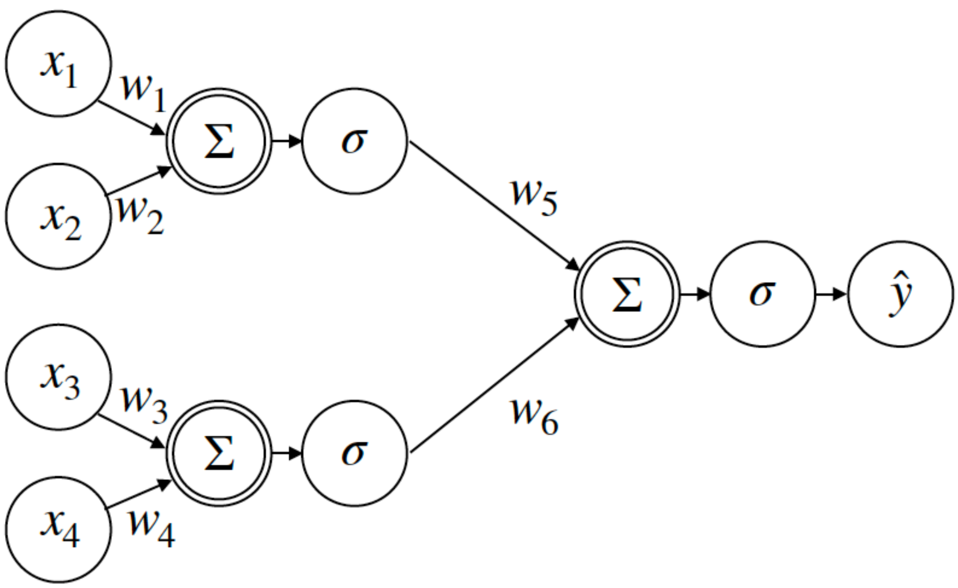
Xiang Chen
1004704992
November 29, 2020

CSC420 Assignment 4

Q1) Deep Learning (30 marks)



From the question, we know $\sigma(x) = \dfrac{1}{1+e^{-x}}$, $L(y, \hat{y}) = ||y - \hat{y}||_2^2$.

Based on the chain rule, we will use backward propagation.

The last level, we can get $\dfrac{\partial \mathscr{L}}{\partial \hat{y}} = 2||y - \hat{y}||$.

The second last level which is from $\hat{y}$ to σ_2, we can get 1.

The third last level which is from σ_2 to $\sum_2$, we can get $\dfrac{\partial \sigma_2}{\partial \sum_2} = \dfrac{e^{\Sigma_2}}{(e^{\Sigma_2}+1)^2}$.

The third last level which is from $\sum_2$ to lower level σ_1, we can get $\dfrac{\partial \sum_2}{\partial \sigma_1} = w_6 = -0.2$.

The fourth last level from σ_1 to level $\sum_1$, we can get $\frac{\partial \sigma_1}{\partial \sum_1} = \frac{e^{w_3x_3+w_4x_4}}{(e^{w_3x_3+w_4x_4}+1)^2}$

The first level which is x_3.

Therefore, we can have the following.

$$\frac{\partial \mathscr{L}}{\partial w_3} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_2} \cdot \frac{\partial \sigma_2}{\partial \sum_2} \cdot \frac{\partial \sum_2}{\partial \sigma_1} \cdot \frac{\partial \sigma_1}{\partial \sum_1} \cdot \frac{\partial \sum_1}{\partial x_3}$$

$$= -0.3 \cdot \frac{e^{w_3x_3+w_4x_4}}{(e^{w_3x_3+w_4x_4}+1)^2} \cdot -0.2 \cdot \frac{e^{\sum_2}}{(e^{\sum_2}+1)^2} \cdot 1 \cdot 2||y-\hat{y}|| \quad (1.1)$$

$\sum_0 = x_1w_1 + x_2w_2 = 0.675 + 0.693 = 1.368$

$\sum_1 = x_3w_3 + x_4w_4 = -0.072 - 1.36 = -1.432$

$\sigma_0 = \frac{1}{1+e^{-1.368}} = 0.7970568313$

$\sigma_1 = \frac{1}{1+e^{1.432}} = 0.1927872523$

$\sum_2 = 0.637645464 - 0.03855745 = 0.599088014$

$\sigma_2 = \hat{y} = \frac{1}{1+e^{-0.599088014}} = 0.6454476305$

$2||y-\hat{y}|| = 2 \cdot (0.6454476305 - 0.5) = 0.290895261$

We put what we have into (1.1), and we can get the following.

$$-0.3 \cdot \frac{e^{w_3x_3+w_4x_4}}{(e^{w_3x_3+w_4x_4}+1)^2} \cdot -0.2 \cdot \frac{e^{\sum_2}}{(e^{\sum_2}+1)^2} \cdot 1 \cdot 2||y-\hat{y}||$$

$\rightarrow 0.3 * 0.1556203277 * 0.2 * 0.2288449868 * 0.290895261$

$\rightarrow 6.21577986 * 10^{-4}$

Therefore, by using backward propagation we have the following

$$\frac{\partial \mathscr{L}}{\partial w_3} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_2} \cdot \frac{\partial \sigma_2}{\partial \sum_2} \cdot \frac{\partial \sum_2}{\partial \sigma_1} \cdot \frac{\partial \sigma_1}{\partial \sum_1} \cdot \frac{\partial \sum_1}{\partial x_3} = 6.21577986 * 10^{-4}$$

Q2) Camera Models (70 marks)

1. (30 marks)

This question is really similar to what we have proved in the class.

In the class we know the following formula.

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = KX = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x + tD_x \\ V_y + tD_y \\ V_z + tD_z \end{bmatrix} = \begin{bmatrix} fV_x + ftD_x + p_x V_z + tp_x D_z \\ fV_y + ftD_y + p_y V_z + tp_y D_z \\ V_z + tD_z \end{bmatrix}$$

$$x = \lim_{t\to\infty} \frac{fV_x + ftD_x + p_x V_z + tp_x D_z}{V_z + tD_z} = \frac{fD_x + p_x D_z}{D_z}$$

$$y = \lim_{t\to\infty} \frac{fV_y + ftD_y + p_y V_z + tp_y D_z}{V_z + tD_z} = \frac{fD_y + p_y D_z}{D_z}$$

In this question, we just need to follow this up.

From the question, we can have the following.

$$\begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_0 + td_x \\ Y_o + td_y \\ Z_o + td_z \end{pmatrix} = \begin{pmatrix} fX_0 + ftd_x + p_x Z_0 + p_x t d_z \\ fY_0 + ftd_y + p_y Z_0 + p_y t d_z \\ Z_o + td_z \end{pmatrix}$$

We take the limit as above, and we can get the following.

$x = \dfrac{fd_x + p_x d_z}{d_z}$ and $y = \dfrac{fd_y + p_y d_z}{d_z}$.

Therefore, we have the vanishing point which is $(\dfrac{fd_x + p_x d_z}{d_z}, \dfrac{fd_y + p_y d_z}{d_z})$.

2. (40 marks)

From the part a, we know all lines with the same direction will converge at some point, which is $(\frac{fd_x + p_x d_z}{d_z}, \frac{fd_y + p_y d_z}{d_z})$. We can see it does not related to specific position of the line. For all lines with same directions will converge at the same point.

Let assume $\overrightarrow{n} = < n_x, n_y, n_z >$. Since we only need to consider about the lines laying on the plane, we could have the following.

$$n_x d_x + n_y d_y + n_z d_z = 0 \ (2.2.1)$$

We can get the following from (2.2.1).

$$d_x = -\frac{n_y d_y + n_z d_z}{n_x} \ (2.2.2)$$

We put (2.2.2) into the vanishing point.

We could have the following.

$$V_x = \frac{fd_x + p_x d_z}{d_z}$$

$$\rightarrow -f \cdot \frac{n_y \cdot d_y + n_z \cdot d_z}{n_x \cdot d_z} + p_X$$

$$\rightarrow -\frac{f \cdot n_y \cdot d_y + f \cdot n_z \cdot d_z}{n_x \cdot d_z} + p_x$$

$$\rightarrow -\frac{f \cdot n_y \cdot d_y}{n_x \cdot d_z} - \frac{f \cdot n_z \cdot d_z}{n_x \cdot d_z} + p_x$$

$$\rightarrow -\frac{f \cdot n_y \cdot d_y}{n_x \cdot d_z} - \frac{f \cdot n_z}{n_x} + p_x$$

$$\rightarrow -\frac{n_y}{n_x} \cdot \frac{f \cdot d_y}{d_z} - \frac{f \cdot n_z}{n_x} + p_x$$

$$\rightarrow -\frac{n_y}{n_x} \cdot (V_y - p_y) - \frac{f \cdot n_z}{n_x} + p_x$$

$$V_x = -\frac{n_y}{n_x} \cdot (V_y - p_y) - \frac{f \cdot n_z}{n_x} + p_x \ \ (2.2.3)$$

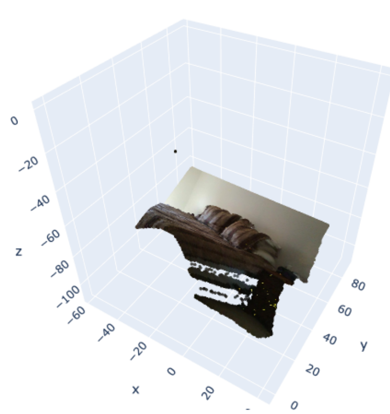
From (2.2.3), we can see V_x and V_y actually have the linear relationship since $f,\ n_x,\ n_y,\ p_x,\ and\ \ p_y$ are all constant and given us.

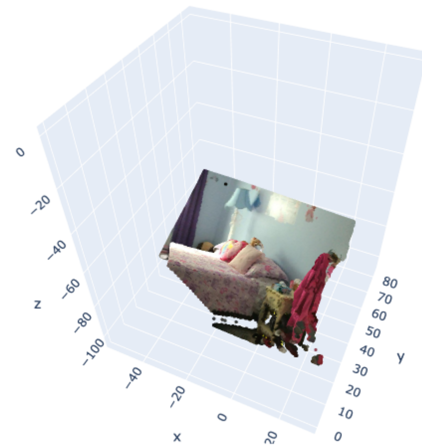
Q3) Projection (50 marks + 10 bonus marks)

Here is the code for this question.

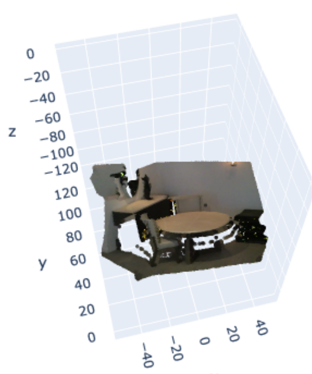
```python
def compute_point_cloud(imageNumber):
    '''
     This function provides the coordinates of the associated 3D scene point
     (X; Y;Z) and the associated color channel values for any pixel in the
     depth image. You should save your output in the output_file in the
     format of a N x 6 matrix where N is the number of 3D points with 3
     coordinates and 3 color channel values:
     X_1,Y_1,Z_1,R_1,G_1,B_1
     X_2,Y_2,Z_2,R_2,G_2,B_2
     X_3,Y_3,Z_3,R_3,G_3,B_3
     X_4,Y_4,Z_4,R_4,G_4,B_4
     X_5,Y_5,Z_5,R_5,G_5,B_5
     X_6,Y_6,Z_6,R_6,G_6,B_6
     .
     .
     .
     .
    '''
    depth, rgb, extrinsics, intrinsics = get_data(imageNumber)
    # rotation matrix
    R = extrinsics[:, :3]
    # t
    t = extrinsics[:, 3]
    width, height = depth.shape
    inverse_intrinsics = np.linalg.inv(intrinsics)
    inverse_R = np.linalg.inv(R)
    res = np.zeros((width*height,6), dtype="float")
    for i in range(width):
        for j in range(height):
            homo = np.array([depth[i][j]*j, depth[i][j]*i, depth[i][j]])
            cam = np.dot(inverse_intrinsics, homo)
            world = np.dot(inverse_R, cam) - t
            # Need to put - before y coordinate to achieve same with what have shown in demo
            res[i*height+j, 0], res[i*height+j, 1], res[i*height+j, 2] = world[0], -world[1], world[2]
            res[i*height+j, 3], res[i*height+j, 4], res[i*height+j, 5] = rgb[i][j][0], rgb[i][j][1], rgb[i][j][2]
    return res
```

Here are the output and we can see they are exactly the same with what have shown in the lecture.





0
−20
−40
−60
−80
−100
z
−40
−20
0
20
x
80
70
60
50
40
30
20
10
0
y



0
−20
−40
−60
−80
−100
−120
z
120
100
80
60
40
20
0
y
−40
−20
0
20
40
x