Expectation Backpropagation: Parameter-Free Training of Multilayer Neural Networks with Continuous or Discrete Weights (with appendix)

Daniel Soudry[1], **Itay Hubara**[2], **Ron Meir**[2]
(1) Department of Statistics, Columbia University
(2) Department of Electrical Engineering, Technion, Israel Institute of Technology
`daniel.soudry@gmail.com,itayhubara@gmail.com,rmeir@ee.technion.ac.il`

Abstract

Multilayer Neural Networks (MNNs) are commonly trained using gradient descent-based methods, such as BackPropagation (BP). Inference in probabilistic graphical models is often done using variational Bayes methods, such as Expectation Propagation (EP). We show how an EP based approach can also be used to train deterministic MNNs. Specifically, we approximate the posterior of the weights given the data using a "mean-field" factorized distribution, in an online setting. Using online EP and the central limit theorem we find an analytical approximation to the Bayes update of this posterior, as well as the resulting Bayes estimates of the weights and outputs.

Despite a different origin, the resulting algorithm, Expectation BackPropagation (EBP), is very similar to BP in form and efficiency. However, it has several additional advantages: (1) Training is parameter-free, given initial conditions (prior) and the MNN architecture. This is useful for large-scale problems, where parameter tuning is a major challenge. (2) The weights can be restricted to have discrete values. This is especially useful for implementing trained MNNs in precision limited hardware chips, thus improving their speed and energy efficiency by several orders of magnitude.

We test the EBP algorithm numerically in eight binary text classification tasks. In all tasks, EBP outperforms: (1) standard BP with the optimal constant learning rate (2) previously reported state of the art. Interestingly, EBP-trained MNNs with binary weights usually perform better than MNNs with continuous (real) weights - if we average the MNN output using the inferred posterior.

1 Introduction

Recently, Multilayer[1] Neural Networks (MNNs) with deep architecture have achieved state-of-the-art performance in various supervised learning tasks [17, 20, 12]. Such networks are often massive and require large computational and energetic resources. A dense, fast and energetically efficient hardware implementation of trained MNNs could be built if the weights were restricted to discrete values. For example, with binary weights, the chip in [19] can perform 10^{12} operations per second with $1\mathrm{mW}$ power efficiency. Such performances will enable the integration of massive MNNs into small and low-power electronic devices.

Traditionally, MNNs are trained by minimizing some error function using BackPropagation (BP) or related gradient descent methods [21]. However, such an approach cannot be directly applied if the weights are restricted to binary values. Moreover, crude discretization of the weights is usually quite

---

[1] *i.e.*, having more than a single layer of adjustable weights.

destructive [26]. Other methods have been suggested in the 90's (*e.g.*, [29, 5, 24]), but it is not clear whether these approaches are scalable.

The most efficient methods developed for training Single-layer[2] Neural Networks (SNN) with binary weights use approximate Bayesian inference, either implicitly [8, 3] or explicitly [30, 28]. In theory, given a prior, the Bayes estimate of the weights can be found from their posterior given the data. However, storing or updating the full posterior is usually intractable. To circumvent this problem, these previous works used a factorized "mean-field" form the posterior of the weights given the data.

As explained in [28], this was done using a special case of the widely applicable Expectation Propagation (EP) algorithm [25] - with an additional approximation that the fan-in of all neurons is large, so their inputs are approximately Gaussian. Thus, given an error function, one can analytically obtain the Bayes estimate of the weights or the outputs, using the factorized approximation of the posterior. However, to the best of our knowledge, it is still unknown whether such an approach could be generalized to MNNs, which are more relevant for practical applications.

In this work we derive such generalization, using similar approximations (section 3). The end result is the Expectation BackPropagation (EBP, section 4) algorithm for online training of MNNs where the weight values can be either continuous (*i.e.*, real numbers) or discrete (*e.g.*, ± 1 binary). Notably, the training is parameter-free (with no learning rate), and insensitive to the magnitude of the input. This algorithm is very similar to BP. Like BP, it is very efficient in each update, having a linear computational complexity in the number of weights.

We test the EBP algorithm (section 5) on various supervised learning tasks: eight high dimensional tasks of classifying text into one of two semantic classes, and one low dimensional medical discrimination task. Using MNNs with two or three weight layers, EBP outperforms both standard BP, as well as the previously reported state of the art for these tasks [11]. Interestingly, the best performance of EBP is usually achieved using the Bayes estimate of the output of MNNs with *binary* weights. This estimate can be calculated analytically, or by averaging the output of several such MNNs, with weights sampled from the inferred posterior.

2 Preliminaries

General Notation A non-capital boldfaced letter $\mathbf{x}$ denotes a column vector with components x_i, a boldfaced capital letter $\mathbf{X}$ denotes a matrix with components X_{ij}. Also, if indexed, the components of $\mathbf{x}_l$ are denoted $x_{i,l}$ and those of $\mathbf{X}_l$ are denoted $X_{ij,l}$. We denote by $P(x)$ the probability distribution (in the discrete case) or density (in the continuous case) of a random variable X, $P(x|y) = P(x,y)/P(y)$,$\langle x \rangle = \int xP(x)\,dx$, $\langle x|y \rangle = \int xP(x|y)\,dx$, $\mathrm{Cov}(x,y) = \langle xy \rangle - \langle x \rangle \langle y \rangle$ and $\mathrm{Var}(x) = \mathrm{Cov}(x,x)$. Integration is exchanged with summation in the discrete case. For any condition $\mathcal{A}$, we make use of $\mathcal{I}\{A\}$, the indicator function (*i.e.*, $\mathcal{I}\{A\} = 1$ if A holds, and zero otherwise), and $\delta_{ij} = \mathcal{I}\{i = j\}$, Kronecker's delta function. If $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ then it is Gaussian with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and we denote its density by $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Furthermore, we use the cumulative distribution function $\Phi(x) = \int_{-\infty}^{x} \mathcal{N}(u|0,1)\,du$.

Model We consider a general feedforward Multilayer Neural Network (MNN) with connections between adjacent layers (Fig. 2.1). For analytical simplicity, we focus here on deterministic binary (± 1) neurons. However, the framework can be straightforwardly extended to other types of neurons (deterministic or stochastic). The MNN has L layers, where V_l is the width of the l-th layer, and $\mathcal{W} = \{\mathbf{W}_l\}_{l=1}^{L}$ is the collection of $V_l \times V_{l-1}$ synaptic weight matrices which connect neuronal layers sequentially. The outputs of the layers are $\{\mathbf{v}_l\}_{l=0}^{L}$, where $\mathbf{v}_0$ is the input layer, $\{\mathbf{v}_l\}_{l=1}^{L-1}$ are the hidden layers and $\mathbf{v}_L$ is the output layer. In each layer,

$$\mathbf{v}_l = \mathrm{sign}(\mathbf{W}_l \mathbf{v}_{l-1}) \tag{2.1}$$

where each sign "activation function" (a neuronal layer) operates component-wise (*i.e.*, $\forall i$: $(\mathrm{sign}(\mathbf{x}))_i = \mathrm{sign}(x_i)$). The output of the network is therefore

$$\mathbf{v}_L = g(\mathbf{v}_0, \mathcal{W}) = \mathrm{sign}(\mathbf{W}_L \mathrm{sign}(\mathbf{W}_{L-1} \mathrm{sign}(\cdots \mathbf{W}_1 \mathbf{v}_0)))\ . \tag{2.2}$$

---

[2] *i.e.*, having only a single layer of adjustable weights.

We assume that the weights are constrained to some set $\mathcal{S}$, with the specific restrictions on each weight denoted by $S_{ij,l}$, so $W_{ij,l} \in S_{ij,l}$ and $\mathcal{W} \in \mathcal{S}$. If $S_{ij,l} = \{0\}$, then we say that $W_{ij,l}$ is "disconnected". For simplicity, we assume that in each layer the "fan-in" $K_l = |\{j|S_{ij,l} \neq \{0\}\}|$ is constant for all i. Biases can be optionally included in the standard way, by adding a constant output $v_{0,l} = 1$ to each layer.

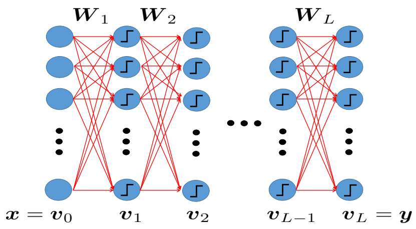

Figure 2.1: Our MNN model (Eq. 2.2).

Task We examine a supervised classification learning task, in Bayesian framework. We are given a *fixed* set of sequentially labeled data pairs $D_N = \left\{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right\}_{n=1}^{N}$ (so $D_0 = \emptyset$), where each $\mathbf{x}^{(n)} \in \mathbb{R}^{V_0}$ is a data point, and each $\mathbf{y}^{(n)}$ is a label taken from a binary set $\mathcal{Y} \subset \{-1, +1\}^{V_L}$. For brevity, we will sometimes suppress the sample index n, where it is clear from the context. As common for supervised learning with MNNs, we assume that for all n the relation $\mathbf{x}^{(n)} \to \mathbf{y}^{(n)}$ can be represented by a MNN with known architecture (the 'hypothesis class'), and unknown weights $\mathcal{W} \in \mathcal{S}$. This is a reasonable assumption since a MNN can approximate any deterministic function, given that it has sufficient number of neurons [18] (if $L \geq 2$). Specifically, there exists some $\mathcal{W}^* \in \mathcal{S}$, so that $\mathbf{y}^{(n)} = f\left(\mathbf{x}^{(n)}, \mathcal{W}^*\right)$ (see Eq. 2.2). Our goals are: (1) estimate the most probable $\mathcal{W}^*$ for this MNN, (2) estimate the most probable $\mathbf{y}$ given some (possibly unseen) $\mathbf{x}$.

3 Theory

In this section we explain how a specific learning algorithm for MNNs (described in section 4) arises from approximate (mean-field) Bayesian inference, used in this context (described in section 2).

3.1 Online Bayesian learning in MNNs

We approach this task within a Bayesian framework, where we assume some prior distribution on the weights - $P(\mathcal{W}|D_0)$. Our aim is to find $P(\mathcal{W}|D_N)$, the posterior probability for the configuration of the weights $\mathcal{W}$, given the data. With this posterior, one can select the most probable weight configuration - the Maximum A Posteriori (MAP) weight estimate

$$\mathcal{W}^* = \mathrm{argmax}_{\mathcal{W} \in \mathcal{S}} P(\mathcal{W}|D_N) \ , \tag{3.1}$$

minimizing the expected zero-one loss *over the weights* ($\mathcal{I}\{\mathcal{W}^* \neq \mathcal{W}\}$). This weight estimate can be implemented in a single MNN, which can provide an estimate of the label $\mathbf{y}$ for (possibly unseen) data points $\mathbf{x}$ through $\mathbf{y} = g(\mathbf{x}, \mathcal{W}^*)$. Alternatively, one can aim to minimize the expected loss *over the output* - as more commonly done in the MNN literature. For example, if the aim is to reduce classification error then one should use the MAP output estimate

$$\mathbf{y}^* = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} \sum_{\mathcal{W}} \mathcal{I}\{g(\mathbf{x}, \mathcal{W}) = \mathbf{y}\} P(\mathcal{W}|D_N) \ , \tag{3.2}$$

which minimizes the zero-one loss ($\mathcal{I}\{\mathbf{y}^* \neq g(\mathbf{x}, \mathcal{W})\}$) over the outputs. The resulting estimator does not generally have the form of a MNN (*i.e.*, $\mathbf{y} = g(\mathbf{x}, \mathcal{W})$ with $\mathcal{W} \in \mathcal{S}$), but can be approximated by averaging the output over many such MNNs with $\mathcal{W}$ values sampled from the posterior. Note that averaging the output of several MNNs is a common method to improve performance.

We aim to find the posterior $P(\mathcal{W}|D_N)$ in an online setting, where samples arrive sequentially. After the n-th sample is received, the posterior is updated according to Bayes rule:

$$P(\mathcal{W}|D_n) \propto P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \mathcal{W}\right) P(\mathcal{W}|D_{n-1}) \ , \tag{3.3}$$

for $n = 1, \ldots, N$. Note that the MNN is *deterministic*, so the likelihood (per data point) has the following simple form[3]

$$P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \mathcal{W}\right) = \mathcal{I}\left\{g\left(\mathbf{x}^{(n)}, \mathcal{W}\right) = \mathbf{y}^{(n)}\right\} . \tag{3.4}$$

---

[3] MNN with stochastic activation functions will have a "smoothed out" version of this.

Therefore, the Bayes update in Eq. 3.3 simply makes sure that $P\left(\mathcal{W}|D_n\right)=0$ in any "illegal" configuration (*i.e.*, any $\mathcal{W}^0$ such that $g\left(\mathbf{x}^{(k)},\mathcal{W}^0\right)\neq\mathbf{y}^{(k)}$) for some $1\leq k\leq n$. In other words, the posterior is equal to the prior, restricted to the "legal" weight domain, and re-normalized appropriately. Unfortunately, this update is generally intractable for large networks, mainly because we need to store and update an exponential number of values for $P\left(\mathcal{W}|D_n\right)$. Therefore, some approximation is required.

3.2 Approximation 1: mean-field

In order to reduce computational complexity, instead of storing $P\left(\mathcal{W}|D_n\right)$, we will store its factorized ('mean-field') approximation $\hat{P}\left(\mathcal{W}|D_n\right)$, for which

$$\hat{P}\left(\mathcal{W}|D_n\right)=\prod_{i,j,l}\hat{P}\left(W_{ij,l}|D_n\right)\ , \tag{3.5}$$

where each factor must be normalized. Notably, it is easy to find the MAP estimate of the weights (Eq. 3.1) under this factorized approximation $\forall i,j,l$

$$W^*_{ij,l}=\mathrm{argmax}_{W_{ij,l}\in S_{ij,l}}\hat{P}\left(W_{ij,l}|D_N\right)\ . \tag{3.6}$$

The factors $\hat{P}\left(W_{ij,l}|D_n\right)$ can be found using a standard variational approach [7, 30]. For each n, we first perform the Bayes update in Eq. 3.3 with $\hat{P}\left(\mathcal{W}|D_{n-1}\right)$ instead of $P\left(\mathcal{W}|D_{n-1}\right)$. Then, we project the resulting posterior onto the family of distributions factorized as in Eq. 3.5, by minimizing the *reverse* Kullback-Leibler divergence (similarly to EP [25, 28]). A straightforward calculation shows that the optimal factor is just a marginal of the posterior (appendix A, available in the supplementary material). Performing this marginalization on the Bayes update and re-arranging terms, we obtain a Bayes-like update to the marginals $\forall i,j,l$

$$\hat{P}\left(W_{ij,l}|D_n\right)\propto\hat{P}\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},W_{ij,l},D_{n-1}\right)\hat{P}\left(W_{ij,l}|D_{n-1}\right)\ , \tag{3.7}$$

where

$$\hat{P}\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},W_{ij,l},D_{n-1}\right)=\sum_{\mathcal{W}':W'_{ij,l}=W_{ij,l}}P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\mathcal{W}'\right)\prod_{\{k,r,m\}\neq\{i,j,l\}}\hat{P}\left(W'_{kr,m}|D_{n-1}\right) \tag{3.8}$$

is the marginal likelihood. Thus we can directly update the factor $\hat{P}\left(W_{ij,l}|D_n\right)$ in a single step. However, the last equation is still problematic, since it contains a generally intractable summation over an exponential number of values, and therefore requires simplification. For simplicity, from now on we replace any $\hat{P}$ with P, in a slight abuse of notation (keeping in mind that the distributions are approximated).

3.3 Simplifying the marginal likelihood

In order to be able to use the update rule in Eq. 3.7, we must first calculate the marginal likelihood $P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},W_{ij,l},D_{n-1}\right)$ using Eq. 3.8. For brevity, we suppress the index n and the dependence on D_{n-1} and $\mathbf{x}$, obtaining

$$P\left(\mathbf{y}|W_{ij,l}\right)\ =\sum_{\mathcal{W}':W'_{ij,l}=W_{ij,l}}P\left(\mathbf{y}|\mathcal{W}'\right)\prod_{\{k,r,m\}\neq\{i,j,l\}}P\left(W'_{kr,m}\right)\ , \tag{3.9}$$

where we recall that $P\left(\mathbf{y}|\mathcal{W}'\right)$ is simply an indicator function (Eq. 3.4). Since, by assumption, $P\left(\mathbf{y}|\mathcal{W}'\right)$ arises from a feed-forward MNN with input $\mathbf{v}_0=\mathbf{x}$ and output $\mathbf{v}_L=\mathbf{y}$, we can perform the summations in Eq. 3.9 in a more convenient way - layer by layer. To do this, we define

$$P\left(\mathbf{v}_m|\mathbf{v}_{m-1}\right)\ =\ \sum_{\mathbf{W}'_m}\prod_{k=1}^{V_m}\left[\mathcal{I}\left\{v_{k,m}\sum_{r=1}^{V_{m-1}}v_{r,m-1}W'_{kr,m}>0\right\}\prod_{r=1}^{V_{m-1}}P\left(W'_{kr,m}\right)\right] \tag{3.10}$$

and $P\left(\mathbf{v}_l|\mathbf{v}_{l-1},W_{ij,l}\right)$, which is defined identically to $P\left(\mathbf{v}_l|\mathbf{v}_{l-1}\right)$, except that the summation is performed over all configurations in which $W_{ij,l}$ is fixed (*i.e.*, $\mathbf{W}'_l\,:\,W'_{ij,l}=W_{ij,l}$) and we set

$P(W_{ij,l})=1$. Now we can write recursively $P(\mathbf{v}_1)=P(\mathbf{v}_1|\mathbf{v}_0=\mathbf{x})$

$$\forall m\in\{2,..,l-1\}:\;P(\mathbf{v}_m)=\sum_{\mathbf{v}_{m-1}}P(\mathbf{v}_m|\mathbf{v}_{m-1})\,P(\mathbf{v}_{m-1}) \tag{3.11}$$

$$P(\mathbf{v}_l|W_{ij,l})=\sum_{\mathbf{v}_{l-1}}P(\mathbf{v}_l|\mathbf{v}_{l-1},W_{ij,l})\,P(\mathbf{v}_{l-1}) \tag{3.12}$$

$$\forall m\in\{l+1,l+2,..,L\}:\;P(\mathbf{v}_m|W_{ij,l})=\sum_{\mathbf{v}_{m-1}}P(\mathbf{v}_m|\mathbf{v}_{m-1})\,P(\mathbf{v}_{m-1}|W_{ij,l}) \tag{3.13}$$

Thus we obtain the result of Eq. 3.9, through $P(\mathbf{y}|W_{ij,l})=P(\mathbf{v}_L=\mathbf{y}|W_{ij,l})$. However, this computation is still generally intractable, since all of the above summations (Eqs. 3.10-3.13) are still over an exponential number of values. Therefore, we need to make one additional approximation.

3.4 Approximation 2: large fan-in

Next we simplify the above summations (Eqs. 3.10-3.13) assuming that the neuronal fan-in is "large". We keep in mind that i,j and l are the specific indices of the fixed weight $W_{ij,l}$. All the other weights beside $W_{ij,l}$ can be treated as independent random variables, due to the mean field approximation (Eq. 3.5). Therefore, in the limit of a infinite neuronal fan-in ($\forall m:\,K_m\to\infty$) we can use the Central Limit Theorem (CLT) and say that the normalized input to each neuronal layer, is distributed according to a Gaussian distribution

$$\forall m:\;\mathbf{u}_m=\mathbf{W}_m\mathbf{v}_{m-1}/\sqrt{K_m}\sim\mathcal{N}(\boldsymbol{\mu}_m,\boldsymbol{\Sigma}_m)\;. \tag{3.14}$$

Since K_m is actually finite, this would be only an approximation - though a quite common and effective one (*e.g.*, [28]). Using the approximation in Eq. 3.14 together with $\mathbf{v}_m=\mathrm{sign}(\mathbf{u}_m)$ (Eq. 2.1) we can calculate (appendix B) the distribution of $\mathbf{u}_m$ and $\mathbf{v}_m$ sequentially for all the layers $m\in\{1,\dots,L\}$, for any given value of $\mathbf{v}_0$ and $W_{ij,l}$. These effectively simplify the summations in 3.10-3.13 using Gaussian integrals (appendix B).

At the end of this "forward pass" we will be able to find $P(\mathbf{y}|W_{ij,l})=P(\mathbf{v}_L=\mathbf{y}|W_{ij,l})$, $\forall i,j,l$. This takes a polynomial number of steps (appendix B.3), instead of a direct calculation through Eqs. 3.11-3.13, which is exponentially hard. Using $P(\mathbf{y}|W_{ij,l})$ and Eq. 3.7 we can now update the distribution of $P(W_{ij,l})$. This immediately gives the Bayes estimate of the weights (Eq. 3.6) and outputs (Eq. 3.2).

As we note in appendix B.3, the computational complexity of the forward pass is significantly lower if $\boldsymbol{\Sigma}_m$ is diagonal. This is true exactly only in special cases. For example, this is true if all hidden neurons have a fan-out of one - such as in a 2-layer network with a single output. However, in order to reduce the computational complexity in cases that $\boldsymbol{\Sigma}_m$ is not diagonal, we will approximate the distribution of $\mathbf{u}_m$ with its factorized ('mean-field') version. Recall that the optimal factor is the marginal of the distribution (appendix A). Therefore, we can now find $P(\mathbf{y}|W_{ij,l})$ easily (appendix B.1), as all the off-diagonal components in $\boldsymbol{\Sigma}_m$ are zero, so $\boldsymbol{\Sigma}_{kk',m}=\sigma^2_{k,m}\delta_{kk'}$.

A direct calculation of $P(\mathbf{v}_L=\mathbf{y}|W_{ij,l})$ for every i,j,l would be computationally wasteful, since we will repeat similar calculations many times. In order to improve the algorithm's efficiency, we again exploit the fact that K_l is large. We approximate $\ln P(\mathbf{v}_L=\mathbf{y}|W_{ij,l})$ using a Taylor expansion of $W_{ij,l}$ around its mean, $\langle W_{ij,l}\rangle$, to first order in $K_l^{-1/2}$. The first order terms in this expansion can be calculated using backward propagation of derivative terms

$$\Delta_{k,m}=\partial\ln P(\mathbf{v}_L=\mathbf{y})/\partial\mu_{k,m}\,, \tag{3.15}$$

similarly to the BP algorithm (appendix C). Thus, after a forward pass for $m=1,\dots,L$, and a backward pass for $l=L,\dots,1$, we obtain $P(\mathbf{v}_L=\mathbf{y}|W_{ij,l})$ for all $W_{ij,l}$ and update $P(W_{ij,l})$.

4 The Expectation Backpropagation Algorithm

Using our results we can efficiently update the posterior distribution $P(W_{ij,l}|D_n)$ for all the weights with $O(|\mathcal{W}|)$ operations, according to Eqs. 3.7. Next, we summarize the resulting general algorithm - the Expectation BackPropgation (EBP) algorithm. In appendix D, we exemplify how to apply the algorithm in the special cases of MNNs with binary, ternary or real (continuous) weights. Similarly to the original BP algorithm (see review in [22]), given input $\mathbf{x}$ and desired output $\mathbf{y}$, first we perform a forward pass to calculate the mean output $\langle \mathbf{v}_l \rangle$ for each layer. Then we perform a backward pass to update $P\left(W_{ij,l}|D_n\right)$ for all the weights.

Forward pass In this pass we perform the forward calculation of probabilities, as in Eq. 3.11. Recall that $\langle W_{kr,m} \rangle$ is the mean of the posterior distribution $P\left(W_{kr,m}|D_n\right)$. We first initialize the MNN input $\langle v_{k,0} \rangle = x_k$ for all k and calculate recursively the following quantities (Eqs. B.5-B.7) for $m = 1, \dots, L$ and all k

$$\mu_{k,m} = \frac{1}{\sqrt{K_m}} \sum_{r=1}^{V_{m-1}} \langle W_{kr,m} \rangle \langle v_{r,m-1} \rangle \quad ; \quad \langle v_{k,m} \rangle = 2\Phi\left(\mu_{k,m}/\sigma_{k,m}\right) - 1\,. \tag{4.1}$$

$$\sigma_{k,m}^2 = \frac{1}{K_m} \sum_{r=1}^{V_{m-1}} \langle W_{kr,m}^2 \rangle \left(\delta_{m,1}\left(\langle v_{r,m-1} \rangle^2 - 1\right) + 1\right) - \langle W_{kr,m} \rangle^2 \langle v_{r,m-1} \rangle^2\,, \tag{4.2}$$

where $\boldsymbol{\mu}_m$ and $\boldsymbol{\sigma}_m^2$ are, respectively, the mean and variance of $\mathbf{u}_m$, the input of layer m (Eq. 3.14), and $\langle \mathbf{v}_m \rangle$ is the resulting mean of the output of layer m.

Backward pass In this pass we perform the Bayes update of the posterior (Eq. 3.7) using a Taylor expansion. Recall Eq. 3.15. We first initialize[4] (Eq. C.8)

$$\Delta_{i,L} = y_i \frac{\mathcal{N}\left(0|\mu_{i,L}, \sigma_{i,L}^2\right)}{\Phi\left(y_i \mu_{i,L}/\sigma_{i,L}\right)}\,. \tag{4.3}$$

for all i. Then, for $l = L, \dots, 1$ and $\forall i, j$ we calculate (Eqs. C.9-C.10, C.6 and 3.7)

$$\Delta_{i,l-1} = \frac{2}{\sqrt{K_l}} \mathcal{N}\left(0|\mu_{i,l-1}, \sigma_{i,l-1}^2\right) \sum_{j=1}^{V_m} \langle W_{ji,l} \rangle \Delta_{j,l}\,. \tag{4.4}$$

$$\ln P\left(W_{ij,l}|D_n\right) = \ln P\left(W_{ij,l}|D_{n-1}\right) + \frac{1}{\sqrt{K_l}} W_{ij,l} \Delta_{i,l} \langle v_{j,l-1} \rangle + C\,, \tag{4.5}$$

where C is some unimportant constant (which does not depend on $W_{ij,l}$).

Output Using the posterior distribution, the optimal configuration can be immediately found through the MAP weights estimate (Eq. 3.6) $\forall i, j, l$

$$W_{ij,l}^* = \mathrm{argmax}_{W_{ij,l} \in S_{ij,l}} \ln P\left(W_{ij,l}|D_n\right)\,. \tag{4.6}$$

The output of a MNN implementing these weights would be $g\left(\mathbf{x}, \mathcal{W}^*\right)$ (see Eq. 2.2). We define this to be the ‘deterministic‘ EBP output (EBP-D).

Additionally, the MAP output (Eq. 3.2) can be calculated directly

$$\mathbf{y}^* = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} \ln P\left(\mathbf{v}_L = \mathbf{y}\right) = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} \left[\sum_k \ln\left(\frac{1 + \langle v_{k,L} \rangle}{1 - \langle v_{k,L} \rangle}\right)^{y_k}\right] \tag{4.7}$$

using $\langle v_{k,L} \rangle$ from Eq. 4.1, or as an ensemble average over the outputs of all possible MNN with the weights $W_{ij,l}$ being sampled from the estimated posterior $P\left(W_{ij,l}|D_n\right)$. We define the output in Eq. 4.7 to be the Probabilistic EBP output (EBP-P). Note that in the case of a single output $\mathcal{Y} = \{-1, 1\}$, so this output simplifies to $y = \mathrm{sign}\left(\langle v_{k,L} \rangle\right)$.

---

[4] Due to numerical inaccuracy, calculating $\Delta_{i,L}$ using Eq. 4.3 can generate nonsensical values ($\pm\infty$, NaN) if $|\mu_{i,L}/\sigma_{i,L}|$ becomes to large. If this happens, we use instead the asymptotic form in that limit

$$\Delta_{i,L} = -\frac{\mu_{i,L}}{\sigma_{i,L}^2 \sqrt{K_L}} \mathcal{I}\left\{y_i \mu_{i,L} < 0\right\}$$

5 Numerical Experiments

We use several high dimensional text datasets to assess the performance of the EBP algorithm in a supervised binary classification task. The datasets (taken from [11]) contain eight binary tasks from four datasets: ‘Amazon (sentiment)’, ‘20 Newsgroups’, ‘Reuters’ and ‘Spam or Ham’. Data specification (N =#examples and M =#features) and results (for each algorithm) are described in Table 1. More details on the data including data extraction and labeling can be found in [11].

We test the performance of EBP on MNNs with a 2-layer architecture of $M \to 120 \to 1$, and bias weights. We examine two special cases: (1) MNNs with real weights (2) MNNs with binary weights (and real bias). Recall the motivation for the latter (section 1) is that they can be efficiently implemented in hardware (real bias has negligible costs). Recall also that for each type of MNN, the algorithm gives two outputs - EBP-D (deterministic) and EBP-P (probabilistic), as explained near Eqs. 4.6-4.7.

To evaluate our results we compare EBP to: (1) the AROW algorithm, which reports state-of-the-art results on the tested datasets [11] (2) the traditional Backpropagation (BP) algorithm, used to train an $M \to 120 \to 1$ MNN with real weights. In the latter case, we used both Cross Entropy (CE) and Mean Square Error (MSE) as loss functions. On each dataset we report the results of BP with the loss function which achieved the minimal error. We use a simple parameter scan for both AROW (regularization parameter) and the traditional BP (learning rate parameter). Only the results with the optimal parameters (*i.e.*, achieving best results) are reported in Table 1. The optimal parameters found were never at the edges of the scanned field. Lastly, to demonstrate the destructive effect of naive quantization, we also report the performance of the BP-trained MNNs, after all the weights (except the bias) were clipped using a sign function.

During training the datasets were repeatedly presented in three epochs (in all algorithms, additional epochs did not reduce test error). On each epoch the examples were shuffled at random order for BP and EBP (AROW determines its own order). The test results are calculated after each epoch using 8-fold cross-validation, similarly to [11]. Empirically, EBP running time is similar to BP with real weights, and twice slower with binary weights. For additional implementation details, see appendix E.1. The code is available on the author’s website.

The minimal values achieved over all three epochs are summarized in Table 1. As can be seen, in all datasets EBP-P performs better then AROW, which performs better then BP. Also, EBP-P usually perfroms better with binary weights. In appendix E.2 we show that this ranking remains true even if the fan-in is small (in contrast to our assumptions), or if a deeper 3-layer architecture is used.

Dataset	#Examples	#Features	Real EBP-D	Real EBP-P	Binary EBP-D	Binary EBP-P	AROW	BP	Clipped BP
Reuters news I6	2000	11463	14.5%	11.35%	21.7%	**9.95%**	11.72%	13.3%	26.15%
Reuters news I8	2000	12167	15.65%	**15.25%**	23.15%	16.4%	15.27%	18.2%	26.4%
Spam or ham d0	2500	26580	1.28%	1.11%	7.93%	**0.76%**	1.12%	1.32%	7.97%
Spam or ham d1	2500	27523	1.0%	**0.96%**	3.85%	**0.96%**	1.4%	1.36%	7.33%
20News group comp vs HW	1943	29409	5.06%	4.96%	7.54%	**4.44%**	5.79%	7.02%	13.07%
20News group elec vs med	1971	38699	3.36%	3.15%	6.0%	**2.08%**	2.74%	3.96%	14.23%
Amazon Book reviews	3880	221972	2.14%	2.09%	2.45%	**2.01%**	2.24%	2.96%	3.81%
Amazon DVD reviews	3880	238739	**2.06%**	2.14%	5.72%	2.27%	2.63%	2.94%	5.15%

Table 1: Data specification, and test errors (with 8-fold cross-validation). Best results are boldfaced.

6 Discussion

Motivated by the recent success of MNNs, we developed the Expectation BackPropagation algorithm (EBP - see section 4) for approximate Bayesian inference of the synaptic weights of a MNN. Given a supervised classification task with labeled training data and a prior over the weights, this deterministic online algorithm can be used to train deterministic MNNs (Eq. 2.2) without the need to tune learning parameters (*e.g.*, learning rate). Furthermore, each synaptic weight can be restricted to some set - which can be either finite (*e.g.*, binary numbers) or infinite (*e.g.*, real numbers). This opens the possibility of implementing trained MNNs in power-efficient hardware devices requiring limited parameter precision.

This algorithm is essentially an analytic approximation to the intractable Bayes calculation of the posterior distribution of the weights after the arrival of a new data point. To simplify the intractable Bayes update rule we use several approximations. First, we approximate the posterior using a product of its marginals - a 'mean field' approximation. Second, we assume the neuronal layers have a large fan-in, so we can approximate them as Gaussian. After these two approximations each Bayes update can be tractably calculated in polynomial time in the size of the MNN. However, in order to further improve computational complexity (to $O\left(|\mathcal{W}|\right)$ in each step, like BP), we make two additional approximations. First, we use the large fan-in to perform a first order expansion. Second, we optionally[5] perform a second 'mean field' approximation - to the distribution of the neuronal inputs. Finally, after we obtain the approximated posterior using the algorithm, the Bayes estimates of the most probable weights and the outputs are found analytically.

Previous approaches to obtain these Bayes estimates were too limited for our purposes. The Monte Carlo approach [27] achieves state-of-the-art performance for small MNNs [32], but does not scale well [31]. The Laplace approximation [23] and variational Bayes [16, 4, 14] based methods require real-value weights, tuning of the learning rate parameter, and stochastic neurons (to "smooth" the likelihood). Previous EP [30, 28] and message passing [8, 3] (a special case of EP[7]) based methods were derived only for SNNs.

In contrast, the EBP allows parameter free and scalable training of various types of MNNs (deterministic or stochastic) with discrete (*e.g.*, binary) or continuous weights. In appendix F, we see that for continuous weights EBP is almost identical to standard BP with a specific choice of activation function $s\left(x\right)=2\Phi\left(x\right)-1$, CE loss and learning rate $\eta=1$. The only difference is that the input is normalized by its standard deviation (Eq. 4.1, *right*), which depends on the weights and inputs (Eq. 4.2). This re-scaling makes the learning algorithm invariant to the amplitude changes in the neuronal input. This results from the same invariance of the sign activation functions. Note that in standard BP algorithm the performance is directly affected by the amplitude of the input, so it is a recommended practice to re-scale it in pre-processing [22].

We numerically evaluated the algorithm on binary classification tasks using MNNs with two or three synaptic layers. In all data sets and MNNs EBP performs better than standard BP with the optimal constant learning rate, and even achieves state-of-the-art results in comparison to [11]. Surprisingly, EBP usually performs best when it is used to train binary MNNs. As suggested by a reviewer, this could be related to the type of problems examined here. In text classification tasks have large sparse input spaces (bag of words), and presence/absence of features (words) is more important than their real values (frequencies). Therefore, (distributions over) binary weights and a threshold activation function may work well.

In order to get such a good performance in binary MNNs, one must average over the output the inferred (approximate) posterior of the weights. The EBP-P output of the algorithm calculates this average analytically. In hardware this output could be realizable by averaging the output of several binary MNNs, by sampling weights from $P\left(W_{ij,l}|D_n\right)$. This can be done efficiently (appendix G).

Our numerical testing mainly focused on high-dimensional text classification tasks, where shallow architectures seem to work quite well. In other domains, such as vision [20] and speech [12], deep architectures achieve state-of-the-art performance. Such deep MNNs usually require considerable fine-tuning and additional 'tricks' such as unsupervised pre-training [12], weight sharing [20] or momentum[6]. Integrating such methods into EBP and using it to train deep MNNs is a promising direction for future work. Another important generalization of the algorithm, which is rather straightforward, is to use activation functions other than $\mathrm{sign}\left(\cdot\right)$. This is particularly important for the last layer - where a linear activation function would be useful for regression tasks, and joint activation functions[7] would be useful for multi-class tasks[6].

Acknowledgments The authors are grateful to C. Baldassi, A. Braunstein and R. Zecchina for helpful discussions and to A. Hallak, T. Knafo and U. Sümbül for reviewing parts of this manuscript. The research was partially funded by the Technion V.P.R. fund, by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), and by the Gruss Lipper Charitable Foundation.

---

[5] This approximation is not required if all neurons in the MNN have a fan-out of one.

[6] Which departs from the online framework considered here, since it requires two samples in each update.

[7] *i.e.*, activation functions for which $\left(f\left(\mathbf{x}\right)\right)_i\neq f\left(x_i\right)$, such as softmax or argmax.

Appendix

A The mean-field approximation

In this section we derive Eqs. 3.7 and 3.8. Recall Eq. 3.5,

$$\hat{P}\left(\mathcal{W}|D_n\right) = \prod_{i,j,l} \hat{P}\left(W_{ij,l}|D_n\right) \,,$$

where $\hat{P}\left(\mathcal{W}|D_n\right)$ is an approximation of $P\left(\mathcal{W}|D_n\right)$. In this section we answer the following question - suppose we know $\hat{P}\left(\mathcal{W}|D_{n-1}\right)$. How do we find $\hat{P}\left(\mathcal{W}|D_n\right)$? It is a standard approximation to answer this question using a variational approach (see [7], and note that [30, 28] also used the same approach for a SNN with binary weights), through the following two steps:

1. We use the Bayes update (Eq. 3.3) with $\hat{P}\left(\mathcal{W}|D_{n-1}\right)$ as our prior

   $$\begin{aligned} \tilde{P}\left(\mathcal{W}|D_n\right) &\propto P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\mathcal{W}\right)\hat{P}\left(\mathcal{W}|D_{n-1}\right) \\ &= P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\mathcal{W}\right)\prod_{i,j,l}\hat{P}\left(W_{ij,l}|D_{n-1}\right) \,, \end{aligned} \tag{A.1}$$

   where $\tilde{P}\left(\mathcal{W}|D_n\right)$ is some "temporary" posterior distribution.

2. We project $\hat{P}\left(\mathcal{W}|D_n\right)$ onto $\tilde{P}\left(\mathcal{W}|D_n\right)$ by minimizing the reverse Kullback-Leibler divergence (e.g., as in the expectation propagation algorithm [25, 7])

   $$D_{KL}\left(\tilde{P}\left(\mathcal{W}|D_n\right)||\hat{P}\left(\mathcal{W}|D_n\right)\right) = \sum_{\mathcal{W}}\tilde{P}\left(\mathcal{W}|D_n\right)\log\left(\frac{\tilde{P}\left(\mathcal{W}|D_n\right)}{\hat{P}\left(\mathcal{W}|D_n\right)}\right)$$

   with the normalization constraint $\sum_{W_{ij,l}}\hat{P}\left(W_{ij,l}|D_n\right) = 1\;\forall i,j,l$.

The second step can be easily performed using Lagrange multipliers, forming a Lagrangian

$$\begin{aligned} L\left(\hat{P}\left(\mathcal{W}|D_n\right)\right) &= \sum_{\mathcal{W}'\in\mathcal{S}}\tilde{P}\left(\mathcal{W}'|D_n\right)\log\left(\frac{\tilde{P}\left(\mathcal{W}'|D_n\right)}{\prod_{k,r,m}\hat{P}\left(W'_{kr,m}|D_n\right)}\right) \\ &+ \sum_{k,r,m}\lambda_{kr,m}\left(1-\sum_{W'_{kr,m}\in S_{kr,m}}\hat{P}\left(W'_{kr,m}|D_n\right)\right) \,. \end{aligned}$$

The minimum is found by differentiating and equating to zero

$$0 = \frac{\partial L\left(\hat{P}\left(\mathcal{W}|D_n\right)\right)}{\partial\hat{P}\left(W_{ij,l}|D_n\right)} = -\frac{\sum_{\mathcal{W}':W'_{ij,l}=W_{ij,l}}\tilde{P}\left(\mathcal{W}'|D_n\right)}{\hat{P}\left(W_{ij,l}|D_n\right)} - \lambda_{ij,l} \,.$$

Using this equation together with the normalization constraint $\sum_{W_{ij,l}}\hat{P}\left(W_{ij,l}|D_n\right) = 1\;\forall i,j,l$ we obtain the result of the minimization through marginalization

$$\hat{P}\left(W_{ij,l}|D_n\right) = \sum_{\mathcal{W}':W'_{ij,l}=W_{ij,l}}\tilde{P}\left(\mathcal{W}'|D_n\right) \,, \tag{A.2}$$

which is a known result [7, p. 468]. Finally, we can combine step 1 (Bayes update) with step 2 (projection) to a single step

$$\hat{P}\left(W_{ij,l}|D_n\right) = \sum_{\mathcal{W}':W'_{ij,l}=W_{ij,l}} P\left(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\mathcal{W}'\right)\prod_{k,r,m}\hat{P}\left(W'_{kr,m}|D_{n-1}\right) \,.$$

This step is exactly Eqs. 3.7 and 3.8 combined.

B Forward propagation of probabilities

In this section we simplify the summations in Eqs. (3.10-3.13), by assuming that the fan-in of all of the connections is "large", *i.e.*, $\forall m : K_m \to \infty$. Using this approximation and the CLT we can write $\forall m \geq 1$

$$\mathbf{u}_m = \frac{1}{\sqrt{K_m}} \mathbf{W}_m \mathbf{v}_{m-1} \sim \mathcal{N}\left(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m\right) . \tag{B.1}$$

$$\mathbf{v}_m = \operatorname{sign}\left(\mathbf{u}_m\right) \tag{B.2}$$

with $\mathbf{v}_0 = \mathbf{x}$. Recall (from Eq. 3.9) that $W_{ij,l}$ is a specific weight which is fixed (so i, j and l are "special" indexes), while all the other weights $W_{kr,m}$ (for which $k \neq i$ or $r \neq j$ or $m \neq l$) are independent variables. We first consider a simple special case.

B.1 Special case: a diagonal $\boldsymbol{\Sigma}_m$

In this section we assume initially that $\boldsymbol{\Sigma}_m$ in Eq. B.1 is diagonal, so

$$P\left(\mathbf{u}_m\right) = \prod_k \mathcal{N}\left(u_{k,m} | \mu_{k,m}, \sigma^2_{k,m}\right) \tag{B.3}$$

with $\sigma^2_{k,m} = \Sigma_{kk,m}$. Therefore, $\forall m \in \{1, \dots, l-1\}$, we can use Eq. B.2 to obtain

$$P\left(\mathbf{v}_m\right) = \prod_k P\left(v_{k,m}\right) = \prod_k \Phi\left(v_{k,m}\mu_{k,m}/\sigma_{k,m}\right) . \tag{B.4}$$

These distributions, which are the approximate solution of Eqs. 3.11-3.13, immediately give

$$\left\langle v_{k,m}\right\rangle = v_{k,0}\delta_{0m} + \left(1 - \delta_{0m}\right)\left(2\Phi\left(\mu_{k,m}/\sigma_{k,m}\right) - 1\right) . \tag{B.5}$$

Note that $W_{kr,m}$ and $v_{k,m-1}$ are independent for a fixed m (from Eqs. 3.5 and 2.1). Therefore, it is straightforward to derive

$$\mu_{k,m} = \left\langle u_{k,m}\right\rangle = \frac{1}{\sqrt{K_m}} \sum_{r=1}^{V_{m-1}} \left\langle W_{kr,m}\right\rangle \left\langle v_{r,m-1}\right\rangle \tag{B.6}$$

$$\sigma^2_{k,m} = \operatorname{Var}\left(u_{k,m}\right) = \frac{1}{K_m} \sum_{r=1}^{V_{m-1}} \left\langle W^2_{kr,m}\right\rangle \left\langle v^2_{r,m-1}\right\rangle - \left\langle W_{kr,m}\right\rangle^2 \left\langle v_{r,m-1}\right\rangle^2 . \tag{B.7}$$

Since the value $\mathbf{v}_0$ is given ($\mathbf{v}_0 = \mathbf{x}$), and for $m \geq 1$, $\mathbf{v}_m$ are binary vectors, we have $\left\langle v^2_{r,m-1}\right\rangle = 1 + \delta_{m0}\left(v^2_{r,0} - 1\right)$. Importantly, if we know $\mathbf{x}$ and $P\left(W_{kr,m}\right)$ Eqs. B.5-B.7 can be calculated together in a sequential "forward pass" for $m = 1, 2, ..., l-1$. We continue in the same manner for $m \geq l$, with slight modifications, since $W_{ij,l}$ is fixed. For $m = l$ we need to respectively replace $\left\langle W_{ij,l}\right\rangle$ and $\left\langle W^2_{ij,l}\right\rangle$ with fixed values $W_{ij,l}$ and $W^2_{ij,l}$ so

$$\mu_{i,l}\left(W_{ij,l}\right) = \mu_{i,l} + \frac{1}{\sqrt{K_l}}\left(W_{ij,l} - \left\langle W_{ij,l}\right\rangle\right)\left\langle v_{j,l-1}\right\rangle \tag{B.8}$$

$$\sigma^2_{i,l}\left(W_{ij,l}\right) = \sigma^2_{i,l} - \frac{1}{K_l}\left\langle W_{ij,l}\right\rangle^2 \operatorname{Var}\left(v_{j,l-1}\right) . \tag{B.9}$$

For $m > l$ we continue as in Eqs. B.4-B.7, except we replace $P\left(v_{k,m}\right), \left\langle v_{k,m}\right\rangle$, $\mu_{k,m}$ and $\sigma^2_{k,m}$ with $P\left(v_{k,m}|W_{ij,l}\right), \left\langle v_{k,m}|W_{ij,l}\right\rangle$, $\mu_{k,m}\left(W_{ij,l}\right)$ and $\sigma^2_{k,m}\left(W_{ij,l}\right)$, respectively, to emphasize that they depend on $W_{ij,l}$. The end result of this calculation is $P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)$.

B.2 General case: non-diagonal $\boldsymbol{\Sigma}_m$

In this section we perform the forward propagation (Eqs. 3.11-3.13) without assuming that $\boldsymbol{\Sigma}_m$ is diagonal. For simplicity, in this section we will usually suppress explicit dependence on $W_{ij,l}$ in our notation (keeping in mind that we should respectively replace $\left\langle W_{ij,l}\right\rangle$ and $\left\langle W^2_{ij,l}\right\rangle$ with $W_{ij,l}$ and $W^2_{ij,l}$).

Using these equations it is straightforward to derive $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$ for each layer and $\forall k, k'$

$$\begin{aligned}
\mu_{k,m} &= \frac{1}{\sqrt{K_m}} \sum_{r=1}^{V_{m-1}} \langle W_{kr,m} \rangle \langle v_{r,m-1} \rangle && \text{(B.10)} \\
\Sigma_{kk',m} &= \mathrm{Cov}\left(u_{k,m}, u_{k',m}\right) && \text{(B.11)} \\
&= \frac{1}{K_m} \delta_{kk'} \sum_{r=1}^{V_{m-1}} \mathrm{Var}\left(W_{kr,m}\right) \left\langle v_{r,m-1}^2 \right\rangle \\
&+ \frac{1}{K_m} \sum_{r=1}^{V_{m-1}} \sum_{r'=1}^{V_{m-1}} \langle W_{kr,m} \rangle \langle W_{k'r',m} \rangle \mathrm{Cov}\left(v_{r,m-1}, v_{r',m-1}\right) .
\end{aligned}$$

Therefore, for $m > 1$ and $\forall k$

$$\begin{aligned}
\langle v_{k,m} \rangle &= P\left(u_{k,m} > 0\right) - P\left(u_{k,m} < 0\right) \\
&= 2P\left(u_{k,m} > 0\right) - 1 \\
&= 2\Phi\left(\mu_{k,m} / \Sigma_{kk,m}\right) - 1 && \text{(B.12)}
\end{aligned}$$

and $\forall k' \neq k$:

$$\begin{aligned}
\langle v_{k,m} v_{k',m} \rangle &= P\left(\mathrm{sign}\left(u_{k,m} u_{k',m}\right) > 0\right) - P\left(\mathrm{sign}\left(u_{k,m} u_{k',m}\right) < 0\right) \\
&= P\left(u_{k,m} > 0, u_{k',m} > 0\right) + P\left(u_{k,m} < 0, u_{k',m} < 0\right) \\
&- P\left(u_{k,m} > 0, u_{k',m} < 0\right) - P\left(u_{k,m} < 0, u_{k',m} > 0\right) \\
&= 2P\left(u_{k,m} > 0, u_{k',m} > 0\right) + 2P\left(u_{k,m} < 0, u_{k',m} < 0\right) - 1 .
\end{aligned}$$

Note that for $m \geq l$, all these results depend on $W_{ij,l}$ (this dependency was suppressed, for brevity). Lastly, we obtain

$$P\left(\mathbf{y} | W_{ij,l}\right) = P\left(\forall k : \, u_{k,L} y_k > 0\right) . \tag{B.13}$$

All that remains is to find is to substitute $P\left(\mathbf{y} | W_{ij,l}\right)$ into Eq. 3.7 and perform the update rule.

B.3 Computational Complexity

What is the computational complexity of a direct implementation the resulting update rule for each weight? We first consider the complexity for the general case of a non-Diagonal $\boldsymbol{\Sigma}_m$. Denoting $S = \max_{i,j,l} |S_{ij,l}|$, $V = \max_l V_l$ and ϵ as the required (relative) precision for calculating $P\left(\mathbf{y} | W_{ij,l}\right)$, we can now find the worst case asymptotic complexity $\tilde{O}\left(\cdot\right)$ (*i.e.*, neglecting logarithmic factors) of all steps in the update rule. To do this, we first note that

- Given $P\left(W_{ij,l}\right)$, calculating $\langle W_{ij,l} \rangle$ is $O\left(S\right)$.
- Calculating $\Phi\left(x\right)$ is $\tilde{O}\left(1\right)$ [9].
- The current state-of-the-art complexity of calculating Gaussian orthant probabilities in d dimensions and relative precision ϵ:
  - For $d \leq 3$ it is $\tilde{O}\left(1\right)$ [13].
  - For $d \geq 4$ it is $\tilde{O}\left(d^3 \epsilon^{-2}\right)$ [10], and sometimes lower (*e.g.*, see $d = 4$ in [15]).

Therefore , in the non-diagonal case (appendix B.2)

- Eq. B.10 for all layers is $O\left(S \left|\mathcal{W}\right|\right)$.
- Eq. B.11 for all layers is $O\left(S \left|\mathcal{W}\right| V\right)$.
- Eq. B.12 for all neurons and all layers is $\tilde{O}\left(LV\right)$.
- Eq. **??** for all neurons and all layers is $\tilde{O}\left(\left|\mathcal{W}\right|\right)$.
- Eq. B.13 is $\tilde{O}\left(V_L^3 \epsilon^{-2}\right)$ if $V_L > 3$ or $\tilde{O}\left(1\right)$ otherwise.

Summing all contributions, in the worst case, the total computational complexity of a single update step (Eq. 3.7) for all weights and weight values is $\tilde{O}\left(S^2\left|\mathcal{W}\right|^2 V\right)$ if $V_L \leq 3$ or $\tilde{O}\left(S^2\left|\mathcal{W}\right|^2 V + V_L^3\epsilon^{-2}\left|\mathcal{W}\right| S\right)$ if $V_L > 3$.

If instead, $\mathbf{\Sigma}_m$ is diagonal, then, using a similar analysis, it is straightforward to show that computational complexity of a "naive" single update step (Eq. 3.7) for all weights and weight values is instead $O\left(S^2\left|\mathcal{W}\right|^2\right)$. However, this complexity can be further reduced to a linear complexity $O\left(S^2\left|\mathcal{W}\right|\right)$ by exploiting the fact that many similar operations are shared by the updates of different weights. We explain how this is done in Appendix C.

C Backward propagation of derivatives

In this section we calculate the Taylor expansion of $\ln P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)$ in $W_{ij,l}$ around $\langle W_{ij,l}\rangle$. Initially, we perform a similar Taylor expansion without the log. This yields, to first order

$$P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right) = P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l} = \langle W_{ij,l}\rangle\right) + \left(W_{ij,l} - \langle W_{ij,l}\rangle\right)\left[\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)}{\partial W_{ij,l}}\right]_{W_{ij,l}=\langle W_{ij,l}\rangle} \tag{C.1}$$

We re-write this expression to first order using the notation as in appendix B.1. First, using the chain rule, B.8 and Eq. B.9 we obtain

$$\begin{aligned}\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)}{\partial W_{ij,l}} &= \frac{\partial \mu_{i,l}}{\partial W_{ij,l}}\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)}{\partial \mu_{i,l}} + \frac{\partial \sigma_{i,l}^2}{\partial W_{ij,l}}\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)}{\partial \sigma_{i,l}^2} \\ &= \frac{1}{\sqrt{K_l}}\langle v_{j,l-1}\rangle\frac{\partial}{\partial \mu_{i,l}}P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right)\,. \qquad \text{(C.2)}\end{aligned}$$

Next, from Eqs. B.8-B.9

$$\mu_{i,l}\left(W_{ij,l} = \langle W_{ij,l}\rangle\right) = \mu_{i,l} \tag{C.3}$$

$$\sigma_{i,l}^2\left(W_{ij,l}\right) = \sigma_{i,l}^2 + O\left(K_l^{-1}\right) \tag{C.4}$$

and therefore

$$P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l} = \langle W_{ij,l}\rangle\right) = P\left(\mathbf{v}_L = \mathbf{y}\right) + O\left(K_l^{-1}\right) \tag{C.5}$$

Putting Eqs. C.1-C.5 together, we obtain

$$P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right) = P\left(\mathbf{v}_L = \mathbf{y}\right) + \frac{1}{\sqrt{K_l}}\left(W_{ij,l} - \langle W_{ij,l}\rangle\right)\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{i,l}}\langle v_{j,l-1}\rangle + O\left(K_l^{-1}\right)\,.$$

Taking the logarithm of this expression, we obtain to first order

$$\begin{aligned}\ln P\left(\mathbf{v}_L = \mathbf{y}|W_{ij,l}\right) &= \ln\left[P\left(\mathbf{v}_L = \mathbf{y}\right) + \frac{1}{\sqrt{K_l}}\left(W_{ij,l} - \langle W_{ij,l}\rangle\right)\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{i,l}}\langle v_{j,l-1}\rangle + O\left(K_l^{-1}\right)\right] \\ &= \ln\left[1 + \frac{1}{\sqrt{K_l}}\langle v_{j,l-1}\rangle\left(W_{ij,l} - \langle W_{ij,l}\rangle\right)\frac{\partial P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{i,l}}/P\left(\mathbf{v}_L = \mathbf{y}\right) + O\left(K_l^{-1}\right)\right] + C \\ &= C + \frac{1}{\sqrt{K_l}}W_{ij,l}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{i,l}}\langle v_{j,l-1}\rangle + O\left(K_l^{-1}\right)\,. \qquad \text{(C.6)}\end{aligned}$$

where C is some constant that does not depend on $W_{ij,l}$.

As we show next, the derivative term can be calculated efficiently to first order, using the chain rule. From Eq. B.4, we have

$$\begin{aligned}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{k,L}} &= \frac{\partial}{\partial \mu_{k,L}}\left[\sum_{r=1}^{V_L}\ln\Phi\left(y_r\mu_{r,L}/\sigma_{r,L}\right)\right] \qquad \text{(C.7)} \\ &= y_k\frac{\mathcal{N}\left(0|\mu_{k,L},\sigma_{k,L}^2\right)}{\Phi\left(y_k\mu_{k,L}/\sigma_{k,L}\right)}\,, \qquad \text{(C.8)}\end{aligned}$$

where we used the fact that for $a = \pm 1$

$$\frac{d}{dx}\Phi\left(ax/b\right) = a\mathcal{N}\left(0|x,b^2\right)\ .$$

From Eq. B.6-B.7, we have

$$\begin{aligned}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial\left\langle v_{k,m-1}\right\rangle} &= \sum_{r=1}^{V_m}\left[\frac{\partial \mu_{r,m}}{\partial\left\langle v_{k,m-1}\right\rangle}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{r,m}} + \frac{\partial \sigma^2_{r,m}}{\partial\left\langle v_{k,m-1}\right\rangle}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \sigma^2_{r,m}}\right] \\ &= \sum_{r=1}^{V_m}\left[\frac{1}{\sqrt{K_m}}\left\langle W_{rk,l}\right\rangle\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{r,m}} + O\left(K_m^{-1}\right)\right]\ . \qquad \text{(C.9)}\end{aligned}$$

Finally, from Eq. B.5, we have

$$\begin{aligned}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial \mu_{k,m}} &= \frac{\partial\left\langle v_{k,m}\right\rangle}{\partial \mu_{k,m}}\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial\left\langle v_{k,m}\right\rangle} \\ &= 2\mathcal{N}\left(0|\mu_{k,m},\sigma^2_{k,m}\right)\frac{\partial \ln P\left(\mathbf{v}_L = \mathbf{y}\right)}{\partial\left\langle v_{k,m}\right\rangle}\ . \qquad \text{(C.10)}\end{aligned}$$

Importantly, we can calculate Eqs. C.8-C.10 and C.6 in a single backward pass (for $l = L, \ldots, 1$), after a single forward propagation of probabilities (Eqs. B.5-B.7, for $l = 1, \ldots, L$), similarly to the BP equations (Eqs. 1.4-1.6 in [22, Eqs. 1.4-1.6]). This reduces the computational complexity of the algorithm to $O\left(S\left|\mathcal{W}\right|\right)$.

D Examples for weight restrictions

In this section we explain how to implement the EBP algorithm for: (1) Binary weights (2) Ternary weights (3) Real-valued weights. Note that, similarly to BP, the algorithm uses $O\left(\left|\mathcal{W}\right|\right)$ computation steps for each update of the posterior (which is the minimal amount of steps required for any algorithm that updates all the weights) in all these examples. This computational complexity is retained as long as the restriction sets ($S_{ij,l}$) are finite, and even in some cases when they are not finite (e.g., section D.3). For simplicity, we denote in this section $\nu_{k,l} = \left\langle v_{k,l}\right\rangle$.

D.1 Binary weights

Suppose $W_{ij,l}$ can assume only binary ± 1 values, so $S_{ij,l} = \{-1, 1\}$. For convenience, we will parametrize the distribution of $W_{ij,l}$ so that

$$P\left(W_{ij,l}|D_n\right) = \frac{e^{h^{(n)}_{ij,l}W_{ij,l}}}{e^{h^{(n)}_{ij,l}} + e^{-h^{(n)}_{ij,l}}}\ . \qquad \text{(D.1)}$$

In the forward pass (Eq. 4.1-4.2), we can use this parametrization to compute $\left\langle W_{ij,l}\right\rangle = \tanh\left(h_{ij,l}\right)$, $\left\langle W^2_{ij,l}\right\rangle = 1$ and $\mathrm{Var}\left(W_{ij,l}\right) = 1 - \tanh^2\left(h_{ij,l}\right) = \mathrm{sech}^2\left(h_{ij,l}\right)$. In the backward pass we substitute Eq. D.1 into Eq. 4.5, and find that the parameter $h^{(n)}_{ij,l}$ should be incremented each time according to

$$h^{(n)}_{ij,l} = h^{(n-1)}_{ij,l} + \frac{1}{\sqrt{K_l}}\Delta_{i,l}\nu_{j,l-1}. \qquad \text{(D.2)}$$

Finally, we note the MAP estimate (Eq. 4.6) of the weight configuration for the MNN is obtained by simple clipping

$$W^*_{ij,l} = \mathrm{sign}\left(h_{ij,l}\right)\ . \qquad \text{(D.3)}$$

D.2 Ternary weights

Suppose $S_{ij,l} = \{-1, 0, 1\}$. For convenience, we will parametrize the distribution of $W_{ij,l}$ so that

$$P\left(W_{ij,l}|D_n\right) = \frac{\exp\left(W_{ij,l}h^{(n)}_{ij,l} + \left(W^2_{ij,l} - 1\right)g^{(n)}_{ij,l}\right)}{e^{h^{(n)}_{ij,l}} + e^{-h^{(n)}_{ij,l}} + e^{-g^{(n)}_{ij,l}}} \qquad \text{(D.4)}$$

In the forward pass

$$\begin{aligned}\left\langle W_{ij,l}\right\rangle &= \frac{e^{h_{ij,l}^{(n)}}-e^{-h_{ij,l}^{(n)}}}{e^{h_{ij,l}^{(n)}}+e^{-h_{ij,l}^{(n)}}+e^{-g_{ij,l}^{(n)}}}\\ \left\langle W_{ij,l}^{2}\right\rangle &= \frac{e^{h_{ij,l}^{(n)}}+e^{-h_{ij,l}^{(n)}}}{e^{h_{ij,l}^{(n)}}+e^{-h_{ij,l}^{(n)}}+e^{-g_{ij,l}^{(n)}}}\,.\end{aligned}$$

In the backward pass, we substitute Eq. D.4 into Eq. 4.5, obtaining

$$\begin{aligned}h_{ij,l}^{(n)} &= \frac{1}{2}\ln\frac{P\left(W_{ij,l}=1|D_{n}\right)}{P\left(W_{ij,l}=-1|D_{n}\right)}\\ &= h_{ij,l}^{(n-1)}+\frac{1}{\sqrt{K_{l}}}\Delta_{i,l}\nu_{j,l-1}\,,\end{aligned}$$

as for the binary weights. Similarly,

$$\begin{aligned}g_{ij,l}^{(n)} &= \frac{1}{2}\ln\left[\frac{P\left(W_{ij,l}=1|D_{n}\right)P\left(W_{ij,l}=-1|D_{n}\right)}{P\left(W_{ij,l}=0|D_{n}\right)}\right]\\ &= g_{ij,l}^{(n-1)}+0\,.\end{aligned}$$

Therefore, the parameter $g_{ij,l}$ is not updated. In the end we choose the MAP weight configuration (using Eq. 4.6)

$$W_{ij,l}^{*}=\mathcal{I}\left\{|h_{ij,l}|>g_{ij,l}\right\}\operatorname{sign}\left(h_{ij,l}\right)\,.$$

Therefore, the initial conditions on $g_{ij,l}$ (*i.e.*, the prior) act as a threshold which generates sparse weights.

D.3 Real-valued weights

Suppose $S_{ij,l}=\mathbb{R}$, so $W_{ij,l}$ can receive any real value. A naive implementation of the algorithm would require an infinite number of updates, for each possible value of $W_{ij,l}$. A simple way to circumvent this is to as assume that each real weight can be written as an infinite sum of binary weights $W_{ij,l}^{\alpha}$

$$W_{ij,l}=\lim_{A\to\infty}\frac{1}{\sqrt{A}}\sum_{\alpha=1}^{A}W_{ij,l}^{\alpha}\,,$$

where each binary weight is parametrized as in D.1 with parameter $h_{ij,l}^{\alpha(n)}$, and we denote

$$h_{ij,l}^{(n)}=\lim_{A\to\infty}\frac{1}{\sqrt{A}}\sum_{\alpha=1}^{A}h_{ij,l}^{\alpha(n)}\,.$$

Using Eq. D.2, we obtain for each binary weight

$$h_{ij,l}^{\alpha(n)} = h_{ij,l}^{\alpha(n-1)}+\frac{1}{\sqrt{A}}\frac{1}{\sqrt{K_{l}}}\Delta_{i,l}\nu_{j,l-1}. \tag{D.5}$$

This immediately gives

$$h_{ij,l}^{(n)} = h_{ij,l}^{(n-1)}+\frac{1}{\sqrt{K_{l}}}\Delta_{i,l}\nu_{j,l-1}.$$

Assuming $h_{ij,l}^{\alpha(0)}\propto1/\sqrt{A}$, from Eq. D.5 we also have $h_{ij,l}^{\alpha(n)}\propto1/\sqrt{A}$. Therefore, after step n,

$$\begin{aligned}\left\langle W_{ij,l}\right\rangle &= \lim_{A\to\infty}\frac{1}{\sqrt{A}}\sum_{\alpha=1}^{A}\tanh\left(h_{ij,l}^{\alpha(n)}\right)\\ &= \lim_{A\to\infty}\frac{1}{\sqrt{A}}\sum_{\alpha=1}^{A}h_{ij,l}^{\alpha(n)}=h_{ij,l}^{(n)}\end{aligned}$$

and

$$\mathrm{Var}\left(W_{ij,l}\right) \quad = \quad \lim_{A\to\infty}\frac{1}{A}\sum_{\alpha=1}^{A}\left[1-\tanh^{2}\left(h_{ij,l}^{\alpha(n)}\right)\right]=1$$

Therefore, using CLT, we have after the n-th update

$$W_{ij,l}=\lim_{A\to\infty}\frac{1}{\sqrt{A}}\sum_{\alpha=1}^{A}W_{ij,l}^{\alpha}\sim\mathcal{N}\left(h_{ij,l}^{(n)},1\right)$$

And so, our MAP estimate of $W_{ij,l}$ is

$$W_{ij,l}^{*}=\mathrm{argmax}_{W_{ij,l}}P\left(W_{ij,l}|D_{n}\right)=h_{ij,l}^{(n)}\,. \tag{D.6}$$

E Numerical Experiments - additional details

E.1 Implementation

We tested the EBP algorithm in two special cases - a binary MNN (Algorithm 1) and a real MNN (Algorithm 2). In the first, after each update step, the MAP configuration for all the binary weights, is given by taking the sign of $h_{ij,l}$ (Eq. D.3), and the value of $h_{i0,l}$ is used for the biases (Eq. D.6). The EBP-D output is then obtained by substituting the MAP configuration into Eq. 2.2 and obtaining $y=v_L$. The EBP-P output is calculated using $\mathrm{y}=\mathrm{sign}\left(\langle v_L\rangle\right)$. The second MNN has real weights, and is trained by Algorithm 2. The MAP configuration is given by $h_{ij,l}$ itself for all the synaptic weights and biases (Eq. D.6). Again, in the EBP-D $y=v_L$ and in EBP-P the output is calculated using $\mathrm{y}=\mathrm{sign}\left(\langle v_L\rangle\right)$.

All algorithms were run using Matlab 2013b. Note that EBP on real MNNs (Algorithm 2) is very similar to BP, and hence should have similar running times - as was observed in practice. EBP on binary MNNs (Algorithm 1) performs additional non-linear operations on the weights ($\tanh\left(\cdot\right)$, or $\mathrm{sech}\left(\cdot\right)$), and therefore is expected to be somewhat slower. In practice, it was two times slower than BP if the values of the non-linear operation were saved and re-used, or five times slower if these values were not saved (to reduce memory requirements).

In all data sets we centralized (removed the means) and normalized the input (so $\mathrm{std}=1$), as recommended for BP [22]. Both in BP and EBP algorithms we used uniform initial conditions, with std=1, as recommended for BP [22], so $\sqrt{K_l/3}h_{ij,l}^{(0)}\sim\mathrm{U}\left[-1,1\right]$ for EBP, and similarly for $W_{ij,l}^{(0)}$ in BP. In BP we used an activation function $f\left(x\right)=1.7159\tanh\left(2x/3\right)$ for the hidden neurons, as recommended by [22]. If cross-entopy (CE) loss was used, then in the output neurons we used the relevant logistic activation functions $f\left(x\right)=\left(1+e^{-x}\right)^{-1}$ [6]. Parameter scans for the learning rate in BP was performed over

$$\left\{10^{-4},3\cdot10^{-4},5\cdot10^{-4},8\cdot10^{-4},10^{-3},3\cdot10^{-3},\right.$$
$$\left.5\cdot10^{-3},8\cdot10^{-3},10^{-2},3\cdot10^{-2},5\cdot10^{-2},8\cdot10^{-2},0.1\right\}$$

and in AROW we scanned the regularization parameter over $\left\{10^{k}\right\}_{k=-4}^{4}$ (the results were rather insensitive to changes in that parameter). The optimal parameters (which yield the best performance), given in Fig. E.1 are never near the edges of the scanned range.

E.2 Additional results

Small fan-in. To check whether the algorithm can work if the large fan-in assumption is incorrect, we also performed small-scale classification using the Pima Indians Diabetes dataset [2]. The set contains 768 instances with 8 features and 2 classes. The task was to identify the $\mathrm{label}\in\{-1,+1\}$, using a $8\to200\to1$ MNN classifier. Classification error was calculated using 10-fold cross validation, so we can compare with the previous best results reported in [1]. Results are shown in Table 2: as can be seen, EBP-P still exhibits the best performance with binary weights, and the second best performance with real weights.

Algorithm 1 A single update step of the the Expectation BackPropagation (EBP) algorithm for fully connected binary MNNs - with binary synaptic weights and real bias. We denote $\nu_{k,l} = \langle v_{k,l} \rangle$, $\tanh(h_{ij,l}) = \langle W_{ij,l} \rangle$, and $\mathcal{H}$ as the set of all $h_{ij,l}$.

```
Function $[\boldsymbol{\nu}_L, \mathcal{H}_{\text{next}}] = \text{UpdateStepBinaryMNN}(\mathbf{x}, \mathbf{y}, \mathcal{H})$
  % Forward pass
  Initialize
      $\forall k:\ \nu_{k,0} = x_k, \forall l:\ \nu_{0,l} = 1$
  for $m = 1$ to $L$ do
    $\forall k$:
      $\mu_{k,m} = \frac{1}{\sqrt{K_{m-1}}}\left[h_{k0,m} + \sum_{r=1}^{V_{m-1}} \tanh(h_{kr,m})\,\nu_{r,m-1}\right]$
      $\sigma_{k,m}^2 = \frac{1}{K_{m-1}}\left[1 + \sum_{r=1}^{V_{m-1}} \left[\left(1-\nu_{r,m-1}^2\right)(1-\delta_{1m}) + \nu_{r,m-1}^2 \operatorname{sech}^2(h_{kr,m})\right]\right]$
      $\nu_{k,m} = 2\Phi(\mu_{k,m}/\sigma_{k,m}) - 1$
  end for
  % Backward pass
  Initialize
      $\Delta_{i,L} = y_i \frac{\mathcal{N}\left(0|\mu_{i,L}, \sigma_{i,L}^2\right)}{\Phi(y_i \mu_{i,L}/\sigma_{i,L})}$
  for $l = L$ to $1$ do
      $\forall i:\ \Delta_{i,l-1} = \frac{2}{\sqrt{K_{l-1}}} \mathcal{N}\left(0|\mu_{i,l-1}, \sigma_{i,l-1}^2\right) \sum_{j=1}^{V_m} \tanh(h_{ji,l})\,\Delta_{j,l}$
      $\forall i,j:\ h_{ij,l}^{\text{next}} = h_{ij,l} + \frac{1}{\sqrt{K_{l-1}}} \Delta_{i,l} \nu_{j,l-1}$
  end for
```

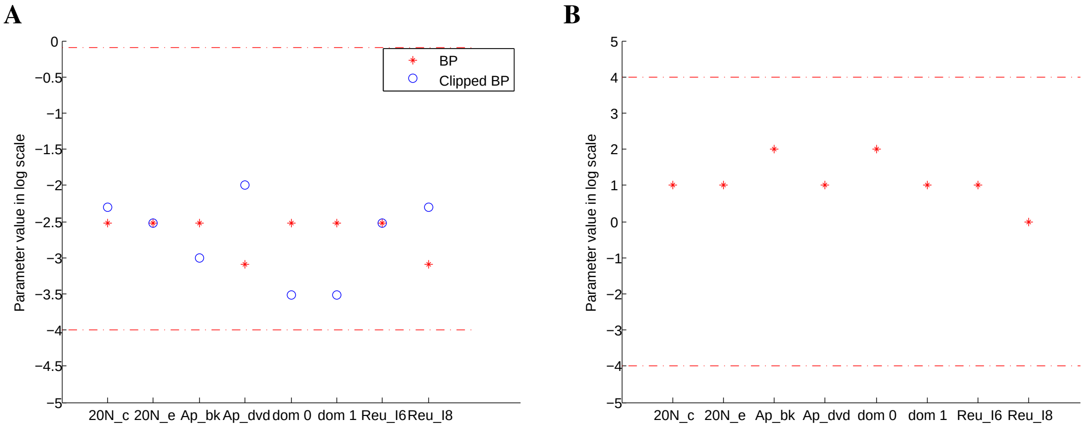

Figure E.1: Optimal parameter values for **(A)** BP and **(B)** AROW. Red lines - Minimum and maximal values of the scan.

Algorithm 2 A single update step of the algorithm for a fully connected MNNs with real weights and bias. We denote $\nu_{k,l} = \langle v_{k,l} \rangle$, $h_{ij,l} = \langle W_{ij,l} \rangle$, and $\mathcal{H}$ as the set of all $h_{ij,l}$.

```
Function $[\boldsymbol{\nu}_L, \mathcal{H}_{\text{next}}] = \text{UpdateStepReaMNN}\left(\mathbf{x}, \mathbf{y}, \mathcal{H}\right)$
  % Forward pass
  Initialize
      $\forall k:\ \nu_{k,0} = x_k, \forall l:\ \nu_{0,l} = 1$
  for $m = 1$ to $L$ do
    $\forall k$:
        $\mu_{k,m} = \frac{1}{\sqrt{K_{l-1}}} \sum_{r=0}^{V_{m-1}} h_{kr,m} \nu_{r,m-1}$
        $\sigma_{k,m}^2 = 1 + \frac{1}{K_{l-1}} \sum_{r=0}^{V_{m-1}} \left[ \left( \nu_{r,m-1}^2 - 1 \right) \delta_{m1} + \left( 1 - \delta_{m1} \right) \left( 1 - \nu_{r,m-1}^2 \right) h_{kr,m}^2 \right]$
        $\nu_{k,m} = 2\Phi\left( \mu_{k,m} / \sigma_{k,m} \right) - 1$
  end for
  % Backward pass
  Initialize
      $\Delta_{i,L} = y_i \frac{\mathcal{N}\left(0 | \mu_{i,L}, \sigma_{i,L}^2\right)}{\Phi\left(y_i \mu_{i,L} / \sigma_{i,L}\right)}$
  for $l = L$ to $1$ do
      $\forall i:\ \Delta_{i,l-1} = \frac{2}{\sqrt{K_{l-1}}} \mathcal{N}\left(0 | \mu_{i,l-1}, \sigma_{i,l-1}^2\right) \sum_{j=1}^{V_m} h_{ji,l} \Delta_{j,l}$
      $\forall i,j:\ h_{ij,l}^{\text{next}} = h_{ij,l} + \frac{1}{\sqrt{K_{l-1}}} \Delta_{i,l} \nu_{j,l-1}$
  end for
```

Dataset	Previous Best[1]	Real EBP-D	Real EBP-P	Binary EBP-D	Binary EBP-P	BP	Clipped BP
Pima Indians diabetes	22.3%	23.82%	22.11%	26.18%	**21.6%**	22.9%	34.9%

Table 2: Pima Indians Diabetes dataset - Test error.

Deeper architectures. To verify that the MNN's depth does not effect our conclusions, we test our algorithms using the same setup (except for more training epochs - eight instead of three) on a a deeper architecture of $M \to 1000 \to 100 \to 1$. The MNN was tested on three of the datasets. Results are described in Table 3. As can be seen, for these datasets EBP-P exhibits the best performance with binary weights, and the second best performance with real weights. This is the same as in the 2-layer case, except for the Reuters news I8 dataset - where before EBP performed better with real weights than with binary weights.

Dataset	Real EBP-D	Real EBP-P	Binary EBP-D	Binary EBP-P	BP	Clipped BP
Reuters news I8	15.7%	15.5%	21.5%	**15.25%**	17.2%	25.4%
20News group comp vs HW	5.06%	5.01%	6.2%	**4.39%**	8.26%	12.75%
Spam or ham d0	1.12%	0.88%	3.08%	**0.72%**	1.92%	10.54%

Table 3: Test error for a 3-layer MNN.

F Comparison with Backpropagation

The EBP algorithm for MNNs with real weights (summarized in Algorithm 2) is almost identical to the standard BP algorithm, when the variables $h_{ij,l}$ are interprets as the real-valued weights in a BP algorithm. To see this, recall [22] that in BP we wish to train a MNN of the form

$$\begin{aligned} \mathbf{u}_l &= \mathbf{W}_l \mathbf{v}_{l-1} \\ \mathbf{v}_l &= f(\mathbf{u}_l), \end{aligned}$$

$\forall l = 1, \ldots, L$, where $f(\cdot)$ is some sigmoid function. The training is done by minimizing an error function $E(\mathbf{y}, \mathbf{v}_L)$, through the following recursive equations. First, we initialize

$$\Delta_{i,L} = - \quad \eta \frac{\partial E(\mathbf{y}, f(\mathbf{u}_l))}{\partial u_{i,L,}}, \tag{F.1}$$

where E is some non-negative error function and η is a learning rate. Then, for $l = L, \ldots, 1$ and $\forall i, j$ we calculate

$$\Delta_{i,l-1} = f'(u_{i,l-1}) \sum_{j=1}^{V_m} W_{ji,l}^{(n-1)} \Delta_{j,l}. \tag{F.2}$$

$$W_{ij,l}^{(n)} = W_{ij,l}^{(n-1)} + \Delta_{i,l} v_{j,l-1}. \tag{F.3}$$

where f' is the derivative of f Comparing with EBP for a MNN with real-valued weights (summarized in Algorithm 2), we find that it is nearly identical. Specifically, in BP, we just need to substitute $W_{ij,l} = h_{ij,l}/\sqrt{K_l}$, $v_{k,m} = \langle v_{k,m} \rangle$, use $\eta = 1$, the activation function

$$f(u_{k,l}) = 2\Phi(u_{k,l}/\sigma_{i,l}) - 1$$

and the "cross-entropy" [6] error function

$$E(\mathbf{y}, \mathbf{v}_L) = -\ln P(\mathbf{v}_L = \mathbf{y}) = -\sum_i \ln\left(\frac{1 + y_i v_{i,L}}{2}\right).$$

The only difference is that the input u to each neuron is scaled adaptively through $\sigma_{i,l}$ - which depends on the inputs and weights (Eq. 4.2). This implies that the EBP algorithm is invariant to changes in the amplitude of of the input $\mathbf{x}$ (*i.e.*, $\mathbf{x} \to c\mathbf{x}$, where $c > 0$). This preserves the amplitude invariance of the sign activation function we used in the original MNN (Eq. 2.2). Note that in the standard BP algorithm the performance is directly affected by the amplitude of the input, so it is a recommended practice to re-scale it in pre-processing [22]. Interestingly, is also recommended practice to use the cross-entropy error function for classification tasks [6], and to scale initial conditions $W_{ij,l}^{(0)} \sim 1/\sqrt{K_l}$ [22]. These rather heuristic practices naturally arise in EBP, which was derived from first principles.

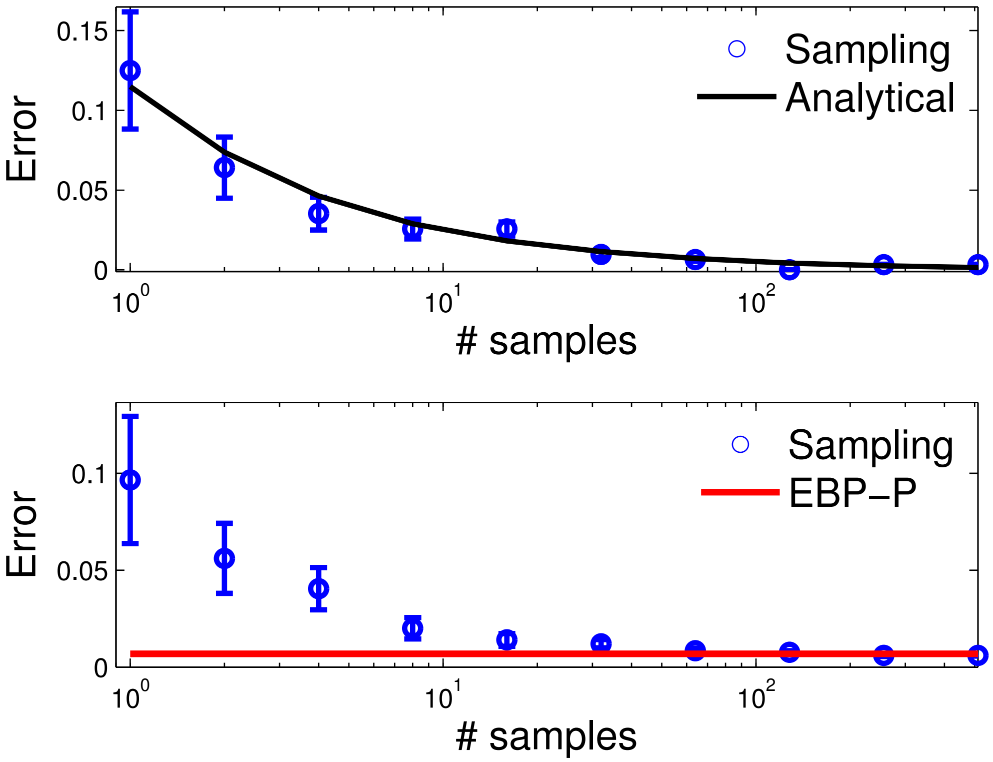

Figure G.1: Approximating EBP-P output by averaging the output of a random sample of MNNs with binary weights (#samples=$\left\{2^k\right\}_{k=0}^{9}$ shown). *Top figure:* $P\left(y_{\mathrm{EBP-P}} \neq y_{\mathrm{sampling}}\right)$ - sign error between the sampling output and analytical output. We show the analytically predicted error (Eq. G.2), as well as the empirical error from the sampling simulation. *Bottom figure:* The test classification error of the sampling output y_{sampling} in comparison to the same error of the analytical $y_{\mathrm{EBP-P}}$ output. Note that already for 16 samples (the fifth point), we get a comparable error. Error bars give the 95% confidence intervals.

G Sampling the weights

The EBP-P output (Eq. 4.7) is the MAP estimate of the MNN output (Eq. 3.2) which typically gives the best performance empirically (Table 1). In the paper we calculated the EBP-P output analytically (Eq. 4.7). Motivated by hardware applications, we would like, instead, to calculate the EBP-P output by averaging the output of several MNNs with binary weights. In this section we explain how this is done, and calculate analytically and numerically the approximation error for such MNNs with a single output neuron (*i.e.*, $V_L = 1$ - a binary classification task).

Originally the EBP-P output was derived from an ensemble average over the output of all such MNNs when the weights are distributed according to the posterior

$$P\left(\mathcal{W}|D_N\right) = \prod_{i,j,l} P\left(W_{ij,l}|D_N\right) \tag{G.1}$$

where $P\left(W_{ij,l}|D_N\right)$ is the weight distribution given by the algorithm (section 4) after training. In the binary output case, the classification according to the EBP-P output was peformed according to (Eq. 4.7)

$$\begin{aligned} y_{\mathrm{EBP-P}} &= \operatorname{sign}\left(\langle v_L\rangle\right) \\ &= \operatorname{sign}\left(\mu_L\right) , \end{aligned}$$

where either $\langle v_L\rangle$ or μ_L can calculated analytically (Eqs. 4.1-4.2) from the distribution over $\mathcal{W}$ (Eq. G.1). However, in order to implement this output in hardware using binary MNNs, we will use instead the following sampling-based procedure.

First, we generate S samples of the weights $\left\{\mathcal{W}^{(s)}\right\}_{s=1}^{S}$ by sampling from the inferred distribution (Eq. G.1). Then, we use each sample $\mathcal{W}^{(s)}$ to calculate the input to the last layer in the MNN (Eq. 2.2) for each example $\mathbf{x}$ in the test set

$$\forall s:\ u_L^{(s)} = \mathbf{W}_L^{(s)}\text{sign}\left(\mathbf{W}_{L-1}^{(s)}\text{sign}\left(\cdots\mathbf{W}_1^{(s)}\mathbf{x}\right)\right)\ .$$

Then, we perform the classification according to

$$y_{\text{sampling}} = \text{sign}\left(\frac{1}{S}\sum_{s=1}^{N}u_L^{(s)}\right)\ .$$

Due to the CLT approximation (Eq. 3.14), we have

$$\frac{1}{S}\sum_{s=1}^{N}u_L^{(s,n)} \sim \mathcal{N}\left(\mu_L, \frac{\sigma_L^2}{S}\right)\ .$$

Therefore, it is straightforward to calculate the following error probability

$$\begin{aligned} P\left(y_{\text{EBP}-\text{P}} \neq y_{\text{sampling}}\right) &= P\left(\text{sign}\left(\frac{1}{S}\sum_{s=1}^{N}u_L^{(s)}\right) \neq \text{sign}\left(\mu_L\right)\right) \\ &= \Phi\left(-\sqrt{S}\left|\frac{\mu_L}{\sigma_L}\right|\right) \\ &= \Phi\left(-\sqrt{S}\left|\Phi^{-1}\left(\frac{\langle v_L\rangle+1}{2}\right)\right|\right)\ . \end{aligned} \tag{G.2}$$

Therefore, asymptotically, the error will decay exponentially fast in S, since

$$\lim_{x\to\infty}\Phi\left(-x\right) \sim \frac{1}{x\sqrt{2\pi}}e^{-x^2/2}\ .$$

Next, we examine numerically this convergence speed, using the Spam or ham d0 dataset. Importantly, the above sampling-based method only assumes that the CLT theorem can be used to approximate the input to each neuronal layer. This is only approximately true for finite fan-in K. Empirically, we noticed that CLT was usually accurate - except in the input layer for a few examples. In these cases, the inputs were "heavy tailed" - so a single feature (an input component) was much stronger then all the others[8]. On these examples, the sampling output y_{sampling} might not converge to $y_{\text{EBP}-\text{P}}$, even if $S \to \infty$. To correct for this, we split any "strong" features which cause this issue (before any pre-processing). Specifically on this dataset (Spam or ham d0), we select any feature which contained examples deviating more than 140 standard deviations from the mean . Then we split that feature into five identical features with their original value divided by five. In total, this results in a modest 26% increase in the number of features (*i.e.*, the size of the input layer). As can be seen on Figure G.1, the sampled output quickly converges to the anaytical output of EBP-P.

References

[1] Http://www.is.umk.pl/projects/datasets.html.

[2] K Bache and M Lichman. UCI Machine Learning Repository. *http://archive.ics.uci.edu/ml*, 2013.

[3] C Baldassi, A Braunstein, N Brunel, and R Zecchina. Efficient supervised learning in networks with binary synapses. *PNAS*, 104(26):11079–84, 2007.

[4] D Barber and C M Bishop. Ensemble learning for multi-layer networks. In *Advances in Neural Information Processing Systems*, pages 395–401, 1998.

[5] R Battiti and G Tecchiolli. Training neural nets with the reactive tabu search. *IEEE transactions on neural networks*, 6(5):1185–200, 1995.

[6] C M Bishop. *Neural networks for pattern recognition*. 1995.

---

[8] This can happen, for example, due to common pre-processing procedure of normalizing features by their standard deviation. If the values of a certain feature are usually zero except in a few similar positive examples, then these positive values can drastically increase due to this type of normalization.

[7] C M Bishop. *Pattern recognition and machine learning*. Springer, Singapore, 2006.

[8] A Braunstein and R Zecchina. Learning by message passing in networks of discrete synapses. *Physical review letters*, 96(3), 2006.

[9] R P Brent and P Zimmermann. *Modern computer arithmetic*. Cambridge University Press, New York, 2011.

[10] B Cousins and S Vempala. A cubic algorithm for computing gaussian volume. *arXiv preprint arXiv:1306.5829*, 2013.

[11] K Crammer, A Kulesza, and M Dredze. Adaptive regularization of weight vectors. *Machine Learning*, 91(2):155–187, March 2013.

[12] G E Dahl, D Yu, L Deng, and A Acero. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. *Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[13] A Genz. Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing*, 14(3):251–260, 2004.

[14] A Graves. Practical variational inference for neural networks. *Advances in Neural Information Processing Systems*, pages 1–9, 2011.

[15] A J Hayter and Y Lin. The evaluation of two-sided orthant probabilities for a quadrivariate normal distribution. *Computational Statistics*, 27(3):459–471, June 2011.

[16] G E Hinton and D Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT '93*, 1993.

[17] G E Hinton, L Deng, D Yu, G E Dahl, A R Mohamed, N Jaitly, A Senior, V Vanhoucke, P Nguyen, T N Sainath, and B Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[18] K Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(1989):251–257, 1991.

[19] R Karakiewicz, R Genov, and G Cauwenberghs. 1.1 TMACS/mW Fine-Grained Stochastic Resonant Charge-Recycling Array Processor. *IEEE Sensors Journal*, 12(4):785–792, 2012.

[20] A Krizhevsky, I Sutskever, and G E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[21] Y LeCun and L Bottou. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Y LeCun, L Bottou, G B Orr, and K R Müller. Efficient Backprop. In G Montavon, G B Orr, and K-R Müller, editors, *Neural networks: Tricks of the Trade*. Springer, Heidelberg, 2nd edition, 2012.

[23] D J C MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 472(1):448–472, 1992.

[24] E Mayoraz and F Aviolat. Constructive training methods for feedforward neural networks with binary weights. *International journal of neural systems*, 7(2):149–66, 1996.

[25] T P Minka. Expectation Propagation for Approximate Bayesian Inference. *NIPS*, pages 362–369, 2001.

[26] P Moerland and E Fiesler. Neural Network Adaptations to Hardware Implementations. In *Handbook of neural computation*. Oxford University Press, New York, 1997.

[27] R M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

[28] F Ribeiro and M Opper. Expectation propagation with factorizing distributions: a Gaussian approximation and performance results for simple models. *Neural computation*, 23(4):1047–69, April 2011.

[29] D Saad and E Marom. Training Feed Forward Nets with Binary Weights Via a Modified CHIR Algorithm. *Complex Systems*, 4:573–586, 1990.

[30] S A Solla and O Winther. Optimal perceptron learning: an online Bayesian approach. In *On-Line Learning in Neural Networks*. Cambridge University Press, Cambridge, 1998.

[31] N Srivastava and G E Hinton. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning*, 15:1929–1958, 2014.

[32] H Y Xiong, Y Barash, and B J Frey. Bayesian prediction of tissue-regulated splicing using RNA sequence and cellular context. *Bioinformatics (Oxford, England)*, 27(18):2554–62, October 2011.