Sequentially training diverse deep ensemble members

Geoffrey Liu
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
geoffreyliu@fas.harvard.edu

1 Introduction

Ensembles of deep neural networks are becoming increasingly popular since they have shown improvements in accuracy and uncertainty over a single neural network. However, independent training of each ensemble member doesn't guarantee diverse predictions and the user has to rely on the randomness of initialization, mini-batching in stochastic gradient descent, and the number of local minima in the loss surface to achieve diverse ensemble members. Moreover, independently training deep neural networks in an ensemble often leads to members with very similar functional forms, this leads to minimal improvements in the quality of the ensemble prediction and uncertainty estimates.

Recent work has introduced interesting ways of forcing diversity between ensemble members through the addition of a *repulsive force* or *diversity term* into the neural-network loss function during training. This is a novel method to *enforce* functional diversity between ensemble members during the training process. Most of these diverse ensemble methods use joint or end-to-end training which leads to issues of dominance (one ensemble member is much better than the rest), over-reliance (ensemble performance degrades when members are removed, or failure to converge (Webb et al., 2019).

In this project, I propose a new way of maintaining functional diversity during the training of deep neural network ensembles, which to my knowledge, has not been investigated before. I propose to sequentially train ensemble members, with each member forced to be different to each previous ensemble member through including a repulsive force into the training procedure. We analyze the performance of our ensembling method on different regression and classification tasks using synthetic data. We also compare the performance of our ensembling method on image classification tasks using MNIST and FashionMNIST.

2 Background and Literature Review

In this section, I provide a background into deep ensembles and frame where the project lies within the current literature.

2.1 Deep Ensembles

Ensembles where multiple deep neural networks are trained individually and independently (deep ensembles), and their predictions averaged, have been widely successful for improving both the accuracy and predictive uncertainty of single neural networks. Lakshminarayanan et al. (2017) are often referenced as the baseline for ensembles of deep neural networks. They show that a simple collection of independently trained neural networks with the same architecture can achieve higher accuracy and reasonable uncertainty quantification compared to a single deterministic network. These ensembles have become ubiquitous, both in practice and the academic literature, because of their conceptual simplicity and ease of implementation. They provide a quick and simple boost in accuracy over a single network when required, for example, state-of-the-art benchmark results are

Preprint. Under review.

often achieved by ensembles of neural networks. At the time of writing, the top performing models ImageNet (Wortsman et al., 2022) uses ensembles to achieve their highest score.

Deep ensembles also provide good uncertainty quantification and are still widely used as the benchmark to beat (Nado et al., 2021). One reason is that deep ensembles provide an effective mechanism for approximate Bayesian model averaging over different '*basins of attraction*', in other terms, different posterior modes or local optima (Wilson, Izmailov, 2020). Despite this view, independently trained deep ensemble members often collapse into the same or very similar functional form into training, which may negatively affect the uncertainty estimates of the ensemble and therefore its performance for uncertainty quantification.

The uncertainty quantification for deep ensembles relies predominately on:

- the randomness in weight initialization.
- the noise in stochastic gradient descent from random mini-batching.
- the number of local optima in the loss surface of the model.

2.1.1 Adding more randomness

To add more diversity into deep ensemble training, different methods have been proposed that focus mainly on adding randomness into the other parts of the neural network training process not mentioned in the list above. Wenzel et al. (2020) propose hyperparameter ensembles, where randomness is introduced into the hyperparameter used for training each ensemble member, specifically, the dropout rate and weight placed on L_2 weight regularization. Including this additional source of randomization is beneficial to ensemble training.

Recently, Grewal, Bui (2021) proposed another source of randomization by training deep ensemble members with different optimization algorithms, and different network architectures (e.g. layer sizes, activation functions, etc). This can be viewed as an extension of hyperparameter ensembles by considering the neural architecture and optimization algorithm as a part of the neural network hyperparmeters. Different optimization algorithms such as Adam and SGD have implicit biases towards basins with different types of geometric properties in the loss surface. For example, SGD is biased towards flat basins while Adam is biased towards sharper basins in the loss surface (Zhou et al., 2020). Using different optimization algorithm to train each member allows for more efficient exploration of different types of local optima in the loss surface, that deep ensembles already exploit. The proposal to include different model classes in an ensemble exploits the different types of each neural architecture and thus may also increases the functional diversity of the ensemble itself.

2.2 Repulsive ensembles

So far, we observe that deep ensembles rely exclusively on randomness of the training process, or introducing randomness to a set of hyperparmeters used to train each ensemble member. This is to try and find different local optima in the posterior or loss surface, thus increasing the overall diversity of the ensemble. Instead of relying on randomness to achieve this affect, a natural idea is to explore the possibility of explicitly engineering a procedure that forces each ensemble member into a different regions of the posterior.

Recent work into deep ensembles introduced the idea of adding a *repulsive force* between ensemble members to promote diversity between the trained networks. The repulsive force is added as a term into the loss function, thus forcing each ensemble member to balance fitting the data and maintaining diversity compared to the other members of ensemble. There are two ways of characterizing the repulsive force between neural networks. The first is to place a repulsive force on the model parameters i.e. *parameter space* or *weight space* repulsion. This penalizes a ensemble member when its model parameters (or weights) are close to that of another ensemble member. The second is to place a repulsive force on the outputs of the neural networks i.e. *function space* repulsion. This penalizes one ensemble member when the outputs of the member are close to that of another ensemble member, thus leading to function space diversity across the ensemble members.

Although function space repulsion and weight space repulsion are equivalent (through some complex non-linear mapping), this distinction is important in practice. Due to overparameterization of deep neural networks, models with very different weights may map to the same function thus forcing the

weights between ensemble members to differ does not guarantee a functionally diverse ensemble (D'Angelo, Fortuin, 2021).

Different authors have proposed different ways of characterizing the repulsive force between ensemble members. D'Angelo, Fortuin (2021) use a kernelized repulsive term, specifically the RBF kernel, in the gradient descent update to encourage diversity between members. This update rule was inspired by the update rule by the Stein Variational Gradient Descent (SVGD) (Liu, Wang, 2016), which is a variational inference algorithm that uses a set of n particles to approximate the bayesian posterior, where the true posterior is recovered as $n \to \infty$. Loosely speaking, SVGD can be viewed as using n evenly weighted dirac-delta distributions to approximate the posterior, where the location of each of distribution is optimized such that the ensemble minimizes the Kullback-Leibler divergence the posterior. By framing the deep ensemble as the set of particles, D'Angelo, Fortuin (2021) use this to show that their update rule leads to an ensemble that is a valid approximation the posterior of the Bayesian Neural Network, in the sense that this ensembles approximates the BNN posterior as the number of ensemble members grow.

The loss function used to train the ensemble inD'Angelo, Fortuin (2021) is given by,

$$\mathcal{L} = \underbrace{\sum_{j=1}^{M} \log \pi(f_j \mid D)}_{\text{Loss}} + \underbrace{\frac{1}{M} \sum_{j=1}^{M} \sum_{k=j}^{M} k(f_j, f_k)}_{\text{Repulsive Force}}, \tag{1}$$

where f_j is the outputs jth ensemble member, $\pi(f_j \mid D)$ is the posterior of the jth ensemble member, k is a radial basis function kernel.

Webb et al. (2019) characterize the repulsive force between ensemble members in a different manner. Instead of viewing each ensemble member on its own, they study ensemble training in an end-to-end framework, treating networks in the ensemble jointly as components of a single larger network. The authors formulate a loss function that allows the user to control the balance between optimizing for loss of each individual ensemble members and the function space diversity between the ensemble members.

The loss function used to train the ensemble in Webb et al. (2019) is given by,

$$\begin{aligned} \mathcal{L}_\lambda &= \lambda \sum_{j=1}^{M} KL(p \| \bar{q}) + (1-\lambda) \frac{1}{M} \sum_{j=1}^{M} KL(f \| g_j) \\ &= \underbrace{\frac{1}{M} \sum_{j=1}^{M} KL(p \| q_j)}_{\text{Individual Loss}} + \underbrace{\frac{\lambda}{M} \sum_{j=1}^{M} -KL(\bar{q} \| q_j)}_{\text{Repulsive Force}}, \end{aligned} \tag{2}$$

where $KL(p\|q_j)$ is the KL divergence between the predictions of ensemble member j and the target distribution (i.e. labels), q_j is the softmax probabilities of ensemble member j, and $KL(\bar{q}\|q_j)$ is the KL divergence between q_j and the average ensemble prediction $\bar{q}$.

These two approaches to diverse ensemble training use slightly different methods to characterize repulsion between ensemble members, Webb et al. (2019) chooses to use the negative Kullback-Leibler divergence while D'Angelo, Fortuin (2021) uses the radial basis function kernel to characterize the repulsive force. Although both suggest placing the repulsive force in function space, Webb et al. (2019) considers the case where each member is repelled from the ensemble mean, while D'Angelo, Fortuin (2021) choose to repel each member from every other member of the ensemble. Furthermore, D'Angelo, Fortuin (2021)'s approach considers training in a fully Bayesian manner, where in the gradient update step, the gradient of the implicit functional prior is also calculated.

Both approaches, however, train the ensemble jointly i.e. each ensemble member is trained together with all other ensemble members, and the number of ensembles is pre-specified before the training process. A question that both authors leave for future work is to investigate what happens if the networks are trained sequentially i.e. the first network is trained and fixed, then a second network is trained with a repulsive force against all previous network, and so on.

3 Our approach to ensembling

3.1 Sequentially training diverse ensembles

In this section, I present our method for training diverse deep ensembles. In my formulation, I use three alterations from previous methods for constructing deep ensembles, namely,

- Sequential training of ensembles
- Training deterministically and using different repulsive forces
- Adding in a parameter to control the weight of the repulsive force

Incorporating these three elements into the training of our ensemble, we loss function for training our ensemble is,

$$\mathcal{L}_{\lambda}^{j} = \underbrace{\lambda\, \ell(p\|q_j)}_{\text{Loss}} \;+\; \underbrace{(1-\lambda)\frac{1}{j}\sum_{k=1}^{j} -D(q_j, q_k)}_{\text{Repulsive Force}}\,, \tag{3}$$

where $\mathcal{L}^j$ represents the loss function for the jth ensemble member, $\ell(p\|q_j)$ is a loss function between the target distribution (data), p and the output from the jth ensemble member, q_j, and d is a function, not necessarily symmetric, that scores the distance between ensemble members. The parameter λ controls the weight placed on fitting the data and the weight placed on functional diversity compared to the previous ensemble members.

3.2 Explanation of modelling choices

The first choice training process is to consider the sequential training of deep ensemble members instead of joint training of ensembles. This has two main advantages, the first addresses the issues faced by Webb et al. (2019); the joint training of ensembles according to Equation 2 can lead to over-reliance on ensemble averaging, that is, the rapid degradation of the ensemble performance as individual ensemble members are dropped. Deep ensembles are robust to this phenomenon, since each ensemble member is independently and thus removing member from the ensemble impacts the ensemble average minimally. The second advantage to sequential training is that the number of ensemble members does not need to be pre-specified. Thus, the number of ensemble members can be increased or decreased during the training process according to the users needs. This may be useful, especially when the number of ensemble members needed to reach performance saturation is not known in advance. On the other hand, choosing to train the ensemble in a sequential manner limits the ability to parallelize the training process as each ensemble member depends on the previous members. However, this is a minor issue and other forms of deep ensemble methods such as hyper-deep ensembles also incur this limitation.

The second choice is to train the ensemble deterministically, in the spirit of deep ensembles as discussed in subsection 2.1, as opposed to approximating the Bayesian Neural Network Posterior as in D'Angelo, Fortuin (2021). Although, we allow for the use of different types of repulsive forces, such as a kernelized repulsive force, KL divergence, or even simple cost functions such as the $\mathcal{L}_1$ or $\mathcal{L}_2$ cost function.

The third choice is incorporate a weight λ that balances the fitting of the data and the repulsive force or functional diversity between ensemble members, allowing for a continuous transition between independently trained deep ensembles ($\lambda = 1$) and strongly repulsive deep ensembles ($\lambda = \epsilon$).

The differences between Equations1, 2 and 3. The repulsive force in my method is applied to all ensemble members as in Equation 1 as opposed to the ensemble mean. The tradeoff between the repulsive force and the loss function is explicit as in Equation 2 allowing for easier tuning of the weight placed on the repulsive force. In Equation 1, the weight placed on the repulsive force is implicit as during the training process the lengthscale of the RBF kernel changes using a heuristic based on the median pairwise distance between ensemble members.

4 Experiments

In this section, we will detail some of the experiments that we run using our new ensembling method. We use independently trained deep ensembles, that only differ by initial weight initialization and random mini-batching to train the ensemble

4.1 Synthetic data

4.1.1 2-D Classification

To test our ensembling method on classification, we generated a simple synthetic classification dataset with 5 classes. These classes generated using 5 2-dimensional isotropic Gaussian distributions with a variance of 2, equally spaced apart on a ring of radius 5 around the origin. The dataset used for training was generated by combining 50 samples from each of the 5 Gaussian distributions.

We trained 100 ensemble members using our ensembling method and 100 ensemble members from a deep ensemble for comparison. Each ensemble member was a standard multi-layer perceptron with 2 hidden layers of width 50 plus a bias term, and ReLU were used for all activations in the network. The Adam optimizer with learning rate set to 0.001 was used for training for all ensembles methods.

Our ensemble was trained with the Hellinger distance as the repulsive force, where the repulsive force is placed on the predicted probabilities after softmax. which is given by Equation 4. This distance is often viewed as a probabilistic analogue to the euclidean distance.

$$H(p,q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2} \tag{4}$$

In our classification experiments, it appears that the Hellinger distance performed well however other distance functions such as the L_2 cost, RBF Kernel could be used instead. Furthermore, another choice is whether to apply the repulsive force to logits directly or to the softmax probabilities.

Figure 1 shows the results of our ensemble method versus a standard deep ensemble. This plot displays the log-entropy of the ensemble mean, taken before softmax, over the classes. In this plot, we observe that we are able to use this metric to identify the in and out of distribution areas in this simple case. We can see that the areas where this metric is larger (red) corresponds to the areas where we the data do not lie.

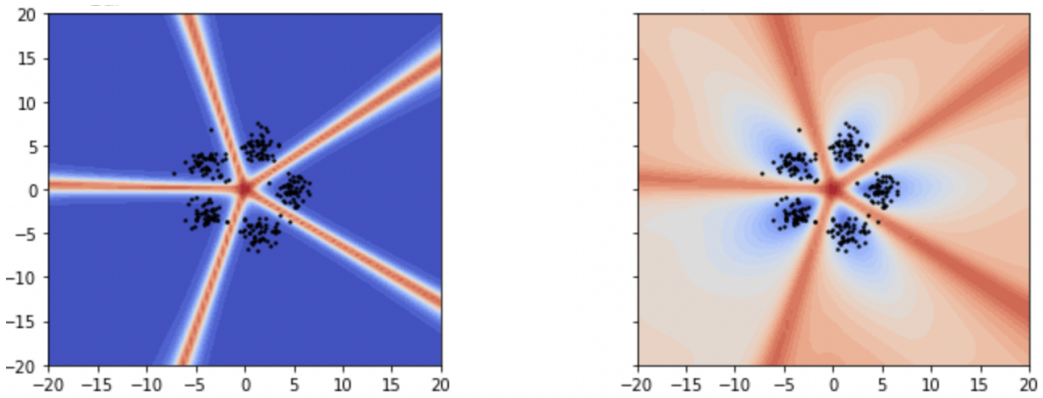

Figure 1: Out of distribution detection on 2-D classification task using log-entropy (blue=low, red=high). The standard deep ensemble is on the left and our ensemble is on the right

4.1.2 1-D Regression

In the regression case, we assessed our ensembling method of fitting a simple synthetic regression dataset. First, 50 x values were uniform randomly generated on the interval $[1, 2]$ and another 50 on the interval $[4.5, 6]$. Then the targets were generated using $y = x\sin(x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.2)$.

Our ensemble was trained with the RBF Kernel as the repulsive force, with lengthscale set to be equal to 1. We trained 50 ensemble members using our ensembling method and 50 ensemble members from a deep ensemble for comparison. Each ensemble member was a standard multi-layer perceptron with 2 hidden layers of width 50 plus a bias term, and LeakyReLU were used for all activations in the network. The Adam optimizer with learning rate set to 0.001 was used for training for all ensembles methods.

These results reported in Figure 2 consisting of the mean and percentiles of the predictions from the ensemble, with different specifications of the repulsive force. The target function is coloured in red, and the datapoints used for training are displayed in red. The ensemble mean is given by the black line and the percentiles of the ensemble predictions is given by the blue shaded areas. The lightest shade of blue shows the (5, 95) percentiles and the darkest shade of blue represents the (45, 55) percentiles of the ensemble predictions.

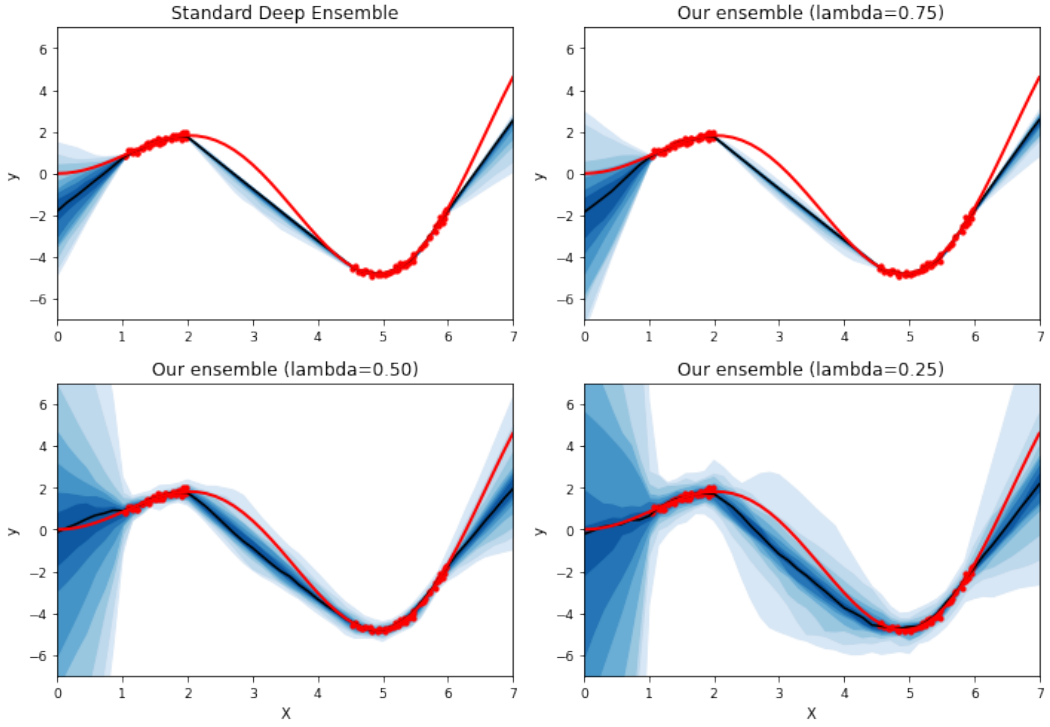

Figure 2: Predictive means and uncertainties from our ensembling method versus standard deep ensembles on a synthetic 1-D regression dataset.

We observe that as the the weight placed on the repulsive force increases ($\lambda \searrow$) the predictive uncertainties of the ensemble increases. The standard deep ensemble fails to capture the *in-between* uncertainty of the gap between the datapoints, while our ensembling method does when λ is sufficiently low. By looking at the plots for the different values of λ, we can observe that our ensembling method provides greater functional diversity, as measured by the percentiles of the ensemble predictions, compared to the standard deep ensemble. Morover, with $\lambda = 0.5$ and $\lambda = 0.25$, our ensembling method captures true function within the 90% prediction interval of the ensemble. Our ensemble trained with a weak repulsive force, $\lambda = 0.75$, and the standard deep ensemble fail to capture the true function, although the repulsive force does appear to still increase functional diversity as the prediction intervals in the tails are wider.

Before we move on to the next experiment, it is important to note that a kernelized repulsive force was the only specification that worked in the regression case. Other forms of distance functions such as the KL-divergence were ill-defined (unless some distributional form over the predictions was assumed). Using simple cost functions, such as the L_1 or L_2 cost as a repulsive force did not provide meaningful results as these distance metrics are unbounded, thus each ensemble member after the first diverged to $\pm\infty$.

4.2 MNIST and FashionMNIST

Next, we investigate the performance of our method on simple image classification tasks. The results are reported in Table 1. To train our model, we used the Hellinger distance as the repulsive force on the softmax probabilities using $\lambda = 0.5$ as the weight for the repulsive force. We trained 10 ensemble members using our ensembling method and 10 standard deep ensemble members for comparison. We observe that our training procedure yields the lowest expected calibration error (ECE) across our different experiments.

For the model architecture, we used a MLP with 3 hidden layers of width 300, 200, and 100 respectively and ReLU were used for all activations in the network. The Adam optimizer with learning rate set to 0.001 was used for the optimization procedure.

The FashionMNIST experiments were performed in a similar manner to the MNIST experiments, the key difference being that we included an extra convolutional neural network architecture for comparison. The convolutional neural network was a standard LeNet-5, three convolutional layers with subsampling passed into 2 fully connected layers.[1]

Dataset	Ensemble	Accuracy	ECE	MCE	Brier	NLL
MNIST (MLP)	Deep Ensemble	**0.984**	0.0053	0.201	**0.00240**	**0.0489**
	Our Ensemble	0.980	**0.0027**	**0.192**	0.00310	0.1242
Corrupted MNIST	Deep Ensemble	0.736	0.1995	**0.343**	0.04489	1.4644
	Our Ensemble	**0.761**	**0.1469**	0.349	**0.03781**	**0.9184**
FashionMNIST (MLP)	Deep Ensemble	0.908	0.0939	0.383	0.01502	**0.3150**
	Our Ensemble	**0.909**	**0.0730**	**0.262**	**0.01470**	0.3508
FashionMNIST (CNN)	Deep Ensemble	**0.931**	0.0701	0.400	0.01120	**0.2125**
	Our Ensemble	0.928	**0.0185**	**0.255**	**0.01081**	0.4059

Table 1: Comparison of deep ensembles and our ensemble on image classification using MNIST and FashionMNIST

4.3 Out of distribution (OOD) detection on image classifiers

Table 2 and Table 3 shows the entropy of the mean ensemble predictions trained on MNIST and using rotated MNIST, where each image on the test batch of MNIST are rotated a fixed angle counterclockwise, SVHN, and FashionMNIST as the OOD datasets.

Table 2 shows that the entropy increases at a much faster rate when the ensemble is trained using our method for increasing rotations of MNIST. Although, the entropy on the uncontaminated test set is also slightly higher for our ensembling method versus a deep ensemble. Table 3 also confirm that our ensemble has higher entropies using SVHN and F-MNIST as OOD datasets.

Figure 3 shows the histogram of entropies instead of the average entropy over the whole OOD dataset. We see that our ensemble method has a smaller peak around 0 entropy for the OOD data, which indicates that our ensemble may perform better on OOD detection compared to a standard deep ensemble.

5 Future work

There are many experiments that are left as future work, for example, we do not believe that our experiments on the OOD abilities of our ensemble method are exhaustive, and many more metrics

[1] https://github.com/google/uncertainty-baselines/tree/main/baselines/mnist

Metric	Ensemble	MNIST	MNIST (15^o)	MNIST (30^o)	MNIST (45^o)	MNIST (60^o)
Entropy	Deep Ensemble	0.0267	0.0706	0.2088	0.3831	0.4780
	Our Ensemble	0.0329	0.0871	0.2653	0.4922	0.6236

Table 2: Comparison of average ensemble entropies using rotated MNIST as the OOD dataset.

Metric	Ensemble	MNIST	SVHN	F-MNIST
Entropy	Deep Ensemble	0.0267	0.5025	0.3616
	Our Ensemble	0.0329	0.7882	0.4710

Table 3: Comparison of average ensemble entropies using SVHN and FashionMNIST as the OOD dataset.

could be tested for OOD detection, however, we leave this as future work. Furthermore, for our image and OOD detection tasks we only considered the case where the repulsive force weight $\lambda = 0.5$. It is unknown at this point in time how our ensemble method varies with different values of λ, or whether optimizing different values of λ will yield better results.

Although the results ensembling framework are interesting, our experiments mostly consider toy data and small image datasets. It is unknown whether or not the properties of our ensembling method extends to larger networks and data. One simple way to extend these experiments is to train our ensemble on a more complex image classification task such as CIFAR-10/100.

6 Conclusion

In this project, we propose a new method for training diverse deep ensemble members and show that this framework displays some promising trends and results on synthetic data and small image classification tasks. Our method indeed increases the functional diversity of the ensemble compared to standard deep ensemble training, however more empirical analysis is needed to determine the quality of our approach versus other ensemble methods that have improved upon deep ensembles.

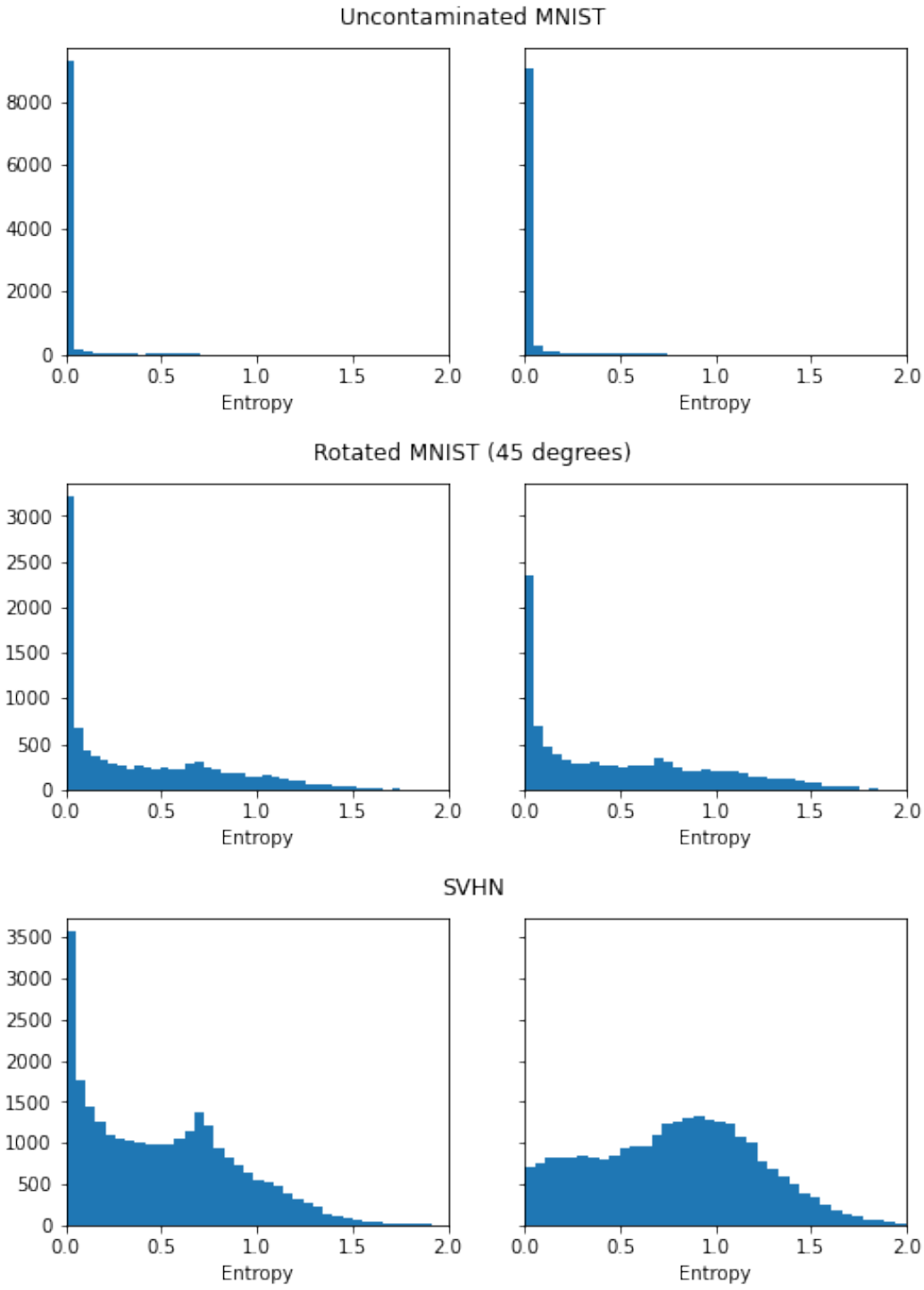

Figure 3: Comparison of the histograms of entropies on rotated MNIST with 45^o rotation and SVHN.

References

D'Angelo Francesco, Fortuin Vincent. Repulsive Deep Ensembles are Bayesian // Advances in Neural Information Processing Systems. 2021.

Grewal Yashvir, Bui Thang D. Diversity is All You Need to Improve Bayesian Model Averaging. 2021.

Lakshminarayanan Balaji, Pritzel Alexander, Blundell Charles. Simple and scalable predictive uncertainty estimation using deep ensembles // Advances in neural information processing systems. 2017. 30.

Liu Qiang, Wang Dilin. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. 2016.

Nado Zachary, Band Neil, Collier Mark, Djolonga Josip, Dusenberry Michael, Farquhar Sebastian, Filos Angelos, Havasi Marton, Jenatton Rodolphe, Jerfel Ghassen, Liu Jeremiah, Mariet Zelda, Nixon Jeremy, Padhy Shreyas, Ren Jie, Rudner Tim, Wen Yeming, Wenzel Florian, Murphy Kevin, Sculley D., Lakshminarayanan Balaji, Snoek Jasper, Gal Yarin, Tran Dustin. Uncertainty Baselines: Benchmarks for Uncertainty & Robustness in Deep Learning // arXiv preprint arXiv:2106.04015. 2021.

Webb Andrew M., Reynolds Charles, Chen Wenlin, Reeve Henry, Iliescu Dan-Andrei, Lujan Mikel, Brown Gavin. To Ensemble or Not Ensemble: When does End-To-End Training Fail? 2019.

Wenzel Florian, Snoek Jasper, Tran Dustin, Jenatton Rodolphe. Hyperparameter Ensembles for Robustness and Uncertainty Quantification // Advances in Neural Information Processing Systems. 33. 2020. 6514–6527.

Wilson Andrew G, Izmailov Pavel. Bayesian Deep Learning and a Probabilistic Perspective of Generalization // Advances in Neural Information Processing Systems. 33. 2020. 4697–4708.

Wortsman Mitchell, Ilharco Gabriel, Gadre Samir Yitzhak, Roelofs Rebecca, Gontijo-Lopes Raphael, Morcos Ari S., Namkoong Hongseok, Farhadi Ali, Carmon Yair, Kornblith Simon, Schmidt Ludwig. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. 2022.

Zhou Pan, Feng Jiashi, Ma Chao, Xiong Caiming, Hoi Steven, E Weinan. Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning. 2020.

A Appendix

A.1 Overlaps with current research

This project had no significant overlaps with ongoing research. I am doing a project on variational inference and some research on Neural Linear Models with WeiWei, however none of these have significant overlaps with what this project.

A.2 Skills developed

One important skill that I developed throughout this project were the ability to understand and read the literature on deep ensembles. This was challenging at first since there were many metrics and different ways of training ensembles, and their connections to other types of models e.g. SNGP. The second important skill that I developed was to code in Jax + Haiku. I found this challenging at first since Jax jitted functions need to be true functions unlike normal python functions, keeping track of random number generator keys vs. setting one seed and being done, keeping track of all the states for ResNet training, and also learning how to manually write the L2 weight regularization while skipping the batchnorm layer weights. There were alot of headaches however after this project I would say that I am more comfortable with Jax, and more capabable of using Jax to implement neural network architectures.