IPNN: An Incremental Probabilistic Neural Network for Function Approximation and Regression Tasks

Milton Roberto Heinen and Paulo Martins Engel
Informatics Institute, Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brazil, CEP 91501–970
Emails: mrheinen@inf.ufrgs.br, engel@inf.ufrgs.br

***Abstract*—This paper presents a new probabilistic neural network model, called IPNN (for Incremental Probabilistic Neural Network), which is able to learn continuously probability distributions from data flows. The proposed model is inspired in the Specht's general regression neural network, but have several improvements which makes it more suitable to be used in on-line and robotic tasks. Moreover, IPNN is able to automatically define the network structure in an incremental and on-line way, with new units added whenever necessary to represent new training data. The experiments performed using the proposed model shows that IPNN is able to approximate continuous functions using few probabilistic units.**

***Keywords*-Probabilistic neural networks; General regression neural networks; Incremental learning; Gaussian mixture models; Semi-parametric methods; Bayesian methods.**

I. Introduction

Probabilistic neural network (PNN) [1], [2] is a feed-forward artificial neural network (ANN) based on the Bayes strategy for decision making and nonparametric estimators of conditional probability density functions (pdf). Its most important advantage over other ANN models is that the training is easy and instantaneous. In fact, in a PNN learning occurs after a single presentation of each pattern (the procedure is not iterative), and new information can be used whenever it becomes available. Another advantage of PNN is that it is guaranteed to asymptotically approach the Bayes' optimal decision surface provided that the class probability density functions are smooth and continuous [3].

The main limitation of the original PNN architecture proposed by Specht [1], [2] is that it requires a separate neuron for each training pattern, which makes the computation very slow for large databases requiring a large amount of space in memory. To avoid this limitation, in [4] a clustering algorithm based on stochastic gradient descend is proposed to find a reduced set of representative exemplars to be used as the nodes for PNN. Unfortunately, the solution proposed in [4] requires labeled training samples, which can only be applied for classifying data. For the general regression neural network (GRNN) [5], which is the probabilistic network model used for estimation of continuous variables, the Traven's algorithm is not directly applicable. Moreover, it uses radially symmetric pdf's, i.e., covariance matrices of the form $\mathbf{C} = \sigma^2\mathbf{I}$ (where σ is smoothing parameter and $\mathbf{I}$ is the identity matrix), which according to [6] is not robust with respect to affine transformations of the feature space and requires a careful adjustment of the smoothing parameter.

In [7] the Tråvén's work is extended so that no constraints are imposed on the covariance structure of the mixture components (pattern units). However, in this algorithm the number of mixture components is assumed to be fixed, which prevents its use in continuous learning tasks. Although other algorithms have been proposed to automatically set the mixture components of both PNN and GRNN [3], [8]–[11], most of then are based on off-line methods (e.g. the EM algorithm [12]), which requires that the complete training set is previously known and fixed [13]. For on-line applications, in which the learning process must occur continuously (e.g. mobile robots [14]), these algorithms are not very useful.

In [15] the incremental probabilistic neural network suggested in [16] is used for classification in a versatile electronic nose. Although at a first glance this incremental PNN may be interesting, it has the same drawbacks of the Specht's PNN, i.e., it uses one radially symmetric pdf for each training pattern, which does not scale well when the training data are abundant and the learning process must occur perpetually. Moreover, this classifier accesses the previous trained data patterns during the retraining phase, which must be avoided in an incremental algorithm [17].

In this paper a new probabilistic neural network model, called IPNN (for Incremental Probabilistic Neural Network) [18], is proposed. It is based on the Specht's GRNN model [5], but have several improvements which makes it more suitable to be used in on-line applications operating in stochastic environments. The main advantages of IPNN over other probabilistic models are:

- It does not require a neuron for each training pattern;
- The number of pattern units is not limited nor fixed (new units are incrementally added when necessary);
- Full covariance matrices are used in the pattern units;
- The mixture components (pattern units) are continuously adjusted to fit the distributions of the input data;
- IPNN does not require that the complete training set be previously known and/or fixed (each training pattern can be immediately used and discarded);
- It is not necessary any normalization in the input data.

978-0-7695-4210-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SBRN.2010.13

Although in this paper we will focus just in regression tasks, all improvements described above can be used either for classification, regression or clustering tasks as well. The rest of this paper is organized as follows. Section II describes the proposed model in details. Section III presents some experiments performed with IPNN. Finally, Section IV provides some final remarks and perspectives.

II. Incremental probabilistic neural network

This section describes the new probabilistic neural network proposed in this paper, called IPNN [18]. It is based on Specht's PNN [2] and GRNN [5] models, but has several improvements which makes it more adequate to be used in on-line and continuous learning tasks. Figure 1 shows a diagram of the IPNN model. Its structure is similar to the RBF network [19], but the learning algorithm (described in Subsection II-A) is quite different.

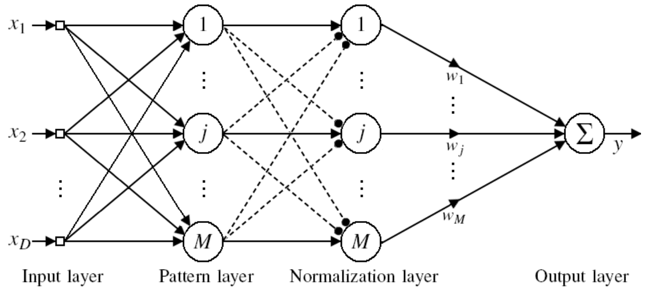

Figure 1. Diagram of the IPNN model

As other neural network models, the first layer of IPNN has just distribution units, but unlike GRNN and PNN in the proposed model it is not necessary to normalize/rescale the input variables (in Specht's models this is necessary because the kernels have the same width in each direction). The second layer, composed by probabilistic neurons (called pattern units in the Specht's model), is implemented using multivariate Gaussian distributions, i.e., the component densities $p(\mathbf{x}|j)$ are computed through:

$$p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\} \quad (1)$$

where $\mathbf{x}$ is the input data, D is the dimensionality of $\mathbf{x}$ (number of input variables), $\boldsymbol{\mu}_j$ is the mean and $\mathbf{C}_j$ is the covariance matrix of the jth pattern unit. The third layer, which has always the same size of the pattern layer, computes the posterior probability $p(j|\mathbf{x})$ for each unit j using the Bayes rule, i.e.:

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)\,p(j)}{\sum_{q=1}^{M} p(\mathbf{x}|q)\,p(q)} \quad (2)$$

where is $p(j)$ is the prior probability of j. The dotted lines in Figure 1 represent inhibitory connections used for normalization in (2). The fourth layer computes the actual network output y through:

$$y = \textstyle\sum_{j=1}^{M} w_j\, p(j|\mathbf{x}) \quad (3)$$

where M is the current number of pattern units. Although Figure 1 shows just one unit in the output layer, IPNN can have an arbitrary number of outputs (i.e., y can be a vector).

As described above, in IPNN the number of pattern units is not predefined or fixed. Moreover, it uses non-restricted, full multivariate covariance matrices in the pattern units. These improvements are possible because IPNN uses a new learning algorithm, described in the next subsection, which is able to incrementally learn Gaussian mixture models in an on-line and continuous way.

A. Learning algorithm

Algorithm 1 presents a detailed pseudocode description of the learning algorithm used by IPNN, which is based in our previous work in statistical learning [20].

Like other supervised ANN models, IPNN has two operation modes, called *learning* and *regression*. But unlike most part of these models, in IPNN these operations don't need to occur separately, i.e., the learning and regression tasks can be intercalated. In fact, even with just one training pattern the neural network can be used in the regression mode (IPNN can immediately use the acquired knowledge), and the estimates become more precise as more training data arrives. Moreover, the learning process can occur perpetually, i.e., the neural network can always be updated as new input data arrives without retraining.

Initially the IPNN network has no units in the pattern layer. When the first training data arrives, the first pattern unit is created and it is centered in this input data, i.e., the distribution parameters are initialized through:

$$\boldsymbol{\mu}_1 = \mathbf{x}^1;\ \mathbf{C}_1 = \sigma_{ini}^2\mathbf{I};\ w_1 = d^1;\ p(1) = 1;\ sp_1 = 1,$$

where σ_{ini} is a configuration parameter which defines the initial radius of $\mathbf{C}$ (the pdf is initially circular but it changes to reflect the actual data distribution as new input data arrives), d^1 is the desired response received at time $t = 1$, $p(1)$ is the prior probability of the first unit (it is required that $\sum_{j=1}^{M} sp_j = 1$) and sp is an accumulator which stores the summation of the posterior probabilities $p(j|\mathbf{x})$, i.e.:

$$sp_j = \textstyle\sum_{t=1}^{T} p(j|\mathbf{x}^t) \quad (4)$$

where T is the current time and $\mathbf{x}^t$ is the input data received at time t. It is important to say that a normalization unit is created whenever a new unit is added to the pattern layer, i.e., both layers have always the same size.

When a new training data arrives, the pattern layer is activated and the component densities $p(\mathbf{x}|j)$ of each unit j are computed using Equation 1. The algorithm then decides if it is necessary to create a new component for the current

Algorithm 1 *Summary of the IPNN learning algorithm*

```
for all training data (x, d) do
    {Compute the probability of $\mathbf{x} \in j$th unit}
    for all pattern unit $j$ do
        $p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\}$
    end for
    {Create a new unit if necessary}
    if $M < 1 \cup p(\mathbf{x}|j) < \frac{\tau_{nov}}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}}, \forall j$ then
        $M = M + 1$
        $\boldsymbol{\mu}_M = \mathbf{x}$
        $\mathbf{C}_{M,ik} = \sigma_{ini}^2 \mathbf{I}$
        $w_M = d$
        $sp_M = 1$
        $p(M) = \begin{cases} 1 & \text{if } M = 0 \\ (\sum_{j=1}^{M} sp_j)^{-1} & \text{otherwise} \end{cases}$
        $p(\mathbf{x}|M) = \left\{(2\pi)^{D/2}\sqrt{|\mathbf{C}_M|}\right\}^{-1}$
        $p(j) = \frac{p(j)}{\sum_{q=1}^{M} p(q)} \quad \forall j$
    end if
    {Compute the posterior probabilities}
    for all pattern unit $j$ do
        $p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)\,p(j)}{\sum_{q=1}^{M} p(\mathbf{x}|q)\,p(q)}$
    end for
    {Incremental estimation step}
    for all pattern unit $j$ do
        $sp_j = sp_j + p(j|\mathbf{x})$
        $\boldsymbol{\mu}_j = \boldsymbol{\mu}_j + \frac{p(j|\mathbf{x})}{sp_j}(\mathbf{x}-\boldsymbol{\mu}_j)$
        $\mathbf{C}_j = \mathbf{C}_j - (\boldsymbol{\mu}_j - \boldsymbol{\mu}_j^{old})(\boldsymbol{\mu}_j - \boldsymbol{\mu}_j^{old})^T +$
                $\frac{p(j|\mathbf{x})}{sp_j}\left[(\mathbf{x}-\boldsymbol{\mu}_j)(\mathbf{x}-\boldsymbol{\mu}_j)^T - \mathbf{C}_j\right]$
        $p(j) = \frac{sp_j}{\sum_{q=1}^{M} sp_q}$
        $w_j = w_j + \frac{p(j|\mathbf{x})}{sp_j}(d - w_j)$
    end for
end for
```

data point $\mathbf{x}^t$ based on the posterior probabilities $p(\mathbf{x}|j)$ of component membership, i.e.:

$$p(\mathbf{x}|j) < \frac{\tau_{nov}}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}}, \quad \forall j \tag{5}$$

where τ_{nov} is a configuration parameter which specifies a minimum value for the acceptable likelihood. If (5) returns true, the new input data $\mathbf{x}$ is not consider a member of any existing component. In this case, a new unit k is created centered in $\mathbf{x}^t$, i.e.:

$$\boldsymbol{\mu}_k = \mathbf{x}^t;\ \mathbf{C}_k = \sigma_{ini}^2\mathbf{I};\ w_j = d^t;\ p(k) = \left(\sum_{j=1}^{M} sp_j\right)^{-1};\ sp_k = 1.$$

After this, the prior probabilities of all j units are adjusted to sum one through

$$p(j) = \frac{p(j)}{\sum_{q=1}^{M} p(q)} \tag{6}$$

Otherwise (if Equation 5 returns false for at least one unit), the existing mixture model is updated using the following recursive equations derived from the classical Robbins-Monro stochastic approximation method [21]:

$$sp_j = sp_j + p(j|\mathbf{x}) \tag{7}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\mu}_j + \frac{p(j|\mathbf{x})}{sp_j}(\mathbf{x}-\boldsymbol{\mu}_j) \tag{8}$$

$$\begin{aligned}\mathbf{C}_j = \mathbf{C}_j - (\boldsymbol{\mu}_j - \boldsymbol{\mu}_j^{old})(\boldsymbol{\mu}_j - \boldsymbol{\mu}_j^{old})^T +\\ \frac{p(j|\mathbf{x})}{sp_j}\left[(\mathbf{x}-\boldsymbol{\mu}_j)(\mathbf{x}-\boldsymbol{\mu}_j)^T - \mathbf{C}_j\right]\end{aligned} \tag{9}$$

$$p(j) = \frac{sp_j}{\sum_{q=1}^{M} sp_q} \tag{10}$$

where $p(j|\mathbf{x})$ is the posterior probability computed by (2) and $\boldsymbol{\mu}_j^{old}$ refers to the value of $\boldsymbol{\mu}_j$ at time $t-1$ (i.e., before updating). Finally, the output weight w_j is updated using the desired response d^t, i.e.:

$$w_j = w_j + \frac{p(j|\mathbf{x})}{sp_j}(d^t - w_j) \tag{11}$$

The learning algorithm used by IPNN has just two configuration parameters, the initial radius σ_{ini} and the novelty parameter τ_{nov}. The initial radius is similar to the Traven's smoothing parameter [1], but it in IPNN σ_{ini} only defines the initial radius of the pdf's (their size and format rapidly changes as new training data arrives) and thus is not very critical. The only requirement for σ_{ini} is be large enough to avoid singularities. In our experiments, described in the next section, we have simply used $\sigma_{ini} = (\mathbf{x}_{max} - \mathbf{x}_{min})/10$.

The τ_{nov} parameter, on the other hand, is more critical and must be defined carefully. It indicates how distant $\mathbf{x}$ must be from $\boldsymbol{\mu}_j$ to be consider a non-member of j. For instance, $\tau_{nov} = 0.01$ indicates that $p(\mathbf{x}|j)$ must be lower than one percent of the Gaussian height (probability in the center of the Gaussian) for $\mathbf{x}$ be considered a non-member of j. If $\tau_{nov} < 0.01$, few pattern units will be created and the regression will be coarse. If $\tau_{nov} > 0.01$, more pattern units will be created and consequently the regression will be more precise. In the limit, if $\tau_{nov} = 1$ one unit per training pattern will be created, which corresponds to the Traven's model [5] but using multivariate Gaussian kernels.

III. Experiments

This section describes the experiments performed with the proposed model to evaluate its performance in on-line regression tasks. The configuration parameters used in these experiments are $\tau_{nov} = 0.01$ and $\sigma_{ini} = (\mathbf{x}_{max} - \mathbf{x}_{min})/10$. It is important to say that no exhaustive search was performed to optimize these parameters. The first experiment, introduced in [5], is a simple problem with one independent variable in which the regression technique needs to model a "plant" which is an amplifier that saturates in both polarities and has an unknown offset. The training data used in this experiment is sparse – just five samples at x = -2, -1, 0, 1 and 2. The red line in Figure 2 shows the input/output characteristic of this plant, and the blue line shows the regression obtained in this experiment for the interval $[-4, 4]$. Five units were added to the pattern layer during the learning process.

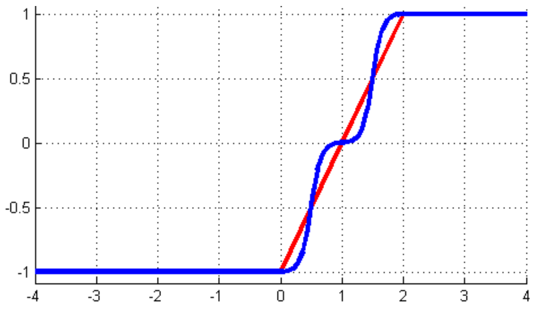

Figure 2. simple "plant" estimated by IPNN

Comparing this result with those presented in [5], it can be noticed that IPNN was able to reproduce the best GRNN result (which was obtained in [5] using $\sigma = 0.3$ computed through the holdout method) even using default configuration parameters, i.e., the proposed model was able to automatically define the best configuration (mixture model parameters) for this problem. It is important to say that IPNN (and GRNN, too) does not have any random initialization and/or decision, and therefore the obtained results are always the same for the same dataset.

The second experiment, also performed in [5] to highlight the GRNN advantages over other neural network models, is a more complex plant described by the equation:

$$u(k) = \begin{cases} \sin(2\pi k/250) & \text{if } \ k \leq 500 \\ 0.8\sin(2\pi k/250) + 0.2sin(2\pi k/25) & \text{if } \ k > 500 \end{cases}$$

where k is an integer value in the interval $[1, 1000]$ (i.e., 1000 training samples were used). This plant was originally introduced in [22] for the control of nonlinear dynamical systems using back-propagation neural networks. Figure 3 shows the results obtained in this experiment. In this figure, the training samples are shown using red dots, and the output estimated by IPNN is shown using a blue line.

The root mean square (RMS) error obtained in the Figure 3 experiment was 0.065, and 43 probabilistic units were added during the learning process. The time required for learning was 0.11 seconds in a typical computer[1]. Comparing these results with those presented in [5], it can be noticed that IPNN is able to obtain a similar performance (although in [5] the RMS error is not informed) using less pattern units than the GRNN model (the Specht's GRNN model would create one pattern unit for each training data, i.e., 1000 pattern units). Moreover, the IPNN learning process

[1]All experiments were executed in a Dell Optiplex 755 computer, Intel(R) Core(TM)2 Duo CPU 2.33GHz processor and 1.95GB of RAM.

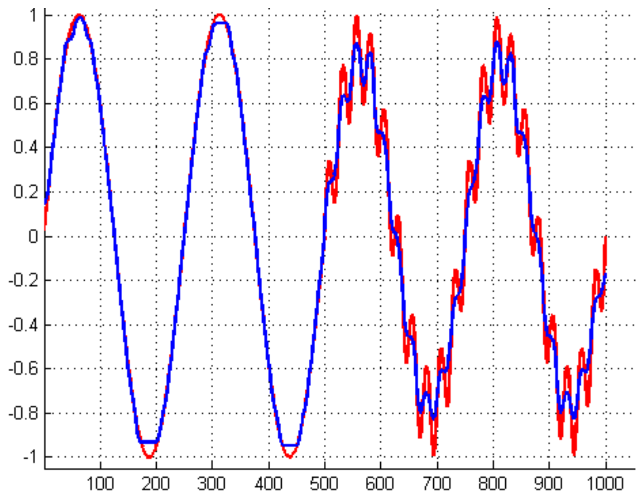

Figure 3. Characteristics of the complex plant estimated by IPNN

was completely on-line and incremental, not requiring the selection of specific patterns for training.

The next experiment shows the capacity of the proposed model to fit more complex distributions, in this case a distribution composed by two inputs and one output generated using the following equation:

$$d = sin(x_1)/x_1 \; sin(x_2)/x_2$$

Figure 4 shows the target surface to be approximated by IPNN. The training set consists of 2500 samples randomly chosen in the intervals $x_1 \in [-10, 10]$ and $x_2 \in [-10, 10]$. Figure 5 shows the surface approximated using IPNN. The number of pattern units added during the learning process was 32, and the time required to perform this experiment was 0.53 seconds in the same computer described above.

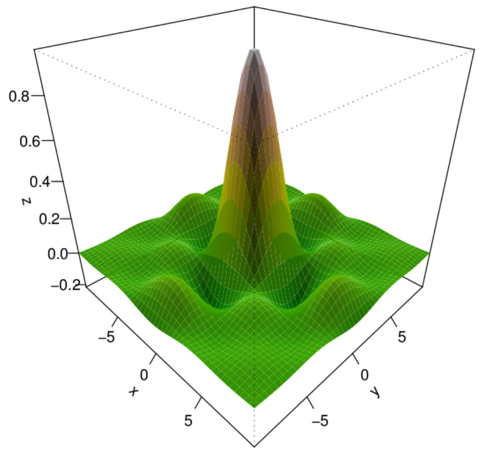

Figure 4. Surface to be approximated by IPNN

It can be noticed that Figure 5 results are very satisfactory, because the learning algorithm was able to find a good solution (the RMS error was 0.0706) using only 1.28% of the number of pattern units required for the standard GRNN. This experiment was repeated presenting the patterns in different orders but the results were practically the same.

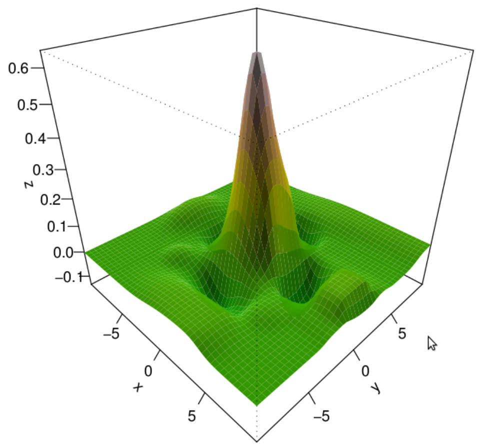

Figure 5. Surface approximated by IPNN (RMS = 0.0706)

As a final example, the proposed model was used in a task where the learning system really needs to operate on-line: to compute the desired speeds for a mobile robot following the walls of the environment shown in Figure 6 (this environment was originally used in [23]), where the squares represent the robot trajectory (each color represents the most active pattern unit at each time) and the black arrow shows the robot starting position and direction. This experiment is relevant because in most mobile robot control tasks it is not possible to predict all situations that may occur in the real world, an thus mobile robot needs to learn from experience while it is interacting with the environment. Moreover, for a mobile robot to adapt to new situations the learning process must occur continuously.

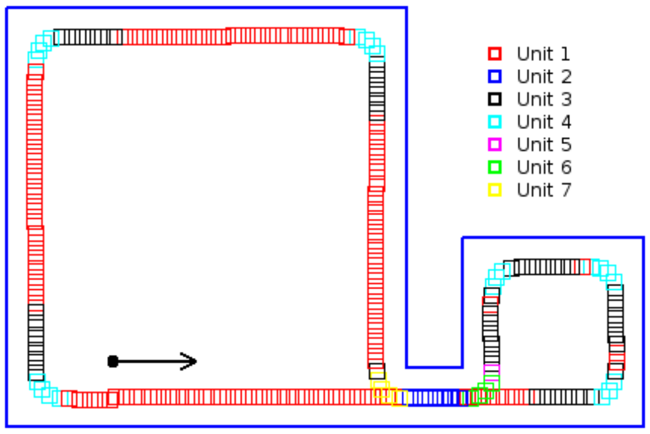

Figure 6. Environment used in the robotic task

In this experiment, the input data consist of a sequence of 4 continuous values (s_1, s_2, s_3, s_4) corresponding to the readings of a sonar array located at the left/right side (s_1, s_4) and at $-10°/+10°$ from the front (s_2, s_3) of a robot, generated using the Pioneer 3-DX simulator software ARCOS (Advanced Robot Control & Operations Software). The output data correspond to the speeds to be applied to the right (v_1) and left (v_2) motors of the mobile robot. In these experiments, the robot was manually controlled to perform one loop in the environment shown in Figure 6, and the task consists in predict the next motor actions at each time t.

Figure 7 shows the results obtained in this experiment, where the x axis corresponds to the simulation time t in seconds and the y axis corresponds to the difference between the right and left speeds, i.e., $y_d(t) = v_1 - v_2$. A positive value in $y_d(t)$ corresponds to a left turn in the robot trajectory and a negative value corresponds to a right turn. The red line in Figure 7 represents the desired $y_d(t)$ values and the blue line represents the difference between the IPNN outputs, i.e.: $y_o(t) = y_1$ - y_2. It is important to say that the IPNN used in this experiment has really two outputs, i.e., the difference $y_o(t)$ was used in Figure 7 just for visualization purposes.

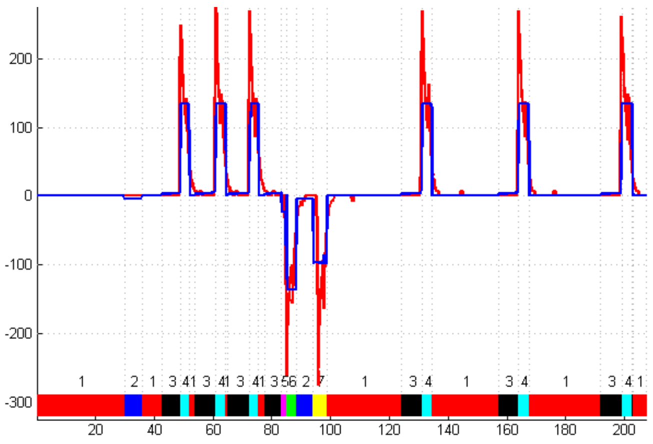

Figure 7. Results obtained in the robotic task (RMS = 25.49)

It can be noticed in Figure 7 that IPNN was able to approximate the desired output with reasonable accuracy (the RMS error was 25.49, which corresponds to 6.37% of the robot's average speed) using few probabilistic units (just seven) and no memory of past perceptions and actions (e.g., recurrent connections). The main differences between $y_d(t)$ and $y_o(t)$ are in the extremes, i.e., the approximation performed by IPNN is smoother than the target function.

Another interesting feature which can be noticed in this experiment is that IPNN is able to create useful representations in the pattern and normalization layers. This can be observed in Figure 6, where the colors in the robot's trajectory represent the units with the highest posterior probability $p(\mathbf{x}|j)$ (i.e., the so called *maximum likelihood hypothesis*). It can be noticed in Figure 6 that the pattern units are related to some the persistent "concepts" like *wall at right* (1: red), *corridor* (2: blue), *curve at left* (4: cyan), *curve at right* (6: green) and so on. The lower band in Figure 7 also shows maximum likelihood hypothesis at each time step, which stands out the network outputs produced by each probabilistic unit.

IV. Conclusion

In this paper a new probabilistic neural network model, called IPNN, is proposed. This proposed model, which can be considered an improved version of the Specht's GRNN [5], is able to perform regression tasks in an incremental and continuous way. Moreover, it uses full multivariate Gaussian probability density functions (pdf) in the probabilistic units, which allows more precise approximations using less artificial neurons in the pattern layer. The experiments performed using the proposed model shows that IPNN is able to approximate continuous functions in a very accurate way. Besides, the learning process occurs in an on-line and incremental way without requiring that the complete training set is previously known and fixed. This is necessary if the neural network has to be used in continuous learning tasks as robotic control in unexplored environments. Future developments will use the proposed model in control and learning tasks using a real Pioneer 3-DX mobile robot.

Acknowledgment

This work is supported by CNPq, an entity of the Brazilian government for scientific and technological development.

References

[1] D. F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, San Diego, CA, 1988, pp. 525–532.

[2] ——, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118, 1990.

[3] L. Rutkowski, "Adaptive probabilistic neural networks for pattern classification in time-varying environment," *IEEE Trans. Neural Networks*, vol. 15, no. 4, pp. 811–827, Jul. 2004.

[4] H. G. H. Tråvén, "A neural network approach to statistical pattern classification by "semiparametric" estimation of probability density functions," *IEEE Trans. Neural Networks*, vol. 2, no. 3, pp. 366–377, May 1991.

[5] D. F. Specht, "A general regression neural network," *IEEE Trans. Neural Networks*, vol. 2, no. 6, pp. 568–576, 1991.

[6] D. Montana, "A weighted probabilistic neural network," in *Advances in Neural Information Processing Systems*, vol. 4. Morgan Kaufmann, 1992, pp. 1110–1117.

[7] J. Ćwik and J. Koronacki, "Probability density estimation using a gaussian clustering algorithm," *Neural Computing and Applications*, vol. 4, pp. 149–160, 1996.

[8] M. R. Berthold and J. Diamond, "Constructive training of probabilistic neural networks," *Neurocomputing*, vol. 19, pp. 167–183, 1998.

[9] F. Delgosha and M. B. Menhaj, "Fuzzy probabilistic neural networks: A practical approach to the implementation of bayesian classifier," in *Fuzzy Days*, ser. LNCS, vol. 2206. Berlin, Germany: Springer-Verlag, 2001, pp. 76–85.

[10] R. K. Y. Chang, C. K. Loo, and M. V. C. Rao, "Enhanced probabilistic neural network with data imputation capabilities for machine-fault classification," *Neural Computing and Applications*, vol. 18, pp. 791–800, 2009.

[11] O. Polat and T. Yildirim, "FPGA implementation of a general regression neural network: An embedded pattern classification system," *Digital Signal Processing*, vol. 20, pp. 881–886, 2010.

[12] A. P. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[13] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Boston, MA: Addison-Wesley, 2006.

[14] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, ser. Intelligent Robotics and Autonomous Agents. Cambridge, MA: MIT Press, 2006.

[15] N. Bhattacharyya, A. Metla, R. Bandyopadhyay, B. Tudu, and A. Jana, "Incremental PNN classifier for a versatile electronic nose," in *Proc. 3rd Int. Conf. Sensing Technology*. Tainan, Taiwan: IEEE Press, Nov. 2008, pp. 242–247.

[16] T. Hoya, "On the capability of accommodating new classes within probabilistic neural networks," *IEEE Trans. Neural Networks*, vol. 14, no. 2, pp. 450–453, Jul. 2003.

[17] M. L. Yoshida and E. R. Hruschka Jr., "Quasi-incremental bayesian classifier," in *Proc. Int. Workshop on Knowledge Discovery from Ubiquitous Data Streams (IWKDUDS-2007)*, vol. 1, Warsaw, Poland, Sep. 2007.

[18] M. R. Heinen and P. M. Engel, "An incremental probabilistic neural network for regression and reinforcement learning tasks," in *Proc. 20th Int. Conf. Artificial Neural Networks (ICANN 2010)*, ser. LNCS. Thessaloniki, Greece: Springer-Verlag, Sep. 2010, to appear.

[19] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2008.

[20] P. M. Engel and M. R. Heinen, "Incremental learning of multivariate gaussian mixture models," in *Proc. 20th Brazilian Symposium on AI (SBIA)*, ser. LNCS. SÃ£o Bernardo do Campo, SP, Brazil: Springer-Verlag, Oct. 2010, to appear.

[21] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[22] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.

[23] S. Nolfi and J. Tani, "Extracting regularities in space and time through a cascade of prediction networks: The case of a mobile robot navigating in a structured environment," *Connection Science*, vol. 11, no. 2, pp. 125–148, 1999.