ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 19-24

Projection Learning: Alternative Approaches to the Computation of the Projection

Konrad Weigl and Marc Berthod*

INRIA - B.P. 93 - 06902 Sophia Antipolis Cedex France

Abstract. We introduced in [13] [14] [15] the paradigm of considering a feedforward neural network with one hidden layer as a non-orthogonal base in a function space and have derived a new learning algorithm, *Projection Learning*: It consists simply in shifting the manifold spanned by the base in such a way that the distance to the function to be approximated is minimized. We have used up to now the metric tensor of the base to compute the projection of the function to be approximated onto the manifold. In the present paper we show alternative approaches to that computation: An approach via recursive least squares, which is also an approach used in system identification and by the Kalman filter algorithm, and two approaches postulated to exist in the primate visual system, and implementable via an auxiliary neural network.
Keywords: Tensor Theory, Projection Operators, Metric Tensors, Radial Basis Functions, Multi-layer perceptrons

1 The general approach

In the present paper we shall concentrate on the mathematical aspects. Refer to the papers above and [16] for the paradigm. Our aim is to teach a network to approximate as well as possible a function F given by examples $F(x_k), k \in \{1,..,M\}$, where M is the number of examples. Let $g_i, i \in \{1,..,N\}$ be the set of arbitrary differentiable functions computed by the N hidden-layer neurons, and $g_i(x_k), k \in \{1,..,M\}, i \in \{1,..,N\}$ be the output values computed by the hidden-layer neurons for given inputs x_k. We shall assume a linear output neuron[2]. The problem belongs to the class of separable non-linear least squares [4] [5] [6]. For a given set of weights from input to hidden layer, we can then express the problem of computing the optimal set $A^i, i \in \{1,..,N\}$[3], of hidden-to output layer as a least-squares problem, assuming the number of examples greater or equal to the number of hidden-layer neurons:

$$\sum_i A^i g_i(x_k) = F(x_k)\ , \forall k \qquad (1)$$

* email: weigl@sophia.inria.fr berthod@sophia.inria.fr
Correspondence to: Konrad Weigl

[2] We have shown in [16] the extension to a non-linear output neuron with invertible activation function; extension to multiple output neurons is trivial

[3] The superscripts do not indicate power exponents!

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 19-24

Since the system is overdetermined, there will be no exact solution; however, the solution found $A(x_k) = \sum_i A^i g_i(x_k)$ will be the orthogonal projection of the function $F(x_k)$ to be approximated unto the linear submanifold in function space spanned by the hidden-layer neurons. Since we want the function $A(x_k)$ to be as close as possible to the function $F(x_k)$, all that remains to be done is to shift the submanifold by changing the input weights in such a way that that distance is minimized, using e.g. gradient descent. Let D be the distance to be minimized, usually a mean-squared error, and γ the vector of all the input to hidden-layer weights:

$$D(F, A(\gamma)) = \sum_x (F(x) - A(\gamma, x))^2 = \sum_x (F(x) - \sum_\nu A^\nu g_\nu(x))^2 \longrightarrow \quad (2)$$

$$\frac{dD(F,A(\gamma))}{d\gamma} = \frac{d(\sum_x (F(x) - \sum_\nu A^\nu g_\nu(x))^2)}{d\gamma} = \sum_x [-2(D(F, A(\gamma)) \frac{d\sum_\nu A^\nu g_\nu(x)}{d\gamma}] \longrightarrow \quad (3)$$

$$\frac{dD(F, A(\gamma))}{d\gamma} = \sum_x [-2(D(F, A(\gamma))(\frac{d\sum_\nu A^\nu}{d\gamma} g_\nu(x) + \frac{d\sum_\nu g_\nu(x)}{d\gamma} A^\nu] \quad (4)$$

The term $\frac{d\sum_\nu A^\nu}{d\gamma}$ involves computing the derivative of every component of a matrix w.r.t. every component of its inverse and goes thus with N^4 if N is the rank of the matrix. We have shown in [16] that $D(F, A(\gamma)) \frac{d\sum_\nu A^\nu}{d\gamma}$ simply cancels out, leaving the final formula:

$$\frac{dD(F, A(\gamma))}{d\gamma} = -2 \sum_x (F - \sum_\nu A^\nu g_\nu(x)) \sum_\nu A^\nu \frac{dg_\nu(x)}{d\gamma} \quad (5)$$

We compute usually $A^\nu = \sum_\mu g^{\mu\nu} A_\mu$, which is the direct solution of the normal equations: $g^{\mu\nu}$ are the components of the *contravariant metric tensor*, the inverse of the *covariant metric tensor* $g_{\mu\nu}$, which in turn is defined as: $g_{\mu\nu} = < g_\mu, g_\nu >$, $<,>$denoting the scalar product, summing over all x_k. The *covariant components* A_μ are computed by: $A_\mu = < F, g_\mu >$.
Here, however, we shall introduce two other options, which are of interest because they shows links to other neural network approaches resp. to other estimation algorithms.

2 Computation of the hidden to output layer weights by gradient descent

One option is the approach used by Daugman, Pattison and Pece [2] [9] [8][4] to compute expansions of Gabor functions or Poggio [10] for Hyperbasis functions of fixed shape: They compute A^ν using the minimization of an energy expression, and implement it via neural networks of different topologies: $\forall \mu$

$$< F, g_\mu > = \sum_x g_\mu(x) \sum_\nu A^\nu g_\nu(x) \longrightarrow < F, g_\mu > = \sum_\nu g_{\mu\nu} A^\nu \quad (6)$$

[4] See also [3]

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 19-24

which gives the energy:

$$E = \sum_{\mu}(< F, g_{\mu} > - \sum_{\nu} g_{\mu\nu} A^{\nu}(t))^2 \tag{7}$$

and thus the following expression:

$$\frac{dA^{\sigma}(t)}{dt} = -\frac{dE}{dA^{\sigma}} = 2\sum_{\mu}(< F, g_{\mu} > - \sum_{\nu} g_{\mu\nu} A^{\nu}(t)) g_{\sigma\mu} \tag{8}$$

The algorithm can be implemented as above, i.e. via gradient descent; since the energy is convex, a deterministic approach is sufficient.

3 Computation of the hidden to output layer weights by recursive least squares

Another approach is recursive least squares computation, such as is used in the Kalman filter or generally in system identification [12]: We compute A^{ν} iteratively, which avoids the need to invert a matrix, except once at the start, to initialize the system: We take directly the approach from [12], pg. 322 ff.: Writing the covariant metric tensor as a function of a variable t:

$$g_{\mu\nu}(t) = \sum_{s=0}^{t} g_{\nu}(s) g_{\mu}(s) \tag{9}$$

Once t is equal to the total number of dimensions of our system, thus in our case the number of examples we have, we obtain: $g_{\mu\nu}(t) = g_{\mu\nu}$.
We can also write:

$$g_{\mu\nu}(t) = g_{\mu\nu}(t-1) + g_{\mu}(t) g_{\nu}(t) \tag{10}$$

We define $g^{\mu\nu}(t)$ as the components of the inverse of $g_{\mu\nu}(t)$, and generally:

$$A_{\nu}(t) = \sum_{\mu} g_{\mu\nu}(t) A^{\mu}(t) \; and \; A^{\nu}(t) = \sum_{\mu} g^{\mu\nu}(t) A_{\mu}(t) \tag{11}$$

where

$$A_{\mu}(t) = \sum_{s=0}^{t} g_{\mu}(s) F(s) \; Thus : A^{\nu}(t) = \sum_{\mu} g^{\mu\nu}(t)[A_{\mu}(t-1) + F(t) g_{\mu}(t)] \tag{12}$$

Inserting from above:

$$A^{\nu}(t) = \sum_{\mu} g^{\mu\nu}(t)[g_{\mu i}(t-1) A^{i}(t-1) + F(t) g_{\mu}(t)] \tag{13}$$

since we have:

$$g_{\mu\nu}(t-1) = g_{\mu\nu}(t) - g_{\mu}(t) g_{\nu}(t) \tag{14}$$

we can write:

$$A^{\nu}(t) = \sum_{\mu} g^{\mu\nu}(t)[g_{\nu i}(t) A^{i}(t-1) - g_{\mu}(t) g_{i}(t) A^{i}(t-1) + F(t) g\mu(t)] \tag{15}$$

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 19-24

Contracting $g^{\mu\nu}$ with $g_{\nu i}$ to $\delta_i^\mu \sum_\nu g^{\mu\nu} g_{\nu i} = \delta_i^\mu$, where δ_i^μ is the kronecker delta, we obtain:

$$A^\nu(t) = A^\nu(t-1) - \sum_\mu \sum_i g^{\mu\nu}(t) g_\mu(t) g_i(t) A^i(t-1) + F(t) \sum_\mu g^{\mu\nu}(t) g_\mu(t)] \quad (16)$$

From which we obtain:

$$A^\nu(t) = A^\nu(t-1) + g^\nu(t)[F(t) - \sum_i g_i(t) A^i(t-1)] \quad (17)$$

which allows us to compute A^ν recursively, by letting t run through all the examples. A result from [12] allows for recursive computation of the inverse $g^{\mu\nu}(t)$:

$$g^{\mu\nu}(t) = g^{\mu\nu}(t-1) - \frac{\sum_i \sum_j g^{\mu i}(t-1) g_{ij}(t) g^{j\nu}(t-1)}{1 + \sum_\mu \sum_\nu g_\mu(t) g^{\mu\nu}(t-1) g_\nu(t)} \quad (18)$$

From that equation we obtain an equation for $g^\mu(t)$, also in [12]:

$$g^\mu(t) = \frac{\sum_\nu g^{\mu\nu}(t-1) g_\nu(t)}{1 + \sum_\mu \sum_\nu g_\mu(t) g^{\mu\nu}(t-1) g_\nu(t)} \quad (19)$$

4 Comparison of methods used to compute projection

The method is generally faster than backpropagation by one order of magnitude or more (Cf. fig. 2 and [16]), but the choice of fastest implementation depends on circumstances: The direct inversion is definitely the fastest if the number of nodes is not too big, since the complexity of the method increases with the complexity of the matrix inversion; The neural-network approach via the computation of the contravariant components is suitable if we already have a neural-network architecture, furthermore it might be more suitable if we want the network to be fault-tolerant; the approch via recursive least squares is interesting if we have little storage space w.r.t. the number of samples we want to train or if we simply want to train online, i.e. we process the data as it comes in, and delete it after having used it once [5]. It is also possible to apply recursive least squares in parallel, while the base itself moves, if we introduce a suitable forgetting factor.

5 Conclusion and outlook

We believe that the approaches outlined here will provide further options to the application of the projection learning paradigm: It demonstrates the flexibility of implementation of the algoritm, giving it another interesting facet, besides its superior speed and its lack of parameters to adjust as compared to classical approaches.

[5] See [11] for such an application with backpropagation

We have applied our algorithm to the detection of inhabited sites on satellite images, with remarkable results [17]. We show in fig. 1 the general behaviour of the algorithm, and in fig. 2 a comparison of projection learning to backpropagation, both implemented with standard gradient descent.
We are of the opinion that the general paradigm presented, along with other approaches [1] [7], will shed new light on the intrinsic nature of neural mapping systems, and will thus deepen our understanding of such systems.

References

1. Biegler-Koenig, F., and Baermann, F., A Learning Algorithm for Multi-Layered Neural Networks Based on Linear Least Squares, in: Neural Networks, Vol. 6, o. 1pp. 127-131, 1993, Pergamon Press
2. Daugman, J., Complete Discrete 2-D Gabor Transform by Neural Network for Image Analysis and Compression, in: IEEE trans. ASSP, vol. 36, No. 7, July 1988, pg. 1169-1179
3. Gaál, G., Population Coding by Simultaneous Activities of Neurons in Intrinsic Coordinate Systems Defined by their Receptive Field Weighting Functions, in: Neural Networks, Vol. 6, pp. 499-515, 1993
4. Golub, G.H., and Pereyra, V., The Differentiation of Pseudo-inverses and Nonlinear Least-Squares Problems Whose Variables Separate, SIAM, J. Numer. Analysis, Vol. 10, No. 2, April 1973, pg. 413-431
5. Kaufman, L., A Variable Projection Method for Solving Separable Nonlinear Least Squares Problems, BIT 15 (1975), pg. 49-57
6. Krogh, F.T., Efficient Implementation of a Variable Projection Algorithm for Nonlinear Least-Squares Problems, Numerical Mathematics, Communications of the ACM, March 1974, Vol. 17, Number 3.
7. Pati, Y.C., and Krishnaprasad, P.S., Analysis and Synthesis of Feedforward Neural Networks Using Discrete Affine Wavelet Transformation, in: IEEE Trans. on Neural Networks, Vol. 4, No. 1, January 1993
8. Pattison, Tim R., Relaxation Network for Gabor Image Decomposition, Biological Cybernetics, 67, pg. 97-102, 1992
9. Pece, Redundancy Reduction of a Gabor Representation: A Possible Computational Role of Feedback from Primary Visual Cortex to Lateral Geniculate Nucleus, Proc. ICANN92, Brighton, North-Holland, Elsevier, Publishers, 1992
10. Poggio, T., and Girosi, F., Networks for Approximation and Learning, in Proc. IEEE, Vol. 78. No. 9, Sept. 1990, pg. 1488
11. Riemschneider, K.-R., Geraueschklassifikation mit Neuronalen Netzen, Universitaet der Bundeswehr Hamburg, Technische Informatik, Hamburg, Germany; also personal communication
12. Soederstroem, T., and Stoica, P., System Identification, Prenctice Hall, New York et al., 1989, pg. 320 ff.
13. Weigl, K., and Berthod, M., Metric Tensors and Dynamical Non-Orthogonal Bases: An Application to Function Approximation, invited talk, in Proc. WOPPLOT 1992, Workshop on Parallel Processing: Logic, Organization and Technology, Springer Lecture Notes in Computer Sciences, to be published.
14. Weigl, K., and Berthod, M., Non-orthogonal Bases and Metric Tensors: An Application to Artificial Neural Networks, in New Trends in Neural Computa-

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 19-24

tion, Proc. IWANN'93, International Workshop on Artificial Neural Networks, Springer Lecture Notes in Computer Sciences, vol. 686.

15. Weigl, K., and Berthod, M., Non-orthogonal Bases and Metric Tensors, invited talk, in: Workshop on Neural Networks, Huening H. et al. eds, Aachen 1993, Verlag der Augustinus Buchhandlung, ISBN 3-86073-140-8
16. Weigl, K., and Berthod, M., Neural Networks as Dynamical Bases in Function Space, research report INRIA, to be published.
17. Weigl, K., Giraudon, G., and Berthod, M., A Fast New Neural Network Learning Algorithm and an Application to the Detection of Inhabited Areas on Satellite Images, submitted to the conference on Computer Vision and Pattern Recognition, Seattle, Washington, 1994

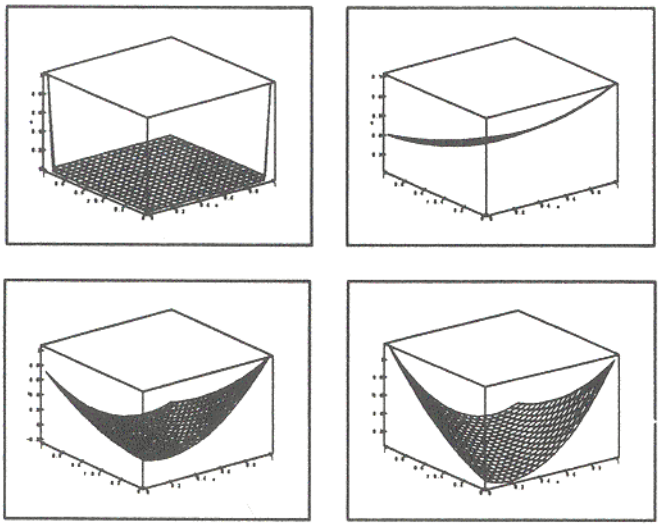

Fig. 1: Evolution of the system; two sigmoidal filters, four samples; top left image shows the original XOR-function; the remaining images, top left linewise to bottom right, show the evolution of the system. Data: 188 iterations, 760 msecs on Sparc 10

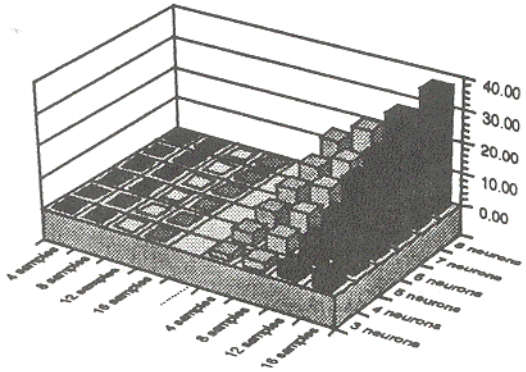

Fig. 2: Time to convergence, averaged over 20 runs each: Projection learning on left, backpropagation on right: Both same initial random weights, convergence time in secs