Hierarchical Indian Buffet Neural Networks for Bayesian Continual Learning

University of Oxford   University of Oxford   University of Oxford   University of Oxford

Abstract

We place an Indian Buffet process (IBP) prior over the structure of a Bayesian Neural Network (BNN), thus allowing the complexity of the BNN to increase and decrease automatically. We further extend this model such that the prior on the structure of each hidden layer is shared globally across all layers, using a Hierarchical-IBP (H-IBP). We apply this model to the problem of resource allocation in Continual Learning (CL) where new tasks occur and the network requires extra resources. Our model uses online variational inference with reparameterisation of the Bernoulli and Beta distributions, which constitute the IBP and H-IBP priors. As we automatically learn the number of weights in each layer of the BNN, overfitting and underfitting problems are largely overcome. We show empirically that our approach offers a competitive edge over existing methods in CL.

1 INTRODUCTION

Humans have the ability to continually learn, consolidate their knowledge and leverage previous experiences when learning a new set of skills. In Continual Learning (CL) an agent must also learn continually, presenting several challenges including learning online, avoiding forgetting and efficiently allocating resources for learning new tasks. In CL, a neural network model is required to learn a series of tasks, one by one, and remember how to perform each. After learning each task, the model loses access to the data. More formally the model is given a set of M tasks sequentially $\mathcal{T}_i$ for $i = 1, ... M$. Where each

task is comprised of a dataset with input $x \in \mathbb{R}^d$ and output $y \in \mathbb{R}$ samples: $\mathcal{T}_1 : \mathcal{D}_1 = \{(x_{1,i}, y_{1,i})\}$ for $i = 1, ... n_1$, $\mathcal{T}_2 : \mathcal{D}_2 = \{(x_{2,i}, y_{2,i})\}$ for $i = 1, ... n_2$ and so on until $\mathcal{T}_M : \mathcal{D}_M = \{(x_{M,i}, y_{M,i})\}$ for $i = 1, ... n_M$. Although the model will lose access to the training dataset for task $\mathcal{T}_t$, it will be continually evaluated on the test sets for all previous tasks $\mathcal{T}_i$ for $i \leq t$. For a comprehensive review of the CL scenarios see [van de Ven and Tolias, 2018, Hsu et al., 2018].

The principal challenges to CL are threefold, firstly models need to overcome *catastrophic forgetting* of old tasks; a neural network will exhibit forgetting of previous tasks after having learnt a few tasks [Goodfellow et al., 2015]. Secondly, models need to leverage knowledge transfer from previously learnt tasks for learning a new task $\mathcal{T}_t$. And finally, the model needs to have enough neural resources available to learn a new task and adapt to the complexity of the task at hand.

One of the main approaches to CL involves the use of the natural sequential learning approach embedded within Bayesian inference. The prior for task $\mathcal{T}_i$ is the posterior which is obtained from the previous task $\mathcal{T}_{i-1}$. This enables knowledge transfer and offers an approach to overcome catastrophic forgetting. Previous Bayesian CL approaches have leveraged Laplace approximations [Kirkpatrick et al., 2017, Ritter et al., 2018] and variational inference [Nguyen et al., 2018, Swaroop et al., 2018, Zeno et al., 2018] to aid computational tractability. Whilst Bayesian methods solve the first and second objectives above, the third objective of ensuring that the BNN has enough neural resources to adapt its complexity to the task at hand is not necessarily achieved. For instance, additional neural resources can alter performance on MNIST classification (see Table 1 in [Blundell et al., 2015]). This is a problem as the amount of neural resources required for a current task, may not be enough (or may be redundant) for a future task. Propagating a poor approximate posterior from one task will alter performance for all subsequent tasks.

Non-Bayesian neural networks use additional neurons

---

to learn new tasks and prevent overwriting previous knowledge thus overcoming forgetting. The neural networks which have been trained on previous tasks are frozen and a new neural network is appended to the existing network for learning a new task [Rusu et al., 2016]. The problem with this approach is that of scalability: the number of neural resources increases linearly with the number of tasks. The scalability issue has been tackled with selective retraining and expansion with a group regulariser [Yoon et al., 2018]. However this solution is unable to shrink and so are vulnerable to overfitting if misspecified when starting CL. Moreover knowledge transfer and prevention of catastrophic forgetting are not solved in a principled manner, unlike approaches couched in a Bayesian framework.

As the resources required are typically unknown in advance, we propose a BNN which adds or withdraws neural resources automatically in response to the data. This is achieved by drawing on Bayesian nonparametrics to learn the structure of each hidden layer of a BNN. Thus, the model size adapts to the amount of data seen and the difficulty of the task. This is achieved by using a binary latent matrix Z, distributed according to an Indian Buffet Process (IBP) prior [Griffiths and Ghahramani, 2011]. The IBP prior on an infinite binary matrix, Z, allows inference on which and how many neurons are required for each data point in a task. The weights of the BNN are treated as draws from non-interacting Gaussians [Blundell et al., 2015]. Catastrophic forgetting is overcome by repeated application of the Bayesian update rule, embedded within variational inference [Nguyen et al., 2018]. We summarise the contributions as follows. We present a novel BNN using an IBP prior and its hierarchical extension to automatically learn the complexity of each hidden layer according to the task difficulty. The model's effective use of resources is shown to be useful in CL. We derive a variational inference algorithm for learning the posterior distribution of the proposed models. In addition, our model elegantly bridges two separate CL approaches: expansion methods and Bayesian methods (more commonly referred to as regularisation based methods in CL literature).

2 INDIAN BUFFET NEURAL NETWORKS

We introduce variational Bayesian approaches to CL in Section 2.1. We present the IBP prior in Section 2.2 and the IBP prior on the latent binary matrix Z is then applied to a BNN such that the complexity of each hidden layer can be learnt from the data in Section 2.3. In Section 3, the Hierarchical IBP prior (H-IBP) is introduced and applied to the BNN to encourage a

more regular structure. Thus, the use of an IBP and H-IBP prior over the hidden states of the BNN can be readily used together with the Bayesian CL framework presented, and so automatically adapt its complexity according to the task.

2.1 Bayesian CL

The CL process can be decomposed into Bayesian updates where the approximate posterior for $\mathcal{T}_{t-1}$ can be used as a prior for task $\mathcal{T}_t$. Variational CL (VCL) [Nguyen et al., 2018] uses a BNN to perform the prediction tasks where the network weights are independent Gaussians. The variational posterior from previous tasks is used as a prior for new tasks. Consider learning the first task $\mathcal{T}_1$, and ϕ are the variational random variables, then the variational posterior is $q_1(\phi|\mathcal{D}_1)$. For the subsequent task, access to $\mathcal{D}_1$ is lost and the prior will be $q_1(\phi|\mathcal{D}_1)$, optimisation of the ELBO will yield the variational posterior $q_2(\phi|\mathcal{D}_2)$. Generalising, the negative ELBO for the t-th task is:

$$\mathcal{L}(\phi, \mathcal{D}_t) = D_{\mathrm{KL}}\left[q_t(\phi)||q_{t-1}(\phi|\mathcal{D}_{t-1})\right] \\ - \mathbb{E}_{q_t}[\log p(\mathcal{D}_t|\phi)]. \tag{1}$$

The first term acts to regularise the posterior such that it is close to previous task's posterior and the second term is the log-likelihood of the data for the current task.

2.2 IBP prior

Matrix decomposition aims to represent the data X as a combination of latent features: $X = ZA + \epsilon$ where $X \in \mathbb{R}^{N \times D}$, $Z \in \mathbb{Z}_2^{N \times K}$, $A \in \mathbb{R}^{K \times D}$ and ϵ is an observation noise. Each element in Z corresponds to the presence or absence of a latent feature from A. Specifically, $z_{ik} = 1$ corresponds to the presence of a latent feature A_k in observation X_i and $k \in \{1, \cdots, \infty\}$ all columns in Z with $k > K$ are assumed to be zero. In a scenario where the number of latent features K is to be inferred, then the IBP prior on Z is suitable [Doshi-Velez et al., 2009].

One representation of the IBP prior is the stick-breaking formulation [Teh et al., 2007]. The probability π_k is assigned to the column z_k for $k \in \{1, \cdots, \infty\}$, whether a feature has been selected is determined by $z_{nk} \sim \mathrm{Bern}(\pi_k)$. This parameter π_k is generated according to the following stick-breaking process: $v_k \sim \mathrm{Beta}(\alpha, 1)$, and $\pi_k = \prod_{i=1}^{k} v_i$, thus π_k decreases exponentially with k. The Beta concentration parameter α controls how many features one expects to see in the data, the larger α is, the more latent features are present.

2.3 An IBP BNN

Consider a BNN with k_l neurons for each layer $l \in \{1, \dots L\}$ layers. Thence, for an arbitrary activation f, the binary matrix Z is applied elementwise $h_l = f(h_{l-1} W_l) \circ Z_l$ where $h_{l-1} \in \mathbb{R}^{N \times k_{l-1}}$, $W_l \in \mathbb{R}^{k_{l-1} \times k_l}$, $Z_l \in \mathbb{Z}_2^{N \times k_l}$, and where $\circ$ is the elementwise product and N is the number of data points per batch. We have ignored biases for simplicity. Z_l is distributed according to an IBP prior. The IBP prior has some suitable properties for this application: the number of neurons sampled grows with N and the promotion of "rich get richer" scheme for neuron selection [Griffiths and Ghahramani, 2011]. For convenience, we term the IBP BNN as IBNN for the remainder of the paper.

The number of neurons selected grow or contract according to the variational objective; which depends on the complexity of the data. This allows for efficient use of neural resources which is crucial to a successful CL model. The variational objectives for the IBP prior and BNN are introduced further down the line in Section 2.4 and Section 3.2. Additionally, the "rich get richer" scheme is useful since the common neurons are selected across tasks enabling knowledge transfer and preventing forgetting.

As a standard practice in variational inference with a Bayesian nonparametric prior, we use a truncation level K, to the maximum number of features in the variational IBP posterior. [Doshi et al., 2009] present bounds on the marginal distribution of X in a matrix factorisation setting and show that the bound decreases exponentially as K increases. A similar behaviour is expected for our application.

2.4 SSVI

Structured stochastic VI (SSVI) has been shown to perform better inference of the IBP posterior than mean-field VI in deep latent variable models [Singh et al., 2017]. Hence, this inference method has been chosen for learning and presented next.

A separate binary matrix Z_j can be applied to each layer $j \in \{1, \dots J\}$ of a BNN. The subscript j is dropped for clarity. The structured variational approximation is: $q(\phi) = \prod_{k=1}^{K} q(v_k) q(\boldsymbol{w}_k) \prod_{i=1}^{N} q(z_{ik}|v_k)$, where the random variables are $\phi = \{v_k, \boldsymbol{w}_k, Z_k\}$ and the variational parameters $\varphi = \{\alpha_k, \beta_k, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k\}$ for all k. The variational distributions over ϕ are defined below and the variational IBP posterior is truncated to K. The constituent distributions are $q(v_k) = \text{Beta}(\alpha_k, \beta_k)$, $\pi_k = \prod_{i=1}^{k} v_i$, $q(z_{ik}) = \text{Bern}(\pi_k)$ and the BNN weights are independent draws from $q(\boldsymbol{w}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2 \mathbb{1})$. Having defined the structured variational objective,
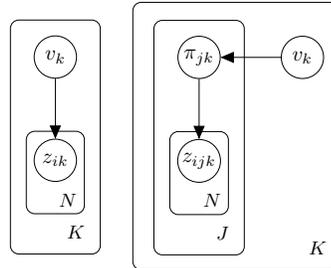

Figure 1: The graphical model for the structured variational posterior approximation for **L**, the IBP and **R**, the H-IBP[1]. Using the language of the eponymous IBP prior metaphor, k (dishes) indicates the neurons, the number selected K adjusts flexibly. j (restaurants) is the number of layers which is fixed. i (customers) is a data point.

the negative ELBO is:

$$
\begin{aligned}
\mathcal{L}(\phi, \mathcal{D}) &= D_{\text{KL}}(q(\boldsymbol{v})||p(\boldsymbol{v})) + D_{\text{KL}}(q(\boldsymbol{w})||p(\boldsymbol{w})) \\
&\quad - \sum_{i=1}^{|\mathcal{D}|} \mathbb{E}_{q(\phi)} \left[\log p(y_i|\mathbf{x}_i, \mathbf{z}_i, \boldsymbol{w}) \right] \\
&\quad + \sum_{i=1}^{|\mathcal{D}|} D_{\text{KL}}(q(\mathbf{z}_i|\boldsymbol{\pi})||p(\mathbf{z}_i|\boldsymbol{\pi})).
\end{aligned}
\tag{2}
$$

We note that the Bernoulli random variables have an explicit dependence on the stick-breaking probabilities in the structured variational approximation. However, this explicit dependence between parameters is removed in a mean field approximation. The variational parameter Beta parameters will control the expected hidden state size for the BNN and is automatically inferred. For sequential Bayesian updates the posterior is then used as the next task's prior, thus a prior only needs to be designed for the first task.

3 HIERARCHICAL IBNN

In the previous section, we presented the IBNN model to allow a BNN to automatically select the number of neurons for each layer according to the data. For a multi-layer BNN, one can apply the IBP prior independently for each layer. However, we can add an inductive bias to ensure that the inferred number of neurons are similar for across all layers of an MLP and ensure that information is shared across layers. To do this, we propose a hierarchical IBP prior [Thibaux and Jordan, 2007] for neuron selection across multiple layers. The number of neurons from all layers are generated from the same global prior, thus will discourage irregular structure in the BNN (a BNN

with adjacent wide and narrow layers might be inferred when using independent priors on each hidden layer). Of course, this property might not be desirable for all use cases, however the majority of BNNs used in the literature have a regular structure. We term our model as HIBNN and present the graphical model which describes the hierarchical IBP prior in Figure 1.

3.1 Our HIBP

The global probability of selecting the neuron positioned at the k-th index across all layers is defined according to a stick-breaking process:

$$\pi_k^0 = \prod_{i=1}^{k} v_i, \quad v_k \sim \text{Beta}(\alpha, 1), \quad k = 1, \cdots \infty. \quad (3)$$

Child IBPs are defined over the structure of each individual hidden layer of a BNN which depend on π_k^0 to define the respective Bernoulli probabilities of selecting neuron k in layer j:

$$\pi_{jk} \sim \text{Beta}\left(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)\right), \quad (4)$$

for $j \in \{1, \cdots J\}$, $k \in \{1, \cdots \infty\}$, where J is the number of layers in a BNN, α_j are hyperparameters [Thibaux and Jordan, 2007, Gupta et al., 2012]. The selection of the k-th neuron in the j-th layer by a particular data point i in the dataset of size N is thus:

$$z_{ijk} \sim \text{Bern}(\pi_{jk}), \, i = 1, \cdots N. \quad (5)$$

Notice that if k is small, π_k^0 is close to 1 then the shape parameter of the child Beta distribution will be large. At the same time the scale parameter will be small. So the Bernoulli probability in Equation (5) will be close to 1, as k increases π_k^0 and π_{jk} decrease. To infer the posterior $p(\boldsymbol{v}, Z, \boldsymbol{w}|\mathcal{D})$, we perform SSVI.

3.2 SVI

A structured variational posterior distribution which retains properties of the true posterior is desired such that the global stick-breaking probabilities influence child stick-breaking probabilities of each layer of the BNN. Let us define the variational distributions for our hidden variables as follows, $q(v_k^0) = \text{Beta}(\alpha_k^0, \beta_k^0)$ and $q(\pi_{jk}) = \text{Beta}(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0))$, $\pi_k^0 = \prod_{i=1}^{k} v_i^0$, $q(z_{ijk}) = \text{Bern}(\pi_{jk})$ and the weights of the BNN are drawn from $q(\boldsymbol{w}_{jk}) = \mathcal{N}(\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1})$.

The structured variational distribution is defined as follows

$$q(\phi) = \prod_{k=1}^{K} q(v_k^0) \prod_{j=1}^{J} q(\pi_{jk}|v_k^0)q(\boldsymbol{w}_{jk}) \prod_{i=1}^{N} q(z_{ijk}|\pi_{jk}) \quad (6)$$

---

[1]The mean-field approximation removes all edges from these graphical models.

where $\phi = \{\alpha_k^0, \beta_k^0, \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}\}$ for all j and k, up to the variational truncation, K. Having defined the structured variational distribution, the negative ELBO is:

$$\begin{aligned}
\mathcal{L}(\phi, \mathcal{D}) = \,& \text{KL}(q(\boldsymbol{v}^0)||p(\boldsymbol{v}^0)) + \sum_{j=1}^{J} \text{KL}(q(\boldsymbol{\pi}_j|\boldsymbol{v}^0)||p(\boldsymbol{\pi}_j|\boldsymbol{v}^0)) \\
& + \sum_{j=1}^{J} \text{KL}(q(\boldsymbol{w}_j)||p(\boldsymbol{w}_j)) \\
& - \sum_{j=1}^{J} \sum_{i=1}^{|\mathcal{D}|} \mathbb{E}_{q(\phi)}[\log p(y_i|\mathbf{x}_i, \mathbf{z}_{ij}, \boldsymbol{w}_j)] \\
& + \sum_{j=1}^{J} \sum_{i=1}^{|\mathcal{D}|} \text{KL}(q(\mathbf{z}_{ij}|\boldsymbol{\pi}_j)||p(\mathbf{z}_{ij}|\boldsymbol{\pi}_j)).
\end{aligned}$$

$$(7)$$

The child stick-breaking variational parameters for each layer are conditioned on the global stick-breaking parameters and the binary masks z_{ijk} for each neuron k in each layer j are conditioned on the child stick-breaking variational parameters. Thus, the variational structured posterior is able to capture dependencies of the prior. The learnable parameters are α_k^0, β_k^0, $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\sigma}_{jk}$ for all k neurons and for all layers j.

3.2.1 Inference

The variational posterior is obtained by optimising Equation (7) using structured stochastic VI. For inference to be tractable, we utilise three reparameterisations. The first is for the Gaussian weights [Kingma and Welling, 2014]. The second is an implicit reparameterisation of the Beta distribution [Figurnov et al., 2018]. The third reparameterisation uses a Concrete relaxation to the Bernoulli distribution [Maddison et al., 2017, Jang et al., 2017]. Details of these are in the Supplementary material, Section A.

4 RELATED WORKS

IBP priors
An IBP prior has been used in VAEs to automatically learn the number of latent features. Stick-breaking probabilities have been placed directly as the VAE latent state [Nalisnick and Smyth, 2017]. The IBP prior has been used to learn the number of features in a VAE hidden state using mean-field VI [Chatzis, 2018] with black-box VI [Ranganath et al., 2014] and structured VI [Hoffman and Blei, 2015, Singh et al., 2017]. As an alternative to truncation, [Xu et al., 2019] use a Russian roulette sampling scheme to sample from the infinite sum in the stick-breaking process for the IBP. Model selection for BNNs has been performed with the Horse-shoe prior over weights [Ghosh et al., 2019].

The IBP prior has been employed in BNNs to induce sparcity [Panousis et al., 2019] and simultaneously to our work for CL [Kumar et al., 2019]. Both of these approaches apply the IBP prior differently to our work and previous work applying the IBP to VAEs. Also [Kumar et al., 2019] deviates from sequential Bayes by storing masks over weights for each task and uses design choices which mean the IBP is not the sole means of selecting weights, we expand on this in Section G. Instead of using an IBP prior, Bernoulli distributions (or its Concrete relaxation) is used to select the width and number of layers in a BNN [Dikov et al., 2019]. This approach is not non-parametric and so not as desirable for CL. Recently, and subsequently to our work [Mehta et al., 2020] have proposed a CL approach where weights matrices are factorised and the IBP prior applied to the diagonal in the factorisation. This work does not use sequential Bayes for CL but rather different neural network weights are used for different tasks and thus alleviates forgetting.

Bayesian. Repeated application of Bayes' rule can be used to update a model given the arrival of a new task. Previous work has used Laplace approximations [Kirkpatrick et al., 2017, Ritter et al., 2018] and variational inference [Nguyen et al., 2018, Zeno et al., 2018, Ahn et al., 2019]. Bayesian methods can also be intuitively thought of as a weight space regularisation. Explicit regularisation in weight space have also proved successful in CL [Zenke et al., 2017, Chaudhry et al., 2018, Schwarz et al., 2018]. Our method builds upon [Nguyen et al., 2018] as the framework for learning continually. None of these works deal with the issue of resource allocation to alleviate potential overfitting or underfitting problems in CL. The model we present adapts its size for CL, the scenario where BNNs need adapt to a changing data distribution or to *concept drift* in CL has been studied too [Kurle et al., 2020].

Adding. Non-Bayesian CL approaches use additional neural resources to learn new tasks and remember previous tasks. One approach boils down to learning individual networks for each task [Rusu et al., 2016]. More efficient use of resources can be done by selective retraining of neurons and expansion with a group sparsity regulariser [Yoon et al., 2018]. However this approach is unable to shrink and continues to expand if it overfits on the first task. Another approach uses reinforcement learning by adding neural resources by penalising the complexity of the task network in the reward function [Xu and Zhu, 2018]. Recently [Rao et al., 2019] propose an unsupervised CL model (CURL) in scenarios
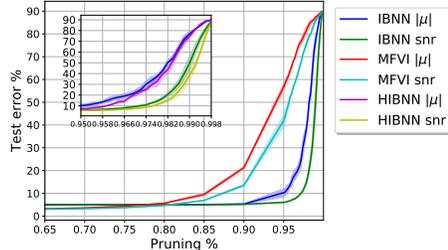

Figure 2: The weight pruning curves show test error versus the percentage of weights which have been zeroed out according to the magnitude of the variational mean and snr ($|\mu|/\sigma$). The HIBNN and IBNN are much sparser than a MFVI BNN and the HIBNN is more robust to pruning and therefore sparser than the IBNN.
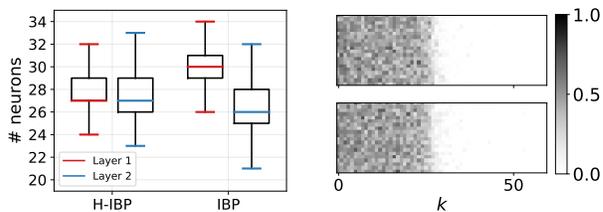

Figure 3: The number of active neurons in each layer of the IBNN and HIBNN. The HIBNN introduces an inductive bias which encourages and enables a regular structure. The Z matrices for the HIBNN show the neurons which are being learnt on the left[2].

that lack a task identifier. CURL is adaptive, insofar that if a new task is detected then a new component is added to the mixture of Gaussians in the model. Task learning in CL can be modelled as a mixture of expert models. The experts are distributed according to a Dirichlet prior [Lee et al., 2020]; new experts can be added to the mixture automatically.

5 EXPERIMENTAL RESULTS

To demonstrate the effectiveness of the IBP and H-IBP priors on determining the size of the BNN, we perform weight pruning to see whether the pruned weights coincide with the weights dropped by the IBP and H-IBP priors in Section 5.1. Furthermore, we then use the IBNN and HIBNN in a CL setting in Section 5.2. Also in the supplementary material, further continual learning results and all experimental details are outlined. Unless explicitly stated, all curves are an average of 5 independent runs $\pm$ one standard error. By test error, we refer to $(1 - \text{accuracy}) \times 100$[3].

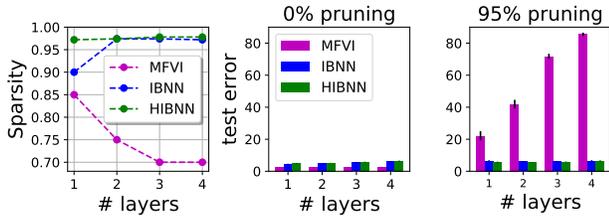[3]Our code is available at `https://github.com/xxx` (released upon publication)

Figure 4: **Left**, as the depth of the IBNN and HIBNN increases the networks tend to remain very sparse while the MFVI BNN is becomes less sparse. **Middle** and **Right**, the HIBNN and IBNN remain robust to pruning with increasing depth.

5.1 BNN

We perform weight pruning to see whether the IBP posterior sensibly selects neurons through the binary matrix Z. Weights are pruned in two ways. The first is pruning according to $|\mu|$: zeroing out weights according to the magnitude of their mean. Important weights will be large in absolute value and so pruned last. Secondly, according to signal to noise ratio: $|\mu|/\sigma$ (snr). Weights with high uncertainty will also be zeroed out first. Weight pruning is performed on MNIST and compared to a mean-field BNN [Blundell et al., 2015] (denoted MFVI in plots). The pruning accuracies in Figure 2 demonstrate that the HIBNN is indeed much sparser than a BNN and that pruning according to snr is more robust, as expected. The HIBNN is more robust to pruning than the IBNN due to its inductive bias leading to the more regular structure Figure 3. The baseline BNN has two layers with hidden state sizes of 200, the HIBNN and IBNN use a variational truncation of $K = 200$ for fair comparison. The HIBNN and IBNN achieves an accuracy of 0.95 before pruning while MFVI achieves 0.98 before pruning. This gap in performance is due to the approximate inference of the H-IBP and IBP posteriors and the various reparameterisations used, in particular the Concrete reparameterisation which is applying a 'soft' mask on the hidden layers of the HIBNN. The IBNN and HIBNN are slightly less sparse compared to Sparse Variational Dropout which is specifically designed to be sparse [Molchanov et al., 2017], see Section B in the supplement.

Depth As we increase the depth we see that the HIBNN and IBNN remain sparse while the MFVI's sparsity decreases, Figure 4. We measure the sparsity as the pruning percentage at which the accuracy drops over 10% (the kinks in Figure 4). There is little variation of the accuracy with depth for all models before pruning, however after pruning 95% of the weights with the snr our models retain their performance while the
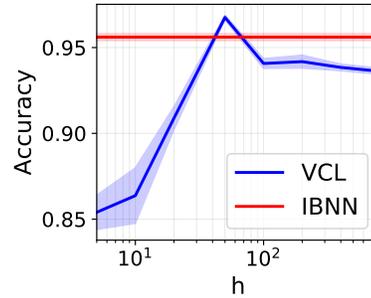

Figure 5: Test accuracies for Permuted MNIST `CL1`, for different VCL widths. Our model yields good results without having to specify a width, only a small range of values outperforms our model.

MFVI BNN performs worse with depth since it becomes less sparse with depth Figure 4. See the supplement for the same analysis on fashion-MNIST, Section C.2.

5.2 Continual learning

Aim Approximate inference of the IBP and H-IBP posteriors is challenging in a stationary setting making the performance attenuated in comparison to a BNN with only independent Gaussian weights. Despite this, the approximate IBP and H-IBP posteriors are useful in non-stationary CL setting, where the amount of resources are unknown beforehand. In Figure 5, one can see that the average accuracies across all CL tasks for permuted MNIST vary considerably with the hidden state size for VCL hence the benefit of our model which automatically infers the hidden state size for each task, see Figure 6. The details of the experiment will be introduced below.

Choice of scenarios Three different CL scenarios are used for evaluation [van de Ven and Tolias, 2018]. The first is *task incremental learning* (`CL1`) where the task identifier is given during evaluation. The second is *domain incremental learning* (`CL2`), the task identifier needs to be inferred at test time. The domain increases with each new task and the models are required to perform binary classification. The third is *incremental class learning* (`CL3`), the task identifier and specific class need to be inferred at test time. The models are required to do multi-class classification for each task. During training, the task identifiers are given for all scenarios[4]. Using a multi-head architecture and the predictive entropy we can infer the task; the head with the lowest predictive entropy is chosen for `CL2` and `CL3` [Von Oswald et al., 2020]. For

---

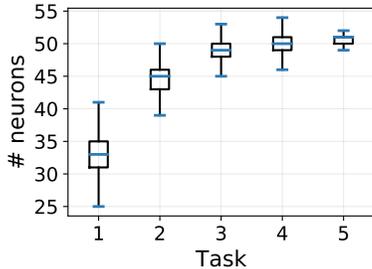[4] *Task-free CL* is a more challenging scenario and requires the model the infer new tasks during training.

Figure 6: Number of active neurons[5]selected by the IBP variational posterior for each task, for permuted MNIST CL1. Our model is able adapt and manage resources effectively.

Permuted MNIST CL2 a single-head architecture is used [van de Ven and Tolias, 2018].

Baselines We compare our models to VCL [Nguyen et al., 2018], since the IBNN and HIBNN models build on top of it. We also compare to EWC [Kirkpatrick et al., 2017] and SI [Zenke et al., 2017]. We compare to DEN [Yoon et al., 2018] which is an expansion method which expands a neural network by a fixed number of weights for each new task, uses regularisation to mitigate overfitting and freezes weights from previous tasks. For DEN the predictive entropy is used for inferring the correct head for CL2 and CL3. Another baseline used is GEM [Lopez-Paz and Ranzato, 2017], which uses replay as its primary mechanism for alleviating forgetting. We hypothesise that GEM will be insensitive to model size. It also achieves strong results [Hsu et al., 2018]. VCL is the fairest comparison for our model and will also utilise uncertainties for CL2 and CL3. Our objective is to demonstrate that limited or excessive neural resources can cause problems in CL in comparison to our adaptive model.

Model All models use a single layer with varying hidden state sizes. The use of a single layer is enough as MNIST is a simple task. The results for EWC in Table 1 on Split MNIST outperform those presented in [Hsu et al., 2018, van de Ven and Tolias, 2018] which use larger models. We report accuracies for our non adaptive baselines (EWC, SI, GEM and VCL) over a set of hidden state sizes $\mathcal{H} = \{10, 50, 100, 400\}$. A hidden state of 10 might seem small but we also set the IBP prior parameter $\alpha = 5$ for task 1. This corresponds to only 5 neurons being selected by each data point in expectation. The initial hidden state size for DEN is set to 50.

Permuted MNIST The permuted MNIST benchmark involves performing multiclass classification on MNIST where in each task, the pixels have been shuffled by a fixed permutation. Our model is able to overcome overfitting and underfitting which result in increased forgetting which affect VCL. See Figure 15 in the supplement, for a per task accuracy breakdown. In contrast, the IBNN expands continuously Figure 6. There is a small gap in performance between our model and VCL for $h = 50$ due to the approximations used for inference of the variational IBP posterior. Our method outperforms all regularisation based methods and DEN[6] on all CL scenarios and provides comparable results to GEM, see Table 1.

Split MNIST The split MNIST benchmark for CL involves a sequence of classification tasks of MNIST and more difficult variants with background noise and background images denoted S+ϵ and S+img. For EWC, SI and VCL notice a considerable difference in performance with hidden state size in Table 1. GEM is sensitive for CL3 only. EWC and SI perform well for CL1 only. The IBNN outperforms VCL as it is not susceptible to overfitting or underfitting and thus propagating a subsequent poor posterior for a new task resulting in forgetting.

Split MNIST is a simple task which doesn't show overfitting, hence the use of the MNIST variant datasets where the IBNN outperforms all VCL models of different sizes as it is not susceptible to overfitting or underfitting. Indeed our method outperforms not only VCL but EWC and SI and performs comparably to GEM. When analysing the performance of the VCL baselines, we notice they have a tendency to overfit on the second task and propagate a poor approximate posterior and hence underperform in comparison to the IBNN model, Figures 13 and 16 in the supplement. The IBNN increases its capacity over the course of CL, Figure 14 in the supplement. The standard errors for VCL and our method on CL3 are large due to the severity of making mistakes for multiclass classification.

DEN performs well on all Split MNIST tasks and variants due to its "time-stamped inference" which segregates parts of the network per task and so uncertainties over seen tasks are well defined thus the good results for CL2 and CL3. Indeed removing it renders performance comparable to IBNN, see Section D in the supplement. Statistical processes which mimic this could be an interesting direction for Bayesian expansion methods in CL.

---

[5]We define a neuron as active by aggregating all neurons where $z_{ik} > 0.1$ for data point $\mathbf{x_i}$ and neuron k.

[6]It is not clear how to reconcile single-head networks and DEN, thus CL2 for DEN is omitted.

Table 1: Average test accuracies on MNIST and variants over 5 runs. For EWC, SI, GEM and VCL the median accuracy is taken from hidden state sizes, $\mathcal{H} = \{10, 50, 100, 400\}$. We also show the range between max and min average accuracies in $\mathcal{H}$. The models with the best median or mean accuracy are highlighted. If the IBNN mean accuracy lies within the min-max range then our model is also highlighted. Our IBNN achieves good performance overall compared to the baselines which can underfit/overfit. DEN performs very poorly on permuted MNIST.

	EWC (max, min)	SI (max, min)	GEM (max, min)	DEN	VCL (max, min)	IBNN
P CL1	90.0 (94.0, 79.1)	91.8 (95.1, 82.9)	94.8 (95.9, 88.4)	91.4±0.5	93.9 (96.8, 86.9)	**95.6±0.2**
P CL2	88.2 (93.0, 78.2)	89.2 (93.3, 84.1)	**95.6 (96.7, 88.1)**	-	88.7 (95.1, 75.2)	**93.7±0.6**
P CL3	59.3 (66.9, 24.0)	47.5 (55.4, 18.1)	**94.6 (95.8, 87.3)**	63.9±19.2	84.0 (94.1, 40.5)	**93.8±0.3**
S CL1	98.9 (99.0, 94.8)	98.1 (99.2, 95.5)	98.1 (98.2, 98.0)	**99.1±0.1**	96.6 (98.0, 93.7)	95.3±2.0
S CL2	63.7 (74.8, 63.3)	78.0 (80.1, 72.9)	94.0 (94.8, 92.2)	**98.9±0.1**	87.4 (94.9, 81.9)	91.0±2.2
S CL3	19.9 (21.8, 19.8)	18.8 (19.6, 15.4)	89.2 (89.6, 87.8)	**99.1±0.1**	69.0 (78.9, 66.3)	85.5±3.2
S+ϵ CL1	94.6 (96.5, 81.8)	88.4 (91.1, 72.7)	95.6 (95.7, 95.1)	**97.2±0.2**	89.2 (90.3, 86.3)	95.1±1.1
S+ϵ CL2	72.7 (74.6, 68.7)	66.1 (72.1, 61.8)	78.2 (78.7, 77.2)	84.8±16.7	69.1 (70.2, 61.7)	**89.7±3.8**
S+ϵ CL3	18.8 (19.0, 13.7)	13.7 (16.7, 10.4)	74.1 (74.4, 70.7)	**90.9±12.8**	32.5 (37.8, 30.0)	78.7±11.7
S+img CL1	88.2 (90.0, 78.1)	80.8 (87.4, 67.8)	91.7 (91.8, 91.1)	**93.8±0.8**	87.1 (87.9, 85.7)	91.6±1.2
S+img CL2	67.7 (73.9, 62.9)	63.3 (67.0, 58.5)	**74.1 (76.4, 74.0)**	79.1±13.1	70.4 (75.2, 65.3)	**80.5±7.8**
S+img CL3	17.0 (18.0, 12.0)	13.4 (16.4, 10.3)	63.9 (64.5, 56.3)	**91.6±5.1**	54.3 (66.1, 39.9)	66.2±13.4

Table 2: Accuracies for tasks of increasing difficulty. Accuracies are an average over 5 runs. VCL use different hidden state sizes, $\mathcal{H} = \{50, 100, 400\}$, we show the range between the max and min average accuracies centre at the median. Our models are able to overcome underfitting.

	VCL (max, min)	IBNN	HIBNN
CL1	**82.3 (86.1, 81.2)**	80.8±2.0	**81.3±1.8**
CL2	79.9 (83.1, 76.7)	78.7±1.5	**81.5±1.4**

Figure 7: Inferred structure for the HIBNN and IBNN respectively. Our models can infer varied structures, expand and contract.

To test the expansion capabilities of our models we devise a set tasks of increasing difficulty: two tasks from MNIST, followed by two from fashion MNIST, followed by two from CIFAR10. We compare the HIBNN and IBNN models to VCL with two layers and widths in $\mathcal{H} = \{50, 100, 400\}$. Our models have two layers with $K = 200$. The larger width VCL networks perform well but smaller ones exhibit forgetting due to underfitting. For the HIBNN we allow the hyperparameter α_j to increase for each new dataset seen i.e. every two tasks. As new tasks are seen we should add more neural resources and this can be controlled by the α_j hyperparameter of the HIBNN. We perform random search over the IBP and H-IBP parameters for the IBNN and HIBNN models, Section F.1. The HIBNN performs as well or better than VCL, also the IBNN and HIBNN can have very different structure after performing all tasks as a result.
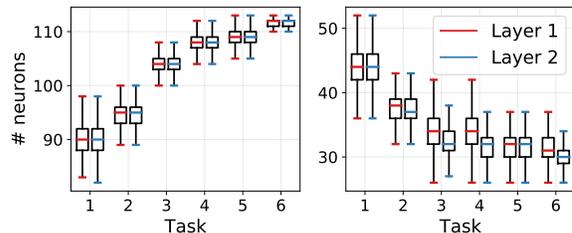
6 CONCLUSION

Model size is an important contributing factor for CL performance. Most CL methods assume a perfectly selected model. Our novel Bayesian CL framework nonparametrically adapts the complexity of a BNN to the task difficulty. Our model is based on the IBP prior for selecting the number of neurons for each task and uses stochastic variational inference. The models presented reconcile two different approaches to CL: Bayesian or regularisation based approaches and dynamic architecture approaches through the use of a IBP and H-IBP prior. Solutions which adapt resources, to prevent overfitting or underfitting, are essential for successful Bayesian CL approaches to avoid poor approximate posteriors being used as priors for a new task. This requirement provides our core motivation for a Bayesian model which adds and removes resources automatically.

7 ACKNOWLEDGEMENTS

R.

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA. USENIX Association.

[Ahn et al., 2019] Ahn, H., Lee, D., Cha, S., and Moon, T. (2019). Uncertainty-based Continual Learning with Adaptive Regularization. In *Neurips*.

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

[Chatzis, 2018] Chatzis, S. P. (2018). Indian Buffet Process deep generative models for semi-supervised classification. Technical report.

[Chaudhry et al., 2018] Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. S. (2018). Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *European Conference on Computer Vision (ECCV)*.

[Dikov et al., 2019] Dikov, G., van der Smagt, P., and Bayer, J. (2019). Bayesian Learning of Neural Network Architectures. In *AISTATS*.

[Dillon et al., 2017] Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., and Saurous, R. A. (2017). TensorFlow Distributions.

[Doshi et al., 2009] Doshi, F., Miller, K., Van Gael, J., and Teh, Y. W. (2009). Variational inference for the Indian Buffet Process. In *Artificial Intelligence and Statistics*, pages 137–144.

[Doshi-Velez et al., 2009] Doshi-Velez, F., Miller, K. T., Van Gael, J., Teh, Y. W., Van, J., Yee, G. ., and Teh, W. (2009). Variational Inference for the Indian Buffet Process. Technical report.

[Figurnov et al., 2018] Figurnov, M., Mohamed, S., and Mnih, A. (2018). Implicit Reparameterization Gradients. In *Neural Information Processing Systems*.

[Ghosh et al., 2019] Ghosh, S., Yao, J., and Doshi-Velez, F. (2019). Model Selection in Bayesian Neural Networks via Horseshoe Priors. *Journal of Machine Learning Research*, 20(182):1–46.

[Goodfellow et al., 2015] Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2015). An empirical investigation of catastrophic forgetting in gradient-based neural networks.

[Griffiths and Ghahramani, 2011] Griffiths, T. L. and Ghahramani, Z. (2011). The Indian Buffet Process: An Introduction and Review. *Journal of Machine Learning Research*, 12:1185–1224.

[Gupta et al., 2012] Gupta, S. K., Phung, D., and Venkatesh, S. (2012). A Slice Sampler for Restricted Hierarchical Beta Process with Applications to Shared Subspace Learning. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 316–325.

[Hoffman and Blei, 2015] Hoffman, M. D. and Blei, D. M. (2015). Structured Stochastic Variational Inference. In *International Conference on Artificial Intelligence and Statistics*.

[Hsu et al., 2018] Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. (2018). Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. In *Continual Learning Workshop, 32nd Conference on Neural Information Processing Systems*.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[Jang et al., 2017] Jang, E., Gu, S., and Poole, B. (2017). Categorical Reparametrization with Gumbel-Softmax. In *International Conference on Learning Representations*.

[Jankowiak and Obermeyer, 2018] Jankowiak, M. and Obermeyer, F. (2018). Pathwise derivatives beyond the reparameterization trick. In *International Conference on Machine Learning*, pages 2235–2244.

[Kingma and Lei Ba, 2015] Kingma, D. P. and Lei Ba, J. (2015). ADAM: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[Kingma et al., 2015] Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational Dropout and the Local Reparameterization Trick. In *Neural Information Processing Systems*.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

[Kirkpatrick et al., 2017] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

[Kumar et al., 2019] Kumar, A., Chatterjee, S., and Rai, P. (2019). Bayesian Structure Adaptation for Continual Learning. Technical report.

[Kurle et al., 2020] Kurle, R., Cseke, B., Klushyn, A., Van Der Smagt, P., and Günnemann, S. (2020). Continual Learning with Bayesian Neural Networks for Non-Stationary Data. In *ICLR*.

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

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

[Louizos et al., 2017] Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian Compression for Deep Learning. In *Neural Information Processing Systems*.

[Maddison et al., 2017] Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The Concrete Distribution: a Continual Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*.

[Mehta et al., 2020] Mehta, N., Liang, K. J., and Carin, L. (2020). Bayesian Nonparametric Weight Factorization for Continual Learning. Technical report.

[Mohamed et al., 2019] Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. (2019). Monte carlo gradient estimation in machine learning. *ArXiv*, abs/1906.10652.

[Molchanov et al., 2017] Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational Dropout Sparsifies Deep Neural Networks. In *International Conference on Machine Learning*.

[Nalisnick and Smyth, 2017] Nalisnick, E. and Smyth, P. (2017). Stick-Breaking Variational Autoencoders. In *International Conference on Learning Representations*.

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

[Panousis et al., 2019] Panousis, K. P., Chatzis, S., and Theodoridis, S. (2019). Nonparametric Bayesian deep networks with local competition. *36th International Conference on Machine Learning, ICML 2019*, 2019-June(May):8772–8783.

[Ranganath et al., 2014] Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black Box Variational Inference. In *Artificial Intelligence and Statistics*.

[Rao et al., 2019] Rao, D., Visin, F., Rusu, A. A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2019). Continual Unsupervised Representation Learning. In *Neural Information Processing Systems*.

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

[Rusu et al., 2016] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive Neural Networks. In *arXiv:1606.04671v3*.

[Schwarz et al., 2018] Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress and Compress: A scalable framework for continual learning. In *International Conference on Machine Learning*.

[Singh et al., 2017] Singh, R., Ling, J., and Doshi-Velez, F. (2017). Structured Variational Autoencoders for the Beta-Bernoulli Process. In *Workshop on Advances in Approximate Bayesian Inference, Neural Information Processing Systems*.

[Swaroop et al., 2018] Swaroop, S., Nguyen, C. V., Bui, T. D., and Turner, R. E. (2018). Improving and Understanding Variational Continual Learning. In *Continual Learning workshop, Neural Information Processing Systems*.

[Teh et al., 2007] Teh, Y. W., Grür, D., and Ghahramani, Z. (2007). Stick-breaking Construction for the Indian Buffet Process. In *International Conference on Artificial Intelligence and Statistics*.

[Thibaux and Jordan, 2007] Thibaux, R. and Jordan, M. I. (2007). Hierarchical Beta processes and the Indian buffet process. In *Artificial Intelligence and Statistics*, pages 564–571.

[van de Ven and Tolias, 2018] van de Ven, G. M. and Tolias, A. S. (2018). Three scenarios for continual learning. In *NeurIPS Continual Learning workshop*.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

[Von Oswald et al., 2020] Von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. (2020). Continual Learning with Hypernetworks. In *International Conference on Learning Representations*.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[Xu and Zhu, 2018] Xu, J. and Zhu, Z. (2018). Reinforced Continual Learning. In *Neural Information Processing Systems*.

[Xu et al., 2019] Xu, K., Srivastava, A., and Sutton, C. (2019). Variational Russian Roulette for Deep Bayesian Nonparametrics. In *Proceedings of the 36th International Conference on Machine Learning*.

[Yoon et al., 2018] Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2018). Lifelong Learning with Dynamically Expandable Networks. In *International Conference on Learning Representations*.

[Zenke et al., 2017] Zenke, F., Poole, B., and Ganguli, S. (2017). Continual Learning Through Synaptic Intelligence. In *International Conference on Machine Learning*.

[Zeno et al., 2018] Zeno, C., Golan, I., Hoffer, E., and Soudry, D. (2018). Task Agnostic Continual Learning Using Online Variational Bayes. In *Bayesian Deep Learning workshop, Neural Information Processing Systems*.

APPENDIXES

A   MODEL AND INFERENCE DETAILS

In this section, we present the variational BNN with the structure of the hidden layer determined by the H-IBP variational posterior (a.k.a. HIBNN). It is straightforward to apply the following methodology for a simpler model with independent IBP variational posteriors determining the structure of each hidden layer (referred to the IBNN in the main paper).

We derive a structured variational posterior where dependencies are established between global parameters and local parameters [Hoffman and Blei, 2015]. Once the variational posterior is obtained, we can follow the VCL framework [Nguyen et al., 2018] for CL. The following set of equations govern the stick-breaking H-IBP model for an arbitrary layer $j \in \{1, ..., J\}$ with k neurons in each layer of a BNN:

$$v_k \sim \text{Beta}\,(\alpha, 1)\,, \qquad\qquad \text{for } k = 1, \ldots, \infty, \tag{8}$$

$$\pi_k^0 = \prod_{i=1}^{k} v_i, \qquad\qquad \forall\, k, \tag{9}$$

$$\pi_{jk} \sim \text{Beta}(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)), \qquad\qquad \forall\, k, j = 1, \ldots, J, \tag{10}$$

$$z_{ijk} \sim \text{Bern}(\pi_{jk}), \qquad\qquad \forall\, j, k, i = 1, \ldots, N, \tag{11}$$

$$\boldsymbol{w}_{jk} \sim \mathcal{N}(\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1}), \qquad\qquad \forall\, j, k, \tag{12}$$

$$h_{jk} = f(\boldsymbol{h}_{j-1} \boldsymbol{w}_{jk}) \circ z_{ijk} \qquad\qquad \forall\, i, j, k. \tag{13}$$

The index k denotes a particular neuron, j denotes a particular layer, and $\boldsymbol{w}_{jk}$ denotes a column (of W_j) from the weight matrix mapping hidden states from layer $j - 1$ to layer j, such that $h_j = f(h_{j-1} W_j) \circ Z_j$. We denote $\circ$ as the elementwise multiplication operation. The binary matrix Z_j controls the inclusion of a particular neurons in layer j. The dimensionality of our variables are as follows $\boldsymbol{w}_{jk} \in \mathbb{R}^{k_{j-1}}$, $h_{j-1} \in \mathbb{R}^{k_{j-1}}$, $h_j \in \mathbb{R}^{k_j}$ and $z_{ijk} \in \mathbb{Z}_2 = \{0, 1\}$.

The closed-form solution to the true posterior of the H-IBP parameters and BNN weights involves integrating over the joint distribution of the data and hidden variables, $\phi = \{\boldsymbol{Z}, \boldsymbol{\pi}, \boldsymbol{\pi^0}, \boldsymbol{w}\}$. Since it is not possible to obtain a closed-form solution to this integral, variational inference together with reparameterisations of the variational distributions are used [Kingma and Welling, 2014, Figurnov et al., 2018] to employ gradient based methods. The variational approximation used is

$$q(\phi) = \prod_{k=1}^{K} q(v_k^0; \alpha_k^0, \beta_k^0) \prod_{j=1}^{J} q(\pi_{jk}|v_k^0) q(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}) \prod_{i=1}^{N} q(z_{ijk}|v_k^0), \tag{14}$$

where the variational posterior is truncated up to K, the prior is still infinite [Nalisnick and Smyth, 2017]. The set of variational parameters which we optimise over are $\boldsymbol{\varphi} = \{\boldsymbol{\alpha^0}, \boldsymbol{\beta^0}, \boldsymbol{\mu}, \boldsymbol{\sigma}\}$ and $q(\phi)$ is the variational distribution. Each term in Equation (14) is specified as follows

$$q(v_k^0; \alpha_k^0, \beta_k^0) = \text{Beta}(v_k; \alpha_k^0, \beta_k^0), \tag{15}$$

$$\pi_k^0 = \prod_{i=1}^{k} v_i^0, \tag{16}$$

$$q(\pi_{jk}|v_k^0) = \text{Beta}(\pi_{jk}; \alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)) \tag{17}$$

$$q(z_{ijk}|v_k^0) = \text{Bern}(z_{ijk}; \pi_{jk}), \tag{18}$$

$$q(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}) = \mathcal{N}(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1}), \tag{19}$$

where α_j are hyperparameters [Gupta et al., 2012]. Now that we have defined our structured variational approximation in Equation (14) we can write down the objective

$$\arg\min D_{\text{KL}}\,(q(\phi)||p(\phi|\mathcal{D})) = \arg\min D_{\text{KL}}\,(q(\phi)||p(\phi)) - \mathbb{E}_{q(\phi)}[\log p(\mathcal{D}|\phi)]. \tag{20}$$

In the above formula, $q(\phi)$ is the approximate posterior and $p(\phi)$ is the prior. By substituting Equation (14), we obtain the negative ELBO objective:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\phi}, \mathcal{D}) = {}& \mathrm{KL}(q(\boldsymbol{v^0})||p(\boldsymbol{v^0})) + \sum_{j=1}^{J} \mathrm{KL}(q(\boldsymbol{\pi}_j|v^0)||p(\boldsymbol{\pi}_j|v^0)) + \mathrm{KL}(q(\boldsymbol{w_j})||p(\boldsymbol{w_j})) \\
& - \sum_{j=1}^{J}\sum_{i=1}^{|\mathcal{D}|} \mathbb{E}_{q(\boldsymbol{\phi})}[\log p(y_i|\boldsymbol{x_i}, \boldsymbol{z_{ij}}, \boldsymbol{w_j})] + \sum_{j=1}^{J}\sum_{i=1}^{|\mathcal{D}|} \mathrm{KL}(q(\boldsymbol{z_{ij}}|\boldsymbol{\pi_j})||p(\boldsymbol{z_{ij}}|\boldsymbol{\pi_j})).
\end{aligned}
\tag{21}
$$

Estimating the gradient of the Bernoulli and Beta variational parameters requires a suitable reparameterisation. Samples from the Bernoulli distribution in Equation (18) arise after taking an `argmax` over the Bernoulli parameters. The `argmax` is discontinuous and a gradient is undefined. The Bernoulli is reparameterised as a Concrete distribution [Maddison et al., 2017, Jang et al., 2017]. Additionally the Beta is reparameterised implicitly [Figurnov et al., 2018] to separate sampling nodes and parameter nodes in the computation graph and allow the use of stochastic gradient methods to learn the variational parameters φ of the approximate H-IBP posterior. The mean-field approximation for the Gaussian weights of the BNN is used, w in Equation (19) [Blundell et al., 2015, Nguyen et al., 2018]. In the next sections we detail the reparameterisations used to optimise Equation (21).

A.1 BGSihn

The variational posterior over the weights of the BNN are diagonal Gaussian $\boldsymbol{w}_{jk} \sim \mathcal{N}(\boldsymbol{w}_{jk}|\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1})$. By using a reparameterisation, one can represent the BNN weights using a deterministic function $\boldsymbol{w}_{jk} = g(\epsilon; \boldsymbol{\varphi})$, where $\epsilon \sim \mathcal{N}(0, \mathbb{1})$ is an auxiliary variable and $g(\epsilon; \boldsymbol{\varphi})$ a deterministic function parameterised by $\boldsymbol{\varphi} = (\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk})$, such that:

$$
\nabla_\varphi \mathbb{E}_{q(\boldsymbol{w}_{jk};\boldsymbol{\varphi})} \left[f(\boldsymbol{w}_{jk}) \right] = \mathbb{E}_{q(\epsilon)} \left[\nabla_\varphi f(\boldsymbol{w}_{jk}) \right]|_{\boldsymbol{w}_{jk}=g(\epsilon;\boldsymbol{\varphi})},
\tag{22}
$$

where f is an objective function, for instance Equation (21). The BNN weights can be sampled directly through the reparameterisation: $\boldsymbol{w}_{jk} = \boldsymbol{\mu}_{jk} + \boldsymbol{\sigma}_{jk}\epsilon$. By using this simple reparameterisation the weight samples are now deterministic functions of the variational parameters $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\sigma}_{jk}$ and the noise comes from the independent auxiliary variable ϵ [Kingma and Welling, 2014]. Taking a gradient of our ELBO objective in Equation (21) the expectation of the log-likelihood may be rewritten by integrating over ϵ so that the gradient with respect to $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\sigma}_{jk}$ can move into the expectation according to Equation (22) (the backward pass). In the forward pass, $\epsilon \sim q(\epsilon)$ is sampled to compute $\boldsymbol{w}_{jk} = \boldsymbol{\mu}_{jk} + \boldsymbol{\sigma}_{jk}\epsilon$.

A.2 BBeha

Implicit reparameterisation gradients [Figurnov et al., 2018, Mohamed et al., 2019] is used to learn the variational parameters for Beta distribution. [Nalisnick and Smyth, 2017] propose to use a Kumaraswamy reparameterisation. However, in CL a Beta distribution rather than an approximation is desirable for repeated Bayesian updates.

There is no simple inverse of the reparameterisation for the Beta distribution like the Gaussian distribution presented earlier. Hence, the idea of implicit reparameterisation gradients is to differentiate a standardisation function rather than have to perform its inverse. The standardisation function is given by the Beta distribution's CDF.

Following [Mohamed et al., 2019], the derivative required for general stochastic VI is:

$$
\begin{aligned}
\nabla_\varphi \mathbb{E}_{q(v^0;\varphi)} \left[f(v^0) \right] &= \mathbb{E}_{q(\epsilon)} \left[\nabla_\varphi f(v^0) \right]|_{v^0=g(\epsilon;\varphi)} \\
&= \mathbb{E}_{q(v^0;\varphi)} [\nabla_{v^0} f(v^0) \nabla_\varphi v^0],
\end{aligned}
\tag{23}
$$

where f is an objective function, i.e. Equation (21), $\varphi = \{\alpha_k^0, \beta_k^0\}$ are the Beta parameters. The implicit reparameterisation gradient solves for $\nabla_\varphi v^0$, above, using implicit differentiation [Figurnov et al., 2018]:

$$
\nabla_\varphi v^0 = -(\nabla_z g_\varphi^{-1}(v^0;\varphi))^{-1} \nabla_\varphi g_\varphi^{-1}(v^0;\varphi),
\tag{24}
$$

where $v^0 = g(\epsilon; \varphi)$, $g(.)$ is the inverse CDF of the Beta distribution and $\epsilon \sim \text{Unif}[0, 1]$ and so $\epsilon = g^{-1}(v^0; \varphi)$ is the standardisation path which removes the dependence of the distribution parameters on the sample. The key idea in implicit reparameterisation is that this expression for the gradient in Equation (24) only requires differentiating the standardisation function $g_\varphi^{-1}(v^0; \varphi)$ and not inverting it. Given $v^0 = g(\epsilon; \varphi) = F^{-1}(v^0; \varphi)$ then using Equation (24) the implicit gradient is:

$$\begin{aligned}
\nabla_\varphi v^0 &= \nabla_\varphi F^{-1}(v^0; \varphi) \\
&= \frac{\nabla_\varphi F(v^0; \varphi)}{p_\varphi(v^0; \varphi)},
\end{aligned} \tag{25}$$

where $p_\varphi(v^0; \varphi)$ is the Beta PDF. The CDF of the Beta distribution, is given by

$$F(v^0; \varphi) = \frac{B(v^0; \varphi)}{B(\varphi)} \tag{26}$$

where $B(v^0; \varphi)$ and $B(\varphi)$ are the incomplete Beta function and Beta function, respectively. The derivatives of $\nabla_\varphi F(v^0; \varphi)$ do not admit simple analytic expressions. Thus, numerical approximations have to be made, for instance, by using Taylor series for $B(v^0; \varphi)$ [Jankowiak and Obermeyer, 2018, Figurnov et al., 2018].

In the forward pass, $v^0 \sim q_\varphi(v^0)$ is sampled from a Beta distribution or alternatively from two Gamma distributions. That is, if $v_1^0 \sim \text{Gamma}(\alpha_k^0, 1)$ and $v_2^0 \sim \text{Gamma}(\beta_k^0, 1)$, then $\frac{v_1^0}{v_1^0 + v_2^0} \sim \text{Beta}(\alpha_k^0, \beta_k^0)$.

Then, in the backward pass we estimate the partial derivatives w.r.t. $\varphi = \{\alpha_k^0, \beta_k^0\}$ as can be taken by using Equation (25) and previous equations. Implicit reparameterisation of the Beta distribution is readily implemented in `TensorFlow Distributions` [Dillon et al., 2017].

A.3   Bernoulli

The Bernoulli distribution can be reparameterised using a continuous relaxation to the discrete distribution and so Equation (22) can be used.

Consider a discrete distribution $(\alpha_1, \cdots \alpha_K)$ where $\alpha_j \in \{0, \infty\}$ and $D \sim \text{Discrete}(\alpha) \in \{0, 1\}$, then $P(D_j = 1) = \frac{\alpha_j}{\sum_k \alpha_k}$. Sampling from this distribution requires performing an `argmax` operation, the crux of the problem is that the `argmax` operation doesn't have a well defined derivative.

To address the derivative issue above, the Concrete distribution [Maddison et al., 2017] or Gumbel-Softmax distribution [Jang et al., 2017] is used as an approximation to the Bernoulli distribution. The idea is that instead of returning a state on the vertex of the probability simplex like `argmax` does, these relaxations return states inside the inside the probability simplex (see Figure 2 in [Maddison et al., 2017]). The Concrete formulation and notation from [Maddison et al., 2017] are used. We sample from the probability simplex using

$$X_j = \frac{\exp((\log \alpha_j + G_j)/\lambda)}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\lambda)}, \tag{27}$$

with temperature hyperparameter $\lambda \in (0, \infty)$, parameters $\alpha_j \in (0, \infty)$ and i.i.d. Gumbel noise $G_j \sim \text{Gumbel}(0, 1)$. This equation resembles a `softmax` with a Gumbel perturbation. As $\lambda \to 0$ the `softmax` computation approaches the `argmax` computation. This can be used as a relaxation of the variational Bernoulli distribution and can be used to reparameterise Bernoulli random variables to allow gradient based learning of the variational Beta parameters downstream in our model.

When performing variational inference using the Concrete reparameterisation for the posterior, a Concrete reparameterisation of the Bernoulli prior is required to properly lower bound the ELBO in Equation (21). If $q(z_{ijk}; \pi_{jk}|v_k^0)$ is the variational Bernoulli posterior over sparse binary masks z_{ijk} for weights $\boldsymbol{w}_{jk}$ and $p(z_{ijk}; \pi_{jk}|v_k^0)$ is the Bernoulli prior. To guarantee a lower bound on the ELBO, both Bernoulli distributions require replacing with Concrete densities, i.e.,

$$\text{KL}\left[q(z_{ijk}; \pi_{jk}|v_k^0)||p(z_{ijk}; \pi_{jk}|v_k^0)\right] \geq \text{KL}\left[q(z_{ijk}; \pi_{jk}, \lambda_1|v_k^0)||p(z_{ijk}; \pi_{jk}, \lambda_2|v_k^0)\right], \tag{28}$$

where $q(z_{ijk}; \pi_{jk}, \lambda_1|v_k^0)$ is a Concrete density for the variational posterior with parameter π_{jk}, temperature parameter λ_1 given global parameters v_k^0. The Concrete prior is $p(z_{ijk}; \pi_{jk}, \lambda_2|v_k^0)$. Equation (28) is evaluated numerically
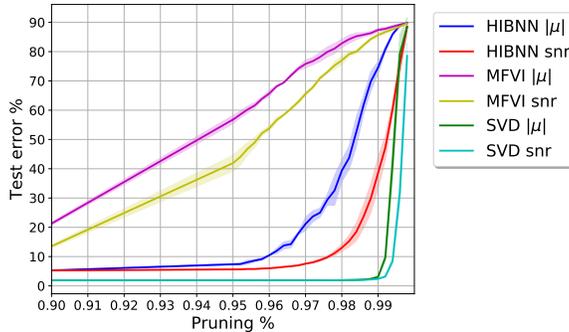
Figure 8: Test errors for different pruning cut-offs for the HIBNN and with a BNN trained with Sparse Variational Dropout (SVD) on MNIST. SVD is more sparse than the HIBNN.

by sampling from the variational posterior (we will take a single Monte Carlo sample [Kingma and Welling, 2014]). At test time one can sample from a Bernoulli using the learnt variational parameters of the Concrete distribution [Maddison et al., 2017].

In practice, the log transformation is used to alleviate underflow problems when working with Concrete probabilities. One can instead work with $\exp(Y_{ijk}) \sim \text{Concrete}(\pi_{jk}, \lambda_1 | v_k^0)$, as the KL divergence is invariant under this invertible transformation and Equation (28) is valid for optimising our Concrete parameters [Maddison et al., 2017]. For binary Concrete variables one can sample from $y_{ijk} = (\log \pi_{jk} + \log \epsilon - \log(1 - \epsilon))/\lambda_1$ where $\epsilon \sim \text{Unif}[0, 1]$ and the log-density (before applying the sigmoid activation) is $\log q(y_{ijk}; \pi_{jk}, \lambda_1 | v_{jk}) = \log \lambda_1 - \lambda_1 y_{ijk} + \log \pi_{jk} - 2 \log(1 + \exp(-\lambda_1 y_{ijk} + \log \pi_{jk}))$ [Maddison et al., 2017]. The reparameterisation $\sigma(y_{ijk}) = g(\epsilon; \pi_{jk})$, where σ is the sigmoid function, enables us to differentiate through the Concrete and use a similar formula to Equation (22).

B   COMPARISON OF HIBNN AND SPARSE VARIATIONAL DROPOUT

Using the IBP and H-IBP priors for model selection induces sparsity as a side effect. Other priors such as Sparse Variational Dropout (SVD) [Molchanov et al., 2017] or a horse-shoe prior on weights [Louizos et al., 2017] are specifically employed for sparsity. By comparing the effect of weight pruning between a HIBNN and a BNN employing SVD, we can see that the HIBNN is not as sparse as SVD, although SVD is specifically designed to be sparse, unlike our method which employs a non-parametric prior for CL, but sparsity is a side effect. The accuracies obtained from SVD before pruning are 98.1 ± 0.1 compared to 95 ± 0.0. The performance gap is due to the the difficulty in inference and with the variational approximations and reparameterisations, as noted when comparing to a BNN with no special prior in Section 5.1. In terms of pruning, SVD is slightly more robust; performance starts to degrade after 99% of weights are pruned in comparison to the HIBNN's 98%, see Figure 8.

The SVD BNN uses a two layer BNN with 200 neurons and the HIBNN with a variational truncation of $K = 200$ for a fair comparison. The H-IBP prior parameters and the initialisation of the variational posterior are $\alpha_k^0 = 4.2$ and $\beta_k^0 = 1.0$ for all k, the hyperparameter $\alpha_j = 4$ for all layers j. The Concrete temperatures used are $\lambda_1 = 0.7$ for the variational posterior and $\lambda_2 = 0.7$ for the Concrete prior. The prior on the Gaussian weights was set to $\mathcal{N}(0.0, 0.7)$ for these parameters where found to work well on split MNIST and were assumed to also work well for multiclass MNIST too.

Both networks are optimised using Adam [Kingma and Lei Ba, 2015] with a decaying learning rate schedule starting at 10^{-3} at a rate of 0.87 every 1000 steps, for 200 epochs and using a batch size of 128. Weight means are initialised with their ML parameters found after training for 100 epochs and $\log \sigma^2 = -6$. Local reparameterisation [Kingma et al., 2015] is employed. SVD is trained for 100 epochs while our method for 200 epochs, as it requires more epochs to converge.
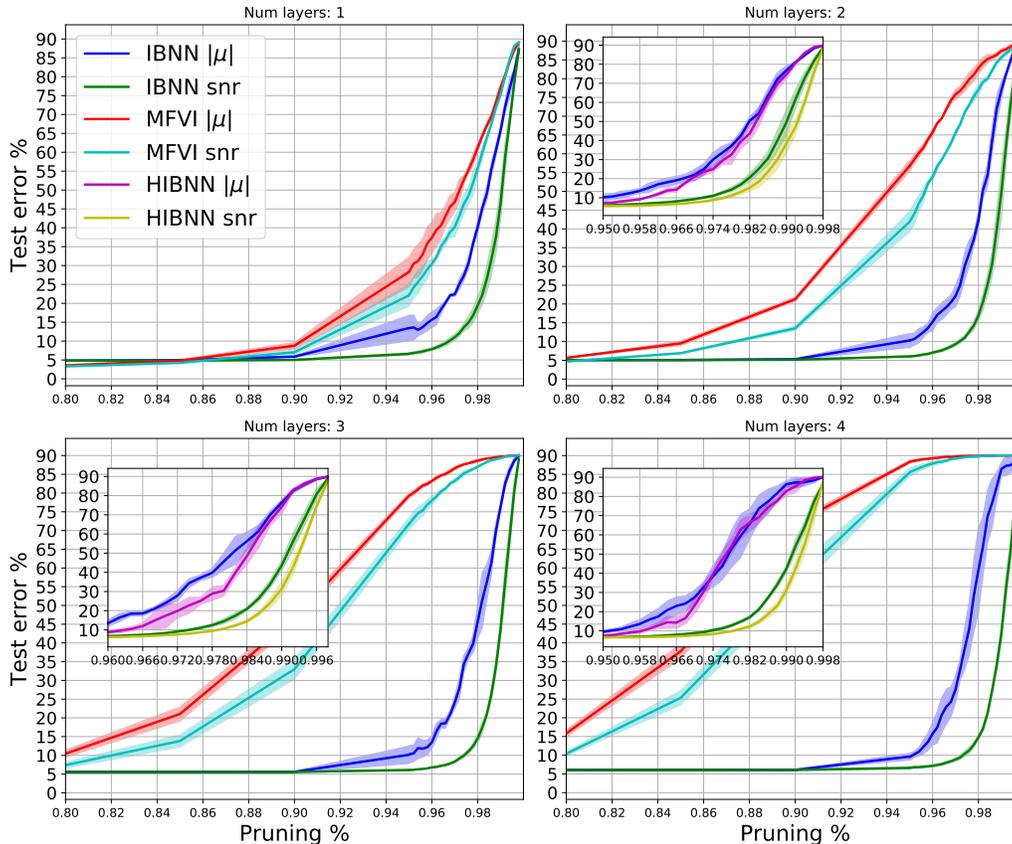
Figure 9: Weight pruning experiments for the IBNN and HIBNN on MNIST for models of different depth. As the depth increases the IBNN and HIBNN become more sparse.

C WEIGHT PRUNING FURTHER RESULTS

C.1 MNIST

To investigate the behaviour of the novel BNN introduced in this paper we perform simple and intuitive weight pruning experiments. The Gaussian weights of the BNN are zeroed out in two particular orders. The first is by the magnitude of the variational mean: $|\mu|$ the second is by the variational signal to noise ratio (snr): $|\mu|/\sigma$. For the IBNN and HIBNN, weights are pruned by according to these two metrics at the same time as having the sparse matrix Z acting on each layer. As Z is sparse by design we expect our models to be sparse in comparison to a BNN where the weights are modelled by mean-field variational inference (MFVI) [Blundell et al., 2015]. Indeed the IBNN and HIBNN are sparse in comparison to MFVI and become sparser with increased depth. In Figure 9 the weights are pruned and the accuracy on the test set is measured for networks of different sizes. MFVI is less sparse as the depth increases. Accuracy, does not vary for all models of depth 1 to 4. The MFVI models have a hidden state size of 200 and the IBNN and HIBNN use a variational truncation of $K = 200$ for fair comparison.

C.2 Fashion MNIST

We repeat this analysis for the fashion MNIST (fMNIST) [Xiao et al., 2017]. Similarly to MNIST the IBNN and HIBNN networks become sparser with increasing depth since the the drop off in accuracies arise for higher pruning percentages. There is little difference in the performance or sparsity between the IBNN and HIBNN.
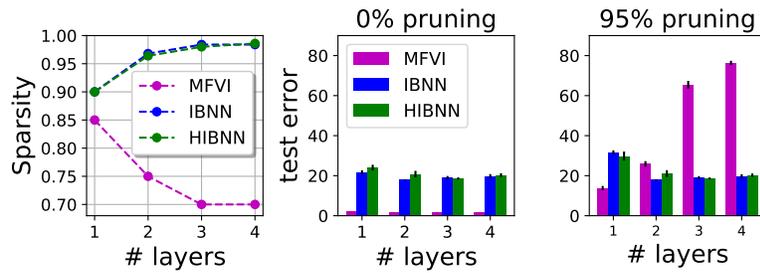
Figure 10: fMNIST network depth ablation. **L** , as the depth of the IBNN and HIBNN increases the networks tend to remain very sparse while the MFVI BNN is becomes less sparse. **M & R** , the HIBNN and IBNN remain robust to pruning with increasing depth.
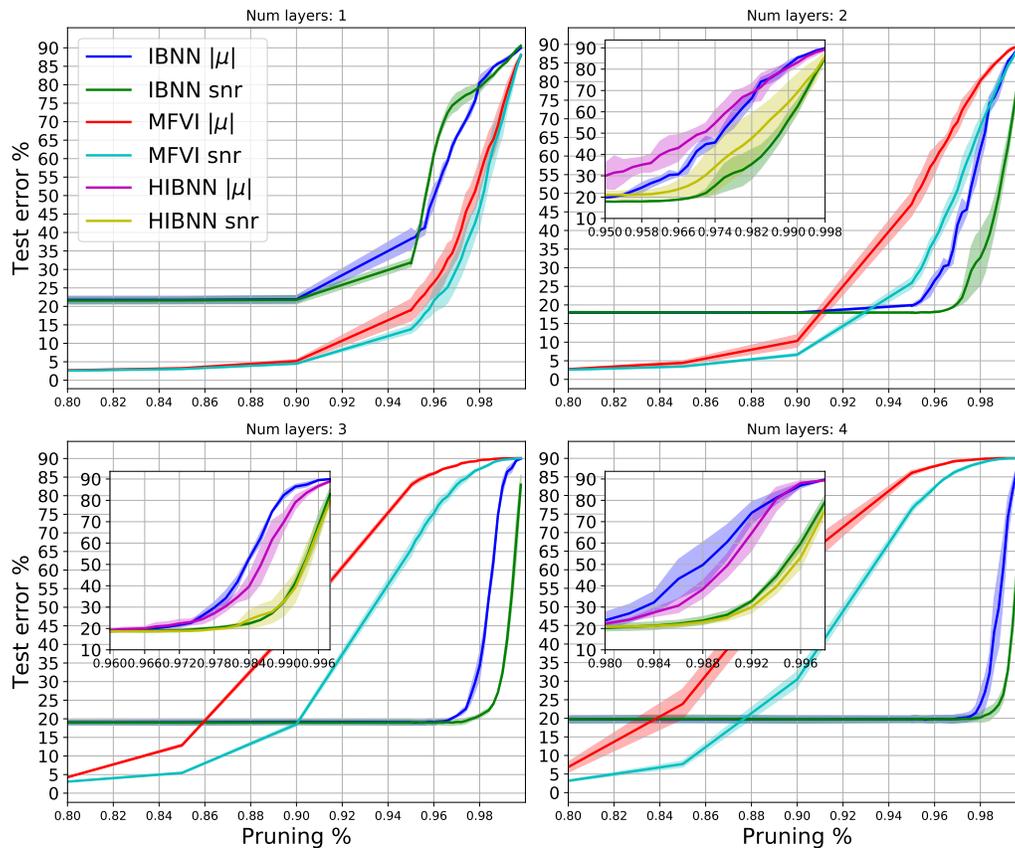

Figure 11: Weight pruning for the IBNN and HIBNN in comparison to MFVI on fashion-MNIST for models of different depth. As the depth increases the IBNN and HIBNN become more sparse.

Table 3: Average test accuracy on MNIST CL experiments over 5 runs. After removing the time-stamping feature from DEN the IBNN achieves comparable performance. The best model between the IBNN and DEN with no time-stamping is highlighted.

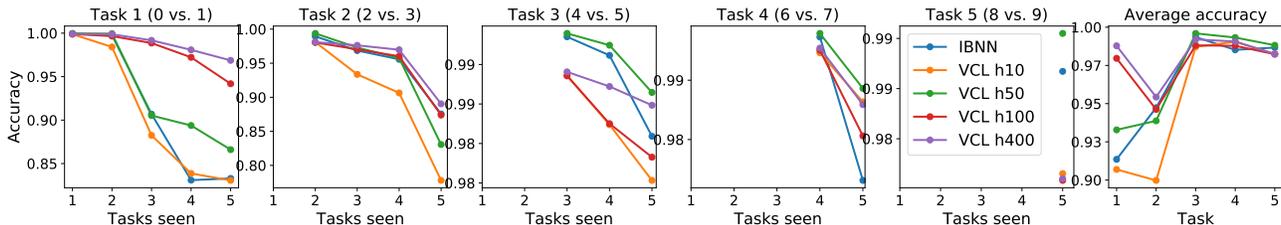|	DEN	DEN - no ts	IBNN
P CL1	$91.4_{\pm 0.5}$	$91.6_{\pm 0.5}$	$\mathbf{95.6_{\pm 0.2}}$
P CL3	$63.9_{\pm 19.2}$	$75.1_{\pm 21.0}$	$\mathbf{93.8_{\pm 0.3}}$
S CL1	$99.1_{\pm 0.1}$	$\mathbf{98.8_{\pm 0.1}}$	$95.3_{\pm 2.0}$
S CL2	$98.9_{\pm 0.1}$	$\mathbf{98.9_{\pm 0.1}}$	$91.0_{\pm 2.2}$
S CL3	$99.1_{\pm 0.1}$	$\mathbf{93.6_{\pm 10.4}}$	$85.5_{\pm 3.2}$
S$+\epsilon$ CL1	$97.2_{\pm 0.2}$	$\mathbf{95.1_{\pm 1.9}}$	$\mathbf{95.1_{\pm 1.1}}$
S$+\epsilon$ CL2	$84.8_{\pm 16.7}$	$\mathbf{91.1_{\pm 7.0}}$	$89.7_{\pm 3.8}$
S$+\epsilon$ CL3	$90.9_{\pm 12.8}$	$54.1_{\pm 24.1}$	$\mathbf{78.7_{\pm 11.7}}$
S+img CL1	$93.8_{\pm 0.8}$	$\mathbf{91.7_{\pm 2.1}}$	$91.6_{\pm 1.2}$
S+img CL2	$79.1_{\pm 13.1}$	$\mathbf{98.9_{\pm 0.1}}$	$80.5_{\pm 7.8}$
S+img CL3	$91.6_{\pm 5.1}$	$46.7_{\pm 14.3}$	$\mathbf{66.2_{\pm 13.4}}$

Figure 12: Split MNIST CL1, task accuracies versus the number of tasks the model has performed a Bayesian update. Our model is compared to VCL benchmarks with different numbers of hidden states denoted hx, $x \in \{10, 50, 100, 400\}$. For Split MNIST the tasks are simple and no overfitting is observed in VCL.

D   DYNAMICALLY EXPANDING NETWORKS AND TIME-STAMPING

DEN [Yoon et al., 2018] "time-stamps" parts of its network allowing specific parts of the network to be used by a specific tasks thereby achieving good performance and good uncertainties over tasks which have been seen for CL2 and CL3. This is at the cost of not allowing backward transfer from newly added task weights. Removing the time-stamping feature (and making DEN more inline with the IBNN) we see that it achieves similar performance for the more difficult MNIST variants, see Table 3. This motivates segregating parts of a BNN to perform a specific task so as to mimic time-stamping for future research.

E   OVERFITTING AND UNDERFITTING IN VCL

All task accuracies are an average of 5 runs and by action neurons we mean that $z_{ijk} > 0.1$ for a datapoint $\mathbf{x_i}$ in layer j and for neuron k.

For CL on split MNIST, the IBNN is able to outperform the 10 neuron VCL baseline as it underfits. On the other hand, the larger VCL networks slightly outperform the IBNN, see Figure 12. The IBP prior enables the BNN to expand from a median of 11 neurons for the first task to 14 neurons for later tasks, see Figure 14.

Regarding CL on MNIST with random background noise, it is clear that the IBNN is able to outperform VCL for all widths considered. The baseline models overfit on the second task and propagate a poor approximate posterior which affects subsequent task performance, see Figure 13. The IBNN expands over the course of the 5 tasks from a median of 11 neurons to 14 neurons in Figure 14.

VCL networks with a small width of 10 neurons on Permuted MNIST display underfitting due to a lack of capacity which translates into forgetting in CL. On the over hand, a VCL model which is overparameterised with a width
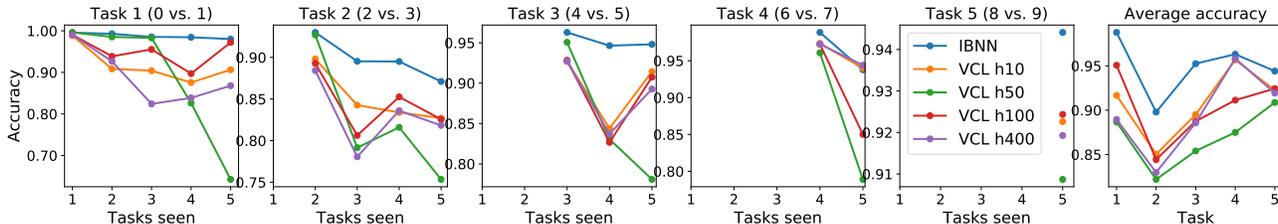
Figure 13: Split MNIST with random background noise for `CL1`, task accuracies versus the number of tasks the model has performed a Bayesian update. Our model is compared to VCL benchmarks with different numbers of hidden states denoted hx, $x \in \{10, 50, 100, 400\}$. Our model is able to counter overfitting issues and outperform VCL.
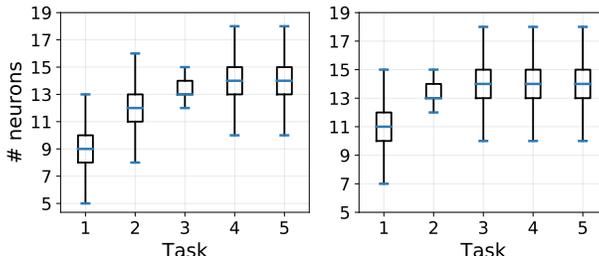

Figure 14: Box plots of the number of active neurons selected by the IBP variational posterior for each task, for **L** Split MNIST and **R** Split MNIST background noise variant for `CL1`. For both datasets our model expands its capacity over the course of CL.

of 100 displays overfitting and will subsequently propagation of a poor posterior and results in forgetting for all future tasks, Figure 15. The same phenomena of forgetting due to underfitting due to restricted model size and overfitting due to overparameterisation is observed for the Split MNIST with random background images dataset in Figure 16.

F EXPERIMENT DETAILS

We summarise all dataset sizes in Table 4.

MNIST No preprocessing of the data is performed like [Nguyen et al., 2018]. For the permuted MNIST experiments, the BNNs used for the baselines and the IBNN consist of a single layer with ReLU activations. For the IBNN, the variational truncation parameter is set to $K = 100$. For the first task the parameters of the Beta prior and the variational Beta distribution are initialised to $\alpha_k = 5$ and $\beta_k = 1$ for all k. The temperature parameters of the Concrete distributions for the variational posterior and priors are set to $\lambda_1 = 1$ and $\lambda_2 = 1$ respectively. For each batch, 10 samples are averaged from the IBP priors as this made training stable. The
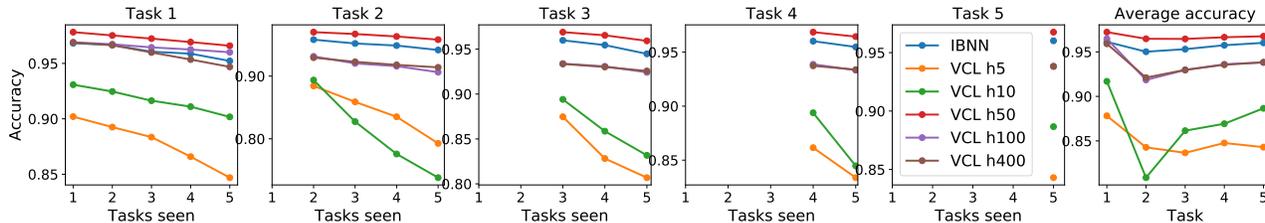

Figure 15: Permuted MNIST task accuracies versus the number of tasks the model has seen and performed a Bayesian update. Our model is compared to VCL with different numbers of hidden states sizes. Our model has a very good performance versus most models for Permuted MNIST.
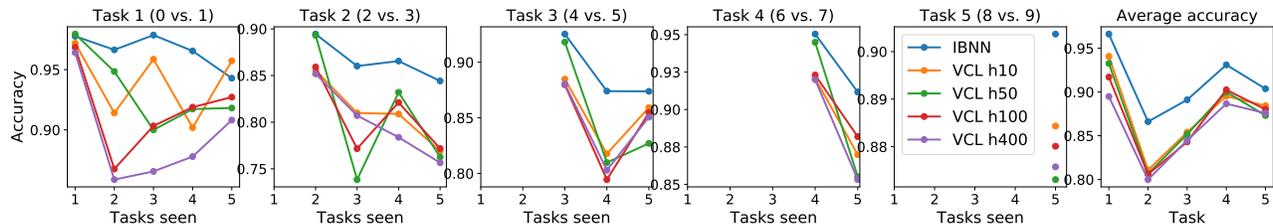
Figure 16: Split MNIST with background images, task accuracies versus the number of tasks the model has seen and performed a Bayesian update. Our model is compared to VCL with different hidden state sizes. Our model adapts its complexity overcomes forgetting better than VCL.

Gaussian weights of the BNNs have their means and variances initialised with the maximum likelihood estimators found after training for 100 epochs with Adam and the variances initialised to $\log \sigma^2 = -6$ for the first task. For the CL Adam is also used 200 epochs and an initial learning rate of 0.001 which decays exponentially with a rate of 0.87 every 1000 iterations for each task. The CL experiments are followed the implementation of VCL [Nguyen et al., 2018].

IBNN The BNNs used for the baselines and the IBNN consist of a single layer with ReLU activations. The variational truncation parameter is set to $K = 100$. For CL1 and for the first task, the parameters of the Beta prior and the variational Beta distribution are initialised to $\alpha_k = 5$ and $\beta_k = 1$ for all k. The temperature parameters of the Concrete distributions for the variational posterior and priors are set to $\lambda_1 = 0.7$ and $\lambda_2 = 0.7$, respectively. The prior for the Gaussian weights is $\mathcal{N}(0, 0.7)$. The Gaussian weights of the BNNs have their means initialised with the maximum likelihood estimators, found after training a NN for 100 epochs with Adam. The variances initialised to $\log \sigma^2 = -6$. The Adam optimiser is used to for 600 epochs, the default in the original VCL paper [Nguyen et al., 2018] for Split MNIST. To ensure convergence of the IBNN, the first task needs to be trained for 20% more epochs. An initial learning rate of 0.001 which decays exponentially with a rate of 0.87 every 1000 iterations is employed.

F.1 HPs

Random search is performed on some of the experiments presented in the main paper by sampling over a discrete set of values or over a range for some hyperparameters listed below. We denote curly brackets $\{\ldots\}$ as a discrete set and square brackets $[\ldots]$ as a range.

HIBNN CL1 and CL2 for CL experiments with increasing task difficulty (Table 2).

- Concrete posterior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$.

- Concrete prior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$.

- Child IBP base α: $\{1, 2, 3, 4, 5\}$.

- Child IBP multiple of α after each new dataset seen in during CL: $\{1, 2, 3\}$

- Global IBP prior, α^0: $[5, 25]$

IBNN CL1 and CL2 for CL experiments with increasing task difficulty (Table 2) and IBNN CL3 for Split MNIST with background noise and CL2 and CL3 for Split MNIST with background images (Table 1).

- Concrete prior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$

- Concrete posterior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$
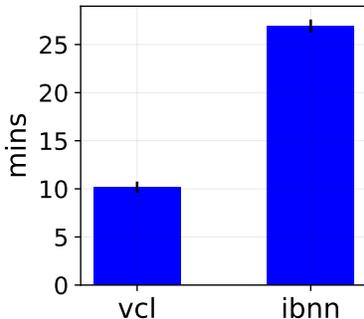
- IBP prior, α: $[5, 25]$

Figure 17: Time to run one permuted MNIST task for our model and VCL. Our model takes longer due to inference of the IBP variational posterior.

Table 4: Dataset sizes used for experiments. Split MNIST's variants is obtained from `https://sites.google.com/a/lisa.iro.umontreal.ca/public_static_twiki/variations-on-the-mnist-digits`.

Dataset	Train set size	Test set size
Permuted MNIST	$50,000$	$10,000$
Split MNIST	$60,000$	$10,000$
Split MNIST + noise	$52,000$	$10,000$
Split MNIST + images	$52,000$	$10,000$
CIFAR 10	$50,000$	$10,000$

F.2 Blsn

The implementations for EWC, SI and GEM is from [Hsu et al., 2018] and uses default hyperparameters. For DEN we use the implementation provided by the authors with default hyperparameters [Yoon et al., 2018]. For VCL we likewise use the implementation provided by the authors with default hyperparameters [Nguyen et al., 2018].

F.3 in

Comparing the training time of one Permuted MNIST task for 200 epochs can be seen in Figure 17. The IBNN takes longer to train. Stochastic variational inference of the IBP posterior involves taking multiple samples from it in forward pass. Then in the backward pass, gradients of samples from the Concrete distribution and implicit gradients from samples from the Beta distribution need to be calculated; this explains the longer training times in comparison to VCL. Improving the inference scheme is a good direction for further work.

G   DETAILED COMPARISON WITH RELATED WORKS

The related papers [Panousis et al., 2019] and [Kumar et al., 2019] apply the IBP prior to a BNN like in our model, however they apply it in a very different fashion. In these works, the IBP prior is applied to each layer such that $Z \in \mathbb{Z}_2^{d \times k}$, where d is input dimension or input hidden layer size and k is the output dimension. Crucially, Z is not sampled for each data point. Our model applies $Z \in \mathbb{Z}_2^{n \times k}$, such that each point selects neurons according to the IBP, this remains closer to the original formulation of the IBP prior for matrix factorisation, Section 2.2 and to related works applying the IBP prior to VAEs.

Despite this important difference with Bayesian Structure Adaptation for CL (BSCL) [Kumar et al., 2019], the results for BSCL are strong and suggest some interesting design choices and ideas that can be leveraged to use the IBP prior for CL with BNNs. It is promising to see similar ideas being put forward in this area. There are some design choices which are quite different from ours in three main regards.

1. An additional variational parameter for each Gaussian neural network weight is introduced. The IBP prior model is $z_k \sim \text{Ber}(\pi_k)$ where $\pi_k = \prod_{i=1}^k v_i$ and where $v_i \sim \text{Beta}(\alpha, \beta)$ for each layer, for a neuron k. However in the BSCL implementation, the Bernoulli probability is $\tilde{\pi}_k = p_k + \prod_{i=1}^k v_i$, where p_k is an additional learnable parameter. Thus the Bernoulli parameters are being directly learnt, similarly to [Dikov et al., 2019]. The

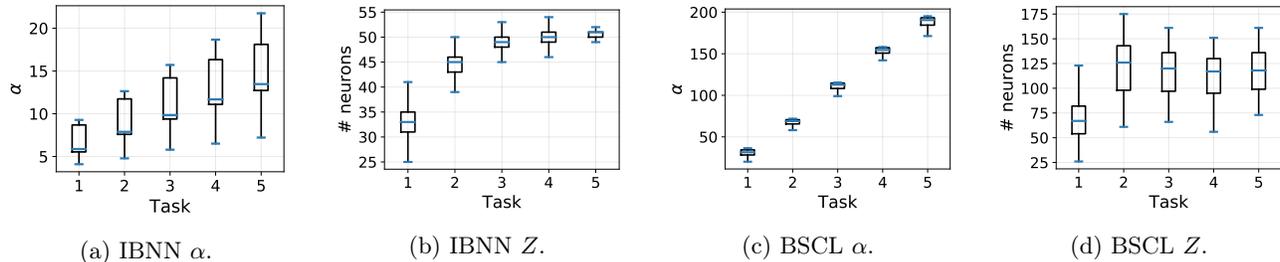(a) IBNN α.   (b) IBNN Z.   (c) BSCL α.   (d) BSCL Z.

Figure 18: Comparing our model to BSCL [Kumar et al., 2019] for a simple 5 task CL experiment on Permuted MNIST. In our model the variational parameters α which control the number of neurons, increases for each task, Figure 18a, as does the resulting number of neurons, Figure 18b. While for BSCL as α increases, Figure 18c, this doesn't translate into additional weights becoming activated, Figure 18d. Since the additional parameter p_k for all k is influencing the number of neurons which are active in addition to the IBP prior term item 1. We use the outputs of the Concrete distribution as Z and define a neuron as active if $z_k > 0.1$ for neuron k.

Table 5: Average test accuracy on MNIST variants for *task incremental learning* (CL1) experiments over 5 runs. Despite the many design choices BSCL [Kumar et al., 2019], performs similarly to our method which is simpler and more correct.

	BSCL	IBNN
S+ϵ CL1	**95.2**±1.5	**95.1**±1.1
S+img CL1	**92.9**±1.0	91.6±1.2

parameter α directly controls the number of active items in the IBP prior [Griffiths and Ghahramani, 2011] and so this additional parameter p_k can potentially dominate α in controlling the expansion of Z see Figure 18.

2. The implementation of BSCL does not strictly follow sequential Bayes, in particular the Beta distribution (which is part of the IBP and controls expansion). In sequential Bayes, like VCL, the variational posterior of the Beta distribution (the implementation uses a relaxation, the Kumaraswamy distribution), is used as the prior for the forthcoming task. The previous task's variational Beta posterior parameters are $\alpha \in \mathbb{R}_+^K$ and $\beta \in \mathbb{R}_+^K$ respectively, the new task's Beta prior parameters are then set to $\max(\alpha) \in \mathbb{R}_+$ and 1 respectively where K is the variational truncation[7]. This explains the large increase in the values of α seen in Figure 18 and so allows a lot of expansion in the model.

3. BSCL requires more memory in storing Z's from each task. The Z's are used as a mask to then finetune the network which can result in better performance. For each task, Z's for all weights are stored and recalled thus alleviating forgetting, rather than relying on sequential variational Bayes like VCL and our method. We apply sequential Bayes on the IBP and so generate Z's for each new task without having to store any parameters from previous tasks.

Despite these design choices, BSCL performs similarly to our model for the more difficult MNIST variants, see Table 5. We use the implementation provided by the authors with default parameters used for Split MNIST CL experiments. We also perform weight pruning on MNIST using a 2 layer BSCL network with variational truncation $K = 200$ using the default parameters for Split MNIST CL experiments. We see from Figure 19 that despite having a strong 0% pruning accuracy of 98 ± 0.1 in comparison to the IBNN's 95 ± 0.0. As we prune weights by the signal to noise ratio (snr), $|\mu|/\sigma$, this is less robust than pruning by $|\mu|$ which shows that BSCL has not learnt proper variational variances.

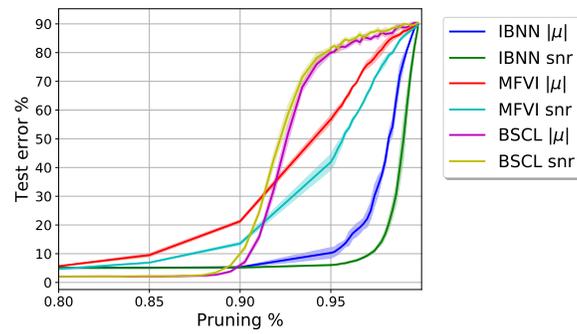[7]https://github.com/scakc/NPBCL/blob/master/ibpbnn.py#L939

Figure 19: Weight pruning on MNIST for BSCL compared to our model and a BNN with independent Gaussian weights (MFVI). BSCL is less robust to pruning than our model and MFVI, also it is less robust when pruning with signal to noise ratio: $|\mu|/\sigma$, thus the variational variances are not being learnt properly.