PRACTICAL NEURON-LEVEL PRUNING FRAMEWORK FOR BAYESIAN NEURAL NETWORKS

Vaclav Kubon
TU Delft
Honours Programme Bachelor

ABSTRACT

Bayesian Neural Networks (BNNs) offer uncertainty quantification but are computationally expensive, limiting their practical deployment. This paper introduces a neuron-level pruning framework that reduces BNN complexity while preserving predictive performance. Unlike existing weight-level pruning techniques, our approach removes entire neurons, enabling significant memory savings and inference speedups without requiring specialized hardware. We propose a pruning loss based on the Wasserstein distance, balancing model sparsity and predictive accuracy. Our method is fully automatic, eliminating the need for manual hyperparameter tuning. Experimental results on UCI regression and Fashion MNIST datasets demonstrate that our framework can prune over 80% of neurons while maintaining predictive distribution integrity. Additionally, we validate the Lottery Ticket Hypothesis in the Bayesian setting, showing that pruned subnetworks retain performance and learn faster when retrained. This work represents a step toward making BNNs more scalable for real-world applications.

Keywords Bayesian Neural Networks · Compression · Pruning · Wasserstein Distance

1 Introduction

The remarkable success of deterministic neural networks (DNNs) achieved in domains of NLP or vision can be largely attributed to the scaling of the training data and the number of model parameters [1]. The lottery ticket hypothesis paper [2] shows that although scaling up the model is needed to improve performance, once already trained, much smaller sub-networks with equivalent performance exist. Network pruning is one popular family of methods for finding such subnetworks by removing unimportant weights based on criteria such as weight magnitude [3][4], L1 norm [5] or loss sensitivity [6]. The compressed network takes less memory and processes inputs faster.

Compressing Bayesian Neural Networks (BNNs) is even more rewarding. Firstly, each weight takes the form of a prior distribution defined by multiple parameters [7]. Thus we get greater memory saving for each pruned weight. Secondly, wider adoption of BNNs is currently hindered by the higher computational costs compared to DNNs [8]. For instance, a BNN needs to execute multiple forward passes to estimate the posterior distribution for a single input. Thirdly, because of their ability to quantify prediction uncertainty, BNNs are often deployed on safety-critical edge devices such as sensors [9] or autonomous systems [10], where computational and memory constraints currently prevent us from running larger and more accurate networks [11].

Despite the potential, there is little research into whether BNNs can be pruned and how. We demonstrate that even though BNN weights are sampled from a distribution, activations of BNN

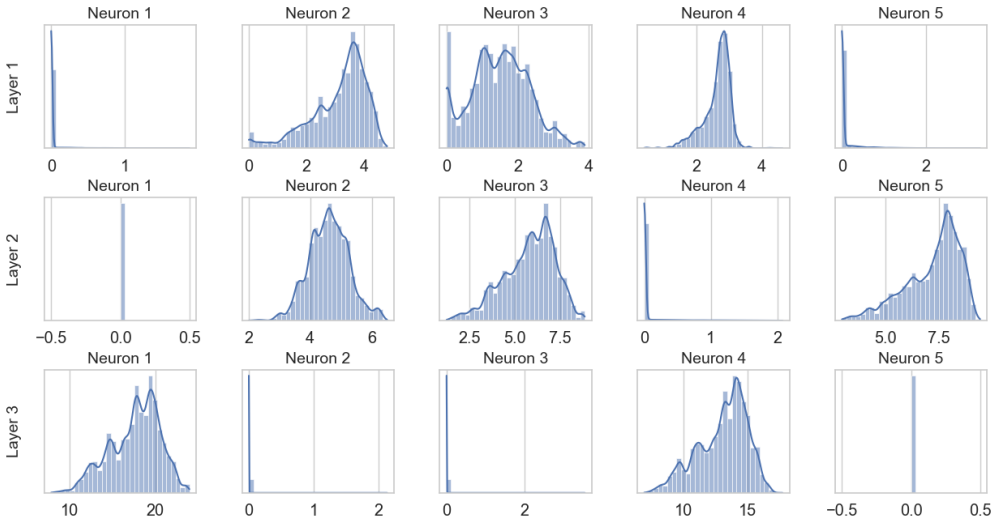


Figure 1: Empirical activation distributions for each neuron, as defined in Section 5. Each distribution represents activations over N data points for M samples. The BNN has 3 layers with a hidden dimension of 5, using the *ReLU* activation function.

neurons tend to get sparse as BNN learn. This effect is exemplified in Figure 1 and further discussed in Section 6. The empirical observation also demonstrates that pruning on the level of neurons is possible.

Traditional pruning techniques from DNNs cannot be directly applied to BNNs because of fundamental differences in performance measures. As a consequence of sampling weights from a distribution after several forward passes, we obtain a predictive distribution at the output (more in 3.1), which uniquely captures uncertainty in predictions. Therefore, when pruning the network, we not only care about maintaining the correctness of the network's decisions—given by point estimates of the predictive distribution and measured with accuracy/MSE—but "maintaining performance" refers to preserving the shape of the full predictive distribution. Maintaining the predictive uncertainty is critical, as it directly influences how trustworthy or reliable the decisions of the BNN are in practical scenarios.

To demonstrate this challenge, consider a popular DNN pruning technique that removes neurons with smaller activation magnitudes. An intuitive adaptation to the Bayesian setting would be considering the expectation of activations. For deterministic networks, removing neurons with low activations typically has minimal impact on network behavior. Extending this logic to BNNs by considering the expectation of neuron activations seems intuitive but overlooks a crucial complexity: due to the stochastic nature of weights and activations in BNNs, the expectation of the entire network's predictive distribution is not equal to simply propagating expectations layer by layer. Each layer's distribution depends on the full distributions from previous layers, making the impact of pruning on the network's overall functional behavior unclear and difficult to predict.

In this work, we define a general pruning loss that uses the Wasserstein metric to measure the trade-off between sparsity and deviation from the original predictive distribution. Furthermore, we derive an approximate solution to this objective in the form of a simple pruning criterion for measuring the impact of individual neurons on the pruning loss. Lastly, we present a routine that combines the loss and the criterion to find an optimal prune ratio. Our approach was able to prune majority of neurons in our experiment with UCI regression datasets and over 80% for the Fashion MNIST dataset.

Besides significant compression, our method proves to be very reliable at preserving the predictive posterior distribution. Furthermore, our method does not require the user to tune any hyperparameters such as the pruning ratio or pruning thresholds. Unlike other approaches for

BNN pruning, our method prunes at the neuron level, allowing it to eliminate entire rows of weights instead of creating sparse matrices, making our approach the first to enable substantial speedups in the inference of BNNs on conventional hardware."

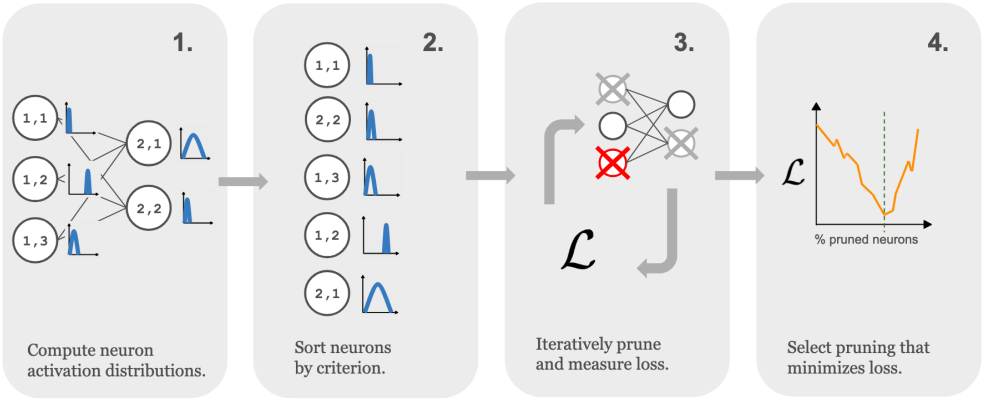


Figure 2: High-level overview of our neuron-level pruning pipeline described in Section 5.

2 Related Work

Compression techniques for neural networks usually fall into one of four strategies for reducing the model's memory footprint. Model *pruning* tries to reduce the number of used model parameters [3]. *Quantisation* reduces the floating-point precision of parameters [12] and can also be applied to BNN posterior distributions [13]. Thirdly, *knowledge distiliation* trains a smaller model to replicate knowledge learned by a bigger one [14]. Lastly, BNNs uniquely offer the option to promote sparsity before training via *prior choice* [15]. This work focuses on removing neurons and thus falls into the model pruning.

We can also differentiate between techniques that can prune on the level of individual weights and ones that prune entire neurons, filters or layers (structured pruning) [16]. The distinction is important because only structured pruning can achieve practical speedup without requiring special hardware or software. Pruning of individual weights results in sparse matrices while pruning of neurons corresponds to eliminating whole rows of weight matrices - shrinking its dimensions. The only other existing pruning techniques are unstructured, making our method the first to accelerate inference in practice. The drawback of structured pruning is that its reduced flexibility in selecting what to prune often limits the achievable pruning ratio compared to unstructured techniques [17]. We now briefly describe each of the competing BNN pruning methods.

2.1 SNR and SPR Criterions

The most common approach used to identify components of the networks that can be removed are heuristics such as SNR [18][19] or SPR [20] where central moments are computed with respect to the approximate posterior distribution $q(w)$ Both heuristics are then used as a ranking function where parameters with lower values are pruned first.

$$\text{SNR}(w) = \frac{|\mathbb{E}_{q(w)}[w]|}{\sqrt{\mathbb{V}_{q(w)}[w]}}, \tag{1}$$

$$\text{SPR}(w) = |\mathbb{E}_{q(w)}[w]| + \sqrt{\mathbb{V}_{q(w)}[w]}, \tag{2}$$

Both heuristics lack a clear stopping mechanism, so the user needs to tune the threshold manually for every network and choose which values of SNR or SPR are acceptable, making

the method impractical. Although our work also uses a criterion, we find the optimal prune ratio automatically based on an interpretable and steerable loss function. As defined in Section 5, our criterion also measures moments (mean and variance) but not over the parameter space as SNR and SPR, but over neuron activations/function space of the network.

This is important for the performance of the network, i.e., its behaviour within the considered data regime in function space. Pruning weights based solely on weight-space criteria overlook the impact on the network's performance. Additionally, note that for neural network architectures, the mapping between weight space and function space is generally intractable [source], making it challenging to reason about the network's function-space behavior from the weight space alone.

2.2 Variational Free Energy (VFE) Minimization

The VFE minimisation method [21], checks for each parameter whether pruning it worsens the VFE (ELBO) loss. It then makes this process efficient by leveraging Bayesian model reduction [22] to recompute the ELBO of the pruned model without executing costly forward passes with data. This approach also comes with a stopping criterion.

By ensuring the ELBO does not increase after pruning a parameter we have a guarantee that the approximate posterior distribution $q(w)$ remains unchanged throughout pruning. Since the pruning does not take into consideration the dataset $\mathcal{D}$, the same problems stemming from operating in weight space arise.

$q(w)$, approximates posterior $p(w|\mathcal{D})$ which is only one of two terms necessary for computing the predictive posterior distribution $p(y|x, \mathcal{D})$. That is, they are minimizing a measure between the unpruned and pruned distribution of the weights. Although we have a guarantee that $q(w)$ is maintained through pruning, changes in parameters w can affect the later term. Thus we don't have a guarantee that in the process of pruning, we preserve BNN's predictive distribution and therefore its performance. Our pruning loss function explicitly guarantees this.

BNN Pruning Method	SNR, SPR	VFE Minimization	Our Method
Pruning level	Weight level	Weight level	Neuron level
Pruning space	Weight space	Weight space	Function space
Speedup on conventional hardware	No	No	**Yes**
Stopping criterion	No	Yes	**Yes**
Formal performance guarantees	No	**Yes**	No
Steerable	No	No	**Yes**

Table 1: Comparison of pruning methods for Bayesian Neural Networks. Our neuron-level approach differs from prior work by operating in function space and enabling practical speedups.

3 Preliminaries

The following sections provide background on concepts necessary for understanding the pruning algorithm discussed later in this work. Additionally, we introduce the notation used for pruning and describe the different distributions within BNNs.

3.1 Bayesian Neural Networks (BNNs)

For an input vector $x \in \mathbb{R}^{n_0}$, we consider fully connected neural network $f^w : \mathbb{R}^{n_0} \to \mathbb{R}^{n_{K+1}}$ with of the following form for $l = 0, \ldots, L$ and $k = 0, \ldots, K$:

$$\begin{aligned} z_0 &= x, & \zeta_{k+1} &= W_k \left(z_k^T, 1\right)^T, \\ z_k &= \varphi_k(\zeta_k), & f^w(x) &= \zeta_{K+1}, \end{aligned}$$

where K is the number of hidden layers, n_k is the number of neurons in layer k, $\phi_k : \mathbb{R}^{n_k} \to \mathbb{R}^{n_k}$ is a vector of activations (one for each neuron) in layer k, and $W_k \in \mathbb{R}^{n_k \times n_{k+1}}$ is the

matrix of weights and biases that correspond to the kth layer of the network. We denote the vector of parameters by $f^w = (W_0^T, \ldots, W_K^T)^T$ and the mapping from ζ_k to ζ_{k+1} by $f^w_{k_1:k_2} : \mathbb{R}^{n_{k_1}} \to \mathbb{R}^{n_{k_2}}$. ζ_{K+1} is the final output of the network (or the logit in the case of classification).

BNNs differ by assuming a prior distribution $p(w)$ over the parameters w and a likelihood function $p(y|x, w)$. Consequently, the neuron activation z for a single input x is also a random variable, whose distribution is later used to compute the neuron importance. We denote the activation of ith neuron in layer k as $z_{k,i}(x)$.

The likelihood is generally assumed to be a Gaussian distribution in the case of regression and a categorical distribution for classification. Then, given a training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N_D}$, learning amounts to computing the posterior distribution $p(w|\mathcal{D})$ through Bayes' rule. The posterior predictive distribution over an input x^* is finally obtained by marginalizing the posterior over the likelihood.

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, w)\, p(w|\mathcal{D})\, dw. \tag{3}$$

Having a full predictive posterior distribution, rather than a just the point estimate, provides a better understanding of the model's uncertainty, which is crucial for robust decision-making. This allows us to account for uncertainty by either double-checking or disregarding the model's predictions in ambiguous cases—a consideration that is especially important in safety-critical applications [8]. For regression we then obtain network's final output (decision) by computing the mean of the predictive posterior distribution samples [7], i.e. $\hat{y}(x^*) = \mathbb{E}_{y\sim p(y|x^*,\mathcal{D})}[y]$.

However, due to the non-linearity introduced by the neural network architecture, the computation of the posterior distribution $p(w|\mathcal{D})$ and consequently of the posterior predictive distribution $p(y^*|x^*, \mathcal{D})$ is intractable. To address this challenge, we employ variational inference methods, which approximate the exact posterior distribution $p(w|\mathcal{D})$ with a surrogate posterior $q(w)$ by optimizing over a chosen family of candidate distributions.

In our work, we focus on two specific optimization approaches within the Gaussian variational inference framework: (1) Variational Online Gauss-Newton (VOGN) [23], a second-order online variational inference method that leverages gradient descent to update both the mean and variance of the weights, enabling scalability for large-scale Bayesian deep learning. (2) Stochastic Variational Inference (SVI), a first-order optimization technique that minimizes the evidence lower bound (ELBO) using stochastic gradient-based updates. In our implementation, we use the Adam optimizer. While our method is compatible with any approximate inference technique, including sample-based methods like Markov Chain Monte Carlo (MCMC), we demonstrate through these two approaches that the choice of inference method impacts the network's pruning potential.

To compare the network's behaviour with different parts of it being pruned we define a pruning mask $\alpha = \{0, 1\}^{|w|}$. For any BNN distribution p_x we denote the distribution under a pruning α as p_x^α. A fully pruned network where all neurons are set to zero would correspond to $p_x^{\bar{0}}$ and the original unpruned network would have $p_x^{\bar{1}}$.

3.2 Wasserstein Metric

The Wasserstein metric, also known as the Earth Mover's Distance (EMD), is a distance measure between two probability distributions. Formally rooted in optimal transport theory, the Wasserstein metric provides an intuitive way to quantify how much "work" is required to transform one distribution into another. Formally, the Wasserstein distance of order $\rho \geq 1$ between probability measures p and q in $\mathbb{R}^d$ is defined as

$$\mathbb{W}_\rho(p, q) := \inf_{\gamma\in\Gamma(p,q)} \left(\int_{\mathbb{R}^d\times\mathbb{R}^d} |x - y|^\rho \, \gamma(dx, dy) \right)^{1/\rho}, \tag{4}$$

where $\Gamma(\mu, \nu)$ denotes the set of all joint distributions γ on $\mathbb{R}^d \times \mathbb{R}^d$. Closeness in the $\rho-$Wasserstein distance implies closeness in the ρth moment. For example, using the 2-Wasserstein metric or $\mathbb{W}_2$ focuses on the first two moments of the distribution: the mean (expectation) and variance. To measure the activity of neurons Section 5 uses the fact in some specific cases, like when comparing some measure p with a Dirac at zero δ_0, we can simply compute $\mathbb{W}_2(p, \delta_0)$ from the first and second moments directly. The proof is provided below.

$$\begin{aligned}\mathbb{W}_2^2(p, \delta_0) &= \inf_{\gamma \sim \Gamma(p, \delta_0)} \mathbb{E}_{x,y\sim\gamma}\left[\|x - y\|^2\right] \\ &= \mathbb{E}_{x\sim\gamma}\left[\|x - 0\|^2\right] \\ &= \sum_i \mathbb{V}_{x\sim\gamma}[x_i] + \mathbb{E}_{x\sim\gamma}\left[x_i\right]^2 \end{aligned} \tag{5}$$

This characteristic is advantageous because it allows us to assess both the accuracy of the predictions (through closeness in the first moment) and the reliability of the uncertainty estimates (through closeness in the second moment). In contrast, while the KL divergence is widely used in variational inference, it is not a true metric and cannot, in general, be decomposed in terms of moments. The ability to express the Wasserstein distance in this special case directly via the first and second moments—particularly when comparing a distribution to a Dirac delta—is a key reason we chose it over the more commonly used KL divergence.

In Section 5 we will see that the Wasserstein metric plays a key role in formulating the pruning objective. There, we encounter the problem that $\mathbb{W}_2(p, q)$ is intractable when p and q are continuous distributions [24]. To get around this we can replace p and q with empirical distribution estimates $\tilde{p} = \sum_{i=1}^N \frac{1}{N}\delta_{c_i}$ and $\tilde{q} = \sum_{i=1}^N \frac{1}{N}\delta_{z_i}$, with $\{c_i\}_{i=1}^N$ and $\{z_i\}_{i=1}^N$ being samples from p and q. This makes $\mathbb{W}_2(\tilde{p}, \tilde{q})$ computable.

4 Problem Formulation

Assuming that our initial trained network is optimal and we want to find a subnetwork that maintains its performance we would want any pruning algorithm to account for the two following objectives.

Firstly minimize the size of the BNN, which corresponds to maximizing the number of pruned neurons. Secondly, we want to preserve the performance of the BNN on the dataset. This corresponds to preserving the shape of the predictive posterior distribution $p(y^*|x^*, \mathcal{D})$ and consequently of the decision $\hat{y}(x^*)$. That is, we aim to prune the parameters of the BNN so that the pruned BNN is close according to the 2-Wasserstein distance to the original BNN.

There is a clear tradeoff between the two objectives. We would expect that as we prune more neurons the predictive distribution will deviate from the original predictive distribution. Finding a sweet spot between the two objectives is the most general problem we're trying to solve with any pruning algorithm in the Bayesian setting. We formalise the problem below:

Problem 1. *Let $\mathcal{D} \subset \mathbb{R}^{n_0}$ be a finite set of input points. Then, for a BNN f^w with a joint predictive distribution $p_\mathcal{D}$ at the points in $\mathcal{D}$, and regularization weight $\lambda \in [0, \infty)$, find a mask $\alpha \in \{0, 1\}^{|w|}$ such that:*

$$\alpha^* = \underset{\alpha\in\{0,1\}^{|w|}}{\operatorname{argmin}} \frac{\mathbb{W}_2(p_\mathcal{D}^{\bar{1}}, p_\mathcal{D}^{\alpha})}{\mathbb{W}_2(p_\mathcal{D}^{\bar{1}}, p_\mathcal{D}^{\bar{0}})} + \frac{\lambda}{|w|}\sum_{i=1}^{|w|}\alpha^{(i)} \tag{6}$$

To ensure that the values of both objectives vary on the same scale and one term cannot dominate the other one we normalise both terms to make their values fall between 0 and 1. The normalisation term $\mathbb{W}_2(p_\mathcal{D}^{\bar{1}}, p_\mathcal{D}^{\bar{0}})$ is the Wasserstein-2 distance between an unpruned and completely pruned network which corresponds to an upper bound on $\mathbb{W}_2(p_\mathcal{D}^{\bar{1}}, p_\mathcal{D}^{\alpha})$.

The lambda parameter is an optional tunable hyperparameter that can give more priority to either objective. That is, $\lambda > 1$ means we are fine with greater drops in performance as long

as we prune more neurons. $\lambda < 1$ would make the pruning extra careful to not sacrifice any performance for a smaller network.

Finding an optimal solution faces two intractable problems. Firstly $\mathbb{W}_2(p^{\bar{1}}_{\mathcal{D}}, p^{\alpha}_{\mathcal{D}})$ is intractable. Secondly, it's infeasible to simply brute force the $2^{|w|}$ possible masks to find an optimal one. In the following section, we present a heuristic criterion derived from the loss function that as we show in our experiment section consistently finds good local optima.

5 Methodology

Approximating the Intractable Pruning Objective. A core part of the pruning objective defined in Equation 1 is measuring changes in output distributions, making it computationally infeasible to optimize directly. To address this challenge, we develop a tractable approximation in the form of a neuron-level criterion.

The term $\mathbb{W}_2(p^{\bar{1}}_{\mathcal{D}}, p^{\alpha}_{\mathcal{D}})$ is intended to minimize changes in the output distribution to preserve performance. We leverage the fact that the output predictive distribution directly depends on the distributions of individual neurons. Intuitively, if the changes measured by the Wasserstein distance at the neuron level are small, then $\mathbb{W}_2(p^{\bar{1}}_{\mathcal{D}}, p^{\alpha}_{\mathcal{D}})$ will also remain small. In other words, we want to identify neurons whose removal would minimally affect the network's output.

When a neuron is pruned, it always outputs 0, which corresponds to its output distribution being a Dirac delta distribution at zero, δ_0. Given this, we can measure how much a neuron's distribution *would* change if we were to prune it by computing the Wasserstein distance between its original activation distribution and the distribution it would have after pruning Formally, for input x, and denoting $p_{i,k,x}$ the activation distribution at neuron i in layer k, this is $\mathbb{W}(p_{i,k,x}, \delta_0)$. We prioritize pruning neurons that minimize this criterion first, as they are the ones whose pruning would least affect the shape of the predictive distribution. In practice, we consider the average activation distribution across all input points in a subset of the dataset $\mathcal{D}_{\text{prune}}$, which we set aside specifically for pruning, and is denoted $p_{i,k}$

The distribution $p^{\alpha}_{i,k}$ is influenced by the already pruned neurons, as indicated by mask α, since pruning earlier neurons alters the activations of later neurons.

Having a sufficiently large pruning dataset and combining activations across many input points is crucial because a neuron may be inactive for certain inputs while being active and potentially significant for others. Thus, aggregating activations across diverse input samples helps capture a more representative measure of a neuron's overall contribution to the network's output distribution.

When computing $\mathbb{W}(p_{i,k}, \delta_0)$, we again encounter the challenge that the Wasserstein distance is intractable for continuous distributions. We get around this by replacing the continuous distributions with their empirical counterparts, as described in Section 3.2, allowing us to compute $\mathbb{W}_2(p_{i,k}, \delta_0) \approx \mathbb{W}_2(\tilde{p}_{i,k}, \delta_0)$. Here, empirical distribution $\tilde{p}_{i,k}$ is defined by the union of samples of the activation at each data point, enabling practical computation.

To obtain the final form of the criterion for measuring neuron activity, we apply Equation 5 from earlier chapters, rewriting $\mathbb{W}_2(\tilde{p}^{\alpha}_{i,k}, \delta_0)$ as $\mathbb{V}_{\tilde{z}\sim\tilde{p}^{\alpha}_{i,k}}[\tilde{z}] + \mathbb{E}_{\tilde{z}\sim\tilde{p}^{\alpha}_{i,k}}[\tilde{z}]^2$.

Objective Optimization via Iterative Pruning. With the approximate objective defined, we can now outline how it is used to determine the optimal pruning strategy for the network. As outlined in the previous section, there are $2^{|w|}$ possible pruning configurations. The neuron-level criterion helps us reduce this search space to linear complexity by leveraging the intuition that pruning less active neurons first results in minimal changes to the output distribution. Thus, we prioritize pruning these neurons first.

In practice, our pruning algorithm begins by sorting all neurons in ascending order based on their precomputed criterion value. We then iteratively prune neurons, adding one additional neuron to the pruning mask at each step. (Note that the algorithm can be sped up by pruning multiple neurons at each step; however, this approach risks missing the optimal solution.) At

each iteration, we obtain a new pruning mask α, which we evaluate using the pruning cost function from Problem 1.

From the definition of the cost function, we expect the cost to be very high at the start (when no neurons are pruned) and at the end (when all neurons are pruned, maximizing the term penalizing deviations from the original predictive distribution). Somewhere in between, we find a pruning configuration that minimizes the cost, balancing the tradeoff between maintaining performance and maximizing compression, as parametrized by λ. This trend is visible in the pruning cost plotted over the iterations in Figure 8. We take the pruning mask corresponding to the lowest cost as the optimal solution.

Algorithm 1 Pruning Neurons Based on Neuron Activity Criterion

Ensure: Pruned network mask α^* that minimizes pruning loss
1: **Step 1: Find criterion for every neuron**
2: **for** each neuron $i = 1, 2, \ldots, |w|$ **do**
3: Compute criterion $\mathbb{W}_2(\tilde{Z}_i^{\alpha}, \tilde{\delta}_0)$
4: **end for**
5: **Step 2: Sort neurons by their criterion**
6: Sort neurons in ascending order based on $\mathbb{W}_2(\tilde{Z}_i^{\alpha}, \tilde{\delta}_0)$
7: **Step 3: Prune neurons iteratively and compute pruning loss**
8: **for** $i = 1, 2, \ldots, N$ **do**
9: Prune neurons 1 to i, creating mask α_i
10: Compute pruning loss $\frac{\mathbb{W}_2(p_{\mathcal{D}}^{\mathrm{I}}, p_{\mathcal{D}}^{\alpha})}{\mathbb{W}_2(p_{\mathcal{D}}^{\mathrm{I}}, p_{\mathcal{D}}^{0})} + \frac{\lambda}{|w|} \sum_{j=1}^{|w|} \alpha^{(j)}$
11: **end for**
12: **Step 4: Select optimal mask**
13: Choose mask α^* that minimizes pruning loss $\mathcal{L}_i$
14: **return** α^*

6 Experiments

The following section firstly benchmarks our pruning framework on the UCI regression datasets [1] and describes the associated challenges with trying to compare pruning techniques. Secondly, we use our technique to validate the lottery ticket hypothesis [2] in the Bayesian setting.

6.1 Regression with UCI Datasets

In Section 2, we reviewed the two existing techniques for pruning Bayesian Neural Networks (BNNs). However, a direct comparison between our approach and these methods is problematic for three reasons.

Firstly, as explained in Section 1, our method prunes parameters at the neuron level, unlike other methods that focus on pruning individual weights. In deterministic networks, pruning individual weights provides greater flexibility, which consistently allows weight-level pruning methods to outperform neuron-level approaches [17]. For Bayesian networks, we also found that our method could not achieve the same pruning ratios as [21]. However, neuron-level pruning compensates for this by reducing the dimensions of the weight matrices. Unlike the sparse matrices created by weight-level pruning, this reduction enables practical usability.

Secondly, as argued in Section 4, any pruning algorithm must balance the tradeoff between achieving greater compression and maintaining performance. Thus, even though other works present achieved pruning ratios, it is often unclear at what cost to performance these were obtained. Our method addresses this by recognizing that prioritizing either compression or performance maintenance depends on the use case of the network to be pruned. To this end, we introduce the hyperparameter λ. We demonstrate the steerability of our pruning approach in Section 6.1.4.

[1] https://archive.ics.uci.edu/datasets

Lastly, our method, specifically (though it is unclear if this applies to other methods), appears to be sensitive to the choice of hyperparameters. This includes obvious factors like network size—where more overparameterized networks allow for greater pruning—but also extends to hyperparameters such as the activation function and optimizer.

In light of these considerations, rather than attempting a direct comparison with other methods—which are designed for different use cases and do not result in practical speedups—we adopt a more comprehensive approach. Specifically, we benchmark our method by presenting statistics on the achieved pruning ratios with respect to different hyperparameter settings.

6.1.1 Network Size

First, we examine the hypothesis that larger networks make individual neurons less critical, allowing for higher pruning ratios. In Figure 3, we illustrate the achieved prune ratios across varying network configurations, focusing on the number of layers (depth) and neurons per layer (width). Notably, the right-hand figure highlights a consistent trend across all datasets: in single-layer networks, as the width increases, more neurons become less active. This enables higher pruning ratios to be achieved without any loss in performance.

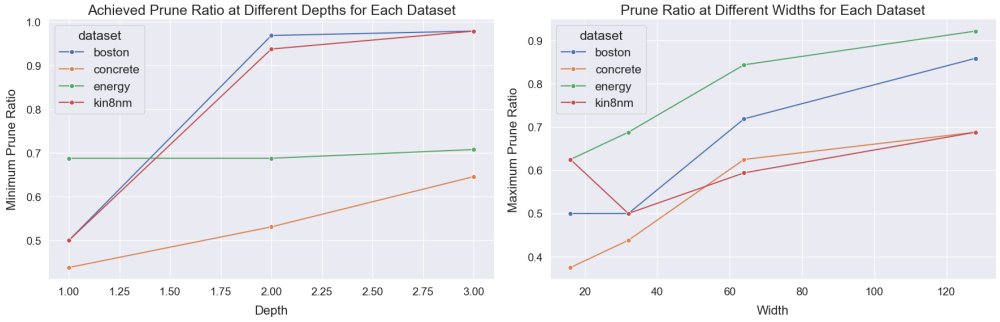


Figure 3: Maximum prune ratio achieved for increasing depth/width. For the depth graph, a fixed width of 32 neurons was used. For the width graph, one layer was used.

6.1.2 Activation Function

The activation function plays a significant role in determining the effectiveness of our pruning technique. With ReLU, many neurons exhibit the inactivity pattern shown in Figure 1, characterized by a single peak around the origin. When these neurons are masked, their output remains at 0, making pruning straightforward. In contrast, Tanh produces activation distributions with two peaks, as seen in Figure 4. In practice, using Tanh consistently leads to worse pruning results, regardless of the configuration of other parameters, as illustrated in Figure 5.

6.1.3 Optimizer

Similarly to activations, we plotted the achieved prune ratios for two BNN optimizers, VOGN and SVI, to see if either results in networks with more pruning potential. In Figure 6 we see that through all hyperparameters configuration, SVI tends to outperform VOGN. The reason for this is not entirely clear, but it is likely that the second-order nature of VOGN results in less sparse networks, meaning that the weight posteriors remain less deterministic and retain more uncertainty. This increased uncertainty makes pruning more challenging, as removing weights is more likely to significantly alter the posterior predictive distribution.

6.1.4 Lambda Hyperparamter

The final hyperparameter we investigate is λ, which is specific to our technique and determines how cautious the model should be in preserving performance while pruning more neurons. In Figure 7, we observe that the achieved prune ratio increases across all datasets as λ is increased. This occurs because the pruning cost function, described in Section 4, increasingly

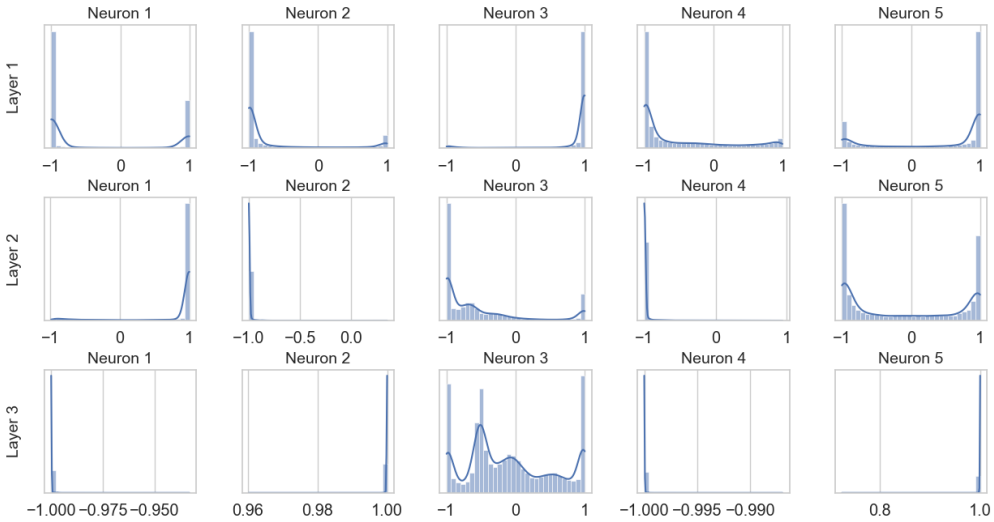


Figure 4: Empirical activation distributions for each neuron, as defined in Section 5. Each distribution represents activations over N data points for M samples. The BNN has 3 layers with a hidden dimension of 5, using the *Tanh* activation function.

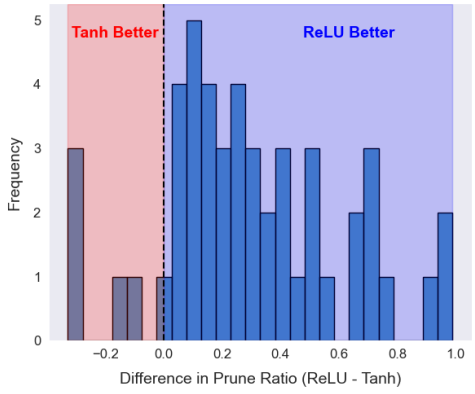


Figure 5: Differences in Prune Ratios with ReLU and Tanh (With All Other Hyperparameters Held Equal).

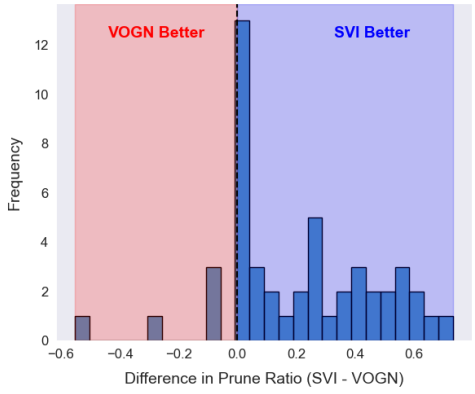


Figure 6: Differences in Prune Ratios with SVI and VOGN optimizers (With All Other Hyperparameters Held Equal).

rewards higher prune ratios. However, the right subfigure shows that for larger λ values, the performance—measured by RMSE for regression datasets—declines more significantly. This indicates that while higher pruning can be achieved, it comes at the expense of performance, demonstrating that our method is steerable.

As practical guidance, we found that values of λ between 0.5 and 1.5 typically offer a good trade-off between compression and performance. If reducing network size is the primary goal and some performance drop is acceptable, higher values (e.g., $\lambda > 1.5$) are appropriate. Conversely, when preserving predictive performance is critical, setting λ below 0.5 is advisable.

6.2 Classification with Fashion MNIST

One limitation of UCI datasets is their small number of features, which necessitates smaller networks and makes overparameterization more likely. To test our model on a more complex dataset and demonstrate that our method can also prune classification models, we applied our pruning method to the Fashion MNIST dataset []. This dataset contains thousands of 28 by 28

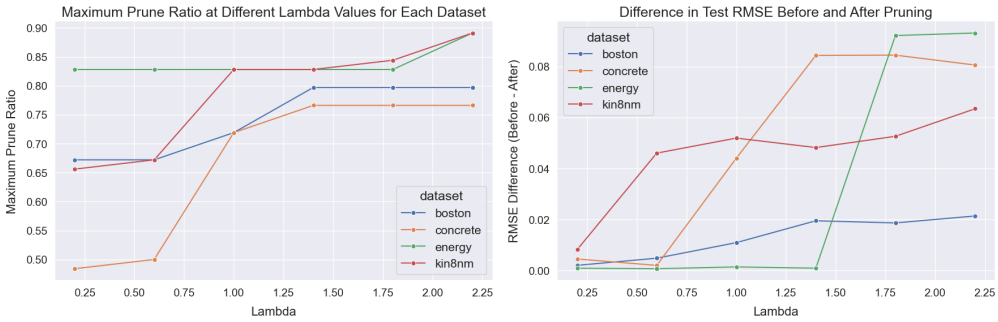


Figure 7: On the left, prune ratio achieved with different λ values. On the right, performance drops in RMSE before and after pruning for different λ values

grayscale images of clothing items across 10 classes. When flattened, these images yield 784 input features, requiring a more complex network compared to UCI datasets, which typically have around 10 features.

We trained a 2-layer BNN with a hidden dimension of 128. Based on results from the previous section, we used the ReLU activation function and SVI optimizer, as these yielded the best performance. Before initiating pruning, we ensured the network was adequately trained, achieving 87% accuracy. Since we cannot directly use outputs for comparing shapes of predictive distributions in classification due to the softmax layer, our pruning cost function operates on the network logits instead.

Figure 8 provides detailed insights into the progression of our pruning algorithm when run with $\lambda = 0.3$. The top left plot shows the pruning objective we aimed to minimize, with the minimum occurring at approximately 0.8. The top right plot decomposes the cost further into two components: the shape cost (deviation from initial output predictive distribution) and the prune ratio cost (proportion of neurons pruned).

Our experiments revealed two key findings. First, our pruning cost function demonstrates convex behavior in practice, yielding a single global cost minimum. This behavior likely occurs because the algorithm correctly prunes neurons in order of their importance until reaching a smaller subset of core neurons. When these core neurons are pruned, the predictive distribution collapses, manifesting as a sharp increase in the shape cost component.

Second, as shown in the bottom right plot, since we chose a low value of λ, the algorithm successfully identified the maximum level of pruning that maintained the original model's accuracy, without explicitly considering accuracy during the pruning process. These results demonstrate that our method can effectively prune a significant number of neurons and be steerable even in more complex networks and classification problems, making it viable in practice.

6.3 Validating Lottery Ticket Hypothesis In Bayesian Setting

As a second experimental case study, we chose to validate the Lottery Ticket Hypothesis. For DNNs, it suggests that large overparameterized networks often achieve better performance but contain "winning tickets"—small, sparse subnetworks capable of achieving the same performance. Furthermore, these winning tickets not only match the performance of the original network but, when trained from scratch with the same weight initialization, can learn faster than the full network.

In Section 6.1.1, we already demonstrated that larger networks can be pruned more extensively, confirming the existence of sparse subnetworks with equivalent performance. To validate the second part of the hypothesis—that these subnetworks learn faster—we first trained a network and used our pruning technique to identify the winning ticket. We then trained the subnetwork from scratch (training a subnetwork corresponds to training it with its pruning mask, α). In Figure 9, we observe that when the subnetwork is trained with the same weight initialization

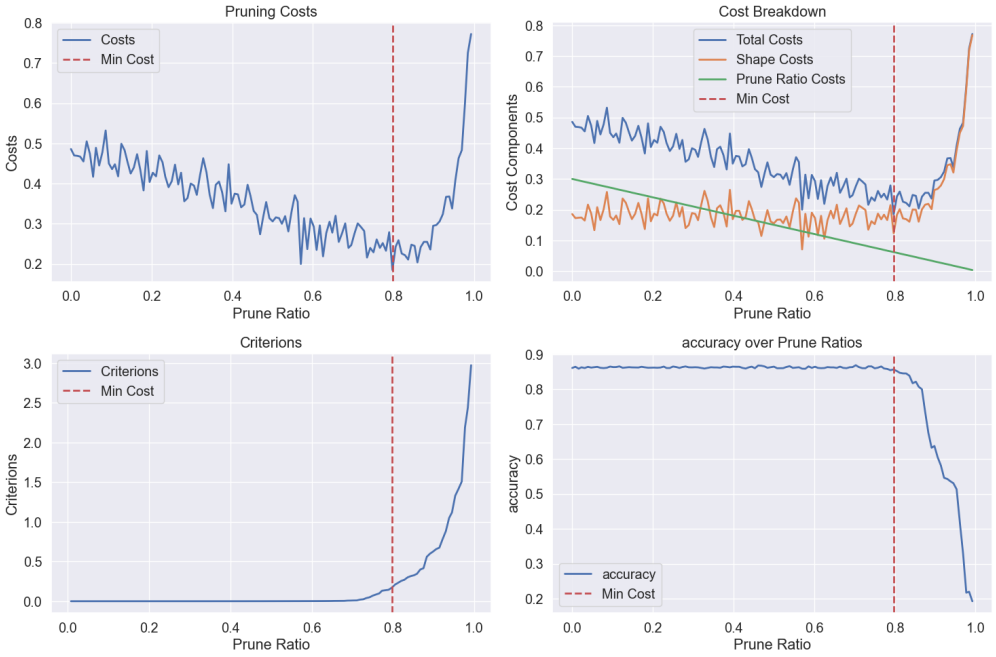
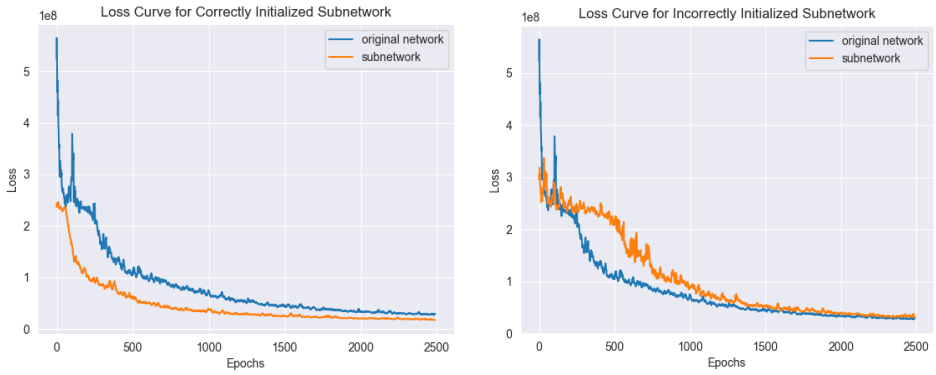


Figure 8: Fashion MNIST neural network pruning experiments. **Top left**: Pruning costs versus prune ratio, showing total cost increasing sharply after 80% optimal pruning. **Top right**: Decomposition into shape costs (distribution change) and prune ratio costs (sparsity), illustrating the trade-off between model compression and performance. **Bottom left**: Heuristic criterion value for the last pruned neuron, demonstrating increased neuron importance at higher compression. **Bottom right**: Test accuracy remains stable until critical pruning threshold at 80%, followed by rapid performance degradation.

as the original network, the loss curve converges faster. However, with a different random initialization, the training not only slows down but also achieves worse RMSE than both the original large networks and the correctly initialized subnetwork.



(a) (b)

Figure 9: Comparison of a subnetwork trained from scratch (**Left**) with weight initialization from the original network vs different random initialization (**Right**).

7 Discussion, Limitations & Conclusion

We formalized a general pruning objective in the Bayesian setting, conceptualizing it as a trade-off between maintaining network performance and maximizing neuron pruning. Specifically, we created a steerable cost function that balances preserving the shape of the predictive distribution with the goal of reducing network complexity. Furthermore, we introduced an approximate solution to the pruning objective through a novel neuron-level criterion that measures neuronal activity. We developed an accompanying algorithm for iteratively finding an optimal pruning solution.

Through comprehensive experiments, we demonstrated the versatility of our approach, successfully pruning the majority of neurons across both classification and regression models. This broad applicability underscores the method's robustness and potential for widespread implementation. Our neuron-level pruning approach, unlike previous methods that produce only sparse prunings, offers significant practical advantages. By enabling more aggressive neuron reduction, our method can lead to substantial speedups on current hardware. Consequently, we can simultaneously decrease the memory footprint, making them more viable for safety-critical edge devices, and accelerate inference times—a critical step towards making BNNs more scalable and mainstream.

Future directions. In Section 6.1, we observed that the proportion of pruned neurons is influenced by hyperparameter choices, particularly the optimizer and activation function, which somewhat limits our method's generality. The case of the VOGN optimizer is especially significant given its popularity. It would be important to investigate whether this discrepancy stems from VOGN creating less sparse networks and to explore ways to enhance pruning effectiveness for VOGN-trained networks.

A current limitation of our pruning algorithm is the need to recompute the pruning cost, requiring multiple forward passes after each pruned neuron. However, since we empirically observed that the pruning cost function of a well-trained network exhibits convex behavior, we could potentially find the global minimum more efficiently using techniques such as Bayesian optimization.

While our pruning objective is general, our pruning algorithm represents just one approximate solution among many possible approaches. Future work could explore alternative criteria, potentially designed to work more effectively with different activation functions and optimizers, and compare their relative performances.

Finally, making BNNs truly scalable requires addressing both training and inference speed, whereas our method currently focuses solely on inference. An intriguing direction for future research would be to investigate why well-trained networks consistently develop numerous inactive neurons, as we empirically observed. This understanding could potentially be leveraged to accelerate training, perhaps through implementing progressive pruning during earlier training phases.

References

[1] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[2] Jonathan Frankle and Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2019. arXiv: 1803.03635 [cs.LG]. URL: https://arxiv.org/abs/1803.03635.

[3] S. Hanson and L. Pratt. "Comparing biases for minimal network construction with back-propagation". In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 1. 1988.

[4] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV]. URL: https://arxiv.org/abs/1510.00149.

[5] Yang He et al. *Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks*. 2018. arXiv: 1808.06866 [cs.CV]. URL: https://arxiv.org/abs/1808.06866.

[6] Yann LeCun, John Denker, and Sara Solla. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[7] Radford M. Neal. *Bayesian Learning for Neural Networks*. Vol. 118. Springer Science & Business Media, 2012.

[8] Theodore Papamarkou et al. *Position: Bayesian Deep Learning is Needed in the Age of Large-Scale AI*. 2024. arXiv: 2402.00809 [cs.LG]. URL: https://arxiv.org/abs/2402.00809.

[9] Minjung Lee, Jinsoo Bae, and Seoung Bum Kim. "Uncertainty-aware soft sensor using Bayesian recurrent neural networks". In: *Advanced Engineering Informatics* 50 (2021), p. 101434. ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2021.101434. URL: https://www.sciencedirect.com/science/article/pii/S1474034621001865.

[10] Fabio Arnez et al. "Towards Dependable Autonomous Systems Based on Bayesian Deep Learning Components". In: (Sept. 2022), pp. 65–72. ISSN: 2641-810X. DOI: 10.1109/EDCC57035.2022.00021. URL: https://doi.ieeecomputersociety.org/10.1109/EDCC57035.2022.00021.

[11] Ke Zhang et al. "Compacting Deep Neural Networks for Internet of Things: Methods and Applications". In: *IEEE Internet of Things Journal* 8.15 (2021), pp. 11935–11959. DOI: 10.1109/JIOT.2021.3063497.

[12] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV]. URL: https://arxiv.org/abs/1510.00149.

[13] Yibo Yang, Robert Bamler, and Stephan Mandt. "Variational bayesian quantization". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10670–10680.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML]. URL: https://arxiv.org/abs/1503.02531.

[15] Christos Louizos, Karen Ullrich, and Max Welling. "Bayesian compression for deep learning". In: *Advances in neural information processing systems* 30 (2017).

[16] Zehao Huang and Naiyan Wang. *Data-Driven Sparse Structure Selection for Deep Neural Networks*. 2018. arXiv: 1707.01213 [cs.CV]. URL: https://arxiv.org/abs/1707.01213.

[17] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. *A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations*. 2024. arXiv: 2308.06767 [cs.LG]. URL: https://arxiv.org/abs/2308.06767.

[18] Alex Graves. "Practical variational inference for neural networks". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, pp. 2348–2356. ISBN: 9781618395993.

[19] Charles Blundell et al. *Weight Uncertainty in Neural Networks*. 2015. arXiv: 1505.05424 [stat.ML]. URL: https://arxiv.org/abs/1505.05424.

[20] Eric Nalisnick. *On Priors for Bayesian Neural Networks*. 2018.

[21] Jim Beckers et al. "Principled Pruning of Bayesian Neural Networks through Variational Free Energy Minimization". In: *IEEE Open Journal of Signal Processing* (2023), pp. 1–9. ISSN: 2644-1322. DOI: 10.1109/ojsp.2023.3337718. URL: http://dx.doi.org/10.1109/OJSP.2023.3337718.

[22] Karl Friston, Thomas Parr, and Peter Zeidman. *Bayesian model reduction*. 2019. arXiv: 1805.07092 [stat.ME]. URL: https://arxiv.org/abs/1805.07092.

[23] Kazuki Osawa et al. *Practical Deep Learning with Bayesian Principles*. 2019. arXiv: 1906.02506 [stat.ML]. URL: https://arxiv.org/abs/1906.02506.

[24] Cédric Villani et al. *Optimal transport: old and new*. Vol. 338. Springer, 2009.