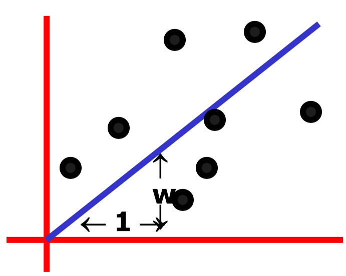
Regression and Classification with Neural Networks

Andrew W. Moore
Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

 Sep 25th, 2001

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

Simplest case: Out(*x*) = *wx* for some unknown *w*.

Given the data, we can estimate *w*.

1-parameter linear regression

Assume that the data is formed by

$$y_i = wx_i + \text{noise}_i$$

where…

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

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

- mean wx
- variance σ^2

Bayesian Linear Regression

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

We have a set of datapoints (x_1,y_1) (x_2,y_2) … (x_n,y_n) which are EVIDENCE about w.

We want to infer w from the data.

$$P(w|x_1, x_2, x_3,\ldots x_n, y_1, y_2\ldots y_n)$$

- You can use BAYES rule to work out a posterior distribution for w given the data.
- Or you could do Maximum Likelihood Estimation

Maximum likelihood estimation of *w*

Asks the question:

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

<=>

For what *w* is

$P(y_1, y_2 \ldots y_n \mid x_1, x_2, x_3, \ldots x_n, w)$ maximized?

<=>

For what *w* is

$$\prod_{i=1}^{n} P(y_i \mid w, x_i) \text{ maximized?}$$

For what *w* is

$$\prod_{i=1}^{n} P(y_i \mid w, x_i) \quad \text{maximized?}$$

For what *w* is

$$\prod_{i=1}^{n} \exp\left(-\frac{1}{2}\left(\frac{y_i - wx_i}{\sigma}\right)^2\right) \text{ maximized?}$$

For what *w* is

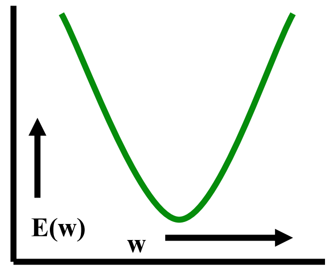
$$\sum_{i=1}^{n} -\frac{1}{2}\left(\frac{y_i - wx_i}{\sigma}\right)^2 \quad \text{maximized?}$$

For what *w* is

$$\sum_{i=1}^{n} \left(y_i - wx_i\right)^2 \quad \text{minimized?}$$

Linear Regression

The maximum likelihood *w* is the one that minimizes sum-of-squares of residuals



$$E = \sum_i (y_i - w x_i)^2$$

$$= \sum_i y_i^2 - \left(2 \sum x_i y_i\right) w + \left(\sum x_i^2\right) w^2$$

We want to minimize a quadratic function of *w*.

Linear Regression

Easy to show the sum of squares is minimized when

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

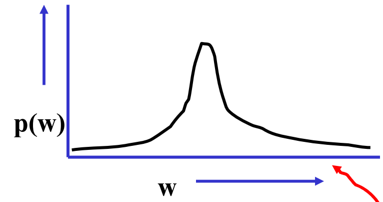
The maximum likelihood model is $\mathrm{Out}(x) = wx$

We can use it for prediction

Linear Regression

Easy to show the sum of squares is minimized when

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



The maximum likelihood model is $\mathrm{Out}(x) = wx$

We can use it for prediction

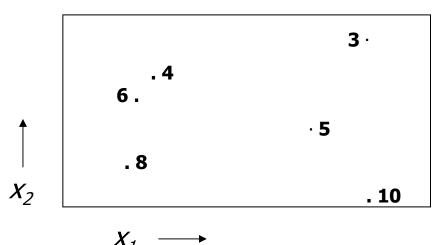
Note: In Bayesian stats you'd have ended up with a prob dist of *w*

And predictions would have given a prob dist of expected output

Often useful to know your confidence. Max likelihood can give some kinds of confidence too.

Multivariate Regression

What if the inputs are vectors?



2-d input example

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

Write matrix X and Y thus:

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

(there are R datapoints. Each input has m components)

The linear regression model assumes a vector $\boldsymbol{w}$ such that

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

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

Multivariate Regression

Write matrix X and Y thus:

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

(there are R datapoints. Each input

IMPORTANT EXERCISE: PROVE IT !!!!!

The linear regression model assumes a vector $\boldsymbol{w}$ such that

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

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

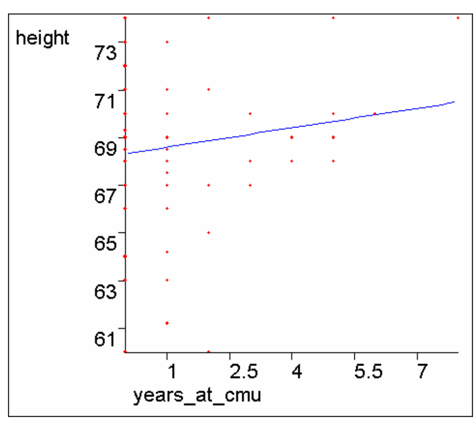
Multivariate Regression (con't)

The max. likelihood $\mathbf{w}$ is $\mathbf{w} = (X^TX)^{-1}(X^TY)$

X^TX is an m x m matrix: i,j'th elt is $\sum_{k=1}^{R} x_{ki}x_{kj}$

X^TY is an m-element vector: i'th elt $\sum_{k=1}^{R} x_{ki}y_k$

What about a constant term?



We may expect linear data that does not go through the origin.

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

Can you guess??

The constant term

- The trick is to create a fake input "X_0" that always takes the value 1

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

Before:

$Y = w_1X_1 + w_2X_2$

…has to be a poor model

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

After:

$Y = w_0X_0 + w_1X_1 + w_2X_2$

$= w_0 + w_1X_1 + w_2X_2$

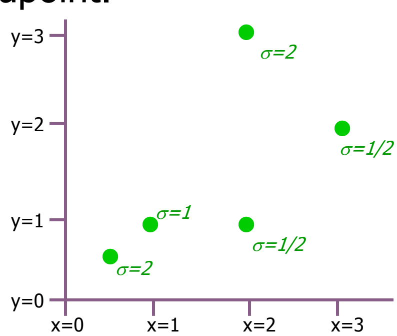
…has a fine constant term

In this example, You should be able to see the MLE w_0, w_1 and w_2 by inspection

Regression with varying noise

- Suppose you know the variance of the 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

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

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

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

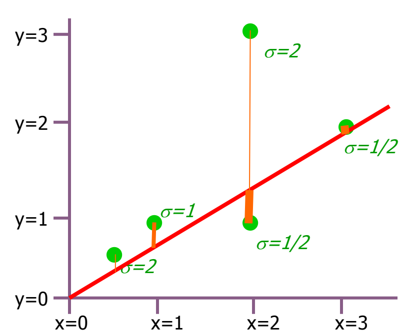
$$\frac{\left(\sum_{i=1}^{R} \frac{x_i y_i}{\sigma_i^2} \right)}{\left(\sum_{i=1}^{R} \frac{x_i^2}{\sigma_i^2} \right)}$$

Trivial algebra

This is Weighted Regression

- We are asking to minimize the weighted sum of squares

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



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

Weighted Multivariate Regression

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

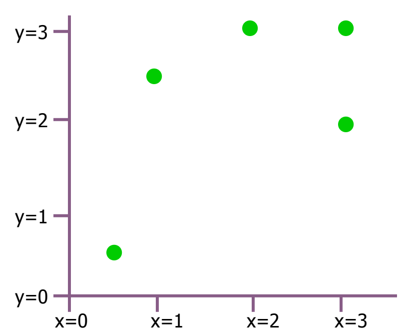
(WX^TWX) is an *m* x *m* matrix: i,j'th elt is $$\sum_{k=1}^{R} \frac{x_{ki}x_{kj}}{\sigma_i^2}$$

(WX^TWY) is an *m*-element vector: i'th elt $$\sum_{k=1}^{R} \frac{x_{ki}y_k}{\sigma_i^2}$$

Non-linear Regression

- Suppose you know that y is related to a function of x in such a way that the predicted values have a non-linear dependence on w, e.g:

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

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

$$\underset{w}{\text{argmin}} \sum_{i=1}^{R} \left(y_i - \sqrt{w + x_i} \right)^2 =$$

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

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

Setting dLL/dw equal to zero

Non-linear MLE estimation

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

$$\underset{w}{\text{argmin}} \sum_{i=1}^{R} \left(y_i - \sqrt{w + x_i} \right)^2 =$$

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

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

Setting dLL/dw equal to zero

We're down the algebraic toilet

So guess what we do?

Non-linear MLE estimation

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

$$\ldots w + x_i\Big)^2 =$$

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

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

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

Suppose we have a scalar function $f(w): \Re \rightarrow \Re$

We want to find a local minimum.
Assume our current weight is *w*

GRADIENT DESCENT RULE: $w \leftarrow w - \eta \frac{\partial}{\partial w} f(w)$

η is called the LEARNING RATE. A small positive number, e.g. η = 0.05

GRADIENT DESCENT

Suppose we have a scalar function $f(w): \Re \rightarrow \Re$

We want to find a local minimum.

Assume our current weight is *w*

GRADIENT DESCENT RULE: $$w \leftarrow w - \eta \frac{\partial}{\partial w} f(w)$$

η is called the LEARNING RATE. A small positive number, e.g. η = 0.05

Recall Andrew's favorite default value for anything

QUESTION: Justify the Gradient Descent Rule

Gradient Descent in "m" Dimensions

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

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

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

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

Equivalently

$$w_j \leftarrow w_j - \eta \frac{\partial}{\partial w_j} f(\mathrm{w})$$

....where w_j is the *j*th weight

"just like a linear feedback system"

What's all this got to do with Neural Nets, then, eh??

For supervised learning, neural nets are also models with vectors of **w** parameters in them. They are now called weights.

As before, we want to compute the weights to minimize sum-of-squared residuals.

Which turns out, under "Gaussian i.i.d noise" assumption to be max. likelihood.

Instead of explicitly solving for max. likelihood weights, we use **GRADIENT DESCENT** to **SEARCH** for them.

"Why?" you ask, a querulous expression in your eyes.
"Aha!!" I reply: "We'll see later."

Linear Perceptrons

They are multivariate linear models:

$$\text{Out}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

And "training" consists of minimizing sum-of-squared residuals by gradient descent.

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

QUESTION: Derive the perceptron training rule.

Linear Perceptron Training Rule

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

Gradient descent tells us we should update **w** thusly if we wish to minimize *E:*

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

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

Linear Perceptron Training Rule

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

Gradient descent tells us we should update **w** thusly if we wish to minimize *E:*

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

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 tells us we should update **w** thusly if we wish to minimize *E:*

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

...where...

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

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

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

The "Batch" perceptron algorithm

1) Randomly initialize weights w_1 w_2 … w_m

2) Get your dataset (append 1's to the inputs if you don't want to go through the origin).

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

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

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

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

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

A RULE KNOWN BY MANY NAMES

The LMS Rule

The delta rule

The Widrow Hoff rule

Classical conditioning

The adaline rule

If data is voluminous and arrives fast

Input-output pairs ($\mathbf{x}$,y) come streaming in very quickly. THEN

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

observe ($\mathbf{x}$,y)

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

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

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 disadvantages):

- Biologically plausible
- With very very many attributes each iteration costs only O(mR). If fewer than m iterations needed we've beaten Matrix Inversion
- More easily parallelizable (or implementable in wetware)?

GD Disadvantages (MI advantages):

- It's moronic
- It's essentially a slow implementation of a way to build the XTX matrix and then solve a set of linear equations
- If m is small it's especially outageous. If m is large then the direct matrix inversion method gets fiddly but not impossible if you want to be efficient.
- Hard to choose a good learning rate
- Matrix inversion takes predictable time. You can't be sure when gradient descent will stop.

Gradient Descent vs Matrix Inversion for Linear Perceptrons

GD Advantages (MI disadvantag):

- Biologically plausible
- With very very many attrib each i onl (mR). If fewer than m iterations nee rsion
- More easily parallelizable (o

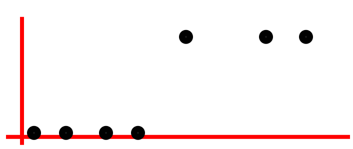
GD Disadvantag

- It's moronic
- It's essentially a e XTX matrix and then solve a se
- If m is small it's especi hen the direct matrix inversion me possi e 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.

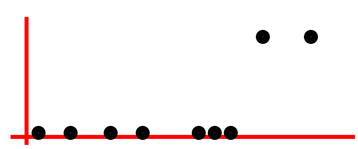
But we'll
soon see that
GD
has an important extra
trick up its sleeve

Perceptrons for Classification

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



or



We can do a linear fit.

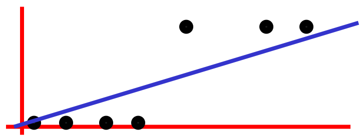
Our prediction is 0 if out($\boldsymbol{x}$)≤1/2

1 if out($\boldsymbol{x}$)>1/2

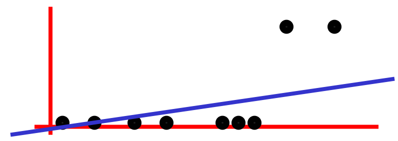
WHAT'S THE BIG PROBLEM WITH THIS???

Perceptrons for Classification

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



or



Blue = Out(x)

We can do a linear fit.

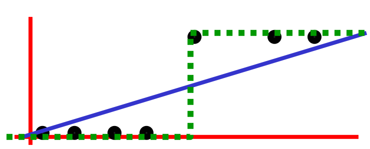
Our prediction is 0 if out($\boldsymbol{x}$)≤½

1 if out($\boldsymbol{x}$)>½

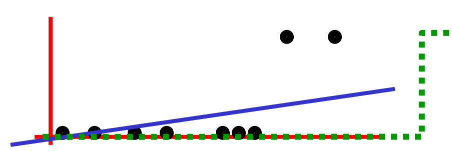
WHAT'S THE BIG PROBLEM WITH THIS???

Perceptrons for Classification

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



or



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

Classification with Perceptrons I

Don't minimize $\sum (y_i - \mathrm{w}^T \mathrm{x}_i)^2$.

Minimize number of misclassifications instead. [Assume outputs are +1 & -1, not +1 & 0]

$$\sum \left(y_i - \mathrm{Round}\left(\mathrm{w}^T \mathrm{x}_i\right)\right)$$

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

NOTE: CUTE & NON OBVIOUS WHY THIS WORKS!!

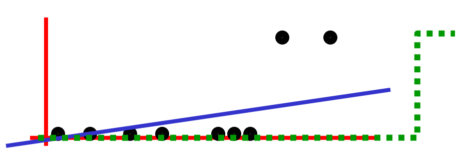
The gradient descent rule can be changed to:

if ($\boldsymbol{x}_i$,y_i) correctly classed, don't change

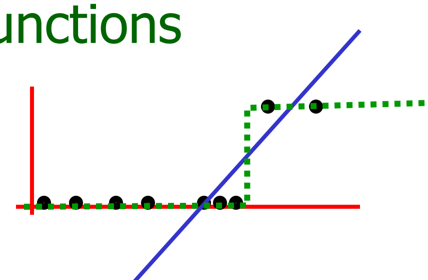
if wrongly predicted as 1 $\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{x}_i$

if wrongly predicted as -1 $\quad \boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}_i$

Classification with Perceptrons II: Sigmoid Functions

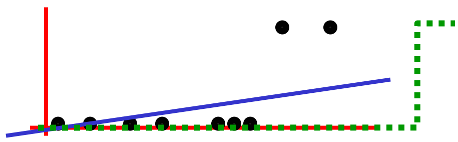


Least squares fit useless

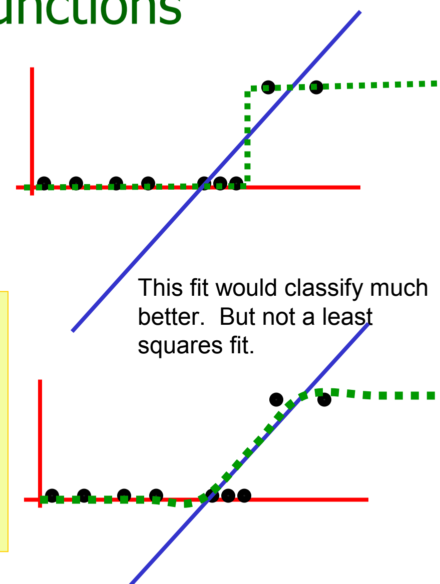


This fit would classify much better. But not a least squares fit.

Classification with Perceptrons II: Sigmoid Functions



Least squares fit useless



SOLUTION:

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

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

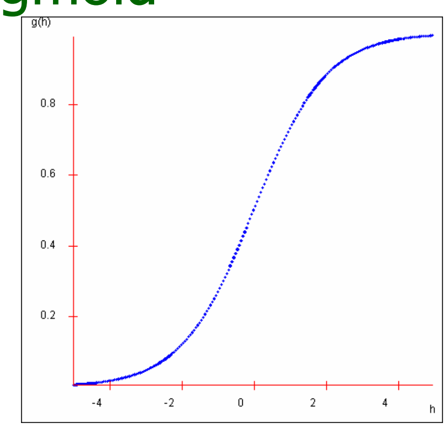
where $g(x): \Re \rightarrow (0,1)$ is a squashing function

The Sigmoid

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

Note that if you rotate this curve through 180º centered on *(0,1/2)* you get the same curve.

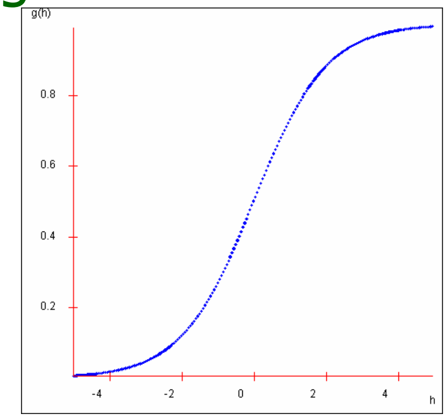
i.e. *g(h)=1-g(-h)*



Can you prove this?

The Sigmoid

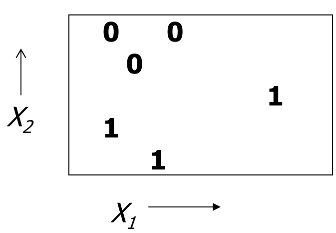
$$g(h) = \frac{1}{1+\exp(-h)}$$



Now we choose ***w*** to minimize

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

Linear Perceptron Classification Regions



We'll use the model $\quad \text{Out}(\boldsymbol{x}) = g(\boldsymbol{w}^T(\boldsymbol{x},1))$

$$= g(w_1x_1 + w_2x_2 + w_0)$$

Which region of above diagram classified with +1, and which with 0 ??

Gradient descent with sigmoid on a perceptron

First, notice $g'(x) = g(x)(1 - g(x))$

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

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

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

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

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

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

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

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:

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

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

$$\delta_i = y_i - g_i$$

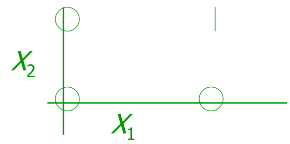
Other Things about Perceptrons

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

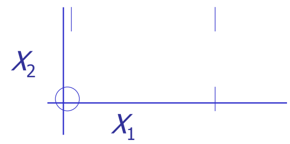
Perceptrons and Boolean Functions

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?

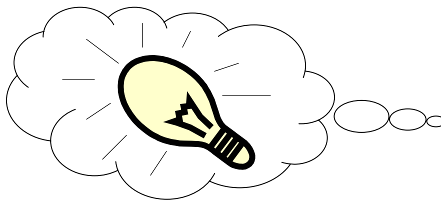
  $f(x_1, x_2) = x_1 \forall x_2 =$

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

Multilayer Networks

The class of functions representable by perceptrons is limited

$$\text{Out}(\mathrm{x}) = g\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) = g\left(\sum_j w_j x_j\right)$$



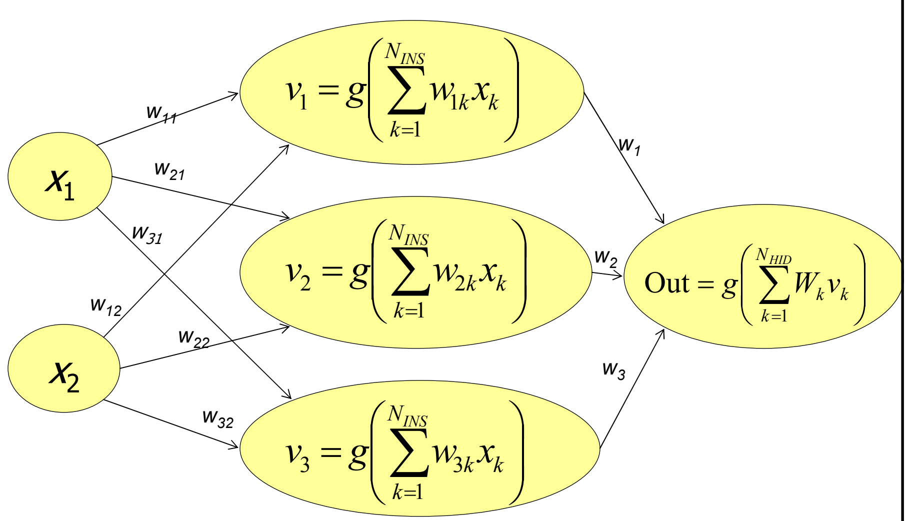
Use a wider representation !

$$\text{Out}(\mathrm{x}) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_{jk}\right)\right)$$

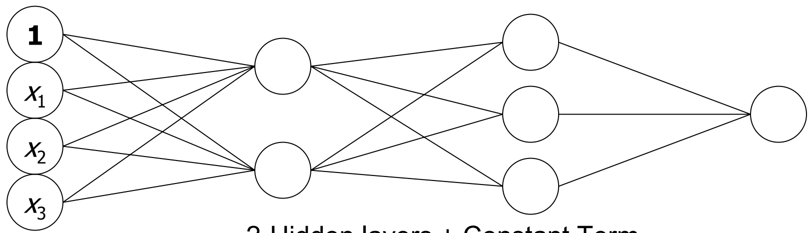
This is a nonlinear function
Of a linear combination
Of non linear functions
Of linear combinations of inputs

A 1-HIDDEN LAYER NET

$N_{INPUTS} = 2$ $N_{HIDDEN} = 3$

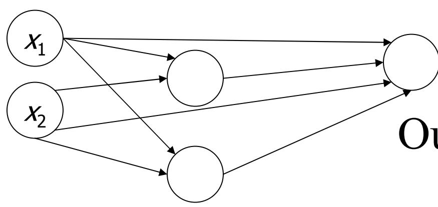


OTHER NEURAL NETS



2-Hidden layers + Constant Term

"JUMP" CONNECTIONS



$$\text{Out} = g\left(\sum_{k=1}^{N_{INS}} w_{0k} x_k + \sum_{k=1}^{N_{HID}} W_k v_k \right)$$

Backpropagation

$$\text{Out(x)} = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_k \right) \right)$$

Find a set of weights $\{W_j\}, \{w_{jk}\}$

to minimize

$$\sum_i \left(y_i - \text{Out}(\text{x}_i) \right)^2$$

by gradient descent.

That's it!
That's the backpropagation algorithm.

Backpropagation Convergence

Convergence to a global minimum is <u>not</u> guaranteed.

•In practice, this is not a problem, apparently.

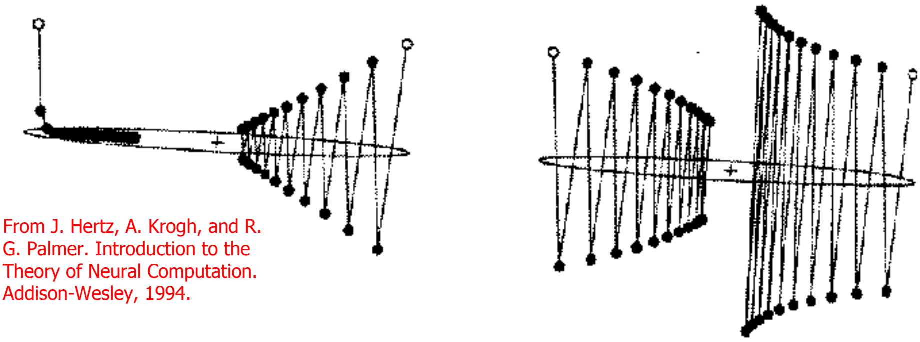
Tweaking to find the right number of hidden units, or a useful learning rate η, is more hassle, apparently.

IMPLEMENTING BACKPROP: 🗁 Differentiate Monster sum-square residual 🗎 Write down the Gradient Descent Rule 🗏 It turns out to be easier & computationally efficient to use lots of local variables with names like h_j o_k v_j net_i etc…

Choosing the learning rate

- This is a subtle art.
- Too small: can take days instead of minutes to converge
- Too large: diverges (MSE gets larger and larger while the weights increase and usually oscillate)
- Sometimes the "just right" value is hard to find.

Learning-rate problems



From J. Hertz, A. Krogh, and R. G. Palmer. Introduction to the Theory of Neural Computation. Addison-Wesley, 1994.

FIGURE 5.10 Gradient descent on a simple quadratic surface (the left and right parts are copies of the same surface). Four trajectories are shown, each for 20 steps from the open circle. The minimum is at the + and the ellipse shows a constant error contour. The only significant difference between the trajectories is the value of η, which was 0.02, 0.0476, 0.049, and 0.0505 from left to right.

Improving Simple Gradient Descent

Momentum

Don't just change weights according to the current datapoint.

Re-use changes from earlier iterations.

Let $\Delta\mathbf{w}(t)$ = weight changes at time t.

Let $-\eta \frac{\partial E}{\partial w}$ be the change we would make with regular gradient descent.

Instead we use

$$\Delta\mathbf{w}(t+1) = -\eta \frac{\partial E}{\partial \mathbf{w}} + \alpha \Delta\mathbf{w}(t)$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$

momentum parameter

Momentum damps oscillations.

A hack? Well, maybe.

Momentum illustration

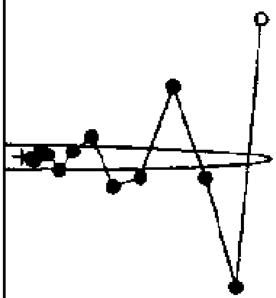


FIGURE 6.3 Gradient descent on the simple quadratic surface of Fig. 5.10. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

Improving Simple Gradient Descent

Newton's method

$$E(\mathbf{w}+\mathbf{h}) = E(\mathbf{w}) + \mathbf{h}^T \frac{\partial E}{\partial \mathbf{w}} + \frac{1}{2}\mathbf{h}^T \frac{\partial^2 E}{\partial \mathbf{w}^2}\mathbf{h} + O(|\mathbf{h}|^3)$$

If we neglect the $O(h^3)$ terms, this is a ***quadratic form***

Quadratic form fun facts:

If $y = c + \boldsymbol{b}^T \boldsymbol{x} - 1/2\, \boldsymbol{x}^T \boldsymbol{A}\, \boldsymbol{x}$

And if $\boldsymbol{A}$ is SPD

Then

$\boldsymbol{x}^{opt} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is the value of $\boldsymbol{x}$ that maximizes y

Improving Simple Gradient Descent

Newton's method

$$E(\mathbf{w}+\mathbf{h}) = E(\mathbf{w}) + \mathbf{h}^T \frac{\partial E}{\partial \mathbf{w}} + \frac{1}{2}\mathbf{h}^T \frac{\partial^2 E}{\partial \mathbf{w}^2}\mathbf{h} + O(|\mathbf{h}|^3)$$

If we neglect the $O(h^3)$ terms, this is a ***quadratic form***

$$\mathbf{w} \leftarrow \mathbf{w} - \left[\frac{\partial^2 E}{\partial \mathbf{w}^2}\right]^{-1} \frac{\partial E}{\partial \mathbf{w}}$$

This should send us directly to the global minimum if the function is truly quadratic.

And it might get us close if it's locally quadraticish

Improving Simple Gradient Descent

Newton's method

$$E(\mathbf{w}+\mathbf{h}) = E(\mathbf{w}) + \mathbf{h}^T \frac{\partial E}{\partial \mathbf{w}} + \frac{1}{2}\mathbf{h}^T \frac{\partial^2 E}{\partial \mathbf{w}^2}\mathbf{h} + O(|\mathbf{h}|^3)$$

I... neglect the $O(h^3)$ terms, this is a ***quadratic form***

BUT (and it's a big but)…

That second derivative matrix can be expensive and fiddly to compute.

If we're not already in the quadratic bowl, we'll go nuts.

...mum if the

functio...

And it might get us clos... ...aticish

Improving Simple Gradient Descent

Conjugate Gradient

Another method which attempts to exploit the "local quadratic bowl" assumption

But does so while only needing to use $\frac{\partial E}{\partial \mathbf{w}}$

and not $\frac{\partial^2 E}{\partial \mathbf{w}^2}$

It is also more stable than Newton's method if the local quadratic bowl assumption is violated.

It's complicated, outside our scope, but it often works well. More details in Numerical Recipes in C.

BEST GENERALIZATION

Intuitively, you want to use the smallest, simplest net that seems to fit the data.

HOW TO FORMALIZE THIS INTUITION?

1. Don't. Just use intuition
2. Bayesian Methods Get it Right
3. Statistical Analysis explains what's going on
4. Cross-validation

Discussed in the next lecture

What You Should Know

- How to implement multivariate Least-squares linear regression.
- Derivation of least squares as max. likelihood estimator of linear coefficients
- The general gradient descent rule

What You Should Know

- Perceptrons
  - → Linear output, least squares
  - → Sigmoid output, least squares
- Multilayer nets
  - → The idea behind back prop
  - → Awareness of better minimization methods
- Generalization. What it means.

APPLICATIONS

To Discuss:

- What can non-linear regression be useful for?
- What can neural nets (used as non-linear regressors) be useful for?
- What are the advantages of N. Nets for nonlinear regression?
- What are the disadvantages?

Other Uses of Neural Nets…

- Time series with recurrent nets
- Unsupervised learning (clustering principal components and non-linear versions thereof)
- Combinatorial optimization with Hopfield nets, Boltzmann Machines
- Evaluation function learning (in reinforcement learning)