Asian Journal of Probability and Statistics

17(2): 61-85, 2022; Article no.AJPAS.86132
ISSN: 2582-0230

Initialization and Estimation of Weights and Bias using Bayesian Technique

Oryiema Robert [a*], David Angwenyi [a], Nyogesa Achiles [a] and Jacob On'gala [a]

[a] *Masinde Muliro University of Science and Technology, P.O.Box 190-50100, Kakamega, Kenya.*

Authors' contributions

This work was carried out in collaboration among all authors. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/AJPAS/2022/v17i230420

Open Peer Review History:
This journal follows the Advanced Open Peer Review policy. Identity of the Reviewers, Editor(s) and additional Reviewers, peer review comments, different versions of the manuscript, comments of the editors, etc are available here: https://www.sdiarticle5.com/review-history/86132

Original Research Article

Received: 10 February 2022
Accepted: 15 April 2022
Published: 30 April 2022

Abstract

Researches on artificial neural network models have shown that the method used to initialize and estimate weights and bias always determines the rate at which the network will converge and how efficient the model will perform. Although there are several methods that can be used to initialize and estimate network weights and bias, Bayesian approach is widely considered as a more efficient method for modelling artificial neural networks because it can easily compute the inverses of covariance matrices with high dimension that otherwise are computationally expensive. This study has developed a new filter, the First order Extended Ensemble Filter(FoEEF),that applies numerical solution in solving the inverses of high dimensional covariance matrices from $I\hat{t}o^{'}s$ stochastic state-space dynamical models. The research applies the new FoEEF Filter to initializes and estimates the weights and bias of artificial neural network. Comparison on the performance of FoEEF filter in estimating weights and bias are done against the performance of EKF on function estimations by using two different functions, $sin(x)+Q$ function and $3sin(x)^3 - 3sin(x)^2 - sin(x) + 1$ function, within 18 and 22 epochs. Emphasis of the performance is placed on the rate at which the models converge. This study gauges the performance by determining the minimum value of the mean square error produced from each epoch and average mean square error minimum value.The outcome from the study showed that FoEEF filter had the lowest value

**Corresponding author: E-mail: robertoryiema@gmail.com, oriema@gmail.com;*

of mean square error and average mean square error which was achieved with the least number of epochs compared to EKF filter. This study concluded that the new FoEEF performed better than EKF and is a more suitable filter for that can be used for initialization and estimation of weights and bias in artificial neural network.

Keywords: *Bayesian technique; filtering; estimation; state space dynamical system; extended ensemble kalman filter.*

2020 Mathematics Subject Classification: 62F15; 62F40; 62M05.

1 Introduction

Complex interaction of multiple factors simultaneously with non linear dynamics and computer science has allowed for the creation of intelligent agents like artificial neural network.[1] which can be used, among other things, to estimate mathematical functions.

Artificial neural network model is a model that is used to handle non linear data and approximate functions or other mathematical operators [1, 2, 3, 4].

Initialization and weight estimation in artificial neural network can be done by using different methods such as random process, maximum likelihood method, Bayesian technique, Cauchy initialization method etc. The major problem encountered when using any of this methods is computing the inverse of matrices. when such matrices are high dimensional, their computation becomes inefficient and expensive, i.e. the computation of their inverse takes a lot of time. To overcome this problem, this study developed FoEEF filter that uses empirical estimate to compute the inverse of high dimension matrix. From the study, it can be seen that FoEEF method initializes and estimates the weights and bias in a more efficient and way.

The central theme in artificial neural network models is analogy borrowed from the structure and function of human nervous system cells.

The building blocks and functionality of human nervous system is a combination of millions of neurons that are interconnected with each neuron having three parts which are

- Dendrite: This is the input part. Dendrite receive information as signals from other neurons or the surroundings.
- Cell body: The information processing part. It processes all information it receives from dendrite.
- Axon: This is the output terminal. Axon link neuron to each other by sending signals from soma to other neurons.

Fig. (1.1) shows three part structural components of a neuron in the human nervous system.[5]

Similar to human nervous system, artificial neural network is a combination of several neurons. Each neuron has three parts as shown in figure(1.2). These three parts can be described as

- Input node: The node receives input data x_i which are mathematical variables for $x_i|_{i=1,2.....n}$. Each x_i is assigned weight $w_{ji}|_{i=1,2.....n,j=1,2,....m}$ according to its importance.
- Activation and Transfer function: The product of x_i and w_i are summed to form a function $f(x) = \sum_{i=1}^{n} x_i w_{ji} + b_j$ where b_j is a bias. The function $f(x)$ is then transformed by the transfer function $\varphi(f(x))$ to obtain output.
- Output: The value $y_j|_{j=1,2.....m}$ such that has $y_j = \varphi(f(x))|_{\forall j}$.

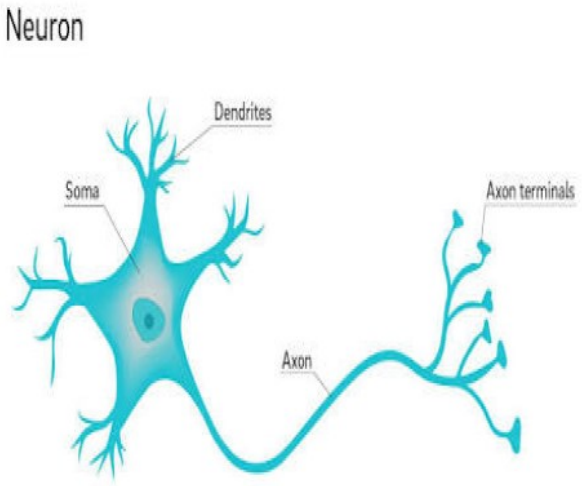

Fig. 1.1. Biological Neural Network Components

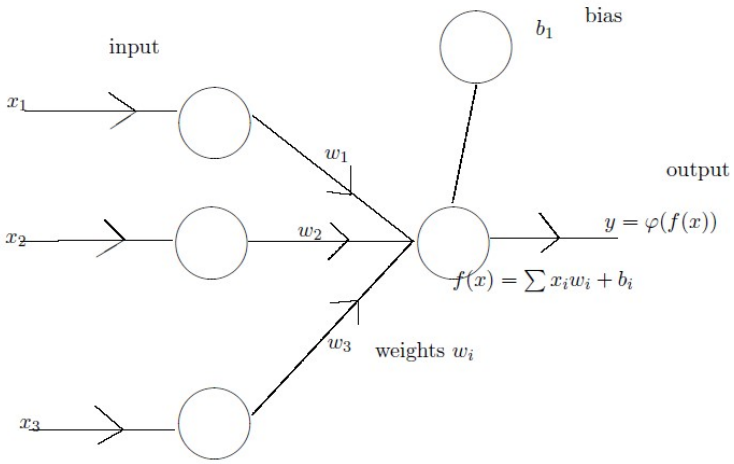

Fig. 1.2. Simple Artificial Neural Network

The representation in Fig. (1.2) is of a simple artificial neural network which can be referred to as a single layer perceptron composed of input nodes and output nodes only(no hidden layer). Artificial neural network generally are multi-layered with several complex layers in their architecture. Such networks are called multiple-layers perceptron.

Complex architecture in artificial neural network from input neuron, hidden neuron to the output neuron is represented in Fig. (1.3). This is based on a feed forward MLP where x_i is the input variable, w_{ji} represent the weight, the function $f(x)$ represents summation of all the products of variable and the weight, φ represents the transfer function that act on $f(x)$ and h_j is the output value.

Several neurons are arranged in layers which are always three, input layer, hidden layer and output layer. The columns of neurons in this layers are called nodes or units and each node or unit in a

layer is connected to every node or unit in the next layer. For easier understanding of multiple layer Perceptron, an architecture with input layer having two nodes, hidden layer with two nodes, output layer with two nodes and assumed bias of zero value to each node is represented by Fig. (1.3).

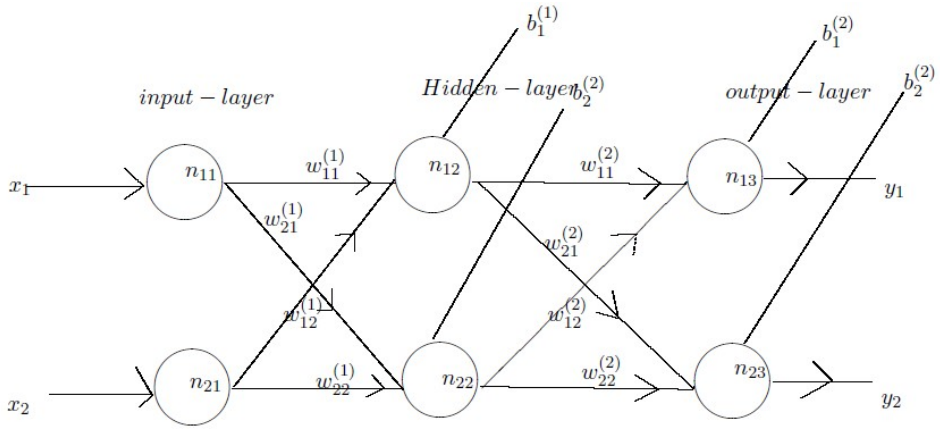

Fig. 1.3. Simple three layered perceptron

Where n_{pk} represents the pth neuron in the kth layer, $w_{ji}^{(k)}$ represents the weight from the ith node in the k^{th} layer to the j^{th} node in the $(k+1)^{th}$ layer.

Neuron n_{11} and neuron n_{21} form the input layer, neuron n_{12} and neuron n_{22} form the hidden layer and neuron n_{13} and neuron n_{23} form the output layer.

x_1 and x_2 are input variables to neuron n_{11} and neuron n_{21} respectively.

There are two output from neuron n_{11}. These are $\varphi(x_1 w_{11}^{(1)})$ and $\varphi(x_1 w_{21}^{(1)})$ that are the value for neuron n_{12} and neuron n_{22} respectively.

Similarly, the two output values from neuron n_{21} are $\varphi(x_2 w_{12}^{(1)})$ that is the value for neuron n_{12} and $\varphi(x_2 w_{22}^{(1)})$ that is the value of neuron n_{22}.

If we take $v_1^{(1)}$ and $v_2^{(1)}$ as the weighted sums at the hidden layer given by

$$v_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 \tag{1.1}$$

and

$$v_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 \tag{1.2}$$

Then

$$\begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & ww_{22}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

1.1 First Order Extended Ensemble Filter

The estimation of weight w_i and bias b_i can be done using different methods with different results. the best method is one which will make the neural network model to converge faster with a higher level of efficiency. In this research, a new method is developed, the FoEEF filter that estimates the weights and bias in a more efficient way.

The prediction state, kalman gain and innovation processes of the filter are

Prediction

$$f(w_t^i)dt + g(w_t^i)q^{\frac{1}{2}}(t)dw_t^{*i} \tag{1.3}$$

Gain

$$ph_w(w_i^i)r^{-1}(t) \tag{1.4}$$

Innovation

$$(dz_t - 0.5(h(w_t^i) + h(\hat{w}_t))dt) \tag{1.5}$$

Diffusion error

$$w_t^* \approx (0, Q_t) \tag{1.6}$$

Observation error

$$v_t^* \approx (0, R_t) \tag{1.7}$$

is applied to estimate weights and bias as a Bayesian method

$$f(w_t^i)dt + g(w_t^i)q^{\frac{1}{2}}(t)dw_t^{*i} \tag{1.8}$$

Gain

$$ph_w(w_i^i)r^{-1}(t) \tag{1.9}$$

Innovation

$$(dz_t - 0.5(h(w_t^i) + h(\hat{w}_t))dt) \tag{1.10}$$

Diffusion error

$$w_t^* \approx (0, Q_t) \tag{1.11}$$

Observation error

$$v_t^* \approx (0, R_t) \tag{1.12}$$

is applied to estimate weights and bias as a Bayesian method

1.2 Literature Review

Different methods can be used to initialize feed forward back propagation algorithms [6]. However, these methods tend to have a slower rate of convergence to the global minimum. To improve on the rate of convergence, several techniques have been applied since it is known that the rate at which the network convergences is highly sensitive to initial values assigned to weights and bias since the point at which weight estimation starts should be as close as possible to the desired outcome[6] [7] [8]. It is known that small values of initial weights leads to local minimum while large values end up with saturated neurons.

The most common initialization method is random initialization in which weights are generated from a small interval of between $[-\epsilon, +\epsilon]$.

Random initialization method has some disadvantages. First, it is difficult to pick the correct initial values of weights using this method. Secondly, for high value initial weights, the transfer function maps the high values of weights close to 1 and low values of weights close to zero, a complexity that greatly reduces rate of convergence in ANN training and makes the gradient decent to vanish[9]. Most importantly, solving the inverse of high dimensional matrices, which is an exact solution, is extremely difficult and limits the performance of neural network.

Back propagation algorithm with training input $n = |X|$, L number of layers in which K represents the k^th layer, $(K = 1, 2, ...L)$, $N(i)$ number of neurons in a layer K and activation function in layer

K and input x represented by $a_i^{(K,x)}$ can be developed from equation(2.1)

$$a_i^{(K,x)} = \sigma(\sum_{j=1}^{N}(N_K)w_{ji}a_j^{(K-1,x)}) \tag{1.13}$$

Algorithm 1.1. *General Back-propagation training algorithm [1] Data: L number of layers N(k) number of neurons in layer k, for k = 1,...,L* $w_{ij}^{(k)}$ *initial weights, for i = 1,...,N(k), j = 1,...,N(k-1), k = 2,...,L. X set of training inputs,* $n = |X|$ $y^{(x)}$ *desired output for all training inputs* $x \in X$ η *learning rate Result:* $w_{ij}^{(k)}$ *final weights, for i = 1,...,N(k), j = 1,...,N(k-1), k = 2,...,L, such that* $a^{(L,x)}y^{(x)}, \forall x \in X$ ***begin*** *while* $\exists x \in X : a^{(L,x)} \neq y^{(x)}$ *do for* $x \in X$ *do (for each training input)* $a^{(1,x)} = \sigma(x)$ *for k=2,...,L do* $z^{(k,x)} = w^{(k)}a^{(k-1,x)}, a^{(k,x)} = \sigma(z^{(k,x)})$ $d^{(L,x)} = (a^{(L,x)} - y^{(x)}).\sigma\prime(z^{(L,x)})$, *((.) component wise) product for k=L-1,...,2 do* $d^{(k,x)} = ((w^{(k+1)}))^T d^{(k+1,x)}).\sigma\prime(z^{(k,x)})$ *(right superscript T stands for transpose operator) for k=L,...,2 do* $w^{(k)} = w^{(k)} - \frac{\eta}{n}\Sigma_{x\in X}d^{(k,x)}(a^{(k-1,x)})^T$

In this back-propagation random initialization method, although solutions of inverse matrix are optimal, accumulation of covariance matrices, particularly high dimensional matrices, significantly slows down the process making it to be computationally expensive.

Drago *et al*[11] applied statistically controlled activation weight initialization (SCAWI) method to determine initial weights of their model. Their research used sigmoid function as transfer function and uses paralysed neuron percentage (PNP) to test how many times a neuron is in paralysed state. A paralysed state of a neuron is when it is saturated, a situation in which the magnitude of at least one of its output is greater than or equal to a predefined threshold value. The predefined value controls the need of a very large value of weights that makes back propagation impossible.

The main draw back to this model is that it does not operate outside adaptive momentum back propagation(AMBP) algorithm which is a modified version of Back Propagation algorithm and also PNP are computationally expensive.

Lehtokangas *et al* [12] used forward selection initialization(FSI) method to determine initial weight of their model in which the aim is to find the set of network weights that minimizes a performance function. The method uses a large pool of randomly initialised hidden units to select the best from the pool using the objective function used in the hidden unit training. The selected best is trained in a normal way. Their study used Cascade-Correlated (CC) algorithm that starts its learning with minimal network then adds hidden units one by one at each separate hidden layer thereby avoiding prior assumptions that need to be made on the initial size and structure of the neural network. The method has the advantage of learning very fast, determines its own size and topology and requires no back propagation.

Other methods applied in weight initialization includes partial least square method by Liu *et al* [13], the method of evolutionary algorithm by Huskan *et al* [14] used to initialise from a set of solutions, (OIVS) model which by Shimodaira [8] which is based on the equations that represent the characteristics of a node's information transformation mechanism.

Thangairulappan [6] proposed a single layer neural network that applies Cauchy inequality method based on sensitivity analysis to initialise weights. This method ensures that the output of hidden neurons are contained within the active region so that only relevant computations of covariance matrices are performed.

Their model was simulated on iris data set, breast cancer, two spiral and modelling function approximation.

In the iris data set, the minimum mean square error(MSE) was 0.000353 in a span of 0.1 seconds. This was a faster rate of convergence with less number of epochs as compared with random initialization method which had MSE of 0.000358 and 0.000356 within 0.2 and 0.4 seconds respectively. In the spiral dataset, they used neural net with three different architecture, that is 3-8-1, 3-10-1 and 3-11-1. the MSE obtained for the three were 0.078954, 0.078961 and 0.078950 respectively within 2.2, 3.0 and 2.5 seconds against random initialization methods with MSE of 0.078982, 0.078833 and 0.078929 within 1.7, 2.6 and 2.4 seconds.

For the case of three input non linear function, they used 4-8-1, 4-10-1 and 4-11-1 network structure. The MSE for the 4 structures were 0.000136, 0.000138 and 0.000137 respectively. The corresponding time span 0.9, 0.6 and 0.8 seconds. As much as the rate of convergence is improved and the number of epochs are reduced, major limitation of this Cauchy initialization method is that it is applicable only on a single layer neural network hence it cannot be applied in machine learning, deep learning or Artificial Intelligence neural networks.

D'Errico *et al* [15] developed a new form of Bayesian filter, the customised Kalman Filter which was later applied by Nadir *et al* [7] to initialize and estimate weights and bias of Artificial Neural Network. Customised filter uses circulant matrix to solve matrix inverse which reduces the computational cost and thus increases the efficiency of the network as compared to random initialization method. Their research simulated data on the field of character recognition, and then compared the rate of convergence using their customised filter method and random method.
Taking the dynamic of the process as

$$\mathbf{w}_{t+1} = A_t\mathbf{w}_t + B_t\mathbf{u}_t + \mathbf{P}_t \tag{1.14}$$

$\mathbf{w}$ as weight, $\mathbf{A}$,$\mathbf{B}$ are process matrix relating the state from $t+1$ to t^{th} process state where $\mathbf{P}$ is a white noise matrix and u is optional.
The measurements is given by

$$\mathbf{m}_t = \mathbf{w}_t + \mathbf{r}_t \tag{1.15}$$

where $\mathbf{r}$ representing measurement uncertainty.

The estimated Kalman filter is represented by

$$\hat{\mathbf{w}}_t = \mathbf{w}_t^- + K_t(\mathbf{m}_t - \mathbf{w}_t^-) \tag{1.16}$$

where K is kalman gain matrix and

$$w_t^- = A_{t-1}\hat{w}_{t-1} + B_{t-1}u_{t-1} \tag{1.17}$$

for some given prior estimate w_0^- of w_0.

The back propagation algorithm for classical Kalman filter [7].

D'Errico *et al* [15] and Murru *et al* [7] used Bayesian method in developing a simple Kalman gain matrix to minimize the error of measurement by considering each state of w_t as mutually independent normal random variable.

The approach used in Nadir *et al* [7] research on neural network introduced a correlation among w_t treating the weights as a random sampling process. Taking $\mathbf{w}_t$ and $\mathbf{M}_t$ as multi-variant random variable with

From Kalman filter equations

$$w_0^- = \hat{w}_{-1} = \hat{w}_{expt} \tag{1.18}$$

$$\hat{w}_t = w^- + K_t(z_t - H_t w_t^-) \tag{1.19}$$

$$w_t^- = A_{t-1}\hat{w}_{t-1} + B_{t-1}U_{t-1} \tag{1.20}$$

Where

- K_t is the Kalman gain matrix
- $\hat{w}_{t-1}$ is a pre-process estimate from judgement of expert.

kalman gain assigns suitable weights to z_t and w_t^- using a minimum root mean square error MSE by the minimization of $E(e_t^T e_t)$ with $e_t = |\hat{w}_t - w_t|$ and T is a transposition operator.

Kalman filter implements Bayesian estimation method [7] by customizing Kalman equations (1.1), (1.2) and (1.3)into a Bayesian form. Taking w and Z as the stochastic component of w_t^- and z_t respectively, and f_t as a probability density function at step t, then equations (1.1), (1.2) and (1.3)can be expressed in Bayesian form as

$$f(w/z) = \frac{f(z/w)f(w)}{\int_{-\infty}^{+\infty} f(z/w)f(w)dw} \tag{1.21}$$

where $f(w/z)$ is the posterior density, $f(w)$ is a normally distributed prior density with mean w_t^- and co-variance Q_t i.e. $f(w) \sim N(w_t^-, Q_t)$, $f(z/w)$ is normally distributed likelihood function with mean z_t and covariance R_t i.e. $f(z/w) \sim N(z_t, R_t)$ and the denominator $\int_{-\infty}^{+\infty} f(z/w)f(w)dw$ as a normalization factor.

From equation (1.4) it follows that

$$\begin{aligned} f(w/z) &\propto f(z/w)f(w) \\ = f(w/z) &\propto N(w_t^-, Q_t)N(z_t, R_t) \\ &= N(\hat{w}, P_t) \\ = N\left(\tfrac{R_t w_t^- + Q_t z_t}{R_t + Q_t}, \quad (\tfrac{1}{R_t} + \tfrac{1}{Q_t})^{-1} \right) \end{aligned} \tag{1.22}$$

From the equations (1.2), (1.3) and (1.4), the posterior probability density function is Gaussian with either $f(w/z) = N(\hat{w}_t, Q_t)$ where

$$\hat{w}_t = \frac{R_t w_t^- + Q_t z_t}{R_t + Q_t} \tag{1.23}$$

Equation (1.6) is a metro logical form of of Kalman filter equation (1.2) and the recursion for the final estimate $\hat{w}$ should stop at f_T, $0 \leq t \leq T$.

The equation (1.6) can be expressed in the form

$$\hat{w}_t = (Q_t^{-1} + R_t^{-1})^{-1}(Q_t^{-1} w_t^- + R_t^{-1} z_t) \tag{1.24}$$

$$P_t = (Q_t^{-1} + R_t^{-1})^{-1} \tag{1.25}$$

Their numerical result concluded that the rate of convergence was faster with application of Extended Kalman Filter at the weight initialization stage using the algorithm.

Algorithm 1.2. *Weights initialization algorithm based on Customised Kalman filter [1] Data: L number of layers N(k) number of neurons in layer k, for k = 1,...,L X set of training inputs, n = —X— $y^{(x)}$ desired output for all training inputs $x \in X$ $Q_0(k)$, for k=2,...,L w_0^- prior estimate of $\hat{w}^{(k)}$, for k=2,....,L $m_0(k) measurements of \hat{w}^{(k)}$ for k=2,....,L $R_0(k)$, for k=2,....,L Result $\hat{w}_2^{(k)}$*

optimized initial weights for back-propagation algorithm for k=2,....,L ***begin*** *for k=2,....,L do for t =0,1,2 do* $\hat{w}_t(k) = (Q_t^{-1}(k) + R_t^{-1}(k))^{-1}(Q_t^{-1}(k)w_t^-(k) + R_t^{-1}(k)m_t(k))$ $Q_{t+1}(k) = (Q_t^{-1}(k) + R_t^{-1}(k))^{-1}$ $w_{t+1}^-(k) = \hat{w}_t(k)$ $m_{t+1}(k) = Rnd(-h, h)$ *random sample in the interval (-h,h)* $(R_{t+1}(k))_{ii} = \frac{1}{N(k)N(k-1)}\Sigma_{x\in X}||d^{(k,x)}||^2, \forall i$ $(R_{t+1}(k))_{lm} = o.7, \forall lm$

The research used randomly sampled data between the interval $(-h, h)$ for the measurement $m_t(k)$. The co-variance matrix value from the main diagonal and chosen from sensitivity analysis was set to 0.7 involving Pearson coefficient. Diagonal of co-variance matrices $(R_t(k))$ were used as uncertainty of the measurement with high value of reflecting less accuracy of $m_t(k)$.

Their results determined that that their algorithm converged faster than random initialization and involved less number of iteration process.

The result from their research showed that the estimated weight $\hat{w}_t$ of the weight w_t depends on the balancing between prior estimation and measurements of w_t which is constructed to minimize the error $E((\hat{w}_t - w_t)(\hat{w}_t - w_t)^T)$. It also established that it is efficient to apply of Kalman filter as an optimal estimation method, just like the Least Square Estimation method or Minimum Mean Square Error estimation methods.

Estimation of weights and bias of Artificial neural network models depends on initialization that should make the initial values as close as possible to the target value. Initialization algorithms involves computation of matrix products and determination of their inverses. In such process the rate of learning is hampered by the process of computing the products and inverses of the matrices particularly when the dimensions of these matrices are very large which significantly reduces the time it takes to train the model or for the model to converge. Solutions to this problem have been proposed by several researchers using different approaches as mentioned in the literature review above. The current research done by Nadir and Miuru used Bayesian technique based on customised kalman filter with an algorithm that applies cyclic matrix. Cyclic matrices has less number of operations (matrix multiplications) than usual Kalman filter and in determining the inverses, cyclic matrix performs in a fast way since each row is a cyclic shift of the row above it. They can be diagonalized by using the Discrete Fourier Transform. The Customised filter is still significantly slower that the new First order Extended Ensemble Filter that avoids computation of matrices in its algorithm by using empirical estimates method to determine their products and inverses.

1.3 Training and Testing of Artificial Neural Network

This study trains and tests the First order Extended Ensemble Kalman Filter on artificial neural network model is performed on two functions,$sin(x) + Q$ and $sin(x).^3 + sin(x).^2 + sin(x) + Q$ with emphasis on efficiency and rate of convergence of the model.

In the training and testing process, data is split into two sets. 70% for the data is used to train the neural network and the remaining 30% of the data is used for testing.

The two processes are guided by Algorithm (5.1.1).

Algorithm 1.3. *Algorithm of First order Extended Ensemble Filter(FoEEF) [1] L is the set of layers in the neural network* $N(k)$ *the number of neurons in G for* $k = 1, 2, 3, 4.....L$ η *is the learning rate* $y^{(w)}$ *Target output* z_0 *Initial observations/measurement* R_0 *for* $k = 1.....L$ $([w_1^1, w_2^1, ...w_M^1], ... w_1^N, w_2^N, ...w_M^N)$ *Generate weights w randomly* $\hat{w}_0 = E(w_0)$.

$= \frac{1}{M}\sum_{i=1}^{M} w_i$.

Initialize the filter object with initial values of the state. $P_0 = E(w_0 - \hat{w}_0)(w_0 - \hat{w}_0)^T$

$= \frac{1}{1-M}\sum_{i=1}^{M}(w_0 - \hat{w}_0)(w_0 - \hat{w}_0)^T$
Initialize with empirical estimate of state covariance matrix P.

while $S(G) = 2$ *to* S^* *do for* $t = 0, 1, 2$ *do* z_{t+1} = *random.sample Random numbers sampling (-h,h)*

$h_w = \frac{\partial h}{\partial w}$ *given* $\hat{w}_t$ *The Jacobian of* h_w $w_{t+1}^i = w_t^i + ph_w(w_t^i)r^{-1}(t)(dz_t - 0.5(h(w_t^i) + h(\hat{w}_t))dt)$

State propagation for each ensemble i. $\hat{w}_{t+1} = E(w_{t+1})$.

$= \frac{1}{M}\sum_{i=1}^{M} w_{ji}$.
$P_{t+1} = E(w_t - \hat{w}_t)(w_t - \hat{w}_t)^T$

$= \frac{1}{1-M}\sum(w_{t+1} - \hat{w}_{t+1})(w_{t+1} - \hat{w}_{t+1})^T$
$(R_{t+1}(G))_{ii} = \frac{1}{N(G)N(G-1)}\sum_w ||d^{(G,w)}||^2$ $(R_{t+1}(K))_{lm} = 0.7$

2 Simulation, Analysis and conclusion

2.1 Introduction

Training and testing of artificial neural network models is a process that first involves first assigning to the network initial values of weights and bias. This process is called initialisation. The second part is to estimate the value of weights and bias using a back propagation method by updating those weights and bias upto the point at which the neural network is capable of functioning independently. This weights and bias are updated by minimising the difference between the output from the network and the target.

The neural network is then tested to determine the level at which it has learned enough in terms of the speed of convergences and efficiency.

One of the major applications of artificial neural network are facial recognition, character recognition, classification and mathematical ***function estimation***.

Experimentation in this research is based on function estimation.

A function estimation can be described in this case as an underlying mathematical function or mapping function that maps any set of input data to a set of output data with consistency, efficiency and without bias. The neural network tries to best approximate this mapping function. Basically artificial neural network are function approximation algorithm.

This study uses simulated data based on algorithm (5.1.1) to train the new FoEEF filter. In the research experiment, two functions are estimated,

- $sin(x) + Q$
- $3sin(x)^3 - 3sin(x)^2 - sin(x) + 1$

The experiments are simulated using Matlab software.

In the experimentation, two epochs, that is 18 and 22 epochs, are used. The two epochs are run in two different neural network structures, the 1-4-1 and 2-4-1 structures. The performance from the experiment generates by FoEEF filter is then compared with results generated by using EKF filter

with special emphasis on the rate of converge and efficiency. The aim of the experiment is to gauge which between the two filters convergence faster and is more efficient.

The research performs and analyses the experiment in three parts.

1. Part one involves analysis of observation made from graphical output after training and testing the neural network.

   In the output analysis for figures four, five, six,seven eight, nine, ten, eleven, twelve, thirteen and fourteen, the blue lines represent the target values and red lines represent the outputs from the filters.
   Analysis in this part is done after the filters are trained and tested by observing the extend at which the output values from the filters(red line) is mapped into target values(blue lines) and the level at which the training and testing output deviates away from their targets. An efficient filter is one whose output line is perfectly mapped into the target line with small deviation.

2. In the second part, the study tabulates the output values for analysis. A table with the number of epochs, convergence points and computed mean square error is constructed. The experiment aims to compare which of the two filters have minimum value of the convergence, the filter with the smallest least square error and the average mean square error. In the analysis, the filter that gives the smallest value for both convergence and mean square error is the one that is more efficient.

3. In the third part, convergence values (x-axis) is plotted against computed mean square error(y-axis) to determine which of the two filters has low distribution of mean square error.

2.2 Training and testing FoEEF and EKF on 1-4-1 network using sin(x)+Q function with 18 epoch

In this first experiment, sin(x)+Q function is used to train FoEEF and EKF on a 1-4-1 neural network structure using 18 epochs.

Fig. (4) and Fig. (5) represents training and testing of FoEEF and EKF within 18 epochs.

The blue lines represent the target values assigned for training and testing while the red lines represent output values from neural network after the training and testing. the results from the first experiment are represented in Figs. (5) and (6) below ***Training and testing of FoEEF filter.***

2.2.1 Training and testing of EKF filter

Graphical Analysis: From Fig. (4) that represent training and testing of FoEEF algorithm and Fig. (5) that represents training and testing of EKF algorithm, it can be observed that FoEEF has captured the training and testing data set in a much faster way than EKF. This is indicated by the fact that in the training data set, there is one single red line that is well matched with the blue line with little or no significant deviation of red lines (output values) from the blue lines (target values) when compared to the training data set in figure (5) which has three red lines with two of the lines significantly deviating away from the target values.

Fig. (4) also has output from FoEEF test data set which can be said to be more efficient than than the EKF test data set. This is because the two lines(red and blue) from FoEEF test result in Fig. (4) are closely marched when compared to EKF test result in Fig.(5) which has some small

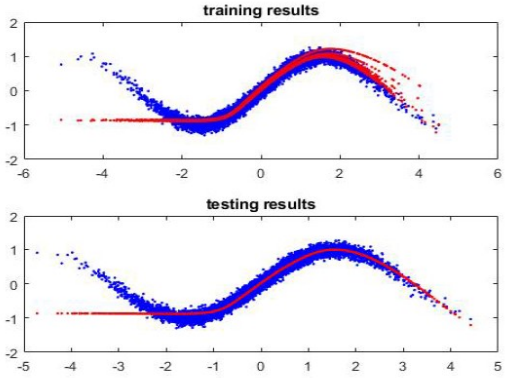

Fig. 4. Experimentation on FoEEF filter within 18 epoch

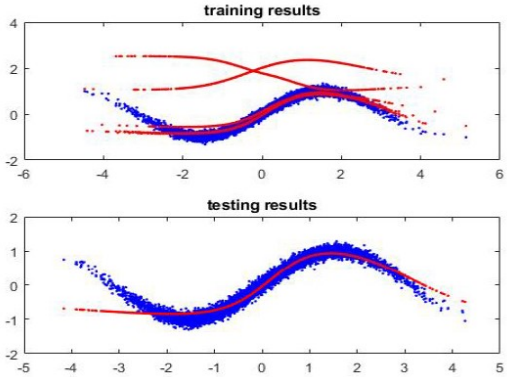

Fig. 5. Experimentation on EKF within 18 epoch

deviations at the two tips of the graph

Conclusion: Initialisation of weights and bias of artificial neural network takes place in the very early stages of the training process. Different models initialise neural networks with different values giving different outcomes. Observations that are made from the figures by comparing performances of FoEEF and EKF shows that FoEEF performs better than EKF and therefore it is more efficient filter than EKF in initialising and estimating the weights and bias of artificial neural network.

Table of output: Table (1) represent results from simulated data. The table has the eighteen number of epoch for FoEEF and EKF, the convergence value for each epoch and the corresponding mean square error measurement.

Table 1. Result from FoEEF and EKF on sin(x)+Q for 18 epochs

Epoch	EKF	MSE(EKF)	FoEEKF	MSE(FoEEF)
1	3.3945	0.41949	-0.8508	0.2861
2	2.9451	0.0393	0.6685	0.96914
3	3.9115	1.35649	-0.5747	0.06698
4	0.2775	6.09750	-0.8553	0.29096
5	4.6630	3.67177	0.25361	0.3243
6	5.9955	10.55396	0.9713	1.65685
7	6.3740	13.1564	-0.8796	0.3177
8	0.1157	6.9227	0.1044	0.17664
9	2.1579	0.3468	-0.2657	0.00251
10	-0.8320	12.8079	-0.8707	0.3078
11	1.6702	1.15909	-0.7729	0.20885
12	1.5647	1.39739	-0.8719	0.30914
13	5.6730	8.56256	0.2716	0.3451
14	-0.3333	9.48709	-0.7147	0.1590
15	-0.7894	12.5019	-0.8773	0.31518
16	-0.2454	8.95094	-0.8584	0.2943
17	3.4936	0.55769	-0.8391	0.27374
18	-0.3536	9.61256	0.7208	1.07472

2.2.2 Analysis from the table

Observation made from table (1) shows that the minimum value of mean square error(MSE) for FoEEF is 0.002519036. This minimum value occurs at the *9th* epoch for FoEEF. Comparatively, a run of EKF gives the minimum value of mean square error(MSE) of 0.039317. This values occurs at the *2nd* epoch.

The average mean square error for FoEEF is 0.409945 while the average mean square error for EKF is 5.97785833. From the table, it can be observed that EKF has a high value of average mean square error when compared to the value of the mean square error of FoEEF.

The comparison are relayed on the array below.

$Algorithm$	$network$	$Epoch$	$MinMSE$	$AverageMSE$
$FoEEF$	$1-4-1$	9	0.00251	0.409945
EKF		2	0.0393	5.97785833

Conclusion from the table: The minimum value of mean square error for EKF is ***fifteen times numerically larger*** than the minimum value of mean square error for FoEEF filter. This is a significant difference in efficiency measurement between the two filters.

The average values for the mean square error for the two filters shows that EKF filter has a ***fourteen times numerically larger*** average mean square error than FoEEF filter .

This research concludes that FoEEF filter convergence faster than EKF filter because of its very lower mean square error of 0.00251 when compared to a much higher mean square error for EKF filter of 0.0393.

The research also concludes that FoEEF filter is more efficient than EKF filter given that the the average mean square error for FoEEf filter of 0.409945 is far much lower than the mean square error for EKF filter of 5.97785833.

Analysis from the graph on the distribution: A analysis on distribution of the mean square error of FoEEF filter and the mean square error of EKF filter is shown on Fig. (6) below. The figure represent a plot of mean square error(MSE) from the output measurement of FoEEF (blue lines) filter and the mean square error from output measurement of EKF(red lines) filter.

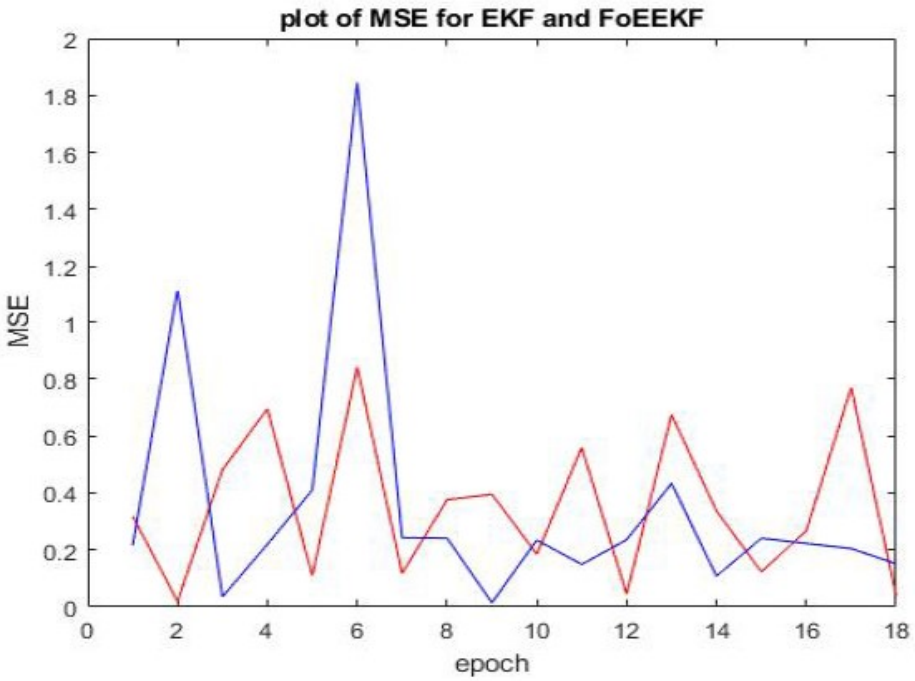

Fig. 6. Training and Testing using sin(x)+Q in 1-4-1 Architecture for 18 epoch

The trajectory observed from figure(6) shows the distribution of mean square error of FoEEF filter to be lower than the distribution of mean square error of EKF.

Conclusion: FoEEF filter is more efficient than EKF filter in initialising the weights and bias. This is indicated by figure (6) in which it can be observed that the trend of the distribution of mean square error of FoEEF filter is much lower than the trend of the distribution of mean square error of EKF filter.

2.3 Training and testing of FoEEF and EKF on 1-4-1 network using sin(x)+Q function with 22 epoch

Second experiment on training and testing is done on FoEEF and EKF filters using the same functional estimation sin(x)+Q function on the same 1-4-1 network structure but within 22 epochs. The aim of this experiment is to gauge the performance of the two FoEEF filter and EKF filters when the number of epochs are different.

The Figs. (7) and (8) are outputs after training and testing FoEEF filter and EKF filter on 22 epochs respectively

Training and testing FoEEF: Fig. (7) shows the graphical output from training and testing of FoEEF filter on 22 epochs.

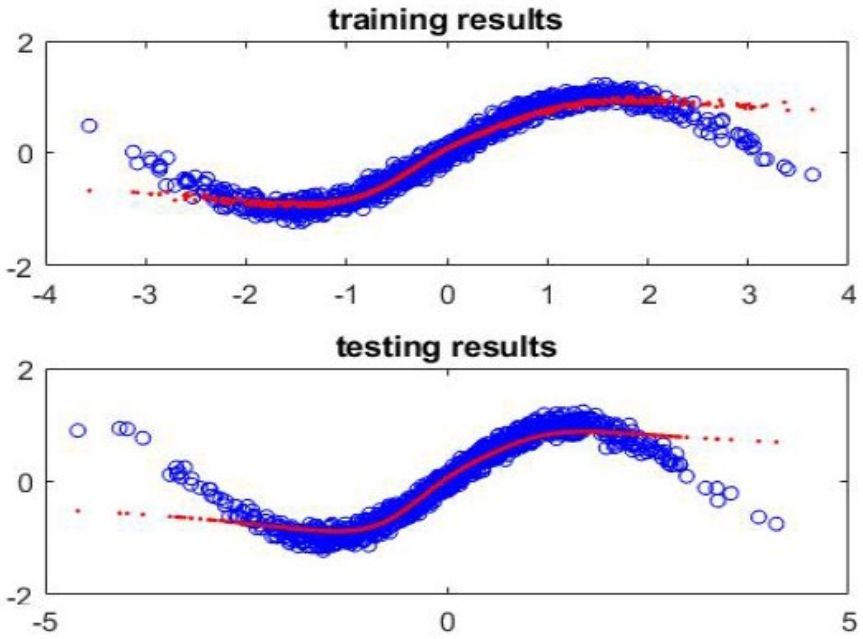

Fig. 7. Estimation with 22 epoch for FoEEF

Training and testing of EKF filter on 22 epochs: Fig. (8) shows the graphical representation of the output from training and testing of EKF filter on 22 epochs.

Analysis from the graph of figures 7 and 8: The observation made from training and testing of FoEEF filter in figure (7) shows that the output values (red lines) from FoEEF filter is more closely mapped to the target values (blue lines). Deviations of output from the target is therefore noticeably small when compared to deviations from EKF filter in figure(8). This is because in figure(8), EKF training data results shows some two red lines that are significant deviating from the target value(blue lines). This large deviation shows that FoEEF filter has captured the training data faster than EKF filter.

Conclusion: FoEEF converges faster that EKF filter hence it is a better filter than EKF filter and therefore initialise weights and bias in artificial neural network in a faster and more efficient way than EKF filter. This shows that FoEEF filter has a faster rate of convergence.

Analysis from the table: Table (2) represents output values for each of the 22 epochs, the values at which the network convergences, computed mean square error(MSE) for both FoEEF and EKF filters, average MSE for for both FoEEF and EKF.

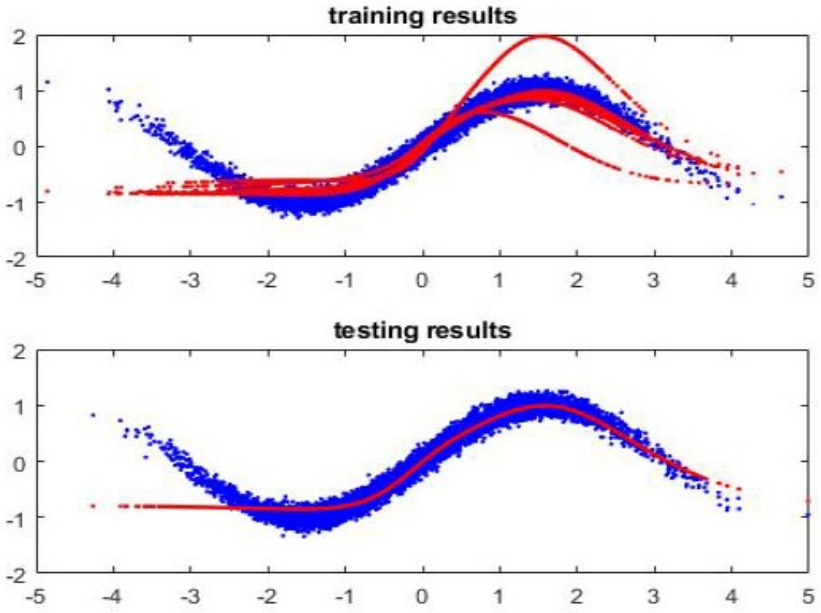

Fig. 8. Estimation with 22 epoch for EKF

From Table(2), it is be observed that the minimum mean square error value for our new filter(FoEEF) is 0.008114226. This value occurs at the very first epoch in the 22 epoch training. The minimum mean square error value for EKF filter is 1.00538 which occurs at the second epoch. It is observed that the EKF filter mean square error value is ***one hundred and twenty four times*** lager than the mean square error of FoEEF filter. This difference can be considered as highly significant.

The table also shows the values obtained from EKF filter to be generally lager than values from FoEEF filter. The average mean square error for FoEEF filter is 0.53299 while that for EKF filter is 8.24229. The value from EKF filter is ***fifteen times lager*** than the value for FoEEF filter.

The results form the table are summarised below

Algorithm	*network*	*Epoch*	*MinMSE*	*averageMSE*
FoEEF	$1-4-1$	1	0.008114226	0.53299
EKF		2	1.00538	8.24229

Conclusion: The difference between the minimum mean square error for FoEEF filter(0.00811) and and that of EKF filter(1.00538) is major. EKF gives ***one hundred and twenty four times numerically larger*** values of mean square error than FoEEF filter.

The research computed the average mean square errors for both filters. The average mean square error from EKF filter output(8.24229) is ***fifteen times numerically larger*** when compared to the mean square error from FoEEF filter output(0.53299)

The analysis from the experiment shows how our new FoEEF filter converges faster than EKF filter. It also shows that the efficiency exhibited by our new FoEEF filter is better than EKF filter.

Table 2. Result from FoEEF and EKF on sin(x)+Q for 22 epochs

Epoch	EFK	MSE(EKF)	TIME(S)	EEFK	MSE(FoEEF)
1	0.02224	7.42327	0.0032	-0.22581	0.00811
2	1.74412	1.00538	0.0025	0.56729	0.78001
3	-1.09165	14.73380	0.0019	-0.523918	0.04327
4	-1.14944	15.18078	0.0019	-0.59115	0.07577
5	0.11160	6.94430	0.0019	0.50770	0.67830
6	0.70720	4.16000	0.0017	0.77863	1.19797
7	0.90889	3.37794	0.0017	-0.65613	0.11576
8	-0.92212	13.46112	0.0017	-0.65416	0.11443
9	-0.82698	12.77205	0.0018	0.73897	1.11273
10	0.54916	4.82965	0.0019	-0.11160	0.20239
11	-0.68750	11.79454	0.0018	-0.55788	0.058562
12	0.70103	4.1852	0.0019	-0.36431	0.00234
13	0.11190	6.94273	0.0019	0.73301	1.10020
14	0.83977	3.63680	0.0019	-0.64186	0.28214
15	0.59241	4.64142	0.0016	0.85714	1.37601
16	-0.95617	13.71213	0.0018	0.96247	1.63422
17	0.02536	7.40626	0.0019	0.78799	1.21855
18	0.07577	7.13444	0.0017	-0.58947	0.074848
19	-0.9251	13.48361	0.0017	-0.67951	0.13222
20	-0.76719	12.34828	0.0017	-0.68205	0.13407
21	0.74899	3.99126	0.0017	0.59137	0.82312
22	-0.11072	8.16549	0.0019	0.4330	0.56086

The results analysed from Table(2) proves that FoEEF filter is more suitable at weight and bias initialisation and estimation in artificial neural network than EKF.

Distribution Analysis: A graphical analysis is done by plotting the values of MSE for each epoch.

Observation from Fig.(9), a plot of the distribution of mean square error from FoEEF and EKF filters for each epoch shows that generally, the new FoEEF filter has lower distribution of mean square error for each number of epoch than EKF filter. This implies that our new FoEEF filter is much more efficient in most stages of epochs than the EKF filter.

3 Estimating FoEEF and EKF models using the function y=$3sin(x)^3-3sin(x)^2-sin(x)+1$ in 2-4-1 Neural Network with 18 and 22 Epochs

In the third experiment, the study performed a further investigation of FoEEF and EKF filters on a more complex trigonometric function using a 2-4-1 neural network structure which was run on two epochs. The first experiment was done using 18 epochs and the second was done using 22 epochs.

Training and testing of FoEEF filter using 18 epochs: Fig.(10) below represent training and testing of FoEEF within 18 epochs on y=$3sin(x)^3 - 3sin(x)^2 - sin(x) + 1$ function.
Training and testing of EKF filter using 18 epochs: Fig.(10) represent training and testing of EKF within 18 epochs on y=$3sin(x)^3 - 3sin(x)^2 - sin(x) + 1$ function.

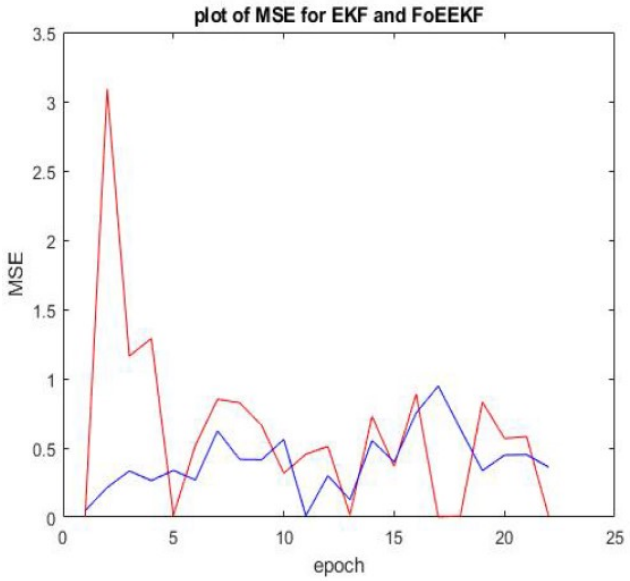

Fig. 9. Distribution of MSE on 22 epoch

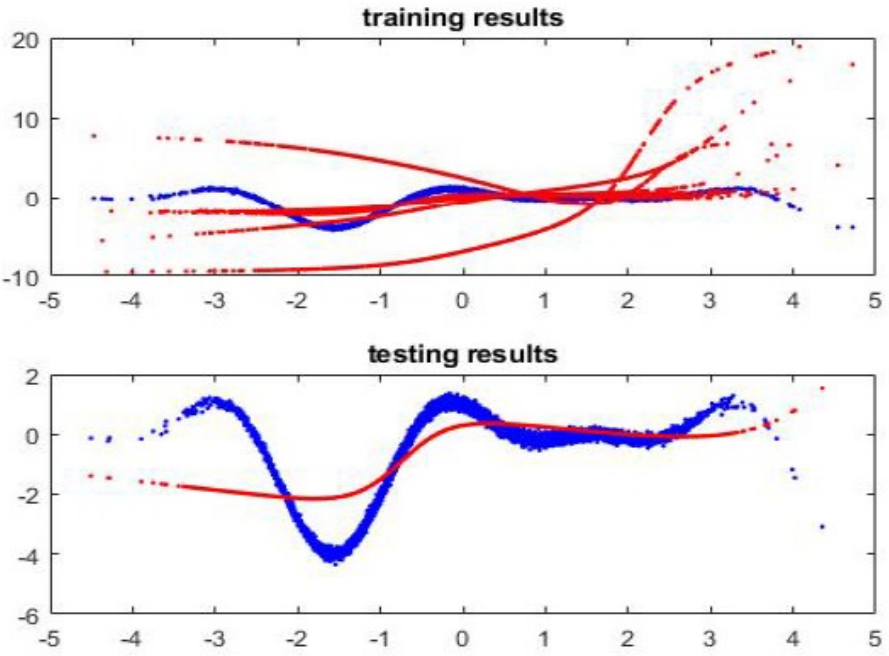

Fig. 10. Estimation of FoEEF with 18 epoch

Graphical Analysis and conclusion: As the function to be estimated becomes more complex, it

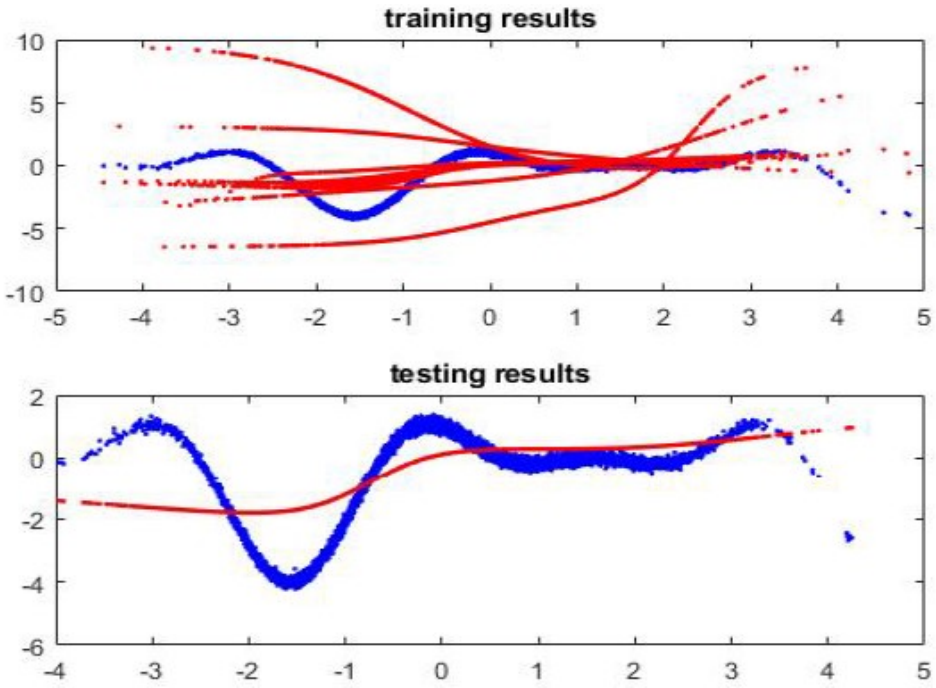

Fig. 11. Estimation of EKF with 18 epoch

can be observed that EKF filter represented by Fig. (11) looses track and becomes less efficient in the training process. This is indicated by the red lines (output values) from the neural network in Fig. (11) which has five very large deviations away from the blue lines (target values). Comparing deviation of training data set in figure (11) with deviations of the same training data set in Fig.(10) that represent output of FoEEF filter, it can be seen that there are only two small deviation of red lines in figure (10) from the target(blue line).

At the validation stage, EKF filter performance(red lines) represented by figure(11) is even less efficient since it can not estimate the target function(blue line) well. Output from EKF filter gives a near straight line that does not match the target Values at all. When this is compared with output from FoEEF filter, it can be seen that FoEEF filter curves and matches the target values in a in a better way than EKF filter.

Analysis from Table: Table (3) below represents output from simulation done the on the function y=$3sin(x)^3-3sin(x)^2-sin(x)+1$ that is run within 18 epochs. The table contains epochs numbers, the convergence value for each epoch and computed mean square error for each epoch.

Analysis and Conclusion: From the table(3) it can be seen that the minimum mean square error for the new filter(FoEEF) is 0.005711518 which occurs at the *5th* epoch. comparatively, the minimum value of mean square error for EKF filter is 6.01600 which occurs at the *8th* epoch.

Table 3. Result from FoEEF and EKF for 18 epochs

Epoch	EKF	MSE(EKF)	FoEEF	MSE(FoEEF)
1	1.03893	2.91685	-5.52982	27.18506
2	-3.69554	41.50398	5.30047	31.54359
3	0.24000	6.28408	0.57006	0.78491
4	6.2074	11.97623	-0.79780	0.23224
5	-0.04161	7.77533	-0.24031	0.00571
6	0.05272	7.25809	-1.36830	1.10757
7	-0.14537	8.36473	0.20387	0.27015
8	0.2940	6.01600	0.17390	0.23989
9	-1.34882	16.77429	0.35022	0.4437
10	0.27221	6.12362	0.24281	0.31215
11	-0.00124	7.55184	0.13221	0.20079
12	-0.47969	10.41035	0.32776	0.41429
13	-0.37608	9.75250	-1.43684	1.25653
14	-1.76350	20.34296	-0.02634	0.08383
15	-1.62469	19.1101	0.36190	0.45940
16	-0.08141	7.99885	0.31150	0.39362
17	0.19908	6.49091	0.12774	0.19680
18	0.26072	6.18064	0.03616	0.12394

This difference is major and significant since it shows that the mean square error for EKF filter is ***three times lager*** than that of FoEEF filter. In terms of efficiency, the new FoEEF filter is more efficient than EKF filter when functions become complex since the experiment shows that FoEEF filter has a smaller convergence and average mean square error than EKF filter.

The array below shows values of MSE square error and MSE.

Algorithm	*network*	*Epoch*	$MinMSE$	$averageMSE$
$FoEEF$	$2-4-1$	5	0.005711518	3.6252
EKF		8	6.01600	11.629

Distribution Analysis: Fig.(12) is a plot of the mean square error(y-axis) vs the epoch in a 18 epoch simulation. the figure show the distribution of FoEEF filter(blue line) and EKF filter(red line). The aim of the graph in Fig. (12) is to determine which of the two filter has a lower distribution of the mean square error from the experimentation.

From Fig. (12), it be seen that the distribution of the mean square error of FoEEF filter is lower than than of EKF filter. Most of the values for the new filter as observed from Fig. (12) lies on the lower part of the axis just next to the x-axis that has a zero value.
The study concludes that FoEEF filter is more efficient than EKF filter when estimating complex function.

3.1 Training FoEEF and EKF using 22 epochs

Fig. (13) below represent training and testing of FoEEF with 22 epochs on y=$3sin(x)^3 - 3sin(x)^2 - sin(x) + 1$ function.

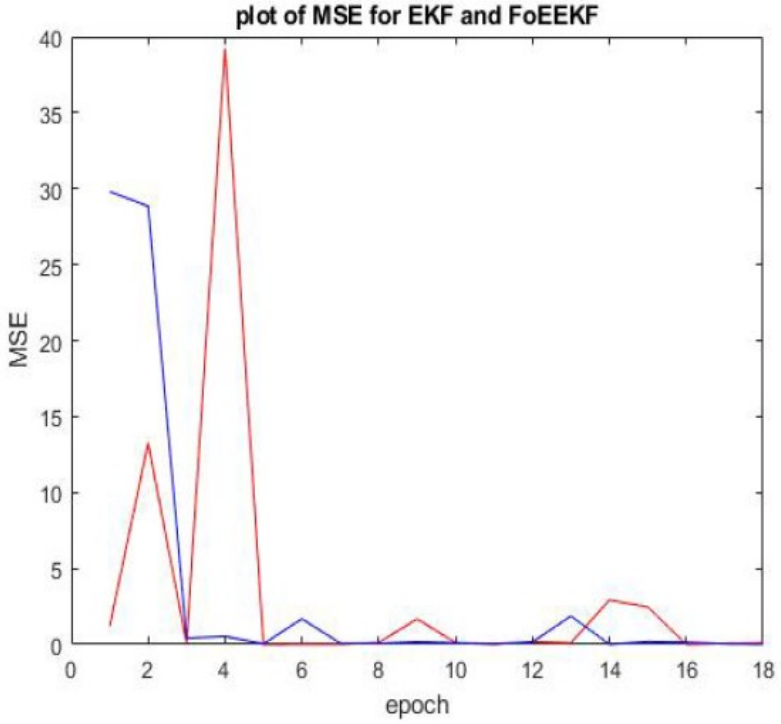

Fig. 12. Distribution of MSE for 18 epoch

3.1.1 Training and testing of FoEEF filter using 22 epochs

Training and testing EKF filter using 22

Graphical Analysis: Figs.(13) and (14) shows the training and testing of the new FoEEF filter and EKF filter respectively. It can be seen from the two figures that the output from the new FoEEF filter represented by red lines in figure (13) is closely matched to the target values that is represented by the blue lines for both training and testing process. This study compared the output from figure(13) with that of figure(14). Training and testing of FoEEF in figure(13) matched the target values than output from EKF filter in figure(14) which can be seen to have three lines(red) that significantly deviates away from the target line(blue).

Conclusion: When the results from the two graphs of figure (13) and figure (14) are compared, this study concludes that the new FoEEF filter initialises and estimates the weights and bias of the training data in a more efficient and faster way than EKF filter. This is shown by how much the outputs represented by graphs from figure(13) and (14) matches with the lines that represent the values of the target. Output lines(red) from figure(13) matched the target value in a more efficient way than the graph in figure (14) and therefore FoEEF filter is a better filter than EKF filter and is more suitable for use in initialising and estimating weights and bias of artificial neural network.

This study did tabulated the result from simulation on FoEEF and EKF filters for the number of epochs, the convergence value for each epoch, the mean square error for each filter at each epoch and the average mean square error for each filter.

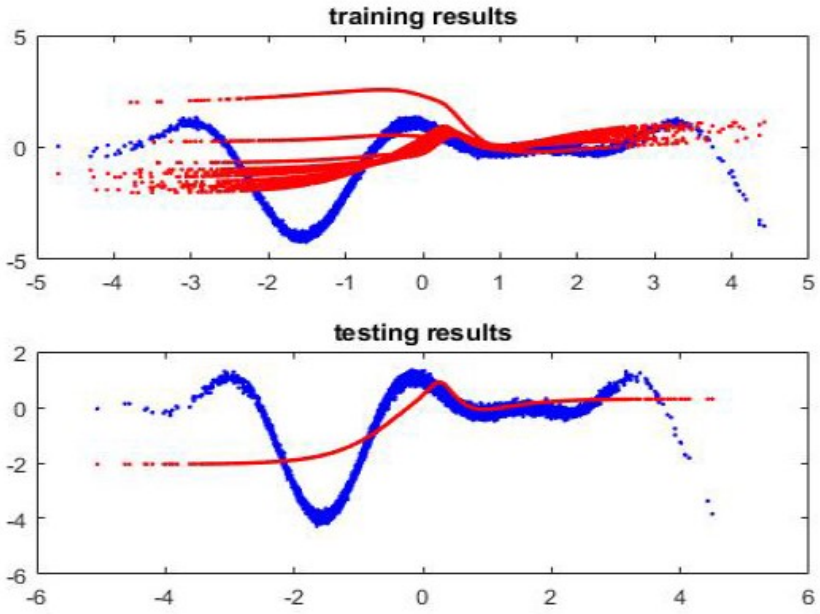

Fig. 13. Estimation of FoEEF with 22 epoch

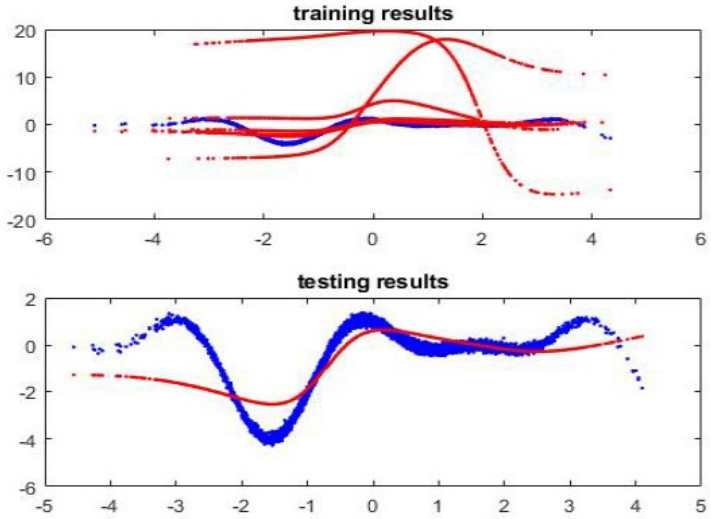

Fig. 14. Estimation of EKF with 22 epoch

The main aim is to determine which filter has the smallest convergence value, mean square error and average mean square error which is are measures of efficiency of the filters. The results for the experiment are represented in Table(4).

Table 4. FoEEF with 22 epochs

Epoch	FoEEF	MSE(FoEEF)	EKF	MSE(EKF)
1	0.9736	0.965	15.1323	1.1970
2	0.5432	0.251	17.95141	13.252
3	0.9984	0.0387	1.5642	0.08711
4	-0.4093	0.47	0.9690	39.22
5	-0.6336	0.0008	-2.0727	0.00181
6	0.5359	0.263	-0.2952	0.01163
7	0.1549	0.703	0.4780	0.0081
8	-1.1825	0.054	0.6004	0.1219
9	-1.0108	1.791	-1.4501	1.6735
10	0.2109	0.248	-0.8224	0.1071
11	0.6503	0.804	-0.1132	0.0029
12	-0.3222	1.883	0.6436	0.1802
13	0.2102	0.043	0.2525	0.1029
14	0.2453	0.086	0.5311	2.9184
15	-0.8497	1.275	0.3889	2.3624
16	0.0075	1.0053	0.9860	0.0068
17	-0.026	0.093	-1.56835	0.06463
18	0.09418	1.990	0.43938	0.09977
19	-0.4749	0.067	-0.6328	4.71
20	-1.3722	0.356	0.30358	1.52
21	-1.432	0.108	0.33715	1.44
22	0.399	0.7494	0.64559	0.799

Analysis from the table: Observations from tables (4) shows that the minimum mean square error(MSE) for FoEEF filter has a value of 0.0008 and occurs at the *5th* epoch. The minimum mean square error value for EKF filter is 0.0068 and occurs at the *16th* epoch.
The study computed the ratio of mean square error of FoEEF and EKF filters. The mean square error of EKF filter is ***eight times numerically larger*** than the mean square error of FoEEF filter.

The average mean square error of EKF filter was found to be 3.1767 while the average mean square error for FoEEF filter is 0.602. The study also computed the ratio of the average mean square error of EKF and FoEEF filters. The average mean square error for EKF filter is ***five times numerically larger*** than the average mean square error for FoEEF filter.

The array below represents the number of epochs, minimum mean square error for FoEEF filter and EKF filter and the corresponding average mean square error for both filter.

$Algorithm$	$Network$	$Epoch$	$MinMSE$	$AverageMSE$
$FoEEF$	$2-4-1$	5	0.0008	0.602
EKF		16	0.0068	3.1767

Conclusion: Values from the table shows that the mean square error of EKF filter is eight times lager than that of FoEEF filter. The average mean square error for both FoEEF and EKF filters was computed and the value for EKF filter was five times numerically larger than that of of FoEEF filter.

This result from the experiment shows that the new FoEEF filter is more efficient that EKF filter hence the new FoEEF filter is more suitable for initialising and estimating weights and bias of artificial neural network model.

4 Conclusion

This study proposes a new method of initialising and estimating weights and bias in artificial neural network(ANN) by using First order Extended Ensemble Filter(FoEEF). the new proposed method uses numerical estimate to compute covariances of high dimensional matrices. The results from the performance of the proposed (FoEEF) was compared with the performance of Extended Kalman Filter(EKF) to gauge the eficiency of the new filter using back propagation algorithm. The study showed that the new (FoEEF) method performed better than (EKF) in terms of rate of convergence, time, epoch and mean square error. This could be seen from the graphical.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Girosi F, Jones M, Poggio T. Regularization theory and neural network architectures. Neural Comput. 1995;7:219,269.

[2] Poggio T, Girosi F. Regularization algorithms for learning that are equivalent to multilayer networks. Science. 1990b;247:978,982.

[3] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Networks. 1989;2:359,366.

[4] Ito Y. Representation of functions by superpositions of a step or sigmoidal function and their applications to neural network theory. Neural Networks. 1991;4:385,394.

[5] Pavan vadapall. Biological Neural Network: Importance, Components and Comparison; 2021.

[6] Thangairulappan K, Subramanian S. A New Weight Initialization Method Using Cauchys Inequality Based on Sensitivity Analysis, Journal of Intelligent Learning Systems and Applications. 2011;3:242,248.

[7] Nadir M, D'Errico E. An algorithm for concurrent estimation of time-varying quantities; 2012. Available:http://iopscience.iop.org 0957-0233/23/4/045008

[8] Shimodaira H. A Weight Value Initialization Method for Improved Learning Performance of the Back Propagation Algorithm in Neural Networks, Proceedings of the sixth Internation Conference on Tools with Artificial Intelligence, New Orleans. 6,9 November 1994;672, 675. DOI:10.1109/TAI.1994.346429

[9] Saurabh Y. Weight Initialization Techniques in Neural Networks; 2018.

[10] Angwenyi D. Time-Continuous State and Parameter Estimation with application to SPDEs. PhD Thesis. Institute Fur Mathematik AG Numerische Mathematik. University of Potsdam; 2019.

[11] Drago P, Ridella S. Statistically Controlled Activation Weight Initialization (SCAWI), IEEE Transactions on Neural Networks. 1992;3(4):899-905.

[12] Lehtokangas M, Saarinen J, Kaski K, Huuntanen P. Initializing Weights of a multilayer perceptron network by using orthogonal least squares algorithm. Neural Computer. 1995;7:982,999.

[13] Liu Y, Zhou CF, Chen YW. Weight Initialization of Feedforward Neural Networks by means of Partial Least Squares, International Conference on Maching Learning and Cybernetics, Dalian. 2006;3119,3122, 13,16.

[14] Huskan M, Goerick C. Fast Learning for Problem Classes Using Knowledge Based Network Initialization. Proceedings of International Conference on Neural Networks. Como, 24-27 July 2000;619-624.

[15] DErrico GE. A la Kalman filtering for metrology tool with application to coordinate measuring machines IEEE Trans. Ind. Electron. At Press; 2011.
DOI:10.1109/TIE.2011.2162212

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
https://www.sdiarticle5.com/review-history/86132