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.

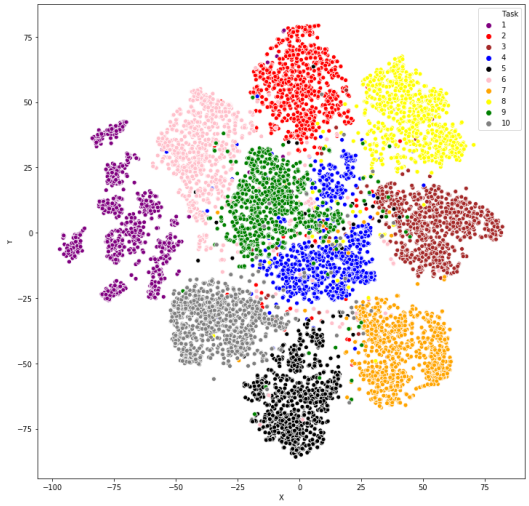

Figure 6: The t-SNE plot for the intermediate features ϕ of 10 tasks from the permuted MNIST benchmark.

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:

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

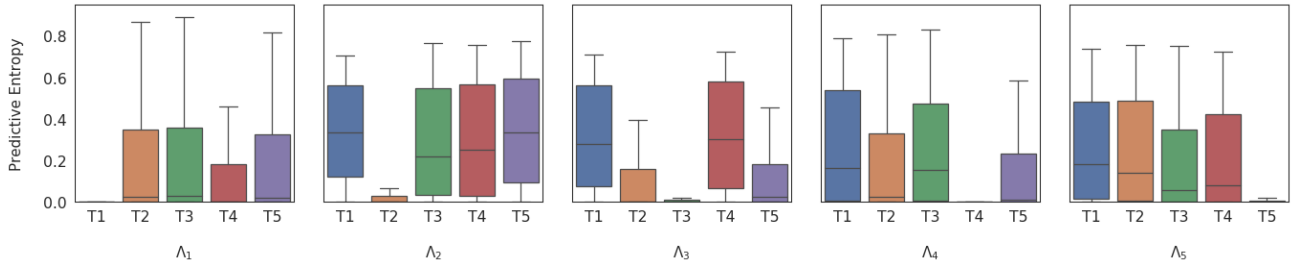

Figure 7: (Section E.1) Uncertainty in the incremental task setting for Split MNIST dataset. Each of the 5 plots depicts the uncertainxty of the test sets when task-specific parameter Λ_t is used. The y-axis denotes the uncertainty (as the predictive entropy), and x-axis denotes the test sets ($\mathcal{T}_1$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

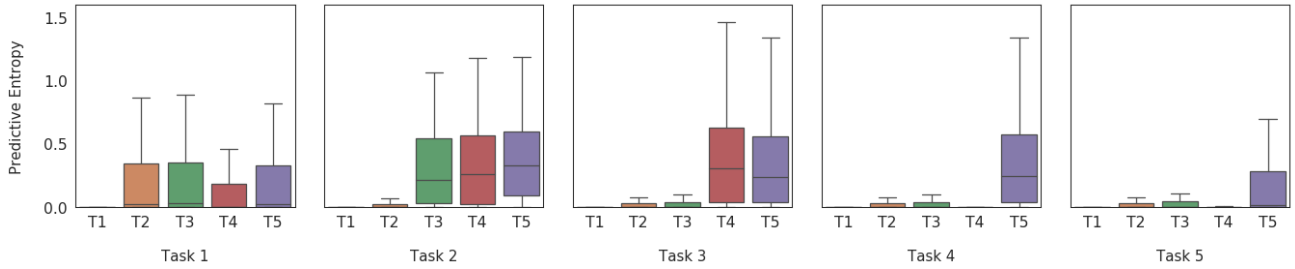

Figure 8: (Section E.2) Uncertainty in the incremental class setting for Split MNIST dataset. We compute the uncertainty of the test sets after training on each task in the sequence. The y-axis denotes the uncertainty (as the predictive entropy), and x-axis denotes the test sets ($\mathcal{T}_1$ through $\mathcal{T}_5$) 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.

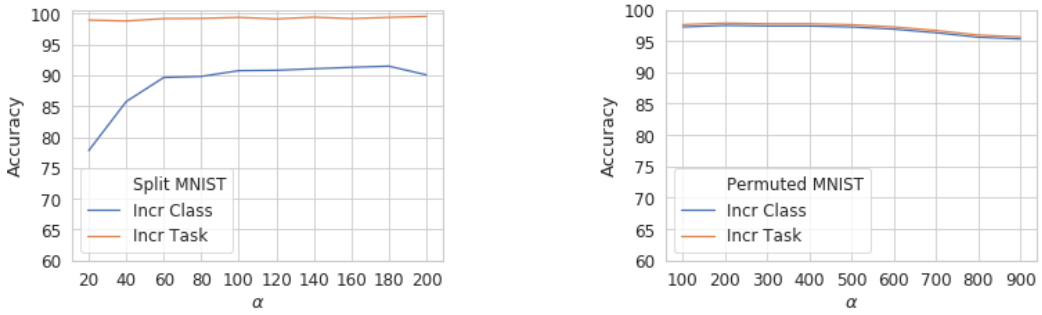

Figure 9: Ablation study on α.

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.

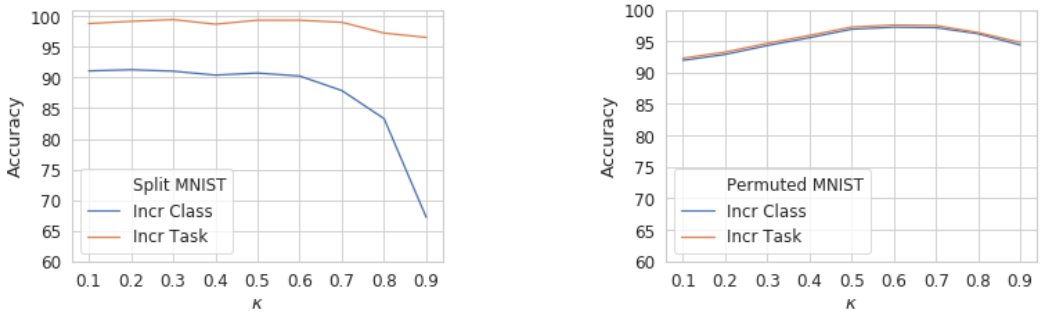

Figure 10: Ablation study on κ. We use $\kappa = 0.5$ for all experiments in the main text.

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.