Efficient Model Averaging for Deep Neural Networks

Michael Opitz, Horst Possegger and Horst Bischof

Institute for Compute Graphics and Vision,
Graz University of Technology,
Graz, Austria
{michael.opitz,possegger,bischof}@icg.tugraz.at

Abstract. Large neural networks trained on small datasets are increasingly prone to overfitting. Traditional machine learning methods can reduce overfitting by employing bagging or boosting to train several diverse models. For large neural networks, however, this is prohibitively expensive. To address this issue, we propose a method to leverage the benefits of ensembles without explicitely training several expensive neural network models. In contrast to Dropout, to encourage diversity of our sub-networks, we propose to maximize diversity of individual networks with a loss function: DivLoss. We demonstrate the effectiveness of DivLoss on the challenging CIFAR datasets.

1 Introduction

Ensemble methods such as bagging [1], boosting (e.g. [2]), or more specifically Random Forests [3], have shown great success in improving generalization performance of machine learning methods. They combine several diverse classifiers to a single predictor, e.g. by averaging their responses. This reduces the generalization error compared to the individual classifiers, since an ensemble of diverse classifiers reduces the variance term in the bias-variance trade-off.

Unfortunately, traditional ensembling methods such as bagging or boosting are prohibitively expensive for neural networks. Large neural networks need several days to train, e.g. [4, 5]. Further, especially for real-world applications with real-time requirements, evaluating an ensemble of several networks at test time is computationally too expensive. Additionally, for systems with low memory capacity, such as embedded systems, employing large ensembles is infeasible.

Previous work [6] proposes Dropout to randomly omit neurons of the hidden layers to implement efficient model averaging for neural networks. This can be interpreted as an efficient combination of an exponential number of different neural networks. However, we found that individual sub-networks trained by Dropout have low diversity. This is due to the fact that these sub-networks share all their parameters with each other and only rely on random feature subsampling to encourage diversity. Further, Dropout is applied on the hidden layers of a neural network.

In contrast to this we focus on efficient model averaging on the output layer of a neural network. To this end, we divide the last hidden layer of a neural network into several, possibly overlapping, groups and optimize a loss function for each of the groups individually rather than over the full output layer. We group the neurons during training such that the ensemble can be mapped back to a regular neural network. By doing this no additional computational cost is incurred at runtime. Instead of relying on sub-sampling of training samples and features, as done by Dropout, we propose a loss function to maximize diversity of our individual network predictors. With this loss function we can effectively balance diversity and discriminativeness of our sub-networks and achieve competitive accuracy to Dropout. We name our method DivLoss.

As our experiments show, sub-networks trained with DivLoss have a larger diversity compared to sub-networks trained with Dropout. Further, we demonstrate that our method can outperform Dropout on the CIFAR-10 and CIFAR-100 datasets. Finally, we show that our method benefits from the decorrelation of hidden units, similar to [7].

The remainder of this paper is structured as follows. In Section 2 we discuss related work. Next, in Section 3 we review preliminaries on learning theory and introduce our DivLoss. In Section 4 we demonstrate effectiveness of our method in several experiments.

2 Related Work

Improving performance of neural networks for supervised learning problems has recently received a lot of attention from the research community. There is a lot of work which is complementary to our method.

A simple, yet effective way to improve accuracy is data augmentation, e.g. [4]. During training, before showing an input sample to the network, a transformation can be applied on the training sample, which preserves the label of the sample. For example mirroring, crops, affine transformations and photometric transformations can be used for image categorization.

Another way to improve neural networks are activation functions. Recently proposed activation functions are more expressive than standard activation functions such as sigmoid or tanh, or are presumably easier to optimize than standard regularization functions, e.g. [8–12].

Since deeper networks are exponentially more expressive than shallow networks, and training very deep networks is challenging due to exploding and vanishing gradients [13], there is a line of work which focuses on enabling training of deeper neural networks. These methods add residual connections or use gating functions from lower to higher layers to enable a better gradient flow in the network and reduce the vanishing and exploding gradient problem [14, 15].

Further, some recent contributions focus on improving optimization algorithms for training deep neural networks. They propose accelerated first-order gradient methods specifically designed for neural networks, e.g. [16–19]. These methods focus on reducing the training time (i.e. fewer iterations), and presumably let the network converge to a better local minimum. Additionally, Ioffe et al. [20] leverage batch statistics to normalize inputs to activation functions. This reduces the internal covariate shift and significantly accelerates training.

Further, there are methods which aim to improve the weight initialization of neural networks. This is especially useful for training very large neural networks, as these models do not converge if the weight initialization is not carefully tuned, e.g. [9, 21–23].

Since networks presumably perform better if their hidden features are discriminative, several methods propose auxiliary loss layers on top of hidden layers to regularize neural networks [24, 25]. Further, Cogswell et al. [7] use auxiliary functions to decorrelate hidden neurons. This enables the network to learn more diverse features and reduces redundancy in the representation of deep networks.

Some methods change the structure of the networks, e.g. by adding additional 1×1 convolutions on top of convolutional layers [26], using layers of multiple scales [27], adding an "Inception" layer, consisting of convolutions of different sizes combined with max-pooling [24] or replacing 5×5 convolutions with 3×3 convolutions [5].

Closely related to our method are contributions which leverage the benefits of ensembles to improve generalization performance of neural networks. Recently, Hinton et al. [28] propose to leverage the "dark knowledge" of neural networks to train a network on the predictions of an ensemble to improve accuracy of the new model. The ensemble predictions are used as soft-labels in combination with the original labels to train a new network achieving better accuracy compared to individual networks of the ensemble. This idea is extended by Romero et al. [29] to train a wide teacher network and a smaller network, which mimics the predictions of the teacher network on the output and hidden layers. In contrast to this kind of work, we leverage the benefits of ensembles without explicitly training several full networks to improve performance of a single neural network. We argue that our method is complementary to these approaches, as better individual predictors result in better ensemble performance. This results in more accurate soft-labels which are useful for these methods.

Another promising line of research focuses on improving accuracy by efficient model averaging. The most prominent work is Dropout [6], which randomly omits hidden units from the network during training. Wan et al. [30] generalize this idea to randomly omit weights of the network during training. Stochastic Pooling [31] introduces a pooling method which samples the activations of the receptive fields, rather than just taking the max or the mean. These methods rely on random noise to increase diversity of neural networks. In contrast to these methods we propose a loss function to increase diversity.

Most closely related to our work is the pioneering work of negative correlation learning [32], which also uses a loss function to reduce correlation of different networks in an ensemble. However, networks trained with negative correlation learning do not share parameters, which is prohibitively expensive for training large neural networks. Further, negative correlation learning focuses on non-computer vision related regression problems and penalizes the correlation of predictions. We show that optimizing cross entropy can achieve better accuracy compared to negative correlation learning for computer vision related classification tasks. Further, we compare this method in a more modern setting, with larger networks, larger datasets and recent contributions such as ReLU activations or Dropout.

3 Towards Efficient Neural Network Ensembles

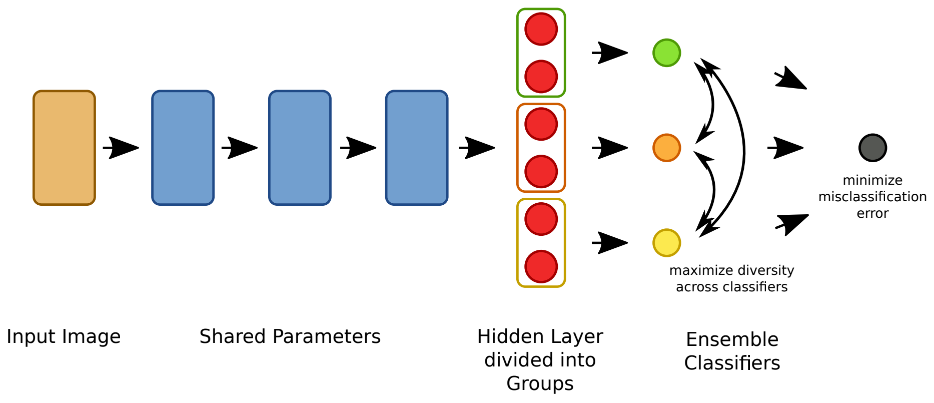

Fig. 1. We divide the last hidden layer into several possibly overlapping groups. Neurons of each group are combined into a classifier. For each of these classifiers we separately optimize a loss which minimizes the training error (e.g. cross-entropy). To increase diversity of the classifiers we add a separate loss between classifiers.

Given a fully annotated dataset, we want to efficiently train a neural network ensemble to obtain a single highly accurate neural network model. However, training and evaluating several independent neural networks on a dataset is computationally expensive, especially for very large networks. Hence, we propose to share most parameters between the individual models, as illustrated in Fig. 1. We divide the last hidden layer into several groups, which is indicated by the respective color. Groups might overlap and share parameters with each other. Further, in contrast to standard ensembles, we train our network ensemble jointly and not sequentially. With this strategy expensive computations for shared parameters can be re-used among different neural network models. Additionally, due to parameter sharing, we can map our networks back to a regular neural network at test time. Hence, DivLoss does not impose any additional computational cost at test time.

As we will discuss in Section 3.1, one key-requirement for ensembles is to reduce correlation among individual models and make them diverse. However, by sharing the feature representation as well as the training set, the individual classifiers will make highly correlated decisions. To address this issue, we propose to maximize the pairwise cross entropy between different classifiers of the ensemble. As we will see, this increases the diversity of classifiers and improves generalization performance.

3.1 Learning Theory

One well-known theoretical result in machine learning is the bias-variance trade-off, e.g. [33]. It states that the generalization error can be decomposed into a bias and variance term. Here, we briefly review the main results of Ueda et al. [34], which analyze the bias-variance trade-off in context of neural network ensembles. For the sake of clarity, we stick to the notation introduced by Ueda et al.

The purpose of learning methods is to construct a model $f(x;\theta)$ that approximates an unknown target function $g(x)$. θ is a parameter vector which is estimated by leveraging a set of i.i.d. samples $z^N = \{z_1, z_2, \dots, z_N\}$, where $z_i = (x_i, y_i)$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ and N is the total number of training samples. z^N is the realization of a random sequence $Z^N = \{Z_1, \dots, Z_N\}$, whose ith component consists of a random vector $Z_i = (X_i, Y_i)$. Hence, each z_i is generated from an unknown joint probability function $p(x, y)$. The parameters θ of the neural network are estimated by an optimization algorithm given the dataset z^N:

$$\hat{\theta}(z^N) = \arg\min_{\theta} \sum_{i=1}^{N} (y_i - f(x_i;\theta))^2/N. \tag{1}$$

Note that since $\hat{\theta}$ depends on a z^N, the estimated predictor $f(x;\hat{\theta}(z^N))$ is also a realization of a random variable $f(x;Z^N)$. Further, Ueda et al. [34] introduces a new random variable $Z_0 = (X_0, Y_0) \in \mathbb{R}^{d+1}$, which has a distribution identical to that of Z_i, but is independent of Z_i for all i. The generalization error of the estimator can then be defined as

$$GErr(f) = E_{Z^N}\left\{E_{Z_0}\left\{\left[Y_0 - f(X_0;Z^N)\right]^2\right\}\right\}, \tag{2}$$

where $E_{Z_0}\{\cdot\}$ and $E_{Z^N}\{\cdot\}$ denotes expectation with respect to the distribution Z_0 and Z^N, respectively. This generalization error decomposes into the well-known bias and variance terms

$$GErr(f) = E_{X_0}\left\{Var\{f|X_0\} + Bias\{f|X_0\}^2\right\} + \sigma^2, \tag{3}$$

where σ^2 denotes the irreducible noise, $Var\{f|X_0 = x_0\}$ and $Bias\{f|X_0 = x_0\}$ are conditional variance and bias given $X_0 = x_0$

$$Var\{f|X_0\} = E_{Z^N}\left\{\left(f(X_0;Z^N) - E_{Z^N}\{f(X_0;Z^N)\}\right)^2\right\}, \tag{4}$$

$$Bias\{f|X_0\} = E_{Z^N}\left\{f(X_0;Z^N)\right\} - g(X_0). \tag{5}$$

For an ensemble, let $f_1, f_2, \dots, f_M$ denote M estimators, where the mth estimator is separately trained on $z^N_{(m)}$, i.e. the training set for the mth estimator, $m = 1, \dots, M$. The output of the ensemble is the average of the estimators:

$$f^{(M)}_{ens}(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x; z^N_{(m)}) \tag{6}$$

Ueda et al. [34] derive the following generalization error of the ensemble estimator $GErr(f^{(M)}_{ens}) =$

$$E_{X_0}\left\{\frac{1}{M}\overline{Var}(X_0) + \left(1 - \frac{1}{M}\right)\overline{Cov}(X_0) + \overline{Bias}(X_0)^2\right\} + \sigma^2, \tag{7}$$

where $\overline{Var}(\cdot)$, $\overline{Bias}(\cdot)$ and $\overline{Cov}(\cdot)$ are variance, bias and covariance of the M estimators, defined as follows

$$\begin{aligned}
\overline{Var}(X_0) &= \frac{1}{M}\sum_{m=1}^{M} Var\{f_m|X_0\},\\
\overline{Cov}(X_0) &= \frac{1}{M(M-1)}\sum_{m}\sum_{m' \neq m} Cov\{f_m, f_{m'}|X_0\},\\
\overline{Bias}(X_0) &= \frac{1}{M}\sum_{m=1}^{M} Bias\{f_m|X_0\}.
\end{aligned} \tag{8}$$

Interestingly, the correlation between individual estimators $\overline{Cov}(X_0)$ is part of this generalization bound. Hence, low correlated and diverse classifiers are desirable to achieve good ensemble performance. Similar results were observed for other popular ensemble methods, such as Random Forests. For this specific learning method Breiman et al. [3] show an upper bound on the error which depends on the strength (i.e. inverse proportional to the bias) and correlation between individual models.

Motivated by these results, in addition to reducing the variance term, we aim to reduce $\overline{Cov}(X_0)$ of our ensemble. Unfortunately, directly minimizing $\overline{Cov}(X_0)$ is impossible, since the distribution of $p(x, y)$ is unknown so we cannot compute expectations over it. Ensemble methods typically subsample the training set or features to reduce correlation among estimators [3]. In the context of neural networks these ideas have been leveraged by Dropout [6], which subsamples different hidden features for each network for each training sample. In contrast to this work, we propose using a loss function to increase the diversity among several networks in the following section.

3.2 Efficient Model Averaging for Deep Neural Networks

To create our individual predictors we divide the last hidden layers into several, possibly overlapping, groups and optimize a loss function for each of these groups separately (recall Fig. 1). The architecture of our ensemble method allows mapping it back to a regular neural network at test time. Hence, by our ensemble method, no additional computational cost is incurred at runtime and negligible additional cost is incurred at training time. Training time is dominated by computing the forward and backward passes of the convolution layers.

For the sake of clarity, to avoid cluttering the notation, we here consider only non-overlapping groups of hidden units. To implement overlapping groups we simply share a subset of weights between classifiers. Let x_i denote the activations of the last hidden layer ($i \in \{1 \dots H\}$) and W the output weight matrix $W \in \mathbb{R}^{H \times D}$ with entries w_{ij}, where H is the number of hidden neurons and D the number of outputs units of the neural network. We group C non-overlapping neurons in the hidden layer to classifiers. For classifiers with softmax activation, we define the logit (i.e. the inputs to the last softmax nonlinearity) c_{bj} of such a classifier as

$$c_{bj} = \sum_{i=(b-1)\cdot C}^{b \cdot C} x_i \cdot w_{ij} + b_{bj}, \tag{9}$$

where b_{bj} denotes the bias term, b is the block index and $j \in \{1, \dots, C\}$ indicates the output class.

We define the ensemble logit as average of the $B = H/C$ individual classifiers

$$o_j = \frac{1}{B} \sum_{b=1}^{B} c_{bj}. \tag{10}$$

The final classifier output is defined as softmax function over o_j. By setting our method up this way, we can map it back to a regular neural network at test time, hence, imposing no additional runtime overhead. For non-overlapping groups we can push the scaling factor $\frac{1}{B}$ back into the last weight matrix W. For overlapping groups we have to scale weights which are used by multiple classifiers by an appropriate scaling factor. The ensemble prediction can then be computed by a simple forward pass. Note that by setting our network up this way, it corresponds to taking the geometric mean of the individual classifier softmax outputs and re-normalizing them to a probability distribution:

$$\begin{aligned} \sigma(o_j) &= \frac{e^{\frac{1}{B} \cdot \sum_{b=1}^{B} c_{bj}}}{Z} = \frac{\left(\prod_{b=1}^{B} e^{c_{bj}}\right)^{\frac{1}{B}}}{Z} \\ &= \frac{\left(\prod_{b=1}^{B} \sigma(c_{bj}) \cdot Z_b\right)^{\frac{1}{B}}}{Z} = \frac{\left(\prod_{b=1}^{B} \sigma(c_{bj})\right)^{\frac{1}{B}}}{\hat{Z}}, \end{aligned} \tag{11}$$

where Z_b denotes the normalization for the softmax activation of the bth classifier, Z denotes the normalization for the softmax of the classifier ensemble and

$$\hat{Z} = \frac{Z}{\left(\prod_{b=1}^{B} Z_b\right)^{\frac{1}{B}}} = \frac{\sum_{j=1}^{D}\left[\left(\prod_{b=1}^{B} Z_b\right)^{\frac{1}{B}}\left(\prod_{b=1}^{B}\sigma(c_{bj})\right)^{\frac{1}{B}}\right]}{\left(\prod_{b=1}^{B} Z_b\right)^{\frac{1}{B}}} \tag{12}$$
$$= \sum_{j=1}^{D}\left(\prod_{b=1}^{B}\sigma(c_{bj})\right)^{\frac{1}{B}}.$$

We see that the geometric mean of the normalizations of the individual classifier, i.e. $\left(\prod_{b=1}^{B} Z_b\right)^{\frac{1}{B}}$, is independent of j, can be pulled out of the sum and cancels with the denominator. Hence, the ensemble output is proportional to the geometric mean of the responses of the individual classifiers.

We want both, our final ensemble and our individual classifiers to be discriminative on our training set. To this end, we minimize the cross entropy on both, the ensemble predictions and the predictions of the individual classifiers by introducing the loss

$$\mathcal{L}_{discr} = \sum_{i=1}^{N}\left(\mathcal{H}(y^{(i)}, \sigma(o^{(i)})) + \lambda_{parts}\cdot\left(\frac{1}{B}\sum_{b=1}^{B}\mathcal{H}(y^{(i)}, \sigma(c_b^{(i)}))\right)\right), \tag{13}$$

where N is the total number of training samples and $y^{(i)}$ is the label of the ith training sample. With a slight abuse of notation $\sigma(o^{(i)})$ denotes the softmax activations of the full ensemble for the ith sample, $\sigma(c_b^{(i)})$ denotes the softmax activation for the ith sample of the bth classifier. The parameter λ_{parts} is a hyperparameter which balances the influence of the individual classifiers and the ensemble and is set by (cross-)validation. We typically sweep it out on a log scale, i.e. $2^{\{0,1,2,3\}}$. Finally, $\mathcal{H}(p, q)$ denotes the cross entropy between probability distributions p and q.

3.3 Enforcing Diversity

Naively applying Equation (13) to a learning problem will result in several individual classifiers, which all have highly correlated predictions. Hence, according to the bias-variance-correlation trade-off there is no benefit in such a setup. To address this problem, we propose to *maximize* the cross entropy between all classifier pairs. Cross entropy is employed in logistic regression and in most neural networks for classification as loss function. It measures the dissimilarity between two probability distributions and is typically used to minimize dissimilarity between ground-truth label and predicted label in supervised learning problems.

In contrast to that, to encourage diversity for different classifiers, we propose to maximize the cross entropy (i.e. maximize dissimilarity or minimize similarity) between all pairs of classifiers. More formally, we define the following loss function:

$$\mathcal{L}_{diversity} = \frac{1}{B \cdot (B-1)} \sum_{i=1}^{N} \sum_{b=1}^{B} \sum_{b' \neq b} -\mathcal{H}(\sigma(c_b^{(i)}), \sigma(c_{b'}^{(i)})), \tag{14}$$

where N is the number of samples, B the number of classifiers, $\sigma(c_b^{(i)})$ denotes the output for the ith sample from the bth classifier and $\mathcal{H}$ is the cross entropy between the two classifiers.

Our final loss function $\mathcal{L}$ is a combination of $\mathcal{L}_{discr}$ and $\mathcal{L}_{diversity}$:

$$\mathcal{L} = \mathcal{L}_{discr} + \lambda_{diversity} \cdot \mathcal{L}_{diversity} \tag{15}$$

where $\lambda_{diversity}$ is a hyperparameter, balancing the influence of the diversity loss and the discriminative loss. The parameter is set by (cross-)validation on a log scale, i.e. $10^{\{2,3,4\}}$. We call this loss function DivLoss, as it encourages diversity between individual predictors of an ensemble.

3.4 Loss Function on Hidden Layers

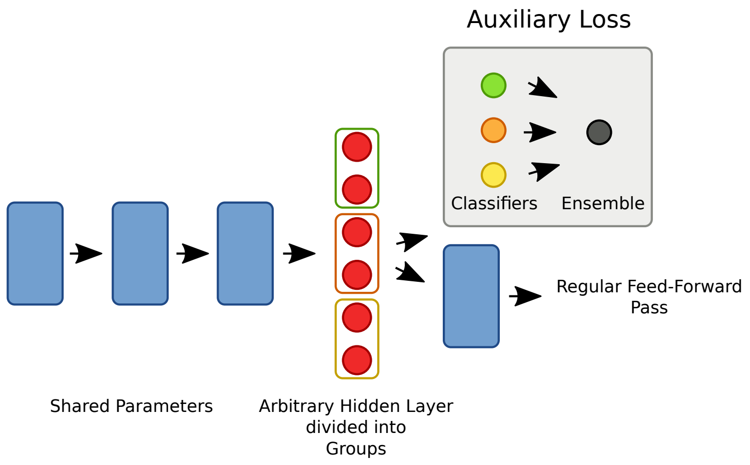
Auxiliary Loss
Classifiers
Ensemble
Regular Feed-Forward Pass
Shared Parameters
Arbitrary Hidden Layer divided into Groups

Fig. 2. We can apply our method on top of any hidden layer in a neural network.

Compared to Dropout, our method is applied only on the output layer of a neural network, and not on an arbitrary hidden layer. For very large networks, however, it might be beneficial to apply regularization already on top of hidden layers. To address this issue, inspired by deeply supervised networks [24, 25], we propose to apply our ensemble layer on top of intermediate hidden layers as auxiliary layer (see Fig. 2). During training time, we can divide any hidden layer of a network into possibly overlapping groups, as indicated by the corresponding color, and optimize our loss on them. The next layer in the regular feed forward pass receives all neurons from this hidden layer as input (i.e. it does not operate on individual groups).

Note that this setup does not introduce any additional computational cost during test time, since these auxiliary layers are just used during training time, and not during test time. In Section 4.3 we show that this setup can indeed improve accuracy.

4 Evaluation

In this section we provide a detailed evaluation of our method on CIFAR-10 and CIFAR-100 [35]. These datasets each consist of $50,000$ training images and $10,000$ test images of size 32×32. CIFAR-10 has 10 object classes, wheras CIFAR-100 has 100 object classes. Both datasets have a uniform class distribution, i.e. there are $6,000$ images per class in CIFAR-10, from which $1,000$ are in the test set, and 600 images per class in CIFAR-100, from which 100 are in the test set. For pre-processing, following [7], we subtract the mean of the training set from the images.

We run our experiments on a regular desktop machine with a NVIDIA GTX 770 GPU and a Core i5-4570 CPU with 3.20 GHz and implement our method in Theano [36]. For training parameters (learning rate, momentum, weight decay) we use the standard Caffe learning parameters for the CIFAR-10 Quick architecture. As network architecture we use a larger version of the CIFAR-10 Quick architecture, which is proposed by Cogswell et al. [7]. The architecture is C-$64 \times 5 \times 5$, MP-$3 \times 3(2 \times 2)$, C-$64 \times 5 \times 5$, AP-$3 \times 3(2 \times 2)$, C-$128 \times 5 \times 5$, AP-$3 \times 3(2 \times 2)$, FC-128, FC-128, FC-D, where C-$F \times S \times S$, denotes a convolution layer with F filters of size $S \times S$, MP-$N \times N(S \times S)$ denotes a max pooling layer of size $N \times N$ with stride $S \times S$, AP-$N \times N(S \times S)$ denotes an average pooling layer of size $N \times N$ with stride $S \times S$, and FC-N denotes a fully connected layer of size N. Each layer except the last two fully connected layers are followed by a ReLU nonlinearity. The last fully connected layer is our output layer, which has an output dimensionality of $D = 10$ in our CIFAR-10 experiments and a dimensionality of $D = 100$ in our CIFAR-100 experiments and uses a softmax nonlinearity. The last *hidden* layer uses no nonlinearity (i.e. is a linear layer).

4.1 Comparison with Negative Correlation Learning

In this section we compare maximizing cross-entropy loss between classifiers to minimizing the correlation, as proposed by negative correlation learning [32]. Hence, we directly penalize the correlation of the ensemble for a training sample with the following function:

$$\frac{1}{D} \sum_{j=1}^{D} \sum_{b=1}^{B} \sum_{b' \neq b} (\sigma(c_{bj}) - \sigma(o_j)) \cdot (\sigma(c_{b'j}) - \sigma(o_j)), \tag{16}$$

where D is the number of classes, c_{bj} are logits of the bth classifier for the jth class and o_j is the logit of the ensemble output. We also experimented by directly penalizing the logits as opposed to their softmax activations, but achieved best results with the above formulation. We compare negative correlation learning to the cross entropy loss and the absolute correlation, i.e.:

$$\frac{1}{D}\sum_{j=1}^{D}\sum_{b=1}^{B}\sum_{b'\neq b}\left|(\sigma(c_{bj})-\sigma(o_j))\cdot(\sigma(c_{b'j})-\sigma(o_j))\right|. \tag{17}$$

We run the experiments on CIFAR-10 and summarize the results in Table 1. We observe more accurate results for maximizing cross-entropy than for minimizing correlation. We hypothesize that this is because cross-entropy is a more natural fit to measure diversity for classifiers which output a probability distribution compared to correlation. Further, penalizing the absolute value of the correlation works better for classification problems, since it encourages classifers to be weakly correlated, as opposed to negatively correlated.

Table 1. Comparison to Negative Correlation Learning on CIFAR-10.

Method	Test Acc.
Cross Entropy	**82.3**
Absolute Correlation	81.28
Negative Correlation	80.9
Baseline	80.86

4.2 Diversity

In this section we analyze the effect on the diversity of our ensemble and compare it to a network trained with Dropout. To measure diversity, we count the number of disagreements of all classifier pairs.

$$\frac{0.5}{B\cdot(B-1)}\sum_{b=1}^{B}\sum_{b'\neq b}\frac{1}{N}\sum_{i=1}^{N}f_b(x_i)\neq f_{b'}(x_i), \tag{18}$$

where B is the number of classifiers and $f_b(x_i)$ is the bth classifier output for the ith sample (i.e. the label prediction). The higher this number, the more diverse the classifier outputs are.

Since Dropout is an approximate average of an exponential number of neural networks, we sub-sample 16 sub-networks and analyze their correlation on the validation set. We execute this experiment 10 times and compare this to a network trained with our method consisting of 16 sub-networks on CIFAR-10. We report the diversity of individual sub-networks, the average accuracy of these 16 individual sub-networks and the accuracy of the full ensemble in Table 2. Since for the Dropout experiments, we sub-sample 16 sub-networks and repeat the experiment 10 times, we report mean and standard deviation of the diversity and the average accuracy of the individual sub-networks.

Interestingly, in Table 2 we see that our method trains sub-networks which are more diverse on the validation set compared to Dropout. Further, individual sub-networks are less accurate compared to Dropout, but complement each other better, since we tie them together with a global loss.

Table 2. Diversity of a network trained with Dropout and our method on CIFAR-10.

Method	Diversity	Avg. Sub-Network Acc.	Ensemble Acc.
Dropout	0.071 ± 0.0025	**0.799** ± 0.00073	81.07
Ours	**0.240**	0.744	**82.3**

4.3 CIFAR-10

In this section, we evaluate our method on the CIFAR-10 dataset. To make a fair comparison, we use the same architecture as proposed in [7], i.e. we double the number of hidden units and convolution filters of the Caffe 10 Quick architecture and add an additional fully connected layer to our network. This architecture will be denoted "Baseline". We split the training set into $10,000$ validation images and $40,000$ training images to determine hyperparameters (i.e. our weighting parameter, dropout rates, DeCov [7] hyperparameters). Our method benefits from a large number of non-overlapping groups, as diversity can be easier maximized if no parameters are shared among groups. To enable a fair comparison to existing work, we fix the hidden layer size to 128. We divide the last hidden layer into non-overlapping groups with 8 hidden units, as sub-networks with a smaller number of hidden units fail in our experiments to learn anything meaningful. As in [7] our network takes 32×32 patches as input and we do not apply any kind of data augmentation. We shuffle the training dataset after each epoch and employ early stopping.

We apply our ensembling method on top of the output layer of the neural network and report our results in Table 3. We see that our method can significantly outperform Dropout [6] and achieves similar results to DeCov [7]. Additionally, our method can benefit from DeCov as well as Dropout. We hypothesize that DeCov helps a neural network to develop more decorrelated features, which help building more diverse classifiers. With Dropout the generalization performance of individual networks of our ensemble increases, hence the performance of the full ensemble improves. For a fair comparison, we also apply re-shuffling and early stopping to DeCov [7], which improves the overall accuracy by 0.38.

To show that our method works on auxiliary layers, we additionally apply our loss on the first fully connected layer. We observe a notable increase in accuracy from 82.3 to **83.44** for our method.

4.4 CIFAR-100

We re-use the same network architecture for the CIFAR-100 experiment. Since the number of hidden units (128) is quite small compared to the number of

Table 3. CIFAR-10 classification accuracy.

Method	Test Acc.
DeCov [7]	81.68
DeCov + re-shuffling & early stopping	82.06
Baseline	80.86
Dropout	81.07
DivLoss	82.3
DivLoss + Dropout	82.52
DivLoss + Decov	**82.95**

classes, we perform weight sharing for our classifiers. We fix the number of hidden units of a single classifier to 64 and randomly group hidden units to a classifier. We use 16 classifiers in our experiments. Further, we split the dataset into $10,000$ validation images and $40,000$ training images and determine our hyperparameters (i.e. the weighting parameter, dropout rates, DeCov hyperparameter) on the validation set.

Our results are summarized in Table 4. All 3 regularization methods (DivLoss, Dropout, DeCov) achieve similar results on CIFAR-100 and can significantly improve over a baseline method which just uses weight decay as regularization. Further, we observe that we can combine DivLoss with Dropout and DeCov, to increase accuracy.

Table 4. CIFAR-100 classification accuracy.

Method	Test Acc.
DeCov [7]	45.10
DeCov + re-shuffling & early stopping	49.61
Baseline	47.38
Dropout	49.44
DivLoss	49.42
DivLoss + Dropout	49.9
DivLoss + DeCov	**50.08**

When we additionally apply our loss function as auxiliary layer on the first fully connected layer, we observe an increase in accuracy from 49.42 to **50.32**.

5 Conclusion

We proposed an ensemble method which improves the generalization performance of neural networks by efficient model averaging. Motivated by learning theory, we propose to optimize a loss function to increase the diversity of the individual classifiers of the ensemble. Our method can be trained end-to-end with stochastic gradient descent and momentum. Further, we showed that our method outperforms or achieves competitive performance compared to Dropout and DeCov on the challenging CIFAR datasets. Since we setup our method so that it can be mapped back to a regular neural network, no additional runtime cost is incurred at test time. At training time we impose negligible additional runtime cost for computing the responses and the loss for our sub-networks. This overhead is, however, negligible, since most of the time during training is spent computing forward and backward passes of convolution layers.

Our experiments show that our method benefits especially from very wide networks where the number of hidden units is large compared to the number of classes. In such networks diversity of sub-networks can be better maximized as they have less shared parameters.

Compared to Dropout, which is an approximate ensemble of exponentially many classifiers sharing the same parameters, our method relies on a smaller number of classifiers. To enforce diversity, Dropout relies on randomly omitting neurons from the hidden layers. In contrast to that, our method employs a loss function to encourage diversity of individual classifiers. Due to backpropagation, the diversity also affects the hidden layers (i.e. the feature representation) of the network and, similar to Dropout, encourages a diverse feature representation.

Future work will analyze larger sub-networks consisting of several layers with separate (non-shared) weights.

Acknowledgement. This work was supported by the Austrian Research Promotion Agency (FFG) project Vision+.

References

1. Breiman, L.: Bagging Predictors. Machine Learning (ML) **24** (1996) 123–140
2. Freund, Y., Schapire, R.E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences (JCSS) **55** (1997) 119 – 139
3. Breiman, L.: Random Forests. Machine Learning (ML) **45** (2001) 5–32
4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems (NIPS). (2012)
5. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: Proceedings of the International Conference on Learning Representations (ICLR). (2014)
6. Nitish Srivastava and Geoffrey Hinton and Alex Krizhevsky and Ilya Sutskever and Ruslan Salakhutdinov: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research (JMLR) **15** (2014) 1929–1958
7. Cogswell, M., Ahmed, F., Girshick, R.B., Zitnick, L., Batra, D.: Reducing Overfitting in Deep Networks by Decorrelating Representations. In: Proceedings of the International Conference on Learning Representations (ICLR). (2016)
8. Nair, V., Hinton, G.E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: Proceedings of the International Conference on Machine Learning (ICML). (2010)
9. He, K., Zhang, X., Ren, S., Sun, J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV). (2015)

10. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier Nonlinearities Improve Neural Network Acoustic Models. In: Proceedings of the International Conference on Machine Learning (ICML). (2013)
11. Goodfellow, I., Warde-farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout Networks. In: Proceedings of the International Conference on Machine Learning (ICML). (2013)
12. Clevert, D., Unterthiner, T., Hochreiter, S.: Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In: Proceedings of the International Conference on Learning Representations (ICLR). (2016)
13. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies. In Kremer, S.C., Kolen, J., eds.: Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
14. Srivastava, R.K., Greff, K., Schmidhuber, J.: Training Very Deep Networks. In: Advances in Neural Information Processing Systems (NIPS). (2015)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. arXiv:1512.03385 (2015)
16. Duchi, J., Hazan, E., Singer, Y.: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research (JMLR) **12** (2011) 2121–2159
17. Tieleman, T., Hinton, G.: Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning (2012)
18. Zeiler, M.D.: ADADELTA: An Adaptive Learning Rate Method. arXiv:1212.5701 (2012)
19. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Proceedings of the International Conference on Learning Representations (ICLR). (2015)
20. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Proceedings of the International Conference on Machine Learning (ICML). (2015)
21. Mishkin, D., Matas, J.: All You Need is a Good Init. In: Proceedings of the International Conference on Learning Representations (ICLR). (2016)
22. Glorot, X., Bengio, Y.: Understanding the Difficulty of Training Deep FeedForward Neural Networks. In: Artificial Intelligence and Statistics Conference (AISTATS). (2010)
23. Krähenbühl, P., Doersch, C., Donahue, J., Darrell, T.: Data-dependent Initializations of Convolutional Neural Networks. In: Proceedings of the International Conference on Learning Representations (ICLR). (2016)
24. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going Deeper with Convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2015)
25. Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-Supervised Nets. In: Artificial Intelligence and Statistics Conference (AISTATS). (2015)
26. Lin, M., Chen, Q., Yan, S.: Network in Network. In: Proceedings of the International Conference on Learning Representations (ICLR). (2014)
27. Sermanet, P., LeCun, Y.: Traffic Sign Recognition with Multi-Scale Convolutional Networks. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN). (2011)
28. Hinton, G., Vinyals, O., Dean, J.: Distilling the Knowledge in a Neural Network. arXiv:1503.02531 (2015)

29. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: FitNets: Hints for Thin Deep Nets. In: Proceedings of the International Conference on Learning Representations (ICLR). (2015)
30. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of Neural Networks using DropConnect. In: Proceedings of the International Conference on Machine Learning (ICML), JMLR Workshop and Conference Proceedings (2013)
31. Zeiler, M.D., Fergus, R.: Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. In: Proceedings of the International Conference on Learning Representations (ICLR). (2013)
32. Liu, Y., Yao, X.: Ensemble Learning via Negative Correlation. Neural Networks **12** (1999) 1399–1404
33. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. 2 edn. Wiley, New York (2001)
34. Ueda, N., Nakano, R.: Generalization Error of Ensemble Estimators. In: Proceedings of the IEEE International Conference on Neural Networks (ICNN). (1996)
35. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images (2009)
36. The Theano Development Team: Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688 (2016)