Hierarchical Indian Buffet Neural Networks for Bayesian Continual Learning Supplementary material

Samuel Kessler[1] **Vu Nguyen**[2] **Stefan Zohren**[1] **Stephen J. Roberts**[1]

[1]University of Oxford
[2]Amazon Research Adelaide

1 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:

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

$$\pi_k^0 = \prod_{i=1}^{k} v_i, \qquad \forall\, k, \tag{2}$$

$$\pi_{jk} \sim \text{Beta}(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)), \qquad \forall\, k, j = 1, \ldots, J, \tag{3}$$

$$z_{ijk} \sim \text{Bern}(\pi_{jk}), \qquad \forall\, j, k, i = 1, \ldots, N, \tag{4}$$

$$\boldsymbol{w}_{jk} \sim \mathcal{N}(\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2\, \mathbb{1}), \qquad \forall\, j, k, \tag{5}$$

$$h_{jk} = f(\boldsymbol{h}_{j-1}\boldsymbol{w}_{jk}) \circ z_{ijk} \qquad \forall\, i, j, k. \tag{6}$$

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

$$q(\phi) = \prod_{k=1}^{K} q(v_k^0; \alpha_k^0, \beta_k^0) \prod_{j=1}^{J} q(\pi_{jk}|v_k^0) q(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}) \prod_{i=1}^{N} q(z_{ijk}|v_k^0), \tag{7}$$

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

Supplement for the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021).

in Equation (7) is specified as follows

$$q(v_k^0; \alpha_k^0, \beta_k^0) = \text{Beta}(v_k; \alpha_k^0, \beta_k^0), \tag{8}$$

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

$$q(\pi_{jk}|v_k^0) = \text{Beta}(\pi_{jk}; \alpha_j \pi_k^0, \alpha_j(1-\pi_k^0)) \tag{10}$$

$$q(z_{ijk}|v_k^0) = \text{Bern}(z_{ijk}; \pi_{jk}), \tag{11}$$

$$q(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}) = \mathcal{N}(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \, \mathbb{1}), \tag{12}$$

where α_j are hyperparameters [Gupta et al., 2012]. Now that we have defined our structured variational approximation in Equation (7) we can write down the objective

$$\arg\min D_{\text{KL}}\left(q(\boldsymbol{\phi})||p(\boldsymbol{\phi}|\mathcal{D})\right) = \arg\min D_{\text{KL}}\left(q(\boldsymbol{\phi})||p(\boldsymbol{\phi})\right) - \mathbb{E}_{q(\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{\phi})]. \tag{13}$$

In the above formula, $q(\boldsymbol{\phi})$ is the approximate posterior and $p(\boldsymbol{\phi})$ is the prior. By substituting Equation (7), we obtain the negative ELBO objective:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}, \mathcal{D}) = \text{KL}(q(\boldsymbol{v^0})||p(\boldsymbol{v^0})) + \sum_{j=1}^{J} \text{KL}(q(\boldsymbol{\pi}_j|\boldsymbol{v}^0)||p(\boldsymbol{\pi}_j|\boldsymbol{v}^0)) + \text{KL}(q(\boldsymbol{w_j})||p(\boldsymbol{w_j})) \\ - \sum_{j=1}^{J}\sum_{i=1}^{|\mathcal{D}|} \mathbb{E}_{q(\boldsymbol{\phi})}[\log p(y_i|\boldsymbol{x_i}, \boldsymbol{z_{ij}}, \boldsymbol{w_j})] + \sum_{j=1}^{J}\sum_{i=1}^{|\mathcal{D}|} \text{KL}(q(\boldsymbol{z_{ij}}|\boldsymbol{\pi_j})||p(\boldsymbol{z_{ij}}|\boldsymbol{\pi_j})).\end{aligned} \tag{14}$$

Estimating the gradient of the Bernoulli and Beta variational parameters requires a suitable reparameterisation. Samples from the Bernoulli distribution in Equation (11) 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 $\boldsymbol{\varphi}$ of the approximate H-IBP posterior. The mean-field approximation for the Gaussian weights of the BNN is used, $\boldsymbol{w}$ in Equation (12) [Blundell et al., 2015, Nguyen et al., 2018]. In the next sections we detail the reparameterisations used to optimise Equation (14).

1.1 THE VARIATIONAL GAUSSIAN WEIGHT DISTRIBUTION REPARAMETERISATION

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:

$$\nabla_\varphi \mathbb{E}_{q(\boldsymbol{w}_{jk};\varphi)}\left[f(\boldsymbol{w}_{jk})\right] = \mathbb{E}_{q(\epsilon)}\left[\nabla_\varphi f(\boldsymbol{w}_{jk})\right]|_{\boldsymbol{w}_{jk}=g(\epsilon;\varphi)}, \tag{15}$$

where f is an objective function, for instance Equation (14). 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 (14) 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 (15) (the backward pass). In the forward pass, $\epsilon \sim q(\epsilon)$ is sampled to compute $\boldsymbol{w}_{jk} = \boldsymbol{\mu}_{jk} + \boldsymbol{\sigma}_{jk}\epsilon$.

1.2 THE IMPLICIT BETA DISTRIBUTION REPARAMETERISATION

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:

$$\begin{aligned}\nabla_\varphi \mathbb{E}_{q(v^0;\varphi)}\left[f(v^0)\right] &= \mathbb{E}_{q(\epsilon)}\left[\nabla_\varphi f(v^0)\right]|_{v^0=g(\epsilon;\varphi)} \\ &= \mathbb{E}_{q(v^0;\varphi)}[\nabla_{v^0} f(v^0)\nabla_\varphi v^0],\end{aligned} \tag{16}$$

where f is an objective function, i.e. Equation (14), $\varphi = \{\alpha_k^0, \beta_k^0\}$ are the Beta parameters. The implicit reparameterisation gradient solves for $\nabla_\varphi v^0$, above, using implicit differentiation [Figurnov et al., 2018]:

$$\nabla_\varphi v^0 = -(\nabla_z g_\varphi^{-1}(v^0;\varphi))^{-1}\nabla_\varphi g_\varphi^{-1}(v^0;\varphi), \tag{17}$$

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 (17) 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 (17) the implicit gradient is:

$$\begin{aligned}\nabla_\varphi v^0 &= \nabla_\varphi F^{-1}(v^0;\varphi) \\ &= \frac{\nabla_\varphi F(v^0;\varphi)}{p_\varphi(v^0;\varphi)},\end{aligned} \tag{18}$$

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

$$F(v^0;\varphi) = \frac{B(v^0;\varphi)}{B(\varphi)} \tag{19}$$

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 (18) and previous equations. Implicit reparameterisation of the Beta distribution is readily implemented in `TensorFlow Distributions` [Dillon et al., 2017].

1.3 THE VARIATIONAL BERNOULLI DISTRIBUTION REPARAMETERISATION

The Bernoulli distribution can be reparameterised using a continuous relaxation to the discrete distribution and so Equation (15) 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

$$X_j = \frac{\exp((\log \alpha_j + G_j)/\lambda)}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\lambda)}, \tag{20}$$

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 (14). 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.,

$$\text{KL}\left[q(z_{ijk}; \pi_{jk}|v_k^0)||p(z_{ijk}; \pi_{jk}|v_k^0)\right] \geq \text{KL}\left[q(z_{ijk}; \pi_{jk}, \lambda_1|v_k^0)||p(z_{ijk}; \pi_{jk}, \lambda_2|v_k^0)\right], \tag{21}$$

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 (21) is evaluated numerically 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 (21) 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 (15).

2 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 1.

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.

3 WEIGHT PRUNING FURTHER RESULTS

3.1 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

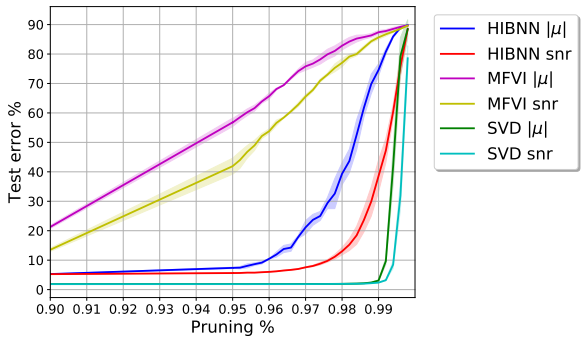


Figure 1: Test errors for different pruning cut-offs for the HIBNN and with a BNN trained with Sparse Variational Dropout (SVD) on MNIST. SVD is more sparse than the HIBNN.

Table 1: 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±0.5	91.6±0.5	**95.6±0.2**
P CL3	63.9±19.2	75.1±21.0	**93.8±0.3**
S CL1	99.1±0.1	**98.8±0.1**	95.3±2.0
S CL2	98.9±0.1	**98.9±0.1**	91.0±2.2
S CL3	99.1±0.1	**93.6±10.4**	**85.5±3.2**
S+ϵ CL1	97.2±0.2	**95.1±1.9**	**95.1±1.1**
S+ϵ CL2	84.8±16.7	**91.1±7.0**	**89.7±3.8**
S+ϵ CL3	90.9±12.8	54.1±24.1	**78.7±11.7**
S+img CL1	93.8±0.8	**91.7±2.1**	**91.6±1.2**
S+img CL2	79.1±13.1	**98.9±0.1**	80.5±7.8
S+img CL3	91.6±5.1	46.7±14.3	**66.2±13.4**

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 2 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.

3.2 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.

4 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 1. This motivates segregating parts of a BNN to perform a specific task so as to mimic time-stamping for future research.

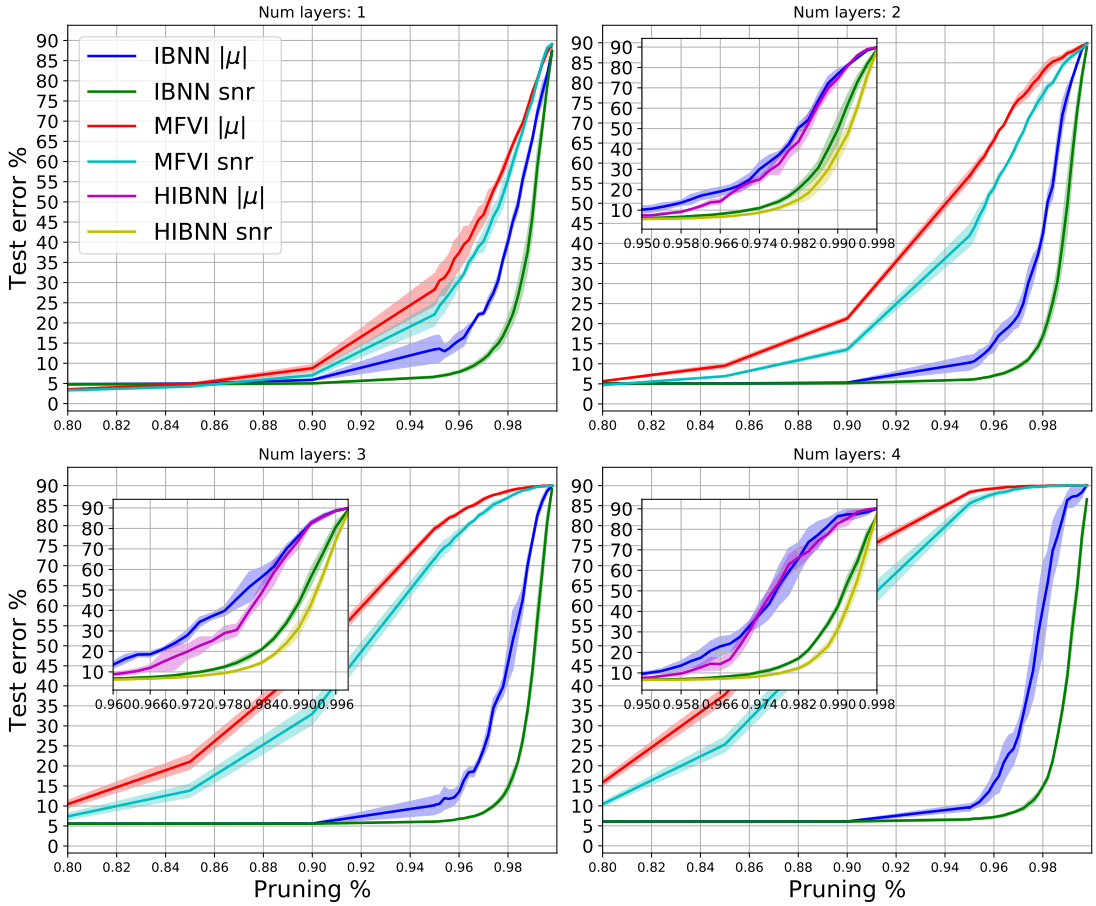


Figure 2: Weight pruning experiments for the IBNN and HIBNN on MNIST for models of different depth. As the depth increases the IBNN and HIBNN become more sparse.

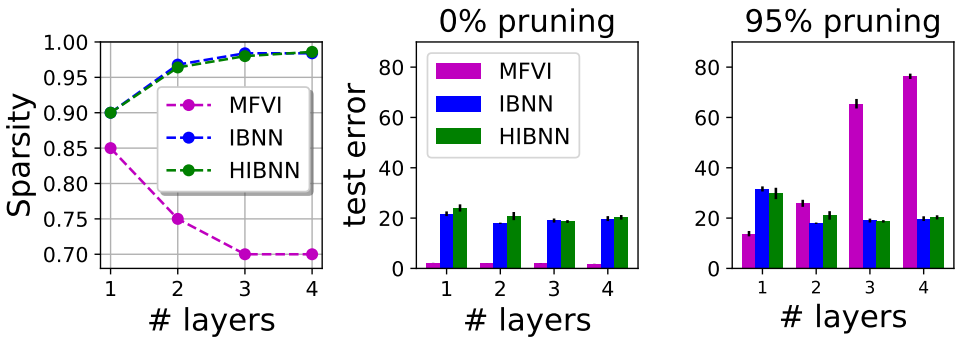


Figure 3: fMNIST network depth ablation. **Left**, as the depth of the IBNN and HIBNN increases the networks tend to remain very sparse while the MFVI BNN is becomes less sparse. **Middle & Right**, the HIBNN and IBNN remain robust to pruning with increasing depth.

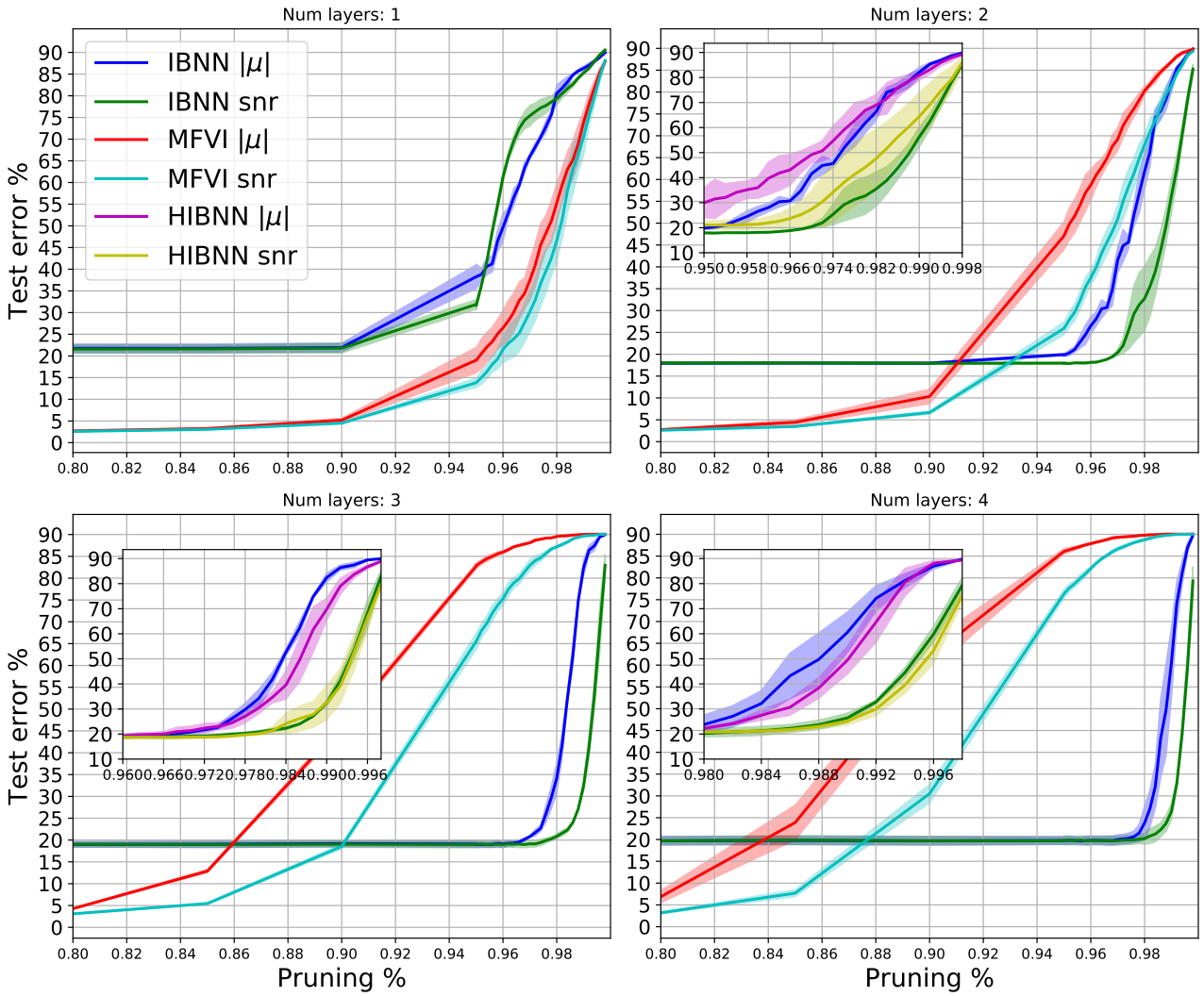


Figure 4: Weight pruning for the IBNN and HIBNN in comparison to MFVI on fashion-MNIST for models of different depth. As the depth increases the IBNN and HIBNN become more sparse.

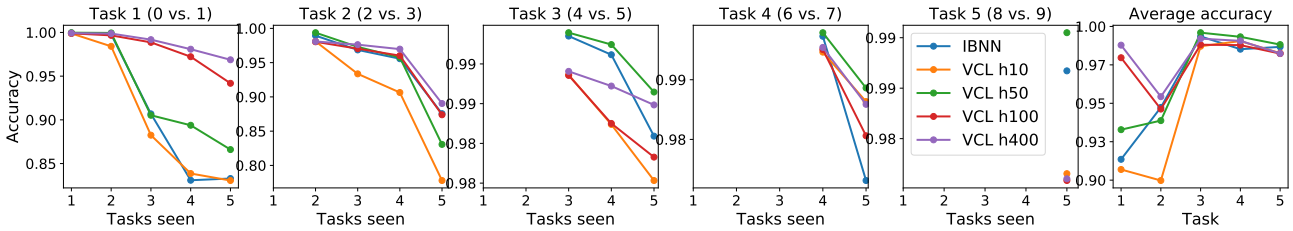


Figure 5: Split MNIST `CL1`, task accuracies versus the number of tasks the model has performed a Bayesian update. Our model is compared to VCL benchmarks with different numbers of hidden states denoted hx, $x \in \{10, 50, 100, 400\}$. For Split MNIST the tasks are simple and no overfitting is observed in VCL.

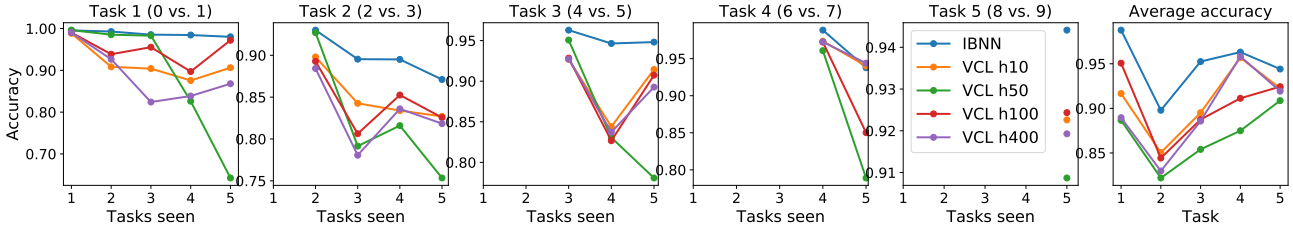


Figure 6: Split MNIST with random background noise for `CL1`, task accuracies versus the number of tasks the model has performed a Bayesian update. Our model is compared to VCL benchmarks with different numbers of hidden states denoted hx, $x \in \{10, 50, 100, 400\}$. Our model is able to counter overfitting issues and outperform VCL.

5 OVERFITTING AND UNDERFITTING IN VCL

Preliminaries. 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 5. 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 7.

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 6. The IBNN expands over the course of the 5 tasks from a median of 11 neurons to 14 neurons in Figure 7.

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 of 100 displays overfitting and will subsequently propagation of a poor posterior and results in forgetting for all future tasks, Figure 8. The

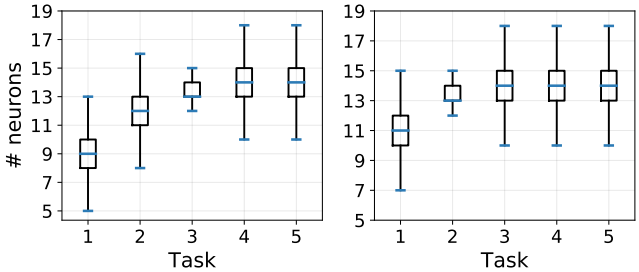


Figure 7: Box plots of the number of active neurons selected by the IBP variational posterior for each task, for **Left** Split MNIST and **Right** Split MNIST background noise variant for `CL1`. For both datasets our model expands its capacity over the course of CL.

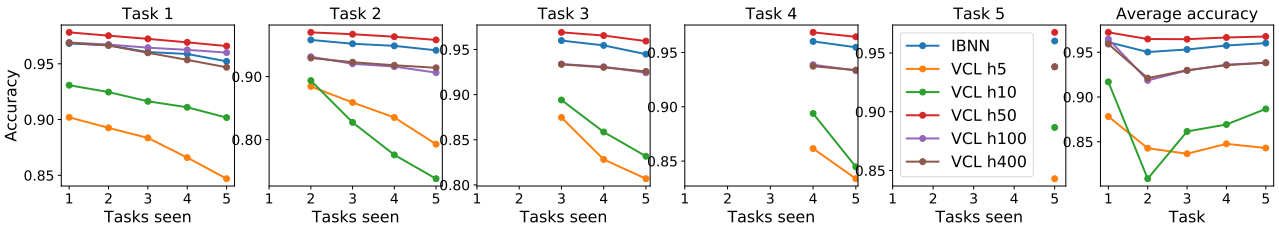


Figure 8: Permuted MNIST task accuracies versus the number of tasks the model has seen and performed a Bayesian update. Our model is compared to VCL with different numbers of hidden states sizes. Our model has a very good performance versus most models for Permuted MNIST.

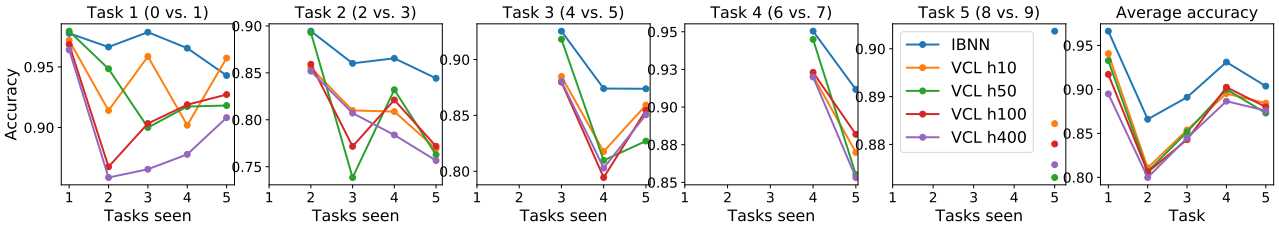


Figure 9: Split MNIST with background images, task accuracies versus the number of tasks the model has seen and performed a Bayesian update. Our model is compared to VCL with different hidden state sizes. Our model adapts its complexity overcomes forgetting better than VCL.

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 9.

6 EXPERIMENTAL DETAILS

We summarise all dataset sizes in Table 2.

Permuted 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 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].

Split MNIST and variants. 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.

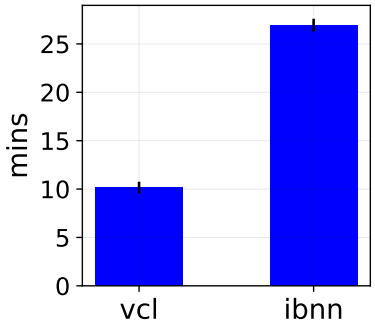


Figure 10: Time to run one permuted MNIST task for our model and VCL. Our model takes longer due to inference of the IBP variational posterior.

6.1 HYPERPARAMETER OPTIMISATION DETAILS

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: $[5, 25]$

6.2 BASELINES IMPLEMENTATIONS

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].

6.3 TRAINING TIME

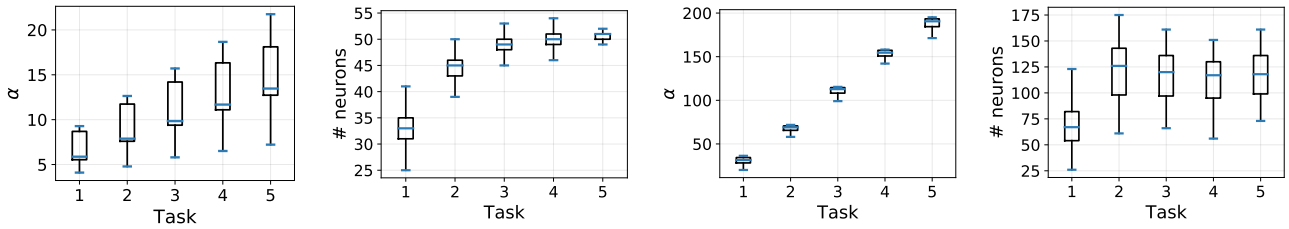
Comparing the training time of one Permuted MNIST task for 200 epochs can be seen in Figure 10. The IBNN takes longer to train. Stochastic variational inference of the IBP posterior involves taking multiple samples from it in forward pass. Then in the backward pass, gradients of samples from the Concrete 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.

7 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.

Table 2: 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	$50,000$	$10,000$



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

Figure 11: 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 11a, as does the resulting number of neurons, Figure 11b. While for BSCL as α increases, Figure 11c, this doesn't translate into additional weights becoming activated, Figure 11d. 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.

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 2.3 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 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 11.
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[1]. This explains the large increase in the values of α seen in Figure 11 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 3. We

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

Table 3: 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	$\mathbf{95.2}\pm\mathbf{1.5}$	$\mathbf{95.1}\pm\mathbf{1.1}$
S+img CL1	$\mathbf{92.9}\pm\mathbf{1.0}$	91.6 ± 1.2

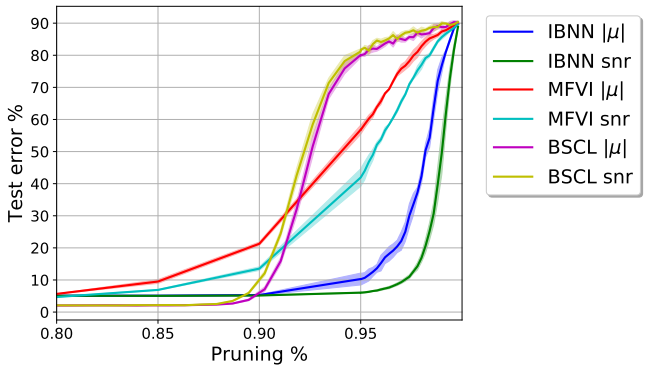


Figure 12: Weight pruning on MNIST for BSCL compared to our model and a BNN with independent Gaussian weights (MFVI). BSCL is less robust to pruning than our model and MFVI, also it is less robust when pruning with signal to noise ratio: $|\mu|/\sigma$, thus the variational variances are not being learnt properly.

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 12 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.

References

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

Georgi Dikov, Patrick van der Smagt, and Justin Bayer. Bayesian Learning of Neural Network Architectures. In *AISTATS*, 2019.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. TensorFlow Distributions. 2017.

Michael Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit Reparameterization Gradients. In *Neural Information Processing Systems*, 2018.

Thomas L Griffiths and Zoubin Ghahramani. The Indian Buffet Process: An Introduction and Review. *Journal of Machine Learning Research*, 12:1185–1224, 2011.

Sunil Kumar Gupta, Dinh Phung, and Svetha Venkatesh. 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, 2012.

Matthew D Hoffman and David M Blei. Structured Stochastic Variational Inference. In *International Conference on Artificial Intelligence and Statistics*, 2015.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. In *Continual Learning Workshop, 32nd Conference on Neural Information Processing Systems*, 2018.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparametrization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. In *International Conference on Machine Learning*, pages 2235–2244, 2018.

Diederik P Kingma and Jimmy Lei Ba. ADAM: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational Dropout and the Local Reparameterization Trick. In *Neural Information Processing Systems*, 2015.

Abhishek Kumar, Sunabha Chatterjee, and Piyush Rai. Bayesian structure adaptation for continual learning. *arXiv preprint arXiv:1912.03624*, 2019.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian Compression for Deep Learning. In *Neural Information Processing Systems*, 2017.

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

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *ArXiv*, abs/1906.10652, 2019.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational Dropout Sparsifies Deep Neural Networks. In *International Conference on Machine Learning*, 2017.

Eric Nalisnick and Padhraic Smyth. Stick-Breaking Variational Autoencoders. In *International Conference on Learning Representations*, 2017.

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

Konstantinos P. Panousis, Sotirios Chatzis, and Sergios Theodoridis. Nonparametric Bayesian deep networks with local competition. *36th International Conference on Machine Learning, ICML 2019*, 2019-June(May):8772–8783, 2019.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

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