Solution

January 18, 2021

1 Homework 0

1.1 Problem 1

1. Gradient of Lagrangian

$$\nabla_x \mathcal{L} = (Ax - b)^T A + 2\lambda x^T$$

(or as a colume vector)

$$\nabla_x \mathcal{L} = A^T(Ax - b) + 2\lambda x$$

2. Unconstrained lesat square

$$x = A^\dagger b$$

(or for overdetermined system)

$$x = (A^T A)^{-1} A^T b$$

3.a

$$A^T(Ax - b) + 2\lambda x = 0$$
$$(A^T A + 2\lambda I)x = A^T b$$
$$x = (A^T A + 2\lambda I)^{-1} A^T b$$

3.b

Consider eigendecomposition $A^T A = UDU^T$. Since $A^T A$ is PSD, D has non-negative diagonal entires $d_0, ..., d_n$, and U is orthonormal.

$$x = (UDU^T + 2\lambda UU^T)^{-1} A^T b = (U(D + 2\lambda I)U^T)^{-1} A^T b = U(D + 2\lambda I)^{-1} U^T A^T b$$

$$x^T x = b^T AU(D + 2\lambda I)^{-T} U^T U(D + 2\lambda I)^{-1} U^T A^T b = b^T AU(D + 2\lambda I)^{-1}(D + 2\lambda I)^{-1} U^T A^T b$$

Let $k = U^T A^T b$, M be a diagonal matrix with

$$M_{i,i} = (d_i + 2\lambda)^2$$

now

$$x^T x = k^T M k = \sum_i \frac{k_i^2}{(d_i + 2\lambda)^2}$$

Since $d_i \geq 0, \lambda \geq 0$, the denominator of each term increases when λ increases. Since $k_i^2 \geq 0$, $x^T x$ is monotonically decreasing.

4. Implement

[33]:
```
import numpy as np
npz = np.load('HW0_P1.npz')
A = npz['A']
b = npz['b']
eps = npz['eps']
A.shape, A.dtype, b.shape, b.dtype
```

[33]: ((100, 30), dtype('float64'), (100,), dtype('float64'))

[34]:
```
def solve(A, b, eps):
    x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

    # case 1
    if x @ x < eps:
        return x

    # case 2
    d, U = np.linalg.eigh(A.T@A)  # SVD of A may be faster
    k = U.T@(A.T@b)

    def func(lam):
        return ((k / (d + 2 * lam))**2).sum() - eps

    # find a valid pair func(a) > 0, func(b) < 0
    lo = 0
    hi = 1
    while func(hi) > 0:
        lo, hi = hi, hi * 2

    # bisect
    thres = 0.0001
    while True:
        mi = (lo+hi) / 2
        v = func(mi)
        if abs(hi-lo) < thres:
            break
        if v > 0:
            lo = mi
        else:
            hi = mi
```

```
    lam = mi
    x = U@(np.diag(1/(d + 2 * lam))@(U.T@(A.T@b)))
    return x
```

[20]:
```
def solve(A, b, eps):
    x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

    # case 1
    if x @ x < eps:
        return x

    # case 2
    d, U = np.linalg.eigh(A.T@A)  # SVD of A may be faster
    k = U.T@(A.T@b)

    def func(lam):
        return ((k / (d + 2 * lam))**2).sum() - eps

    def dfunc(lam):
        return -4 * ((k**2 / (d+2*lam)**3)).sum()

    # Newton, should converge in less than 10 iterations
    lam = 0
    while True:
        lam2 = lam - func(lam) / dfunc(lam)
        if abs(lam-lam2) < 1e-6:
            break
        lam = lam2
    x = U@(np.diag(1/(d + 2 * lam))@(U.T@(A.T@b)))
    return x
```

[35]:
```
# Evaluation code, you need to run it, but do not modify
x = solve(A,b,eps)
print('x norm square', x@x)
print('optimal value', ((A@x - b)**2).sum())
```

```
x norm square 0.49999768025215285
optimal value 17.220131015463245
```

[29]:
```
print(x)
```

```
[ 0.0979502  -0.12841589  0.0495367   0.06482796  0.04341133  0.06206455
 -0.16418614  0.03840045  0.30916126 -0.1238789   0.06729939 -0.01284795
 -0.03535082 -0.10851547 -0.02132291 -0.12418829  0.18965628 -0.15722834
 -0.17646289  0.04182677  0.09246236  0.11353722 -0.10293015 -0.03047977
  0.03294803 -0.23714231 -0.14864573 -0.07861532  0.15917405 -0.22602551]
```

1.2 Problem 2

(2.1) Your proof here

By definition

$$P = \frac{\beta}{\alpha + \beta}$$

Since

$$\Pr(P < t) = \Pr(\frac{\beta}{\alpha + \beta} < t) = \Pr(\beta < \frac{t}{1-t}\alpha)$$

From picture below, we can easily see

$$F(t) = \Pr(\beta < \frac{t}{1-t}\alpha) = \begin{cases} \frac{t}{2(1-t)} & 0 <= t <= 0.5 \\ \frac{3t-1}{2t} & 0.5 < t <= 1 \end{cases}$$

$$f(0) = F'(0) = 0.5$$
$$f(0.5) = F'(0.5) = 2$$

(2.2) Your proof here

Uniform in parallelogram:

Let $(B - A)$, $(C - A)$ be the columns of matrix T, let random variable $X = [\alpha, \beta]^T$. Denote point A by vector b. Denote points P' by random variable Y.

Then $Y = TX + b$.

By hint 2, given y in the parallelogram,

$$f_Y(y) = f_X(T^{-1}(y - b))|\det(T^{-1})| = f|\det(T^{-1})| = (\text{constant})$$

So P' is uniformly distributed.

Uniform in triangle:

Let the density of Y be g. Let the event P' lands in triangle ABC be E. Let random variable Z represent point P.

$$\begin{aligned} f_Z(z) &= f_Z(z|E)\Pr(E) + f_Z(z|\neg E)\Pr(\neg E) = f_Y(z|E)\Pr(E) + f_Y(B + C - z|\neg E)\Pr(\neg E) \\ &= f_Y(z) + f_Y(B + C - z) = 2g \end{aligned}$$

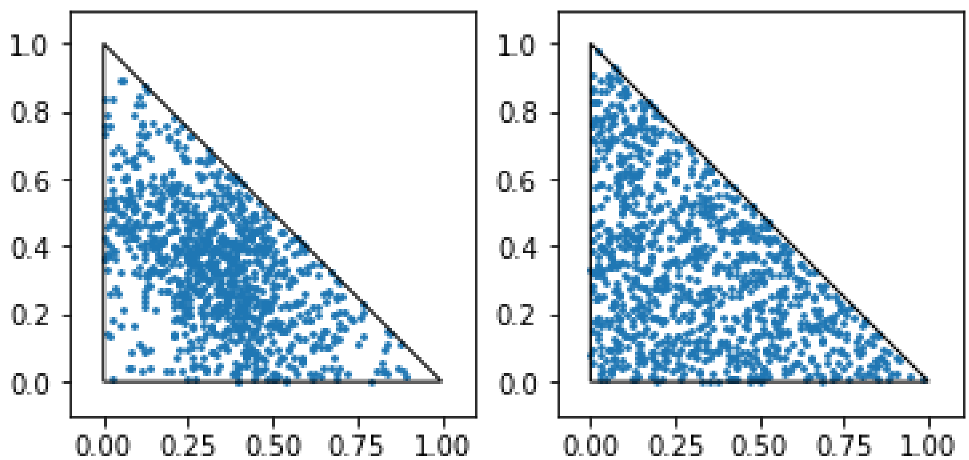
So P is uniformly distributed.

[367]:
```
A = np.random.random((1000, 3))
A /= A.sum(-1)[:, None]
P = (A@pts)
draw_background(0)
plt.scatter(P[:,0], P[:, 1], s=3)
```

```
A = np.random.random((1000, 2))
P = A @ pts[1:]
mask = P.sum(-1) > 1
P[mask] = np.array((1,1)) - P[mask]
draw_background(1)
plt.scatter(P[:,0], P[:, 1], s=3)
```

```
/home/fx/.local/lib/python3.8/site-packages/ipykernel/ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[367]: <matplotlib.collections.PathCollection at 0x7f06647fd3d0>

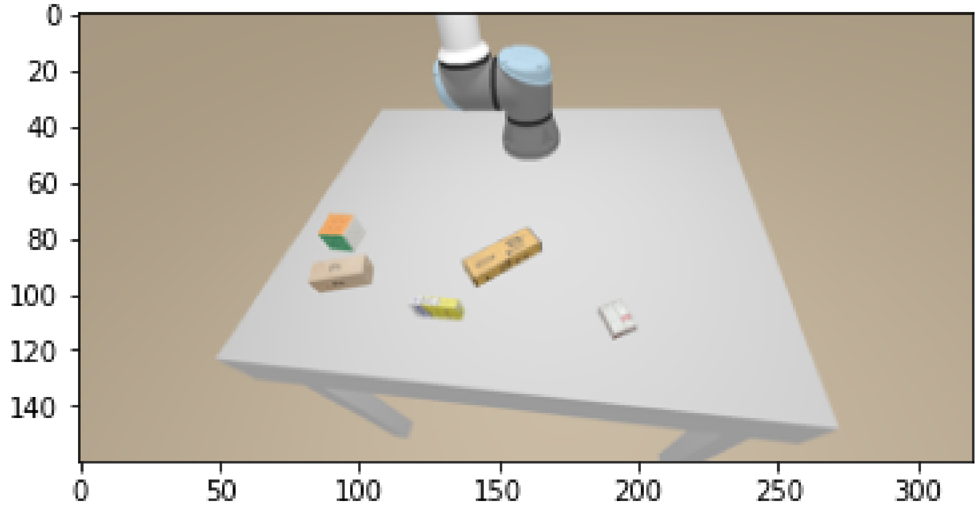


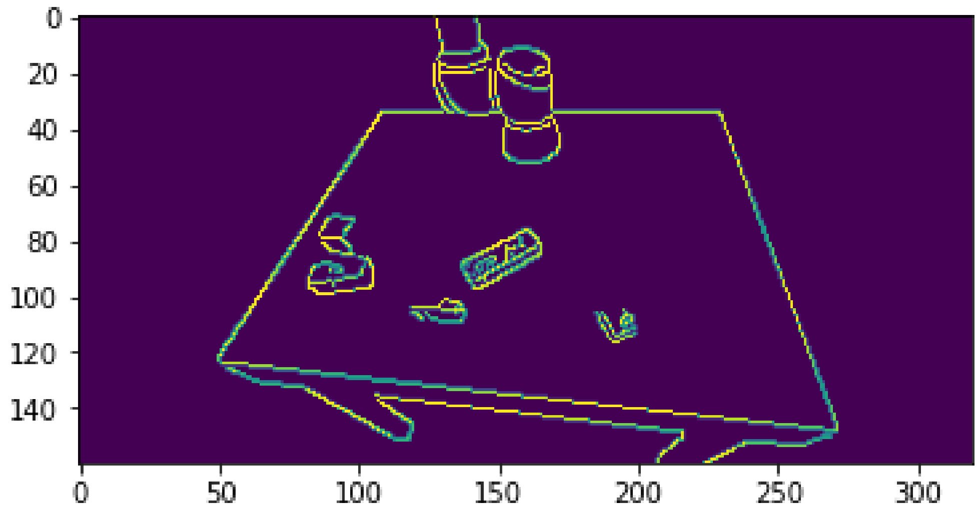
1.3 Problem 3

[368]:
```
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)
```

[369]:
```
plt.figure()
plt.imshow(images[0])
plt.figure()
plt.imshow(edges[0])
```

[369]: <matplotlib.image.AxesImage at 0x7f06ab2ce0d0>





[370]:
```
images.shape, edges.shape, images.max(), edges.max()
```

[370]: ((1000, 160, 320, 3), (1000, 160, 320), 255, 255)

[1]:
```
# Build and train your neural network here, optionally save the weights
```

[ ]:
```
import torch
from torch import nn
import torch.nn.functional as F
import numpy as np
from torch.utils.data import DataLoader
```

```python
class EdgeDetection(nn.Module):
    def __init__(self):
        super().__init__()
        self.c1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(64, 64, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
        )
        self.c2 = nn.Sequential(
            nn.Conv2d(64, 128, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(128, 128, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
        )
        self.c3 = nn.Sequential(
            nn.Conv2d(128, 256, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
        )
        self.c4 = nn.Sequential(
            nn.Conv2d(256, 512, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
        )

        self.p1 = nn.MaxPool2d(2)
        self.p2 = nn.MaxPool2d(2)
        self.p3 = nn.MaxPool2d(2)

        self.d1 = nn.ConvTranspose2d(128, 64, 2, 2)
        self.d2 = nn.ConvTranspose2d(256, 128, 2, 2)
        self.d3 = nn.ConvTranspose2d(512, 256, 2, 2)

        self.dc1 = nn.Sequential(
            nn.Conv2d(128, 64, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(64, 64, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(64, 1, 1),
            nn.Sigmoid(),
        )

        self.dc2 = nn.Sequential(
```

```python
            nn.Conv2d(256, 128, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(128, 128, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
        )

        self.dc3 = nn.Sequential(
            nn.Conv2d(512, 256, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, padding=1, padding_mode="reflect"),
            nn.ReLU(),
        )

    def forward(self, x):
        x1 = self.c1(x)
        x2 = self.c2(self.p1(x1))
        x3 = self.c3(self.p2(x2))
        x4 = self.c4(self.p3(x3))

        y3 = torch.cat([x3, self.d3(x4)], dim=1)
        y2 = torch.cat([x2, self.d2(self.dc3(y3))], dim=1)
        y1 = torch.cat([x1, self.d1(self.dc2(y2))], dim=1)

        output = self.dc1(y1).squeeze(1)

        return output


class Dataset(torch.utils.data.Dataset):
    def __init__(self, file):
        super().__init__()
        npz = np.load(file)
        self.images = npz["images"][:1000]
        self.edges = npz["edges"][:1000]

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img = (
            torch.tensor(self.images[idx] / 255.0, dtype=torch.float32)
            .permute(2, 0, 1)
            .contiguous()
        )
        edge = torch.tensor(self.edges[idx] > 0, dtype=torch.float32).
↪contiguous()
        return {"image": img, "edge": edge}
```

```
batch_size = 10

dataset = Dataset("train.npz")
train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True,␣
 ↪num_workers=2)

print_freq = 20
epochs = 20

model = EdgeDetection()
optim = torch.optim.Adam(model.parameters(), lr=0.0001)
model.cuda()

for epoch in range(epochs):
    print_count = 0
    print_loss = 0
    epoch_step = 0
    for data in train_loader:
        epoch_step += 1
        print_count += 1
        result = model(data["image"].cuda())
        optim.zero_grad()
        loss = F.binary_cross_entropy(result, data["edge"].cuda())
        print_loss += loss.item()
        loss.backward()
        optim.step()

        if print_count % print_freq == 0:
            print(f"[{epoch+1}/{epochs}][{epoch_step}/{len(train_loader)}]")
            print(f"loss: {print_loss / print_freq}")
            print_loss = 0

    # torch.save(model, f"model_{epoch+1}.pth")
```

[9]:
```
def test(model, img):
    model.eval()
    img = img / 255.
    img = torch.tensor(img, dtype=torch.float32).cuda().permute(2,0,1)
    return model(img[None])[0].data.cpu().numpy()
```

[12]:
```
# Test on the testing set
import numpy as np
import matplotlib.pyplot as plt
npz = np.load("test.npz")
test_images = npz["images"]
```

```
plt.figure(figsize=(10, 10))
for i, img in enumerate(test_images[:4]):
    plt.subplot(4, 2, i * 2 + 1)
    plt.imshow(img)

    plt.subplot(4, 2, i * 2 + 2)
    # edge = evaluate your model on the test set, replace the following line
    # edge = np.zeros(img.shape[:2])
    edge = test(model, img)
    plt.imshow(edge)
```