Adaptive and efficient nonlinear channel equalization for underwater acoustic communication

Dariush Kari a,*, Nuri Denizcan Vanli b, Suleyman S. Kozat a

a Department of Electrical and Electronics Engineering, Bilkent University, Ankara, 06800, Turkey
b Laboratory of Information and Decision Systems, Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, United States

HIGHLIGHTS

- An efficient and effective method for underwater acoustic channel equalization is proposed.
- The algorithm outperforms state-of-the-art methods in underwater channels.
- A completely adaptive piecewise linear equalizer using hierarchical partitioning of the received signal space.
- A guaranteed performance bound without any statistical assumptions on the data.

ARTICLE INFO

Article history:
Received 10 March 2017
Received in revised form 20 May 2017
Accepted 7 June 2017
Available online 12 June 2017

Keywords:
Underwater acoustic communication
Nonlinear channel equalization
Piecewise linear equalization
Adaptive filter
Self-organizing tree

ABSTRACT

We investigate underwater acoustic (UWA) channel equalization and introduce hierarchical and adaptive nonlinear (piecewise linear) channel equalization algorithms that are highly efficient and provide significantly improved bit error rate (BER) performance. Due to the high complexity of conventional nonlinear equalizers and poor performance of linear ones, to equalize highly difficult underwater acoustic channels, we employ piecewise linear equalizers. However, in order to achieve the performance of the best piecewise linear model, we use a tree structure to hierarchically partition the space of the received signal. Furthermore, the equalization algorithm should be completely adaptive, since due to the highly non-stationary nature of the underwater medium, the optimal mean squared error (MSE) equalizer as well as the best piecewise linear equalizer changes in time. To this end, we introduce an adaptive piecewise linear equalization algorithm that not only adapts the linear equalizer at each region but also learns the complete hierarchical structure with a computational complexity only polynomial in the number of nodes of the tree. Furthermore, our algorithm is constructed to directly minimize the final squared error without introducing any ad-hoc parameters. We demonstrate the performance of our algorithms through highly realistic experiments performed on practical field data as well as accurately simulated underwater acoustic channels.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Underwater acoustic (UWA) domain has become an important research field due to proliferation of new and exciting applications [1,2]. However, due to poor physical link quality, high latency, constant movement of waves and chemical properties of water, the underwater acoustic channel is considered as one of the most adverse communication mediums in use today [3–5]. These adverse properties of the underwater acoustic channel should be equalized by in order to provide reliable communication [2,3,6–11]. To combat the effects of long and time varying channel impulse response (CIR), orthogonal frequency division multiplexing (OFDM) seems to be an elegant solution [12]. Nevertheless, the cyclic prefix in such systems have to be longer than the CIR, however, in practice, the UWA channels possess long CIRs [2,12]. In addition, adding a long cyclic prefix to the data block results in a severe reduction in the data transmission rate [12,13]. Therefore, it is reasonable to use an effective channel equalizer before the OFDM detector in such receivers, to reduce the inter-symbol interference (ISI) to a level that can be compensated for by a relatively small cyclic prefix [13].

Furthermore, due to rapidly changing and unpredictable nature of underwater environment, constant movement of waves and transmitter–receivers, such processing should be adaptive [2,7,8,10]. However, there exist significant practical and theoretical difficulties to adaptive signal processing in underwater applications, since the signal generated in these applications show
high degrees of non-stationarity, limit cycles and, in many cases, are even chaotic. Hence, the classical adaptive approaches that rely on assumed statistical models are inadequate, since there is usually no or little knowledge about the statistical properties of the underlying signals or systems involved [3,14,15]. In this paper, we introduce a completely novel approach to adaptive channel equalization that is mathematically guaranteed to work uniformly for all possible signals, without any explicit or implicit statistical assumptions on the underlying signals or systems [16].

Although linear equalization is the simplest equalization method, it delivers an extremely inferior performance compared to that of the optimal methods, such as Maximum A Posteriori (MAP) or Maximum Likelihood (ML) methods [9,17]. Nonetheless, the high complexities of the optimal methods, and also their need of the channel information [9,18] make them practically infeasible for UWA channel equalization, because of the extremely large delay spread of UWA channels [17,19,20]. As a nonlinear alternative, in [21] the authors employ a Volterra filter in the equalizer, which is a kind of polynomial extension to the linear filters. However, Volterra filters cannot completely model the strong nonlinearity unless there is a priori knowledge about the channel and also suffer from high computational complexity in very long underwater channels.

In recent years, there has been a growing research on using artificial neural networks (ANN) for wireless channel equalization [22], since these methods can form arbitrary nonlinear decision boundaries. For instance, multilayer perceptron (MLP) based equalizers exhibit a superior performance to the conventional decision feedback equalizers (DFE). However, as the equalizer length increases (which is the case in UWA channels), the performance of the MLP-based equalizers deteriorates [23]. In addition, the major limitation of the MLP-based equalizer is its slow convergence [22].

Note that there are more advanced ANN-based methods in the wireless communications literature, e.g., functional-link ANN-based [24,25], wavelet ANN-based [26], radial basis function (RBF)-based [27], and recurrent neural network (RNN)-based [28,29] equalizers. Nevertheless, the large computational complexity due to the extensive training [22] of neural network based methods hinders their application in equalizing long underwater acoustic channels. Therefore, we introduce a piecewise linear method, which has the capability of modeling strong nonlinearities, while maintaining a low computational complexity.

In piecewise linear equalization methods, the space of the received signal is partitioned into disjoint regions, each of which is then fitted a linear equalizer [17,30]. We use the term “linear” to refer generally to the class of “affine” rather than strictly linear filters. In its most basic form, a fixed partition is used for piecewise linear equalization, i.e., both the number of regions and the region boundaries are fixed over time [30,31]. To estimate the transmitted symbol with a piecewise linear model, at each specific time, exactly one of the linear equalizers is used [17]. The linear equalizers in every region should be adaptive such that they can match the time varying channel response. However, due to the non-stationary statistics of the channel response, a fixed partition over time cannot result in a satisfactory performance. Hence, the partitioning should be adaptive as well as the linear equalizers in each region.

To this aim, we use a novel piecewise linear algorithm, in which not only the linear equalizers in each region, but also the region boundaries are adaptive [16]. Therefore, the regions are effectively adapted to the channel response and follow the time variations of the best equalizer in highly time varying UWA channels. In this sense, our algorithm can achieve the performance of the best piecewise linear equalizer with the same number of regions, i.e., the linear equalizers as well as the region boundaries converge to their optimal linear solutions.

Nevertheless, due to the non-stationary channel statistics, there is no knowledge about the number of regions of the best piecewise linear equalizer, i.e., even with adaptive boundaries, the piecewise linear equalizer with a certain number of regions, does not perform well. Thus, we use a tree structure to construct a class of models, each of which has a different number of regions [30,32]. Each of these models can be then deployed to construct a piecewise linear equalizer with adaptive filters in each region and also adaptive region boundaries [16]. In [32], the authors choose the best model (subtree) represented by a tree over a fixed partition. Nevertheless, the final estimates of all of these models should be effectively combined to achieve the performance of the best piecewise linear equalizer within this class [16]. For this purpose, we assign a weight to each model and linearly combine the results generated by each of them. However, due to the high computational complexity resulted from running a large number of different models, we introduce a technique to directly combine the node estimates to produce the exactly same result. Furthermore, the algorithm adaptively adjusts the node combination weights and the region boundaries as well as the linear equalizers in each region, to achieve the performance of the best piecewise linear equalizer. As a result, in highly time varying UWA channels, we significantly outperform other piecewise linear equalizers constructed over fixed partitions.

Our algorithm is shown (i) to provide significantly improved bit error rate (BER) performance over the conventional linear and piecewise linear equalizers in realistic UWA experiments (ii) and to have guaranteed performance bounds without any statistical assumptions. Note that the proposed algorithm minimizes the final soft squared error, with a computational complexity only polynomial in the number of nodes of the tree. In our algorithm, we avoid any artificial weighting of models with highly data dependent parameters, instead, “directly” minimize the squared error. Hence, the introduced approach significantly outperforms the other tree based approaches such as [30], as demonstrated in our simulations.

The paper is organized as follows: In Section 2, we describe our framework mathematically and introduce the notations. Then, in Section 3, we first present an algorithm to hierarchically partition the space of the received signal. We then present an upper bound on the performance of the promised algorithm and construct the algorithm. In Section 4, we show the performance of our method using highly realistic simulations, and then conclude the paper with Section 5.

2. Problem description

2.1. Notations

All vectors are column vectors and denoted by boldface lower case letters and all matrices are denoted by boldface upper case letters. For a vector $x$, $x^T$ is the ordinary transpose. $a^*$ is the conjugate of the complex number $a$.

2.2. Setup

As depicted in Fig. 1, we denote the received signal by $r(t)$, $r(t) \in \mathbb{R}$, and our aim is to determine the transmitted symbols $\{b(t)\}_{t \geq 0}$, which are sent through the channel every $T_s$ seconds. To transmit the symbols $\{b(t)\}_{t \geq 0}$, we use the raised cosine pulse shaping filter $g(t)$, which generates the continuous time signal $\tilde{b}(t)$, and then up-convert the signal to the carrier frequency $f_c$, and send it through the channel. Using the linear time varying convolution between $\tilde{b}(t)$ and $c(t, \tau)$, the received signal at time $t$ is

$$y(t) = \int_0^{\tau_{\text{max}}} \tilde{b}(t - \tau)c(t, \tau)d\tau + v(t),$$

where $\tau_{\text{max}}$ is a parameter that determines the maximum delay of the channel.
where \( c(t, \tau) \) is the channel response at time \( t \) related to an impulse launched at time \( t - \tau \), \( \tau \) is the delay time, \( \tau_{\text{max}} \) represents the maximum delay spread, and \( v(t) \) represents the ambient noise of the channel. The received signal is first passed through a bandpass filter centered at the carrier frequency \( f_c \) to remove unwanted disturbances [20]. The output of the bandpass filter is then down-converted, matched filtered, and sampled every \( T_s \) seconds (\( T_s \) is the symbol duration) to obtain a discrete time signal. With a small abuse of notation, in the rest of the paper, we denote the discrete sampling times by \( t \), such that the discrete time channel model is represented as [33]

\[
y(t) = \sum_{k=0}^{L} b(t - k) c(t, k) + v(t), \tag{1}
\]

where \( L = \lfloor \tau_{\text{max}} / T_s \rfloor \) indicates the length of the CIR. In the following sections, we use the discrete time model of the channel to address the equalization problem.

### 2.3. Doppler compensation

In order to compensate for the Doppler effects, a linear interpolation method is used to convert the sampling rate of the signal [20]. The complex baseband signal (after the matched filter), is sampled at four times the symbol rate and shown by \( y(t') \). The output of the interpolator is then down-sampled to two samples per symbol shown by \( r(t') \), which is finally used as the input to the equalizer [20]. The adaptive resampling algorithm is given as

\[
r(t') = (l - 1)y(t' + 1) + l_1 y(t'),
\]

\[
l_{l+1} = l + K_p \phi_l,
\]

\[
\phi_l = \arg(\hat{b}(t)\hat{b}^*(t)),
\]

where \( K_p \in [10^{-6}, 10^{-4}] \) is the phase tracking constant, and \( \hat{b}(t) \) is the output of the equalizer. In addition, in the training phase, \( \hat{b}(t') = b(t') \), and in the decision directed phase, \( \hat{b}(t) \) indicates the hard estimate of the \( b(t) \). Also, note that \( t' \in \{1, 3, 5, \ldots\} \) and \( t'' \in \{1, 2, 3, \ldots\} \) [20].

### 2.4. Equalization problem

A linear channel equalizer can be constructed as \( \hat{b}(t) = w^T(t)r(t) \), where \( r(t) \triangleq [r(t), \ldots, r(t - h + 1)]^T \) is the received signal vector (after Doppler compensation) at time \( t \), \( w(t) \triangleq [w_0(t), \ldots, w_{h-1}(t)]^T \) is the linear equalizer at time \( t \), and \( h \) is the equalizer length. Note that since \( r(t) \) is sampled at twice the symbol rate (see Section 2.3), using a length \( h \) equalizer corresponds to involving \( h/2 \) symbols for the equalization. The tap weights \( w(t) \) can be updated using any adaptive filtering algorithm such as the least mean squares (LMS) or the recursive least squares (RLS) algorithms [34] to minimize the squared error loss function, where the soft error at time \( t \) is \( c(t) = b(t) - \hat{b}(t) \).

However, we can get a significantly better performance by using adaptive nonlinear equalizers, because such linear equalization methods usually yield unsatisfactory performance in real life scenarios. Thus, we employ piecewise linear equalizers, which serve as the most natural and computationally efficient extension to linear equalizers [31]. The block diagram of a sample adaptive piecewise linear equalizer is shown in the Fig. 2. In such equalizers, the space of the received signal (here, \( \mathbb{R}^2 \)) is partitioned into disjoint regions, to each of which a different linear equalizer is assigned.

As an example, in Fig. 3, we use the received signal \( r(t) \) to estimate the transmitted bit \( b(t) \). We partition the space \( \mathbb{R}^2 \) into two regions \( w_1 \in \mathbb{R}^2 \) and \( w_2 \in \mathbb{R}^2 \) in these regions respectively. Hence the estimate \( \hat{b}(t) \) is calculated as

\[
\hat{b}(t) = \begin{cases} w_1^T(t)r(t) + c_1(t) & \text{if } r(t) \in R_1, \\ w_2^T(t)r(t) + c_2(t) & \text{if } r(t) \in R_2. \end{cases}
\]
where $c_1(t) \in \mathbb{R}$ and $c_2(t) \in \mathbb{R}$ are the offset terms, which can be embedded into $\mathbf{w}_1$ and $\mathbf{w}_2$, i.e., $\mathbf{w}_j = [\mathbf{w}_j^T \ c_j^T]^T, j = 1, 2$, and $r \triangleq [r^T \ 1]^T$. Hence the above expression can be rewritten as

$$
\hat{b}(t) = \begin{cases} 
\mathbf{w}_1^T(t)\mathbf{r}(t) & \text{if } r(t) \in R_1 \\
\mathbf{w}_2^T(t)\mathbf{r}(t) & \text{if } r(t) \in R_2.
\end{cases}
$$

We then update the equalizer’s coefficients using the LMS algorithm as

$$
\mathbf{w}_1(t + 1) = \mathbf{w}_1(t) + \mu_1 e(t) \mathbf{r}(t) \quad \text{if } r(t) \in R_1 \\
\mathbf{w}_2(t + 1) = \mathbf{w}_2(t) + \mu_2 e(t) \mathbf{r}(t) \quad \text{if } r(t) \in R_2.
$$

Note that, in our method (which will be discussed in the next section), we use LMS-based methods instead of other adaptive methods such as recursive least squares (RLS)-based methods due to the computational complexity constraints arising from running several different filters in parallel. However, one can straightforwardly extend our algorithm to use any other adaptive method.

To obtain a general expression, consider that we use a partition $P$ with $N$ subsets (regions) to divide the space of the received signal into disjoint regions, i.e., $P = \{R_1, \ldots, R_N\}$. The possible regions are $\cup_{j=1}^N R_j$, and $\hat{b}(t) = \hat{b}(t) = \mathbf{w}_j^T(t)\mathbf{r}(t)$ if $r(t) \in R_j$, which can be rewritten using indicator functions as

$$
\hat{b}(t) = \sum_{j=1}^N \hat{b}_j(t) \text{id}_j(r(t)) = \sum_{j=1}^N \mathbf{w}_j^T(t)\mathbf{r}(t) \text{id}_j(r(t)),
$$

where the indicator function $\text{id}_j(r(t))$ determines whether $r(t)$ lies in the region $R_j$ or not, i.e., $\text{id}_j(r(t)) = 1$ if $r(t) \in R_j$, and $\text{id}_j(r(t)) = 0$ otherwise.

**Remark 1.** Note that this algorithm can be directly applied to decision feedback equalizers (DFE). In this scenario, we partition the space of the extended received signal vector, i.e., we append the past decided symbols to the received signal vector as $\mathbf{r}(t) = [r(t), \ldots, r(t - h + 1), \hat{b}(t - 1), \ldots, \hat{b}(t - h_t + 1)]^T$, where $h_t$ is the length of the feedback part of the equalizer (i.e., we partition the obnervation as $\mathbf{x}[t-h_t]_\text{re} \mathbf{x}[t-h_t]_\text{im}$). Also, $\hat{b}(t) = \mathbb{Q}(\hat{b}(t))$ denotes the quantized estimate of the transmitted bit $b(t)$. Furthermore, corresponding to this extension in the received signal vector, we merge the feed-forward and feedback equalizers in each region to obtain an extended filter of length $h + h_t$ as $\mathbf{w}(t) = [\mathbf{w}_j^T(t) \ f_j(t)]^T$, where $f_j(t)$ represents the feedback filter corresponding to the $j$th region at time $t$. Hence, the $j$th region estimate is calculated as $\hat{b}_j(t) = \mathbf{w}_j^T(t) \mathbf{r}(t)$. In the next section, we extend these expressions to the case of an adaptive partition, both in the region boundaries and number of regions, and introduce our final algorithm.

### 3. Adaptive partitioning of the received signal space

#### 3.1. An adaptive piecewise linear equalizer with a specific partition

Here, we consider a specific partition with a certain number of regions. However, due to the non-stationary nature of underwater channel, a fixed partitioning over time cannot match well to the channel response, i.e., the partitioning should be adaptive. Hence, we use a partition with adaptive boundaries, although the number of regions is still fixed. To this end, we use hyper-planes with adaptive direction vectors (a vector orthogonal to the hyper-plane) as boundaries. Here, $\mathbf{n}$ refers to the direction vector of a hyperplane.

As an example, consider a partition with two regions (i.e., one boundary) as depicted in Fig. 3. The indicator functions for these regions are calculated as $\text{id}_1(r(t)) = \sigma(r(t))$ and $\text{id}_2(r(t)) = 1 - \sigma(r(t))$, where $\sigma(r(t)) = 1$ if $r(t) \in R_1$, and $\sigma(r(t)) = 0$ if $r(t) \in R_2$. Hence, $\sigma(r) = \frac{1}{1 + e^{r^T \mathbf{n} + b}}$ has a decision boundary at $r^T \mathbf{n} + b = 0$, which yields

$$
\sigma(r) = \begin{cases} 
1 & \text{if } r^T \mathbf{n} + b \ll 0 \\
0 & \text{if } r^T \mathbf{n} + b \gg 0
\end{cases}
$$

which can be used to simply update the direction vector $\mathbf{n}$ using the LMS algorithm, resulting in an adaptive boundary. For simplicity, with a small abuse of notation, we redefine $\mathbf{n}$ and $r$ as $r \triangleq [r^T \ b]^T$ and $r \triangleq [r^T \ 1]^T$, hence (3) can be rewritten as $\sigma(r) = \frac{1}{1 + e^{r^T \mathbf{n} + b}}$. By using the LMS algorithm to update the direction vector $\mathbf{n}$,

$$
\mathbf{n}(t + 1) = \mathbf{n}(t) - \frac{\mu}{2} \nabla_m (n) = \mathbf{n}(t) - \mu e(t) \frac{\partial \hat{b}(t)}{\partial \mathbf{m}(t)}
$$

$$
= \mathbf{n}(t) + \mu e(t) \left( \frac{\partial \text{id}_1(r(t)) \hat{b}_1(t)}{\partial \mathbf{m}(t)} + \frac{\partial \text{id}_2(r(t)) \hat{b}_2(t)}{\partial \mathbf{m}(t)} \right)
$$

$$
= \mathbf{n}(t) + \mu e(t) \sigma(r) \sigma(r - 1) \left( \frac{\partial \hat{b}_1(t)}{\partial \mathbf{m}(t)} - \frac{\partial \hat{b}_2(t)}{\partial \mathbf{m}(t)} \right) \mathbf{r}(t),
$$

since

$$
\frac{\partial \sigma(r)}{\partial \mathbf{n}} = -\mathbf{r} e^T \mathbf{n} + b = -\mathbf{r} \sigma(r)(1 - \sigma(r)).
$$

Since the region boundaries as well as the linear filters in each region are adaptive, if every filter converges, this equalizer can perform better than other piecewise linear equalizers with the same number of regions.

**Remark 2.** The piecewise linear equalizers are not limited to the BPSK modulation and one can easily extend these results to higher order modulation schemes like quadrature amplitude modulation (QAM) or pulse amplitude modulation (PAM). However, for the complex valued data (e.g., in QAM modulations) the separating function should change as $\sigma(r) \triangleq \frac{1}{1 + e^{r^T \mathbf{n} + b}}$, where the subscripts “re” and “im” denote the real and imaginary part of each vector respectively.

#### 3.2. The completely adaptive equalizer based on a turning boundaries tree

The block diagram of a sample adaptive piecewise linear equalizer with adaptive regions is shown in Fig. 4. Given a fixed number of regions, we can achieve the best piecewise linear equalizer with
the algorithm described in Section 3.1. However, there is no a priori knowledge about the number of regions of the best piecewise linear equalizer, and the best linear equalizer will change in time, due to the highly non-stationary nature of underwater medium. In order to provide an acceptable performance with a relatively small computational complexity, we hierarchically partition the space of the received signal, i.e., $\mathbb{R}^d$. Every node of the tree represents a region and is fitted a linear equalizer, as shown in Fig. 5. As shown in Fig. 4, each node $j$ provides its own estimate $\hat{b}_j(t)$, which are then combined to generate the final estimate $\hat{b}(t)$ as

$$\hat{b}(t) = \sum_{j=1}^{2^d+1} u_j(t)w_j^T(t)r(t) = u^T(t)\hat{b}(t),$$

where $u(t) = [u_1(t), \ldots, u_{2^d+1}(t)]^T$ is the combination weight vector, which is updated each time, and $\hat{b}(t) = [\hat{b}_1(t), \ldots, \hat{b}_{2^d+1}(t)]^T$ is the vector of the node estimates.

As depicted in Fig. 6, this tree introduces a number of partitions with different number of regions, each of which can be separately used as a piecewise linear equalizer [16]. In our proposed tree structure, each node represents a region that is the union of the regions assigned to its left and right children [35], as shown in Fig. 5. The root node is denoted by 1, and the left and right children of the node $j$ are denoted by $2j$ and $2j+1$, respectively. Obviously the root node indicates the whole space of the received signal, i.e., $\mathbb{R}^d$. The estimate generated by node $j$ is calculated as $\hat{b}_j(t) = w_j^T(t)r(t)$. In addition, $\alpha_d$ represents the number of partitioning trees with depth $\leq d$. Hence, $\alpha_{d+1} = \alpha_d^2 + 1$, which shows that there are a doubly exponential number of models embedded in a depth-$d$ tree (See Fig. 6), each of which can be used to construct a piecewise linear equalizer [30]. Each model consists of a number of nodes. However, the number of regions (leaf nodes) in each model can be different with that of other models, as shown in Fig. 6, e.g., $P_2$ has 2 regions, while $P_3$ has 4 regions. Therefore, we implicitly run all of the piecewise linear equalizers constructed based on these partitions, and linearly combine their results to estimate the transmitted bit. Then, by adaptively learning the combination weights, we achieve the best estimate at each time.

To clarify, suppose the corresponding space of the received signal vector is two dimensional, i.e., $r(t) \in \mathbb{R}^2$, and we partition this space using a depth-2 tree as in Fig. 5. A depth-2 tree is represented by three separating functions $\sigma_1(r(t))$, $\sigma_2(r(t))$ and $\sigma_3(r(t))$, which are defined using three hyper-planes with direction vectors $n_1(t)$, $n_2(t)$ and $n_3(t)$, respectively (See Fig. 5). The left and right children of the node $j$ are $2j$ and $2j+1$, respectively, therefore, the indicator functions are defined as

$$id_1(r) = 1$$
$$id_2(r) = \sigma_i(r) \times id_j(r)$$
$$id_{2j+1}(r) = (1 - \sigma_j(r)) \times id_j(r),$$

where $j \leq 2^d - 1$ and $\sigma_j(r) \triangleq \frac{1}{\|n_j \|}$. Due to the tree structure, three separating hyper-planes generate four regions, each corresponding to a leaf node on the tree given in Fig. 5. The partitioning is defined in a hierarchical manner, i.e., $r(t)$ is first processed by $\sigma_1(r(t))$ and then by $\sigma_i(t)$, $i = 2, 3$. A complete tree defines a doubly exponential number, $O(2^d)$, of models each of which can be used to partition the space of the received signal vector. As an example, a depth-2 tree defines 5 different partitions as shown in Fig. 6, each of which constructed using the leaves and the nodes of the original tree.
Consider the fifth model in Fig. 6, i.e., $P_5$, which consists of 4 disjoint regions, each corresponding to a leaf node of the original complete tree in Fig. 5, labeled as 4, 5, 6, and 7. At each region, say the 4th region, we generate the estimate $\hat{b}_4(t) = w_4^T(t)r(t)$, where $w_4(t) \in \mathbb{R}^3$ is the tap weights of the linear equalizer assigned for any $r(t)$. We emphasize that any $P_i$, $i = 1, \ldots, 5$ can be used in a similar fashion to construct a piecewise linear channel equalizer. Based on these model estimates, the final estimate of the transmitted bit $b(t)$ is obtained by

$$\hat{b}(t) = \sum_{j} \hat{b}_j(t)\hat{u}_j(t) = \hat{b}(t)^T u(t),$$

where $\hat{b}(t)$ is the estimate of the transmitted bit $b(t)$ generated by the $k$th piecewise linear channel equalizer, $k = 1, \ldots, \alpha$. We use the LMS algorithm to update the weighting vector $u(t)$. Note that in our method, which is given in Algorithm 1, we linearly combine the estimates generated by all $\alpha$ models, using the weighting vector $u(t)$ to weight the estimate of the transmitted bit $b(t)$, such that we can achieve the best performance on the tree.

Under the moderate assumptions on the cost function that $e^2(u(t))$ is a $\lambda$-strong convex function [36] and also its gradient is upper bounded by a constant number, the following theorem provides an upper bound on the error performance of our algorithm (given in Algorithm 1).

**Theorem 1.** Let $\{b(t)\}_{t \geq 1}$ and $\{r(t)\}_{t \geq 1}$ represents arbitrary and real-valued sequences of transmitted bits and channel outputs. The algorithm for $\hat{b}(t)$ given in Algorithm 1 when applied to any sequence with an arbitrary length $L \geq 1$ yields

$$E \left\{ \sum_{t=1}^{L} (b(t) - \hat{b}(t))^2 \right\} - \min_{z \in \mathbb{R}^{d}} E \left\{ \sum_{t=1}^{L} (b(t) - z^T \hat{b}(t))^2 \right\} \leq O(\log L),$$

where $z$ is an arbitrary constant combination weight vector, used to combine the results of all models.

**Outline of the proof:** Since we use a stochastic gradient method to update the weighting vector in Algorithm 1, from Chapter 3 of [37] it can be straightforwardly shown that

$$\sum_{t=1}^{L} (b(t) - \hat{b}(t))^2 - \min_{z \in \mathbb{R}^{d}} \sum_{t=1}^{L} (b(t) - z^T \hat{b}(t))^2 \leq O(\log L),$$

in a strong deterministic sense, which is a well known result in computational learning theory [37]. Taking the expectation of both sides of this deterministic bound yields the result in (8).

This theorem implies that the algorithm given in Algorithm 1 asymptotically achieves the performance of the optimal linear combination of the $\alpha$ $\approx (1.5)^2$ different adaptive piecewise linear equalizers, represented using a depth-$d$ tree, in the MSE sense, with a computational complexity $O(b4^d)$ (i.e., only polynomial in the number of nodes). Moreover, note that as the data length increases and each region becomes dense enough, the linear equalizer in each region, converges to the corresponding linear MMSE equalizer in that region [30]. In addition, since in our algorithm the tree structure is also adaptive, it can follow the data statistics effectively even when the channel is highly time varying. Therefore, our algorithm outperforms the conventional methods and asymptotically achieves the performance of the best piecewise linear equalizer.

By adjusting the combination weights using LMS algorithm to achieve the performance of the best piecewise linear equalizer we obtain

$$u(t + 1) = u(t) + \frac{1}{\mu} \nabla w(t)e^2(t) = u(t) + \eta e(t) \hat{b}(t).$$

Note that, as depicted in Fig. 6, each model weight equals the sum of the weights assigned to its leaf nodes, hence, we have $u_i(t) = \sum_{j \in P_i} u_j(t)$, which in turn results in the following node weights update algorithm

$$u_j(t + 1) = u_j(t) + \mu e(t) \hat{b}(t) i.d(r(t)),$$

where $u_j(t)$ denotes the weight assigned to the $j$th node at time $t$. So far, we have shown how to construct a piecewise linear equalizer using separating functions and how to combine the estimates of all models to achieve the performance of the best piecewise linear equalizer. However, there are a doubly exponential number of these models, hence it is computationally prohibited to run all of these models and combine their results. In order to reduce this complexity while reaching exactly the same result, we directly combine the node estimates, i.e., instead of running all possible models, we combine the node estimates with special weights, which yields the same result. We now illustrate how to directly combine the node weights in our algorithm. The overall estimate using all models contributions is

$$\hat{b}(t) = \sum_{i=1}^{a_d} \hat{b}_i(t)u_i(t)$$

$$= \sum_{i=1}^{a_d} \hat{b}_i(t) \left( \sum_{j \in P_i} u_j(t) \right)$$

$$= \sum_{i=1}^{a_d} \left( \sum_{k \in P_i} \left( \sum_{j \in P_i} u_j(t) \hat{b}_i(t) \right) \right) u_j(t)$$

$$= \sum_{i=1}^{a_d} \left( \sum_{k \in P_i} \hat{b}_i(t) u_j(t) \right) i.d(r(t)),$$

where $j$ and $k$ indicate two arbitrary nodes. For each node $k$, we define $z_k(t) \triangleq \sum_{j \in P_i} u_j(t)$ $\hat{b}_i(t)$. Hence we have $\hat{b}(t) = \sum_{i=1}^{a_d} \left( \sum_{k \in P_i} z_k(t) u_j(t) \right) i.d(r(t))$.

Consider that $\Gamma = \{\Gamma_1, \ldots, \Gamma_{a_d} \}$ is the family of models (subtrees) in all of which the node $k$ is a leaf node, where $\theta_k(d_k)$ denotes the number of such models. Therefore the final estimate of our algorithm can be rewritten as

$$\hat{b}(t) = \sum_{k=1}^{2^{d+1}} \zeta_k \left( \sum_{j \in \Gamma_1} u_j(t) + \cdots + \sum_{j \in \Gamma_{a_d} \eta_k(d_k)} u_j(t) \right),$$

We denote by $\rho(j_0,k)$ the number of models in all of which the nodes $j_0$ and $k$ appear as the leaf nodes simultaneously. The weight of each node $j_0$ (i.e., $\eta_k$) appears in the above expression exactly $\rho(j_0,k)$ times, which yields the following expression for the final estimate

$$\hat{b}(t) = \sum_{k=1}^{2^{d+1}} z_k(t) \beta_k(t),$$

where $\beta_k(t) \triangleq \sum_{j \in \Gamma_k} \zeta_k \left( \sum_{j \in \Gamma_1} u_j(t) + \cdots + \sum_{j \in \Gamma_{a_d} \eta_k(d_k)} u_j(t) \right)$.
shown that 

\[ \sum_{j=1}^{2d+1} u_{j_0}(t) \rho(j_0, k). \]

We now illustrate how to calculate \( \rho(j, k) \) in a depth-\( d \) tree. Here, \( \theta_\ell(d_j) \) denotes the number of models extracted from a depth-\( d \) tree, in all of which \( j \) is a leaf node. It can be shown that \( \theta_\ell(d_j) = \prod_{k=1}^{d_j} \alpha_{d-k} \), where \( d_j = \lceil \log_2 j \rceil \) denotes the depth of the \( j \)th node [16]. To calculate \( \rho(j, k) \), we first note that \( \rho(j, k) = \rho(k, j) \) and \( \rho(j, j) = \theta_\ell(j) \). Therefore, we obtain

\[
\rho(j, k) = \begin{cases} 
\theta_\ell(j) & \text{if } j = k \\
\theta_{d-1}(d_k - l - 1) & \text{if } j \neq k,
\end{cases}
\]

where, \( l \) represents the depth of the nearest common ancestor of the nodes \( j \) and \( k \) in the tree, i.e., an ancestor of both nodes \( j \) and \( k \), none of the children of that is a common ancestor of \( j \) and \( k \). This parameter can be calculated using the following algorithm.

Assume that, without loss of generality, \( j \leq k \). Obviously if \( j \) is an ancestor of \( k \), it is also the nearest common ancestor, i.e., \( l = d_j \). However, if \( j \) is not an ancestor of \( k \), we define \( j' \triangleq 2^{d_j - d_j} \), which is a grandchild of the node \( j \). Hence, the nearest common ancestor of \( j' \) and \( k \) is that of \( j \) and \( k \). The following procedure computes the parameter \( l \).

\[
l = 0; \\
\delta = d_k; \\
\text{while } (l \leq d_k) \text{ do} \\
\quad \delta = \delta - l; \\
\quad \text{if } (j', k \leq 2^{d_j - 1} + 2^d \text{ or } j', k \geq 2^{d_j - 1} + 2^d) \text{ then } l = l + 1; \\
\quad \text{else } \text{stop;}
\]

In order to update the region boundaries, we update their direction vectors as follows

\[
n_j(t + 1) = n_j(t) - \frac{1}{2} \mu \nabla n_j(t) e^2(t),
\]

where \( \nabla n_j(t) e^2(t) \) is the derivative of \( e^2(t) \) with respect to \( n_j(t) \). Since \( e(t) = b(t) - \hat{b}(t) \) the updating expression can be calculated as follows

\[
n_j(t + 1) = n_j(t) - \frac{1}{2} \mu \nabla n_j(t) e^2(t)
\]

\[= n_j(t) + \mu e(t) \frac{\partial \hat{b}(t)}{\partial n_j(t)}
\]

\[= n_j(t) + \mu e(t) \sum_{k=1}^{2d+1} \frac{\partial \hat{b}(t)}{\partial z_k(t)} \frac{\partial z_k(t)}{\partial n_j(t)}
\]

\[= n_j(t) + \mu e(t) \sum_{k=1}^{2d+1} \beta_k(t) \hat{b}(t) \frac{\partial i d_k(r(t))}{\partial n_j(t)}
\]

\[= n_j(t) + \mu e(t) \sum_{k=1}^{2d+1} \beta_k(t) \hat{b}(t) \frac{\partial \sigma_j(r(t))}{\partial n_j(t)}
\]

However note that not all of the \( i d_k(r(t)) \) functions involve \( \sigma_j(r(t)) \), i.e., only the nodes of the subtree with the root node \( j \) are included.

Hence,

\[
\sum_{k=1}^{2d+1} \beta_k(t) \hat{b}(t) \frac{\partial i d_k(r(t))}{\partial \sigma_j(r(t))}
\]

\[= \sum_{m=0}^{d-d_j-1} \sum_{s=0}^{2^{d-1}+1} \beta_m(n_j(t) + 1) \frac{\partial i d_m(n_j(t))}{\partial \sigma_j(r(t))} \frac{\partial i d_m(n_j(t))}{\partial \sigma_j(r(t))}
\]

We have presented the Algorithm 1 for a “turning boundaries tree” (TBT) equalizer, which is completely adaptive to the channel response. Especially in our algorithm both the number of regions and the region boundaries as well as the linear equalizers in each region are adaptive. We emphasize that the learning rates and initial values of all filters can be different.

4. Simulations

We show the efficacy of our algorithm (TBT) over a synthetic and a real world acoustic channel and compare our algorithm to state-of-the-art equalization methods including: the Fixed Boundaries Tree (FBT) equalizer, Finest Partition with Fixed Boundaries (FF), Finest Partition with Turning Boundaries (FT) (all having depths \( d = 2 \)), linear normalized LMS (NLMS) equalizer, Variable Step Size LMS (VSLMS) of [38] and, Subband Adaptive Filter [39]. In addition, we implement and compare the performance of our algorithm and the NLMS in a DFE structure, denoted by DFE-TBT and DFE-NLMS, respectively. The Finest Partition refers to the partition consisted of all leaf nodes of the tree (the \( P_n \) model in Fig. 6). Also, we use FBT to refer to an equalizer with “fixed” boundaries, which adaptively update the node weights as well as TBT algorithm. We use the LMS algorithm in the linear equalizer of each node for all algorithms. For the direction vector at node \( j \), i.e., \( n_j \), we set the \( d_j \)-th element to 1, and all other elements to zero, where \( d_j \) indicates the depth of this node on the tree. All other filters are initialized by all zero vectors.

4.1. Experiments on a simulated channel

4.1.1. Setup

In this section, we illustrate the performance of our algorithm under a highly realistic UWA channel equalization scenario. The UWA channel response is generated using the algorithm introduced in [40], which presents highly accurate modeling of the real life UWA communication experiments. The simulation configurations and the parameters used for simulating the channel are presented in the Table 1. We sent 60000 bits generated by
Compute $\rho(j, k)$ for all pairs $[j, k]$ of nodes; 
for $t = 1$ to $L$ do 
    $r = [r(t), \ldots, r(t - h + 1)]^T$; 
    for $k = 1$ to $2^d - 1$ do 
        $\sigma_k = \frac{1}{1 + e^{\mu R_k}}$; 
    end 
    $id_1 = 1$; 
    for $k = 1$ to $2^d - 1$ do 
        $id_{2k} = id_k \sigma_k$; 
        $id_{2k+1} = id_k (1 - \sigma_k)$; 
    end 
    $b = 0$; 
    for $k = 1$ to $2^d + 1 - 1$ do 
        $b_k = w_k ^ T r$; 
        $z_k = b_k id_k$; 
        $\hat{b}_k = 0$; 
        for $j = 1$ to $2^d + 1 - 1$ do 
            $\hat{b}_k = \hat{b}_k + u_j \rho(j, k)$; 
        end 
        $\hat{b}(t) = \hat{b}(t) + z_k \hat{b}_k$; 
    end 
if train mode then 
    $\hat{b} = \hat{b}(t)$; 
else 
    $\hat{b} = Q(\hat{b}(t))$; 
end 
$e = b - \hat{b}(t)$; 
for $k = 1$ to $2^d + 1 - 1$ do 
    $w_k = w_k + \mu_k e id_k r$; 
    $u_k = u_k + \eta e z_k$; 
end 
for $j = 1$ to $2^d - 1$ do 
    $d_j = \lfloor \log_2(j) \rfloor$; 
    for $m = 0$ to $d - d_j - 1$ do 
        for $p = 0$ to $2^{m} - 1$ do 
            $i = 2^{m+1} j + p$; 
            $S_1 = S_1 + \beta_i \hat{b}_i \frac{id_i}{\sigma_i}$; 
        end 
        for $p = 2^m$ to $2^{m+1} - 1$ do 
            $i = 2^{m+1} j + p$; 
            $S_2 = S_2 + \beta_i \hat{b}_i \frac{id_i}{\sigma_i}$; 
        end 
        $S = S + S_1 - S_2$; 
    end 
$\eta_j = \eta_j + \xi e \sigma (\sigma - 1) S r$; 
end 

Algorithm 1: Turning Boundaries Tree (TBT) Equalizer

repeating a Turyn sequence [41] (with a length of 28 bits), after pulse shaping with a raised cosine filter with a roll-off factor of 0.25, over the simulated UWA channel shown in Fig. 7. In addition, the system setup is the same as one described in Section 2.2. Also, we have calculated the SNR after down converting and matched filtering (i.e., from the baseband signal). The step sizes are set to $\mu = 0.08$ for all equalizers except the SAF, which has a step size of 0.01. Furthermore, in SAF we use 4 subbands, in all of which we use the same step size. In all algorithms we have used length 5 equalizers. Also, the length of feedback part in DFEs is set to 3. The results are averaged over 10 repetitions, and show the extremely superior performance of our algorithm over other methods

4.1.2 Results and discussion

Fig. 8b shows the normalized time accumulated squared errors of the equalizers, when $SNR = -5$ dB. We emphasize that the TBT equalizer significantly outperforms its competitors, where the FBT equalizer cannot provide a satisfactory result at the low SNRs, since it commits to the initial partitioning structure. Note that the TBT equalizer adapts its region boundaries and can successfully perform channel equalization even for a highly difficult UWA channel. The Fig. 8a shows the bit error rate performance in different SNRs for different equalizers.

In the second experiment, again, we sent 60000 symbols of repeated Turyn sequence over the simulated channel and used TBT algorithm with different depths to equalize the channel. The results, as shown in Fig. 9, demonstrate that increasing the depth of the tree improves the performance. However, as the depth of the tree increases, the effect of the depth diminishes. This is because increasing the depth introduces finer partitions, i.e., the partitions with more regions. As the number of the regions in a partition increases, the data congestion in each region decreases, hence, the linear filters in these regions cannot fully converge to their MMSE solutions. As a result, the estimates of these regions (nodes) will be contributed to the final estimate with a much lower combination weight than other nodes, which are also present in a lower depth tree. Therefore, although increasing the depth of the tree improves the result, we cannot get a significant improvement in the performance by only increasing the depth.
Moreover, the evolution of node combination weights in the synthetic channel [40] experiment is shown in Fig. 10, which indicates the contribution of each node’s estimate in the final estimate. This figure shows that node 1, the root node, has the largest weight at the early stages of the algorithm, while its weight decreases as the time passes. On the other hand, the weights assigned to nodes with finer partitions, e.g., nodes 4, 6, and 7, gradually increase. Therefore, as the finer models receive sufficient amount of data to be trained with, they contribute more to the final nonlinear estimation.

4.2. Experiments on a real world dataset

To evaluate the efficacy of our algorithm in a real life scenario, we use a dataset provided by the Applied Research Laboratory (ARL) at University of Texas-Austin, on November 2009 [42,43]. The maximum depth of the lake is about 37 meters. The distance between the transmitter and the receiver is in range of 73 – 267 meters, and there is a towing motion of the transducer at speeds of $\sim 5$ km/h at varying depths of at most 5 m [42].

We use one of the packets described in this dataset, which consists of 4096 BPSK modulated symbols which are later pulse shaped by a raised cosine filter with a roll-off factor of 1. Also, the first 52 symbols of the transmitted data frame, is consisted of 4 repetitions of a Barker sequence of length 13. The power spectrum of the transmitted and received signals are depicted in Fig. 11. The sampling rate at the transmitter and receiver is 200 kHz, the symbol rate is 15.625 kHz, and the carrier frequency is 62.5 kHz. We have set the step size of all methods to 0.0001, the depth of trees to 2, and the number of subbands in SAF method to 16.

The receiver consists of a frame synchronizer block such that it receives a base banded packet of the data, searches for four consecutive blocks of Barker sequence each of length 13, which precedes the data frame. Then, the data frame passes through the Doppler compensation filter (discussed in Section 2.3) and is resampled at a rate of 4 samples per symbol. Finally, the equalizer removes the ISI effect from the data. Fig. 12 shows the MSE performances of different equalizers on the described dataset. Furthermore, the Table 2 indicates the resulting bit error rates using each equalizers.

As depicted in Fig. 12 and Table 2, our method significantly outperforms the other methods in this real life experiment. Moreover, as...
Table 2
The BER performance of different methods over the ARLUT dataset.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NLMS</th>
<th>DFE-NLMS</th>
<th>VSLMS</th>
<th>SAF</th>
<th>FBT</th>
<th>TBT</th>
<th>DFE-TBT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BER</td>
<td>0.5112</td>
<td>0.5068</td>
<td>0.4880</td>
<td>0.3447</td>
<td>0.2976</td>
<td>0.2903</td>
<td>0.2971</td>
</tr>
</tbody>
</table>

Fig. 11. The power spectrum of the transmitted and received signals in ARLUT dataset, obtained using "Welch" method in MATLAB.

Fig. 12. The MSE performance comparison of different methods in the experiment over ARLUT dataset.

5. Conclusion

We study nonlinear UWA channel equalization using hierarchical structures, where we partition the received signal space using a nested tree structure and use different linear equalizers in each region. In this framework, we introduce a tree-based piecewise linear equalizer that both adapts its linear equalizers in each region as well as its tree structure to best match to the underlying channel response. Our algorithm asymptotically achieves the performance of the best linear combination of a doubly exponential number of adaptive piecewise linear equalizers represented on a tree with a computational complexity only polynomial in the number of tree nodes. Since our algorithm directly minimizes the squared error and avoid using any artificial weighting coefficients, it strongly outperforms the conventional linear and piecewise linear equalizers as shown in our experiments.

Acknowledgments

This work is in part supported by Turkish Academy of Sciences, Outstanding Researcher Programme, and TUBITAK, Contract No:112E161.

References


