CIS680: Vision & Learning
Assignment 2.a: Gradient manipulation.
Due: Oct. 16, 2018 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.
- There is no single answer to most problems in deep learning, therefore the questions will often be underspecified. You need to fill in the blanks and submit a solution that solves the (practical) problem. Document the choices (hyperparameters, features, neural network architectures, etc.) you made in the write-up.
- The assignment will describe the task on a high level. You are supposed to find out how to complete the assignment in the programming framework of your choice. While the text of the assignment should be sufficient to understand the task, you are welcome to read the references that will describe the used concepts in more detail.
- All the code should be written in Python. You should use either Tensorflow or PyTorch to complete this homework.
- The CIFAR-10 dataset can be downloaded from [1] and the MNIST dataset can be downloaded from [6]. PyTorch and Keras include Dataset classes for both datasets. You are free to use to them if you want.
- You must submit your solutions online on **Canvas**. Your code should contain **3 Python files**, one for each part. Submit your code compressed into a ZIP file named "1_<penn_key>.zip". Jupyter notebooks are acceptable. Submit your **PDF report** to a separate assignment called "HW2 PDF Submission". Note that you should include all results (answers, figures) in your report.

Introduction

In this homework, we are continuing to explore the mathematical tools on which deep learning is based, while also moving towards real world network architectures. We will train simple CNNs on the CIFAR-10 and MNIST datasets, experiment with gradient approximation techniques, and use gradients to create adversarial images.

1 Data Pre-processing and Augmentation (30%)

Large amounts of image data are essential for good performance of deep learning methods on computer vision tasks. However, there are simple techniques of "augmenting" data that allow to artificially enlarge an existing dataset to get even better performance. In this part, you will train a CNN on a complex, but small, dataset and experience how image processing plays an important role in the performance of the network.

1. (6%) Train a network with architecture shown in Table 1 using the raw images of CIFAR-10.

   Hint: You may start with the demo code in the lecture "Practical Guide". Change the maximum of training iterations to 10,000 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. (6%) Train the same network, but instead of feeding raw images, normalize 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. (6%) Train the same network, but in addition, flip the images randomly (with 50% chance) during training (before image normalization). Note that you should not flip the images during evaluation.

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

4. (6%) Train the same network, but in addition, pad the images with 4 zero 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.

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

Table 1: Network architecture for part 1.

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

2 Binary networks (35%)

Binary neural networks (BNNs, [5]) are neural networks in which some of the computation is binarized. This might be beneficial from a few perspectives, including faster computation, smaller power consumption and the regularization effect. In this question, you have to implement a network that has binary activation values: either +1 or -1. You will use the Sign function as the activation function:

$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{1}$$

The gradient of the Sign function is zero almost everywhere, which makes it impossible to train a BNN with gradient descent. Instead, a "straight-through" gradient approximator can be used [2, 4]:

$$\nabla_{\sim}(\text{Sign}) = \mathbf{1}_{|x|\leq 1}, \tag{2}$$

where $\mathbf{1}$ is the indicator function. In other words, the approximation of the gradient is one if the pre-activation value is within -1 to +1 range, and zero otherwise.

1. (10%) Train a CNN with architecture in table 2 on the MNIST dataset. Normalize the images in the $(-1, 1)$ range before feeding them in the network. Report training and testing curves. You should be able to reach 99% accuracy.

Layers	Hyper-parameters
Convolution 1	Kernel size = (3, 3, 32), Padding=1 (SAME), ReLU activation.
Convolution 2	Kernel size = (3, 3, 64), Stride=2, Padding=1, ReLU activation.
Convolution 3	Kernel size = (3, 3, 128), Stride=2, Padding=1, ReLU activation.
Convolution 4	Kernel size = (3, 3, 128), Stride=2, Padding=1, ReLU activation.
Convolution 5	Kernel size = (3, 3, 128), Stride=2, Padding=1, ReLU activation.
Fully Connected	Output channels = 100. ReLU activation
Fully Connected	Output channels = 10. Softmax activation

Table 2: Network architecture for part 2.

2. (20%) Implement the Sign activation function and the "straight-through" gradient estimator. For this, you will need to implement a custom gradient function (MySign.backward() in PyTorch, and MySignGrad() in TensorFlow).

3. (5%) Modify the CNN from part 1 of this question to use Sign instead of ReLU in all layers except the output layer. Report the testing and training accuracy plots of the resulting BNN. You should be able to reach comparable accuracy. Why does the approximate gradient that we use makes sense for training a neural network?

3 Adversarial Images (35%)

In this part you will see how you can use the gradients of the network to generate adversarial images. Using these images that look almost identical the original you will be able to fool different neural networks. You will also see that these images also affect different neural networks and expose a security issue of CNNs that malicious users can take advantage of. An example is shown in Figure 1. You are encouraged to read the relevant papers [3, 7] before solving this part.

1. (10%) Use the trained network from question 2 to generate adversarial images with constraints. The constraints that you have are

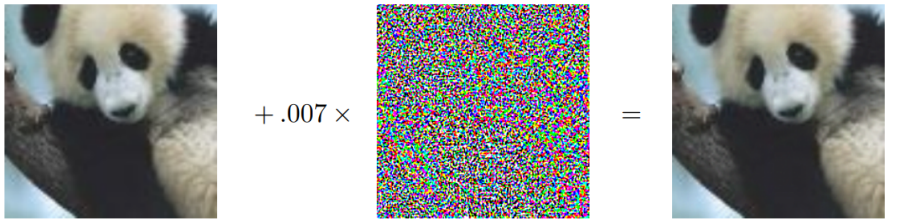
   (a) You are not allowed to erase parts of the image, i.e. $I_{pert} \geq I$ at each pixel location.

   (b) The perturbed image has to take valid values, i.e. $-1 \leq I_{pert} \leq 1$.

   The algorithm works as follows:

   (a) Let I be a test image of your dataset that you want to perturb that is classified correctly by the network. Let I_ϵ be the perturbation that you should initialize with zeros.

Figure 1: An adversarial example demonstrated in [3].



(b) Feed $I_{pert} = I + I_\epsilon$ in the network.

(c) Calculate the loss given the ground truth label (y_{gt}). Let the loss be $L(\mathbf{x}, y \mid \theta)$ where θ are the learned weights.

(d) Compute the gradients with respect to I_{pert}, i.e., $\nabla_{I_{pert}} L(I_{pert}, y_{gt} \mid \theta)$. Using backpropagation, compute $\nabla_{I_\epsilon} L(I_\epsilon, y_{gt} \mid \theta)$, i.e. the gradients with respect to the perturbed image.

(e) Round up to a small perturbation and add to the input image, i.e., $I_\epsilon = I_\epsilon + \epsilon \, \text{sign}\left(\nabla_{I_\epsilon} L(I_\epsilon, y_{gt})\right)$, where ϵ is a small constant of your choice.

(f) Repeat (a)-(d) until the network classify the input image I_{pert} as an arbitrary wrong category with confidence (probability) at least 90%.

Generate 2 examples of adversarial images. Describe the difference between adversarial images and original images.

2. (10%) For a test image from the dataset, choose a target label y_t that you want the network to classify your image as and compute a perturbed image. Note that this is different from what you are asked in part 1, because you want your network to believe that the image has a particular label, not just misclassify the image. You need to modify appropriately the loss function and then perform gradient descent as before. You should still use the constraints from part 1.

3. (10%) Repeat part 1, with the additional constraint that the perturbation has to be binary. You should use the binary activation from the previous question for this part.

4. (5%) Use the adversarial images you generated in the previous parts and feed them in the network from question 2. What do you observe?

References

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

[2] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

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

[4] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning.

[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[6] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

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