Programming Assignment-2
Convolutional Neural Networks

(Maitreya Suin, maitreyasuin21@gmail.com)

Due date: Sep 27th, 11:55 pm

Note:

1. 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.

2. Submit a single zip file in the moodle named as PA2_Rollno.zip containing report and folders containing corresponding codes.

3. Read the problem fully to understand the whole procedure.

4. Late submissions will be evaluated for reduced marks and for each day after the deadline we will reduce the weightage by 10%.

Loading the MNIST data:

You can make use of the binary files that you downloaded for the 1st assignment. You can also make use of the any other library to load the data.

(You are advised to use Google Colab(GPU) for significantly faster compilation of the code.)

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). 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 picking up on main part of the digit in the image as opposed to background or something else? One way of investigating which part of the image some classification prediction is coming from 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. Take 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.

a)**Non-Targeted Attack:** The idea is to generate some image that is designed to make the neural network have a certain output. Initialize a matrix of size of an MNIST image with Gaussian noise centered around 128. Let this matrix be *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]

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 ***x =x + step size ∗ d***

(So you are 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 a digit 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?

b)**Targeted Attack:** Can we generate some adversarial example that looks like some other number? 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] - ß * mse(generated_image,target_image)

target_image is what we want our adversarial example to look like. You can use image of any other digit.

So what we're 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 (minimize the mse).
(***ß*** 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 ***x =x + step size ∗ d***

Submission:

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

C)**Adding Noise:** Initialize a matrix of the size of your input with zeros. Let us call this matrix **noise**. Let your input image be **x** of original_class. Now obtain **x_noise = x + noise**. x_noise will be your input to your neural network.
Now, we want our image of original_class to be classified as target_class.
So, similar to above methods, our cost function will be:
C = logits[target_class]
But, now we will update the noise. Calculate the gradient(*d*) of the cost function with respect to noise, and update noise as:
noise = noise + α*d

(For faster convergence, you can take *α = 1.0 / (d.std() + 1e-8)*.
You can make the noise value small, to make the generated image 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)*

Optional:

1) ***Optimization over images:*** We want to create an image such that if we pass it through a trained network, it should maximize the probability of a particular class.
-Start with a zero image.
-Let $S_c(I)$ be the score for the particular class (logits[target_class])
- Objective function :

$$J(I) = S_c(I) - \lambda \|I\|_2^2$$

(We are using a simple regularizer. Any other sophisticated regularizer will improve performance)

-Compute the gradient(*d)* of *J(I*) w.r.t. the input image using backpropagation.

*x = x + step_size*d*

Repeat this for 200 iterations.

Submission:

- Show 10 images for each of the 10 classes. Do those images look like a number?
- Plot cost vs optimization iteration.

Now, instead of maximizing score for a particular class, you can try to maximize some intermediate conv layer features or neurons. What do you observe?

2) Can you think of some ways to prevent the adversarial attacks described above?