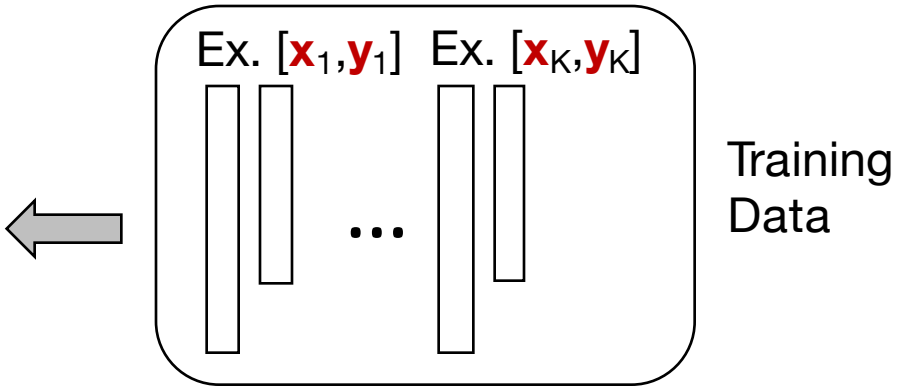
Machine Learning in Imaging

BME 590L
Roarke Horstmeyer

Lecture 6: Beyond linear classification and model complexity

Summary of machine learning pipeline:

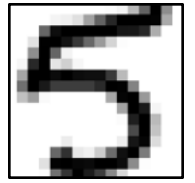
1. Network Training



E.g., images of 1's and 5's with labels:

What we need for network training:

1. Labeled examples

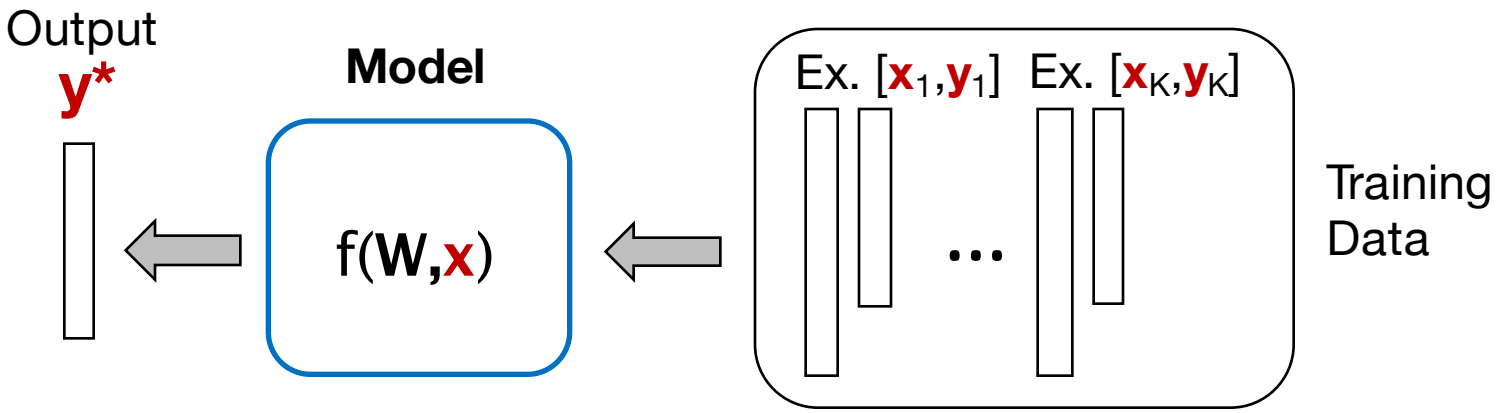
x_1 =  y_1 = +1

x_2 =  y_2 = +1

⋮ ⋮

Summary of machine learning pipeline:

1. Network Training

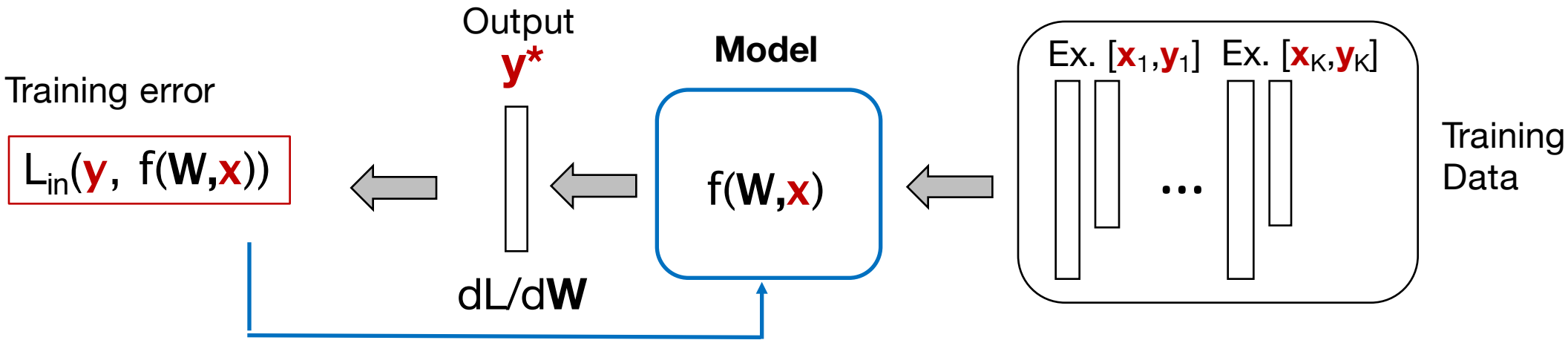


What we need for network training:

1. **Labeled examples**
2. **A model and loss function**

Summary of machine learning pipeline:

1. Network Training

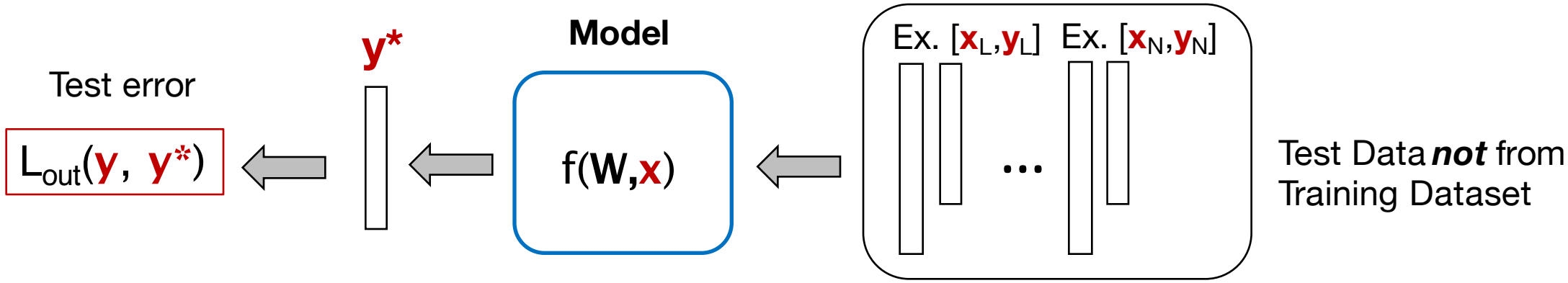


What we need for network training:

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

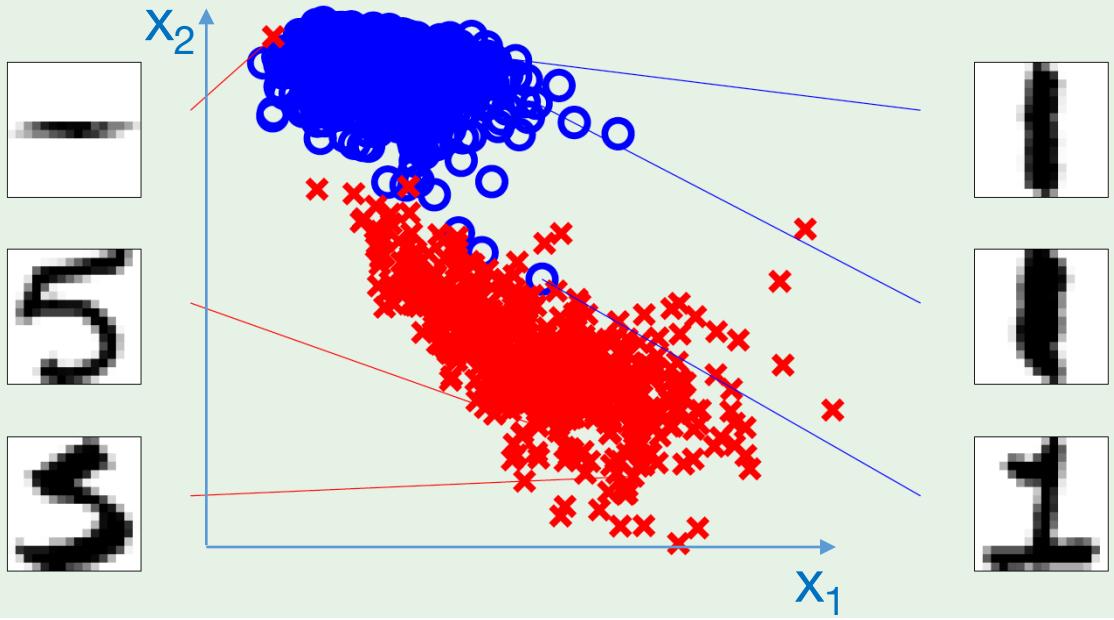


What we need for network testing:

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

Illustration of features

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



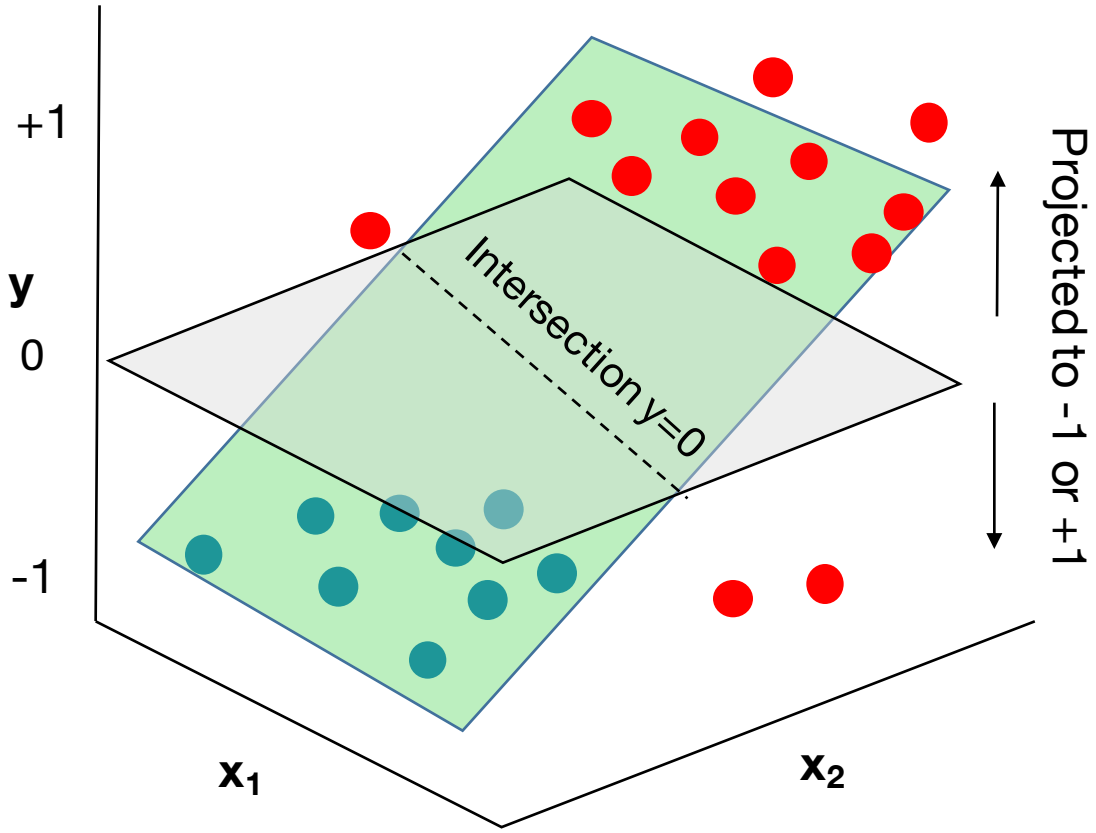
Linear classification:

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?



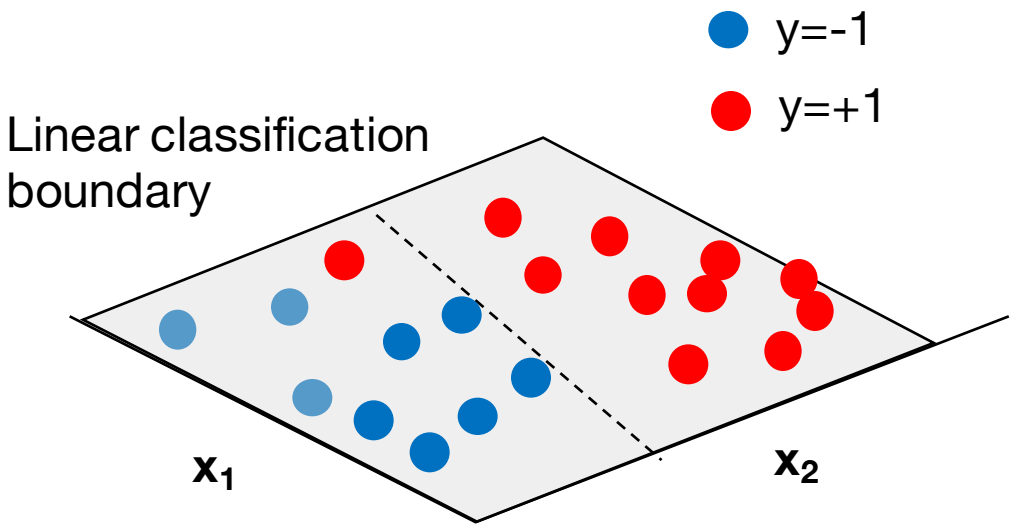
With sgn() operation:

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

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

- 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:

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

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

Closed-form solution available!

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

In the rest of this class: solve via gradient descent

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

Steepest descent and the best step size ε

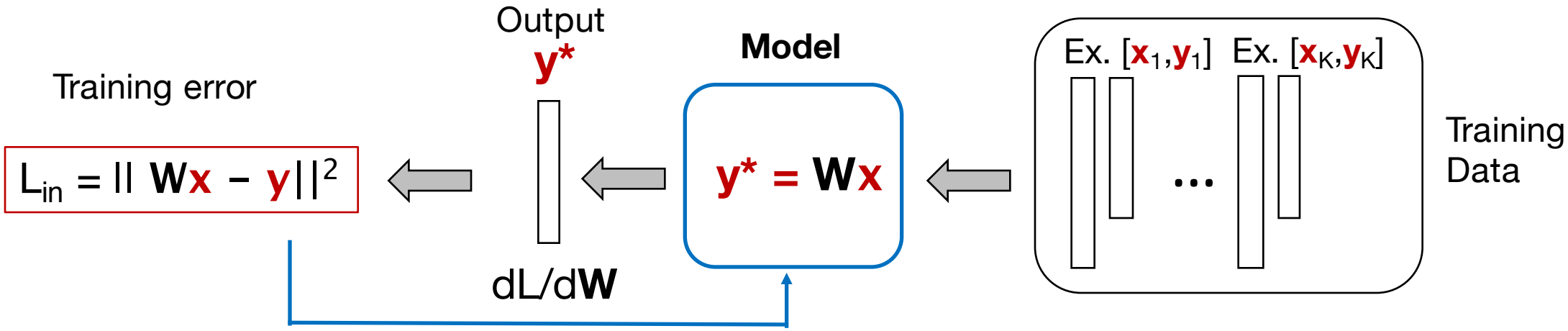
1. Evaluate function f($\mathbf{x}^{(0)}$) at an initial guess point, $\mathbf{x}^{(0)}$
2. Compute gradient $\mathbf{g}^{(0)} = \nabla_x f(\mathbf{x}^{(0)})$
3. Next point $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \boldsymbol{\varepsilon}^{(0)}\mathbf{g}^{(0)}$
4. Repeat – $\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)}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

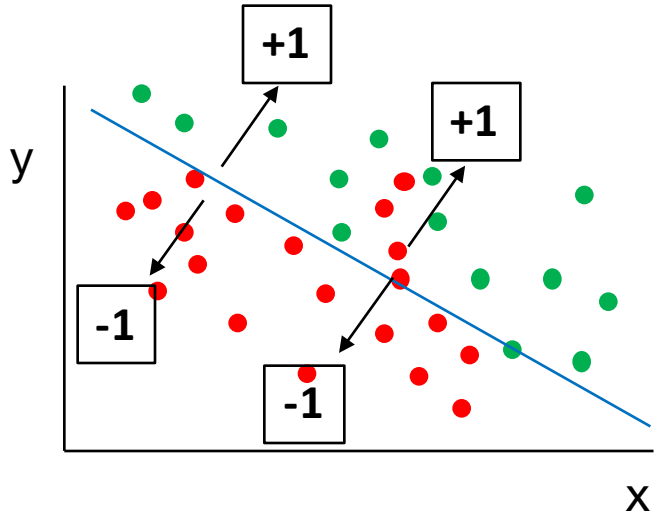
$$L = \frac{1}{N}\sum_{i=1}^{N}(w^T x_i - y_i)^2$$

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

```
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
```

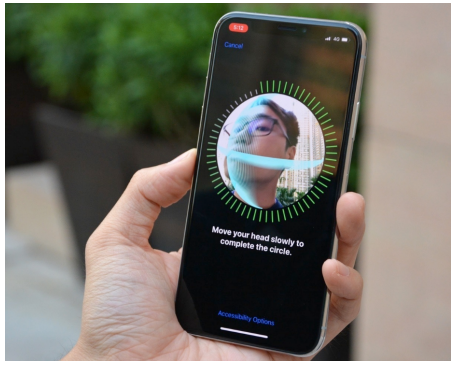
The linear classification model – what's not to like?



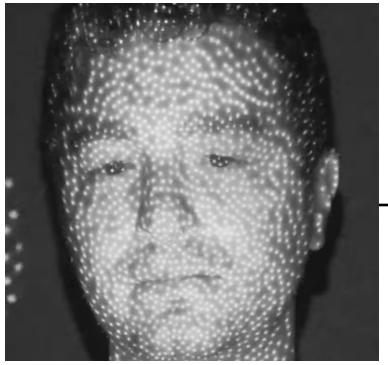


1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data

Cost functions matter: a simple example



x



f(x, W) → +1 = You

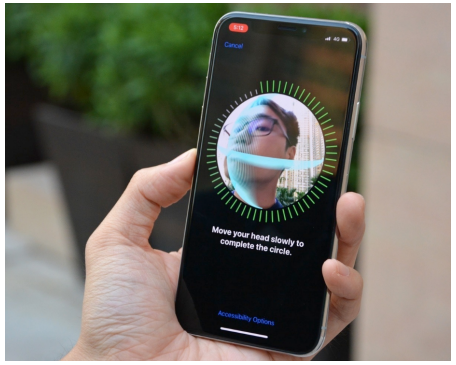
-1 = Bad guy

Two types of error: false accept and false reject

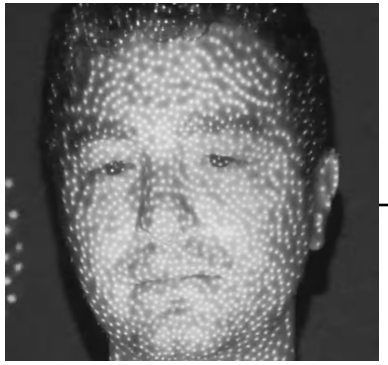
f(x, W)

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

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?

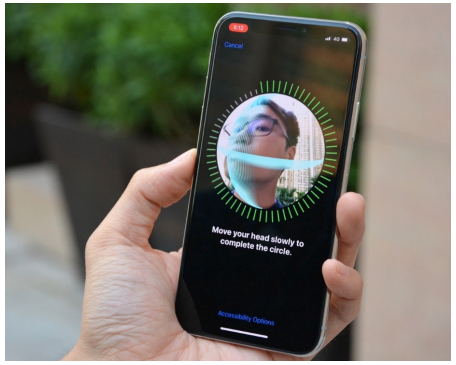
f(x, W)

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

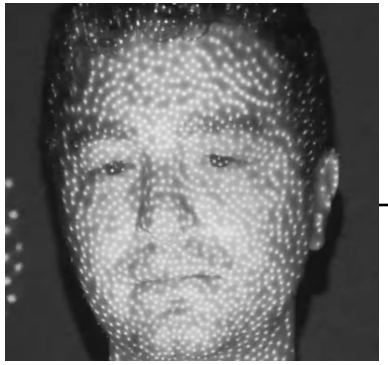
It's you, but you can't get in...

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?

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

Penalty for intruder

Large penalty for annoyance…

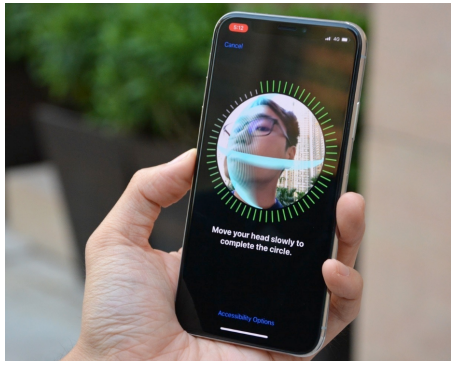
f(x, W)

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

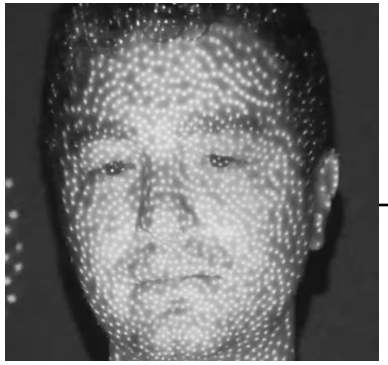
It's you, but you can't get in…

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?

L_{in} = **100,000** ReLU[f(x, W)-y] + ReLU[y-f(x, W)]

BIG penalty for intruder

Don't mind about annoyance…

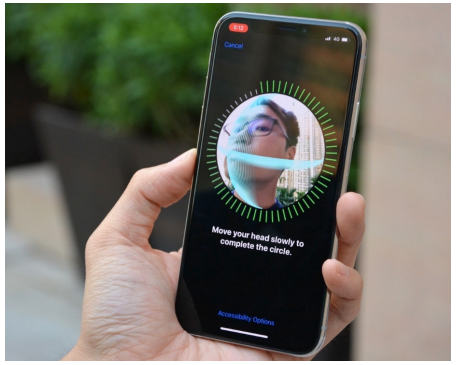
f(x, W)

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

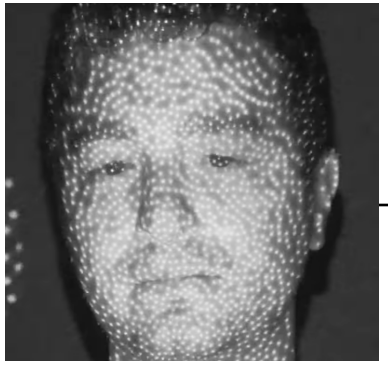
It's you, but you can't get in…

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:

$P(y = -1 \mid f(x,W) = +1)$

$P(y = +1 \mid f(x,W) = -1)$

Establish L, W to balance 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

Machine learning and probability

- Probability measures help determine when to use certain cost functions
- In previous case, we can measure probability of seeing a particular label, given model & data:

$$p(\mathbf{y}|\mathbf{w},\mathbf{x})$$

- In general, to find the best model, we'd like to infer **w**, having at hand our labeled data:

Maximum Likelihood Estimation $p(\mathbf{w}|\mathbf{x}_1,\ldots\mathbf{x}_N;\mathbf{y}_1,\ldots\mathbf{y}_N)$

- These two quantities are connected via Bayes' Theorem

$$p(\mathbf{w}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{x})}$$

With 2 conditioned variables: $p(\mathbf{w}|\mathbf{x},\mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{x},\mathbf{y})}$

$$p(\mathbf{w}|\mathbf{x},\mathbf{y}) \propto p(\mathbf{x},\mathbf{y}|\mathbf{w}) \propto p(\mathbf{y}|\mathbf{x},\mathbf{w})$$

Machine learning and probability

- Probability measures help determine when to use certain cost functions
- In previous case, we can measure probability of seeing a particular label, given model & data:

$$p(\mathbf{y}|\mathbf{w},\mathbf{x})$$

- In general, to find the best model, we'd like to infer **w**, having at hand our labeled data:

Maximum Likelihood Estimation $p(\mathbf{w}|\mathbf{x}_1,\ldots\mathbf{x}_N;\mathbf{y}_1,\ldots\mathbf{y}_N)$

- These two quantities are connected via Bayes' Theorem

$$p(\mathbf{w}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{x})}$$

With 2 conditioned variables:

$$p(\mathbf{w}|\mathbf{x},\mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{x},\mathbf{y})}$$

What you want: but hard to vary data to find model…

$$p(\mathbf{w}|\mathbf{x},\mathbf{y}) \propto p(\mathbf{x},\mathbf{y}|\mathbf{w}) \propto p(\mathbf{y}|\mathbf{x},\mathbf{w})$$

What you can do: test the model, check the result

Linear classification is the maximum likelihood for Gaussian data

- Given a close relationship between $p(\mathbf{w}|\mathbf{x}, \mathbf{y}) \longleftrightarrow p(\mathbf{y}|\mathbf{x}, \mathbf{w})$:

  Maximum likelihood estimation asks the question,

$$\text{For what } \mathbf{w} \text{ is } p(\mathbf{y}_1, \ldots \mathbf{y}_N | \mathbf{x}_1, \ldots \mathbf{x}_N; \mathbf{w}) \text{ maximized?}$$

$$\text{For what } \mathbf{w} \text{ is } \prod_{i=1}^{N} p(\mathbf{y}_i, |\mathbf{x}_i, \mathbf{w}) \text{ maximized?}$$

- Let's assume our labels are a "noisy" Gaussian process that surround the correct label:

$$y_i = \mathbf{w}^T \mathbf{x}_i + n \qquad (n \text{ is zero-mean Gaussian noise})$$

- Then, the above cond. prob. for labels can be expressed as a multivariate Gaussian

Linear classification is the maximum likelihood for Gaussian data

For what $\mathbf{w}$ is $\prod_{i=1}^{N} p(\mathbf{y}_i, |\mathbf{x}_i, \mathbf{w})$ maximized?

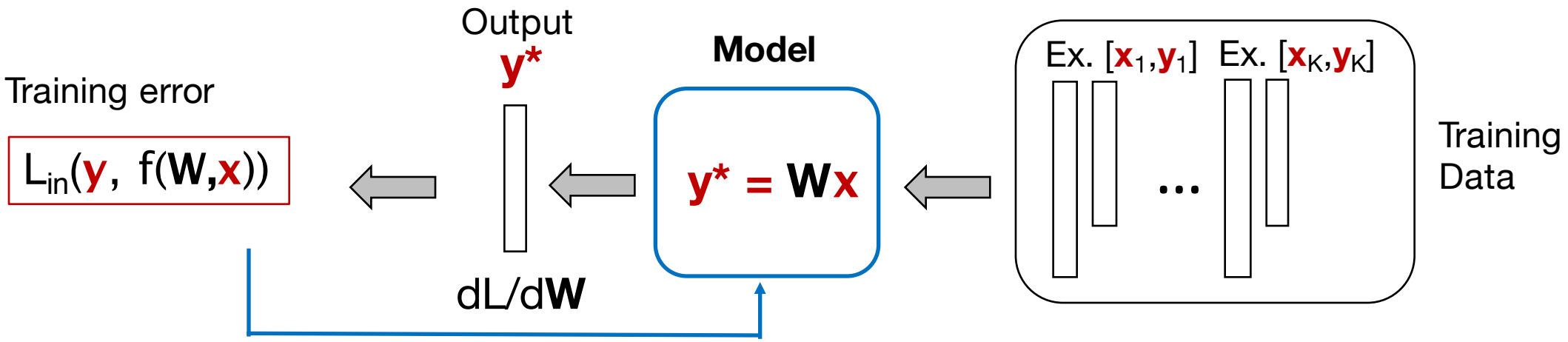
For what $\mathbf{w}$ is $\prod_{i=1}^{N} \exp\left(\frac{-(\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right)$ maximized?

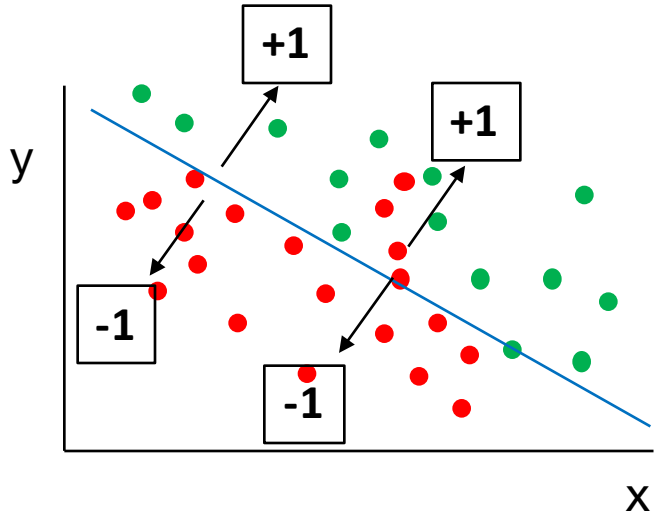
For what $\mathbf{w}$ is $\sum_{i=1}^{N} \frac{-(\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}$ maximized?

For what $\mathbf{w}$ is $\sum_{i=1}^{N} (\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2$ minimized?

Summary: Linear classification assumes labels are a Gaussian random process, and then thresholds them to ether -1 or +1

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



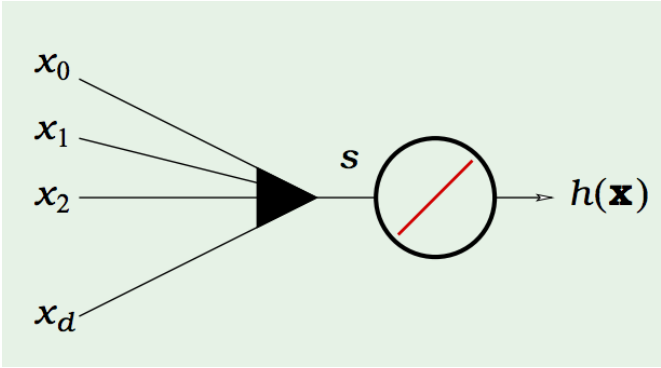


1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data

Let's think about the labels as a probabilistic measure:

Linear regression:

$$h(x) = s \in (-\infty, \infty)$$

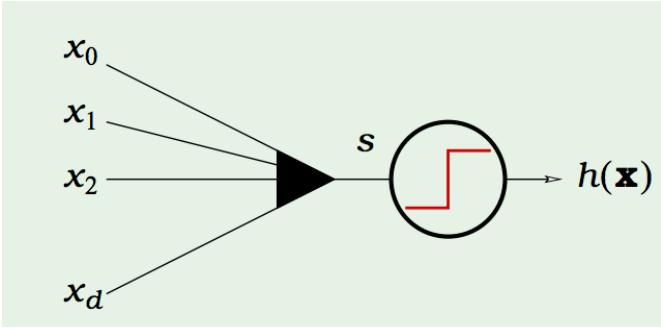


Not a probabilistic mapping

Linear classifier:

$$h(x) = \text{sign}(w_o^T x_j)$$

$$h(x) = \text{sign}(s) \in \{0, 1\}$$

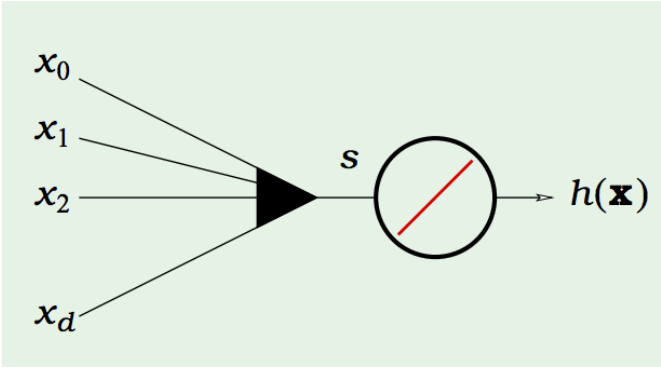


Probabilistic, but all-or-nothing

Let's think about the labels as a probabilistic measure:

Linear regression:

$$h(x) = s \in (-\infty, \infty)$$

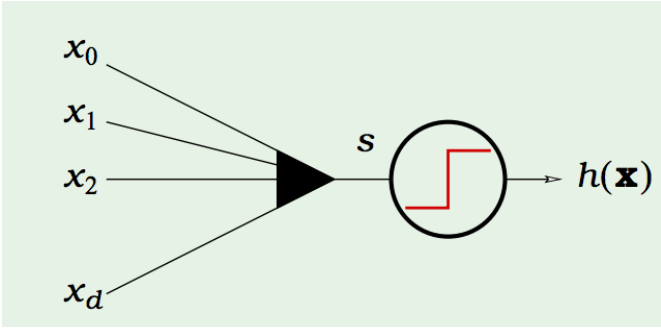


Not a probabilistic mapping

Linear classifier:

$$h(x) = \text{sign}(w_o^T x_j)$$

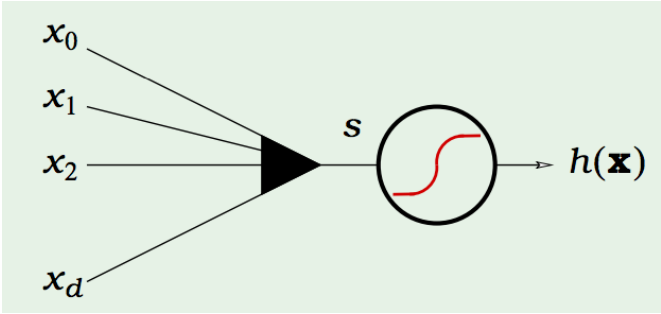
$$h(x) = \text{sign}(s) \in \{0, 1\}$$



Probabilistic, but all-or-nothing

Logistic classifier:

$$h(x) = \theta(s) \in [-1, 1]$$



Probabilistic

Probability interpretation

$h(\mathbf{x}) = \theta(s)$ is interpreted as a probability

Example. Prediction of heart attacks

Input $\mathbf{x}$: cholesterol level, age, weight, etc.

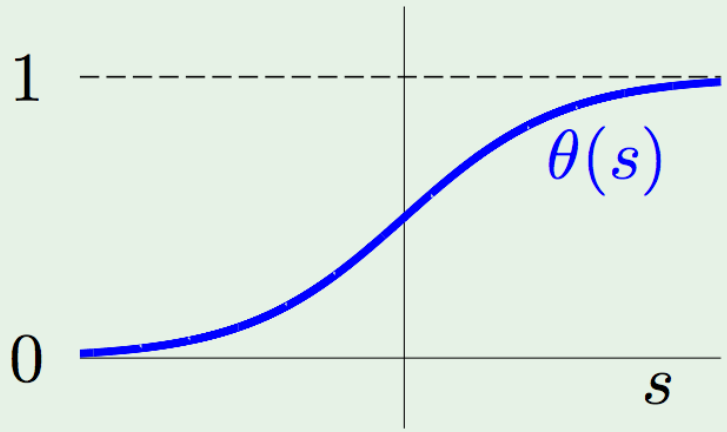
$\theta(s)$: probability of a heart attack

The signal $s = \mathbf{w}^{\mathsf{T}}\mathbf{x}$ "risk score"

The logistic function θ

The formula:

$$\theta(s) = \frac{e^s}{1 + e^s}$$



soft threshold: uncertainty

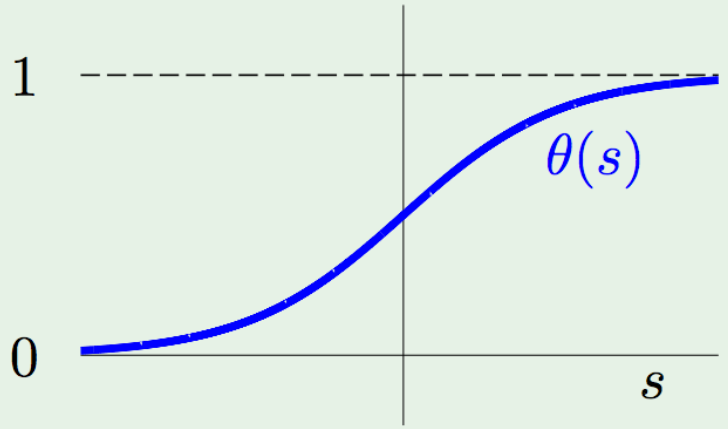
sigmoid: flattened out 's'

From linear classification to logistic classification

- Let's re-derive a cost function for the case where labels are treated as probabilities
  - You'll use derivatives of this more often than not in Tensorflow
- During learning, we will again have two classes (in this simple example), y= +/- 1
- map these binary values onto a [0,1] probability distribution, f(x):

Formula for likelihood

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

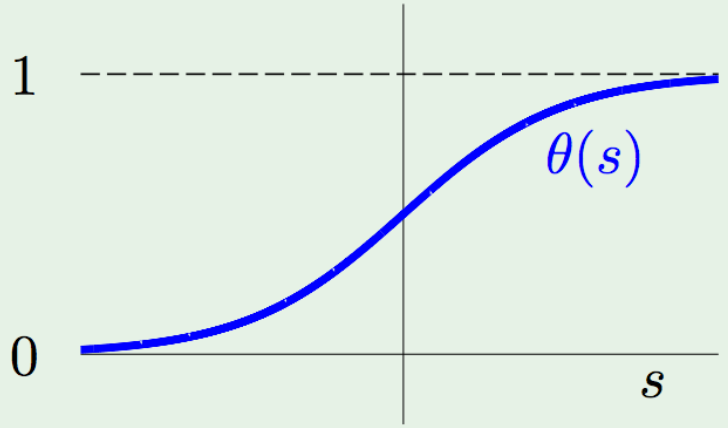


$h(\mathbf{x}) = \theta(s)$ is interpreted as a probability

Formula for likelihood

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

$h(\mathbf{x}) = \theta(s)$ is interpreted as a probability

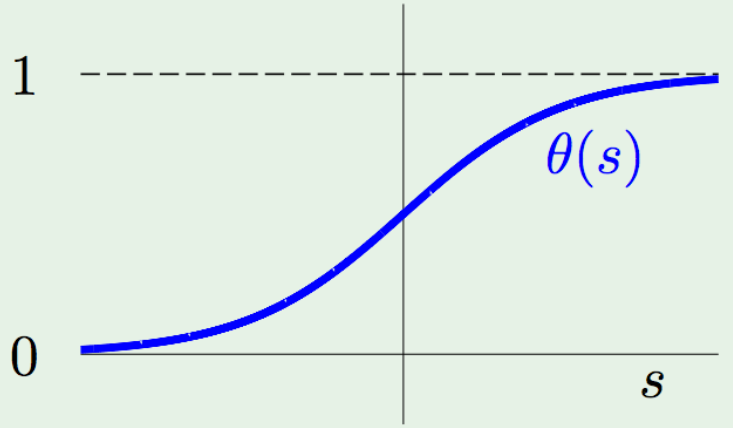


Rough Interpretation:

- If score x is high, then should map to y=+1 with high probability
- h(x) = θ(x) is large for large value of x
- If score x is small, then should map to y=-1 with high probability
- h(x) ~0 for small values of x, so 1- h(x) ~1 is high probability to y=-1 mapping

Formula for likelihood

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$



Substitute $h(\mathbf{x}) = \theta(\mathbf{w}^\mathsf{T}\mathbf{x})$, noting $\theta(-s) = 1 - \theta(s)$

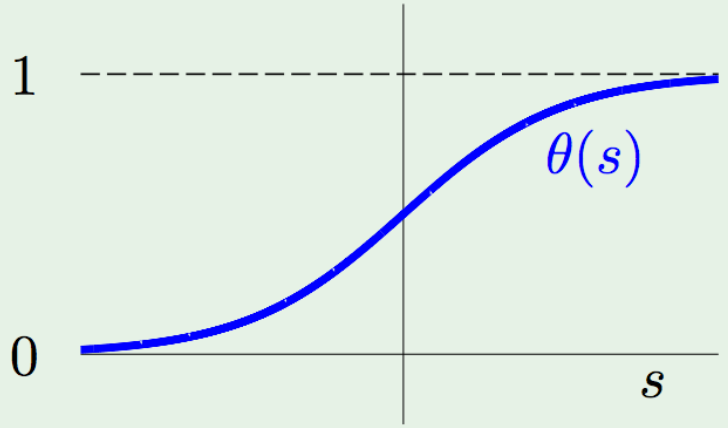
$$P(y \mid \mathbf{x}) = \theta(y\ \mathbf{w}^\mathsf{T}\mathbf{x})$$

Likelihood of $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ is

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^\mathsf{T}\mathbf{x}_n)$$

Formula for likelihood

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$



Substitute $h(\mathbf{x}) = \theta(\mathbf{w}^\mathsf{T}\mathbf{x})$, noting $\theta(-s) = 1 - \theta(s)$

This is the probability of the labels, given the data. We'd like to maximize this probability!

*Like the linear regression case, but now the probability of classes given the data is not Gaussian distributed, but instead follows the sigmoid curve (is bound to [0,1], which is more realistic)

$$P(y \mid \mathbf{x}) = \theta(y\ \mathbf{w}^\mathsf{T}\mathbf{x})$$

Likelihood of $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ is

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^\mathsf{T}\mathbf{x}_n)$$

Maximizing the likelihood

Minimize $$-\frac{1}{N}\ln\left(\prod_{n=1}^{N}\theta(y_n\ \mathbf{w}^{\mathsf{T}}\ \mathbf{x}_n)\right)$$

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

$$\left[\theta(s)=\frac{1}{1\ +e^{-s}}\right]$$

$$E_{\text{in}}(\mathbf{w})\ =\frac{1}{N}\sum_{n=1}^{N}\underbrace{\ln\left(1+e^{-y_n\mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)}_{\mathrm{e}\left(h(\mathbf{x}_n),y_n\right)}$$

"cross-entropy" error

How to minimize L_{in}

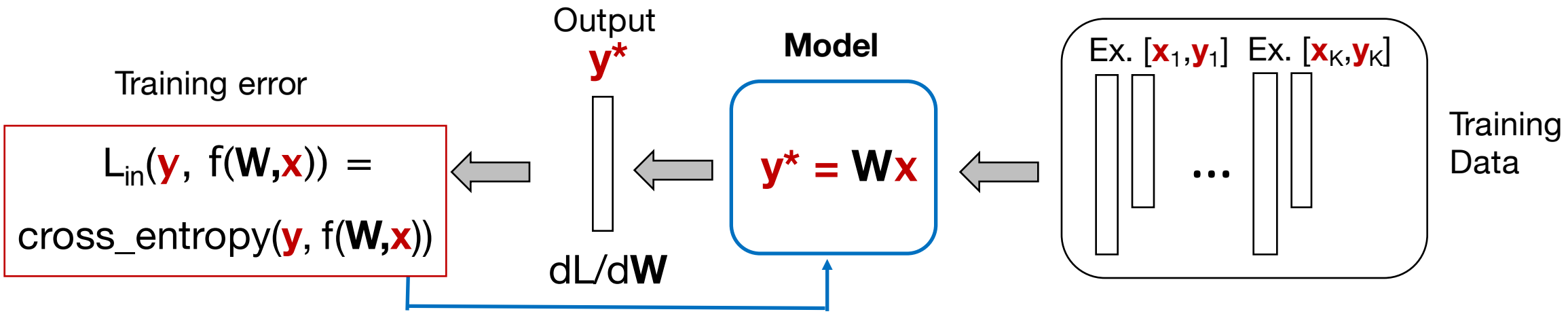
For logistic regression,

$$L_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right) \quad \longleftarrow \textbf{iterative} \text{ solution}$$

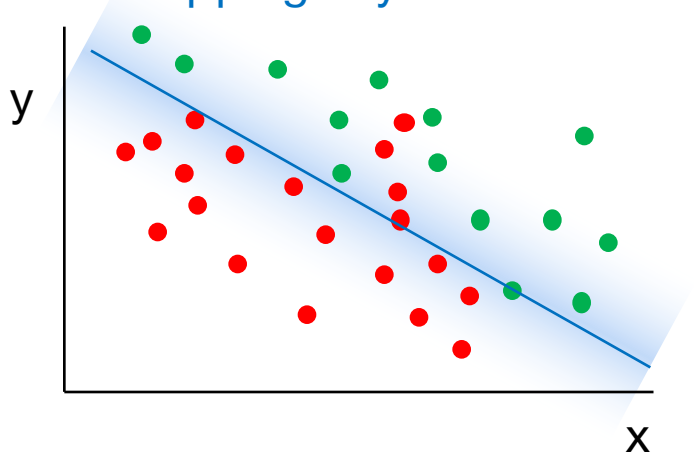
Compare to linear regression:

$$L_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n\right)^2 \quad \longleftarrow \text{closed-form solution}$$

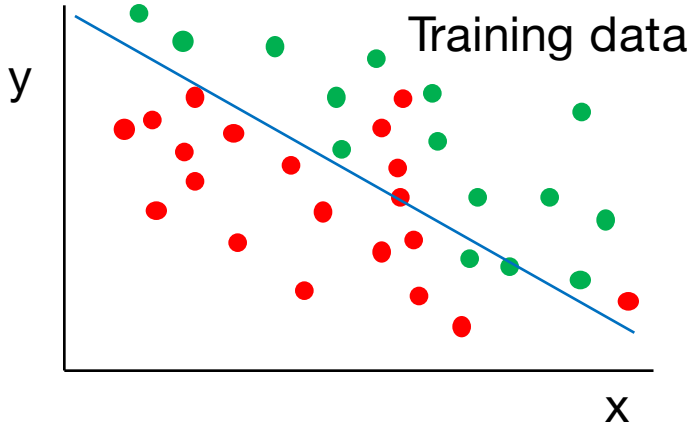
The linear classification model – what's not to like?



Probabilistic mapping to y

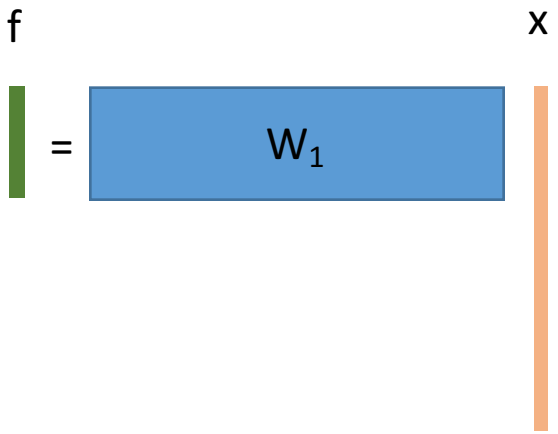


1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data



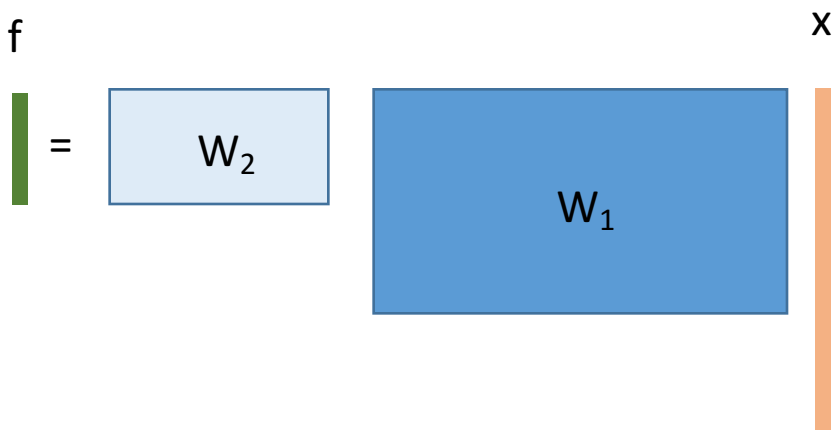
$$f = W_1 x$$

Learned *f:* not flexible



Can we add flexibility by multiplying with another weight matrix?

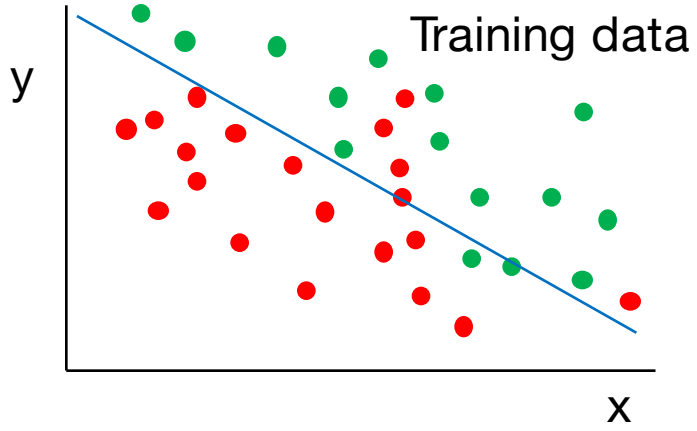
$$\begin{cases} f_1 = W_1 x + b_1 \\ f_2 = W_2 f_1 + b_2 \end{cases}$$



$$f_2 = W_2(W_1 x + b_1) + b_2$$

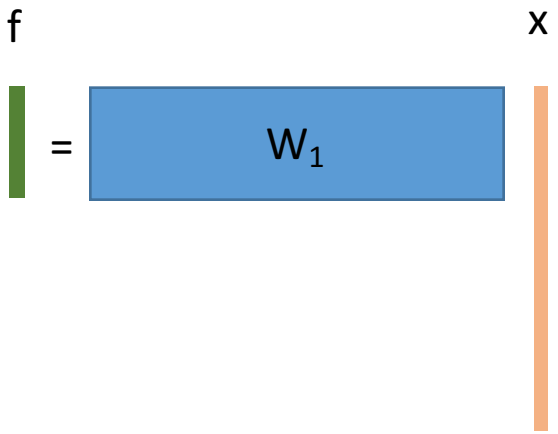
$$f_2 = W' x + b'$$

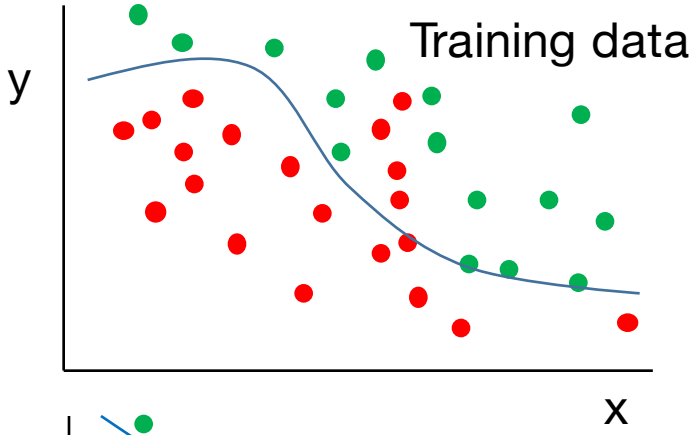
Unfortunately not…



$$f = W_1 x$$

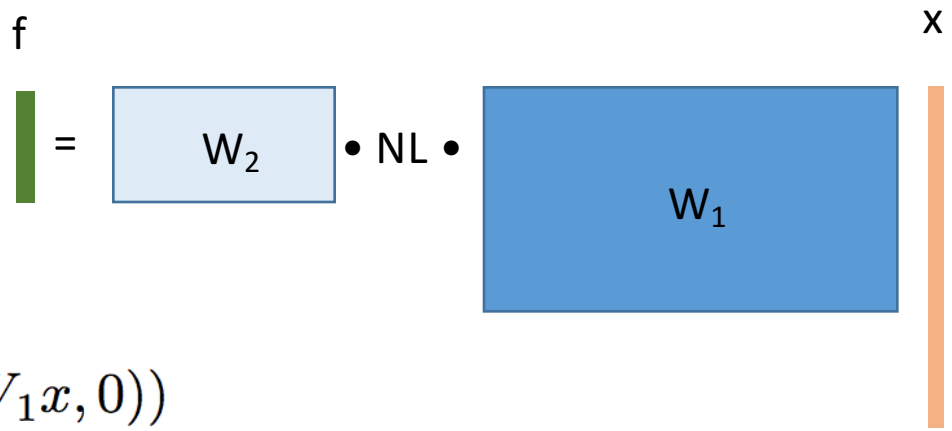
Learned *f:* not flexible

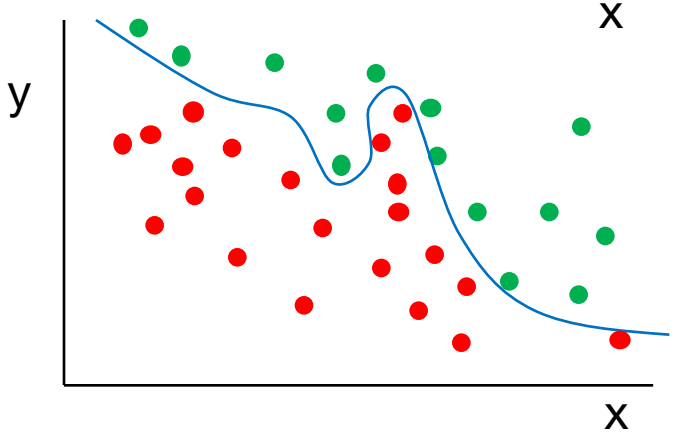




$$f = W_2 \max(W_1 x, 0)$$

Learned *f:* a bit flexible





$$f = W_3 \max(0, W_2 \max(W_1 x, 0))$$

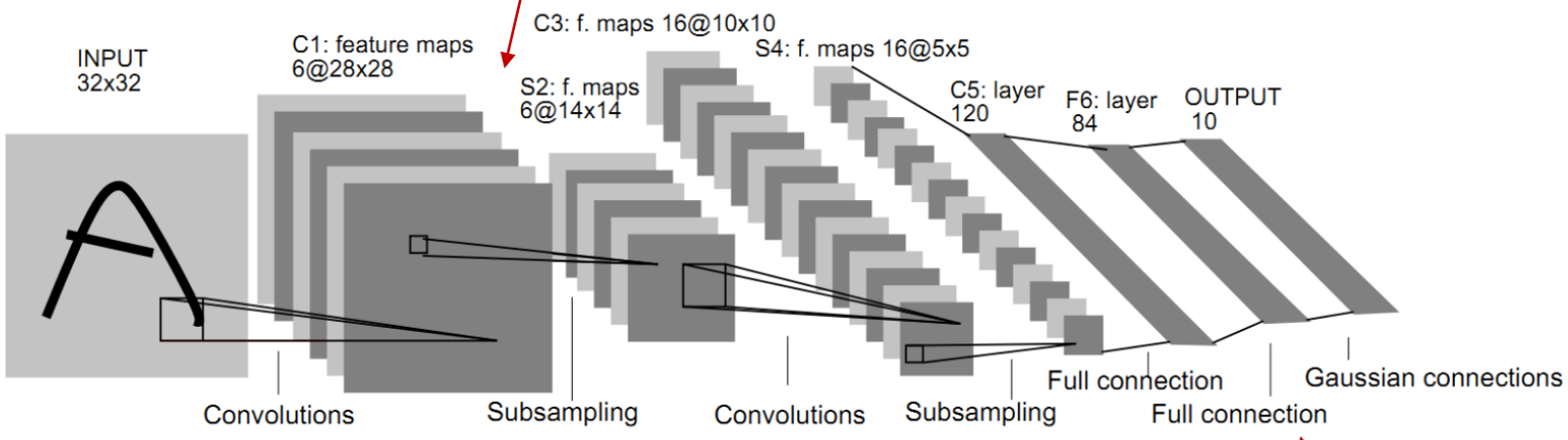
Learned *f:* more flexible

Does it generalize???

We can keep adding these “layers”…

Getting us to Convolutional Neural Networks

After, apply non-linearity and sub-sampling



Each matrix W is a convolution matrix

Repeat a few times

At the end, use a full W for a final matrix multiplication

Getting us to Convolutional Neural Networks