The Profit of Learning Exceptions*

Antal van den Bosch, Ton Weijters, Jaap van den Herik
University of Maastricht, MATRIKS, Department of Computer Science
PO Box 616, NL-6200 MD Maastricht, the Netherlands

Walter Daelemans
Institute for Language Technology and AI (ITK),
Tilburg University, PO Box 90153, NL-5000 LE Tilburg, the Netherlands

Abstract

For many classification tasks, the set of available task instances can be roughly divided into regular instances and exceptions. We investigate three learning algorithms that apply a different method of learning with respect to regularities and exceptions, viz. (i) back-propagation, (ii) cascade back-propagation (a constructive version of back-propagation), and (iii) information-gain tree (an inductive decision-tree algorithm). We compare the bias of the algorithms towards learning regularities and exceptions, using a task-independent metric for the typicality of instances. We have found that information-gain tree is best capable of learning exceptions. However, it outperforms back-propagation and cascade back-propagation only when trained on very large training sets.

1 Introduction

For many classification tasks, the set of available task instances can be roughly divided into a *core* of regular instances and a *periphery* consisting of exceptions. A successful learning algorithm must be able to learn the core

*In: Proceedings of Benelearn-95, TR/IRIDiA/95-16. Brussels: ULB. pp. 118–126

as well as the periphery. If one wishes to investigate the learnability of a set of instances of which a large portion is peripherial, a metric is needed that decides whether an instance belongs to the core or to the periphery. The metric should be independent of any model or theory of the task. Zhang (1992) presents a metric which can be used for this purpose. He defines the *typicality* of an instance i with classification c as i's average similarity to instances with the same classification c, divided by its average similarity to instances with another classification than c. Regular instances have typicalities larger than 1; exceptions have typicalities smaller than 1.

Given two instances, i_1 and i_2, each having n features, the similarity of these instances, $sim(i_1, i_2)$, is formalised by Zhang (1992) as

$$sim(i_1, i_2) = 1 - \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\frac{i_1^j - i_2^j}{max^j - min^j}\right)^2}$$

where i_1^j is the value of the jth feature of instance i_1, and max^j and min^j are the maximum and minimum values of the jth feature, respectively. In our experiments, we use seven non-numeric (symbol-valued) features. In the case of symbol-valued features the part of the formula between braces equals 1 if i_1^j and i_2^j have different symbolic values, and 0 otherwise.

2 The Task

The task adopted for our experiments is English hyphenation, i.e., the real-world problem of finding the positions in an English spelling word at which a hyphen ('-') can be placed. For English hyphenation, a fairly large number of cases exist that obey to a few simple, pronunciation-based principles (Treiman and Zukowski, 1990). Moreover, the *morphological principle* introduces a large amount of periphery, since it states that morphological boundaries must receive hyphens, regardless of any other applicable principle. This leads to hyphenations such as <ac-cord-ance>, rather than <ac-cor-dance>, because there is a morphological boundary between the stem <accord> and the suffix <-ance>. From CELEX, a collection of lexical data bases of English, German and Dutch, we derived a data base consisting of 89,019 hyphenated English words. From this data base we extracted three subsets of increasing magnitude: a subset of 200 words (henceforth D1), a subset of 2,000 words (henceforth D2), and a subset of 20,000 words (henceforth D3). We refer to the full data set of 89,019 words as D4.

Instead of using words as instances, we used the *windowing* technique (cf. Sejnowski and Rosenberg, 1987) to convert words into fixed-length instances. An instance consists of a focus letter surrounded by 3 left context letters and 3 right context letters, including blanks before and after words. Dataset D1 thus contains 1,729 instances, D2 16,980, D3 168,549, and D4 750,874. An instance is associated with the classification '1' if a hyphen may be placed before the focus letter, and '0' otherwise.

3 Three Algorithms

The hyphenation task is presented to three learning algorithms: (i) back-propagation (BP), (ii) cascade back-propagation (CBP), and (iii) information-gain tree (IG tree). First, BP is included because of its status as a benchmark algorithm. It is known for its ability to learn regular instances as well as exceptions. However, the latter ability is often limited due to storage limitations. Therefore we have selected IG tree as a symbolic inductive-learning algorithm which has no limitations in storage. For reasons of comparison, we have opted for CBP, an extension of BP, that has no storage limitations. The application of each learning algorithm to each data set was performed using a 10-fold cross-validation setup, except for the application of CBP to D4 which was performed only once due to limitations in computational resources.

3.1 Back-propagation

Back-propagation uses a network architecture with fully connected multiple layers of units. Learning takes place by adjusting the weights of the connections between all units according to the error signal at the output layer (Rumelhart, Hinton, and Williams, 1986). The number of connections follows from the numbers of layers and units predefined beforehand.

In our experiments with BP, we adopted the coding technique used by Sejnowski and Rosenberg (1987), i.e., each of the 7 input letters are locally coded. Moreover, a fixed hidden layer of 50 units was used. Test experiments with different numbers of hidden units showed a worse performance with less units, and no significant performance increase with more units. The output layer contained one unit, representing the class (0 or 1). The learning rate was set at 0.1, with a momentum of 0.4. Next, the *update tolerance threshold* was set to 0.2, i.e., only when the error of the output unit exceeded this threshold, back-propagation took place. This threshold enabled the algorithm to spend less effort on instances already learned. All BP experiments were run for 30 cycles, since there the error on the training material started to converge.

3.2 Cascade back-propagation

Cascade back-propagation (CBP; Van den Bosch, Weijters, and Van den Herik, 1995) is a variant of BP in that it uses the same connection-strength adjustment rules. Its architecture coincides with the architecture of Fahlman and Lebière's (1990) cascade-correlation network.

The first difference between CBP and BP is that CBP starts off without any hidden layers. The input layer and the output layer are fully connected. CBP trains the network until the mean-squared-output error converges. CBP then freezes the connections in the network, and adds one hidden unit which is fully connected to the input and output layer. The new unit is trained with the generalised delta learning rule until the mean-squared-output error again converges. CBP's next step is to freeze the new unit's incoming and outgoing connections. Thereafter, it adds a second hidden unit. This unit is fully connected to the input layer, the output layer *and* all previously-added hidden units. Then the network is again trained using the generalised delta learning rule. This is repeated until a newly added unit does not lead to a decrease of the mean-squared-output error (after a predefined number of possible attempts).

The second difference with BP is that in our CBP implementation, we set all target class values (0 or 1) already classified correctly to 0. The new hidden unit is then trained on the remaining outputs of class 1 previously classified as 0. Thus the task for the new hidden unit ideally becomes simpler as more hidden units 'solve' more training examples.

The third difference is the slight increase of the learning rate during training (an empirically established increase of 0.02 per added unit). The parameter settings in our CBP experiments were kept identical to those used with BP, except for the stop criterion. CBP experiments were run as long as 20 attempted new units did not result in a lower mean-squared-output error.

3.3 Information-Gain Tree

The information-gain tree (IG-tree) algorithm is a data-oriented, symbolic, inductive decision-tree algorithm. It can be considered as a optimisation of IBL (Instance-Based Learning; Aha, Kibler, and Albert, 1991). For a detailed description of the IG-tree algorithm, we refer to Daelemans, Van den Bosch, and Weijters (1995). The idea is that the IG-tree algorithm compresses a set of classification-task instances into a decision-tree structure. Instead of storing full instances as paths in the tree, the algorithm decides to reduce the instances to precisely those input features that properly disambiguate the instance from other instances within the training set. The algorithm is then able to classify new instances by matching them to stored parts of the reduced instances.

In our experiments, the standard IG-tree algorithm was used (Daelemans *et al.*, 1995). Since IG tree is a symbolic learning algorithm, the 7-letter input patterns were not encoded by binary values, but by the letters themselves.

4 Results

An experiment is a 10-fold cross-validation application of one of the three learning algorithms to one of the four data sets. We computed for each experiment (i) the classification performance on test instances, and (ii) the average typicality of misclassified test instances. The results are graphically displayed: the graphs represent the algorithms, the data points our four data sets.

The left diagram of Figure 1 displays the percentages of misclassified test instances. We report the following three observations. First, all three algorithms show a decrease in their generalisation error when trained on a larger data set. Second, BP outperforms the other two algorithms when trained on all but the largest data set (D4). Third, when trained on the massive number of about 676,000 instances, IG tree was found to perform significantly better than both BP and CBP (using t-tests).

The right diagram of Figure 1 displays the average typicality of misclassified test instances. The higher the value, the less exceptions are misclassified, relative to regular instances. For BP and CBP, the *portion* of misclassified exceptions remained roughly constant over all data sets. For IG tree, how-

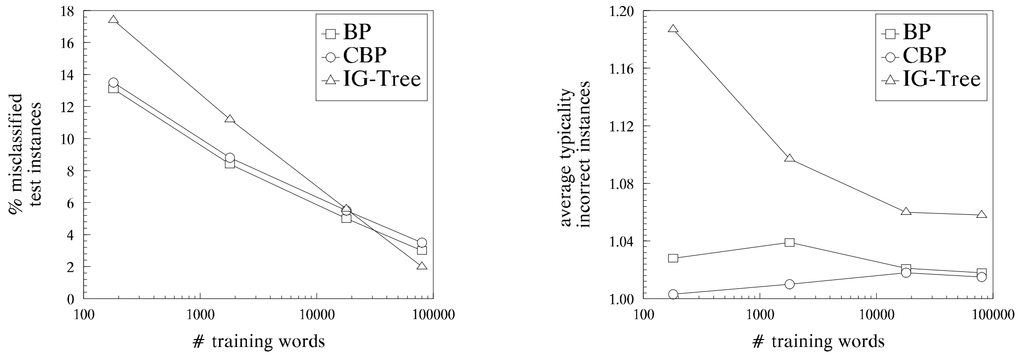

Figure 1: *Percentages of misclassified test instances (left), and average typicality of misclassified test instances (right) for* BP, CBP, *and* IG *tree, trained on the four increasing data sets.*

ever, the *portion* of misclassified exceptions increased with larger data sets (we explicitly remark that the *number* of misclassified exceptions, of course, decreases with larger data sets). Moreover, we see that IG tree makes relatively less classification errors on exceptions as compared to BP and CBP, regardless of any training-set size. The differences in average typicality of misclassified instances between both IG tree and BP, and IG tree and CBP, were found to be significant for all training-set sizes (using t-tests).

When considering the storage capacities, we note that CBP and IG tree created larger models when trained on larger amounts of data. When trained on D1, D2, D3, and D4,CBP developed the following networks (characterised by their number of units with the number of connections between parentheses): 10 (3,005), 42 (13,293), 51 (16,371), and 70 (23,135), respectively; IG tree created trees with on the average 966, 6,473, 34,991, and 64,182 nodes, respectively.

We now focus on the difference between the two connectionist learning algorithms, BP and CBP. When monitoring the errors made by the two algorithms during training, some qualitative differences in the learning of exceptions become visible. Figure 2 displays the distributions of typicalities of misclassified training instances, for BP (left) and CBP (right). The data displayed in Figure 2 were obtained from training both algorithms on the first 10-fold cross-validation partitioning of D2.

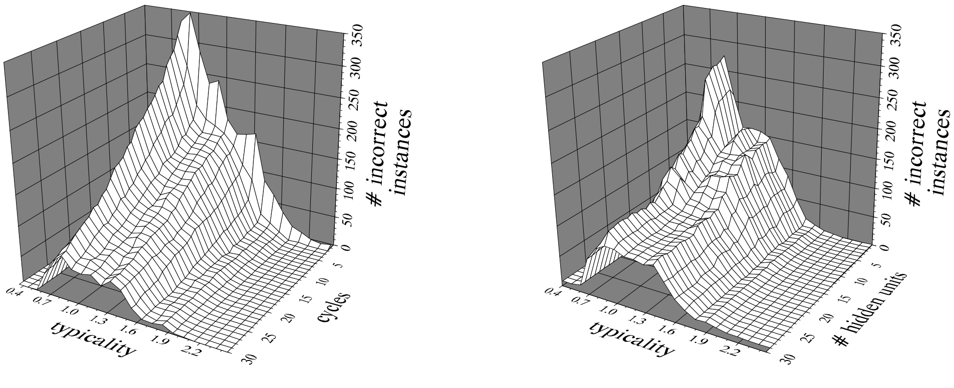

Figure 2: *Typicality distributions, computed over intervals of 0.1, of misclassified instances during training of the* BP *(left) and* CBP *(right) networks applied to the first partitioning of* D2*. The* BP *network was monitored during 30 cycles; the* CBP *network during the appending of 29 hidden units.*

As can be seen from Figure 2, BP is a steady learner which gradually decreases the number of misclassified training instances regardless of their typicality. CBP, however, less gradually learns a considerable number of slightly exceptional instances (seen in Figure 2, right, as the peak at typicality 0.9), but is less successful in learning slightly typical instances (the less pronounced peak at typicality 1.3).

5 Conclusions

We trained BP, CBP, and IG tree on data sets with many exceptions. We found that the three algorithms increased their generalisation performance when training-set size was increased. BP performed better than the other algorithms when trained on the smaller data sets. IG tree outperformed BP and CBP only when trained on the largest data set, D4. Moreover, IG tree made significantly less misclassifications on exceptions than BP and CBP, for all training-set sizes. Hence, applying IG tree leads to a better performance than applying any of the connectionist algorithms, under the assumption that the amount of training instances is large enough.

From these results we conclude that for the hyphenation task, the size of the training set determines the most profitable learning algorithm. For any given real-world task with a large periphery, there might be a training-set size below which the method of BP and CBP is more profitable, and above which the method of IG tree is more profitable.

Acknowledgements

We gratefully recognise Eric Postma for the valuable discussions and technical assistance, and thank Fred Wan for his help with the significance tests.

References

Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 7, pp. 37–66.

Daelemans, W., and Van den Bosch, A. (1992). A neural network for hyphenation. In I. Aleksander and J. Taylor (Eds.), *Artificial Neural Networks 2*, volume 2, pp 1647–1650. Amsterdam: North-Holland.

Daelemans, W., Van den Bosch, A., and Weijters, T. (1995). IG-tree: A variant of IBL. submitted.

Fahlman, S. E. and Lebiere, C. (1990). *The Cascade-correlation Learning Architecture.* Technical Report CMU-CS-90-100, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing*, volume 1: Foundations, pp. 318–362. Cambridge, MA: The MIT Press.

Sejnowski, T.J., and Rosenberg, C.S. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1, pp. 145–168.

Treiman, R., and Zukowski, A. (1990). Toward an understanding of English syllabification. *Journal of Memory and Language*, 29, pp. 66–85.

Van den Bosch, A., Weijters, A., and Van den Herik, H.J. (1995). Scaling effects with greedy and lazy machine-learning algorithms. In *Proceedings of the Dutch AI Conference, NAIC-95*, forthcoming.

Zhang, J. (1992). Selecting typical instances in instance-based learning. In *Proceedings of the International Machine Learning Conference 1992*, pp. 470–479.