Compressive Sensing

In this project, we will explore an imaging technology that revolutionaized medicine in the 19th centuary - Computed Tomograph (CT).

At a high level, CT has an array of X-ray transmitters on one side of a person and another array of detectors on the other side of this person. As X-rays go thru this person's body, different body parts attenuate or reduces differently X-ray's energy, the resulting signal captured at the detectors can be used to back-project or resemble the true image of the person.

The system design principle is that a linear function can model as X-ray goes thru this person, and another linear function at the dectectors.The image recovery can be done using Lasso Linear Regression.

This starts looking very similar to Neural Network!

Load Packages

In [1]:
```
import scipy.io as sio

import numpy as np

import random
random.seed(2)

import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

The observed image has a dimension 50 × 50, in grey scale. That's to say, there are 2500 pixels in total, same as the true image.

CT machine can't effectively measure all of the 2500 pixels, but can at measuring linear combination of pixels. In fact, there are n = 1300 linear combinations, with the weights in the linear combination being random, in fact, independently distributed as N(0,1). Because the machine is not perfect, we don't get to observe this directly, but we observe a noisy version. These measurements are given by the entries of the vector:

> $$y = Ax + \epsilon$$

where y ∈ R^{1300}, A ∈ $R^{1300\times2500}$, and ε ~ $N(0, 25 \times I_{1300})$ where I_n denotes the identity matrix of size n×n.

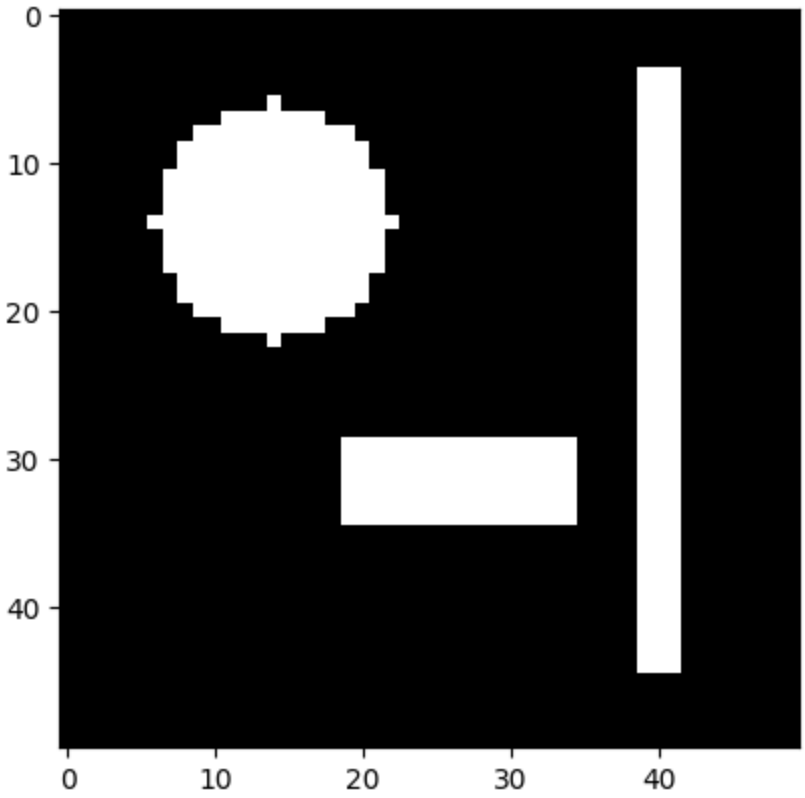
The main idea is although the number of measurements n = 1300 is smaller than the image dimension p = 2500, the true image is sparse. Thus we can recover the sparse image using few measurements exploiting its structure. This is the idea behind the field of compressed sensing.

Load Observed Image

In [24]:
```
matFile = sio.loadmat('cs.mat')
imdata = matFile['img']
print(imdata.shape)
```

```
(50, 50)
```

In [25]:
```
plt.imshow(imdata, cmap='gray')
plt.show()
```



Back-Projection at Detection Layer

Reshape imdata to fit into an updated observer linear function form:

$$y = Ax + \epsilon$$

y is also the output of the linear function at the previous layer (person), while x is the observed image loaded from the previous step.

In [28]:
```
X_d = imdata.reshape((2500,1))
print("X_d shape:",X_d.shape)
```

```
X_d shape: (2500, 1)
```

Create independent weight/coefficient A:

In [30]:
```
A = np.random.normal(0, 1, (1300,2500))
print("A shape:", A.shape)
```

```
A shape: (1300, 2500)
```

Create epsilon noise and reshape to 1300x1 dimension:

In [31]:
```
cm = 25 * np.eye(1300)
eps_d = np.random.multivariate_normal(mean=np.zeros(1300), cov=cm, size=1).T
print("eps_d shape:",eps_d.shape)
```

```
eps_d shape: (1300, 1)
```

Finally, plugging into the linear function to compute y:

In [32]:
```
y_d = A.dot(X_d) + eps_d
print("y_d shape:", y_d.shape)
```

```
y_d shape: (1300, 1)
```

Back-Project at Person Layer

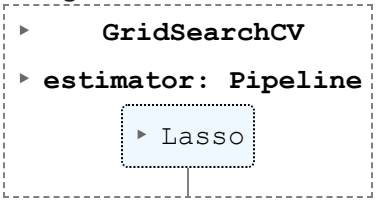
Apply LASSO regression and look for optimal hyperparameters.

> $$min_x \parallel y - Ax \parallel_2^2 + \lambda \parallel x \parallel_1$$

In [35]:
```
pipeline = Pipeline([
    ('clf', Lasso(max_iter=5000))
])

parameters = [
    {'clf__alpha': np.arange(0.01, 3, 0.01)}
]

cv_lasso = GridSearchCV(pipeline,
                  param_grid=parameters,
                  scoring='neg_mean_squared_error',
                  cv=5, n_jobs=-1, verbose=2)

cv_lasso.fit(A, y_d)
```

```
Fitting 5 folds for each of 299 candidates, totalling 1495 fits
```

Out[35]:



In [36]:
```
print(cv_lasso.best_estimator_)
print(cv_lasso.best_params_)
```

```
ALPHA = cv_lasso.best_params_['clf__alpha']
print("lambda or alpha selected:", ALPHA)
```

```
Pipeline(steps=[('clf', Lasso(alpha=0.04, max_iter=5000))])
{'clf__alpha': 0.04}
lambda or alpha selected: 0.04
```

Reconstruct True Image

In [40]:
```
model_lasso = Lasso(alpha = ALPHA)
model_lasso.fit(A, y_d)
X_o = model_lasso.coef_
plt.imshow(X_o.reshape(50,50), cmap='gray')
plt.show()
```