Joint Training of Neural Network Ensembles

Andrew M. Webb[1], Charles Reynolds[1], Dan-Andrei Iliescu[2], Henry Reeve[3], Mikel Luján[1], and Gavin Brown[1]

{andrew.webb,charles.reynolds,mikel.lujan,gavin.brown}@manchester.ac.uk
{iliescudanandrei,henrywjreeve}@gmail.com

[1]School of Computer Science, University of Manchester
[2]Department of Computer Science and Technology, University of Cambridge
[3]School of Computer Science, University of Birmingham

Abstract

We examine the practice of joint training for neural network ensembles, in which a multi-branch architecture is trained via single loss. This approach has recently gained traction, with claims of greater accuracy per parameter along with increased parallelism. We introduce a family of novel loss functions generalizing multiple previously proposed approaches, with which we study theoretical and empirical properties of joint training. These losses interpolate smoothly between independent and joint training of predictors, demonstrating that joint training has several disadvantages not observed in prior work. However, with appropriate regularization via our proposed loss, the method shows new promise in resource limited scenarios and fault-tolerant systems, e.g., IoT and edge devices. Finally, we discuss how these results may have implications for general multi-branch architectures such as ResNeXt and Inception.

1 Introduction

Ensembles of neural networks are now common in the Deep Learning literature. It is well-known that ensembles often top the leaderboard of Kaggle competitions. A recent paper (Dutt et al., 2018a) proposed training ensemble architectures via a *single* loss function, and claims that it allows one to *"significantly reduce the number of parameters"* while maintaining accuracy. The idea of joint training of ensembles highlights the structural similarities between ensembles and multi-branch networks, illustrated in Figure 1. As expected in the fast-moving literature, this architecture is already gaining traction, both as a component within new methods (Furlanello et al., 2018) and in applications (Anastasopoulos and Chiang, 2018).
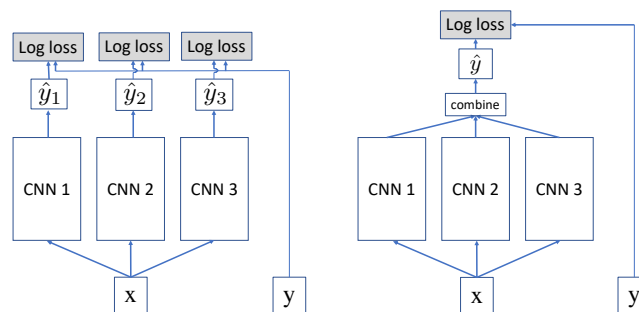

Figure 1: Computational graphs for an ensemble of CNNs (left) versus 'joint training' of a multi-branch architecture (right). We study a family of losses, Eq. (4), which smoothly interpolate between these two, highlighting the pros and cons of joint training.

There are however a number of open questions around this idea, most prominently, *when* & *why* does it work? When training an ensemble of predictors, the common folklore is that *diversity* of members is beneficial, hence the success of Random Forests and Bagging. In joint training, predictors and their parameters are coupled together, sharing a loss function—seemingly the opposite of diversity. This would suggest that the better strategy is independent training, but empirical results suggest otherwise. There is clearly a subtle relation between the concept of diversity in ensembles and that of joint training in multi-branch architectures, and our work aims to unpick this relation. Whilst we choose to focus on joint training in global multi-branch architectures (which can be viewed as ensembles, as in Figure 1), we believe this work has relevance for *within*-network branching.

Addressing the question of ensembles versus multi-branch architectures, Dutt et al. (2018a) explicitly claim their method is not an ensemble, because in joint training one has a single model made up of parallel branches, *"similar in spirit to the residual block in ResNet and ResNeXt, ... but is done at the full network level"*. Interestingly Xie et al. (2017), also argue that *"it is imprecise to view [ResNeXt] as ensembling, because the members to be aggregated are trained jointly"*. We introduce a family of *modular loss functions*, with a hyperparameter λ, that allow us to interpolate between independent ($\lambda = 0$) and joint ($\lambda = 1$) training. Our work generalizes both the independent and coupled ('FC') training methods of Dutt et al. (2018a), and also generalizes an older method for regression ensembles, Negative Correlation Learning (NCL) (Liu and Yao, 1999). We describe a distributed training procedure for our loss, and highlight two applications for partial evaluation of ensembles: it can be used to create *robust* ensembles and *specialist* sub-ensembles.

2 Background, Notation, and Terminology

2.1 Terminology: Modular Networks vs Ensembles

In this paper we consider two views of multi-branch architectures. Following Sharkey (1999), we refer to these models as *ensembles* consisting of *ensemble members* if constituent parts are trained separately and are functionally redundant. Conversely, we call the model a *modular neural network* consisting of *modules* if the model is trained as a whole, with its parts interacting. In this paper we introduce a training procedure that interpolates between these extremes, and therefore we may use the terminology interchangeably, or to emphasize one view or the other.

2.2 Base Models for NEF-Distributed Targets

We assume a standard supervised learning scenario, with training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ drawn i.i.d. from $P(\mathcal{X}\mathcal{Y})$. For generality, we assume y is distributed according to a natural exponential family (NEF) whose parameters are some unknown function of input $\mathbf{x}$, i.e., we assume a Generalized (non-)Linear Model. Formally, we assume that the target y, conditioned on $\mathbf{x}$, follows a distribution:

$$q(y \mid \eta(\mathbf{x})) = h(y) \cdot \exp(\eta(\mathbf{x}) \cdot y - A(\eta(\mathbf{x}))) \quad . \tag{1}$$

The functions h and A are fixed by the choice of target distribution, e.g. fixed-variance Gaussian, Categorical, etc. The problem is to learn the unknown function $\eta(\mathbf{x})$, i.e. the natural parameter of the target distribution as a function of the input. We approximate $\eta(\mathbf{x})$ with $\hat{\eta}(\mathbf{x}, \boldsymbol{\theta})$, the (pre-activation) final layer of a neural network with parameters $\boldsymbol{\theta}$. We often write this $\hat{\eta}(\mathbf{x})$, dropping the dependence on the parameters. We set the final layer activation function as the inverse of the canonical link function f for the relevant target distribution. See Table 1 for some example inverse link functions and log-normalizer functions.

Table 1: Example natural exponential families, their canonical inverse link functions, and their log-normalizers.

Distribution	Inverse link f^{-1}	Log normalizer A
Gaussian[1]	η	$\frac{\eta^2 + \log(2\pi)}{2}$
Categorical	$e^{\boldsymbol{\eta}}/(1 + \sum_{i=1}^{K-1} e^{\eta_i})$	$\ln(1 + \sum_{i=1}^{K-1} e^{\eta_i})$
Poisson	e^{η}	e^{η}

---

[1]Assuming fixed unit variance.

The output of a network $\hat{y}(\mathbf{x}) = f^{-1}(\hat{\eta}(\mathbf{x}))$ is an estimate of the conditional mean of the target distribution. The negative log-likelihood (NLL) of the target value for a given example reduces to well-known loss functions, e.g. the Gaussian distribution gives squared loss, or a Categorical distribution gives the cross-entropy loss for classification.

2.3   Normalized Geometric Mean Ensemble Models

Suppose we have a set of M base models $q(y \mid \hat{\eta}_j(\mathbf{x}))$. Our ensemble will be the normalized geometric mean,

$$\bar{q}(y \mid \mathbf{x}) \overset{\text{def}}{=} Z^{-1} \prod_{j=1}^{M} q(y \mid \hat{\eta}_j(\mathbf{x}))^{1/M} = q(y \mid \bar{\eta}(\mathbf{x})) \ , \tag{2}$$

$$\text{where } \bar{\eta}(\mathbf{x}) = \frac{1}{M} \sum_{j=1}^{M} \hat{\eta}_j(\mathbf{x}) \quad , \tag{3}$$

where Z is a normalization term. I.e., the ensemble distribution is of the same type as the base models, and has as its natural parameter estimate $\bar{\eta}(\mathbf{x})$ the arithmetic mean of the estimates. The ensemble conditional mean estimate is $\bar{y}(\mathbf{x}) = f^{-1}(\bar{\eta}(\mathbf{x}))$, which is also a generalized f-mean of the outputs $\bar{y}(\mathbf{x}) = f^{-1}(\frac{1}{M} \sum_j f(\hat{y}_j(\mathbf{x})))$, where f is again the canonical link function.

2.4   Concrete Examples

In the case of regression with Gaussian noise (assuming unit variance), the inverse link is the identity, and the ensemble conditional mean estimate is the *arithmetic mean* of the individual estimates of the ensemble members. This is common practice in combining regression predictors, and is used in NCL (Liu and Yao, 1999). In classification with a Categorical target, the canonical inverse link is softmax, and the natural parameters of a model are its logits. The ensemble conditional mean estimate is a *normalized geometric mean* of the probabilities, or equivalently softmax of the arithmetic mean of the ensemble member logits. This is how predictions are combined in the coupled ('FC') method of Dutt et al. (2018a).

3   Between Independent and Joint Training

In this section we describe a framework for training 'modular' architectures, which has explicit control of the degree of module interactions, and include as special cases both independent and joint training. This explicit control allows for a 'forensic' analysis of the joint training idea, analyzing how and why it works.

3.1   Modular Losses

Let $\mathcal{H} = \{\hat{\eta}_j\}_{j=1}^{M}$ denote the set of natural parameter estimates, i.e. the final layer activation vectors from a set of M networks. We propose the following *modular loss function*:

$$L_\lambda(p, \mathcal{H}) \overset{\text{def}}{=} \lambda \mathrm{D}(p \,\|\, \bar{q}) + (1 - \lambda) \frac{1}{M} \sum_{j=1}^{M} \mathrm{D}(p \,\|\, q_j) \tag{4}$$

where D is the KL-divergence, q_j is short for $q(y \,|\, \hat{\eta}_j(\mathbf{x}))$ and $\bar{q}$ for $q(y \,|\, \bar{\eta}(\mathbf{x}))$, and p is the data distribution. The modular loss in (4) is a convex combination of two extremes. At $\lambda = 1$ we minimize the NLL of the modular network, which is *exactly* the coupled ('FC')[2] method of Dutt et al. (2018a). At $\lambda = 0$, we train each member of the ensemble independently (with learning rate scaled by $1/M$).

---

[2]The 'SM' and 'LL' coupled training methods of Dutt et al. (2018a) can be shown to actually be independent training with a scaled learning rate (see Appendix).

Our loss can be re-expressed in two alternative ways, which give some insight into its properties. By simply dividing through by the constant λ (if $\lambda \neq 0$), we have:

$$L_\lambda(p, \mathcal{H}) \propto \mathrm{D}(p \,\|\, \bar{q}) + \frac{\gamma}{M} \sum_{j=1}^{M} \mathrm{D}(p \,\|\, q_j) \tag{5}$$

where $\gamma = \frac{(1-\lambda)}{\lambda}$. When $\gamma = 0$, this is exactly the NLL of the jointly trained system, and as γ grows, the NLL is regularized by the individual member losses. Alternatively (4) can be rearranged to

$$L_\lambda(p, \mathcal{H}) = \frac{1}{M} \sum_{j=1}^{M} \mathrm{D}(p \,\|\, q_j) - \frac{\lambda}{M} \sum_{j=1}^{M} \mathrm{D}(\bar{q} \,\|\, q_j) \ . \tag{6}$$

This expression is in fact a parameterization of the ambiguity decomposition (Heskes, 1998; Krogh and Vedelsby, 1994; Hansen and Heskes, 2000), which shows how the log likelihood of a product of experts decomposes into an average log likelihood, minus a target-independent interaction term. This interaction term—known as the 'Ambiguity'—is the source of the common mantra about ensembles: that they have better performance than the average member. As a parameterization of the ambiguity decomposition, the modular loss generalizes NCL—which can be motivated by the ambiguity decomposition for squared loss (Brown et al., 2005)—to targets following any natural exponential family. We conclude that the ambiguity form of the loss (6) sheds light on the relation between joint training and ensemble diversity. When $\lambda = 1$, (4) is trivially $\mathrm{D}(p \,\|\, \bar{q})$—the full joint training method—and in the following section we rule out $\lambda > 1$. Therefore it can be seen from (6) that $\lambda = 1$ also maximizes the contribution of ambiguity term in the loss, which quantifies the 'diversity' of module predictions.

The coefficient λ allows us to smoothly interpolate the training procedure, and may be interpreted as training an ensemble while encouraging diversity, or as training a single modular network with a regularization term that encourages each module to fit the data independently.

For a training example $(\mathbf{x}, y)$, each KL divergence term of (4) can be written, up to an irrelevant additive constant, as a negative log-likelihood, and so the loss can be written as

$$-\lambda \ln q(y \mid \bar{\eta}) - (1 - \lambda) \frac{1}{M} \sum_{j=1}^{M} \ln q(y \mid \hat{\eta}_j) + C \ . \tag{7}$$

A useful property from the GLM framework is that $\partial L_\lambda / \partial \hat{\eta}_j$ takes the same form for all target distributions:

$$\frac{\partial L_\lambda}{\partial \hat{\eta}_j} = \frac{1}{M} \left(\hat{y}_j(\mathbf{x}) - y - \lambda(\hat{y}_j(\mathbf{x}) - \bar{y}(\mathbf{x})) \right) . \tag{8}$$

3.2 An Upper Bound on λ

Here we show that the λ parameter should be bounded $\lambda \leq 1$, as the loss surface displays pathological behaviour beyond this point. The NLL for a module with natural parameter estimate $\hat{\eta}$ may be written

$$L(\hat{\eta}) = \hat{\mathbb{E}}_n[A(\hat{\eta}) - \hat{\eta} \cdot y] - \hat{\mathbb{E}}_n[\ln h(y)] \quad , \tag{9}$$

where $\hat{\mathbb{E}}_n[\cdot]$ denotes expectation over the data distribution. For a set of natural parameter estimates $\mathcal{H} = \{\hat{\eta}_j\}_{j=1}^{M}$ with arithmetic mean $\bar{\eta}$, the modular loss may be written

$$L_\lambda(\mathcal{H}) = (1 - \lambda) \cdot \frac{1}{M} \sum_{j=1}^{M} L(\hat{\eta}_j) + \lambda \cdot L(\bar{\eta}) \quad . \tag{10}$$

Assumption 1. $\forall \mu \in \mathbb{R}^K$ *there exists a sequence of natural parameter estimates* $(v_l)_{l \in \mathbb{N}} \subset \mathbb{R}^K$ *with* $\lim_{l \to \infty} (A(v_l) - v_l \cdot \mu) = \infty$.

Theorem 1. *Suppose that $M \geq 2$ and that Assumption 1 holds, and fix $\lambda > 1$. There exists a sequence of sets of natural parameter estimates $\{\mathcal{H}^l\}_{l \in \mathbb{N}}$, where $\mathcal{H}^l = \{\hat{\eta}_j^l\}_{j=1}^M$ with arithmetic mean $\bar{\eta}^l$, with modular loss $\lim_{l \to \infty} L_M(\mathcal{H}^l; \lambda) = -\infty$ while the overall NLL $\lim_{l \to \infty} L(\bar{\eta}^l) = +\infty$.*

Proof. Let $\mu = \hat{\mathbb{E}}_n[y]$. By Assumption 1 we can form sequence $(v_l)_{l \in \mathbb{N}} \subset \mathbb{R}^k$ with

$$\lim_{l \to \infty} (A(v_l) - v_l \cdot \mu) = \infty \quad . \tag{11}$$

Now choose a sub-sequence $(u_l)_{l \in \mathbb{N}} \subset (v_l)_{l \in \mathbb{N}}$ so that

$$\lim_{l \to \infty} \left(\frac{A(u_l) - u_l \cdot \mu}{A(v_l) - v_l \cdot \mu} \right) = \infty \quad . \tag{12}$$

This is always possible, by (11). We may construct $\mathcal{H}^l$ by taking $\hat{\eta}_1^l = u_l, \hat{\eta}_2^l = M \cdot v_l - u_l$ and $\hat{\eta}_i^l = 0$ for $i > 2$. Hence, $\bar{\eta}^l = v_l$. It follows from (11) that $\lim_{l \to \infty} L(\bar{\eta}^l) = \infty$. On the other hand, it follows from (12) that

$$\lim_{l \to \infty} \frac{\frac{1}{M} \sum_{j=1}^M L(\hat{\eta}_j^l)}{L(\bar{\eta}^l)} \geq \frac{1}{M} \cdot \lim_{l \to \infty} \frac{L(\hat{\eta}_1^l)}{L(\bar{\eta}^l)} = \infty \quad , \tag{13}$$

and when $\lambda > 1$ we have $\lim_{l \to \infty} L_\lambda(\mathcal{H}) = -\infty$. $\square$

In other words, with $\lambda > 1$ we may achieve arbitrarily low modular loss while the ensemble NLL is arbitrarily large. Assumption 1 is trivially verified for the Gaussian, Categorical, and Poisson distribution.

3.3   Parallelized and Distributed Training

In this section we describe how the modular loss is particularly amenable to distributed implementations. If the modules share the same architecture, then matrix multiply and convolution operations of a modular network may be parallelized on a single device; a per-module dense matrix multiply may be replaced with a single batched matrix multiply, and a per-module convolution may be replaced by a grouped convolution—included in frameworks such as Caffe (Jia et al., 2014) and PyTorch (Paszke et al., 2017), and natively in cuDNN (Chetlur et al., 2014).

Training and inference with the modular loss can also be distributed across devices. The distributed procedure is provably identical to processing on a single machine; we omit a formal proof, but it is trivial by the observation that for any parameter θ in network j, we have that $\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \eta_j} \frac{\partial \eta_j}{\partial \theta}$, and the term $\frac{\partial L}{\partial \eta_j}$ in (8) can be computed locally if $\bar{y}$ is broadcast to all compute nodes.

Distributed training works as follows. Each computational node performs a forward pass, computing $\hat{\eta}_j$, and sends it to an aggregation node. The aggregator computes the average $\bar{\eta}$ and uses the appropriate inverse link to compute the prediction $\bar{y}$. The aggregator sends $\bar{y}$ to the computational nodes, which then do a backward pass starting from (8). For mini-batch of size B, and K classes, $B \times K$ values for $\bar{y}$ must be broadcast, independently of the size of the network. This communication overhead could be compared to a popular alternative to distributed training, data parallelization (Chu et al., 2006). This requires that each node send the entire gradient matrices, with the communication overheads growing with the model size, and typically the speed-up is sub-linear beyond a dozen nodes (Keuper and Preundt, 2016).

4   Experiments

We focus in this section on the modular loss in the domain of classification. For a summary of Negative Correlation Learning, which is a special case of the modular loss in the case of regression with Gaussian noise, see Chen et al. (2012). In the following sections, we describe small scale MLP experiments on the Fashion-MNIST dataset, larger scale convolutional network experiments on CIFAR-10, and large scale DenseNet experiments on CIFAR-100. Code for the following experiments is provided at `github.com/grey-area/modular-loss-experiments`.

4.1   Small Scale MLPs on Fashion-MNIST

Dataset.   The Fashion-MNIST dataset (Xiao et al., 2017) consists of 60,000 28×28 greyscale images of items of clothing, with 10 classes. We use the predefined train/test split, but hold out 10,000 of the training examples for early stopping. No data augmentation is applied, and we apply mean and standard deviation normalization.

Architectures. We train ensembles of single hidden-layer MLPs with ReLU activations. We train four configurations, each with $\sim$815K parameters: a single module with 1024 hidden nodes (1-M-1024-H), 16 modules with 64 hidden nodes each (16-M-64-H), 64 modules with 16 nodes (64-M-16-H), and 256 modules with 4 nodes (256-M-4-H).

Training. Each model is trained with SGD, batch size 100. We evaluate λ values $\{0.0, 0.5, 0.8, 0.9, 0.95, 0.99, 1.0\}$ over 5 trials of random train/validation splits and initializations. We tune the learning rate independently for each configuration and λ, and use momentum 0.9. We train for 200 epochs, before computing test error at the epoch where validation error was minimized.

Results. Figure 2 shows the average test error and standard errors for each configuration and λ value. Joint training ($\lambda = 1$) achieves a significantly lower error rate than independent training ($\lambda = 0$) for the 256-M-4-H configuration, with a large collection of very small neural networks. This effect is reduced as we move towards the 16-M-64-H configuration, with a smaller collection of larger neural networks. Additionally, there is a small rise in test error at $\lambda = 1$.
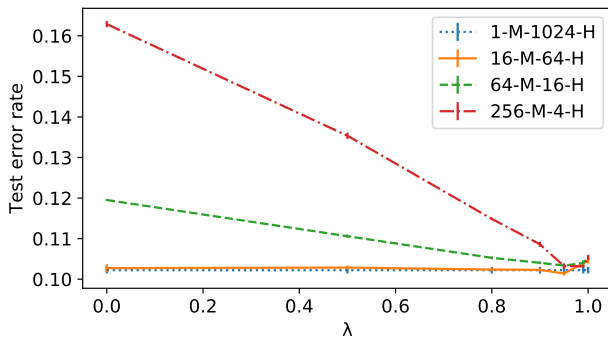

Figure 2: Ensemble test error against λ for MLPs on Fashion-MNIST. The width and parameter count of each ensemble is the same across each configuration. Joint training (higher λ values) leads to much lower error rates when ensemble members are small. The effect is reduced as the base architecture grows.

The ensemble of 256 networks has comparable test error, at $\lambda = 0.95$, to the monolithic network of 1024 hidden nodes and has an equal number of parameters, but is much more parallelizable. This result is summarized in Table 2.

Table 2: A single large MLP and an ensemble of smaller MLPs with $\sim$815K parameters. Joint training performs well, but a little regularization ($\lambda = 0.95$) improves accuracy further.

Architecture	Test error
1-M-1024-H	10.2%
256-M-4-H (independent)	16.3%
256-M-4-H (joint)	10.5%
256-M-4-H ($\lambda = 0.95$)	**10.3%**

Figure 3 shows, for the 256-M-4-H configuration—in which joint training strongly outperforms independent training—the ensemble cross entropy loss and the average of the cross entropy losses of the individual modules. There is a large rise in the average module loss between $\lambda = 0.99$ and $\lambda = 1$. This is unsurprising, as in joint training the losses of the individual modules are not targeted directly. This observation suggests that using λ slightly less than 1 during joint training may greatly increase the robustness of the ensemble to having modules removed during inference, which we explore in the following section.

Robustness. Here we evaluate the 256-M-4-H configuration's robustness to the removal of modules during inference for each of the λ values $\{0.0, 0.99, 1.0\}$. Two use cases for such partial evaluation are 'budgeted
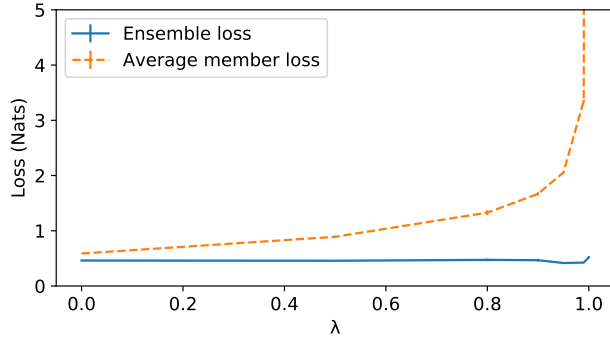
Figure 3: Ensemble cross entropy loss and the average of the ensemble member losses for the 256-M-4-H configuration. The average member loss grows sharply as λ tends to 1.

batch classification'—where a fixed time/computation budget is shared unevenly across examples that differ in importance or difficulty (Huang et al., 2018)—and 'anytime prediction'—in which inference accuracy is improved with increased available resource (Grubb and Bagnell, 2012).

Figure 4 shows the effect of randomly dropping a given number of the 256 modules from the ensemble. Accuracy is averaged over 20 trials of random module removal for each training trial. Independent training ($\lambda = 0$) gives a highly robust ensemble, which, conversely, can be seen to be redundant; adding modules does not increase the test accuracy. Joint training ($\lambda = 1$) results in much higher test accuracy when the entire ensemble is used, but the ensemble is not at all robust. Using a slightly lower λ value (0.99) does not reduce the test accuracy when the whole ensemble is used, but gives a much more robust ensemble; its accuracy when $\sim$90% of modules are dropped is the same as the $\lambda = 1$ ensemble when $\sim$50% of modules are dropped.
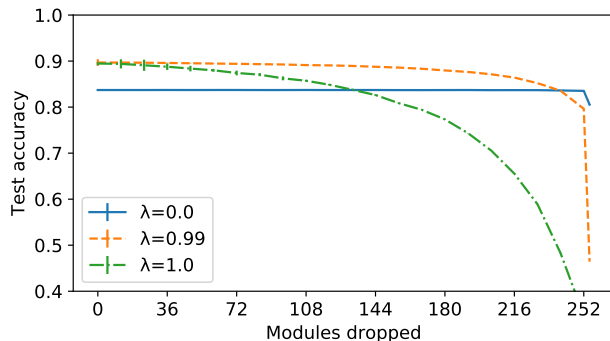

Figure 4: Independent training ($\lambda = 0$) gives robust ensembles; however, the ensemble members are redundant. Joint training ($\lambda = 1$) gives brittle ensembles. Slightly reducing the λ value retains the benefits of joint training while adding robustness.

As established in Section 3, the modular loss is a generalization of the NCL method (Liu and Yao, 1999), to targets following any natural exponential family distribution. It is interesting to note that Reeve et al. (2018) prove NCL has an equivalence with Dropout (Srivastava et al., 2014) on the final combining layer. It turns out this equivalence does not hold for the general modular loss, but we believe there is a qualitative similarity, deserving of investigation in future, which would explain the robustness property observed here.

Module Specialization. We have seen that lower λ values give more robust ensembles. In this section we show that ensembles trained with higher λ values, by virtue of the their model diversity, contain *specialist* sub-ensembles. Per-branch specialization during joint training of branching architectures has been observed in the literature (Krizhevsky et al., 2012).

We take the 256-M-4-H configuration, trained on the Fashion-MNIST training set. We split the remainder of the dataset in half, as validation and test sets, and replace the 10 class problem with 10 binary one-vs-all

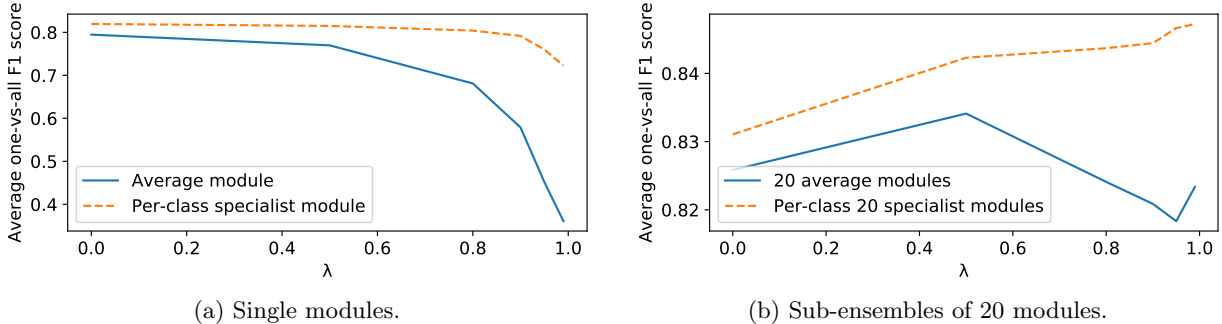(a) Single modules.　　　　　　　　　　　(b) Sub-ensembles of 20 modules.

Figure 5: We create 10 binary problems from the Fashion-MNIST dataset and evaluate the average F1 score of randomly selected sub-ensembles to the average score of per-problem specialist sub-ensembles. Higher λ values create model diversity, which can be leveraged to create better specialist sub-ensembles.

classification problems. We evaluate the expected F1 score on the test set averaged across these binary problems of a single module and a subset of 20 modules each chosen at random.

We compare the average F1 scores of random sub-ensembles to specialists chosen using model selection on the validation set. For each of the binary sub-problems, we choose the module/subset of 20 modules that maximizes the F1 score on the validation set, and then evaluate on the test set. Since we cannot evaluate each of the $\binom{256}{20}$ sub-ensembles of size 20 for model selection, we evaluate 200 chosen at random. The results are averaged over the five training trials.

As shown in Figure 5, the gap in performance between a randomly chosen sub-ensemble and the selected specialist sub-ensembles grows with λ. This indicates the existence of sub-problem specialist modules with higher λ values, or at least the presence of model diversity that can be leveraged in order to build specialist sub-ensembles. As shown in Figure 5b, there are cases where increasing the λ parameter will decrease the performance of randomly chosen sub-ensembles (due to reduced robustness) while increasing the performance of specialist sub-ensembles.

4.2   Small Convolutional Networks on CIFAR-10

Dataset.　The CIFAR-10 dataset (Krizhevsky, 2009) consists of 32×32 colour images with 10 balanced classes. We use the predefined train/test split, and hold out 10,000 of the training examples as a validation set for early stopping. We apply per-channel mean and standard deviation normalization, and apply the standard flip and crop data augmentation used for this dataset (see, e.g., He et al. (2016)).

Architecture.　We train ensembles of 10 CNNs with ReLU activations, with the number of filters doubling when the spatial resolution halves, in the style of VGG (Simonyan and Zisserman, 2014). We apply global pooling before the final fully connected layer, in the style of MobileNets (Howard et al., 2017). The architecture is described in Table 3. The ensemble has $\sim$2.8M parameters.

Table 3: The architecture used in the CIFAR-10 experiments.

Type/stride	Filter shape	Input size
Conv / s1	$3 \times 3 \times 3 \times 32$	$32 \times 32 \times 3$
Conv / s2	$3 \times 3 \times 32 \times 64$	$30 \times 30 \times 32$
Conv / s1	$3 \times 3 \times 64 \times 64$	$14 \times 14 \times 64$
Conv / s2	$3 \times 3 \times 64 \times 128$	$12 \times 12 \times 64$
Conv / s1	$3 \times 3 \times 128 \times 128$	$5 \times 5 \times 128$
Avg. pool	Pool 3×3	$3 \times 3 \times 128$
FC	128×10	128
Softmax	–	10

Training. Each configuration is trained using SGD with a batch size of 100. We tune learning rates for each λ value, and use momentum parameter 0.9. We train for 400 epochs, before reporting the test error rate at the epoch at which validation error is minimized. We evaluate the λ values $\{0.0, 0.5, 0.8, 0.9, 0.95, 0.97, 0.99, 1.0\}$ over 5 trials of random training/validation splits and parameter initializations.

Results. Figure 6 shows the average test error rate and standard errors for each λ value. As in Section 4.1, the error rate initially decreases with λ. However, unlike the MLP experiment, we see a sharp increase in error rate as λ approaches 1. We believe this is related to a 'module dominance' phenomenon that occurs in this experiment.
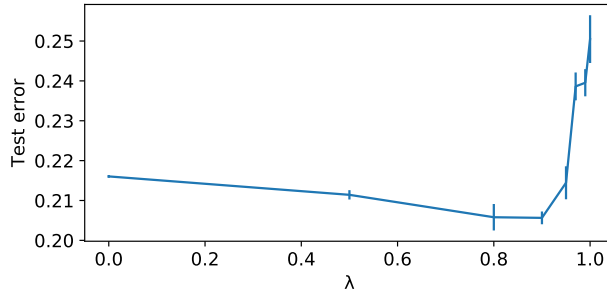

Figure 6: An ensemble of convolutional nets on CIFAR-10. The sharp rise in test error as λ tends to 1 may be due to a single module fitting the data, reducing the magnitude of the loss, and trapping the other modules near their initialization points.

Module Dominance. When training jointly ($\lambda = 1$), we might expect each module individually to achieve a high test error. However, we find that a single module achieves a relatively low error rate (~ 0.43) on its own and the remainder of the modules make only small contributions to the ensemble predictions. Figure 7 shows the ensemble and module validation error rates over time for a single trial when training with $\lambda = 1$. Figure 8 shows some of the 3×3 filters in the first layer of these modules, showing a small number of strong filters when $\lambda = 1$. This effect was not observed in the regression case (Liu and Yao, 1999) as the ensemble prediction is an
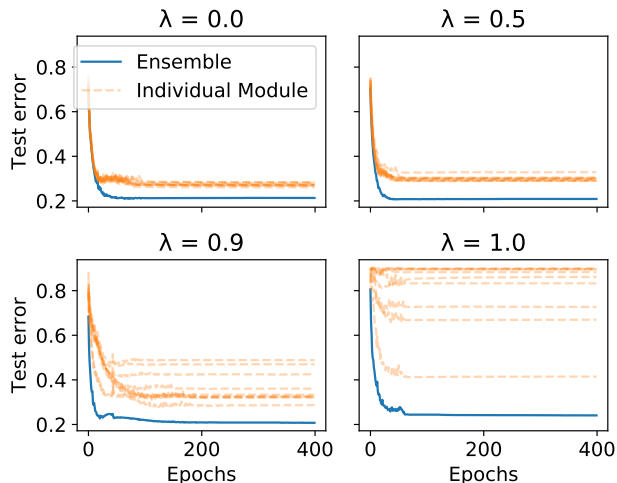

Figure 7: Ensemble and module test errors in one trial. At $\lambda = 1$, one module has significantly lower error rates than the others.

arithmetic mean. For classification, the ensemble prediction is a normalized geometric mean. As a result, a single module closely fitting the data can arbitrarily reduce the ensemble error, explained by the well-known 'veto' effect in Products of Experts (Welling, 2007). If such a module fits the data quickly, the ensemble error will drop, and the parameters of other modules can be prevented from moving far from their initial values.
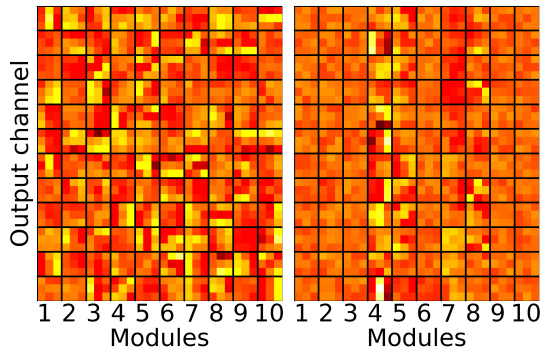
9

Figure 8: A visualization of 3×3 filters in the first layer of 10 CNNs trained independently (left) and jointly (right). In joint training, only module 4 has strong filters (in this trial).

4.3   DenseNet Experiments

Dataset.   CIFAR-100 (Krizhevsky, 2009) consists of 32×32 colour images with 100 classes. We use the predefined train and test examples, per-channel mean and standard deviation normalization, and the standard data augmentation.

Architectures.   We train three configurations of ensembles of DenseNet-BC networks (Huang et al., 2017). We train 4 modules with a depth of 100 and a growth rate of 12 (DN-100-12-4), which is a configuration used in Dutt et al. (2018a). We also train ensembles of 8 and 16 smaller DenseNet modules, in each case designing the ensemble to fill the memory of an NVIDIA Tesla V100 GPU. We describe each configuration in Table 4.

Our DenseNet implementation is based on Amos and Kolter (2017). As described in Section 3.3, we use batched matrix multiplies and grouped convolutions to implement the ensemble as a single PyTorch model.

Table 4: The architectures used in the DenseNet experiments.

Name	Depth	k	Modules	Parameters
DN-100-12-4	100	12	4	3.2M
DN-82-8-8	82	8	8	2.1M
DN-64-6-16	64	6	16	1.7M

Training.   We use the training procedure described by Huang et al. (2017); Dutt et al. (2018a). We use SGD with a batch size of 64, and initial learning rate of 0.1 decreased by a factor of 10 at epochs 150 and 225. We use a momentum value of 0.9. We evaluate the λ values $\{0.0, 0.5, 0.9, 1.0\}$ over 3 trials of parameter initialization.

Results.   Table 5 summarizes the mean test errors across configurations and λ values. Each row contains the results for a DenseNet ensemble configuration, with the minimum error rate in bold. In every case joint training ($\lambda = 1$) is suboptimal. In all but DN-64-6-16, which is the configuration with the largest ensemble of smallest base architectures, independent training ($\lambda = 0$) achieves the lowest ensemble test error. The first row results, for DN-100-12-4, are in agreement with Dutt et al. (2018a, Table 1).

Table 5: Error rates for DenseNet ensembles. Independent training is optimal (bold) for all but the smallest base architecture.

λ:	0.0	0.5	0.9	1.0
DN-100-12-4	**17.8%**	18.8%	20.3%	22.1%
DN-82-8-8	**19.9%**	20.0%	22.4%	25.7%
DN-64-6-16	25.0%	**23.5%**	25.9%	29.6%

Figure 9 shows the ensemble and module test error over time. With joint training ($\lambda = 1$), we again observe the module dominance effect, as a single module has significantly lower test error than the rest. The effect is reduced as the base architecture size decreases and the ensemble size increases. To further examine the effects of module dominance, we examine confusion matrices (see Figure 10) and the entropy of predicted class distributions, per module, averaged over the test set (Table 6). When $\lambda = 0$, strong leading diagonals in confusion matrices indicate that all modules achieve low error on all classes. With increasing λ, the diagonals become less pronounced, as modules specialize towards certain classes. At $\lambda = 1$ there is a sharp transition in behaviour, with vertical banding indicating that some modules are strongly biased to a small number of classes.
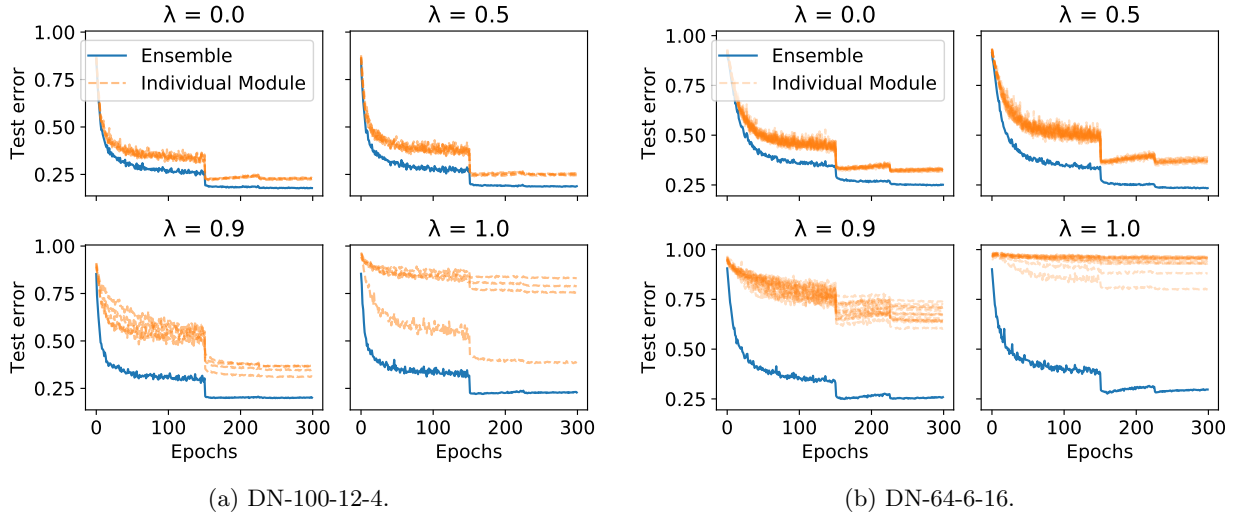

(a) DN-100-12-4.

(b) DN-64-6-16.

Figure 9: We create 10 binary problems from the Fashion-MNIST dataset and compare the average F1 score of randomly selected sub-ensembles to the average score of per-problem specialist sub-ensembles. Higher λ values create model diversity, which can be leveraged to create better specialist sub-ensembles.
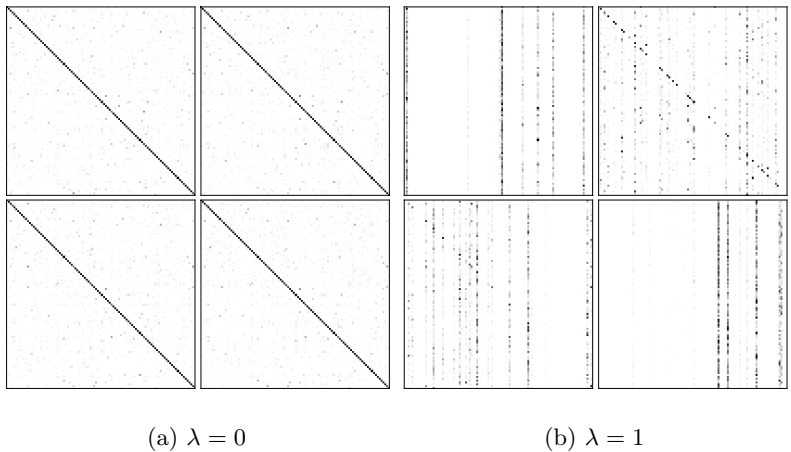

(a) $\lambda = 0$

(b) $\lambda = 1$

Figure 10: Confusion matrices (x-axis predicted class, y-axis true class) of 4 modules from a single trial of DN-64-6-16. At $\lambda = 0$ we see strong leading diagonals, indicating low error rates across classes. At $\lambda = 1$ (joint training) modules have faint diagonals and strong vertical banding, indicating a bias towards certain classes.

Further insight is gained from measuring the entropy of the predicted class distributions across modules, summarized in Table 6. At $\lambda = 1$, some modules have much lower entropy than when trained with $\lambda = 0$. However, we also see that some modules have much *higher* entropy, indicating that there are under-confident as well as over-confident modules.

11

Table 6: Output entropy stats on test set, 1 trial of DN-64-6-16.

λ:	0.0	0.5	0.9	1.0
$\min(H[q_i])$	1.0271	0.9957	1.0625	0.1394
$\max(H[q_i])$	1.0772	1.0576	2.4385	2.2232
$\mathrm{mean}(H[q_i])$	1.0439	1.0205	1.8681	0.9895
$\mathrm{std}(H[q_i])$	0.0135	0.0195	0.3673	0.6446

5 Conclusions

We have studied the idea of 'joint' training in neural network ensembles, unpicking the relation between ensembles and multi-branch architectures trained via a single loss. We find numerous interesting dynamics not previously observed, including modules dominating predictions and the existence of specialist sub-ensembles. We also find a general trend: if we use low-complexity modules, joint training performs well, but for larger architectures it becomes less effective. In the extreme of modules that are large, state-of-the-art CNNs, we find that independent training of ensemble members achieves the lowest test error rates.

We also provide insight into the relation between joint training and ensemble diversity—showing that joint training can be expressed as a loss equally balancing individual member losses and their divergences from each other (Heskes, 1998).

Finally an unexpected benefit emerges, in that regularized joint training (i.e. our modular loss with $\lambda < 1$) seems well suited to creating *fault-tolerant* systems, robust to removal of modules, whilst retaining accuracy. This may have implications for partial evaluation in resource limited scenarios, e.g. IoT and edge devices on limited power budgets.

Here we have studied ensembles, i.e. 'global' multi-branch architectures where each branch is a separate network. In ongoing work we are extending the ideas of this paper to architectures with multiple branches *within* a single network, such as ResNeXt and Inception. In particular, we believe the robustness/partial evaluation results of Section 4.1 can be extended to such architectures, allowing inference-time trade-off between computation and model accuracy.

References

B. Amos and J. Z. Kolter. A PyTorch implementation of DenseNet. `github.com/bamos/densenet.pytorch`, 2017.

A. Anastasopoulos and D. Chiang. Leveraging translations for speech transcription in low-resource settings. *arXiv:1803.08991*, 2018.

G. Brown, J. L. Wyatt, and P. Tiňo. Managing diversity in regression ensembles. *JMLR*, 6:1621–1650, 2005. `dl.acm.org/citation.cfm?id=1046920.1194899`.

H. Chen, A. G. Cohn, and X. Yao. Ensemble learning by negative correlation learning. In *Ensemble Machine Learning*, pages 177–201. Springer, 2012.

S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv:1410.0759*, 2014.

C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288. MIT Press, 2006. `dl.acm.org/citation.cfm?id=2976456.2976492`.

A. Dutt, D. Pellerin, and G. Quénot. Coupled ensembles of neural networks. *ICLR Workshops*, 2018a. `arxiv.org/abs/1709.06053`.

A. Dutt, D. Pellerin, and G. Quénot. A PyTorch implementation of Coupled Ensembles. `github.com/vabh/coupled_ensembles`, 2018b.

T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. Born again neural networks. *arXiv:1805.04770*, 2018.

A. Grubb and J. A. Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, 2012.

J. V. Hansen and T. Heskes. General bias/variance decomposition with target independent variance of error functions derived from the exponential family of distributions. In *ICPR*, volume 2, pages 207–210, 2000.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, June 2016.

T. Heskes. Selecting weighting factors in logarithmic opinion pools. In *NIPS*, pages 266–272. The MIT Press, 1998.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, pages 2261–2269, 2017. doi: 10.1109/CVPR.2017.243.

G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018. `openreview.net/forum?id=Hk2aImxAb`.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

J. Keuper and F. J. Preundt. Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. In *2nd Workshop on Machine Learning in HPC Environments*, pages 19–26, 2016. `doi.org/10.1109/MLHPC.2016.006`.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. In *NIPS*, pages 231–238. MIT Press, 1994. `dl.acm.org/citation.cfm?id=2998687.2998716`.

Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural networks*, 12 10:1399–1404, 1999.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

H. Reeve, T. Mu, and G. Brown. Modular dimensionality reduction. In *European Conference on Machine Learning*, 2018.

A. J. C. Sharkey. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer, 1999.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, pages 1929–1958, 2014. `jmlr.org/papers/v15/srivastava14a.html`.

M. Welling. Product of experts. *Scholarpedia*, 2(10):3879, 2007. doi: 10.4249/scholarpedia.3879. revision #137078.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 5987–5995, 2017.

Appendix

We claim that the 'LL' and 'SM' coupled training methods of Dutt et al. (2018a) for ensembles of classifiers are actually equivalent to independent training, up to a scaling of learning rate. We demonstrate that here.

Suppose we have a collection of M neural networks for a K class classification problem. Let $q_k^{(m)}$ denote the kth post-softmax output of the mth neural network for a given example. Let $\mathbf{y}$ be the one hot-encoded true label. The cross entropy loss $L^{(m)}$ of the mth network is

$$L^{(m)} = - \sum_k y_k \log q_k^{(m)} \quad . \tag{14}$$

The 'LL' coupled training method of Dutt et al. (2018a) has as its loss function L_{LL} the arithmetic mean of the cross entropy loss functions for each network. I.e.,

$$L_{LL} = \frac{1}{M} \sum_m L^{(m)} \quad . \tag{15}$$

It follows from the fact that $\partial L^{(m)} / \partial q_k^{(n)} = 0$ if $m \neq n$—i.e., that the cross entropy loss of one network does not depend on the output of another—that

$$\frac{\partial L_{LL}}{\partial q_k^{(m)}} = \frac{1}{M} \frac{\partial L^{(m)}}{\partial q_k^{(m)}} \quad . \tag{16}$$

In words, the gradient of the 'LL' loss with respect to a given network output—and therefore the gradient with respect to the network parameters—is the same as when training independently, scaled by a factor $1/M$.

The 'SM' coupled training method of Dutt et al. (2018a) works as follows. First, take the log of the probabilities $q_k^{(m)}$, and then take the arithmetic mean across networks. The key point here is that the result is not a vector of log probabilities; it is un-normalized. An inspection of the authors' provided code (Dutt et al., 2018b) shows that, in the 'SM' method, this un-normalized log probability vector is given as input to the `NLLLoss` loss function provided by PyTorch, which expects log probabilities. The result is that the cross entropy loss is applied to the un-normalized probabilities

$$\tilde{q}_k = \exp \left(\frac{1}{M} \sum_m \log q_k^{(m)} \right) \quad , \tag{17}$$

and that, if $\mathbf{y}$ is the one hot-encoded true label, the loss function that is effectively used is

$$L_{SM} = - \sum_k y_k \log \tilde{q}_k \tag{18}$$

$$= - \sum_k y_k \frac{1}{M} \sum_m \log q_k^{(m)} \tag{19}$$

$$= \frac{1}{M} \sum_m \sum_k y_k \log q_k^{(m)} \tag{20}$$

$$= \frac{1}{M} \sum_m L^{(m)} \tag{21}$$

$$= L_{LL} \quad . \tag{22}$$

This suffices to demonstrate that the 'LL' and 'SM' methods are equivalent to independent training up to a scaling of learning rate. We also provide code that empirically demonstrates the equivalence at `github.com/grey-area/modular-loss-experiments`.