HJB equation based learning scheme for neural networks

Conference Paper · May 2017

DOI: 10.1109/IJCNN.2017.7966134

4 authors, including:

Vipul Arora
Indian Institute of Technology Kanpur
23 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Tharun Reddy
Indian Institute of Technology Kanpur
9 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Ajay Yadav
Swami Keshwanand Rajasthan Agricultural University - Bikaner
20 PUBLICATIONS **80** CITATIONS
SEE PROFILE

HJB Equation Based Learning Scheme for Neural Networks

Vipul Arora
Faculty of Linguistics,
University of Oxford, OX1 2HG
Email: vipul.arora@ling-phil.ox.ac.uk

Laxmidhar Behera, Tharun Kumar Reddy, Ajay Pratap Yadav
Department of Electrical Engineering,
IIT Kanpur, India 208016
Email: {lbehera,tharun}@iitk.ac.in, ajaypratapyadav@gmail.com

***Abstract*—A control theoretic approach is presented in this paper for both batch and instantaneous updates of weights in feed-forward neural networks. The popular Hamilton-Jacobi-Bellman (HJB) equation has been used to generate an optimal weight update law. The main contribution in this paper is that a closed form solutions for both optimal cost and weight update can be achieved for any feed-forward network using HJB equation. The proposed approach has been compared with some of the existing best performing learning algorithms. It is found as expected that the proposed approach is faster in convergence in terms of computational time. Some of the benchmark test data such as 8-bit parity, breast cancer and credit approval, as well as 2D Gabor function have been used to validate our claims.**

I. Introduction

Since the advent of the popular back propagation (BP) algorithm [**?**], issues concerning fast convergence have caught the attention of researchers. One of the very first works by Hagan and Menhaj [**?**] makes use of the Levenberg-Marquardt (LM) algorithm for training weights of multi-layered networks in batch mode. Several methods have been proposed in order to improve the performance and convergence rate of BP algorithm. Gradient descent scheme that is at the heart of BP uses only first derivative. Newton's method [**?**], [**?**] enhances the performance by making use of the second derivative of the error function that is computationally intensive. Improving upon this, Gauss Newton's method approximates the second derivative with the help of first derivatives and hence, provides a simpler way of optimization. The LM algorithm [**?**], [**?**] modifies the Gauss Newton's method to improve the convergence rate, by interpolating between the gradient descent and the Gauss-Newton method. Faster convergence has been reported using extended Kalman Filtering (EKF) approach [**?**] and recursive least square approach [**?**]. Cong and Liang [**?**] derive an adaptive learning rate for a particular structure of neural networks using resilient BP algorithm with an aim of stability in uncertain environment. Man *et al.* [**?**] use BP with Lyapunov stability theory, while Mohseni and Tan [**?**] use BP with variable structure system for robust and fast convergence. Using Lyapunov function and stability principle, Behera *et al.* [**?**] propose a scheme for adaptive learning rate associated with BP algorithm. Ludwig and Nunes [**?**] propose modifications to the BP algorithm using maximum-margin based objective function. Asaduzzaman *et al.* [**?**] add noise to the gradient, while Ninomiya [**?**] proposes stochastic quasi-Newton method to improve convergence properties.

In this paper, we suggest using Hamilton-Jacobi-Bellman (HJB) equation for solving this problem. The HJB equation comes from dynamic programming [**?**], which is a popular approach for optimal control of dynamical systems [**?**], [**?**].

In the proposed scheme, the weight update law has been converted into a control problem and dynamic optimization has been used to derive the update law. The derivation of the HJB based weight update law is surprisingly simple and straightforward. The closed form solution for the optimal cost and optimal weight update law have simple structures. The proposed approach has been compared with some of the existing best performing learning algorithms and is found to be faster in convergence in terms of computational time.

The rest of the paper has been organized as follows. Sec. II represents the supervised training of FFNN as a control problem and derives optimal weight update law using HJB equations. Sec. III derives HJB based weight update law in instantaneous mode, with a special case of single output neural network. Simulation results are presented in Sec. IV. A detailed discussion on convergence behavior has been made in Sec. V. Concluding remarks are provided in Sec. VI.

II. HJB based Offline Learning of FFNN

Consider a two-layer FFNN, with $N_0 - N_1 - N_2$ structure, where N_0, N_1, N_2 stand for the number of neurons in the input, hidden and output layers, respectively. Fig. 1 shows a block schematic of the same. The non-linearity is introduced by using a sigmoidal activation function at each neuron.

$$h_{i_1} = \sum_{i_0} w_{i_1 i_0} x_{i_0} \tag{1}$$

$$v_{i_1} = \frac{1}{1 + e^{-h_{i_1}}} \tag{2}$$

$$h_{i_2} = \sum_{i_1} w_{i_2 i_1} v_{i_1} \tag{3}$$

$$y_{i_2} = \frac{1}{1 + e^{-h_{i_2}}} \tag{4}$$

where, i_0, i_1, i_2 index the neurons in the input, hidden and output layers, respectively.

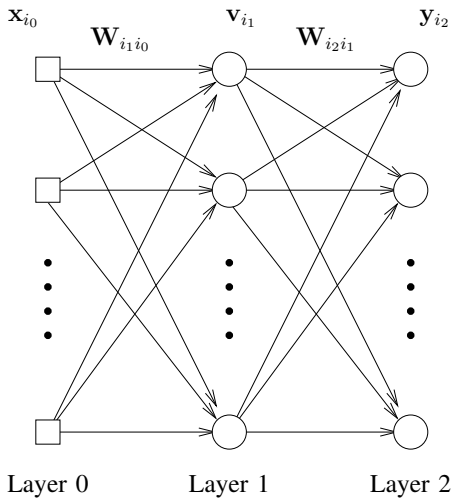

Fig. 1. Schematic of a two-layer FFNN.

The output $\mathbf{y}_p \in \mathbb{R}^{N_l}$ (l is the index of output layer) for a given input pattern $\mathbf{x}_p \in \mathbb{R}^{N_0}$ can be written as,

$$\mathbf{y}_p = \mathbf{f}(\widehat{\mathbf{w}}, \mathbf{x}_p) \tag{5}$$

Here, $\widehat{\mathbf{w}} \in \mathbb{R}^{N_w}$ is the vector of weight parameters involved in the FFNN to be trained, with total N_w weight parameters. The derivative of $\mathbf{y}$ w.r.t. time t is

$$\dot{\mathbf{y}}_p = \frac{\partial \mathbf{f}(\widehat{\mathbf{w}}, \mathbf{x}_p)}{\partial \widehat{\mathbf{w}}} \dot{\widehat{\mathbf{w}}} = \mathbf{J}_p \dot{\widehat{\mathbf{w}}} \tag{6}$$

where, $\mathbf{J}_p = \frac{\partial \mathbf{f}(\widehat{\mathbf{w}}, \mathbf{x}_p)}{\partial \widehat{\mathbf{w}}}$ is the Jacobian matrix, whose elements are $J_{p,ij} = \partial y_{p,i}/\partial w_j$. The desired output is given by $\mathbf{y}_p^d = \mathbf{f}(\mathbf{w}, \mathbf{x}_p)$ and its derivative with respect to time is

$$\dot{\mathbf{y}}_p^d = \mathbf{J}_p \dot{\mathbf{w}} = 0. \tag{7}$$

The estimation error is $\mathbf{e}_p = \mathbf{y}_p^d - \mathbf{y}_p$ and its derivative w.r.t. time t is

$$\dot{\mathbf{e}}_p = \dot{\mathbf{y}}_p^d - \dot{\mathbf{y}}_p = -\mathbf{J}_p \dot{\widehat{\mathbf{w}}}, \text{ as } \dot{\mathbf{y}}_p^d = 0 \tag{8}$$

The optimization of the neural network weights is formulated as a control problem.

$$\dot{\mathbf{e}}_p = -\mathbf{J}_p \dot{\widehat{\mathbf{w}}} = -\mathbf{J}_p \mathbf{u} \tag{9}$$

The control input updates the weights as $\mathbf{u} = \dot{\widehat{\mathbf{w}}}$.

In the batch mode, all the patterns are learnt simultaneously. Hence, the dynamics can be written jointly as

$$\dot{\mathbf{e}} = -\mathbf{J}\mathbf{u} \tag{10}$$

where, $\mathbf{e} = [\mathbf{e}_1^\intercal, \mathbf{e}_2^\intercal, ..., \mathbf{e}_{N_p}^\intercal]^\intercal$, $\mathbf{J} = [\mathbf{J}_1^\intercal, \mathbf{J}_2^\intercal, ..., \mathbf{J}_{N_p}^\intercal]^\intercal$. Hence, $\mathbf{e}$ is an $N_l N_p \times 1$ vector, $\mathbf{J}$ is an $N_l N_p \times N_w$ matrix and $\mathbf{u}$ is an $N_w \times 1$ vector.

A. Optimal Weight Update

The cost function is defined over time interval $(t, T]$ as

$$V(\mathbf{e}(t)) = \int_t^T L(e(\tau), \mathbf{u}(\tau)) d\tau \tag{11}$$

$$\text{where, } L(\mathbf{e}, \mathbf{u}) = \frac{1}{2}(\mathbf{e}^\intercal \mathbf{e} + \mathbf{u}^\intercal \mathbf{R} \mathbf{u}) \tag{12}$$

with $\mathbf{R}$ as a constant $N_w \times N_w$ matrix. Here, t signifies the iterations to update $\mathbf{w}$. Our goal is to find an optimal weight update law $\mathbf{u}(t)$ which minimizes the aforementioned cost function. We can assume $T \to \infty$. We use the following formulation which is popularly known as the Hamilton-Jacobi-Bellman (HJB) equation [**?**].

$$\min_{\mathbf{u}} \left\{ \frac{dV^*}{d\mathbf{e}} \dot{\mathbf{e}}(t) + L(\mathbf{e}(t), \mathbf{u}(t)) \right\} = 0 \tag{13}$$

Putting the expressions for $\dot{\mathbf{e}}(t)$ and $L(\mathbf{e}(t), \mathbf{u}(t))$ from eqs. (10) and (12), respectively,

$$\min_{\mathbf{u}} \left\{ -\frac{dV^*}{d\mathbf{e}} \mathbf{J}\mathbf{u}(t) + \frac{1}{2}\mathbf{e}(t)^\intercal \mathbf{e}(t) + \frac{1}{2}\mathbf{u}(t)^\intercal \mathbf{R}\mathbf{u}(t) \right\} = 0 \tag{14}$$

Here, $\frac{dV^*}{d\mathbf{e}}$ is $1 \times N_l N_p$ vector. Differentiating with respect to $\mathbf{u}$, we get the optimal update law as

$$\mathbf{u}^*(t) = \mathbf{R}^{-1}\mathbf{J}^\intercal \left(\frac{dV^*}{d\mathbf{e}} \right)^\intercal \tag{15}$$

In order to find the expression for $\left(\frac{dV^*}{d\mathbf{e}} \right)$, we put the optimal $\mathbf{u}$ from eq. (15) in eq. (14),

$$\mathbf{e}(t)^\intercal \mathbf{e}(t) - \left(\frac{dV^*}{d\mathbf{e}} \right) \mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal \left(\frac{dV^*}{d\mathbf{e}} \right)^\intercal = 0 \tag{16}$$

A proper solution of eq. (16) should lead to $\mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal$ to be positive definite.

This is an under-determined system of equations. However, the optimal input must stabilize the system. The stability of the system can be analyzed with the help of a Lyapunov function defined as

$$\mathcal{V}(\mathbf{e}) = \frac{1}{2}\mathbf{e}^\intercal \mathbf{e} \tag{17}$$

The equilibrium point $\mathbf{e} = 0$ is stable if $\dot{\mathcal{V}}(\mathbf{e})$ is negative definite.

$$\dot{\mathcal{V}}(\mathbf{e}(t)) = \mathbf{e}^\intercal \dot{\mathbf{e}} \tag{18}$$

$$= -\mathbf{e}^\intercal \mathbf{J}\mathbf{u}^*(t) \tag{19}$$

$$= -\mathbf{e}^\intercal \mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal \left(\frac{dV^*}{d\mathbf{e}} \right)^\intercal \tag{20}$$

If one selects the following form of $dV^*/d\mathbf{e}$,

$$\frac{dV^*}{d\mathbf{e}} = \mathbf{e}(t)^\intercal \mathbf{C}(t)^\intercal \tag{21}$$

where, $\mathbf{C}(t)$ is chosen to be a positive definite matrix, then $\dot{\mathcal{V}}(\mathbf{e}(t))$ becomes negative definite. In order to find the expression for $\mathbf{C}(t)$, we substitute eq. (21) in eq. (16),

$$\mathbf{e}(t)^\intercal \left(\mathbf{I} - \mathbf{C}(t)^\intercal \mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal \mathbf{C}(t) \right) \mathbf{e}(t) = 0 \tag{22}$$

where, $\mathbf{I}$ is $N_p \times N_p$ identity matrix. For this to be true for all $\mathbf{e}(t)$,

$$\mathbf{C}(t)^\intercal \ \mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal \ \mathbf{C}(t) = \mathbf{I} \tag{23}$$

To find a solution for $\mathbf{C}(t)$, we decompose $\mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\intercal$ into eigenvectors $\mathbf{U}$ and a diagonal matrix of eigenvalues $\mathbf{\Sigma}$. Since, $\mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal$ is a symmetric positive definite matrix, $\mathbf{U}^\intercal\mathbf{U}^{=}\mathbf{U}\mathbf{U}^\intercal = \mathbf{I}$ and all eigenvalues are positive. Now, $\mathbf{C}(t)$ can assume the following form to satisfy eq. (23)

$$\mathbf{C}(t) = \mathbf{U}\mathbf{\Sigma}^{-\frac{1}{2}}\mathbf{U}^\intercal \tag{24}$$

This form assures $\mathbf{C}(t)$ to be positive definite, so that the system is stable around $\mathbf{e} = 0$. However, for numerical stability while implementation, a small positive term ($= 10^{-4}\mathbf{I}$) is added to Σ so as to avoid numerical instability.

Finally, we get the optimal weight update law by combining equations (15) and (21) as

$$\dot{\hat{\mathbf{w}}} = \mathbf{u}^*(t) = \mathbf{R}^{-1}\mathbf{J}^\intercal\mathbf{C}(t)\mathbf{e}(t) \tag{25}$$

where, $\mathbf{C}(t)$ is given by eq. (24).

For the FFNN considered in eqs. (2)-(4), the Jacobian can be obtained as follows. For the output layer,

$$\frac{\partial f(\mathbf{w}, \mathbf{x}_p)}{\partial w_{i_2 i_1}} = y_{p,i_2}(1 - y_{p,i_2})v_{i_1} \tag{26}$$

and for the hidden layer,

$$\frac{\partial f(\mathbf{w}, \mathbf{x}_p)}{\partial w_{i_1 i_0}} = \sum_{i_2} y_{p,i_2}(1 - y_{p,i_2})w_{i_2 i_1} v_{i_1}(1 - v_{i_1})x_{p,i_0} \tag{27}$$

B. Levenberg-Marquardt Modification

The LM modification is applied to the optimal HJB based learning scheme, described above, by changing Eq. (24) as

$$\mathbf{C}(t) = \mathbf{U}\mathbf{\Sigma}_\mu^{-\frac{1}{2}}\mathbf{U}^\intercal \tag{28}$$

where, $\mathbf{\Sigma}_\mu$ is a diagonal matrix of the eigenvalues of $\mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal + \mu\mathbf{I}$, and $\mathbf{U}$ consists of columns as the corresponding eigenvectors. Here, $\mathbf{I}$ is the identity matrix and μ is the LM parameter. When a weight update $\mathbf{u}$ leads to an increase in error $\mathbf{e}^\intercal\mathbf{e}$, μ is increased by a fixed factor β and a new $\mathbf{u}$ is computed. This step is repeated until the error decreases and the final $\mathbf{u}$ is used to update the FFNN weights. On the other hand, when a $\mathbf{u}$ leads to decrease in error, it is used to update the FFNN weights and μ is divided by β.

III. HJB based Online Learning for FFNN

In the online mode, the weight vector is updated with each input pattern. In this case, the network dynamics can be presented as

$$\mathbf{y} = \mathbf{f}(\widehat{\mathbf{w}}) \tag{29}$$

where the network output is simply observed as a function of network weights only. Here the desired output $\mathbf{y}^d$ is observed and the network response is compared to find

$$\mathbf{e} = \mathbf{y}^d - \mathbf{y}. \tag{30}$$

Thus the network dynamics can be derived following the approach given in Sec. II.

The problem is to find $\mathbf{u}$ so as to minimize

$$V(\mathbf{e}(t)) = \int_t^\infty \frac{1}{2}(\mathbf{e}^\intercal\mathbf{e} + \mathbf{u}^\intercal\mathbf{R}\mathbf{u})d\tau \tag{31}$$

One should note that this cost function is not pertaining to any specific pattern rather the network is subjected to various inputs while the instantaneous error $\mathbf{e}$ is computed according to Eq. (30).

The optimal instantaneous weight update law, as derived using the HJB equation and Lyapunov stability criterion, is given by

$$\dot{\hat{\mathbf{w}}} = \mathbf{u}^*(t) = \mathbf{R}^{-1}\mathbf{J}^\intercal\mathbf{C}(t)\mathbf{e}(t) \tag{32}$$

$$\text{where, } \mathbf{C}(t) = \mathbf{U}\mathbf{\Sigma}^{-\frac{1}{2}}\mathbf{U}^\intercal \tag{33}$$

with $\mathbf{U}$ and $\mathbf{\Sigma}$ as the eigenvectors and eigenvalues of $\mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal$.

A. Single output Network with Online Learning

Let us consider a special case of single output network trained in an online fashion. For such a case $\mathbf{J}$ is a $1 \times N_w$ matrix, therefore the term $\mathbf{J}\mathbf{J}^\intercal$ is a scalar. Assuming $\mathbf{R} = r\mathbf{I}$, the corresponding HJB equation can be written as

$$e^2 - \left(\frac{dV^*}{de}\right)\frac{1}{r}\mathbf{J}\mathbf{J}^\intercal\left(\frac{dV^*}{de}\right)^\intercal = 0 \tag{34}$$

$$\frac{dV^*}{de} = e\sqrt{\frac{r}{\mathbf{J}\mathbf{J}^\intercal}}$$

where, $e = y_d - y$. Therefore, the optimal learning algorithm becomes

$$\mathbf{u}^* = \frac{1}{(r\mathbf{J}\mathbf{J}^\intercal)^{1/2}}\mathbf{J}^\intercal e \tag{35}$$

IV. Experiments and Discussion

The efficiency of the proposed HJB based optimal learning scheme is substantiated with the help of several benchmark learning problems. Notably, the purpose of the presented work is to optimize a given cost function and not explicitly to design robust classification algorithms. Hence, the following experiments aim at optimising the FFNN for a given cost function. Nevertheless, in order to show the generalizability for classification, several experiments have also been included with separate training and testing data over real datasets (Sec. IV-D, Sec. IV-E).

The proposed HJB algorithm has been compared with popular learning schemes, viz., Adadelta, Adagrad, Adam, BP, LF and LM. Adadelta, Adagrad and Adam algorithms have been implemented using Keras library [**?**]. BP is the standard back-propagation learning scheme with a constant learning rate η. LF scheme is a Lyapunov function based learning scheme developed by Behera *et al.*[**?**] and has one tunable parameter μ. LM algorithm is an offline algorithm with two tunable parameters - μ, β. In the proposed offline as well as online HJB algorithms, $\mathbf{R}$ has been set to $r\mathbf{I}$, where $\mathbf{I}$ is an identity matrix and $r > 0$ is a constant tunable parameter. The *success score* is defined as the number of trials

TABLE I
AVERAGE NUMBER OF EPOCHS (WITH AVERAGE RUN TIME) FOR SUCCESSFUL CONVERGENCE FOR MODULO-2 FUNCTION LEARNING

(a) Offline algorithms

FFNN architecture	BP $\eta = 1$	LF $\mu = 0.05$	HJB $r = 0.1$
3-4-1	3047 (7.6 s)	1087 (9.5 s)	3519 (6.0 s)
3-6-1	2880 (4.2 s)	2801 (9.9 s)	190 (0.4 s)
3-8-1	1309 (2.4 s)	480 (10.4 s)	134 (0.5 s)
3-10-1	1046 (1.4 s)	447 (0.7 s)	54 (0.09 s)
3-15-1	2544 (3.5 s)	670 (0.9 s)	49 (0.07 s)

FFNN architecture	LM $\mu = 10^{-2}, \beta = 10$	HJB-LM $\mu = 10^{-2}, \beta = 10$
3-4-1	266 (7.0 s)	63 (0.3 s)
3-6-1	37 (0.15 s)	25 (0.05 s)
3-8-1	22 (0.06 s)	26 (0.06 s)
3-10-1	18 (0.05 s)	22 (0.05 s)
3-15-1	17 (0.05 s)	25 (0.05 s)

(b) Online algorithms

FFNN architecture	BP $\eta = 1$	LF $\mu = 0.2$	HJB $r = 5$
3-4-1	-	5843 (10.9 s)	-
3-6-1	6210 (11.0 s)	2429 (4.6 s)	2939 (5.8 s)
3-8-1	6773 (15.0 s)	3565 (8.8 s)	1722 (4.3 s)
3-10-1	6123 (10.8 s)	3919 (7.3 s)	762 (1.5 s)
3-15-1	3532 (6.4 s)	4922 (9.5 s)	731 (1.5 s)

TABLE II
AVERAGE SUCCESS SCORE, IN %, FOR MODULO-2 FUNCTION LEARNING

FFNN	Offline			Online			Offline	
architecture	BP	LF	HJB	BP	LF	HJB	LM	HJB-LM
3-4-1	24	24	28	0	20	0	84	84
3-6-1	80	88	96	36	68	96	96	100
3-8-1	84	92	100	56	72	100	100	100
3-10-1	90	96	100	80	76	100	100	100
3-15-1	100	100	100	100	80	100	100	100

which successfully converge to zero output error over the total number of trials.

$$\text{Success score} = \frac{\text{No. of successfully converging trials}}{\text{Total no. of trials}} \quad (36)$$

All the algorithms have been implemented in MATLAB R2013a on an Intel(R) Xeon(R) 2.40GHz CPU with 6GB RAM. Each experiment consists of multiple trials, each starting from a random initial point. For a fair comparison, in a trial, same initial point is chosen for all the learning schemes.

A. Modulo-2 function

The first example is based on the modulo-2 function, $f : \mathbb{R}^2 \to \mathbb{R}$. Here, 9 patterns are formed using $x_1 \in \{0, 1, 2\}$ and $x_2 \in \{0, 1, 2\}$. The desired output is given by

$$y = f(x_1, x_2) = \begin{cases} 1 & \text{if } (x_1 + x_2) \text{ is odd} \\ 0 & \text{if } (x_1 + x_2) \text{ is even} \end{cases} \quad (37)$$

The FFNN is learned using various algorithms in offline as well as online modes. The simulation is performed for 25 different trials. The results are tabulated in Tables I and II, along with the parameter values used. Table I presents the average number of epochs as well as average run time for successful convergence of each algorithm. Table II shows the success score, as defined in eq. (36), for each algorithm.

It can be observed from Table II that, for all these algorithms, achieving 100% success rate with lesser number of hidden neurons is difficult, but it improves with the increase in the number of hidden neurons. In the case of batch (offline) learning, the success score of the proposed HJB algorithm is much higher than both BP and LF algorithms. The proposed HJB algorithm achieves more than 95% success score with 6 or more hidden neurons. The offline LM algorithm is able to achieve a high success score even with 4 hidden neurons. Nevertheless, the HJB-LM algorithm performs even better. It achieves 100% success score with 6 or more hidden neurons. On the other hand, learning in the online mode is obviously more difficult, as shown by lower success scores achieved by the online BP, LF and HJB schemes. With 4 hidden neurons, BP and HJB schemes were found unable to converge to global minimum. Still, HJB scheme achieves more than 95% success score with 6 or more hidden neurons. This implies high probability for convergence to the global minimum.

The corresponding average rates of convergence, can be seen in Table I. In offline mode, the proposed HJB algorithm converges much faster than both BP and LF algorithms. The number of epochs required for HJB scheme to converge reduces with the increase in the number of hidden neurons, and accordingly, the training time goes down. However, the convergence time improves drastically by introducing the LM modification, as shown by the results of HJB-LM scheme. On the other hand, for online learning, the number of epochs is higher than that for offline learning. Still, the proposed HJB

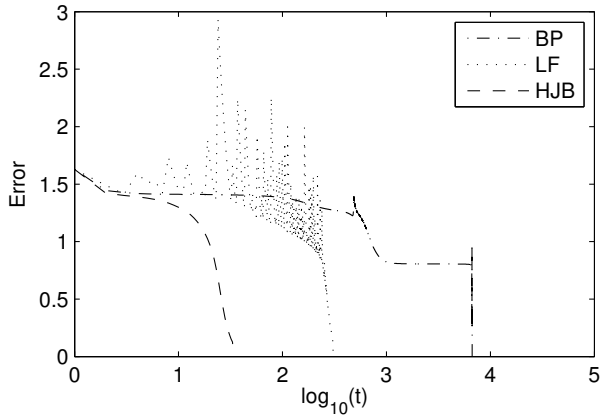

Fig. 2. Comparison of convergence rates of different algorithms for Modulo-2 function in offline learning. Here, t refers to the number of epochs.

algorithm converges significantly faster as compared to BP and LF algorithms.

The rate of convergence for one trial each of BP, LF and HJB schemes has been shown in Fig. 2. Here, the errors drop most quickly with the proposed HJB algorithm, followed by LF and BP, respectively. The improvement in the convergence rate with the proposed HJB scheme over that of the other schemes is quite significant, differing by orders of magnitude.

B. 8-bit Parity

The 8 dimensional parity problem is also like the XOR problem where the output is 1 if the number of ones in the input is odd. In this problem, 0 and 1 are replaced by 0.1 and 0.9 respectively. The network architecture has one bias (+1) in the input. The simulations are performed for 20 trials with different random initialization points. Table III shows the simulation results for different algorithms with 9-30-1 FFNN. Both BP and LF based algorithms fail to converge in all the 20 trials. However, the HJB based algorithm converges within 845 epochs with 95% success score. Interestingly, if one uses HJB LM algorithm, the convergence is achieved within 152 epochs with 100% success score. However, it can be seen that LM scheme performs better than the HJB-LM scheme. Nevertheless, the convergence rate of LM degrades with the increase in the network size. This can be seen in Table IV, where the HJB-LM scheme converges faster than even LM for a 9-50-1 architecture. Table IV compares the epoch-wise classification error for Adadelta, Adagrad, Adam, BP and LF algorithms along with the proposed HJB algorithm. With the increase in the size of network, accuracy has improved for Adagrad, Adadelta and Adam algorithms.

C. 2D-Gabor Function

In this simulation, Gabor function is used for system identification problem which is represented by the following

TABLE III
8 BIT PARITY: 9-30-1 ARCHITECTURE

Algorithm	Avg Epochs	Success rate	Parameters
Adadelta	2500	97%	$\eta = 0.94$
Adagrad	2800	96%	$\eta = 0.12$
Adam	1500	99.6%	$\eta = 0.001$
BP Offline	-	0%	$\eta = 0.05$
LF Offline	-	0%	$\mu = 0.10$
HJB Offline	845	95%	$r = 5$
LM	130	100%	$\mu = 0.001, \beta = 10$
HJB LM	152	100 %	$\mu = 0.001, \beta = 10$

TABLE IV
8 BIT PARITY PROBLEM : 9-50-1

Algorithm	Avg Epochs	Success rate	Parameters
Adadelta	1500	98.5%	$\eta = 0.94$
Adagrad	1500	97%	$\eta = 0.12$
Adam	1000	99.8%	$\eta = 0.001$
HJB Offline	465	100%	$r = 5$
LM	129	100%	$\mu = 0.001, \beta = 10$
HJB LM	97	100 %	$\mu = 0.001, \beta = 10$

TABLE V
2D GABOR FUNCTION: 3-6-1 ARCHITECTURE

Algorithm (Online)	Avg RMS error (200 epochs)	(400 epochs)	(2000 epochs)	Parameters
Adadelta	0.1123	0.0815	0.0416	$\eta = 0.98$
Adagrad	0.0525	0.0412	0.0612	$\eta = 0.01$
Adam	0.0314	0.0443	0.043	$\eta = 0.001$
BP	0.1439	0.1404	0.0311	$\eta = 0.50$
BP	0.1222	0.0615	0.0313	$\eta = 0.95$
LF	0.0703	0.0354	0.0305	$\mu = 0.35$
HJB	0.0488	0.0326	0.0310	$r = 1.5$
HJB	0.0479	0.0369	0.0360	$r = 1$

equation.

$$g(x_1, x_2) = \frac{1}{2\pi(0.5)^2} e^{-(x_1^2 - x_2^2)/(2(0.5)^2)} \times \cos(2\pi(x_1 + x_2)) \tag{38}$$

For this problem, 100 test data are randomly generated in the range [0, 1]. The network architecture used in this problem is 3-6-1, with one bias (+1) input. The simulation is run for online learning till the rms error is 0.01 or a maximum of 2000 epochs. Table V shows the rms error averaged over 20 runs for each method, after 200, 400 and 2000 epochs of learning. One can observe that the error decay rate is the fastest for the proposed HJB algorithm as compared to that of the other algorithms. Figure 3 compares the epoch-wise error decay for Adadelta, Adagrad, Adam, BP and LF algorithms along with the proposed HJB algorithm.

D. Breast Cancer Data

Breast Cancer problem [**?**] is a classification problem where the neural networks is fed with 9 different inputs representing different medical attributes while the output represents class of cancer. The data set has 699 data points of which 600 have been used for training and the remaining ones for testing. The misclassification rate is calculated as the percentage of test points wrongly classified by the algorithm. For this problem,

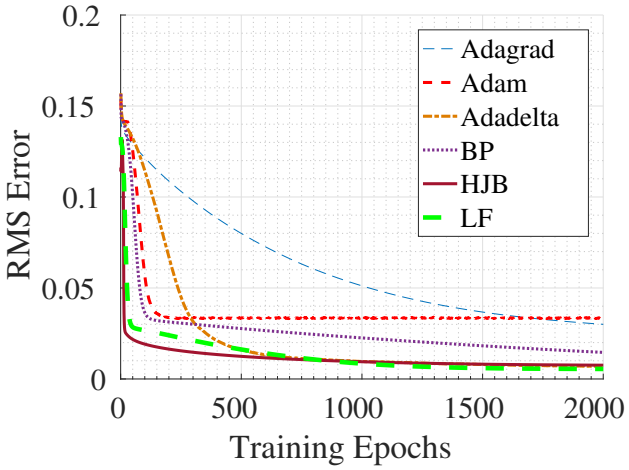

Fig. 3. Average RMS error as a function of number of training epochs for 2-D Gabor function using various algorithms

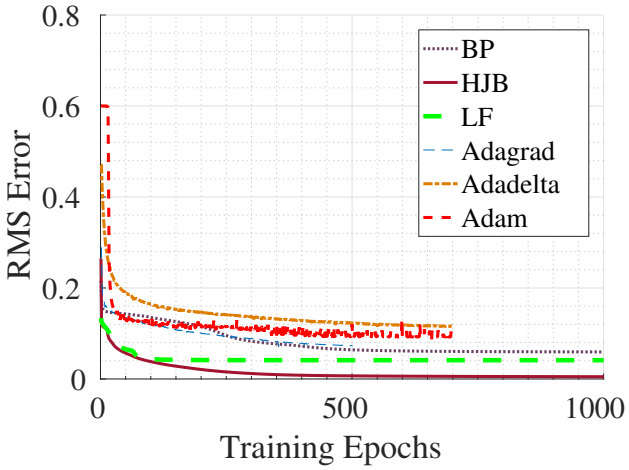

Fig. 4. Comparison of error decay for various algorithms while training for Breast Cancer problem.

TABLE VI
BREAST CANCER: 10-15-1 ARCHITECTURE

Algorithm	Avg. Epochs	Misclassification rate	Parameters
Adadelta	700	15.47	$\eta = 0.94$
Adagrad	500	14.34	$\eta = 0.06$
Adam	700	7.64	$\eta = 0.001$
BP Online	6000*	2.95	$\eta = 0.20$
LF Online	3058	2.85	$\mu = 0.20$
HJB Online	1020	2.05	$r = 2.5$

* Error does not fall below the set threshold while training

a network with 10-15-1 architecture, having one bias (+1) node in the input layer, is trained in an online way. The simulations are performed for 20 different trials. Table VI shows the simulation results. Note that BP algorithm is never converging to 0.01 rms error within 6000 iterations. Fig. 4 shows the decay of error with successive epochs for Adadelta, Adagrad, Adam, BP and LF algorithms along with the proposed HJB algorithm. The proposed HJB algorithm shows the best performance during training.

E. Credit Approval Data Set

Credit approval [**?**] is also classification problem with 14 inputs and a binary output. The output represents the approval or rejection of credit card. The data set has 690 data points of which 80% i.e. 552 points are used for training and the rest are used for testing. For this problem, we have selected a network with 15-25-1 architecture, with one bias (+1) node in input layer. The training is carried out in online fashion for 20 different trials, starting from different initializations of network weights. Table VII shows the simulation results. Since none of the methods converged to the rms error of 0.01, the training is stopped after 2000 epochs. We have also provided the average rms error after 1000 epochs for each algorithm. We can see that the HJB based algorithm reaches to a better rms error within 1000 epochs as compared to the error for BP and LF based algorithm. Also, note that the BP algorithm does not make much improvement from 1000 to next 2000 epochs.

TABLE VII
CREDIT APPROVAL DATA: 15-25-1 ARCHITECTURE

Algorithm	Avg RMS (2000 epochs)	Avg RMS error (1000 epochs)	Parameters
BP Online	0.1826	0.1991	$\eta = 0.20$
LF Online	0.1379	0.1810	$\mu = 0.20$
HJB Online	0.0776	0.1250	$r = 2.5$

F. Parameter Tuning

The proposed HJB offline and online algorithms have one tunable parameter, viz., r. The value of this parameter plays a crucial role in determining the convergence properties, as shown in Fig. 5. Notably, BP and LF algorithms also have

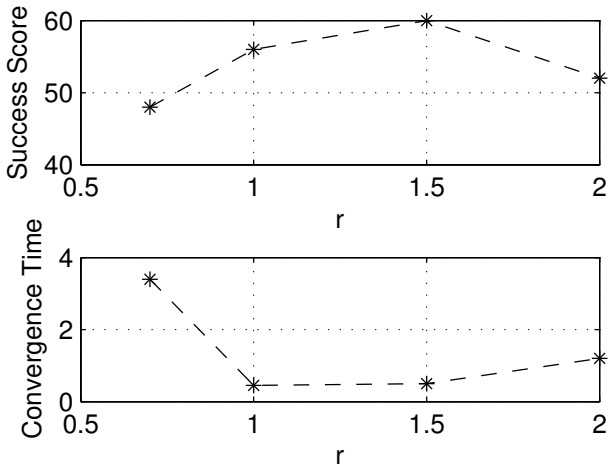

Fig. 5. Effect of varying the parameter r on the success score and convergence time of HJB offline algorithm for modulo-2 function with 3-4-1 architecture

TABLE VIII
WEIGHT UPDATE LAWS OF VARIOUS ALGORITHMS

Algorithm	Update law, $\dot{\hat{\mathbf{w}}} = \mathbf{u}$
BP	$\mathbf{u} = \eta \mathbf{J}^\intercal \mathbf{e}$
LF	$\mathbf{u} = \mu \frac{\|\|\mathbf{e}\|\|^2}{\|\|\mathbf{J}^\intercal \mathbf{e}\|\|^2} \mathbf{J}^\intercal \mathbf{e}$
LM	$\mathbf{u} = [\mathbf{J}^\intercal \mathbf{J} + \mu \mathbf{I}]^{-1} \mathbf{J}^\intercal \mathbf{e}$
HJB	$\mathbf{u} = \frac{1}{r} \mathbf{J}^\intercal \mathbf{C} \mathbf{e}$
single output online HJB	$\mathbf{u} = \frac{1}{(r \mathbf{J}\mathbf{J}^\intercal)^{1/2}} \mathbf{J}^\intercal e$

one parameter each, i.e., η and μ, respectively. On the other hand, LM and the proposed HJB-LM algorithms have two parameters each.

Derivation of the optimal values of the parameters has not been dealt with in this work. For the experiments, the parameters have been set manually.

V. CONVERGENCE BEHAVIOR

This section analyzes the convergence behavior of various algorithms used in this paper. The iterative update laws of these algorithms are summarized in Table VIII.

A. Computation time and complexity

It has been observed that the proposed HJB algorithm is much faster than the other algorithms for both online as well as offline learning. It will be useful at this stage to compare the computational complexity of one iteration of each of these algorithms. All the algorithms discussed in this paper are based on deterministic iterative update of weights. Moreover, all of these involve the calculation of the Jacobian $\mathbf{J}$. They mostly differ in their weight update schemes, $\mathbf{u}$.

Certainly, a single iteration of HJB and LM schemes involves more computations than BP and LF schemes. However, the number of iterations required for HJB and LM are significantly small as compared to those for BP and LM. Hence, the former ones take much less convergence time than the later ones.

The Jacobian $\mathbf{J}$ is an $N_l N_p \times N_w$ matrix. HJB involves calculation of the eigenvalues of $\mathbf{J}\mathbf{J}^\intercal$ ($N_l N_p \times N_l N_p$), while LM involves calculation of the inverse of $\mathbf{J}^\intercal \mathbf{J}$ ($N_w \times N_w$). Hence, in the offline mode, where N_p is large, one iteration of LM is supposed to take lesser time than HJB. Nevertheless, the number of iterations required for HJB is lesser than that for LM, leading to reduction in the computation time for the former. Moreover, as the size of network, N_w, increases, the performance of LM deteriorates quickly; but since $\mathbf{J}\mathbf{J}^\intercal$ is independent of N_w, HJB is not affected drastically. On the other hand, in online mode, $N_p = 1$, and hence, HJB scheme becomes even faster.

VI. CONCLUSION

HJB equation is well known for dynamic optimization. In this paper, the weight update process in FFNN has been formulated as a dynamic optimization problem. Applying HJB equations, the optimal weight update laws have been derived in both batch and online (instantaneous) modes. This paper analytically analyzes the convergence behavior of various popular learning algorithms as why they are liable to be stuck in local minima. It is shown that the Lyapunov function drags the solution to the local minimum.

The proposed scheme ensures faster convergence rate as compared to the existing schemes, including LM. Moreover, the weight update law using HJB equation has been derived for both offline as well as online modes, whereas LM can be applied only in the offline mode. Our study also shows that the proposed HJB scheme works well even with large network size, while the LM method deteriorates in performance as the size of the network increases, as also observed by Xie *et al* [**?**].