Supplementary Material: Nonparametric Bayesian Deep Networks with Local Competition

A. Details on Model Initialization and Training

For the Gaussian posteriors of the network weights, e.g. the feedforward weights, $q(\boldsymbol{W}) \sim \prod_{j,k,u} \mathcal{N}(w_{j,k,u}|\mu_{j,k,u}, \sigma^2_{j,k,u})$, we initialize the means $\mu_{j,k,u}$ by sampling from $\mathcal{N}(0, 0.01)$ and the standard deviations $\sigma_{j,k,u}$ by sampling from $\mathcal{N}(0.0005, 0.01)$. The same procedure is applied in the convolutional variant of our approach. The initialization of the rest of the variational parameters are described in the main text, in Section 4.1. Concerning the preprocessing of the used data, the digits of the MNIST dataset are only rescaled to lie in the $[-1, 1]$ interval; for CIFAR-10, we employ the preprocessing described in Zagoruyko & Komodakis (2016). The output layers of all trained networks are Softmax classifiers.

Then, the resulting expression of the network ELBO, that is optimized via ADAM, takes the form:

$$\begin{aligned}\mathcal{L}(\phi) = \mathbb{E}_{q(\cdot)}\Big[\log p(\mathcal{D}|\boldsymbol{u}, \boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\xi}) - \Big(KL\big[\, q(\{\mathbf{Z}\}) \,||\, p(\{\mathbf{Z}|\boldsymbol{u}\})\,\big] + KL\big[\, q(\{\boldsymbol{u}\}) \,||\, p(\{\boldsymbol{u}\})\,\big] \\ + KL\big[\, q(\{\boldsymbol{\xi}|\boldsymbol{W}, \boldsymbol{Z}\}) \,||\, p(\{\boldsymbol{\xi}\})\,\big] + KL\big[\, q(\{\boldsymbol{W}\}) \,||\, p(\{\boldsymbol{W}\})\,\big] \Big)\Big]\end{aligned} \tag{1}$$

where we postulate:

$$q(\boldsymbol{u}) = \prod_k q(u_k), \;\; \text{s.t.} : q(u_k) \approx \text{Kumaraswamy}(u_k|a_k, b_k) \tag{2}$$

and ϕ are the parameters targeted by the optimization algorithm (described next).

In the ELBO expression (1), the KL divergences break into sums over the components of the entailed vectors of latent variables, as a result of our mean-field assumption. In our work, all the entailed KL divergences are computed approximately, by drawing MC samples from the corresponding posteriors. In this context, we employ appropriate reparameterization tricks to reduce the induced variances, as described in the main text.

Hence, for the stick-variables u_k we obtain:

$$\begin{aligned}KL[q(u_k)|p(u_k)] &= \mathbb{E}_{q(u_k)}[\log q(u_k) - \log p(u_k)] \\ &\approx \log q(\hat{u_k}) - \log p(\hat{u_k}), \;\; k = 1, \ldots, K\end{aligned} \tag{3}$$

where $\hat{u}_k$ is a reparameterized sample from the Kumaraswamy distribution (Eq. (10) in the original text) with parameters a_k and b_k.

Turning to the component utility indicators z, the corresponding KL divergences can be calculated as follows (the expressions below concern a simple feedforward network; the expressions pertaining to the convolutional variant can be obtained similarly):

$$\begin{aligned}KL[q(z_{j,k})|p(z_{j,k})] &= \mathbb{E}_{q(z_{j,k}),q(u_k)}[\log q(z_{j,k}) - \log p(z_{j,k})] \\ &\approx \log q(\hat{z}_{j,k}) - \log p(\hat{z}_{j,k}|\prod_{i=1}^{k} \hat{u_i}), \;\; j = 1, \ldots, J, \;\; k = 1, \ldots, K\end{aligned} \tag{4}$$

where the $\hat{z}_{j,k}$ samples are reparameterized via the Gumbel-Softmax trick with parameters $\tilde{\pi}_{j,k}$ and $1 - \tilde{\pi}_{j,k}$ (Eq. (11) in the main text).

We use a similar procedure to calculate the KL terms for the winning indicator variables ξ_{nku}:

$$\begin{aligned} KL[q(\xi_{nku})|p(\xi_{nku})] &= \mathbb{E}_{q(\xi_{nku}),q(w),q(z)}\left[\log q\left(\xi_{n,k,u}|\text{softmax}\left(\sum_{j=1}^{J} w_{j,k,u}\cdot z_{j,k}\cdot x_{n,j}\right)\right) - \log p(\xi_{n,k,u})\right] \\ &\approx \log q\left(\hat{\xi}_{n,k,u}|\text{softmax}\left(\sum_{j=1}^{J} \hat{w}_{j,k,u}\cdot \hat{z}_{j,k}\cdot x_{n,j}\right)\right) - \log p(\hat{\xi}_{nku}), \forall n,j,k,u \end{aligned} \tag{5}$$

where the samples $\hat{\xi}_{nku}$ are reparameterized via the Gumbel-Softmax trick, with parameters vector given at the right-hand side of Eq. (2) in the main text.
The Gaussian weights yield:

$$\begin{aligned} KL[(q(w_{j,k,u})|p(w_{j,k,u})] &= \mathbb{E}_{q(w_{j,k,u})}[\log q(w_{j,k,u}) - \log p(w_{j,k,u})] \\ &\approx \log q(\hat{w}_{j,k,u}) - \log p(\hat{w}_{j,k,u}),\ \forall j,k,u \end{aligned} \tag{6}$$

where $\hat{w}_{j,k,u} = \mu_{j,k,u} + \sigma_{j,k,u}\hat{\epsilon}$, and $\epsilon \sim \mathcal{N}(0,1)$.

Finally, the posterior expectation of the data log-likelihood $\log p(\mathcal{D}|\boldsymbol{u}, \boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\xi})$ is essentially the negative categorical cross-entropy between the data labels and the class probabilities generated by the penultimate Softmax layer. This posterior expectation is computed by using the reparameterized samples of the entailed latent variables that we described above.

On this basis, the optimization task targets the parameters set, ϕ, of the entailed posteriors: this comprises the a_k and b_k hyperparameters of the $\text{Kumaraswamy}(u_k|a_k, b_k)$ posteriors, the $\mu_{j,k,u}$ and $\sigma_{j,k,u}$ hyperparameters of the $\mathcal{N}(w_{j,k,u}|\mu_{j,k,u}, \sigma^2_{j,k,u})$ posteriors, and the $\tilde{\pi}_{j,k}$ hyperparameters of the $q(z_{j,k})$ posteriors.

B. A Note on Bit Precision

As mentioned in the main text (Section 4.2), the representation of floating-point data in the binary interchange formats, according to the IEEE 754-2008 convention, considers 3 different quantities: a) 1-bit sign, b) w exponent bits and c) $t = p - 1$ precision in bits (Zuras et al., 2008). The sign bit represents the sign of the number to be represented, and the exponent bits the integer part. For example, the 32-bit format uses $w = 8$; thus, the exponent can take values in the $[-127, 127]$ interval.
As noted in Louizos et al. (2017), one can design a self-defined format based on the *overflow, underflow* and *unit round-off*. By knowing the exponent and precision in bits, one can calculate the aforementioned values. In all cases, the values of the weights required $w = 3$ exponent bits. To calculate the bit precision, we exploit the Bayesian posterior uncertainty, as encoded in the posterior weights variance, on a layer-wise basis. Specifically, we calculate the mean across the posterior weights matrix of the layer, and compute the necessary unit round-off to store the weights as proposed in (Louizos et al., 2017). We obtain the final number of bits required by adding 1 sign bit, the 3 exponent bits and the precision bits.

C. Additional Experiments: VGG-like Architecture

We additionally consider a VGG-like architecture[1] comprising 13 convolutional layers, a feedforward layer and an output layer; we train it on the CIFAR-10 dataset. The architecture consists of 3×3 kernels, with the first two layers comprising 64 feature maps, followed by two layers with 128 feature maps, three layers with 256 feature maps and 6 layers with 512 feature maps. Finally, the dense layer comprises 512 units. Similar to the previous experiments, we split the architecture to LWTA blocks comprising two competing feature maps. For our approach, the considered VGG-like architecture is initially trained with LWTA activations, without training the component (LWTA block) utility posteriors, $q(z)$; after convergence, we fine-tune the full model to infer component utility and perform pruning. The obtained results are depicted in Table 1. To measure the network reduction effectiveness, we use the model size (r_W) metric (Dai et al., 2018), which corresponds to the ratio of number of non-zero weights in the resulting network compared to the original network. We observe that our approach outperforms the related alternatives on all frontiers: prediction error, the r_W metric, and weight compression.

[1] http://torch.ch/blog/2015/07/30/cifar.html

Table 1. Experiments with the VGG-like network on CIFAR-10.

Method	Error(%)	r_w	Bit precision
VIBNet (Dai et al., 2018)	8.8	5.3	23/23/23/23/23/23/23/23/23/23/23/23/23/23/23/23
BC-GNJ (Louizos et al., 2017)	8.6	6.57	10/10/10/10/8/8/8/5/5/5/5/5/6/7/11
BC-GHS (Louizos et al., 2017)	9.0	5.4	11/12/9/14/10/8/5/5/6/6/6/8/11/17/10
SB-LWTA-2	8.5	5.18	10/10/7/10/7/5/6/5/7/7/6/6/9/14/10

References

Dai, B., Zhu, C., Guo, B., and Wipf, D. Compressing neural networks using the variational information bottleneck. In *Proc. ICML*, 2018.

Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. In *Proc. NIPS*, 2017.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *Proc. BMVC*, 2016.

Zuras, D. et al. IEEE Standard for Floating-Point Arithmetic. Technical report, IEEE, August 2008.