Preferred Networks Intern Screening 2018 Coding Task for Machine Learning / Mathematics Fields

Changelog

- 2018-05-01 : Initial version

Notice

- Please use one of the following programming languages:
  - C, C++, Python, Ruby, Go, Java, Rust
- In problems 1-3, please use only the standard library of your chosen language. You are not allowed to use multi-dimensional array libraries such as NumPy, Eigen, or any external libraries (unless you are the author of the library).
- In problem 4, feel free to use any library that you want.
- Do not share or discuss any details of this coding task with anyone.
- Please tackle the task by yourself. Do not share or discuss this coding task with anyone including other applicants. If we find an evidence of leakage, the applicant will be disqualified. If one applicant allows another applicant to copy the answer, both applicants will be disqualified.
- We expect you to spend up to two days for this task. You can submit your work without solving all of the problems. Please do your best without neglecting your coursework.

Things to submit

Please submit the following. The name of files and directories is also specified.

- Source code
  - Please create a directory called `src`. Put your code in this directory.
- A text file that explains how to build and run your code
  - Please name the file `README.txt`, `README.md`, or something like that.
  - Please specify the version of the language that you used. For example, Python 2.7, Python 3.6, C++11 etc.
  - You may write a short explanation of your code as auxiliary material.
- Regarding the image files generated in problem 3
  - Please create a directory called `problem3`. Put the generated images in this directory.
  - The image file corresponding to `pgm/1.pgm` should be located at `problem3/1.pgm`, the file corresponding to `pgm/2.pgm` should be at `problem3/2.pgm`, and so on.

- A report for problems 2,3,4
  - Please name the file `report.pdf`, `report.txt`, `report.md`, `report.doc`, or something like that.
  - Describe the accuracy of the predictor for problem 2, the comparison of results between the implemented algorithm and baseline algorithm for problem 3, and your experimental methods and results for problem 4.
  - The report should have either 1 or 2 pages in A4 format for problems 2,3,4, including figures and tables.

Evaluation

It is desirable that your submission satisfies the following. (These are not mandatory. Your submission does not need to satisfy all of them if you are too busy.)

- The source code is readable for other programmers.
- There is an appropriate amount of unit tests and/or verification code to ensure the source code is correct.
- It is easy to reproduce the experimental results.
- The submitted report is concise and easy to follow.

How to submit

- Create a zip archive with all of the submission materials and submit it on this form. Due date is Monday, May 14th, 2018, 23:59 JST.

Inquiry

- If you have any question on this problem description, please contact us at intern2018@preferred.jp. An updated version will be shared with all applicants if any change is made. Note that we cannot comment on the approach or give hints to the solution.

Task description

This task is about adversarial examples. You are given an image dataset of hand-written characters and a predictor (that is, a classifier) for the dataset. The objective is to modify the original input image by adding a small amount of noise so that the predictor will missclassify it. That is, we want to make an input image that will deceive the predictor.

Back-story: Thanks to the advancement of machine learning and deep learning, prediction models are going to be used in important applications, such as autonomous driving and image authentication. One motivation for adversarial examples is considering the risk that malicious users may abuse such predictors.

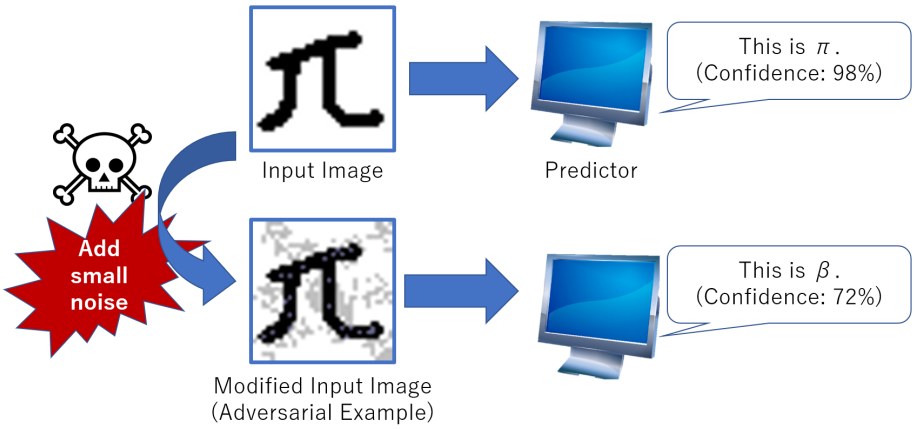

Figure 1: Adversarial examples

Problem 1

Implement functions that perform the following vector/matrix operations. You will then need to use these functions in the later problems. Further, write a simple unit test for each function. You may freely define the data types for vector/matrix and the interface of the functions. Assume that all vectors in this task are column-vectors.

- Sum of vectors: Given two vectors x and y, return the sum $x + y$.
- Vector-matrix product: Given a matrix A and a vector x, return the product Ax.
- Transpose: Given a matrix A, return its transpose $A^\top$.
- ReLU: Given a vector x, return a vector where the i-th element is $\max(0, x_i)$. The dimension of the returned vector must be equal to x.
- Softmax: Given a vector x, return a vector where the i-th element is $e^{x_i} / \sum_{j=1}^{n} e^{x_j}$. Here, n is the dimension of x.

Problem 2

The `pgm` directory contains hand-written images in PGM format. They are grayscale 32x32 images. The format of the file is as follows.

```
P2
```

```
32 32
255
(Pixel value at (1,1)) (Pixel value at (1,2)) ... (Pixel value at (1,32))
...
(Pixel value at (32,1)) (Pixel value at (32,2)) ... (Pixel value at (32,32))
```

Here, the first three lines are fixed text. As for the following lines, we denote a pixel at the i-th row in the j-th column by (i, j). A pixel value is an integer between 0 and 255, inclusive.

First, please implement a function that reads an image file and converts the image to a 1024-dimensional vector. Each element in the vector should be normalized as a real value between 0 and 1. Let us denote the converted vector by x. Then, $x_1, ..., x_{32}$ should correspond to the pixels (1,1), ..., (1,32) respectively, $x_{33}, ..., x_{64}$ to (2,1), ..., (2,32), and so on. To normalize the pixel values, divide the pixel value by 255.

The image files represent hand-written characters, consisting of 23 labels. In `labels.txt`, the label of each image file is specified. The first line of `labels.txt` is the label of `pgm/1.pgm`, the second line is that of `pgm/2.pgm`, and so on. Labels are numbered from 1 to 23.

`param.txt` contains the parameters of a predictor to classify these hand-written characters. Given a 1024-dimensional vector, the predictor returns a 23-dimensional real vector. The definition of the predictor is as follows. In the following, let $N = 1024$, $C = 23$, and $H = 256$ for simplicity.

- The predictor has six parameters $W_1, b_1, W_2, b_2, W_3, b_3$. Here, W_1, W_2, W_3 are parameter matrices with dimension $H \times N, H \times H, C \times H$, respectively. Further, b_1, b_2, b_3 are parameter vectors with dimension H, H, C, respectively.
- For an input vector $x \in \mathbb{R}^N$, we denote the returned vector of the predictor by $f(x) \in \mathbb{R}^C$. This is calculated as follows.
  - $a_1 = W_1 x + b_1$
  - $h_1 = \mathrm{ReLU}(a_1)$ (Here, ReLU is a ReLU function in problem 1.)
  - $a_2 = W_2 h_1 + b_2$
  - $h_2 = \mathrm{ReLU}(a_2)$
  - $y = W_3 h_2 + b_3$
  - For $i = 1, ..., C$, $f(x)_i = e^{y_i} / \sum_{j=1}^{C} e^{y_j}$.
- From the definition, the output $f(x)$ is a probability vector: Each element is non-negative and their sum is 1. The i-th element of $f(x)$ represents a probability that x belongs to the label i. We refer the index at which the probability is maximized $\arg\max_{i=1,...,C} f(x)_i$ to a predicted label.

`param.txt` describes the parameter values, which is formatted as follows. Each matrix/vector is represented by a space-separated text, possibly containing multi lines.

```
(Text representation of the matrix W_1, containing H lines)
(Text representation of the vector b_1, containing 1 line)
(Text representation of the matrix W_2, containing H lines)
(Text representation of the vector b_2, containing 1 line)
(Text representation of the matrix W_3, containing C lines)
(Text representation of the vector b_3, containing 1 line)
```

Please write code that reads the parameter file `param.txt` and the image files in `pgm` directory. Your code should output a predicted label $\arg\max_{i=1,\ldots,C} f(x)_i$. Further, output the accuracy of the predictor by comparing the predicted labels against the corresponding true labels in `labels.txt`. If your code is correct, the accuracy should be above 80%. Describe the accuracy of this predictor in a report.

Problem 3

Now you will implement a method for generating adversarial examples, called Fast Gradient Sign Method (FGSM). We create a small noise $\varepsilon \in \mathbb{R}^N$ and add the noise to the input vector x to form a new vector $\tilde{x} := x + \varepsilon$, which is very similar with x. The objective is that the prediction $f(\tilde{x})$ is far from correct prediction. In FGSM, we assume that the parameters of the predictor are known for simplicity. The methodology of FGSM is as follows.

- Let t be a correct label ($1 \leq t \leq C$). We define a real-valued function called cross-entropy loss $L(x)$, which represents how the prediction is far from the correct answer, as follows.
  - $L(x) := L := -\log f(x)_t = -y_t + \log \sum_{j=1}^{C} e^{y_j}$
- In FGSM, we take $\varepsilon := \varepsilon_0 \text{sign}(\nabla_x L)$. Here, $\varepsilon_0 > 0$ is a parameter of FGSM. sign is a function that, given a vector, returns a vector where each element is +1 if the input vector is positive, and -1 otherwise. Note that the dimention of the differential $\nabla_x L$ is equal to the dimension of x.
- $\nabla_x L$ can be calculated as follows.
  - $\nabla_y L = -\delta_t + f(x)$ (Here, let δ_t be a vector where only the t-th element is 1 and other elements are 0.)
  - $\nabla_{h_2} L = W_3^T (\nabla_y L)$
  - $\nabla_{a_2} L = \text{Backward}(\nabla_{h_2} L, a_2)$ (Here, for two vectors p and q with the same dimension, let $\text{Backward}(p, q)$ be a vector where the i-th element is p_i if q_i is positive, 0 otherwise.)
  - $\nabla_{h_1} L = W_2^T (\nabla_{a_2} L)$
  - $\nabla_{a_1} L = \text{Backward}(\nabla_{h_1} L, a_1)$
  - $\nabla_x L = W_1^T (\nabla_{a_1} L)$

Implement the algorithm of FGSM. Run with $\varepsilon_0 = 0.1$ for each image in `pgm` directory. Then, convert the generated vectors $\tilde{x}$ to images in PGM format. Submit the converted image files. Note that you will have to convert reals to integers between 0 and 255 in an appropriate way.

To verify if your FGSM implementation works well, let us consider a baseline method where we replace $\mathrm{sign}(\nabla_x L)$ with a random vector with ± 1 values in FGSM. Compare the performance of FGSM and the baseline algorithm with various ε_0. Write the result in the report briefly.

Problem 4

Please perform additional experiments with one of the themes below. Summarize the method and your experimental results in the report. In this problem, you may use external libraries.

- FGSM can be applied multiple times. Let $\tilde{x}_1$ be the output of FGSM for input x, $\tilde{x}_2$ be the output for $\tilde{x}_1$, and so on. Check if doing this improves the performance of the algorithm.
- Since the images in this task are binary, a method for defense towards adversarial examples generated by FGSM is that we binarize the input image before we use a predictor. Consider an attack method for this defense method, and report the performance.
- In the `extra` directory, there are additional parameter files. Suppose that the defense side, who will use a predictor, reads multiple parameter sets and decides a prediction by some sort of majority vote. Implement this defense method. Further, consider an attack method for this defense method if you need, and report the performance.

Remarks

We used HASYv2 dataset (ODbL license) for this coding task.

Thoma, Martin. (2017). HASYv2 - Handwritten Symbol database [Data set]. Zenodo. http://doi.org/10.5281/zenodo.259444

Fast Gradient Sign Method (FGSM) is described in the paper by Goodfellow et al: https://arxiv.org/pdf/1412.6572.pdf