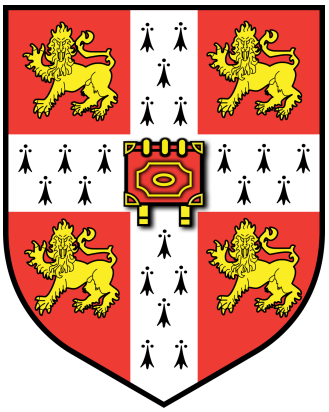


UNIVERSITY OF CAMBRIDGE

MLMI4

ADVANCED MACHINE LEARNING

Variational Continual Learning

Authors:
David Goldfarb
Vishaal Udandarao
Alejandro Santorum Varela

Total word count: 4971

March 30, 2022

Contents

Introduction

In the following report, *Variational Continual Learning* [1] is replicated and the ensuing results are compared to the original authors'. The original work is extended through the use of Convolutional Neural Networks, introduction of the local reparameterization trick, and examination of Variational Generative Replay [2] in order to improve both accuracy and performance.

Continual learning, also known as incremental learning, is a general version of online learning where data arrive continuously in a possibly non i.i.d. way: tasks can change over time or new tasks can appear. Continual learning methods must adapt to fit all tasks without needing to be retrained on previously-seen data. Continual learning is an active research area because real world tasks usually evolve over time and dataset sizes grow exponentially thereby increasing the cost of retraining. Additionally, many situations with strong inter-task similarities would benefit from being modelled jointly to leverage multi-task transfer learning.

One significant issue in continual learning is maintaining previous knowledge from explored data while learning new data. Often times, naive methods suffer from *catastrophic forgetting*, wherein performance on past tasks degrades significantly as new tasks are introduced [3]. On the contrary, never learning new task data is also undesirable due to failing to address the fundamental continual learning problem. This trade-off between *stability*, maintaining performance on old tasks, and *plasticity*, adapting to new tasks, is

the heart of continual learning and one which Variational Continual Learning seeks to address.

Thus, selection of this paper was motivated by both the practicality of continual learning in the contemporary computing landscape and the novelty of exploring the stability-plasticity trade-off.

The approach developed in *Variational Continual Learning* is to use a single model, but maintain uncertainty over its parameters to model a large number of tasks generically and automatically.

A general framework for taking this approach to continual learning is Bayesian Inference. Bayesian inference retains a distribution over model parameters that indicates the plausibility of any setting given the observed data. When new data arrive, we combine what previous data have told us about the model parameters, the previous posterior, with what the current data are telling us, the likelihood. The new posterior is obtained after multiplying and re-normalising these two quantities. However, exact Bayesian Inference is often intractable and so the posteriors must be estimated using approximate methods for neural networks. Notably, the merging of online variational inference (VI) and Monte Carlo VI for neural networks to yield Variational Continual Learning (VCL). Moreover, VCL is extended through combination with a past-data summarisation method reminiscent of an episodic memory termed a *coreset*.

Before continuing with the remainder of the report, it should be noted that reference implementations of VCL, VGR, and The Local Reparameterisation Trick were used as starting points from which to replicate results and construct our aforementioned extensions. Our final code and experimental results can be found on Github.

Continual Learning by Approximate Bayesian Inference

The continual learning inference problem can be introduced by considering a stream of T datasets, each arriving at time t, of some size N_t, and consisting of input-output pairs $\boldsymbol{D}_t = \{x_t^{(n)}, y_t^{(n)}\}_{n=1}^{N_t}$. In the formation of a discriminative model operating in this domain, we seek some distribution which places a probability over the output given the input and some parameters $\boldsymbol{\theta}$.

To determine appropriate $\boldsymbol{\theta}$ for this model, we establish a posterior distribution of the parameters: $p(\boldsymbol{\theta}|\boldsymbol{D}_{1\to T})$. If we are to take a Bayesian approach considering all datasets to be independent, we can deconstruct this posterior into a prior $p(\boldsymbol{\theta})$ and likelihood, thus allowing us to fashion the proportional relationship:

$$p(\boldsymbol{\theta}|\boldsymbol{D}_{1\to T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^{T} p(\boldsymbol{D}_t|\boldsymbol{\theta})$$

Notably, we can compress the above proportionality to a more concise, incremental update rule suitable for continual learning. At some time t, an un-normalized posterior can always be found through:

$$p(\boldsymbol{\theta}|\boldsymbol{D}_{1\to t-1})p(\boldsymbol{D}_t|\boldsymbol{\theta})$$

However, due to the frequent intractability of this posterior, approximations must be developed. For the purposes of novelty, especially in the context of continual learning with neural networks, VCL is explored.

VCL utilizes KL divergence to form an approximate distribution over $\boldsymbol{\theta}$ after exposure to t datasets, $q_t(\boldsymbol{\theta})$, which exists in some family of posteriors Q. In particular,

$$q_t(\boldsymbol{\theta}) = \arg\min_{q\in Q} KL(q(\boldsymbol{\theta})||\frac{1}{Z(t)}q_{t-1}(\boldsymbol{\theta})p(\boldsymbol{D}_t|\boldsymbol{\theta}))$$

Although this method is capable of exactly recovering the posterior $p(\boldsymbol{\theta}|\boldsymbol{D}_{1\to t})$ if such a distribution is representable within Q, there are numerous sources of potential errors which can accrue. To ameliorate these errors, a *coreset* is used. A coreset, C_t, is a subset of points in $\boldsymbol{D}_{1\to t}$ which is introduced before prediction intended to "remind" a model about past datasets (similar to an *episodic memory* [4]).

Methods

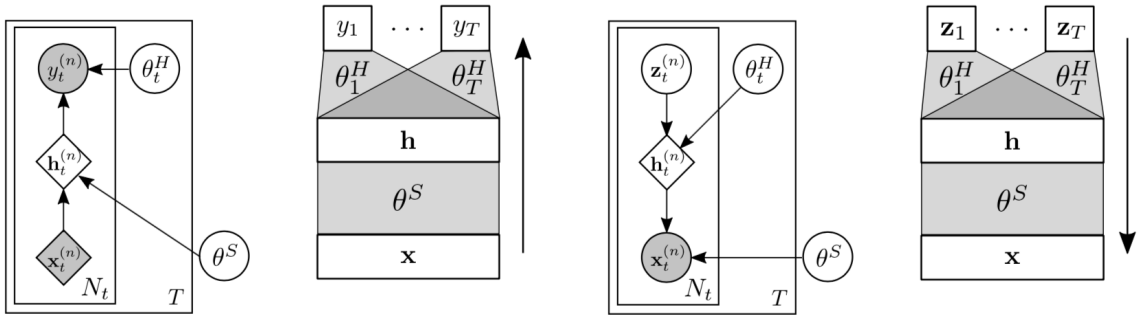
Variational Continual Learning in Deep Discriminative Models

VCL can be applied to a variety of discriminative probabilistic models. In this project, we choose deep fully-connected (FC) neural networks. If the data arrive in an i.i.d. way or only the input distribution $p(\boldsymbol{D}_{1\to T})$ changes over time, then a *single-head* discriminative neural network suffices. One example of such a task is classification of cats and dogs over different species: the same task head can be used to distinguish between a Siamese/Pomeranian and a Tabby/Husky. In other cases the tasks might involve different output variables, so *multi-head* networks are used to predict each output even though the first layers might share their parameters. One example of this situation is one task which distinguishes between pictures of an apartment or house, and another task which distinguishes between pictures of a cars, trucks, and boats. The task heads are of different size and classes, but share a common, early set of parameters for common, low-level image feature extraction.

This multi-task learning approach is further studied in [8] and illustrated in Fig. 1.

For VCL $q(\boldsymbol{\theta})$ has to be specified, where $\boldsymbol{\theta}$ is a D dimensional vector formed by stacking the parameters of the network. We use a Gaussian mean-field approximate posterior $q_t(\boldsymbol{\theta}) = \prod_{d=1}^{D}\mathcal{N}(\theta_{t,d};\mu_{t,d},\sigma_{t,d}^2)$ (MFVI). Furthermore, a multivariate Gaussian with diagonal covariance is used as a prior, $p(\boldsymbol{\theta})$.

Taking the most general case, before task k is modelled, the posterior distribution over the k-th head parameters will remain at the prior, $q_t(\boldsymbol{\theta}_k^H) = p(\boldsymbol{\theta}_k^H)$. This means the variational approximation can be grown incrementally, starting from the prior, as each task is encountered. Additionally, only the posterior distributions over head parameters whose



(a) Multi-head discriminative network (b) Multi-head generative network

Figure 1: Illustrations of multi-head networks investigated in this project, including the graphical model (left) and network architecture (right). Figure 1a shows how some network parameters $\boldsymbol{\theta}^S$ might be shared in the first layers during training, and how each task t has its own head parameterised with $\boldsymbol{\theta}_t^H$. Figure 1b represents a multi-head generative model with shared parameters. The head of the network generate the intermediate level representations from the latent variables $\mathbf{z}$. Image taken from [1]

tasks are present in the current dataset D_t need to be updated. The network is trained by minimizing the negative online variational free-energy or maximizing the variational lower bound to the online marginal likelihood

$$\mathscr{L}_{\text{VCL}}^t(q_t(\boldsymbol{\theta})) = \sum_{n=1}^{N_t} \mathbb{E}_{\boldsymbol{\theta}\sim q_t(\boldsymbol{\theta})}\left[\log(p(y_t^{(n)})|\boldsymbol{\theta},\mathbf{x}_t^{(n)})\right] - KL(q_t(\boldsymbol{\theta})||q_{t-1}(\boldsymbol{\theta})) \tag{1}$$

with respect to the variational parameters $\{\mu_{t,d},\sigma_{t,d}\}_{d=1}^D$.

Since we are using MFVI Gaussians for $q_t(\boldsymbol{\theta})$, KL-divergence can be computed analytically, however, the expected log-likelihood requires approximation. Simple Monte Carlo as well as the (local) reparameterization trick to compute the gradients [9, 10] are applied to bridge intractability.

Variational Continual Learning in Deep Generative Models

Deep generative models (DGMs) have been growing in popularity recently. They are based in passing a simple noise variable through a deep neural network, and have been shown to be capable of generating realistic images [5], sounds [6], and videos sequences [7]. In this project we extend VCL to include variational auto-encoders (VAEs) [16].

Let $p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})p(\mathbf{z})$ be a model for observed data $\mathbf{x}$ and latent variables $\mathbf{z}$. Typically, the prior over latent variables $p(z)$ is Gaussian, and the parameters $p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})$ are defined by a deep neural network. Given a dataset $D = \{\mathbf{x}^{(n)}\}_{n=1}^N$, a standard VAE would learn the parameters $\boldsymbol{\theta}$ by approximating the maximum likelihood estimator (MLE). This done by maximizing the variational lower bound with respect to $\boldsymbol{\theta}$ and ϕ:

$$\mathscr{L}_{\text{VAE}}(\boldsymbol{\theta},\phi) = \sum_{n=1}^{N} \mathbb{E}_{q_\phi(\mathbf{z}^{(n)}|\mathbf{x}^{(n)})}\left[\log \frac{p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)},\boldsymbol{\theta})p(\mathbf{z}^{(n)})}{q_\phi(\mathbf{z}^{(n)}|\mathbf{x}^{(n)})}\right] \tag{2}$$

where ϕ are the variational parameters of the approximate posterior or "encoder" $q_\phi(\boldsymbol{z}|\boldsymbol{x})$.

However, approximation using MLE is not desirable in continual learning because MLE does not provide uncertainty estimates for the decoder weights $\boldsymbol{\theta}$ which are crucial for weighting the information learned from old data. VCL instead approximates the full posterior distribution over decoder parameters, $q_t(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|D_{1\to T})$, after observing the t-th dataset by maximizing the full variational lower bound with respect to q_t and ϕ:

$$\begin{aligned} &\mathscr{L}^t_{\text{VCL}}(q_t(\boldsymbol{\theta}), \phi) \\ &= \mathbb{E}_{q_t(\boldsymbol{\theta})}\left[\sum_{n=1}^{N_t} \mathbb{E}_{q_\phi(\mathbf{z}^{(n)}|\mathbf{x}^{(n)})}\left[\log \frac{p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}, \boldsymbol{\theta}) p(\mathbf{z}^{(n)})}{q_\phi(\mathbf{z}^{(n)}|\mathbf{x}^{(n)})}\right]\right] - KL(q_t(\boldsymbol{\theta})||q_{t-1}(\boldsymbol{\theta})) \end{aligned} \tag{3}$$

where the encoder network $q(\mathbf{z}^{(n)}|\mathbf{x}^{(n)})$ is parameterised by a task-specific ϕ.

Analogous to multi-head discriminative models, the generative model can be divided into shared and task-specific parts as shown in Figure 1b. This architecture improves performance when data are composed of a common set of structural primitives, such as strokes in handwritten digits, that are selected by high level variables, namely character identities.

Extensions

Bayesian Inference with CNNs

The VCL paper only uses Bayesian inference on the weights of FC networks *i.e.* it estimates posterior distributions over weights $p(\boldsymbol{\theta}|\boldsymbol{D}_{1\to T})$ where the $\boldsymbol{\theta}$s are weight matrices for FC layers. We extend this framework and apply posterior weight inference on the weights of CNN filters [14, 15].

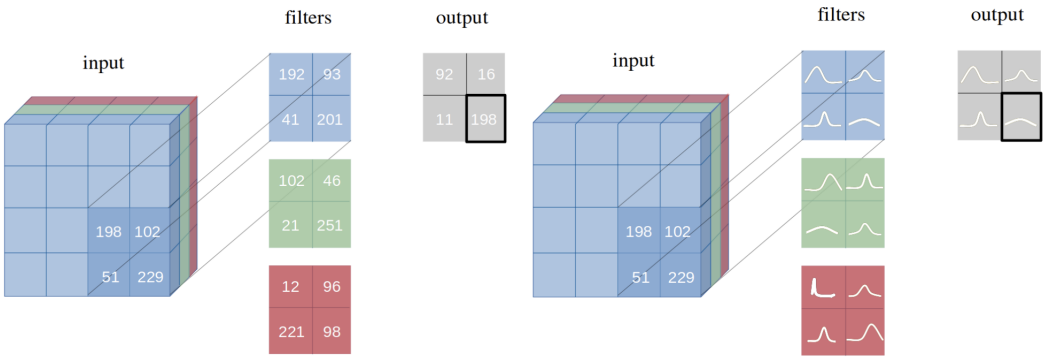


Figure 2: A depiction of point-estimate based CNN kernels (left) vs our method of using distributions over CNN kernels (right). Image taken from [14]

Consider a convolutional layer. It consists of a kernel of size $K \times K$ with I input channels and O output channels. This convolutional layer has a filter weight of dimension $O \times I \times$

$K \times K$. However, these are simply point estimate weights and do not retain uncertainty estimates over their values. We would therefore like to maintain Gaussian distributions over each of these CNN filter weights. To do this, we simply parameterise each kernel weight with a filter mean and filter variance. Hence, we now maintain two $O \times I \times K \times K$ tensors, one for the mean and one for the variance, to capture the parameters of a single convolutional layer. This process is succinctly depicted in Figure 2.

Once we have parameterised our convolutional layer weights to ensure they follow a Bayesian paradigm, we can use the same MFVI approximation used by the VCL paper to do the posterior inference over CNN weights. We consider each filter weight to be sampled from an isotropic Gaussian distribution. With this framework, we retain the efficiency of VCL's architecture as well as improve performance by using CNN layers as strong upstream feature-extraction layers. We preserve the task-specific architectures as specified in the previous section and only use Bayesian convolutional layers as feature-extraction layers.

Local Reparameterisation Trick

For training VCL models, we have to compute an expectation over the log-likelihood of the observed data (Equation 1, 1st term). Since this computation is usually intractable, we use a Monte-Carlo sampling approximation by sampling several point-estimate weights from our weight distribution (as parameterised by μ and σ over our network layers) and propagate our inputs through these "sampled layers". Finally, we compute an average over all these outputs to estimate the aforementioned expectation. This requires backpropagation through a sampling operation: an intractable operation since we require nodes in our neural network computational graph to be deterministic for taking derivatives. To mitigate this, Kingma et. al [16] proposed the reparamaterisation trick (see Figure 3).

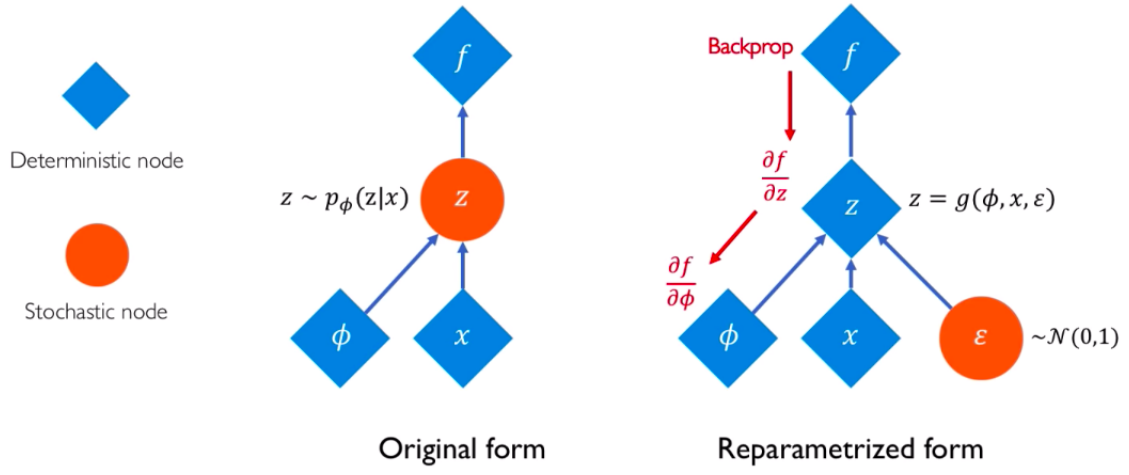


Figure 3: Reparameterisation trick. Image taken from [17].

Using the reparameterisation trick, we decouple the sampling operation into two operations: one sampling a random noise variable and the other using a deterministic variable.

Assume we need to sample a one-dimensional weight $w \sim \mathcal{N}(\mu, \sigma^2)$. We can rewrite the random variable w as

$$w = \mu + \sigma * \varepsilon$$

where

$$\varepsilon \sim \mathcal{N}(0,1)$$

This reroutes all the sampling operations through the noise variable ε allowing us to back-propagate through the deterministic μ and σ variables and helping us optimise them with SGD.

There is one problem with the reparameterisation trick however. We need to sample a vector from a standard spherical isotropic Gaussian distribution that has the same dimension as the weights for the mean and variance. If these weights are very large, then sampling can become prohibitively expensive. In our case, sampling weights for an FC network can become quite large and inefficient leading to slow training and optimisation.

To mitigate this, Kingma et. al [10] proposed the local reparameterisation trick (LRT). The idea is to replace the inefficient sampling of large weight matrices with a smaller sampling of activations[1]. Since the activations are direct functions of the sampled weights, LRT proposes to compute a mean and variance of activations for a particular node. This parameterises a distribution over the activations which we then sample from. This sampling operation is much cheaper since activations are typically much smaller in dimension than FC weights. This operation is shown in the equations below:

Traditional activation:

$$z = w^T x + b$$

Reparameterisation trick:

$$w \sim \mathcal{N}(\mu_w, \sigma_w); b \sim \mathcal{N}(\mu_b, \sigma_b); z = w^T x + b$$

Local reparameterisation trick:

$$z_\mu = \mu_w^T x + \mu_b; z_\sigma = \sigma_w^T x + \sigma_b; z \sim \mathcal{N}(z_\mu, z_\sigma)$$

Variational Generative Replay

The VCL framework uses coresets as a tool for mitigating catastrophic forgetting. By using coresets as an episodic memory enhancement scheme prior to making predictions, VCL ensures that it performs a "pseudo-revision" of the previous tasks and hence retains key information about these previous tasks. The VCL paper implements two specifc coreset methods that both employ a similar strategy of explicitly storing images from previous tasks in a memory buffer to revisit in the future. A brief description of the two coreset methods used follows:

[1]Note that this trick only improves efficiency of sampling and the subsequent training if the activations have smaller dimension than the weights. In the case of CNNs where the activations are comparable in size and at times even larger than the kernel weights and biases, we use normal reparameterisation since LRT will not help improve sampling efficiency.

- *Random-Coreset:* At every task, we randomly sample images from the entire task dataset and add them to our coreset.
- *K-Center-Coreset:* At every task, we use the K-Center algorithm [22] to pick samples to add to our coreset. This method returns samples that are guaranteed to be maximally spread out in the task data space so as to exhaustively capture the task data distribution.

One salient limitation of the coreset methods described above is that we need to store the coreset images in memory across several tasks. For the MNIST case, this memory requirement is not large and hence it is viable to use explicitly-stored coresets for obtaining improved performance. However, this memory requirement grows exponentially as we move into the RGB image regime and increase image sizes. Using coresets can become prohibitively expensive in such cases.

To work around this, several recent works have proposed using pseudo-rehearsal based schemes [18, 19, 20]. These schemes draw inspiration heavily from Complementary Learning Systems theory [23] which posits that the brain's hippocampus can be seen as a generative replay buffer for revisiting past experiences. However, the hippocampus does not explicitly encode the entirety of some past experience rather, it instead stores a compact representation. Similarly, pseudo rehearsal methods aim to train a generative model at the end of each task instead of having to store images of previous tasks explicitly in a memory buffer. This enables systems to then generate task-specific samples from a specific generative model on the fly. Thus, an expensive memory requirement is mitigated because we no longer need to store images from previous tasks in a buffer but can instead use the generative model to produce task-specific samples for us. All the previous task information is now compressed into the parameters of the generative model leading to lesser memory requirement.

Here, we follow the approach taken by Farquhar et. al [19]. We train a GAN [21] at the end of every task to capture a task-specific image distribution. The GAN generator consists of two nearest-neighbour upsampling layers and three convolutional layers with leaky ReLU activations. The GAN discriminator consists of four convolutional layers with leaky ReLU activations. The GAN discriminator is trained to predict an image's task identity as well as whether it is a real or generated image. We store all the previously trained task-GANs to generate image samples for our coreset updates. Hence, at task t, our coreset now consists of samples from all the GANs trained from task 1 to $t-1$. We retain the exact same training protocol followed by VCL except for this coreset sampling procedure.

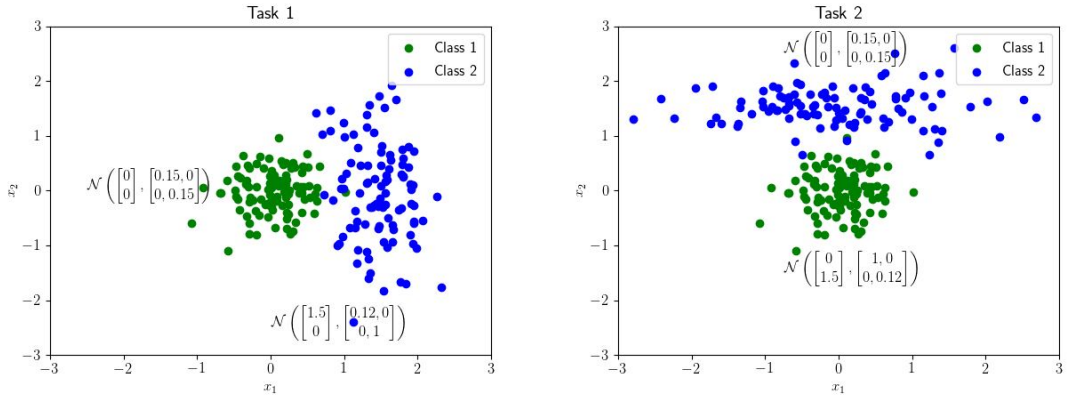
Experiments

In this section several experiments evaluate the performance of VCL in different tasks.

Toy dataset

We apply the VCL framework on a toy classification task[2] and understand some of its properties by analysing the change in decision boundaries as we transition through tasks.

We generate a synthetic dataset comprising of two binary classification tasks. For this, we sample input points from two distinct bivariate isotropic Gaussian distributions within a task. Each individual class within a task consists of 100 datapoints. Subsequently we train on 160 datapoints (80 from each class) and test on the held-out 40 datapoints (20 from each class). Figure 4 shows the generated dataset.



(a) Task 1 (b) Task 2

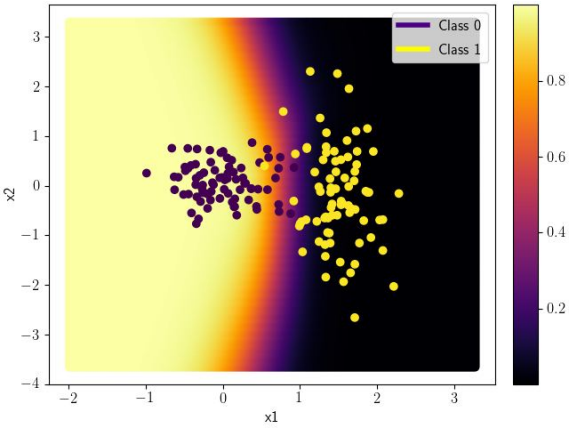
Figure 4: Synthetic dataset generated for the toy experiment

We parameterise our shared network as a single layer FC network with 10 hidden units. We use two task heads, each of which distinguishes between two classes. We train our network for 120 epochs with an Adam optimiser using a learning rate of 0.005. At training and inference time, we average over 10 samples of our weight means and variances for estimating the loss value and performing predictions. We plot the prediction probability contours spanning the 2D grid surrounding our input data points and visualise them in Figure 5.

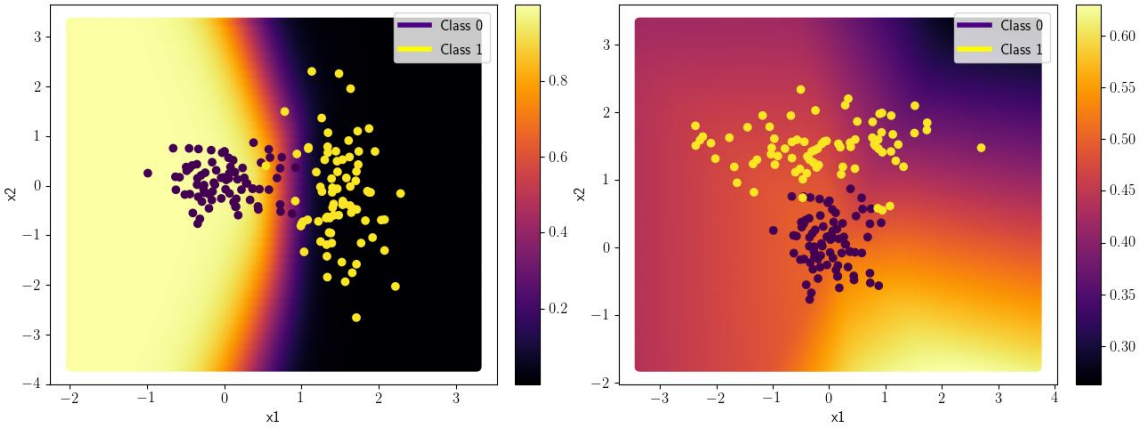
We observe good performance after training and testing solely on task 1 as expected. When we then train on task 2, we clearly see that the model retains its good performance on task 1 despite being retrained only on task 2 data. This exhibits the model's ability to retain information about the task 1 without falling prey to catastrophic forgetting. Further, the model also performs sufficiently well on task 2 as expected since it has been trained only on task 2 data at that stage[3].

[2]Replication of Appendix D in the main paper

[3]The contour of Figure 5c appears murkier than 5a and 5b. This owes to the inherent lower separability between the two classes present in the second task as compared to the first



(a) Train on task 1, Eval on task 2



(b) Train on task 2, Eval on task 1

(c) Train on task 2, Eval on task 2

Figure 5: Visualisation of the probability contours obtained by VCL on the toy experiment

Hence, this empirically validates the efficacy of VCL to adapt to new tasks while ensuring good performance on previously seen tasks. The VCL model therefore demonstrates itself to be a good continual learning model showing an effective balance between stability and plasticity.

Permuted MNIST

The dataset is a MNIST variant where each of the ten tasks is the multi-class classification of a different, fixed, arbitrary permutation of the input pixels. VCL is run using a FC network with two hidden layers of 100 hidden units each with ReLU activations. Three versions of VCL are evaluated: Vanilla VCL (no coreset), VCL with a random coreset, and VCL with a coreset chosen by the K-center algorithm, employing a coreset of 200 data points for each task. Each experiment is repeated three times using different seeds and the results are averaged.

After replicating the experiments of this section in [1], Figure 6 shows the average test set accuracy on all observed tasks comparing the aforementioned methods. The results are almost identical to the original paper [1]: after 10 tasks, VCL achieves 90.5% average

accuracy, but it is outperformed by VCL combined with a coreset. Both VCL with random coreset and VCL with K-center coreset achieve 93% average accuracy. Similarly, using just a K-center coreset performs poorly, resulting in 61% accuracy, and just a random coreset performs badly with a 79% average accuracy. Note that [1] also compares the performance of VCL with other continual learning algorithms: EWC, SI, and LP, however in this report only [1]'s novel contributions, VCL and its coreset variants, are investigated.

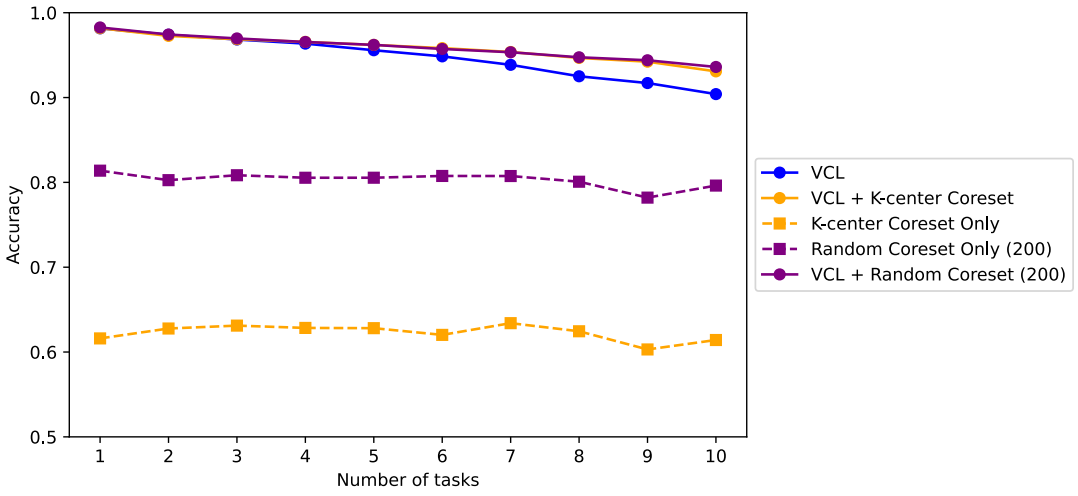


Figure 6: Average test set accuracy on all observed tasks in Permuted MNIST experiment. Note that average accuracy is cumulative up to and including the current task.

The effect of coreset size is also studied. The results of which are illustrated in Figure 7, where the average test set accuracy is plotted for VCL using different coreset sizes. This representation is improved in Figure 8, where the effect of coreset size is better visualized.

Using just coresets tends to perform poorly, above all if the coreset size is low, although if the coreset size is increased considerably (to 5000) the coreset will be fully representative of the task and achieve good performance (94% accuracy).

However, if VCL is used with a coreset, it is shown that the size of the coreset does not need to be as large to get similar performance. After 10 tasks, VCL with a coreset of size 1000 is enough to achieve 95% average accuracy, and the best performing model is VCL with a coreset size of 5000 with a resulting average accuracy of 95.7%, similar to the 95.5% figure found by [1].

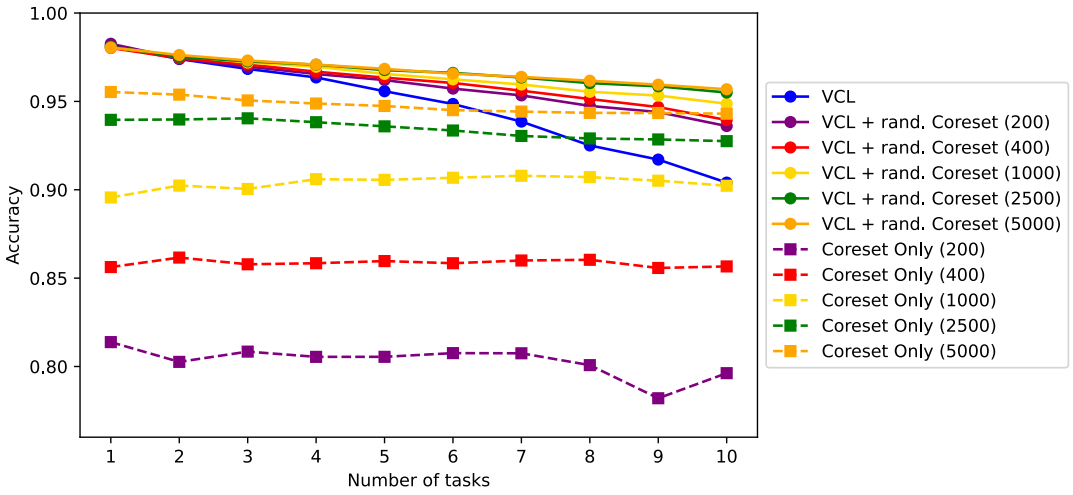


Figure 7: Comparison of the effect of coreset sizes in Permuted MNIST experiment. Note that average accuracy is cumulative up to and including the current task.

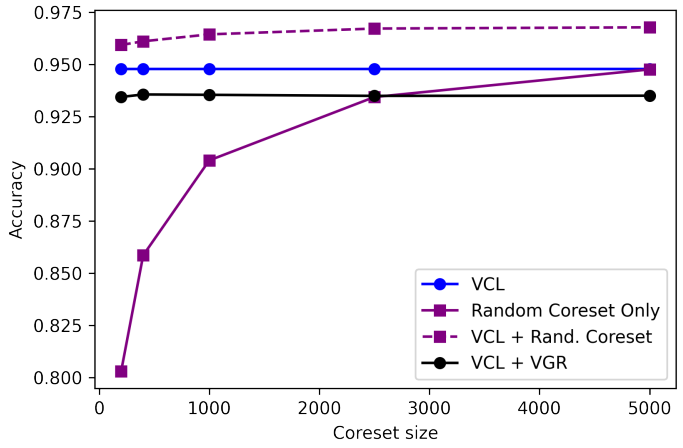


Figure 8: Comparison of the effect of coreset sizes in Permuted MNIST experiment.

Split MNIST

Split MNIST is a variant of the MNIST dataset built to assess continual learning tasks. Five binary classification tasks are considered sequentially: distinguish between 0 and 1, 2/3, 4/5, 6/7 and 8/9. Three versions of VCL are investigated: Vanilla VCL, VCL with a random coreset and VCL with a coreset selected by the K-center method. All variants use a FC multi-head network with two hidden layers of 256 units each and ReLU activations.

The experiment is conducted using three different seeds and the results are averaged. These results are shown in Fig. 9, where the average test set accuracy for each individual task is plotted as well as the cumulative average accuracy over all tasks (the rightmost plot has been replicated in Fig. 10 for clarity) . First, VCL can be seen to be satisfying

its purpose as a continual learning approach as all model variants appear to perform well on past tasks after training on new task. Moreover, systems using just coresets do not appear to be effective continual learning approaches as they perform worse as the number of tasks increases. Vanilla VCL achieves a satisfactory 97.5% average accuracy, but is further improved by coreset variants, obtaining an overall accuracy over 98% and match the original paper's results.

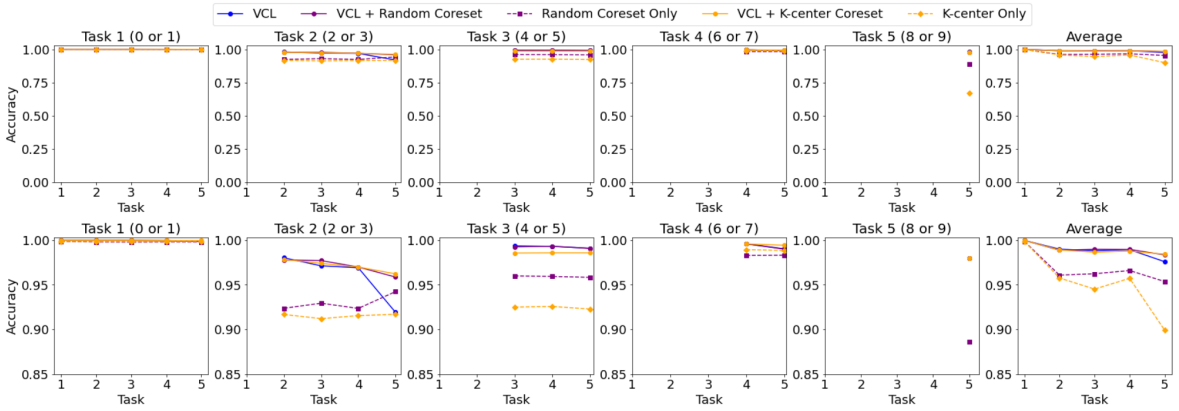


Figure 9: Average test set accuracy on all observed tasks in Split MNIST experiment. The last column shows the average accuracy over all tasks. The bottom row is a zoomed version of the top row.

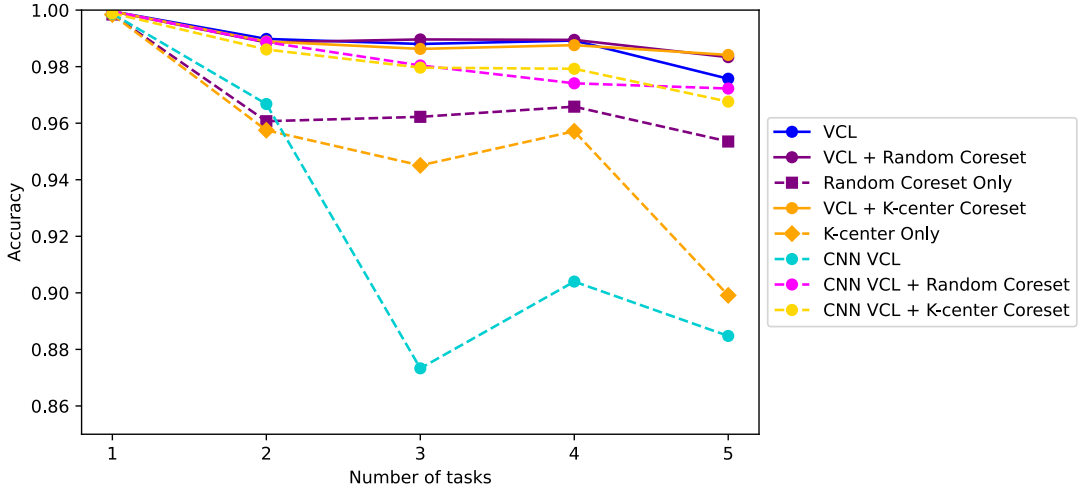


Figure 10: Average test set accuracy on all observed tasks in Split MNIST experiment including results with CNNs.

Model	Number of parameters	Accuracy
VCL	268800	97.57%
VCL + Coreset		98.41%
CNN VCL	9768	88.48%
CNN VCL + Coreset		96.76%

Table 1: Accuracy-complexity trade-off between CNNs and FC networks

CNN

In addition, all VCL variants were explored using a convolutional neural network as well, the architecture of which consists of two convolutional layers followed by two fully connected hidden layers of 48 units all with ReLU activations. The first convolution consists of a $4 \times 1 \times 3 \times 3$ filter with a stride of one, and the second convolution consists of a $1 \times 4 \times 3 \times 3$ filter with a stride of two. Notably, this CNN architecture has significantly fewer parameters than the aforementioned fully connected architecture, as can be observed in Table 1.

Plotted in Figure 10 is the same experimental setup described to generate the plots in Figure 9, however with the inclusion of three CNN VCL variants: Vanilla CNN VCL, CNN VCL with a Random Coreset, and CNN VCL with a K-centered coreset. Notably, Vanilla CNN VCL achieves worse performance than FC using only coresets, while CNN VCL with either coreset method improves performance to the same order of magnitude as Vanilla FC VCL.

Notable about these results is two things. At a basic level VCL methods can be applied to CNN architectures, implying that VCL works for a variety of architectures which may be implemented given particular structure in the underlying task data. Secondly, a coreset is more useful in improving the performance of a less complex and lower parameter model. Exemplifying this fact is Table 1 which demonstrates that the CNN architecture employed has $\frac{9768}{268800} \approx 3\%$ of the parameters of the FC network, but with a coreset is able to achieve roughly the same accuracy scale as its FC variant. Although analysis of this accuracy-complexity trade-off is unsophisticated, it serves to communicate both the flexibility of VCL and the utility of a coreset.

Variational Generative Replay

For both Split MNIST and Permuted MNIST the following architecture and training approach was taken. A GAN was trained for 200 epochs on each MNIST digit using the same data which is used for the main training process. We use 6000 generated digits per class, sampled fresh for each task, and initialize network weights using the previous task. We use a batch size of 256 times the number of seen tasks, ensuring that the number of batches is held constant. The GAN is trained with an Adam optimizer with learning rate $\alpha = 2 * 10^{-4}$ and $\beta_1 = 0.5$.

Plotted in Figure 11 is the same experimental setup described to generate the plot in Figure 9, but includes the use of GANs to generate equivalent images in a coreset. Notably, VCL+VGR is the best performing system. Likely, this is because in Split MNIST, images are semantically coherent and our convolutional GAN is able to model task distributions more effectively than a fixed buffer of selected images. Thus VGR serves as the most effective coreset enhancement scheme for this task.

However, as plotted in Figure 12, which is the same experimental setup used to generate the plot in Figure 7, the performance of a VCL+VGR system does not improve over Vanilla VCL much. This is likely due to the notion that coherent spatial semantics are lost in Permuted MNIST, and thus a convolutional GAN is not able to capture distinct task

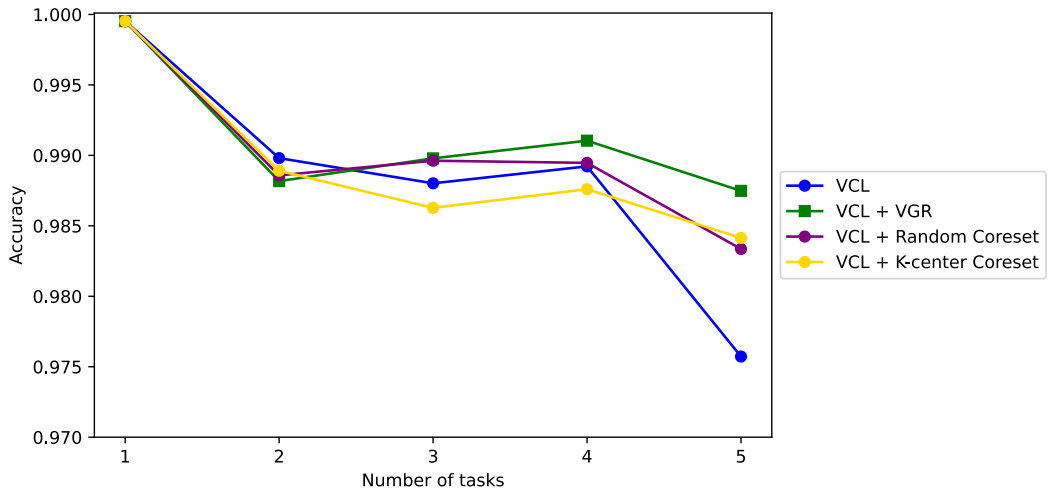


Figure 11: Comparison of the effect of VGR when using VCL with coreset in Split MNIST experiment.

distributions effectively. Thus, generated images from such a GAN do not boost performance as real samples maintained in an explicit coreset do. Notably, accuracy does not degrade in spite of this fact, with the VCL+VGR system achieving comparable performance to Vanilla VCL after the task battery. The lack of accuracy degradation suggests that either VCL is robust to an inadequate coreset.

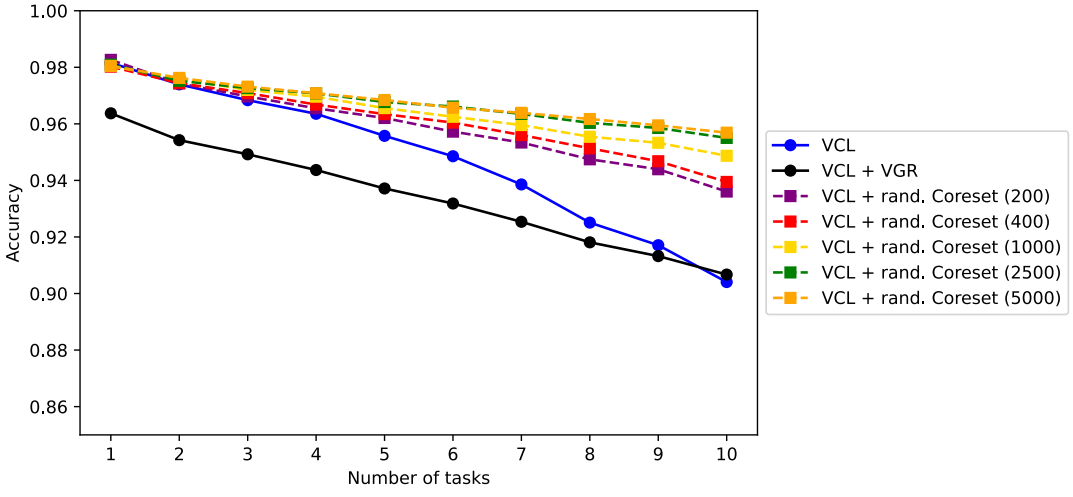


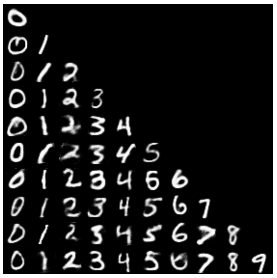
Figure 12: Comparison of the effect of VGR when using VCL with coreset in Permuted MNIST experiment.

Generative Modelling

In this section, we describe the application of the VCL framework for the task of generating images of digits (MNIST) and letters (NotMNIST). The task of MNIST digit generation is to generate samples of digit 0 in task 1, digit 1 in task 2 etc. Similarly, for NotMNIST letter generation each subsequent task generates the letters A to J. Each dataset thus has 10 different generative tasks. We apply the model architecture as shown in Figure 1b. Our shared network consists of an FC network with 500 hidden units using ReLU activations. Our task-specific heads also consist of an FC network with 500 hidden units using ReLU activations. We use a latent variable of size 50 and use a sample size of 10 for Monte Carlo samples to compute expected log-likelihood.

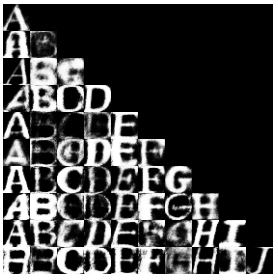
In Figure 14, we show the MNIST and NotMNIST images generated by VCL and a naive online learning method. The naive method is a model using the same architecture as VCL but trained using only the VAE objective. At every new task, the naive method simply trains on that particular task data incrementally. The images in the i_{th} row are generated by the model after training all tasks up to and including task i. The images in the j_{th} column are generations from task j.

(a) *Naive*



(b) *VCL*

(c) *Naive*



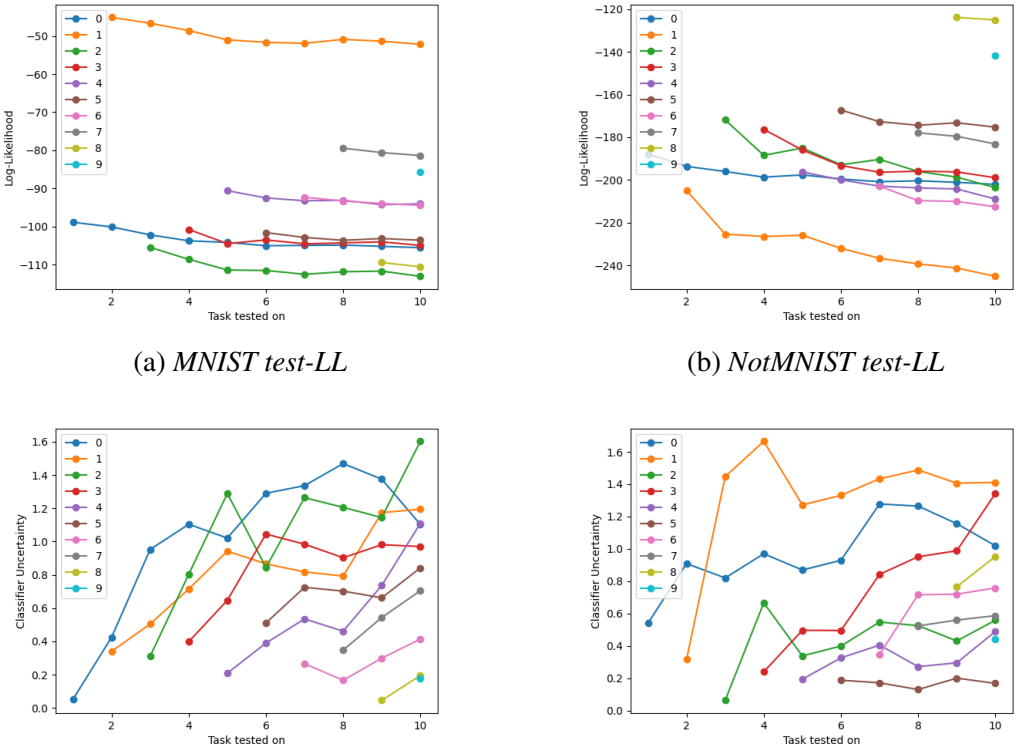
(d) *VCL*

Figure 13: Generated image samples from VCL and the naive method. We observe that even after training on the 10th task, the generation quality of VCL on the earlier tasks is preserved. The naive model however catastrophically forgets old tasks.

We quantitatively evaluate the generation quality of the VCL generated task-specific images using two metrics:

1. *Estimated test-LL*: We compute the estimated log-likelihood of the test set samples using importance sampling. We use 5000 importance samples for computing the expected log-likelihood.

2. *Classifier Uncertainty (CU)*: Firstly, we train a single-layer FC network to classify MNIST digits/NotMNIST characters. We then produce predictions using this classifier on the generated samples and compute the KL divergence between the classifier output softmax probabilities with the one-hot ground truth class labels. Ideally, we would like the classifier to be confident about its predictions on the generated images resulting in an optimal KL divergence of 0.



(a) *MNIST test-LL* (b) *NotMNIST test-LL*

(c) *MNIST CU* (d) *NotMNIST CU*

Figure 14: The first row shows the test log-likelihood values on both datasets. The second row depicts the classifier uncertainty on the datasets. Each of the lines are produced by a model trained until the particular task it is labelled as. For example, the red lines in the plots are produced by a model trained until task 4.

From Figure 14, we observe nearly-flat lines of the log-likelihoods with very slight negative gradients. These imply that the average test log-likelihoods over all tasks do not drastically decrease over additional tasks. We also observe that CU are quite low for all tasks, indicating that VCL does not suffer from catastrophic forgetting. This is because VCL can retain enough task information from the earlier tasks to generate plausible samples such that a well-trained classifier predicts the right classes on the generated samples with high confidence. These results strongly corroborate the effectiveness of VCL as a continual generative learning model.

Conclusion

The application of Bayesian Inference to Continual Learning in the form of VCL is a productive endeavor. In particular, such application boasts improved performance in discriminative and generative continual learning tasks as demonstrated by improved performance over representative continual learning benchmarks such as Split MNIST, Permuted MNIST, and MNIST/NotMNIST glyph generation. Furthermore, the application of episodic memory enhancement schemes has been shown to be midly productive at improving the stability and accuracy of dense models, and very productive at improving the stability of relatively parameter-sparse models such as CNNs. Generative replay schemes as a means of episodic memory enhancements were also proven to be effective if the generative model is suitable for capturing task-specific structure.

If given more time to pursue the study of VCL, the following areas would be of great research interest. Firstly, it would be enlightening to apply our aforementioned CNN VCL to a more complex task, such as CIFAR. Additionally, all above experiments analyzed a fixed incoming order of novel tasks, but it would be informative to understand how system performance responds to permutations in incoming task order and perhaps construct adversarial task orderings. Lastly, it would be useful to refine VGR for non-spatially coherent data in Permuted MNIST, perhaps a fully-connected architecture, to investigate if VGR, given a suitable architecture, is superior to an explicitly maintained coreset for the full gamut of continual learning benchmarks.

References

[1] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, Richard E. Turner. Variational Continual Learning. *International Conference on Learning Representations*. 2018.

[2] Noel Loo, Siddharth Swaroop, Richard E Turner. Generalized Variational Continual Learning. *International Conference on Learning Representations*. 2020.

[3] James Kirkpatrick et. al. Overcoming Catastrophic Forgetting in Neural Networks. 2016.

[4] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems*. 2017.

[5] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. 2018.

[6] Huzaifah, M. and Wyse, L.. Deep generative models for musical audio synthesis. 2020.

[7] Ruihan Yang et. al.. Insights from Generative Modeling for Neural Video Compression. 2021.

[8] Bart Bakker and Tom Heskes. Task clustering and gating for Bayesian multitask learning. *Journal of Machine Learning Research*. 2003.

[9] Tim Salimans and David A. Knowles. Fixed-form variational posterior approximation through stochastic linear regression. *Bayesian Analysis*. 2013.

[10] Diederik P. Kingma, Tim Salimans, Max Welling. Variational Dropout and the Local Reparameterization Trick. *Advances in Neural Information Processing Systems*. 2015

[11] S. Grossberg. Studies of mind and brain : neural principles of learning, perception, development, cognition, and motor control. *Boston studies in the philosophy of science*. 1982.

[12] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *TPAMI*. 2020.

[13] Sebastian Farquhar, Yarin Gal. Towards Robust Evaluations of Continual Learning. *Workshop on Lifelong Learning: A Reinforcement Learning Approach at ICML*. 2018.

[14] Kumar Shridhar, Felix Laumann and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. *ArXiv*. 2019

[15] Kumar Shridhar, Felix Laumann, Adrian Llopart Maurin and Martin Olsen, Marcus Liwicki. Bayesian convolutional neural networks with variational inference. *ArXiv*. 2018

[16] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*. 2014

[17] Sayak Paul. "Reparameterization" trick in Variational Autoencoders. *Medium*. 2020

[18] Hanul Shin, Jung Kwon Lee, Jaehong Kim and Jiwon Kim. Continual Learning with Deep Generative Replay. *Advances in Neural Information Processing Systems*. 2017

[19] Sebastian Farquhar and Yarin Gal. A Unifying Bayesian View of Continual Learning. *NeurIPS 2018 workshop on Bayesian Deep Learning*. 2019

[20] Jary Pomponi, Simone Scardapane and Aurelio Uncini. Pseudo-Rehearsal for Continual Learning with Normalizing Flows. *LifelongML workshop, ICML*. 2020

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio. Generative Adversarial Networks, *Advances in Neural Information Processing Systems*. 2014

[22] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, *Theoretical Computer Science*. 1985

[23] Randall C O'Reilly, Rajan Bhattacharyya, Michael D Howard and Nicholas Ketz. Complementary learning systems. *Cognitive Science*. 2014