Continual Learning using a Bayesian Nonparametric Dictionary of Weight Factors

Nikhil Mehta[1], **Kevin J Liang**[1,2], **Vinay K Verma**[1], **Lawrence Carin**[1]
[1]Duke University [2]Facebook AI

Abstract

Naively trained neural networks tend to experience catastrophic forgetting in sequential task settings, where data from previous tasks are unavailable. A number of methods, using various model expansion strategies, have been proposed recently as possible solutions. However, determining how much to expand the model is left to the practitioner, and often a constant schedule is chosen for simplicity, regardless of how complex the incoming task is. Instead, we propose a principled Bayesian nonparametric approach based on the Indian Buffet Process (IBP) prior, letting the data determine how much to expand the model complexity. We pair this with a factorization of the neural network's weight matrices. Such an approach allows the number of factors of each weight matrix to scale with the complexity of the task, while the IBP prior encourages sparse weight factor selection and factor reuse, promoting positive knowledge transfer between tasks. We demonstrate the effectiveness of our method on a number of continual learning benchmarks and analyze how weight factors are allocated and reused throughout the training.

1 Introduction

Deep learning, trained primarily on a single task under the assumption of independent and identically distributed (*i.i.d.*) data, has made enormous progress in recent years. However, when naively trained sequentially on multiple tasks, without revisiting previous tasks, neural networks are known to suffer catastrophic forgetting (McCloskey & Cohen, 1989; Ratcliff, 1990): the ability to perform old tasks is often lost while learning new ones. In contrast, biological life is capable of learning many tasks throughout a lifetime from decidedly non-*i.i.d.* experiences, acquiring new skills and reusing old ones to learn fresh abilities, all while retaining important previous knowledge. As we strive to make artificial systems increasingly more intelligent, natural life's ability to learn continually is an important capability to emulate.

Continual learning (Parisi et al., 2019) has attracted considerable attention recently in machine learning research, and a number of desiderata have emerged. Models should be able to learn multiple tasks sequentially, with the eventual number and complexity of tasks unknown. Importantly, new tasks should be learned without catastrophically forgetting previous ones, ideally without having to keep any data from previous tasks to re-train on. Models should also be capable of positive transfer: previously learned tasks should help with the learning of new tasks. Knowledge transfer between tasks maximizes sample efficiency, with this particularly important when data are scarce.

A number of methods (Rusu et al., 2016; Zhang et al., 2019; Lee et al., 2020) address continual learning through expansion: the model is grown with each additional task. By diverting learning to new network components for each task, these approaches mitigate catastrophic forgetting by design, as previously learned parameters are left undisturbed. A key challenge for these strategies is deciding when and how much to expand the network. While it is typically claimed that this can be tailored to the incoming task, doing so requires human estimation of how much expansion is needed, which is not a straightforward process. Instead, a preset, constant expansion is commonly employed for each new task.

Rather than relying on engineered heuristics, we choose to let the data dictate the model-expansion rate, employing a Bayesian nonparametric approach. Specifically, we couple rank-1 weight factor (WF) dictionary learning with the Indian Buffet Process (IBP) (Ghahra-

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

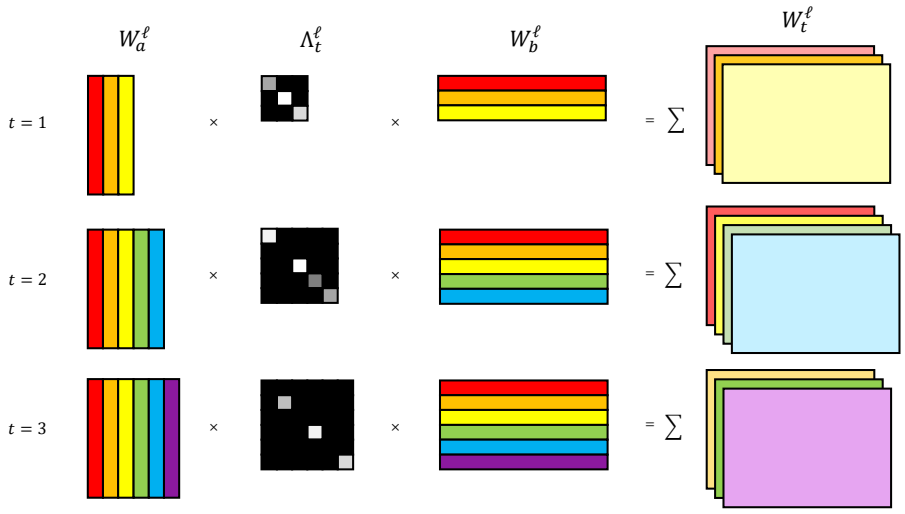


Figure 1: Layer-wise weight factors for continual learning. Dictionaries of weight factors $\mathbf{W}_a^\ell$ and $\mathbf{W}_b^\ell$ are shared across all tasks, and a task-specific sparse diagonal matrix $\mathbf{\Lambda}_t^\ell$ specifies the active factors (in this figure, elements of $\mathbf{\Lambda}_t^\ell$ that are black are zero, and brighter shades correspond to larger numbers). The weighted sum of the active weight factors yields the weight matrix for a particular task. The number of factors (columns of $\mathbf{W}_a^\ell$ and rows of $\mathbf{W}_b^\ell$) grows as needed with more tasks, with select factors being reused in future tasks. Best viewed in color.

mani & Griffiths, 2006), creating a framework we call **IBP-WF**. An IBP-based formulation allows automatic scaling of the network, but only as needed, even if the number or complexity of future tasks is unknown initially. An IBP prior also naturally encourages recycling of previously learned skills, enabling positive transfer between tasks, which other expansion methods tend to either ignore or deal with in a more *ad hoc* manner. Finally, Bayesian modeling enables model sampling, allowing for both ensembling models for increased accuracy and uncertainty estimation, which are important but rarely discussed topics in continual learning.

Our main contributions are as follows. (*i*) We introduce learning a rank-1 weight factor dictionary for a neural network expected to perform multiple tasks. We then introduce the Indian Buffet Process as a prior for each task's weight factor selection, showing why the IBP is a natural choice given continual learning's desiderata. (*ii*) We introduce a simple-but-effective method based on feature statistics for inferring task identity (ID) in incremental class settings. (*iii*) The effectiveness of IBP-WF is demonstrated on a number of continual learning tasks, outperforming other methods. We also visualize the weight factor usage across tasks, confirming both sparsity and reuse of these factors.

2 Methods

2.1 Weight Factor Dictionary Learning

Consider a multilayer perceptron (MLP) with layers $\ell = 1, ..., L$. In a continual learning setting, we would like this neural network to learn multiple tasks. Given differences between tasks, the neural network may require a different set of weight matrices $\{\mathbf{W}_t^\ell\}_{\ell=1}^L$ for each task t. While $\{\mathbf{W}_t^\ell\}_{\ell=1}^L$ could be learned separately for each task, such a model does not incorporate knowledge reuse, and the total number of model parameters grows linearly with the number of tasks T. While immune from catastrophic forgetting, such an approach is inefficient in both computation and data.

Instead of completely independent models for each task, we propose constituting $\mathbf{W}_t^\ell$ as follows:

$$\mathbf{W}_t^\ell = \mathbf{W}_a^\ell \mathbf{\Lambda}_t^\ell \mathbf{W}_b^\ell \qquad \text{s.t.} \quad \mathbf{\Lambda}_t^\ell = \text{diag}(\boldsymbol{\lambda}_t^\ell) \tag{1}$$

where $\mathbf{W}_a^\ell \in \mathbb{R}^{J \times F}$ and $\mathbf{W}_b^\ell \in \mathbb{R}^{F \times M}$ are global parameters shared across tasks and $\boldsymbol{\lambda}_t^\ell \in \mathbb{R}^F$ is a task-specific vector. The determination of F is explained in Section 2.2, but in general F is chosen such that after T tasks, the total number of parameters of this factorized model is $(J + T + M) \cdot F$, which is significantly less than the $J \cdot M \cdot T$ parameters that would result from learning each task independently. We may equivalently express (1) as the weighted sum of rank-1 matrices formed from the outer product of the vectors corresponding to the columns $\mathbf{W}_a^\ell$ and rows of $\mathbf{W}_b^\ell$:

$$\mathbf{W}_t^\ell = \sum_{k=1}^{F} \boldsymbol{\lambda}_{t,k}^\ell \left(\mathbf{w}_{a,k}^\ell \otimes \mathbf{w}_{b,k}^\ell \right) \tag{2}$$

where $\otimes$ denotes the outer product, and the pair $\mathbf{w}_{a,k}^\ell$ and $\mathbf{w}_{b,k}^\ell$ is the k^{th} column and row of $\mathbf{W}_a^\ell$ and $\mathbf{W}_b^\ell$, respectively. Under this construction, the pairs of

corresponding columns of $\mathbf{W}_a^\ell$ and rows of $\mathbf{W}_b^\ell$ can be interpreted as a dictionary of weight *factors*, while the values in $\boldsymbol{\lambda}_t^\ell$ are the factor *scores* for a particular task (see Figure 1). By sharing these global weight factors, the model can reuse features and transfer knowledge between tasks, with $\boldsymbol{\lambda}_t^\ell$ selecting and weighting the factors for a particular task t. We construct the factor scores $\boldsymbol{\lambda}_t^\ell$ as the following element-wise product:

$$\boldsymbol{\lambda}_t^\ell = \mathbf{r}_t^\ell \odot \mathbf{b}_t^\ell \tag{3}$$

where $\mathbf{b}_t^\ell \in \{0,1\}^F$ indicates the *active* factors for task t and $\mathbf{r}_t^\ell \in \mathbb{R}^F$ specifies the corresponding factor *strength*. By imposing sparsity with $\mathbf{b}_t^\ell$, we concentrate skills for each task into specific factors, leaving room for learning other tasks with other factors.

Generalizing to convolutional kernels While learning the weight factors in (1) was formulated for fully connected layers, it can be generalized to other types of layers as well, including 2D convolutional layers. Unlike the 2D weight matrices comprising the fully connected layers of a MLP, convolutional kernels are 4D: in addition to number of input and output channels (C_{in} and C_{out}), they also have two spatial dimensions denoting the height (H) and width (W) of the convolutional filter. While learning 4D tensor factors is certainly possible, in practice H and W tend to be small (*e.g.* $H = W = 3$), so we instead choose to reshape the kernel ($\mathbb{R}^{H\times W\times C_{in}\times C_{out}}$) into a 2D matrix ($\mathbb{R}^{(HWC_{in})\times C_{out}}$). We then proceed with the same weight factor learning as in (1).

2.2 Indian Buffet Process for Weight Factors

Critical to the proposed layer-wise weight factors is the number of factors F: too few and the model lacks sufficient expressivity to model every task; too many and the model consumes more memory and computation than necessary. To further complicate matters, the number of necessary factors likely increases monotonically as the model encounters more tasks. While a particular choice of F may be appropriate for T tasks, it may no longer be sufficient after T', with $T' > T$.

Rather than setting it as a constant, we let F grow naturally with the number of tasks. There are a number of expansion strategies for continual learning that have been proposed over the years (Rusu et al., 2016; Hung et al., 2019; Zhang et al., 2019; Lee et al., 2020). Many of these expand the model by a constant amount per task, or rely on the model designer to specify a schedule or heuristics for the size of the expansion. These hand-tuned strategies can be brittle, and require expert knowledge on the complexity of incoming tasks. Additionally, prior works do not use weight factor dictionaries, so expansion involves adding additional nodes to each hidden layer or learning entirely new models, which can increase test-time computation.

Instead, we employ Bayesian nonparametrics, inferring in a principled manner the scores for the proposed rank-1 weight factors for each task and the total number of factors F needed. In particular, we impose the stick-breaking construction of the IBP (Ghahramani & Griffiths, 2006) as a prior for factor selection:

$$v_{t,i}^\ell \sim \text{Beta}(\alpha, 1) \tag{4}$$

$$\pi_{t,k}^\ell = \prod_{i=1}^{k} v_{t,i}^\ell \tag{5}$$

$$b_{t,k}^\ell \sim \text{Bernoulli}(\pi_{t,k}^\ell) \tag{6}$$

where α is a hyperparameter controlling the expected number of nonzero factor scores, and $k = 1, 2, ...F$ indexes the factor. For the global parameters ($\mathbf{W}_a^\ell$ and $\mathbf{W}_b^\ell$) and the local factor strength ($\mathbf{r}_t^\ell$), we use point estimates. See Appendix A for an overview of the IBP and the connection of IBP-WF with IBP-based dictionary learning. Leveraging the IBP in conjunction with dictionary learning provides a number of natural advantages within the context of continual learning:

Dynamic control of F The IBP allows the number of factors F to be determined nonparametrically and dynamically, growing only as necessary given the complexity of each individual task. Simpler tasks (or ones similar to previous tasks) may require learning fewer new factors, while more complex ones lead to more, all inferred automatically. While F can theoretically grow unbounded, it does so harmonically – much slower than the requisite linear growth of the number of tasks.

Factor reuse and positive transfer Given that continual learning is often deployed when tasks are at least somewhat correlated, training independent models can lead to learning redundant features, which is inefficient both in training data and test time computation. On the other hand, the construction of $\boldsymbol{\pi}_t^\ell$ (see (5)) actively encourages reuse of existing weight factors, prioritizing recycling previously learned skills for new tasks over creating new ones, which leads to positive forward transfer. This is in contrast to other methods whose only source of transfer is initializing from the previous task's weights, whose transfer advantage may be quickly wiped away by gradient descent.

Catastrophic forgetting mitigation The newly learned weight factors in W_a and W_b are frozen at the end of a task. This mirrors the freezing of previously learned weights in existing expansion methods. By blocking the gradients to weights learned from previous tasks, we avoid forgetting the model's ability to perform older tasks. Note that while factors learned from a previous task are frozen, the factor scores may change

with each incoming task allowing the model to control the usage of a previously trained factor.

Constant inference time cost At test time, IBP weight factors (outer product of each column of $\mathbf{W}_a^\ell$ and row of $\mathbf{W}_b^\ell$) can be pre-computed; given a task, the appropriate factors can be retrieved, weighted, and summed as needed to retrieve $\{\mathbf{W}_t^\ell\}_{\ell=1}^L$. Imposing the IBP prior on the usage of factors induces a prior distribution of Poisson(α) on the number of active factors. Thus, the number of nonzero factors have a prior expectation of α, regardless of the number of tasks T, so the expected forward computation of the model does not grow with T. This avoids one of the pitfalls of other expansion methods (*e.g.,* Rusu et al. (2016); Lee et al. (2020); Kumar et al. (2019)), whose inference-time computation scale with T.

2.3 Variational Inference

To determine which factors should be active for task t, we perform variational inference to infer the posterior of parameters $\theta_t = \{\mathbf{b}_t^\ell, \mathbf{v}_t^\ell\}_{\ell=1}^L$. We assume the following variational distributions:

$$q(\theta_t^\ell) = q(\mathbf{b}_t^\ell)q(\mathbf{v}_t^\ell) \tag{7}$$

$$\mathbf{b}_t^\ell \sim \text{Bernoulli}(\boldsymbol{\pi}_t^\ell) \tag{8}$$

$$\mathbf{v}_t^\ell \sim \text{Kumaraswamy}(\mathbf{c}_t^\ell, \mathbf{d}_t^\ell) \tag{9}$$

We learn the variational parameters $\{\boldsymbol{\pi}_t^\ell, \mathbf{c}_t^\ell, \mathbf{d}_t^\ell\}_{\ell=1}^L$ with Bayes by Backprop (Blundell et al., 2015). As the Beta distribution lacks a differentiable parameterization, we use the similar Kumaraswamy distribution (Kumaraswamy, 1980) as the variational distribution for $\mathbf{v}_t^\ell$. We also use a soft relaxation of the Bernoulli distribution (Maddison et al., 2017) in (6) and (8) to allow backpropagation through discrete random variables. The objective for each task is to maximize the following variational lower bound:

$$\begin{aligned}\mathcal{L}_t = &\sum_{n=1}^{N_t} \mathbb{E}_q \log p\left(y_t^{(n)} | \theta_t, x_t^{(n)}, \mathbf{W}_a, \mathbf{W}_b, \mathbf{r}_t\right) \\ &- \underbrace{\text{KL}\left(q\left(\theta_t\right) || p\left(\theta_t\right)\right)}_{\mathscr{R}}\end{aligned} \tag{10}$$

where N_t is the number of training examples in task t, $\mathbf{W}_a = \{\mathbf{W}_a^\ell\}_{\ell=1}^L$, $\mathbf{W}_b = \{\mathbf{W}_b^\ell\}_{\ell=1}^L$ and $\mathbf{r}_t = \{\mathbf{r}_t^\ell\}_{\ell=1}^L$. The variational lower bound in (10) is maximized with respect to task-specific parameters (θ_t, v_t) and global parameters $(\mathbf{W}_a, \mathbf{W}_b)$. Note that in (10) the first term provides label supervision and the second term ($\mathscr{R}$) regularizes the posterior not to stray too far from the IBP prior. We use a mean-field approximation, allowing expansion of the second term as follows:

$$\mathscr{R} = \sum_{\ell=1}^{L} \text{KL}\left(q(\mathbf{b}_t^\ell) || p(\mathbf{b}_t^\ell | \mathbf{v}_t^\ell)\right) + \text{KL}\left(q(\mathbf{v}_t^\ell) || p(\mathbf{v}_t^\ell)\right) \tag{11}$$

Variational continual learning (VCL) (Nguyen et al., 2018) addresses catastrophic forgetting using online inference, *i.e.*, the posterior inferred from the most recent task is used as a prior for the incoming task. However, recent work (Farquhar & Gal, 2018, 2019) suggests that approximate online inference often does not succeed in mitigating catastrophic forgetting in realistic continual learning settings, as methods based solely on approximate inference rely on a simple prior to capture everything learned on all previous tasks. Thus, instead of performing online inference for all parameters $\{\mathbf{r}_t^\ell, \mathbf{b}_t^\ell, \mathbf{v}_t^\ell\}$, we only apply online inference for $\mathbf{v}_t^\ell$ and learn task-specific parameters $\{\mathbf{r}_t^\ell, \mathbf{b}_t^\ell\}$. Note that online inference over $\mathbf{v}_t^\ell$ encourages the reuse of factors from previous tasks while having task-specific parameters allows the model to easily adapt to a new task by using new factors. For the second term in (11), we derive an analytic approximation of Kullback-Leibler (KL) divergence between two Kumaraswamy distributions. The derivation and more details on doing online inference with (10) and (11) are included in Appendices B and C.

Preserving knowledge If all of $\mathbf{W}_a^\ell$ and $\mathbf{W}_b^\ell$ were free to move without constraint, then catastrophic forgetting may still occur. Indeed, the model could "reuse" a factor from a previous task and then repurpose it entirely, undermining the ability to do the former task. To prevent this, the weight factors (*i.e.,* the columns of $\mathbf{W}_a^\ell$ and rows of $\mathbf{W}_b^\ell$) with factor probability $\pi_{t,k}^\ell > \kappa$ are locked (*e.g.*, with a stop gradient operator) at the conclusion of a task. Weight factors below the threshold κ are left free to be modified by future tasks. Throughout our experiments, we set the threshold as $\kappa = 0.5$, but this can be adjusted based on tolerance for forgetting. We include an ablation study on selecting κ in the Appendix H.2. Alternatively, other regularization methods (*e.g.,* Kirkpatrick et al. (2017)) can be used to prevent important factors from drifting too far, but we leave this combination to future work.

2.4 Task Inference at Test Time

IBP-WF addresses catastrophic forgetting and allows for positive knowledge transfer. However, as with many continual learning methods, IBP-WF requires the task identity associated with each input at test time in order to select the proper $\{\boldsymbol{\Lambda}_t^\ell\}_{\ell=1}^L$. The validity of this assumption has occasionally been questioned (Farquhar & Gal, 2018; Aljundi et al., 2019; Lee et al., 2020). We outline here a mechanism for enabling IBP-WF to operate in an incremental class setting, inferring the

task identity at test time. Given a data point x, we can infer the task identity by defining the probability of x belonging to a particular task t as follows:

$$P(t|x) \propto P(x|t)P(t) \tag{12}$$

However, using (12) requires learning a generative model $P(x|t)\ \forall t \in \{1, 2, ..., T\}$, which can be expensive in both computation and the number of parameters. To alleviate this issue, we propose a simple yet effective alternative: we define an approximation to $P(x|t)$ by using the feature distribution induced by an intermediate hidden layer of the trained neural network. In particular, we approximate (12) by using $P(t|\phi(x))$ as a surrogate for $P(t|x)$:

$$P(t|x) \approx P(t|\phi(x)) = \frac{P(\phi(x)|t)P(t)}{\sum_{t'} P(\phi(x)|t')P(t')} \tag{13}$$

where ϕ is an intermediate layer defined using the proposed weight factorization as shown in (1) with task-specific weights of the first task. We work with the feature space induced by the parameters of the first task as they are accessible by the training data of all tasks that follow. Next, we assume $P(\phi(x)|t)$ to be a Gaussian distribution: $P(\phi(x)|t) = \mathcal{N}(\phi(x)|\mu_t, \Sigma_t)$, where the parameters are the empirical estimates using the training data.

$$\begin{aligned} \hat{\mu}_t &= \frac{1}{N_t} \sum_{n=1}^{N_t} \phi(x_t^{(n)}), \\ \hat{\Sigma}_t &= \frac{1}{N_t} \sum_{n=1}^{N_t} (\phi(x_t^{(n)}) - \hat{\mu}_t)(\phi(x_t^{(n)}) - \hat{\mu}_t)^T \end{aligned} \tag{14}$$

where $x_t^{(n)}$ is a training sample from task t. When we train our model on task $t > 1$, we use the task-specific weights learned for the first task to compute $\{\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\}$. The parameters $\{\mu_t, \Sigma_t\}$ are stored to infer test-time task identity. While the features may not be exactly Gaussian distributed, this assumption has been shown to work well in deep learning (Heusel et al., 2017; Lee et al., 2018), and we find it effective in practice; see Appendix D.2 for task inference accuracy experiments and t-SNE visualizations of ϕ. Notably, we achieve similar accuracy to generative model task inference (Lee et al., 2020), with a far cheaper method.

Considering the marginal task distribution $P(t) \propto N_t$, the task identity can be inferred as follows:

$$\begin{aligned} \hat{t} = \arg\min_t \Bigg[& \frac{\log |\hat{\Sigma}_t|}{2} - \log(N_t) \\ & + \frac{1}{2}(\phi(x) - \hat{\mu}_t)^T \hat{\Sigma}_t^{-1} (\phi(x) - \hat{\mu}_t) \Bigg] \end{aligned} \tag{15}$$

where $\hat{t}$ is the inferred task and I is the identity matrix. While such a strategy does require storing statistics $\hat{\mu}_t$ and $\hat{\Sigma}_t$, the total size of these is still considerably smaller than parameter statistics required by certain regularization methods (*e.g.,* EWC (Kirkpatrick et al., 2017)), as well as the coresets or generative models used by replay methods (Nguyen et al., 2018; Shin et al., 2017; van de Ven & Tolias, 2018).

Remark: (Informal) The approximation in (13) is exact if ϕ is an invertible map since $P(s|t) = M \times P(x|t)$ with $M = \left|\det \frac{\partial \phi^{-1}}{\partial s}\right|$ when $s = \phi(x)$. See Appendix D.1 for a formal proof.

3 Related Works

There have been a number of diverse continual learning methods that have been proposed in recent years, most of which can be roughly grouped by strategy into a few categories, with some overlap. Regularization-based approaches (Kirkpatrick et al., 2017; Zenke et al., 2017; Li & Hoiem, 2017; Nguyen et al., 2018; Aljundi et al., 2018; Schwarz et al., 2018; Ritter et al., 2018) add a loss term constraining the network parameters to remain close to solutions of previously learned tasks. Others use replay (Kirkpatrick et al., 2017; Lopez-Paz & Ranzato, 2017; Shin et al., 2017; Nguyen et al., 2018; Rolnick et al., 2019), which retrains the model on samples from earlier tasks, either from a saved core set or with a generative model that must be learned.

Another class of continual learning methods rely on expansion, the approach taken by IBP-WF. Progressive Neural Networks (Rusu et al., 2016) learn a new neural network column for each new task, with previous columns' features as additional inputs. While avoiding catastrophic forgetting by design, both memory and computation grow linearly with the number of tasks T, just as if one were to learn independent models per task. Side-tuning (Zhang et al., 2019) learns a separate "side" network for each task, adding the output to a shared base model; while this experiences linear growth T of the model size, it reduces the cost by keeping each side network small. As an alternative to constant growth, Reinforced Continual Learning (Xu & Zhu, 2018) uses an LSTM (Hochreiter & Schmidhuber, 1997) controller and REINFORCE (Williams, 1992) to determine the expansion rate, while Dynamically Expandable Networks (Yoon et al., 2018) expand by a constant amount before using sparsity regularization and loss-based heuristics to prune away unused units. Pruning between tasks is also utilized by Hung et al. (2019), where the pruning and re-training is used to prevent excessive growth of the model. MNDPT (Veniat et al., 2021) adds new modules to the model with new tasks, reusing modules of older similar tasks.

Table 1: The average accuracy of seen tasks after learning on a sequence of tasks using a MLP.

Method	Replay	Split MNIST		Permuted MNIST	
		Incremental Task	Incremental Class	Incremental Task	Incremental Class
Adagrad		98.24 ± 0.59	19.73 ± 0.12	90.78 ± 0.18	27.59 ± 1.07
EWC		98.64 ± 0.87	19.89 ± 0.04	92.49 ± 0.34	23.97 ± 3.21
SI		99.16 ± 0.52	19.71 ± 0.10	95.45 ± 0.59	56.88 ± 4.93
MAS		99.23 ± 0.18	19.58 ± 0.11	96.76 ± 0.26	49.95 ± 2.53
LwF		99.61 ± 0.05	22.31 ± 0.51	81.47 ± 0.38	30.63 ± 0.76
VCL		96.79 ± 0.35	19.43 ± 0.02	91.33 ± 0.93	16.21 ± 0.59
Naive rehearsal	✓	99.39 ± 0.11	85.97 ± 0.75	96.75 ± 0.19	96.53 ± 0.11
VCL-Coreset	✓	98.75 ± 0.06	85.15 ± 0.61	93.46 ± 0.49	66.96 ± 4.10
GEM	✓	98.56 ± 0.08	88.28 ± 0.26	97.14 ± 0.09	96.88 ± 0.05
DGR	✓	99.54 ± 0.05	91.61 ± 0.26	93.74 ± 0.24	92.96 ± 0.53
RtF	✓	99.66 ± 0.03	**92.56** ± 0.21	97.31 ± 0.01	96.23 ± 0.04
IBP-WF (Ours)		**99.69** ± 0.05	92.40 ± 0.15	**97.52** ± 0.06	**97.50** ± 0.06

A few works have also explored continual learning from a Bayesian nonparametric perspective. Lee et al. (2020) combine the Dirichlet process with a mixture of experts, where each expert is a neural network responsible for a subset of the data. While this approach does allow the data to dictate model expansion, mixing only occurs at the prediction representation, as opposed to throughout the model as in IBP-WF. This mixture of experts thus can lead to redundant feature learning and unnecessary extra computation. Recently, there have been other attempts to apply IBP to learn the structure of a neural network for continual learning. Kumar et al. (2019) proposed Bayesian Structure Adaptation for Continual Learning (BSCL), which expands the hidden units in each layer using a binary mask for the weight filters, with an IBP as the prior of the mask. Since BSCL uses an IBP over the entire weight matrix, the inference parameters grow quadratically with the layer size requiring more memory and making it hard to scale to large networks, whereas we use the IBP to model the factor scores where the inference parameters only grow linearly with the layer size. IBP Bayesian Neural Networks (IBNN) (Kessler et al., 2020) have used sequential Bayes to apply IBP to the hidden layer activations of a fully connected neural network. In contrast, we expand the number of factors of the weight matrix in each layer, allowing us to scale our method to deeper networks with convolutional layers.

4 Experiments

We evaluate our method in two settings, which we call incremental *task* learning and incremental *class* learning. In incremental task learning, the task identity (ID) of each sample is revealed at test time. In this case, we can simply use the Λ_t from the task ID given. On the other hand, in incremental class learning, we are not given task IDs during testing. This is the more difficult case, with many earlier continual learning methods tending to do poorly. We address this challenge by using the approach described in Section 2.4, inferring the task identity by using the training statistics at an intermediate layer. For task inference in our incremental-class experiments, we consider ϕ in (13) to be the representation after the first layer, as it performed best. We purposely have chosen to *not* supplement IBP-WF with replay, to isolate the advantages of using IBP and weight factors. This puts IBP-WF at a disadvantage compared to replay-based methods. Nevertheless, IBP-WF outperforms or is comparable to replay-based methods. Note that *with* replay (where we no longer freeze the IBP "dishes" after they are learned by a given task), the IBP-WF performance is likely to improve further (via backward transfer); we reserve that for future work.

We additionally perform an ablation study over the effect of the IBP, and then visualize some IBP-WF weight factors to verify some of its behavior. The description of baselines and the training details are in Appendices F and G, respectively. Ablation studies for IBP-WF's α and κ are also included in Appendices H.1 and H.2. All experiments are run on a NVIDIA Titan X GPU.

4.1 Datasets and Architectures

We evaluate IBP-WF on a number of common continual learning benchmarks. For each, the model is trained on a series of classification tasks arriving in sequence. This is done *without* revisiting the data from previous

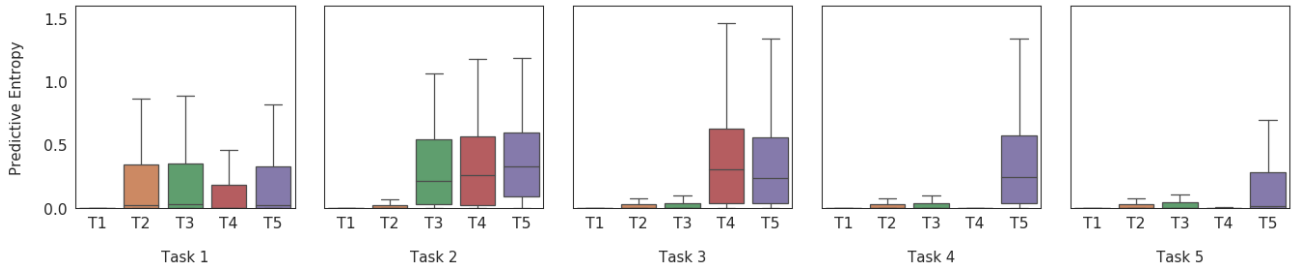
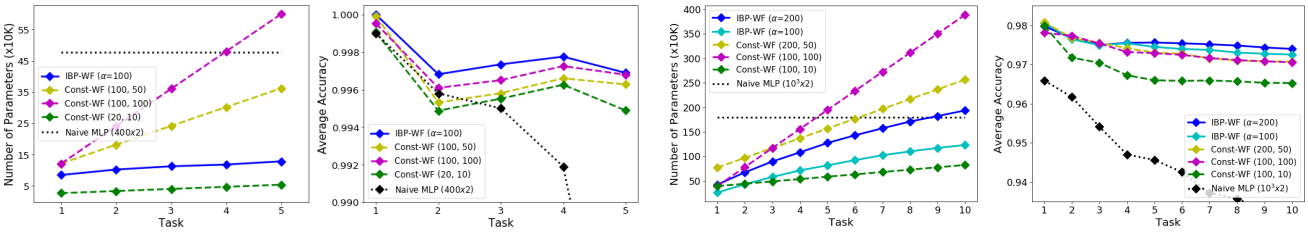


Figure 2: Uncertainty in the incremental class setting for Split MNIST dataset. Each plot depicts the uncertainty on the test sets after training on each task sequentially. The y-axis denotes the uncertainty (as the predictive entropy in nats), and x-axis denotes the test sets ($\mathcal{T}_1$ through $\mathcal{T}_5$) for each task.



(a) Split MNIST (5 tasks)

(b) Permuted MNIST (10 tasks)

Figure 3: Parameter usage (left) and Average accuracy (right) for IBP-WF and Const-WF. We see that IBP-WF performs favorably by learning the required number of factors for each task in contrast to Const-WF.

tasks, unless otherwise stated (*e.g.,* some baselines use a memory buffer for replay to relax this constraint). The standard train/validation/test splits were used.

Split MNIST Following Zenke et al. (2017), the 10 digit classes of the MNIST (LeCun et al., 1998) dataset are split into a series of 5 binary classification tasks: 0 vs 1, 2 vs 3, 4 vs 5, 6 vs 7, and 8 vs 9. In the incremental task setting, where the task ID is given, this reduces to a binary classification problem during testing. In the incremental class setting, without task labels, each model must predict one of $2t$ classes, up to a maximum of 10 once all tasks have been seen.

Permuted MNIST First used to characterize catastrophic forgetting in neural networks by Goodfellow et al. (2013), Permuted MNIST has remained a common continual learning benchmark. The first classification task is typically chosen to be the MNIST dataset, unchanged. Each subsequent task consists of the same 10-way digit classification, but with the pixels of the entire MNIST dataset randomly permuted in a consistent manner. An arbitrary number of tasks can be generated in this manner; for our experiments, we use 10 tasks. In the incremental-task setting, test-time evaluation is a 10-way classification problem, while in incremental class learning we have up to 100 classes.

Results are shown in Table 1. In addition to these results, we also compare against additional baselines in Appendix F. IBP-WF outperforms other methods in most cases and is inferior only to replay-based methods in one setting. Unlike replay-based methods though, IBP-WF does not require saving data examples or separately learning bulky generative models. Compared with non-replay methods, we see significant improvement, especially in the incremental class setting. Additionally, due to the Bayesian nature of IBP-WF, one can quantify the predictive uncertainty, which is a desirable property of a model, especially in decision making. Uncertainty estimates can also be used to detect out-of-distribution samples (Gal, 2016) and adversarial attacks (Smith & Gal, 2018). We demonstrate the former in Figure 2, where data from unseen tasks can be identified by the model's significantly higher uncertainty. See Appendix E for additional details on uncertainty estimation methodology.

Split CIFAR10 We split the CIFAR10 (Krizhevsky, 2009) dataset into a sequence of 5 binary classification tasks (see Figure 5 for the class pairings). Similar to Split MNIST, this is a binary classification problem at test time in the incremental task setting, and $2t$-wise classification in the incremental class setting.

State-of-the-art classification performance on CIFAR10 is typically achieved with convolutional neural networks. We demonstrate that IBP-WF can scale by using ResNet-20 (He et al., 2016) as our architecture, with separate batch normalization layers for each task. For task inference at test time, we take the average

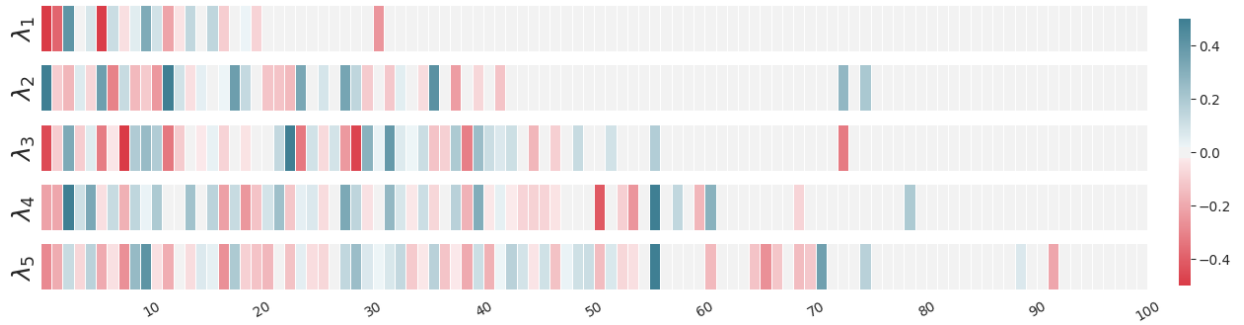


Figure 4: (Horizontal axis: Weight factors in the dictionary, vertical axis: Factor scores for 5 tasks in Split CIFAR10). High overlap in *earlier* factor scores indicate that tasks reuse factors that were learned by previously seen tasks. The IBP prior encourages the reuse of previously learned factors, while inducing sparsity on the total number of active factors.

Table 2: The average accuracy of all seen tasks after learning the task sequence.

Method	Replay	Split CIFAR10	
		Incremental Task	Incremental Class
Adagrad		71.56 ± 1.73	19.59 ± 0.02
L_2		74.36 ± 0.83	16.86 ± 0.08
EWC		75.91 ± 1.64	18.84 ± 0.06
Online EWC		88.34 ± 1.06	17.54 ± 0.34
SI		87.19 ± 2.06	19.06 ± 0.09
MAS		85.68 ± 1.36	16.29 ± 0.14
Naive rehearsal	✓	87.79 ± 0.88	34.24 ± 1.38
IBP-WF (Ours)		**90.94** ± 2.65	**40.40** ± 0.21

across spatial dimensions H and W to get the feature statistics ϕ and then proceed with the parameter estimation procedure introduced in (14). We keep a buffer of 400 images from previous tasks for the naive rehearsal baseline. Table 2 shows the results on Split CIFAR10. We again see that IBP-WF performs well relative to the baseline methods.

4.2 IBP Ablation Study

To demonstrate the benefits of the IBP prior, we perform an ablation study comparing IBP-WF with a variant without the IBP for weight factor selection and expansion, in which factor usage and model growth must be manually set. As the expansion rate is constant, we call this Const-WF(ν, ω), parameterized by the starting number of factors (ν) in the first task and number of new factors (ω) added per task. We compare IBP-WF with Const-WF for several corresponding settings in Figure 3. Importantly, the IBP allows for automatic expansion as needed, with sublinear harmonic parameter growth, without human intervention. IBP-WF's factor reuse also results in positive transfer, yielding better accuracy, despite having fewer parameters than Const-WF.

4.3 Visualizations

Weight factor utilization Central to our method is the IBP prior that controls the growth of the number of factors and encourages the model to reuse factors. This controlled growth makes IBP-WF more efficient than independent models, while the reuse allows for positive knowledge transfer between tasks. The factor usage is visualized by plotting the expected factor scores $\mathbb{E}[\boldsymbol{\lambda}_t]$ for the first layer of a model trained on the Split CIFAR10 in Figure 4. One can clearly see the impact of using the IBP as regularization: early factors are prioritized in earlier tasks, and new factors are used with later tasks. We emphasize that the number of new parameters is not defined directly by some preset schedule, but rather is inferred from the data.

The sparsity induced by the IBP can also be seen. With each new task, an increasing number of factor scores have nonzero entries, as the model adapts the number of factors F based on the task objective. However, even for a later task, the probability of a factor being active remains high for only a few. As a result, each *draw* from the posterior tends to be sparse, regularized by the IBP to have α active factors in expectation. Another appealing aspect of using an IBP is that the *rate* of allocating a new factor decreases with tasks. Finally, following the "rich-get-richer" principle (here for "rich", or widely utilized, factors), the IBP encourages that factors are reused based on the total number of prior tasks using it.

CIFAR10 filters We also visualize the first layer convolutional representations for a model trained with IBP-WF on Split CIFAR10 (Figure 5). We observe an interesting property of the model: the feature maps in earlier tasks are similar compared to the diverse feature maps for later tasks. This can be attributed to early tasks using few factors due to the regularizing effect induced by the IBP on the rank of the weight filter.

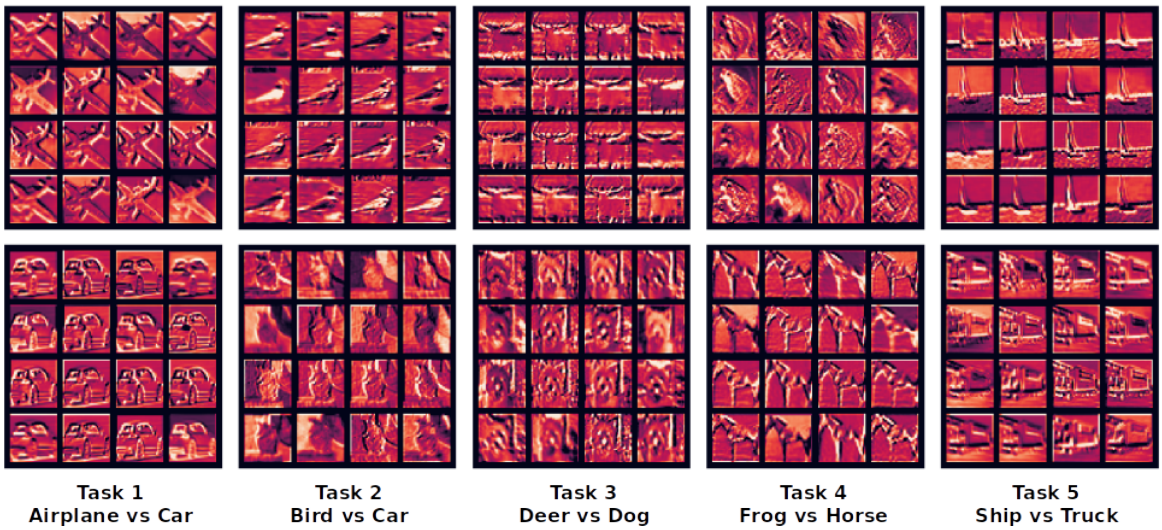


Figure 5: The first layer representations for each class in a trained IBP-WF ResNet-20. Each 4×4 grid shows the feature representations after convolution using 16 kernels in the first layer.

However, as the model observes more data, the filters become more diverse since the number of active factors increase, resulting in varied features maps. This shows that as the model sees more tasks, the complexity of the layer increases through the newly invoked filters.

5 Conclusions

An expansion-based approach combining a dictionary of weight factors with the IBP has been introduced, which we call IBP-WF. This synergy provides important characteristics within the context of continual learning, including knowledge reuse across tasks, data-driven model expansion, and catastrophic-forgetting mitigation. We also propose a simple and efficient task-inference scheme, utilizing feature statistics for each task and enabling operation in incremental class settings. A number of experiments on common continual-learning benchmarks show the effectiveness of IBP-WF. Ablation studies demonstrate the effectiveness of the IBP over linear expansion, and visualizations of the inferred factor scores and weights illustrate the regularization effects of our method. Notably, the motivation of IBP-WF is orthogonal to a number of other continual learning strategies, and combining some of these with IBP-WF is a promising direction for future work. For example, IBP-WF can readily be augmented with replay, and the Dirichlet process mixture model (as in Lee et al. (2020)) may be a natural Bayesian nonparametric alternative to our feature statistic method for inferring tasks.

Acknowledgements: This work was supported by the DARPA L2M Program.

References

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory Aware Synapses: Learning What (not) to Forget. *European Conference on Computer Vision*, 2018.

Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-Free Continual Learning. *Computer Vision and Pattern Recognition*, 2019.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. *International Conference on Machine Learning*, 2015.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12 (Jul):2121–2159, 2011.

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

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

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Zoubin Ghahramani and Thomas L Griffiths. Infinite Latent Feature Models and the Indian Buffet Process. *Neural Information Processing Systems*, 2006.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. *Neural Information Processing Systems*, 2014.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013.

I. S. Gradshteyn and I. M. Ryzhik. *Table of integrals, series, and products*. Elsevier/Academic Press, Amsterdam, seventh edition, 2007.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *Neural Information Processing Systems*, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, Picking and Growing for Unforgetting Continual Learning. *Neural Information Processing Systems*, 2019.

Samuel Kessler, Vu Nguyen, Stefan Zohren, and Stephen Roberts. Hierarchical indian buffet neural networks for bayesian continual learning. *arXiv*, pp. arXiv–1912, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, 2017.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.

Abhishek Kumar, Sunabha Chatterjee, and Piyush Rai. Nonparametric Bayesian Structure Adaptation for Continual Learning. *arXiv preprint arXiv:1912.03624*, 2019.

Ponnambalam Kumaraswamy. A Generalized Probability Density Function for Double-Bounded Random Processes. *Journal of Hydrology*, 1980.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks. *Neural Information Processing Systems*, 2018.

Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning. *International Conference on Learning Representations*, 2020.

Zhizhong Li and Derek Hoiem. Learning Without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

Kevin J Liang, Geert Heilmann, Christopher Gregory, Souleymane O Diallo, David Carlson, Gregory P Spell, John B Sigman, Kris Roe, and Lawrence Carin. Automatic Threat Recognition of Prohibited Items at Aviation Checkpoint with X-ray Imaging: A Deep Learning Approach. *SPIE Anomaly Detection and Imaging with X-Rays (ADIX) III*, 2018a.

Kevin J Liang, Chunyuan Li, Guoyin Wang, and Lawrence Carin. Generative Adversarial Network Training is a Continual Learning Problem. *arXiv preprint arXiv:1811.11083*, 2018b.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. *Neural Information Processing Systems*, 2017.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *International Conference on Learning Representations*, 2017.

Michael McCloskey and Neal J Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *The Psychology of Learning and Motivation*, 1989.

Joseph Victor Michalowicz, Jonathan M. Nichols, and Frank Bucholtz. *Handbook of Differential Entropy*. Chapman and Hall/CRC, 2013. ISBN 1466583169.

Eric Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders, 2016.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

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

German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 2019.

Roger Ratcliff. Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions. *Psychology Review*, 1990.

Dezső Ribli, Anna Horváth, Zsuzsa Unger, Péter Pollner, and István Csabai. Detecting and Classifying Lesions in Mammograms with Deep Learning. *Scientific reports*, 8 (1):1–7, 2018.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting. *Neural Information Processing Systems*, 2018.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience Replay for Continual Learning. *Neural Information Processing Systems*, 2019.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016.

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A Scalable Framework for Continual Learning. *International Conference on Machine Learning*, 2018.

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

Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.

Yee Whye Teh, Dilan Grür, and Zoubin Ghahramani. Stick-breaking construction for the indian buffet process. *Artificial Intelligence and Statistics*, 2007.

Gido M van de Ven and Andreas S Tolias. Generative Replay with Feedback Connections as a General Strategy for Continual Learning. *arXiv preprint arXiv:1809.10635*, 2018.

Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *International Conference on Learning Representations*, 2021.

Ronald J Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 1992.

Ju Xu and Zhanxing Zhu. Reinforced Continual Learning. *Neural Information Processing Systems*, 2018.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong Learning with Dynamically Dxpandable Networks. *International Conference on Learning Representations*, 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. *International Conference on Machine Learning*, 2017.

Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-Tuning: Network Adaptation via Additive Side Networks. *arXiv preprint arXiv:1912.13503*, 2019.

Supplementary Material

A Review of Indian Buffet Process (IBP) and its connection to IBP-WF

Indian Buffet Process The Indian Buffet Process (IBP) is a stochastic process defining a probability distribution over a binary matrix (Z) with finite rows (T) and an unbounded number of columns ($K \to \infty$). The binary matrix can be interpreted as an assignment matrix, with the rows representing a finite number of objects (sometimes referred to as the "customers") and the columns representing an unbounded number of features (referred to as the "dishes"), where $z_{ik} = 1$ if an object i has the k^{th} feature or otherwise $z_{ik} = 0$. Consider $z_{ik}|\mu_k \sim \text{Bernoulli}(\mu_k)$, where $\mu_k \sim \text{Beta}(\frac{\alpha}{K}, 1)$ is the prior probability that the feature k is active and α is the strength parameter. If we marginalize μ_k and take the limit $K \to \infty$, we get the IBP. The IBP is often described using a culinary metaphor: supposing that there is a restaurant that serves a buffet with infinitely many dishes, then we can describe the IBP as follows:

- The first customer enters the restaurant and takes a serving from Poisson(α) dishes.
- Each t^{th} customer that follows moves along the buffet sampling dishes based on their popularity; the customer takes a serving of the k^{th} dish with the probability $\frac{m_k}{t}$, where m_k is the number of customers who have previously taken dish k. The customer then tries Poisson($\frac{\alpha}{t}$) number of *new* dishes.

A sample from the above process can be summarized with the binary matrix Z, where z_{ik} represents whether the i^{th} customer took a serving from dish k or not. For an in-depth view, we refer the reader to the comprehensive review by Ghahramani & Griffiths (2006).

Key properties of IBP: (1) The total number of dishes chosen can grow arbitrarily. (2) The likelihood of adding new dishes is given by Poisson($\frac{\alpha}{t}$). Thus, as t increases, the tendency to add new dishes decreases. (3) As the number of customers increase, the tendency of a new customer to reuse previously served dishes increases.

Stick-breaking Construction for IBP Teh et al. (2007) proposed an alternative representation for the IBP where the feature probabilities (μ_k) are not integrated. Let the ordered sequence of $\{\mu_k\}_{k=1}^K$ be $\pi_1 > \pi_2 > ... > \pi_K$ such that $\pi_k = \mu_l$, where $1 \leq \{k, l\} \leq K$. We can construct $\{\pi_k\}_{k=1}^K$ as follows:

$$v_k \overset{i.i.d.}{\sim} \text{Beta}(\alpha, 1) \qquad \pi_k = \prod_{j=1}^{k} v_l \tag{16}$$

In the limit $K \to \infty$, the above is referred to as the stick-breaking construction of IBP. The stick-breaking construction and the standard IBP representation are different representations of the same nonparametric object (see Section 3 in Teh et al. (2007) for the proof). In practice, we use a truncated version of the stick-breaking process, where a large enough K is chosen.

Connection with the proposed IBP-WF for continual learning Recall the filter construction in the proposed weight factor dictionary learning:

$$W_t = \sum_{k=1}^{F} \lambda_t \left(w_{a,k} \otimes w_{b,k}\right), \qquad \boldsymbol{\lambda}_t = \mathbf{r}_t \odot \mathbf{b}_t \tag{17}$$

where F represents the number of active factors, t represents the task and k represents the row and column of the corresponding W_a and W_b matrices. For brevity, here we suppress the superscript ℓ from equation (2) denoting the layer. The binary vector b_t is generated using the stick-breaking construction of IBP. Following the same culinary metaphor as in the standard IBP, the weight factor $(w_{a.k} \otimes w_{b,k})$ and the task t are analogous to the "dish" and "customer" respectively. Central to our setup is the growth of F as the model encounters new tasks; IBP-WF inherits the properties of IBP described above: as more tasks are seen, the rate of adding new weight factors decreases while the likelihood of reusing previously learned factors simultaneously increases.

B Kullback-Leibler (KL) Divergence Derivation

Nalisnick & Smyth (2016) gave an approximate form for the KL divergence between Kumaraswamy and Beta distributions. We apply online variational inference for v, which requires the KL divergence between two Kumaraswamy distributions (See online inference for $\boldsymbol{v}_t^\ell$ in Appendix C). Here we derive the analytical form to approximate the KL divergence between two Kumaraswamy distributions q and p.

$$\mathrm{KL}\left(q_v\left(a,b\right)||p_v\left(\alpha,\beta\right)\right) = \mathbb{E}_{q_v}\left[\log\frac{q_v\left(a,b\right)}{p_v\left(\alpha,\beta\right)}\right] \tag{18}$$

where $q_v\left(a,b\right) = abv^{a-1}\left(1-v^a\right)^{b-1}$ and $p_v\left(\alpha,\beta\right) = \alpha\beta v^{\alpha-1}\left(1-v^\alpha\right)^{\beta-1}$.

$$\mathrm{KL}\left(q_v\left(a,b\right)||p_v\left(\alpha,\beta\right)\right) = \underbrace{\mathbb{E}_{q_v}\left[\log q_v\left(a,b\right)\right]}_{\mathscr{T}1} - \underbrace{\mathbb{E}_{q_v}\left[\log p_v\left(\alpha,\beta\right)\right]}_{\mathscr{T}2} \tag{19}$$

where the first term is the Kumaraswamy entropy (Michalowicz et al., 2013):

$$\mathscr{T}1 = \log ab + \frac{a-1}{a}\left(-\gamma - \Psi(b) - \frac{1}{b}\right) - \frac{b-1}{b} \tag{20}$$

where γ is Euler's constant and Ψ is the Digamma function. For the second term, we write the expectation as:

$$\begin{aligned}\mathscr{T}2 &= \mathbb{E}_{q_v}\log\left(\alpha\beta v^{\alpha-1}\left(1-v^\alpha\right)^{\beta-1}\right) \\ &= \mathbb{E}_{q_v}\left[\log\alpha\beta + (\alpha-1)\log v + (\beta-1)\log\left(1-v^\alpha\right)\right] \\ &= \log\alpha\beta + (\alpha-1)\,\mathbb{E}_{q_v}\log v + (\beta-1)\,\mathbb{E}_{q_v}\log\left(1-v^\alpha\right)\end{aligned} \tag{21}$$

In the above equation, the expectation of the log term can be computed using Gradshteyn & Ryzhik (2007) (4.253):

$$\mathbb{E}_{q_v}\log v = \frac{1}{a}\left(-\gamma - \Psi(b) - \frac{1}{b}\right) \tag{22}$$

The third term involves taking the expectation of $\log\left(1-v^\alpha\right)$ which can approximated with a Taylor series:

$$\log\left(1-v^\alpha\right) = -\sum_{m=1}^{\infty}\frac{1}{m}v^{m\alpha} \tag{23}$$

Note that the infinite sum in (23) converges since $0 < v < 1$. From the monotone convergence theorem, we can take the expectation inside the sum:

$$\begin{aligned}\mathbb{E}_{q_v}\left[\log\left(1-v^\alpha\right)\right] &= -\sum_{m=1}^{\infty}\frac{1}{m}\mathbb{E}_{q_v}v^{m\alpha} \\ &= -\sum_{m=1}^{\infty}\frac{b}{m}\,\mathrm{B}\left(\frac{m\alpha}{a}+1,b\right) \\ &= -\sum_{m=1}^{\infty}\frac{\alpha b}{m\alpha+ab}\,\mathrm{B}\left(\frac{m\alpha}{a},b\right)\end{aligned} \tag{24}$$

where $\mathrm{B}(.,.)$ is the beta function and $b\,\mathrm{B}\left(\frac{m\alpha}{a}+1,b\right)$ is the $(m\,\alpha)^{\text{th}}$ moment of the Kumaraswamy distribution with parameters a and b. As the low-order moments dominate the infinite sum, we only use the first 10 terms to approximate (24) in our experiments. Using (20) and (21) we have:

$$\mathrm{KL}\left(q_v\left(a,b\right)||p_v\left(\alpha,\beta\right)\right) = \log\frac{ab}{\alpha\beta} - \frac{b-1}{b} + \frac{a-\alpha}{a}\left(-\gamma-\Psi(b)-\frac{1}{b}\right) + \sum_{m=1}^{\infty}\frac{\alpha b\,(\beta-1)}{m\alpha+ab}\,\mathrm{B}\left(\frac{m\alpha}{a},b\right) \tag{25}$$

C Inference

Recall that to determine which factors should be active for a particular task t, we perform variational inference to infer the posterior of parameters $\theta_t = \{\mathbf{b}_t^\ell, \mathbf{v}_t^\ell\}_{\ell=1}^L$. The following variational distributions were used:

$$q(\theta_t^\ell) = q(\mathbf{b}_t^\ell)q(\mathbf{v}_t^\ell) \tag{26}$$

$$\mathbf{b}_t^\ell \sim \text{Bernoulli}(\boldsymbol{\pi}_t^\ell) \tag{27}$$

$$\mathbf{v}_t^\ell \sim \text{Kumar}(\mathbf{c}_t^\ell, \mathbf{d}_t^\ell) \tag{28}$$

The objective for each task is to maximize the variational bound:

$$\mathcal{L}_t = \sum_{n=1}^{N_t} \mathbb{E}_q \log p\left(y_t^{(n)} | \theta_t, x_t^{(n)}, \mathbf{W}_a, \mathbf{W}_b, \mathbf{r}_t\right) - \text{KL}\left(q\left(\theta_t\right) || p\left(\theta_t\right)\right) \tag{29}$$

where N_t is the number of training examples in task t. We use the mean-field approximation, so the second term can be expressed as

$$\text{KL}\left(q(\theta_t)||p\left(\theta_t\right)\right) = \text{KL}\left(q(\mathbf{b}_t^\ell)||p(\mathbf{b}_t^\ell|\mathbf{v}_t^\ell)\right) + \text{KL}\left(q(\mathbf{v}_t^\ell)||p(\mathbf{v}_t^\ell)\right) \tag{30}$$

VCL (Nguyen et al., 2018) addresses catastrophic forgetting using online inference, *i.e.*, the posterior inferred from the most recent task is used as a prior for the incoming task. However, more recent work (Farquhar & Gal, 2018, 2019) suggests that online inference often does not succeed in mitigating catastrophic forgetting in realistic continual learning settings, as methods based solely on online inference rely on the prior capturing everything learned on all previous tasks. Thus in (30), instead of performing online inference for $\{\mathbf{b}_t^\ell, \mathbf{v}_t^\ell\}$, we only apply online inference for $\boldsymbol{v}_t^\ell$, while learning task-specific parameters $\{\mathbf{r}_t^\ell, \mathbf{b}_t^\ell\}$.

Inference for $\mathbf{v}_t^\ell$: Starting with the first task ($t = 1$), we initialize the prior $p(\mathbf{v}_1^\ell) = \text{Beta}(\alpha, 1)$ and learn the posterior $q(\mathbf{v}_1^\ell) = \text{Kumar}(\mathbf{c}_1^\ell, \mathbf{d}_1^\ell)$ using Bayes by Backprop (Blundell et al., 2015). Note that $\text{Beta}(\alpha, 1)$ has the same density function as $\text{Kumar}(\alpha, 1)$. For all the following tasks, the prior $p(\mathbf{v}_t^\ell) = q(\mathbf{v}_{t-1}^\ell)$ and the posterior $q(\mathbf{v}_t^\ell) = \text{Kumar}(\mathbf{c}_t^\ell, \mathbf{d}_t^\ell)$ is learned in the same way as in task 1. Note that we use mean-field approximation for the posterior: $q(\text{v}_{t,i}^\ell) = \text{Kumar}(c_{t,i}^\ell, d_{t,i}^\ell)$. We use (25) to compute the KL divergence between the posterior and the prior in (30).

Inference for $\mathbf{b}_t^\ell$: We use the $\text{BernoulliConcrete}_\lambda$ distribution (Maddison et al., 2017) as the soft approximation of the Bernoulli distribution for both the prior and the posterior. We fix $\lambda = 2/3$ for all our experiments. We employ the prior $p(\text{b}_{t,k}^\ell) = \text{BernoulliConcrete}_\lambda(\pi_{t,k}^\ell)$, where $\pi_{t,k}^\ell := \prod_{i=1}^{i=k} \text{v}_{t,i}^\ell$ and $\text{v}_{t,i}^\ell \sim q(\text{v}_{t,i}^\ell)$. The posterior is then $q(\text{b}_{t,k}^\ell) = \text{BernoulliConcrete}_\lambda(\bar{\pi}_{t,k}^\ell)$, where $\bar{\pi}_{t,k}^\ell$ is learned using Bayes by Backprop. We use the the KL divergence and reparameterization for the $\text{BernoulliConcrete}_\lambda$ as given by Maddison et al. (2017).

D Task Inference at Test Time

Recall from (13), we approximate $P(t|x)$ with $P(t|\phi(x))$, where:

$$P(t|x) \approx \frac{P(\phi(x)|t)P(t)}{\sum_{t'} P(\phi(x)|t')P(t')} = P(t|\phi(x)) \tag{31}$$

In the following section, we show that under a certain assumption (namely Assumption 1 in D.1), this approximation is exact with $P(t|x) = P(t|\phi(x))$. However, in practice this assumption may not hold without an explicit hard constraint; hence we consider (13) an approximation. Nevertheless, we feel it is important to show this connection. In section D.2, we show empirical results on employing task inference as described in (31) over commonly used continual learning benchmarks.

D.1 Proof

Let $\phi : \mathcal{X} \to \mathcal{S}$ be the transformation function. We will assume ϕ is differentiable. For the transformation $\phi : \mathcal{X} \to \mathcal{S}$, and for a distribution P defined over $\mathcal{X}$, let P_ϕ be the distribution induced by ϕ over $\mathcal{S}$.

Assumption 1 *The transformation ϕ is a one-to-one function. Without loss of generality assume $\mathcal{S}$ to be the image of $\mathcal{X}$ under ϕ with $\psi : \mathcal{S} \to \mathcal{X}$ to be the inverse of ϕ, such that $\psi(\phi(x)) = x$.*

Lemma 1 *(Remark, main text). If Assumption 1 holds, then $P(t|\phi(x)) = P(t|x)\ \forall\, x \in \mathcal{X}$, $t \in \{1, 2, ...T\}$.*

Proof. Let $M_\phi(s) = \left|\det \frac{\partial \phi^{-1}(s)}{\partial s}\right|$ be the absolute of the determinant of the Jacobian of $\psi(s)$. Consider $s = \phi(x)$ and $x = \psi(s)$.

$$P_\phi(t|s) = \frac{P_\phi(t, s)}{P_\phi(s)} \tag{32}$$

$$\overset{(a)}{=} \frac{P(t, \psi(s))\, M_\phi(s)}{P(\psi(s))\, M_\phi(s)} \tag{33}$$

$$= \frac{P(t, x)}{P(x)} = P(t|x) \tag{34}$$

where (a) follows from the change of variable formula. Note that $M_\phi(s)$ can also be written as $\left|\det \frac{\partial \phi(x)}{\partial x}\right|^{-1}$ if ψ is continuously differentiable (it is not however a requirement for Lemma 1).

D.2 Quantitative Results for Task Inference and Visualizations

The procedure introduced in Section 2.4 is used for identifying the task identity during evaluation when it is otherwise unavailable. Thus, incremental class performance is highly dependent on task inference accuracy. We report the task inference accuracy in Table 3.

Additionally, we visualize with a t-SNE plot the distribution of the intermediate features ϕ from 10 tasks of permuted MNIST in Figure 6. The features across tasks are noticeably clustered, which allows our task inference method to infer task identity from simple feature statistics. While the accuracy for CIFAR10 is much lower than for MNIST, this is partially attributable to the inherent challenge of sequentially learned task inference for CIFAR10: CIFAR10 proves challenging for the generative models commonly used by replay methods for task inference as well. For example, we find that learning a separate VAE (Kingma & Welling, 2013)[1] for each task resulted in a task inference accuracy of 41%. In general, Nalisnick et al. (2018) showed that the density estimates from generative models can lead to poor Out-of-Distribution detection. However, a comprehensive study is required for further analysis. We leave further exploration of generative models for task inference for future work.

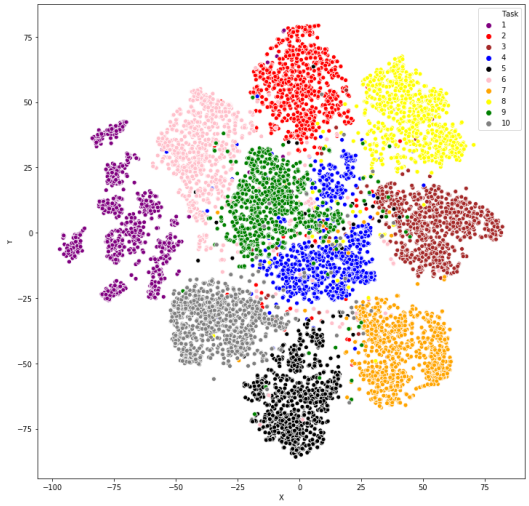


Figure 6: The t-SNE plot for the intermediate features ϕ of 10 tasks from the permuted MNIST benchmark.

Table 3: The task inference accuracy using (31).

Split MNIST	Permuted MNIST	CIFAR10
92.63 ± 0.12	99.98 ± 0.01	43.62 ± 0.16

[1]For VAE task inference, we used an encoder with layers 3(input)-32(conv)-64(conv)-128(fc), with a decoder that had a deconvolution architecture symmetrical to the encoder. We used the ELBO to approximate $P(x|t)$.

E Uncertainty Estimation

A desired behaviour from a model is to return the uncertainty (or confidence) associated each prediction. Neural networks are prone to have high confidence when the input lies outside of the training distribution. For such inputs, we want our model to have high uncertainty (or low confidence) associated with the predictions. Unlike neural networks trained as point-estimates (using MLE/MAP), Bayesian neural networks provide a natural framework to estimate uncertainty associated with the prediction. We estimate the uncertainty in a continual setting for both incremental task and incremental class settings. Note that non-Bayesian continual learning methods do not have principled method to estimate uncertainty. Our estimate of uncertainty is based on the predictive entropy defined as:

$$\mathbb{H}\left[y^*|x^*, \mathcal{D}_{train}\right] = -\sum_c \Big(p(y^* = c|x^*, \mathcal{D}_{train}) \log p(y^* = c|x^*, \mathcal{D}_{train})\Big) \tag{35}$$

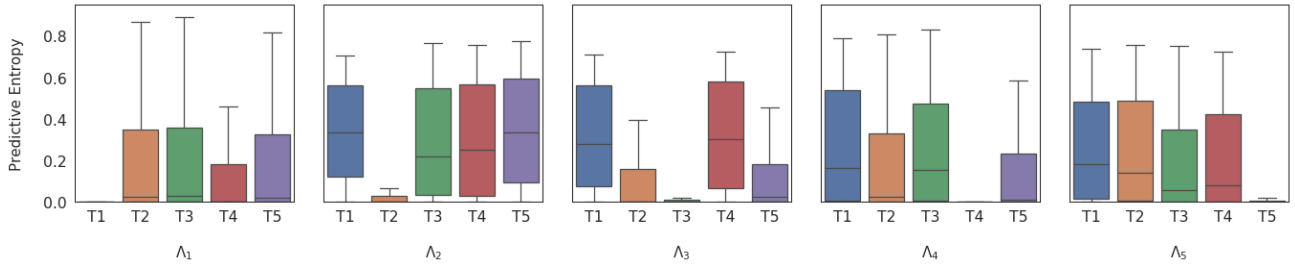


Figure 7: (Section E.1) Uncertainty in the incremental task setting for Split MNIST dataset. Each of the 5 plots depicts the uncertainxty of the test sets when task-specific parameter Λ_t is used. The y-axis denotes the uncertainty (as the predictive entropy), and x-axis denotes the test sets ($\mathcal{T}_1$ through $\mathcal{T}_5$).

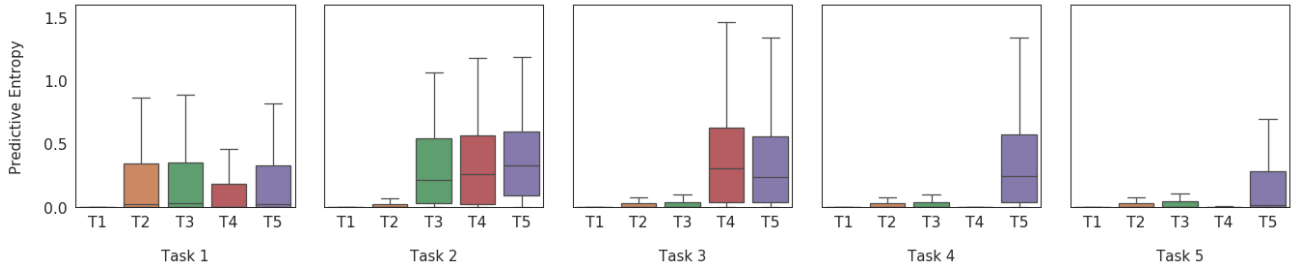


Figure 8: (Section E.2) Uncertainty in the incremental class setting for Split MNIST dataset. We compute the uncertainty of the test sets after training on each task in the sequence. The y-axis denotes the uncertainty (as the predictive entropy), and x-axis denotes the test sets ($\mathcal{T}_1$ through $\mathcal{T}_5$) for each task. Since we do not know the task (and the corresponding Λ_t), the predictive entropy is computed by marginalizing over all tasks. All the seen classes have low uncertainty compared to unseen ones.

E.1 Incremental Task Learning:

Recall that in incremental task learning, we know the task identity at test time. Hence, we compute the predictive distribution by doing a forward pass using the task-specific parameters in IBP-WF; we can write $p(y^* = c|x^*) = p(y^* = c|x^*, t^*)$, where t^* is the associated task-identity with the input x^* during testing. Following Gal (2016), we approximate the predictive distribution by using an ensemble of M neural networks sampled from

the posterior distribution:

$$p(y^* = c|t = t^*, x^*) = \underbrace{\frac{1}{M}\sum_{m=1}^{M} p(y^* = c|x^*; \theta_{t^*}^{(m)})}_{\rho_{t^*,c}^M}, \text{ where } \theta_{t^*}^{(m)} \sim q(\theta_{t^*}) \tag{36}$$

where $\theta^{(1)}...\theta^{(M)}$ are M samples drawn from $q(\theta_{t^*})$. Using this we can compute a biased estimate[2] of the predictive entropy as follows:

$$\mathbb{H}[y^*|x^*] = -\sum_c p(y^* = c|x^*) \log p(y^* = c|x^*) \tag{37}$$

$$= -\sum_c p(y^* = c|x^*, t^*) \log p(y^* = c|x^*, t^*) \tag{38}$$

$$= -\sum_c \left(\frac{1}{M}\sum_{m=1}^{M} p(y^* = c|x^*; \theta_{t^*}^{(m)})\right) \log \left(\frac{1}{M}\sum_{m=1}^{M} p(y^* = c|x^*; \theta_{t^*}^{(m)})\right) \tag{39}$$

$$= -\sum_c \left(\rho_{t^*,c}^M\right) \log \rho_{t^*,c}^M \tag{40}$$

Figure 7 shows the uncertainty estimates for the test sets in the Split MNIST dataset. We denote the test set for a task $t \in \{1...5\}$ as $\mathcal{T}_t$. As it can be seen in Figure 7, given a task-identity t, the uncertainty for the test set $\mathcal{T}_t$ when used with parameters Λ_t is significantly smaller compared to the uncertainty of test sets $\{\mathcal{T}_t' \,|\, t' \neq t\}$. One application of computing uncertainties would be an out-of-distribution test in the continual learning setting. However, we leave exploring such extensions for future work. We use $M = 100$ to compute the uncertainty.

E.2 Incremental Class Learning:

For the incremental class setting, we do not have access to the task-identity of a given test point. We use the task inference mechanism from Section 2.4 in the main paper. To infer the predictive distribution, we marginalize over the task-identities:

$$p(y^* = c|x^*) = \sum_{t'} p(y^* = c, t = t'|x^*) \tag{41}$$

$$= \sum_{t'} p(y^* = c|t = t', x^*) p(t = t'|x^*) \tag{42}$$

$$\approx \sum_{t'} p(y^* = c|t = t', x^*) \frac{P(\phi(x)|t')P(t')}{\sum_t P(\phi(x)|t)P(t)} \tag{43}$$

$$= \sum_{t'} \rho_{t',c}^M \, P(t'|\phi(x)) \tag{44}$$

$$\mathbb{H}[y^*|x^*] = -\sum_c \left(\left(\sum_{t'} \rho_{t',c}^M \, P(t'|\phi(x))\right) \log \left(\sum_{t'} \rho_{t',c}^M \, P(t'|\phi(x))\right)\right) \tag{45}$$

We use (45) to estimate the uncertainty in the incremental class continual setting for the Split MNIST dataset. Figure 8 shows the uncertainty of test sets after training on each task. As shown, initially when the model is trained on the first task, the uncertainty of $\mathcal{T}_2$-$\mathcal{T}_5$ is higher than the uncertainty of $\mathcal{T}_1$. As the training progresses, the uncertainty of the corresponding task decreases while still maintaining a low estimate of the uncertainty of the test sets for the previous tasks. This provides further evidence that our proposed method IBP-WF mitigates catastrophic forgetting. We use $M = 100$ to compute the uncertainty.

[2]The estimate is biased since $\mathbb{H}[.]$ is a non-linear function. The bias will decrease as M increases.

F Baselines

We compare IBP-WF with a number of other approaches outlined as follows:

Fine-tuning The model is trained by a stochastic gradient descent algorithm, seeing each task in sequence. At the conclusion of each task, the "final" trained model for a task is used as the initialization for the next task. This represents the naive approach to training on sequential task data, where catastrophic forgetting was first recognized. We compare against models trained by vanilla stochastic gradient descent (**SGD**) with constant learning rate, as well as by adaptive learning rate methods **Adam** (Kingma & Ba, 2014) and **Adagrad** (Duchi et al., 2011).

Regularization Methods Recognizing that training on a new task may result in a model's parameters moving away from an optimum for a previous task, a number of continual learning strategies attempt to constrain the model parameters from drifting too far while learning a new task. A simple way to do so is to apply an L_2 loss on the model parameters' distance from previous task solutions. **EWC** (Kirkpatrick et al., 2017) refines this by weighting the L_2 by parameter importance, using the Fisher Information; **Online EWC** (Schwarz et al., 2018; Liang et al., 2018b) uses an online version that provides better scaling. **SI** (Zenke et al., 2017) also weights the L_2 regularization by importance, with the importance weighting instead coming from the amount a parameter contributed to reducing the loss over its trajectory. **MAS** (Aljundi et al., 2018) computes parameter importance as well, but with respect to the model output rather than the loss. **LwF** (Li & Hoiem, 2017) leverages knowledge distillation (Hinton et al., 2015) principles, using previous model outputs as additional training objectives. **VCL** uses Bayesian neural networks, using the posterior of the previous task as the prior for the next. We also compare against a recent expansion method called **IBNN** (Kessler et al., 2020) that uses IBP to adapt the structure of a Bayesian neural network. The accuracy for IBNN is taken directly from Kessler et al. (2020).[3]

Replay Methods As catastrophic forgetting can be attributed to not seeing previous parts of the data distribution, another class of methods employ experience replay: refreshing the model on old tasks while learning new ones. **Naive Rehearsal** accomplishes this by keeping examples from old tasks in a buffer and assembling them into "replay" minibatches. This runs the risk of overfitting the samples in the buffer, so **GEM** (Lopez-Paz & Ranzato, 2017) proposes instead using these as inequality restraints: the model should not increase the loss on saved samples. These saved samples can also used for re-training or fine-tuning the model, which **VCL** (Nguyen et al., 2018) does with its coresets. Regardless of how stored samples are used, however, in certain settings, data is private (Ribli et al., 2018) or classified (Liang et al., 2018a), and keeping data may be considered as violating continual learning criteria. As an alternative, **DGR** (Shin et al., 2017) and **RtF** (van de Ven & Tolias, 2018) propose generative models (Goodfellow et al., 2014) as a source of replay. Such approaches avoid carrying around older data, but require learning (and storing) generative models for each task, which may need to be quite large depending on the complexity of the dataset.

We use the codebase from (Hsu et al., 2018; van de Ven & Tolias, 2019) as our continual learning "sandbox." Best efforts were made to keep the model capacity consistent in all methods for a fair comparison.

Table 4: The average accuracy of seen tasks after learning on a sequence of tasks using a MLP.

Method	Split MNIST		Permuted MNIST	
	Incremental Task	Incremental Class	Incremental Task	Incremental Class
SGD	96.95 ± 0.46	19.46 ± 0.04	90.54 ± 0.03	8.46 ± 0.36
Adam	95.18 ± 2.64	19.71 ± 0.08	91.70 ± 1.89	16.13 ± 0.71
L_2	98.32 ± 0.73	22.52 ± 1.08	94.01 ± 0.27	16.43 ± 0.63
Online EWC	99.09 ± 0.12	19.77 ± 0.04	93.62 ± 0.25	42.40 ± 2.68
IBNN	95.30 ± 2.00	85.50 ± 3.20	95.6[3] ± 0.20	93.8[3] ± 0.30
IBP-WF (Ours)	**99.69** ± 0.05	92.40 ± 0.15	**97.52** ± 0.06	**97.50** ± 0.06

G Experiment Setup

We describe the experimental configuration used:

[3]The IBNN performance on permMNIST is based on only 5 tasks, whereas other methods in Table 4 use 10 tasks.

G.1 Split MNIST

Following (Hsu et al., 2018), we use the standard train/test split, with 60K training images (6K images per digit) and 10K test images (1K images per digit). Standard normalization of the images was the only preprocessing done, without any data augmentation strategies used for any of the algorithms.

Baselines: All baseline methods use the same neural network architecture: a MLP with two hidden layers of 400 nodes each, followed by a softmax layer. For GEM and naive rehearsal, a buffer of 400 images were saved to replay previous tasks. For DGR and RtF a 2-layer symmetric variational autoencoder(Kingma & Welling, 2013) was learned for each task. We used ReLU as the non-linearity in both the hidden layers. All the baseline models, except VCL, RtF and HIBNN, were trained for 10 epochs per task with a mini-batch size of 128 with Adam (Kingma & Ba, 2014) optimizer ($\beta_1 = 0.9, \beta_2 = 0.999, lr = 0.001$) as the default unless explicitly stated. VCL was trained for 50 epochs. The results for RtF were taken from the original paper by van de Ven & Tolias (2018), which was trained for 2000 steps with a batch size of 128. Note that RtF has twice the number of parameters compared to IBP-WF. The results for HIBNN in table 4 were taken from Kessler et al. (2020), which was trained for 200 epochs. For EWC online, EWC, SI, GEM and MAS, the regularization coefficient was set to 400, 100, 300, 0.5 and 1.0 respectively.

IBP-WF: IBP-WF used the same neural architecture as the baselines, except there was only a single hidden layer. The prior parameter for IBP was set to $\alpha = 100$. The model is expanded for 10 epochs (using equation (10) in the main paper) with a learning rate of 0.001 and fine-tuned with a fixed number of factors for 5 epochs. A mini-batch size of 32 was used. We used the stick-breaking construction for IBP, which was truncated at $K = 400$, *i.e.* the total budget on the number of allowed factors was 400.

G.2 Permuted MNIST

We use the standard train/test split of the MNIST dataset. Each task consists of the same 10-way digit classification, but with the pixels of the entire MNIST dataset randomly permuted in a consistent manner. We generate 10 such tasks using 10 random permutations in our experiments.

Baselines: All the baseline methods use the same neural network architecture: a MLP with two hidden layers of 1000 nodes each, followed by a softmax layer. We used ReLU as the non-linearity in both the hidden layers. For GEM and naive rehearsal, a buffer 1.1K images were saved to replay previous tasks. For DGR and RtF a 2-layer symmetric variational autoencoder was learned for each task. All the baseline models, except RtF and VCL, were trained for 15 epochs per task with a mini-batch size of 128 with Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.999, lr = 0.001$) as the default unless explicitly stated. For VCL, the model was trained for 100 epochs. The results for RtF were taken from van de Ven & Tolias (2018), which was trained for 5000 iterations. For EWC online, EWC, SI, GEM and MAS, the regularization coefficient was set to 500, 500, 1.0, 0.5 and 0.01 respectively.

IBP-WF IBP-WF used the same neural architecture as the baselines. The prior parameter was set to $\alpha = 700$. We train for 15 epochs for each task using IBP (using equation (10) in the main paper) with a mini-batch size of 64. The model was then fine-tuned for 5 epochs with a fixed number of factors. The stick-breaking process for IBP was truncated at $K = 1000$ for both the hidden layers.

G.3 CIFAR10

We split the CIFAR10 (Krizhevsky, 2009) dataset into a sequence of 5 binary classification tasks. Similar to Split MNIST, this is a binary classification problem at test time in the incremental task setting, and 10-way classification in the incremental class setting.

Baselines: We use ResNet-20 (He et al., 2016) for all the baselines. We used standard data augmentation methods (random crop, horizontal flips and standard normalization) while training. All the baselines models were trained for 160 epochs per task with a mini-batch size of 128. A learning rate of $lr = 0.001$ was used. For naive rehearsal, a buffer of 400 images were saved to replay previous tasks. For EWC online, EWC, and SI, the regularization coefficient was set to 3000, 100 and 2 respectively.

IBP-WF: We scale our IBP-WF method to ResNet-20 by factorizing convolutional layers. The training was carried out for 160 epochs with a learning rate of $lr = 0.001$ and a mini-batch size of 128. There was no fine-tuning done for this experiment. The truncation parameters for the stick-breaking process was set to 200 for all the layers. We used task-specific batch normalization parameters for our implementation. We set the IBP hyperparameter α to be 40 for all the convolutional layers and 32 for the final fully-connected layer.

H Ablation Studies

H.1 Selecting α

The hyperparameter α controls the behavior of the Indian Buffet Process prior, which for IBP-WF provides a regularization effect for both the number of active (nonzero) factors per task, as well as the expected rate at which new factors are added (expansion). Specifically, α *is* the prior's expected number of factors per task, and as such should be a value on the order of (but preferably less than) the rank of the weight matrix. We sweep α and plot overall final performance of IBP-WF on Split MNIST and Permuted MNIST in both incremental class and incremental task settings in Figure 9. We observe that excessively low values of α lead to poorer performance, as there are not enough factors to learn each task, but otherwise IBP-WF exhibits low sensitivity to α over a very wide range of values, showing relative robustness to α.

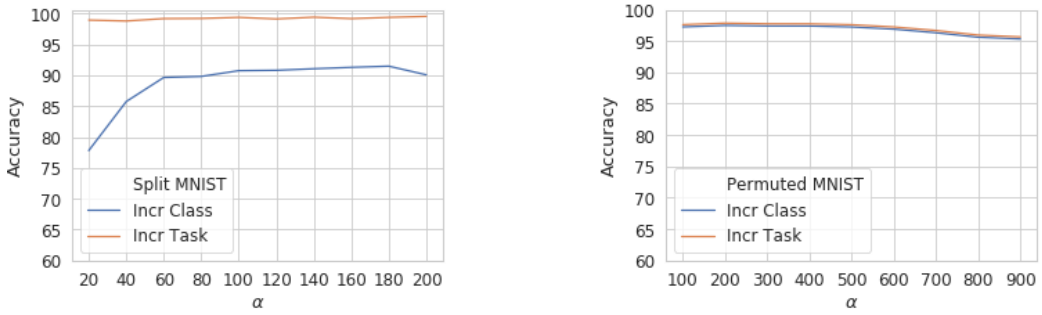


Figure 9: Ablation study on α.

H.2 Selecting κ

IBP-WF preserves past knowledge by selectively freezing weight factors that played a key role in previous tasks. We define this criterion as factors whose probability $\pi^{\ell}_{t,k}$ exceed a threshold κ. As with α, we sweep κ and plot IBP-WF's performance on Split MNIST and Permuted MNIST in both incremental class and incremental task settings in Figure 10. We observe a decline in performance if κ is set too high for incremental class learning in Split MNIST, as it likely leads to not enough factors being preserved, but overall there is a wide range of settings of κ that give good performance.

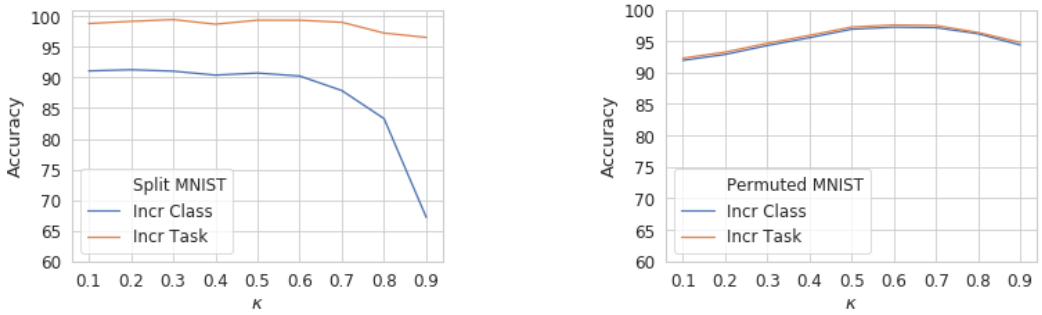


Figure 10: Ablation study on κ. We use $\kappa = 0.5$ for all experiments in the main text.

References

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory Aware Synapses: Learning What (not) to Forget. *European Conference on Computer Vision*, 2018.

Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-Free Continual Learning. *Computer Vision and Pattern Recognition*, 2019.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. *International Conference on Machine Learning*, 2015.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

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

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

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Zoubin Ghahramani and Thomas L Griffiths. Infinite Latent Feature Models and the Indian Buffet Process. *Neural Information Processing Systems*, 2006.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. *Neural Information Processing Systems*, 2014.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013.

I. S. Gradshteyn and I. M. Ryzhik. *Table of integrals, series, and products*. Elsevier/Academic Press, Amsterdam, seventh edition, 2007.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *Neural Information Processing Systems*, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, Picking and Growing for Unforgetting Continual Learning. *Neural Information Processing Systems*, 2019.

Samuel Kessler, Vu Nguyen, Stefan Zohren, and Stephen Roberts. Hierarchical indian buffet neural networks for bayesian continual learning. *arXiv*, pp. arXiv–1912, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, 2017.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.

Abhishek Kumar, Sunabha Chatterjee, and Piyush Rai. Nonparametric Bayesian Structure Adaptation for Continual Learning. *arXiv preprint arXiv:1912.03624*, 2019.

Ponnambalam Kumaraswamy. A Generalized Probability Density Function for Double-Bounded Random Processes. *Journal of Hydrology*, 1980.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks. *Neural Information Processing Systems*, 2018.

Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning. *International Conference on Learning Representations*, 2020.

Zhizhong Li and Derek Hoiem. Learning Without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

Kevin J Liang, Geert Heilmann, Christopher Gregory, Souleymane O Diallo, David Carlson, Gregory P Spell, John B Sigman, Kris Roe, and Lawrence Carin. Automatic Threat Recognition of Prohibited Items at Aviation Checkpoint with X-ray Imaging: A Deep Learning Approach. *SPIE Anomaly Detection and Imaging with X-Rays (ADIX) III*, 2018a.

Kevin J Liang, Chunyuan Li, Guoyin Wang, and Lawrence Carin. Generative Adversarial Network Training is a Continual Learning Problem. *arXiv preprint arXiv:1811.11083*, 2018b.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. *Neural Information Processing Systems*, 2017.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *International Conference on Learning Representations*, 2017.

Michael McCloskey and Neal J Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *The Psychology of Learning and Motivation*, 1989.

Joseph Victor Michalowicz, Jonathan M. Nichols, and Frank Bucholtz. *Handbook of Differential Entropy*. Chapman and Hall/CRC, 2013. ISBN 1466583169.

Eric Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders, 2016.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

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

German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 2019.

Roger Ratcliff. Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions. *Psychology Review*, 1990.

Dezső Ribli, Anna Horváth, Zsuzsa Unger, Péter Pollner, and István Csabai. Detecting and Classifying Lesions in Mammograms with Deep Learning. *Scientific reports*, 8(1):1–7, 2018.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting. *Neural Information Processing Systems*, 2018.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience Replay for Continual Learning. *Neural Information Processing Systems*, 2019.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016.

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A Scalable Framework for Continual Learning. *International Conference on Machine Learning*, 2018.

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

Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.

Yee Whye Teh, Dilan Grür, and Zoubin Ghahramani. Stick-breaking construction for the indian buffet process. *Artificial Intelligence and Statistics*, 2007.

Gido M van de Ven and Andreas S Tolias. Generative Replay with Feedback Connections as a General Strategy for Continual Learning. *arXiv preprint arXiv:1809.10635*, 2018.

Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *International Conference on Learning Representations*, 2021.

Ronald J Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 1992.

Ju Xu and Zhanxing Zhu. Reinforced Continual Learning. *Neural Information Processing Systems*, 2018.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong Learning with Dynamically Dxpandable Networks. *International Conference on Learning Representations*, 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. *International Conference on Machine Learning*, 2017.

Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-Tuning: Network Adaptation via Additive Side Networks. *arXiv preprint arXiv:1912.13503*, 2019.