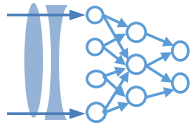



# Lecture 5: A gentle introduction to optimization

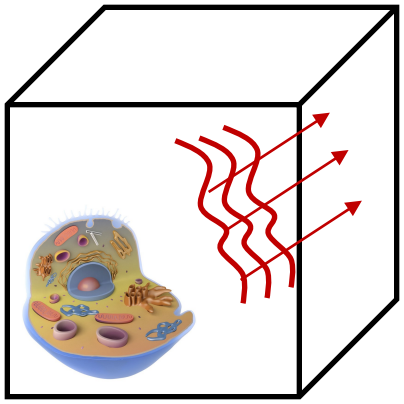
Machine Learning and Imaging

BME 590L
Roarke Horstmeyer

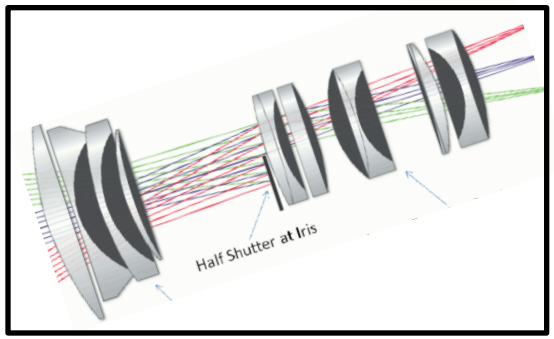
# ML+Imaging pipeline

deep imaging

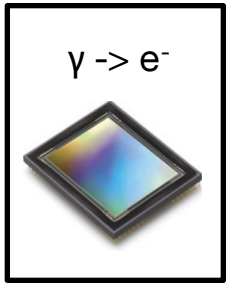
Real World



Measurement device

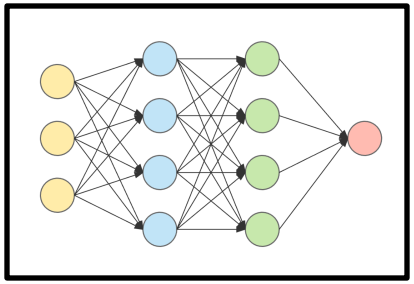



Digitization




Last Class

Machine Learning



This Class

## Discrete convolution

$$V(x_o) = \int_{-\infty}^{\infty} U(x_i)h(x_o - x_i)dx_i$$

$$v[x_0] = \sum_{x_i=-M}^{M} u[x_i]h[x_o - x_i]$$

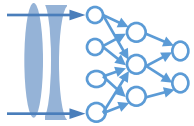
## Discrete 2D convolution

$$V(x_o, y_o) = \iint_{-\infty}^{\infty} U(x_i, y_i)h(x_o - x_i, y_o - y_i)dx_i dy_i$$

$$v[x_0, y_o] = \sum_{y_i=-L}^{L} \sum_{x_i=-M}^{M} u[x_i, y_i]h[x_o - x_i, y_o - y_i]$$

## Convolutions as a big matrix multiplication

$$(u * h)[x] = \sum_{m=0}^{N+M-2} u[m]h[x-m] \longrightarrow y = u * h = \begin{bmatrix} h_1 & 0 & \dots & 0 & 0 \\ h_2 & h_1 & \dots & \vdots & \vdots \\ h_3 & h_2 & \dots & 0 & 0 \\ \vdots & h_3 & \dots & h_1 & 0 \\ h_{m-1} & \vdots & \dots & h_2 & h_1 \\ h_m & h_{m-1} & \vdots & \vdots & h_2 \\ 0 & h_m & \dots & h_{m-2} & \vdots \\ 0 & 0 & \dots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & 0 & \dots & h_m \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix}$$




## Last thing – matrix and vector derivatives

$$\mathbf{u} = \mathbf{W}\mathbf{v}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} =$$

## Last thing – matrix and vector derivatives

$$\mathbf{u} = \mathbf{W}\mathbf{v}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} =$$

$$\mathbf{u}_3 = W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M$$

$$\frac{\partial u_3}{\partial v_2} = \frac{\partial}{\partial v_2}(W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M) = \frac{\partial}{\partial v_2}W_{3,2}v_2 = W_{3,2}$$

## Last thing – matrix and vector derivatives

$$\mathbf{u} = \mathbf{W}\mathbf{v}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} =$$

$$\mathbf{u}_3 = W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M$$

$$\frac{\partial u_3}{\partial v_2} = \frac{\partial}{\partial v_2}(W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M) = \frac{\partial}{\partial v_2}W_{3,2}v_2 = W_{3,2}$$

$$\frac{\partial u_i}{\partial v_j} = W_{i,j}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} = \mathbf{W}$$

## Last thing – matrix and vector derivatives

$$\mathbf{u} = \mathbf{W}\mathbf{v}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} =$$

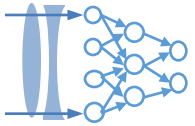
$$\mathbf{u}_3 = W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M$$

$$\frac{\partial u_3}{\partial v_2} = \frac{\partial}{\partial v_2}(W_{3,1}v_1 + W_{3,2}v_2 + ... + W_{3,M}v_M) = \frac{\partial}{\partial v_2}W_{3,2}v_2 = W_{3,2}$$

$$\frac{\partial u_i}{\partial v_j} = W_{i,j}$$

$$\frac{d\mathbf{u}}{d\mathbf{v}} = \mathbf{W}$$

- When confused, write out one entry, solve derivative and generalize
- Use dimensionality to help (if **x** has N elements, and **y** has M, then d**y**/d**x** must be NxM
- Take advantage of *The Matrix Cookbook*:
  - https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf




**Mathematical Optimization**: "Selection of a best element (with regard to some criterion) from a set of available alternatives"

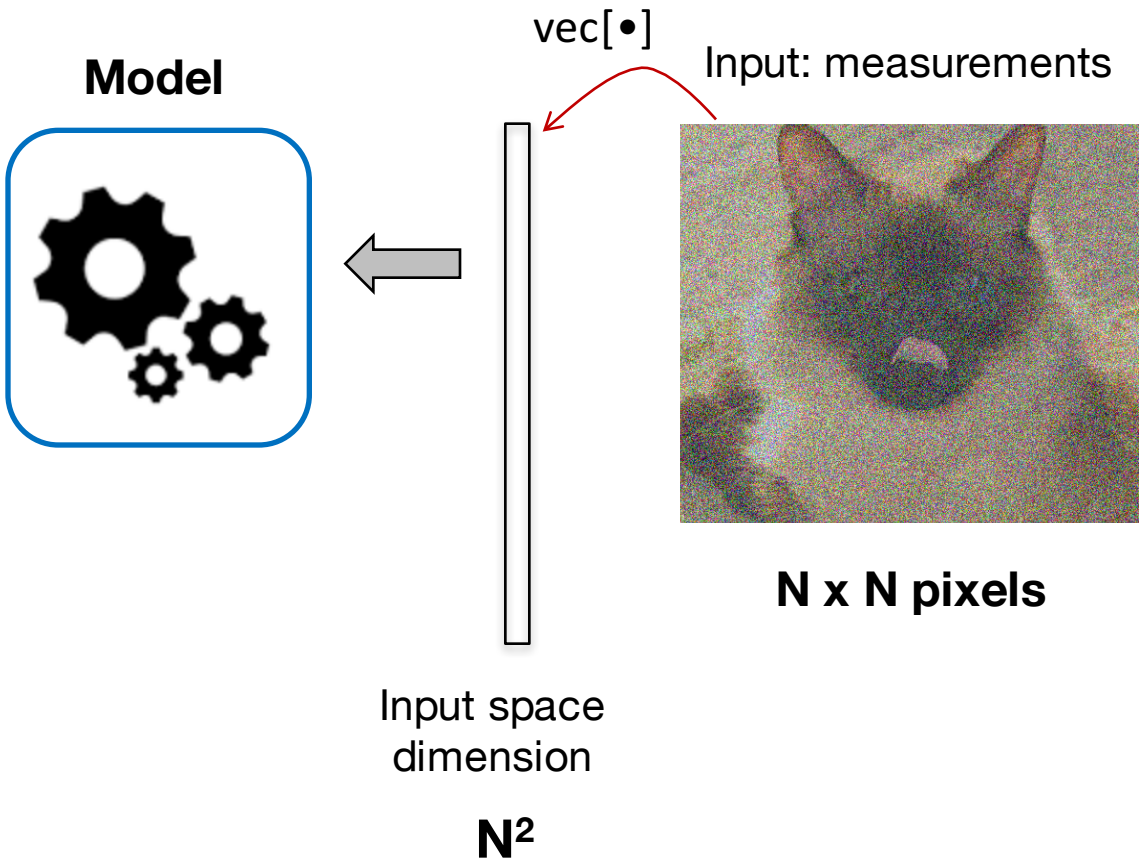
**3 elements:**

1) Your desired output (a better image, a clean signal, a classification of "cat" or "dog", etc.)

2) A model of what you are looking for - how you form the desired output from your measured data

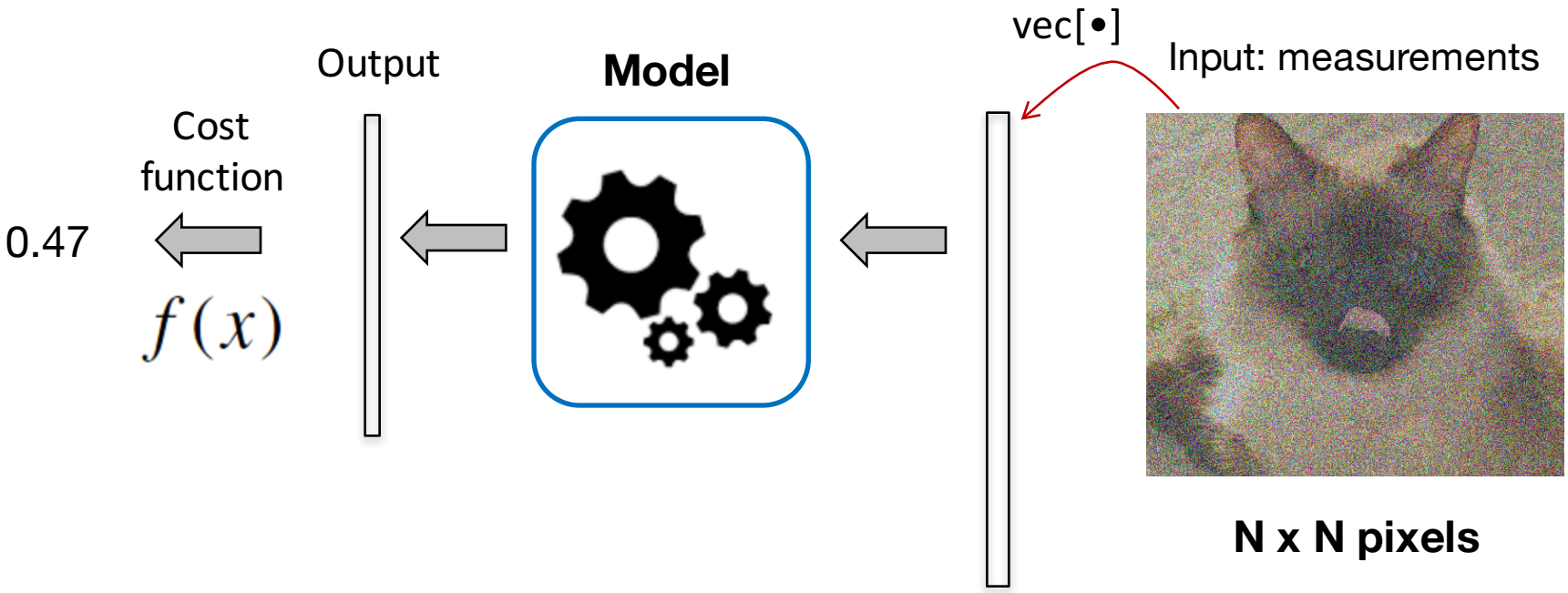
3) A cost function, to measure how close you're getting to the answer (the cost function minimum)

# Generalized optimization pipeline



# Generalized optimization pipeline

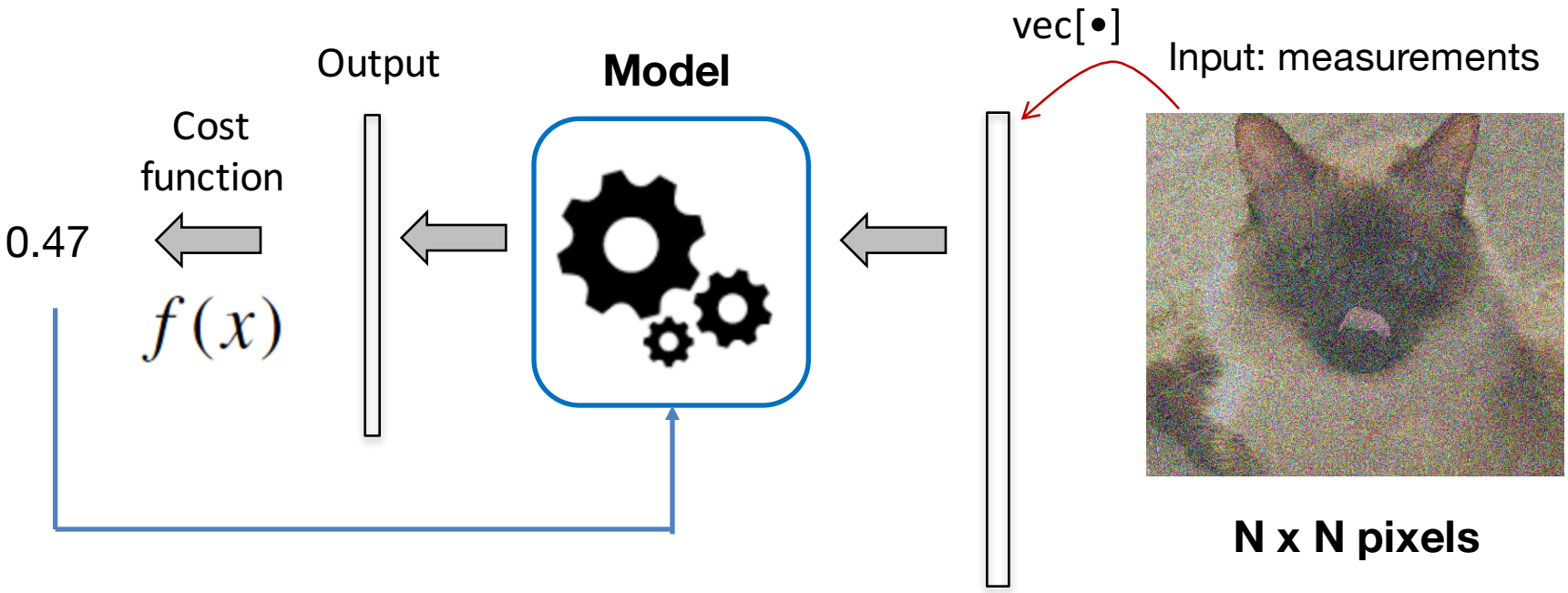
deep imaging




How well did we do?

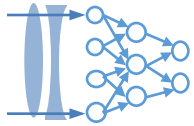
# Machine learning: update model to decrease error

deep imaging




How well did we do?

De-noising: "What is the closest image to what I detected, except without so many fluctuations"?

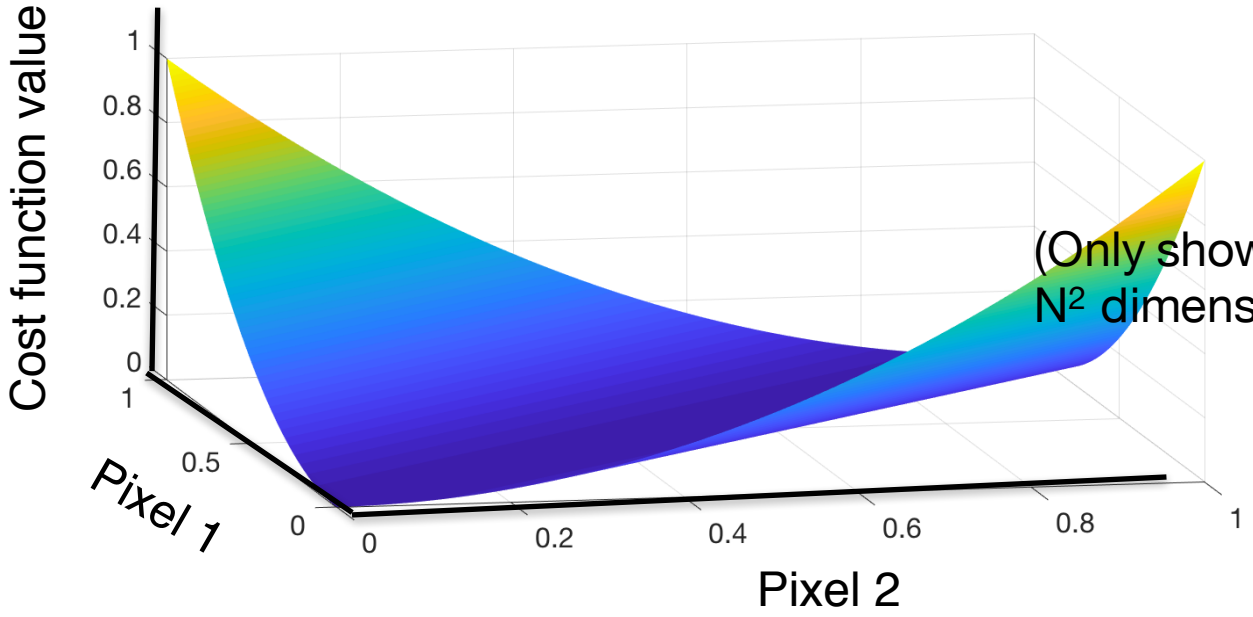



Input dimension: N x N image

Output dimension: N x N image

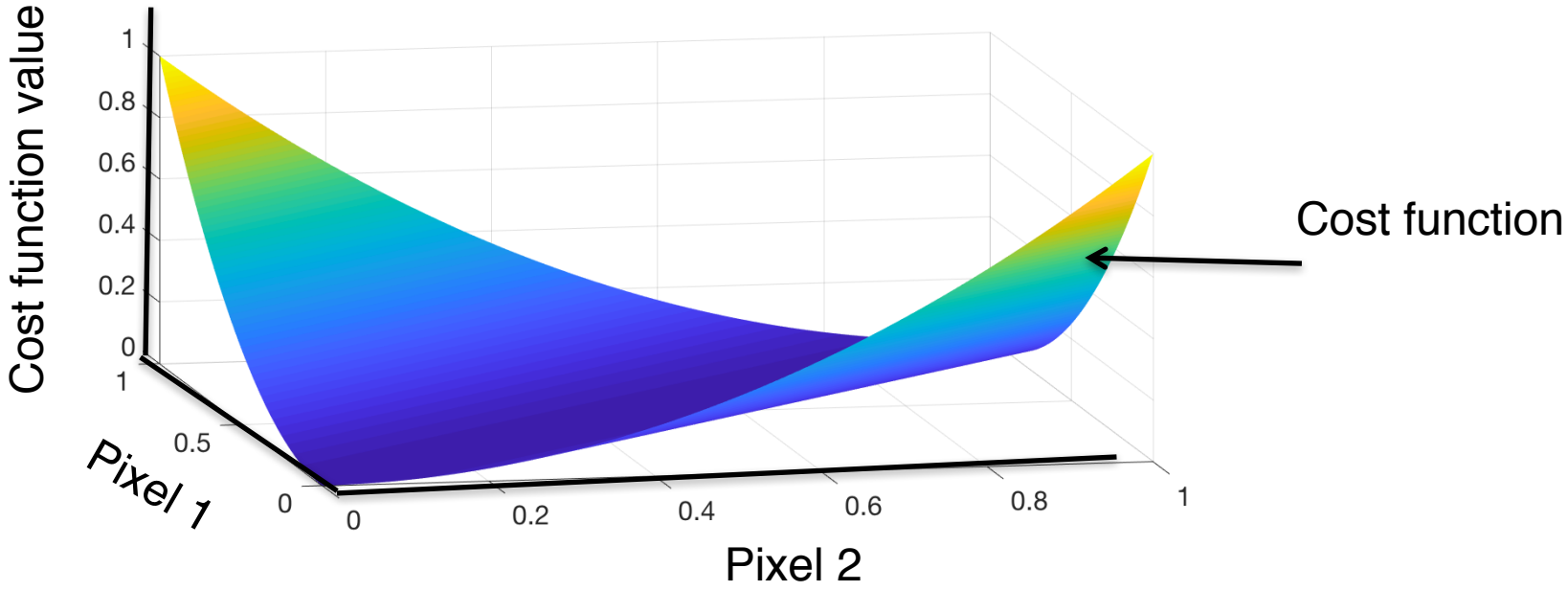
De-noising: "What is the closest image to what I detected, except without so many fluctuations"?

Cost function: "Don't let nearby pixels vary around too much"



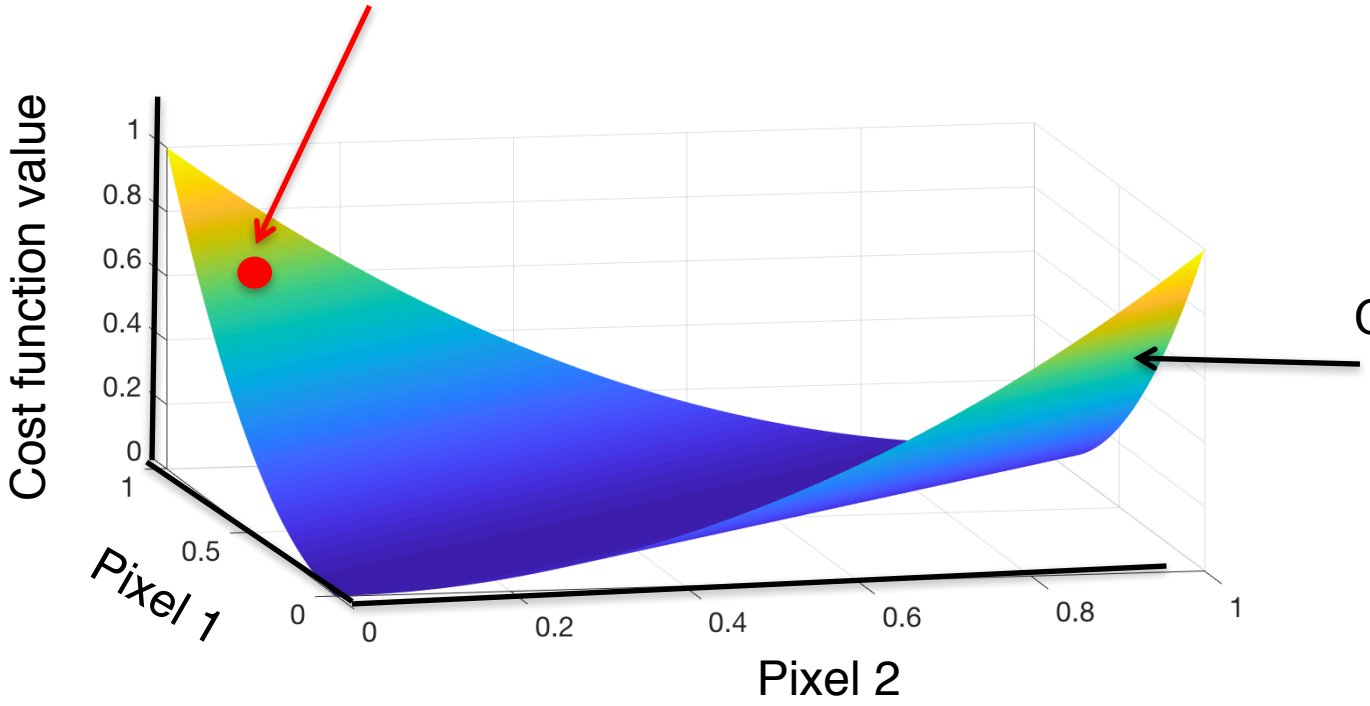

(Only showing 2 of the $N^2$ dimensions)

De-noising: "What is the closest image to what I detected, except without so many fluctuations"?




De-noising: "What is the closest image to what I detected, except without so many fluctuations"?

Start with a guess




Cost function

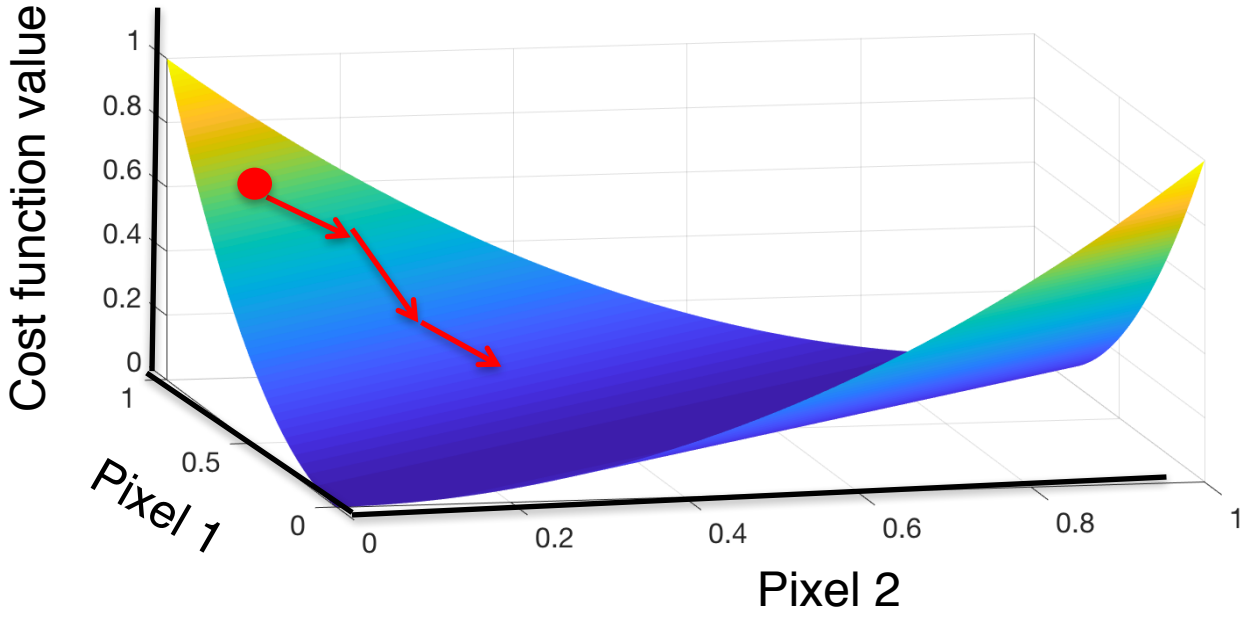
Noisy image

Pixel value 1

Pixel value 2

⋮

"Descend" to minimize cost function
(tweak values of each pixel)




Note: This part computers are really good at!

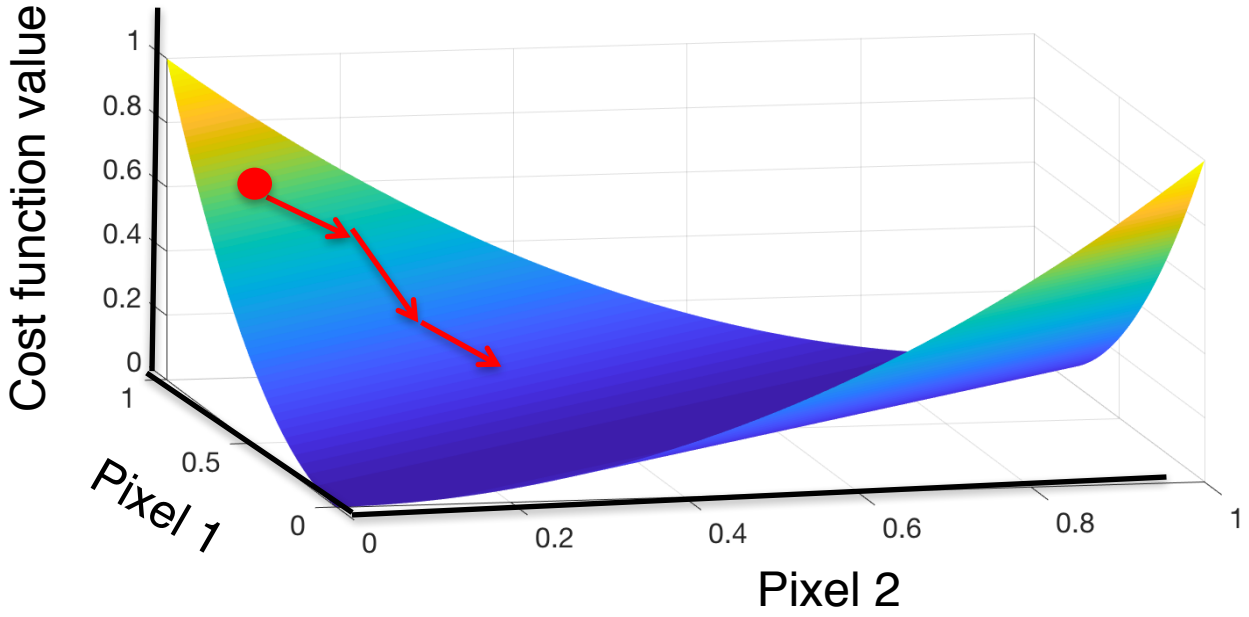
Noisy image

Pixel value 1

Pixel value 2

⋮

"Descend" to minimize cost function
(tweak values of each pixel)




Note: This part computers are really good at!

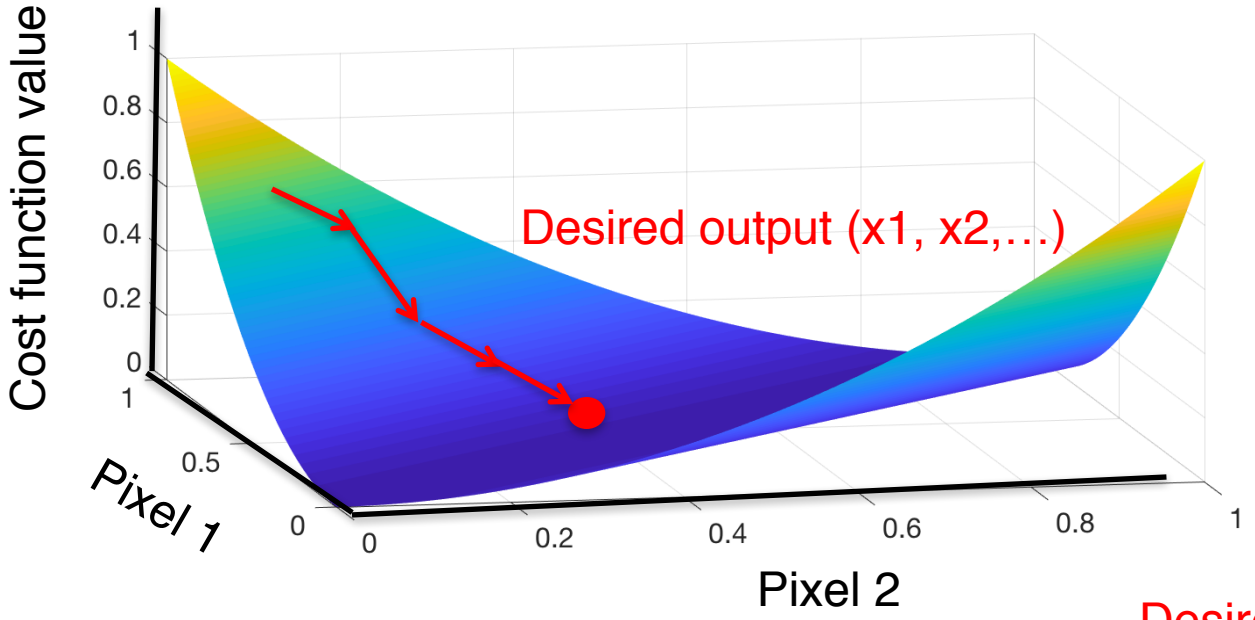
Noisy image

Pixel value 1

Pixel value 2

⋮

"Descend" to minimize cost function
(tweak values of each pixel)




Note: This part computers are really good at!

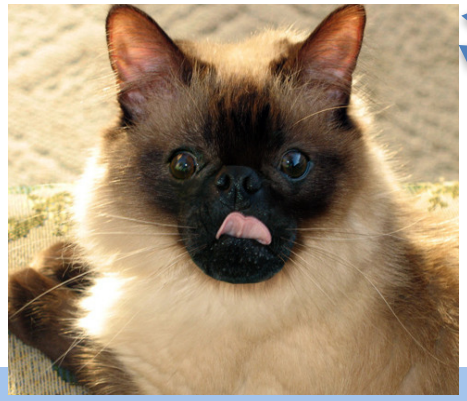
Noisy image

Pixel value 1

Pixel value 2

⋮

Desired output

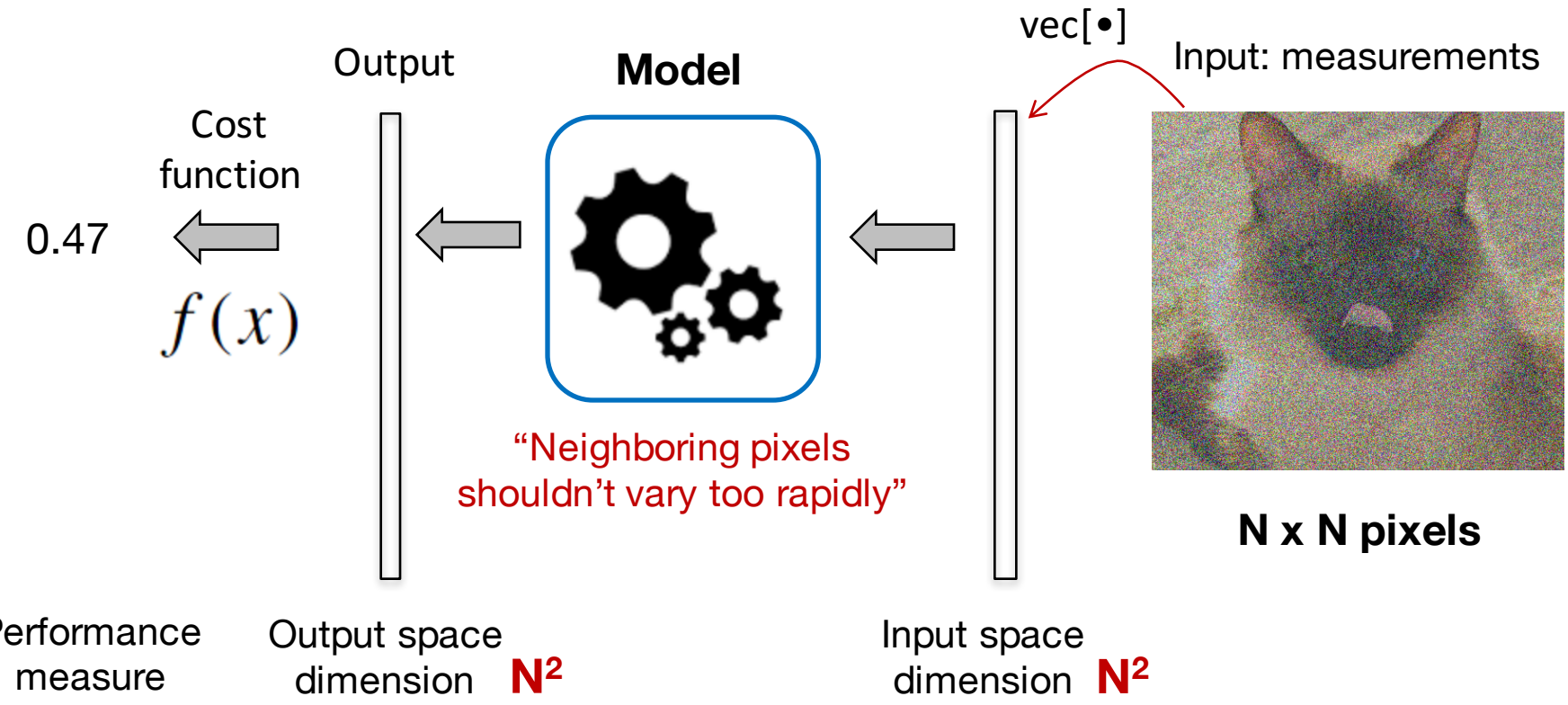


x1

x2

⋮

# Optimization pipeline for denoising



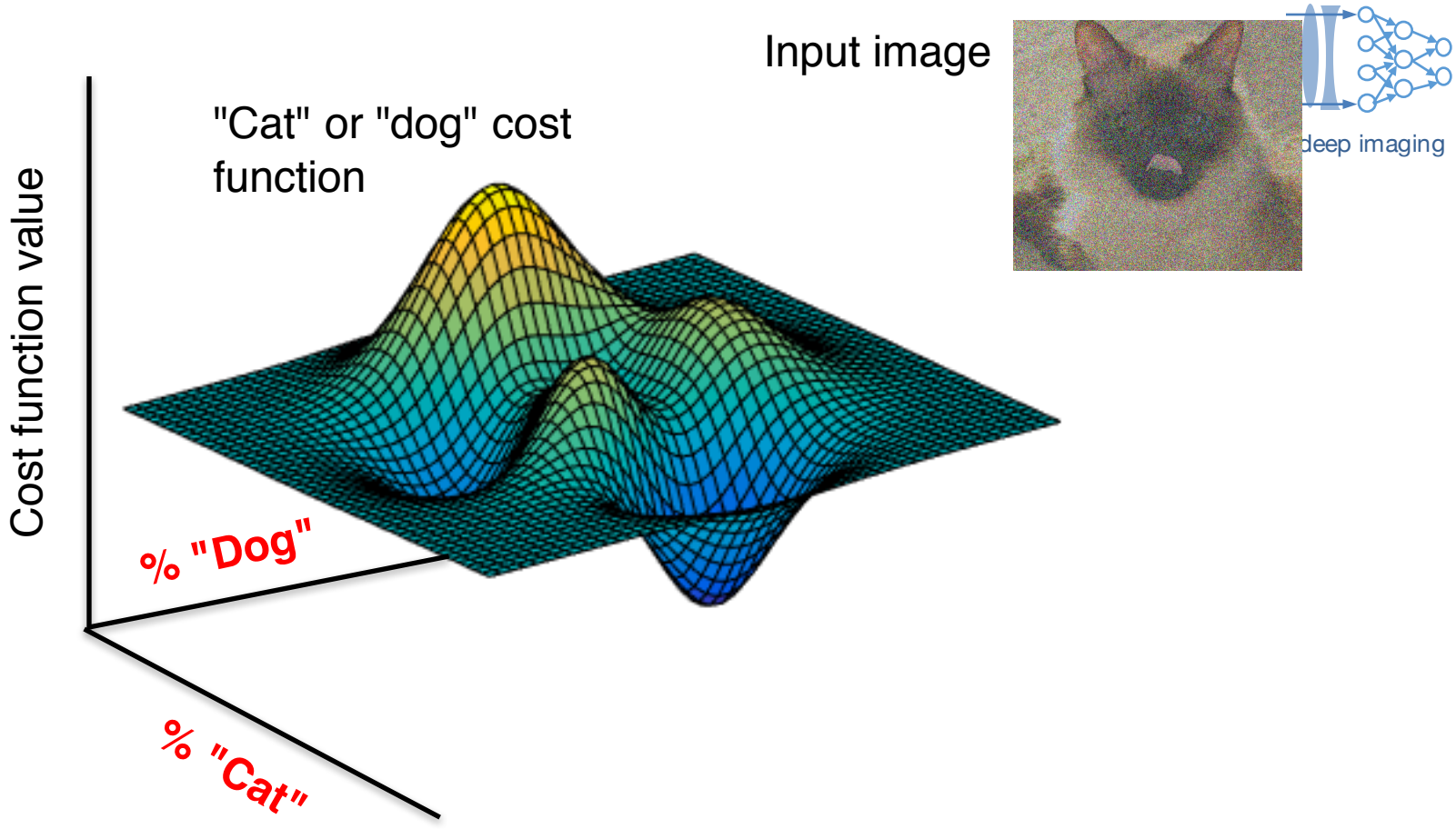
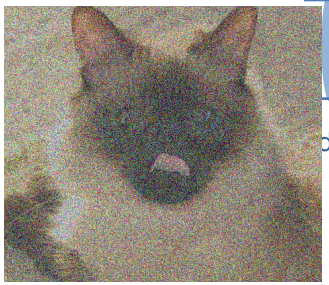


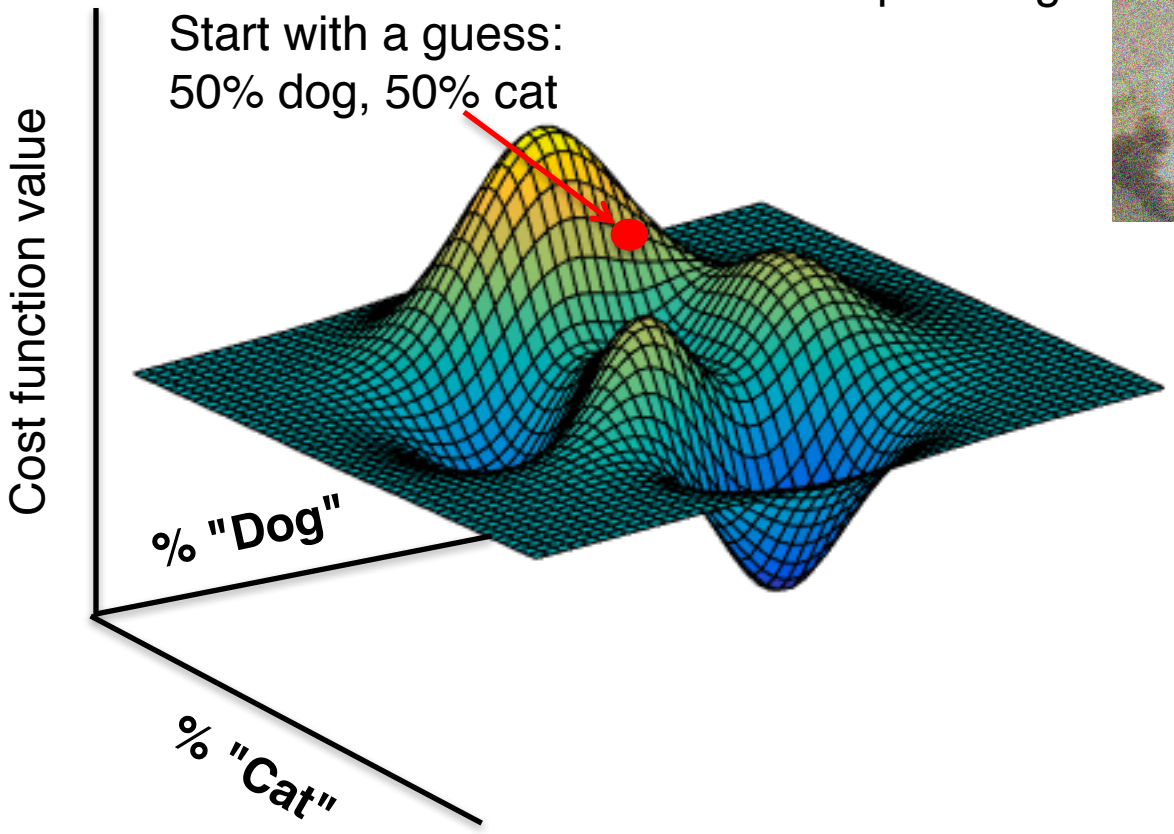

Image Classification: "Is the image of a dog or a cat"?

Input image

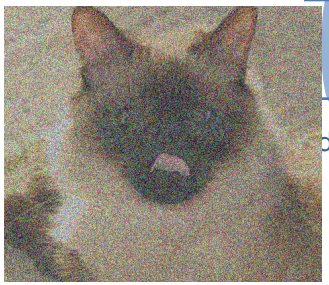


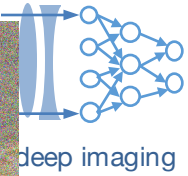
deep imaging

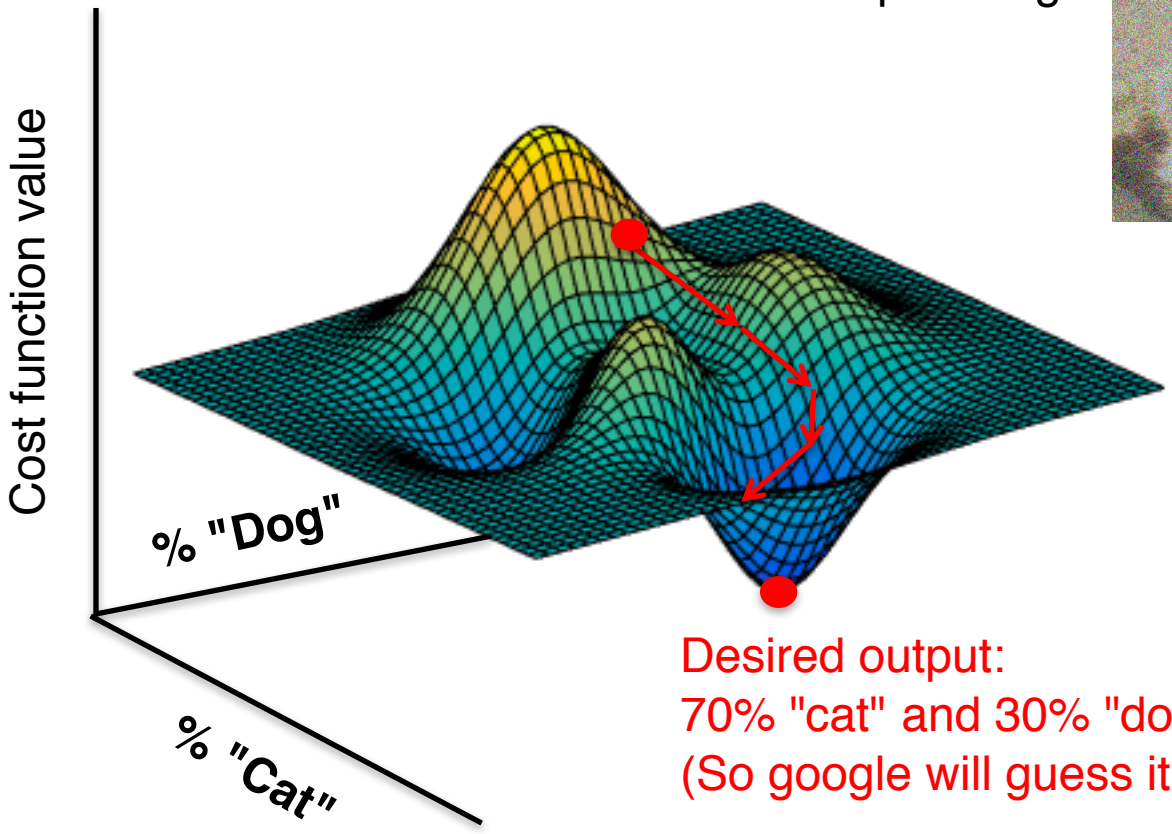
Start with a guess:
50% dog, 50% cat



Input image



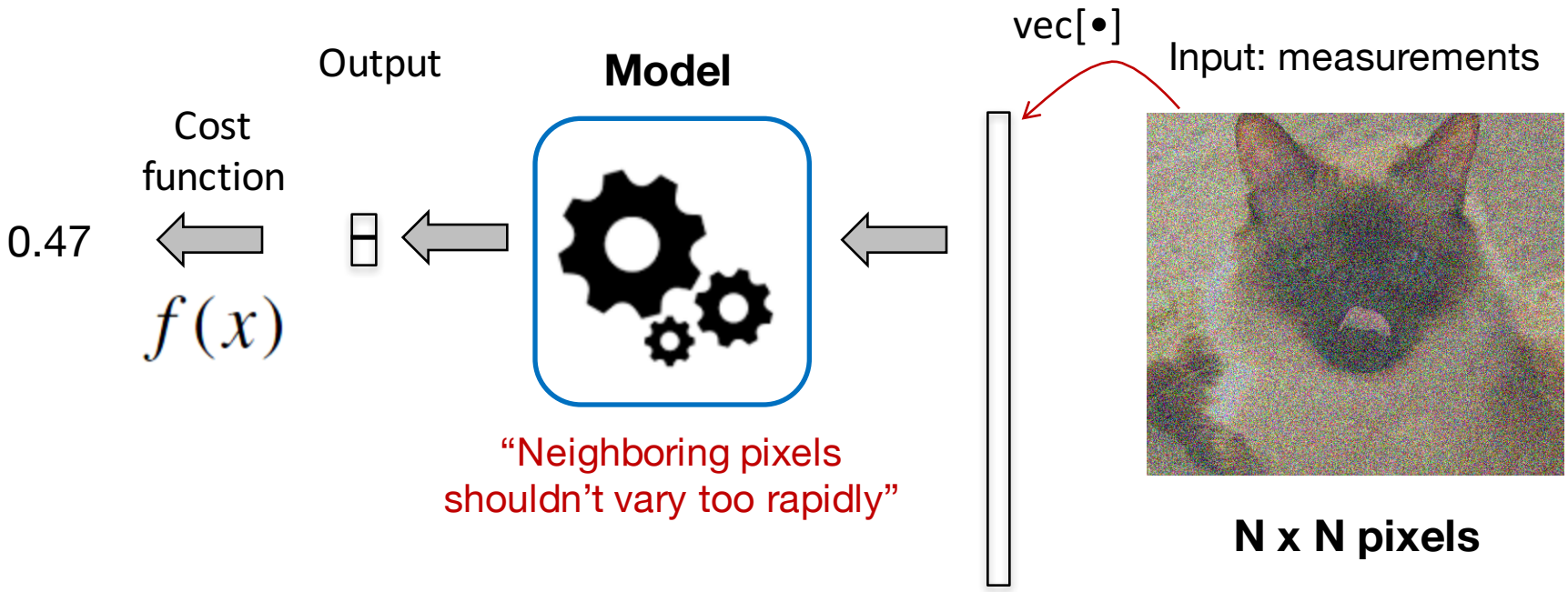



Desired output:
70% "cat" and 30% "dog"
(So google will guess it is likely a cat)

# Optimization pipeline for classification

deep imaging



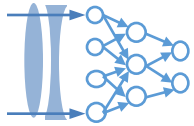

Performance measure

Mean-squared error

Output space dimension **2**

Input space dimension **$N^2$**

# A simple example: spectral unmixing




(For whatever reason, whenever I get confused about optimization, I think about this example...it's a good one)

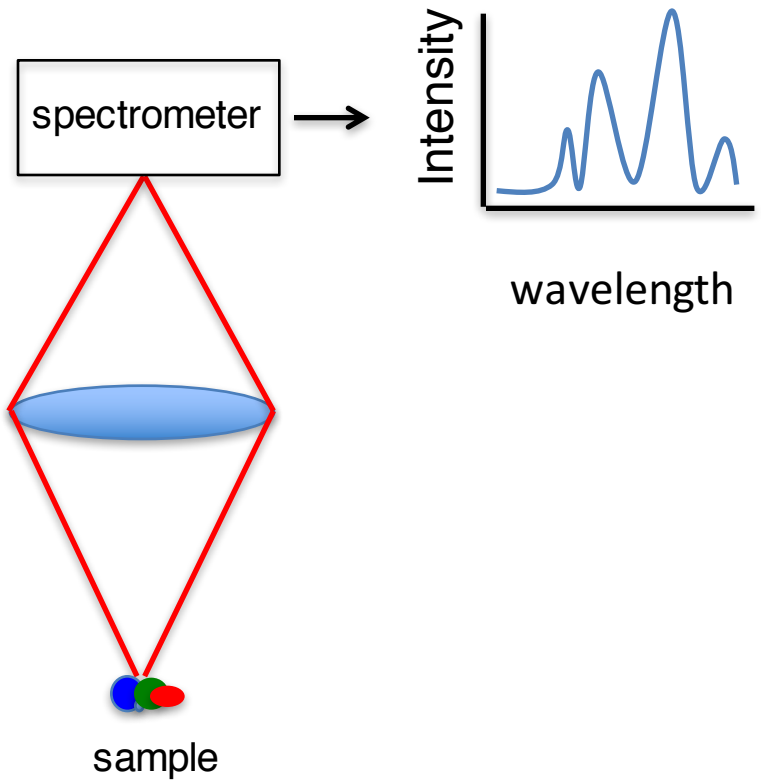
# A simple example: spectral unmixing

deep imaging

(For whatever reason, whenever I get confused about optimization, I think about this example...it's a good one)

**The setup:**

- measure the color (spectral) response of a sample (e.g., how much red, green and blue there is, or several hundred measurements of its different colors).
- You know that the sample can only contain 9 different fluorophores.
- What % of each fluorophores is in your sample?



# A simple example: spectral unmixing

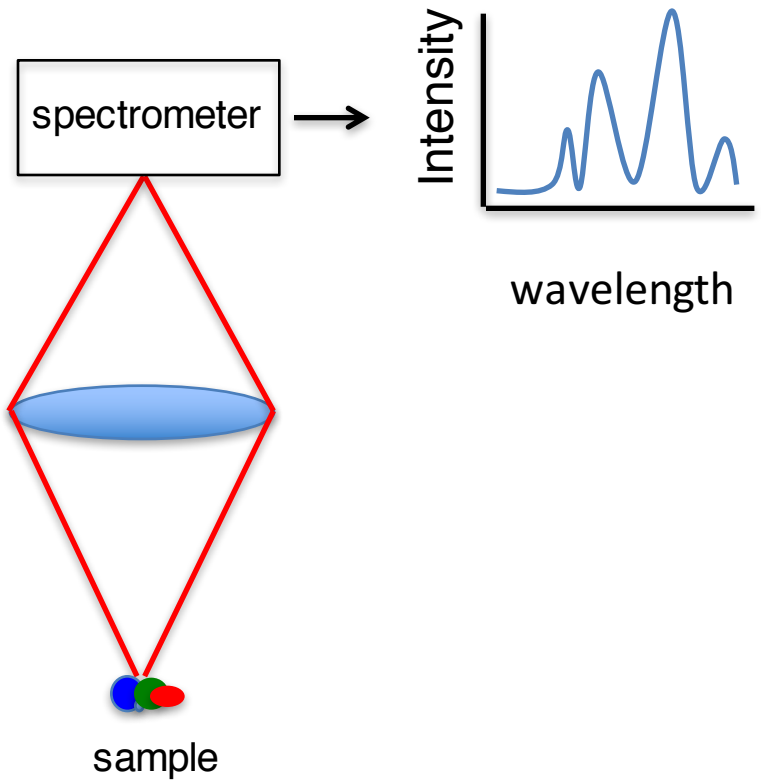
deep imaging

**3 elements of optimization:**

1) Desired output

2) The model

3) The cost function



# A simple example: spectral unmixing

deep imaging

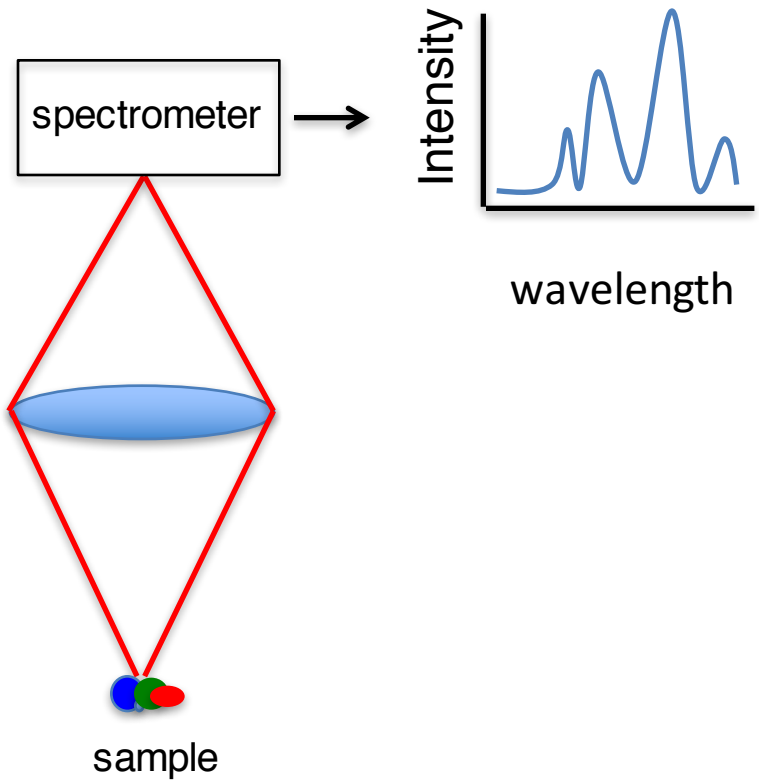
**3 elements of optimization:**

1) Desired output

What % of each of the 9 fluorophores

2) The model

3) The cost function



# A simple example: spectral unmixing

deep imaging

**3 elements of optimization:**

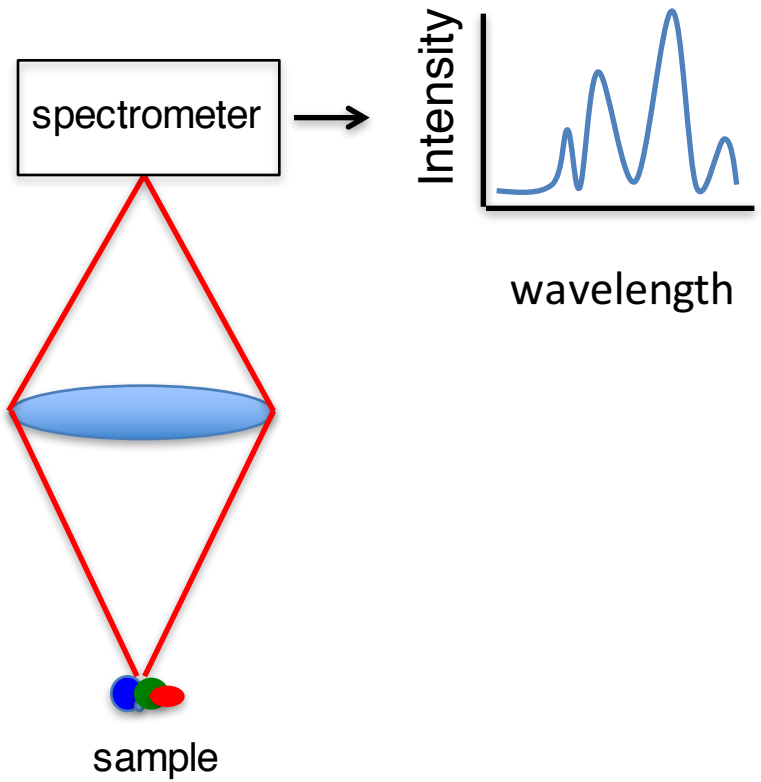
1) Desired output

What % of each of the 9 fluorophores

2) The model

"Dictionary" of the 9 different spectra

3) The cost function



# A simple example: spectral unmixing

deep imaging

**3 elements of optimization:**

1) Desired output

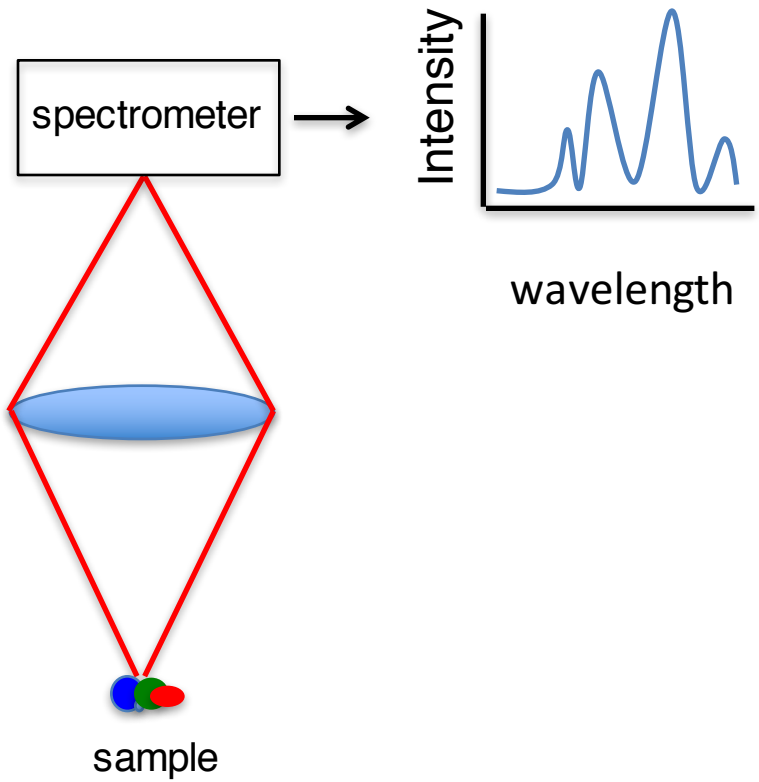
What % of each of the 9 fluorophores

2) The model

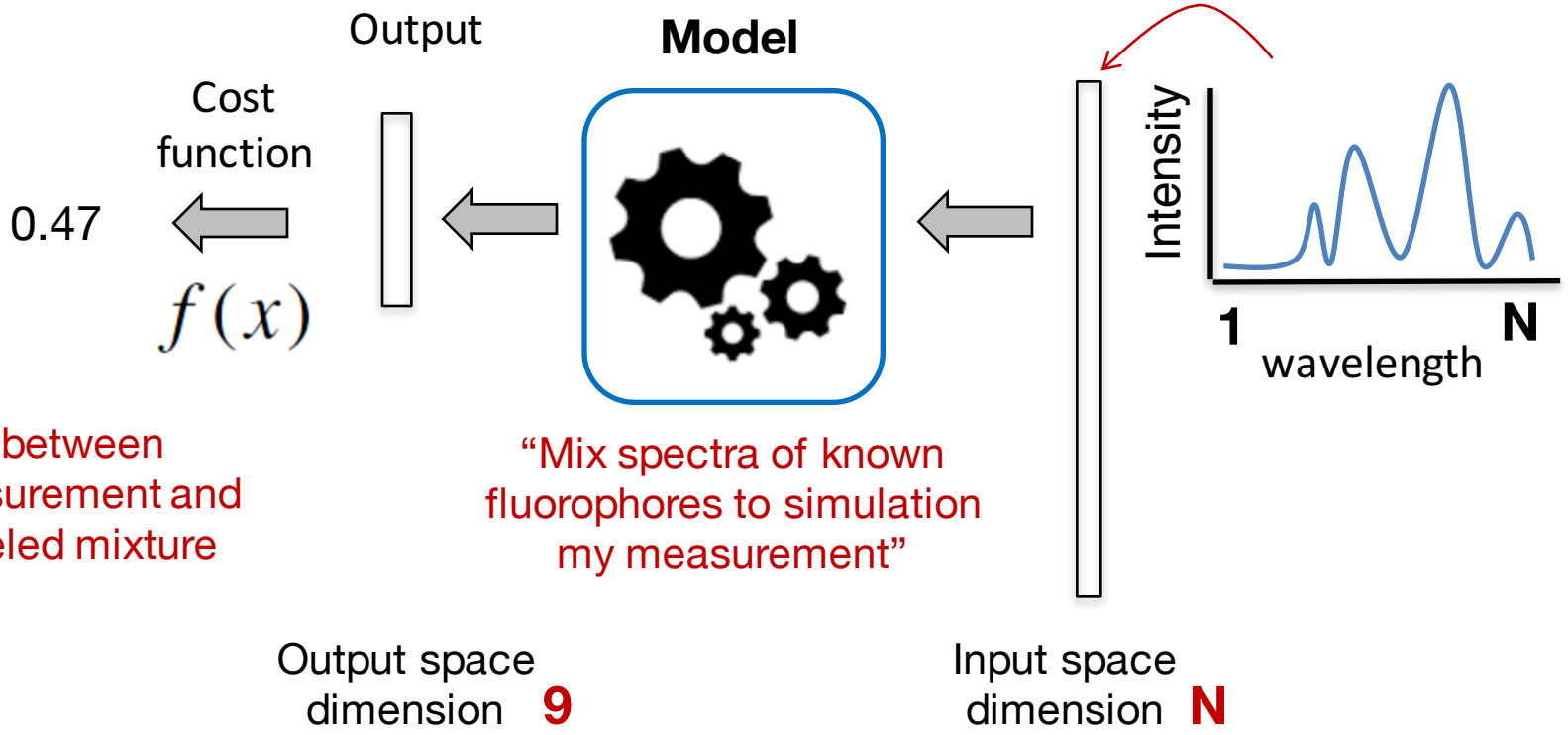
"Dictionary" of the 9 different spectra

3) The cost function

Minimum mean squared error (to start)



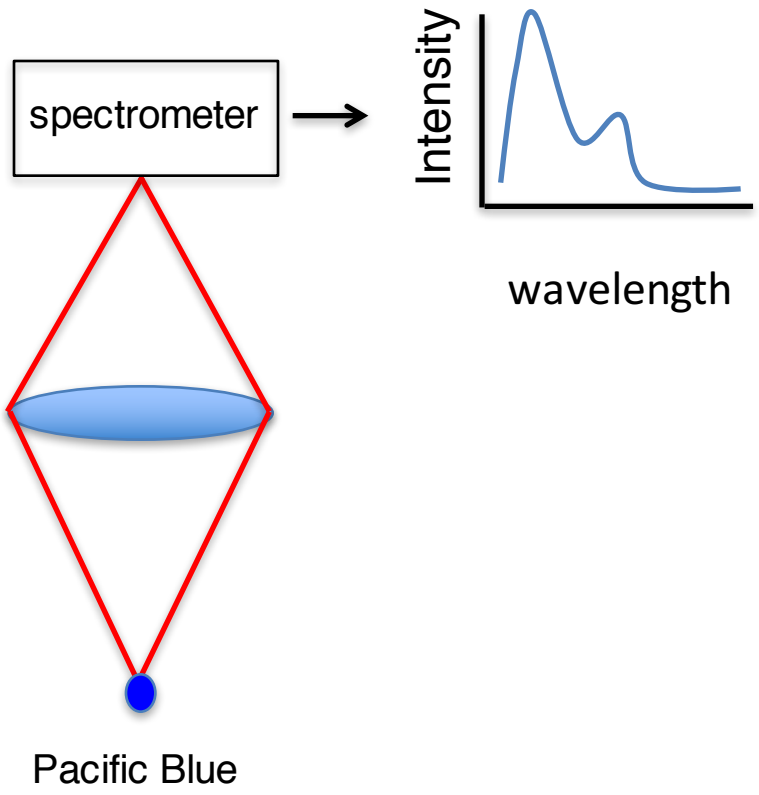
# Optimization pipeline for spectral unmixing



# Mathematical model for spectral unmixing

deep imaging

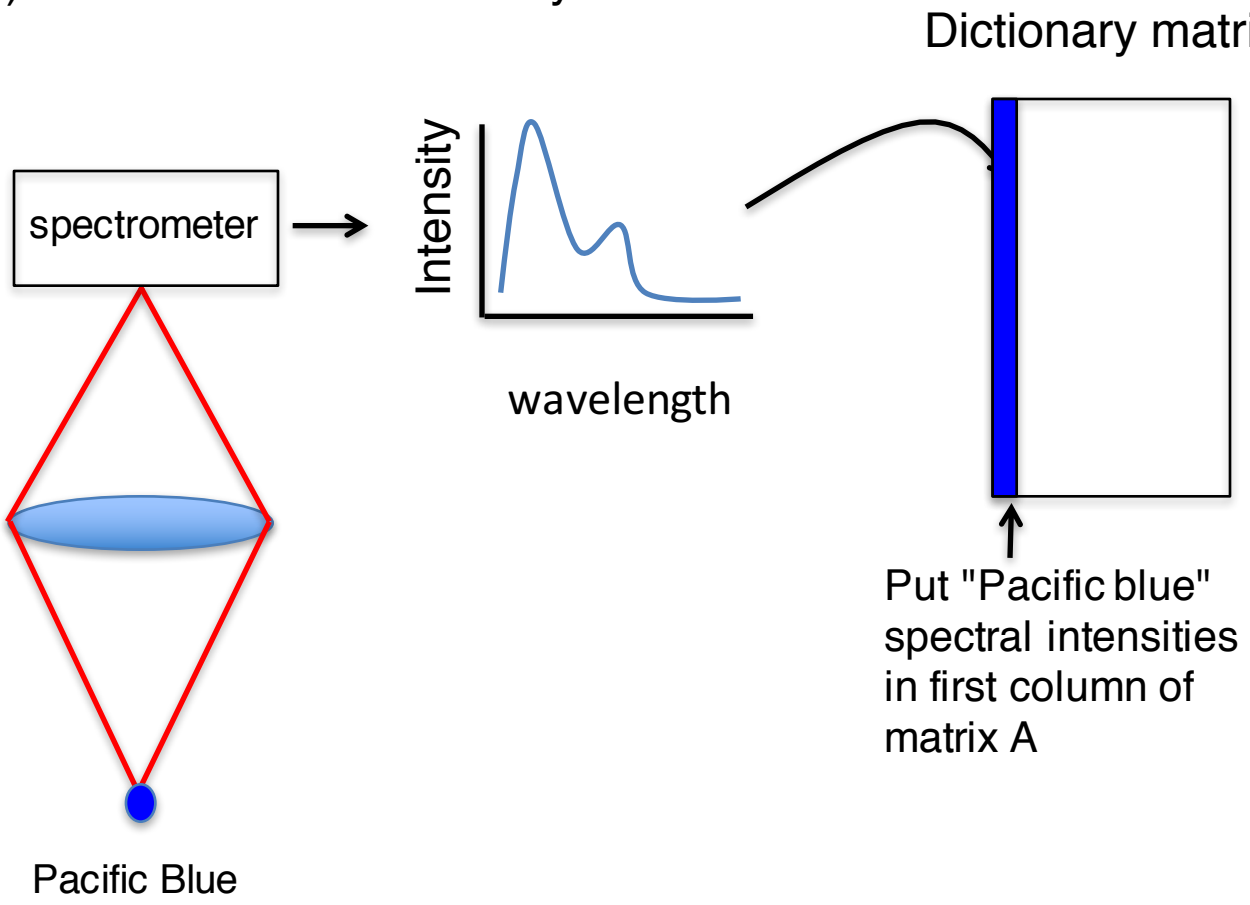
a) First make the "dictionary":



# Mathematical model for spectral unmixing

deep imaging

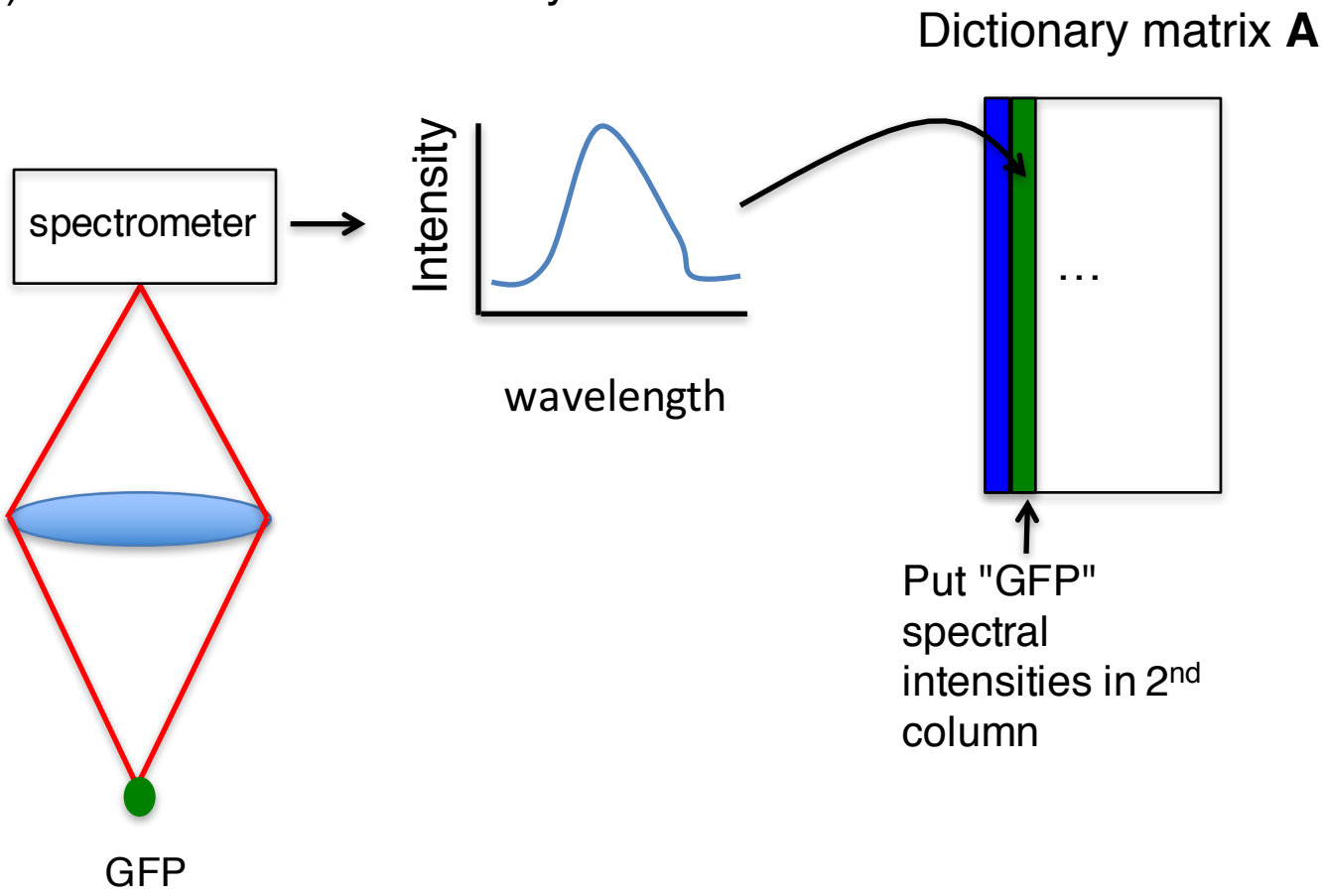
a) First make the "dictionary":



# Mathematical model for spectral unmixing

deep imaging

a) First make the "dictionary":

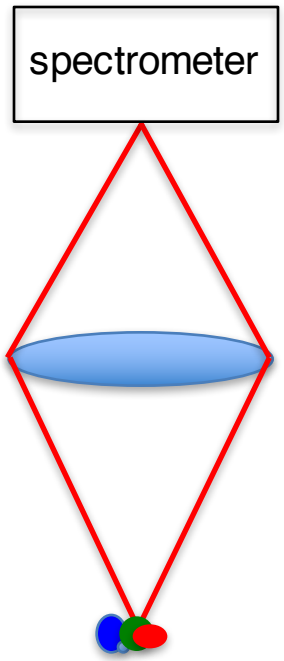


# Mathematical model for spectral unmixing

deep imaging

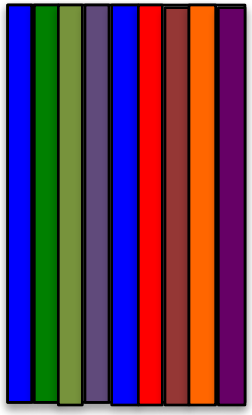
b) Model the unknown sample %'s

(the desired output)



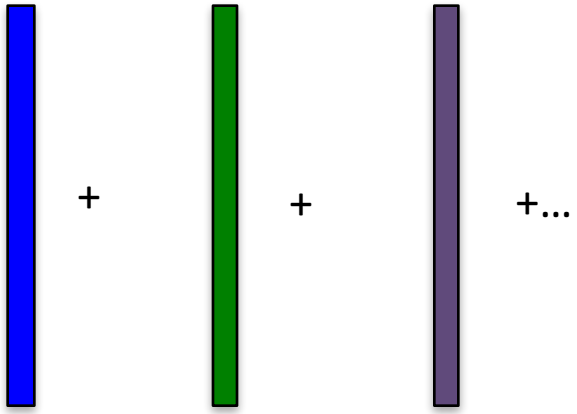

**Unknown sample y**

Dictionary matrix **A**



9 possible spectra

Some mixture…

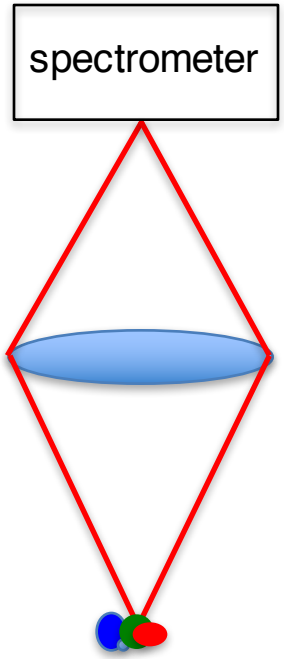


# Mathematical model for spectral unmixing

deep imaging

b) Model the unknown sample %'s

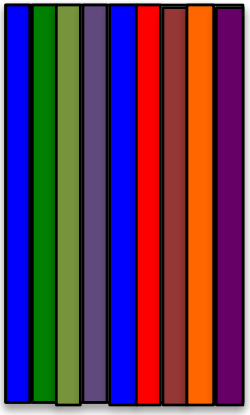
(the desired output)



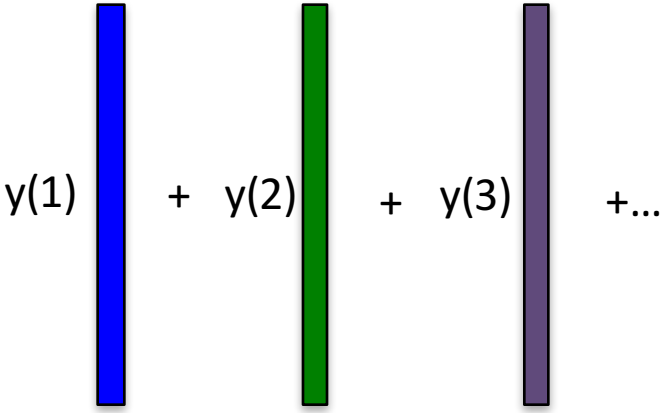

**Unknown sample y**

Each weight in x is percentage:

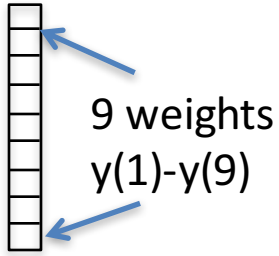
Dictionary matrix **A**



9 possible spectra




Unknown sample %'s **y**

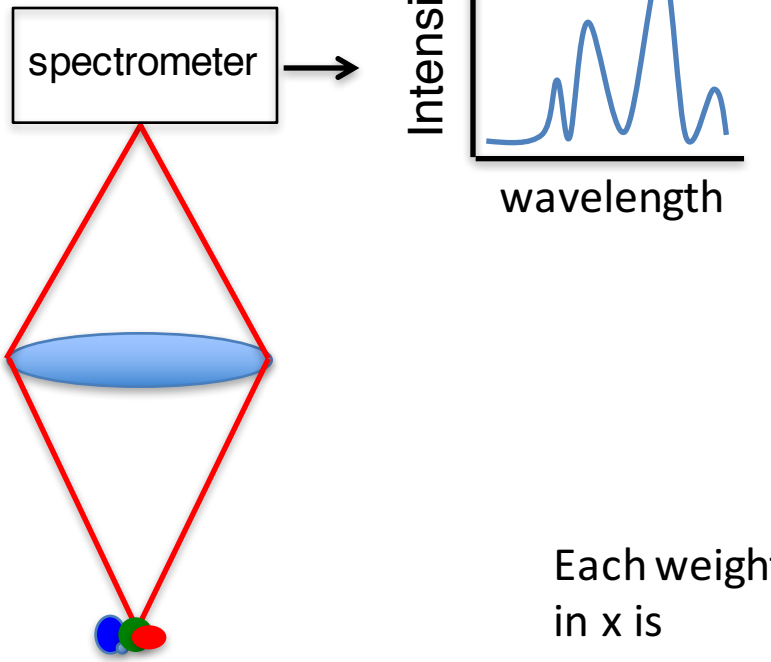


# Mathematical model for spectral unmixing

deep imaging

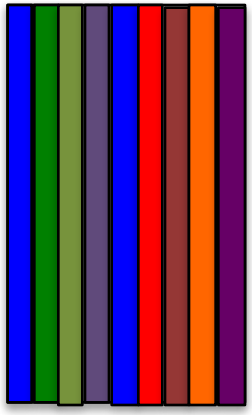
b) Model the unknown sample %'s

**New measurement x**



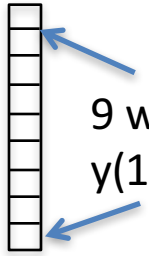

**Unknown sample y**

Dictionary matrix **A**

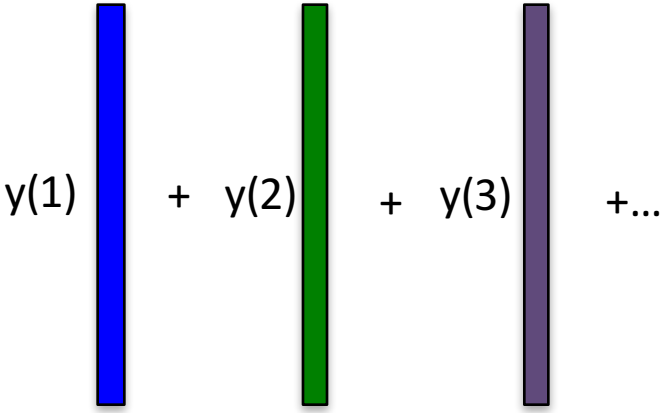


9 possible spectra

Unknown sample %'s **y**

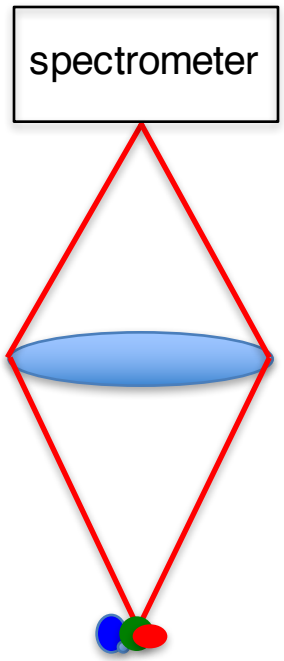



Each weight in x is percentage:

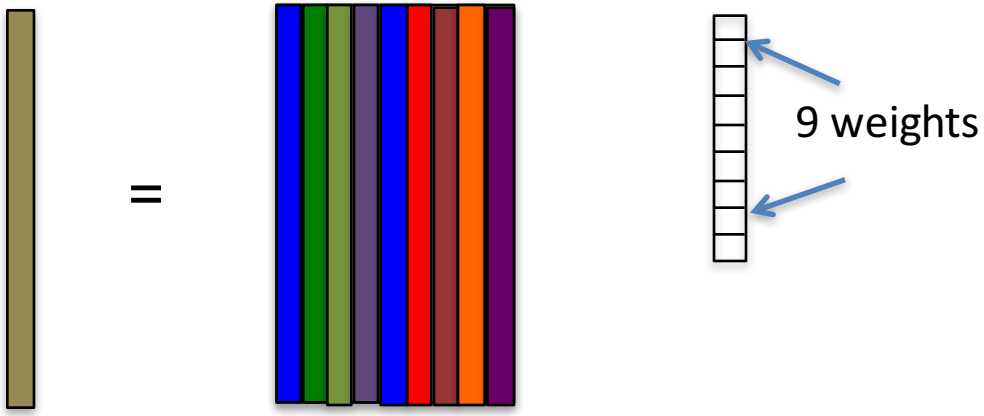


# Mathematical model for spectral unmixing

deep imaging




**Unknown sample y**



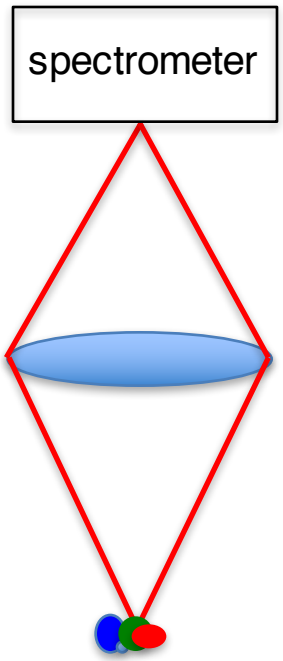

Matrix equation: **x=Ay**

This is your model!

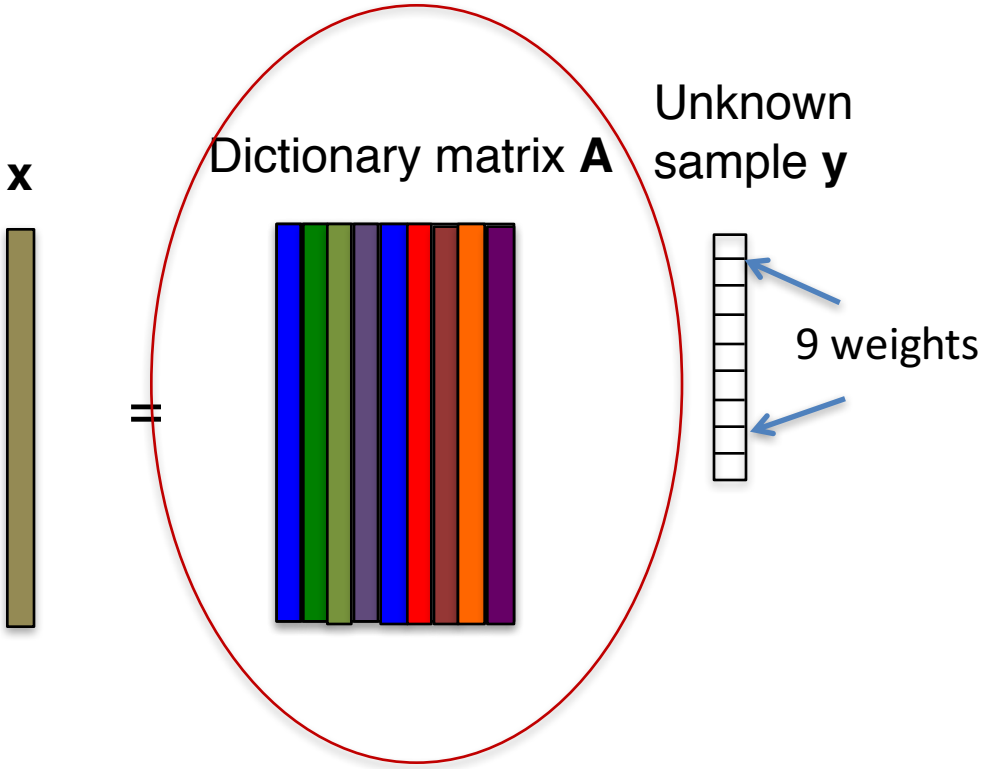
**Goal: Given A and x, find y**

# Mathematical model for spectral unmixing

deep imaging

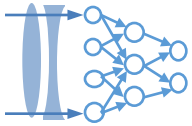



**Unknown sample y**




Data in A can be thought of , in some sense, as "training data"

# Cost function for spectral unmixing

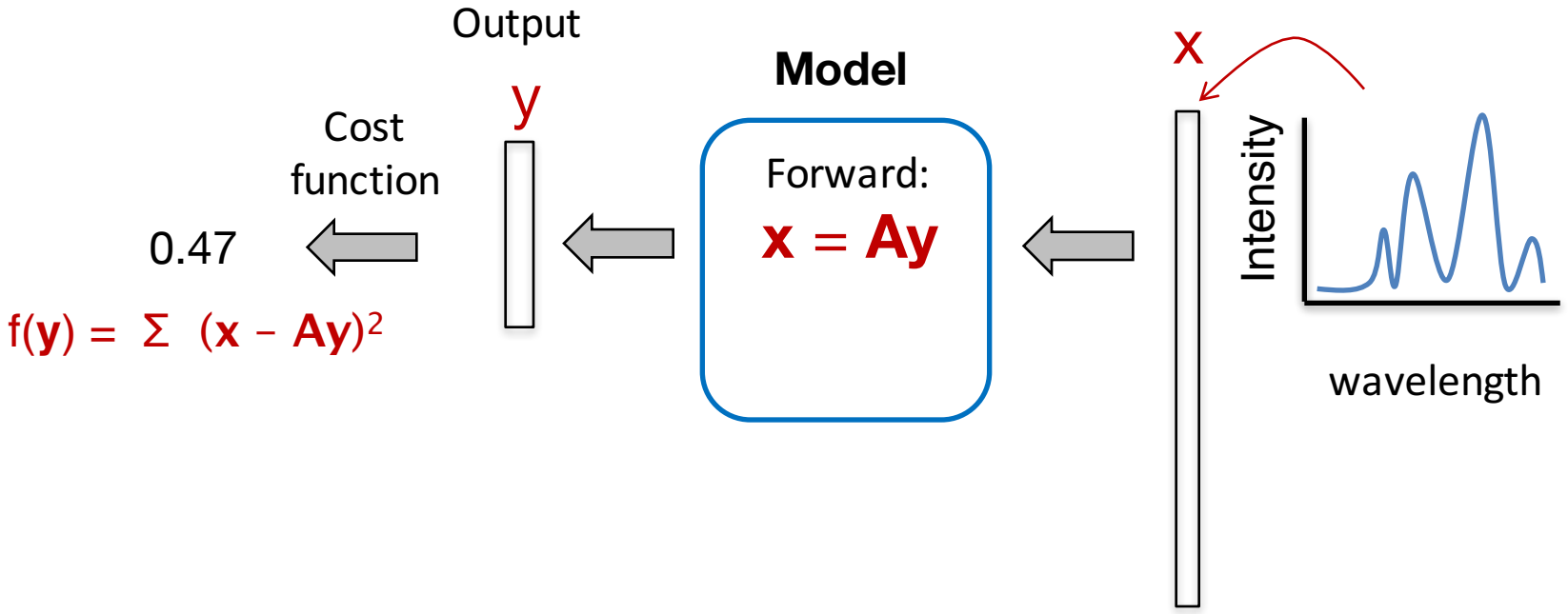



**x** = **Ay** won't always be true, due to noise (actually, **x** = **Ay**+**n**)

Common cost function is minimum mean-squared error:

$$\text{Cost function } f(\mathbf{y}) = \sum_{\substack{\text{spectral} \\ \text{measurements}}} (\mathbf{x} - \mathbf{Ay})^2$$

# Optimization pipeline for denoising



# Cost function for spectral unmixing

deep imaging

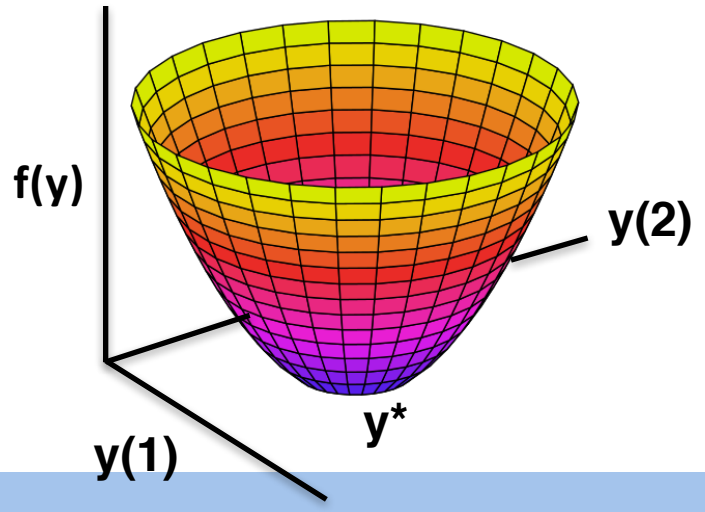
**x** = **Ay** won't always be true, due to noise (actually, **x** = **Ay**+**n**)

Common cost function is minimum mean-squared error:

$$\text{Cost function } f(\mathbf{y}) = \sum_{\text{spectral measurements}} (\mathbf{x} - \mathbf{Ay})^2$$

Find mixture **y** of known spectra **A** that is as close as possible to measurement **x**

$$\mathbf{y}^* = \text{minimize } f(\mathbf{y})$$

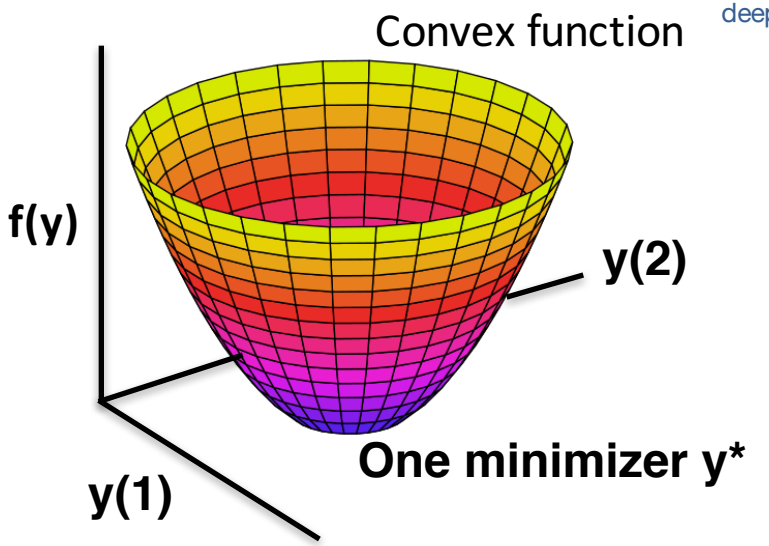


# Cost function for spectral unmixing

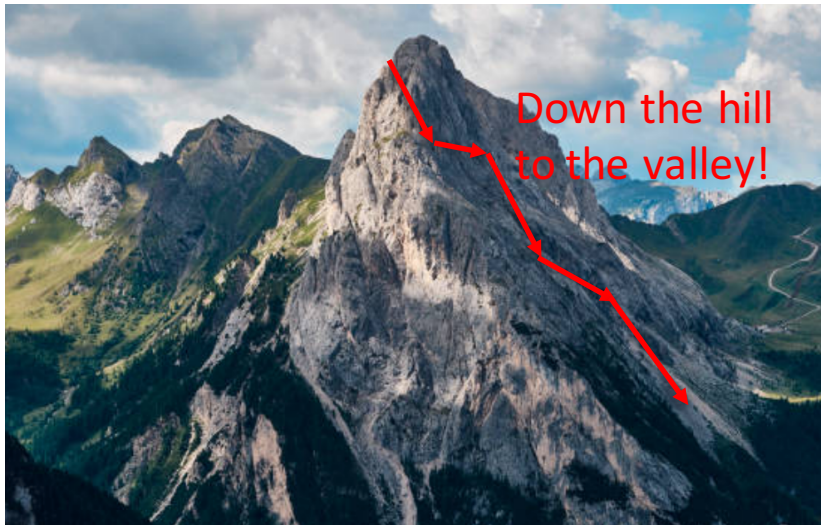
deep imaging

$$f(\mathbf{y}) = \sum (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




f(**y**) is convex, so finding **y*** is easy via its gradient:

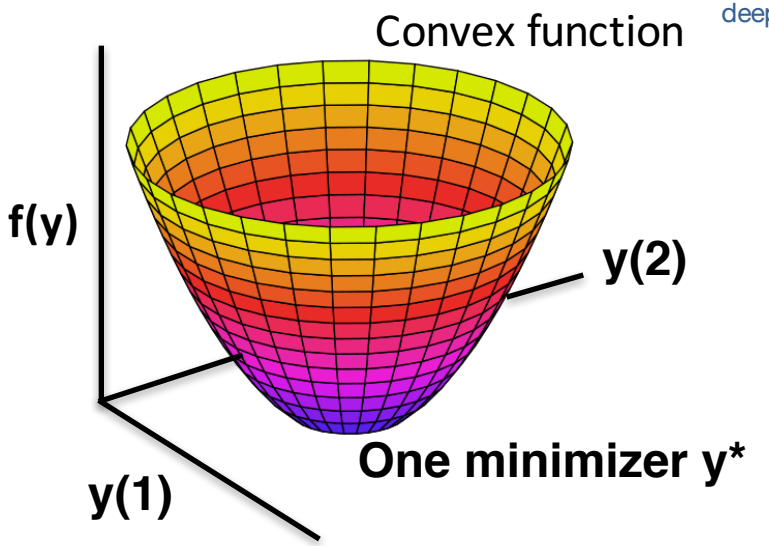


# Cost function for spectral unmixing

deep imaging

$$f(\mathbf{y}) = \sum (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




f(**x**) is convex, so finding **x*** is easy via its gradient:

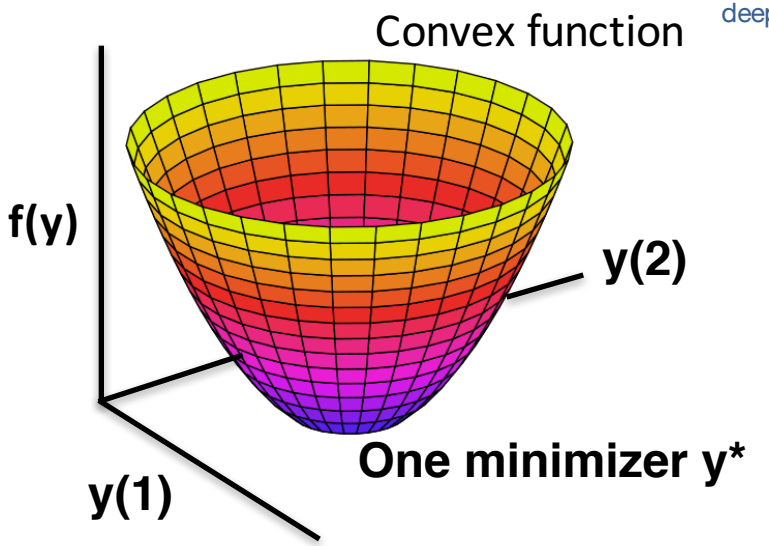
$$d/d\mathbf{y}\, f(\mathbf{y}) = d/d\mathbf{y} \sum (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

# Cost function for spectral unmixing

deep imaging

$$f(\mathbf{y}) = \Sigma \ (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




f(**x**) is convex, so finding **x*** is easy via its gradient:

$$d/d\mathbf{y}\ f(\mathbf{y}) = d/d\mathbf{y}\ \Sigma\ (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

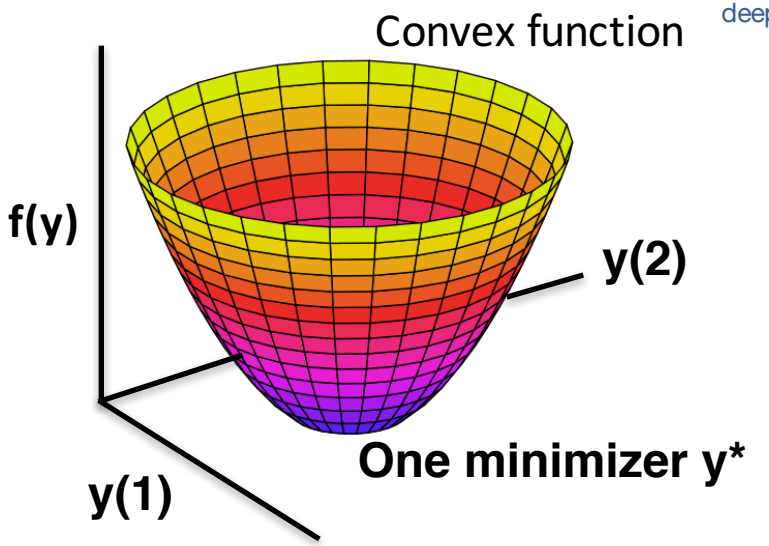
$$df/d\mathbf{y} = \Sigma\ d/d\mathbf{y}\ (\ \mathbf{x} - \mathbf{A}\mathbf{y})^2$$

# Cost function for spectral unmixing

deep imaging

$$f(\mathbf{y}) = \sum (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




f(**x**) is convex, so finding **x*** is easy via its gradient:

$$d/d\mathbf{y}\ f(\mathbf{y}) = d/d\mathbf{y}\ \sum (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

$$df/d\mathbf{y} = \sum d/d\mathbf{y}\ (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

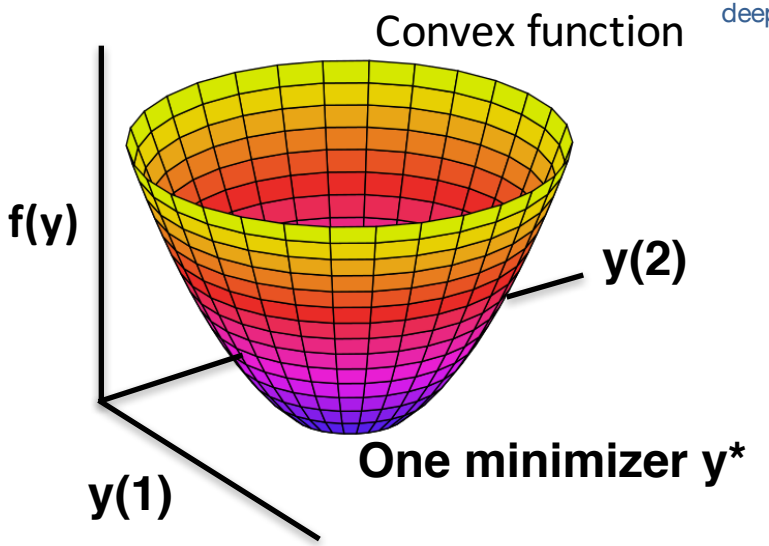
$$df/dy[j] = \sum 2\ (\mathbf{x} - \mathbf{A}\mathbf{y}) .* \mathbf{a}(:,j)$$

# Cost function for spectral unmixing

deep imaging

$$f(\mathbf{y}) = \sum (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




f(**x**) is convex, so finding **x*** is easy via its gradient:

$$d/d\mathbf{y}\ f(\mathbf{y}) = d/d\mathbf{y}\ \sum (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

$$df/d\mathbf{y} = \sum d/d\mathbf{y}\ (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

$$df/dy[j] = \sum 2\ (\mathbf{x} - \mathbf{A}\mathbf{y}) \cdot^* \mathbf{a}(:,j)$$

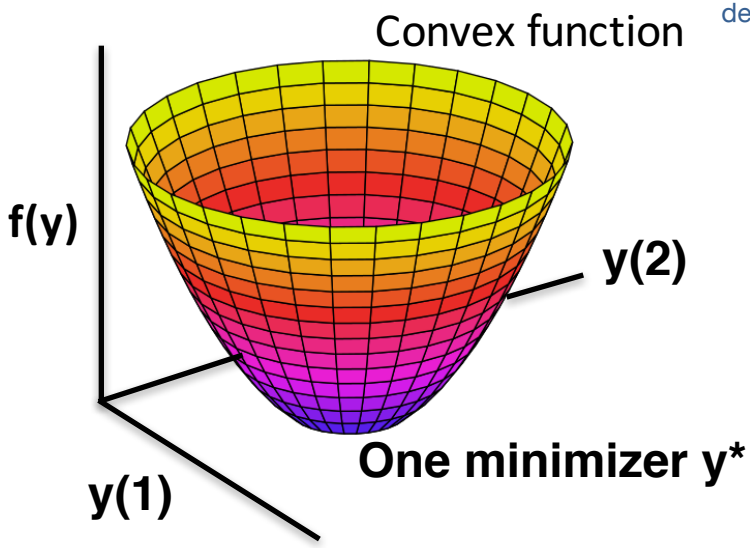
$$df/d\mathbf{y} = \mathbf{A}^T(\mathbf{x} - \mathbf{A}\mathbf{y}^*)$$

# Cost function for spectral unmixing

deep imaging

$$f(\mathbf{y}) = \Sigma \ (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




Method 1: *Gradient descent* – follow gradient downhill to solution **y***

**Algorithm 4.1** An algorithm to minimize $f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$ with respect to $\boldsymbol{x}$ using gradient descent, starting from an arbitrary value of $\boldsymbol{x}$.

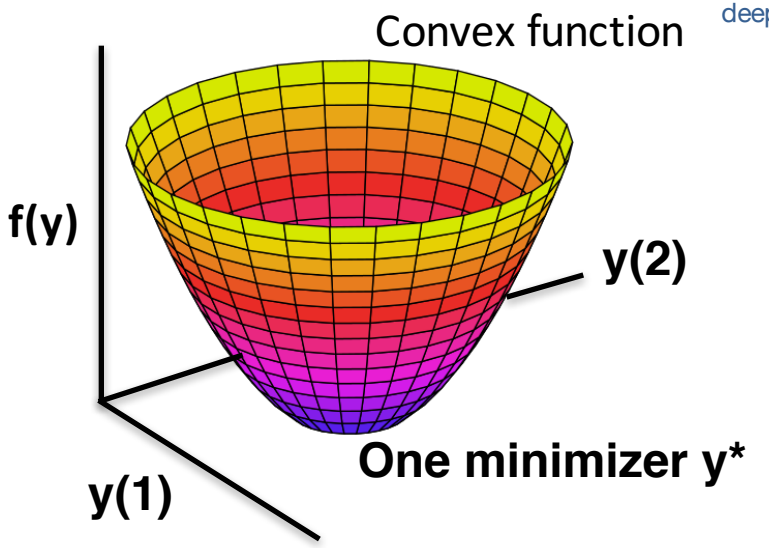
Set the step size ($\epsilon$) and tolerance ($\delta$) to small, positive numbers.
**while** $||\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}||_2 > \delta$ **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{x} - \epsilon \left(\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}\right)$
**end while**

# Cost function for spectral unmixing

deep imaging

$$f(\mathbf{y}) = \Sigma \; (x - \mathbf{A}\mathbf{y})^2$$

spectral measurements




Method 2: *Direct solution* – set derivative to 0 to find **y*** directly

$$df/d\mathbf{y} = \mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{y}^*) = 0$$ ← y* is where gradient of f(y) is zero

$$\mathbf{A}^T\mathbf{x} = \mathbf{A}^T\mathbf{A}\mathbf{y}^* \longrightarrow$$

$$\boxed{(\mathbf{A}^T\mathbf{A})^{-1}\,\mathbf{A}^T\,\mathbf{x} = \mathbf{y}^*}$$ "Moore-Penrose Pseudo-inverse"

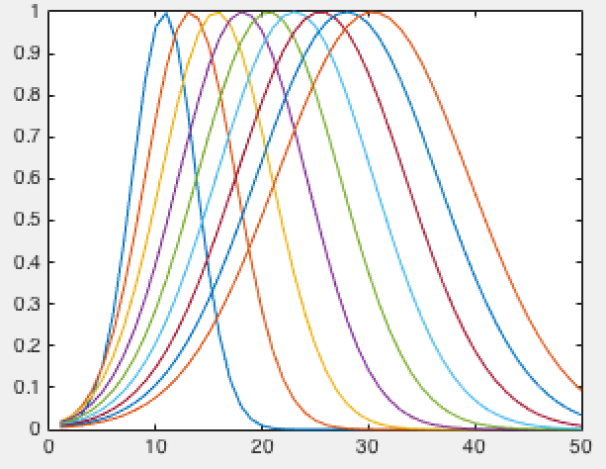
(Note: setting gradient to 0 and solving is hard to do for non-linear problems…)

# Example unmixing with the pseudo-inverse

deep imaging

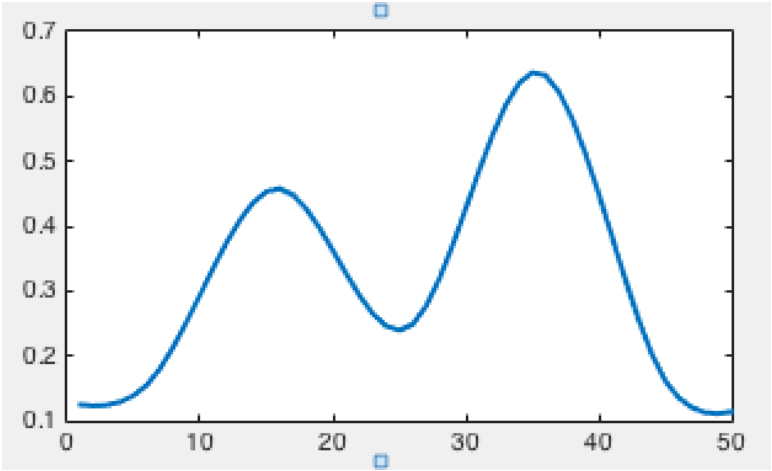
Moore-Penrose Pseudo-inverse: $y^* = (A^T A)^{-1} A^T x$

**Example dictionary A**
**9 spectra**



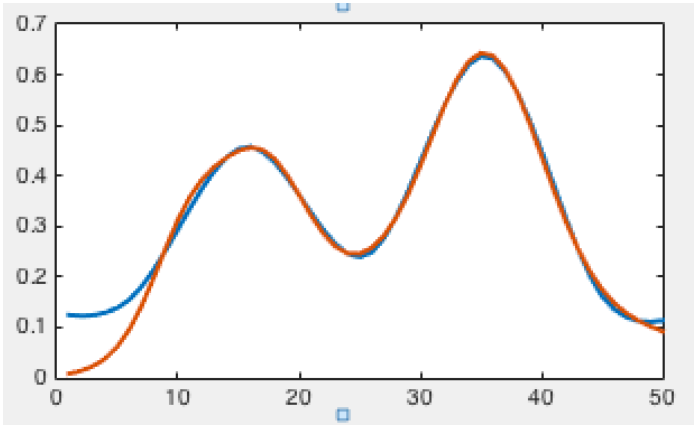

**Example y, compute Ay**

**Example detected spectra x**




**Compute pseudo-inverse, x*=Ay* is red curve:**

**Good fit!**

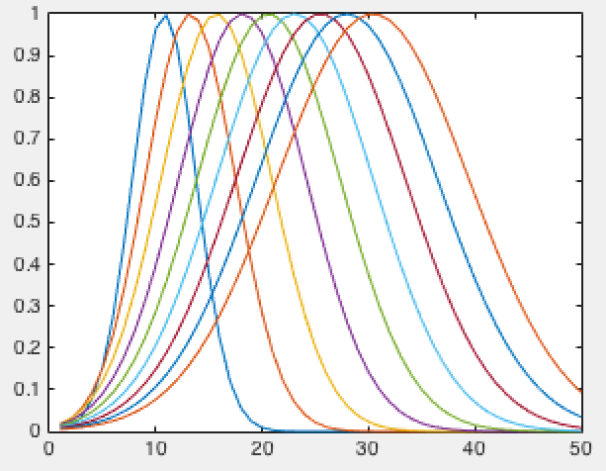


# Example unmixing with the pseudo-inverse

deep imaging

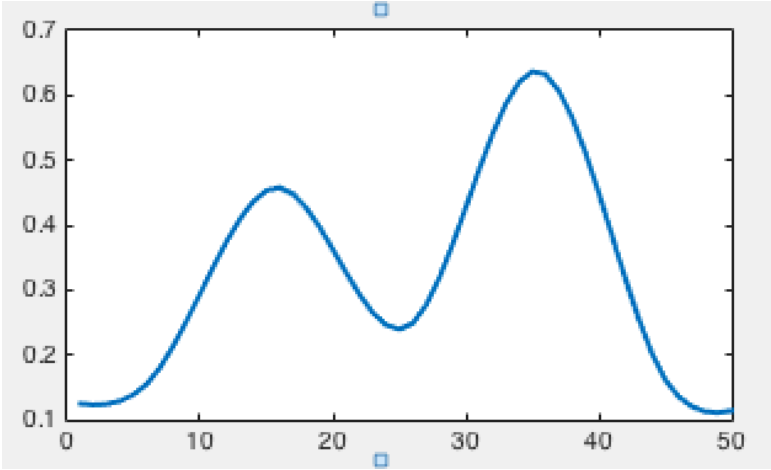
Moore-Penrose Pseudo-inverse: $y^* = (A^T A)^{-1} A^T x$

**Example dictionary A**
**9 spectra**



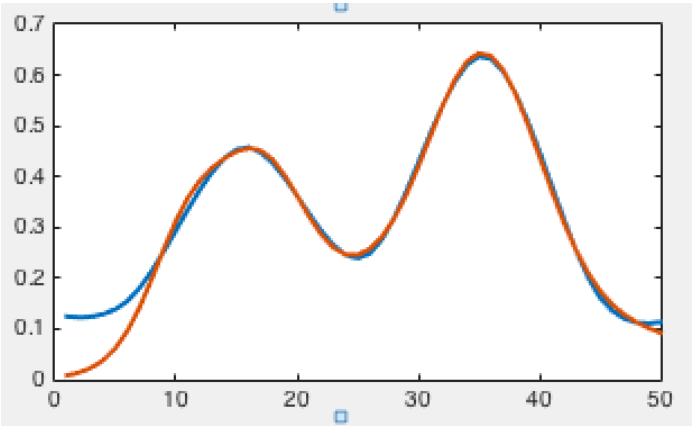
**Example y, compute Ay**

**Example detected spectra x**



**Compute pseudo-inverse, x*=Ay* is red curve:**

**Good fit!**



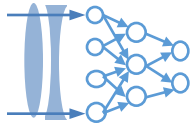
**PROBLEM:**

**y* = [0.2, -1.1, -1.6, …]**

**Solution has negative weights!**

**Not physically possible…**

# Example unmixing with the pseudo-inverse

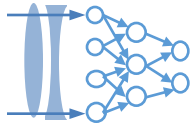



Moore-Penrose Pseudo-inverse: $\mathbf{y^* = (A^T A)^{-1} A^T x}$

```
n = 50; %number of pixels
m = 9; %number of spectral
A=zeros(n,m); %known dictionary of spectra
for j=1:m
    A(:,j) = exp(-(linspace(-1,1,n)+.5-.1*j+.2).^2/(.03*j));
end
%Simulate some spectra
b = imresize(rand([5,1]),[n 1]);
x_opt = A\b;
%Show results
figure;plot(b); hold all; plot(A*x_opt);
```

Pseudo-inverse = one line

# Spectral un-mixing with a positivity constraint




## Option 1: Add a constraint

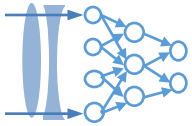
Minimize $f(\mathbf{x}) = \Sigma \ ( \mathbf{b} - \mathbf{A}\mathbf{x})^2$ Convex cost function

Subject to $\mathbf{x} >= 0$ Convex constraint

*When you have constraints, can use **CVX**, convex toolbox for Matlab

http://cvxr.com/cvx/

# Spectral un-mixing with a positivity constraint




## Option 1: Add a constraint

```
%%%%%%%%%%%%%
addpath '/users/Roarke/Documents/Matlab/cvx'; cvx_setup;
cvx_begin
    variable xc(m);
    minimize( norm(A*xc-b) );
    subject to
        xc >= 0;
cvx_end
%Show results
figure;plot(b); hold all; plot(A*xc);
%%%%%%%%%%%%%
```

# Spectral un-mixing with a positivity constraint

deep imaging

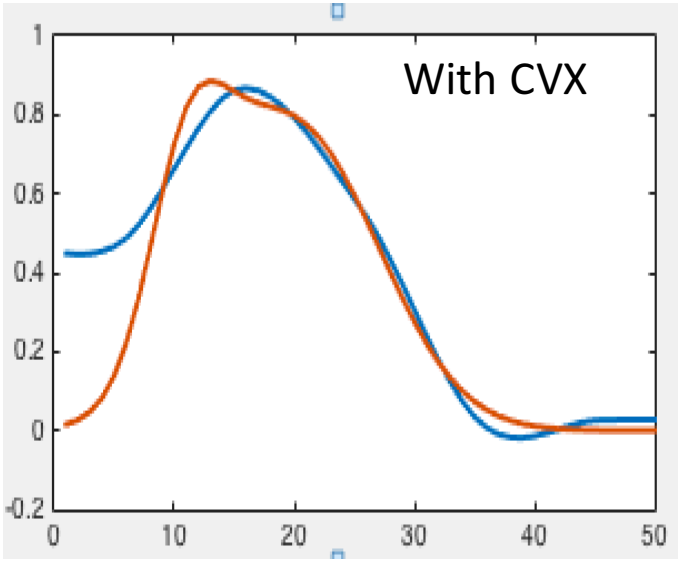
## Option 1: Add a constraint

Minimize $f(\mathbf{y}) = \Sigma \ (x - \mathbf{A}\mathbf{y})^2$ Convex cost function

Subject to $\mathbf{y} >= 0$ Convex constraint

*When you have constraints, can use **CVX**, convex toolbox for Matlab

http://cvxr.com/cvx/




**y***

0.3612
0.2238
0.0006
0.0000
0.7336
0.0000
0.0000
0.0000
0.0000

# Spectral un-mixing with a positivity constraint

deep imaging

## Option 1: Add a constraint

Minimize $f(\mathbf{y}) = \Sigma\ (\mathbf{x} - \mathbf{A}\mathbf{y})^2$ Convex cost function

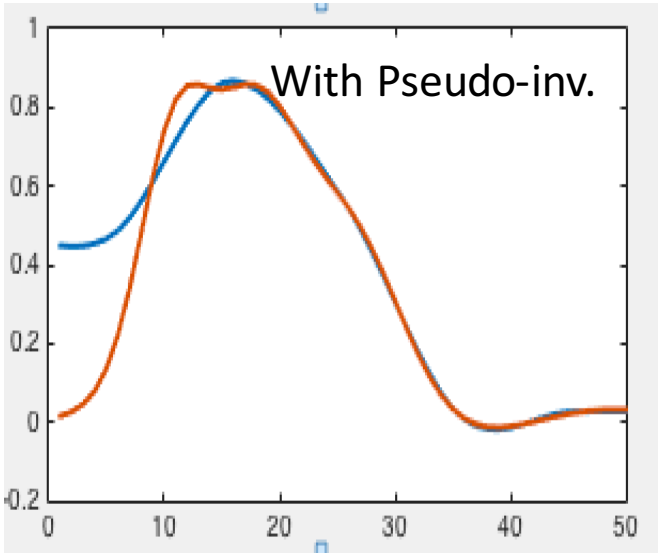
Subject to $\mathbf{y} >= 0$ Convex constraint

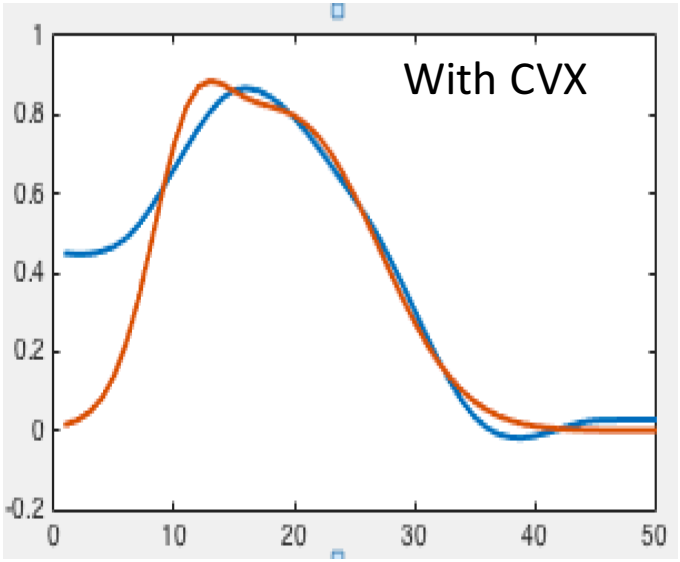
*When you have constraints, can use **CVX**, convex toolbox for Matlab

http://cvxr.com/cvx/

**y***

0.6683
-1.5880
5.7848
-11.7459
17.9304
-20.1231
18.3572
-10.7984
2.8557

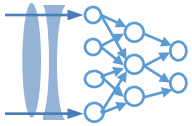



**y***

0.3612
0.2238
0.0006
0.0000
0.7336
0.0000
0.0000
0.0000
0.0000

# Spectral un-mixing with a positivity constraint




## Option 2: Modify cost function

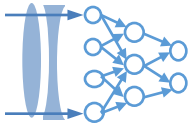
Minimize $f(\mathbf{z}) = \Sigma \ (\mathbf{x} - \mathbf{A}\mathbf{z}^2)^2$

$\mathbf{z}^2 = \mathbf{y}$ is dummy variable, will change cost function and gradient

*When you don't have constraints but can find the gradient, use **Minfunc**

https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html

# Spectral un-mixing with a positivity constraint




## Option 2: Modify cost function

Minimize $f(\mathbf{z}) = \Sigma \ (\mathbf{x} - \mathbf{A}\mathbf{z}^2)^2$

$\mathbf{z}^2 = \mathbf{y}$ is dummy variable, will change cost function and gradient

```
%%%%%%%%%%%%
%3. Minfunc
addpath '/users/Roarke/Documents/Matlab/minFunc_2012';
startVec = ones(m,1);
spectrum_anonymous = @(startVec)spectrum(startVec, b, A);
%Evaluate with minfunc
[xm, msevalue, moreinfo] = minFunc(@(startVec)spectrum_anonymous(startVec), startVec, options);
figure;plot(b); hold all; plot(A*abs(xm).^2);
```

# Spectral un-mixing with a positivity constraint

deep imaging

## Option 2: Modify cost function

Minimize $f(\mathbf{z}) = \Sigma \ (\mathbf{x} - \mathbf{A}\mathbf{z}^2)^2$

$\mathbf{z}^2 = \mathbf{y}$ is dummy variable, will change cost function and gradient

```
%%%%%%%%%%%%%
%3. Minfunc
addpath '/users/Roarke/Documents/Matlab/minFunc_2012';
startVec = ones(m,1);
spectrum_anonymous = @(startVec)spectrum(startVec, b, A);
%Evaluate with minfunc
[xm, msevalue, moreinfo] = minFunc(@(startVec)spectrum_anonymous(startVec), startVec, options);
figure;plot(b); hold all; plot(A*abs(xm).^2);
```

```
function [err_function, grad_function] = spectrum(input_vec, b, A)

%for direct pseudo-inverse - no constraints or dummy
 %err_function = norm(A*input_vec - b);
 %grad_function = A'*(A*input_vec - b);

 err_function = norm(A*abs(input_vec).^2 - b);
 grad_function = A'*((A*abs(input_vec).^2 - b) .* conj(A*input_vec));
```

# Spectral un-mixing with a positivity constraint

deep imaging

## Option 2: Modify cost function

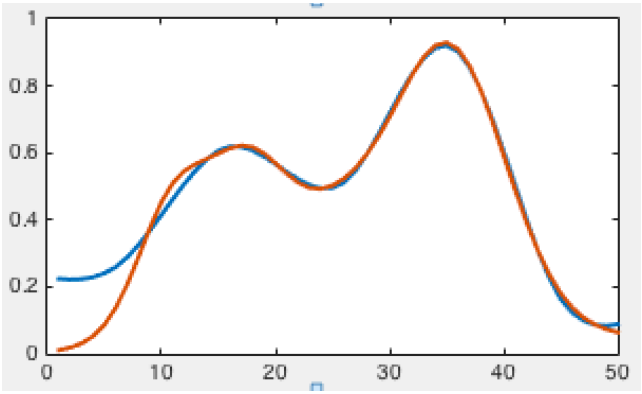
Minimize $f(\mathbf{z}) = \Sigma \ (\mathbf{x} - \mathbf{A}\mathbf{z}^2)^2$

$\mathbf{z}^2 = \mathbf{y}$ is dummy variable, will change cost function and gradient

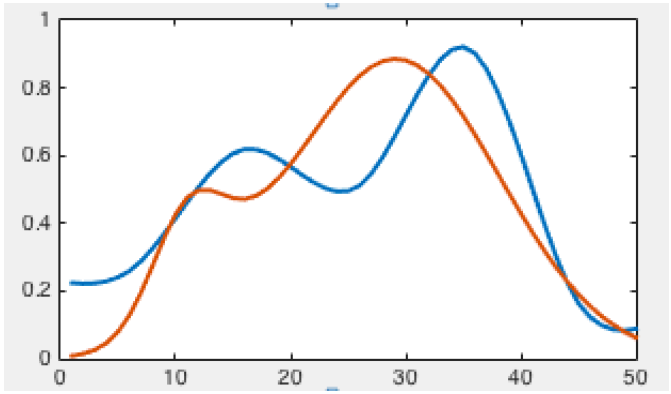
*When you don't have constraints but can find the gradient, use **Minfunc**

https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html

Pseudo-inverse




Dummy variable w/ minfunc

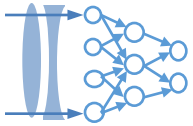



**y***

0.5013
0.3345
0.1811
0.0367
0.0132
0.0705
0.2626
0.5080
0.7539

**Not working too well, gradient could be wrong?**

# Other bells and whistles




1) Sometimes see solutions where x values get really big

Fix this with a "*regularizer*":

$$\text{Minimize } f(\mathbf{x}) = \Sigma\ (\mathbf{b} - \mathbf{Ax})^2 + C^*\Sigma\ (x)^2$$

"Don't let x vary too much"

Choose constant C appropriately

2) If you think your signal is "sparse", then it probably has mostly zeros. Can include this in your model with an "L1" cost function:

$$\text{Minimize } f(\mathbf{x}) = \Sigma\ |\ \mathbf{b} - \mathbf{Ax}\ |$$

- An extremely simple modification with pretty strong implications