PROGRAMMABLE TRANSFER FUNCTIONS FOR NEURAL NETWORKS*

Stephen W. Soliday

North Carolina Agricultural and Technical State University
Department of Electrical Engineering / MIPAR†

Abstract

Implementing neural networks in the computational environment of computing requires a careful analysis of loop optimization. The transfer function of the network's processing elements, and their associated derivatives, should be designed in such a way that it is both fully programmable and contain as few in-loop floating point operations as possible. The purpose of this discussion is to analyze some of the well behaved transfer functions traditionally used in fully interconnected networks, and to derive a faster transfer function that meets the above requirements.

KEYWORDS: Neural–Network, Transfer–Functions, Instruction–Length Floating–Point, Loop–Optimization.

1 INTRODUCTION

Fully interconnected networks, such as the BPN[1], require that both the processing element's transfer function and the first derivative of that transfer function undergo massive repetitions.

As an example, a BPN with four input layer PE's[2], eight first hidden layer PE's, five second hidden layer PE's and three output layer PE's; will perform the transfer function a minimum of twenty times per iteration. Now multiply that against 100 samples of training data and 50 samples of test data and you have now performed the transfer function 3000 times for just one pass of the training cycle. In a similar manner it can be determined that it was necessary to perform the derivative sixteen times for each training sample, and 1600 times for each training cycle. A typical training session for a BPN might contain $20,000$ cycles of training before a minimum mean square error for the testing set of data may be found. This means that the transfer function will have been performed 60 million times and its derivative will have been used 32 million times.

As a second example, consider a similar three layer network performing a low resolution image transformation[3]. There are 256 by 256 pixels in the input image and the same in the output image. The two hidden layers also contain the same number of pixels as the two images. For just one sample of training data the transfer function will be used $262,144$ times and the derivative $196,608$ times. For a training cycle of twenty training images and five test images the transfer function will be used 6.5 million times and the derivative 983 thousand times for just one of possibly $20,000$ training cycles.

2 REVIEW OF BPN

The traditional back propagation network consists of two or more layers of fully interconnected processing elements. A connection represents a weight or scale between the output of one PE and the input of another. Each processing element makes a weighted sum of the outputs from the previous layer. These pairs are the output of

*This paper appeared in the First Industry/Academy Symposium on Research for Future Supersonic and Hypersonic Vehicles, vol 1, pp 142–147, Dec 1994

†MIPAR: Machine Intelligence and Power Associated Research

[1] BPN: Back Propagation Network

[2] PE: Processing Element

[3] Only layers that contain fully interconnected weights and which under go training are considered layers.

a previous layer's PE and the weight associated with that connection. This sum then becomes the independent variable of a transfer function. The output of the transfer function then becomes the output of that PE. The input layer is not usually fully connected, instead the transfer function is applied to each input Xn and this becomes the output for the input layer PE's. See [Wasserman], for a complete description of the back propagation network and its training.

3 TRANSFER FUNCTION

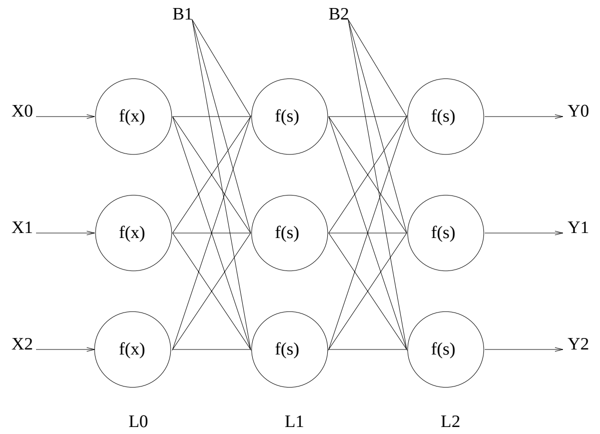

Figure 1: Typical BPN

It can be clearly seen that fast executing transfer functions and derivatives must be used. The transfer function and its derivative must have certain properties. First, the transfer function must be a well behaved monotonically increasing, bounded function, and second, its first derivative should be continuous. In networks such as the BPN each processing element is connected to every processing element in the previous layer and every processing element in the forward layer, Figure1. In an application such as the image processing example discussed previously, a very large number of previous–layer outputs are being summed, leading to very large sums. In the course of training the weights can sometimes become quite large. If the transfer function did not have a bounded output it is clear that a runaway effect or a combinatorial explosion would occur after several layers. Rridiculously large if not infinite results would be produced. To prevent this, transfer functions whose outputs are bounded are needed. The two most common functions are the logistic function and the hyperbolic tangent.

$$Y = \frac{1}{1+e^{-X}}, \quad \text{and} \qquad Y = \tanh(X) \tag{1}$$

Equation 1a is bound at $\pm\infty$ by 1 and 0 respectively. Equation 1b is bound at $\pm\infty$ by 1 and -1 If Equation 1a is scaled to match the bounds of Equation 1b then its form becomes:

$$Y = \frac{2}{1+e^{-X}} - 1 \implies \frac{e^X - 1}{e^X + 1} \tag{2}$$

Equation 1b may be placed in its exponential form:

$$Y = \frac{e^{2X} - 1}{e^{2X} + 1} \tag{3}$$

It can be seen that both equations are of the form:

$$Y = \frac{e^{2mX} - 1}{e^{2mX} + 1}, \quad \text{and} \qquad \frac{\partial Y}{\partial X} = \frac{4me^{2mX}}{(1+e^{2mX})^2} \tag{4}$$

Where ($m = 0.5$ for the logistics and $m = 1.0$ for the hyperbolic tangent) m is the slope of the transfer function as it passes through its point of inflection. The bounds, point of inflection and slope at the inflection point should be programmable. This is accomplished through scaling and offset. In Section5. this process will be examined in detail.

4 ANALYSIS OF FLOATING POINT

Before actually designing a programmable transfer function, the way a computer computes basic mathematical functions should be addressed. Most FPU's[4] do not calculate the natural exponent directly[5]. For example, in the Intel 387 math coprocessor, the machine instructions necessary for calculating the natural exponent are a combination of $2X - 1$ and $Log_2 X$ denoted as **F2XM1** and **FYL2X** respectively[Cyrix]. Using the log identity:

$$X = \exp\left[Y \cdot \log_e [X]\right] \tag{5}$$

And in FPU notation the natural exponent may be calculated.

$$EXP[X] = FAAD\left[F2XM1\left[FYL2X[X,1]\right],1\right] \tag{6}$$

There is however a single machine instruction for determining what is known as a partial arc tangent. That is where the arc tangent is determined from Y and X and only returns the angle in an interval between $-\pi$ and π. Heuristics are used to determine the proper quadrant. In a program, AppendixA, setup to time the number of function calls made per second, it was determined that **ATAN** out performed both the **TANH** and the **LOGISTIC** (Equations2 and3) in calls per second. Equation4 shows that the **TANH** and the **LOGISTIC** equations are for all purposes equal in computation speed.

5 DESIGN

As seen in Section4, the arc tangent function is the clear choice *based solely on its calculation speed.* The arc tangent function must now be scaled and offset to meet the requirements discussed in Section3.

$$Y = \tan^{-1}(X) \tag{7}$$

The first condition is that the function must be well behaved. The arc tangent function is continuous for all real numbers $(-\infty \leq X \leq \infty)$. Its output is finite $\left(-\frac{1}{2}\pi \leq Y \leq \frac{1}{2}\pi\right)$ and the function should be monotonically increasing to assure smooth response to gradient decent The second condition is that the first derivative must also be continuous and bounded. The first derivative of the arc tangent is:

$$\frac{\partial Y}{\partial X} = \frac{2}{1+4X^2} \tag{8}$$

The added bonus of the arc tangent function can be seen in its first derivative. There are no function calls to be made. It should be noted at this point that the derivative of the logistics based function may be determined without function calls. The trade off is maintaining a copy of the outputs of all of the processing elements. As is usually the case you must sacrifice storage for speed. If Equation1 is written as:

$$f(X) = \frac{1}{e^{-2mX}+1} \tag{9}$$

Then its derivative must may be written as:

$$\frac{\partial}{\partial X} f(X) = 2mf(X)\left(1 - f(X)\right) \tag{10}$$

This method performs one less multiplication instruction than Equation8, however it will not be programmable in the sense of Section5, and it requires that storage be allocated to save the outputs of the PEs during the forward pass of the network. This may not be a storage consideration in control problems where the typical network has a few layers of fifteen or twenty PEs each. When you get into image processing problems the number of PEs can

[4]FPU: Floating Point Unit or Emulation software

[5]The one exception found was the DEC RISC on the DECStation 5000, see Table2 in the appendix

typically be 196608. That amounts to 0.78 megabytes for single precision storage and 1.6 megabytes for double precision storage. Equation8 requires no additional storage for saving outputs.

Care must be taken with the X^2 in the denominator of Equation8. Mathematically this equation is bounded to zero at ∞, but computationally there is the chance of receiving an over-flow error when the X^2 is calculated for very large values. Likewise, if an over-flow is not encountered, the results of the X^2 computation could be so large as to produce an under-flow error when the division is performed. This exception may be handled in the same way the built in intrinsic functions handle bounded equations. A finite representation of ∞ must be declared. Returning a 0 if $|X| = 10^{10}$ for example would solve the problem.

The equation must be offset and scaled in order to meet the condition of programmability. The following parameters will be introduced:

$$Y = f\left(X, X_o, A, B, \alpha\right) \tag{11}$$

X the independent variable representing the input.

X_o constant representing position with respect to the independent variable of the point of inflection.

A constant representing the upper bound of Y.

B constant representing the lower bound of Y.

α constant representing the slope of the transfer function at the point of inflection.

The position of the point of inflection with respect to the dependent variable will always be the midpoint between A and B. Thus manipulated the transfer function and its first derivative will be:

$$Y = \tan^{-1}\left[\frac{\alpha\pi}{A-B}\left(X - X_o\right)\right]\left(\frac{A-B}{\pi}\right) + \left(\frac{A+B}{2}\right) \tag{12}$$

And:

$$\frac{\partial Y}{\partial X} = \frac{\alpha\left(A-B\right)^2}{\left[\left(A-B\right)^2 + \pi^2\alpha^2X_o^2\right] + X\left(\pi^2\alpha^2X - 2\pi^2\alpha^2X_o\right)} \tag{13}$$

Now the task of removing all the constant sub-expressions can be handled. Any time equations are placed in a loop or are likely to under go repetitive calls, all constant sub expressions should be removed. This reduces the number of math instructions that are performed on each iterative step, increasing the speed of execution, and in most cases, reducing the size of the executable. One example of the two final equations could be:

$$Y = P_1 \cdot \tan^{-1}\left(P_2 \cdot X + P_3\right) + P_4, \text{ and } \quad \frac{\partial Y}{\partial X} = P_5/\left(P_6 + X \cdot \left(P_7 \cdot X + P_8\right)\right) \tag{14}$$

$$\begin{array}{lcllcl} P_1 & = & \left(A-B\right)/\pi & P_2 & = & \alpha\pi/\left(A-B\right) \\ P_3 & = & -\alpha\pi X_o/\left(A-B\right) & P_4 & = & \left(A+B\right)/2 \\ P_5 & = & \alpha\left(A-B\right)^2 & P_6 & = & \left(A-B\right)^2 + \alpha^2\pi^2X_o^2 \\ P_7 & = & \alpha^2\pi^2 & P_8 & = & -2\alpha^2\pi^2X_o^2 \end{array} \tag{15}$$

These two equations represent the smallest number of in-loop calculations needed to perform the transfer function and its derivative. The constant expressions P_n would be performed only once in a setup routine. These constants represent a parameter list and may be stored either as global variables or more preferably in a data structure that would either be a member of an OOP object or passed as the argument to the transfer function[6].

[6] OOP: Object Oriented Programming. In C++ for example a set of functions may be associated with a group of storage variables. This allows the apparent use of global variables while still maintaining a system of encapsulation.

6 CONCLUSION

Implementing neural networks requires a careful analysis of loop optimization. One of the most common programming errors, is to transport mathematical expressions from theory directly to programming without any analysis of their timing requirements. One of the easiest methods of loop optimization is to remove all constant sub expressions from within the loop structure.

The transfer functions of the network's processing element, and its associated derivative, should be designed in such a way that they are both fully programmable and contain as few in-loop floating point operations as possible. In the sections above, an example of removing constant sub expressions was shown. The function that was derived will give the programmer a well behaved and fully programmable transfer function. This transfer function was also, through timing tests, shown to be the fastest transfer function of the continuous sigmoidal type.

In every different application of BPN, a range of average input sums for a PE may be determined. This is usually done by multiplying the amplitude of the transfer function by the maximum desired weight and by the number of PEs in the previous layer:

$$\mathbf{operatingRange} = \mathbf{N_{layer-1}}\,(\mathbf{A}-\mathbf{B})\cdot\ \mathbf{maxConnectionWeight_{desired}}$$

This operating range represents the maximum and minimum (*transfer function are usually symmetrical*) values processed through the transfer function. The slope parameter of the transfer function describes the slope of the function through the point of inflection. If the response of the first derivative is also examined it will be revealed that α in Equation13 relates to the slope of the derivative's two points of inflection or the half amplitudes as follows:

$$\text{slope} = \begin{cases} \pm\frac{3\pi\sqrt{3}}{16}\alpha^2 & |(X-X_o) = \mp\frac{2}{\alpha\pi}\left(\frac{1}{\sqrt{3}}\right), \quad \text{point of inflection} \\ \pm\frac{\pi}{4}\alpha^2 & |(X-X_o) = \mp\frac{2}{\alpha\pi}, \quad \frac{1}{2}\text{ amplitude} \end{cases} \tag{16}$$

The slope of the derivative is proportional to the convergence speed for gradient decent type training[Wasserman]. The ability to adjust the slope within the operating range of the transfer function can have significant effects on decreasing convergence time of BPN networks.

A APPENDIX

A program in **C** was compiled to determine the number of function calls that could be made every second by various CPUs. Table1 is a complete run of this program. This table was produced on a Pentium running at 60Mhz[7]. The list represents the standard math functions available under ANSI-C[Osborne]. Table2 is a comparison of the two functions used in this article. The bench mark program was run across the different platforms and the function calls per second were recorded.

B ACKNOWLEDGEMENT

This research was supported in part by the Advanced Research Projects Agency under contract N00600–93-K–2051. I greatly appreciate their support, without which this paper could not have been successful.

[7]Pentium is a registered trade name for the Intel 586 CPU

Table 1: Pentium P5-60

FUNCTION	Calls/Second
arccos(x)	2.1039e+05
arcsin(x)	2.5589e+05
arctan(x)	4.6926e+05
ceil(x)	1.7637e+06
cos(x)	6.5488e+05
cosh(x)	3.2457e+05
exp(x)	3.8256e+05
floor(x)	1.7606e+06
log(x)	5.3850e+05
log10(x)	5.3850e+05
sin(x)	6.5574e+05
sinh(x)	3.2103e+04
sqrt(x)	7.1327e+06
tan(x)	4.2052e+05
tanh(x)	2.9762e+04

Table 2: Executions per second vs. CPU/Function Call

SYSTEM	EXP(X)	ATAN(X)
Pentium P5-60 GCC/Linux	3.8256e+05	4.6926e+05
PC 486-DX2/66 BC4/MSDos	1.7961e+05	2.2031e+05
NeXT M68025 GCC/Mach	4.5907e+05	5.9033e+05
DecStation 5000 GCC/ULTRIX	6.5474e+05	4.1936e+05

References

[Wasserman] P.D. Wasserman "Neural Computing, Theory and Practice", Van Nostrand Reinhold, ISBN: 0-44-220743-3

[Cyrix] Cyrix "FasMath 83D87 User's Manual", P.O. Box 850118, Richardson, TX 75085-0118, (214) 234-8387

[Osborne] Osborne "C the complete reference", Osborn McGraw Hill, ISBN:0-07881263-1