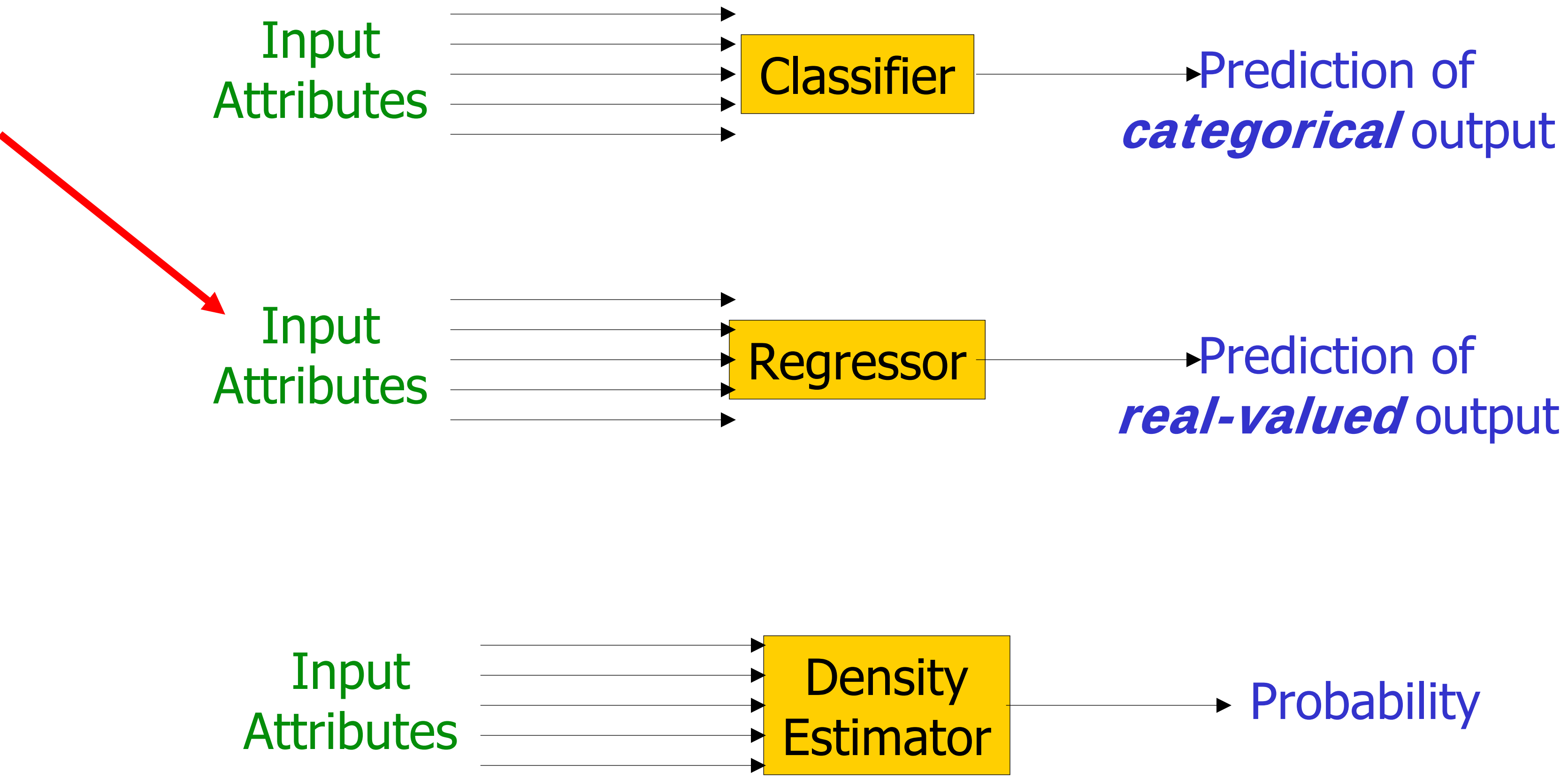
Linear Classifiers and Regressors

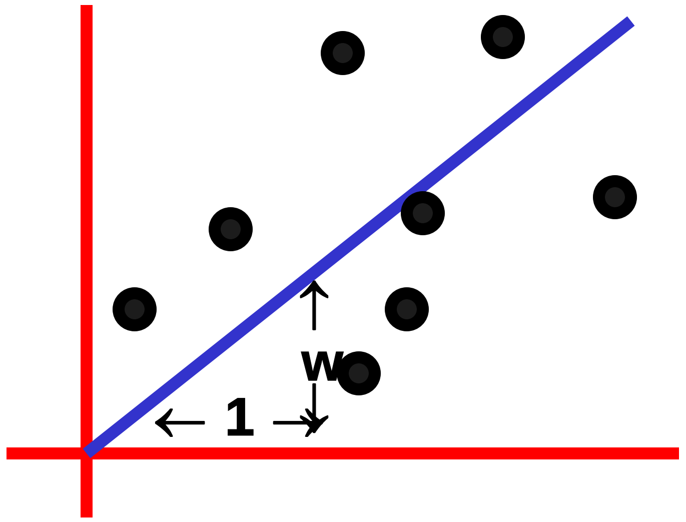
“Borrowed” with permission from Andrew Moore (CMU)

Single-Parameter Linear Regression

Regression vs Classification



Linear Regression



DATASET

inputs	outputs
$x_1 = 1$	$y_1 = 1$
$x_2 = 3$	$y_2 = 2.2$
$x_3 = 2$	$y_3 = 2$
$x_4 = 1.5$	$y_4 = 1.9$
$x_5 = 4$	$y_5 = 3.1$

Linear regression assumes

expected value of output *y* given input *x*, *E[y|x]*, is linear.

Simplest case: Out(x) = $w \times x$ for some unknown *w*.

Challenge: Given dataset, how to estimate *w*.

1-parameter linear regression

Assume data is formed by

$$y_i = w \times x_i + \text{noise}_i$$

where…

- noise signals are independent
- noise has *normal* distribution with mean *0* and unknown variance σ^2

$P(y|w,x)$ has a normal distribution with

- mean $w \times x$
- variance σ^2

Bayesian Linear Regression

$$P(y|w,x) = \text{Normal}(\text{mean } w \times x \, ; \ \text{var } \sigma^2)$$

Datapoints (x_1,y_1) (x_2,y_2) … (x_n,y_n)
are EVIDENCE about w.

Want to infer w from data:

$$P(w \mid x_1, x_2,\ldots, x_n, \ y_1, y_2 \ldots, y_n)$$

- ?? use BAYES rule to work out a posterior distribution for w given the data ??

- Or Maximum Likelihood Estimation ?

Maximum likelihood estimation of *w*

Question:

"For what value of *w* is this data most likely to have happened?"

$\Leftrightarrow$

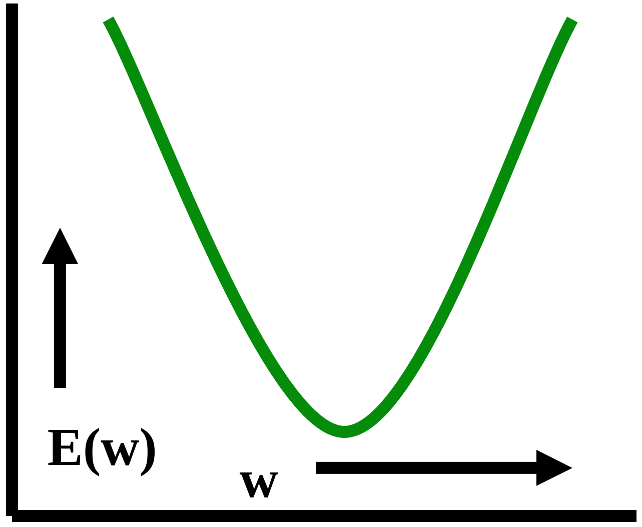
What value of *w* maximizes

$$P(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n, w) = \prod_{i=1}^{n} P(y_i \mid w, x_i) \quad ?$$

$$
\begin{aligned}
w^* &= \arg\max \left\{ \prod_{i=1}^{n} P(y_i | w, x_i) \right\} \\
&= \arg\max \left\{ \prod_{i=1}^{n} \exp(-\frac{1}{2}(\frac{y_i - w x_i}{\sigma})^2) \right\} \\
&= \arg\max \left\{ \sum_{i=1}^{n} -\frac{1}{2}\left(\frac{y_i - w x_i}{\sigma}\right)^2 \right\} \\
&= \arg\min \left\{ \sum_{i=1}^{n} \left(y_i - w x_i\right)^2 \right\}
\end{aligned}
$$

Linear Regression

Maximum likelihood w
minimizes
$E(w)$ =
sum-of-squares of residuals



$$\mathrm{E}(w) = \sum_i (y_i - wx_i)^2 = \sum_i y_i^2 - \left(2\sum x_i y_i\right)w + \left(\sum x_i^2\right)w^2$$

⇒ Need to minimize a quadratic function of w.

Linear Regression

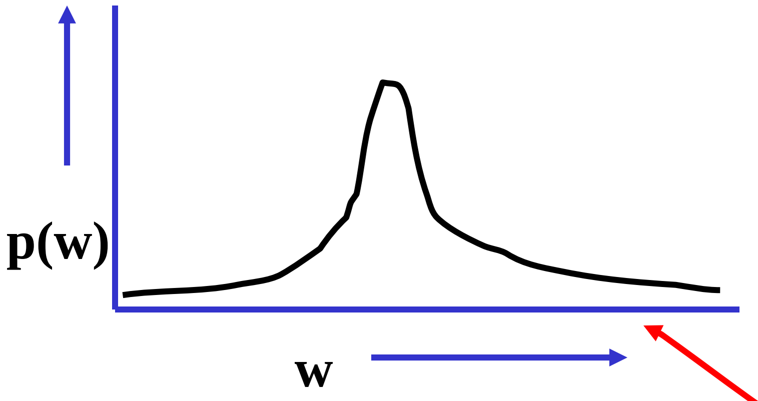
Sum-of-squares minimized when

$$w = \frac{\sum x_i y_i}{\sum x_i^2}$$

The maximum likelihood model is

$$\text{Out}(x) = w \times x$$

Can use for **prediction**



Note: Bayesian stats would provide a prob dist of *w*

… and predictions would give a prob dist of expected output

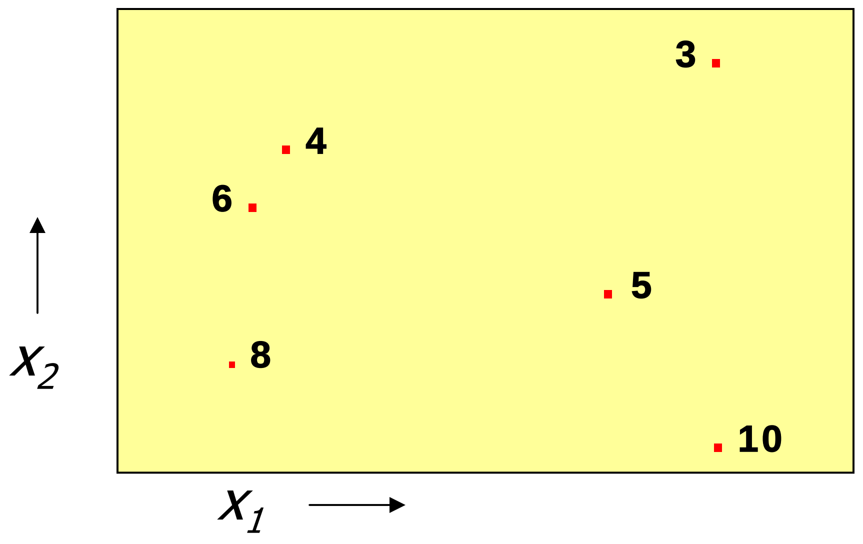
Often useful to know your confidence.

Max likelihood also provides kind of confidence!

Multi-variate Linear Regression

Multivariate Regression

What if inputs are *vectors*?



Input is 2-d;

Output value is “height”

Dataset has form

$\mathbf{x}_1$	y_1
$\mathbf{x}_2$	y_2
$\mathbf{x}_3$	y_3
.: .	:
$\mathbf{x}_R$	y_R

Multivariate Regression

R datapoints; each input has *m* components … as Matrices:

$$\mathbf{x} = \begin{bmatrix} \ldots\ldots\mathbf{x}_1\ldots\ldots \\ \ldots\ldots\mathbf{x}_2\ldots\ldots \\ \vdots \\ \ldots\ldots\mathbf{x}_R\ldots\ldots \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1m} \\ x_{21} & x_{22} & \ldots & x_{2m} \\ & & \vdots & \\ x_{R1} & x_{R2} & \ldots & x_{Rm} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_R \end{bmatrix}$$

IMPORTANT EXERCISE: PROVE IT !!!!!

Linear regression model assumes ∃ vector ***w*** s.t.

$$\text{Out}(\boldsymbol{x}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = w_1 x[1] + w_2 x[2] + \ldots + w_m x[m]$$

Max. likelihood $\boldsymbol{w} = (X^{\mathsf{T}}X)^{-1}(X^{\mathsf{T}}Y)$

Multivariate Regression (con't)

The max. likelihood ***w*** is ***w*** = $(X^T X)^{-1}(X^T Y)$

$X^T X$ is $m \times m$ matrix: i,j'th elt = $$\sum_{k=1}^{R} x_{ki} x_{kj}$$

$X^T Y$ is m-element vector: i'th elt = $$\sum_{k=1}^{R} x_{ki} y_k$$

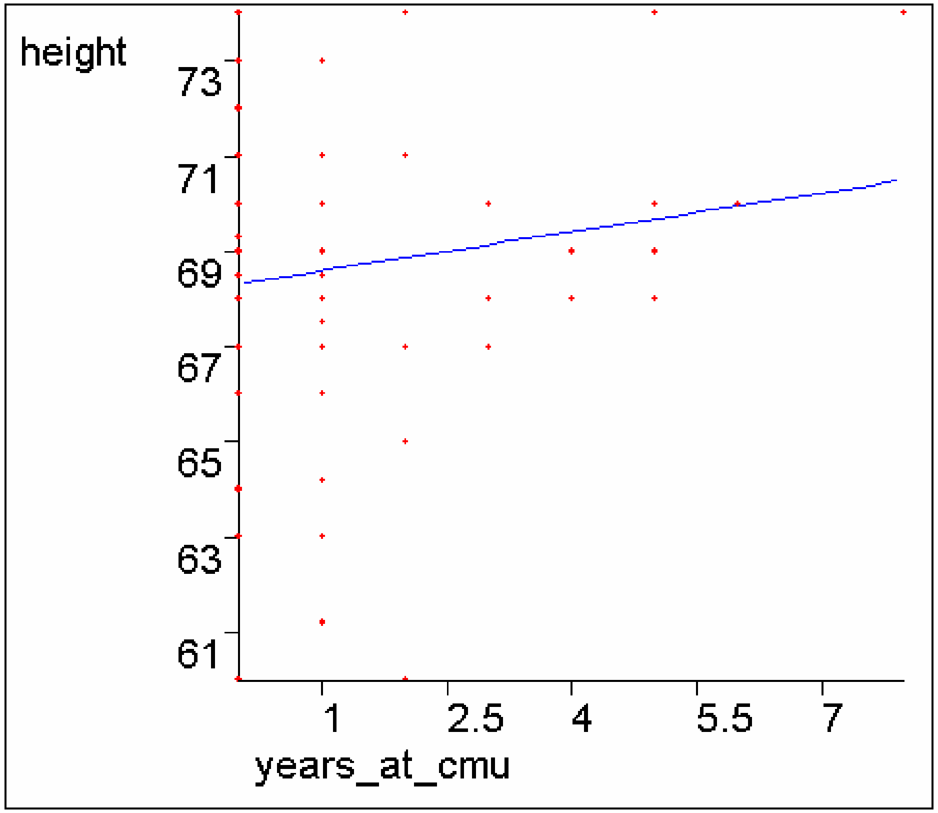
Constant Term in Linear Regression

What about a constant term?

What if
 linear data does not go through origin (0,0,…0) ?

Statisticians and Neural Net Folks all agree on a simple obvious hack.

Can you guess??



The constant term

- Trick: create fake input "X_0" that always takes value 1

X_1	X_2	Y
2	4	16
3	4	17
5	5	20

X_0	X_1	X_2	Y
1	2	4	16
1	3	4	17
1	5	5	20

Before:

$Y = w_1 X_1 + w_2 X_2$

…is a poor model

After:

$Y = w_0 X_0 + w_1 X_1 + w_2 X_2$

$= w_0 + w_1 X_1 + w_2 X_2$

…is good model!

Here, you should be able to see MLE w_0, w_1, w_2 by inspection

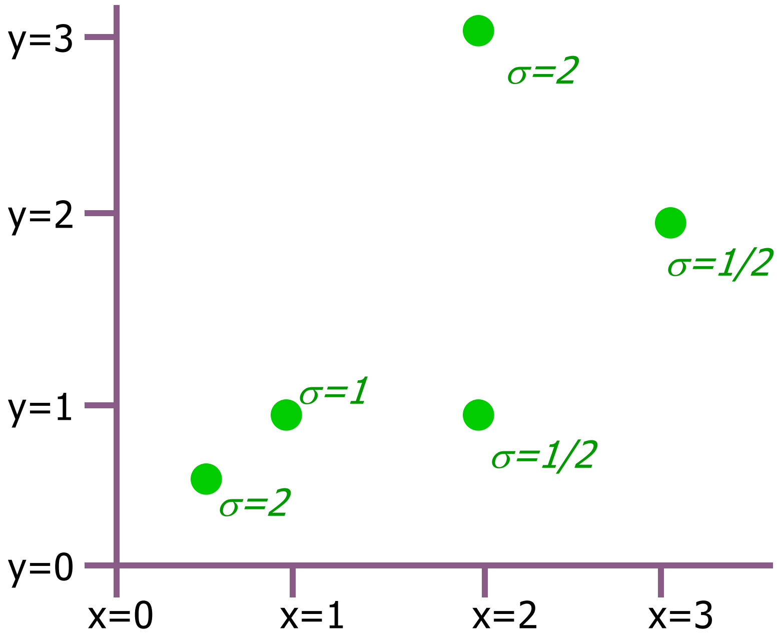
Heteroscedasticity...

Linear Regression with varying noise

Regression with varying noise

- Suppose you know variance of noise that was added to each datapoint.

x_i	y_i	σ_i^2
½	½	4
1	1	1
2	1	1/4
2	3	4
3	2	1/4



Assume $$y_i \sim N(wx_i, \sigma_i^2)$$

What's the MLE estimate of w?

MLE estimation with varying noise

$$\underset{w}{\operatorname{argmax}} \log p(y_1, y_2, \ldots, y_R \mid x_1, x_2, \ldots, x_R, \sigma_1^2, \sigma_2^2, \ldots, \sigma_R^2, w)$$

$$= \underset{w}{\operatorname{argmin}} \sum_{i=1}^{R} \frac{(y_i - wx_i)^2}{\sigma_i^2}$$

Assuming i.i.d. and then plugging in equation for Gaussian and simplifying.

$$= \left(w \text{ such that } \sum_{i=1}^{R} \frac{x_i(y_i - wx_i)}{\sigma_i^2} = 0 \right)$$

Setting dLL/dw equal to zero

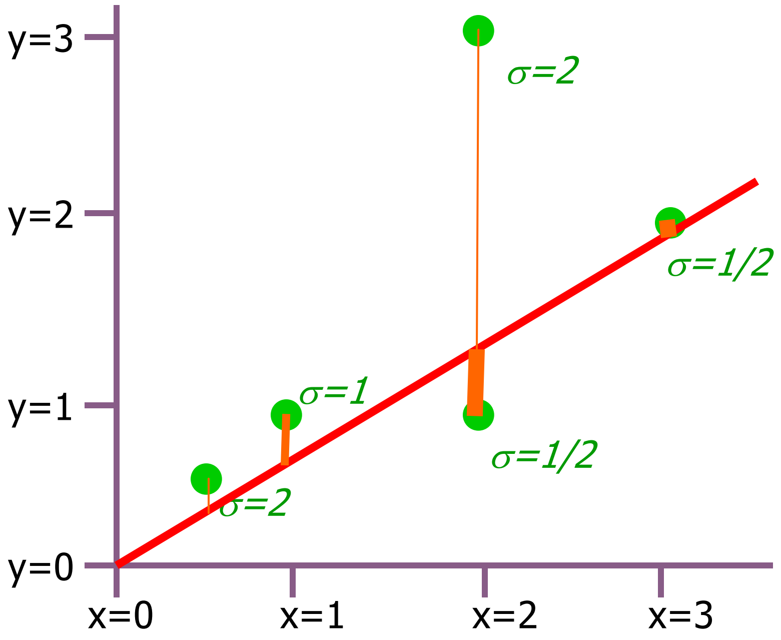
$$= \frac{\left(\sum_{i=1}^{R} \frac{x_i y_i}{\sigma_i^2} \right)}{\left(\sum_{i=1}^{R} \frac{x_i^2}{\sigma_i^2} \right)}$$

Trivial algebra

This is Weighted Regression

- How to minimize weighted sum of squares ?

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{R} \frac{(y_i - wx_i)^2}{\sigma_i^2}$$



where weight for i'th datapoint is $\frac{1}{\sigma_i^2}$

Weighted Multivariate Regression

The max. likelihood ***w*** is ***w*** = $(WX^T WX)^{-1}(WX^T WY)$

$(WX^T WX)$ is an *m* x *m* matrix: i,j'th elt is

$$\sum_{k=1}^{R} \frac{x_{ki} x_{kj}}{\sigma_i^2}$$

$(WX^T WY)$ is an *m*-element vector: i'th elt

$$\sum_{k=1}^{R} \frac{x_{ki} y_k}{\sigma_i^2}$$

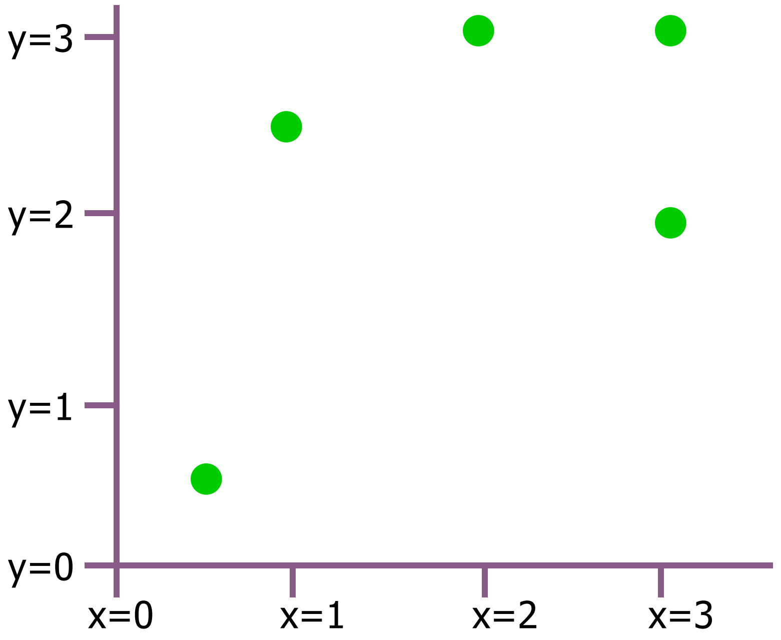
Non-linear Regression

(Digression…)

Non-linear Regression

Suppose y is related to function of x
in that predicted values have a *non-linear dependence* on w:

x_i	y_i
½	½
1	2.5
2	3
3	2
3	3



Assume $y_i \sim N(\sqrt{w + x_i}, \sigma^2)$

What's the MLE estimate of w?

Non-linear MLE estimation

$$\underset{w}{\text{argmax}} \log p(y_1, y_2, \ldots, y_R \mid x_1, x_2, \ldots, x_R, \sigma, w) =$$

$$+ x_i \Big)^2$$

$$+ x_i \qquad = 0 \Big)$$

Assuming i.i.d. and then plugging in equation for Gaussian and simplifying.

Setting dLL/dw equal to zero

Common (but not only) approach:

Numerical Solutions:

- Line Search
- Simulated Annealing
- Gradient Descent
- Conjugate Gradient
- Levenberg Marquart
- Newton's Method

We're down the algebraic toilet

Also, special purpose statistical-optimization-specific tricks such as E.M. (See Gaussian Mixtures lecture for introduction)

So guess what we do?

GRADIENT DESCENT

Goal: Find a local minimum of $f: \Re \rightarrow \Re$

Approach:

1. Start with some value for w
2. GRADIENT DESCENT: $w \leftarrow w - \eta \frac{\partial}{\partial w} \mathrm{f}(w)$
3. Iterate … until bored …

η = LEARNING RATE = small positive number, e.g.
$\eta = 0.05$

Good default value for anything !

QUESTION: Justify the Gradient Descent Rule

Gradient Descent in “m” Dimensions

Given $\mathrm{f}(\mathbf{w}) : \Re^m \rightarrow \Re$

$$\nabla \mathrm{f}(\mathrm{w}) = \begin{pmatrix} \frac{\partial}{\partial w_1}\mathrm{f}(\mathrm{w}) \\ \vdots \\ \frac{\partial}{\partial w_m}\mathrm{f}(\mathrm{w}) \end{pmatrix}$$ points in direction of steepest ascent.

$\left|\nabla \mathrm{f}(\mathrm{w})\right|$ is the gradient in that direction

GRADIENT DESCENT RULE: $\boxed{\mathrm{w} \leftarrow \mathrm{w} - \eta \nabla \mathrm{f}(\mathrm{w})}$

Equivalently $\boxed{w_j \leftarrow w_j - \eta \frac{\partial}{\partial w_j}\mathrm{f}(\mathrm{w})}$ ….where w_j is jth weight

“just like a linear feedback system”

Linear Perceptron

Linear Perceptrons

Multivariate linear models: $\boxed{\text{Out}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}}$

"Training" $\equiv$ minimizing sum-of-squared residuals…

$$\mathrm{E} = \sum_k \left(\text{Out}(\mathrm{x}_k) - y_k\right)^2$$
$$= \sum_k \left(\mathrm{w}^T\mathrm{x}_k - y_k\right)^2$$

by gradient descent…

$\rightarrow$ perceptron training rule

Linear Perceptron Training Rule

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent:
to minimize *E*,
update **w** …

$$\boxed{w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}}$$

So what's $\frac{\partial E}{\partial w_j}$?

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{R} \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

$$= \sum_{k=1}^{R} 2(y_k - \mathbf{w}^T \mathbf{x}_k) \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}_k$$

…*where*

$$\delta_k = y_k - \mathbf{w}^T \mathbf{x}_k$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{m} w_i x_{ki}$$

$$= -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

Linear Perceptron Training Rule

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent:
to minimize *E*,
update **w** ...

$$\boxed{w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}}$$

...where...

$$\frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

$$\boxed{w_j \leftarrow w_j + 2\eta \sum_{k=1}^{R} \delta_k x_{kj}}$$

We frequently neglect the 2 (meaning we halve the learning rate)

The "Batch" perceptron algorithm

1) Randomly initialize weights $w_1\ w_2\ \ldots\ w_m$

2) Get your dataset
   (append 1's to inputs to avoid going thru origin).

3) for $i = 1$ to R $\boxed{\delta_i := y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i}$

4) for $j = 1$ to m $\boxed{w_j \leftarrow w_j + \eta \sum_{i=1}^{R} \delta_i x_{ij}}$

5) if $\sum \delta_i^2$ stops improving then stop.
   Else loop back to 3.

$$\delta_i \leftarrow y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$$

$$w_j \leftarrow w_j + \eta \delta_i x_{ij}$$

A RULE KNOWN BY MANY NAMES

The LMS Rule

The delta rule

The Widrow Hoff rule

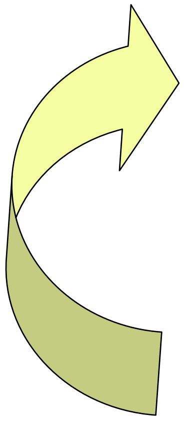
The adaline rule

Classical conditioning

If data is voluminous and arrives fast

If input-output pairs (***x***,*y*) come in very quickly. Then

Don't bother remembering old ones.
Just keep using new ones.



observe (**x**,*y*)

$$\delta \leftarrow y - \mathrm{w}^{\mathrm{T}}\mathrm{x}$$

$$\forall j \;\; w_j \leftarrow w_j + \eta\,\delta\,x_j$$

Gradient Descent vs Matrix Inversion for Linear Perceptrons

GD Advantages (MI disadvantages):

-
-
-

GD Disadvantages (MI advantages):

-
-
-
-
-

Gradient Descent vs Matrix Inversion for Linear Perceptrons

GD Advantages (MI disadvantag):

- Biologically plausible
- With very very many attrib each on (mR). If fewer than m iterations n fa ersion
- More easily parallelizable (or

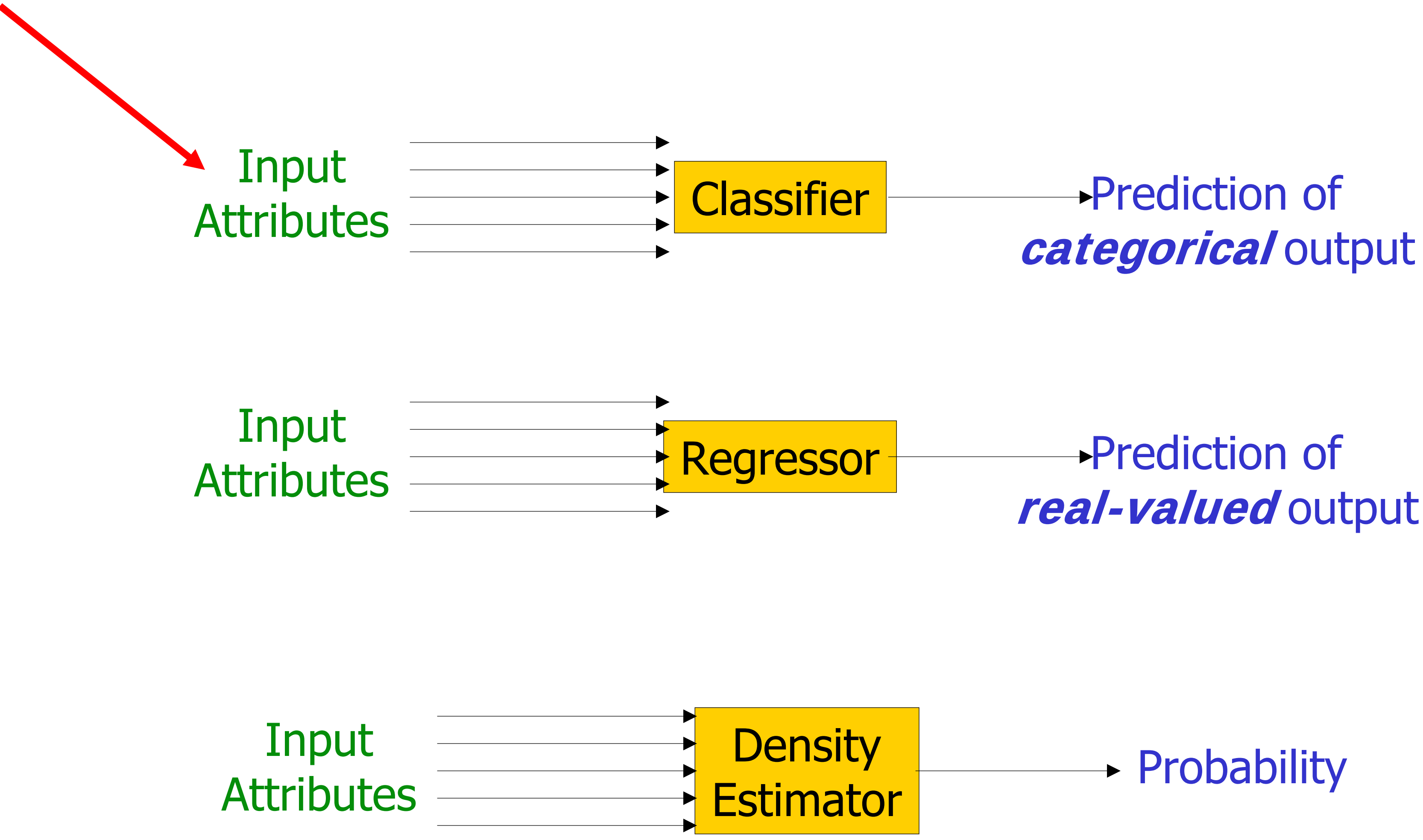
GD Disadvantag

- It's moronic
- It's essentially a X matrix, then solve a set of li
- If *m* is small it's especi then the direct matrix inversion met npossible if you want to be efficient.
- Hard to choose a good lear g rate
- Matrix inversion takes pred table time. You can't be sure when gradient descent will stop.

But we'll
soon see that
GD
has an important extra
trick up its sleeve

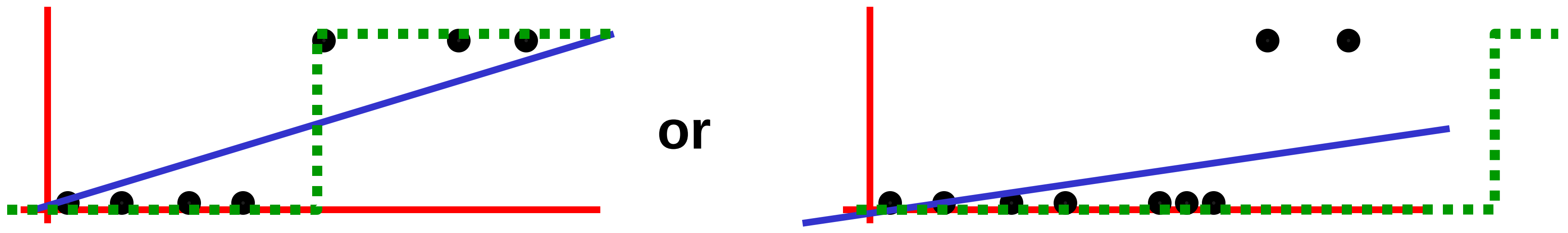
Linear Perceptron
…for Classification

Regression vs Classification



Perceptrons for Classification

What if all outputs are 0's or 1's ?



Blue = Out(x)

Green = Classification

We can do a linear fit.

Our prediction is 0 if out($\boldsymbol{x}$)≤½

1 if out($\boldsymbol{x}$)>½

WHAT'S THE BIG PROBLEM WITH THIS???

Classification with Perceptrons I

Don't minimize $\boxed{\sum \left(y_i - \mathrm{w}^{\mathrm{T}} \mathrm{x}_i \right)^2 \;!}$

Instead, minimize # misclassifications: $\boxed{\sum \left(y_i - \mathrm{Round}\left(\mathrm{w}^{\mathrm{T}} \mathrm{x}_i \right) \right)}$

where Round(x) = $\begin{cases} -1 \text{ if } x<0 \\ 1 \text{ if } x\geq 0 \end{cases}$

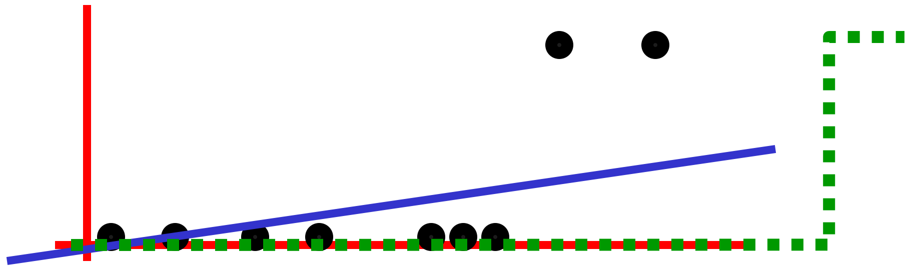
[Assume outputs are +1 & -1, not +1 & 0]

NOTE: CUTE & NON OBVIOUS WHY THIS WORKS!!

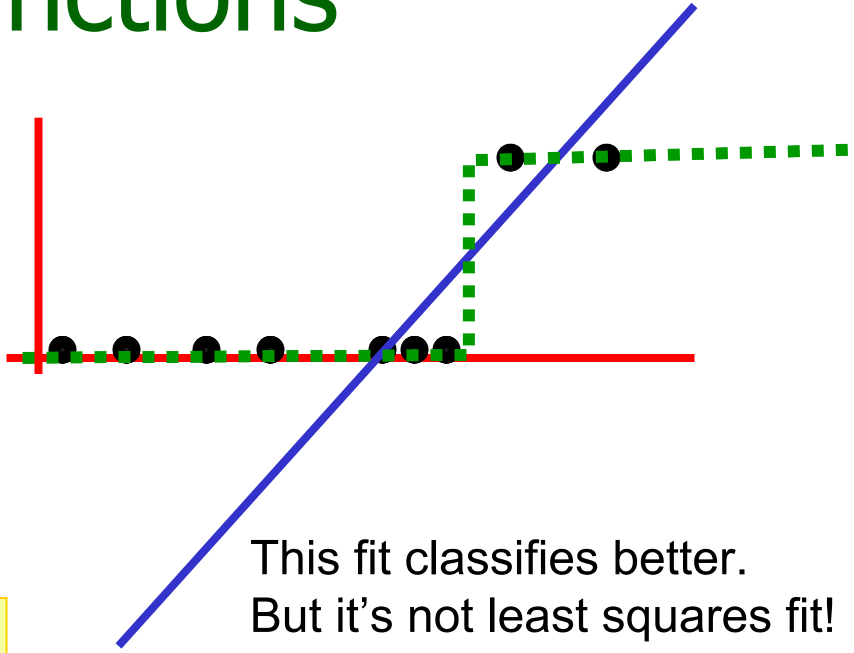
New *gradient descent rule*:

if $(\boldsymbol{x}_i, y_i)$ correctly classed, don't change	
if wrongly predicted as 1	$\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{x}_i$
if wrongly predicted as -1	$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}_i$

Classification with Perceptrons II: Sigmoid Functions



Least squares fit useless



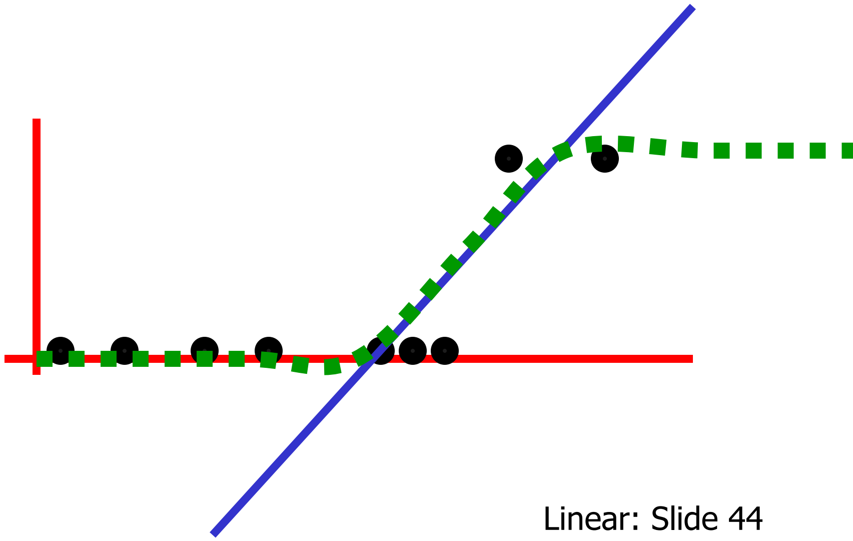
This fit classifies better.
But it's not least squares fit!

SOLUTION:

Instead of Out($\boldsymbol{x}$) = $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}$

We'll use Out($\boldsymbol{x}$) = $g(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x})$

where $g : \Re \rightarrow (0,1)$ is a *squashing* function



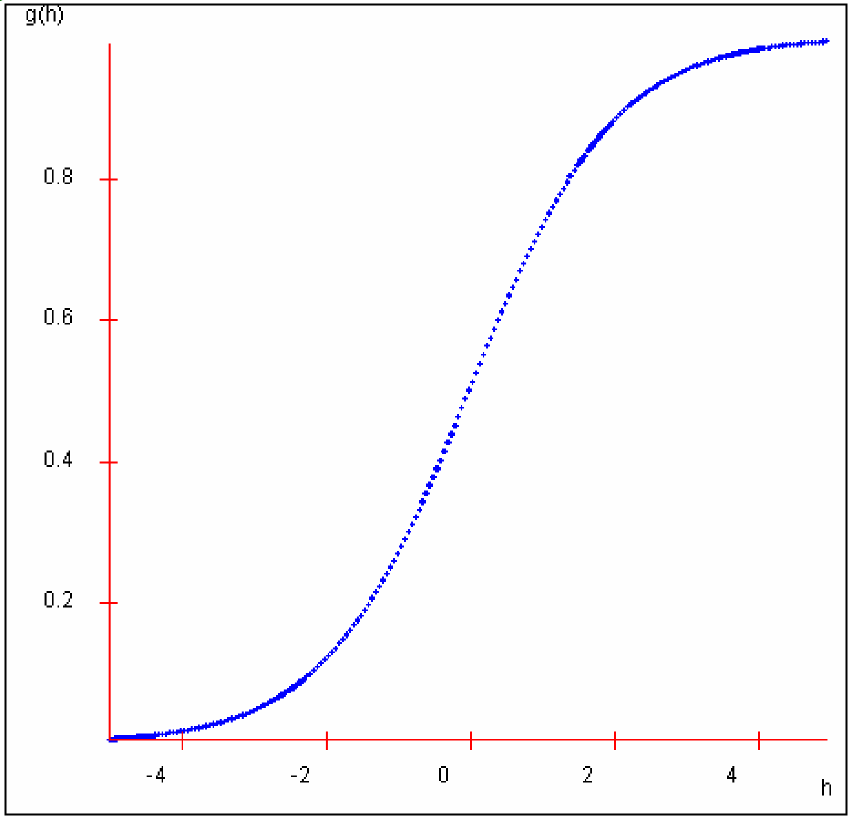
The Sigmoid

$$g(h) = \frac{1}{1+\exp(-h)}$$

Rotating curve 180^{o} centered on *(0,1/2)* produces same curve.

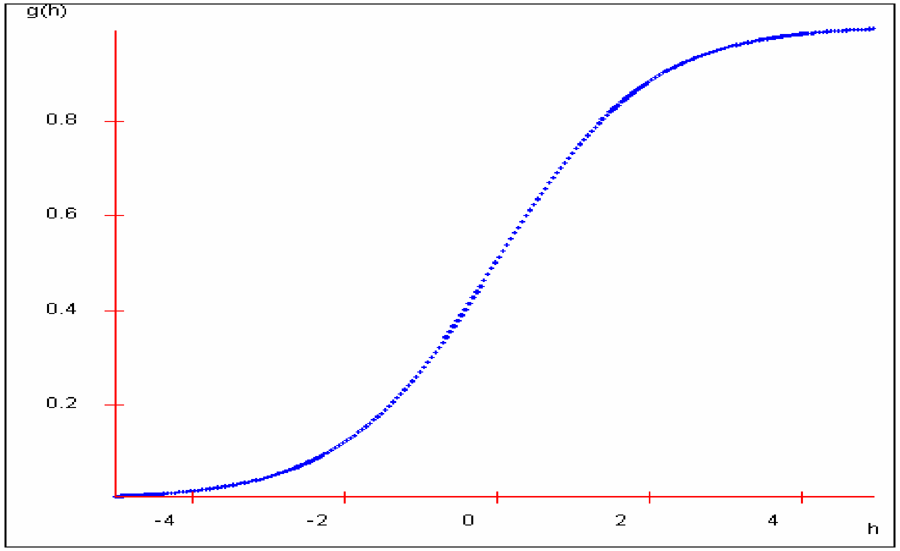
i.e. *g(h) = 1 – g(-h)*

Can you prove this?



The Sigmoid

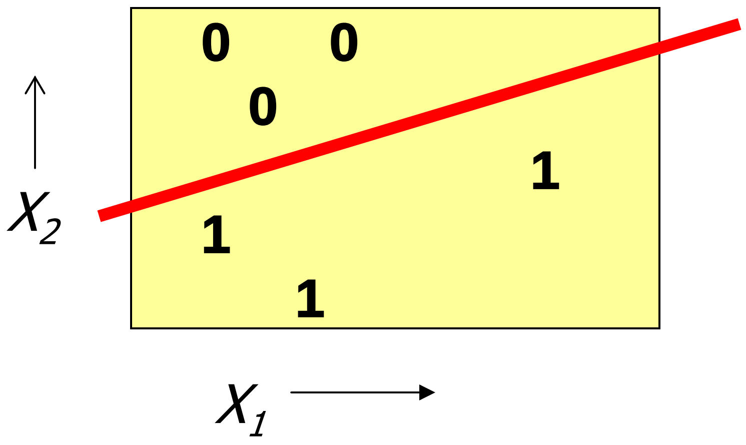
$$g(h) = \frac{1}{1+\exp(-h)}$$



Choose ***w*** to minimize

$$\sum_{i=1}^{R}\left[y_i - \mathrm{Out}(\mathrm{x}_i)\right]^2 = \sum_{i=1}^{R}\left[y_i - g(\mathrm{w}^{\mathrm{T}}\mathrm{x}_i)\right]^2$$

Linear Perceptron Classification Regions



Use model: $\text{Out}(\boldsymbol{x}) = g(\boldsymbol{w}^T \times (1,\boldsymbol{x}))$

$$= g(\, w_0 + w_1 x_1 + w_2 x_2 \,)$$

In diagram… which region classified +1, and which 0 ??

Gradient descent with sigmoid on a perceptron

Note $g'(x) = g(x)\left(1 - g(x)\right)$

Proof: $g(x) = \dfrac{1}{1+e^{-x}}$ so $g'(x) = \dfrac{-e^{-x}}{\left(1+e^{-x}\right)^2}$

$$= \frac{1-1-e^{-x}}{\left(1+e^{-x}\right)^2} = \frac{1}{\left(1+e^{-x}\right)^2} - \frac{1}{1+e^{-x}} = \frac{-1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = -g(x)\left(1-g(x)\right)$$

$$\text{Out(x)} = g\left(\sum_k w_k x_k\right)$$

$$\text{E} = \sum_i \left(y_i - g\left(\sum_k w_k x_{ik}\right)\right)^2$$

$$\frac{\partial \text{E}}{\partial w_j} = \sum_i 2\left(y_i - g\left(\sum_k w_k x_{ik}\right)\right)\left(-\frac{\partial}{\partial w_j} g\left(\sum_k w_k x_{ik}\right)\right)$$

$$= \sum_i -2\left(y_i - g\left(\sum_k w_k x_{ik}\right)\right) g'\left(\sum_k w_k x_{ik}\right)\frac{\partial}{\partial w_j}\sum_k w_k x_{ik}$$

$$= \sum_i -2\delta_i g\left(\text{net}_i\right)\left(1 - g\left(\text{net}_i\right)\right) x_{ij}$$

where $\delta_i = y_i - \text{Out}(\text{x}_i)$ $\quad \text{net}_i = \sum_k w_k x_k$

The sigmoid perceptron update rule:

$$w_j \leftarrow w_j + \eta \sum_{i=1}^{R} \delta_i g_i \left(1 - g_i\right) x_{ij}$$

where $\quad g_i = g\left(\sum_{j=1}^{m} w_j x_{ij}\right)$

$$\delta_i = y_i - g_i$$

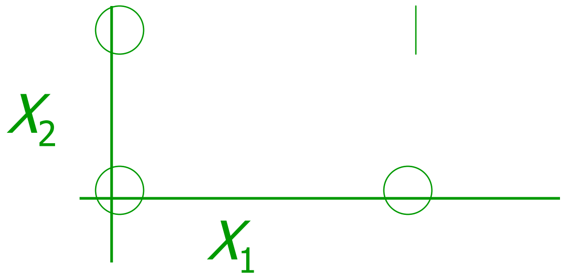
Other Things about Perceptrons

- Invented and popularized by Rosenblatt (1962)
- Even with sigmoid nonlinearity,
  correct convergence is guaranteed !
- Stable behavior for overconstrained and underconstrained problems

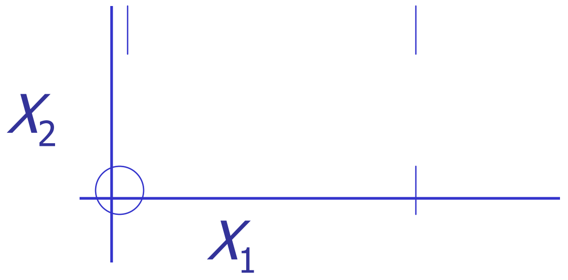
Perceptrons and Boolean Functions

If inputs are all 0's and 1's and outputs are all 0's and 1's…

- Can learn the function $x_1 \wedge x_2$



- Can learn the function $x_1 \vee x_2$.



- Can learn <u>any</u> conjunction of literals, e.g.

$x_1 \wedge \sim x_2 \wedge \sim x_3 \wedge x_4 \wedge x_5$

QUESTION: WHY?

Perceptrons and Boolean Functions

- Can learn any disjunction of literals

  e.g. $x_1 \wedge \sim x_2 \wedge \sim x_3 \wedge x_4 \wedge x_5$

- Can learn majority function

  $$f(x_1, x_2 \ldots x_n) = \begin{cases} 1 \text{ if } n/2 \; x_i\text{'s or more are} = 1 \\ 0 \text{ if less than } n/2 \; x_i\text{'s are} = 1 \end{cases}$$

- What about the exclusive or function?

  $f(x_1, x_2) = x_1 \forall x_2 =$

  $(x_1 \wedge \sim x_2) \vee (\sim x_1 \wedge x_2)$