Programming Assignment-2: Convolutional Neural Networks

Prasan Shedligeri, ee16d409@ee.iitm.ac.in

Due date: Sep 17th, Sunday, 11:55pm

Note:

1. For any questions, please schedule a time with TAs before deadline according to their convenience. Please use moodle dicussion 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%.

---

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.

We will start with very basic networks, evaluate performance and then move on to powerful networks. Use Relu non-linearity for training the neural network.

Experiments

Baseline with one convolutional layer

Extract feature maps from the single convolutional layer and then pass it to the max pool layer to reduce the dimensions. Flatten the output and then use a 10 neuron fully connected layer to get the logits. Use a softmax classifier over these logits.
Overall architecture: input - conv (32 3x3 filters, stride 1, zero padding of 1) - 2x2 maxpool with stride 2 - fully connected (10 outputs) - softmax classifier

2 convolutional layers

Modify the above network to include one more conv layer with 32 3x3 filters. Use 2x2 maxpooling after the conv layer to reduce the dimensions. Classify using the features extracted from the 2 conv layers.
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 (10 outputs) - softmax classifier

2 convolutional layers + 1 hidden fully connected layer

Modify the above network by adding one fully connected layer with 500 neurons after the second convolutional layer.
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 (500 outputs)- fully connected (10 outputs) - softmax classifier

Loading the MNIST data

You can make use of the binary files that you downloaded for the first assignment. You can also make use of the python library $sklearn$ and download the MNIST data using $sklearn.datasets$. If using $sklearn$ then make sure that you shuffle the data properly and divide the whole data into training (50000 images), validation (10000 images) and test (10000 images) sets.

Submissions

For each of the experiments above, you are required to show the plot of training error, validation error and prediction accuracy over the training progress. For each of the experiments, 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. Use standard training techniques as described in the previous assignment. Use a $l2$ regularization over only the weights of the neural network.

2 Generating Adversarial examples

CNNs are discriminative models. They try to learn a separating hyperplane between each of the classes. This makes it easier to fool them. All one needs to do is find the noise matrix which when added to an image of one class (which is on one side of the hyperplane), gets classified with high probability as the wrong class (images has shifted to the other side of the hyperplane). We will call this matrix as adversarial noise.

Experiment

Load the pre-trained MNIST classification model from the previous problem. Use the model which gave the highest accuracy. Once the pre-trained model is loaded, make sure that network weights don't change during any of your operations.

We'll do two major changes to obtain the adversarial noise from the pre-trained network.

- Initialize a matrix of size of your input with a zeros. Let's call this matrix $noise$. Let your input image be x. Now obtain $x\_noise = x + noise$. x_noise will be your input to your neural network. Note that the values of x_noise should always be between 0 and 255.

- We need to fix a class for which you want to generate the adversarial noise. For instance, we will generate it for class 3. Pass x_noise through the trained neural network. You will get probabilities at the output layer. Now calculate the cost for each of the examples assuming that the true label is class 3, irrespective of whether the input x belonged to class 3 or not. Note that, the input example might be any digit but you are telling the network that it belongs to class 3. Since, all the weights of the network are fixed, it will try to tweak the only variable, in this case $noise$, such that x_noise is classified as class 3, irrespective of x. Using the calculated cost, get the gradients $grad$, with respect to $noise$. Now do a gradient ascent over the variable, $noise$.

  $$noise^{(k)} = noise^{(k-1)} + \alpha * grad$$

  where α is the step size and k=1,2,3,...

Submissions

- Plot the training as well as the validation error over the progress of optimization. Also show the accuracy of prediction.
- Show the generated adversarial noise as an image for each of the classes of MNIST. In all, you should show 10 images which are the adversarial noise for each of the class in MNIST.
- Sample a fixed set of 9 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.

3 Visualizing Convolutional Neural Network

In this section we will try to visualize what fires a particular neuron in the neural network.

Experiment

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. Initialize a matrix of size of an MNIST image with gaussian noise centered around 128. Let this matrix be x_init Starting with this noise

matrix we will try to maximize the probability of this matrix being classified as class $target$. Here target can take values between 0 to 9.

Let, $logits$ be your output at the last layer before softmax. Define, $cost = logits[:, target]$, where $target$ is a fixed class. Obtain the gradient of $cost$ with respect to x_init. Smoothen the gradients with a gaussian kernel of variance, $sigma = \frac{k * 4}{num\_iter + 0.5}$, where $k = 1, 2, 3, ..., num\_iter$, where num_iter is the number of iterations for which you will do the gradient ascent. Now, do a gradient ascent over x_init along with weight decay.

$$x\_init = (1 - \alpha) * x\_init$$
$$x\_init = x\_init + step\_size * gradients$$

where $step\_size = \frac{1.0}{std(gradients)}$ and α is weight decay parameter (typically 0.0001). Now, pass this updated x_init again to the pre-trained network and repeat the above steps for about 200 iterations. Make sure that the input always lies in the range of the minimum and maximum value of the input for which the original network was trained.

Submission

- For the above experiment you should report the 10 final x_init images obtained through maximizing each of the 10 neurons in the output of the neural network. Also show, the plot of the cost over optimization iterations. In this experiment, the cost should increase because we are doing gradient ascent.

- Now, instead of maximizing the neurons in the final layer, we will maximize the neurons at the output of maxpooling layer after the second convolutional layer. This is the output just before the fully connected layers. This output will be of shape [1,7,7,32], where 7 is the spatial size of the feature map and 32 is the number of feature maps. Maximize the activation of the central neuron of each of the feature maps by following the procedure described above. Report the final x_init for any of the 10 out of 32 feature maps.