The Application of a Radial Basis Function Network to Supervised Terrain Classification

[1]Tiny du Toit and [2]Hennie Kruger
School of Computer Science and Information Systems
North-West University
Potchefstroom, South Africa
e-mail: [1]Tiny.DuToit@nwu.ac.za, [2]Hennie.Kruger@nwu.ac.za

***Abstract*—In this paper, inertial contact sensor-based terrain classification is performed with a Radial basis function network. Compared to the more popular Multilayer perceptrons, Radial basis function networks are also intelligent techniques and universal approximators, but with a much simpler structure and shorter training time. It has been shown that Radial basis function networks are efficient classifiers and, consequently may be used for terrain classification. For the experiments, a mobile robot platform recorded vibration training data with an inertial measurement unit while traversing five different terrains: asphalt, carpet, dirt, paving, and tiles. The composition of these terrains induces specific vibrations in the mobile platform, which are measured by the inertial measurement unit. The vibration signatures comprise the mobile robot's linear acceleration, orientation, and the earth's magnetic field. In contrast to most terrain classification techniques found in literature, no pre-processing of the data is performed. This reduces the computational overhead needed for real-time classification. A Radial basis function network is then trained using a hybrid conjugate gradient descent method and *k*-fold cross-validation. Identification of the terrain is performed in real time. The classification capability is empirically compared to that obtained by a Multilayer perceptron, a Naïve Bayes method and a Support Vector Machine, which have also been successfully applied to terrain classification in literature. It was found that the Radial basis function network outperformed the Support Vector Machine and Naïve Bayes techniques by a relatively large margin. The Multilayer perceptron, although performing slightly better than the Radial basis function network, has some disadvantages compared to the Radial basis function network. Consequently, the Radial basis function network, with no pre-processing of the input data, may be used successfully as an alternative contact sensor-based terrain classification method.**

Keywords–classification; inertial measurement unit; MLP; RBFN; sensor; terrain classification.

I. INTRODUCTION

Mobile robots are employed on various types of terrain [1] in many different operational fields, such as supply and logistics, surveillance, search and rescue missions, agricultural applications, transportation, cleaning, inspection and entertainment [2][3]. For these operations, it may be necessary to traverse some indoor or off-road terrain which might influence the vehicle's performance. The efficiency of these vehicles can be improved by their detection of their environment. This act of identifying the type of terrain being traversed from a list of candidate terrains such as dirt, sand, or gravel, is called terrain classification [4].

Factors, such as friction, cohesion, damping, stiffness and surface irregularity comprise the terrain interface that is presented to the mobile robot [5]. It may be beneficial to identify the current terrain type as the terrain conditions may have an influence on both the planning stages and motion control of the vehicle's trip. Once the mobile robot's control system has knowledge of the surface on which it is travelling, it will be easier to maneuver over uneven terrain or around obstacles, which allows the vehicle to traverse the terrain most effectively. In particular, awareness of the terrain type will enable the vehicle to drive at higher speeds, enable the mobile robot to choose an appropriate driving mode, prevent physical damage, keep wheels from sinking into the ground and obtain an automated driving process which is terrain-dependent.

Research on the identification of terrain types can be divided into two groups: methods relying on noncontact sensors [4] - [8] and methods utilizing contact sensors [9] - [12]. Examples of noncontact sensors are vision sensors and laser scanners. A vision sensor, such as a charge-coupled device (CCD) camera, uses techniques that extract textures and colors from the sensor data to classify this information into variable terrains, like forests and the sky. Unfortunately, the performance of these techniques is highly dependent on environmental factors, such as lighting conditions and climate effects and consequently, the sensor information can be distorted. Laser scanner sensor data that are obtained from a terrain are converted into frequency information. Learning algorithms then use this information to classify the terrain. A disadvantage of such a method is that it needs numerous data points which may hinder real-time classification. As the mobile robot traverses the specific terrain, these terrain properties combined with the robot dynamics produce vibrational signatures in body motion. Methods based on contact sensors, however, classify a terrain using sensor information, such as the vibration frequency or the slope ratio of the mobile robot's body into the terrain type.

The aim of this paper is to perform terrain classification using a Radial basis function network (RBFN) and then to compare the results to a Multilayer perceptron (MLP) neural network [13], the Naïve Bayes method and the Support Vector Machine (SVM) technique, which have also been

successfully applied to this problem to provide context. The main focus, however, is on the comparison between the RBFN and the MLP and, consequently only these two methods will be discussed in detail.

Broomhead and Lowe [14] proposed the RBFN in 1988. This type of neural network model forms a unifying link among many different research fields, such as pattern recognition, regularization, function approximation, noisy interpolation, and medicine. The model has become increasingly popular due to its topological structure and neurons that are tuned locally. In addition, it has become a good alternative to the MLP, since it has capabilities equivalent to those of the MLP model, but with a simpler structure and can be trained much faster. Previous studies have shown that RBFNs in general are efficient classifiers [2][15]. In one study in particular [2], a RBF network has been used for terrain classification where a Discrete Fourier transform was implemented to perform feature extraction. Unfortunately, such pre-processing of the data is a time-consuming task, which may prevent the real-time identification of the terrain.

The MLPs that are trained by the backpropagation rule is one of the most important neural network techniques used for nonlinear modeling [16]. Their greatest benefit is that no *a priori* knowledge of the particular functional form is required. Feedforward MLPs are mostly utilized to estimate relationships between input and target variables. They often exhibit superior performance in comparison to more classical methods. In contrast to common belief, they are not a black box tool. The scientific understanding of empirical phenomena subject to neural network modeling can be considerably enhanced. Formal statistical inference can be performed using estimates obtained from neural network learning as the basis. Statistical tests of specific scientific hypotheses that are of interest become possible. The capability of MLPs to extract interactive and complex nonlinear effects extends the power of such tests beyond those possible with more traditional methods, such as linear regression analysis.

Terrain classification will be performed based on real-time vibration data obtained from an inertial measurement unit (IMU) contact sensor. No pre-processing of the data as reported in some previous studies is performed. The assumption is that the output of the IMU sensor is influenced by the vibrations induced in the platform while traversing different terrains. The test vehicle, a Lego Mindstorms EV3 mobile robot, is augmented by an IMU mounted on a Raspberry Pi 2 computer. Data that is collected from the IMU on the moving test vehicle is used as the terrain signature. This signature will then be classified as one of five predetermined terrains - asphalt, carpet, dirt, paving, or tiles.

The remainder of the paper is organized as follows. In Section II, the relatively simple structure and training of the RBFN will be discussed. A variant of the gradient descent method is used for training. The well-known MLP architecture and backpropagation training algorithm are considered in Section III. Specific issues related to artificial neural network model building are examined in Section IV. Experiments performed to determine the accuracy of terrain classification using a RBFN, an MLP, the Naïve Bayes method and an SVM model will be considered in Section V. The results that were obtained will be examined in Section VI. Finally, some concluding remarks and future work will be presented in Section VII.

II. RADIAL BASIS FUNCTION NETWORKS

In this section, the RBFN architecture and training of the model will be considered.

A. Architecture

A RBFN is a feedforward neural network with three layers $(J_1 - J_2 - J_3)$ [15][17][18] as shown in Figure 1. In the input, hidden and output layers there are J_1, J_2 and J_3 neurons, respectively. The bias in the output layer is denoted by $\phi_0(\vec{x}) = 1$ while the nonlinearity at the hidden nodes is denoted by the $\phi_k(\vec{x})$'s. Each hidden layer node uses a Radial basis function (RBF), denoted by $\phi(r)$ for its nonlinear activation function. The hidden layer performs a nonlinear transformation of the input. This nonlinearity is then mapped into a new space by the output layer which acts as a linear combiner. Normally, all hidden nodes utilize the same RBF; the RBF nodes have the nonlinearity $\phi_k(\vec{x}) = \phi(\vec{x} - \vec{c}_k), k = 1, \dots, J_2$, where $\vec{c}_k$ denotes the center or prototype of the *k*th node and $\phi(\vec{x})$ is an RBF. An extra neuron in the hidden layer can model the biases of the output layer neurons. This neuron has a constant activation function $\phi_0(r) = 1$. The RBFN determines a global optimal solution for the adjustable weights in the minimum mean square error (MSE) sense by using the method of linear optimization. The output of the RBF network, provided by input $\vec{x}$, is given by

$$y_i(\vec{x}) = \sum_{k=1}^{J_2} w_{ki}\phi(\|\vec{x} - \vec{c}_k\|), i = 1, \dots, J_3, \qquad (1)$$

where $y_i(\vec{x})$ is the *i*th output, w_{ki} denotes the connection weight from the *k*th hidden neuron to the *i*th output unit, and $\|\cdot\|$ is the Euclidian norm. The RBF usually utilizes the Gaussian function $\phi(\cdot)$ and such a model is normally called the Gaussian RBF network.

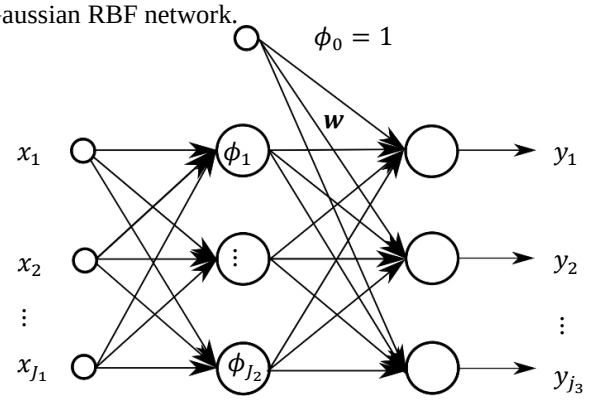

Figure 1. RBF network architecture [16]

Given a set of N pattern pairs $\{(\vec{x}_p, \vec{y}_p)|p = 1, \dots, N\}$, (1) can be expressed in matrix form as

$$\boldsymbol{Y} = W^T \Phi \tag{2}$$

where $\boldsymbol{W} = [w_1, \dots, w_{J_3}]$ is a $J_2 \times J_3$ matrix, $\vec{w}_i = (w_{1i}, \dots, w_{J_2 i})^T$, $\Phi = [\vec{\phi}_1, \dots, \vec{\phi}_N]$ is a $J_2 \times N$ matrix, $\vec{\phi}_p = (\phi_{p,1}, \dots, \phi_{p,J_2})^T$ is the hidden layer output for the pth sample, specifically, $\phi_{p,k} = \phi(\|\vec{x}_p - \vec{c}_k\|)$, $\boldsymbol{Y} = [y_1\ y_2\ \dots y_N]$ is a $J_3 \times N$ matrix, and $\vec{y}_p = (y_{p,1}, \dots, y_{p,J_3})^T$.

The RBFN is a universal approximator [17]. If the RBF is appropriately chosen, the RBF network can theoretically approximate any continuous function arbitrarily well. The Gaussian RBF is expressed as $\phi(r) = \exp(-r^2/2\sigma^2)$ where $r > 0$ represents the distance from a data point $\vec{x}$ to a center $\vec{c}$ and σ is utilized to control the smoothness of the interpolating function. The Gaussian RBF is a localized RBF with the property that $\phi(r) \to 0$ as $r \to \infty$.

Training of a RBFN is usually performed by a two-phase strategy. During the first phase, suitable centers $\vec{c}_k$ and their corresponding standard deviations, σ_k, also known as widths or radii are determined. The network weights $\boldsymbol{W}$ are adjusted in the second phase. The training approach that is followed in this research is the supervised learning of all the parameters by the relatively simple gradient descent method.

B. Training

There is one output unit for each of the five terrain class values (asphalt, carpet, dirt, paving, and tiles). The model trained for the ith output unit (class value) is given by:

$$y_i(x_1, x_2, \dots, x_m) = g\left(w_{i,0} + \sum_{k=1}^{b} w_{i,k} \exp\left(-\sum_{j=1}^{m} \frac{(x_j - c_k)^2}{2\sigma_{global}^2}\right)\right), \tag{3}$$

where the activation function $g(\cdot)$ is a logistic function [19]. A Gaussian RBF network with the same global variance parameter σ_{global} for all RBF centers still has universal approximation capability [17]. The appropriate parameter values for $w_{i,k}$ and σ_{global} are found by identifying a local minimum of the penalized squared error on the training data. Given p classes, the error function can be expressed as

$$L_{SSE} = \left(\frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{p}\left(y_{k,i} - f_i(\vec{x}_k)\right)^2\right) + \left(\lambda \sum_{i=1}^{p}\sum_{k=1}^{b} w_{i,k}^2\right), \tag{4}$$

where $y_{k,i} = 0.99$ if data point $\vec{x}_i$ has the ith class value, and $y_{k,i} = 0.01$ otherwise. Instead of using 1.0 and 0.0, the values 0.99 and 0.01 are used to aid the optimization process. Additionally, in (4), L_{SSE}, is divided by n, the number of training data points, as this was determined through empirical observation to improve convergence with the optimization methods used [20]. Standard calculus is utilized to find the corresponding partial derivatives, which consist of the gradients of the error function with respect to the network parameters. Backpropagation is employed to calculate the partial derivatives in the same manner as in Multilayer perceptrons. The hybrid conjugate gradient descent method specified by [21] is used for training.

Initialization of the network parameters is another important aspect of the training procedure. The initial weights of the output layer are sampled from $\mathcal{N}(0, 0.1)$. This strategy was empirically determined based on the familiar heuristic of choosing small, randomly distributed initial weights [20].

As the k-means algorithm is often used to train the hidden layer of the RBFN in an unsupervised process, it is utilized to determine the initial hidden unit centers c_k. Furthermore, the initial value of the variance parameter σ_{global} is set to the maximum squared Euclidian distance between any pair of cluster centers. This ensures that the initial value of the variance parameter is not too small. The learning process is accelerated on a multi-core computer by parallelizing the calculation of the error function and its gradient on a user-specified number of threads.

Artificial neural networks (ANNs) such as RBFNs and MLPs can be considered as techniques that lie in machine learning middle ground, somewhere between artificial intelligence and engineering [22]. They use heuristic methods, because very often there is no theoretical basis to support the decisions about the ANN implementation, as well as mathematical techniques, such as mean-square error minimization. ANNs are comprised of a large class of various architectures. The RBFN and MLP are two of the most widely used neural network architectures in literature for regression and classification problems [23]. To put the application of the RBFN on terrain classification in context, an MLP constructed for the same purpose is also examined. Both types of neural network structures are good in pattern classification problems and also robust classifiers with the ability to generalize for imprecise input data. A general difference between the RBFN and MLP is that the RBFN performs a local type of learning, which is responsive only to a limited section of the input space. In contrast, the MLP is a more distributed type of approach. The output of an RBFN is produced by mapping distances between the input vectors and center vectors to outputs through a radial function, whereas the MLP output is produced by linear combinations of the outputs of hidden layer nodes in which a weighted average of the inputs is mapped by every neuron through a sigmoid function. In the next section, the MLP architecture and training procedure are considered.

III. MULTILAYER PERCEPTRONS

Similar to a RBFN, the MLP neural network is capable of arbitrary input-output mapping [24]. With its powerful universal approximation capability, it has been shown that MLPs with an appropriate number of hidden neurons can implement any continuous function. The MLP is extensively used in classification, regression, prediction, system identification, control, feature extraction, and associative memory. An MLP, like a RBFN, is estimated by a supervised procedure where the network constructs the model based on examples in the data with known outputs.

A. Architecture

In most cases, an MLP has several layers of nodes. External information is received at the first or lowest layer. The problem solution is obtained at the highest layer which is an output layer. Between the input layer and output layer there are one or more intermediate layers called the hidden layers. The number of hidden layers is a very important parameter in the network. Bordering nodes are normally fully connected from a lower layer to a higher layer. No lateral connection between neurons in the same layer, or feedback connection is possible. The MLP estimates a functional relationship, which can be written as $y = f(x_1, x_2, \dots, x_m)$, where $x_1, x_2, \dots, x_m$ are m independent variables and y is the dependent variable. Functionally, the MLP in this sense is equivalent to a nonlinear multiple regression model.

A single hidden layer MLP network with h neurons (Figure 2) and c outputs has the following form:

$$y_c(x_1, x_2, \dots, x_m) = g\left(w_0 + \sum_{k=1}^{h} w_k \tanh\left(w_{0k} + \sum_{j=1}^{m} w_{jk} x_j\right)\right), \quad (5)$$

where $g(\cdot)$ is the activation function, and w_i, w_{jk} the weights.

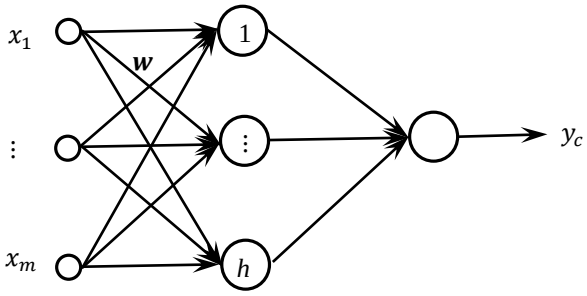

Figure 2. MLP network architecture

The model in (5) can be expressed in matrix form as $y_c = g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, where y_c is the output, $g(\cdot)$ the activation function, $\boldsymbol{W} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,m} \\ \cdots & \cdots & \cdots \\ w_{h,1} & \cdots & w_{h,m} \end{bmatrix}$ a $[H \times M]$ weight matrix, $\boldsymbol{x} = [x_1, x_2, \dots, x_m]$ the input vector, and $\boldsymbol{b} = [b_1, b_2, \dots, b_h]$ the bias vector [18].

B. Training

The backpropagation algorithm used to train an MLP was first discovered by [25] and later popularized by [26]. During the training phase, a set of input-output pairs is utilized for training and is repeatedly presented to the network. When training is stopped, the performance of the network is tested. The learning algorithm includes a forward-propagating step, followed by a backward-propagating step. On the whole, the algorithm is as follows:

***input**: training set, weight vector w*
***output**: optimal weight vector w**
repeat
  repeat
    repeat
      Initialize the weights w to small random values.
      Select an instance t, which is a data point from the training set.
      Apply the network input vector to the network.
      Calculate the network output vector z.
      For each of the outputs c, calculate the errors, which is the difference (δ) between the target output and the network output.
      Minimize this error by calculating the necessary updates for the weights (Δw).
      Add the calculated weights' updates (Δw) to the accumulated total updates (ΔW).
    ***until** number of instances comprises an epoch*
    Adjust the weights (w) of the network by ΔW.
  ***until** all instances in the training set are considered. This forms one iteration.*
***until** the error for the entire system (error δ or cross-validation set error) is satisfactorily low, or a pre-defined number of iterations is completed.*

Algorithm 1. Backpropagation algorithm

During training, the backpropagation algorithm performs gradient descent on the error surface by adjusting each weight in proportion to the gradient of the error surface at its location. It is well known that gradient descent can sometimes cause networks to get stuck in a local minimum in the error surface should such a local minimum exist. These local minima correspond to a partial solution for the network given the training data. At best, a global minimum is desired (the lowest error value possible), however, the local minima are surrounded by higher error values and the network usually does not escape these local minima by employing the standard algorithm. To get out of a local minimum, special techniques should be used. These include

varying the number of hidden units, changing the learning parameter (α), but especially by using the momentum term (η) in the algorithm. This term is generally chosen between 0 and 1. Taking into account the momentum term, the formula for modifications of weights at epoch $t + 1$ is given by

$$\Delta w_{kj}(t+1) = \eta \delta_k x_m + \alpha \Delta w_{kj}(t), \quad (6)$$

where j denotes the specific neuron. The network can oscillate, or more seriously, get stuck in a local minimum with incorrect values of these parameters.

Regardless of the many favorable characteristics of ANNs, constructing a neural network model for a particular problem is a nontrivial task [24]. Modeling issues that have an effect on the performance of an ANN must be carefully taken into account to ensure the successful application of the ANN. These issues are briefly examined next.

IV. ARTIFICIAL NEURAL NETWORK MODELING ISSUES

One of the critical decisions that must be made when building an ANN model is to determine a suitable architecture, specifically the number of layers, the number of nodes in each of the layers, and the number of connections that join the nodes. Additional network design decisions comprise the choice of activation functions for the hidden and output nodes, the training algorithm, data normalization or transformation methods, training and test data sets, and performance metrics.

A. Network architecture

An ANN is normally formed by layers of nodes. All the input nodes are grouped in the input layer, all the output nodes are in the output layer and the hidden nodes are allocated in one or more hidden layers in the middle. When constructing the ANN, the following variables must be determined:

- the number of input nodes;
- the number of hidden layers and hidden nodes; and
- the number of output nodes.

Selection of these parameters is inherently dependent on the problem. Many different methods to determine the optimal architecture of an ANN exist, but many of these methods are relatively complex in nature and difficult to implement. Examples include the network information criterion [27], the polynomial time algorithm [28], the canonical decomposition technique [29] and the pruning algorithm [30][31]. In addition, none of these methods is able to guarantee the optimal solution for all problems. Currently, there is no simple explicit method to choose these parameters. The guidelines are either based on simulations obtained from limited experiments or heuristic in nature. Therefore, the design of an ANN can be considered more of an art than a science.

A.1 Number of input nodes

The number of input nodes coincide with the number of variables in the input vector used to model target values. Given a specific problem, the number of inputs is usually transparent and relatively easy to choose.

A.2 Number of hidden layers and nodes

Many successful applications of neural networks are highly dependent on the hidden layer(s) and nodes. The hidden nodes in the hidden layer(s) enable a neural network to detect features, capture patterns in the data and to perform complex nonlinear mappings between input and output variables. It is evident that without hidden nodes, simple perceptrons with linear output nodes are equivalent to linear statistical forecasting models. Since theoretical works show that a single layer is sufficient for ANNs to approximate any complex nonlinear function to any desired accuracy [32], a single hidden layer is often used for modeling purposes. Unfortunately, one hidden layer networks may involve a very large number of hidden nodes, which is undesirable in that the network generalization ability and training time will get worse. Two or more hidden layer MLPs may provide more benefits for some types of problem [33][34]. Many authors focus on this problem by considering more than one hidden layer.

Determining the optimal number of hidden nodes is a crucial yet complicated issue. In most cases, networks with fewer hidden nodes are favored as they overfit less and usually have a better generalization ability. However, networks with too few hidden nodes may not have enough power to model and learn the data. There is no theoretical principle for choosing this parameter though a number of systematic approaches exist. Methods for increasing hidden nodes and pruning out unwanted hidden nodes have been proposed. A grid search method used to determine the optimal number of hidden nodes was put forward by [35]. The most common way to establish the number of hidden nodes is by means of experiments or trial-and-error. Various rules of thumb have also been suggested such as each weight should have at least ten input data points (referring to the sample size), and the number of hidden nodes should be determined by the number of input patterns. Some researchers have presented empirical rules to assist in avoiding the overfitting problem by restricting the number of hidden nodes [24]. Additionally, the number of hidden nodes was limited by a heuristic constraint by [36]. A number of practical guidelines exist in the case of the common one hidden layer networks, which include $\frac{n}{2}$ [37], $2n$ [38] and $2n + 1$ [39], where n denotes the number of input nodes. Nevertheless, none of these heuristic choices works well for all problem contexts.

A.3 Number of output nodes

As in the case of the number of input nodes, the number of output nodes is relatively easy to determine as it is directly related to the problem being modeled.

B. Interconnection of nodes

The behavior of a network is essentially determined by the connections between nodes. In most applications, the networks are fully connected with all nodes in one layer being connected to all the nodes in the next, higher layer, excluding the output layer. Sparsely connected networks [40] or direct connections between input nodes and output nodes [41] are, however, possible. The latter may be beneficial to predictive accuracy since it can be utilized to model the linear structure in the data and might increase the recognition power of the network.

C. Activation function

The activation function determines the relationship between the inputs and outputs of a neuron and the rest of a network. This function establishes a degree of nonlinearity that is valuable for most ANN applications. In theory, any differentiable function can be used as an activation function, but in practice, only a small number of activation functions are used. Some heuristic rules exist for the selection of the activation function. When learning about average behavior such as terrain classification, [42] suggests logistic activation functions.

D. Training algorithm

Training of a neural network is an unconstrained nonlinear minimization problem where weights of a network are iteratively adjusted to minimize the overall squared error or mean between the actual and desired output values for all the output nodes over all inputs patterns. Many different optimization methods to use for neural network training exist. Currently, there is no algorithm available to guarantee the global optimal solution for a general nonlinear optimization problem in a reasonable amount of time. In practice, all optimization algorithms suffer from the local optima problem. A solution to this problem is to use the available optimization method, which produces the "best" local optima if the true global solution is not available. The backpropagation algorithm is the most widely used training method.

E. Data normalization

Nonlinear activation functions such as the hyperbolic tangent function usually have a squashing role in restricting or compressing the possible output from a node to typically (0,1) or (-1, 1). Often, data normalization is performed before the training process begins. When nonlinear activation functions are used at the output nodes, the desired output values must be transformed to the range of the actual network outputs. Even when a simple linear output transfer function is utilized, it may still be beneficial to standardize the outputs together with the inputs to facilitate network learning, meet algorithm requirements and to avoid computational problems. Four methods to normalize inputs are presented by [43]: along channel (independent input variable) normalization, across channel (each input vector independently) normalization, mixed channel (combinations of along and across) normalization, and external normalization where all the training data are normalized into an explicit range.

F. Training and test samples

A training and test sample are typically involved when building an ANN model. The training sample is used for developing the model and the test sample for evaluating the predictive ability of the model. At times a set called the validation sample is also put to use to avoid the overfitting problem or to determine a stopping point for the training process. An important issue is the division of the data into the training and validation sets. One common approach is to use k-fold cross-validation [44] where a data set ($\mathcal{D}$) is randomly split into k mutually exclusive subsets (the folds) $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$. A model is then trained and tested k times; for each time $t \in \{1, 2, \ldots, k\}$, it is trained on $\mathcal{D} \setminus D_t$ and tested on $\mathcal{D}_t$. The cross-validation estimate of accuracy is the overall number of correct classifications divided by the number of instances in the data set.

G. Performance measures

In spite of many performance measures for an ANN model, such as training time and modeling time, the most important measure of performance is the prediction accuracy the model can produce beyond training data. Nevertheless, academics and practitioners do not universally accept a suitable measure of accuracy for a given problem. An accuracy measure is frequently defined in terms of the prediction error, which is the difference between the desired (actual) and the predicted value. There are a number of accuracy measures in the prediction literature and each has its advantages and limitations [45].

In the next section, the experiments that are performed to determine the RBFN terrain classification accuracy and comparison with an MLP, Naïve Bayes method and SVM will be discussed. Modeling issues in Section IV will be taken into account to construct the best RBFN and MLP architectures.

V. EXPERIMENTAL DESIGN

The aim of the experiments is to identify the type of terrain being traveled on by a mobile robot from a list of candidate terrains. Figure 3 shows the Lego Mindstorms EV3 experimental platform used in the investigation. The mobile robot has a Raspberry Pi 2 computer attached to the

front with a Sense HAT inertial measurement unit (IMU) in turn connected to the Raspberry Pi. The Sense HAT is readily available and includes the following sensors: a gyroscope, an accelerometer, and a magnetometer. The mobile robot platform is battery powered and moves on rubber treads. An additional battery pack (not shown) is mounted on top and powers the Raspberry Pi computer. The five terrain types used in the study are displayed in Figures 4 to 8.

Figure 3. Lego Mindstorms EV 3 mobile robot

The terrain (asphalt, carpet, dirt, paving, or tiles) on which the mobile robot is currently travelling is identified in real time. The assumption is that the vibrations induced in the test vehicle and measured by the output of the IMU sensor represent a signature which can be used to accurately classify a terrain. The data for each terrain is sampled at an irregular rate of $\approx 16\frac{2}{3}$ Hz for a 600-second duration. The RBFN is then trained offline using the RBFN training scheme discussed in Section II (B) and the MLP by the backpropagation algorithm discussed in Section III (B). Three outdoor terrains (asphalt, dirt, and paving) and two indoor terrains (carpet and tiles) were analyzed.

Figure 4. Asphalt

Figure 5. Carpet

The RBFN architecture for this specific problem has five outputs that serve to identify the terrain type. Each of the output values $y_i \in [0,1]$ denotes the likelihood that a given signal presented as an input to the RBFN matches one of the five candidate terrains. In addition, the RBFN architecture has twelve inputs, which correspond to the dimension of the input signal data point. Each of these input signal data points received from the Sense HAT IMU can be denoted as:

$$[p\ r\ y\ a_x\ a_y\ a_z\ g_x\ g_y\ g_z\ m_x\ m_y\ m_z],$$

where $p, r,$ and y denote the pitch, roll and yaw (measured in degrees), a is the linear acceleration (m/s^2) measured in three dimensions (a_x, a_y and a_z), g is the rate of turn (degrees/seconds), also measured in three dimensions (g_x, g_y and g_z) and m denotes the earth's magnetic field

(gauss), measured in three dimensions (m_x, m_y and m_z) of the mobile robot, respectively.

Figure 6. Dirt

Figure 7. Paving

Figure 8. Tiles

The Weka system [20] was used for data processing, presentation, classifier training and testing. The terrain classification training data set contained twelve inputs, five outputs and a total of 49993 IMU sensor samples.

Before training started, all inputs in the data were normalized to the [0, 1] interval. This data was transformed back into the original space when predictions were produced. These same transformations were performed for new inputs when the predictions were made.

For the experiments, 10-fold cross-validation was performed. Results obtained by the RBFN were compared to those found by the MLP model, and default SVM and Naïve Bayes techniques. The latter are two popular methods found in the literature used for supervised terrain classification [11][12]. Sigmoid activation functions were utilized for the hidden and output nodes of the MLP architecture. A grid search was applied to obtain the best number of RBFN and MLP hidden nodes and number of hidden layers for the MLP. Coincidentally, the best results were obtained with 120 hidden nodes for both the RBFN and single hidden layer fully connected MLP. In the following section, the results will be discussed.

VI. DISCUSSION

The classification accuracy results obtained by the experiments are shown in Figure 9.

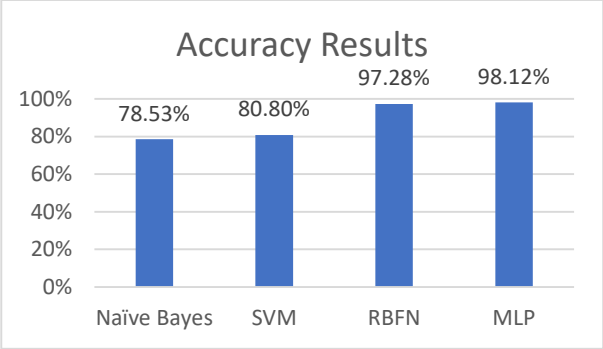

Figure 9. Terrain classification results

From the figure it can be observed that the machine learning algorithms, ordered from worst classification accuracy to best, are the Naïve Bayes, SVM, RBFN and MLP. The latter two techniques produced nearly the same classification accuracy with the MLP slightly outperforming the RBFN. The results presented in Figure 9 show that the RBFN is a feasible terrain classification technique compared to the former two models. Although the MLP exhibited a slightly higher classification performance than the RBFN, the results indicate the good predictive capability of the RBFN. In addition, the RBFN applied to terrain classification has the following advantages and disadvantages:

- Compared to the MLP, the RBFN has less model complexity, exhibits better comprehensibility, is easier to construct due to its simpler fixed three-layer structure and has a fast learning algorithm.
- Special techniques exist to increase the interpretability of RBFNs, thereby reducing the black box effect of neural networks in general [46].
- With regards to generalization, RBFNs can respond well for patterns which are not used for training [47].
- The stability of the designed RBFN model is enhanced by its strong tolerance to input noise [47].
- An ensemble of RBFN models can be constructed to increase the accuracy of a RBFN model. In some cases, this ensemble model can surpass an MLP model [48] in terms of classification accuracy.
- No pre-processing of the input sensor data is performed as in some previous studies.
- Classification of the terrain can be performed in real time because of the onboard IMU contact sensor.
- In terms of predictive accuracy, the RBFN outperformed the Naïve Bayes technique and the SVM model by a relatively large margin.
- A limitation of the RBFN model, however, is that it has greater difficulties if the number of hidden units is large and it is more sensitive to dimensionality [23].

Based on the small difference in classification accuracy between the RBFN and the MLP and the advantages of the RBFN, it can be concluded that it is reasonable to consider the RBFN as a competitive method for supervised terrain classification.

VII. Conclusion and Future Work

In this paper, real-time classification of five given terrains was performed with a RBFN. In contrast to some other techniques found in the literature, no pre-processing of the mobile robot platform's IMU vibration sensor data was performed. Eliminating feature extraction reduces the computational overhead needed to identify the terrain in real-time. The results have shown that even without feature extraction, the RBFN is still a feasible model for contact sensor-based terrain classification compared to other popular models used for this task. It can be used as an alternative to the MLP model due to its simpler structure and shorter training times. The RBFN has the capability to accurately recognize complex vibration signature patterns and can easily adapt to new terrain signatures by providing the model with new training examples. Unfortunately, compared to the other techniques, offline training of the model can be time consuming.

Future work includes a more detailed comparison with the existing methods. Metrics, such as latency (velocity) can be included in the results. Finally, an investigation into the feasibility of the RBFN model applied to other types of robots and how they must be adapted for this task can be performed.

Acknowledgment

The authors would like to thank Mr. Ryno Marx for assembling the mobile robot platform and for acquiring the vibration sensor data for the five terrains.

References

[1] J. V. du Toit and H. A. Kruger, "Terrain Classification Using a Radial Basis Function Network," in Proceedings of the Twelfth International Multi-Conference on Computing in the Global Information Technology (ICCGI), Nice, France, 2017, pp. 11-16.

[2] T. Kurban and E. Besdok, "A Comparison of RBF neural network training algorithms for inertial sensor-based terrain classification," Sensors, vol. 9, pp. 6312-6329, 2009.

[3] D. Sadhukhan, "Autonomous ground vehicle terrain classification using internal sensors," Florida State University, Master's thesis, 2004.

[4] L. Ojeda, J. Borenstein, G. Witus, and R. Karlsen, "Terrain characterization and classification with a mobile robot," Journal of Field Robotics, vol. 23(2), pp. 103-122, 2006.

[5] F. L. Garcia Bermudez, R. C. Julian, D. W. Haldane, P. Abbeel, and R. S. Fearing, "Performance analysis and terrain classification for a legged robot over rough terrain," "IEEE/RSJ International Conference on Intelligent Robots and Systems", Vilamoura, Algarve, Portugal, October 7-12, 2012.

[6] A. Angelova, L. Matthies, D. Helmick, and P. Perona, "Fast terrain classification using variable-length representation for autonomous navigation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Minneapolis, MN, USA, 2007, pp. 1-8.

[7] A. Talukder et al., "Autonomous terrain characterization and modelling for dynamic control of unmanned vehicles," in Proceedings of the IEEE Conference on Intelligent Robots and Systems (IROS), Lausanne, Switzerland, 2002.

[8] R. Manduchi, A. Castano, A. Talukder, and L. Matthies, "Obstacle detection and terrain classification for autonomous off-road navigation," Autonomous Robots, vol. 18, pp. 81-102, 2005.

[9] B. Park, J. Kim, and J. Lee, "Terrain feature extraction and classification for mobile robots utilizing contact sensors on rough terrain," Procedia Engineering, vol. 41, pp. 846-853, 2012.

[10] R. Jitpakdee and T. Maneewarn, "Neural networks terrain classification using inertial measurement unit for an autonomous vehicle," SICE Annual Conference, The University Electro-Communications, Japan, 2008.

[11] C. C. Ward and K. Iagnemma, "Speed-independent vibration-based terrain classification for passenger vehicles," Vehicle System Dynamics, vol. 47, no. 9, pp. 1095-1113, 2009.

[12] M. Happold, M. Ollis, and N. Johnson, "Enhancing supervised terrain classification with predictive unsupervised learning," Robotics: Science and Systems II, University of Pennsylvania, Philadelphia, 2006.

[13] T. Y. Kim, G. Y. Sung, and J. Lyou, "Robust terrain classification by introducing environmental sensors," IEEE International Workshop on Safety Security and Rescue Robotics (SSRR), 2010.

[14] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," Complex Systems, vol. 2, no. 3, pp. 321-355, 1988.

[15] C. S. K. Dash, A. K. Behera, S. Dehuri, and S.-B. Cho, "Radial basis function neural networks: a topical state-of-the-art survey," Open Computer Science, vol. 6, no. 1, pp. 33-63, 2016.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, D. E. Rumelhart and J. L. McClelland, Eds., vol. 1, pp. 318-362, MIT Press, Cambridge, Mass, USA, 1986.

[17] Y. Wu, H. Wang, B. Zhang, and K.-L. Du, "Using radial basis function networks for function approximation and classification," International Scholarly Research Network, Applied Mathematics, Volume 2012, doi:10.5402/2012/324194.

[18] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, "Neural network design," 2nd edition, Martin Hagan, USA, 2014.

[19] E. Frank, "Fully supervised training of Gaussian Radial basis function networks in WEKA," [Online]. http://www.cs.waikato.ac.nz/~ml/publications/2014/rbf_networks_in_weka_description.pdf 2018.11.10.

[20] E. Frank, M. A. Hall, and I. H. Witten, "The WEKA workbench. Online appendix for 'Data mining: Practical machine learning tools and techniques'," Morgan Kaufmann, Fourth Edition, 2016.

[21] Y. H. Dai and Y. Yuan, "An efficient hybrid conjugate gradient method for unconstrained optimization," Annals of Operations Research, 103, pp. 33-47, 2001.

[22] S. E. Jørgensen and B. D. Fath, "Multilayer Perceptron," Encyclopedia of Ecology, Academic Press, pp. 2455-2462, 2008.

[23] I. Yilmaz and O. Kaynar, "Multiple regression, ANN (RBF, MLP) and ANFIS models for prediction of swell potential of clayey soils," Expert Systems with Applications, 38, pp. 5958-5966, 2011.

[24] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," International Journal of Forecasting, vol. 14, pp. 35-62, 1998.

[25] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," PhD thesis, Harvard University, 1974.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", Nature, 323 (6088):533–536, 1986.

[27] N. Murata, S. Yoshizawa, and S. Amari, "Network information criterion determining the number of hidden units for an artificial neural network model," IEEE Transactions on Neural Networks, 5 (6), pp. 865-872, 1994.

[28] A. Roy, L. S. Kim, and S. Mukhopadhyay, "A polynomial time algorithm for the construction and training of a class of multilayer perceptrons," Neural Networks, 6, pp. 535-545, 1993.

[29] Z. Wang, C. D. Massimo, M. T. Tham, and A. J. Morris, "A procedure for determining the topology of multilayer feedforward neural networks," Neural Networks, 7 (2), pp. 291-300, 1994.

[30] J. Sietsma and R. Dow, "Neural net pruning–Why and how?," In: Proceedings of the IEEE International Conference on Neural Networks, 1, pp. 325-333, 1988.

[31] M. Cottrell, B. Girard, Y. Girard, M. Mangeas, and C. Muller, "Neural modeling for time series: a statistical stepwise method for weight elimination," IEEE Transactions on Neural Networks, 6 (6), pp. 1355-1364, 1995.

[32] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematical Control Signals Systems, 2, pp. 303–314, 1989.

[33] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," MIT Press, 2016.

[34] A. R. Barron, "A comment on Neural networks: A review from a statistical perspective," Statistical Science, 9 (1), pp. 33-35, 1994.

[35] W. L. Gorr and D. Nagin, J. Szczypula, "Comparative study of artificial neural network and statistical models for predicting student grade point averages," International Journal of Forecasting, 10, pp. 17-34, 1994.

[36] G. Lachtermacher and J. D. Fuller, "Backpropagation in time-series forecasting," Journal of Forecasting, 14, pp. 381-393, 1995.

[37] S. Kang, "An Investigation of the Use of Feedforward Neural Networks for Forecasting," PhD Thesis, Kent State University, 1991.

[38] F. S. Wong, "Time series forecasting using backpropagation neural networks," Neurocomputing, 2, pp. 147–159, 1991.

[39] R. Hecht-Nielsen, "Neurocomputing," Addison-Wesley, Menlo Park, CA, 1990.

[40] S. T. Chen, D. C. Yu, and A. R. Moghaddamjo, "Weather sensitive short-term load forecasting using nonfully

connected artificial neural network," In: Proceedings of the IEEE/ Power Engineering Society Summer Meeting, 91, SM 449-8 PWRS, 1991.

[41] K. A. Duliba, "Contrasting neural nets with regression in predicting performance in the transportation industry," In: Proceedings of the Annual IEEE International Conference on Systems Sciences, 25, 1991, pp. 163-170.

[42] C. C. Klimasauskas, "Applying neural networks. Part 3: Training a neural network," PC-AI, May/June, 20-24, 1991.

[43] E. M. Azoff, "Neural Network Time Series Forecasting of Financial Markets," John Wiley and Sons, Chichester, 1994.

[44] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," International Joint Conference on Artificial Intelligence (IJCAI), 1995.

[45] S. Makridakis, S. C. Wheelwright, and V. E. McGee, "Forecasting: Methods and Applications," 2nd ed. John Wiley, New York, 1983.

[46] L. Wang and X. Fu, "A simple rule extraction method using a compact RBF neural network", Advances in Neural Networks (ISNN): Second International Symposium on Neural Networks, Lecture notes in Computer Science, 3496, pp. 682-687, 2005.

[47] H. Yu, T. Xie, S. Paszczyñski, and B. M. Wilamowski, "Advantages of Radial Basis Function Networks for Dynamic System Design", IEEE Transactions on Industrial Electronics, vol. 58, no. 12, 2011.

[48] B. T. Pham, A. Shirzadi, D. T. Bui, I. Prakash, and M.B. Dholakia, "A hybrid machine learning ensemble approach based on a Radial Basis Function neural network and Rotation Forest for landslide susceptibility modeling: A case study in the Himalayan area, India", International Journal of Sediment Research, vol. 33, no. 2, pp. 157-170, 2018.