Neural Networks: a replacement for Gaussian Processes?

Matthew Lilley and Marcus Frean

Victoria University of Wellington, P.O. Box 600,
Wellington, New Zealand
marcus@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/∼ marcus

Abstract. Gaussian processes have been favourably compared to back-propagation neural networks as a tool for regression. We show that a recurrent neural network can implement exact Gaussian process inference using only linear neurons that integrate their inputs over time, inhibitory recurrent connections, and one-shot Hebbian learning. The network amounts to a dynamical system which relaxes to the correct solution. We prove conditions for convergence, show how the system can act as its own teacher in order to produce rapid predictions, and comment on the biological plausibility of such a network.

1 Introduction

Multi-layer Perceptron (MLP) neural networks are powerful models of broad applicability, able to capture non-linear surfaces of essentially arbitrary complexity. However such networks have their drawbacks, two of which we highlight here. Firstly, the learning algorithm is generally believed to be implausible from a biological point of view, for example in requiring synapses to act bi-directionally, and being very slow to train. Secondly, there is no uncertainty model in a neural network implemented in this way: given an input, the network produces an output directly, and nothing else. This is undesirable – the real world is dominated by uncertainty, and predictions without uncertainty are of limited value.

Gaussian process regression is not based on any biological model, but provides an explicit uncertainty measure and does not require the lengthy 'training' that a neural network does. While techniques for obtaining uncertainty from a neural network exist [9], [6] they are additions to the architecture, whereas Gaussian processes have uncertainty as a fundamental component arising naturally from a Bayesian formulation. Indeed, predictions made by neural networks approach those made by Gaussian processes as the number of hidden units tends to infinity. There are good arguments for Gaussian processes being considered a replacement for supervised neural networks [7].

Here we show that neural networks can themselves implement Gaussian process regression, in a way that has interesting parallels with neural circuitry.

2 Gaussian Process regression

Suppose we are given training data D consisting of input patterns $\{\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_n\}$, each of which is a vector, paired with their associated scalar output values $\mathbf{t} = \{t_1, t_2 \ldots t_n\}$. MLP networks can be thought of as imperfectly transforming this data into a set of representative weights. The actual data is not directly involved in making predictions for a new target t given a new input vector $\mathbf{x}$. The process of training the network (setting the weights) is slow, but the predictions are fast.

Gaussian processes make predictions in a way that is fundamentally different to MLP networks. Rather than capturing regularities in the training data via a set of representative weights, they apply Bayesian inference to explicitly compute the posterior distribution over possible output values t given all the data D and the new input x. This process involves $\mathbf{C}$, a covariance matrix generated using a covariance function $Cov(x, x'; \Theta)$ where Θ are hyper-parameters. Although a variety of other alternatives are possible [13], a typical form for the covariance function is

$$Cov(\mathbf{x}, \mathbf{x}') = \theta_1 \exp\left(-\frac{(\mathbf{x}-\mathbf{x}')^2}{2\theta_2^2}\right) + \theta_3\, \delta_{\mathbf{x},\mathbf{x}'} \ .$$

θ_1 determines the relative scale of the noise in comparison with the data. θ_2 characterises the distance in x over which t is expected to vary significantly. θ_3 models white noise in measurements and δ is the delta function.

Essentially, the covariance matrix determines the scale and orientation of a Gaussian distribution amongst the variables $\mathbf{t}$. The task of regression is to find the distribution $P(t|D, \mathbf{x}, \mathbf{C}, \Theta)$, conditioning on the n input-output pairs corresponding to the training data, together with the new input. For a Gaussian process this conditioning process can be done analytically, resulting in a 1-D Gaussian distribution characterised by the following (see e.g. [2] for a derivation):

$$\texttt{mean} = \mathbf{k}^T\mathbf{C}^{-1}\mathbf{t}, \qquad \texttt{variance} = \kappa - \mathbf{k}^T\mathbf{C}^{-1}\mathbf{k}\,. \tag{1}$$

Here $\mathbf{C}_{ij} = Cov(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{k}$ is the vector of individual covariances $\mathrm{k}_j = Cov(\mathbf{x}_j, \mathbf{x})$ between the new input $\mathbf{x}$ and each of those in the data set. κ is $Cov(x, x)$, a constant for stationary convariance functions. For the above it is $\theta_1 + \theta_3$.

3 Gaussian processes as neural networks

In this section we show how relatively simple neural circuitry could carry out the operations required for Gaussian process inference.

Firstly, notice that the vector $\mathbf{k}$ can be thought of as the output of a layer of radial basis function (RBF) units, given input pattern $\mathbf{x}$. Each RBF unit arises from a previously observed input vector, and calculates its response to the new input using a Gaussian receptive field centered on the original vector, just as so-called "grandmother cells" [4] show peak activity for a particular input pattern, and progressively less response the greater the difference between the current stimulus and this pattern.

The primary task appears at first to be inversion of $\mathbf{C}$, but this is not strictly necessary - it is sufficient to find $\mathbf{k}^T\mathbf{C}^{-1}$. Thus the problem can be reformulated as follows: given a matrix $\mathbf{C}$ and a vector $\mathbf{k}$, we wish to find $\mathbf{C}^{-1}\mathbf{k}$. Supposing that $\mathbf{C}^{-1}\mathbf{k} = \mathbf{g}$, pre-multiplying by $\mathbf{C}$ gives $\mathbf{k} = \mathbf{Cg}$. The problem then reduces to iteratively improving $\mathbf{g}$ until the difference between $\mathbf{Cg}$ and $\mathbf{k}$ is sufficiently small. Gibbs [2] defines a measure $Q = \mathbf{g^T}.(\mathbf{k} - \frac{1}{2}\mathbf{Cg})$, the gradient of which is:

$$\nabla_g Q = (\mathbf{k} - \mathbf{Cg}) \ . \tag{2}$$

This gradient is zero at the solution to our problem. Gibbs uses a conjugate gradient routine to locate the solution. However for our purposes note that since $\nabla^2_g Q = -\mathbf{C} < 0$, $\mathbf{g}$ can be iteratively improved by simply taking a small step in the direction of the gradient,

$$\mathit{\Delta}\mathbf{g} = \eta(\mathbf{k} - \mathbf{Cg}) \ . \tag{3}$$

In the Appendix we show that this algorithm converges on the exact solution, and we derive an optimal value for η.

The change to each component of $\mathbf{g}$ is a linear function of itself at a previous time-step, which suggests a network of linear neurons that integrate their inputs over time. The input needs to be $\mathbf{k} - \mathbf{Cg}$, so we have direct input of $\mathbf{k}$ together with inhibitory recurrent connections between all the $\mathbf{g}$ neurons, with weights $-\mathbf{C}_{ij}$. The change to the neuron's output activity is simply the sum of these, times a constant η. Once this network converges we have only to take the dot product with the vector of targets (equation 1), which is easily achieved via a second layer of weights whose values are set to their respective target outputs (Figure 1).

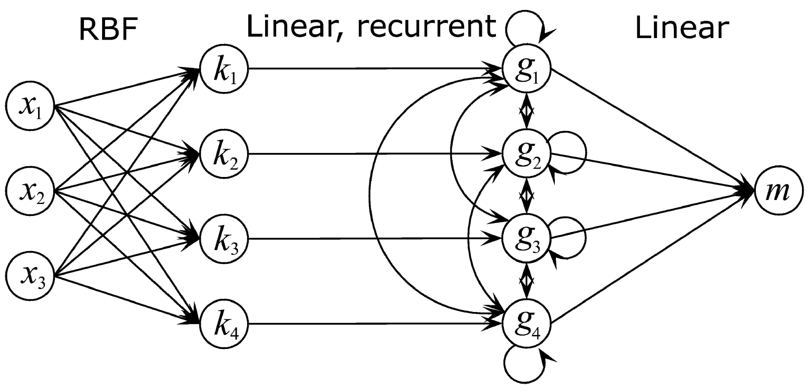

Fig. 1. A network architecture that implements Gaussian process regression. It converges on the mean m of the predicted output distribution, given input $\mathbf{x}$. Connections from $\mathbf{k}$ to $\mathbf{g}$ have weights of 1, recurrent connections have weights $-\mathbf{C}_{ij}$, and those from $\mathbf{g}$ to the output m have weights $\mathbf{t}$. In this case the input is a 3-dimensional vector and inference is carried out on the basis of 4 input-output pairs

The rate of convergence depends on the choices for hyperparameters, as shown in Figure 2 for a representative range of outcomes. θ_2 plays a crucial role, as it effectively determines the expected number of datapoints involved in each prediction. If θ_2 is small then the new input is unlikely to be close to any previous data and therefore $\mathbf{k} \approx 0$, or it may be close to just one previous input, in which case only one element of $\mathbf{k}$ is significantly non-zero. Larger values of θ_2 make both $\mathbf{k}$ and $\mathbf{C}$ less sparse, and intuitively one can see that this will require more iterations to take account of the corresponding interrelationships.

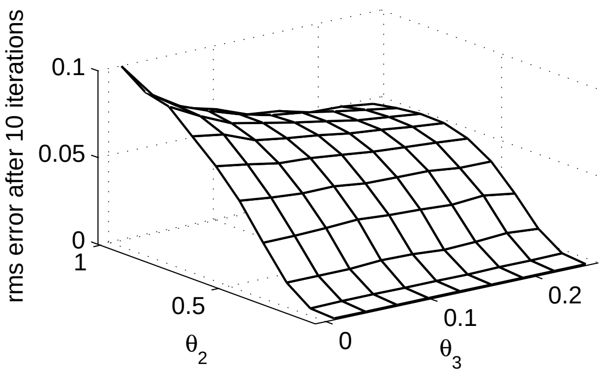

Fig. 2. The rms error between g and $\mathrm{k}^T\mathrm{C}^{-1}$ after 10 iterations of the dynamics. We used 100 data points chosen at random from within a 10-dimensional hypercube. θ_1 was fixed at 2.0 and the convergence rate for various values of θ_2 and θ_3 explored. Each vertex is an average over 100 independent runs. The rate parameter η was set to the value derived in the Appendix

The various connections in the system need to be set to particular values in order for this procedure to work. Firstly, the RBF units must each be centered on a unique input pattern. We don't address exactly how this might be implemented here, but one can imagine a constructive process in which a novel input pattern $\mathbf{x}_{n+1}$ triggers the recruitment of a new cell whose output is, and remains, maximal for that pattern. By thinking of the network in this constructive manner it is also clear how the other connections might be set: another new cell g_{n+1} is similarly recruited, and receives input from $\mathbf{x}_{n+1}$ with a synaptic weight of one. Its weights both to and from any other $\mathbf{g}$ cell, say g_i, need to be $-Cov(\mathbf{x}_i, \mathbf{x}_{n+1})$, which is simply the value taken by k_i for the current input. Indeed this is locally available as the instantaneous[1] value of g_i, amounting to a form of (anti) Hebbian learning. Finally the synaptic weight from g_{n+1} to the "output" must be set to t_{n+1}, which we may assume is the output cell's current value. In this way a network is both constructed and "learned" by local mechanisms as input-output pairs are presented to it.

[1] *i.e.* the value g_i takes, prior to being perturbed by the recurrent neural dynamics

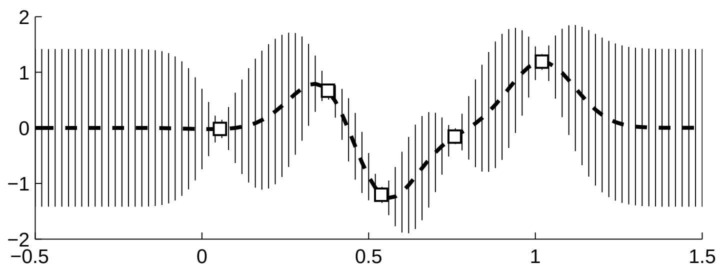

Fig. 3. A 1-dimensional example for illustrative purposes. There are 5 data points (squares). Predictions were then made by running the dynamics for 100 iterations, for a range of inputs. The dashed line is the mean and vertical bars indicate one standard deviation of the uncertainty, calculated as described in the text

The variance of the prediction is given by the expression $\mathbf{k}^T\mathbf{C}^{-1}\mathbf{k}$. Part of this ($\mathbf{k}^T\mathbf{C}^{-1}$) has already been calculated by the network which determines the mean, so we can reuse the previous calculation and essentially get the variance for (almost) no added cost. All that remains is to find the dot product of the $\mathbf{g}$ vector with $\mathbf{k}$. Initially this seems like a trivial problem, but from a biological perspective it poses some difficulty. A possible mechanism is suggested by the process known as *shunting inhibition*, proposed as a possible mechanism for neurons to divide numbers [1] in which the output of one neuron inhibits the transmission of charge between two other neurons. As the elements of $\mathbf{k}$ are between 0 and 1, computing $\mathbf{k}^T\mathbf{C}^{-1}\mathbf{k}$ can be considered to be scaling the elements of $\mathbf{k}^T\mathbf{C}^{-1}$ by the elements of $\mathbf{k}$, a task to which shunting inhibition seems ideally suited. Against this, some precise wiring is now required, as the $i^{\mathtt{th}}$ $\mathbf{k}$ neuron must gate the effect of the $i^{\mathtt{th}}$ $\mathbf{g}$ neuron on the output.

3.1 Faster predictions over time

The inference mechanism described here is very fast to learn (one-shot Hebbian), but is slow in making predictions due to the iterative process by which it arrives at the solution. However, we can use the iterative algorithm to generate “targets” with which to learn a second, single layer forward-feed network which runs in parallel with the Gaussian process mechanism and attempts to learn weights corresponding to $\mathbf{C}^{-1}$. Given $\mathbf{k}$ this secondary network can then directly compute $\mathbf{k^T C^{-1}}$ in a single pass (see Figure 4). One can think of the inversion network as an oracle, which generates training data for the direct network given the raw inputs. We have shown experimentally and analytically [5] that this process converges exponentially quickly to the correct solution.

4 Discussion

A multi-layer perceptron neural network has Gaussian process behaviour when the number of hidden neurons tends to infinity, provided weight decay is em-

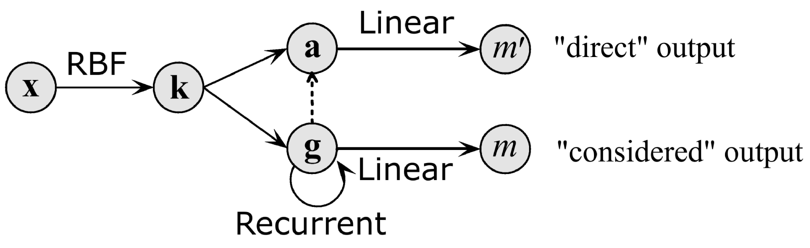

Fig. 4. Schematic representation of a parallel network that produces fast predictions and is trained on targets provided by the slower iterative process. The output of the **k** neuron is used as an input to the neurons labeled **a** on the "direct" path. They use the target of the **g** neurons (shown by the dotted line) to adjust their weights. Another possibility (not investigated further here) would be to learn a direct linear mapping from **k** to m'

ployed [9]. We have argued that the converse is also true in the sense that the calculations required to calculate the expected output can be carried out by a simple neural network. In effect an infinite number of hidden units in a feed-forward architecture can be replaced by a merely finite number, together with recurrent connections and the ability to accumulate activity over time.

Recovery from intermittent errors can be shown to be exponentially fast [5]. This leads to the appealing property that accuracy improves exponentially with time: early results are rough, later results become exact.

However there are some difficulties with our proposal. Calculating the correct variance is considerably more problematic than finding the mean. While shunting inhibition is a potential mechanism for achieving this, it does require some rather precise neural wiring. Similarly, we have avoided dealing with the setting of hyperparameters Θ. While there are several possible avenues that might be pursued [5] they all appear to involve further additions to the architecture described here.

There is an interesting relationship between the algorithm presented here and a neural architecture suggested for faithful recoding of sensory input data by the visual cortex[14], [15]. Essentially the Daugman algorithm minimizes the difference between sensory input and internal template-based models by step-wise gradient descent. The dynamics are similar to those we describe except that there is feedback from the equivalent of the **g** layer (the internal model) *back* to the **k** layer (the sensory input), rather than recurrent connections within **g**. Perfoming this simplistic gradient descent on $Q = -\frac{1}{2}(\mathbf{g} - \mathbf{C}\mathbf{k})^T(\mathbf{g} - \mathbf{C}\mathbf{k})$ is dimensionally inconsistent [6]. Fortunately, this is not a fatal problem and can be remedied by premultiplying Q by its curvature, which is simply $\mathbf{C}^{-1}$ [5]. This leads to the remarkable conclusion that our algorithm is actually a covariant form of the biologically inspired algorithm proposed by Daugman.

References

1. Dayan, P., and Abbott, L. *Theoretical Neuroscience*. Massachusetts Institute of Technology, 2001, p. 189.
2. Gibbs, M. N. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997.
3. Hebb, D. O. *The Organisation of Behaviour*. Wiley, New York, 1949.
4. J.Y. Lettvin, H.R. Maturana, W. M., and Pitts, W. *The Mind: Biological Approaches to its Functions*. Interscience Publishers, 1968, ch. 7, pp. 233–258.
5. Lilley, M. Gaussian processes as neural networks. Honours thesis, Victoria University of Wellington, 2004. Available from `http://www.mcs.vuw.ac.nz/people/Marcus-Frean`
6. MacKay, D. *Information Theory, Inference, and Learning Algorithms*. University Press, 2003, ch. 34.
7. MacKay, D. J. Gaussian processes - a replacement for supervised neural networks? Lecture notes for a tutorial at NIPS 1997. `http://www.inference.phy.cam.ac.uk/mackay/gpB.pdf`
8. McIntosh, H. V. *Linear Cellular Automata*. Universidad Autonoma de Puebla, 1987, ch. 9.4.
9. Neal, R. Priors for infinite networks. Tech. rep., University of Toronto, 1994.
10. Petersen, K. The matrix cookbook. Technical University of Denmark, 2004. `http://2302.dk/uni/matrixcookbook.html`
11. Weisstein, E. W. Eigen decomposition theorem. `http://mathworld.wolfram.com/EigenDecompositionTheorem.html`.
12. Weisstein, E. W. Positive definite matrix. `http://mathworld.wolfram.com/PositiveDefiniteMatrix.html`.
13. Williams, C. K. I., and Rasmussen, C. E. Gaussian processes for regression. *Advances in Neural Information Processing Systems 8* (1996), 514–520.
14. Daugman, J. Complete Discrete 2-D Gabor Transforms by Neural Networks for Image Analysis and Compression. *IEEE Trans. ASSP*, vol.36 no. 7 (1988), pp. 1169-1179
15. Pece, A.E.C. Redundancy reduction of a Gabor representation: a possible computational role for feedback from primary visual cortex to lateral geniculate nucleus, *Unpublished manuscript*, 1993.

Appendix

Here we prove that our algorithm is correct using infinite series, which clarifies the conditions under which the inversion converges, and leads to a technique to force otherwise non-convergent matrices to converge. We start by assuming the value of $\mathbf{g}$ at time zero is $\mathbf{0}$. Then, at time t, $\mathbf{g}_t = \mathbf{g}_{t-1} + \mathbf{k}^T - \mathbf{C}\mathbf{g}_{t-1}$ which is $\mathbf{g}_{t-1}(\mathbf{I} - \mathbf{C}) + \mathbf{k}^T$. The closed form for $\mathbf{g}$ at time t is then

$$\mathbf{g}(t) = \mathbf{k}^T \sum_{i=0}^{t-1} (\mathbf{I} - \mathbf{C})^i \ .$$

Multiplying both sides by $(\mathbf{I} - \mathbf{C})$, subtracting from $\mathbf{g}(t)$, and right-multipling by $\mathbf{C^{-1}}$ yields

$$\mathbf{g}(t) = \mathbf{k}^T \left(\mathbf{I} - (\mathbf{I} - \mathbf{C})^t\right) \mathbf{C}^{-1} \tag{4}$$

Taking the limit as $t \to \infty$ gives $\mathbf{g}(t) = \mathbf{k}^T\mathbf{C}^{-1}$, as required. In order for $\mathbf{g}(t)$ to converge, we must assume that $\lim_{n\to\infty}(\mathbf{I} - \mathbf{C})^n = 0$. Making use of the eigen-decomposition theorem [11] we can rewrite $\mathbf{I} - \mathbf{C}$ in terms of the matrix D which has the eigenvalues of $\mathbf{I} - \mathbf{C}$ along its diagonal so that $\mathbf{I} - \mathbf{C} = P^{-1}DP$, and since $(P^{-1}DP)^n = P^{-1}D^nP$ all that remains is to show that

$$\lim_{n\to\infty} P^{-1}D^nP \tag{5}$$

is defined and finite. Because D is diagonal, $[D^n]_{ij} = [D]_{ij}^n$ and so we conclude that if for all eigenvalues λ_i of $\mathbf{I} - \mathbf{C}$, $|\lambda_i| < 1$ then this is simply the zero matrix, which is defined, and finite. Otherwise, the limit is infinite, and therefore the algorithm fails. In general, $|\lambda_i| \not< 1$, but we can force the condition by introducing a new parameter, η, as follows. If $\mathbf{g}_t = \mathbf{g}_{t-1} + \eta\mathbf{k}^T - \eta\mathbf{C}\mathbf{g}_{t-1}$ then by a similar process to which equation 4 was derived, we have

$$\mathbf{g}(t) = \eta\mathbf{k}^T\left(\mathbf{I} - (\mathbf{I} - \eta\mathbf{C})^t\right)(\eta\mathbf{C})^{-1} \tag{6}$$

If we choose η such that the eigenvalues of $\mathbf{I} - \eta\mathbf{C}$ are of magnitude less than one, then equation 5 will converge, and ultimately equation 6 will converge also. It is an identity that the eigenvalues λ of $\mathbf{I} + \alpha M$ are $1 + \alpha\lambda$ [10], and the eigenvalues of a positive definite matrix are all positive or zero [12], therefore by letting α be $-\frac{1}{\max\lambda}$, we guarantee that all eigenvalues of $\mathbf{I} - \mathbf{C}$ are of magnitude equal to or less than one. Imposing the condition that θ_3 is non-zero effectively prevents the matrix from being ill-conditioned. The largest eigenvalue of $\mathbf{C}$ is strictly less than the maximal row sum (for a symmetric matrix) [8], which in turn is bounded by $N(\theta_1 + \theta_3)$, for a matrix having N columns.

$$\eta_{estimate} = |N(\theta_1 + \theta_3) + 1|^{-1} \tag{7}$$

Equation 7 gives a tractable way to approximate an appropriate value of η. Empirical evidence suggest that using the estimate described in equation 7 indeed has similar performance to using the inverse of the largest eigenvalue of $\mathbf{I} - \mathbf{C}$, which appears to be optimal.