JACOBS
UNIVERSITY

Aspects of Sum-Product Decoding of Low-Density-Parity-Check Codes on Neural Networks

by

Robin Nyombi Schofield Ssenyonga

submitted as a guided research thesis for the Bachelor of Science in Electrical and Computer Engineering

Prof. Dr-Ing. Werner Henkel
Name and title of supervisor

Date of Submission: August 26, 2015

Transmission Systems Group (TrSys) — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicated resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise stated; where my results derive from computer programs, these computer programs have been written by me unless otherwise stated. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Signature

Place, Date

Abstract

Neural network rules and coding theory concepts have been associated in the past. It is known that in relation to neural networks, iterative decoding exhibits some nonlinear dynamics. The purpose of this thesis, therefore, is to set forth an evaluation of the concept of neural network design related to Low-Density Parity-Check (LDPC) code structures and algorithmic aspects. A basic introduction to neural networks is given as major aspects of existing network structures are looked into. New ideas relating to the LDPC decoding sum-product algorithm are then introduced and explored in a non-specific manner later in the thesis.

Contents

List of Figures

1 Introduction

As far back as the 1940's, neural networks have been an area of extensive research in an attempt to realize a computational model of the human brain. The majority of this research is concerned with problems in pattern recognition and use of standard neural networks for neural computing. Research has also been conducted in the area of error control coding with applications in communications and computer science.

For nearly as long as the neural networks have been considered, even more significant advances have been made in communications through coding. Shannon published his groundbreaking paper [1] that would eventually lead to advances in information and coding theory such as Reed-Solomon codes, convolutional codes, Turbo and Low-Density Parity-Check (LDPC) codes: all made with an aim of solving Shannon's classical problem of transmitting messages over a noisy channel such that the receiver can receive a message with high probability in what is considered reliable transmission. LDPC codes were first discovered by Gallagher in 1963 [2] but were not really in use until they were "re-discovered" by MacKay and Neal because of their close to ideal performance according to Shannon's limit. Since a lot of work has been directed towards achieving low complexity efficient decoding of LDPC codes, this makes them ideal for a study in relation to neural networks and optimal neural network design, if suitable links can be observed.

The idea of using Artificial Neural Networks to decode LDPC codes is by itself not an entirely new idea [3, 4], however one of the biggest limitations has always been the length of the codes being constrained by the length that is reasonable to train. Using the inherent pattern recognition and generalization abilities of a properly trained neural network can at a constant time enable very high speed, non iterative LDPC decoding, with some significant error performance levels on short codes. Some research has been done on the use of neural networks for purposes of decoding LDPC codes using the multi-layer feed-forward network [4] which does not use the typical probabilistic metrics used in the known decoding methods. We clearly know that standard neural networks are by far not able to compete with the known decoding algorithms for such codes and knowing that this has been successfully implemented as shown in these papers already comes as a surprise. We also note that short codes (which are the ones worked with) have a lot of cycles and hence message passing would not be a suitable decoding algorithm in this case anyway.

My aim in writing this thesis is therefore not to apply standard neural networks for decoding purposes of LDPC codes but rather to check the similarities and of course differences in the design and structure of both the sum-product algorithm and the standard neural network models. Using the extended features of the algorithm in decoding LDPC codes, we then set forth possible ways to perhaps represent the decoding algorithm on some of the neural network structures. We would of course like to understand the actual sum-product algorithm as a kind of neural network and with this thesis, we provide some insight into how this could be done. The "resulting neural networks" actually realize the sum-product algorithm without any major degradation. Attempting to apply standard neural networks for decoding would have only led to failure, since there are optimum algorithms out there, which a neural network can, of course, never beat in performance.

This thesis is organized as follows. In the following, we present a brief outline of the individual sections that comprise the work done. We present the reader with our motivation for this thesis as well as the state of the art contributions that have been made. The second section defines LDPC codes and provides some detail on their design by linear programming. The third section explains the main idea behind neural networks. The different existing structures are explored in this section. The fourth section gives a thorough discussion of the contribution of our work and addresses the current results. Connections between the two topics are drawn. Section 5 concludes with a short comparative discussion and points out possible work topics to complement this work.

2 Low-Density Parity-Check Codes

These, as the name suggests, are a class of linear error-correcting block codes which are represented by a sparsely structured parity-check matrix. For this thesis, the equivalent graphical representation of the codes will also be considered. The code has a true rate $R = k/n$ for a matrix **H** with dimensions $(n-k) \times n$ where n is the length of the codeword and k is the length of the information word.

LDPC codes are represented using bipartite graphs, which have variable nodes corresponding to the elements of the codeword and check nodes corresponding to the parity-check constraints, as well as the edges connecting them. In order to describe how the parity-check matrix of an LDPC code maps to the corresponding graph we introduce the parity-check matrix below with given n and k, the 1s in the matrix indicate which nodes are connected, i.e., the 1s represent edges. Let us consider the parity-check matrix[1] below with $n = 8$ and $k = 4$

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

[1] This is not a typical LDPC matrix since the typical one is usually sparse

The matrix above is represented in Fig. 1 below.

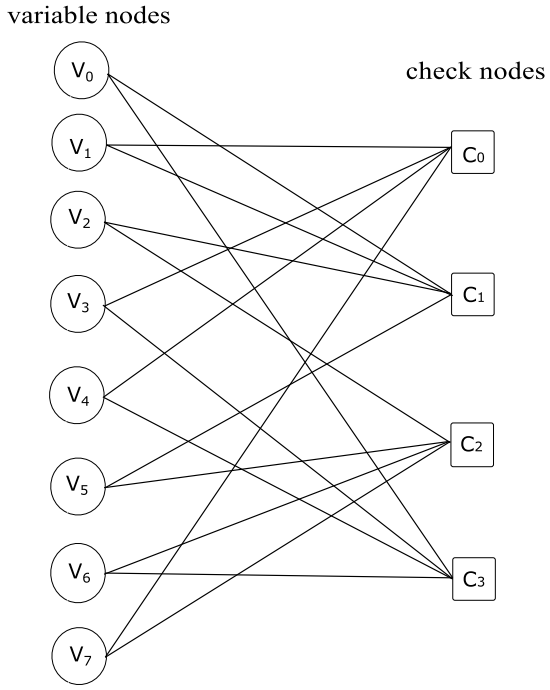

Figure 1: (2,4) regular Tanner graph

2.1 Regular and irregular codes

LDPC codes are generally divided into regular and irregular codes.

On the one hand, regular LDPC codes are so called if the number of 1s per column and per row are constant with the relation $\frac{n-k}{n} = \frac{t_c}{t_r}$, where t_c and t_r are the number of 1s per column and the number of 1s per row, respectively. Graphically, there is the same number of incoming edges for every variable node and also for all the check nodes as shown in Fig. 1 above. In the graph, the variable nodes represent the bits of the codeword while the check nodes define the parity-check conditions. Regular LDPC codes are parametrized by (n, t_c, t_r).

On the other hand, irregular LDPC codes do not have a constant number of 1s in its rows and columns. Their degree distributions are therefore not parametrized using the number of 1s in columns and rows of the matrix but rather with polynomials from the edge and node perspectives. Irregular LDPC codes have been proven to have a significantly better performance than their regular counterparts. Below are the equations used for the degree distributions as in [5].

Using the more commonly used edge perspective[2], we define the variable node degree distribution as

$$\lambda(x) \;=\; \sum_{i=2}^{d_{vmax}} \lambda_i x^{i-1} \tag{2}$$

and the check node degree distribution as

$$\rho(x) \;=\; \sum_{i=2}^{d_{cmax}} \rho_i x^{i-1}\,, \tag{3}$$

where d_{vmax} and d_{cmax} are the maximum variable and check node degrees, respectively, λ_i and ρ_i are the proportions of edges connected to variable and check nodes, respectively.

Using the node perspective, we define the variable node degree distribution as

$$\Lambda(x) \;=\; \sum_{i=1}^{l_{max}} \Lambda_i x^i \tag{4}$$

and the check node degree distribution as

$$P(x) \;=\; \sum_{i=1}^{r_{max}} P_i x^i\,, \tag{5}$$

where l_{max} and r_{max} are the maximum variable and check node degrees, respectively, Λ_i and P_i are the number of variable nodes of degree i and number of check nodes of degree i respectively. We define these parameters such that for a code of length n and rate R, $\sum_i \Lambda_i = n$ and $\sum_i P_i = n(1-R)$.

2.2 Decoding by message passing

LDPC codes are decoded iteratively using a message passing algorithm called the sum-product algorithm (SPA). The SPA decoding technique involves exchange of messages between the variable and the check nodes iteratively along the edges of the Tanner graph.

The variable nodes receive messages from the check nodes to which they are connected and the outgoing messages are then a function of all the incoming messages except the one along the edge on which the outgoing message is to be sent. The same procedure for passing messages is used for the check nodes as well. How the messages pass is essential for a good performance of the algorithm whose efficiency is degraded by a code with short cycles. The girth of a given graph is the length of the shortest cycle and it should therefore be maximized so as to have more efficient decoding. A tree-like structure is desirable since it ensures the independence of the messages and avoids short loops. For this reason, it is always better to have a sparse **H** matrix so as to reduce the dependency between messages or at least its effect on the code performance.

[2]The edge perspective is better to use especially for asymptotic analysis, see [5]

The messages are represented as random variables expressed as log-likelihood ratios (LLRs). The LLR of a received value y is the soft decision decoding metric whose sign determines the bit value and the amplitude is a measure of accuracy in prediction [6]. It can be expressed as $L(x) = \ln \frac{p(x=0|y)}{p(x=1|y)}$ which in turn is expressed in terms of the intrinsic LLR and the a-priori known information which is given by the channel input distribution. $L(x)$ is then equal to $L_{ch} + L_a(x)$. L_{ch} takes the form of the maximum likelihood ratio, i.e., $\ln \frac{p(y|x=0)}{p(y|x=1)}$ hence contains information from the channel. The a-priori information, $L_a(x)$, is simply $\ln \frac{p(x=0)}{p(x=1)}$.

The update rules for the algorithm where $v_m^{(l)}$ represents the LLR message from variable node v_m and the same for the check nodes are shown in the equations below:

Variable node:

$$v_m^{(l)} = u_0 + \sum_{k=1,k\neq m}^{d_v-1} u_k^{(l-1)} \quad \forall\, m = 1, ..., i\ , \tag{6}$$

Check node:

$$u_k^{(l)} = 2\tanh^{-1}\left(\prod_{m=1,m\neq k}^{d_c-1} \tanh\left(\frac{v_m^{(l)}}{2}\right)\right) \quad \forall\, k = 1, ..., j\ . \tag{7}$$

The SPA update rules for the variable and check nodes are illustrated in Fig. 2 below, where u_0 gives the intrinsic LLR of the codeword bit corresponding to the variable node in question based on channel information (LLR of the channel observation).

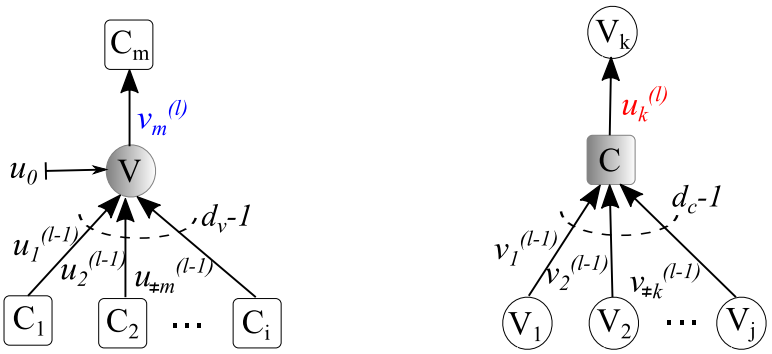

Figure 2: Sum-product algorithm update rules

For every discrete time step[3] l, one iteration of the message exchange begins with the variable nodes processing the information received from the check nodes in the previous iteration and transmitting the messages to their neighbors, continued with the check nodes processing the information received during the same iteration and sending it back to the variable nodes to be used in the following iteration [6]. The initial message from any check node is taken to be 0, such that in the first iteration the variable nodes will transmit the intrinsic information from the channel to the check nodes for them to process the information and send it back to the variable node side to be available during the second iteration.

[3]The current time step, l, is used in superscript

An iteration is complete when all messages have been computed once by all previous equations. We can then compute the a-posteriori ratio after, say, L iterations using:

$$v_{app,n} \;=\; u_0 \;+\; \sum_{k=1}^{i} u_k^L \quad \forall \text{ variable nodes } n = 1, ..., N \;. \tag{8}$$

The final decision on the binary values of the variable nodes is made by (as in [7]):

$$\hat{m}_n = \frac{1 \;-\; \text{sign}(v_{app,n})}{2} \quad \forall \text{ variable nodes } n = 1, ..., N \;. \tag{9}$$

Over a cycle-free graph, we generally use the message passing scheme on factor graphs and easily calculate the different a-priori probabilities (APPs) at the bit nodes by factorizing Bayes' rules. However, since LDPC codes are of finite length in general, this is not the case, however, we only still use the algorithm when we have not too short cycles in the codes. This is why it is important to make the **H** matrix sparse.

Density evolution

In order to optimize the degree distribution of an irregular LDPC code, the decoding behavior has to be investigated. Density evolution is a asymptotic tool used to predict the decoding performance of the algorithm by tracking the distribution (probability density functions) of the messages. We can use it to design codes of finite length that have very good performance. It provides mutual information between iterations and has been demonstrated in [8] and [9] for Gaussian distributions.

The mutual information message from variable to the check nodes in the l^{th} iteration is denoted as $x_{vc}^{(l)}$ while the one between check and variable nodes is $x_{vc}^{(l)}$. Updating the mutual information messages is then done by the evolution equations [7] as

$$x_{cv}^{(l)} \;=\; 1 \;-\; \sum_{j=2}^{d_{cmax}} \rho_j J\left((j-1)J^{-1}\left(1 - x_{vc}^{(l)}\right)\right), \tag{10}$$

$$x_{vc}^{(l)} \;=\; \sum_{i=2}^{d_{vmax}} \lambda_i J\left(\frac{2}{\sigma^2} + (i-1)J^{-1}\left(x_{cv}^{(l-1)}\right)\right), \tag{11}$$

where

$$J(m) \;=\; 1 \;-\; \frac{1}{\sqrt{4\pi m}} \int_{\mathbb{R}} \log_2(1 + e^{-v}) e^{-\frac{(v-m)^2}{4m}} dv \tag{12}$$

as derived in [7] computes the mutual information as a function of the mean and so[4] $x_v \;=\; J(m)$ and v is a random variable such that $v \sim \mathcal{N}(m, 2m)$. These equations are applied for consistent densities. A density $f(x)$ is considered consistent iff $f(x) = e^x f(-x)\; \forall x \in \mathbb{R}$.

[4] J is a strictly monotonous continuous function whose inverse allows us to calculate the mean of messages of such a density determined by $(m, 2m)$

When we combine equations (10) and (11) above, we obtain the density evolution as an explicit function of the degree distributions, the channel noise variance, and the mutual information from iteration $l-1$ as

$$x_{vc}^{(l)} = F\left(\boldsymbol{\lambda}, \boldsymbol{\rho}, \boldsymbol{\sigma}^2, x_{vc}^{(l-1)}\right) \tag{13}$$

for which we like to ensure that the mutual information increases at every iteration and so density evolution is guaranteed to predict the decoding behavior because $x_{vc}^{(l)} > x_{vc}^{(l-1)}$.

2.3 Design by Linear Programming

The linear programming optimization algorithm is adopted to find suitable variable node degree distributions in the LDPC code design. [6, 10, 11] use it in for unequal error protection modulation and key reconciliation for channels susceptible to eavesdropping. The approaches used in the mentioned works are derived from topics beyond the scope of this thesis and since this thesis does not explore these areas in any detail, this summary is barely based on these works. The optimization method continues from (13) and considers $\rho(x)$ to be fixed so as to make the function F linear in $\boldsymbol{\lambda}$. The optimization of $\lambda(x)$ is done, with knowledge of $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}^2$, through a rate maximization criterion.

The rate R, given by

$$R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx} = 1 - \frac{\sum_{j\geq 2} \rho_j/j}{\sum_{i\geq 2} \lambda_i/i}, \tag{14}$$

is maximized subject to constraints that $x_{vc}^{(l)} > x_{vc}^{(l-1)}$, $\sum_j \rho_j = 1$, and $\lambda(1) = 1$ as in [8].

It should also be noted that when designing irregular LDPC codes, it is desirable for the check nodes to have lower degrees because the probability that the check function is fulfilled decreases as the degree gets bigger, whereas for variable nodes it is desirable to have it mixed up but certainly have some high degree nodes because this means that more extrinsic messages are delivered along the edges to the variable nodes and they then converge faster, i.e., only after a few iterations. Once the desired degree distribution is obtained, we can use one of several algorithms to construct the parity-check matrix. Most of these construction algorithms are mentioned on pages 228-232 in [7].

3 Neural Networks

Largely inspired by biological neurons, Artificial Neural Networks (ANNs) are essentially computational models that have particular properties that allow them to adapt, learn, and organize data. There are two main kinds of neural networks, namely feed-forward and recurrent networks. The main difference between the networks is easily whether or not the networks use feedback paths. Feed-forward networks process the received signals in input nodes and pass the computation results to the output or the input of the next layer. In this architecture, the processed signals in a layer are not fed back to that layer in any form. Therefore, more complex functions can only be implemented through adding layers to the network. Although, networks without feedback are widely used, feedback has been found to have a profound effect on the overall performance of a network and brings the network richer dynamics. This way, the network is able to implement complex functions with fewer layers than feedforward networks. For both types of networks, the signals are received by input source nodes and are sent out all the way through to the output neurons.

3.1 Multi-layer Perceptron

The pioneering work of McCulloch and Pitts [12] describes the threshold function used for basic logic computations. This is widely regarded as the first major contribution to neural network theory and it is on this that we base our neuron model as well as define typical architectures for neural networks. The artificial neuron has a weight vector **w** = $(w_{j1}, w_{j2}, ..., w_{jm})$ and a threshold or bias b_j. The weights are analogous to the strength of the dendritic connections in biological neurons. The bias is considered the value that must be surpassed by the inputs before an artificial neuron will become active, i.e., be greater than zero. However, it is usually written as an additive term. The activation of a neuron is the sum of the inner product of the weight vector with an input vector **x** = $(x_1, x_2, ..., x_m)$ and the bias. Figure 3 below shows a clear representation of a model of the neuron which includes an activation function[5] for the output. In order to satisfy the requirement for non-linearity, a commonly used $\tanh$-sigmoid function is used.

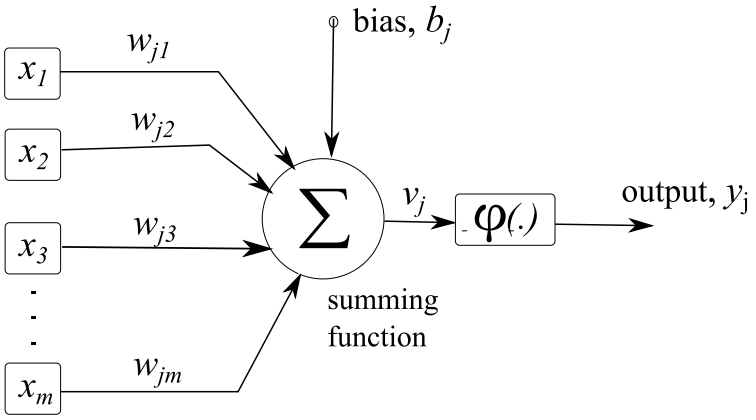

Figure 3: A simple artificial neuron model

[5]An activation function adds smooth non-linearity to the output of each neuron. Denoted by $\varphi(\cdot)$

We use this neuron as a building block of neural networks. In 1958, Frank Rosenblatt then conceived the perceptron a.k.a. single layered feedforward network which offers only one trainable weight layer. With only one input layer and one output layer, the perceptron was used primarily for linear classification. The perceptron is therefore perhaps the simplest form of an ANN used in most neural network literature. Conventionally though, neurons are organized in layers so as to increase the computational power of the network models. This then leads us to the multi-layered perceptron (feedforward network) that has at least one hidden layer of neurons between the input and output layers. The multi-layer perceptron uses the so-called back-propagation algorithm described below to train the network.

Back-propagation algorithm

The back-propagation algorithm is mentioned here as the standard and most important method for training neurons in feed-forward neural networks. It is particularly similar to the Least Mean Squares algorithm and this could perhaps give some insight into how it works, since feed-forward networks are much less complex than their recurrent counterparts. The algorithm uses the summed squared error as its metric such that the aim is to minimize it by changing the synaptic weights of the given network.

The model is largely based on the one used in [13]. Given an input vector pattern $\mathbf{u}(n)$ and an output vector pattern $\mathbf{y}(n)$ at iteration n, we shall use the notations below to describe components used in the algorithm:

- $u_i(n)$ is the i-th element of the input vector.
- $d_j(n)$ is the expected output of the neuron j.
- $y_k(n)$ is the k-th element of the output vector. It is the output function signal of neuron k.
- $w_{ij}(n)$ is the synaptic weight connecting the output of neuron j to the input of neuron i. The correction applied to this parameter is $\Delta w_{ij}(n)$. We also use b_j to represent the bias which is the effect of a synapse weight of w_{j0} connected to a fixed input of 1.
- $v_j(n)$ is the intermediate signal that is a weighted sum of all inputs and bias of neuron j which is fed into the activation function.
- m_l is the number of nodes in layer l. So the number of nodes in the first hidden layer would be m_1.
- r is the learning rate of the algorithm. [13] shows that the smaller the value of r is, the less we need to make changes in the weights of the network per iteration and that a very large r, even though good for fast learning, may not be ideal because it can make the network unstable.

In the feedforward case where **u** is the input, **y** is the output, and **x** is the vector of hidden units' values, "messages"[6] are passed forward (unaltered weights) as function signals and error signals are passed backward (with the local gradient recursively so as to change the synaptic connection weights of the network.

[6]Messages here are our input training data $\mathbf{u}(n)$

We activate hidden units using

$$x_i^{p+1}(n) = \varphi\left(\sum_j w_{ij}^p x_j^p(n)\right) \ , \tag{15}$$

where p is the layer such that the input for the first layer of units has the components of $\mathbf{u}(n)$ as the input.

The $v_j(n)$ at the input of the activation functions and the output $y_j(n)$ of the activation functions are mathematically expressed below:

$$v_j(n) \ = \ \sum_{i=0}^{m} w_{ji}(n) y_i(n) \ \text{ and any output } y_j(n) \ = \ \varphi_j(v_j(n)) \tag{16}$$

The error signal $e_j(n) = d_j(n) - y_j(n)$ is computed at the output layer at iteration n. Given the error signals, we can use $\mathscr{E}(n)$ as the instantaneous sum of error squares at time n, i.e., $\mathscr{E}(n) \ = \ \sum_n e_j^2(n)$. This can also be referred to as the error energy whose average over all N values of n is $\bar{\mathscr{E}} \ = \ \frac{1}{N}\mathscr{E}(n)$.

Just as in a Least Mean Squares (LMS) adaptation, we apply a change in weights $\Delta w_{ji}(n)$ proportional to sensitivity factor $\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)}$ regarding the weights $w_{ji}(n)$. After getting the error signals, we pass these error signals backward from the output side to alter the weights of each neuron and compute the local gradient for each neuron.

The dependence of $\Delta w_{ji}(n)$ on $\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)}$ is illustrated with use of the learning rate (step size) r such that $\Delta w_{ji}(n) = -r \cdot \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)}$. Employing the chain rule, we can express the sensitivity factor for different nodes as follows:

1. Output nodes:

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} \ = \ \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{17}$$

   Quite clearly, simple values can be attached to the four partial derivatives involved in that, the first one is simply $e_j(n)$, the second is -1, the third is the value of $\varphi'(v_j(n))$, and the last one is $y_i(n)$. These are all known values after the forward passing is done and the errors are known.

   The update on the weights is made such that $w_{ji}(n+1) \ = \ w_{ji}(n) \ + \ \Delta w_{ji}(n)$

2. Hidden nodes:
   The error signals for hidden neurons is not as clear as that of output ones. Here we have to recursively find the error signal of a single neuron in terms of error signals of all other neurons to which the one in question is directly connected. In this case, we would still use the chain rule as in Eq. (17), but the first three partial derivatives are treated differently so that we build on again from $\frac{\partial \mathscr{E}(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$. The chain rule is applied again, this time ignoring the error signal of the neuron as shown below:

$$\frac{\partial \mathscr{E}(n)}{\partial v_j(n)} \ = \ \frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \cdot \varphi'(v_j(n))$$

Since $\mathscr{E}(n)$ depends on the errors of the output neurons denoted with index k,

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi'(v_j(n)) w_{kj}(n)$$

so $\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = -\sum_k e_k(n) \varphi'(v_j(n)) w_{kj}(n) \cdot \varphi'(v_j(n)) \cdot y_i(n)$ and the weight update is made just as for the output nodes.

A stopping criterion has to be defined by means of a threshold value for convergence, setting a specific number of iterations or obtaining diminishing errors after which the training is stopped.

3.2 Hopfield Networks

Inspired by the spin system in physics, John Hopfield came out with his model in 1982, the Hopfield network is a multi-loop feedback network in which the output of each neuron is fed back with delay to each of the other neurons in the network. It is some type of recurrent neural network for which a Lyapunov function[7] governs the dynamics of the network. For such a network, we assume a random initial state such that the state of the network evolves to some final state that is ideally a local minimum of the governing Lyapunov function. In our model we shall describe below, we use the binary neurons (based on the McCulloch-Pitts model in [12]) which are updated one at a time. A simple diagram below shows an example of a weighted undirected graph to represent a Hopfield network.

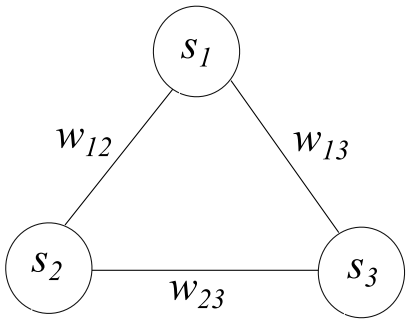

Figure 4: A Hopfield network with 3 neurons

We shall define some notations to be used for this section of the review:

- w_{ij} denotes synaptic weight connection between neuron j and neuron i. These are the components of the weight matrix **W**
- s_i denotes the state/activity of the neuron i.

[7] A Lyapunov function is some scalar function used to prove stability of the equilibrium point of a nonlinear system.

We use single shot learning[8] such that we compute the weights in the matrix **W** with the learning rule for a set of patterns $\mathbf{s}^p$ being

$$w_{ij} = \sum_{p=1}^{P} s_i^p s_j^p \ . \tag{18}$$

The activation function used here is the common binary threshold function such that everything above 0 is $+1$, else -1 (this means that if the states are equal then we increase the weight by 1, otherwise we reduce it by 1).

The Hopfield net can either be used with discrete time or continuous time. For the standard discrete time model, we assume that there is no self-feedback, i.e., $w_{kk} = 0 \ \forall k$ neurons and that the weights are symmetric such that $w_{ij} = w_{ji}$. Because of the asynchronous treatment[9] of the network units and the assumptions made for this model, convergence is guaranteed for a network with the energy function (with states **s**, bias **b**, and weights **w**) shown below:

$$E = -\sum_i s_i b_i - \frac{1}{2}\sum_j \sum_{i,i\neq j} w_{ij} s_i s_j \tag{19}$$

where the $\frac{1}{2}$ appears because each pair is counted twice in the double summation.

We choose a neuron j at random and update the neuron using the update rule below. A chosen pattern is entered in the network by either setting all or part of the available nodes to a particular value. Iterations are then made with the asynchronous updating till no changes occur and we have the lowest energy E:

$$s_j(n+1) = \varphi\Big(\sum_i w_{ji} s_i(n) - b_i\Big) \tag{20}$$

The spin model in statistical physics can also be seen as a more detailed instance of a Hopfield network. This model has been used for idealized magnetic systems and defined with magnetically coupled spins ± 1 for the states, magnetic coupling J for the weights and applied field H (we might use h_n for a non-constant field) for the bias. This basic introduction of this complex subject is based on the publication by Advani *et al.* in [14], Chapter 31 in MacKay's book [15], and Chapter 11 of Haykin [13]

As a parallel to the standard model, we can define the energy function of given state **s** as:

$$E(\mathbf{s} \mid \mathbf{J}, H) = -\frac{1}{2}\sum_{i,j} J_{ij} s_i s_j - \sum_j H s_j$$

with **J** simply replacing **W**, H replacing the bias b_i, and the spin states having a similar function as those in the standard model insofar as to represent spin degrees of freedom which can only takes values ± 1. **J** (chosen randomly to have i.i.d. zero mean Gaussian components) is the magnetic coupling between any two spins and H here denotes the applied field which is assumed to be constant.

[8] Each training pattern is used exactly once.

[9] Each neuron is probed in a random manner, but with equal probability.

Allowing $P(\mathbf{s})$ to be the probability of state[10] $\mathbf{s}$, we define, for a body at equilibrium temperature T, this probability to be

$$P(\mathbf{s}) = \frac{1}{Z(\beta, \mathbf{J})} e^{-\beta E(\mathbf{s}|\mathbf{J}, H)} \tag{21}$$

where

$$Z(\beta, \mathbf{J}) = \sum_{\mathbf{s}} e^{-\beta E(\mathbf{s}|\mathbf{J}, H)} \tag{22}$$

is called the partition function and $\beta = \frac{1}{k_B T}$[11] is an inverse temperature [14].

Many of the details in the corresponding papers are left out, because they are way beyond the scope of this thesis. The intricate details concerning the heat capacities, replica and cavity methods for the interested reader are given in [14] and [15].

3.3 Echo-State Networks

Unlike other recurrent neural networks in which no single property defines their structure, Echo-State networks are defined solely by the property of having or not having an echo state prior to their training. A network found to have echo states for one set of training data may not have them for another and for this reason, the training data is sampled from a specifically chosen set of data or interval. Even though this property is mentioned with high acclaim, we can only provide a solid proof for the non-existence of echo states in a network.

In echo-state networks, only the weights of the output units are trained. Input to hidden unit connections are of fixed weight, hidden to hidden unit connections are randomly selected. We select the random hidden to hidden unit weights and the scale of the input to hidden unit connection weights in such a way that the length of the activity vector does not really change with iteration which then allows the input to echo through the network.

Given a discrete time neural network with M input units, N hidden units, and P output units, we give the notations to describe components in our echo-state network approach:

- $\mathbf{u}(n)$ is the input vector of size M at iteration time n. We use it together with target output $\mathbf{d}(n)$ so as to have training data.
- $\mathbf{x}(n)$ is the activation state vector of size N for the hidden units. It is seen in [16, 17] as a function of the previous inputs which brings about the use of "echo" to describe this state.
- $\mathbf{y}(n)$ is the output vector of size P at iteration time n.
- $\mathbf{W}^{in}$ is the weight matrix for the input weights [$N \times M$].
- $\mathbf{W}$ is the weight matrix for the hidden units' weights [$N \times N$].
- $\mathbf{W}^{out}$ is the changeable weight matrix connecting to the output units [$P \times (M + N + P)$].
- $\mathbf{W}^{back}$ is the weight matrix connecting output units back to the hidden units [$N \times P$]

[10] This also counts as the asymptotic distribution of states under Boltzmann distribution.

[11] k_B is Boltzmann's constant, 1.3806488 $\times 10^{-23}$ $m^2 kg\ s^{-2} K^{-1}$

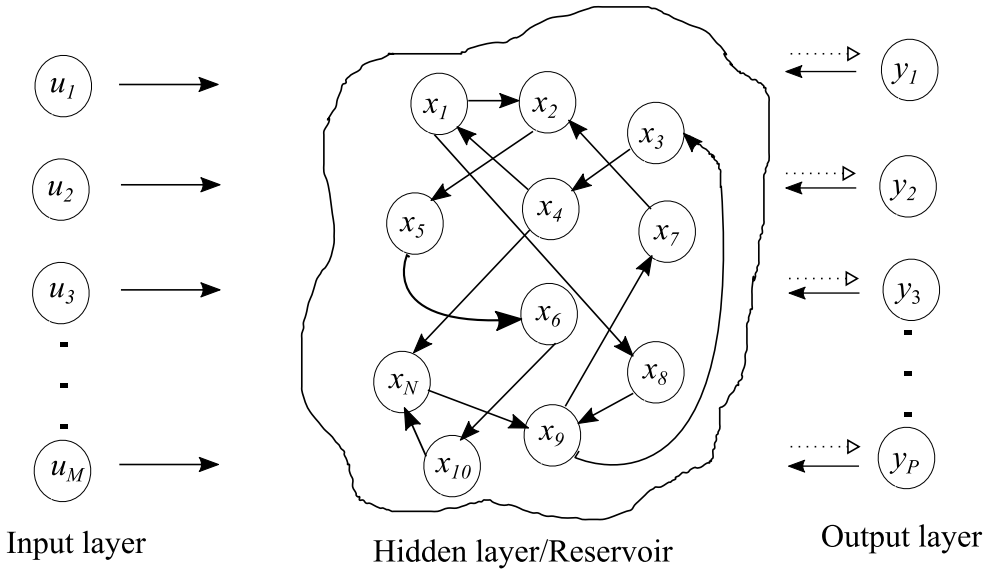

Figure 5: Simple echo-state network architecture. Dashed lines show possible connections that can also be omitted.

The basic idea is to have a so-called reservoir which one uses as a source of neuronal dynamics from which the desired output can be modeled. The intuition behind echo-state networks is that for an untrained network with training data $(\mathbf{u}(n), \mathbf{d}(n))$ from $\mathcal{U}$ and $\mathcal{D}$, respectively, the network has echo states with respect to the sets $\mathcal{U}$ and $\mathcal{D}$ if for every left-infinite training sequence $(\mathbf{u}(n), \mathbf{d}(n-1))$ and for all state sequences $\mathbf{x}(n), \mathbf{x}'(n)$ compatible with the training sequence in that $\mathbf{x}$(n+1) and $\mathbf{x}'$(n+1) fulfill (25), then it holds that $\mathbf{x}(n) = \mathbf{x}'(n)$ for all $n \leq 0$ as in [16]. This can be seen as such that for a long enough input sequence, the $\mathbf{x}(n)$ should not depend on the initial conditions applied before the input. $\mathbf{x}(n)$ is, as defined, an echo and is therefore a function of previously presented input $\mathbf{u}(n)$, $\mathbf{u}(n-1), \ldots$

We randomly generate a matrix $\mathbf{W}_0$ and find its $|\lambda_{max}|$, normalize it such that $\mathbf{W}_1 = \frac{1}{|\lambda_{max}|}\mathbf{W}_0$, and then we come up with a suitable $\mathbf{W}$ scaled as $\alpha\mathbf{W}_1$ with the spectral radius of $\mathbf{W}$ being α. Choosing α is therefore carefully done since defines the convergence speed.

Below is a conjecture taken directly from Jäger's publication in [16]. It further defines how we can be sure that we have an echo-state network to work with.

Let δ and ε be two very small positive numbers. There exists a network size N, such that when an N sized dynamical reservoir is randomly constructed by

1. randomly generating a sparse weight matrix $\mathbf{W}_0$ with spectral radius $|\lambda_{max}|$ by sampling the weights from a uniform distribution over [-1,1]
2. normalizing $\mathbf{W}_0$ to a matrix $\mathbf{W}_1$ with unit spectral radius by putting $\mathbf{W}_1 = \frac{1}{|\lambda_{max}|}\mathbf{W}_0$
3. scaling $\mathbf{W}_1$ to $\mathbf{W}_2$ = (1-δ) $\mathbf{W}_1$, whereby $\mathbf{W}_2$ obtains a spectral radius of (1-δ)

then the network ($\mathbf{W}_{in}$, $\mathbf{W}$, $\mathbf{W}_{back}$) is an echo-state network with a very high probability $1-\varepsilon$.

We compute output weights w_i^{out} using the fact that $\mathbf{d}(n)$ should be approximated by

$$\mathbf{y}(n) \;=\; f^{out}\left(\sum_{i=1}^{N} w_i^{out} x_i(n)\right) \;, \tag{23}$$

which is a simple regression model of $\mathbf{d}(n)$ on hidden states $x_i(n)$. We use the (Normalized) Mean Squared Error[12] as the metric to train the network, The training on the network is carefully elaborated in [16]. It entails having set up the untrained network with corresponding matrices $\mathbf{W}$, $\mathbf{W}^{in}$, and $\mathbf{W}^{back}$. We have already described, in some detail, how to generate the $\mathbf{W}$ matrix. How the other two matrices are chosen is trivial as long as $\mathbf{W}$ is carefully modeled. $\mathbf{W}^{in}$ can be generated from the same distribution as $\mathbf{W}$. The network is initialized using $\mathbf{x}(0) = \mathbf{d}(0) = \mathbf{0}$ and updated with an input training sequence for $n = 0, \ldots, T$ such that $\mathbf{x}(n+1) = \varphi\left(\mathbf{W}^{in}(\mathbf{u}(n+1)) + \mathbf{W}(n)\right)$. An initial transient is chosen such that after some time n_{min}, the network state is determined by the preceding input history. Training the output matrix $\mathbf{W}^{out}$ involves use of the target output data $\mathbf{d}(n)$. We place values of $\mathbf{x}(n)$ produced from the respective $\mathbf{u}(n)$ into a matrix $\mathbf{X}$, where each column is a sequence $\mathbf{x}(n)$ for all n. The same is done to create a matrix $\mathbf{D}$ with components from all sequences $\mathbf{d}(n)$[13]. Expressing Eq. (23) in matrix form[14], we obtain

$$\mathbf{W}^{out} \;=\; \mathbf{D}\mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T\right)^{-1} \tag{24}$$

as the linear regression model to find the matrix $\mathbf{W}^{out}$.

For the trained network, activation of the hidden units is given by

$$\mathbf{x}(n+1) \;=\; \varphi\left(\mathbf{W}^{in}\mathbf{u}(n+1) \;+\; \mathbf{W}\mathbf{x}(n) \;+\; \mathbf{W}^{back}\mathbf{y}(n)\right) \tag{25}$$

The output is determined according to

$$\mathbf{y}(n+1) \;=\; f^{out}\left(\mathbf{W}^{out}\left(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n)\right)\right) \tag{26}$$

[12] MSE = $\frac{1}{P}\sum_i^P \left(\mathbf{d}(n) - \mathbf{y}(n)\right)^2$ is minimized.

[13] Matrices are such $\mathbf{X} \in \mathbb{R}^{N\times T}$ and $\mathbf{D} \in \mathbb{R}^{P\times T}$

[14] We now use the desired output $\mathbf{d}(n)$ for training and for purposes of clarity, ignore f^{out}, i.e., consider it to be linear.

4 Relations between LDPC Codes and Neural Networks

Continuing Henkel's work in [18], we can easily make the connection between neural networks and LDPC-like codes in terms of their use of the sigmoid function as an activation function in the back-propagation algorithm and for the other network models mentioned in Section 3.

4.1 The neuron in the Tanner graph under message passing

We have the popular sigmoid activation in neurons which compares to the use of the $\tanh$-function in updates of the check nodes under message passing. The standard model of a neuron in the back-propagation algorithm has a hyperbolic tangent activation function and after drawing some mathematical similarities, we can easily see a neuron model inside LDPC decoding. The equations below rigorously illustrate this close relationship.

From the equations (6) and (7), we now define $v_k^{\mathcal{T}} := \tanh(v_k/2)$.

With this, the variable-node equation becomes

$$v_m{}^{\mathcal{T}^{(l)}} = \tanh\left(\left[u_0 + \sum_{k=1,k\neq m}^{i} u_k^{(l-1)}\right]/2\right) \quad \forall m = 1...i \tag{27}$$

This also leads to a modified check-node equation

$$u_k{}^{\mathcal{T}^{(l)}} = \prod_{m=1,m\neq k}^{j} v_m{}^{\mathcal{T}^{(l)}} \quad \text{or} \quad \ln u_k{}^{\mathcal{T}^{(l)}} = \sum_{m=1,m\neq k}^{j} \ln v_m{}^{\mathcal{T}^{(l)}} \tag{28}$$

Another simplification option solely looking into the check-node equation in log domain:

$$\ln\tanh\frac{u_k^{(l)}}{2} = \sum_{m=1,m\neq k}^{j} \ln\tanh\frac{v_m^{(l)}}{2}, \forall k = 1...j \tag{29}$$

v_m are LLRs, i.e., we can rewrite

$$\ln\tanh\frac{v_m^{(l)}}{2} = \ln\frac{\frac{v(0)}{v(1)} - 1}{\frac{v(0)}{v(1)} + 1} = \ln\frac{v(0) - v(1)}{v(0) + v(1)} \tag{30}$$

Furthermore,

$$\begin{pmatrix} V(0) \\ V(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} v(0) \\ v(1) \end{pmatrix} \tag{31}$$

using a 2×2 DFT relation.

Further defining $\mathcal{V} = \ln\frac{V(1)}{V(0)}$ and $\mathcal{U} = \ln\frac{U(1)}{U(0)}$, yields for the check node side

$$\mathcal{U}_k^{(l)} = \sum_{m=1,m\neq k}^{j} \mathcal{V}_m^{(l)}\,, \quad \forall k = 1...j \tag{32}$$

Also in comparison to the neuronal model in Fig. 3, we show a figurative flow of LDPC sum-product decoding in Fig. 6 below, which when carefully disassembled, would contain behaviorisms of the described neuron model. On the variable node side, one can take the channel information to be some sort of bias and the extrinsic messages to be the incoming vector **x** of the neuron. At this point after summation[15], the variable to check node messages can be seen as v_j, the output of the summing function. The output y_j is then represented by the output of the $\tanh$ function as in the neuron model.

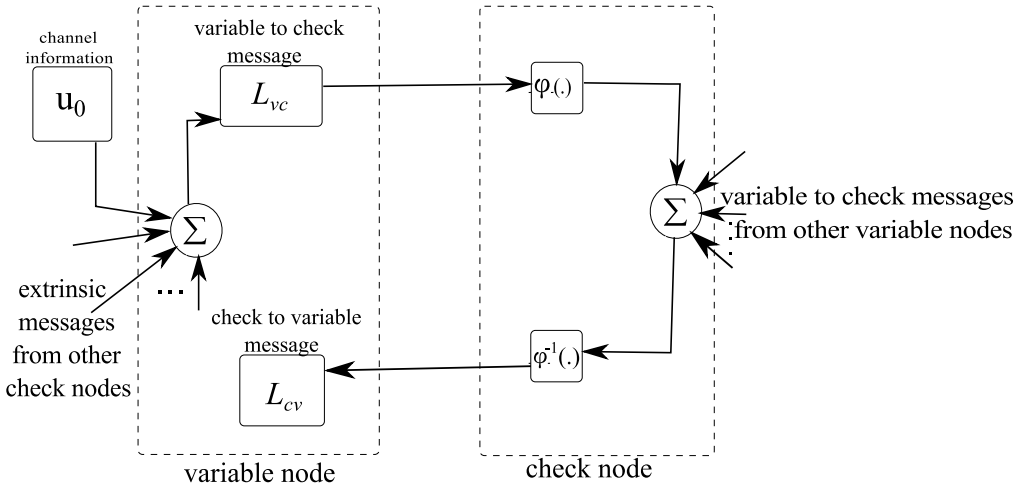

Figure 6: Sum-product algorithm update rules in a neuron sense

On the check node side, the connection to LDPC codes is rather not that simple. We have seen that the check node sums up messages in the DFT domain. There is a non-linear non-neural network like function used, i.e., the $\tanh^{-1}$ function.

4.2 Comparisons between the perceptron and LDPC graphs

In structure, single layered perceptron networks bear a certain similarity to factor graphs used in decoding LDPC codes with no cycles. However, the perceptron does not have the same bidirectional flow of information along the edges of the graph like for LDPC codes. Factor graphs representing LDPC codes are mainly known to be directed acyclic graphical models and as far graphical representation goes, the neural networks that have so far been introduced in this thesis can be viewed as directed graphs. Unlike over edges of an LDPC graph, the signal-flow in neural networks is hugely influenced by the weights of the interconnecting edges. Although LDPC message-passing decoding has the same sigmoid-type function which we have used as a basis to check whether there are even more similarities between the decoding algorithm and standard networks, there are some fundamental differences between the LDPC message-passing decoding and the perceptron, which perhaps apply to all neural networks in general:

- Multi layer perceptrons have at least one layer of hidden units which is quite inapplicable to LDPC codes. This, in addition to having to train networks, is very dissimilar to LDPC edge processing.

[15]This summation is in the neuron model as well

- Unlike variable and check nodes, the neuron model we use does not pass different messages over different edges. It rather transmits the same message over all its outgoing edge(s).
- A neuron cannot have access to the individual messages received over its links from its neighbors. The only quantity that is available to a neuron during the decision-making process is a weighted sum over the received messages.
- The neuron model we use does not perform over Galois fields as in LDPC codes which usually operate in GF(2).

The individual neurons that we use are also limited in such a way that they cannot really communicate probabilities since they mainly work with thresholds.

4.3 Message passing in the Hopfield network

All particles in the magnetic spin system that we described communicate with each other so as to reach an energetically favorable state. As an analogy to LDPC codes, this is essentially like finding the correct codeword and this *communication*, although not shown by fully connected graphs, is done through the constant passing of messages which in the end implies indirect dependence on other nodes. In [14], a message passing approach to analyzing statistical mechanics systems is introduced. Using message passing algorithms to understand neuronal activity for a given matrix J is investigated for magnetic systems. Although the system employs message passing as a tool to compute marginals, we can use this to get some insights on how to implement sum-product algorithm decoding of LDPC codes on these such networks.

Some of the criticism to Hopfield networks in their relation to decoding of LDPC codes on graphs are that they generally are fully connected networks and the neurons cannot be divided up into a set of variable and check nodes since they all serve the same function.

4.4 Structural comparisons between echo-state networks and LDPC graphs

Just as we do not consider the full complexity of real biological neurons when making artificial neural networks, in this section, we shall ignore or simply overlook some of the complexities in both topics in order to give some motivation for drawing these comparisons. Looking at it on the surface, sparse connectivity used in the hidden layer connections is most stand-out and yet most trivial aspect of echo-state networks that can immediately be connected to the given LDPC **H** matrices. If we liken variable nodes to the hidden units and the check nodes to the output layer units, I would suggest approaching the comparison in two different ways.

In the first approach, we re-model the structure of ESNs such that the hidden nodes are not connected through the matrix **W**, i.e., **W** = 0. We can then rigorously show a structural relationship between ESNs and LDPC decoding graphs. The unchanged output layer would represent the check nodes. A shortcoming of this, however, is that removal of the reservoir connections limits the functionality of the graph as an echo-state network.

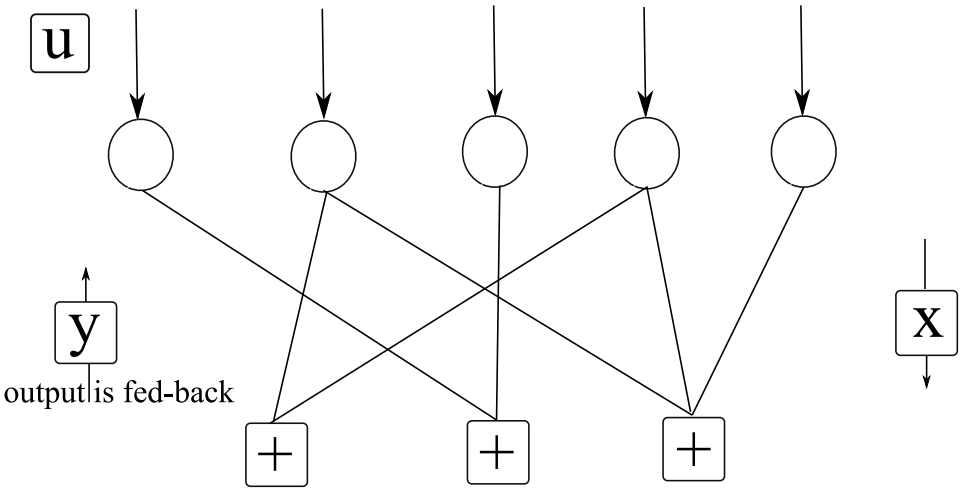

Figure 7: Echo-state network architecture for LDPCs in the first approach

Equations (25) and (26) become

$$\mathbf{x}(n+1) \;=\; \varphi\left(\mathbf{W}^{in}\mathbf{u}(n+1) \;+\; \mathbf{W}^{back}\mathbf{y}(n)\right) \tag{33}$$

and

$$\mathbf{y}(n+1) \;=\; f^{out}\left(\mathbf{W}^{out}\mathbf{x}(n+1)\right) \tag{34}$$

f^{out} in (26) is usually linear but in this case would be replaced by the $\tanh^{-1}$ as is on the check node side of LDPC codes. Here, matrices $\mathbf{W}^{out}$ and $\mathbf{W}^{back}$ are so carefully chosen so as to correctly map the messages to the outgoing vectors $\mathbf{x}(n)$ and $\mathbf{y}(n)$, respectively. **W** is totally ignored and $\mathbf{W}^{in}$ is a linear mapping (e.g., identity matrix multiplication) of input LLRs from $\mathbf{u}(n)$.

$\mathbf{x}(n+1)$ is the output of the variable nodes' layer.[16] $\mathbf{y}(n+1)$ is the output of the check nodes' layer and is initialized with $\mathbf{y}(0) = \mathbf{0}$. Figure 8 shows an example of message exchange from a check node j to a variable node i. The $\mathbf{x}_{\setminus i}(n+1)$ is the variable nodes' output vector excluding the information from variable node i.

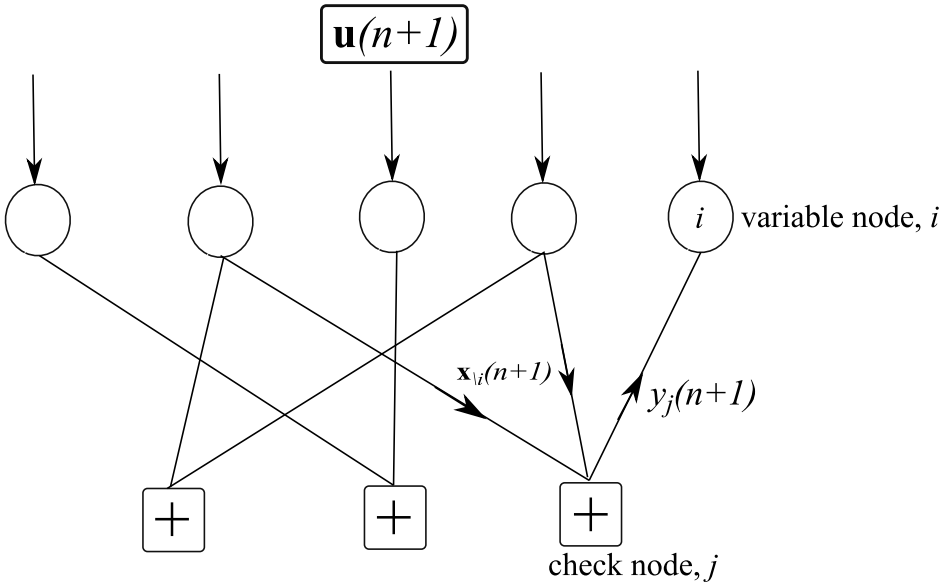

Figure 8: Message exchange from check to variable node in the first approach

The second approach incorporates ideas from Binary Markov sources in order to model the matrices involved in describing the network. In this case the variable nodes would ideally have connection amongst each other as in a Markov source model and the original echo-state network we described. The state of the node is the same as the value (either 0 or 1) we get while decoding. Figure 9 below and the few steps thereafter illustrate this approach.

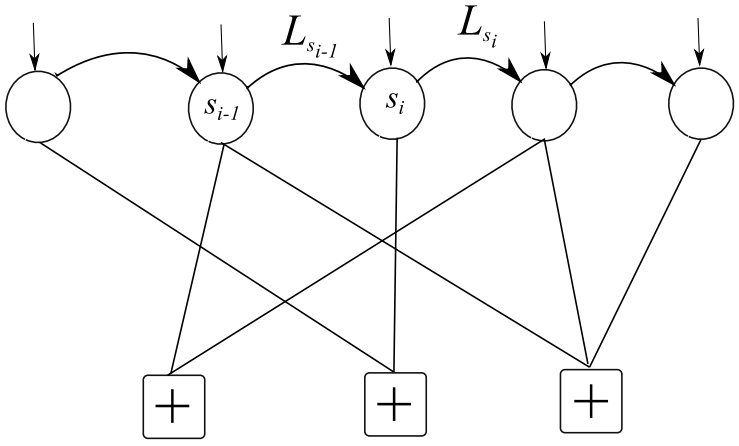

Figure 9: Echo-state network architecture in the second approach

[16] $\mathbf{x}(n)$ is not of interest when computing $\mathbf{x}(n+1)$ because $\mathbf{W}$ is 0.

Defining the LLR of node i as $L_{s_i} = \ln \frac{p(s_i=0)}{p(s_i=1)}$, we can find it using the formulations below to demonstrate the use of Markov states in LLR computation.

$$L_{s_i} = \ln \frac{p(s_i = 0 \mid s_{i-1} = 0)p(s_{i-1} = 0) + p(s_i = 0 \mid s_{i-1} = 1)p(s_{i-1} = 1)}{p(s_i = 1 \mid s_{i-1} = 0)p(s_{i-1} = 0) + p(s_i = 1 \mid s_{i-1} = 1)p(s_{i-1} = 1)} \tag{35a}$$

$$L_{s_i} = \ln \frac{p(s_i = 0 \mid s_{i-1} = 0)\frac{p(s_{i-1}=0)}{p(s_{i-1}=1)} + p(s_i = 0 \mid s_{i-1} = 1)}{p(s_i = 1 \mid s_{i-1} = 0)\frac{p(s_{i-1}=0)}{p(s_{i-1}=1)} + p(s_i = 1 \mid s_{i-1} = 1)} \tag{35b}$$

$$L_{s_i} = \ln \frac{p(s_i = 0 \mid s_{i-1} = 0)e^{L_{s_{i-1}}} + p(s_i = 0 \mid s_{i-1} = 1)}{p(s_i = 1 \mid s_{i-1} = 0)e^{L_{s_{i-1}}} + p(s_i = 1 \mid s_{i-1} = 1)} \tag{35c}$$

We can also a maximum a posteriori probability (MAP) decoding once we know the received word $\mathbf{y}$. We solve for $\hat{s}_i = \text{argmax}\, p(s_i \mid \mathbf{y})$ such that

$$\hat{s}_i = \underbrace{p(s_i)}_{\text{a-priori info.}} + \underbrace{p(y_i \mid s_i)}_{\text{intrinsic likelihood}} + \underbrace{p(\mathbf{y}_{\setminus i} \mid s_i)}_{\text{extrinsic information}} \tag{36}$$

The sum-product algorithm can efficiently compute the conditional density $p(s_i \mid \mathbf{y})$ through factorization.

5 Conclusions

In this thesis, we have proposed various simple yet insightful ways to look at sum-product decoding of LDPC codes implemented with neural networks. We have also presented other possible relations based on more than just the decoding algorithm. We have used message passing under conditions that are somewhat dissimilar to those in the decoding algorithm when implemented over Hopfield networks. These connections can easily be built upon so as to yield interesting results for neural networks. Ideas to make the MLP neural networks better could emerge since we may take the advantage of the probabilistic structure of the sum-product algorithm. Ideas about use of the known connections between mean squared error (IMMSE) and mutual information as well as optimization of degree distribution of codes can possibly be incorporated into neural network design. Since we know how to optimize LDPC codes, maybe we can use these tools to optimize neural networks and in so doing, improve the performance of the neural networks.

To conclude this thesis, I would like to say that although the work is very theoretical, it serves as a detailed literature review on two substantially vast topics and I hope that the interested reader can use this as motivation for a venture into understanding neural networks better in relation to graph-based coding. It is something that I will personally continue pursuing.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[2] R. G. Gallager, "Low-Density Parity-Check Codes," 1963.

[3] D. Blackmer, F. Schwaner, and A. Abedi, "Non iterative decoding of Low Density Parity Check codes using artificial neural networks," *ICWN'11*, 2011.

[4] A. Karami, M. Attari, and H. Tavakoli, "Multi Layer Perceptron Neural Networks Decoder for LDPC Codes," pp. 1–4, Sept 2009.

[5] T. Richardson and R. Urbanke, *Modern Coding Theory*. New York, NY, USA: Cambridge University Press, 2008.

[6] A. Filip, "Physical-layer security," Master Thesis, Jacobs University, 2013. [Online]. Available: http://trsys.faculty.jacobs-university.de/wp-content//uploads/2014/03/MSc_thesis_Alexandra_Filip.pdf

[7] W. Henkel, *Channel Coding*. Bremen, Germany: Jacobs University, Spring 2014.

[8] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657–670, Feb 2001.

[9] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb 2001.

[10] N. von Deetzen, "Modern coding schemes for unequal error protection," Ph.D. dissertation, Jacobs University Bremen, Shaker Verlag, 2009. [Online]. Available: http://trsys.faculty.jacobs-university.de/wp-content//uploads/2014/02/PhD_thesis_Neele_von_Deetzen.pdf

[11] N. Islam, O. Graur, W. Henkel, and A. Filip, "LDPC code design aspects for physical-layer key reconciliation." San Diego, California: IEEE International Global Communications Conference, 2015.

[12] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, Dec. 1943. [Online]. Available: http://dx.doi.org/10.1007/bf02478259

[13] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[14] M. Advani, S. Lahiri, and S. Ganguli, "Statistical mechanics of complex neural systems and high dimensional data," *arXiv preprint arXiv:1301.7115*, 2013.

[15] D. J. Mackay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. [Online]. Available: http://www.inference.phy.cam.ac.uk/itprnn/book.pdf

[16] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach," *Fraunhofer Institute AIS, GMD Report*, vol. 159, Dec. 2013. [Online]. Available: http://minds.jacobs-university.de/sites/default/files/uploads/papers/ESNTutorialRev.pdf

[17] ——, “Echo state network,” *Scholarpedia*, vol. 2, no. 9, p. 2330, 2007.

[18] *Compressive Sensing, LDPC Codes, and Neural Networks*. University of Bremen: Compressed Sensing Workshop, 2015. [Online]. Available: http://www.math.uni-bremen.de/zetem/cms/media.php/247/ZeTeM-News_CSW.pdf

[19] M. Bodén, “A guide to recurrent neural networks and backpropagation,” in *In the Dallas project, SICS technical report T2002:03, SICS*, 2002.

[20] T. Ott and R. Stoop, “The neurodynamics of belief propagation on binary Markov Random Fields,” Cambridge, MA, pp. 1057–1064, 2006. [Online]. Available: http://books.nips.cc/papers/files/nips19/NIPS2006_0460.pdf

[21] Z. Mei and L. Wu, “LDPC Codes for Binary Markov sources.”