CIS680: Vision & Learning
Assignment 2: CIFAR-10 and Vanishing Gradients
Due: Oct. 20, 2017 at 11:59 pm

Instructions

- This is an **individual** assignment. "**Individual**" means each student must hand in their **own** answers, and each student must write their **own** code in the homework. It is admissible for students to collaborate in solving problems. To help you actually learn the material, what you write down must be your own work, not copied from any other individual. You must also list the names of students (maximum two) you collaborated with.
- All the code should be written in Python. You are welcome to use Tensorflow or PyTorch to complete this homework.
- The CIFAR-10 dataset can be downloaded from [1]. In this homework, use train batch #1 for training and test batch for evaluation, each of which contains 1,000 images for 10 classes.
- You must submit your solutions online on **Canvas**. Compress your files into a ZIP file named "1_<penn_key>.zip", which should contain **1 PDF report** and **3 folders containing the Python code for each part**. Note that you should include all the figures in your report.

Overview

This homework aims at helping you familiarize yourself with deep learning libraries, such as Tensorflow or Pytorch. Along the road, you will first get your hands dirty with data pre-processing/augmentation. Once you are able to build a network from scratch, you will look into the gradients and experience the issue of vanishing gradients. After fixing

vanishing gradients, you can play with gradients to create adversarial images and verify phenomenon of cross-model perturbations.

This homework consists of three parts.

1. Pre-process and augment the images on-the-fly using Tensorflow or PyTorch.
2. Build your own network. Observe the problem of vanishing gradients and try to solve it.
3. Discover adversarial images by gradient ascent. Apply the adversarial images to another model to experiment the phenomenon of cross-model perturbations.

1 Data Pre-processing and Augmentation (20%)

Deep learning is a data-driven appraoch. That said, a large amount of images are essential to the success of applying deep learning in solving computer vision tasks. In this part, you will experience how image processing plays an important role in the performance of a convolutional neural network.

Note that in this part, you should **not** use Numpy/Scipy other than reading images.

1. (5%) Train a network with architecture shown in Table 1 using the raw images of CIFAR-10.

   Hint: Start with the demo code in the lecture "Practical Guide". Change the maximum of training iterations to 2,000 and steps of an epoch to 100 (with batch size 100). Also, be mindful of what's fed into the network.

   Plot the training accuracy over training iterations and report the final test accuracy.

2. (5%) Following the previous question. Instead of feeding raw images, normalizing images to zero mean and unit standard deviation.

   Plot the training accuracy over training iterations and report the final test accuracy. Explain the results compared to the previous question.

3. (5%) Following the previous question. In addition, flip the images randomly (with 50% chance) during training (before image normalization). Note that you should not flip the image during evaluation.

   Plot the training accuracy over training iterations and report the final test accuracy. Explain the results compared to the previous question.

Layers	Hyper-parameters
Convolution 1	Kernel size = (5, 5, 32). Followed by BatchNorm and ReLU.
Pooling 1	Average operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 2	Kernel size = (5, 5, 32). Followed by BatchNorm and ReLU.
Pooling 2	Average operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 3	Kernel size = (5, 5, 64). Followed by BatchNorm and ReLU.
Pooling 3	Average operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Fully Connected	Output channels = 64. Followed by BatchNorm and ReLU.
Softmax	(With fully connected layer) output channels = 10.

Table 1: Network architecture for part 1.

4. (5%) Following the previous question. In addition, zero pad the images with 4 pixels on each side (after normalization) and crop a random 32×32 region of images during training. Note that you should not flip/pad/crop images during evaluation.

   Plot the training accuracy over training iterations and report the final test accuracy. Explain the results compared to the previous question.

2 CNNs and Vanishing Gradients on CIFAR-10 (40%)

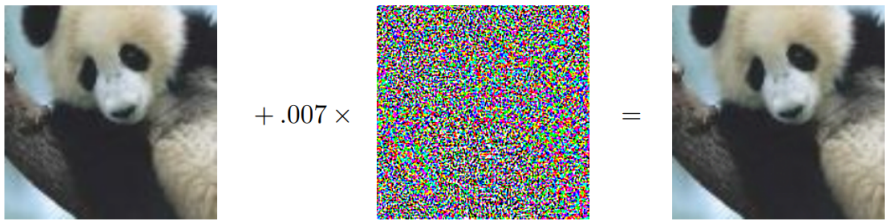
In this part, you will design your own convolutional neural network on the CIFAR-10 dataset. After adding multiple layers, you will observe the problem of vanishing gradients, i.e., gradients can not be back-propagated through many layers. Finally, you will try to discover a solution to overcome it.

1. (10%) Design a convolutional neural network composed of only convolution, BatchNorm, ReLU activations, pooling and softmax layers. (No fully connected layer except for softmax.) The test accuracy has to reach 68% with the same training procedure. Feel free to use any image pre-processing or augmentation methods.

   Provide a table that describes the network architecture. Plot the training accuracy over training iterations and report the final test accuracy.

2. (10%) Following the previous question, let's define the magnitude of gradients in one (convolutional) layer to be $G = \max_i |\frac{\partial \mathcal{L}}{\partial w_i}|$, where $\mathcal{L}$ is the cross-entropy loss, w_i is the i-th weight in the layer.

Figure 1: An adversarial example demonstrated in [2].



Plot the magnitude of gradients in the first and last convolutional layers over training iterations.

3. (10%) Between the first and second convolutional layers, add 20 convolutional layers, each of which has kernel size (5,5), padding = 2, stride = 1. Also, remove all the BatchNorm layers. Train the modified network on CIFAR-10 dataset again.

   Plot the training accuracy over training iterations. Plot the magnitude of gradients in the first and last convolutional layers over training iterations. Explain what happened and why.

4. (10%) Solve the problem you observed in the previous question without removing any convolutional layer or adding any BatchNorm layer.

   State your solution and explain why it works.

3 Adversarial Images (40%)

Besides monitoring gradients, you can actually manipulate the gradients to create troubles for the network. In this part, you will generate adversarial images for a CNN. Using those adversarial images, you'll be able to fool the network, making it wrong in an invisible way. Furthermore, these adversarial images would also affect other CNNs. It exposes a security issue of CNNs which hackers might take advantage of. An example is shown in Figure 1.

You are encouraged to read the relevant papers [2, 3] before solving this part.

1. (30%) Use the trained network from part 1 to generate adversarial image.

   The algorithm works as follows:

   (a) Feed a test image ($\mathbf{x}$) you want to perturb into the network.

   (b) Calculate the loss given the ground truth (y). Let the loss be $J(\mathbf{x}, y \mid \theta)$ where θ is the learned weights.

   (c) Compute the gradients with respect to the input image, i.e., $\bigtriangledown_{\mathbf{x}} J(\mathbf{x}, y \mid \theta)$.

   (d) Round up to a small perturbation and add to the input image, i.e., $\mathbf{x} = \mathbf{x} + \epsilon \text{sign}(\bigtriangledown_{\mathbf{x}} J(\mathbf{x}, y \mid \theta))$, where ϵ is a small constant of your choice.

   (e) Repeat (a)-(d) until the network classify the input image $\mathbf{x}$ as an arbitrary wrong category with confidence (probability) at least 99%.

   For each category, generate two such adversarial images. Display them side by side with the original unperturbed images. (40 images in total.) Describe the difference between adversarial images and original images.

2. (10%) Feed those adversarial images (generated in the previous question) into the CNN trained in part 2.

   Report the test accuracy and explain why the test accuracy is low.

References

[1] CIFAR-10. https://www.cs.toronto.edu/ kriz/cifar.html.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[3] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016.