Machine Learning in Imaging

BME 590L
Roarke Horstmeyer

Lecture 5: Ingredients for Machine Learning

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

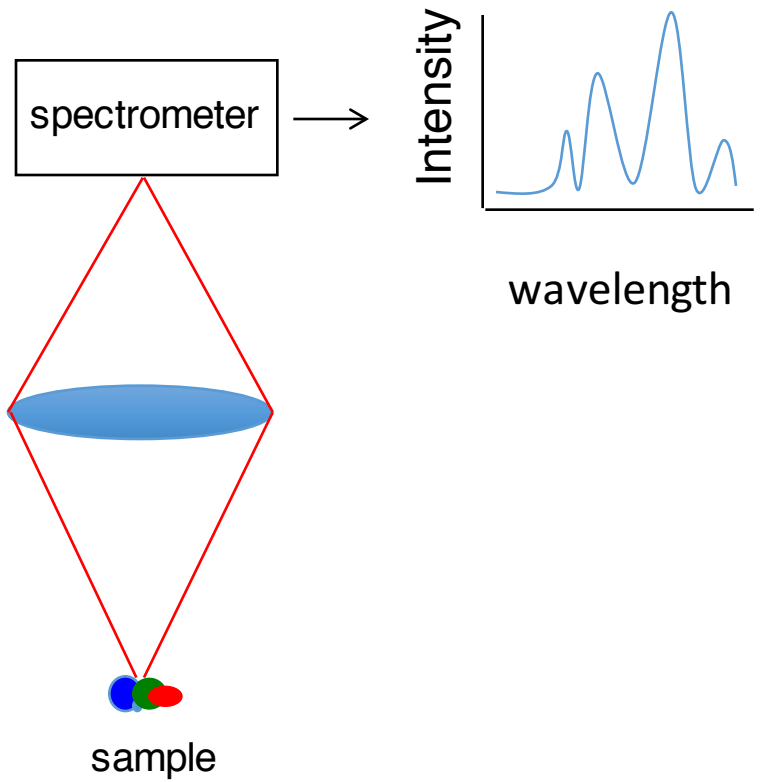
(For whatever reason, whenever I get confused about optimization, I think about this example)

The setup:

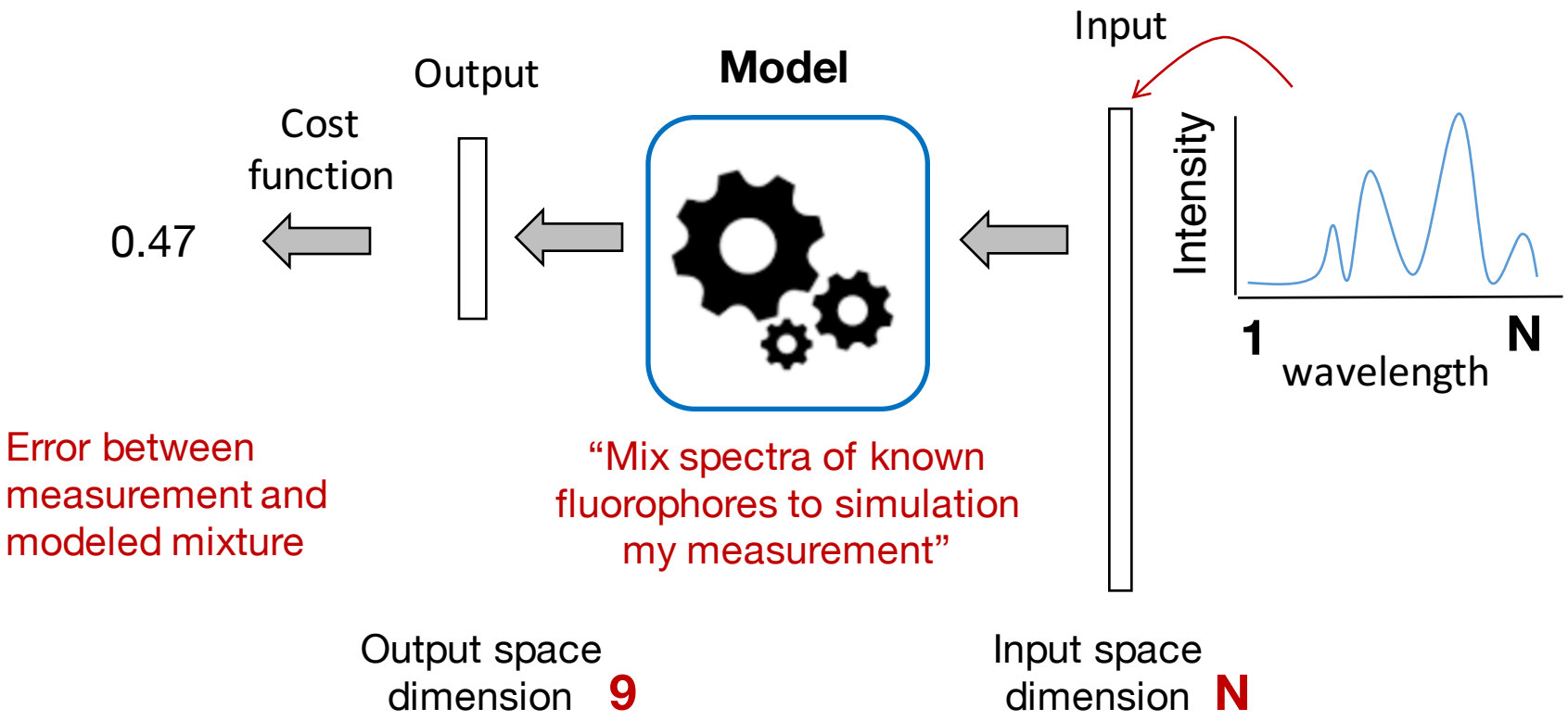
- measure the color (spectral) response of a sample (e.g., how much red, green and blue there is, or several hundred measurements of its different colors).

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

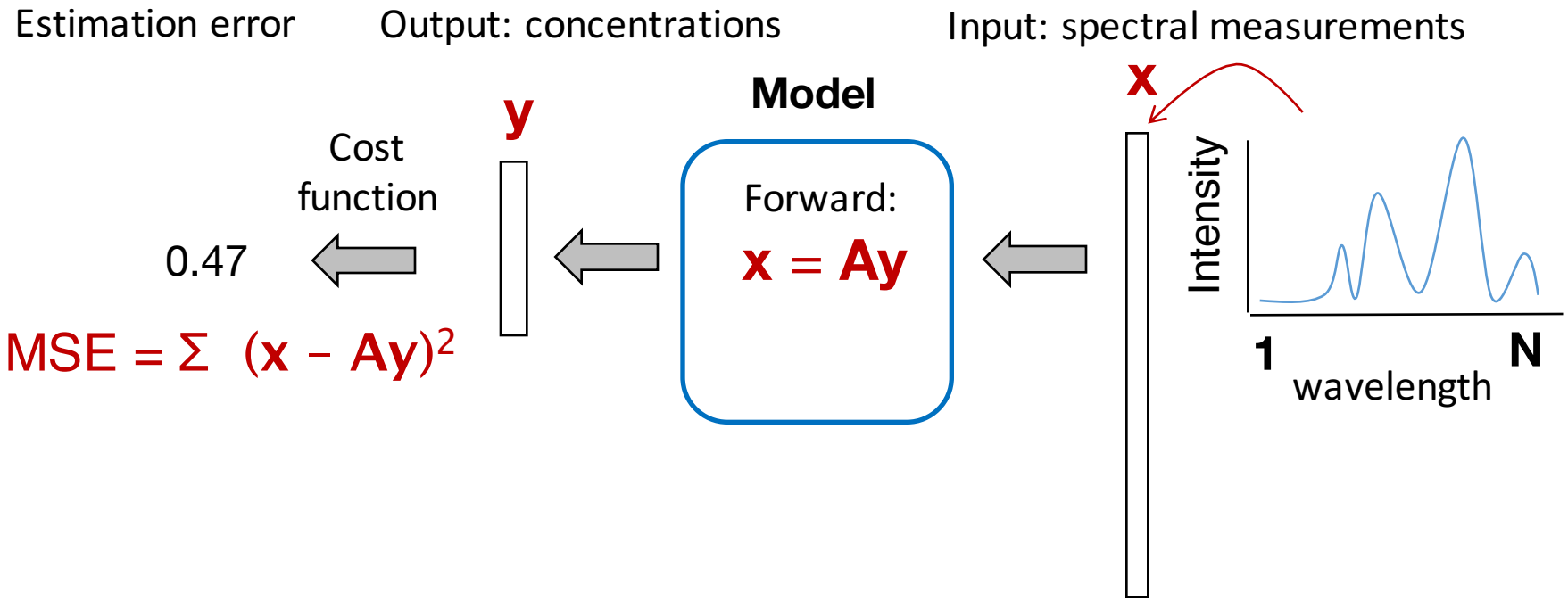
- What % of each fluorophores is in your sample?



Optimization pipeline for spectral unmixing

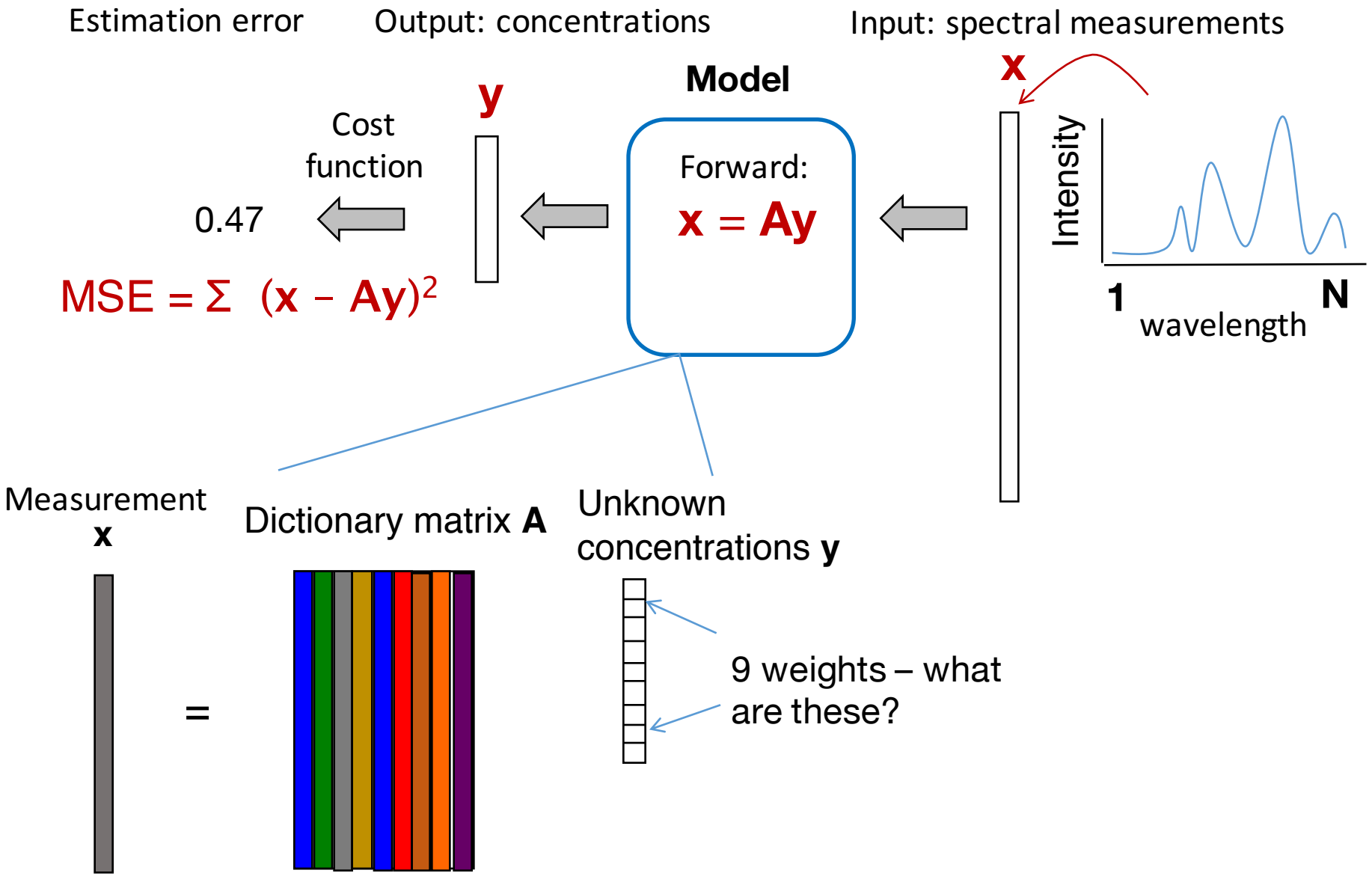


Optimization pipeline for spectral unmixing

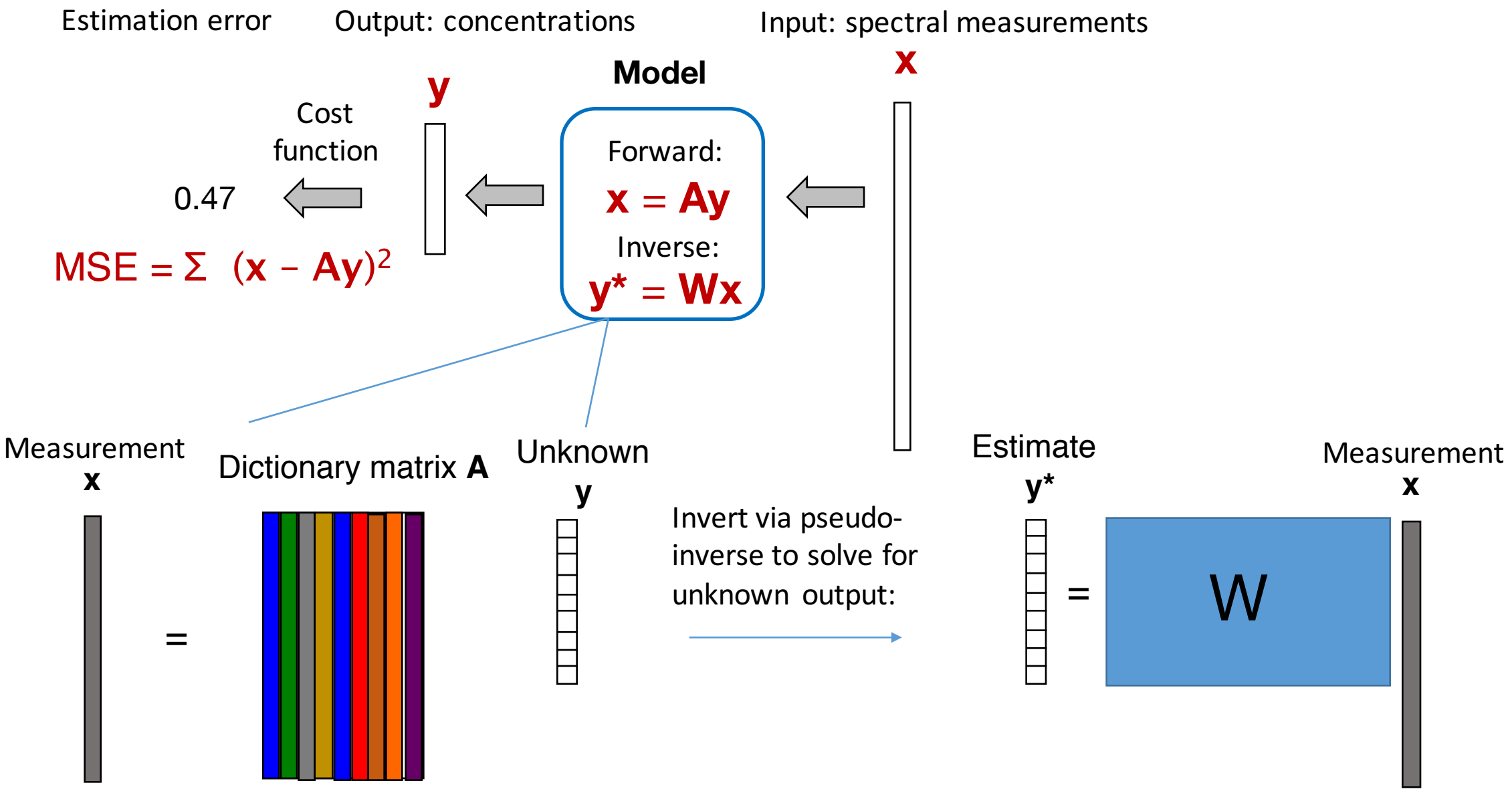


*Note: notation changed from last time to be consistent with we'll use in the future

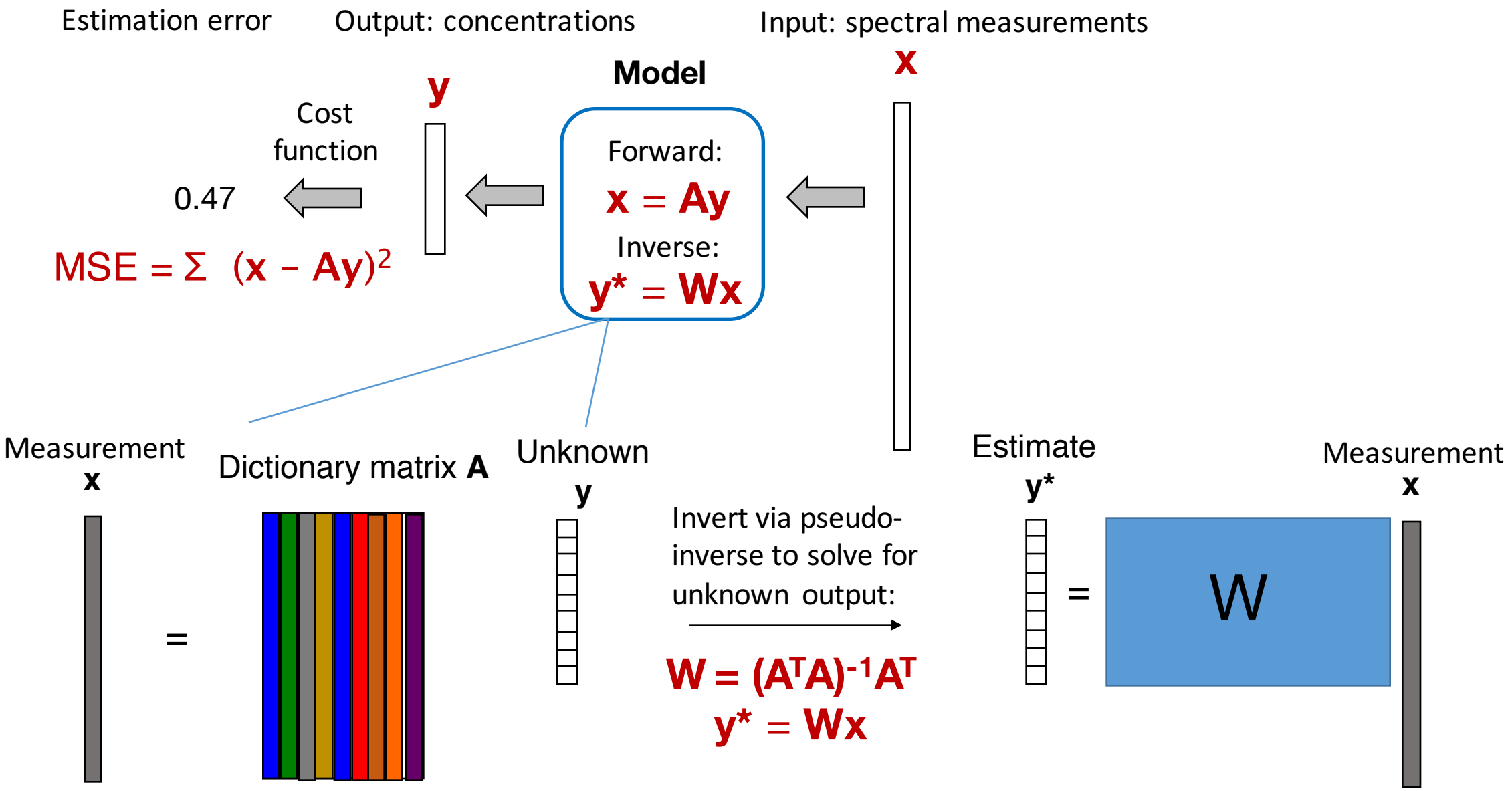
Optimization pipeline for spectral unmixing



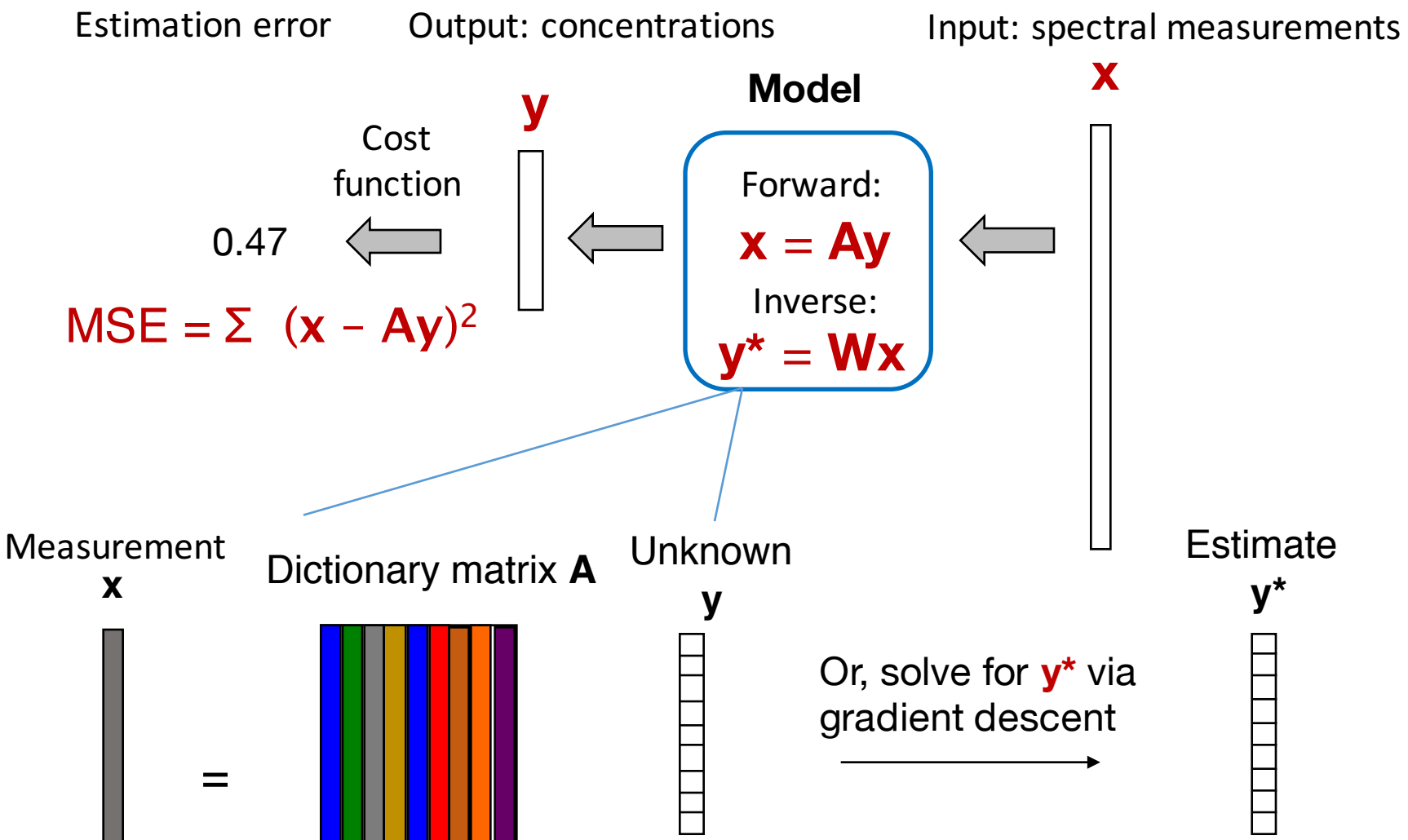
Optimization pipeline for spectral unmixing



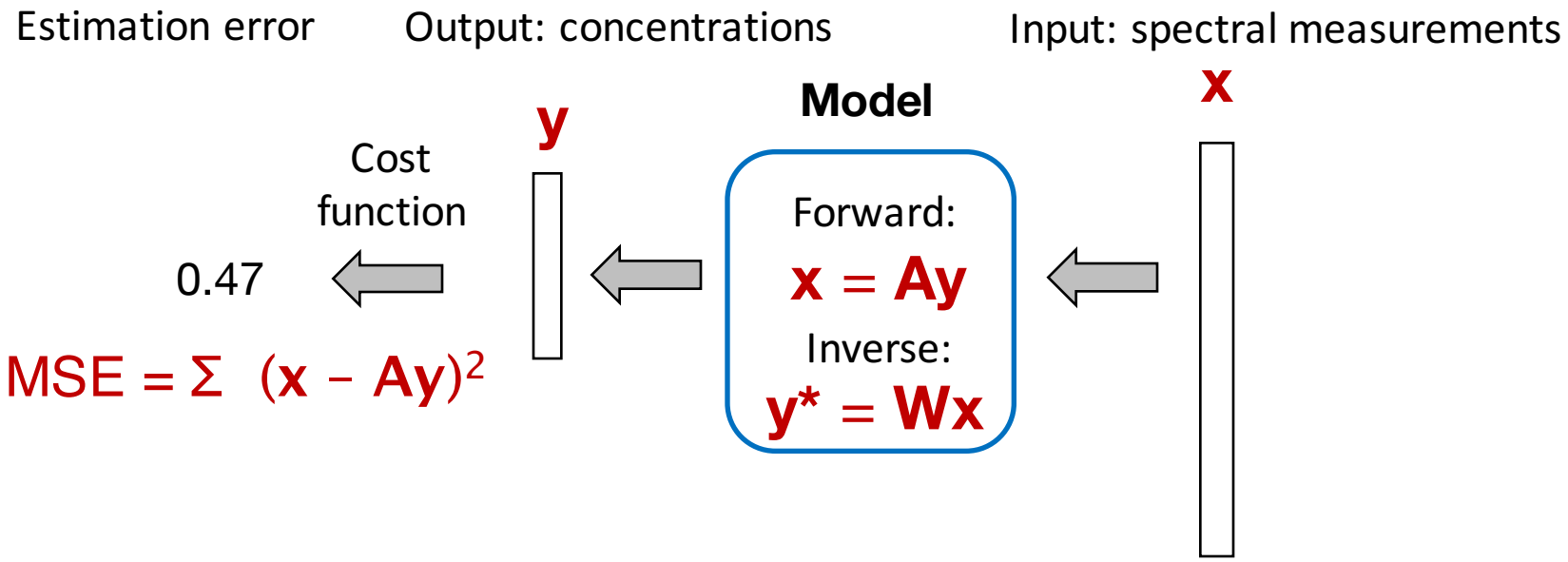
Optimization pipeline for spectral unmixing



Optimization pipeline for spectral unmixing

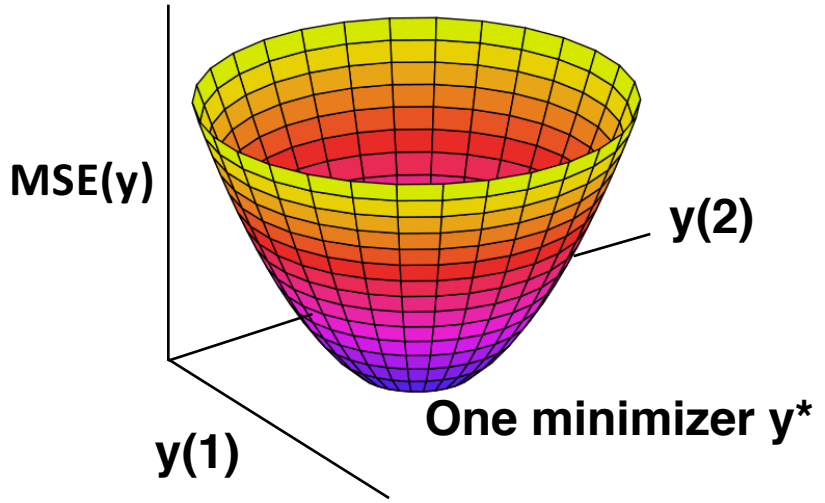


Optimization pipeline for spectral unmixing

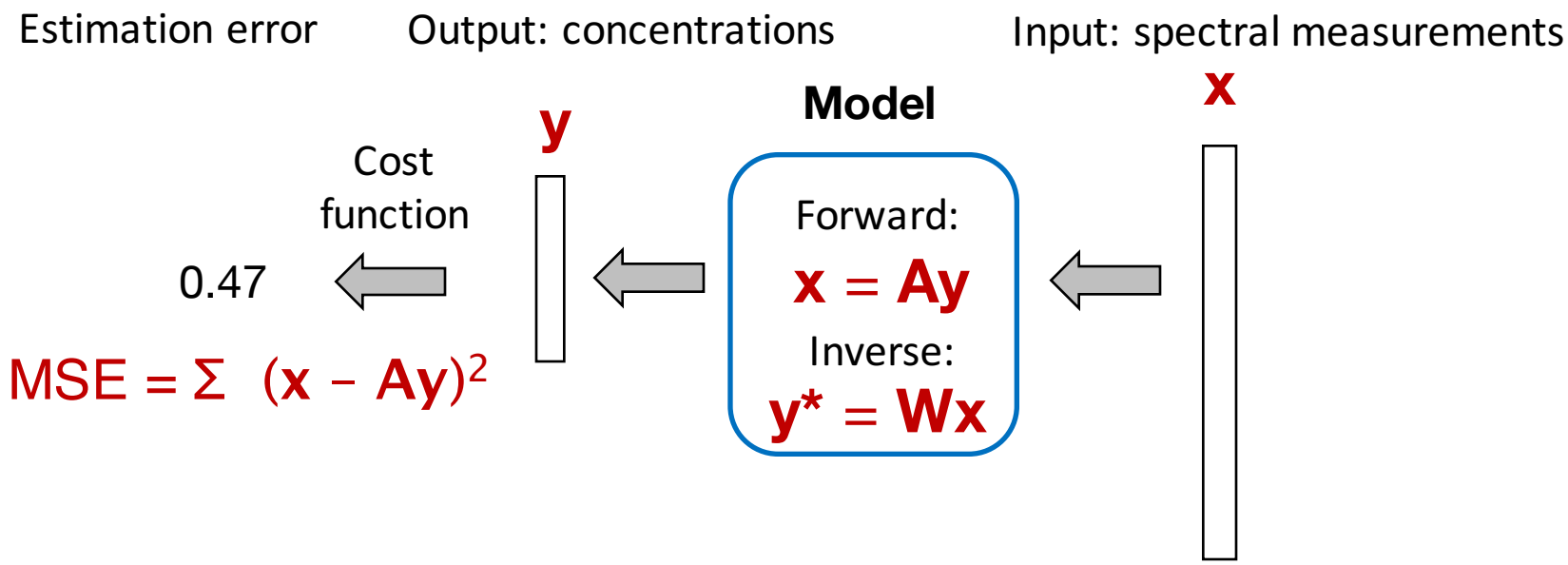


$$MSE(\mathbf{y}) = \sum_{\text{spectral measurements}} (\mathbf{x} - \mathbf{A}\mathbf{y})^2$$

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

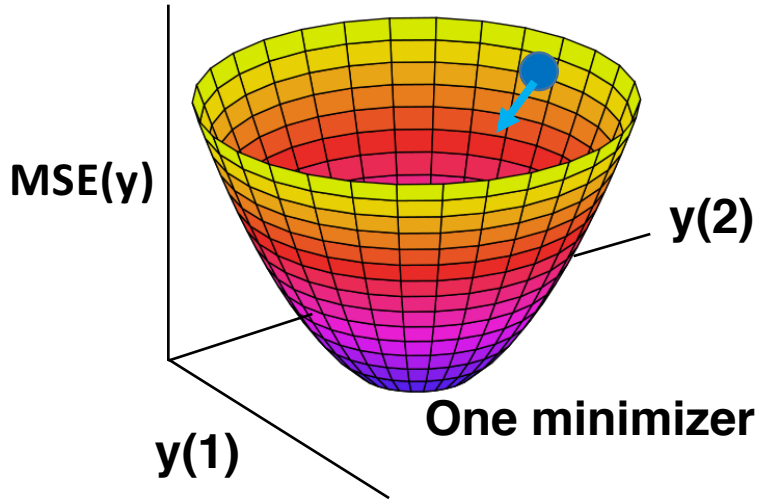


Optimization pipeline for spectral unmixing



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

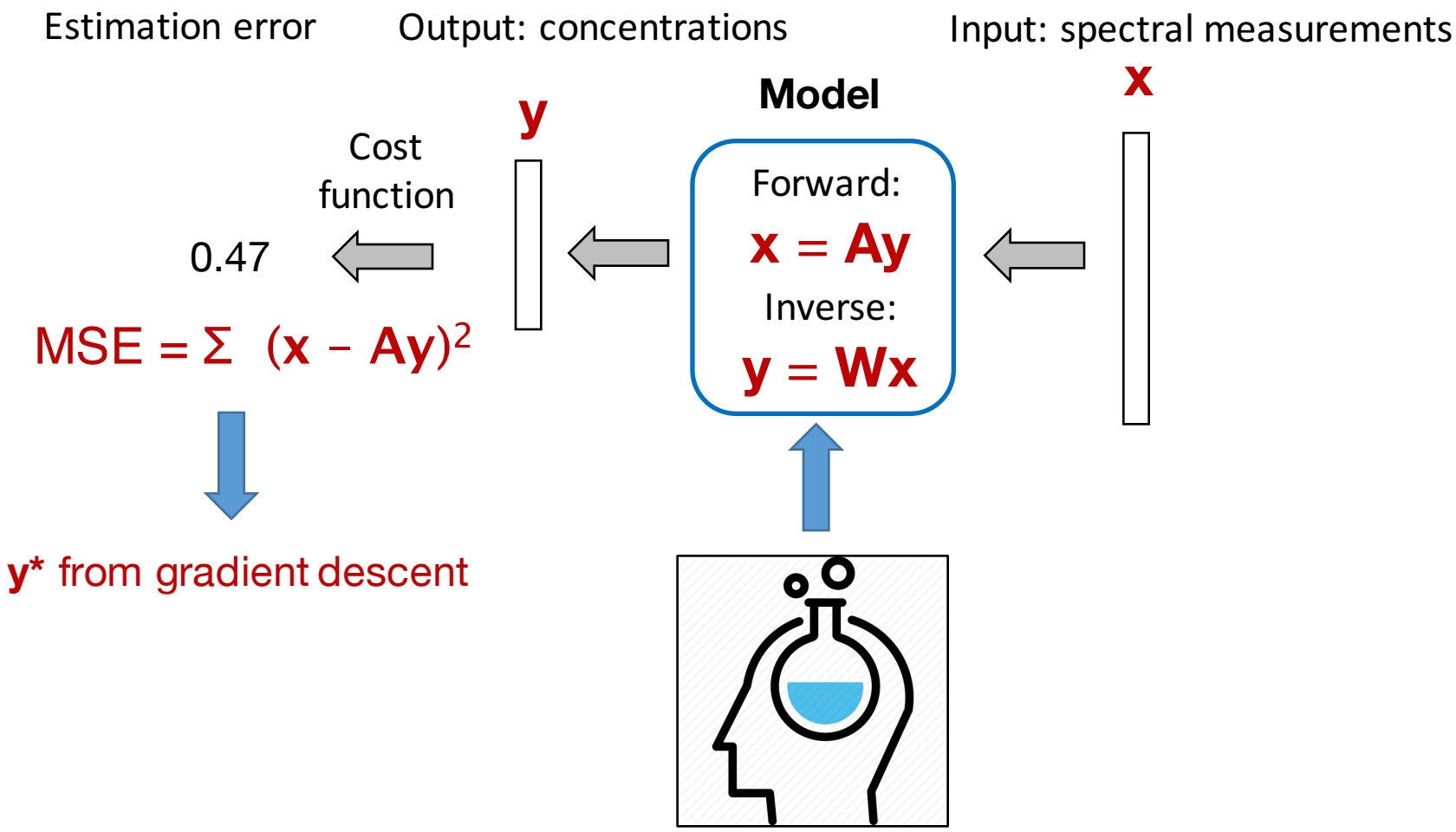
$$d/d\mathbf{y}\ MSE = \mathbf{A}^T(\mathbf{x} - \mathbf{A}\mathbf{y})$$



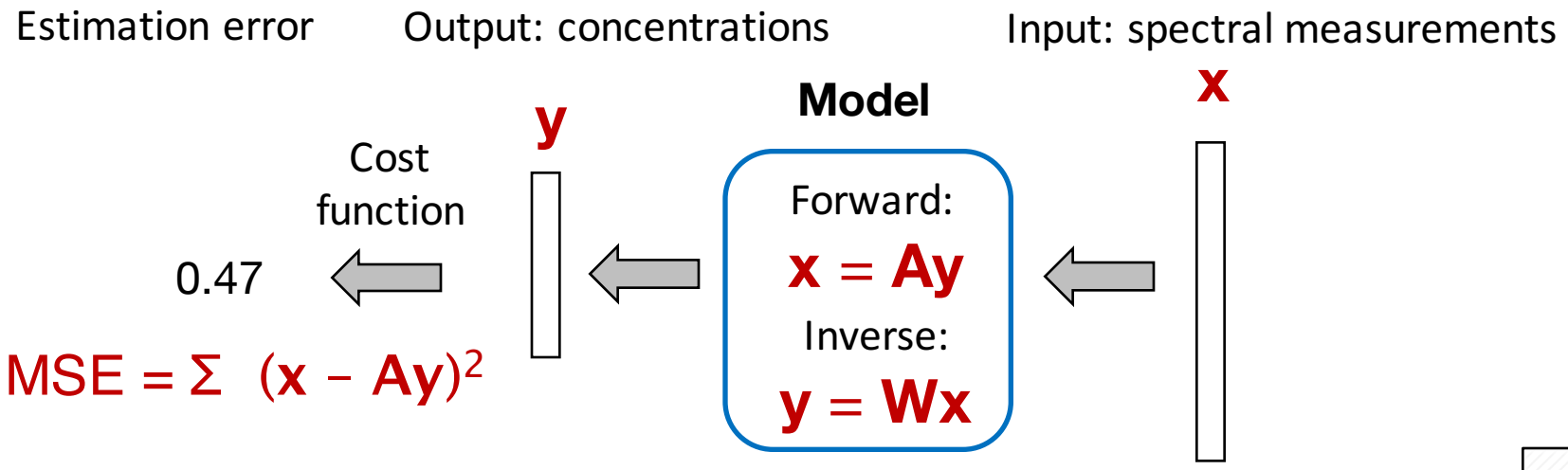
1. Guess output **y**
2. Evaluate slope
3. If its large, take a step:

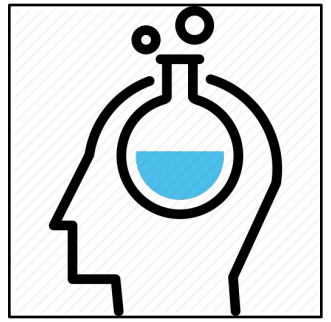
$$\mathbf{y} - \mathbf{y} + \varepsilon\mathbf{A}^T(\mathbf{x} - \mathbf{A}\mathbf{y})$$

Optimization pipeline for spectral unmixing



Optimization pipeline for spectral unmixing

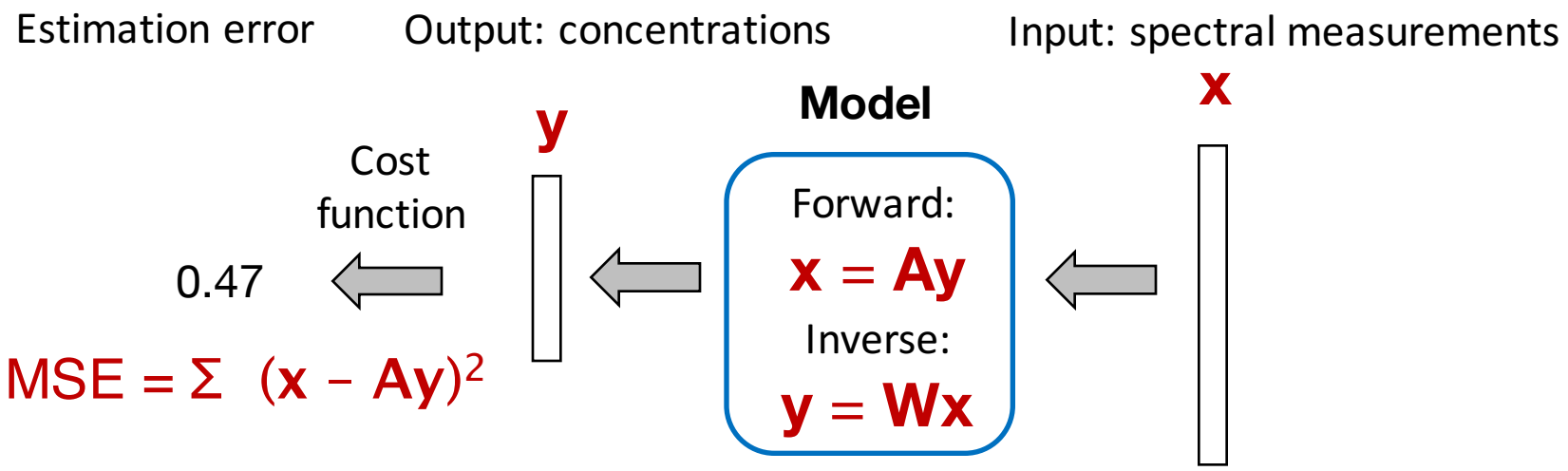




A and **W** from first principles

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

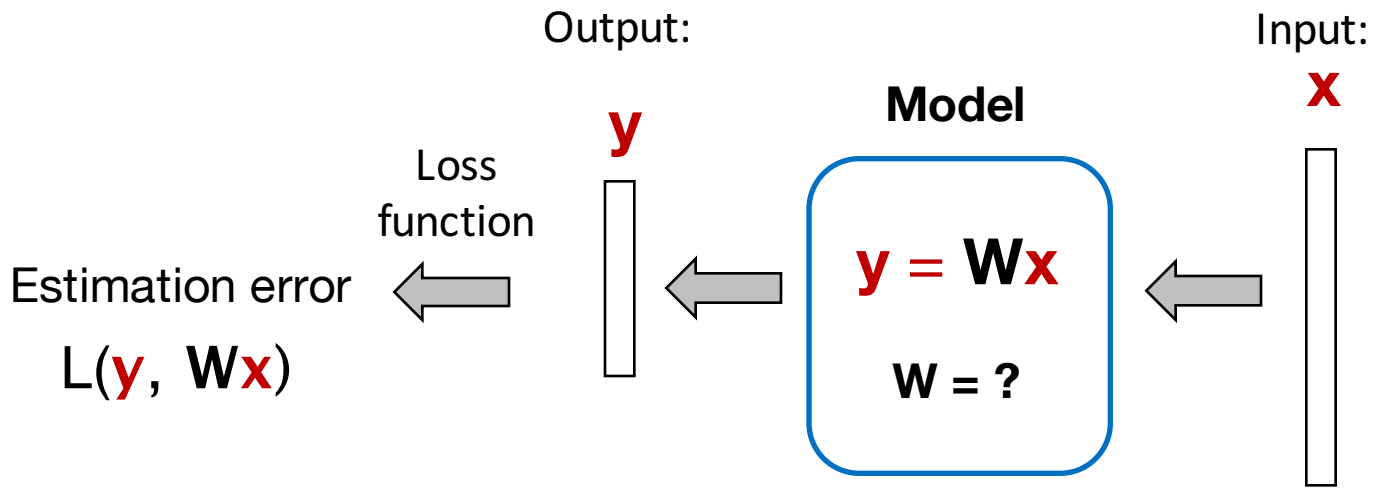
Optimization pipeline for spectral unmixing



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***

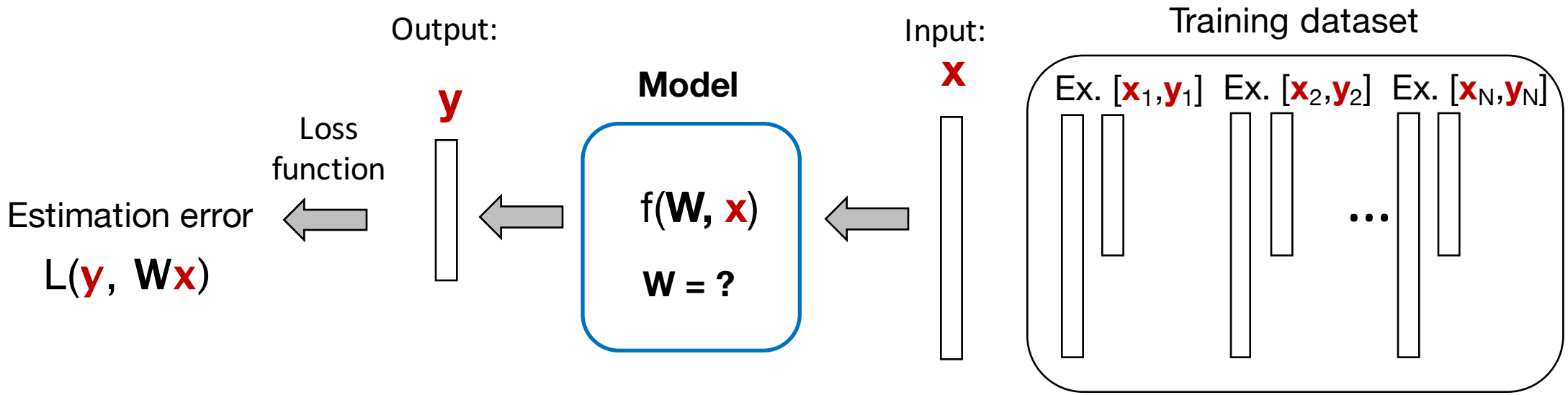
Pipeline for machine learning



Changes for machine learning framework:

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

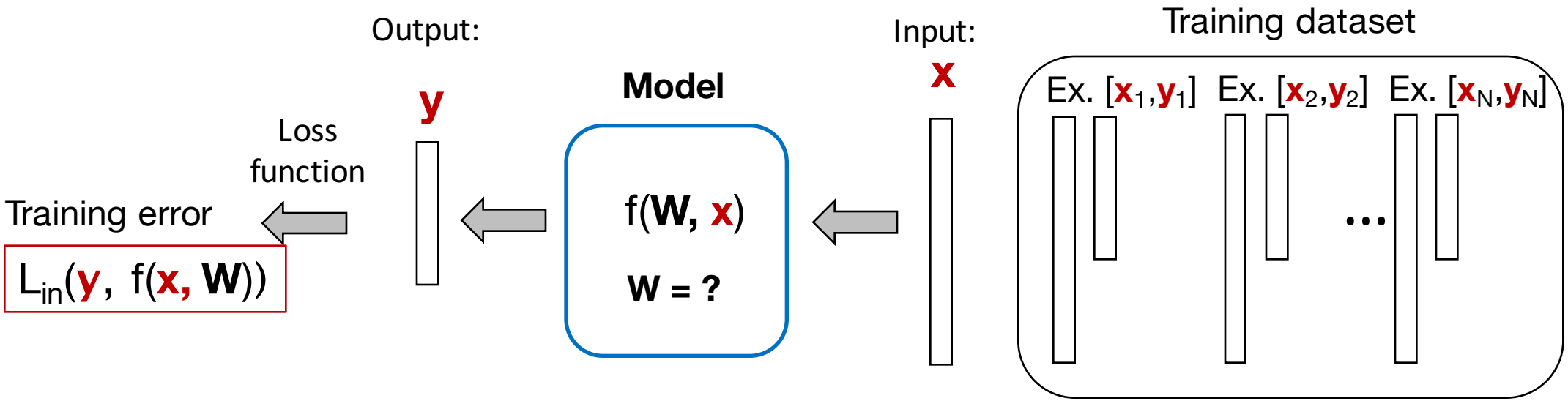
Pipeline for machine learning



Changes for machine learning framework:

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

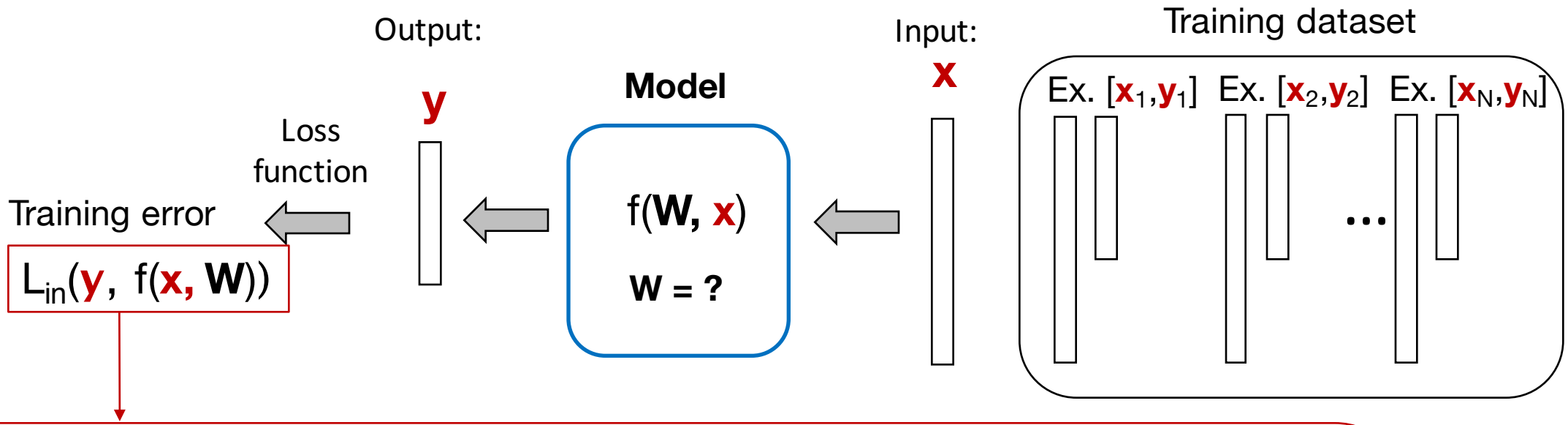
Pipeline for machine learning



Changes for machine learning framework:

1. Now just establish the mapping from inputs to outputs (here, matrix **W**)
2. Must first determine mapping f(**x**, **W**) using large set of "training" data
3. Use a *loss function* L that depends upon the training inputs (x,y) *and* the model (**W**)

Pipeline for machine learning

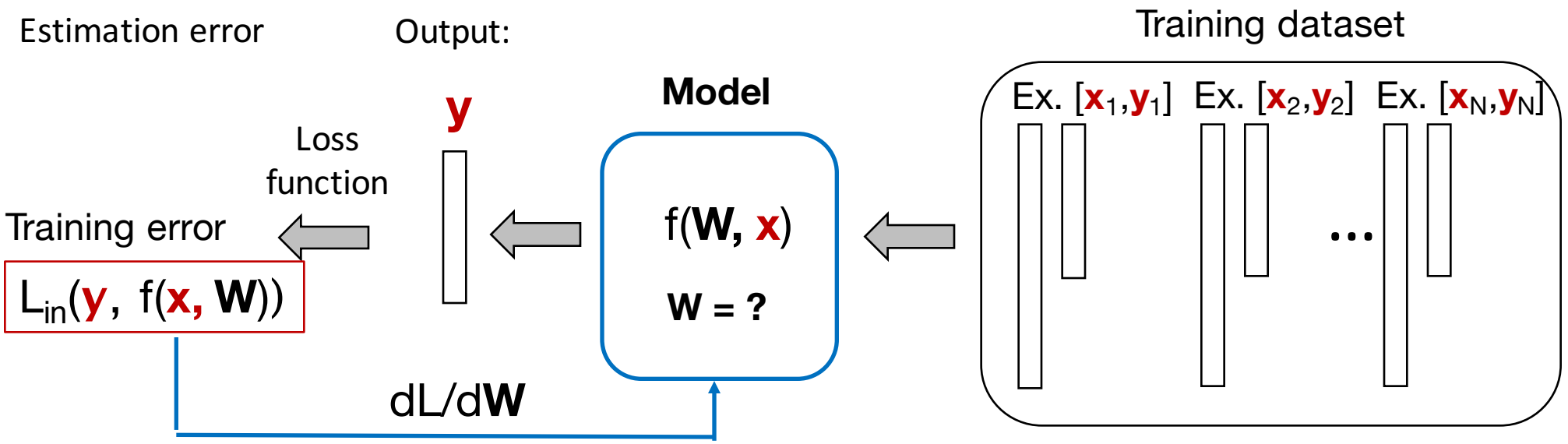


Training Error ("in class error"):

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

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

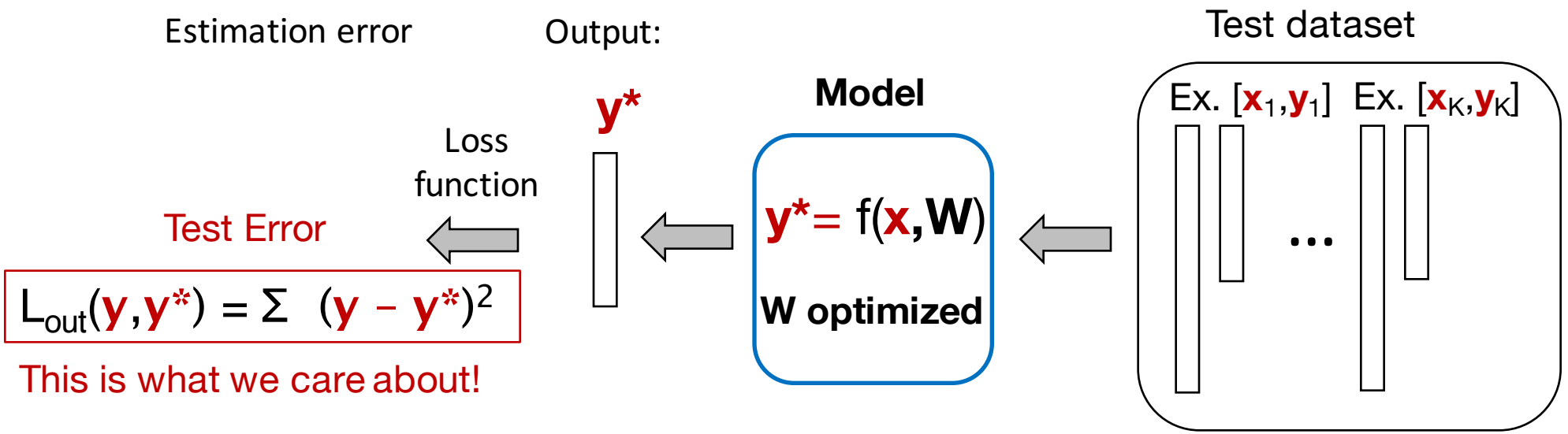
Pipeline for machine learning



Changes for machine learning framework:

1. Now just 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. 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 to guide gradient descent on L

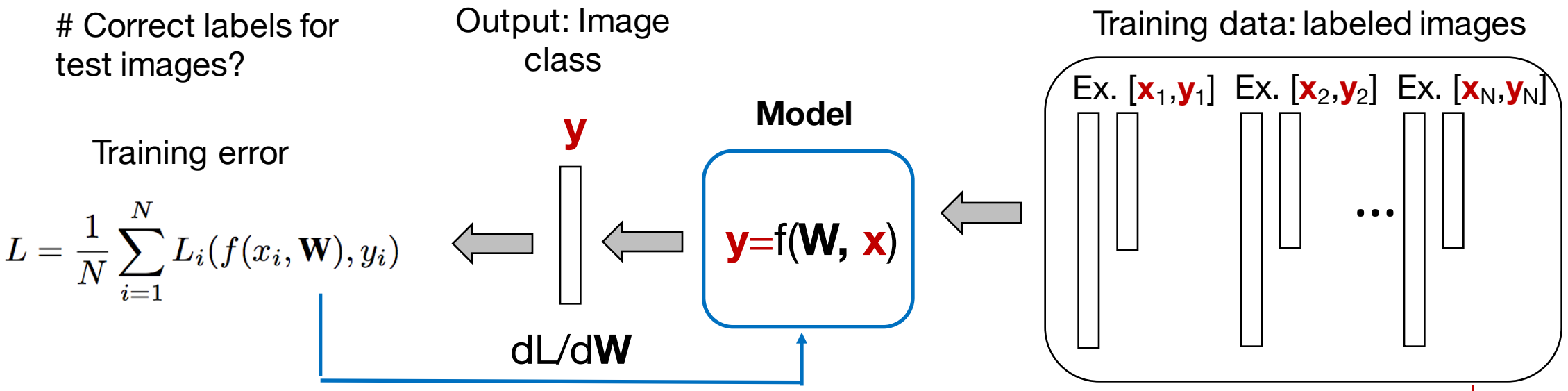
Pipeline for machine learning



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 to guide gradient descent on L
5. Then evaluate model accuracy by sending *new* **x** through and comparing output **y*** to "test" data **y**

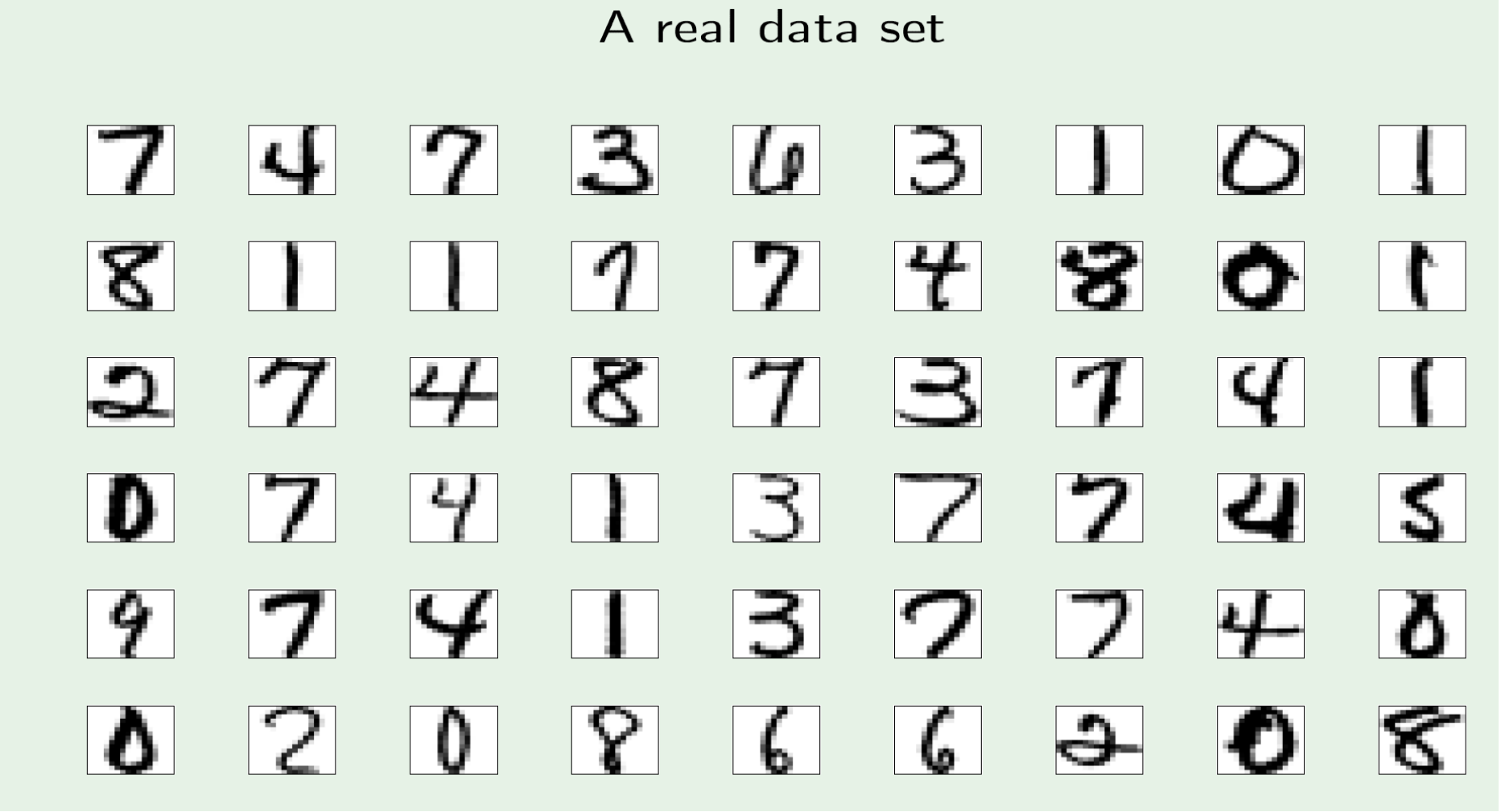
Example: machine learning for image classification



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

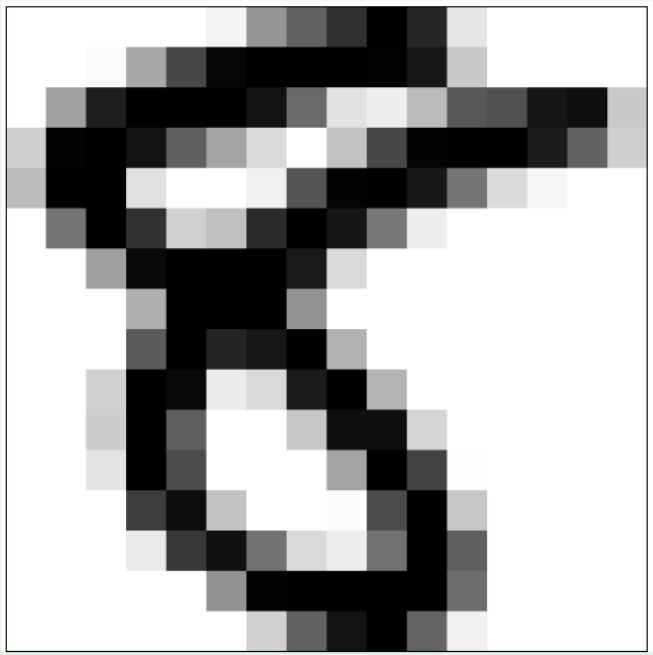


From Yaser Abu Mustafa, Machine Learning course at Caltech

Input representation

'raw' input $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_{256})$

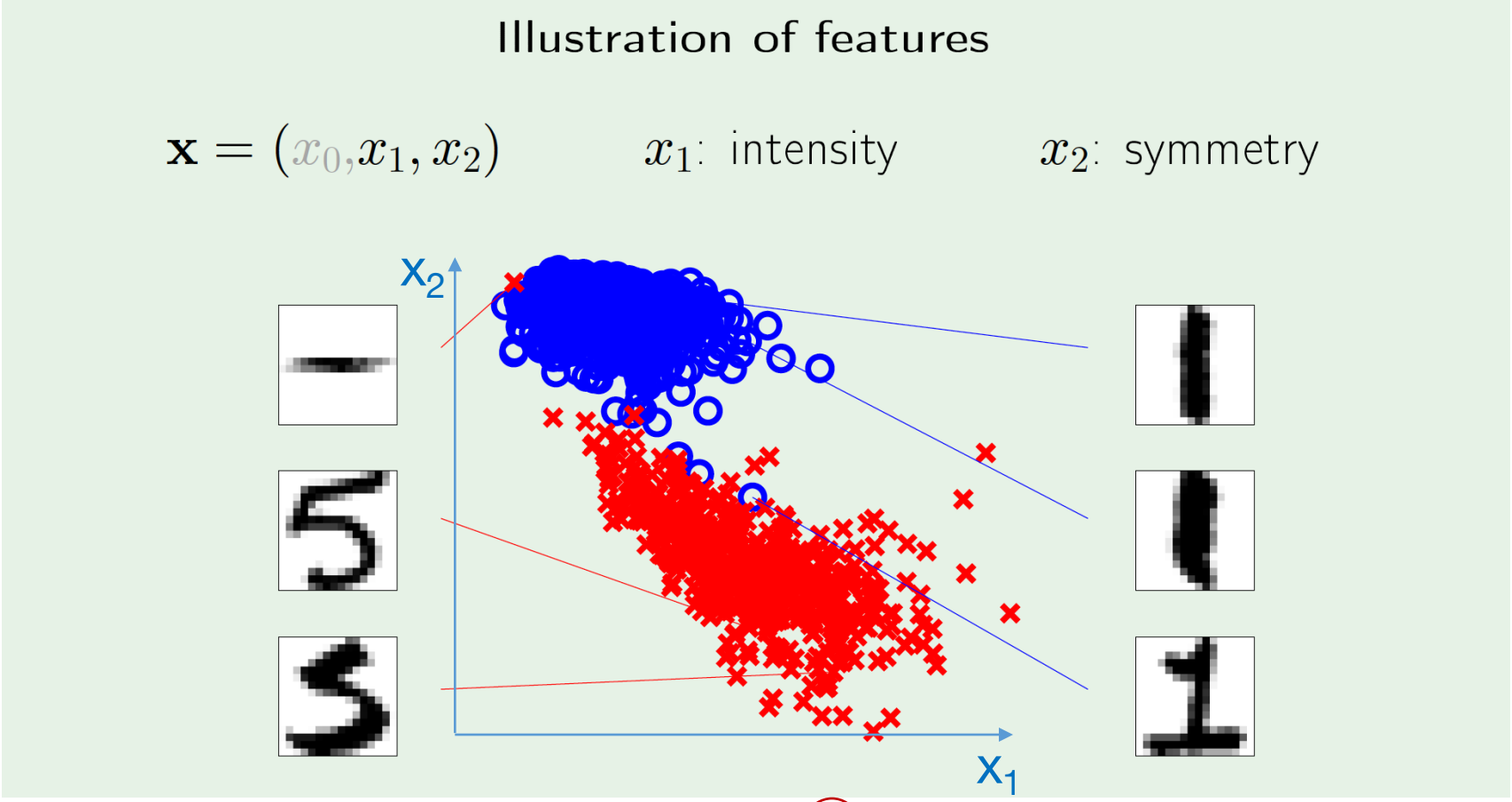
linear model: $(w_0, w_1, w_2, \cdots, w_{256})$



Features: Extract useful information, e.g.,

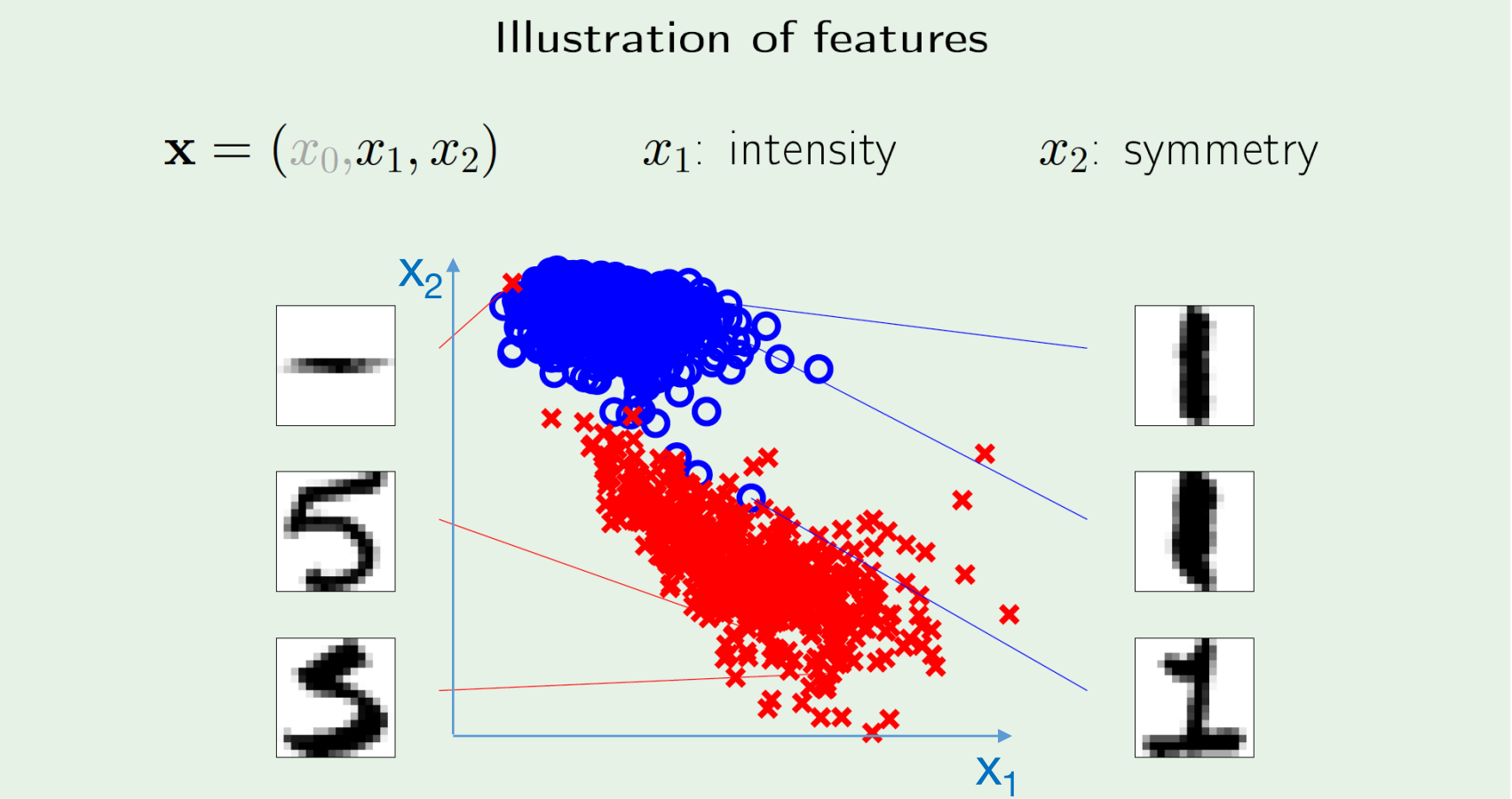
intensity and symmetry $\mathbf{x} = (x_0, x_1, x_2)$

linear model: (w_0, w_1, w_2)



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**



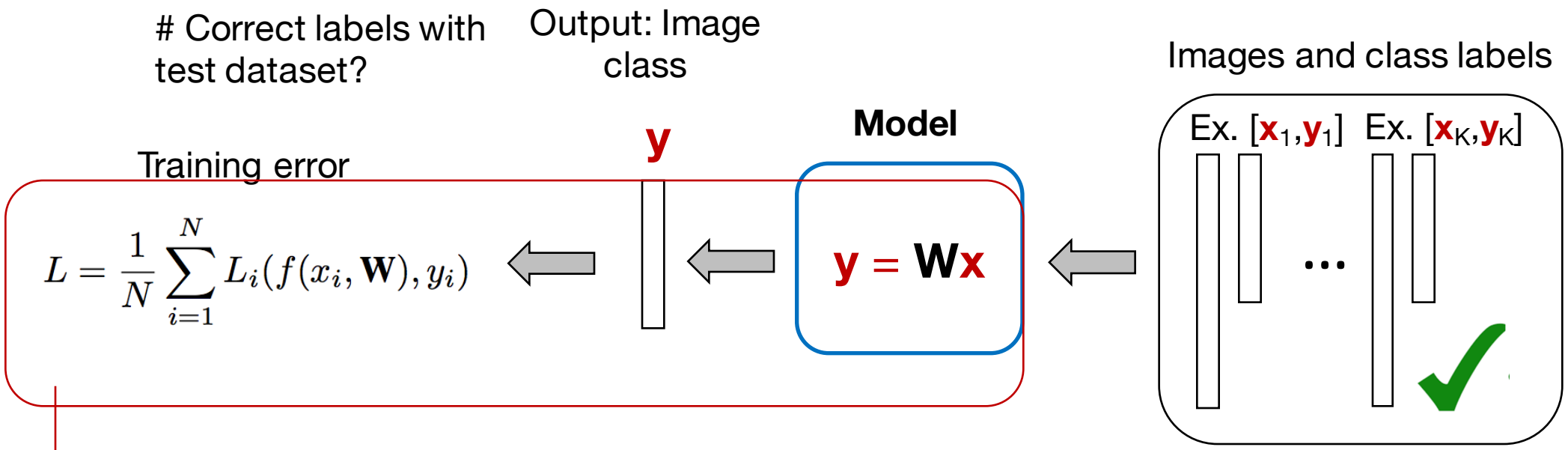
Dataset: 1000 examples of 1's and 5's mapped to $\mathbf{x}_j = (1, x_1, x_2)$, with associated label $y_j = 1$ or -1

$[\mathbf{X}_{train}, \mathbf{Y}_{train}] = [\mathbf{x}_j, \mathbf{y}_j]$ 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** $\{(x_i, y_i)\}_{i=1}^{N}$

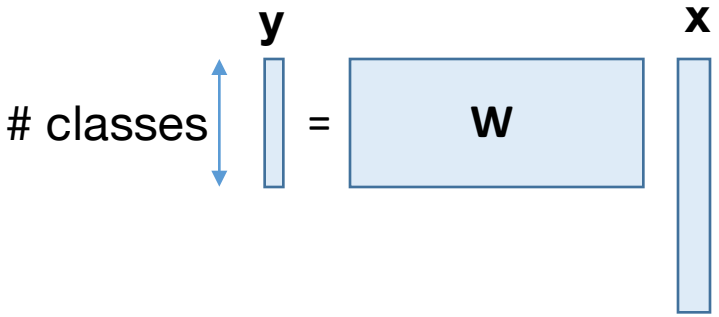
2. A model and loss function

Let's start with a simpler approach: linear regression

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

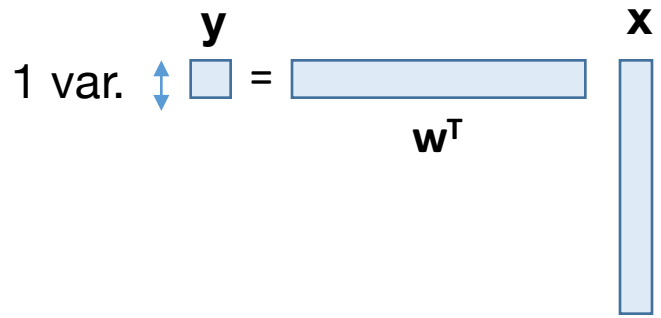
General linear model:

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



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

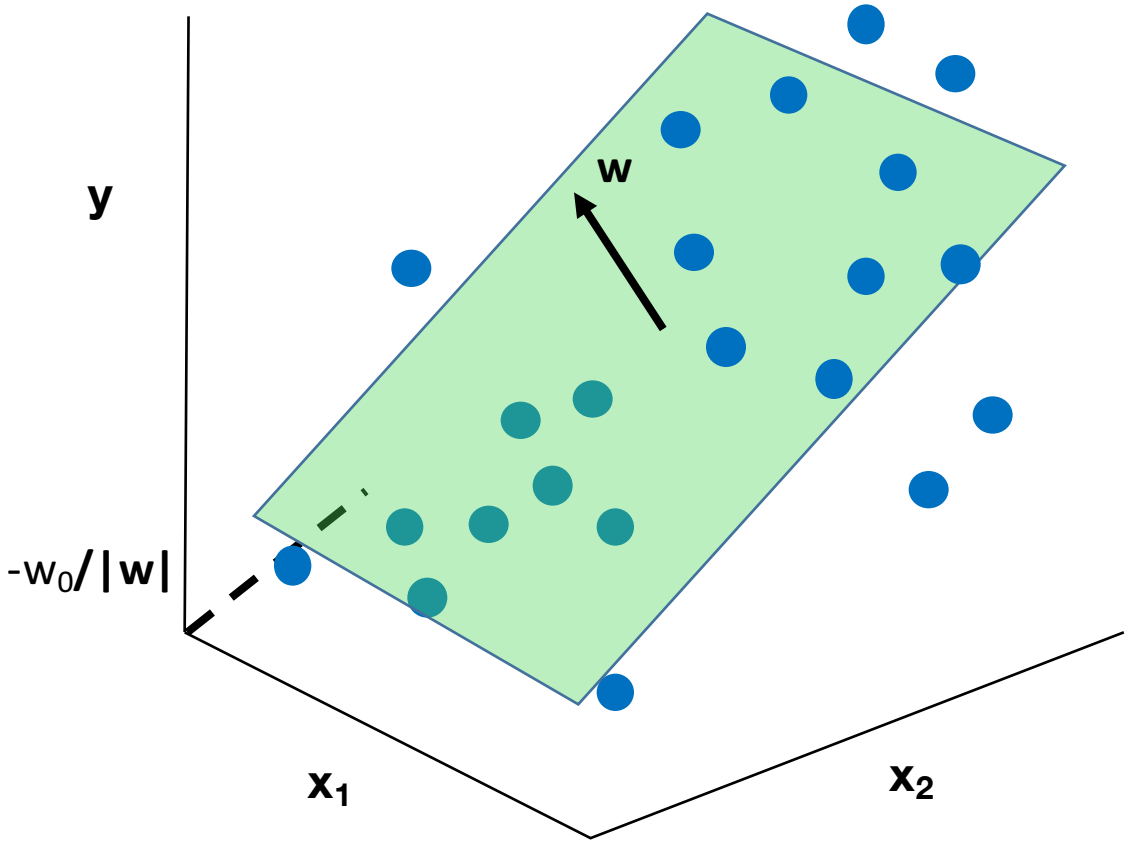
$$L = \frac{1}{N}\sum_{i=1}^{N} (w^T x_i - y_i)^2$$

Where labels determined by thresholding

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

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

Why does linear regression with sgn() achieve classification?



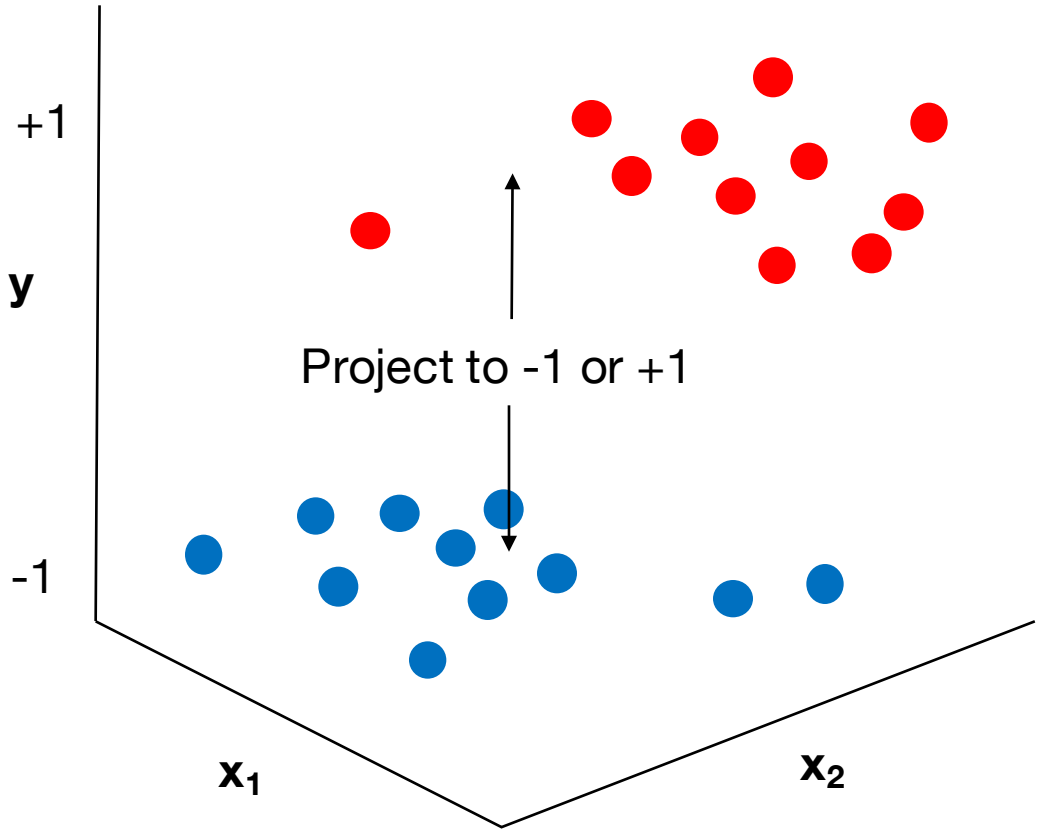
Without sgn(): regression for best fit

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

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

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

Why does linear regression with sgn() achieve classification?



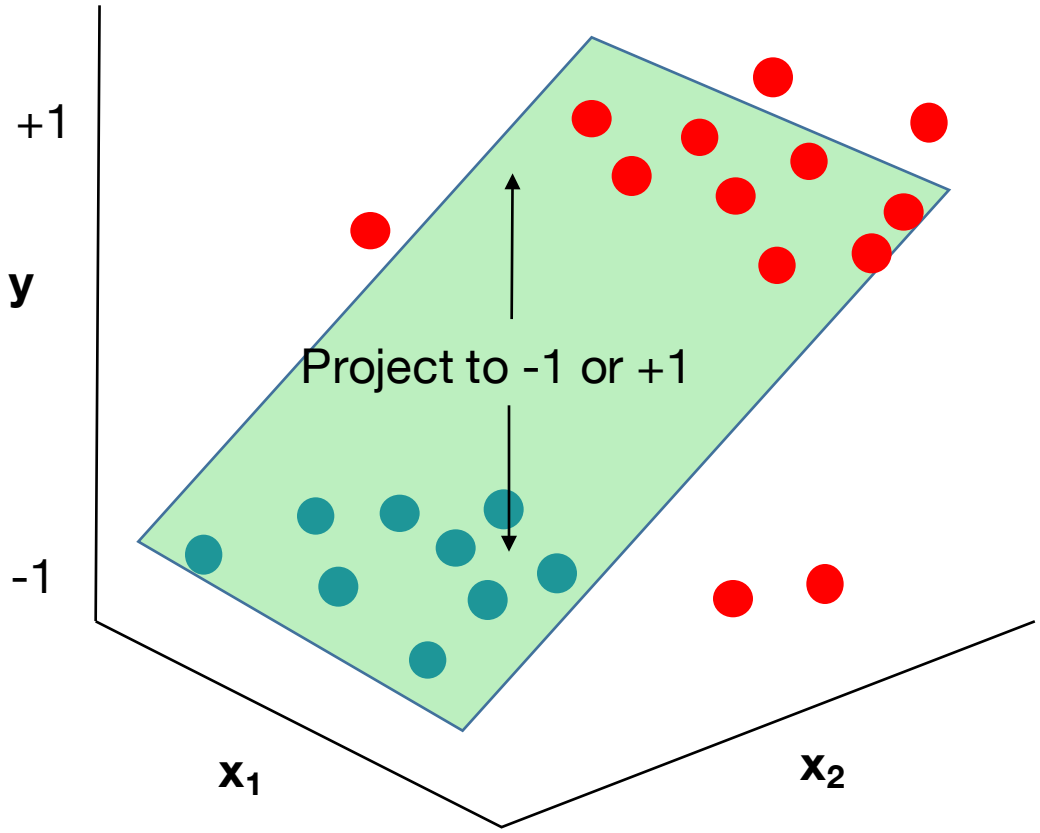
Without sgn(): regression for best fit

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

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

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

Why does linear regression with sgn() achieve classification?



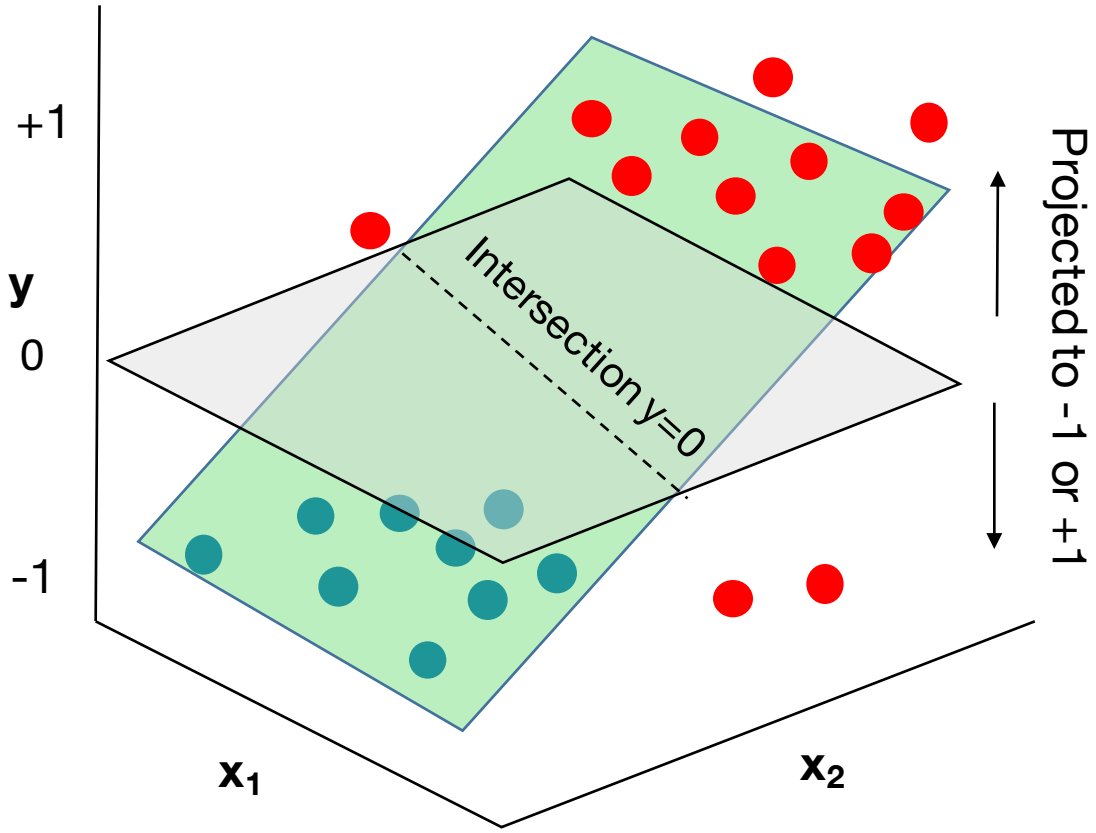
Without sgn(): regression for best fit

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

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

- 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?



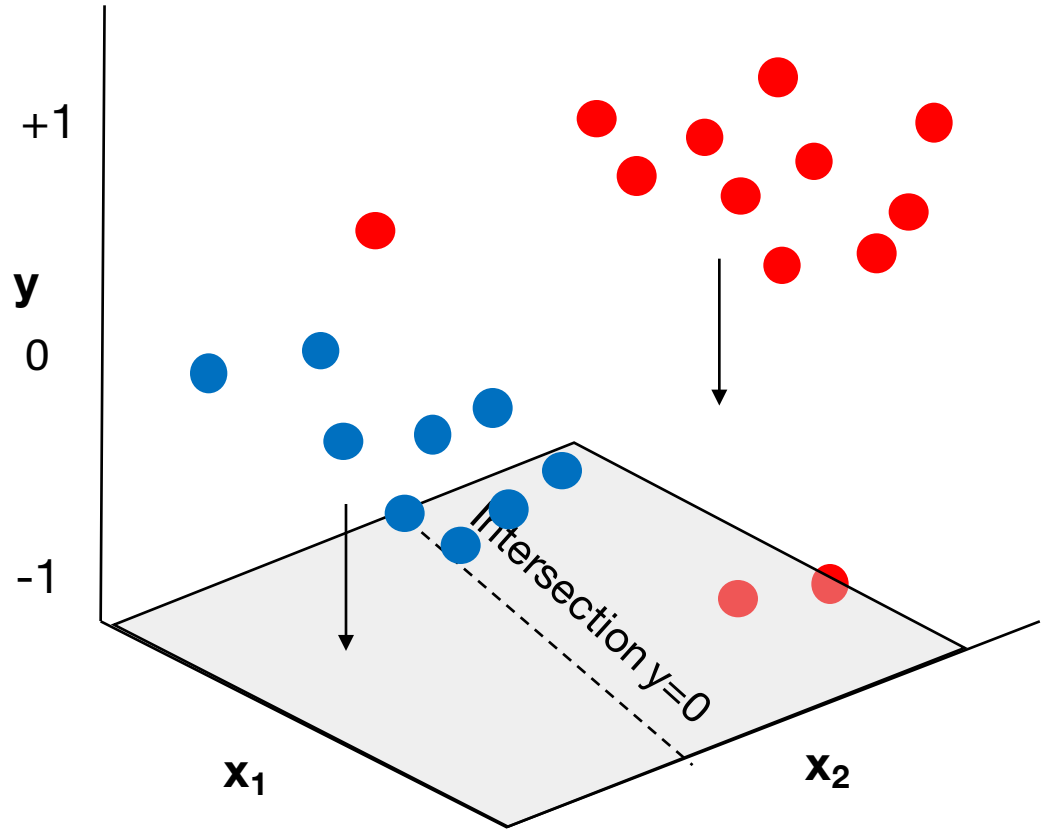
With sgn() operation:

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

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

- 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:

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

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

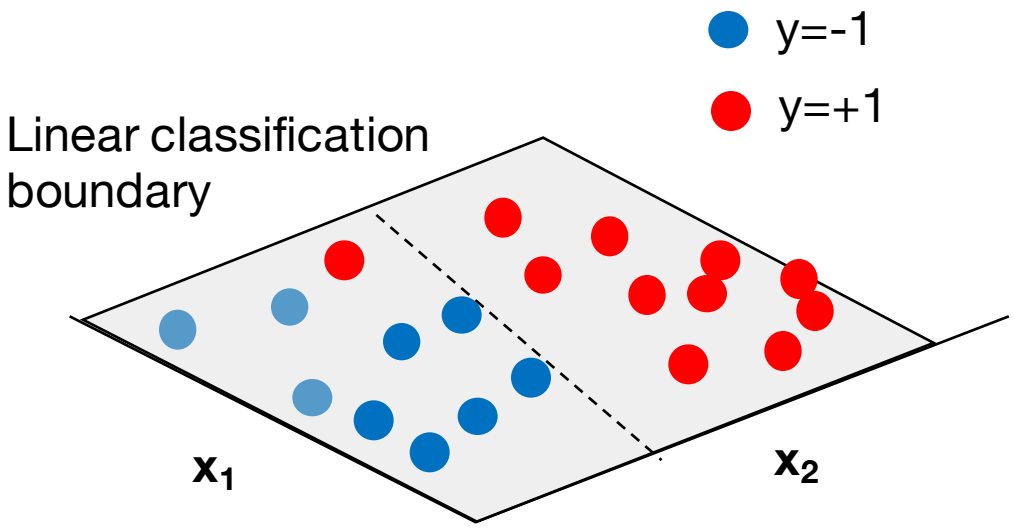
- y axis isn't really needed now & can view this decision boundary in 2D

Why does linear regression with sgn() achieve classification?

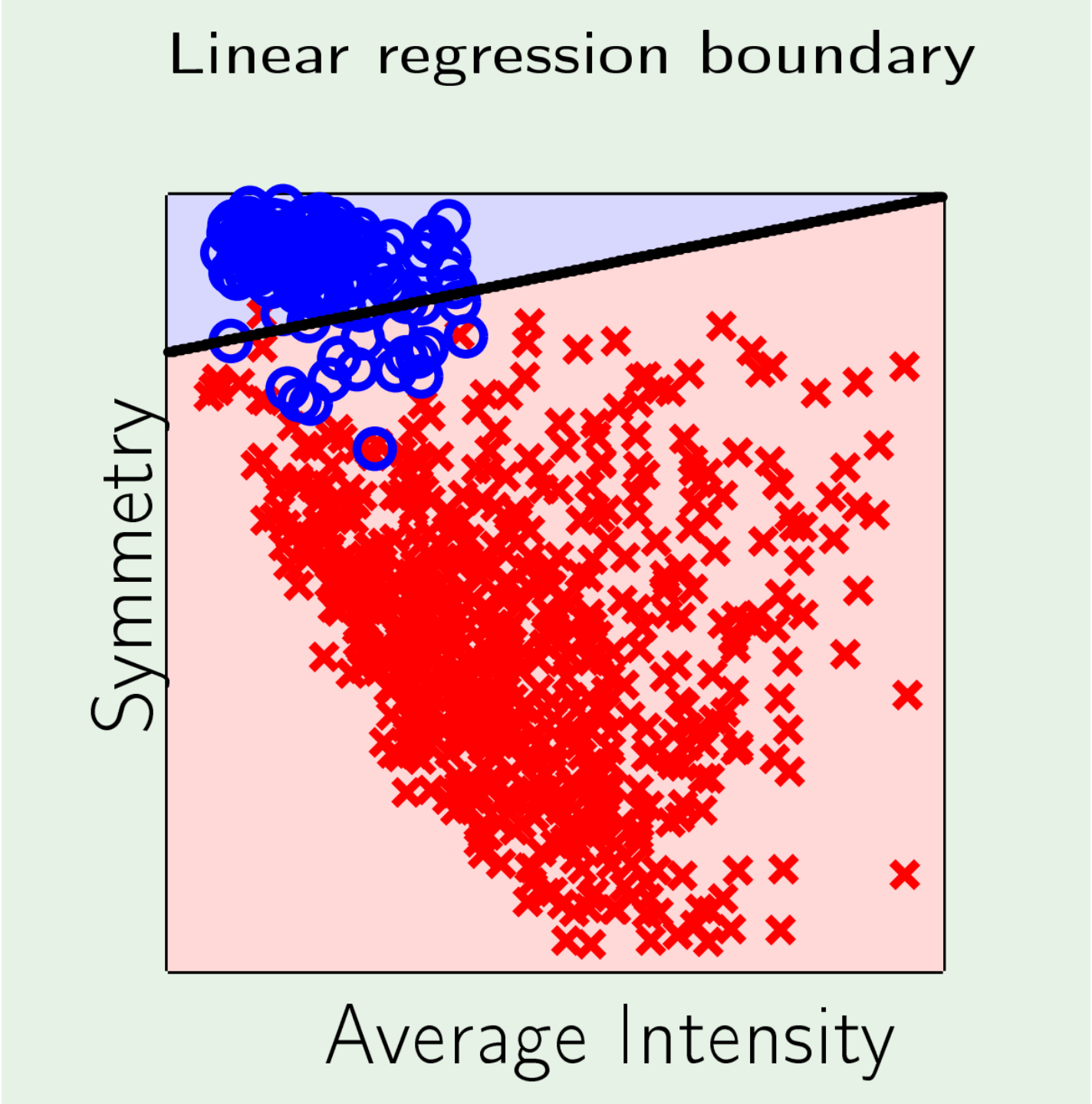
With sgn() operation:

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

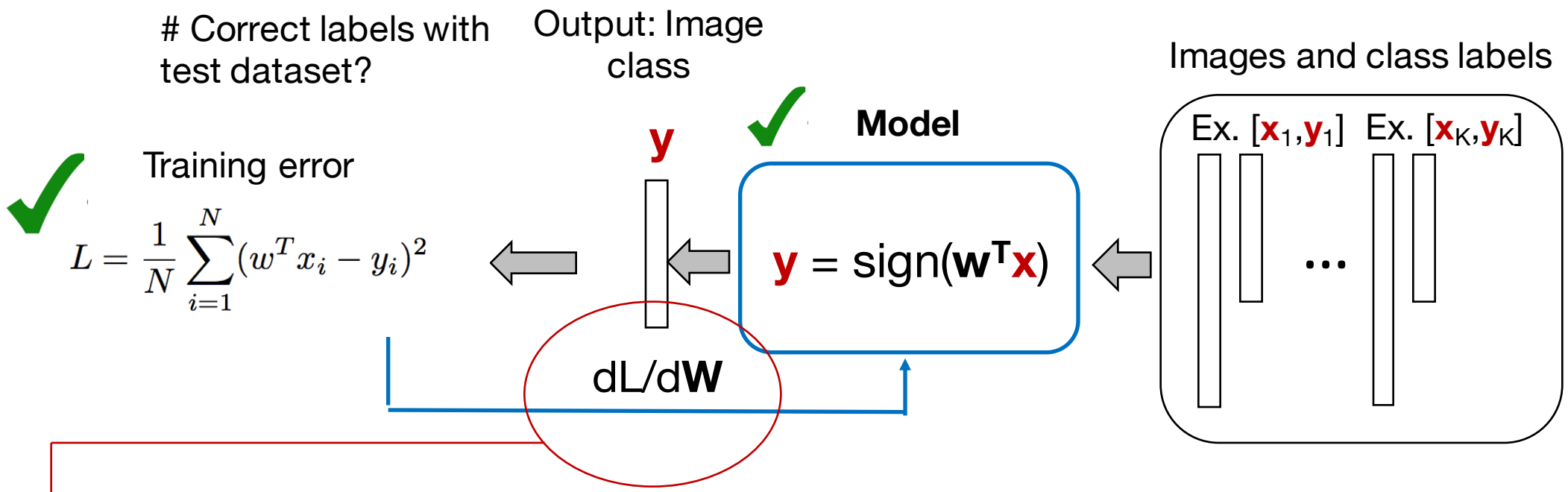
$$L = \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$



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


Linear regression boundary
Symmetry
Average Intensity

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)

$$L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n\right)^2$$

$$L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n)^2$$

$$= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

where $\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1{}^{\mathsf{T}}— \\ —\mathbf{x}_2{}^{\mathsf{T}}— \\ \vdots \\ —\mathbf{x}_N{}^{\mathsf{T}}— \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

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:

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

The linear regression algorithm

1: Construct the matrix $\mathbf{X}$ and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ as follows

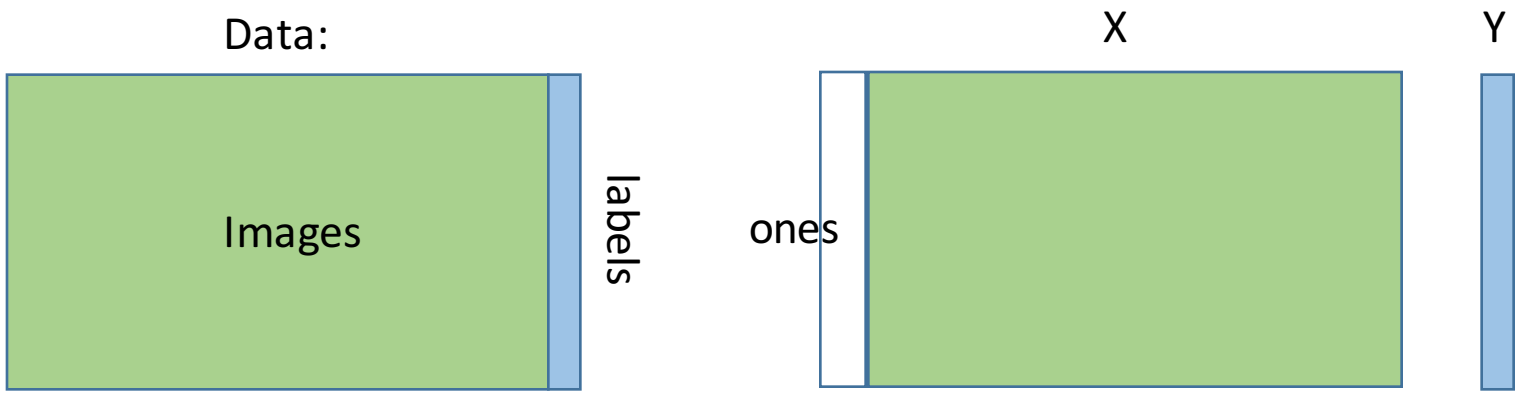
$$\underbrace{\mathbf{X} = \begin{bmatrix} —\mathbf{x}_1^\mathsf{T}— \\ —\mathbf{x}_2^\mathsf{T}— \\ \vdots \\ —\mathbf{x}_N^\mathsf{T}— \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2: Compute the pseudo-inverse $\mathbf{X}^\dagger = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}$.

3: Return $\mathbf{w} = \mathbf{X}^\dagger\mathbf{y}$.

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

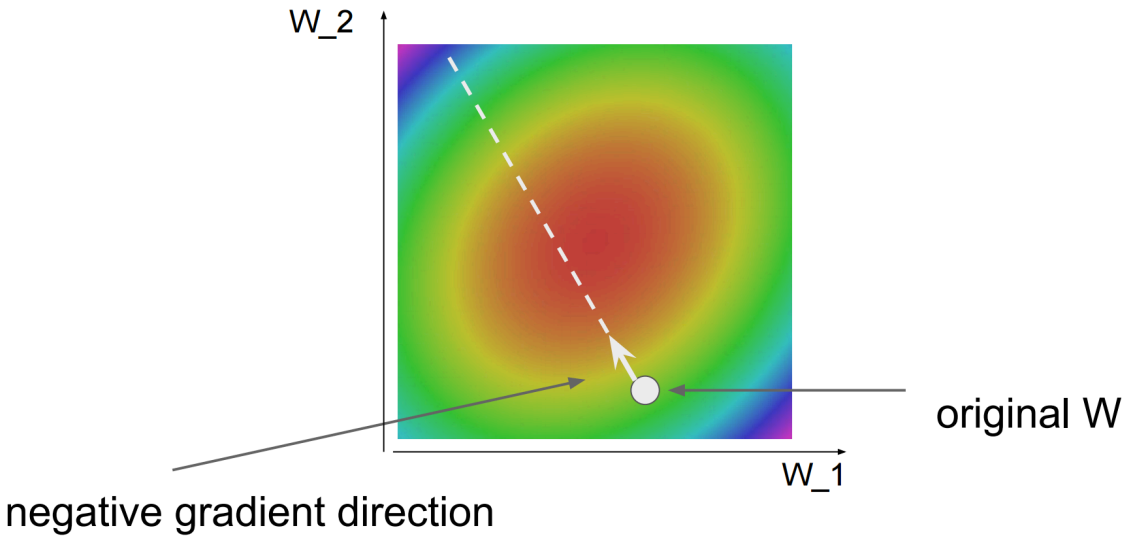
(harder)

Gradient descent: The iterative recipe

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:

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

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

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

$dL(W_1)/dW_1$ = 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:

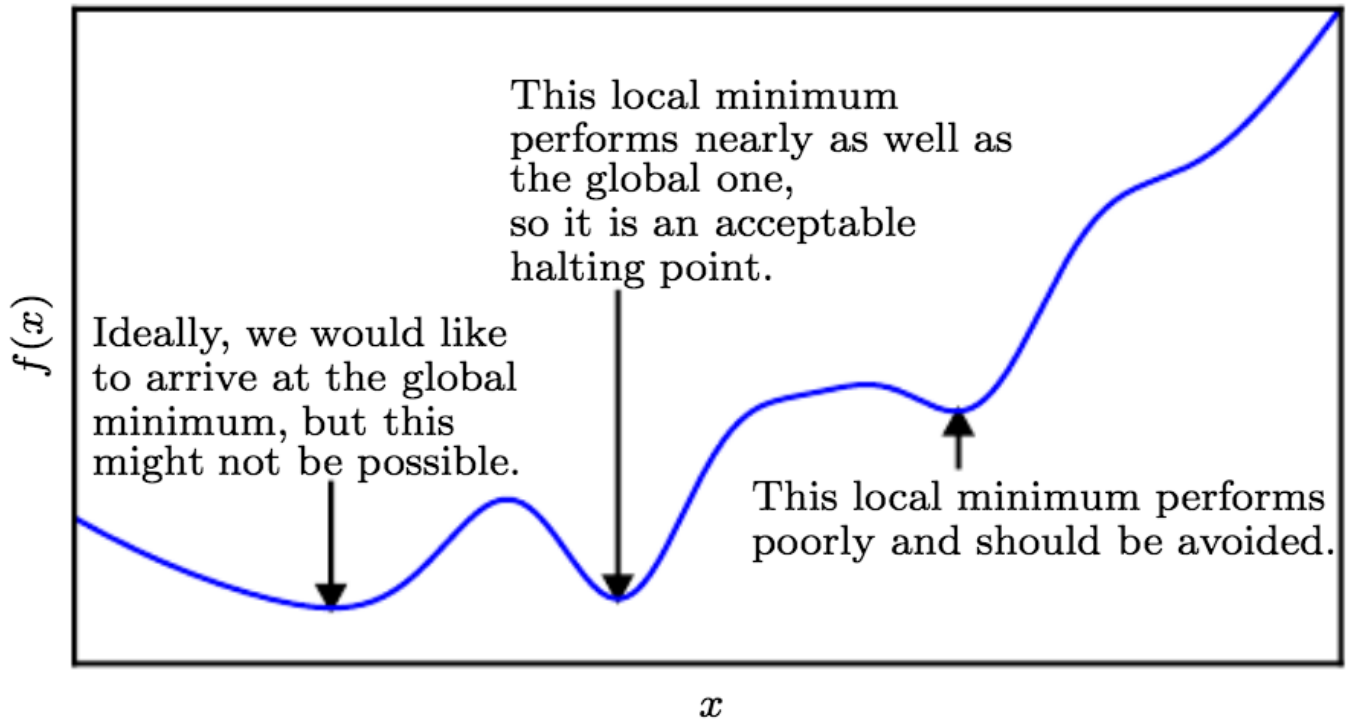
(easier)

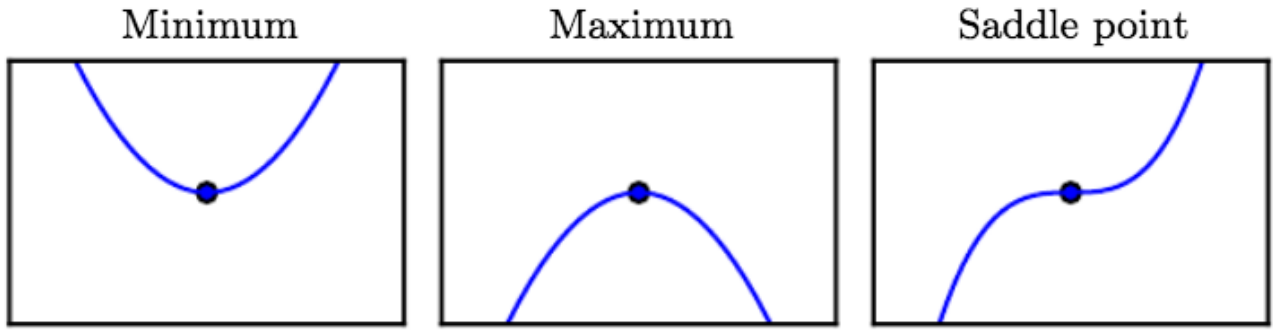
1. Pseudo-inverse (this is one of the few cases with a closed-form solution)
2. Numerical gradient descent
3. Gradient descent on the cost function with respect to W

(harder)

Some quick details about gradient descent

- For non-convex functions, local minima can obscure the search for global minima
- Analyzing critical points (plateaus) of function of interest is important
- Critical points at $df/dx = 0$
- 2nd derivative d^2f/dx^2 tells us the type of critical point:
  - Minima at $d^2f/dx^2 > 0$
  - Maxima at $d^2f/dx^2 < 0$





Some quick details about gradient descent

Often we'll have functions of m variables

$f : \mathbb{R}^n \to \mathbb{R}$ (e.g., f(**x**) = Σ (**Ax-y**)^2)

Will take partial derivatives $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ and put them in gradient vector **g:** $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$

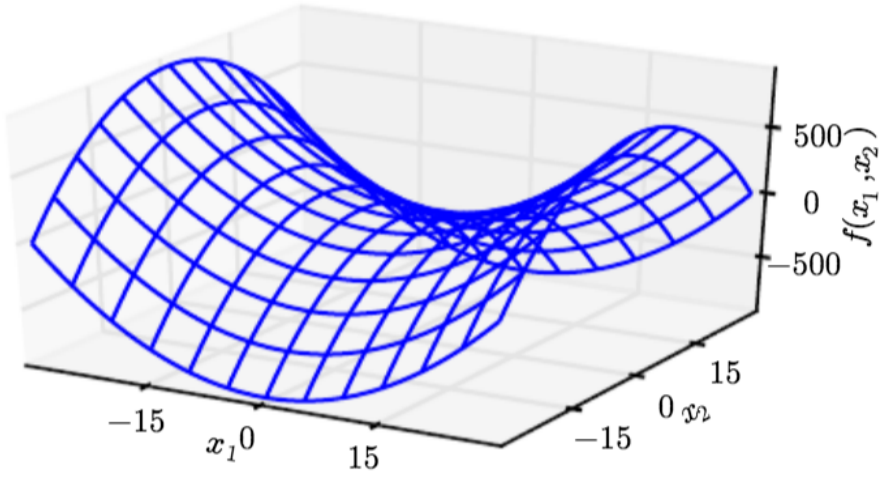
Will have many second derivatives: $\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$ Hessian Matrix

In general, we'll have functions that map m variables to n variables

$\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$ (e.g., **f**(**x**) = **Wx**, **W** is n x m)

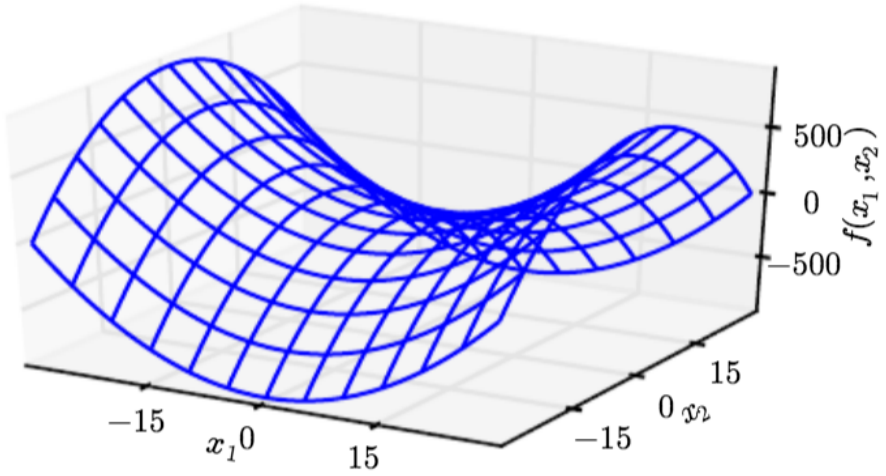
$\boldsymbol{J} \in \mathbb{R}^{n \times m}$ of $\boldsymbol{f}$: $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$ Jacobian Matrix

Quick example



$$f(\mathbf{x}) = x_1^2 - x_2^2$$

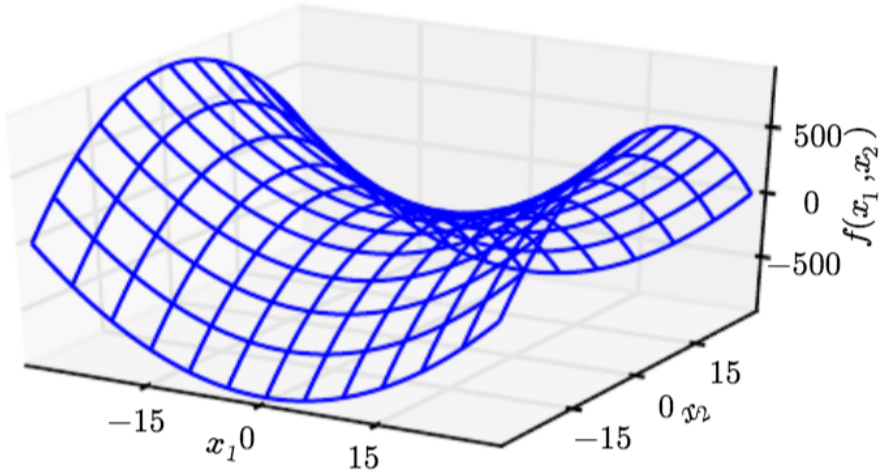
Quick example



$$f(\mathbf{x}) = x_1^2 - x_2^2$$

$$g = \begin{bmatrix} 2x_1 \\ -2x_2 \end{bmatrix}$$

Quick example



$$f(\mathbf{x}) = x_1^2 - x_2^2$$

$$g = \begin{bmatrix} 2x_1 \\ -2x_2 \end{bmatrix} \qquad H = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

- Convex functions have positive semi-definite Hessians (Trace >= 0)
- Trace/eigenvalues of Hessian are useful evaluate critical points & guide optimization

Steepest descent and the best step size ε

1. Evaluate function $f(\mathbf{x}^{(0)})$ at an initial guess point, $\mathbf{x}^{(0)}$
2. Compute gradient $\mathbf{g}^{(0)} = \nabla_x f(\mathbf{x}^{(0)})$
3. Next point $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \boldsymbol{\varepsilon}^{(0)}\mathbf{g}^{(0)}$
4. Repeat – $\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \boldsymbol{\varepsilon}^{(n)}\mathbf{g}^{(n)}$, until $|\mathbf{x}^{(n+1)}-\mathbf{x}^{(n)}| <$ threshold t

```
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x
    cur_x -= gamma * df(prev_x)
    previous_step_size = abs(cur_x - prev_x)
    iters+=1
```

Steepest descent and the best step size ε

What is a good step size $\boldsymbol{\varepsilon}^{(n)}$?

Steepest descent and the best step size ε

What is a good step size $\boldsymbol{\varepsilon}^{(n)}$?

To find out, take 2nd order Taylor expansion of *f* (a good approx. for nearby points):

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

Steepest descent and the best step size ε

What is a good step size $\boldsymbol{\varepsilon}^{(n)}$?

To find out, take 2nd order Taylor expansion of *f* (a good approx. for nearby points):

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

Then, evaluate at the next step:

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$$

Steepest descent and the best step size ε

What is a good step size $\boldsymbol{\varepsilon}^{(n)}$?

To find out, take 2nd order Taylor expansion of *f* (a good approx. for nearby points):

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

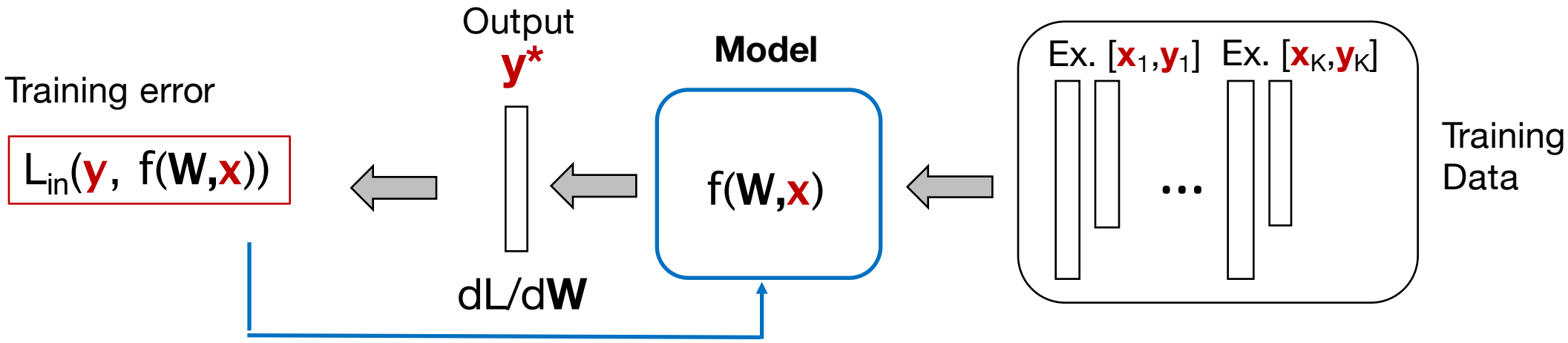
Then, evaluate at the next step:

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$$

Solve for optimal step (when Hessian is positive): $\boxed{\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}}$

Summary of machine learning pipeline:

1. Network Training

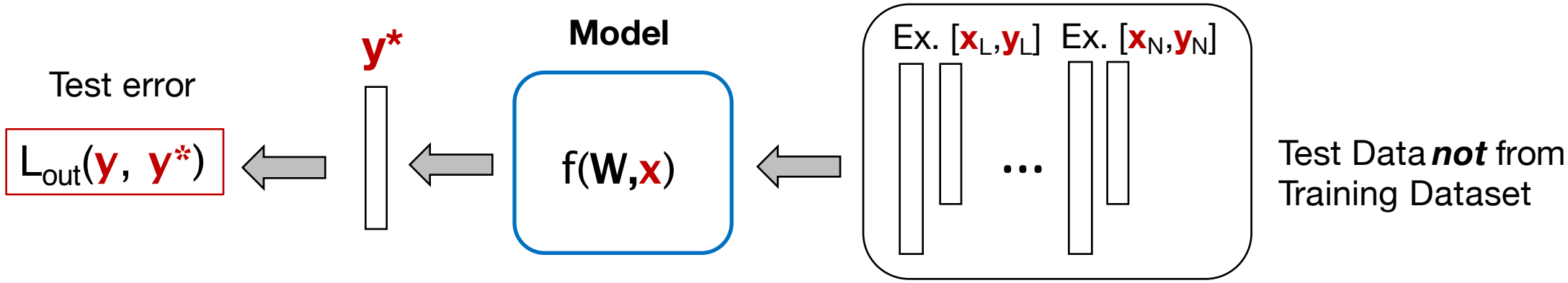


What we need for network training:

1. **Labeled examples**
2. **A model and loss function**
3. **A way to minimize the loss function L**

Summary of machine learning pipeline:

2. Network Testing



What we need for network testing:

4. *Unique* labeled test data
5. Evaluation of model error