EE5179 : Deep Learning for Imaging

Programming Assignment 2: Convolutional Neural Networks

Due by: September 23, 2022

Instructions

- Program in python for this assignment.
- Post any doubts you have on moodle. This will be helpful to your peers as well.
- Submit the codes (the actual notebook and a PDF version) and your assignment report in a zip file titled PA2_RollNumber.zip in the submission link provided on moodle.

Preliminaries

- It is recommended to use Google Colab (as in the tutorials) or Jupyter/iPython notebooks for the assignment. The notebook format is very convenient to work (and evaluate!) with and additionally, Colab provides GPU access as well. Click here to learn more about how GPUs can be accessed using Colab.
- The dataset can be downloaded from here or you can make use of the inbuilt dataset from Pytorch.
  Note: Check if the labels are in one-hot format, or appropriately convert them to one-hot format before training and testing the network.

1 MNIST classification using CNN

Build a simple network to classify MNIST data using the following architecture

- input
- conv1 (32 3×3 filters, stride 1, zero padding 1)
- 2×2 maxpool with stride 2
- conv2 (32 3×3 filters, stride 1, zero padding 1)
- 2×2 maxpool with stride 2
- fully connected layer (500 outputs)
- fully connected layer (10 outputs)
- softmax classifier

Separate the dataset into training, validation and test sets with equal representation from all 10 classes of images. Use ReLU non-linearity for training the neural network.

1. Show the plot of training error, validation error and prediction accuracy as the training progresses. At the end of training, report the average prediction accuracy for the whole test set of 10000 images.
2. Plot randomly selected test images showing the true and predicted class labels.
3. Report the dimensions of the input and output at each layer.
4. How many parameters does your network have? How many of thes are in the fully connected layers and how many are in the convolutional layers?
5. How many neurons does your network have? How many of these are in the fully connected layers and how many are in the convolutional layers?
6. Use batch-normalization. Does it improve the test accuracy? Does it affect training time?

2 Visualizing the Convolutional Neural Network

1. Plot the the conv1 layer filters. Do you observe interesting patterns?
2. Plot filters of a higher layer and compare them with conv1 layer filters.
3. Visualize the activations of the convolutional layers. What do you observe as you go deeper?
4. **Occluding parts of the image :** Suppose that the network classifies an image of a digit successfully. How can we be certain that it is actually observing the main part of the digit in the image as opposed to background or something else? One way of investigating this is by plotting the probability of the class of interest as a function of the position of an occluder object. Pick any 10 random images and occlude parts of the images iteratively with a patch (e.g. a grey patch). Observe the probability of the class of interest and plot the probability in a grid as a function of the patch position. (You can use x axis and y axis to denote the position of the patch in xy plane, and plot the probability as intensity). Based on this experiment, report if the learning meaningful or not.

3 Adversarial Examples

Adversarial examples are inputs to a neural network that result in an incorrect output from the network. For the following experiments, load the pre-trained MNIST model with the highest accuracy from the first problem. Do not change the trained weights in any of your operations.

3.1 Non-Targeted Attack

The idea is to generate an image that is designed to make the neural network produce a certain output. Initialize a matrix to be the same size of an MNIST image with Gaussian noise centered around 128. Let this matrix be $\mathbf{X}$. Beginning with this noise matrix, try to maximize the probability of this matrix to be categorized as some target class. Here target class can take values between 0 to 9. Define the cost function as :

$$C = logits[targetClass] \tag{1}$$

where *logits* is the network output before final softmax operation. Find the derivatives ($\mathbf{d}$) of the cost function with respect to the input x using backpropagation. Now, set

$$\mathbf{X} = \mathbf{X} + stepsize \times \mathbf{d} \tag{2}$$

This essentially tries to increase the target class score by gradient ascent.

1. Show the generated image for each of the MNIST classes.
2. Is the network always predicting *targetClass* with high confidence for the generated images?
3. Do the generated images look like a number? If not, can you think of some reason?
4. Plot the cost function. Is it increasing or decreasing?

3.2 Targeted Attack

Can we generate some adversarial example that looks like a particular digit, but the network will classify it as something else? For example, can we generate an image of digit 2, which the network will classify as 5? For this,change the cost function to:

$$C = logits[targetClass] - \beta \times MSE(generatedImage, targetImage) \tag{3}$$

where *targetImage* is what we want our adversarial example to look like and β is very small (e.g. 0.001, tune it according to the output). We are trying to increase the target class score for the generated image, but also want the generated image to look like some target image (by minimizing the MSE). Find the derivatives ($\mathbf{d}$) of the cost function with respect to the input x using backpropagation. Now, set

$$\mathbf{X} = \mathbf{X} + stepsize \times \mathbf{d} \tag{4}$$

1. Show the generated image for each of the MNIST classes. Do the generated images now look like a number?

3.3 Adding Noise

Initialize a zero matrix with the same size as your input. Let this matrix be $\mathbf{N}$. If $\mathbf{X}$ is your original input image of a particular class, generate

$$\mathbf{X}_N = \mathbf{X} + \mathbf{N} \tag{5}$$

$\mathbf{X}_N$ is now input to the neural network. In this experiment, we want our image of original class to be classified as *targetClass*. Similar to the previous method, the cost function will be:

$$C = logits[targetClass] \tag{6}$$

But this time, we will update the noise. Calculate the gradient ($\mathbf{d}$) of the cost function with respect to noise, and update noise as:

$$\mathbf{N} = \mathbf{N} + \alpha \times \mathbf{d} \tag{7}$$

You can make the noise value small, to make the generated image look very similar to the original image.

1. Show the generated adversarial image, and the noise for each of the classes of MNIST.

2. Sample a fixed set of 10 test examples from the dataset. Add the adversarial noise and classify. Show the true class and the predicted class. Repeat this for all the 10 generated adversarial noise matrices.

(In all the 3 parts, do not modify the trained network parameters. For the experiments in 3.1 and 3.2, calculate the gradient w.r.t. to the image, and for case 3.3 calculate the gradient w.r.t noise. Then update the image/noise only.)
