EE6132

Programming Assignment 2: Convolutional Neural Networks

Due Date: Oct 28, 2021

Instructions

1. You may do the assignment in MATLAB, Python.
2. For any questions, please schedule a time with TAs before deadline according to their convenience. Please use moodle discussion threads for posting your doubts and also check it before mailing to TAs, if the same question has been asked earlier.
3. Submit a single zip file in the moodle named as *PA2_Rollno.zip* containing report and folders containing corresponding codes.
4. Read the problem fully to understand the whole procedure.
5. Do not copy any part from any source including your friends, seniors or the internet.
6. Late submissions will be evaluated for reduced marks and for each day after the deadline we will reduce the weightage by 5%.

Dataset : You can make use of any library to load the data.

1 MNIST classification using CNN :

In this part we will explore CNNs for classifying MNIST data. For this assignment you can make use of any of your favourite deep learning libraries (**Tensorflow** or **Pytorch**. High-level API like **Keras** is **not** allowed. If you planning to use anything else, please contact the TAs before proceeding). Use Relu non-linearity for training the neural network.

Overall architecture : input - conv1 (32 3x3 filters, stride 1, zero padding of 1) - 2x2 maxpool with stride 2 - conv2 (32 3x3 filters, stride 1, zero padding of 1) - 2x2 maxpool with stride 2 - fully connected(500outputs) - fully connected(10outputs) - softmax-classifier

Submissions

- For the experiment above, you are required to 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.
- You should also plot randomly selected test images showing the true class label as well as predicted class label.

- Write down the dimensions of the input and output at each layer.
- Exactly how many parameters does your network have? How many of these are in the fully connected layers and how many are in the convolutional layers?
- Exactly 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?
- Use batch-normalization. Does it improve the test accuracy? Does it affect training time?

2 Visualizing Convolutional Neural Network :

- Plot the conv1 layer filters. Do you observe interesting patterns?
- Plot filters of a higher layer. Compare it with conv1 layer filters.
- Visualize the activations of the convolutional layers. What do you observe as you go deeper?
- **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. So, occlude parts of the original image iteratively with a patch (e.g. a grey patch), and look at the probability of the class of interest. Then, 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). Pick any 10 random images, and plot. Is the learning meaningful?

3 Adversarial Examples :

[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.]
Adversarial examples are inputs to a neural network that result in an incorrect output from the network.

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 of size of an MNIST image with Gaussian noise centered around 128. Let this matrix be $\mathbf{X}$. Starting with this noise matrix we will try to maximize the probability of this matrix being classified as some target class. Here target class can take values between 0 to 9. Define the cost function as :

$$C = logits[target\_class] \tag{1}$$

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

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

(So we are essentially trying to increase the target class score by gradient ascent)

Submission :

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

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:

$$C = logits[target\_class] - \beta \times MSE(generated\_image, target\_image) \tag{3}$$

where *target_image* is what we want our adversarial example to look like. You can use image of any other digit. So what were doing now is 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). (β is very small e.g. 0.0001, tune it according to the output) Find the derivatives (d) of the cost function with respect to the input X using backpropagation. Now, set -

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

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

Adding Noise : Initialize a matrix of the size of your input with zeros. Let us call this matrix **N**. Let your input image be **X** of original_class. Now obtain

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

$\mathbf{X}_N$ will be our input to the neural network. Now, we want our image of *original_class* to be classified as *target_class*. So, similar to the previous methods, our cost function will be:

$$C = logits[target\_class] \tag{6}$$

But, now 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.

Submission :

- Show the generated adversarial image, and the noise for each of the classes of MNIST.
- 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 first two cases, calculate the gradient w.r.t. to the image, and for the last case calculate the gradient w.r.t noise. Then update the image/noise only)

–end–