Final Project: Implementing and Training Convolutional Neural Networks

Purposes

Understanding the structure, forward propagation, training and tuning Convolutional Neural Networks (CNNs).

Implementation Tasks

Class of CNN (15 points). We are going to implement the famous LeNet-5 structure from scratch. The main structure is presented in Figure 1. It consists of two convolutional layers (with ReLU actitivations), two pooling layers (max pooling), and a fully connected (FC) neural network which has two hidden layers and one output layer. The hidden layers also have ReLU activations, and the output layer has identity activation (the Gaussian connections in the figure are ignored). The first convolutional layer has 6 filters each of which has the size $5 \times 5 \times k$ wherein k is the number of color channels of the input image. The second convolutional layer has 16 filters each of which has the size $5 \times 5 \times 6$. In both of the max pooling layers, the kernel size is 2×2 and the stride is 2.

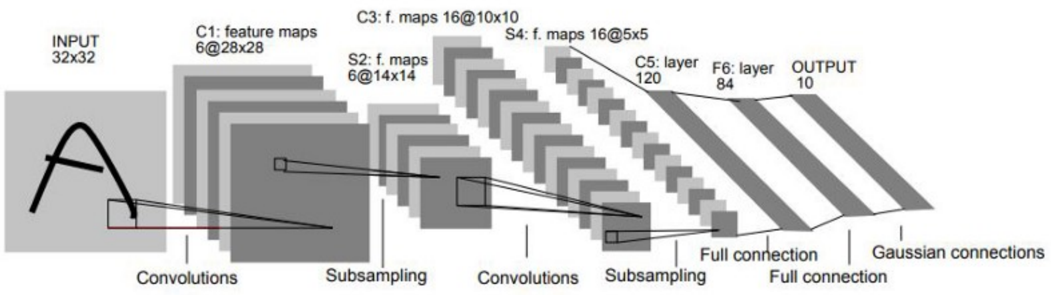

Figure 1: The architecture of LeNet-5.

You are required to write a Python class for CNN, and create an object of it as LeNet-5. The class should have the following components:

- (3 points) A *constructor* which creates a CNN according to the given structure. Here, you may design the specification of the CNN structure.
- (6 points) A *forward* function which perform a forward propagation on the neural network.
- (6 points) The *data fields* which explicitly define each layer as well as the activations in the neural network.

Make sure that your convolutional layers can carry multiple filters. You may define auxiliary or help classes and functions. *However, using any existing machine learning library except data downloading and preprocessing is forbidden and leads to a 0 grade.* Specifically, you need to write a function which performs 3D convolutions with padding in the first two dimensions. Note that a 3D convolution can be viewed as the sum of the 2D convolutions of all channels.

Convolution with padding. We give an example here. The convolutional layer in a CNN often performs convolutions with 0 padding (taught in the class). Figure 2 illustrates a convolution of two matrices with padding 1. That is, the first matrix is extended by adding 1 row to the top and bottom respectively and 1 column to the left and right respectively. The number 1 is specified by the padding size.

Bias. You use a single bias for each filter. You need to figure out the partial derivative $\partial L/\partial b$ for each bias in backpropagation.

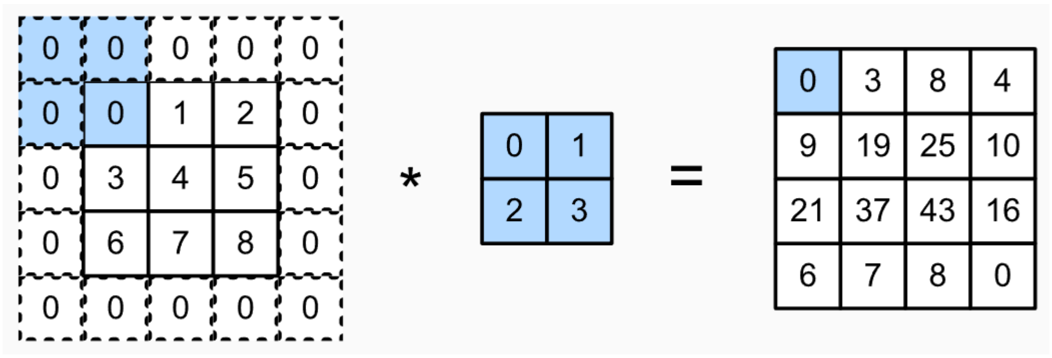

Figure 2: convolution of two matrices with padding 1.

Backpropagation on CNN (25 points). These are the most difficult tasks in this project. Your implementation will be evaluated in the following way.

- (10 points) Write a backpropagation function for a single convolution. You need to perform a convolution of the rotated filter and the matrix of the partial derivatives to the output. Make sure your padding size is correct. Please read the slides for the details.
- (5 points) Write a backpropagation function for a single max pooling operation.
- (10 points) Write a function for a complete backprapagation on a CNN. This function should be a member function of your CNN class.

There are some details need more attention. In the backpropagation of a single convolution, firstly, for each filter F_i, you need to compute a gradient matrix $\frac{\partial L}{\partial F_i}$. Since we may have multiple channels, the convolution should be 3D. An illustration is given in Figure 3. Secondly, you may have multiple filters. Then, the gradient to each channel of the input can be obtained as the sum of the backpropagation of each channel of the output image. An illustration is given in Figure 4.

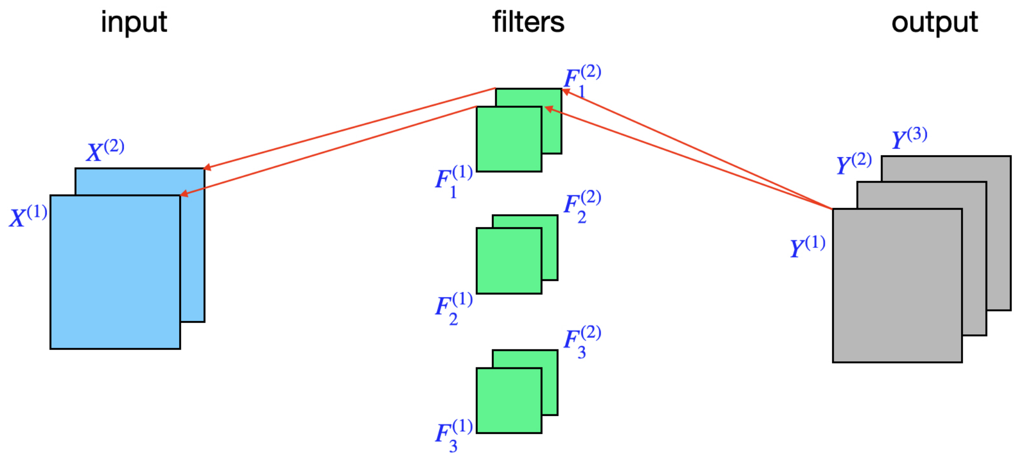

Figure 3: We use $X^{(i)}$ to indicate the i-th channel of the input, $F_j^{(i)}$ to indicate the i-th channel of the j-th filter, and $Y^{(i)}$ to indicate the i-th channel of the output. We know that $Y^{(i)} = X^{(1)} \circledast F_i^{(1)} + X^{(2)} \circledast F_i^{(2)} + b_i$ where b_i is the bias of F_i. Hence, if we have the gradient $\partial L/\partial Y^{(i)}$, the gradient $\partial L/\partial F_i^{(j)}$ can be computed as $\partial L/\partial F_i^{(j)} = X^{(j)} \circledast (\partial L/\partial Y^{(i)})$. For example, in the above figure, we have that $\partial L/\partial F_1^{(1)} = X^{(1)} \circledast (\partial L/\partial Y^{(1)})$ and $\partial L/\partial F_1^{(2)} = X^{(2)} \circledast (\partial L/\partial Y^{(1)})$.

Testing Your Implementation (10 points). You are required to test your implementation using the following two datasets.

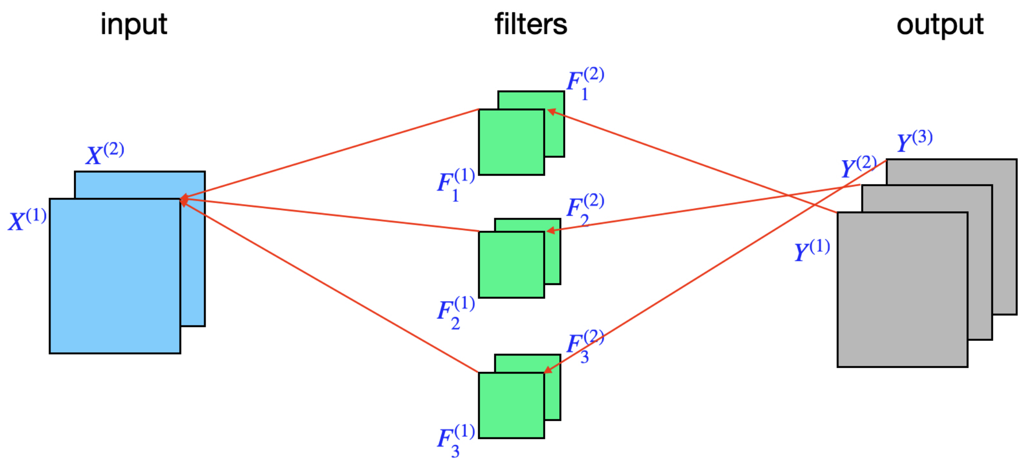

Figure 4: We know that $X^{(i)} \circledast F_j^{(i)}$ is computed as a part of $Y^{(j)}$ for all filters. Therefore, the calculation of $\partial L/\partial X^{(i)}$ requires to use $F_j^{(i)}$ and $Y^{(i)}$ for all possible j. By the chain rule, we have that $\partial L/\partial X^{(i)} = \sum_j (\partial L/\partial Y^{(j)})(\partial Y^{(j)}/\partial X^{(i)}) = \sum_j ((F_j^{(i)})' \circledast (\partial L/\partial Y^{(j)}))$ where $(F_j^{(i)})'$ denotes the 180^o rotation of $F_j^{(i)}$, and the convolution requires a $(m-1)$-padding such that m is the width/height of the filter. For example, in the above figure, we have that $\partial L/\partial X^{(1)} = (F_1^{(1)})' \circledast (\partial L/\partial Y^{(1)}) + (F_2^{(1)})' \circledast (\partial L/\partial Y^{(2)}) + (F_3^{(1)})' \circledast (\partial L/\partial Y^{(3)})$.

- **MNIST.** We already used these handwriting digits in the midterm project and Assignment 3. This time, we will see the power of LeNet-5 on recognizing the numbers. Make sure that all images are represented in the scale of 32×32, otherwise the scale of the input to the first FC layer will not be correct.

- **CIFAR-10.** The CIFAR-10 dataset consists of 60000 images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. More information can be found in its homepage. Each image is of the size 32×32 and has 3 color channels. Therefore, the filters in the first convolutional layer should be of the size $5 \times 5 \times 3$. You may find an example of training a LeNet-5 by CIFAR-10 using PyTorch via this link.

In both of the tests, we are required to use mini-batches for training. The batch size may be 64, 100 or 128 which depends on the configuration of your computer. A learning rate from 0.001 to 0.01 will do well, and you may use the Adam algorithm for optimization.

Project Report (10 points). Please write a project report which contains the following content at least.

- Description and explanation of the data structure for CNN in your implementation. Points (out of the 6 points for the data fields) will be taken off if the data structure design is not appropriate, e.g., redundant information is kept, redundant computation is required on the data structure, wrong representation of CNN, etc..

- (5 points) A proof to clarify that the (3D) size of $\partial L/\partial F$ computed by the method taught in our class is always same as the size of F for each filter F. We assume that the filter size is $m \times m \times k$.

- (5 points) A proof to clarify that the (3D) size of $\partial L/\partial X$ computed by the method taught in our class is always same as the size of X for any input image X. We assume that the image size is $n \times n \times k$.

- A section showing the experimental report of the two test cases. (Points are shared with the implementation tests.)

- A section of clarifying the contribution of each team member. Each team member will receive a grade based on her/his contribution to the project. We expect team members to have equal contributions.

The student who has almost no contribution or only give the wrong implementation will receive a 0 grade. Besides, no AI tool is allowed to use in the completion of this project. Any plagiarism or AI generated content will lead to a 0 grade to all team members.