International Journal of Soft Computing 6 (5-6): 190-209, 2011
ISSN: 1816-9503

Performance Analysis of Various Artificial Intelligent Neural Networks for GPS/INS Integration

[1]M. Malleswaran, [2]V. Vaidehi, [3]A. Saravanaselvan and [1]M. Mohankumar
[1]Department of Electronics and Communication Engineering,
Anna University of Technology Tirunelveli, Tirunelveli, India
[2]Department of Computer Technology, MIT Campus, Anna University, Chennai, India
[3]Department of Electricals and Electronics Engineering, National Engineering College, Kovilpatti, India

Abstract: Aircraft system mainly relies on Global Positioning System (GPS) to provide accurate position values consistently. However, GPS receivers may encounter frequent GPS absence due to ephemeric error, satellite clock error, multipath error and signal jamming. To overcome these drawbacks generally GPS is integrated with Inertial Navigation System (INS) mounted inside the vehicle to provide a reliable navigation solution. INS and GPS are commonly integrated using a Kalman Filter (KF) to provide a robust navigation solution. In the KF approach the error model of both INS and GPS are required, this leads to the complexity of the system. This research work presents New Position Update Architecture (NPUA) which consists of various Artificial Intelligent Neural Networks (AINN) that integrates both GPS and INS to overcome the drawbacks in Kalman filter. The various artificial intelligent neural networks that includes both Static and dynamic networks described for the system are Radial Basis Function Neural Network (RBFNN), Back Propagation Neural Network (BPN), Forward only Counter Propagation Neural network (FCPN), Full Counter Propagation Neural network (Full CPN), Adaptive Resonance Theory-Counter Propagation Neural network (ART-CPN), Constructive Neural Network (CNN), Higher Order Neural Networks (HONN) and Input Delayed Neural Networks (IDNN) to predict the INS position error during GPS absence, resulting in different performance. The performance of the different AINNs are analyzed in terms of Root Mean Square Error (RMSE), Performance Index (PI), Number of epochs and Execution Time (ET).

Key words: GPS, INS, NPUA, KF, AINN, RBFNN, BPN, FCPN, Full CPN, ART-CPN, CNN, HONN, IDNN

INTRODUCTION

Navigation using GPS: The last two decades have shown an increasing trend in the use of positioning and navigation technologies in land vehicle applications. Most of the current vehicular navigation systems rely on the Global Positioning System that is capable of providing accurate position and velocity information. To be able to provide such accurate measurements, the GPS needs at least four satellites with good geometry. In addition, there must be a direct line of sight between the GPS antenna and those satellites. Unfortunately, this is not always possible since a GPS signal may be lost during jamming when driving around obstacles like driving through tall buildings, overpasses or tunnels on highways, tree-lined streets, etc. or when operating in poor weather conditions. The satellite signal blockage results in deterioration of the overall position accuracy (Sharaf and Noureldin, 2007, 2005; Noureldin *et al*., 2004).

Navigation using INS: Inertial Navigation System is a Self-Contained Autonomous System. It incorporates three orthogonal accelerometers and three orthogonal gyroscopes which measure three linear accelerations and three angular rates, respectively. A set of mathematical transformations and integrations with respect to time, known as the mechanization equation are applied to the raw measurements from the INS sensors to determine position, velocity and attitude information. Unfortunately, the INS cannot replace the GPS or operate on a stand-alone basis. During the mechanization procedure, the accuracy of INS position components deteriorates with time due to the integral sensor errors that exhibit considerable long-term growth. These errors include white noise, correlated random noise, bias instability and angle random walk. The errors are stochastic in nature and can cause a significant degradation in the INS performance over a long period of operation (Sharaf *et al*., 2005; Noureldin *et al*., 2004).

Corresponding Author: M. Malleswaran, Department of Electronics and Communication Engineering,
Anna University of Technology Tirunelveli, Tirunelveli, India

Existing INS/GPS data fusion techniques: In order to overcome the problems associated with the GPS and INS, the two systems are often paired together so that the drawbacks associated with each system are eliminated. The INS/GPS data fusion is commonly performed using a Kalman Filter (KF) in real time (Hosteller and Andreas, 1983; Noureldin *et al*., 2004, 2011). This method requires a Dynamic Model of both INS and GPS errors, a Stochastic Model of the inertial sensor errors and a priori information about the covariance and gain values of the data provided by both systems. Data fusion employing a KF has been widely used and is considered as the benchmark for INS/GPS integration. There are however, several considerable drawbacks to its use (Hosteller and Andreas, 1983; Noureldin *et al*., 2004, 2011). These include the following:

- The necessity of accurate stochastic modeling which may not be possible in the case of low cost and tactical grade sensors
- The requirement for a priori information of the system and measurement covariance matrices for each new sensor which could be challenging to accurately determine
- Relatively poor accuracy during long GPS absence
- The weak observability of some of the error states that may lead to unstable estimates of other error states
- The necessity to tune the parameters of the stochastic model and a priori information for each new sensor system

More recently, several techniques based on Artificial Intelligence (AI) have been proposed to replace KF in order to eliminate the above mentioned drawbacks (Hosteller and Andreas, 1983; Noureldin *et al*., 2004, 2011). The main idea behind all of these methods is to mimic the latest vehicle dynamics by training the AI module during the availability of the GPS signals.

In case of GPS absence, all these models operate in the prediction mode to correct the inaccuracies in INS outputs. In this research, researchers aim at developing a different artificial intelligent neural networks for GPS/INS Integration to find the optimum solution in terms of Positioning accuracy, performance index, number of epochs and execution time. Such technique combines the advantages of some of the existing models with the advantages of dynamic neural network architectures. In this way, it should be possible to model the INS position with greater accuracy and suitable tradeoff between Neural Network parameters (Kumar, 2004).

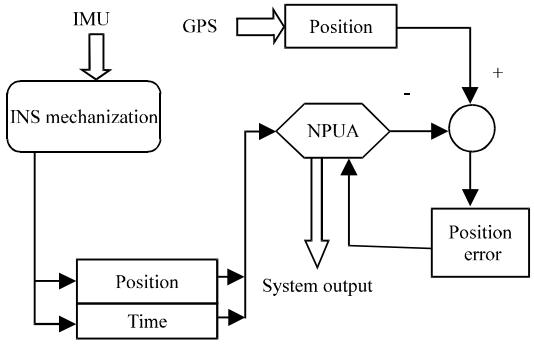

Fig. 1: System architecture

Proposed INS/GPS data fusion technique: The proposed data fusion technique introduces New Position Update Architecture (NPUA) which involves the Artificial Neural Network (ANN) in it. It is derived from the concept of Position Update Architecture (PUA) (Yun-Wen and Chiang, 2008a; Nouerldin *et al*., 2009). The NPUA can act in both prediction mode and training mode. The NPUA receives input like position and time through INS mechanization. Desired outputs are provided by system in training mode when there is no GPS absence. When there is GPS absence the system output is obtained by executing in prediction mode. The proposed system configuration is shown in Fig. 1. The different ANNs like RBF, BPN, FCPN, Full CPN, ART-CPN, CNN, HONN and IDNN can be used in NPUA. The different ANN uses different algorithms.

ARTIFICIAL INTELLIGENT NETWORKS

RBF-Radial Basis Function neural network: Radial functions are a special class of functions. Their characteristic feature is that their response decreases or increases monotonically with distance from a central point and they are radically symmetric. The centre, the distance scale and the precise shape of the radial function are parameters of the model. There are a variety of radial functions available in literature. The most commonly used one is the Gaussian radial filter which in case of a scalar input is:

$$h(x) = \exp\left(-\frac{(x-c)^2}{\beta^2}\right) \quad (1)$$

Among the static and dynamic neural networks, the RBF-NN is a commonly used structure which is shown in Fig. 2. The design of a RBF-NN in its most basic form consists of three separate layers. The input layer is the set of source nodes (sensory units). The second layer is

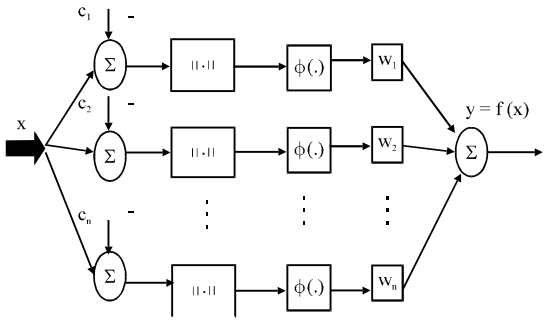

Fig. 2: Radial basis function neural network architecture

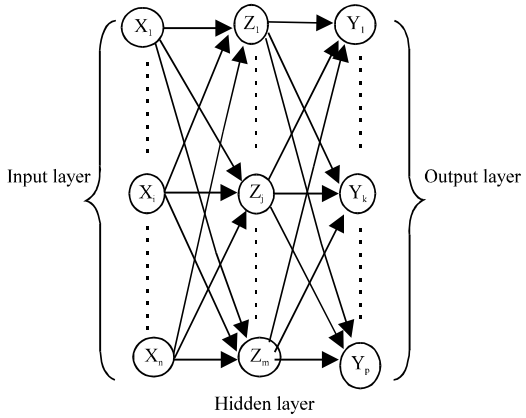

Fig. 3: BPN architecture

a hidden layer of high dimension. The output layer gives the response of the network to the activation patterns applied to the input layer. The transformation from the input space to the hidden-unit space is nonlinear. On the other hand, the transformation from the hidden space to the output space is linear.

The advantage of using RBF is simpler in architecture and it uses universal approximation methodologies. The drawbacks of using RBF is that it has fixed topology and also failed to attain the property of dynamicity.

BPN-Back Propagation Neural network: BPN uses gradient-descent based delta learning rule and Least Mean Square algorithm for the training process (Ching-Piao and Lee, 1999). The network has the following: one input layer, number of hidden layers, one output layer. The architecture of BPN is shown in Fig. 3. During the training procedure of BPN, first the input vector is presented to the input node. The weight factors and bias are initialized randomly. The hidden node (Z_j) sums its weighted signals Z_{-inj} and applies the activation function (Z_j) as shown in the following Eq. 2 and 3:

$$Z_{-inj} = V_{oj} + \sum_{i=1}^{n} X_j V_{ij} \quad (2)$$

$$Z_j = f(Z_{-inj}) \quad (3)$$

It send the signal to all the output nodes. The activation functions can be bipolar sigmoidal, binary sigmoidal, hyperbolic tangential and Gaussian. The output nodes (y_k) sums its weighted input signals (y-ink) and applies its activation function to calculate the output signal (y_k) as shown in the following Eq. 4 and 5:

$$y - inj = w_{ok} + \sum_{j=1}^{p} Z_j w_{jk} \quad (4)$$

$$y_k = f(y - ink) \quad (5)$$

The output node (y_k) receives a target pattern corresponding to an input pattern and error information is calculated as:

$$\delta_k = (t_k - y_k) f(y - inj) \quad (6)$$

The hidden nodes (Z_j) sums its delta inputs from nodes in the output layer and the error information term is calculated as:

$$\delta_{-inj} = \sum_{k=1}^{m} \delta_j w_{jk} \quad (7)$$

$$\delta_j = \delta_{-inj} f(Z_{-inj}) \quad (8)$$

The weights (V_{ij}) and bias (V_{oj}) of the network between input layer and hidden layer are updated by:

$$V_{ij}(new) = V_{ij}(old) + \Delta V_{ij} \quad (9)$$

$$V_{oj}(new) = V_{oj}(old) + \Delta V_{oj} \quad (10)$$

The weight correction term (ΔV_{ij}) and bias correction term (ΔV_{oj}) are given by the following:

$$\Delta V_{ij} = \alpha \delta_j X_i \quad (11)$$

$$\Delta V_{oj} = \alpha \delta_j \quad (12)$$

where, α is the learning rate. The weights (w_{jk}) and bias (w_{ok}) of the network between hidden layer and ouput layer are updated by:

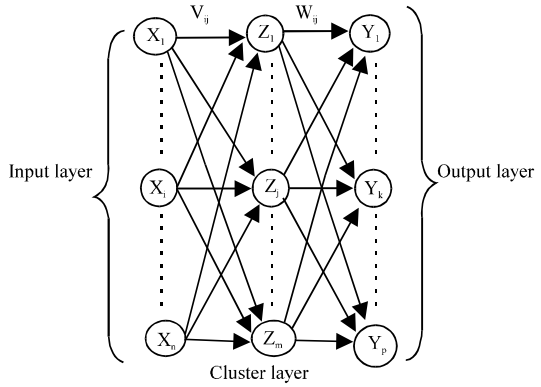

Fig. 4: Forward only CPN architecture

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} \quad (13)$$

$$w_{ok}(new) = w_{ok}(old) + \Delta w_{ok} \quad (14)$$

The weight correction term (Δw_{jk}) and bias correction term (Δw_{ok}) are given by the following:

$$\Delta w_{jk} = \alpha \delta_k Z_j \quad (15)$$

$$\Delta w_{ok} = \alpha \delta_k \quad (16)$$

The advantage of using BPN is better accurate result can be obtained using Gaussian and Binary Sigmoidal functions. The number of epochs consumed by the network is reduced when Hyperbolic Tangential (HTF) and Bipolar Sigmoidal (BPSF) functions are used.

The drawback of BPN is it has fixed topology and lack of dynamicity, the learning process has intensive calculations. Computing time is increased when increasing the hidden neurons. The number of epochs consumed by the network is more when Gaussian and Binary Sigmoidal functions are used.

FCPN-Forward only Counter Propagation Network: The Forward-only Counter Propagation Network (FCPN) is a combination of the Kohonen Self-Organizing map and the output layer. Figure 4 shows the architecture of the CPN which appears to be same as that of the back propagation net. The net consists of three layers: input layer, cluster layer (Kohonen layer) and output layer (Grossberg layer). The training procedure for the FCPN comprises two steps. First, an input vector is presented to the input node. The nodes in the cluster layer then compete among themselves (winner's take all strategy) for the right to learn the input vector. The weights of the network are adjusted automatically during the learning process. Unsupervised learning is used in this step to cluster input to separate distinct clusters of input data. Second, the weight vectors between the cluster and output layers are adjusted using supervised learning to reduce the errors between the CPN outputs and the corresponding desired target outputs (Yen-Chang, 2001).

During the 1st step, the Euclidean distance between the input and weight vectors is calculated. The winner node is selected based on comparing the input vector X $(X_1, X_2, ..., X_n)^T$ and the weight vectors V_{ij} $(V_{1j}, V_{2j}, ..., V_{nj})^T$. The winning node Z_j has the weight vector w_{jk} $(w_{1j}, w_{2j}, ..., w_{nj})^T$, winner-take-all operation that permits this cluster node J to be the most similar to the input vector. The weights of the cluster node J are adjusted. The weight vector of the winner is updated according to time t:

$$V_{iJ}(new) = (1-\alpha)V_{iJ}(old) + \alpha X_i \quad (17)$$

Where:

α = The learning rate

X_i = The ith node of input layer

After training the weights from the input layer to the cluster layer, the weights from the cluster layer to output layer are trained. Each training pattern inputs the input layer and the associated target vector is presented to the output layer. The competitive signal is a variable, assuming a value of 1 for the winning node and a value of 0 for other nodes of the cluster layer. Each output node k has a calculated input signal w_{jk} and target output y_k. The weights between the winning cluster node and the output layer nodes are updated as follows:

$$w_{Jk}(new) = (1-\beta)w_{Jk}(old) + \beta y_k \quad (18)$$

where, w_{Jk} denotes the weights from the cluster layer to output layer and represents the learning rate. The competitive signal of cluster layer Z_j is computed by:

$$Z_j = \begin{cases} 1 \text{ if } j = J, \ J \text{ is winning node} \\ 0 \ \text{ otherwise} \end{cases} \quad (19)$$

The output node k is given by:

$$y_k = \sum_{j=1}^{p} w_{jk} Z_j \quad (20)$$

y_k is the FCPN's kth computed output. The FCPN classifies the input vector to most similar cluster nodes and then outputs the prediction result. The learning speed of FCPN is fast compared to other neural networks owing to the use of the efficient learning algorithm.

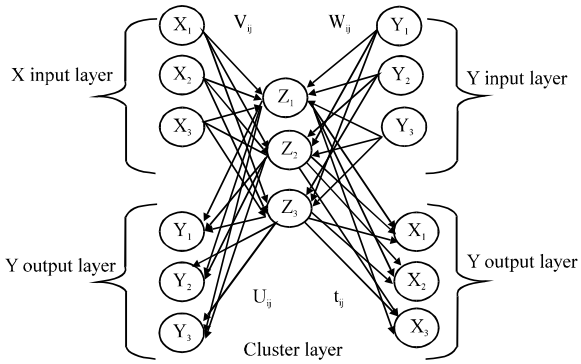

Fig. 5: Full CPN architecture

The advantage of using FCPN is that which is simple in architecture and consumes less number epochs when compared to RBF and BPN. FCPN will give better results when compared to RBF and BPN. The drawbacks of FCPN is that it has fixed topology and property of dynamicity is missing and it has two phase learning process.

Full CPN-Full Counter Propagation Neural network: The architecture of Full CPN resembles an Instar and Outstar Model. The layers included are two input layers, a common cluster layer and two output layers as shown in Fig. 5. The training phases include instar modeled training and outstar modeled training similar to the training procedure of FCPN. Full CPN also has two steps. The 1st step is carried out during instar modeled training and the second step is carried out during outstar modeled training. During the instar modeled training, the Euclidean distance between the input and weight vectors is calculated. The winner node is selected based on comparing the input vectors $X = [X_1, X_2,..., X_n]^T$ $Y = [y_1, y_2,..., y_n]^T$ and the weight vectors V_{ij} $(V_{1j}, V_{2j},..., V_{nj})^T$, w_{kj} $(w_{1j}, w_{2j},..., w_{nj})^T$, respectively. The winning node Z_j has the weight vector V_{ij} $(V_{1j}, V_{2j},..., V_{nj})^T$, w_{kj} $(w_{1j}, w_{2j},..., w_{nj})^T$, winner-take-all operation that permits this cluster node J to be the most similar to the input vector.

The weights of the cluster node J are adjusted. The weight vectors of the winner are updated using enhanced LMS weight updating rule:

$$V_{iJ}(new) = (1-\alpha)V_{iJ}(old) + \alpha X_i \quad (21)$$

$$w_{kJ}(new) = (1-\beta)w_{kJ}(old) + \beta y_k \quad (22)$$

Where:

α and β = The learning rates
X_i = The ith node of X input layer
y_k = The kth node of Y input layer

After training the weights from the input layers to the cluster layer in instar modeled training, the weights from the cluster layer to output layers are trained. Each training pattern inputs the input layers and the associated target vectors are presented to the output layers. The competitive signal is a variable, assuming a value of 1 for the winning node and a value of 0 for other nodes of the cluster layer. Each output node k has a calculated input signal U_{Jk} and target output y_k. Similarly, output node i has a calculated input signal t_{Ji} and target output X_i. The weights between the winning cluster node and the output layer nodes are updated as follows:

$$U_{Jk}(new) = (1-a)U_{Jk}(old) + ay_k^* \quad (23)$$

$$t_{Ji}(new) = (1-b)t_{Ji}(old) + bX_i^* \quad (24)$$

Where:

U_{Jk} and t_{Ji} = The weights from the cluster layer to output layers
a, b = The learning rates

The competitive signal of cluster layer Z_j is computed by:

$$Z_j = \begin{cases} 1 \text{ if } j = J,\ J \text{ is winning node} \\ 0 \text{ otherwise} \end{cases} \quad (25)$$

and the output node i and k are given by:

$$y_k^* = \sum_{j=1}^{p} U_{jk} Z_j \quad (26)$$

$$X_i^* = \sum_{j=1}^{p} t_{ji} Z_j \quad (27)$$

Where:

y_k^* = The FCPN's kth computed output
X_i^* = The Full CPN computed output

The advantage of using Full CPN is that it has simple in architecture, consumes less number of hidden neurons when compared to RBF, BPN and FCPN and produce more accurate results. The disadvantage of Full CPN is that it has two phase learning process and consume more number of epochs.

ART-CPN-Adaptive Resonance Theory-Counter Propagation Neural network: Adaptive Resonance theory nets are designed to allow the user to control the degree of similarity of patterns placed on the same cluster. The adaptive resonance theory algorithm proposed by Gross berg is a special neural network that can cluster the training patterns. The appropriate initial weight CPN's ith

vectors can be obtained in contrast to the conventional unsupervised learning algorithms. The ART-CPN combines Adaptive Resonance theory and counter propagation network to develop a new prediction network model. The parameters make the network automatically generate the algorithm redesigns the relative similarity between the input vector and the weight vectors for a cluster node. The ART-CPN algorithm uses the Kohonen and Grossberg learning rule for updating the weights of the winning node. The network has fast learning speed and good prediction performance (Grossberg, 1987).

From the geometric perspective, the similarity of two vectors is the distance metric of the two different vectors. The p-norm metric is commonly used and was defined as follows:

$$X = [X_1, X_2, ..., X_n]^T \text{ and } Y = [y_1, y_2, ..., y_n]^T$$

p-norm metric;

$$\|X - Y\| = (\sum_{i=1}^{n} |X_i - Y_i|)^{1/p}, 1 \le p \le \infty \quad (28)$$

The following conditions must be satisfied:

$$C1: d(x, y) \ge 0 \quad (29)$$

$$C2: d(x, y) = d(y, x) \quad (30)$$

$$C3: d(x, y) = d(x, z) + d(z, y) \quad (31)$$

The p-norm is generally used to determine whether two patterns are the same class. The distance between two vectors is as follows:

$$p = 1, \|X - Y\| = \sum_{i=1}^{n} |X_i - Y_i| \quad (32)$$

$$p = 2, \|X - Y\| = (\sum_{i=1}^{n} |X_i - Y_i|)^{1/2} \quad (33)$$

Equation 31 is Manhattan distance and Eq. 32 is Euclidean distance. The competition learning uses Euclidean distance for determining the winner. The Euclidean distance between the input vector and weight vectors is calculated and weight vector which has the smallest Euclidean distance from the input vector is considered as the winning unit.

ARTCPN uses the mean of Manhattan distance to calculate the similarity between the input and weight vectors. Consider m unlabeled training patterns to have n dimensional attributes using a set of the vectors (X_1, X_2, ..., X_m), the similarity between two vectors X1 and X2 is calculated as follows:

$$X_1 = [X_{11}, X_{12}, ..., X_{1n}]^T \text{ and } X_2 = [X_{21}, X_{22}, ..., X_{2n}]^T$$

Similarity between two vectors $V_{s(X_1,X_2)}$, between X_1 and X_2 is given by:

$$V_{s(X_1, X_2)} = 1 - D_{s(X_1, X_2)} \quad (34)$$

$$D_{s(X_1, X_2)} = \left[\sum_{i=1}^{n} D_{s(X_1, X_2)}\right]^{1/n} \quad (35)$$

where, $D_{s(X_1,X_2)}$ is mean of Manhattan distance. The training procedure for the forward-only counter propagation net includes two steps in learning process. The ART-CPN simultaneously trains the weights of the input layer, cluster layer and output layer. An input vector (X) is presented to the cluster node, then the $V_{s(X_j)}$ of the weight vector (V_{ij}) is calculated. Each training vector is presented to the input layer and the associated target vector is presented to the output layer. The nodes in the cluster layer compete (winner-take-all) for the right to learn the input vector. The maximum $V_{s(X_j)}$ is the winning node. The winning node sends a signal of 1 to the output layer. Each output node k has a calculated input signal w_{Jk} and target vector. The learning rule updates the weights of the winning nodes. Meanwhile, the learning rule updates the weights from the input layer to the cluster nodes as:

$$V_{iJ}(new) = (1 - \alpha)V_{iJ}(old) + \alpha X_i \quad (36)$$

where, J denotes the winning node. The learning rule updates the weights from the cluster nodes to the output layer:

$$w_{Jk}(new) = (1 - \beta)w_{Jk}(old) + \beta y_k \quad (37)$$

The competitive signal of cluster layer Z_j is computed by:

$$Z_j = \begin{cases} 1 \text{ if } j = J, \text{ J is winning node} \\ 0 \text{ otherwise} \end{cases} \quad (38)$$

The learning rule for the weights from the cluster nodes to the output nodes can be expressed using the delta function:

$$w_{Jk}(new) = (1 - \beta Z_j)w_{Jk}(old) + \beta Z_j y_k \quad (39)$$

The training of the weights from the input nodes to the cluster nodes continues at a low learning rate with gradually reducing learning rate for the weights from the cluster nodes to the output nodes. The output node k is given by:

$$y_k = \sum_{j=1}^{p} w_{jk} Z_j \quad (40)$$

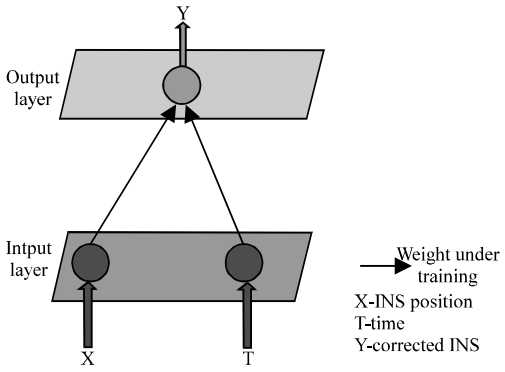

Fig. 6: Initial topology

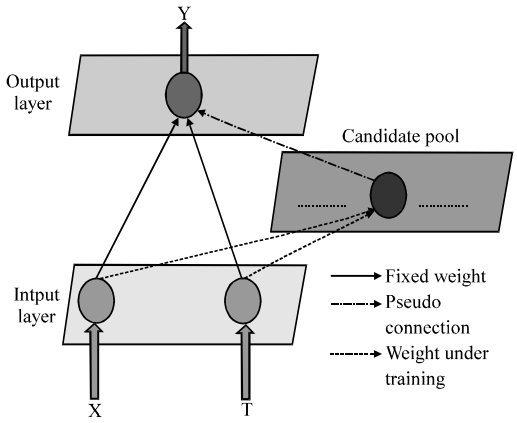

Fig. 7: Adding first hidden neuron

In this algorithm setting the vigilance parameter can generate the number of the cluster nodes. If the vigilance is set to be high, then it gathers a large number of cluster nodes. After training, the weights of the cluster nodes are distributed in a statistically optimal manner that improves the accuracy performance. The learning speed of ART-CPN is extremely fast due to the one step learning process and the efficient learning algorithm.

The advantage of using ART-CPN is that it gives more accurate results when compared with BPN, RBF, FCPN, Full CPN. Uses one way learning process and use of vigilance parameter of ART reduces the number of mathematical updates.

The convergence of solution may obtain at the very first epoch too. It consumes less number of epochs when compared to RBF, BPN, Full CPN but it failed in the aspect of producing dynamicity because of the fixed topology.

CNN-Constructive Neural Network: The CNN architecture starts with a minimal topology, consisting only of input neurons and output neurons. The 1st step of CNN training procedure begins with the minimal topology for entire training data set until no further improvement can be achieved.

The minimal topology as shown in Fig. 6 is implemented using Kohonen's algorithm (Hagan *et al*., 1996). During this process, there is no need to back propagate the output position error (between the network output and the GPS updates) through hidden neurons (Noureldin *et al*., 2004, 2011; Chiang *et al*., 2008; El-Sheimy *et al*., 2008; Hosteller and Andreas, 1983).

The second step is the recruitment of the first hidden neuron. A pool of candidate neurons that have different sets of randomly initialized weights is applied to reduce the sensitivity of initial weights. The recruitment of the 1st hidden neuron can be completed in a two-step process. During the first step of recruitment, each candidate neuron is connected to each of the input neurons but not to the output neurons. All the candidate neurons receive the trainable input connections from the external inputs and from all pre-existing hidden neurons. In addition, all candidate neurons receive the same residual error for each training pattern fed back from the output neurons as shown in Fig. 7. The weights on connecting the input neurons and candidate neurons are adjusted to maximize the correlation between the output of each candidate neuron and the residual error at the output neuron. The pseudo connection is applied to deliver the error information from the output neurons but not to forward propagate the output of candidate neurons.

A number of passes over the training data are executed and the inputs of all the candidate neurons are adjusted after each pass. The goal of this adjustment is to maximize S and to find the correlation between, the output of a candidate neuron and the residual output error observed at unit (o) as indicate in the Eq. 41:

$$S = \sum_{i=1}^{n} (Z_i - Z_1)(re_i - ar) \quad (41)$$

Where:

Z_i = Output of candidate neuron
Z_1 = Mean of output candidate neuron
re_i = Residual error at output neuron
ar = The average residual error of output neuron

The (S) indices of all the candidate neurons in the pool are computed simultaneously and the candidate neuron with the highest value of (S) is recruited after all the (S) indices stop improving.

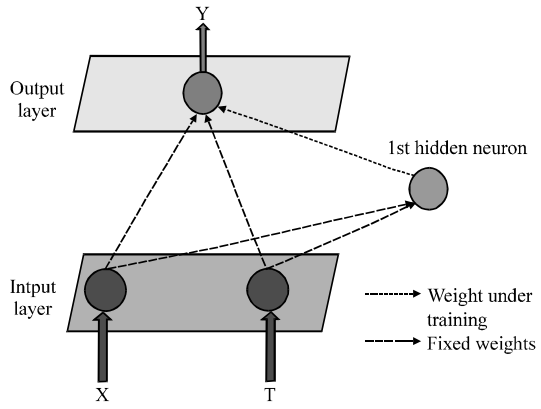

Fig. 8: Recruitment of first hidden neuron

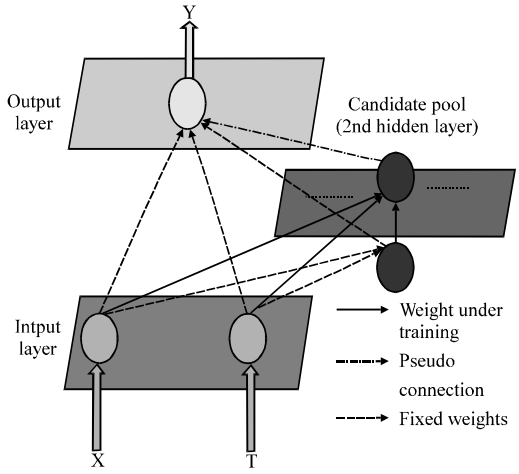

Fig. 9: Adding second hidden neuron

During the 2nd step of recruitment process only a single layer of weights are trained. The incoming weights of the winning neurons are frozen and the winner becomes the hidden neuron which is inserted into the active network in the 2nd step of recruitment. The new hidden neuron is then connected to the output neurons and the weights on connection become adjustable. All connections to the output neurons are trained as shown in Fig. 8. In other words, the weights connecting the input neurons and the output neurons are trained again using the quick propagation algorithm. On the other hand, the new weights connecting the hidden neurons and output neurons are trained for the first time. The second hidden neuron is then recruited using the same process as shown in Fig. 9 and 10. This unit receives input signals from both input neurons and previously recruited hidden neurons. All weights connecting input neurons and candidate hidden neurons are adjusted to recruit the second hidden neuron. The values of the weights are then frozen as soon as the hidden neuron is added to the

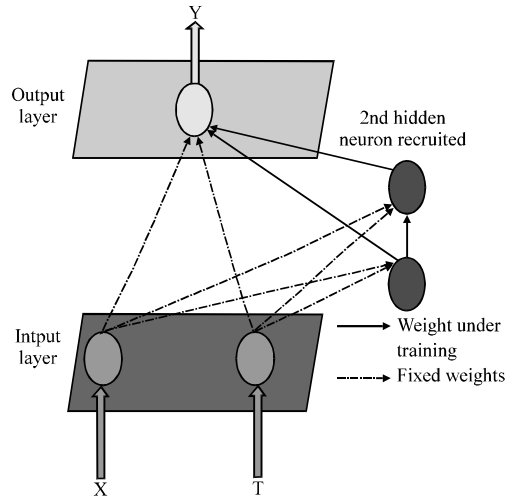

Fig. 10: Recuritmrnt of second hidden neuron

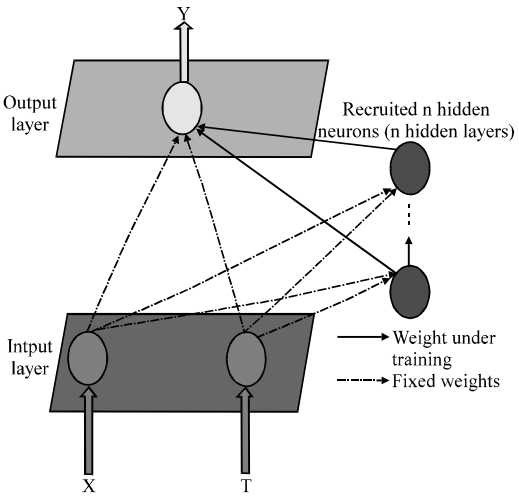

Fig. 11: Finalized CNN

active network. All the connections to the output neurons are then established and trained. The process of recruiting new neurons, training their weights from the input neurons and previously recruited hidden neurons, then freezing the weights and training all connections to the output neurons is continued until the error reaches the training error goal or the maximum number of epochs. The finalized CNN topology shown in Fig. 11 with n hidden neurons and n hidden layers.

New dynamic learning algorithm: The new dynamic learning algorithm used in CNN includes the following steps:

Step 1: Start with required input and output units.

Step 2: Train the net using Kohonen algorithm until the error reaches a minimum. If the error is negligible, stop the training process. Else, compute residual error (re) for each training pattern, the average residual error (are) and move to step 3:

$$re = ((y-ay1).^2)/2 \quad (41)$$

$$are = sum(re)/n \quad (42)$$

Where:

y = Target output for input vector
ay1 = Computed output for input vector

The following steps are done in Kohonen's algorithm:

Step 2a: Set learning rate, initialize weights.

Step 2b: While stopping condition is false, do steps 2c-2h.

Step 2c: For each input vector x, do steps 2d-2f.

Step 2d: For each j, compute squared Euclidean distance:

$$D_j = \sum_{i=1}^{n}(w_{ij} - x_i)^2; i = 1 \text{ to } n \text{ and } j = 1 \text{ to } m \quad (43)$$

Where:

w_{ij} = Weights between input unit and output unit
x_i = Input vector

Step 2e: Find index J when D_j is minimum.

Step 2f: For all units J with the specified neighborhood of J and for all i, update the weights:

$$w_{ij}(new) = (1-\alpha)\, w_{ij}(old) + \alpha x_i;\ i = 1 \text{ to } n \quad (44)$$

Step 2g: Update learning rate.

Step 2h: Test stopping condition.

Step 3: Then, recruit first hidden unit.

Step 4: A candidate unit z is connected to each input unit. Initialize weights from input units to z.

Step 5: Train these weights to maximize S. When no change in weights, they are frozen and the first hidden neuron is added. Maximum correlation is given by:

$$S = \sum_i (z(i) - z1)(re(i) - are) \text{ where } i = 1 \text{ to } n \quad (45)$$

Where:

z (i) = Output of candidate neuron
z1 = Mean of output candidate neuron
S = Magnitude of correlation

Step 6: Train all weights between input unit and hidden unit, hidden units and output units using quick propagation algorithm. If acceptable error is reached, the stop else move to step 7. Quick propagation weight updating for input to hidden layer is given:

dw1 (i, j) = a.*err (j).*x (i); (during first iteration)

$$dw1(i, j) = (s1(i)/(sold1(i) - s1(i))).*dwold1(i, j) \quad (46)$$

$$s1 = a.*err.*x \quad (47)$$

$$w1_{new} = w1_{old} + dw1 \quad (48)$$

Where:

dw1 = Weight change between input and hidden layer
dwold1 = Previous weight change
err = Error (difference between target and actual output)
x = Input at input layer
a = learning rate
w1 = Weight between input and hidden layer

Quick propagation weight updation for hidden to output layer:

dw2 (i, j) = a.*err(j).*z (i); (during first iteration) (49)

$$dw2(i, j) = (s1(i)/(sold1(i) - s1(i))).*dwold2(i, j) \quad (50)$$

$$s1 = a.*err.*x \quad (51)$$

$$w2_{new} = w2_{old} + dw2 \quad (52)$$

Where:

dw2 = Weight change between input and hidden layer
dwold2 = Previous weight change
err = Error (difference between target and actual output)
z = Output at hidden layer
a = Learning rate
w2 = Weight between hidden and output layer

Step 7: While stopping condition is false, do steps 8-10, i.e., add another hidden neuron.

Step 8: A candidate unit z is connected to each input unit and each previously added hidden unit.

Step 9: Train these weights to maximize S. When these weights stop changing, they are frozen.

Step 10: Train all the weights to the output units. If acceptable error or minimum number of unit has been searched, stop the process else proceed the training.

The advantage of using CNN is on the fly construction of architecture and minimal topology. CNN gives better accuracy when compared to other networks. The disadvantage of using CNN is requires more mathematical knowledge.

IDNN-Input Delay Neural Network: If INS position errors are examined, one can determine that they are accumulative, usually grow over time and follow a certain trend. It may not be possible to mimic and appropriately model this trend with an AI Based Model that relates the INS error to the corresponding INS output for a certain time instant. Therefore, a collection of particular number of past INS position sequence has to be presented to the model in order to capture the trend of the error pattern, thus establishing an accurate model of the INS errors. This can be realized by employing the tapped delay line approach by which the last k values of a signal are simultaneously presented at the input layer of the network (Noureldin *et al*., 2011).

The static neural network is transformed into a dynamic neural network by incorporating a memory and associator units at the input layer of input delay neural network. This Input Delayed Model is trained to learn sequential or time varying pattern of the INS samples. The memory holds past samples of INS values and the associator uses the memory to predict future events. The network processes time varying pattern of the INS samples applied as inputs. This Dynamic Model neural network considers a cluster inputs for the training process whereas the static neural networks produces the future outputs based on current samples position alone. In other words, we can say that the static neural networks output does not considers the past events. Because static neural networks are memory less topology that is effective for complex non linear static mapping. In fact, INS position error prediction is a procedure where previous errors have to be seriously considered. A fixed number of past events are selected and introduced to the input layer of the network.

Therefore, the network is involved with a static memory that is specifically expert knowledge-dependent. This has a beneficial effect on the prediction accuracy especially in the case of GPS absence. This model relies on input delay elements at the input layer so that the output INS position error is modeled based on the present and past samples of the corresponding INS position. The impact of different delay elements present at the input layer improves the overall positioning accuracy (Noureldin *et al*., 2011).

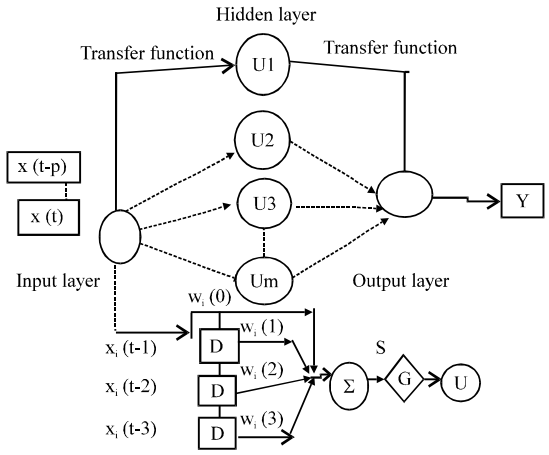

Fig. 12: Input delayed neural network architecture

Another way of dealing with temporal patterns is the use of an internal time delay operator within the MLPNN network. This leads to the time delay neural network also referred to as the Input-Delay Neural Network (IDNN). In this case, the static MLPNN is transformed into a dynamic network by replacing each static synaptic weight with a finite weight factor. Thus, the number of the embedded time delays provides the network with a short-term memory. The number of neurons associated with the input layer is equal to the number of input variables therefore, the IDNN integrates temporal context information implicitly and it thus recognizes temporal patterns that have arbitrary time intervals or arbitrary lengths of temporal effects. Thus, the IDNN is suitable for situations where temporal patterns should be considered. This has a beneficial effect on the prediction accuracy which is the major objective of this study. Furthermore, the IDNN can be trained even with the standard back-propagation algorithm (Bishop, 1995). The general architecture of an input-delayed neural network in addition to zooming on the internal structure of a single neuron is shown in Fig. 12. The architecture consists of a tapped delay line that involves the L most recent inputs. In this example, researchers show three delay elements represented by the operator D. For a case of L delay elements and an input variable x (t), the network processes x (t), x (t-1), x (t-2), ... and x (t-L) where L is known as the tapped delay line memory length (Noureldin *et al*., 2011; Bishop, 1995). Therefore, the input signal S_i (t) to the neuron U_i (Fig. 12) is given as:

$$S_i(t) = \sum_{k=0}^{p} w_i(k)\, x(t-L) + b_i \qquad (53)$$

Where:

w_i (k) = The synaptic weight for neuron i
b_i = Bias

Then, the output of this neuron (U_i) is obtained by processing S_i (t) by the non-linear activation function G (.), chosen as a sigmoid activation function of neuron i:

$$U_i = G(\sum_{k=0}^{p} w_i(k)x(t-k) + b_i) \quad (54)$$

$$G(S_i(t)) = \frac{1}{1+e^{-S_i(t)}} \quad (55)$$

The output of the IDNN, assuming that it has one output neuron j, a single hidden layer with m hidden neurons and one input variable as shown in Fig. 12 is given by:

$$y_j(t) = F(\sum_{i=1}^{m} w_{ji}U_i + \alpha_j) \quad (56)$$

where, F(.) is the transfer activation function of the output neuron j (which can be chosen to be a sigmoid or a linear function), α_j is its bias and w_{ji} is the weight between the neurons of the hidden layer and the neuron of the output layer. During the update procedure, researchers use a second-order back-propagation variation; namely the Levenberg-Marquardt Back-Propagation (LMBP). The network training process is performed by providing input-output data to the network which targets minimizing the error function by optimizing the network weights. LMBP uses the second derivative of the error matrix (E) to update the weights of the network in a recursive fashion (Haykin, 1994; Bishop, 1995). The advantage of IDNN is that due to its dynamic and delayed values of input we can predict the future values which will be useful when there is an absence of GPS signal.

New dynamic back-propagation learning algorithm: The training criteria of the network involves new dynamic back propagation learning algorithm based on input delayed model of IDNN (Noureldin *et al*., 2011; Sivanandam *et al*., 2002):

- The weight is initialized as small random values
- The input signal x_i is given to each input unit and the input unit transmits the signal to all hidden layers
- Each hidden unit sums its weighted input signals as:

$$Z_{-inj} = \sum_{L=1}^{d} W_i(k)x(t-L) + \alpha_i \quad (57)$$

and the activation function is applied for training the input signal:

$$Z_j = f(Z_{-inj}) \quad (58)$$

- The hidden layer output is given as input to output unit. Each output unit sums it weighted input signals as:

$$y-ink = F(\sum_{i=1}^{N} W_{jk}H_i + \beta_j) \quad (59)$$

and the activation function to calculate the output signals:

$$y_k = f(y_{-ink}) \quad (60)$$

- Comparator function of the trained output neuron j and input neuron i is given as:

$$C_j = C(F(\sum_{i=1}^{N} W_{jk}H_i + \beta_j) \quad (61)$$

- Each output unit receives a GPS target pattern. According to the received GPS signal corresponding error information is calculated as:

$$\delta_k = (t_k - y_k) \times f(y_{-ink}) \quad (62)$$

- Each hidden unit receives the feedback from output unit and sums its delta inputs from the units in the layer above:

$$\delta_{-inj} = F(\sum_{k=1}^{m} \delta_j \times w_{jk}) \quad (63)$$

The error information is calculated as:

$$\delta_j = \delta_{-inj} \times f(Z_{-inj}) \quad (64)$$

- Each output unit updates its bias and weight
- The weight correction term is given by:

$$\Delta w_{jk} = \alpha\delta_k Z_j \quad (65)$$

And the bias correction term is given by:

$$\Delta\beta_j = \alpha\delta_k \quad (66)$$

Therefore:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} \quad (67)$$

$$\beta_j(new) = \beta_j(old) + \Delta\beta_j \quad (68)$$

- Each hidden unit updates its bias and weights The weight correction term:

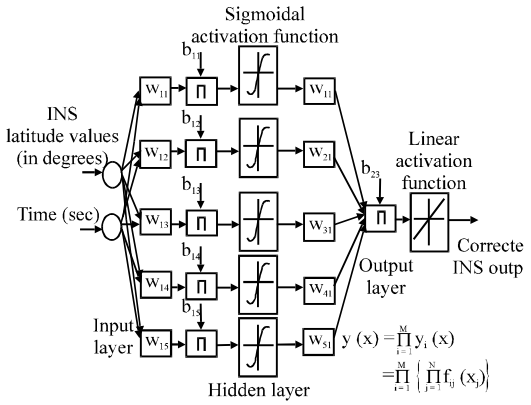

Fig. 13: Extended multiplicative neural network architecture

$$\Delta w_i(k) = \alpha \delta_i X_i \tag{69}$$

The bias correction term:

$$\Delta \alpha_i = \alpha \delta_k \tag{70}$$

Therefore:

$$w_{ik}(new) = w_{ik}(old) + \Delta w_{ik} \tag{71}$$

$$\alpha_i(new) = \alpha_i(old) + \Delta \alpha_i \tag{72}$$

- After the errors are minimized, the training process was stopped

HONN-Higher Order Neural Networks

MNN-Multiplicative Neural Network: The Multiplicative Neural Network (MNN) can be considered as a special case of the Multi Layer Perceptron (MLP) and therefore, the Back Propagation (BP) algorithm forms the basis for deriving the MNN learning rule. It is one of the higher order neural networks where the replacement of summation at hidden node and output nodeby multiplication results in more powerful mapping (Burse *et al*., 2008; Schmitt, 2002; Heywood and Noakes, 1995; Giles and Maxwell, 1987). The neuron contains units which multiply their inputs instead of summing them and thus allow inputs to interact nonlinearly (Burse *et al*., 2009; Iyoda *et al*., 2002).

Multiplicative node functions allow direct computing of polynomial inputs and approximate higher order functions with fewer nodes (Chiang *et al*., 2008). The architecture of MNN is shown in Fig. 13. The output of the multiplicative neural network is given by:

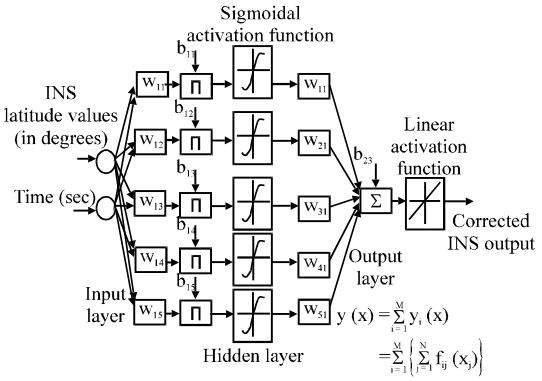

Fig. 14: Sigma Pi-neural network architecture

$$y(x) = \prod_{i=1}^{M} y_i(x) = \prod_{i=1}^{M} \left\{ \prod_{j=1}^{N} f_{ij}(x_j) \right\} \tag{73}$$

$$y_i(x) = \prod_{j=1}^{N} f_{ij}(x_j) \tag{74}$$

Where:

$y_i(x)$ = Output of the hidden layer where i = 1, 2, ..., M
$y(x)$ = Output of the neural network
f_{ij} = Bipolar sigmoidal activation function
x_j = N dimensional input where j = 1, 2, ..., N

The multiplicative neural network gives superior error performance and performance index but it takes more time and more number of epochs to carry out the training process when compared to the sigma Pi-NN.

SPN-Sigma-Pi Neural network: The Sigma-Pi neural network is one of the higher order neural networks and Back Propagation (BP) algorithm forms the basis of deriving the Sigma-Pi neural network learning rule. In this architecture, the replacement of summation at the hidden node by multiplication and output node by summation, results in more powerful mapping (Yun-Wen and Chiang, 2008a, b).

They allow neural networks to learn multiplicative interactions of arbitrary degree. The general and extended architecture of Sigma-Pi ANN is shown in Fig. 14. The output of the Sigma-Pi neural network is given by:

$$y(x) = \sum_{i=1}^{M} y_i(x) = \sum_{i=1}^{M} \left\{ \prod_{j=1}^{N} f_{ij}(x_j) \right\} \tag{75}$$

$$y_i(x) = \prod_{j=1}^{N} f_{ij}(x_j) \tag{76}$$

Where:

$y_i(x)$ = Output of the hidden layer where i = 1, 2, ..., M
$y(x)$ = Output of neural network

f_{ij} = Bipolar sigmoidal activation function
x_j = N dimensional input where j = 1, 2, ..., N

The Sigma-Pi network includes the following advantages: superior error performance in nonlinear input-output mapping, less execution time and improved performance index.

ISPN-Improved Sigma-Pi Neural network: Sigma-Pi algorithm implemented along with Dynamic Weight Pruning (DWP) algorithm is referred to as Improved Sigma-Pi algorithm. The network is initially converged using Sigma-Pi algorithm.

Weight pruning algorithms typically require network convergence before pruning is applied. Two tests involved in DWP algorithm are; Stability Criteria test, Weight Significance test. In Stability Criteria test, the point at which weights or neurons are representative of their target state must be identified. In the Weight Significance test, significance of weights local to the target neuron is estimated. These two tests form the basis of our dynamic weight pruning process. Pruning following network convergence only requires a weight significance test.

Dynamic weight pruning algorithm used in SPN includes the following steps

Step 1: Initialize Sigma-Pi training until the network converges.

Step 1a: Initialize weight to small random variables.

Step 1b: While stopping condition is false, do steps 3-10.

Step 1c: For each input signal, do steps 4-9.

Step1d: Each input receives the input signal x_i and transmits this signal to all of the hidden units.

Step 1e: Each hidden unit multiplies its weighted input signals as:

$$y_{jn}^{n} = f\left(\prod_{jn-1}^{Nn-1}\left(w_{jnjn-1}y_{jn-1}^{n-1} + b_{jnjn-1}\right)\right) \quad (77)$$

Step 1f: Each output unit multiplies its weighted input signals as:

$$y_{k} = f\left(\prod_{jn=1}^{Nn-1}\left(w_{kjn}y_{jn}^{n} + b_{kjn}\right)\right) \quad (78)$$

Step 1g: Mean square error function is computed as:

$$E_{MSE} = \frac{1}{2PK}\sum_{k=1}^{K}\sum_{p=1}^{P}\left(y_{dk}^{p} - y_{k}^{p}\right)^{2} \quad (79)$$

Step 1h: Weights and biases are updated as:

$$w_{i}^{new} = w_{i}^{old} + \Delta w_{i} \quad (80)$$

$$b_{i}^{new} = b_{i}^{old} + \Delta b_{i} \quad (81)$$

Step 1i: Weights and Bias correction terms are given by:

$$\Delta w_{kjn} = -\eta\frac{\partial E_{MSE}}{\partial w_{kjn}} \quad (82)$$

$$= \eta\delta_{k}\frac{\left[\prod_{jn=1}^{Nn-1}(w_{kjn}y_{jn}^{n} + b_{kjn})\right]}{(w_{kjn}y_{jn}^{n} + b_{kjn})} \times y_{jn}^{n} \quad (83)$$

$$\delta_{k} = \frac{1}{PK}\left[\sum_{k=1}^{K}\sum_{p=1}^{P}\left(y_{dk}^{p} - y_{k}^{p}\right)\left[(1/2)\left(1 + y_{k}^{p}\right)\left(1 - y_{k}^{p}\right)\right]\right] \quad (84)$$

$$\Delta b_{kjn} = \eta\delta_{k}\frac{\left[\prod_{jn=1}^{Nn}\left(w_{kjn}y_{jn}^{n} + b_{kjn}\right)\right]}{\left(w_{kjn}y_{jn}^{n} + b_{kjn}\right)} \quad (85)$$

$$= \frac{\Delta w_{kjn}}{y_{jn}^{n}} \quad (86)$$

Step 1j: Test the stopping condition, i.e., number of epochs.

Step 2: Initialize count of DWP criteria count to 1.

Step 3: Calculate DWP count.

Step 4: Check whether the weight values are set to zero.

Step 4a: Perform Stability Criteria test, i.e., the point at which the weights or neuron are representative of their target state is identified.

Step 4b: If the test satisfies, decrement DWP Criteria Count and re-introduce the weight values:

$$\text{DWP criteria count} = \text{DWP criteria count} - 1 \quad (87)$$

$$\text{wnew (i)} = \text{w (i)} \quad (88)$$

Step 4c: If not, increment DWP criteria count and replace the weight values:

$$\text{DWP criteria count} = \text{DWP criteria count} + 1 \quad (89)$$

$$wnew(i) = w(i)+0.5 \quad (90)$$

Step 5: If the weight values are rather than zero, perform weight sudation.

Step 5a: Perform Stability Criteria test.

Step 5b: It the test satisfies, then perform Weight Significant test, i.e., the significance of weights to the target neuron is identified.

Step 5c: If the Weight Significance test satisfies, check whether DWP criteria count is greater than DWP count.

Step 5d: Assign the weight values to DWP and increment DWP criteria count:

$$DWP\ shortlist = w(i) \quad (91)$$

$$DWP\ criteria\ count = DWP\ criteria\ count+1 \quad (92)$$

Step 6: Consider the next weight and repeat steps 4-5d.

Step 7: Check whether previous and current weight values are same. Then, set the redundant weight values as zero:

$$w_{prev} = w_{current} = 0 \quad (93)$$

Step 8: Consider the next Neuron and repeat steps 4-7.

Step 9: Continue Sigma-Pi algorithm and repeat steps 1a-1j.

MATERIALS AND METHODS

Along each of the East, North and vertical directions, the INS position and the time (t) are the inputs to one of the NN modules while the error in the corresponding INS position is the module output. During the availability of the GPS signal, the NN module operates in the training mode. In order to train the network, the INS position error provided at the output of this NN module should be compared to a certain target or a desired response. In this case, the INS position error is the difference between the INS original position and the corresponding GPS position. The difference between NN module output and the true position error is the estimation error of the NN module. In order to minimize this error, the NN module is trained to adjust the NN parameters that are continuously updated according to the least square criterion until reaching certain the minimal Mean Square Error (MSE). The training procedure continues and is repeated for all GPS/INS data windows until a GPS absence is detected. When the satellite signal is blocked (during GPS absence), the system is switched into the prediction mode where the NN module is used to process the INS position P_{INS} at the input and predict the corresponding position error using the latest NN parameters obtained before losing the satellite signals.

The estimated error of current INS position for GPS presence is seriously considered for the prediction mode training process in order to predict the corrected INS position during GPS absence in prediction mode. The prediction mode training procedure of the neural network starts after finding the estimation error of current INS position just before lose of satellite signal. The pattern of INS and GPS position components are used to train the neural network module to reflect the latest vehicle dynamics and the INS error trend. The neural network module is trained until a certain number of MSE is reached. To provide a complete navigation solution for a aircraft vehicle, the longitude, latitude and altitude components are considered and applied to the NN module to estimate the MSE.

RESULTS AND DISCUSSION

The real time trajectory path shown in Fig. 15 was taken into consideration and the training algorithms for different ANNs were implemented. This path shows Bangalore to Mumbai air route in India and sample values for GPS and INS are taken for the commercial aircraft. The GPS values are directly taken from Google map software database for this air route and INS values are taken from standard database of Inertial Measurement Unit (IMU) for this route. The INS sample values are taken as the input vector and the GPS sample values are taken as the target vector.

Training was performed for latitude, longitude and altitude components of the aircraft vehicle. The actual output, i.e., corrected INS value for all the three components were found using different artificial intelligence based neural networks. The sample values obtained from inertial measurement unit varies in terms of seconds of latitude and longitude component. So, the continuous change in values of latitude and longitude samples are seriously considered for the training process. The performance of the various artificial intelligent neural networks for GPS/INS integration module was examined in training mode (GPS presence) and prediction mode (GPS absence). Over the whole trajectory of normal

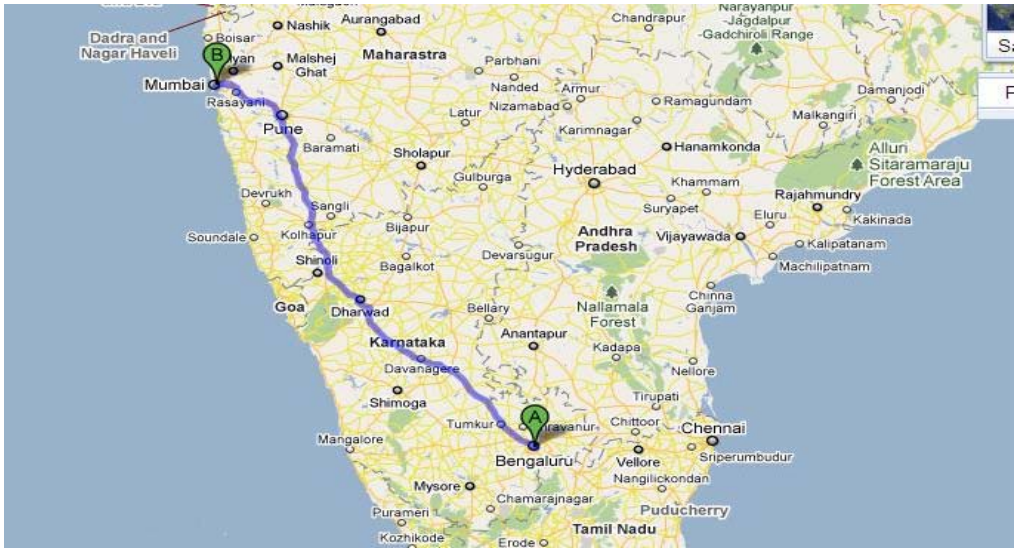

Fig. 15: Trajectory path

mode test, no natural GPS absences were detected and thus the corrected INS position of each sample was predicted and simulation result was plotted by taking air travel time in x axis and target value, original INS value and corrected INS value in y axis. During training of various neural networks, network over-fitting problem was encountered. Network over-fitting is a classical machine learning problem. It usually occurs when the network captures the internal local patterns of the training data set rather than recognizing the global pattern of the data set. It is important to realize that the specification of the training samples is a critical factor in producing a neural network output which is capable of making correct response. Two procedures have been evaluated to overcome the problem of over-fitting namely, early stopping and regularization. In this ANN Model, early stopping procedure was used to solve the network over-fitting problem.

The aim of early stopping is to mimic the prediction of future individuals from the population (Nouerldin *et al*., 2009; Haykin, 1994). The second method utilized to avoid the over-fitting problem and to optimize the NN Model is the regularization technique. This is known to be a very desirable procedure when the scaled conjugate gradient descent method is adopted for training (Noureldin *et al*., 2011; Bishop, 1995). In this research, early stopping criterion procedure is used to accurately model the INS position error. During the training stage, the module performs the function of understanding the input/output mapping. The early stopping criterion was chosen due to its suitability for real time implementation as the computational time is a substantial limitation. On the other hand, artificial GPS absences were intentionally introduced to real time trajectory in order to test its ability to accurately predict the INS errors and provide reliable INS position information.

Table 1: Architectural complexity of RBF, BPN, FCPN and Full CPN neural networks

RBF	BPN	FCPN	Full CPN
Simple in Architecture (3 layer with feed forward approach)	Complex in nature when compared to RBF (Though it is an 3 layer architecture it's having both feed forward and feedback propagation)	Simple in architecture (3 layer with counter propagation approach)	Complex in nature compared with FCPN (Because of its two way counter propagation)

Table 2: Architectural complexity of ART-CPN, CNN, HONN and IDNN neural networks

ART-CPN	CNN	HONN	IDNN
Complex because of the resonance and counter propagation	Though it is dynamic, selection and recruitment of hidden nodes become complex	Complex in nature because of the summation and multiplicative process in the node	Complex due to delayed input and the need of memory and associator circuits

A time of absence of GPS signals at subsequent intervals were selected at different locations on the trajectory path based on the consideration of GPS jamming and multipath error.

The architectural complexity of neural networks is shown in Table 1 and 2. The algorithmic complexity of neural networks is shown in Table 3 and 4. Feedback provision of neural networks are shown in Table 5. Table 6 shows the activation function which is used in between input to hidden layer of the various neural networks.

The analysis was performed for the following parameters: Root Mean Square Error (RMSE), Performance Index (PI), number of epochs and execution time for different neural networks.

The root mean square error and performance index are calculated from the target output and computed output according to the equation given:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}(y_i - ay_i))^2}{m}}$$

$$PI = \sqrt{\frac{\sum_{i=1}^{m}(y_i - ay_i))^2}{\sum_{i=1}^{m}|y_i|}}$$

Where:

y_i = The target output
ay_i = The computed output
m = The number of data patterns
$|y_i|$ = Modulus of target output

Table 3: Algorithmic complexity of RBF, BPN, FCPN and Full CPN neural networks

RBF	BPN	FCPN	FULL CPN
Simple algorithm like LMS algorithm	Complex due to backing and comparing of error with previous and present values in the output layer	It is a combination of supervised learning and unsupervised learning	It has two way training process because of counting propagation nature

Table 4: Algorithmic complexity of ART-CPN, CNN, HONN and IDNN neural networks

ART-CPN	CNN	HONN	IDNN
It uses Grossberg leaning for resonance adaptive training and kohonen algorithm for CPN	Quick propagation algorithm and upstart algorithm are used to dynamically train the CNN	Weight pruning algorithm is used to remove the zeros and redundancy of weights during updation process	New dynamic leaning algorithm is used to make the dynamic nature of inputs

The time taken by the neural network to complete the training process is referred to as execution time. The simulated results for latitude, longitude and altitude components for the various artificial intelligent neural networks are shown in Fig. 16-18, respectively.

During the training of all the intelligent networks INS position error value was calculated, so that it can be stored and used during the prediction mode. The mean square error value was calculated for each and every epoch in all networks. The performance measure graph was plotted with mean square error value in y axis and number of epochs in x axis for RBFNN, BPN, FCPN, Full CPN, ARTCPN, CNN, MNN, SPN and IDNN, respectively. Figure 19 shows latitude MSE, longitude MSE and

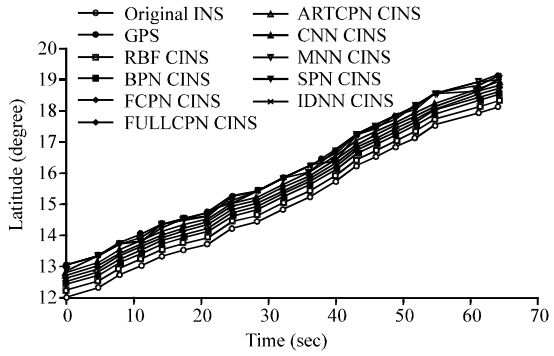

Fig. 16: Latitude output

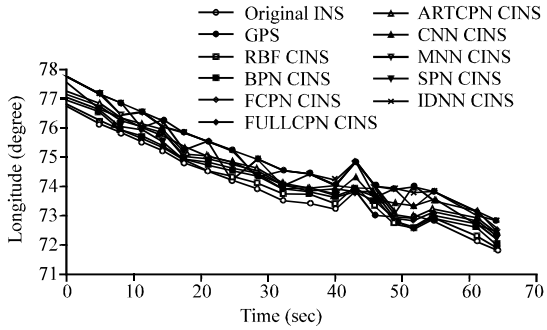

Fig. 17: Longitude output

Table 5: Feedback provision of neural networks

RBF	BPN	FCPN	Full CPN	ART-CPN	HONN	IDNN
No feedback is required	Feedback from output layer to hidden layer is needed to get error accuracy	No feedback is required for learning	Feedback is required from input layer to cluster layer and from cluster layer to input layer	Feedback is not required	Feedback provision is not required	No feedback is required

Table 6: Type of activation function used for neural networks

RBF	BPN	FCPN	Full CPN	CNN	HONN	IDNN
Guassian function is used between input layer to hidden layer	Bipolar sigmoidal function is used	Guassian function is used like RBF NN	Guassian function is used like RBF NN	Bipolar sigmoidal function is used	Bipolar sigmoidal function is used	Bipolar sigmoidal function is used

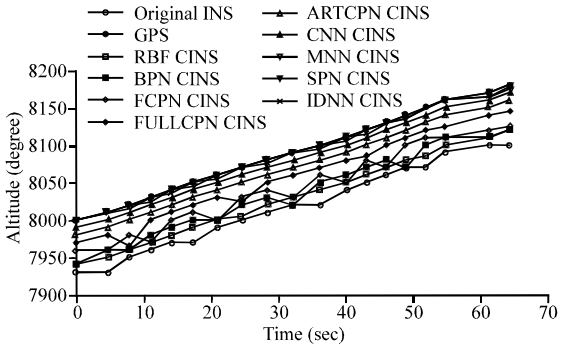

Fig. 18: Altitude output

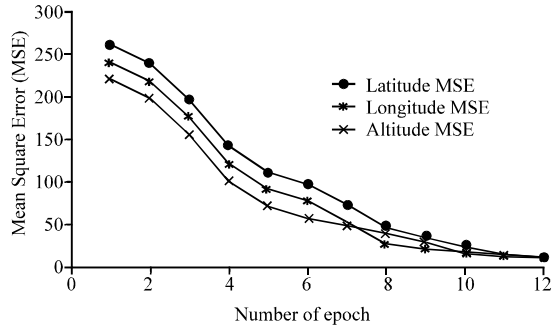

Fig. 19: RBF error performance

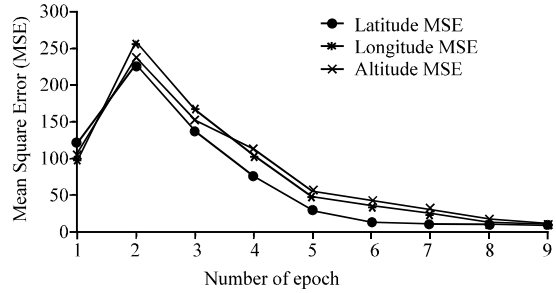

Fig. 20: BPN error performance

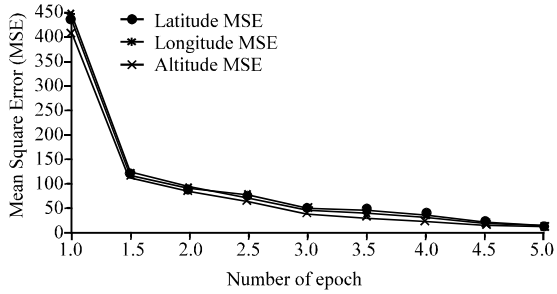

Fig. 21: FCPN error performance

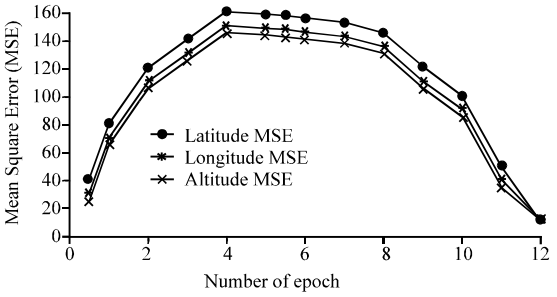

Fig. 22: FULLCPN error performance

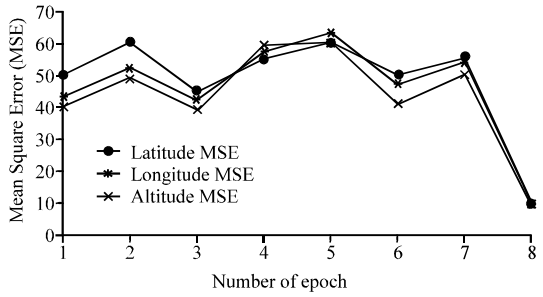

Fig. 23: ARTCPN error performance

altitude MSE of RBFNN. From mean square error graph of RBFNN it was found that the MSE value decreases monotonically with distance from a central point and they are radially symmetric. But as the central point weight computed based on Kohanen feature maps the error was decreased gradually. Figure 20 shows latitude MSE, longitude MSE and altitude MSE of BPN. From mean square error graph of BPN it was found that the MSE value increases initially and goes to a high peak due to error propagation. But as the weight factors and bias are trained based on gradient descent learning rule the error decreases and reaches the minimum value. From Fig. 21, it was inferred that the MSE value of FCPN goes to an high value at the beginning of the training but as a result of tuning the synaptic weight between the input layer and cluster layer using Kohonen weight updating method during the first phase and updating the weight between cluster layer and output layer during the second phase, the MSE value was reduced. Figure 22 shows latitude MSE, longitude MSE and altitude MSE of Full CPN. From mean square error graph of Full CPN it was found that the MSE value first increases gradually and then reaches to a maximum level. But as the selection of winner node based on enhanced weight updating rule the error starts decreasing gradually and reaches to a minimum level. Figure 23 shows latitude MSE, longitude MSE and altitude MSE of ART- CPN. From mean square error graph of ART-CPN it was found that the MSE value dynamically varies from low to high and high to low and vice versa.

But as the weights of the winning node was computed based on Kohonen and Gross berg learning

Table 7: Comparison performance of artificial neural networks

Criteria	RBF	BPN	FCPN	FULL CPN	ART CPN	CNN	MNN	SPN	IDNN
Latitude MSE	3.6812	3.1702	2.3323	1.9780	1.7959	1.2450	0.8741	0.6552	0.6452
Longitude MSE	3.6501	2.2034	2.0453	1.6779	1.3641	0.9085	0.7041	0.5212	0.5206
Altitude MSE	2.6522	2.2996	2.0734	1.7464	1.4647	0.90454	0.9290	0.6612	0.6138
Latitude RMSE	1.9186	2.0421	1.5272	1.4064	1.3401	1.1158	0.9349	0.8094	0.8032
Longitude RMSE	1.9105	1.4844	1.4301	1.2954	1.1680	0.8253	0.8391	0.7219	0.7215
Altitude RMSE	1.6285	1.5164	1.4399	1.3215	1.2103	0.9501	0.9638	0.8131	0.7834
Latitude PI	0.2478	0.2018	0.1128	0.0957	0.0869	0.0602	0.0598	0.0189	0.0145
Longitude PI	0.0378	0.0276	0.0256	0.0210	0.0171	0.0120	0.0117	0.0158	0.0139
Altitude PI	0.0967	0.0734	0.0662	0.0557	0.0467	0.0288	0.0245	0.0147	0.0124
No of epoch	14	9	8	17	8	7	7	5	7
Latitude ET	11.7910	7.3290	8.6745	14.7456	10.4589	12.5634	18.0891	12.5101	7.8290
Longitude ET	10.2390	9.5801	7.9784	13.6487	10.7456	11.6745	16.7801	11.6302	10.5801
Altitude ET	10.7180	9.7020	7.4567	14.0747	9.8765	11.7947	17.2871	11.8010	10.5670

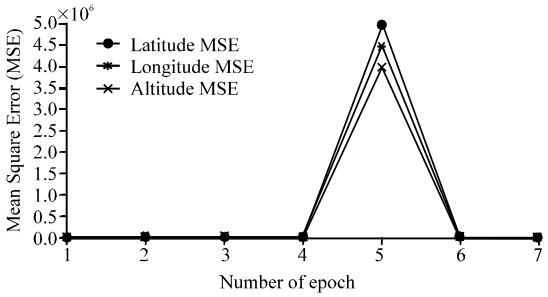

Fig. 24: CNN error performance

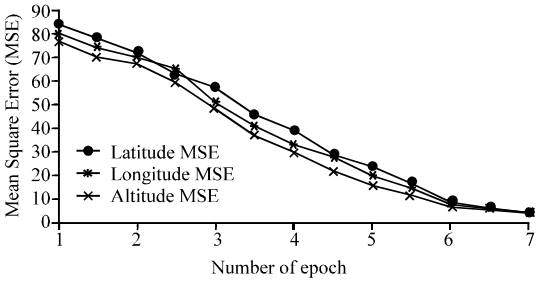

Fig. 25: MNN error performance

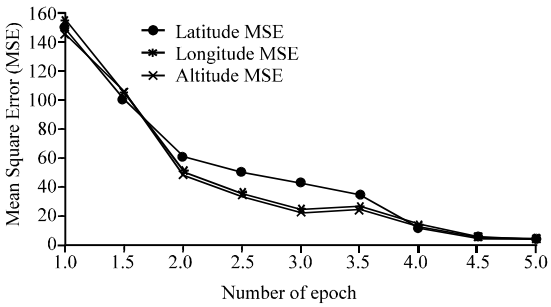

Fig. 26: SPN error performance

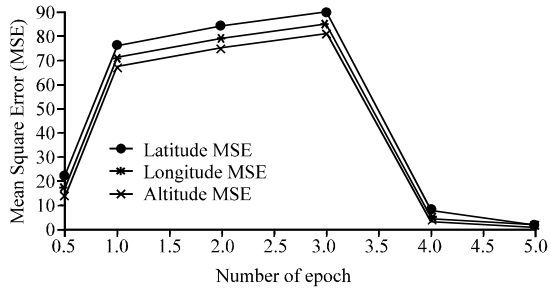

Fig. 27: IDNN error performance

rule the error has been reduced at the end of training. Figure 24 shows latitude MSE, longitude MSE and altitude MSE of CNN. In CNN, the mean square error first suddenly goes to a high peak value due to all candidate neurons receive the same residual error for each training pattern fed back from the output neurons. But as the weights on connecting the input neurons and candidate neurons are adjusted the error was decreases suddenly to very minimum level.

Figure 25 and 26 shows latitude MSE, longitude MSE and altitude MSE of MNN and SPN, respectively. In this case, the mean square error value of MNN and SPN is monotonically decreases from high to low and finally reaches to very minimum level. This is because of replacement of summation at hidden node and output node by multiplication in MNN and replacement of summation at hidden node by multiplication and output node by summation. As a result, the error was decreased monotonically and finally reaches as minimum level.

Figure 27 shows latitude MSE, longitude MSE and altitude MSE of IDNN. From mean square error of IDNN it was found that the MSE value increases initially and goes to a maximum level and once again increases slightly from the previous level due to back propagation of errors. But as the weight factors and bias values of time delayed elements are trained based on Least Mean Square (LMS) rule the error decreases gradually and then reduces to a very minimum level. The result obtained during the training process of different Neural Networks is shown in Table 1. The architectural complexity of Neural networks is shown in Table 7.

CONCLUSION

From the resultant table and its analysis, it was found that IDNN and SPN gives more accurately corrected INS value when compared to RBFNN,BPN, FCPN, FULL CPN, ART-CPN, CNN and MNN. It is also found that latitude RMSE, longitude RMSE and altitude RMSE of IDNN, SPN and MNN are less when compared to RMSE values of RBF, BPN, FCPN, Full CPN and ART-CPN and CNN so IDNN, SPN and MNN are more suitable in terms of error accuracy. The RMSE values of RBF, BPN are the highest when compared to the other, so RBF and BPN cannot be used to give accurate results. Similarly latitude PI, longitude PI and altitude PI of IDNN, SPN and MNN are lesser than the PI values of RBF, BPN, FCPN, Full CPN ART-CPN and CNN, so IDNN, SPN and MNN are also more suitable in terms of improved performance. Among the intelligent networks analyzed, RBF, BPN has the highest PI value, so RBF, BPN cannot give optimum performance.

FCPN, Full CPN, ART-CPN and CNN give better performance and accuracy when compared to RBF, BPN. Full CPN is more accurate when compared to FCPN. Full CPN also has lesser PI values when compared to FCPN, so it shows better performance than FCPN. Although, Full CPN has less RMSE values and PI values than that of FCPN but it takes more number of epochs to obtain accurate position value.CNN takes least number of epochs when compared to BPN, FCPN, Full CPN and ART-CPN.

But it takes high performance index and high execution time for the training process when compared to the IDNN, MPN and SPN. Moreover, the error accuracy of CNN is somewhat lower than IDNN, MNN and SPN. RBFNN, BPN takes lesser number of epochs when compared to Full CPN but it consumes more execution time and less performance in terms of error accuracy and performance index.

From the overall statistical analysis, the training of IDNN, SPN and MNN models shows superior performance in mean square error, root mean square error, performance index, execution time and least number of epochs when compared to all other Intelligent Neural Networks.

So, IDNN, SPN and MNN are optimized neural network for GPS/INS Integration in terms of error accuracy and improved performance even during the signal blockages in GPS. So, IDNN, SPN and MNN networks are proposed to predict the accurate position of the moving vehicle in training mode as well as prediction mode.

REFERENCES

Bishop, C.M., 1995. Neuarl Networks for Pattern Recognition. Clarendon Press, NewYork, ISBN: 9780198538646, Pages: 482.

Burse, K., R.N. Yadav and S.C. Shrivastava, 2008. Complex channel equalization using polynomial neuron model. Proceedings of the IEEE 3rd International Symposium on Information Technology, August 26-29, 2008, Kuala Lumpur, Malaysia, pp: 771-775.

Burse, K., R.N. Yadav, S.C. Shrivastava and V.P.S. Kirar, 2009. A compact pi network for reducing bit error rate in dispersive FIR channel noise model. Proceedings of the World Congress on Science, Engineering and Technology, February 25-27, 2009, Penang, Malaysia, pp: 235-238.

Chiang, K.W., A. Noureldin and N. El-Sheimy, 2008. Constructive neural-networks-based MEMS/GPS integration scheme. IEEE Trans. Aerospace Electronic Syst., 44: 582-594.

Ching-Piao, T. and T.L. Lee, 1999. Back-propagation neural network in tidal-level forecasting. J. Waterway, Port, Coastal, Ocean Eng., 125: 4-8.

El-Sheimy, N., K.W. Chiang and A. Noureldin, 2008. Developing a low cost MEMS IMU/GPS integration scheme using constructive neural networks. IEEE Trans. Aerosp. Electron. Syst., 44: 582-594.

Giles, C.L. and T. Maxwell, 1987. Learning, invariance and generalization in high-order neural networks. Applied Opt., 26: 4972-4978.

Grossberg, S., 1987. Competitive learning: From interactive activation to adaptive resonance. J. Cognitive Sci., 11: 23-63.

Hagan, M.T., H.B. Demuth and M.H. Beale, 1996. Neural Network Design. 1st Edn., PWS Publishing Co., Boston, MA, USA., ISBN: 0-53494332-2.

Haykin, S., 1994. Neural Networks: A Comprehensive Foundation. 1st Edn., Macmillan Publishing Co., New York, USA.

Heywood, M. and P. Noakes, 1995. A framework for improved training of sigma-pi networks. Inst. Electr. Electron. Eng. Trans. Neural Networks, 6: 893-903.

Hosteller, L. and R. Andreas, 1983. Nonlinear kalman filtering techniques for terrin-aided navigation. Inst. Electr. Electron. Eng. Trans. Autom. Control, 28: 315-323.

Iyoda, E.M., K. Hirota and F.J. von Zuben, 2002. Sigma-pi cascade extended hybrid neural network. J. Adv. Comput. Intell. Intell. Inform., 6: 126-134.

Kumar, S., 2004. Neural Networks: A Classroom Approach. 1st Edn., Tata Mc-Graw Hill Publications, USA., ISBN-10: 0-07-048292-6, pp: 169.

Nouerldin, A., T. Karamat, M. Eberts and A. El-Shafie, 2009. Performance enhancement of MEMS based INS/GPS integration for low cost navigation applications. IEEE Trans. Veh. Technol., 58: 1077-1096.

Noureldin, A., A. El-Shafie and M. Bayoumi, 2011. GPS/INS integration utilizing dynamic neural networks for vehicular navigation. Inform. Fusion, 12: 48-57.

Noureldin, A., R. Sharaf, A. Osman and N. EI-Sheimy, 2004. INS/GPS data fusion technique utilizing radial basis functions neural networks. Proceedings of the IEEE Position Location and Navigation Symposium, April 26-29, 2004, Monterey, California, pp: 527-537.

Schmitt, M., 2002. On the complexity of computing and learning with multiplicative neural networks. Neural Comput., 14: 241-301.

Sharaf, R. and A. Noureldin, 2007. Sensor integration for satellite-based vehicular navigation using neural networks. IEEE Trans. Neural Networks, 18: 589-594.

Sharaf, R., A. Noureldin, A. Osman and N. EI-Sheimy, 2005. Online INS/GPS integration with a radial basis function neural network. IEEE Sys. Mag., 20: 8-14.

Sivanandam, S.N., S. Sumathi and S.N. Deepa, 2002. Introduction to Neural Networks Using Matlab 6.0. Tata McGraw-Hill Publication, New Delhi..

Yen-Chang, C., 2001. A counter propagation fuzzy-neural network modeling approach to real time stream flow prediction. J. Hydrol., 245: 153-164.

Yun-Wen, H. and K.W. Chiang, 2008a. An intelligent and autonomous MEMS IMU/GPS integration scheme for low cost land navigation applications. GPS Solutions, 12: 135-146.

Yun-Wen, H. and K.W. Chiang, 2008b. An intelligent navigator for seamless INS/GPS integrated land vehicle navigation application. Applied Soft Comp., 8: 723-733.