SIMPLIFICATION OF PRUNED MODELS

Andrea Bragagnolo, Carlo Alberto Barbano

andrea.bragagnolo@unito.it, carlo.barbano@unito.it

1 Simplification procedure overview

The simplification procedure consists of three steps:

1. Batch Normalization fusion
2. Bias propagation
3. Channels removal

2 Batch Normalization fusion

A vast amount of modern neural networks use Batch Normalization as a way to improve generalization. Given an input x, we can define the output of Batch Normalization as:

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{1}$$

where γ and β represent, respectively, the weights and bias of the layer and are learned using standard backpropagation procedures; μ and σ^2 represent the mean and variance computed over a batch. During training this layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. Let us denote this approximations as $\widehat{\mu}$ and $\widehat{\sigma}^2$. Furthermore each parameter is defined for each channel of the input feature map. We will denote this as γ_c, β_c, $\widehat{\mu}_c$ and $\widehat{\sigma}_c^2$ for channel c.

Once the network is trained all the parameters of its layers are frozen. We can therefore implement a frozen Batch Normalization as a 1×1 convolution. Given a feature map F of shape $C \times H \times W$, we can compute its normalized version $\widehat{F}$ as:

$$\begin{bmatrix} \widehat{F}_{1,i,j} \\ \widehat{F}_{2,i,j} \\ \vdots \\ \widehat{F}_{C,i,j} \end{bmatrix} = \begin{bmatrix} \frac{\gamma_1}{\sqrt{\widehat{\sigma}_1^2+\epsilon}} & 0 & \dots & 0 \\ 0 & \frac{\gamma_2}{\sqrt{\widehat{\sigma}_2^2+\epsilon}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\gamma_C}{\sqrt{\widehat{\sigma}_C^2+\epsilon}} \end{bmatrix} \cdot \begin{bmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{C,i,j} \end{bmatrix} + \begin{bmatrix} \beta_1 - \gamma_1 \frac{\widehat{\mu}_1}{\sqrt{\widehat{\sigma}_1^2+\epsilon}} \\ \beta_2 - \gamma_2 \frac{\widehat{\mu}_2}{\sqrt{\widehat{\sigma}_2^2+\epsilon}} \\ \vdots \\ \beta_C - \gamma_C \frac{\widehat{\mu}_C}{\sqrt{\widehat{\sigma}_C^2+\epsilon}} \end{bmatrix}$$

which can be implemented as a 1×1 convolution.

Usually a Batch Normalization layer follows a Convolutional layer, we can therefore fuse them into one. Given a Convolutional layer with weights W and bias b, we can substitute both layer with a new Convolutional layer of weights W_{fuse} and bias b_{fuse} defined as:

$$W_{fuse} = \gamma \frac{W}{\sqrt{\widehat{\sigma}^2 + \epsilon}} \tag{2}$$

$$b_{fuse} = \gamma \frac{b - \widehat{\mu}}{\sqrt{\widehat{\sigma}^2 + \epsilon}} + \beta \tag{3}$$

3 Bias propagation

This step is necessary if biases are presents in the model's hidden layers, or are introduced by the fusion of batch normalization layers. Neurons with zeroed-out channels might have non-zero bias, and so they will fire a constant output value. Hence, a neuron cannot immediately be removed if the corresponding bias is nonzero. These values, however, can be propagated and accumulated into the biases of the next layer. This operation can be repeated until all of the biases have been propagated to the last layer of the network. After a bias has been propagated, it can then be set to zero in the original neuron, which in turn allows the removal of the whole weight channel.

3.1 Linear layers

We denote as $L_1 = \langle A, a \rangle$ and $L_2 = \langle B, b \rangle$ two sequential linear layers. A and a denote the weight matrix and bias vector of L_1, of size $N \times M$ and N respectively. B and b denote the weight matrix and bias vector of L_2 of size $T \times N$ and T respectively. We also denote as f the activation function (e.g. ReLU). A forward pass for L_1 consists in:

$$y = f(xA^T + a) \tag{4}$$

(where x represents an input vector of size M) and for L_2:

$$z = yB^T + b \tag{5}$$

Focusing on Equation 4, we can visualize the vector-matrix product:

$$y = \begin{bmatrix} x_0 & x_1 & \dots & x_M \end{bmatrix} \cdot \begin{bmatrix} A_{0,0} & A_{1,0} & \dots & A_{N,0} \\ A_{0,1} & A_{1,1} & \dots & A_{N,1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{0,M} & A_{1,M} & \dots & A_{N,M} \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix}^T = \begin{bmatrix} xA_0^T + a_0 \\ xA_1^T + a_1 \\ \vdots \\ xA_N^T + a_N \end{bmatrix}^T$$

We now suppose that some output channel of A has been zeored-out following the application of some pruning criterion, e.g. every entry in A_1 is zero. The multiplication becomes:

$$y = \begin{bmatrix} x_0 & x_1 & \dots & x_M \end{bmatrix} \cdot \begin{bmatrix} A_{0,0} & 0 & \dots & A_{N,0} \\ A_{0,1} & 0 & \dots & A_{N,1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{0,M} & 0 & \dots & A_{N,M} \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix}^T = \begin{bmatrix} xA_0^T + a_0 \\ a_1 \\ \vdots \\ xA_N^T + a_N \end{bmatrix}^T$$

We now focus on the forward pass of L_2. As example, we analyze what happens with the first neuron B_0. If we rewrite Equation 5 focusing on B_0 we obtain:

$$z_0 = f(xA_0^T + a_0)B_{0,0} + \boldsymbol{f(a_1)B_{0,1}} + \dots + f(xA_N^T + a_N)B_{0,N} + b_0 \tag{6}$$

We now focus on the forward pass of L_2. As example, we analyze what happens with the first neuron B_0. If we rewrite Equation 5 focusing on B_0 we obtain:

$$z_0 = f(xA_0^T + a_0)B_{0,0} + \boldsymbol{f(a_1)B_{0,1}} + \dots + f(xA_N^T + a_N)B_{0,N} + b_0 \tag{7}$$

The term $f(a_1)B_{0,1}$ is a constant which can be accumulated into b_0. The same reasoning can be extended to all neurons in L_2, by adding $f(a_1)$ multiplied with the respective incoming weight to the neuron bias. The new set of biases $\widehat{b}$ for the layer can be written as:

$$\widehat{b} = \begin{bmatrix} b_0 + f(a_1)B_{0,1} \\ b_1 + f(a_1)B_{1,1} \\ \vdots \\ b_T + f(a_1)B_{T,1} \end{bmatrix}^T$$

and the original bias a_1 can be set to zero in L_1, resulting in $\widehat{a} = [a_0, 0, a_1, \ldots, a_N]$. This procedure can be applied when multiple neurons are pruned in L_1 and the general rule to obtain the updated biases $\widehat{b}$ is as follows:

$$\widehat{b} = \begin{bmatrix} b_0 + \sum_i f(a_i)B_{0,i} \\ b_1 + \sum_i f(a_i)B_{1,i} \\ \vdots \\ b_T + \sum_i f(a_i)B_{T,i} \end{bmatrix}^T$$

where i represents the indices of zeroed channels in L_1. After the bias propagation procedure, the layers L_1 and L_2 can be replaced by $\widehat{L}_1 = \langle A, \widehat{a} \rangle$ and $\widehat{L}_2 = \langle B, \widehat{b} \rangle$ respectively.

3.2 Convolutional layers

A similar reasoning can be applied for convolutional layers. However, the propagation process needs to take into account whether the convolution employs zero-padding on the input tensor or not.

For the sake of simplicity, using the same notation of Section 3.1, let us consider two sequential convolutional layers $L_1 = \langle A, a \rangle$ and $L_2 = \langle B, b \rangle$. We also assume that L_1 has one input channel and two output channels (A has shape $2 \times 1 \times H_1 \times W_1$ and a is a vector of length 2), while L_2 has two input channels and one output channels (B has shape $1 \times 2 \times H_2 \times W_2$, and b is a vector of length 1).

The forward pass for L_1 is:

$$y = \left(\begin{bmatrix} x * A_0 \\ x * A_1 \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \right)^T = \left(\begin{bmatrix} F^0 \\ F^1 \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \right)^T = \begin{bmatrix} F^0 + a_0 \\ F^1 + a_1 \end{bmatrix}^T \tag{8}$$

where $*$ represents the convolution operation and x is a properly sized input. In this context, the addition operation $+$ between the resulting feature map $F^i = x * A_i$ and the corresponding bias value a_i will perform a shape expansion of a_i to match the feature map shape, for example:

$$F^i + a_i = \begin{bmatrix} F^i_{0,0} & \ldots & F^i_{0,H_{out}} \\ \vdots & \ddots & \vdots \\ F^i_{W_{out},0} & \ldots & F^i_{W_{out},H_{out}} \end{bmatrix} + \begin{bmatrix} a_i & \ldots & a_i \\ \vdots & \ddots & \vdots \\ a_i & \ldots & a_i \end{bmatrix} \tag{9}$$

We now assume that the second channel A_1 of L_1 has been zeroed out after the application of some pruning criterion, hence if we consider $F^1 + a_1$ we obtain:

$$F^1 + a_1 = \begin{bmatrix} 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{bmatrix} + \begin{bmatrix} a_1 & \ldots & a_1 \\ \vdots & \ddots & \vdots \\ a_1 & \ldots & a_1 \end{bmatrix} = \begin{bmatrix} a_1 & \ldots & a_1 \\ \vdots & \ddots & \vdots \\ a_1 & \ldots & a_1 \end{bmatrix} \tag{10}$$

Thus, y becomes:

$$\begin{bmatrix} F^0 + a_0 \\ \overline{a_1} \end{bmatrix}^T \tag{11}$$

where $\bar{}$ denotes that the element shape has been expanded.

We now analyze what happens with L_2. For the sake of simplicity, we assume that $W_{out} = H_{out} = 3$, that $W_2 = H_2 = 2$ and that every value of B is equal to 1. We also consider a stride value of 1 for L_2.

Convolution without padding (or "same" padding): This is the simpler case, and it is similar to the linear layers (Section 3.1). The forward pass of L_2 can be expressed as follows:

$$z = f(F^0 + a_0) * B_{0,0} + \boldsymbol{f(\overline{a_1}) * B_{0,1}} + b_0 \tag{12}$$

The factor $f(\overline{a_1}) * B_{0,1}$ is constant and can be accumulated into b_0. Visualizing it, we obtain:

$$\widehat{b}_0 = \begin{bmatrix} f(a_1) & f(a_1) & f(a_1) \\ f(a_1) & f(a_1) & f(a_1) \\ f(a_1) & f(a_1) & f(a_1) \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + b_0 = \begin{bmatrix} 4f(a_1) & 4f(a_1) \\ 4f(a_1) & 4f(a_1) \end{bmatrix} + b_0 \tag{13}$$

In this case, the updated bias can be converted as a scalar replacing the original value b_0: given that the resulting matrix is constant, we can directly factor out $4f(a_1)$ and set a_1 to 0 in L_1, obtaining a new bias $\widehat{b}_0 = 4f(a_1) + b_0$ which will be used from now on in L_2[1].

The same reasoning can be extended to the case of multiple neurons in the convolution layer and multiple pruned channel in the preceding layer: each bias value will be updated according to the rule in Equation 13. The general rule to obtain the new bias vector $\widehat{b}$ can be expressed as follows:

$$\widehat{b} = \begin{bmatrix} b_0 + \sum_i f(\overline{a_i}) * B_{0,i} \\ b_1 + \sum_i f(\overline{a_i}) * B_{1,i} \\ \vdots \\ b_{C_{out}} + \sum_i f(\overline{a_i}) * B_{C_{out},i} \end{bmatrix} \tag{14}$$

where i represents the indices of the zeroed output channels in L_1.

Convolution with zero-padding If the convolution applies zero-padding to the input values, then the bias cannot be accumulated into a scalar, as the resulting matrix will not be constant. To show this, we rewrite Equation 13 applying a zero-padding of size 1 along each dimension of the input tensor:

$$\widehat{b}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & f(a_1) & f(a_1) & f(a_1) & 0 \\ 0 & f(a_1) & f(a_1) & f(a_1) & 0 \\ 0 & f(a_1) & f(a_1) & f(a_1) & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + b_0 = \begin{bmatrix} f(a_1) & 2f(a_1) & 2f(a_1) & f(a_1) \\ 2f(a_1) & 4f(a_1) & 4f(a_1) & 2f(a_1) \\ 2f(a_1) & 4f(a_1) & 4f(a_1) & 2f(a_1) \\ f(a_1) & 2f(a_1) & 2f(a_1) & f(a_1) \end{bmatrix} + b_0 \tag{15}$$

[1]Notice that this can be applied for every choice of values for B. Of course the resulting bias factor will change accordingly

In this case, the new bias value need to be maintained in a matrix form, i.e.:

$$\widehat{b}_0 = \begin{bmatrix} f(a_1)+b_0 & 2f(a_1)+b_0 & 2f(a_1)+b_0 & f(a_1)+b_0 \\ 2f(a_1)+b_0 & 4f(a_1)+b_0 & 4f(a_1)+b_0 & 2f(a_1)+b_0 \\ 2f(a_1)+b_0 & 4f(a_1)+b_0 & 4f(a_1)+b_0 & 2f(a_1)+b_0 \\ f(a_1)+b_0 & 2f(a_1)+b_0 & 2f(a_1)+b_0 & f(a_1)+b_0 \end{bmatrix} \tag{16}$$

To obtain the updated biases in case of multiple neurons and multiple channels, the same rule of Equation 14 can be applied, keeping in mind that in this case it will result into a tensor of shape $C_{out} \times H_{out} \times W_{out}$ instead of a vector. This introduces a constraint on the feature map size, hence the model can only ever be used at a fixed input size. However, given that the whole simplification procedure is executed on an already trained model, before deploying to production, it should not represent a major issue.

3.3 Residual connections

While the above process works fine for simple feed-forward models, special care must be taken to handle residual connections. As an example, let us consider the case of two linear layers $L_1 = \langle A, a \rangle$ and $L_2 = \langle C, c \rangle$, whose outputs y and t are summed together in a residual connection, followed by another layer $L_3 = \langle B, b \rangle$:

$$y + t = \left(\begin{bmatrix} xA_0^T & + & a_0 \\ 0 & + & a_1 \\ \vdots & & \\ 0 & + & a_{N-1} \\ xA_N^T & + & a_N \end{bmatrix} + \begin{bmatrix} 0 & + & c_0 \\ \hat{x}C_1^T & + & c_1 \\ \vdots & & \\ 0 & + & c_{N-1} \\ \hat{x}C_N^T & + & c_N \end{bmatrix} \right)^T \tag{17}$$

where 0 denotes that a channel was pruned. The residual (sum) operation introduces a new constraint: only biases corresponding to matching pruned channels in L_1 and L_2 can be propagated to the next layer. To see why, we can rewrite Equation 17 as Equation 7 and obtain:

$$\begin{aligned} z_0 =& f(xA_0^T + a_0 + c_0)B_{0,0} + f(a_1 + \hat{x}C_1^T + c_1)B_{0,1} + \cdots + \\ &+ \boldsymbol{f(a_{N-1} + c_{N-1})B_{0,N-1}} + f(xA_N^T + a_N + \hat{x}C_N^T + c_N)B_{0,N} + b_0 \end{aligned} \tag{18}$$

It is clear that even if multiple channels are pruned from L_1 and L_2, only the factor $f(a_{N-1} + c_{N-1})B_{0,N-1}$ becomes a constant. In this case, we opt not to propagate any bias and employ an expansion scheme (Sec. 4.1) to achieve a speed-up in the convolution operations anyways.

4 Channels removal

Once the biases have been propagated and removed from the hidden layers, the weight matrices corresponding to zeroed channels can actually be removed. The process, which we call simplification, is actually quite simple. For each layer L, we denote with W^L the corresponding weight tensor, with shape $N \times I \times W \times H$ for convolutional layers and $N \times I$ for linear layers. The simplification consists of two steps:

1. Remove all the input channels corresponding to zeroed channels in the previous layer (none if it is the input layer):

$$\widehat{W}^L = \left[W_{0,i}^L, W_{1,i}^L, \ldots, W_{N,i}^L \right] \tag{19}$$

   where i is the indices of the remaining output channels in W^{L-1}. The resulting weight tensor will be of shape $N \times I_s \times W \times H$ for convolutional layers and $N \times I_s$ for linear layers, where $I_s \leq I$ is the number of remaining output channels in the previous layer $L - 1$.

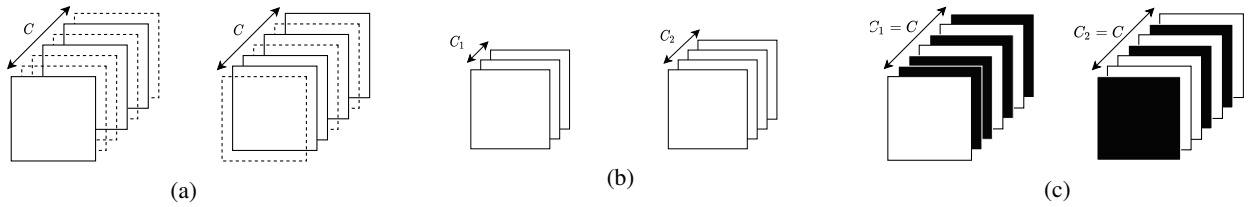

Figure 1: Pruned weight matrices: a dotted line indicates a zeroed channel *(a)* simplified weight matrices *(b)* expanded weight matrices: black slices mean that the channel is a zero matrix *(c)*.

2. Remove all the output channels corresponding to zeroed neurons:

$$\widetilde{W}^L = \left[\widehat{W}^L_j\right]_{\forall j \in J} \tag{20}$$

where J is the set of indices corresponding to the remaining output channels. The resulting tensor will be of shape $N_s \times I_s \times W \times H$ for convolutional layers and $N_s \times I_s$ for linear layers, where $N_s \leq N$ is the number of remaining (non-zero) output channels of L.

4.1 Residual connections

Residual or skip connections introduce a constraint on the output size of a layer. A residual connection consists in the sum of the output of two layers L_1 and L_2. As an example, we assume L_1 and L_2 to be convolutional layers, with their respective output Z_1 and Z_2 being of size $C_1 \times H \times W$ and $C_2 \times H \times W$ (ignoring the batch size) . To be able to compute $Z_1 + Z_2$, C_1 must be equal to C_2. However, after the simplification step, it is possible that the output sizes differ, depending on whether some output channels in L_1 and/or L_2 were removed.

To address this issue, we perform an expansion operation on the output tensors. Assuming the original size (before the simplification) was C, then Z_1 and Z_2 are expanded to the original number of channels before performing the addition. The process is illustrated in the Fig. 1. After the expansion step, we sum the layers biases (which were not propagated as explained in Sec. 3.3).

5 Implementation

The implementation of the simplification procedure can be found at `https://github.com/EIDOSlab/simplify`.