Constraining of Weights using Regularities

Joost N. Kok[1], Elena Marchiori[0,1,2], Massimo Marchiori[3], Claudio Rossi[4]

[1] *Univ. of Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands*
[2] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
[3] *Univ. of Padova, via Belzoni 7, 35131 Padova, Italy*
[4] *Univ. of Venezia, via Torino 155, 30173 Mestre-Venezia, Italy*

Abstract. In this paper we study how global optimization methods (like genetic algorithms) can be used to train neural networks. We introduce the notion of regularity, for studying properties of the error function that expand the search space in an artificial way. Regularities are used to generate constraints on the weights of the network. In order to find a satisfiable set of constraints we use a constraint logic programming system. Then the training of the network becomes a constrained optimization problem. We also relate the notion of regularity to so-called network transformations.

1. Introduction

The training of a neural network consists of finding a set of weights that minimizes the neural network's error criterion. Most of the standard methods use local gradient search techniques. Often this works well, but there are cases in which it is problematic, for example for recurrent networks, or networks with non-differentiable error criteria. Nevertheless, this type of networks can be very useful in applications (for example in applications based on time sequences). Moreover, a local method can get stuck in local minima. Therefore, alternative approaches based on global optimization methods have been proposed. However, there is a well-known problem with the application of global optimization methods to the training of neural networks. Firstly, these methods can be relatively slow. It depends on the application whether this is a problem or not. Secondly, there is the so-called "competing conventions problem" (see e.g. [3,4]). When one chooses a representation (in this case for a neural network), then it can be the case that the same individual has more than one representation. This enlarges (in an artificial way) the search space. The standard approach is to find a clever representation such that this does not happen. In this paper we take a different approach. We see the problem as a constrained optimization problem, and through the constraints we avoid the "competing conventions problem". First, the notion of regularity is introduced for studying properties of the error function of the networks. Next, these properties are used

[0]The work of the second author was partially supported by SION, a department of the NWO, the National Foundation for Scientific Research, The Netherlands

in the development of suitable algorithms for the generation of constraints on the weights of the network. It is of interest to relate the notion of regularity to transformations of networks. Due to the formal analysis we get insight in the "competing conventions problem": caused by regularities, or dually, by network transformations. Moreover, the constraints give us a way to see how the competing conventions problem works out in practice: we can do optimizations with and without the constraints that avoid the "competing conventions problem". Then we propose to use a framework based on a constraint logic programming system in order to generate by means of the constraint logic program the constraints for the competing conventions problem and to integrate them with other constraints (for example shared weights, domain constraints etc.). This system checks for the satisfiability of the constraints. Then a global optimization method (in this paper a genetic algorithm) is used for finding the optimal set of weights.

2. Regularities of the Error Function

We are interested in the derivation of suitable constraints over the connection weights that avoid the "competing conventions problem". To localize these constraints we investigate ***regularities*** in the error function of the network. The error function $E(\tilde{w})$ maps the sequence $\tilde{w}$ of the network's weights into a real number which measures the discrepancy between the outcome of the network and the 'ideal' outcome for example given by a training set. The aim is to find an optimal sequence of weights, i.e., a suitable sequence $\tilde{w}$ which minimizes $E(\tilde{w})$ (we view the biases as weights). We do not put any restrictions on the error criterion.

First, we define a convenient syntax for expressing regularities. Consider the set $\mathcal{T}(Var, Fun)$ of terms, denoted by s, t, also possibly subscripted, with Fun the standard collection of constants (e.g. 1, 3.5, etc.) and functors (e.g. $+$, $\cdot$, $|\ |$, etc.) for the real numbers, and with Var a set of variables, denoted by $x, y, z, \ldots$ possibly subscripted. A sequence $s_1, \ldots, s_n$ of terms is also denoted by $\tilde{s}_n$, while C denotes a sequence of constraints on $\mathcal{T}(Var, Fun)$, like the ordering relations on real numbers (equality, disequality, $>$, $<$, $<=$, $>=$), and is identified with the conjunction of its elements.

Definition 2..1 (Regularities) An $2n$-tuple $(\tilde{s}_n, \tilde{t}_n)$ of terms in $\mathcal{T}(Var, Fun)$ together with a sequence $C(\tilde{s}_n, \tilde{t}_n)$ of constraints on the variables of the tuple is called a *regularity w.r.t. C (of rank n)*. An *error function* $E(w_1, \ldots, w_n)$ *satisfies* the regularity $(\tilde{s}_n, \tilde{t}_n)$ if $\models \forall\,(C(\tilde{s}_n, \tilde{t}_n) \rightarrow E(\tilde{s}_n) = E(\tilde{t}_n))$, where $\forall$ quantifies over all the variables of the terms, and where $\models$ stands for the logical consequence w.r.t. the standard theory of the real numbers. □

Note that we use universal quantification over the variables, in particular over the weights. For instance, for a real number $r > 0$, the regularity $(x_1, x_2, x_1 + r, x_2 + r)$ w.r.t. $x_1 \geq 0, x_2 \geq 0$ specifies that the error function E is periodic on the domain of the non-negative reals. In the sequel we shall omit C if it is the empty sequence (corresponding to *true*). Moreover, we shall focus on regularities whose first n terms are distinct variables, denoted by $(x_1, \ldots, x_n, t_1, \ldots, t_n)$. We treat four kinds of regularities: for each kind of regularities, we study possible constraints on the weights which can be derived from the regularities in order to restrict the search space for the training of a neural network. We also give for each regularity a "neural" example, and show how a network that does not satisfy the constraints, can be transformed into a network that does satisfy them.

Equality Regularities An *equality regularity* is a regularity such that at least two of the t_i's coincide. For instance, for a binary error function E the regularity (x_1, x_2, x_1, x_1) states that the value of E does only depend on the first argument of the error function, that is, for all real numbers r, r' we have that $E(r, r')$ is equal to $E(r, r)$. From this regularity one can derive the equality constraint $x_1 = x_2$. In this way, we constrain the value of the second argument to be the same as the first argument. For a neural network, this kind of regularity can stem from the fact that a part of the network does not contribute to the outcome (i.e. for example a part of the network is disconnected). Assume that we have a network with a disconnected part which does not satisfy the constraints. Then we can transform the network by replacing the weights in the disconnected part by other weights, such that the equality constraints are satisfied.

Product Regularities Let α denote a term of $\mathcal{T}(Var, Fun)$. A *product regularity* with respect to α is a regularity in which some of the t_i's are of the form $\alpha \cdot x_i$. For instance, for a binary error function E the regularity $(x_1, x_2, x \cdot x_1, x_2)$ w.r.t. $x > 0$ says that the value of E does depend on x_2 and only on the sign of x_1. The constraints on the weights that one can derive are of the form $x_1 = r$, where r is any positive real number.

For instance, consider in a neural network a unit u that has a hard limiting transfer function, say $g(x) = 1$ if $x > 0$ and $g(x) = -1$ otherwise. Then the error function has this kind of regularity, because we can multiply all the incoming weights with a positive number, and we do not change the output of the unit. If we have such a unit in a network that does not satisfy the constraint, then we can scale (by dividing by $\frac{|w|}{r}$), all the weights of the incoming arcs of the unit. In this way the selected weight w becomes either equal to r or to $-r$. This network transformation does not affect the outcome of the neural network, and hence not the error function.

Inversion Regularities Another type of regularities are the so-called *inversion regularities*, where for $1 \leq i \leq n$, the term t_i is either equal to x_i or to $-x_i$, and at least one of the t_i is equal to $-x_i$. The constraints on the weights one can derive from these regularities fix the sign of one x_i either by the constraint $x_i \geq 0$ or by $x_i \leq 0$.

Such regularities appear in the error function of neural networks with units with odd transfer functions. A transfer function g is odd if for all real numbers r, $g(r) = -g(-r)$. A typical example is a sigmoid transfer function. If we have a network with node u that has an odd transfer function, and that does not satisfy the constraints, then we we can transform the network by changing the sign of the weights of all the input and output arcs of u. This transformation does not effect the outcome of the neural network, hence it does not effect the error function.

Permutation Regularities Another class of regularities are the permutation regularities. Let σ be a permutation of $1, \ldots, n$. Then $(x_1, \ldots, x_n, x_{\sigma(1)}, \ldots, x_{\sigma(n)})$ is called a *permutation regularity* (w.r.t. σ). The constraints that one can derive from these regularities enforce a total ordering on the elements that can be permuted.

For example, such regularities appear in the error function of a feedforward neural network, where we can permute hidden units of the same layer. Here groups of weights in the network can be exchanged without changing the error function. We can give a total ordering on these groups of weights as follows. Assume that we have a function, that given the incoming and outgoing weights of a node, yields a real number. Then we can order the nodes according to these numbers. The function can be for example, the weight on the leftmost incoming weight, but also the sum of all the weights. The first possibility yields constraints that for each layer, except the output layer and its predecessor, enforce an ordering on the weights of the output arcs of *one* unit. Note that if a network does not satisfy such constraints, then we can permute nodes such that it does satisfy them. This kind of transformations does not affect the outcome of the network, hence it does not effect its error function.

For neural networks, the effects of these regularities can be exponential. For example, the inversion regularities can enlarge the search space by a factor 2^n, and the permutation regularities by a factor $n!$, where n is the number of hidden nodes.

3. Training of Constrained Neural Networks

The previous section showed how regularities are related to constraints on the weights of a neural network. In this section we present some algorithms for the generation of such constraints. We consider permutation regularities and introduce two algorithms for generating constraints on the weights of a feedforward network, and two algorithms for recurrent networks. These algorithms have been implemented as constraint logic programs in the programming language ECL^{i}PS^e (ECRC Common Logic Programming System), and are integrated together with a genetic algorithm in a single framework. We discuss some results on the training of neural networks performed using this framework.

The following two algorithms generate constraints on the weights of a feedforward network in order to fix its geometry. They yield constraints for the permutation regularities, and are based on the ideas sketched in the subsection on permutation regularities. For each layer, the first algorithm acts on the weights of the output arcs of one unit, while the second algorithm acts on the weights of more than one unit. Observe that we assume that the weights of the network can already be subject to some given constraints.

Algorithm 1 For every layer, except for the last two (i.e. the output layer and its predecessor) do:
1. find a unit u of L and let $w_1, \ldots, w_n$ be the weights of its output arcs;
2. find a permutation σ of $1, \ldots, n$ s.t. the constraint $w_{\sigma(1)} < \ldots < w_{\sigma(n)}$ is satisfiable.

A more general algorithm is the following one:
Algorithm 2 Let f be a function from sequences of real numbers to real numbers. For every hidden layer $L = (u_1, \ldots, u_n)$ do:
1. for every unit u of L let W_u denote the value of f when applied to the sequence of weights of all its incoming arcs;
2. find a permutation σ of $1, \ldots, n$ s.t. the constraint $W_{u_{\sigma(1)}} < \ldots < W_{u_{\sigma(n)}}$ is satisfied.

We now generalize these two algorithms to recurrent networks.
Algorithm 3 Consider the network obtained by dropping the self-cycling arcs, that is arcs going from a unit to the same unit. Mark the input units. Then, repeat the following procedure, until all units are marked:
1. select an unmarked unit u;
2. take a unit u' such that there is an arc going from u' to u;
3. consider all the arcs going from u' to unmarked units, mark the units; let $w_1, \ldots, w_n$ be the sequence of the corresponding weights;
4. find a permutation σ of $1, \ldots, n$ s.t. $w_{\sigma(1)} < \ldots < w_{\sigma(n)}$ is satisfiable.

Algorithm 4 Let f be a function from sequences of real numbers to real numbers. Let $(u_1, \ldots, u_n)$ be all the hidden units of the network.
1. for every unit u let W_u denote f applyed to the sequence of weights of all its incoming arcs;
2. find a permutation σ of $1, \ldots, n$ s.t. the constraint $W_{u_{\sigma(1)}} < \ldots < W_{u_{\sigma(n)}}$ is satisfiable.

Note that the previous algorithms generate strict orderings on some weights. However, one can replace $<$ by $\leq$ in those cases where the resulting constraints are not satisfiable.

We give now some results on the training of neural networks with or without constraints. For this we use a framework [1] which integrates the constraint logic programming ECL^{i}PS^e by ECRC for the generation of constraints, and GENOCOP [2] as a global optimizer.

We did two series of experiments. The first series on a standard data set about Iris flowers, a classic pattern recognition problem with four parameters describing an Iris flower. We used a feedforward neural network with four input units, four hidden units and three output units. Figure 1 shows a typical run with error function the sum over the absolute values of errors. The addition of the "regularities" constraints improves the convergence. We have considered also runs with different error-criteria, namely the standard squared sum of errors, and the sum over the absolute values of the cube of the errors, and obtained analogous results.

The second series of experiments employs a data set containing the sine of the natural numbers $1, 2, \ldots, 50$. The neural network is used to predict the

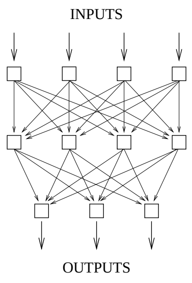
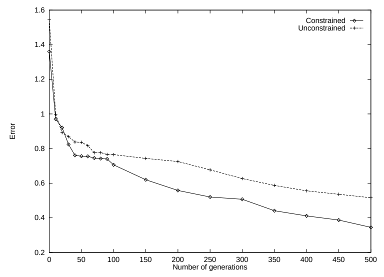


Figure 1: The network for the Iris



A typical run

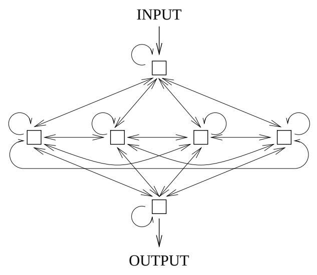
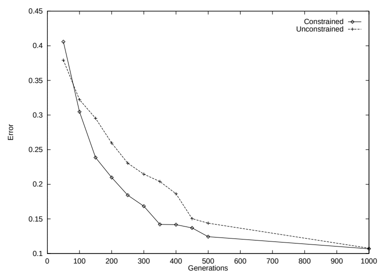


Figure 2: The network for the ST



A typical run

next value. In the experiments we used a recurrent network with one input, four hidden units and one output unit: the transfer function is the sigmoid on $[-1, 1]$ for the hidden units, and a linear one for the output. A typical run of the system with standard error function is shown in Figure 2. We also see here that the "regularities" constraints improve the convergence.

References

[1] J.N. Kok, E. Marchiori, M. Marchiori, and C. Rossi. Evolutionary training of CLP-constrained neural networks. In *Proc. PACT'96*.

[2] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1994.

[3] H.J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5:589–593, 1992.

[4] D. Thierens, J. Suykens, J. Vanderwalle, and B. De Moor. Genetic weight optimization of a feedforward neural network controller. In *Proc. of the Conf. on Artificial Neural Nets and Genetic Algorithms*, pages 658–663. Springer-Verlag, 1993.