CSE 559A: Fall 2017

Problem Set 5

Due: (Nov 30 + 1 Day Extension) = Dec 1, 2017. 11:59 PM

Instructions

Please read the late submission and collaboration policy on the course website:
`http://www.cse.wustl.edu/~ayan/courses/cse559a/`

All code will be tested on Anaconda for Python 3.6, available from `https://www.anaconda.com/download`.

This problem set is provided as a ZIP file. You are reading the main problem set PDF in the root `pset5/` directory. Look in the `pset5/code` sub-directory for code for different problems, with data in `pset5/code/inputs`. You will be required to modify and fill-in various functions in the `.py` files in the code directory. Running this code (e.g., with `python ./prob1.py` and so on) will create output images in the `pset4/code/outputs` directory.

Complete the source code files in `pset5/code/` by filling out the required functions. Create a PDF report file with LaTeX. Use the template provided on the course website (scroll down to the resources section at the bottom). In particular, make sure you fill out your name/wustl key (top of the .tex file) to populate the page headers. Also fill out the final information section describing how long the problem set took you, who you collaborated / discussed the problem set with, as well as any online resources you used. The main body of the report should contain responses to questions and results and figures for the programming questions as asked for. Place this report file as `solution.pdf`, and include it in a ZIP file with your source code and results, and upload it to blackboard. Try to avoid including the `inputs` directory in your zip file, since this time it has a large dataset file.

PROBLEM 1 **(Total: 25 points)**

In Lectures 18 and 19 you learned about the Simple Linear Iterative Clustering (SLIC) approach to segments an image into "superpixels," or perceptually meaningful regions that still respect contours of the objects in the image. SLIC solves for the assignment of K cluster labels to N pixels ($K << N$) that minimizes the squared distance between an augmented (r, g, b, y, x) vector at every pixel and the mean of all augmented vectors assigned to the same label.

(a) The algorithm starts with a collection of K cluster centers initialized on an equally sampled grid across the image. Implement the `get_cluster_centers method`, which takes as an input the image and number of clusters and returns a $K \times 2$ array of cluster centers. We provide the $get\_gradients$ method from Problem Set 1, which you should use to refine your initial, evenly spaced, grid as explained in Lecture 19 so that none of your initial cluster centers lie on a sharp boundary. The support code will call your `get_cluster_centers method` and create the output images 'outputs/prob1a_K_centers.jpg' for multiple values of K. Include these images in your report. **(10 points).**

(b) Implement the `slic` function. It takes as input the image, number of clusters and cluster centers computed in part (a). It should return a $W \times H$ image where each pixel is assigned a label, $\{0, 1, \ldots K-1\}$. Your implementation should not search over all possible assignments of clusters to all possible pixels – instead, you should only consider assigning a cluster label to those pixels in a $2S \times 2S$ local window around that cluster center ($S = \sqrt{(N/K)}$) as discussed in Lecture 19. You should also experiment with different relative weights of the spatial component in the augmented vector as discussed in Lecture 18.

The support code will run your slic function with multiple values of K. Include in your report the output images, 'outputs/prob1b_K.jpg' for each different value of K. Also include the `spatial_weight` that you selected and explain why you selected that weight. **(15 points).**

NOTE: For the purpose of this assignment, you do not need to convert the RGB images to LAB color space.

PROBLEM 2 **(Total: 15 points)**

We begin by looking at our own implementation of an autograd system as described in the class. Most of the framework is implemented in `edf.py`. It is then used for training a classifier to label hand-written digits as $0 - 10$ (this is a smaller version of the MNIST dataset), using a neural-network with a single hidden layer in `mnist.py`. We strongly suggest you begin by looking at the existing code and trying to understand the implementation of various parts of the gradient computation and update pipeline.

(a) Try different parameters of batch size, learning rate, and number of hidden units, and comment on the relative performance of the classifier. Keep an eye on both the soft-max loss as well as the accuracy, for both the train and val sets. Look at the `xavier` function being used to initialize the weights. Why are the limits of the uniform distribution being chosen in that way ? **(5 points)**.

(b) The provided code implements batched stochastic gradient descent. Extend this to use momentum. Define the `init_momentum` and `momentum` functions in `edf.py`, and then un-comment the corresponding lines in `mnist.py`. Comment on how the behavior of training changes (consider combining momentum with different batch sizes and learning rates). **(10 points)**

PROBLEM 3 **(Total: 60 points)**

Implement a 2D convolution layer, as a class `conv2` in `edf.py`. The function will take two nodes as input, corresponding to an "image" f which is an array of size (Batch Size) $\times H \times W \times$ (Input-Channels), and a kernel k of size $K_1 \times K_2 \times$ (Input-Channels) $\times$ (Output-Channels). The layer should produce the output of a "valid" convolution (or rather a correlation) without padding, defined as follows:

$$g[b, y, x, c_2] = \sum_{k_y} \sum_{k_x} \sum_{c_1} f[b, y + k_y, x + k_x, c_1] k[k_y, k_x, c_1, c_2].$$

Implement the forward function **(20 points)**, as well as the backward function that back-propagates gradients to both the input image **(20 points)**, and to the kernel **(20 points)**. Test this layer with the `mnist_conv.py` file which attempts MNIST classification with a convolution layer as its single hidden layer.

New (V3): You may also test this layer with `mnist_conv2.py` file which uses a deeper network with multiple convolutional layers. This will let you check whether back-propagation to the inputs is implemented correctly in your layer.

(Extra Credit +10 points): To keep things simple, we made downsampling a separate layer in `edf.py`. But this also means that the conv layer is doing many more computations than it has to. Try to write a more efficient version of the conv layer that takes "stride" as input, and produces a downsampled convolved output.