The Hebbian-LMS Learning Algorithm

Bernard Widrow, Youngsik Kim, and Dookun Park
Department of Electrical Engineering, Stanford University, CA, USA

Abstract—Hebbian learning is widely accepted in the fields of psychology, neurology, and neurobiology. It is one of the fundamental premises of neuroscience. The LMS (least mean square) algorithm of Widrow and Hoff is the world's most widely used adaptive algorithm, fundamental in the fields of signal processing, control systems, pattern recognition, and artificial neural networks. These are very different learning paradigms. Hebbian learning is unsupervised. LMS learning is supervised. However, a form of LMS can be constructed to perform unsupervised learning and, as such, LMS can be used in a natural way to implement Hebbian learning. Combining the two paradigms creates a new unsupervised learning algorithm that has practical engineering applications and provides insight into learning in living neural networks. A fundamental question is, how does learning take place in living neural networks? "Nature's little secret," the learning algorithm practiced by nature at the neuron and synapse level, may well be the Hebbian-LMS algorithm.

I. Introduction

Donald O. Hebb has had considerable influence in the fields of psychology and neurobiology since the publication of his book "The Organization of Behavior" in 1949 [1]. Hebbian learning is often described as: "neurons that fire together wire together." Now imagine a large network of interconnected neurons whose synaptic weights are increased because the presynaptic neuron and the postsynaptic neuron fired together. This might seem strange. What purpose would nature fulfill with such a learning algorithm?

In his book, Hebb actually said: "*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*"

"Fire together wire together" is a simplification of this. Wire together means increase the synaptic weight. Fire

Digital Object Identifier 10.1109/MCI.2015.2471216
Date of publication: 13 October 2015

together is not exactly what Hebb said, but some researchers have taken this literally and believe that information is carried with the timing of each activation pulse. Some believe that the precise timing of presynaptic and postsynaptic firings has an effect on synaptic weight changes. There is some evidence for these ideas [2]–[4] but they remain controversial.

Neuron-to-neuron signaling in the brain is done with pulse trains. This is AC coupling and is one of nature's "good ideas", avoiding the effects of DC level drift that could be caused by the presence of fluids and electrolytes in the brain. We believe that the output signal of a neuron is the neuron's firing rate as a function of time.

Neuron-to-neuron signaling in computer simulated artificial neural networks is done in most cases with DC levels. If a static input pattern vector is presented, the neuron's output is an analog DC level that remains constant as long as the input pattern vector is applied. That analog output can be weighted by a synapse and applied as an input to another neuron, a "postsynaptic" neuron, in a layered network or otherwise interconnected network.

The purpose of this paper is to introduce a new learning algorithm that we call Hebbian-LMS. It is an implementation of Hebb's teaching by means of the LMS algorithm of Widrow and Hoff. With the Hebbian-LMS algorithm, unsupervised or autonomous learning takes place locally, in the individual neuron and its synapses, and when many such neurons are connected in a network, the entire network learns autonomously. One might ask, what does it learn? This question will be considered below where applications will be presented.

There is another question that can be asked: Should we believe in Hebbian learning? Did Hebb arrive at this idea by doing definitive biological experiments, by "getting his hands wet"? The answer is no. The idea came to him by intuitive reasoning. Like Newton's theory of gravity, like Einstein's theories of relativity, like Darwin's theory of evolution, it was a thought experiment propounded long before modern knowledge and instrumentation could challenge it, to refute it, or verify it. Hebb described synapses and synaptic plasticity, but how synapses and neurotransmitters worked was unknown in Hebb's time. So far, no one has contradicted Hebb, except for some details. For example, learning with "fire together wire together" would cause the synaptic weights to only increase until all of them reached saturation. That would make an uninteresting neural network, and nature would not do this. Gaps in the Hebbian learning rule will need to be filled, keeping in mind Hebb's basic idea, and well-working adaptive algorithms will be the result. The Hebbian-LMS algorithm will have engineering applications, and it may provide insight into learning in living neural networks.

The current thinking that led us to the Hebbian-LMS algorithm has its roots in a series of discoveries that were made since Hebb, from the late 1950's through the 1960's. These discoveries are reviewed in the next three sections. The sections beyond describe Hebbian-LMS and how this algorithm could be nature's algorithm for learning at the neuron and synapse level.

II. Adaline and the LMS Algorithm, from the 1950's

Adaline is an acronym for "Adaptive Linear Neuron." A block diagram of the original Adaline is shown in Figure 1. Adaline was adaptive, but not really linear. It was more than a neuron since it also included the weights or synapses. Nevertheless, Adaline was the name given in 1959 by Widrow and Hoff.

Adaline was a trainable classifier. The input patterns, the vectors X_k, $k = 1, 2, \cdots, N$, were weighted by the weight vector $W_k = [w_{1k}, w_{2k}, \cdots, w_{nk}]^T$, and their inner product was the sum $y_k = X_k^T W_k$. Each input pattern X_k was to be classified as a $+1$ or a -1 in accord with its assigned class, the "desired response." Adaline was trained to accomplish this by adjusting the weights to minimize mean square error. The error was the difference between the desired response d_k and the sum y_k, $e_k = d_k - y_k$. Adaline's final output q_k was taken as the sign of the sum y_k, i.e. $q_k = SGN(y_k)$, where the function $SGN(\cdot)$ is the signum, take the sign of. The sum y_k will henceforth be referred to as $(\mathrm{SUM})_k$.

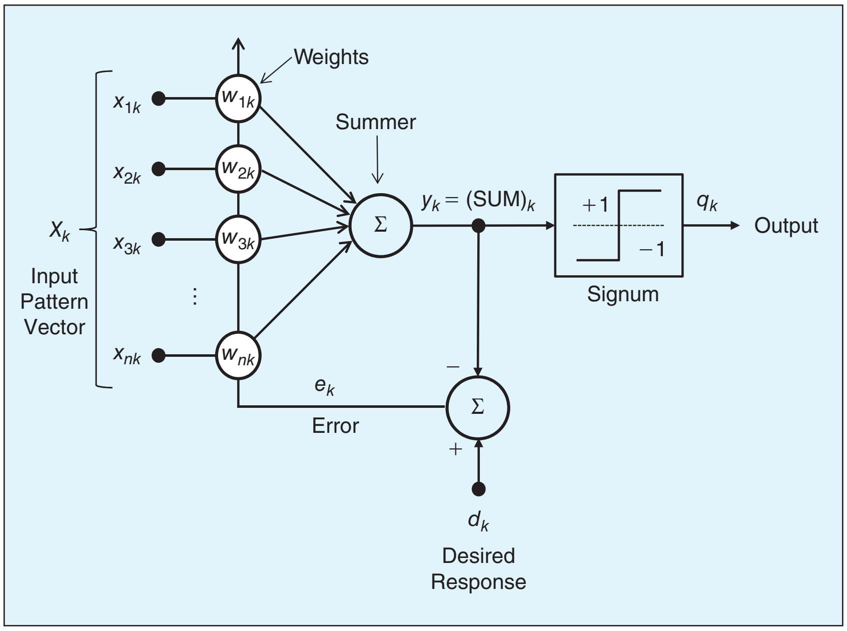


FIGURE 1 Adaline (Adaptive linear neuron.)

The weights of Adaline were trained with the LMS algorithm, as follows:

$$W_{k+1} = W_k + 2\mu e_k X_k, \tag{1}$$

$$e_k = d_k - X_k^T W_k. \tag{2}$$

Averaged over the set of training patterns, the mean square error is a quadratic function of the weights, a quadratic "bowl." The LMS algorithm uses the methodology of steepest descent, a gradient method, for pulling the weights to the bottom of the bowl, thus minimizing mean square error.

The LMS algorithm was invented by Widrow and Hoff in 1959 [5]. The derivation of this algorithm is given in many references. One such reference is the book "Adaptive Signal Processing" by Widrow and Stearns [6]. The LMS algorithm is the most widely used learning algorithm in the world today. It is used in adaptive filters that are key elements in all modems, for channel equalization and echo canceling. It is one of the basic technologies of the internet and of wireless communications. It is basic to the field of digital signal processing.

The LMS learning rule is quite simple and intuitive. Equations (1) and (2) can be represented in words:

"With the presentation of each input pattern vector and its associated desired response, the weight vector is changed slightly by adding the pattern vector to the weight vector, making the sum more positive, or subtracting the pattern vector from the weight vector, making the sum more negative, changing the sum in proportion to the error in a direction to make the error smaller."

A photograph of a physical Adaline made by Widrow and Hoff in 1960 is shown in Figure 2. The input patterns of this Adaline were binary, 4 × 4 arrays of pixels, each pixel having a value of +1 or −1, set by the 4 × 4 array of toggle switches. Each toggle switch was connected to a weight, implemented by a potentiometer. The knobs of the potentiometers, seen in the photo, were manually rotated during the training process in accordance with the LMS algorithm. The sum was displayed by the meter. Once trained, output decisions were +1 if the meter reading was positive, and −1 if the meter reading was negative.

The earliest learning experiments were done with this Adaline, training it as a pattern classifier. This was supervised learning, as the desired response for each input training pattern was given. A video showing Prof. Widrow training Adaline can be seen online [https://www.youtube.com/watch?v=skfNlwEbqck].

III. Unsupervised Learning with Adaline, from the 1960's

In order to train Adaline, it is necessary to have a desired response for each input training pattern. The desired response indicated the class of the pattern. But what if one had only input patterns and did not know their desired responses, their classes? Could learning still take place? If this were possible, this would be unsupervised learning.

In 1960, unsupervised learning experiments were made with the Adaline of Figure 2 as follows. Initial conditions for the weights were randomly set and input patterns were presented without desired responses. If the response to a given input pattern was already positive (the meter reading to the right of zero), the desired response was taken to be exactly +1. A response of +1 was indicated by a meter reading half way on the right-hand side of the scale. If the response was less than +1, adaptation by LMS was performed to bring the response up toward +1. If the response was greater than +1, adaptation was performed by LMS to bring the response down toward +1.

If the response to another input pattern was negative (meter reading to the left of zero), the desired response was taken to be exactly −1 (meter reading half way on the left-hand side of the scale). If the negative response was more positive than −1, adaptation was performed to bring the response down toward −1. If the response was more negative than −1, adaptation was performed to bring the response up toward −1.

With adaptation taking place over many input patterns, some patterns that initially responded as positive could ultimately reverse and give negative responses, and vice versa. However, patterns that were initially responding as positive were more likely to remain positive, and vice versa. When the process converges and the responses stabilize, some responses would cluster about +1 and the rest would cluster about −1.

The objective was to achieve unsupervised learning with the analog responses at the output of the summer (SUM) clustered at +1 or −1. Perfect clustering could be achieved if the training patterns were linearly independent vectors whose number were less than or equal to the number of weights. Otherwise, clustering to +1 or −1 would be done as well as possible in the least squares sense. The result was that similar patterns were similarly classified, and this simple unsupervised learning algorithm was an automatic clustering algorithm. It was called "bootstrap learning" because Adaline's quantized output was used as the desired response. This idea is represented by the block diagram in Figure 3.

Research done on bootstrap learning was reported in the paper "Bootstrap Learning in Threshold Logic Systems," presented by Bernard Widrow at an International Federation of Automatic Control (IFAC) conference in 1966 [7]. This work led to the 1967 Ph.D. thesis of William C. Miller, at the time a student of Professor Widrow, entitled "A Modified Mean Square Error Criterion for use in Unsupervised Learning" [8]. These papers described and analyzed bootstrap learning.

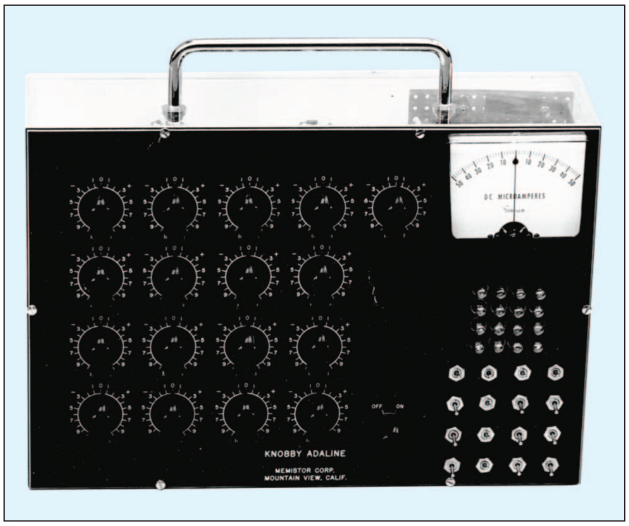


FIGURE 2 Knobby Adaline.

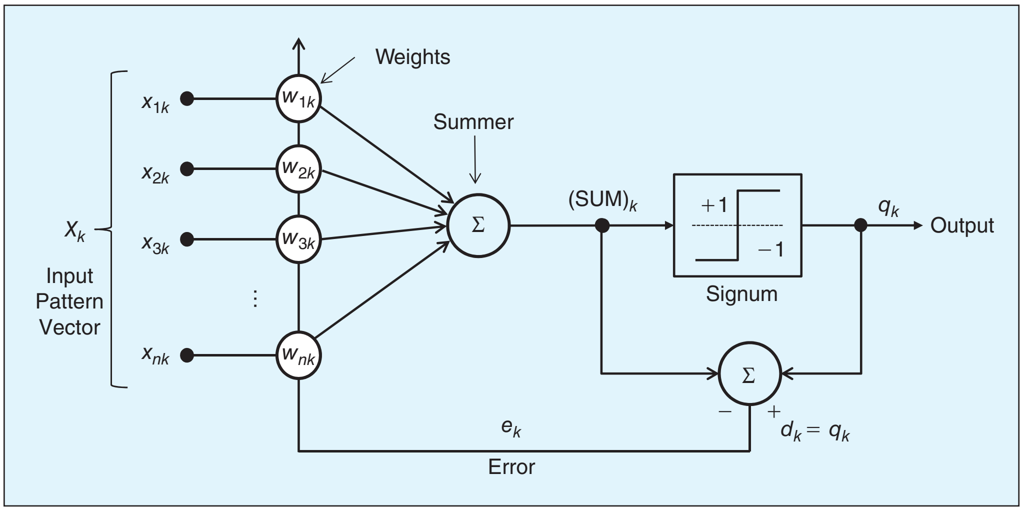


FIGURE 3 Adaline with bootstrap learning.

Figure 4 illustrates the formation of the error signal of bootstrap learning. The shaded areas of Figure 4 represent the error, the difference between the output q_k and the sum $(\text{SUM})_k$:

$$e_k = SGN((\text{SUM})_k) - (\text{SUM})_k. \tag{3}$$

The polarities of the error are indicated in the shaded areas. This is unsupervised learning, comprised of the LMS algorithm of Equation (1) and the error of equation (3).

When the error is zero, no adaptation takes place. In Figure 4, one can see that there are three different values of (SUM) where the error is zero. These are the three equilibrium points. The point at the origin is an unstable equilibrium point. The other two equilibrium points are stable. Some of the input patterns will produce sums that gravitate toward the positive stable equilibrium point, while the other input patterns produce sums that gravitate toward the negative stable equilibrium point. The arrows indicate the directions of change to the sum that would occur as a result of adaptation. All input patterns will become classified as either positive or negative when the adaptation process converges. If the training patterns were linearly independent, the neuron outputs will be binary, +1 or −1.

IV. Robert Lucky's Adaptive Equalization, from the 1960's

In the early 1960's, as Widrow's group at Stanford was developing bootstrap learning, at the same time, independently, a project at Bell laboratories led by Robert W. Lucky was developing an adaptive equalizer for digital data transmission over telephone lines [9][10]. His adaptive algorithm incorporated what he called "decision directed learning", which has similarities to bootstrap learning.

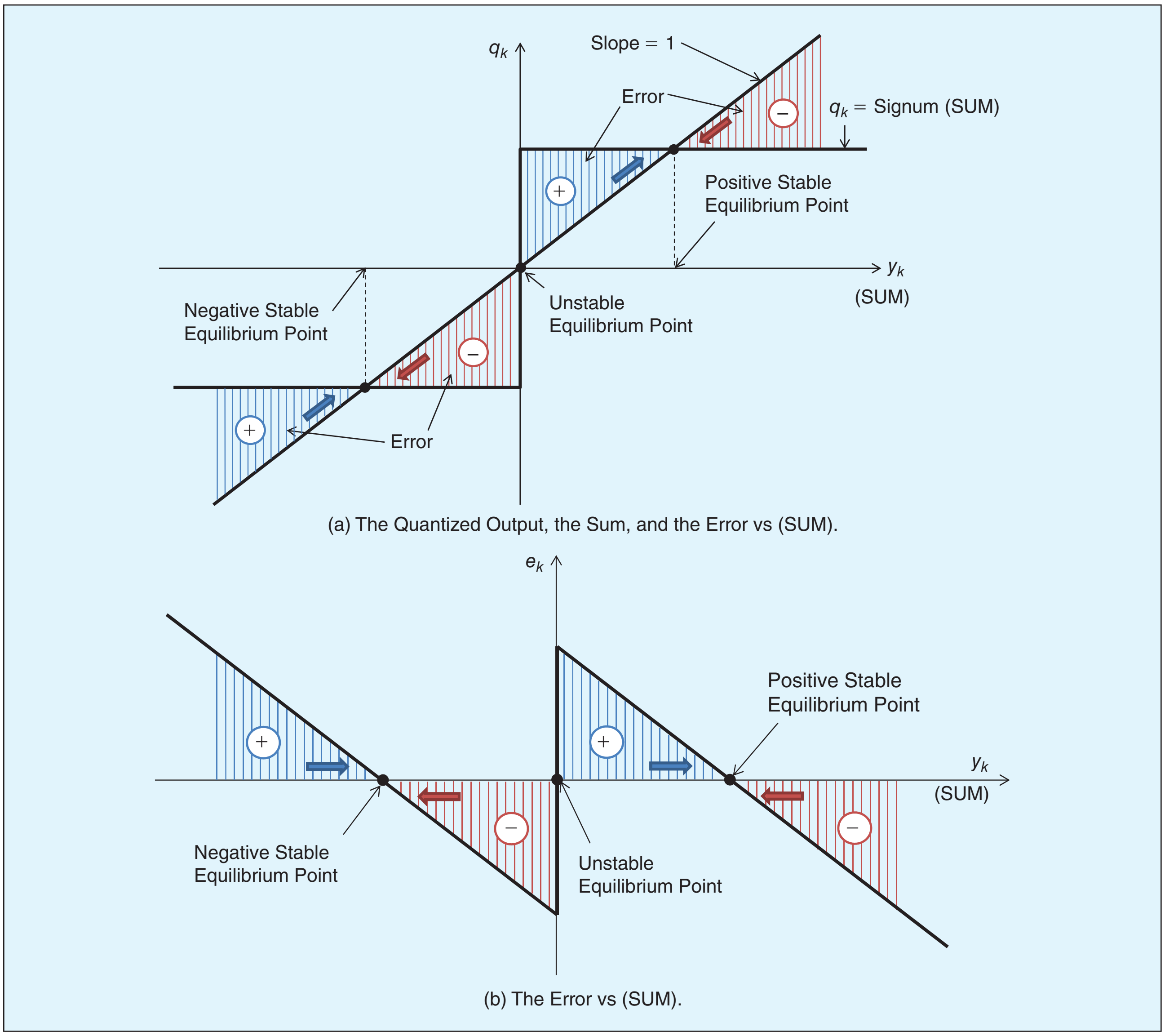
Lucky's work turned out to be of extraordinary significance. He was using an adaptive algorithm to adjust the weights of a transversal digital filter for data transmission over telephone lines. The invention of his adaptive equalizer ushered in the era of high speed digital data transmission.

Telephone channels ideally would have a bandwidth uniform from 0 Hz to 3 kHz, and a linear phase characteristic whose slope would correspond to the bulk delay of the channel. Real telephone channels do not respond down to zero frequency, are not flat in the passband, do not cut off perfectly at 3 kHz, and do not have linear phase characteristics. Real telephone channels were originally designed for analog telephony, not for digital data transmission. These channels are now used for both purposes.

Binary data can be sent by transmitting sharp positive and negative impulses into the channel. A positive pulse is a ONE, a negative pulse is a ZERO. If the channel were ideal, each impulse would cause a sinc function response at the receiving end of the channel. When transmitting data pulses at the Nyquist rate for the channel, a superposition of sinc functions would appear at the receiving end. Sampling or strobing the signal at the receiving end at the Nyquist rate and adjusting the timing of the strobe to sample at the peak magnitude of a sinc function, it would be possible to recover the exact binary data stream as it was transmitted. The reason is that when one of the sinc functions has a magnitude peak, all the neighboring sinc functions would be having zero crossings and would not interfere with the sensing of an individual sinc function. There would be no "intersymbol interference," and perfect transmission at the Nyquist rate would be possible (assuming low noise, which is quite realistic for land lines).

The transfer function of a real telephone channel is not ideal and the impulse response is not a perfect sinc function with uniformly spaced zero crossings. At the Nyquist rate, intersymbol interference would happen. To prevent this, Lucky's idea was to filter the received signal so that the transfer function of the cascade of the telephone channel and an equalization filter at the receiving end would closely approximate the ideal transfer function with a sinc-function impulse response. Since every telephone channel has its own "personality" and can change slowly over time, the equalizing filter would need to be adaptive.

Figure 5 shows a block diagram of a system that is similar to Lucky's original equalizer. Binary data are transmitted at the Nyquist rate as positive and negative pulses into a telephone channel. At the receiving end, the channel output is



q_k
Slope = 1
Error
q_k = Signum (SUM)
+
−
Positive Stable Equilibrium Point
y_k
(SUM)
Negative Stable Equilibrium Point
Unstable Equilibrium Point
Error
(a) The Quantized Output, the Sum, and the Error vs (SUM).
e_k
Positive Stable Equilibrium Point
y_k
(SUM)
Negative Stable Equilibrium Point
Unstable Equilibrium Point
(b) The Error vs (SUM).

FIGURE 4 Bootstrap learning.

inputted to a tapped delay line with variable weights connected to the taps. The weighted signals are summed. The delay line, weights, and summer comprise an adaptive transversal filter. The weights are given initial conditions. All weights are set to zero except for the first weight, which is set to the value of one. Initially, there is no filtering and, assuming that the telephone channel is not highly distorting, the summed signal will essentially be a superposition of sinc functions separated with Nyquist spacing. At the times when the sinc pulses have peak magnitudes, the quantized output of the signum will be a binary sequence that is a replica of the transmitted binary data. The quantized output will be the correct output sequence. The quantized output can accordingly be taken as the desired output, and the difference between the quantized output and the summed signal will be the error signal for adaptive purposes. This difference will only be usable as the error signal at times when the sinc functions are at their peak magnitudes. A strobe pulse samples the error signal at the Nyquist rate, timed to the sinc function peak, and the error samples are used to adapt the weights. The output decision is taken to be the desired response. Thus, decision-directed learning results.

With some channel distortion, the signum output will not always be correct, at peak times. The equalizer can start with an error rate of 25% and automatically converge to an error rate of perhaps 10^{-8}, depending on the noise level in the channel.

Figure 6(a) shows the output of a telephone channel without equalization. Figure 6(b) shows the same channel with the same dataflow after adaptive equalization. These patterns are created by overlaying cycles of the waveform before and after equalization. The effect of equalization is to make the impulse responses approximate sinc functions. When this is done, an "eye pattern" as in Figure 6(b) results. Opening the eye is the purpose of adaptation. With the eye open and when

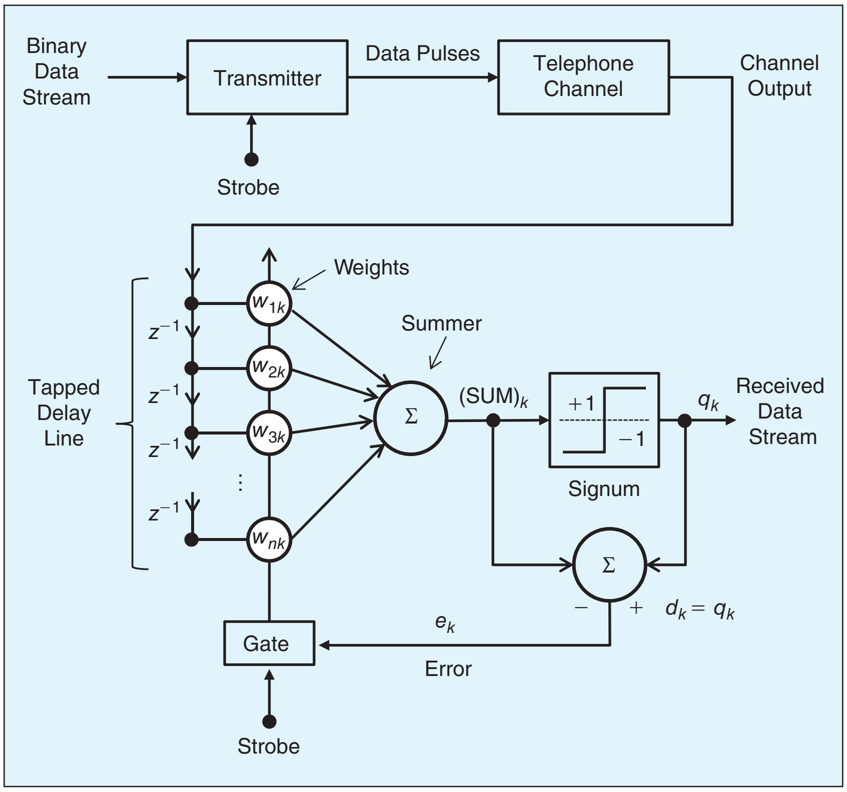


FIGURE 5 Decision-directed learning for channel equalization.

transmitter begins with a known pseudorandom sequence of pulses, a world standard known to the receiver. During the handshake, the receiver knows the desired responses and adapts accordingly. This is supervised learning. The receiving adaptive filter converges and now, actual data transmission can commence. Decision directed equalization takes over and maintains the proper equalization for the channel by learning with the signals of the channel. This is unsupervised learning. If the channel is stationary or only changes slowly, the adaptive algorithm will maintain the equalization. However, fast changes could cause the adaptive filter to get out of lock. There will be a "dropout," and the transmission will need to be reinitiated.

Adaptive equalization has been the major application for unsupervised learning since the 1960's. The next section describes a new form of unsupervised learning, bootstrap learning for the weights of a single neuron with a sigmoidal activation function. The sigmoidal function is closer to being "biologically correct" than the signum function of Figures 1, 3, and 5.

sampling at the appropriate time, ones and zeros are easily discerned. The adaptive algorithm keeps the ones tightly clustered together and well separated from the zeros which are also tightly clustered together. The ones and zeros comprise two distinct clusters. This is decision directed learning, similar to bootstrap learning.

In the present day, digital communication begins with a "handshake" by the transmitting and receiving parties. The

V. Bootstrap Learning with a Sigmoidal Neuron

Figure 7 is a diagram of a sigmoidal neuron whose weights are trained with bootstrap learning. The learning process of Figure 7 is characterized by the following error signal:

$$\text{error} = e_k = SGM((\mathrm{SUM})_k) - \gamma \cdot (\mathrm{SUM})_k. \tag{4}$$

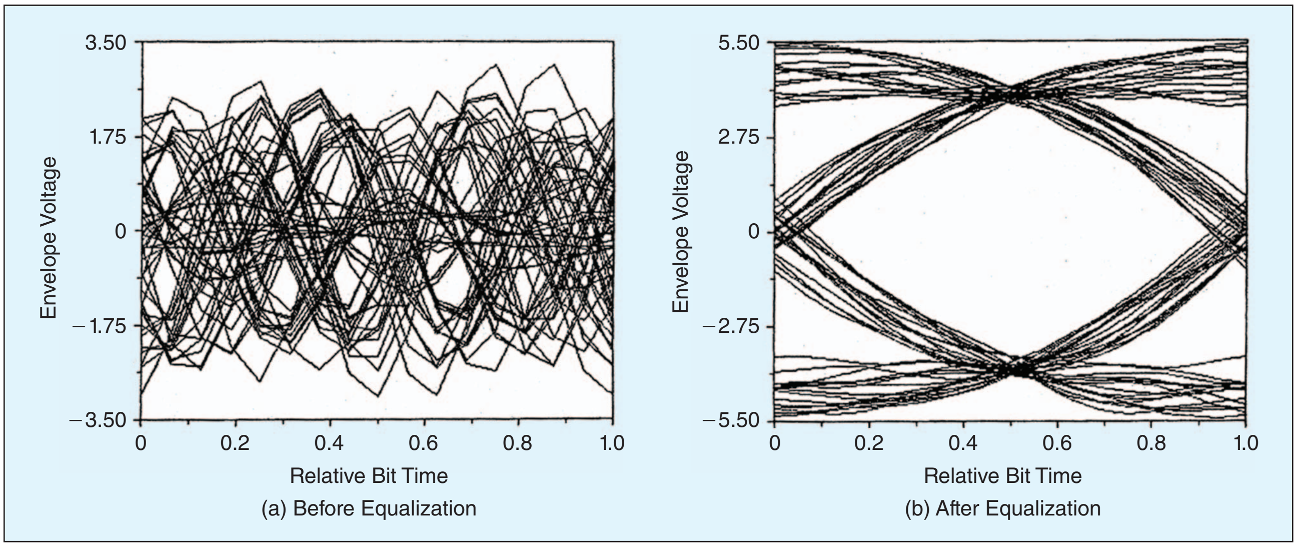


FIGURE 6 Eye patterns produced by overlaying cycles of the received waveform. (a) before equalization. (b) after equalization. Figure 10.14 of Widrow and Sterns [6], courtesy of Prentice Hall.

The sigmoidal function is represented by $SGM(\cdot)$. Input pattern vectors are weighted, summed, and then applied to the sigmoidal function to provide the output signal, $(\text{OUT})_k$. The weights are initially randomized, then adaptation is performed using the LMS algorithm (1), with an error signal given by (4).

Insight into the behavior of the form of bootstrap learning of Figure 7 can be gained by inspection of Figure 8. The shaded areas indicate the error, which is the difference between the sigmoidal output and the sum multiplied by the constant γ, in accordance with equation (4). As illustrated in the figure, the slope of the sigmoid at the origin has a value of 1, and the straight line has a slope of γ. These values are not critical, as long as the slope of the straight line is less than the initial slope of the sigmoid. The polarity of the error signal is indicated as $+$ or $-$ on the shaded areas. There are two stable equilibrium points, a positive one and a negative one, where

$$SGM((\text{SUM})) = \gamma \cdot (\text{SUM}), \tag{5}$$

and the error is zero. An unstable equilibrium point exists where $(\text{SUM}) = 0$.

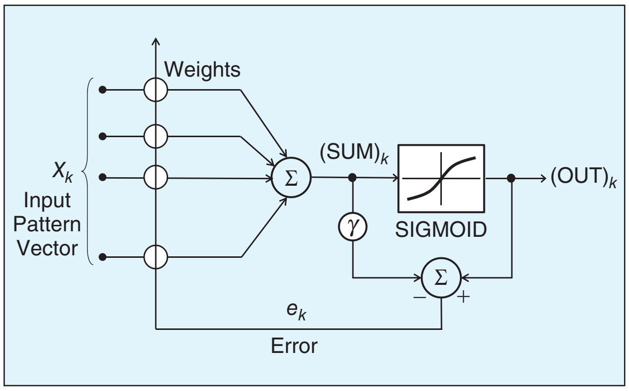


FIGURE 7 A sigmoidal neuron trained with bootstrap learning.

When (SUM) is positive, and SGM ((SUM)) is greater than $\gamma \cdot (\text{SUM})$, the error will be positive and the LMS algorithm will adapt the weights in order to increase (SUM) up toward the positive equilibrium point. When (SUM) is positive and $\gamma \cdot (\text{SUM})$ is greater than $SGM((\text{SUM}))$, the error

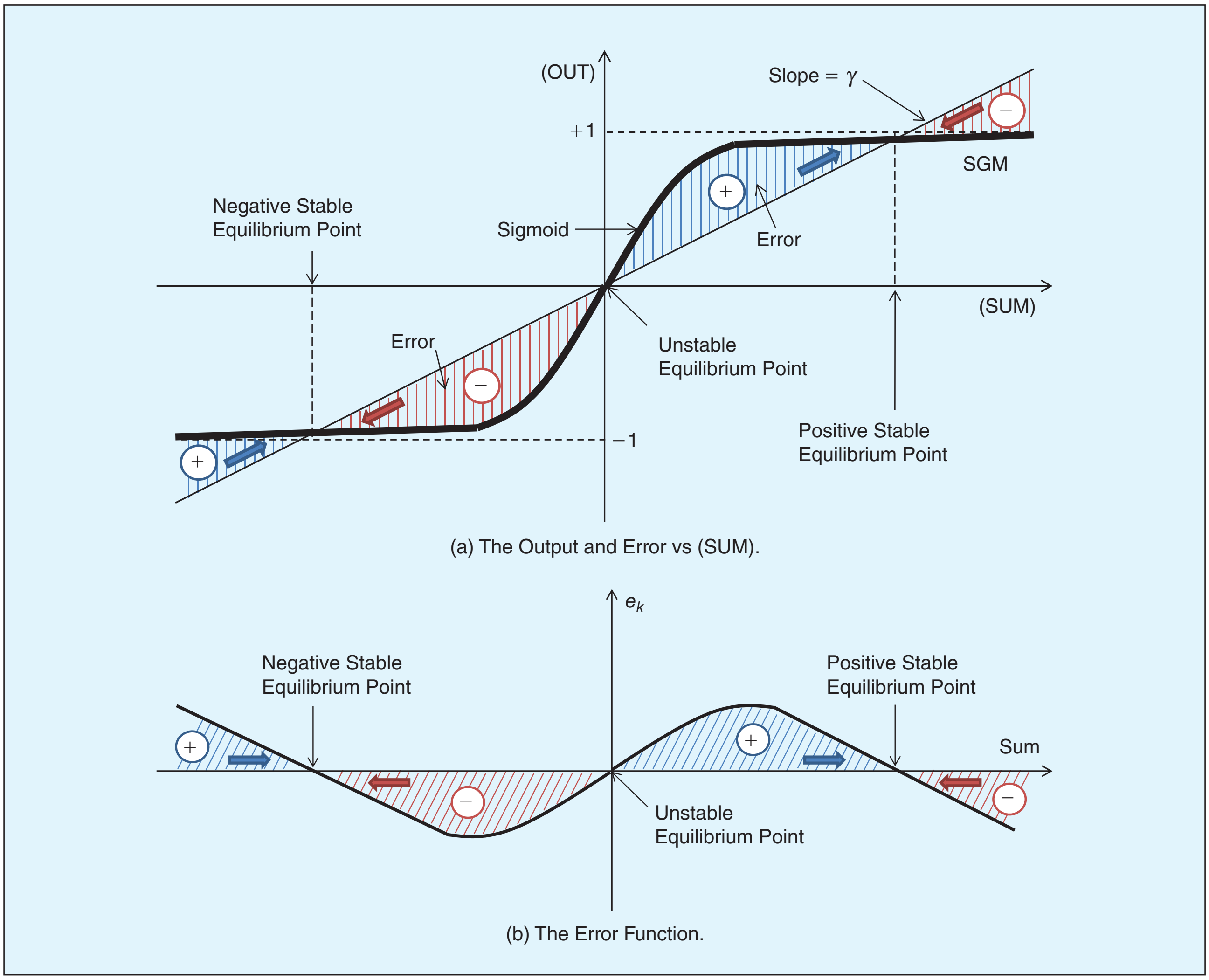


FIGURE 8 The error of the sigmoidal neuron trained with bootstrap learning.

The purpose of this paper is to introduce a new learning algorithm that we call Hebbian- LMS. It is an implementation of Hebb's teaching by means of the LMS algorithm of Widrow and Hoff.

will reverse and will be negative and the LMS algorithm will adapt the weights in order to decrease (SUM) toward the positive equilibrium point. The opposite of all these will take place when (SUM) is negative.

When the training patterns are linearly independent and their number is less than or equal to the number of weights, all input patterns will have outputs exactly at either the positive or negative equilibrium point, upon convergence of the LMS algorithm. The "LMS capacity" or "capacity" of the single neuron can be defined as being equal to the number of weights. When the number of training patterns is greater than capacity, the LMS algorithm will cause the pattern responses to cluster, some near the positive stable equilibrium point and some near the negative stable equilibrium point. The error corresponding to each input pattern will generally be small but not zero, and the mean square of the errors averaged over the training patterns will be minimized by LMS. The LMS algorithm maintains stable control and prevents saturation of the sigmoid and of the weights. The training patterns divide themselves into two classes without supervision. Clustering of the values of (SUM) at the positive and negative equilibrium points as a result of LMS training will prevent the values of (SUM) from increasing without bound.

VI. Bootstrap Learning with a More "Biologically Correct" Sigmoidal Neuron

The inputs to the weights of the sigmoidal neuron in Figure 7 could be positive or negative, the weights could be positive or negative, and the outputs could be positive or negative. As a biological model, this would not be satisfactory. In the biological world, an input signal coming from a presynaptic neuron must have positive values (presynaptic neuron firing at a given rate) or have a value of zero (presynaptic neuron not firing). Some synapses are excitatory, some inhibitory. They have different neurotransmitter chemistries. The inhibitory inputs to the postsynaptic neuron are subtracted from the excitatory inputs to form (SUM) in the cell body of the postsynaptic neuron. Biological weights or synapses behave like variable attenuators and can only have positive weight values. The output of the postsynaptic neuron can only be zero (neuron not firing) or positive (neuron firing) corresponding to (SUM) being negative or positive. The postsynaptic neuron and its synapses diagrammed in Figure 9 have the indicated properties and are capable of learning exactly like the neuron and synapses in Figure 7. The LMS algorithm of equation (1) will operate as usual with positive excitatory inputs or negative inhibitory inputs. For LMS, these are equivalents of positive or negative components of the input pattern vector.

LMS will allow the weight values to remain within their natural positive range even if adaptation caused a weight value to be pushed to one of its limits. Subsequent adaptation could bring the weight value away from the limit and into its more normal range, or it could remain saturated. Saturation would not necessarily be permanent (as would occur with Hebb's original learning rule).

The neuron and its synapses in Figure 9 are identical to those of Figure 7, except that the final output is obtained from a "half sigmoid." So the output will be positive, the weights will be positive, and some of the weighted inputs will be excitatory, some inhibitory, equivalent to positive or negative inputs. The (SUM) could be negative or positive.

The training processes for the neurons and their synapses of Figure 7 and Figure 9 are identical, with identical stabilization points.

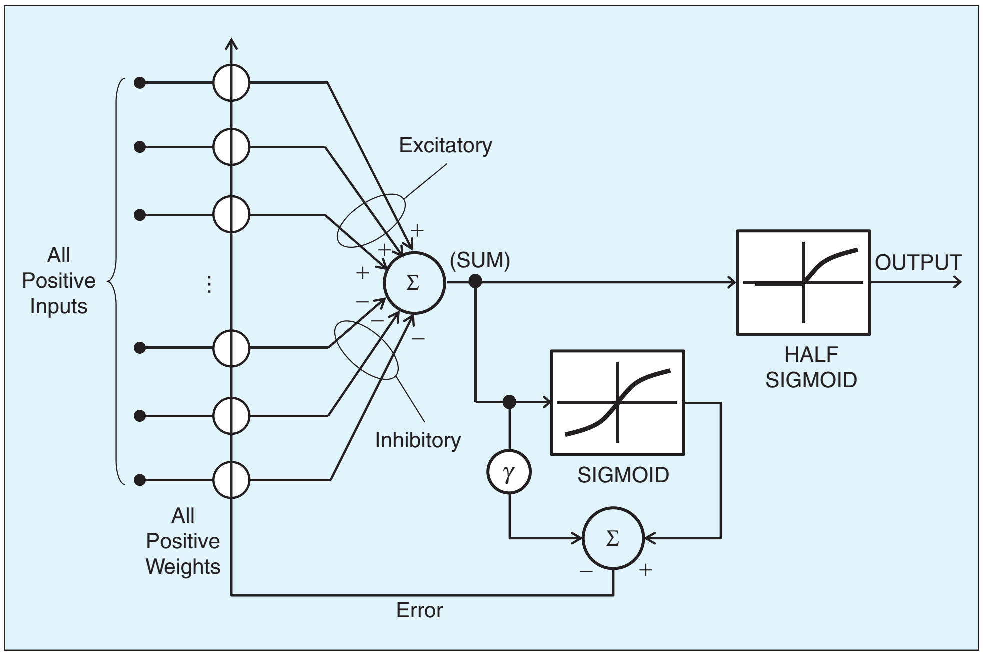


FIGURE 9 A postsynaptic neuron with excitatory and inhibitory inputs and all positive weights trained with Hebbian-LMS learning. All outputs are positive. The (SUM) could be positive or negative.

The error signals are obtained in the same manner, and the formation of the error for the neuron and synapses of Figure 9 is illustrated in Figure 10. The error is once again given by Equation (4). The final output, the output of the "half sigmoid", is indicated in Figure 10. Figure 10(a) shows the error and output. Figure 10(b) shows the error function. When the (SUM) is negative, the neuron does not fire and the output is zero. When the (SUM) is positive, the firing rate, the neuron output, is a sigmoidal function of the (SUM). The learning algorithm is

$$\begin{aligned} W_{k+1} &= W_k + 2\mu e_k X_k, \\ e_k &= SGM(X_k^T W_k) - \gamma X_k^T W_k = SGM((\text{SUM})_k) \\ &\quad - \gamma \cdot (\text{SUM})_k. \end{aligned} \tag{6}$$

Equation (6) is a form of the Hebbian-LMS algorithm.

The LMS algorithm requires that all inputs to the summer be summed, not some added and some subtracted as in Figure 9. Accordingly, when forming the X-vector, its excitatory components are taken directly as the outputs of the correspondingly connected presynaptic neurons while its inhibitory components are taken as the negative of the outputs of the correspondingly connected presynaptic neurons. Performing the Hebbian-LMS algorithm of equation (6), learning will then take place in accord with the diagram of Figure 9.

With this algorithm, there is no inputted desired response with each input pattern X_k. Learning is unsupervised. The parameter μ controls stability and speed of convergence, as is the case for the LMS algorithm. The parameter γ has the value of 1/2 in the diagram of Figure 10, but could have any positive value less than the initial slope of the sigmoid function.

$$0 < \gamma < \frac{d}{d\xi} SGM(\xi)\Big|_{\xi=0}. \tag{7}$$

The neuron output signal is given by:

$$(\text{OUT})_k = \begin{cases} SGM(X_k^T W_k), & X_k^T W_k \geq 0 \\ 0, & X_k^T W_k < 0 \end{cases}. \tag{8}$$

Equation (6) represents the training procedure for the weights (synapses). Equation (8) describes the signal flow through the neuron. Simulation results are represented in Figure 11.

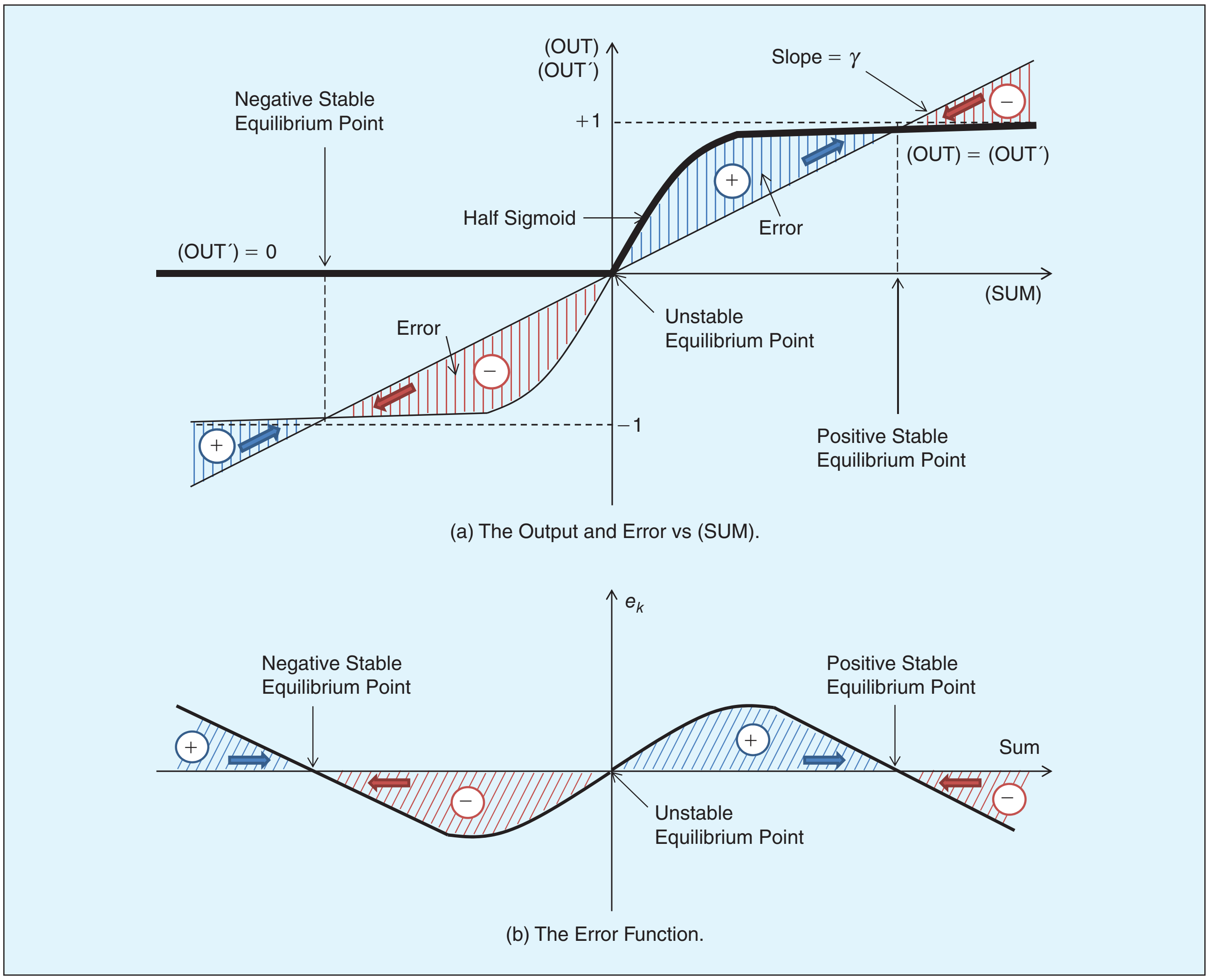

(OUT)
(OUT´)
Slope = γ
Negative Stable Equilibrium Point
+1
(OUT) = (OUT´)
Half Sigmoid
Error
(OUT´) = 0
(SUM)
Error
Unstable Equilibrium Point
−1
Positive Stable Equilibrium Point
(a) The Output and Error vs (SUM).
e_k
Negative Stable Equilibrium Point
Positive Stable Equilibrium Point
Sum
Unstable Equilibrium Point
(b) The Error Function.

FIGURE 10 The error of the sigmoidal neuron with rectified output, trained with bootstrap learning.

Computer simulation was performed to demonstrate learning and clustering by the neuron and synapses of Figure 9. Initial values for the weights were chosen randomly, independently, with uniform probability between 0 and 1. There were 50 excitatory and 50 inhibitory weights. There were 50 training patterns whose vector components were chosen randomly, independently, with uniform probability between 0 and 1. Initially some of the input patterns produced positive (SUM) values, indicated in Figure 11(a) by blue crosses, and the remaining patterns produced negative (SUM) values, indicated in Figure 11(a) by red crosses. After 100 iterations, some of the reds and blues have changed sides, as seen in Figure 11(b). After 2000 iterations, as seen in Figure 11(c), clusters have began to form and membership of the clusters has stabilized. There are no responses near zero. After 5000 iterations, tight clusters have formed as shown in Figure 11(d). At the neuron output, the output of the half sigmoid, the responses will be binary, 0's and approximate 1's. Upon convergence, the patterns selected to become 1's or those selected to become 0's are strongly influenced by the

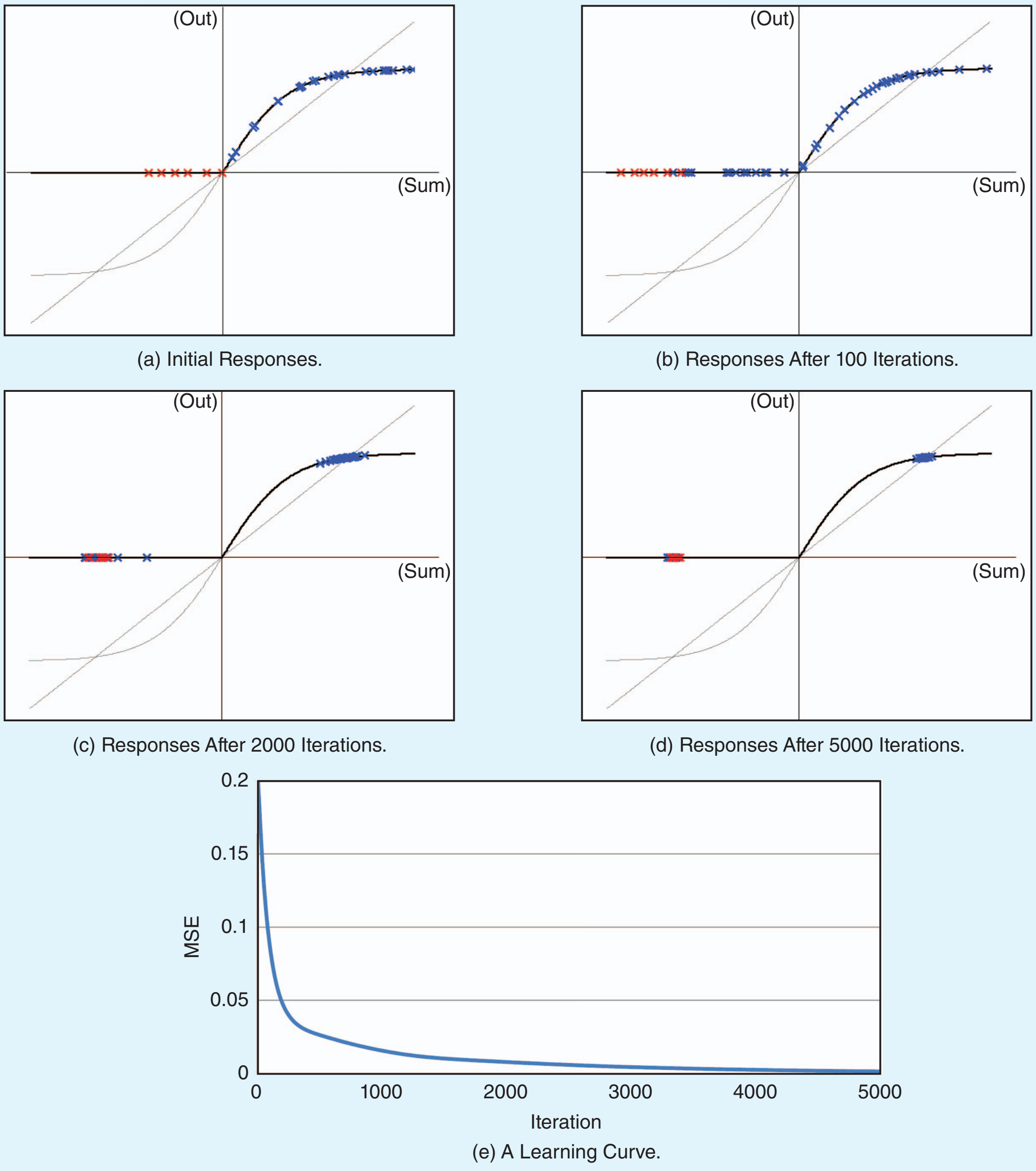


FIGURE 11 A learning experiment.

random initial conditions, but not absolutely determined by initial conditions. The patterns would be classified very differently with different initial weights.

A learning curve, mean square error as a function of the number of iterations, is shown in Figure 11(e). When using a supervised LMS algorithm, the learning curve is known to be a sum of exponential components [6]. With unsupervised LMS, the theory has not yet been developed. The nature of this learning curve and the speed of convergence have only been studied empirically.

The method for training a neuron and synapses described above can be used for training neural networks. The networks could be layered structures or could be interconnected in random configurations like a "rat's nest." Hebbian-LMS will work with all such configurations. For simplicity, consider a layered network like the one shown in Figure 12.

The example of Figure 12 is a fully connected feedforward network. A set of input vectors are applied repetitively, periodically, or in random sequence. All of the synaptic weights are set randomly initially, and adaptation commences by applying the Hebbian-LMS algorithm independently to all the neurons and their input synapses. The learning process is totally decentralized. In nature, all of the synapses would be adapted simultaneously, so the speed of convergence for the entire network would not be much less than that of the single neuron and its input synapses. If the first layer were trained until convergence, then the second layer were trained until convergence, then the third layer were trained until convergence, the convergence for this three-layer example would be three times slower than that of a single neuron and its input synapses. Training the network all at once would be even faster with totally parallel operation.

If the input patterns were linearly independent vectors, the output of the first layer neurons would be binary after convergence. Since the input synapses of each of the first layer neurons were set randomly and independently, the outputs of the first layer neurons would be different from neuron-to-neuron. After convergence, the outputs of the second layer neurons would also be binary, but different from the outputs of the first layer. The outputs of the third layer will also be binary after convergence.

If the input patterns are not linearly independent vectors, the outputs of the first layer neurons will not be purely binary. The outputs of the second layer will be closer to binary. The outputs of the third layer will be even closer to binary. The number of training patterns that is equal to the number of input synapses of each of the output layer neurons is the capacity of the network. It is shown in [11] that when applying patterns that are distinct but not necessarily linear independent to a nonlinear process such as layers of a neural network, the outputs of the layers will be distinct and linearly independent. If the number of training patterns is less than or equal to the network capacity, the inputs to the output layer synapses will be linearly independent and the outputs of the output layer neurons will be perfectly binary. If the number of training patterns exceeds the network capacity, the network output will be 0's and "fuzzy" 1's, close to binary.

If one were to take one of the trained in vectors and place a large cloud of vectors randomly disposed in a cluster about it, inputting all the vectors in the cluster without further training would result in identical binary output vectors at the output layer. This will be true as long as the diameter if the cluster is not "too large." How large this would be depends on the number and disposition of the other training vectors. The result is that noisy or distorted input patterns in a cluster can be identified as equivalent to the associated training pattern. The network determines a unique output pattern, a binary representation for each training pattern and the patterns in its cluster. This is a useful property for pattern classification.

With unsupervised learning, each cluster "chooses" its own binary output representation. The number of clusters that the network can resolve is equal to the network capacity, equal to the number of weights of each of the neurons of the output layer.

Another application is the following. Given a network trained by Hebbian-LMS. Let the weights be fixed. Inputting a pattern from one of the clusters of one of the training patterns will result in a binary output vector. The sum of the squares of the errors of the output neurons will be close to zero. Now, inputting a pattern not close to any of the training patterns will result in an output vector that will not be binary and the sum of the squares of the errors of the output neurons will be large.

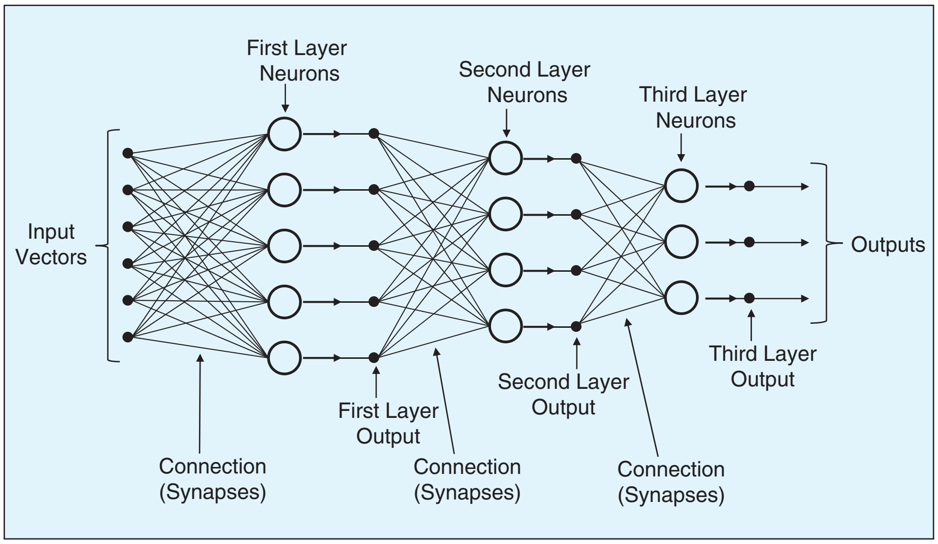


FIGURE 12 An example of a layered neural network.

The Hebbian-LMS algorithm exhibits homeostasis about the two equilibrium points, caused by reversal of the error signal at these equilibrium points.

So, if the input pattern is close to a training pattern, the output error will be close to zero. If the input pattern is distinct from all the training patterns, the output error will be large. One could use this when one is not asking the neural network to classify an input pattern, merely to indicate if the input pattern has been trained in, i.e., seen before or not, deja vu, yes or no? This could be used as a critical element of a cognitive memory system [12].

In yet another application, a multi-layer neural network could be trained using both supervised and unsupervised methods. The hidden layers could be trained with Hebbian-LMS and the output layer could be trained with the original LMS algorithm.

An individual input cluster would produce an individual binary "word" at the output of the final hidden layer. The output layer could be trained with a one-out-of-many code. The output neuron with the largest (SUM) would be identified as representing the class of the cluster of the input pattern.

A three layer purely Hebbian-LMS network was simulated with 100 neurons in each layer. The input patterns were 50-dimensional, and the network outputs, binary after training, were 100-bit binary numbers. A set of training patterns was generated as follows. Ten random vectors were used as representing ten clusters. Clusters were formed as clouds about the ten original vectors. Each cloud contained 100 randomly disposed points. The ten 50-dimensional clusters are shown in Figure 13(a) in two dimensions. The axes were chosen as the first two principal components.

All 1000 vectors were trained. The network was not "told" which vector belonged to which of the clusters. The 1000 input vectors were not labeled in any way. After convergence, the network produced 100-bit output words for each input vector. Ten distinct 100-bit output words were observed, each corresponding to one of the clouds. For a given 100-bit output word, all input vectors that caused that output word were given a specific color. The colored input points are shown in Figure 13(b). The colored points associate exactly as they did in the input clouds.

The uncolored points were trained into the network and they were "colored by the network." The network automatically produced unique representations for each of the clouds. This was a relatively easy problem since the number of clouds, 10, was much less than the network capacity, 100.

Figure 14 illustrates how Hebbian-LMS creates binary outputs after the above training with the 1000 patterns. One of the neurons in the output layer was selected and histograms were constructed for its (SUM) before and after training, and for its half-sigmoid output before and after training. The histograms show that, before training, the histogram of the (SUM) was not binary and the histogram of the half-sigmoid output appears to be almost binary but it is not so. Observing the colors, one can see that some of the clusters were split apart. After training, the histogram of the half-sigmoid output shows the clusters to be intact and all together.

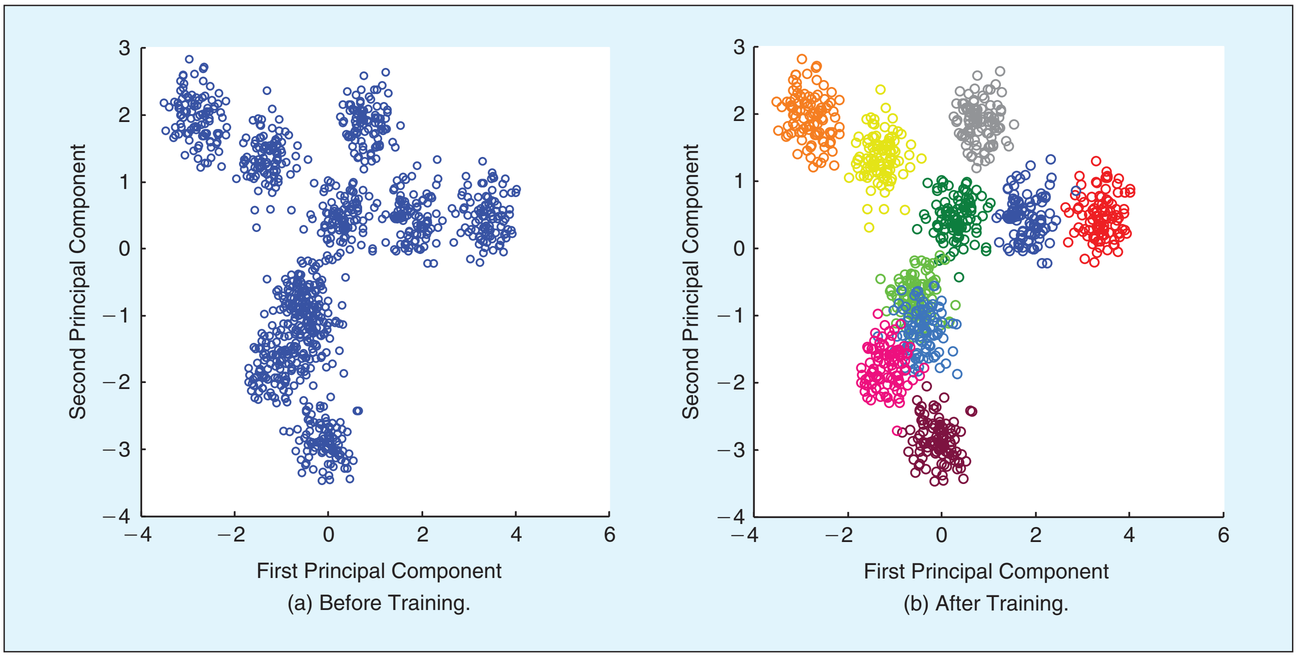


FIGURE 13 50-dimensional input vectors plotted along the first two principal components.

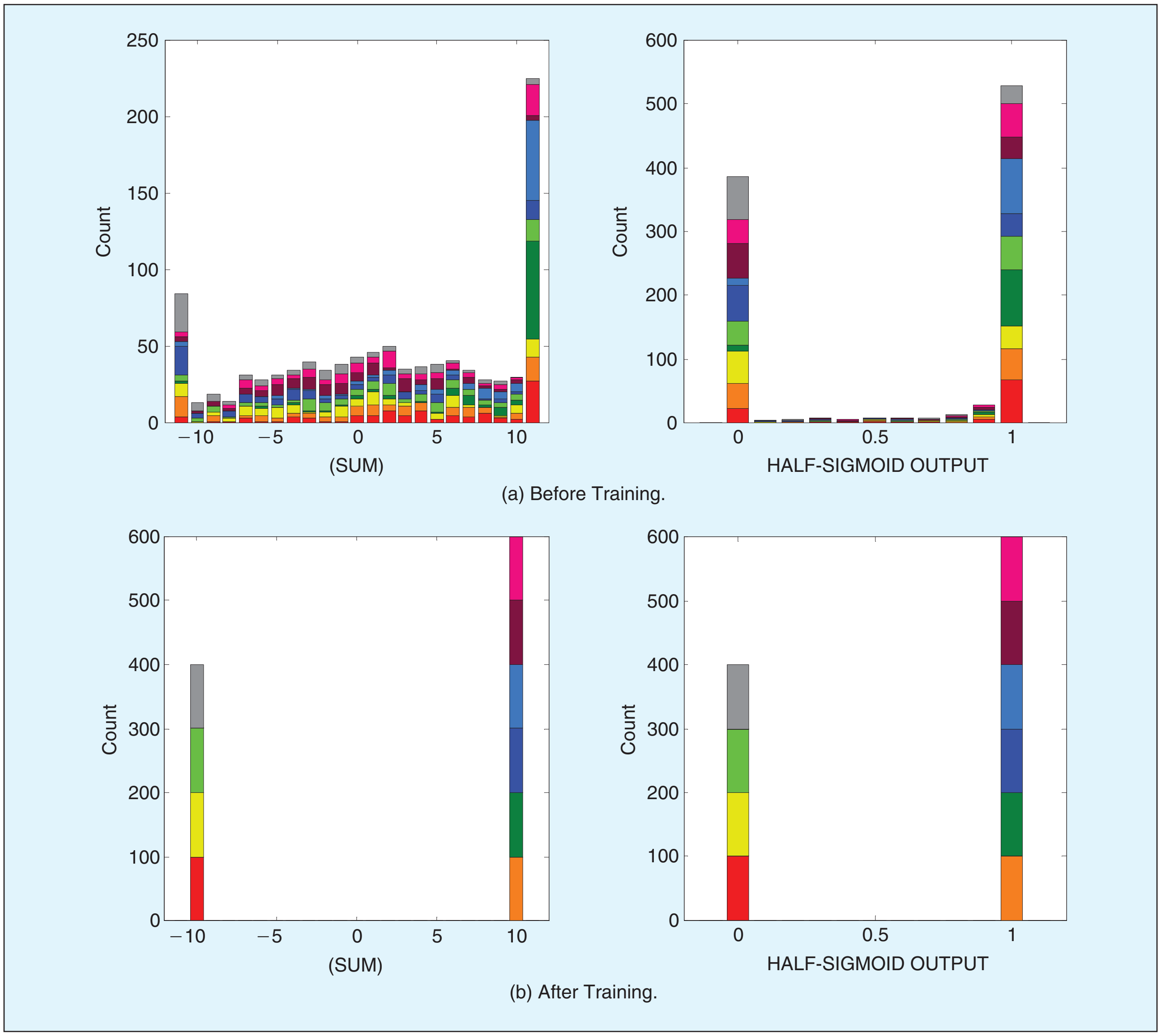


FIGURE 14 Histogram of responses of a selected neuron in the output layer of a three layer Hebbian-LMS network.

VII. Other Clustering Algorithms

A. K-Means Clustering

The K-means clustering algorithm [13][14] is one of the most simple and basic clustering algorithms and has many variations. It is an algorithm to find K centroids and to partition an input dataset into K clusters based on the distances between each input instance and K centroids. This algorithm is usually fast to converge, relatively simple to compute, and effective in many cases. However, the number of clusters "K" is unknown in the beginning and it has to be determined by heuristic methods.

B. Expectation-Maximization (EM) Algorithm

EM is an algorithm for finding maximum likelihood estimates of parameters in a statistical model [15]. When the model depends on hidden latent variables, this algorithm iteratively finds a local maximum likelihood solution by repeating two steps: E-step and M-step. Its convergence is well known [16] and the K-means clustering algorithm is a special case of the EM algorithm. Same as with the K-means algorithm, the number of clusters has to be determined prior to applying this algorithm.

C. DBSCAN Algorithm

Density-based spatial clustering of application with noise (DBSCAN) is one of the well-known density-based clustering algorithms [17]. It repeats the process of grouping close points together until there is no point left to group. After grouping, the points that do not belong to any group become outliers and are labeled as noise. In spite of the popularity and effectiveness of this algorithm, its performance significantly depends on two threshold variables that determine the grouping.

D. Comparison Between Clustering Algorithms

We have tested several clustering methods with artificial datasets such as the multivariate Gaussian random dataset and some of the datasets from the UCI Machine Learning Repository [18]. Overall performance of clustering with the Hebbian-LMS algorithm is comparable to the results obtained with the existing algorithms. These existing algorithms require us to determine model parameters manually or to use heuristic methods. Hebbian-LMS requires us to choose a value of the parameter μ, the learning step. In most cases, this choice is not critical and can be made like choosing μ for supervised LMS as described in detail in [6].

VIII. A General Hebbian-LMS Algorithm

The Hebbian-LMS algorithm applied to the neuron and synapses of Figure 9 results in a nicely working clustering algorithm, as demonstrated above, but its error signal may not correspond exactly to nature's error signal. How nature generates the error signal will be discussed below.

It is possible to generate the error signal in many different ways. The error signal is a function of the (SUM) signal. A most general form of Hebbian-LMS is diagrammed in Figure 15. The learning algorithm can be expressed as

$$W_{k+1} = W_k + 2\mu e_k X_k, \tag{9}$$

$$e_k = f((\text{SUM})_k) = f(X_k^T W_k). \tag{10}$$

The neuron output can be expressed as

$$(\text{OUT})_k = \begin{cases} SGM((\text{SUM})_k) = SGM(X_k^T W_k), & (\text{SUM})_k > 0 \\ 0, & (\text{SUM})_k < 0 \end{cases}. \tag{11}$$

For this neuron and its synapses to adapt and learn in a natural way, the error function f((SUM)) would need to be nature's error function. To pursue this further, it is necessary to incorporate knowledge of how synapses work, how they carry signals from one neuron to another, and how synaptic weight change is effected.

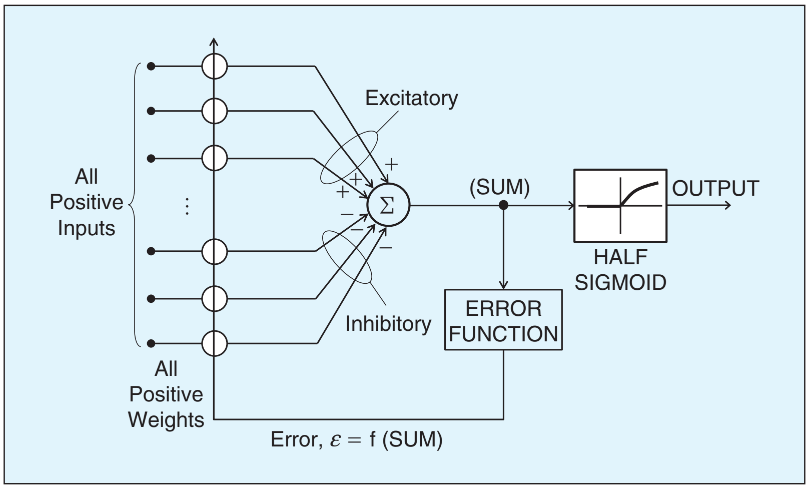


FIGURE 15 A general form of Hebbian-LMS.

IX. The Synapse

The connection linking neuron to neuron is the synapse. Signal flows in one direction, from the presynaptic neuron to the postsynaptic neuron via the synapse which acts as a variable attenuator. A simplified diagram of a synapse is shown in Figure 16(a) [19]. As an element of neural circuits, it is a "two-terminal device".

There is a 0.02 micron gap between the presynaptic side and the postsynaptic side of the synapse which is called the synaptic cleft. When the presynaptic neuron fires, a protein called a neurotransmitter is injected into the cleft. Each activation pulse generated by the presynaptic neuron causes a finite amount of neurotransmitter to be injected into the cleft. The neurotransmitter lasts only for a very short time, some being re-absorbed and some diffusing away. The output signal of the presynaptic neuron is proportional to its firing rate. Thus, the average concentration of neurotransmitter in the cleft is proportional to the presynaptic neuron's firing rate.

Some of the neurotransmitter molecules attach to receptors located on the postsynaptic side of the cleft. The effect of this on the postsynaptic neuron is either excitatory or inhibitory, depending on the nature of the synapses [19]–[23]. A synaptic effect results when neurotransmitter attaches to its receptors. The effect is proportional to the average amount of neurotransmitter present and the number of receptors. Thus, the effect of the presynaptic neuron on the postsynaptic neuron is proportional to the presynaptic firing rate and the number of receptors present. The input signal to the synapse is the presynaptic firing rate, and the synaptic weight is proportional to the number of receptors. The weight or the synaptic "efficiency" described by Hebb is increased or decreased by increasing or decreasing the number of receptors. This can only occur when neurotransmitter is present [19]. Neurotransmitter is essential both as a signal carrier and as a facilitator for weight changing. A symbolic representation of the synapse is shown in Figure 16(b).

The effect of the action of a single synapse upon the postsynaptic neuron is actually quite small. Signals from thousands of synapses, some excitatory, some inhibitory, add in the postsynaptic neuron to create the (SUM) [19] [24]. If the (SUM) of the positive and negative inputs is below a threshold, the postsynaptic neuron will not fire and its output will be zero. If the (SUM) is greater than the threshold, the postsynaptic neuron will fire at a rate that increases with the magnitude of the (SUM) above the threshold. The threshold voltage within the postsynaptic neuron is a "resting potential" close to −70 millivolts. Summing in the postsynaptic neuron is accomplished by Kirchoff addition.

Learning and weight changing can only be done in the presence of neurotransmitter in the synaptic cleft. Thus, there will be no weight changing if the presynaptic neuron is not firing, i.e., if the input signal to the synapse is zero. If the presynaptic neuron is firing, there will be weight change. The number of receptors will gradually increase (up to a limit) if the postsynaptic neuron is firing, i.e., when the (SUM) of the postsynaptic neuron has a voltage above threshold. Then the synaptic membrane that the receptors are attached to will have a voltage above threshold, since this membrane is part of the postsynapitc neuron. See Figure 17. All this corresponds to Hebbian learning, firing together wiring together. Extending Hebb's rule, if the presynaptic neuron is firing and the postsynaptic neuron is not firing, the postsynaptic (SUM) will be negative and below the threshold, the membrane voltage will be negative and below the threshold, and the number of receptors will gradually decrease.

There is another mechanism having further control over the synaptic weight values, and it is called synaptic scaling [25]–[29]. This natural mechanism is implemented chemically for stability, to maintain the voltage of (SUM) within an approximate range about two set points. This is done by scaling up or down all of the synapses supplying signal to a given neuron. There is a positive set point and a negative one, and they turn out to be analogous to the equilibrium points shown in Figure 8. This kind of stabilization is called homeostasis and is a phenomenon of regularization that takes place all over living systems. The Hebbian-LMS algorithm exhibits homeostasis about the two equilibrium points, caused by reversal of the error signal at these equilibrium points. See Figure 8. Slow adaptation over thousands of adapt cycles, over hours of real time, results in homeostasis of the (SUM).

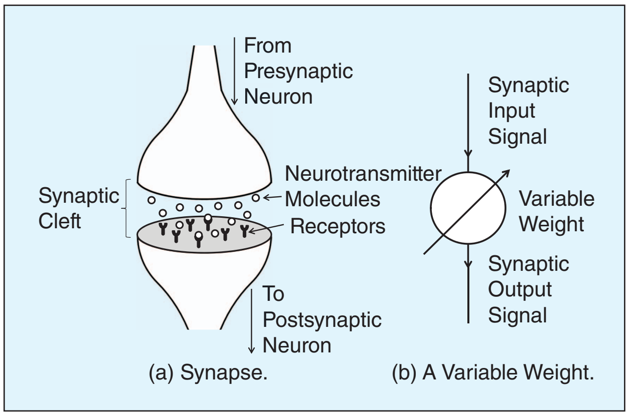


FIGURE 16 A synapse corresponding to a variable weight.

An exaggerated diagram of a neuron, dendrites, and a synapse is shown in Figure 17. This diagram suggests how the voltage of the (SUM) in the soma of the postsynaptic neuron can influence the voltage of the membrane.

Activation pulses are generated by a pulse generator in the soma of the postsynaptic neuron. The pulse generator is energized when the (SUM) exceeds the threshold. The pulse

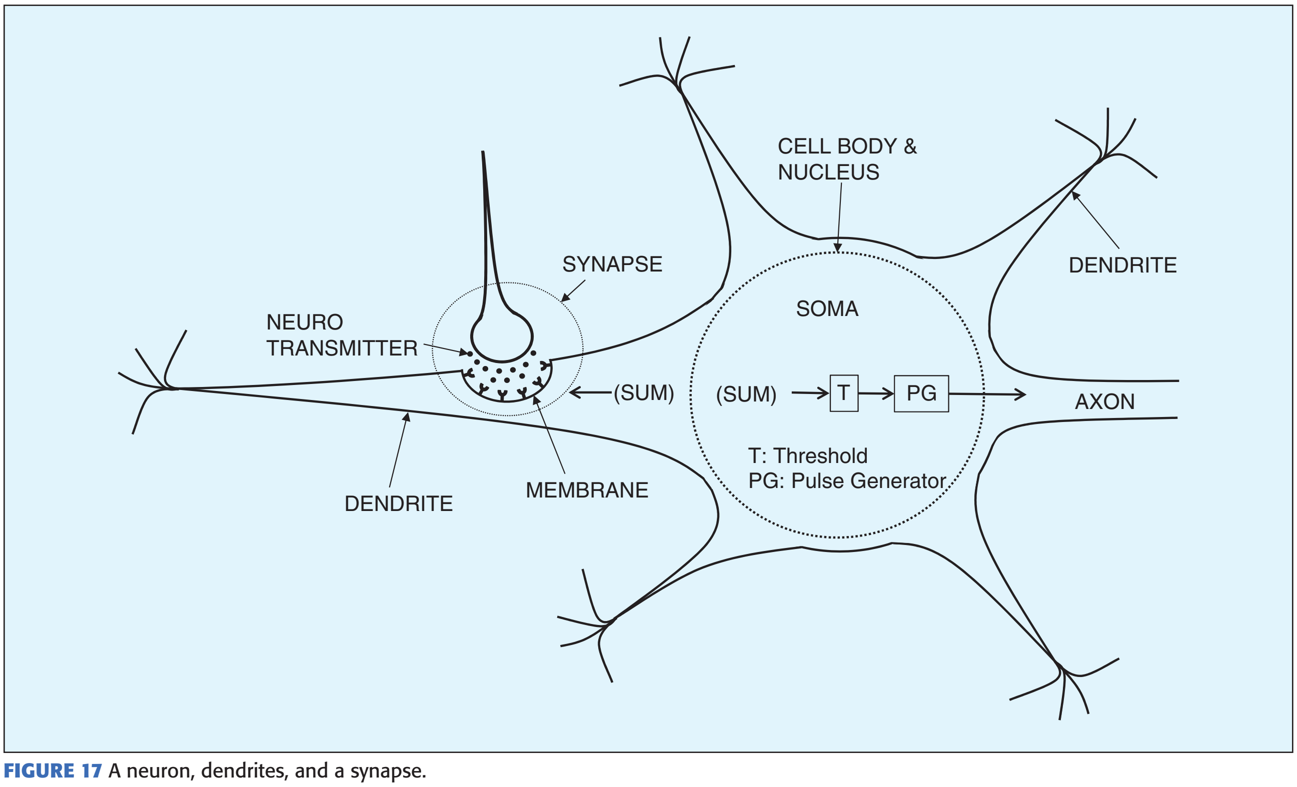


FIGURE 17 A neuron, dendrites, and a synapse.

The question remains, when nature performs learning in neural networks, is this done with an algorithm similar to Hebbian-LMS? No one knows for sure, but *"if it walks like a duck, quacks like a duck, and looks like a duck, maybe it is a duck."*

generator triggers the axon to generate electrochemical waves that carry the neuron's output signal. The firing rate of the pulse generator is controlled by the (SUM). The output signal of the neuron is its firing rate.

X. Postulates of Synaptic Plasticity

The above description of the synapse and its variability or plasticity is based on a study of the literature of the subject. The literature is not totally clear or consistent however. Experimental conditions of the various studies are not all the same, and the conclusions can differ. In Figure 18, a set of postulates of synaptic plasticity have been formulated representing a "majority opinion."

A group of researchers have developed learning algorithms called "anti-Hebbian learning [30]–[32]": "Fire together, unwire together." This is truly the case for inhibitory synapses. We call this in the above postulates an extension of Hebb's rule. Anti-Hebbian learning fits the postulates and is therefore essentially incorporated in the Hebbian-LMS algorithm.

XI. The Postulates and the Hebbian-LMS Algorithm

The Hebbian-LMS algorithm of equations (6), (7), and (8), and diagrams in Figures 9 and 10 as applied to both excitatory and inhibitory inputs performs in complete accord with the biological postulates of synaptic plasticity. The second postulate, for excitatory inputs, is "fire together wire together." The third postulate, for inhibitory inputs, is "fire together un-wire together." This is a clear extension of Hebb's rule.

An algorithm based on Hebb's original rule would cause all the weights to converge and saturate at their maximum values after many adaptive cycles. Weights would only increase, never decrease. A neural network with all equal weights would not be useful. Accordingly, Hebb's rule is extended to apply to both excitatory and inhibitory synapses and to the case where the presynaptic neuron fires and the postsynaptic neuron does not fire. Synaptic scaling to maintain stability also needs to be taken into account. The Hebbian-LMS algorithm does all this.

XII. Nature's Hebbian-LMS Algorithm

The Hebbian-LMS algorithm performs in accord with the synaptic postulates. These postulates indicate the direction of synaptic weight change, increase or decrease, but not the rate of change. On the other hand, the Hebbian-LMS algorithm of equation (6) not only specifies direction of weight change but also specifies rate of change. The question is, could nature be implementing something like Hebbian-LMS at the level of the individual neuron and its synapses and in a full blown neural network?

The Hebbian-LMS algorithm changes the individual weights at a rate proportional to the product of the input signal

POSTULATES OF PLASTICITY

1) Excitatory and Inhibitory - no weight change with presynaptic neuron not firing: no neurotransmitter in gap.

2) Excitatory - gradual increase in number of neuroreceptors with both pre and postsynaptic neurons firing: neurotransmitter in gap, membrane voltage above threshold. (Hebb's rule)

3) Inhibitory - gradual decrease in number of neuroreceptors with both pre and postsynaptic neurons firing: neurotransmitter in gap, membrane voltage above threshold. (Extension of Hebb's rule)

4) Excitatory - gradual decrease in number of neuroreceptors with presynaptic neuron firing and postsynaptic neuron not firing: neurotransmitter in gap, membrane voltage below threshold. (Extension of Hebb's rule)

5) Inhibitory - gradual increase in number of neuroreceptors with presynaptic neuron firing and postsynaptic neuron not firing: neurotransmitter in gap, membrane voltage below threshold. (Extension of Hebb's rule)

6) Postulates 2)–5) reverse if $|(\text{SUM})|$ exceeds the equilibrium point values. Synaptic scaling (natural homeostasis) keeps (SUM) within small ranges about the equilibrium points, thus stabilizing firing rate of post-synaptic neuron. (Beyond Hebb's rule)

FIGURE 18 Postulates of plasticity.

and the error signal. The Hebbian-LMS error signal is roughly proportional to the (SUM) signal for a range of values about zero. The error drops off and the rate of adaptation slows as (SUM) approaches either equilibrium point. The direction of adaptation reverses as (SUM) goes beyond the equilibrium point, creating homeostasis.

In the synaptic cleft, the amount of neurotransmitter present is proportional to the firing rate of the presynaptic neuron, i.e., the input signal to the synapse. By ohmic conduction, the synaptic membrane voltage is proportional to the voltage of the postsynaptic soma, the (SUM), which determines the error signal. The rate of change in the number of neurotransmitter receptors is approximately proportional to the product of the amount of neurotransmitter present and the voltage of the synaptic membrane, negative or positive. This is all in agreement with the Hebbian-LMS algorithm. It is instructive to compare the drawings of Figure 17 with those of Figures 9 and 15. In a functional sense, they are very similar. Figures 9 and 15 show weight changing being dependent on the error signal, a function of the (SUM), and the input signals to the individual weights. Figure 17 indicates that the (SUM) signal is available to the synaptic membrane by linear ohmic conduction and the input signal is available in the synaptic cleft as the concentration of neurotransmitter.

The above description of synaptic plasticity is highly simplified. The reality is much more complicated. The literature on the subject is very complicated. The above description is a simplified high-level picture of what occurs with adaptation and learning in neural networks.

The question remains, when nature performs learning in neural networks, is this done with an algorithm similar to Hebbian- LMS? No one knows for sure, but "*if it walks like a duck, quacks like a duck, and looks like a duck, maybe it is a duck*."

XIII. Conclusion

The Hebbian learning rule of 1949, "fire together wire together", has stood the test of time in the field of neurobiology. The LMS learning rule of 1959 has also stood in the test of time in the field of signal processing and telecommunications. This paper has introduced several forms of a Hebbian-LMS algorithm that implements Hebbian-learning by means of the LMS algorithm. Hebbian-LMS extends the Hebbian rule to cover inhibitory as well as excitatory neuronal inputs, making Hebbian learning more "biologically correct." At the same time, Hebbian-LMS is an unsupervised clustering algorithm that is very useful for automatic pattern classification.

Given the available parts of nature's neural networks, namely neurons, synapses, and their interconnections and wiring configurations, what kind of learning algorithms could be implemented? There may not be a unique answer to this question. But it may be possible that nature is performing Hebbian-LMS in at least some parts of the brain.

Acknowledgment

We would like to acknowledge the help that we have received from Neil Gallagher, Naren Krishna, and Adrian Alabi.

References

[1] D. O. Hebb, *The Organization of Behavior.* New York: Wiley, 1949.
[2] G.-Q. Bi and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *J. Neurosci.*, vol. 18, no. 24, pp. 10464–10472, 1998.
[3] G.-Q. Bi and M.-M. Poo, "Synaptic modifications by correlated activity: Hebb's postulate revisited," *Annu. Rev. Neurosci.*, vol. 24, pp. 139–166, Mar. 2001.
[4] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neurosci.*, vol. 3, no. 9, pp. 919–925, 2000.
[5] B. Widrow and M. E. Hoff Jr., "Adaptive switching circuits," in *Proc. IRE WESCON Convention Rec.*, 1960, pp. 96–104.
[6] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1985.
[7] B. Widrow, "Bootstrap learning in threshold logic systems," in *Proc. Int. Federation Automatic Control*, 1966, pp. 96–104.
[8] W. C. Miller, "A modified mean square error criterion for use in unsupervised learning," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1967.
[9] R. W. Lucky, "Automatic equalization for digital communication," *Bell Syst. Tech. J.*, vol. 44, no. 4, pp. 547–588, 1965.
[10] R. W. Lucky, "Techniques for adaptive equalization for digital communication," *Bell Syst. Tech. J.*, vol. 45, no. 2, pp. 255–286, 1966.
[11] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, "The No-Prop algorithm: A new learning algorithm for multilayer neural networks," *Neural Netw.*, vol. 37, pp. 182–188, Jan. 2012.
[12] B. Widrow and J. C. Aragon, "Cognitive memory," *Neural Netw.*, vol. 41, pp. 3–14, 2013.
[13] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-means clustering algorithm," *J. Roy. Stat. Soc. Ser. C (Appl. Stat.)*, vol. 28, no. 1, pp. 100–108, 1979.
[14] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical Statistics Probability*, 1967, vol. 1, no. 14, pp. 281–297.
[15] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc. Ser. B (Methodol.)*, vol. 39, no. 1, pp. 1–38, 1977.
[16] C. F. J. Wu, "On the convergence properties of the EM algorithm," *Ann. Stat.*, vol. 11, no. 1, pp. 95–103, 1983.
[17] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowledge Discovery Data Mining*, 1996, vol. 96, no. 34, pp. 226–231.
[18] M. Lichman. (2013). UCI machine learning repository. [Online]. Available: http://archive.ics.uci.edu/ml
[19] D. Purves, G. J. Augustine, D. Fitzpatrick, A.-S. LaMantia, J. O. McNamara, and S. M. Wiliams, *Neuroscience.* Sunderland, MA: Sinauer Associates, Inc., 2008.
[20] H. R. Wilson and J. D. Cowan, "Excitatory and inhibitory interactions in localized populations of model neurons," *Biophys. J.*, vol. 12, no. 1, pp. 1–24, 1972.
[21] C. van Vreeswijk and H. Sompolinsky, "Chaos in neural networks with balanced excitatory and inhibitory activity," *Science*, vol. 274, no. 5293, pp. 1724–1726, 1996.
[22] C. Luscher and R. C. Malenka, "NMDA receptor-dependent long-term potentiation and long-term depression (LTP/LTD)," *Cold Spring Harb Perspect Biol.*, vol. 4, no. 6, pp. 1–15, 2012.
[23] M. A. Lynch, "Long-term potentiation and memory," *Physiol. Rev.*, vol. 84, no. 1, pp. 87–136, 2004.
[24] J. F. Prather, R. K. Powers, and T. C. Cope, "Amplification and linear summation of synaptic effects on motoneuron fring rate," *J. Neurophysiol.*, vol. 85, no. 1, pp. 43–53, 2001.
[25] G. G. Turrigano, K. R. Leslie, N. S. Desai, N. C. Rutherford, and S. B. Nelson, "Activity-dependent scaling of quantal amplitude in neocortical neurons," *Nature*, vol. 391, no. 6670, pp. 892–896, 1998.
[26] G. G. Turrigano and S. B. Nelson, "Hebb and homeostasis in neuronal plasticity," *Curr. Opin. Neurobiol.*, vol. 10, no. 3, pp. 358–364, 2000.
[27] G. G. Turrigano, "The self-tuning neuron: Synaptic scaling of exitatory synapses," *Cell*, vol. 135, no. 3, pp. 422–435, 2008.
[28] N. Virtureira and Y. Goda, "The interplay between Hebbian and homeostasis synaptic plasticity," *J. Cell Biol.*, vol. 203, no. 2, pp. 175–186, 2013.
[29] D. Stellwagen and R. C. Malenka, "Synaptic scaling mediated by glial TNF-α," *Nature*, vol. 440, no. 7087, pp. 1054–1059, 2006.
[30] V. R. Kompella, M. Luciw, and J. Schmidhuber, "Incremental slow feature analysis: Adaptive low-complexity slow feature updating from high-dimensional input streams," *Neural Comput.*, vol. 24, no. 11, pp. 2994–3024, 2012.
[31] Z. K. Malik, A. Hussain, and J. Wu, "Novel biologically inspired approaches to extracting online information from temporal data," *Cogn. Comput.*, vol. 6, no. 3, pp. 595–607, 2014.
[32] Y. Choe, "Anti-Hebbian learning," in *Encyclopedia of Computational Neuroscience,* Berlin, Germany: Springer-Verlag, 2014, pp. 1–4.