Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

Efficient Deep Ensembles by Averaging Neural Networks in Parameter Space

Philip Norris Mitchell

Supervised by Dr. Antonio Agudo and Dr. Adrià Ruiz

September, 2021

I would like to thank my two supervisors Dr. Antonio Agudo and Dr. Adrià Ruiz for their guidance and enthusiasm. They were generous with their time and always open to hold stimulating discussions and offer invaluable feedback.

Lastly, I would like to thank Dr. Juanjo Rué, the coordinator of the MAMME program, for his constant support throughout the entire Master's program. I cannot recall of an email which he did not respond to in < 24 hours.

Abstract

Although deep ensembles provide large accuracy boosts relative to individual models, their use is not widespread in environments in which computational constraints are limited, as deep ensembles require storing M models and require M forward passes at prediction time. One established method of overcoming these computational challenges, while mildly preserving accuracy, is known as knowledge distillation. We propose a novel computationally equivalent method, which we name *permAVG*. In the *permAVG* method we directly average in parameter space, such that a M model ensemble can be reduced to a single model simply by averaging over the weights. However, due to the fact that each model lies in its own distinct local optimum, naively averaging over the weights leads to poor results, as the model average most likely does not live in a valley of the loss landscape. For this reason, we propose to exploit the symmetries of the loss landscape by learning permutations, such that all M models can be permuted into the same local optimum and can thereafter safely be averaged. We find the *permAVG* ensemble of two models to significantly increase accuracy relative to individual models and to often match and occasionally outperform both knowledge distillation and deep ensembles of small sizes.

Keywords

Ensemble learning, Deep ensembles, Knowledge distillation, Permutation learning.

1. Introduction

In contrast to many classical machine-learning algorithms, neural networks are typically ensembled by simply averaging over the predictions of M models which are independently trained on the same dataset. The training process of the individual models varies only in their random initialization and/or random shuffling of the training data and each model is therefore able to find a distinct local optimum [14]. Such ensembles of neural networks are often referred to as *deep ensembles*. Despite this simplistic approach, deep ensembles work remarkably well and are generally able to outperform other ensembling approaches in terms of accuracy. For example, [1] finds that an ensemble of ten networks on the CIFAR10 [26] dataset can *easily* improve test accuracy by 5%.

However, despite the considerable accuracy gains, deep ensembles face both storage and computational challenges that prevent them from widespread use [21]. A deep-ensemble requires storing M models and given that several architectures consist of millions of parameters, even a linear scaling of storage requirements can be prohibitive. Furthermore, a deep ensemble of M models requires M forward passes at prediction time, which in the case of many real-time systems may be too costly.

In order to overcome these computational constraints, several novel ensembling techniques have been developed, the most notable of which is known as knowledge distillation [20]. The key concept behind knowledge distillation is the belief that the logits –soft labels– of a trained model contain more valuable information than the hard labels of the training data, as the magnitudes of the inter-class differences reveal information regarding class similarities. Therefore, it may be more effective to train a smaller *student* model to match the distribution of a much larger *teacher* model, than to train a model solely on hard training labels. In the case of deep ensembles, a single model can be trained to match the average logits of all M models and therein an ensemble can be distilled into a single model. However, although knowledge distillation has been shown to result in significant accuracy increases relative to individual models [20], it also has been shown to lead to inconsistent and unintuitive results. For example, [8] finds that more accurate teacher models can lead to less accurate student models and that knowledge distillation fails on certain datasets such as IMAGENET [27].

In order to overcome the computational constraints of deep ensembles and to provide a more robust alternative to knowledge distillation, we propose a novel ensemble method. As opposed to deep ensembles, in which models are averaged in function space, we propose averaging in parameter space. In this manner we can easily reduce an ensemble of M models into a single model, simply by averaging their parameters. However, although linear models such as linear regression can simply be averaged in parameter space [11] due to the convexity of their objective function, non-linear models such as neural networks cannot due to the fact that each model finds a unique local optimum. Hence, whereas linear models may benefit from parameter averaging as each model is an estimation of the same globally optimal parameters, averaging neural networks can be a detriment as the average of two distant optima may not lie in a valley of the loss landscape.

However, the fact that two networks find different local optima does not imply that the networks are functionally different. In fact, one network could merely be a permutation of the other. By permuting the rows of a $m \times n$ weight matrix W_i and the columns of the subsequent weight matrix W_{i+1}, one can create $m!$ different networks which are equivalent in function space, but not in parameter space [2]. We hypothesize that the M models in a deep ensemble are therefore all estimations of the same underlying function and that they vary mostly in the specific permutation of their weights. Hence, by learning the correct permutation, we can translate each network to the same local optimum in parameter space. Thereafter, we can simply average their parameters as they are all estimates of the same true local optimum.

Following this reasoning, we propose a novel end-to-end neural network framework which we name permAVG (permutation averaging). The permAVG method takes M models as input and learns the optimal weight permutations of all M models such that their average is optimal. We propose permAVG as a computationally equivalent but perhaps more accurate and stable alternative to knowledge distillation. We demonstrate the permAVG approach on both the CIFAR10 [26] and FMNIST [44] datasets. We find that for ensembles of two models, permAVG can significantly increase accuracy relative to individual models and can often match and occasionally outperform both deep ensembles and knowledge distillation. This in itself is sufficient to make permAVG a viable ensemble technique, when both storage and computational resources are limited.

1.1 Outline

The format of the paper is as follows. In section 2 we first review the motivations behind the most established traditional ensembling methods such as bagging, boosting and Bayesian model averaging. In section 3, we explore various ensembling methods that are applied to neural networks. In addition to deep ensembles, we explore other methods such as knowledge distillation and Bayesian approximations. Thereafter, in section 4 we review the most prominent theories regarding the effectiveness of deep ensembles. Next, in section 5.1 we formally propose our novel approach. As the permAVG approach will require learning permutation matrices, we review how permutation matrices can be learned in the context of neural networks in section 5.2. Thereafter, in section 5.3, we compare the permAVG approach to the exisiting neural network ensemble methods discussed in section 3. Finally, in section 6 we discuss the implementation details of the permAVG method and in section 7 we present the experimental results. Lastly, we review all of our findings and suggest directions of possible future research in section 8.

2. Ensemble theory

We begin with the theoretical background on ensemble methods and explore variance reduction and uncorrelated errors between models as two justifications for the effectiveness of ensembles. Afterwards we explore three common ensemble methods known as bagging, boosting, and Bayesian model averaging. The majority of the formulas and content are derived from [2].

2.1 Notation for section 2

We follow the following notation conventions throughout section 2:

$P = p(\mathbf{x}, y) =$ the underlying probability distribution of the data
$\mathcal{D} =$ the set of all possible datasets
$D =$ A single dataset where $D \in \mathcal{D}$
$(\mathbf{x}, y) =$ a single data point with input $\mathbf{x}$ and output y, where $(\mathbf{x}, y) \in D$
$N =$ the number of points in a given dataset, $|D|$
$M =$ the number of models

$f(x, D) = f(x)$ = A learned model (or hypothesis) mapping inputs **x** to outputs y
$h(x)$ = The true regression function which minimizes expected square loss

2.2 Ensembling as a means of variance reduction

From a frequentist perspective, ensemble methods are often motivated by the bias-variance decomposition from statistical decision theory in which the reducible error of the expected square loss can be decomposed into bias and variance components [2]:

$$\mathbb{E}_D[(h(x) - f(x, D))^2] = (\mathbb{E}_D[f(x, D)] - h(x))^2 + \mathbb{E}_D[(f(x, D) - \mathbb{E}_D[f(x, D)])^2]. \tag{1}$$

The first component in Eq. (1) is the square bias, which represents the deviation of the average prediction of the learning procedure over all possible datasets from that of the true regression function. The second component is the variance, which represents the average deviation of any individual model from the average prediction of the learning procedure over all possible datasets. In other words, the reducible error consists of a combination of how *wrong* a learning procedure is on average (bias) and to what extent the learning procedure is sensitive to specific datasets (variance). Often, as the commonly used term "bias-variance tradeoff" suggests, decreasing one component often leads to an increase in the other and hence a critical aspect of model selection lies in finding the optimal balance between bias and variance. For example, models exhibiting low training error but high test error, a phenomenon often referred to as overfitting, are often constrained with regularization, which reduces variance but can increase bias [41]. However, whereas model selection is predominantly concerned with adjusting model flexibility to achieve an optimal tradeoff, ensemble methods such as bagging keep model flexibility constant, and therefore also the bias, while reducing the variance by averaging over multiple models built on different datasets.

The fact that averaging over multiple datasets can reduce variance is clear in the extreme case when the model is an average over all possible datasets ($f_A(x, D) = \mathbb{E}_D[f(x, D)]$). In this case the variance component in Eq. (1) is minimized and vanishes completely. This extreme case suggests that even averaging over a few datasets may also reduce the variance and therefore the expected error [2, 18]. This intuition can also be shown formally. We follow the derivation of [2].

Consider the following expression for the m-th model $f(x)_m$, where $\epsilon_m(x)$ represents the deviation of $f(x)_m$ from the true regression function $h(x)$ as:

$$f(x)_m = h(x) + \epsilon_m(x). \tag{2}$$

Its expected square error is given by:

$$\mathbb{E}_x[(f(x)_m - h(x))^2] = \mathbb{E}_x[\epsilon_m(x)^2]. \tag{3}$$

Now let us define a single model $f(x)_{ENS}$, which is an ensemble over all models $f(x)_m$, in which the models are ensembled simply by averaging over the outputs of each model:

$$f(x)_{ENS} = \frac{1}{M} \sum_{m=1}^{M} f(x)_m \tag{4}$$

We now define the expected square error for both $f(x)_{ENS}$ and the average expected square error of all M models individually. Let these be E_{ENS} and E_{AV}, respectively:

$$E_{ENS} = \mathbb{E}_x[(\sum_m^M \frac{1}{M}\epsilon_m(x))^2], \tag{5}$$

$$E_{AV} = \mathbb{E}_x[\sum_m^M \frac{1}{M}\epsilon_m(x)^2]. \tag{6}$$

We can then prove the following inequality:

$$E_{ENS} \leq E_{AV}. \tag{7}$$

The inequality implies that in the expectation, the ensemble of M models always outperforms or matches the average performance of the M models. Note that the inequality and its derivation do not explicitly show through which mechanism ensembles reduce expected square error (e.g., through variance reduction), but only show that on average the claim is true. We will now walk through the derivation.

Consider Jensen's inequality for the convex function $\phi(x) = x^2$:

$$(\sum_{i=1}^M \lambda_i x_i)^2 \leq \sum_{i=1}^M \lambda_i x_i^2. \tag{8}$$

Ignoring the $\mathbb{E}$ in Eq. (5) for both E_{AV} and E_{ENS} and identifying $\lambda_i = \frac{1}{M}$ and $x_i = \epsilon_m(x)$ we find the inequality:

$$(\sum_m^M \frac{1}{M}\epsilon_m(x))^2 \leq \sum_m^M \frac{1}{M}\epsilon_m(x)^2. \tag{9}$$

As Eq. (9) holds for all x, it must also hold for the expectation with respect to x. Hence Eq. (7) is confirmed.

2.3 Ensembling and uncorrelated errors between models

So far we have primarily motivated ensemble methods by reducing the variance component in the bias-variance decomposition. However, it is also insightful to understand ensembles from an uncorrelated error perspective.

Firstly, it is important to note that the bias-variance decomposition as defined in Eq. (1) applies only to the square error loss function. It does not generalize to all other loss functions, for example the zero-one loss which is often used in classificiation [18]. Whereas averaging over models built on different datasets is guaranteed to improve or match performance relative to individual models in the MSE loss setting (see Eq. (7)), averaging over models in the zero-one loss setting can provably impair performance. The following example derived from [18] illustrates this point. Consider M independent classification models $(m_1, m_2, \ldots, m_M)$ in which all models have an error rate $e = 0.51$. In other words, all models are slightly

worse than random. If we ensemble the M models through majority vote, we attain the ensemble model m_{ENS}:

$$m_{ENS} = \sum_{i=1}^{M} I(m_i(x) = 1), \tag{10}$$

where m_{ENS} follows a binomial distribution $B(M, 1-e)$ and therefore the $P(m_{ENS} > \frac{M}{2})$ approaches 0 as M approaches ∞. Of course, in the case that $e < 0.5$ the result is reversed and probability of correct classification approaches 1. Following this result, it is theoretically possible to create a strong classifier as an ensemble of *weak learners*, where a *weak learner* is a model that is slightly better than random [36].

Secondly, it should be noted that the bias-variance decomposition applies only in the case of a single model and that a generalization of the bias-variance decomposition entails a correlation component. Thus far, we have treated an ensemble model as a single model, whose prediction is composed of the average over the predictions of M different models. However, an ensemble model can equally be represented in a multidimensional setting, in which we are trying to reduce the expected square error across M models. Following this representation, the expected error has a bias-variance-covariance decomposition [6, 42]:

$$\mathbb{E}[(\frac{1}{M}f(x, D_i)_i) - h(x)] = \left\{ \frac{1}{M}\sum_{i=1}^{M}(\mathbb{E}_{D_i}[f(x, D_i)_i - h(x)]) \right\}^2 + \frac{1}{M}\left\{ \frac{1}{M}\sum_{i=1}^{M}(f(x, D_i)_i - \mathbb{E}_{D_i}[f(x, D_i)_i])^2 \right\}$$
$$+ (1 - \frac{1}{M})\left\{ \sum_{i}^{M}\sum_{j\neq i}^{M}(f(x, D_i)_i - \mathbb{E}_{D_i}[f(x, D_i)_i])(f(x, D_j)_j - \mathbb{E}_{D_j}[f(x, D_j)_j]) \right\}, \tag{11}$$

Note that the datasets D_i are each sampled with respect to their own probability distribution and hence the expectation operator $\mathbb{E}_{D_i}$ can vary for each model. Additionally the expectation operator $\mathbb{E}$ in Eq. (11) with omitted subscript, implicitly takes the expectation with respect to the random variable from which D_i was sampled. Additionally, note that each model need not be the same and hence $\mathbb{E}_{D_i}[f(x, D_i)_i]$ must not equal $\mathbb{E}_{D_i}[f(x, D_i)_j]$.

The expressions within the curly braces represent the average **bias**, **variance** and **covariance** across all M models, respectively. Considering a two-dimensional example ($M = 2$), we see in Eq. (11) that if both models are negatively correlated, that the expected ensemble error decreases. Hence by promoting diversity across models, one can reduce expected square error [5].

We can support this finding through a comparison of E_{ENS} and E_{AV} from before. Recall the expected error functions of both in Eq. (5). Assuming that errors have zero mean ($\mathbb{E}[\epsilon_m(x)] = 0$) and zero covariance between different models ($\mathbb{E}[\epsilon_m(x)\epsilon_l(x)] = 0$ for $m \neq l$) we find [2]:

$$E_{ENS} = \mathbb{E}_x[(\sum_{m}^{M}\frac{1}{M}\epsilon_m(x))^2] = \mathbb{E}_x[(\frac{1}{M^2}\sum_{m}^{M}\sum_{l}^{M}\epsilon_m(x)\epsilon_l(x)] = \frac{1}{M^2}\sum_{m}^{M}\mathbb{E}_x[\epsilon_m(x)^2] = \frac{1}{M}E_{AV}, \tag{12}$$

Hence, assuming uncorrelated errors between models, we can reduce the average model accuracy of M models by a factor of $\frac{1}{M}$ simply by ensembling the M models. Note that this result only holds in the MSE loss setting.

2.4 Summary

To summarize, the core results from the previous subsections are: 1) the fact that an ensemble of M models outperforms (or matches) the average of M models in terms of accuracy and that this performance boost is believed to be the result of a reduction in variance. Note that this result only holds in the expectation over all datasets and only in the MSE loss setting. 2) uncorrelated models in both the MSE and zero-one loss setting lead to improved accuracy.

2.5 Theoretical limitations

Although the theoretical benefits of ensembling are impressive, there are a few practical considerations which diminish their theoretical effectiveness. Firstly, in practice, errors made by separate models are rarely uncorrelated and models typically make similar errors on inputs which are difficult to classify (or for which regression prediction is difficult) [2]. Secondly, all theoretical conclusions are derived assuming the availability of multiple datasets from the true probability distribution P. However, in reality, there is only one available dataset.

Nevertheless, despite these shortcomings, multiple different ensemble methods have been devised and they have shown considerable success empirically and in practice.

2.6 Bagging

One technique to overcome the lack of multiple datasets is known as bootstrapping [13], in which several artificial datasets are generated by uniformly sampling from the only available dataset with replacement. Then, assuming we have generated M datasets, we can train M individual models each on a separate dataset. Finally, by averaging over the output of each of the M models (or through majority vote for classification models) we can create a single ensemble model. This ensemble method is known as bagging [4]. According to Breiman [4], bagging primarily helps *instable* learning procedures such as decision trees, in which a small deviation in the dataset leads to large deviations in model predictions. Following this definition of instability, one can interpret *instable* learning procedures as those whose error stems primarily from the variance component in Eq. (1). An example of a *stable* learning procedure for which bagging can impair performance is k-nearest-neighbours. Lastly, it must be noted that bootstrapped datasets are sampled from a probability distribution $P^{Bootstrap}$, which is only a discrete approximation of the true data-generating distribution P and therefore the true distribution P is usually *smoother and more spread out* than $P^{Bootstrap}$ [3]. For this reason, bagging is a trade-off between accuracy gains as a result of reduction in variance and accuracy reduction as a result of increased bias due to the fact that $P^{bootstrap}$ is only an approximation of P.

2.7 Boosting

In addition to bagging, boosting [15] is another similar but slighty more complex ensemble method. There are several boosting algorithm variants and here we only describe *Adaboost*. There are three main differences between boosting and bagging. Firstly, in boosting, models are built in sequence and in each iteration the current bootstrap sampling distribution is dependent on the previous iteration. In the first iteration, bagging and boosting are identical and both build a model based on a bootstrapped dataset in which data points have been sampled uniformly from the original dataset. However, whereas bagging repeats the

same procedure indefinitely, boosting adjusts the bootstrap sampling distribution by weighting misclassified points more heavily. Secondly, whereas bagging assigns equal weight to each model in the final vote, boosting weights each model's vote dependent on their misclassification rate. Thirdly and most relevant to the theoretic discussion on ensemble methods, boosting is primarily applied to high bias *weak learners*, whereas bagging is most effective for high variance *strong learners* [43]. In other words, boosting primarily increases accuracy through bias reduction, whereas bagging primarily increases accuracy through variance reduction.

2.8 Bayesian model averaging

Another form of ensemble methods is motivated by the Bayesian perspective, in which the available data is believed to be the only data [2]. For this reason the notion of averaging over different datasets is not applicable. In Bayesian inference one would like to find the posterior distribution $p(\theta|x)$, which represents the uncertainty over the parameters θ given an input x. This posterior distribution is calculated using the Bayes' theorem:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}. \tag{13}$$

Following the Bayesian paradigm, the best individual model is the model that maximizes the posterior probability. This model is typically referred to as the Maximum a Posteriori (MAP) estimate. However, although this single model may be the best individual model, by reducing our estimate to a single point estimate, we are ignoring the remaining uncertainty of the posterior distribution [19]. For example , consider a hypothesis space consisting of only three models $(\theta_1, \theta_2, \theta_3)$ in which the respective posterior probabilities are $(0.4, 0.3, 0.3)$. θ_1 may be the best (MAP) estimate, however it is incorrect with high probability (0.6). In order to incorporate all uncertainty into the model selection process, we can form a single model by averaging over the predictions of all possible models weighted by their posterior probability:

$$\int p(y|x,\theta)p(\theta|x)d\theta. \tag{14}$$

This form of model averaging is known as Bayesian model averaging (BMA) and is essentially a weighted ensemble over the predictions of all models in the hypothesis space. BMA is most helpful when the number of training samples is small and therefore the number of hypotheses with large posterior probabilities is high [10].

3. Ensembling Neural Networks

3.1 Deep Ensembles

Despite the numerous and successful ensemble methods developed, the most important of which we have described in section 2, neural networks are most commonly ensembled by simply averaging the output of M networks, all of which have the same architecture, the same optimization algorithm and the same training data [1]. The only difference between each model is its random initialization and/or random shuffling of the training data, which generally leads to different solutions as each model finds a different local optimum [1]. These ensembles are usually referred to as *deep ensembles*. As an example of the performance boost

one can expect from such deep ensembles, [1] finds that an ensemble of 10 networks on the CIFAR100 dataset [26] can *easily* improve test accuracy by 5%.

Despite the simplicity of this approach, ensembling neural networks in this manner often outperforms other ensembling methods in terms of both accuracy and/or computational efficiency. For example [30] finds that deep ensembles outperform bagged neural network ensembles. They attribute this to the fact that bagging reduces the amount of unique datapoints in the training set, as probabilistically about 37% of all points will not be sampled through uniform bootstrap sampling (assuming a sufficiently large set of points). In addition to bagging, boosting methods are also not commonly employed, as boosting requires the neural networks to be trained in sequence and therefore they cannot be trained in parallel [28]. Given that training a neural network can take days or even weeks [21], being able to train all M networks in parallel is strongly desired. Given that each model in a deep ensemble is trained independently from one another, one can train all M networks in a deep ensemble in parallel. Lastly, [14] show that deep ensembles are superior to Bayesian model averaging approaches. They show that for 10 ensembles with average accuracy of 71% on the CIFAR10 dataset [26], deep ensembles increase accuracy by more than 4% relative to four different Bayesian approximation methods.

Although deep ensembles offer considerable accuracy improvement, they do have a considerable shortcoming due to their large storage and computational requirements. We focus on the computational requirements required at prediction time, as knowledge distillation and our proposed alternative approach both require training M individual models initially. The first and most critical drawback is the fact that building M networks with N parameters requires storing $M \times N$ parameters. Given that networks often have millions of parameters such as Alexnet (60M) [27] or $\text{BERT}_{\text{Large}}(340M)$ [9], means that even a linear scaling of storage requirements can be prohibitive. Although such storage requirements may be tolerated in the case of one-off competitions such as Alexnet and Kaggle competitions, certain hardware limitations such as those of mobile devices make the use of deep ensembles infeasible [21]. Lastly, as the output of a deep ensemble is the average of each of its M networks, evaluating an input requires a forward pass through M networks. For this reason, even in cases in which storage constraints are not the bottleneck, utilizing deep ensembles may be infeasible due to time constraints at prediction time.

3.2 Knowledge distillation

In order to overcome the computational challenges of deep ensembles, several methods have been devised to reduce storage and computational bottlenecks. The most successful of these approaches is known as knowledge distillation [16, 20]. The key idea behind knowledge distillation is to train a much larger network when computational constraints are less restrictive and to transfer its *knowledge* to a much smaller, more computationally efficient network, which can be used in production. The large network is referred to as the *teacher*, while the smaller network is referred to as the *student*. It should be noted that although deep ensembles are an ideal candidate for knowledge distillation, as all M models in the ensemble can be distilled into a single model, that knowledge distillation can in theory be applied to any arbitrary architecture and therefore its use-cases are by no means restricted to deep ensembles.

The main motivating idea behind knowledge distillation is that the logits –soft labels– of the teacher model are more informative than the true hard labels of the training data, as the relative probabilities between classes reveal information about class similarities [20]. Given that a teacher model has found a model that generalizes well, it may make more sense to have a smaller model mimic the generalization of the teacher model, rather than discovering its own generalization through hard label training. In practice, typically a combination of both *soft* and *hard* labels leads to optimal knowledge transfer. The knowl-

edge distillation process consists of training the student network with the following loss function (formula from [25]):

$$\text{Loss} = (1-\alpha)L_{CE}(p^s(1), y) + \alpha L_{KL}(p^s(\tau), p^t(\tau)), \tag{15}$$

where L_{CE} is the standard cross entropy loss and L_{KL} is the Kullback-Leibler divergence between the student and teacher output distributions:

Kullback-Leibler divergence is a non-negative measure of disimilarity between two distributions $p(x)$ and $q(x)$, which is minimized (=0) only when $p(x) = q(x)$ [2]. Hence by minimizing KL divergence, the class probabilities of the student network should match that of the teacher.

Note that the loss includes two hyperparameters α and τ. α controls the relative importance between the student matching the class distribution of the teacher network and the student learning individually. A commonly used α in practice is $\alpha = 0.9$, which clearly favors distribution matching [8]. The τ parameter is the temperature of the softmax function. The softmax function is a continuous approximation of the argmax function and by varying the temperature τ, we move from a one-hot encoded vector (as $\tau \to 0$) to a uniform distribution (as $\tau \to \infty$) [23]. [20] suggests that when the class distribution has higher entropy, it is better for learning. This is confirmed by the fact that in practice typically temperatures of 3, 4, and 5 are used [8] and hence higher entropy is induced into the soft targets relative to the standard softmax layer, where $\tau = 1$. Lastly the τ^2 term in the L_{KL} loss, is needed to maintain the contribution weights α and $1-\alpha$, as the gradients of the soft targets scale by $\frac{1}{\tau^2}$ [20].

Empirically, [20] applies knowledge distillation in speech recognition and find that a deep ensemble of 10 models increased accuracy by 2.2%, whereas the distilled ensemble increased accuracy by 1.8%, resulting in a gap of only 0.3%. Additionally, they find another benefit of knowledge distillation to be drastic improvement in training time. Defining the baseline accuracy as the average accuracy of an individual model trained with cross-entropy, they find that the distilled model attained accuracy of -2% relative to the baseline after only seeing 3% of the data (20M datapoints), whereas the baseline model attained accuracy of -14.4% relative to the baseline when trained on the same limited dataset.

3.3 Drawback of Knowledge distillation

Despite the success of knowledge distillation, it also has some significant drawbacks.

Firstly, although there have been some attempts to understand knowledge distillation [7, 8, 33], there is no theoretical consensus as to why knowledge distillation works. We briefly provide an overview of the main theories and mysteries concerning knowledge distillation.

[33] finds that the key success factors of knowledge distillation are data geometry, optimization bias and strong monotonicity. However, their learnings are sourced from the study of deep linear networks and it is unclear whether their findings hold in the nonlinear case. [7] analyses knowledge distillation from the perspective of visual concepts, which they define as image regions for which the network discards the least amount of information and therefore are most important for the networks output. They state that an individual model trained on hard labels discovers visual concepts sequentially (in different epochs) and that network training consists of two phases as per the information-bottleneck theory, in which training initially focuses on learning a multitude of features (phase 1) and subsequently focuses on discarding the least important ones (phase 2). On the other hand, student models learn visual concepts simultaneously and without significant *detours*, i.e., first learning and later discarding irrelevant features. Lastly, [8] studies knowledge distillation empirically and shed some light on some of the unintuitive properties and mysteries

of knowledge distillation. They find that increasing teacher accuracy does not necessarily increase student accuracy and can in fact be a detriment. Initially, student accuracy increases with that of the teacher network, however at some point the relationship is reversed. Their core hypothesis for this behaviour is that smaller networks do not have sufficient capacity to match the output distribution of the very large teacher network. Furthermore, similarly to other previous works, they find that knowledge distillation is unsuccessful on the IMAGENET [27] dataset and that all student models have lower accuracy than the baseline. This begs the question as to why certain datasets can be distilled and why others cannot.

3.4 Other approaches

Although deep ensembles may currently be the most successful ensembling approach, there do nevertheless exist other approaches. We mention these approaches primarily as they will be used in section 4 to help explore why deep ensembles work, as well as help motivate our novel ensemble approach. We stress clearly however that the accuracy gains as a result of deep ensembling drastically outperform those of the following methods.

The first method is simply to average over several predictions made with different weight samples, as opposed to predicting based on a single point estimate [14]. This is essentially a Bayesian model averaging approach. However, despite their Bayesian motivation, several of these methods only mildly resemble a full Bayesian treatment. For example, many of these methods do not learn a posterior distribution and are therefore unable to explicitly quantify uncertainty. Rather, given a trained solution, they employ various random sampling strategies to sample different models which are close to the trained solution in some sense. The simplest example of such an approach is random subspace sampling, in which models are sampled by adding random vectors to the trained solution [14]. Secondly, the methods which attempt to calculate a posterior distribution are limited to posterior approximations, as computing the posterior in the context of neural networks is typically intractable [14] and exact methods such as Monte Carlo Markov Chain (MCMC) do not scale to such high dimensions [24]. Nevertheless, disregarding the specific motivations, all of these methods conceptually amount to sampling from the same basin of attraction as that of the trained solution [14], in an attempt to reduce model uncertainty. For this reason we refer to all of these methods as Bayesian approximations.

Two noteworthy examples of such Bayesian approximations are Monte Carlo dropout (MC dropout) and Gaussian approximations such as the Stochastic-Weight-Averaging (SWA-Gaussian or SWAG) [31] approach. MC dropout is inspired by dropout [40], which is a regularization technique in which weights are randomly *dropped* (set to 0) with probability $p_{dropout}$ during the training process. Whereas traditional dropout is applied during the training process, MC dropout is applied at prediction time. By sampling weights from a fully trained model, in which each weight is selected with probability $1 - p_{dropout}$, we are able to create an ensemble of models over which we can average [14]. The *SWAG* approach on the other hand, aims to learn the parameters of a Gaussian distribution based on the mean and variance of weights observed during training time [31]. While there are variations such as learning a single multi-dimensional Gaussian [31] or learning a one-dimensional Gaussian for each weight [14], they all result in a Gaussian distribution from which one can sample multiple models and over which one can average.

The greatest weakness of both MC dropout and SWAG lies in their lack of model diversity. In MC dropout all models are highly correlated, as they share significant weight overlap [12]. Similarly, the Gaussian distribution approximated by SWAG is based on mean and variance estimations that stem from the same initialization point [12] and the same SGD trajectory. The SGD trajectory has been shown to converge rapidly to a low-dimensional subspace [17], leaving many directions unexplored. Such orthogonal

directions will thus have underestimated variances and therefore be underrepresented in the final posterior approximation.

A second ensemble approach is to average over multiple different weight samples directly in weight space and to use this average as the parameters for the final model. Note that we use the terms weight space and parameter space interchangeably. This is almost the same approach as that of Bayesian approximations, however they differ in two key points. Firstly, whereas Bayesian approximations average over predictions made by different models in function space, weight-averaging methods average directly in weight space and therefore only require a single forward pass at prediction time. Secondly, whereas Bayesian approximations are motivated by model uncertainty, weight-averaging methods are motivated by the geometry of the loss landscape. It is believed that averaging over multiple weight samples from the same basin of attraction leads to *wider* solutions [22]. Width in this sense refers to the step size from the solution for which accuracy is constant. As the loss landscape of the test data is to a degree a shifted version of that of the training data, it is believed that *wider* solutions are more likely to overlap with the test data.

An example of such a method is SWA [22] which is in fact the precursor to the SWAG method discussed previously. The final model parameters are an average over several weights samples from throughout the SGD trajectory. Although [31] finds SWAG to generally be superior, the magnitude of the accuracy gains of MC dropout, SWAG, and SWA are all comparable.

To summarize, we have presented two different ensembling approaches to highlight the following: 1) The fact that one can ensemble in function space or weight space and the results are comparable, given that all methods sample from the same basin of attraction. 2) The largest weakness of all such methods lies in their lack of model diversity, as weight samples are highly correlated. 3) There exists both a statistical and optimization motivation to model averaging in neural network ensembles, namely reducing model uncertainty and finding *wider* solutions.

All three of these observations are fundamental to the proposed permAVG approach. The permAVG approach seeks to find the correct permutations, such that all M models become samples of the same basin of attraction. Thereafter, similar to the SWA approach, the weights can simply be averaged. Due to the model averaging benefits discussed above and the fact that the M weight samples in the permAVG method stem from independently trained models and are therefore likely to have low correlation, we can expect the permAVG approach to outperform the Bayesian approximation methods.

4. Understanding Deep ensembles

Despite the simplicity of the deep ensembling approach, the source of its effectiveness is not entirely understood. In this section we will explore the key ideas and empirical observations regarding deep ensembles from the works [1, 14].

[14] compares deep ensembles to multiple Bayesian approximation methods, including those described in the section 3.4 (MC dropout and SWAG). They find that whereas each model in a deep ensemble finds an independent mode, Bayesian approximations are limited to weight sampling from a single mode. Note that their use of the term mode is inspired by the fact that the posterior distribution of a neural network is multi-modal and that each local minimum in weight space corresponds to a mode in the posterior distribution. Hence, we will use the term mode interchangeably with the terms basin of attraction and the notion of valley in the loss landscape. Evidence for their claim can be seen in Fig. 4, which was extracted from their paper [14]. The authors initially plot the training trajectory of three independently trained models, each of which finds a unique mode. Thereafter they apply Monte Carlo dropout to each of the solutions and

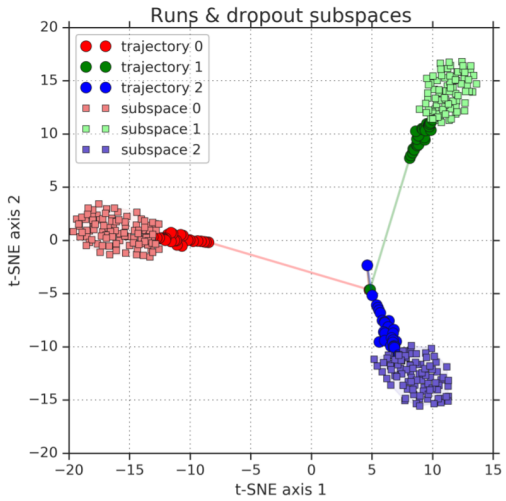

Figure 1: **The training trajectories of three different models and subsequent Monte Carlo dropout samples.** Note that each model finds its own mode, while Monte Carlo dropout, samples from the same mode of the trained model on which it is based. Image extracted from Fort *et al.* [14].

plot the resulting weight samples. It can clearly be seen that MC dropout amounts to sampling from the same mode as that of the trained solution on which it is based. Additionally, the authors show the same behaviour for three other Bayesian approximation methods including the SWAG method.

This observation is relevant to the permAVG approach, as it depicts why neural networks cannot simply be averaged in parameter space, whereas Bayesian approximations can be [1]. If we imagine taking the average over the solutions of the three networks in Fig. 4, it is clear that the average will not lie in any of the three modes and hence there is a large chance that the average model will not live in any valley of the loss landscape. The Bayesian approximations on the other hand, are all weight samples from the same mode and as evidenced by the *SWA* approach can thus benefit from being averaged in weight space.

The fact that each model finds a unique mode however, does not directly imply that each model is learning a unique function. There are symmetries in the weight space such that two models may be distant in weight space, but be functionally equivalent. In fact, for every hidden layer with H_i units, there are $H_i!$ identical functions and for networks of multiple hidden layers the number of identical functions is their product $H_1! x \dots x H_N!$, where N is the number of hidden layers [2] [2]. Hence a central open question regarding deep ensembles is to what extent the individual models find distinct functions versus to what extent the functions are similar.

Both [14] and [1] argue that the functions are distinct. [14] finds that there is considerable diversity between functions and that this diversity is significantly higher that that of the Bayesian approximations, where diversity is measured by the percentage of inputs on which the models disagree. Additionally, [1] hypothesizes that ensembles work due to a *multi-view* structure in the data, in which the correct classification can be achieved using multiple different features. For example, they suggest that depending on the perspective of an image of a car (front, side, back, etc.), different features can be used to classify the car correctly. For example an image from the front may contain features such as windows, wheels and

[1] Note that although they can be averaged in parameter space as evidenced by SWA, most Bayesian approximations are still generally averaged in function space.

[2] We omit symmetries due to sign changes, as our novel approach focuses on permutation symmetries.

side-view mirrors, while an image from the back may contain features such as windows, wheels and tail lights. Any individual model that learns the *front* (*back*) features may generally be able to classify the *back* (*front*) images correctly as there is significant overlap, however an ensemble of both would be able to classify better on average as it makes predictions on the basis of all four possible features. From this perspective the individual models are all believed to be functionally different, as they focus on separate features. Although, this multi-view structure makes intuitive sense, we find no convincing argument that such structure is the underlying mechanism of deep ensembles.

Our permAVG approach will explore to what extent the functions are similar, by exploring to what extent functions only differ by a permutation. If we are able to learn permutations such that all models can be permuted to the same mode, then this is evidence that all models are in fact similar and perhaps an estimation of the same underlying function.

5. Ensembling parameters in weight space

We first compare neural networks to linear models and review why the former cannot simply be averaged in weight space, whereas the latter can. Thereafter, we introduce the notion of permutation invariance, which is a property we exploit in order to permute networks into the same mode, such that they can simply be averaged in weight space. Thereafter, we formally define the permAVG network and describe the final optimization problem. In section 5.2 we discuss how permutation matrices can be learned in the context of neural networks. Lastly, in section 5.3 we compare the permAVG approach to all neural network ensemble methods mentioned in section 3.

5.1 Permutation averaging

As we have mentioned in section 4, neural networks cannot simply be averaged in parameter space. For this reason, neural networks are commonly ensembled in function space, as is the case for deep ensembles. This is in contrast to linear models, which can be simply averaged in parameter space. As a specific example, we limit ourselves to linear regression. In the case of linear regression, averaging over M models in parameter space is equivalent to averaging over the M models in function space [11]:

$$\overline{y} = \frac{1}{M}\sum_{m=1}^{M} X\beta_m = X\frac{\sum_{m=1}^{M}\beta_m}{M} = X\overline{\beta}. \tag{16}$$

Additionally, as the objective function is convex, each model is an estimation of the same true globally optimal parameters. For this reason, the estimations of each individual model may vary due to different datasets, but on average they are equal to the optimal parameters. Note that the second part only holds for unbiased learning procedures such as linear regression.

In order to overcome this drawback in neural networks, we aim to exploit their permutation invariance property. We provide an illustrative example. Consider the following two neural networks $f(x)_1$ and $f(x)_2$, where ϕ is any arbitrary activation function as:

$$f(x)_1 = W_2\phi(W_1x), \tag{17}$$
$$f(x)_2 = W_2P^T\phi(PW_1x), \tag{18}$$
$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}, \tag{19}$$
$$W_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix}, \tag{20}$$
$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{21}$$

Let each individual weight w_i be distinct. We set P to a specific permutation matrix for the sake of illustration, as setting $P = I_2$ would mean that both networks share the same weights. Let v_1 and v_2 be the respective weight vectors for $f(x)_1$ and $f(x)_2$, in other words the coordinates of their parameters in weight space.

In function space (in terms of their outputs) both networks are equivalent, $f(x)_1 = f(x)_2$. However, in weight space the distance between the weight vectors $d(v_1, v_2) > 0$ and can be large. Hence, while averaging the networks in function space ($\frac{f(x)_1+f(x)_2}{2} = f(x)_1 = f(x)_2$) returns the same output, averaging in the weight space returns a new network $f(x)_{wAVG}$ as:

$$f(x)_{wAVG} = \overline{W_2}\phi(\overline{W_1}x), \tag{22}$$

with:

$$\overline{W_1} = \begin{bmatrix} \frac{w_1+w_3}{2} & \frac{w_2+w_4}{2} \\ \frac{w_3+w_1}{2} & \frac{w_4+w_1}{2} \end{bmatrix}, \tag{23}$$
$$\overline{W_2} = \begin{bmatrix} \frac{w_5+w_6}{2} & \frac{w_6+w_5}{2} \\ \frac{w_7+w_8}{2} & \frac{w_8+w_7}{2} \end{bmatrix}. \tag{24}$$

We label averaging over non-permuted weight matrices as *naive* averaging. Generally $f(x)_{wAVG} \neq f(x)_1 = f(x)_2$.

We visualize the previous concepts in Fig. 2. Figure 2 is a 2D representation [3] of both networks $f(x)_1$ and $f(x)_2$ in weight space. Note that permuting the rows of W_1 in $f(x)_1$ by the permutation P and permuting the columns of W_2 by $P^T = P^{-1}$, results in the network $f(x)_2$. Although, both networks remain functionally equivalent, in weight space both networks lie in distinct modes. If one takes the average of both $f(x)_1$ and $f(x)_2$ in weight space, we see that the average W_{avg} does not lie in either of the two modes. For this reason, naively averaging the weights generally leads to low accuracy networks.

Returning to our discussion on deep ensembles, we saw that each individual model in a deep ensemble finds a unique mode and that each model has roughly the same accuracy. Let $f(x)_A$ and $f(x)_B$ be two models in the two-model deep ensemble $f(x)_{AB}$. Similar to our example above, we can average models in a deep ensemble in function space, as $f(x)_A \approx f(x)_B$. However, naively averaging the weights, performs poorly as the distance between models in weight space may be large and the naive average of both most likely does not lie in a valley of the loss landscape (see Fig. 2). Empirically, our tests support this claim and we find that naive averaging over 6 models already results in near-random accuracy.

[3] This could be a t-SNE or PCA representation. The details do not matter for the example.

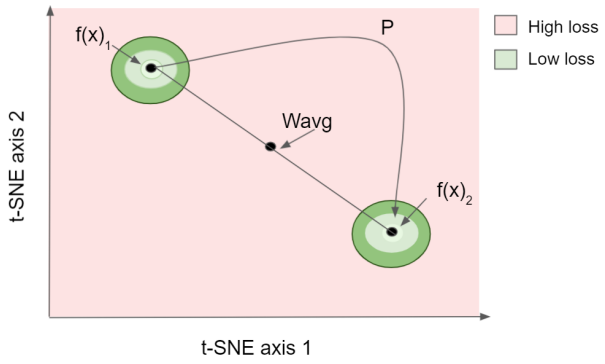

Figure 2: **2D visualization of the networks $f(x)_1$ and $f(x_2)$ in weight space.** Note that by simply permuting the rows of W_1 and the columns of W_2, we create a functionally equivalent network $f(x_2)$, which lies in a different mode in the loss landscape. Also note that their average W_{avg}, does not lie in a valley of the loss landscape.

The core hypothesis of our paper is the following: The fact that each model in an ensemble finds a unique mode is a consequence of the fact that each model was trained independently with a unique random seed. However, different modes do not necessarily imply uniqueness, as the loss landscape is full of symmetries such as permutation invariance. Hence we hypothesize that all models in an ensemble are approximations of the same underlying function and that they differ primarily in the specific permutation of their weights and not in the uniqueness of their functions. Therefore, we will attempt to learn these latent permutations, such that by applying their inverse to each model in the ensemble, we can translate each model into the same mode in the loss landscape. Thereafter, as all models are then samples of the same mode, we can average the models in weight space similar to SWA. Due to the fact that each model was initially trained independently, it is believed that these weight samples will be significantly less correlated than those of Bayesian approximations and will therefore lead to substantially higher performance. Additionally, averaging in weight space, reduces the ensemble to a single model which only requires a single forward pass at prediction time. This is in contrast to deep ensembles and Bayesian approximation methods, which require a forward pass for every model in the ensemble.

We now frame this approach as an optimization problem. We first note that the actual optimization problem does not learn discrete permutation matrices, but rather a continuous relaxation of permutation matrices known as doubly stochastic matrices (DSM matrix). As opposed to discrete permutation matrices, where every row and column contains a single element with value 1 and all other elements with value 0, DSM matrices relax this restriction by only requiring all rows and columns to sum to 1. A full description of DSM matrices and how they can be learned in the context of neural networks will be given in section 5.2.

Let $W_{l,n}$ represent the weight matrix of the n-th model in the l-th layer and let $P_{l,n}$ be the DSM matrix shuffling the weights $W_{l,n}$. Let $b_{l,n}$ be the bias of the n-th model in the l-th layer. We define the following network of l-layers and n-models $f(x)_{permAVG}$:

$$
\begin{aligned}
&f(x)_{permAVG} = \\
&\frac{1}{N}\sum_{n=1}^{N}(W_{(l,n)}P_{(l-1,n)}^{-1}\left\{\dots\left\{\phi(\frac{1}{N}\sum_{n=1}^{N}P_{(2,n)}(W_{(2,n)}P_{(1,n)}^{-1}\left\{\phi(\frac{1}{N}\sum_{n=1}^{N}P_{(1,n)}(W_{(1,n)}\mathbf{x}+b_{1,n}))\right\}+b_{2,n}))\right\}\dots\right\}+b_{l,n}) \\
&P_{(l,n)}\in[0,1]^{dxd}, \sum_i P_{(l,n)}^{ij}=1, \sum_j P_{(l,n)}^{ij}=1 \ , \qquad (25)
\end{aligned}
$$

We will refer to this network as permutation averaging (permAVG), which differs from *naive averaging* in the sense that we are averaging over permuted weights. We will now discuss some key aspects of the permAVG expression in Eq. (25). Firstly, it is important to note the we do not permute the weights in the final layer l, as the notion of permuting logits is nonsensical. Secondly, as opposed to the permutation invariance example in Eq. (22), the distinction between P^{-1} and P^T is important as the transpose of a DSM matrix is not equivalent to its inverse [34]. This is in contrast to permutation matrices where $P^T = P^{-1}$ holds. Lastly, the inclusion of the P^{-1} term stems from the fact that in the single model case, P^{-1} is required to maintain permutation invariance. We illustrate the importance of this permutation invariance in Eq. (26) in the case of a two-model, two-layer permAVG ensemble. To simplify notation we exclude the bias term and assume a linear activation function as:

$$f(x)_1 = W_{21}W_{11}x, \tag{26}$$

$$f(x)_2 = W_{22}W_{12}x, \tag{27}$$

$$\begin{aligned}\text{permAVG} &= \frac{1}{2}W_{21}P_1^{-1}\left(\frac{P_1W_{11}x + P_2W_{12}x}{2}\right) + \frac{1}{2}W_{22}P_2^{-1}\left(\frac{P_1W_{11}x + P_2W_{12}x}{2}\right) \\ &= \frac{1}{2}W_{21}\left(\frac{W_{11}x + P_1^{-1}P_2W_{12}x}{2}\right) + \frac{1}{2}W_{22}\left(\frac{P_2^{-1}P_1W_{11}x + W_{12}x}{2}\right).\end{aligned} \tag{28}$$

The first component of the result in Eq. (26) is a combination of the original network $f(x)_1$ and an almost identical network, which deviates from $f(x)_1$ only in the replacement of W_{11} with a permutation of W_{12}. In this sense, we have essentially frozen $f(x)_1$ and are learning a permutation relative to this network. For example, perhaps it is optimal to permute W_{12} such that its weights are as close as possible to W_{11} [4]. Similarly, in the second component of Eq. (26), we are permuting the weights W_{11} relative to the network $f(x)_2$. If we did not include P^{-1} in the permAVG formulation, then we would not be learning optimal permutations relative to the existing networks, but rather searching through all possible permutations of the weight matrices, most of which would lead to low accuracy networks. In this sense, including P^{-1} can be seen as an inductive bias, which limits the search of permutations to those that are close to the original networks. In practice, we find the inclusion of P^{-1} to be essential and that its omission leads to an inability to learn optimal permutations.

We now state the optimization problem of the permAVG approach. Suppose that we have D data points of the form $(\mathbf{x}_i, y_i)$. Let τ be a hyperparameter controlling the "softness" of the DSM matrices. τ will be discussed in detail in section 5.2, but for now it is sufficient to understand that for small τ, DSM matrices approximate hard permutation matrices, whereas for large τ, the rows and columns of a DSM matrix approximate a uniform distribution [32]. We now define the optimization problem:

$$\min_{\tau, P(1,1),\ldots,P(l,n)} \sum_{i=1}^{D} \text{loss}(f(x)_{permAVG}, y_i), \tag{29}$$

where loss is any arbitrary loss function such as cross entropy. Formally, for a deep ensemble of l-layers and n-models, such an optimization problem requires learning $(l-1)N$ DSM matrices. In practice however, one only needs to learn $(l-1)(N-1)$ DSM matrices, as all permutations of N weight matrices can be expressed both in terms of N or $N-1$ permutation matrices. The solution to Eq. (29) is the optimal shuffling of each weight matrix such that their average produces the network of highest accuracy.

[4] This is just an example and permuting weights to be close may not be optimal.

5.2 Learning permutations

As described in section 5.1, the permAVG approach requires optimizing over permutation matrices. For this reason, we provide an overview of permutation learning in the context of neural networks.

A permutation matrix is a binary square matrix, in which each row and each column have a single element with value 1 and all other elements with value 0 . Due to the fact that permutation matrices form a discrete set, they are unsuited for gradient-based optimization which requires differentiability [35]. For this reason it is better to consider a continuous relaxation of permutation matrices, namely the Doubly Stochastic Matrices (DSM). A DSM matrix is a non-negative matrix, in which all rows and columns must sum to 1. This relaxation follows from the fact that the set of permutation matrices form the vertices of the Birkhoff polytope, which is the set of all DSM matrices. In other words, all DSM matrices can be expressed as a convex combination of permutation matrices [35].

One method to learn a permutation matrix is through a parameterization by a square cost matrix, in which the learned permutation matrix is the solution to the Linear Assignment Problem (LAP) [32]. Consider a LAP in which n people need to be assigned to n jobs and that the cost of each person doing a specific job varies. This can be represented in a cost matrix X, where row i refers to person i and column j refers to job j and hence the element x_{ij} represents the cost of person i doing job j. Let P_n be the set of all permutation matrices of size $n \times n$. The optimal assignment can be framed as the following optimization problem:

$$M(X) = \arg\max_{P \in P_n} tr(P^T X). \tag{30}$$

Note that in order to be consistent with [32], we frame the problem as a maximization problem, although intuitively, minimization may seem more appropriate as the goal of a LAP is typically to minimize assignment cost. However, finding the argmax of a negated cost matrix is equivalent to finding the argmin of the original cost matrix and therefore the choice is irrelevant. Additionally, note that the optimal assignment P, is not the permutation matrix which maximizes the trace of the cost matrix - P^T, but its inverse.

[32] defines $M(X)$ as the matching operator, which given a cost matrix as input, returns a permutation matrix. As the number of $n \times n$ permutation matrices is $n!$, solving the matching operator is intractable for large n. For this reason the hard matching problem is relaxed by replacing P_n with the set of all DSM matrices B_n (B for Birkhoff polytope) and adding an entropic regularization term, which forms the following *soft* LAP [37] (formula from [32]):

$$S(\frac{X}{\tau}) = \arg\max_{P \in B_n} tr(P^T X) + \tau h(P), \tag{31}$$

$$\text{with} \quad h(P) = -\sum_{i,j} P_{i,j} \log(P_{i,j}), \tag{32}$$

where $h(P)$ is the entropy of the DSM matrix, τ is a temperature parameter controlling the *softness* of the DSM matrices and $S()$ is the Sinkhorn operator which we will define in a moment. The key result in [32] is the proof that as $\tau \to 0$, $M(X) = S(\frac{X}{\tau})$. In other words, for small τ one can approximate the hard permutation matching operator by the solution to the soft LAP.

Lastly, the solution to the soft LAP (see Eq. (31)) is given by the Sinkhorn algorithm. Sinkhorn proved that for any positive matrix there exists a unique corresponding DSM matrix and that this DSM matrix

can be found as the limit of alternating row and column normalizations [38, 39]. This procedure is known as the Sinkhorn-Knopp algorithm. For the rest of this work we use the Sinkhorn-Knopp algorithm variant as defined by [32], which works as follows (algorithm is copied verbatim):

$$\begin{aligned} S(X)^0 &= \exp(X) \\ S(X)^l &= T_c(T_r(S(X)^{l-1})) \\ S(X)^l &= \lim_{l \to \infty} S(X)^l \end{aligned} \tag{33}$$

where T_r and T_c represent row and column normalizations respectively. For example, to normalize row i we divide each $x_{i,j} \in$ row i by $\sum_i x_{i,j}$. Note that Eq. (33) deviates from the original sinkhorn algorithm [38] only by the initial step in which we take the exponential of X. This is done to guarantee that X is a positive matrix, as otherwise there is no guarantee that the algorithm will converge nor that a unique DSM matrix exists [39]. Additionally, all steps in Eq. (33) are differentiable [35], which means that we can embed such a procedure into a neural network. Lastly, note that in practice we cannot take the limit and therefore the algorithm is only run for a number of iterations (e.g., 100).

Finally, we will relate the previous material on permutation matrices to that of learning permutation matrices in neural networks. As an example, suppose that our network is given a scrambled input sequence $\pi(x_1, x_2, \dots, x_n)$ and we would like to learn the permutation π. The goal of the network will be to learn the parameters of an *nxn* cost matrix. This cost matrix will then be passed through a Sinkhorn layer [35], which will select a final DSM matrix by applying the Sinkhorn algorithm to the learned cost matrix. As all operations in the Sinkhorn layer are differentiable, the errors can be backpropagated and the parameters of the cost matrix updated using gradient descent.

5.3 Comparison to existing ensembling methods

Now that we have motivated and defined the permAVG approach, we will highlight its differences to deep ensembles, knowledge distillation and Bayesian approximations.

Firstly, we hypothesize that the permAVG method will be superior to the Bayesian approximation and weight-averaging approaches mentioned in section 3.4 in terms of accuracy. The primary argument for this is model diversity. Whereas all models in the Bayesian approximation and weight-averaging methods are based on the same training trajectory and same solution of a single model, the models on which the permAVG method is based are all independently trained. For this reason, the permAVG ensemble is expected to have significantly less inter-model correlation. Furthermore, the permAVG method has the additional ability to optimize over different sets of models. Whereas Bayesian approximations select a single set of models, the permAVG method selects a set of models in every gradient descent iteration and makes future steps in directions of a more optimal set. An analogous procedure adapted to Bayesian approximation methods would amount to optimizing over the solutions of several ensembles. For example, to build X MC dropout ensembles of M models each and subsequently choose the optimal ensemble out of all X ensembles.

In comparison with deep ensembles, we see no reason why the permAVG approach should be superior in terms of accuracy. The motivation to average the M models in weight space rather than in function space, stems from computational and not performance constraints. For this reason, we view the accuracy of deep ensembles as an upper bound on the accuracy of the permAVG method. Similarly, knowledge distillation is also essentially upper-bounded by the accuracy of the deep ensemble on which it is based, as the student

network is attempting to match the output distribution of the teacher. In this manner we believe the potential accuracy of knowledge distillation and the permAVG method to be similar, but we make no claims regarding which method is more accurate. We do however hypothesize that the permAVG method should be more robust, as knowledge distillation is known to fail in some instances due to insufficient student capacity or mysteriously for certain datasets [8].

Lastly, in terms of storage and computational requirements, the permAVG approach is superior to both deep ensembles and Bayesian approximations. Firstly, permAvg reduces an ensemble of M models into a single model. In contrast, deep ensembles require storing M models. Secondly, at prediction time both deep ensembles and Bayesian approximations of M models, require M forward passes, while the permAvg method only requires a single forward pass. Compared to knowledge distillation, both methods perform identically in terms of storage and computational requirements. The only computational drawback of the permAVG method is that it requires training M models initially, whereas Bayesian approximations only require training a single model. However, deep ensembles, knowledge distillation, and the permAVG method all require M independently trained models as input and this appears to be a key factor in their ability to significantly increase accuracy.

6. Implementation details and overview of experiments

We test the permAVG approach on both the FMNIST [44] and CIFAR10 [26] datasets. FMNIST [44] is a slightly more difficult alternative to the MNIST [29] dataset. It consists of 60,000 train and 10,000 test images, each of which is a 28×28 grayscale image of a single clothing item. There are 10 clothing items in total. The CIFAR10 [26] consists of 50,000 train and 10,000 test images, each of which is a 32×32 color image of an object or animal. There are 10 classes in total. The class distribution in both datasets is perfectly balanced such that one can expect a network with randomly assigned weights to attain an accuracy of 10%.

We train multiple multilayer perceptrons (MLP) with 2, 3 and 4 layers per dataset, respectively. These are the individual models from which we form the deep ensembles, the permAVG models and the knowledge distillation models. We will refer to these individual models as submodels from this point on. The exact architecture for the submodels for both datasets can be seen in Appendix A in Fig. 10. Thereafter, we build permAVG models consisting of 2, 3, 4, 5 and 6 submodels each, for each of the different number of layers 2, 3, and 4. In total this leads to 15 distinct (num_models x num_layers) combinations.

We now provide an overview of the implementation of the permAVG approach. The permAVG model takes in two inputs: the N submodels, each of L layers, that form the deep ensemble and the Sinkhorn temperature τ. Note that one should *freeze* the parameters of the N submodels, as we do not want to optimize over these parameters. Then for each of the N models and the first $L-1$ layers, we initialize a cost matrix. Note that this amounts to a cost matrix for each DSM matrix we would like to learn and that the final layer L does not require learning any DSM matrices. The cost matrices are saved in a dictionary, where they can be accessed by the key (model_index, layer_index). Based on our experiments, we found initializing all cost matrices to the Identity matrix to perform better than random initialization. After the cost matrices have been initialized, we are able to run a forward pass.

An overview of the forward pass is given in Algorithm 1. The algorithm works as follows: We loop through each of the L layers and for each layer we calculate a weighted sum of the N model's permuted

output, where the weighting is $\frac{1}{N}$ for all models. The output of each individual submodel is a permutation of the original submodel's output. Note that permuting the output is equivalent to permuting the weights and bias of a submodel $[P(Wx + b) = (PW)x + Pb]$. The permutation of each submodel is calculated as follows: Given the model_index and layer_index, we retrieve the cost matrix associated with the specfic layer of the submodel. Thereafter, we run the Sinkhorn algorithm on the cost matrix using the τ we specified and for 100 iterations. This converts the cost matrix into a DSM matrix. Note that we do not alter the cost matrix of the submodel with model_index 1, as one can express all permutations of N separate sequences with either N or $N-1$ permutations. Therefore, as all cost matrices were initialized with the Identity, the DSM matrix of submodel 1 will always be the Identity matrix. Hence the output of submodel 1 will always be equivalent to its permuted output. After we have computed the output of a given layer we apply the ReLU activation function. Lastly, for all layers except for layer_index 1, we must apply P^{-1} to the previous layer's output. To accomplish this, for each submodel, we permute the submodel's input by the inverse of the DSM matrix from the previous layer. Note that the previous DSM matrix can simply be found by looking up the key (model_index, layer_index-1) in the cost matrices dictionary and applying the Sinkhorn algorithm.

Given the aforementioned forward pass algorithm, we can calculate the output for a given input. We train the permAVG model for 20 epochs using the ADAM optimizer, a learning rate of 0.1, and a cosine annealling scheduler. We use cross-entropy as the loss function. Initially, we set $\tau = 1$, although we also experiment with various different τ.

We now provide an overview of all experiments. In section 7.1, we compare the accuracy of deep ensembles, knowledge distillation, the permAVG approach and the average of the submodels (baseline). Thereafter, in section 7.2 we investigate what type of permutations the permAVG approach is learning, by analysing the differences between the original weights and the permuted weights along criteria such as distance, standard deviation and correlation. In section 7.3, we propose an experiment to support our claim that networks are being permuted into the same mode. In section 7.4, we investigate the sparsity of the DSM matrices, in order to understand their proximity or distance to hard permutation matrices. Lastly, in section 7.5 we explore the impact of the Sinkhorn temperature τ on accuracy.

7. Experimental Results

7.1 permAVG performance

In Fig. 3 we plot the average accuracy for deep ensembles (deepEns), for our proposed permAVG approach, for Knowledge Distillation (KD) and for the individual models of which the ensembles are composed (submodels) for both the CIFAR10 [26] and FMNIST [44] datasets. The average accuracy of the submodels is regarded as the baseline. The exact accuracies are given in Fig. 11 in Appendix A.

In order to increase readability, we refer to the different ensembling methods as follows: A N-submodel, L-layer KD is a knowledge distillation model with a teacher consisting of N submodels each of L layers. Note that the more complete description is "N-submodel, L-layer KD model", but we drop the word "model" to increase readability. Similarly we will refer to a "N-submodel, L-Layer permAVG model" as a "N-submodel, L-Layer permAVG".

As expected, regardless of model width (number of layers), the accuracy of deep ensembles increases with the size of the ensemble. This trend is more apparent in the CIFAR10 [26] dataset, although the same pattern can be seen in the FMNIST [44] dataset. The 6-submodel deep ensemble increases average accuracy

Algorithm 1 A forward pass of a permAVG model

$models \leftarrow [M_1, M_2, \dots, M_N]$
$num\_models \leftarrow N$
$num\_layers \leftarrow L$
$cost\_matrices \leftarrow \{(M_1, L_1) : C_{11}, \dots, (M_1, L_L) : C_{1L}, \dots, (M_N, L_1) : C_{N1}, \dots, (M_N, L_L) : C_{NL}\}$
$sinkhorn\_temp \leftarrow \tau$
$sinkhorn\_iterations \leftarrow 100$
$x \leftarrow input$
for $layer\_index = 1 \dots num\_layers$ **do**
  if $layer\_index \neq 1$ **then**
    $x \leftarrow RELU(layer\_output)$
  end if
  $layer\_output \leftarrow 0$
  for $model\_index, model \in models$ **do**
    $cost\_matrix \leftarrow cost\_matrices[model\_index, layer\_index]$
    if $model\_index \neq 1$ **then**
      $perm\_matrix \leftarrow SINKHORN(cost\_matrix, sinkhorn\_temp, sinkhorn\_iteratioms)$
    else
      $perm\_matrix \leftarrow cost\_matrix$
    end if
    $model\_input \leftarrow x$
    if $layer\_index \neq 1$ **then**
      $prev\_cost\_matrix \leftarrow cost\_matrices[model\_index, layer\_index - 1]$
      if $model\_index \neq 1$ **then**
        $prev\_perm\_matrix \leftarrow SINKHORN(prev\_cost\_matrix, sinkhorn\_temp, sinkhorn\_iterations)$
      else
        $prev\_perm\_matrix \leftarrow prev\_cost\_matrix$
      end if
      $model\_input \leftarrow matmul(inv(prev\_perm\_matrix), model\_input)$
    end if
    $model\_output \leftarrow model(model\_input)$
    $model\_output \leftarrow matmul(perm\_matrix, model\_output)$
    $layer\_output += model\_output * \frac{1}{num\_models}$
  end for
end for

by 4.3% relative to the 2-submodel deep ensemble for CIFAR10 [26] and by 1% for FMNIST [5]. KD accuracy scales similarly with the number of models in the CIFAR10 [26] dataset, however not in the FMNIST [44] dataset. The 6-submodel KD increases average accuracy by 2.9% relative to the 2-submodel KD for CIFAR10 [26] and by 0.4% for FMNIST [44]. Note that although the FMNIST [44] average is positive, it is small and there exist several examples where larger ensembles underperform the 2-submodel ensemble. For example the 4-submodel, 3-layer KD performs -1.3% worse than its 2-submodel counterpart. In contrast to both deep ensembles and knowledge distillation, the accuracy of the permAVG approach decreases with

[5] The accuracies were normalized by minimum accuracy of the two-submodel, two-layer model such that the results are comparable across datasets.

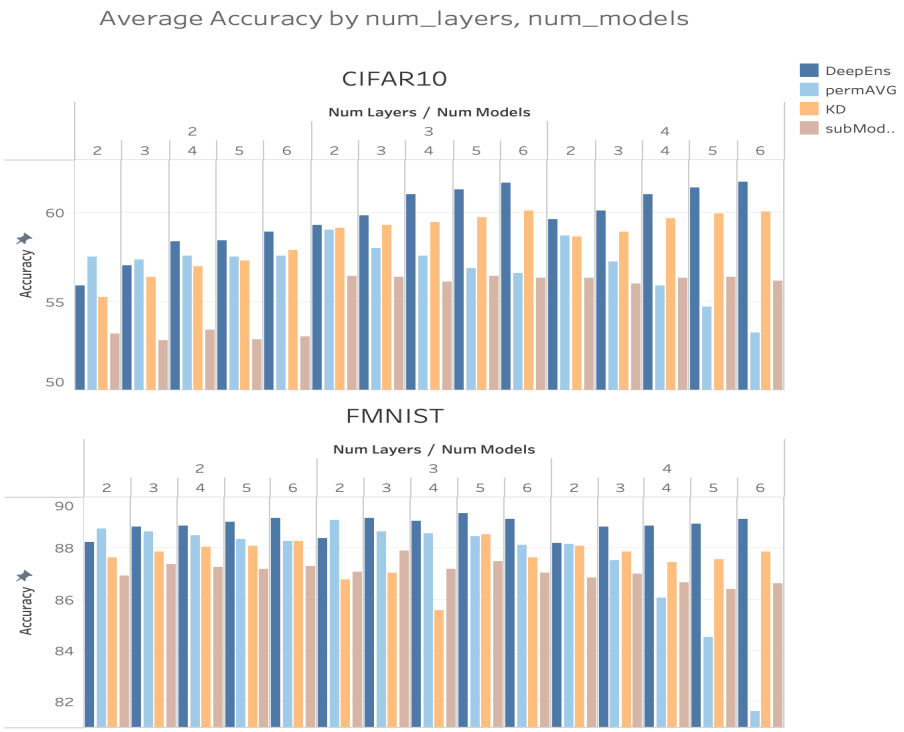

Figure 3: **Top:** Average accuracy on CIFAR10 [26] for deepEnsemble ("deepEns"), permAVG, knowledge distillation ("KD") and the submodels. **Bottom:** The same repeated for the FMNIST [44] dataset.

ensemble size. Compared to the 2-submodel permAVG, the 6-submodel permAVG has -3% lower average accuracy in CIFAR10 [26] and -4.5% in FMNIST [44]. Note that these averages are heavily impacted by the 6-submodel, 4-layer case, where permAVG scores $-7,3\%$ and -9.4% lower than the 2-submodel case in CIFAR10 [26] and FMNIST [44], respectively. In fact the 6-submodel, 4-layer permAVG even performs worse than baseline. However, despite the inability to scale with ensemble size, the 2-submodel permAVG offers significant accuracy improvement relative to the baseline and can often match or even outperform the 2-submodel deep ensemble and KD counterparts. Compared to the 2-submodel deep ensemble, 2-submodel KD and 2-submodel baseline, the 2-submodel permAVG has 0.3%, 1.3%, and 5.6% higher average accuracy respectively in CIFAR10 [26] and 0.4%, 1.3% and 1.9% in FMNIST [44]. However, when we compare the 2-submodel permAVG to the 6-submodel deep ensemble and 6-submodel KD, relative accuracy becomes -3.8% and -1.5% in CIFAR10 [26] and -0.5% and $+0.85\%$ in FMNIST [44]. To summarize, based on our results, the 2-submodel permAVG is able to significantly increase the accuracy compared to the baseline, is able to outperform the 2-submodel deep ensemble and 2-submodel KD, and is able to outperform the 6-submodel KD in FMNIST [44].

From these results we highlight the following key takeaways: 1) The 2-submodel permAVG is a viable ensemble method, which can give comparable results to knowledge distillation and deep ensembles of small sizes. In cases in which submodel training is very expensive, the permAVG approach could even be the optimal choice. 2) The permAVG approach outperforms Bayesian approximation methods in terms of accuracy. As a 2-submodel permAVG can outperform a 2-submodel deep ensemble and as a 2-submodel deep ensemble can drastically outperform Bayesian approximation methods, we conclude that a 2-submodel per-

mAVG should outperform Bayesian approximations. [14] finds that 2-submodel deep ensembles outperform Bayesian approximations by around 2% on the CIFAR10 [26] dataset. 3) There is no evidence that the permAVG technique is more robust than knowledge distillation for the CIFAR10 [26] and FMNIST [44] datasets. Although, knowledge distillation has some strange results such as the low performance of the 4-submodel, 3-layer KD, it otherwise consistently outperforms the baseline. 4) A core weakness of the permAVG approach is its inability to scale with ensemble size. Perhaps by identifying the source of this scalability bottleneck we can even further increase the accuracy gains from the permAVG approach. It is interesting to note that whereas knowledge distillation never outperforms the deep ensemble, the permAVG method can. For this reason, the potential of the permAVG method may not even be upper-bounded by that of the deep ensemble.

7.2 Exploring weight permutations

Previously, we have seen that the permAVG method is able to outperform the baseline and occasionally even deep ensembles and knowledge distillation. Furthermore, we find naive averaging of the weights to perform extremely poorly. For example, naively averaging two submodels with around 52.5% accuracy each, results in a model with accuracy of 38%. Naively averaging 6 submodels already results in near-random accuracy of 10% [6]. Therefore, it is clear that permuting the weights prior to averaging has significant impact. In order to investigate what the permutations have *learned*, we explore various statistics of the weight matrices before and after they have been permuted.

In Fig. 4 one can see the comparison of weight matrices, before and after they have been permuted, in terms of distance, standard deviation and correlation. In order to illustrate how these statistics are calculated and what they imply, we provide the following example. Let W_1 and W_2 be the following two weight matrices from the first layer of an ensemble of two models:

$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}, \quad W_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix}, \tag{34}$$

Suppose that the permAVG method has learned that it is optimal to permute W_2 by $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Hence the permuted weights are:

$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} PW_2 = \begin{bmatrix} w_7 & w_8 \\ w_5 & w_6 \end{bmatrix}. \tag{35}$$

The final weight matrix in the naive average (W_{AVG}) and the permAVG ($W_{permAVG}$) case would then be:

$$W_{AVG} = \begin{bmatrix} \frac{w_1+w_5}{2} & \frac{w_2+w_6}{2} \\ \frac{w_3+w_7}{2} & \frac{w_4+w_8}{2} \end{bmatrix}, \tag{36}$$

$$W_{permAVG} = \begin{bmatrix} \frac{w_1+w_7}{2} & \frac{w_2+w_8}{2} \\ \frac{w_3+w_5}{2} & \frac{w_4+w_6}{2} \end{bmatrix}. \tag{37}$$

[6] Both CIFAR10 [26] and FMNIST [44] have 10 classes and each class comprises 10% of the dataset. Hence a random prediction would be correct 10% of the time.

The permAVG approach has learned that it is optimal to align row 1 of W_1 with row 2 of W_2 before averaging. We would like to investigate why this alignment is optimal relative to the initial alignment of W_{AVG}. For this reason we calculate various inter-row statistics such as distance, standard deviation and correlation. In order to demonstrate how these statistics are calculated, we provide the expression for each statistic in the W_{AVG} case:

$$\text{mean distance} = \frac{1}{2}\text{dist}(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}) + \frac{1}{2}\text{dist}(\begin{bmatrix} w_3 \\ w_4 \end{bmatrix}, \begin{bmatrix} w_7 \\ w_8 \end{bmatrix}), \tag{38}$$

$$\text{mean correlation} = \frac{1}{2}\text{corr}(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}) + \frac{1}{2}\text{corr}(\begin{bmatrix} w_3 \\ w_4 \end{bmatrix}, \begin{bmatrix} w_7 \\ w_8 \end{bmatrix}), \tag{39}$$

$$\text{mean std} = \frac{1}{2}(\frac{1}{2}\text{std}(w_1, w_5) + \frac{1}{2}\text{std}(w_2, w_6)) + \frac{1}{2}(\frac{1}{2}\text{std}(w_3, w_7) + \frac{1}{2}\text{std}(w_4, w_8)). \tag{40}$$

Now to provide a concrete example, suppose the following: $dist(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}) >> dist(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \begin{bmatrix} w_7 \\ w_8 \end{bmatrix})$. In other words, the distance between row 1 of W_1 and row 1 of W_2 is much larger than that of row 1 of W_1 and row 2 of W_2. This implies that the permAVG method has learned permutations which align the rows such that their mean distance is small.

We return to Fig. 4. It is clear in both datasets that the permuted weights have reduced mean distance, reduced mean standard deviation and increased mean correlation. The reduction in distance and standard deviation suggest that weights are being aligned such that they are closest. This supports our hypothesis that solutions in different modes may be permuted into the same mode. Furthermore, we find the correlation prior to permutation to be very close to 0. This indicates that the individual models are very diverse, which is as expected as all models were trained independently. This supports our claim that the permAVG method should significantly outperform Bayesian approximation methods, as the weights are significantly less correlated. The positive correlation after permutation, reiterates the same message as that of distance and standard deviation, namely that the permutations are trying to match rows such that their values are closest. Interestingly, it is in the correlation statistic that we can see the most pronounced impact of increasing model complexity. For large number of layers and submodels, the permAVG model is unable to match rows such that they have high correlation.

7.3 Exploring the single mode hypothesis

The main motivation behind the permAVG approach is the hypothesis that all submodels are an estimation of the same underlying function and that they only differ in their specific permutation of their weights. Hence, by learning the correct permutation, we can permute the submodels such that they all lie in the same mode in the loss landscape and can thus be averaged in weight space similar to SWA. Furthermore, we have empirically seen that naive averaging of weights leads to very poor results and we have hypothesized that this would occur as the individual submodels lie in distinct and perhaps distant modes and that therefore, their average does not lie in a mode in the loss landscape (see Fig. 2). In order to test these two hypotheses, we perform the following experiment: We select two submodels and calculate their accuracy and total loss for several different weighted averages in both the naive average and permAVG case as:

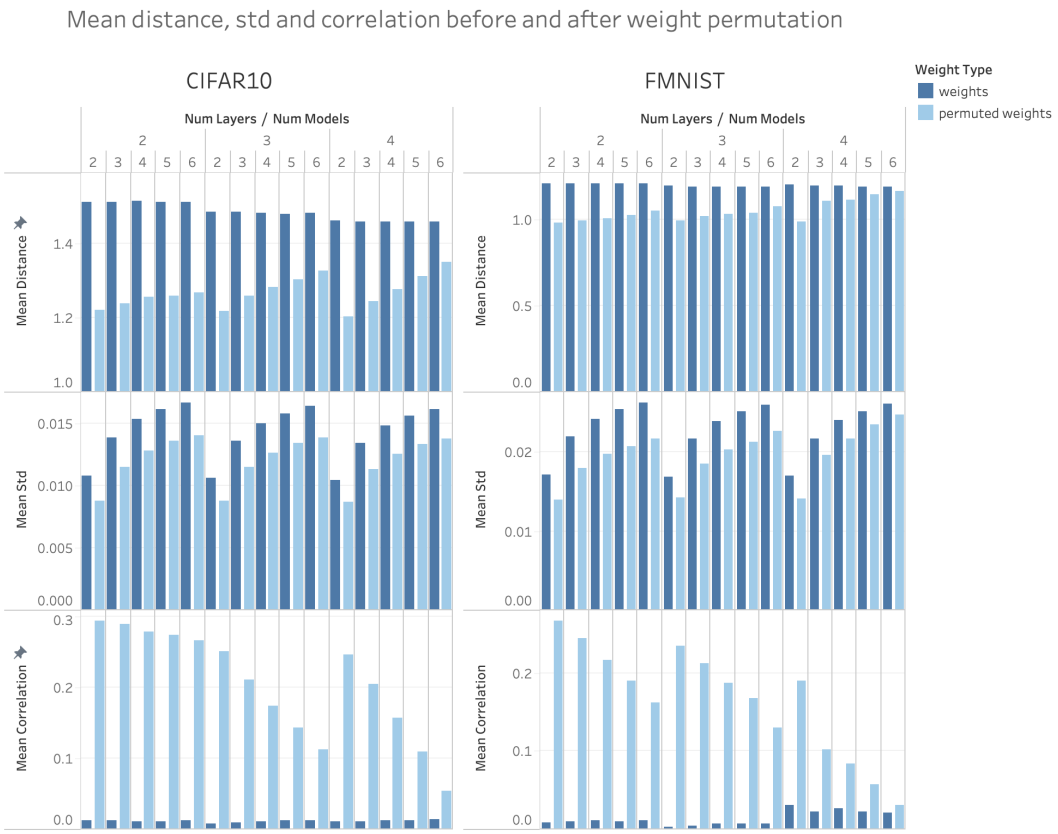

Figure 4: **Mean distance, standard deviation and correlation between weight rows of weight matrices prior to and post permutation.** The differences between the original row alignment and the post permutation alignment, illustrate the criteria which the permAVG method has learned to optimize.

$$W_{AVG} = \alpha W_1 + (1 - \alpha) W_2, \tag{41}$$

$$W_{permAVG} = \alpha W_1 + (1 - \alpha) P W_2. \tag{42}$$

The parameter α defines the weighting of the average. In the naive average case, we predict that weighted averages that are close to the original models (e.g., $\alpha = 0.9$ or $\alpha = 0.1$) should be more performant than $\alpha = 0.5$, as the prior most likely remains in the mode of one of the individual submodels, while the latter may not lie in a mode at all. In contrast, in the permAVG approach we expect both models to lie in the same mode and therefore $\alpha = 0.5$ must lie in the same mode as well. In other words, we should not see a decrease in accuracy (increase in total loss) at $\alpha = 0.5$ in the permAVG case. Additionally, if both models are indeed in the same mode, then we would expect their average at $\alpha = 0.5$ to perform better than $\alpha = 0$ or $\alpha = 1$, as model averaging should reduce model uncertainty and/or find *wider* solutions. The results can be seen in Fig. 5.

In the case of naive averaging, we see that the accuracy is lowest (total loss is highest) at $\alpha = 0.5$. This supports the idea that the midpoint between two models, most likely does not lie in either mode of the two models. For the permAVG approach, we see that accuracy is highest (total loss is lowest) at around

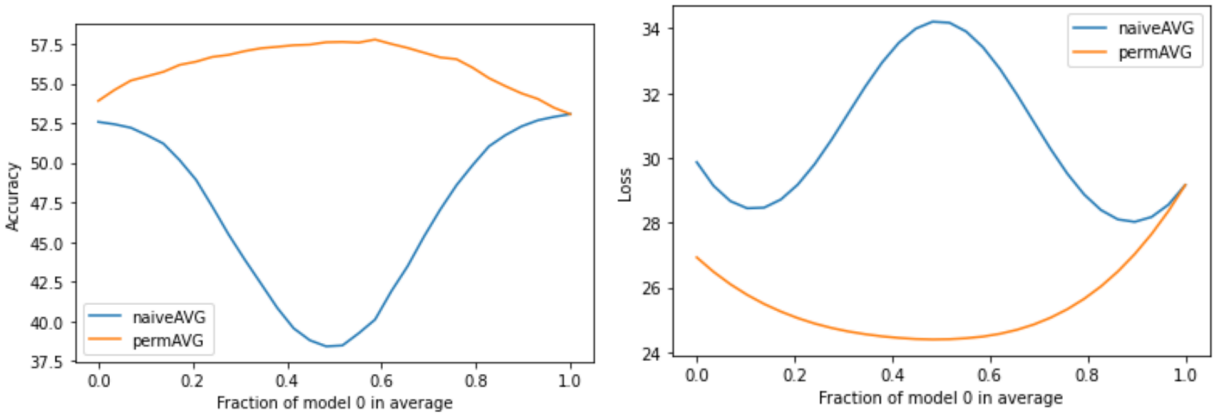

Figure 5: **Considering two models M_0 and M_1. Left:** Accuracy for several different weighted averages, where the x-axis represents the α of the weighting. $\alpha = 1$ ($\alpha = 0$) is the accuracy of M_0 (M_1) individually. **Right:** The same repeated for total cross entropy loss over a test set.

$\alpha = 0.5$. Therefore, both models most likely lie in the same mode, as there is no drop at their midpoint. Additionally, their average is more accurate than each model individually, which is what we would expect from averaging two diverse models in the same mode.

7.4 Exploring permutation sparsity

Although we have theoretically motivated the permAVG approach on the basis of hard permutation matrices, in practice we must resort to DSM matrices which allow for gradient-based optimization. This concession however, is by no means a limitation, as DSM matrices offer more flexibility than permutation matrices. In addition to the ability to shuffle rows, which both permutation matrices and DSM matrices share, DSM matrices have the additional ability to alter the values of the elements in the weight matrices. The tradeoff between row shuffling and value alteration in DSM matrices is governed by its sparsity or "softness". Harder DSM matrices have their mass in each row and column concentrated on few elements and are thus close to true permutation matrices, where all mass is concentrated on a single element. In this sense, harder DSM matrices are more limited to row shuffling. Softer DSM matrices on the other hand, have their row and column mass distributed across many elements and therefore are able to alter the values of individual weights.

In order to support our hypothesis that individual submodels only differ in the specific permutation of their weights, we measure the sparsity of the learned DSM matrices. If the learned DSM matrices are sparse, then they are in fact close to true permutation matrices and our claim is supported. If the learned DSM matrices are soft, then simply permuting the weights of the submodels was not sufficient and our claim is weakened. We measure sparsity by measuring the average uniformity of the rows. In the case of a permutation matrix, only one row element has value 1 while all other elements have value 0. In this sense we are maximally distant from a uniform distribution. On the other hand, if a row forms a uniform distribution, then we have maximum uniformity. Hence, for each DSM matrix we calculate the mean normalized entropy of the rows. As entropy is maximized in the case of a uniform distribution [2] and minimized by a distribution in which all probability mass is centered on one element, the mean normalized entropy will be 0 in the case of a permutation matrix and 1 in the case of a DSM matrix with uniformly distributed rows. In other words, the larger the mean normalized entropy, the *softer* the DSM matrix.

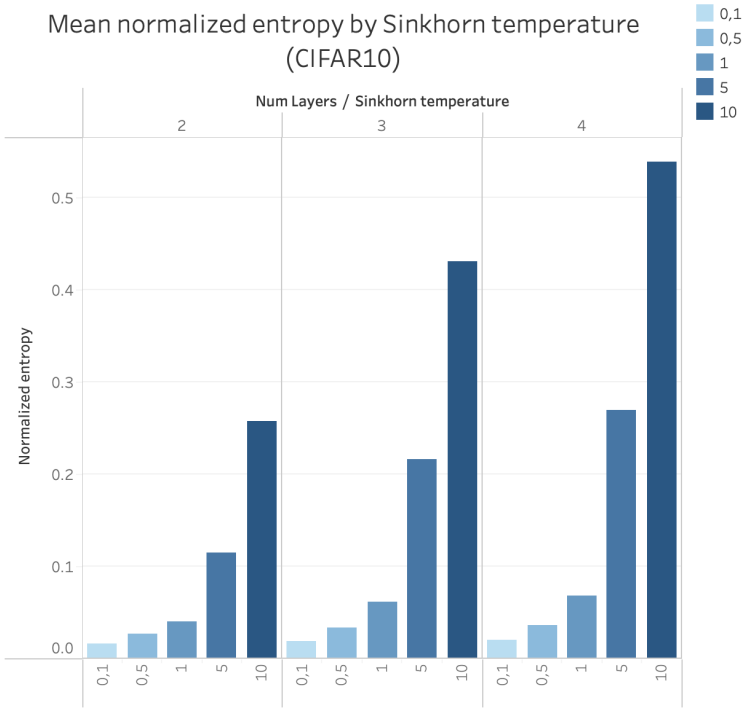

Figure 6: **Mean normalized entropy for the learned DSM matrices by different number of layers and Sinkhorn temperatures.** A score of 0 indicates a permutation matrix, while a score of 1 indicates a DSM matrix with uniformly distributed rows.

In Fig. 6 the mean normalized entropy scores are given for different number of layers and Sinkhorn temperatures (τ). There are three important observations. Firstly, the normalized entropy increases with τ. This is as expected, as when $\tau \to 0$ the DSM matrix approaches a permutation matrix. Secondly, as the number of layers increases, the normalized entropy increases. This indicates, that as the complexity of the optimization process increases, the more difficult it is to align weights simply through row shuffling. Lastly, for $\tau = 1$, the normalized entropy is around 0.06, which is very close to 0. Hence, the DSM matrices we have learned and which have led to significant accuracy increases, are very close to hard permutation matrices.

In addition to the normalized entropy scores, we visualize three different learned DSM matrices for $\tau = 0.1, 1, 10$ in Fig. 7. It can be seen that the *learning* of $\tau = 0.1$ is quite limited. Based on the fact that all cost matrices are initialized with the Identity matrix and that $\tau = 0.1$ still has significant mass on the diagonal post training, indicates that it has not changed significantly relative to initialization. On the other hand $\tau = 1, 10$ deviate completely from initialization and appear to learn permutations. However, it is clear that $\tau = 1$ is closer to an actual permutation matrix, as one can see far more mass concentration on individual elements.

7.5 Exploring the τ hyperparameter

The permAVG approach contains a hyperameter τ, which controls the softness of the DSM matrices. As seen in section 7.4, we have seen that large τ such as 5 and 10 lead to DSM matrices that are significantly

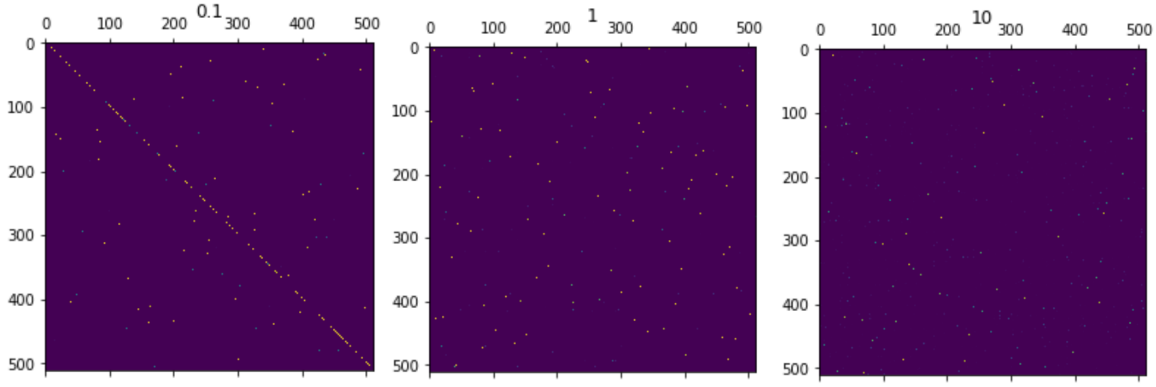

Figure 7: **Visualization of the learned DSM matrices for varying levels of "softness"** ($\tau = 0.1, 1, 10$)**.** For $\tau = 0.1$, the DSM matrices are unable to meaningfully deviate from initialization, which is the Identity matrix. For $\tau = 1$ there is clear concentration of mass on individual elements, which resembles a hard permutation matrix. $\tau = 10$ has significantly less mass concentration than $\tau = 1$ and therefore deviates more from a hard permutation matrix.

more sparse than $\tau \leq 1$ and that $\tau \leq 1$ leads to DSM matrices which are very close to hard permutation matrices. In order to investigate the impact of DSM softness on performance, we calculate the accuracy of several permAVG models with different τ (0.1,0.5,1,5,10). The results are depicted for the CIFAR10 [26] dataset in Fig. 8. Interestingly, the maximum accuracy is attained at $\tau = 1$. This is interesting, as one may expect softer DSM matrices to perform better, as they have greater flexibility. However, it appears as if DSM matrices that are closer to true permutation matrices are more performant. This finding supports our theory that submodels only differ in a single permutation. It should be noted however, that very small τ such as 0.1 perform the worst and therefore that strict permutation matrices would not perform well.

7.6 Sequential optimization

We have seen in section 7.1 that the permAVG method is unable to scale with ensemble size. One theory regarding the cause of this scalability bottleneck, could be the complexity of the optimization process. For example, the 6-submodel, 4-layer permAVG requires learning 15 DSM matrices simultaenously. In order to reduce optimization complexity, we propose an adapted optimization procedure in which the DSM matrices are learned sequentially. The process works as follows: We initially learn a permAVG model consisting of two submodels. We then freeze the parameters of the learned DSM matrices, add a new submodel and then learn the DSM matrices only of the new submodel. We continue the procedure until we reach the desired ensemble size.

The results on both CIFAR10 [26] and FMNIST [44] can be seen in Fig. 9. Unfortunately, the sequential optimization procedure is unable to significantly improve performance relative to the 2-submodel permAVG. In the CIFAR10 [26] dataset the procedure can only slightly increase accuracy by 0.12% in the 2-layer case and cannot improve accuracy in the 3 and 4-layer cases. In FMNIST [44], accuracy is increased 0.16% and 0.17% for 2 and 3-layers respectively and is unable to improve accuracy in the 4-layer case. Such mild accuracy increases are most likely not worth the additional computational requirements of such a sequential approach. In conclusion, we find that the sequential approach is unable to significantly improve accuracy. This raises doubts that the complexity of the optimization process is the true source of the performance bottleneck.

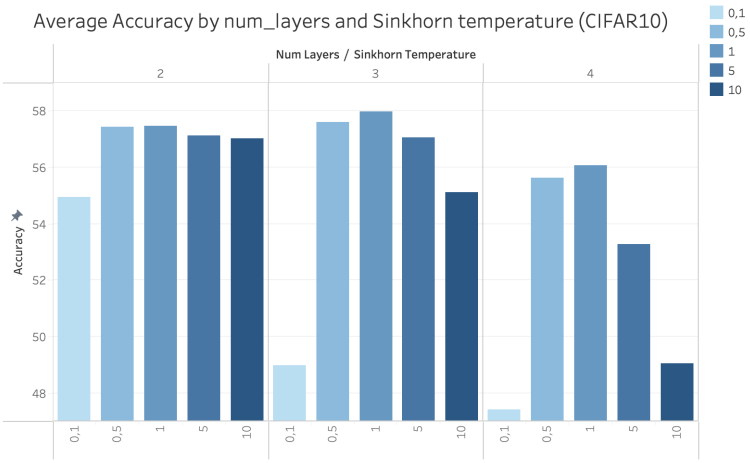

Figure 8: **Accuracy of different number of layers and Sinkhorn temperatures τ on CIFAR10**. Highest accuracy is attained at $\tau = 1$, in which the DSM matrices are close to hard permutation matrices (Fig. 6, 7). Interestingly, increasing the flexibility of the DSM matrices ($\tau = 5, 10$) does not result in higher accuracy. This suggests that optimal DSM matrices primarily focus on shuffling the rows of the weight matrices, rather than changing the values of the weights. However, solely row shuffling does not seem to be optimal either, as the hardest DSM matrices ($\tau = 0.1$), attain the lowest accuracy.

8. Conclusion

In conclusion, we have proposed a novel ensembling method to overcome the computational constraints of deep ensembles and therefore be more applicable in resource constrained environments. Whereas deep ensembles are ensembled in function space, we proposed ensembling neural networks in parameter space. However, whereas linear models can be ensembled in this manner, neural networks cannot, as each independently trained network finds a distinct mode in the loss landscape. Naively averaging over networks in different modes leads to poor results, as the average model most likely does not lie in a valley of the loss landscape. In order to overcome this drawback, we proposed learning permutations such that all networks can be permuted into the same mode. Thereafter, emulating Bayesian approximations and weight-averaging procedures such as SWA, the networks can simply be averaged in weight space. However, as opposed to such methods which exhibit strong inter-model correlations due to the dependence on a single training trajectory and solution, the permAVG approach averages over independently trained and therefore far more diverse models. To theoretically justify why neural networks can be permuted into the same mode, we build upon the fact that the loss landscape is full of symmetries such as permutation invariance and hence different modes need not imply distinct functions. We hypothesized that all networks in a deep ensemble may in fact be estimations of the same underlying function and therefore only vary in the specific permutation of their weights.

Based on our experimental results, we find the permAVG approach to be a viable ensemble method and to be competitive to both deep ensembles and knowledge distillation. The 2-submodel permAVG is able to signficiantly increase accuracy relative to the baseline and Bayesian approximation methods and is able to match and occasionally outperform deep ensembles and knowledge distillation. In cases when training resources are limited and submodel training is costly, the 2-submodel permAVG may in some cases be the superior method. The greatest weakness of the permAVG approach relative to the deep ensemble

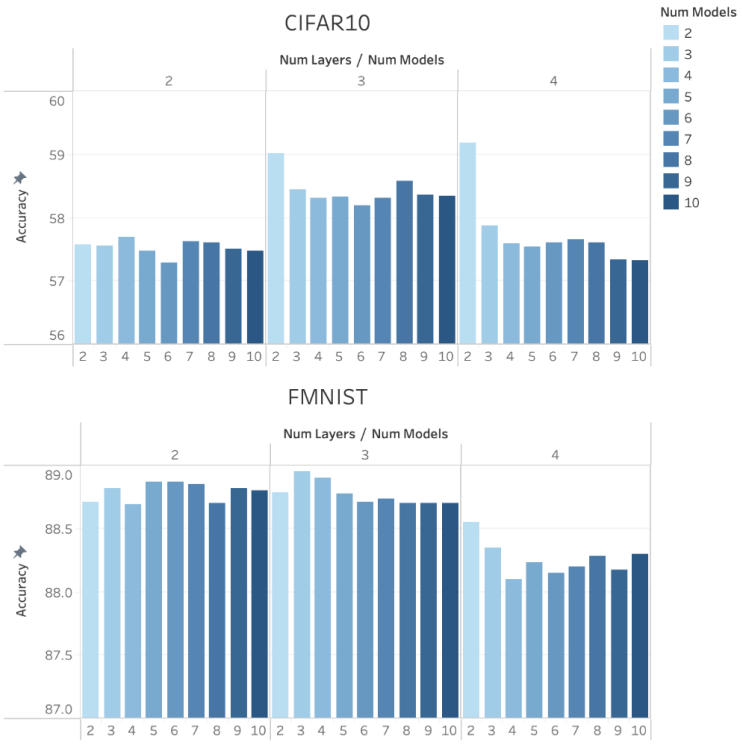

Figure 9: **permAVG accuracy results using a sequential optimization procedure on both CIFAR10 [26] and FMNIST [44]**. Although there are marginal improvements, such mild accuracy improvements do not warrant the additional computational costs required of such a sequential approach.

and knowledge distillation approaches is its inability to scale with ensemble size. We hypothesized that the scaling bottleneck may be a results of the complexity of the optimization problem and proposed a sequential optimization procedure as a solution. Unfortunately, the sequential procedure was unable to meaningfully increase accuracy.

Furthermore, we investigated the properties of the *learned* DSM matrices, to understand in what sense they are optimal. We saw that the optimal DSM matrices were learned with $\tau = 1$ and that they were very close to hard permutation matrices. This implies that the optimal DSM matrices primarily shuffle the rows of the weight matrices, rather than changing the values of the elements. Furthermore, we saw that the permAVG approach was aligning the rows of the weight matrices such that the values of the elements were closest. This was clear as the permuted weight matrices had lower mean distance, lower standard deviation and higher correlation. Lastly, we saw that the original weight matrices exemplified almost no correlation, which supports the statement that there is little correlation between the models in a deep ensemble.

Lastly, we summarize all evidence for and against our hypothesis that all networks can be permuted into the same mode, as they are approximations of the same underlying function. The performance of the 2-submodel permAVG suggests that there is merit to learning permutations and hence supports the possibility that two networks can be permuted into the same mode. Additionally, as our experiment in section 7.3 showed, the average at $\alpha = 0.5$ of the permuted submodels attained the highest accuracy and lowest total loss, while the average of the original submodels led to the lowest accuracy and highest total loss. This implies that in the former case, both submodels lie in the same mode and hence so does

their average, while in the latter case the average most likely does not lie in either mode of the original submodels. Lastly, we saw that the learned DSM matrices are in fact close to hard permutation matrices and hence that the optimal operation was dominantly row shuffling. This implies, that one is able to increase accuracy simply by shuffling the rows of the weight matrices prior to averaging, which is in line with our hypothesis. However, despite these positive findings, the inability of the permAVG model to scale past 2-submodels raises doubts as to whether one can simply permute all submodels to the same mode. This suggests that not all submodels in a large deep ensemble may be functionally equivalent and hence that functions may differ in more than just a permutation.

Future work: Due to the significant accuracy gains of the 2-submodel permAVG, a natural outstanding question is whether these results can be extrapolated to larger ensemble sizes. In this sense, the full potential of the permAVG method is unknown and unlike knowledge distillation its potential does not seem to be upper-bounded by that of the deep ensemble. Future research could investigate this scalability bottleneck and perhaps suggest alternative solutions. Additionally, such potential solutions may benefit from a deeper understanding of why the permAVG method works. Although, we have provided evidence for the fact that 2-submodels can be permuted into the same mode, it could be investigated whether this is really a result of the permutation symmetries in the loss landscape or whether the permutations are forced. For example, one could test the permAVG method on the following two different submodel scenarios: 1) Creating an initial submodel and calculating a second submodel by applying a known hard permutation to the initial submodel and perhaps adding some random noise. 2) Creating two submodels which are know to be functionally different. In the first case, one could analyse to what extent the permAVG method is able to uncover the true permutation. In the second case one could understand, whether being functionally equivalent is even necessary for the permAVG approach. In fact, if the permAVG method does not strongly rely on the fact that both models are functionally equivalent, then it could even have broader potential for application.

Lastly, one could investigate to what extent the permAVG approach can be extended to other neural network architectures beyond the multilayer perceptron. For example, is it possible to apply the permAVG approach to the convolutional filters of a CNN?

References

[1] Zeyuan Allen-Zhu and Yuanzhi Li. *Towards Understanding Ensemble, Knowledge Distillation and Self-Distillation in Deep Learning*. 2021. arXiv: 2012.09816 [cs.LG].

[2] Christopher M. Bishop. "Pattern Recognition and Machine Learning". In: Springer, 2006. Chap. 1,3,14.

[3] L. Breiman. "Arcing Classifiers". In: 1998.

[4] Leo Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140.

[5] Gavin Brown, Jeremy L. Wyatt, and Peter Tiňo. "Managing Diversity in Regression Ensembles". In: *J. Mach. Learn. Res.* 6 (Dec. 2005), pp. 1621–1650. ISSN: 1532-4435.

[6] Gavin Brown et al. "Diversity creation methods: a survey and categorisation". In: *Inf. Fusion* 6 (2005), pp. 5–20.

[7] Xu Cheng et al. "Explaining Knowledge Distillation by Quantifying the Knowledge". In: *CoRR* abs/2003.03622 (2020). arXiv: 2003.03622. URL: https://arxiv.org/abs/2003.03622.

[8] Jang Hyun Cho and Bharath Hariharan. "On the Efficacy of Knowledge Distillation". In: *CoRR* abs/1910.01348 (2019). arXiv: 1910.01348. URL: http://arxiv.org/abs/1910.01348.

[9] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[10] Thomas G. Dietterich. "Ensemble Methods in Machine Learning". In: *MULTIPLE CLASSIFIER SYSTEMS, LBCS-1857*. Springer, 2000, pp. 1–15.

[11] Carsten F. Dormann et al. "Model averaging in ecology: a review of Bayesian, information-theoretic, and tactical approaches for predictive inference". In: *Ecological Monographs* 88.4 (2018), pp. 485–504. DOI: https://doi.org/10.1002/ecm.1309. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.1002/ecm.1309. URL: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1002/ecm.1309.

[12] Nikita Durasov et al. "Masksembles for Uncertainty Estimation". In: *CoRR* abs/2012.08334 (2020). arXiv: 2012.08334. URL: https://arxiv.org/abs/2012.08334.

[13] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability 57. Boca Raton, Florida, USA: Chapman & Hall/CRC, 1993.

[14] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. *Deep Ensembles: A Loss Landscape Perspective*. 2020. arXiv: 1912.02757 [stat.ML].

[15] Yoav Freund and Robert E. Schapire. "A Short Introduction to Boosting". In: *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1999, pp. 1401–1406.

[16] Jianping Gou et al. "Knowledge Distillation: A Survey". In: *CoRR* abs/2006.05525 (2020). arXiv: 2006.05525. URL: https://arxiv.org/abs/2006.05525.

[17] Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. "Gradient Descent Happens in a Tiny Subspace". In: *CoRR* abs/1812.04754 (2018). arXiv: 1812.04754. URL: http://arxiv.org/abs/1812.04754.

[18] T. Hastie, R. Tibshirani, and J. H. Friedman. "The elements of statistical learning: data mining, inference, and prediction". In: 3rd ed. Springer, 2009. Chap. 8.7.

[19] Max Hinne et al. "A Conceptual Introduction to Bayesian Model Averaging". In: *Advances in Methods and Practices in Psychological Science* 3 (June 2020), p. 251524591989865. DOI: 10.1177/2515245919898657.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[21] Gao Huang et al. *Snapshot Ensembles: Train 1, get M for free*. 2017. arXiv: 1704.00109 [cs.LG].

[22] Pavel Izmailov et al. "Averaging Weights Leads to Wider Optima and Better Generalization". In: *CoRR* abs/1803.05407 (2018). arXiv: 1803.05407. URL: http://arxiv.org/abs/1803.05407.

[23] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML].

[24] Laurent Valentin Jospin et al. "Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users". In: *CoRR* abs/2007.06823 (2020). arXiv: 2007.06823. URL: https://arxiv.org/abs/2007.06823.

[25] Taehyeon Kim et al. "Comparing Kullback-Leibler Divergence and Mean Squared Error Loss in Knowledge Distillation". In: *CoRR* abs/2105.08919 (2021). arXiv: 2105.08919. URL: https://arxiv.org/abs/2105.08919.

[26] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[28] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. 2017. arXiv: 1612.01474 [stat.ML].

[29] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[30] Stefan Lee et al. "Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks". In: *CoRR* abs/1511.06314 (2015). arXiv: 1511.06314. URL: http://arxiv.org/abs/1511.06314.

[31] Wesley Maddox et al. "A Simple Baseline for Bayesian Uncertainty in Deep Learning". In: *CoRR* abs/1902.02476 (2019). arXiv: 1902.02476. URL: http://arxiv.org/abs/1902.02476.

[32] Gonzalo Mena et al. *Learning Latent Permutations with Gumbel-Sinkhorn Networks*. 2018. arXiv: 1802.08665 [stat.ML].

[33] Mary Phuong and Christoph H. Lampert. "Towards Understanding Knowledge Distillation". In: *CoRR* abs/2105.13093 (2021). arXiv: 2105.13093. URL: https://arxiv.org/abs/2105.13093.

[34] P. S. S. N. V. Prasada Rao. "On Generalized Inverses of Doubly Stochastic Matrices". In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 35.1 (1973), pp. 103–105. ISSN: 0581572X. URL: http://www.jstor.org/stable/25049855.

[35] Rodrigo Santa Cruz et al. "Visual Permutation Learning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.12 (2019), pp. 3100–3114. DOI: 10.1109/TPAMI.2018.2873701.

[36] Robert E. Schapire. "The strength of weak learnability". In: *Machine Learning* 5.2 (1990), pp. 197–227. ISSN: 0885-6125. DOI: 10.1023/A:1022648800760. URL: http://www.cs.princeton.edu/~schapire/papers/strengthofweak.pdf.

[37] Ronan Sicre et al. "Automatic discovery of discriminative parts as a quadratic assignment problem". In: *CoRR* abs/1611.04413 (2016). arXiv: 1611.04413. URL: http://arxiv.org/abs/1611.04413.

[38] Richard Sinkhorn. "A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices". In: *Annals of Mathematical Statistics* 35 (1964), pp. 876–879.

[39] Richard Sinkhorn and Paul Knopp. "Concerning nonnegative matrices and doubly stochastic matrices". In: *Pacific Journal of Mathematics* 21 (1967), pp. 343–348.

[40] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958. URL: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[41] R. Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society (Series B)* 58 (1996), pp. 267–288.

[42] N. Ueda and R. Nakano. "Generalization error of ensemble estimators". In: *Proceedings of International Conference on Neural Networks (ICNN'96)*. Vol. 1. 1996, 90–95 vol.1. DOI: 10.1109/ICNN.1996.548872.

[43] Geoffrey Webb and Zijian Zheng. "Multistrategy Ensemble Learning: Reducing Error by Combining Ensemble Learning Techniques". In: *Knowledge and Data Engineering, IEEE Transactions on* 16 (Sept. 2004), pp. 980–991. DOI: 10.1109/TKDE.2004.29.

[44] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

A. Additional visualizations

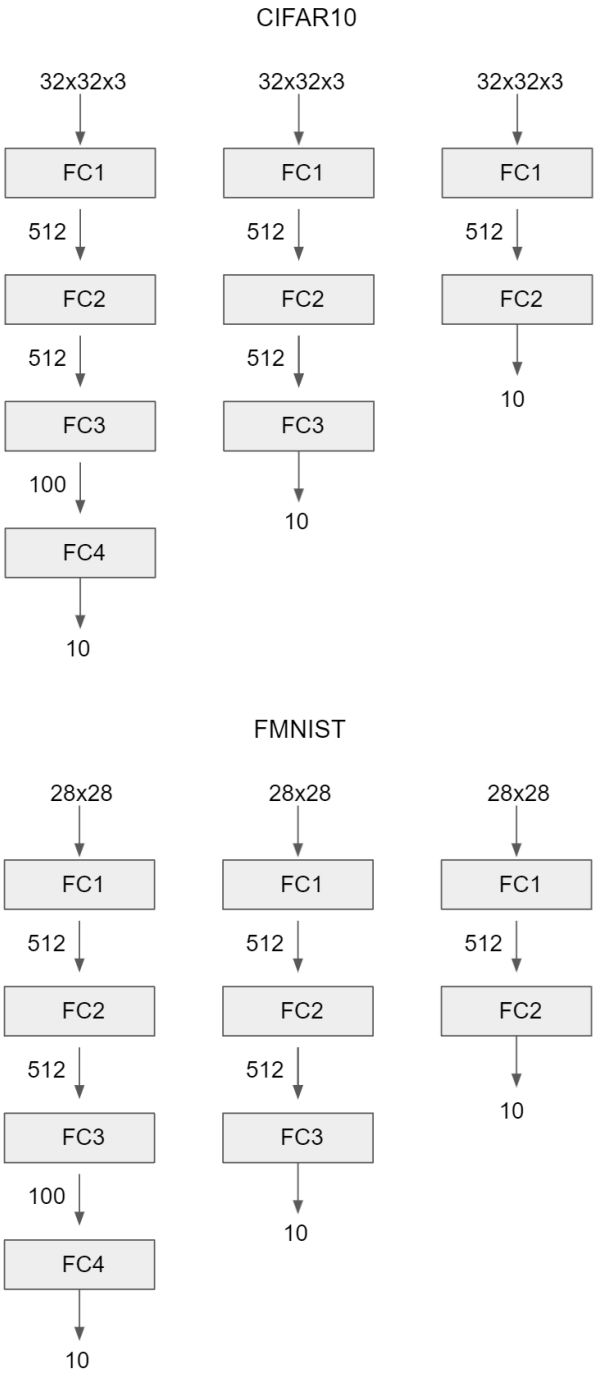

Figure 10: **Top:** The multilayer perceptron architectures for the submodels of 4,3 and 2 layers for the CIFAR10 [26] dataset. **Bottom:** The same repeated for the FMNIST [44] dataset.

CIFAR10

Num Layers	Num Models	DeepEns	permAVG	KD	submodels
2	2	55.90	57.55	55.26	53.22
	3	57.03	57.37	56.41	52.84
	4	58.42	57.61	56.97	53.40
	5	58.47	57.50	57.30	52.88
	6	58.92	57.56	57.92	53.01
3	2	59.31	59.06	59.13	56.45
	3	59.87	58.03	59.30	56.40
	4	61.02	57.61	59.50	56.14
	5	61.32	56.89	59.73	56.42
	6	61.72	56.58	60.10	56.34
4	2	59.63	58.71	58.68	56.34
	3	60.15	57.28	58.96	56.01
	4	61.03	55.91	59.69	56.36
	5	61.45	54.72	59.98	56.37
	6	61.74	53.26	60.07	56.17

FMNIST

Num Layers	Num Models	deepEns	permAVG	KD	submodels
2	2	88.23	88.77	87.65	86.92
	3	88.82	88.66	87.88	87.38
	4	88.87	88.52	88.04	87.25
	5	89.04	88.35	88.10	87.19
	6	89.19	88.28	88.28	87.29
3	2	88.39	89.09	86.76	87.07
	3	89.17	88.66	87.03	87.91
	4	89.05	88.56	85.56	87.19
	5	89.37	88.45	88.54	87.49
	6	89.14	88.12	87.65	87.05
4	2	88.19	88.17	88.10	86.87
	3	88.82	87.53	87.85	86.99
	4	88.88	86.06	87.44	86.68
	5	88.96	84.52	87.56	86.40
	6	89.13	81.65	87.85	86.64

Figure 11: **Left:** Average accuracy on CIFAR10 [26] for deepEnsemble (DENOTED BY deepEns), permAVG, KD and the submodels. **Right:** The same repeated for the FMNIST [44] dataset.