DS 310 Machine Learning

Vasant G. Honavar
Dorothy Foehr Huck and J. Lloyd Huck Chair in Biomedical Data Sciences and Artificial Intelligence
Professor of Data Sciences, Informatics, Computer Science and Engineering, Bioinformatics & Genomics, Public Health Sciences and Neuroscience
Director, Center for Artificial Intelligence Foundations and Scientific Applications
Associate Director, Institute for Computational and Data Sciences
Pennsylvania State University

vhonavar@psu.edu
http://faculty.ist.psu.edu/vhonavar
http://ailab.ist.psu.edu

Function approximation (Regression)

- Function approximation is like classification except the labels are real valued

Example applications:

Predicting

- Stock value
- Income
- Power consumption

K nearest neighbor Function Approximator

Learning Phase

For each training example $(X_i, f(X_i))$, store the example in memory

Approximation phase

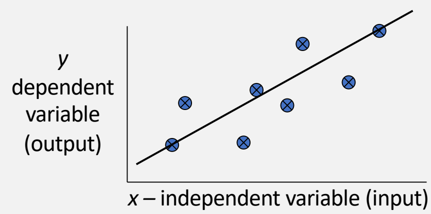
Given a query instance X_q, identify the k nearest neighbors $X_1 \ldots X_k$ of X_q

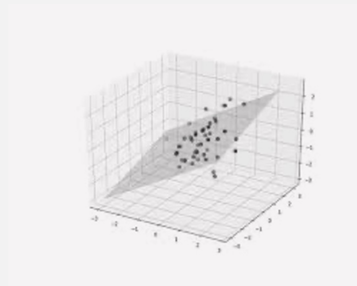
$$g(X_q) \leftarrow \frac{\sum_{l=1}^{K} f(X_i)}{K}$$

Value of a function (e.g., price of a product) at a query point is simply the average or inverse distance weighted average of the value of the function at the k nearest neighbors of the query point

Regression

- For classification the output is nominal
- In regression the output is continuous
- Linear regression is perhaps the simplest approach
  - Fit data with the best hyper-plane (line when the function is defined with respect to a single variable) which "goes through" the points





Simple Linear Regression

- In the simplest case, we have one (input), independent variable *x*, and one (output) dependent variable *y*
  - Multiple linear regression assumes an input vector **x**
  - Multivariate linear regression assumes an output vector **y**
- We will "fit" the points with a linear hyper-plane (line in the simplest case)
- Which line should we use?
  - Choose an objective function
  - For simple linear regression we choose sum squared error (SSE)
    - $\Sigma (d_i - y_i)^2 = \Sigma (e_i)^2$
  - Thus, find the line which minimizes the sum of the squared residues (e.g. least squares)

Hyperplane for multiple linear regression

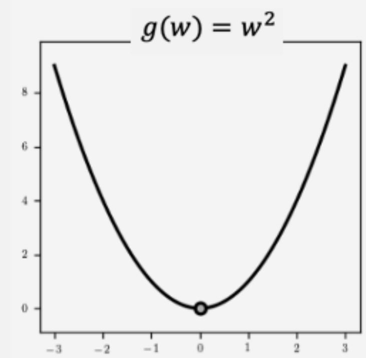
Digression: Minimizing functions

Minima of a function

- In many applications, machine learning included, we are often interested in minimizing a function of many variables.
- For a function $g(\mathbf{w})$ of N variables $w_1 \cdots w_N$, this problem is formally phrased as

$$\underset{\mathbf{w}}{\text{minimize}}\; g(\mathbf{w})$$

- That is, examine the value of $g(\mathbf{w})$ over all possible values of $\mathbf{w}$ in the domain of $g(\mathbf{w})$ and pick one or more where the value of $g(\mathbf{w})$ is minimum (over the range of $g(\mathbf{w})$.



- What is the smallest value of $g(w) = w^2$?
- What is the value of w at which this occurs?

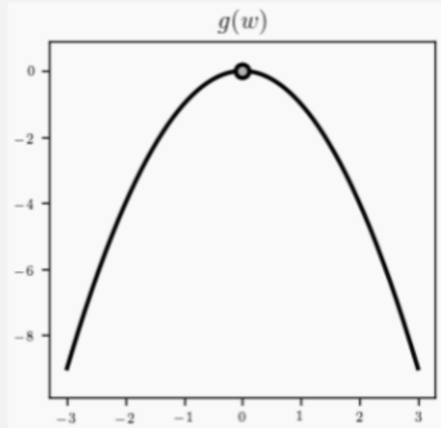
- Obviously, the minimum is smaller than any other value of the function.
- Specifically the smallest value - the global minimum of this function - seemingly occurs close to $w^{\star} = 0$
- Formally a point $w^{\star}$ gives the smallest point on the function if

$$g(w^{\star}) \leq g(w) \quad \text{for all } w.$$

- This is called the *zero-order definition of a global minimum* of a function.

- Suppose we multiply the quadratic function in the previous example by -1.
- The function flips upside down - now its global minima lie at $w^\star = \pm\infty$
- Now the point $w^\star = 0$ that used to be a global minimum is now global maximum - i.e., where the value of the function is the largest, i.e.,

$$g(w^\star) \geq g(w) \text{ for all } w.$$

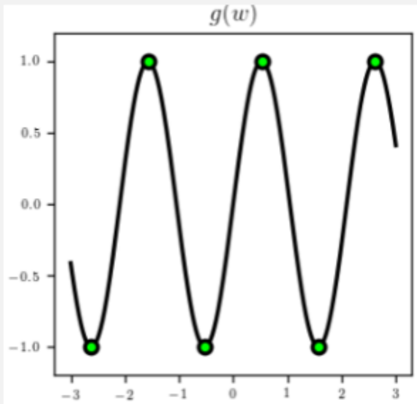


- These concepts of *minima* and *maxima* of a function are always related to each via multiplication by ***-1***.
- That is, any point that is a minima of a function ***g*** is a maxima of the function ***-g***, and vice-versa.

$$\underset{\mathbf{w}}{\text{maximize}} \;\; g\left(\mathbf{w}\right) = -\underset{\mathbf{w}}{\text{minimize}} \;\; g\left(\mathbf{w}\right).$$

Example

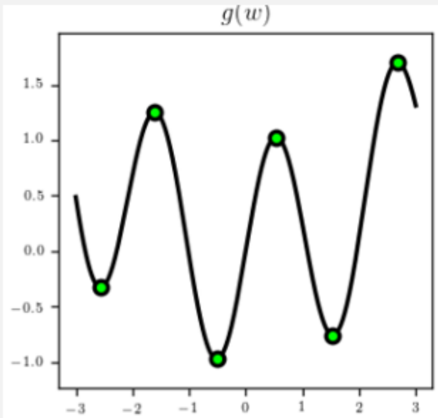
- Let us look at the sinusoid function $g(w) = \sin(2w)$
- Over the range we have plotted the function - that there are three global minima and three global maxima (marked by green dots).



- Technically speaking, this function has an infinite number of global maxima and minima
- Why?

Example: Minima and maxima of the sum of a sinusoid and a quadratic

- Let's look at a weighted sum of the previous two examples, the function $g(w) = \sin(3w) + 0.1w^2$ over a region of its input space.



- We have a global minimum around $w^\star = -0.5$ and a global maximum around $w^\star = 2.7$
- The point around $w^\star = 0.8$ is a local maximum.
- The point around $w^\star = 1.5$ is a local minimum
- Can you identify other local minima/maxima?

The zero order condition for optimality

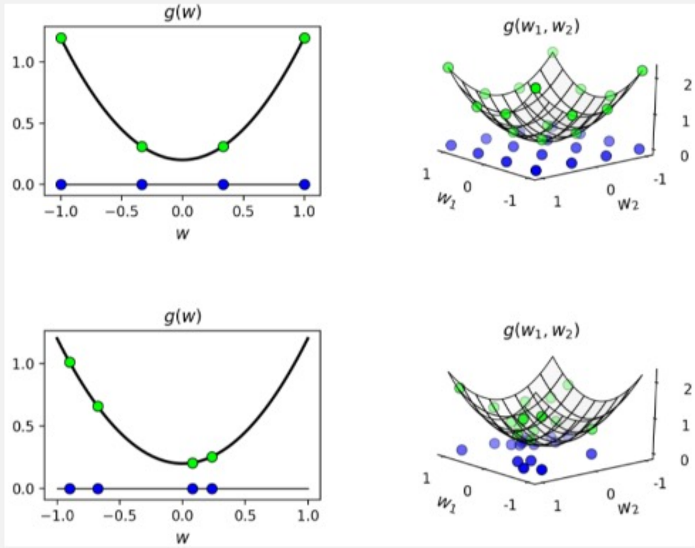
A point $\mathbf{w}^\star$ is

- a global minimum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \leq g(\mathbf{w})$ for all $\mathbf{w}$
- a global maximum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \geq g(\mathbf{w})$ for all $\mathbf{w}$
- a local minimum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \leq g(\mathbf{w})$ for all $\mathbf{w}$ near $\mathbf{w}^\star$
- a local maximum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \geq g(\mathbf{w})$ for all $\mathbf{w}$ near $\mathbf{w}^\star$

- Why zero order? Because it depends only on the function $g(\mathbf{w})$ and nothing else.
- Higher order definitions refer to the values of first, second … order derivatives of $g(\mathbf{w})$

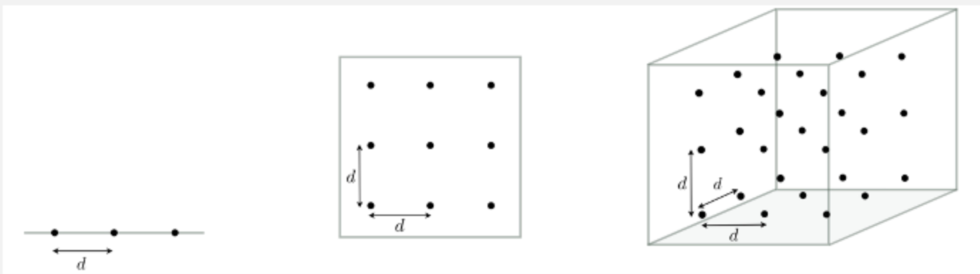
Finding global maxima / minima: A naïve approach

- Evaluate the function at a large number of points
- Among them, choose the input at which the function is the lowest as the approximate global minimum
- How can we choose the points at which to evaluate the function?
  - Uniformly
  - Randomly
- While this naïve approach works for functions of few variables, it fails for functions of more than 2 or 3 variables
- Why? The number of points at which the function needs to be evaluated grows exponentially with the number of variables


$g(w)$
w
$g(w_1, w_2)$
w_1
w_2
$g(w)$
w
$g(w_1, w_2)$
w_1
w_2

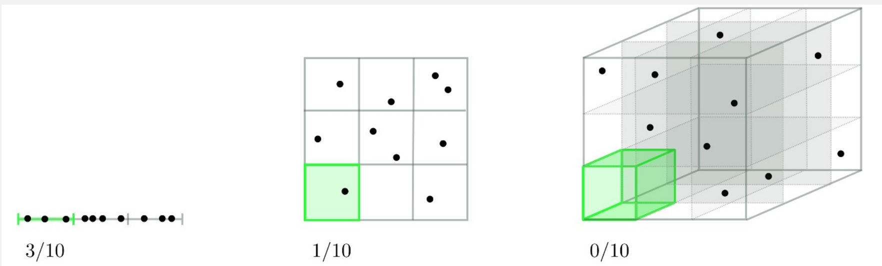
Curse of dimensionality

- While the naïve approach works for functions of few variables, it fails for functions of more than 2 or 3 variables
- Why? If sampled uniformly, the number of points at which the function needs to be evaluated grows exponentially with the number of variables



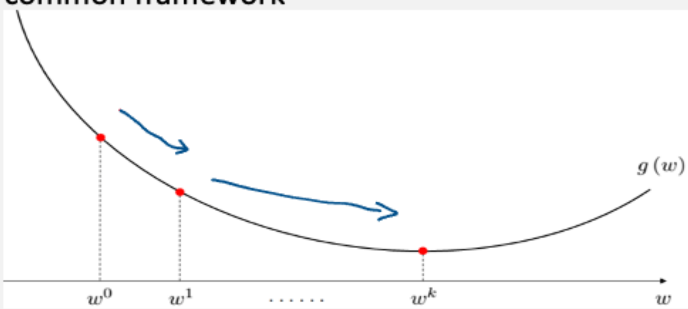
Curse of dimensionality

- The problem does not go away if we sample points randomly
- As the dimensionality n increases, smaller the fraction of samples in a n-dimensional volume

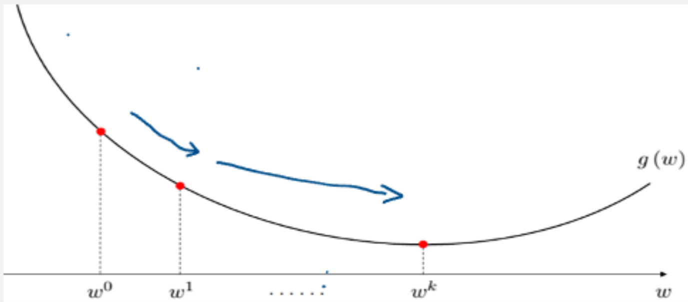


Local optimization methods

- *Local optimization methods* work by evaluating the function at a single point and sequentially updating it until an approximate minimum is reached
- Local optimization methods are by far the most popular optimization methods in machine learning
- While details vary, all local optimization methods fit into a common framework



Local optimization methods



- Starting with an initial point $\mathbf{w}^0$, local optimization methods iteratively update the current point such that:

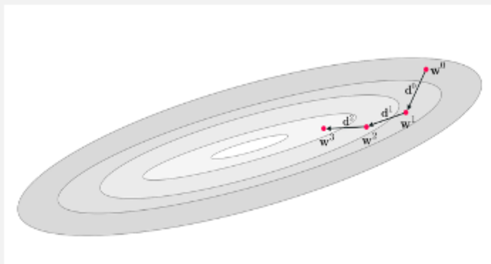
  $g\left(\mathbf{w}^0\right) > g\left(\mathbf{w}^1\right) > \cdots > g\left(\mathbf{w}^K\right)$

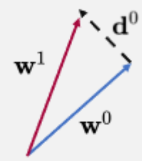
- Unlike global optimization methods, local optimization scales gracefully with the number of dimensions

Local optimization methods – a general framework

- To update the initial point $\mathbf{w}^0$ to $\mathbf{w}^1$, such that. $g(\mathbf{w}^0) > g(\mathbf{w}^1)$, a *descent direction* $\mathbf{d}^0$ is found
- Once such a descent direction is found the point is updated

$$\mathbf{w}^1 = \mathbf{w}^0 + \mathbf{d}^0$$





In general, $\mathbf{w}^k = \mathbf{w}^{k-1} + \mathbf{d}^{k-1}$ where $g(\mathbf{w}^0) > g(\mathbf{w}^1) > g(\mathbf{w}^2) > \cdots > g(\mathbf{w}^K)$

Local optimization methods

- How do we find $\mathbf{d}^{k\text{-}1}$?
- A multitude of methods exist – they differ from each other in how the descent direction is found

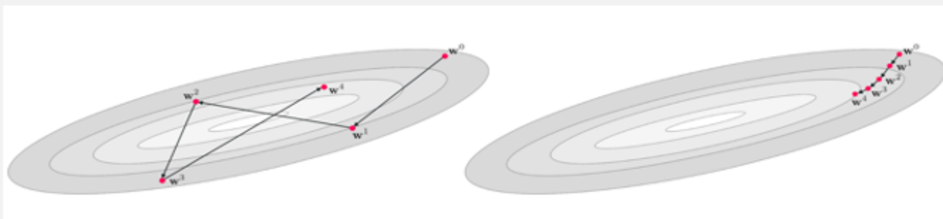
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \mathbf{d}^{k-1}$$

$$\left\|\mathbf{w}^k - \mathbf{w}^{k-1}\right\|_2 = \left\| \left(\mathbf{w}^{k-1} + \mathbf{d}^{k-1}\right) - \mathbf{w}^{k-1}\right\|_2 = \left\|\mathbf{d}^{k-1}\right\|_2.$$

- $\|A\|_2$ is the 2nd norm of the vector $A = \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix}$
- $\|A\|_2 = \sqrt[2]{\sum_{i=1}^{N} a_i^2}$
- Hence, the distance moved as a result of update is equal to the length of the vector defining the descent direction

Local optimization methods

- Depending on how our descent direction vectors are obtained, we may or may not have control over their length
  - All we ask is that they point in the right direction, 'down hill'
- Even if they point in the right direction - towards points the function value is lower - that their *length* could be problematic
  - If the length is too large, we may overshoot the minimum
  - If the length is too small, we may take forever to reach the minimum



Local optimization methods

- Most local optimization methods use a *step size parameter* (called a *learning rate* parameter in ML)

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{d}^{k-1}$$

  - In the simplest case, the step size is fixed
  - In the most general case, it can vary from step to step
- Now,

$$\left\|\mathbf{w}^k - \mathbf{w}^{k-1}\right\|_2 = \left\| \left(\mathbf{w}^{k-1} + \alpha \mathbf{d}^{k-1}\right) - \mathbf{w}^{k-1}\right\|_2 = \alpha \left\|\mathbf{d}^{k-1}\right\|_2.$$

- We can adjust the step length by choosing α

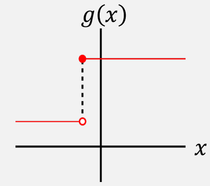
Calculus review

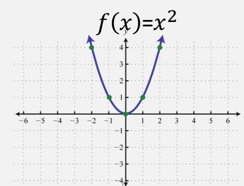
- A function of a real variable $f(x)$ is differentiable at a point a if $\lim_{\epsilon \to 0} \frac{f(a+\epsilon)-f(a)}{\epsilon}$ exists
- The limit is called the derivative of $f(x)$ at $x = a$
- The the derivative of $f(x)$ is denoted by $\frac{df}{dx}$
- If f is differentiable at a, then f must be continuous at a
  - $g(x)$ is not continuous, not differentiable
  - $f(x)$ is continuous and differentiable
  - $h(x)$ is continuous but not differentiable at $x = 0$

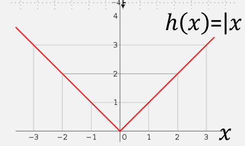
$$\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{d\left(\frac{u}{v}\right)}{dx} = \frac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$$







Calculus review

$$f(x) = x^2 + 3x$$

$$\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{df}{dx} =$$

Examples

$$f(x) = x(x+3)$$

$$\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{df}{dx} =$$

Examples

$$f(x) = \frac{x(x+3)}{x^2}$$

$$\frac{d\left(\frac{u}{v}\right)}{dx} = \frac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$$

$$\frac{df}{dx} =$$

Partial derivatives and chain rule

Let $f(\mathbf{X}) = f(x_{0,}x_1, x_2, \ldots\ldots x_n)$

$\frac{\partial f}{\partial x_i}$ is obtained by treating all $x_i \mid i \neq j$ as constant.

Chain rule

Let $z = \varphi(u_1 \ldots . u_m)$

Let $u_i = f_i(x_{0,}x_1 \ldots\ldots x_n)$

Then $\forall k \ \frac{\partial z}{\partial x_k} = \sum_{i=1}^{m} \left(\frac{\partial z}{\partial u_i}\right)\left(\frac{\partial u_i}{\partial x_k}\right)$

- $z = f(u, v) = u^2 + 2v$
- $u = f_1(x, y) = 2x$+y
- $v = f_2(x, y) = x^2 + y$
- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial z}{\partial v}\frac{\partial v}{\partial x}$

$\frac{dz}{dx} = 2u(2) + 2(2x)$

$\frac{dz}{dx} = 4(2x + y) + 4x$

$\frac{dz}{dx} = 4(3x + y)$

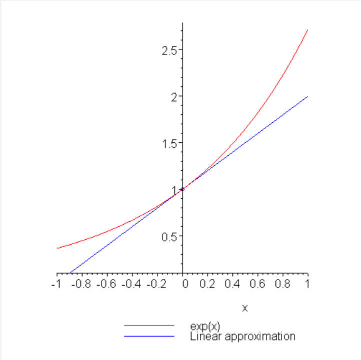
Taylor series approximation

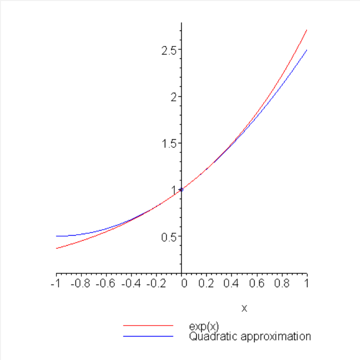
Suppose $f(x)$ is differentiable (i.e., its derivatives $\frac{df}{dx}, \frac{d^2f}{dx^2}, \cdots \frac{d^nf}{dx^n}$ exist) and $f(x)$ is continuous in the neighborhood of x_0. Then the *Taylor Series* expansion of a function $f(x)$ around $x = x_0$ is given by:

$$f(x) = f(x)\mid_{x=x_0} + \frac{df}{dx}\mid_{x=x_0}(x-x_0) + \frac{1}{2}\frac{d^2f}{dx^2}\mid_{x=x_0}(x-x_0)^2 + \ldots + \frac{1}{n!}\frac{d^nf}{dx^n}(x-x_0)$$

- Suppose $f(x) = x^2 + 1$
- $\frac{df}{dx} = 2x$
- Suppose we want to approximate f(x) at $x = 1.01$ given $f(1) = 2$
- $f(1.01) \approx f(1) + 2(1.01 - 1) \approx 2.02$
- $f(x + \Delta x) \approx f(x) + \left.\frac{df}{dx}\right|_x \Delta x$

Example


2.5
2
1.5
1
0.5
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1
x
exp(x)
Linear approximation


2.5
2
1.5
1
0.5
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1
x
exp(x)
Quadratic approximation

Taylor series approximation of multi-variable functions

The concepts introduced above extend quite naturally to the case of multivariate functions (i.e., functions of several variables). Consider a multivariate function $f(\mathbf{X}) = f(x_0, \ldots, x_n))$. Now we have *partial derivatives* that represent the rate of change of $f(\mathbf{X})$ with respect to each variable x_i. A partial derivative with respect to x_i is computed by taking the derivative of $f(x_0, \ldots x_n)$ by treating $\forall j \neq i$, x_j as though it were a constant.

Taylor Series can be used to approximate a function of several variables in a neighborhood where the function is continuous and differentiable. For example, the Taylor Series expansion for the function $\phi(x_1, x_2)$ around $\mathbf{X_0} = (x_{01}, x_{02})$ is given by:

$$\phi(\mathbf{X_0}) + \frac{\partial \phi}{\partial x_1} |_{\mathbf{X}=\mathbf{X}_0} (x_1 - x_{01}) + \frac{\partial \phi}{\partial x_2} |_{\mathbf{X}=\mathbf{X}_0} (x_2 - x_{02}) + \frac{1}{2}\frac{\partial^2 \phi}{\partial x_1^2} |_{\mathbf{X}=\mathbf{X}_0} (x_1 - x_{01})^2 + \frac{1}{2}\frac{\partial^2 \phi}{\partial x_2^2} |_{\mathbf{X}=\mathbf{X}_0} (x_2 - x_{02})^2 + \ldots$$

Taylor series approximation of multi-variable functions

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}$$

$$f(\mathbf{x_0} + \mathbf{a}) \approx f(\mathbf{x_0}) + \nabla^T \mathbf{x}\big|_{\mathbf{x}=\mathbf{x_0}} \mathbf{a}$$

$$\nabla \mathbf{x} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_N} \end{bmatrix}$$

Taylor Series Approximation of Multivariate Functions

Let $f(\mathbf{X}) = f(x_{0,} x_1, x_2, \ldots\ldots x_n)$ be differentiable and continuous at $\mathbf{X}_0 = (x_{00,} x_{10}, x_{20}, \ldots\ldots x_{n0})$

Then

$$f(\mathbf{X}) \approx f(\mathbf{X}_0) + \sum_{i=0}^{n} \left.\left(\frac{\partial f}{\partial x}\right)\right|_{\mathbf{X}=\mathbf{X}_0} (x_i - x_{i0})$$

Minimizing functions using Gradient descent

- Suppose we want to minimize $f(z)$ with respect to z
- Suppose we start at $z = z_0$
- Suppose we want to move to $z = z_1$ such that $f(z_1) < f(z_0)$
- Change in z, $\Delta z = z_1 - z_0$
- Change in f, $\Delta f = f(z_1) - f(z_0)$
- Gradient of f at z_0 = $\left.\frac{df}{dz}\right|_{z_0} \approx \frac{\Delta f}{\Delta z}$

$$\Delta f = f(z_1) - f(z_0) = \frac{\partial f}{\partial z}|_{z=z_0} \Delta z$$

- We want $\Delta f < 0$ (f decreases as we move from $z_0$to z_1)
- We should choose. $\Delta z = -\eta \frac{\partial f}{\partial z}|_{z=z_0}$ where $\eta > 0$
- $\Delta f = -\eta \left(\left.\frac{df}{dz}\right|_{z_0}\right)^2$ is never positive (as desired)
- Hence, we must update z in the direction of the negative gradient of f

Minimizing / Maximizing Multivariate Functions

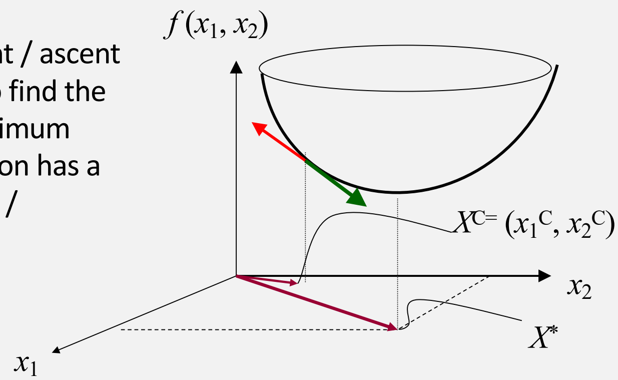
To find $\mathbf{X}^*$ that minimizes $f(\mathbf{X})$, we change current guess $\mathbf{X}^C$ in the direction of the negative gradient of $f(\mathbf{X})$ evaluated at $\mathbf{X}^C$

$$\mathbf{X}^C \leftarrow \mathbf{X}^C - \eta \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \ldots\ldots\ldots\ldots \frac{\partial f}{\partial x_n} \right) \Bigg|_{\mathbf{X}=\mathbf{X}^C} \quad \text{(why?)}$$

for small (ideally infinitesimally small)

Minimizing / Maximizing Functions

Gradient descent / ascent is guaranteed to find the minimum / maximum when the function has a single minimum / maximum



The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$
2: `for` $k = 1...K$
3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$
4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^K$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^K$

- When does gradient descent stop?
- Technically (when α is chosen well) the algorithm will *halt near stationary points of a function, typically minima or saddle points*.
- How do we know this? By the very form of the gradient descent step itself.
- Say the step

$$\mathbf{w}^{k} = \mathbf{w}^{k-1} - \alpha\nabla g\left(\mathbf{w}^{k-1}\right)$$

does not move from the prior point $\mathbf{w}^{k-1}$ significantly.

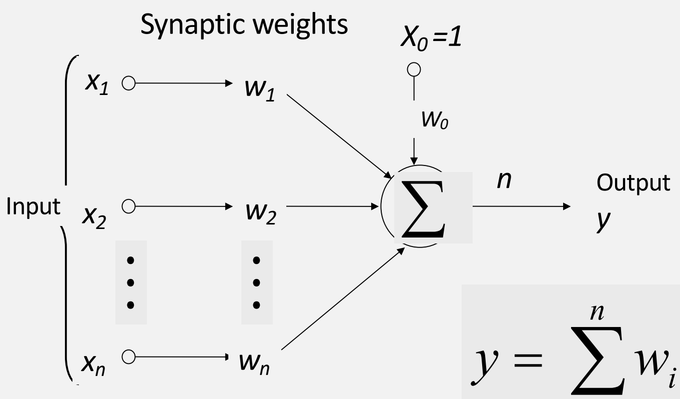
- Then this can mean only one thing: *that the direction we are traveling in is vanishing* i.e.,$-\nabla g\left(\mathbf{w}^{k}\right) \approx \mathbf{0}_{N\times 1}$
- This is - by definition - a minimum, or saddle point) of the function.

Simple Linear Regression

- In the simplest case, we have one (input), independent variable *x*, and one (output) dependent variable *y*
  - Multiple linear regression assumes an input vector **x**
  - Multivariate linear regression assumes an output vector **y**
- We will "fit" the points with a linear hyper-plane (line in the simplest case)
- Which line should we use?
  - Choose an objective function
  - For simple linear regression we choose sum squared error (SSE)
    - $\Sigma\ (d_i - y_i)^2 = \Sigma\ (e_i)^2$
  - Thus, find the line which minimizes the sum of the squared residues (e.g. least squares)

Hyperplane for multiple linear regression

Linear regression



$$y = \sum_{i=0}^{n} w_i x_i$$

Learning Task

$\mathbf{W} = [W_0 ....W_n]^T$ is the weight vector

$\mathbf{X}_p = \left[X_{0p} ....X_{np}\right]^T$ is the pth training sample

$y_p = \sum_i W_i X_{ip} = \mathbf{W} \bullet \mathbf{X}_p$ is the output of the neuron for input $\mathbf{X}_p$

$d_p = f\left(\mathbf{X}_p\right)$ is the desired output for input $\mathbf{X}_p$

$e_p = \left(d_p - y_p\right)$ is the *error* of the neuron on input $\mathbf{X}_p$

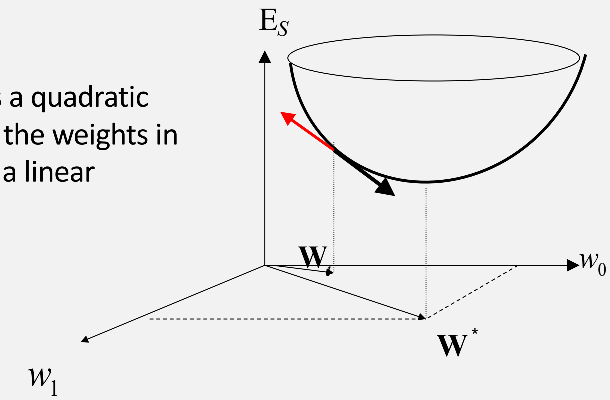
$S = \left\{\left(\mathbf{X}_p, d_p\right)\right\}$ is`a (multi) set of training examples

$E_S(\mathbf{W}) = E_S(W_0, W_1, ...W_n) = \frac{1}{2}\sum_p e_p^2$ is the estimated error of $\mathbf{W}$ on training set S

Goal: Find $\mathbf{W}^* = \underset{\mathbf{W}}{\arg\min}\, E_S(\mathbf{W})$

Learning linear functions

The error is a quadratic function of the weights in the case of a linear function



Learning linear functions

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

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\left\{\sum_p e_p^2\right\} = \frac{1}{2}\left(\sum_p \frac{\partial}{\partial w_i}\left(e_p^2\right)\right)$$

$$= \frac{1}{2}\left(\sum_p \left(2e_p\right)\frac{\partial e_p}{\partial w_i}\right) = \sum_p e_p \left(\frac{\partial e_p}{\partial y_p}\right)\left(\frac{\partial y_p}{\partial w_i}\right) = \sum_p e_p (-1)\left(\frac{\partial}{\partial w_i}\left(\sum_{j=0}^{n} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip} + \sum_{j \neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip}\right) + \frac{\partial}{\partial w_i}\left(\sum_{j \neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right) x_{ip}$$

$$w_i \leftarrow w_i + \eta \sum_p \left(d_p - y_p\right) x_{ip}$$

Batch Update

$$w_i \leftarrow w_i + \eta \sum_p \left(d_p - y_p\right) x_{ip}$$

Per sample Update

$$w_i \leftarrow w_i + \eta \left(d_p - y_p\right) x_{ip}$$

General Algorithm

- Algorithm (Model algorithm for n-dimensional unconstrained minimization). Let x_k be the current estimate of x^*.
  - [Test for convergence] If the conditions for convergence are satisfied, the algorithm terminates with x_k as the solution.
  - [Compute a search direction] Compute a non-zero n-vector p_k, the direction of the search.
- Different algorithms differ primarily in their choice of the search direction

Momentum update

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = -\eta \left. \frac{\partial E}{\partial w_i} \right|_{w_i = w_i(t)} + \alpha \Delta w_i(t-1) \text{ where } 0 < \alpha < 1$$

$$= -\eta \sum_{\tau=0}^{t} \alpha^{t-\tau} \left. \frac{\partial E}{\partial w_i} \right|_{w_i = w_i(\tau)}$$

The momentum update allows effective learning rate to increase when feasible and decrease when necessary. Converges for $0 \leq \alpha < 1$

Locally weighted regression

Locally weighted regression involves calculating an approximation of the function value for a given input based on its nearest neighbors when needed during the approximation phase as opposed to during the learning phase.

Let the approximation be of the form

$$g(X) = w_0 + \sum_{i=1}^{N} w_i x_i$$

in a small neighborhood around a query X_q

Locally weighted regression

$$g(X) = w_0 + \sum_{i=1}^{N} w_i x_i$$

Minimize the error over the K nearest neighbors of X_q

$$E(X_q) = \frac{1}{2} \sum_{X \in KNN(X_q)} \left(f(X) - g(X)\right)^2$$

$$w_i \leftarrow w_i - \eta \frac{\partial E(X_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{X \in KNN(X_q)} \left(f(X) - g(X)\right) x_i$$

Locally weighted regression

Minimize the error over all the neighbors of X_q in the training set weighted by an inverse function of distance to the neighbors

$$E_2(X_q) = \frac{1}{2} \sum_{X:(X, f(X)) \in D} \left(f(X) - g(X)\right)^2 \phi\left(d(X_q, X)\right)$$

$$w_i \leftarrow w_i - \eta \frac{\partial E_2(X_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{X:(X, f(X)) \in D} \phi\left(d(X_q, X)\right)\left(f(X) - g(X)\right) x_i$$

Locally weighted regression

Minimize the error over all the neighbors of X_q in the training set weighted by an inverse function of distance over the *K* nearest neighbors

$$E_2(X_q) = \frac{1}{2} \sum_{X:(X,f(X))\in D} \left(f(X) - g(X)\right)^2 \phi\left(d(X_q, X)\right)$$

$$w_i \leftarrow w_i - \eta \frac{\partial E_2(X_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{X:(X,f(X))\in D} \phi\left(d(X_q, X)\right)\left(f(X) - g(X)\right) x_i$$