Deep Class-Conditional Gaussians for Continual Learning

Thomas L. Lee
School of Informatics
University of Edinburgh
T.L.Lee-1@sms.ed.ac.uk

Amos Storkey
School of Informatics
University of Edinburgh
a.storkey@ed.ac.uk

Abstract

The current state of the art for continual learning with frozen, pre-trained embedding networks are simple probabilistic models defined over the embedding space, for example class conditional Gaussians. As yet, in the task-incremental online setting, it has been an open question how to extend these methods to when the embedding function has to be learned from scratch. In this paper, we propose DeepCCG, an empirical Bayesian method which learns online both a class conditional Gaussian model and an embedding function. The learning process can be interpreted as using a variant of experience replay, known to be effective in continual learning. As part of our framework, we decide which examples to store by selecting the subset that minimises the KL divergence between the true posterior and the posterior induced by the subset. We demonstrate performance task-incremental online settings, including those with overlapping tasks. Our method outperforms all other methods, including several other replay-based methods.

1 Introduction

Real world use of deep learning methods can often necessitate dynamic updating of solutions on non-stationary data streams [10, 3]. This is one of the main problems studied in continual learning and as a result, continual learning is of increasing interest to the machine learning community, with many proposed approaches [28] and settings [17, 3, 7]. The setting considered here is *task-incremental online learning* where a data stream is split into a sequential set of *tasks* and methods are given information about what the current task is [35, 30]. Each task is encapsulated by a representative dataset (considered to be sampled i.i.d. from a task distribution) which is given to a method batch by batch. Different tasks will generally be associated with different distributions, as well as different target problems, which are summarised in a task objective function. We focus on classification [17], where the classes being considered vary between tasks. The overall objective is, after seeing all of the tasks, to perform well on all of them (given memory constraints).

Currently one of the best ways to approach continual learning is to use a frozen pretrained embedding function and define a simple probabilistic model on top to classify the data [27, 14]. However, in some real-world settings it is necessary to learn the embedding function online, for example it might be impossible to pretrain the embedding function due to a lack of data or due to distribution shifts a frozen pretrained encoder may become outdated and so perform badly [27]. Therefore, it is an interesting question to see how these methods can be adapted to settings when the embedding function must be continually learnt. We explore this question, presenting the method DeepCCG, where we use a class conditional Gaussian model on top of a neural network embedding function. A key part of our method is that the class conditional Gaussian model can be re-estimated on a memory buffer as the embedding function changes, reducing forgetting. Additionally, another key part of DeepCCG is a method for *selecting* what samples to store in memory, where we select examples that preserve the means of the per-class clusters formed by the embedded data.

In our experiments we look at two specific settings, the commonly used disjoint tasks setting, where each task contains different classes to any other [7] and an underexplored shifting window setting, where there is an overlap between what classes are in each task. The reason we look at a setting with an overlap between tasks is to explore a methods ability for positive transfer, which is more rewarded in this setting as there is greater shared information [4]. The results of our experiments show that DeepCCG performed the best out of all methods tested, gaining an average performance improvement of 2.145% in terms of mean average accuracy, showing the potential of our approach.

2 Task-incremental setting

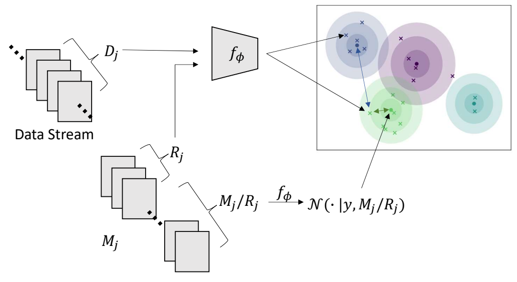

Figure 1: Diagram of DeepCCG's training routine. At time j the learner is given a sample of data D_j and has a memory set of stored datapoints M_j. The memory is split into the replay data R_j and the rest M_j/R_j. Learning happens by taking a gradient step on the parameters of the embedding function ϕ using a conditional marginal likelihood function over D_j and R_j, where M_j/R_j is used to induce a posterior over the means of the per-class Gaussians and so define the conditional marginal likelihood function used.

Task-incremental online continual learning is a setting where the learner is given a sequence of batches of data one by one, along with a task identifier for each batch. Different batches may be drawn from different tasks according to some unknown non-stationary process. The goal is for the learner to be able to perform well on all tasks at the end of training [17, 30]. Specifically, we consider classification tasks. Let X denote a shared data space, and C denotes the set of all classes being considered. Let $t \in T$ denote a particular task, which generates data $S_t = \{(x_i, y_i)|i = 1, 2, \ldots\}$, containing data instances $x_i \in X$ and labels $y_i \in C_t$, where $C_t \subset C$ are the subset of classes relevant for this task. The learner is provided with a temporal sequence (of length N) of data batches and task identifiers $((D_j, t_{D_j})|j = 1, 2, \ldots, N)$, where j indexes batches and t_{D_j} denotes the task identifier for batch D_j. At each time step j, the batch D_j is a sample from a single task and the *task identifier* t_{D_j} determines this task, indicating the classes which are relevant for that task. During training, the learner receives each batch in sequential order; after training on a batch the learner must discard all the data it does not store in its fixed-size memory buffer. At test time, the learnt model is given a previously-unseen test set from all the classes seen during training. This setting is commonly used in the continual learning literature [5, 30].

3 Deep class conditional Gaussians

We propose an empirical Bayesian method to contiually learn both the embedding function and a class conditional Gaussian model, which we name DeepCCG (we also provide a more general method in Appendix B). We use a neural network for the embedding function f_ϕ and have a fixed size memory buffer storing previously seen examples and their task identifiers. We define a class conditional Gaussian model in the embedding space Z and use the inverse model of that class conditional Gaussian to define $p(y|z, t, \theta)$, where the parameters $\theta = \{\mu_y|y \in C\}$, that is the parameters of this model are the means of the per-class Gaussians. The reason we chose to use a class conditional Gaussian model is due to it having state-of-the art performance when using a frozen pretrained embedding function [27, 14] and that by using its associated conjugate prior, the posterior distribution for θ is tractable and easy to compute. Likewise, the marginal likelihood given by the inverse model is fully tractable. More formally, for each task t we define

$$y|t \sim \mathrm{Cat}(C_t, (1/|C_t|)\mathbf{1}),\ z|y \sim \mathcal{N}(\mu_y, \mathbf{I}),\ \mu_y \sim \mathcal{N}(\mathbf{0}, \mathbf{V}_0) \tag{1}$$

In this paper, we choose $\mathbf{V}_0 = a\mathbf{I}$, where in practice we take $a \to \infty$, and $\mathrm{Cat}(C_t, (1/|C_t|)\mathbf{1})$ is the uniform categorical distribution over the classes of task t.

Learning the embedding To update the embedding function f_ϕ, in response to the arrival of batch D_j, with task identifier t_{D_j}, and having received the memory buffer M_j, we take the following steps (shown in Figure 1). First, we select a set $R_j \subset M_j$, of size r, from the memory buffer. Then we condition on the rest of the examples, M_j/R_j, to obtain the posterior predictive distributions $p(y|f_\phi(x), t_{(x,y)}, M_j/R_j)$ for each sample $(x, y) \in D_j \cup R_j$, where $t_{(x,y)}$ is the task identifier for the sample. The posterior predictive distributions can be shown to be (see Appendix D for the derivation):

$$p(y|z = f_\phi(x), t_{(x,y)}, M_j/R_j) = \frac{\mathcal{N}\left(z|\overline{M^Z_{j,y}/R^Z_j}, (1 + \frac{1}{m-r})\mathbf{I}\right)}{\sum_{c \in C_{t_{(x,y)}}} \mathcal{N}\left(z|\overline{M^Z_{j,c}/R^Z_j}, (1 + \frac{1}{m-r})\mathbf{I}\right)} \tag{2}$$

where $M^Z_{j,y} = \{f_\phi(x)|(x, y') \in M_j \wedge y' = y\}$ are the embeddings of samples in the memory buffer with class y. R^Z_j is defined likewise and we use the notation $\overline{S}$ to denote the mean of the elements of a set S. As in meta-learning [12, 16], the posterior predictive distributions are used as a fixed likelihood term for each individual unseen data item, leading to a log-marginal likelihood as the objective in the gradient decent update used:

$$\phi_{j+1} = \phi_j + \eta \nabla_\phi \sum_{(x,y) \in D_j \cup R_j} \log(p(y|z = f_\phi(x), t_{(x,y)}, M_j/R_j)). \tag{3}$$

where $t_{(x,y)}$ is the task identifier associated with sample (x, y). This update method can be seen as a variant experience repay, a very effective continual learning approach [35, 39, 25].

Sample selection The second key component to DeepCCG is the sample selection for the memory buffer. Because all the information available to inform the value of the parameters so far is encapsulated in the full posterior over the parameters, we target choosing a set of memory samples that best recreate that posterior, preventing as much parameter information loss as possible. Hence, to perform sample selection we minimise the KL divergence between two posterior distributions: the posterior over parameters induced by the new memory being optimized, and the posterior induced by the current batch and the old memory. We keep the number of samples in memory for each class balanced, therefore minimising the KL divergence is equivalent to minimising the per-class KL divergence which reduces to the squared Euclidean distance between the means of a class's embedded data. Formally,

$$M_{j+1,y} = \underset{M'_y \subseteq (D_{j,y} \times \{t_{D_j}\}) \cup M_{j,y}, |M'|=m}{\arg\min} (\|\overline{D^Z_{j,y} \cup M^Z_{j,y}} - \overline{M'^Z_y}\|_2^2), \tag{4}$$

where $M_{j+1,y}$ is the new memory to be selected for class y, $M_{j,y}$ is the is the set of samples of class y in memory when receiving the current batch of data and $D_{j,y}$ is the set of samples of class y in the current batch.

Performing the minimization in Eq. 4 is computationally hard, so we utilise a relaxation of the problem using *lasso* [13], whereby our method selects the new memory by assigning to each embedded input z_i a zero-to-one weight β_i and performing gradient decent on the loss

$$\mathcal{L}(\beta; D_{j,y}, M_{j,y}) = \|\overline{D^Z_{j,y} \cup M^Z_{j,y}} - \frac{1}{\|\beta\|_1} \sum_{\substack{i|(x_i,y_i) \in \\ D_{j,y} \cup M_{j,y}}} \beta_i z_i\|_2^2 + \lambda \|\beta\|_1. \tag{5}$$

Then, after termination, our method selects the m samples with the largest weights to be the samples stored in memory, where they are stored along with their task identifiers.

4 Experiments

Benchmarks In our experiments we look at two different settings: disjoint tasks and a shifting window setting. Furthermore, in these settings we utilize three datasets CIFAR-100 [21], MiniImageNet [36] and CIFAR-10 [21]. For disjoint tasks, we split the datasets using the classes of the samples, splitting evenly the classes across a certain number of tasks and assigning all samples with that class to that task [7, 5]. We split CIFAR-10 into 5 tasks where there are 2 classes per task and for CIFAR-100 and MiniImageNet we split the dataset into 20 tasks with 5 classes per task. While in the shifting window setting we split the datasets up into tasks by fixing an ordering of the classes $c_1, \ldots, c_k$ and construct the tth task by selecting a set of samples from classes $c_t, \ldots, c_{t+l}$, where l is the window length, there is no overlap between any two of the task datasets and each task has the same number of samples per-class. For CIFAR-10 we use a window length of 2 and for CIFAR-100 and MiniImageNet we use a window length of 5. Additionally, for all experiments we train with 500 samples per-class. We also perform an ablation of our method in Appendix F.

Methods We compare DeepCCG to representative methods of the main paradigms of continual learning: EWC [18, 20], PackNet [24], ER-Reservoir [1], EntropySS [38] and GSS [2].

We also compare against two baselines: SGD which is when learning is performed using SGD with no modification and Multi-Task which is a rough upper bound and is when we learn the base neural network offline, training on the same number of samples per batch as the replay methods tested, which is 20 (10 new samples and 10 replayed), with the same number of batches, to more fairly upper bound their performance. All methods use the same underlying embedding network for all experiments which is a modified ResNet18 with six times fewer filters across all layers and Instance Normalisation [34] instead of Batch Normalisation layers [19], which is similar to other work [25, 9]. Additionally, all methods use a batch size of 10, with all replay methods having the same sample replay size of 10 and a memory size of 10 samples per-class for all experiments.

Table 1: Results of experiments on CIFAR-10, where we report mean average accuracy with their standard errors across three independent runs.

Method	**Shifting Window**	**Disjoint Tasks**
EWC	58.68 ± 1.883	63.33 ± 0.871
PackNet	69.29 ± 2.272	66.97 ± 1.471
ER-reservoir	70.44 ± 0.806	66.71 ± 0.896
A-GEM	57.63 ± 2.163	57.28 ± 2.606
EntropySS	67.93 ± 0.625	64.96 ± 0.911
GSS	71.55 ± 1.45	67.30 ± 1.268
DeepCCG (ours)	$\mathbf{74.65 \pm 2.00}$	$\mathbf{69.29 \pm 0.889}$
SGD	63.21 ± 2.088	63.58 ± 0.306
Multi-Task (**UB**)	96.20 ± 0.690	73.37 ± 1.820

Table 2: Results of experiments on CIFAR-100 and MiniImageNet, where we report mean average accuracy with their standard errors across three independent runs.

	CIFAR-100		**MiniImageNet**	
Method	**Shifting Window**	**Disjoint Tasks**	**Shifting Window**	**Disjoint Tasks**
EWC	36.65 ± 1.073	42.39 ± 0.787	32.42 ± 1.129	29.57 ± 0.689
PackNet	40.21 ± 0.961	50.28 ± 0.578	34.15 ± 1.280	37.86 ± 1.710
ER-reservoir	54.05 ± 0.626	58.31 ± 1.084	41.04 ± 1.539	40.29 ± 1.077
A-GEM	29.01 ± 1.449	39.00 ± 0.745	26.97 ± 1.264	30.08 ± 1.855
EntropySS	51.80 ± 0.700	56.75 ± 0.806	40.03 ± 0.609	41.12 ± 0.513
GSS	48.20 ± 0.332	49.92 ± 0.496	37.91 ± 0.493	38.77 ± 0.980
DeepCCG (ours)	$\mathbf{56.62 \pm 0.288}$	$\mathbf{60.46 \pm 0.243}$	$\mathbf{42.18 \pm 0.449}$	$\mathbf{43.04 \pm 0.638}$
SGD	35.63 ± 1.271	42.50 ± 1.403	33.23 ± 0.674	31.61 ± 0.696
Multi-Task (**UB**)	90.42 ± 1.866	59.76 ± 0.487	89.81 ± 0.250	48.46 ± 1.211

Results We see from Table 2 and Table 1 that DeepCCG performs the best out of all the methods compared. For the shifting window setting, on CIFAR-100, MiniImageNet and CIFAR-10, DeepCCG gets mean average accuracies of 56.62%, 42.18% and 74.13%, respectively, which is 2.57%, 1.14% and 3.1% better than any other method. For the disjoint tasks setting DeepCCG also performs well, where for CIFAR-100, MiniImageNet and CIFAR-10, DeepCCG achieves a mean average accuracy of 60.46%, 43.04% and 69.29% which is 2.15%, 1.92%, 1.99% better than any other method, respectively. It is important to note however that in the shifting window setting DeepCCG is still far below the multi-task performance of 90.42%, 89.81% and 96.20% for CIFAR-100, MiniImageNet and CIFAR-10, respectively. This shows that in settings with more overlap between tasks there is still much work to be done on continual learning methods.

5 Conclusions

In this work we have demonstrated that using a replay based empirical Bayesian procedure for continual learning is a promising direction and can expand the use of simple probabilistic models from when the embedding function is frozen to when it is learnt online. We have proposed DeepCCG, a method to learn continually both a class conditional Gaussian model and the embedding function it is defined on top of. As part of the method we proposed a way of selecting samples to store in memory by approximately selecting the best subset that minimises the KL divergence between the posterior induced by all the currently accessible data and the posterior induced by the subset. The results show that DeepCCG performs well for task-incremental online learning, outperforming all other methods in our experiments. For future work, potential directions would be to look at adapting this approach to class-incremental learning and in settings with imbalanced and/or noisy data [33, 40, 2].

References

[1] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online Continual Learning with Maximal Interfered Retrieval. In *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*, pages 11849–11860, 2019.

[2] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient Based Sample Selection for Online Continual Learning. In *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*, pages 11816–11825, 2019.

[3] Antreas Antoniou, Massimiliano Patacchiola, Mateusz Ochal, and Amos Storkey. Defining Benchmarks for Continual Few-shot Learning. *arXiv preprint arXiv:2004.11967*, 2020.

[4] Jihwan Bang, Heesu Kim, YoungJoon Yoo, Jung-Woo Ha, and Jonghyun Choi. Rainbow Memory: Continual Learning with a Memory of Diverse Samples. In *Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8218–8227, 2021.

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient Lifelong Learning with A-GEM. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[6] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On Tiny Episodic Memories in Continual Learning. *arXiv preprint arXiv:1902.10486*, 2019.

[7] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A Continual Learning Survey: Defying Forgetting in Classification Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. doi: 10.1109/TPAMI.2021.3057446.

[8] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided Continual Learning with Bayesian Neural Networks. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[9] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal Gradient Descent for Continual Learning. *arXiv preprint arXiv:1910.07104*, 2019.

[10] Sebastian Farquhar and Yarin Gal. Towards Robust Evaluations of Continual Learning. *arXiv preprint arXiv:1805.09733*, 2018.

[11] Sebastian Farquhar and Yarin Gal. A Unifying Bayesian View of Continual Learning. *arXiv preprint arXiv:1902.06494*, 2019.

[12] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional Neural Processes. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713, 2018.

[13] Trevor Hastie, Robert Tibshirani, and Jerome H Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[14] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 220–221, 2020.

[15] Christian Henning, Maria Cervera, Francesco D' Angelo, Johannes von Oswald, Regina Traber, Benjamin Ehret, Seijin Kobayashi, Benjamin F. Grewe, and João Sacramento. Posterior Meta-Replay for Continual Learning. In *Proceedings of the 35th conference on the Advances in Neural Information Processing Systems*, pages 14135–14149, 2021.

[16] Timothy M Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J Storkey. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. doi: 10.1109/TPAMI.2021.3079209.

[17] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. In *Proceedings of the 3rd Continual Learning Workshop, at the 32nd Conference on the Advances in Neural Information Processing Systems*, 2018.

[18] Ferenc Huszár. Note on the Quadratic Penalties in Elastic Weight Consolidation. *Proceedings of the National Academy of Sciences*, 115(11):E2496–E2497, 2018. doi: 10.1073/pnas.1717042115.

[19] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, 2015.

[20] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[21] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *Preprint*, 2009.

[22] Richard Kurle, Botond Cseke, Alexej Klushyn, Patrick van der Smagt, and Stephan Günnemann. Continual Learning with Bayesian Neural Networks for Non-Stationary Data. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[23] Sanae Lotfi, Pavel Izmailov, Gregory Benton, Micah Goldblum, and Andrew Gordon Wilson. Bayesian Model Selection, the Marginal Likelihood, and Generalization. *arXiv preprint arXiv:2202.11678*, 2022.

[24] Arun Mallya and Svetlana Lazebnik. PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[25] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the Role of Training Regimes in Continual Learning. In *Proceedings of the 33rd conference on the Advances in Neural Information Processing Systems*, pages 7308–7320, 2020.

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

[27] Oleksiy Ostapenko, Timothee Lesort, Pau Rodríguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. Foundational models for continual learning: An empirical study of latent replay. *arXiv preprint arXiv:2205.00329*, 2022.

[28] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A review. *Neural Networks*, 113:54 – 71, 2019. ISSN 0893-6080.

[29] Francesco Pelosin. Simpler is better: off-the-shelf continual learning through pretrained backbones. *arXiv preprint arXiv:2205.01586*, 2022.

[30] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. GDumb: A Simple Approach that Questions our Progress in Continual Learning. In *Procceding of the 16th European Conference on Computer Vision*, pages 524–540, 2020.

[31] Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual Unsupervised Representation Learning. In *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*, 2019.

[32] Dongsub Shim, Zheda Mai, Jihwan Jeong, Scott Sanner, Hyunwoo Kim, and Jongseong Jang. Online Class-Incremental Continual Learning with Adversarial Shapley Value. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

[33] Shengyang Sun, Daniele Calandriello, Huiyi Hu, Ang Li, and Michalis Titsias. Information-Theoretic Online Memory Selection for Continual Learning. In *Proceedings of the 10th International Conference on Learning Representations*, 2022.

[34] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: the Missing Ingredient for Fast Stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[35] Gido M van de Ven and Andreas S Tolias. Three Scenarios for Continual Learning. *arXiv preprint arXiv:1904.07734*, 2019.

[36] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, and Daan Wierstra. Matching Networks for One Shot Learning. In *Proceedings of the 30th Conference on the Advances in neural information processing systems*, 2016.

[37] Jeffrey S Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[38] Felix Wiewel and Bin Yang. Entropy-Based Sample Selection for Online Continual Learning. In *Proceedings of the 28th European Signal Processing Conference*, pages 1477–1481, 2021. doi: 10.23919/Eusipco47968.2020.9287846.

[39] Tongtong Wu, Massimo Caccia, Zhuang Li, Yuan-Fang Li, Guilin Qi, and Gholamreza Haffari. Pretrained Language Models in Continual Learning: A Comparative Study. In *Proceedings of the 10th International Conference on Learning Representations*, 2022.

[40] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online Coreset Selection for Rehearsal-based Continual Learning. In *Proceedings of the 10th International Conference on Learning Representations*, 2022.

[41] Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task Agnostic Continual Learning Using Online Variational Bayes. *arXiv preprint arXiv:1803.10123*, 2018.

A Related work

When using frozen pretrained embedding functions in continual learning, simple metric-based probabilistic models have been shown to have state-of-the-art performance [27, 14, 29]. For example, [14] show that online linear discriminant analysis (LDA) on top of a frozen pretrained backbone is the best performing method in their experiments and [27] show that class-conditional Gaussian models perform the best in certain settings. These probabilistic models have the advantage that it is possible to learn the same estimate of parameters given any ordering of the data, including i.i.d orderings. However, in many cases frozen pretrained embedding functions cannot be used, which is the case we look at in this work, due to a lack of relevant data for pretraining or due to distribution shift a pretrained embedding function will perform badly [27].

When the embedding function has to be learnt, there are three main paradigms for solving continual learning problems: *regularisation*, *parameter-isolation* and *replay* [7]. *Regularisation* methods aim to prevent catastrophic forgetting by adding terms to the loss which try to prevent the network from erasing the information of previous tasks [20, 18]. *Parameter-Isolation* methods look at controlling what parameters of a neural network are used for what tasks [24], which can often be seen as a *hard* version of regularisation methods. Finally, *Replay* methods aim to solve continual learning problems by storing a subset of previously seen samples, which are then trained on alongside new incoming data. This is the approach our method is part of. Replay methods have been shown to have competitive if not the best performance across many settings [35, 39, 25]. The standard replay method is experience replay (ER) [6, 1], which in the setting explored in this work where we can not store all the current task's data, selects samples to store using reservoir sampling [37], we call this variant ER-reservoir. One of the main questions to be answered by a replay-based approach is how to select what samples to store in memory. While reservoir sampling has been shown to be very effective [38] there have been other methods proposed for sample selection, for example ones which use information-theoretic criteria [38] and others maximising the diversity of the gradients of stored examples [2]. There has also been a Bayesian method proposed to select samples called InfoGS [33], which is somewhat similar to our method but only uses the probability model to select samples, not for prediction or training, and unlike our work looks at *class-incremental learning*, where methods do not have access to task identifiers.

There has been considerable work on using Bayesian methods in continual learning [26, 8, 22], perhaps inspired by the fact that true online Bayesian inference cannot suffer from catastrophic forgetting [26]. Bayesian perspectives have mainly been used for regularisation based methods, where a popular approach is to use variational inference [26, 11, 41]. These variational inference methods focus on the offline continual learning setting where a learner has access to all of the data of a task at the same time and, in previous work, are often limited to being used in conjunction with small neural networks, mainly due to the need to sample multiple networks when calculating the loss [15, 26]. Therefore, these methods are not suited to the settings we consider in this paper. Bayesian methods have also been used in generative replay based approaches, where instead of storing and replaying real samples they use generated pseudo-samples [31]. However, when it comes to replay, with real examples, there has been relatively little work on using Bayesian methods, specifically for empirical Bayesian methods where there is a requirement to prevent forgetting when fitting the embedding function, which we aim to help to fill in by proposing our method.

In this work we look at two types of settings, one where the sets of class labels are disjoint between tasks and another where there is an overlap been the classes seen in each task. Disjoint tasks is a commonly explored setting [7] while having some class overlap between tasks is not often studied, but arguably is more realistic. Slightly tangential to the settings considered here, there has been some work looking at class overlap settings where blurry tasks are considered [30, 4, 2], that is when a disjoint task setting is modified by adding to each task a small number of samples sampled from any class, often around 10% of the original task's size.

B Continual learning of embedding functions and probabilistic models

To solve the problem of exploiting simple probabilistic models, like LDA, when the embedding function has to be learnt online, we propose an empirical Bayesian method (shown in Algorithm 1). Let Z denote an embedding space. We assume that we have some embedding function $f_\phi : X \to Z$ and a fixed memory of size m, which counts the number of items (each an element of $X \times C \times T$) that can fit in a memory buffer. The memory buffer stores as one element a sample (x, y) and its associated task index $t_{(x,y)}$—in common with previous work [5]. We propose to use a Bayesian probabilistic model with parameters θ, such that

$$p(y|x, t_{(x,y)}) = \int p(y|f_\phi(x), \theta, t_{(x,y)})p(\theta)d\theta, \tag{6}$$

and assume the labels are conditionally independent given the task, θ and the data instances (each an element of X). Therefore if f_ϕ is fixed, we can update our beliefs about the parameters on seeing batch D_j and its task identifier t_{D_j} by simply using Bayes rule, calculating the posterior $p(\theta|D_j, D_{<j})$. However, in our setting, we need to update f_ϕ continually, which would change the value of $p(\theta|D_j, D_{<j})$. We cannot recompute $p(\theta|D_j, D_{<j})$ as we are unable to store all the previous samples in memory. Therefore, we propose to store in the buffer a set of representative samples with which we can compute an approximate posterior after a change in f_ϕ. To achieve this goal, after every batch, we select a subset of data points, from the previous memory and current batch, to store in memory. We choose the subset which minimises the KL divergence between the posterior distribution induced by the subset and the posterior distribution induced by the whole data,

$$M_{j+1} = \underset{M' \subseteq (D_j \times \{t_{D_j}\}) \cup M_j, |M'|=m}{\arg\min} \mathrm{KL}(p(\theta|D_j, t_{D_j}, M_j) \| p(\theta|M')), \tag{7}$$

where M_j is the memory which was carried forward when receiving the jth batch of data.

To update the embedding function, for each arriving batch D_j, task identifier t_{D_j} and corresponding memory M_j, the method randomly subsamples $R_j \subset M_j$ of given size r. Then we update f_ϕ using

$$\phi_{j+1} = \phi_j + \eta \nabla_\phi \log p(D_j^Y, R_j^Y | D_j^Z, t_{D_j}, R_j^Z, S_{tasks}, M_j/R_j), \tag{8}$$

where $S_{tasks} = \{t_{(x,y)} | (x, y) \in R_j\}$, η is a learning rate, $D_j^Z = \{f_\phi(x) | (x, y) \in D_j\}$, $D_j^Y = \{y | (x, y) \in D_j\}$ and R_j^Z and R_j^Y are defined likewise. This is a gradient step on a conditional marginal log-likelihood (i.e. we compute the marginal log-likelihood on some of the data having conditioned on the rest). Conditioning on part of the data helps to reduce the sensitivity of learning in respect to a potentially uninformative prior and has been shown in previous work to increase generalisation performance [23].[1]

[1]The marginal likelihood is the average likelihood over models generated from the prior, while the conditional likelihood is the average likelihood over models from the posterior induced by M_j/R_j. Averaging over an

Algorithm 1 General approach, learning step at time j

```
1: input $D_j$ (current batch), $t_{D_j}$ (current task identifier), $M_j$ (current memory), $\phi_j$ (current embedding function)
2:
3: Update embedding function
4: $R_j = \mathrm{UniformSample}(M_j)$
5: $\phi_{j+1} = \phi_j + \eta \nabla_\phi \log p(D_j^Y, R_j^Y | D_j^Z, t_{D_j}, R_j^Z, S_{tasks}, M_j/R_j)$
6:
7: Update memory
8: $M_{j+1} = \underset{M' \subseteq (D_j \times \{t_{D_j}\}) \cup M_j, |M'|=m}{\arg\min} \mathrm{KL}(p(\theta | D_j, t_{D_j}, M_j) \| p(\theta | M'))$
```

One way to view of our update procedure for the embedding function f_ϕ is that it is a variant of experience replay, with a different loss function. Experience replay is a highly performing method to prevent forgetting [35, 39, 25], so therefore our method to learn the embedding function should be resistant to forgetting. Additionally, we note that their exists methods to select what examples to replay [1, 32] which are complementary/orthogonal to this work, and could be combined with our approach. We further note that while our general formulation requires the continual learning setting to provide task identifiers, if $p(y|x, \theta, t_{(x,y)}) = p(y|x, \theta)$, where the distribution of x can still shift between tasks, our general approach can be used in task-agnostic settings as well, which we leave to future work.

C Algorithm for DeepCCG

Algorithm 2 DeepCCG, learning step at time j

```
1: input $D_j$ (current batch), $t_{D_j}$ (current task identifier), $M_j$ (current memory), $\phi_j$ (current embedding function)
2:
3: Update embedding function
4: $R_j = \mathrm{UniformSample}(M_j)$
5: $\phi_{j+1} \leftarrow \phi_j + \eta \nabla_\phi \sum_{(x,y) \in D_j \cup R} \log(p(y|z = f_\phi(x), t_{(x,y)}, M_j/R_j))$
6:
7: Update memory
8: for each class $c$ in $M_j$ do
9:     initialise $\beta$
10:    for 1 to $B$ do
11:        $\beta \leftarrow \beta + \eta \nabla_\beta \mathcal{L}(\beta; D_{j,c}, M_{j,c})$
12:    end for
13:    Set $M_{j+1}$ to be the set of samples, with their task identifiers, with the $m$ largest $\beta$ values
14: end for
```

D Details of learning the embedding function

For our deep class conditional Gaussian method (DeepCCG), to compute the update to the embedding function f_ϕ, when given a batch D_j and a corresponding memory M_j, we proceed using the following steps. Firstly, we randomly sample $R_j \subset M_j$ of size r from memory. Then using only the other samples in memory, M_j/R_j, we compute the posterior density for each class mean— μ_c, with $c \in C$:

$$p(\mu_c | M_j/R_j) = p(\mu_c | M_{j,c}^z / R_j^z) \tag{9}$$

$$= \mathcal{N}\left(\mu_y | \overline{M_{j,c}^z / R_j^z}, \frac{1}{m-r}\mathbf{I}\right), \tag{10}$$

where $M_{j,c}^z = \{f_\phi(x) | (x, y) \in M_j \wedge y = c\}$ are the embeddings of points in the memory buffer with class y. R_j^z is defined likewise and we use the notation $\overline{S}$ to denote the mean of the elements of a set

uninformative prior will often result in a flat marginal likelihood which is not informative of what embedding function fits the data well. Whereas, conditioning the marginal likelihood on M_j/R_j, the posterior we average over is more informative, having some belief of where the position of embedded data instances should be, and so provides a better signal to fit the embedding function.

S. Then we compute the posterior distribution of the embedded inputs $z \in D_j^z \cup R_j^z$ for each class $c \in C$ utilizing

$$p(z|c, M_j/R_j) = p(z|c, M_{j,c}^z/R_j^z) \tag{11}$$

$$= \int p(z|c, \mu_c) p(\mu_c | M_{j,c}^z/R_j^z) d\mu_c \tag{12}$$

$$= \mathcal{N}\left(z | \overline{M_{j,c}^z/R_j^z}, (1 + \frac{1}{m - r})\mathbf{I}\right). \tag{13}$$

Next, we compute the posterior predictive for each sample $(x, y) \in D_j \cup R_j$ with a task identifier $t_{(x,y)}$, which is known for samples in the current batch and is stored by our method for samples stored in memory, and where $z = f_\phi(x)$ using

$$p(y|z, t_{(x,y)}, M_j/R_j) = \frac{p(z|y, M_{j,y}^z/R_j^z) p(y|t_{(x,y)})}{\sum_{c \in C} p(z|Y = c, M_{j,c}^z/R_j^z) p(Y = c|t_{(x,y)})} \tag{14}$$

$$= \frac{p(z|y, M_{j,y}^z/R_j^z)}{\sum_{c \in C_{t_{(x,y)}}} p(z|Y = c, M_{j,c}^z/R_j^z)} \tag{15}$$

Finally, we update the embedding function by performing a gradient step using the formula

$$\phi_{j+1} = \phi_j + \eta \nabla_\phi \sum_{(x,y) \in D_j \cup R_j} \log(p(y|z = f_\phi(x), t_{(x,y)}, M_j/R_j)), \tag{16}$$

which can be seen as a per-sample conditional marginal log-likelihood.

By using Eq. 13 and 15, the closed form of the posterior predictive distribution for a sample (x, y) with task identifier $t_{(x,y)}$ and where $z = f_\phi(x)$ is

$$p(y|z, t_{(x,y)}, M_j/R_j) = \frac{p(z|y, M_{j,y}^z/R_j^z)}{\sum_{c \in C_{t_{(x,y)}}} p(z|Y = c, M_{j,c}^z/R_j^z)} \tag{17}$$

$$= \frac{\mathcal{N}\left(z|\overline{M_{j,y}^z/R_j^z}, (1 + \frac{1}{m-r})\mathbf{I}\right)}{\sum_{c \in C_{t_{(x,y)}}} \mathcal{N}\left(z|\overline{M_{j,c}^z/R_j^z}, (1 + \frac{1}{m-r})\mathbf{I}\right)} \tag{18}$$

E Additional Experimental details

In addition to the experimental details mentioned in the main body of the paper, there are a few more details to mention. Firstly, for methods with hyperparameters we performed a grid search using the same experimental set up as the real experiments and 10% of the training data as validation data. This led to the selection of the hyperparameters of EWC and PackNet, the two methods with free hyperparameters, shown in Table 3. Secondly, we use the same learning rate of 0.1 and the standard gradient decent optimiser for all methods. These were chosen by looking at the commonly selected values in previous work [25] and were shown to be performative for all methods tested. Finally, all experiments were run on a laptop with a single NVIDIA GeForce GTX 1050 GPU.

Table 3: Values selected for the hyperparameters of EWC and PackNet, which are the regularisation coefficient and the percentage of available filters to be used per task, respectively. SW stands for shifting window and DT stands for disjoint tasks.

	CIFAR-100		**MiniImageNet**		**CIFAR-10**	
Method	**SW**	**DT**	**SW**	**DT**	**SW**	**DT**
EWC	6	6	2	9	1	6
PackNet	0.1	0.1	0.05	0.2	0.3	0.3

Table 4: Results of performing an ablation on DeepCCG, where we report results on DeepCCG using reservoir sampling instead of its own sample selection method (DeepCCG-reservoir), when we replace the class-conditional Gaussian model with a standard classifier head (DeepCCG-standardHead) and when we do both modifications, in which case our method reduces to the ER-reservoir method. We report the mean average accuracy with their standard errors across three independent runs on the CIFAR-100 shifting window setting.

Method	**Average accuracy**
ER-reservoir	54.05 ± 0.626
DeepCCG-reservoir	49.95 ± 0.743
DeepCCG-standardHead	44.05 ± 0.264
DeepCCG	56.62 ± 0.288

Table 5: Results of experiments looking at the effect of memory buffer size m for the replay methods tested, using the shifting window setting on CIFAR-100. We report mean average accuracy with their standard errors across three independent runs.

Method	**m=750**	**m=1000**	**m=1250**
ER-reservoir	53.17 ± 0.656	54.05 ± 0.626	55.22 ± 0.592
A-GEM	27.18 ± 1.091	29.01 ± 1.449	27.87 ± 0.172
EntropySS	50.52 ± 0.725	51.80 ± 0.700	53.34 ± 0.372
GSS	47.15 ± 0.766	48.20 ± 0.332	46.18 ± 0.341
DeepCCG (ours)	$\mathbf{53.42 \pm 0.460}$	$\mathbf{56.62 \pm 0.288}$	$\mathbf{57.98 \pm 0.514}$

F Ablation Study

One way to view DeepCCG is as a relative of ER-reservoir, where the methods differ in the sample section mechanism used and the probabilistic model defined on top of the embedding function. Therefore, to analyse where the performance of DeepCCG comes from, we ablate our method by separately changing these two components to what they are in ER-reservoir. The results are presented in Table 4 and show that both components on their own do not improve performance over ER-reservoir but together they do. We go into detail of the ablation of both components, the sample selection mechanism and probabilistic model, in the next two paragraphs and then present an ablation on the memory size used for the replay methods tested, where we show that DeepCCG performs the best over all the memory sizes tested on.

Sample selection To explore the performance of our memory selection mechanism, we compare our full method to when we select samples using reservoir sampling (DeepCCG-reservoir), showing that the full method performs the best in the setting tested on. DeepCCG achieves a mean average accuracy 56.62% whereas DeepCCG-reservoir obtains a lower mean average accuracy of 49.95% (see Table 4). This shows that our memory selection mechanism is necessary for DeepCCG to obtain good performance. Additionally, we predict that our memory selection mechanism will also perform well in settings with noisy, OOD and/or imbalanced data, unlike reservoir sampling [33, 40, 2], which we leave to future work to validate.

Class conditional Gaussian model To understand the performance of using a class conditional Gaussian model instead of the standard discriminative softmax model, we look at the performance of DeepCCG without using the class conditional Gaussian model. That is, we evaluate the performance of DeepCCG when we replace the class conditional Gaussian model with a standard output head (a fully connected layer, then a softmax), while still using our sample selection method—calling it DeepCCG-standardHead. The results show that DeepCCG-standardHead performs poorly, achieving an average accuracy of 44.05% (see Table 4), which is significantly worse than when using reservoir sampling, as in ER-reservoir, which gets a mean average accuracy of 54.05%. This is to be expected as our sample selection strategy is designed with the assumption that our embedding function will cluster the data into per-class clusters which may not be true when using a standard multi-head output layer, while when using a class conditional Gaussian model the embedding function will be encouraged to create per-class clusters.

Memory size One additional useful experiment is looking at the relationship between performance and the size of memory used for DeepCCG. Therefore, in Table 5 we show the performance of DeepCCG and the other replay methods compared against with varying memory size. Table 5 shows that when the memory size is increased to $m = 1250$, DeepCCG has an improvement in average accuracy relative to other methods, as it achieves 2.76% better than any other method for $m = 1250$, while for $m = 1000$ it achieves 2.57% better than any other method, a 0.19% improvement. When the memory size is decreased to $m = 750$, we see that DeepCCG's performance drops more than other methods as it is only 0.25% better than other methods in this case. Therefore, our experiments show that compared to other replay methods DeepCCG's performance increases the most when m increases and that for small buffer sizes DeepCCG performs less well, potentially due to the fact that the examples in the memory buffer are used to infer the posterior over the means of the per-class Gaussians and so the method needs a given amount of examples to specify the means well. We also note that in our experiments the performance ranking of the methods does not change with m..