Supplementary Material: To Ensemble or Not Ensemble: When does End-To-End Training Fail?

Andrew Webb[1], Charles Reynolds[1], Wenlin Chen[1], Henry Reeve[2], Dan Iliescu[3], Mikel Luján[1], and Gavin Brown[1]

[1] University of Manchester, UK
[2] University of Bristol, UK
[3] University of Cambridge, UK

A Re-writing the Joint Training Loss

Here we show that the convex combination form and the 'ambiguity' form of the loss are equivalent. Starting from the convex combination form:

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

we use the *ambiguity decomposition* (Heskes, 1998):

$$\mathrm{D}(p \,\|\, \bar{q}) = \frac{1}{M}\sum_{j=1}^{M} \mathrm{D}(p \,\|\, q_j) - \frac{1}{M}\sum_{j=1}^{M} \mathrm{D}(\bar{q} \,\|\, q_j)\,. \tag{2}$$

Substituting the right-hand side for the $\mathrm{D}(p \,\|\, \bar{q})$ term in (1), we obtain the ambiguity form of the loss:

$$L_\lambda = \frac{1}{M}\sum_{j=1}^{M} \mathrm{D}(p \,\|\, q_j) - \frac{\lambda}{M}\sum_{j=1}^{M} \mathrm{D}(\bar{q} \,\|\, q_j)\,. \tag{3}$$

B Experimental Details

Here we specify details such as dataset, model architecture, and training for the modular loss experiments.

B.1 Spending a Fixed Parameter Budget—MLPs / Fashion-MNIST

Dataset. We use the Fashion-MNIST dataset (Xiao et al., 2017) with the predefined train/test split, holding out 10,000 training examples as a validation set for early stopping. We apply mean and standard deviation normalization, and no data augmentation.

Architectures. We use single layer MLPs with ReLU activations, in four configurations each with ∼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. We train for 200 epochs of SGD, batch size 100, momentum 0.9, and tune the learning rate independently for each configuration and λ. Final reported test error is that at the epoch where validation error was minimized. Results are averaged over 5 trials of random train/validation splits and initializations.

B.2 High Capacity Individual Models—DenseNets / CIFAR-100

Dataset. We use the CIFAR-100 (Krizhevsky, 2009) dataset with the predefined train/test split, per-channel mean and standard deviation normalization, and the standard data augmentation (see, e.g., He et al. (2016)).

Architectures. We train ensembles of DenseNet-BC networks (Huang et al., 2017). We train 4 modules with a depth of 100 and growth rate 12 (DN-100-12-4)—a configuration used in Dutt et al. (2020). We also train ensembles of 8 and 16 smaller DenseNet modules. Our DenseNet implementation is based on Amos and Kolter (2017).

Table 1. DenseNet architectures.

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

Training. We evaluate λ values {0.0, 0.5, 0.9, 1.0} over 3 trials of parameter initialization. We use the training procedure described by Huang et al. (2017); Dutt et al. (2020). We use SGD with batch size 64. The initial learning rate of 0.1 is decreased by a factor of 10 at epochs 150 and 225, with momentum 0.9.

B.3 Intermediate Capacity—Small ConvNets / CIFAR-100

Dataset. We use the CIFAR-100 dataset (Krizhevsky, 2009) with the predefined train/test split, holding out 10,000 training examples 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.

Architecture. We train ensembles of 16 CNNs with ReLU activations. We apply global pooling before the final fully connected layer, in the style of MobileNets (Howard et al., 2017). The networks are fully convolutional, and we evaluate a variety of architectures of varying complexity. The architectures are described in Table 2.

Table 2. ConvNet architectures. Each architecture is fully convolutional. Each convolution layer has a 3×3 kernel with no dilation. The 'Filters' column indicates the number of output features of each layer. Bold indicates a stride of 2 for a layer, otherwise a stride of 1 is used.

Parameters	Layers	Filters
0.07M	3	32,**64**,**64**
0.14M	4	32,**64**,64,**128**
0.29M	5	32,**64**,64,**128**,128
0.60M	6	32,64,**64**,128,128,**256**
1.20M	7	32,64,64,**128**,128,256,**256**

Training. We evaluate λ values $\{0.0, 0.1, 0.2, \ldots, 1.0\}$ over 5 trials of random training/validation splits and parameter initializations. We use SGD with batch size 128, with learning rate 0.1, momentum 0.9, and weight decay $10^{-}4$. We train for 400 epochs, decaying learning rate by a factor of 10 at epochs 200 and 300, before reporting test error at the epoch at which validation error is minimized.

C Effect of λ on the Condition Number of the Hessian

In this section we show that for $\lambda > 0$, any stationary points—in the special case of scalar model outputs—has a Hessian with both positive and negative eigenvalues, and so all stationary points are saddle points. Further, we show that the condition number of the Hessian grows as λ tends to 1 from below.

For a given input pattern, let the target y be distributed according to a single-parameter exponential family distribution with scalar parameter η. Let $\hat{\eta}_j$ be the parameter value prediction for the jth model of a collection of M models, and let $\bar{\eta} = \frac{1}{M}\sum_{j=i}^{M} \hat{\eta}_j$ be the ensemble prediction. Let $\hat{y}_j = g(\hat{\eta}_j)$ and $\bar{y} = g(\bar{\eta})$ be the conditional mean estimates of the jth model and ensemble model respectively, where g is the canonical inverse link function of the distribution. We have

$$\frac{\partial L_\lambda}{\partial \hat{\eta}_j} = \frac{1}{M}\Big((1-\lambda)\hat{y}_j + \lambda\bar{y} - y\Big), \tag{4}$$

where L_λ is the modular loss, and the entries of the Hessian are given by

$$\frac{\partial^2 L_\lambda}{\partial \hat{\eta}_i \partial \hat{\eta}_j} = \begin{cases} \frac{1}{M}\left(1 - \lambda(1-\frac{1}{M})\right) \cdot g'(\hat{\eta}_i) & \text{if i = j} \\ \frac{\lambda}{M^2} \cdot g'(\bar{\eta}) & \text{otherwise} \end{cases}. \tag{5}$$

From (4), for $\lambda \neq 0$ any stationary point of the loss must have $\hat{y}_i = \hat{y}_j = \bar{y} = y$, and therefore $c \stackrel{\text{def}}{=} g'(\hat{\eta}_i) = g'(\hat{\eta}_j) = g'(\bar{\eta}) = g'(\eta)$ for all i, j, and the Hessian takes the form

$$H = \begin{bmatrix} q & r & r & \dots & r \\ r & q & r & \dots & r \\ r & r & q & \dots & r \\ \vdots & \vdots & & \ddots & \vdots \\ r & r & r & \dots & q \end{bmatrix} \quad , \tag{6}$$

with diagonal entries

$$q = \frac{1}{M}\left(1 - \lambda\left(1 - \frac{1}{M}\right)\right) \cdot c \tag{7}$$

and off-diagonal entries

$$r = \frac{\lambda}{M^2} \cdot c\,. \tag{8}$$

This matrix is $H = r \cdot J_M + (q - r) \cdot I_M$, where J_M is the $M \times M$ matrix of ones and I_M is the $M \times M$ identity matrix. The eigenvalues of J_M are M with multiplicity 1 and 0 with multiplicity $M - 1$. Therefore, the eigenvalues of H are

$$\omega_1 = q + (M - 1) \cdot r = \frac{c}{M} \tag{9}$$

$$\omega_2 = q - r = \frac{c}{M} \cdot (1 - \lambda)\,. \tag{10}$$

From this, we can see that for $\lambda > 1$ the Hessian at the stationary point has both positive and negative eigenvalues, and therefore the stationary point is a saddle point, therefore the models will diverge. Moreover, for $1 > \lambda > 0$, the condition number is

$$\kappa(H) = \frac{\omega_1}{\omega_2} = \frac{1}{1 - \lambda}\,, \tag{11}$$

which tends to infinity as λ tends to 1 from below. This may suggest that optimization may be problematic for λ close to 1, and that we might see significantly different learning behaviour between first- and second-order methods in this regime.

Note that in the case of the Bernoulli distribution, the loss surface with respect to the parameter estimates $\hat{\eta}_j$ has no stationary points, but a similar argument can be made with limits.

D Equivalence of 'Coupled Ensembles' Training Methods

We prove here that the 'LL' and 'SM' coupled training methods of Dutt et al. (2020) 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

$$L^{(m)} = -\sum_k y_k \log q_k^{(m)} \quad . \tag{12}$$

The 'LL' coupled training method of Dutt et al. (2020) has as its loss function L_{LL} the arithmetic mean of the cross entropy loss functions for each network. I.e.,

$$L_{LL} = \frac{1}{M} \sum_m L^{(m)} \quad . \tag{13}$$

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

$$\frac{\partial L_{LL}}{\partial q_k^{(m)}} = \frac{1}{M} \frac{\partial L^{(m)}}{\partial q_k^{(m)}} \quad . \tag{14}$$

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. (2020) 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., 2018) 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

$$\tilde{q}_k = \exp\left(\frac{1}{M} \sum_m \log q_k^{(m)} \right) \quad , \tag{15}$$

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

$$L_{SM} = -\sum_k y_k \log \tilde{q}_k \tag{16}$$

$$= -\sum_k y_k \frac{1}{M} \sum_m \log q_k^{(m)} \tag{17}$$

$$= \frac{1}{M} \sum_m L^{(m)} = L_{LL} \quad . \tag{18}$$

This suffices to demonstrate that the 'LL' and 'SM' methods are equivalent to independent training up to a scaling of learning rate.

Bibliography

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

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

Dutt, A., Pellerin, D., and Quénot, G. (July 2020). Coupled ensembles of neural networks. *Neurocomputing*, 396:346–357.

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

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

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

Huang, G., Liu, Z., v. d. Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, pages 2261–2269.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

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