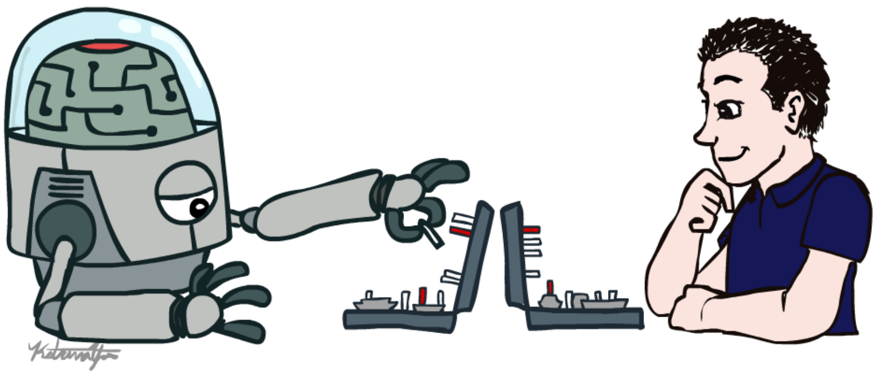
CSE 3521:
Introduction to Artificial Intelligence



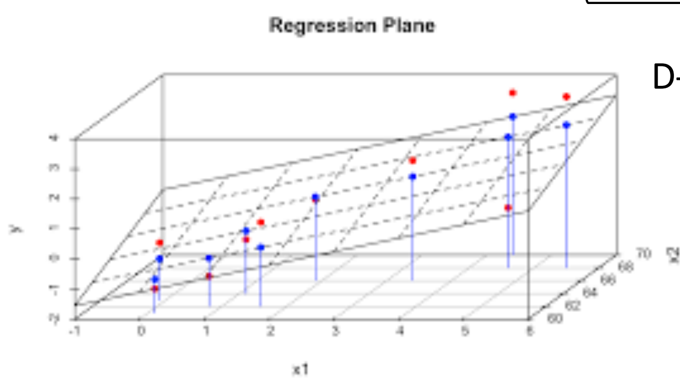
[Many slides are adapted from the UC Berkeley. CS188 Intro to AI at UC Berkeley and previous CSE 3521 course at OSU.]

THE OHIO STATE UNIVERSITY

Regression (curve fitting): what model?

- Relationship (function):
  - Linear
    - $f(x) = a\,x + b$ or $f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = \underbrace{w[1]x[1] + \;\dots\dots + w[D]}_{\text{D-dimensional input}}x[D] + b$

1-dimensional input



  - Non-linear
    - $f(x) = a\,x^2 + \; bx + c$
- Parameter Estimation
  - Given data points and desired labels/target values $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$
  - Find values for parameters a, b, c, $\boldsymbol{w}$, etc.
  - Aim to find parameters that make $f(\boldsymbol{x}_n)$ as close as possible to y_n

Regression (curve fitting): close?

- Difference or error between predictions and labels
  - Total error: error over all training data instances
- Different types of total error
  - Sum of Absolute Error (SAE)

$$E_1 = \sum_i |y_i - f(\boldsymbol{x}_i)|$$

  - Maximum Error

$$E_\infty = \max_i |y_i - f(\boldsymbol{x}_i)|$$

  - Sum of Squared Error (SSE) or residual sum of squares (RSS)

$$E_2 = \sum_i [y_i - f(\boldsymbol{x}_i)]^2$$

Linear least squares

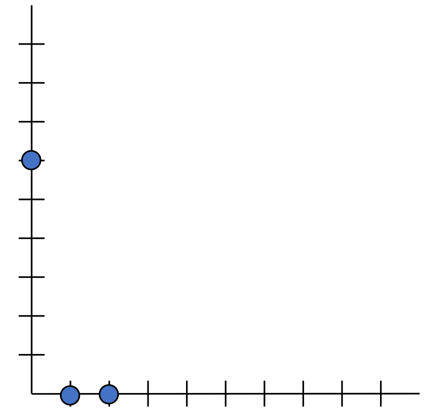
- f is a linear function (linear combination of feature variables)
  - Linear in parameters: The function is a sum of terms $x[d]$, where parameters are *only* coefficients of those terms

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = w[1]x[1] + \dots \dots + w[D]x[D] + b$$

  - Thus, only works when we expect relationship is a line/plane/hyperplane
- Minimize sum of squared error
- Why SSE?
  - Short answer: The math is easier!
  - Other answers: You will see more after probabilistic modeling
- Estimation of model parameters derived from
  - Calculus
  - Linear Algebra

Example: 1-dimensional data

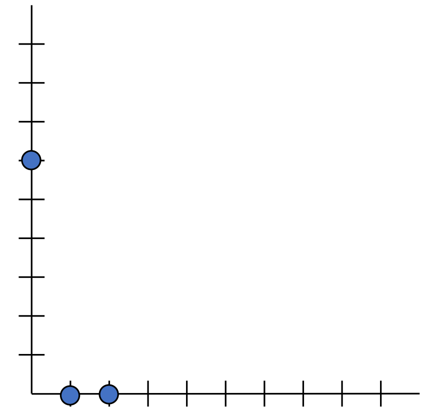
- Find a straight line through points (0,6), (1,0), and (2,0)
  - Model equation: y=*w*x + *b*
- Need values for parameters *w*, *b* that satisfy



$$6 = w \cdot 0 + b$$
$$0 = w \cdot 1 + b$$
$$0 = w \cdot 2 + b$$

Linear system of equations

$$y = w \cdot x + b$$
$$6 = w \cdot 0 + b$$
$$0 = w \cdot 1 + b$$
$$0 = w \cdot 2 + b$$



$$\begin{bmatrix}6\\0\\0\end{bmatrix} = w \cdot \begin{bmatrix}0\\1\\2\end{bmatrix} + b \cdot \begin{bmatrix}1\\1\\1\end{bmatrix} \longrightarrow \begin{bmatrix}6\\0\\0\end{bmatrix} = \begin{bmatrix}0 & 1\\1 & 1\\2 & 1\end{bmatrix}\begin{bmatrix}w\\b\end{bmatrix}$$

Expand x with 1

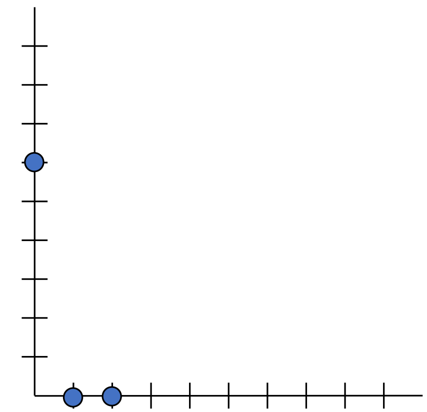
Combine w with b

$$\mathbf{y} = \tilde{\boldsymbol{X}}^T \, \tilde{\boldsymbol{w}}$$

$\tilde{\boldsymbol{w}} = [w, b]$ <u>unsolvable</u> for these actual points! But…

Minimizing the error

- Find a set of parameters (*w*, *b*) that "best" fit the data
  - Minimize the difference between $\mathbf{y}$ and $\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}}$
  - $\begin{bmatrix}6\\0\\0\end{bmatrix} - \begin{bmatrix}0 & 1\\1 & 1\\2 & 1\end{bmatrix}\begin{bmatrix}w\\b\end{bmatrix}$

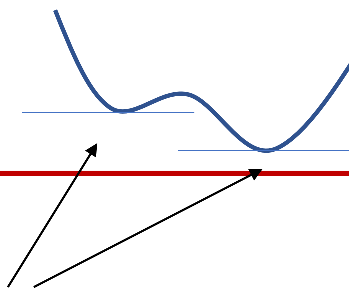


- "Least squares" approximation
  - Look for the best $\widetilde{\boldsymbol{w}}$ that makes sum of the squared error *(SSE)* as small as possible

$$E = \sum_i (wx_i + b - y_i)^2 = \sum_i \left([w, b]\begin{bmatrix}x_i\\1\end{bmatrix} - y_i\right)^2 = \sum_i (\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}}_i - y_i)^2 = \sum_i \left({\widetilde{\boldsymbol{x}}_i}^T\widetilde{\boldsymbol{w}} - y_i\right)^2$$

$$= \left[\left({\widetilde{\boldsymbol{x}}_1}^T\widetilde{\boldsymbol{w}} - y_1\right), \dots\dots, \left({\widetilde{\boldsymbol{x}}_N}^T\widetilde{\boldsymbol{w}} - y_N\right)\right]\begin{bmatrix}{\widetilde{\boldsymbol{x}}_1}^T\widetilde{\boldsymbol{w}} - y_1\\ \vdots \\ {\widetilde{\boldsymbol{x}}_N}^T\widetilde{\boldsymbol{w}} - y_N\end{bmatrix} = \left\|\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - \mathbf{y}\right\|_2^2$$

Minimizing the error

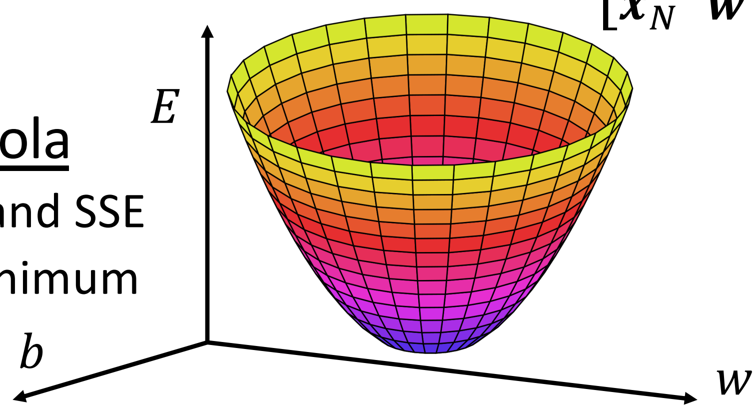


- Many error functions are minimized by calculus
  - Derivatives (gradients) are zero when function at a (local) minimum
- Error function

$$E = \sum_i (wx_i + b - y_i)^2 = \sum_i \left([w, b] \begin{bmatrix} x_i \\ 1 \end{bmatrix} - y_i \right)^2 = \sum_i (\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i - y_i)^2$$

$$= [(\widetilde{\boldsymbol{x}}_1{}^T \widetilde{\boldsymbol{w}} - y_1), \dots \dots, (\widetilde{\boldsymbol{x}}_N{}^T \widetilde{\boldsymbol{w}} - y_N)] \begin{bmatrix} \widetilde{\boldsymbol{x}}_1{}^T \widetilde{\boldsymbol{w}} - y_1 \\ \vdots \\ \widetilde{\boldsymbol{x}}_N{}^T \widetilde{\boldsymbol{w}} - y_N \end{bmatrix} = \left\| \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y} \right\|_2^2$$

- *E* has the shape of a <u>parabola</u>
  - Due to linear the function and SSE
  - Local minimum = global minimum



Minimizing the error

- We have two unknowns (*w*, *b*), therefore we have two derivatives
  - Both derivatives are zero at the function minimum
- Compute the <u>partial derivatives</u> and set to 0

$$\frac{\partial E}{\partial w} = \frac{\partial}{\partial w}\sum_i (w \cdot x_i + b - y_i)^2 = 0$$
$$= 2\sum_i (w \cdot x_i + b - y_i) \cdot x_i = 0$$

$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b}\sum_i (w \cdot x_i + b - y_i)^2 = 0$$
$$= 2\sum_i (w \cdot x_i + b - y_i) = 0$$

Minimizing the error

- Substitute in data

Recall:
$(0, 6), (1, 0), (2, 0)$

$$\frac{\partial E}{\partial w} = \cancel{2}\sum_i (w \cdot x_i + b - y_i) \cdot x_i = 0$$
$$= (w \cdot 0 + b - 6) \cdot 0 + (w \cdot 1 + b - 0) \cdot 1 + (w \cdot 2 + b - 0) \cdot 2 = 0$$
$$= 5w + 3b = 0$$

$$\frac{\partial E}{\partial b} = \cancel{2}\sum_i (w \cdot x_i + b - y_i) = 0$$
$$= (w \cdot 0 + b - 6) + (w \cdot 1 + b - 0) + (w \cdot 2 + b - 0) = 0$$
$$= 3w + 3b - 6 = 0$$
$$3w + 3b = 6$$

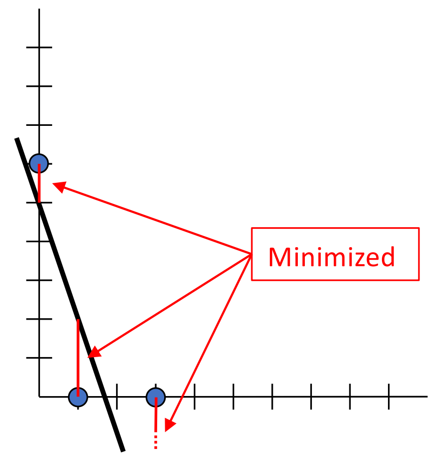
Minimizing the error

- Solve for model parameters

$$5w + 3b = 0$$

$$3w + 3b = 6$$

$$\boxed{\begin{aligned} w &= -3 \\ b &= 5 \end{aligned}}$$



$y = -3x + 5$

Linear algebra formulation

- Solving systems of equations can also be done with linear algebra:

$$\begin{bmatrix} 5 & 3 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 6 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \end{bmatrix}$$

- Recall we originally formulated our problem by $\mathbf{y}$ and $\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}}$. Is there some relationship with the above?

Linear algebra formulation

- Yes!

$$\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 3 \end{bmatrix}$$

$$\tilde{\boldsymbol{X}}\boldsymbol{y} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix} \qquad \tilde{\boldsymbol{w}} = \begin{bmatrix} w \\ b \end{bmatrix}$$

- We can rewrite the previous solution as:

$$\begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 6 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \end{bmatrix}$$

$$(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T)\tilde{\boldsymbol{w}} = \tilde{\boldsymbol{X}}\boldsymbol{y}$$

$$\tilde{\boldsymbol{w}} * = (\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T)^{-1}\tilde{\boldsymbol{X}}\boldsymbol{y}$$

Optimal solution

Linear algebra formulation

$$\tilde{\boldsymbol{w}} * = \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T\right)^{-1}\tilde{\boldsymbol{X}}\boldsymbol{y}$$

This is equivalent to the partial derivative formulation we derived earlier.

In other words, linear least squares is as simple as correctly forming your $\tilde{\boldsymbol{X}}$ matrix and $\boldsymbol{y}$ vector and then applying standard linear algebra operations to obtain a closed-form solution

Linear algebra formulation

- Another way of derivation (for D-dimensional data)

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = w[1]x[1] + \ \ldots\ldots + w[D]x[D] + b = [\boldsymbol{w}^{\boldsymbol{T}}, b]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix} = \widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}}$$

$$\widetilde{\boldsymbol{X}} = [\widetilde{\boldsymbol{x}}_1, \quad \ldots, \quad \widetilde{\boldsymbol{x}}_N] = \begin{bmatrix} | & & | \\ \widetilde{\boldsymbol{x}}_1 & \ldots & \widetilde{\boldsymbol{x}}_N \\ | & & | \end{bmatrix}$$

$$E = \left\|\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - \mathbf{y}\right\|_2^2 = \left(\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - \boldsymbol{y}\right)^T\left(\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - \boldsymbol{y}\right) = \left(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{X}} - \boldsymbol{y}^T\right)\left(\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - \boldsymbol{y}\right)$$
$$= \widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - 2\boldsymbol{y}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} + \boldsymbol{y}^T\boldsymbol{y} = \widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - 2\boldsymbol{y}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} + \mathbf{Const.}$$

$$\nabla_{\widetilde{\boldsymbol{w}}}E = \begin{bmatrix} \frac{\partial E}{\partial \widetilde{\boldsymbol{w}}[1]} \\ \vdots \\ \frac{\partial E}{\partial \widetilde{\boldsymbol{w}}[D]} \end{bmatrix} = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - \mathbf{2}\widetilde{\boldsymbol{X}}\boldsymbol{y}{=}0$$

$$\widetilde{\boldsymbol{w}} * = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1}\widetilde{\boldsymbol{X}}\boldsymbol{y}$$

Linear algebra formulation

- Works for any model with **linear** combination of parameters
- Need enough examples for unknowns (i.e., parameters)
  - D+1 unknowns → D+1 equations/points (unique)
    - Often need many more than D to get a good result
      - Noise
      - Overfitting
- Another explanation:
  - $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T$ is (D+1)-by-(D+1)
  - $\left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}$ exists only if $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T$ is of full rank: $N \geq D + 1$

Extension: non-linear models (1-dimension)

- Consider polynomials…

$$y = g + cx + bx^2 + ax^3 \quad = g \cdot 1 + \cdots$$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\boldsymbol{X}}^T = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{bmatrix} \qquad \widetilde{\boldsymbol{w}} = \begin{bmatrix} g \\ c \\ b \\ a \end{bmatrix}$$

$$N \geq 4$$

Though g is a constant term, you can think of it as being the coefficient for 1 (equivalently x^0 in polynomials)

Note that the order in $\widetilde{\boldsymbol{w}}$ doesn't matter, as long as the rows of $\widetilde{\boldsymbol{w}}$ correspond with the columns of $\widetilde{\boldsymbol{X}}^T$

Extension: non-linear models (2-dimension)

- Consider 2nd-order polynomials…

$$y = b + w[1]x[1] + w[2]x[2] + w[3](x[1]x[2]) + w[4](x[1]^2) + w[5](x[2]^2)$$

- How many training data are needed?

Extension: non-linear models

- Or non-polynomial terms…
  - But parameters are still linear in terms of these terms

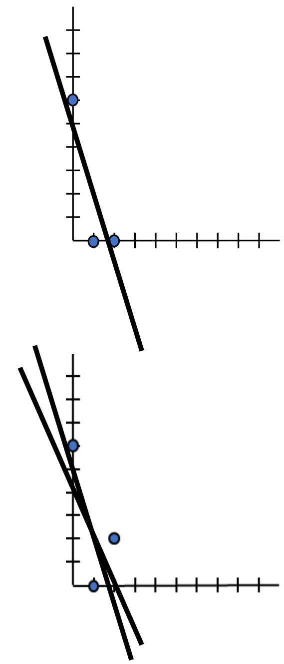
$$y = a\frac{1}{x} + b\log x + c\,e^{x}$$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\boldsymbol{X}}^T = \begin{bmatrix} 1/x_1 & \log x_1 & e^{x_1} \\ 1/x_2 & \log x_2 & e^{x_2} \\ \vdots & \vdots & \vdots \\ 1/x_N & \log x_N & e^{x_N} \end{bmatrix} \qquad \widetilde{\boldsymbol{w}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$N \geq 3$$

Problems

- Requires relationship to be *linear*
- Matrix inverse is expensive for large numbers of parameters (complex models)
  - In general, O($(D+1)^3$)
  - Sometime, matrix inverse might even not exist
- <u>Outliers (noise in the "label" or "input features")</u>
  - Data point(s) which don't match the pattern
  - Squared error is large, large influence on SSE
  - Thus, outliers can significantly affect results



Notes

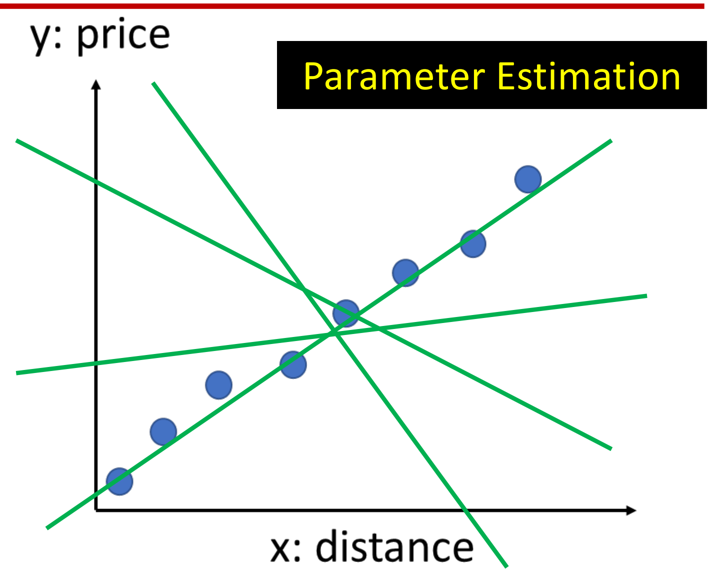
- In different textbooks or papers
  - b is sometimes denoted as $w[0]$;
  - $\boldsymbol{w}$ is called weights or parameters or parameter vector, and b as bias;
  - sometimes, $\widetilde{\boldsymbol{w}}$ is called parameters, too;
  - sometimes, people write $\boldsymbol{w}$, but they actually mean $\widetilde{\boldsymbol{w}}$
  - sometimes, people write $\widetilde{\boldsymbol{X}}$ (or $\boldsymbol{X}$) to denote $\begin{bmatrix} -\widetilde{\boldsymbol{x}}_1^T - \\ \vdots \\ -\widetilde{\boldsymbol{x}}_N^T - \end{bmatrix}$ (or $\begin{bmatrix} -\boldsymbol{x}_1^T - \\ \vdots \\ -\boldsymbol{x}_N^T - \end{bmatrix}$)
- So please pay attention to contexts when you read papers, textbooks, or assigned reading material.

Summary

- Linear regression (LR)
  - Fit a linear line to labeled data:
    - For one-dimensional cases, $f(\boldsymbol{x}) = wx + b$
    - For multi-dimensional cases, $f(\boldsymbol{x}) = \sum_{d=1}^{D} w[d]x[d] + b$
      $= f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$
  - Loss/error: absolute, square, etc.
- Parameter estimation for LR
  - For the sum of square error: $E = \sum_i [y_i - f(\boldsymbol{x}_i)]^2$
  - The closed-form solution that minimizes E (if we have the inverse)
    - $\widetilde{\boldsymbol{w}}* = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1}\widetilde{\boldsymbol{X}}\boldsymbol{y},$ $\widetilde{\boldsymbol{w}}* = \begin{bmatrix}\boldsymbol{w}*\\ b*\end{bmatrix}, \widetilde{\boldsymbol{X}} = [\widetilde{\boldsymbol{x}}_1, \ \ldots, \ \widetilde{\boldsymbol{x}}_N] = \begin{bmatrix}\boldsymbol{x}_1 & \ldots & \boldsymbol{x}_N\\ 1 & & 1\end{bmatrix}$
  - Careful about how the data is represented

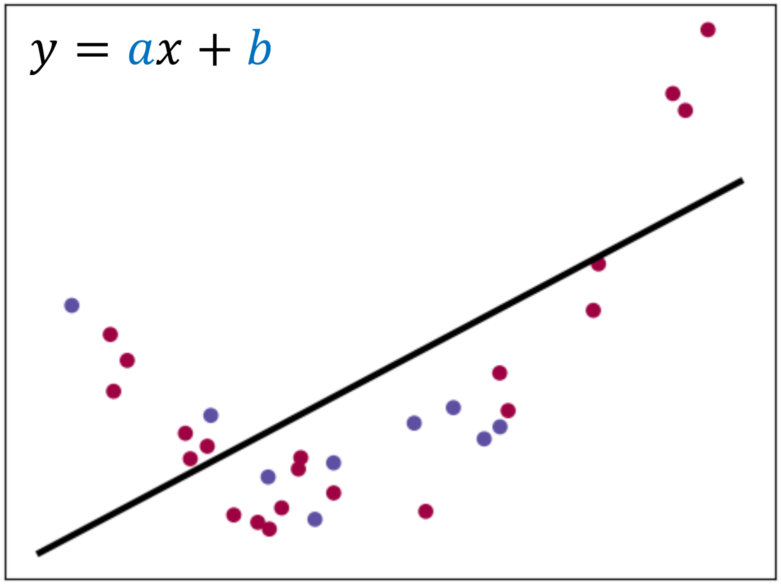
Concatenate b at the end of the column vector ***w***

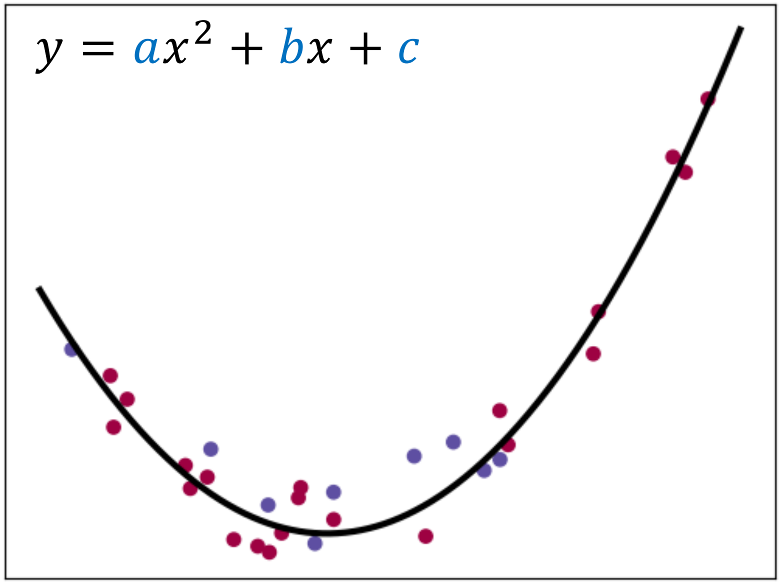
Concatenate 1 at the end of each column vector



Recap: 1-dimensional case

- The closed-form solution is applicable to some non-linear regression
  - Polynomial, or when $f(\boldsymbol{x}_i)$ is the linear combination of terms derived by $\boldsymbol{x}$.
  - a, b, c just are just to indicate what to estimate. You can change the notations.





Recap: 1-dimensional case

$y = g + cx + bx^2 + ax^3$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\boldsymbol{X}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ x_1^3 & x_2^3 & \cdots & x_N^3 \end{bmatrix} \qquad \widetilde{\boldsymbol{X}}^T = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{bmatrix} \qquad \widetilde{\boldsymbol{w}} = \begin{bmatrix} g \\ c \\ b \\ a \end{bmatrix}$$

$$\widetilde{\boldsymbol{w}}* = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}\widetilde{\boldsymbol{X}}\boldsymbol{y}$$

x_i: the i-th data instance
x[d]: the d-th dimension of a data instance
x_i[d]: the d-th dimension of the i-th data instance

Topics

- Linear w.r.t. *x* or *w*
- Non-linear least squares
  - Estimate parameters for non-linear functions w/o closed form solutions
- Gradient descent
  - Alternate solution methods
  - Not limited to SSE error functions

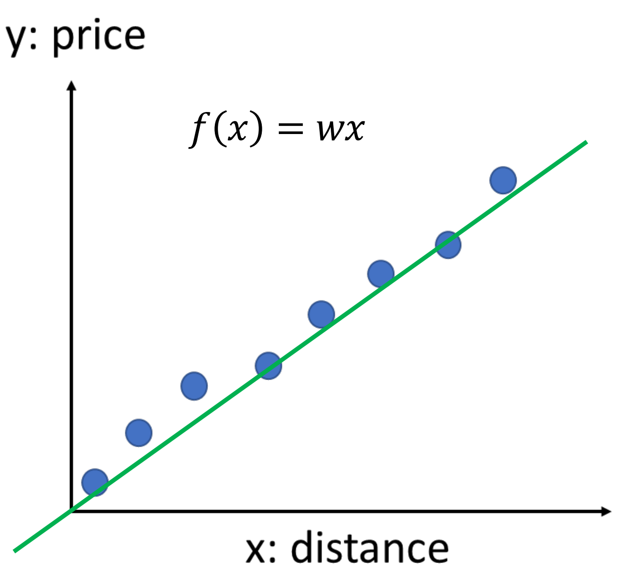
We will focus on 1-dimensional cases first!

In finding the solution

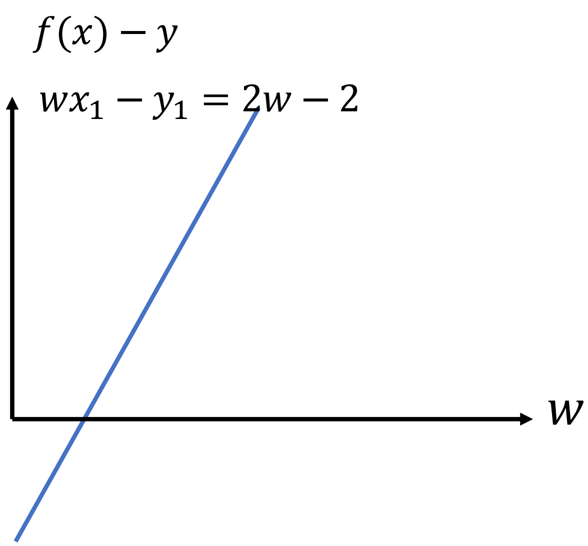
- The training data instances (x_i, y_i) are fixed
- The variables are the parameters to estimate
- A machine learning algorithm finds the parameters to minimize *E*
- For the users, we care about given a future x, what the prediction will be

$$E = \sum_i [y_i - f(\boldsymbol{x}_i)]^2$$

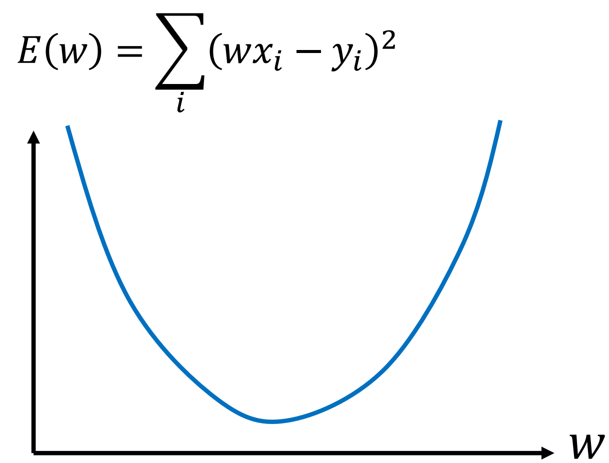
Different figures



For simplicity, only estimate *w*

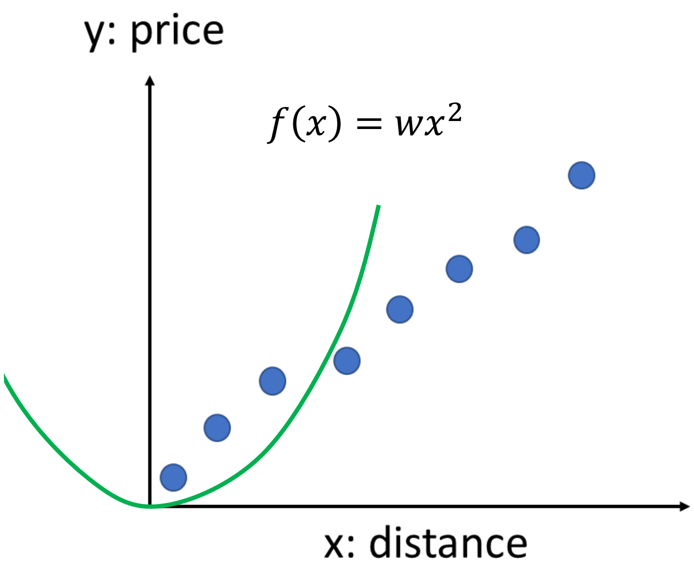


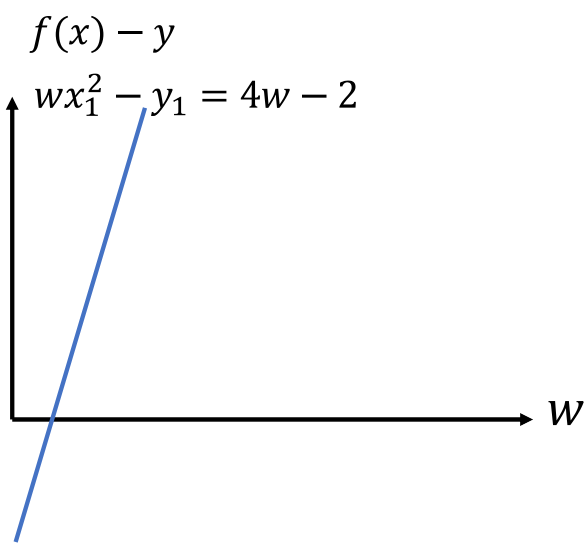
Let $(x_1, y_1) = (2, 2)$



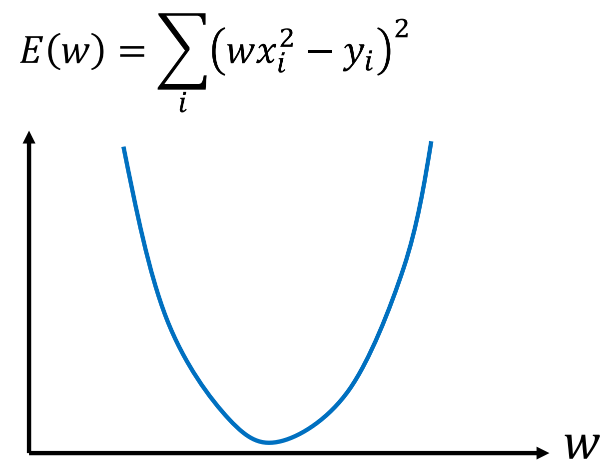
Parabolic!

Different figures





Let $(x_1, y_1) = (2, 2)$

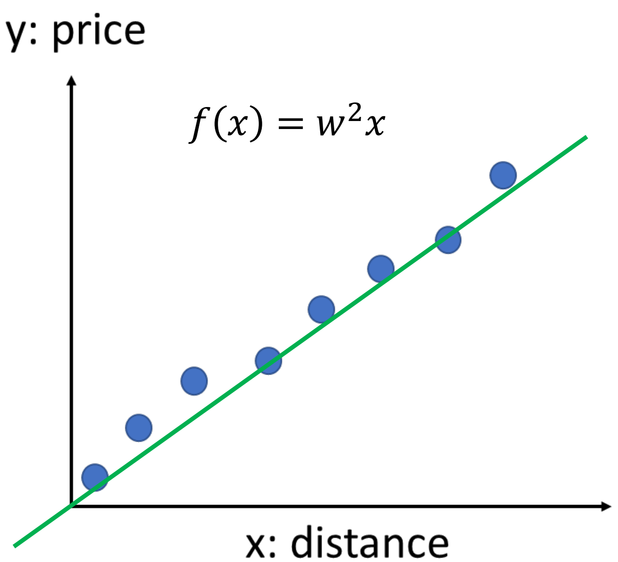


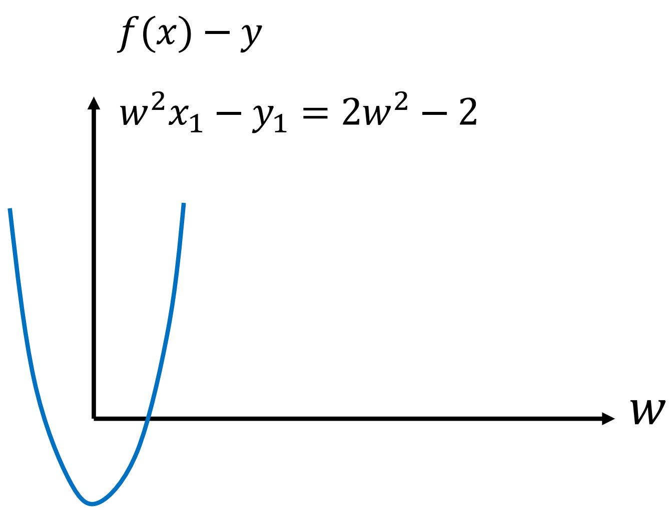
Nonlinear regression as $f(x) = wx^2$ is not a straight line w.r.t. x.

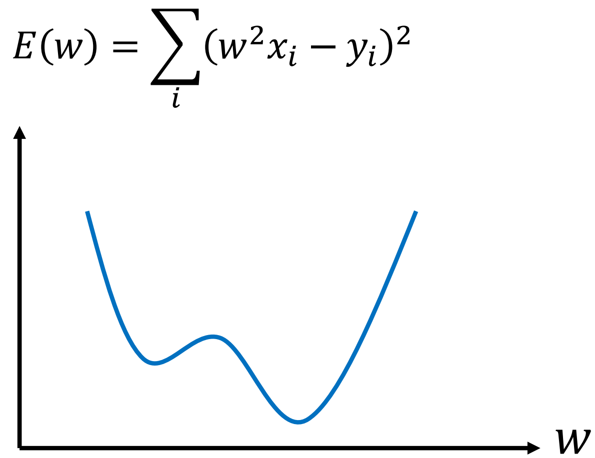
Still a straight line w.r.t. w!

Parabolic!

Different figures







Let $(x_1, y_1) = (2, 2)$

Linear regression as $f(x) = w^2 x$ is a straight line w.r.t. x.

Not a straight line w.r.t. w!

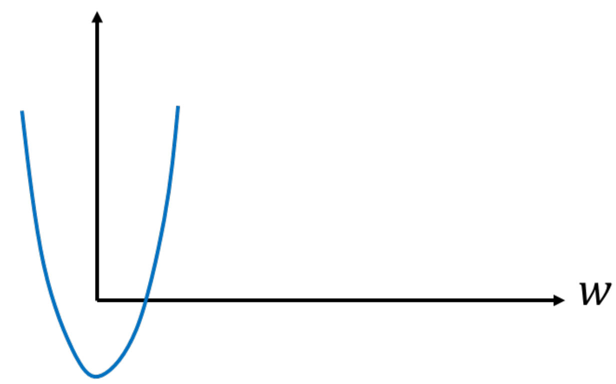
Not parabolic!

When …

To indicate that *w* is the parameter!

- Given a random (x, y), when the prediction function $f(x\boxed{;w})$ or $f(x\boxed{;w}) - y$ is not a straight line (i.e., nonlinear) w.r.t. *w*
  - there may be no closed form solution
  - *E* may not be parabolic anymore

$w^2 x_1 - y_1 = 2w^2 - 2$



Let $(x_1, y_1) = (2, 2)$

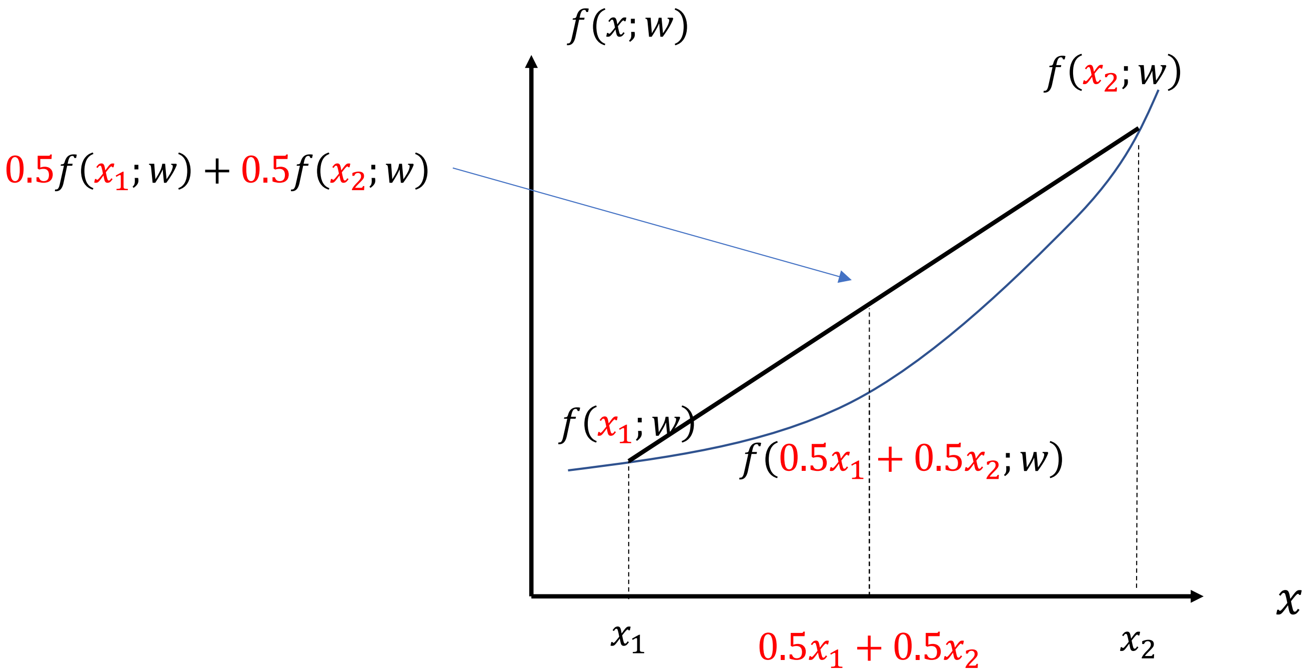
- The closed form solution $\tilde{\boldsymbol{w}}* = \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T\right)^{-1}\tilde{\boldsymbol{X}}\boldsymbol{y}$ exists
  - for the SSE loss $E(w) = \sum_i (f(x; w) - y_i)^2$
  - for $f(x; w)$ linear w.r.t. *w*

- We say linear or nonlinear regression, w.r.t. *x*

How to check if $f(x;w)$ linear w.r.t. x or w?

- If you want to check w.r.t. x
  - Ex. $f(x;w) = wx$
- Find 2 random x: x_1 and x_2
  - Ex. $x_1 = 1, x_2 = 2$
- Find 2 random scalars (even for *D*-dimensional cases): v_1 and v_2
  - Ex. $v_1 = 1, v_2 = 2$
- If $f(v_1x_1 + v_2x_2; w) = v_1f(x_1; w) + v_2f(x_2; w)$
  - $f(x;w)$ is linear w.r.t. x; otherwise, not
  - Need to hold for all possible x_1, x_2, v_1, v_2
- Practice:
  - Is $f(x;w)$ linear w.r.t. x?

Interpretation 1



Interpretation 2

x

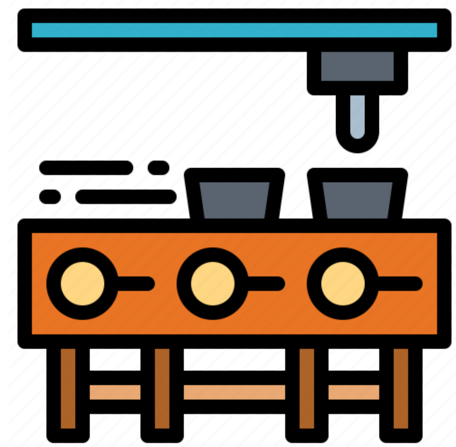
f

$f(x;w)$

x_1

x_2

$x_1 + x_2$



$f(x_1;w)$

$f(x_2;w)$

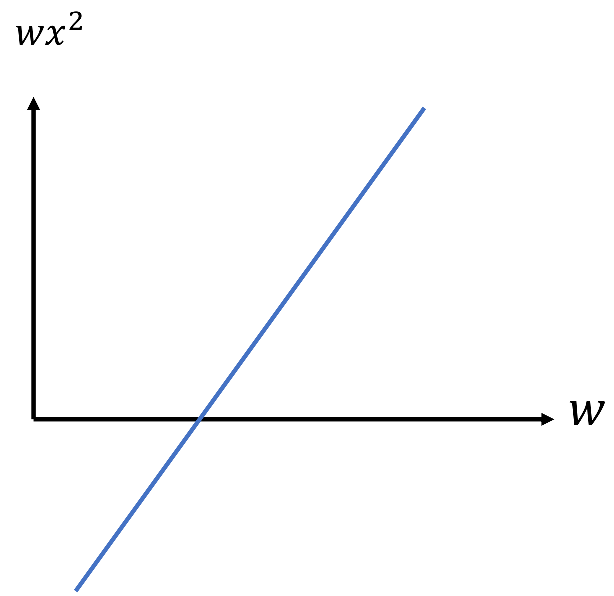
$f(x_1 + x_2;w)$

How to check if $f(x;w)$ linear w.r.t. x or w?

- If you want to check w.r.t. w
  - Ex. $f(x;w) = wx^2$
- Find 2 random w: w_1 and w_2
  - Ex. $w_1 = 1, w_2 = 2$
- Find 2 random scalars (even for *D*-dimensional cases): v_1 and v_2
  - Ex. $v_1 = 1, v_2 = 2$
- If $f(x;\ v_1 w_1 + v_2 w_2) = v_1 f(x; w_1) + v_2 f(x; w_2)$
  - $f(x;w)$ is linear w.r.t. w; otherwise, not
  - Need to hold for all possible w_1,w_2, v_1, v_2
- Practice:
  - Is $f(x;w)$ linear w.r.t. w?

Practice

- wx^2
  - Is it linear w.r.t. w?
  - Let $w_1 = 1, w_2 = 2, v_1 = 1, v_2 = 2$
  - $(v_1 w_1 + v_2 w_2)\ x^2 = 5x^2$
  - $v_1(w_1 x^2) + v_2(w_2 x^2) = 5x^2$



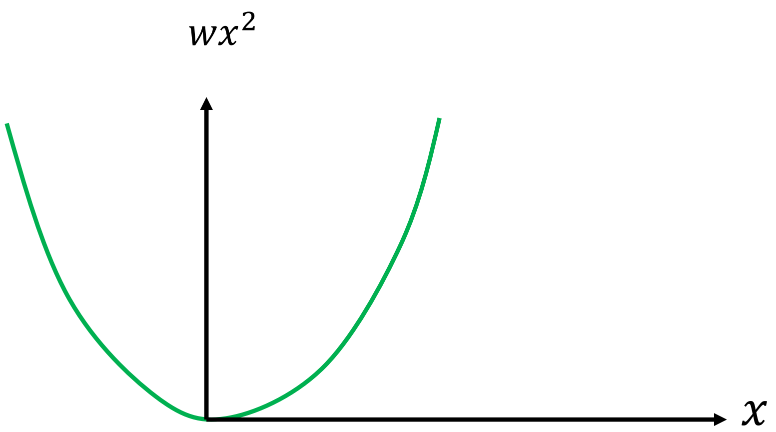
Practice

- wx^2
    - Is it linear w.r.t. x?

Practice

- wx^2
  - Is it linear w.r.t. x?

  - Let $x_1 = 1, x_2 = 2, v_1 = 1, v_2 = 2$
  - $w(v_1x_1 + v_2x_2)^2 = 25w$
  - $v_1(w{x_1}^2) + v_2(w{x_2}^2) = 9w$



Topics

- Linear w.r.t. x or w
- Non-linear least squares
  - Estimate parameters for non-linear functions
- Gradient descent
  - Alternate solution methods
  - Not limited to SSE error functions

Non-linear regression

- For many cases when $f(x; w)$ is not linear w.r.t. x, $f(x; w)$ is not linear w.r.t. w
- Examples:
  - $f(x; w) = e^{wx}$
  - $f(x; w) = \sin(wx)$

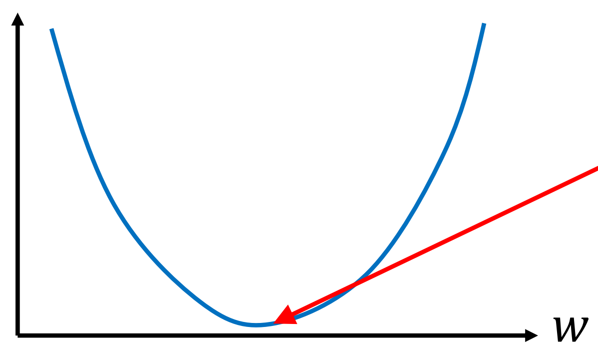
- For non-linear regression, $\boldsymbol{x}$ and $\boldsymbol{w}$ very likely have different dimensionality
- Example:
  - $f(x; w) = e^{w[1]x} + \sin(w[2]x)$

- We will say $\boldsymbol{x}$ is D-dimensional and $\boldsymbol{w}$ is K-dimensional

When $f(x; w)$ is nonlinear w.r.t. w

- Squared-error function is no longer parabolic w.r.t. the parameters
  - Can not assume local minimums of error functions are always global minimums
- Often no closed-form solution
  - Need numerical approximation

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



❖ $\frac{\partial E}{\partial w} = 0$ means global minimums

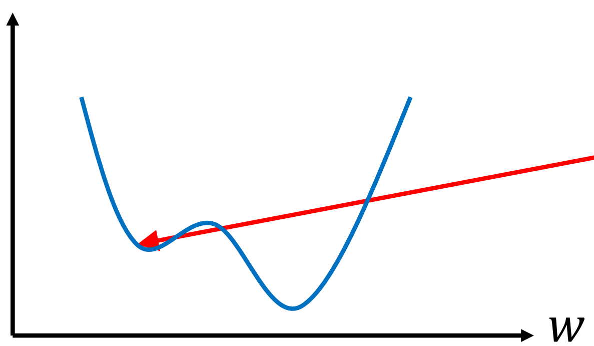
❖ Closed-forms to find w^* for $\frac{\partial E}{\partial w} = 0$

When $f(x; w)$ is nonlinear w.r.t. w

- Squared-error function is no longer parabolic w.r.t. the parameters
  - Can not assume local minimums of error functions are always global minimums

- Often no closed-form solution
  - Need numerical approximation

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



- ❖ $\frac{\partial E}{\partial w} = 0$ does not mean global minimums
- ❖ No closed-forms to find w^* for $\frac{\partial E}{\partial w} = 0$

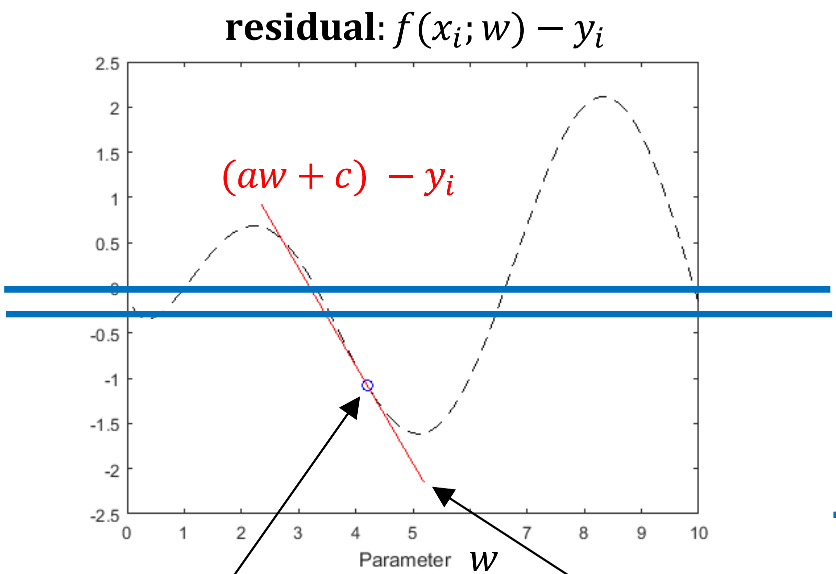
How to solve? Approximation via linearization

- The problem is the nonlinear $f(\boldsymbol{x}; \boldsymbol{w})$ w.r.t. parameters $\boldsymbol{w}$
  - So non-parabolic squared error function E w.r.t. parameters $\boldsymbol{w}$

- Idea: force the model f be linear w.r.t. parameters
  - Local – around current guess for parameters
    - This means it must be possible to approximate a function with a line locally
  - Use truncated Taylor series
  - Solve linear system as with linear least squares
  - Update current guess and repeat

- Note: $\boldsymbol{w}$ may have a very different dimensionality from $\boldsymbol{x}$

Local Linearization (1-dim cases)

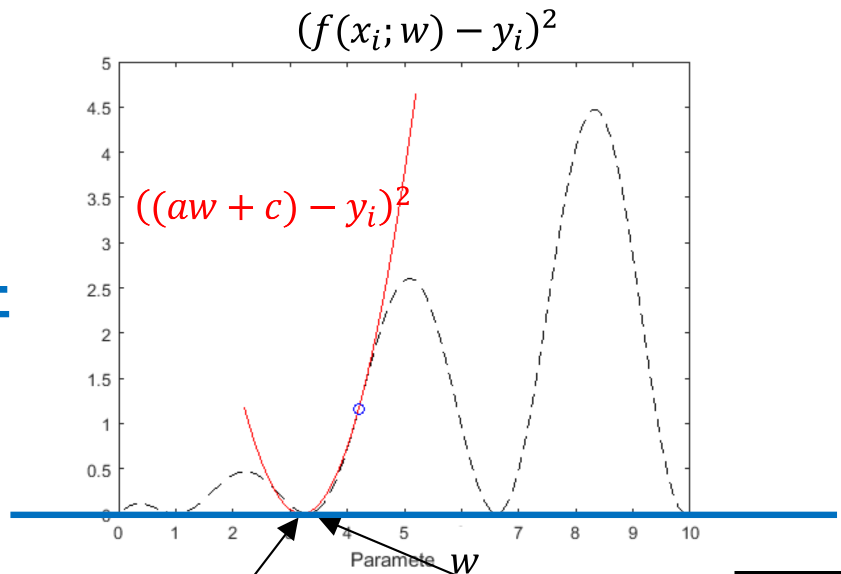


Initial guess: w'=4.2

Linearize around guess

Note: a, c will be known; w is to be solved

Want to find where the **red line** intersect the **blue line**!



New guess: w'=3.2031

True minima: w^*=3.3069

You know how to find w' to minimize $((aw + c) - y_i)^2$

Very close even after just one iteration!

Local Linearization (a 1-dim case for w and x)

- Recall the model (per data point *i*)

$$f(x_i; w)$$

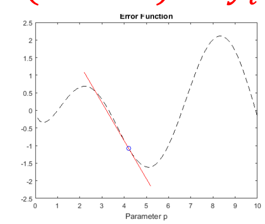
Read: f of x_i given parameters w

- Taylor series expansion (assume w is one-dimensional)

Constant term

Linear (first-order) term

$(aw + c) - y_i$



$$f(x_i; w) = \boxed{f(x_i; w')} + \boxed{\frac{\mathrm{d}f(\mathrm{x}_i; w')}{\mathrm{d}w}(w - w')} + \text{higher-order terms (H.O.T.)}$$

Variable to solve

Current guess

Variation "around" w'

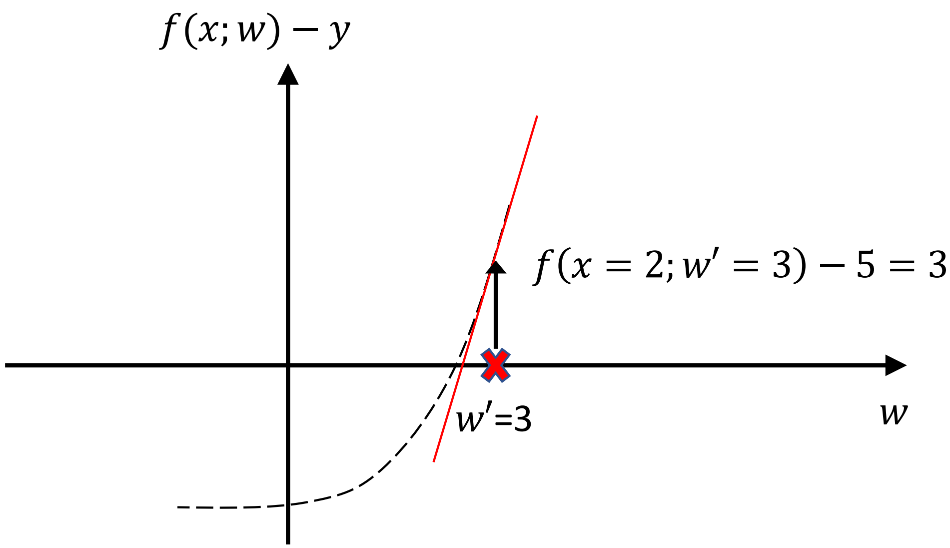
- Linearize: drop higher-than-linear terms

$$f(x_i; w) - y_i \approx f(x_i; w') + \frac{\mathrm{d}f(\mathrm{x}_i; w')}{\mathrm{d}w}(w - w') - y_i \; = J(w')\Delta w - \Delta y_i(w')$$

$\Delta y_i(w') = y_i - f(x_i; w')$ $\qquad$ $\Delta w = w - w'$ $\qquad$ <u>Jacobian</u>: $J_i(w') = \frac{\mathrm{d}f(x_i; w')}{\mathrm{d}w}$

Local Linearization (a 1-dim case for w and x)

- Example:
  - $f(x; w) = x^w$
  - $(x, y) = (2, 5)$



- Initial guess
  - w'=3
  - $\frac{\mathrm{d}f\left(x=2;\, w'=3\right)}{\mathrm{d}w} = \log(2) \times 2^3 = 5.545$
  - $J(w')\Delta w - \Delta y_i(w') = 5.545(w-3) - (5-8) = 5.545(w-3) + 3$

Minimizing the “sum of square error”

- Then, sum of squared error:

$$E'(\Delta w) = \sum_i [J_i(w')\Delta w - \Delta y_i(w')]^2 = \sum_i [J_i \Delta w - \Delta y_i]^2$$

Note, to simplify, we are not stating the dependencies on the current guess w’, <u>but it is always there!</u>

- “Least squares” approximation (take derivative and set it to 0)

$$0 = \frac{\mathrm{d}}{\mathrm{d}\Delta w} E'(\Delta w) = \sum_i \frac{\mathrm{d}}{\mathrm{d}\Delta w} [J_i \Delta w - \Delta y_i]^2$$

$$= 2 \sum_i [J_i \Delta w - \Delta y_i] \frac{\mathrm{d}}{\mathrm{d}\Delta w} [J_i \Delta w - \Delta y_i]$$

$$= 2 \sum_i [J_i \Delta w - \Delta y_i] J_i$$

Minimizing the sum of square error (cont.)

- "Least squares" approximation (cont.)

$$\sum_i J_i \,\Delta y_i = \sum_i J_i \, J_i \Delta w$$

- Rewriting in matrix form gives:

$$\boldsymbol{J}^T \Delta \boldsymbol{y} = (\boldsymbol{J}^T \boldsymbol{J}) \Delta w \qquad \Delta w = (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \Delta \boldsymbol{y}$$

$$\boldsymbol{J} = \begin{bmatrix} J_1 \\ \vdots \\ J_N \end{bmatrix}, \qquad \Delta \boldsymbol{y} = \begin{bmatrix} \Delta y_1 \\ \vdots \\ \Delta y_N \end{bmatrix}$$

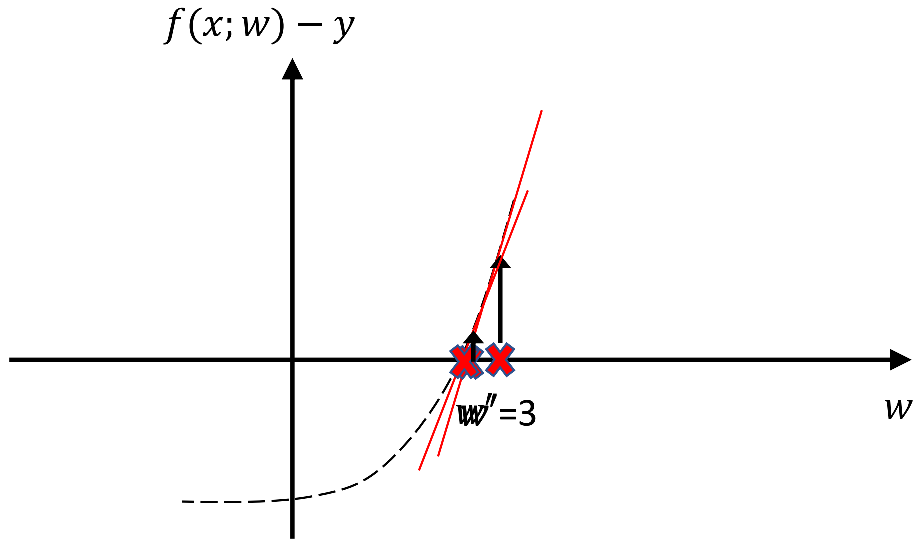
Reminder: These should be $J_i(w')$ and $\Delta y_i(w')$ as they change with the current guess w'!

- Note: Compare with linear least squares
  - LLSQ reminder: $\widetilde{\boldsymbol{X}} \boldsymbol{y} = \left(\widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \right) \widetilde{\boldsymbol{w}}$

Local Linearization (a 1-dim case for w and x)

- Example:
  - $f(x;w) = x^w$
  - $(x,y) = (2,5)$

- Algorithm
  - Initial: w'=3
  - If not converge:
    - $w' \coloneqq w' + \Delta w$



Gauss-Newton Algorithm

1. Get initial guess for parameters $\widehat{w}^{(1)}$
2. For t=1…∞ (or until we decide to stop)
    1. Find error for current guess ($\widehat{w}^{(t)}$)
    $$\Delta y_i = y_i - f(\mathbf{x}_i; \widehat{w}^{(t)})$$
    2. Find Jacobian around current guess
    $$J_i = \frac{df(\mathbf{x}_i; \widehat{w}^{(t)})}{dw}$$
    3. Calculate change in guess
    $$\Delta \boldsymbol{w} = (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \Delta \boldsymbol{y}$$
    4. Make new guess
    $$\widehat{w}^{(t+1)} = \widehat{w}^{(t)} + \Delta \widehat{w}$$

Example

- Log-linear model

$$f(x;\ \boldsymbol{w}) = e^{wx}$$

- Data points
  - $x_1 = 0.5, y_1 = 1$
  - $x_2 = 1.5, y_2 = 2$
  - $x_3 = 3, y_3 = 3$

Example - Jacobian

- Equation is simple, so closed-form Jacobian is possible

$$\frac{df(x;\boldsymbol{w})}{dw} = x\ e^{wx}$$

$$J(\boldsymbol{w}) = \begin{bmatrix} \frac{df(x_1;\boldsymbol{w})}{dw} \\ \frac{df(x_2;\boldsymbol{w})}{dw} \\ \frac{df(x_3;\boldsymbol{w})}{dw} \end{bmatrix} = \begin{bmatrix} 0.5\ e^{0.5w} \\ 1.5\ e^{1.5w} \\ 3\ e^{3w} \end{bmatrix}$$

Example

- Initial $\widehat{w}^{(1)} = 1$

$x_1 = 0.5, y_1 = 1$
$x_2 = 1.5, y_2 = 2$
$x_3 = 3, y_3 = 3$

$$J(\widehat{w}^{(1)}) = \begin{bmatrix} \frac{df(x_1;\, \widehat{w}^{(1)})}{dw} \\ \frac{df(x_2;\, \widehat{w}^{(1)})}{dw} \\ \frac{df(x_3;\, \widehat{w}^{(1)})}{dw} \end{bmatrix} = \begin{bmatrix} 0.5\, e^{0.5\, \widehat{w}^{(1)}} \\ 1.5\, e^{1.5\, \widehat{w}^{(1)}} \\ 3\, e^{3\, \widehat{w}^{(1)}} \end{bmatrix} = \begin{bmatrix} 0.5\, e^{0.5} \\ 1.5\, e^{1.5} \\ 3\, e^{3w} \end{bmatrix}$$

$$\Delta \boldsymbol{y}\left(\widehat{w}^{(1)}\right) = \begin{bmatrix} y_1 - f(\boldsymbol{x}_1; \widehat{w}^{(1)}) \\ y_2 - f(\boldsymbol{x}_2; \widehat{w}^{(1)}) \\ y_3 - f(\boldsymbol{x}_3; \widehat{w}^{(1)}) \end{bmatrix} = \begin{bmatrix} 1 - e^{0.5\, \widehat{w}^{(1)}} \\ 2 - e^{1.5\, \widehat{w}^{(1)}} \\ 3 - 3e^{3\, \widehat{w}^{(1)}} \end{bmatrix} = \begin{bmatrix} 1 - e^{0.5} \\ 2 - e^{1.5} \\ 3 - 3e^{3} \end{bmatrix}$$

Local Linearization (let $\boldsymbol{w}$ be K-dim)

- Recall the model (per data point *i*)

$$f(\boldsymbol{x};\boldsymbol{w})$$

- Taylor series expansion

$$f(\boldsymbol{x};\boldsymbol{w}) = f(\boldsymbol{x}_i;\boldsymbol{w}') + \sum_{k=1}^{K} \frac{\partial f(\boldsymbol{x}_i;\boldsymbol{w}')}{\partial w[k]} (w[k] - w'[k]) + \text{higher-order terms (H.O.T.)}$$

$$= f(\boldsymbol{x}_i;\boldsymbol{w}') + \begin{bmatrix} \frac{\partial f(\boldsymbol{x}_i;\boldsymbol{w}')}{\partial w[1]} & \cdots & \frac{\partial f(\boldsymbol{x}_i;\boldsymbol{w}')}{\partial w[K]} \end{bmatrix} \begin{bmatrix} (w[1] - w'[1]) \\ \vdots \\ (w[K] - w'[K]) \end{bmatrix} + \text{H.O.T.}$$

$$= f(\boldsymbol{x}_i;\boldsymbol{w}') + \nabla_{\boldsymbol{w}} f(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}') + \text{H.O.T.}$$

Local Linearization (let $\boldsymbol{w}$ be K-dim)

- Linearize: drop higher-than-linear terms

$$f(\boldsymbol{x};\boldsymbol{w}) - y_i \approx f(\boldsymbol{x};\boldsymbol{w}') + \left\{\sum_{k=1}^{K} \frac{\partial f(\boldsymbol{x}_i;\boldsymbol{w}')}{\partial w[k]} (w[k] - w'[k])\right\} - y_i$$

$$= \left\{\sum_{k=1}^{K} J_{ik}(\boldsymbol{w}')\Delta\boldsymbol{w}[k]\right\} - \Delta y_i(\boldsymbol{w}')$$

$$\Delta y_i(\boldsymbol{w}') = y_i - f(\boldsymbol{x};\boldsymbol{w}')$$

$$\Delta\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{w}'$$

<u>Jacobian matrix</u>: $\boldsymbol{J} = \begin{bmatrix} \frac{\partial f(\boldsymbol{x}_1;\boldsymbol{w}')}{\partial w[1]} & \cdots & \frac{\partial f(\boldsymbol{x}_1;\boldsymbol{w}')}{\partial w[K]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\boldsymbol{x}_N;\boldsymbol{w}')}{\partial w[1]} & \cdots & \frac{\partial f(\boldsymbol{x}_N;\boldsymbol{w}')}{\partial w[K]} \end{bmatrix}$ $\quad J_{ik} = J[i,k] = \frac{\partial f(\boldsymbol{x}_i;\boldsymbol{w}')}{\partial w[k]}$

Minimizing the “sum of square error” (let $\boldsymbol{w}$ be K-dim)

- Then, sum of squared error:

$$E'(\Delta\boldsymbol{w}) = \sum_i \left[\left\{\sum_{k=1}^{K} J_{ik}(\boldsymbol{w}')\Delta\boldsymbol{w}[k]\right\} - \Delta y_i(\boldsymbol{w}')\right]^2 = \sum_i \left[\left\{\sum_{k=1}^{K} J_{ik}\Delta\boldsymbol{w}[k]\right\} - \Delta y_i\right]^2$$

- “Least squares” approximation (take partial derivatives and set them to 0)

$$0 = \frac{\partial}{\partial \Delta w[k]} E'(\Delta\boldsymbol{w}) = \sum_i \frac{\partial}{\partial \Delta w[k]} \left[\left\{\sum_{k=1}^{K} J_{ik}\Delta\boldsymbol{w}[k]\right\} - \Delta y_i\right]^2$$

$$= 2\sum_i \left[\left\{\sum_{j=1}^{K} J_{ij}\Delta\boldsymbol{w}[k]\right\} - \Delta y_i\right] \frac{\partial}{\partial \Delta w[k]} \left[\left\{\sum_{k=1}^{K} J_{ik}\Delta\boldsymbol{w}[k]\right\} - \Delta y_i\right]$$

$$= 2\sum_i \left[\left\{\sum_{j=1}^{K} J_{ij}\Delta\boldsymbol{w}[k]\right\} - \Delta y_i\right] J_{ik}$$

Minimizing the "sum of square error" (let $\boldsymbol{w}$ be K-dim)

- "Least squares" approximation (cont.)

$$\sum_i J_{ik}\,\Delta y_i = \sum_i \sum_{j=1}^{K} J_{ik} J_{ij} \Delta \boldsymbol{w}[k]$$

- Rewriting in matrix form gives:

$$\boldsymbol{J}^T \Delta \boldsymbol{y} = (\boldsymbol{J}^T \boldsymbol{J})\,\Delta \boldsymbol{w}$$

Reminder: These should be $J_i(\boldsymbol{w}')$ and $\Delta y_i(\boldsymbol{w}')$ as they change with the current guess $\boldsymbol{w}'$!

Gauss-Newton Algorithm (let $\boldsymbol{w}$ be K-dim)

1. Get initial guess for parameters $\widehat{\boldsymbol{w}}^{(1)}$
2. For t=1…∞ (or until we decide to stop)
   1. Find error for current guess ($\widehat{\boldsymbol{w}}^{(t)}$)
      $$\Delta y_i = y_i - f(\mathbf{x}_i;\, \widehat{\boldsymbol{w}}^{(t)})$$
   2. Find Jacobian around current guess
      $$J_{ik} = \frac{\partial f\left(\mathbf{x}_i; \widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[k]}$$
   3. Calculate change in guess
      $$\Delta \boldsymbol{w} = (\boldsymbol{J}^T\, \boldsymbol{J})^{-1} \boldsymbol{J}^T\, \Delta \boldsymbol{y}$$
   4. Make new guess
      $$\widehat{\boldsymbol{w}}^{(t+1)} = \widehat{\boldsymbol{w}}^{(t)} + \Delta \widehat{\boldsymbol{w}}$$

Example

- Log-linear model

$$f(x;\ \boldsymbol{w}) = e^{w[1]x+w[2]}$$

- Data points
  - $x_1 = 0.5,\ y_1 = 1.6487$
  - $x_2 = 1.5,\ y_2 = 0.6065$
  - $x_3 = 3,\ y_3 = 0.1353$

Example - Jacobian

- Equation is simple, so closed-form Jacobian is possible

$$\frac{\partial f(x;\boldsymbol{w})}{\partial w[1]} = x\, e^{w[1]x+w[2]}$$

$$\frac{\partial f(x;\boldsymbol{w})}{\partial w[2]} = e^{w[1]x+w[2]}$$

$$J(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial f(x_1;\boldsymbol{w})}{\partial w[1]} & \frac{\partial f(x_1;\boldsymbol{w})}{\partial w[2]} \\ \frac{\partial f(x_2;\boldsymbol{w})}{\partial w[1]} & \frac{\partial f(x_2;\boldsymbol{w})}{\partial w[2]} \\ \frac{\partial f(x_3;\boldsymbol{w})}{\partial w[1]} & \frac{\partial f(x_3;\boldsymbol{w})}{\partial w[2]} \end{bmatrix} = \begin{bmatrix} 0.5\, e^{0.5w[1]+w[2]} & e^{0.5w[1]+w[2]} \\ 1.5\, e^{1.5w[1]+w[2]} & e^{1.5w[1]+w[2]} \\ 3\, e^{3w[1]+w[2]} & e^{3w[1]+w[2]} \end{bmatrix}$$

Problems

- Similar problems to linear least squares
  - Matrix inverse is expensive
  - Outliers!

$$\Delta \boldsymbol{w} = (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \Delta \boldsymbol{y}$$

- Initial guess
  - Very sensitive to choice of values
  - Cannot guarantee global minimum
  - Where do we get it from?
    - Expert knowledge
    - OR, try random values (multiple times!)

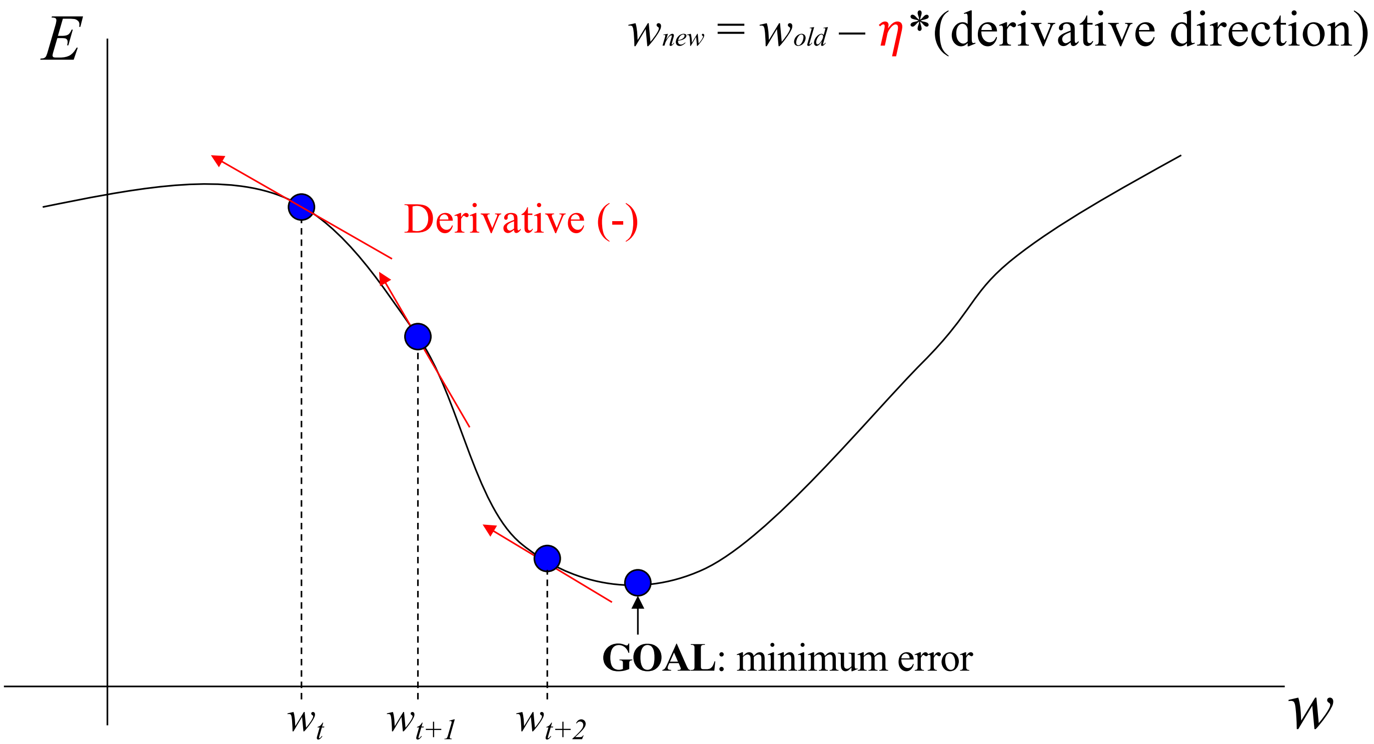
Topics

- Brief overview of machine learning: Part 3
  - Training vs. testing
  - Parameter estimation
- Non-linear least squares
  - Estimate parameters for non-linear functions
- Gradient descent
  - Alternate solution methods
  - Not limited to SSE error functions

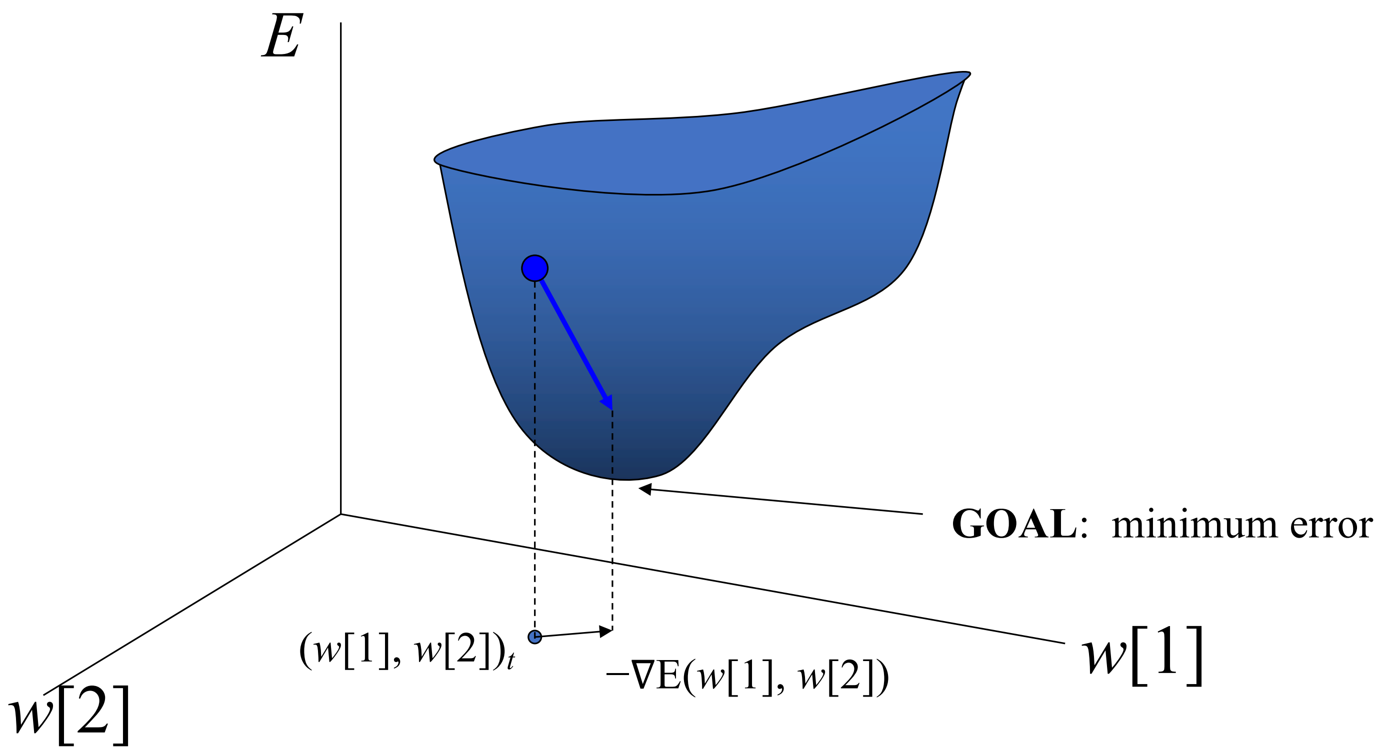
Gradient Descent (GD)

- Simple (and faster?) method for optimization
  - $f(\boldsymbol{x}; \boldsymbol{w})$ linear or nonlinear w.r.t. parameters $\boldsymbol{w}$
  - Sum or square losses or other losses
- A general method to find local minimum for a multi-parameter error function
  - Search through "parameter space" to find minimum error
- Main idea
  - Evaluate error function for <u>current guess</u> of parameters
  - Determine change in parameters that <u>decrease error</u>
    - From <u>gradient</u> of error function
  - Update parameters in this new change
  - Repeat process until converge (or hit max iterations)

Problem Visualization



Problem Visualization



Why does gradient descent work?

- Recall the Taylor expansion
  - But now apply it to the loss function $E(\boldsymbol{w})$ directly, not to $f(\boldsymbol{x}_i; \boldsymbol{w})$

$$E(\boldsymbol{w}) = E(\boldsymbol{w}') + \sum_{k=1}^{K} \frac{\partial E(\boldsymbol{w}')}{\partial w[k]} (w[k] - w'[k]) + \text{higher-order terms (H.O.T.)}$$

$$= E(\boldsymbol{w}') + \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}') + \text{H.O.T.} = E(\boldsymbol{w}') + \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T \Delta\boldsymbol{w} + \text{H.O.T.}$$

- Approximate $E(\boldsymbol{w})$ by the linear terms: $E'(\Delta\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T \Delta\boldsymbol{w}$
- Which direction, i.e., $\Delta\boldsymbol{w}$ with $\|\Delta\boldsymbol{w}\|_2^2 = \mathbf{1}$, can minimize $E'(\Delta\boldsymbol{w})$?
  - Answer: direction of $\Delta\boldsymbol{w} = -\frac{\nabla_{\boldsymbol{w}} E(\boldsymbol{w}')}{\|\nabla_{\boldsymbol{w}} E(\boldsymbol{w}')\|_2}$
  - Interpretation: move in the direction of $-\nabla_{\boldsymbol{w}} E(\boldsymbol{w}')$ is the most efficient for minimization
  - Only work for small $\Delta\boldsymbol{w}$ (so a scale η) due to the nature of linear Taylor expansion

Algorithm

Initialize parameters: $\hat{\boldsymbol{w}}^{(1)} = [w[1], w[2], \ldots, w[K]\ ]^T$

Error function: $E(\boldsymbol{w})$

Compute gradients:

$$\nabla E\left(\hat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial E\left(\hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]} \\ \vdots \\ \frac{\partial E\left(\hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[K]} \end{bmatrix}$$

Update parameters: $\hat{\boldsymbol{w}}^{(t+1)} = \hat{\boldsymbol{w}}^{(t)} - \eta \cdot \nabla E\left(\hat{\boldsymbol{w}}^{(t)}\right)$

[Repeat until converges]

Learning rate

Compute Gradients: Least Squares (w is 1-dim)

Error function: $$E(\boldsymbol{w}) = \sum_i [f(\boldsymbol{x_i}; \boldsymbol{w}) - y_i]^2$$

Error gradient: $$\nabla E(\boldsymbol{w})$$

$$\frac{dE(\widehat{\boldsymbol{w}}^{(t)})}{dw} = \frac{\partial}{\partial w} \sum_i [f(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}) - y_i]^2$$

$$= 2 \sum_i (f(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}) - y_i) \frac{df(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)})}{dw}$$

Recall: Gauss-Newton

$$2 \sum_i (f(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}) - y_i) \frac{df(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)})}{\partial w} = 2 \sum_i -\Delta y_i J_i(\widehat{\boldsymbol{w}}^{(t)}) \Rightarrow \nabla E(\boldsymbol{w}) = -2 J^T(\boldsymbol{w}) \Delta \boldsymbol{y}(\boldsymbol{w})$$

Compute Gradients: Least Squares (*w* is *K*-dim)

Error function: $$E(\boldsymbol{w}) = \sum_i [f(\boldsymbol{x_i}; \boldsymbol{w}) - y_i]^2$$

Error gradient: $$\nabla E(\boldsymbol{w})$$

$$\frac{\partial E(\hat{\boldsymbol{w}}^{(t)})}{\partial w[k]} = \frac{\partial}{\partial w[k]} \sum_i \left[f\left(\boldsymbol{x_i}; \hat{\boldsymbol{w}}^{(t)}\right) - y_i\right]^2$$

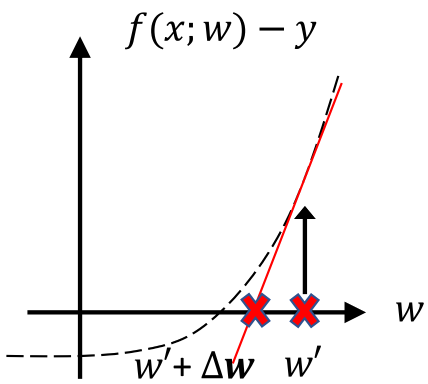
$$= 2 \sum_i \left(f\left(\boldsymbol{x_i}; \hat{\boldsymbol{w}}^{(t)}\right) - y_i\right) \frac{\partial f\left(\boldsymbol{x_i}; \hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[k]}$$

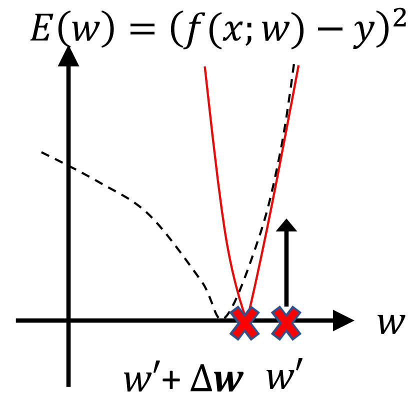
Recall: Gauss-Newton

$$2 \sum_i \left(f\left(\boldsymbol{x_i}; \hat{\boldsymbol{w}}^{(t)}\right) - y_i\right) \frac{\partial f\left(\boldsymbol{x_i}; \hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[k]} = 2 \sum_i -\Delta y_i \, J_{ik}\left(\hat{\boldsymbol{w}}^{(t)}\right) \Rightarrow \nabla E(\boldsymbol{w}) = -2\, J^T(\boldsymbol{w}) \Delta \boldsymbol{y}(\boldsymbol{w})$$

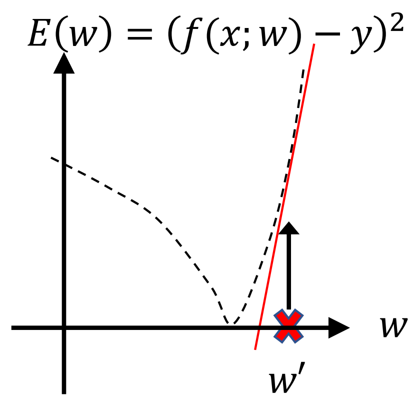
Gradient descent vs Gauss-Newton work

- Gauss-Newton
  - For square loss $\sum_i [f(\boldsymbol{x_i}; \boldsymbol{w}) - y_i]^2$
  - Linearization on $f(x; \boldsymbol{w})$ at $\boldsymbol{w}'$
  - Solve the approximated least square with the closed-form solution to get $\Delta\boldsymbol{w}$
  - Update $\boldsymbol{w}'$ by $\boldsymbol{w}' + \Delta\boldsymbol{w}$





- Gradient descent
  - For any kind of loss $E(\boldsymbol{w})$
  - Linearization in $E(\boldsymbol{w})$ at $\boldsymbol{w}'$
  - Set $\Delta\boldsymbol{w}$ to the direction of $-\nabla_{\boldsymbol{w}} E(\boldsymbol{w}')$
  - Update $\boldsymbol{w}'$ by $\boldsymbol{w}' + \eta\Delta\boldsymbol{w}$



Gradient descent vs Gauss-Newton work

- Gauss-Newton can be seen as "quadratic approximation" of $E(\boldsymbol{w})$

$$E'(\Delta \boldsymbol{w}) = \sum_i [\nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T \Delta \boldsymbol{w} - \Delta y_i(\boldsymbol{w}')]^2$$

$$= \Delta \boldsymbol{w}^T \left(\sum_i \nabla_{\boldsymbol{w}} E(\boldsymbol{w}') \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T \right) \Delta \boldsymbol{w} - 2 \sum_i \Delta y_i(\boldsymbol{w}')^T \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T \Delta \boldsymbol{w} + \sum_i \Delta y_i(\boldsymbol{w}')^T \Delta y_i(\boldsymbol{w}')$$

- As we use the quadratic term, the approximation is good for a larger $\Delta \boldsymbol{w}$
  - Thus we can directly update $\boldsymbol{w}'$ by $\boldsymbol{w}' + \Delta \boldsymbol{w}$ (without a learning rate η)
  - In practice, you can also do $\boldsymbol{w}'$ by $\boldsymbol{w}' + \eta \Delta \boldsymbol{w}$

Direct 2-order Taylor expansion approximation

- Taylor series expansion ($\Delta \boldsymbol{w} = \boldsymbol{w} - \boldsymbol{w}'$)

$$E(\Delta \boldsymbol{w}) = E(\boldsymbol{w}') + \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')^T \Delta \boldsymbol{w} + \frac{1}{2} \Delta \boldsymbol{w}^T H(\boldsymbol{w}') \Delta \boldsymbol{w} + \text{H.O.T.}$$

- Hessian matrix $H(\boldsymbol{w}')$

$$H(\boldsymbol{w}') = \begin{bmatrix} \frac{\partial^2 E(\widehat{\boldsymbol{w}}_t)}{\partial w[1] \partial w[1]} & \cdots & \frac{\partial^2 E(\widehat{\boldsymbol{w}}_t)}{\partial w[1] \partial w[K]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\widehat{\boldsymbol{w}}_t)}{\partial w[K] \partial w[1]} & \cdots & \frac{\partial^2 E(\widehat{\boldsymbol{w}}_t)}{\partial w[K] \partial w[K]} \end{bmatrix}$$

- Newton's method
  - For any kind of loss $E(\boldsymbol{w})$
  - Quadratic approximation on $E(\boldsymbol{w})$ at $\boldsymbol{w}'$
  - Solve the quadratic approximation to get $\Delta \boldsymbol{w} = -H^{-1}(\boldsymbol{w}') \nabla_{\boldsymbol{w}} E(\boldsymbol{w}')$
  - Update $\boldsymbol{w}'$ by $\boldsymbol{w}' + \Delta \boldsymbol{w}$
- Pros: faster convergence; Cons: $H(\boldsymbol{w}')$ and $H^{-1}(\boldsymbol{w}')$ are time consuming

Benefits, Issues, Helpers

- Benefits of gradient descent
  - No expensive matrix operations (i.e., inversion)
  - Non-SSE error functions (as long as gradient is well formed)
  - Easy to parallelize (supercomputers!)
- Issues
  - Can be misled by local minima
    - Get stuck at smaller valleys
  - Can get stuck on a flat plain
    - Flat region in state-space function
  - Can overshoot back and forth
    - Oscillate from side to side
    - Learning rate too large
- Helpers
  - Use random-restart to handle some problems
    - Do process multiple times with random initial parameters
  - Methods exist for selecting appropriate (best) learning rate at each iteration

Summary

- Gauss-Newton non-linear least squares
  - Iterative approximation
  - Linearize model around current parameter guess
  - Apply Linear Least Squares methods to linearized model
- Gradient descent
  - “Valley descending” (opposed to “hill climbing”)
  - Make the best update in parameters
    - In direction of negative gradient of error function