Ex3)

linear separation with the least square principle:

$x^{(n)} \Rightarrow$ training vector , $y(n)$ class label

$\hat{y} = W^T X \Rightarrow$ linear classifer , $w_0 = 0$

* the cost/loss function

$$E(w) = \sum_{n=1}^{N} (y^{(n)} - W^T x^{(n)})^2 \quad \text{---- (1)}$$

$$E(w) = (y - W^T x)(y - W^T x)^T \quad \text{---- (2)}$$

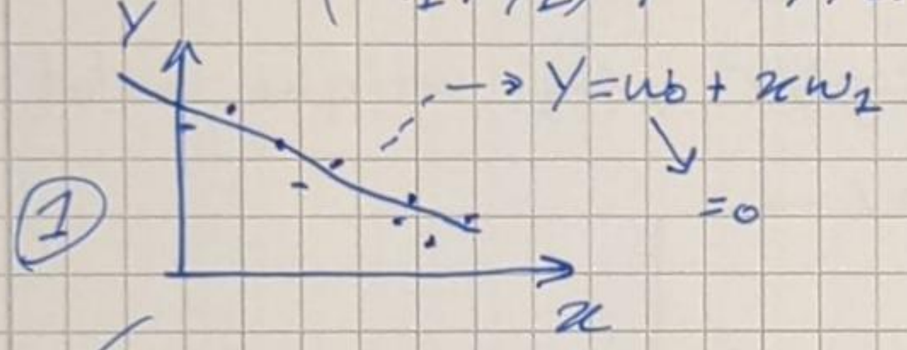
(a) The $E(w)$ function minimum is always exists but not unique.

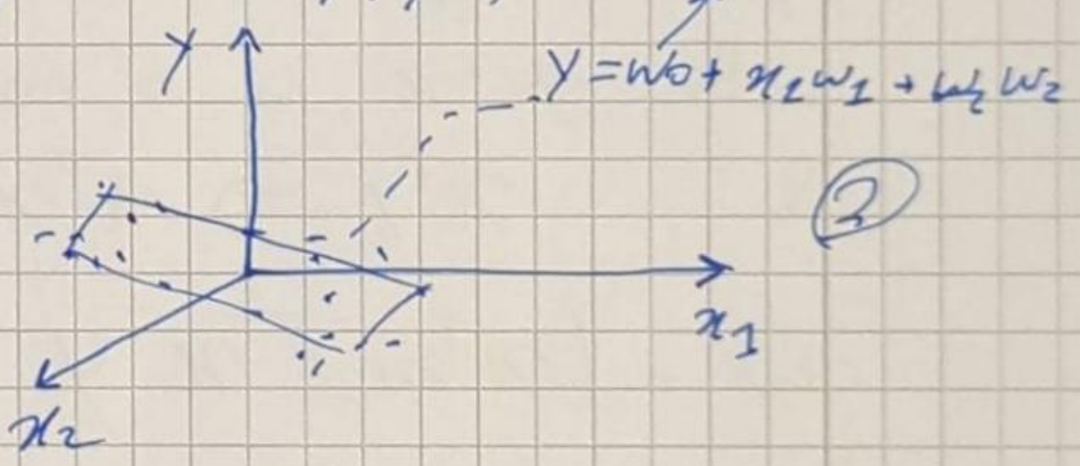
The training vectors represent independent random draws.

Therefore, the X represents a $N \times p$ matrix with each row an input vector and y is an N-vector of the outputs in the training set.

The input/output of observation pairs:

$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)$





the point in (1) lies

is (x_p, y_p) of the slope w_1 as in the two dimensions as $w_0 + x w_1 = y$ is close to $w_0 + x_p w_1 = y$

the point in (2) a data of three dimensions is defind as $w_0 + x_1 w_1 + x_2 w_2 + \dots + x_N w_N = y$

which holds to a hyperplane coordinate slop of $N = 2$

By presenting the linear models and least squares and giving a X^T as an input vector $X^T = (X_1, X_2, \dots X_p)$ the summation would be used as:

$$\hat{y} = \hat{w}_0 + \sum_{n=1}^{N} X^{(n)} \hat{w}$$

Using the $(u-v)^T = u^T - v^T$

$$E(w) = (y - w^T x)(y - w^T x)^T \qquad x \in \mathbb{R}$$

$$\Rightarrow E(w) = (y - w^T x)(y^T - w x^T) \qquad x = x^T$$

$$\Rightarrow yy^T - y^T w x - y w x^T + w^T w x x^T \Longrightarrow y^T x w = (y^T x w)^T$$

$$\Rightarrow yy^T - 2w^T x^T y + w^T w x x^T \Rightarrow y w^T x^T$$

$$\Rightarrow y w x^T$$

differentiate with respect to $\underline{w}$

$$\frac{\partial}{\partial w} E(w) = \frac{\partial}{\partial w} yy^T - 2\frac{\partial}{\partial w} w^T x^T y + \frac{\partial}{\partial w} w^T x^T x w$$

$$= 0 - 2x^T y + 2x^T x w$$

(b) By differentiating with respet to $\underline{w}$, to get the normal equation by setting the derivative equal to 0

$$-2x^T y + 2x^T x w = 0$$

$$x^T y - x^T x w = 0 \Longrightarrow x^T (y - xw) = 0$$

By assuming that the x is a full colounm rank with $x^T x$ is a nonsigular, the unique solution is $x^T y \Rightarrow x^T x w$

$$\hat{w} = (x^T x)^{-1} x^T y$$

Ex3_Timo

May 10, 2022

1 Exercise 3

[ ]:
```
# Done by Munther Odeh and Timo Marks
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image
from skimage.color import rgb2gray
from skimage.filters import gabor_kernel
from skimage.util import img_as_float
from scipy import ndimage as ndi
plt.rcParams.update({'font.size': 16, 'legend.title_fontsize': 16, 'legend.
 ↪fontsize': 16, "axes.labelsize": 16, "axes.labelpad": 4})
```

2 Image Loading

[ ]:
```
image_filename = ["python-hero.jpg", "python-code.jpg", "sun-set.jpg",␣
 ↪"sun-set-rotated.jpg"]
image_colour = []
image_gray = []
shrink = (slice(0, None, 3), slice(0, None, 3))

for img_file in image_filename:
    img = plt.imread("Images/"+img_file)
    img = img_as_float(img)[shrink]
    image_colour.append(img)
    image_gray.append(rgb2gray(img))
    gray_img = rgb2gray(img)
```

[ ]:
```
# Does the actual computation
def power(image, kernel):
    # Normalize images for better comparison.
    image = (image - image.mean()) / image.std()
    return np.sqrt(ndi.convolve(image, np.real(kernel), mode='wrap')**2 +
                   ndi.convolve(image, np.imag(kernel), mode='wrap')**2)

results = []
```

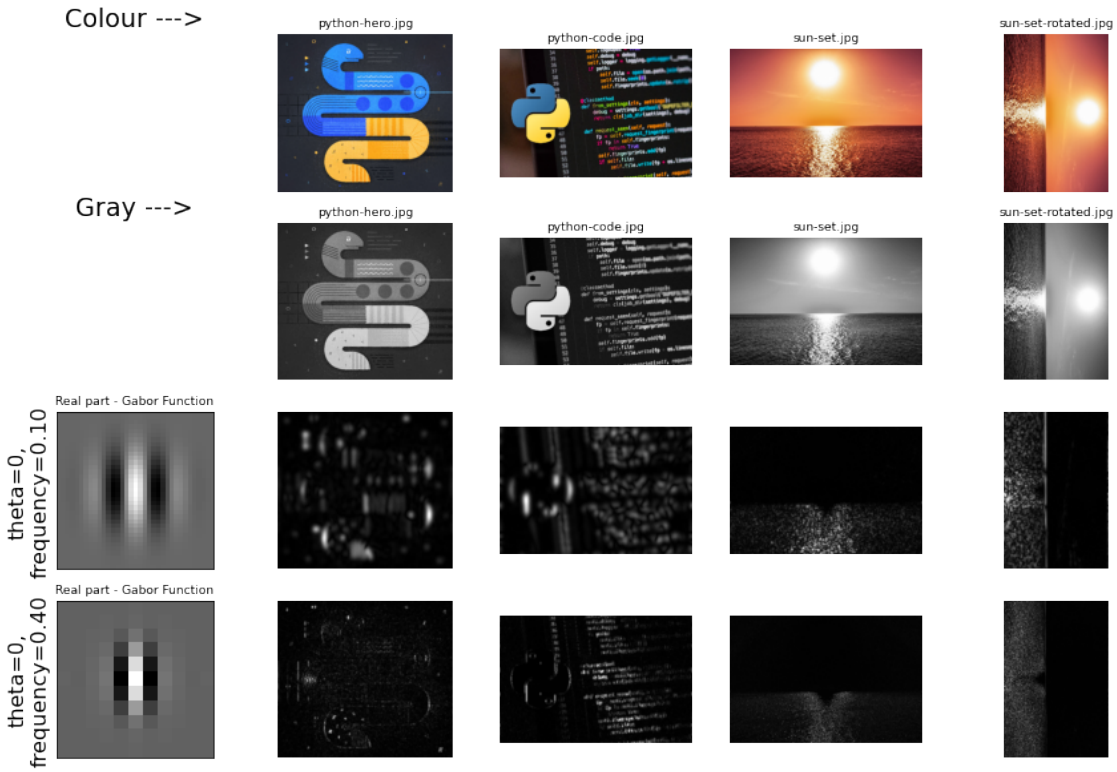
```
kernel_params = []
for theta in [0]:
    theta = theta / 4. * np.pi
    for frequency in [0.1, 0.4]:
        kernel = gabor_kernel(frequency, theta=theta)
        params = 'theta=%d,\nfrequency=%.2f' % (theta * 180 / np.pi, frequency)
        kernel_params.append(params)
        # Save kernel and the power image for each image
        results.append((kernel, [power(img, kernel) for img in image_gray]))

fig, axes = plt.subplots(nrows=len(kernel_params)+2,␣
 ↪ncols=len(image_filename)+1, figsize=(15, 10))
# Plot original images
axes[0][0].axis('off')
axes[0][0].set_title("Colour --->")
for label, img, ax in zip(image_filename, image_colour, axes[0][1:]):
    ax.imshow(img)
    ax.set_title(label, fontsize=9)
    ax.axis('off')

# Plot gray image
axes[1][0].axis('off')
axes[1][0].set_title("Gray --->")
# From now on plot every picture in gray values
plt.gray()
for label, img, ax in zip(image_filename, image_gray, axes[1][1:]):
    ax.imshow(img)
    ax.set_title(label, fontsize=9)
    ax.axis('off')

# Plot Gabor Function
for label, (gabor, powers), ax_rows in zip(kernel_params, results, axes[2:]):
    ax = ax_rows[0] # First column
    ax.imshow(np.real(gabor))
    ax.set_ylabel(label)
    ax.set_title("Real part - Gabor Function", fontsize=9)
    ax.set_xticks([])
    ax.set_yticks([])

    # Plot Gabor
    for gabor_result, ax in zip(powers, ax_rows[1:]):
        ax.imshow(gabor_result)
        ax.axis('off')
```



The receptive field can be described for example with the Gabor function. With these two used Gabor function you can detect vertical borders in the image. Blurry areas are not detected. See for example the sun in the two sun set images. The sun set images also show, that you can detect the horizon if you rotate the image by 90 degrees. In the correct oriented image the horizon is not detected, because it is horizontal and not vertical. You could also detect the horizon if you rotate the Gabor function by 90 degrees.

The Gabor function with the higher frequency creates a sharper output image. The "Code" can be seen better and the ripples are finer. However we get more noise "inside" of the snake in the "python-hero.jpg".

In conclusion, the Gabor function can be used to detect borders of any angle in the image, if you rotate the image or rotate the Gabor function, which would be the more practical approach.