Comparison of TDNN Training Algorithms in Brain Machine Interfaces

Yiwen Wang, Sung-Phil Kim, Jose C. Principe
Computational NeuroEngineering Lab, University of Florida, Gainesville, FL 32611, USA

Abstract – Linear or non-linear models are used in Brain Machine Interfaces (BMIs) to map the neural activity to the associated behavior, typically the primate’s hand position. Linear models assume a linear relationship between neural activity and hand position that may not be the case. A solution would be time-delay neural network (TDNN) that provides effectively a nonlinear combination of linear models. However, this model results in a drastic increase of free parameters and slow convergence when trained by an error back-propagation learning rule. We propose to train the TDNN by scaled conjugate gradient, which avoids time-consuming linear search, coupled with weight decay to reduce the free parameters number and produce generally faster convergence.

I. INTRODUCTION

Brain Machine Interface (BMI) research groups have utilized diverse linear and nonlinear modeling frameworks [1-8] to decode neural activities and control a prosthetic device to perform intended movements. The inputs to these models are usually multichannel neuronal recordings collected from selected regions of a primate’s brain. Although the simple training strategies in the linear models are attractive, the limitation of linear mappings may hinder generalization capabilities. Furthermore, linear models suffer from an explosion in the number of parameters and the proper selection of inputs in terms of cost function.

Time-Delay Neural Network (TDNN) will replace the linear models in this work due to its ability to approximate arbitrary nonlinear mapping in functional spaces. This topology is more powerful than the linear finite impulse response (FIR) filters because each hidden processing element (PE) output can be a nonlinear adaptive basis for the output space utilized by projection of the high dimensional input data. These nonlinear projections can be linearly combined to form the outputs. The TDNN architecture consists of the delayed versions of the firing counts, which effectively implements a short-term memory mechanism. However, additional delays in the already high dimensional neuronal input (hundred of neurons) bring in a huge number of free parameters. If such a large TDNN is trained by simple error back-propagation algorithm, the convergence of parameters typically need long iterations, since the error back-propagation algorithm only adjusts the weights in the steepest descent direction, and it is also sensitive to the learning rate. Hence, we propose to utilize the scaled conjugate gradient algorithm [10] to determine the step length of the weight update along the conjugate gradient direction, which simultaneously avoids the time-consuming line search, to get a faster convergence. We also implement a sparse gradient algorithm [9] to derive a sparse weight matrix to decrease model complexity. The algorithm will be described in the next section followed by the design of TDNN in BMIs in section III.

II. THE LEARNING ALGORITHMS OF TDNN

In this section, we review three different algorithms used to train TDNN: error back-propagation learning rule, scaled conjugate gradient, and sparse gradient algorithm coupled with scaled conjugate gradient.

A. The Error Back-propagation Learning Rule

The weights of time-delay neural network can be trained using error back-propagation with Mean Square Error (MSE) criterion, which updates the network weights and biases in the negative direction of the gradient. Let us denote the weight vector as \( \mathbf{w} = [w_1, w_2, ..., w_L]^T \). Consider the cost function,

\[
J(\mathbf{w}) = E(e_k^2),
\]

where, \( E(\cdot) \) is the mean operator. The error \( e_k \) is the difference between the desired signal \( d_k \) and model output \( y_k = f(\mathbf{w}^T \mathbf{x}_k) \) at a time instance \( k \). The local gradients of the cost with respect to weight is given by,

\[
\mathbf{g}_k = -\nabla J(\mathbf{w}_k),
\]

where \( \mathbf{g}_k \) is the \( k \)th negative of gradient vector in term of performance function.

The update of the weight vector \( \mathbf{w} \) in error back-propagation is given by,

\[
\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha \cdot \mathbf{g}_k
\]

where \( \alpha \) is the constant learning rate that should be chosen carefully. If the learning rate is made too large, the algorithm becomes unstable. If it is set too small, the algorithm will need long iterations to converge weights.

B. Scaled Conjugate Gradient Algorithm
The error back-propagation only performs along the negative of the gradient, which does not necessarily produce the fastest convergence. And it is also very sensitive to the chosen learning rate, which may cause an unstable result or a long-time convergence. A conjugate gradient algorithm updates weights along the conjugate gradient direction and results in a generally faster convergence to reach the minimum of the cost function when compared with the steepest descent direction. However, a line search is required to determine the optimal distance to move along the current search direction. Then the next search direction is determined so that it is conjugate to previous search directions. The scaled conjugate gradient algorithm, developed by Moller [10], combines the model-trust region approach with the conjugate gradient algorithm, and successfully avoids the time-consuming line search.

\[
\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{h}_k
\]  

\[
\alpha_k = \frac{g_k^T \cdot h_k}{h_k^T (J'(\mathbf{w}_k + \sigma_k \mathbf{h}_k) - J'(\mathbf{w}_k)) + \xi_k |h_k|^2}
\]  

\[
\mathbf{h}_{k+1} = \mathbf{g}_{k+1} + \gamma_k \mathbf{h}_k \quad (h_0 = \mathbf{g}_0)
\]  

\[
\gamma_k = \frac{|\mathbf{g}_{k+1}|^2 - \mathbf{g}_{k+1}^T \cdot \mathbf{g}_k}{\mathbf{g}_k^T \cdot \mathbf{h}_k}
\]

where \( \mathbf{g}_k \) is the \( k \)th negative of gradient vector derived from (2). \( \sigma_k \) is the change in weight for second derivative approximation. \( \alpha_k \) is the learning step determined by conjugate gradient described in (5), (6) and (7). \( \mathbf{h}_k \) is a conjugate direction, along which the algorithm updates the weight vector in \( L \) (the number of all weights and biases) steps to reach the minimum of the cost function. Otherwise, the algorithm will be restarted by initializing \( \mathbf{h}_{k+1} \) to the current steepest descent direction \( \mathbf{g}_{k+1} \). The scalar \( \xi_k \) in (5) will be raised when the denominator of the learning step \( \alpha_k \) is less than zero, which guarantees that the scaled conjugate gradient still obeys the positive definiteness of the Hessian matrix without implementing the line search.

C. Combining a sparse algorithm into scaled conjugate gradient algorithm

A sparse gradient algorithm has been proposed in [9] to modify the cost function as the sum of the MSE and a penalty term of the weight vector with an adaptive regularization parameter, which shows the asymptotic convergence of the penalty sequence to zero. The cost function is given by,

\[
J(\mathbf{w}, \lambda) = E(e^2) + \beta \lambda \left[ \frac{1}{L} \sum_{i=1}^{L} |w_i|^p - \theta \right] - \beta \lambda^2
\]  

where \( \theta \) is a constant fixing the sum of the \( p \)th powers of the absolute values of the individual elements \( w_i \) of the weight vector \( \mathbf{w} \). We can constrain the \( p \) norm of the weight vector \( \mathbf{w} \), which will play different roles on cost function convergence. \( \beta > 0 \) is a positive stabilization constant that affects the convergence of the adaptive penalty factor \( \lambda \). It also balances the constraint term against the MSE. Then the stochastic gradients of the cost are,

\[
\frac{\partial t(\mathbf{w}, \lambda)}{\partial w_p} = -2e_k \cdot f'(w_i)x_{ki} + \beta \lambda \left[ pw_i^{p-1} \text{sign}(w_i^{p-1}) / L \right]
\]  

\[
\frac{\partial t(\mathbf{w}, \lambda)}{\partial \lambda} = \beta \left[ \frac{1}{L} \sum_{i=1}^{L} |w_i|^p - \theta \right] - 2 \beta \lambda
\]

where \( w_{ki} \) denotes the \( i \)th elements in the vector \( \mathbf{w}_k \) and \( x_{ki} \) denotes the \( i \)th elements in the vector \( \mathbf{x}_k \).

Our contribution is to integrate this constraint into the scaled conjugate formulation. Here we minimize (8) with respect to \( \mathbf{w} \) using scaled conjugate gradient algorithm given by (4~7), where \( \mathbf{g}_k \) is the \( k \)th negative of gradient vector derived from (2) and (9). The update of adaptive penalty factor \( \lambda \) is given by

\[
\lambda_{k+1} = \lambda_k + \eta \frac{\text{sgn}(\lambda_k - \theta) - 2 \beta \lambda_k}{\lambda_k}
\]  

where \( \eta \) is small positive step-size with a condition \( 0 < 2 \beta \eta \ll 1 \) to ensure \( \lambda \) asymptotically converging to

\[
\lambda^* = \frac{1}{2} \sum_{i=1}^{L} |w_i|^p - \theta \xrightarrow{k \to \infty} 0.
\]

Lagrange \( * = \frac{1}{2} \sum_{i=1}^{L} |w_i|^p - \theta \xrightarrow{k \to \infty} 0.\)

The sparse conjugate gradient algorithm estimates both a sparse weight matrix and decreases the computation complexity in TDNN training. Therefore we expect that our combination of the two algorithms will produce faster convergence and better control of generalization of the TDNN.

III. BRAIN MACHINE INTERFACES MODEL DESIGN

Dr. Nicolelis primate laboratory at Duke University provided the neural recordings in the modeling analysis presented here. An adult, female monkey was instrumented with high-density microelectrode arrays [11], implanted stereotaxically in seven cortical neural structures that are involved in the controlling fine arm and hand movements. The task involves the presentation of a randomly placed target on a computer monitor in front of the monkey. The monkey used a hand-held joystick to move the computer cursor so that it intersects the target. The hand position was recorded in real time along with the corresponding neural activity from
multiple channels [8]. 192 neurons were recorded; the neuronal firing times of all sorted spikes were binned in non-overlapping windows of 100ms and the behavior datasets were downsampled and low-pass filtered to 10 Hz. This data set was segmented into three exclusive parts: 9,000 × 192 samples for model training, 1,000 × 192 samples for model validation and 3,000 × 192 samples for model testing.

A. Design of Time-delay Neural Network

The topology of TDNN is shown in fig.1. The input vector includes 10 most recent spike counts from 192 neuron channels. 20 hyperbolic tangent nonlinearity PEs in the hidden layer produced the best results. The output is normalized into the range of [-1,1], accommodating different types of output PEs. The input layer's weights and biases are initialized according to the Nguyen-Widrow initialization algorithm [12].

![Figure 1. The topology of TDNN in BMI model design](image)

B. Parameters Settings

We trained TDNN by all three algorithms described in section II. In the error back-propagation learning rule, we set the constant learning rate \( \alpha = 0.01 \). In the scaled conjugate gradient algorithm, we set \( \sigma_k = 5 \times 10^{-5} \) and \( \xi_k = 5 \times 10^{-7} \). In the sparse gradient algorithm, considering a separate constrain for each coordinate of hand positions, we chose \( \theta_x = 0.002 \) and \( \theta_y = 0.001 \) for x-, and y-coordinate, respectively. The constraint parameter \( \rho \) was set to 2. The step-size for the Lagrange (11) updates was chosen to be 0.001 and \( \beta \) was set to 1.5. All the trainings were stopped when the cross validation error continuously increased for more than 5 steps.

C. Experiment and Comparison Results

Fig.2 shows the performance of the models trained by the sparse algorithm coupled with scaled conjugate gradient on test data. Only 250 samples out of 3000 are shown here for clarity. The model output (denoted by solid lines) can track the x- and y-coordinate of hand position trajectory (dotted lines) most of the time, but cannot catch the peaks well, which is probably because the \( \rho \) norm of the weights smoothes the output.

![Figure 2. Model output and the desired output](image)

We compared the three algorithms in 10 runs with different initial weights. Table I shows the comparison results of the training epochs, the average MSE, and the maximum correlation coefficient (CC) computed between the desired trajectory and the model output.

| TABEL I | COMPARASION RESULTS OF THREE ALGORITHMS |
|---------------------|---------------------|---------------------|---------------------|
|                  | Back-propagation | Scaled Conjugate Gradient | Sparse Algorithm & Scaled Conjugate Gradient |
| Epoch Mean       | 358.1             | 64.4                 | 56.2                 |
| Epoch Variance   | 288.2             | 17.9                 | 11.5                 |
| MSE              | 0.0728            | 0.0531               | 0.0533               |
| CC               | 0.4970            | 0.4654               |                      |

From Table I, we can see that, among the three algorithms, error back-propagation obtained the largest MSE than the other two algorithms. And it also took much more epochs to converge with larger variance, which indicated the lack of robustness. The sparse gradient algorithm coupled with the scaled conjugate gradient algorithm has shown both faster convergences along with a smaller variance among runs when compared with the scaled conjugate gradient. However, the proposed algorithm performed at the same MSE and CC level as scaled conjugate gradient. Probably this result is due to the few training steps that are insufficient for the weight decay to work. The \( L_2 \) norm penalty (\( \rho = 2 \)) in the sparse gradient algorithm is also not the
most appropriate. The $L_1$ norm penalty of the weight plays more important role than $L_2$ when error is small.

IV. CONCLUSIONS

Time-delay neural networks provide a nonlinear mapping to translate brain activity into prediction of hand movements. However, the application of TDNN in BMIs yields too many free parameters. Training the weights by error back-propagation learning with constant step size easily results in a long convergence time and poor generalization. Scaled conjugate gradient performs faster along the conjugate direction without time-consuming linear search. Sparse gradient can simplify the weight matrix and improve the TDNN generalization by modifying a penalty term of the weight vector with an adaptive regularization parameter. This paper presents the combination of sparse gradient algorithm and scaled conjugate gradient for BMI modeling. However, we do not obtain better generalization performance in this experiment. A TDNN with 20 hidden PEs and 10 time delays of 192 channels of neuronal signals produces more than 35,000 free parameters, and we only have $9,000 \times 192$ samples for training and cross-validation. We believe that if enough data are provided to train a $L_1$ norm penalty ($p=1$) on the weights, we will get better generalization performance in the future. Another issue that must be better controlled is the speed of the adaptation with the weight decay, because our results suggest that there is not enough time during the fast adaptation to let the weight decay take effect.

ACKNOWLEDGEMENTS

This work was supported by DARPA sponsored grant no. ONR-450595112. We thank Dr. Miguel Nicolelis for providing access to the data used in this study.

REFERENCES


