CIE6032 and MDS6232: Homework #2

Due on Sunday, November 4th, 2018, 5:30pm (DYB 224)

1. **[50 Points]**

Equivariance is an appealing property when design neural network operations. It means that transforming the input image (e.g., translation) will also transform the output feature maps similarly after certain operations.

Formally, denote the image coordinate by $x \in \mathbb{Z}^2$, and the pixel values at each coordinate by a function $f: \mathbb{Z}^2 \rightarrow \mathbb{Z}^K$, where K is the number of image channels. A convolution filter can also be formulated as a function $w: \mathbb{Z}^2 \rightarrow \mathbb{Z}^K$. Note that f and w are zero outside the image and filter kernel region, respectively. The convolution operation (correlation indeed for simplicity) is thus defined by

$$[f * w](x) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K} f_k(y) w_k(y - x). \tag{1}$$

a. **[15 Points]** Let L_t be the translation $x \rightarrow x + t$ on the image or feature map, i.e., $[L_t f](x) = f(x - t)$. Prove that convolution has equivariance to translation:

$$[[L_t f] * w](x) = [L_t [f * w]](x), \tag{2}$$

which means that first translating the input image then doing the convolution is equivalent to first convolving with the image and then translating the output feature map. **(Hints: Use the formula (1) for the proof.)**

b. **[15 Points]** Let $L_\mathbf{R}$ be the 90°-rotation on the image or feature map, where

$$\mathbf{R} = \begin{bmatrix} \cos(\pi/2) & -\sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{bmatrix}, \tag{3}$$

then $[L_\mathbf{R} f](x) = f(\mathbf{R}^{-1}x)$. However, convolution is not equivariant to rotations, i.e., $[L_\mathbf{R} f] * w \neq L_\mathbf{R} [f * w]$, which is illustrated by Figure 1 ((a) is not equivalent to (b) rotated by 90°). In order to establish the equivalence, the filter also needs to be rotated (i.e. (b) is equivalent to (c) in Figure 1). Prove that:

$$[[L_\mathbf{R} f] * w](x) = L_\mathbf{R} [f * [L_{\mathbf{R}^{-1}} w]](x) \tag{4}$$

(Hints: Use the formula (1) for the proof.)

c. **[20 Points]** To make convolution equivariant to rotations, we need to extend the definition of convolution and transformation. Recall a group $(G, \otimes)$ in algebra is a set G , together with an binary operation $\otimes$, which satisfies four requirements:

Closure $a \otimes b \in G, \forall a, b \in G$.

Associativity $(a \otimes b) \otimes c = a \otimes (b \otimes c), \forall a, b, c \in G$.

Identity element There exists a unique $e \in G, e \otimes a = a \otimes e = a, \forall a \in G$.

Inverse element $\forall a \in G, \exists a^{-1} \in G, a \otimes a^{-1} = a^{-1} \otimes a = e$.

We can formulate 90°-rotation and translation by a group $(G, \otimes)$ consisting of

$$g(r,u,v) = \begin{bmatrix} \cos(r\pi/2) & -\sin(r\pi/2) & u \\ \sin(r\pi/2) & \cos(r\pi/2) & v \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where $r \in \{0,1,2,3\}$ and $(u,v) \in \mathbb{Z}^2$. $G = \{g\}$ and $\otimes$ is matrix multiplication. Translation is a special case of G when $r = 0$ (i.e., $g(0,u,v)$) and rotation is a special case of G when $u = v = 0$ (i.e., $g(r,0,0)$).

A key concept is to extend the definition of both the feature f and the filter w to G . Imagine the feature map is duplicated four times with rotation of 0°, 90°, 180°, and 270°. Then $f(g)$ is the feature values at particular rotated pixel coordinate, and the convolution operation becomes

$$[f * w](g) = \sum_{\mathbf{h} \in G} \sum_{k=1}^{K} f_k(\mathbf{h}) w_k(g^{-1}\mathbf{h}). \tag{6}$$

A rotation-translation $u \in G$ on the feature map is thus $[L_u f](g) = f(u^{-1}g)$.
Prove that under such extensions, the convolution is equivariant to rotation-translation:

$$[[L_u f] * w](g) = L_u [f * w](g). \tag{7}$$

Briefly explain how to implement this group convolution with traditional convolution and by rotating the feature map or filter.
(Hints: Please read the paper "*__Group Equivariant Convolutional Networks__*").

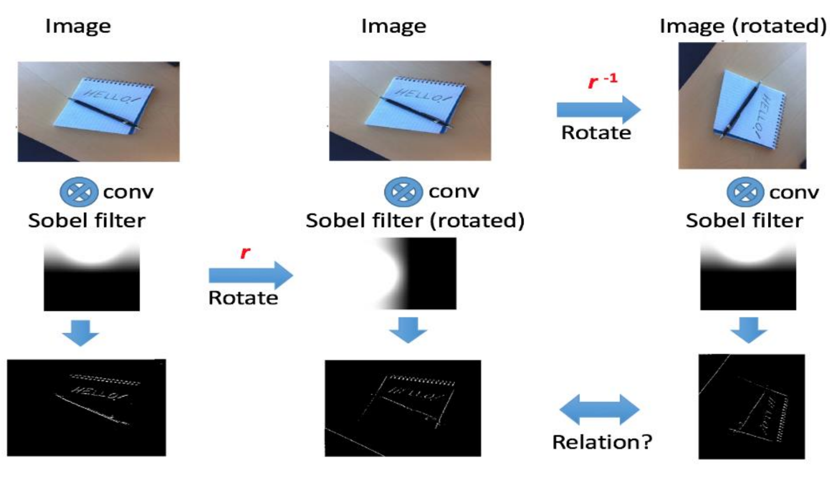

Figure 1: Equivariance relationship between convolution and rotation. (a) An image is convolved with a Sobel filer to detect horizontal edges. (b) The filter is rotated counterclockwise and then convolves the original image. (c) The image is first rotated clockwise, then it is convolved with the filter.

2. [50 Points]

In this problem we will implement a convolutional neural network to solve the problem of image classification on the CIFAR-10 dataset.

The network should consist of two convolutional layers and two fully connected layers. The first convolution has a kernel size of 5, stride of 1 and outputs 6 channels (without padding); the second convolution shares the same kernel setting and outputs 16 channels. The two fully connected layers have the number of output neurons as 120, 84, respectively. The last layer before the loss is an additional FC layer with ten outputs (the number of object classes on CIFAR-10). Max pooling with stride of 2 is appended after each convolution. And we use ReLU as the activation function throughout the network. Note that you may optionally add the dropout layer after each fully connected layer, depending on the training behavior of your network. The loss is a softmax loss. Other training specifications (recommended and yet not guaranteed): we use SGD as the optimization method; the base learning rate is 0.001. Momentum and weight decay are set to be 0.9 and 0.0005. The learning rate policy is step, which is the way of reducing learning rates. You can refer to AlexNet paper or AlexNet tutorial for details. The initial weights of filters are Gaussian distributed with zero mean and 0.01 std. You do not need to worry about these settings. They are already set by default in the code we provide.

The starter code can be downloaded at https://github.com/icemansina/CUHK_SZ_DL/blob/master/Assignment2/simple_cifar10.py , which includes the general framework, dataset loading and evaluation process.

Note: We have intentionally designed the network to be small so that you can even run the network fast in CPU mode. For adventurers, you can augment the width of the network above to get a sense of the super-efficiency of GPU acceleration (also report it in your submission).

a. **[10 points]** Set up the network architecture specified above. Draw the training curve and test accuracy for at least 5 epoches. Visualize the filters learned in the first convolutional layer. Report and analyze the performance improvement of using softmax loss and regularization.

b. **[20 points]** Weight initialization. A good initialization of the network should ensure the variance of signals among layers constant. For a convolution or fully connected layer, we have

$$\mathrm{y}_l = \mathbf{W}_l \mathrm{x}_l + \mathrm{b}_l , \tag{8}$$

where y_l, x_l are the output and input maps of layer l, W_l, b_l denote the weights and bias of the filter. We also have $\mathrm{x}_l = f\left(\mathrm{y}_{l-1}\right)$ where f is the activation function (we use ReLU throughout the discussion). Let y_l, x_l, w_l represent the random variables of each element in y_l, x_l, $\mathbf{W}_l$, respectively. Assume that w_l has zero mean and elements in x_l, $\mathbf{W}_l$ are mutually independent and these two are also independent. Then we have:

$$Var[y_l] = n_l Var[w_l x_l], \tag{9}$$

where n_l is the number of neurons in layer l. In case you are not familiar with the conclusion above, there is a blog. Verify the following:

$$Var[y_l] = n_l Var[w_l] E[x_l^2] \tag{10}$$

$$E[x_l^2] = \frac{1}{2} Var[y_{l-1}]. \tag{11}$$

Putting Eqn. 11 back into Eqn. 10 and considering all layers from 1 to L, we have derived the key to the initialization design:

$$Var[y_L] = Var[y_1]\left(\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l]\right). \tag{12}$$

A good initialization method should avoid reducing or magnifying the magnitudes of input signals exponentially. A sufficient condition from Eqn. 12 is thereby:

$$\frac{1}{2} n_l Var[w_l] = 1, \quad \forall l \tag{13}$$

From above, we derive a proper way of initializing the weights in the network. Similar conclusion holds for a backpropagation case (see paper for details).

In this question, you need to do two things.

(d1) Prove Eqn. 10 and Eqn. 11.

(d2) Now based on the network designed previously, *implement such an idea* to initialize weights in your network and verify if such a design works. Report the test accuracy compared with the result in question a.

c. **[10 points]** Batch normalization. We have discussed the technique of batch normalization in the lecture. Add BN layer (using existing library in Mxnet) after each convolution and FC layer (except the very last FC for classification) and verify the effectiveness of BN via test accuracy.

d. **[10 points]** Try other ideas. For example, investigate the effect of data augmentation (rotation, scaling, etc.); different optimization policies (Adam, RMSProp, etc.). Note that for each new component, you have to write your own code/layer instead of borrowing from the existing library. And some brief analysis should also be appended.