Lecture 6: Ingredients for Machine Learning

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer

Outline

- Review spectral unmixing (last class)
- From optimization to machine learning
- Ingredients for ML
- Example: linear classification of images
  - Train/test data
  - Linear regression model
  - 3 ways to solve

Last time: simple example of spectral unmixing

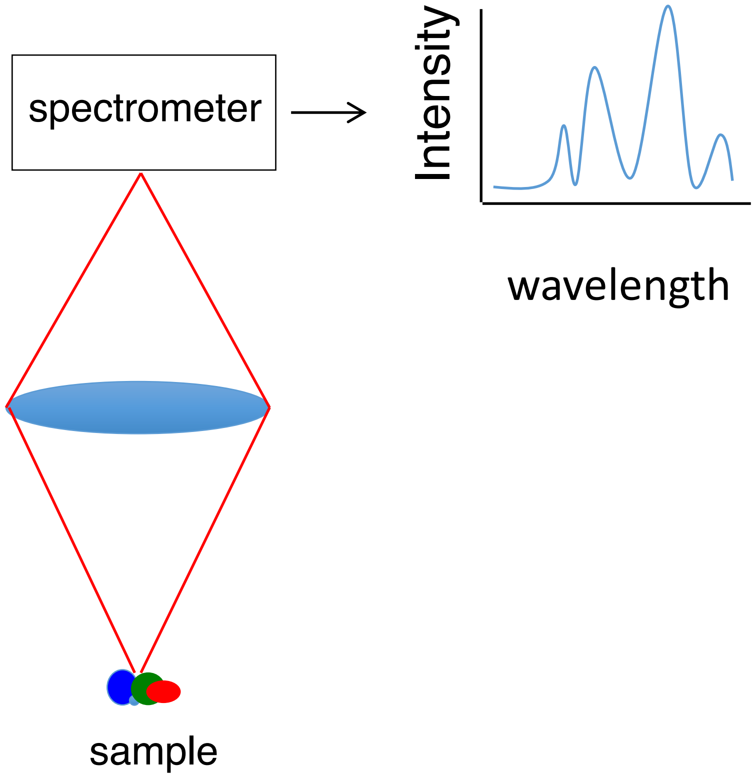
(For whatever reason, whenever I get confused about optimization, I think about this example)

The setup:

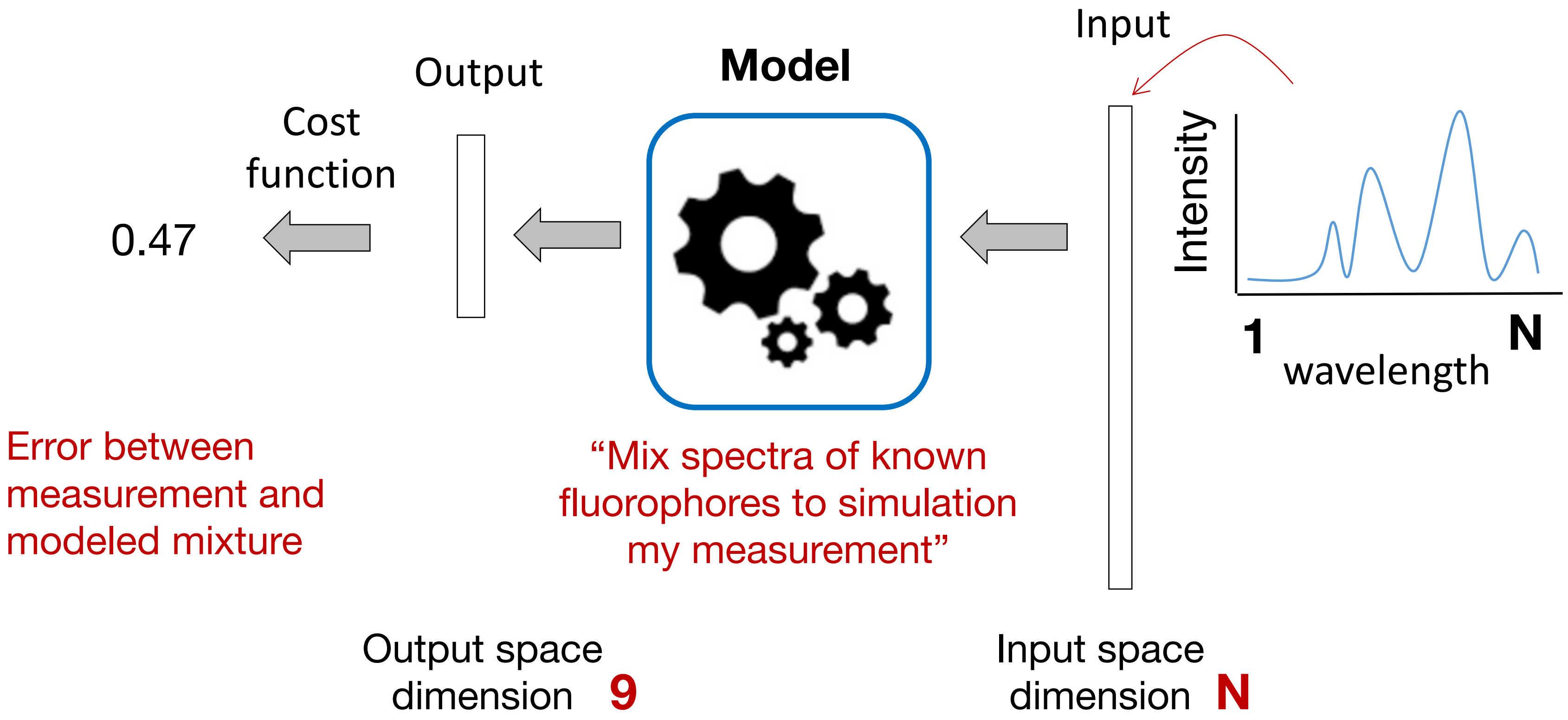
- measure the color (spectral) response of a sample (e.g., how much red, green and blue there is, or several hundred measurements of its different colors).

- You know that the sample can only contain 9 different fluorophores.

- What % of each fluorophores is in your sample?



Optimization pipeline for spectral unmixing



deep imaging

Optimization pipeline for spectral unmixing

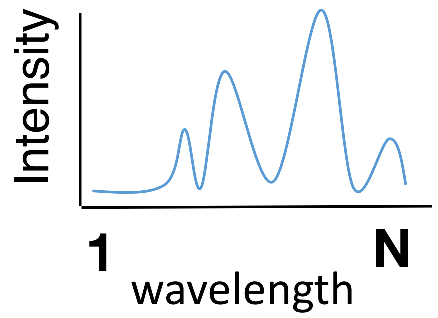
Estimation error | Output: concentrations | Input: spectral measurements

Model

Cost function

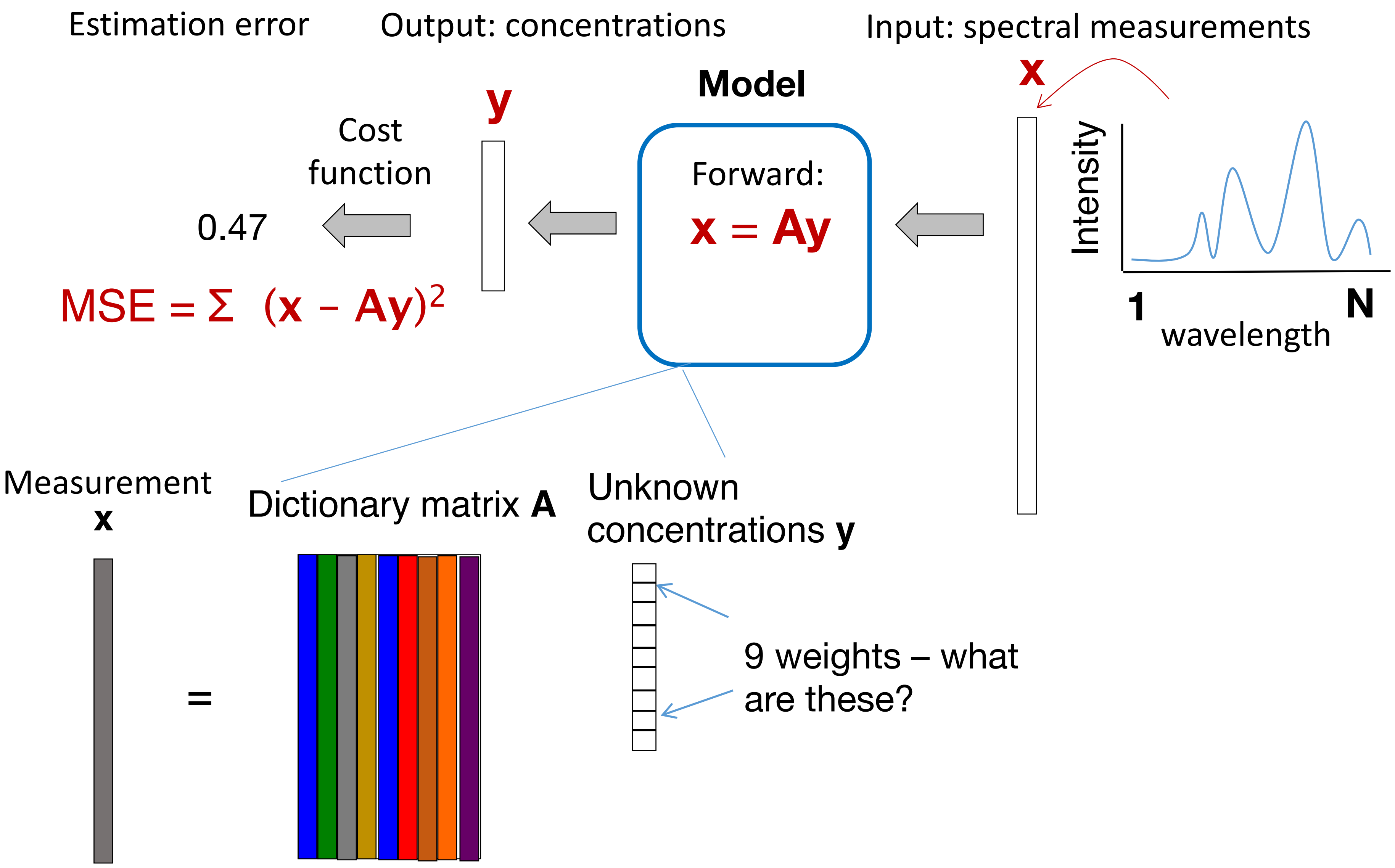
0.47 ← **y** ← Forward: $\mathbf{x} = \mathbf{A}\mathbf{y}$ ← **x**

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



*Note: notation changed from last time to be consistent with we'll use in the future

Optimization pipeline for spectral unmixing



Cost function for spectral unmixing

$\mathbf{x} = \mathbf{Ay}$ won't always be true, due to noise (actually, $\mathbf{x} = \mathbf{Ay}+\mathbf{n}$)

Common cost function is minimum mean-squared error:

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

Cost function for spectral unmixing

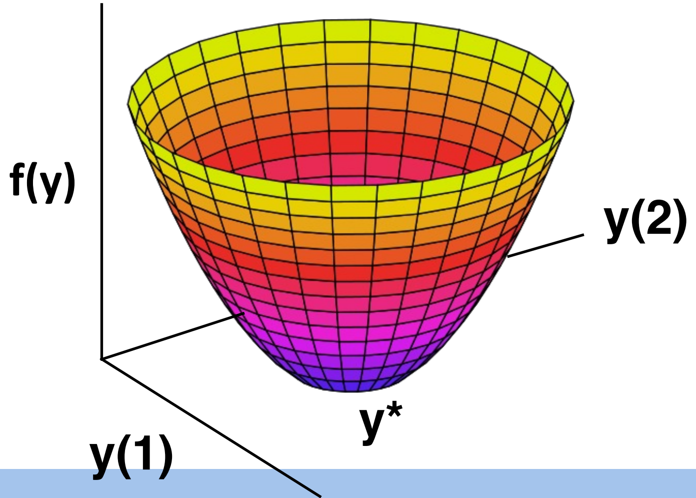
x = **Ay** won't always be true, due to noise (actually, **x** = **Ay**+**n**)

Common cost function is minimum mean-squared error:

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

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

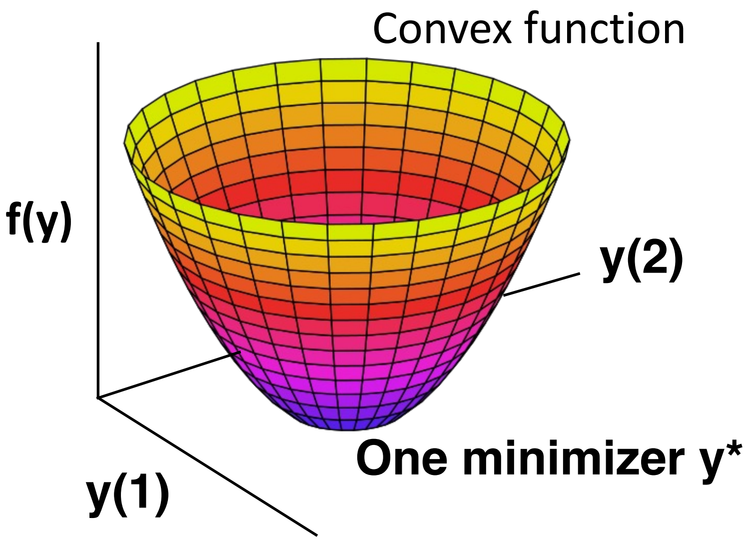
$$\mathbf{y}^* = \text{minimize } f(\mathbf{y})$$



Cost function for spectral unmixing

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

spectral
measurements

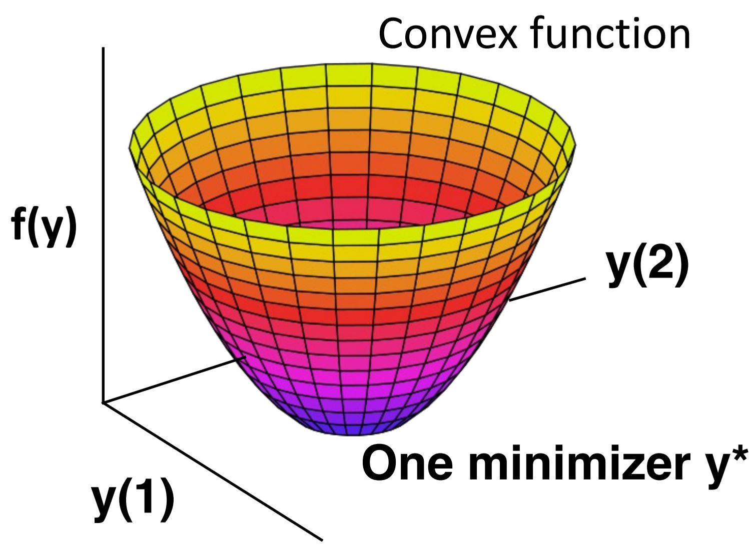


f(**y**) is convex, so finding **y*** is easy via its gradient:

Cost function for spectral unmixing

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

spectral
measurements



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

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

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

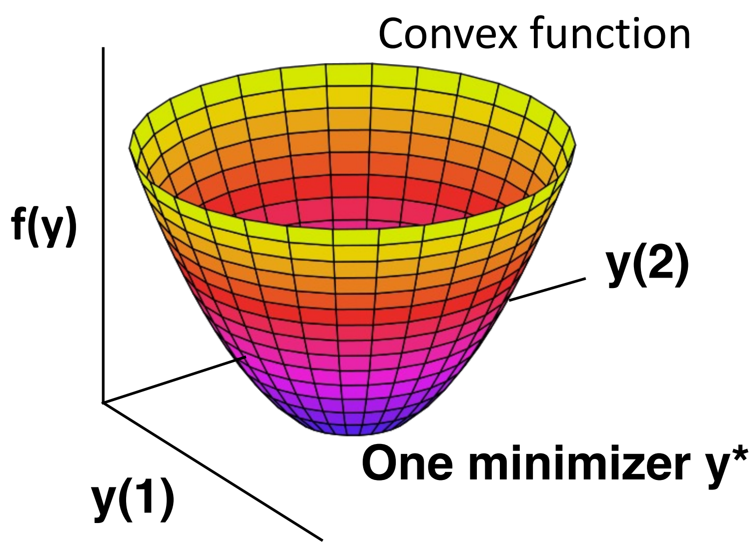
$$df/dy[j] = \Sigma\; -2\; \mathbf{a}(:,j) * (\; \mathbf{x} - \mathbf{Ay})$$

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

Cost function for spectral unmixing

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

spectral
measurements



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

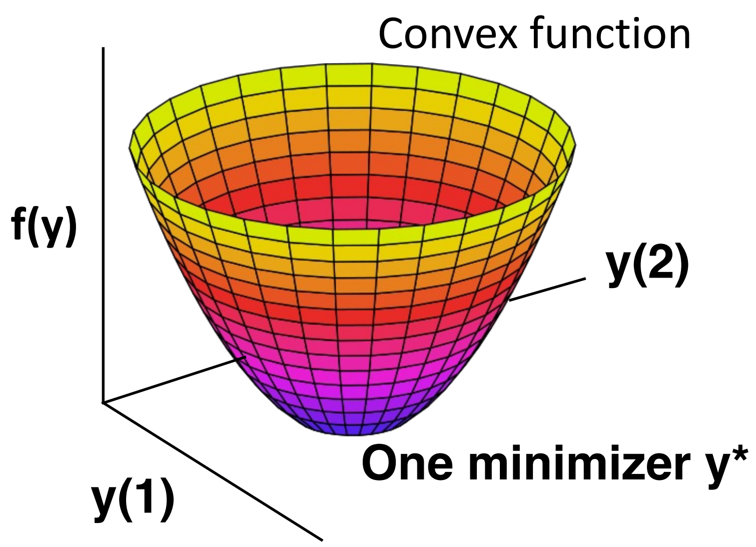
$$df/d\mathbf{y} = \mathbf{A}^T(\ \mathbf{x} - \mathbf{A}\mathbf{y}^*) = 0 \longleftarrow$$ y* is where gradient of f(y) is zero

Cost function for spectral unmixing

deep imaging

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

spectral measurements



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

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

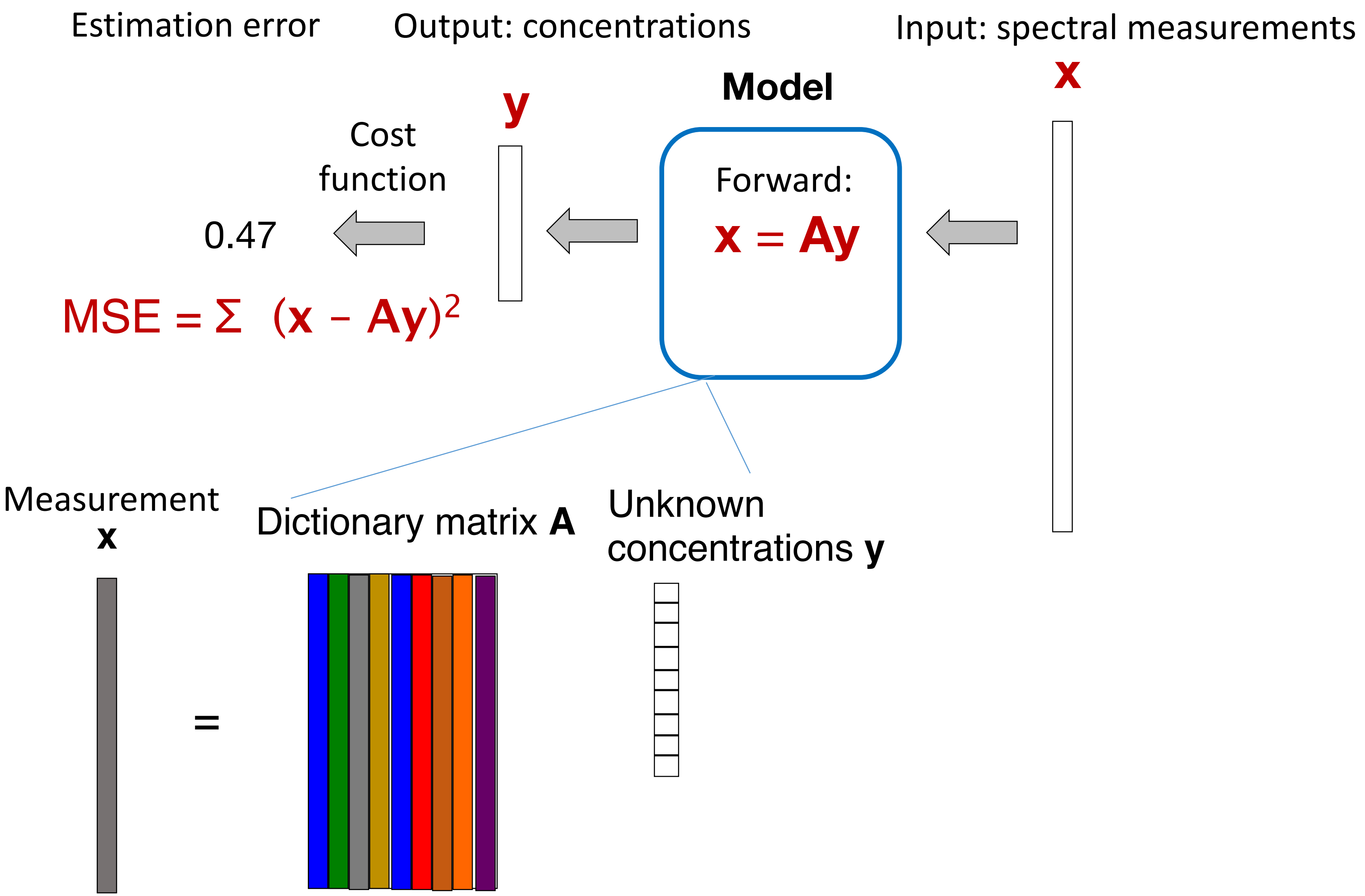
$$\mathbf{A}^T\,\mathbf{x} = \mathbf{A}^T\mathbf{Ay}^*$$ ⟶

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

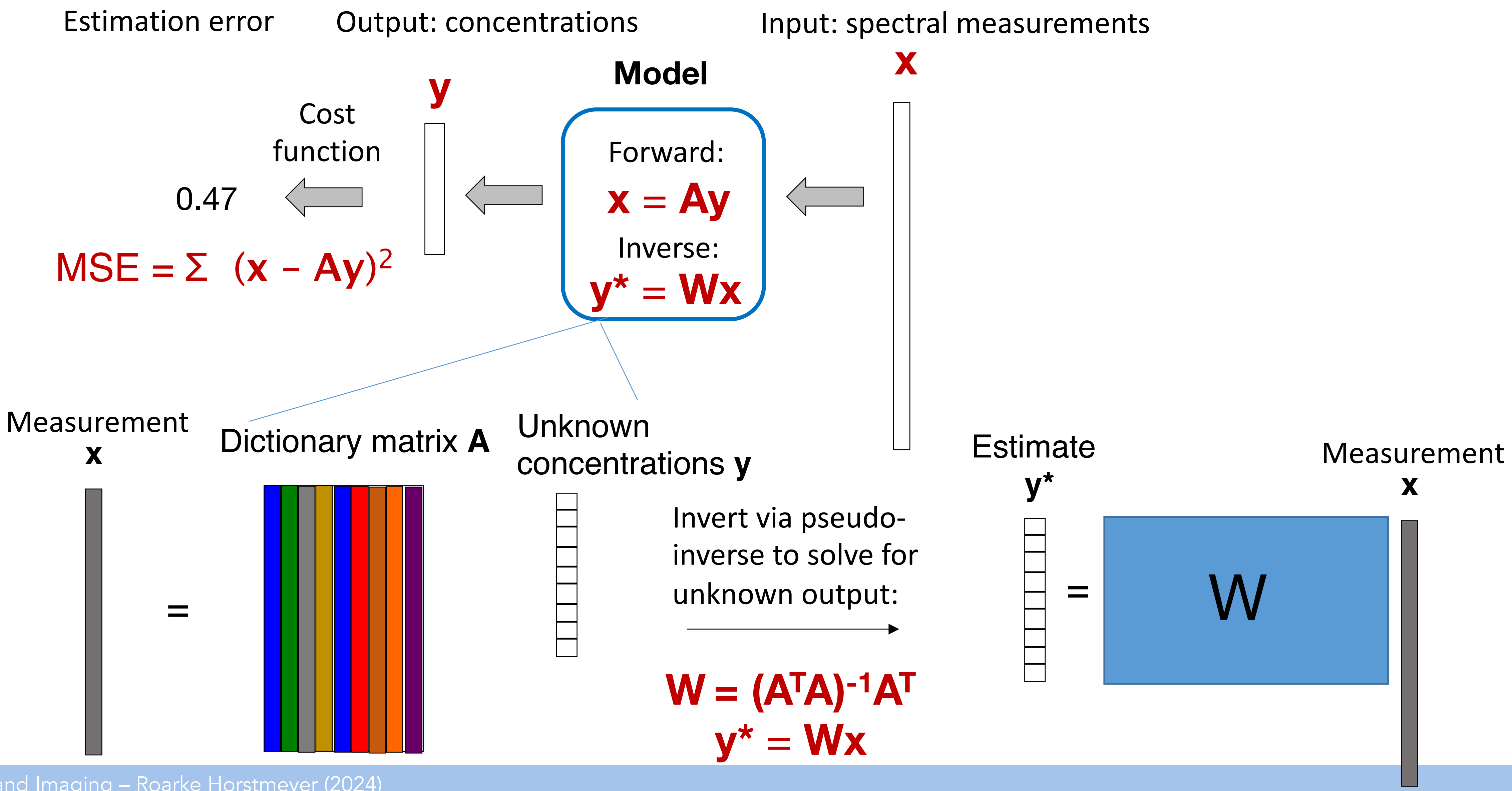
W

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

Optimization pipeline for spectral unmixing



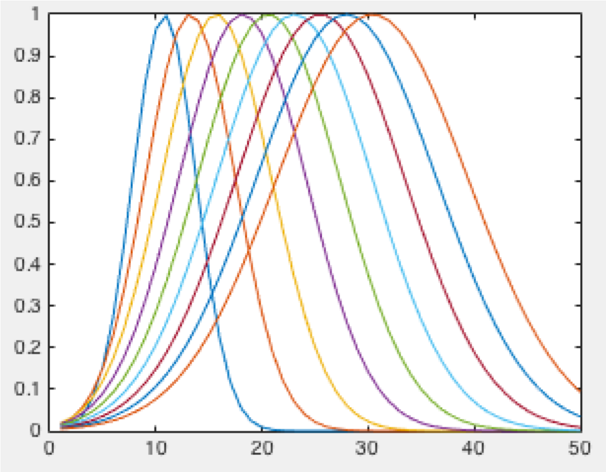
Optimization pipeline for spectral unmixing



Example unmixing with the pseudo-inverse

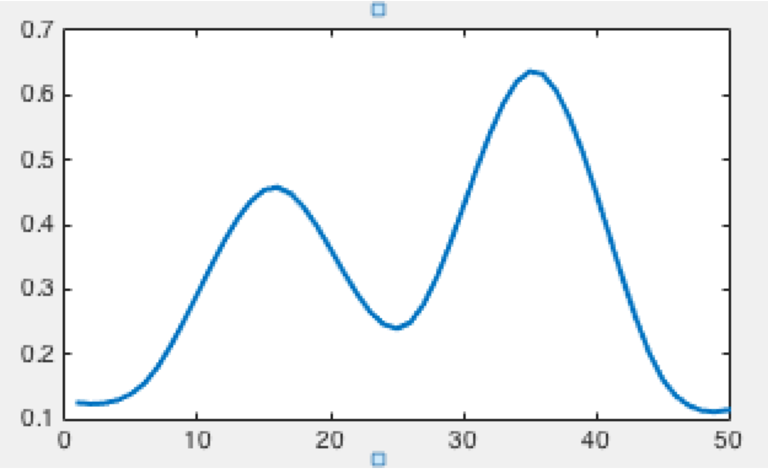
Moore-Penrose Pseudo-inverse: $\boxed{\mathbf{y^* = (A^T A)^{-1} A^T x}}$

Example dictionary A
9 spectra



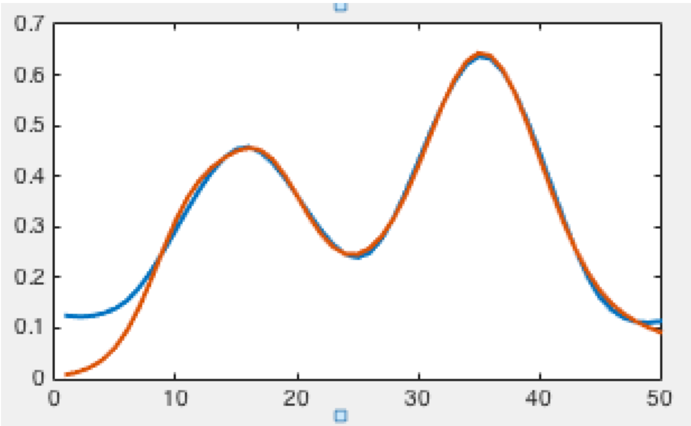
Example y, compute Ay

Example detected spectra x



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

Good fit!



Example unmixing with the pseudo-inverse

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

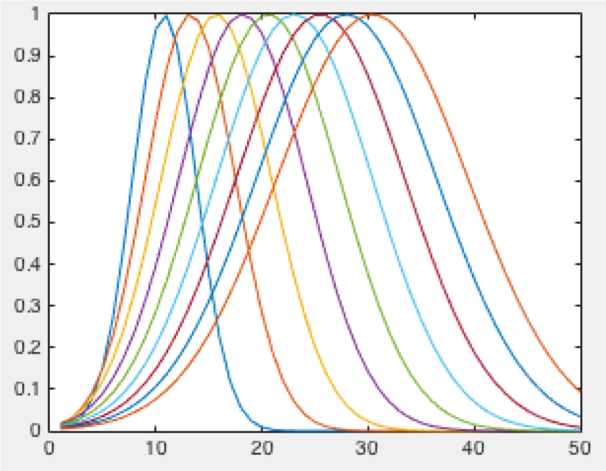
```
n = 50; %number of pixels
m = 9; %number of spectral
A=zeros(n,m); %known dictionary of spectra
for j=1:m
    A(:,j) = exp(-(linspace(-1,1,n)+.5-.1*j+.2).^2/(.03*j));
end
%Simulate some spectra
b = imresize(rand([5,1]),[n 1]);
x_opt = A\b;
%Show results
figure;plot(b); hold all; plot(A*x_opt);
```

Pseudo-inverse = one line

Example unmixing with the pseudo-inverse

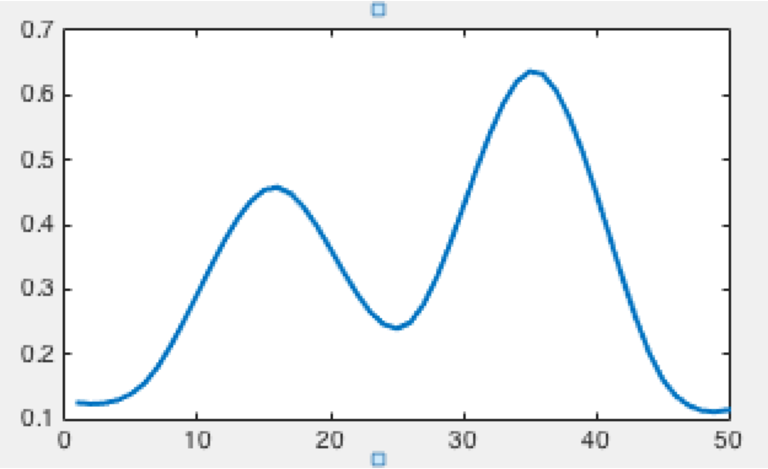
Moore-Penrose Pseudo-inverse: $\boxed{y^* = (A^T A)^{-1} A^T x}$

Example dictionary A
9 spectra



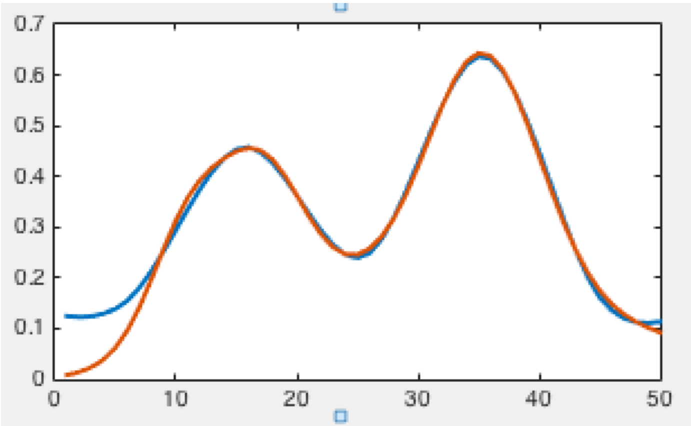
Example y, compute Ay

Example detected spectra x



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

Good fit!



Natural question – when can't we exactly solve for x from Ay=x?

Example 1: **A** represents an under-determined set of equations:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Natural question – when can't we exactly solve for x from Ay=x?

Example 1: **A** represents an under-determined set of equations:

A = [1 1 1]	**x** = [1 0]	Solve for a, b and c:	a + b + c = 1	*Infinite solutions exist*
[1 1 1]	**y** = [a b c]		a + b + c = 0	

Example 2: **A** represents an over-determined set of equations:

Natural question – when can't we exactly solve for x from Ay=x?

Example 1: **A** represents an under-determined set of equations:

A = [1 1 1]
[1 1 1]

x = [1 0]
y = [a b c]

Solve for a, b and c:

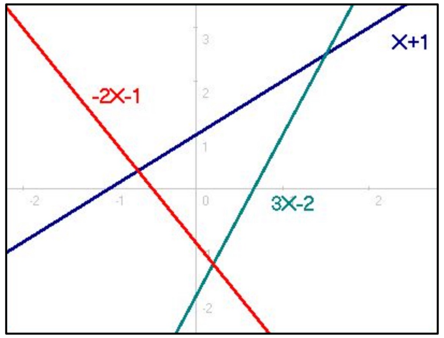
a + b + c = 1
a + b + c = 0

Infinite solutions exist

Example 2: **A** represents an over-determined set of equations:

$$\begin{bmatrix} 2 & 1 \\ -3 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix}$$

No solutions exist



Natural question – when can't we exactly solve for x from Ay=x?

Example 1: **A** represents an under-determined set of equations:

A = [1 1 1]
[1 1 1]

x = [1 0]
y = [a b c]

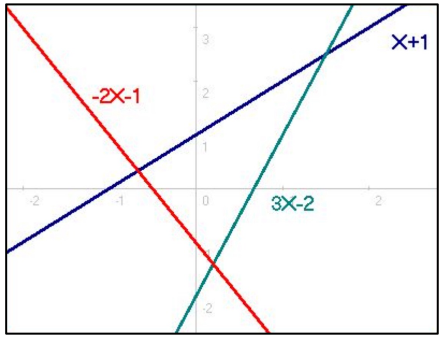
Solve for a, b and c: a + b + c = 1
a + b + c = 0

Infinite solutions exist

Example 2: **A** represents an over-determined set of equations:

$$\begin{bmatrix} 2 & 1 \\ -3 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix}$$

No solutions exist



General rule: If A is not invertible, it has a "nullspace" – a nonzero solution to **Ay = 0** in which **y ≠ 0**

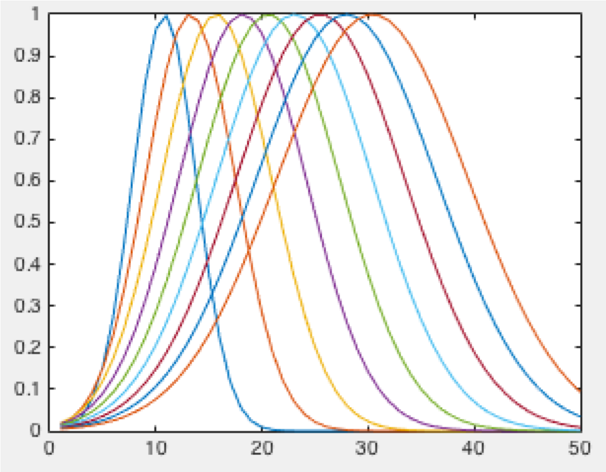
If that is the case, then it is generally challenging to invert x=Ay for y, given x.

For more detail: See *Introduction to Linear Algebra*, Gilbert Strang, Chapters 2.3, 3.2, 3.4, 4.3

Example un-mixing with the pseudo-inverse

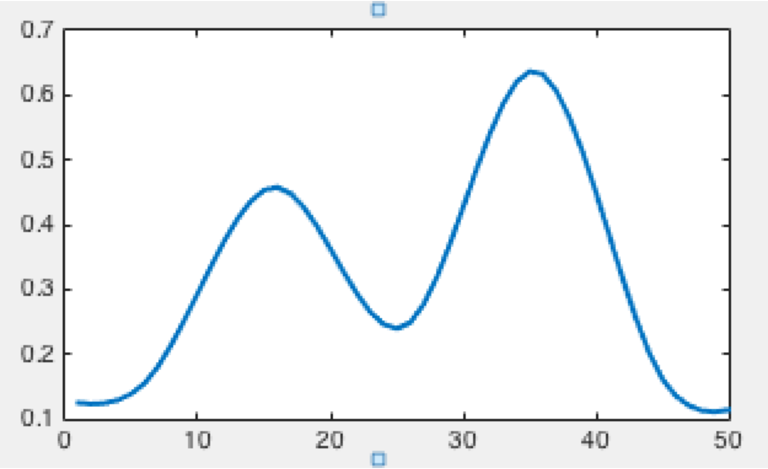
Moore-Penrose Pseudo-inverse: $\boxed{\mathbf{y^* = (A^T A)^{-1} A^T x}}$

Example dictionary A
9 spectra



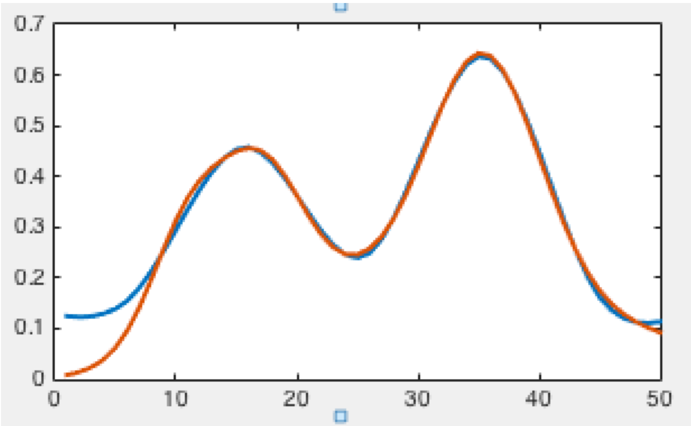
Example y, compute Ay →

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!

Negative light not physically possible…

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/

Spectral un-mixing with a positivity constraint

deep imaging

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

```
%%%%%%%%%%%%%%
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);
%%%%%%%%%%%%%%
```

Vector Norm: $\sqrt{\Sigma\,(Ax_c - b)^2}$

Vector Norms – a quick aside

deep imaging

DEF: A norm is a function $\| \cdot \| : R^n \to R$ that satisfies

$$(1)\quad \|x\| \geq 0, \text{ and } \|x\| = 0 \text{ only if } x = 0,$$
$$(2)\quad \|x + y\| \leq \|x\| + \|y\|,$$
$$(3)\quad \|\alpha x\| = |\alpha|\,\|x\|.$$

Important Norms:

$$\|x\|_1 = \sum_{i=1}^{m} |x_i|,$$

$$\|x\|_2 = \left(\sum_{i=1}^{m} |x_i|^2 \right)^{1/2} = \sqrt{x^* x},$$

$$\|x\|_\infty = \max_{1 \leq i \leq m} |x_i|,$$

$$\|x\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{1/p} \quad (1 \leq p < \infty).$$

Example:

$$x = \begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix}$$

$$\|x\|_1 = 10$$

$$\|x\|_2 = \sqrt{4 + 25 + 9} \approx 6.1644$$

$$\|x\|_\infty = 5$$

$$\|x\|_p = \sqrt[p]{2^p + 5^p + 3^p}$$

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

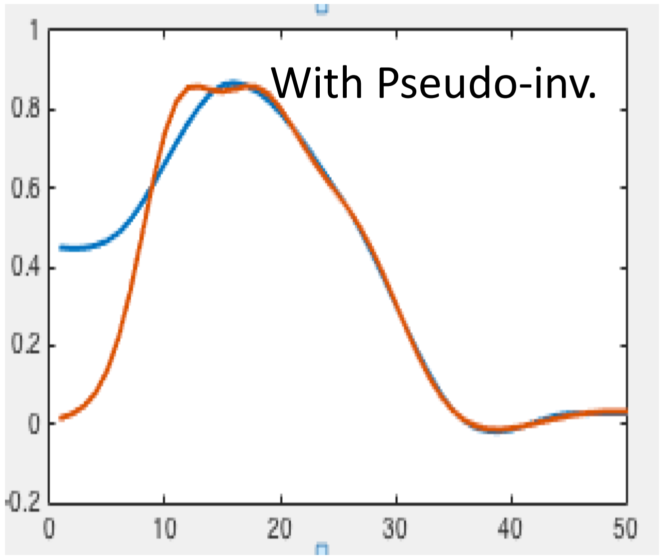
Subject to $\mathbf{y} >= 0$ Convex constraint

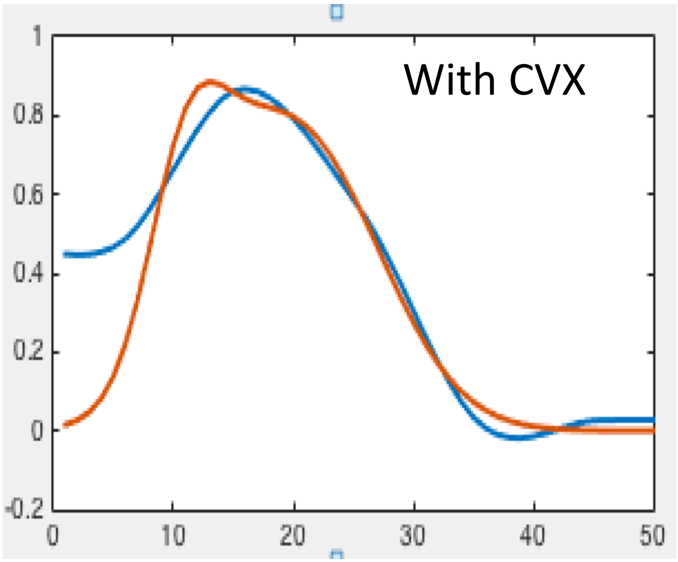
*When you have constraints, can use **CVX**, convex toolbox for Matlab

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

deep imaging

Option 2: Modify cost function

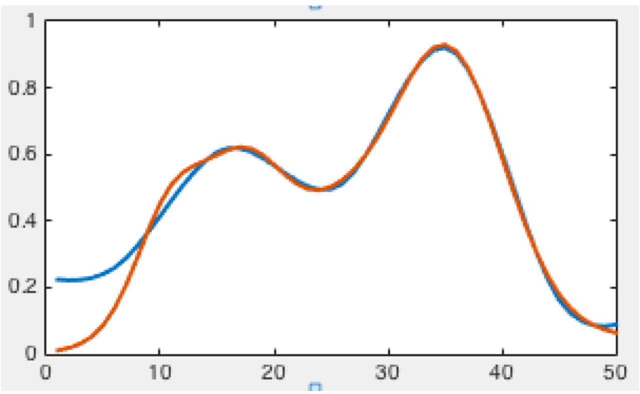
Minimize $f(\mathbf{z}) = \Sigma \ (\mathbf{x} - \mathbf{A}\mathbf{z}^2)^2$

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

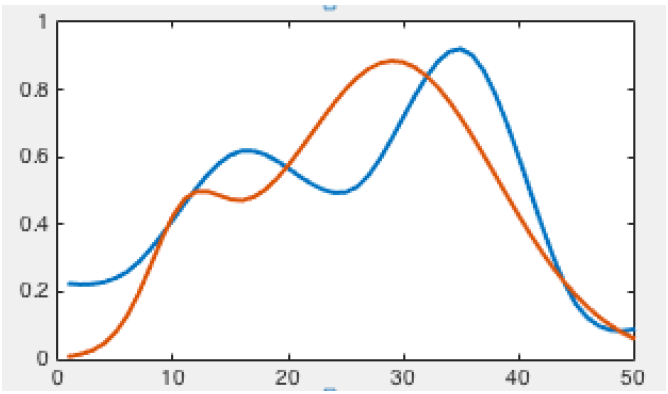
*When you don't have constraints but can find the gradient, use **Minfunc**

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?

More typical strategy to solve for y*: gradient descent

Estimation error

Output: concentrations

Input: spectral measurements

Model

$\mathbf{x}$

$\mathbf{y}$

Cost function

Forward:

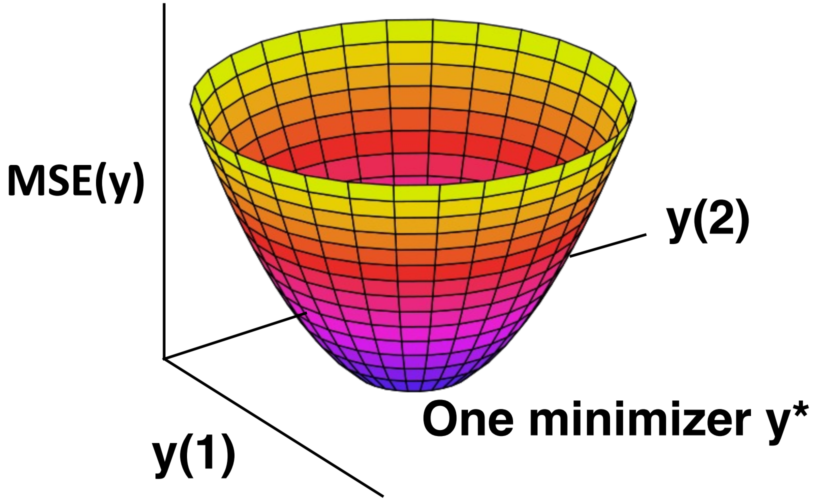
$\mathbf{x} = \mathbf{A}\mathbf{y}$

0.47

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

$$d/d\mathbf{y}\ \text{MSE} = 2\ \mathbf{A}^T(\mathbf{x} - \mathbf{A}\mathbf{y})$$

Slope at any location y



More typical strategy to solve for y*: gradient descent

Estimation error | Output: concentrations | Input: spectral measurements

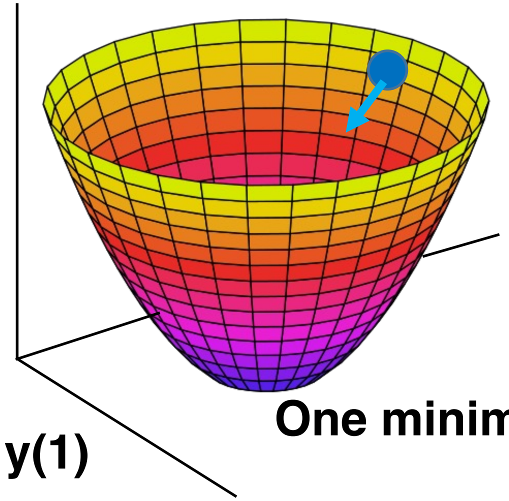
Model

Cost function

0.47 ← $\mathbf{y}$ ← Forward: $\mathbf{x} = \mathbf{A}\mathbf{y}$ ← $\mathbf{x}$

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

$$d/d\mathbf{y}\ \mathrm{MSE} = 2\ \mathbf{A}^T(\mathbf{x} - \mathbf{A}\mathbf{y})$$



1. Guess output $\mathbf{y}_i$
2. Evaluate slope
3. If its large, take a step:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \varepsilon\mathbf{A}^T(\mathbf{x} - \mathbf{A}\mathbf{y}_i)$$

deep imaging

Optimization pipeline for spectral unmixing

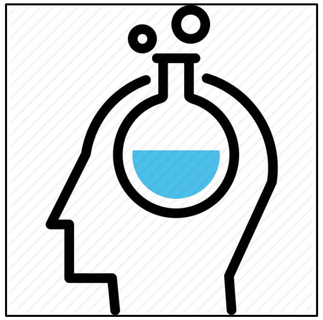
Estimation error | Output: concentrations | Input: spectral measurements

Model

$\mathbf{x}$ → Forward: $\mathbf{x} = \mathbf{A}\mathbf{y}$, Inverse: $\mathbf{y} = \mathbf{W}\mathbf{x}$ → $\mathbf{y}$ → Cost function → 0.47

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

Optimization: You only care about finding the best solution **y***



A and **W** from first principles

Optimization pipeline for spectral unmixing

Estimation error

Output: concentrations

Input: spectral measurements

Model

$\mathbf{x}$

$\mathbf{y}$

Cost function

0.47

MSE $= \Sigma\ (\mathbf{x} - \mathbf{Ay})^2$

Forward:

$\mathbf{x} = \mathbf{Ay}$

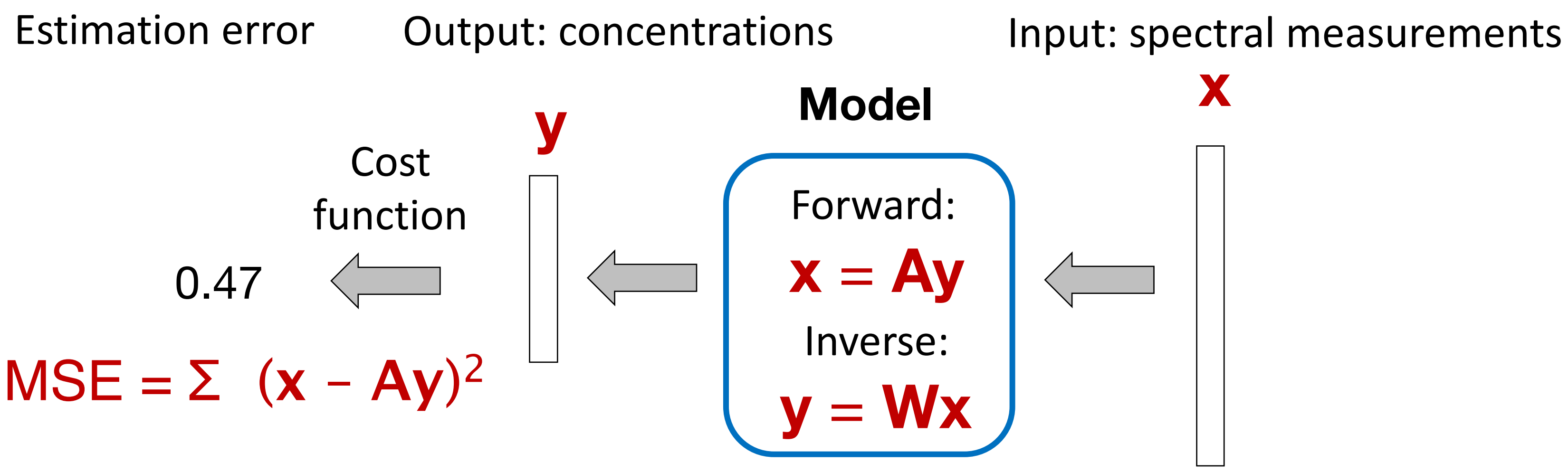
Inverse:

$\mathbf{y} = \mathbf{Wx}$

Machine Learning: You don't know what A and W are!

- **You do not have access to known forward or inverse models**
- **But, example (input, output) data is available to try to figure it out**

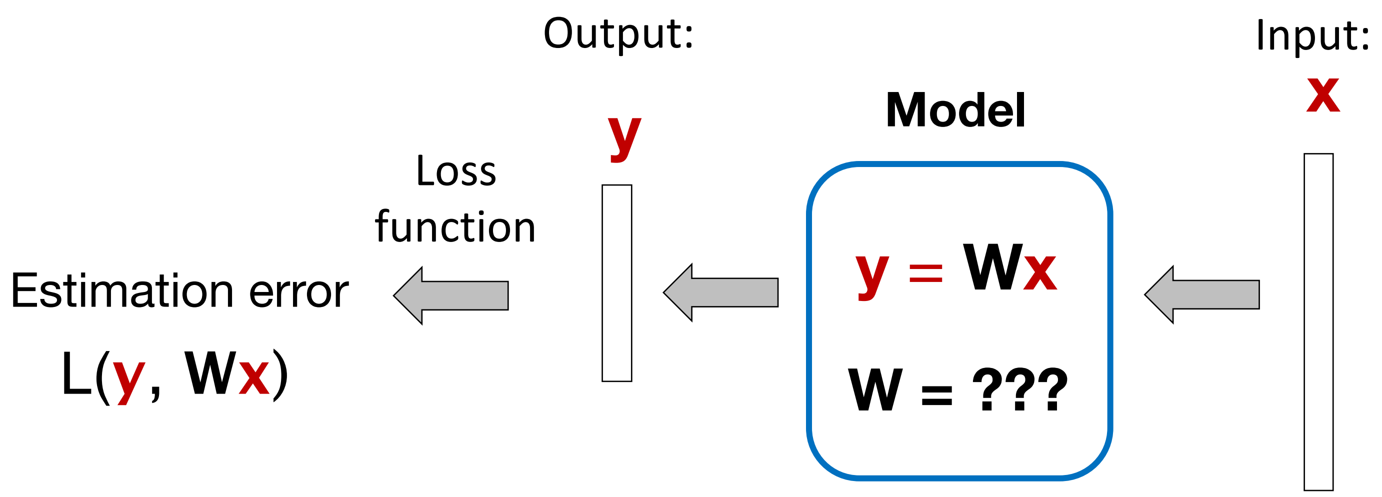
Optimization pipeline for spectral unmixing



Optimization: You only care about finding the best solution **y***

Machine Learning: You first care about finding the model **W**, then you'll use that to find the best solution **y***

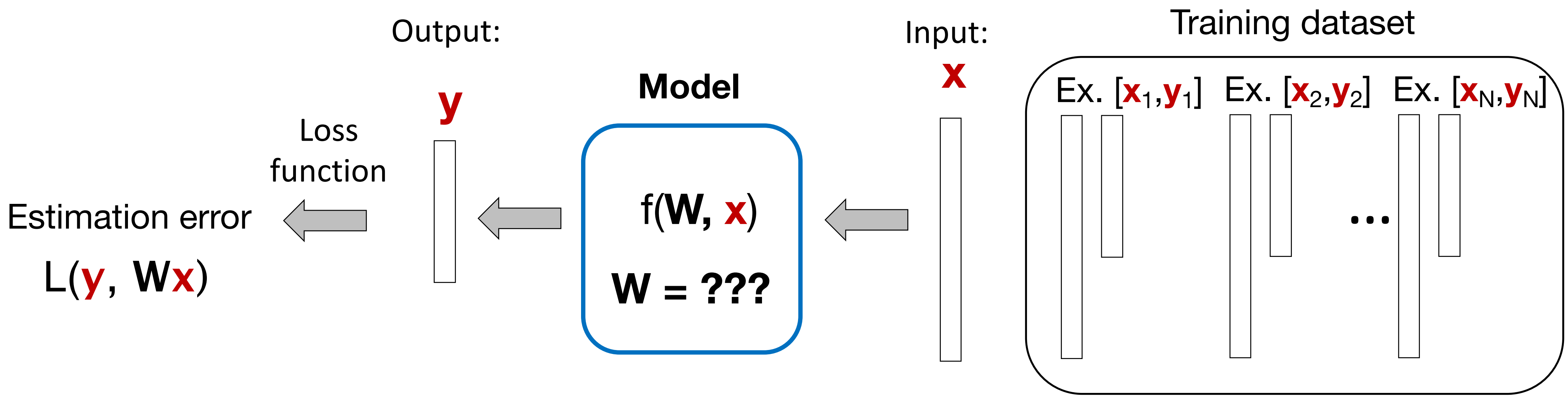
Pipeline for machine learning



Changes for machine learning framework:

1. Now must establish the mapping from inputs to outputs (here, matrix **W**)

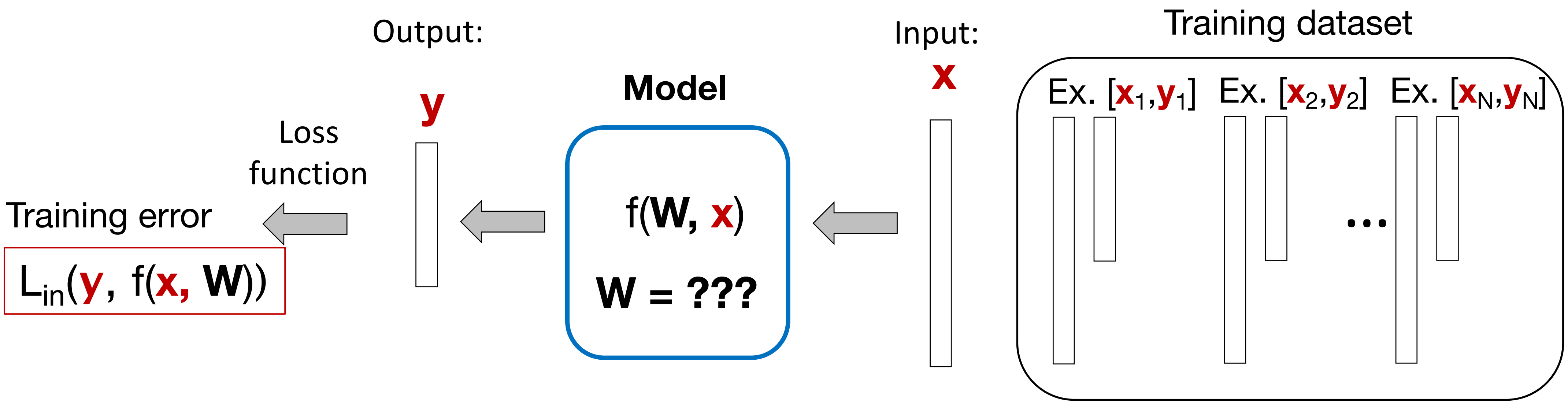
Pipeline for machine learning



Changes for machine learning framework:

1. Now must establish the mapping from inputs to outputs (here, matrix **W**)
2. Using large set of "training" data to first determine mapping f(**x, W**) = **Wx**

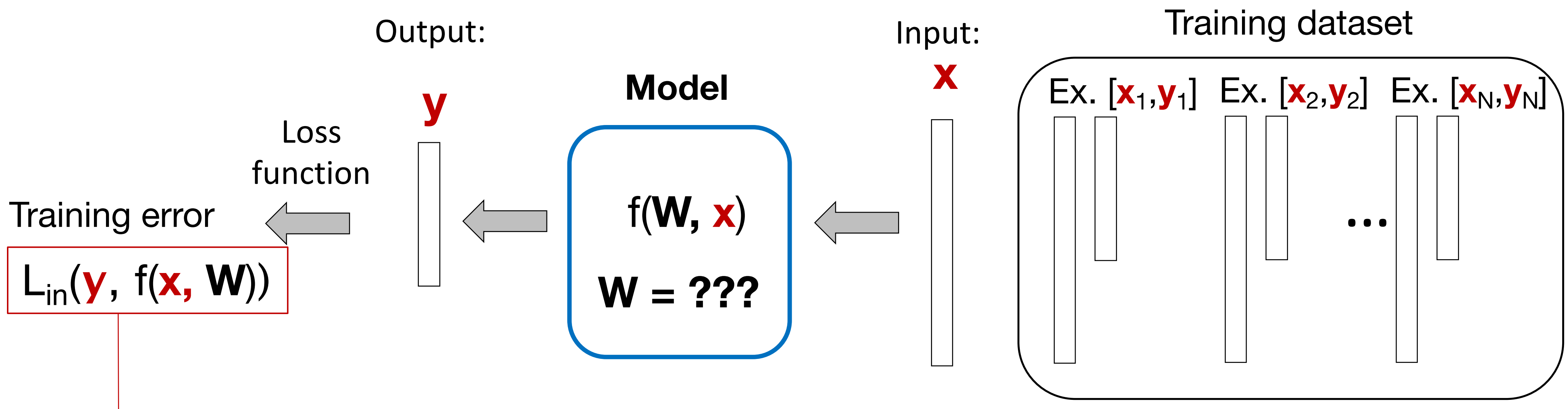
Pipeline for machine learning



Changes for machine learning framework:

1. Now must establish the mapping from inputs to outputs (here, matrix **W**)
2. Using large set of "training" data to first determine mapping $f(\mathbf{x}, \mathbf{W}) = \mathbf{Wx}$
3. To do so, use a *loss function* L that depends upon the training inputs (x,y) *and* the model (**W**)

Pipeline for machine learning

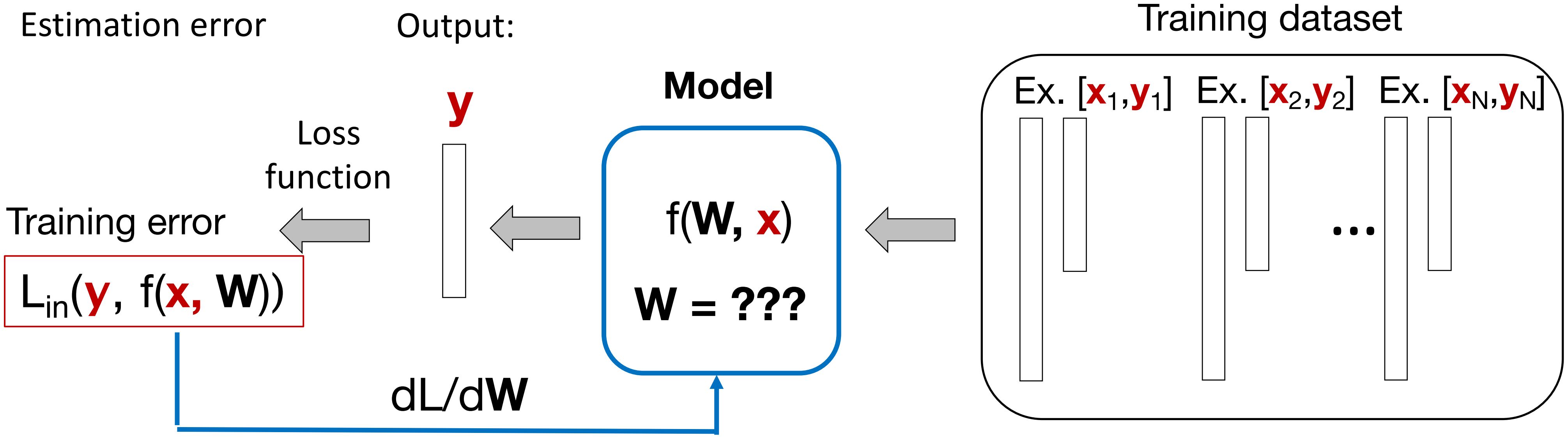


Training Error (“in class error”):

- L_{in} compares modeled output, f($\mathbf{x}_i$, **W**), with the *correct* output that has been *labeled*
- Assume error caused by each labeled example is equally important and sum them up:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

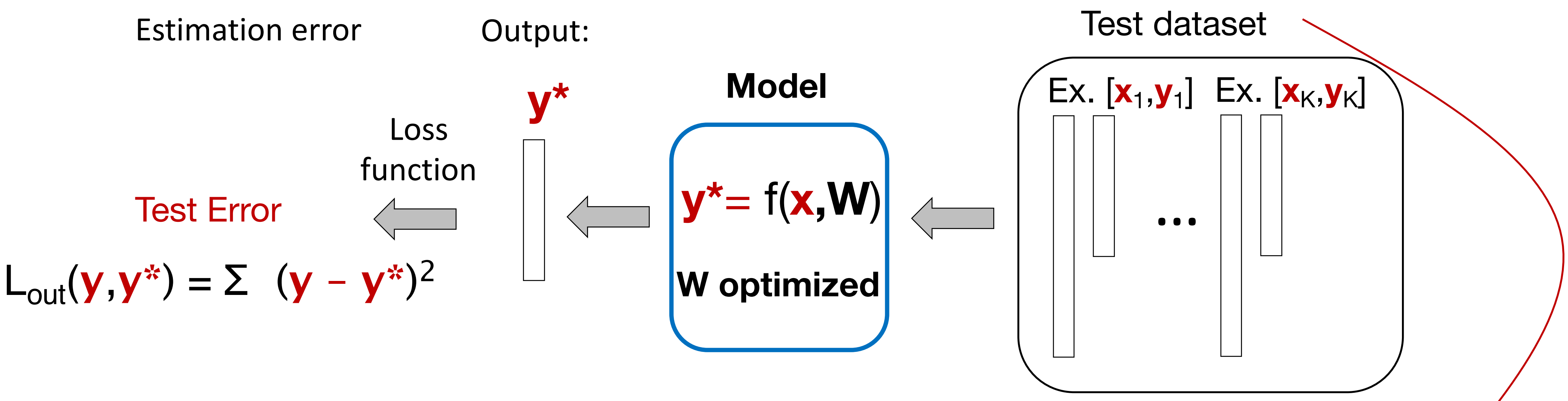
Pipeline for machine learning



Changes for machine learning framework:

1. Now must establish the mapping from inputs to outputs (here, matrix **W**)
2. Using large set of “training” data to first determine mapping f(**x**, **W**) = **Wx**
3. To do so, use a *loss function* L that depends upon the training inputs (x,y) *and* the model (**W**)
4. Find optimal mapping (**W**) using the training data, guided by gradient descent on L

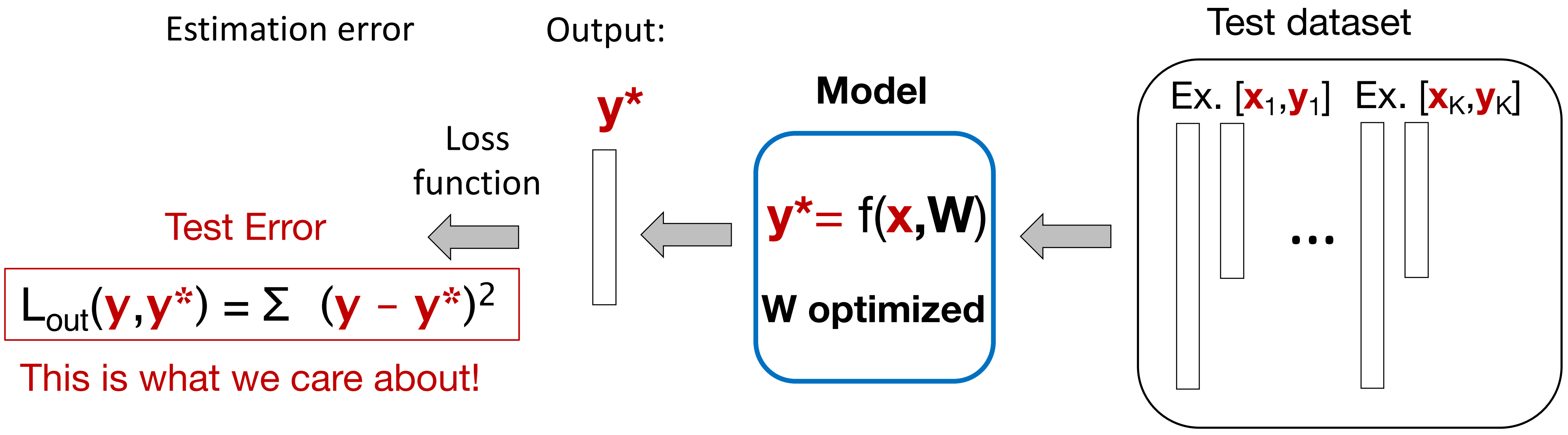
Pipeline for machine learning



In a *separate step*, we then need to do the following to test the network:

1. valuate model accuracy by sending *new* **x** through – need *new, unique* data *with label*

Pipeline for machine learning



In a *separate step*, we then need to do the following to test the network:

1. valuate model accuracy by sending *new* **x** through – need *new, unique* data *with label*
2. Compare output **y*** to known “test data” label **y**
3. Evaluate performance with an error equation L_{out}

Example: machine learning for image classification

deep imaging

Correct labels for test images?

Training error

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

Output: Image class

y

Model

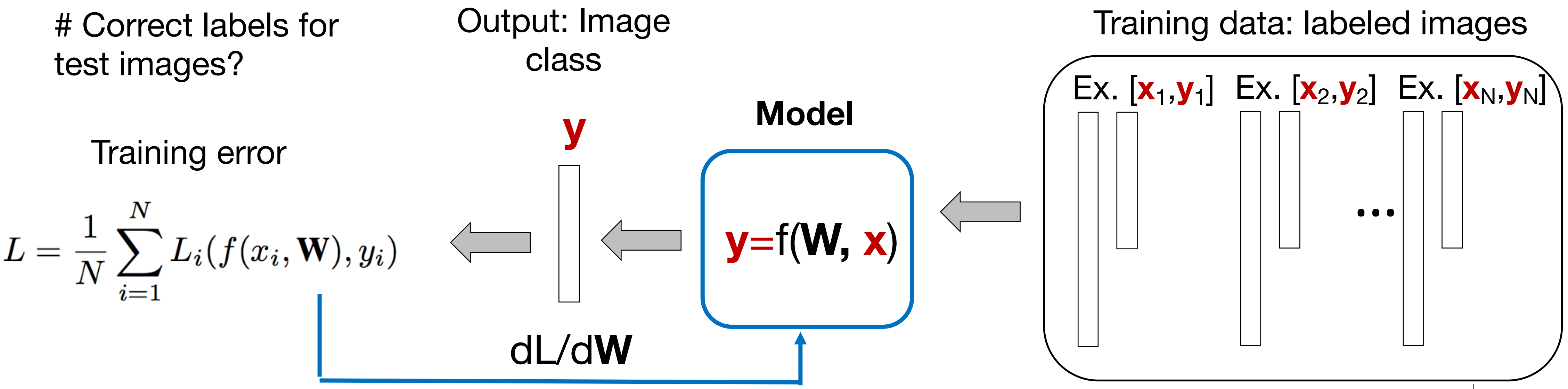
y=f(**W**, **x**)

dL/d**W**

Training data: labeled images

Ex. [$\mathbf{x}_1$,$\mathbf{y}_1$] Ex. [$\mathbf{x}_2$,$\mathbf{y}_2$] ... Ex. [$\mathbf{x}_N$,$\mathbf{y}_N$]

Example: machine learning for image classification



Let's consider a simple example – image classification. What do we need for training?

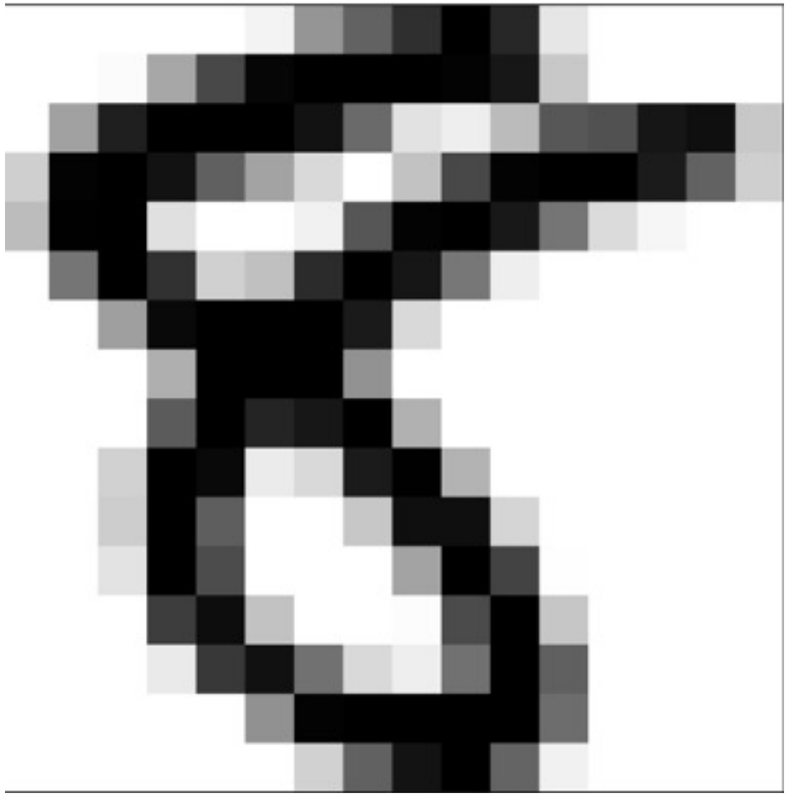
1. Labeled examples $\{(x_i, y_i)\}_{i=1}^{N}$

Example: machine learning for image classification

https://en.wikipedia.org/wiki/MNIST_database

MNIST image set: http://yann.lecun.com/exdb/mnist/

Example: MNIST image dataset



X = 28x28 pixel matrix

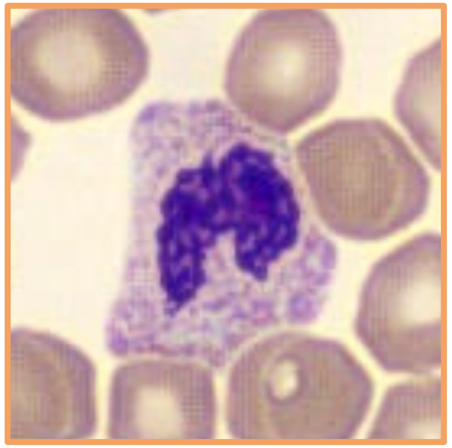
x = vec[**X**] = 784-long vector

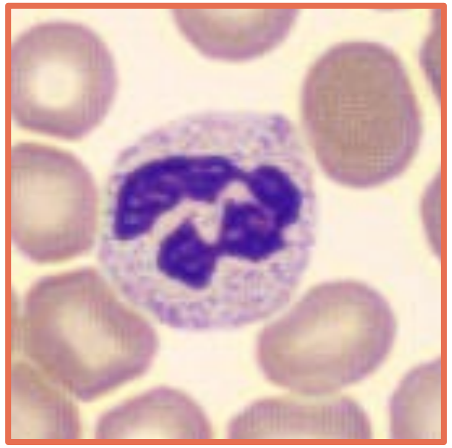
Linear model would require W = 784 element matrix

Start simple: use **x** = (x_0, x_1, x_2) to describe **intensity** and **symmetry** of image X

Linear model can now use smaller w = (w_0, w_1, w_2)

Example images for later in the class: blood cells





X = 384 x 384 **x 3** pixel matrix

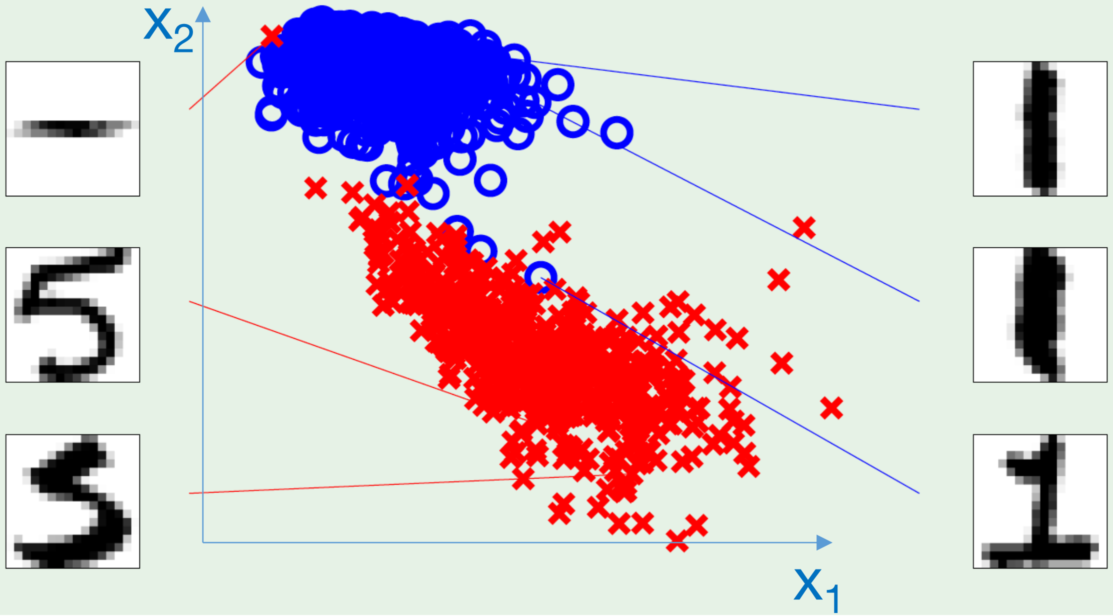
(3^{rd} matrix dimension is Red, Green or Blue pixel values)

x = vec[**M**] = 442,368-long vector

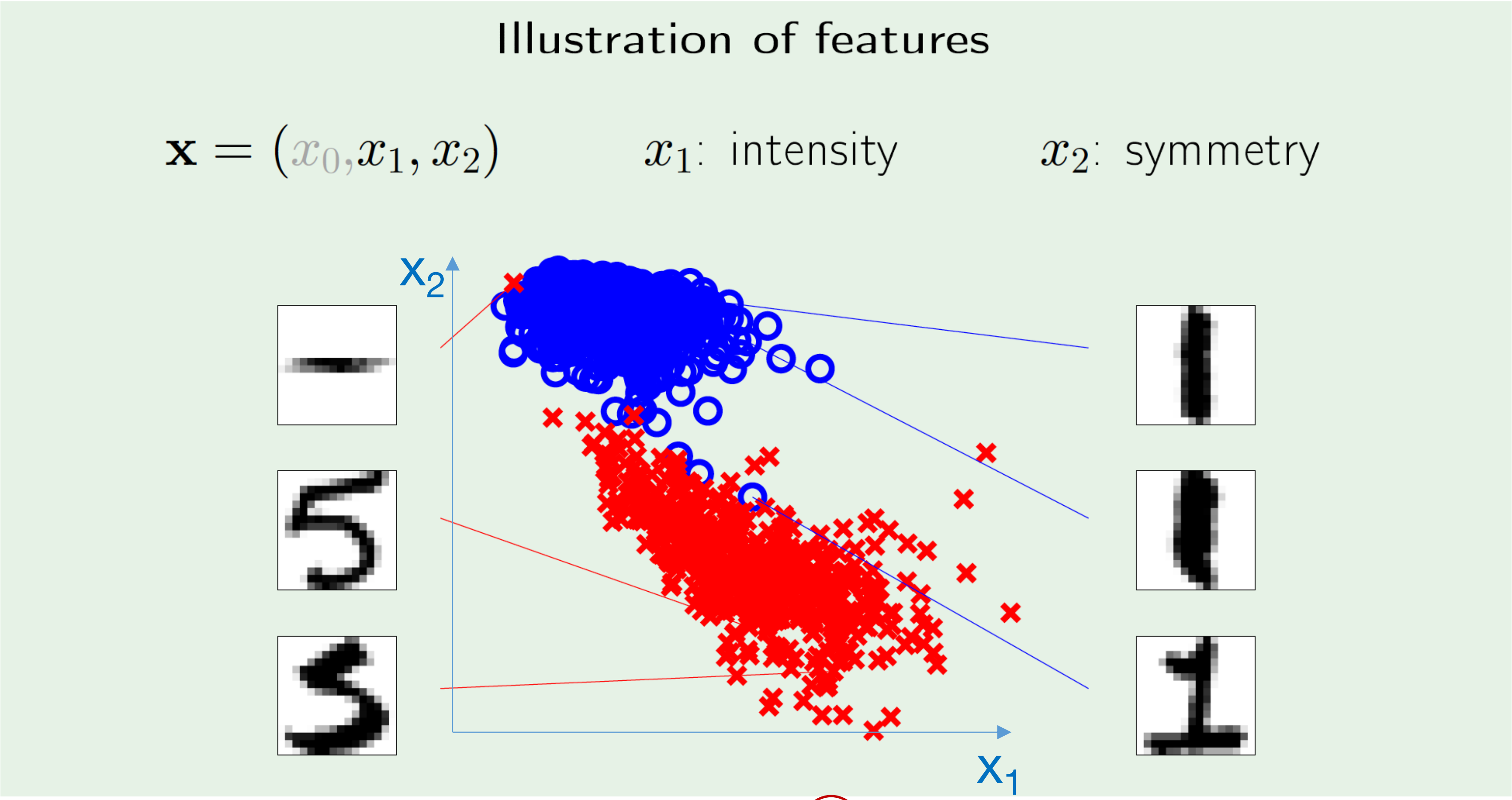
Linear model would require W = 442,368 element matrix

Illustration of features

$\mathbf{x} = (x_0, x_1, x_2)$ x_1: intensity x_2: symmetry



Caltech Learning from Data: https://work.caltech.edu/telecourse.html



Dataset: 1000 examples of 1's and 5's mapped to $\mathbf{x_j} = (1, x_1, x_2)$, with associated label $y_j = 1$ or -1

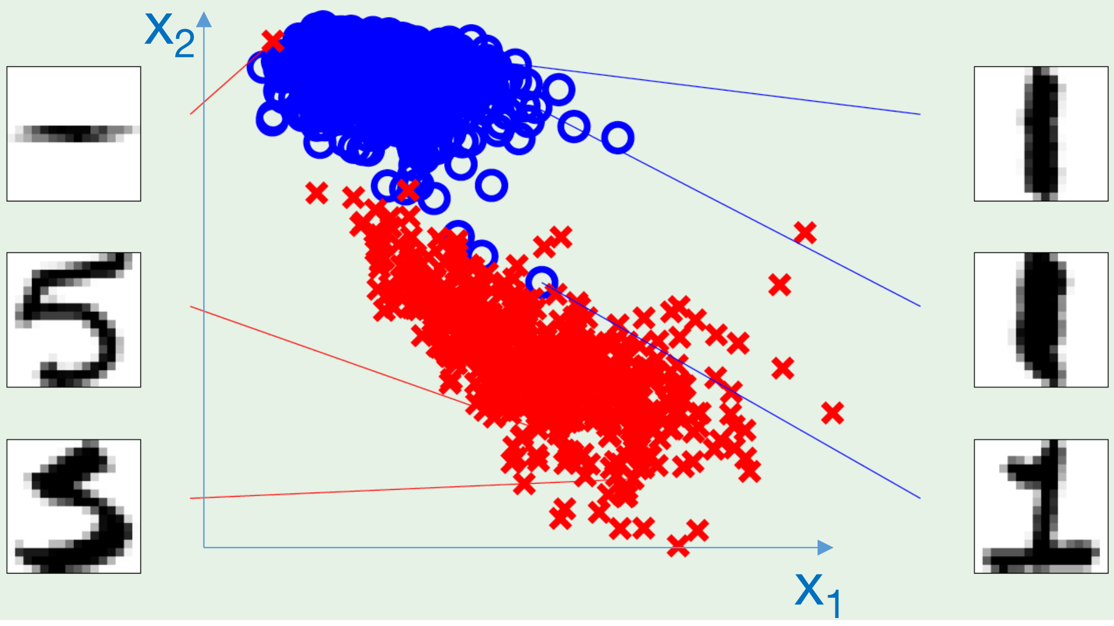
This transforms wx+b into **wx**

$$[w_0\ w_1\ w_2] \begin{vmatrix} 1 \\ x_1 \\ x_2 \end{vmatrix} = w_0 + w_1 x_1 + w_2 x_2$$

Caltech Learning from Data: https://work.caltech.edu/telecourse.html

Illustration of features

$\mathbf{x} = (x_0, x_1, x_2)$ $\quad x_1$: intensity $\quad x_2$: symmetry



Dataset: 1000 examples of 1's and 5's mapped to $\mathbf{x_j} = (1, x_1, x_2)$, with associated label $y_j = 1$ or -1

$[\mathbf{X}_{train}, \mathbf{Y}_{train}] = [\mathbf{x}_j, \mathbf{y}_j]$ for n=1 to 750 $\quad [\mathbf{X}_{test}, \mathbf{Y}_{test}] = [\mathbf{x}_j, \mathbf{y}_j]$ for n=751 to 1000

Labeled data:

Training dataset	Test dataset
n=1-750	n=751-1000

Example: machine learning for image classification

Correct labels with test dataset?

Output: Image class

Images and class labels

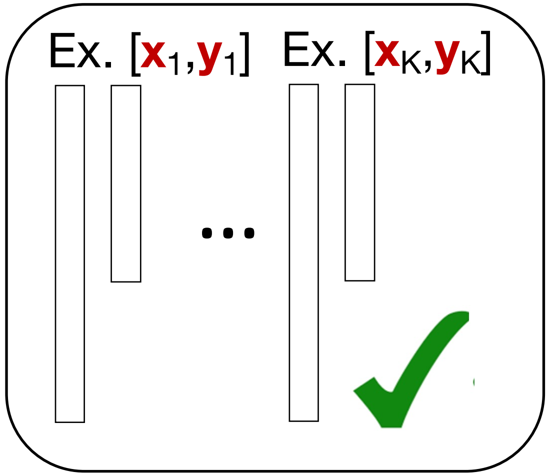
Training error

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

$\mathbf{y}$

Model

$\mathbf{y} = \mathbf{Wx}$



Let's consider a simple example – image classification. What do we need for training?

✓ **1. Labeled examples** $\{(x_i, y_i)\}_{i=1}^{N}$

2. A model and loss function

Example: machine learning for image classification

deep imaging

Correct labels with test dataset?

Output: Image class

Images and class labels

Model

$\mathbf{y}$

Training error

Ex. [$\mathbf{x}_1$,$\mathbf{y}_1$] Ex. [$\mathbf{x}_K$,$\mathbf{y}_K$]

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

$\mathbf{y} = \mathbf{W}\mathbf{x}$

...

Let’s consider a simple example – image classification. What do we need for training?

✓ **1. Labeled examples** $\{(x_i, y_i)\}_{i=1}^{N}$

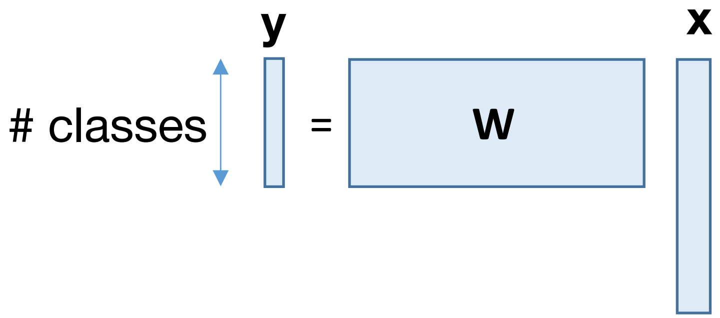
2. A model and loss function

Let’s start with a simpler approach: linear regression

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

General linear model:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(\mathbf{W}\mathbf{x}_i, y_i)$$



Let's start with a simpler approach: linear regression

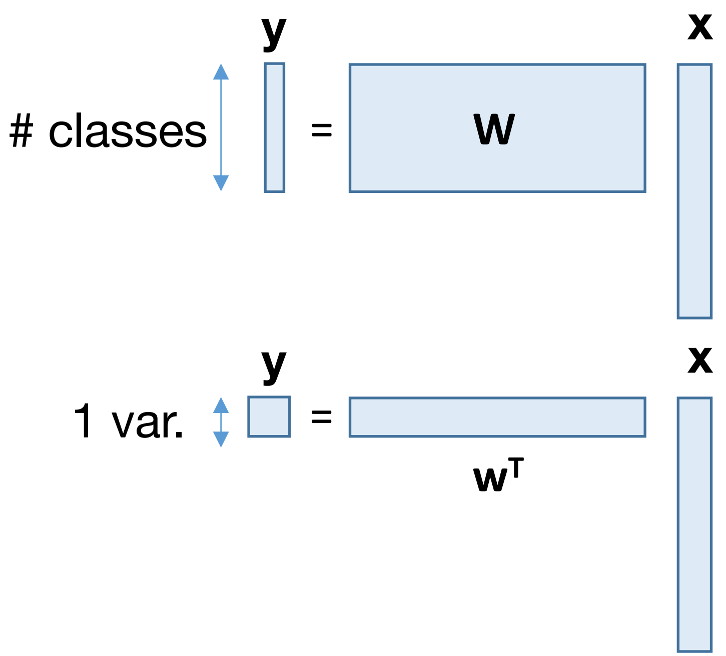
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

General linear model:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(\mathbf{W}\mathbf{x}_i, y_i)$$

Assume 1 class =
1 linear fit

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(w^T x_i \ , \ y_i)$$



Let’s start with a simpler approach: linear regression

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \mathbf{W}), y_i)$$

General linear model:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(\mathbf{W}\mathbf{x}_i, y_i)$$

Assume 1 class = 1 linear fit

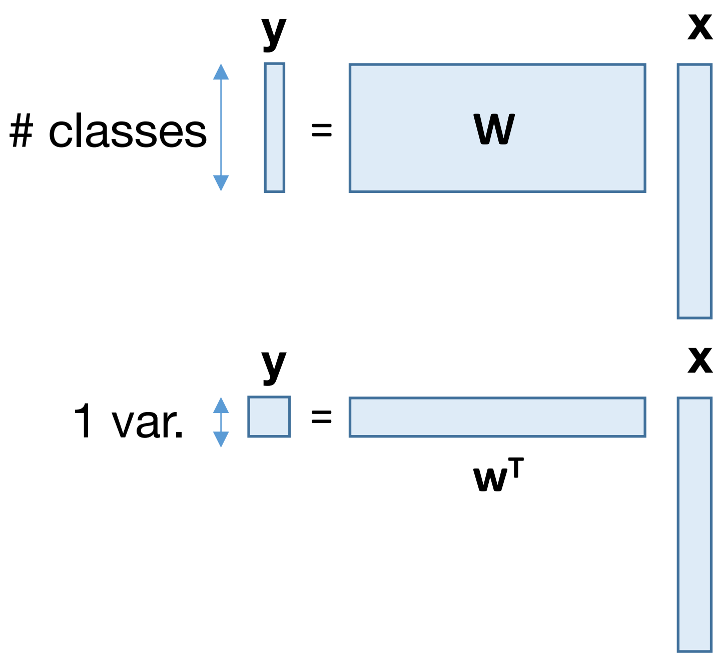
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(w^T x_i \;\;, \;\; y_i)$$

Use MSE error model

$$L = \frac{1}{N}\sum_{i=1}^{N} (w^T x_i - y_i)^2$$

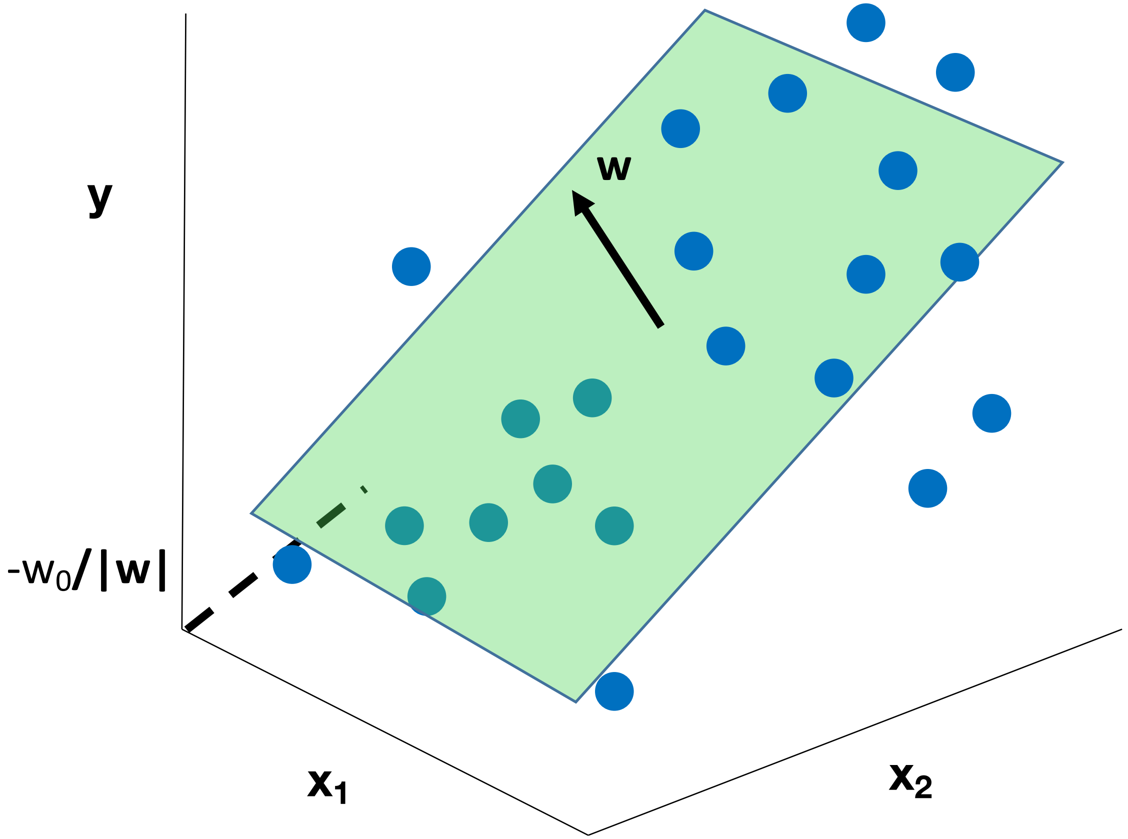
Where labels determined by thresholding

$$f(\mathbf{x}_i) = y_i^* = \mathrm{sgn}(\mathbf{w}^T \mathbf{x}_i)$$



$$\mathrm{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

Why does linear regression with sgn() achieve classification?



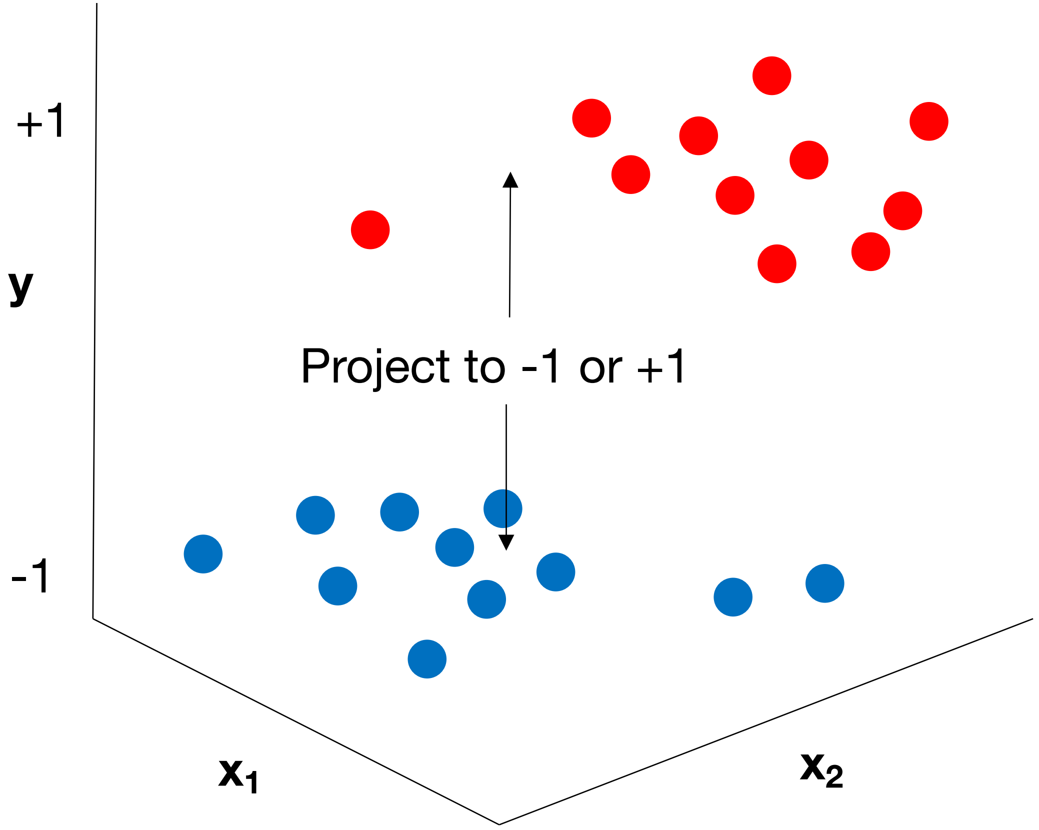
Without sgn(): regression for best fit

$$f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i$$

$$L = \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

- If y_i can be anything, minimizing *L* makes **w** the plane of best fit

Why does linear regression with sgn() achieve classification?



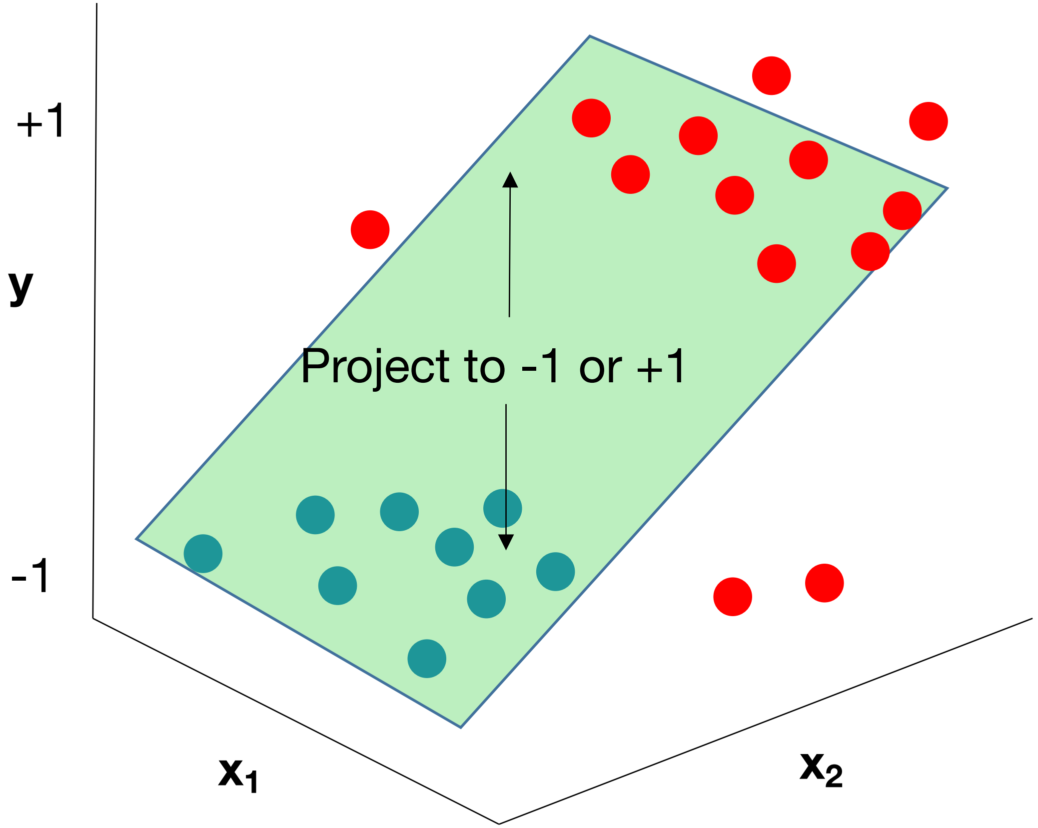
Without sgn(): regression for best fit

$$f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i$$

$$L = \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

- y_i can only be -1 or +1, which defines its class

Why does linear regression with sgn() achieve classification?



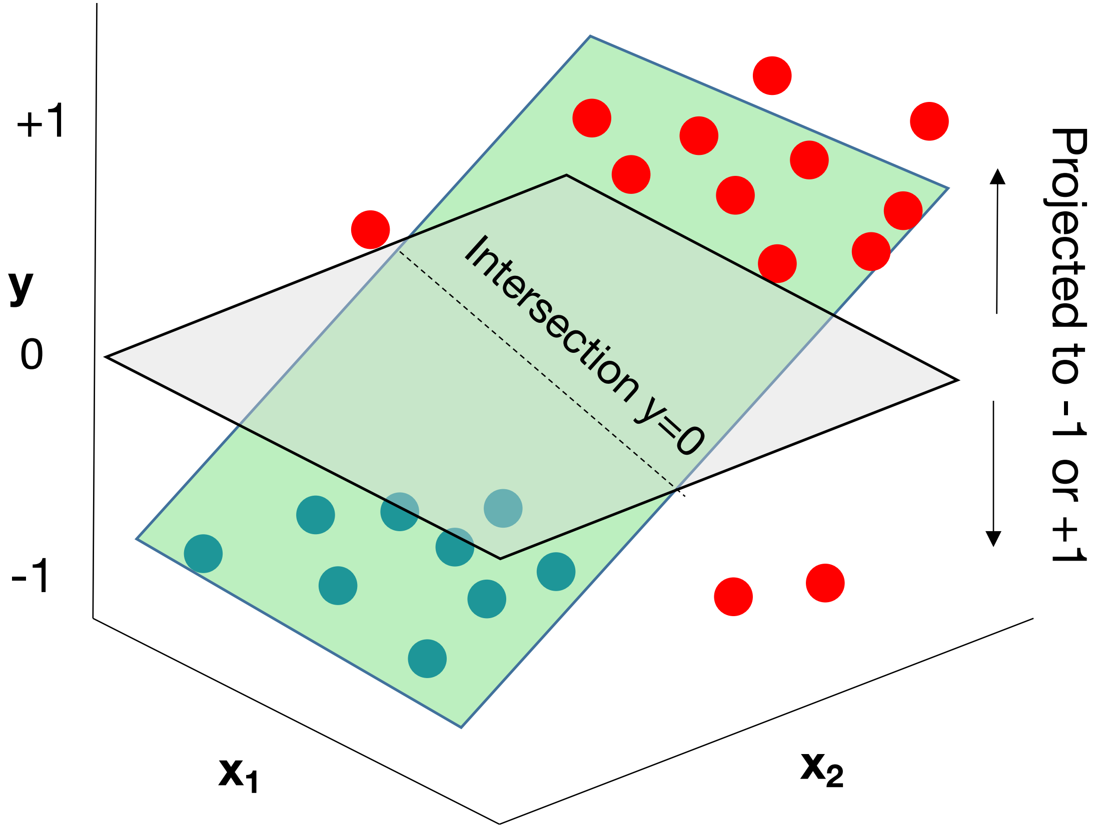
Without sgn(): regression for best fit

$$f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i$$

$$L = \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

- y_i can only be -1 or +1, which defines its class
- Can still find plane of best fit

Why does linear regression with sgn() achieve classification?



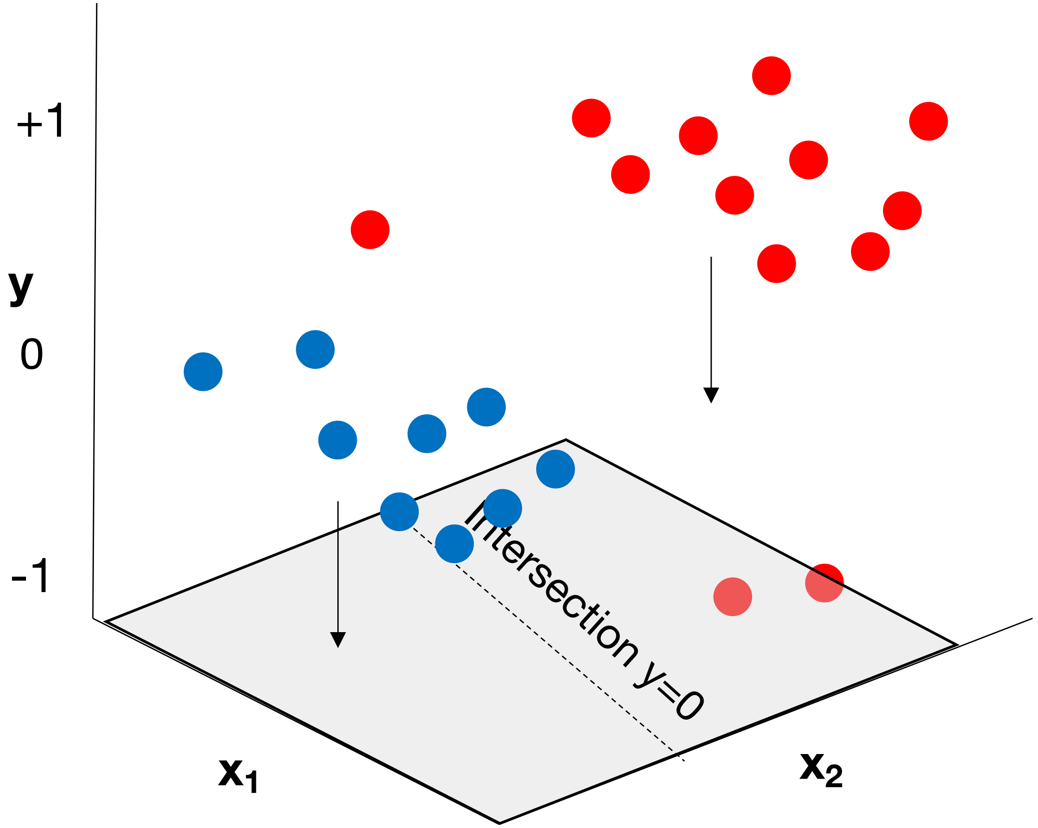
With sgn() operation:

$$f(\mathbf{x}_i) = y_i^* = \text{sgn}(\mathbf{w}^T \mathbf{x}_i)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

- Anything point to one side of y=0 intersection is class +1, anything on the other side of intersection is class -1

Why does linear regression with sgn() achieve classification?



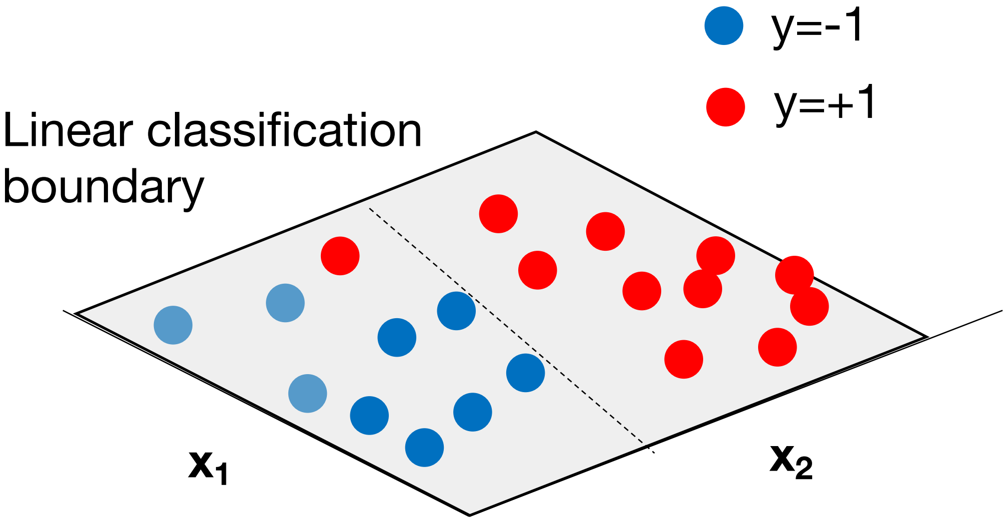
With sgn() operation:

$$f(\mathbf{x}_i) = y_i^* = \mathrm{sgn}(\mathbf{w}^T \mathbf{x}_i)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

- y axis isn't really needed now & can view this decision boundary in 2D

Why does linear regression with sgn() achieve classification?

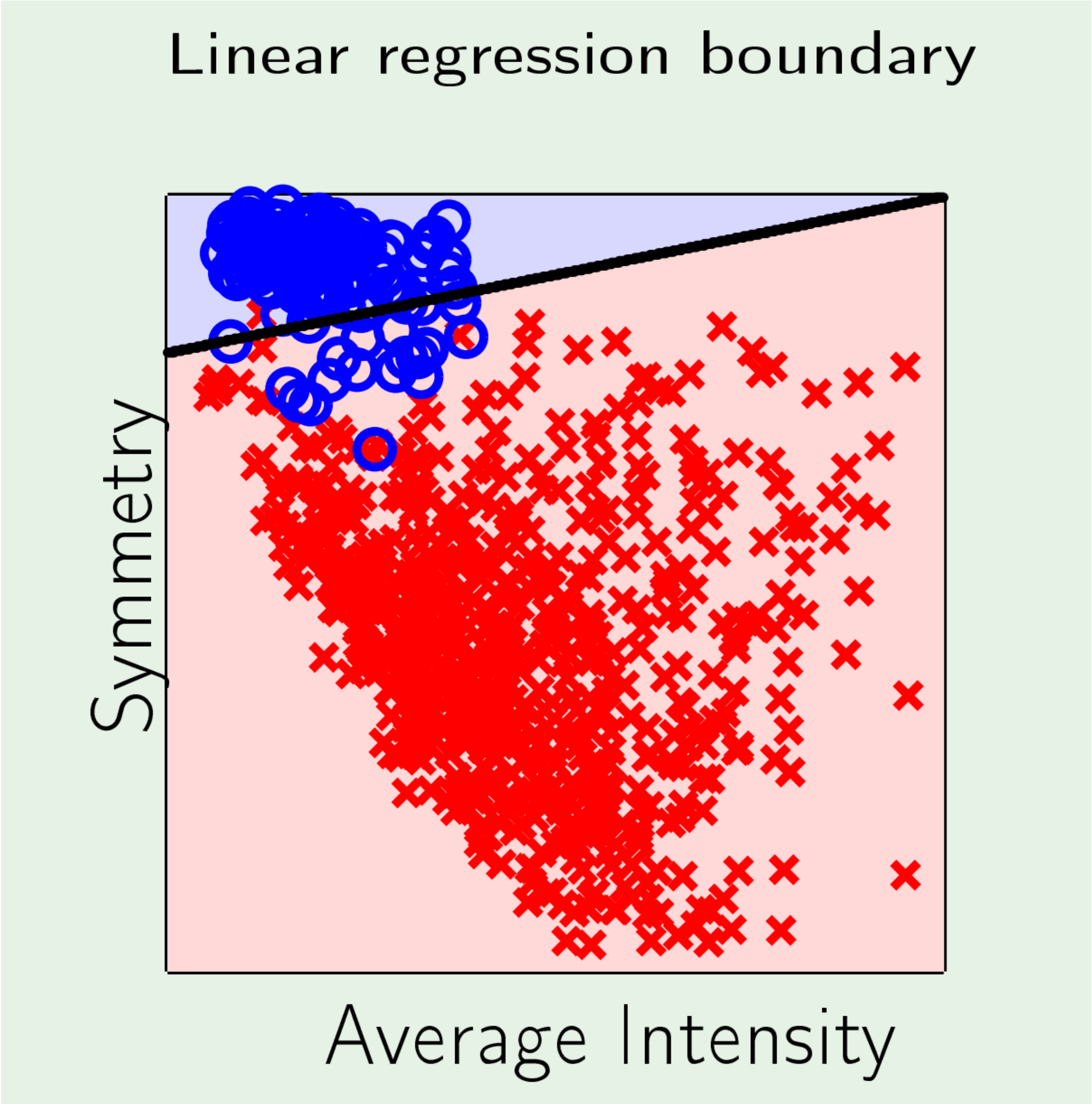


With sgn() operation:

$$f(\mathbf{x}_i) = y_i^* = \mathrm{sgn}(\mathbf{w}^T\mathbf{x}_i)$$

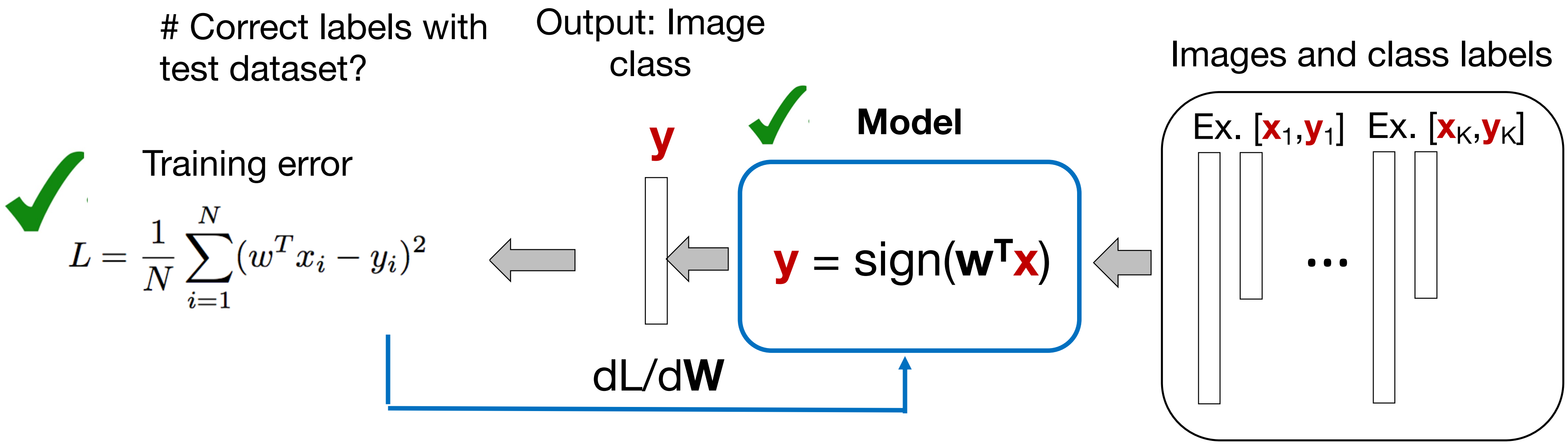
$$L = \frac{1}{N}\sum_{i=1}^{N}(w^T x_i - y_i)^2$$

Sign operation takes linear regression and makes it a classification operation!



Caltech Learning from Data: https://work.caltech.edu/telecourse.html

Example: machine learning for image classification



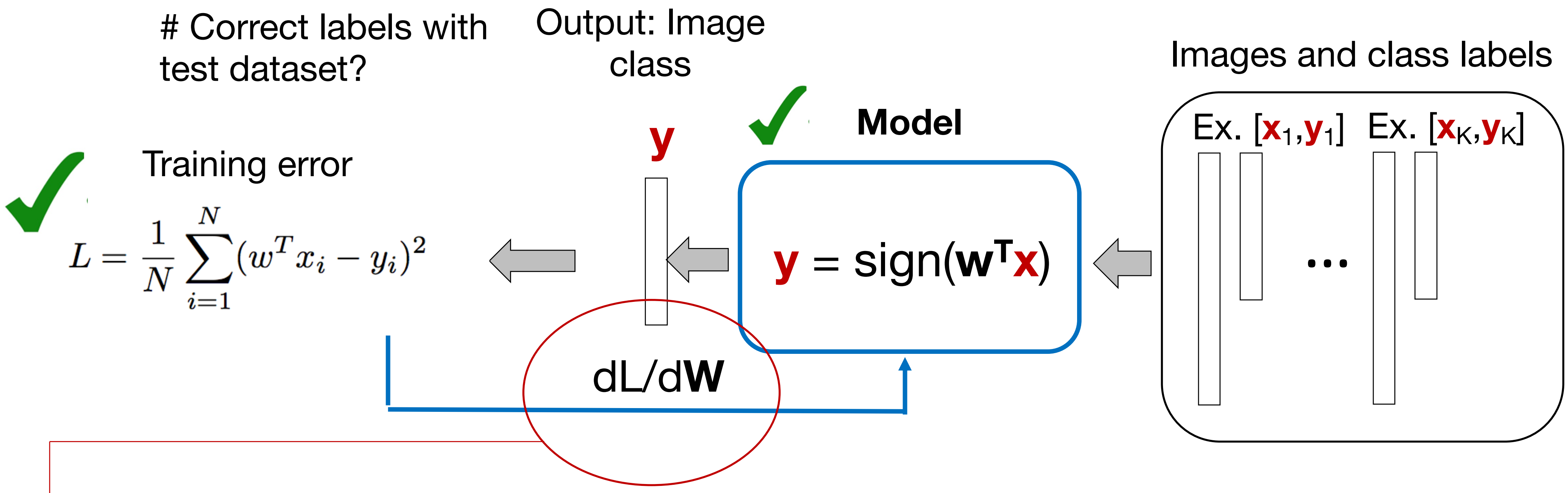
Let's consider a simple example – image classification. What do we need for training?

1. Labeled examples

2. A model and loss function

3. A way to minimize the loss function L

Example: machine learning for image classification



Let’s consider a simple example – image classification. What do we need for training?

1. Labeled examples

2. A model and loss function

3. A way to minimize the loss function L

3 methods to solve for w^T in the case of linear regression:

(easier)

1. Pseudo-inverse (this is one of the few cases with a closed-form solution)
2. Numerical gradient descent
3. Gradient descent on the cost function with respect to W

(harder)

deep imaging

3 methods to solve for w^T in the case of linear regression:

(easier)

1. Pseudo-inverse (this is one of the few cases with a closed-form solution)
2. Numerical gradient descent
3. Gradient descent on the cost function with respect to W

(harder)

Next class: We'll talk more about gradient descent methods!