related issues. In fact, capacity saturation can lead to catastrophic forgetting. But it is not the only cause for catastrophic forgetting. As the model is trained on one data distribution for a long time, it can forget its “learning” from the previous data distributions regardless of how much effective capacity it has. While an undercapacity model could be more susceptible to catastrophic forgetting, having sufficient capacity (by, say, using very large models) does not protect against catastrophic forgetting (as we demonstrate in section 5). Interestingly, a model that is immune to catastrophic forgetting could be more susceptible to capacity saturation (because it uses more of its capacity to retain the previously acquired knowledge). We demonstrate this effect as well in section 5. It is important to think about catastrophic forgetting and capacity saturation together, as solving just one problem does not take care of the other problem. Further, the role of capacity saturation and capacity expansion in lifelong learning is an underexplored topic.
Motivated by these challenges, we compile a list of desirable properties that a model should fulfill to be deemed suitable for lifelong learning settings:

1. **Knowledge retention**: As the model learns to solve new tasks, it should not forget how to solve the previous ones.
2. **Knowledge transfer**: The model should be able to reuse knowledge acquired during previous tasks to solve current tasks. If the tasks are related, this knowledge transfer would lead to faster learning and better generalization over the lifetime of the model.
3. **Parameter efficiency**: The number of parameters in the model should ideally be bounded, or grow at most sublinearly, as new tasks are added.
4. **Model expansion**: The model should be able to increase its capacity on the fly by “expanding” itself.

The model expansion characteristic comes with additional constraints. In a true lifelong learning setting, the model would experience a continual stream of training data that cannot be stored. Hence, any model would at best have access to only a small sample of the historical data. In such a setting, we cannot rely on past examples to train the expanded model from scratch and a zero-shot knowledge transfer is desired. Considering the parameter efficiency and the model expansion qualities together implies that we would also want the computational and memory costs of the model to increase only sublinearly as the model trains on new tasks.

We propose to unify the gradient episodic memory (GEM) model (Lopez-Paz & Ranzato, 2017) and the Net2Net framework (Chen, Goodfellow, & Shlens, 2015) to develop a model suitable for lifelong learning. The GEM model provides a mechanism to alleviate catastrophic forgetting while allowing for improvement in the previous tasks by beneficial backward transfer of knowledge. Net2Net is a technique for transferring knowledge from a smaller, trained neural network to larger, untrained neural network. We discuss both models in detail in (section 2).

One reason hindering research in lifelong learning is the absence of standardized training and evaluation benchmarks. For instance, the vision community benefited immensely from the availability of the Imagenet data set (Deng et al., 2009), and we believe that availability of a standardized benchmark would help to propel and streamline research in the domain of lifelong learning. Creating a good benchmark set up to study different aspects of lifelong learning is extremely challenging. Lomonaco and Maltoni (2017) proposed a new benchmark for continuous object recognition (CORe50) in the context of computer vision. Lopez-Paz and Ranzato (2017) considered different variants of the MNIST and CIFAR-100 data sets for lifelong supervised learning. These benchmarks help study-specific challenges like catastrophic forgetting by abstracting out the other challenges, but they are
quite far from a real-life setting. Another limitation of the existing benchmarks is that they are largely focused on nonsequential tasks and there has been no such benchmark available for lifelong learning in the context of sequential supervised learning. Sequential supervised learning, like reinforcement learning, is a sequential task and, hence, more challenging than one-step supervised learning tasks. However, unlike reinforcement learning, the setup is still supervised and hence is easier to focus on the challenges in lifelong learning in isolation from the challenges in reinforcement learning.

In this work, we propose a simple and intuitive curriculum-based, benchmark for evaluating lifelong learning models in the context of sequential supervised learning. We consider a single task setting where the model starts with the first data distribution (the simplest data distribution) and subsequently progresses to the more difficult data distributions. We can consider each data distribution as a task by itself. Each task has well-defined criteria of completion, and the model can start training on a task only after learning over all the previous tasks in the curriculum. Each time the model finishes a task, it is evaluated on all the tasks in the curriculum (including the tasks that it has not been trained on so far) so as to compare the performance of the model in terms of both catastrophic forgetting (for the previously seen tasks) and generalization (to unseen tasks).

If the model fails to learn a task (as per predefined criteria of success), we expand the model and let it train on the current task again. The expanded model is again evaluated on all the tasks just like the regular, unexpanded model. Performing this evaluation step enables us to analyze and understand how the model expansion step affects the model’s capabilities in terms of generalization and catastrophic forgetting. We describe the benchmark and the different tasks in detail in section 3.

Our main contributions are as follows:

1. We tackle the two main challenges of lifelong learning by unifying gradient episodic memory (a lifelong learning technique to alleviate catastrophic forgetting) with Net2Net (a capacity expansion technique).
2. We propose a simple benchmark of tasks for training and evaluating models for learning sequential problems in the lifelong learning setting.
3. We show that both GEM and Net2Net, which were originally proposed for feedforward architectures, are indeed useful for recurrent neural networks as well.
4. We evaluate the proposed unified model on the proposed benchmark and show that the unified model is better suited to the lifelong learning setting as compared to the two constituent models.
2 Related Work

We review the prominent works dealing with catastrophic forgetting, capacity saturation, and model expansion because these are the important aspects of lifelong learning.

2.1 Catastrophic Forgetting. Much of the work in the domain of catastrophic forgetting can be broadly classified into two approaches:

1. Model regularization: A common and useful strategy is to freeze parts of the model as it trains on successive tasks. This can be seen as locking in the knowledge about how to solve different tasks in different parts of the model so that training on the subsequent tasks cannot interfere with this knowledge. Sometimes the weights are not completely frozen and are regularized to not change too much as the model trains across different tasks. This approach is adopted by the elastic weight consolidation (EWC) model (Kirkpatrick et al., 2016). As the model trains through the sequence of tasks, the learning is slowed down for weights that are important to the previous tasks. Liu et al. (2018) extended this model by reparameterizing the network to approximately diagonalize the Fisher information matrix of the network parameters. This reparameterization leads to a factorized rotation of the parameter space and makes the diagonal Fisher information matrix assumption (of the EWC model) more applicable. Chaudhry, Dokania, Ajanthan, and Torr (2018) presented RWalk, a generalization of EWC and path integral (Zenke, Poole, & Ganguli, 2017) with a theoretically grounded Kullback-Leibler divergence-based perspective along with several new metrics. One downside of such approaches is the loss in the effective trainable capacity of the model as more and more model parameters are regularized over time. This seems counterintuitive given the desirable properties that we want the lifelong learning systems to have (see section 1).

2. Rehearsing using previous examples: When learning on a given task, the model is also shown examples from previous tasks. This rehearsal setup (Silver & Mercer, 2002) can help in two ways. If the tasks are related, training on multiple tasks helps in transferring knowledge across the tasks, and if the tasks are unrelated, the setup still helps to protect against catastrophic forgetting. Rebuffi, Kolesnikov, Sperl, and Lampert (2017) proposed the iCaRL model, which focuses on the class-incremental learning setting whereas the number of classes (in the classification system) increases; the model is shown examples from the previous tasks. Generally, this strategy requires saving some training examples per task. In practice, the cost of saving some data samples (in terms of memory requirements) is much smaller than the memory requirements of the model. In the rehearsal setup, however, the computational cost of training the model increases with each new task as the model has to rehearse the previous tasks as well.
Mensink, Verbeek, Perronnin, and Csurka (2012) proposed the nearest mean classifier (NCM) model in the context of large-scale, multiclass image classification. The idea is to use distance-based classifiers where a training example is assigned to the class “nearest” to it. The setup allows adding new classes and new training examples to existing classes at a near-zero cost. Thus, the system can be updated on the fly as more training data become available. Further, the model could periodically be trained on the complete data set (collected thus far). Li and Hoiem (2016) proposed the learning without forgetting (LwF) approach in the context of computer vision tasks. The idea is to divide the model into different components. Some of these components are shared between different tasks, and some of the components are task specific. When a new task is introduced, the existing network is used to make predictions for the data corresponding to the new task. These predictions are used as the “ground-truth” labels to compute a regularization loss that ensures that training on the new task does not affect the model’s performance on the previous task. Then a new task-specific component is added to the network and the network is trained to minimize the sum of loss on the current task and the regularization loss. The “addition” of new components per task makes the LwF model parameter inefficient.

Li and Hoiem (2016) proposed using the distillation principle (Hinton, Vinyals, & Dean, 2015) to incrementally train a single network for learning multiple tasks by using data only from the current task. Lee, Kim, Jun, Ha, and Zhang (2017) proposed incremental moment matching (IMM), which incrementally matches the moment of the posterior distribution of the neural network, which is trained on the first and the second tasks, respectively. While this approach seems to give strong results, it is evaluated only on data sets with very few tasks. Serrà, Surís, Miron, and Karatzoglou (2018) proposed using hard attention targets (HAT) to learn pathways in a given base network using the ID of the given task. The pathways are used to obtain the task-specific networks. The limitation of this approach is that it requires knowledge about the current task ID.

The recently proposed GEM approach (Lopez-Paz & Ranzato, 2017) outperforms many of these models while enabling positive transfer on the backward tasks. It uses an episodic memory that stores a subset of the observed examples from each task. When it is training on a given task, an additional constraint is added such that the loss on the data corresponding to the previous tasks does not increase, though it may or may not decrease. One limitation of the model is the need to compute gradients corresponding to the previous task at each learning iteration. Given that GEM needs to store only a few examples per task (in our experiments, we stored just one batch of examples), the storage cost is negligible. Given the strong performance and low memory cost, we use GEM as the first component of our unified model.
2.2 Capacity Saturation and Model Expansion. The problem of capacity saturation and model expansion has been extensively studied from different perspectives. Some work has explored model expansion as a means of transferring knowledge from a small network to a large network to ease the training of deep neural networks (Gutstein, Fuentes, & Freudenthal, 2008; Furlanello, Lipton, Tschannen, Itti, & Anandkumar, 2018). Analogously, the idea of distilling knowledge from a larger network to a smaller network has been explored in Hinton et al. (2015) and Romero et al. (2014). The majority of these approaches focus on training the new network on a single supervised task where the data distribution does not change much and the previous examples can be reused several times. This is not possible in a true online lifelong learning setting where the model experiences a continual stream of training data and has no access to previously seen examples again.

Chen et al. (2015) proposed using function-preserving transformations to expand a small, trained network (referred to as the teacher network) into a large, untrained network (referred to as the student network). Their primary motivation was to accelerate the training of large neural networks by first training small neural networks (which are easier and faster to train) and then transferring their knowledge to larger neural networks. They evaluated the technique in the context of single-task supervised learning and mentioned continual learning as one of the motivations. Given that Net2Net enables the zero-shot transfer of knowledge to the expanded network, we use this idea of function preserving transformations to achieve zero-shot knowledge transfer in the proposed unified model.

Rusu et al. (2016) proposed the idea of progressive networks that explicitly supports the transfer of features across a sequence of tasks. The progressive network starts with a single column or model (neural network) and new columns are added as more tasks are encountered. Each time the network learned a task, the newly added column (corresponding to the task) is “frozen” to ensure that “knowledge” cannot be lost. Each new column uses the layer-wise output from all the previous columns to explicitly enable transfer learning. As a new column is added per task, the number of columns (and hence the number of network parameters) increases linearly with the number of tasks. Further, when a new column is added, only a fraction of the new capacity is actually utilized; thus, each new column is increasingly underutilized. Another limitation is that during training, the model explicitly needs to know when a new task starts so that a new column can be added to the network. Similarly, during inference, the network needs to know the task to which the current data point belongs to so that it knows which column to use. Aljundi, Chakravarty, and Tuytelaars (2016) build on this idea and use a network of experts where each expert model is trained for one task. During inference, a set of gating autoencoders are used to select the expert model to query. This gating mechanism helps to reduce the dependence on knowing the task label for the test data points.
Table 1: Comparison of Different Models in Terms of the Desirable Properties They Fulfill.

<table>
<thead>
<tr>
<th>Property-Model</th>
<th>Knowledge Retention</th>
<th>Knowledge Transfer</th>
<th>Parameter Efficiency</th>
<th>Model Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IMM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>iCaRL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NCM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LwF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GEM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Net2Net</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Progressive Nets</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Network of Experts</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Piggyback</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HAT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Mallya, Davis, and Lazebnik (2018) proposed a piggyback approach to train the model on a base task and then learn different bit masks (for parameters in the base network) for different tasks. One advantage as compared to progressive networks is that only one bit is added per parameter of the base model (as compared to one new parameter per parameter of the base model). The shortcoming of the approach is that knowledge can be transferred only from the base task to the subsequent tasks, and not between different subsequent tasks.

Table 1 compares the different lifelong learning models in terms of the desirable properties they fulfill. The table makes it very easy to determine which combination of models could be feasible. If we choose a parameter-inefficient model, the unified model will be parameter inefficient, which is clearly undesirable. Further, we want at least one of the component models to have an expansion property so that the capacity can be increased on the fly. This narrows the choice of the first model to Net2Net. Since this model lacks both knowledge retention and knowledge transfer, we could pick IMM, GEM, or HAT as the second component. IMM is evaluated for very few tasks, while HAT requires the task IDs to be known beforehand. In contrast, GEM is reported to work well for a large number of tasks (Lopez-Paz & Ranzato, 2017). Given these considerations, we chose GEM as the second component. Now, the unified model has all the four properties.

3 Tasks and Benchmark

In this section, we describe the tasks, training, and evaluation setup that we proposed for benchmarking the lifelong learning models in the
context of sequential supervised learning. In a true lifelong learning setting, the training distribution can change arbitrarily, and no explicit demarcation exists between the data distribution corresponding to the different tasks. This makes it extremely hard to study how model properties like catastrophic forgetting and generalization capability evolve with the training. We sidestep these challenges by using a simple and intuitive curriculum-based, setup where we can have full control over the training data distributions. This setup gives us explicit control over when the model experiences different data distributions and in what order. Specifically, we train the models in the curriculum style setup (Bengio, Louradour, Collobert, & Weston, 2009) where the tasks are ordered by difficulty. We discuss the rationale behind using the curriculum approach in section 3.5. We consider the following three tasks as part of the benchmark.

3.1 Copy Task. The copy task is an algorithmic task introduced in Graves, Wayne, and Danihelka (2014) to test whether the training network can learn to store and recall a long sequence of random vectors. Specifically, the network is presented with a sequence of randomly initialized, seven-bit vectors. Each such vector is followed by an eighth bit, which serves as a delimiter flag. This flag is zero at all time steps except for the end of the sequence. The network is trained to generate the entire sequence except the delimiter flag. The different levels are defined by considering input sequences of different lengths. We start with input sequences of length 5 and increase the sequence length in threes until the maximum sequence length, 62 (20 levels). We can consider arbitrarily large sequences but restrict ourselves to a maximum sequence length of 62 because none of the considered models were able to learn all these sequences. We report the bit-wise accuracy metric.

3.2 Associative Recall Task. The associative recall task is another algorithmic task introduced in Graves et al. (2014). In this task, the network is shown a list of items where each item is a sequence of randomly initialized eight-bit binary vectors, bounded on the left and the right by the delimiter symbols. First, the network is shown a sequence of items, and then it is shown one of the items (from the sequence). The model is required to output the item that appears next from the ingested sequence. We set the length of each item to be 3. The levels are defined in terms of the number of items in the sequence. The first level considers sequences with 5 items, and the number of items is increased in threes per level, going until 20 levels, where there are 62 items per sequence. We report the bit-wise accuracy metric.

3.3 Sequential Stroke MNIST Task. The sequential stroke MNIST (SSMNIST) task was introduced in Gülçehre, Chandar, and Bengio (2017) with an emphasis on testing the long-term dependency modeling
capabilities of the RNNs. In this task, each MNIST digit image \( I \) is represented as a sequence of quadruples \( \{dx_i, dy_i, eos_i, eod_i\}_{i=1}^T \). Here, \( T \) is the number of pen strokes needed to define the digit, \( (dx_i, dy_i) \) denotes the pen offset from the previous to the current stroke (it can be 1, −1, or 0), \( eos_i \) is a binary-valued feature to denote the end of a stroke and \( eod_i \) is another binary-valued feature to denote the end of the digit. The average number of strokes per digit is 40. Given a sequence of pen stroke sequences, the task is to predict the sequence of digits corresponding to each pen stroke sequence in the given order. This is an extremely challenging task as the model is first required to predict the digits based on the pen stroke sequence, count the number of digits, and then generate the digits in the same order as the input after having processed the entire sequence of pen strokes. The levels are defined in terms of the number of digits that make up the sequence. Given that this task is more challenging than the other two tasks, we use a sequence of length 1 (i.e., single-digit sequences) for the first level and increase the sequence length in steps of 1. We again consider 20 levels and report the per digit accuracy as the metric.

3.4 Benchmark. So far, we have considered the setup with three tasks and have defined multiple levels within each task. Alternatively, we could think of each task as a task distribution and each level (within the task) as a task (within a task distribution). From now on, we employ the task-distribution/task notation to keep the discussion consistent with the literature in lifelong learning where multiple tasks are considered. Thus, we have three “task distributions” (copy, associative recall, and SSMNIST) and multiple tasks (in increasing order of difficulty) per “task distribution.” To be closely aligned with the true lifelong learning setup, we train all the models in an online manner where the network sees a stream of training data. Further, none of the examples are seen more than once. A common setup in online learning is to train the model with one example at a time. Instead, we train the model using minibatches of 10 examples at a time to exploit the computational benefits in using minibatches. However, we ensure that every minibatch is generated randomly and that none of the examples are repeated so that a separate validation or test data set is not needed. For each task (within a task distribution), we report the current task accuracy as an indicator of the model’s performance on the current task. If the running average of the current task accuracy, averaged over last \( k \) batches, is greater than or equal to \( c \% \), the model is said to have “learned” the task, and we can start training the model on the next task. If the model fails to learn the current task, we stop the training procedure and report the number of tasks completed. Every model is trained for \( m \) number of minibatches before it is evaluated to check if it has learned the task. Since we consider models with different capacity, some models could learn the task faster, thus experiencing fewer examples. This setup ensures that each model is trained on the same number of examples. This training procedure is repeated for all
the task distributions. $k$, $m$, and $c$ are the parameters of the benchmark and can be set to any reasonable value as long as they are kept constant for all tasks in a given “task distribution.” Specifically, we set $k = 100$, and $m = 10,000$ for all the tasks. $c = 80$ for copy and $c = 75$ for associative recall and SS MNIST.

In the lifelong learning setting, it is very important for the model to retain knowledge from the previous tasks while generalizing to the new ones. Hence, each time the model learns a task, we evaluate it on all the previous tasks (that it has been trained on so far) and report the model’s performance (in terms of accuracy) for each previous task. We also report the average of all these previous task accuracies and denote it as the per-task-previous-accuracy. When the model fails to learn a task and its training is stopped, we report both the individual per-task-previous-accuracy metrics and the average of these metrics, denoted as previous-task-accuracy. While the per-task-previous-accuracy metric can be used as a crude approximation to quantify the effect of catastrophic forgetting, we highlight that the metric, on its own, is an insufficient metric. Consider a model that learns to solve just one task and terminates training after the second task. When evaluated for backward transfer, it would be evaluated only on the first task. Now consider a model that just finished training on the tenth task. When evaluated for backward transfer, it would be evaluated on the first nine tasks. The per-task-previous-accuracy metric favors models that stop training early; hence, the series of per-task-previous-accuracy metrics is a more relevant measure.

Another interesting aspect of lifelong learning is the generalization to unseen tasks. Analogous to the per-task-previous-accuracy and previous-task-accuracy, we consider the per-task-future-accuracy and future-task-accuracy. There are no success criteria associated with this evaluation phase, and the metrics are interpreted as a proxy of the model’s ability to generalize to future tasks. In our benchmark, the tasks are closely related, which makes it reasonable to test generalization to new tasks. Note that the benchmark tasks can have levels beyond 20 as well. We limited our evaluation to 20 levels because none of the models could complete all the levels.

In the context of lifelong learning systems, the model needs to expand its capacity once it has saturated to make sure it can keep learning from the incoming data. We simulate this scenario in our benchmark setting as follows. If the model fails to complete a given task, we use some capacity expansion technique and expand the original model into a larger model. Specifically, since we are considering RNNs, we expand the size of the hidden state matrix. The expanded model is then allowed to train on the current task for 20,000 iterations. From there, the expanded model is evaluated (and trained on subsequent tasks) just like a regular model. If the expanded model fails on any task, the training is terminated. Note that this termination criterion is part of our evaluation protocol. In practice, we can evaluate the model as
many times as we want. In the ablation studies, we consider a case where
the model is expanded twice.

3.5 Rationale for Using Curriculum-Style Setup. For all the three task
distributions, it can be reasonably argued that as the sequence length in-
creases, the tasks become more challenging as the model needs to store and
retrieve a much longer sequence. Hence, for each task distribution, we de-
fine a curriculum of tasks by controlling the length of the input sequences.
We note that our experimental setup is different from the real-life setting in
two ways. First, in real life, we may not know beforehand which data point
belongs to which data (or task) distribution. Second, in real life, we have no
control over the difficulty or complexity of the incoming data points. For the
benchmark, we assume perfect knowledge of which data points belong to
which task and assume full control over the data distribution. This trade-off
has several advantages:

1. Because the tasks are arranged in increasing order of difficulty, it
becomes much easier to quantify the change in the model’s perfor-
mance as the evaluation data distribution becomes different from the
training data distribution.
2. It enables us to extrapolate the capacity of the model with respect to
the unseen tasks. If the model is unable to solve the $n$th task, it is un-
likely to solve any of the subsequent tasks because they are harder
than the current task. Thus, we can use the number of tasks solved
(while keeping other factors like optimizer fixed) as an ordinal indi-
cator of the model’s capacity.
3. As the data distribution becomes harder, the model is forced to use
more and more of its capacity to learn the task.
4. In general, given $n$ tasks, there are $n!$ ways of ordering the task, and
the model should be evaluated on all these combinations because the
order of training tasks could affect the model’s performance. Hav-
ing the notion of the curriculum gives us a natural way to order the
tasks.

To highlight the fact that curriculum-based training is not trivial, we
show the performance of LSTM in the SSMNIST task in Figure 1. We can
see that training on different tasks makes the model highly susceptible to
overfitting to any given task and less likely to generalize across tasks.

Capacity saturation can happen because of two reasons in our proposed
benchmark:

1. The model is operating in a lifelong learning setting, whereas when
the model learns a new task, it also needs to spend some capacity to
retain knowledge about the previous tasks.
2. As the sequence length increases, the new tasks require more capacity to be learned.

Given these factors, it is expected that as the model learns new tasks, its capacity would be strained, thus necessitating solutions that enable the model to increase its capacity on the fly.

4 Model

In this section, we first describe how the rehearsal setup is used in the GEM model and how the function-preserving transformations can be used in the Net2Net model. Next, we describe how we extend the Net2Net model for RNNs. Then we describe how the proposed model leverages both these mechanisms in a unified lifelong learning framework.

4.1 Gradient Episodic Memory. In this section, we provide a brief overview of GEM (Lopez-Paz & Ranzato, 2017) and how it is used for alleviating catastrophic forgetting while ensuring positive transfer on the backward tasks.

The basic idea is to store some input examples corresponding to each task (that the model has been trained on so far) in a memory buffer $B$. In
practice, the buffer would have a fixed size, say, $B_{\text{size}}$. If we know $T$, the number of tasks that the model would encounter, we could reserve $B_{\text{size}}/T$ number of slots for each task. Alternatively, we could start with the first task, using all the slots for storing the examples from the first task. As we progress through tasks, we keep reducing the number of memory slots per task. While selecting the examples to store in the buffer, we save just the last few examples from each task. Specifically, we store only one minibatch of examples (10 examples) per task and find that even these small amounts of data are sufficient.

As the model is training on the $l$th task, care is taken to ensure that the current gradient updates do not increase the loss on the examples already saved in the memory. This is achieved as follows. Given that the model is training on the $l$th task, we first compute the parameter gradient with respect to the data for the current task, which we denote as the current task gradient, $g_l$. Then a parameter gradient is computed corresponding to each of the previous tasks and is denoted as the previous task gradient. If the current gradient $g_l$ increases the loss on any of the previous tasks, it is projected to the closest gradient $\tilde{g}_l$ (where closeness is measured in terms of the $L_2$ norm) such that the condition is not violated. Whether the current task gradient increases the loss on any of the previous tasks can be checked by computing the dot product between the current task gradient and the previous task gradient (corresponding to the given previous task). The projected gradient update $\tilde{g}_l$ can be obtained by solving the following set of equations:

$$\begin{align*}
\text{minimize}_{\tilde{g}_l} & \quad \frac{1}{2} \|g_l - \tilde{g}_l\|^2_2 \\
\text{subject to} & \quad \langle \tilde{g}_l, g_k \rangle \geq 0 \text{ for all } k < l. 
\end{align*}$$

To solve equation 4.1 efficiently, the authors use the primal of a quadratic program (QP) with inequality constraints

$$\begin{align*}
\text{minimize}_z & \quad \frac{1}{2} z^\top C z + p^\top z, \\
\text{subject to} & \quad A z \geq b, 
\end{align*}$$

where $C \in \mathbb{R}^{p \times p}$, $p \in \mathbb{R}^p$, $A \in \mathbb{R}^{(t-1) \times p}$, and $b \in \mathbb{R}^{t-1}$. The dual problem of equation 4.2 is

$$\begin{align*}
\text{minimize}_{u,v} & \quad \frac{1}{2} u^\top Cu - b^\top v \\
\text{subject to} & \quad A^\top v - Cu = p, \\
v & \geq 0.
\end{align*}$$
If \((u^*, v^*)\) is a solution to equation 4.3, then there is a solution \(z^*\) to equation 4.2 satisfying \(Cz^* = Cu^*\) (Dorn, 1960).

The primal GEM QP, equation 4.1, can be rewritten as

\[
\begin{align*}
\text{minimize}_{z} & \quad \frac{1}{2}z^\top z - g^\top z + \frac{1}{2}g^\top g, \\
\text{subject to} & \quad Gz \geq 0,
\end{align*}
\]

where \(G = -(g_1, \ldots, g_{t-1})\), and the constant term \(g^\top g\) is discarded. This new equation is a QP on \(p\) variables (where \(p\) is the number of parameters of the neural network). Since the network could have a lot of parameters, it is not feasible to solve this equation, and the dual of the GEM QP is considered:

\[
\begin{align*}
\text{minimize}_{v} & \quad \frac{1}{2}v^\top GG^\top v + g^\top G^\top v, \\
\text{subject to} & \quad v \geq 0,
\end{align*}
\]

since \(u = G^\top v + g\) and the term \(g^\top g\) is constant. This is a QP on \(t - 1 \ll p\) variables (where \(t\) is the number of observed tasks so far). The solution for the dual problem, equation 4.4, \(v^*\), can be used to recover the projected gradient update as \(\tilde{g} = G^\top v^* + g\). The authors recommend adding a small constant \(\gamma \geq 0\) to \(v^*\), which helps to bias the gradient projection to updates that favored beneficial backward transfer.

We refer to this projection step as computing the GEM gradient and the resulting update as the GEM update. Since the projected gradient is only constrained to not increase the loss on the previous examples, a beneficial backward transfer is possible.

There are several downsides of using the GEM model. First, the projection of the current task gradient regularizes the model, thereby decreasing its effective capacity. This effect can be seen in Figure 2 where for all the three task distributions, the green curve (large LSTM model, which does not use the GEM update) consistently outperforms the the red curve (LSTM model, which uses the GEM update) in terms of both current task accuracy and number of tasks completed. We counter this limitation by using the functional transformations to enable capacity expansion. Another downside is the cost, in terms of both computation and memory, of storing and rehearsing over the previous examples. We found that for all our experiments, storing just 10 examples per task is sufficient to benefit from the GEM model. Hence, the memory footprint of storing the training examples is very small and almost negligible as compared to the memory cost of persisting with different copies of the model. The computational overhead of computing the GEM gradient could be reduced to some extent by controlling the frequency at which the model rehearses using the previous examples and...
future work could look at a more systematic approach to eliminate or re-
duce this computational cost.

4.2 Net2Net. Training a lifelong learning system on a continual stream of data can be seen as training a model with an infinite number of data. As the model experiences more and more data points, the size of its effective training data set increases, and eventually the network would have to expand its capacity to continue training. Net2Net (Chen et al., 2015) proposed a very simple technique, based on function-preserving transformations, to achieve zero-shot knowledge transfer when expanding a small, trained network (referred to as the teacher network) into a large, untrained network (referred to as the student network). Given a teacher network represented by the function $y = f(x, \theta)$ (where $\theta$ refers to the network parameters), a new set of parameters $\phi$ is chosen such that $\forall x. f(x, \phi) = g(x, \theta)$. Chen et al. (2015) considered two variants of this approach: Net2WiderNet, which increases the width of an existing network, and Net2DeeperNet, which increases the depth of the existing network. The main benefit of using function-preserving transformations is that the student network immediately performs as well as the original network without having to go through a period of low performance.

We use the Net2WiderNet for expanding the capacity of the model. The Net2WiderNet formulation is as follows.

Assume that we start with a fully connected network where we want to widen layers $i$ and $i + 1$. The weight matrix associated with layer $i$ is $W^{(i)} \in \mathbb{R}^{m \times n}$, and that associated with layer $i + 1$ is $W^{(i+1)} \in \mathbb{R}^{n \times p}$. Layer $i$ may use any element-wise nonlinearity. When we widen layer $i$, the weight matrix $W^{(i)}$ expands into $U^{(i)}$ to have $q$ output units where $q > n$. Similarly, when we widen layer $i + 1$, the weight matrix $W^{(i+1)}$ expands into $U^{(i+1)}$ to have $q$ input units.

A random mapping function $g : \{1, 2, \ldots, q\} \rightarrow \{1, 2, \ldots, n\}$, is defined as

$$g(j) = \begin{cases} 
  j & j \leq n \\
  \text{random sample from } \{1, 2, \ldots, n\} & j > n.
\end{cases}$$

For expanding $W^{(i)}$, the columns of $U^{(i)}$ are randomly chosen from $W^{(i)}$ using $g$ as shown:

$$U_{k,j}^{(i)} = W_{k,g(j)}^{(i)}.$$

Notice that the first $n$ columns of $W^{(i)}$ are copied directly into $U^{(i)}$. 

Downloaded from http://www.mitpressjournals.org/doi/pdf/10.1162/neco_a_01246 by guest on 21 May 2021
The rows of $U^{(i+1)}$ are randomly chosen from $W^{(i+1)}$ using $g$ as shown:

$$U^{(i+1)}_{j,h} = \frac{1}{|\{x|g(x) = g(j)\}|} W^{(i+1)}_{g(j),h}.$$ 

Similar to the previous case, the first $n$ rows of $W^{(i+1)}$ are copied directly into $U^{(i+1)}$.

The replication factor, given by $\frac{1}{|\{x|g(x) = g(j)\}|}$, is introduced to make sure that the output of the two models is exactly the same. This procedure can be easily extended to multiple layers. Similarly, the procedure can be used for expanding convolutional networks (where layers will have more convolution channels) as convolution is multiplication by a doubly block circulant matrix.

Once the training network has been expanded, the newly created larger network can continue training on the incoming data. In theory, there is no restriction on how many times the Net2Net transformation is applied, though we use the transformation only once for most of our experiments.

While Chen et al. (2015) mention lifelong learning as one of their motivations, they focused only on transfer learning from a smaller network to a larger network for the single-task setup. Second, they considered the Net2Net transformation in the context of feedforward and convolutional models. Our work is the first attempt to use Net2Net-style function transformations for model expansion in the context of lifelong learning or even for sequential models.

### 4.3 Extending Net2Net for RNNs

In this section, we discuss the applicability of the Net2Net formulation for the RNNs in the context of lifelong learning.

The Net2WiderNet transformation makes two recommendations about the training of the student network. The first is that the learning rate for the student network may be reduced by an order of 10. This argument seems useful in the original setup in which Net2Net is proposed: training the student model over the same data on which the teacher model was trained. In the context of lifelong learning, the model does not see the same data again, and the data distribution changes with the task. Hence, the argument about lowering the learning rate does not apply. Our preliminary experiments showed that reducing the learning rate degrades the performance of the model. Hence, we decided not to reduce the learning rate after the expansion.

The second, and more important, recommendation is that a small amount of random noise should be added to the student network to break the symmetry. In our initial experiments, we found that adding noise is a requirement and the model without noise performs extremely poorly. This is in contrast to the feedforward setting, where the model works quite well even without using noise.
In the case of RNNs, when we apply the Net2Wider transformation, the condition number of the hidden-to-hidden matrices increases drastically, and it becomes ill conditioned. Recall that the condition number is defined as the ratio of the largest singular value of the matrix to its smallest singular value. The ideal condition number would be 1 (as is the case of orthogonal matrices), and ill-conditioned networks are harder to train. Without adding noise, the condition number becomes infinity after expansion. This is due to the presence of correlated rows in the matrix. One way to get around this problem is to add a small amount of noise, which helps to precondition the weight matrices and hence reduce their condition number. The issue with adding random noise is that it breaks down the equality condition and hence comes with a trade-off: a higher amount of random noise reduces the condition number more (makes it better conditioned) but pushes the output of the newly instantiated student network away from the predictions of the old teacher network.

To that end, we propose a simple extension to the noise addition procedure that ensures that the output of the student and the teacher networks remains the same while taking care of the preconditioning aspect. Let us say that we had the weight matrix $W_{m \times n}$, which we expanded into $U_{m \times p}$ using the Net2WiderNet transformation (where $p > n$). $U$ would have some columns of $W$ replicated. Let us say that the $i$th column was replicated $j$ times. Then we would generate a noise matrix of small, random values of size $m \times j$. The columns from this noise matrix would be added to columns that were replicated from $i$th column of the input matrix $W$. The noise matrix is generated such that for any row in the matrix, the sum of elements in that row of the noise matrix is 0. It can be shown mathematically that this transformation gives the same output as the case of no noise. We have to employ this procedure to make sure that the random noise we add sums up to 0. Since the given noise is random, it eliminates the correlation between rows and columns of the expanded weight matrix. Since the noise sums up to 0, it does ensure that the output of the student network is the same as that of the teacher network.

How do we generate a matrix of random values where the sum of values along each row is 0? We describe a technique to generate a vector of random numbers such that the values sum to 1, and then we can use the technique multiple times to sample multiple rows to form the matrix. Let us say we want to generate a vector of random values of length $k$ such that the values sum to 1. We first sample $k - 1$ random points in the range $(0, 1)$. Note that all of these $k - 1$ values will be smaller than 1 and larger than 0. We added the numbers 0 and 1 to this sequence and sorted the sequence in the ascending order. This gives us a sorted sequence of $k + 1$ points where each point lies in the range $[0, 1]$ with the first value being 0 and the last value being 1. We take the pairwise difference of values between the adjacent points (second value − first value), (third value − second value) and so on. Summing up this sequence of values would give us (last value − first
value) as all the other terms would cancel out. Since the first value is 0 and the last value is 1, the sum of the sequence of resulting $k$ points is 1. From this sequence of numbers, we can subtract $1/k$, and the resulting sequence would exactly sum to 0. These steps are also described in algorithm 1. Additionally, we scale the noise so that it is in the same range as the magnitude of the weights of the teacher network. Scaling the noise does not change the sum of the noise elements as both the positive and the negative elements get scaled by the same amount and still cancel each other. We use this strategy while using the expansion step.

4.4 Unified Model. We now describe how we combine the catastrophic forgetting solution (GEM) and the capacity expansion solution (functional transformations) to come up with a more suitable model for lifelong learning. Given a task distribution, we randomly initialize a model, reset the episodic memory to be empty, and start training the model on the first task (the simplest task). Once a task is learned, the model starts training on the subsequent, more difficult tasks. When we are training the model on the $l$th task, the episodic memory already has some examples corresponding to the first $l-1$ tasks. The current task gradient is projected with respect to the previous task gradients to ensure that it does not increase the loss associated with any of the examples in the episodic memory. The projected GEM gradient is used to update the weights of the model (GEM update). The model is trained on the current task for a fixed number of iterations. The last $m$ training examples from the current task are stored in episodic memory for use in the subsequent tasks. In general, the $m$ examples can be selected with some more sophisticated strategy though Lopez-Paz and Ranzato (2017) reports, and we validate, that using just the last $m$ samples works well in practice.

---

**Algorithm 1:** Generating a Random-Valued Vector of Lenght $k$ Where the Values Sum to 0.

1: **procedure** GENERATOR($k$)
2: Sample $k-1$ random points in the range $(0, 1)$.
3: Add values 0 and 1 to the sequence of sampled values.
4: Sort the sequence and create a new sequence by subtracting the pairwise values from the sorted sequence.
5: The resulting sequence of $k$ values will sum to 1 (described in the text).
6: From each of the values, subtract $1/k$ to ensure that the resulting sequence of random values sums up to 0.
If the model completes learning the current task (i.e., achieves a threshold amount of accuracy after training), it can start training on the next task. If the model fails to learn the current task and has not been expanded so far, the model is expanded to a larger model and is allowed to train further on the current task. Once the expanded model is trained, it is reevaluated to check if it has learned the task. If it has, the model progresses to the next task; otherwise, the training procedure is terminated. Irrespective of how much the current task accuracy is, the model is evaluated on all the tasks to measure its previous task accuracy and future task accuracy.

4.5 Analysis of the Computational and Memory Cost of the Proposed Model. As noted in section 1, an important desideratum in lifelong learning models is that the computational and the memory costs of the model should ideally grow sublinearly as the model is trained on new tasks. In the context of our proposed model, the computational and memory costs can change in two ways.

First, the Net2Net component expands the model. In this case, the expanded model would take more resources (in terms of both parameters and time) than the earlier model. We note that the expansion step does not happen for every new task and is performed only when the model’s capacity saturates. This is in contrast to approaches like Rusu et al. (2016), where a new copy of the network is added every time a new task is introduced, thus increasing both the parameters and the computing time linearly with the number of tasks. In our case, the frequency of expansion is sublinear in the number of tasks.

Second, the GEM model stores some examples (in a buffer) from the previous tasks and performs gradient computation with respect to those examples, along with the gradient computation for the current examples. As noted in section 4.1, we could keep the buffer size fixed and replace some examples from the previous tasks as new examples are observed, while making sure that all the tasks are represented through examples in the buffer. In practice, we found that storing a few examples per task is sufficient to get benefit from the GEM model (also observed by Lopez-Paz & Ranzato, 2017), making the memory footprint negligible. As noted earlier, future work could look at some systematic ways of selecting the examples from the buffer, thus reducing the computational overhead.

One beneficial side effect of using Net2Net expansion is the zero-shot knowledge transfer that further amortizes the cost of training a newly initialized larger model from either a smaller, pretrained model or from a data set corresponding to the tasks encountered so far.

5 Experiments

5.1 Models. For each task distribution, we consider a standard recurrent (LSTM) model operating in the lifelong learning setting. We consider
the different aspects of training a lifelong learning system and describe how the model variants can account for these aspects. We start with an LSTM model with a hidden state size of 128 and refer to this model as the small-Lstm model. This model has sufficient capacity to learn the first few tasks. We start training this model as described in section 3.4. To avoid catastrophic forgetting, we could also use the GEM update when training the model. The resulting model is referred to as the small-Lstm-Gem model. After learning some tasks, the model would have used up all its capacity (since it is retaining the knowledge of the previous tasks as well). In this case, we could expand the model’s capacity using the Net2Net transformation, and the model with this capability is referred to as the small-Lstm-Gem-Net2Net model. This is the model we propose. Alternatively, we could have started the training with a larger model (large-Lstm model) and could have used the GEM update (large-Lstm-Gem model) to counter forgetting. The strategy of always starting training with a large network would not work in practice because in the lifelong learning setting, we do not know what network would be sufficiently large to learn all the tasks beforehand. If we start with a very large model, we would need a lot more computational resources to train the model and the model would be very prone to overfitting. Our proposed model (small-Lstm-Gem-Net2Net) gets around this problem by increasing the capacity on the fly as and when needed. For the large-Lstm model family, we set the size of the hidden layer to be 256. Our empirical analysis shows that it is possible to expand the models to a size much larger than their current size without interfering with the GEM update.

For the performance on the current task, a large-Lstm model can be treated as the gold standard since this model has the largest capacity among all the models considered. Unlike the models that use the GEM Update, this model does not have to “use” some of its capacity for retaining the knowledge of the previous tasks. For the performance on the previous tasks (catastrophic forgetting), we consider the large-Lstm-Gem model as the gold standard, as this model has the largest capacity among all the models and is specifically designed to counter catastrophic forgetting. While we do not have a gold standard for the Future Task Accuracy, both large-Lstm-Gem and large-Lstm are reasonable models to compare. Overall, we have three different gold standards for three setups (and metrics) and compare our proposed model to these different gold standards (each specialized for a specific use case).

5.2 Hyper Parameters. All the models are implemented using PyTorch 0.4.1 (Paszke et al., 2017). Adam optimizer (Kingma & Ba, 2014) is used with a learning rate of 0.001. We used one-layer LSTM models with hidden dimensions of size 128 and 256. Net2Net is used to expand these models of size 128 to 256. For the GEM model, we keep one minibatch (10 examples) of data per task for obtaining the projected gradients. We follow the guidelines
and hyperparameter configurations as specified in the respective papers for both the GEM and Net2Net models.

**5.3 Results.** Figure 2 shows the trend of the current task accuracy for the different models on the three task distributions. In these plots, a higher curve corresponds to the model that has higher accuracy on the current task and models that learn more tasks are spread out more along the x-axis. We compare the performance of the proposed model small-Lstm-Gem-Net2Net with the gold standard large-Lstm model. We also compare with the large-Lstm-Gem model as both this model and the proposed model are constrained to use some of their capacity on the previous tasks. Hence, it provides a more realistic estimate of the strength of the proposed model. It also allows us to study the effect of the GEM Update on the model’s effective capacity (in terms of the number of tasks cleared). The blue dotted line corresponds to the expansion step when the model is not able to learn the current task and had to expand. This shows that using the capacity expansion technique from Net2Net enables learning on newer tasks. We highlight that before expansion, the proposed model small-Lstm-Gem-Net2Net had a much smaller capacity (128 hidden dimensions) as compared to the other two models, which started with a much larger capacity (256 hidden dimensions). This explains why the larger models have much better performance in the initial stages. After expansion, the proposed model overtakes the GEM-based model in all cases (in terms of the number of tasks solved). We can observe that in all cases, the large-Lstm model outperforms the large-Lstm-Gem model, which suggests that using the Gem Update comes at the cost of reducing the capacity for the current task. Using capacity expansion techniques with GEM enables the model to account for this loss of capacity.

Figure 3 shows the trend of the previous task accuracy for the different models. A higher bar corresponds to better accuracy on the previous tasks (more resilience to catastrophic forgetting). We compare the performance of the proposed model, small-Lstm-Gem-Net2Net, with the gold standard model, large-Lstm-Gem. We also compare with the large-Lstm model to demonstrate that the GEM Update is essential for good performance on the previous tasks. The most important observation is the relative performance of the proposed small-Lstm-Gem-Net2Net model and the large-Lstm-Gem model. The small-Lstm-Gem-Net2Net model started as a smaller model, consistently learned more tasks than the large-Lstm-Gem model, and is still almost as good as the large-Lstm-Gem model in terms of previous task accuracy. This shows that the proposed model is very robust to catastrophic forgetting while being very good at learning the current task. We also observe that for all three task distributions, the models using the GEM Update are more resilient to catastrophic forgetting as compared to the models without the Update.

Figure 4 shows the trend of the future task accuracy for different models. A higher bar corresponds to better accuracy on the future (unseen) tasks.
Figure 2: Current task accuracy for the different models on the three task distributions (copy, associative recall, and SSMNIST respectively). On the x-axis, we plot the index of the task on which the model is training currently, and on the y-axis, we plot the accuracy of the model on that task. Higher curves have higher current task accuracy, and curves extending more have completed more tasks. For all the three task distributions, our proposed small-Lstm-Gem-Net2Net model clears either more levels or the same number of levels as the large-Lstm-Gem model. Before the blue dotted line, the proposed model has much less capacity (hidden size of 128) as compared to the other two models, which have a larger hidden size (256). Hence, the larger models have better accuracy initially. The capacity expansion technique allows our proposed model to clear more tasks than it would have cleared otherwise.
Figure 3: Previous task accuracy for the different models on the three task distributions (copy, associative recall, and SSMNIST, respectively). Different bars represent different models. On the y-axis, we plot the average previous task accuracy (averaged for all the tasks that the model learned). Higher bars have better accuracy on the previously seen tasks and are more robust to catastrophic forgetting. For all three task distributions, the proposed models are very close in performance to the large-Lstm-Gem models and much better than the large-Lstm models.
Figure 4: Future task accuracy for the different models on the three task distributions (copy, associative recall, and SSMNIST, respectively). Different bars represent different models, and on the y-axis, we plot the average future task accuracy (averaged for all the tasks that the model learned). Higher bars have better accuracy on the previously unseen tasks and are more beneficial for achieving knowledge transfer to future tasks. Although the proposed model does not have any component for specifically generalizing to the future tasks, we expect the proposed model to generalize at least as well as the large-Lstm-Gem model and comparable to the large-Lstm model. Interestingly, our model outperforms the large-Lstm model for the copy task and is always better than (or as good as) the large-Lstm-Gem model.
Figure 5: Accuracy of the different models (small-Lstm-Gem-Net2Net, large-Lstm-Gem and large-Lstm respectively) as they are trained and evaluated on different tasks for the copy and the SSMNIST task distributions. On the x-axis, we show the task on which the model is trained, and on the y-axis, we show the accuracy corresponding to the different tasks on which the model is evaluated. We observe that for the large-Lstm model, the high accuracy values are concentrated along the diagonal, which indicates that the model does not perform well on the previous task. In the case of both the small-Lstm-Gem-Net2Net and large-Lstm-Gem models, the high values are in the lower diagonal region, indicating that the two models are quite resilient to catastrophic forgetting.

Since we do not have any gold standard for this setup, we consider both large-Lstm-Gem and large-Lstm models, both reasonable models for a comparison. The general trend is that our proposed model is quite close to the reference models for two out of three tasks. Note that both larger models started training with a much larger capacity and, further, the large-Lstm model is not constrained by the GEM Update and, hence, the maximum amount of effective capacity. This could be one reason why the model can outperform our proposed model for one of the tasks.

In Figure 5, we also consider the heat map plots where we plot the accuracy of different models (for different “task distributions”) as they are trained and evaluated on different tasks. As pointed out in section 3.4, the aggregated metrics (e.g., current task accuracy, previous task accuracy) are not sufficient to compare the performance of different models, and fine-grained analysis is useful for having a holistic view. We observe that for the large-Lstm model, the large values are concentrated along the diagonal,
while for the small-Lstm-Gem-Net2Net and the large-Lstm-Gem models, the high values are concentrated in the lower diagonal region, indicating that the two models are quite resilient to catastrophic forgetting. Note as well that while the large-Lstm-Gem model appears to be more resilient to catastrophic forgetting, the small-Lstm-Gem-Net2Net model consistently clears more tasks. Even though we are evaluating the models for all the tasks in the benchmark, we are restricting the heat map to show only evaluation results for the highest task index that the model could solve. This results in square-shaped heat maps, which are easier to analyze.

It is important to note that we are using a single proposed model (small-Lstm-Gem-Net2Net) and comparing it with gold standard models in three contexts: performance on the current task, performance on the backward tasks, and performance on the future tasks. Our model can provide strong performance on all three tasks by countering catastrophic forgetting and by using capacity expansion.

6 Conclusion

In this work, we study the problem of capacity saturation and catastrophic forgetting in lifelong learning in the context of sequential supervised learning. We propose to unify gradient episodic memory (a catastrophic forgetting alleviation approach) and Net2Net (a capacity expansion approach) to develop a model that is more suitable for lifelong learning. We also propose a curriculum-based evaluation benchmark where the models are trained on a task with increasing levels of difficulty. This enables us to sidestep some of the challenges that arise when studying lifelong learning. We conduct experiments on the proposed benchmark tasks and show that the proposed model is better suited for the lifelong learning setting as compared to the two individual models. As future work, we want to address the computational overhead associated with the GEM Update step.

Appendix

A.1 Results

A.1.1 Current Task Accuracy

Table 2: Current Task Accuracy for the Models for the Copy Task Distribution.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>95.03</td>
<td>96.77</td>
<td>96.77</td>
</tr>
<tr>
<td>2</td>
<td>91.29</td>
<td>98.07</td>
<td>97.85</td>
</tr>
<tr>
<td>3</td>
<td>95.11</td>
<td>98.04</td>
<td>96.283</td>
</tr>
<tr>
<td>4</td>
<td>91.83</td>
<td>97.86</td>
<td>94.308</td>
</tr>
<tr>
<td>5</td>
<td>88.03</td>
<td>96.48</td>
<td>91.446</td>
</tr>
</tbody>
</table>
Table 2: Continued.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>83.58</td>
<td>94.44</td>
<td>88.27</td>
</tr>
<tr>
<td>7</td>
<td>83.107</td>
<td>91.55</td>
<td>84.61</td>
</tr>
<tr>
<td>8</td>
<td>85.67</td>
<td>88.00</td>
<td>81.66</td>
</tr>
<tr>
<td>9</td>
<td>82.82</td>
<td>86.06</td>
<td>79.52</td>
</tr>
<tr>
<td>10</td>
<td>81.25</td>
<td>83.90</td>
<td>77.64</td>
</tr>
<tr>
<td>11</td>
<td>77.12</td>
<td>82.01</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>80.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>78.78</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: The row with the asterisk denotes the task at which the proposed model expanded. Capacity expansion technique allows our proposed model to clear more tasks than it would have cleared otherwise. The proposed small-Lstm-Gem-Net2Net model clears more levels than the large-Lstm-Gem model.

Table 3: Current Task Accuracy for the Different Models for the Associative Recall Task Distribution.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75.99</td>
<td>76.17</td>
<td>76.17</td>
</tr>
<tr>
<td>2</td>
<td>74.57</td>
<td>76.57</td>
<td>75.63</td>
</tr>
<tr>
<td>3</td>
<td>75.6</td>
<td>76.9</td>
<td>75.79</td>
</tr>
<tr>
<td>4</td>
<td>76.1</td>
<td>76.03</td>
<td>75.49</td>
</tr>
<tr>
<td>5</td>
<td>75.29</td>
<td>75.3</td>
<td>74.99</td>
</tr>
<tr>
<td>6</td>
<td>74.56</td>
<td>74.9</td>
<td>74.38</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>75.38</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>75.5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>75.38</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>75.07</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>74.8</td>
<td></td>
</tr>
</tbody>
</table>

Notes: The row with the asterisk denotes the task at which the proposed model expanded. The capacity expansion technique allows our proposed model to clear more tasks than it would have cleared otherwise. The proposed small-Lstm-Gem-Net2Net model clears as many levels as the large-Lstm-Gem model.

Table 4: Current Task Accuracy for the Different Models for the SS-MNIST Task Distribution.

<table>
<thead>
<tr>
<th>Task Id</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89.71</td>
<td>90.59</td>
<td>90.59</td>
</tr>
<tr>
<td>2</td>
<td>77.67</td>
<td>86.71</td>
<td>86.33</td>
</tr>
<tr>
<td>3</td>
<td>73.86</td>
<td>88.08</td>
<td>86.51</td>
</tr>
</tbody>
</table>
Table 4: Continued.

<table>
<thead>
<tr>
<th>Task Id</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>74.437</td>
<td>88.12</td>
<td>84.68</td>
</tr>
<tr>
<td>5</td>
<td>71.14</td>
<td>89.32</td>
<td>79.13</td>
</tr>
<tr>
<td>6</td>
<td>67.84*</td>
<td>90.5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>61.24</td>
<td>90.97</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>90.3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>89.89</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>88.49</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>81.64</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>74.4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>67.8</td>
<td></td>
</tr>
</tbody>
</table>

Notes: The row with the asterisk denotes the task at which the proposed model expanded. Capacity expansion technique allows our proposed model to clear more tasks than it would have cleared otherwise. The proposed small-Lstm-Gem-Net2Net model clears more levels than the large-Lstm-Gem model.

A.1.2 Previous Task Accuracy. Note that the proposed models are very close in performance to the large-Lstm-Gem models and much better than the large-Lstm models.

Table 5: Previous Task Accuracy for the Different Models for the Copy Task Distribution.

<table>
<thead>
<tr>
<th></th>
<th>large-Lstm</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>58.15</td>
<td>82.97</td>
<td>90.76</td>
</tr>
</tbody>
</table>

Table 6: Previous Task Accuracy for the Different models for the Associative Recall Task Distribution.

<table>
<thead>
<tr>
<th></th>
<th>large-Lstm</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>75.2</td>
<td>76.4</td>
<td>76.6</td>
</tr>
</tbody>
</table>

Table 7: Previous Task Accuracy for the Different Models for the SSMNIST Task Distribution.

<table>
<thead>
<tr>
<th></th>
<th>large-Lstm</th>
<th>small-Lstm-Gem-Net2Net</th>
<th>large-Lstm-Gem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>51.31</td>
<td>83.01</td>
<td>82.92</td>
</tr>
</tbody>
</table>
A.1.3 Future Task Accuracy

Table 8: Future Task Accuracy for the Different Models for the Copy Task Distribution.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>large-Lstm</td>
<td>56.55</td>
</tr>
<tr>
<td>small-Lstm-Gem-Net2Net</td>
<td>60.08</td>
</tr>
<tr>
<td>large-Lstm-Gem</td>
<td>56.55</td>
</tr>
</tbody>
</table>

Table 9: Future Task Accuracy for the Different Models for the Associative Recall Task Distribution.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>large-Lstm</td>
<td>74.10</td>
</tr>
<tr>
<td>small-Lstm-Gem-Net2Net</td>
<td>73.22</td>
</tr>
<tr>
<td>large-Lstm-Gem</td>
<td>73.14</td>
</tr>
</tbody>
</table>

Table 10: Future Task Accuracy for the Different Models for the SSMNIST Task Distribution.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>large-Lstm</td>
<td>37.01</td>
</tr>
<tr>
<td>small-Lstm-Gem-Net2Net</td>
<td>26.52</td>
</tr>
<tr>
<td>large-Lstm-Gem</td>
<td>22.58</td>
</tr>
</tbody>
</table>

A.2 Accuracy of Different Models for Different Task Distributions and Tasks. In Figures 6–11, we plot the accuracy of the different models.

Figure 6: small-Lstm-Gem-Net2Net model for the Copy task.
Figure 7: Large-Lstm-Gem model for the Copy task.

Figure 8: Large-Lstm model for the Copy task.
Figure 9: small-Lstm-Gem-Net2Net model for the SSMNIST task.

Figure 10: large-Lstm-Gem model for the SSMNIST task.
as they are trained and evaluated on different tasks for the copy and the SSMNIST task distributions. On the x-axis, we show the task on which the model is trained, and on the y-axis, we show the accuracy corresponding to the different tasks on which the model is evaluated. We observe that for the large-Lstm model, the high accuracy values are concentrated along the diagonal, which indicates that the model does not perform well on the previous task. In the case of both the small-Lstm-Gem-Net2Net and large-Lstm-Gem model, the high values are in the lower diagonal region, indicating that the two models are quite resilient to catastrophic forgetting.

References


Received December 23, 2018; accepted August 14, 2019.