TCM Decoding Using Neural Networks

Edit J. Kaminsky
University of New Orleans, ejbourge@uno.edu

Nikhil Deshpande

Recommended Citation

Kaminsky, Edit J., and Nikhil Deshpande, "TCM Decoding using Neural Networks," Engineering Applications of Artificial Intelligence, Vol 16, no. 5-6, Aug.-Sept., 2003, pp. 473-489.

ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

PERGAMON

Engineering Applications of Artificial Intelligence ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Engineering Applications of ARTIFICIAL INTELLIGENCE

www.elsevier.com/locate/engappai

TCM decoding using neural networks

Edit J. Kaminsky[a,*], Nikhil Deshpande[b]

[a] *Department of Electrical Engineering, University of New Orleans, EN 852-Lakefront Campus, New Orleans, LA 70148, USA*
[b] *Annotate.net, 3 South Main Street, Suite 5B, Allentown, NJ 08501, USA*

Received 27 March 2002; received in revised form 19 May 2003; accepted 9 June 2003

Abstract

This paper presents a neural decoder for trellis coded modulation (TCM) schemes. Decoding is performed with Radial Basis Function Networks and Multi-Layer Perceptrons. The neural decoder effectively implements an adaptive Viterbi algorithm for TCM which learns communication channel imperfections. The implementation and performance of the neural decoder for trellis encoded 16-QAM with amplitude imbalance are analyzed.
© 2003 Elsevier Ltd. All rights reserved.

PACS: 07.05.Mh; MSC 98B12; MSC 98B35

Keywords: Trellis coded modulation; Viterbi decoding; Neural networks; Radial basis function networks; Perceptrons; Communication channel imperfections

1. Introduction

Most applications of neural networks to communications systems have concentrated on nonlinear filtering and equalization (Sheng Chen and Mulgrew, 1993; Al-Mashaq and Reed, 1994) or on routing for ATM systems (Mehmet and Kamoun, 1993; Lee, 1993). More recently, work has been performed in the decoding area (Buckley and Wicker, 1999). Several other neural network applications to communications, including channel modeling, echo and interference cancellation, and receiver design are summarized in Haykin (2000). In Buckley and Wicker (2000), a neural system composed of two neural networks is used to predict turbo decoding error and request re-transmission if deemed necessary. We present here a novel application: neural decoding for trellis coded modulation (TCM) systems.

The neural decoder consists of radial basis function networks (RBFN) (NeuralWare Technical Publications Group, 1993) with adaptive centers to adjust to channel imperfections and noise conditions. The signal mapper, which ultimately produces the binary output, uses multi-layer perceptrons (MLP) (Werbos, 1974; Hagan et al., 1996; Fa-Long and Unbenhauen, 1997). It has already been proven (Petsche and Dickinson, 1987) that near maximum likelihood (ML) estimation can be performed using trellis-like neural networks. Moreover, Petsche and Dickinson (1990) recognized the similarities between the structure of trellis convolutional codes and neural networks, and applied concepts from coding to generate fault tolerant, self-repairing neural networks. We exploit these similarities—in the opposite direction—to design neural networks to decode TCM signals in communications systems with imperfections. Unlike the networks used in Petsche and Dickinson (1990), which are fully connected and update the weights based on Grossberg differential equations, we have networks whose connections depend on the trellis code being used, and use competition between layers of RBFNs.

Standard Viterbi algorithm (Viterbi, 1971) decoding of TCM is intolerant to channel or receiver variations in the reference signals. The Viterbi decoder's behavior does not depend on the channel itself or on how the noise-free signals in the subset may be modified by the channel or the receiver. We present the design of a system which is able to learn the channel/receiver characteristics and then bases decoding decisions on the additional information acquired.

*Corresponding author. Tel.: +1-504-280-5616; fax: +1-504-280-3950.
E-mail address: ejbourge@uno.edu (E.J. Kaminsky).

Also, because neural networks are parallel machines, the speed limitation of the standard Viterbi algorithm is alleviated by using a neural decoder. Decoding complexity of the neural decoder is still proportional to the number of encoder states, 2^{υ}, where υ is the encoder memory.

Standard decoding of TCM is performed in two phases (Wicker, 1995): path selection and subset decoding. The neural decoder, therefore, is composed of two interconnected subnets operating in parallel. The requirements for Viterbi-like decoding from one network (for path selection) and signal classification with signal-to-bit mapping by the other network (for subset decoding) lead to very different designs. The neural Viterbi decoder network tasked with performing path selection requires elements that classify received sequences with elements that have overlapping decision regions, and a feedforward architecture with constraints on the connections between layers. The signal classifier/mapper is to behave as a simple classifier which maps non-binary signal inputs to binary output signals. These requirements led us to choose RBFN for the Viterbi decoder (path selector) and multi-layer perceptrons (MLP) for the signal classifier/mapper. This means that the estimated signals from the RBFN are the inputs to a signal classification network that identifies the signals in the constellation and generates the decoded bits.

The overall goals of this paper are to: (1) present an adaptive TCM-decoding scheme suitable for parallel hardware implementation, and (2) provide insight into the overlapping concepts between the coding theory and neural networks disciplines. We stress the latter by discussing the appropriateness of utilizing RBFNs with Gaussian transfer functions in the path selection stage of TCM decoding, which effectively compute the likelihood functions, providing not only the best path at the output, but a 'confidence measure' as to the certainty that the selected path was actually transmitted.

We use trellis-encoded 16-QAM throughout to exemplify our work. Adaptation to channel imperfections is demonstrated with a simple system with amplitude imbalance.

The rest of this paper is organized as follows: Section 2 presents a short review of TCM, trellises, and the Viterbi decoding algorithm and metrics as applied to TCM; it also describes the trellis-encoded 16 QAM system using a rate-1/2 convolutional encoder. Section 3 summarizes RBFN concepts. In Section 4 the design and implementation of the neural TCM decoder is explained. Section 5 describes the simulation strategies. Simulation results for the 16-QAM TCM system with AWGN and amplitude imbalance are given in Section 6. Concluding remarks are presented in Section 7, followed by references.

2. Trellis coded modulation

Trellis coded modulation (TCM) is a combined coding and modulation scheme proposed by Ungerboeck (1982) to be used primarily over band-limited channels where bandwidth efficiency is a priority. TCM schemes use redundant modulation in combination with a finite-state convolutional encoder. TCM expands the signal space to provide redundancy for coding and then performs joint coding and modulation so as to maximize the minimum Euclidean distance between coded signal sequences. Ungerboeck proposed the following steps to improve the bit error rate over uncoded systems: (1) Add one redundant bit to every m source bits via a rate $m/(m+1)$ convolutional encoder; (2) expand the signal constellation from 2^m to 2^{m+1} signals; and (3) use the $m+1$ encoded bits to select signals from the expanded constellation. Thus, the modulator maps the data bits into one of $M = 2^{m+1}$ possible signals for transmission over the communications channel with a bandwidth efficiency of m information bits/s/Hz. TCM schemes, nonetheless, are neither limited to one redundant bit, nor to 1D and 2D constellations. Work by Wei (1987); Kaminsky et al. (2002), and others have sought to reduce the normalized redundancy by using multidimensional constellations.

Ungerboeck's signal assignment rules, restated in Wicker (1995), are: (1) signals in the same, lowest partition (subset) are assigned to parallel transitions in the trellis; (2) signals in the preceding partition are assigned to transitions that start or stop at the same state; and (3) all signals are used equally often. The encoded bits select a subset from the constellation partition, and the remaining bits select the point to be transmitted from within the selected subset, as shown in Fig. 1.

Each node in the convolutional encoder's trellis diagram represents the state of the encoder at a given time. A branch between two states represents the transition between these states and is labeled with the output code sequence for convolutional codes, or subsets for TCM, associated with that transition. Every

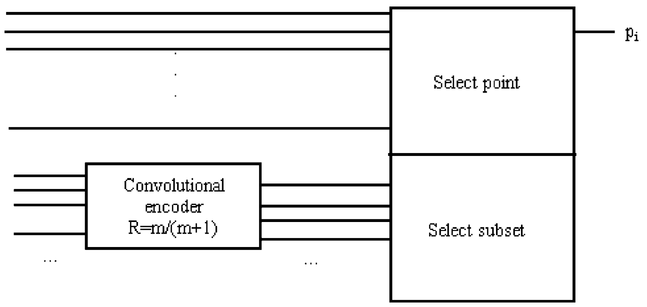

Fig. 1. Typical trellis encoder showing encoded bits selecting subset and uncoded bits selecting signal from within the selected subset. The modulated signal point selected, p_i, is transmitted.

possible sequence of transmitted subsets corresponds to a unique path through the trellis.

At the receiver, a sequence decoder must make a nearest-neighbor decision as to which sequence out of the entire coded signal space the received signal sequence is nearest to. The most probable error is between signals that are closest in terms of Euclidean distance (ED). Ungerboeck designed codes to maximize the free distance and at the same time preserve bandwidth efficiency by incorporating the redundancy into the signal set itself. The free distance is defined by

$$d_{\text{free}} = \text{Min}(d_{\text{par}}, d), \tag{1}$$

where d_{par} is the distance between parallel trellis branches, and d is the minimum distance between paths of length longer than 1 branch. For a detailed discussion of distance between paths, error-correcting capability of a code, and the generator function of a convolutional code or trellis refer to Clark and Cain (1981) or Wilson (1996).

2.1. Trellis coded 16-QAM

We present here the trellis encoded 16-QAM system that we use throughout this paper to exemplify the general neural TCM decoder design. This is also the system simulated for which we present our neural decoding results. The neural decoder can easily be extended to more complex systems with encoders of higher rate and larger constellations.

It is assumed we want to send 3 bits/s/Hz for which uncoded 8-PSK could be used if no error correction were desired. To use TCM with one redundant bit, the constellation size must be doubled to 16 points. The signal set of choice for the expanded constellation is, then, 16-QAM. The two 4-level streams are modulated using a carrier and a 90° phase-shifted version of the carrier to provide the in-phase (I) and quadrature-phase (Q) signals which are transmitted over the noisy channel.

White noise is assumed to be inherent in the channel. Adding noise to the signal is equivalent to adding independent random noise to each of the I and Q channels, which cause the constellation point to move relative to its true (noise-free) position.

Redundancy is introduced following Ungerboeck's rules by using a rate 1/2 convolutional encoder. One information bit enters the encoder, and the resulting two coded bits are used to select one of the four possible subsets. The remaining two uncoded bits are used to select one of the 4 possible signals within the selected subset. We can consider the overall rate of the TCM system to be of rate 3/4.

Fig. 2 shows the expanded constellation, the 16-QAM signal set, partitioned following the rules for set partitioning proposed in Ungerboeck (1987). The points in the 16-QAM constellation are signed permutations of the following basic 2D points: (1,1), (1,3), and (3,3). The minimum distance between points in 16-QAM is therefore 2. The constraints placed by the set-partitioning rules result in the points within each subset having a greater minimum distance than the parent constellation; this distance, called the free distance of the system, is 4, for an asymptotic gain of 3 dB.

Fig. 3 shows the complete TCM encoder, including the convolutional encoder of rate 1/2. Fig. 4 is the corresponding 4-state trellis diagram. One uncoded bit, x^1, enters the convolutional encoder, and two encoded bits are produced at its output, y^0 and y^1. These two bits select one of four possible subsets, denoted S_0 through

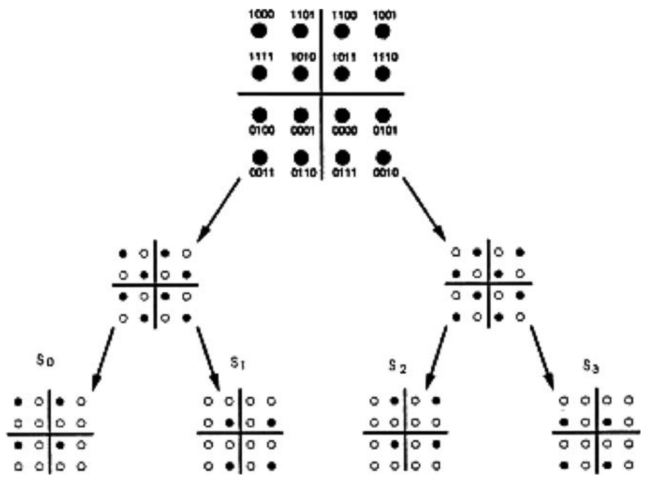

Fig. 2. Partition of 16-QAM into 4 subsets.

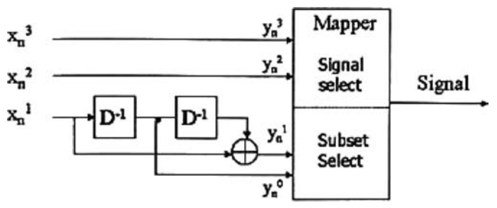

Fig. 3. TCM with convolutional encoder of rate 1/2 and constraint length of 2.

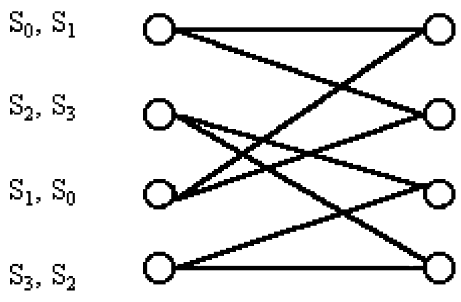

Fig. 4. Four state trellis diagram.

S_3 in Figs. 2 and 4. Each of these subsets contains 4 points, and it is the other two original bits, x^2 and x^3 that select, from within the selected subset, the point to be transmitted. Fig. 4 also shows the subset assignment to each branch of the trellis. For example, the transition from the second state to the fourth state requires transmission of a signal from subset S_3. If the remaining uncoded bits are 01, then the point transmitted is $(1, -3)$.

2.2. Viterbi algorithm

The Viterbi algorithm (VA) was proposed in Viterbi (1971) for decoding convolutional codes in memoryless noise. Small modifications to the VA need to be made for decoding TCM sequences. Costs are associated with each branch in the trellis, and a path is sought through the trellis for which the total cost is minimized.

The task of the TCM decoder is to estimate the path that most closely resembles that of the received signals. This maximum-likelihood (ML) problem is akin to finding the shortest distance through the TCM trellis. Wicker proves in Wicker (1995) that the path selected by the Viterbi decoder is the maximum likelihood path.

Consider an information sequence $\mathbf{x}$ encoded into a codeword $\mathbf{y}$ which is then sent through a communications channel with additive white Gaussian noise (AWGN). The decoder must take the received vector $\mathbf{r}$ and generate an estimate $\hat{\mathbf{y}}$ of the transmitted codeword.

The inputs $\mathbf{x}$, each of length m, are the input to a rate m/k convolutional encoder; as stated previously, we will use $k = m + 1$, so only one redundant bit is used per symbol. The output of the encoder is the sequence $\mathbf{y}$. Memoryless channel noise corrupts this output vector producing the received signal $\mathbf{r}$:

$$\mathbf{r} = \mathbf{y} + \mathbf{n}, \tag{2}$$

where $\mathbf{n}$ is the zero-mean Gaussian random variable representing the AWGN. The decoder generates an ML estimate $\hat{\mathbf{y}}$ as in (3):

$$\hat{\mathbf{y}} \quad \text{for max } p(\mathbf{r}|\hat{\mathbf{y}}). \tag{3}$$

Wicker (1995) summarizes the operation of the VA as follows: Let the node corresponding to state j at time t be $N_{j,t}$. Metrics $V(N_{j,t})$ are assigned to each node in the trellis using the following steps:

- set $V(N_{0,0}) = 0$ and $t = 1$.
- at time t, compute the path metric for all paths entering each node.
- set $V(N_{k,t})$ equal to the best partial path metric entering the node corresponding to state k at time t. In case of a tie, randomly assign $V(N_{k,t})$. This is called the survivor branch.
- all non-survivor branches are deleted from the trellis.
- continue until end of transmission (t_{final}).
- start at the last node at t_{final} and follow the surviving branches back through the trellis. The path generated in this manner corresponds to the ML path.

The metric most often used for soft decoding of TCM is the squared Euclidean distance (SED), but any monotonic function of the SED may be used. The performance criterion used in the analysis of a Viterbi decoder is minimization of the probability of error event, where an error event occurs when a path leaves the true path in the trellis, reemerges with the true path at a later time, and is declared a survivor.

Since the VA finds the most likely path through the encoder's trellis, it minimizes the sequence or event error probability. The bit error probability is closely related to the event error probability, and hence the VA also results in a small, but not necessarily minimum, value of bit error probability (Lin and Costello, 1983).

2.3. TCM decoding

TCM decoding is performed in two steps:

- *Signal selection at each branch* (*subset decoding*): At each branch in the trellis, the decoder compares the received signal to all the signals in the subset assigned to that branch. The identity of the signal closest to the received signal in Euclidean distance is saved. The branch is labelled with this metric.
- *Path selection*: The VA, as described above, is then applied to the trellis, with surviving partial paths corresponding to partial signal sequences that are closest to the received sequence. Thus, the VA is used to find the signal through the code trellis with the minimum overall metric (most often the sum of squared distances) from the sequence of received signals chosen by subset decoding.

2.4. Metrics

As mentioned above, there is no unique metric universally used in the VA. If the only requirement is to select the most likely sequence to have been transmitted, conditioned on the received sequence, any metric may be used. The Hamming distance is the most widely used metric in hard Viterbi decoding, while the Euclidean distance is most commonly used for soft decoding. If we have the added requirement of obtaining some information about the reliability of the decision, a function of the distance and the channel characteristics should be used. For example, we may use the probability that the particular branch is the most likely branch as the metric; we may also use the log of this function, or any other monotonic function of it. Again, it is only the 'additional information' obtained that changes, not the path selected. The standard ML decoding does not deliver a reliability measure along with the selected sequence.

The path metric for path i, $V^{(i)}$, is given by the sum of the B branch metrics, μ_j, in that path (Proakis, 2000):

$$V^{(i)} = \sum_{j=1}^{B} \mu_j^{(i)}. \tag{4}$$

The path with the maximum (or minimum, depending on how the metrics are defined) metric is chosen as the ML path.

It is shown in Bossert (1999) that the MAP probability is proportional to an exponential function of the path metric:

$$P(x)P(y|x) \sim \mathrm{e}^{V}, \tag{5}$$

where this is true for the overall metric as well as the partial metrics.

We can easily understand the decoding strategy if we look at the problem as hypothesis testing: There are M possible hypotheses (sequences), S; the decision maker (the decoder) produces a decision based on the received sequence $\mathbf{r}$. The decision is made in favor of the sequence $\mathbf{y_i}$ if $\mathbf{r} \in D_i$, the decision region of $\mathbf{y_i}$ (Wilson, 1996). An error (false hypothesis selected) occurs if the received vector is within the wrong decision region. The conditional probability of error is then

$$P(e|\mathbf{S}_i) = P(\mathbf{r} \in D_i^c|\mathbf{S}_i) = \int_{D_i^c} f(\mathbf{r}|\mathbf{S}_i)\,\mathrm{d}\mathbf{r}, \tag{6}$$

where the superscript c indicates complement.

The overall probability of error is

$$P(e) = \sum_{i=0}^{M-1} P_i P(e|\mathbf{S}_i) = \sum_{i=0}^{M-1} P_i \int_{D_i^c} f(\mathbf{r}|\mathbf{S}_i)\,\mathrm{d}\mathbf{r}. \tag{7}$$

The decision then becomes

$$\hat{\mathbf{S}}_i = \arg\max_{S_i}[P_i f(\mathbf{r}|\mathbf{S}_i)]. \tag{8}$$

Using Bayes's rule we obtain

$$P(\mathbf{S}_i|\mathbf{r}) = \frac{P_i f(\mathbf{r}|\mathbf{S}_i)}{f(\mathbf{r})}, \tag{9}$$

where $f(\mathbf{r})$ is the marginal pdf, and is independent of i. So maximization of the prior, maximizes the posterior.

We use the fact that maximization over i of *any* monotonic function of $P_i f(\mathbf{r}|\mathbf{S}_i)$ is optimal.

Often, it is the product of metrics that is minimized (or maximized). We will shortly use the fact—recently discussed in detail in Kschischang et al. (2001)—that we can also perform the "minimum of sums" operation instead of the "sum of products" operation to achieve the same ML path.

2.5. Performance measures

The upper bound on the node error probability is important and therefore presented here. If $\mathbf{y}$ is the transmitted coded sequence and $\mathbf{r}_k$ is the sequence that diverges from $\mathbf{y}$ at some point in the trellis an reemerges after k branches, and $P(\mathbf{y} \to \mathbf{r}_k)$ is the pairwise error probability of the two sequences, then the union bound for the probability of error is given by

$$P_e \leqslant \sum_{k=1}^{\infty} \sum_{\mathbf{y}} \sum_{\mathbf{r_k}} P(\mathbf{y})P(\mathbf{y} \to \mathbf{r}_k), \tag{10}$$

where the innermost summation is for the probability of error for all the components of the diverging sequence of length k, the next summation is for all possible transmitted sequences, and the outermost summation is for all possible transmitted sequence errors which diverge for 1 branch, 2 branches, and so on. A simplified expression for the union bound on P_e is

$$P_e \leqslant Q\left(\sqrt{\frac{d_{\text{free}}^2 E_S}{2N_o}}\right) \times \exp\left(\frac{d_{\text{free}}^2 E_S}{4N_o}\right) T(D)|_{D=\exp(-E_S/4N_o)}, \tag{11}$$

where $T(D)$ is the weight enumerating function (also called the generator function) of the convolutional code, as defined in Wicker (1995), Lin and Costello (1983), and Wilson (1996). E_S is the average symbol energy, and the variance of each component of the AWGN is $\sigma^2 = N_o/2$. The free distance was given in (1).

If the signal to noise ratio is sufficiently high, a more tractable and widely used approximation for the probability of error is

$$P_e \approx N(d_{\text{free}})Q\left(\sqrt{\frac{d_{\text{free}}^2 E_S}{2N_o}}\right), \tag{12}$$

where $N(d_{\text{free}})$ is the error coefficient, i.e. the average number of sequences that are at free distance d_{free} from the transmitted sequence.

3. Radial basis function networks

Radial basis function networks (RBFN) have radially symmetric internal representations of the hidden processing elements or pattern units (PU). Each PU has a center, a distance measure, and a transfer function. The output of an RBFN is a multidimensional function of the distance between the input vector and the center or prototype vector. The input-layer neurons of the RBFN feed the input vector to the hidden layer. The hidden units compute the distance between the input vector and the pattern unit's center.

We denote the input vector as

$$\mathbf{X} = [x_1, x_2, x_3, \ldots, x_n]^{\mathrm{T}} \tag{13}$$

and the center of each hidden layer neuron i as

$$\mathbf{C}_i = [c_{1i}, c_{2i}, c_{3i}, \ldots, c_{ni}]^{\mathrm{T}} \tag{14}$$

where n is the dimension of the input vector. The output of each neuron i in the hidden layer is given by

$$h_i = f_i(\|\mathbf{X} - \mathbf{C}_i\|). \tag{15}$$

The connection between the hidden and output layers is weighted and each neuron in the output layer has a linear input/output relationship thereby implementing summation only. A diagram of an RBFN is shown in Fig. 5. Notice that the prototype vectors (centers) are stored as the connection weights between input and hidden layers.

The weights of the RBFN, i.e., the centers, C_{jk}, and the output weights, W_{jk}, must be computed. A clustering algorithm is used for the centers, while one of the many available training algorithms (Fa-Long and Unbenhauen, 1997) are used to select the weights in the output layer. If the fixed channel structure is known precisely, the clustering part of the training algorithm for the RBFN can be eliminated in favor of predetermined node centers.

The units in an RBFN are 'selective' for some range of input signal space. We will use this 'selective region' to be the 'decision region', D_i, in Eq. (6). Not only do we determine whether the received signal is within the region, but how close it is to its center.

Many functions are appropriate for the mapping of (15). Any strictly positive radially symmetric kernel with a unique maximum at its center, and which drops off rapidly to zero away from the center, is appropriate (Hassoun, 1995). Due to the nature of noise assumed (AWGN), we choose to use the Gaussian function for all neurons

$$h_i = \exp\left(\frac{-\|\mathbf{X} - \mathbf{C}_i\|^2}{\sigma^2}\right), \tag{16}$$

where σ is another RBFN parameter to be determined. This third parameter, σ, is the standard deviation of the neuron's receptive field, indicative of the width of the receptive field of the neuron, and usually obtained by the nearest neighbor heuristic (NeuralWare Technical Publications Group, 1993). In our application, this parameter is the standard deviation of the AWGN in the communications channel. The nearest neighbor heuristic indeed yields this result. The norm in (16) is the Euclidean norm. Under this view, one may interpret (16) as the probability of observing a certain sequence under a Gaussian distribution (were constants are neglected or normalized).

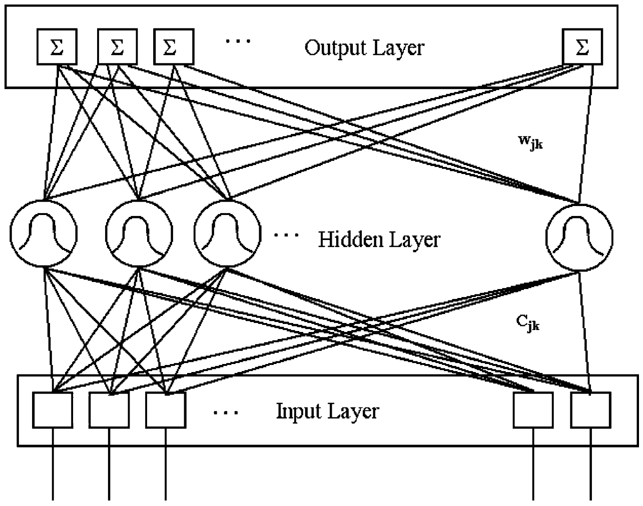

Fig. 5. Radial basis function network. The components of the center vectors (prototypes) are c_{jk} and the output layers' weights are w_{jk}.

4. Neural TCM decoder

In order to make the rest of this presentation clear, we define several terms and indicate the equivalences between the NN terms and the corresponding TCM terms. Please refer to the diagrams of Figs. 4 and 6, where the similarities between the TCM trellis and the RBFN architecture of our neural decoder become clear and intuitive.

- Layer: An RBFN neural network layer. There is one layer per each decoding signalling interval or, equivalently, one layer per each symbol in the sequence. We denote the number of layers as Γ.
- Dnodes: A decoder node. There is one Dnode per trellis node, where a trellis node is a starting or ending state in the encoder's trellis.
- Subnode: A subnode of a Dnode. There is one subnode per partition subset, and therefore as many subnodes per Dnode as there are branches leaving or entering a trellis node.
- PU: A subnode processing unit. Each PU corresponds to a signal in the partitioned subset. There are as many PU's in each subnode as there are signals in each partition subset.

The neural TCM decoder consists of two interconnected neural networks operating in parallel. We describe each in what follows.

4.1. RBFN decoder

A radial basis function network (RBFN) can be thought of as a multidimensional function that depends on the distance between the input vector and the center vector (Fa-Long and Unbenhauen, 1997). This 'distance between vectors' is analogous to the metric of the VA as

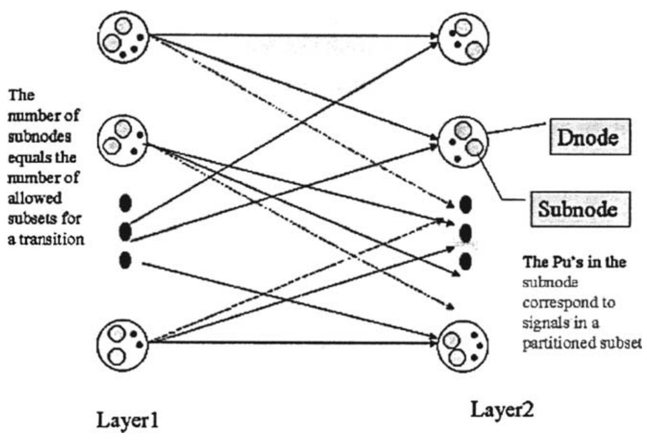
The number of subnodes equals the number of allowed subsets for a transition
Dnode
Subnode
The Pu's in the subnode correspond to signals in a partitioned subset
Layer1
Layer2

Fig. 6. Dnode and subnode architectures of the TCM decoder.

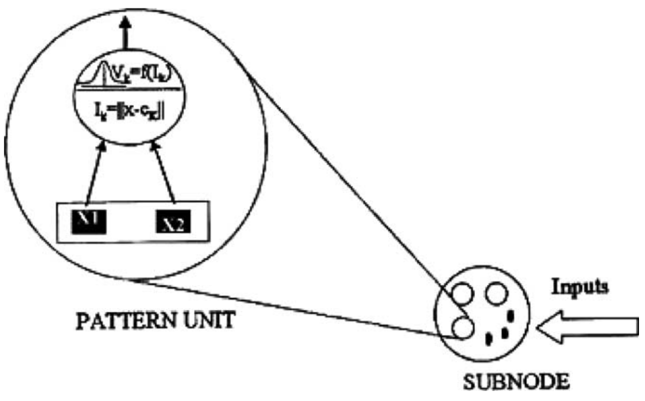

Fig. 7. TCM RBFN subnode and node architectures.

applied to TCM, where the distance between the transmitted vector and the received vector is of consequence. The functionality of a PU in an RBFN, then, is very similar to decoding in a TCM system since both compute distances between a vector (input pattern vector in RBFN, received vector in TCM) and a prototype (the center in RBFN, the possible transmitted signals in TCM).

For an n-dimensional input vector, each pattern unit first calculates the distance between the input vector and its center or prototype vector. In the ideal case, these centers are known to be equal to the expanded constellation used in TCM, or the mapped constellation points if the channel characteristics are known. In the case of unknown or varying channels, the fixed constellation signals are used as the initial state for the centers' training. This distance measure is then mapped to an output h_i (the metric in the neural TCM decoder) by a function which depends on the ED between the input vector and the center of the PU, as given in (16).

At this point it is pertinent to address the following question: "Why a Gaussian transfer function for the PU?". We know each branch in Viterbi decoding is labelled with a metric directly proportional to the distance between the received signal and the winning signal of the subset associated with that branch. Consider the case of a slightly noisy signal at the input to a PU. If the noise is a small, the signal will be slightly displaced from its position in the constellation (i.e., it will be slightly away from the center of the correct PU). We want the PU to classify this as being close to its stored center with a high degree of confidence; as the effect of noise becomes more pronounced, the confidence of the PU that the received signal is actually related to the signal stored in its center should decrease. The exponent in the Gaussian function ensures high values for small distances, and a decrease for values that are spaced relatively far away from zero. The standard deviation term, σ, in the Gaussian function (16) controls the variance of the distribution, and therefore the width of the transfer function. Functions that overlap among neighboring PUs are advantageous as in a low SNR environment the received signal at the appropriate PU may be spaced far enough from the center that the output of the PU may be lower than the output of a nearest neighbor PU in that node, i.e., a PU whose center is closer in ED to the PU under consideration. This signal would cause an error event if h_i is very narrow, as an erroneous branch would be followed in the state trellis. The value of σ, then, should be chosen based on the signal constellation being used and the expected noise level; if unknown or varying, an adaptive $\sigma(t)$ may be used.

We can think of the Gaussian transfer functions as yielding a "membership value" that describes to what degree the input pattern fits within the cluster's region.

Each of these RBFN PUs, then, computes the distance between the received vector and its stored prototype vector, and outputs the metric h_i. Several, say L, such PUs form a subnode in our network, as shown in Fig. 7. If there are a total of D states in the trellis, T transitions per state (incoming or outgoing branches at each trellis node), and $L = 2^l$ points per subset, we would have a total of D Dnodes in a layer, T subnodes per Dnode (for a total of DT subnodes per layer), and L PUs per subnode (for a total of LDT PUs per layer). A subnode, then, consists of as many PUs as there are signals in each subset.

Each node in the trellis is associated with the subsets that are assigned to the branches leaving the node, as shown in Fig. 4. This would mean that each Dnode in our net would have as constituents the subnodes that represent the subsets associated with the node under consideration in the trellis. A subnode therefore performs the first part of the first step in the VA by computing the distances from the centers of each of its PUs to the received signal.

The second part of the first step in decoding is keeping only the signal with the best metric. The input (received signal) is fed to all the PUs in the Dnodes. The output of the Dnodes is from the PU (signal) in a particular subnode (subset) whose output is a maximum (best metric). This operation is performed by a modified competitive transfer function $\mathbb{C}$ for the subnode. This transfer function, for each subnode t, is given by

$$\mathbb{C}_t = \max(PU_t). \qquad (17)$$

After the competitive function is applied, only the signal closest to the received signal in each subnode is kept, and its metric stored. At this point there will be a total of TD of these 'winners', T per Dnode in each layer.

The first layer of the neural decoder consists of Dnodes where each Dnode corresponds to a state of the convolutional encoder. The Dnodes in this first layer are connected to Dnodes in subsequent layers such that the subsets associated with the Dnode represent valid trellis transitions from the state being represented. Our neural decoder is only partially connected, and the total number of connections is bounded by a linear function of N, where N is the number of neurons in the network.

The next layer, and every succeeding layer of the decoder, will be identical to the first layer. Connections are formed between the Dnodes of the consecutive layers only when there exists a valid transition between the corresponding states of the trellis. Each Dnode also contains simple neurons that perform the addition operation for two inputs; this implements the cumulative metric computation as the VA proceeds. One input to this subsystem is the connection from the preceding layer and the other input is the received symbol delayed by k, where k is the layer position. Fig. 8 depicts two layers of the neural decoder architecture.

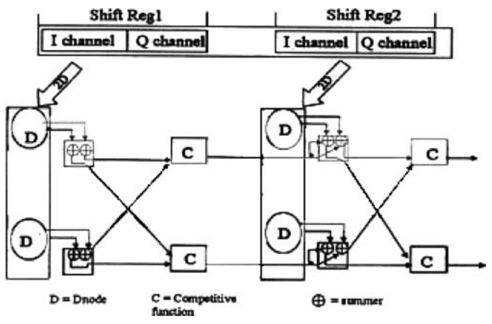

Fig. 8. Neural decoder architecture showing two layers of two Dnodes each.

The delayed inputs are sent to the neural net via a shift register or delay line as shown in Fig. 9. Signals are not fed to the neural decoder until the shift register is entirely filled with data during initialization.

To compare paths at each node where two or more branches are incident, a neuron with the modified competitive transfer function in (17) is incorporated as shown in Fig. 8 by the block labelled C. This neuron picks the Dnode with the largest path metric as the winner, and enables it to transmit its cumulative path metric to the succeeding Dnode additive element. The competitive layer output is also used as a gate input to the previous layer along the path as shown in Fig. 9.

The next step in giving the network the functionality of a Viterbi decoding algorithm is incorporating the next parameter of consequence, the decoding depth Γ, i.e., determining the number of layers in the network. The ML algorithm assumes that the entire transmitted

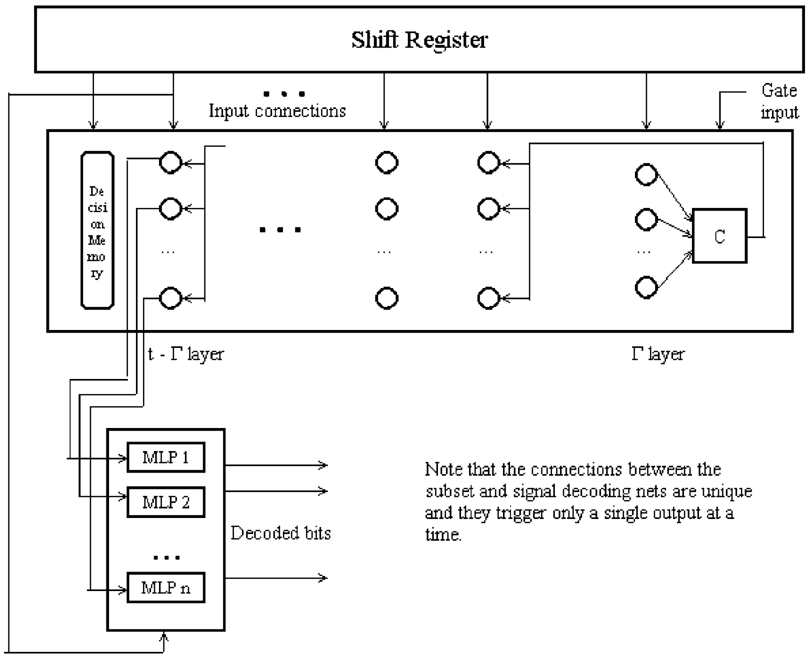
Shift Register
Input connections
Gate input
Decision Memory
C
t - Γ layer
Γ layer
MLP 1
MLP 2
MLP n
Decoded bits
Note that the connections between the subset and signal decoding nets are unique and they trigger only a single output at a time.

Fig. 9. Neural TCM decoder architecture.

sequence is received before the path is decoded via the Viterbi algorithm. Practically, of course, this is not feasible because the decoder cannot be allowed to have an infinite delay before releasing the decoded bits. We have chosen to output the $(t-\Gamma)$th information block on the surviving path with the best metric. The value of the decoding depth Γ is given in Forney (1974) as $\Gamma \approx 5.8\upsilon$, where υ is the length of the largest shift in the convolutional encoder. The probability of truncation error decreases exponentially with Γ (Forney, 1974). At low SNR, this probability of truncation error becomes negligible if $\Gamma \geqslant 5.8\upsilon$. The number of layers of Dnodes, then, is equal to the decoding depth Γ. At the Γth layer, a decoding decision is made on the $(t-\Gamma)$th received symbol. The decoder is designed such that the information bits associated with a surviving branch at time t are released when the decoder begins operation on the branches at time $t+\Gamma$.

A gate input is available to activate the decoded subnode (subset) within a Dnode. This subnode activates its unique signal decoding MLP network—discussed in the next section—that has already performed pattern recognition on the received signal to decode the remaining bits. The other MLP nets are not triggered and their signal decoding results are ignored or discarded. An element of parallelism is seen here, where the MLP signal decoding structure performs its task before it is activated based on the subset decoding net. Apart from enabling the associated signal decoding net, each Dnode has fixed bits stored in a memory element attached to the Dnode which correspond to the m decoded bits related to the k (usually $m+1$) encoded bits that select the subset. The Dnode, then, also serves to strip off the redundancy in the signal.

The early simulations of our neural system exposed a tendency of the network to have its branch metrics strongly influenced by noise in any of the final stages of the net, thereby often making decisions at the Γth layer incorrectly. We found that under noisy conditions the Dnode outputs tended to collapse together, and the separation between the minimum and maximum Dnode output was reduced compared to clean signal outputs where the separation was greater. To alleviate this problem, a weighting metric for each layer was developed as given by (18)

$$W_i = \max(Dnode_i) - \min(Dnode_i), \tag{18}$$

where W_i is the weighting metric of layer i, and $Dnode_i$ is the output of a Dnode in that layer. All the outputs of the Dnodes of the layer are then weighted by (18) and fed as inputs to the last layer so that the final sum at the competitive layer is a weighted sum of the winning Dnodes' outputs. It is possible that the branch metrics may become so small relative to the path metrics of the surviving paths that the branch metrics make no contribution during the update of the path metric. This may cause the decoder to fail catastrophically. This problem is remedied by using (18).

One of the disadvantages of RBFN, as stated by Fa-Long and Unbenhauen (1997), is that the initial learning phase is an unsupervised clustering algorithm. This phase is made to work to our advantage for the training of the PUs. Throughout the design we have made the assumption that the decoder has total knowledge of the TCM system at the transmitter end. This enables us to fix the centers of the PUs before we begin training. This assignment is the initialization step for an adaptive K-means algorithm that is used in the training phase of the RBFNs. Any number of sources or events may disturb this signal. Sinusoidal interference, amplitude imbalance, amplitude non-linearities, and phase jitter are shown in Winch (1993) to be potential problems on the received signal. If the errors in the training sequence are not too severe, then the PUs actually 'learn' these imperfections by shifting the position of their centers and, if needed, the σ in (16). The center of the winning PU is modified by

$$\mathbf{C}_i^{\text{new}} = \mathbf{C}_i^{\text{old}} + \alpha(\mathbf{X} - \mathbf{C}_i^{\text{old}}), \tag{19}$$

where $\mathbf{X}$ is, as before, the input symbol vector, $\mathbf{C}_i$ is the center for the ith PU, and α is the learning rate that either decreases as time increases or is fixed at a value smaller or equal to 0.5. In the former case training can be stopped either when $\alpha = 0$, or after a fixed number of iterations, or after it is observed that the PU has adapted to the channel/receiver conditions. If required, a very small α value may be fixed and training may continue throughout the operation of the decoder, enabling the decoder to adapt to long term drifts in the environment.

4.2. MLP signal mapper

In the previous section we mentioned the issue of subset decoding, i.e., decoding the signal from within the selected subset, and obtaining the output bits. These operations are performed by networks in parallel with the RBFNs. We describe this mapping/signal classifier here. The results of those networks not receiving a trigger signal are neglected. The function of the signal decoding net involves recognition of the signal in the subset and the mapping of the signal into bits to be the output of the decoder. We can use simple pattern recognition networks because the choice of signals in the subset decoded are due to the uncoded transmitted bits of TCM which are independent and memoryless events.

Our technique is based on the work presented by Bose and Garga (1993)—which somewhat adds complexity to the solution but keeps our complete system " neural"—in which an exact procedure is proposed that not only determines the number of layers and the number of neurons in each layer, but also the connection weights for the desired multilayer, feedforward topology. This technique is largely dependent on the construction of Voronoi diagrams (Okabe and Sugihara, 1992) over the set of points to be classified. Voronoi, and earlier Descartes and Dirichlet, considered the set of points regularly spaced in m dimensional space generated by linear combinations of m linearly independent vectors with integer coefficients. The Voronoi diagram generated by this set of points serves the purpose of partitioning the space into mutually congruent polyhedra.

The problem at this stage is to correctly classify the signal within the subset selected by the RBFN implementation of the VA, and to obtain the corresponding decoded bits. This is the subset decoding part of TCM decoding, and we decode at each stage of the partition (refer to Fig. 2). Noise forces the signal to be transformed from a point in signal space to a cloud distributed around the constellation point. Voronoi diagrams can be used to generate the decision boundaries around the constellation clouds.

It is well known (Minsky and Papert, 1990) that a single perceptron can solve linearly separable problems where the two pattern classes can be separated by a hyperplane. A layer of n neurons can represent the equation of n distinct hyperplanes. Any Voronoi region of interest can be expressed as the intersection of a finite number of closed half-spaces which are in turn defined by hyperplanes. It would take a single perceptron in a second layer to implement the intersection of the n hyperplanes in the first layer. Extending this result to several such regions of interest, we need a two-layered net having several neurons in the upper layer, where the number of neurons in the second layer is the number of Voronoi polygons. If several such polygons make up a particular class (partitioned set), then the union of all the associated Voronoi polygons for the class may be implemented by a third layer which combines the outputs from the relevant neurons in the second layer. The number of neurons in the third layer is equal to the number of patterns required to be classified.

The design algorithm from Bose and Garga (1993) is only summarized in what follows.

4.2.1. Algorithm for MLP design

- For all the classes of patterns described by the specified points in the feature space, construct the convex hulls C_i, using any procedure described in Okabe and Sugihara (1992).
- If there are only two classes and the interiors of the two convex hulls do not intersect, then a hyperplane may be situated between the two hulls as a decision boundary. The equation of the hyperplane in n-dimensional space can be written (Bose and Garga, 1993) as
$$H = \sum_{i=1}^{n} \lambda_i x_i + \delta, \tag{20}$$
where x_i are the coordinates of a point in n-dimensional space, and λ_i and δ are constants. Relating this to the equation of a multilayered perceptron we let λ be the connection weights, δ be the neuron bias, and x_i be the neuron inputs.
- If there are more than two classes, then for each class form a convex hull $\hat{C}_i$ over the specified point *not* belonging to the same ith class by merging the remaining convex hulls.
If the interiors of each pair of convex hulls C_i and $\hat{C}_i$ do not intersect, then the classes are linearly separable and a hyperplane exists that separates the points in that class from other points. The number of neurons in the layer is equal to the number of classes.
If the interiors of some convex hulls C_i and $\hat{C}_i$ intersect, then the classes are not linearly separable. A cluster of Voronoi cells is created for each class. If a cluster is convex, then it can be represented by the intersection of a finite number of hyperplanes as in (20). Let the half spaces described be denoted as H^+ and H^- and be given by (21) and (22), respectively.
$$H^+ = \sum_{i=1}^{n} \lambda_i x_i + \delta > 0 \tag{21}$$
and
$$H^- = \sum_{i=1}^{n} \lambda_i x_i + \delta < 0. \tag{22}$$
Each neuron in the first layer is then used to transform the input x_i to a boolean variable where 1 implies the input pattern lies in H^+ and 0 implies that the input pattern lies in the H^- space. Note that at each layer a hard limit transfer function is used for the transformation. A single neuron in a second layer can be used to combine the output of the first layer boolean transformation using the AND boolean operator. The number of neurons in the first layer is equal to the number of hyperplanes necessary to implement all the cluster edges.
- If the cluster is not convex it can be represented as a union of a finite number of convex regions. These convex regions for each class are generated by the second layer and then combined using an OR boolean operator in a third layer. The number of neurons in this third layer is equal to the number of classes.
- The AND and OR prototypes are easily derived (see, for example Hassoun, 1995 or Hagan et al.,

1996). The connection weight assignment between the neurons of the first and second layers is as follows:

$$W_i = \begin{cases} 1 & x_i \in H^+, \\ -1 & x_i \in H^-, \\ 0 & x_i \notin H^+ \vee H^-. \end{cases} \tag{23}$$

4.2.2. 16-QAM MLP mapper

We present the design of the MLP network for the 16-QAM system. The subsets to be classified are shown in Fig. 2, as S_0 through S_3. Let us examine any one subset, say S_0, as the results for this subset can be easily extended to all other subsets. The points at $(-3,3)$ and $(1,-1)$ form one subset due to a zero bit ($Y^2 = 0$) and (1,3) and $(-3,-1)$ form the other subset due to a unity bit ($Y^2 = 1$). Only two hyperplanes are necessary to separate these four 'regions'.

Following the algorithm described in the previous subsection, the convex hull for the two classes were created. As the two convex hulls C_1 and C_2 intersect at a point, we move on down the algorithm to the construction of the Voronoi diagram over all the specified points. The equations of these hyperplanes are given by (24) and (25), for the in-phase and quadrature-phase hyperplanes, respectively.

$$H_I = I + 1, \tag{24}$$

$$H_Q = Q - 1. \tag{25}$$

Using the half-space equations given in (21) and (22), we can write the hyperplane equations as

$$H_I = H_I^+ + H_I^-, \tag{26}$$

$$H_Q = H_Q^+ + H_Q^-. \tag{27}$$

Let us denote the region of the subset S_0 caused by an input bit zero as S_{00}, and S_{01} as the subset caused by an input bit equal to one. The binary relation between the classes and the regions generated by the hyperplanes are given by

$$S_{00} = (H_I^- \cap H_Q^+) \cup (H_I^+ \cap H_Q^-), \tag{28}$$

$$S_{01} = (H_I^- \cap H_Q^-) \cup (H_I^+ \cap H_Q^+). \tag{29}$$

Since there are two hyperplanes required, the first layer will contain just two neurons with bias $\delta_I = 1$ for the I-channel inputs, and $\delta_Q = -1$ for the Q-channel. The intersection of the two half-spaces is performed by an AND neuron in the second layer. Since there is one AND neuron for every intersection, the number of neurons in the second layer is 4. These four intersections have to be combined to obtain the output for the required two subsets ('classes'). Thus, two OR neurons are required in the final layer. Using (23), the connection weights w_{ij} for the ith neuron in the second layer via the jth neuron in the first layer are given by the elements (i,j) of

$$W_2^{\mathrm{T}} = \begin{bmatrix} -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{30}$$

The biases in the second layer are

$$B_2^{\mathrm{T}} = [-0.5 \quad -0.5 \quad 0.5 \quad -1.5]. \tag{31}$$

The weights and biases of the third layer are

$$W_3^{\mathrm{T}} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \tag{32}$$

$$B_3^{\mathrm{T}} = [-0.5 \quad -0.5]. \tag{33}$$

The resulting network is shown in Fig. 10. The design of the networks for the other 3 subsets of the 16-QAM partition is done in the same way, with the following hyperplane equations:

$$\begin{aligned} S_1: &\quad H_I = I + 1; \quad H_Q = Q + 1, \\ S_2: &\quad H_I = I - 1; \quad H_Q = Q + 1, \\ S_3: &\quad H_I = I - 1; \quad H_Q = Q - 1. \end{aligned} \tag{34}$$

At this point there is still one bit left to decode in the four state trellis, Y_n^3. This decoding can be established by tapping the described net at points as shown in Fig. 10. The competitive layer selects the largest perceptron output, and based on the signal-bit mapping at the transmitter, decides if the bit is to be inverted or not.

5. Simulations

Monte Carlo (Jeruchim, 1984) computer simulations using Importance Sampling (Shanmugam and Balaban, 1980) were performed to estimate the performance of the neural TCM decoder for 16-QAM. We characterize the performance by the probability of error P_e.

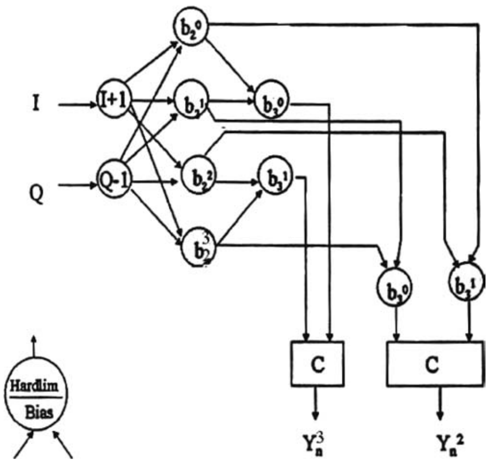

Fig. 10. Multilayer perceptron signal decoder architecture.

Estimates are given by the sample means over N samples

$$\hat{p}_0 = \frac{1}{N}\sum_{i=1}^{N} \delta_0(v_i), \tag{35}$$

where $\delta_0(v)$ is defined as

$$\delta_0(v) = \begin{cases} 1, & v > V_T, \\ 0, & v < V_T. \end{cases} \tag{36}$$

The estimate of the probability of error, $\hat{P}_e$, is given by

$$\hat{P}_e = \frac{n_e}{N_s}, \tag{37}$$

where n_e is the number of errors counted and N_s is the number of symbols transmitted. This estimate converges to its true value as $N_s \to \infty$.

For a finite number of transmitted symbols, the reliability of the estimator is quantified in terms of the confidence interval. The expression

$$P[h_1 \leqslant P_e \leqslant h_2] = 1 - \alpha \tag{38}$$

defines the confidence level of the estimate. In other words, we constrain our estimate to lie between the upper and lower bounds with a prescribed probability. To obtain a close estimate of the probability of error, the difference between the upper and lower bounds must be small. A rule of thumb for a given P_e is that the number of observations required is of the order of $10/P_e$. For high SNR, this may mean extremely long simulation times, so a technique is needed to shorten the simulation runs.

Importance sampling (IS) is based on the fact than when random numbers are drawn from a normal distribution to simulate additive white Gaussian noise (AWGN) in a communications system, the most common or 'expected' value is the mean of the distribution, which is zero. This causes no errors most of the time, and therefore does not lead to any indication of the worst case performance of the system. Values from the rarely occurring portion of the distribution, the tail, are required. In order to achieve this end Shanmugam and Balaban (1980) proposed to bias the distribution before the simulation.

If the original pdf is $f_Y(y)$, a new pdf, $f_Y^*(y)$ is introduced to generate more errors. The P_e can be written as

$$\begin{aligned} P_e &= \int_{-\infty}^{\infty} \delta_0(y)\frac{f_Y(y)}{f_Y^*(y)} f_Y^*(y)\,\mathrm{d}y \\ &= \int_{-\infty}^{\infty} \delta_0^*(y) f_Y^*(y)\,\mathrm{d}y, \end{aligned} \tag{39}$$

where

$$\delta_0^*(y) = \delta_0(y)\frac{f_Y(y)}{f_Y^*(y)} \tag{40}$$

and $\delta_0(y)$ is given by (36); (40) can be interpreted as weighting the errors counted by the factor

$$W(y) = \frac{f_Y(y)}{f_Y^*(y)} \tag{41}$$

or, if the sample of the RV is selected from the new pdf $f_Y^*(y)$, its probability of being selected has increased by the factor $B(y)$, called the bias and given by

$$B(y) = \frac{f_Y^*(y)}{f_Y(y)}. \tag{42}$$

The P_e can now be estimated by

$$\hat{P}_e = \frac{1}{N_{\mathrm{IS}}}\sum_{i=1}^{N_{\mathrm{IS}}} \delta_0^*(y_i), \tag{43}$$

where N_{IS} is the number of samples used in the importance sampling simulations.

An unbiasing scheme is applied after the approximation $\hat{P}_e$ of (43) is computed. Shanmugam and Balaban (1980) suggested a biasing scheme of the form

$$B(y) = \frac{c}{[f_Y(y)]^{\alpha}}, \tag{44}$$

where the constants c and α are chosen so that the sampling size savings factor, r in (45), is maximized, while ensuring the new distribution is a valid pdf by satisfying (46):

$$r = \frac{N_{\mathrm{MC}}}{N_{\mathrm{IS}}}, \tag{45}$$

where N_{MC} is the number of samples needed without IS, approximately $10/P_e$.

$$\int_{-\infty}^{\infty} B(y) f_Y(y)\,\mathrm{d}y = 1. \tag{46}$$

The approximate expressions for the optimum sample size saving factor r and the biasing factor α are, from Shanmugam and Balaban (1980), given by

$$r_{\mathrm{opt}} = \sqrt{1-\alpha^2}\,\frac{Q(T)}{Q(T\sqrt{1+\alpha})} \tag{47}$$

and

$$\alpha_{\mathrm{opt}} = \frac{-3 + \sqrt{9 + 4T^2(1+T^2)}}{2T^2}, \tag{48}$$

where T is the decision threshold and Q is the complimentary error function, and independence of the variables is assumed.

The choice of the subset in the TCM system is based on the convolutional encoding of m input bits to form $m+1$ encoded bits. This property of TCM violates the assumption that the λ input bits influencing the decoding decision are independent. However, the noise is AWGN and delivers uncorrelated samples.

IS can then be applied exclusively to the noise and the probability of error for the system can be written

(Hecht-Nielsen, 1990) as

$$P_e = \int_{-\infty}^{\infty} \left\{ \delta_0[g(\mathbf{x}+\mathbf{n})] f_{\mathbf{x}}(\mathbf{x}) \frac{f_N(\mathbf{n})}{f_N^*(\mathbf{n})} \right\} f_N^*(\mathbf{n})\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{n}, \tag{49}$$

where the bold typeface denotes that **x** and **n** are vectors and $f_{\mathbf{x}}(\mathbf{x})$ and $f_N(\mathbf{n})$ are the density function for the signal and noise vectors, respectively.

A critical parameter in the application of IS to our neural Viterbi decoder is the identification of the λ bits that affect a single decoding decision. An ideal decoder would base its decision on an infinite sequence of inputs, but as mentioned previously, practical considerations force the decoder to make a decision for some decoding depth Γ. A decoding decision is made on the oldest of the Γ symbols stored in memory. Thus, the decoder is definitely influenced by these Γ symbols. Each symbol in our simulation is part of a subset that is selected by the convolutional encoder of rate $m/(m+1)$ at the transmission end of the system. The decoder decision thus depends on $\Gamma * m$ input bits. Herro and Novack (1988) have conjectured that this decision will also depend on the $\Gamma * m$ input bits prior to the symbol to be decoded as well as the m bits of the current block. If we use the minimum decoding depth as given by Forney (1974), then the expression for λ bits that affect a decoding decision, according to Herro and Novack (1988), is

$$\lambda = m(10\upsilon + 1), \tag{50}$$

where υ is the shift register length (memory order) of the encoder and m is the number of input bits to the encoder at a time. For our simulation with the rate 1/2 encoder of Fig. 3, $\upsilon = 2$, $m = 1$, and $\lambda = 21$. Thus, 42 input bits affect a symbol decision, but as 2 bits choose the subset (group of symbols) of that signal, only 21 input signals influence the receiver decision. The choice of this parameter is critical as proven in Shanmugam and Balaban (1980), since extraneous inputs may cause a slight increase in the variance of the estimate of P_e of the system. On the other hand, not accounting for some inputs leads to a biased estimator.

The steps carried out for the biased noise simulation are:

- The P_e of the system at a particular signal to noise ratio (SNR) is defined by E_S/N_0, where E_S is the average symbol energy (10 for the 16-QAM alphabet), and N_0 is the two-sided noise power spectral density of the noise with standard deviation σ; this is obtained, for high SNR, based on the approximate expression for the upper bound of P_e as given by Ungerboeck (1982):

$$P_e \approx N(d_{\text{free}}) Q\left(\frac{d_{\text{free}}}{2\sigma}\right), \tag{51}$$

where $N(d_{\text{free}})$ is the average number of sequences that are at distance d_{free} from the transmitted sequence. Table 1 shows the order of P_e for various AWGN power for the coded 16-QAM system. Eq. (51) is used with $d_{\text{free}} = 4.0$ and $N(d_{\text{free}}) = 2$.
- The sample size required by a counting estimator (Monte Carlo) for the listed probabilities of error for a normalized estimate error ε as in Shanmugam and Balaban (1980) is calculated next. The value of ε chosen for our simulations was 0.5 which implies that the estimate of the probability of error for the system would lie within $\pm P_e$ of the actual value with a probability of 95%. Table 1 also shows the order of magnitude of the sample size for the MC simulations, N, for different values of P_e, with

$$N = \frac{1}{\varepsilon^2 P_e}. \tag{52}$$

- The sample size saving factor r and the bias value α for various SNRs are obtained from Herro and Novack (1988) in their plot of bias vs. reduction factor for the (2,1,2) code which we are using for our simulation. In case such data are not available, the values of the biasing parameter α and sample size saving factor r are calculated from (48) and (47).
- For each of these r, the sample size of the IS method is calculated using (45). Simulations for sample numbers above 10^4 were reduced via IS with the parameters shown on Table 2.
- AWGN is added to the 2-dimensional QAM signal along each dimension and the bias for each point is calculated using (42). The value of α for the particular SNR is computed from (48).
- The simulation begins and the associated 2D bias values are saved. The 2D biased system is converted

Table 1
Approximate probability of error for 16-QAM

σ	$\frac{d_{\text{free}}}{2\sigma}$	$Q\left(\frac{d_{\text{free}}}{2\sigma}\right)$	P_e	N
0.8	2.50	6.20×10^{-3}	1.2×10^{-2}	10^2
0.7	2.86	2.13×10^{-3}	4.3×10^{-3}	10^3
0.6	3.33	4.29×10^{-4}	8.6×10^{-4}	10^4
0.5	4.00	3.17×10^{-5}	6.3×10^{-5}	10^5
0.4	5.00	2.86×10^{-7}	5.7×10^{-7}	10^7

Table 2
IS parameters

σ	r	α
0.6	05	0.37
0.5	20	0.52
0.4	50	0.60

into a 1D system using

$$b_i = \sqrt{b_{iI}^2 + b_{iQ}^2}, \tag{53}$$

where the I and Q specify the in-phase and quadrature-phase dimensions, respectively.

- Each decoder error is then weighted by the product of the $\lambda/2-1$ one-dimensional biases proceeding the current symbol, the $\lambda/2-1$ biases succeeding the current symbol, as well as the one dimensional bias associated with the current symbol itself.
- The IS probability of error, $\hat{P}_e$, is computed using the weighed errors with Eq. (43), which is the unbiased estimate (Jeruchim, 1984) of the probability of error.

Once subset decoding using the neural decoder is established, signal recognition within the chosen subset was simulated. Two-dimensional noise was added to the constellation and its effect on the pattern recognition network was then studied. The noise margins in the pattern recognition problem case are higher because of the set partitioning techniques, and this is reflected in the plots of probability of missclassification vs. SNR, discussed in the next section.

6. Results

We obtained encouraging results from both neural networks—the RBFN and the MLP—designed and implemented to perform adaptive Viterbi decoding of trellis-encoded 16-QAM. The performance of the neural decoder was compared against the node error probability bound. The graph of the node error probability vs. SNR (E_S/N_0) is shown in Fig. 11 for the AWGN case, where the baseline curve for comparison, uncoded 8-PSK, was computed using the bound in (12) and is also shown. A gain of almost 3 dB is achieved with the neural decoder for operating BER as high as 10^{-3}.

The goal was to achieve the same performance for the neural decoder as with the analytical method (bound) for, at least, high SNR. We observed that the P_e of the neural decoder converged to that of the bound given by Ungerboeck (1987), for SNR $\geqslant 9$ dB. The performance of the standard Viterbi decoder is indistinguishable from that of the neural decoder in cases where AWGN is the only impairment. Simulations for SNR higher than 10 dB were too time-consuming when using the simple IS method implemented. More powerful IS techniques have been recently introduced in Kim and Iltis (2000), and could be used for higher SNR.

The value of σ of the PU transfer function is of critical importance in optimum performance. At high SNR and $\sigma = 1.5$, the decoder missclassifies more noisy signals than with $\sigma = 2.0$. A higher spread factor of 2.5 proved to give a slight increase in P_e. Further investigation into spread factors greater than 2.5 produced similar results. In fact, $\sigma = 2$ proved to be optimum, and higher values yielded no significant improvement or worsening of the resulting P_e. Further authentication of this value is that it conforms to the value obtained using the P-nearest heuristic for the design of an RBFN as given by Fa-Long and Unbenhauen (1997). Also, $\sigma = 2$ is the distance between nearest neighbors in the simulated 16-QAM constellation.

As previously mentioned, the centers of the PU may (and should) be set to the known constellation centers, if a pre-selected set, known a priori, is available, as it would be in normal circumstances. In order to adapt to

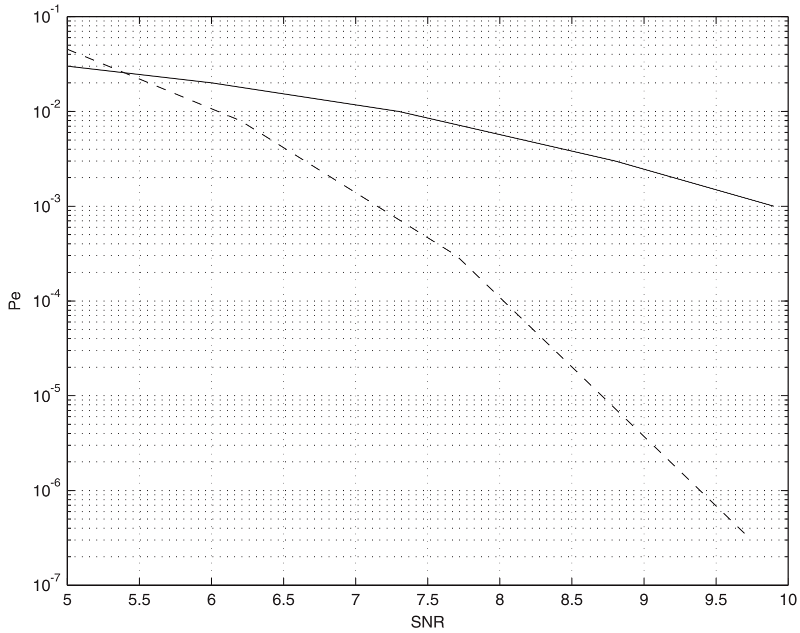

Fig. 11. Probability of error vs. symbol SNR for the neural decoder (dashed) and the baseline 8-PSK uncoded system (solid).

channel imperfections, often time-varying, adaptation of the center of the PUs is performed.

We chose to use a clustering algorithm to initialize the PUs before the network was simulated, assuming the (I, Q) values of the constellation used are unknown. A training sequence of 10 000 samples was originally used for each PU, and shown to be much larger than actually needed; the training sequence was then reduced to 100 samples without performance degradation. The nature of the noise determines the final centers of the PUs; if only AWGN is present, the trained centers correspond very closely to the noiseless case, i.e. ± 1 and ± 3 in each dimension. The concept of 'center learning' for the PUs was validated by simulating a common impairment in communication channels: amplitude imbalance. Amplitude imbalance occurs when the amplitude levels of the quadrature carriers are not set correctly which, in turn, may be the result of a non-linear modulator on an inaccurate D/A converter. A learning rate of 0.05 was used and it was found that the PUs quickly adapted their centers to account for the amplitude imbalance.

To compare the performance of the trained and untrained center case, amplitude imbalance was simulated on the Q channel where coordinates 1 and -1 were mapped to 1.6 and -1.3, respectively. The unbalanced and noisy constellation is shown in Fig. 12. The signal cloud around the centers is due to the AWGN which was assumed to interfere with the training sequence as well. The results obtained for the simulation are shown in Fig. 13. The plot shows that the trained PU centers helped the neural subset decoder perform better than a net with fixed PU centers, and better than a fixed Viterbi decoder. Approximately 0.5 dB improvement is obtained using the adaptive neural decoder. The third curve is given as a reference, and corresponds to the probability of error bound if no amplitude imbalance is present.

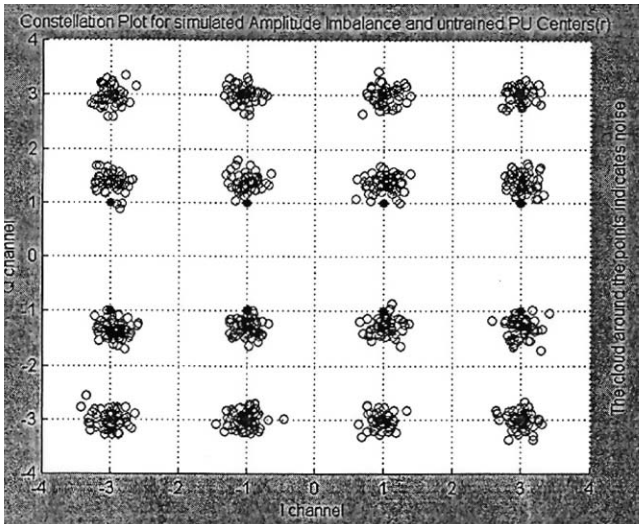

Fig. 12. Noisy 16-QAM constellation with amplitude imbalance in the Q channel. Dark dots at $\pm 1$and ± 3 indicate the centers prior to adaptation.

The pattern recognition network developed performed well under all SNR conditions. The pattern classification results for the neural networks corresponding to the subset S_0 are shown in Fig. 14. The other three signal classifiers have similar curves, and the overall system has the same curve because only the results of

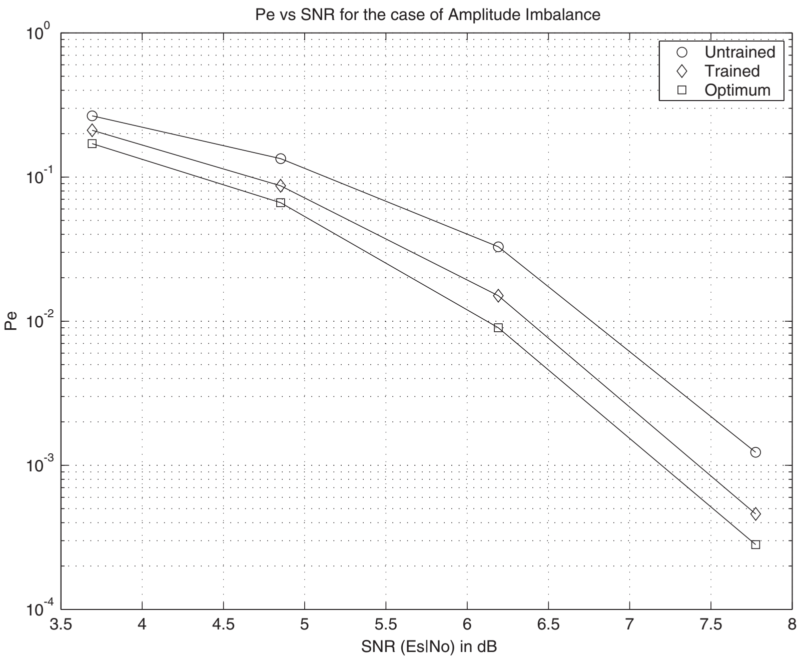

Fig. 13. Probability of error vs. symbol SNR for the system with amplitude imbalance. Trained neural, untrained neural (equal to standard VA decoding), and ideal without amplitude imbalance curves are shown.

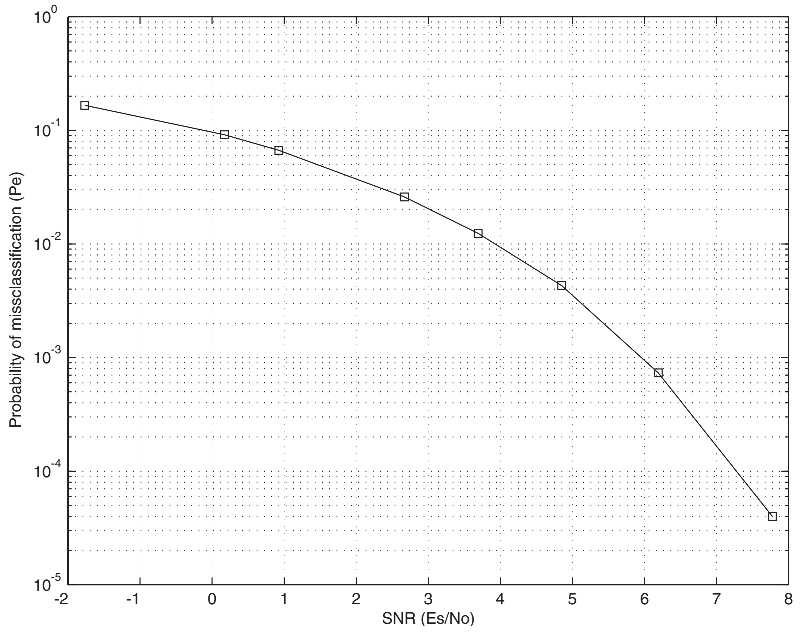

Fig. 14. Probability of error vs. SNR for the MLP signal mapper/classifier for subset S_0.

one of the MLP networks are used each time, the rest are neglected. The P_e vs. SNR indicates excellent performance. An added feature of the MLP classifier is that the signals are simultaneously mapped into their corresponding output bits by the neural network.

The overall performance of the neural TCM decoder is a combination of the two subsystems. The functionality of the TCM decoder was separated into a subset decoding network which estimated the subset being sent (the RBFN path selector networks) and the signal selector network (the MLP classifier/mapper).

The performance of the neural system exactly matches that of the ideal Viterbi decoder if no effects other than AWGN are introduced by the channel. The performance of the neural system is better than the standard Viterbi decoder when other imperfections are included, due to the adaptive nature of the neurons in the neural decoder.

7. Conclusions

Directed graphs in Viterbi decoding share a commonality with artificial neural networks in physical structure as well as functionality. The metric in the Viterbi algorithm as applied to TCM, and the basic functional unit of a RBFN are related with regards to the Euclidean distance parameter both calculate. We exploited this similarity by implementing an adaptive Viterbi decoder for TCM using radial basis function networks.

We also showed that simple neural networks are capable of mapping non-binary signals into the bits being transmitted in a digital communications system, therefore decoding non-binary modulation schemes. The design technique for MLP's proposed in Bose and Garga (1993) was validated in a communications systems environment. Simple neural networks like MLPs were shown to be sufficient to perform subset signal decoding and mapping, without the need for training.

The adaptive decoder presented is able to learn channel conditions as well as equipment defects as they occur. This property was observed and tested in the subset decoding net where the center of the pattern units in the RBFN adapted to the amplitude imbalance condition and the probability of error of the neural decoder was lower than that of a fixed decoder.

The neural decoder system presented in this paper was applied to decode trellis-encoded 16-QAM, but can easily be implemented for other TCM systems, including those using multidimensional constellations. Extensions to channels others than AWGN, such as Rayleigh fading channels, are also possible. Having the TCM decoder adapt to the fading channel characteristics will improve the system's performance over those of fixed detectors.

References

Al-Mashaq, K., Reed, I., 1994. The use of neural nets to combine equalization with decoding for severe intersymbol interference channels. IEEE Transactions on Neural Networks 5 (6), 982–988.

Bose, N., Garga, A., 1993. Neural network design using voronoi diagrams. IEEE Transactions on Neural Networks 4 (5), 778–787.

Bossert, M., 1999. Channel Coding for Telecommunications. Wiley, New York, NY.

Buckley, M., Wicker, S., 1999. A neural network for predicting decoder error in turbo decoders. IEEE Communications Letters 3 (5), 145–147.

Buckley, M.E., Wicker, S.B., 2000. The design and performance of a neural network for predicting turbo decoding error with applications to hybrid ARQ protocols. IEEE Transactions on Communications 48 (4), 566–576.

Clark, G., Cain, J., 1981. Error-Correction Coding for Digital Communications. Plenum Press, New York.

Fa-Long, Unbenhauen, R., 1997. Applied Neural Networks for Signal Processing, 2nd Edition. Cambridge University Press, Cambridge.

Forney, G., 1974. Convolution codes II: maximum likelihood decoding. Information and Control 25, 222–266.

Hagan, M., Demuth, H., Beale, M., 1996. Neural Network Design. PWS Publishing Company, Boston.

Hassoun, M., 1995. Fundamentals of Artificial Neural Networks. MIT Press, Cambridge, MA.

Haykin, S., 2000. Adaptive digital communication receivers. IEEE Communications Magazine 39 (12), 106–114.

Hecht-Nielsen, R., 1990. Neurocomputing. Addison-Wesley, Reading, MA.

Herro, M., Novack, J., 1988. Simulated Viterbi decoding using importance sampling. IEEE Proceedings 135 (2), 133–294.

Jeruchim, C., 1984. Techniques for estimating the bit error rate in the simulation of digital communication systems. IEEE Journal on Selected Areas in Communications SAC-2 (1), 153–169.

Kaminsky, E., Ayo, J., Cartwright, K., 2002. TCM without constellation expansion penalty. Journal of Communications and Networks 4 (2), 1–7.

Kim, K., Iltis, R.A., 2000. An importance sampling technique for a symbol-by-symbol TCM decoding/equalization. IEEE Transactions on Communications 48 (7), 1141–1150.

Kschischang, F., Frey, B., Loeliger, H.-A., 2001. Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory 47 (2), 498–519.

Lee, S.-L., 1993. Neural networks for routing of communication networks with unreliable components. IEEE Transactions on Neural Networks 4 (5), 854–863.

Lin, S., Costello, D., 1983. Error Control Coding: Fundamentals and Applications. Prentice-Hall, Englewood Cliffs, NJ.

Mehmet, M., Kamoun, F., 1993. Neural networks for shortest path computation and routing in computer networks. IEEE Transactions on Neural Networks 4 (3), 941–954.

Minsky, M., Papert, S., 1990. Perceptrons, Expanded Edition. MIT Press, Cambridge, MA.

NeuralWare Technical Publications Group, 1993. Neural Computing, Vol. NC. NeuralWare, Inc., Pittsburg, PA.

Okabe, B., Sugihara, K., 1992. Spatial Tesselations—Concepts and Applications of Voronoi Diagrams. Wiley, New York.

Petsche, T., Dickinson, B., 1987. A trellis-structured neural network. Proceedings of the 1987 Conference on Neural Information Processing Systems, San Diego, CA, pp. 592–601.

Petsche, T., Dickinson, B., 1990. Trellis codes, receptive fields and fault tolerant self-repairing neural networks. IEEE Transactions on Neural Networks 1 (2), 154–166.

Proakis, J., 2000. Digital Communications. Electrical and Computer Engineering, 4th Edition. McGraw-Hill, Boston, MA.

Shanmugam, K., Balaban, P., 1980. A modified Monte-Carlo simulation technique for the evaluation of error rate in digital communication systems. IEEE Transactions on Communications COM-28 (11), 1916–1924.

Sheng Chen, G., Mulgrew, B., 1993. A clustering technique for digital communications channel equalization using radial basis function networks. IEEE Transactions on Neural Networks 4 (4), 570–578.

Ungerboeck, G., 1982. Channel coding with multilevel/phase signals. IEEE Transactions on Information Theory IT-28 (1), 55–66.

Ungerboeck, G., 1987. Trellis-coded modulation with redundant signal sets, part I: introduction. IEEE Communications Magazine 25 (2), 5–11.

Viterbi, A., 1971. Convolution codes and their performance in communication systems. IEEE Transactions on Information Theory IT-13, 260–269.

Wei, L., 1987. Trellis-coded modulation with multidimensional constellations. IEEE Transactions on Information Theory IT-33, 483–501.

Werbos, P., 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. Thesis, Harvard University.

Wicker, B., 1995. Error Control Systems for Digital Communication and Storage. Prentice Hall, Englewood Cliffs, NJ.

Wilson, S., 1996. Digital Modulation and Coding. Prentice-Hall, Upper Saddle River, NJ.

Winch, G., 1993. Telecommunication Transmission Systems. McGraw-Hill, New York.