Group Equivariant Convolutional Networks supplementary material

Taco S. Cohen T.S.COHEN@UVA.NL
University of Amsterdam

Max Welling M.WELLING@UVA.NL
University of Amsterdam
University of California Irvine
Canadian Institute for Advanced Research

This document contains supplementary material for the paper "T.S. Cohen, M. Welling, Group Equivariant Convolutional Networks. Proceedings of the International Conference on Machine Learning (ICML), 2016".

1. Equivariance Derivations

We claim in the paper that planar correlation is not equivariant to rotations. Let $f : \mathbb{R}^2 \to \mathbb{R}^K$ be an image with K channels, and let $\psi : \mathbb{R}^2 \to \mathbb{R}^K$ be a filter. Take a rotation r about the origin. The ordinary planar correlation $\star$ is not equivariant to rotations, i.e., $[L_r f] \star \psi \neq L_r[f \star \psi]$. Instead we have:

$$\begin{aligned}
[[L_r f] \star \psi](x) &= \sum_{y \in \mathbb{Z}^2} \sum_k L_r f_k(y) \psi_k(y - x) \\
&= \sum_{y \in \mathbb{Z}^2} \sum_k f_k(r^{-1}y) \psi_k(y - x) \\
&= \sum_{y \in \mathbb{Z}^2} \sum_k f_k(y) \psi_k(ry - x) \\
&= \sum_{y \in \mathbb{Z}^2} \sum_k f_k(y) \psi_k(r(y - r^{-1}x)) \\
&= \sum_{y \in \mathbb{Z}^2} \sum_k f_k(y) L_{r^{-1}} \psi(y - r^{-1}x)) \\
&= f \star [L_{r^{-1}} \psi](r^{-1}x) \\
&= L_r [f \star [L_{r^{-1}} \psi]](x)
\end{aligned} \tag{1}$$

Line by line, we used the following definitions, facts and manipulations:

1. The definition of the correlation $\star$.
2. The definition of L_r, i.e. $L_r f(x) = f(r^{-1}x)$.
3. The substitution $y \to ry$, which does not change the summation bounds since rotation is a symmetry of the sampling grid $\mathbb{Z}^2$.
4. Distributivity.
5. The definition of L_r.
6. The definition of the correlation $\star$.
7. The definition of L_r.

A visual proof can be found in (Dieleman et al., 2016).

Using a similar line of reasoning, we can show that pooling commutes with the group action:

$$\begin{aligned}
P L_h f(g) &= \max_{k \in gU} L_h f(k) \\
&= \max_{k \in gU} f(h^{-1}k) \\
&= \max_{hk \in gU} f(k) \\
&= \max_{k \in h^{-1}gU} f(k) \\
&= P f(h^{-1}g) \\
&= L_h P f(g)
\end{aligned} \tag{2}$$

2. Gradients

To train a G-CNN, we need to compute gradients of a loss function with respect to the parameters of the filters. If we use the fast algorithm explained in section 7 of the main paper, we only have to implement the gradient of the indexing operation (section 7.1, "filter transformation"), because the 2D convolution routine and its gradient are given.

This gradient is computed as follows. The gradient of the loss with respect to cell i in the input of the indexing operation is the sum of the gradients of the output cells j that index cell i. On current GPU hardware, this can be implemented efficiently using a kernel that is instantiated for each cell j in the *output* array. The kernel adds the value of the gradient of the loss with respect to cell j to cell i of array that holds the gradient of the loss with respect to the input of the indexing operation (this array is to be initialized at zero). Since multiple kernels write to the same cell i, the additions must be done using atomic operations to avoid concurrency problems.

Alternatively, one could implement the filter transformation using a precomputed permutation matrix. This is not as efficient, but the gradient is trivial, and most computation graph / deep learning packages will have implemented the matrix multiplication and its gradient.

3. G-conv calculus

Although the gradient of the filter transformation operation is all that is needed to do backpropagation in a G-CNN for a split group G, it is instructive to derive the analytical gradients of the G-correlation operation. This leads to an elegant "G-conv calculus", included here for the interested reader.

Let feature map k at layer l be denoted $f^l_k = f^{l-1} \star \psi^{lk}$, where f^{l-1} is the stack of feature maps in the previous layer. At some point in the backprop algorithm, we will have computed the derivative $\partial L / \partial f^l_k$ for all k, and we need to compute $\partial L / \partial f^{l-1}_j$ (to backpropagate to lower layers) as well as $\partial L / \partial \psi^{lk}_j$ (to update the parameters). We find that,

$$\begin{aligned}\frac{\partial L}{\partial f^{l-1}_j(g)} &= \sum_{h,k} \frac{\partial L}{\partial f^l_k(h)} \frac{\partial f^l_k(h)}{\partial f^{l-1}_j(g)} \\ &= \sum_{h,k} \frac{\partial L}{\partial f^l_k(h)} \left[\sum_{h',k'} \frac{\partial f^{l-1}_{k'}(h')}{\partial f^{l-1}_j(g)} \psi^{lk}_{k'}(h^{-1}h') \right] \\ &= \sum_{h,k} \frac{\partial L}{\partial f^l_k(g)} \psi^{lk}_j(h^{-1}g) \\ &= \left[\frac{\partial L}{\partial f^l} \star \psi^{l*}_j \right](g)\end{aligned} \tag{3}$$

where the superscript * denotes the involution

$$\psi^*(g) = \psi(g^{-1}), \tag{4}$$

and ψ^l_j is the set of filter components applied to input feature map j at layer l:

$$\psi^l_j(g) = (\psi^{l1}_j(g), \ldots, \psi^{lK_l}_j(g)) \tag{5}$$

To compute the gradient with respect to component j of filter k, we have to G-convolve the j-th input feature map with the k-th output feature map:

$$\begin{aligned}\frac{\partial L}{\partial \psi^{lk}_j(g)} &= \sum_h \frac{\partial L}{\partial f^l_k(h)} \frac{\partial f^l_k(h)}{\partial \psi^{lk}_j(g)} \\ &= \sum_h \frac{\partial L}{\partial f^l_k(h)} \left[\sum_{h',k'} f^{l-1}_{k'}(h') \frac{\partial \psi^{lk}_{k'}(h^{-1}h')}{\partial \psi^{lk}_j(g)} \right] \\ &= \sum_h \frac{\partial L}{\partial f^l_k(h)} f^{l-1}_j(hg) \\ &= \left[\frac{\partial L}{\partial f^l_k} * f^{l-1}_j \right](g)\end{aligned} \tag{6}$$

So we see that both the forward and backward passes involve convolution or correlation operations, as is the case in standard convnets.

References

Dieleman, Sander, De Fauw, Jeffrey, and Kavukcuoglu, Koray. Exploiting Cyclic Symmetry in Convolutional Neural Networks. In *International Conference on Machine Learning*, 2016.