Online Learned Continual Compression with Adaptive Quantization Modules

Lucas Caccia$^{1,2,3}$ Eugene Belilovsky$^{4,2}$ Massimo Caccia$^{4,2,5}$ Joelle Pineau$^{1,2,3}$

Abstract
We introduce and study the problem of Online Continual Compression, where one attempts to simultaneously learn to compress and store a representative dataset from a non i.i.d data stream, while only observing each sample once. A naive application of auto-encoders in this setting encounters a major challenge: representations derived from earlier encoder states must be usable by later decoder states. We show how to use discrete auto-encoders to effectively address this challenge and introduce Adaptive Quantization Modules (AQM) to control variation in the compression ability of the module at any given stage of learning. This enables selecting an appropriate compression for incoming samples, while taking into account overall memory constraints and current progress of the learned compression. Unlike previous methods, our approach does not require any pretraining, even on challenging datasets. We show that using AQM to replace standard episodic memory in continual learning settings leads to significant gains on continual learning benchmarks. Furthermore we demonstrate this approach with larger images, LiDAR, and reinforcement learning agents.

1. Introduction
Interest in machine learning in recent years has been fueled by the plethora of data being generated on a regular basis. Effectively storing and using this data is critical for many applications, especially those involving continual learning. In general, compression techniques can greatly improve data storage capacity, and, if done well, reduce the memory and compute usage in downstream machine learning tasks (Gueguen et al., 2018; Oyallon et al., 2018). Thus, learned compression has become a topic of great interest (Theis et al., 2017; Ballé et al., 2016; Johnston et al., 2018). Yet its application in reducing the size of datasets bound for machine learning applications has been limited.

This work focuses on the following familiar setting: new training data arrives continuously for a learning algorithm to exploit, however this data might not be iid, and furthermore there is insufficient storage capacity to preserve all the data uncompressed. We may want to train classifiers, reinforcement learning policies, or other models continuously from this data as it’s being collected, or use samples randomly drawn from it at a later point for a downstream task. For example, an autonomous vehicle (with bounded memory) collects large amounts of high-dimensional training data (video, 3D lidar) in a non-stationary environment (e.g. changing weather conditions), and overtime applies an ML algorithm to improve its behavior using this data. This data might be transferred at a later point for use in downstream learning. Current learned compression algorithms, e.g. (Torfason et al., 2018), are not well designed to deal with this case, as their convergence speed is too slow to be usable in an online setting.

In the field of continual/lifelong learning (Thrun & Mitchell, 1995), which has for now largely focused on classification, approaches based on storing memories for later use have emerged as some of the most effective in online settings (Lopez-Paz et al., 2017; Aljundi et al., 2018; Chaudhry et al.; 2019; Aljundi et al., 2019). These memories can be stored as is, or via a generative model (Shin et al., 2017). Then, they can either be used for rehearsal (Chaudhry et al., 2019; Aljundi et al., 2019) or for constrained optimization (Lopez-Paz et al., 2017; Chaudhry et al.; Aljundi et al., 2018). Indeed many continual learning applications would be greatly improved with replay approaches if one could afford to store all samples. These approaches are however inherently limited by the amount of data that can be stored.

Learning a generative model to compress the previous data stream thus seems like an appealing idea. However, learning generative models, particularly in the online and non-stationary setting, continues to be challenging, and can greatly increase the complexity of the continual learning task. Furthermore, such models are susceptible to catastrophic forgetting (Aljundi et al., 2019). An alternate approach is to learn a compressed representation of the data,
which can be more stable than learning generative models.

Learned compression in the online and non-stationary setting itself introduces several challenges illustrated in Fig 1. Firstly the learned compression module must be able to decode representations encoded by earlier versions of itself, introducing a problem we refer to as representation drift. Secondly, the learned compressor is itself susceptible to catastrophic forgetting. Finally, the learned compression needs to be adaptive to maintain a prescribed level of reconstruction quality even if it has not fully adapted to the current distribution.

In this work we demonstrate that the VQ-VAE framework (van den Oord et al., 2017; Razavi et al., 2019), originally introduced in the context of generative modeling and density estimation, can be used online to effectively address representation drift while achieving high compression. Furthermore, when augmented with an internal replay mechanism it can overcome forgetting. Finally we use propose to use multiple gradient-isolated compression levels to allow the compressor to adaptively store samples at different compression scales, based on the amount of data, storage capacity, and effectiveness of the model in compressing samples.

The main contributions of this work are: (a) we introduce and highlight the online learned continual compression (OCC) problem and its challenges. (b) We show how representation drift, one of the key challenges, can be tackled by effective use of codebooks in the VQ-VAE framework. (c) We propose an architecture using multiple VQ-VAE’s, adaptive compression scheme, stream sampling scheme, and self-replay mechanism that work together to effectively tackle the OCC problem. (d) We demonstrate this can yield state-of-the-art performance in standard online continual image classification benchmarks and demonstrate the applications of our OCC solution in a variety of other contexts.

2. Related Work

Learned compression has been recently studied for the case of image compression. Works by Theis et al. (2017); Ballé et al. (2016); Johnston et al. (2018) have shown learned compressions can outperform standard algorithms like JPEG. These methods however are difficult to adapt for online settings as they do not directly address the challenges of the OCC problem (e.g. representation drift).

Continual Learning research currently focuses on overcoming catastrophic forgetting (CF) in the supervised learning setting, with some limited work in the generative modeling and reinforcement learning settings. Most continual learning methods can be grouped into three major families.

Some algorithms dynamically change the model’s architecture to incorporate learning from each task separately (Rusu et al., 2016; Li & Hoiem, 2018; Fernando et al., 2017). Although these methods can perform well in practice, their introduction of task-specific weights requires growing compute and memory costs which are problematic for the online setting. Another set of techniques employ regularization to constrain weights updates in the hope of maintaining knowledge from previous tasks. Notable methods in this class include (Kirkpatrick et al., 2017; Huszár, 2017; Zenke et al., 2017; Nguyen et al., 2017; Chaudhry et al., 2018). However, this set of approaches has been shown to be inefficient in the online setting (Chaudhry et al., 2019).

The last family of methods encapsulates all that have a mechanism to store information about the previous data distributions. This memory then serves as a tool for the continual learner to rehearse previous tasks. The simplest instantiation of this method is to keep and sample from a buffer of old data to retrain the model after every update (Chaudhry et al., 2019). This approach is widely used in RL where it is known as Experience Replay (ER) (Lin, 1993; Mnih et al., 2013; Andrychowicz et al., 2016). Another method, known as Generative Replay (GR) (Shin et al., 2017), uses generative modeling to store past task distributions. The continual learner then trains on generated samples to alleviate CF. Other notable examples are Gradient Episodic Memory (GEM) (Lopez-Paz et al., 2017), iCarl (Rebuffi et al., 2017), and Maximally Interfered Retrieval (MIR) (Aljundi et al., 2018).
We consider the problem setting where a stream of samples arrives from different distributions $D_t$ changing over time $t = 1 \ldots T$. We have a fixed storage capacity of $C$ bytes where we would like to store the most representative information from all data distributions $D_1, \ldots D_T$. There is notably a trade-off in quality of information versus the amount of samples stored. We propose to use a learned compression model, and most crucially, this model must also be stored within the $C$ bytes, to encode and decode the data samples. Another critical requirement is that at anytime $t$ the content of the storage (data and/or compression model) be usable for downstream applications. An important challenge, illustrated in Figure 1, is that the learned compression module will change over time, while we still need to be able to decode the memories in storage.

3.2. Vector Quantized VAE for Online Compression

The VQ-VAE is a discrete auto-encoder which relies on a vector quantization step to obtain discrete latent representations. An embedding table, $E \in \mathbb{R}^{K \times D}$ consisting of $K$ vectors of size $D$, is used to quantize encoder outputs. Given an input (e.g. an RGB image), the encoder first encodes it as a $H_h \times W_h \times D$ tensor, where $H_h$ and $W_h$ denote the height and width of the latent representation. Then, every $D$ dimensional vector is quantized using a nearest-neighbor lookup on the embedding table.

In this section we outline our approach to the online continual compression problem. First we will review the VQ-VAE and highlight the properties making it effective for representational drift. Then we will describe our adaptive architecture, storage, and sampling scheme.

3.1. Problem Setting: Online Continual Compression

Lidar compression is considered in (Tu et al., 2019) and (Caccia et al., 2018). Both approaches use a similar projection from 3D $(x, y, z)$ coordinates to 2D cylindrical coordinates, and leverage deep generative models to compress the data. However, neither accounts for potential distribution shift, nor for online learning. In this work we show that using this 2D projection in conjunction with our model allows us to mitigate the two issues above for lidar data.

3. Methodology

Critically, the embedding tables are updated independently from the encoder and decoder, namely by minimizing $\min_{e} \|sg[enc(x)]_{ij} - e\|$, where $sg$ is the stop gradient operator.

Observe in the case of online compression, if the embedding table is fixed, then a change in the encoder parameters and therefore a change in the encoder output for a given input will not change the final quantized representation $z$, unless it is sufficiently large, thus we can observe that if the embeddings change slowly or are fixed we can greatly improve our control of the representational drift. This effect is illustrated in Figure 2. On the other hand we do need to adapt the embedding table, since randomly selected embeddings would not cover well the space of encoder outputs.

3.3. Adaptive Quantization Modules

To address issues of how to optimize storage and sampling in the context of Online Continual Compression we introduce Adaptive Quantization Modules (AQM). We use AQM to collectively describe the architecture, adaptive multi-level storage mechanism, and data sampling method used. Together they allow effectively constraining individual sample quality and memory usage while keeping storage a faithful representation of the overall distribution.

AQM uses an architecture consisting of a sequence of VQ-VAEs, each with a buffer. The AQM approach to online continual compression is overall summarized in Algorithm
3.3.1. ARCHITECTURE AND TRAINING

Each AQM module contains a VQ-VAE and a corresponding buffer of adaptive capacity. A diagram of the architecture is given in Figure 3. We will denote the output after quantization of each module \( i \) as \( z_q^i \) and the set of codebook indexes used to obtain \( z_q^i \) as \( a^i \). Note that \( a^i \) are the discrete representations we actually store. Each subsequent module produces and stores an \( a^i \) requiring fewer bits to represent.

For RGB images, the compression rate at a given level is given by \( \frac{H \times W \times 3 \times \log_2(256)}{N_c \times H_{hi} \times W_{hi} \times |\log_2(K_i)|} \). Here \( K_i \) is the number of embeddings in the codebooks, \( (H_{hi}, W_{hi}) \) the spatial dimension of the latent representation and \( N_c \) the number of codebooks.

VQVAE-2 (Razavi et al., 2019) also uses a multi-scale hierarchical organization, where unlike with our AQM, the top level models global information such as shape, while the bottom level, conditioned on the top one, models local information. This architecture is tailored for generative modeling, it is less attractive for compression, as both the bottom and top quantized representations must be stored for high-quality reconstructions. Furthermore in AQM each module is learned in a greedy manner using the current estimate of \( z_q^{i-1} \) without passing gradients between modules similar to (Belilovsky et al., 2019; Nøkland & Eidnes, 2019). A subsequent module is not required to build representations which account for all levels of compression, thus minimizing interference across resolutions. This allows the modules to each converge as quickly as possible with minimal drift at their respective resolution, particularly important in the online continual learning case.

### Algorithm 1: AQM LEARNING WITH SELF-REPLAY

**Input:** Learning rate \( \alpha \), EXTERNALLEARNER

**Initialize:** AQM Memory \( \mathcal{M} \); AQM Parameters \( \theta_{aqm} \)

for \( t \in 1..T \) do

\% Fetch data from current task

for \( n \in 1..N \) do

\( B \leftarrow B_{inc} \)

if \( t > 1 \) then

\% Fetch data from buffer

\( B_{re} \sim \text{SAMPLE}(\mathcal{M}, \theta_{aqm}) \)

\( B \leftarrow (B_{inc}, B_{re}) \)

end

\% Update AQM

\( \theta_{aqm} \leftarrow \text{ADAM}(\theta_{aqm}, B, \alpha) \)

\% Send data to external learner

\text{EXTERNALLEARNER}(B)

if \( t > 1 \) then

\% Update indices

\text{UpdateBufferRep}(\mathcal{M}, \theta_{aqm})

\% Save current indices

\text{AddToMemory}(\mathcal{M}, B_{inc}, \theta_{aqm})

end

end

Algorithm 1: AQM LEARNING WITH SELF-REPLAY

**Input:** Learning rate \( \alpha \), EXTERNALLEARNER

**Initialize:** AQM Memory \( \mathcal{M} \); AQM Parameters \( \theta_{aqm} \)

for \( t \in 1..T \) do

\% Fetch data from current task

for \( n \in 1..N \) do

\( B \leftarrow B_{inc} \)

if \( t > 1 \) then

\% Fetch data from buffer

\( B_{re} \sim \text{SAMPLE}(\mathcal{M}, \theta_{aqm}) \)

\( B \leftarrow (B_{inc}, B_{re}) \)

end

\% Update AQM

\( \theta_{aqm} \leftarrow \text{ADAM}(\theta_{aqm}, B, \alpha) \)

\% Send data to external learner

\text{EXTERNALLEARNER}(B)

if \( t > 1 \) then

\% Update indices

\text{UpdateBufferRep}(\mathcal{M}, \theta_{aqm})

\% Save current indices

\text{AddToMemory}(\mathcal{M}, B_{inc}, \theta_{aqm})

end

end

\[ \text{Algorithm 1: AQM LEARNING WITH SELF-REPLAY} \]

\[ \text{Input: Learning rate } \alpha, \text{ EXTERNALLEARNER} \]

\[ \text{Initialize: AQM Memory } \mathcal{M}; \text{ AQM Parameters } \theta_{aqm} \]

\[ \text{for } t \in 1..T \text{ do} \]

\[ \% \text{ Fetch data from current task} \]

\[ \text{for } n \in 1..N \text{ do} \]

\[ B \leftarrow B_{inc} \]

\[ \text{if } t > 1 \text{ then} \]

\[ \% \text{ Fetch data from buffer} \]

\[ B_{re} \sim \text{SAMPLE}(\mathcal{M}, \theta_{aqm}) \]

\[ B \leftarrow (B_{inc}, B_{re}) \]

\[ \text{end} \]

\[ \% \text{ Update AQM} \]

\[ \theta_{aqm} \leftarrow \text{ADAM}(\theta_{aqm}, B, \alpha) \]

\[ \% \text{ Send data to external learner} \]

\[ \text{EXTERNALLEARNER}(B) \]

\[ \text{if } t > 1 \text{ then} \]

\[ \% \text{ Update indices} \]

\[ \text{UpdateBufferRep}(\mathcal{M}, \theta_{aqm}) \]

\[ \% \text{ Save current indices} \]

\[ \text{AddToMemory}(\mathcal{M}, B_{inc}, \theta_{aqm}) \]

\[ \text{end} \]

\[ \text{end} \]

**3.3.2. MULTI-LEVEL STORAGE**

Our aim is to store the maximum number of samples in an allotted \( C \) bytes of storage, while ensuring that their quality, and our ability to reconstruct them. Samples are thus stored at different levels based on the compressors’ current ability. The process is summarized in Algorithm 2.

### Algorithm 2: ADAPTIVECOMPRESS

**Input:** datapoint \( x \), AQM with \( L \) modules, threshold \( d_{th} \)

\% Forward all modules, store encodings

\( \{z_q^i, a^i\}_{i=1..L} = \text{ENCODE}(x) \)

for \( i \in 1..L \) do

\% Decode from level \( i \) to output space

\( \hat{x} = \text{DECODER}(z_q^i) \)

\% Check reconstruction error

\( \text{if } \text{MSE}(\hat{x}, x) < d_{th} \text{ then return } a_i, i \)

\% Otherwise, return original input

\( \text{return } x, 0 \)

**Algorithm 2: ADAPTIVECOMPRESS**

**Input:** AQM with \( L \) modules, threshold \( d_{th} \)

\% Forward all modules, store encodings

\( \{z_q^i, a^i\}_{i=1..L} = \text{ENCODE}(x) \)

for \( i \in 1..L \) do

\% Decode from level \( i \) to output space

\( \hat{x} = \text{DECODER}(z_q^i) \)

\% Check reconstruction error

\( \text{if } \text{MSE}(\hat{x}, x) < d_{th} \text{ then return } a_i, i \)

\% Otherwise, return original input

\( \text{return } x, 0 \)

For RGB images, the compression rate at a given level is given by \( \frac{H \times W \times 3 \times \log_2(256)}{N_c \times H_{hi} \times W_{hi} \times |\log_2(K_i)|} \). Here \( K_i \) is the number of embeddings in the codebooks, \( (H_{hi}, W_{hi}) \) the spatial dimension of the latent representation and \( N_c \) the number of codebooks.

VQVAE-2 (Razavi et al., 2019) also uses a multi-scale hierarchical organization, where unlike with our AQM, the top level models global information such as shape, while the bottom level, conditioned on the top one, models local information. This architecture is tailored for generative modeling, it is less attractive for compression, as both the bottom and top quantized representations must be stored for high-quality reconstructions. Furthermore in AQM each module is learned in a greedy manner using the current estimate of \( z_q^{i-1} \) without passing gradients between modules similar to (Belilovsky et al., 2019; Nøkland & Eidnes, 2019). A subsequent module is not required to build representations which account for all levels of compression, thus minimizing interference across resolutions. This allows the modules to each converge as quickly as possible with minimal drift at their respective resolution, particularly important in the online continual learning case.
Online Learned Continual Compression with Stacked Quantization Modules

Figure 3. Architecture of Adaptive Quantization Modules. Each level uses its own loss and maintains its own replay buffer. Yellow dotted lines indicate gradient isolation between modules.

3.3.3. Self-Replay and Stream Sampling

As shown in Alg. 1 our AQM is equipped with an internal experience replay mechanism (Mnih et al., 2013), which reconstructs a random sample from storage and uses it to perform an update to the AQM modules, while simultaneously freeing up overall memory if the sample can now be compressed at a later AQM module. This has the effect of both reducing forgetting and freeing memory. In practice we replay at the same rate as incoming samples arrive and thus replay will not increase asymptotic complexity of the online learning. Finally, for efficiency the replay can be coupled to an external online learner querying for random samples from the overall memory.

Since we would like the AQM to work in cases of a fixed memory capacity it must also be equipped with a mechanism for selecting which samples from the stream to stores and which to delete from memory. Reservoir Sampling (RS) is a simple yet powerful approach to this problem, used successful in continuain learning (Chaudhry et al., 2019). It adds a sample from the stream with prob. \( p = \frac{\text{buffer capacity}}{\text{points seen so far}} \) while remove a random sample. However, RS is not directly compatible with AQM primarily because the amount of samples that can be stored varies over time. This is because samples at different levels have different memory usage and memory can be freed by replay. We thus propose an alternative scheme, which maximally fills available memory and selects non-uniformly samples for deletion. Specifically when a larger amount of samples are added at one point in the stream, they become more likely to be removed. The details of this stream sampling method are provided in Appendix A.

3.3.4. Drift Control via Codebook Stabilization

As mentioned previously, a good online compressor must control its representational drift, which occurs when updates in the auto-encoder parameters creates a mismatch with the static representations in the buffer. Throughout the paper we measure representational drift by comparing the following time varying quantity: \( \text{DRIFT}_t(z) = \text{RECON ERR}(\text{Decode}(\theta_t; z), x) \) where \( \theta_t \) the model parameters at time \( t \) and \( (z, x) \) is a stored compressed representation and its original uncompressed datapoint respectively. For all experiments on images, \( \text{RECON ERR} \) is simply the mean squared error.

As illustrated in Sec 3.2 a slow changing codebook can allow to control drifting representations. This can be in part accomplished by updating the codebook with an exponential moving average as described in (van den Oord et al., 2017, Appendix A), where it was used to reduce the variance of codebook updates. This alone is insufficient to fully control drift, thus once a given module yields satisfactory compressions on the data stream, we freeze the module’s embedding matrix but leave encoder and decoder parameters free to change and adapt to new data. Moreover, we note that fixing the codebook for a given module does not affect the reconstruction performance of subsequent modules, as they only need access to the current module’s decoder which can still freely change.

4. Experiments

We evaluate the efficacy of the proposed methods on a suite of canonical and new experiments. In Section 4.1 we present results on standard supervised continual learning benchmarks on CIFAR-10. In Section 4.2 we evaluate other downstream tasks such as standard iid training applied on the storage at the end of online continual compression. For this evaluation we consider larger images from Imagenet, as well as on lidar data. Finally we apply AQM on observations of an agent in an RL environment.

4.1. Online Continual Classification

Although CL has been studied in generative modeling (Ramapuram et al., 2017; Lesort et al., 2018; Zhai et al., 2019; Lesort et al., 2019) and reinforcement learning (Kirkpatrick et al., 2017; Fernando et al., 2017; Riemer et al., 2018), supervised learning is still the standard for evaluation of new methods. Thus, we focus on the online continual classification of images for which our approach can provide a complement to experience replay. In this setting, a new
Table 1. Shared head results on disjoint CIFAR-10. Total memory per class $M$ measured in sample memory size. We report (a) Accuracy, (b) Forgetting (lower is better).

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Accuracy ($\uparrow$)</th>
<th>$M = 20$</th>
<th>$M = 50$</th>
<th>Forgetting ($\downarrow$)</th>
<th>$M = 20$</th>
<th>$M = 50$</th>
</tr>
</thead>
<tbody>
<tr>
<td>iid online</td>
<td>60.8 ± 1.0</td>
<td>60.8 ± 1.0</td>
<td>N/A</td>
<td>73.5 ± 1.7</td>
<td>70.7 ± 4.5</td>
<td></td>
</tr>
<tr>
<td>iid offline</td>
<td>79.2 ± 0.4</td>
<td>79.2 ± 0.4</td>
<td>N/A</td>
<td>50.5 ± 2.4</td>
<td>35.4 ± 2.0</td>
<td></td>
</tr>
<tr>
<td>GEM (Lopez-Paz et al., 2017)</td>
<td>16.8 ± 1.1</td>
<td>17.1 ± 1.0</td>
<td>N/A</td>
<td>49 ± 2.4</td>
<td>40.6 ± 1.1</td>
<td></td>
</tr>
<tr>
<td>iCarl (5 iter)</td>
<td>28.6 ± 1.2</td>
<td>33.7 ± 1.6</td>
<td>N/A</td>
<td>85.4 ± 0.7</td>
<td>85.4 ± 0.7</td>
<td></td>
</tr>
<tr>
<td>fine-tuning</td>
<td>18.4 ± 0.3</td>
<td>18.4 ± 0.3</td>
<td>N/A</td>
<td>50.5 ± 2.4</td>
<td>35.4 ± 2.0</td>
<td></td>
</tr>
<tr>
<td>ER</td>
<td>27.5 ± 1.2</td>
<td>33.1 ± 1.7</td>
<td>N/A</td>
<td>50.2 ± 2.0</td>
<td>30.2 ± 2.3</td>
<td></td>
</tr>
<tr>
<td>ER-MIR (Aljundi et al., 2019)</td>
<td>29.8 ± 1.1</td>
<td>40.0 ± 1.1</td>
<td>N/A</td>
<td>71.5 ± 2.8</td>
<td>67.2 ± 3.9</td>
<td></td>
</tr>
<tr>
<td>Gumbel AE (Riemer et al., 2018)</td>
<td>25.5 ± 2.0</td>
<td>28.8 ± 2.9</td>
<td>N/A</td>
<td>50.5 ± 1.1</td>
<td>42.8 ± 1.3</td>
<td></td>
</tr>
<tr>
<td>AQM (ours)</td>
<td>39.9 ± 0.8</td>
<td>46.2 ± 0.8</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Forgetting ($\downarrow$) is measured as the decrease in accuracy from the last task, and therefore subset of classes, is provided explicitly to the learner (Farquhar & Gal, 2018; Aljundi et al., 2019).

For this set of experiments, we report accuracy, i.e. $\frac{1}{T} \sum_{i=1}^{T} R_{T,i}$, and forgetting, i.e. $\frac{1}{T-1} \sum_{i=1}^{T-1} \max(R_{i,i}) - R_{T,i}$ with $R \in \mathbb{R}^{T \times T}$ representing the accuracy matrix where $R_{i,j}$ is the test classification accuracy on task $j$ when task $i$ is completed.

**Baselines** A basic baseline for continual supervised learning is Experience Replay (ER). It consists of storing old data in a buffer to replay old memories. Although relatively simple recent research has shown it is a critical baseline to consider, and in some settings is actually state-of-the-art (Chaudhry et al., 2019; Aljundi et al., 2019; Rolnick et al., 2018). AQM can be used as an add-on to ER that incorporates online continual compression. In addition we consider the following baselines. **iid online** (upper-bound) trains the model with a single-pass through the data on the same set of samples, but sampled iid. **iid offline** (upper-bound) evaluates the model using multiple passes through the data, sampled iid. We use 5 epochs in all the experiments for this baseline. **fine-tuning** trains continuously upon arrival of new tasks without any forgetting avoidance strategy. **iCarl** (Rebuffi et al., 2017) incrementally classifies using a nearest neighbor algorithm, and prevents catastrophic forgetting by using stored samples. **GEM** (Lopez-Paz et al., 2017) uses stored samples to avoid increasing the loss on previous task through constrained optimization. It has been shown to be a strong baseline in the online setting. It gives similar results to the recent A-GEM (Chaudhry et al.). **ER-MIR** (Aljundi et al., 2019) controls the sampling of the replays to bias sampling towards samples that will be forgotten. We note that the ER-MIR criteria is orthogonal to AQM, and both can be applied jointly. **Gumbel AE** (Riemer et al., 2018) learns an autoencoder for ER using the Gumbel softmax to obtain discrete representations.

We evaluate with the standard CIFAR-10 split (Aljundi et al., 2018), where 5 tasks are presented sequentially, each adding two new classes. Evaluations are shown in Table 1. Due to our improved storage of previous data, we observe significant improvement over other baselines at various memory sizes. We can contrast AQM’s performance with ER’s to understand the net impact of our compression scheme. Specifically, AQM improves over ER by 12.4% and 13.1% in the $M=20$ and $M=50$ case, highlighting the effectiveness of online compression. Our approach only lags ER-MIR in forgetting in the $M=50$ setting. However, this method is orthogonal to ours and could be used jointly.

To compare directly to reporting in (Riemer et al., 2018) we also benchmarked our implementation on the Incremental CIFAR-100 multi-head experiment (Lopez-Paz et al., 2017) with the same settings as in (Riemer et al., 2018). By using AQM we were able to get 65.3 vs the reported 43.7 using a buffer of size 200. To specifically isolate the advantage of gumbel softmax versus the vector quantization for drift, we replaced the vector quantization approach with gumbel softmax in an AQM. We observed significantly less drift in the case where vector quantization is used. Full details of this experiment are described in the supplementary materials along with visualizations.

The CIFAR-10 dataset has a low resolution ($3 \times 32 \times 32$) and uses a lot of data per task (10K samples). These two characteristics might leave the online compression problem easier than in a real-life scenario. Specifically, if the first tasks are long enough and the compression rate is not too large, the model can quickly converge and thus not incur too much representation drift. Indeed, we found that using a single module is already sufficient for this task. For
Online Learned Continual Compression with Stacked Quantization Modules

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>5.2 ± 0.2</td>
</tr>
<tr>
<td>2 Module AQM (ours)</td>
<td>23.2 ± 1.1</td>
</tr>
<tr>
<td>Ablate 2nd Module</td>
<td>20.5 ± 1.3</td>
</tr>
<tr>
<td>Ablate Fixing Codebook</td>
<td>19.2 ± 0.6</td>
</tr>
<tr>
<td>Ablate Decoupled Training</td>
<td>16.5 ± 0.7</td>
</tr>
<tr>
<td>Ablate Adaptive Compression</td>
<td>13.1 ± 3.2</td>
</tr>
</tbody>
</table>

Table 2. Imagenet offline training evaluation from online continual compression. We see a clear gain over a standard Reservoir sampling approach. We then ablate each component of our proposal showing each component is important. Note storage used in each experiment is identical (including accounting for model sizes). These reasons, we now study the AQM in more challenging settings presented in the next section.

4.2. Offline Evaluation on Larger Images

Besides the standard continual classification setup, we propose several other evaluations to determine the effectiveness of the stored data and compression module after learning online compression. We also perform a detailed ablation to study the efficacy of each component in AQM.

Offline training on Imagenet We compare the effectiveness of the stored memories of AQM after a certain amount of online continual compression. We do this by training in a standard iid way an offline classification model using only reconstructions obtained from the storage sampled after online continual compression has progressed for a period of time. In each case we would have the same sized storage available. We note that simply having more stored memories does not amount to better performance as their quality may be severely degraded and affected by drift.

Using this evaluation we first compare a standard reservoir sampling approach on uncompressed data to a 2 module AQM using the same size storage. We observe that performance is drastically increased using the compressed samples. We then use this to perform a series of ablations to demonstrate each component of our proposal is important. Specifically (a) we restrict AQM to have only one module, (b) instead of decoupled training we train modules end-to-end, (c) we remove adaptive compression, thus all samples are stored in the most compressed block, regardless of quality, and (d) we do not stabilize the codebook, the embedding matrices of every block are never fixed. We observe that all these elements contribute to successfully storing a representative set of data for the distribution online.

Drift Ablation We have seen the importance of codebook freezing when dealing with high dimensional datasets. However, judging solely from the final downstream task performance it’s difficult to see if the model continues adapting after freezing. As alluded in Sec 3.2 there is a tradeoff between keeping recoverable representations and a model’s ability to continue to adapt. To shed some light on this, we run the following experiment: we run a vanilla VQ-VAE on the same 20 task mini-imagenet stream, without storing any samples. When it reaches a pre-specified performance threshold, we fix the codebook, and store compressed held-out data from the first task. We then continue to update the VQ-VAE parameters, and the memory is kept fixed for the rest of the stream. We apply self-replay but no other AQM mechanisms (e.g. no sampling from the input stream and no adaptive compression).

We monitor how well the VQ-VAE can adapt by looking at the streaming reconstruction cost, measured on the incoming data before an update. We also monitor the drift of samples stored in the buffer. Results are presented in Figure 4. They demonstrate that drift is controlled by stabilizing the codebook, while the model can still improve at nearly the same rate. Further analysis is included in the Appendix.

LiDAR Range data can be very large and storage inefficient, it is also often collected by vehicles in non-stationary environments. Efficient storage can be important for having more representative data in downstream applications. We thus evaluate data collection with AQM on the Kitti Dataset (Geiger et al., 2013), which contains 61 LiDAR scan recordings, each belonging to either the “residential”, “road”, “city” environments. The online compression is presented one by one with scans from each environment. The data is processed as in (Caccia et al., 2018), where points from the same elevation angle are sorted in increasing order of azimuth angle along the same row. This yield a 2D grid, making it compatible with the same architecture used in the previous experiments. As in (Caccia et al., 2018; Tu et al., 2019), we report the reconstruction cost in Symmetric Nearest Neighbor Root Mean Squared Error (SNNRMSE) which allows to compare two point clouds. Note AQM can also be adapted to use task relevant criteria besides MSE.

We show qualitative results in Figure 5 and in the supplementary materials. Observe that we are able to effectively re-
Online Learned Continual Compression with Stacked Quantization Modules

4.3. Atari RL Environments

Another application of online continual compression is for preserving the states of an reinforcement learning agent operating online. These agents may often learn new tasks or enter new rooms thus the observations will often be highly non-iid. Furthermore many existing reinforcement learning algorithms already rely on potentially large replay buffers which can be prohibitive (Mnih et al., 2014; Rolnick et al., 2018) to run and may greatly benefit from an approach such as the AQM to run concurrently with reinforcement learning algorithms. We thus perform a proof of concept for the AQM for storing the state sequence encountered by an RL learner in the atari environment (Bellemare et al., 2013). We use the dataset and tasks introduced in (Anand et al., 2019), which runs a random or learned policy in the atari environments and provides a set of classification tasks to evaluate whether key information about the state is preserved. Results are shown Table 3. We run the online learning with the AQM on the data stream observed by the random agent. We use the same observations and optimization as in (Anand et al., 2019) and report the F1 results of a linear probe directly on states for our reconstructions after online compression and the originals. Results for 3 environments are shown in Table 3 and examples in in Fig 6 and the Appendix. We find that AQM can well preserve the critical information while compressing the state by 16x. The reference accuracies achieved by our classifier are similar to those in (Anand et al., 2019). However, we do not control for the representation size unlike those evaluations of various unsupervised models.

Table 3. Results on RL probing tasks from (Anand et al., 2019) with linear probe applied to original observation and to reconstructions from AQM after online compression. Acc is averaged for each game over game specific prediction.

<table>
<thead>
<tr>
<th>Game</th>
<th>Cls Input</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pong</td>
<td>Orig. State</td>
<td>86.7</td>
</tr>
<tr>
<td></td>
<td>AQM Recon</td>
<td>86.8</td>
</tr>
<tr>
<td>Ms Pacman</td>
<td>Orig. State</td>
<td>89.4</td>
</tr>
<tr>
<td></td>
<td>AQM Recon</td>
<td>88.3</td>
</tr>
<tr>
<td>Pitfall</td>
<td>Orig. State</td>
<td>68.2</td>
</tr>
<tr>
<td></td>
<td>AQM Recon</td>
<td>66.7</td>
</tr>
</tbody>
</table>

5. Conclusion

We have introduced online continual compression. We demonstrated vector quantization can be used to control drift and how to create mechanisms that allow maintaining quality and maximizing memory usage. These allowed learning compression while compressing. We have shown effectiveness of this online compression approach on standard continual classification benchmarks, as well as for compressing larger images, lidar, and atari data. We believe future work can consider dealing with temporal correlations for video and reinforcement learning tasks, as well as improved prioritization of samples for storage.

References


Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. Learning to learn...