DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING AND QUANTIZATION

Angelo Catalani 1582230

Abstract— Neural Network are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources.
Compressing a neural network with pruning and quantization with low loss of accuracy is the aim of this project

1 Introduction

I have taken into consideration the following papers:

1. [1] : deals with specific pruning issues: regularization terms, threshold choice, and parameter co-adaption
2. [2] : is the sequel of the previous paper : pruning and quantization

2 Pruning

The pruning technique I have implemented as described in the papers consists of three steps :

1. train connectivity : the neural network is trained for a given n ber of epochs
2. prune connection : remove all the weights below a threshold
3. train the weights : re-train the reduced neural network to l the final weights and repeat from step 2

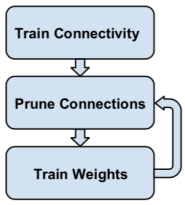

Fig. 1. Pruning process

It is significant to note that:

1. the first step is conceptually different from the way a neural network is normally trained because in this step we are interested in finding the important connection rather than the final weights
2. retraining the pruned neural network is necessary for the accuracy since after the remotion of some connection (step 2),in general the accuracy will drop
3. pruning works under the hypothesis that the network is overparametrized so that it solves not only memory/complexity issues but also can reduce the risk of overfitting

The regularization terms used in the loss function, tends to lower the magnitude of the weight matrices, so that more weights will be close to zero and good candidates for being pruned.
In particular, I have chosen L2 regularization because gives better results than L1([1]).
Deep neural network can be affected by the vanishing gradient problem.Even if [1] does not deals directly with that problem it notes that when a neural network struggles to update its weights consistently, it will not be able to recover from pruning of some neurons during the iterative pruning train.
To deals with this issue, as written in [1] I have pruned the convolutional layers and the dense layers on different iterations, so that the error caused by the pruning at each iteration is attenuated.
In [1], the threshold value is obtained as a quality parameter multiplied by the standard deviation of a layer's weights.
This choice is justified by the fact that as it is the case of my experiments, the weights of a dense/convolutional layers are distributed as a gaussian of zero mean so that the weights in the range of the positive and negative standard deviation are 68% of the total.

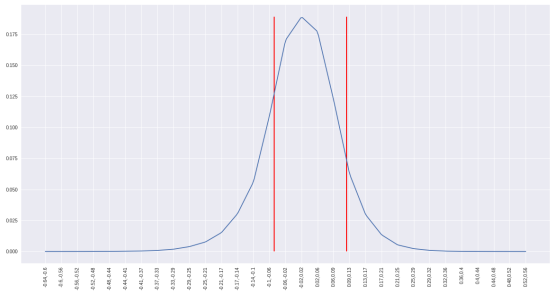

Fig. 2. Weight distribution (with +/- std in red) before pruning for a dense layer of Lenet300-100

3 Quantization And Weight Sharing

This section is described in [2].
After the network has been pruned, the network is further compressed by reducing the number of bits to represent the single weights.
In particular I have applied k-means (with 1 dimensional data) to the weights of each layer so that the new weights of the layer are the centroids to which the original weights belong to.
Crucial to this step is the choice of k and the centroid initialization.
The choice of k is a consequence of the number of bits used in this step : if we want to compress the layers weights to n bits, we can use up to 2^n centroids.
The tradeoff between accuracy-compression is due to the number of bits used : the more the bits, the more the accuracy, the more the space required.
The paper describes 3 different technique regarding the centroid initialization:

1. forgy : random choice among the weights
2. density-based : consider the cumulative distribution of weights (cdf) and takes the x-values at different fixed y-values (cdf)
3. linear : consider equal sized intervals as centroids between the minimum and maximum weight

In order to fully differentiate the initialization methods, it is important to note the weights of a single layer are distributed as a bimodal distribution after the pruning : this means that :

1. forgy and density-based will place the weights around the two peaks because it is where weights are concentrated and the cdf varies the most. The drawback is that very few centroids will have large absolute value which results in poor representation of the few large weights.

2. linear equally space the range of weights, so it does not suffer from the previous problem

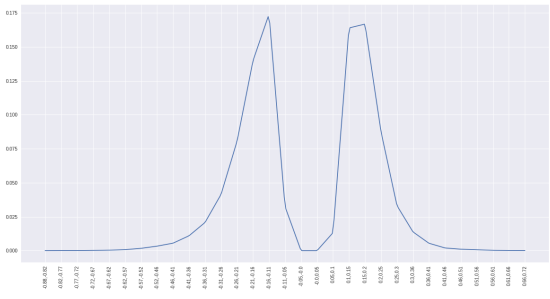

Fig. 3. Weight distribution after pruning for a dense layer of Lenet300-100

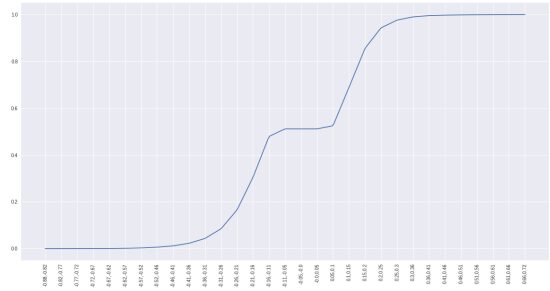

Fig. 4. Cumulative distribution for a dense layer of Lenet300-100

At the end of this process, each layer will have at most 2^n different values.
In addition to this, the paper perform the fine tuning of the centroids C_k in the following way :

1. compute normally the gradient of the loss with respect to the weights : $\frac{\partial L}{\partial W_{ij}}$

2. sum the gradients whose weights belongs to a given centroid : $\frac{\partial L}{\partial C_k} = \sum_{i,j} \frac{\partial L}{\partial W_{ij}} \cdot \mathbb{1}(I_{ij} = k)]$

I have not implemented the fine tuning of the centroids because:

1. the loss of accuracy after quantization is almost inexistent with respect to the pruned model

2. the latency is eccessive : for each single batch in every epoch, I should have scanned all the gradients

4 Experiment

I have run my experiment on the MNIST classification problem using the Colaboratory , with the neural networks named : LeNet300-100 and LeNet5.
Each element in the dataset is an image of size : 28 x 28 and the output class is a vector of 10 values obtained using the one hot encoding.
LeNet300-100 has in total:

1. a fully connected layer of size 300 : $28 \cdot 28 \cdot 300 = 235200$ weights plus 300 bias weights

2. a fully connected layer of size 100 : $300 \cdot 100 = 30000$ weights plus 100 bias weights

3. a fully connected layer of size 10 : $10 \cdot 100 = 1000$ weights plus 4 bias weights

The second one :

1. a convolutional layer with 20 filters of shape 5 by 5 : $20 \cdot 5 \cdot 5 \cdot 1 = 500$ weights plus 20 bias weights

2. a convolutional layer with 50 filters of shape 5 by 5 : $50 \cdot 5 \cdot 5 \cdot 20 = 25000$ weights plus 50 bias weights

3. a fully connected layer of size 256 : $256 \cdot 2450 = 627200$ weights plus 256 bias weights (2450 is the flattened output of the previous layer)

4. a fully connected layer of size 10 : $10 \cdot 256 = 2560$ weights plus 10 bias weights

The loss function is the categorical cross entropy with L2 regularization and the optimizer algorithm is Adam with 0.001 as learning rate.

4.1 Test 1

It is executed with LeNet300-100 with 200 epochs for normal training and 30 epochs for pruning.
In the following table the threshold is the quality parameter that in the end is multiplied with the standard deviation for the real threshold used in the pruning.
It is significant to note that a lower threshold value is used for the biases and last dense layer.
This choice is justified by the fact that they have a small number of weights so that they are less affected by the over-parametrization.

Table 1. Test 1 pruning results

LAYER	THRESHOLD	ZERO WEIGHTS
fc1	1	184192 on 235200
fc1 bias	0.1	84 on 300
fc2	1	22164 on 30000
fc2 bias	0.1	26 on 100
fc3	0.5	351 on 1000
fc3 bias	0	0 on 10

The accuracy before pruning is : 98.24% and after is : 98.21% (different execution can lead to higher/lower accuracy in the range of +/- 0.1%)..
This is the final accuracy after the different quantization techniques with 2,5,8 bits are applied after the pruning:

Table 2. Test 1 quantization accuracy

QUANTIZATION	2-BITS	5-BITS	8-BITS
Linear	97.52%	98.20%	98.20%
Forgy	97.63%	98.21%	98.21%
Density	97.93%	98.18%	98.20%
Kmeans++	97.70%	98.18%	98.21%

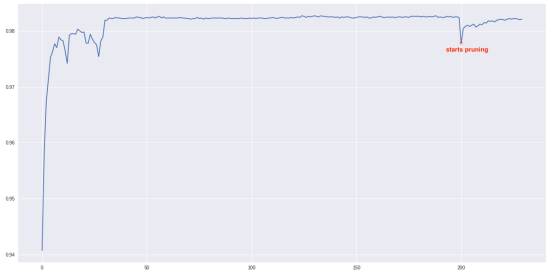

Fig. 5. Accuracy during the training for test 1 : when pruning starts at epoch 200 the accuracy slightly drops

4.2 Test 2

It is executed with LeNet300-100 with 200 epochs for normal training and 500 epochs for pruning, with higher hard thresholds : now the threshold is a constant value that is not multiplied by the standard deviation of the weights.
In addition to this the loss function does not have the regularization terms.

Table 3. Test 2 pruning results

LAYER	HARD THRESHOLD	ZERO WEIGHTS
fc1	0.25	234521 on 235200
fc1 bias	0	0 on 300
fc2	0.25	29030 on 30000
fc2 bias	0.0	0 on 100
fc3	0	0 on 1000
fc3 bias	0	0 on 10

The accuracy before pruning is : 98.22% and after is : 96.11%. This is the final accuracy after the different quantization technic with 2,5,8 bits are applied after the pruning:

Table 4. Test 2 quantization accuracy

QUANTIZATION	2-BITS	5-BITS	8-BITS
Linear	31.39%	95.74%	96.11%
Forgy	44.14%	93.31%	95.99%
Dense	60.80%	95.63%	96.08%
Kmeans++	31.10%	95.83%	96.13%

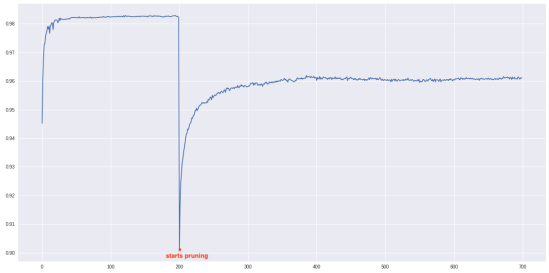

Fig. 6. Accuracy during the training for test 2

4.3 Test 3

It is executed with LeNet5 with 100 epochs for normal training and 30 epochs for pruning.

It is important to stress the fact that as said before, to avoid the vanishing gradient problem, first the convolutional layers are pruned and then the dense layers (the same for the following test 4).

Table 5. Test 3 pruning results

LAYER	THRESHOLD	ZERO WEIGHTS
conv1	1	0 on 500
conv1 bias	0.1	0 on 20
conv2	1	7826 on 25000
conv2 bias	0.1	0 on 50
fc1	0.5	624960 on 627200
fc1 bias	0	256 on 256
fc2 bias	0	420 on 2560
fc2 bias	0	0 on 10

The accuracy before pruning is : 99.24% and after is : 99.29%. This is the final accuracy after the different quantization techniques with 2,5,8 bits are applied after the pruning:

Table 6. Test 3 quantization accuracy

QUANTIZATION	2-BITS	5-BITS	8-BITS
Linear	99.17%	98.29%	99.29%
Forgy	95.66%	99.30%	99.29%
Density	99.16%	99.27%	99.28%
Kmeans++	99.14%	99.28%	98.28%

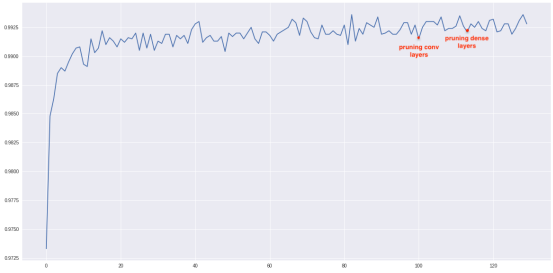

Fig. 7. Accuracy during the training for test 3

4.4 Test 4

It is executed with LeNet35 with 70 epochs for normal training and 100 epochs for pruning, with higher hard thresholds.
In addition to this the loss function does not have the regularization terms.

Table 7. Test 4 pruning results

LAYER	HARD THRESHOLD	ZERO WEIGHTS
conv1	0	0 on 500
conv1 bias	0	0 on 20
conv2	0.25	24542 on 25000
conv2 bias	0	0 on 50
fc1	0.5	620797 on 627200
fc1 bias	0	0 on 256
fc2 bias	0.1	982 on 2560
fc2 bias	0	0 on 10

The accuracy before pruning is : 99.16% and after is : 99.12%. This is the final accuracy after the different quantization techniques with 2,5,8 bits are applied after the pruning:

Table 8. Test 4 quantization accuracy

QUANTIZATION	2-BITS	5-BITS	8-BITS
Linear	94.10%	99.09%	99.04%
Forgy	96.29%	98.79%	99.01%
Dense	97.57%	99.02%	99.03%
Kmeans++	95.01%	99.02%	90.02%

Fig. 8. Accuracy during the training for test 4

REFERENCES

[1] Song Han, Jeff Pool, John Tran, William J. Dally *Learning both Weights and Connections for Efficient Neural Networks*

[2] Song Han, Huizi Mao, John Tran, J. Dally *DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING*