Probabilistics Models of Visual Cortex

Homework 2

Feitong Yang

Department of Psychological and Brain Sciences

Question 1

1) Bayes Decision Theory

Bayes Decision Theory framework: Given a noisy input $x \in X$, we want to make a decision, for example a binary categorization, $y \in Y$ under the uncertainty. The decision is made to minimize the overall risks.

In visual research, we usually have to make inference from the retina image I to the real world stimulus S. In this case, $i \in I$ is the noisy input of the decision making process, and we want to decide what stimulus, $s \in S$, (in a revserse engineering way), result in i on the retina. Therefore, $i \in I$ corresponds to $x \in X$ in the theoretical model, whereas $s \in S$ corresponds to $y \in Y$ in the model.

We have to take several factors into account in making this decision. For example, 1) the knowledge of the distribution of y regardless of observations of x; 2) given the assumption that y is the generative source, what is the probability of $x \in X$ to be an observation; 3) What is the cost of making different decisions when a ground truth may or may not be the decision that is being made. These three factors are prior information, likelihood, and cost of decision, respectively.

The Bayes Decision Theory is to make an decision that take all these three factors into account, and make a decision out of the input such that the overall cost of the decision is minimized.

prior: prior distribution refers to the distribution of the output, or generative source, that is $y \in Y$ in the model, regardless of what the observations/inputs are. This distribution describes our general knowledge before observation.

likelihood functions: likelihood functions is the function of input $x \in X$ given $y \in Y$, $p(x|y)$. That is, given the generative source, what is the probability / likelihood of having x as an observation

loss functions: loss function depends on the decision rule $\alpha(x)$ and the ground truth decision y^*. A decision

rule $\alpha(x)$ is a function (or a rule) that map the input x onto a decision $y \in Y$. Sometimes, a wrong decision can be very expensive, for example, when the radar system treat a bird as a plane and thus activtes the whole defense system. Othertimes, a wrong decision does not hurt. The loss functions is to describe such cost of a decision that is being made when a true decision y^* that should have been made.

special forms:

1) maximum a posterior estimation: find an estimator that maximize the posterior distribution.

When the loss function penalizes all errors by the same amount, that is, if $\alpha(x) \neq y$ then $L(\alpha(x), y) = K_1$; if $\alpha(x) = y$ then $L(\alpha(x), y) = K_2$; and $K_1 > K_2$, then the Bayes rule corresponds to the maximum a posterior estimator $\alpha(x) = \text{argmax}\ p(y|x)$ that is, to find the estimator that achieves the maximum of the posterior distribution.

2) maximum likelihood estiamtion: find an estimator that maximize the likelihood function.

If, in addition to the assumption of the loss function in 1), all the prior probability for $y \in Y$ are the same, e.g. in the binary decision case, $p(y = 1) = p(y = -1)$, we know that $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$, and all $p(y)$ is the same, so the Bayes rule reduce to maximum likelihood estimator $\alpha(x) = \text{argmax}\ p(x|y)$.

2) Signal Detection Theory

false positive: when there is no effect, but the decision, by the decision rule, claims that there is an effect. This claim makes a *false positive* error. For example, when a patient is not affected by cold, but the doctor said that the patient has cold. The doctor is making a false positive error.

false negative: when there is an effect, but the decision, by the decision rule, claims that there is no effects. This claim makes a *false negative* error. For example, when a patient is actually affected by cold, but the doctor said that the patient is healthy. The doctor is making a false negative error.

Now, we assume that the target distribution $X_T \sim N(\mu_T, \sigma^2)$ and the distractor distribution $X_D \sim N(\mu_D, \sigma^2)$, and we also assume that $\mu_T > \mu_D$. The decision $y = 1$ means a decision that regards x as a target, whereas $y = -1$ means a decision that regards x as a distractor.

Given a decision threshold t, the probability of false positive:

$$p(\text{False Positive}) = \int_t^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu_D)^2}{2\sigma^2})$$

and the probability of false negative:

$$p(\text{False Negative}) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu_T)^2}{2\sigma^2})$$

The higher the threshold t, the lower the false positive probability, the higher the false negative probability. In contrast, the lower the threshold t, the lower the false negative probability, the hight ther false positive probability.

Because the Bayes risk is $R(\alpha) = \sum_x p(x) \sum_y L(\alpha(x), y) p(Y|x)$, we define

1. The cost of true positive $L(a(x) = 1, y = 1) = T_p$
2. The cost of true negative $L(a(x) = -1, y = -1) = T_n$
3. The cost of false positive $L(a(x) = 1, y = -1) = F_p$
4. The cost of false negative $L(a(x) = -1, y = 1) = F_n$

The decision rule reduce to

$$\log \frac{p(x|y=1)}{p(x|y=-1)} > \log \frac{T_n - F_p}{T_p - F_n} + \log \frac{p(y=-1)}{p(y=1)}$$

That is, we denote the threshold $t = \log \frac{T_n - F_p}{T_p - F_n} + \log \frac{p(y=-1)}{p(y=1)}$

3) Bayes Decision Theory for edge detection

To use Bayes Decision Theory for edge detection, we assume we have a ground truth or a benchmarked data so that for each pixel we have the intensity of image as well as a lable $y \in \{-1, 1\}$ to denote whether this pixel is a part of an edge ($y = 1$) or not ($y = -1$). Using this ground truth and edge detection algorithms, such as the first derivative filters, we can calculate the likelihood functions of whether a pixel is a part of an edge. Given such a ground truth as prior knowledge, we can also define a loss function. As for the prior, we know statistically most image patches do not contain edges. In this way, we have the three parts of the Bayes decision framework, and we can have a balanced decision rule and a treshold for the edge decision, which is discussed below, after the next paragraph.

First order derivative filter can detect the large intensity differences between neighboring pixels, which is likely to be an edge. The second order derivative filter tries to find the zero-crossing to detect an edge. The second order derivatives scales down the intensity range and pays attention to the zero-crossing. It may not work well of a curved edge, nor does it localize an edge as good as first order derivative filters. The first order derivative filters can easily detect edges of various orientations as well as curved edges.

Given the hierarchical nature of visual processing, and the difficulty of edge detection, the penality of false negative, which is missing an real edge, should be much large. Whereas false positive, which is detecting something that is not actually an edge, should have small penality because higher processing may be able to

ignore some details. That is, we'd rather not miss an edge, and It's OK to have more false positive.

Question 2

1) Fourier Transformation

using the functional analysis we can express an image as a weighted sum of basis functions:

$$I(x) = \sum_i \alpha_i b_i(x)$$

where $b_i(x)$ are basis functions and the α_i are coefficients. Importantly, when the basis are orthogonal, we can solve the coefficients α_i as

$$\alpha_i = \sum_x I(x) b_i(x)$$

In a special case, we can choose a set of sinusoid functions as basis functions in the above functional anaylsis, and we have the Fourier analysis. The calculation of α_i is called the Fourier transform. Since α_i is the coefficients for a basis sinusoid functions of frequency ω, we use $\hat{I}(\omega)$ to denote coefficients.

The **Fourier transform formula** states the calculation of the coefficients

$$\hat{I}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} I(x) \exp\left(i\omega \cdot x\right) dx$$

Inverse Fourier transform formula states the calculation of using the coefficients to construct the images

$$I(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{I}(\omega) \exp\left(i\omega \cdot x\right) d\omega$$

Here $\exp\left(i\omega \cdot x\right) = \cos(\omega \cdot x) + i \sin(\omega \cdot x)$

The sinusoid functions are orhogonal basis functions, and such property ensures the existence of a simple expression for the coefficients.

2) Sparse Coding

The idea of sparse coding is to seek a representation in terms of an over-complete set of basis functions, which include sinusoid functions, impulse functions, and so on, and a criterion which selects an efficient representation so that only a small number of basis functions are activated for each image.

The sparse coding algorithm does not assume the shape of basis functions. Instead, it goes with basic logic of

functional analysis, with an unique energe function that take the *sparsity* into account, that it, it penalize the absolute value of coeffcients. The formula is as follows:

$$E(\alpha) = \sum_{x}(I(x) - \sum_{i=1}^{N} \alpha_i b_i(x))^2 + \lambda \sum_{i=1}^{N} |\alpha_i|$$

The basis functions trained by sparse coding algorithms include a set of different basis functions, such as sinusoid functions, impulse funtions, or gabor-like functions, whereas Fourier transformation only results in sinusoid basis functions. The sinusoid functions handles the smooth intensity changes very well, but provides a very bad results of impulse signals, such as an isolated bright spot. The impulse functions can handle such case much better. So the sparse coding basis functions can use both kinds of basis and result in the minimal number of activations of basis functions to express the same image that requires a large amount of Fourier transformation basis functions. This is how these two algorithms differ.

The λ coefficient controls the degree of sparcity. We can easily see that if $\lambda = 0$, then the formula reduce to a simple quadratic cost function where no sparsity is taken into account. The larger the value of λ, the more sparse the result would be. Let us take a look at a one-dimensional case to understand this: Given a function $f(x; a) = (x - a)^2 + \lambda|x|$ and the rule $\hat{x} = \operatorname{argmin} f(x; a)$. if $\lambda = 0, x = a$ is the solution. However, if $\lambda > 0$, without loss of generality, let's assume $a > 0$, we have the following two scenarios:

1. $x < 0, f(x; a) = (x - a)^2 - \lambda \cdot x > a^2$
2. $x \geq 0, f(x; a) = (x - a)^2 + \lambda \cdot x = x^2 - (2a - \lambda)x + a^2$

when $\lambda = 2a, f(x; a) = x^2 + a^2, x \geq 0$, thus $x = 0, f(x; a) = a^2$ is the solution for the optimization problem.

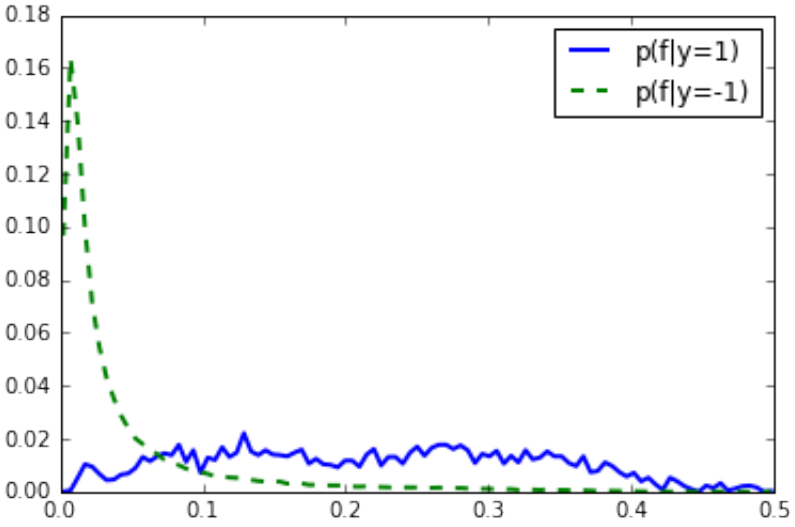
Question 3

Project 1 Bayesian Decision Making of Edge Detection

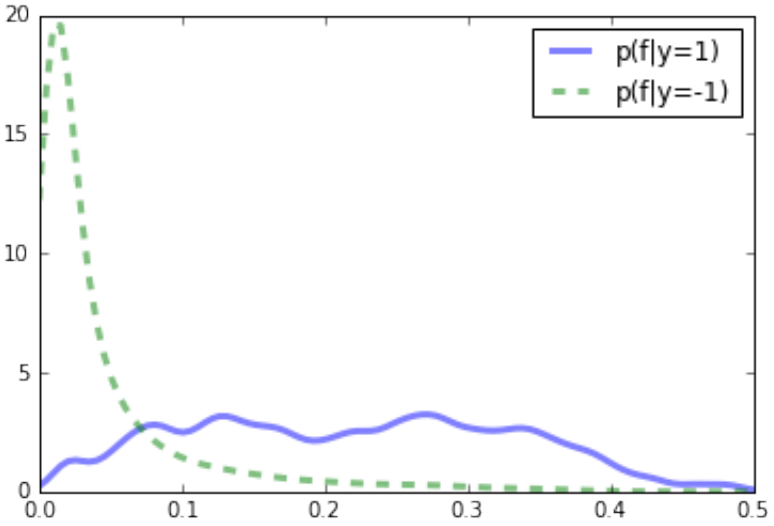
1) apply the edge detection algorithm on the butterfly (35010)

See the code in appendix for details. Here are the results from the code

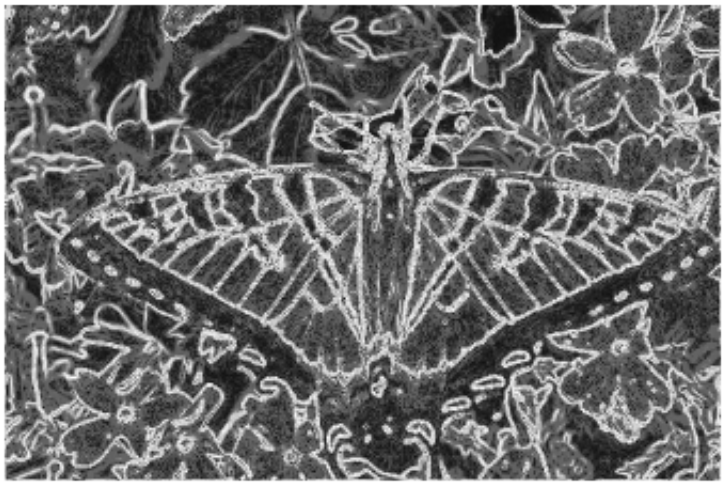
The (Normalized) Histogram of on/off edge pixels



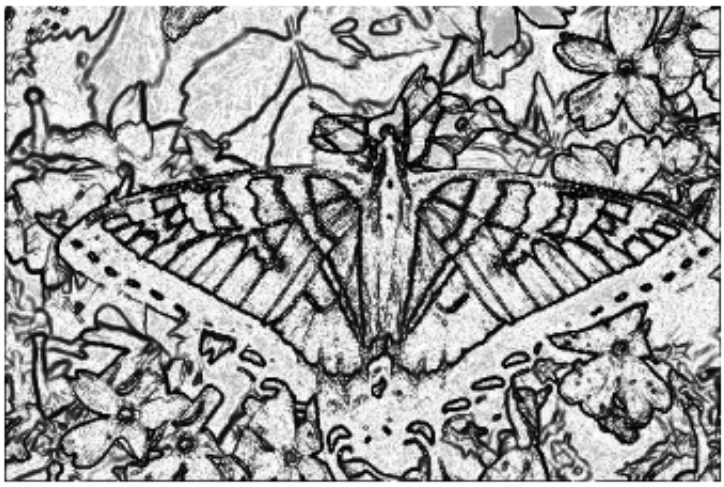
The estimated density function



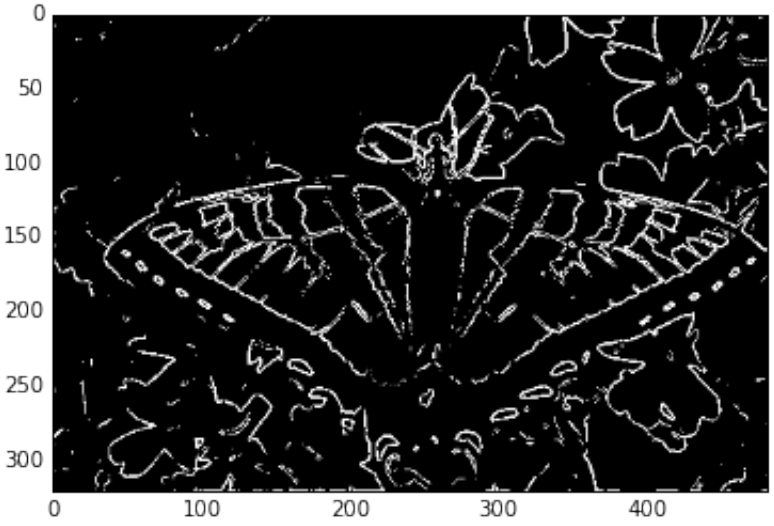
$P(\frac{dI}{dx}|\text{on edge})$



$P(\frac{dI}{dx}|\text{off edge})$



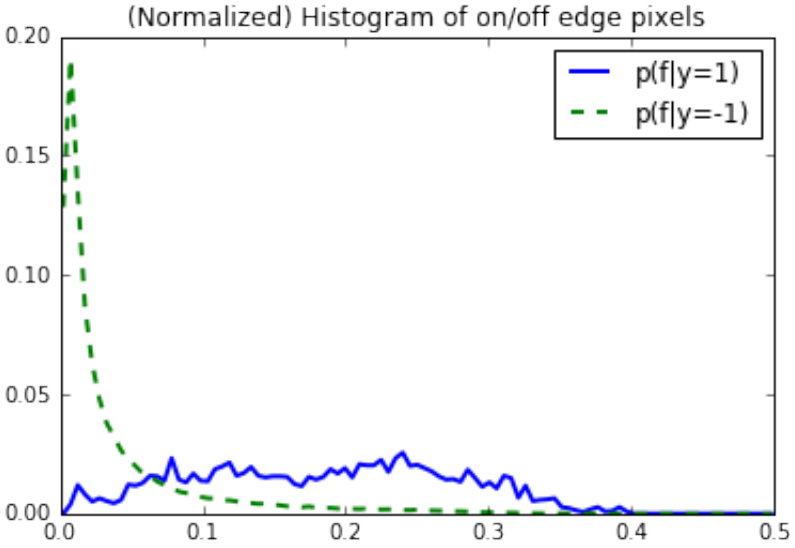
Using interactive widget, I found that threshold $T = 1.5$ yields the best result in my opinion, see the figure below



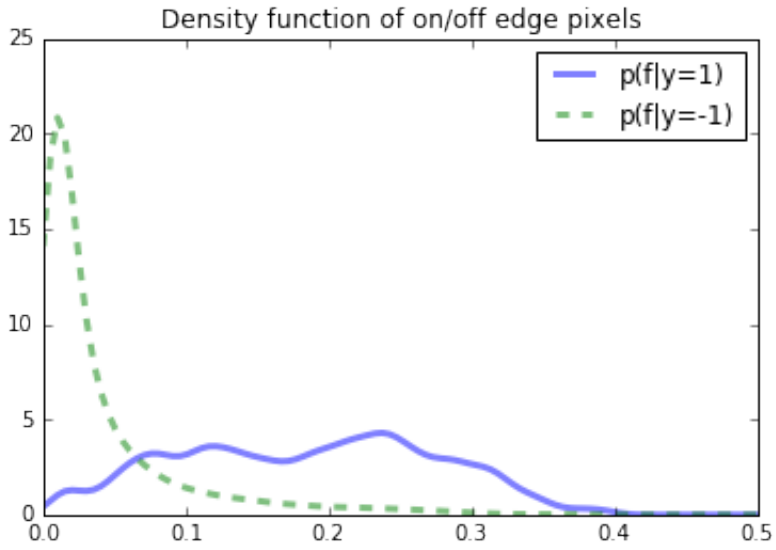
2) Use $\frac{dG*I}{dx}$ instead of $\frac{dI}{dx}$ for edge detection where G is a Gaussian. Show results for a couple of different variances `sigma` .

A) When `sigma = 0.5`

The (Normalized) Histogram of on/off edge pixels

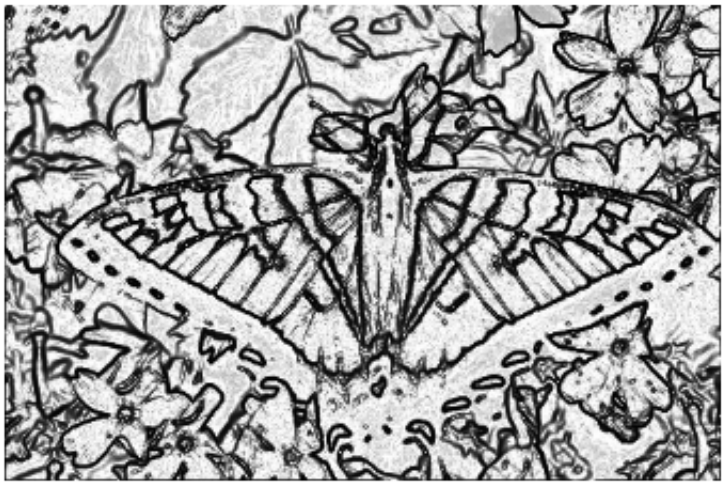


The estimated density function

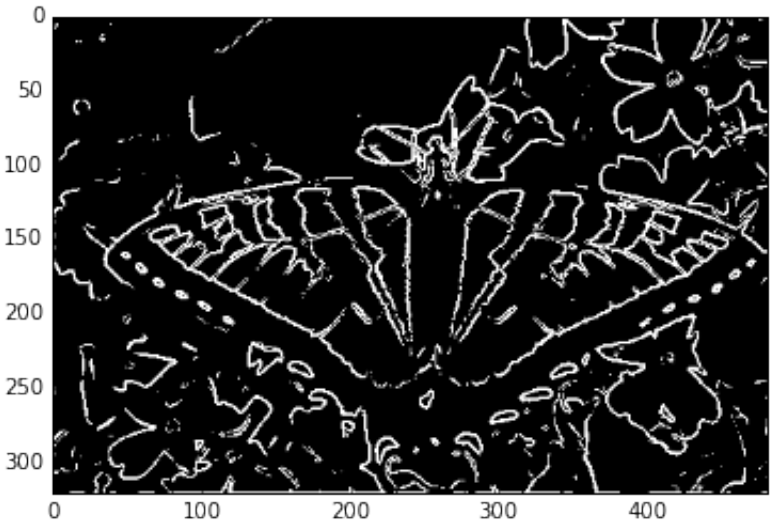


$P(\frac{dI}{dx}|\text{on edge})$

$P(\frac{dI}{dx}|\text{off edge})$

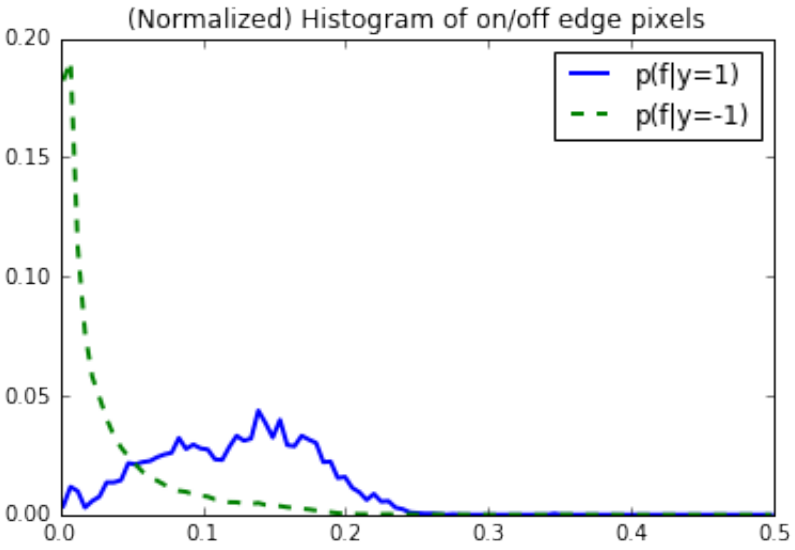


Using interactive widget, I found that threshold $T = 1.0$ yields the best result in my opinion, see the figure below

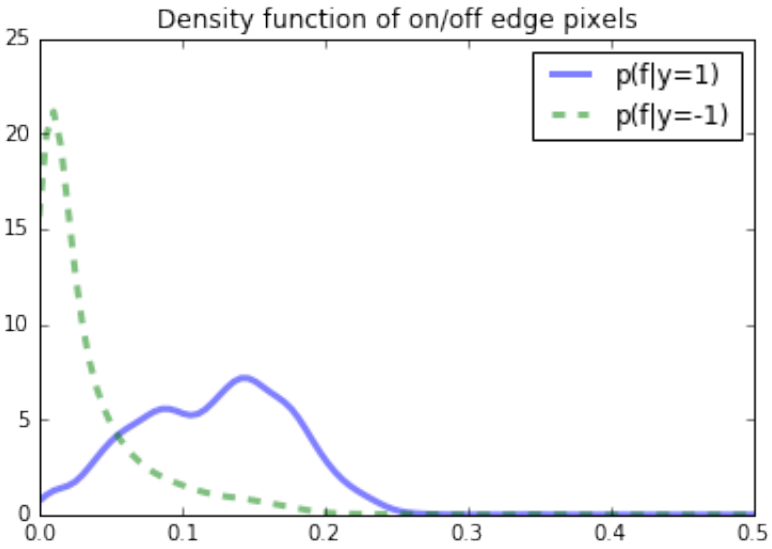


B) When `sigma = 1.0`

The (Normalized) Histogram of on/off edge pixels



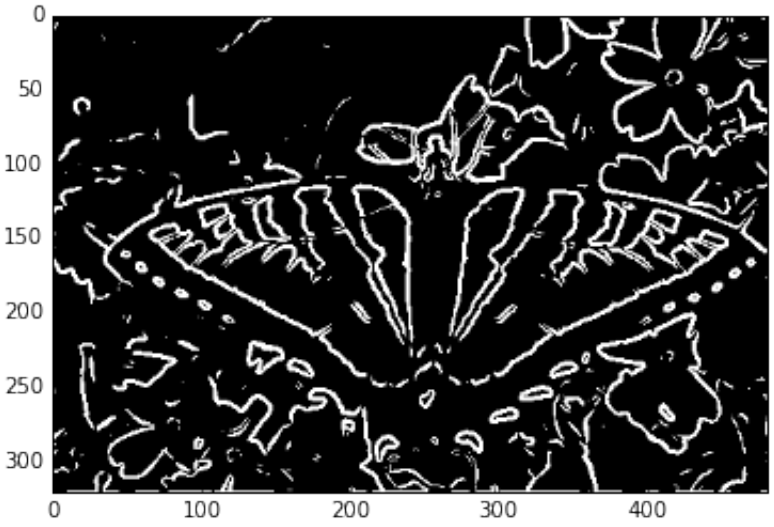
The estimated density function



$P(\frac{dI}{dx}|\text{on edge})$

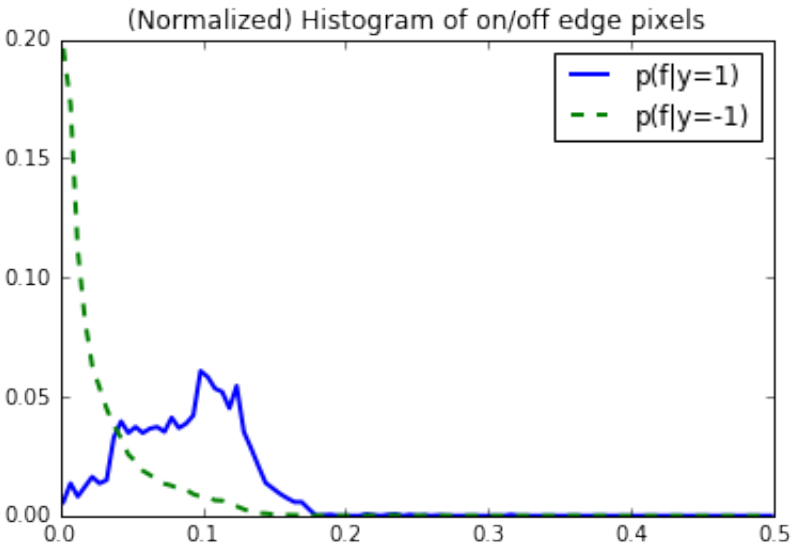
$P(\frac{dI}{dx}|\text{off edge})$

Using interactive widget, I found that threshold $T = 0.8$ yields the best result in my opinion, see the figure below

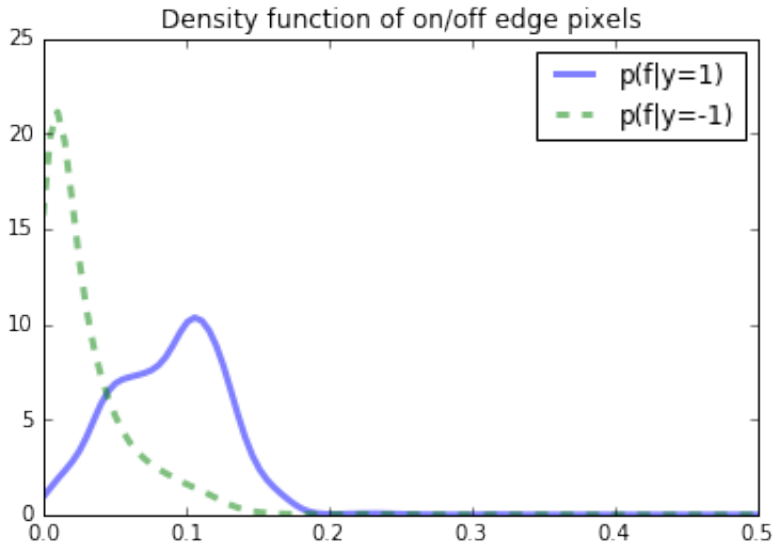


C) When `sigma = 1.5`

The (Normalized) Histogram of on/off edge pixels



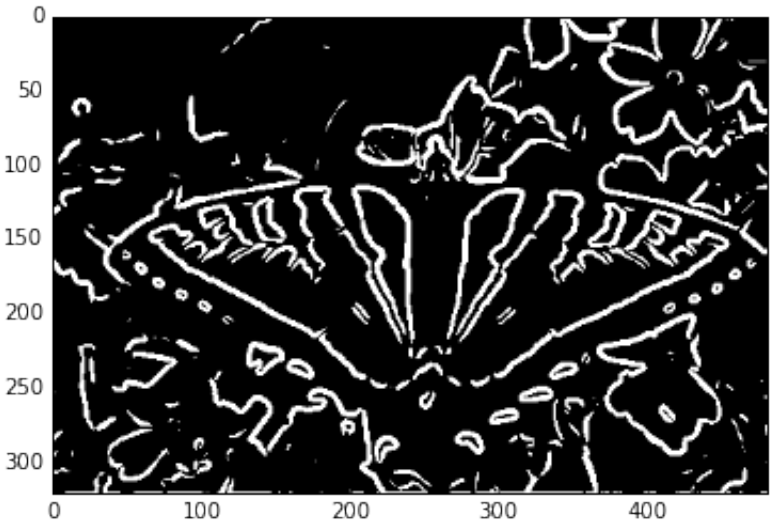
The estimated density function



$P(\frac{dI}{dx}|\text{on edge})$

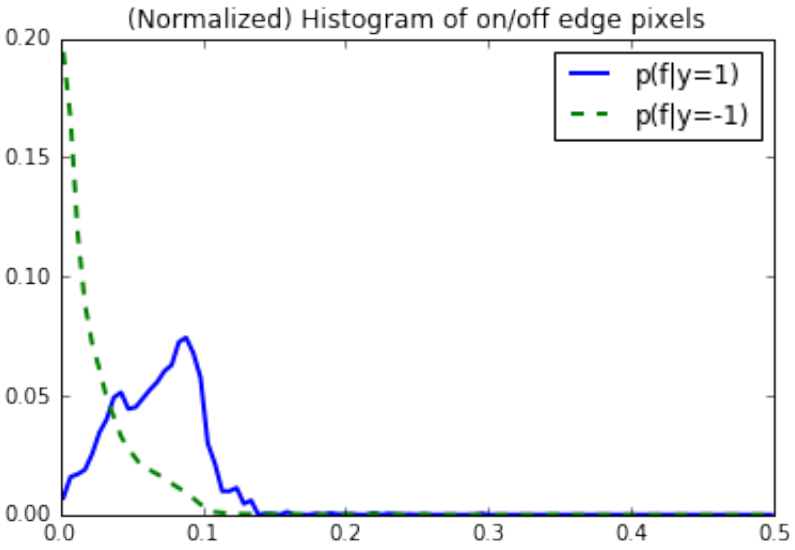
$P(\frac{dI}{dx}|\text{off edge})$

Using interactive widget, I found that threshold $T = 0.7$ yields the best result in my opinion, see the figure below

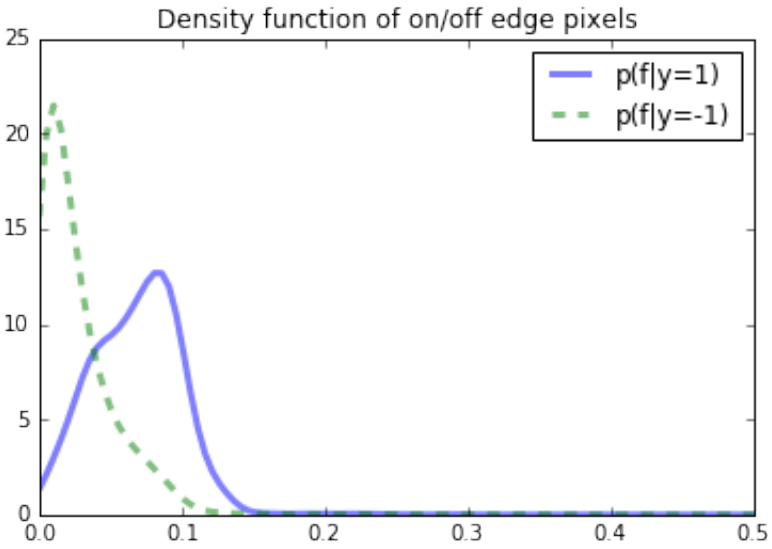


D) When `sigma = 2.0`

The (Normalized) Histogram of on/off edge pixels



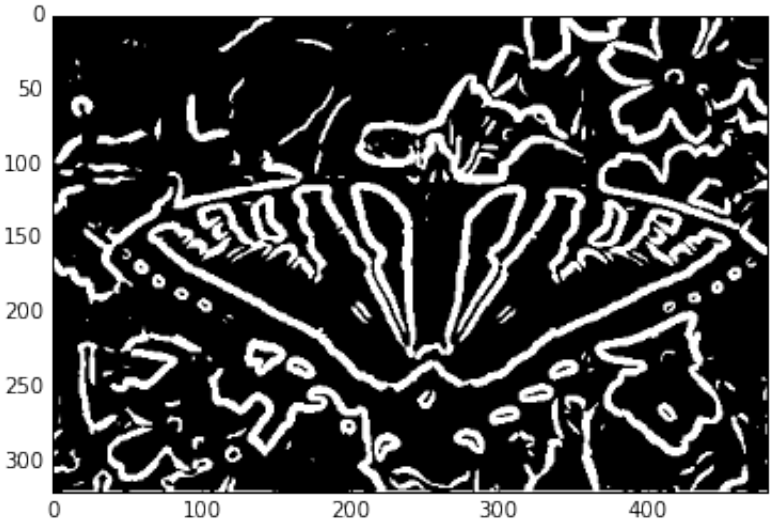
The estimated density function



$P(\frac{dI}{dx}|\text{on edge})$

$P(\frac{dI}{dx}|\text{off edge})$

Using interactive widget, I found that threshold $T = 0.2$ yields the best result in my opinion, see the figure below



Summary

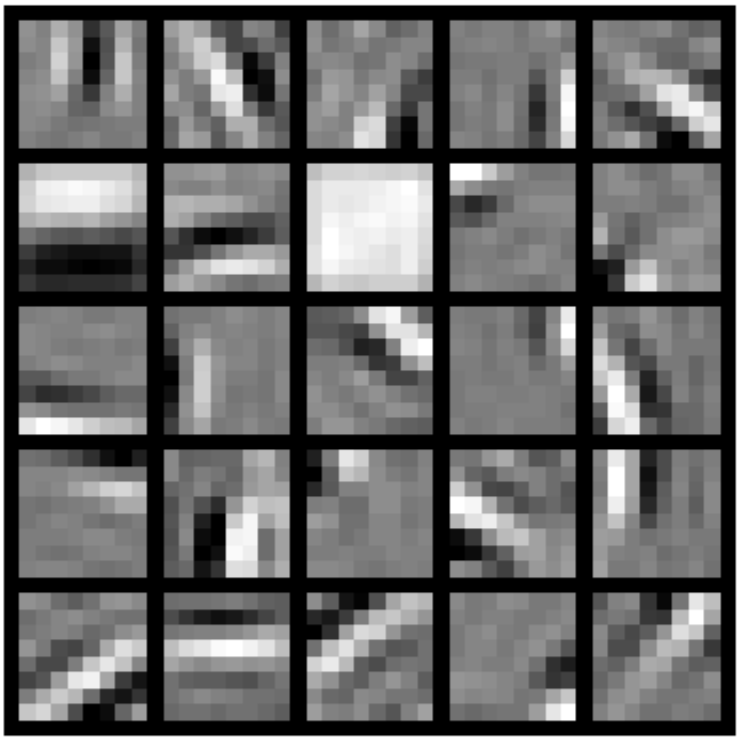
The higher the `sigma`, the thinker the line in results, and the lower the threshold. The Gaussian filter smoothed the whole image, thus clear out some small edges for details, but remained large edges such as the contour of the butterfly. But the smoothing also made it harder to preserve the details of th butterfly, especially details at its head.

Project 2 Sparse Coding (in collaboration with Aditya)

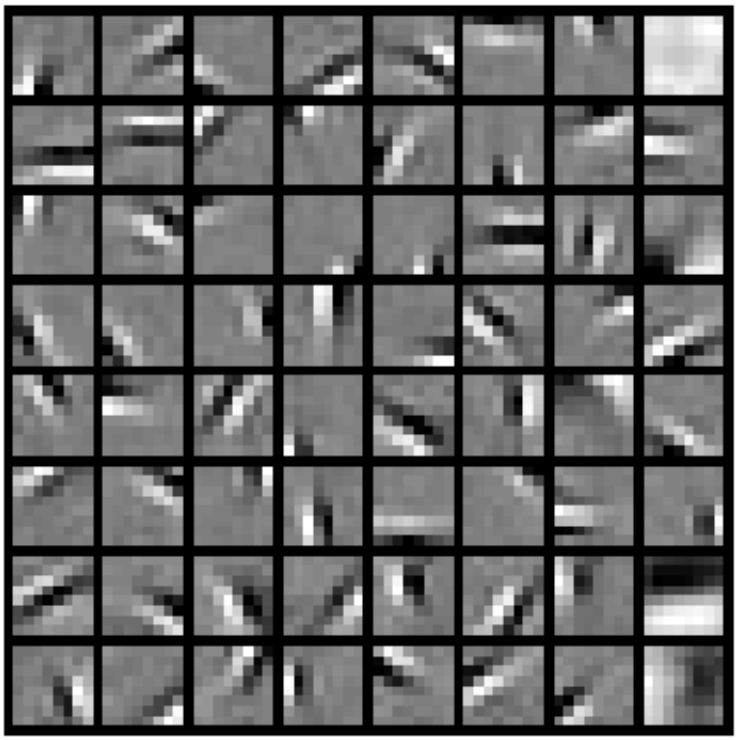
1) Sparse coding of natural images.

Results:

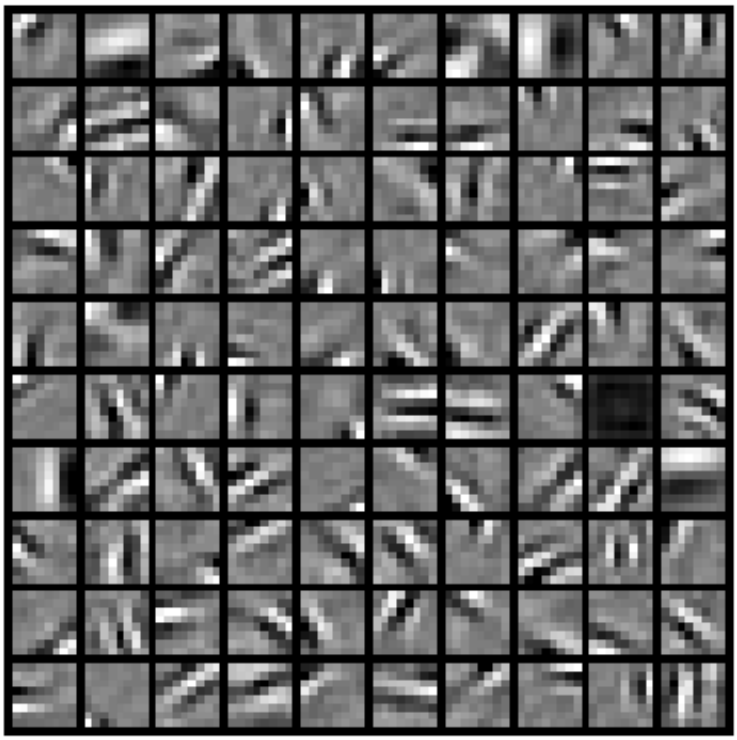
1) $M = 25$: Basis



2) $M = 64$: Basis

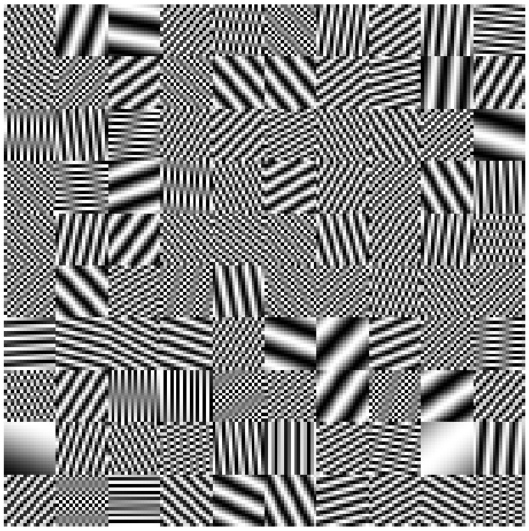


3) $M = 100$: Basis

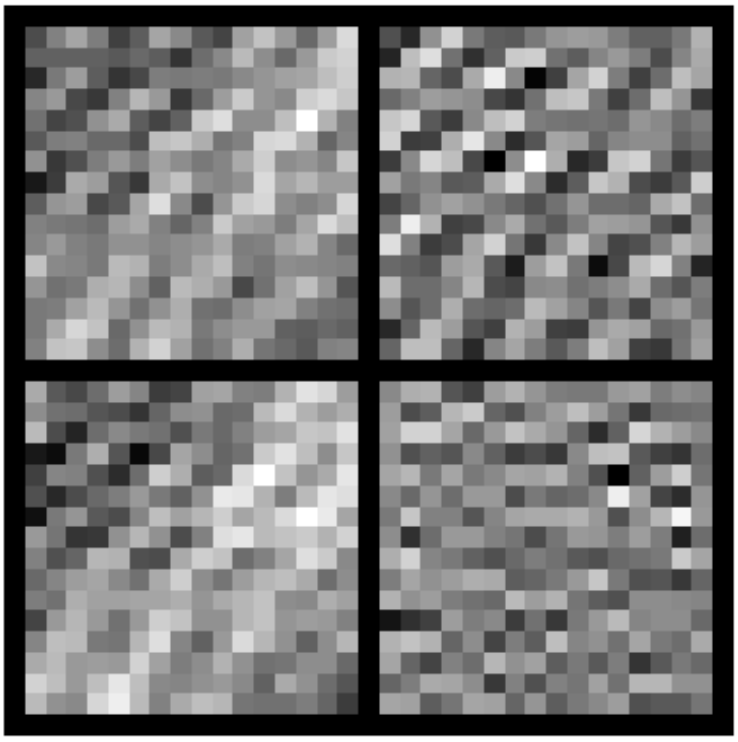


2) Sparse coding Learn From High Frequence Gabor

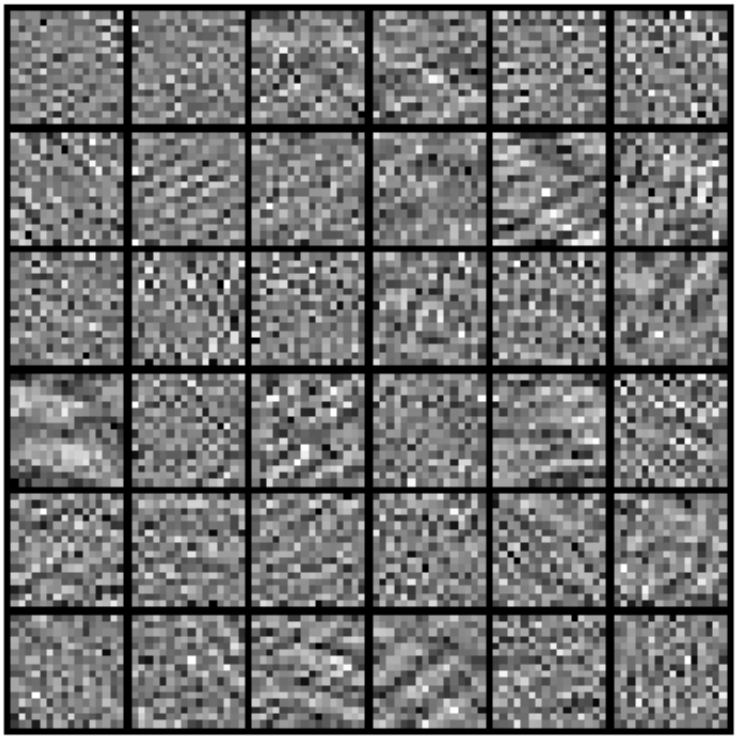
The high frequency bank: Note that the frequency for each gabor is randomly generated, so there are a few gabors that have quite low frequence. However, the overall gabor bank is full of high frequency gabors. Here is only a selection of 100 out of 10000 garbors generated by `gensin.m` : `omeganorm = 1e3`



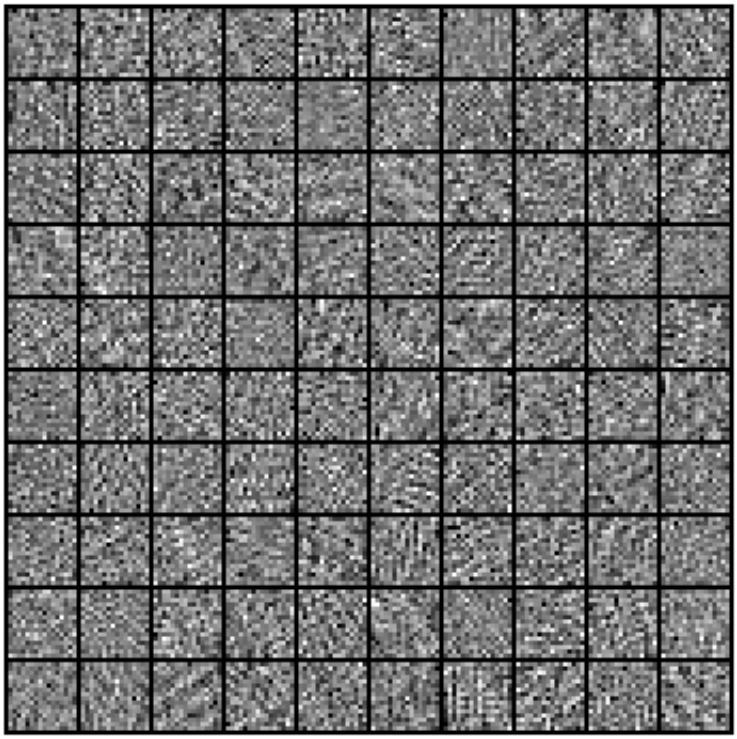
A) $M = 4$: Basis



B) $M = 36$: Basis



C) $M = 100$: Basis



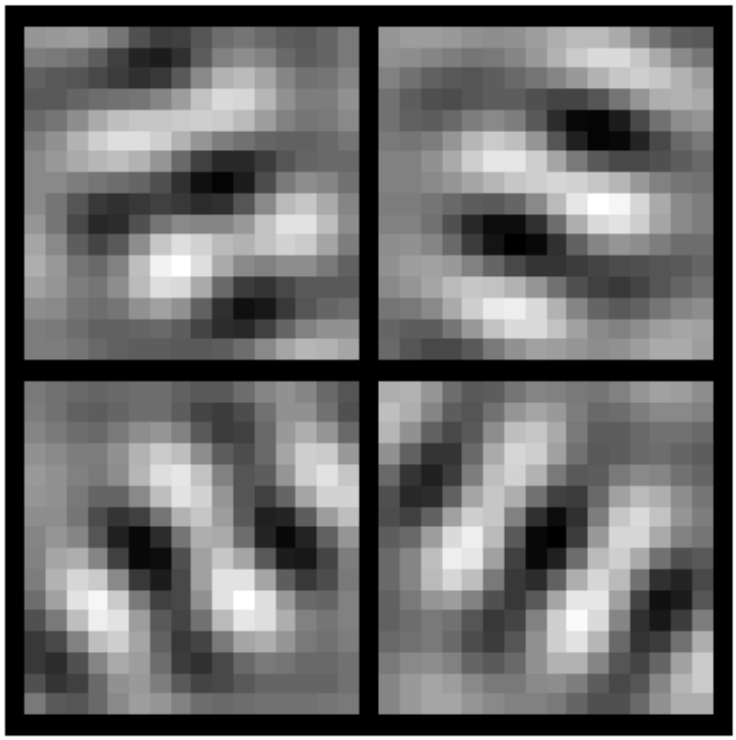
summary

The larger the M, the finer the basis functions are, and the more various the basis functions becomes. When $M = 4$, the learned basis functions do not have clear patterns. When M is large, each basis contains more details, we can see some checkboard-like patterns, as well as orientations in some basis functions. The larger the M, the more similar the basis functions are to the original sinusoid functions. Because the sparse coding is to learn a over-compelete set basis functions such that only minimal numbers of basis functions need to be activated in order to represent an image. When M is large enough, we can know that each basis function would be a image. In that case, each image only activates one basis function.

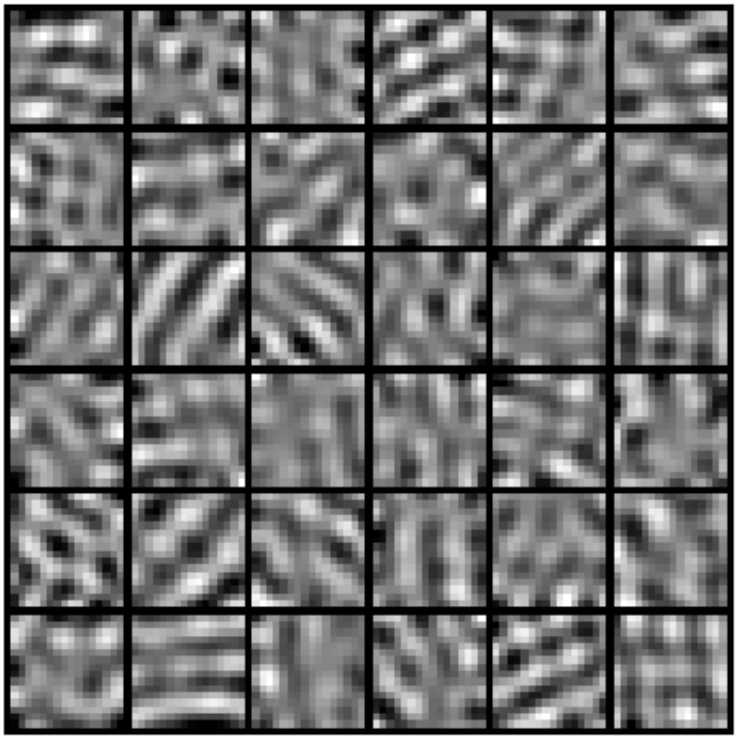
3) Sparse coding Learn From Low Frequence Gabor

The low frequency bank: Here is only a selection of 100 out of 10000 garbors generated by `gensin.m` : `omeganorm = 1`

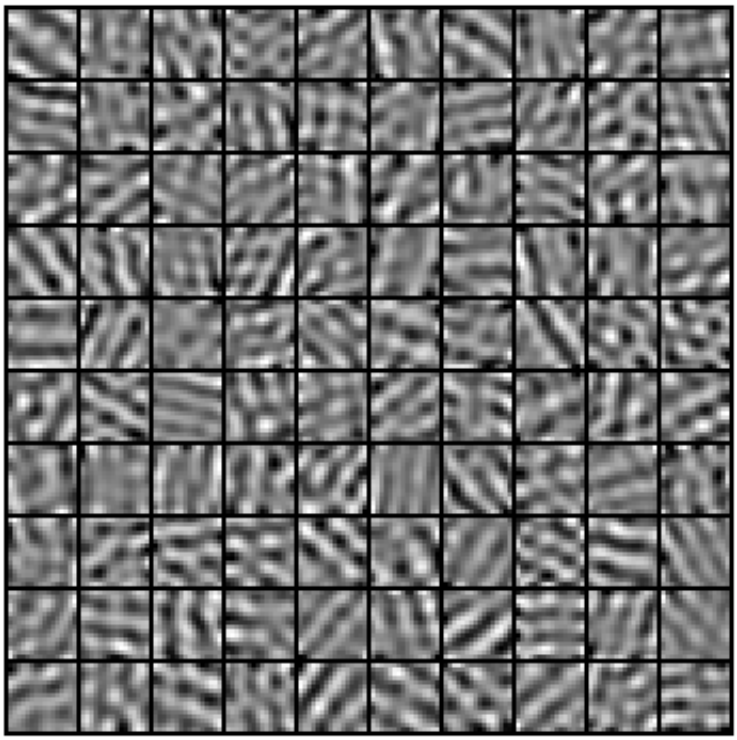
A) $M = 4$: Basis



B) $M = 36$: Basis



C) $M = 100$: Basis



summary

The larger the M, the similar the basis functions to original sinusoid functions, the clearer the sinusoid patterns in basis functions

Comparison

We can see that the same learning algorithm produces different basis functions when different input sinusoid banks are used. We can see huge differences in inputs. The high frequency sinusoid functions bank contains lots of sinusoid functions that have sharp contrast, thin lines, or even checkboard-like discontiuous-intensity small pixels. In contrast, the low frequency sinusoid bank contains lots of sinusoid functions with raletively thicker lines, and all sinusoid functions are quite similar to each other in spatial frequency, but only different in orientation.

When learned from these two bank of images, the sparsenet algorithm captured the statistical regularities in the input set. Thus, when learned from the high frequenc bank, the output basis functions are usually have more fine details or larger changes in terms of neighboring pixel intensity. In contrast, when learned from low frequency bank, the output bases are usually blurry, and neighboring pixels change intensity gradually.

Appendix: CODE

Project 1

1) apply the edge detection algorithm on the butterfly (35010)

CODE:

```
#Cell 1: initialization
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def myimshow(im):
    plt.figure()
    plt.axis('off')
    plt.imshow(im, cmap=plt.gray())

#Cell 2: read image
# Show image and edge labeling
id = '35010' # butterfly

def loadData(id):
    from skimage.color import rgb2gray
    edgeMap = plt.imread('data/edge/boundaryMap/' + id + '.bmp'); edgeMap = edgeMap[:,:,0]
    im = plt.imread('data/edge/trainImgs/' + id + '.jpg')
    grayIm = rgb2gray(im)
    return [grayIm, edgeMap]

[im, edgeMap] = loadData(id)
myimshow(edgeMap); # title('Boundary')
myimshow(im * (edgeMap==0)); # title('Image with boundary')

#Cell 3: Define function to filter image
def dIdx(im):
    # Compute magnitude of gradient
    # 'CentralDifference' : Central difference gradient dI/dx = (I(x+1)- I(x-1))/ 2
    dx = (np.roll(im, 1, axis=1) - np.roll(im, -1, axis=1))/2
    dy = (np.roll(im, 1, axis=0) - np.roll(im, -1, axis=0))/2
    mag = np.sqrt(dx**2 + dy**2)
    return mag

def dgIdx(im, sigma=1.5):
```

```
    from scipy.ndimage import gaussian_filter
    gauss = gaussian_filter(im, sigma = sigma)
    dgauss = dIdx(gauss)
    return dgauss

dx = dIdx(im)
dgI = dgIdx(im)

#Cell 4: Compute the Pon/Poff distribution
def kde(x):
    # Kernel density estimation, to get P(dI/dx | on edge) and P(dI/dx | off edge) from data
    from scipy.stats import gaussian_kde
    f = gaussian_kde(x, bw_method=0.01 / x.std(ddof=1))
    return f

def ponEdge(im, edgeMap):
    # Compute on edge histogram
    # im is filtered image

    # Convert edge map to pixel index
    flattenEdgeMap = edgeMap.flatten()
    edgeIdx = [i for i in range(len(flattenEdgeMap)) if flattenEdgeMap[i]]

    # find edge pixel in 3x3 region, shift the edge map a bit, in case of inaccurate boundary
    [offx, offy] = np.meshgrid(np.arange(-1,2), np.arange(-1,2)); offx = offx.flatten(); offy =
    maxVal = np.copy(im)
    for i in range(9):
        im1 = np.roll(im, offx[i], axis=1) # x axis
        im1 = np.roll(im1, offy[i], axis=0) # y axis
        maxVal = np.maximum(maxVal, im1)

    vals = maxVal.flatten()
    onEdgeVals = vals[edgeIdx]

    bins = np.linspace(0,0.5, 100)
    [n, bins] = np.histogram(onEdgeVals, bins=bins)
    # n = n+1 # Avoid divide by zero

    pon = kde(onEdgeVals)

    return [n, bins, pon]


def poffEdge(im, edgeMap):
    flattenEdgeMap = edgeMap.flatten()
```

```
    noneEdgeIdx = [i for i in range(len(flattenEdgeMap)) if not flattenEdgeMap[i]]

    vals = im.flatten()
    offEdgeVals = vals[noneEdgeIdx]

    bins = np.linspace(0,0.5, 100)
    n, bins = np.histogram(offEdgeVals, bins=bins)

    # n = n+1
    # p = n / sum(n)

    poff = kde(offEdgeVals)

    return [n, bins, poff]

dx = dIdx(im)
[n1, bins, pon] = ponEdge(dx, edgeMap)
[n2, bins, poff] = poffEdge(dx, edgeMap)

plt.figure(); # Plot on edge
# title('(Normalized) Histogram of on/off edge pixels')
plt.plot((bins[:-1] + bins[1:])/2, n1.astype(float)/sum(n1), '-', lw=2, label="p(f|y=1)")
plt.plot((bins[:-1] + bins[1:])/2, n2.astype(float)/sum(n2), '--', lw=2, label="p(f|y=-1)")
plt.legend()

plt.figure()
# title('Density function of on/off edge pixels')
plt.plot(bins, pon(bins), '-', alpha=0.5, lw=3, label="p(f|y=1)")
plt.plot(bins, poff(bins), '--', alpha=0.5, lw=3, label="p(f|y=-1)")
plt.legend()

#Cell 5: compute Pon
%%time
ponIm = pon(dx.flatten()).reshape(dx.shape) # evaluate pon on a vector and reshape the vector t
myimshow(ponIm)

#Cell 6: compute Poff
%%time
poffIm = poff(dx.flatten()).reshape(dx.shape) # Slow, evaluation of this cell may take several
myimshow(poffIm)

#Cell 7: Draw ROC curve
def ROCpoint(predict, gt):
    # predict = (log(ponIm/poffIm)>=T)
    truePos = (predict==True) & (gt == predict)
```

```
    trueNeg = (predict==False) & (gt == predict)

    falsePos = (predict==True) & (gt != predict)
    falseNeg = (predict==False) & (gt != predict)

    y = np.double(truePos.sum()) / np.sum(gt == True)
    x = np.double(falsePos.sum()) / np.sum(gt == False)
    return [x, y]

p = []
for T in np.arange(-5, 5, step=0.1):
    predict = (np.log(ponIm/poffIm)>=T)
    p.append(ROCpoint(predict, gt))
x = [v[0] for v in p]
y = [v[1] for v in p]
plt.plot(x, y)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')

#Cell 8: Interactively find the best result
from IPython.html.widgets import interact, interactive, fixed
def demoThreshold(T):
    predict = (np.log(ponIm/poffIm)>=T)
    plt.figure(1)
    plt.imshow(predict)
    p = ROCpoint(predict, gt)
    plt.figure(2)
    plt.plot(x, y)
    plt.plot(p[0], p[1], '*')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')

# compute ROC curve
p = []
for T in np.arange(-5, 5, step=0.1):
    predict = (np.log(ponIm/poffIm)>=T)
    p.append(ROCpoint(predict, gt))
x = [v[0] for v in p]
y = [v[1] for v in p]

interact(demoThreshold, T=(-5, 5, 0.1))
```

2) Use $\frac{dG*I}{dx}$ instead of $\frac{dI}{dx}$ for edge detection where G is a Gaussian. Show results for a couple of

different variances `sigma` .

CODE: “`python

cell 1 for demostration, prepare for interaction

def plotThreshold(condition, testT): ponIm = ponIms[condition] poffIm = poffIms[condition] gt = (edgeMap!=0) # Ground-truth labels # compute ROC curve p = [] for iterT in np.arange(-5, 5, step=0.1): predict = (np.log(ponIm/poffIm)>=iterT) p.append(ROCpoint(predict, gt)) x = [v[0] for v in p] y = [v[1] for v in p] predict = (np.log(ponIm/poffIm)>=testT) plt.figure(1) plt.imshow(predict) p = ROCpoint(predict, gt) plt.figure(2) plt.plot(x, y) plt.plot(p[0], p[1], ‘*’) plt.xlabel(‘False positive rate’) plt.ylabel(‘True positive rate’)

Cell 2: Edge Detection pipline

def EdgeDetectionGaussian(sigma = 1.5): dgIx = dgIdx(im, sigma) [n1, bins, pon] = ponEdge(dgIx, edgeMap) [n2, bins, poff] = poffEdge(dgIx, edgeMap)

```
plt.figure(); # Plot on edge
plt.title('(Normalized) Histogram of on/off edge pixels')
plt.plot((bins[:-1] + bins[1:])/2, n1.astype(float)/sum(n1), '-', lw=2, label="p(f|y=1)")
plt.plot((bins[:-1] + bins[1:])/2, n2.astype(float)/sum(n2), '--', lw=2, label="p(f|y=-1)")
plt.legend()

plt.figure()
plt.title('Density function of on/off edge pixels')
plt.plot(bins, pon(bins), '-', alpha=0.5, lw=3, label="p(f|y=1)")
plt.plot(bins, poff(bins), '--', alpha=0.5, lw=3, label="p(f|y=-1)")
plt.legend()

## pon
print "working on P-ON"
%%time
ponIm = pon(dgIx.flatten()).reshape(dgIx.shape)
myimshow(ponIm)
## poff, time consuming
print "working on P-OFF"
%%time
poffIm = poff(dgIx.flatten()).reshape(dgIx.shape)
myimshow(poffIm)

return (ponIm, poffIm)
```

Cell 3 A case where sigma = 0.5

nSigmas = 4 step = 0.5 sigmas = (np.arange(nSigmas) + 1) * step ponIms = []; poffIms = []; conditions = np.arange(nSigmas) print ‘conditions:’, conditions print ‘sigmas:’, sigmas

Cell 4: run the different conditions

for sigma in sigmas: ponTmp, poffTmp = EdgeDetectionGaussian(sigma = sigma) ponIms.append(ponTmp) poffIms.append(poffTmp)

Cell 5: Interactive results

interact(plotThreshold, condition = (0,3,1), testT=(-5, 5, 0.1))

```
#### Project 2

##### 1) Sparse coding of natural images.

CODE:

```matlab
load IMAGES

dim1 = [64, 25];
dim2 = [64, 64];
dim3 = [64, 100];

% M = 25
A1 = rand(dim1)-0.5;
A1 = A1*diag(1./sqrt(sum(A1.*A1)));
figure(1), colormap(gray)
A = A1;
sparsenet

% M = 64
A2 = rand(dim2)-0.5;
A2 = A2*diag(1./sqrt(sum(A2.*A2)));
figure(1), colormap(gray)
A = A2;
sparsenet

% M = 100
A3 = rand(dim3)-0.5;
A3 = A3*diag(1./sqrt(sum(A3.*A3)));
figure(1), colormap(gray)
A = A3
sparsenet
```

2) Sparse coding Learn From High Frequence Gabor

CODE:

Learning Code `sparseFromGabor.m` (adapted from sparsenet):

```
% sparsefromGabor.m - simulates the sparse coding algorithm
```
```

```
%
% Adapted from sparsenet.m
%
% Before running you must first define A and load Garbor Images.
% See the README file for further instructions.


gabor_sz=[16 16];

num_trials=10000;
batch_size=100;

num_images=size(patchsin,2);
%image_size=size(patchsin,1);
image_size = gabor_sz(1);
BUFF=4;

[L M]=size(A);
sz=sqrt(L);

eta = 1.0;
noise_var= 0.01;
beta= 2.2;
sigma=0.316;
tol=.01;

VAR_GOAL=0.1;
S_var=VAR_GOAL*ones(M,1);
var_eta=.001;
alpha=.02;
gain=sqrt(sum(A.*A))';

X=zeros(L,batch_size);

display_every=100;

h=display_network(A,S_var);

% main loop

for t=1:num_trials

    % choose an image for this batch

    selection = randperm(num_images, batch_size);
```

```
    X = patchsin(:,selection);

    % calculate coefficients for these data via conjugate gradient routine

    S=cgf_fitS(A,X,noise_var,beta,sigma,tol);

    % calculate residual error

    E=X-A*S;

    % update bases

    dA=zeros(L,M);
    for i=1:batch_size
        dA = dA + E(:,i)*S(:,i)';
    end
    dA = dA/batch_size;

    A = A + eta*dA;

    % normalize bases to match desired output variance

    for i=1:batch_size
        S_var = (1-var_eta)*S_var + var_eta*S(:,i).*S(:,i);
    end
    gain = gain .* ((S_var/VAR_GOAL).^alpha);
    normA=sqrt(sum(A.*A));
    for i=1:M
        A(:,i)=gain(i)*A(:,i)/normA(i);
    end

    % display

    if (mod(t,display_every)==0)
        display_network(A,S_var,h);
    end

end
```

Learn from the gabor bank with differnt M

```
load('patchsin_highfre.mat');

dim1 = [16*16 4];
dim2 = [16*16 36];
dim3 = [16*16 100];

A1 = rand(dim1)-0.5;
A1 = A1*diag(1./sqrt(sum(A1.*A1)));
figure(1), colormap(gray)
A = A1;
sparseFromGabor

A2 = rand(dim2)-0.5;
A2 = A2*diag(1./sqrt(sum(A2.*A2)));
figure(1), colormap(gray)
A = A2;
sparseFromGabor

A3 = rand(dim3)-0.5;
A3 = A3*diag(1./sqrt(sum(A3.*A3)));
figure(1), colormap(gray)
A = A3
sparseFromGabor
```

3) Sparse coding Learn From Low Frequence Gabor

Learn from the gabor bank with differnt M, similar to the second code block in 2).