Homework 0

Due: Tue. 09/29/2022, 11:59 PM

- Homework 0 tests your background knowledge on optimization, probabilities, simple neural nets, and programming experiences, which are prerequisites that we will not thoroughly review in this course. You are **allowed** to consult any external resources but you must cite them. You are also **allowed** to discuss with each other but you need to acknowledge them. However, your submission must be your own work; specifically, you must **not** share your code or proof. TAs and the Professor will not provide help on this problem set.
- We estimate that at most 7 hours are needed to finish it. If not, you might feel that the course is progressing too fast.
- Your submission should be a single PDF file, containing proof, code, and results. We recommend using Jupyter Notebook and export as PDF. We have provided a solution template as **HW0.ipynb**
- This homework is worth 5/100 of your final grade. However, it is **mandatory**, meaning you have to finish and submit it.
- If you are in the waitlist and intend to enroll, you have to submit this homework on time.

1 Optimization (7pt)

We consider the following constrained least square problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|Ax-b\|_2^2 \\ \text{subject to} \quad & x^T x \leq \epsilon, \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, \epsilon \in \mathbb{R}, \epsilon > 0$. The variable is $x \in \mathbb{R}^n$.
The solution to this problem will be used as a subroutine in subsequent geometry homeworks and we need an efficient solver. The solution to this optimization problem has to satisfy the Karush–Kuhn–Tucker (KKT) conditions:

$$\begin{cases} \nabla_x \mathcal{L}(x,\lambda) = 0 & (1) \\ x^T x \leq \epsilon & (2) \\ \lambda \geq 0 & (3) \\ \lambda(x^T x - \epsilon) = 0 & (4) \end{cases}$$

where $\mathcal{L}(x,\lambda) = \frac{1}{2}\|Ax-b\|_2^2 + \lambda(x^T x - \epsilon)$ is the Lagrangian of the problem.

1. Write down the gradient of the Lagrangian $\nabla_x \mathcal{L}$. (1pt)

   Next, We will solve this problem by considering 2 cases based on Condition (2).

2. Case 1: $x^T x < \epsilon$. Then, $\lambda = 0$ by Condition (4). To satisfy the KKT condition (1), it is equivalent to solving the unconstrained least square problem with objective $\frac{1}{2}\|Ax-b\|_2^2$. If its solution satisfies (2), we are done. Write down the closed-form solution to the unconstrained least-square problem. You do not need to show intermediate steps. (1pt)

3. Case 2: $x^T x = \epsilon$.

   (a) Set $\nabla_x \mathcal{L} = 0$, express x in terms of λ, i.e. $x = h(\lambda)$ (1pt)

   (b) Prove $h(\lambda)^T h(\lambda)$ is monotonically decreasing for $\lambda \geq 0$. *Hint:* You might need the fact that $A^T A = U \Lambda U^T$. (2pt)

   By the monotone property of $h(\lambda)^T h(\lambda)$, we can solve $x^T x = \epsilon$ by line search over λ (e.g., bisection method or Newton's iterative method). You will implement it in the following programming assignment.

4. (Programming assignment) Solve a provided instance of this problem. (2pt)

   - You can load the data by

```
npz = np.load('./HW0_P1.npz')
A = npz['A']
b = npz['b']
eps = npz['eps']
```

   - You are **not** allowed to use external optimization libraries that solve this problem directly. Linear algebra functions in numpy are allowed, e.g. numpy.linalg.eig, numpy.linalg.svd, numpy.linalg.lstsq, etc.

2 Probability (8pt)

Given a triangle $\triangle ABC$, one common geometric processing question is how to sample n points uniformly inside the triangle.
A straight-forward idea is to interpolate the vertices by barycentric coordinates: we first sample $\alpha, \beta, \gamma \sim U([0,1])$, where $U([0,1])$ is the uniform distribution on $[0,1]$; then, we normalize them to obtain $\alpha' = \frac{\alpha}{\alpha+\beta+\gamma}$, $\beta' = \frac{\beta}{\alpha+\beta+\gamma}$, $\gamma' = \frac{\gamma}{\alpha+\beta+\gamma}$ so that $\alpha' + \beta' + \gamma' = 1$; finally, we obtain a sample $P = \alpha' A + \beta' B + \gamma' C$ inside $\triangle ABC$. However, this straight-forward idea is wrong: it does not assure that P is uniformly sampled in $\triangle ABC$.

1. We seek to rigorously disprove the above algorithm for a lower-dimensional setup, which samples points on a line segment $\overline{AB}$: If we sample $\alpha, \beta \sim U([0,1])$ and normalize by $\alpha' = \frac{\alpha}{\alpha+\beta}$ and $\beta' = \frac{\beta}{\alpha+\beta}$ to obtain $P = \alpha' A + \beta' B$, then P is not uniformly distributed between A and B. For simplicity, we assume that $A = 0$ and $B = 1$.

   - Show the cumulative density function of P, i.e., $\Pr(P \leq t)$. (Hint: $\Pr(A) = \int_x 1_{[x \in A]} f(x) \mathrm{d}x$ where $1_{[x \in A]}$ is the indicator function and $f(x)$ is the density function of random variable x.) (2pt)
   - Compute the value of the density function of P at $P = 0$ and $P = 0.5$. (1pt)

2. A correct algorithm for uniformly sampling points in $\triangle ABC$ is the following:

   (a) Sample $\alpha \sim U([0,1])$ and $\beta \sim U([0,1])$

   (b) $P' = A + \alpha(B - A) + \beta(C - A)$

   (c) If P' inside $\triangle ABC$, accept by letting $P = P'$. Otherwise, let $P = B + C - P'$.

   Question:

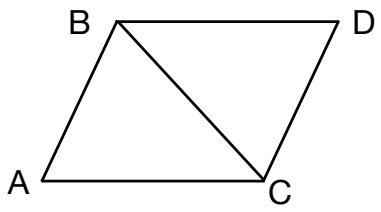


Figure 1: Figure for P2

- Prove that P' is uniformly distributed inside the parallelogram $ABDC$. (Hint: Show the density function *pdf* directly. Do not compute the *cdf*.) Hint 2: Suppose $\mathbf{y} = H(\mathbf{x})$ and H is bijective and differentiable, J is the Jacobian of $\mathbf{y}$ with respective to $\mathbf{x}$, $\mathbf{x}$ has density f, and $\mathbf{y}$ has density g, then $g(\mathbf{y}) = f(H^{-1}(\mathbf{y}))|\det(J^{-1})|$.) (2pt)
- Prove that P is uniformly distributed in $\triangle ABC$. (1pt)

3. Given a triangle whose vertices are at $A = (0, 0)$, $B = (0, 1)$, and $C = (1, 0)$, write a program to simulate the wrong algorithm at the beginning of this problem and the correct algorithm shown above. Make a comparison of them by sampling 1000 points inside the triangle. Include the code and the plot in your submission. (2pt)

   For your convenience, we provide the following codes:

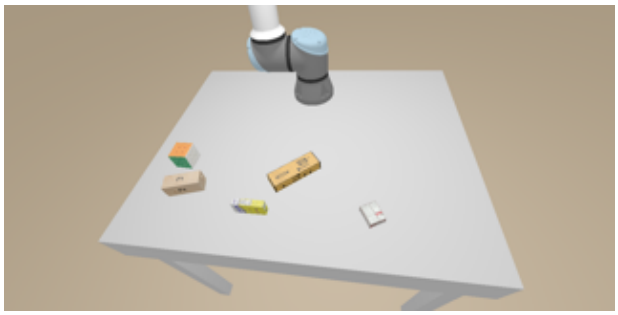
```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

pts = np.array([[0,0], [0,1], [1,0]])
def draw_background(index):
    # DRAW THE TRIANGLE AS BACKGROUND
    p = Polygon(pts, closed=True, facecolor=(1,1,1,0), edgecolor=(0, 0, 0))

    plt.subplot(1, 2, index + 1)

    ax = plt.gca()
    ax.set_aspect('equal')
    ax.add_patch(p)
    ax.set_xlim(-0.1,1.1)
    ax.set_ylim(-0.1,1.1)

# YOUR CODE HERE

draw_background(0)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (incorrect method)
plt.scatter(0.4+0.2*np.random.randn(1000),
            0.4+0.2*np.random.randn(1000), s=3)

draw_background(1)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (correct method)
plt.scatter(0.4+0.2*np.random.randn(1000),
            0.4+0.2*np.random.randn(1000), s=3)
```

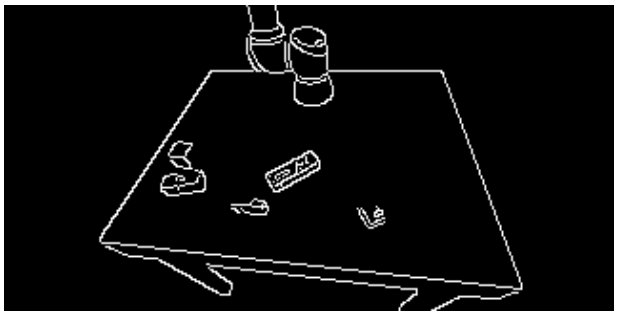
```
plt.show()
```

3 Training a Neural Network (5pt)

For this problem, you will build a neural network edge detector from scratch. We provide the training data **train.npz**. You can load the training data by

```
import numpy as np
npz = np.load("train.npz")
images = npz["images"]  # array with shape (N,Width,Height,3)
edges = npz["edges"]  # array with shape (N,Width,Height)
# images[0] and edges[0] are shown in the figure below
# Please pay attention that the pixel values have range 0-255
```





After training, you need to test your model on 4 testing images provided in **test.npz**.
The network should take images as input and output the edges. You must build the networks from scratch. For example, in PyTorch, you are allowed to use layers such as **torch.nn.Conv2d** and **torch.nn.Sequential**. However, you are not allowed to use packages such as **torchvision**.
In this course, we provide support for the PyTorch framework. You are still allowed to use Tensorflow or other frameworks, however, we will not be able to provide feedbacks other than commenting on your results. For this homework, you only need to plot the results of your edge detector on the 4 test images. You can find setup details in the provided notebook file. Your results will only be graded based on visual appearance, and you do not need to achieve very high accuracy. (5pt)
Hints:

1. At least 2G RAM is required for the dataset and data processing.
2. A UNet (`https://arxiv.org/abs/1505.04597`) can easily solve this problem. You will get full points if you correctly replicate their architecture, but much smaller networks are enough for this problem.
3. The training should take less than 20 minutes on any Nvidia GTX 10XX or RTX 20XX GPU. If you do not have a GPU, you can try Google Colab (Login with your .edu Google account for free GPU resources). You can save your network weights on Google Colab and load them on your local machine for plotting images. If you have to train on CPU, 5 hours should be enough.
4. Note: this problem can be solved by traditional edge detection, but the purpose of this problem is to make sure you know how to design a neural network on your own.