Homework 0

Problem 1

1. Gradient of Lagrangian

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

2. Unconstrained least square

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

3.a

$$\begin{aligned} x &= (A^T A + 2\lambda I_{n \times n})^{-1} A^T b \\ \implies h(\lambda) &= (A^T A + 2\lambda I_{n \times n})^{-1} A^T b \end{aligned}$$

3.b We have $h(\lambda)^T h(\lambda) = b^T A(A^T A + 2\lambda I_{n\times n})^{-2} A^T b$, observe that the term $(A^T A + 2\lambda I_{n\times n})^{-1}$ is symmetric. We can write $h(\lambda)^T h(\lambda) = ||(A^T A + 2\lambda I_{n\times n})^{-1} A^T b)||_2$.

We can observe that the 2 norm of a vector will reduce if each component of the vector is reduced. So we will try to show that component of the vector $(A^T A + 2\lambda I_{n\times n})^{-1} A^T b)$ reduces for increasing $\lambda \geq 0$.

Observe that diagonal elements of $(A^T A + 2\lambda I_{n\times n})$ will increase if $\lambda \geq 0$ is increased, which implies the diagonal elements of $(A^T A + 2\lambda I_{n\times n})^{-1}$ actually decrease. Consequently, components of the vector $(A^T A + 2\lambda I_{n\times n})^{-1} A^T b)$ since each row of the matrix $(A^T A + 2\lambda I_{n\times n})^{-1}$ acts as a set of coefficients for the linear combination of the components of the vector $A^T b$. Hence we observe that since the diagonal elements have reduced, the linear combination from each row have reduced and hence each component of the vector $(A^T A + 2\lambda I_{n\times n})^{-1} A^T b)$. Hence the norm decreases as $\lambda \geq 0$ is increased $\implies h(\lambda)^T h(\lambda)$ is monotonically decreasing

Alternatively, observe that $A^T A$ is a positive semidefinite matrix, because for any vector $z \in R^n$, $z^T (A^T A) z = (Az)^T (Az) = ||Az||_2^2$, hence all eigenvalues of $A^T A$ are greater than or equal to 0. Now if $\lambda / geq 0$, the matrix $2\lambda I_{n\times n}$ consists of all positive values along the diagonal. Hence the eigenvalues of $A^T A + 2\lambda I_{n\times n}$ increases, but then since eigenvalues of the inverse matrix is the recicropal of the matrix, we can conclude that the eigen values of $A^T A + 2\lambda I_{n\times n})^{-1}$decreases, and hence the components of $A^T A + 2\lambda I_{n\times n})^{-1} A^T b)$ decreases which means the norm decreases. Hence we prove that $h(\lambda)^T h(\lambda)$ decreases.

Hence we have $x = h(\lambda) \implies x^T x - \epsilon = h(\lambda)^T h(\lambda) - \epsilon$, hence our aim is now to find a value of $\lambda > 0$ which is a root of $h(\lambda)^T h(\lambda) - \epsilon$ which would also minimise the objective function

4. Implement

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative
%matplotlib inline
```

```
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The text.latex.preview rcparam was deprecated in Matplotlib 3.3 and wi
ll be removed two minor releases later.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The mathtext.fallback_to_cm rcparam was deprecated in Matplotlib 3.3 a
nd will be removed two minor releases later.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle: Support for set
ting the 'mathtext.fallback_to_cm' rcParam is deprecated since 3.3 and
will be removed two minor releases later; use 'mathtext.fallback : 'c
m' instead.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The validate_bool_maybe_none function was deprecated in Matplotlib 3.3
and will be removed two minor releases later.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The savefig.jpeg_quality rcparam was deprecated in Matplotlib 3.3 and
will be removed two minor releases later.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The keymap.all_axes rcparam was deprecated in Matplotlib 3.3 and will
be removed two minor releases later.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The animation.avconv_path rcparam was deprecated in Matplotlib 3.3 and
will be removed two minor releases later.
In /home/sambaran/anaconda3/envs/spinningup/lib/python3.6/site-package
s/matplotlib/mpl-data/stylelib/_classic_test.mplstyle:
The animation.avconv_args rcparam was deprecated in Matplotlib 3.3 and
will be removed two minor releases later.
```

In [3]:
```
#Implementing Newtons method to find value of lambda that is root of h^T

def newton(f, x0, eps = 1e-4, iterations = 500):
    '''
    : Newtons iterative method to find root of a function
    : Newtons update : x -> x - f(x) / f'(x)
    : input --> function, previous estimate, max iterations, threshold t
    : output --> next estimate of x
    '''

    x = x0
    for iter in range(iterations):
        df = derivative(f, x, 1e-10)
        x_next = x - f(x)/df
        if np.linalg.norm(x_next - x) < 1e-9:
            print(f'optimal value at {x_next} found in {iter + 1} iterat
            break
        x = x_next
    return x
```

In [4]:
```
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
```

Out[4]:
```
((100, 30), dtype('float64'), (100,), dtype('float64'))
```

In [5]:
```
def solve(A, b, eps):
    # your implementation here
    h = lambda l : np.linalg.inv(A.T @ A + 2 * l * np.eye(A.shape[1])) @
    f = lambda l : h(l).T @ h(l) - eps #this is the function we want to

    l0 = 0 #starting point value of lambda

    l = newton(f, l0)

    return h(l) #once we find desired lambda value, x = h(lambda)
```

In [6]:
```
# Evaluation code, you need to run it, but do not modify
x = solve(A,b,eps)
print('x norm square', x@x)  # x@x should be close to or less then eps
print('optimal value', ((A@x - b)**2).sum())
```

```
optimal value at 0.837045756847062 found in 5 iterations
x norm square 0.4999999999991626
optimal value 17.22012713194599
```

In [7]:
```
eps
```

Out[7]:
```
array(0.5)
```

Problem 2

(2.1)

A = 0, B = 1 $\alpha, \beta \sim \mathcal{U}[(0,1)]$

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

$P = \alpha' A + \beta' B = \beta' = \frac{\beta}{\alpha+\beta}$

CDF of P

Prob($P \leq t) = Prob(\frac{\beta}{\alpha+\beta} \leq t)$

$= Prob(\beta(1-t) \leq t\alpha) = Prob(\beta \leq \frac{t\alpha}{1-t})$

Now we see that above cdf depends on the value of random variable α, hence to get the CDF of P, we need to marginalize wrt α

$\int_{\gamma=0}^{1} Prob(\beta \leq \frac{t\gamma}{1-t} | \alpha = \gamma) p_\alpha(\alpha = \gamma) d\gamma$ The CDF of $\beta \sim \mathcal{U}[(0,1)]$ is just $x, 0 \leq x \leq 1$ and 0 otherwise, hence we have

$\int_{\gamma=0}^{1} Prob(\beta \leq \frac{t\gamma}{1-t}) d\gamma$

Let $u = \frac{t\gamma}{1-t} \implies du = \frac{t}{1-t}$

$\int_{u=0}^{\frac{t}{1-t}} Prob(\beta \leq u)(\frac{1-t}{t}) du$

Case 1. $\frac{t}{1-t} \leq 1 \implies 0 \leq t \leq \frac{1}{2}$, then CDF of P

= $\frac{1-t}{t} \int_{u=0}^{\frac{t}{1-t}} u du = \frac{1-t}{t} (\frac{t}{1-t})^2 \frac{1}{2} = \frac{t}{2(1-t)}$

Case 2. $\frac{t}{1-t} \geq 1 \implies \frac{1}{2} < t \leq 1$, then CDF of P

$= \frac{1-t}{t} (\int_{u=0}^{1} u du + \int_{u=1}^{\frac{t}{1-t}} 1 du) = \frac{1-t}{t} (\frac{1}{2} + \frac{t}{1-t} - 1) = \frac{3t-1}{2t}$

Therefore, CDF of P is as follows

$$F_P(P \leq t) = \begin{Bmatrix} \frac{t}{2(1-t)} & 0 \leq t \leq \frac{1}{2} \\ \frac{3t-1}{2t} & \frac{1}{2} < t \leq 1 \\ 0 & otherwise \end{Bmatrix}$$

For the pdf values, first of all we observe that the CDF value at $t = 0$ is continuous, since at $t \to 0^-$, $F_P(P) = 0$ but for $t \to 0^+$, $F_P(P) = \frac{1}{2}$. Hence the differential of the CDF does not exist at $t = 0$. SO to get the pdf of P at t = 0, we use

$$f_P(P=0) = \frac{F_P(t \to 0^+)}{t} = \lim_{t \to 0^+} \frac{t}{2(1-t)t} = \frac{1}{2}$$

At t = 0.5, the CDF is left and right continuous. Hence it is differentiable at t = 0.5, and hence pdf of P at t = 0.5 is

$$f_P(t=0.5) = \frac{d}{dt}(\frac{3}{2} - \frac{1}{2t})|_{t=0.5} = 2$$

(2.2)

Correct Sampling algorithm

The correct algorithm is as follows :

1. Sample $\alpha \sim \mathcal{U}[0,1], \beta \sim \mathcal{U}[0,1]$
2. Let A, B, C, D be the vertices of the parallelogram. Let $P' = A + \alpha(B-A) + \beta(C-A)$
3. If P' is inside ΔABC then select $P = P'$, else select $P = B + C - P'$

A. To prove : P' has a uniform distribution inside parallelogram ABDC

Proof :

First of all we will show that for $\alpha \sim \mathcal{U}[0,1], \beta \sim \mathcal{U}[0,1]$, the point generated P' will always lie inside parallelogram ABDC. For proving this, we will consider the side AB || X axis for simplicity. If we consider values of $\alpha = 1$ and $\beta = 1$, then we have :-

$$P'_x = A_x + \alpha(B_x - A_x) + \beta(C_x - A_x)$$
$$\implies P'_x = B_x + C_x - A_x$$

Now under the assumption stated above of side AB || X axis, we can easily observe that $(B_x - A_x) + C_x = D_x$, hence

$$P'_x = D_x$$

.

Now doing the same for the Y component of P'

$$P'_y = B_y + C_y - A_y$$
$$\implies P'_y = B_y = D_y$$

since

$$A_y = C_y$$

Similary we can show for $\alpha, \beta = 0, 0$, generated point $P' = A$.

Hence we show that for the maximum values of α, β the point P' is the point D, and for minimum values of α, β the point P' is A which concludes the fact the the sampling algorithm showed above will always give a point $P' \in ABDC$

Now we try to show that P' follows a uniform distribution inside this region ABDC. We have already concluded from above that any point from inside the parellogram will satisfy $0 \leq \alpha, \beta \leq 1$.

Consider $\mathcal{T} = [\alpha, \beta]$ as a Random Variable. Since α, β are two independent uniform random variables, joint distribution of $\mathcal{T}$ can be described as

$$p_{\mathcal{T}}(\alpha, \beta) = \begin{Bmatrix} 1 & 0 \leq \alpha, \beta \leq 1 \\ 0 & otherwise \end{Bmatrix}$$

Now

$$P'_x = A_x + \alpha(B_x - A_x) + \beta(C_x - A_x)$$
$$P'_y = A_y + \alpha(B_y - A_y) + \beta(C_y - A_y)$$

which are transformations of the random variables $\mathcal{T} = [\alpha, \beta]$. So $P' = [P'_x, P'_y]$ is the transformed random variable and we are interested in its distribution. We can use the change of density formula to get the pdf of P'. The change of density function for $Y = H(X)$ is given as

$$p_Y(Y = y) = \frac{p_X(H^{-1}(x))}{|det(J)|}$$

where $J = \frac{dH}{dX}$ is the Jacobian matrix of H w.r.t X

For our transformation, $P' = \begin{bmatrix} P'_x \\ P'_y \end{bmatrix} = \begin{bmatrix} A_x + \alpha(B_x - A_x) + \beta(C_x - A_x) \\ A_y + \alpha(B_y - A_y) + \beta(C_y - A_y) \end{bmatrix}$,

$$J = \frac{dP'}{d\mathcal{T}} = \begin{bmatrix} \frac{dP'_x}{d\alpha} & \frac{dP'_x}{d\beta} \\ \frac{dP'_y}{d\alpha} & \frac{dP'_y}{d\beta} \end{bmatrix} = \begin{bmatrix} B_x - A_x & C_x - A_x \\ B_y - A_y & C_y - A_y \end{bmatrix}$$

Hence, for P' inside parallelogram ABDC, we get

$$p_{P'}(P'_x, P'_y) = \frac{p_{\mathcal{T}}(\alpha, \beta)}{|det(J)|}$$
$$\implies p_{P'}(P'_x, P'_y) = \frac{1}{|(B_x - A_x)(C_y - A_y) - (B_y - A_y)(C_x - A_x)|}$$

since $p_{\mathcal{T}}(\alpha, \beta) = 1$ inside parallelogram as we already showed that for points inside ABDC, $0 \leq \alpha, \beta \leq 1$.

Hence we observe that the pdf of P' for points inside the parallelogram is just a constant value that depends on the coordinate points of the parallelogram. Hence using the algorithm shown above, P' follows a uniform distribution inside parallelogram ABDC.

To Prove : P = B + C - P' has uniform distribution inside ΔABC

Proof :

From the above sampling process, we get points P' inside the parallelogram ABDC. For points that lie outside ΔABC, we prove that the transformation $P = B + C - P'$ will bring these points back inside ΔABC.

So if P' lies outside the ΔABC implies P' lies above line BC. The equation of the line BC is given by

$$\frac{y - C_y}{x - C_x} = \frac{B_y - C_y}{B_x - C_x}$$
$$\implies y = C_y + \frac{B_y - C_y}{B_x - C_x}(x - C_x)$$

For P' to lie outside the line BC, it should satisfy

$$P'_y > C_y + \frac{B_y - C_y}{B_x - C_x}(P'_x - C_x)$$
$$\implies (B_y - C_y)(P'_x - C_x) < (P'_y - C_y)(B_x - C_x)$$
$$\implies P'_x(B_y - C_y) < (P'_y - C_y)(B_x - C_x) + C_x(B_y - C_y)$$

Now, using the above inequality, we want to prove that $P = B + C - P'$ will result in the point P inside ΔABC i.e. P will lie below the line BC.

To prove this, we need to show that $P_y < f(P_x)$ where $f(x)$ is the equation of line BC.

Putting $P_x = B_x + C_x - P'_x$ into the equation of the line BC, we have

$$f(P_x) = C_y + \frac{B_y - C_y}{B_x - C_x}(B_x + C_x - P'_x - C_x)$$
$$= C_y + \frac{B_y - C_y}{B_x - C_x}(B_x - P'_x)$$
$$= \frac{(B_y - C_y)(B_x - P'_x) + C_y(B_x - C_x)}{B_x - C_x}$$
$$= \frac{B_y B_x - C_y C_x - P'_x(B_y - C_y)}{B_x - C_x}$$

Now using the inequality stated above
$P'_x(B_y - C_y) < (P'_y - C_y)(B_x - C_x) + C_x(B_y - C_y)$, we have

$$f(P_x) > \frac{B_y B_x - C_y C_x - (P'_y - C_y)(B_x - C_x) + C_x(B_y - C_y)}{B_x - C_x}$$

(We are now subtracting by a bigger number, so resulting number will be less than the original)

$$\implies f(P_x) > \frac{B_y B_x - C_y C_x - P'_y(B_x - C_x) + C_y B_x - C_y C_x + C_y C_x - C_x B_y}{B_x - C_x}$$
$$\implies f(P_x) > \frac{B_y(B_x - C_x) + C_y(B_x - C_x) - P'_y(B_x - C_x)}{B_x - C_x}$$
$$\implies f(P_x) > B_y + C_y - P'_y$$
$$\implies f(P_x) > P_y$$

which proves that P_y lies below the line BC, hence P lies inside ΔABC if P' lies outside it.

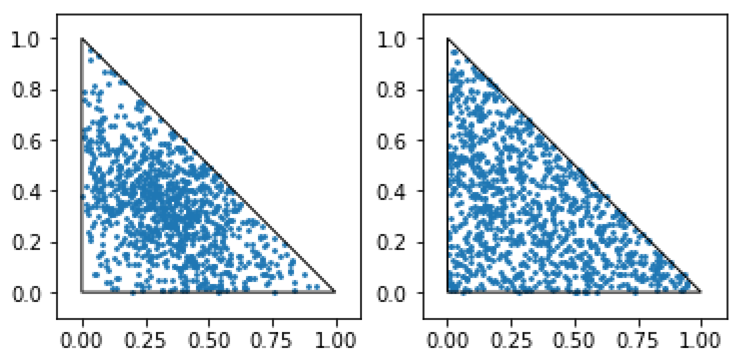
This completes our proof that P will lie inside ΔABC if P' lies outside. And since we already prove that P' has uniform distribution in $ABDC$, we arrive at the conclusion that P will have a uniform distribution in ΔABC.

In [8]:
```
def incorrect(points, n_samples = 1000):
    incorrect_samples = []
    A,B,C = points
    for n in range(n_samples):
        alpha, beta, gamma = np.random.uniform(0,1), np.random.uniform(0
        total = alpha + beta + gamma
        alpha /= total
        beta /= total
        gamma /= total
        incorrect_samples.append(alpha * A + beta * B + gamma * C)
    return np.array(incorrect_samples)
```

In [9]:
```
def area(x1,y1,x2,y2,x3,y3):
    return abs((x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0)

def inside_triangle(points,P):
    x1,y1,x2,y2,x3,y3 = points.flatten()
    x_p,y_p = P
    area_ABC = area(x1,y1,x2,y2,x3,y3)
    area_PAB = area(x1,y1,x_p,y_p,x2,y2)
    area_PAC = area(x_p, y_p, x1, y1, x3,y3)
    area_PBC = area(x_p, y_p, x2, y2, x3, y3)
    return area_ABC == area_PAB + area_PAC + area_PBC
```

In [10]:
```
def correct(points, n_samples = 1000):
    correct_samples = []
    A,B,C = points
    for n in range(n_samples):
        alpha, beta = np.random.uniform(0,1), np.random.uniform(0,1)
        P_prime = A + alpha * (B - A) + beta * (C - A)
        if inside_triangle(points,P_prime):
            correct_samples.append(P_prime)
        else:
            correct_samples.append(B + C - P_prime)
    return np.array(correct_samples)
```

In [11]:
```
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,

    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)
incorrect_samples = incorrect(pts)
plt.scatter(incorrect_samples[:,0], incorrect_samples[:,1], s=3)

draw_background(1)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (correct method)
correct_samples = correct(pts)
plt.scatter(correct_samples[:,0], correct_samples[:,1], s=3)

plt.show()
```

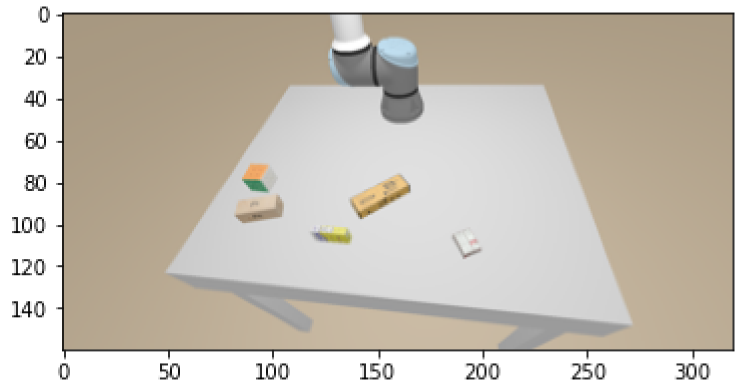


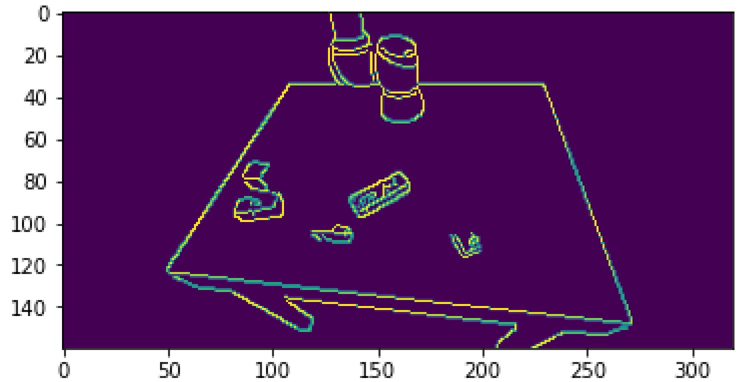
Problem 3

In [12]:
```
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)
```

In [13]:
```
plt.figure()
plt.imshow(images[0])
plt.figure()
plt.imshow(edges[0])
```

Out[13]: <matplotlib.image.AxesImage at 0x7ff26428b780>





In [14]:
```
images.shape, edges.shape, images.max(), np.unique(edges)
```

Out[14]: ((1000, 160, 320, 3), (1000, 160, 320), 255, array([0, 255], dtype=uint8))

In [15]:
```
import torch
```

In [16]:
```
def conv_block(in_channels = 3, out_channels = 64):
    conv1 = torch.nn.Conv2d(in_channels, out_channels, kernel_size = 3,
    conv2 = torch.nn.Conv2d(out_channels, out_channels, kernel_size = 3,
    relu = torch.nn.ReLU(inplace= True)
    conv_down = torch.nn.Sequential(conv1, relu, conv2, relu)
    return conv_down
```

In [17]:
```
#Build the UNet
class UNet(torch.nn.Module):
    def __init__(self, in_channels = 3, out_channels = 1):
        super().__init__()

        #UNet encoder architecture
        self.down_conv1 = conv_block(in_channels = 3, out_channels = 64)
        self.down_conv2 = conv_block(in_channels = 64, out_channels = 12
        self.down_conv3 = conv_block(in_channels = 128, out_channels = 2
        self.down_conv4 = conv_block(in_channels = 256, out_channels = 5
        self.down_conv5 = conv_block(in_channels = 512, out_channels = 1
        self.maxpool = torch.nn.MaxPool2d(kernel_size = 2, stride = 2)

        #UNet decoder architecture
        self.upsample4 = torch.nn.ConvTranspose2d(in_channels = 1024, ou
        self.upsample3 = torch.nn.ConvTranspose2d(in_channels = 512, out
        self.upsample2 = torch.nn.ConvTranspose2d(in_channels = 256, out
        self.upsample1 = torch.nn.ConvTranspose2d(in_channels = 128, out

        self.up_conv4 = conv_block(in_channels = 1024, out_channels = 51
        self.up_conv3 = conv_block(in_channels = 512, out_channels = 256
        self.up_conv2 = conv_block(in_channels = 256, out_channels = 128
        self.up_conv1 = conv_block(in_channels = 128, out_channels = 64)

        #Final 1X1 conv layer
        self.final_conv = torch.nn.Conv2d(in_channels = 64, out_channels

    def forward(self, input):
        x1 = self.down_conv1(input) #Copy + Crop to corresponding upsamp
        y1 = self.maxpool(x1) #Input to next down conv

        # print(x1.shape)
        # print(y1.shape)

        x2 = self.down_conv2(y1) #Copy + Crop to corresponding upsample
        y2 = self.maxpool(x2) #Input to next down conv

        # print(x2.shape)
        # print(y2.shape)

        x3 = self.down_conv3(y2) #Copy + Crop to corresponding upsample
        y3 = self.maxpool(x3) #Input to next down conv

        # print(x3.shape)
        # print(y3.shape)

        x4 = self.down_conv4(y3) #Copy + Crop to corresponding upsample
        y4 = self.maxpool(x4) #Input to next down conv

        # print(x4.shape)
        # print(y4.shape)

        x5 = self.down_conv5(y4) #Copy + Crop to corresponding upsample

        # print(x5.shape)
```

```
        #x5 is output of encoder network
        #Decoder network starts

        out4 = self.upsample4(x5)
        # print(f'out4 shape : {out4.shape}')

        out4 = torch.cat([x4,out4], 1)
        # print(f'out4 shape after concatentation : {out4.shape}')

        out3 = self.upsample3(self.up_conv4(out4))
        # print(f'out3 shape : {out3.shape}')

        out3 = torch.cat([x3, out3], 1)
        # print(f'out3 shape after concatentation : {out3.shape}')

        out2 = self.upsample2(self.up_conv3(out3))
        # print(f'out2 shape : {out2.shape}')

        out2 = torch.cat([x2, out2], 1)
        # print(f'out2 shape after concatentation : {out2.shape}')

        out1 = self.upsample1(self.up_conv2(out2))
        # print(f'out1 shape : {out1.shape}')

        out1 = torch.cat([x1, out1], 1)
        # print(f'out1 shape after concatentation : {out1.shape}')

        out1 = self.up_conv1(out1)
        # print(f'out1 shape after final double conv layer : {out1.shape

        segmentation = self.final_conv(out1)
        # print(f'segmentation output : {segmentation.shape}')



        return segmentation
```

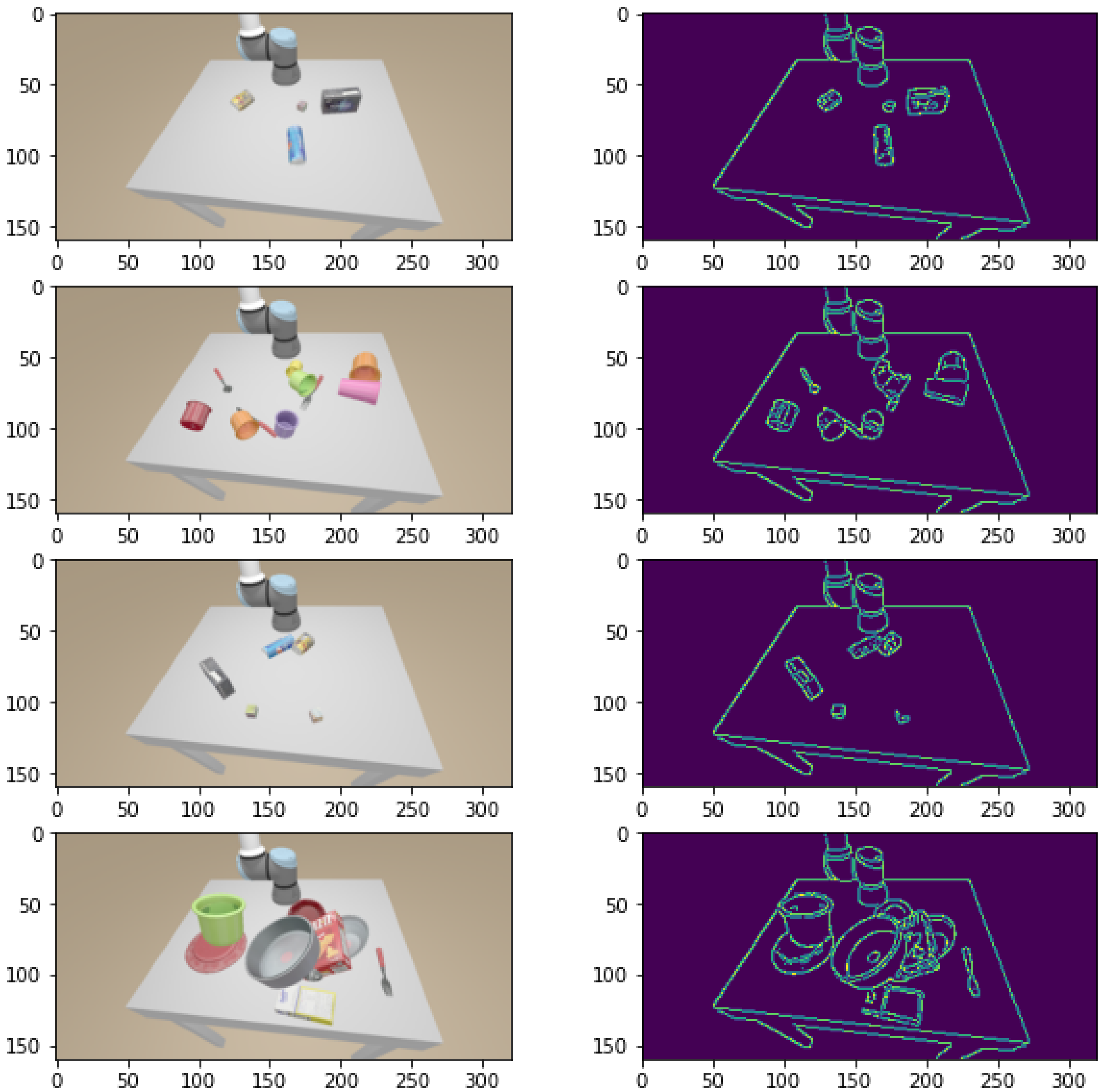
In [18]:

```
torch.__version__
```

Out[18]: '1.10.2'

In [19]:
```
# Test on the testing set
model = UNet()
checkpoint = torch.load('/home/sambaran/UCSD/CSE291/HW0/HW0/checkpoint-U
device = ('cuda' if torch.cuda.is_available() else 'cpu')
npz = np.load("test.npz")
test_images = npz["images"]

model.load_state_dict(checkpoint['model_state_dict'])
model.to(device)
model.eval()
plt.figure(figsize=(10, 10))
with torch.no_grad():
    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 follow
        img = torch.tensor(img/255, dtype = torch.float32)
        img = img.transpose(0,2).transpose(1,2).to(device).unsqueeze(0)
        print(img.max())
        edge = torch.sigmoid(model(img))
        edge = (edge > 0.5).float()

        plt.imshow(edge.detach().cpu()[0,0])
```

```
tensor(1., device='cuda:0')
tensor(1., device='cuda:0')
tensor(1., device='cuda:0')
tensor(1., device='cuda:0')
```



In [ ]: