deep imaging

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer

Lecture 7: Gradient descent and going beyond linear classification

Summary of machine learning pipeline:

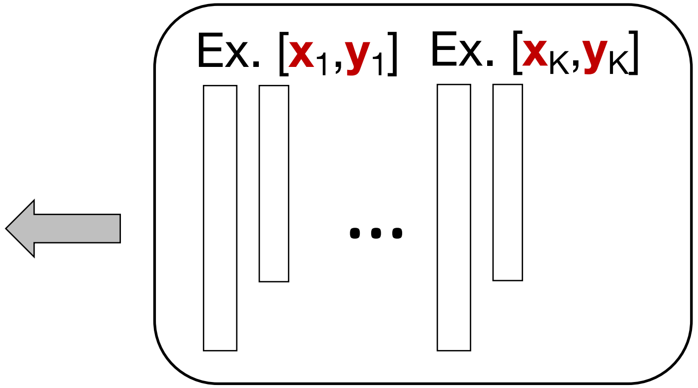
1. Network Training

What we need for network training:

1. Labeled examples

Summary of machine learning pipeline:

1. Network Training

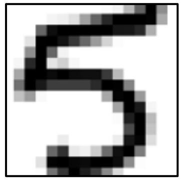


Training Data

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

What we need for network training:

1. Labeled examples

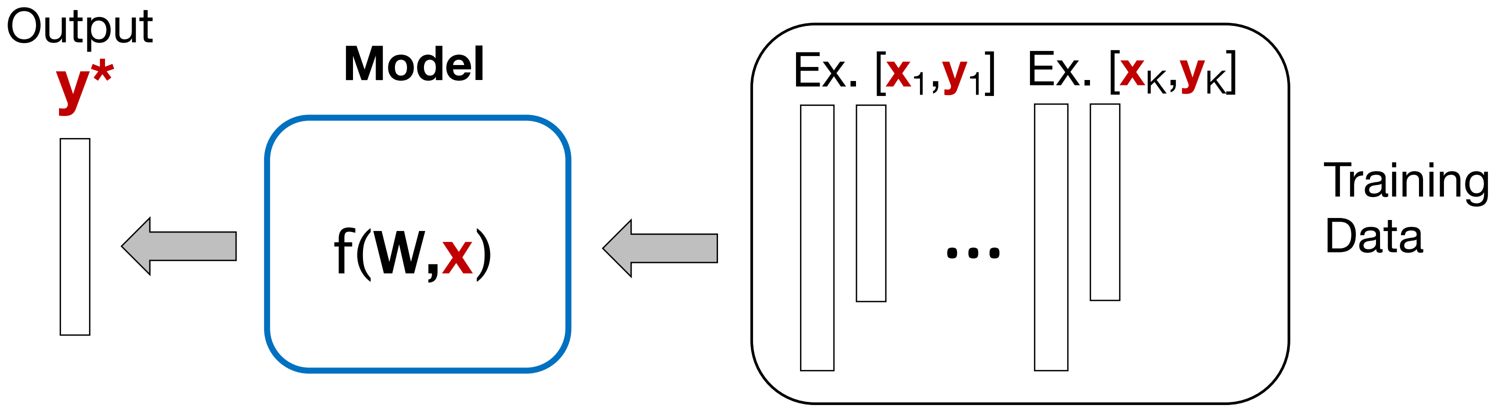
$\mathbf{x}_1$ =  $\mathbf{y}_1$ = +1

$\mathbf{x}_2$ =  $\mathbf{y}_2$ = -1

⋮ ⋮

Summary of machine learning pipeline:

1. Network Training



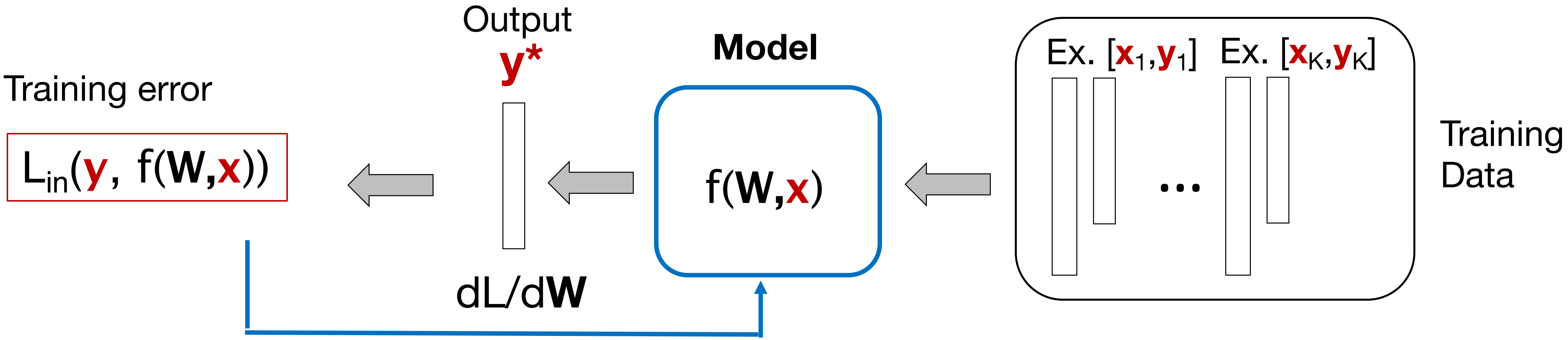
What we need for network training:

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**

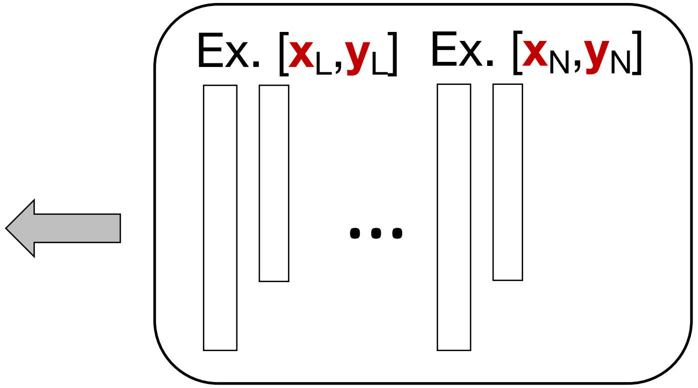
Summary of machine learning pipeline:

2. Network Testing

What we need for network testing:

Summary of machine learning pipeline:

2. Network Testing



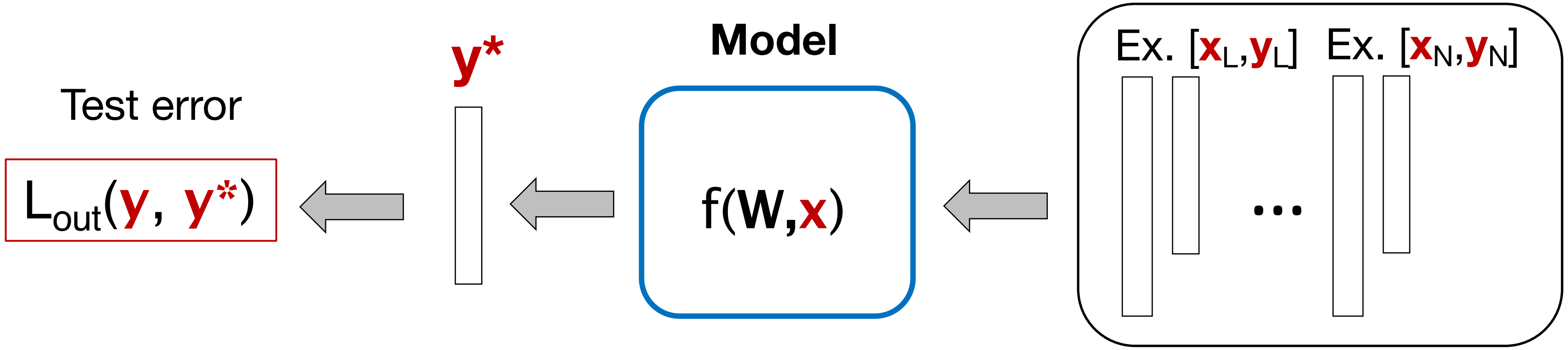
Test Data ***not*** from Training Dataset

What we need for network testing:

4. *Unique* labeled test data

Summary of machine learning pipeline:

2. Network Testing



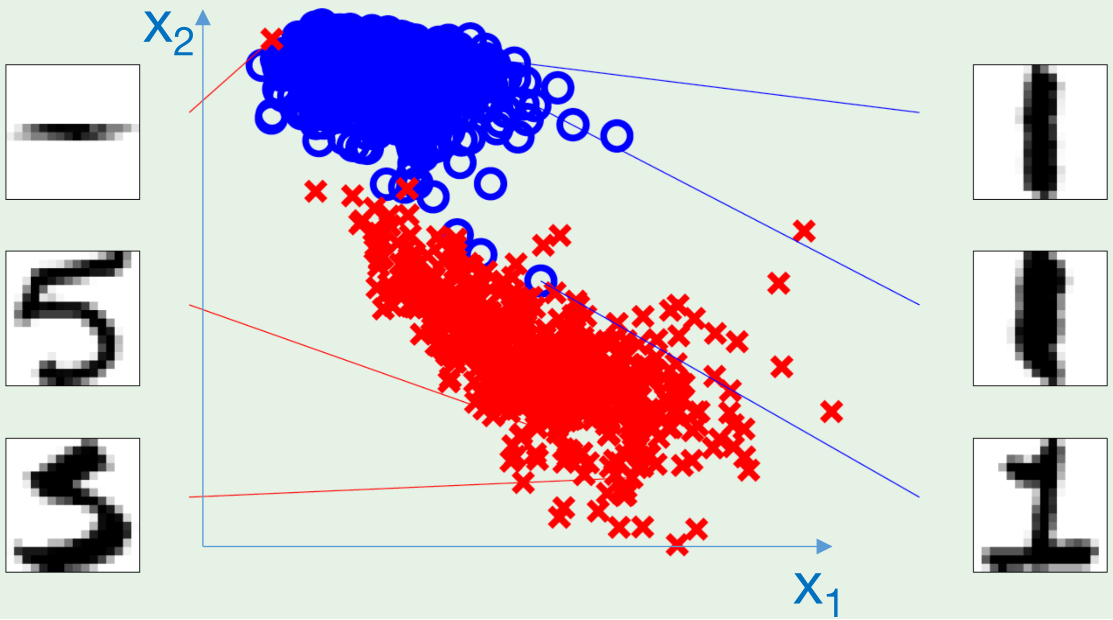
Test Data ***not*** from Training Dataset

What we need for network testing:

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

Illustration of features

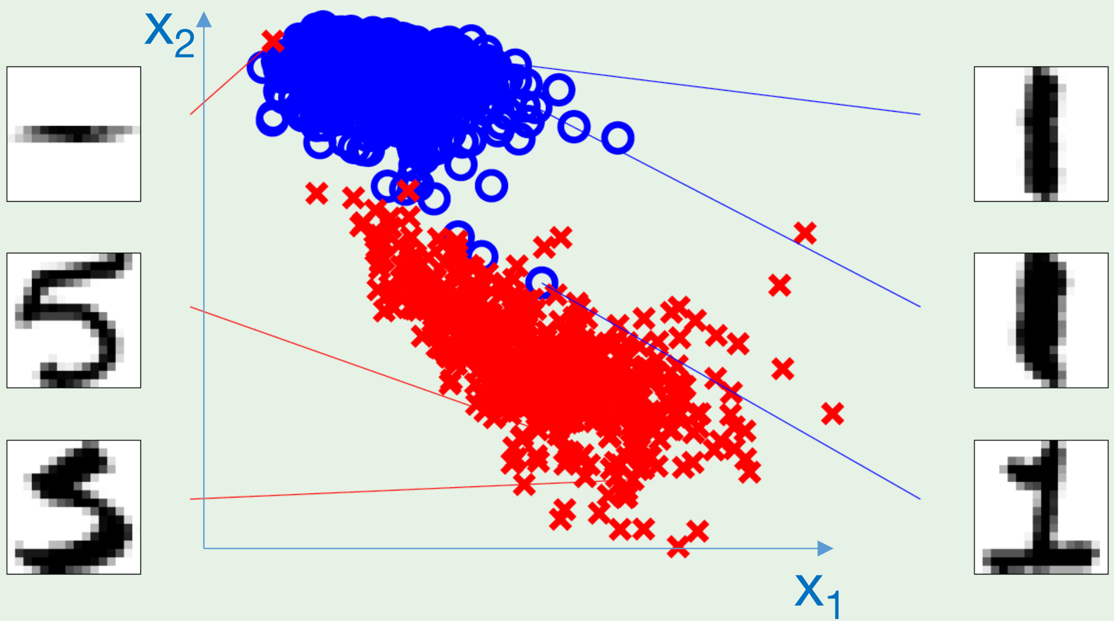
$\mathbf{x} = (x_0, x_1, x_2)$ x_1: intensity x_2: symmetry



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



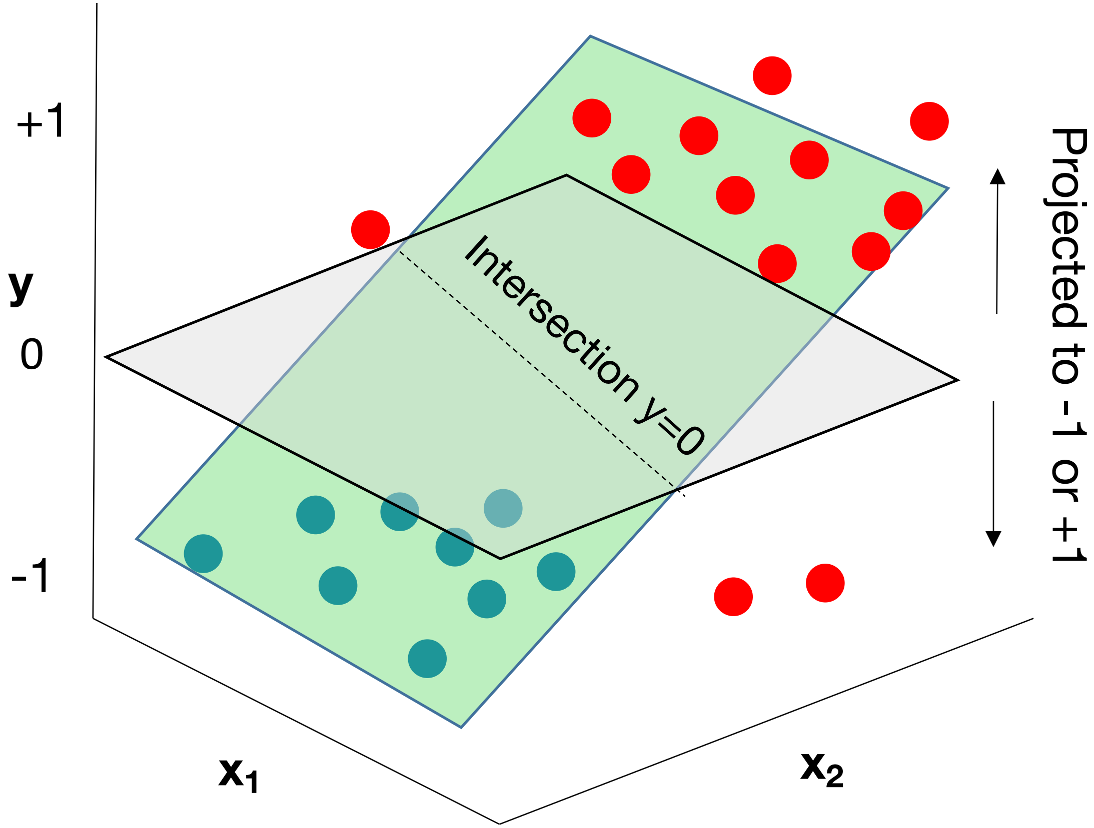
Linear classification:

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?



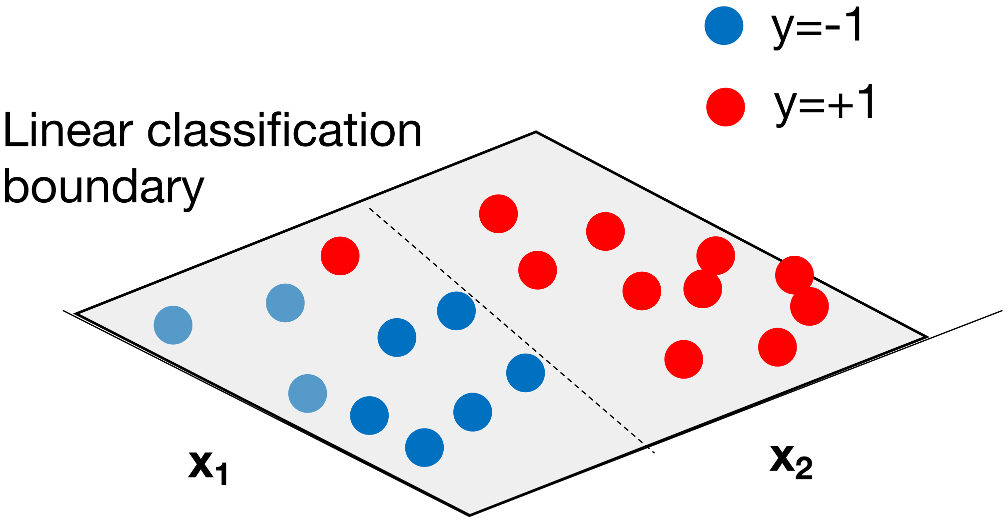
With sgn() operation:

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

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

- 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:

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

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

Closed-form solution available for this boundary line **w** via pseudo-inverse (see last lecture's notes)

Let's consider some other strategies to solve for **w**...

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

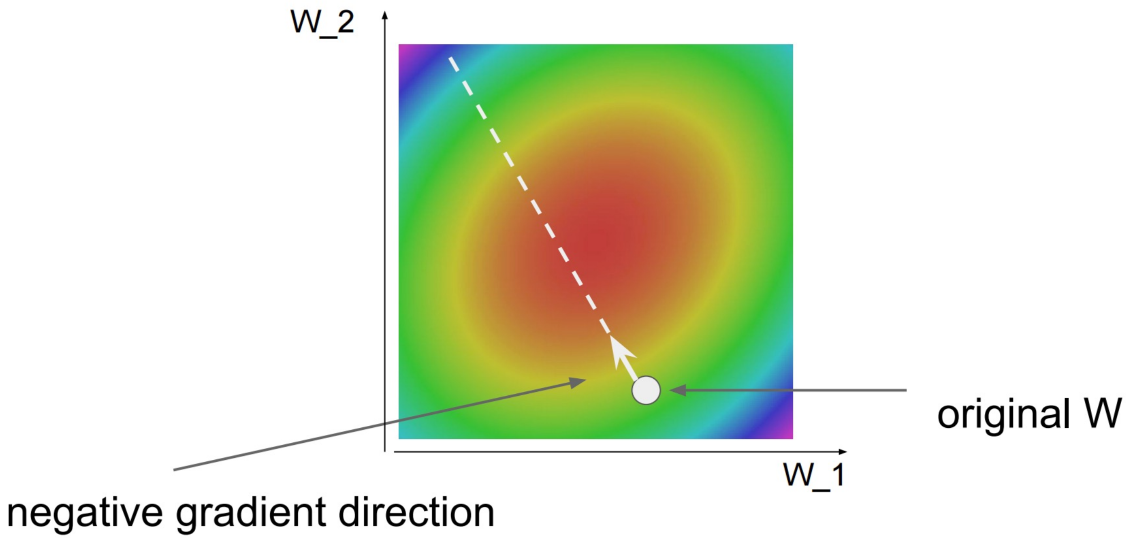
(harder)

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



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 $|\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)}| <$ 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

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 $|\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)}| < \text{threshold } t$

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

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

```
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x
    cur_x -= 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

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):

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

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

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):

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

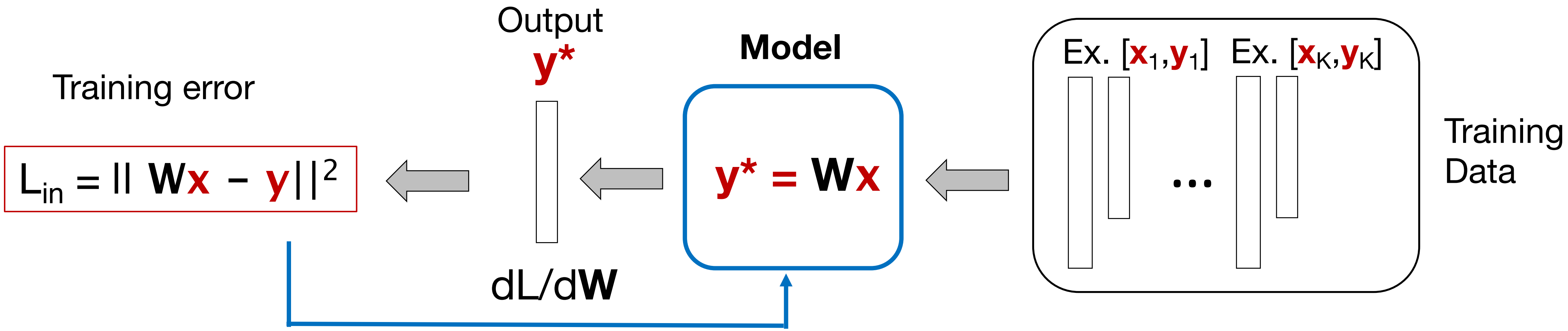
Then, evaluate at the next step:

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

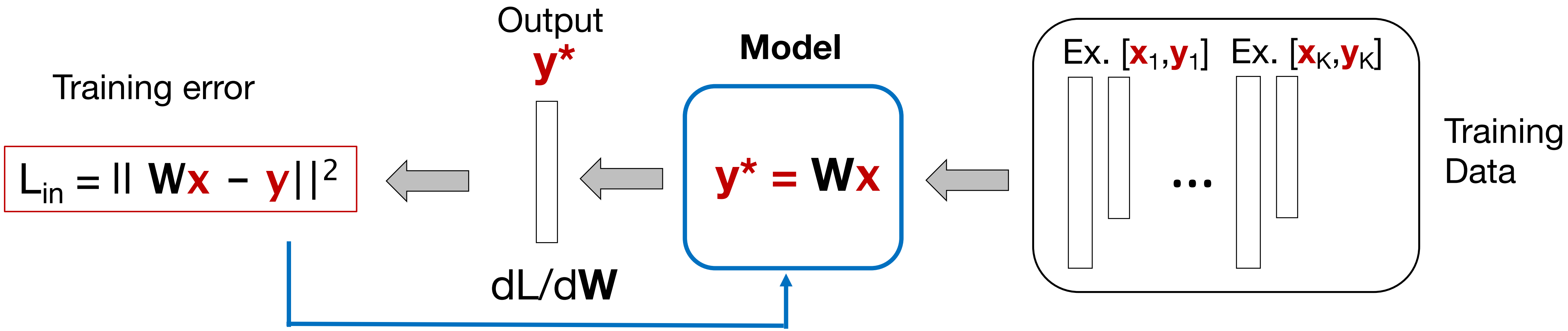
Solve for optimal step (when Hessian is positive): $\boxed{\epsilon^* = \frac{\boldsymbol{g}^\top\boldsymbol{g}}{\boldsymbol{g}^\top\boldsymbol{H}\boldsymbol{g}}.}$

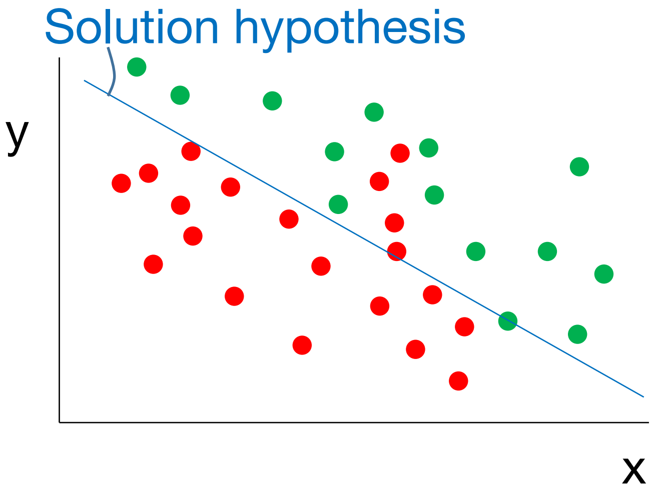
J. R. Shewchuck, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain"

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



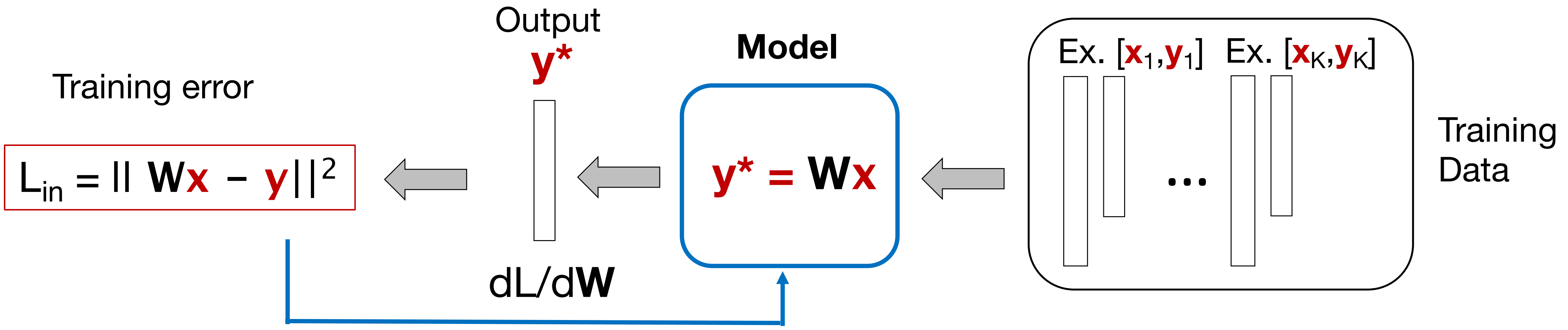
The linear classification model – what's not to like?

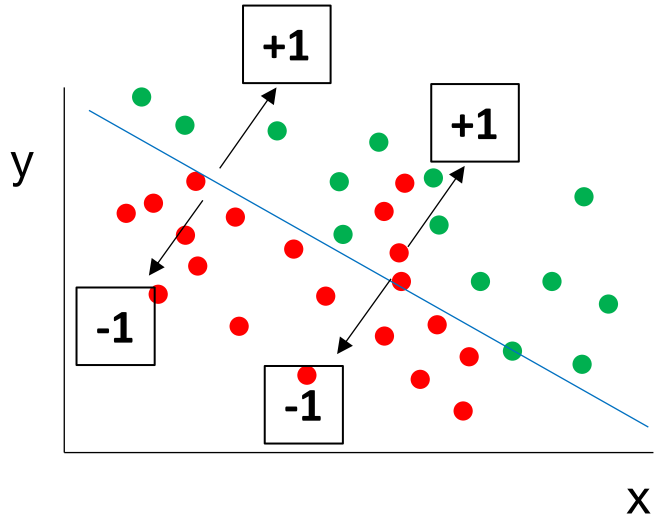




1. Can only separate data with lines (hyper-planes)…

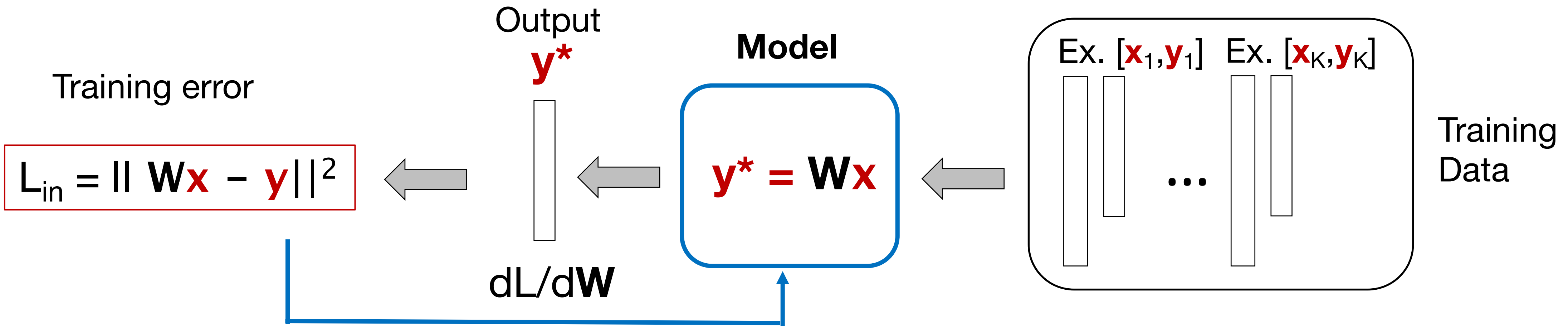
The linear classification model – what's not to like?

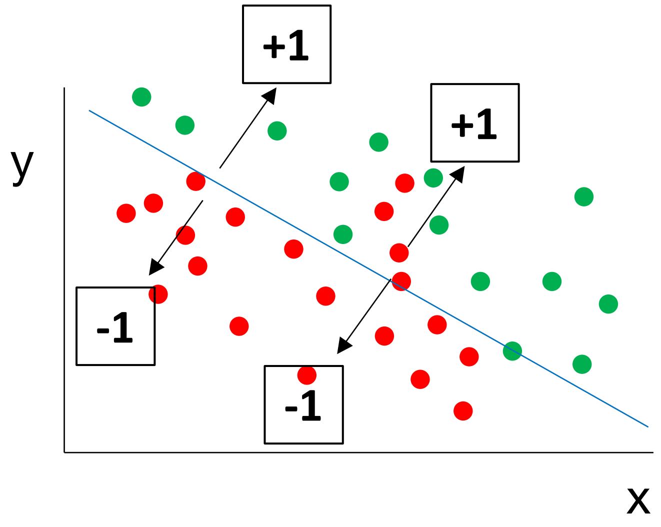




1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)

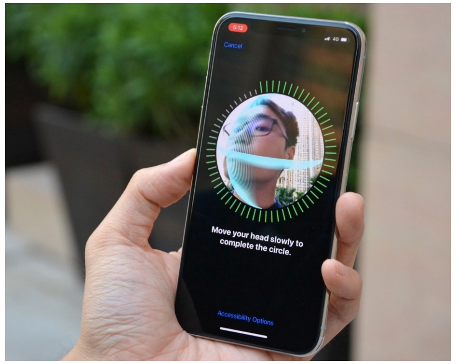
The linear classification model – what's not to like?



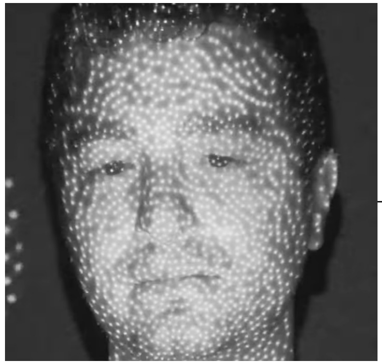


1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data

Cost functions matter: a simple example



x

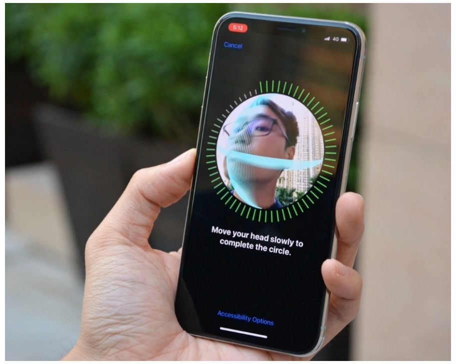


$$f(\mathbf{x}, W) \longrightarrow \begin{cases} +1 = \text{You} \\ -1 = \text{Bad guy} \end{cases}$$

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

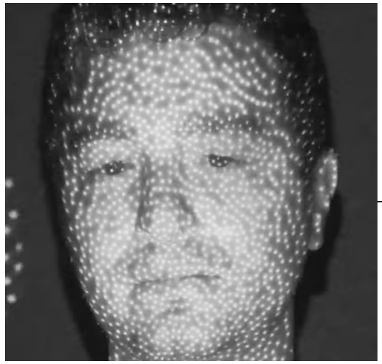
Cost functions matter: a simple example

deep imaging



→

x



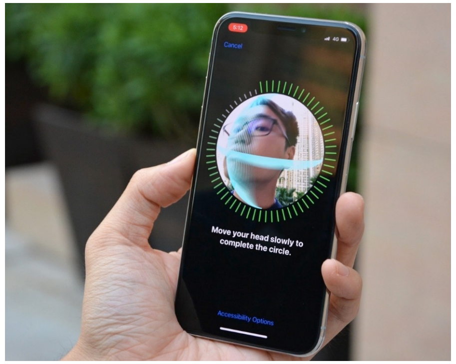
→ $f(\textcolor{red}{x}, W)$ → +1 = You / -1 = Bad guy

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

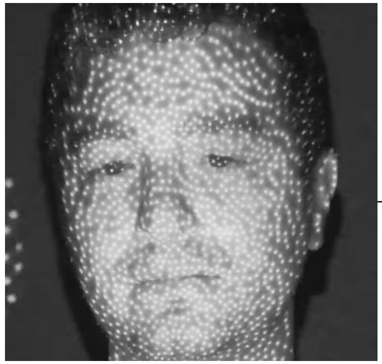
$f(\textcolor{red}{x}, W)$

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?

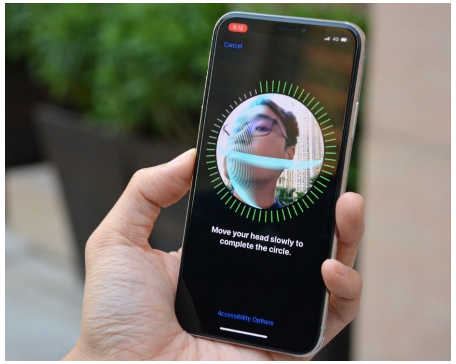
$f(x, W)$

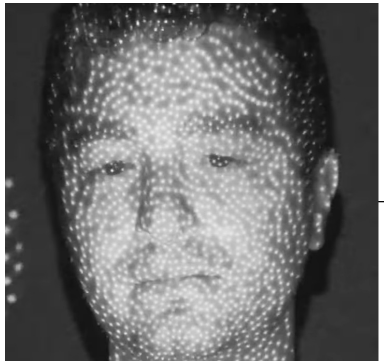
y \ $f(x, W)$	+1	-1
+1	No Error	False reject
-1	False accept	No Error

It's you, but you can't get in…

Letting an intruder in

Cost functions matter: a simple example



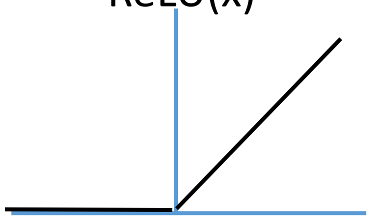


$x \rightarrow f(x, W) \rightarrow \begin{cases} +1 = \text{You} \\ -1 = \text{Bad guy} \end{cases}$

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



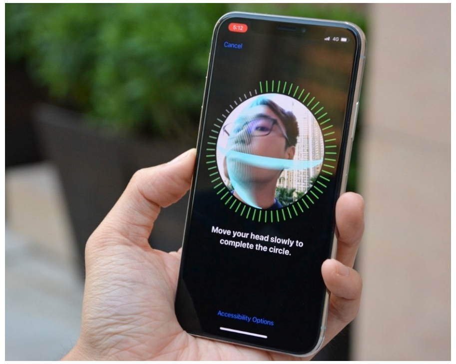
y \ f(x, W)	+1	-1
+1	No Error	False reject
-1	False accept	No Error

It's you, but you can't get in…

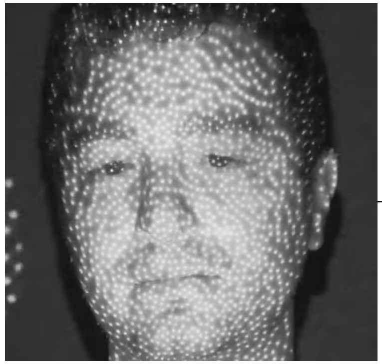
Letting an intruder in

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

Cost functions matter: a simple example



$\rightarrow$ x



$\rightarrow f(x, W) \rightarrow$

+1 = You

-1 = Bad guy

Two types of error: false accept and false reject

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

$$L_{in} = \underbrace{\mathrm{ReLU}[f(x, W)-y]}_{\text{Penalty for intruder}} + \underbrace{\mathbf{10}\ \mathrm{ReLU}[y-f(x, W)]}_{\text{Large penalty for annoyance...}}$$

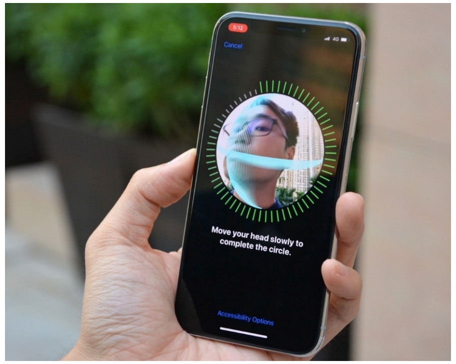
f(x, W)

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…

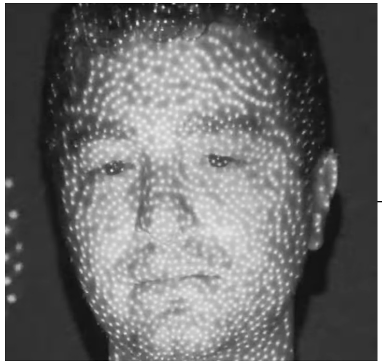
Letting an intruder in

Cost functions matter: a simple example



$\rightarrow$

x



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

What if you're a CIA agent?

f(x, W)

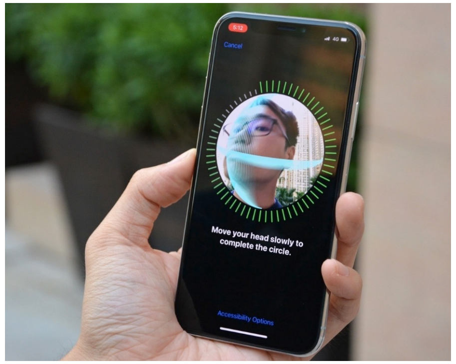
y \ f(x, W)	+1	-1
+1	No Error	False reject
-1	False accept	No Error

False reject: It's you, but you can't get in…

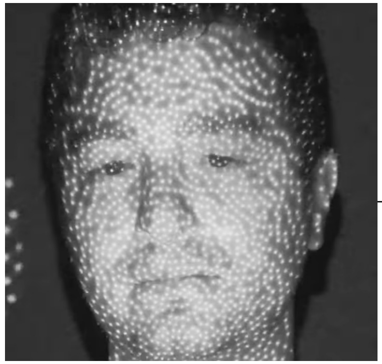
False accept: Letting an intruder in

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…

$f(x, W)$

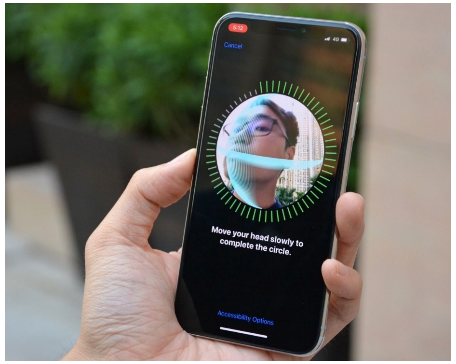
y \ $f(x, W)$	+1	-1
+1	No Error	False reject
-1	False accept	No Error

It's you, but you can't get in…

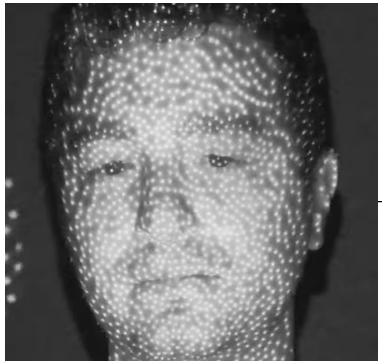
Letting an intruder in

Cost functions matter: a simple example

deep imaging



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 and minimize these probabilities

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

Machine learning and probability

deep imaging

- 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:

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

Machine learning and probability

deep imaging

- 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

$$p(\mathbf{w}|\mathbf{x}_1, ...\mathbf{x}_N; \mathbf{y}_1, ...\mathbf{y}_N)$$

Machine learning and probability

deep imaging

- 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 $p(\mathbf{w}|\mathbf{x}_1,...\mathbf{x}_N;\mathbf{y}_1,...\mathbf{y}_N)$

- 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

deep imaging

- 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 $\quad p(\mathbf{w}|\mathbf{x}_1,\ldots\mathbf{x}_N;\mathbf{y}_1,\ldots\mathbf{y}_N)$

- 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

deep imaging

- 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,

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

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,

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

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

Linear classification is the maximum likelihood for Gaussian data

deep imaging

- 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:

$$y_i = \mathbf{w}^T \mathbf{x}_i + n$$ (*n* 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?

$$\downarrow$$

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?

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?

$\downarrow$

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?

$\downarrow$

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

Linear classification is the maximum likelihood for Gaussian data

deep imaging

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

$\downarrow$

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?

$\downarrow$

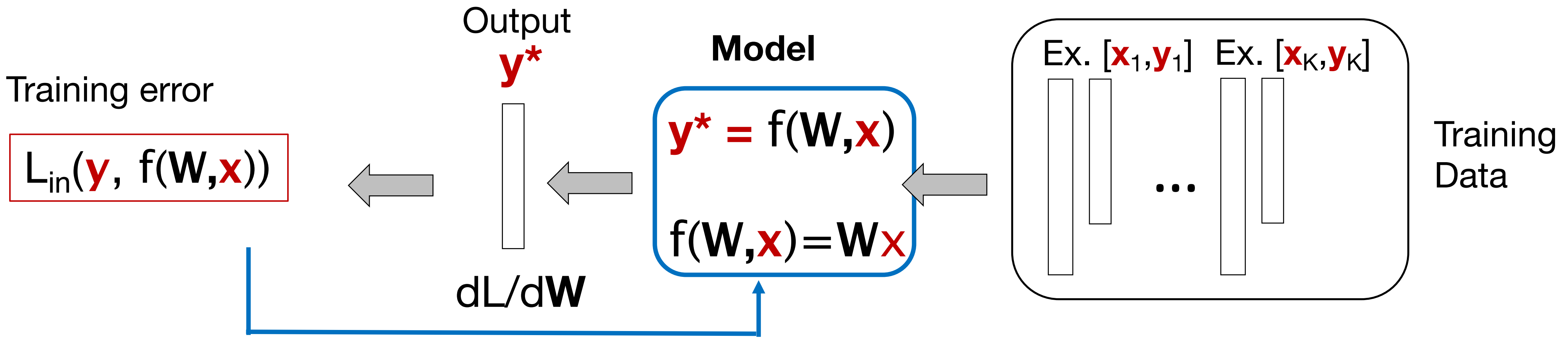
For what $\mathbf{w}$ is $\sum_{i=1}^{N} \frac{-(\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}$ maximized?

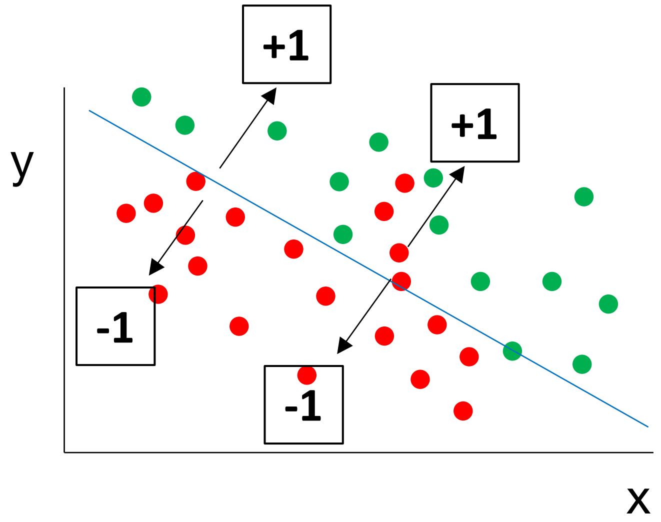
$\downarrow$

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

Summary: Linear classification with MSE assumes model output deviates from true labels via a Gaussian random process. Is this fair, given that labels are ether -1 or +1?

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

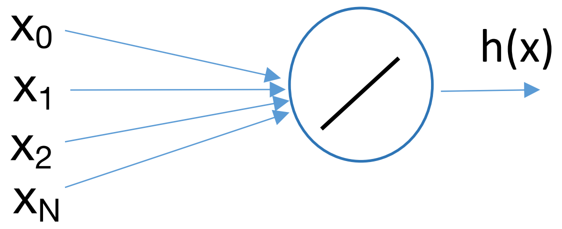




1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data

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

Linear regression: predict some output h(x) from x

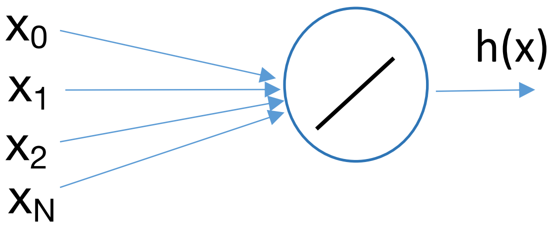


Let's think about the labels as a probabilistic measure:

Linear regression: predict some output h(x) from x

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

Linear regression



Not a probabilistic mapping

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

Linear regression: predict some output h(x) from x

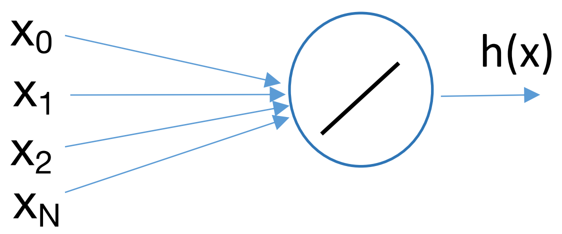
$$h(x) = x \in (-\infty, \infty)$$

Linear classifier: predict binary output h(x) from x

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

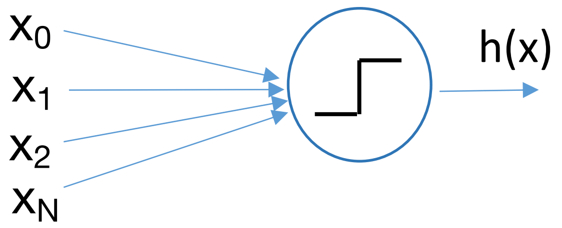
$$h(x) = \mathrm{sign}(x) \in \{0, 1\}$$

Linear regression



Not a probabilistic mapping

Sign(x)



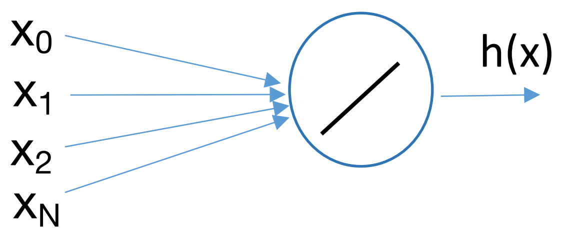
Probabilistic, but all-or-nothing: either 0, or 1

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

Linear regression:

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

Linear regression



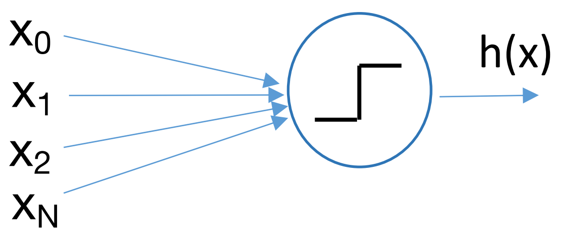
Not a probabilistic mapping

Linear classifier:

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

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

Sign(x)



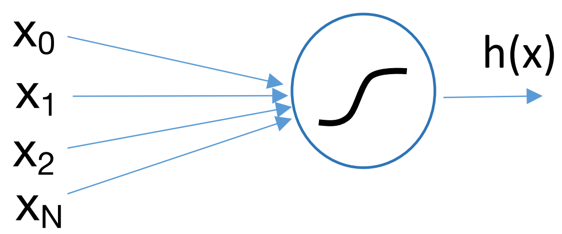
Probabilistic, but all-or-nothing: either 0, or 1

Logistic classifier:

$$\boxed{h(x) = \theta(x) \in [0, 1]}$$

Any value between 0 and 1

θ(x)



Probabilistic: continuous value between 0 and 1

Probabilistic interpretation of function that maps outputs to labels, $h(x) = \theta(x)$

Example: You are trying to predict the probability that a patient may have a certain form a disease, $\theta(\mathbf{x})$, given a number of observations and measurements, $\mathbf{x}$

Example: You are trying to predict the probability of rain tomorrow, $\theta(\mathbf{x})$, given a set of satellite image data, $\mathbf{x}$

deep imaging

Probabilistic interpretation of function that maps outputs to labels, h(x) = θ(x)

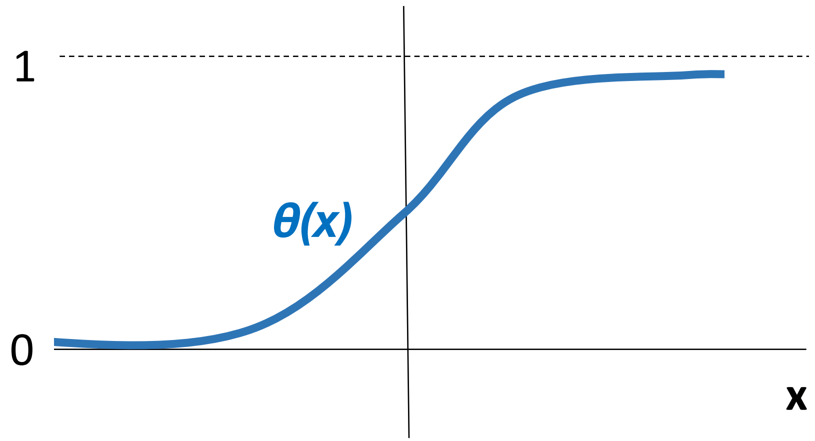
Example: You are trying to predict the probability that a patient may have a certain form a disease, θ(**x**), given a number of observations and measurements, **x**

Example: You are trying to predict the probability of rain tomorrow, θ(**x**), given a set of satellite image data, **x**

The Logistic Function *θ*

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

Also called Sigmoid function



- Use soft threshold to map any number to [0,1] range
- Sigmoid "flattens out" x

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 this approach more often than not in Tensorflow…

From linear classification to logistic classification

deep imaging

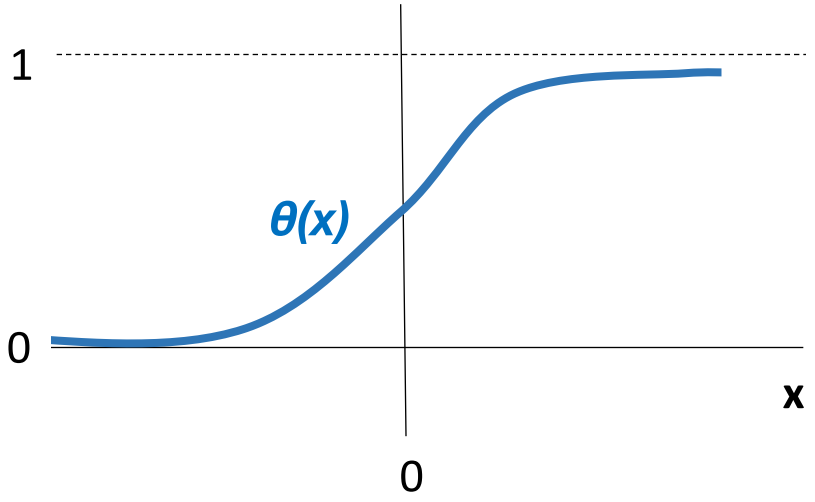
- Let's re-derive a cost function for the case where labels are treated *as probabilities*
  - You'll use this approach more often than not in Tensorflow…
- During learning, we will again have two classes (in this simple example), y= +/- 1
- map these binary values onto a [0,1] probability distribution

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 this approach 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

Formula for likelihood using the logistic function θ, given binary labels

$$P(y \mid \mathbf{x}) = \begin{cases} \theta(\mathbf{x}) & \text{For } y = +1 \\ & \end{cases}$$

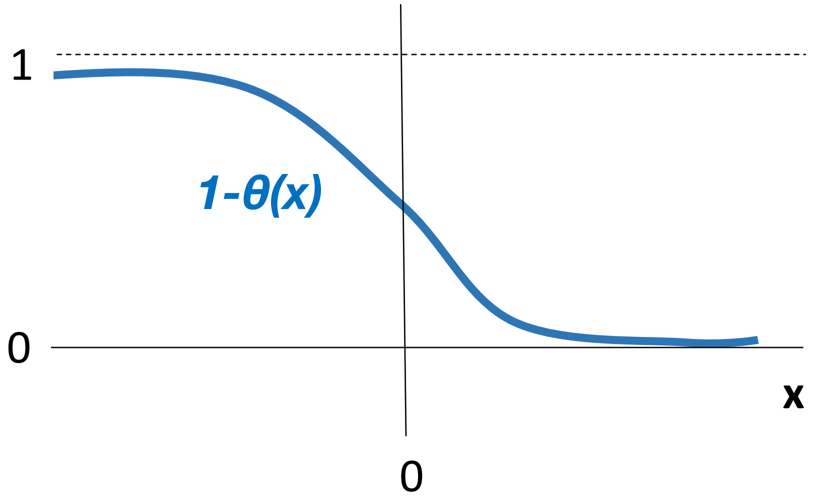


From linear classification to logistic classification

- Let's re-derive a cost function for the case where labels are treated *as probabilities*
  - You'll use this approach 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

Formula for likelihood using the logistic function θ, given binary labels

$$P(y \mid \mathbf{x}) = \begin{cases} \theta(\mathbf{x}) & \text{For } y = +1 \\ 1 - \theta(\mathbf{x}) & \text{For } y = -1 \end{cases}$$

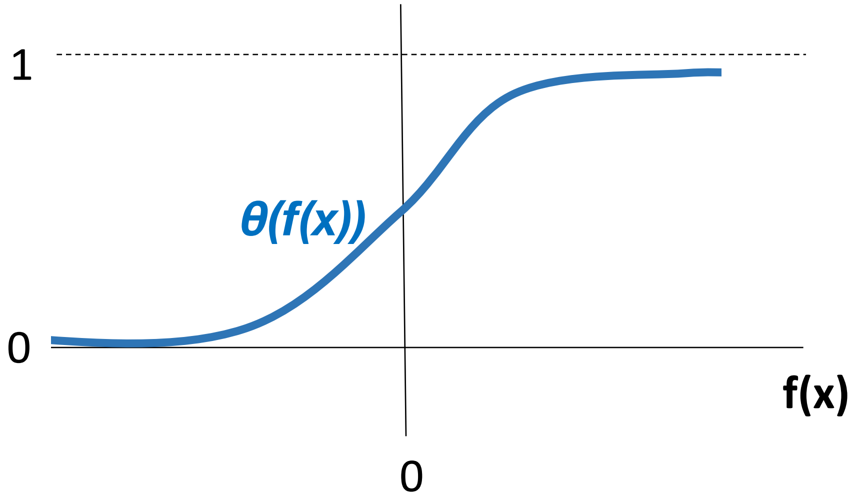


From linear classification to logistic classification

- Let's re-derive a cost function for the case where labels are treated as probabilities
  - You'll use this approach 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

Formula for likelihood using the logistic function θ, given binary labels

$$P(y \mid f(\mathbf{x})) = \begin{cases} \boxed{\theta(f(\mathbf{x})) \quad \text{For } y = +1} \\ 1 - \theta(f(\mathbf{x})) \quad \text{For } y = -1 \end{cases}$$



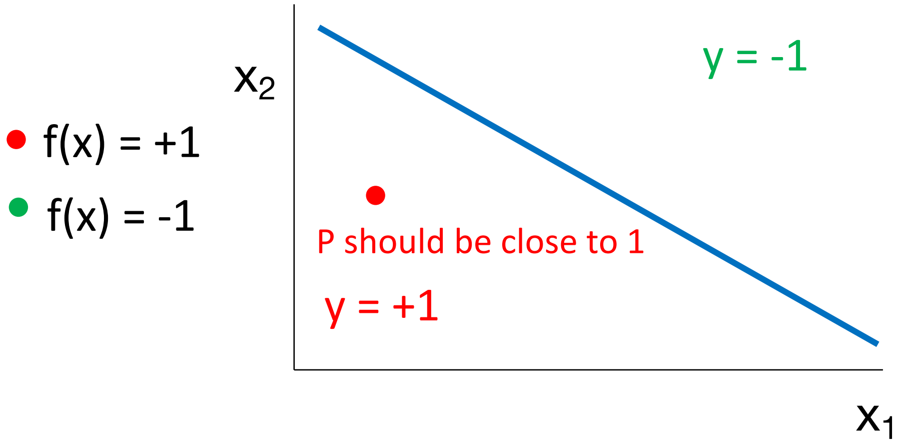
From linear classification to logistic classification

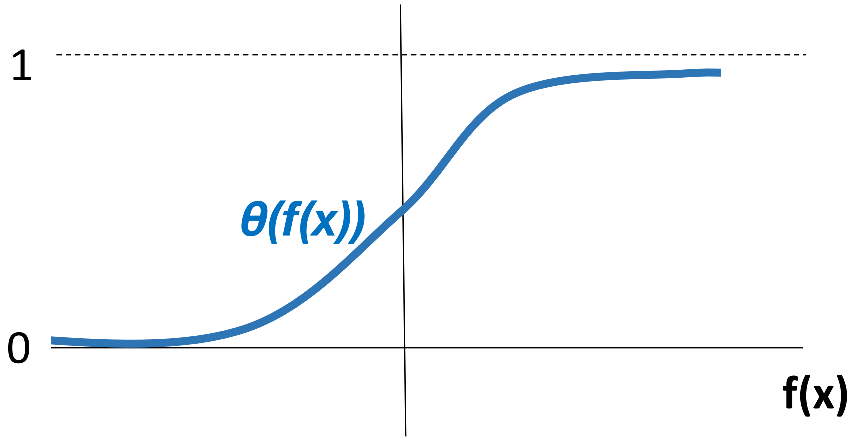
deep imaging

- Let's re-derive a cost function for the case where labels are treated as probabilities
  - You'll use this approach 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

Formula for likelihood using the logistic function θ, given binary labels

$$P(y \mid f(\mathbf{x})) = \begin{cases} \boxed{\theta(f(\mathbf{x})) \quad \text{For } y = +1} \\ 1 - \theta(f(\mathbf{x})) \quad \text{For } y = -1 \end{cases}$$





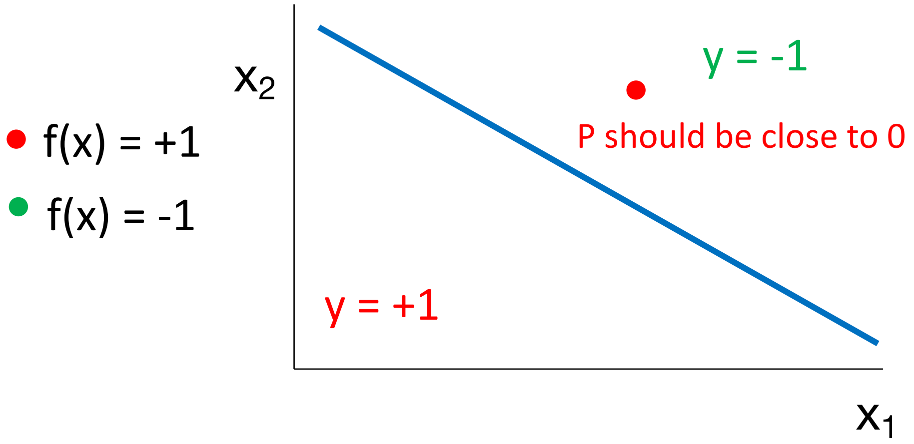
- If network output f(x) is large, then should map to y=+1 with high probability
- θ(f(x)) is large for large value of x

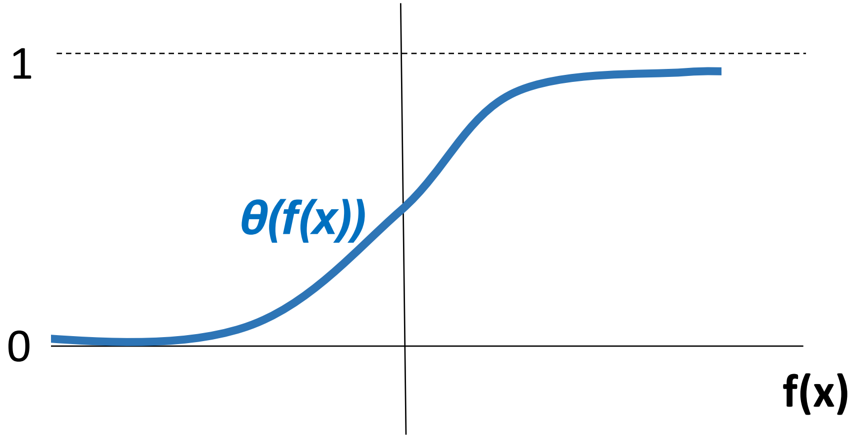
From linear classification to logistic classification

- Let's re-derive a cost function for the case where labels are treated as probabilities
  - You'll use this approach 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

Formula for likelihood using the logistic function θ, given binary labels

$$P(y \mid f(\mathbf{x})) = \begin{cases} \boxed{\theta(f(\mathbf{x})) \quad \text{For } y = +1} \\ 1 - \theta(f(\mathbf{x})) \quad \text{For } y = -1 \end{cases}$$



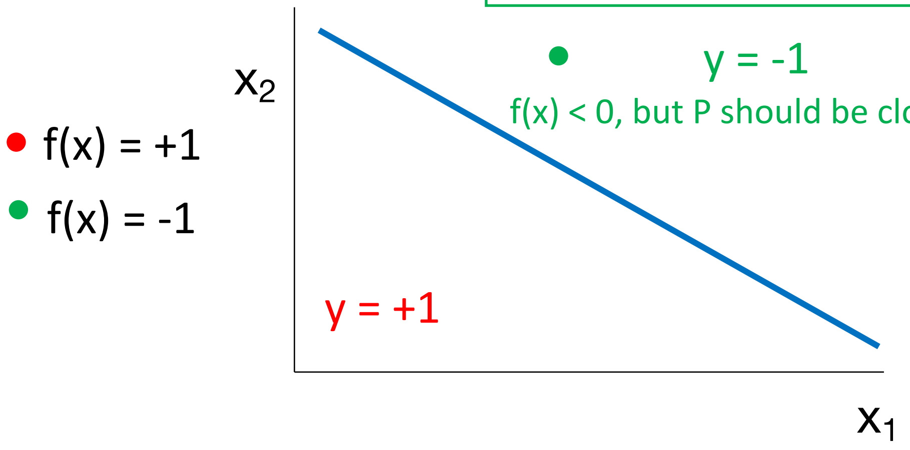


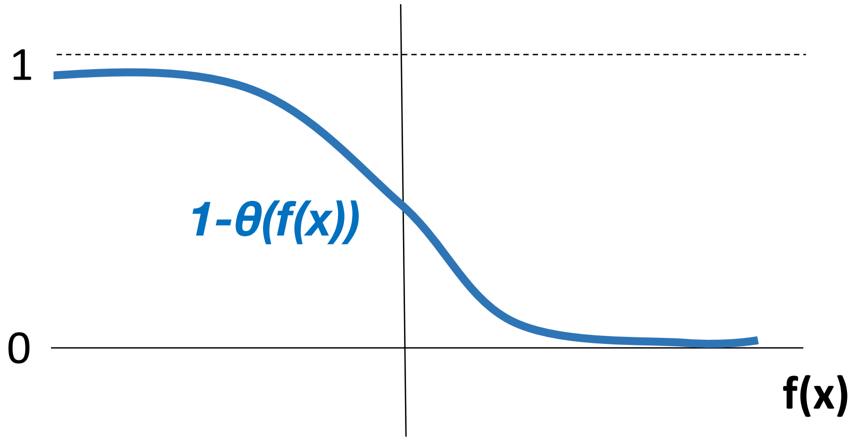
From linear classification to logistic classification

- Let's re-derive a cost function for the case where labels are treated as probabilities
  - You'll use this approach 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

Formula for likelihood using the logistic function θ, given binary labels

$$P(y \mid f(\mathbf{x})) = \begin{cases} \theta(f(\mathbf{x})) & \text{For } y = +1 \\ \boxed{1 - \theta(f(\mathbf{x})) \quad \text{For } y = -1} \end{cases}$$





- If network output f(x) is small, then should map to y=-1 with high probability
- θ(f(x)) ~0 for small values of f(x), so 1- θ(f(x)) ~1 is high probability to y=-1 mapping

Deriving cost function for logistic classification for probabilistic outputs

Instead of mapping f(**x**) to either +1 or -1 with the sign operator, let's use θ to map it to lie between 0 and 1:

$$P(y \mid \mathbf{x}) = \begin{cases} \theta(f(\mathbf{x})) & \text{For } y = +1 \\ 1 - \theta(f(\mathbf{x})) & \text{For } y = -1 \end{cases}$$

Deriving cost function for logistic classification for probabilistic outputs

deep imaging

Instead of mapping $f(\mathbf{x})$ to either +1 or -1 with the sign operator, let's use θ to map it to lie between 0 and 1:

$$P(y \mid \mathbf{x}) = \begin{cases} \theta(f(\mathbf{x})) & \text{For } y = +1 \\ 1 - \theta(f(\mathbf{x})) & \text{For } y = -1 \end{cases}$$

We'll stick with the case of linear classification, where $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

Also, please note that for the logistic function, $\theta(-a) = 1 - \theta(a)$.

Deriving cost function for logistic classification for probabilistic outputs

Instead of mapping f(**x**) to either +1 or -1 with the sign operator, let's use θ to map it to lie between 0 and 1:

$$P(y \mid \mathbf{x}) = \begin{cases} \theta(f(\mathbf{x})) & \text{For } y = +1 \\ 1 - \theta(f(\mathbf{x})) & \text{For } y = -1 \end{cases}$$

We'll stick with the case of linear classification, where $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

Also, please note that for the logistic function, $\theta(-a) = 1 - \theta(a)$.

So, we can summarize the case-based definition above with a single function,

$$P(y \mid \mathbf{x}) = \theta(y\, f(\mathbf{x}))$$

Deriving cost function for logistic classification for probabilistic outputs

Instead of mapping f($\mathbf{x}$) to either +1 or -1 with the sign operator, let's use θ to map it to lie between 0 and 1:

$$P(y \mid \mathbf{x}) = \begin{cases} \theta(f(\mathbf{x})) & \text{For } y = +1 \\ 1 - \theta(f(\mathbf{x})) & \text{For } y = -1 \end{cases}$$

We'll stick with the case of linear classification, where $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

Also, please note that for the logistic function, $\theta(-a) = 1 - \theta(a)$.

So, we can summarize the case-based definition above with a single function,

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

Here, y = +/-1 flips $\theta(\mathbf{w}^T\mathbf{x})$ to be either $\theta(\mathbf{w}^T\mathbf{x})$ or $\theta(-\mathbf{w}^T\mathbf{x}) = 1 - \theta(\mathbf{w}^T\mathbf{x})$

Deriving cost function for logistic classification for probabilistic outputs

Similar to the linear classification case, the likelihood of observing *N* independent outputs is given by,

$$P(y_1,y_2\ldots\, y_N \mid \mathbf{x}_1, \mathbf{x}_2, \ldots\mathbf{x}_N) = \prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n)$$

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

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)

$$\text{Maximize} \quad P(y_1,y_2\ldots\, y_N \mid \mathbf{x}_1, \mathbf{x}_2, \ldots\mathbf{x}_N) = \prod_{n=1}^{N} \theta(y_n\, \mathbf{w}^T\mathbf{x}_n)$$

Deriving cost function for logistic classification for probabilistic outputs

deep imaging

Maximize $P(y_1, y_2 \ldots y_N \mid \mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N) = \prod_{n=1}^{N} \theta(y_n \, \mathbf{w}^T \mathbf{x}_n)$

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

Deriving cost function for logistic classification for probabilistic outputs

deep imaging

Maximize $P(y_1, y_2 \ldots y_N \mid \mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^T \mathbf{x}_n)$

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

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

Use relationship $$\theta(a) = \frac{1}{1 + e^{-a}}$$

Deriving cost function for logistic classification for probabilistic outputs

Maximize $P(y_1, y_2 \ldots y_N \mid \mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^T \mathbf{x}_n)$

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

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

Use relationship $$\theta(a) = \frac{1}{1 + e^{-a}}$$

Minimize $$\boxed{L_{in}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}})}$$

Cross entropy error for logistic classification

Typically requires iterative solution to minimize

Deriving cost function for logistic classification for probabilistic outputs

deep imaging

Maximize $P(y_1, y_2 \ldots y_N \mid \mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^T \mathbf{x}_n)$

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

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

Use relationship $$\theta(a) = \frac{1}{1 + e^{-a}}$$

Minimize $$L_{in}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}})$$

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

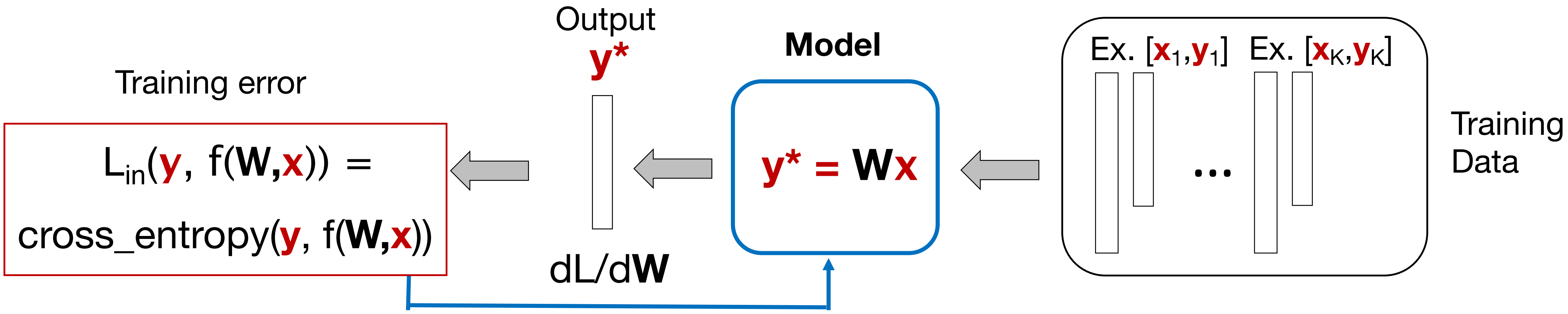
Cross entropy error for logistic classification

Typically requires iterative solution to minimize

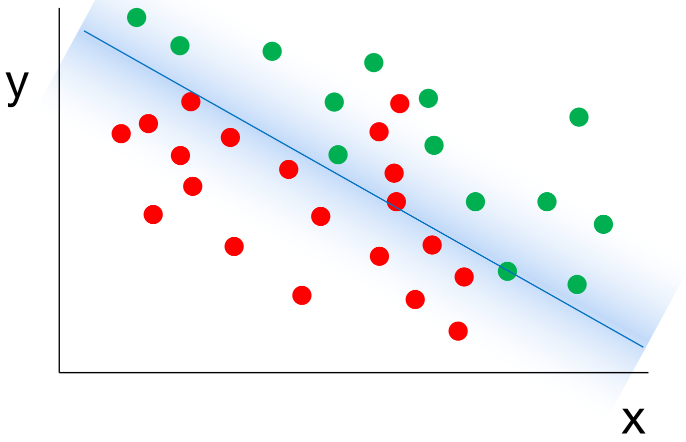
Mean-square error for linear classification

Closed form solution available

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

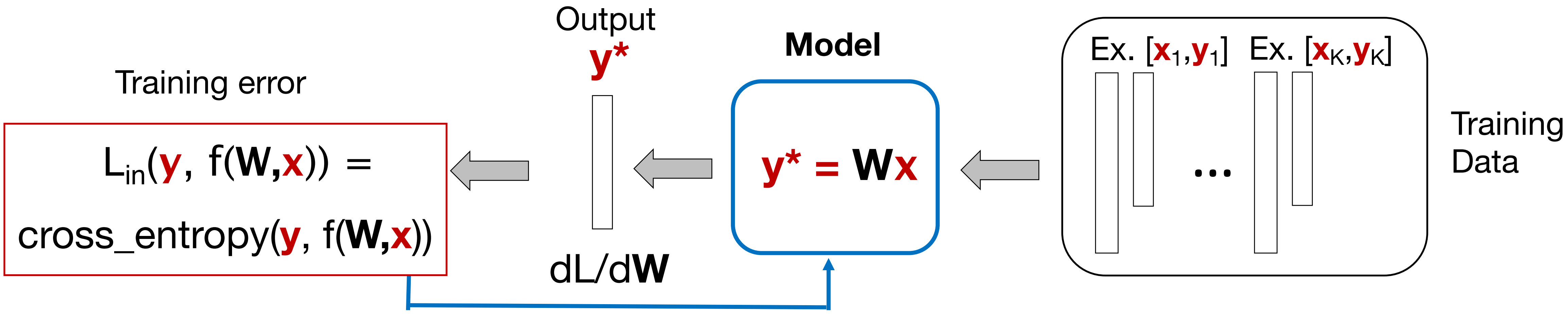


Probabilistic mapping to y

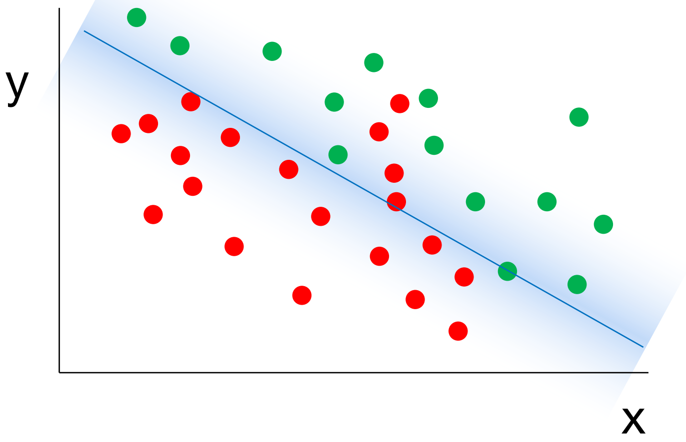


deep imaging

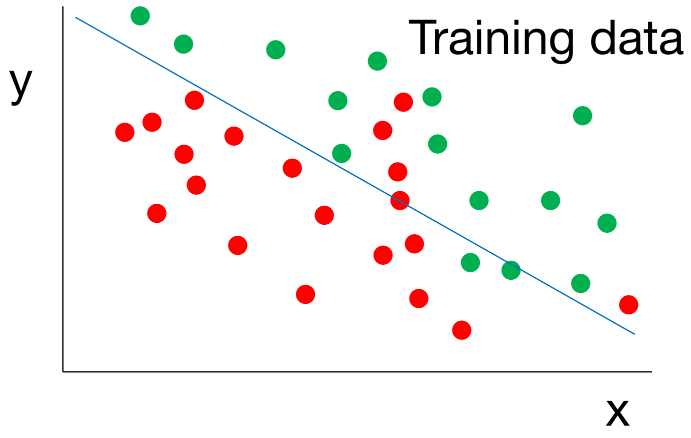
The linear classification model – what's not to like?



Probabilistic mapping to y

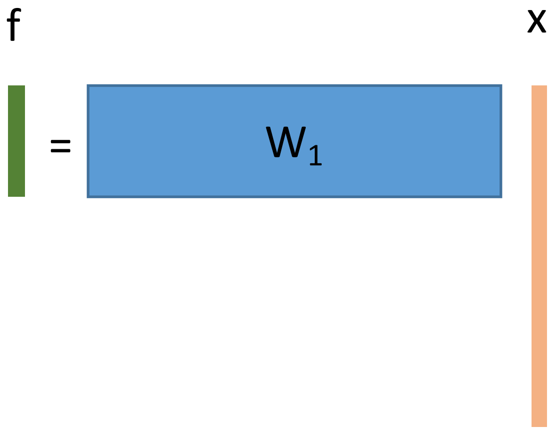


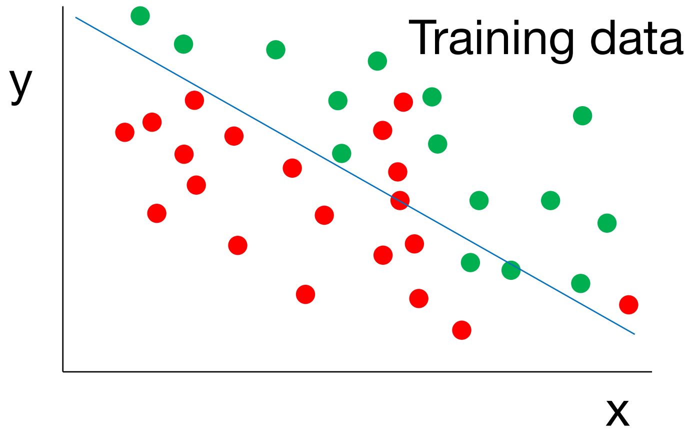
1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data



$$f = W_1 x$$

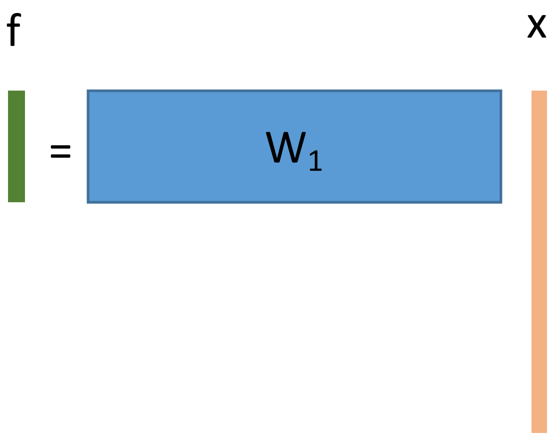
Learned *f:* not flexible





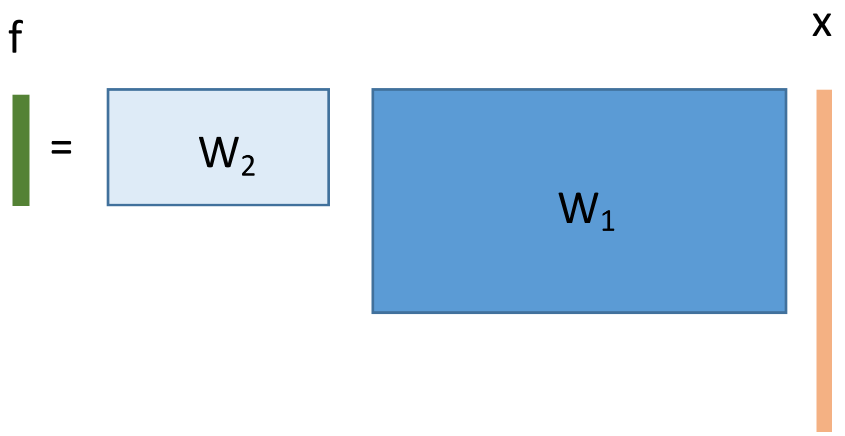
$$f = W_1 x$$

Learned *f:* not flexible



Can we add flexibility by multiplying with another weight matrix?

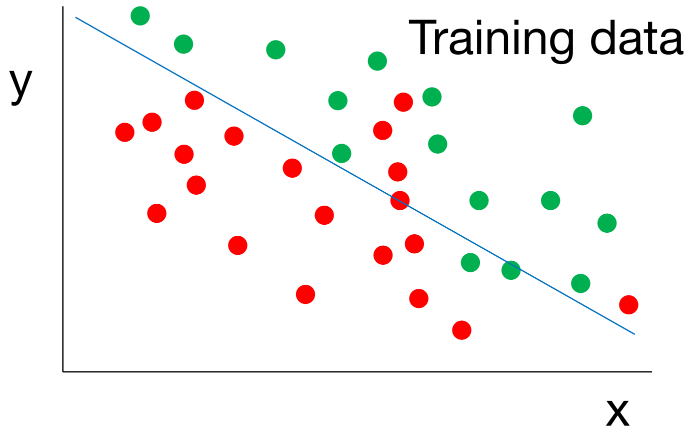
$$\begin{cases} f_1 = W_1 x + b_1 \\ f_2 = W_2 f_1 + b_2 \end{cases}$$



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

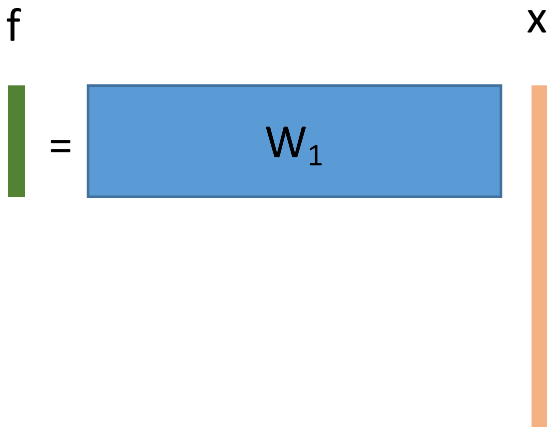
$$f_2 = W' x + b'$$

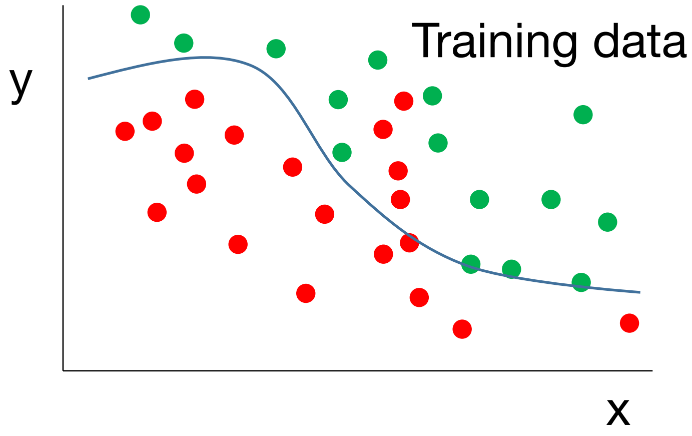
Unfortunately not…



$$f = W_1 x$$

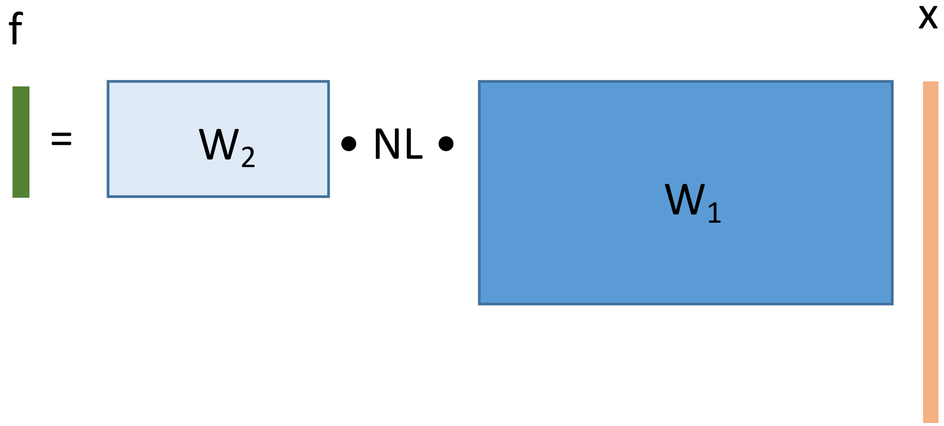
Learned *f:* not flexible



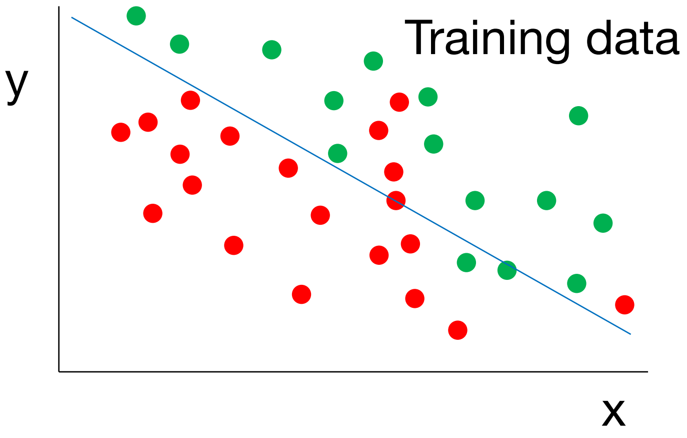


$$f = W_2 \max(W_1 x, 0)$$

Learned *f:* a bit flexible

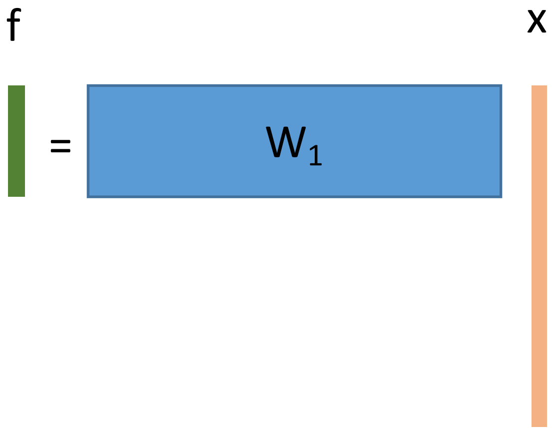


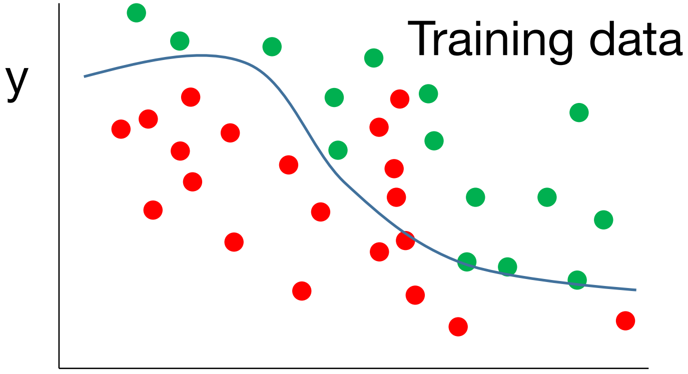
deep imaging



$$f = W_1 x$$

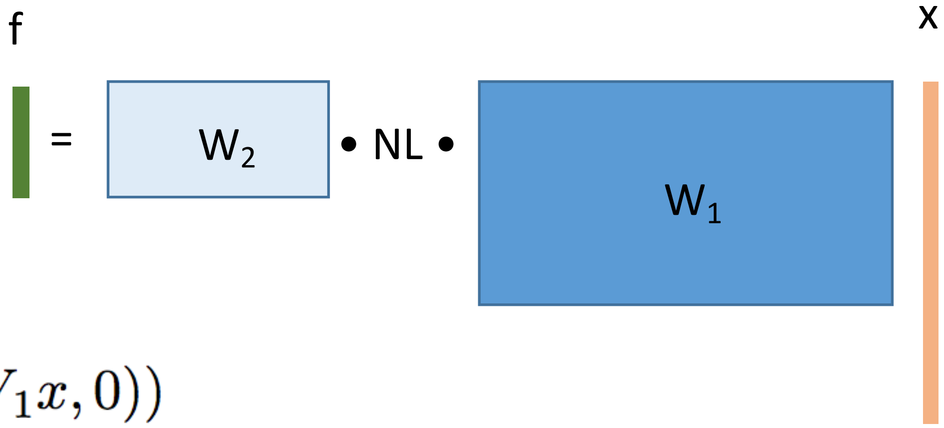
Learned *f:* not flexible

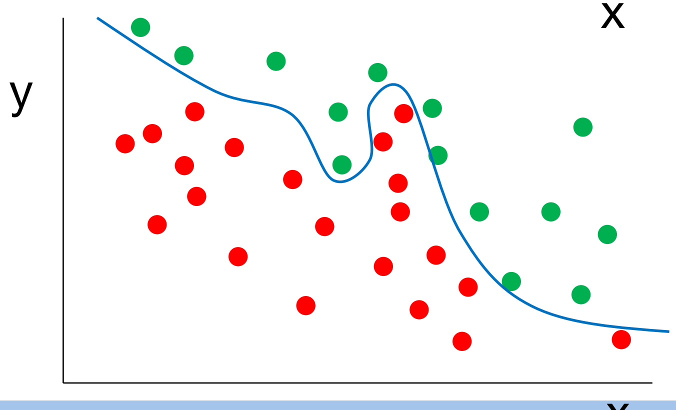




$$f = W_2 \max(W_1 x, 0)$$

Learned *f:* a bit flexible





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

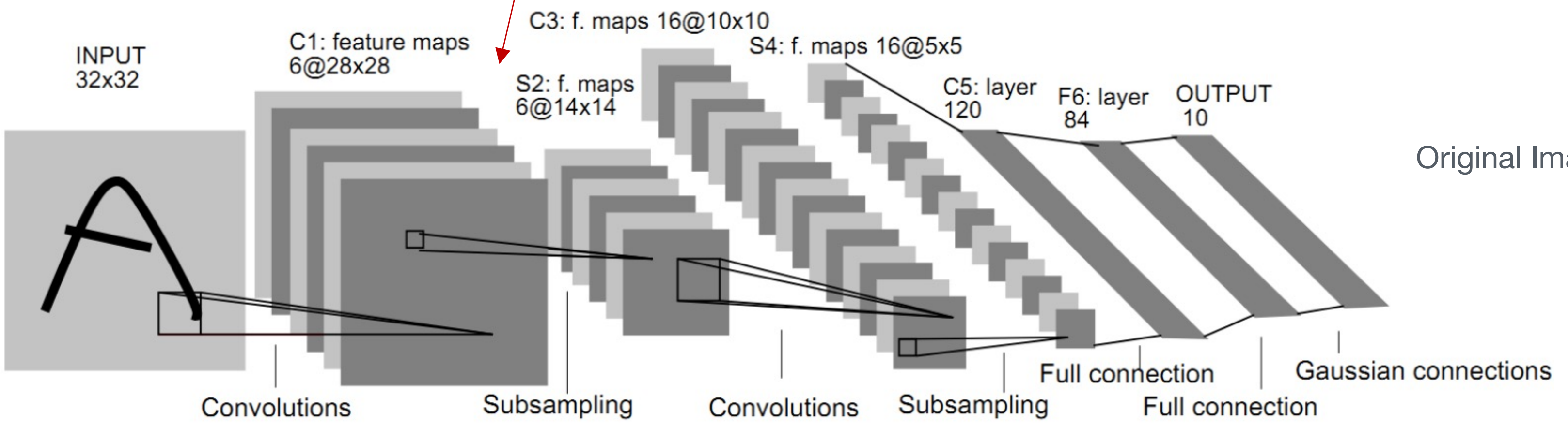
Learned *f:* more flexible

Does it generalize???

We can keep adding these “layers”…

Getting us to Convolutional Neural Networks

After, apply non-linearity and sub-sampling



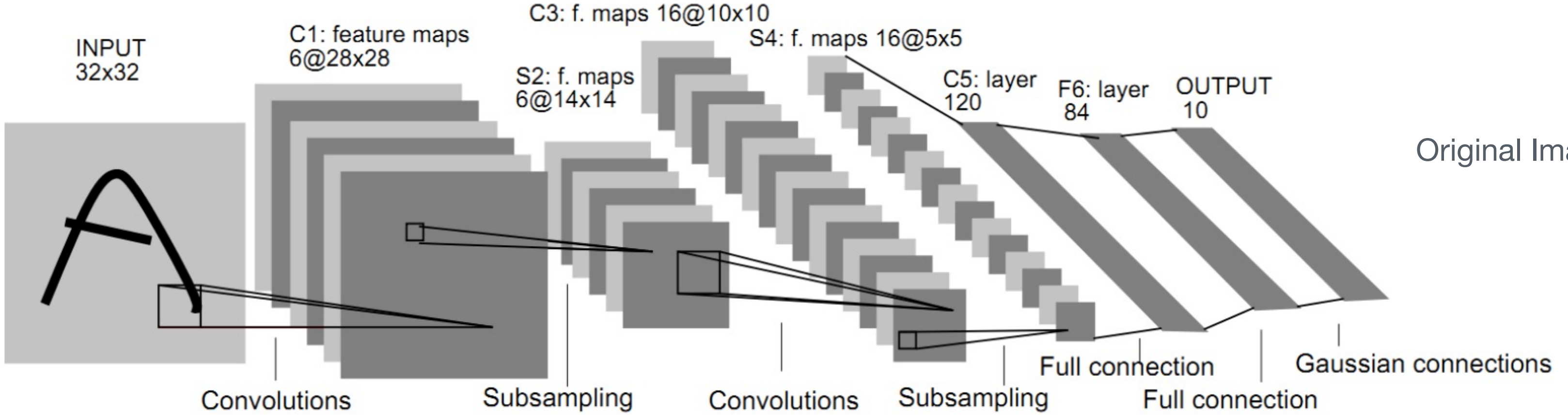
Original Image published in [LeCun et al., 1998]

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



Original Image published in [LeCun et al., 1998]

In practice, this process is repeated many times: