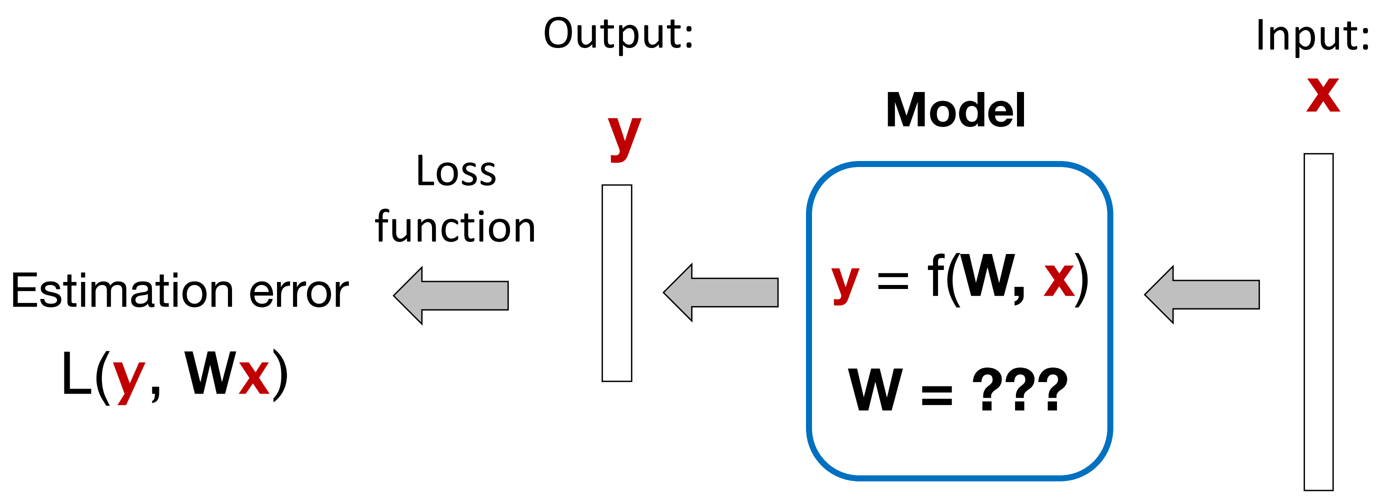
deep imaging

Lecture 6, Part 2: Ingredients for Machine Learning

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer

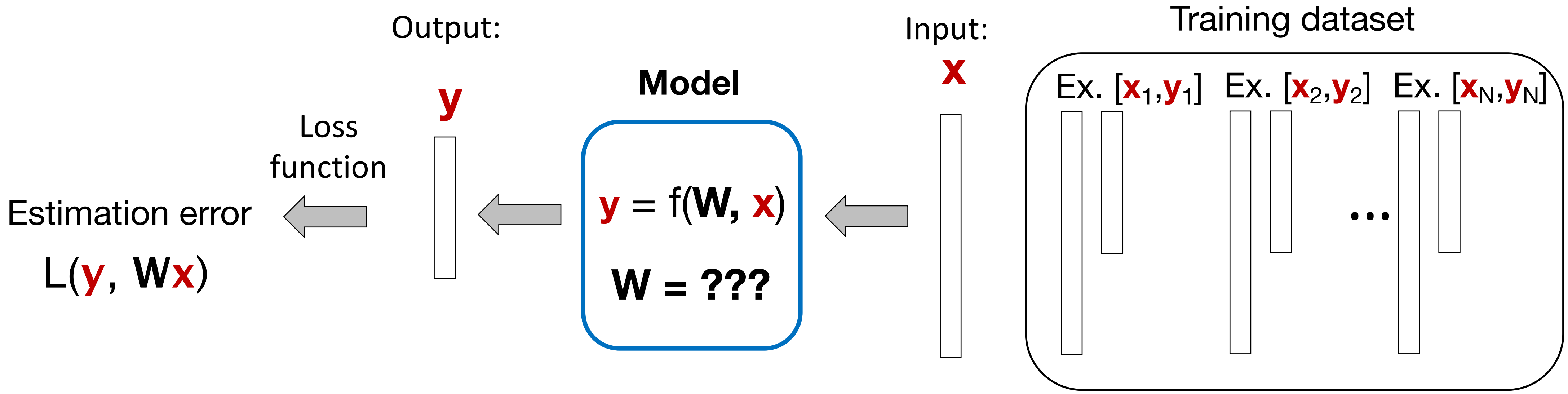
Pipeline for machine learning



Changes for machine learning framework:

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

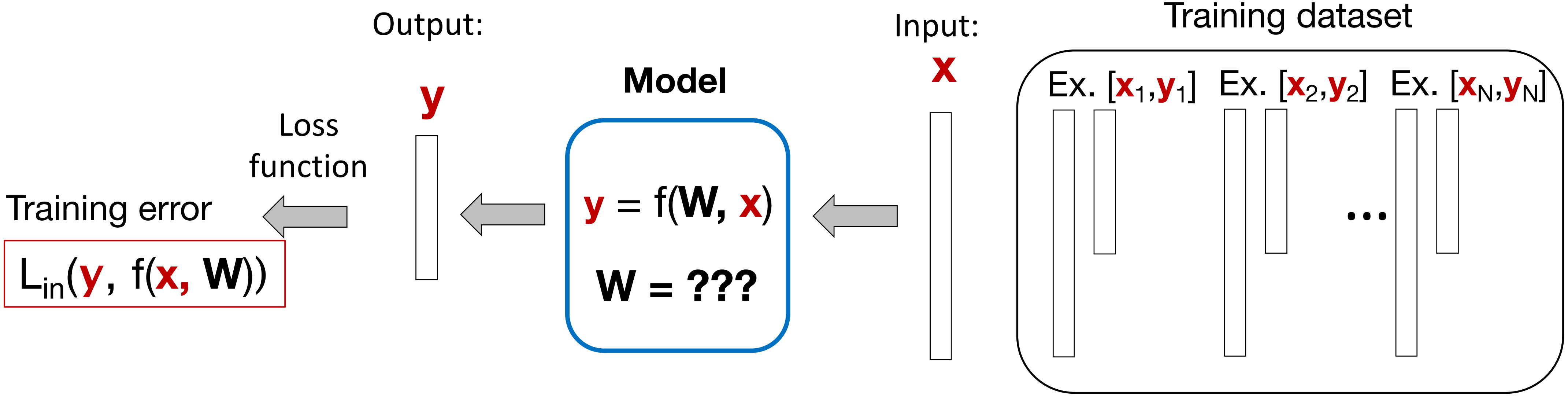
Pipeline for machine learning



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**)

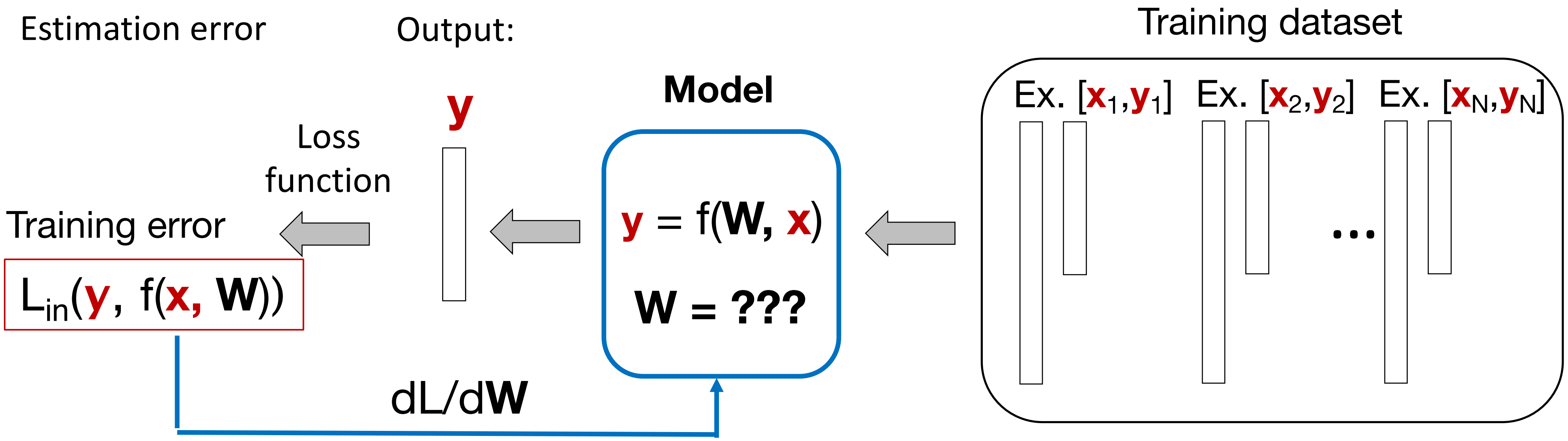
Pipeline for machine learning



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**)
3. To do so, use a *loss function* L that depends upon the training inputs (x,y) *and* the model (**W**)

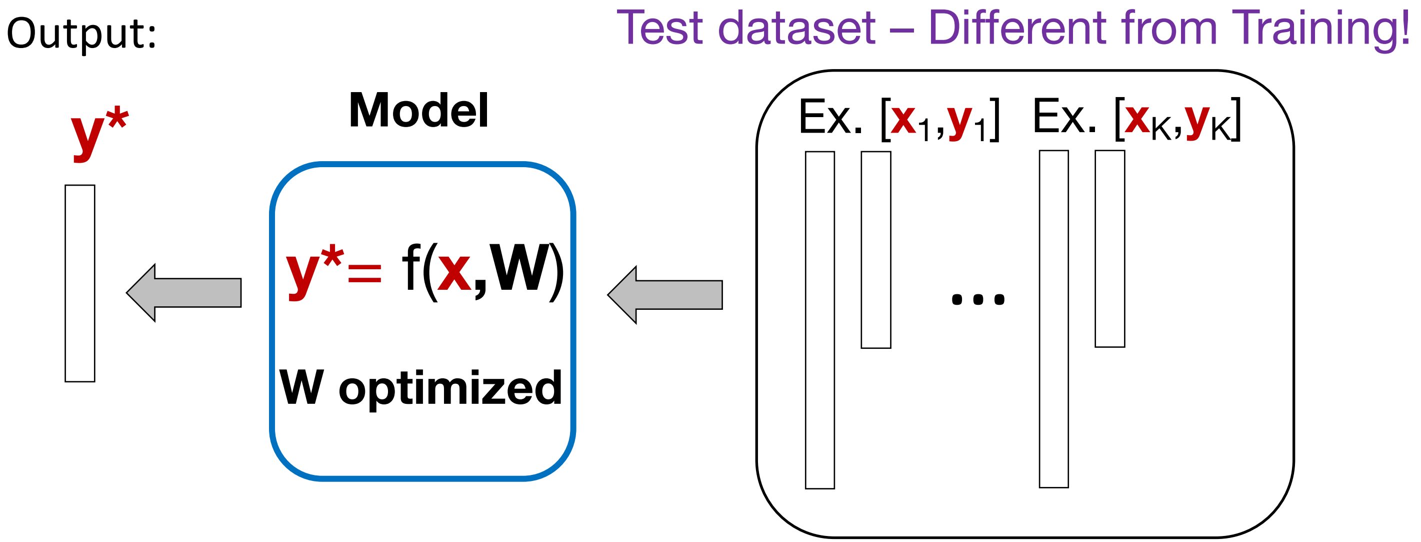
Pipeline for machine learning



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**)
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

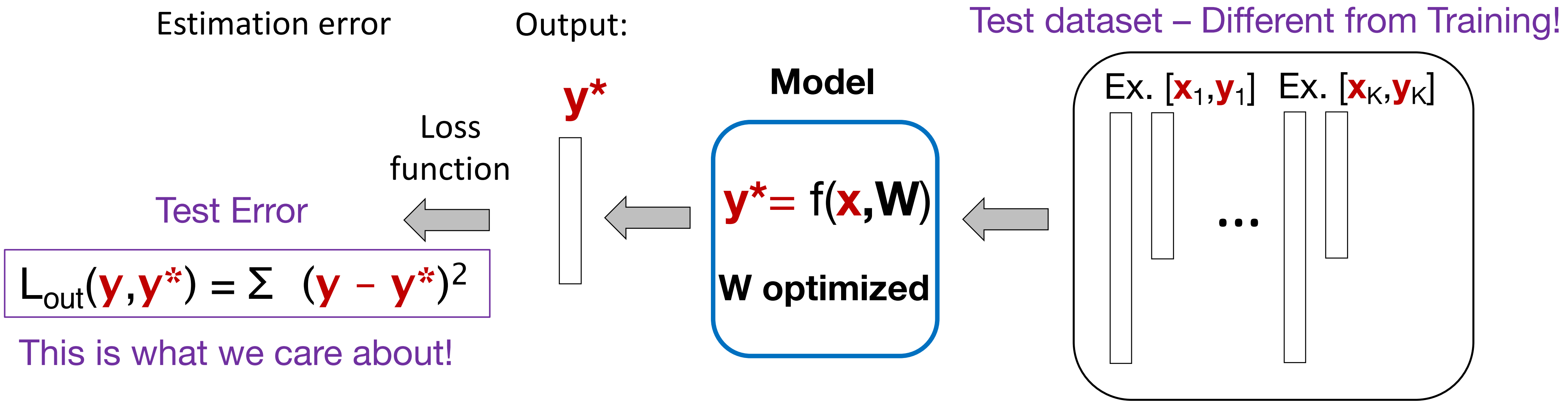


In a *separate step*, we then need to do the following to test the network:

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

deep imaging

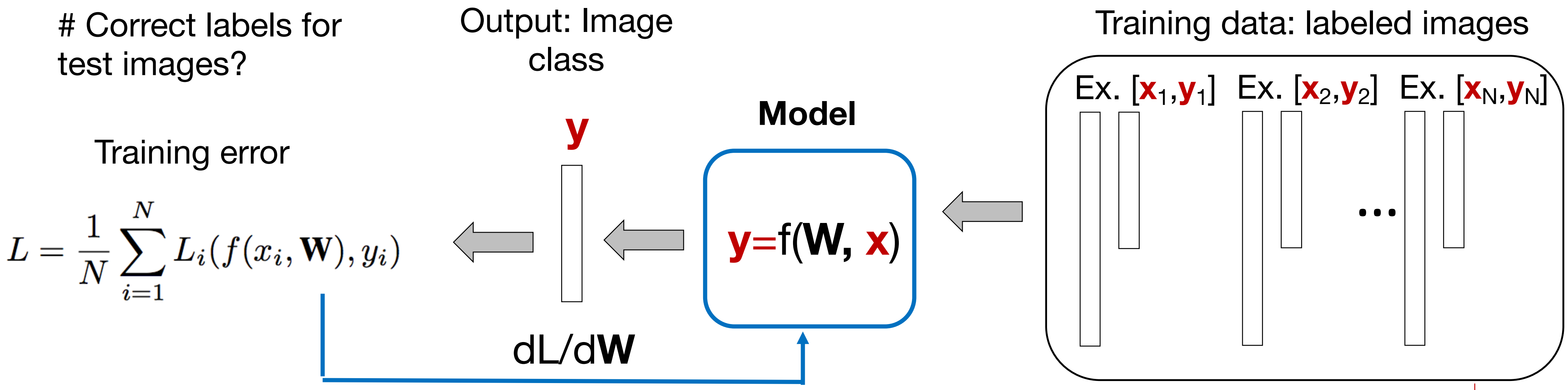
Pipeline for machine learning



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



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

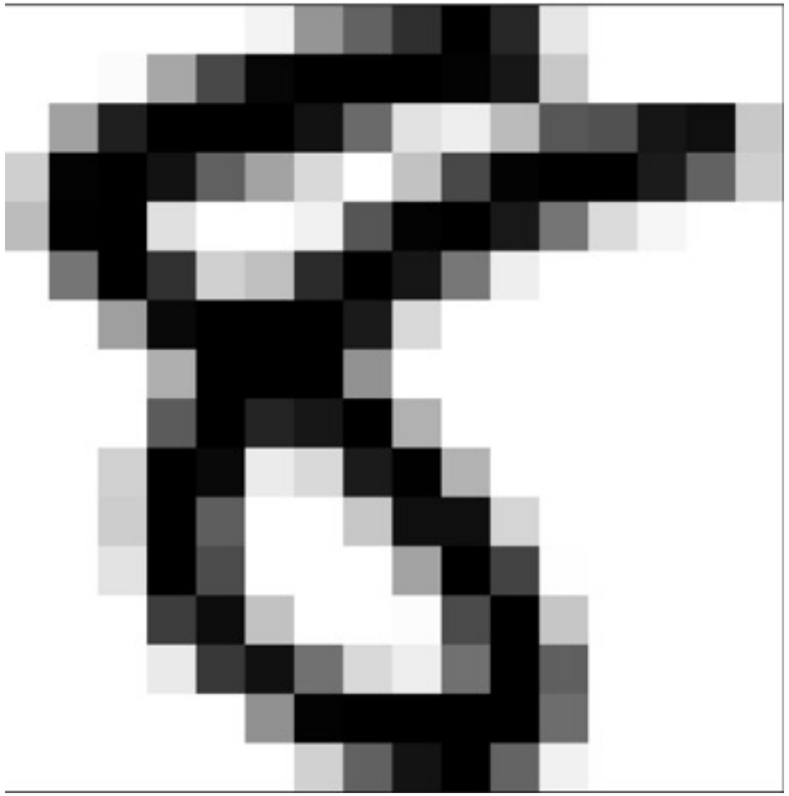
1. Labeled examples $\{(x_i, y_i)\}_{i=1}^{N}$

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



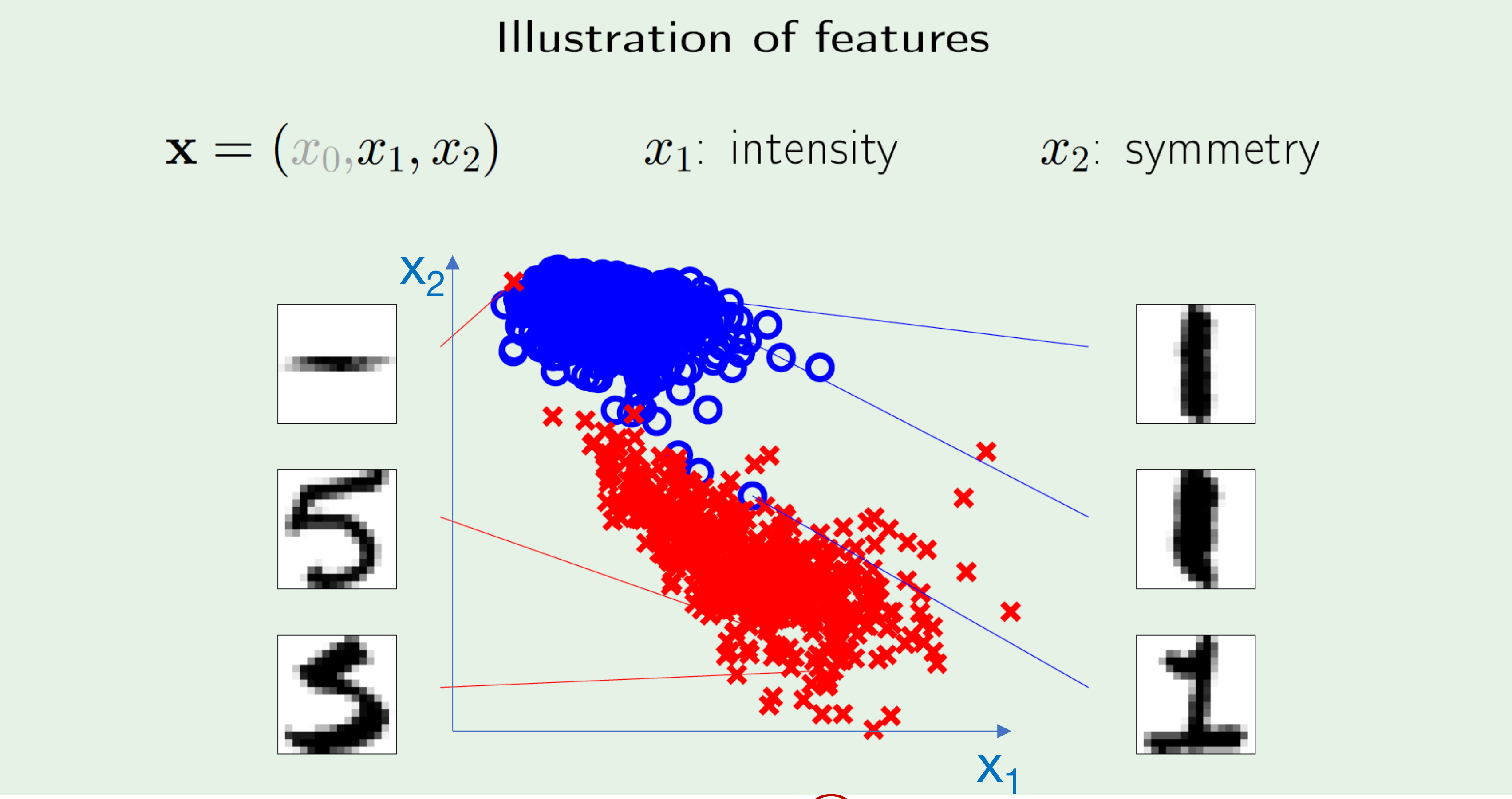
X = 28x28 pixel matrix

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)



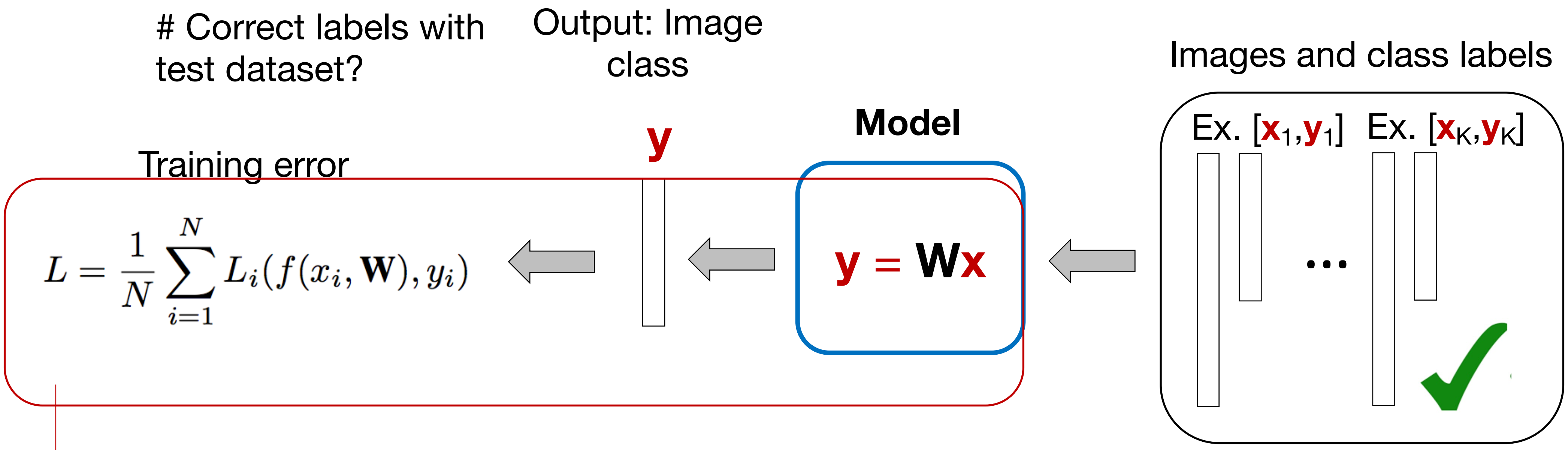
Dataset: 1000 examples of 1's and 5's mapped to $\mathbf{x_j} = (1, x_1, x_2)$, with associated label y_j = 1 or -1

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

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

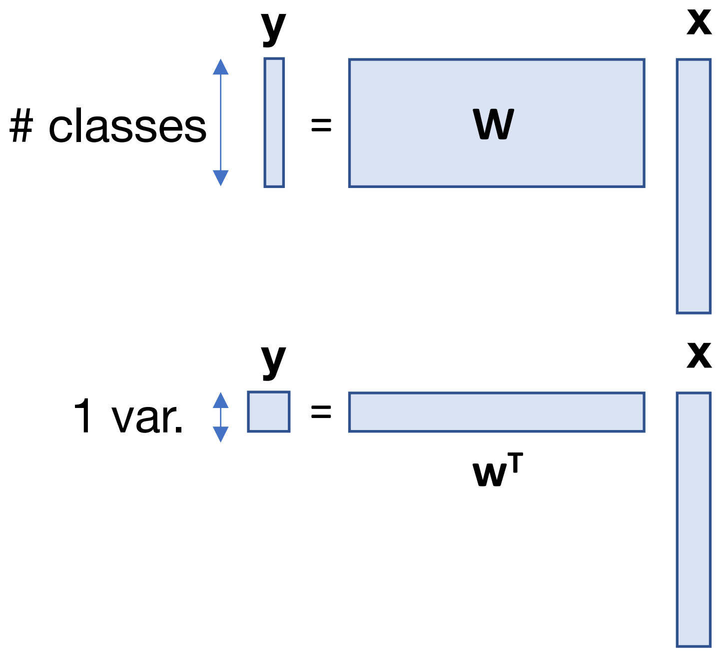
Let’s start with a simpler approach: linear regression

deep imaging

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

General linear model:

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



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

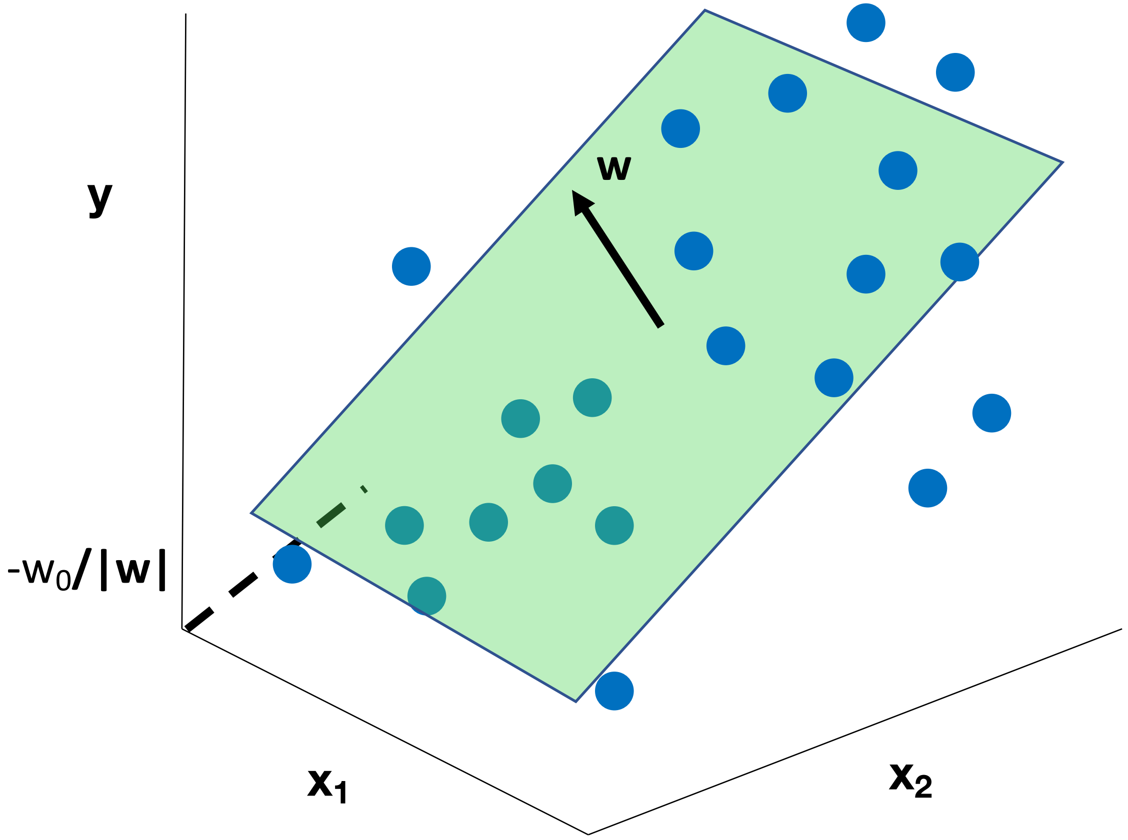
$$L = \frac{1}{N}\sum_{i=1}^{N} (w^T x_i - y_i)^2$$

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?



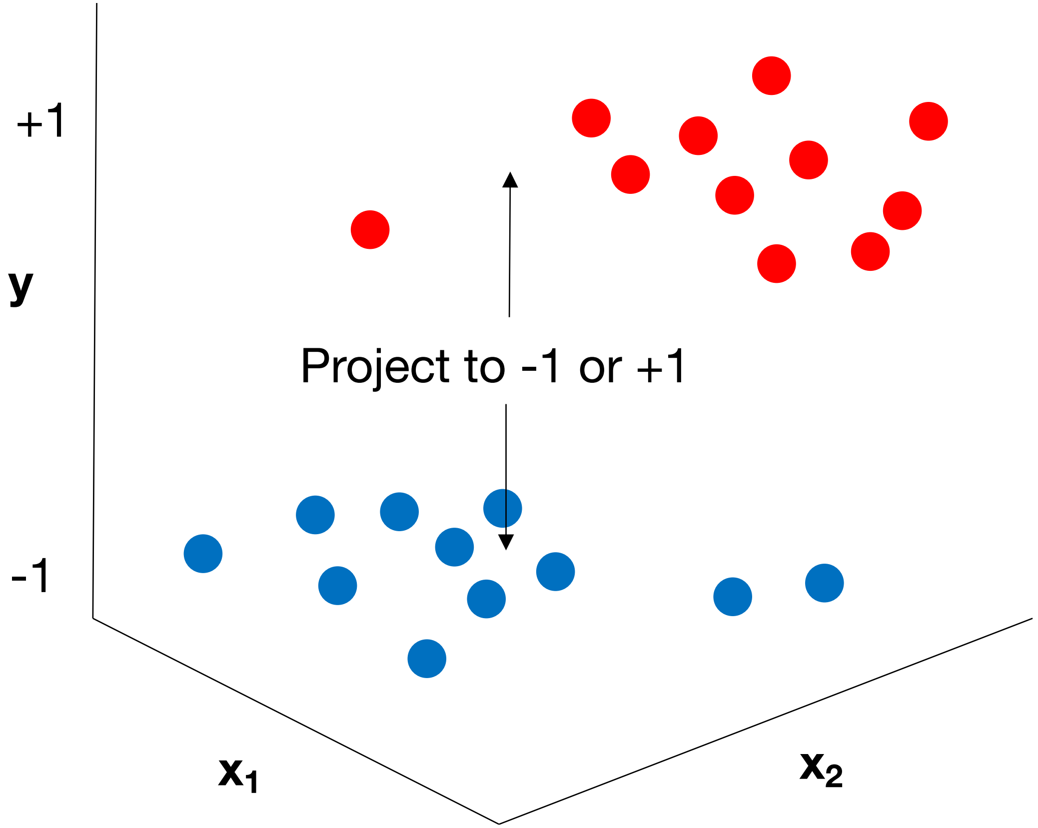
Without sgn(): regression for best fit

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

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

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

Why does linear regression with sgn() achieve classification?



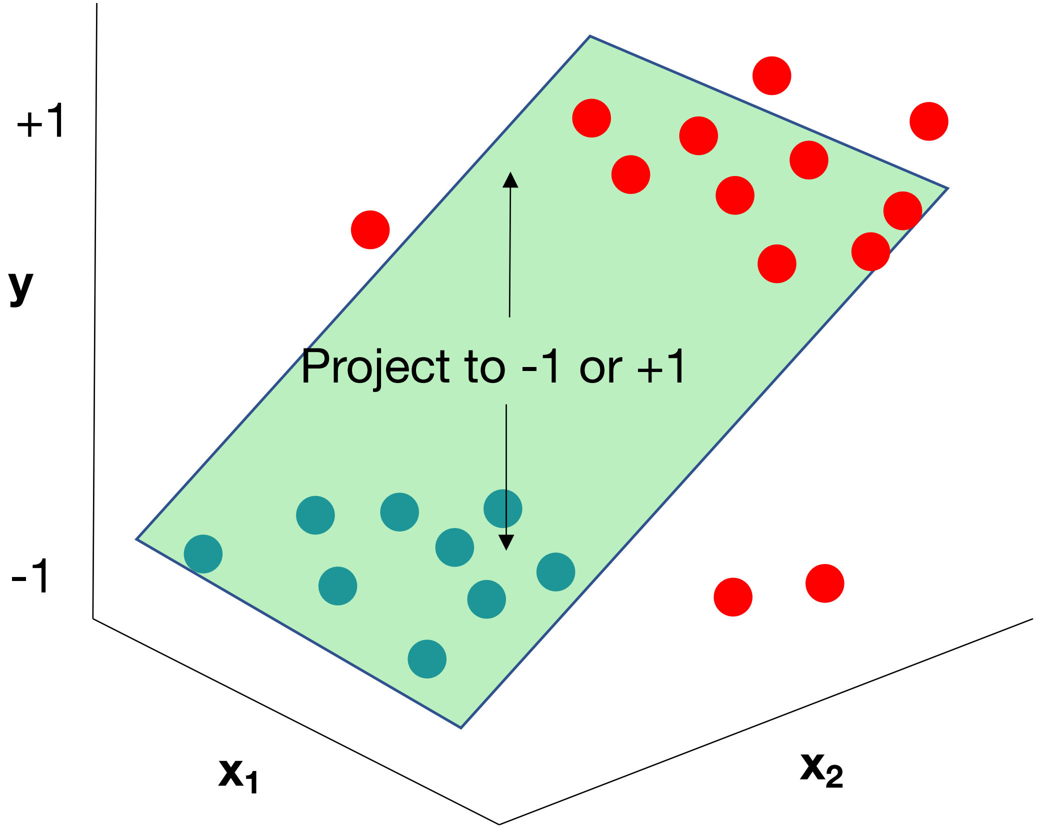
Without sgn(): regression for best fit

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

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

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

Why does linear regression with sgn() achieve classification?



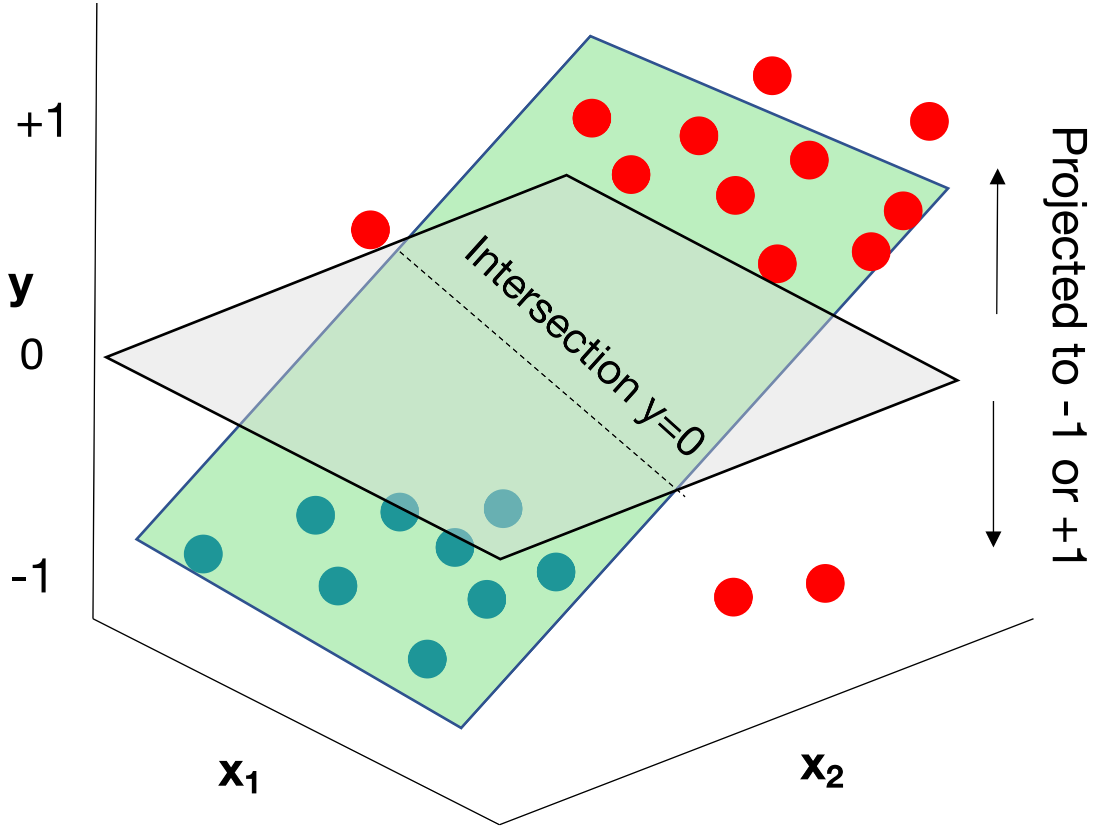
Without sgn(): regression for best fit

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

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

- 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?



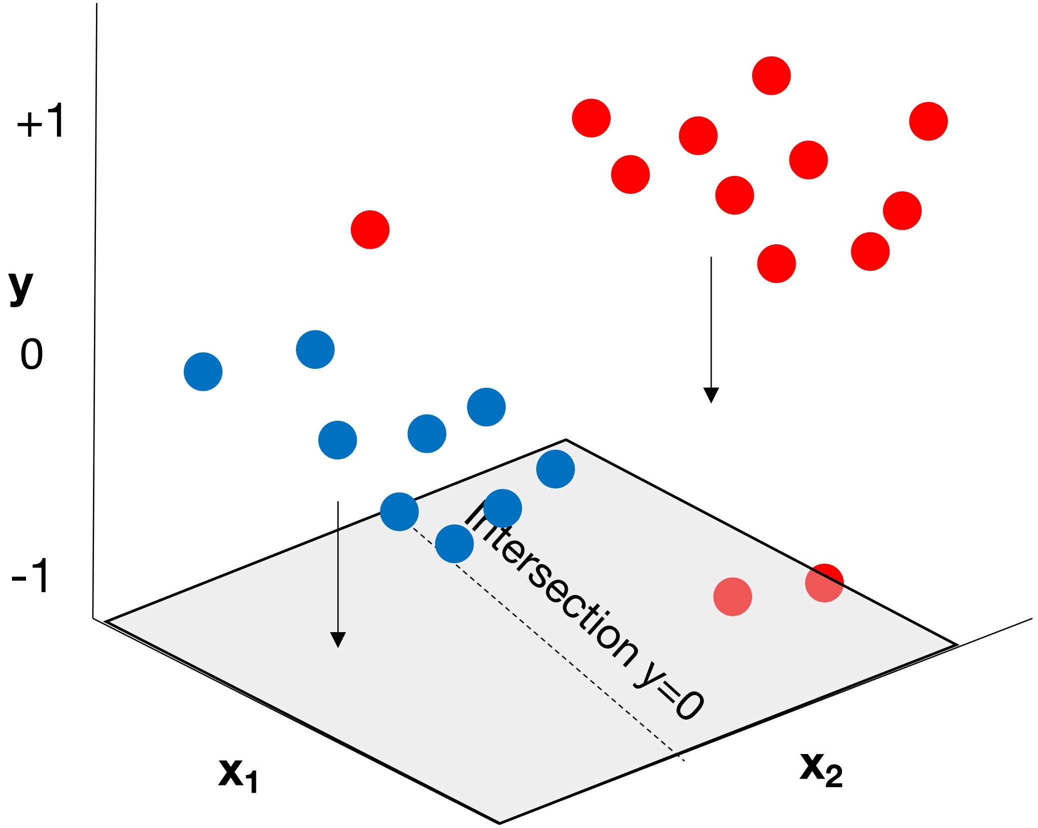
With sgn() operation:

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

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

- 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:

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

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

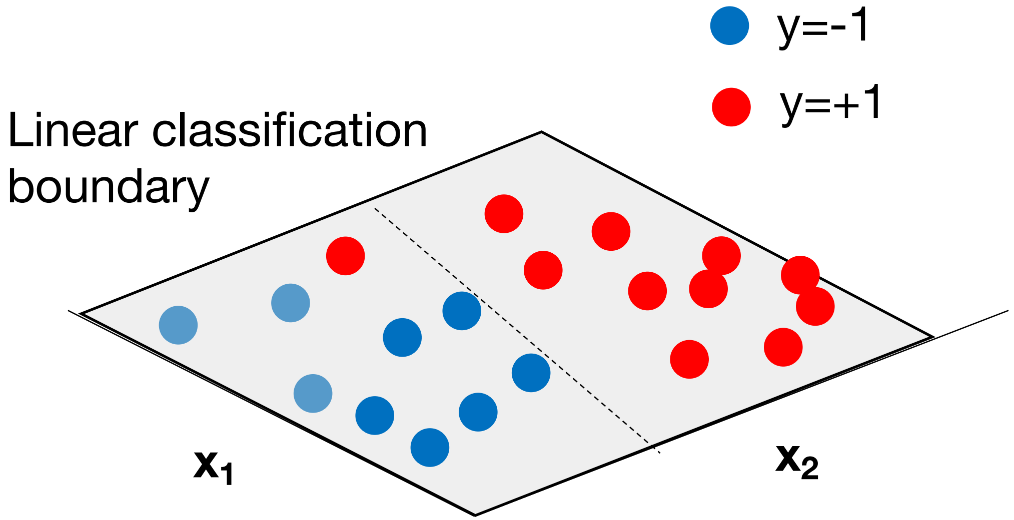
- y axis isn't really needed now & can view this decision boundary in 2D

Why does linear regression with sgn() achieve classification?

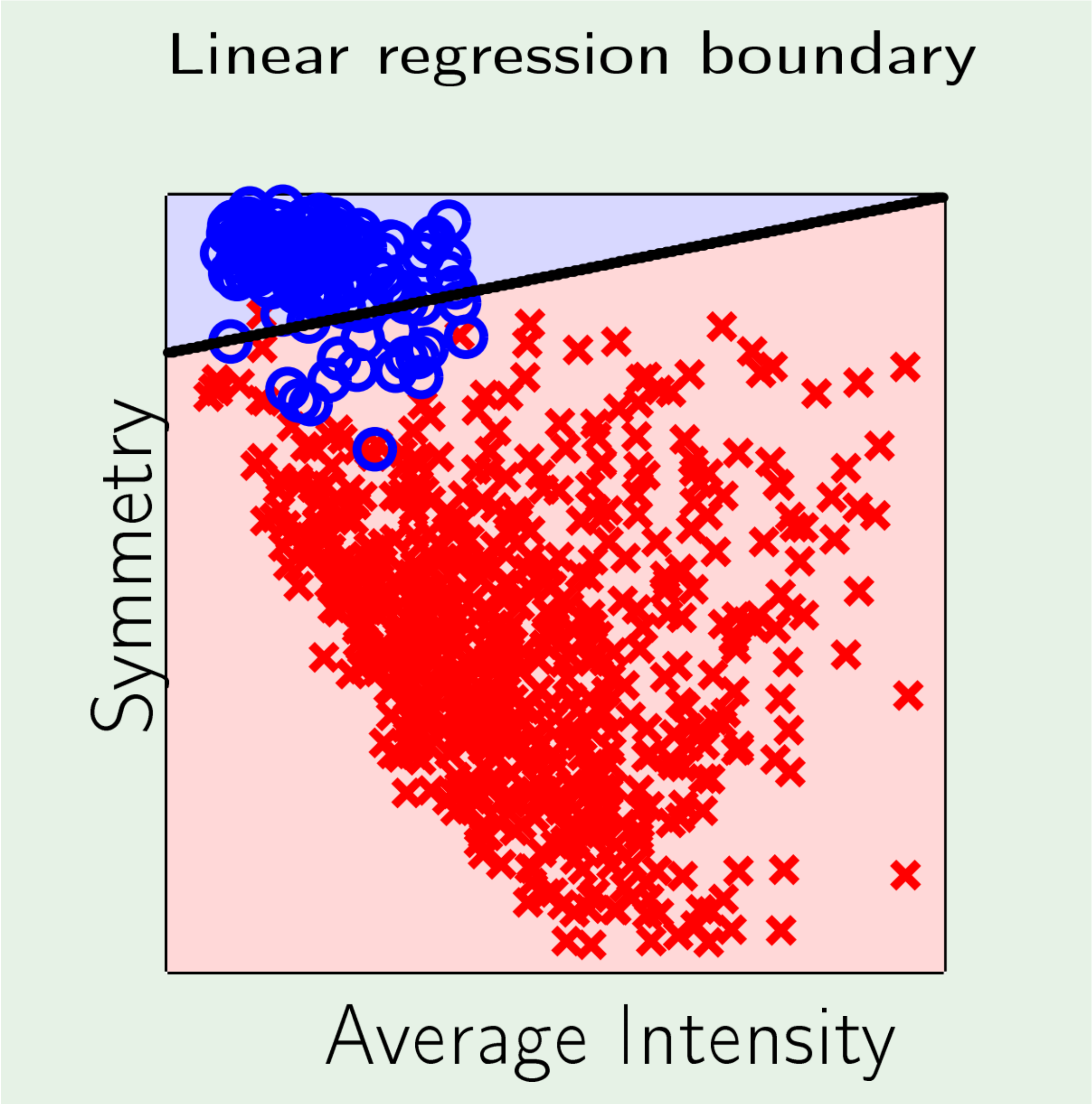
With sgn() operation:

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

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



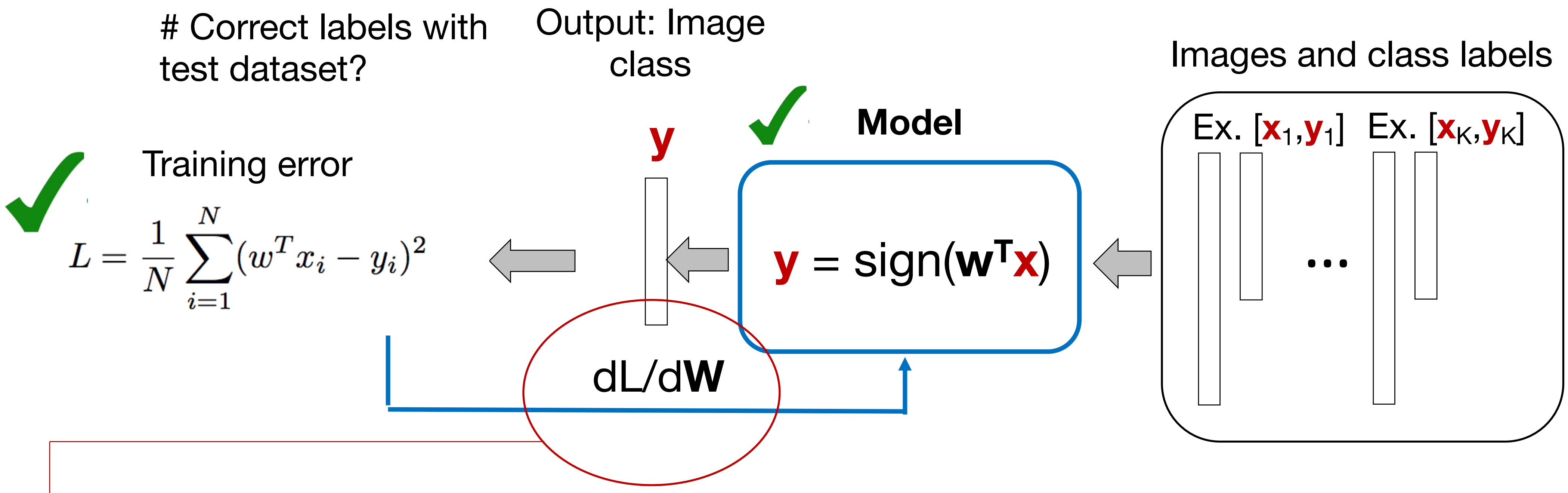
Sign operation takes linear regression and makes it a classification operation!



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

deep imaging

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

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)

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* into 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!

deep imaging

1. Turning linear regression *for unknown weights W* into 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!

$$X = \begin{bmatrix} \boxed{\quad x_1 \quad} \\ \boxed{\quad x_2 \quad} \\ \\ \boxed{\quad x_N \quad} \end{bmatrix}$$

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

1. Turning linear regression *for unknown weights W* into 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!

$$X = \begin{bmatrix} \boxed{x_1} \\ \boxed{x_2} \\ \vdots \\ \boxed{x_N} \end{bmatrix}$$

Each training image is 1 row of X

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

Each training label is 1 entry of y

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

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

deep imaging

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

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

Take derivative wrt w and set to 0:

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

Solution is pseudo-inverse:

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

Steps for Pseudo-inverse:

1. Construct matrix X and vector y from data

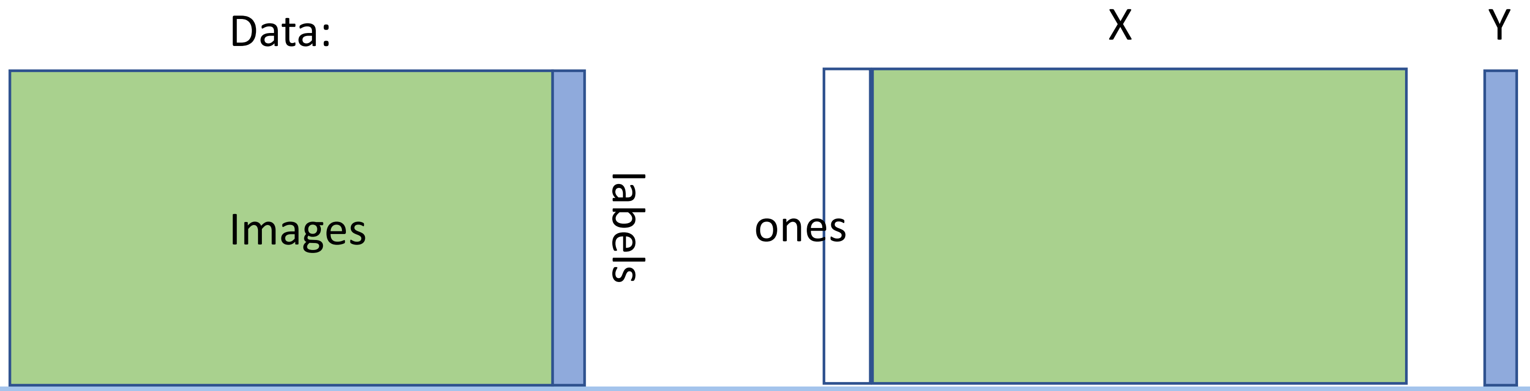
   Each training image is 1 row of X

   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)
```



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)

Gradient descent: The iterative recipe

deep imaging

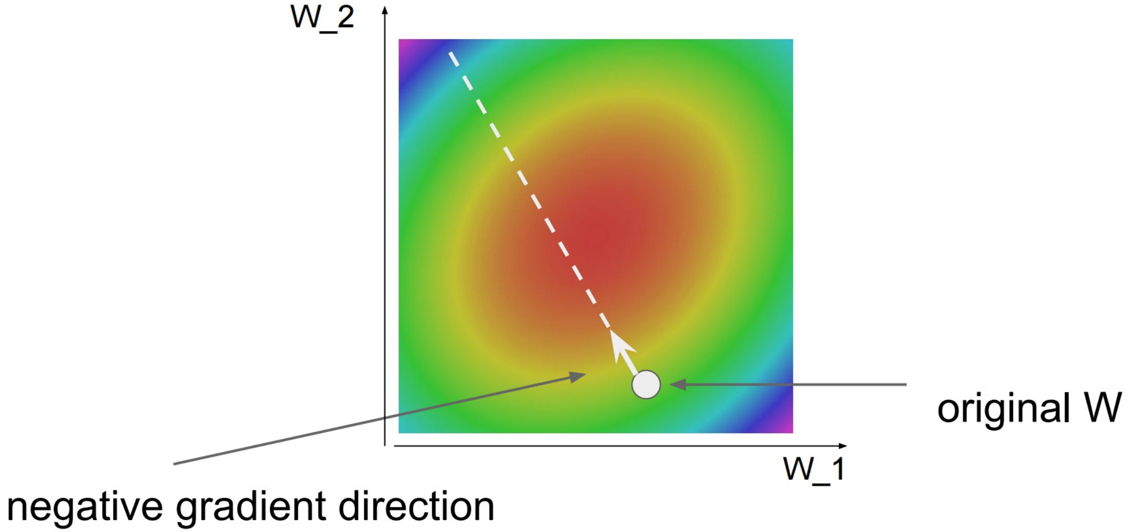
Initialize: Start with a guess of **W**

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:

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

Gradient descent: Numerical evaluation example

With a matrix, compute this for each entry:

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

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:

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

<u>Example:</u>

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

$\boxed{dL(W_1)/dW_1 = 10}$

Gradient descent: Numerical evaluation example

deep imaging

With a matrix, compute this for each entry:

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

<u>Example:</u>

$W = [1,2;3,4]$ $\quad$ $W_1+h = [1.001,2;3,4]$ $\quad$ $dL(W_1)/dW_1 = 12.8\text{-}12.79/.001$

$L(W, x, y) = 12.79$ $\quad$ $L(W_1+h, x, y) = 12.8$

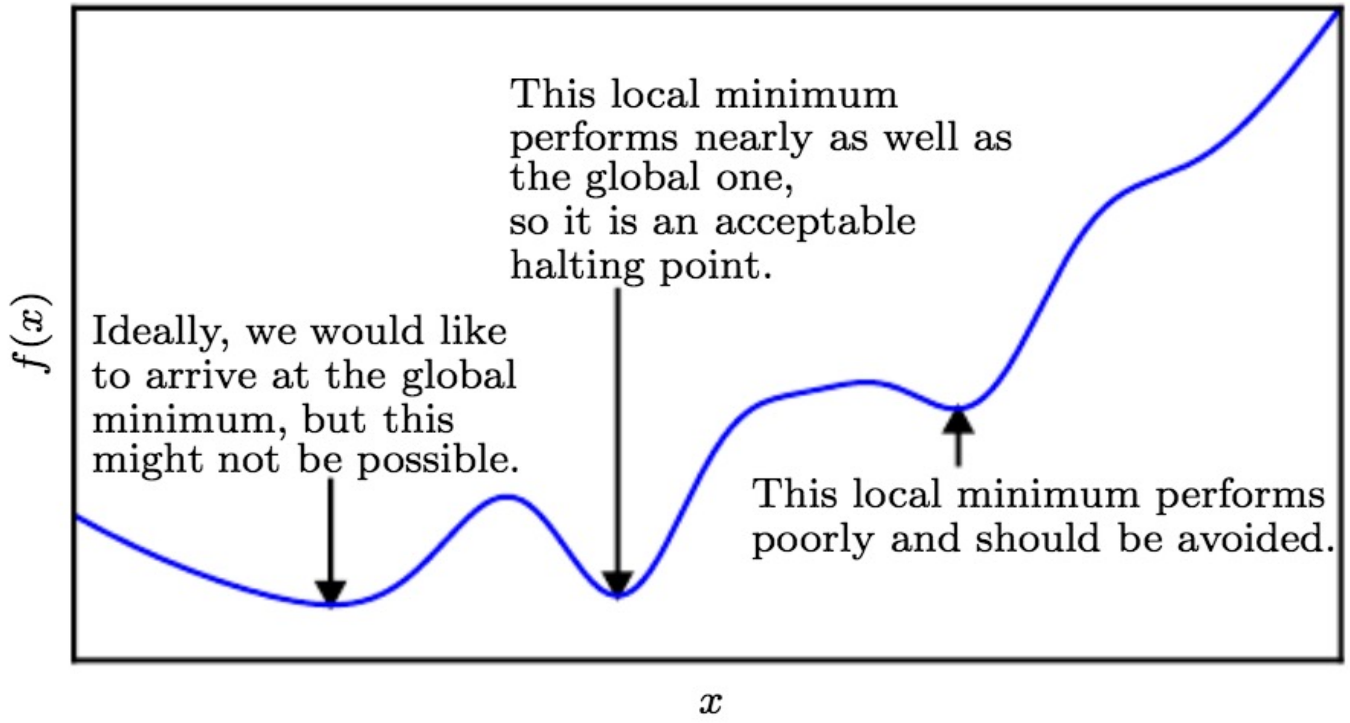
$\boxed{dL(W_1)/dW_1 = 10}$

- Repeat for all entries of **W**, dL/d**W** will have NxM entries for NxM matrix
- **This is a “brute force” approach – not ideal, but sometimes helpful**

Some quick details about gradient descent

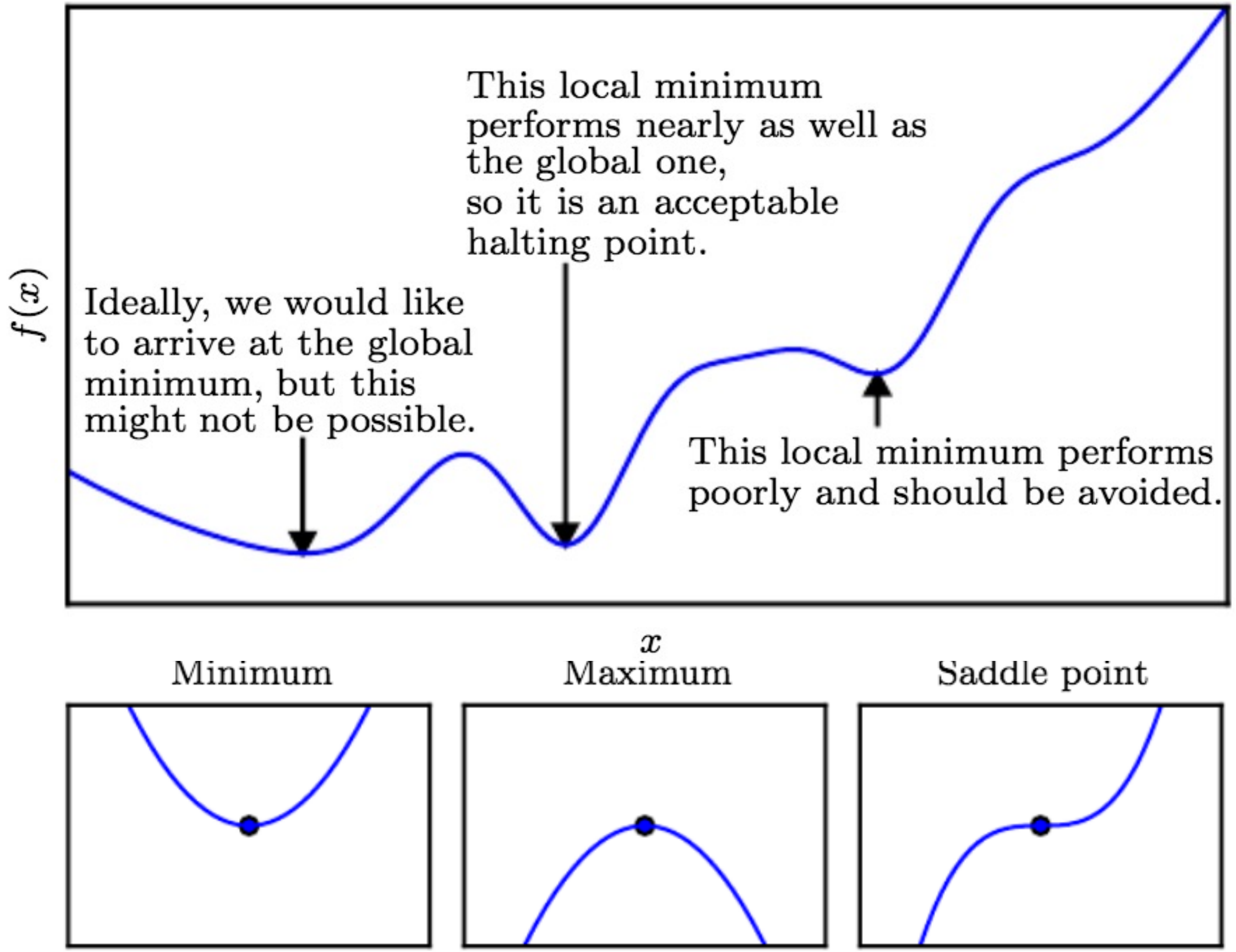
- For non-convex functions, local minima can obscure the search for global minima

- Analyzing critical points (plateaus) of function of interest is important



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 $df/dx = 0$
- 2^{nd} 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

deep imaging

Often we'll have functions of m variables

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

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

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)

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

We 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

deep imaging

Some quick details about gradient descent

Often we'll have functions of m variables

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

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

We 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

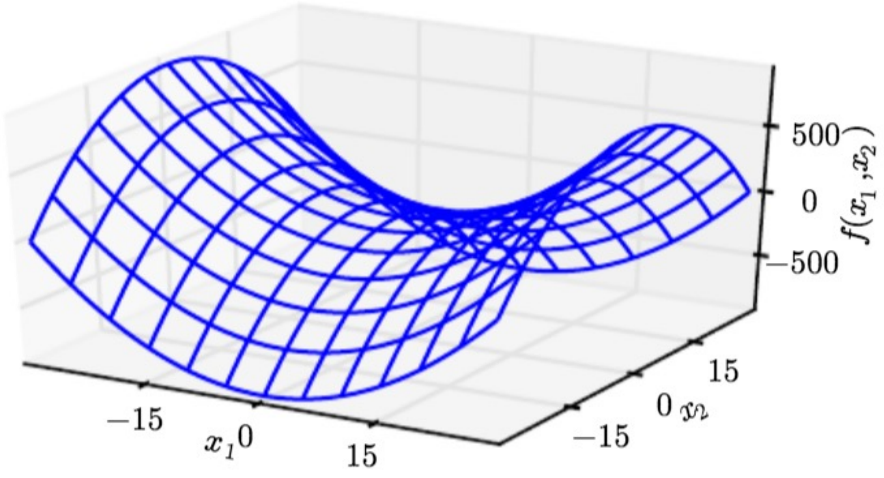
In general, we'll have functions that map m variables to n variables

$\boldsymbol{f} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ (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

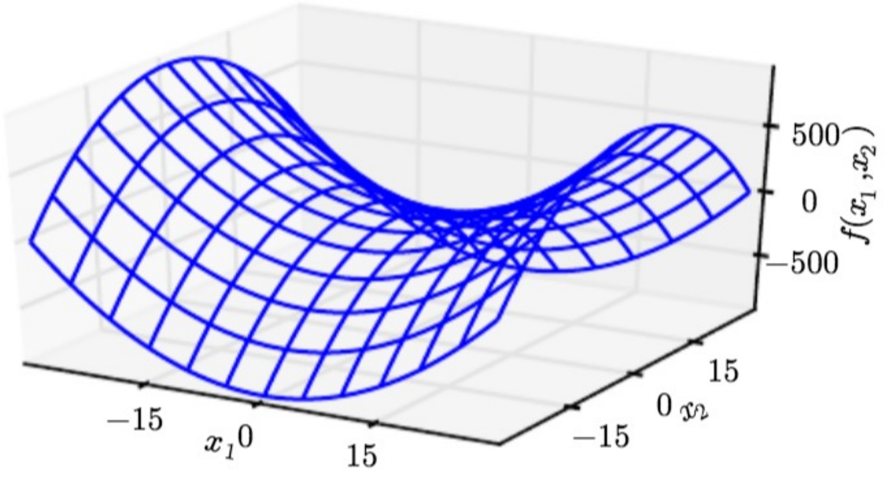
deep imaging

Quick example



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

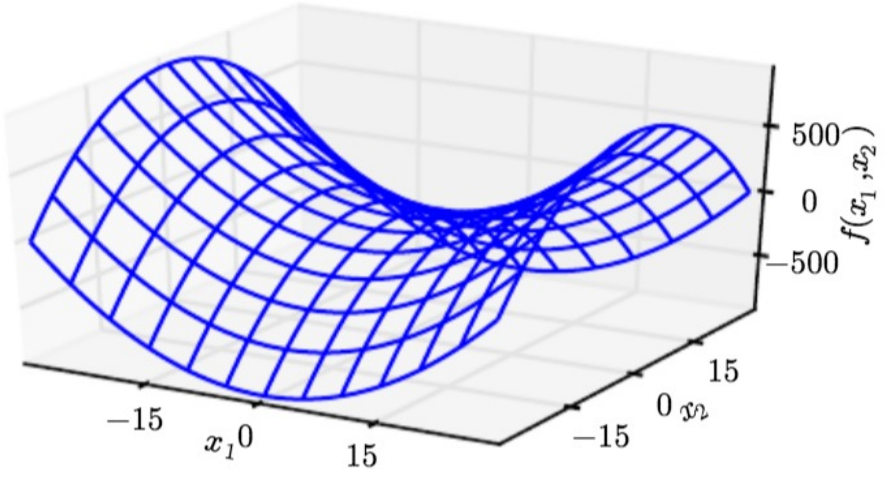
Quick example



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

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

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 $\mathbf{g}^{(0)} = \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)})$
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 $|\mathbf{x}^{(n+1)}-\mathbf{x}^{(n)}| < \text{threshold } t$

```
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x
    cur_x -= epsilon * df(prev_x)
    previous_step_size = abs(cur_x - prev_x)
    **Update epsilon - see next slide
    iters+=1
```

Steepest descent and the best step size ε

We computed this – computers can too in interesting ways

deep imaging

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

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

1. Evaluate function $f(\mathbf{x}^{(0)})$ at an initial guess point, $\mathbf{x}^{(0)}$
2. Compute gradient $\mathbf{g}^{(0)} = \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)})$
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 $|\mathbf{x}^{(n+1)}-\mathbf{x}^{(n)}| < \text{threshold } t$

```
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x
    cur_x -= epsilon * df(prev_x)
    previous_step_size = abs(cur_x - prev_x)
    **Update epsilon - see next slide
    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

deep imaging

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): $\boxed{\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}.}$

J. R. Shewchuck, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain"

deep imaging

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

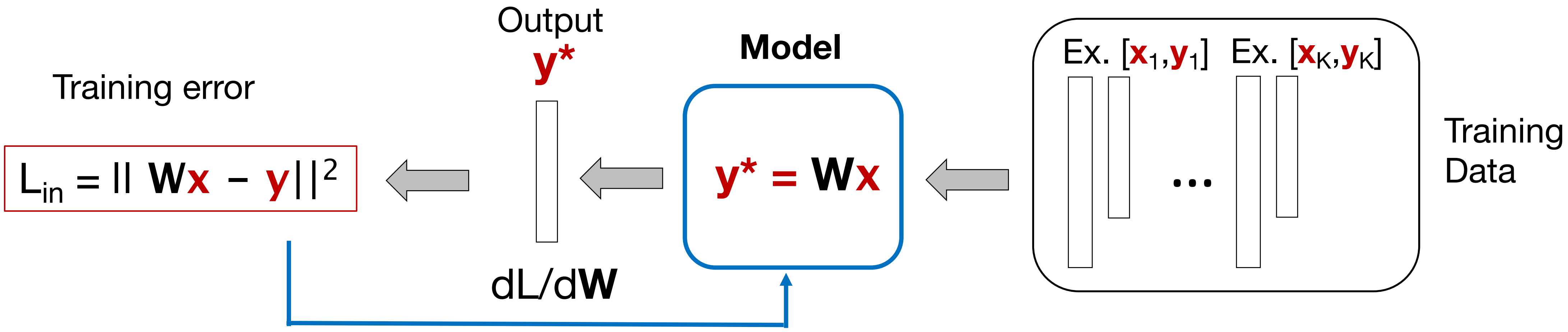
(easier)

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

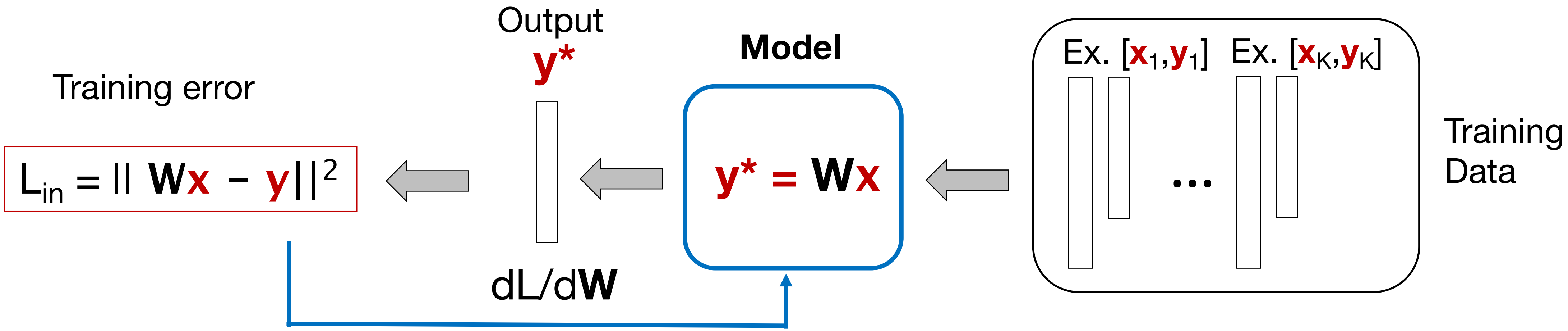
(harder)

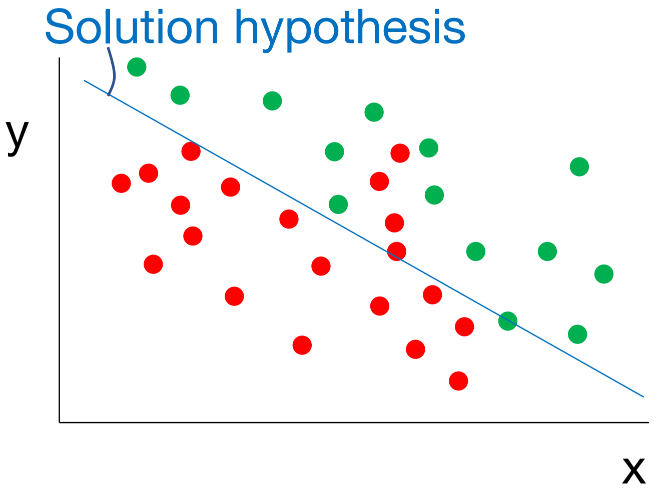
Next : We'll understand why linear regression doesn't work so well, and extend things beyond this simple starting point

The linear classification model – what's not to like?



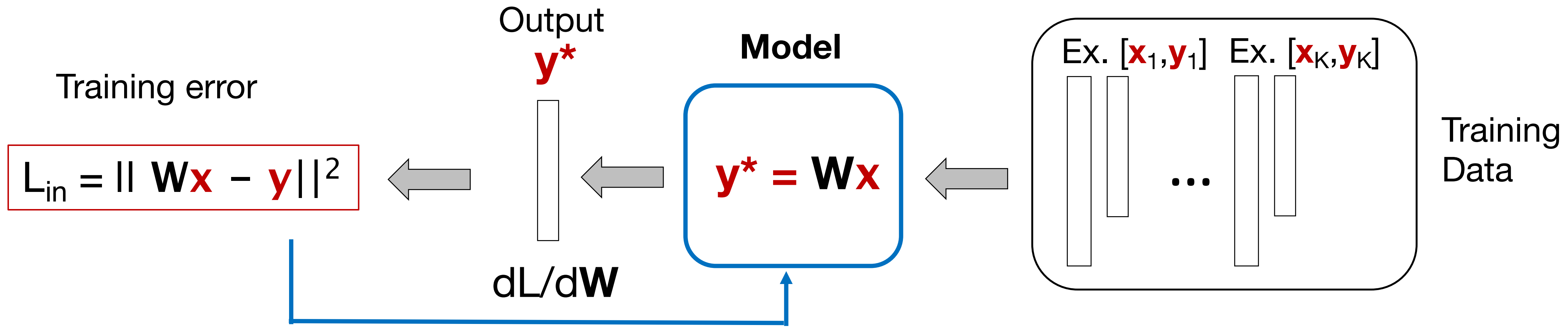
The linear classification model – what's not to like?

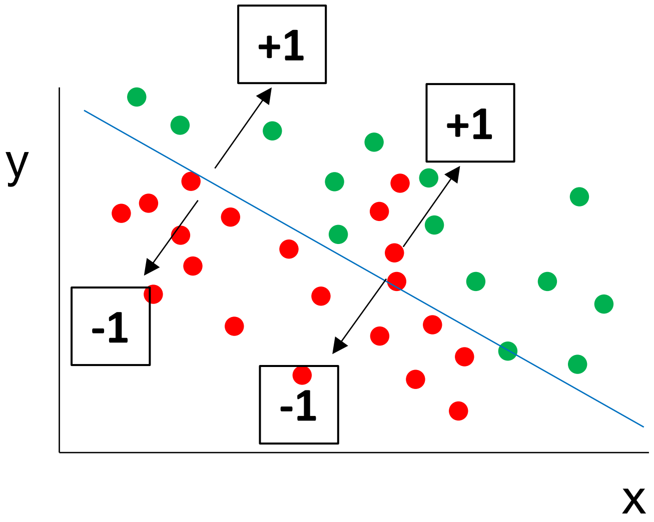




1. Can only separate data with lines (hyper-planes)…

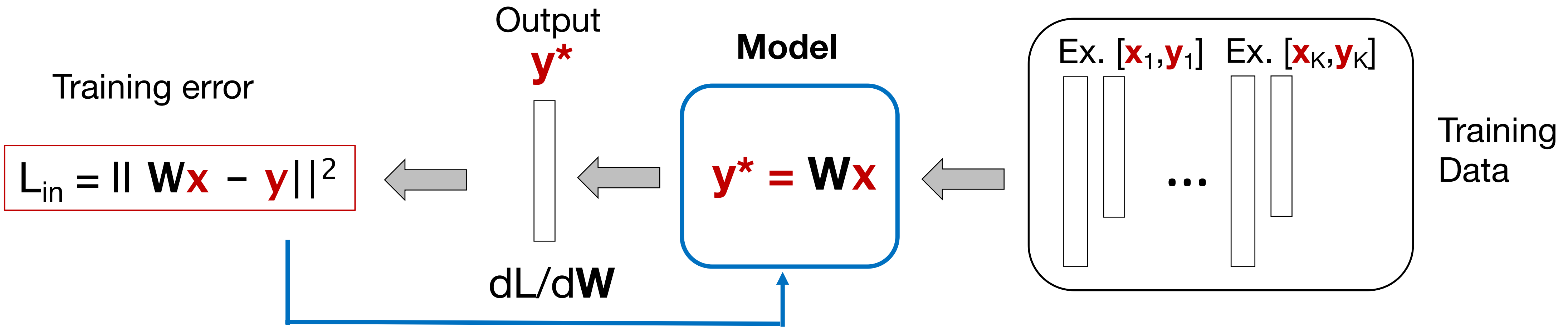
The linear classification model – what's not to like?

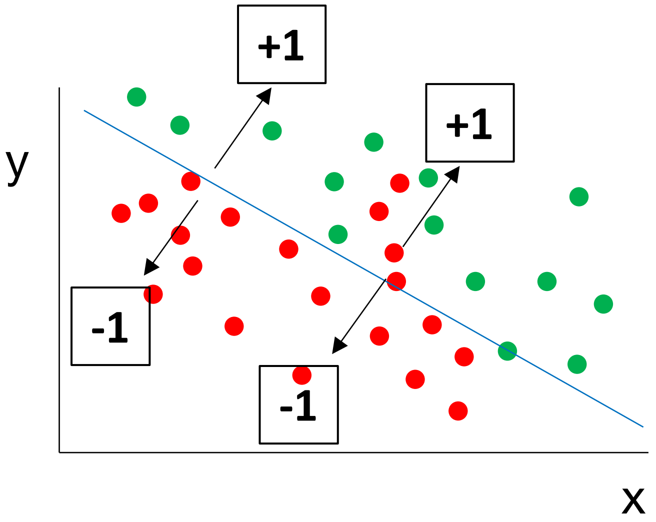




1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)

The linear classification model – what's not to like?

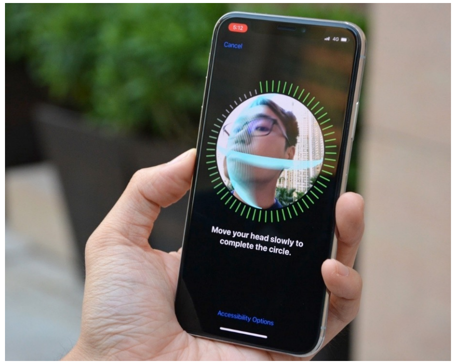




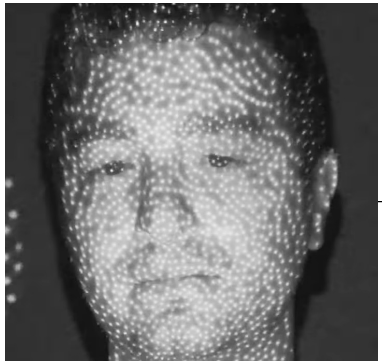
1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data

Cost functions matter: a simple example

deep imaging



$\color{red}{x}$



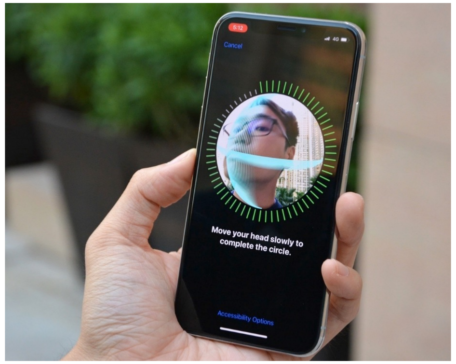
$f(\color{red}{x}, W)$

+1 = You

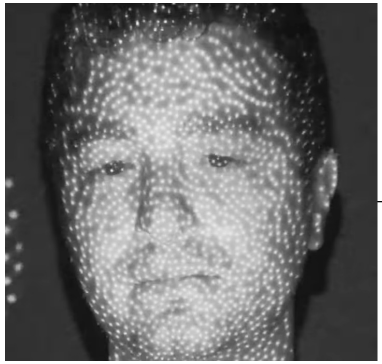
-1 = Bad guy

https://www.cnet.com/how-to/apple-face-id-everything-you-need-to-know/

Cost functions matter: a simple example



x



$f(x, W)$ → +1 = You / -1 = Bad guy

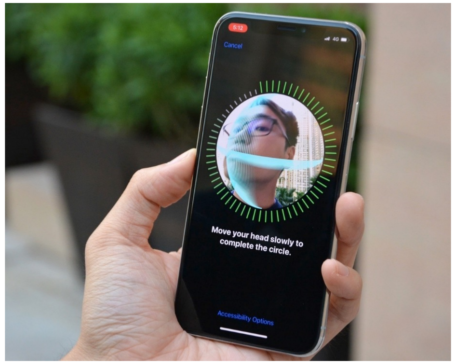
Two types of error: false accept and false reject

$f(x, W)$

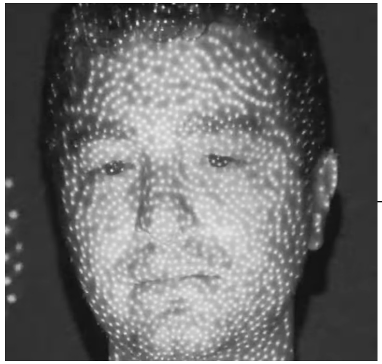
		+1	-1
y	+1	No Error (you/you)	False reject
	-1	False accept	No Error (bad guy/ bad guy)

https://www.cnet.com/how-to/apple-face-id-everything-you-need-to-know/

Cost functions matter: a simple example



x



$\rightarrow$ f(x, W) $\rightarrow$ { +1 = You ; -1 = Bad guy }

<u>Two types of error</u>: false accept and false reject

On a standard phone, what's a good cost function?

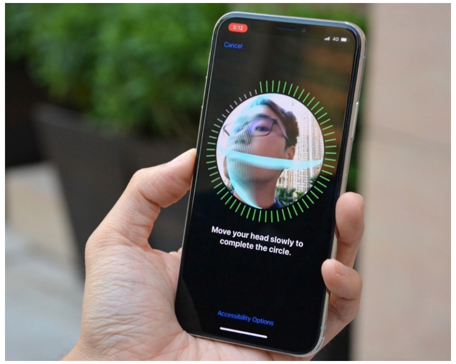
f(x, W)

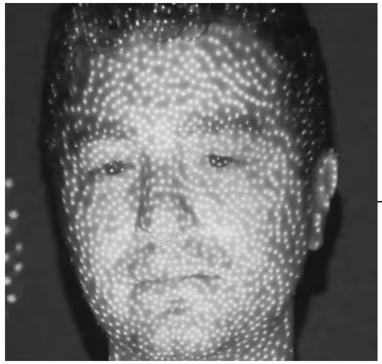
y	+1	-1
+1	No Error	False reject
-1	False accept	No Error

It's you, but you can't get in…

Letting an intruder in

Cost functions matter: a simple example





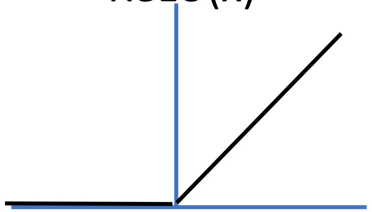
x

f(x, W) → +1 = You ; -1 = Bad guy

Two types of error: false accept and false reject

On a standard phone, what's a good cost function?

ReLU(x) = 0, x < 0
= x, x >= 0



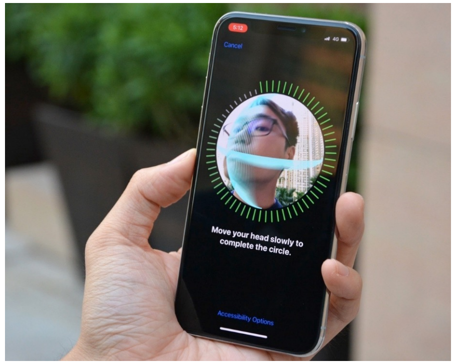
f(x, W)

y \ f(x, W)	+1	-1
+1	No Error	False reject
-1	False accept	No Error

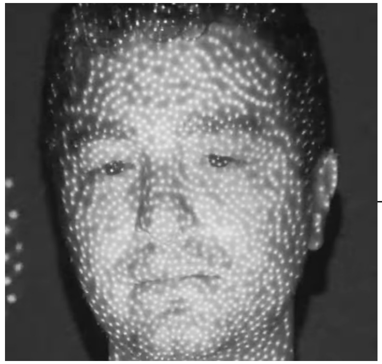
It's you, but you can't get in…

Letting an intruder in

Cost functions matter: a simple example



x



$f(x, W) \rightarrow$ +1 = You / -1 = Bad guy

<u>Two types of error</u>: false accept and false reject

On a standard phone, what's a good cost function?

$$L_{in} = \mathrm{ReLU}[f(x, W)-y] + \mathbf{10}\ \mathrm{ReLU}[y-f(x, W)]$$

Penalty for intruder

Large penalty for annoyance…

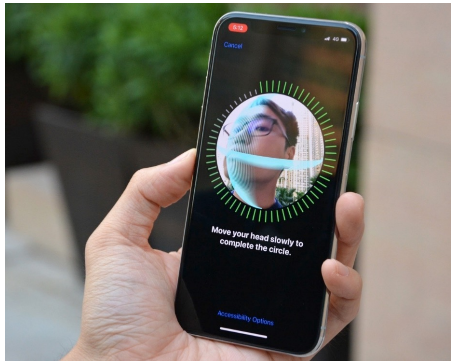
f(x, W)

y \ f(x, W)	+1	-1
+1	No Error	False reject
-1	False accept	No Error

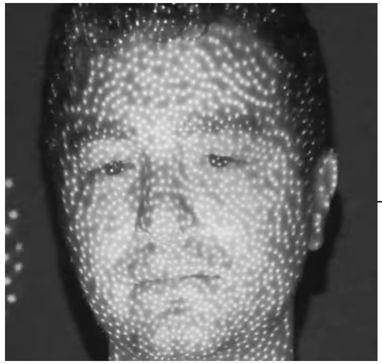
It's you, but you can't get in…

Letting an intruder in

Cost functions matter: a simple example



x



f(x, W)

+1 = You

-1 = Bad guy

What if you're a CIA agent?

f(x, W)

		+1	-1
y	+1	No Error	False reject
	-1	False accept	No Error

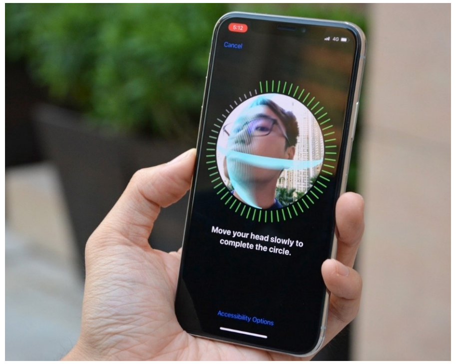
It's you, but you can't get in…

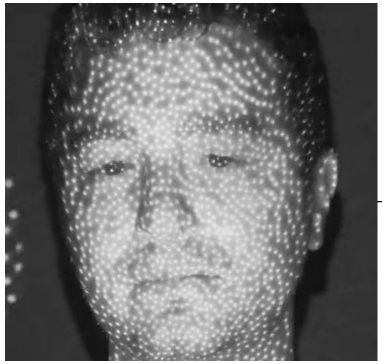
Letting an intruder in

Cost functions matter: a simple example

deep imaging

x





$f(x, W)$ → +1 = You / -1 = Bad guy

What if you're a CIA agent?

$$L_{in} = \mathbf{100{,}000}\ \mathrm{ReLU}[f(x, W)-y] + \mathrm{ReLU}[y-f(x, W)]$$

BIG penalty for intruder

Don't mind about annoyance...

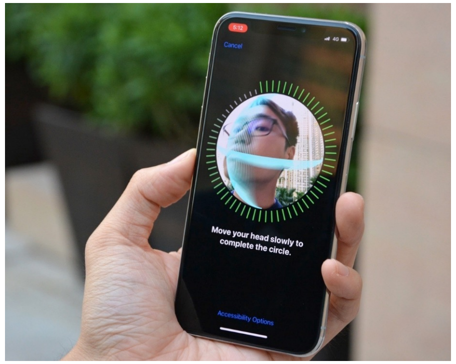
f(x, W)

y \ f(x, W)	+1	-1
+1	No Error	False reject
-1	False accept	No Error

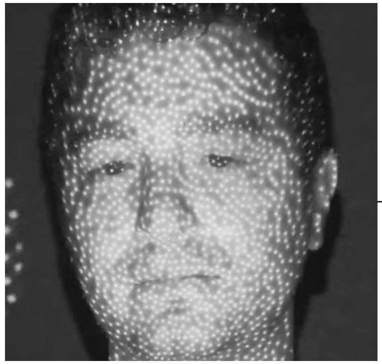
It's you, but you can't get in...

Letting an intruder in

Cost functions matter: a simple example



x



f(x, W)

+1 = You

-1 = Bad guy

Establishing cost function tied to conditional probabilities:

P(y = -1 | f(x,W) = +1)

P(y = +1 | f(x,W) = -1)

Establish L, W to balance and minimize these probabilities

f(x, W)

y	+1	-1
+1	No Error	False reject
-1	False accept	No Error

It's you, but you can't get in…

Letting an intruder in