IMPROVING THE PERFORMANCE OF A RECURRENT NEURAL NETWORK CONVOLUTIONAL DECODER

Klaus Hueske and Jürgen Götze
Information Processing Lab
University of Dortmund
Otto-Hahn-Str. 4, 44221 Dortmund, Germany
Email: klaus.hueske@uni-dortmund.de

Edmund Coersmeier
Nokia GmbH
Nokia Research Center
Meesmannstrasse 103, 44807 Bochum, Germany
Email: edmund.coersmeier@nokia.com

***Abstract*—The decoding of convolutional error correction codes can be described as combinatorial optimization problem. Normally the decoding process is realized using the Viterbi Decoder, but also artificial neural networks can be used. In this paper optimizations for an existing decoding method based on an unsupervised recurrent neural network (RNN) are investigated. The optimization criteria are given by the decoding performance in terms of bit error rate (BER) and the computational decoding complexity in terms of required iterations of the optimization network. To reduce the number of iterations and to improve the decoding performance, several design parameters, like shape of the activation function and level of self-feedback of the neurons are investigated. Furthermore the initialization of the network, the use of parallel decoders and different simulated annealing techniques are discussed.**

I. Introduction

The decoding of convolutional channel codes can be described as a non-linear optimization problem, where the decoder has to minimize the distance between the received noisy code sequence and the estimated code sequence to fulfill the maximum likelihood criterion. This minimization is generally performed by the Viterbi algorithm [1], which determines the shortest path in a weighted graph, given by the trellis representation of the convolutional encoder.

However, the decoding process can also be realized using so-called optimization networks. In 1983 Farrell and Rudolph [2] presented a decoder for linear block codes using local optimization. A decoding algorithm for convolutional codes that is based on RNNs was presented by Hämäläinen and Henriksson [3] in 1999. Further extensions to this method by Berber [4], [5] led to a soft output decoder, which can be used as component decoder for Turbo codes.

The implementation complexity of the Viterbi algorithm is determined by two parts: The so-called Add-Compare-Select (ACS) operation [6], which has to be performed for all nodes in the graph and the memory requirement, as all paths and metrics of the trellis graph have to be stored. Furthermore the complexity is growing exponentially with the memory length of the encoder [7], which is significant, as an increased memory length will improve the error correction qualities of the convolutional code.

In contrast, the complexity of the RNN based decoder is dominated by simple arithmetic functions, i.e. additions and multiplications. The storage requirements depend on the number of neurons in the network, i.e. the number of decoded bits and is independent of the memory length of the convolutional encoder. However, the number of required iterations to stabilize the network is influenced by the memory length.

Even if the Viterbi decoder is an optimum maximum likelihood decoder, the RNN based decoder is an interesting alternative in specific applications: Firstly, as the algorithm is based on additions and multiplications it seems to be more suitable for general purpose processor architectures that do not have special acceleration for ACS operations. Secondly, in terms of adaptive decoding methods, like e.g. used in cognitive radio approaches [8], the decoding performance can be easily adapted by varying the number of iterations. An adaptive behavior in terms of a trade-off between iterations and decoding performance can be realized. Thirdly, the possibility of parallel implementations for artificial neural networks allows systems enabling high data throughput and low latency.

To achieve the best decoding performance while using as little iterations as possible, different optimization approaches for the decoder are discussed in this paper. The presented methods are derived using the original decoder design as presented in [3].

The paper is organized as follows: In section II the underlying decoding problem will be described. Furthermore the structure of the RNN decoder is derived using the gradient descent algorithm. In section III optimizations for the RNN decoder are presented. At first the used activation function and the level of self-feedback between the neurons are discussed. The initialization of the neurons can be optimized using information obtained from the noisy received sequence. Parallel decoders and a deterministic simulated annealing technique are introduced to reduce the risk of sub optimum results. Conclusions are drawn in section IV.

II. RNN Decoder Basics

The RNN based decoder can be used with arbitrary code rate convolutional codes. To simplify the derivation of the decoder structure, the code rate is limited to $R = 1/n$ throughout this paper. Furthermore most practically used codes are based on rates $R = 1/n$, applying puncturing to obtain higher code rates.

The encoding process can be described as a matrix-vector multiplication

$$\boldsymbol{c}_b = \boldsymbol{G}^T \boldsymbol{a}_b, \tag{1}$$

where $\boldsymbol{G}$ is the generator matrix of the convolutional encoder in time domain and $\boldsymbol{a}_b$ the information bearing input sequence of length l. The resulting coded sequence of length nl is given by $\boldsymbol{c}_b$. All sequences are $\in GF(2)$, i.e. contain binary values $\{0, 1\}$.

As we use an antipodal transmission scheme the additive group $\{0, 1\}$ of $GF(2)$ can be transformed to the multiplicative group $\{-1, 1\}$. The components of the resulting antipodal coded vector $\boldsymbol{c}$ can then be described as

$$c_k^{(q)} = -\prod_{p=1}^{K} (-a_{k+1-p})^{g_p^{(q)}}, \tag{2}$$

with $c_k^{(q)} \in \{-1, 1\}$ the k-th bit of the output sequence of the q-th encoder output. The exponent $g_p^{(q)}$ is the p-th component of the impulse response of the q-th encoder output. The range of $q = 1...n$ is limited by the number of encoder outputs n, the index $p = 1...K$ is limited by the constraint length K of the used code.

Example: Throughout this paper we will use an example code with $R = 1/2$ and $K = 3$. The encoder in polynomial notation is given by $(D^2 + 1)$ and $(D^2 + D + 1)$. A section of the resulting code sequence obtained from equation (2) is depicted below.

$$\boldsymbol{c}_k = \begin{pmatrix} c_k^{(1)} \\ c_k^{(2)} \end{pmatrix} = \begin{pmatrix} -a_k a_{k-2} \\ a_k a_{k-1} a_{k-2} \end{pmatrix}$$

During transmission the code sequence $\boldsymbol{c}$ will be corrupted by channel influences like noise and interference. The noisy received sequence $\boldsymbol{r} \in \mathbb{R}^{nl}$ is then given as $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$, with $\boldsymbol{e} \in \mathbb{R}^{nl}$ describing the error. For maximum likelihood decoding we have to estimate a valid code sequence $\hat{\boldsymbol{c}}$, that minimizes the distance to the received vector $\boldsymbol{r}$. This can be described as minimization problem

$$\min_{\hat{\boldsymbol{c}}} \|\boldsymbol{r} - \hat{\boldsymbol{c}}\|. \tag{3}$$

The expression $\|...\|$ is defined as hamming weight in case of hard decision, which means that a hard limiter is used in the receiver's detector. However, to improve the error correction performance, soft decision is used in most of the receivers. The so-called soft bits contain information about the reliability of the detector's decision. As the hamming weight is not applicable for these non-binary sequences, for soft decision the distance is defined by the Euclidean norm.

Optimization problems in context with artificial neural networks are often described using a so-called energy function [9]. We can define this function as

$$E(\hat{\boldsymbol{c}}) = \|\boldsymbol{r} - \hat{\boldsymbol{c}}\|_2^2 = \sum_{k=0}^{l-1} \sum_{q=1}^{n} \left[r_k^{(q)} - \hat{c}_k^{(q)} \right]^2. \tag{4}$$

We can also define this energy function subject to the estimated data sequence $\hat{\boldsymbol{a}}$, using the encoder rule given in equation (2). The energy function, which has to be minimized iteratively by the optimization network, is then given as

$$E(\hat{\boldsymbol{a}}) = \sum_{k=0}^{l-1} \sum_{q=1}^{n} \left[r_k^{(q)} + \prod_{p=1}^{K} (-\hat{a}_{k+1-p})^{g_p^{(q)}} \right]^2. \tag{5}$$

Example: For the example code with $n = 2$ we obtain two separated sums, according to the encoder outputs:

$$E(\hat{\boldsymbol{a}}) = \sum_{k=0}^{l-1} \left[r_k^{(1)} + \hat{a}_k \hat{a}_{k-2} \right]^2 + \sum_{k=0}^{l-1} \left[r_k^{(2)} - \hat{a}_k \hat{a}_{k-1} \hat{a}_{k-2} \right]^2 \tag{6}$$

The minimization of $E(\hat{\boldsymbol{a}})$ describes a combinatorial optimization problem, i.e. we have to find discrete estimates $\hat{\boldsymbol{a}} \in \{-1, 1\}^l$. This can be realized by applying an optimization network based on an RNN using the gradient descent method [3]. The corresponding network structure can be derived from the gradient equation

$$\nabla E(\hat{\boldsymbol{a}})_k = \frac{\partial E(\hat{\boldsymbol{a}})}{\partial \hat{a}_k}. \tag{7}$$

Example: The gradient for the example code can be obtained by calculating the partial derivations of $E(\hat{\boldsymbol{a}})$. Assuming the $\hat{\boldsymbol{a}} \in \{-1, 1\}^l$, all quadratic terms can be omitted as $1^2 = (-1)^2 = 1$. The gradient component k is then given as

$$\nabla E(\hat{\boldsymbol{a}})_k = \frac{\partial E(\hat{\boldsymbol{a}})}{\partial \hat{a}_k} = 2(r_k^{(1)} \hat{a}_{k-2} - r_k^{(2)} \hat{a}_{k-1} \hat{a}_{k-2} - r_{k+1}^{(2)} \hat{a}_{k+1} \hat{a}_{k-1} + r_{k+2}^{(1)} \hat{a}_{k+2} - r_{k+2}^{(2)} \hat{a}_{k+1} \hat{a}_{k+2} + 5\hat{a}_k) \tag{8}$$

If we figure the energy function $E(\hat{\boldsymbol{a}})$ as a surface with maxima and minima, we can use the gradient information to determine the way to the steepest descent, which should be the shortest way to a minimum of the surface. An iterative algorithm using these relations is the gradient descend method [10]. The iterative update rule for a component $\hat{a}_k$ of the sequence to be minimized is given as

$$\hat{a}_k^{[i+1]} = \hat{a}_k^{[i]} - \mu \frac{\partial E(\hat{\boldsymbol{a}}^{[i]})}{\partial \hat{a}_k}, \tag{9}$$

with i the iteration index and μ the step size [10]. Using a so-called activation function f_A [11], we can define the $\hat{a}_k$ as neurons of a recurrent neural network. The minimization process is realized by iterating all neurons in this network. The update rule for a neuron $\hat{a}_k$ is then given as

$$\hat{a}_k^{[i+1]} = f_A(\hat{a}_k^{[i]} - \mu \frac{\partial E(\hat{\boldsymbol{a}}^{[i]})}{\partial \hat{a}_k}). \tag{10}$$

The decoding problem is a non-linear problem, i.e. the corresponding error surface has multiple minima, which leads to possible local suboptimum solutions during the optimization process. To reduce the risk of getting stuck in a local minimum, simulated annealing techniques [11] can be applied to the network. A common way is to add Gaussian noise during the update process [4], which is also known as Gaussian annealing. For the network to stabilize, the variance of the

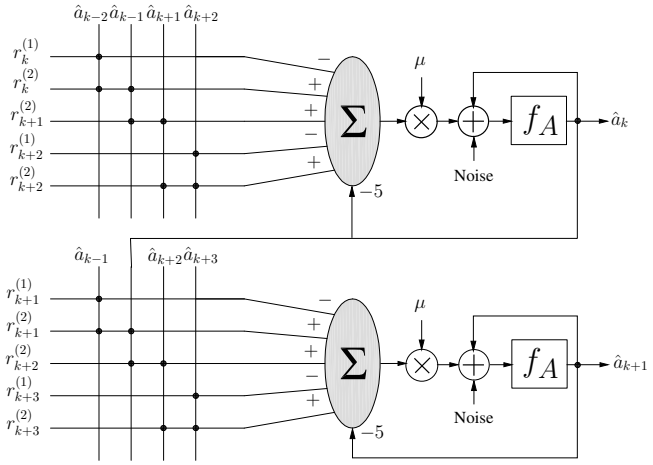

Fig. 1. Structure of the RNN convolutional decoder

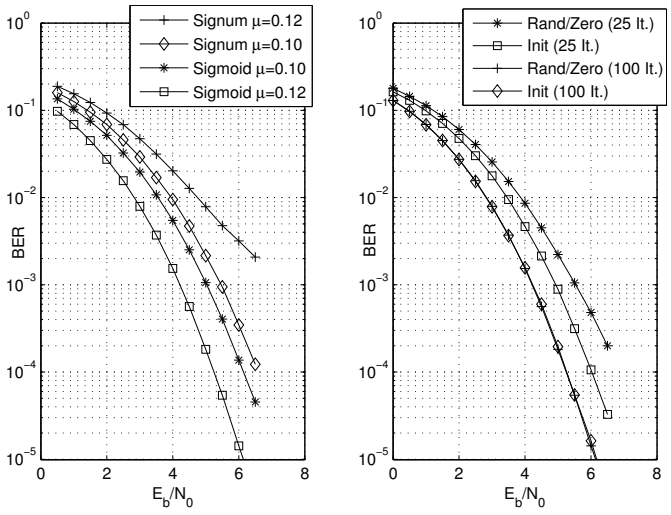

Fig. 2. Influence of activation function and self-feedback factor (left) and different initialization methods (right)

noise has to be decreased during the iteration process. With $n^{[i]}$ the sampled Gaussian noise, the modified update rule is

$$\hat{a}_k^{[i+1]} = f_A(\hat{a}_k^{[i]} - \mu \frac{\partial E(\hat{\boldsymbol{a}}^{[i]})}{\partial \hat{a}_k} + n^{[i]}). \tag{11}$$

The resulting structure of the recurrent neural network for the example code is depicted in Figure 1. Note that this structure is valid for arbitrary μ, i.e. allows self-feedback of the neurons.

III. RNN Decoder Optimizations

If and how fast the optimization network stabilizes depends on the chosen initialization of the neurons, the shape of the used activation function and value of the step size μ. Therefore also the quality of the decoder in terms of decoding errors is influenced by these options. Optimizations of these design parameters are discussed in this section.

A. Activation Function and Self-Feedback

The step size μ introduced with the gradient descent algorithm determines the level of self-feedback of the neurons and can therefore also be called feedback factor (compare equations (8) and (9)). In the original paper by Hämäläinen and Henriksson [3], the feedback factor is always chosen in a way such that the self-feedback disappears in the network structure.

Example: For the self-feedback to disappear for the example network, the factor has to be chosen as $\mu = 0.1$. Comparing equations (8) and (9) one can see that the $\hat{a_k}$ are then eliminated in the update expression.

The avoidance of self-feedback is required to guarantee a stable network when using signum activation functions. In the original design [3] the signum function was chosen to enable an easy hardware implementation. However, in literature (e.g [11]) the so-called sigmoid function $f_A(x) = \tanh(x)$ is widely discussed in order to reduce the risk of getting stuck in local minima.

Introducing a partial self-feedback (increasing μ) in a network using sigmoid activation functions does not significantly affect the stability, but can increase the convergence rate of the network, due to the increased step size. To clarify this, BER simulations for both, signum and sigmoid activation functions using different values μ, are presented in Figure 2 (left hand side). The simulations are based on the example code and use soft bits obtained from a Binary Phase Shift Keying (BPSK) modulated transmission over an Additive White Gaussian Noise (AWGN) channel. The decoder network uses 100 iterations.

Comparing the networks without self-feedback ($\mu = 0.1$) we can see that the sigmoid activation function is superior to the signum activation function, as the risk of getting stuck in local minima is reduced.

Enabling self-feedback ($\mu = 0.12$) when using signum activation functions leads to oscillations in the network, which makes a stabilization more difficult or even impossible. The result is a significantly increased BER. However, when using the sigmoid function, the network is still stable and, due to the increased μ, the network converges faster, which results in a decreased BER, when using a fixed number of iterations. Even if the implementation complexity of the sigmoid function is higher compared to the signum function, the number of required iterations to obtain identical results can be decreased significantly.

B. Initialization

The presence of local minima is one the most difficult problems to deal with in optimization networks. The use of Gaussian annealing and sigmoid functions were already presented to reduce the risk of these sub optimum solutions.

A topic that was not mentioned yet is the initialization of the network neurons before starting the iteration process. In most cases optimization networks are initialized by assigning random values or zeros to all neurons in the network [3]. It is obvious that for optimization problems with multiple

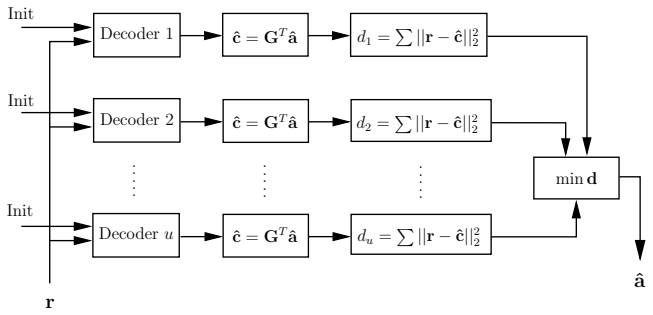

Fig. 3. Structure of the multiple network decoder

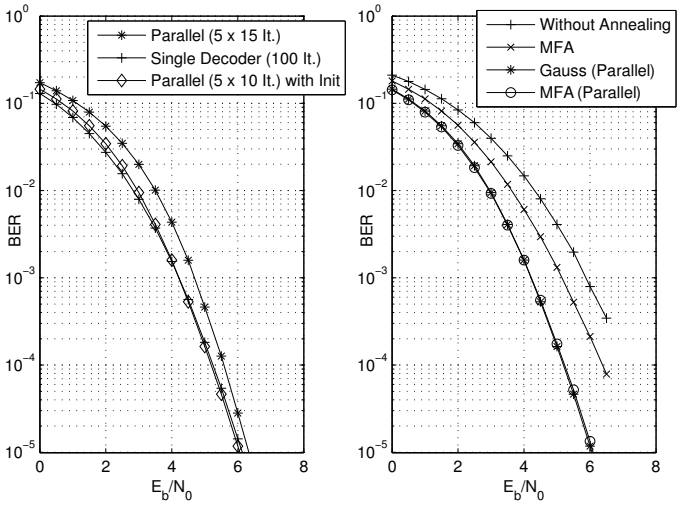

Fig. 4. Performance of a decoder using multiple networks (left) and different annealing techniques (right)

minima the initialization values have a huge influence on which minimum will be reached. Hence a starting point should be determined that is as close as possible to the optimum global solution.

In [2] the received noisy data $\boldsymbol{r}$ was used as initialization for the optimization network. A similar approach can be used for the proposed decoder by applying the inverse encoder to the received vector $\boldsymbol{r}$. Thus, a first guess of the transmitted codeword can be obtained by multiplying the hard decision $\boldsymbol{r}_b$ of $\boldsymbol{r}$ by the right inverse of the generator matrix, i.e.

$$\hat{\boldsymbol{a}}_{init} = \boldsymbol{G}^{-T}\boldsymbol{r}_b. \tag{12}$$

The right inverse of the generator matrix can be derived from the Invariant Factor Decomposition (IFD) of the generator matrix, like presented in [12].

Example: The right inverse generator matrix for the example code in frequency domain is given as

$$\boldsymbol{G}^{-1}(D) = \begin{pmatrix} 1+D \\ D \end{pmatrix}.$$

The decoding performance of the proposed initialization method is compared to random and zero initialization in Figure 2 (right hand side) using BER simulations. If the number of iterations is high (e.g. 100), the BER performance of the decoder is independent from the used initialization method, because its impact is minimized by the large number of added Gaussian noise samples.

However, for a limited number of iterations (e.g. 25), the inverse encoder initialization performs better than the random/zero initialization. This means, the proposed initialization cannot outperform the random initialization, but it can reduce the required number of iterations to achieve a specific BER, which was also described in [2].

C. Multiple Networks

The output of the optimization network is determined by the input sequence $\boldsymbol{r}$, the initialization of the neurons and the applied Gaussian noise during the iteration process. This means, even for identical input sequences $\boldsymbol{r}$ multiple networks can produce different results. This fact can be used to further reduce the problem with local minima.

Like suggested in [3] a number of u independent networks is fed by an identical input sequence $\boldsymbol{r}$. The results are then re-encoded (compare equation (2)) and the distance d to the received vector $\boldsymbol{r}$ is calculated. The output sequence with corresponding minimum distance d will then be chosen as final decoder output. The configuration of the decoder is depicted in Figure 3.

The use of multiple networks is only reasonable, if the overall number of iterations can be reduced. Figure 4 (left hand side) shows that a decoder with $u = 5$ networks with 15 iterations each (adding up to 75 iterations) performs worse than a single decoder using 100 iterations.

However, in section III-B it was mentioned that the initialization of neurons using inverse encoding is most reasonable for a small number of iterations. Hence, the two approaches, multiple networks and initialization, are combined. The resulting BER for this decoder with $u = 5$ and 10 iterations each is also displayed in Figure 4 (left hand side). The combined approach can reduce the overall number of iterations by 50%, while obtaining a similar BER performance.

An efficient implementation of the multiple network decoder can be realized on parallel architectures, as all networks process independent data.

D. Mean Field Annealing

To reduce the problem with local minima, Gaussian noise is added during the iteration process. Besides this Gaussian annealing technique, other annealing methods are known, like e.g. Boltzmann annealing [11] and mean field annealing (MFA) [13], which is also called deterministic annealing, as no stochastic calculations have to be performed.

In optimization networks using MFA, the shape of the activation function f_A is modified during the iteration process. This is realized by modifying the sigmoid activation function to

$$f_A(x) = \tanh(\frac{x}{T}), \tag{13}$$

with T the so-called artificial temperature of the system.

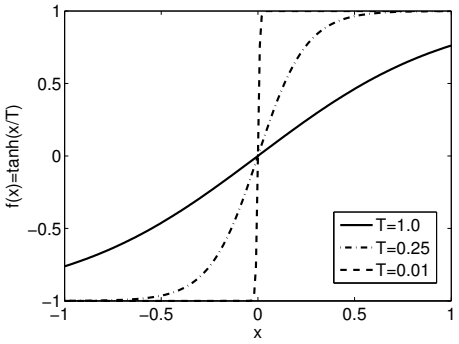

Fig. 5. Activation Function for Mean Field Annealing

Beginning with an initial T, the temperature will be lowered during the iteration process, which causes the shape of the activation function to change. Beginning with a nearly linear characteristic, the shape will change to the classical sigmoid function and finally approximate the signum function. This behaviour is clarified in Figure 5.

Finding the right initial temperature is crucial to obtain a good decoding performance. For quadratic energy functions the initial temperature can be derived from the eigenvalues of the connection weight matrix of the neurons, like shown in [14]. But as the presented decoder is a high order recurrent network [15], this method cannot be applied. To compare the annealing techniques, here the initial temperature is chosen heuristically. The results are given in Figure 4 (right hand side). One can see that MFA outperforms the sigmoid function with constant shape, but does not reach the sigmoid function in combination with Gaussian annealing.

To improve the decoding performance, again parallel decoders are used, each starting at a different initial temperature. The best result is then computed according to section III-C. The performance of the decoder combining parallel decoding and MFA can reach the decoder using Gaussian annealing, like shown in Figure 4 (right hand side).

IV. Conclusion

The presented optimizations for the RNN based convolutional decoder allow a reduction of required iterations and increase the error correction performance, compared to the original decoder. The importance of a suitable choice for the activation function and the self-feedback factor was shown. Using the proposed initialization method can further increase the decoder performance, while the complexity is only slightly increased. The use of parallel decoders allows parallel implementations and can further reduce the overall number of required iterations. Mean field annealing was introduced as an alternative annealing technique. In Figure 6 it is shown that the proposed optimized decoder performs very well compared to the Viterbi decoder, while offering several design parameters that allow a flexible use in specific systems, like e.g. cognitive radio applications. The parallel algorithmic structure allows high-speed hardware implementations, while the underlying arithmetic functions (additions and multiplications) also enable an efficient implementation on general purpose processor architectures.

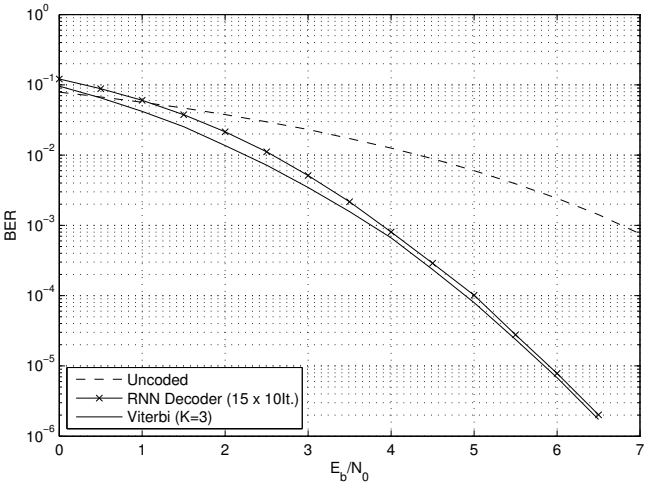

Fig. 6. Simulation Results for the RNN decoder and the Viterbi decoder

References

[1] A.J. Viterbi: "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Technology*, **13**:260–269, April 1967.

[2] K.H. Farrel et al.: "Decoding by local optimization", *IEEE Trans. Information Theory*, **29**:740–743, 1983.

[3] A. Hämäläinen & J. Henriksson: "Convolutional Decoding Using Recurrent Neural Networks", *Proceedings of Inter. Joint Conf. on Neural Networks*, **5**:3323-3327, San Diego, 1999.

[4] S.M. Berber & V. Kecman: "Convolutional decoders based on artificial neural networks", *Proceedings of IEEE International Conference on Neural Networks*, **2**:1551–1556, 2004.

[5] S.M. Berber: "Soft decision output decoding (SONNA) algorithm for convolutional codes based on artificial neural networks", *Proceedings of IEEE International Conference on Intelligent Systems*, **2**:530–534, 2004.

[6] K. K. Parhi: "An improved pipelined MSB-first Add-Compare-Select Unit Structure for Viterbi Decoders", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, **51**:504–511, 2004.

[7] S. Lin & D. J. Costello, Jr.: *Error Control Coding*, 2. edition, Pearson Prentice Hall, 2004.

[8] E. Coersmeier, K. Hueske et al.: "Combining Cognitive Radio and Software Radio Approach for Low Complexity Receiver Architecture", *Proc. of the Int. Sym. on Advanced Radio Technologies (ISART)*, 2007.

[9] J. Hertz, A. Krogh & R. G. Palmer: *Introduction to the Theory of Neural Computation*, Perseus Books Group, January 1991.

[10] S. Haykin: *Adaptive Filter Theory*, 3. edition, Prentice Hall, 1996.

[11] S. Haykin: *Neural Networks: A Comprehensive Foundation*, 2. edition, Prentice Hall, 1998.

[12] R. Johanesson & K. Sh. Zigangirov: *Fundamentals of Convolutional Coding, IEEE Series on Digital & Mobile Communication*, IEEE Press, 1999.

[13] C. Peterson & B. Söderberg: *Neural Optimization*, from *The Handbook of Brain Theory and Neural Networks*, 2. edition, Bradford Books/The MIT Press, 2001.

[14] C. Peterson & B. Söderberg: "Combinatorial Optimization with Feedback Artificial Neural Networks", *Proceedings of ICANN '95 International Conference on Artificial Neural Networks*, October 1995.

[15] L.R. Medsker & L.C. Jain: *Recurrent Neural Networks - Design and Applications*, CRC Press, Berlin, 2000.