Aayush Ankit

BME 595 (Deep Learning) Final Project Report

Exploring Synapse Pruning for Energy-Efficient Deep Neural Networks

Introduction: Deep Learning Networks (DLNs) are a class of machine learning algorithms which are inspired from hierarchical organization of synapses in the human brain. DLNs have lately shown state-of-the-art results across various cognitive applications including image recognition, speech recognition and natural language processing. However, DLNs are both computationally as well memory intensive which impedes their deployment in embedded systems. Most of the standard DLNs for instance AlexNet have too many parameters that makes it impossible to the DLNs on the on-chip memory. This necessitates storing the network parameters on DRAM and only keeping the data currently being used by the computational core on on chip-memory (SRAM). For instance, AlexNet and VGG-16, which have achieved high accuracies on ImageNet have around 61 million and 130 million parameters respectively. A typical energy profile on 45nm CMOS technology shows that DRAM accesses consume 100 orders higher energy than other components (computation units, SRAM etc.) []. Hence, on one hand increased DRAM accesses easily exceed the power budget for most embedded applications but on the other hand, the size of any typical DLN necessitates their usage.

In this work, we explore synapse pruning which aims to dynamically optimize a DLN architecture during training by removing insignificant synapses []. This helps in realizing an application specific DLN architecture with greatly reduced parameters without loss in classification accuracy. Further, parameter reduction provides opportunities to utilize several compression techniques to efficiently store the network parameters on DRAM. Subsequently, this can lead to great reduction in the number of DRAM accesses for the DLN computation thereby, allowing energy-efficient acceleration of DLNs. For implementing synapse pruning, a new module has been designed in Torch called “nn.Prune”. This can be augmented to the modules containing trainable parameters (“nn.Linear” and “nn.SpatialConvolution”) in any typical neural network realized in Torch. The implementations have been tested on MNIST and CIFAR-10 datasets using two Convolutional Neural Networks (CNNs) namely (1) Lenet and (2) modified Lenet (Lenet augmented with additional convolutional layer).

Previous work: This work is based on the approach proposed by Han et. al. [] to learn the connections in a deep network by using iterative pruning. Figure 1 shows the iterative pruning technique and its effect on a typical neural network structure. Additionally, other works on network pruning have used it to reduce network complexity and over-fitting. An early approach to pruning was based on biased weight decay []. Recently, HashedNets [] have been proposed to reduce model sizes by using a hash function to randomly group connection weights into hash buckets.

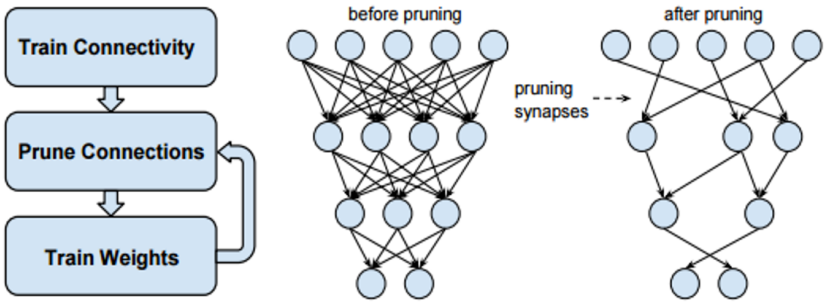

Figure 1: Iterative network pruning to learn the optimal connections

Contribution: We implemented the iterative pruning approach in Torch by designing new modules called “nn.Prune” to update the forward and backward propagation steps in a typical DLN based on “masks”. The mask is used to keep track of the parameters which have been pruned by removing the parameters having magnitude lesser than a threshold.

I implemented the “nn.Purne” module for “nn.Linear” layer (from Figure 2) that stores the trainable parameters for fully connected layers in the CNN. I also contributed equally in designing the “nn.Prune” layer for “nn.SpatialConvolution” (from Figure 2) that houses the trainable parameters for convolutional layers in a CNN. Additionally, I also designed the “Prune Control” (shown in Figure 2) script which controls the pruning threshold and pruning rates for various “nn.Prune” modules in the CNN. Further, the prune control script also controls the starting and stopping conditions of the pruning process while training.

For the analysis part, I ran the simulations on MNIST and CIFAR-10 datasets using the Lenet network structure to analyze the pruning capacity of the networks for a given accuracy requirement. Additionally, I also analyzed the training effort needed to train the networks with the iterative pruning approach. The next section shows the results from the analysis obtained by both the team members (me and Chankyu Lee).

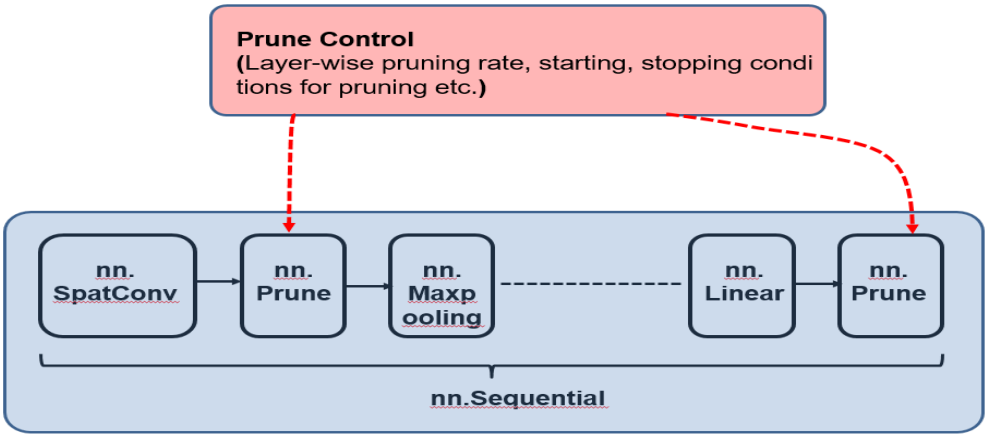

Figure 2: Implementation setup which shows a DLN with “nn.Prune” modules and the “Prune Control” script.

Results: In this section, we discuss our combined results on parameter reduction and training effort obtained on our benchmarks (shown in Figure 3). Figure 3 shows the detailed configuration of all the CNN topologies used and the corresponding datasets. Please note that due to time constraints, we couldn’t finish the evaluation of the 4th benchmark in Figure 3.

Dataset	Neural Network (NN) structure
MNIST	28x28 – 6c5 – 16c5 – 120 – 84 – 10
MNIST	28x28 – 6c3 – 16c3 – 36c4 – 240-120 – 84 – 10
CIFAR-10	3x32x32 – 6c5 – 16c5 – 120 – 84 – 10
CIFAR-10	3x32x32 – 6c3 – 16c3 – 36c4 – 240-120 – 84 – 10

Figure 3: Experimental Benchmarks used for evaluation of the proposed approach

Figure 4 shows the fraction of significant synapses obtained for different networks at different accuracy requirements. Parameter reduction is defined as the percentage of synapses which could be successfully pruned (1-significant synapses). For instance, for MNIST-2c-99 that is the MNIST trained on the CNN with 2 convolutional layers (see Figure 3) has around 40% significant synpses when rained for 99% classification accuracy.

As shown in Figure 4., the DLNs can be pruned effectively for the given accuracy requirement using our proposed approach. We obtained parameter reductions of 38% to 79% (61 % on average) across all our benchmarks. Further, our analysis underscores that networks with lower accuracy demands can be pruned much more than the networks with higher accuracy requirements. Moreover, we also observe that the percentage of synapses which can be pruned are higher for a bigger network (for instance MNIST-3c) in comparison to a smaller network (MNIST-2c) at iso-accuracy. As expected, the percentage of significant synapses are much higher for CIFAR-10 when trained on the CNN with 2 convolutional layers (from Figure 3). This is because of the fact that the two layered CNN obtains much lower accuracy on CIFAR-10 than MNIST which underscores the fact that the number of redundant parameters are much less on the network trained for CIFAR-10.

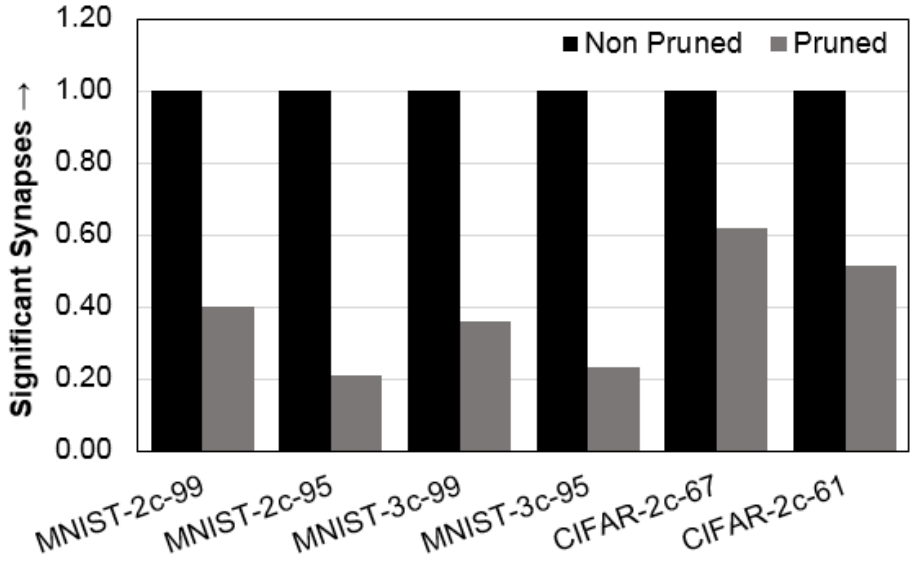

Figure 4: Fraction of significant synapses

Next, we discuss the effect on pruning on the training effort. We define training effort as the training time (iterations) needed by the network to train with iterative pruning. As shown, in Figure 5, the training effort for the pruned network is higher that the network with no pruning. This is due to the fact that pruning is an approximation approach which tries to reduce the parameter and thereby capacity of the network. Hence, pruning makes it harder for back-propagation algorithm to train the network for a required accuracy. Although, our pruning approach leads to significant increase in training time and training energy, the testing phase (classification) will be much more energy-efficient. Typically, a DLN is trained a few times but subsequently used millions of times after deployment. So, our approach targets the more critical testing (classification) phase.

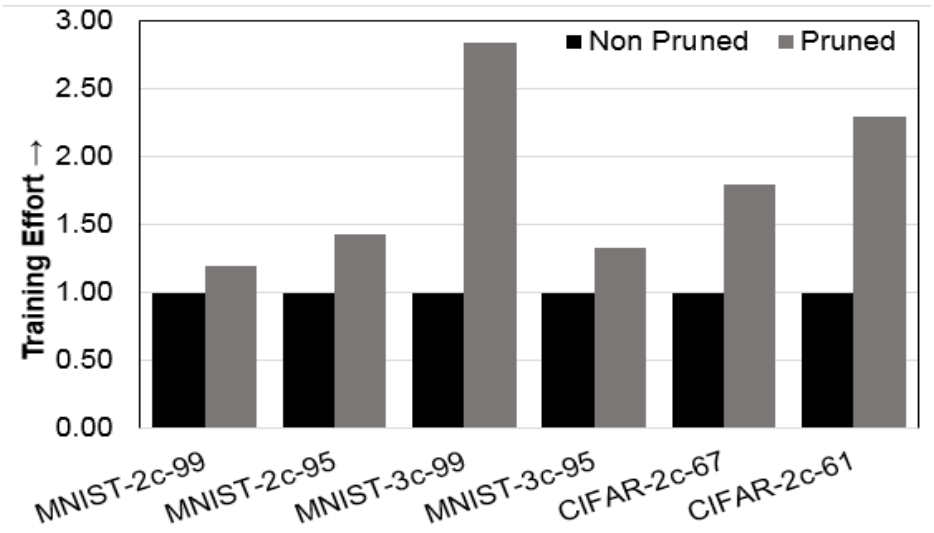

Figure 5: Training effort with and without pruning

Conclusions: We analyzed the effect of network pruning in DLN training to analyze it effect on parameter reduction and training effort. Its worth noting that pruning is a simple technique but shows much promise towards energy-efficient realization of Deep Learning Networks. In future, we plan to continue this work and develop sophisticated prune control techniques which can achieve higher pruning abilities at lower training efforts.