Deliverable title: Report on algorithm: compare weight perturbation vs back propagation resource Requirement

Task title: On-Chip Training for SRAM based Artificial Neural Networks
Task ID:3058.001 (Status: Active)
Start Date:1-Sep-2021
Projected End Date:31-Aug-2024
Task University: Indian Institute of Technology/Bombay
Principal Investigator: Udayan Ganguly (IIT Bombay)
Task Leader: Udayan Ganguly (IIT Bombay)
Thrust Area: India Research Program (IRP)

Author(s): Vivek Saraswat, Udayan Ganguly
Affiliation: Indian Institute of Technology/Bombay, India

Deliverable Due Date: Dec 31, 2021

This work was supported in part by Semiconductor Research Corporation (SRC)

Abstract

Backpropagation has long been the standard method to calculate the weight gradients at different layers of a deep neural network. It is very powerful and general scheme but assumes very precise calculation power is available with the system performing it. Hence, as an algorithmic option it is unparalleled. It makes use of forward passes to calculate node outputs and activations and is followed by a backward pass to propagate the error gradients from output layer to internal hidden layers. It is a fast process. However, it is prone to error accumulation. Numerical techniques that perturb the model parameters or output nodes to calculate the effect directly on the final output layer loss are slower alternatives that are very simple in terms of their hardware requirements. We describe these alternatives namely: (a) Weight Perturbation and (b) Node Perturbation using an example of a multiple layer perceptron, the equivalence with a backpropagation training model is established and finally a operations and memory comparison is presented across the three techniques.

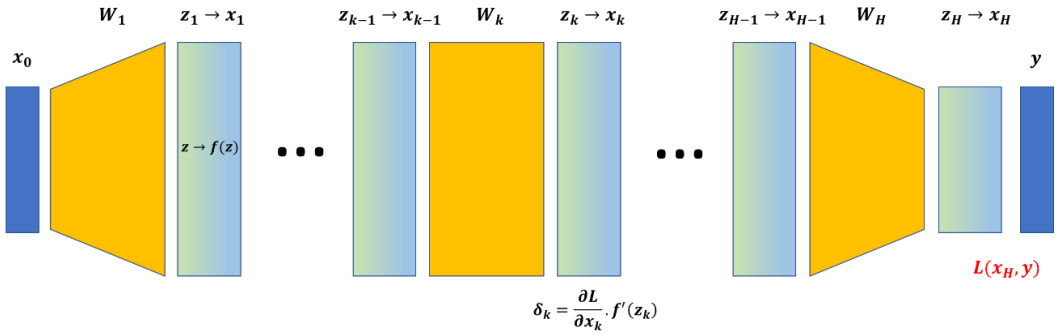
I. Introduction:

Neural networks are powerful tools for classification tasks. However, in order to do that, they make use of abundant training data to train the network model well. Typically, for every input training sample that the network processes, there is an expected output attached to it. This defines the loss function between the model's output and the expected output. The purpose of training is to minimize this loss function as training samples are shown progressively. The objective is to find an update to the model parameters. This is an easy task for a single layer perceptron as the model parameters directly affect the loss and the update calculation becomes straight forward. However, most practical use case networks are deep neural networks with many layers stacked together to form the output and hence there are model parameters internal to the network that indirectly affect the output and hence loss. Backpropagation of output layer gradients to internal layers in a backward pass to enable hidden layer weight updates is a very popular and general scheme for training of deep neural networks. As expected, these are computationally heavy networks with flow of information in forward and backward direction and require precise calculations in

order to avoid accumulating errors in computations. There are numerical perturbation alternatives to calculating change in loss that iterate over the model parameters or outputs that can be much slower but are much simpler circuits without need of precise calculations. These methods are described and compared against in this report.

II. Perturbation Techniques and Backpropagation:

Using an example of a multi-layer perceptron, we describe the network architecture, the input/outputs, the loss calculation and the update calculation in this section (Fig. 1). Consider a multi-layer perceptron with H layers, each layer is fully connected and followed by an activation function to form the next layer's inputs. For each input sample, there is an expected output which is used to calculate a loss function L. The objective of training is to calculate the gradient of the loss function with respect to the network parameters specifically the weights. As described earlier, backpropagation involves a forward pass to calculate the layer outputs, z and activations, x. The update to weights in any layer requires the output error gradient of that layer and the previous layer inputs. In order to generate the output error gradients for each layer, a backward pass of error gradients starting from the output layer needs to back propagated through the network using the same weights as used in the forward pass. This completes the training description using backpropagation.



Training requirement → $\partial L / \partial w_{k,ij}$

Backpropagation:

$$\frac{\partial L}{\partial w_{k,ij}} = \frac{\partial L}{\partial x_{k,j}} . \frac{\partial x_{k,j}}{\partial z_{k,j}} . \frac{\partial z_{k,j}}{\partial w_{k,ij}}$$
$$= \frac{\partial L}{\partial x_{k,j}} . f'(z_{k,j}) . x_{k-1,i}$$
$$= \delta_k . x_{k-1,i}$$

$$\frac{\partial L}{\partial x_{k,j}} = \Sigma_m \frac{\partial L}{\partial x_{k+1,m}} . \frac{\partial x_{k+1,m}}{\partial z_{k+1,m}} . \frac{\partial z_{k+1,m}}{\partial x_{k,j}}$$
$$= \Sigma_m \frac{\partial L}{\partial x_{k+1,m}} . f'(z_{k+1,m}) . w_{k+1,jm}$$
$$= \Sigma_m \delta_{k+1,m} . w_{k+1,jm}$$

(Repeat till you reach $\frac{\partial L}{\partial x_H}$)

In words:
1. Weight gradient depends on output error gradient of any layer and the input to that layer
2. Output error gradient of any layer can be found by backpropagating the output layer loss gradient

Weight Perturbation:

$$\frac{\partial L}{\partial w_{k,ij}} = \frac{\Delta L}{\Delta w_{k,ij}}$$

In words:
Perform a forward pass with perturbed weight to get ΔL

Node Perturbation:

$$\frac{\partial L}{\partial w_{k,ij}} = \frac{\partial L}{\partial z_{k,j}} . \frac{\partial z_{k,j}}{\partial w_{k,ij}}$$
$$= \frac{\Delta L}{\Delta z_{k,j}} . x_{k,i}$$

In words:
Perform a forward pass with perturbed node to get ΔL

Fig. 1. A multi-layer perceptron of fully connected networks showing the input, weights, node outputs, activations, network output, expected output, loss, and propagated error gradient. The training requirement is specified and a description of three gradient calculation techniques are described in formulae and words.

Unlike backpropagation, where the weight update is calculated using a backpropagated error gradient, several numerical perturbation techniques depend on calculating the actual final loss gradient with respect to all parameters – output as well as hidden layers. This requires performing a baseline forward pass. This is followed by a perturbed forward pass where either (a) a weight is perturbed slightly[1,2] or (b) a node output is perturbed[3,4] to calculate numerically an approximation to the output error gradient. Clearly this needs to repeated for all weights/nodes to complete the process of gradient calculation for each weight. For small enough perturbations, this method is equivalent to backpropagation (Fig. 2).

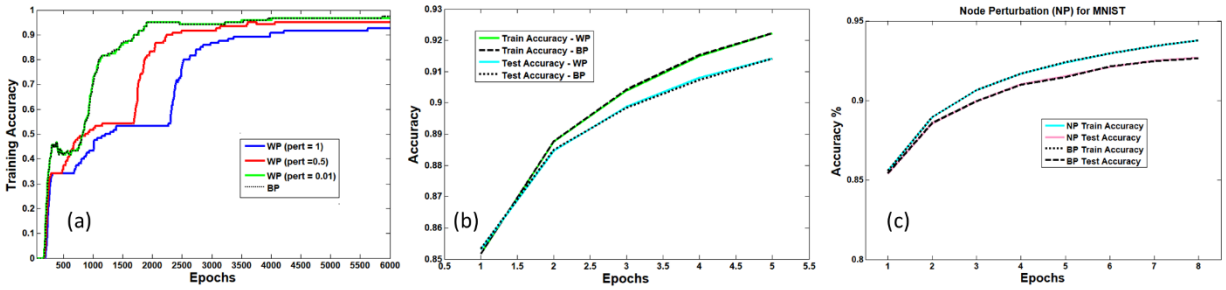


Fig. 2. (a) Training vs Epochs for a Fischer Iris classification network to show equivalence with backprop at lower perturbation levels, MNIST training and testing accuracy vs epochs for (b) weight perturbation and (c) node perturbation to show equivalence with backpropagation

III. Comparison of computations and memory:

A detailed comparison of the time complexity of the operations and the resource complexity of the memory needed to perform each of the techniques is summarized in Fig. 3. On the face of it, the perturbation techniques perform worse on the number of computations needed and are nearly similar in terms of the memory requirement to complete the training. However, one big difference when talking about this comparison in the light of hardware accelerators is the dependence of the algorithm only on forward passes and the dependence of updates on the true loss function gradient at any point in time. These are strong motivators for on-chip learning ASIC ANN chips where precise backpropagation is riddled with precision issues and variability problems. They are not built for precise math and power hungry accurate multibit calculations. Hence scenarios where precision cannot be guaranteed, power cannot be allocated and training time can be relaxed, perturbation techniques are the methods of choice and hence worth exploring under these hardware constraints.

[1] Simultaneous Inference and Training Using On-FPGA Weight Perturbation Techniques | IEEE Conference Publication | IEEE Xplore

[2] Weight perturbation: an optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks | IEEE Journals & Magazine | IEEE Xplore

[3] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=58323

[4] http://papers.nips.cc/paper/608-summed-weight-neuron-perturbation-an-on-improvement-over-weight-perturbation.pdf

Let H be the number of hidden layers

No. of nodes per layer → n_i , where i = 0,1,2 . . . , H

Total no. of weights in network, $N = \sum_{i=0}^{H-1} n_i n_{i+1}$ (For fully connected)

Total no of nodes in the network (except input layer), $n = \sum_{i=1}^{H} n_i$

	Back Propagation	**Node Perturbation**	**Weight Perturbation**
Description	- Output obtained (after FF) is passed backward through the network - Calculating intermediate error terms δ_i by multiplying with weight transpose matrix W_i^T - Product of these error terms with intermediate activations (obtained during FF) to get gradients for weight update.	- Approximate gradients by perturbing nodes in the hidden and output layers - Add small perturbation after MAC and before applying activation function - Forward difference formula is applied only to the gradient wrt weighted sum of inputs (Δs)	- Gradient approximated by perturbing a particular weight in the network and calculating the difference in output error before and after perturbation. - These gradients for weight update are approximated using the forward difference formula.
Data Flow	Involves ***forward and backward pass of data*** to determine weight updates. **1 forward pass and 1 backward pass per example**	Involves ***only forward pass of data*** to approximate weight updates. → simpler hardware **$\sim n$ forward passes per example**	Involves ***only forward pass of data*** to approximate weight updates. → simpler hardware **~N forward passes per example**
Computation Complexity	- ***Multiplication with transpose weight matrix W_i^T*** (during backward pass of error) - Large ***matrix operations involved in backward pass*** of errors and in calculating gradients.	- ***No need for weight transpose*** matrix. (FF is enough to estimate gradients) - Need to multiply approximated gradient (single scalar value) with intermediate activations (required for weight update calculation) - Weight update for all weight connected to pert node is obtained in one FF	- ***No need for weight transpose*** matrix. (FF is enough to estimate gradients) - ***Difference between error before and after perturbation*** and then scale it (depending on learning rate and perturbation size) - Weight update for only pert weight is obtained in one FF
Memory requirement or Stored Data	- ***Input, Output and Intermediate activations*** corresponding to one forward pass - ***Error vectors δ_i*** for every layer in the network – to calculate weight updates and previous layer errors - ***Weight update matrix*** ΔW_i - **Total memory =** $\sim 2n + N$	- ***Input and Intermediate activations*** corresponding to one forward pass - ***Weight update matrix*** ΔW_i - **Total memory =** $\sim n + N$	- ***Error for no perturbation*** (at the beginning of every batch) - ***Perturbation error*** calculated for all weights in the network - ***Output activation vector*** - ***Weight update matrix*** ΔW_i - *No intermediate activations and error vectors δ_i.* - **Total memory =** $\sim N$
No. of unit multiplication operations per example	• Forward pass ---> N • Backward pass of error vector δ_i (multiplication with W_i^T and derivative of activation function) ---> $N + n$ • Calculation of ΔW_i ---> N ***Linear*** relation with total no. of weights [$\sim 3N$] $O(n^2)$, n is number of nodes	• Forward pass (no perturbation)---> N • Forward pass (with perturbation) ---> *N/ pert node* *i.e.* $N * n$*(overall)* • Scaling of intermediate activations using $\frac{\Delta E}{\Delta s}$ ---> N Relation with total no of weights [~$N(n+2)$] $O(n^3)$, n is number of nodes	• Forward pass (no perturbation)---> N • Forward pass (with perturbation) ---> *N/ weight* *i.e.* N^2*(overall)* • Scaling of $\Delta E(pert)$ ---> N ***Quadratic*** relation with total no. of weights [$N(N+2)$] $O(n^4)$, n is number of nodes

Fig. 3. Comparison table for backpropagation and numerical approximation and perturbation techniques of gradient calculation