On Efficient Lifelong Learning with A-GEM

Arslan Chaudhry\textsuperscript{1}, Marc Aurelio Ranzato\textsuperscript{2}, Marcus Rohrbach\textsuperscript{2}, Mohamed Elhoseiny\textsuperscript{2}
\textsuperscript{1}University of Oxford, \textsuperscript{2}Facebook AI Research
arslan.chaudhry@eng.ox.ac.uk, \{ranzato, mrf, elhoseiny\}@fb.com

Abstract
In lifelong learning, the learner is presented with a sequence of tasks, incrementally building a data-driven prior which may be leveraged to speed up learning of a new task. In this work, we investigate the efficiency of current lifelong approaches, in terms of sample complexity, computational and memory cost. Towards this end, we first introduce a new and a more realistic evaluation protocol, whereby learners observe each example only once and hyper-parameter selection is done on a small and disjoint set of tasks, which is not used for the actual learning experience and evaluation. Second, we introduce a new metric measuring how quickly a learner acquires a new skill. Third, we propose an improved version of GEM \cite{lee2017overcoming}, dubbed Averaged GEM (A-GEM), which enjoys the same or even better performance as GEM, while being almost as computationally and memory efficient as EWC \cite{riemer2017elapse} and other regularization-based methods. Finally, we show that all algorithms including A-GEM can learn even more quickly if they are provided with task descriptors specifying the classification tasks under consideration. Our experiments on several standard lifelong learning benchmarks demonstrate that A-GEM has the best trade-off between accuracy and efficiency\textsuperscript{1}.

1 Introduction
Arguably, a good set of desiderata for lifelong learning (LLL) are: 1) \textbf{Sample efficiency}: the learner should need only a handful of samples provided one-by-one in a single pass for each task, 2) \textbf{Memory efficiency}: the memory and size of the architecture should not grow with the number of tasks, 3) \textbf{Time efficiency}: the learner should not engage in computations which cannot be easily performed in real time. While existing LLL methods \cite{lee2017overcoming, riemer2017elapse, yoon2017thermo, donahue2017long} perform well in various settings, they fail on at least one of these essential desiderata, as shown in Fig \ref{fig:teaser} and described in Sec. \ref{sec:efficient}.

In this work, we propose an evaluation methodology and an algorithm that better match our desiderata, namely learning efficiently from a stream of tasks. First, we propose a new learning paradigm, whereby the learner performs cross validation on a set of tasks which is disjoint from the set of tasks actually used for evaluation (Sec. \ref{sec:efficient}). In this setting, the learner will have to learn and will be tested on an entirely new sequence of tasks and it will perform just a single pass over this data stream. Second, we build upon GEM \cite{lee2017overcoming}, an algorithm which leverages a small episodic memory to perform well in a single pass setting, and propose a small change to the loss function which not only improves GEM performance, but it also makes GEM orders of magnitude faster at training time; we dub this variant of GEM, A-GEM (Sec. \ref{sec:implement}). Third, we explore the use of compositional task descriptors in order to improve the few-shot learning performance within LLL showing that with this additional information the learner can pick up new skills more quickly (Sec. \ref{sec:compositional}). Fourth, we introduce a new metric to measure the speed of learning, which is useful to quantify the ability of a learning algorithm to learn a new task (Sec. \ref{sec:metric}). And finally, we demonstrate the better trade-off between average accuracy and computational/memory cost of A-GEM on a variety of benchmarks and against several baselines (Sec. \ref{sec:experiments}).

\textsuperscript{1}Under review as a conference paper at ICLR 2019.
2 Learning Protocol

We consider two streams of tasks, described by the following ordered sequences of datasets \( \mathcal{D}^{CV} = \{ \mathcal{D}_1, \ldots, \mathcal{D}_{T^{CV}} \} \) and \( \mathcal{D}^{EV} = \{ \mathcal{D}_{T^{CV}+1}, \ldots, \mathcal{D}_T \} \), where \( \mathcal{D}_k = \{ (x^k_i, y^k_i) \}_{i=1}^{n_k} \) is the dataset of the \( k \)-th task, \( T^{CV} < T \) (in all our experiments \( T^{CV} = 3 \) while \( T = 20 \)), and we assume that all datasets are drawn from the same distribution over tasks. To avoid cluttering the notation, we let the context specify whether \( \mathcal{D}_k \) refers to the training or test set of the \( k \)-th dataset.

\( \mathcal{D}^{CV} \) is the stream of datasets which will be used during cross-validation; \( \mathcal{D}^{CV} \) allows the learner to replay all samples multiple times for the purposes of model hyper-parameter selection. Instead, \( \mathcal{D}^{EV} \) is the actual dataset used for final training and evaluation on the test set; the learner will observe training examples from \( \mathcal{D}^{EV} \) once and only once, and all metrics will be reported on the test sets of \( \mathcal{D}^{EV} \).

Each example in these datasets consists of a triplet defined by an input \( (x^k \in \mathcal{X}) \), task descriptor \( (t^k \in T \), see Sec. 5 for examples) and a target vector \( (y^k \in \mathcal{Y}) \), where \( y^k \) is the set of labels for task \( k \) and \( y^k \subset \mathcal{Y} \). While observing the data, the goal is to learn a predictor \( f_\theta : \mathcal{X} \times T \rightarrow \mathcal{Y} \), parameterized by \( \theta \in \mathbb{R}^F \) (a neural network in our case), that can map a test pair \( (x, t) \) to a target \( y \).

3 Metrics

In addition to Average Accuracy \((A_T)\) and Forgetting Measure \((F_T)\) [2] after seeing \( T \) tasks, we define a new metric, the Learning Curve Area \((\text{LCA} \in [0,1])\), that captures how quickly a model learns.

Let \( a_{k,i,j} \in [0,1] \) be the accuracy evaluated on the test set of task \( j \), after the model has been trained with the \( i \)-th mini-batch of task \( k \). Assuming the first learning task in the continuum is indexed by 1 (it will be \( T^{CV} + 1 \) for \( \mathcal{D}^{EV} \)) and the last one by \( T \) (it will be \( T^{CV} \) for \( \mathcal{D}^{CV} \)), let us first define an average b-shot performance (where \( b \) is the mini-batch number) after the model has been trained for all the \( T \) tasks as: \( Z_b = \frac{1}{T} \sum_{k=1}^{T} a_{k,b,k} \). LCA at \( \beta \) is the area of the convergence curve \( Z_b \) with \( b \in [0, \beta] \):

\[
\text{LCA}_\beta = \frac{1}{\beta+1} \int_0^\beta Z_b db = \frac{1}{\beta+1} \sum_{b=0}^{\beta} Z_b
\]

LCA has an intuitive interpretation. \( \text{LCA}_0 \) is the average 0-shot performance, the same as forward transfer in Lopez-Paz & Ranzato [10]. \( \text{LCA}_\beta \) is the area under the \( Z_b \) curve, which is high if the 0-shot performance is good and if the learner learns quickly. In particular, there could be two models with the same \( A_T \), but very different \( \text{LCA}_\beta \) because one learns much faster than the other while they both eventually obtain the same final accuracy. Since we are interested in models that learn quickly, we will consider small values of \( \beta \).

4 Averaged Gradient Episodic Memory \((\text{A-GEM})\)

In this work we build upon GEM [10], as it has proven to work well in a single pass setting. The inner optimization loop of GEM becomes prohibitive in terms of memory and compute time when the size of the episodic memory \( \mathcal{M} \) and the number of tasks is large. To alleviate this computational burden, we propose a more efficient version of GEM, called Averaged GEM \((\text{A-GEM})\).

Whereas GEM ensures that at every training step the loss of each individual previous tasks, approximated by the samples in episodic memory, does not increase, \( \text{A-GEM} \) tries to ensure that at every training step the average episodic memory loss over the previous tasks does not increase. Formally, while learning task \( t \), the objective of \( \text{A-GEM} \) is:

\[
\text{minimize }_{\theta} \ell(f_\theta, \mathcal{D}_t) \quad \text{s.t. } \ell(f_\theta, \mathcal{M}) \leq \ell(f_{\theta}^{t-1}, \mathcal{M}) \quad \text{where } \mathcal{M} = \bigcup_{k<t} \mathcal{M}_k
\]

The corresponding optimization problem reduces to:

\[
\text{minimize }_{\tilde{g}} \frac{1}{2} ||g - \tilde{g}||_2^2 \quad \text{s.t. } \tilde{g}^T g_{ref} \geq 0
\]

where \( g_{ref} \) is a gradient computed using a batch randomly sampled from the episodic memory, \( (x_{ref}, y_{ref}) \sim \mathcal{M} \), of all the past tasks. In other words, \( \text{A-GEM} \) replaces the \( t-1 \) constraints of GEM with a single constraint, where \( g_{ref} \) is the average of the gradients from the previous tasks computed from a random subset of the episodic memory.
The constrained optimization problem of Eq. \ref{eq:3} can now be solved very quickly; when the gradient $g$ violates the constraint, it is projected via: $\tilde{g} = g - \frac{g^T g_{ref}}{g_{ref}^T g_{ref}} g_{ref}$. This makes A-GEM not only memory efficient, as it does not need to store the parameter gradient vector of each previous task, but also orders of magnitude faster than GEM because 1) it only needs to compute the gradient on an additional minibatch, 2) it does not need to solve any QP but just an inner product, and 3) it will incur in less violations particularly when the number of tasks is large.

5 Joint Embedding Model Using Compositional Task Descriptors

Borrowing ideas from literature in few-shot learning \cite{18b, 16, 3, 14}, we propose the use of a joint embedding model. Let $x^k \in X$ be the input (e.g., an image), $t^k$ be the task descriptor in the form of a matrix of size $C_k \times A$, where $C_k$ is the number of classes in the $k$-th task and $A$ is the total number of attributes for each class in the dataset. The joint embedding model consists of a feature extraction module, $\phi_\theta : x^k \rightarrow \phi_\theta(x^k)$, where $\phi_\theta(x^k) \in \mathbb{R}^D$, and a task embedding module, $\psi_\omega : t^k \rightarrow \psi_\omega(t^k)$, where $\psi_\omega(t^k) \in \mathbb{R}^{C_k \times D}$. During training, the parameters $\theta$ and $\omega$ are learned by minimizing the cross-entropy loss (with the additional constraint of Eq. \ref{eq:3}):

$$
\ell_k(\theta, \omega) = \frac{1}{N} \sum_{i=1}^{N} - \log(p(y^k_i | x^k_i, t^k_i, \theta, \omega)) \quad \text{where} \quad p(c | x^k_i, t^k_i, \theta, \omega) = \frac{\exp\left([\phi_\theta(x^k_i) \psi_\omega(t^k_i)]_c\right)}{\sum_j \exp\left([\phi_\theta(x^k_i) \psi_\omega(t^k_i)]_j\right)}
$$

(4)

where $c = y^k_i$ is the target class of the $i$-th example of task $k$ ($x^k_i$, $t^k_i$, $y^k_i$) and $[a]_i$ denotes the $i$-th element of the vector $a$. Note that the architecture and loss functions are general, and apply not only to A-GEM but also to any other LLL model (e.g., regularization based approaches). See Sec. 6 for the actual choice of parameterization of these functions.

6 Experiments

We consider four dataset streams; Permuted MNIST \cite{5} which is a variant of MNIST \cite{9} where each task has a certain random permutation of the input pixels applied to all the images of that task; Split CIFAR \cite{15} consists of splitting the original CIFAR-100 dataset \cite{6} into 20 disjoint subsets, where each subset is constructed by randomly sampling 5 classes without replacement from a total of 100 classes; Split CUB which is an incremental version of the CUB dataset \cite{13} of 200 bird categories split into 20 disjoint subsets of classes similar to CIFAR; and Split AWA which is an incremental version of the AWA dataset \cite{7} of 50 animal categories, where each task is constructed by sampling 5 classes with replacement from the original of 50 classes. Although a class may appear in different tasks in Split AWA, the training data of each class is split into disjoint sets, so that no training sample is seen more than once. On Permuted MNIST and Split CIFAR we provide integer task descriptors. On Split CUB and Split AWA, we assemble together the attributes of the classes belonging to the current task to form a task descriptor.

In terms of architectures, we use a fully-connected network with two hidden layers of 256 ReLU units each for Permuted MNIST, a reduced ResNet18 for Split CIFAR \cite{10}, and a standard ResNet18 \cite{4} for Split CUB and Split AWA. For a given dataset stream, all models use the same architecture and all models are optimized via stochastic gradient descent with mini-batch size equal to 10. We refer to the joint-embedding model version of these models by appending the suffix ‘-JE’ to the method name.

6.1 Results

On Split CIFAR, GEM and A-GEM have the average accuracy of 61.2 and 62.9, and forgetting of 0.06 and 0.07. This shows that A-GEM and GEM perform comparably in terms of average accuracy and forgetting, but A-GEM has much lower time (about 100 times faster) and memory cost (about 10 times lower). We obtained similar findings in Permuted MNIST.

Fig. \ref{fig:results} show the overall results on Split CUB and Split AWA. First, we notice that A-GEM achieves the best average accuracy with both the standard and the joint embedding model. Second, A-GEM has the lowest forgetting among methods that use a fixed capacity architecture. Third, we found that although PROG-NN \cite{11} has no forgetting by construction, it has the worst memory cost by the end of training – as its number of parameters grows super-linearly with the number of tasks. On these two datasets, PROG-NN runs out of memory (OoM). Fourth, EWC performs only slightly better than VAN on this single pass LLL setting. Next, all methods perform similarly in terms of LCA, with A-GEM slightly better than all the other approaches. Finally, the use of task descriptors improves
average accuracy across the board with A-GEM a bit better than all the other methods we tried. All joint-embedding models using task descriptors have better LCA performance.

Fig. 2 shows the effectiveness of task descriptors in the few-shot regime. Fig. 3 shows the 0-shot performance of LLL methods over time, showing a clear advantage of using compositional task descriptors and A-GEM. Overall, A-GEM offers the best trade-off between average accuracy performance and efficiency in terms of sample, memory and computational cost.

7 Conclusion

We studied the problem of Lifelong Learning when the learner can only do a single pass over the input data stream. Our experiments show that our approach, A-GEM, has the best trade-off between average accuracy and computational/memory cost. Compared to the original GEM algorithm, A-GEM is about 100 times faster and has 10 times less memory requirements; compared to regularization based approaches, it achieves significantly higher average accuracy at a small increase of compute/memory requirements. By using compositional task descriptors all methods can improve their few-shot performance, with A-GEM often being the best.
References


