CS222 Homework 1

Stable Matching and Algorithm Analysis

Exercises for Algorithm Design and Analysis by Li Jiang, 2018 Autumn Semester

- 潘佳萌

- 516030910510

1 Introduction

The number of parameters in a deep neural network is usually very large, which helps with its learning capacity but also hinders its scalability and practicality due to memory/time inefficiency and overfitting. The most simplest yet popular weight removal method is to prune out weak weights by simple thresholding. Another way to induce sparsity on weights is by ‘1-regularization, which can obtain a compact, memory-efficient network at the expense of small reduction in the prediction accuracy. A common methodology for inducing sparsity in weights and activations is called pruning. Pruning is the application of a binary criteria to decide which weights to prune: weights which match the pruning criteria are assigned a value of zero. Pruned elements are ”trimmed” from the model: we zero their values and also make sure they don’t take part in the back-propagation process. As the paper(Combined Group and Exclusive Sparsity for Deep Neural Networks) said, Iterative pruning is another effective method for obtaining a sparse network while maintaining high accuracy.

2 Prun

A common methodology for inducing sparsity in weights and activations is called pruning. Pruning is the application of a binary criteria to decide which weights to prune: weights which match the pruning criteria are assigned a value of zero. Pruned elements are ”trimmed” from the model: we zero their values and also make sure they don’t take part in the back-propagation process.

We can prune weights, biases, and activations. Biases are few and their contribution to a layer’s output is relatively large, so there is little incentive to prune them. We usually see sparse activations following a ReLU layer, because ReLU quenches negative activations to exact zero ($ReLU(x) : max(0, x)$). Sparsity in weights is less common, as weights tend to be very small, but are often not exact zeros.

2.1 Weight Prun

Weights pruning, or model pruning, is a set of methods to increase the sparsity (amount of zero-valued elements in a tensor) of a network’s weights. In general, the term ’parameters’ refers to both weights and bias tensors of a model. Biases are rarely, if ever, pruned because there are very few bias elements compared to weights elements, and it is just not worth the trouble.

Pruning requires a criteria for choosing which elements to prune - this is called the pruning criteria. The most common pruning criteria is the absolute value of each element: the element’s absolute value is compared to some threshold value, and if it is below the threshold the element is set to zero (i.e. pruned) . This is implemented by the distiller.MagnitudeParameterPruner class. The idea behind this method, is that weights with small l1-norms (absolute value) contribute little to the final result (low saliency), so they are less important and can be removed.

A related idea motivating pruning, is that models are over-parametrized and contain redundant logic and features. Therefore, some of these redundancies can be removed by setting their weights to zero.

And yet another way to think of pruning is to phrase it as a search for a set of weights with as many zeros as possible, which still produces acceptable inference accuracies compared to the dense-model (non-pruned model). Another way to look at it, is to imagine that because of the very high-dimensionality of the parameter space, the immediate space around the dense-model’s solution likely contains some sparse solutions, and we want to use find these sparse solutions.

2.2 algorithm

Magnitude is the most basic pruner: it applies a thresholding function, thresh(.), on each element, wi, of a weights tensor. A different threshold can be used for each layer's weights tensor. Because the threshold is applied on individual elements, this pruner belongs to the element-wise pruning algorithm family.

$$thresh(w_i) = \begin{cases} w_i : \ if|w_i| > \lambda \\ 0 : if|w_i| < \lambda \end{cases} \tag{1}$$

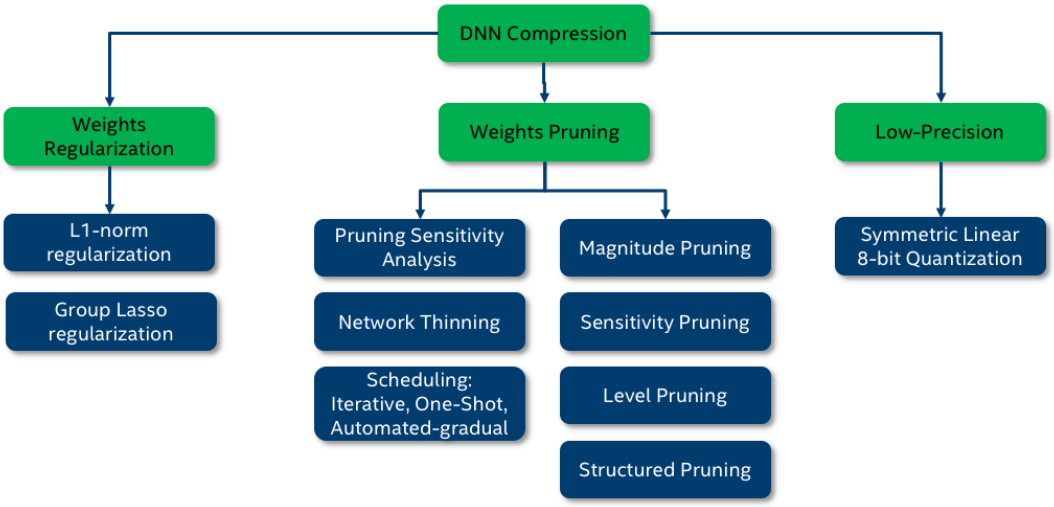

图 1: algorithm

Algorithm 1 Pruning Algorithm

$current\_itr = 0$
while training **do**
  for all parameters **do**
    $param = (param$ **and** $mask)$
    if $current\_itr > start\_itr$ **and** $current\_itr < end\_itr$ **then**
      if $(current\_itr \bmod freq) == 0$ **then**
        if $current\_itr < ramp\_itr$ **then**
          $\epsilon = \theta * (current\_itr - start\_itr + 1)/freq$
        else
          $\epsilon = (\theta * (ramp\_itr - start\_itr + 1) + \phi * (current\_itr - ramp\_itr + 1))/freq$
        end if
        $mask = abs(param) < \epsilon$
      end if
    end if
  end for
  $current\_itr \mathrel{+}= 1$
end while

图 2: algorithm