Exploiting Ensemble Diversity for Automatic Feature Extraction

Gavin Brown, Xin Yao, Jeremy Wyatt, Heiko Wersing, Bernhard Sendhoff

2002

Preprint:

This is an accepted article published in Proceedings of the 9th International Conference on Neural Information Processing - ICONIP. The final authenticated version is available online at: https://doi.org/[DOI not available]

EXPLOITING ENSEMBLE DIVERSITY FOR AUTOMATIC FEATURE EXTRACTION

Gavin Brown, Xin Yao, Jeremy Wyatt

School of Computer Science,
University of Birmingham,
Birmingham, B15 2TT,
United Kingdom

Heiko Wersing, Bernhard Sendhoff

Honda R&D Europe (Germany) GmbH,
Carl-Legien-Str.30,
63073 Offenbach/Main,
Germany

ABSTRACT

We present an automatic method, based on a neural network ensemble, for extracting multiple, diverse and complementary sets of useful classification features from high-dimensional data. We demonstrate the utility of these *diverse representations* for an image dataset, showing good classification accuracy and a high degree of dimensionality reduction. We then outline a number of possible extensions to the project in an evolutionary computation context.

1. INTRODUCTION

1.1. Ensemble Learning

Neural network ensembles offer a number of advantages over a single neural network system. They have the potential for improved generalization, lower dependence on the training set, and reduced training time. Sharkey [8] provides a excellent summary of the literature up to 1998, while Dietterich [3] summarises up to 2000.

Training a neural network generally involves a delicate balance of various factors. The *bias-variance decomposition* [4] states that the mean square error of an estimator (in our case, a neural network) is equal to the bias squared plus the variance. There is a trade-off here — with more training, it is possible to achieve lower bias, but at the cost of a rise in variance. Krogh and Vedelsby [5] extend this concept to ensemble errors, showing how the bias can be seen as the extent to which the averaged output of the ensemble members differs from the target function, and the variance is the extent to which the ensemble members disagree. Ueda and Nakano [9] further provide a detailed proof of how the decomposition can be extended to *bias-variance-covariance*. From this result, one way to decrease the error is clear: decrease the covariance, ideally making it strongly negative - though too large a decrease in covariance can cause a rise in bias and variance. This means that an ideal ensemble consists of highly correct classifiers that disagree as much as possible (balancing the covariance against the bias and variance), empirically verified by Opitz and Shavlik [7] among others. Such an idea has also been shown using rule-based systems in an evolutionary context, demonstrating the generality of the idea and its associated techniques.

Negative Correlation (NC) Learning [6] is an efficient ensemble training method which can easily be implemented on top of standard backpropagation in feedforward networks. It incorporates a measure of ensemble diversity into the error function of each network: thus each network not only decreases its error on the function, but also increases its diversity from other network errors. The procedure has the following form: take a set of neural networks and a training pattern set, each pattern in the set is presented and backpropagated on, *simultaneously*, by the networks.

In the standard backpropagation algorithm, the error function for the output layer nodes is

$$\frac{1}{2}(F_i(n) - d(n))^2,$$

where $F_i(n)$ is the output of network i on pattern n, and $d(n)$ is the desired response for that pattern. In NC-learning, the error function becomes

$$\frac{1}{2}(F_i(n) - d(n))^2 + \lambda p_i(n), \tag{1}$$

where $p_i(n)$ is

$$(F_i(n) - F(n)) \sum_{j \neq i} (F_j(n) - F(n)), \tag{2}$$

and λ is an adjustable strength parameter for the penalty. $F(n)$ is the output of the ensemble on pattern n. A common ensemble output function (and used throughout this paper) is a simple average of the networks in the ensemble, i.e.,

$$F(n) = \frac{1}{N} \sum_{i=1}^{N} F_i(n). \tag{3}$$

Proc. Int. Conf. Neur. Inf. Proc. - ICONIP, vol.4, pp. 1786-1790, 2002

In this case we have an overall error function of

$$\frac{1}{2}(F_i(n) - d(n))^2 - \lambda(F_i(n) - F(n))^2 \qquad (4)$$

As can be seen from (4), each network receives lower error for moving its response closer to the target response, and away from the mean response of all the other networks — this is a trade-off, controlled by the penalty strength parameter, λ. When $\lambda = 0.0$, the networks ignore the other errors, and this is termed *independent training*, equivalent to not using NC at all.

Previous work [1] has shown the effectiveness of this technique under different circumstances.

1.2. Motivation

During ensemble training, each network is encouraged, either directly through NC-learning, or indirectly through the random weight initialisation, to specialise to a different part of the dataset. These networks will be forming internal representations of the data in their hidden layer. Since they have specialism to different portions of the problem, they are likely to have developed different representations. Such diverse representations can be used effectively as features in further classification steps. In essence, an ensemble is used here as a feature extractor, rather than as a classifier.

Although single neural networks have been used in feature extraction and dimension reduction, This is the first time, to our best knowledge, neural network ensembles have been used in feature extraction. In the rest of this paper, we will describe in detail the method and algorithm we proposed. Experimental results are also presented to demonstrate the effectiveness of our proposed method and algorithm. It is interesting to observe that our automatically extracted features performed similarly to those extracted manually by human beings based on neuroscience knowledge.

2. DATASET

The dataset used in our experimental study is based on a subset of the COIL-100 database. We used only the first 20 objects in order to facilitate comparison with the previous work. Previous work by Wersing and Korner [10] investigated how a hierarchy of biologically inspired feature detectors can create a useful re-representation of raw image data. Figure 2 illustrates the basic idea, though for more details the reader is referred to the original paper [10].

The input to the hierarchy is the original grayscale image data, 128 by 128 pixels. An example is shown in figure 1.

The first stage of processing applies Gabor filters at four orientations, followed by spatial pooling, producing four images of 16 by 16 pixels — this is the 'C1' data, a vector of length 1024. The second stage applies their technique,

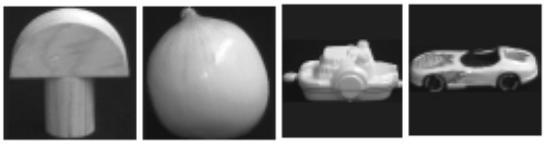

Figure 1: Some images from the COIL-100 database

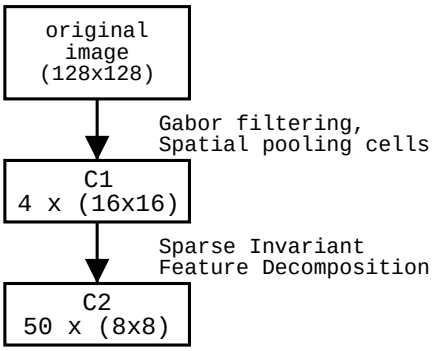

Figure 2: Wersing and Korner's hierarchy

sparse invariant feature decomposition, producing fifty 8 by 8 images, so a vector of length 3200 — this is the 'C2' data. This is then used in a classification technique such as k-Nearest Neighbour.

This architecture demonstrated very good performance, attributed to the feature detectors being localized to particular portions of the image. The disadvantage lies in the large amount of time spent on design and tuning of various parameters, including the connection patterns and size of localised fields. In this paper we try to remove a large portion of the design step by automatically encouraging specialisation to features within the image. We perform a classification on a set of features extracted from the C1 data, and compare with the same classification technique on the C2 data.

The C1 data was rescaled to be between 0 and 1. Each object has 72 views at rotations of 5 degrees. We used 4 views for training, and 68 for testing, giving a total of 80 vectors for training and 1360 vectors for testing. We used standard backprop with a learning rate of 0.1. Training was stopped when the reduction in MSE was less than 10^{-5}, over the previous 25 iterations. We performed 100 trials of each experiment, from random initial weights.

Our benchmark performance throughout these experiments will be the testing rate achieved by the k-Nearest Neighbour classifier, with $k = 1$, on the raw C1 data and C2 data. With 4 training views at 0, 90, 180, and 270 degrees, testing rate on C1 data was 1151 from 1360, so 84.6%. For the C2 data, it was 1176, or 86.4%.

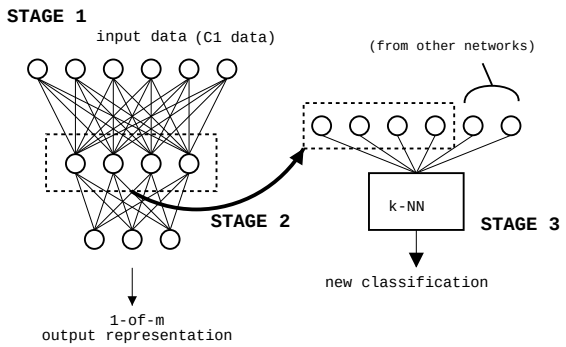

Figure 3: Stages of the node extraction process

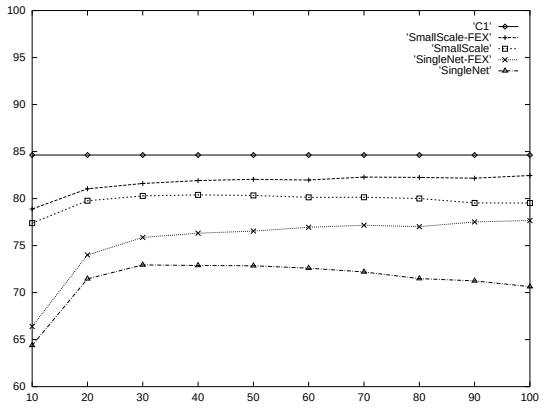

Figure 4: Testing classification rate for SingleNet versus SmallScale, varying the number of hidden nodes

3. EXTRACTING FEATURES

As preliminary work we used standard Multilayer Perceptrons (MLPs) to classify the data, with three experimental setups. The first was a single MLP, where we vary the number of hidden nodes, we label this *SingleNet*. The second was an ensemble of 3 networks, and again we vary the number of hidden nodes per network, we label this *Smallscale*. The third was an ensemble of networks each with 30 hidden nodes, where we vary the number of networks, we label this *Largescale*. Results are summarised in figure 4 and compared to results using the hidden node extraction method, which we term *FEX*, for *Feature EXtraction*.

3.1. Method

In this set of experiments we take the hidden node representation of each pattern to a further classification stage. By this we mean we train the network initially, then pass each pattern through the first layer of weights, recording the hidden node activations, which will be the new *intermediate* representation. We feed this to a k-Nearest Neighbour classifier, though in principle any statistical classifier could be applied. We use the *SingleNet*, *Smallscale*, and *Largescale* setups, taking all hidden nodes outputs from all networks as the re-representation of the pattern. For example if we have 30 networks each with 30 hidden nodes, the k-NN classifier will work on vectors of length 900. Figure 3 illustrates three stages. The first is to train an ensemble of MLPs towards a unary representation. The second stage is to take the hidden node outputs and feed them to another classifier. The third stage performs a k-Nearest Neighbour classification ($k = 1$) on the new representation.

3.2. Results

Figure 4 shows a comparison between testing performance on single networks and ensembles, varying the number of hidden nodes per network. The C1 data is of dimension 1024, and with a kNN can provide 84.6% performance. The features extracted from the SmallScale ensemble (100 hidden nodes in each of 3 nets) obtained a comparable 82.5% (over 100 trials), with a maximum performance equalling the C1 data at 84.6% but with a dimension of just 300. Feature extraction from a single network with 90 hidden nodes obtained 77.5%, while using an ensemble with 3 nets each with 30 hidden nodes obtained 81.6%. This result shows clearly the advantage of extracting hidden nodes from an ensemble rather than a single network.

Table 1 shows performance of *LargeScale* with and without FEX, for 10, 20 and 30 networks each of 30 hidden nodes. We can see comparable performance to the C1 data obtaining on average 84.2% with the 30x30 architecture. We can also see peak performance *exceeding* the original C1 data when using extracted features, at 85.4% on the 10x30 architecture. This was again with only 300 features rather than the 1024 of the C1 data.

	10	20	30
LargeScale	82.9 (84.2)	83.3 (84.5)	83.5 (84.7)
LargeScale-FEX	83.7 (85.4)	84.0 (85.3)	84.2 (85.4)

Table 1: *LargeScale* testing accuracy, mean and (max) over 100 trials

4. PRUNING FEATURES

4.1. Method

In this section we attempt to selectively prune subsets of networks and hidden nodes that may be redundant. In this way we can reduce the storage space necessary for the k-NN classifier. The pruning occurs at stage 2, as illustrated in figure 3, after we have trained an ensemble. The following algorithm is applied:

```
for each output class c
  pick candidate net(s) with low MSE
  for each candidate network
    identify nodes with high contribution
  endfor
  add hidden nodes to feature set
endfor
```

For each class we first identify one or more candidate networks that may be useful for correct classification. We choose the networks with lowest MSE on instances of this class. This has the effect of pruning away any networks which are damaging to the overall ensemble performance. Next we identify hidden nodes within this set of candidates that contribute most to the ensemble performance (termed the *contribution to activation*). This is done by noting the activation of the relevant output node, with and without each hidden node. The nodes which cause the largest change in activation are selected as useful for classification.

If two particular classes are found to utilise the same hidden node, an option is to include this twice in the extracted features. Preliminary work revealed that this does not seem to be a useful step, so repeated nodes were omitted from the final experiments.

At the end of this algorithm, a subset of the hidden nodes from various networks in the ensemble will have been selected. It is not necessarily the case that all networks will contribute hidden nodes.

We note that many other algorithms to calculate *contribution to activation* could be used here. One potential is to measure the correlation between the inputs and outputs of the hidden nodes, which would be a non-linear, non-monotonic correlation that requires careful consideration. Another is to allow a co-evolutionary algorithm to identify nodes. This is discussed for future work in the conclusions of this paper.

4.2. Results

We vary the M parameter, which is the percentage of hidden nodes picked from a given specialist network for a particular class. In our experiments we picked one specialist per class.

Table 2 shows performance for various ensemble architectures for different M parameters, while table 3 corresponds to it and shows the actual number of features extracted from the setups.

The last column in table 2, labelled '100' means 100% of hidden nodes were chosen per specialist net that was selected. It may be the case that not all networks are chosen to be specialists, so this allows a pruning of unnecessary networks. The 10x30 architecture (10 nets each with 30 hidden nodes) needs only 251 features on average (over 100 trials) to obtain 83.5% accuracy. This can be compared to the 84.6% of using the C1 data with 1024 features. The dimensionality is cut in quarter, while only sacrificing 1.1% accuracy.

We can see from tables 2 and 3 that taking just 10% of nodes from each network with the 30 network ensembles allows a significant reduction in the number of necessary features, while only conceding 2 or 3 percent accuracy. This result shows the robustness of the automatically extracted features.

M	10	25	50	75	100
1x30	75.1	75.9	75.9	75.9	75.9
1x90	77.0	77.5	77.5	77.5	77.5
1x100	77.2	77.7	77.7	77.7	77.7
1x300	79.8	80.5	80.5	80.5	80.5
3x30	78.7	81.0	81.5	81.6	81.6
3x100	80.3	81.9	82.4	82.4	82.5
3x300	81.8	83.0	83.2	83.2	83.2
10x30	79.7	82.3	83.1	83.4	83.5
20x30	80.0	82.6	83.3	83.6	83.8
30x30	80.0	82.5	83.5	83.7	83.9

Table 2: Average accuracy (100 trials) of FEX on different ensemble topologies, applied with node pruning (M parameter)

M	10	25	50	75	100
1x30	25.9	29.4	30.0	30.0	30.0
1x90	76.9	86.9	90.0	90.0	90.0
1x100	84.7	96.6	100.0	100.0	100.0
1x300	215.0	280.1	300.0	300.0	300.0
3x30	44.0	70.8	86.1	88.2	89.7
3x100	145.2	237.0	291.3	297.2	300.0
3x300	392.7	676.9	876.4	894.3	900.0
10x30	53.9	110.3	184.0	218.8	251.7
20x30	56.5	121.9	225.3	288.8	353.7
30x30	57.5	127.2	245.1	327.2	414.2

Table 3: Average number of features extracted (100 trials), with node pruning (corresponds to table 2)

Table 4 shows the accuracy of ensembles with 30 hidden nodes, as we vary the λ parameter. It shows the accuracy increases when we apply the NC algorithm, proving most useful for the larger ensembles. Table 5 shows the number of features extracted from these setups.

Best performance was observed at 85.4% from a 10 network ensemble with $\lambda = 1.0$. This has exceeded the accuracy on the C1 data by 0.8% and exceeded the accuracy when not using NC learning by 1.9%.

Nets	0.0	0.25	0.75	1.0
10	83.5	83.5	83.6	83.8
20	83.8	83.8	83.8	84.0
30	83.9	83.9	83.9	84.2

Table 4: *Largescale-FEX*, varying λ, testing accuracy

Nets	0.0	0.25	0.75	1.0
10	251.6	248.6	262.1	265.2
20	353.7	359.3	380.5	392.8
30	414.2	414.9	437.9	445.8

Table 5: *Largescale-FEX*, varying λ, average number of features extracted

5. CONCLUSIONS

The technique demonstrated has shown comparable classification accuracy to the existing method, though in a significantly lower dimensionality. Use of NC learning was shown to increase accuracy further on feature extraction from the larger ensembles, though not as effectively on the smaller ensembles.

The feature extraction process was applied to single networks and ensembles with equivalent total numbers of hidden nodes. *In all cases, N features extracted from an ensemble of small networks were found to be significantly more useful for classification than N features extracted from a single large network.*

The feature pruning process was found to be successful, at best on the 30 network ensemble reducing a 1024 dimension vector down to just 127, only conceding 2.1% in classification accuracy. Although the FEX technique could not match the accuracy that the C2 data provided, the best observed performance did come close at only 1% less than C2, *using less than one third of the number of features.*

Further work will consist of applying the technique to lower dimensional problems, so the number of hidden nodes can match the number of inputs and explore the re-representation issue rather than the dimensionality reduction issue.

The choice of networks is an obvious step where an evolutionary algorithm could be implemented. Previous work [11] has evolved neural networks for various tasks, showing exceptional resilience to manipulation of hidden nodes. Evolving the choice and combinations of nodes seems an ideal task for a co-evolutionary environment. In this setup an individual would be a certain combination of nodes, and the fitness would be calculated with reference to other combinations in the population. One possible fitness function would be a measurement of how many other individuals in the population get the same subset of patterns correct. This investigation could be well grounded in previous work on implicit fitness sharing [2].

6. REFERENCES

[1] Gavin Brown and Xin Yao. On the effectiveness of negative correlation learning. In *Proceedings of First UK Workshop on Computational Intelligence*, pages 57–62, 2001. Edinburgh, Scotland.

[2] Paul Darwen and Xin Yao. Every niching method has its niche: fitness sharing and implicit sharing compared. In *Proc. of Parallel Problem Solving from Nature (PPSN) IV - Lecture Notes in Computer Science 1141*. Springer-Verlag, 1996.

[3] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proceedings of First International Workshop on Multiple Classifier Systems (MCS 2000)*, pages 1–15, 2000.

[4] Stuart Geman, Elie Bienenstock, and Rene Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[5] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation and active learning. *Advances in Neural Information Processing Systems (NIPS-7)*, 7, 1995.

[6] Yong Liu. *Negative Correlation Learning and Evolutionary Neural Network Ensembles*. PhD thesis, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1998.

[7] David Opitz and Jude Shavlik. Generating accurate and diverse members of a neural-network ensemble. *Advances in Neural Information Processing Systems*, 8, 1996.

[8] Amanda Sharkey. *Multi-Net Systems*, chapter Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems, pages 1–30. Springer-Verlag, 1999.

[9] N. Ueda and R. Nakano. Generalization error of ensemble estimators. In *Proceedings of International Conference on Neural Networks (ICNN96)*, pages 90–95, 1996.

[10] H. Wersing and E. Körner. Unsupervised learning of combination features for hierarchical recognition models. In *Int. Conf. Artif. Neur. Netw. ICANN*, 2002. accepted.

[11] Xin Yao and Yong Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.