Unlearning in Feed-Forward Multi-Nets

L. Spaanenburg*
*Rijksuniversiteit Groningen, Dept. of Mathematics and Computing Science, P.O. Box 800, NL-9700 AV Groningen The Netherlands

Abstract
Multi-nets promise an improved performance over monolithic neural networks by virtue of their distributed implementation. Modular neural networks are multi-nets based on an judicious assembly of functionally different parts. This can be viewed as again a monolithic network, but with more complex neurons (the neural modules). Therefore they will share the same learning problems, notably the unlearning effect. In this paper we will look more closely into the reasons for unlearning and discuss how this can be applied to detect novelties.

1 Introduction

Multi-nets are combinations of neural networks [1]. Different names are used depending on the nature of the combination. *Ensemble* networks are based on a redundant collection. Each part gives more or less the solution; together they give the best. In *hierarchical* networks, partial solutions are structurally combined in a "part-of" fashion. The partial solutions are not necessary redundant but a degree of overlap to ensure continuity is well advised. When the combination is of a different nature (child/parent, co-operation, supervision), the notion of *modular* network is used [2].

Multi-nets are of growing interest, as the impact of redundancy is believed to make them more accurate than single-nets. Moreover, multi-nets can be easier to understand and to modify. The current practice in ensemble networks confirms this expectation. Here, the various solutions to the problem are trained independently, then frozen and subsequently combined in a *voting* arrangement. For modular networks in general, the use of frozen parts is less acceptable and the assembly becomes more of a problem.

The monolithic neural network is a combination of neurons with a characteristic transfer function. This function can be explicitly imposed or locally created from a small sub-network. For instance, a sub-network of neurons with linear transfer can behave as a single neuron with a sigmoid transfer. In this sense, the monolithic neural network is already a modular one in disguise and one may therefore expect that the learning problems will be the same. As it has been noted that with growing problem size the monolithic network has increasing difficulty to learn with sufficient quality [3], one may expect no better from a multi-net.

In both cases, the learning process suffers from the entropy in the example set, which in individual cases can lead to catastrophic forgetting [4] (also called interference or cancellation). This can only be resolved by giving suitable general directions, as (a) by data preprocessing, (b) by inclusion of pre-knowledge or (c) by domain structuring. The claim that for monolithic networks more than 80% of the development time is spent on data preprocessing supports this [5].

Modular neural networks address this learning problem by structuring the multi-net into modules, of which some will merely perform functions that are previously part of the data preprocessing. Such modules can often have a strict mathematical content; in this case the multi-net is also called *heterogeneous*. With suitable initialization, the multi-net shows the structure of the domain problem. The major benefit is to the designer, who wants to prototype his solution directly from the raw input data. This enables an immediate view on the problem and its solution by recalling the multi-net.

Apparently modular neural networks may have an improved performance, if the data interference can be solved by a proper transformation of the input data. The further difference with monolithic networks is the facility to segment the domain solution, again reducing the chance on the appearance of learning problems. In a companion paper we discuss how a modular network can be suitably trained. Here, the focus is rather on how unlearning can be utilized as a detection mechanism for novelties.

After an introduction to the nature of data interference, we will illustrate its operation by analyzing the performance of the ill-famed Exclusive-OR [6]. This small and seemingly trivial network displays clearly all the negative effects of data interference without cluttering by other phenomena. Subsequently we move on to techniques to make the network explicitly vulnerable to external aberrations and demonstrate the use of unlearning in the detection of novelties.

2 Catastrophic Forgetting

Symmetries in the operation of neural learning are usually beneficial as the randomized selection of weights together with the random presentation of examples provide enough leeway to eliminate the detrimental side-effects. However, in function

V. Krková et al. (eds.), *Artificial Neural Nets and Genetic Algorithms*

approximation, example randomization brings in the longest-ruin effect: in the long run enough symmetry is available in the presentation set that unlearning can not be excluded beforehand.

First of all, symmetry can already be present in the example set. When for instance the sine-wave y=sin(x) is trained, one example may force the result y_1 to be caused from x_1 in the first quadrant, while another sees y_1 to be caused from x_2 in the second quadrant. The mixed presentation of such signals will cause internal interference as sin(x) = sin(π-x), leading mostly to increased learning time or even erroneous results.

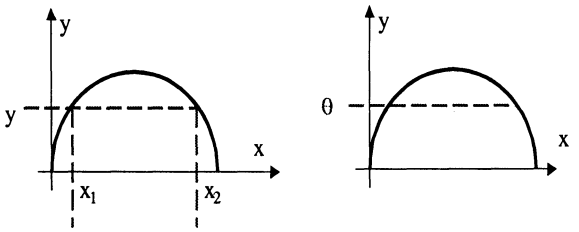

Figure 1 Training a half sine wave.

A second symmetry may come from the architecture of the neural net, more specifically from the discriminatory function that creates the axon value. A number of discriminatory functions have been posed in the past, of which especially the zero-centered sigmoid $f(x) = (1-e^{-x})/(1+e^{-x})$ appears to make the network very vulnerable for symmetry effects. In [4], this discriminatory function has been exploited to make the detrimental occurrence of data interference due to symmetry effects most visible.

The architecture of the fully connected feed-forward net brings a third symmetry as any hidden neuron can be trained to a specific feature. Assuming all required features to be present, the trained function is invariant to the binding of features to hidden neurons. In other words, whereas $y = f(\sum_{k=0}^{n} w_{kj}(f(\sum_{j=0}^{m} w_{ji}x_j)))$, any structural ordering of the m hidden neurons will do.

Connected to the above hidden invariance is the observation that the function of the feed-forward network is based on building solutions from the hidden features. Such hidden features are created in the first layer(s) of the network and do not have to be unique as long as the composition in the output layer provides the correct result. It has been indicated, that from the architecture of the feed-forward net a number of symmetries in the error landscape result that each will provide the desired response on the output [8]. On the other hand, the network may migrate during learning from the one set of hidden features to the other.

The data interference seems caused by indeterminacy and therefore repairable by adding input features. However, the conventional random initialization may often lead to internal indeterminacy which is harder to diagnose and may be facilitated by the many alternatives offered by structural symmetry. The impact of structural alternatives may be offset by realization choices. In [1] it has been shown that the three symmetries will in combination have a severe deleterious effect on the outcome of neural training.

Modules can be used to disperse symmetry in a local and manageable structure. Such symmetries can occur on any level of packaging and will therefore complicate the discussion on modular networks. For the earlier mentioned sine-wave problem, there are a number of suitable, goniometric solutions. The simplest one is to model one quadrant only and to combine this with a phase indicator to model the full signal. However, the potential for unlearning can still be set off by the error disruption caused by module assembly.

3 The XOR Multi-Net

To illustrate the way that data interference leads to unlearning in a modular setting we discuss here a simple experiment, wherein the XOR function is trained directly on a feed-forward network. Though the training is feasible, it is also extremely slow and not guaranteed to terminate. On the other hand, training the AND and the OR function is relatively easy. Then, *connecting* the AND- and OR-modules over a single additional neuron, followed by training for the EXOR, is also relatively easy.

In a way, the difference between the monolithic and the modular approach can be viewed as an initialization problem: by separating in front the two symmetrical parts of the function, the data interference is taken out of the learning. The way we have trained the connected network has been straightforward, as we are able to correctly insert the knowledge in the right place. We start from a trained AND- and OR-function and *merge* them into an overall modular Perceptron by feeding their outputs into a new 2-level module.

When we perform learning on this structure, there appears to go a disruption through the network in the early phases of learning for reason of the sudden contact between pre-trained and "empty" parts. This makes that initially the function output reflects the empty part, while the back-propagated error is intensified when it returns at the interface with the trained parts (Figure 2). Fortunately this disruption soon dies out. We see no difference in the error curve whether the interconnections between the pre-trained functions and the merger are fixed or trainable.

As reference, we have also plotted the learning curve for the monolithic XOR with 4 hidden neurons. Though a 2-2-1 solution is feasible, a 2-3-1 solution is more comfortable to reach by training while 2-4-1 seems optimal. One may assume that the modular construction of a 2-hidden OR and a 2-hidden AND may be equivalent to a 4-hidden monolithic construction, as found earlier to give the optimal response time. Apparently, this is not the case. The learning curve for the 2-4-1 XOR comes only down at around 430 epochs.

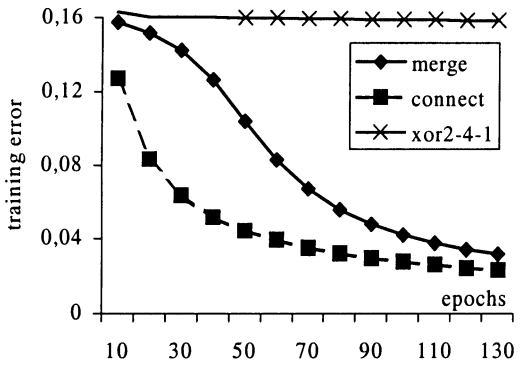

Figure 2 Training with modular inserted knowledge

On closer look, the weight settings have shifted when learning the assembly with trainable interconnect (Figure 3). Though the overall performance does not seem to be affected, the modules are not functionally optimal anymore. For the individual input responses, the change is sometimes for the better, but regretfully the opposite may also be true. It appears that the adaptation is consistent in one direction for those vectors (1='10' and 2='01'), that are redundant for the overall function, and disruptive for the ones (0='00' and especially 3='11'), that are internally conflicting.

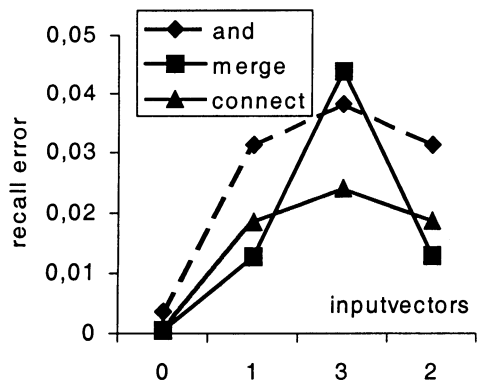

Figure 3 How training of the overall XOR influences the modular parts.

The above example has shown that assembling pre-trained neural modules into a more complex function has two effects. The first is the error disruption caused by the contact between the learned and the unlearned part; the second is the personalization of the modules to the benefit of the overall function. Both effects are potentially harmful to the integrity of the assembly network and the stored knowledge may disappear [9]. Usually this is attacked by considerably lowering the learning rate, leading to lengthy training [10].

The easy alternative seems to assemble only trained networks. In our example, this involves the trained AND- and OR-modules to combine with an XOR-module that was already learned from and(x,y) and nor(x,y) inputs rather than (x,y). However, as training will not be totally perfect, there will remain an error disruption on the interface of the modules. Further it requires complex operations on the data file to produce the partial examples fit for the merged network. The evident success of this approach for real-life problems (for instance [11]) leads to the question whether the modules can be trained simultaneously?

4 Unlearning in Novelty Detection

The desired handling of modular assembly is that the new, unlearned part is first brought into unison with the included knowledge and later on allowed to personalize the module content in such a way that the initial knowledge is always retained and never overruled. In other words, the back-propagation of an error needs to have a different effect on inserted knowledge as compared to fresh ignorance.

The proposition is therefore to have a single learning rate, which is globally effective, and an error-dependent derivation, effective per module. This is a different adaptive behavior than published in [12] where the rate is gradually lowered at the end of learning to improve convergence.

To have a quantitative feeling for the impact of such local learning rates we have first experimented with degrees of error back-propagation on the modular interfaces for the modular EXOR solution. As stated in the discussion of Figure 2, the overall performance will not be effected but the functional correctness of the parts is. Hence in Figure 4 we show the functional correctness of the AND-module after using it in a modular EXOR.

It appears that the amount by which the error is passed back to the learned modules leaves a permanent effect on the initialized knowledge. Even a minimal back propagation sets the destruction process in motion. This can be explained from the observation, that the output reference is lacking for the AND module and therefore even a small error will be used for weight adaptation.

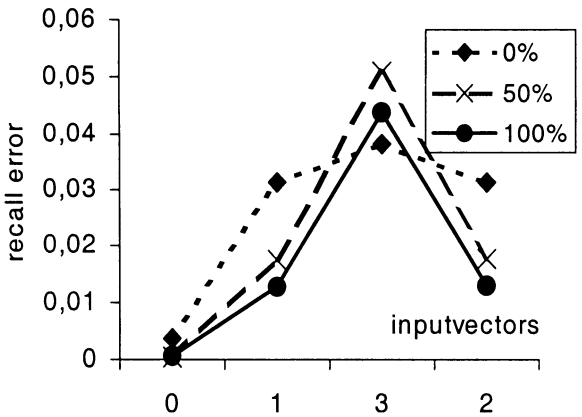

Figure 4 How training of the overall XOR with local learning rates influences the pre-trained AND part.

Where in a companion paper [7] we have continued at this point to find the proper way to learn, the interest lies here in the proper way to unlearn. Apparently, the knowledge within the AND module is affected during the training in a degree, that may cause permanent harm. In Figure 4 there is still room to recover, but at a potentially prolonged learning.

Barakova has already concluded for a range of 9 complex signals [4], that increasing the amount of data interference in the training set has at first little effect. However, learning time increases steeply when the amount crosses a critical value: the *unlearning threshold*. This has been largely explained from the indecisiveness of the learning process.

We conclude here, that even a correct assembly will be vulnerable during post-training. Due to the existing initialization of the parts, the unlearning threshold will have shifted but will still be there. As a consequence, novelties and abnormalities that are large enough to bring the network (or one of its parts) across the threshold will still cause unlearning.

This makes the neural network very suitable for abnormality detection. A neural network trained on existing examples will react on new but different examples by a noticeable prolonged period of post-training. Exceeding the expected training time can be easily measured.

In [13], a comparison is made of several classical and neural fault detectors for a number of signals. It appears that the classical techniques handle random disturbances well, but have a problem with chronic disturbances. On the other hand, a neural network has a degree of robustness for random disturbances, while they react quickly on structural changes: the chronic disturbances.

Some examples are reproduced in Figure 5. They show the learning time in case of five typical noise disturbance: (a) small uniform, (b) medium uniform, (c) block wave, and (d) saw-tooth.

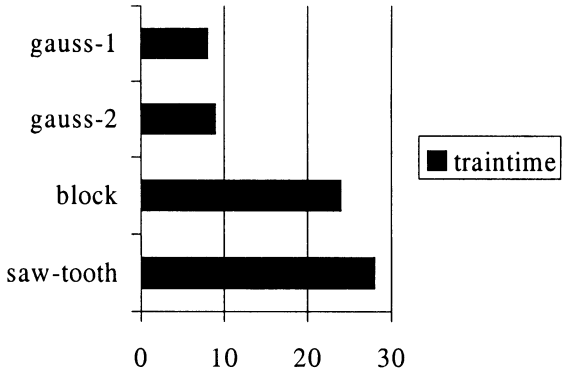

Figure 5 Prolonged training due to disturbance

References

[1] Sharkey, A.J.C. (ed.): Combining artificial neural nets, Heidelberg: Springer 1999.

[2] Caelli, T., Guan, L., and Wan, W.: Modularity in Neural Computing, Proc. of the IEEE *87*, pp. 1497-1518 (1999.

[3] Macready, W.G., Siapas, A.G. and Kauffman, S.A.: Criticality and parallelism in combinatorial optimization, Science *271*, pp. 56-59 (1996).

[4] Barakova, E.I.: Learning Reliability: a study on indecisiveness in sample selection, Ph.D. thesis (Groningen University, Groningen) 1999.

[5] Schuermann, B.: Applications and Perspectives of Artificial Neural Networks, VDI Berichte *1526*, pp. 1-14 (2000).

[6] Sprinkhuizen-Kuyper, I.G., and Boers, E.J.W.: A local minimum for the 2-3-1 XOR network, IEEE Tr. on Neural Networks *10*, pp. 968-971, (1999).

[7] Venema, R.S., and Spaanenburg, L.: Learning feed-forward networks, this proceeding.

[8] Saad, D.: Explicit symmetries and the capacity of multi-layer neural networks, Journal Physics A *27*, pp. 2719-2734 (1994).

[9] Spaanenburg, L., Jansen, W.J., and Nijhuis, J.A.G.: Over multiple rule-blocks to modular nets, Proceedings EUROMICRO'97, pp. 698-705 (1997).

[10] Yamauchi, K., Yamaguchi, Y., and Ishii, N.: Incremental learning methods with retrieving of interfered patterns, IEEE Tr. on Neural Networks *10*, pp. 1351-1365 (1999).

[11] TerBrugge, M.H., Nijhuis, J.A.G., and Spaanenburg, L.: License-Plate Recognition, pp. 263-296, in: Jain, L.C., and Lazzarini, B.: Intelligent Techniques in Character Recognition: Practical Applications, CRC Press 1999.

[12] Yu, X.L., Chen, G.K.C., and Cheng, S.: Dynamic learning rate optimization of the back-propagation Algorithm, IEEE Tr. on Neural Networks *6*, pp. 669-677 (1995).

[13] VanVeelen, M., Nijhuis, J.A.G. and Spaanenburg, L.: Process fault detection through quantitative analysis of learning in neural networks, Proceedings ProRISC'2000, pp. 557-565 (2000).