ConvNet Lab for DD2427

Part 1 : ConvNet building blocks

Part 1.1 : Convolution

Part 1.1.1 : Convolution by a single filter

Changing size

By applying with convolution a filter on an image will reduce it :

H'' = H – (H'-1)
W'' = W – (W'-1)

The filter can't be applied on the extreme borders because it can't get a minus/plus value around.

Laplacian operator

The filter emphasizes the high frequencies that are the edges (high difference between the left and right color)

Trick for keeping size of image

In order to keep the same size of image and avoid finishing with no image at all, we can enlarge the initial image.

We need to enlarge the image on each border of the size (H'-1)/ for the height and (W'-1)/2 for the width. We fill the pixels with the exact value of the neighbor (the values of the pixels of the border of the image).

Part 1.1.2 : Convolution by a filter bank

The number of feature channels K is 3 because we have 3 layers in the filter.

Analyze

- Filter 1 : The result is the same as previous task
- Filters 2-3 : Both filters emphasize as well the borders but it's less strong since the derivative is only on one dimension each time.

Sobel filter

There are not a lot of differences since we have just strength the pixel above and below the center pixel.

Feature channels

The third dimension of W should be equal to the C features of the input.

Filter with 3 dimensions

We are interested in filter with 3 dimensions in order to apply them on the three color channels of the input image (RGB).

Part 1.1.3 : Convolving a batch of images

The result is applied on each image since the size of the result y is :

size(y) = Height * Width * #filters * #images

The number of images is equal to #images thus the filter is applied on every images.

We prefer working on batches of images in order to be faster during computation.

Part 1.2 : Non-linear activation (ReLU)

Linear layers

If all layers are linear, the output would have been linear as well. In this case, we won't be able to classify certain instances properly.

ReLU

This filter prefers really strong edges in the image.

We negate the Laplacian in order to emphasize black edges. Thus, we can remove some « noisy » edges.

The distributions of the color are symmetric when I use a linear function and is not with a non-linear function.

Bias

The response is more selective because I can see that less edges are kept due to the bias.

Extra functionality

In all cases, the results are worth if I applied the functions on the output convolution.

The results on the input image are:

- Max pooling: Emphasize long edges (animal instead of marks)
- Average pooling: Removing all edges

Part 2: Backpropagation

The output derivatives have the same size as the parameters in the network because we are using the chain rule. Thus, the derivatives are computed on each parameters after the other.

Each step, the vector p_l contains the derivative from the output to the layer l in the network. We are going away from the output are getting closer to the input. We can do this since we only need the previous derivative of the previous layer for computing the next one.

Part 2.1: Backward mode verification

The file checkDerivativeNumerically verifies if the numerically approximation of the derivative is close to the analytic derivative. The expression should be done inside the loops (around line 20).

The results are good when the absolute difference is equal to 0.

Part 2.2: Backpropagation

Error

We want to propagate the error from the output to the input. We want to do a “reverse” operation. By adding a parameter to a function at the end, you do the reverse.

No, the absolute difference is equal to 0.

Part 3: Learning a ConvNet for text deblurring

Part 3.1: Preparing the data

The size of the images are 64*64. There are 2550 images for the training set and 902 for the validation set. (imdb.images.set)

The intensity 0 corresponds to white. The convolution operator can't compute close to the boundaries. We have seen in the first question that the image is reduced if we don't enlarge it.

Part 3.2: Defining a ConvNet architecture

Every time we compute a convolution with a filter, the output image will lose 2 pixels. At the end, there will be no image any more.

We don't apply a ReLU on the output image in order to not lose information. Else the image would have been only black and white.

Table analyze

There are 3 layers in this network.

The support for each intermediate tensor is:

1> 3 x 3 x 32
2> 3 x 3 x 32
3> 3 x 3 x 1

The number of feature channels should be equal to the dimension of the filters in the next layer.

Operations needed (N: fixed size of the input image)
Layer 1: NxNx1x 2x3x3x32 = 576xN^2
Layer 2: NxNx32x 2x3x3x32 = 18432xN^2
Layer 3: NxNx32x 2x3x3x1 = 576xN^2

Receptive field

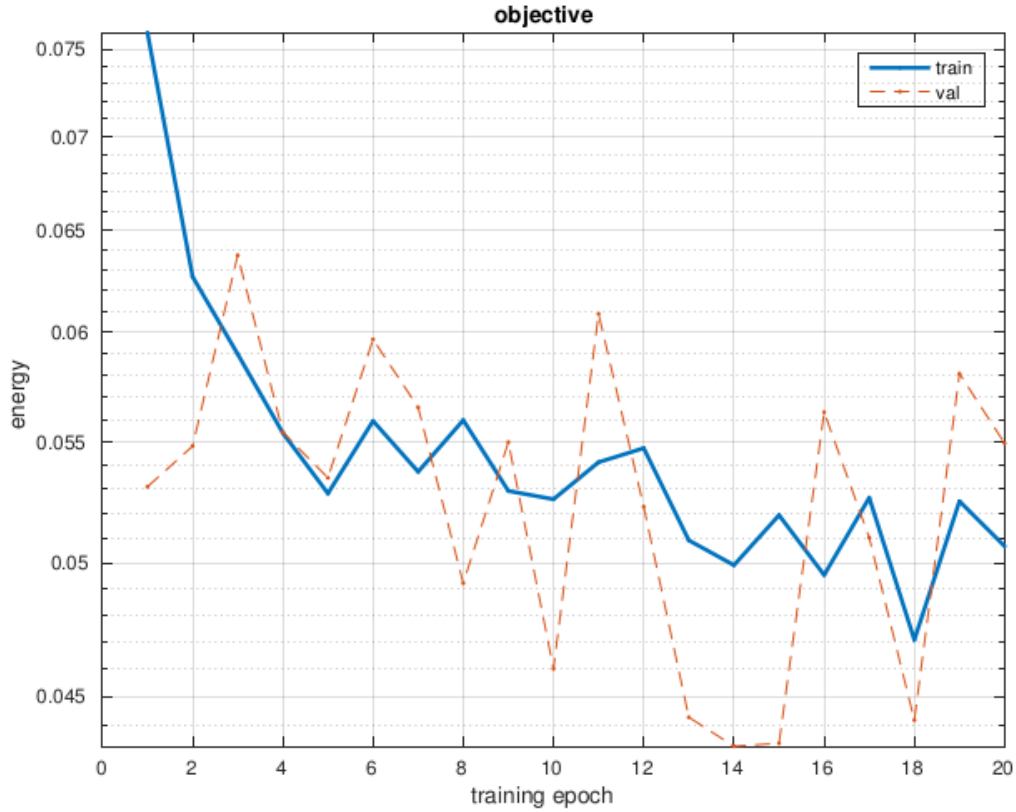
The size is 7.

A large receptive field is better since the blur is spread and the filter need to look further. For knowing the best size, we should test them all.

If we change the receptive field, we need to adapt the pad. With a receptive field of 5 for each layer we get a rf size: 5 → 9 → 13

It's bigger.

Part 3.3: Learning the network



The training energy is decreasing quickly at the beginning and slower after. We finish with an energy around 0.05 .The validation result has sharper change than the training one.

Part 3.4: Evaluate the model

The resulting images are better but the blur is still present. Most of the times, it's still hard to read.

The training set has better looking but the error is almost the same (8%). The validation set is oscillating around the training set (sometimes above, sometimes below).