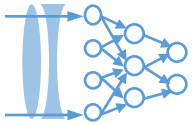


Lecture 6: Ingredients for Machine Learning

Machine Learning and Imaging

BME 590L
Roarke Horstmeyer

Outline

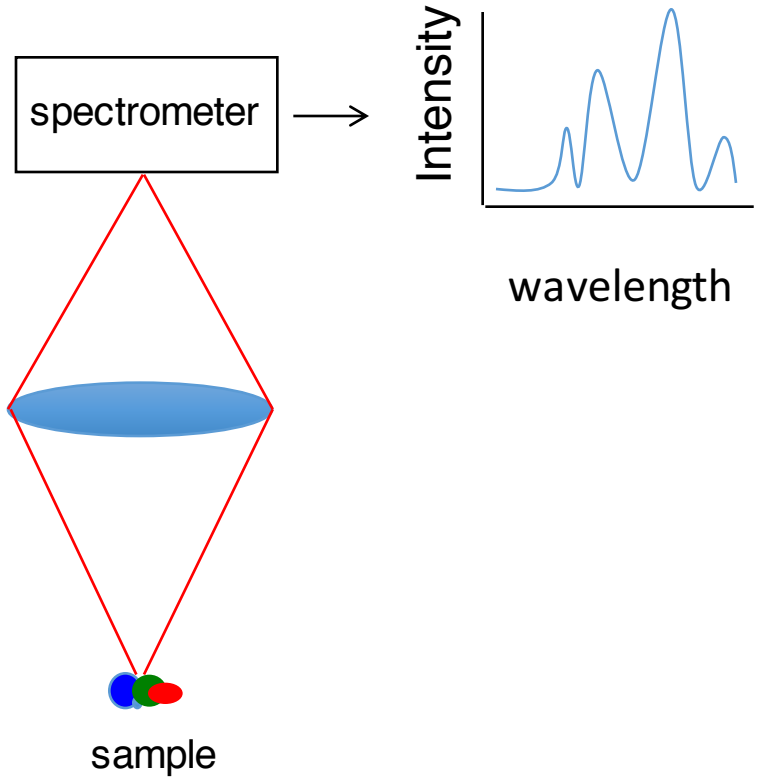
- Review spectral unmixing (last class)
- From optimization to machine learning
- Ingredients for ML
- Example: linear classification of images
  - Train/test data
  - Linear regression model
  - 3 ways to solve

Last time: simple example of spectral unmixing

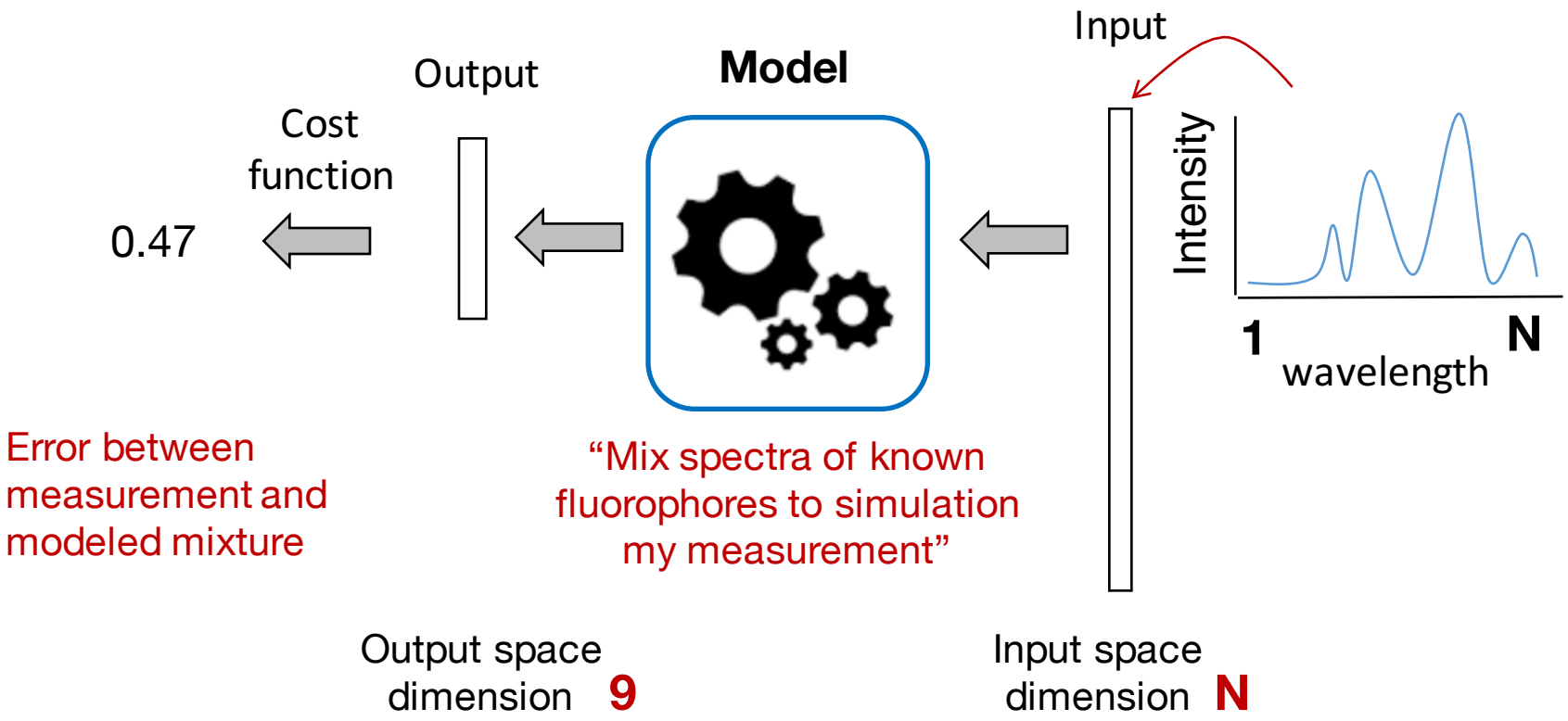
(For whatever reason, whenever I get confused about optimization, I think about this example)

The setup:

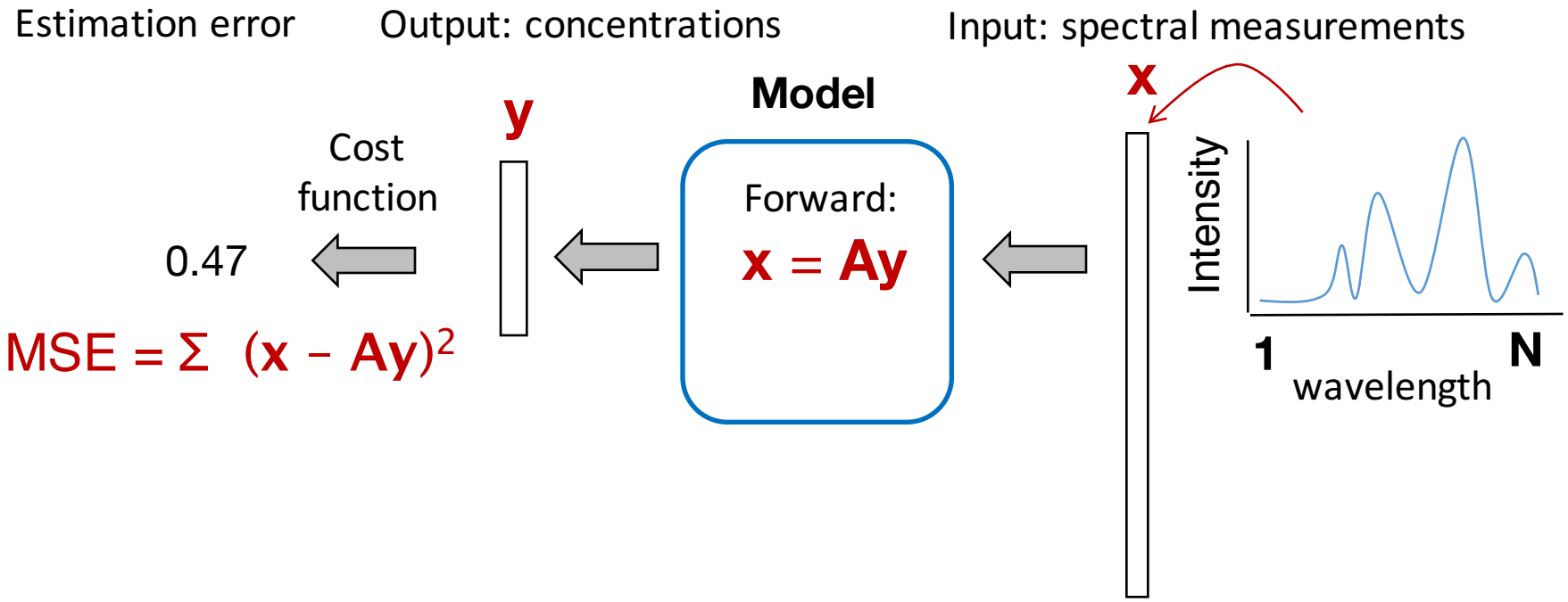
- measure the color (spectral) response of a sample (e.g., how much red, green and blue there is, or several hundred measurements of its different colors).
- You know that the sample can only contain 9 different fluorophores.
- What % of each fluorophores is in your sample?



Optimization pipeline for spectral unmixing

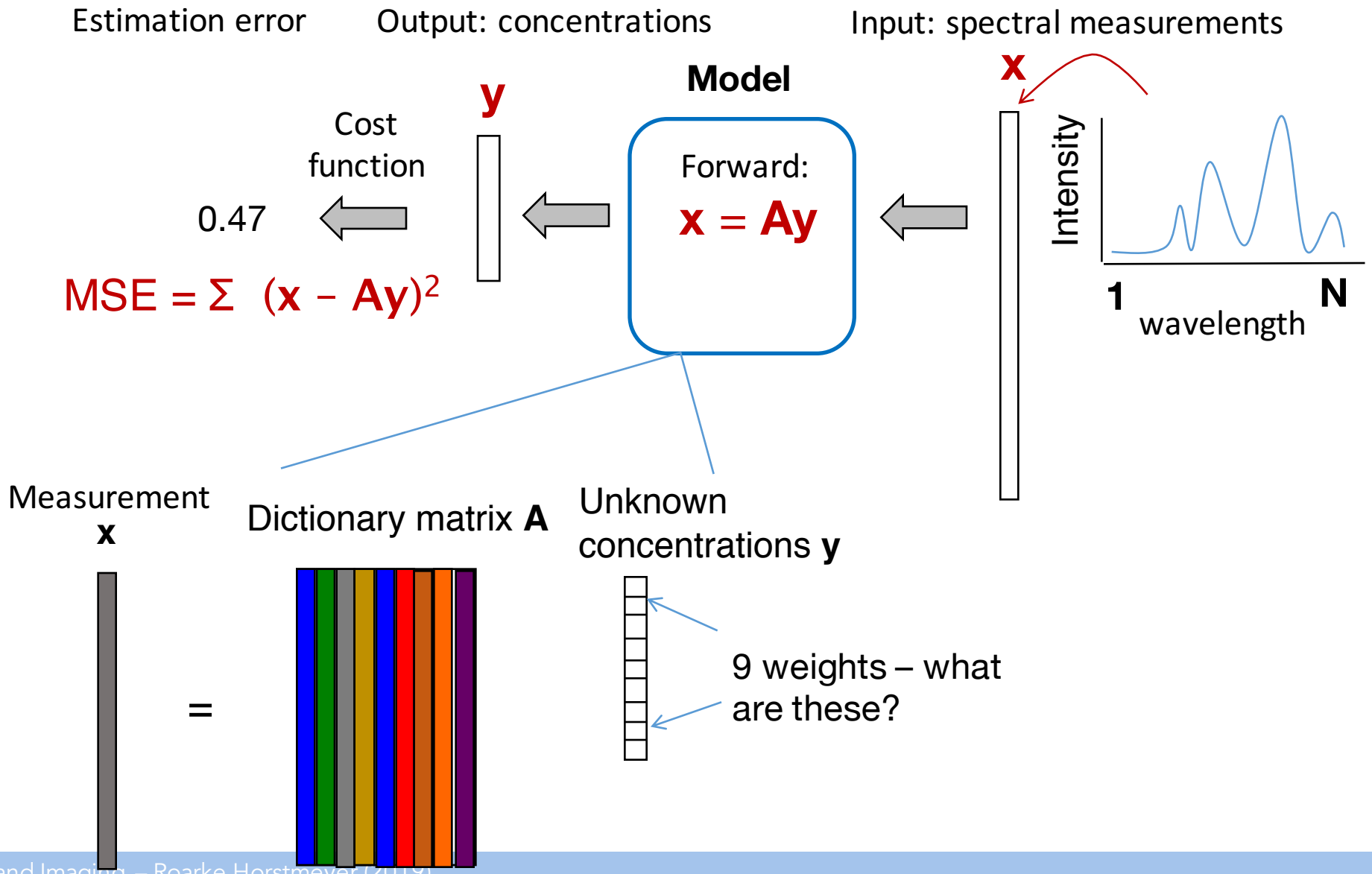


Optimization pipeline for spectral unmixing

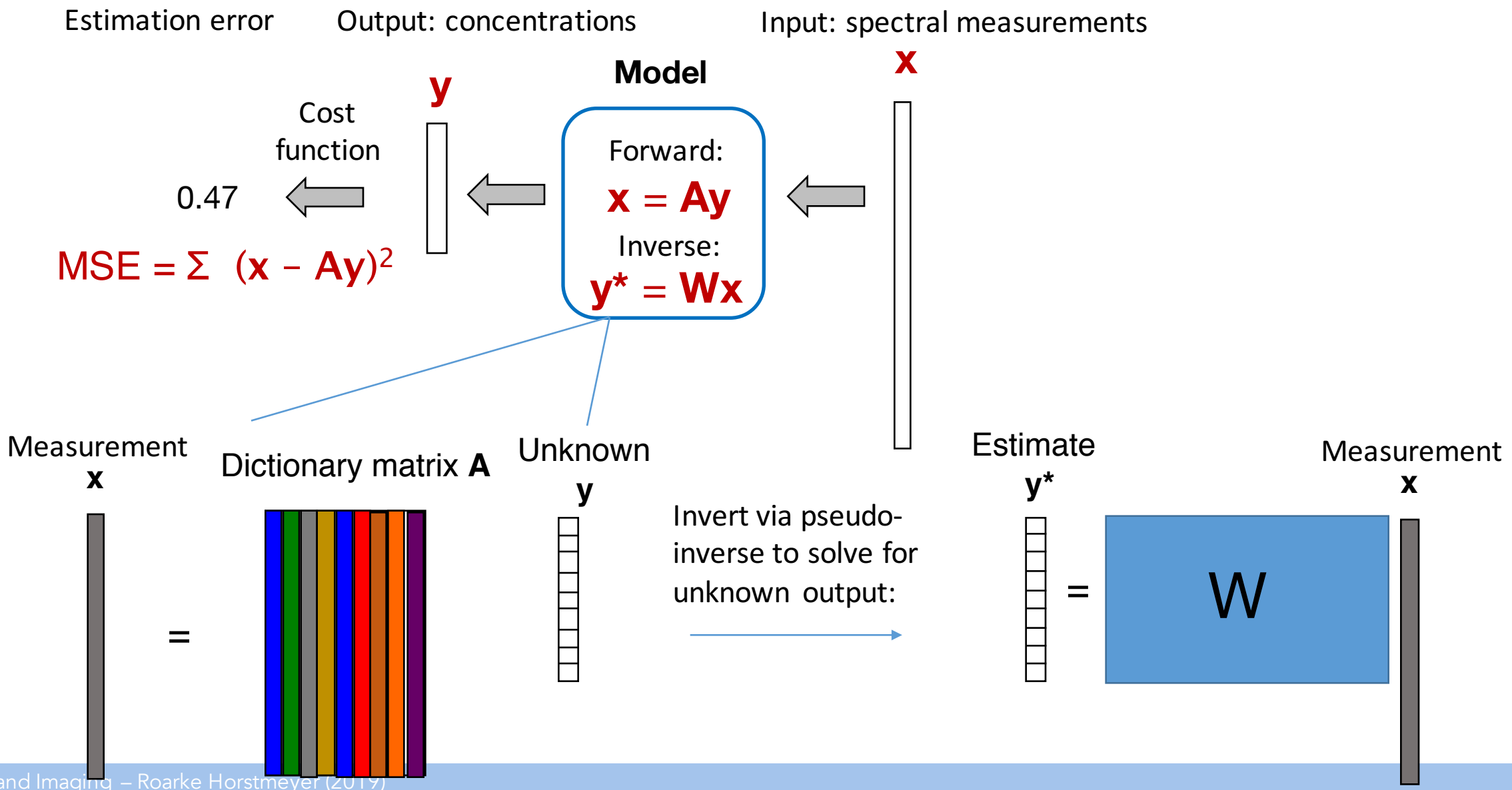


*Note: notation changed from last time to be consistent with we'll use in the future

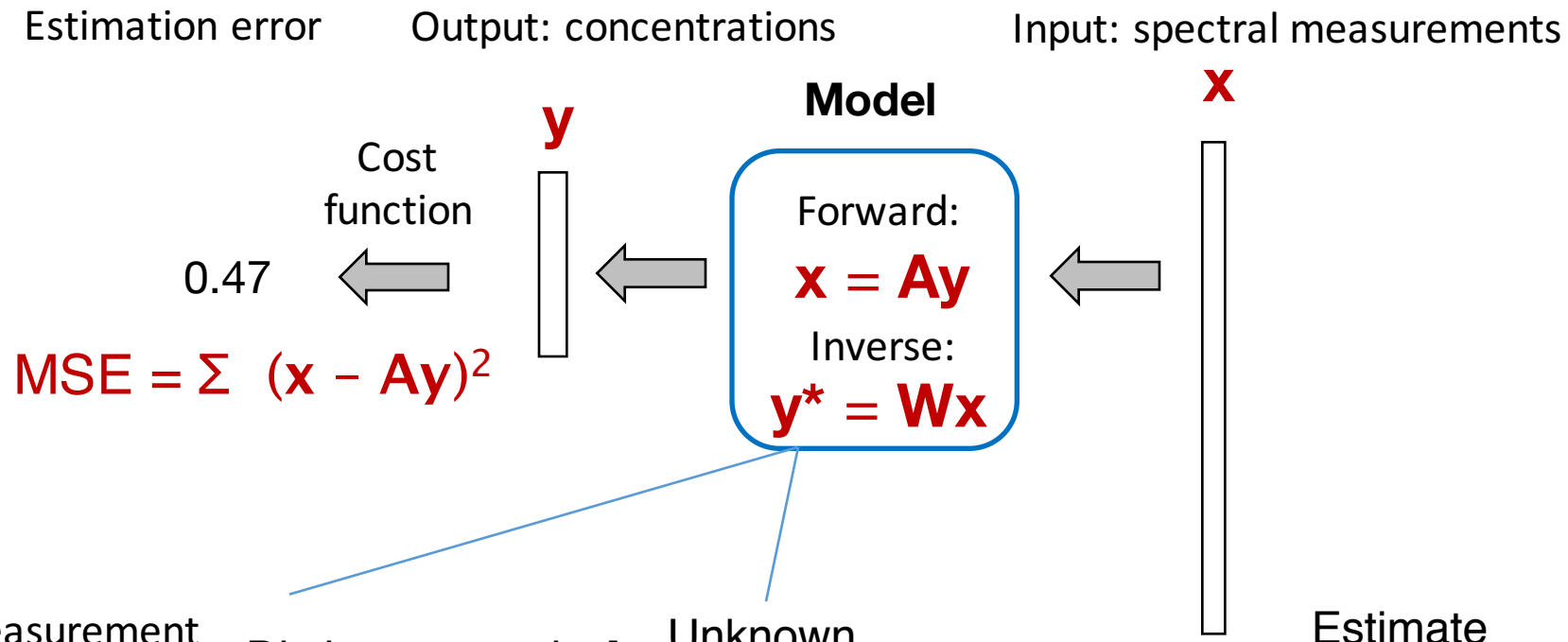
Optimization pipeline for spectral unmixing



Optimization pipeline for spectral unmixing



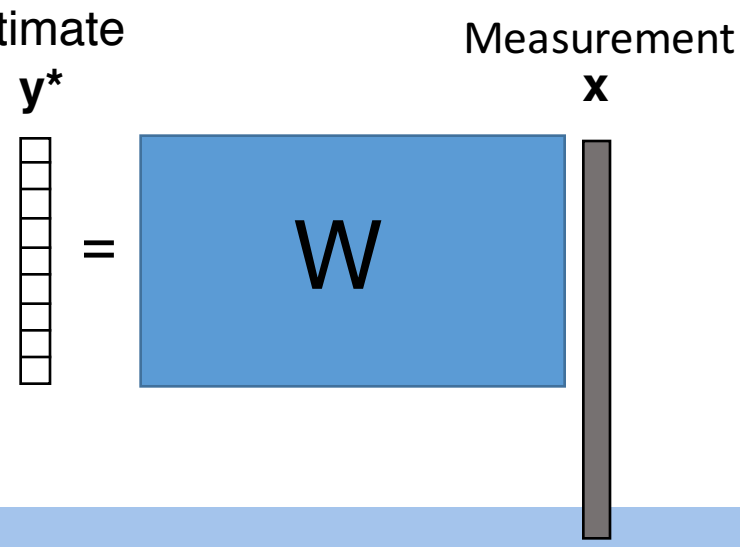
Optimization pipeline for spectral unmixing



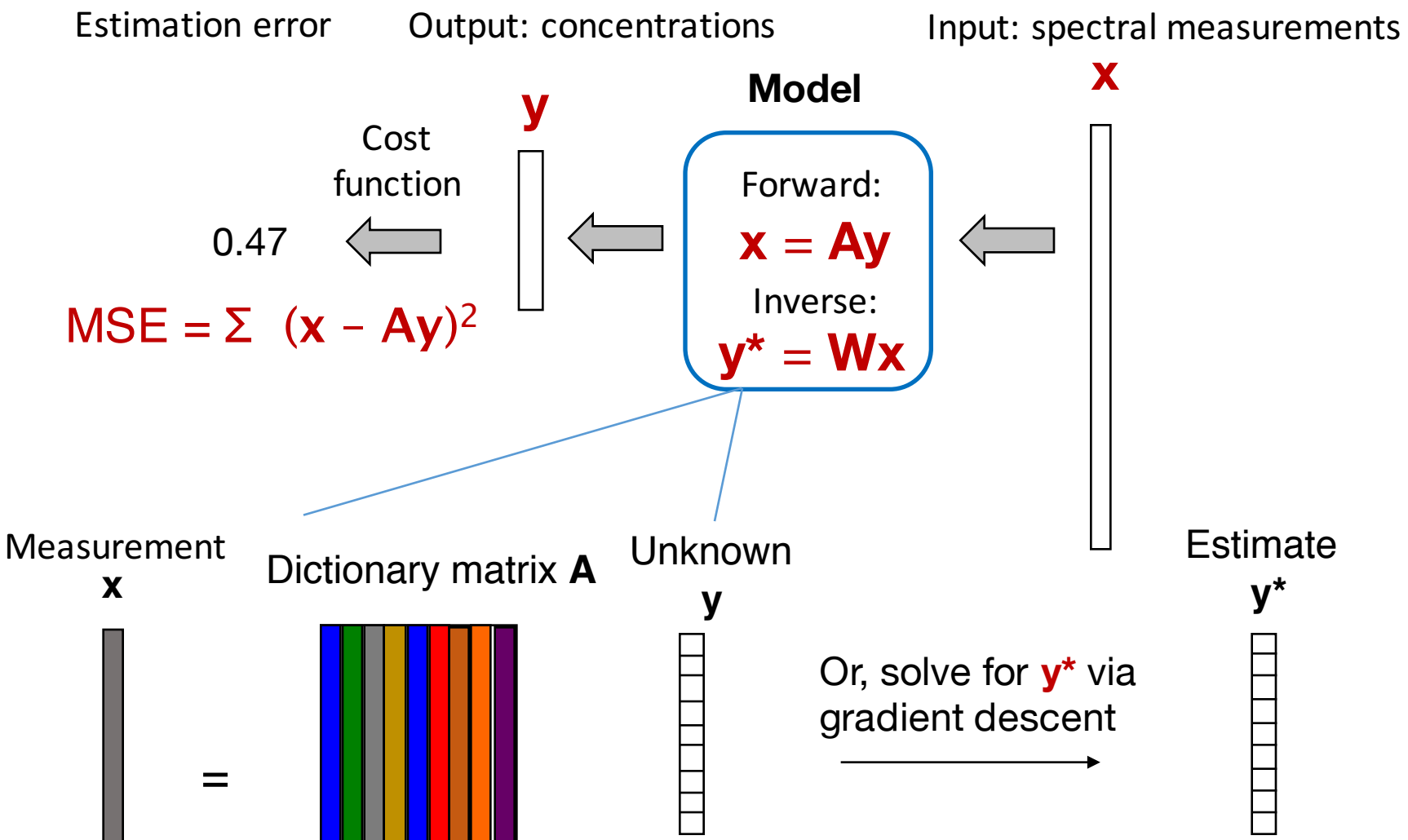
Invert via pseudo-inverse to solve for unknown output:

$$W = (A^T A)^{-1} A^T$$

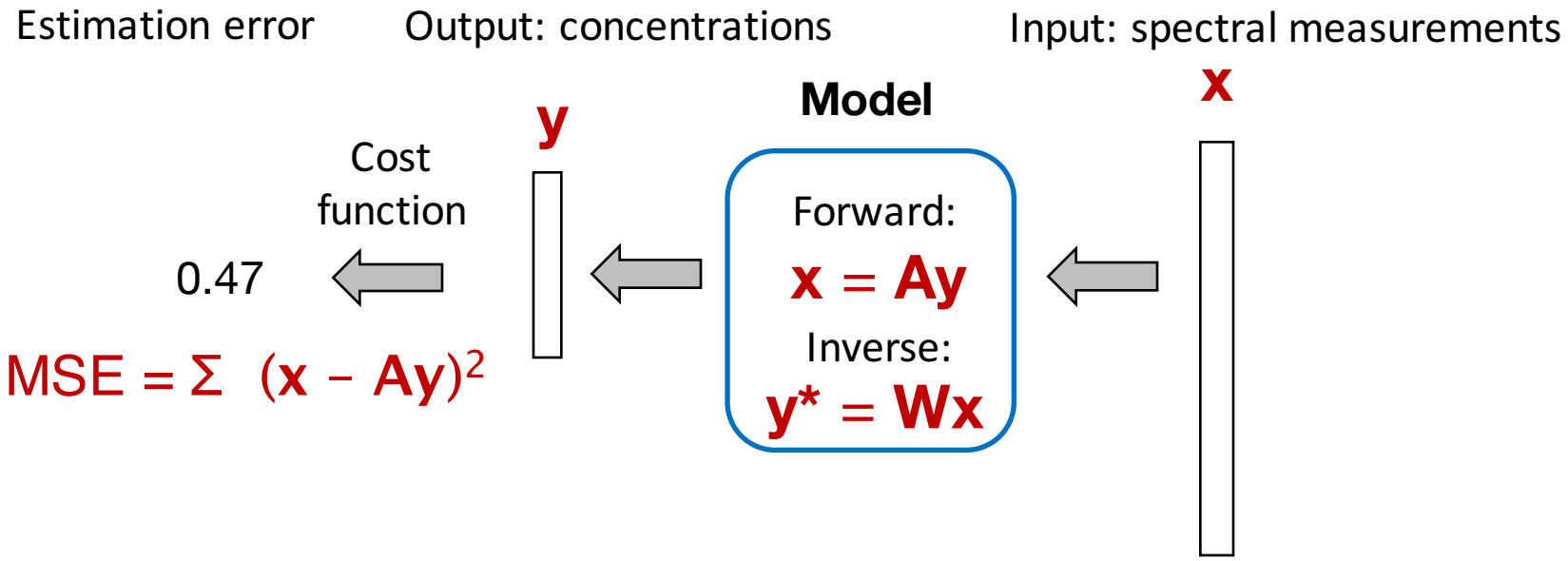
$$y^* = Wx$$



Optimization pipeline for spectral unmixing

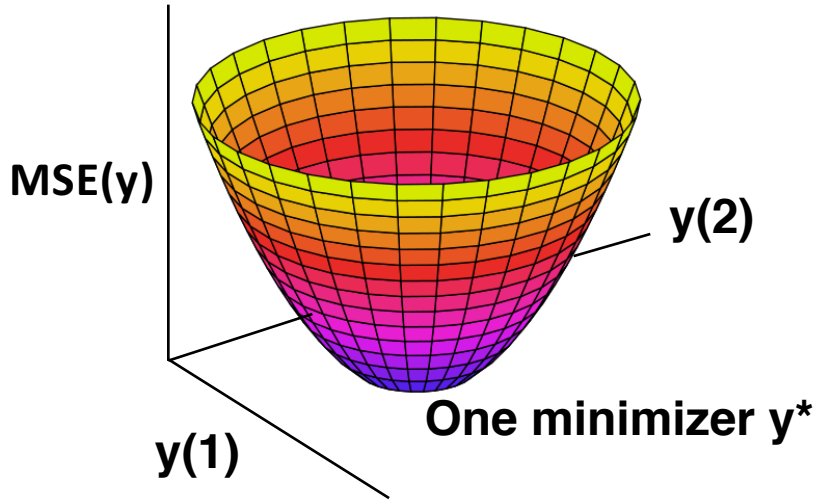


Optimization pipeline for spectral unmixing

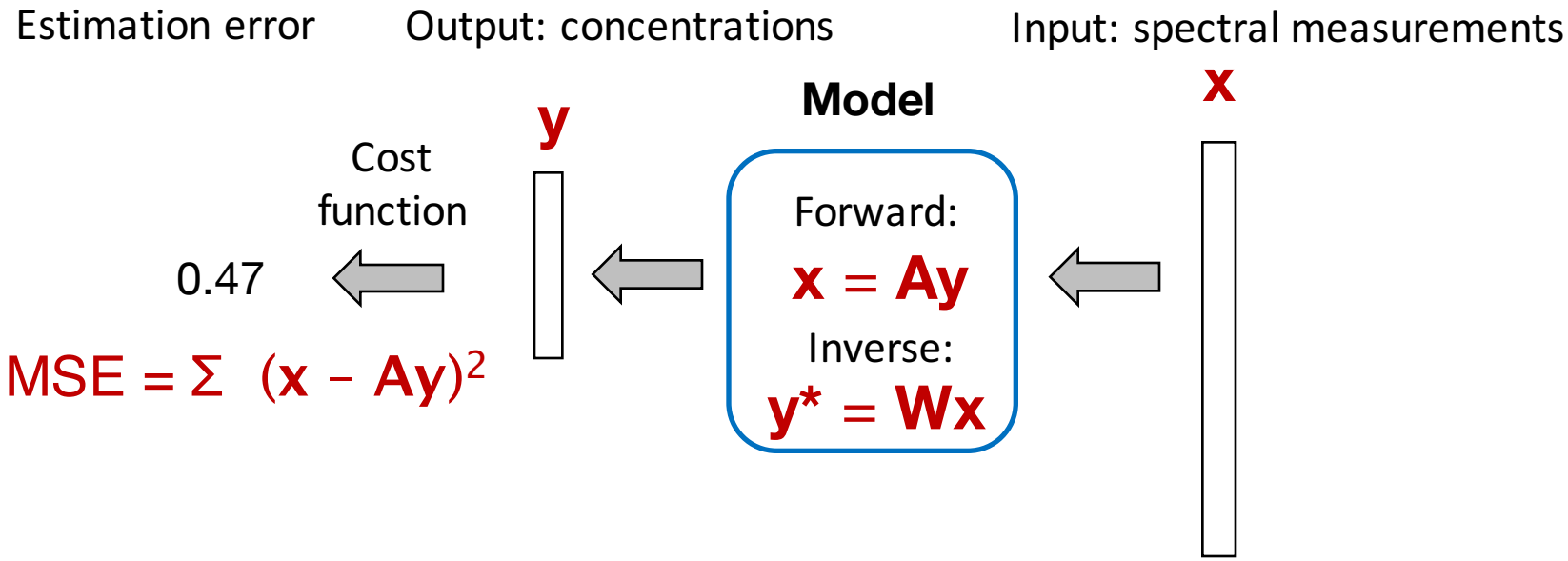


$$MSE(\mathbf{y}) = \sum_{\text{spectral measurements}} (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

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

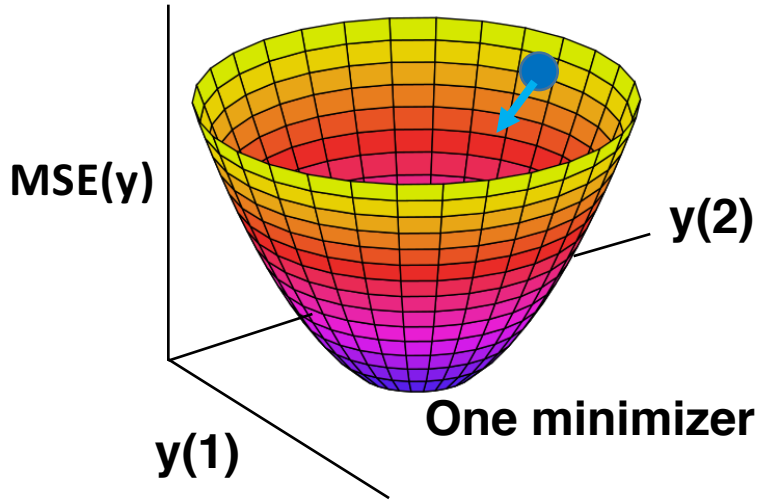


Optimization pipeline for spectral unmixing



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

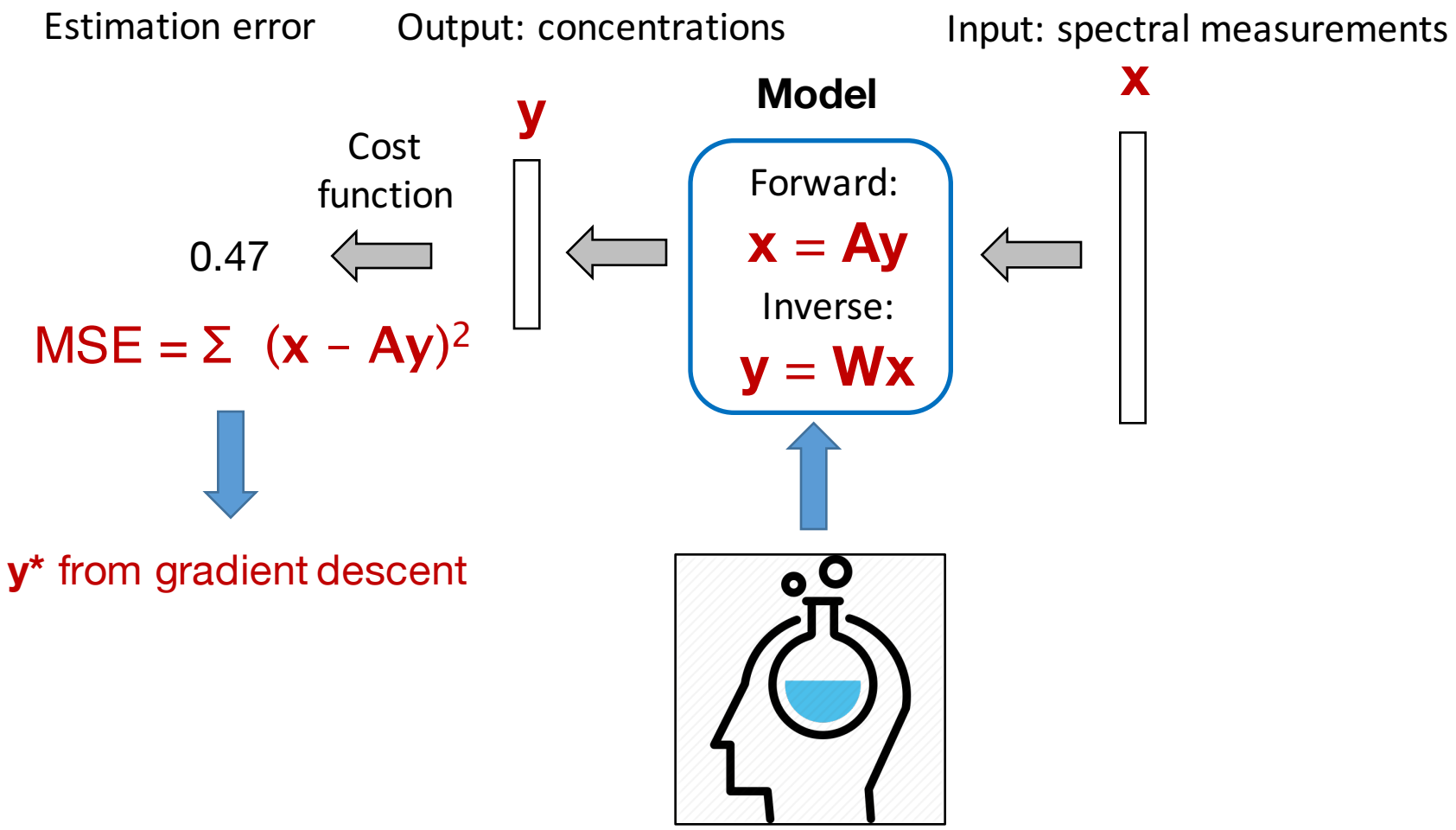
$$d/d\mathbf{y}\ MSE = \mathbf{A}^T(\mathbf{x} - \mathbf{Ay})$$



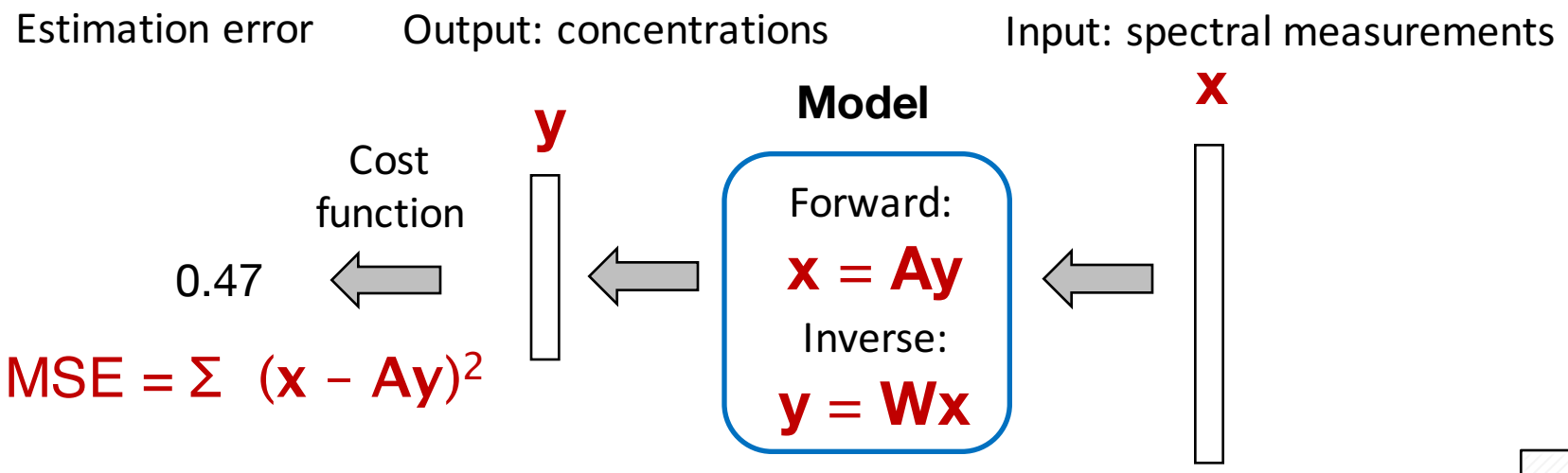
1. Guess output **y**
2. Evaluate slope
3. If its large, take a step:

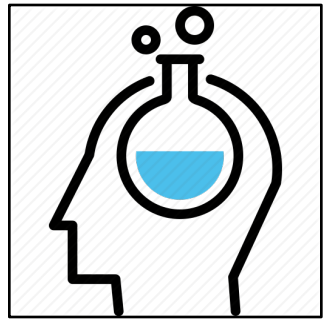
$$\mathbf{y} - \mathbf{y} + \varepsilon\mathbf{A}^T(\mathbf{x} - \mathbf{Ay})$$

Optimization pipeline for spectral unmixing



Optimization pipeline for spectral unmixing

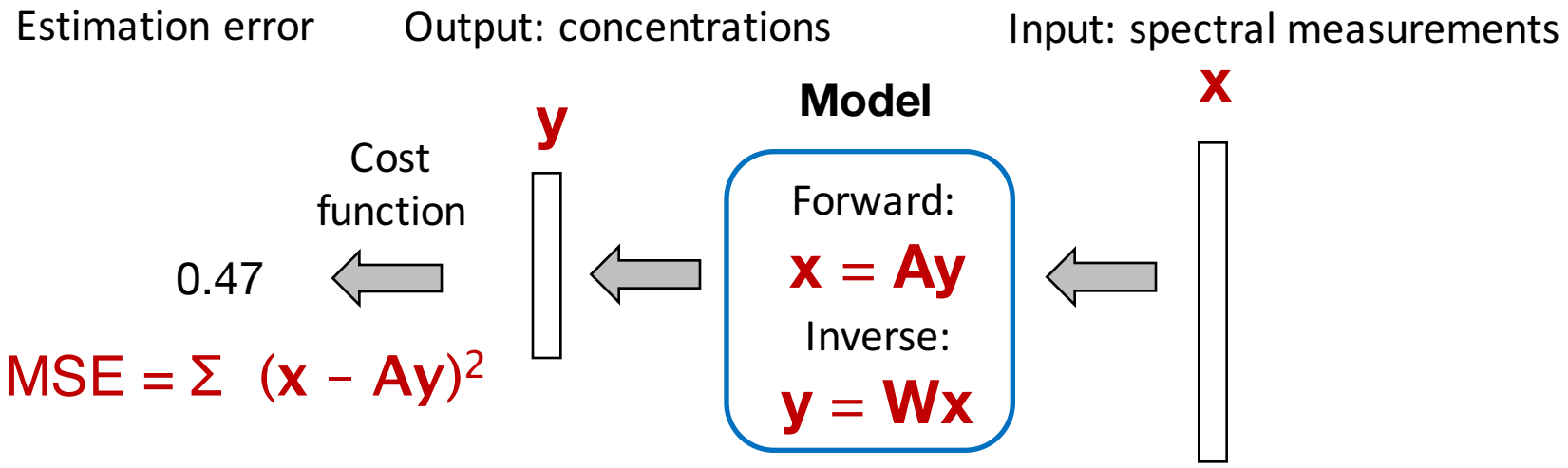




A and **W** from first principles

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

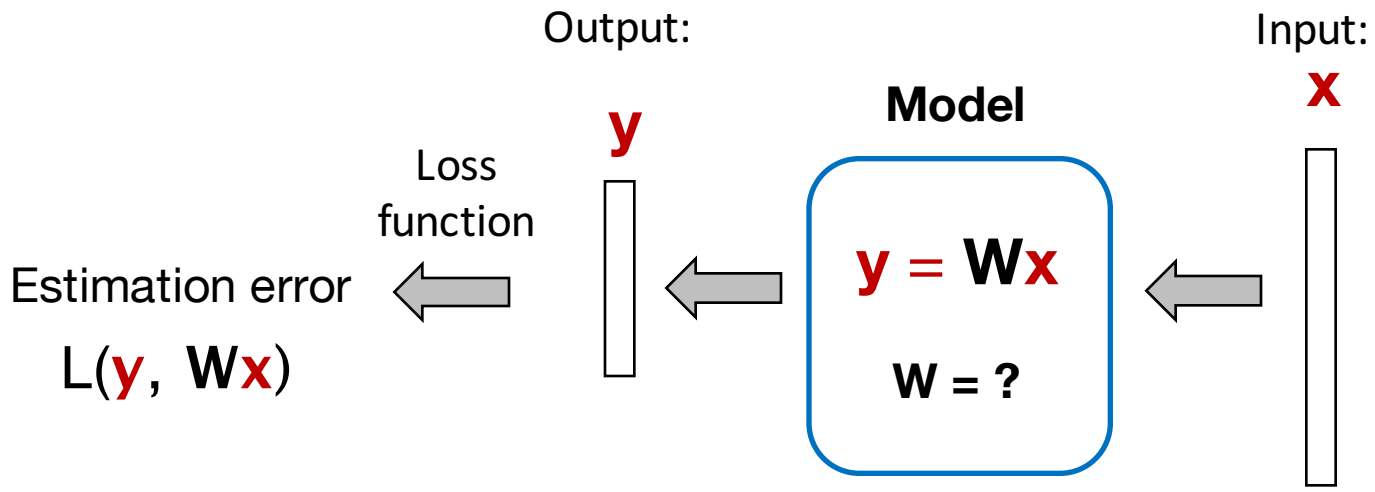
Optimization pipeline for spectral unmixing



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***

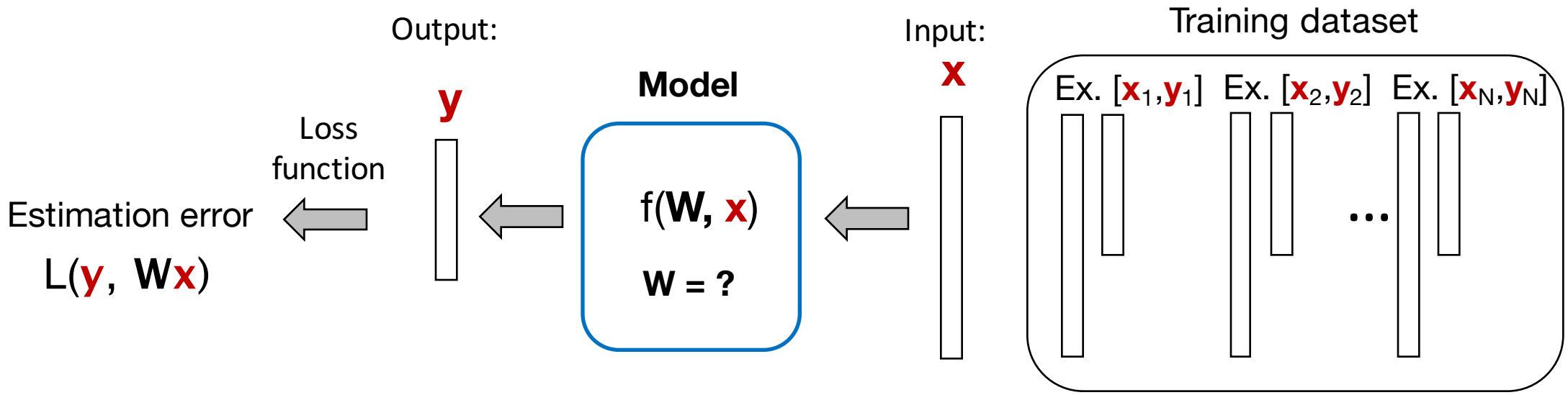
Pipeline for machine learning



Changes for machine learning framework:

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

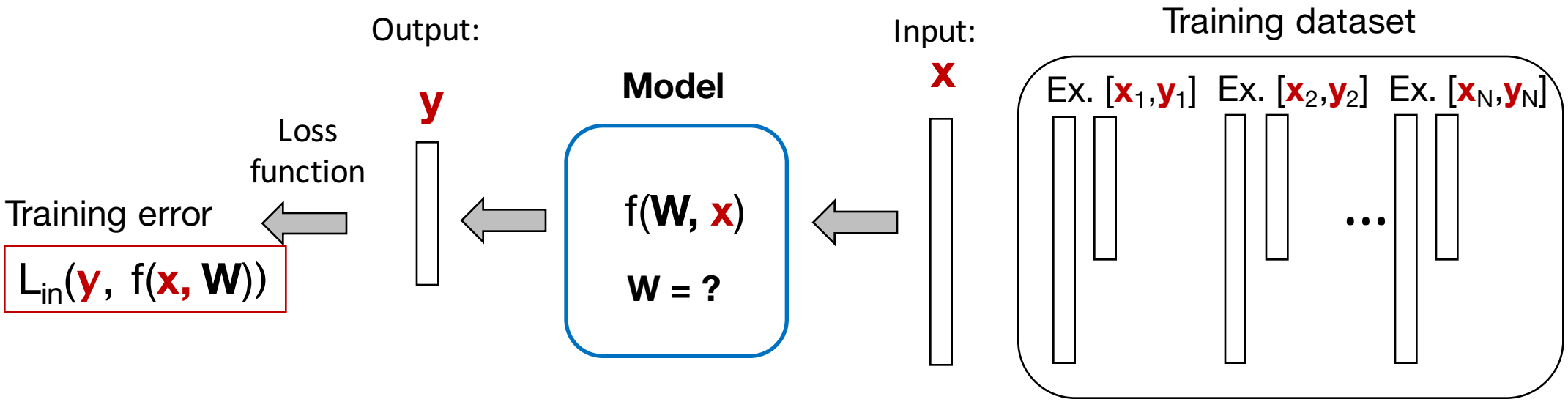
Pipeline for machine learning



Changes for machine learning framework:

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

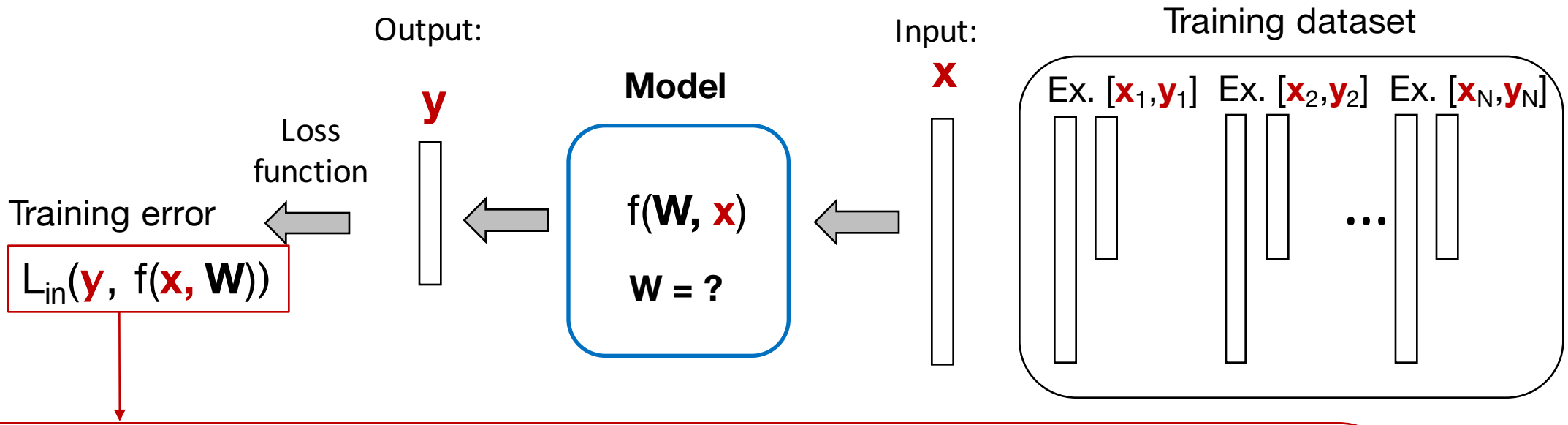
Pipeline for machine learning



Changes for machine learning framework:

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

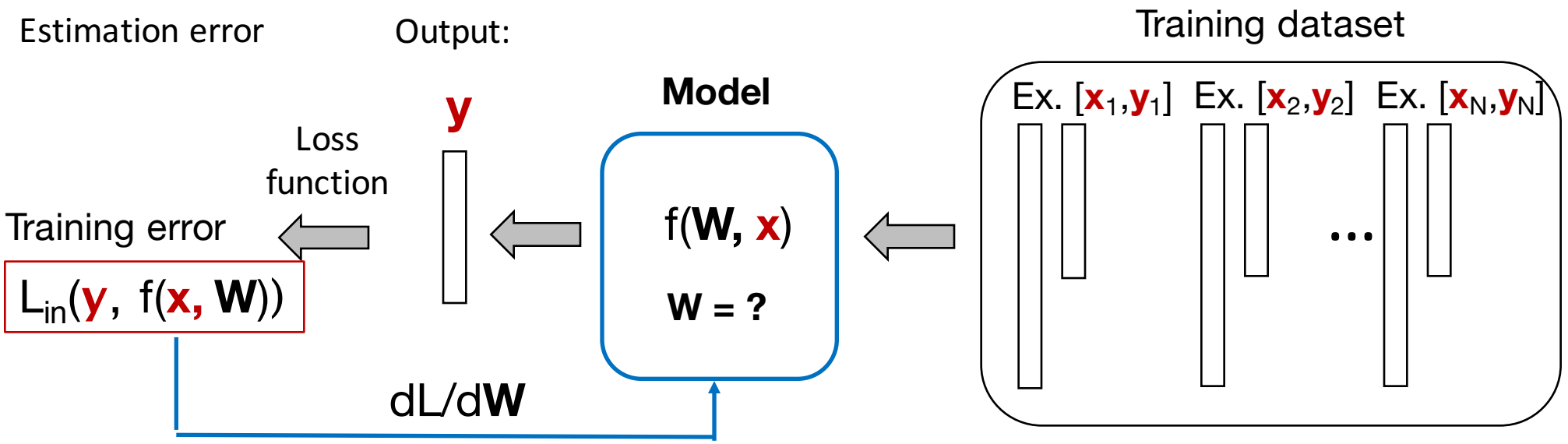


Training Error ("in class error"):

- 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:

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

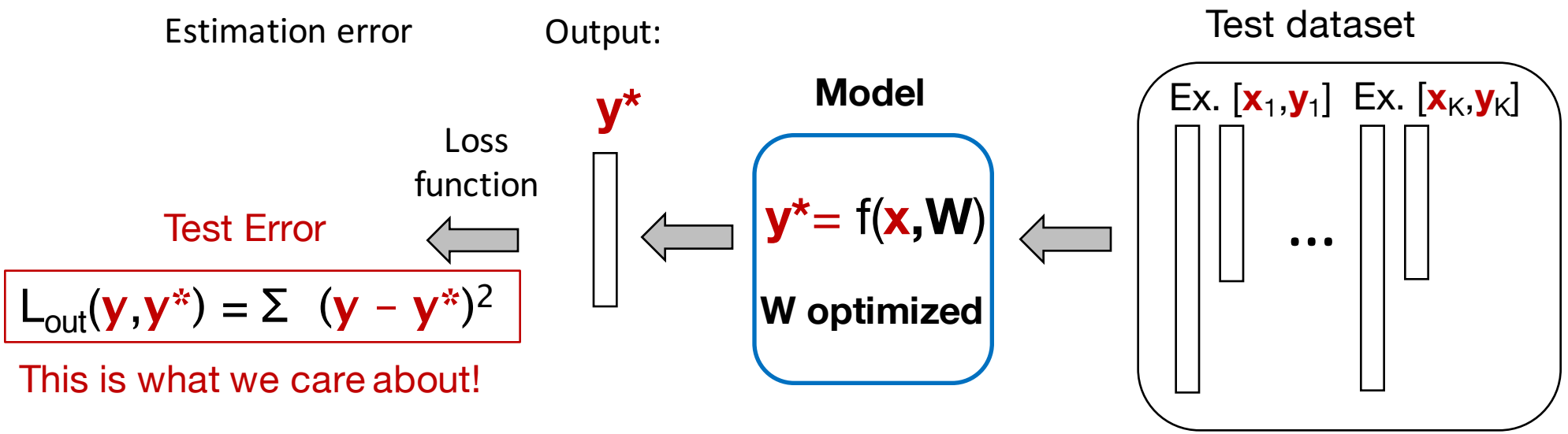
Pipeline for machine learning



Changes for machine learning framework:

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



Changes for machine learning framework:

1. Now just establish the mapping from inputs to outputs (here, matrix **W**)
2. Must first determine mapping (**W**) using large set of "training" data
3. Use a *loss function* L(**x**,**W**) that depends upon the inputs x *and* the model (**W**)
4. Find optimal mapping (**W**), using the training data, guided by gradient descent on L
5. Then, evaluate model accuracy by sending *new* **x** through and comparing output **y*** to "test" data **y**

Example: machine learning for image classification

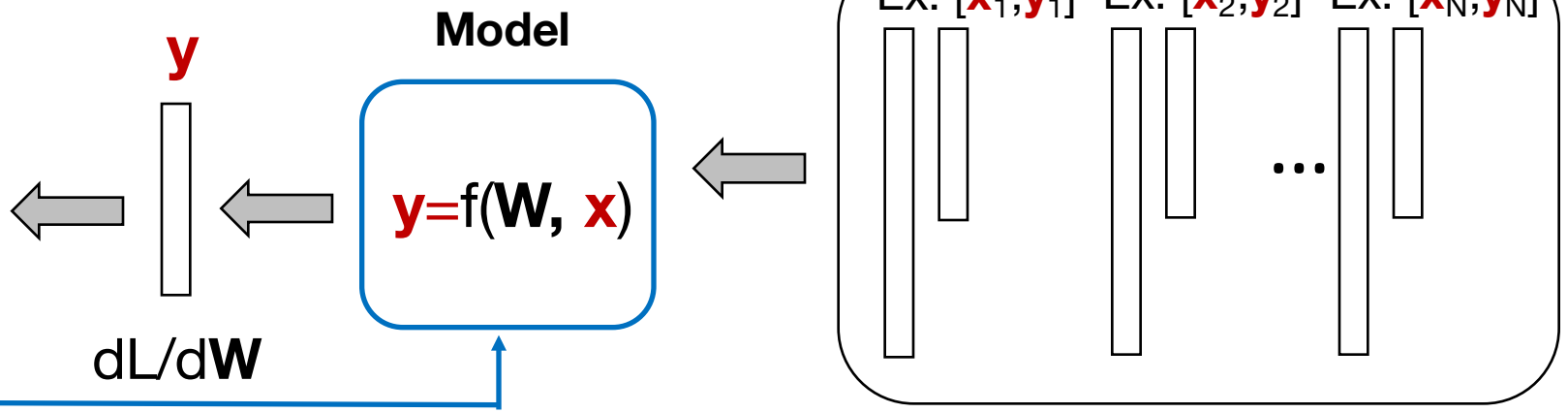
Correct labels for test images?

Output: Image class

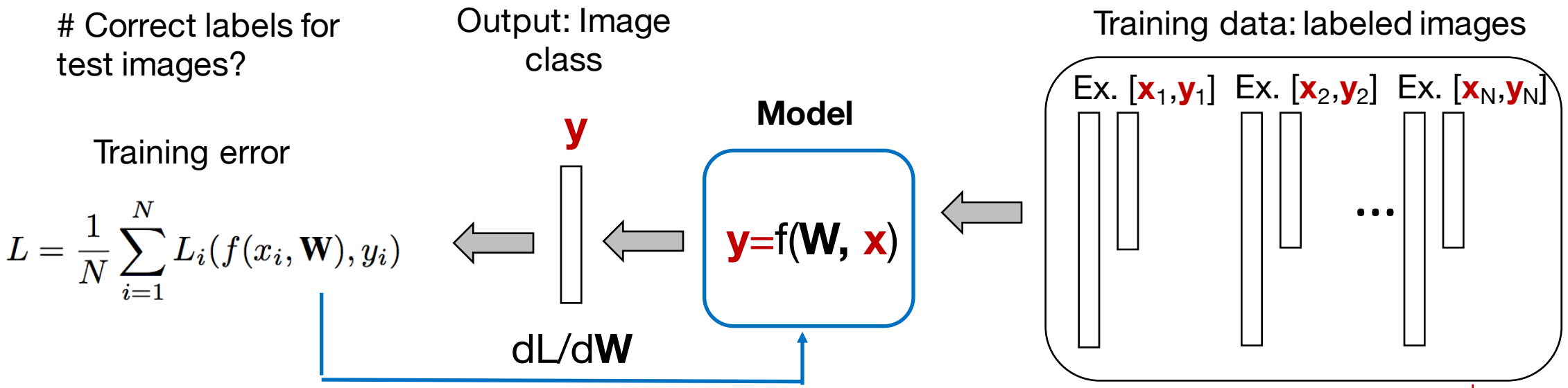
Training data: labeled images

Training error

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



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

M = 28x28 pixel matrix

m = vec[M] = 784-long vector

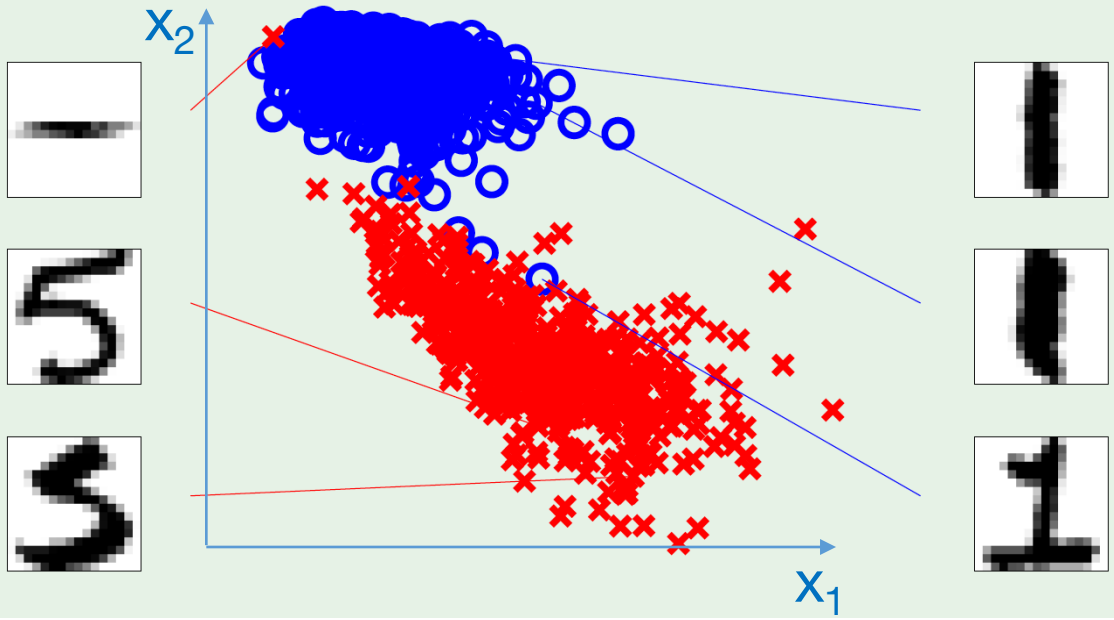
Linear model would require W = 784 element matrix

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

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

Illustration of features

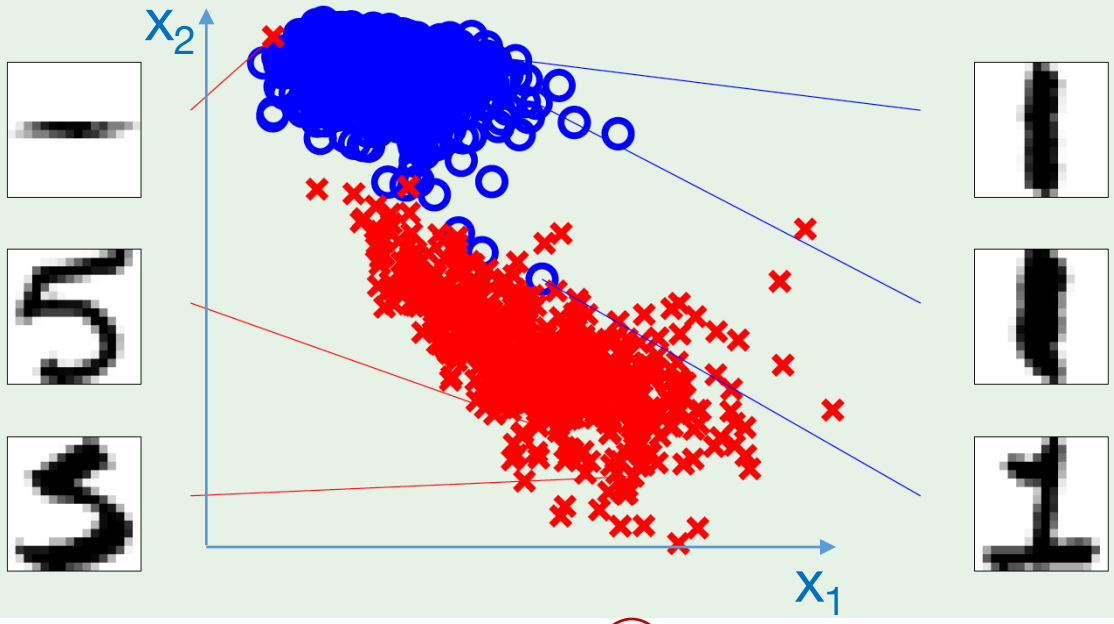
$\mathbf{x} = (x_0, x_1, x_2)$ x_1: intensity x_2: symmetry



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

Illustration of features

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



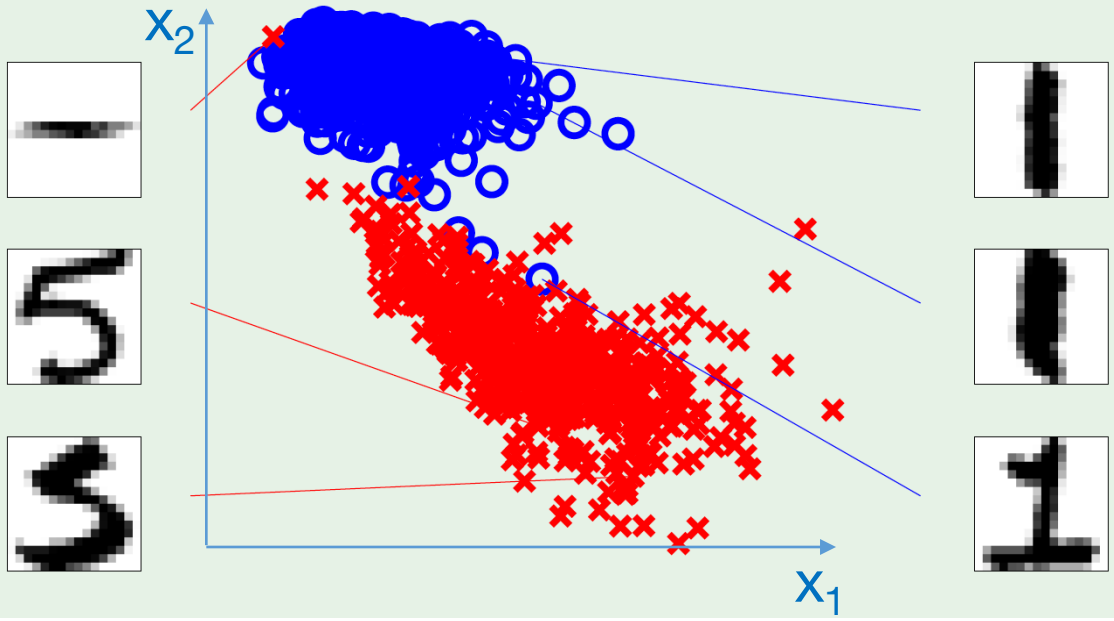
Dataset: 1000 examples of 1's and 5's mapped to $\mathbf{x}_j = (1, x_1, x_2)$, with associated label $y_j = 1$ or -1

This transforms wx+b into **wx**

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

Illustration of features

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



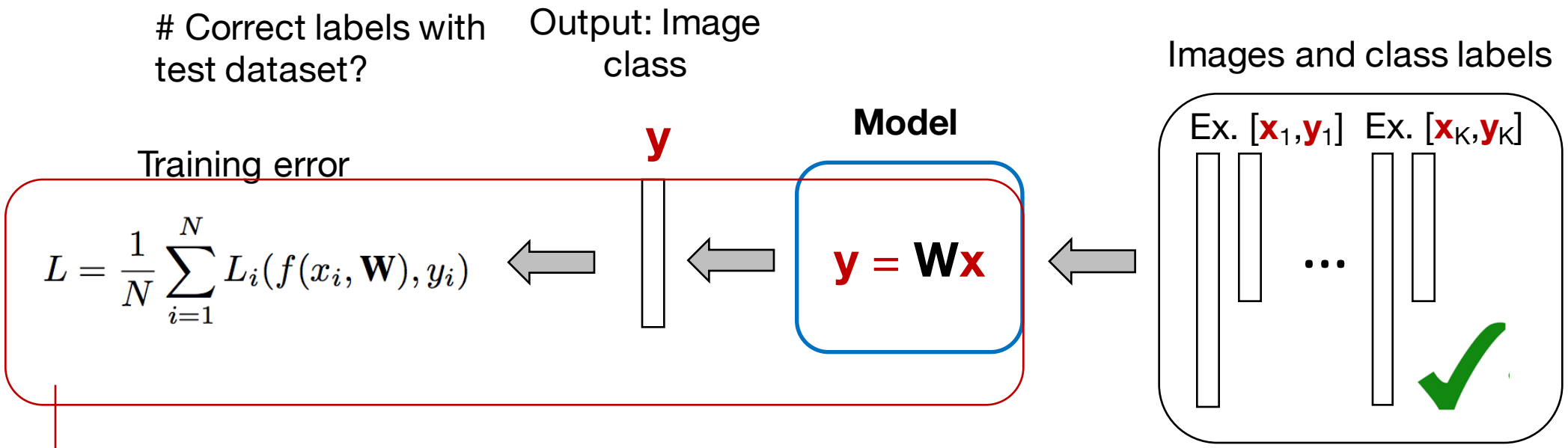
Dataset: 1000 examples of 1's and 5's mapped to $\mathbf{x}_j = (1, x_1, x_2)$, with associated label $y_j = 1$ or -1

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

Labeled data:

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

Example: machine learning for image classification



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

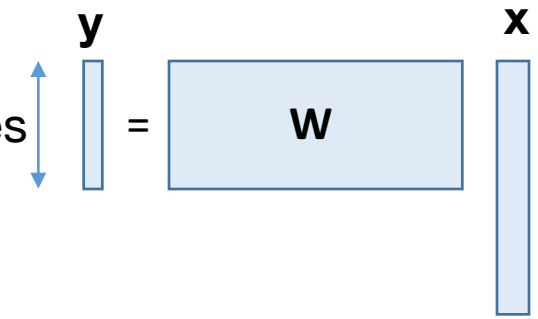
Let's start with a simpler approach: linear regression

deep imaging

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

General linear model:

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



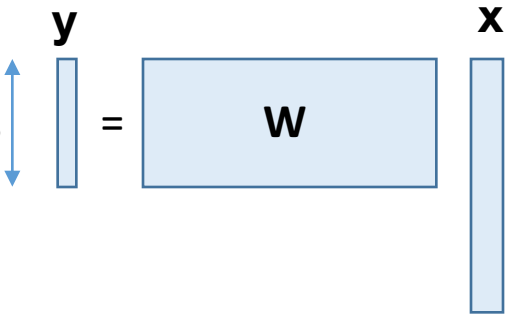
Let's start with a simpler approach: linear regression

deep imaging

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

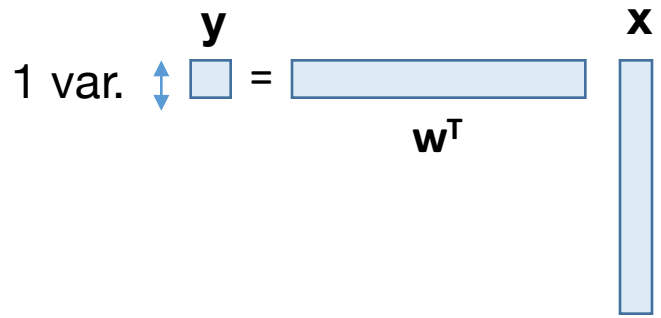
General linear model:

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



Assume 1 class = 1 linear fit

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



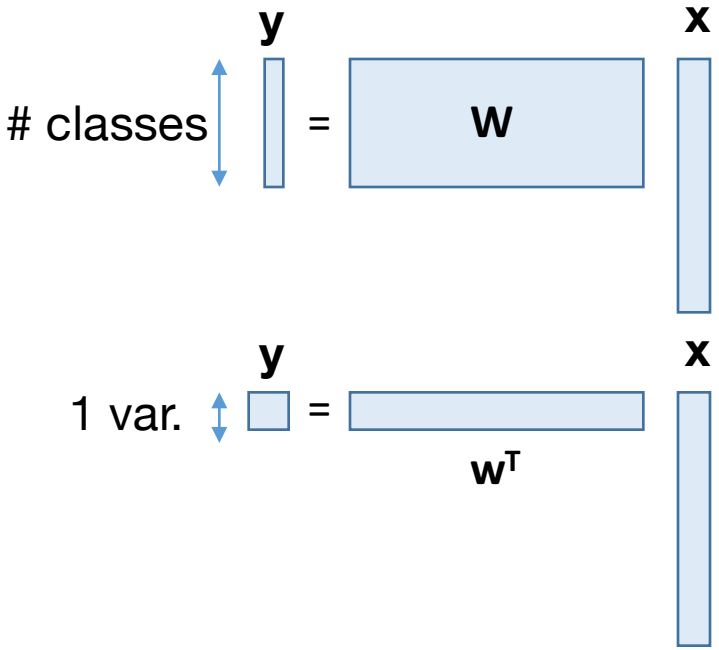
Let's start with a simpler approach: linear regression

deep imaging

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

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

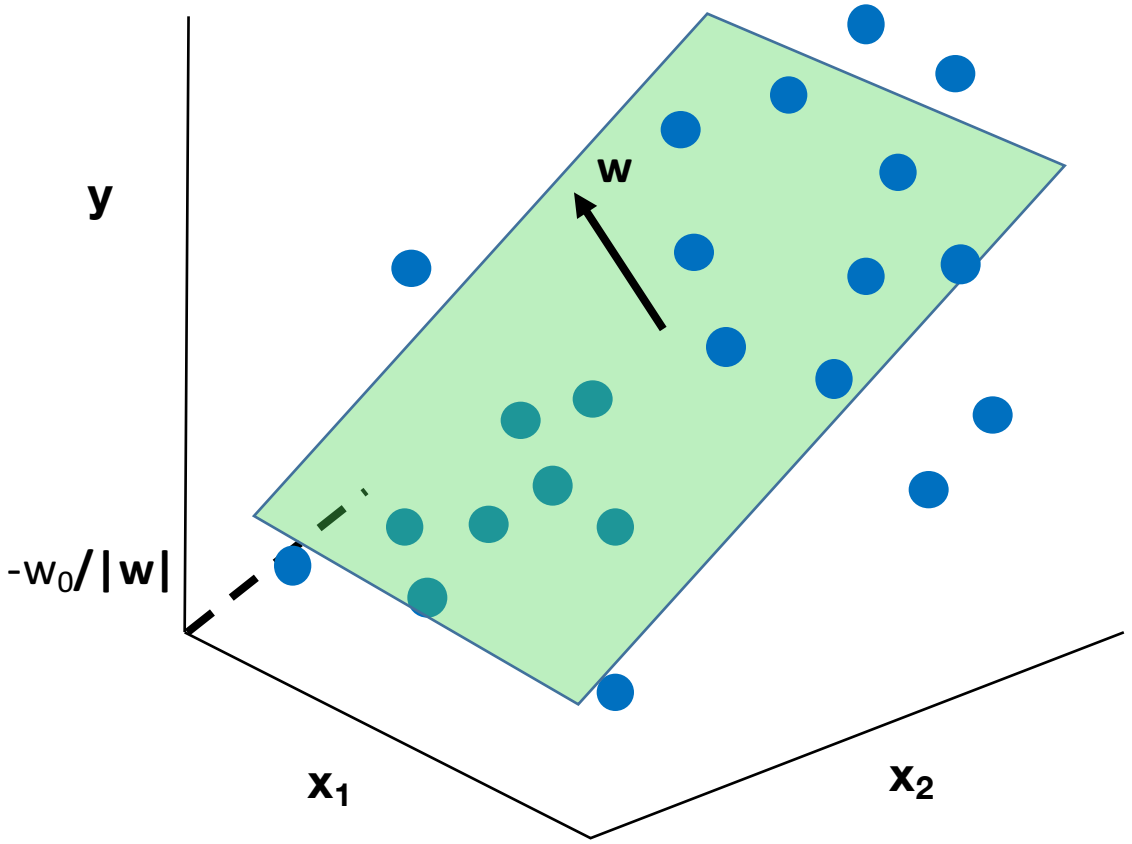
$$L = \frac{1}{N}\sum_{i=1}^{N} (w^T x_i - y_i)^2$$

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?



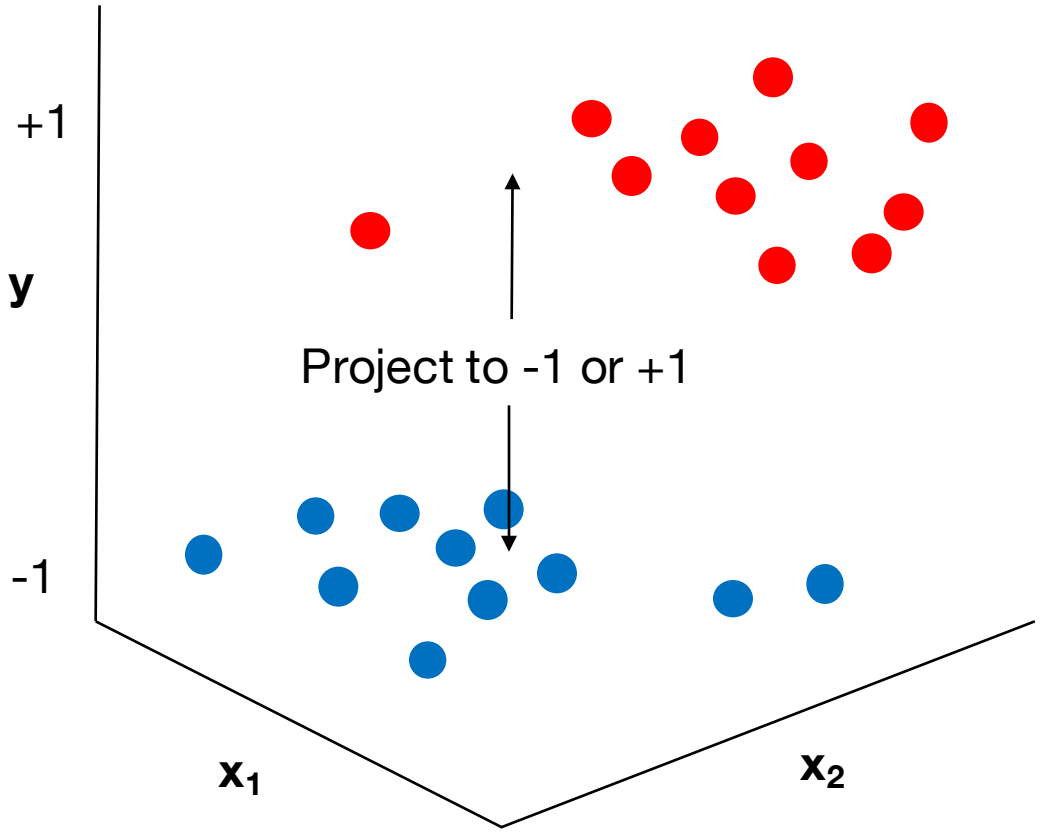
Without sgn(): regression for best fit

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

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

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

Why does linear regression with sgn() achieve classification?



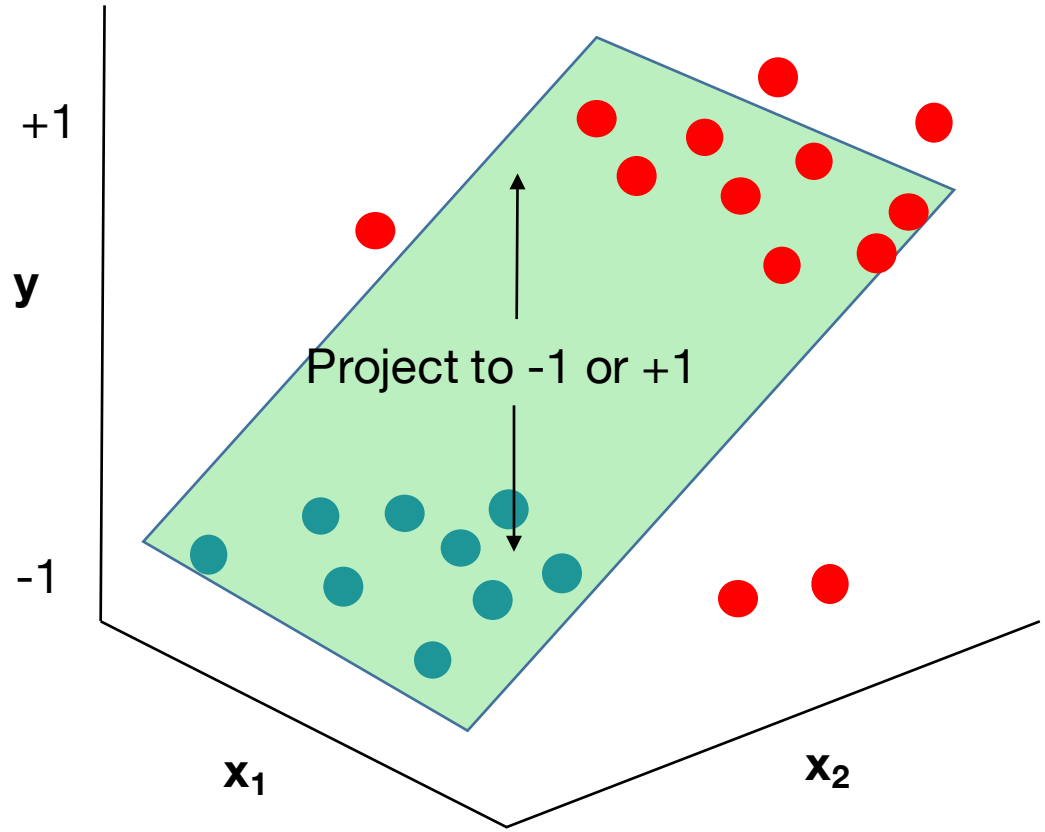
Without sgn(): regression for best fit

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

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

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

Why does linear regression with sgn() achieve classification?



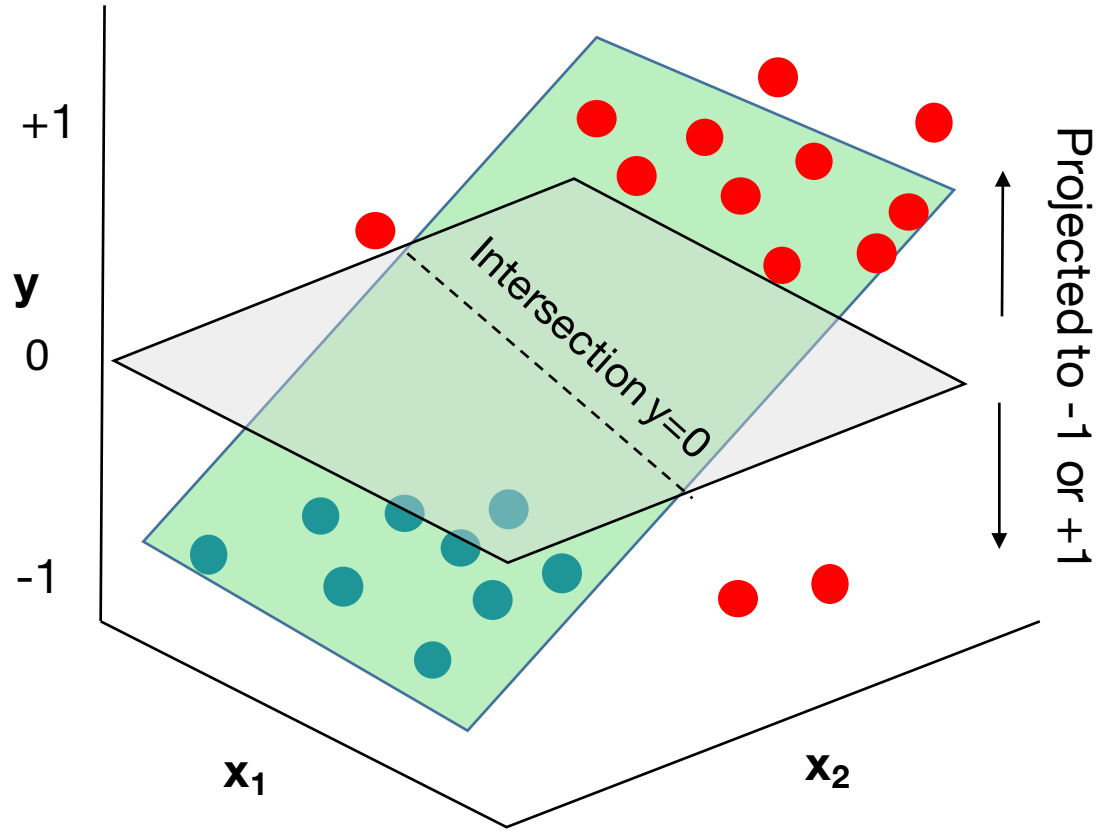
Without sgn(): regression for best fit

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

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

- 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?



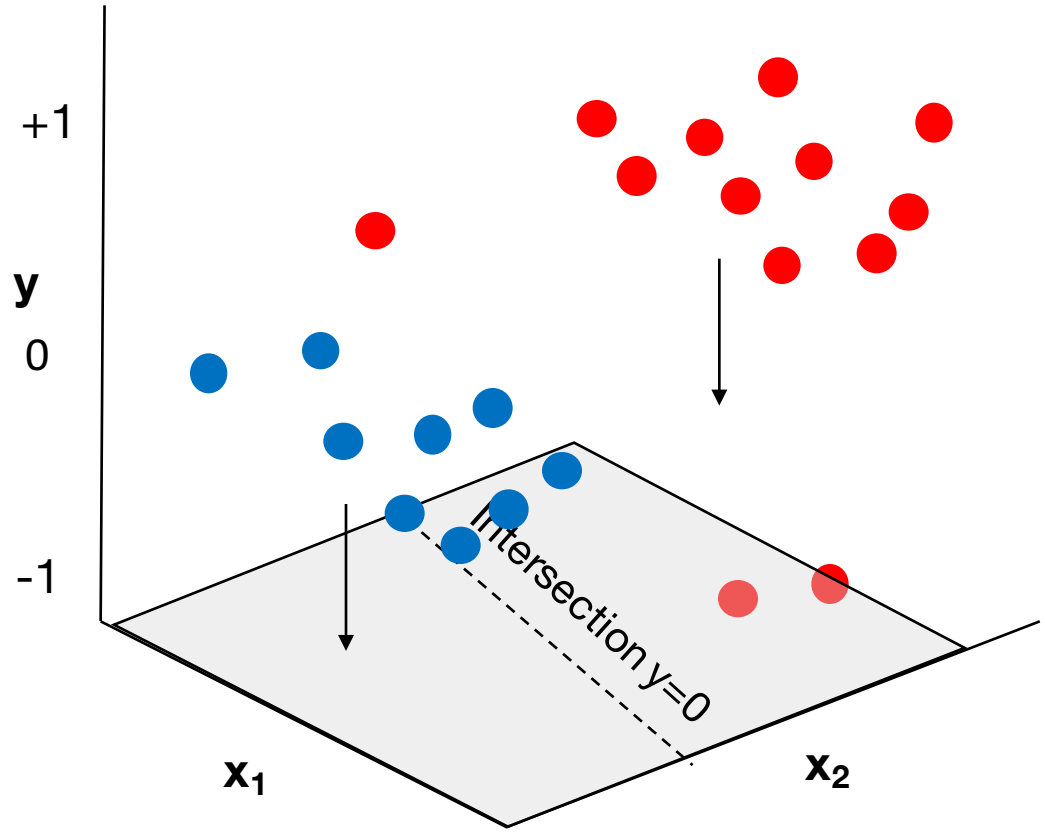
With sgn() operation:

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

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

- 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:

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

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

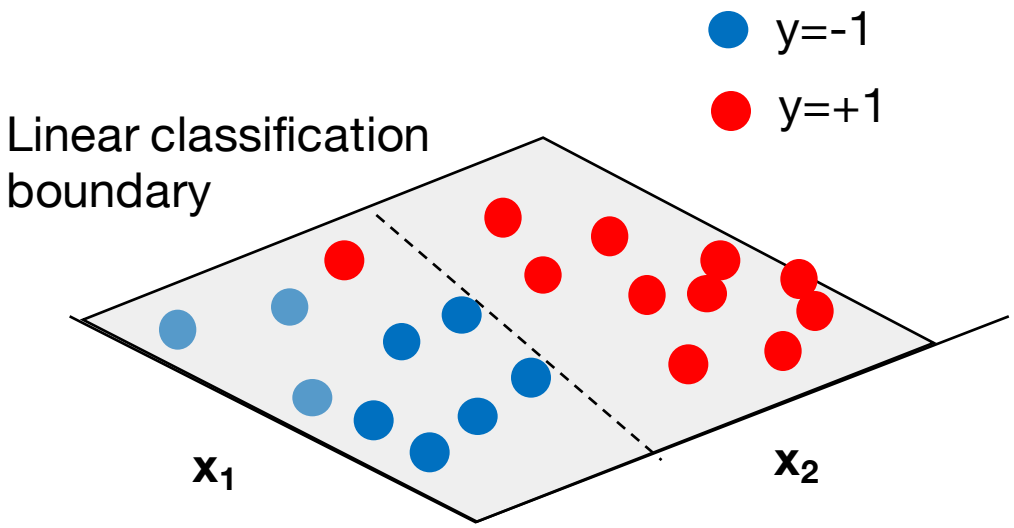
- y axis isn't really needed now & can view this decision boundary in 2D

Why does linear regression with sgn() achieve classification?

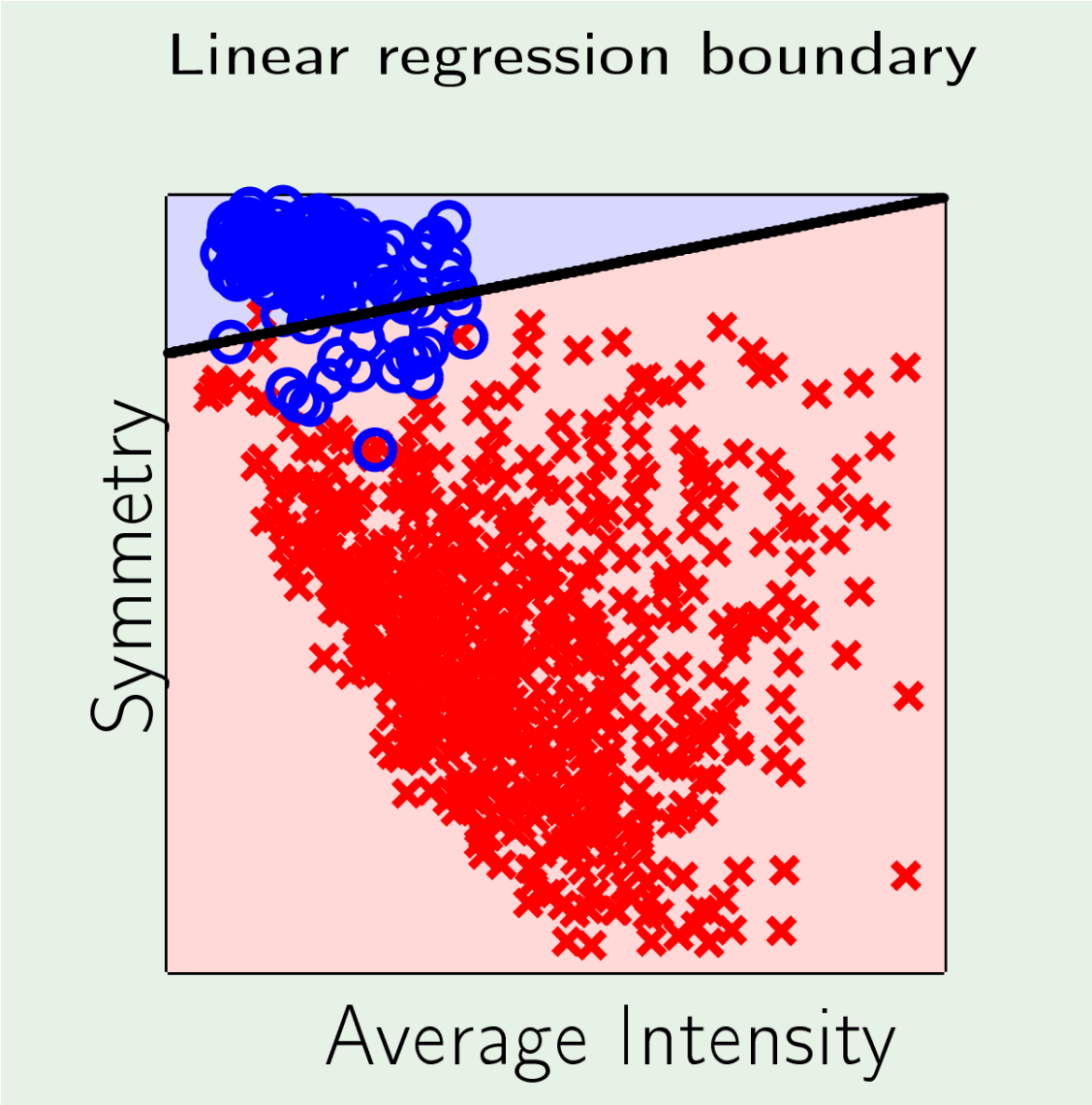
With sgn() operation:

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

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

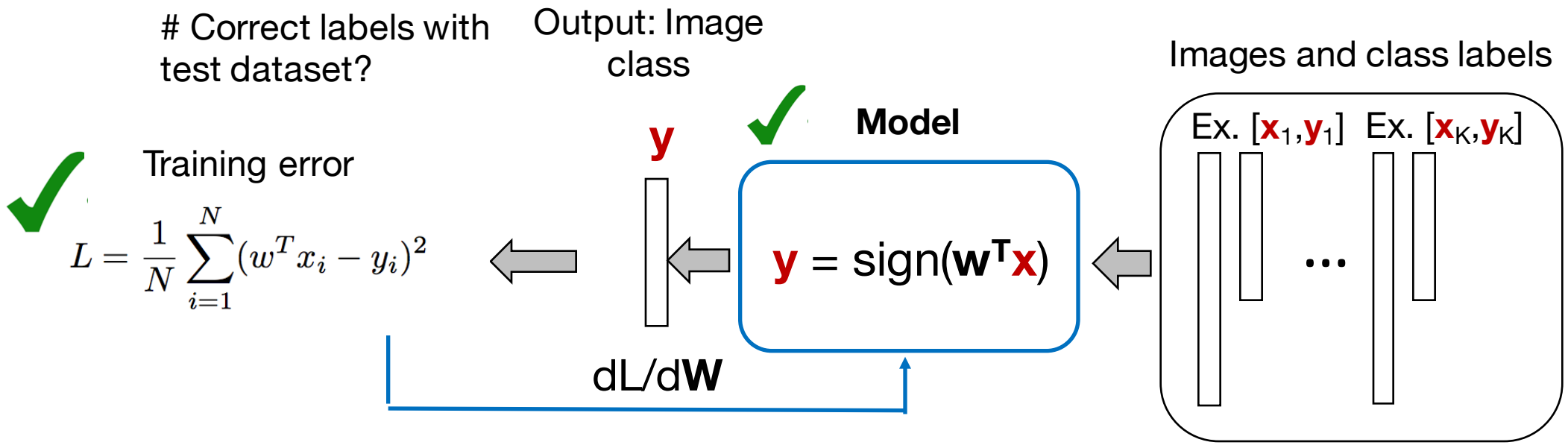


Sign operation takes linear regression and makes it a classification operation!



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

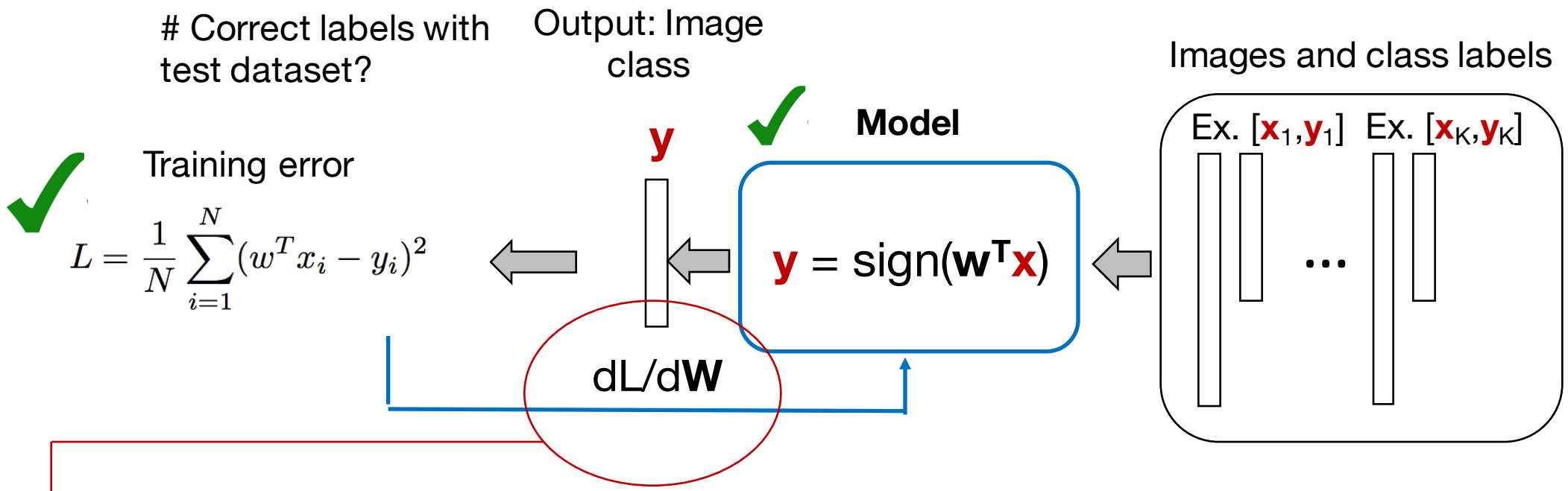
Example: machine learning for image classification



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)

1. Turning linear regression for unknown weights W in to a pseudo-inverse:

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

We are multiplying many x_i's with the same w and are adding them up - let's make a matrix!

$$X = \begin{vmatrix} x_1 \\ x_2 \\ \\ x_N \end{vmatrix}$$

Each training image is 1 row of X

$$y = \begin{vmatrix} y_1 \\ \\ \\ y_N \end{vmatrix}$$

Each training label is 1 entry of y

$$L = \frac{1}{N}\|Xw - y\|^2$$

This is the same form as the pseudo-inverse we were working with before, but now we want to solve for *w*

Write this out as a matrix equation:

$$L = \frac{1}{N} \|Xw - y\|^2$$

Note: Training data goes into “dictionary” matrix

$$L = \frac{1}{N} \left(w^T X^T X w - 2 w^T X^T y + y y^T \right)$$

Write this out as a matrix equation:

$$L = \frac{1}{N} \|Xw - y\|^2$$

Note: Training data goes into “dictionary” matrix

$$L = \frac{1}{N} \left(w^T X^T X w - 2 w^T X^T y + y y^T \right)$$

Take derivative wrt w and set to 0:

$$\nabla L(w) = \frac{2}{N} X^T (Xw - y) = 0$$

Solution is pseudo-inverse:

$$w_o = (X^T X)^{-1} X^T y$$

Write this out as a matrix equation:

$$L = \frac{1}{N} \|Xw - y\|^2$$

Note: Training data goes into “dictionary” matrix

$$L = \frac{1}{N} \left(w^T X^T X w - 2w^T X^T y + yy^T\right)$$

Take derivative wrt w and set to 0:

$$\nabla L(w) = \frac{2}{N} X^T (Xw - y) = 0$$

Solution is pseudo-inverse:

$$w_o = (X^T X)^{-1} X^T y$$

Steps for Pseudo-inverse:

1. Construct matrix X and vector y from data

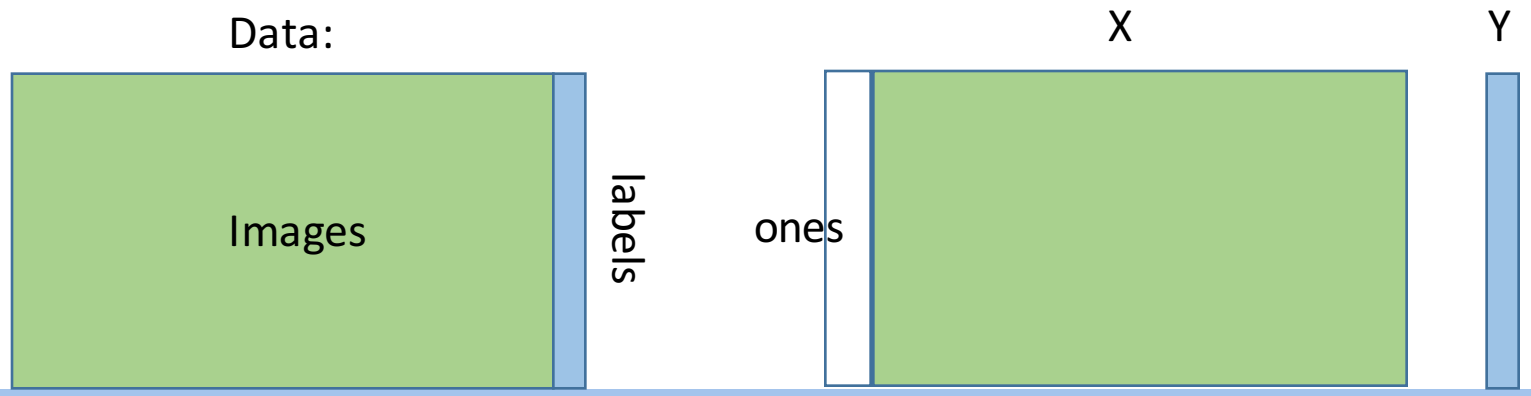
   Each training image is 1 row of X

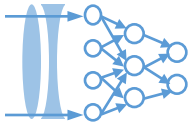
   Each training label is 1 entry of y

2. Compute solution for w_0 via above equation

Example pseudo-code

```
data = np.loadtxt('train_data.txt', dtype=int)
X = numpy.zeros((data.shape[0],data.shape[1]-1))
X[:,0]=1
Y = numpy.zeros((data.shape[0],1))
for row in m:
    X[row,1:X.shape[1]-1] = data[row,0:data.shape[1]:1]
    Y[row] = data[row,data.shape[1]-1]
X_dagger = np.linalg.pinv(X)
w = np.matmul(X_dagger,Y)
```





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)

Gradient descent: The iterative recipe

deep imaging

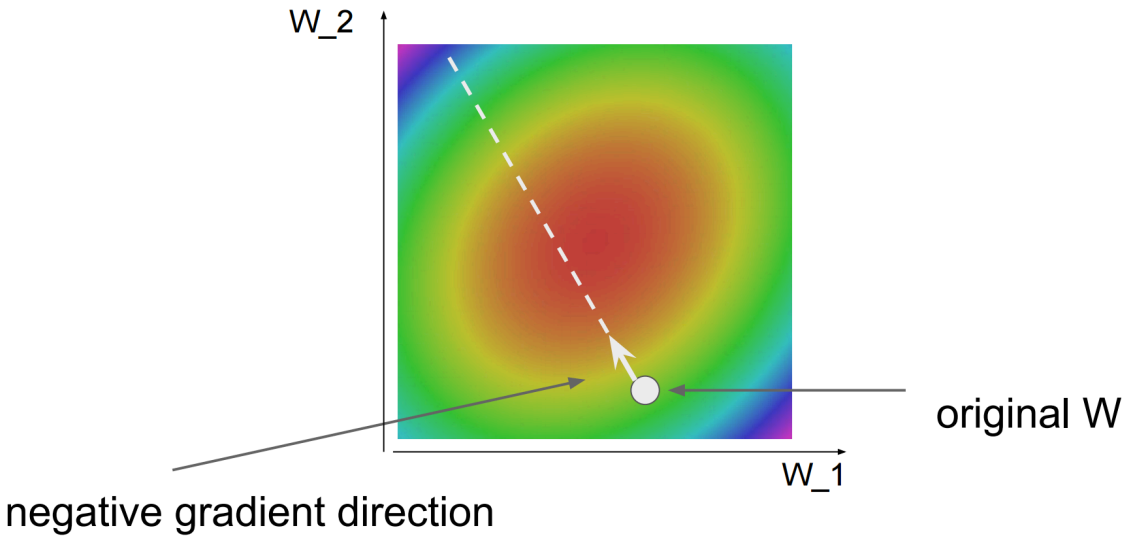
Initialize: Start with a guess of **W**

Until the gradient does not change very much:

dL/d**W** = evaluate_gradient(**W**, **x** ,y ,L)

W = **W** – step_size * dL/d**W**

evaluate_gradient can be achieved numerically or algebraically



Gradient descent: Numerical evaluation example

With a matrix, compute this for each entry:

$$\frac{dL(W_i)}{dW_i} = \lim_{h \to 0} \frac{L(W_i + h) - L(W_i)}{h}$$

Gradient descent: Numerical evaluation example

With a matrix, compute this for each entry:

$$\frac{dL(W_i)}{dW_i} = \lim_{h \to 0} \frac{L(W_i + h) - L(W_i)}{h}$$

Example:

W = [1,2;3,4]
L(W, x, y) = 12.79

Gradient descent: Numerical evaluation example

With a matrix, compute this for each entry:

$$\frac{dL(W_i)}{dW_i} = \lim_{h \to 0} \frac{L(W_i + h) - L(W_i)}{h}$$

Example:

W = [1,2;3,4]
L(W, x, y) = 12.79

W_1+h = [1.001,2;3,4]
L(W_1+h, x, y) = 12.8

Gradient descent: Numerical evaluation example

With a matrix, compute this for each entry:

$$\frac{dL(W_i)}{dW_i} = \lim_{h \to 0} \frac{L(W_i + h) - L(W_i)}{h}$$

Example:

W = [1,2;3,4]
L(W, x, y) = 12.79

W_1+h = [1.001,2;3,4]
L(W_1+h, x, y) = 12.8

d$L(W_1)$/dW_1 = 12.8-12.79/.001

d$L(W_1)$/dW_1 = 10

- Repeat for all entries of **W**, dL/d**W** will have NxM entries for NxM matrix

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 go over how gradient descent works, and then examine the cost function to see why it really matters and how we can extend things beyond the simple case of linear regression