REFERENCES

M. Alizadeh, J. Fernández-Marqués, N. D. Lane, and Y. Gal. An empirical study of binary neural networks' optimisation. *International Conference on Learning Representations (ICLR)*, 2019.

J. Ba, J. Kiros, and G. Hinton. Layer normalization. *arXiv preprint*, arXiv:1607.06450, 2016.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint*, arXiv:1308.3432, 2013.

M. Courbariaux, Y. Bengio, and J.P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Neural Information Processing Systems (NeurIPS)*, 2015.

M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint*, arXiv:1602.02830, 2016.

Y. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. *International Conference on Machine Learning (ICML)*, 2017.

L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *arXiv preprint*, arXiv:1410.8516, 2014.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *International Conference on Learning Representations (ICLR)*, 2017.

J. Gu, C. Li, B. Zhang, J. Han, X. Cao, J. Liu, and D. Doermann. Projection convolutional neural networks for 1-bit cnns via discrete back propagation. *arXiv preprint*, arXiv:1811.12755, 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *International Conference on Machine Learning (ICML)*, 2019a.

J. Ho, E. Lohn, and P. Abbeel. Compression with flows via local bits-back coding. *Neural Information Processing Systems (NeurIPS)*, 2019b.

E. Hoogeboom, J. W. T. Peters, R. van den Berg, and M. Welling. Integer discrete flows and lossless compression. *Neural Information Processing Systems (NeurIPS)*, 2019.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint*, arXiv:1502.03167, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

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

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. *Neural Information Processing Systems (NeurIPS)*, 2016.

F. H. Kingma, P. Abbeel, and J. Ho. Bit-Swap: recursive bits-back coding for lossless compression with hierarchical latent variables. *International Conference on Machine Learning (ICML)*, 2019.

P. Kingma, D. and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Neural Information Processing Systems (NeurIPS)*, 2018.

L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. Biva: A very deep hierarchy of latent variables for generative modeling. *Neural Information Processing Systems (NeurIPS)*, 2019.

M. McDonnell. Training wide residual networks for deployment using a single bit for each weight. *International Conference on Learning Representations (ICLR)*, 2018.

F. Pedersoli, G. Tzanetakis, and A. Tagliasacchi. Espresso: Efficient forward propagation for bcnns. *International Conference on Learning Representations (ICLR)*, 2018.

R. Ranganath, D. Tran, and D. M. Blei. Hierarchical variational models. *International Conference on Machine Learning (ICML)*, 2016.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision (ECCV)*, 2016.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *International Conference on Machine Learning (ICML)*, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic back-propagation and variational inference in deep latent gaussian models. *International Conference on Machine Learning (ICML)*, 2014.

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Neural Information Processing Systems (NeurIPS)*, 2016.

J. Townsend, T. Bird, and D. Barber. Practical lossless compression with latent variables using bits back coding. *International Conference on Learning Representations (ICLR)*, 2019.

J. Townsend, T. Bird, J. Kunze, and D. Barber. Hilloc: Lossless image compression with hierarchical latent variable models. *International Conference on Learning Representations (ICLR)*, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Neural Information Processing Systems (NeurIPS)*, 2017.

A SAMPLES

(a) ResNet VAE
(32-bit weights, 32-bit activations)

(b) Flow+
(32-bit weights, 32-bit activations)

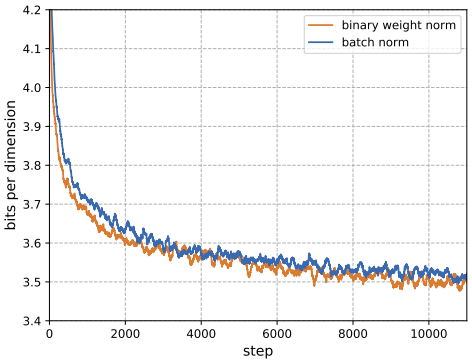
(c) ResNet VAE
(1-bit weights, 32-bit activations)

(d) Flow++
(1-bit weights, 32-bit activations)

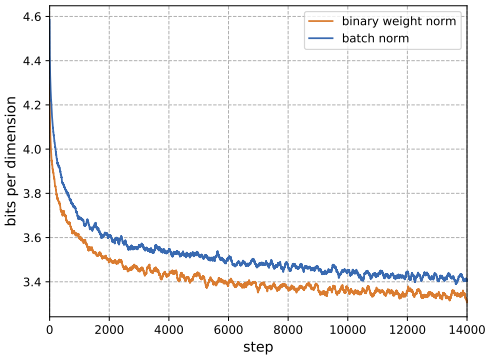
(e) ResNet VAE
(1-bit weights, 1-bit activations)

(f) Flow++
(1-bit weights, 1-bit activations)

Figure 3: Samples from the ResNet VAE (left) and Flow++ (right) models trained on CIFAR. We provide samples from the models with (a)/(b) real-valued weights and activations, (c)/(d) binary weights and real-valued activations, (e)/(f) binary weights and activations.



(a) **Flow++** (1-bit weights + 1-bit activations)



(b) **Flow++** (1-bit weights + 32-bit activations)

Figure 4: Training loss values achieved when using binary weight normalization and batch normalization for the training of binary weighted Flow++ models.

B Ablation of binary weight normalization

To examine the performance of the binary weight normalization (BWN), we perform an ablation against using the more widely used batch normalization (Ioffe & Szegedy, 2015). We simply place a batch normalization operation after every layer, instead of using BWN. Note that this still permits the use of fast binary operations, since the weights and activations are binary valued. We train the Flow++ model with binary weights and both binary and real-valued activations, comparing the two normalization schemes. The results are shown in Figure 4. We can see that BWN is slightly better for the model with binary activations, and significantly better for the model with real-valued activations. Importantly, we have found BWN to be more stable than batch normalization, which can often result in training instabilities. Indeed, to obtain the results we present when using batch normalization, training was restarted many times. BWN is also both faster to compute and simpler, not relying on retaining running averages of batch statistics.

It is also worth noting, that it is not possible to train these binary weighted generative models without any form of normalization, since training is too unstable. This is not surprising, since the binary weights themselves are large in magnitude and can result (in particular with binary activations) in very large layer outputs.

C The ResNet VAE Model

The ResNet VAE model (Kingma et al., 2016) is a hierarchical VAE. We make some, relatively small, improvements over the original model, and now give a full description of the model.

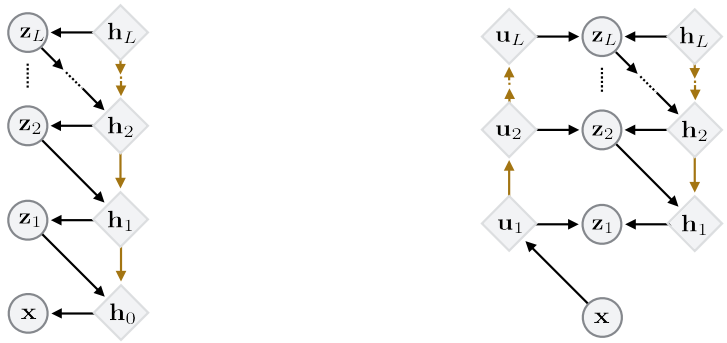
The model has a hierarchy of latent layers, $\mathbf{z}_{1:L}$. The generative model factors as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:L}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_{1:L}) p_{\boldsymbol{\theta}}(\mathbf{z}_L) \prod_{l=1}^{L-1} p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1:L}) \tag{15}$$

The inference model is factored top-down:

$$q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z}_L|\mathbf{x}) \prod_{l=1}^{L-1} q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l+1:L}, \mathbf{x}) \tag{16}$$

There is also a deterministic upwards pass (through the latent layers) performed in the inference model, which produces features used by the posterior, conditioned on just $\mathbf{x}$. We refer to the inference model as bidirectional, since there is both an upwards and downwards pass to be performed. The full graphical model is shown in Figure 5.



(a) Generative model

(b) Bi-directional inference model

Figure 5: Graphical models of the generative and inference models in a hierarchical VAE with bi-directional inference. Stochastic nodes are circular, deterministic nodes are diamond. Green lines indicate residual layers.

The objective is obtained by expanding the usual ELBO:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:L})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_{1:L})\right] - D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}_L|\mathbf{x})||\, p_{\boldsymbol{\theta}}(\mathbf{z}_L)\right) \tag{17}$$

$$- \sum_{l=1}^{L-1} D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}_l|\mathbf{z}_{l+1:L}, \mathbf{x})||\, p_{\boldsymbol{\theta}}(\mathbf{z}_l|\mathbf{z}_{l+1:L})\right) \tag{18}$$

Where D_{KL} is the KL divergence. Both the prior and posterior for a latent layer are diagonal when conditioned on deeper layers. We use logistic distributions for the latent variables, rather than the usual Gaussian distributions. We observed slightly improved performance using logistic distributions, and the parameterization is similar to a Gaussian, with a mean and scale parameter per dimension. The likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_{1:L})$ is a discretized logistic distribution.

In Figure 5 the residual connections are displayed in green, with the non-residual connections in black. The non-residual connections are convolutional layers with ELU activations functions. The residual connections are made from stacks of residual blocks. Each residual block is constructed as:

$$\text{Input} \rightarrow \text{Activation} \rightarrow \text{Conv2D}_{\text{3x3}} \rightarrow \text{Activation} \rightarrow \text{Conv2D}_{\text{3x3}} \tag{19}$$

With a skip connection adding the output to the input. The original implementation uses just one block per layer of latents. We expand this to a stack of blocks, of varying length. This block structure is depicted for the binary case in Figure 1(a)-(b). For the floating-point model we simply use floating-point weight normalized convolutions, rather than BWN convolutions, and ELU activations.

D THE FLOW++ MODEL

Here we describe fully the variational dequantization from the Flow++ model (Ho et al., 2019a), and any alterations we make to the model itself.

As described in Section 2.2, flows are invertible functions constructed from a composition of many simpler invertible functions:

$$\mathbf{f}_{\boldsymbol{\theta}} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ ... \circ \mathbf{f}_L \tag{20}$$

Each $\mathbf{f}_l$ is a coupling layer (10). Coupling layers are parameterized as a stack of convolutional residual blocks, with a convolution layer before and after the stack to project to and from the channel size of the residual stack. Each block is of the form:

$$\text{Input} \rightarrow \text{Activation} \rightarrow \text{Conv2D}_{\text{3x3}} \rightarrow \text{Activation} \rightarrow \text{Gate} \tag{21}$$

Where Gate is a 1×1 convolution followed by a gated linear unit (Dauphin et al., 2017). There is a skip connection adding the input to the output, along with layer normalization (Ba et al., 2016). This block structure is depicted in Figure 1 for the binary case.

Note that the original Flow++ implementation also utilizes an attention mechanism in the coupling layers, which adds significant complexity. We omit this from our model, since their ablations demonstrated that the improvement from the attention mechanism is marginal.

In composition the coupling layers can transform a simple density to approximate the data density. The transformed density is $p_{\boldsymbol{\theta}}(\mathbf{x})$, which we obtain by the change of variables formula (8).

Our data is generally discrete, so we actually require a discrete distribution, not a continuous density. To allow this, the Flow++ model uses variational dequantization. Suppose that the data is in $[0, 1, ..., 255]^D$. We can get a discrete distribution from a continuous density by integrating over the D-dimensional unit hypercube:

$$P_{\boldsymbol{\theta}}(\mathbf{x}) = \int_{[0,1)^D} p_{\boldsymbol{\theta}}(\mathbf{x} + \mathbf{u}) d\mathbf{u} \tag{22}$$

Variational dequantization then proceeds by forming a lower-bound to this discrete distribution by applying Jensen's inequality:

$$\log P_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x} + \mathbf{u}) - \log q_{\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x}) \right] \tag{23}$$

Where $q_{\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x})$ is now a learned component, which "dequantizes" the discrete data. This is itself parameterized as a flow, using a composition of coupling layers as above. So our model in total consists of a main flow $p_{\boldsymbol{\theta}}(\mathbf{x})$ and a dequantizing flow $q_{\boldsymbol{\phi}}(\mathbf{u}|\mathbf{x})$.

E Initialization of BWN Layers

An important aspect of weight normalized layers is the initialization. Since we are normalizing the weights, and not the output of a layer (like in batch normalization), at initialization a weight normalized layer has an unknown output scale. To remedy this, it is usual to use data-dependent initialization (Salimans & Kingma, 2016), in which some data points are used to set the the initial g and b values such that the layer output is approximately unit normal distributed.

This can be applied straightforwardly to BWN layers when training the model end-to-end, that is initializing the model at random and training til convergence. It is common, though, when training binary neural networks for classification, to use two-stage training (Alizadeh et al., 2019). This initializes the underlying weights $\mathbf{v}_{\mathbb{R}}$ of binary layers with the values from a trained model with real-valued weights.

Consider what would happen if we were to try and initialize all the components of a BWN layer with those from a trained layer with real-valued weights. The g and b can be transferred directly, and it is logical to initialize the underlying weights $\mathbf{v}_{\mathbb{R}}$ with the trained $\mathbf{v}$ values. So the magnitude of the overall weight vector $\mathbf{w}$ would remain the same in the BWN layer as in the floating-point layer, since we normalize the $\mathbf{v}_{\mathbb{B}}$ vector and apply the same g, b. This initialization seems reasonable, but fails in practice. We speculate that the reason that this fails is that, although the magnitude of the weight vector remains the same after transfer, the *direction* can be very different, since the sign function will change the direction of $\mathbf{v}_{\mathbb{R}}$[7]. Since we perform dot products with the weight vector, the output from the initialized binary layer is very different from the trained layer.

A more considered approach is to only initialize the underlying weights $\mathbf{v}_{\mathbb{R}}$ with the values from the trained network, and initialize g and b with data-dependent initialization as normal. This way, the data-dependent initialization can compensate for the change of direction that occurs in the binarization of $\mathbf{v}$. This method does train, but slower than initializing at random and training end-to-end. The only difference between training end-to-end and using this reduced form of two-stage training is the initial values, $\mathbf{v}_0$, of the real-valued weights underlying $\mathbf{v}_{\mathbb{B}}$. In the random initialization these are sampled from a Gaussian:

$$\mathbf{v}_0 \sim \mathcal{N}(0, 0.05) \tag{24}$$

We can even normalize the trained real-valued weights such that they have the same mean and variance as the Gaussian (within a weight tensor). This still results in worse performance from the two-stage training.

[7]Note that this effect is stronger in higher dimensional spaces.