Comparing Performance of Neural Networks Applied to a Simplified Recognition Problem

Marcin Paprzycki, Rick Niess, Jason Thomas, Lenny Scardino, William Douglas

Department of Computer Science and Statistics

University of Southern Mississippi Hattiesburg, MS 39406-5106

{m.paprzycki}@usm.edu

Abstract

In this note we present and discuss results of experiments comparing the performance of six neural network architectures (back propagation, recurrent network with dampened feedback, network with multiple hidden layers each with a different activation function, jump connection networks, probabilistic neural networks and general regression neural networks) applied to a simplified multi-font recognition problem.

Introduction

Over the last ten years neural networks (NN) have been successfully applied to a large number of pattern recognition problems (see for instance (Looney 1997) and the references collected there). Existing NN architectures differ in the number and size of layers and the weight adjustment functions applied during the learning process. The learning process itself can proceed with or without supervision and there exists a number of training procedures (backpropagation, conjugate gradient. self-organizing maps etc.). Our review of literature indicates that, in most cases, when a network is applied to a given problem its performance is studied without a comparison to other network architectures and/or training procedures that could have been applied to it as well. In addition, before the data is shown to the network it is often pre-processed by a feature extracting software to reduce the size/domain of the problem. This makes it rather difficult to compare the performance results and establish the best network architecture for the problem/class of problems (as the performance of the network is mediated by the performance of the feature extractor).

In recent years NN software has moved from the academic research laboratories to appear in commercial packages thus allowing someone interested in applying the NN approach to their problem to do so without writing their own NN code. This is particularly attractive to those who are not NN specialists. However, taking into account the above, it can be rather difficult for a non-specialist to find out which NN architecture should be used for her/his problem.

The aim of this note is to report on an attempt to compare the performance characteristics of six NN architectures available in the NeuroShell 2.0 package (Ward 1998) applied to a simplified pattern recognition problem. This software has been used in our earlier work (Bowers and Paprzycki 1999, Bowers et. al., 1999a, Bowers et. al. 1999b, Bowers et. al. 1999c). The results reported there concerned four NN architectures: Probabilistic NN, General Regression NN, Ward Nets and Kohonen Networks. These results were obtained for a relatively small size input data. They indicated that performance of the first three architectures was "acceptable." However, the performance of KSM was definitely sub-par (this result is in agreement with (Fausett 1994, p. 169ff)). In this note we extend our study to include more NN architectures and a significantly larger input data set. Due to the results obtained earlier we have removed KSM from our considerations.

As previously, we used a simplified model-problem of recognition of computer-printed characters (26 upper case Latin letters). This mimics a situation in which computer generated printed characters are scanned, digitized and presented to a computer to be recognized. For simplicity, it was assumed that one letter is presented at a time and that each image presented does indeed represent a letter.

The remaining part of the paper is organized as follows. In the next section we very briefly (due to the lack of space) introduce the NN architectures used in our experiments (details should be found in the literature cited). We follow it with a description of how the data was generated. Finally, we describe the experiments performed and discuss their results.

Neural Network Architectures

Backpropagation (BNN)

Basic feed-forward backpropagation network is one of the best-studied NN architectures. In this note we report results obtained for the standard three-layer architecture: input, hidden, and output. Each node in each layer is connected to all nodes in the next layer and those are the only connections available.

Recurrent Networks (RNN)

Recurrent networks with dampened input layer feedback (Jordan-Elman networks, Elman (1990)) are a modification of the BNN. Here, additional slab of neurons is added to represent a "long-term memory." Results reported in this note are for the network where an additional slab of input

neurons was included. This slab combines inputs from the external data and the standard input slab as well as the inputs originating from it (via loop-back connections). Its outputs are connected to the hidden layer. While a regular feed forward network responds to a given input pattern with exactly the same output every time the given input pattern is presented a recurrent network may respond to the same input patterns which have been presented to it earlier (memory-effect). Recurrent networks are trained the same way as standard backpropagation networks except that training patterns must always be presented in the same order.

Jump Connection Networks (JCN)

The jump connection NN architecture is also based on the backpropagation training procedure. This architecture contains connections that skip layers. In the simplest form, which we have used for our experiments, the three-layer JCN's input layer was connected to both the hidden and to the output layers and the hidden layer was connected to the output layer.

Ward Nets (WNN)

The Ward Nets architecture also uses the standard backpropagation learning algorithm. In the hidden layer they contain multiple slabs of neurons, each utilizing a different activation function. The idea behind this approach is that each slab in the hidden layer will "catch" different characteristics of the input data thus improving pattern recognition capabilities of the network. It is a proprietary design of the Ward Systems; the company that developed the NeuroShell software (Ward 1998). The simple version used in our work contains two "parallel" slabs in the hidden layer that receive input from the input layer and are connected to the output layer. One slab uses Gaussian and the other Gaussian complement activation functions.

Probabilistic Neural Networks (PNN)

Probabilistic Neural Networks are based on an application of a classification theory based on the minimization of the "expected risk" function (Specht 1989, Specht 1990a, Specht 1990b). Here the network's determination is based on a series of measurements and can be expressed in terms of Bayesian decision rules. The key to the classification process is the ability to estimate the probability density functions (PDFs). For NN applications this means that the PDFs have to be estimated on the basis of the training patterns (which are the only source of available information about the data). In 1962, Parzen introduced a class of estimators that asymptotically approach the real density as long as it is smooth and continuous (Parzen 1962) and Specht used it in the PNN design. PNN requires that all information from the training set must be stored and used during testing. In other words, for each input data there is a node in the hidden layer. Training is relatively fast as each input is shown to the network only once, but the time necessary to process the data when the network is already trained is directly proportional to the size of the training set.

General Regression Networks (GRNN)

The general regression networks are very similar to the PNN's as they are also based on estimation of probability density functions. Originally the concept was developed in the statistics literature and known as the Nadaraya-Watson kernel regression and translated to the NN environment by D. Specht (Specht 1991). As in the case of PNN's they use as many neurons in the hidden layer as there are input elements and train fast as each input element is shown to the network only once.

Data generation

In the recent paper (Bowers et. al. 1999a) we presented results based on the digitization of 124 computer fonts represented as 20x20 "bitmaps." However, we were not completely satisfied with the font digitization process. The digitizer software used a black-box procedure to select the font scaling, often resulting in images of the same letter originating from different fonts to be of substantially different size. We have thus developed a new digitizer based on FreeType 1.2, a free TrueType font rendering library available at <u>http://www.freetype.org/</u>. The resulting "bitmaps" represented centered, similarly scaled letters allowing a one-pixel border around the map (so the effective letter image was at most 18x18).

We began with 2450 fonts. These fonts were manually compared to remove identical and similar fonts that exist under different names. Only the basic forms of each font were allowed (no italic or script versions have been used). After two screenings, we ended up with 727 unique fonts. The fonts were then processed by our digitizer to obtain data in the format required by the NeuroShell environment. For all networks we used 400 input nodes (20x20=400, each node corresponding to one element of the input vector) and 26 output nodes (corresponding to the 26 letters of the alphabet). The number of nodes in the hidden layer(s) varied between the NN architectures.

The resulting data was divided into 6 groups of 100, 200, ..., 600 fonts which were used for training (and denoted as data sets 1, 12, 123, 1234, 12345 and 123456). We have used the remaining 127 fonts for testing. We have divided the data into groups alphabetically (based on the font names). Since there is no relationship between the name of the font and its shape this approach did not have any effect on the results.

To reduces the time required to complete our experiments we have divided the training data into two parts the training part and the self-test extraction (default 20% was used here). Then the same data was used for all backpropagation-based networks. This allowed us to complete the experiments in a reasonable time (each run took between 30 minutes and 30 hours depending on the architecture and the computer used to run them; with the longest time required by the recurrent networks). At the same time the open question remains: how representative are these results? Three observations can be made here. First, the PNN and GRNN results are independent of the data extraction as all elements are used in training. Second, since exactly the same data was used for training and internal testing at least the relationship between the architectures should be representative. Third, we have run additional experiments in which we have varied the division of the training data into training and internal testing. The results obtained there indicate that the variation in results depends on the NN architecture. For instance the recurrent networks seem to be extremely stable with almost no variance, while the jump connection networks have large variation. In no experiment, however, did the results of the JCN reach the performance level of other networks. We plan to present a complete report on these results in the near future.

Experimental Results

In the first series of experiments we used each group of fonts for training. We then applied the test data to the trained networks. In each case the default setup provided by the NeuroShell software for the given network was used. This includes parameters characterizing the learning process (e.g. starting learning rate of 0.3, initial values for the neurons and weights, 20% of the input data extracted for internal testing during the learning process and others) as well as the network itself (e.g. activation functions and number of nodes in the hidden layer). The default number of nodes in the hidden layer for the BNN, RNN, and JCN is computed in the same way and for data sets 1, 12, 123, 1234, 12345, and 123456 is 259, 277, 292, 304, 315, and 325, respectively. The default size for the WNN is computed similarly, however the number of neurons in the hidden layer (resulting from the formula) is divided between the slabs. Thus for the architecture with two slabs of neurons in the hidden layer each of them will have half of the neurons and thus for data sets 1, 12, 123, 1234, 12345, and 123456 there were 130, 138, 146, 152, 158, and 162 neurons in each slab, respectively.

Table 1 contains the results of our experiments. The percent of correct answers is reported for each network for all 6 data sets.

Data set	BNN	RNN	JCN	WNN	PNN	GRNN
1	89.74	98.28	81.58	31.70	83.52	83.45
12	91.55	94.81	83.64	28.04	86.90	86.96
123	92.43	94.84	85.52	34.05	88.02	88.34
1234	92.65	96.00	81.39	29.92	89.40	89.55
12345	93.12	95.84	87.39	31.73	90.06	90.06
123456	92.99	96.19	81.89	27.79	90.49	90.81

Table 1. Performance comparison, default settings, percent of correct answers.

The results indicate that out of the six networks the WNN is the least competitive. This result is a bit surprising, as we have found that a slightly different WNN configuration (with three slabs in the hidden layer) performed relatively well (Bowers et. al. 1999c). Since the network used in our experiment also had problems with recognizing the data used for training, this may indicate that the default settings are to blame. We plan to investigate this further.

Out of the remaining networks, the jump connection network is the weakest while the recurrent network outperforms the other networks. The three remaining networks, the basic backpropagation the probabilistic neural network and the general regression network have relatively similar performance. We find it slightly surprising to see that the most basic backpropagation network behaves so well relative to other, more advanced architectures. Another surprise is the fact that PNN and GRNN behave so similarly. In our earlier study we found that PNN outperformed the GRNN in all cases (Bowers et. al. 1999b). This change has to be attributed to the substantial increase in the size of the input data.

Here it must be pointed out that, for both for the PNN and the GRNN, the quality of recognition depends greatly on the value of the smoothing factor σ as the results are highly sensitive in regard to its value. In addition, the optimal value of σ is different for each data set. We were able to find out that for the PNN the best results were obtained for σ values between 1.3 and 1.5 and for the GRNN for the σ values between 0.6 and 0.8. Data reported above should be treated as approximately optimal (it is possible that slightly better outcomes of the experiments exist). Using fixed value of $\sigma = 1.4$ for the PNN or $\sigma = 0.7$ for the GRNN would make the results worse by 2-4 correct answers. Nevertheless this observation has a peculiar effect on the potential of

applying PNN's or GRNN's to practical problems. Typically, it is not possible to know the correct value of the smoothing factor a'priori and thus calibrate the network to deal correctly with possible unknown input patterns. In this context, the NeuroShell allows to use iterative calibration to find the "best" smoothing factor for the problem. We have experimented with this strategy and the results of are discussed below.

In the second series of experiments we modified the total number of nodes in the internal layer by increasing it

to match the size of the input layer (in case of the WNN we have used 200 nodes in each slab). Of course this change was not possible for the PNN or GRNN so they were not tested in this manner. We have left the remaining parameters as defaults and used exactly the same division of the data into training and internal testing sets. Table 2 summarizes the results. As previously, the percentage of correct answers is reported when the test data that was shown to the network.

Data set	BNN	RNN	JCN	WNN
1	89.08	92.71	81.01	36.18
12	91.83	95.12	83.20	28.99
123	92.12	94.84	78.39	29.24
1234	92.77	93.79	73.54	32.27
12345	93.05	95.65	82.95	27.89
123456	93.43	96.43	81.70	27.27

Table 2. Performance comparison, 400 nodes in the inner layer, percent of correct answers.

The WNN network still is the weakest of the four, returning self-test percentages no greater than 38%. Out of the remaining networks, again, the RNN outperforms the other networks, with vanilla backpropagation coming as a close second.

As we can see from the comparison with Table 1, increase of the number of neurons in the hidden layer to 400 appears to have only a minimal effect on the ability of all the networks to recognize the test inputs. This result is slightly surprising, as the difference between the default size and the 400 nodes used here is rather substantial. One thus could expect that the backpropagation-type network with so many neurons in the hidden layer would "memorize" the input data and not be able to generalize well.

In the final series of experiments we have investigated the quality of the iterative calibration procedure that can be applied to establish the optimal smoothing factor for both the PNN and GRNN networks. To do this, a self-test of 20% is extracted from the training data and used to calibrate the network and find the optimal smoothing factor. (Note that this approach results in a substantial increase of training time.)

In Table 3 we have summarized the results of applying the network to the test data when the smoothing factor found by the calibration procedure is used (denoted as PNN(c) and GRNN(c)). For comparison purposes, the results obtained by manually selecting the optimal smoothing factor (from Table 1) are reported again.

Data set	PNN(c)	PNN	GRNN(c)	GRNN
1	81.61	83.52	75.20	83.45
12	84.58	86.90	78.05	86.96
123	87.68	88.02	83.27	88.34
1234	88.68	89.40	85.52	89.55
12345	89.31	90.06	86.46	90.06
123456	89.81	90.49	87.14	90.81

Table 3. Performance comparison, effectiveness of calibration, percent of correct answers.

It can be observed that the calibration procedure is definitely not very successful for the smaller data sizes. However, for the largest data set its performance is very good (this is especially the case for the PNN, where the difference is less than 1%). Note that 20% of the data has been removed to use in the calibration procedure thus

reducing the amount of data used to build the network (here each input element is used to "train" one neuron in the hidden layer). It can be conjectured that if additional 120 fonts were added to the data set 123456 (making the size of the network built equal to that of the one created without extraction) then the performance of calibration could reach that of hand selecting the optimal smoothing factor.

Concluding Remarks

In this note we have compared the performance of six NN architectures applied to a simplified pattern recognition model-problem. We have found that the recurrent neural networks with dampened feedback outperform the remaining architectures (but with basic backpropagation, probabilistic and general regression neural networks performing reasonably well).

We were also able to establish that for our problem the size of the hidden layer does not play such an important role as could have been expected.

Finally, we have found out that for large sizes of input data the calibration procedure provides a reasonable estimate of the optimal smoothing factor to be used in general regression and probabilistic neural networks.

We believe that these results are relatively interesting regardless of the detailed answers to a number of detailed questions that we have posed throughout the paper. It is a well-known fact that recurrent as well as general regression and probabilistic neural networks are often considered impractical as they consume enormous amount of resources (time for RNN and memory for GRNN and PNN). As the available computing power increases we may need to re-evaluate some of these beliefs. Our results indicate that this step may be justified by the promise of better performing networks.

We plan to continue our experiments to address the questions posed above (especially to address the need for multiple runs and a statistical analysis of the data). In addition we will use more involved architectures of each type (for instance backpropagation is available in three variants: with one, two and three hidden layers and similarly the remaining backpropagation based architectures). We will also assess the effects of various parameters, available in the NeuroShell, on the performance of NN architectures (in this way we will try to find the optimal setup for a given architecture for our model-problem). Finally, when the above-described program is completed we will start comparing the performance of the NeuroShell based networks with these available in the Trajan Neural Network environment.

References

Elman, J.L. 1990. Finding structure in time. *Cognitive Science*. 14:179-211

Looney, C.G. 1997. *Pattern Recognition Using Neural Networks*. New York: Oxford University Press.

Bowers, S., Costeines, A. and Paprzycki, M. 1999a. Applying Probabilistic Neural Networks to the Multifont Recognition Problem with Large Training Set, in: Kumar, A.N. et. al. (eds.) *Proceedings of the Twelfth International Florida AI Research Society Conference*, 336-339, Menlo Park: AAAI Press.

Bowers, S., Costeines, A. and Paprzycki, M. 1999b. Evaluating the Performance of Three Neural Network Architectures Applied to the Pattern Recognition Problem, *Proceedings of the 15th Annual Conference on Applied Mathematics*, 29-36, Edmond: University of Central Oklahoma.

Bowers, S., Morrison, J. and Paprzycki, M. 1999c. Comparing Performance of Neural Networks Recognizing Machine Generated Characters, *Proceedings of the First Southern Symposium on Computing*, to appear, Hattiesburg: University of Southern Mississippi.

Bowers, S. and Paprzycki, M. 1999. Applying PNN's to the Multifont Pattern Recognition Problem, in: Bainov, D., (ed.) *Proceedings of the 9th Colloquium on Difference Equations*, 311-318, Utrecht: VSP.

Nadaraya, E.A. 1964 On estimating regression. *Theory Probab. Applic.* 10: 186-90.

Parzen, E. 1962 On Estimation of a Probability Density Function and Mode. *Ann. Math. Stat.* 33:1065-1076.

Specht, D. 1989. Probabilistic Neural Networks for Classification, Mapping, or Associative Memory. In Proceedings IEEE Conference on Neural Networks, vol. 1, 525-532. Los Alamitos: IEEE Press.

Specht, D. 1990a. Probabilistic Neural Networks. *Neural Networks* 3:109-118.

Specht, D. 1990b. Probabilistic Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification. *IEEE Transactions on Neural Networks* 1(1):111-121.

Specht, D. 1991. A General Regression Neural Network, *IEEE Transactions on Neural Networks*. 2:568-576.

Ward. 1998. http://www.wardsystems.com

Watson, G.S. 1964. Smooth regression analysis. *Sankhya*. Series A, 26:359-72.