Report Number: WUCS-89-25

1989-06-01

Back Propagation with Integer Arithmetic

Takayuki Dan Kimura

The present work investigates the significance of arithmetic precision in neural network simulation. Noting that a biological brain consists of a large number of cells of low precision, we try to answer the question: With a fixed size of memory and CPU cycles available for simulation, does a larger sized net with less precision perform better than smaller sized one with higher precision? We evaluate the merits and demerits of using low precision integer arithmetic in simulating backpropagation networks. Two identical backpropagation simulators, ibp and fbp, were constructed on Mac II, ibp with 16 bits integer representations of network... **Read complete abstract on page 2.**

Back Propagation with Integer Arithmetic

Takayuki Dan Kimura

Complete Abstract:

The present work investigates the significance of arithmetic precision in neural network simulation. Noting that a biological brain consists of a large number of cells of low precision, we try to answer the question: With a fixed size of memory and CPU cycles available for simulation, does a larger sized net with less precision perform better than smaller sized one with higher precision? We evaluate the merits and demerits of using low precision integer arithmetic in simulating backpropagation networks. Two identical backpropagation simulators, ibp and fbp, were constructed on Mac II, ibp with 16 bits integer representations of network parameters such as activation values, back-errors, and weights; and fbp with 96 bits floating point representations of the same parameters. The performance of the two stimulators are compared in solving the same Boolean mapping problem, the sine transfer function with eight binary inputs and one analog output. The speed-up ratio from fbp to ibp is a single training cycle in approximately 7.3 for smaller networks and 4.2 for larger ones. However, for total time necessary to obtain a solution net, the speed-up ratio is 163 for the smaller nets, and 121 for the larger nets, because ibp requires much less number of training cycles than fbp. We also found that networks trained by integer arithmetic have more generalization capabilities than those trained by floating point arithmetic. At present time we have no explanation on this matter.

BACK PROPAGATION WITH INTEGER ARITHMETIC

Takayuki Dan Kimura

WUCS-89-25

June 1989

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Abstract

The present work investigates the significance of arithmetic precision in neural network simulation. Noting that a biological brain consists of a large number of cells of low precision, we try to answer the question: With a fixed size of memory and CPU cycles available for simulation, does a larger sized net with less precision perform better than a smaller sized one with higher precision? We evaluate the merits and demerits of using low precision integer arithmetics in simulating backpropagation networks. Two identical backpropagation simulators, *ibp* and *fbp*, were constructed on Mac II, *ibp* with 16 bits integer representations of network parameters such as activation values, back-errors, and weights; and *fbp* with 96 bits floating point representations of the same parameters. The performance of the two simulators are compared in solving the same Boolean mapping problem, the sine transfer function with eight binary inputs and one analog output .

The speed-up ratio from fbp to ibp in a single training cycle is approximately 7.3 for smaller networks and 4.2 for larger ones. However, for total time necessary to obtain a solution net, the speed-up ratio is 163 for the smaller nets, and 121 for the larger nets, because *ibp* requires much less number of training cycles than *fbp*. We also found that networks trained by integer arithmetic have more generalization capabilities than those trained by floating point arithmetic. At present time we have no explanation on this matter.

1. Introduction

Many AI applications, such as speech recognition, character recognition, and pattern classification problems, reduce to the problem of realizing a large scale Boolean function. Traditional rule-based methodologies for digital system design are limited in implementing such Boolean functions. Neural networks, or Parallel Distributed Processing (PDP)[1], offer an alternative design methodology. A neural net implements a Boolean function by adapting its structure to a sequence of known input-output pairs, i.e., it learns the Boolean function through an extensional specification of the function. Recent discovery of new learning rules, such as the generalized delta rule[2], promises a significant advancement of the new design methodology.

There are four major problems associated with the neural net applications: First, no algorithmic method is known to predict the complexity (ex. the number of hidden units) of a solution net for a particular application problem. Second, training of a neural net is computation intensive. Third, unless a stochastical learning algorithm such as the simulated annealing is used, there is always the danger of a training being stuck at a local minima; and the stochastical learning is very slow. Finally, the currently known neural models have fixed number of input units. The first three problems are common to all neural net applications. The last problem is significant in such applications as speech recognition and language processing.

In this work we concentrate on the second problem. Our software strategy for the second problem is to experiment with integer arithmetic based network simulation. The trade off between the number of processing units and their precision is not clearly understood at present time. With the same memory capacity and available CPU cycles, a larger sized net with less precision may perform better than a smaller sized one with higher precision. A biological brain consists of large number of cells of low precision. Against the obvious advantage of speed up, it is expected that a net consisting of lower precision units is more susceptible to the local minima problem.

The present work investigates the significance of arithmetic precision in neural network simulation. In particular, we compare the merits and demerits of using low precision integer arithmetics in simulating backpropagation networks. This is a part of our ongoing investigation on different strategies to develop computationally efficient neural networks. Our experiments consist of the following:

(1) Designing and implementing two identical simulators on Mac II, one with 16 bits integer representations of network parameters such as activation values, back-errors, and weights; and the other with 96 bits floating point representations of the same parameters.

(2) Evaluation of the memory and CPU requirements (computation time) of the two simulators for solving the same Boolean mapping problem. We have chosen the 8 bits in/out SINE transfer function as the target problem.

---

[1] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Volume 1 & 2, MIT Press, 1986.

[2] Y. Le Cun, "A learning scheme for asymmetric threshold network," in *Cognitiva 85*, CESTA-AFCET (ed.) 1985, 599-604.

2. Experiments

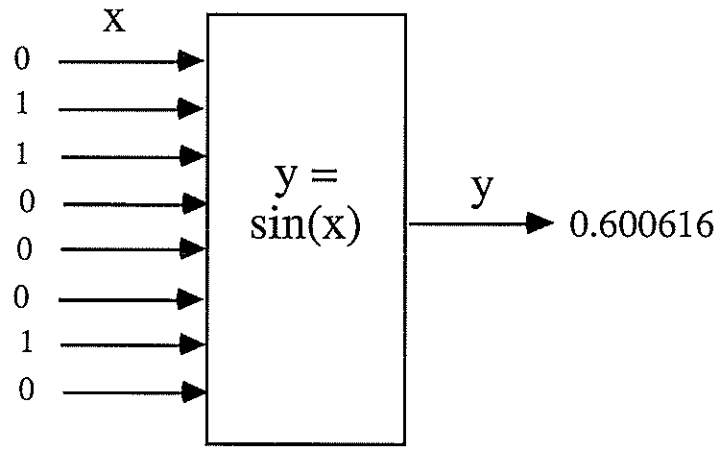
We have constructed two identical simulation programs, *ibp* and *fbp*, in MPW C on Macintosh II, *ibp* with 16 bits integer representations of activation values, back-errors, and weights; and *fbp* with 96 bits floating point representations of the same parameters. Note that Macintosh II has MC68881 floating point processor. Both programs are designed to simulate backpropagation nets with one hidden layer.

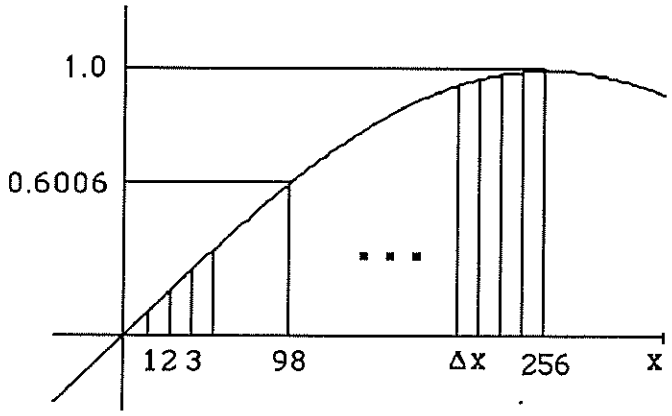
We use the standard sigmoid function,

$$\lambda(x) \equiv \frac{1}{1 + e^{-x}}$$

as the activation function and no momentum term is used. In *ibp*, the evaluation of $\lambda(x)$ is implemented as a single table-look-up operation, where $\lambda(x)$ is a table of 64KB size with 2^{16} entries. The learning rate is fixed to o.5 for both simulators.

The performance of the two simulators are compared by solving the same Boolean mapping problem, the sine transfer function with eight binary inputs and one analog output as shown below.





Training:

$\Delta x = (\pi/2)*(7/256)$

Testing:

$\Delta x = (\pi/2)*(1/256)$

The experiment consists of four simulations. In each simulation, a network is trained by the set of 37 training pairs constructed by sampling every 7th point of the interval $(0,\pi/2)$ divided by 256, and tested by the full 256 input patterns. The performance of a network is

measured by the mean square error (mse) of the difference between the output values and the teaching values. Each simulation is terminated when the mse value is reduced to 0.01 or less. After each presentation of an input pattern, the errors are backpropagated and the connection weights are modified.

3. Results

The results of the four simulations are summarized in the table below.

	type	hidden units	conn.	iteration	time (sec)	speed (ms)	mse (train)	mse (test)
(1)	ibp	8	72	2100	27	12	0.0078	0.0076
(2)	ibp	16	144	1000	46	46	0.0038	0.0039
(3)	fbp	8	72	50000	4410	88	0.0027	0.1282
(4)	fbp	16	144	29000	5597	192	0.0057	0.1097

The first simulation uses the *ibp* simulator to train a network with 8 hidden units (in one layer) and 72 connections. To reach 0.0078 for mse, it required 2100 presentations of a training pair, randomly selected from the set of 37 pairs, and took 27 seconds on Macintosh II, that is to say, each iteration took 12 ms. The resulting network was tested by the entire 256 inputs, and its performance was 0.0076 mse. Similarly, for the other three simulations.

The first and third simulations have the exactly same network architecture, and so are the second and the forth. The speed-up ratio from *fbp* to *ibp* in a single training cycle is approximately 7.3 for smaller networks, i.e., (1) and (3), and 4.2 for larger ones, (2) and (4). However, for total time necessary to obtain a solution net, the speed-up ratio is 163 for (1) and (3), and 121 for (2) and (4), because *ibp* requires much less number of training cycles than *fbp*.

Note that in (1) and (2), there is very little difference between the training mse value and the testing mse value, i.e., 0.0078 vs 0.0076 in (1) and 0.0038 vs 0.0039 in (2). It follows that the networks of (1) and (2) have a good generalization capability. On the other hand in (3) and (4), the differences are significant and their generalization capabilities are poor. Therefore, it is reasonable to conclude that the networks trained by integer arithmetic have more generalization capabilities than those trained by floating point arithmetic. At present time we can offer no explanation on this matter.

Finally, during the training of (1) and (2), we observed many connection weights reaching at the limit (overflow) value. Once that happens, a connection becomes ineffective and the network performance decreases. However, the degradation is short lived as if the rest of the network learns to compensate the loss of the connection in time. We observed none of these phenomena during the simulations of (3) and (4).