Probabilistic Weight Fixing: Large-scale training of neural network weight uncertainties for quantization

Christopher Subia-Waud
School of Electronics & Computer Science
University of Southampton
Southampton, UK
cc2u18@soton.ac.uk

Srinandan Dasmahapatra
School of Electronics & Computer Science
University of Southampton
Southampton, UK
sd@ecs.soton.ac.uk

Abstract

Weight-sharing quantization has emerged as a technique to reduce energy expenditure during inference in large neural networks by constraining their weights to a limited set of values. However, existing methods often assume weights are treated solely based on value, neglecting the unique role of weight position. This paper proposes a probabilistic framework based on Bayesian neural networks (BNNs) and a variational relaxation to identify which weights can be moved to which cluster center and to what degree based on their individual position-specific learned uncertainty distributions. We introduce a new initialization setting and a regularization term, enabling the training of BNNs with complex dataset-model combinations. Leveraging the flexibility of weight values from probability distributions, we enhance noise resilience and compressibility. Our iterative clustering procedure demonstrates superior compressibility and higher accuracy compared to state-of-the-art methods on both ResNet models and the more complex transformer-based architectures. In particular, our method outperforms the state-of-the-art quantization method top-1 accuracy by 1.6% on ImageNet using DeiT-Tiny, with its 5 million+ weights now represented by only 296 unique values. Code available at `https://github.com/subiawaud/PWFN`.

1 Introduction

Weight-sharing quantization is a technique developed to lower the energy costs associated with inference in deep neural networks. By constraining the network weights to take on a limited set of values, the technique can significantly reduce the data-movement costs within hardware accelerators, which represent the primary source of energy expenditure (DRAM read costs can be as much as 200 times higher than multiplication costs [Horowitz, 2014, Jouppi et al., 2017, Sze et al., 2017]). Storing the weight values close to computation and reusing them multiple times also becomes more feasible, thanks to the limited range of possible values. Various studies have explored the effectiveness of weight-sharing quantization, including [Ullrich et al., 2017, Subia-Waud and Dasmahapatra, 2022, Wu et al., 2018b, Han et al., 2015, 2016].

Weight-sharing quantization approaches face a common challenge in determining how much a single weight can be moved to a cluster center. Traditional methods rely on the magnitude or relative movement distances between weight and cluster to decide which weights can be moved to which clusters [Han et al., 2015, Subia-Waud and Dasmahapatra, 2022], irrespective of which filter or layer in the network the weight is located. However, this assumption neglects the fact that weights with the same value may have different roles in the network, and their placement within the architecture may affect their likelihood of being moved to a particular cluster. We posit that context-dependent movement of weights to clusters can, instead, better preserve the representational capacity of the

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

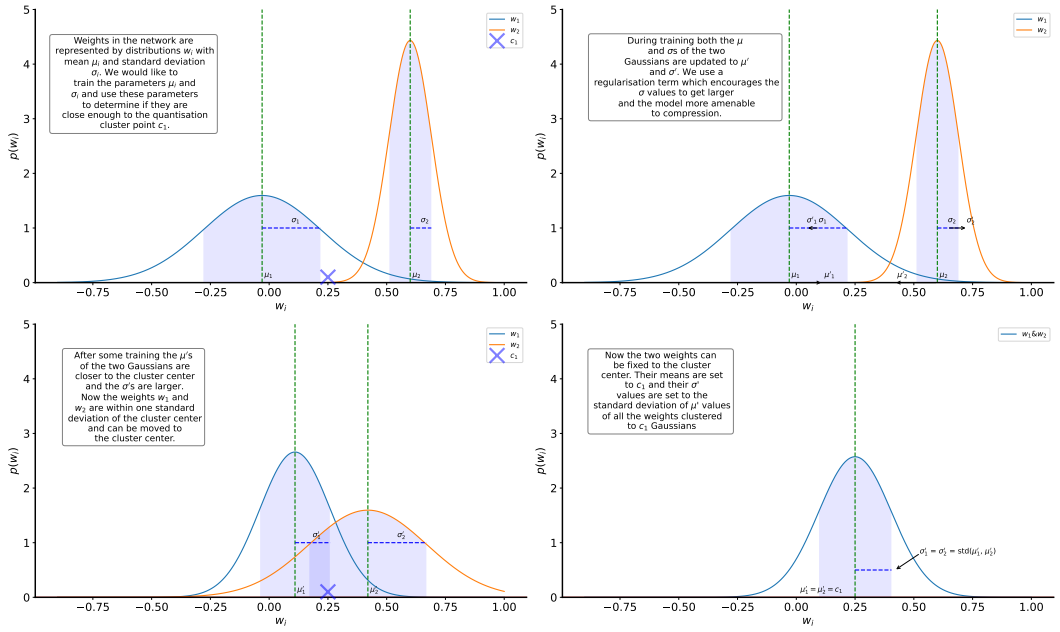

Figure 1: An overview of the PWFN process.

network while reducing its complexity. We build upon previous attempts [Wu et al., 2018b, Achterhold et al., 2018] to use a probabilistic framework to do so. The idea is to use a probability distribution to capture the flexibility in weight values, informing clustering decisions which reduce the entropy of the network and lower the unique parameter count without performance degradation.

We look to make progress on this problem with the use of Bayesian neural networks and a variational relaxation to identify optimal cluster configurations for the compression of neural networks with a method we call probabilistic weight fixing networks (PWFN). By incorporating the insights from the previous works on minimizing the relative weight-to-cluster distance [Subia-Waud and Dasmahapatra, 2022] and favoring additive powers-of-two cluster centroid values [Li et al., 2019b], we propose a novel initialization setting. Further, we discover that a simple regularization term which encourages maximal noise resilience is sufficient to avoid the variance of the weights' distribution from collapsing to zero. This regularization term facilitates the training of model-dataset pairings that were previously considered intractable when employing the variational Bayes-by-backdrop approach [Blundell et al., 2015]. These changes, when combined with a novel iterative clustering procedure, enable us to achieve superior compressibility. We can represent ResNet family models trained on the ImageNet dataset with fewer unique parameters and a reduced weight-space entropy while maintaining higher accuracy than the current state-of-the-art. Furthermore, we demonstrate the effectiveness of applying PWFN to transformer-based architectures, again achieving state-of-the-art results.

Weight-sharing quantization. Consider a neural network that consists of N weights $w = \{w_1, w_2, \ldots, w_N\}$. Each weight is typically unique and can take up to 2^b distinct values, where b is the bit-width of the number system used. However, this poses a challenge since each time a weight is used, at least one memory read is required. Since memory reads dominate the computational cost of inference, it is desirable to reduce them. Weight-sharing quantization addresses this challenge using a reduced pool of cluster centers $c = \{c_1, c_2, \ldots, c_k\}$, with $k \ll N$ and defining a map $w \to c$, where each weight $w_i \in w$ is mapped to a cluster center: $w_i \mapsto c_k \in c$.

Two considerations are pivotal for formulating this process: determining which values should be included in the reduced cluster pool c, and identifying the mapping of weights in w to the clusters in c.

Which values should be in the cluster pool? Insights from the literature highlight that pre-trained weight distributions often adhere to a heavy-tailed distribution [Barsbey et al., 2021, Hodgkinson and Mahoney, 2021, Gurbuzbalaban et al., 2021]. Moreover, hardware implementations show a preference for powers-of-two values for weights due to their cost-effective multiplication properties

[Vogel et al., 2019, Przewlocka-Rus and Kryjak, 2023, Lee et al., 2017]. While some methods advocate for exclusive use of powers-of-two as cluster centers [Zhou et al., 2017], others propose a more exible approach, suggesting additive powers-of-two [Li et al., 2019b, Subia-Waud and Dasmahapatra, 2022] – but only if it doesn't compromise performance. In our study, we align with this perspective, populating the cluster pool with powers-of-two values, but making exceptions for additive powers-of-two when they enhance performance.

How to identify which of the weights should be moved to which cluster? Previous studies have shown that distance metrics can be utilized to determine which xed clusters can accommodate certain weights without negatively affecting the performance. Weight-sharing clustering techniques can rely on these metrics to assign weights to clusters. The commonly used distance metrics include Euclidean distance [Wu et al., 2018b] and Relative movement distance [Subia-Waud and Dasmahapatra, 2022]. However, these metrics implicitly assume that all weights with the same value should be treated equally, irrespective of their location in the network. This may not be valid as moving a small (large) weight by a small (large) distance affects classi cation outcome depending on where the weight is in the network. To overcome this limiting assumption, we apply in this paper a Bayesian neural network (BNN) training approach to obtain a metric to determine the allowed movement of weights to clusters.

2 Related Work

Bayesian Neural Networks.BNNs attempt to use the methods of Bayesian inference in modeling predictive problems. Rather than the weights in a network coming from point estimates (i.e., a single value for each weight), a BNN attempts to model many (ideally all) con gurations of weight values throughout a network and make predictions, weighting each con guration by its probability. Exact Bayesian inference on the weights would require the computation of the integral $P(y|x, D) = \int_w P(y|x; w)P(w|D)dw$ where predictions for each allowed w are averaged over. Unfortunately, the marginalization over $P(w|D)$ is intractable for even simple networks, so approximations are needed. Approaches to this include Laplace approximation [MacKay, 1992], gradient MCMC [Welling and Teh, 2011], expectation propagation updates [Hernández-Lobato and Adams, 2015], and variational methods such as Bayes-by-backprop (BBP) [Blundell et al., 2015]. Using dropout at inference time has also been shown to be a variational approximation [Gal and Ghahramani, 2016] which is much more amenable to scale – albeit with reduced expressive power [Jospin et al., 2022]. BBP, of which our approach uses a variation of, models each of the network weights as coming from a Gaussian distribution and uses the re-parameterization trick for gradient descent to learn the parameters of the weight distributions. We give the derivations to do so in the Appendix. BBP has been demonstrated to work well in more complex model-dataset combinations than other BNN approaches (aside from dropout) but is still not able to scale to modern architectures and problems such as the ImageNet dataset and Resnet family of networks [Jospin et al., 2022].

One reason for the limited applicability is that the weight prior, which is a scaled mixture of two zero-mean Gaussians [Blundell et al., 2015], is not a well-motivated uninformative prior [Fortuin, 2022]. Instead, we initialize our training from a pre-trained network, in the spirit of empirical Bayes, and propose a prior on weight variances that encourages noise-resilience. A noise-resilient network can be thought of as one with a posterior distribution with large variances at its modes for a given choice of weight coordinates. Thus, small changes in weight space do not meaningfully alter the loss. The corresponding characterization of the at minima in the loss landscape is well supported in the optimization literature [Foret et al., 2020, Kaddour et al., 2022].

Quantization. Quantization is a technique used to reduce the number of bits used to represent components of a neural network in order to decrease energy costs associated with multiplication of 32-bit oating-point numbers. There are different approaches to quantization, such as quantizing weights, gradients, and activations. Clipping+scaling quantization maps the weight w_i to $w_i^0 = \text{round}(\frac{w_i}{s})$, where round() is a predetermined rounding function and s is a scaling factor learned channel-wise, layerwise or with even further granularity. Quantization can be performed post-training or with quantization-aware training [Jacob et al., 2018].

Weight sharing quantization uses clustering techniques to group weights and x the weight values to their assigned cluster centroid. These weights are stored as codebook indices, which enables compressed representation methods like Huffman encoding to compress the network further. Unlike

Figure 2: For DeiT small, we show a box plot of the entropies and unique counts per input channel for each Q,K,V by layer and with the mean of each layer (calculated across all attention heads) shown in the black lines.

clipping+scaling quantization, weight sharing techniques share the pool of weights across the entire network. Several studies have proposed different methods for weight sharing quantization. For example, Zhou et al. [Zhou et al., 2017] restrict layerwise rounding of weights to powers-of-two for energy-cheap bit-shift multiplication, while [Li et al., 2019a] suggest additive powers-of-two (APoT) to capture the pre-quantized distribution of weights better. [Wu et al., 2018a] use a spectrally relaxed k-means regularization term to encourage the network weights to be more amenable to clustering and focus on a filter-row codebook for convolution. Similarly, [Stock et al., 2020] and [Fan et al., 2020] focus on quantizing groups of weights into single codewords rather than the individual weights themselves. [Ullrich et al., 2017] use the distance from an evolving Gaussian mixture as a regularization term to prepare the clustering weights. However, their approach is computationally expensive. In contrast, our formulation reduces computation by adding cluster centers iteratively when needed and only considering weights that are not already fixed for a regularization prior. A work closely related to ours is that of [Achterhold et al., 2018], where the authors formulate post-training quantization and pruning as a variational inference problem with a `quantizing prior' but due to its optimization complexity and difficulties with weights not being drawn tightly enough into clusters the method was only demonstrated to work on simple dataset-model combinations and for the case of ternary quantization. WFN [Subia-Waud and Dasmahapatra, 2022] is a recent weight-sharing approach that uses the minimizing of the relative movement distance to determine which weights are to be clustered and has demonstrated that a low weight-space entropy and few unique weights with a single codebook can maintain accuracy. In our work, we go much further in reducing weight-space entropy and unique weight count by using a context-aware distance metric and probabilistic framework in determining which weights can be moved where. Additionally, almost all of the works reviewed do not quantize the first and last layers [Li et al., 2019a, Jung et al., 2019, Zhou et al., 2016, Yamamoto, 2021, Oh et al., 2021, Li et al., 2022] in order to maintain performance and in some cases don't quantize the bias terms [Li et al., 2022], - we challenge this view and attempt a full network quantization, leaving only the batch-norm and layer-norm parameters at full precision.

3 PWFN

In PWFN, we follow Tixing iterations each of which combines a training and a clustering stage in order to reach an outcome of a highly compressed/quantized network with a single whole-network codebook. We model each individual weight w_i as a draw from a distribution with learnable parameters and use these parameters as guidance for the clustering stage. We model each w_i as coming from a Gaussian distribution $\mathcal{N}(\mu_i; \sigma_i)$ and during the training stage we use a form of BBP to train the weight distribution parameters $\mu = (\mu_1, \ldots, \mu_N)$ and $\sigma = (\sigma_1, \ldots, \sigma_N)$ to minimize both the task performance loss and to encourage the weight distributions to tell us exactly how much noise can be added to each weight without affecting performance. Both μ and σ are trained with an additional regularization term that encourages larger values of σ to counter the model reverting to the point estimates with $\sigma_i = 0$ to minimize the classification loss. During the clustering stage, we look

Figure 3: The regularization term acts to stop the uncertainty values from collapsing to zero. This experiment is run using the Cifar10 dataset with ResNet-18, stopping after 30 epochs.

to use this information to move the μ_i values to one of a handful of cluster centers. We favour the cluster centers to be hardware multiplication-friendly powers-of-two, or additive-powers-of-two. After T iterations of training and clustering, each of the weights' distributions in the networks will have their μ values centered on one of the k clusters c in the codebook.

After the T fixing iterations there are two options depending on the downstream usage of the network: either the network can be converted into point estimates and the weights set to the exact μ values giving us a quantized network. Or, we can use the extra information given to us by modelling each weight as a distribution as a way of quantifying uncertainty of a particular prediction. If, after multiple samples of w, a model changes its prediction for a fixed input, this tells us that there is uncertainty in these predictions – with this information being useful for practical settings.

PWFN Training. Consider a network parameterized by weights $w = \{w_1, \ldots, w_N\}$. In PWFN, each weight w_i is not a single value but is instead drawn from a distribution $w_i \sim N(\mu_i, \sigma_i)$, and instead of learning the w_i directly, the learning process optimizes each μ_i and σ_i. In a forward pass, during training, we sample weight values w_i according to its distribution:

$$w_i = \mu_i + \sigma_i \cdot \epsilon, \quad \epsilon \sim N(0, 1). \tag{1}$$

The forward pass is stochastic under fixed μ, σ. If trained correctly, the σ_i values give us information about the amount of noise a particular weight w_i can handle without affecting performance. Said another way, if we can find a configuration $w = (\mu, \sigma)$ which maintains task performance despite the randomness introduced by the σ parameters, then we will know which of the corresponding weights can be moved and to what degree. In PWFN, we train following the BBP optimization process [Blundell et al., 2015] with some changes both in terms of initialization and the priors on μ and σ.

Large σ constraint for w. Given the usual cross-entropy or other performance loss, there is a clear direction of travel during gradient descent towards having small σ values and less uncertainty in the network parameters. A prior on the distribution of weights is therefore needed to prevent the $\sigma = 0$ point estimate solution being found which would leave us with no weight movement information. In the original BBP set-up, the authors looked to prevent vanishing variance by regularising the distribution of weights according to a prior distribution of a mixture of zero-mean Gaussian densities with different variances (the parameters of the prior they find through an exhaustive search). The motivation for doing so was because the empirical Bayes approach didn't perform well due to the network favoring updating these parameters over the posterior (since there are fewer) and the link to the successful spike-and-slab prior [Mitchell and Beauchamp, 1988] — where values are concentrated around 0 (the slab) or another value known as the spike — favoring sparsity. Instead, we hypothesize that a good network can handle the most noise injection whilst maintaining performance. These networks are likely more compressible since they have been trained to accept changes to their weight values without performance degradation during training.

Figure 4: p_t follows the same schedule as [Subia-Waud and Dasmahapatra, 2022] (left). In the middle and right plots, we see that PWFN achieves very small entropy values by majority of weights to only a very small (4 or 5) cluster values and the rest are assigned as outliers, most of which are powers-of-two.

We attempt this by making our values to be large. Networks with large have, probabilistically, more noise added to the values during training and so have to learn to have robust performance under such circumstances. We note that this acts as a push-pull relationship with the performance loss, which favours low values. The motivation is that, much like L_1 norms enforcing sparsity, this formulation will train the network to produce a large for noise-resilient parameter w_i, whilst maintaining a noise-sensitive weight to have a small $_j$ despite the prior pull. The regularised loss function for training the training phases of the algorithm is:

$$\log P(D\mathrm{j} \quad ; \quad) + \quad L_{REG}(\quad); \tag{2}$$

where the regularization term is:

$$L_{REG}(\quad) = \sum_{i=1}^{N} L(\ _i) = \sum_{i=1}^{N} (\ _i \quad S) (\ S \quad \ _i); \tag{3}$$

with $(\ x) = 1$ for $x \quad 0$ and 0 otherwise. The function prevents the optimization from nding a network with a subset of with in nitely large values and dominating the cross entropy term. S is thus a cutoff on how large the values in can be. is a hyperparameter controlling the formation of a noise-resilient network where the majority of the weights can receive noise injection without hurting performance, not just a few. In Figure 3 we illustrate the effect on the distribution of under different values for a ResNet-18 trained on the Cifar-10 dataset. As we increase these values no longer collapse to zero giving us information for downstream clustering.

Initialization using Relative Distance from Powers-of-two. For each weight w_i we need to specify its prior distribution so as to derive the posterior using Bayesian updating. We assume that the posterior distribution is Gaussian with a diagonal covariance matrix $P(w_i; \ _i; \ _i)$ whose parameters $_i$; $_i$ are trained using BBP. To initialize the prior distributions for and $_i$ we set $P^0(\ ) = \prod_i P^0(\ _i)$ where $P^0(\ _i) / \ _{i;w_i}$ for the pre-trained weight value w_i. For a Gaussian posterior we would typically require an unknown to be drawn from a Gamma conjugate prior distribution. Instead, we set to be a known function of the $_i$ at initialization. In [Subia-Waud and Dasmahapatra, 2022] relative distances to the preferred powers of two values for the weight was used to determine weight movement. To favour anchoring weights at powers of two, we set the standard deviations to be smallest $2 \ (^{30})$ at either edge of each interval between the nearest integer powers of two, $(2^{x_i} \quad _i \quad 2^{x_i+1})$ for integer x_i, and largest at the midpoint of the interval. We introduce a parabolic function $_i(\ _i)$ as a product of relative distances of each pre-trained weight value w_i to the nearest lower and upper powers of two:

$$_i(\ _i) = (0 :05)^2 \quad \frac{j2^{x_i} \quad _i j}{j2^{x_i} j} \quad \frac{j \ _i \quad 2^{x_i+1} j}{j2^{x_i+1} j} \quad ; \tag{4}$$

(Full details and visualization is given in the appendix).

PWFN Clustering. In Figure 1 we show a schematic of the clustering stage in which we use the information garnered from the weights' distribution parameters to identify cluster centers and their

assignment. PWFN clustering is a two-step method running for $t = 1, \ldots, T$ iterations. At each step we set a fraction p_t of the weights to be fixed, so that $|W^t_{\text{fixed}}| = N p_t$. The remaining weights at iteration stage t are trainable and called W^t_{free}. We follow the scheme first proposed in [Subia-Waud and Dasmahapatra, 2022] in setting p_t (Figure 4, left). All of the weights w_i that are assigned to W^t_{fixed} will have their μ_i values fixed to one of the set of cluster centers. At the last iteration $|W^T_{\text{free}}| = 0$ and $p_T = 1$, as all weights have been fixed to their allocated cluster centroids.

We next introduce how a cluster center c_k is defined and how the mapping $\mu_i \mapsto c_k \in c$ is performed. Let $R = \{\frac{1}{2^b}, \ldots, \frac{1}{2^{j+1}}, \frac{1}{2^j}, 0, \frac{1}{2^j}, \frac{1}{2^{j+1}}, \ldots, \frac{1}{2^b}\}$ be the set of all powers-of-two up to a precision b. For a weight to be a desired additive power of two, a sum over at most r elements of R is defined to be a cluster center of order r. Formally, for $P(R)$ the power set of R,

$$c^r = \{\sum_{i \in r} i \mid r \in P(R) \wedge |r| \leq r\}. \tag{5}$$

PWFN begins with order $r = 1$, the powers-of-two up to precision b as the proposal cluster set c^1. Next, for each weight $w_i = (\mu_i, \sigma_i)$ in the network, the value of μ_i is used to determine how far away they are from each of the cluster centers using:

$$D_{\text{prob}}(w_i, c_j) = \frac{|\mu_i - c_j|}{\sigma_i}. \tag{6}$$

Interpret this Mahalanobis distance as: "how many sigmas (standard deviations) away is cluster $c_j \in c^r$ from weight w_i". At iteration stage t, for each free weight we define $c^t(i) = \min_{c \in c^r} D_{\text{prob}}(w_i, c)$ as the cluster center that is the fewest sigmas away from $w_i \in W^t_{\text{free}}$. We denote by n^r_k the number of weights with the smallest D_{prob} to cluster c^r_k, i.e., $n^r_k = \sum_i \mathbb{I}[c^r_k = c^r(i)]$. We then take the index k of the cluster with the most number of weights nearest to a cluster, $k = \text{argmax}_k\, n^r_k$. Thus, $c^r_k \in c^r = (c^r_1, \ldots, c^r_k)$ is the cluster with the most number of weights nearest to it.

We then order the weights in W^t_{free} by their distance to c^r_k. In detail, for $W^t_{\text{free}} = [w_1, \ldots, w_i, \ldots, w_n]$, we reorder the weights by permuting the indices $w^0_i = w_{\pi(i)}$ where $\pi : [1, \ldots, n] \to [1, \ldots, n]$ is a permutation $i \mapsto \pi(i)$. The ordered list $[w^0_1, \ldots, w^0_n]$ satisfies

$$D_{\text{prob}}(w^0_i, c^r_k) \leq D_{\text{prob}}(w^0_{i+1}, c^r_k) \tag{7}$$

Next, we need to determine how many of these weights we should assign to c^r_k. To do so, we define a threshold τ and we take the first $\ell(\tau)$ weights from $[w^0_1, \ldots, w^0_n]$ such that:

$$\frac{1}{\ell(\tau)} \sum_{i=1}^{\ell(\tau)} D_{\text{prob}}(w^0_i, c^r_k) \leq \tau. \tag{8}$$

As long as this is possible with $\ell(\tau) > 0$, we have identified both a cluster c^r_k and set of weights $[w^0_1, \ldots, w^0_{\ell(\tau)}]$ which can be moved from W^t_{free} to W^{t+1}_{fixed}. We map the weights in $[w^0_1, \ldots, w^0_{\ell(\tau)}] = [(\mu^0_1, \sigma^0_1), \ldots, (\mu^0_{\ell(\tau)}, \sigma^0_{\ell(\tau)})]$ to a single weight $w_k = (\mu_k, \sigma_k)$ corresponding to cluster c^r_k: $\mu_k = c^r_k$ and $\sigma_k = \text{std}([\mu^0_1, \ldots, \mu^0_{\ell(\tau)}])$ where std computes the standard deviation of its argument. This process is then repeated, finding the next most popular cluster until no weights are assigned a cluster. If $\ell(\tau) = 0$, before enough weights are assigned in iteration t, then we have not been able to find any cluster centers $c_j \in c^r$ which are close enough to any weights, i.e., $D_{\text{prob}}(w_i, c_j) > \tau$ for all weights $w_i \in W^t_{\text{free}}$ and $c_j = c_k$. In this case, we set $r \mapsto r + 1$ and $\tau \mapsto 2\tau$ giving us a higher-order additive powers-of-two set and less restrictive value threshold. Since $|c^{r+1}| > |c^r|$, this increase in r makes more cluster centers available during the next clustering attempt.

Putting it All Together. Putting the training and clustering stages together, we have a process for training a neural network whose weights are from a Gaussian posterior distribution with diagonal covariance matrix by backpropagation (BPP) that favours configurations with long Gaussian tails, which the clustering stage can then use to identify which weights to move and to what extent. This process is repeated for T iterations, with the fraction p_t of weights increasing with each $p_{t+1} > p_t$ until all of the weights are moved from W_{free} to W_{fixed} at iteration T where $p_T = 1$. We give the full algorithm in the Appendix.

Table 1: Full comparison results. (w/o FL-Bias) refers to calculating the metrics without the rst-last layers and bias terms included. `Params' refers to the unique parameter count in the quantized model, entropy is the full weight-space entropy. In-ch, layer, attn refer to whether the method uses a separate codebook for each layer, in-channel and attention head respectively.

		Separate Codebook					
Model	Method	Layer	In-ch	Attn	Top-1 (Ensemble)	Entropy	Params
ResNet-18	Baseline	-	-	-	68.9	23.3	10756029
	LSQ	!	7	-	68.2	-	-
	APoT	!	7	-	69.9	5.7	1430
	WFN	7	7	-	69.7	3.0	164
	PWFN (no prior)	7	7	-	69.3 (69.6)	1.7	143
	PWFN	7	7	-	70.0 (70.1)	2.5	155
ResNet-34	Baseline	-	-	-	73.3	24.1	19014310
	LSQ	!	7	-	71.9	-	-
	APoT	!	7	-	73.4	6.8	16748
	WFN	7	7	-	73.0	3.8	233
	PWFN (no prior)	7	7	-	73.5 (74.4)	1.2	147
	PWFN	7	7	-	74.3 (74.6)	1.8	154
ResNet-50	Baseline	-	-	-	76.1	24.2	19915744
	LSQ	!	7	-	75.8	-	-
	WFN	7	7	-	76.0	4.1	261
	PWFN (no prior)	7	7	-	77.2 (78.1)	3.5	334
	PWFN	7	7	-	77.5 (78.3)	3.4	325
DeiT-Small	Baseline	-	-	-	79.9	16.7	19174713
	LSQ+	!	!	7	77.8	-	-
	Q-ViT	!	!	!	78.1	11.3	3066917
	Q-ViT (w/o FL-Bias)	!	!	!	78.1	10.4	257149
	PWFN (no prior)	7	7	7	78.0 (78.3)	2.7	352
	PWFN	7	7	7	78.1 (78.5)	2.7	356
DeiT-Tiny	Baseline	-	-	-	72.9	15.5	5481081
	LSQ+	!	!	7	68.1	-	-
	Q-ViT	!	!	!	69.6	11.5	1117630
	Q-ViT (w/o FL-Bias)	!	!	!	69.6	10.5	128793
	PWFN (no prior)	7	7	7	71.4(71.6)	2.8	300
	PWFN	7	7	7	71.2 (71.5)	2.8	296
DenseNet161	Baseline	-	-	-	77.8	17.1	26423159
	PWFN	7	7	7	77.6 (78.0)	1.1	125

4 Experiments

We conduct our experimentation on the ImageNet dataset with a wide range of models: ResNets-(18,34,50) [He et al., 2016], DenseNet-161 [Huang et al., 2017] and the challenging DeiT (small and tiny) [Touvron et al., 2021]. For each model, we convert all the parameters in the convolution and linear layers into Gaussian distributions where the mean value is set to be the weight value of the pre-trained model found in the Timm library. Thus, at test time with no further training, we retain the original accuracies. We set the variance parameters according to the setting described in Eq (12). We then apply nine rounds of the described weight xing with three epochs of re-training each round, totalling to 27 epochs of training. We train using SGD with momentum 0.9 with a learning rate of 0.001. For all experiments, we x= 1, = 2 ^{11} which we found using grid-search on the Cifar-10 dataset and works surprisingly well in all settings. For all our experiments we train using 4x RTX8000 GPUs and a batch-size of 128. For the ensemble results, we sample 20 times different weights using the learned weights' distributions and report the mean accuracy.

5 Results

We compare PWFN against a range of quantization approaches where the model weights have been made available so that we can make accurate measurements of entropy and unique parameter count.

Table 2: Comparison of the number of additional training epochs required by different ne-tuning quantization methods.

Method	Num of additional epochs
ApoT	120
PWFN	27
WFN	27
LSQ	90
QviT	300

For the ResNet family, we compare against the current state-of-the-art APoT [Li et al., 2019b][1] and WFN [Subia-Waud and Dasmahapatra, 2022][2]. For the transformer models, there has only been one work released, Q-Vit [Li et al., 2022][3], which has both the model saves and code released. For both APoT and Q-Vit, we compare the 3-bit models which are the closest in terms of weight-space entropy to that achieved by PWFN.

As presented in Table 2, PWFN requires substantially fewer additional training epochs than most methods, save for WFN, highlighting its training ef ciency. We use a straightforward regularization term that encourages an increase in $l1$ and its computational cost is comparable to that of $l1$ regularization. While our approach does lead to greater memory demands due to the additional parameters and their associated gradient updates, the overall simplicity of the method is more ef cient than previous BNN training procedures making it feasible to tackle more complex model-dataset pairings. Additionally, we note that when using the quantized version for inference, there are no extra costs, and the BNN functions as a point-estimate network.

In Table 1 we can see the set of results. PWFN demonstrates superior entropy, unique parameter count and top-1 accuracy across the board. In addition to the point-estimate accuracy using the mean of each of the weights' distributions (the cluster centers), we can additionally sample the weights from the learned distributions to give us an ensemble of models the mean prediction of which gives us further accuracy gains which we show in brackets in the Table. The prior initialization gives a slight but consistent accuracy improvement over using a uniform prior (PWFN (no prior)). We note that for both APoT and Q-Vit, different codebooks are used for different layers and for Q-Vit, different codebooks were additionally used for every attention head and input channel, and the bias terms were left unquantized, pushing up the parameter count and weight-space entropy substantially. We highlight this as a growing trend in the eld, where relaxations such as leaving large parts of the network unquantized, or using different codebooks for ever granular parts of the network, are often used. Each relaxation comes at a cost in hardware, be that support for unquantized elements – such as the rst and last layers – or the use of different codebooks for various parts of the architecture. Figure 2 illustrates the variation in entropy and the count of unique parameters across different layers and attention components. A notable observation from our study is that the weights associated with the `value' component exhibit higher entropy in the nal layer. This observation aligns with the notion that employing a xed quantization scheme for each layer necessitates a relaxation of the quantization constraints speci cally for the last layer, as supported by prior studies [Li et al., 2019a, Jung et al., 2019, Zhou et al., 2016, Yamamoto, 2021, Oh et al., 2021, Li et al., 2022]. Moreover, this highlights an intriguing possibility that in the context of attention networks, such relaxation might be essential only for the 'value' weights, and not for the 'keys' and 'queries'.

In understanding how PWFN is able to compress a network's representation to such a degree compared to WFN we look to how often the previously proposed relative distance threshold is maintained.

In Figure 5, it's evident that while the relative distance threshold established in WFN is, on average, maintained, there are edge-cases where it isn't. This observation suggests that having a context-speci c noise tolerance bene ts subsequent compression stages. Furthermore, the data indicates that these values are typically small (as seen in the left column), have a high frequency of occurrence (depicted in the middle), and are predominantly assigned during the middle (0.6, 0.7) and nal rounds.

[1] https://github.com/yhhhli/APoT_Quantization
[2] https://github.com/subiawaud/Weight_Fix_Networks
[3] https://github.com/YanjingLi0202/Q-ViT

Figure 5: The maximum (top left) and mean (bottom left) relative distance a weight moves to a cluster by cluster value. The maximum relative distance is not well maintained with the number of weights assigned to that cluster (top middle), but the mean relative distance is (bottom middle). The maximum (top) relative distance for each cluster assignment and mean (bottom) relative distance by round are shown in the right-hand column. In all plots, we show in blue the threshold used in WFN.

6 Conclusion

This work proposed PWFN, a training and clustering methodology that can both scale BNNs to complex model-dataset combinations and then use the weight uncertainties to inform quantization decision-making. The result is a set of networks with extremely low weight-space entropies that can be used downstream in accelerator designs to mitigate expensive data movement costs. Additionally, we have seen the potential of the probabilistic aspect of the learned networks with a sampled ensemble giving noticeable accuracy gains. An exciting direction for future work is to explore how the uncertainty estimations and the out-of-distribution performance of neural networks could be enhanced using PWFN to train Bayesian Neural Networks.

7 Acknowledgments

This work was supported by the UK Research and Innovation Centre for Doctoral Training in Machine Intelligence for Nano-electronic Devices and Systems [EP/S024298/1]

References

Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In International Conference on Learning Representations, 2018.

Melih Barsbey, Milad Se dgaran, Murat A Erdogdu, Gael Richard, and Umut Simsekli. Heavy tails in sgd and compressibility of overparametrized neural networks. Advances in Neural Information Processing Systems, 34:29364–29378, 2021.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In International conference on machine learning, pages 1613–1622. PMLR, 2015.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

Vincent Fortuin. Priors in bayesian deep learning: A review. *International Statistical Review*, 90(3): 563–591, 2022.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

Mert Gurbuzbalaban, Umut Simsekli, and Lingjiong Zhu. The heavy-tail phenomenon in sgd. In *International Conference on Machine Learning*, pages 3964–3975. PMLR, 2021.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*, pages 1861–1869. PMLR, 2015.

Liam Hodgkinson and Michael Mahoney. Multiplicative noise and heavy tails in stochastic optimization. In *International Conference on Machine Learning*, pages 4262–4274. PMLR, 2021.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

Jean Kaddour, Linqing Liu, Ricardo Silva, and Matt J Kusner. When do flat minima optimizers work? *Advances in Neural Information Processing Systems*, 35:16577–16595, 2022.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Edward H Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and S Simon Wong. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904. IEEE, 2017.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2019a.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *arXiv preprint arXiv:1909.13144*, 2019b.

Zhexin Li, Tong Yang, Peisong Wang, and Jian Cheng. Q-vit: Fully differentiable quantization for vision transformer. *arXiv preprint arXiv:2201.07703*, 2022.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Toby J Mitchell and John J Beauchamp. Bayesian variable selection in linear regression. *Journal of the american statistical association*, 83(404):1023–1032, 1988.

Sangyun Oh, Hyeonuk Sim, Sugil Lee, and Jongeun Lee. Automated log-scale quantization for low-cost deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 742–751, 2021.

Dominika Przewlocka-Rus and Tomasz Kryjak. Energy efficient hardware acceleration of neural networks with power-of-two quantisation. In *Computer Vision and Graphics: Proceedings of the International Conference on Computer Vision and Graphics ICCVG 2022*, pages 225–236. Springer, 2023.

Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. In *ICLR 2020-Eighth International Conference on Learning Representations*, pages 1–11, 2020.

Christopher Subia-Waud and Srinandan Dasmahapatra. Weight fixing networks. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pages 415–431. Springer, 2022.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Sebastian Vogel, Rajatha B Raghunath, Andre Guntoro, Kristof Van Laerhoven, and Gerd Ascheid. Bit-shift-based accelerator for cnns with selectable accuracy and throughput. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 663–667. IEEE, 2019.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep -means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. *35th Int. Conf. Mach. Learn. ICML 2018*, 12:8523–8532, 2018a.

Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *International Conference on Machine Learning*, pages 5363–5372. PMLR, 2018b.

Kohei Yamamoto. Learnable companding quantization for accurate low-bit neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5029–5038, 2021.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

8 Appendix

Bayes-by-backprop and PWFN The Bayesian posterior neural network distribution $P(w|D)$ is approximated by a distribution $Q(w|\ )$ whose parameters are trained using back-propagation, Bayes-by-backprop (BPP). The approximation is achieved by minimizing the Kullback-Leibler (KL) divergence $D_{KL}[Q||P]$ between P and Q to find the optimal parameters . These parameters instantiate the means $_i$ and variances ^2_i of the PWFN.

$$\begin{aligned} &= \arg\min KL[Q(w|\ )||P(w|D)]; \text{ where} \\ KL[Q(w|\ )||P(w|D)] &= \mathrm{E}_{Q(w|\ )} \log \frac{Q(w|\ )}{P(w|D)} = \mathrm{E}_{Q(w|\ )} \log \frac{Q(w|\ )P(D)}{P(D|w)P(w)} : \end{aligned} \tag{9}$$

The $P(D)$ term does not contribute to the optimization and is dropped, leaving

$$\begin{aligned} = \arg\min & \sum_{Q(w|\ )} [\log Q(w|\ ) \quad \log P(D|w) \quad \log P(w)]; \\ \arg\min_m & \underbrace{\log Q(w^m|\ )}_{\text{prior dependent}} \underbrace{\log P(w^m)} \quad \underbrace{\log P(D|w^m)}_{\text{data dependent}}; \end{aligned} \tag{10}$$

where the expectation value is approximated by samples w^m $Q(w|\ )$ drawn from $Q(w|\ )$ each of which instantiates a neural network.

Gradient descent over each w^m that instantiates a neural network is made possible by the *re-parametrization trick* [Kingma and Welling, 2013, Blundell et al., 2015]. The idea is to regard each sample $w^m = \ + \ _m$ where $_m$ $p(\ )$ is a random draw from some distribution that we take to be an isotropic Gaussian: $p(\ ) = N(0; I)$ with I the N-dimensional identity matrix for the N weights of network W. These weights w^m are used in the forward pass through the network while parameters and are trainable. Then, for any function $f(w)$ we have $\mathrm{E}_{Q(w|\ )}[f(w)] = \mathrm{E}_{p(\ )}[f(w)]$, so that

$$\frac{@}{@} \mathrm{E}_{Q(w|\ )}[f(w; \ )] = \frac{@}{@} \mathrm{E}_{p(\ )}[f(w; \ )] = \mathrm{E}_{p(\ )}[\frac{@f(w; \ )}{@w} \frac{@w}{@} + \frac{@f(w; \ )}{@}].$$

The terms are all calculable, allowing us to draw from a distribution for each weight w^m and backpropagate to the underlying distribution parameters $:= (\ ; \ )$. For $w^m = \ + \ _m$, the derivatives are $\frac{@w^m}{@} = I$, and $\frac{@w^m}{@} = \ _m I$, making the respective gradients

$$r \ = \frac{@f(w; \ ; \ )}{@w} + \frac{@f(w; \ ; \ )}{@} \text{ and } r \ = \frac{@f(w; \ ; \ )}{@w} \ _m + \frac{@f(w; \ ; \ )}{@}. \tag{11}$$

PWFN Clustering Algorithm

Prior Initialization

In addition to the prior initialization described in main paper, we added an reweighting determined by the size of the values in the network. Using the definition of $v = D_{\text{rel}}(\ _i; \text{pow2}_u(\ _i))$ we re-weight by the third quartile $\forall_{0:75}$ and re-write the initialization as:

$$f(\ _i) = 0{:}0025 \quad \frac{D_{\text{rel}}(\ _i; \text{pow2}_u(\ _i)) \quad D_{rel}(\ _i; \text{pow2}_d(\ _i))}{\forall_{0:75}}; \tag{12}$$

while $|W^{t+1}_{\text{fixed}}| \;\; Np_t$ **do**
  fixed$_{\text{new}}$ $\;\;$ [ ]
  while fixed$_{\text{new}}$ *is empty* **do**
    Increase the order $\omega \;\; \omega + 1$
    Increase $\;\; 2$
    $C^{\omega} \;\; \{\sum_{i \in r} \;\; | \; r \in P(R) \wedge |r| \;\; \omega\}$
    for each $i = 1, \ldots, |W^{t+1}_{\text{free}}|$
      $c^{\omega}(i) \;\; \min_{c \in C^{\omega}} D_{\text{prob}}(w_i, c)$
    for each cluster center $c^{\omega}_k \in C^{\omega}$
      $n^{\omega}_k \;\; \sum_i \mathbb{I}[c^{\omega}_k = c^{\omega}(i)]$
    $k \;\; \arg\max_k n^{\omega}_k$
    Sort: $[w'_1, \ldots, w'_N] \;\; [w_1, \ldots, w_N]$, $w'_i = w_{\;(i)}$, $\;$ permutation
      where $D_{\text{prob}}(w'_i, c^{\omega}_k) < D_{\text{prob}}(w'_{i+1}, c^{\omega}_k)$
    $i \;\; 1$, mean $\;\; D_{\text{prob}}(w'_1, c^{\omega}_k)$
    while mean $\;\;$ **do**
      fixed$_{\text{new}}$ $\;\; w'_i$
      mean $\;\; \frac{i}{i+1}$ mean $+ \frac{1}{i+1} \;\; D_{\text{prob}}(w'_{i+1}, c^{\omega}_k)$
      $i \;\; i + 1$
    end
    $t = t \;\;\;\; t = t$
  end
  Assign all the weights means $\;_i \in$ fixed$_{\text{new}}$ to cluster center $c^{\omega}(i)$ and set each of the $\;_i \in$ fixed$_{\text{new}}$ to be the variance of the weight means in fixed$_{\text{new}}$. Finally, move them from W^{t+1}_{free} to W^{t+1}_{fixed}
end

Algorithm 1: Clustering Np_t weights at the t^{th} iteration in PWFN.

and clamp the values to be within the range $[2^{-30}, 0.05]$ giving us our initial variance values.

$$\;_i = \max(0.1, \min(f(\;_i), 2^{-30})). \tag{13}$$