DISCRETE NEURAL NETWORKS AND FINGERPRINT IDENTIFICATION

Steen Sjøgaard
Computer Science Department, Aarhus University
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark
Fax: +45 86 135725
e-mail: steensj@daimi.aau.dk

Abstract. For a large volume of real-world applications of artificial neural networks, computational speed is a crucial factor which often is decisive of success or failure. Especially for real-time applications, a network's ability to quickly and correctly classify as many input stimuli as possible is of vital importance. It is therefore desirable to be able to construct neural networks that are suited for hardware implementation. The notion of *discrete neural networks* (DNN) and the methods for discretization of ordinary or *real neural networks* (RNN) introduced below, fulfill this desire. Furthermore, it is demonstrated how the methods have been successfully applied to the realistic and nontrivial task of fingerprint identification.

DISCRETE NEURAL NETWORKS

Basically, the process of discretizing a RNN involves two steps: (1) discretization of the real solution, and (2) discretization of the matching network itself.

Discretization of a Real Solution

If we contemplate the error landscape (i.e. error-weight space) for a specific network configuration, the best *real solution* is always at the bottom of the deepest valley. If we now construct a grid and superimpose it on the error landscape, we may consider all the grid vertices as the only legal (discrete) weight values. See Figure 1. We shall define a *discrete solution* belonging to a given RNN as a point in weight space consisting of only such vertex determined weight values. For such a discrete solution and the matching network it is now clear that the density of the grid will determine the precision of the weight values and consequently the network error: A very fine-masked

 0-7803-0557-4/92$03.00 © 1992

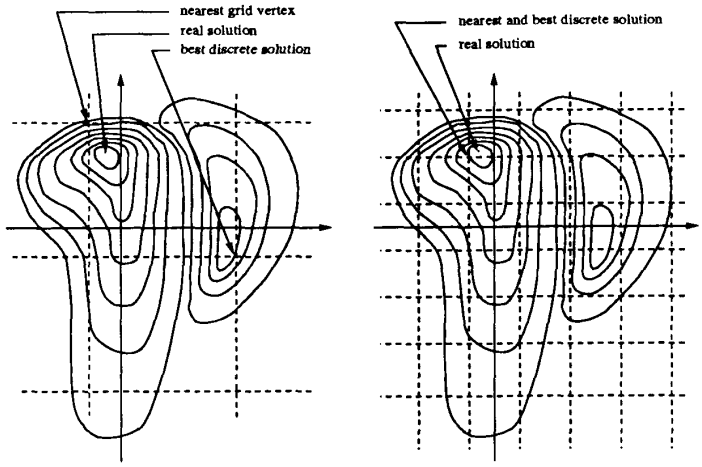

Figure 1: Two different grid densities superimposed on a contour-map of an error landscape containing two valleys.

grid will allow a high precision, while a big-masked grid can only allow a poor precision. Besides, as we can see in Figure 1, a big-masked grid may give a false impression of valley depths and the few number of grid vertices may even result in an exclusion of some valleys.

On the other hand, the overall idea of discretization is to achieve a significant computational speedup. This implies, as we shall see in a little while, that as few distinct weight values as possible should be considered, and thus that the grid density should be as small as possible. Therefore we have a tradeoff between network performance on the one hand, and speedup on the other. This tradeoff we shall refer to as the *error-density relationship*. Unfortunately, it is composed of two interdependent factors which impedes a closer examination of the relationship:

1. Given that we always apply a particular grid vertex relative to the real solution, how fine must the density of the grid be before the discrete solution is acceptable.

2. Given the density of the grid, which of the surrounding vertices is 'best,' i.e. which one gives the best network performance.

However, a series of empirical investigations [1] have shown that the transformation of a real solution into a discrete solution demands only a small series of experiments with distinct grid resolutions. And for each resolution examined, one should always use the grid vertex nearest to the real solution (in Euclidean distance) as the discrete solution belonging to that particular resolution. And then the final selection of the best of these resolution-

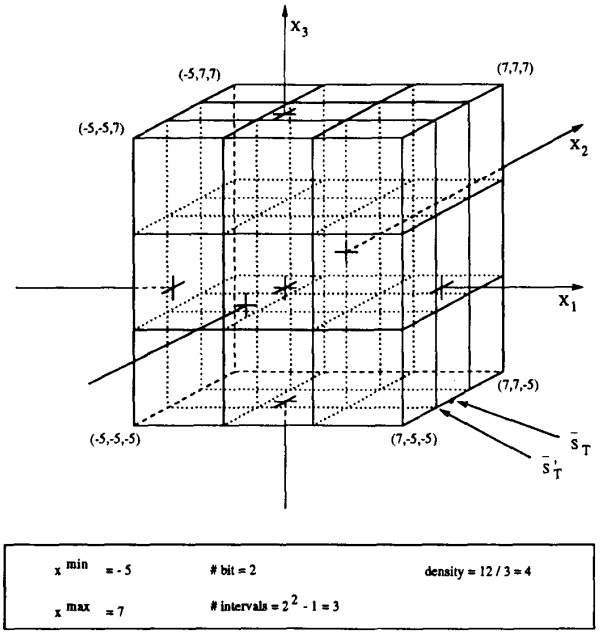

Figure 2: Example of a 3 dimensicnal initial hypercube.

specific solutions should be based on a comparison of the matching network performances. Therefore, given a specific grid resolution, we will now exclusively focus on the transformation of a real solution into the nearest discrete solution.

Let $\mathcal{N}$ be an arbitrary multi-layered feedforward network, and $w_{i,j}$ the weight value from unit j to unit i. The state of this network at time t can be described by a d dimensional vector $\vec{s}_t \in S \subseteq \mathbb{R}^d$, where S is the weight space. Now, let $\vec{s}_T$ denote the real solution belonging to $\mathcal{N}$ for a particular problem. Furthermore, let x_{min} (x_{max}) be the coordinate of the vector $\vec{s}_T$ which is smallest (largest), i.e. if we consider all the d coordinates of $\vec{s}_T$ as real numbers and line them up on the real axis, they will all lie in the interval from x_{min} to x_{max}. The idea with discretization is now to split this large interval into a number of equal sized subintervals and then to round off all the coordinates to the nearest endpoint of the subinterval in which they lie.

First, we define the d dimensional *initial hypercube* H^d by

$$H^d = \{ (x_1, x_2, \cdots, x_d) \in S \mid x_i \in [x_{min}, x_{max}], \ 1 \leq i \leq d \}$$

and since the edge length of H^d is defined by x_{min} and x_{max}, the real solution $\vec{s}_T$ will always lie on one of the edges in this cube. Then, by splitting

each of the edges in the initial hypercube into I equal sized sub-intervals we can now fill out this initial hypercube with C smaller hypercubes, where $C = I^d$. This is illustrated in Figure 2 for $I = 3$ and $C = 3^3 = 27$. Now the real solution $\vec{s}_T$ is inside at least one of these smaller hypercubes and the vertex nearest to the real solution in that hypercube shall be the discrete solution $\vec{s'}_T$ we are looking for. Formally, $\vec{s'}_T$ can be found as follows: Suppose that we want to use γ bit to enumerate 2^γ discrete weight values. Then let

$$\begin{aligned} edgel &= \tfrac{x_{max}-x_{min}}{2^\gamma-1} \\ displ &= x_{max} - \text{TRUNC}\left(\tfrac{x_{max}}{edgel}\right) \cdot edgel \end{aligned}$$

where *edgel* is the length of the small surrounding hypercube, and *displ* is the positive end-point of the sub-interval which contains zero.

If x_i (x'_i) is the i'th coordinate of $\vec{s}_T$ ($\vec{s'}_T$) respectively, then

$$x'_i = \begin{cases} \text{TRUNC}\left(\frac{x_i - displ}{edgel} + \frac{1}{2}\right) \cdot edgel + displ & \text{if } x_i \geq displ \\ \text{TRUNC}\left(\frac{x_i - displ}{-edgel} + \frac{1}{2}\right) \cdot (-edgel) + displ & \text{otherwise} \end{cases}$$

This guarantees that we find the vertex nearest to the real solution in the surrounding hypercube.

Discretization of a Real Neural Network

Given a discrete solution $\vec{s'}_T$ we will now derive a method for hardware representation of of the whole adapted DNN. Let u_i be an arbitrary unit in $\mathcal{N}$, input units excluded. Then the computations carried out by u_i can be described by

$$o_i = f(net_i) = \frac{1}{1 + \exp(-\frac{1}{T} net_i)} \tag{1}$$

where $net_i = \sum_j w_{i,j} o_j$, and T is a (temperature) parameter determining the steepness of the Sigmoid function f. To implement these computations in hardware, we have to transform the real functions into appropriate discrete functions. As all the units in the network apply exactly the same output function f, we intend to implement f as a table, i.e. as a discrete step function. First, we define two quantities net_{min} and net_{max} to reduce the 'interesting' input area to a limited interval:

$$f(net) = f(net_{min}) = 0 \quad \text{for } net \leq net_{min}$$

$$f(net) = f(net_{max}) = 1 \quad \text{for } net \geq net_{max}$$

Then, assume that we want to apply:

- α bits to represent 2^α discrete output values in $I_{output} = [0, 1]$
- β bits to represent 2^β discrete input values in $I_{input} = [net_{min}, net_{max}]$

The 2^{α} possible discrete output values $\overline{o}_j$ for unit j can now be defined as

$$\begin{aligned} \overline{o}_j &= \mathcal{A} \cdot o'_j \\ &= \frac{1}{2^{\alpha}-1} \cdot o'_j \end{aligned} \tag{2}$$

where $o'_j = 0, 1, 2, \cdots, 2^{\alpha}-1$.
And similarly the 2^{β} discrete input values $\overline{net}_i$ for unit i are defined as

$$\begin{aligned} \overline{net}_i &= \mathcal{B} \cdot net'_i + net_{min} \\ &= \frac{net_{max} - net_{min}}{2^{\beta}-1} \cdot net'_i + net_{min} \end{aligned} \tag{3}$$

where $net'_i = 0, 1, 2, \cdots, 2^{\beta}-1$.
From the previous section we know that the 2^{γ} discrete weight values can be defined similarly

$$\begin{aligned} \overline{w}_{i,j} &= \Gamma \cdot w'_{i,j} + w_{min} \\ &= \frac{w_{max} - w_{min}}{2^{\gamma}-1} \cdot w'_{i,j} + w_{min} \end{aligned} \tag{4}$$

for $w'_{i,j} = 0, 1, 2, \cdots, 2^{\gamma}-1$.

Suppose now that $\widehat{net}_i$ is a given input value to unit u_i. We may then write

$$f(\widehat{net}_i) = f(\mathcal{B} \cdot k + net_{min}) = f(\overline{net}_i)$$

for all $\widehat{net}_i \in k$'th subinterval in I_{input}. Thus, by combining Equation 1 and 3 we obtain

$$\begin{aligned} f(\overline{net}_i) &= f(\widehat{net}_i) \\ &= f(\sum_j \overline{w}_{i,j}\overline{o}_j) \\ &= f(\mathcal{A}\Gamma \sum_j (w'_{i,j} + \frac{w_{min}}{\Gamma}) o'_j) \end{aligned}$$

Since $\mathcal{A}$, Γ and w_{min} are constants, and o'_j and $w'_{i,j}$ are integers we can now derive a new discrete output function g

$$g(netstim_i) = \frac{1}{1 + \exp(-\frac{\mathcal{A}\Gamma}{T} netstim_i)} \tag{5}$$

where

$$netstim_i = \sum_j (w'_{i,j} + \frac{w_{min}}{\Gamma}) o'_j$$

which implies

$$f(\overline{net_i}) = g(netstim_i)$$

And as the computation of $netstim_i$ can be carried out very quickly using only a few number of bits, we now have an effective discrete representation of the output function in the shape of g.

A CASE STUDY: FINGERPRINT IDENTIFICATION

The methods described above have been successfully applied to the non-trivial task of fingerprint identification. A large number of experiments with different network configurations and distinct preprocessing techniques were conducted, and as outcome a two-layer feedforward network with 49 input, 20 hidden, and 16 output units was selected [2]. For training and test sets a total of 1696 patterns were scanned. 1139 of these were used for network training, while the remaining 557 patterns comprised the test set. The found real solution consisted of 1336 weight values, and by using the discretization method described above it turned out that $\gamma = 6$ bits were sufficient to transform the real solution into just $2^6 = 64$ discrete weight values. Moreover, experiments with α and β showed that the table version of the Sigmoid function could be represented by just 16 discrete input and output values, i.e. $\alpha = \beta = 4$ bits were sufficient. A comparison of the performances of the networks using the different solutions is shown in Table 1 below:

Network	Solution	Pct. correctly classified patterns
Real	Real	92.8%
Real	Discrete	92.8%
Discrete	Discrete	93.0%

Table 1: Network performances.

CONCLUSION

We have developed a general method for discretization of feed-forward neural networks. Furthermore, we have empirically demonstrated the usefulness of the method by successfully applying it to the nontrivial task of fingerprint identification. Surprisingly, the discrete neural network developed in this way demanded just 4 bits for the table representation of the Sigmoid function, and likewise only 6 bits for the representation of the matching discrete solution. Moreover, Table 1 clearly shows that there is no significant difference in the performance on the test set between the RNN and the DNN. Thus, we conclude that the discretization methods we have proposed have shown themselves to be realistic.

REFERENCES

[1] S. Sjøgaard, Fingerprint Recognition and Discrete Neural Networks, Technical Report, DAIMI IR–108, Aarhus University, January 1992, pp. 18–28.

[2] S. Sjøgaard, Fingerprint Recognition and Discrete Neural Networks, Technical Report, DAIMI IR–108, Aarhus University, January 1992, pp. 39–80.