# Bayesian Statistics as an Alternative to Gradient Descent in Sequence Learning

R. Spiegel[1,2,3]
[1] Ludwig-Maximilians-Universität, Sensory-Motor Learning Lab, Institut für Med. Psychologie, München, Germany
[2] Generation Research Ltd., Bad Tölz, Germany
[3] University of Cambridge, Wolfson College, Cambridge, United Kingdom

***Abstract*—Recurrent neural networks are frequently applied to simulate sequence learning applications such as language processing, sensory-motor learning, etc. For this purpose, they often apply a truncated gradient descent (=error correcting) learning algorithm. In order to converge to a solution that is congruent with a target set of sequences, many iterations of sequence presentations and weight adjustments are typically needed. Moreover, there is no guarantee of finding the global minimum of error in a multidimensional error landscape resulting from the discrepancy between target values and the network's prediction. This paper presents a new approach of inferring the global error minimum right from the start. It further applies this information to reverse-engineer the weights. As a consequence, learning is speeded-up tremendously, whilst computationally-expensive iterative training trials can be skipped. Technology applications in established and emerging industries will be discussed.**

***Index Terms*—Gaussian processes, Error-correction, Bayes theorem, Sequential learning, Recurrent neural networks.**

## I. INTRODUCTION

This article has several aims: First, it will be shown that the output produced by recurrent neural networks relying on gradient descent can be predicted by applying Bayes theorem. In the past, this was demonstrated when taking a localist coding scheme to represent input and target values [1]. In a localist coding scheme, only one unit is active, whilst all other units are inactive, e.g. (1, 0, 0, 0, 0, 0, 0, …, 0). Now it will be shown that Bayes theorem can just as well predict the output after using a distributed coding scheme. In a distributed coding scheme, more than one unit is active, e.g. (1, 0, 1, 0, 0, 0, 1). This not only applies to distributed coding schemes with binary numbers, but also to those with continuous numbers, e.g. (0.9, 0.1, 0.3, 0.1, 0.5, 0.8).

Second, a statistical approach can be used to infer the global minimum of error in the multidimensional error landscape. If there is a discrepancy between the output value that the network predicts and the target value, an error is computed. In the optimal case, the discrepancy between predicted output and target should be zero, which would yield an error of zero. For trivial tasks this may work, but one might want to skip a neural network altogether if the task is so trivial that a target can be predicted easily. For more complex tasks (such as real world scenarios), an error of zero is unlikely. To illustrate this, consider the following toy example (which is trivial as well, but does not yield an error of zero). You have two sequences. In one case, the value of 1 predicts the value of 1, in the other case the value of 1 predicts the value of 0. Both sequences are equally likely. Consequently, you have two different target values (1 and 0). If you iteratively train a neural network on this problem, it would probably settle on a solution where 1 predicts the value 0.5 (because there is an equal amount of training examples where 1 predicts 0 and those where 1 predicts 1). One could also say that the network interpolates. Here, the resulting error is not zero, because the network's prediction of 0.5 neither corresponds to the target of 0, nor to the target of 1. Now consider that you have a multilayer network with a large number of input units, hidden units and output units. As a consequence, there are large weight matrices interconnecting the layers. Each time a prediction is contrasted with the target, the discrepancy between prediction and target is used to adjust the weight matrices. Hence, there is not just one error, but an error for each discrepancy on each unit. As training progresses, the errors on the individual units change. Hence, a multidimensional error landscape results from training a neural network by iteratively adjusting its multidimensional weight matrix. The Backpropagation algorithm that is typically applied to these networks changes the weights in such a way that an error is computed for every output unit, and the weights connected to this unit are changed so that the error is reduced. Because the error landscape is multidimensional, however, it is not clear whether this weight change will actually point in the direction of the global error minimum. Hence, backpropagation poses the danger of dipping from one local minimum into another (or even getting stuck in a local minimum) without ever finding the global minimum of error. This paper will describe an approach to determine the global minimum of error.

Third, it will be shown that the information of knowing the global error minimum is actually sufficient to replace the training of the network (and its iterative weight adjustments) by inferring the weight matrix right from the start. Whilst this causes no problem for feedforward 2-layer networks or multilayer perceptrons, the situation becomes more difficult when using recurrent networks in sequence learning applications. Nevertheless, a solution will be discussed for recurrent neural networks as well.

Fourth, real-world and laboratory-based applications will be discussed in the light of speeding-up the learning process by inferring rather than by training weights. Finally, it will be shown that the usual strength of neural

networks –which is generalizing to novel datasets- is not sacrificed when replacing training by inference.

The approach described in this paper is one way of relating Bayesian statistics to neural networks. This is by far not the only way to make use of Bayes theorem in this context. The way it is applied in this paper, however, seems to have a number of practical benefits that are, to the best of my knowledge, still unknown. David MacKay and Radford Neal provide excellent summaries of the long tradition to relate Bayes theorem to neural networks or to nonlinear parametric models such multilayer perceptrons, or to classification and regression problems [2]-[4]. Prior to going into further details about Bayesian statistics and neural networks, I will give a brief introduction to Bayes theorem and the simple recurrent network. My introduction to Bayes theorem is based on [1] and [5], whilst my summary of the simple recurrent network is based on [1] and [6].

Considering a space of hypotheses $H$, one often aims to find the most probable hypothesis given the observed training data $D$ and given the knowledge of the prior probabilities of hypotheses in $H$ [5]. Referring to the terminology of neural networks, the training data $D$ are usually training examples of a target function and $H$ is the space of target functions. Bayes theorem can compute the posterior probability of a particular hypothesis $h$, $P(h|D)$. This is the probability that hypothesis $h$ holds given the observed training set $D$. This probability is dependent on priors: The prior probability of hypothesis $h$, $P(h)$, as well as the prior probability that training data $D$ will be observed, $P(D)$. This prior probability $P(D)$ does not incorporate any knowledge about which hypothesis $h$ holds. To compute the posterior probability $P(h|D)$, it is further necessary to know the probability that training data $D$ are observed given a situation in which hypothesis $h$ holds. This probability is expressed as $P(D|h)$. Combining these probabilities in Bayes theorem allows to calculate the posterior probability $P(h|D)$ [1], [5]:

$$P(h \mid D) = \frac{P(D \mid h) \quad P(h)}{P(D)} \qquad (1)$$

It is often necessary to find the maximally probable hypothesis $h \in H$ given the training set $D$ or several maximally probable hypotheses if there are two or more hypotheses with equal probabilities. Maximally probable hypotheses are called maximum a posteriori hypotheses. When all of the hypotheses $h \in H$ are equally probable a priori, it is possible to simplify and skip $P(h)$ to consider the hypothesis that maximizes $P(D|h)$ only. This hypothesis is termed maximum likelihood hypothesis. Tom Mitchell [5] was able to show that particular learning algorithms (e.g. error correcting learning algorithms in neural networks, linear regression and polynomial curve fitting) will output maximum a posteriori and maximum likelihood hypotheses: "Bayesian analysis can sometimes be used to show that a particular learning algorithm outputs MAP hypotheses even though it [the algorithm] may not explicitly use Bayes rule or calculate probabilities in any form … a straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis. The significance of this result is that it provides a Bayesian justification … for many neural network … methods…" (p. 164). These methods include the simple recurrent network (SRN). Having summarized Tom Mitchell's earlier work, I now refer to my research on the recurrent network. I will start with a description of the SRN, [6]. The purpose of the SRN is to learn sequences. The SRN receives input from the input units and is trained to predict the next step of the sequence at the output level (= next input being represented as target). The SRN has recurrent (=copy-back) connections from the hidden units to an extra layer of context units. These context units store exact copies of the hidden units, i.e. at the next step in the sequence, they feed the hidden units with the hidden units' activities from one time step ago. So at the following time step, the hidden units have input from the input units as well as from the context units. The context units provide the network with a dynamic memory, because each step in the sequence they will have a different activation and therefore different representation (resulting from all the previous steps in the sequence). Depending on the sequence position, the same inputs can therefore result in alternative predictions of the network. The SRN is trained with the previously mentioned backpropagation learning algorithm. It is displayed in Figure 1.

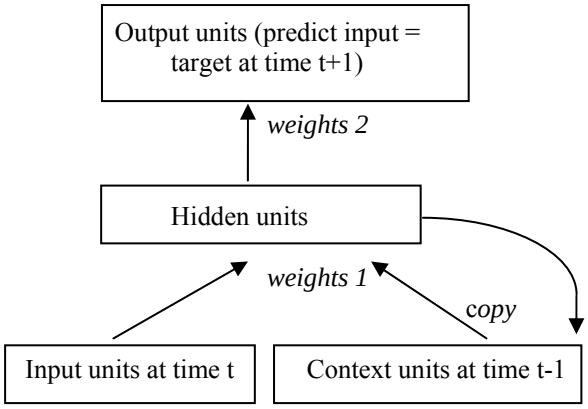


Fig. 1: The simple recurrent network.

Having provided a brief overview with respect to Bayes theorem, how it can be related to neural networks and the simple recurrent network, I will now refer to other approaches where Bayes theorem was discussed with regard to neural networks. Although the SRN is a recurrent network (with copy-back connections from the hidden to the context layer), its main learning principles are somewhat reminiscent of feedforward networks, where activity is fed in one direction (and only error is fed backwards in order to adjust the weights). Following MacKay [2], a feedforward network can be interpreted in terms of a prior probability over nonlinear functions. In addition, the network's learning process can be viewed as the posterior probability distribution over the unknown function. These approaches do not make any direct use of the global minimum of error, i.e. they do not apply this information to reverse-engineer the weights (this is how they differ from the approach that is explained in this paper). Rather, they use other methods to estimate the

weight matrix: The two main approaches are David MacKay's Gaussian approximation method [3] and Radford Neal's Hamiltonian Monte Carlo method to neural networks [4], which is also considered by David MacKay [2]. Because a detailed summary of these approaches already exists in the literature [2], only a brief review will be given here.

I will start with the Hamiltonian Monte Carlo method, and its sister version named Langevin Monte Carlo method. It has to be kept in mind that this approach does not replace but modify gradient descent. MacKay [2] therefore summarizes it as "gradient descent with added noise." As mentioned before, its main aim is to estimate rather than to reverse-engineer the weights. Similar to backpropagation, a gradient is computed and the weights are modified based on this gradient. Therefore, the multidimensional error landscape also exists for this method, and the way each weight is modified is based on similar principles as the earlier described gradient-descent operation. The way it differs from backpropagation, however, is that a noise vector is added. This noise vector is generated from a Gaussian. Subsequently, samples of the weight matrix are generated and a Monte Carlo approximation to the Bayesian predictions is obtained by averaging together the functions that had resulted from these samples. The result of this approach was that Bayesian predictions found by the Langevin Monte Carlo method were better than those predictions using optimized parameters [2]. Langevin Monte Carlo's big brother, the Hamiltonian Monte Carlo, was further able to reduce random walk in the multidimensional error landscape, because it makes use of multiple gradient evaluations at the same time.

The Gaussian approximation method [2]-[3] will be considered next. Unlike backpropagation, Hamiltonian or Langevin Monte Carlo, this approach does not make use of gradient descent anymore. Its use of Bayesian processes also differs from the way Bayesian predictions are obtained in the Monte Carlo method. Rather, the Gaussian approximation method aims at estimating the most probable weights. It could also be expressed in the following way: Let us assume the network tries to predict the target. Now these are the weights to yield the output closest to the target, or better expressed: These are the most probable weights to yield this output. The question arises how these weights can be estimated. In the Gaussian Approximation method, an approximation to the posterior probability is made, where a locally Gaussian posterior probability distribution over each weight value is assumed. Under this assumption the weights are Gaussian-distributed, with mean $\mathbf{w}_{MP}$ (=weight with the highest probability) and variance-covariance matrix $\mathbf{A}^{-1}$. It could be shown that the maximally probable output resulting from the maximally probable weights and input values is also normally distributed [2]. Since the maximally probable output is the one closest to the target and since it can be inferred from the mean of the Gaussian, and because the output values are a function of the weights, it is possible to compute the maximally probable weights. Further research on Gaussian processes in relation to neural networks can be found in [7]-[10].

Additional ways of using Bayes theorem in conjunction with neural networks include the optimal network size (e.g. networks with too many hidden units may generalize poorly to new datasets). Ideas on Bayesian optimization can be found in [2]. For more work on Bayes theorem and neural networks see [11]-[14], for Bayes theorem and recurrent neural networks in particular see [15]-[16].

## II. BAYES THEOREM AND THE SRN

As summarized in the introduction, it had already been shown that the output produced by recurrent neural networks relying on gradient descent can be predicted by Bayes theorem [1]. In this previous publication, this was shown for applying a localist coding scheme of input and target units (i.e. only one unit is activated at any one time during training). Before demonstrating that this finding can be extended to a distributed coding scheme as well, a brief summary of these previous results will be given. Only a toy problem will be described here, because the full details can be found in [1]. Consider the two sequences ABC and ABB. Now consider that A is coded (1, 0, 0), B is coded (0, 1, 0) and C is coded (0, 0, 1). Because the neural network cannot deal with strictly binary representations [17], the localist coding scheme was set to (0.9, 0.1, 0.1), (0.1, 0.9, 0.1) and (0.1, 0.1, 0.9) respectively. These adjustments are common practice among neural modelers [18]. After being trained on the sequences ABC and ABB problem, the network predicts (0.1, 0.9, 0.1) for the 2nd sequential step, because in 100 percent of the training examples, the letter B follows the letter A. With regard to the 3rd step, however, the letters B and C are equally likely (they both occur in 50 percent of the training examples). Therefore, the network predicts something close to (0.1, 0.5, 0.5). These predictions are congruent with a Bayesian interpretation. Now imagine the network is trained on ABA, ABB and ABC and all three sequences are equally likely to occur during training. In this case, the network predicts something close to (0.3666, 0.3666, 0.3666). It does not predict 0.3333, because the target values of 0.9, 0.1 and 0.1 add up to 1.1 (1.1 divided by 3 equals 0.3666, whilst the perfect binary representation of 1, 0, 0 would add up to 1; only 1 divided by 3 equals 0.3333). Nevertheless, these predictions are still congruent with a Bayesian interpretation, which will be explained below.

Considering a Bayesian interpretation, the version space $|VS_{H,D}|$ plays a central role. This is the set of hypotheses from the hypothesis space $H$ that are consistent with the training set $D$. Tom Mitchell writes (p. 162), [5]: "As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to one is shared equally among the remaining consistent hypotheses. The above analysis implies that … every consistent hypothesis has posterior probability $(1/|VS_{H,D}|)$, and every inconsistent hypothesis has posterior probability of 0." With respect to the neural network and its continuous target values of 0.9 and 0.1 in our example, the total probability would not sum to 1, but to 1.1 (a probability is defined by the borders 0 and 1, but since backpropagation only approximates probabilities, an exception is made). Everything else, however, would remain equal to Mitchell's statement on equivalence between Bayesian learning and neural network learning. The following equation is a formal way of expressing the citation on the version space $|VS_{H,D}|$, the prior probability that training data $D$ will be observed, and the prior probabilities of hypothesis $h$ for all hypotheses in the hypothesis space $H$: $P(h)$. Since it can be assumed that the probabilities of each

individual hypothesis accumulate to 1, the prior probability of an individual hypothesis $P(h)$ can be expressed as $1/|H|$. $P(D)$ can be written $|VS_{H,D}|/|H|$, which follows from the theorem of total probability (see Mitchell [5], p. 161). $P(D|h)$ is either zero or one, depending on whether $h$ is consistent or inconsistent with the data in the training set. If it is inconsistent, the numerator of the posterior probability $P(h|D)$ will be multiplied with zero, which gives a posterior probability of zero. If it is consistent with the training set, $P(h|D)$ is calculated as follows (see Mitchell [5], p. 161):

$$P(h \mid D) = \frac{P(D \mid h) \quad P(h)}{P(D)}$$

$$P(h \mid D) = \frac{1 * \frac{1}{|\mathrm{H}|}}{P(D)} \qquad (2)$$

$$= \frac{1 * \frac{1}{|\mathrm{H}|}}{\frac{|VS_{\mathrm{H,D}}|}{|\mathrm{H}|}}$$

$$= \frac{1}{|VS_{\mathrm{H,D}}|}$$

It will now be explained how these equations can be related to a neural network. In the network, $\mathbf{o_i}$ denotes the activity of output unit i and $\boldsymbol{\varepsilon_i}$ denotes a noise term for output unit i. The equivalent of the hypothesis space $H$ from Bayes Theorem is the number of possible competing hypotheses $C$ in the output vector $\mathbf{o}$ (in the localist coding scheme this equals the number of vector components). Thus, one can write $1/|C|$ for the neural network wherever one writes $1/|H|$ in Bayes Theorem. The network's equivalent of the Version Space $|VS_{H,D}|$ (symbolizing the set of hypotheses from the hypothesis space $H$ being consistent with the training data $D$) is the network's Version Space $|VS_{C,T}|$ (in the localist coding scheme this symbolizes the number of active target vector components $C$ from the training set $T$ that are presented to the network at a particular step in the sequence, e.g. if the network was trained on the sequences ABC and ABB, there would be a "competition" between 2 different active vector components at the third step in the sequence, (0, 0, 1), (0, 1, 0)). The equivalent of inconsistent hypotheses would be hypotheses that are not represented in the training set, in this example the vector (1, 0, 0) never appears at the third step in the sequence. Consequently, the third vector component would reveal a value of zero, because the network's equivalent of Bayes Theorem's $P(D|h)$, i.e. $P(T|c)$ would be zero. For the sequential steps consistent with the training set, $P(T|c)$ would reveal a value of 1, and this example applies in Equation 3. After comparing Equations 2 and 3, it becomes evident that both will output equivalent results. The posterior probabilities in Equation 2 might not exactly match the output activities in Equation 3, but this would not change the network's Bayesian interpretation. Equation 2 would exactly match the transitional probabilities (i.e. if there is one possible sequential element to follow, it will be predicted with a probability of 1, if there are two/three/four possible elements to follow, they will be predicted with a probability of 0.5/0.33/0.25). The result of Equation 3 will not match these transitional probabilities exactly, because there is a noise term and no strict binary values can be applied. So if two different sequential elements are to follow at the same position, there are the values 0.9 and 0.1 (they add up to 1, so both elements are predicted with a probability of 0.5). However, if there are three different sequential elements to follow at the same position, there are the values 0.9, 0.1, 0.1 adding up to 1.1 (resulting in an output activity of 0.3666 instead of 0.3333), if there are four different sequential elements, the values 0.9, 0.1, 0.1, 0.1 add up to 1.2 (resulting in an activity of 0.3 instead of 0.25).

$$\mathbf{o}_i = \frac{P(T \mid c) * \frac{1}{|\mathrm{C}|}}{\frac{|VS_{\mathrm{C,T}}|}{|\mathrm{C}|}} \pm \boldsymbol{\varepsilon}_i \qquad (3)$$

$$= \frac{1 * \frac{1}{|\mathrm{C}|}}{\frac{|VS_{\mathrm{C,T}}|}{|\mathrm{C}|}} \pm \boldsymbol{\varepsilon}_i$$

$$= \frac{1}{|VS_{\mathrm{C,T}}|} \pm \boldsymbol{\varepsilon}_i$$

Taking this aspect into account, however, the network nevertheless reveals its equivalence to MAP-hypotheses. Moreover, it would be possible to make a correction to the network's output. Having summarized the equivalence of Bayesian learning and the learning of a neural network with a localist coding scheme, I will now refer to new simulations with a distributed coding scheme. In a distributed coding scheme, input and target vectors take on activities such as (0, 1, 1, 0, 1, 1, 1) or (0.1, 0.9, 0.9, 0.1, 0.9, 0.9, 0.9), e.g. there is more than one unit active per input / target vector. Simulations have shown that the output activities of the neural network can still be predicted by relying on the previously mentioned ideas on Bayes theorem [19]. This principle can be further extended, e.g. when assigning continuous values to the target vectors, e.g. (0.3, 0.8, 0.7, 0.5, 0.2, 0.2, 0.8). Moreover, one could imagine that the same input is associated with different target vectors, but that each target vector has a different probability to be associated with this input, e.g. take the toy problem where you have the sequences ABA, ABB and ABC. Now imagine that the sequential steps AB have already been presented to the network. Then imagine that the target vector for A is presented in only 5 percent of the training examples, the target vector for B in 80 percent of the cases and the target vector for C in the remaining 15 percent of training examples. For demonstration purposes, I will again make use of a toy problem: Assume letter A has the coding (0.7, 0.3, 0.4), letter B the coding (0.1, 0.8, 0.9), letter C the

coding (0.4, 0.1, 0.6). Once the sequence AB has occurred and the task will be to predict the next letter, the output based on statistics would be 0.05*0.7+0.8*0.1+0.15*0.4 for vector component 1 (the result for the 1st vector component thus equals 0.175). Were all three sequences equally likely to occur during training, the result for the 1st vector component would have been (0.7+0.1+0.4) / 3 = 0.4. The output of the other two vector components is computed in an equivalent manner. The 2nd vector component yields an output of 0.67 in the case where there is an unequal amount of training for all three sequences and 0.4 in the case where there is an equal amount of training for all three sequences. The 3rd vector component yields 0.83 for the unequal amount of training and 0.63 for the equal amount of training. Running simulations on the simple recurrent network showed that these predictions (which are purely due to statistical regularities) were approximated very closely [19]. As a result, it will be possible to predict the output of the neural network even before it is trained on this problem. This can be done because the network performs something like a regression-task, where the discrepancy between output values and target values becomes minimal. Certainly the network often approximates the values predicted by the statistical model without reaching them exactly (because there is no guarantee that the global minimum of error is reached). Nevertheless, when many simulations were run and averaged over, the approximation of the network was reasonably close to what had been predicted by statistics. Going back to the Bayesian statistics method and examining the discrepancy between the target values and the output values in the neural network, it became clear that the Bayesian predictions reached the global minimum of error [19], whilst the network was never able to reach a lower summed squared error (an explanation on how to compute this error value is given in [17] and [20]). As a consequence, the information inherent in statistics alone will be sufficient to infer the best possible output of the network after training (i.e. the output where the global minimum of error is reached). Having found a way of reaching the global minimum of error right away without consuming resources for training and without the danger of dipping into a local minimum, one might ask whether it is possible to replace training completely. A similar suggestion was already made by David MacKay [2], who originally applied different methods, but came to the same conclusion after studying feedforward networks (i.e. networks that were not applied to sequential learning). Because the simple recurrent network has a lot of analogies to feedforward backpropagation networks (with the only difference of having recurrent links from hidden to context units), it is reassuring that the results and conclusions overlapped.

## III. REPLACING TRAINING

Because training consists of gradual weight changes based on the discrepancy between target values and output activities, it makes sense to assume that training can be replaced when knowing the minimal discrepancy. In this case, gradual weight changes in the direction of a local or the global minimum might seem superfluous. If the neural network does not have many units, it is straightforward to infer a set of weights where the global minimum of error is reached. If the network size is large, it might be more complicated to infer a set of weights. In any case, there is more than just one way to infer the weights, so different possibilities will be demonstrated. The first example deals with the common statistical techniques Multiple Regression and Logistic Regression, e.g. [21]-[24]. In Multiple Regression, the criterion variable $\hat{Y}$ is estimated by a linear function of the observed values X times the regression coefficients $w_j$ and an added constant $w_0$. Its general form is displayed in Equation 4.

$$\hat{Y} = w_0 + \sum_{j=1}^{J} w_1 X_1 + w_2 X_2 + \ldots w_j X_j \quad (4)$$

This function permits to determine the regression line, which has the minimal discrepancy from the observed values. In the neural network, this is analogous to a function where the global minimum of error becomes minimal. In other words, it is the bias value $w_0$ and the combination of weight values $w_j$ where the discrepancy between target values and predicted output values is minimal. Whilst multiple regression estimates the criterion variable $\hat{Y}$, Bayesian statistics from our example permits to compute the output values where the discrepancy becomes minimal. Consequently, the computed output values would reflect those values that lie on the regression line in Multiple Regression.

Given that the predicted output values in the earlier mentioned backpropagation algorithm are computed by applying the (nonlinear) logistic activation function, the statistical technique Logistic Regression probably shows a better analogy to the network's algorithm. Here we replace the estimated criterion variable $\hat{Y}$ with the logit z, as displayed in Equation 5.

$$z = w_0 + \sum_{j=1}^{J} w_1 X_1 + w_2 X_2 + \ldots w_j X_j \quad (5)$$

Furthermore, we enter the logit z into the logistic function L, containing Euler's number (e=2.718), as displayed in Equation 6.

$$L = \frac{1}{1 + e^{-z}} \quad (6)$$

Consequently, the output is bounded between 0 and 1, making it more analogous to the output in neural networks that apply backpropagation as their algorithm. Taking the examples of Multiple Regression and Logistic Regression, it is easy to see how to reverse-engineer the weights in a neural network such as the simple recurrent network (see Figure 1). The output activities associated with the global minimum of error are known after computing them with the previously described Bayesian statistics method. The input activities are given as well. The weight set from the input to the hidden units is initially set to random values (e.g. between -0.5 and 0.5). This will give a set of activities on the hidden units. Each

step in the sequence, the activities on the hidden units are fed back to the context units, which provide the hidden units with input on the following sequential step. In the beginning, the weights from context to hidden units are also initialized with random values (e.g. once more between -0.5 and 0.5). Due to the copy-back mechanism, the hidden units have different activity values for each step in the sequence. Consequently, every sequential step is associated with a unique representation on the hidden units, i.e. with different activities on the hidden units. Given that we now know these values on the hidden units for each sequential step, and given that we know the output values where the global minimum of error is reached for each step in the sequence, we are able to linearly combine the weight values between the hidden and output layer. They need to be combined in such a way that those output activities are met where the global minimum of error is reached. The biases on the output units can also help in this regard. In the traditional version of backpropagation and the SRN, there is exactly one bias per output unit. An alternative way of dealing with this problem would be to have one bias per sequential step. This would increase the network's values, however, though it would make it even easier to find a combination of weights and biases where the output values with the global minimum of error are reached. It might be asked what can be done if it turns out impossible to find a combination of weights and biases that exactly reach the needed output representation, e.g. in case there are many similar and long sequences. One way would be to increase the network's size by raising the number of hidden units (which would also make the weight matrices bigger). What could also be done is to predetermine the hidden units' activity values in order to make sure that they are different enough from each other at every step in the sequence. Next, a combination of weights between the chosen hidden units' activities and the required output units' activities can be computed. What follows then is to infer the weight values between input units and hidden units and context units and hidden units. Because the hidden units' activities were predetermined, the same holds true for the context units, because they are the hidden units' activities from the previous time step (see Figure 1). The input units' activities are given, so what needs to be done next is to compute the weight matrix from input and context to hidden in much the same way as has been done before (when applying linear algebra to compute the weight matrix from hidden to output). In the feedforward network, it is also possible to infer the weight matrices with David MacKay's Gaussian processes approach [2]. In principle, this approach could also be extended to the simple recurrent network (the exact method will not be part of this paper, though). The following section will discuss practical implications of replacing training by inferring instant weight and bias values.

## IV. PRACTICAL APPLICATIONS

It is worth considering the consequences of replacing training with its gradual weight changes and no guarantee to find the global minimum of error by inferring the global minimum of error and reverse-engineering the weights. First, a tremendous amount of time and processing power could be saved. This is vital for tasks that require a lot of time and processing power due to their complexity, e.g. biomedical and bioinformatics applications of protein folding [25]-[29]. Moreover, it could be applied to forecasting operations, e.g. with respect to energy load [30] or environmental matters [31], language [32]-[34] or sign-language learning [35]-[41], spatio-temporal operations such as pointing [42], spatial response [43]-[47] or sound visualization [48]. In principle, it can be extended to any sequence learning problem where recurrent neural networks have been applied. Given that it is also possible to infer weights for large networks that require an extensive amount of units/nodes and even millions of training trials, the significance of this technique is that cost could be saved and that more complex problems can be solved with the same amount of cost or technical equipment.

## V. CONCLUSIONS

Considering arguments in favor of neural networks and gradient descent, it has often been mentioned that they are able to generalize to novel input, e.g. that the gradual change of weights has resulted in an inner representation of the network that would it permit to deal with novel input [17], [49]-[51]. One example of generalization is the following sequence learning problem [45], [47]: the network is trained on the sequences ABCBA, ABCCCBA, ABCCCCBA, ABCCCCCCBA, ABBCBBA, ABBCCCBBA, ABBCCCCBBA and ABBCCCCCCBBA. In other words, the number of intervening C elements during training is 1, 3, 4 and 6. The number of B elements appearing before or after the C elements is either 1 or 2. If it is 1 B before, there will be 1 B after the C elements. If there are 2 Bs before, there will be 2 Bs after the C elements. If the network is given a new sequence that has not been experienced during training, e.g. with 2 or 5 C elements, it is still able to predict the next sequential element once the first B has appeared after the Cs. Although this or a related type of generalization is typically considered a strength of neural networks implementing gradient descent [45], [47], [49]-[51], there is no reason to believe that it is due to gradient descent itself. Inferring a set of weights that optimally fits the training data by reaching the global minimum of error will result in a similar or even better inner representation of the training set. Consequently, gradient descent alone does no magic by finding a combination of weights that represent the training data. There are a range of statistical methods that can infer a combination of weights representing the training set. As it was pointed out by proponents of the neural network approach [17], [49]-[51], all the knowledge lies in the set of weights and biases. If it is possible to find a combination of weights and biases where the error between the target units and the actual activations becomes sufficiently small, the network has also reached an inner representation that it can benefit from to generalize to new datasets [17], [49]-[51]. The Bayesian statistics described in this paper do something analogous to the neural network approach. They make use of the same type of weight and bias matrices, but rather than gradually adjusting the weights and biases to minimize error by gradient descent, they infer those weights and biases where error becomes minimal. So this is where the only difference lies: Instead of slowly adjusting weights/biases and consuming a lot of

technological resources to find a weight/bias set that cannot even guarantee to reach the global minimum of error, it rapidly infers a weight/bias set that guarantees the global minimum of error. The rest remains equal to the neural network approach, i.e. the ability to generalize stays preserved due to having achieved weights/biases that are associated with the minimal error. Moreover, this Bayesian statistics approach has the additional advantage of saving a significant amount of cost, which is vital for biotechnological or energy-saving data sets, where the number of units could become very large.

## ACKNOWLEDGMENT

R.S. thanks David MacKay for comments and advice when starting to develop this approach (April 2004). He thanks Ernst Pöppel from the Institute of Medical Psychology (LMU, Munich, Germany) for enabling him to establish the Sensory-Motor Learning Lab (since 2004) and Wolfson College Cambridge, University of Cambridge (UK), for electing him a Fellow (2002 to 2006).

## REFERENCES

[1] R. Spiegel, "Relating Bayesian learning to training in recurrent networks," *Proc. of the IEEE International Joint Conference on Neural Networks (IJCNN)*, vol. 2, pp. 908-913, July 2003.

[2] D.J.C. MacKay, *Information theory, inference, and learning algorithms*. Cambridge: Cambridge University Press, 2003.

[3] D.J.C. MacKay, "Bayesian methods for adaptive models," PhD-thesis, California Institute of Technology (USA), 1991.

[4] R.M. Neal, *Bayesian learning for neural networks. New York: Springer*, 1996.

[5] T.M. Mitchell, *Machine learning*. New York: McGraw-Hill, 1997.

[6] J.L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179-211, 1990.

[7] C.K.I. Williams and C.E. Rasmussen, "Gaussian processes for regression," in *Advances in Neural Information Processing Systems*, vol. 8, Cambridge, MA: MIT-Press, 1996.

[8] R.M. Neal, "Monte Carlo implementation of Gaussian process models for Bayesian regression and classification," Technical Report CRG-TR-97-2, Dept. of Computer Science, University of Toronto, 1997.

[9] D. Barber and C.K.I. Williams, "Gaussian processes for Bayesian classification via hybrid Monte Carlo," in *Advances in Neural Information Processing Systems*, vol. 9, Cambridge, MA: MIT-Press, pp. 340-346, 1997.

[10] M. N. Gibbs and D.J.C. MacKay, "Variational Gaussian process classifiers," *IEEE Transactions on Neural Networks*, vol. 11, 1458-1464, 2000.

[11] E.A. Wan, "Neural network classification: a Bayesian interpolation," *IEEE Transactions on Neural Networks*, vol. 1, pp. 303-305, 1990.

[12] C.M. Bishop, *Neural networks for pattern recognition*, Oxford: Oxford University Press, 1995.

[13] D.J.C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, 448-472, 1992.

[14] S. Haykin, *Neural networks. A comprehensive foundation*. Upper Saddle River, N.J.: Prentice-Hall, 1999.

[15] S. Santini and A. Del Bimbo, "Recurrent neural networks can be trained to be maximum a posteriori probability classifiers," *Neural Networks*, vol. 8, 25-29, 1995.

[16] S. Santini, A. Del Bimbo and R. Jain, "Block-structured recurrent neural networks," *Neural Networks*, vol. 8, 135-147, 1995.

[17] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation," in Parallel distributed processing, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA: MIT-Press, vol. 1, pp. 318-362, 1986.

[18] Z. Dienes, G.T. Altmann and S.J. Gao, "Mapping across domains without feedback: a neural network model of transfer of implicit knowledge," *Cognitive Science*, vol. 23, 53-82, 1999.

[19] R. Spiegel, "Predicting the output of a recurrent neural network with a distributed coding scheme by relying on statistical processes," Software developed at the Department of Experimental Psychology, University of Cambridge UK, January 2004.

[20] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning representations by backpropagating errors," *Nature*, vol. 323, pp. 533-536, October 1986.

[21] K. Backhaus, B. Erichson, W. Plinke and R. Weiber, *Multivariate Analysemethoden*. Berlin: Springer, 2003.

[22] J. Schmidhuber, D. Wiestra and J. Gomez, "Evolino: Hybrid neuroevolution / optimal linear search for sequence learning," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 853-858, 2005.

[23] D. Wiestra, F. Gomez and J. Schmidhuber, "Modeling systems with internal state using Evolino," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO)*, New York: ACM-Press, pp. 1795-1802, 2005.

[24] J. Schmidhuber, D. Wiestra and F. Gagliolo, "Training recurrent networks by Evolino," *Neural Computation*, vol. 19, 757-779, 2007.

[25] S.M. Larson and V.S. Pande, "Sequence optimization for native state stability determines the evolution and folding kinetics of a small protein," *Journal of Molecular Biology*, vol. 332, 275-286, September 2003.

[26] S.M. Larson, J.L. England, J.R. Desjarlais and V.S. Pande, "Thoroughly sampling sequence space: large-scale protein design of structural ensembles," *Protein Science*, vol. 11, 2804-2813, December 2002.

[27] N. J. Marianayagam, N.L. Fawzi, T. Head-Gordon, "Protein folding by distributed computing and the denatured state ensemble," *Proceedings of the National Academy of Sciences*, vol. 102, 16684-16689, Nobember 2005.

[28] E. Paci, A. Cavalli, M. Vendruscolo and A. Caflisch, "Analysis of the distributed computing approach applied to the folding of a small beta peptide," *Proceedings of the National Academy of Sciences*, vol. 100, 8217-8222, July 2003.

[29] M. Bodén and J. Hawkins, "Prediction of subcellular localisation using sequence-biased recurrent networks," *Bioinformatics*, vol. 21, 2279-2286, 2005.

[30] B. Kermanshahi, "Recurrent neural network for forecasting next 10 years loads of nine Japanese utilities," *Neurocomputing*, vol. 23, 125-133, 1998.

[31] M. Cheroutre-Vialette and A. Lebert, "Application of recurrent neural network to predict bacterial growth in dynamic conditions," *International Journal of Food Microbiology*, 73, 107-118, March 2002.

[32] S. Wu, I.H. Witten, A.W. Edwards, D.M. Nichols and R. Aquino, "A digital library of language learning exercises," *International Journal of Emerging Technologies in Learning*, vol. 2(1), March 2007.

[33] R. Spiegel, "Cognitive modeling of symbolic-like relationships with the adaptive neural network associator (ANNA)," *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)*, 2003, vol. 4, pp. 2746-2751.

[34] R. Spiegel, "A novel approach to extract rules from sequences of phonemes," *Proc. Cambridge First Postgr. Conf. on Language Research (CamLing)*, 2003, pp. 494-500.

[35] K.O. Nurzyńska and A. Duszeńko, "Interactive system for Polish signed language learning," *International Journal of Emerging Technologies in Learning*, vol. 1(3), 2006.

[36] R. Spiegel, S. Naqvi, J. Ohene-Djan, D.R. Moore and E. Hsiao, "A neuropsychological perspective on measuring sign language learning and comprehension," *International Journal of Emerging Technologies in Learning*, 2(1), March 2007.

[37] J. Ohene-Djan, J. Bassett-Cross, A. Mould and R. Zimmer, "MAK-messenger and finger chat: communications technologies to assist in the teaching of signed languages to the deaf and

hearing," *Proc. 4th IEEE Intern. Conf. Adv. Learn. Technologies*, Joensuu (Finland), 2004.

[38] J. Ohene-Djan, R. Zimmer, M. Gorle and S. Naqvi, "A personalisable electronic book for video-based sign language education," *Educational Technology and Society*, vol. 6, pp. 86-99, October 2003.

[39] J. Ohene-Djan and R. Shipsey, "E-subtitles: emotional subtitles as a technology to assist the deaf and hearing-impaired when learning from television and film," *6th IEEE Intern. Conf. Adv. Learning Technologies (ICALT)*, Kerkrade (Holland): IEEE Computer Society, pp. 464-66, July 2006.

[40] J. Ohene-Djan and S. Naqvi, "An adaptive www-based system to teach British Sign Language," *Proc. 5th IEEE Intern. Conf. Adv. Learning Technologies (ICALT)*, Kaohsiung (Taiwan): IEEE Computer Society, pp. 575-79, July 2005.

[41] S. Naqvi, J. Ohene-Djan and R. Spiegel, "Testing the effectiveness of digital representations of sign language with children," *Instructional Technology and Education of the Deaf Symposium*, Rochester, N.Y.: National Institute for the Deaf, pp. 1-7, June 2005.

[42] K. Jantz, G. Friedland and R. Rojas, "Ubiquitous pointing and drawing," *International Journal of Emerging Technologies in Learning*, 2(1), March 2007.

[43] D.R. Moore, "Exploring gender effects on the spatial probability measure," in *Proc. 5th IEEE Intern. Conf. on Advanced Learning Technologies (ICALT)*, Athens (Greece): IEEE Computer Soc., 2005, pp. 435-36.

[44] S. Naqvi, R. Spiegel, J. Ohene-Djan, D.R. Moore and E. Hsiao, "Measuring sign language comprehension through spatial response," in *Proc. Conf. and Workshop on Assistive Technologies for Vision and Hearing Impairment: Technology for Inclusion*, D.M. Hersh, Ed. Kufstein (Austria): euro-Assist-VHI-4, 2006.

[45] R. Spiegel, "Human and machine learning of spatio-temporal sequences: an experimental and computational investigation," PhD-thesis, University of Cambridge (UK), 2002.

[46] R. Spiegel and I.P.L. McLaren, "Abstract and associatively-based representations in human sequence learning," *Phil. Trans. Roy. Soc. London*, vol. B358, pp. 1277-1283, July 2003.

[47] R. Spiegel and I.P.L. McLaren, "Associative sequence learning," *J. Expt. Psychology: Animal Behavior Processes*, vol. 32, pp. 150-163, April 2006.

[48] J. Azar, H. Abou Saleh, M. Adnan Al-Alaoui, "Sound visualization for the hearing impaired," *International Journal of Emerging Technologies in Learning*, 2(1), March 2007.

[49] J.L. Elman, E. Bates, M.H. Johnson, A. Karmiloff-Smith, D. Parisi and K. Plunkett, *Rethinking Innateness: A Connectionist Perspective on Development*. Cambridge, MA: MIT-Press, 1996.

[50] K. Plunkett and J.L. Elman, *Exercises in Rethinking Innateness*. Cambridge, MA: MIT-Press, 1997.

[51] P. McLeod, K. Plunkett and E.T. Rolls, *Introduction to Connectionist Modelling of Cognitive Processes*. Oxford: Oxford University Press, 1998.

## AUTHORS

**R. Spiegel** is with the Institute of Medical Psychology, Ludwig-Maximilians University (LMU) and its affiliated Generation Research Program Ltd., Goethestr. 31/1, D-80336 Munich, Germany and with the Medical School of the same university, e-mail:rainer.spiegel@campus.lmu.de When carrying out the work presented in this paper, he was at Wolfson College Cambridge, University of Cambridge, United Kingdom (2002 to 2006).

Manuscript received 22 April 2007.

Published as submitted by the author(s).