Bayesian Nonparametric Weight Factorization for Continual Learning

Nikhil Mehta Kevin J Liang Lawrence Carin
Duke University
{nm208, kevin.liang, lcarin}@duke.edu

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 typically 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 us to scale the number of factors of each weight matrix to the complexity of the task, while the IBP prior imposes weight factor sparsity and encourages 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, primarily trained on independent and identically distributed (*i.i.d.*) data for a single task, has made enormous progress in recent years. However, when naively trained on tasks sequentially, without revisiting previous tasks, neural networks are known to suffer catastrophic forgetting [30, 35]: 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.

In machine learning, continual learning [34] has garnered considerable attention in recent years, and a number of desiderata have emerged. Models should be able to learn multiple tasks sequentially, with the eventual number and complexity of tasks initially unknown. Importantly, new tasks should be learned without catastrophically forgetting previous ones, ideally without having to keep data from previous tasks, which may be unavailable due to storage or data privacy concerns. Models should also be capable of positive transfer: previously learned tasks ought to help with the learning of new tasks. Knowledge transfer between tasks maximizes sample efficiency, particularly important in instances where data are scarce.

A number of methods (*e.g.* Rusu et al. [39], Zhang et al. [47], Lee et al. [24]) address the continual learning problem 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.

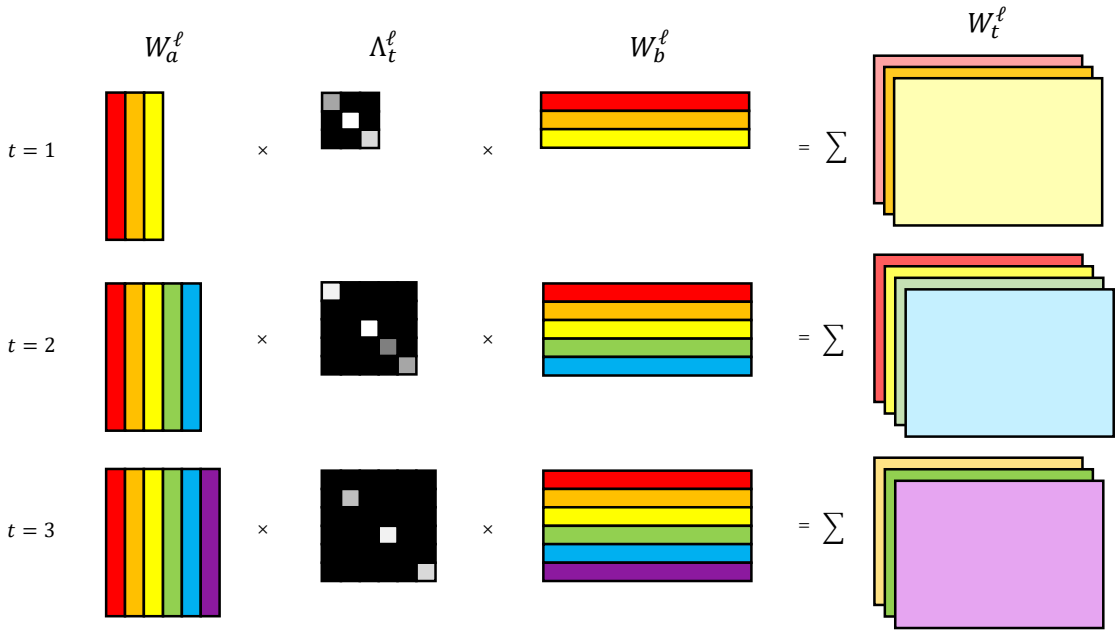

Figure 1: The generative story of our weight factorization proposed for continual learning. Matrices of weight factors W_a^ℓ and W_b^ℓ are shared across all tasks, and a task-specific sparse diagonal matrix Λ_t^ℓ specifies the active factors and their respective weighting for task t. The weighted sum of the active weight factors yields the weight matrix for a particular task. The number of factors (columns of W_a^ℓ and rows of W_b^ℓ) grows as needed with more tasks, with select factors being reused in future tasks. Best viewed in color.

Rather than rely on engineered heuristics, we choose to let the data dictate the model-expansion rate by taking a Bayesian nonparametric approach. Specifically, we adopt a framework coupling weight matrix factorization (WF) with the Indian Buffet Process (IBP) [9], which we call **IBP-WF**. An IBP-based formulation allows us to automatically scale the network as needed, even if the number or complexity of future tasks is initially unknown. An IBP prior also naturally encourages recycling of previously learned skills. This enables positive transfer between tasks, which other expansion methods tend to either ignore or deal with in a more *ad hoc* manner.

Our main contributions are as follows. (*i*) We introduce a weight factorization 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 ID in incremental class settings. (*iii*) The effectiveness of IBP-WF is demonstrated on a number of continual learning tasks, comparing against a number of other methods. We also visualize weight factor usage between tasks, confirming both sparsity and reuse of these factors.

2 Methods

2.1 Layer-wise Weight Factorization

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 $\{W_t^\ell\}_{\ell=1}^L$ for each task t. While $\{W_t^\ell\}_{\ell=1}^L$ could be learned separately for each task, such a model does not incorporate any knowledge reuse, and the total number of model parameters grows linearly with the number of tasks T. While immune to catastrophic forgetting, such an approach is inefficient in both computation and data.

Inspired by Taylor and Hinton [43], Song et al. [42], we instead propose factorizing the model's weights as follows:

$$W_t^\ell = W_a^\ell \Lambda_t^\ell W_b^\ell \tag{1}$$

$$\Lambda_t^\ell = \text{diag}(\boldsymbol{\lambda}_t^\ell) \tag{2}$$

where $W_a^\ell \in \mathbb{R}^{J \times F}$ and $W_b^\ell \in \mathbb{R}^{F \times M}$ 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 W_a^ℓ and rows of W_b^ℓ:

$$W_t^\ell = \sum_{k=1}^{F} \lambda_{t,k}^\ell \left(w_{a,k}^\ell \otimes w_{b,k}^\ell \right) \tag{3}$$

where $\otimes$ denotes the outer product, and the pair $w_{a,k}^\ell$ and $w_{b,k}^\ell$ is the k^{th} column and row of W_a^ℓ and W_b^ℓ, respectively. Under this construction, the pairs of corresponding columns of W_a^ℓ and rows of W_b^ℓ can be interpreted as 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.

We construct the factor scores $\boldsymbol{\lambda}_t^\ell$ as the following element-wise product:

$$\boldsymbol{\lambda}_t^\ell = \boldsymbol{r}_t^\ell \odot \boldsymbol{b}_t^\ell \tag{4}$$

where $\boldsymbol{b}_t^\ell \in \{0, 1\}^F$ indicates the *active* factors for task t and $\boldsymbol{r}_t^\ell \in \mathbb{R}^F$ specifies the corresponding factor *strength*. By imposing sparsity with $\boldsymbol{b}_t^\ell$, we prevent the model from being overly complex for any one task, instead concentrating skills for each task into specific factors.

Factorizing convolutional kernels While this weight factorization was formulated in (1) for fully connected layers, it can be extended to convolutional layers as well. 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 a 4D tensor factorization 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 factorization as in (1).

2.2 Indian Buffet Process for Weight Factors

Critical to the performance of the proposed layer-wise weight factorization is the number of factors F: too few and the model lacks sufficient expressivity to model all the tasks it encounters; 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 [39, 16, 47, 24]. 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 factorization, 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 the scores for the proposed weight factorization for each task and, by extension, the total number of factors F needed. In particular, we impose the stick-breaking construction of

the Indian Buffet Process (IBP) [9] as a prior for task-specific factor selection:

$$r_{t,k}^{\ell} \sim \mathcal{N}(0,1) \tag{5}$$

$$v_{t,i}^{\ell} \sim \text{Beta}(\alpha,1) \tag{6}$$

$$\pi_{t,k}^{\ell} = \prod_{i=1}^{k} v_{t,i}^{\ell} \tag{7}$$

$$b_{t,k}^{\ell} \sim \text{Bernoulli}(\pi_{t,k}^{\ell}) \tag{8}$$

where α is a hyperparameter controlling the expected number of nonzero factor scores, and $k = 1, 2, ...F$ indexes the factor. For global parameters (W_a^{ℓ} and W_b^{ℓ}), we use point estimates.

Leveraging the IBP in conjunction with weight factorization provides a number of advantages within the context of weight factorization and continual learning:

Dynamic control of F IBP allows the number of factors of F to be determined nonparametrically and dynamically, growing only as much necessary given the complexity of each individual incoming task. In other words, 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 to infinity, 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 $\boldsymbol{\pi}_t^{\ell}$ (Equation 7) actively encourages reuse of existing weight factors in a "rich get richer" scheme. This enables positive forward transfer from previous tasks to new ones, as recycling previously learned skills for future tasks is prioritized over creating new ones.

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 During test time, IBP weight factors (product of each column of W_a^{ℓ} and row of W_b^{ℓ}) can be pre-computed; given a task, the appropriate weight factors can be retrieved, weighted, and summed as needed to retrieve $\{W_t^{\ell}\}_{\ell=1}^{L}$. Imposing the IBP prior on the usage of factors induces a prior distribution of Poisson(α) on the number of active factors. This means that the expected value for 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. As we describe in the next section, our model effectively infers the active factors for a given task. This avoids one of the pitfalls of some other expansion methods (*e.g.,* Rusu et al. [39], Lee et al. [24], Kumar et al. [21]), whose inference-time computational costs do scale with T.

2.3 Variational Inference

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

$$q(\boldsymbol{\theta}_t^{\ell}) = q(\boldsymbol{r}_t^{\ell}) q(\boldsymbol{b}_t^{\ell}) q(\boldsymbol{v}_t^{\ell}) \tag{9}$$

$$\boldsymbol{r}_t^{\ell} \sim \mathcal{N}\left(\boldsymbol{\mu}_t^{\ell},\, \text{diag}(\boldsymbol{\sigma}_t^{\ell\,2})\right) \tag{10}$$

$$\boldsymbol{b}_t^{\ell} \sim \text{Bernoulli}(\boldsymbol{\pi}_t^{\ell}) \tag{11}$$

$$\boldsymbol{v}_t^{\ell} \sim \text{Kumaraswamy}(\boldsymbol{c}_t^{\ell}, \boldsymbol{d}_t^{\ell}) \tag{12}$$

We learn the variational parameters $\{\boldsymbol{\mu}_t^{\ell}, \boldsymbol{\sigma}_t^{\ell}, \boldsymbol{\pi}_t^{\ell}, \boldsymbol{c}_t^{\ell}, \boldsymbol{d}_t^{\ell}\}_{\ell=1}^{L}$ with Bayes by Backprop [4]. As the Beta distribution lacks a differentiable parameterization, we use the similar Kumaraswamy distribution [22] as the variational distribution

for v_t^ℓ. We also use a soft relaxation of the Bernoulli distribution [29] in (8) and (11) to allow backpropagation through discrete random variables.

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

$$\mathcal{L}_t = \underbrace{\sum_{n=1}^{N_t} \mathbb{E}_q \log p\left(y_t^{(n)} | \boldsymbol{\theta}_t, x_t^{(n)}\right)}_{\mathcal{L}} - \underbrace{\mathrm{KL}\left(q\left(\boldsymbol{\theta}_t\right) || p\left(\boldsymbol{\theta}_t\right)\right)}_{\mathcal{R}} \tag{13}$$

where N_t is the number of training examples in task t. Note that in (13) the first term ($\mathcal{L}$) provides label supervision and the second term ($\mathcal{R}$) regularizes the posterior not to stray too far from the IBP prior. We use a mean-field approximation, allowing us to expand the second term as follows:

$$\mathcal{R} = \sum_{\ell=1}^{L} \mathrm{KL}\left(q(\boldsymbol{r}_t^\ell) || p(\boldsymbol{r}_t^\ell)\right) + \mathrm{KL}\left(q(\boldsymbol{b}_t^\ell) || p(\boldsymbol{b}_t^\ell | \boldsymbol{v}_t^\ell)\right) + \mathrm{KL}\left(q(\boldsymbol{v}_t^\ell) || p(\boldsymbol{v}_t^\ell)\right) \tag{14}$$

Nguyen et al. [33] 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 [6, 7] 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 (14), instead of performing online inference for all parameters $\{\boldsymbol{r}_t^\ell, \boldsymbol{b}_t^\ell, \boldsymbol{v}_t^\ell\}$, we only apply online inference for $\boldsymbol{v}_t^\ell$ and learn task-specific parameters $\{\boldsymbol{r}_t^\ell, \boldsymbol{b}_t^\ell\}$ with the same prior for all tasks as defined in (5) and (8). Note that the prior for b_t^ℓ depends on the posterior of v_t^ℓ via (7)-(8), and thus online inference over $\boldsymbol{v}_t^\ell$ encourages the reuse of factors while having task-specific parameters allows the model to easily adapt to a new task. We derive an analytical approximation of KL divergence between two Kumaraswamy distributions. The derivation and more details on doing online inference with (13) and (14) are included in Appendices A and B.

Preserving knowledge As currently formulated, if all of W_a^ℓ and W_b^ℓ were free to move without constraint, then catastrophic forgetting may still occur. Indeed, there would be nothing preventing the model from "reusing" a factor from a previous task and then repurposing it entirely for a new one, resulting in the model losing the ability to do the former task. To prevent this, the weight factors (*i.e.,* the columns of W_a^ℓ and rows of W_b^ℓ) with factor probability $\pi_{t,k}^\ell > \kappa$ are locked[1] at the conclusion of a task. Weight factors below the threshold κ are left free to be modified for future tasks. Throughout our experiments, we set the threshold as $\kappa = 0.5$, but this can be adjusted given the situation's tolerance for forgetting. Alternatively, the various regularization continual learning methods (*e.g.,* Kirkpatrick et al. [19]) 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, it requires the task identity associated with each input at test time. The validity of this assumption has occasionally been questioned [6], and a few *task-free* continual learning approaches have been proposed that do not require this assumption [2, 24]. 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:

$$P(t = t'|x) = \frac{P(x|t = t')P(t = t')}{\sum_t P(x|t)P(t)} \tag{15}$$

However, using (15) requires learning a generative model $P(x|t = t')\ \forall t' \in \{1, 2...T\}$, which can be expensive in both computation and the number of parameters. To alleviate this issue, we propose a simple yet effective alternative:

[1] In code, this can be implemented with a stop gradient.

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 (15) by using $P(\phi(x)|t)$ as a surrogate for $P(x|t)$:

$$\begin{aligned} P(t = t'|x) &\approx \frac{P(\phi(x)|t = t')P(t = t')}{\sum_t P(\phi(x)|t)P(t)} \\ &= \frac{P(\phi_{t'}(x))P(t = t')}{\sum_t P(\phi_t(x))P(t)} \end{aligned} \tag{16}$$

where $\phi_t(x)$ is an intermediate layer of the neural network trained on task t, with the intuition that $\phi_t(x)$ captures the statistics of data belonging to the task t. Note that in our proposed weight factorization, ϕ_t is uniquely defined by the task-specific weights as shown in (1). Next, we assume that the trained neural network induces a Gaussian distribution at the given intermediate hidden layer: $P(\phi_t(x)) = \mathcal{N}(\phi_t(x)|\mu_t, \Sigma_t)$, where the parameters are the empirical estimates[2] using the training data:

$$\hat{\mu}_t = \frac{1}{N_t}\sum_{n=1}^{N_t}\phi_t^{(n)}, \quad \hat{\Sigma}_t = \frac{1}{N_t}\sum_{n=1}^{N_t}(\phi_t^{(n)} - \hat{\mu}_t)(\phi_t^{(n)} - \hat{\mu}_t)^T \tag{17}$$

where $\phi_t^{(n)}$ is the hidden layer for the $x^{(n)}$ training sample from task t. These parameters are computed after training on task t and stored to infer test-time task identity. Considering a uniform prior over the marginal task distribution, the task identity during testing can be inferred as follows:

$$\hat{t} = \underset{t}{\operatorname{argmin}} \left[\log|\hat{\Sigma}_t| + (\phi_t - \hat{\mu}_t)^T\hat{\Sigma}_t^{-1}(\phi_t - \hat{\mu}_t)\right] \tag{18}$$

where $\hat{t}$ is the inferred task. Note that while such a strategy does require storing statistics $\hat{\mu}_t$ and $\hat{\Sigma}_t$, the size of these is still considerably smaller than parameter statistics required by certain regularization methods like EWC [19], or the generative models used by certain replay [41, 44] or task-free methods [24].

3 Related Works

There have been a number of diverse continual learning methods that have been proposed in recent years. The strategies of much of the prior work can roughly grouped into a few categories, with some overlap. Regularization-based approaches [19, 46, 25, 33, 1, 40, 37] add a loss term constraining the network parameters to remain close to solutions of previously learned tasks. Others use replay [19, 28, 41, 33, 38], 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 [39] 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 [47] 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. To prevent constant growth of the model, Hung et al. [16] incorporates pruning to compress the network between tasks. Other works also incorporate model compression or enforce representational sparsity [40, 3], learning multiple tasks within a single network. In a way, these are also expansion methods, but with all the "expansion" done upfront: the model is over-parameterized for earlier tasks and slowly consumes capacity as more tasks are learned.

A few recent works have also explored continual learning from a Bayesian nonparametric perspective. Lee et al. [24] 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. Concurrently with our work, Kumar et al. [21] have proposed imposing IBP to expand the hidden units of each layer in a MLP, whereas we expand the number of factors of the weight matrix in each layer, allowing us to scale our method to convolutional layers.

[2]We empirically found that using the pseudo-inverse in the second term and dropping the first term in (18) works well when $\hat{\Sigma}$ is degenerate. Shrinkage estimators can also be used to better approximate Σ.

4 Experiments

We evaluate our method in two different 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, resembling what is often called multi-task learning. 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 early 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 (16) to be the representation after the first layer. We then visualize some IBP-WF weight factors to verify some of its behavior. At test time, we take 100 samples from the posterior (9) and form an ensemble of outputs to make the final prediction. All experiments are run on a single NVIDIA Titan X GPU.

4.1 Baselines

We compare IBP-WF with a number of other approaches, grouped into related families and 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** [17] and **Adagrad** [5].

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** [19] refines this by weighting the L_2 by parameter importance, using the Fisher Information; Schwarz et al. [40], Liang et al. [27] propose an online version that provides better scaling (**Online EWC**). **SI** [46] 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** [1] computes parameter importance as well, but with respect to the model output rather than the loss. **LwF** [25] leverages knowledge distillation [14] principles, using previous model outputs as additional training objectives.

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** [28] proposes instead using these as inequality restraints: the model should not increase the loss on saved samples. Regardless of how stored samples are used, however, in certain settings, data is private [36] or classified [26], and keeping data may be considered as violating continual learning criteria. As an alternative, **DGR** [41] and **RtF** [44] propose generative models [10] as a source of replay. Such approaches avoid carrying around older data, but require learning (and storing) generative models for each task.

We use the codebase from Hsu et al. [15] and van de Ven and Tolias [45] as our continual learning "sandbox." Best efforts were made to keep the model capacity consistent in all methods for a fair comparison. The architectures used are summarized in Section 4.2. Additional details on training can be found in Appendix D.

4.2 Datasets and Architectures

We evaluate IBP-WF on a number of common continual learning benchmarks. For each, the model is presented with a series of classification tasks arriving in a sequence that it trains on. This is done *without* revisiting the data from previous tasks, unless otherwise stated (*e.g.,* using a memory buffer or coreset for certain replay methods).

Method	Replay	Split MNIST		Permuted MNIST	
		Incremental Task	Incremental Class	Incremental Task	Incremental Class
SGD		97.98 ± 0.09	19.46 ± 0.04	94.94 ± 0.24	14.02 ± 1.25
Adam		93.46 ± 2.01	19.71 ± 0.08	93.42 ± 0.56	12.82 ± 0.95
Adagrad		98.06 ± 0.53	19.82 ± 0.09	94.78 ± 0.18	29.09 ± 1.48
L_2		98.18 ± 0.96	22.52 ± 1.08	95.45 ± 0.44	13.92 ± 1.79
EWC		97.70 ± 0.81	19.80 ± 0.05	95.38 ± 0.33	26.32 ± 4.32
Online EWC		98.04 ± 1.10	19.77 ± 0.04	95.15 ± 0.49	42.58 ± 6.50
SI		98.56 ± 0.49	19.67 ± 0.09	96.31 ± 0.19	58.52 ± 4.20
MAS		99.22 ± 0.21	19.52 ± 0.29	96.65 ± 0.18	50.81 ± 2.92
LwF		99.60 ± 0.03	24.17 ± 0.33	69.84 ± 0.46	22.64 ± 0.23
Naive rehearsal	✓	99.34 ± 0.10	84.25 ± 0.92	96.67 ± 0.12	96.25 ± 0.10
GEM	✓	98.27 ± 0.09	89.05 ± 0.35	97.05 ± 0.07	96.72 ± 0.03
DGR	✓	99.47 ± 0.03	91.24 ± 0.33	92.52 ± 0.08	92.19 ± 0.09
RtF	✓	99.66 ± 0.03	**92.56** ± 0.21	97.31 ± 0.01	96.23 ± 0.04
Offline		99.52 ± 0.16	97.53 ± 0.30	98.01 ± 0.04	97.95 ± 0.04
IBP-WF (Ours)		**99.69** ± 0.05	86.59 ± 1.09	**97.37** ± 0.05	**97.03** ± 0.04

Table 1: The average accuracy (%, higher is better) of all seen tasks after learning on a sequence of tasks using a 2-hidden layer MLP. Note that *Offline* is not a continual learning method and refers to the setting where all the data from previous tasks is available.

The standard train/validation/test splits were used.

Split MNIST Following Zenke et al. [46], the 10 digit classes of the MNIST [23] 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 on the other hand, without labels, each model must classify which of $2t$ classes an image belongs to, 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. [11], 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.

For the MNIST tasks, we use a 2-layer MLP with 400 hidden units for Split MNIST and 1000 hidden units for Permuted MNIST as our base model, setting $\alpha = 20$ for incremental task setting and $\alpha = 100$ for incremental class setting. For GEM and naive rehearsal, a buffer of 400 and 1.1K images were saved for Split MNIST and Permuted MNIST, respectively, to replay previous tasks. For DGR and RtF a 2-layer symmetric variational autoencoder [18] was learned for each task.

Results are shown in Table 1. 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, we can quantify uncertainty of our model's predictions; see Appendix C for analysis.

Split CIFAR10 We split the CIFAR10 [20] dataset into a sequence of 5 binary classification tasks (see Figure 3 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.

Method	Replay	Split CIFAR10	
		Incremental Task	Incremental Class
SGD		66.60 ± 7.38	19.52 ± 0.03
Adam		62.49 ± 6.01	19.61 ± 0.01
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	**35.77** ± 1.63

Table 2: The average accuracy (%) of all seen tasks after learning the task sequence using ResNet-20.

State-of-the-art classification results on CIFAR10 are typically achieved with convolutional neural networks, and we demonstrate that IBP-WF can scale by using ResNet-20 [13] as our architecture[3], using separate batch normalization layers for each task. For task inference at test time, we take the average across spatial dimensions H and W to get feature statistics ϕ and then proceed with the parameter estimation procedure introduced in (17). We keep a buffer of 400 images from previous tasks for naive rehearsal. Table 2 shows the results on Split CIFAR10. We again see that IBP-WF performs well relative to the compared baseline methods. As with the MLP used for MNIST, our Bayesian setup allows for model uncertainty estimation.

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 2. 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 number of new parameters is not defined directly by some preset schedule, but rather 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

[3]Please see Appendix D for more details on hyperparameters and the value of α used each layer in the ResNet-20 architecture.

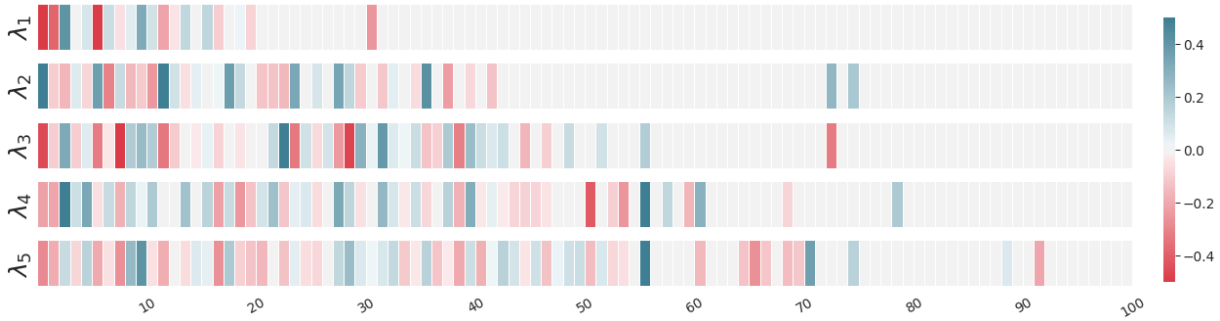

Figure 2: The expectation of the posterior of weight factor scores $\boldsymbol{\lambda}_t^{\ell=1}$ for the 5 tasks of Split CIFAR-10. The IBP prior encourages the reuse of previously used factors and assigns a decreasing probability of allocating new factors with each task. As shown, the model adapts the weight of the layer as the input distribution shifts with different tasks.

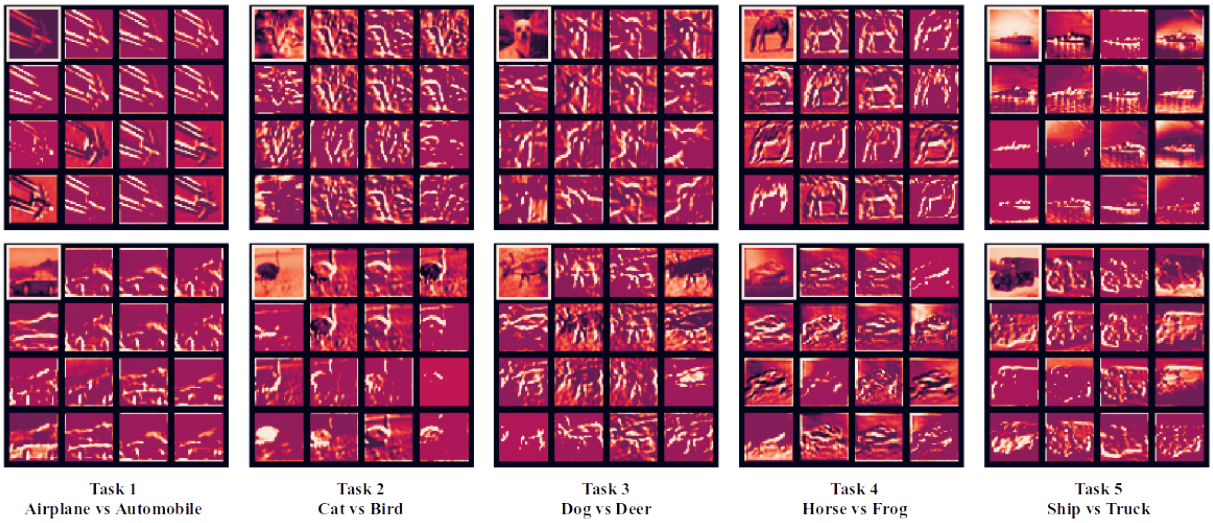

Figure 3: The first layer representations for each class in a trained IBP-WF ResNet-20. Each 4×4 grid shows the input image highlighted in the top left, with the three color channels averaged. The rest of the entries in each grid show the feature representations after convolution for 15 of the convolutional kernels.

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 sparse, regularized by the IBP to have α active factors in expectation. Another appealing aspect of using IBP is that the *rate* of allocating a new factor decreases with tasks. Finally, following the "rich gets richer" principle, 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 3). We observe an interesting property of the model: the feature maps in earlier tasks are similar (for instance see features for Task 1) compared to the diverse feature maps for later tasks. This is attributed to the fact that an early task uses very few factors since IBP induces a regularizing effect on the rank of the weight filter. However, as the model observes more data, the filters become more diverse, resulting in varied features maps (see the features for Task 5).

5 Conclusions

Motivated by the characteristics and desiderata of continual learning, we have introduced an expansion-based approach, combining weight factorization with the Indian Buffet Process, which we call IBP-WF. This synergy provides a number of 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, enabling us to operate in an incremental class setting. A number of experiments on common continual learning benchmarks show the effectiveness of IBP-WF, 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 common continual learning strategies, and combining some of these with IBP-WF is a promising direction for future work. For example, we see the Dirichlet process mixture model (*e.g.,* as used by Lee et al. [24] for expert selection) as a natural Bayesian nonparametric alternative to our simple feature statistic method for inferring tasks.

References

[1] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory Aware Synapses: Learning What (not) to Forget. *European Conference on Computer Vision*, 2018.

[2] R. Aljundi, K. Kelchtermans, and T. Tuytelaars. Task-Free Continual Learning. *Computer Vision and Pattern Recognition*, 2019.

[3] R. Aljundi, M. Rohrbach, and T. Tuytelaars. Selfless Sequential Learning. *International Conference on Learning Representations*, 2019.

[4] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. *International Conference on Machine Learning*, 2015.

[5] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[6] S. Farquhar and Y. Gal. Towards Robust Evaluations of Continual Learning. *arXiv preprint arXiv:1805.09733*, 2018.

[7] S. Farquhar and Y. Gal. A Unifying Bayesian View of Continual Learning. *arXiv preprint arXiv:1902.06494*, 2019.

[8] Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[9] Z. Ghahramani and T. L. Griffiths. Infinite Latent Feature Models and the Indian Buffet Process. *Neural Information Processing Systems*, 2006.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. *Neural Information Processing Systems*, 2014.

[11] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013.

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

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2016.

[14] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[15] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. *arXiv preprint arXiv:1810.12488*, 2018.

[16] C.-Y. Hung, C.-H. Tu, C.-E. Wu, C.-H. Chen, Y.-M. Chan, and C.-S. Chen. Compacting, Picking and Growing for Unforgetting Continual Learning. *Neural Information Processing Systems*, 2019.

[17] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013.

[19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, 2017.

[20] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.

[21] A. Kumar, S. Chatterjee, and P. Rai. Nonparametric Bayesian Structure Adaptation for Continual Learning. *arXiv preprint arXiv:1912.03624*, 2019.

[22] P. Kumaraswamy. A Generalized Probability Density Function for Double-Bounded Random Processes. *Journal of Hydrology*, 1980.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] S. Lee, J. Ha, D. Zhang, and G. Kim. A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning. *International Conference on Learning Representations*, 2020.

[25] Z. Li and D. Hoiem. Learning Without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

[26] K. J. Liang, G. Heilmann, C. Gregory, S. O. Diallo, D. Carlson, G. P. Spell, J. B. Sigman, K. Roe, and L. 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*, 2018.

[27] K. J. Liang, C. Li, G. Wang, and L. Carin. Generative Adversarial Network Training is a Continual Learning Problem. *arXiv preprint arXiv:1811.11083*, 2018.

[28] D. Lopez-Paz and M. Ranzato. Gradient Episodic Memory for Continual Learning. *Neural Information Processing Systems*, 2017.

[29] C. J. Maddison, A. Mnih, and Y. W. Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *International Conference on Learning Representations*, 2017.

[30] M. McCloskey and N. J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *The Psychology of Learning and Motivation*, 1989.

[31] J. V. Michalowicz, J. M. Nichols, and F. Bucholtz. *Handbook of Differential Entropy*. Chapman and Hall/CRC, 2013. ISBN 1466583169.

[32] E. Nalisnick and P. Smyth. Stick-breaking variational autoencoders, 2016.

[33] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational Continual Learning. *International Conference on Learning Representations*, 2018.

[34] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 2019.

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

[36] D. Ribli, A. Horváth, Z. Unger, P. Pollner, and I. Csabai. Detecting and Classifying Lesions in Mammograms with Deep Learning. *Scientific reports*, 8(1):1–7, 2018.

[37] H. Ritter, A. Botev, and D. Barber. Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting. *Neural Information Processing Systems*, 2018.

[38] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne. Experience Replay for Continual Learning. *Neural Information Processing Systems*, 2019.

[39] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016.

[40] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & Compress: A Scalable Framework for Continual Learning. *International Conference on Machine Learning*, 2018.

[41] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual Learning with Deep Generative Replay. *Neural Information Processing Systems*, 2017.

[42] J. Song, Z. Gan, and L. Carin. Factored Temporal Sigmoid Belief Networks for Sequence Learning. *International Conference on Machine Learning*, 2016.

[43] G. W. Taylor and G. E. Hinton. Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style. *International Conference on Machine Learning*, 2009.

[44] G. M. van de Ven and A. S. Tolias. Generative Replay with Feedback Connections as a General Strategy for Continual Learning. *arXiv preprint arXiv:1809.10635*, 2018.

[45] G. M. van de Ven and A. S. Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[46] F. Zenke, B. Poole, and S. Ganguli. Continual Learning Through Synaptic Intelligence. *International Conference on Machine Learning*, 2017.

[47] J. O. Zhang, A. Sax, A. Zamir, L. Guibas, and J. Malik. Side-Tuning: Network Adaptation via Additive Side Networks. *arXiv preprint arXiv:1912.13503*, 2019.

A Kullback-Leibler (KL) Divergence Derivation

Nalisnick and Smyth [32] gave an approximate form for the KL divergence between Kumaraswamy and Beta distributions. Following their steps, we derive the analytical form to approximate the KL divergence between two Kumaraswamy distributions q and p.

$$\mathrm{KL}\left(q_v\left(a,b\right)||p_v\left(\alpha,\beta\right)\right) = \mathbb{E}_{q_v}\left[\log\frac{q_v\left(a,b\right)}{p_v\left(\alpha,\beta\right)}\right] \tag{19}$$

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

$$\mathrm{KL}\left(q_v\left(a,b\right)||p_v\left(\alpha,\beta\right)\right) = \underbrace{\mathbb{E}_{q_v}\left[\log q_v\left(a,b\right)\right]}_{\mathcal{T}1} - \underbrace{\mathbb{E}_{q_v}\left[\log p_v\left(\alpha,\beta\right)\right]}_{\mathcal{T}2} \tag{20}$$

where the first term is the Kumaraswamy entropy. Using Michalowicz et al. [31], we have:

$$\mathcal{T}1 = \log ab + \frac{a-1}{a}\left(-\gamma - \Psi(b) - \frac{1}{b}\right) - \frac{b-1}{b} \tag{21}$$

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

$$\begin{aligned}\mathcal{T}2 &= \mathbb{E}_{q_v}\log\left(\alpha\beta v^{\alpha-1}\left(1-v^\alpha\right)^{\beta-1}\right)\\ &= \mathbb{E}_{q_v}\left[\log\alpha\beta + (\alpha-1)\log v + (\beta-1)\log\left(1-v^\alpha\right)\right]\\ &= \log\alpha\beta + (\alpha-1)\,\mathbb{E}_{q_v}\left[\log v\right] + (\beta-1)\,\mathbb{E}_{q_v}\left[\log\left(1-v^\alpha\right)\right]\end{aligned} \tag{22}$$

In the above equation, the expectation of the log terms can be computed using Gradshteyn and Ryzhik [12] (4.253):

$$\begin{aligned}\mathcal{T}2 &= \log\alpha\beta + (\alpha-1)\,\mathbb{E}_{q_v}\left[\log v\right] + (\beta-1)\,\mathbb{E}_{q_v}\left[\log\left(1-v^\alpha\right)\right]\\ &= \log\alpha\beta + \left(\frac{\alpha-1}{a}\right)\left(-\gamma - \Psi(b) - \frac{1}{b}\right) + (\beta-1)\,\mathbb{E}_{q_v}\left[\log\left(1-v^\alpha\right)\right]\end{aligned} \tag{23}$$

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

$$\log\left(1-v^\alpha\right) = -\sum_{m=1}^{\infty}\frac{1}{m}v^{m\alpha} \tag{24}$$

Note that the infinite sum in (24) converges since $0 < v < 1$. From the monotone convergence theorem, we can take the expectation inside the sum:

$$\begin{aligned}\mathbb{E}_{q_v}\left[\log\left(1-v^\alpha\right)\right] &= -\sum_{m=1}^{\infty}\frac{1}{m}\mathbb{E}_{q_v}v^{m\alpha}\\ &= -\sum_{m=1}^{\infty}\frac{b}{m}\,\mathrm{B}\left(\frac{m\alpha}{a}+1,b\right)\\ &= -\sum_{m=1}^{\infty}\frac{\alpha b}{m\alpha+ab}\,\mathrm{B}\left(\frac{m\alpha}{a},b\right)\end{aligned} \tag{25}$$

where $\mathrm{B}(.,.)$ is the beta function and $b\,\mathrm{B}\left(\frac{m\alpha}{a}+1,b\right)$ is the $(m\,\alpha)$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 (25) in our experiments. Using (21), (23), and (25) we have:

$$\begin{aligned}\mathrm{KL}\left(q_v\left(a,b\right)||p_v\left(\alpha,\beta\right)\right) =& \log\frac{ab}{\alpha\beta} + \frac{a-\alpha}{a}\left(-\gamma-\Psi(b)-\frac{1}{b}\right) - \frac{b-1}{b}\\ &+ (\beta-1)\sum_{m=1}^{\infty}\frac{\alpha b}{m\alpha+ab}\,\mathrm{B}\left(\frac{m\alpha}{a},b\right)\end{aligned} \tag{26}$$

B 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 $\boldsymbol{\theta}_t = \{\boldsymbol{r}_t^\ell, \boldsymbol{b}_t^\ell, \boldsymbol{v}_t^\ell\}_{\ell=1}^L$. The following variational distributions were used:

$$q(\boldsymbol{\theta}_t^\ell) = q(\boldsymbol{r}_t^\ell) q(\boldsymbol{b}_t^\ell) q(\boldsymbol{v}_t^\ell) \tag{27}$$

$$\boldsymbol{r}_t^\ell \sim \mathcal{N}\left(\boldsymbol{\mu}_t^\ell,\, \text{diag}(\boldsymbol{\sigma}_t^{\ell^2})\right) \tag{28}$$

$$\boldsymbol{b}_t^\ell \sim \text{Bernoulli}(\boldsymbol{\pi}_t^\ell) \tag{29}$$

$$\boldsymbol{v}_t^\ell \sim \text{Kumar}(\boldsymbol{c}_t^\ell, \boldsymbol{d}_t^\ell) \tag{30}$$

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

$$\mathcal{L}_t = \sum_{n=1}^{N_t} \mathbb{E}_q \log p\left(y_t^{(n)} \middle| \boldsymbol{\theta}_t, x_t^{(n)}\right) - \text{KL}\left(q\left(\boldsymbol{\theta}_t\right) || p\left(\boldsymbol{\theta}_t\right)\right) \tag{31}$$

where N_t is the number of training examples in task t.

$$\text{KL}\left(q(\boldsymbol{\theta}_t) || p\left(\boldsymbol{\theta}_t\right)\right) = \sum_{\ell=1}^{L} \text{KL}\left(q(\boldsymbol{r}_t^\ell) || p(\boldsymbol{r}_t^\ell)\right) + \text{KL}\left(q(\boldsymbol{b}_t^\ell) || p(\boldsymbol{b}_t^\ell | \boldsymbol{v}_t^\ell)\right) + \text{KL}\left(q(\boldsymbol{v}_t^\ell) || p(\boldsymbol{v}_t^\ell)\right) \tag{32}$$

Nguyen et al. [33] 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 [6, 7] 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 (32), instead of performing online inference for all parameters $\{\boldsymbol{r}_t^\ell, \boldsymbol{b}_t^\ell, \boldsymbol{v}_t^\ell\}$, we only apply online inference for $\boldsymbol{v}_t^\ell$ and learn task-specific parameters $\{\boldsymbol{r}_t^\ell, \boldsymbol{b}_t^\ell\}$.

Inference for $\boldsymbol{v}_t^\ell$: Starting with the first task ($t = 1$), we initialize the prior $p(\boldsymbol{v}_1^\ell) = \text{Beta}(\alpha, 1)$ and learn the posterior $q(\boldsymbol{v}_1^\ell) = \text{Kumar}(\boldsymbol{c}_1^\ell, \boldsymbol{d}_1^\ell)$ using Bayes by Backprop [4]. Note that $\text{Beta}(\alpha, 1)$ has the same density function as $\text{Kumar}(\alpha, 1)$. For all the following tasks, the prior $p(\boldsymbol{v}_t^\ell) = q(\boldsymbol{v}_{t-1}^\ell)$ and the posterior $q(\boldsymbol{v}_t^\ell) = \text{Kumar}(\boldsymbol{c}_t^\ell, \boldsymbol{d}_t^\ell)$ is learned in the same way as in task 1. Note that we use mean-field approximation for the posterior: $q(v_{t,i}^\ell) = \text{Kumar}(c_{t,i}^\ell, d_{t,i}^\ell)$. We use (26) to compute the KL divergence between the posterior and the prior in (32).

Inference for $\boldsymbol{r}_t^\ell$: We use the prior $p(\boldsymbol{r}_t^\ell) = \mathcal{N}(0, 1)$ and learn the posterior $q(\boldsymbol{r}_t^\ell) = \mathcal{N}(\boldsymbol{\mu}_t^\ell, \text{diag}(\boldsymbol{\sigma}_t^{\ell^2}))$ for all tasks $t = 1, 2, ...T$. We use Bayes by Backprop to learn the parameters $\boldsymbol{\mu}_t^\ell$ and $\boldsymbol{\sigma}_t^\ell$.

Inference for $\boldsymbol{b}_t^\ell$: We use the $\text{BernoulliConcrete}_\lambda$ distribution [29] 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 $p(b_{t,k}^\ell) = \text{BernoulliConcrete}_\lambda(\pi_{t,k}^\ell)$, where $\pi_{t,k}^\ell := \prod_{i=1}^{i=k} v_{t,i}^\ell$ and $v_{t,i}^\ell \sim q(v_{t,i}^\ell)$. The posterior is then $q(b_{t,k}^\ell) = \text{BernoulliConcrete}_\lambda(\bar{\pi}_{t,k}^\ell)$, 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 in Maddison et al. [29].

C 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:

$$\mathbb{H}\left[y^* | x^*, \mathcal{D}_{train}\right] = -\sum_c p(y^* = c | x^*, \mathcal{D}_{train}) \log p(y^* = c | x^*, \mathcal{D}_{train}) \tag{33}$$

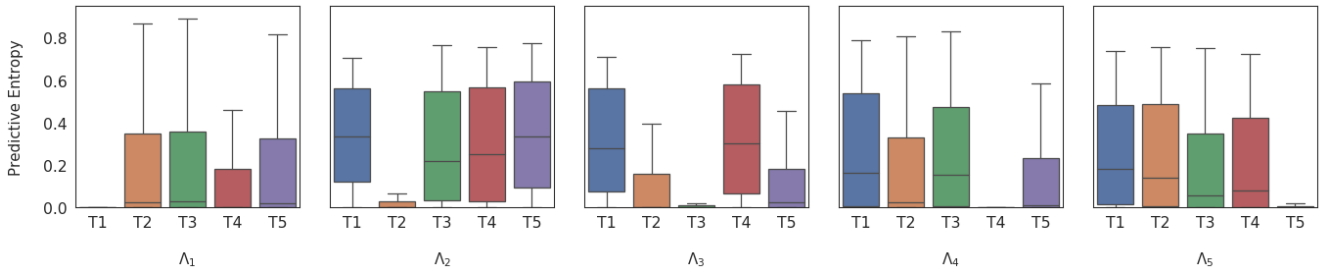

Figure 4: Uncertainty in the incremental task setting for Split MNIST dataset. Each of the 5 plots depicts the uncertainty for the test sets when task-specific parameter Λ_t is used. The y-axis denotes the uncertainty (as the predictive entropy in nats), and x-axis denotes the test sets ($\mathcal{T}_1$ through $\mathcal{T}_5$). We use $M = 100$.

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 $p(y^* = c|x^*, \mathcal{D}_{train}) = p(y^* = c|x^*, t^*, \mathcal{D}_{train})$, where t^* is the associated task-identity with the input x^* during testing. Following Gal [8], we approximate the predictive distribution by using an ensemble of M neural networks sampled from the posterior distribution:

$$p(y^* = c|t = t^*, x^*, \mathcal{D}_{train}) = \underbrace{\frac{1}{M}\sum_{m=1}^{M} p(y^* = c|x^*; \theta_{t^*}^{(m)})}_{\rho_{t^*,c}^{M}}, \text{ where } \theta_{t^*}^{(m)} \sim q(\theta_{t^*}) \tag{34}$$

where $\theta^{(1)}...\theta^{(M)}$ are M samples drawn from $q(\theta_{t^*})$. Using this we can compute a biased estimate[4] of the predictive entropy as follows:

$$\begin{aligned}
\mathbb{H}\left[y^*|x^*, \mathcal{D}_{train}\right] &= -\sum_c p(y^* = c|x^*, \mathcal{D}_{train}) \log p(y^* = c|x^*, \mathcal{D}_{train}) && (35)\\
&= -\sum_c p(y^* = c|x^*, t^*, \mathcal{D}_{train}) \log p(y^* = c|x^*, t^*, \mathcal{D}_{train}) && (36)\\
&= -\sum_c \left(\frac{1}{M}\sum_{m=1}^{M} p(y^* = c|x^*; \theta_{t^*}^{(m)})\right) \log \left(\frac{1}{M}\sum_{m=1}^{M} p(y^* = c|x^*; \theta_{t^*}^{(m)})\right) && (37)\\
&= -\sum_c \left(\rho_{t^*,c}^{M}\right) \log \rho_{t^*,c}^{M} && (38)
\end{aligned}$$

Figure 4 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 4, 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.

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:

$$\begin{aligned}
p(y^* = c|x^*, \mathcal{D}_{train}) &= \sum_{t'} p(y^* = c, t = t'|x^*, \mathcal{D}_{train}) && (39)\\
&= \sum_{t'} p(y^* = c|t = t', x^*, \mathcal{D}_{train}) p(t = t'|x^*, \mathcal{D}_{train}) && (40)\\
&= \sum_{t'} p(y^* = c|t = t', x^*, \mathcal{D}_{train}) \frac{P(\phi_{t'}(x))}{\sum_t P(\phi_t(x))} && (41)\\
&= \sum_{t'} \frac{\rho_{t',c}^{M} P(\phi_{t'}(x))}{\sum_t P(\phi_t(x))} && (42)
\end{aligned}$$

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

$$\mathbb{H}\left[y^*|x^*, \mathcal{D}_{train}\right] = -\sum_c \left(\left(\sum_{t'} \frac{\rho_{t',c}^M P(\phi_{t'}(x))}{\sum_t P(\phi_t(x))} \right) \log \left(\sum_{t'} \frac{\rho_{t',c}^M P(\phi_{t'}(x))}{\sum_t P(\phi_t(x))} \right) \right) \tag{43}$$

We use (43) to estimate the uncertainty in the incremental class continual setting for the Split MNIST dataset. Figure 5 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.

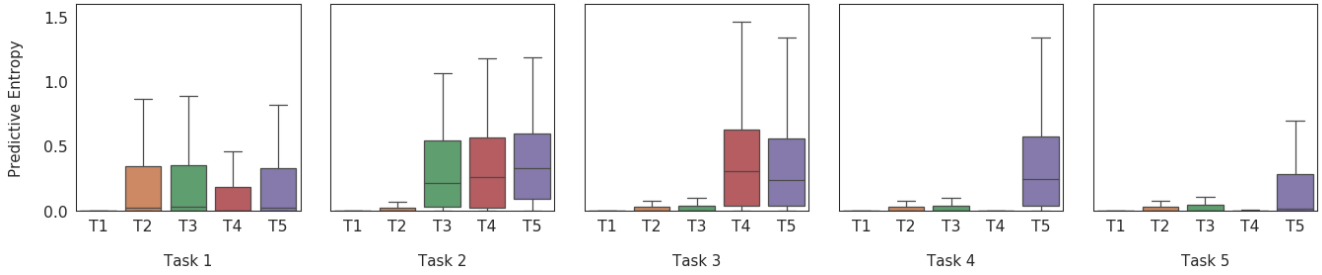

Figure 5: Uncertainty in the incremental class setting for Split MNIST dataset. We compute the uncertainty of the test sets after training on each task in the sequence. The y-axis denotes the uncertainty (as the predictive entropy in nats), and x-axis denotes the test sets ($\mathcal{T}_1$ through $\mathcal{T}_5$) for each task. We use M=100.

D Experiment Setup and Baselines

We describe the experimental configuration used:

D.1 Split MNIST

Following Hsu et al. [15], 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. We used ReLU as the non-linearity in both the hidden layers. All the baseline models were trained for 4 epochs per task with a mini-batch size of 128 with Adam [17] optimizer ($\beta_1 = 0.9, \beta_2 = 0.999, lr = 0.001$) as the default unless explicitly stated. 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 was learned for each task. 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. The prior parameter was set to $\alpha = 20$ for incremental task setting and $\alpha = 100$ for the incremental class setting. Since we are learning a distribution instead of a single point estimate of the neural network parameters, we train the model for 100 epochs per task with a mini-batch size of 128. The stick-breaking process for IBP was truncated at $K = 500$ for both the hidden layers, *i.e.* the total budget on the number of allowed factors was set to 500.

D.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. All the baseline models were trained for 10 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 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. 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 $\alpha = 500$. We train for 50 epochs for each task with a mini-batch size of 128. The stick-breaking process for IBP was truncated at $K = 1000$ for both the hidden layers.

D.3 CIFAR10

We split the CIFAR10 [20] 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 2T-wise classification in the incremental class setting, where T is the number of total tasks seen by the model.

Baselines: We use ResNet-20 [13] 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 100 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. While a 4D tensor factorization 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 factorization as we did for the MLP. The truncation parameters for the stick-breaking process was set to 100 for the first convolutional layer and 200 for all the following layers. The proposed IBP-WF method allowed the model to allocate new factors for a task, while encouraging reuse of already trained factors on previous tasks. We use 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.