10605: Machine Learning with Large Dataset || Carnegie Mellon University

MiniProject-A Programming: Model Pruning Exercise

Saurabh Patil (ssp2@andrew.cmu.edu),

Kaiwei (Kevin) Zhao (kaiweiz@andrew.cmu.edu),

Jaldhir Trivedi (jaldhirt@andrew.cmu.edu)

Method 1: Global Thresholding Magnitude Based Pruning

We collect weights & biases of all layers and sort them according to their absolute values (np.abs(w)). We choose sparsity s% and find the absolute value of s percentile weights in the above weights array. This is our threshold weight value. We make all weights whose absolute value is less than the threshold zero. Thus, the resultant model would be s% sparse. We evaluate the accuracy of the model using the validation dataset to get our sparse model accuracy. We range threshold percentage (s%) from 0 to 95% to get the corresponding accuracy list. We use these two lists to create the Pareto frontier.

Method 2: Layer specific Thresholding Magnitude Based Pruning

We collect weights & biases of individual layers and sort them according to their absolute values (np.abs(w)). We choose sparsity s% and find the absolute value of s percentile weights in the individual weights array. These are our threshold weight values for corresponding . We make all weights whose absolute value is less than the threshold zero. Thus, the resultant model would be s% sparse. We evaluate the accuracy of the model using the validation dataset to get our sparse model accuracy. We range threshold percentage (s%) from 0 to 95% to get the corresponding accuracy list. We use these two lists to create the Pareto frontier.

Method 3: Freeze Layer Retraining Pruning Method

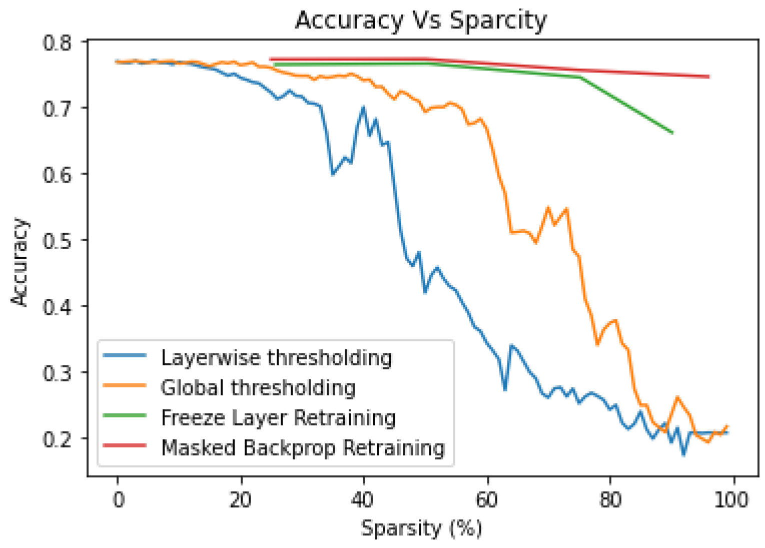
We collect weights & biases of all layers and sort them according to their absolute values (np.abs(w)). We choose sparsity s% and find the absolute value of s percentile weights in the above weights array. This is our threshold weight value. We make all weights whose absolute value is less than the threshold zero. Thus, the resultant model would be s% sparse. Then, we freeze the layers corresponding to the weights that were pruned. After freezing the layers, we retrain the model to update the weights in other layers. We evaluate the accuracy of the model using the validation dataset to get our sparse model accuracy. We range threshold percentage (s%) from 0 to 95% to get the corresponding accuracy list.

Method 4: Retrain with Masked Backprop to Freeze sparse Weights

For this method we use the best-result weights made available using method 3. First, we create a mask for backprop (mask value is 1 for nonzero weights at the beginning and 0 for zero-valued weights). Then, we overwrite the function `Model::train_step` so that every time during backpropagation, we multiply the updated gradient with the mask and keep the zero-valued weights to zero.

We retrain the model so that we get higher accuracy while remaining the same sparsity. However, as in each round, the gradient is first calculated with respect to all weights (whether zero value or not). As a result, those non-zero-valued weights only constituted a small part of the entire gradient and our loss decreased very slowly. We need around 300 epochs of training for the accuracy to rise to around 74% with a sparsity of over 96%. This is the best result we could achieve.

Pareto Frontier for 4 methods



Discussion

Method 1 & 2 are the easiest to implement and test for desired sparsity levels. However, there is no retraining of the model and thus after a threshold the accuracy drops rapidly. Method 1 gives better results than method 2 as layerwise thresholding compulsorily compresses weights no matter how high the threshold is for e.g. the bias weight matrix for the last layer is of shape (5,) now if we want to apply sparsity of 60% 3 out of 5 weights will become sparse and so the resultant model is less capable of approximating complicated functions. Method 3 is particularly potent in the given neural architecture as most of the weights are concentrated in few layers which makes it easy to sparse these layers and freeze them to retain the sparsity during retraining. Method 4 leads to the best result, because it retrains the entire network while retaining the initial sparsity.

As a reflection, we found that method 1 matches our expectation as we read the paper on magnitude-based pruning: we can indeed preserve relative high accuracy (around 70%) while zeroing out almost half of the weights. We saw that the potential for simple magnitude based pruning results in excellent results. These methods were able to sparse 96% of model weights while still retaining validation & test accuracy of 74-75%. The starting solution can be very simple: just zeroing out the weights with the smaller absolute values, and that weight's absolute value is roughly proportional to its significance in the entire model (though not strictly proportional).

The second lesson we learnt was that weight-specific freezing was very hard if we only rely on Tensorflow's existing tools (excluding the pruning tools). We had to delve into the backpropagation mechanism that TF applies so that we can have a weight freezing that works. We also need to utilize Tensorflow's checkpoint because our losses are not monotonically decreasing per epoch. The best validation accuracy lies in the last bunches of epoches, but usually not exactly the last epoch.

Readme

The following files are submitted with this report:

Train_full_model.ipynb : This is the jupyter notebook which trains and saves the uncompressed model to 300 iterations. We use this to save the uncompressed model weights which gives around 76.71% validation accuracy. Saved weights can be loaded into python scripts for pruning.

Uncompressed_model_weights.h5: Weights for uncompressed model with 0 sparsity and 76.71% accuracy.

Global_thres_weights.h5: Weights for pruned model through global thresholding which yields best challenge score.

Layerwise_thre_weights.h5: Weights for pruned model through global thresholding which yields best challenge score.

Method1&2.ipynb: Jupyter notebook for pruning weights using methods Global Thresholds & Layerwise Thresholds. Also used for creating Pareto charts for all 3 methods.

Freeze_layer_retraining_method.ipynb: Jupyter notebook for pruning weights by freezing layers and retraining the weights.

Freeze_layer_training_weights.h5: Weights for pruned model through freezing and retraining weights which yields best challenge score.

my_model_weights.h5: Best challenge score weights as submitted on Canvas

Masking_method.ipynb: Implementation of Method 4.

Code References

Overwrite of function `Model::train_step`:
https://keras.io/guides/customizing_what_happens_in_fit/